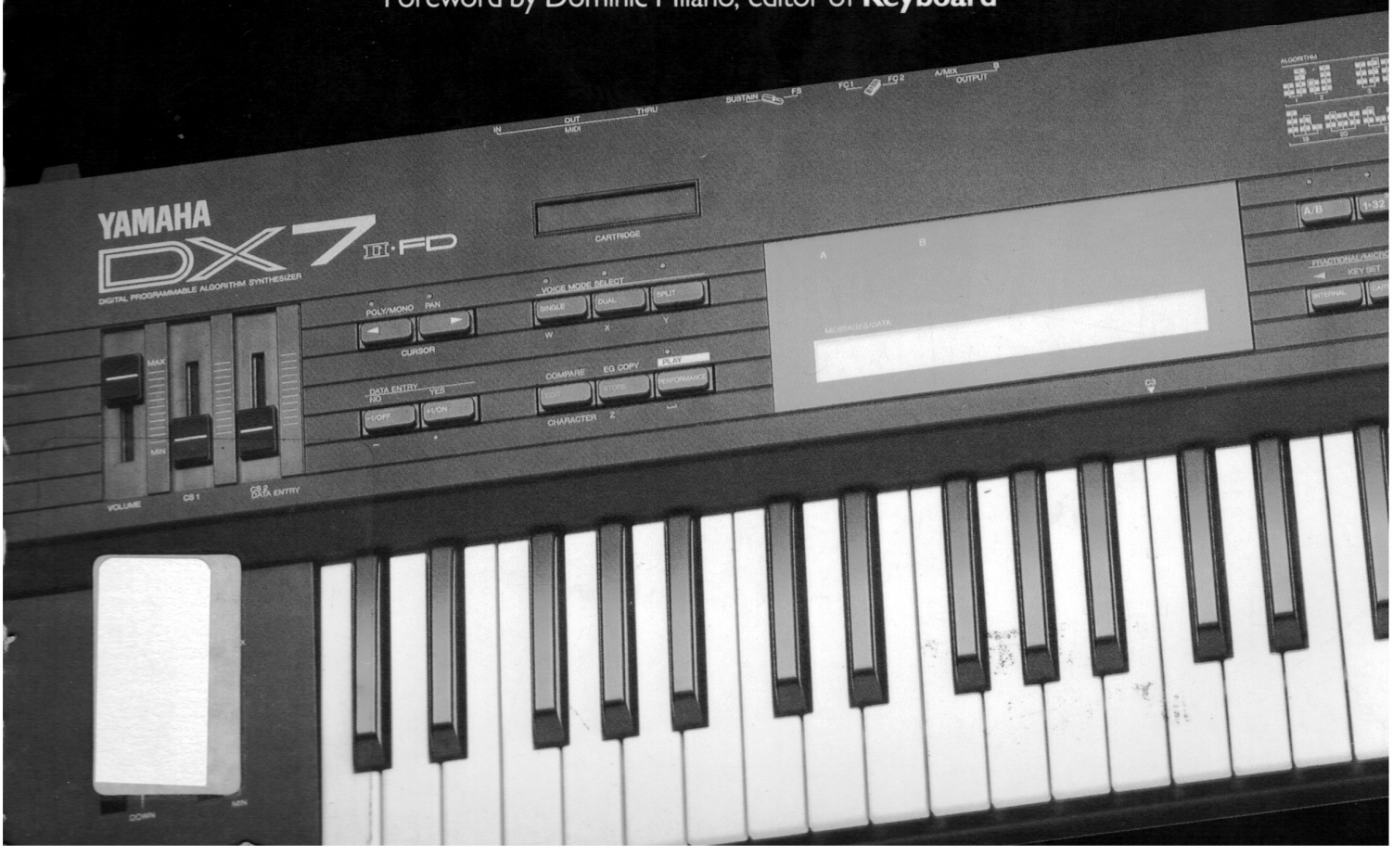
The Complete DX7II.

by Howard Massey.

Everything you need to know about the newest member of the DX family of synthesizers. The comprehensive information presented here is applicable to both the DX7IID and DX7IIFD instruments. The instructional material takes a simple, non-technical approach to this fascinating subject and explains in clear terms the process of digital FM synthesis. With the help of the three soundsheets inside, you may explore the enormous tonal capabilities available to you with this revolutionary and unique instrument. Included are 90 hands-on exercises; over 500 illustrations, diagrams, and photographs; and 6 comprehensive reference appendixes.

Foreword by Dominic Milano, editor of **Keyboard**



The Complete DX7II.

by Howard Massey.

&

Amsco Publications
New York/London/Sydney/Cologne

Acknowledgements

I would like to thank the following individuals for their invaluable help in making this book a reality: David Schoenbach, Mark Koenig, and Bill Hinely of Yamaha's Digital Musical Instruments Division; Barrie Edwards, Dan Earley, Peter Pickow, and all the good folks at Music Sales; Howard Sandroff of the University of Chicago for his input and feedback — particularly with regards to the micro tuning section; Dr. John Chowning of Stanford University; Dominic Milano of Keyboard Magazine; Daniel Shklair (for his soundsheet narration) as well as the rest of the staff of the Center for Electronic Music in New York; and last, but by no means least, to my talented wife, Linda E. Law, for her patience in putting up with my preoccupation with the DX7II throughout this process — as well as for her excellent photographic contributions to this book.

About The Author

Howard Massey has been professionally involved in the world of synthesizers for over a decade. He is a composer, record producer, and recording engineer, in addition to his activities as a session programmer, synthesizer instructor, and author.

Other books authored or co-authored by by Mr. Massey include *The Complete DX7, A Synthesist's Guide To Acoustic Instruments*, and *The Complete Guide to MIDI Software*.

He is the founder and executive director of the Center for Electronic Music, a non-profit organization based in New York City offering educational services in music technology as well as preproduction and production services for electronic music works. He has been teaching workshops and seminars on digital FM synthesis, MIDI, and programmable analog synthesis since 1983, and has appeared as a guest lecturer at the University of Chicago and the Eastman School of Music.

As a composer, he has written numerous television and film themes, ambient electronic soundtracks for holography exhibitions, and even a hit single for Herb Alpert! As a producer/engineer/programmer, he has worked with a broad spectrum of of recording artists: from Kraftwerk to Spandau Ballet; from Roy Buchanan to Sham 69. Born in New York City, he resided in London, England for some four years before returning to the U.S. in late 1982. He is presently lives in Huntington, New York.

Interior photographs by Linda E. Law
Illustrations and layout by Leonard Vogler

International Standard Book Number: 0.8256.1119.9

Exclusive Distributors:
Music Sales Corporation
24 East 22nd Street, New York, NY 10010 USA
Music Sales Limited
8/9 Frith Street, London W1V 5TZ England
Music Sales Pty. Limited
27 Clarendon Street, Artarmon, Sydney NSW 2064 Australia

Printed in the United States of America by
Vicks Lithograph and Printing Corporation

List of Hands On Exercises

Foreword

As musicians demand ever more complexity in sound quality and control, the technology must increase in complexity as well. Frequency modulation synthesis presents a universe of possibilities, which has found an elegant expression in Yamaha's family of "X" instruments, of which the DX7II is the newest and most powerful. But how to explain this ever-developing technology to the musician who must use it?

Howard Massey possesses not only the technical knowledge, but especially the sense for musical relevance. His gift for explaining at a non-trivial level will lead the musician directly to the center of some of the most exciting developments in music technology.today.

Dr. John Chowning
Director, CCRMA
Stanford University

The DX7 was the first synthesizer to go over six figures in units sold. This is in spite of the fact that it is one of the most difficult instruments to learn to program. The second best way I can think of to turn a non-technical neo-musician-cum-proto-human into a snivelling blob of jellyfish ooze would be to lock that person in a room with a DX7 and not let them out until they can learn how to program it. (I won't tell you the first best way — it's a trade secret).

Three years after the the DX's introduction, just when people were feeling comfy cozy with six-operator FM, Yamaha decided to throw a monkey wrench into the soup by bringing out the DX7 Mark II (FD, D, or S — name your poison.)

Imagine yourself alone in a locked room. Just you and the DX7 Mark II. No door is visible, so don't even think about trying to pick the lock. There's no window either. They won't let you out (notice I said "they" — I'm not going to take the blame for coming up with these stupid hypothetical situations) until you figure out how to use the instrument. You're allowed one wish. I know if it were me, I'd wish for a copy of this book. It's the second best way I can think of to avoid turning into the aforementioned proto-human jellyfish ooze. (I won't mention the first — it's a trade secret.)

Dominic Milano
Editor
Keyboard Magazine

Introduction

Why mess with a good thing? The Yamaha DX7 synthesizer, first released in mid-1983, is, as of this writing, the best-selling electronic music instrument ever made. In late 1986, Yamaha decided to cease production of the DX7 and replace it with the upwardly compatible (but considerably more complex and slightly more expensive) DX7II. Why?

Of course, no one outside the company can speak for Yamaha. But I suspect the answer has something to do with not being content to rest on one's laurels. The leading companies in this rapidly expanding field get there by constantly staying one step ahead. Even though the digital FM technology used by the original DX7 is still, to most people, new technology, some of the particular applications of it in the original instrument were limited. Human nature being what it is, three years of phenomenal success with the DX7 no doubt also meant to Yamaha three years of users asking for yet more features - and the upwardly compatible DX7II (meaning that it can work with DX7 patches) appears to represent their answer to these requests.

The unprecedented acceptance of the first DX7 as a "classic" synthesizer had a great deal to do with the fact that it produced unique types of sounds since it used a synthesis system (the aforementioned *digital FM* system) very few people had ever worked with. Since its debut, nearly a dozen more "X" instruments have been manufactured by Yamaha, making the challenge to the DX7II even greater. Has that challenge been successfully met? Only time will tell, of course, but so many new features have been added to this second generation instrument, and the sound quality has been improved to such a degree that chances are good that the DX7II will match or possibly even exceed the commercial success that its forebear enjoyed.

What are some of these features? The DX7II is, first and foremost, a multi-timbral instrument: up to two different sounds can be generated simultaneously and either layered over the entire keyboard or placed on either side of a programmable "split point". Secondly, the keyboard as well as the programming switches have been improved: gone is that questionable feel of the keyboard as well as the mushy and sometimes problematic membrane switches. The data display has also been greatly upgraded: it's now backlit (making it much easier to read), with two lines of 40 characters (as opposed to two lines of 15 characters offered by the original DX7). Information relating to the use of the various real-time controllers (referred to in the original DX7 as "function controls") - are in the DX7II linked to individual voices, and several more controllers (all of which can be changed in performance) have been added. The "keyboard level scaling" control has been greatly enhanced with the addition of "fractional level scaling" - allowing you to

independently set the output levels for individual operators every three semitones. Comprehensive voice detuning parameters provide another powerful new feature, making the sound of the DX7II generally warmer and closer to analog richness - and the fidelity of the sound has itself been greatly improved with the addition of 16-bit (as opposed to 12-bit on the old DX7) processing. Not only can twice as many voices be stored internally, but a new kind of memory, called "performance memory", is available: this allows the user to pre-program up to 32 configurations of voices with specific effects and transpositions. DX7II RAM cartridges hold more data as well, and the "FD" model of this instrument allows for positively enormous amounts of data storage to its built-in disk drive - and this data need not be internally generated data when the drive is used in its "MDR" (MIDI Data Recorder) mode. The MIDI implementation of the instrument itself has been greatly improved, making the DX7II, unlike its immediate ancestor, a usable master keyboard controller. Various "microtunings" - different keyboard scalings - can be accessed or even programmed by the user. Last but by no means least, the DX7II is a true stereo instrument, with extensive and imaginative panning controls built in.

All in all, this second generation instrument presents an impressive set of credentials. But the down side of all of these added features is that it means - for the novice and experienced synthesist alike - so much more to learn.

Enter "The Complete DX7II". Readers of my first book, "The Complete DX7", will already be familiar with the overall format I will be using here. First off, no previous knowledge or experience will be assumed. This may make for slightly boring reading at first for those of you advanced in digital FM programming, but I'll risk that so we can include those of you who may not yet be quite so advanced. Secondly, the principles outlined here will be supplemented with a series of easy-to-follow hands-on exercises (with audio cues sprinkled in from time to time so you can verify that you're doing the exercise correctly), allowing you to apply the principles of FM synthesis as you learn them. As with "The Complete DX7", this book is meant to be a tutorial, and you should work your way through this book with a DX7II and audio cassette recorder (in order to play back your tape copy of the audio cues given on the included sound sheets) handy at all times. It's also worth noting that "The Complete DX7II" is not just a rewrite of the first book. Although the same general format is used, a great deal of new information is presented here.

My intention is to provide a comprehensive yet non-technical "how-to" approach to digital FM synthesis and to the specific programming functions of the DX7II. Despite anything you may have heard to the contrary, digital FM is not the convoluted mystery that some make it out to be. The end result of every manipulation is logical and, more importantly, every change you make to a digital FM sound can be heard instantly - making the whole process of learning what you are doing so much easier.

Here, then, is the general plan of action: We'll begin with a discussion of basic audio theory, applicable not just to digital FM synthesis, but to all types of sound and sound synthesis. Even if you consider yourself reasonably knowledgeable in this area, it is my strongest recommendation that you at least skim through this chapter as it will contain definitions of jargon that we will use throughout the entire book. Chapter Two will give you a quick "guided tour" through

the front panel of the DX7II as well providing a summary of the various types of memories that this instrument uses. In Chapter Three, we will focus on the actual workings of digital FM as we examine the instrument's basic sound-producing unit, the *operator*. Chapters Four, Five, and Six will take you through the *voice initialization* procedure, where we will discuss how to make sounds from scratch. Following this will be a series of chapters that zero in on specific "tools" that are used to mold, shape, and control these sounds, and a chapter dealing specifically with the MIDI interface that allows the DX7II to communicate with the outside world. Last but by no means least will be a discussion of some advanced programming techniques, allowing you to put all of the previous information together into a coherent whole.

The reader is also advised to refer often to the series of appendices attached, which provide useful supplementary materials as well as a "quick reference guide" to the many editing parameters of the DX7II switches.

As The Complete DX7II was going to press, Yamaha announced the addition of a new DX instrument to their product line: the DX7S. This instrument is essentially a single-voice DX7II, offering all the features of the DX7II except the ability to generate two sounds simultaneously, to pan these two sounds via separate stereo outputs, and the ability to store data to disk. The front panel layout of the DX7S is virtually identical to the DX7II, with the main difference simply being a smaller LCD display. Readers should be advised that virtually all of the information in this book (with the possible exception of the "Pan" section in Chapter Fourteen) is applicable to the DX7S as well as the DX7II, though the LCD displays will sometimes be slightly different.

The DX7II provides us with an incredibly large "tool box" of controls that we can apply to our sound, but don't lose sight of the bottom line: *sound*. If there were ever three words of wisdom in the synthesists vocabulary, it would be these: "Use your ears!". So get the wax out, and let's start programming!

Howard Massey
February 1987

Chapter One ◇
Basic Audio Theory

What Is A Synthesizer?

A *synthesizer*, by definition, is any kind of machine or device that can create sounds without the benefit of any kind of physical motion. This is what differentiates electronic instruments from acoustic instruments. The one thing that is common to all acoustic instruments is that the sound they create always originates as something physically vibrating, be it a string, a reed, a skin, or some other kind of tangible thing. We should probably extend our definition somewhat and add that a true synthesizer should allow the user some degree of control over how the sound is shaped and created. If we don't make this disclaimer then we would have to allow instruments such as the home organ and toy-like devices such as the smaller Casio machines to share in the glory! These instruments do create their sounds electronically and not physically but they allow no significant controls to the user.

Within this broad definition there are two different types of synthesizers: analog and digital. These two breeds are as different as the proverbial apples and oranges in the way they operate. Basically, analog synthesizers create their sounds by manipulating electrical signals whereas digital synthesizers are essentially computers which evolve sounds by manipulating numbers.

Numbers?? How can we hear numbers? Allow me, if you will, to answer your rhetorical question with another question: How can we hear voltages? Before we jump right in with a discussion of digital synthesis (which, after all, is the type of synthesis the DX7II uses) it will be helpful to gain a rudimentary understanding of how analog systems work.

What Is Analog?

The word *analog* itself simply means "like" or "similar to". The best way to understand how analog systems work is to examine how a very common home analog system, like the standard record player, operates.

If you are not a Tibetan monk sitting on top of a mountain, you probably own or at least have at some time used a common garden variety stereo system with record player. Odds are you slap a record on weekly, daily, or maybe even hourly (!) without ever thinking about what is physically occurring, and that's fine, because it creates jobs for people like me!

Let's examine the process: First of all, as you know, the needle is bouncing around in the groove of the record, so we start with actual physical motion. The purpose of the cartridge in your tone arm is to "listen" to that physical movement and convert it into an electrical signal. But it's not just any kind of arbitrary electrical signal; it is an *analog* electrical signal, because it will perfectly emulate the physical motion of the needle bouncing around. In other words, whenever the

needle bounces up and down, say a hundred times in a particular second, the electrical signal will wobble, or "bounce", you guessed it, a hundred times per second! This is the basic process of creating an analog: You essentially are converting energy from one form to another, but you preserve the movements of that energy through the conversion.

Okay, let's continue with our tracing of the journey as Bruce Springsteen's voice travels from the grooves of his latest platinum offering to your eager ears. The cartridge sends its electrical signal down the wires of your tone arm and out the record player into your amplifier (or receiver, if it happens to have a radio attached), where the sound is simply amplified, or in plain English, made louder. If the amplifier is a good one, that's about all it will do, and if it's not such a good one it may add various kinds of distortions to the sound, but we won't concern ourselves with that. This amplified signal then travels out of your amplifier/receiver down the speaker wires to your speakers (or headphones, which really are no different from small speakers).

A speaker is just a box with two magnets inside it. One of the magnets is actually an electromagnet and it is mounted firmly on the back wall of your speaker box and the other magnet is sort of suspended in mid-air nearby, attached to a paper cone. The electromagnet is going to be receiving the electrical signal from the speaker wire and will alternately have its magnetic polarity changed as the electrical signal undergoes its changes in voltage (and remember, this electrical signal is an analog of the original needle movements). As this fixed electromagnet goes through its paces it will alternately attract and repel the floating speaker. Because the floating speaker is attached to a paper cone, these small movements back and forth will translate themselves into larger movements of the cone, which will cause air to be pushed back and forth.

This movement of air eventually reaches our ear, and our eardrum is designed to sympathetically vibrate accordingly. These internal vibrations are transmitted to our inner ear where a series of physiological events transform these movements back into an electrical signal. This electrical signal travels through your nerve fibers and up into your brain where you finally perceive all of these various convolutions as a sound: At last, Bruce!!

Let's review what's happened: Our signal has undergone four separate transformations. We started as physical motion (the needle bouncing around), converted to analog electrical (inside the cartridge), converted to analog magnetic (inside the speaker), back to physical (the movements of the speaker cone), and once again back to electrical (in our inner ear). We can trace back still further if we like, to the recording process itself. The sound actually began with Bruce himself singing into a microphone in a recording studio somewhere in Hackensack or thereabouts. So the sound obviously began life as physical motion, air being moved by Bruce's vocal chords. The purpose of the microphone was similar to that of the record player cartridge; to convert physical movement into an analog electrical movement. This electrical signal (containing of course the same type of movements as the original physical source) travels down wires into a mixing console where it is processed in a number of ways and then sent to a tape recorder. The record head of the tape recorder converts this electrical signal yet again to a magnetic signal which can be encoded on tape. The tape is played back and the playback head converts it back again to electrical! (Stick with it, we're nearly done...) In the cutting room this electrical signal from the tape playback is fed to a cutting lathe which then reconverts it back to the physical motion of a needle cutting a

groove in a lacquer plate. This lacquer plate will eventually form the template for the record you finally purchase and play.

It's fairly complicated, but the point is that the original sound in every case undergoes many conversions before you finally get to hear and enjoy it.

The important thing to understand is that all of these conversions are analog ones, meaning that the original qualities of the sound are in all cases preserved as nearly as possible. When you analyze things this way, it's really pretty amazing that your records actually sound reasonably close to the original performance!!

So where has all this gotten us, in terms of synthesizers? Well, analog synthesizers work in a very similar fashion to a record player except that they start at step two, not one. Instead of beginning life as a physical motion, analog synthesizer sounds actually begin in the first place as an electrical signal. This signal will still have to travel down some wires into and through an amplifier and into some speakers (or through a possibly inbuilt preamplifier and some headphones) in order for us to hear it. The difference then between the record player and the synthesizer is solely that in the latter instance, the sound actually originates as an electrical signal of some kind.

What Is Digital?

Back, finally, to our original question; how can we hear numbers?? (Some of you might rightfully be annoyed that I'm presuming you asked this question where in fact I have no way of knowing if you did or not. Sorry, but you'll just have to allow me some poetic license) When somebody uses the word "digital", a little light bulb should switch on over your head and you should immediately think "computer" because these words are virtually synonymous. A digital synthesizer, as mentioned earlier, is a computer that can create sounds by manipulating numbers, typically at very high speeds. Although in a perverted way we could define human beings as the ultimate computers, we still do not have the ability to hear numbers. But we have seen that by using various conversion processes we can hear voltages. Therefore, digital synthesizers need to be able to convert the numbers they generate into voltages so that we can perceive them as sounds. This is accomplished by using a clever little microchip called a digital-to-analog converter, or *DAC* for short. All digital synthesizers contain a DAC; otherwise we couldn't hear the sounds they create. How does this DAC convert numbers into voltages? Well, let's presume, for example, that our computer is generating the following numbers:

0, 2, 4, 6, 8, 6, 4, 2, 0, -2, -4, -6, -8, -6, -4, -2

and let's say that it keeps generating these same numbers over and over again. This stream of numbers is fed in *real time*, that is, as it happens, to the DAC, which then outputs an electrical signal that undergoes the same movements as the numbers, that is, an *analog* electrical signal. What we would derive from this particular stream of numbers would be a smooth, steadily changing voltage which alternated back and forth at whatever rate of speed the computer was sending the numbers. If the computer all of a sudden decided to output a bunch of very large numbers very quickly, we would get a spike in the voltage. In other words, the DAC is "looking" at the numbers fed to it and "graphing" them out electrically. **(see figure 1-1)**

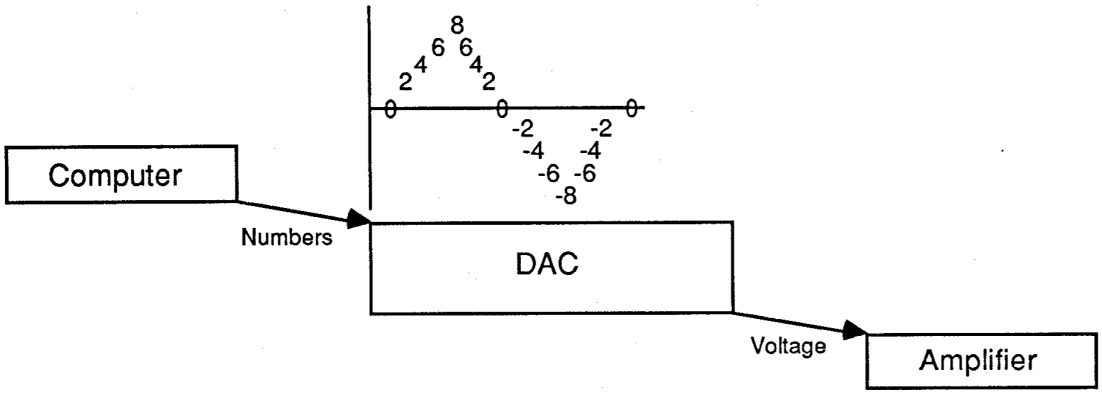


Figure 1-1

Digital synthesizers may generate their sounds with any one of several different methods. The most common of these include subtractive synthesis, additive synthesis, phase distortion synthesis, and frequency modulation (called *FM* for short) synthesis. Until Yamaha introduced the original DX7 in 1983, digital FM synthesis was virtually unknown (except to users of the Synclavier, which incorporates some FM within its primary additive system). Each of these four methods are substantially different and we will present brief summaries of their operations and differences at the end of this chapter. However, now that you've been sufficiently dazzled by the concepts presented above, it's really time to take a step backwards and talk about sound in general, not specifically as applied to synthesizers.

What Is A Sound?

We can define a sound as anything that our brain perceives as a direct result of air moving our eardrum. Sounds obviously can come in an enormous variety, from the sweet tones of a violin to a jackhammer, from the hum of a motor to a crack of lightning, from a swallow to a cough. We can categorize sounds as being musical or nonmusical, loud or soft, gentle or harsh. In all instances, however, we can only perceive sounds if air has somehow been moved (by the originating object itself or by a loudspeaker emulating the movement of the originating object). Furthermore, this movement of air must occur in a back-and-forth manner, or in "waves".

No matter how many different sounds there can be (and the number is certainly infinite), there are surprisingly only three ways of describing a sound and for the purpose of this book we will refer to these as the *three parameters of sound:*

1) Loudness, 2) Pitch (or lack of pitch), and 3) Tonal quality

All sounds, regardless of their origin, can be described with these parameters and these parameters only. Every sound has a particular loudness, or volume (the two terms are not technically synonymous, but for the purposes of this book we will assume that they are). Every sound has a particular pitch, and every sound has a particular kind of tonal quality. What's more, *every* sound *always* has all three of these qualities. There is no sound that has a particular volume and pitch, but no particular tonal quality. There is no sound that has a particular tonal quality and pitch but no particular volume. Furthermore, (and this is an extremely important point), as a rule, all three of these parameters

typically *change* throughout the duration of the sound. *Understanding these principles is really the key to understanding audio synthesis of any type.*

We have stated that the synthesizer by definition will normally allow us a certain degree of control over shaping the sound we create. That means that we need to be able to control each of these three parameters if we are to be in the driver's seat. All synthesizers worthy of the title, be they analog or digital, will allow you these controls to some extent. The DX7II will allow us particularly fine control, but the point is that *everything* we will be doing from here on in will relate back to these same three parameters.

Jargon

Every field of specialization has a jargon associated with it, and the area of audio synthesis is certainly no exception. Jargon sometimes serves merely as an ego device, in order to keep "outsiders" outside, but sometimes jargon exists to make definitions more precise. Synthesizer jargon is largely a mix of these two, but the purpose of this book is to make all of you outsiders "insiders" and so wherever possible we will identify jargon as such and give you "plain English" translations. Besides, once you know the jargon, you'll not only be able to impress your friends but you'll be able to make them feel like "outsiders" too! (and the circle goes 'round and 'round...)

Synthesists generally refer to "volume" as *amplitude*; they generally call "pitch" by the term *frequency*; and "tonal quality" is usually called *timbre*. The first two are technical terms and the third is a musical term, but all three are universally used.

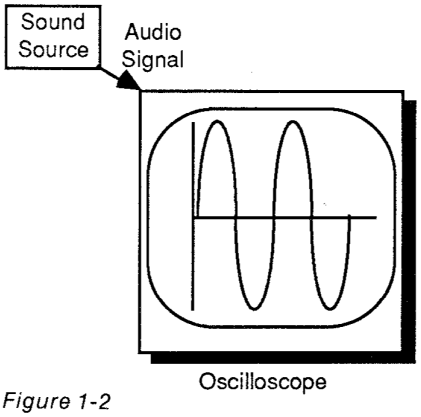


Figure 1-2

The Oscilloscope

For the purposes of this chapter nothing could be more useful to have handy than a device called an *oscilloscope*. This is basically a television-type receiver into which we can feed an audio signal; in short, it enables us to literally look at a sound. Unfortunately, oscilloscopes are rather too large and expensive to be packaged inside this book and so we will have to make do with the following diagrams and a good sense of imagination. **(see figure 1-2)**

The oscilloscope plots the sound fed into it on a graph superimposed on a TV screen. It measures the amplitude, or volume, of the sound over time. Specifically, it shows us the way a sound *changes* volume over time, and typically over very short periods of time (usually a second, a tenth of a second, or even a hundreth of a second).

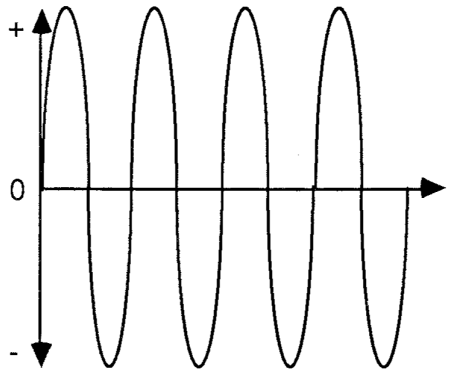


Figure 1-3

Amplitude

The height of the wave is a representation of its *amplitude*. A car horn at a distance of two inches (very loud!!) might look like **figure 1-3.**

The same car horn (i.e., same pitch and tonal quality) at a distance of two hundred feet (much quieter) might look like **figure 1-4.**

Note that nothing about the wave, except its height, changes. The amplitude or volume of a sound is normally measured in *decibels*, or *db* for short. A single decibel is roughly the smallest unit of volume change that the human ear can detect, and sounds louder than about 180 decibels can cause physical pain and/or damage (keep that in mind next time you are tempted to stick your head inside a PA bin at a rock concert!!)

It should be obvious that *all* sounds *always* change amplitude during their duration; after all, no one has yet invented a perpetual sound that lasts forever. Furthermore, *every* sound *always* eventually changes to an amplitude of 0 db (in plain English, every sound eventually ends!).

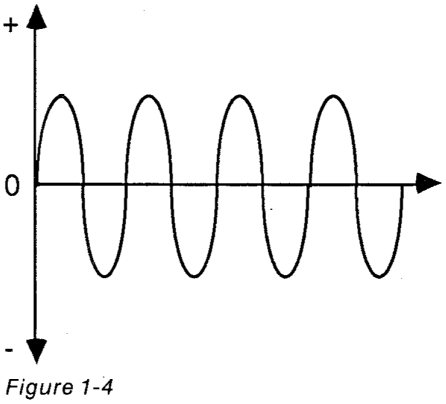


Figure 1-4

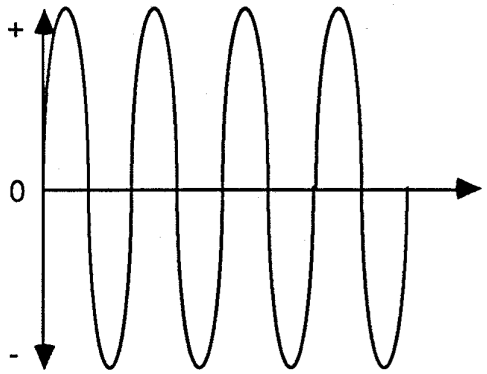


Figure 1-5

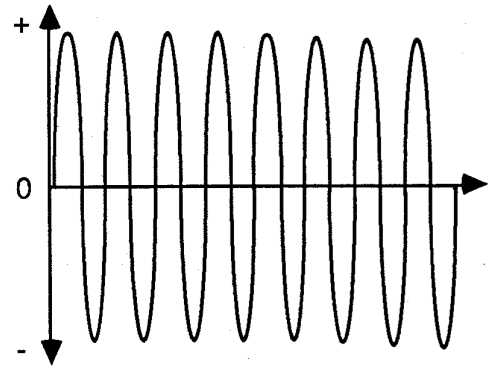


Figure 1-6

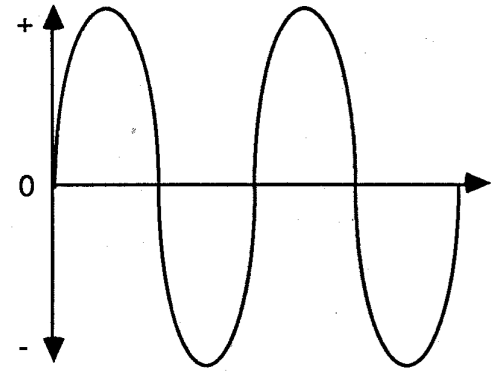


Figure 1-7

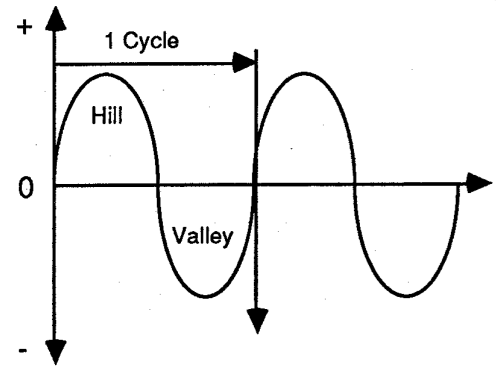


Figure 1-8

Frequency

The substitution of the word *frequency* for "pitch" is actually a very logical one, as the pitch of the sound is represented on the oscilloscope by how frequently the wave reoccurs. In other words, the more waves we have within a given time period, the higher the pitch, or frequency, of the sound. Each time a musical sound goes up an *octave*, its frequency doubles; that is, we have twice as many waves occurring in the same time period. The sound of a piano playing Middle C might look on an oscilloscope like **figure 1-5** whereas the same piano playing C above Middle C might look like **figure 1-6.**

Of course, in these particular examples we are presuming that the piano plays these notes at the same volume both times. Each time a musical sound goes down an octave, it's frequency is halved, so the same piano now playing C below Middle C would look like **figure 1-7.**

Notice that regardless of the height or shape of the wave, we always observe "hills" and "valleys", that is, an up portion followed by a down portion which is the close inverse of the up portion. If you take the up portion, or "hill", and flip it upside down, it will often look just like the "valley". This makes perfect sense in light of the fact, as we stated earlier, that sound always travels in repeating waves. We call one "hill" plus one "valley" a *cycle*, since they are always inseparable. If we then measure how many cycles occur in every second of the sound's duration, we can then make an accurate accounting as to the sound's frequency.**(see figure 1-8)**

The unit of measurement used to describe frequency is the *Hertz*, or *Hz* for short. A sound that generates, for example, exactly one cycle of waves per second is said to have a frequency of one Hertz. Therefore, if we speak, of a sound having a frequency of, say, 100 Hz, what we are saying in plain English is that this sound generates one hundred complete wave cycles per second. A shorthand term for describing 1000 Hertz is the *KiloHertz*, or *kHz* for short.

Human beings have a particularly wide range of perceiving different frequencies. We can typically hear sounds having frequencies as low as 20 Hz, and as high as 20,000 Hz (20 kHz). That is not to say that sounds don't exist above and below these frequencies - they do, but humans cannot hear most of them. Sounds within the range of human hearing are said to be in the *audible* range, and this is generally taken to be in the 20 to 20,000 Hz area. Sounds below the audible range are called *subsonic*, and these are frequencies which we can sometimes feel, but generally not hear. When you listen to most kinds of music at high volumes (such as at a live rock concert), you will often experience the phenomenon of feeling your chest pounding. This is caused by subsonic range vibrations which actually are making your body vibrate. Sounds also exist above the audible range, and these are termed *supersonic*. Many animals are able to hear subsonic and/or supersonic frequencies. This is the way dog whistles work, for example. When you blow into the dog whistle you hear nothing (unless your name is Fido), but your dog certainly does. (Maybe it even sounds like Bruce to him!)

In any event, most synthesizers will allow you to work with not only audible frequencies but with subsonic and sometimes supersonic ones as well. The need for this may not seem obvious now but will become more clear as we delve further into the operation of the DX7II. Bear in mind, again, that the frequency of most sounds generally changes somewhat (often only slightly) throughout the duration of the sound.

Timbre

This is the tricky one of the three to understand. Amplitude and frequency, after all, are pretty straightforward: it's very easy to conceptualize these terms from everyday experiences. But when it comes to tonal quality, or timbre, well, it all seems so subjective...

Fortunately, everything in audio theory is logical and highly mathematical (this is, after all, why computers are so good at synthesizing sounds). We have seen that both amplitude and frequency can easily be measured *quantitatively* (that is, in terms of numbers), and, believe it or not, so too can timbre.

Let's explore this with a simple example. Suppose you walk into a room and you have somebody play, say, "A" above Middle C on a piano that just happens to be sitting in a corner. This particular piano has just been tuned so you can be certain that the note you are hearing has a frequency of 440 Hz ("A440" is used as a reference tuning for most instruments and it simply means that the A above Middle C string is tensioned so that it vibrates basically 440 times per second when struck).

Okay, now you're hearing this note and suddenly (like a Hitchcock movie!) the lights go out. Before you can do anything about it, somebody else walks in, this time with a flute, and begins playing the same note. Right on the heels of the flutist is someone else with a guitar, and then another musician with a saxophone, and one after another they all begin playing the same note, A440. Let's take this improbable scenario one step further and add that all four of these musicians are so adept at their particular instruments that each one is playing their A440 note at exactly the same volume.

So here you are in this darkened room listening to first a piano, then a flute, then a guitar, and then a saxophone, all playing exactly the same pitch (same frequency) at exactly the same volume (same amplitude). Even if our musicians try to get tricky and begin playing their A440's out of order, it's pretty obvious that neither you nor anyone else is going to be fooled. It's easy to tell the difference between a piano and a flute and a guitar and a saxophone, even with the lights out. Why is this? Because each one of these instruments (and every other acoustic instrument, for that matter), has a characteristic *timbre*.

How do we define the characteristic timbre of a sound? There are no units of measurement as there were with amplitude and frequency but instead we will see that timbre is in fact frequency-related. When the bizarre example we just gave was presented, we told you a little white lie, and that LWL (technical jargon for Little White Lie) was that each instrument was playing a frequency of 440 Hz. In fact no acoustic instrument can create a sound that consists of only one frequency. To explain this more clearly, let's examine further just what happened when our pianist played his A above Middle C: the string in question began vibrating as soon as it was hit with the piano hammer. The string vibration, first time around, looked like **figure 1-9.**

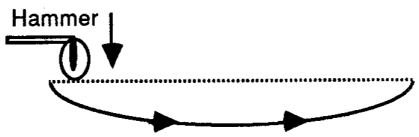


Figure 1-9

As you can see, in the first 1/440th of a second, the vibrational pattern is quite simple. But what happens when this movement reaches the back of the sounding board? At this point (some 2/440ths of a second into the sound), the string begins rebounding in the opposite direction, back towards the front of the board. Once they reach the front, the whole process starts all over again. All of this, of course, is happening in a period of time that is so short that it seems instantaneous to us. The end result? The patterns crisscross, causing a simultaneous string vibration of twice the frequency, or 880 times per second. **(see figure 1-10)**

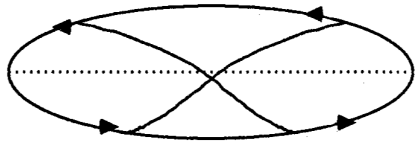


Figure 1-10

Typical pattern of string vibration:

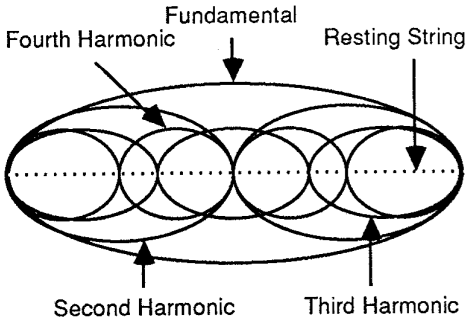


Figure 1-11

This process repeats itself when the string once again reaches the front board, now causing a third vibration of 1320 times per second, and yet again at the other end, causing a fourth vibration of 1760 times per second, and on and on. **(see figure 1-11)**

Eventually, of course, the string runs out of energy and stops vibrating altogether (in other words, the amplitude drops to 0). But until that time, these secondary vibrations, called *overtones*, may be present in astonishing strength, and may even occasionally overshadow the initial, *fundamental* vibration. Very simply, it is the *type* and relative *strength* of these overtones that determines a sound's timbre.

The initial frequency we started our string vibrating at (440 Hz in this example) is called, as noted above, the *fundamental frequency*. Overtones such as those described above whose frequencies are simply whole number multiples of the fundamental are called *harmonic overtones*, or *harmonics* for short. Thus, the second harmonic of any sound will always be the octave of that sound (remember, every time you go up an octave you double the frequency). So too will the fourth harmonic be double that of the second, and the eighth double that of the fourth. In other words, the second, fourth, eighth, sixteenth, and thirty-second harmonics (we rarely hear beyond this point, for reasons which will soon become apparent) are all octaves of the fundamental frequency.

But what about the other harmonics? Well, as luck would have it, the entire western musical system is based upon these overtones, which all are pleasing musical intervals of the fundamental. For example, the third harmonic is a musical fifth higher than the second, and the fourth harmonic is a musical fourth higher than the third. These relationships are important to be aware of when working with an FM system such as that provided by the DX7II.

When we heard that first A440 coming from the piano in the darkened room, what we were in fact hearing was a composite of many different frequencies blended together in a particular way, as follows:

Harmonic	Relationship	Frequency(Hz)
First	Fundamental	440
Second	Octave (8va)	880
Third	Fifth (8va)	1320
Fourth	Octave (16va)	1760
Fifth	Third (16va)	2200

right up until the point where we couldn't hear any more! If you keep multiplying 440 on and on eventually you'll hit the number 20,000 and, remember, unless your name is Spot or Rover, you probably can't hear anything above that frequency anyway. Unless your fundamental is an extremely low note it's rare that you will be able to hear anything higher than the thirtieth or perhaps thirty-second harmonic.

The important point to remember is that any time you hear any natural sound at all (and, for that matter, most electronic sounds), you are never hearing just a single frequency, but instead a composite sound composed of many frequencies blended together in a particular way. It is the type of blend which tells your brain what kind of timbre you are hearing.

I'm afraid we now have to complicate matters a bit further again as we inform you that in all likelihood when we heard our phantom pianist play the A440 we also probably heard a certain amount of, let's see, 3215.7 Hz as well. Also probably a certain amount of 602 Hz, and, um,

14,987.6 Hz, too. These numbers obviously have no mathematical relationship at all to our fundamental of 440 but nonetheless they were probably there due to many complex physical factors, including flaws in the instrument itself. These oddball overtones are called *inharmonics* (or *disharmonics*) and are always present to some degree in all acoustically (and often electronically) generated sounds.

It is the relative amounts of harmonics versus inharmonics that determines whether we consider a sound to be *musical* or *nonmusical.* Musical sounds, such as those our piano, flute, guitar, or saxophone would produce would certainly have much higher proportions of harmonics than inharmonics, but those sounds produced by jackhammers, coughing, wind, or motors humming will generally contain proportionally more inharmonics. The speaking voice has more inharmonics than the singing voice. A classical guitar will contain more harmonics than an electric guitar through a fuzz pedal. Generally, any sound which has a determinable pitch is considered musical and will usually have a basically harmonic overtone content, whereas sounds with no discernable pitch (most percussive sounds, for example) will have a greater inharmonic content.

To finally get to the point, the reason we were able to tell the difference between our four phantom instruments (even in the dark) was because each instrument had a particular *overtone content*; that is, each one had a certain amount of each particular harmonic and also certain amounts of particular inharmonics. These relative amounts not only allow us to tell the difference between different instruments but also between different types of the same instrument and even two supposedly "identical" instruments. When listening to pianos, the overtone content will allow us, for example, to distinguish between a 9' Steinway and a 6' Steinway; or between a Steinway grand and a Steinway upright; or between a Steinway upright and a Baldwin upright; or even between two Baldwin uprights manufactured the same day, standing side by side!

If all of this weren't mind-boggling enough, let's again remind you that timbre, as with the other two parameters of sound, *always* changes, and usually quite drastically, over the duration of the sound. A very common experiment in college-level electronic music courses is to have students record the sound of a single piano note on tape and then splice out just the very beginning (or *attack*) of the sound. When you listen back, your piano sounds nothing like a piano; in fact, it sounds much closer to an organ. The acoustic piano is one instrument whose sound has undergone severe scrutiny by acousticians, and it has been proven to generate one of the most complex sounds in existence. During the few seconds that we hear a single piano note, researchers have discovered that literally thousands of timbral changes occur. Take this along with the amplitude and frequency changes which are simultaneously occurring and the fact that everything becomes much more complex when chords are played (due to interharmonic relationships) and you will realize why - advertising claims notwithstanding - no manufacturer has yet produced a synthesizer that comes even close to recreating the sound of a real acoustic piano. No computer exists yet which can make all of these calculations quickly enough to actually reproduce this extremely complicated series of events.

We have seen that when we observe a sound on an oscilloscope, the height of the wave tells us of its amplitude, and the number of waves occurring per second tells us of its frequency. How can we tell anything about its timbre? The timbre of a sound will be reflected in the SHAPE of the wave on the oscilloscope. Sounds which are smooth and gentle will have smooth and gentle shapes, such as the waveshape of a flute sound. **(see figure 1-12)**

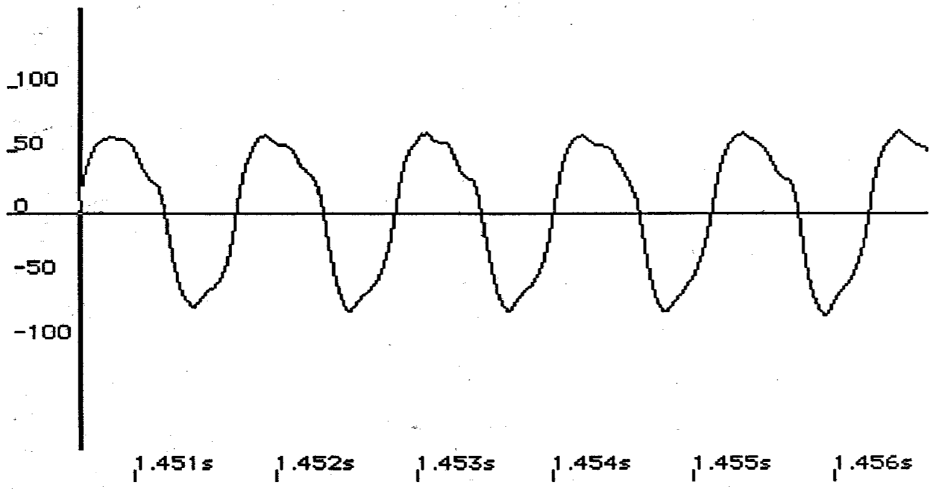


Figure 1-12

On the other hand, sounds which are harsh and raspy will have jittery and fairly irregular shapes, such as the waveshape of a snare drum sound. **(see figure 1-13)**

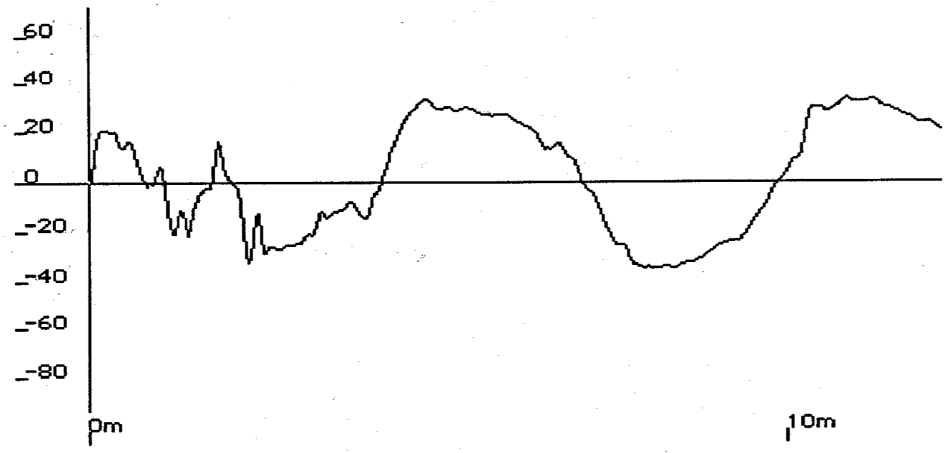


Figure 1-13

The oscilloscope cannot provide us with as accurate a means of defining the timbral parameter as it it does amplitude and frequency, but, as we see, it does help give us a clue as to the tonal quality of the sound we are observing; and, perhaps more importantly, in the way that this tonal quality changes over the duration of the sound. A more sophisticated analysis can be performed on a complex sound with a mathematical operation called a *Fast Fourier Transform.* This is typically performed by computers in research facilities, but some "home" computer programs allow it to be run as well. For more information on this procedure, along with graphic displays of such analyses, the reader is referred to my book, *A Synthesist's Guide to Acoustic Instruments*, published by Amsco Publications.

It cannot be stressed enough that a clear understanding of these three parameters, what they mean, what they are, and what they contribute to a sound, is absolutely vital in the understanding of how any synthesizer (not just the DX7II) works. Refer to this chapter often as you work your way through the rest of this book because it is all too easy to get wrapped up in the technicalities of synthesis and to lose sight of the fact that at all times you are manipulating one of these three parameters!

And Now, Back To Synthesizers

We began this chapter with a brief discussion of analog synthesizers and their system of creating sounds. Analog synthesis is often referred to as *subtractive synthesis* because the electronic components responsible for generating the original signal (the *oscillators*) present complex timbres at the outset. These complex timbres then pass through filtering devices that are used to remove undesired overtones. Hence the term "subtractive" since we always create sounds in analog synthesizers by *removing* overtones.

Digital synthesizers sometimes also use subtractive means of generating sounds. These instruments are often referred to as *hybrid* synthesizers, since they typically contain both analog and digital circuitry. Their sound sources (the aforementioned oscillators) will generally be digital oscillators, while the filtering circuitry is more usually analog. On the other hand, digital synthesizers can sometimes use exactly the opposite system, with a method called *additive synthesis*. The additive digital synthesizer (such as the Synclavier or Fairlight) will allow you to specify a fundamental frequency and then direct it to add overtones of particular harmonics (and sometimes disharmonics) in particular quantities. This lets you literally build sounds from scratch. The DX7II is capable of some limited additive synthesis but this is not the main method used by it.

Another type of digital synthesis system gives you the option of *sampling* real sounds. These instruments are not strictly synthesizers, but are more like digital tape recorders. The original sound (which may come from a microphone, tape, or any other kind of sound source) has to be fed into the computer as an electrical signal, and the internal device that converts this electrical signal into usable digital code (binary numbers) is the opposite of our old friend, the DAC: this is an Analog-To-Digital converter, or *ADC* for short. Essentially (and greatly simplified), if you plug an ADC into a DAC (and put some computer memory in between), you have the makings for a sampling system. **(see figure 1-14)**

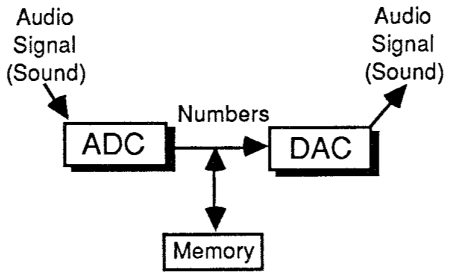


Figure 1-14

Although, as stated earlier, the DX7II does have some limited additive capabilities, it does not have an ADC capable of receiving audio input onboard and so you cannot sample sounds in a DX7II, nor, as of this writing, on any of the Yamaha DX or TX devices.

Another digital synthesis technique is called *phase distortion* and this proprietary method (developed by Casio for their "Z" synthesizers) produces sounds by a technique that is neither subtractive nor additive, but nonetheless similar to many analog systems. For more information on this, the reader is once again directed to *A Synthesist's Guide to Acoustic Instruments* .

The main method of synthesis that the DX7II employs is something called *digital FM* (short for Frequency Modulation). Instead of merely adding waves of different frequency and amplitude together, as additive synthesizers do, this system "collides" waves together. This means that we will not hear the two (or more) colliding waves individually, but

instead we hear the composite result of this collision. Of course, since we are talking about a digital, and not analog, system, there are no actual waves colliding at all. Instead, the microprocessor aboard the DX7II has been trained to quickly compute "What-would-happen-if" calculations. We abstractly give the computer input as to what kinds of waves we wish to theoretically collide, and the DX7II provides for us what would have been the result if these waves had actually existed.

This computer program was developed in the late 60's/early 70's by a team of musicians and computer scientists headed by Dr. John Chowning at Stanford University. Back in the prehistoric (by synthesizer standards) days of the mid-70's the Yamaha Corporation quietly purchased the license to this process and spent over a decade developing and refining the system into the modern DX7II. Although frequency modulation as a means of *modifying* sound has been a staple of analog synthesizers for many years, what sets the DX and TX instruments apart from their ancestors is that they incorporate FM *digitally* and as their primary basis for *generating* sounds. This system is proprietary and for any other manufacturer to use it (such as Synclavier) a royalty to Yamaha must be paid. For this reason, learning a digital FM system such as that employed by the DX7II will be of little or no use to you when you encounter any other type of synthesizer.

Which Is Better - Analog or Digital?

I'll bet a lot of you synthesizer owners out there skipped ahead to this section once you looked at the table of contents! I'm sorry to disappoint you but there unfortunately is no answer to this question. "Better" is too subjective a term, and what I might consider "better", you might consider "expletive deleted". The only statement that can fairly be made is that the two systems are qualitatively different.

Analog systems work with electrical signals which are notoriously unreliable and distorted. We don't notice changes in electrical signals in our daily lives because they are generally minute, but these small imperfections translate directly to true distortions when dealing with sound. Of course, you shouldn't think of the word "distortion" in a necessarily negative way because all acoustic instruments and sound sources *always* produce sound which is distorted somehow. There is no such thing in the real world as a perfect sound. Every Stradivarius has a flaw, and so too does every Stratocaster (even the vintage ones!). It is precisely these small and unpredictable distortions which lend a very "human", or "warm" sound to most good analog synthesizers.

On the other hand, digital machines don't suffer from the same kind of distortions. Because we are making sounds out of numbers, it doesn't matter what the temperature is, for example: the number 3 is still the number 3. It is not true to say that digital synthesizers cannot distort - they do - sometimes on purpose, sometimes not - but in general they will deliver much purer, cleaner sounds than analog. Some critics of digital sound use the term "sterile"; some critics of analog sound accuse it of being "muddy". The debate rages on...

Digital machines are newer and so have greater novelty value at the moment. The word "digital" itself is a buzzword. Analog synthesizers have been around since nearly the turn of the century, and their modern voltage-controlled versions since the 1950's. All too often the public confuses "newer" with "better" (talk to anyone who went out and purchased an 8-track cartridge player ten years ago). Digital is obviously here to stay, and the Yamaha digital FM systems are probably the most powerful and exciting synthesizers commercially

available at the moment, but I for one do not believe that analog is ever going to be completely replaced. Fortunately, digital and analog synthesizers tend to complement one another - what one does well, the other doesn't, and vice-versa. For example, don't hold out too much hope for your DX7II to produce the sound of whistling wind - even though you can get that easily from even the smallest analog systems. But on the other hand, don't expect to be able to get much of a Fender Rhodes sound from even the largest of analog systems - though your DX7II can do that without even blinking.

In general, analog systems are very good at generating lush, warm, powerful sounds; whereas digital systems are better at creating clean, pure, crystalline sounds. They can emulate each other to a certain degree but each excels at something different. The best solution? Buy one of each!

You think I'm kidding, don't you? Before 1982, I would have been. Today, I'm not, thanks to the advent of MIDI. MIDI is a topic that will be covered in detail towards the end of this book but for now we can just explain that MIDI is a language that allows different synthesizers - even radically different ones made by different manufacturers, to work together with each other and with most standard computers.

It's fairly obvious that digital synthesizers can communicate with computers (since they are, in fact, computers themselves), but how can analog systems "talk" to computers? The answer comes from DAC's cousin, the ADC chip. The relatively low cost of this component has encouraged virtually all analog synthesizer manufacturers in recent years to include onboard computers in all their machines. These computers have nothing at all to do with the generation of sound (if they did, the systems wouldn't be analog any more) but they act instead as silent watchdogs, keeping track of where all the voltages are traveling, and how. Once a sound is set up on the analog machine, the ADC can convert all the various routings into a digital code which can then be stored in the memory of another digital device called a *programmer*. This programmer can then recall and re-set-up any sound in its memory by simply routing all the analog voltages in the way that it remembered them as having been. The first so-called "programmable" analog synthesizer appeared in the late '70s (that's how new the technology is) and today all analog machines now are of this variety. It is worth noting that even though these types of hybrid synthesizers have computers on board, they may well not be true digital synthesizers.

In any event, because today ALL synthesizers of every variety - digital, analog, and hybrid - have computers onboard, they all have the capability to communicate with one another and with central computer terminals. The standard for communications is MIDI. The applications are enormous. If you are seriously involved in synthesizers you will probably want to own both digital and analog systems, and then simply hook them together with a MIDI cable.

For now, though, let's just concentrate on the DX7II and learn how we can use this amazing machine to its fullest capabilities.

Chapter Two ◇
The Front Panel and Types Of Memory

Let's begin our discussion of the DX7II with an examination of the physical layout of the instrument. All of the operations of programming the machine are performed with the three sliders and forty-six switches on the front panel, which is laid out in **figure 2-1.**

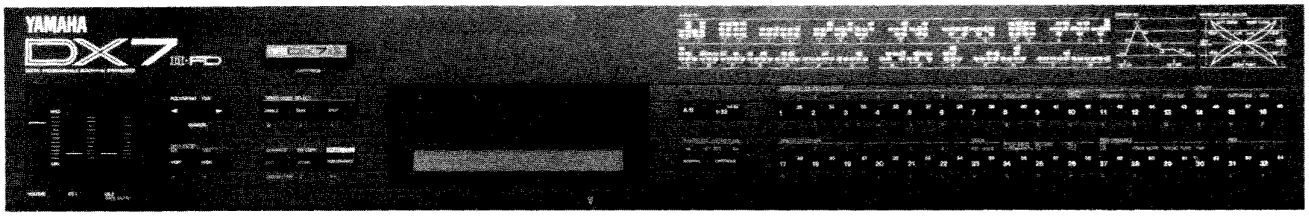


Figure 2-1

Going from left to right, here's what we see: First of all, the volume control slider - its operation is fairly straightforward, but bear in mind that the DX7II outputs a pretty low level signal and so for best signal-to-noise ratio (that is, maximum DX7II sound and minimum noise), it is advisable to usually keep it at or near the top of its range (just be careful that your amp is initially set at a fairly low volume before you do this!).

Immediately to the right of the volume slider are two more sliders, labeled *CS1* and *CS2*. "CS" stands for *continuous slider*, and these will allow you to make instantaneous changes to any DX7II sound you happen to be playing, in real time. The use and pre-programming of these sliders will be covered in greater detail in Chapter Thirteen ("Performance Parameters").

Note also that the CS2 slider also doubles as a *data entry slider* (as labeled below it in green). This will work in conjunction with the set of four switches to its right in order to allow you to select alterable parameters and to make changes to them. Specifically, the way these four switches work is as follows: the upper two, called *cursor switches*, are used to move a computer prompt, called a *cursor* within a display - allowing us to select a particular parameter. These two switches - both with status lights - also have some other special functions - *Pan* and *Poly/Mono* - that we will talk about later. The lower two switches (labeled "yes" and "no" are used, in conjunction with the data entry slider, to enter numbers into the computer, and, occasionally, to answer questions that it will ask us. We'll talk more about this entire *data entry section* very shortly, but in short, these controls are used to send commands and numbers ("data") to the DX7II - in other words, they allow us to "speak" to the computer.

Next, we see a series of six switches, arranged in two rows of three each. The top row and the lower right-hand switch are all labeled in white and are referred to as *play mode select* switches. They let you access two of the main memories onboard - *voice* memory and *performance* memory - and each, like the cursor switches, have status lights above them. We'll be talking about these (and other) memories in great detail towards the end of this chapter. The remaining two switches, labeled *edit* and *store* perform other special functions. Note that all six of these switches have various other labels above and below them (for example, the "store" switch says "EG copy" above it and "Z" below it). This is because they, like the thirty-two switches at the far right-hand side of the instrument, are all *multi-function* switches, meaning that they can each perform more than one operation, depending on what *mode* the instrument is in.

The next thing we encounter is an LED/LCD display. LED stands for "Light Emitting Diode" and is the electronic component giving us the one or two red numbers above the display. The meaning of these numbers will be explained shortly. Below the LED is an LCD, or Liquid Crystal Display, similar to that seen on many digital watches. The DX7II LCD can display up to two lines of 40 "characters" - letters or numbers - each. Taken together, the LED/LCD display is the area where the computer "speaks" to us. In this area, the instrument will be able to show us information about the sound we are currently using, and also to occasionally ask us questions or inform us of operations the machine is performing.

Immediately to the right of the display is another set of four switches, arranged in two rows of two each. These serve special functions, to be covered in detail later. Note that the upper two, like the mode select switches and cursor switches on the left-hand side, all have small status lights above them. The lower two switches also have special green edit functions as well.

Finally, on the far right-hand side of the instrument are a series of thirty-two more switches. These controls, like all the other switches on the instrument, all have labels on them, above them, and below them - telling us that they, too, are multi-function switches. Why do all these switches have so many different functions? The answer lies in the fact that the DX7II needs a whole lot more than just thirty-two main controls; it actually requires over a hundred! Presenting the public with a machine with over a hundred switches can't exactly help sales, and besides, providing all those switches would have been prohibitively expensive. Instead, using common digital design concepts, the Yamaha design engineers provided only thirty-two switches, but made each of these a multi-function switch. Makes perfect ergonomic - and perfect economic - sense.

Let's look at these thirty-two main switches a little more closely. **(see figure 2-2)**

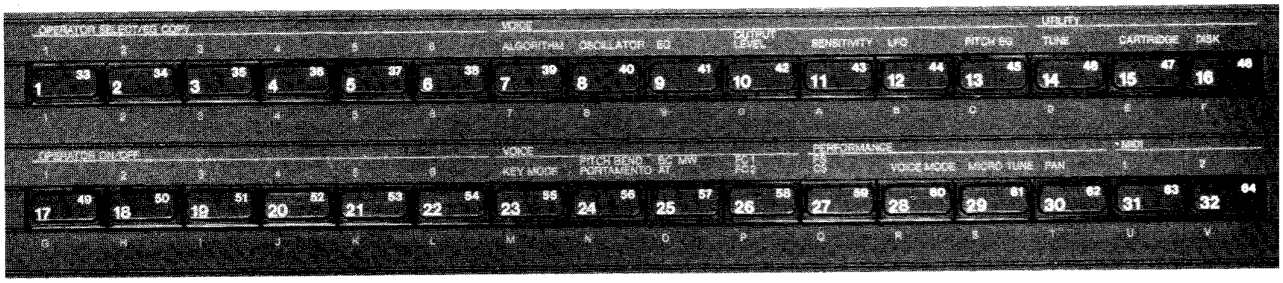


Figure 2-2

As noted above, each of these has something written over it in green and a character below it in brown as well as the two numbers written on it in white. The key to using these switches is to be certain at all times whether they are active for their blue functions, their brown characters, or their white numbers. Each switch can only be active for one of these things at any one time. The way this works is actually pretty simple and logical and is tied in with which of the six switches to the left of the display you have pressed most recently. If your most recent selection was one of the four *mode select* switches (the ones labeled *single, dual, split,* or *performance*), then the thirty-two main switches will be active for their white numbers. If your most recent selection, on the other hand, was the green *edit* switch, then the thirty-two main switches will be active for their green *edit* functions. The brown characters, used for naming sounds, can be accessed only at certain times, and only if you press and hold down the edit switch. We'll talk more about them in Chapter Eight ("Storing Sounds"). Another major mode of operation is the *store* mode, which allows us to actually store data in memory. This, as you might suspect, is accessed by pressing the pink *store* switch (the middle switch in the bottom row of six), but, again, we will cover this operation in greater detail in Chapter Eight.

For the purposes of this book, we will refer to the thirty-two main switches according to their specific mode of operation: for example, "edit switch 10" means main switch number ten, activated in edit mode (for this particular example, "Output Level"). **(see figure 2-3)**

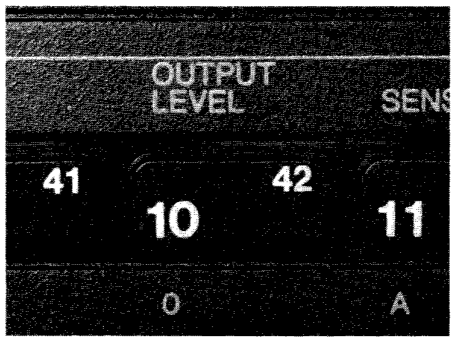


Figure 2-3

Memory

The first thing that all DX7II owners naturally want to do is to play the sounds stored in memory (these sounds are normally referred to as *presets*). Since the DX7II is a completely digital instrument, what we are really doing when we play back these sounds is accessing our instrument's *memory* and calling up digital data stored either inside the memory of our machine or in its memory cartridges or diskettes (more about these later).

This is therefore a good time to actually define the term "memory". Computers are able to "memorize" information by storing large amounts of binary numbers (zeroes and ones) representing this information in coded form within a series of electrical switches. Essentially, switches are quickly being turned on and off; wherever a switch is left on, the computer recognizes a "one" and wherever a switch is left off, the computer recognizes a "zero". Because a computer memory circuit typically contains many thousands, if not millions, of these switches, large amounts of encoded information can thus be stored. This information may be in the form of instructions to tell the computer what to do at all times (program instructions) or it may simply be "voice data", that is, numbers which will eventually be translated by the DAC into a particular sound. All of these instructions fall under the general heading of *software*.

Aha! This is the first time this mysterious word has surfaced in this book and so we should offer a brief explanation to any of you who are not sure what the word means. Computer programmers earn their living by writing these kinds of instructions to a computer. Computers are actually incredibly stupid (but incredibly fast) devices. They need to be told exactly what to do at all times, and these very fine and precise instructions are what the software provides. Software should be differentiated from *hardware*, which are the actual nuts-and-bolts physical workings of the machine. Think about, for example, a home

cassette deck. The casing, meters, tape drive, and switches on the panel are its hardware, but without software, in the form of magnetic instructions on a cassette tape, you won't hear a darned thing! The machine is, in effect, useless without a tape playing in it. Same with the DX7II. If Yamaha's and Stanford's programmers had not developed very specific (and very complex) software to tell the machine what to do at all times (for example, "when the 'internal' and 'Single mode select' switches are depressed and subsequently followed with the depression of a main switch, search and retrieve data stored in the associated internal memory slot"), all we would have for our $2,195 is a pile of switches, displays, and microprocessors attached to a keyboard which cannot make sounds. Not much good to most of us...

(i) *Microprocessors* (logic circuits) inside the actual computer are able to utilize this data in memory to perform computing operations. The memory circuits inside the computer may be capable of keeping their switch positions intact if somehow electrical power can be supplied to them even though the computer itself is turned off. This protective auxiliary power is normally supplied by small batteries, called "back-up batteries" which usually have a long life - five years or more. Alternatively, the data stored in these switches can be stored "off-line" - outside the computer, in devices like tape, floppy disks, or special data cartridges. The computer can electronically retrieve this data (*read* it) by many different means.

Our DX7II, we know, is itself a full-fledged, card-carrying computer and it utilizes both internal and off-line data storage. These storage units are therefore commonly lumped under the umbrella heading of memory. The DX7II thus has both internal and external memory. The internal memory is a sizable memory circuit inside the machine protected by a back-up battery, so it can "remember" data even though the power to the instrument may be off. The external memory can be either the storage media of *data cartridges* (so-called *RAM* or *ROM* cartridges) or, if you have the "FD" model of this instrument, 3 1/2" microfloppy diskettes. The two "internal-cartridge" switches immediately to the right of the LCD display allow us to specify whether we wish to access data stored in the internal memory or in cartridge memory. If you have the "FD" model, disk functions are accessed with edit switch 16 (more about this in Chapter Eight).

A good way to think of memory is to picture a hotel lobby. Behind the hotel desk we see a series of pigeon-holes, used to store messages and keys for guests. A *memory slot* is equivalent to one of these pigeon-holes, with one main difference. In our hotel lobby, any number of messages, or data, can be stored in each slot; but in our computer, only one "message", or piece of data, can be stored in a slot at any one time. If you tell the computer to put data into a particular slot, it will be happy to accommodate you (provided you followed the correct storage procedure) but it will also remove and throw away whatever "message" was previously in that slot. This means that you will have to be extremely careful when storing data in various memory slots because the computer will *always* have to destroy whatever was previously in that slot - and it will do it automatically.

The internal memory of the DX7II has a total of a hundred (!) slots, or pigeon-holes, in which we can store data. These are broken down as follows: there are sixty-four *voice* slots, thirty-two *performance* slots, two *microtuning* slots, one voice *edit recall buffer*, and one performance *edit recall buffer*. Additionally, there are two more temporary "holding areas" - one for voice data and one for performance data. Don't get too blown away by all this jargon - one by one, we'll cover all of these different areas - just stick with us!

When working with the internal memory, we have the choice of either accessing the information and hearing the sound the data will create, (*read* data), or we can replace it with new sound data, (*write* data) back into the same, or into a different memory slot.

The cartridge memory which the DX7II uses comes in two different formats: *RAM* (for *Random Access Memory*) cartridges, and *ROM* (for *Read Only Memory*) cartridges. RAM cartridges work much like the internal memory - you can read the data stored in them and you can write new data to them. ROM cartridges are different, however, in that you can only *read* data stored in them. You cannot write data to any kind of ROM, including a ROM cartridge - which means that information stored there is, in effect, permanent.

In order to get data from a cartridge, we will need to insert it gently into the provided slot on the left side of the instrument. Always have the front label, the one that says "Yamaha Data ROM" or "Yamaha Data RAM", facing you so you can read it without doing a headstand. This will insure that the cartridge is in the right way around. In no event should you ever force a cartridge into the slot: if it doesn't go in easily, you're doing something wrong. **(see figure 2-4)**

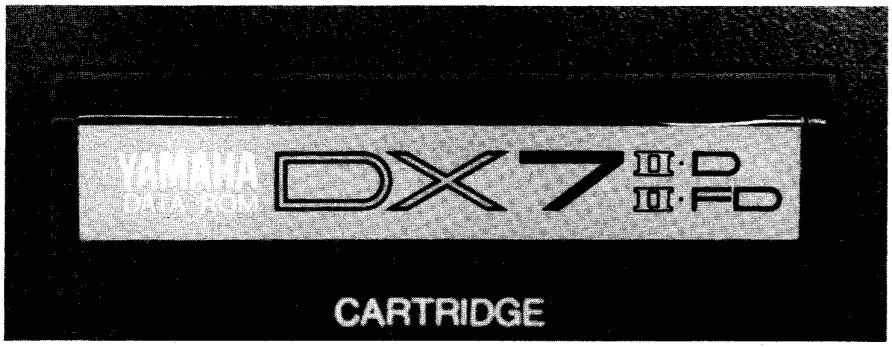


Figure 2-4

When you purchase your DX7II you will normally receive one ROM cartridge and you can also buy RAM cartridges. The DX7II ROM cartridge actually holds a lot more information than either the RAM or the internal memory, holding *two* sets of sixty-four voices, thirty-two performance memories, two microtunings, etc., along with other data called *fractional level scaling* (see Chapter Twelve for a detailed discussion of this function). You can choose which *bank* of data you wish to access at any time with edit switch 15 - and we'll be running an exercise very shortly to try this out. The RAM cartridge, on the other hand, will usually store exactly the same amount of information as the internal memory does (though it can also be formatted to hold other kinds of DX7II data - see Chapters Twelve and Fourteen for more information). Because the DX7II cartridges hold more information than the original DX7 cartridges, they are slightly larger. However, because the DX7II is *upwardly compatible* with the original DX7 (meaning that it can access and play back DX7 voices but the DX7 cannot access and play back DX7II voices), provision has been made for using the smaller DX7 cartridges. This is accomplished with the use of a clever plastic adapter available from your local friendly Yamaha dealer (who may actually not be so friendly when he finds out that that's all you want to purchase from him!).

The ROM cartridges (as well as your internal memory, when you first buy the instrument) contain sounds which, as we mentioned earlier, are called *presets*, simply because they have been pre-programmed by someone employed by Yamaha. While many of these

sounds are quite good and quite useful, we should not lose sight of the fact that these sounds did not come down from the heavens but were actually created by mere human beings. The fact that, for reasons described earlier, the DX7II is not the simplest of synthesizers to program, explains why an undue reliance has been placed on these 128 sounds by DX7II owners everywhere. We can easily, however, modify these presets to taste, or even create brand new "presets" from scratch with the multitude of controls provided on this instrument, and that, after all, is the reason for this book in the first place. If all that were involved in using a DX7II were calling up presets, people like myself would have to find a new line of work!

In any event, playing back preset DX7II sounds is a very simple operation: press any one of the four mode select switches (that is, "single", "dual", "split", or "performance") in order to activate the thirty-two main switches for their white numbers. These switches will now allow us to select and hear any of the sixty-four voices or thirty-two performance memories stored in the internal memory, RAM cartridge, or either bank of the ROM cartridges.

At this point, we should define just what we mean by the terms "voice memory" and "performance memory". A "voice" is just a synonym for a "sound" (or "patch", if you go back to analog terminology) - but remember that because this is a digital synthesizer, we are really referring to a set of numbers, or data. The numbers that make up that particular sound can be accessed ("read") and changed if necessary ("written") in *edit* mode. However, here, all we want to do is to simply read that data into memory. The DX7II memory (whether internal or cartridge) can store sixty-four of these voices, but what sets this instrument apart from its precursor (the DX7) is that it can call up one *or* two of these voices at a time!

If you want to call up only one voice, then you will want to work in *single* voice mode. In this mode, the maximum *polyphony* of the DX7II is sixteen notes - meaning that you won't get any *note robbing* (which means some notes drop out in order to accommodate new notes - more about this in Chapter Three) until you play seventeen or more notes.*

Alternatively, you can call up two voices simultaneously. If you want two voices, there are two different ways you can *allocate* them over the keyboard, depending on whether you choose *dual* mode or *split* mode. In dual mode, both voices (labeled *voice A* and *voice B* for convenience) appear together over the entire length of the five-octave keyboard. In this mode, since each key is actually calling up two voices at once, the total maximum polyphony is halved* - meaning that you can normally only play eight notes at a time before note robbing occurs.

The other option you have available to you should you want to call up two voices simultaneously is to have the two voices over different parts of the keyboard - that is, *split* mode. In this mode, one voice (*voice A*) will normally appear from the bottom of the keyboard on up to Middle C (that's the key with the marker "C3" just above the keyboard) and the other (*voice B*) from C# above Middle C on up. In this instance, Middle C is referred to as the *split point*, which explains the term "split". This split point can be changed, of course, but we'll get to that a bit later. Because the total maximum polyphony of this instrument is sixteen, it stands to reason that each voice in split mode can normally play a maximum of only eight notes before note robbing commences.

* Note that the polyphony of a particular voice will be quartered if you use that voice in a special *key mode* called *unison poly* - or it can even be monophonic if you choose *mono* or *unison mono* key mode. These various types of key modes will be discussed in detail in Chapter Thirteen.

The fourth play mode allows you to access the DX7II's performance memories. Each performance memory slot contains information about whichever voice or two voices you want to use, including the programmable split point (if in split mode), the relative volumes and tunings of each (if in split or dual mode), the uses of the sustain pedal and continuous sliders, and microtuning that is, keyboard scaling and stereo panning information. In this way, you can create a "preset" of one *or* two voices, along with a wealth of other information regarding the voices or any interactions that you want to see occur between the two voices (if in split or dual mode).

Whenever you are in any of the three voice modes (single, dual, or split), you can edit the voice data by simply pressing the green edit switch (to the left of the data display - the leftmost switch of the bottom row row of three). Similarly, when you are in performance mode, you can edit performance data. One real problem to be aware of is that the DX7II *does* let you edit performance data from voice mode, and vice versa. But, as the manual points out, you cannot *store* edited performance data in voice mode, just as you cannot store edited voice data in performance mode! Sounds confusing, I know, but the rule of thumb is simple: even though you can do it, don't alter voice data from performance mode or vice versa. The use of a special part of memory called the *edit recall buffer* will let you, in a roundabout way, recover any incorrectly edited data, but it's a fairly complicated procedure, and one that is therefore not advised.

Another important rule of thumb is this: You can only enter a play mode (that is, single, dual, split, or performance) from another play mode - or from edit mode if and only if you have been editing that play mode. For example, if you are in single mode, you can go to dual, split, or performance mode with ease. But if you are in single *edit* mode - that is, you've been editing a single voice - the only play mode the instrument will let you access is single mode. From there, of course, you can then move on to any other play mode. If you try to go directly to, say, dual mode from performance edit mode, the LCD display will tell you in no uncertain terms, "** Not available in this mode **". This display can be infuriating if you don't understand why the computer is preventing you from escaping a mode you don't want to be in, but just remember the simple rule above and you won't run into this problem.

The LCD display always tells you which of the four play modes you are in. The upper left-hand corner will display either the word "Voice" or the word "Performance". If you see anything other than that, you are in one of the edit modes. If you are in a voice play mode, you will additionally see the word "Single", "Dual", or "Split", confirming (along with the status light above the appropriate switch) which one of those three modes you are in. The display also tells you which memory (internal or cartridge) you are accessing. You select internal or cartridge memory with, of course, the *Internal/Cartridge* switches immediately to the right of the display. If you are accessing a single voice from the internal memory, it will look like **figure 2-5.**

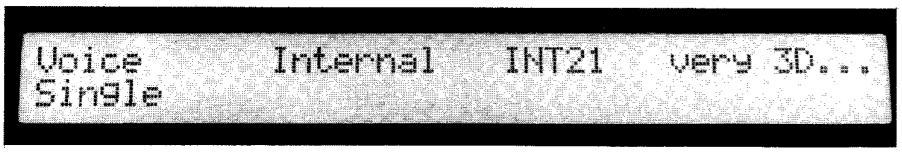


Figure 2-5

Whereas if you are accessing a single voice from cartridge memory, the word "Internal" on the top line will change and the "INT" before the voice number will become a "CRT", like **figure 2-6.**

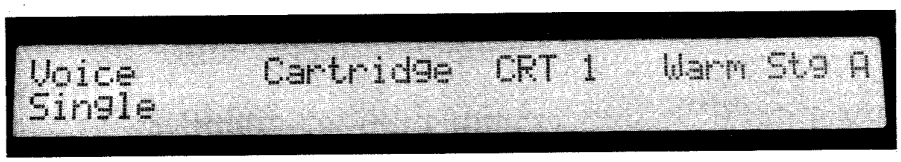


Figure 2-6

As mentioned above, there are sixty-four voices stored in memory at any time - but there are only thirty-two main switches. How then can we access all sixty-four sets of data? Simple - that is the purpose of the *1-32/33-64* switch located directly above the "Cartridge" switch (that is, directly to the left of main switch 1). When pressed, the status light above it will go on and main switches 1-32 now actually become main switches 33-64! Press it again, the status light goes out, and your main switches once again become 1-32. Bear in mind also that when you change from internal to cartridge memory, or vice versa, you will also have to press one of the thirty-two main switches in order to actually change the sound - just pressing the "Internal" or "Cartridge" switch alone will change the display only but will not actually enter the data into the instrument.

If you are accessing two internal voices in dual mode, the display looks like **figure 2-7.** If you have a cartridge physically in the slot, the DX7II can actually access both a cartridge and an internal voice in this mode. You specify which voice you'd like to change by using the *A/B* switch directly to the left of the *1-32/33-64* switch. Press it and the status light goes on - and the "Internal" moves down to the second line. **(see figure 2-8)**

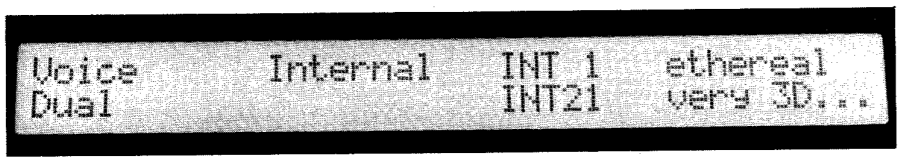


Figure 2-7

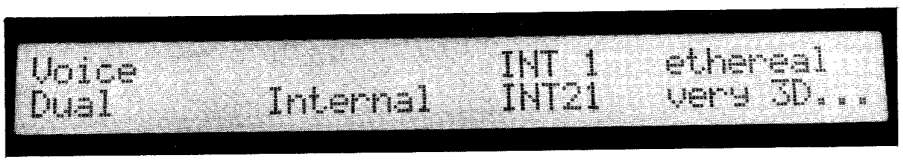


Figure 2-8

At this point, pressing "cartridge" and then any one of the thirty-two main switches (along with the *1-32/33-64 switch*, if necessary) will allow you to blend internal voice A with cartridge voice B (or vice versa), with the display reflecting the change. **(see figure 2-9)**

If you are accessing two internal voices in split mode, the display looks like **figure 2-10.**

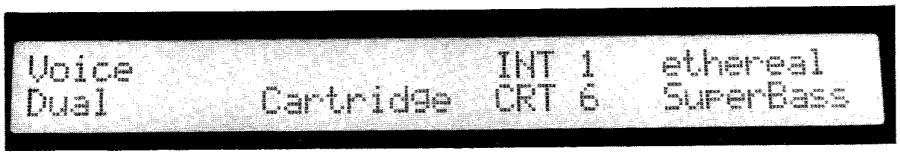


Figure 2-9

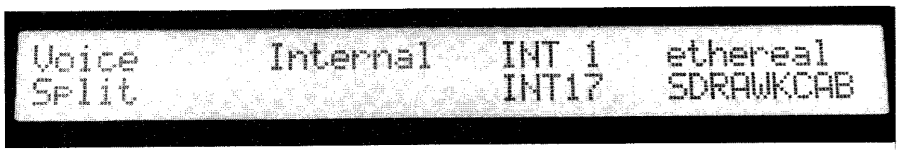


Figure 2-10

As in dual mode, you can combine an internal voice with a cartridge voice by following the same procedure. The display would then reflect that change, like **figure 2-11.**

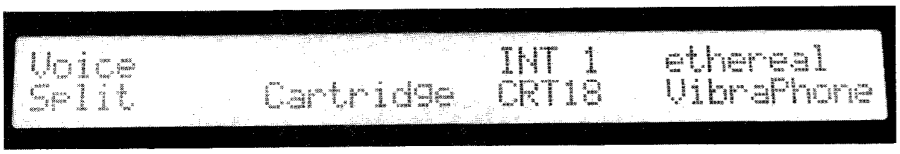


Figure 2-11

In performance mode, the display looks quite different. **(see figure 2-12)**

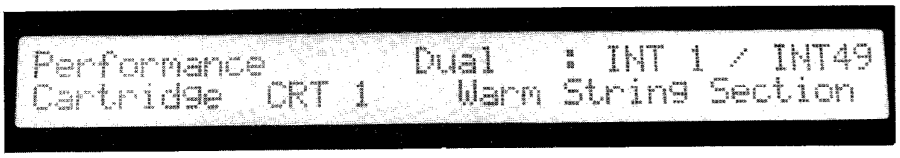


Figure 2-12

Here, the top line shows us the performance memory number along with the voice mode being used and the numbers of the internal and/or cartridge voices employed. As with dual and split mode, you can access voices from both internal and cartridge memory simultaneously.* The bottom line of the LCD tells us whether this performance memory itself has been accessed from the internal memory or from a cartridge. If you call up an internal performance memory that is programmed to access a cartridge voice and there is no cartridge in the slot, however, be prepared for some weird goings-on as the computer - well, to put it mildly - gets a bit confused. The purpose of this book is not to search out all of the bugs that are floating around in the DX7II software but from time to time we will warn you of traps like these that should be avoided.

The LED display is also giving important information - showing which voice numbers are being used for voice A and voice B. Various status lights above the switches will also light as you travel from mode to mode. In the three voice modes, these have considerably more meaning than they do in performance mode, as they are telling you whether you are actively accessing voice A or B, for example, and whether you are accessing voices 1-32 or 33-64. In performance mode, the performance status light naturally lights up as will one of the voice status lights - telling you (as is the LCD) which voice mode is being used for this performance memory. Additionally, the *A/B* and/or *1-32/33-64* lights may be on as well. These are simply showing you their status at the time that the performance memory was written.

* Bear in mind that the performance memory only remembers the numbers of the voices used, not any voice data itself. Therefore, if you store different sounds into your internal or RAM cartridge memory than were originally there when you created and stored the performance memory, you may get unexpected results!

In summary, the LCD, LED, and status light displays provide a means for the computer to "communicate" with us and confirm that it is correctly responding to the commands we have issued. Let's try a couple of quick exercises to access our DX7II's internal and cartridge memories:

Exercise 1:
Accessing the internal memory

1) Look at your LCD in order to determine what mode your DX7II is currently in. If your instrument is already in play mode, simply press the *Single* mode select switch. Observe that the status light above the *Single* switch lights up. If you are currently in an edit mode (the top left-hand word in the display is not "Voice" or "Performance"), check the status lights to determine which edit mode you are in (i.e. if the *Performance* light is on, you are in Performance edit mode; if it is off and the *Dual* is lit, you are in Dual edit mode) and press the appropriate (lit) mode select switch to return to play mode - then press the *Single* mode select switch.

2) Look again at your LCD to determine whether you are accessing your instrument's internal or cartridge memory (look for the word "Internal" to confirm the former or "Cartridge" to confirm the latter). If your DX7II is currently accessing cartridge memory (which, of course, it can only do if there is a cartridge physically in the slot), press the *Internal* switch to change it over to the internal memory.

3) Check the status light above the *1-32/33-64* switch. If it is lit, press the switch to turn it off. If it's already off, you don't need to do anything here.

4) Press any or all of the thirty-two main switches.

5) Observe: LED displays the number of the sound you have selected.

6) Observe: Top line of LCD reads "Voice Internal", followed by "INT" and the number of the voice, followed by the name of the voice.

7) Listen: The sound you selected is there!

8) Try: Selecting each of the first thirty-two sounds stored in your DX7II's internal memory, listening as you do so.

9) Press the *1-32/33-64* switch. Observe that the status light above it lights up.

10) Observe that the number of the voice currently selected has 32 added to it (that is, if you were looking at voice #32, it has now become voice #64) and that the voice name has changed as well. Listen and note that the sound has also changed!

11) Try: Selecting each of the second thirty-two sounds (voices 33-64) stored in your DX7II's internal memory, listening as you do so.

12) Press the Dual mode select switch. Observe that the LCD display has changed accordingly (similar to that in figure 2-7).

13) Check the status light above the A/B switch. If it is on (indicating that we can currently change voice B), press the switch in order to turn it off. If it is off (indicating that we can currently change voice A), then you won't need to do anything here. Observe that the "Internal" in the LCD display changes from the top line to the bottom (or vice versa) as you repeatedly press the A/B switch. Make sure the A/B status light is OFF before you move on to the next step.

14) Check the status light above the *1-32/33-64* switch. If it is on (indicating that we can currently access voices 33 through 64), turn it off. Observe that as you do so, the LCD display changes, subtracting 32 from the currently selected voice.

15) Try: pressing each of the thirty-two main switches in turn, listening as you do so. Observe that the overall sound changes as you blend different voice A sounds with your currently selected voice B.

16) Press the *1-32/33-64* switch (turning the status light on) and try pressing each of the thirty-two main switches again. Observe that the overall sound changes again as you blend yet another 32 voice A sounds with your currently selected voice B.

17) Press the *A/B* switch (turning the status light on and therefore selecting voice B) and repeat steps 14-16 above. Experiment with different combinations of voice A and B. If you are using both audio outputs, note that voices A and B can be sent to different speakers, making for a pleasing stereo effect.

18) Press the *Split* mode select switch (turning on its status light) and observe that the LCD display has changed accordingly (similar to that in figure 2-10).

19) If either the "A/B" or "1-32/33-64" status lights are lit, press the switches in order to turn them off.

20) Try: pressing each of the thirty-two main switches in turn, listening as you do so. Because you are changing only voice A (since the "A/B" light is off and the word "Internal" is therefore on the top line of the LCD), observe that only the keys above Middle C (C3) are given a different tonality. Notes played below C3 (voice B) stay the same.

21) Press the *1-32/33-64* switch (lighting the status light) in order to access voices 33 through 64 for voice A. Observe again that only keys above Middle C change sound.

22) Turn off the "1-32/33-64" status light by pressing the switch and turn on the "A/B" light by pressing its switch. Observe that the "Internal" in the LCD shifts down to the bottom line.

23) Try: pressing each of the thirty-two main switches in turn, listening as you do so. Here we are changing voice B only, and you should observe that only keys below Middle C change tonality.

24) Press the *1-32/33-64* switch again (lighting its status light) and experiment with assigning voice B any of these thirty-two sounds.

25) Press the *Performance* mode switch. Before pressing any more switches, play the keyboard and observe that the same two voices you had last selected are still there! Whenever you first enter Performance mode, the DX7II sets you up with the last voice or voices and last voice mode that you selected. Observe that the LCD confirms this, showing you the same two voices and the same mode (split mode, in this case). Note also that the Performance memory name is "INIT PERF".

26) While keeping your eye (or both eyes, as you like) on the LCD, press any one of the thirty-two main switches and note that the name immediately changes. Play a few notes on the keyboard in order to confirm that the sound has changed as well. You only actually enter Performance play mode *after* pressing one of these main switches. At this point, you are accessing the thirty-two internal performance memories. Note that, while they may or may not be lit, neither the *A/B* nor the *1-32/33-64* switches can be activated since you cannot access individual voices in performance mode, and since there are only thirty-two performance memory slots in total. If you are using both audio outputs, you might also try turning on the *Pan* control by pressing the right cursor switch (immediately to the left of the Single mode select switch) in order to hear some interesting stereo panning effects (the programming of these effects will be covered in detail in Chapter Fourteen).

Exercise 2
Accessing the Cartridge Memory

1) Place a ROM or formatted RAM cartridge (the formatting procedure is described in Chapter Eight - but if the RAM already has data in it, you can be sure it's already formatted. If it's never been used before, set it aside for awhile and use your ROM for this Exercise instead) into the cartridge slot of your DX7II.

2) Check your LCD display to determine if your DX7II is in one of the four play modes (that is, the word "Voice" or "Performance" appears in the upper left-hand corner). If not, press the appropriate mode select switch to return it to play mode.

3) Look again at your LCD to determine whether you are accessing your instrument's internal or cartridge memory (look for the word "Internal" to confirm the former or "Cartridge" to confirm the latter). If your DX7II is currently accessing cartridge memory, you won't have anything else to do here. If, however, it is currently accessing its internal memory, press the *Cartridge* switch to change it over.

4) Check the status light above the *1-32/33-64* switch. If it is lit, press the switch to turn it off. If it's already off, you don't need to do anything here.

5) Press any or all of the thirty-two main switches. Observe: the LED displays the number of the sound you have selected.

6) Observe: top line of LCD reads "Voice Cartridge", followed by "CRT" and the number of the voice, followed by the name of the voice. Listen: The sound you selected is there!

7) Repeat steps 8 through 26 of Exercise 1, this time accessing data stored in your DX7II's cartridge memory.

The Edit Buffer

A couple of pages back, you may remember a somewhat teasing mention of something called the *edit buffer*. Where is this elusive bit of memory and how can we access it? Here's a simple experiment to answer both questions at once:

Exercise 3
Discovering the edit buffer

1) Put a ROM cartridge in the slot of your DX7II.

2) Go to *Single* voice play mode by pressing the appropriate switch, and press the *Cartridge* switch in order to access the cartridge memory.

3) Select a voice from the cartridge at random. Any one will do.

4) Play the keyboard. The sound you selected should be there.

5) Now *remove* the cartridge.

6) Play the keyboard again. Listen. Amazingly, the same sound is still in the machine *even though you removed the cartridge!*

What's going on here? We were able to hear the selected sound because the DX7II read data from the ROM cartridge. When the ROM cartridge was physically removed from the machine, however, somehow the DX7II was still able to access its data. Or was it?

The answer to this mystery is the presence of a wonderful little piece of DX7II memory called the *edit buffer*. This can be thought of as a "sixty-fifth" internal memory slot and the way it works is this: Whenever you go into play mode and select a sound from any memory source - internal or external - the DX7II will *not* actually give you the sound. Instead, what our clever little computer does is to make a *copy* of the sound we've picked. It then automatically puts that copy in the edit buffer. You can think of the edit buffer as being a kind of scratch

pad, a place where you can mess around with the sound to your heart's content, secure in the knowledge that at no time are you actually doing anything to the sound itself. The original is still residing securely in whatever memory slot it was stored in. The only thing you are altering is a *copy* of that sound.

Anytime that you do anything at all on the DX7II, you are always working in the edit buffer. That means that you are never really working with the sound you select, only a copy. This allows you to alter and modify sounds in your DX7II to any degree without ever altering the sounds themselves! In order to actually change the originals, you will have to go through a fairly complicated *store* procedure (covered in detail in Chapter Eight), and this is something you are extremely unlikely to do by accident, since it involves pressing and/or holding down several different switches, in a particular order.

The edit buffer is one of the most wonderful features on an already wonderful machine. It gives you the freedom to experiment and manipulate and it is automatically provided to you at all times. There is no way to bypass this feature, nor could there be any reason why you would particularly want to.

Edit Mode

This, of course, is the mode that lets us actually make alterations to a Voice or Performance memory that we have called up. When you enter this mode (accomplished by simply pressing the green *edit* switch), all of the thirty-two main switches become activated for the functions written above them in green. Voice parameters can be accessed in this way by using main switches 7 through 13 and 23 through 26, and, occasionally (the mystery will soon unfold!) switches 1 through 6 and 17 through 22). The DX7II manual breaks these down into what they call "Voice" and "Voice Edit" parameters, but, believe me, they are all in fact voice parameters (the distinction apparently comes from the fact that edit switches 7 through 13 access parameters that are DX7-compatible, while edit switches 23 through 26 access parameters not stored or transmitted by the DX7. My advice is, think of *all* of them as being voice parameters).

Performance parameters are accessed (and changed, if so desired) by using edit switches 27 through 30. Additionally, there are *utility* parameters (relating to the internal, cartridge, and - if you have the "FD" model - disk memories) as well as *MIDI* parameters (stored in something called *system setup* memory - more about this in Chapter Fifteen), accessed with edit switches 31 and 32. You alter both utility and MIDI parameters by putting the instrument into edit mode, but unlike specific voice and performance edit parameters, they can be changed and stored from *any* edit mode. That is, you can alter them from either performance edit *or* voice edit mode. This special feature aside, they act like most other edit functions.

Once again, we caution you (as does the DX7II manual) to be careful not to get in the bad habit of making edits to performance parameters while in voice edit mode, or of making changes to voice parameters while in performance edit mode. Utility and MIDI changes, as noted above, can be made from any edit mode.

In the various play modes we have discussed, we have observed that the top line of our LCD constantly informs us of which mode we are in, and the same is true of edit mode, although not in such an obvious way. Here, the *lack* of any kind of standard display will tell you that you are in an edit mode. The LCD may be saying almost anything, depending upon precisely which edit parameter you have called up. The

point is, if you *don't* see the word "Voice" or "Performance" in the top left-hand corner of your LCD, then you know that you are in an edit mode. In order to ascertain which edit mode you are in (e.g., single voice edit mode, dual voice edit mode, split voice edit mode, or performance edit mode), simply check the status lights above the various mode select switches, as noted above. Remember again that you can only enter a play mode from edit mode if and only if you were editing that particular play mode - so keep track!

Working in edit mode is a bit different than working in play mode, but the rules are simple. First of all, as stated above, the thirty-two main switches will no longer be active for their white numbers so you won't see any change (apart from the appearance of a mysterious decimal point - more about this in Chapter Eight) in the LED. Secondly, most of the main switches in edit mode provide multiple LCD displays and often you will have to press the switch more than once in order to cycle through all the different ones available. Even when you arrive at the desired display, you will use the *cursor* switches (the two switches with the left-right arrows above the data entry switches) to move a blinking black box called the *cursor* to the appropriate parameter. A cursor is something found in all computers. It is simply a prompt - a way for the computer to try to "get your attention" and let you know that it is waiting for a command to do something specific. In this case, the cursor will be telling you that the DX7II microprocessor is ready and willing to make a change to a particular edit parameter. You will be using the data entry section - the data entry slider and the two data entry switches (marked *-1/off* and *+1/on*) in order to actually enter new data into the selected parameter. The data entry slider (which is inactive until you actually move it) will make rapid changes to the parameter, while the on/off switches will *increment* or *decrement* the data, usually by one number at a time.

The ground rules for working in edit mode, then, are as follows:

a) Press the edit switch to actually put the instrument into edit mode;

b) Select the general "menu" item you wish to change with the appropriate main switch (on the right side of the machine);

c) Select the specific display desired by pressing that main switch repeatedly (something you obviously won't have to do if the particular display you want happens to come up first time around - and the display that comes up is simply the last one that was selected the last time that main edit switch was accessed);

d) Use the cursor switches (on the left side of the instrument) to move the cursor to the specific parameter you wish to change; and, finally,

e) Use the data entry section (also on the left side of the instrument) to actually enter that change.

This is a procedure that is much more complicated to describe than it actually is to use - believe me. With just a little practice, I guarantee that you'll be flying around these switches with ease!

Let's run an Exercise in order to try it out. If you performed Exercise 2 with a DX7II ROM cartridge, you'll probably be happy to hear that there are yet another sixty-four voices as well as another thirty-two performance memories stored in it as well. In order to access the other bank, we'll need to put our instrument into *edit* mode for the first time.

Exercise 4
Dipping Your Toes Into Edit Mode

1) Place a ROM cartridge into your DX7II's cartridge slot.

2) Press the *Cartridge* switch so that the DX7II is accessing cartridge memory, and press the appropriate switches in order to put your instrument into any one of the four *play* modes (if you can't remember how to do this, refer back to Exercises 1 and 2).

3) Press the green *edit* switch.

4) Press main switch #15 (hereinafter referred to as *edit switch 15*) repeatedly until the LCD display reads like **figure 2-13**.

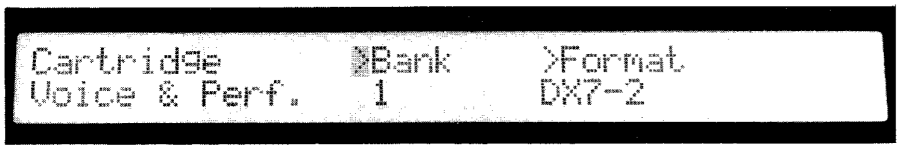


Figure 2-13

5) Press the left and/or right *cursor* switches until the cursor is blinking over the "Bank" display.

6) If the LCD shows that you are currently accessing bank 1, press the *yes* data entry switch to change it to bank 2. If it shows that you are currently accessing bank 2, press the *no* data entry switch to change it to bank 1. Experiment by using the *data entry slider* instead and observe that the same changes occur, but much more quickly.

7) Return your DX7 to the same play mode you started in (remember, the software will not let you return to a different mode) and try, in turn, each of the individual voices and/or performance memories (follow the same general procedure as outlined in Exercise 2). Note that you are now hearing sixty-four different voices and thirty-two different performance memories.

Before you get any further into this book, it will probably be a good idea to spend a little time reviewing this chapter with your DX7II close at hand in order to get thoroughly comfortable with the concept of selecting both play and edit modes and of accessing both internal and cartridge memory. You can experiment with changing any edit parameter at all - there is no way to harm the machine by doing so and of course the edit buffer ensures that you cannot really alter any sound in any kind of permanent way - but at this point most of the edit parameters may make little or no sense. When you feel comfortable with the front panel and working within the instrument's memory structure, read on, as we examine the theory and practice of digital FM synthesis as employed by the DX7II in greater detail.

Chapter Three ◇
The Operator

The heart of the DX7II is a device called an *operator*. This is a software component* which contains everything necessary to produce a sound, and it is laid out like **figure 3-1**.

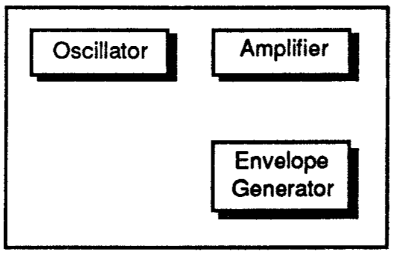


Figure 3-1

The *sound source* of the DX7II is the box in the middle, called the *oscillator*. Before we talk about what the oscillator is and does, we should first define the term "sound source". In every acoustic instrument we have learned that the sound must originate from something physically vibrating. That "something", be it a string, a reed, or a skin, is the sound source. In an analog synthesizer, the sound source would be an electronic component, called an oscillator, which generates a periodically alternating, or "oscillating" *electrical* signal. But in a digital synthesizer such as the DX7II we have learned that our sound source must be a device which regularly - periodically - generates *numbers*. And that is precisely what the oscillator inside the operator does - it is specifically a *digital oscillator*, which means that it is in fact a number generator. This is where our sound begins life.

Inside the operator are two other components which will process the numbers generated by the oscillator, and they are, respectively, the *amplifier* (specifically, a *digital* amplifier), and the *envelope generator* (again, a *digital* envelope generator). As the arrows in the diagram illustrate, the oscillator sends its signal (which are actually high-speed streams of numbers, ones and zeroes) into the amplifier, which exists for the purpose of increasing or decreasing these numbers. This will ultimately serve to increase or decrease the amount of overall signal leaving the operator, and this signal may eventually be converted by the DAC into an audible sound. If that turns out to be the case, then this digital amplifier will be performing basically the same as your stereo amplifier at home; that is, it will serve to make the sound louder or softer.

How does this work? Well, let's suppose that our oscillator is repeatedly sending the following stream of numbers to the amplifier:

0, 2, 4, 6, 8, 10, 8, 6, 4, 2, 0, -2, -4, -6, -8, -10, -8, -6, -4, -2

The amplifier may be instructed (by us, naturally) to multiply all of these numbers by a factor of ten, changing them to:

0, 20, 40, 60, 80, 100, 80, 60, 40, 20, 0, -20, -40, -60, -80, -100, -80, -60, -40, -20

or it might instead multiply them by a tenth, changing them to:

0, .2, .4, .6, .8, 1.0, .8, .6, .4, .2, 0, -.2, -.4, -.6, -.8, -1.0, -.8, -.6, -.4, -.2

* this means that the operator doesn't actually exist physically, just theoretically. The best thing you can do is forget I just said that!

In the first instance, of course, we increased the value of these numbers, and in the second instance we greatly decreased their value. Sending larger numbers to the DAC would result in a wave of higher amplitude, hence more volume. Sending smaller numbers to the DAC would result in the opposite, less volume.

The function of the amplifier, then, would seem to be that of controlling the volume of the sound we hear coming from our DX7II and this is sometimes, but not always, true. This is another mysterious statement we will be explaining shortly. In any event, remember that volume, like the other two parameters of sound, *always* changes over time and you will understand why our amplifier needs to constantly receive instructions from the envelope generator.

An envelope generator is a device found in all synthesizers. It allows the user to cause a change over time to one or more of the three parameters of sound. Again, the DX7II being a digital synthesizer, the envelope generator here is a digital device; that is, it is issuing software instructions to the amplifier. Essentially, what it is doing is telling the amplifier how much to change the numbers being fed it by the oscillator, and when to make these changes. If we could translate these commands into something resembling English, it is as if the envelope generator (or *EG* for short) is continually barking at the amplifier, "get louder! now get softer! slowly!! more slowly!! now louder again! faster!!!".

Of course there is no direct evidence that the EG has the personality and charm of a Marine Drill Instructor, but somehow it wouldn't surprise me... since it issues commands constantly and receives absolute obedience from the amplifier it is controlling.

And this is actually all there is to the operator, just these three components: oscillator, amplifier, and envelope generator. The catch is that there would be no point in having them at all if we, the human beings in charge, could not tell them what to do, and we accomplish that via what are commonly known as data inputs.

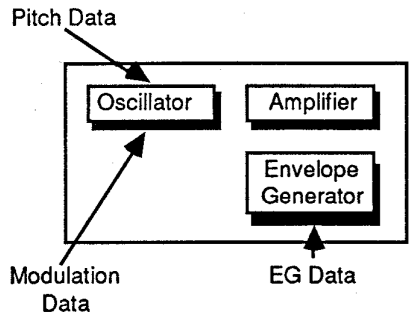


Figure 3-2

Data Inputs

There are three basic types of instructions, or commands, we need to send the operator, and they are set up in **figure 3-2.**

First of all, the EG is telling the amplifier what to do, but how does it know precisely what to tell the amplifier? It does because we give it instructions, via the *EG data input*, and it then passes these commands on. We will be devoting an entire chapter later (Chapter Nine) to the operation of the EG and how it uses the data we input to it, but for now just be aware that this input exists.

Secondly, we need to give instructions to the oscillator; specifically, to tell it what *frequency* to run at and what type of *timbre* to generate. These commands are routed via the *pitch data input* and the *modulation data input*, respectively. Both of these commands are routed directly to the oscillator. Keeping in mind that our digital oscillator is in fact a number generator, we can see that the modulation data tells it what sequence of numbers to generate (as the changes in the stream of numbers will be interpreted by the DAC into waveshape - that is, timbral, changes) and the pitch data tells it how quickly to repeatedly generate these numbers (as the speed with which the pattern is repeated will be interpreted by the DAC into waves generated per second, or frequency).

We must, at all times, send the operator both EG data and pitch data. Otherwise, our amplifier would not know how much signal to pass and our oscillator would not know how often to repeat its numbers.

However, it is not at all necessary to send modulation data to the operator. If no instructions are sent to this input then the oscillator will simply generate a predetermined stream of numbers which will have the effect of causing the DAC to generate a pure fundamental frequency, *with no overtones.*

In Chapter One we pointed out that every acoustic sound has overtones to some degree. However, our DX7II, not being an acoustic instrument, can generate a particular kind of wave, called a *sine* wave, which consists solely of fundamental frequency. The sine wave, then, is the only sound in existence which has no overtones whatsoever. If the operator receives no instructions via the modulation data input, then the only kind of wave it can generate is a sine wave.

The timbre of a sound, you will remember, is reflected in the wave *shape*, and if you look at a sine wave on an oscilloscope, it looks like **figure 3-3.**

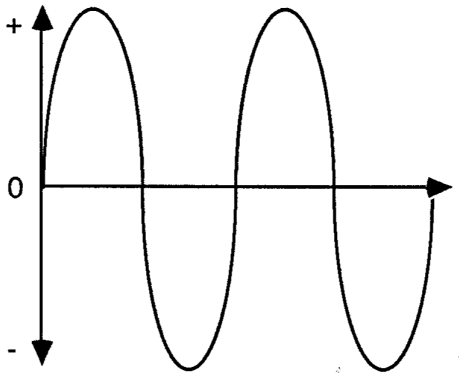


Figure 3-3

Note that the sine wave is smooth and gentle, alternating between positive and negative movements. It's easy to see how our digital oscillator can generate this wave: all it has to do is send the DAC a regular stream of numbers which start at zero, slowly get bigger, just as slowly return to zero, and then go negative the same way, i.e.:

0, 2, 4, 6, 8, 10, 8, 6, 4, 2, 0, -2, -4, -6, -8, -10, -8, -6, -4, -2

or

0, 5000, 10000, 15000, 10000, 5000, 0, -5000, -10000, -15000, -10000, -5000,

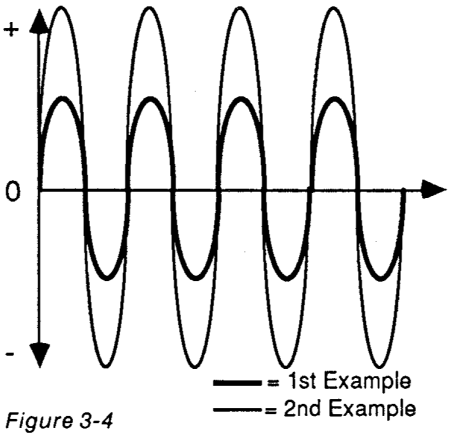


Figure 3-4

Of course, the wave generated by the second example would be much louder than the one in the first example, but they would both have the same shape, (that is, timbre), like in **figure 3-4.**

Remember again that what the DAC is doing is "graphing out" the numbers it receives!

In succeeding chapters we will talk about specific ways of entering data into these inputs - but for now, just be sure you understand what these inputs are and how the numbers entered there will affect the sound.

With this single component - the operator - the DX7II provides us with a very elegant system which allows us to completely shape and control the sound we generate. Relating back to the three parameters of sound (as we often will), we can see how this works: The EG data input allows us to control the volume, or amplitude of the sound, and to change it over time. The pitch data input allows us to determine the pitch, or frequency of the sound; and the modulation data input allows us to specify the timbre we wish to hear. It's very complete, and it's very neatly packaged.

Now comes the shocker: The DX7II (like the original DX7) actually provides us with *six* operators per voice, all of which are completely independent of one another! That means that each one of the six operators has its own oscillator, its own amplifier, and its own EG; and furthermore, that we can input completely different sets of instructions to each regarding amplitude, frequency, and timbre. Bear in mind again that the DX7II - unlike the original DX7 - allows you to generate one or *two* voices simultaneously - meaning that you can in fact have no less than *twelve* operators working together at any one time. This may seem a bit mind-boggling but it is this fact which explains the enormous range of sounds that can be coaxed from this one instrument. Other

Yamaha digital FM machines provide varying numbers of operators per voice, from the four available on the FB01, TX81Z, DX9, DX21, DX27, and DX100, to the forty-eight (!) available on the TX816 rack.

The Output Signal: Carriers and Modulators

By issuing commands to each operator through its data inputs, we have shaped a signal which will eventually contribute to the final sound we hear.

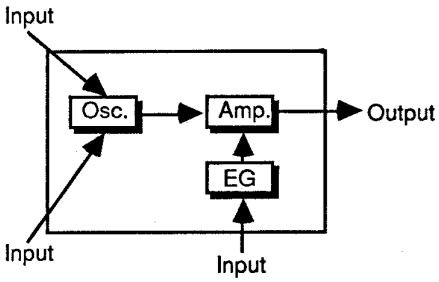


Figure 3-5

The end result of all of these manipulations is that we derive some kind of *output* signal from the operators. Be sure that you clearly understand the difference between *input* signals and *output* signals. **(see figure 3-5)**

Now that we've got some output from our operator, the real question is: What will we do with it? The answer is that we now have two options at our disposal and we will have to pick one or the other.

The first option is obvious: we can send the output signal to the DAC, in which case we will be able to hear the sound. **(see figure 3-6)**

If we decide to do this with a particular operator, then digital FM jargon dictates that we call this operator a *carrier*. A carrier, by definition, is *any operator whose output goes to the DAC*.

On the other hand, we can decide *not* to send the output to the DAC but instead directly to the modulation data input of another operator! **(see figure 3-7)**

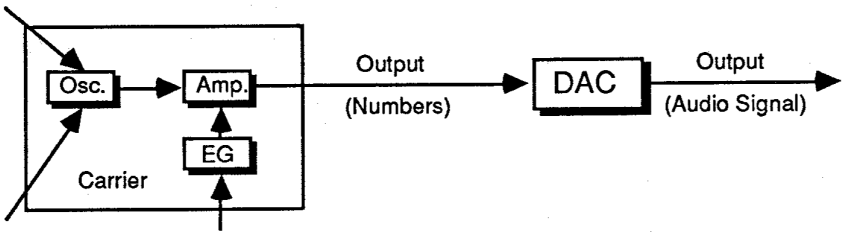


Figure 3-6

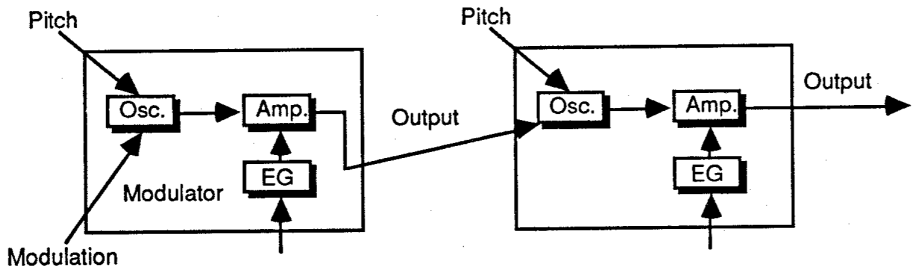


Figure 3-7

If we decide to follow this second option, then the operator in question is called a *modulator*. A modulator, by definition, is *any operator whose output goes to the modulation input of another operator.*

There is no difference whatsoever between a carrier and a modulator apart from where their output is sent!!

Why would we want to plug one operator into another like this? The fact is, this happens to be the *only* way that modulation data can be inputted to an operator. The sole reason for the modulator's existence, then, is to permit the carrier to generate complex timbres because *if no modulation data is present, the only wave that can be produced by an operator is a sine wave.* We will see shortly that while sine waves have their uses, it is easy to get sick of them rather quickly.

To summarize: *Operators can act as either carriers or modulators* (never both). *Carriers produce the actual sounds, and the timbres that they produce are determined by the actions of the modulators.* These two sentences pretty nearly summarizes the entire system of digital FM synthesis as used by the DX7II.

The only question remaining is, how do we know which operators are being used as carriers and which ones are being used as modulators? Furthermore, how do we know which modulators are plugged into which carriers? The answer to that question is the algorithms.

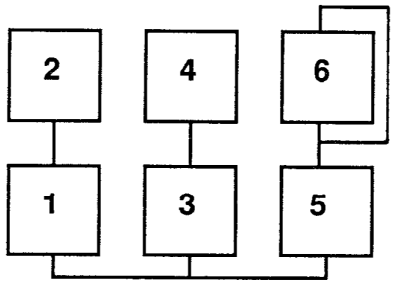


Figure 3-8

Algorithms

An algorithm is a computer term referring to a mathematical way of solving a problem. On the DX7II, the algorithm is a means of configuring operators in order to generate a particular sound.

We are provided with thirty-two different algorithms on this instrument and, as no one at Yamaha expected users to memorize each of them, diagrams illustrating these thirty-two configurations are located on the front panel of the DX7II itself, as shown in **figure 3-8.**

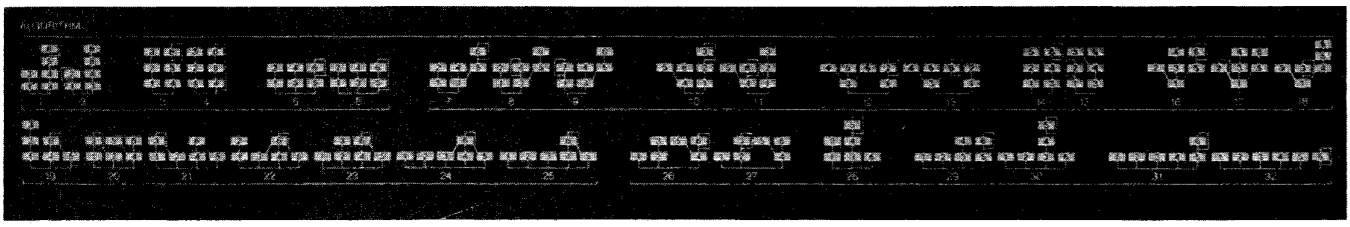


Figure 3-9

The diagrams themselves are extremely simple to understand and interpret, once you learn this basic principle: whichever operators are being used as carriers are always on the *bottom* row. Those operators being used as modulators are stacked above the carriers, and vertical or diagonal lines will always be present to indicate which modulators are plugged into which carriers. The six operators are simply numbered one through six for purposes of identification (though they are all actually identical). **(see figure 3-9)**

The above is a diagram of algorithm #5 (check the front panel of your DX7II to confirm this). As you can see, in this configuration, operators 1, 3, and 5 are being used as carriers, while operators 2, 4, and 6 are set up as modulators. Furthermore, the vertical lines show that operator 2 is modulating operator 1, 4 is modulating 3, and 6 is modulating 5.

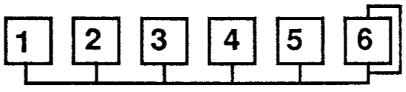


Figure 3-10

Let's take a look at another algorithm, the simplest one of all, algorithm #32. **(see figure 3-10)**

In this configuration, all six operators are on the bottom row. That tells us that all six are being used as carriers. Therefore, if we select this algorithm for a particular sound, we know that a single voice can give us up to six different pitches (simultaneously, from one key!) at six different volumes, but that all six timbres will be identical sine waves. Contrast this with algorithm #5, which we examined above. Choosing that algorithm would allow us to generate a voice with up to three different pitches (again, from one key) at three different volumes, but with three different complex timbres, since each carrier has its own modulator attached. Let's examine a few more algorithms. **(see figure 3-11)**

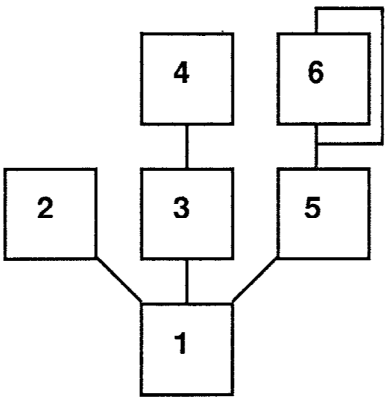


Figure 3-11

This algorithm (#16) has only one carrier - operator 1 - receiving modulation data from no less than five separate modulators, operators 2 through 6! Moreover, operator 4 is actually modulating operator 3, just

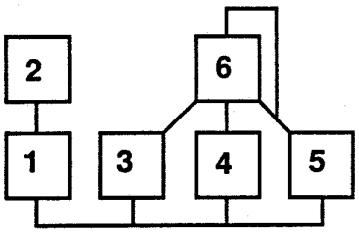


Figure 3-12

as 6 is modulating 5! The reasons and applications for this may not yet be apparent, but we will be covering these so-called "stacked" algorithms in depth later on (Chapter Seven, to be precise). Sometimes, the opposite may occur, with more than one carrier being modulated by a single modulator. **(see figure 3-12)**

This algorithm (#22) has carrier 1 being modulated by modulator 2; as well as carriers 3, 4, and 5 all being modulated by modulator 6. This means that using this configuration will allow us to generate one complex timbre of any pitch and volume (from the *system* of operators 1 and 2) and apparently three other sounds of varying pitch and volume but similar timbre from the system of operators 3, 4, 5, and 6 - since all the carriers here are receiving modulation data from the same modulator (operator 6). We'll learn later on that this isn't exactly the case, but you get the picture... The important point is that whenever you select an algorithm, you are simply telling the DX7II how you want it to configure its various operators - which ones are to be used as carriers, which as modulators, and where the modulation signals should be routed.

Probably the most common question I am asked by DX7II owners is, "how do I know which algorithm is best for a particular sound?"; or, more precisely, "which algorithm is the best for a string/brass/flute/woodwind/Star Wars sound?". The answer, unfortunately, is that there is no answer.* Most algorithms will work for most sounds most of the time. Why then do we have so many available?

What the software engineers at Yamaha have done is provided us with just about any configuration you could possibly need for specific effects. We will examine many of these effects in the pages to come. The process of selecting the "best" algorithm for a particular sound is a fairly inexact one. What you will have to do is to *eliminate* particular algorithms which appear to be unworkable for your specific application and create a short list of algorithms which look as though they might possibly work. Then it's just a matter of making an educated guess and choosing the one that "feels" like the best option. If it turns out, after tweaking your sound for a while, that you've made the wrong choice - no problem. The DX7II (like all the other programmable digital FM synths) allows you to *change* algorithms at any point! Furthermore, if you do change algorithms, the *only* thing that will be changed is the configuration - none of your data entries will be lost. As we get further into the actual programming operations of the machine we will explain further the various criteria for selecting or eliminating particular algorithms in specific situations.

Before we leave the topic of generic operators (that is, not operators specifically as carriers or modulators), we need to touch on one fairly complex - though relatively unimportant control - called *oscillator synchronization.*

Oscillator Synchronization

Although, as mentioned above, this is among the most inconsequential of all the edit parameters (accessed with edit switch 7), it is worth explaining in some detail. Why is this? Because in doing so, we will need to for the first time discuss precisely *how* the DX7II is able to play up to 16 notes at a time. This subject - referred to as *polyphony* - is an important one.

* Though there are a few general rules you can follow - see Chapter Sixteen ("Advanced Programming Techniques") for some of these.

Well, how *do* we get sixteen-note polyphony from this instrument? The obvious answer might be that the DX7II actually has 96 (16 x 6) operators. However, this is *not* the case. The mechanics of how this polyphony occurs can be explained as follows:

First of all, we know that the DX7II is a computer, and that it is constantly running thousands and thousands of computations. When we want to finally hear, say, a Middle C of a flute sound, the DX7II has to come up with a mathematical equation that will deliver just that. It can do all of its computations in advance - except for one final piece of the puzzle - and that is *which note* you wish to hear. The DX7II, after all, has no way of ascertaining that until you actually press a key down on the keyboard. The microprocessor in the machine - the central "brain" - receives this keyboard information a note at a time, even if you play chords! This is OK because remember that computers are capable of doing their thing at speeds so fast that these events appear simultaneous to we mere mortals. As each note value arrives at the microprocessor, it instantly completes the calculation and sends the resulting stream of numbers on their merry way to one of sixteen *tone generators* in the instrument. These tone generators then relay the data on to the DAC. When the next set of keyboard instructions arrive, if the first tone generator is still in use, the microprocessor simply shunts it to the next one, and so on down the line. If all sixteen tone generators are in use, the first one is reallocated for the new data (meaning that the first note we played on the keyboard is *robbed* - thus the terms "*voice robbing*", or "*last-note priority*"), and so on down the line. This entire process is called *voice assignment*:. **(see figure 3-13)**

Voice assignment in the DX7II

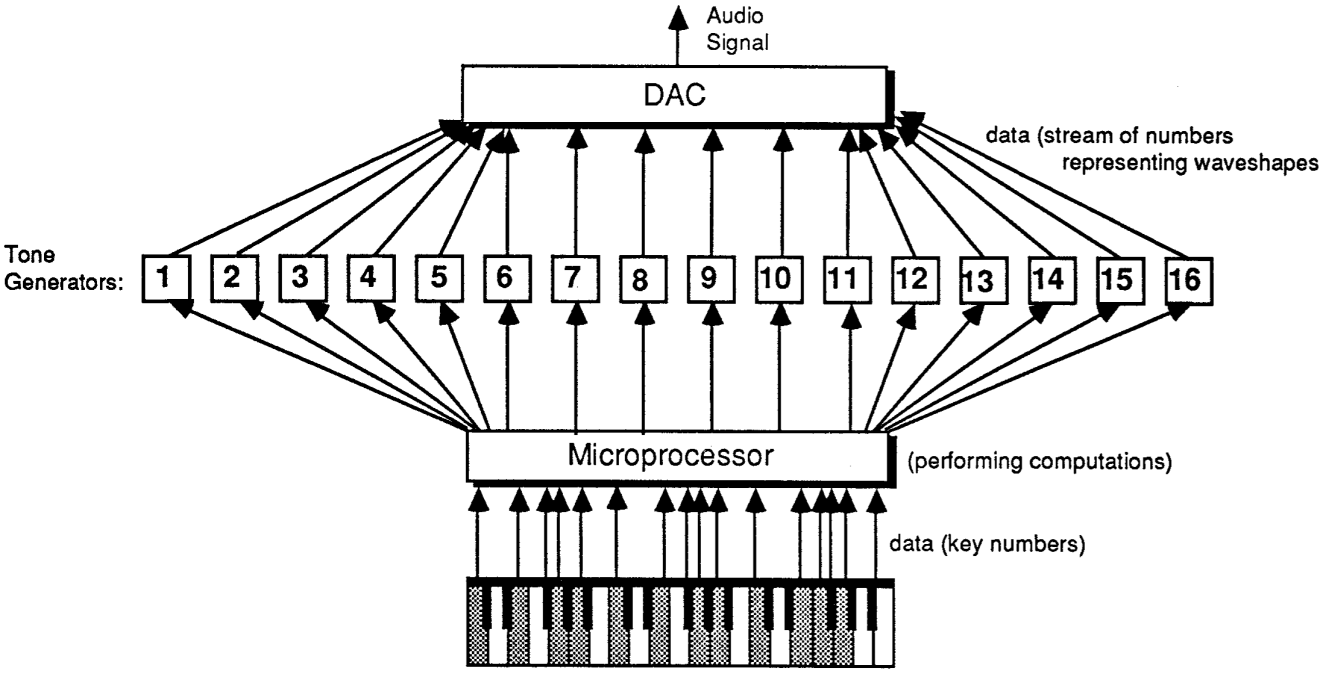


Figure 3-13

All of which brings us to point of this discussion: What is *oscillator sync*? When you press edit switch 7, the LCD display looks like **figure 3-14.**

Figure 3-14

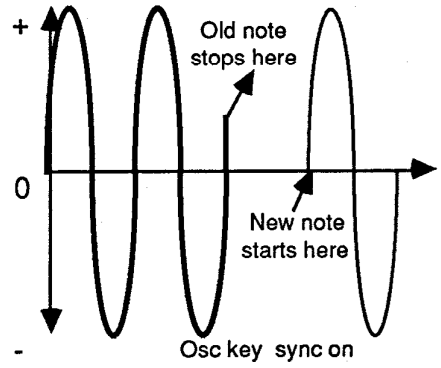


Figure 3-15

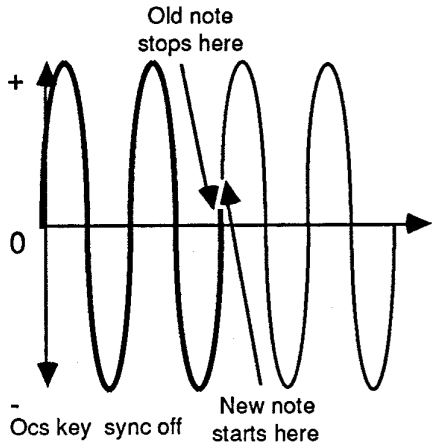


Figure 3-16

The "Osc Sync" parameter (third from left) is the one we are referring to here, and if you move the cursor over it and press the "yes"-"no" switches, you will quickly see that the only two options here are "on" or "off". The purpose of this control is to specify to the DX7II's tone generators how we would like them to output the voice data to the DAC with each new key depression. If we set the oscillator synchronization ON, then each new set of data (that is, each voice), will always begin with the number zero, causing the wave output by the DAC to always start at the *beginning* of its cycle, like in **figure 3-15.**

Alternatively, if we set the oscillator key sync OFF, then each new set of voice data will simply pick up from where it left off last time we let go of the key, like in **figure 3-16.**

What audible difference will we hear by having the Oscillator Sync on or off? This is a very subtle effect, so in most cases, none. However, you may find that having it OFF will make simulated acoustic instrument voices sound a little more natural, since each key depression will produce a wave starting at a slightly different time (meaning that the individual waves will be slightly out of *phase* with one another). Having it ON will tend to sharpen the initial attack of most sounds. In general, this effect will be more pronounced for lower pitches than high ones (since the frequencies of the waves are less, allowing you to hear phase differences more clearly) - but, for the most part, it will make little difference to the sound.

If you're interested in hearing the different effects of Oscillator Synchronization, we'll point out an example or two in the following chapters. However, in the grand scheme of things, it will usually make little impact on the sound you are programming.

Let's move on now to the technique for creating sounds from scratch on this instrument, *voice initialization.*

Chapter Four ◇
Voice Initialization

There are two basic methods of generating a sound on the DX7II. First of all, you can create a sound "from scratch" by manipulating the voice-specific *edit* parameters in various ways and sending data inputs to the six operators. Alternatively, you can work with one of the preset sounds - the ones on your ROM cartridge, or presets you get from other DX7II owners on RAM cartridges - and modify one of these sounds to meet your special needs.

There is no secret to the fact that the latter method is the one used by most DX7II owners most of the time; it's easier to start with a sound that's close to what you need and simply tweak it till you're satisfied with the result. However, the cruel fact of the matter is that it is impossible to purposefully make changes to a sound in digital FM instruments unless you have a reasonable idea of how the sound was created in the first place. Don't get me wrong - I wholeheartedly endorse the concept of modifying preexisting sounds in order to accomplish a particular result. But, as we just mentioned, you will nonetheless need to know a fair bit about how the sound was initially created. Add to that the fact that there will be times when there will be *no* preset available that will be even close to what you need. The bottom line is, no matter which method you end up using, you will still, to some degree, need to learn how to create sounds "from scratch" on this instrument (so be thankful you bought this book!).

There is a wonderfully fast and simple procedure available on the DX7II which allows us to *reset* the edit buffer (just the edit buffer! - which means that we won't actually be doing *anything* to any of the sounds already in memory) to a beginning stage - putting in a "blank page", as it were. This procedure is called *voice initialization* and it enables the DX7II user to establish a completely controllable and predictable environment in which to begin generating a sound.

Before we actually initialize our instrument, we should preview what will occur. First of all, and perhaps most importantly, whatever sound happens to currently be in the edit buffer (that is, whatever sound you are currently hearing), will disappear and be replaced by a single sine wave. Secondly, each one of the various edit parameters will immediately have a number plugged into it. This particular number (which will always be the same for that particular parameter) is called a *default* value. "Default" is simply a computer term for a condition that exists whenever a particular procedure (like voice initialization) is initiated or whenever a computer is first turned on. The DX7II does not actually revert to default values on power-up because the back-up battery prevents it from doing so and instead keeps the most recent data in the edit buffer. This is the reason why you always hear the last sound you were playing whenever you first turn on the DX7II, and why the LCD display always shows you the most recent play mode and

voice number(s) you were accessing. (If, when you powered down, you were most recently in edit mode, the DX7II will nonetheless return you to play mode - although any edit parameter changes you made - even if not stored - will still be present).

A glance at your front panel will show you that switch 14 in edit mode (which we refer to as "edit switch 14") allows us to access *utility* mode parameters concerning tuning. This switch actually would be more accurately labeled "internal memory" since it in fact deals with more than just tuning and in fact deals with several alterations to the internal memory and edit buffer. Nomenclature aside, this is the switch that will allow you to *initialize* your instrument. Put your DX7II into *single voice play* mode (if you can't remember how to do this, refer back to Chapter Two). Then go into edit mode by pressing the edit switch, and press edit switch 14. At this point you will see one of the four displays in your LCD. **(see figure 4-1)**

Figure 4-1(a)

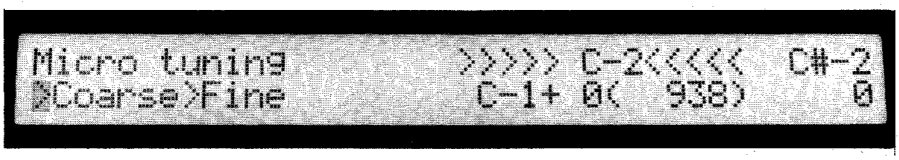


Figure 4-1(b)

Figure 4-1(c)

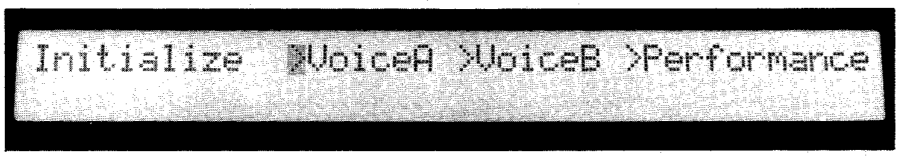


Figure 4-1(d)

Press this switch repeatedly until you see the last display - the one that says "Initialize" in the upper left-hand corner. At this point, pressing the cursor switch will move the blinking cursor over each of the three memory areas that can be initialized - voice A, voice B, or performance. We are in single mode, meaning that voice B isn't assigned to any part of the keyboard, so there's no point in initializing it. Similarly, we aren't yet dealing with any performance parameters (these will be discussed in detail in Chapter Fourteen) so we don't want to initialize the performance memory. In single play mode, the voice you are hearing over the entire keyboard is *always* voice A (the *A/B* switch won't work at all in this mode), so it is therefore voice A that we will want to initialize at this point (in *dual* or *split* mode, of course, you can initialize voice A or voice B). Remember that initializing will not change any of the sounds currently in memory in any way, shape or form! The presence of the edit buffer ensures that. With this safely in mind, let's do it:

6) Be brave: press the "yes" button again (causing the "Are you sure" question to appear again) and answer the question affirmatively, pressing the "yes" button a second time. Observe: the bottom line of the LCD immediately reads "Completed!", confirming that the DX7II has carried out your command. note also that the LED still displays the same voice number as it previously did. This number, of course, is totally irrelevant, since it is the edit buffer and not the voice itself that you have just initialized.

7) Play a few notes on the keyboard and listen. No matter what sound you had in the DX7II previously, you should now hear only a single clear tone (Audio Cue 5A). This tone is a sine wave.

8) Note that as you play different keys on the keyboard, the sine wave plays different pitches, and that you can play chords of up to sixteen notes at any one time.

It is important to realize that initializing doesn't harm anything in the DX7II at all and that you can do it as many times as you like and whenever you like. Why then did the DX7II ask "Are you sure?" before it carried out your command? This is actually a typical computer procedure. Whenever you direct a computer to undertake a severe change in course (even one that can have no permanent effect), like telling it to reset (or, in this case initialize) itself, it will normally ask for human confirmation that this is in fact what you really want it to do. Initializing the DX7II, while totally harmless, is a fairly drastic procedure. What you are in fact instructing the computer to do is change a great many things at once - *all* the edit parameters. After all, if you, for example, had the "Celeste" sound (ROM preset #62, found in bank 1, for those of you who care...) called up prior to initialization, we know that the reason we heard a celeste-like sound is because all the edit parameters had particular numbers plugged into them. Immediately after initializing, the sound was changed completely to just a single sine wave. The explanation is that ALL of its parameters were changed at once.

After we reassured the DX7II that voice initialization was in fact what we wanted it to do, it automatically plugged a particular set of default numbers into all of its edit parameters. What are these numbers? Well, most of them are zeroes: most things on the machine are simply turned off. We will be taking note of the various defaults as we encounter them in this book, or for those of you who can't wait, you can turn to Appendix B. The important thing to understand is that *every time you initialize, all of the edit parameters default exactly the same.*

Exercise 5
Voice initialization

1) If your DX7II isn't currently in *single play mode*, now's the time to do it! (Again, refer to Chapter Two if you can't remember how to get there) Call up any sound you like (remember, this procedure will not and cannot harm this sound in any permanent way).

2) Press the *edit* switch, followed by main switch 14. Press switch 14 repeatedly until the Initialization display (as shown above) appears.

3) Use the *cursor* switches to move the cursor over "Voice A".

4) Press the "yes" button. Observe: the lower line of the LCD now asks the question, "Are you sure?". Play a few notes and listen: the sound you selected is still there.

5) Press the "no" button. Observe: the "Are you sure?" question disappears as the DX7II most definitely takes you at your word. Play a few notes and listen: your sound is still there, untouched by human or digital hands.

way. This means you can initialize your DX7II today, tomorrow, or next year, and all the edit parameters will have exactly the same default numbers entered into them as they do now. This is what we mean when we talk about giving ourselves a completely controlled environment - you can predict in advance precisely which parameters will have what values because they will always reset themselves the same way. This is the real beauty of using the voice initialization procedure - you always know just what's going on in the machine at all times.

When manipulating edit parameters, probably the most important thing you need to know is which *algorithm* you are working in. We can find out just which one we are in at any time by using edit switch 7, labeled (natch!) "ALGORITHM". If you've changed anything in your instrument since the last Exercise, go back and redo it. Now, while the instrument is still in edit mode, press switch 7. Your LCD display should look like **figure 4-2.**

Figure 4-2

The number under the "Alg" in the left side of the display tells us that we are currently in algorithm #1. This is the *default* algorithm. Whenever you initialize, the DX7II puts you into algorithm #1, whether you want to be there or not.

OK, let's suppose you *don't* want to be in algorithm #1. We can change algorithms simply by positioning the cursor over the "Alg" display and using our data entry slider and/or switches (the ubiquitous "yes/no" switches).

Let's do it:

Exercise 6
Changing the algorithm

1) If your DX7II isn't still set from the end of Exercise 5 initialize it by redoing that exercise now.

2) Press main switch 7. (Since we are already in edit mode, it will be activated for its "ALGORITHM" parameters.

3) Observe: the LCD currently shows that we are in algorithm #1. Note that this is a parameter which requires neither a "yes-no" nor an "on-off" answer. Instead, we can here enter a number from 1 to 32 (since there are only 32 different algorithms). This is accomplished in the data entry section by using the "yes/no" switches, which, doubling as "+1/-1" switches, will act to increment or decrement the current number by ones; or by using the data entry slider, which will simply increase or decrease the current number quickly. Therefore,

4) Press the data entry "yes" button once. As you do so, observe that the "1" under the "Alg" changes to a "2". Now press the "no" button and observe the "2" change back to a "1".

5) Now move the data entry slider. As you do so, observe the "Alg" number change rapidly or slowly, up or down, depending on how you move the slider. If the slider was at or near its top position when you began this step, observe that as soon as you moved it to any slight degree, it immediately became activated and the LCD number "jumped" to a value corresponding to the position of the slider. This effect will be

true of ALL parameters we seek to change with the data entry slider as it will always need some small degree of movement in order to become activated.

6) Hold down a note and listen as you make these algorithm changes again (Audio Cue 6A). Note that the sound does not change pitch or timbre, but that it does occasionally change in volume, and that these changes occur in *real time;* that is, as you are moving the slider, you hear these changes occur simultaneously.

Each time you saw the "Alg" number change in the LCD, the DX7II was instantaneously reconfiguring the six operators according to the thirty-two diagrams on the machine. Note that no matter what you do, you cannot enter a number greater than 32 nor less than 1. That's because our DX7II has been instructed by the friendly folk at Yamaha that there are no more than these 32 algorithms available, and that they are numbered one through thirty-two. Again, bear in mind that changing the algorithm is something that you can do at any time - the computer really doesn't care how many times you change your mind, or when.

Why do we hear only a single sine wave when we initialize, and why does it sometimes change in volume, but not pitch or timbre, when we change algorithms? Again, the answer lies in the default settings being provided for us. Each of our six operators can in fact be switched on or off like light bulbs, at will. This means that if you are not using a particular operator at a certain time, you don't have to hear it at all; and conversely, if you want to hear a single operator or groups of operators, you can easily do so. The edit switches that are used in order to turn operators on and off with are switches 17 through 22, happily labeled "OPERATOR ON/OFF". Each of these six switches is associated with a particular operator, i.e. switch 17 with operator 1, switch 18 with operator 2, etc., and each time you depress one of the switches the operator will turn on or off, depending on whether it is already on or off *(its current status).*

This status can be readily determined when using any of the following four edit switches: 8 ("OSCILLATOR"), 9 ("EG"), 10 ("OUTPUT LEVEL"), or 11 ("SENSITIVITY"). We will, of course, be covering all of the parameters shown by each of these, but for now, let's just talk about one specific display shown in the lower left-hand side of the LCD when you press any of these four edit switches, called the *operator status display.* It consists of six "1"s or "0"s, in any combination. A "1" indicates that a particular operator is on and a "0" indicates that it is off. Therefore, a display of, for example, "111000" would mean that operators 1, 2, and 3 were on, and that operators 4, 5, and 6 were off. Put your DX7II into edit mode and press switch 8. Here's what your LCD will look like **figure 4-3.**

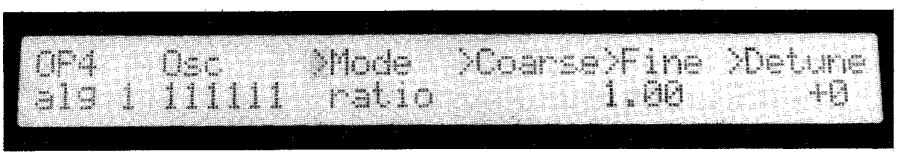


Figure 4-3

The operator status display appears immediately below the word "Osc" when using this particular switch. Edit switches 17 through 22 will now allow us to change this display, therefore changing the sound that we hear. Let's see how they work:

Exercise 7
Turning operators on and off

1) *Initialize* your DX7II from *single play mode* by using the procedure given in Exercise 6. Don't, however, change the current default of algorithm #1.

2) Press edit switch 8. Observe the *operator status display* on the bottom line of the LCD, which reads "111111", telling us that all operators are currently ON. This is the initialization default.

3) Press main switch 17. Since we are already in edit mode, this switch is now activated for its "Operator 1 ON/OFF" parameter.

4) Observe that the operator status display now reads "011111", indicating that operator 1 is now OFF.

5) Press main switch 17 again. Observe that the operator status display now reads "111111" again, showing us that operator 1 is back ON.

6) Repeat steps 3, 4, and 5 above, this time using main switches 18, 19, 20, 21, and 22 instead of switch 17. Observe that each of these switches controls an associated operator, and that *toggling* the switches (that is, pressing them repeatedly) turns them on or off.

You should get comfortable with this procedure since you will often want to hear only particular operators when you are "tweaking" a sound. But we still haven't explained why we are only hearing a single sine wave and why the volume sometimes changes as we change algorithms. The answer will become clear when we listen to what each of the six operators is contributing to the overall sound we are hearing. Let's run an Exercise and do just that:

Exercise 8
Listening to the individual operators

1) Initialize your DX7II from single play mode and press edit switch 8.

2) Turn off all six operators by pressing each of the operator on/off switches (edit switches 17 through 22) once only.

3) Observe that the operator status display now reads "000000".

4) Press edit switch 17 once to change the operator status display to "100000". This indicates that the only operator currently ON is operator 1.

5) Listen as you play a few notes. You should be hearing the same sine wave that you did in Exercise 6.

6) Now turn operator 1 OFF by pressing edit switch 17 again, and turn operator 2 ON by pressing edit switch 18 once.

7) Observe that the operator status display now reads "010000".

8) Play a few notes and listen. No sound? Don't worry, your DX7II isn't broken.

9) Repeat this procedure - turning operator 2 OFF and operator 3 ON, then turning operator 3 OFF and operator 4 ON, etc., Listening to each operator on its own in the same manner.

What conclusion can we reach? Obviously, we can only hear operator 1 for some reason. When we first initialized, all six operators were turned on, yet we now know that we were only *hearing* operator 1. Why should this be? Well, maybe the answer lies in the algorithm. We've found that the default for this parameter is algorithm #1. Aha! Maybe that's it: after all, in algorithm 1, four of our six operators are being used as modulators, and remember, you can NEVER hear modulators, since they don't send any signal to the DAC. **(see figure 4-4)**

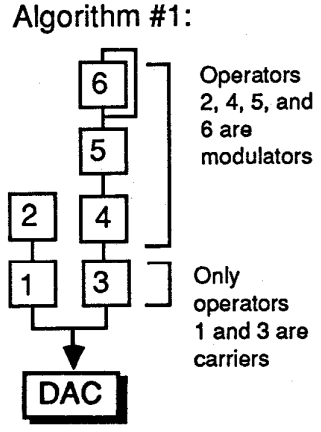


Figure 4-4

Well, let's see if our theory is correct. Repeat Exercise 9 above, but this time right after initializing, select algorithm #32 (if you don't remember how to do this, refer back to Exercise 7 earlier in this chapter). Algorithm #32 doesn't have any modulators, only carriers. **(see figure 4-5)**

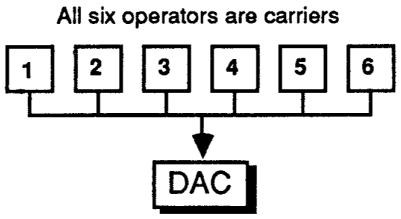


Figure 4-5

If you've just redone Exercise 9 with algorithm #32, then you already know the bad news: we were wrong. Even though we had all six operators sending their signal directly to the DAC, we still were only hearing operator 1. The real solution to this mystery lies in a parameter called *output level.*

This parameter, accessed with edit switch 10 (appropriately labeled "OUTPUT LEVEL"), allows us to independently control just how much signal leaves each of our six operators. This is the first parameter we have encountered which is *operator*-specific; that is, each of the six operators can have a totally different output level if we so desire. Whenever we call up an operator-specific parameter by using edit switches 8, 9, 10, or 11, the LCD will tell us, in the upper left-hand corner, just which operator we are viewing. Obviously, to make a change to that operator, we need to be able to view it - but, unlike the original DX7, here we can view operators whether or not they are actually ON. This is accomplished by means of another set of six switches - edit switches 1 through 6 - labeled "OPERATOR SELECT" (they also have another function - "EG COPY" - which we will get to in Chapter Nine). Like the operator on-off switches (17 through 22), each of these six *operator select* switches is linked to a particular operator, so that pressing switch 1, for example, allows us to view operator 1, while pressing switch 5 allows us to view operator 5. Like the operator on/off switches, these switches will have no effect whatsoever unless you have called up an operator-specific parameter by first pressing edit switches 8, 9, 10, or 11. In those instances, the switches are active and the LCD will always respond by showing you which operators are on and off and which operator you are now looking at.

Pressing edit switch 10 will give you one or the other of the displays shown in **figure 4-6.**

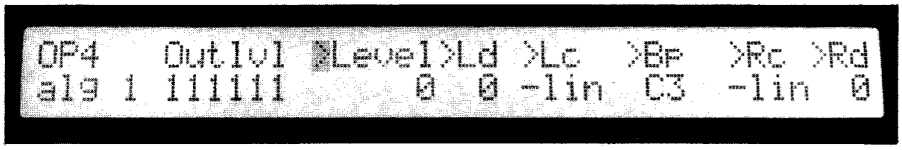


Figure 4-6(a)

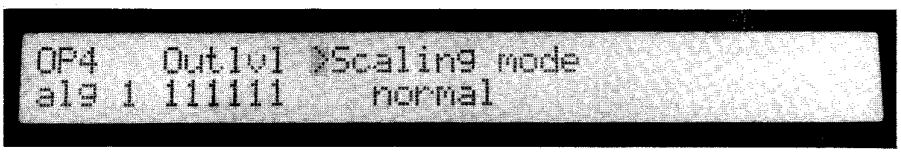


Figure 4-6(b)

The "scaling mode" display shown in **figure 4-6(b)** has to do with a control called *keyboard level scaling* that will be covered in great detail in Chapter Twelve. For now, we want to work with the general output level display shown in **figure 4-6(a).** As usual, you can cycle between the two displays by simply pressing edit switch 10 repeatedly. The parameter we want to access here is the left-most selection, labeled "Level". This is your nominal *output level* control. As we will soon learn, there are many, many factors that will display the final actual output level of a particular operator at any instant, but all such changes

will be relative to the nominal value entered here. The range of this control is 0-99, with 0 representing minimum output level (equivalent to no output at all) and 99 equivalent to maximum output for any operator, under all conditions. Let's try working with this control:

Exercise 9
The output level parameter

1) Initialize your DX7II from single play mode and select algorithm #32 by changing the algorithm default (using edit switch 7).

2) Press edit switch 10 once or twice until you see the general output level display (as shown in figure 4-6(a) above). Observe that the operator status display currently shows the default of "111111" (all operators on).

3) Turn off operators 2 through 6, using the operator on/off switches (edit switches 18 through 22). Observe that the operator status display now reads "100000".

4) Use your cursor switches to locate the blinking cursor over the "Level" (far left) parameter in the LCD.

5) Observe that the upper left-hand corner of your LCD shows you an "OP" number. This can be any of the six operators, depending on which was the most recent one looked at in edit mode.

6) Press edit switch 1 in order to view operator 1. Observe that the upper left-hand corner of the LCD now reads "OP1".

7) Observe the default value for the output level of operator 1 - it is 99. In other words, whenever we initialize our DX7II, operator 1 will automatically be given maximum output level.

8) Play a few keys and listen. Note the volume of the sine wave you are hearing.

9) Go to the data entry section, and using the "yes-no" buttons or the data entry slider, change the output level of operator 1 to a new value of 85, with the LCD showing you the change as you make it.

10) Play a few keys and listen. Note that the sine wave you are hearing is now considerably lower in volume.

11) Change the output level of operator 1 to a new value of 0. Play a few keys and note that the sound is now gone altogether.

12) Restore the output level of operator 1 back to its default value of 99.

13) Turn operators 2 through 6 back on using the operator on/off switches. Observe that the operator status display now reads "111111" again. note that the upper left-hand corner of the LCD confirms that we are still viewing operator 1. Changes made to the operator on/off status will not alter the operator select status - you'll still be viewing the same operator you were viewing before.

14) Now let's take a look at operator 2: press edit switch 2 and note that the upper left-hand corner of the LCD confirms that we are now viewing operator 2. Observe that the default value for the output level of operator 2 is 0!

15) Press edit switch 3 in order to view operator 3 and note its default output level setting of 0. Repeat this general procedure (pressing edit switches 4, 5, and 6 in turn) in order to view operators 4, 5, and 6, and note their default output level values (they will all be 0 as well).

16) Try turning various operators on and off selectively and changing their output levels, listening as you do so. Note that the output level parameter does not change in real time, meaning that you do not hear the volume change occur as you are entering the new data. Instead, you must always *retrigger* the DX7II's tone generators (as discussed in Chapter Three) by pressing the key on the keyboard again in order for that new data to be assimilated by the operator.

What have we learned from all this? The reason we were only hearing operator 1 even though all six operators were turned on, *regardless of algorithm,* is because when we initialize, operator 1 defaults to an output level of 99 (maximum volume) whereas the other five operators default to an output level of 0 (minimum volume). In other words, our clever little microprocessor has set things up so that, even though all six operators are initially on, we are only getting signal from one of them (operator 1). Having an operator on with an output level of zero is like switching your stereo amplifier on but turning the volume control to zero - it doesn't matter how good the record you have spinning is, you won't hear anything.*

OK, that answers one of the questions. What about an explanation for why the volume sometimes changes as we change algorithms? This is an easy one. Different algorithms, as you will observe from the diagrams on your machine, have different numbers of carriers. The DX7II automatically adjusts the relative volume of each carrier, according to the algorithm, so that the total volume of the signal leaving the synthesizer remains more or less constant. That means that operator 1 in algorithm #16, for example, will have twice the volume of that same operator 1 in algorithm #3; three times the volume of the same operator 1 in algorithm #5; and six times the volume than if it were in algorithm #32. **(see figure 4-7)**

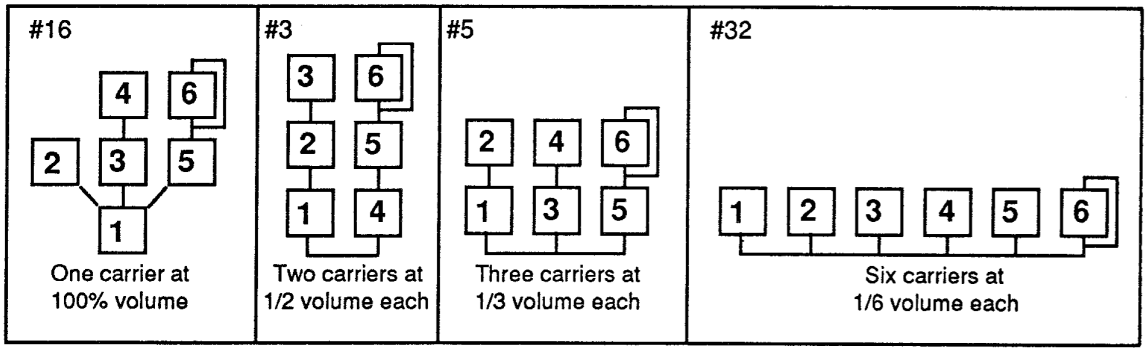


Figure 4-7

If this automatic compensation in carrier level were not performed, then a sound made with algorithm #32 could conceivably be as much as six times louder than a sound made with algorithm #16! Since in the above exercises we now know that we were only hearing operator 1 after all, this explains the changes in volume as we selected various different algorithms.

* It's worth noting that when you *store* a sound in the DX7II's memory (either internal or RAM cartridge), the operator on-off statuses are NOT memorized: all operators will automatically be turned ON - even those you had turned OFF when programming the sound. Therefore, the only way to positively ensure that a sound will have particular operators inactive is to set those operator's output levels to 0 before storing.

Remember that you won't always see an "OP" in the LCD display while editing. The absence or presence of it will tell you quite definitely whether you are working with an operator-specific parameter or not. If you do see it (by virtue of working with a parameter accessed from edit switches 8, 9, 10, or 11), the DX7II is telling you that the parameter you have selected can be individually adjusted for each operator (in other words, it is operator-specific). In that case, edit switches 1 through 6 will allow you to view the individual operators (allowing you to therefore make changes to the selected parameter if desired) and edit switches 17 through 22 will allow you to selectively turn operators on or off (allowing you to hear only those operators you want to work with). If there is no "OP" in the display, then the parameter you have selected affects ALL operators equally.

Let's continue now with a closer look at the workings of CARRIERS. Initialize your DX7II, select algorithm #32, and read on...

Chapter Five ◇
The Carrier

In this chapter we will be examining the actions of carriers alone: that is, operators sending output signals to the DAC that are not themselves receiving modulation data from modulators. As we have learned, the only kind of wave that such an operator can generate is a pure sine wave, containing no overtones whatsoever. Sine waves are used in many types of digital synthesis *additive* as well as FM) as building blocks from which to make complex sounds. While a single sine wave on its own is fairly uninteresting, *combinations* of sine waves of different frequency and amplitude can make for useful and realistic sounds. Combining sine waves together at a common output is essentially the way that additive digital systems (such as the Fairlight CMI or Synclavier) generate sounds, and, as has been mentioned, the DX7II is also capable of some limited additive capabilities. In this chapter we will examine these capabilities, and in the next chapter we will apply this information to the true DX7II sound generation system of digital FM.

We will be working exclusively with algorithm #32 in this chapter, and so although our six carriers will not yet be receiving any modulation data, we know that we will have to send them both *EG* and *pitch* data. The EG data is defaulted to a simple on-off configuration* when our instrument is initialized, so we won't be concerning ourselves with that just yet. However, the pitch data is of great importance as it will determine the frequencies of the various components of the sound we will be generating.

Most of the pitch data is entered into the operator from edit switch 8, labeled "OSCILLATOR". The parameters accessed by this switch are not actually the only data inputs to the oscillator, but they do comprise most of what determines the final pitch of the selected operator. Of course, it's fair to surmise that the keyboard also has something to do with it, but we'll soon see that that's not always the case.

INITIALIZE your DX7II and press main switch 8. The LCD display will look like **figure 5-1.**

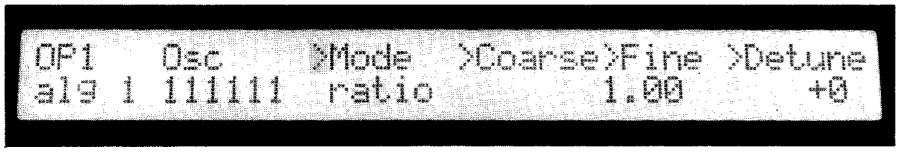


Figure 5-1

* that is, maximum volume immediately when a key is depressed and minimum volume immediately when it is released. See Chapter Nine for more information on this.

Of course, you may not be viewing operator 1 if the last operator you looked at in edit mode wasn't operator 1, since the edit buffer always shows you the most recent operator you were looking at. If your display is currently showing any operator *other* than operator 1, simply press edit switch 1 (the *operator select switch* for operator 1) in order to do so.

The leftmost parameter in this display is called "Mode". At the moment it will be reading "ratio", as this is the default mode for all operators - not just operator 1. You can confirm this by quickly pressing edit switches 2 through 6, allowing you to look at operators 2 through 6. At the end of this, press edit switch 1 again to get operator 1 back in view. Use your *cursor* switches to move the blinking cursor over the "Mode" parameter and press the "yes" switch in the data entry section. The "Mode" now becomes something called "fixed" and the other numbers in the display will change quite drastically as well. **(see figure 5-2)**

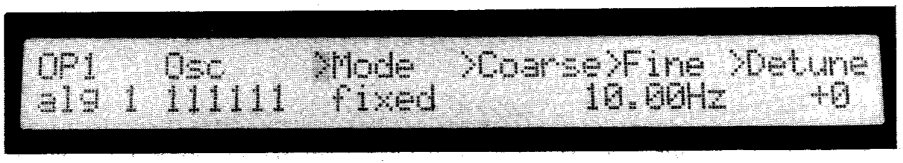


Figure 5-2

Pressing the "yes/no" buttons or moving the data entry slider up and down should confirm for you that "ratio" and "fixed" are the only two options here. What exactly do each of these mean?

"Ratio" mode (the default) means, in plain English, that the operator in question will *track* (that is, respond to) the notes we play on the keyboard. In short, as we play different notes, we will hear different pitches coming from that operator. "Fixed" mode, on the other hand, causes precisely the opposite to happen: the operator in question will *ignore* the notes we play on the keyboard and will simply output a particular fixed frequency no matter what notes we play. We'll explore "fixed" mode in greater detail shortly. For now, let's get a bit further into "ratio" mode.

In the original DX7,this was referred to as "FREQUENCY(RATIO)" mode, and in some ways it was a more accurate description of the process we are selecting. What do we mean by this? Well, obviously, our computer has no way of telling that a C# is a C# or that a G is a G on the keyboard: computers can only "think" in numbers. Instead, all of the keys of the keyboard are in fact numbered - from C1 (the lowest note) to C6 (the highest) - and each of these key numbers are associated in the computer's memory with a particular frequency, in Hz; for example, A3 (middle A) with 440. Whenever you press middle A, then, the computer quickly looks up a little table in its memory and generates the number 440. This number is called the *frequency number*. The next thing the DX7II will do is look up another number, called the *ratio number*, and it will automatically *multiply* the two. The end result of this calculation is the final frequency that we hear coming from that operator. Therefore, the term "FREQUENCY(RATIO)" should be interpreted in the *algebraic* sense, with the parentheses indicating the multiplication function (remember, in high school algebra, when you wanted to multiply 2 by 2, you would write it out 2(2). Hopefully, you came up with 4 as the answer. If not, please return this book immediately for a refund!). The name for this mode has been shortened in the DX7II to just "ratio" mode, but it is precisely the same as the "FREQUENCY(RATIO)" that many of you used to see in the original DX7.

Where does this *ratio number* come from? From us, natch. This is the number displayed in the "Coarse/Fine" parameter immediately to the right of the "Mode" parameter. Next to that display is another one labeled "Detune". This is simply a fine-tuning control which will enable you to make very slight changes to the ratio number. A quick scan through the six operators will reveal what the default value are for these parameters. As we will see, when you initialize, all six operators are given the same default ratio numbers and detuning values. Let's do it:

Exercise 10
The default pitch data values

1) INITIALIZE your DX7II from single voice play mode and select algorithm #32. (if you're not sure how to do this, refer to Exercises 6 and 7 in the preceding chapter).

2) Press switch 8 ("Oscillator"). If you are currently viewing any operator other than operator 1 (which the upper left-hand corner of the LCD will instantly tell you), then press switch 1 in order to view operator 1.

3) Observe that the default *mode* for this operator is "ratio", that the default *ratio number* is 1.00, and that the default *detuning value* for this operator is 0.

4) Press switch 2 in order to view operator 2. Observe that operator 2 has precisely the same default values for these parameters as operator 1.

5) Press switches 3 through 6 in order to view operators 3 through 6. Again, all six operators default to precisely the same values for these parameters.

Exercise 10 has shown us that the ratio number defaults to exactly 1.00, for every operator. This means that, no matter which operator we are listening to, if we play a middle "A", our DX7II will look up the frequency number "440" and multiply it by the ratio number 1.00, resulting in the output of a sine wave of exactly 440 Hz (440 x 1.00 = 440). Of course, changing the ratio number will result in a different frequency altogether being generated when middle "A" is played. The "Coarse" parameter allows us to change the ratio number by whole numbers. Multiplying a particular fundamental frequency by whole numbers, remember, gives us the *harmonic overtones* of that frequency. Therefore, the "Coarse" control will allow us to step through the *harmonic series* for any particular note:

Exercise 11
The Frequency Coarse parameter

1) INITIALIZE your DX7II from single play mode and select algorithm #32. Using edit switches 18 through 22, TURN OFF all operators except operator 1. The *operator status display* should now read "100000".

2) Press switch 8 ("OSCILLATOR") and, if you are not currently viewing operator 1, press switch 1 in order to do so. 3) Observe that the ratio number is currently at its default value of 1.00. Play a short, recognizable sequence of notes and listen (Audio Cue 11A).

4) Use the cursor switch to move the blinking cursor over the "Coarse" parameter and press the "yes" button in the data entry section (which will act to increment this value in steps of whole numbers) in order to CHANGE the ratio number to 2.00.

5) Play the same series of notes on the keyboard and listen. Your melody has now been raised an octave! (Audio Cue 11B) Since the ratio number has now been doubled, all the notes have gone to their second harmonic, which is an octave higher. For example, playing middle "A" on the keyboard now results in an output of 880 Hz (440 x 2.00 = 880).

6) Continue playing the same sequence of notes, and as you do so, CHANGE the value of the ratio number to 3.00. Listen and note that the notes have now been transposed up another musical fifth (to the third harmonic). (Audio Cue 11C) Note that this change occurred in real time, as you were playing the notes. Pitch data changes, unlike output level changes, will occur in real time.

7) Hold down a single key on the keyboard and use the "yes" button in order to slowly CHANGE the ratio number upward till you reach the maximum value of 31.00. Listen as you do so (Audio Cue 11D), and note that all of the resulting transpositions are musically related to the original pitch; after all, we are stepping through harmonic overtones. At the maximum value of 31.00, the DX7II is actually multiplying the notes you play by 31, and so if you are holding down a high note, it may well be supersonic at this point (see if your dog starts acting weird!). Note that using the data entry slider will initiate the same changes as the "yes" button, but more rapidly.

8) Now reverse course and use the "no" button in order to slowly return the ratio number back to 1.00, listening as you do so (Audio Cue 11E). note that below 1.00, you can lower the ratio number to its minimum of 0.50, thereby lowering your note to an octave below where you started (remember, halving the frequency drops it an octave). Note once again that moving the data entry slider will change these values quickly, and also in real time, as before.

Let's stop for a second and think about what we actually did in that last exercise. We started with a single sine wave from operator 1, and we altered its pitch by stepping through the harmonic series. What if we were to add in a second operator, tuned to a different harmonic? Would we then hear a two-note chord? The answer, surprisingly, is no. When the human brain hears two sounds of different pitch but identical *timbre* - as two sine waves undoubtedly are - we perceive it as a single, *complex* sound. In other words, the lower of the two frequencies would sound to us like a fundamental frequency, and the higher one like an *overtone* (for those of you who may be a little lost here, refer back to the "Timbre" section in Chapter One for a quick refresher). The same will happen if we add in a *third* sine wave from a third operator, or a fourth, fifth, or sixth. In short, this process allows us to build up complex timbres via *additive* synthesis. Let's run an exercise to try it out:

Exercise 12
Creating a sound using additive synthesis

1) INITIALIZE your DX7II from single play mode and select algorithm #32. TURN OFF operators 2 through 6 (operator status = "100000"). If you are not already viewing operator 1, press switch 1 in order to do so.

2) Press edit switch 8 (OSCILLATOR) and observe that the ratio number is at its default value of 1.00. Play a few notes on the keyboard and listen to confirm that operator 1 has output level, and press edit switch 10 (OUTPUT LEVEL) to visually confirm that fact.

3) Press edit switch 17 in order to TURN OFF operator 1 and press edit switch 18 in order to TURN ON operator 2 only (operator status now = "010000"). Press edit switch 2 in order to VIEW operator 2. Play a few notes on the keyboard. You should not be hearing anything! Press switch 10 and observe that the explanation lies in the output level of operator 2, which has defaulted to 0. Use the cursor switch to position the cursor over the "Level" parameter and use the data entry section in order to CHANGE the output level of operator 2 to 99. Play a few notes and listen - you should now hear a sound.

4) Press edit switch 8 again and position the cursor over the "Coarse" parameter. Use the data entry section to CHANGE the ratio number for operator 2 to 2.00. Listen and note that the operator is now an octave higher (at its second harmonic).

5) TURN ON operator 1 again ("110000"). Play a few notes and listen. You should be hearing two sine waves playing together *in octaves* (Audio Cue 12A). The fact that we are hearing both sine waves mixed together at the same output gives the result of a single complex sound. This is the way that additive synthesis works.

6) TURN OFF operators 1 and 2 and TURN ON operator 3. The operator status display should now read "001000". VIEW operator 3 and CHANGE its output level from its default of 0 to a new value of 99. CHANGE the ratio number for operator 3 to 3.00. Play a few notes and listen. The operator is now playing every note at its third harmonic - an octave and a fifth higher!

7) Selectively TURNING ON and OFF operators 1, 2, and 3, listen to the following combinations: operators 1 and 3 (Audio Cue 12B); operators 2 and 3 (Audio Cue 12C); and, finally, all three operators together (Audio Cue 12D). Starting to sound familiar? The Hammond organ creates its sound by combining together sine waves of various harmonic frequencies. Our DX7II is obviously able to simulate this method digitally.

8) TURN OFF operators 1, 2, and 3 and TURN ON operator 4 ("000100"). VIEW it, set its output level to 99, and give it an ratio number of 4.00 (fourth harmonic = two octaves higher). Play a few notes on the keyboard to confirm this.

9) Using the same procedures, TURN ON and VIEW operators 5 and 6 one at a time, setting both to an output level of 99 and setting the ratio number for operator 5 to 5.00 (the fifth harmonic) and operator 6 to 6.00 (the sixth harmonic). Listen to each operator independently as you do so.

10) Finally, TURN ON all six operators and listen to the composite result (Audio Cue 12E). 11) Experiment by changing the ratio numbers for various operators to different harmonics and by altering the output levels of different operators in order to achieve different blends. Note that often even small changes to either of these parameters will induce large changes in the overall sound.

Congratulations!! We've just created our first musically useful sound from scratch. Doing organ sounds on the DX7II is particularly easy because these sounds are usually derived from simple sine waves. The trick is in combining several sine waves of different pitch and volume in order to create an interesting composite whole. In doing so, we have stumbled on one of the most important CARDINAL RULES of programming the DX7II:

***** CARDINAL RULE 1: CHANGING THE OUTPUT LEVEL OF A *CARRIER* WILL ALWAYS RESULT IN A CHANGE IN *VOLUME*.**

Throughout the course of this book, we will probably be repeating this so many times that you will be sick of hearing it! But this Cardinal Rule - and just a few others - is really the key to understanding how to program the DX7II. Remember, we warned you way 'back in Chapter One that *everything* we do in programming synthesizers relates back to the three parameters of sound - and we've just learned how to control one of them.

The output level control of a carrier, then, can be thought of as a *mixer*. This is the way that we can control the volume of various parts of our sound - *if* the operators in question are carriers (we'll see in the next chapter that modulators don't respond the same way).

Okay, back to the various pitch data input controls. We've seen that the "Coarse" control allows us to change the ratio number in whole number increments. But what if that's not what we need? Suppose, for example, that we wanted a ratio number of 1.25? Or 9.78?? Or any other *non-whole-number* value? We can accomplish this with the "Fine" parameter in this same display. This will always allow you to as much as nearly double whatever ratio number you currently have set, in fractions of whole numbers. Its range is determined by the value of the "Coarse" control. For example, if you've used the "Coarse" control to set a ratio number of 1.00, then the range of the "Fine" control will be 1.00 to 1.99, in increments of 0.01. On the other hand, if the ratio number has been set by the "Coarse" control as being 2.00, then the range of the "Fine" control is 2.00 to 2.98, in increments of 0.02. A ratio number of 3.00 will yield a "Fine" range of 3.00 to 5.97 in increments of 0.03, and so on. Try it!

Exercise 13
Using the Frequency Fine control

1) INITIALIZE your DX7II from single voice play mode and set it to algorithm #32. Press edit switch 8 (OSCILLATOR) and use edit switches 18 through 22 to TURN OFF operators 2 through 6 ("100000"). Play a few notes and listen. You should be hearing a single sine wave, coming from operator 1 only (Audio Cue 13A).

2) If you are not already viewing operator 1, press edit switch 1 in order to do so. Use the cursor switches to position the blinking cursor over the "Fine" parameter and, using the "yes/no" switches or data entry slider, CHANGE the "Fine" value to 1.50.

3) Play a few notes on the keyboard and listen. The operator is now playing a musical fifth higher than it was previously.* (Audio Cue 13B)

4) Change the "Fine" value for operator 1 to 1.99. Play a few notes and listen. The operator is now playing nearly (but not quite) an octave higher than it was originally. (Audio Cue 13C)

5) Restore the "Fine" value to 1.00 and press your left cursor switch once in order to position the cursor over the "Coarse" parameter. Change this value to 2.00. Play a few notes and confirm that it is now a perfect full octave higher than it was originally. (Audio Cue 13D)

6) Press the right cursor switch once in order to reposition the cursor over the "Fine" parameter and use the data entry slider to move the "Fine" values through their full range - 2.00 to 3.98. Observe that these changes are made in increments of 0.02. As you are making these data changes, play a few notes on the keyboard and listen (Audio Cue 13E). Note that these changes in pitch occur in real time.

* For those of you who aren't sure why this should be, here's the explanation: The third harmonic (a ratio number of 3.00) is an octave and a fifth higher than the fundamental. Halving this (in other words, to a ratio number of 1.50) will drop it back a full octave, yielding a frequency merely a musical fifth higher.

7) Experiment by setting the "Coarse" parameter for operator 1 to various other values and then changing the "Fine" parameter. Try doing the same with the other operators and listen to the composite result of setting more than one operator to various non-whole ratio numbers - the results will probably be more than a little cacophonous since we are blending *inharmonics* rather than *harmonics*. Bear in mind that the entire purpose of the Frequency Fine parameter is to allow you to set an operator to an inharmonic frequency.

One byproduct of the Frequency Fine control is that by using it at all, you will be changing the various Frequency Coarse values available to you - they will no longer just be whole number values. Try changing one of the operators to a Fine value of 1.63, for example. Now move the cursor back to the Coarse control, and, using the "yes" button, step through the various increments available. You'll be surprised to see that the instrument does not offer you the values of 2.63, 3.63, or 4.63; but instead gives you the strange options of 3.26, 4.89, and 6.52. These values aren't quite so mysterious when you realize that 1.63 multiplied by 2.00 equals 3.26; that 1.63 multiplied by 3.00 equals 4.89; and that 1.63 multiplied by 4.00 equals 6.52. In any event, if it appears at any time that the Coarse control is not giving you whole number values, simply go to the Fine control and move the data entry slider down to its lowest position. That will have the effect of dropping the Fine value to its minimum - making it exactly equal to the Coarse value. At this point, the Coarse value will operate normally - calling up whole-number increments only.

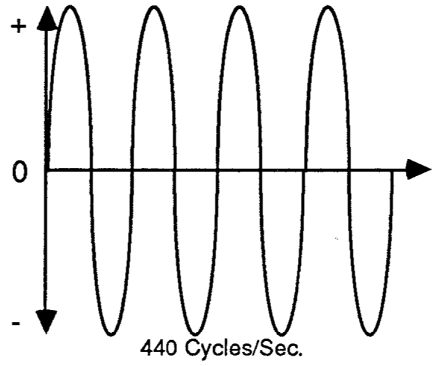


Figure 5-3

The Detune Control

In order to understand the workings of this control, we need to discuss the acoustic phenomenon of *beating*. Whenever we hear two sounds simultaneously of nearly, but not exactly, the same frequency, we will encounter this effect. Let' suppose, for example, that we play A440 on a piano that's just been tuned. On an oscilloscope, the wave might look something like **figure 5-3.**

Now let's wheel another piano into the same room, but this one hasn't been tuned in quite a while, so *it's* Middle A is actually at 439 Hz. On the oscilloscope, the wave would look like **figure 5-4.**

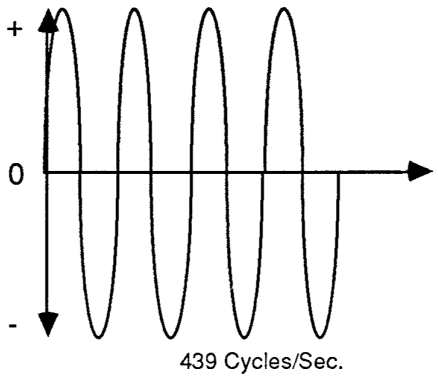


Figure 5-4

Let's get the two pianos next to each other, stretch our arms out, and play Middle A on both pianos simultaneously. The oscilloscope would actually show a composite wave, but if we use our imagination we can theorize that the two waves together look something like **figure 5-5.**

As you can see, there are areas where the two waves cross each other and there are many more areas where they diverge. Because in our example the two waves differed in frequency by exactly one Hz, the amount of time between each cross-over point will be exactly one second. Each time they cross, the sound will increase in volume because the two waves reinforce one another. Each time they diverge, the opposite will happen. In short, we will hear a sound that periodically increases and decreases volume. This is what we call *beating*. The *difference*, in Hz, between our two frequencies dictates how often the beating occurs. It stands to reason, therefore, that the closer together our frequencies are, the less frequently they beat. The further apart they are, the faster the beating (until, at a certain point, the beating disappears altogether and we simply perceive two different pitches - remember, this is a phenomenon that occurs *only* if the two sounds in

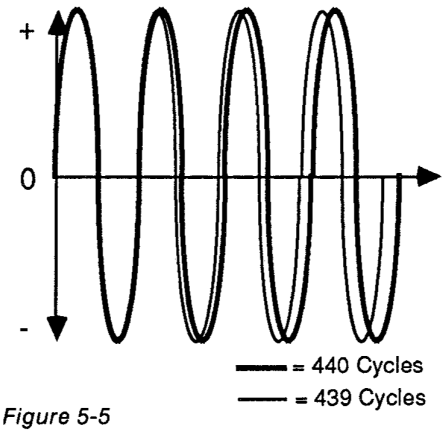


Figure 5-5

question are very close to one another in frequency). For example, an A440 together with an A439 will beat once per second, whereas an A440 together with an A438 will beat twice per second.

This is a technique commonly used by guitarists and bass players everywhere in order to tune their instruments. It involves playing the same note - usually as a harmonic - on two different strings. The tuning of one of the strings is then changed while listening closely to the beating until it slows down and finally disappears. At that point, the two strings are deemed to be perfectly in tune.

Beating is also used to great effect by analog synthesizers. In fact, this is the main explanation for the rich, lush sounds that these instruments are known for. These instruments typically employ two or more analog oscillators as their sound source, and, as we know, it is impossible to expect two voltages to ever remain stable enough long enough to avoid drifting. If you tune two analog oscillators to the "same" frequency, what you are actually doing is tuning them to *nearly* the same frequency. Even then, the differences in frequency between them will constantly be changing. Without the use of some kind of electrical synchronization circuit (which is commonly available but infrequently used), there is NO WAY to get two analog oscillators perfectly in tune with one another. The end result is that the analog synthesizer is usually outputting signals that are beating in varying interesting manners, and this is what is largely responsible for their "warmth".

Acoustic instruments also beat most of the time. In the case of a piano, that's pretty obvious since each note has two or three strings, and the best piano tuner in the world cannot (and won't even want to) get these strings absolutely perfectly in tune with one another. In other acoustic instruments, there are other physical explanations for this phenomenon. The point is that beating is an integral part of most musical sounds in existence. Obviously, we want to be able to induce this in our digital FM synth. And here's where a potential problem arises. Unlike its analog cousins, the DX7II has *digital* oscillators. This means that there is theoretically no potential whatever for distortions or drifting in the signal they generate. We will instead have to program these distortions in, and, even then, they will be "perfect" distortions. This effectively means that it is impossible for a digital synthesizer of any kind to exactly reproduce the unpredictable changes in sound that an analog synthesizer generates. In return for this restriction, however, we know that we have much finer control over what we are actually doing. The good news is that there *is* a method on the DX7II for inducing beating effects - and it is a much more precise method than any found on analog machines.

We've learned that in order to make a sound beat, we need to generate two very similar frequencies. Well, one way we could do that on the DX7II would be to use the Frequency Fine control. This will enable us to generate two sine waves that are slightly out of tune with one another. If, for example, operator 1 is given a ratio number of 1.00 and operator 2 a value of 1.01, then playing Middle A on the keyboard will result in two sine waves, one of 440 Hz (440 x 1.00 = 440) and the other of 444.4 Hz (440 x 1.01 = 444.4), resulting in a beating occurring once per 4.4 seconds (444.4 - 440 = 4.4). Let's try an exercise that first allows us to confirm that no beating occurs in the initialized state, and then allows us to set up the example above and hear the result:

Exercise 14
Using the Frequency Fine control for beating effects

1) INITIALIZE your DX7II from single voice play mode and select algorithm #32. Press edit switch 8 (OSCILLATOR) and use edit switches 18 through 22 to TURN OFF operators 2 through 6 (operator status = "100000"). Play a few notes and listen, confirming that you are hearing operator 1.

2) If you are not already viewing operator 1, press edit switch 1 in order to do so. Observe that operator 1 is at its default pitch data values: "Mode" = Ratio, and a ratio number of 1.00 with a detuning value of 0.

3) TURN OFF operator 1 (by pressing edit switch 17) and TURN ON operator 2 (by pressing edit switch 18). Observe the operator status display, now reading "010000". Press edit switch 2 in order to VIEW operator 2.

4) Press edit switch 10 (OUTPUT) and CHANGE operator 2's output level from the default of 0 to a value of 99.

5) Press edit switch 8 and observe that operator 2 is currently at the same default values as operator 1. Play a few notes and listen. There should be no difference whatsoever in the sound between operators 1 and 2.

6) Press edit switch 1 in order to TURN ON operator 1 again ("110000") and play a few notes, listening to the composite result of operators 1 and 2 together (Audio Cue 14A). Note that, apart from an increase in volume, there is no perceptible difference between the sound of both operators together and the sound of either alone. The overall volume has increased because the two operators are *exactly* in tune with one another and so their waves are reinforcing at every point. (see **figure 5-6**)

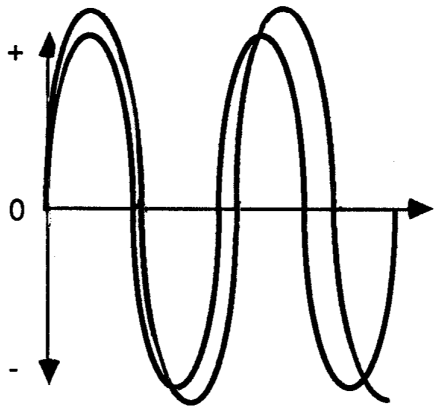


Figure 5-6

7) While still viewing operator 2, use the cursor switch to position the cursor over the "Fine" parameter and press the "yes" button in the data entry section once in order to change the ratio number to a new value of 1.01.

8) Hold down Middle A on the keyboard and listen (Audio Cue 14B). You should be hearing a composite sound of two sine waves *beating* against one another 4.4 times per second.

9) Without making any further data changes, press and hold down "A" above Middle A and listen (Audio Cue 14C). You should now be hearing a composite sound of two sine waves beating against one another *8.8* times per second!* Experiment by playing and holding down some more notes, individually, and in chords, and note that, the higher the note you play, the faster the beating, and vice-versa. Also note that each octave higher produces beating twice as fast, and that each octave lower yields beating half as fast.

10) Press the "yes" switch once more in order to CHANGE the Fine value for operator 2 to 1.02. Hold down a note and observe that the beating is occurring more quickly. Keeping the note held down, continue incrementing the Fine value for operator 2 until the beating disappears and you hear two very sour notes! Reverse course and slowly decrease the Fine value for operator 2 and note that the beating slows down and finally disappears altogether as we once again hear the composite result of two sine waves perfectly in tune with one another.

* since 880 times 1.01 = 888 while 880 times 1.00 = 880.

Effects of Detune Control:

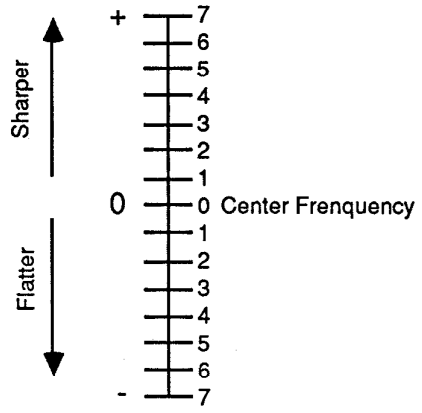


Figure 5-7

So far, so good. But the problem is, the Frequency Fine control will allow us to get our two operators only so close to one another in pitch. If we could somehow get them closer still, we know that we would hear our beating occurring more slowly, and this will often produce a very desirable warmth to our sound. Fear not, because the *Detune* control lets us do just that - it is simply a *finer* fine tuning control!

What the Detune control does is to break down the smallest Frequency Fine increment (which we have seen changes according to the Frequency Coarse value) into fifteen further increments. These are arranged as values of -7 to +7, with 0 as the center point. **(see figure 5-7)**

Using the Detune control in order to detune an operator to a negative value will ever so slightly flatten its note whereas changing it to a positive value will ever so slightly sharpen it. This is the reason why 0 has been provided as a center point - if you don't want a particular operator to be detuned at all, you just set this value to 0. The best way to understand the real meaning of the Detune increments is to imagine, for a moment, that there are also Detune values of "+8" and "-8". If your Coarse value were 1.00, then "+8" would be equal to 1.01 (the next Fine value up), while "-8" would be equal to 0.99 (the next Fine value down). If your Coarse value were 2.00, then "+8" would be 2.02 and "-8" would be 1.98, etc. Of course, there are no "+8" or "-8" Detune values, since you can already get to those numbers with the Fine control.

As you might expect, the initialized default for this parameter is 0 for all operators. Let's try using it:

Exercise 15
Using the Detune control

1) INITIALIZE your DX7II from single voice play mode and select algorithm #32. Press edit switch 8 and then TURN OFF operators 3 through 6 with edit switches 19 through 22 (operator status = "110000").

2) If you are not already doing so, press edit switch 1 in order to VIEW operator 1. Observe the LCD in order to confirm that the default detuning value of 0 is currently set. Now press edit switch 17 in order to TURN OFF operator 1 ("010000").

3) Press edit switch 2 in order to VIEW operator 2 and then press edit switch 10. CHANGE the output level of operator 2 from its default of 0 to a new value of 99, enabling us to hear it.

4) Press edit switch 8 and use the cursor switches in order to position the cursor over the "Detune" parameter. CHANGE the DETUNE value for operator 2 to +7. This will have the effect of making the frequency that operator 2 outputs slightly more sharp than normal - but not quite as much as a Frequency Fine setting of 1.01 would do.

5) Play a few notes and listen, first to operator 2 alone, and then, using the operator on/off switches, to operator 1 on its own (Audio Cue 15A). You may not be able to tell any discernable difference between them since the change in pitch is very slight.

6) TURN ON both operators 1 and 2 ("110000") and hold down a note on the keyboard. Listen! (Audio Cue 15B) You should be hearing the composite sound of two sine waves gently and slowly beating against one another.

7) Holding down the same note, use the "no" button in the data entry section to slowly decrease the Detune value for operator 2 back to 0. Listen (Audio Cue 15C). Note that as you approach 0 the beating slows down and finally disappears altogether at the value of 0. Note also that this parameter changes in real time.

8) Keeping the same note held down, continue using the "no" button to decrease the Detune value for operator 2 into its negative numbers until you reach -7. Listen (Audio Cue 15D) and note that the audible result is the *same* as in step 7 above. Whether we sharpen or flatten one of the frequencies doesn't matter, so long as the *difference* between the two remains constant (i.e. 441 minus 440 equals 1, just as 440 minus 439 *also* equals 1).

9) Experiment by playing chords as well as single notes. Remember that each note is beating at a different rate! (the higher the note, the faster the beating) and so we derive strikingly beautiful effects (Audio Cue 15E). Experiment further by adding in various other operators at the same Coarse and Fine values but at different Detune values and note that very complex and elegant movements in the sound can be obtained in this manner.

Of course, the beatings we can generate using this control will be "perfect" and unchanging effects, but with up to six different operators at our disposal, we can mask the regularity of this movement sufficiently to emulate the unpredictability of the analog synthesizer or the acoustic instrument. Beating is by far one of the most important effects we can use to "humanize" the sounds we create with the DX7II. In general, it's something you do towards the end of a programming session, but it's always worth trying and usually will add a warmth to the sound which is desirable and difficult to achieve otherwise. Unlike the analog synthesizer, however, we aren't forced to have it present if we don't want it - just set the Detune control to 0 and (apart from special circumstances to be revealed later) there will be no beating at all coming from your DX7II.

Fixed Frequency Mode

Earlier on, we mentioned that an alternative existed for the "ratio" mode. In this "ratio" mode, the pitch we hear changes as we play different notes on the keyboard because, even though the ratio number remains constant, the frequency number is continually being updated and revised (by the action of us playing the different notes!). The alternative we have to this situation is to use "fixed" mode, and any operators put into this mode will no longer generate ratio numbers, nor will the computer do its multiplications. Instead, we will use the Coarse, Fine, and Detune controls to designate a particular fixed frequency at which the oscillator in question will generate its signal. This means, in plain English, that no matter what note we play on the keyboard, we will hear the same fixed pitch. There are many potential uses for this effect, but for those of you who are heavily into bagpipes, the first and most obvious is in setting up drones (sometimes called pedal tones).

Before we run an exercise showing you how to do this, let's preview the mechanics of how we set the fixed frequency we require. First, of course, we need to change the operator's mode to "fixed" (accomplished with the "Mode" parameter in the OSCILLATOR display accessed from edit switch 8). The Coarse control in this display will then, as its name implies, set the coarse range, either from 1 Hz to 10 Hz; 10 Hz to 100 Hz; 100 Hz to 1000 Hz; or 1000 Hz to 10,000 Hz. This is accomplished by giving us four set values of 1.000 Hz; 10.00 Hz; 100.0 Hz; and 1000 Hz. As you can see, when you cycle between these values (by pressing the "yes/no" buttons), we are simply shifting the decimal point.

Having set the Coarse range, we can then use the Fine control to zero in on the particular frequency desired, and the Detune control if we wish to make further slight alterations to the pitch.

Exercise 16
Using Fixed Frequency mode

1) INITIALIZE your DX7II from single voice play mode and select algorithm #32. Press edit switch 8 and use edit switches 18 through 22 in order to TURN OFF operators 2 through 6 (operator status display = "100000").

2) If you are not already viewing operator 1, press edit switch 1 in order to do so. Use the cursor switch in order to position the blinking cursor over the "Mode" parameter and press the "yes" button in the data entry section in order to change the MODE for operator 1 to "fixed".

3) Observe that the coarse range is currently 10 Hz - 100 Hz since the LCD shows a fixed frequency value of exactly 10.00 Hz.

4) Play a few notes on the keyboard and listen. You should not be hearing any sound. Why is this? Because 10.00 Hz is a *subsonic* frequency, below the range of human hearing (see Chapter One if this doesn't sound familiar).

5) Position the cursor over the "Coarse" parameter and press the "yes" button in the data entry section once to change this value to 100.0 Hz. The range is now 100 Hz - 1000 Hz and the current fixed frequency is exactly 100 Hz, now in the *audible* range.

6) Play a few notes on the keyboard and listen (Audio Cue 16A). Note that you hear the same low tone, approximately A flat two octaves below Middle A, *no matter which key you press.*

7) Press the "yes" button again and listen (Audio Cue 16B) as you play a few notes on the keyboard. Our range is now 1000 Hz - 10,000 Hz and our fixed frequency is now exactly 1000 Hz. Note that we hear the same high-pitched tone regardless of which key we press on the keyboard.

8) Press the "yes" button one more time and note that the range is now 1 Hz - 10 Hz, the fixed frequency is now exactly 1 Hz, and that playing keys on the keyboard yields no sound (since 1 Hz is subaudio). Press the "yes" button two more times to return us to a range of 100 Hz - 1000 Hz.

9) Press the right cursor switch once in order to position the cursor over the "Fine" parameter, and, using the controls in the data entry section, increase the Fine value slowly. Hold down a note on the keyboard as you do so, listening (Audio Cue 16C). Note that, while the pitch you hear sweeps smoothly, the LCD shows us only particular values in the range 100.0 - 977.2 Hz.* Enter a value of 436.5 Hz in the F FINE parameter.

10) Press the right cursor switch once again in order to position the cursor over the "Detune" parameter, and, using the data entry controls, enter a value of +7 here. Operator 1 is now outputting a fixed frequency of 436.5, sharpened slightly to nearly 440. Listen (Audio Cue 16D).

11) TURN OFF operator 1 and TURN ON operator 2 ("010000"). VIEW operator 2. Press edit switch 10 and, using the data entry controls, enter an output level of 99, enabling us to hear operator 2. Play a few notes on the keyboard and listen.

12) Press edit switch 8 and observe the LCD in order to confirm that operator 2 is at its default values (Mode = ratio; ratio number = 1.00, Detune = 0).

* Why these particular values? Because of size restrictions in the DX7II memory, only a finite number of values can be offered to us; what Yamaha's software engineers did was to divide the entire potential Fine range by the total number of values available, and certain frequencies happened to fall at these points. One unfortunate byproduct of this is that we cannot access an exact 440 Hz frequency for reference tuning with this parameter.

13) TURN ON operator 1 again ("110000") and play Middle A on the keyboard. Listen (Audio Cue 16E). There probably will be some beating, since the two operators we are hearing are close in frequency, but not exactly in tune with one another. Operator 2 should be close to 440 Hz when this key is pressed (depending upon the setting of another control called *Master Tuning* . This is accessed with edit switch 14 and will normally be set at 0, indicating standard tuning. More about this in Chapter 13). Operator 1 is also nearly 440 Hz (436.5 Hz, to be exact, but Detuned up +7).

14) Now try playing several different notes and a melody or two (Audio Cue 16F). Listen and note that, while operator 2 tracks the keyboard (since it is in "ratio" mode), operator 1 is playing Middle A as a pedal tone, providing a drone.

15) Press edit switch 1 in order to VIEW operator 1. Position the cursor over the "Fine" control and use the data entry slider to change the pitch of the drone. Experiment with different pitches, and also try changing the Coarse range. Observe that this works by moving the decimal point in the LCD display (i.e., 436.5 Hz can be changed by a factor of ten up to 4365 Hz, or down to 43.65 or even 4.365 Hz!).

16) Experiment by adding in other operators and setting them to various other fixed frequencies. Note that you can create *chords* that drone (not just individual notes!) by using this technique.

17) Experiment by setting several operators to the same fixed frequency and then using the Detune control to create interesting beating effects *within* the drone!

It would appear at the moment that having the option of subaudio fixed frequencies is a waste. Well, that is certainly true when working with unmodulated carriers only. We will see in the very next chapter, however, that *everything*, changes very drastically when we introduce *modulators* into the picture!

In Chapter Three, we talked about the "Oscillator Sync" function, accessed by edit switch 7. If you'd like to now hear the effect of this control, try initializing your DX7II, select algorithm #32 and set the output level of all six operators to the maximum value of 99. Press edit switch 7 and note that the initialization default for this (non-operator-specific) parameter is "Osc Sync" = ON. Play a low-pitched three-note chord on the keyboard and listen - notice that the sound is sharp and that no beating whatsoever is present since all six operators are tracking the keyboard precisely the same way (with a default ratio number of 1.00), and since all three tone generators are starting their waves at the same time. Now use the "no" button to set the Oscillator Sync to OFF. Play the same low-pitched three-note chord and notice that the volume is now a bit softer and that some gentle movements in the sound can be detected. When the sync was ON, all six operators and all three tone generators output their waves perfectly in *phase* with one another (more about this term in Chapter Six). **(see figure 5-8)**

When the sync was OFF, all six operators (and all three tone generators) started their waves at different times, so that they were slightly *out of phase* (again, more about this in Chapter Six), accounting for the movement and slightly smoothed attack that we heard. **(see figure 5-9)**

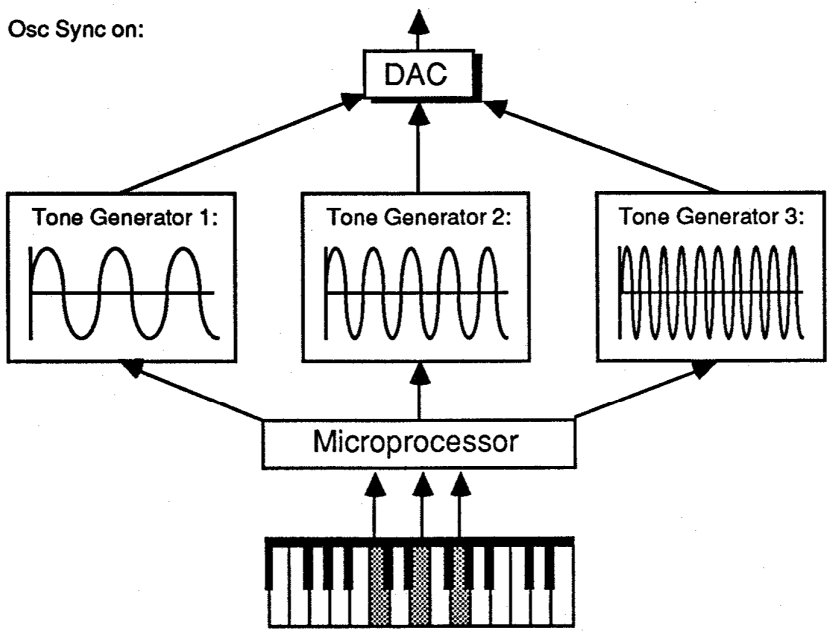


Figure 5-8

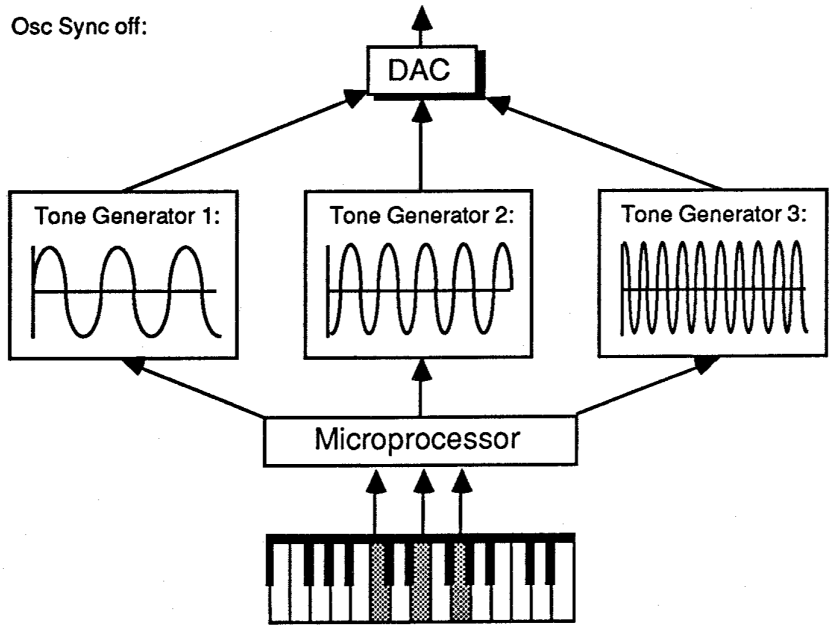


Figure 5-9

This concludes our discussion of the pitch data inputs as applied to an unmodulated carrier. As promised, in the very next chapter, we will explore these same inputs as applied to a modulator - and we will see that things get *very* different! Before you move on, though, I recommend that you continue working on your own with algorithm #32 and get used to the various pitch and output level manipulations we have discussed in this chapter. Remember again that algorithm #32 provides only for *additive* synthesis. When you feel comfortable with all that we have covered here, turn the page and let's make the leap into digital FM as we discover the Wonderful World of Modulators.

Chapter Six ◇
The Modulator

We have learned that the sole purpose of having modulators is to allow us to generate complex timbres - that is, sounds with overtones. Algorithm #32, which we examined in great detail in the previous chapter, allows to do all kinds of wonderful manipulations - but in all cases, we were working with simple sine waves - that is, sounds with no overtones whatsoever.

When we use modulators to cause other operators to generate overtones, there are really only two things that we need to ask ourselves in order to determine what type of sound we hear, and these two questions are:

1) *Which* overtones are we hearing?, and 2) What is the *strength* (i.e., relative amplitude) of each of these overtones?

If we can answer these questions in advance, then we can completely control and predetermine what type of timbres we will be creating. (Bear in mind, again, that until we learn more about envelope generators and a few of the other sound-shaping tools in the DX7II, our sound is still not changing over time, and that this change in a sound is a very important factor. But let's take things one step at a time...)

These two questions can further be described as *qualitative* and *quantitative* ones. The first question - "Which overtones?" - is the qualitative one, and can be answered in a variety of ways. For example, we may be producing *harmonic* overtones exclusively, or *inharmonic* overtones exclusively, or (more often) some combination of both. Next, we will need to know precisely which harmonics, or inharmonics, or combination of each, we are hearing. These are all qualitative questions.

The second question is the quantitative one. Having somehow described which overtones we are generating, we now need to think about the strength of each of these overtones we are hearing, relative to each other and to the fundamental frequency - in other words, how much contribution is each overtone making to the total sound?

As we saw in Chapter One, it is the type and strength of overtones that determines a sound's characteristic timbre, and so asking ourselves these two questions *first* will allow us to selectively generate in the DX7II just about any kind of timbre we can envision.

For each of these questions there is an associated edit parameter that will serve to answer them, and these are as follows:

1) We determine the type of overtones generated (the qualitative question) by something called *frequency ratio*. (Note: this is *not* the aforementioned original DX7 display of "FREQUENCY(RATIO)" - there are no parentheses used here).

2) We determine the relative amplitude (strength) of each of these overtones generated (the quantitative question) by the *output level of the modulator*.

Let's talk about the latter one first, as it's easy to understand. If our modulator is somehow causing the carrier to generate overtones by sending it output signal. **(see figure 6-1)** Then, logically, the more output signal it sends, the more effect it will have - the greater the amplitude of the overtones being generated. This is precisely the way things actually work. And this brings us to another extremely important *Cardinal Rule*:

CARDINAL RULE TWO: CHANGING THE OUTPUT LEVEL OF A MODULATOR WILL ALWAYS RESULT IN A CHANGE IN TIMBRE.

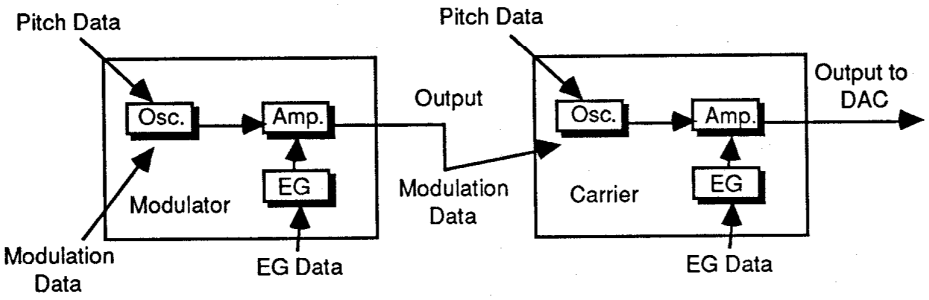


Figure 6-1

Specifically, this will cause a quantitative change in timbre: the greater the output level of the modulator, the greater the amplitude of the overtones we are generating; hence, the *brighter* the sound (the more you "kick" the carrier, the more it "screams"!). The less the output level of the modulator, the lesser the amplitude of the overtones; hence, the *warmer* the sound.

I cannot stress enough how important it is to digest, understand, and thoroughly comprehend these *Cardinal Rules* - after all, that's why they *are* CRs! (for those of you with short memories, Cardinal Rule One, as put forward in the last chapter, states that whenever you change the output level of a *carrier*, you will induce a change in *volume*. This is equally as important as Cardinal Rule Two.)

Now let's go back and talk about the qualitative parameter - the mysteriously previewed *frequency ratio*. In plain English, the frequency ratio is simply the *relative speed* of the operators involved. For example, is the modulator traveling at exactly the same speed as the carrier? Is it traveling at twice the carrier's speed?? Or half its speed?? Or 14.23 times its speed??? **(see figure 6-2)**

Algorithm #1:

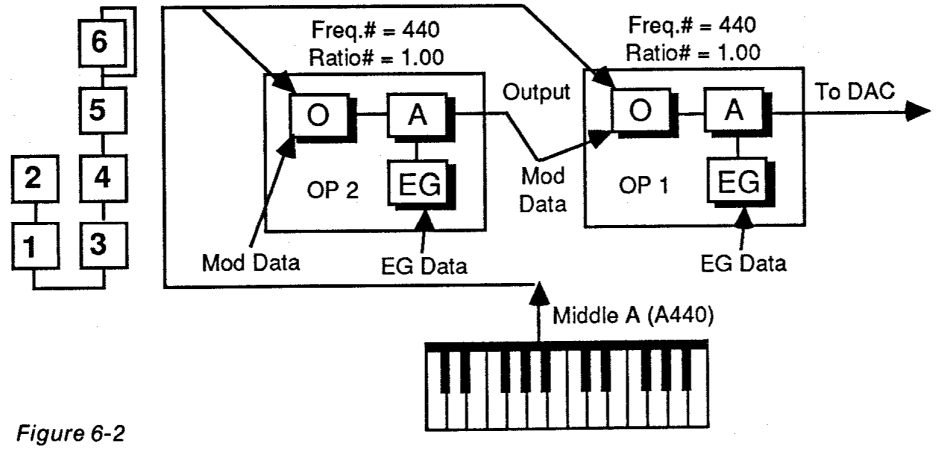


Figure 6-2

In the above example, playing a "Middle A" on the keyboard will cause the carrier to run at exactly 440 Hz, since its ratio number is set to 1.00. The modulator will also travel 440 times per second, since it has the same ratio number. Playing the "A" below "Middle A" will similarly cause both of them to run at 220 Hz. In fact, playing any note at all will cause both of them to travel at *precisely* the same frequency, since they have the same ratio number. The frequency ratio is, in the jargon, expressed as two numbers separated by a colon, as in any algebraic ratio. The convention with digital FM instruments is that the first number of the ratio represents the modulator. Hence, the frequency ratio in the above example would be *1:1*.

On the other hand, suppose we were to double the ratio number for our modulator only. **(see figure 6-3)**

Algorithm #1:

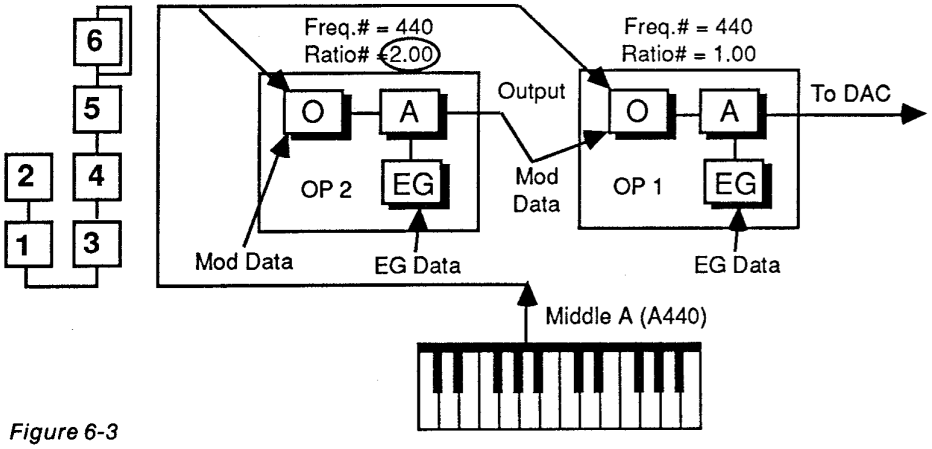


Figure 6-3

Playing "Middle A" on the keyboard would still cause operator 1, the carrier, to run at 440 Hz, since its ratio number is still 1.00. However, since operator 2, the modulator, now has a ratio number of 2.00, it will now output a signal 880 times per second! The frequency ratio in this case would be said to be *2:1*. If we reversed things so that operator 2's ratio number were restored to 1.00, and operator 1's doubled to 2.00. **(see figure 6-4)**

Algorithm #1:

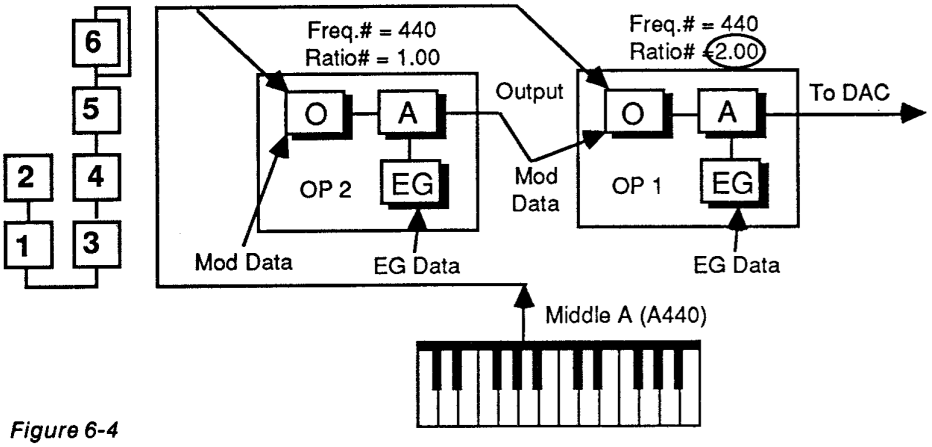


Figure 6-4

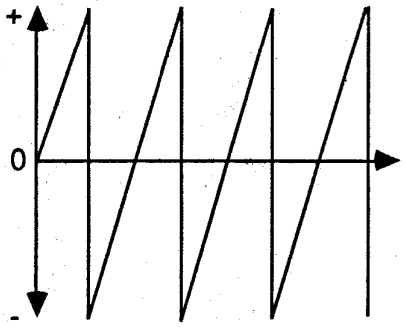
then our frequency ratio would now be *1:2*. In this case, the carrier is traveling at *half* the speed of the modulator. Remember, the first number represents the modulator, and the second, the receiving operator (in this particular example, a carrier).

Frequency ratios can of course be expressed as decimals instead of just whole numbers. For example, a frequency ratio of *4.63:1* would mean that our modulator was always traveling 4.63 times faster than the carrier. A frequency ratio of *2:7* would mean that the *carrier* was always traveling 3.5 times faster than the modulator (think about it!).

For those of you who were never very good at math, this may be a little bit problematic, but stick with it, because this is obviously a very important factor in creating sounds on the DX7II. What we are saying here is that *every time you vary the relative speeds of the modulator and the carrier, you will hear a totally different set of overtones*. The enormous value of this concept is that it allows us to construct on the DX7II virtually ANY kind of waveshape!

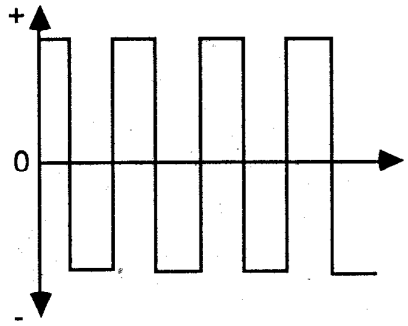
Later on in this chapter we will be talking more about the concept of "frequency ratio" and its importance in creating sounds but for now, let's see how we can use these two controls - the *qualitative* and the *quantitative* - to make some more new sounds.

Analog synthesizers, which usually employ subtractive synthesis systems, generate timbres by starting with one or two fixed waveshapes and then removing whatever overtones are not desired. The fixed waveshapes used in analog synthesis are important to us because analog synthesizers have, after all, been around a lot longer than the DX7II; and because traditional so-called "synthesized" or "electronic" sounds have generally been constructed using these particular waveshapes. The first standard analog timbre, the sine wave, is one with which we have already become very familiar. Two others that we will work with now are called the *sawtooth* (or *ramp*) wave; and the *square* wave. Keeping in mind, of course, that the timbre of a sound determines its wave *shape*, let's take a look at how these two waves appear on an oscilloscope. **(see figures 6-5(a) and 6-5)**



Sawtooth (Ramp) Wave

Figure 6-5(a)



Square Wave

Figure 6-5(b)

These waves have very different shapes because they have very different *overtone* contents. The sawtooth wave (sometimes called a ramp wave) is very rich in harmonics, and actually contains *all* the harmonic overtones, in proportionally decreasing amounts. That is, it has a great deal of fundamental, followed by half as much second harmonic, followed by a third as much third harmonic, etc., etc., until we can't hear any more. **(see figure 6-6)**

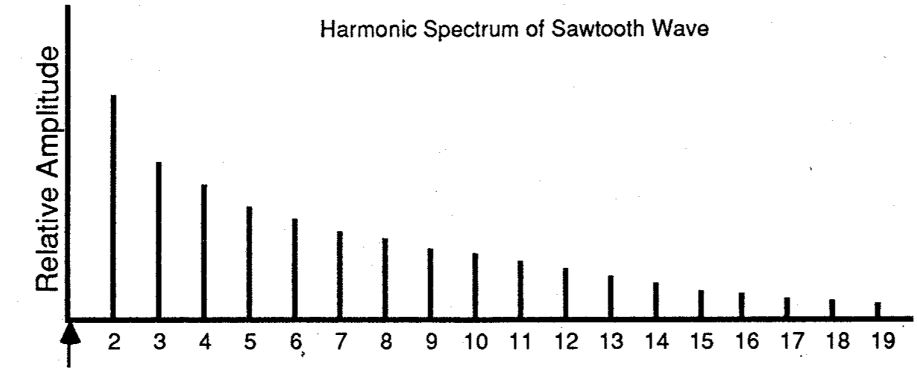


Figure 6-6

This wave, being very musical, contains little or no inharmonic content. The characteristic sound of the sawtooth wave is very bright and buzzy. This makes sense since it contains many overtones, in large quantities (remember - the greater the amplitude of the overtones, the brighter the sound). Typically, sawtooth waves are used to generate the big, brassy sounds that analog synthesizers are so renowned for.

Not wanting to be left out, the DX7II can also generate a sawtooth wave, and quite easily, at that! There is a recipe we can follow, and it goes like this:

RECIPE: Set the frequency ratio to *1:1*. Set the output level of the modulator to maximum (99). Set in a cookie dish and bake well 45 minutes. Serves many. (Sorry 'bout that! Ignore the last two sentences, and let's get back to business...)

Seriously, this *is* the recipe for generating a sawtooth wave on the DX7II and, in just a moment, we'll run an exercise and try it. First, there must surely be those among you who are wondering, Why? Why a sawtooth wave and not a "grpxkdtl" wave?? (somewhere out there, I *know* someone is frantically looking up "grpxkdtl" in a dictionary. For you, sir or madam: there is no such thing.) What's so special about these particular settings (frequency ratio of 1:1 and modulator output level of 99)?? For those who truly want to know, there is a mathematical explanation, but you won't get it here! The theory of digital FM to this degree is unfortunately well beyond the scope of this book. The interested reader is therefore referred to the excellent book, "FM Theory and Application", by Dr. John Chowning (the originator of the digital FMprocess) and David Bristow (one of Yamaha's top programmers and clinicians). In addition, Dr. Chowning has published many scientific papers in the Computer Music Journal and in several Audio Engineering Society publications. These publications explain the many bylaws of digital FM in great technical and mathematical detail. I would steer any of you who are interested in such detailed explanations in that direction. But just as it is unnecessary to understand the theory of combustion engines in order to drive a car, it is unnecessary to understand these complex theorems in order to use a DX7II. We'll only be following a couple of these unsubstantiated recipes, so I ask your indulgence, and just say (in the magic words of the music business) - trust me! They work!!

Now we're about ready to run our exercise. For this exercise, and quite a few more to boot, we will be working not with algorithm #32, as in the past, but instead with the default algorithm, algorithm #1. **(see figure 6-7)**

Algorithm #1:

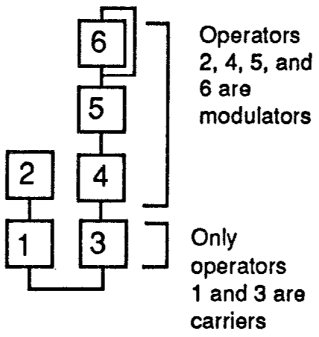


Figure 6-7

Algorithm #32 is obviously unsuitable for our purposes here since it doesn't provide us with any modulators. All of the remaining 31 algorithms do but we might just as well stick with #1, as it is where the DX7II puts us when we initialize. We will not concern ourselves with the right-hand stack of operators 3, 4, 5, and 6; but we will simply be using the simple *system* of operators 1 and 2. **(see figure 6-8)**

As our diagram clearly indicates (and by all means check the algorithm diagram on the machine itself to confirm that our diagram is correct!), in algorithm #1, operator 1 is a carrier, and operator 2 is being used as a modulator, specifically *sending output into the modulation data input of operator 1*. This means that as we increase the output level of operator 2, operator 1 will generate greater and greater amounts of overtones. What type of overtones? In this case, all of the harmonic overtones in a proportionally decreasing fashion, since we will take care to leave both operators at their default ratio numbers of 1.00 (thus setting up a frequency ratio of, you guessed it!, 1:1). This will cause us to generate a particular type of waveshape: the sawtooth wave.

Algorithm #1

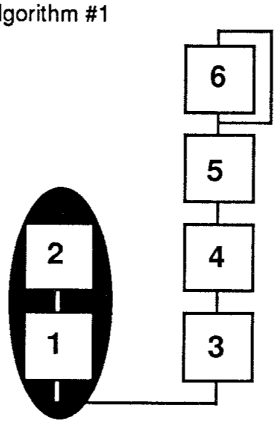


Figure 6-8

Exercise 17
Generating a sawtooth wave

1) INITIALIZE your DX7II from single voice play mode and press edit switch 7 in order to confirm that you are in the default algorithm, algorithm #1.

2) Press edit switch 10 (OUTPUT LEVEL). Press edit switches 19 through 22 in order to TURN OFF operators 3 through 6 ("110000") and press edit switch 1 in order to VIEW operator 1 (if you are not already viewing it).

3) Observe that operator 1 is at its default output level of 99, and press edit switch 2 in order to VIEW operator 2 and confirm that it is at *its* default output level of 0. Listen to the single sine wave. (Audio Cue 17A). Even though both operators are ON, and, even though this algorithm is configured so that operator 2 is modulating operator 1, it is currently having no effect whatever on operator 1 since operator 2's output level is defaulted to 0.

4) Use the cursor switches to position the cursor over the "Level" parameter. Begin slowly raising the data entry slider, thereby increasing the output level of operator 2. Make sure you keep tapping a note on the keyboard as you do this since output level does NOT change in real time (refer to Chapter Four if this doesn't sound familiar).

5) Listen (Audio Cue 17B). As you increase the output level of operator 2, the sine wave begins to change in *timbre*, getting slowly but steadily brighter, as more and more overtones are generated. When you finally reach maximum output level (99), the sound is as bright as it is going to get (Audio Cue 17C). The wave you are now listening to is a sawtooth wave!

6) Experiment by slowly reducing the output level of operator 2 (Audio Cue 17D). Note that the sound gets warmer, as the harmonic content of the sound decreases. Also note that relatively little change occurs with output levels of 0 to 50; as the numbers increase, the change becomes more drastic. The change between output levels 90 and 99, for instance, is far more noticeable than that from 80 to 89.*

7) Experiment by restoring the output level of operator 2 to 99 and then slowly reducing the output level of operator 1 (Audio Cue 17E). Note that, following the dictums of Cardinal Rule 1, *only* the volume of the sound changes, and that no timbral change whatsoever occurs.

For those of you with prior experience in analog synthesis, you have just heard something very similar to the opening and closing of an analog low-pass filter (this is a filter that selectively removes overtones, from highest to lowest). Of course, we're doing nothing of the kind, since the DX7II does not contain filters of any description, but it's really pretty amazing that two totally dissimilar systems will produce exactly the same aural effect!

More importantly, we were able to hear for ourselves just how changing the output levels of modulators and carriers differs. When we altered the output of operator 2, clearly the timbre changed, and in such a way that greater output level resulted directly in a brighter sound. When we altered the output of operator 1, clearly only the volume changed, and in such a way that lesser output level resulted directly in a quieter sound. Next, we'll be experimenting with altering the *qualitative* factor (the frequency ratio) in order to hear *different* (not just less or more) overtones. Let's dip our toes in by following another recipe, this time to build that good old analog standby, the *square* wave.

* This is because the output level is changing exponentially rather than linearly. Chapter Twelve contains a detailed explanation of these terms, but until we get to that point, just take note of this phenomenon.

The square wave was not named by a beatnik in the early '60s, nor is it the sound generated by Lawrence Welk's bubble-blowing machine. Instead, it is an interesting timbre which consists of the fundamental frequency plus only its *odd-numbered* harmonics, in proportionally decreasing amounts. For example, a square wave having a fundamental frequency of 440 Hz will also contain a third as much 1320 Hz (the third harmonic), a fifth as much 2200 Hz (the fifth harmonic), and a seventh as much 3080 Hz (the seventh harmonic), etc. Unlike the sawtooth wave, we will not hear any 880 Hz (the second harmonic), 1760 Hz (the fourth harmonic), or 2640 Hz (the sixth harmonic). **(see figure 6-9)**

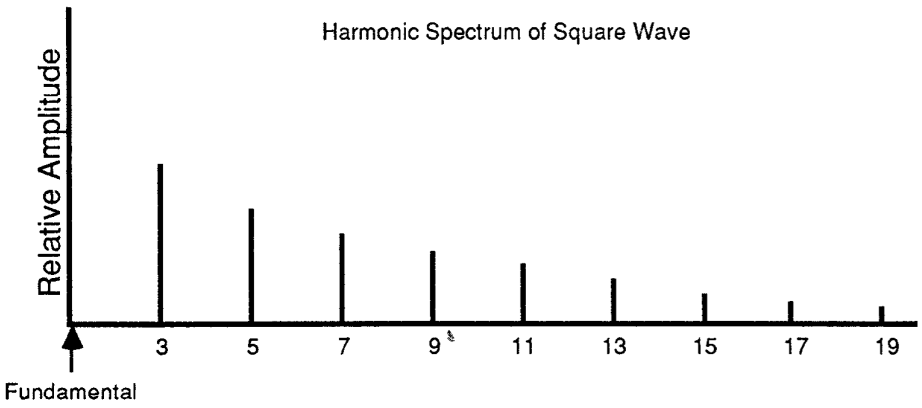


Figure 6-9

This means, of course, that the square wave will sound quite different from the sawtooth wave, and indeed it does - it has a peculiar hollow sound, brought on by the fact that it is literally missing even-numbered harmonics. Again, we will hear few if any inharmonics, since this is a very musical timbre. Square waves tend to sound very clarinetish, and are generally used in analog systems to generate woodwind or stringed sounds.

The recipe for generating a square wave in the DX7II is as follows:

RECIPE: Set the frequency ratio to *2:1* (modulator at exactly twice the frequency of the carrier). Set the output level of the modulator to 71. Voilà! Instant square wave souffle!!

Exercise 18
Generating a square wave

1) INITIALIZE your DX7II from single voice play mode and use edit switches 19 through 22 in order to TURN OFF operators 3 through 6 ("110000").* Leave the default algorithm - #1.

2) Press edit switch 8 (OSCILLATOR) and press edit switch 2 in order to VIEW the ratio number for operator 2. Note that it is currently at its default value of 1.00.

3) Use the cursor switch in order to position the blinking cursor over the "Coarse" parameter. Press the "yes" button in the data entry section in order to change the Coarse value for operator 2 to 2.00. Operator 2 will now travel at precisely twice the speed of operator 1 (which is still at its default Coarse setting of 1.00 - check to confirm) regardless of what key you play.

* By now you should realize that the only way you can use edit switches 17 through 22 to turn operators on and off is if you have previously selected an *operator-specific* edit switch - that is, one of edit switches 8, 9, 10, or 11. In all future exercises, I will assume that you are aware of this and will not make reference to it again.

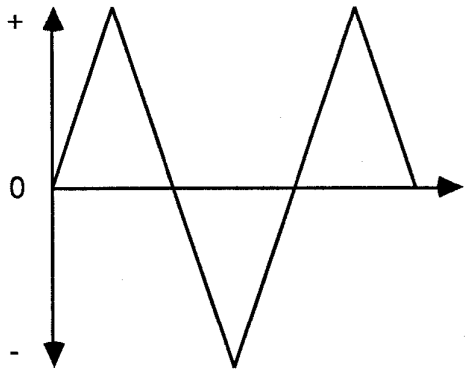


Figure 6-10

4) Play a note on the keyboard and listen (Audio Cue 18A). Even though we have altered the Coarse value for operator 2, we are still only hearing a single sine wave because operator 2's output level is still at its default value of 0 (check to confirm this).

5) Press edit switch 10 (OUTPUT LEVEL) to VIEW this output level value for operator 2. (=0)

6) Using the data entry slider, slowly increase this value to 71, tapping a note on the keyboard and Listening as you do so (Audio Cue 18B). As in the last exercise, you cannot simply hold a note down since output level does not change in real time. Once again, as you increase the output level, the sound becomes brighter. At the exact modulator output level value of 71, you are hearing a pure square wave (Audio Cue 18C). Listen to the typical "clarinet-ish" type of timbre.

7) Experiment by increasing the value of operator 2's output level above 71 (Audio Cue 18D). Once again, the sound becomes brighter, since the amplitude of our overtones is increasing beyond the point typically found in a square wave. But note that we are still only hearing odd-numbered overtones; the *quality* of the overtone content does not change.

8) Experiment by decreasing the value of operator 1's output level and note that once again, *only* the volume of the sound changes (Audio Cue 18E).

The fourth basic analog waveshape, the *triangle* wave, looks like **figure 6-10** on an oscilloscopeand has very little harmonic content; essentially just a bit of the third harmonic and a pinch of the fifth. **(see figure 6-11)**

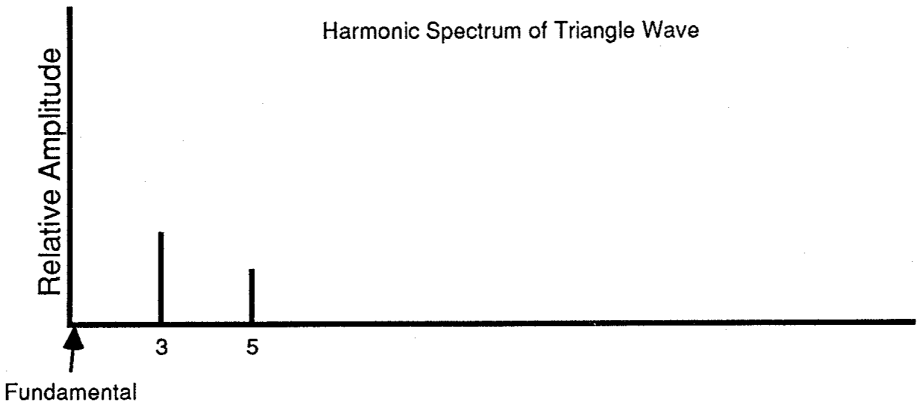


Figure 6-11

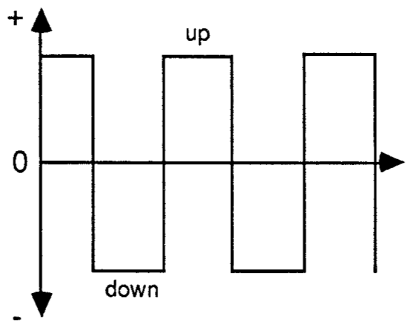
This wave generates a characteristic flute-like sound and is typically used in analog synthesizers for (you guessed it!) flute-ish sounds. We've seen that the square wave also contained the third and fifth harmonics (albeit in much larger quantities) with no in-between harmonics (specifically the second and fourth). Logically, if we can reduce the relative amounts of third and fifth harmonics in our square wave and somehow get rid of any harmonics above that point, we will be able to generate a triangle wave. How do we accomplish this? By returning to our *quantitative* control.

Very simply, lessening the output signal from our modulator (running at twice the speed of our carrier), will have the result of reducing the relative amounts of the third and fifth harmonic to the point where they emulate that found in the triangle wave. If we diminish these relative amounts to the point where we are only getting a very

small quantity of the fifth harmonic, then all the harmonics *above* the fifth will also be reduced in quantity to the point where their contributions to the overall sound are negligible. The end result will be: a triangle wave!

The recipe, then, for generating a triangle wave in the DX7II is very simple: make a square wave, and then reduce the output level of the modulator down to about 45. You will hear a very flute-ish sound, and, on an oscilloscope, you will see a very triangle-ish wave shape. Try making this waveshape on your own.

Before we leave this section, we should briefly touch upon another analog phenomenon, and that is something called *pulse width*. The square wave is in fact a particular kind of wave called a *pulse* wave. This wave, as we have seen, only exists in a simple up-down fashion. **(see figure 6-12)**



Square Wave

Figure 6-12

Remember that in these diagrams we are mapping amplitude (the Y-axis) versus time (the X-axis). The *pulse width control* on an analog synthesizer allows us to vary the *relative* amounts of time that the pulse wave spends in its "up" condition versus its "down" condition (while keeping the *total* time required for one complete cycle the same). For example, we can alter our pulse wave so that it stays "up" far longer than it drops "down". **(see figure 6-13)**

Or we can do precisely the reverse. **(see figure 6-14)**

A pulse wave that stays "up" *exactly* as long as it stays "down" is called a *square* wave. **(see figure 6-15)**

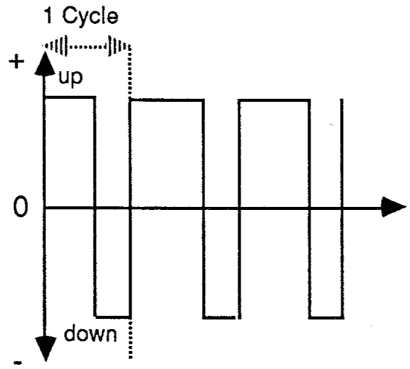


Figure 6-13

Therefore, all square waves are pulse waves but not all pulse waves are square waves (think about it!). In any event, when we "narrow" or "broaden" the pulse width by changing these relative times, what we are in fact doing is changing the timbre, and not the pitch or volume, of the sound. The illustrations above clearly show that at no time is the height of the wave changing, and at no time are we increasing or decreasing the total number of waves generated per second.

How does the timbre change? Well, remember that our pure square wave (which can now be referred to as a 50% pulse wave; i.e. 50% "up" followed by 50% "down") contained only the odd-numbered harmonics. When we change the pulse width, either by narrowing or broadening it, what we are doing is reintroducing those even-numbered harmonics that previously were missing, as we simultaneously reduce the number of odd-numbered harmonics, and of the fundamental itself. The end result is a striking timbral change, resulting in a sound that gets progressively thinner and more nasal. Surprisingly, whether you choose to narrow *or* broaden the pulse width, the aural effect is the same! That is, a 25% pulse wave (one that stays "up" 25% of the time and "down" the remaining 75%) sounds exactly the same as a 75% pulse wave. A 10% pulse wave sounds the same as a 90% pulse wave, etc., etc.

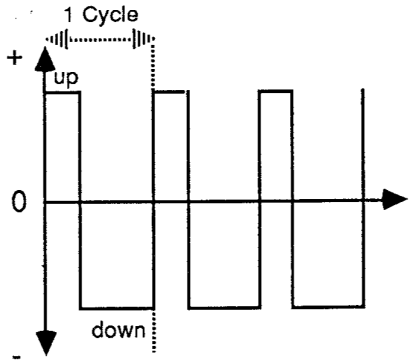


Figure 6-14

This pulse width control is a powerful tool for analog synthesists, as it allows them to further specify harmonic content even after they have selected an initial waveshape (if and only if they have selected a pulse wave as this initial waveshape - this control has no effect whatsoever on the other analog waveshapes). For example, the analog synthesist might choose a 50% pulse wave for a clarinet sound, but a 35% pulse wave instead for an oboe sound, and a 17% pulse wave for a mandolin sound.

In the DX7II, we can also generate various different pulse widths, not with the degree of control found on an analog synthesizer, but enough to get by (after all, with so many more timbres available in the DX7II, it's unlikely that this will worry too many people). The recipe for generating various pulse waves of different width is as follows:

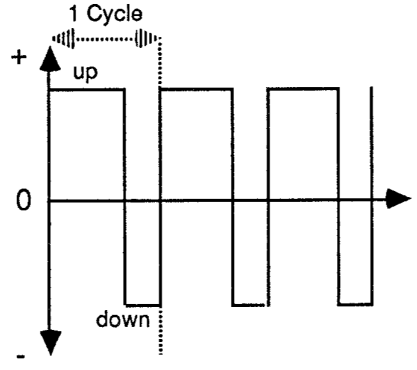
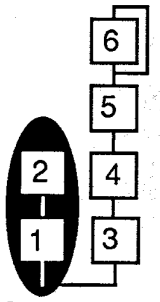


Figure 6-15

Algorithm #1:



(not just modulator and carrier)

Figure 6-16

Algorithm # 5:

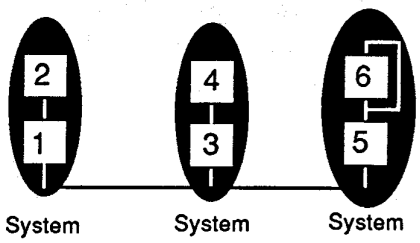


Figure 6-17

Algorithm #1:

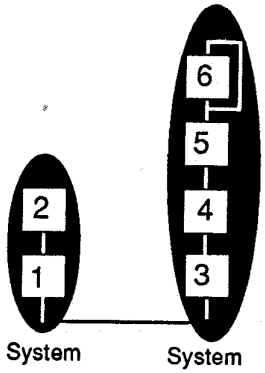


Figure 6-18

Algorithm #16:

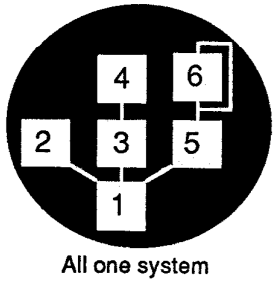


Figure 6-19

RECIPE: Make a square wave (frequency ratio of *2:1* and modulator output level of 71). Now change the ratio number of the *carrier* to any *odd* number (i.e. *2:3, 2:5, 2:7, 2:9,* etc.).

As you raise the ratio number of the carrier, you will hear the sound become thinner and more nasal. In fact, whenever you set up any kind of frequency ratio where the carrier is travelling *faster* than the modulator, this effect, similar to the use of an analog *high-pass filter* (that is, a filter which increasingly removes the lowest components of a sound's harmonic spectrum), will result. Here, what we are in fact doing is altering the pulse width of our square wave, therefore changing the timbre of the sound. Let's try it:

Exercise 19
Generating different pulse waves

1) INITIALIZE your DX7II from single voice play mode and leave it in algorithm #1. TURN OFF operators 3 through 6 ("110000").

2) GENERATE a square wave (if you don't remember how, refer to Exercise 19 above).

3) Listen to the clarinet-like sound made by this 50% square wave (Audio Cue 19A).

4) Press edit switch 8 (OSCILLATOR) and, if necessary, press edit switch 1 in order to VIEW the ratio number value for operator 1. (= 1.00)

5) Use the cursor switch in order to position the blinking cursor over the Coarse parameter and use the "yes" button in the data entry section in order to change this value to 3.00. Listen (Audio Cue 19B).

6) Repeat step 5 above, this time changing the value to 5.00. Listen (Audio Cue 19C).

7) Continue repeating step 5 above, each time changing the value for operator 1's Coarse parameter to another *odd* number only (change to values of 7.00, 9.00, 11.00, 13.00, 15.00, 17.00, 19.00, 21.00, 23.00, 25.00, 27.00, 29.00, and 31.00). Listen (Audio Cue 19D). Note that as the value goes higher, the sound gets thinner and more nasal. Note also that at the very highest settings, inharmonics begin creeping in as the sound begins losing its overall discernible pitch. Note that at no time do we hear volume changes.

8) Experiment. Restore the Coarse value of operator 1 to 1.00, thereby restoring our square wave. Listen to confirm that. Now try changing the F COARSE value for Operator 1 to *even* numbers (2.00, 4.00, 6.00, etc.). Listen (Audio Cue 19E). Note that while the timbre does indeed change, it does not change the same way as before, and that for each of the even-numbered ratios, the sound goes *up* an octave!

Now things are really starting to get interesting! Why did the sound jump up an octave? Well, let's look at exactly what we did: We began by changing the Coarse value for operator 1 to 2.00. The Coarse value for operator 2 had previously also been set to 2.00. Therefore, our frequency ratio was now *2:2, which is exactly the same as 1:1!!* Remember, the frequency ratio is a *relative* number, and so what we heard when we changed the ratio to *2:2* was *not* a pulse wave at all, but instead a *sawtooth* wave *an octave higher!* (Think about it, and it will make sense!)

But something else strange is starting to happen. What we did in that last exercise was change the Coarse value for our *carrier*. Didn't we learn in Chapter Five that altering this number changes the *pitch* of the sound? We did, but I also cautioned you at the end of the chapter that carriers behave very differently once modulators are plugged into them! And here is graphic evidence that this is indeed true.

Once you plug a modulator into a carrier (by virtue of instructing the DX7II to give you an algorithm with such a configuration), you should no longer think about the modulators and carriers as separate entities. Instead, as we briefly alluded to earlier in this book, you should think of *systems*. In Exercises 17, 18, and 19, we were working with the *system* of operators 1 and 2, not just with carrier 1 and modulator 2. **(see figure 6-16)**

With this concept firmly in mind, it would probably be a good idea to go back and reexamine the 32 different algorithms. Note, for example, that algorithm #5 (which we will soon be working with frequently) offers us three independent systems. **(see figure 6-17)** Algorithm #1, for example, only offers us two. **(see figure 6-18)** Algorithm #16, only one. **(see figure 6-19)** Algorithm #32, of course, offered us six systems, with each system being a simple unmodulated carrier. With some other algorithms, it isn't quite so obvious. How many systems, for example, are offered us by algorithm #20? **(see figure 6-20)**

The correct answer is *two*. Operators 1 and 2 are *both* being modulated by operator 3, so altogether this yields one system. Similarly, operator 4 is being modulated by both operators 5 and 6, so they comprise another system. Let's try one more. **(see figure 6-21)**

The correct answer for this algorithm (#24) is *three*. Operators 3, 4, and 5 are all being modulated by operator 6 and so all together they comprise one system. Operators 1 and 2 are simply unmodulated carriers, and each can be considered an independent system.

When dealing with a system of modulator plugged into carrier, it should be clear that we cannot initiate a pitch change by simply changing the pitch data input to the carrier alone. We can't accomplish a pitch change by altering the pitch data input to the modulator alone, either. So how do we accomplish a pitch change to a system? Simple. We change the pitch data input to *both* of them, *the same way*.

This makes sense when you realize that altering the pitch of one or the other alone actually upsets the *frequency ratio:* **(see figure 6-22)**

Changing the frequency ratio cannot and will not ever cause a pitch change, but instead will result in a qualitative timbral change. By changing the pitch data input to both the modulator *and* carrier, *equally*, you maintain the current frequency ratio; hence, only a pitch change occurs. **(see figure 6-23)**

After all, as we have seen, a frequency ratio of 2:2 is the same as 1:1; similarly, a frequency ratio of 4:2 is the same as 2:1, and so is 8:4; 16:8; or 25.4:12.7! In each of these instances, we would be hearing a square wave at a different pitch. Let's try it:

Exercise 20
Altering the pitch of a complex timbre

1) INITIALIZE your DX7II from single voice play mode, leave it in algorithm #1, TURN OFF operators 3 through 6 ("110000"), and GENERATE a square wave with the system of operators 1 and 2. Hold a note down on the keyboard and listen (Audio Cue 20A).

2) Press main switch 8 (OSCILLATOR) and, while continuing to hold the same note down, use the appropriate operator select, cursor, and data entry switches to *double* the Coarse value for operator 1 only from 1.00 to 2.00 (yielding a frequency ratio of 2:2).

3) Listen (Audio Cue 20B). What you are currently hearing is a quasi-sawtooth wave (since operator 2's output level is 71 and not 99), an octave higher.

Algorithm #20:

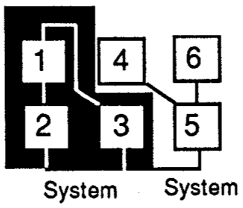


Figure 6-20

Algorithm #24:

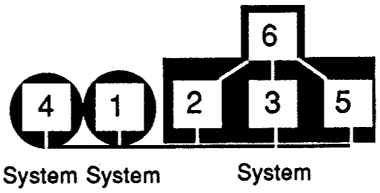


Figure 6-21

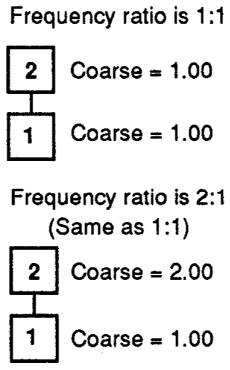


Figure 6-22

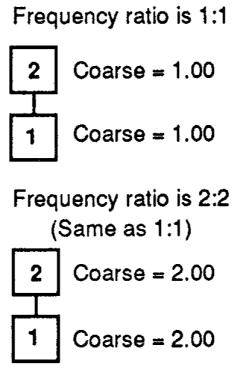


Figure 6-23

4) While continuing to hold down the note on the keyboard, VIEW operator 2 and *double* its Coarse value as well, from 2.00 to 4.00 (making the frequency ratio now 4:2). Listen (Audio Cue 20C). Since a frequency ratio of 4:2 is the same as a ratio of 2:1, what you are now hearing is a *square* wave, an octave higher. Note that we were able to keep a note held down throughout this procedure because we were changing pitch data input values, which, unlike output level values, *do* change in real time.

5) Using the appropriate operator select, cursor, and data entry switches, redouble the Coarse values for *both* operators 1 and 2 (yielding a frequency ratio of 8:4). Listen (Audio Cue 20D). What you are now hearing is the same square wave, up yet another octave. Note that neither the timbre nor the volume of the sound has changed in any way.

6) Experiment by setting different Coarse and/or Fine values for operators 1 and 2, taking care to always keep an overall ratio of 2:1 (in other words, always keeping operator 2 at *twice* the frequency of operator 1). Listen and note that, regardless of the specific values used, the timbre remains a square wave.

All of which brings us happily to Cardinal Rule 3:

CARDINAL RULE THREE: TO OBTAIN A PITCH CHANGE IN A SYSTEM, YOU MUST ALTER THE PITCH DATA INPUT VALUES FOR ALL OPERATORS *EQUALLY*.

The corollary to Cardinal Rule Three states that **altering the pitch data input value for one operator in a system without affecting the other ones *equally*, will ALWAYS result in a (qualitative) timbral change ONLY.**

This Cardinal Rule, like the two others before it, is of great importance. Again, it is my strongest advice that you eat, drink, sleep, digest, comprehend, verify, and thoroughly understand this rule before proceeding any further.

So far, we have limited ourselves to working with only two complex waveshapes, the sawtooth and the pulse. We've seen that setting up a frequency ratio of 1:1 with the proper modulator output level (99) yields a sawtooth wave, and that setting up a frequency ratio of 2:1 with the proper modulator output level (71) yields a square wave. But what happens if we set up a frequency ratio of 3:1? Or 4:1?? Or 27:1??? Well, obviously, changing the frequency ratio will qualitatively change the type of overtones we hear, so each of these manipulations will yield a new and different waveshape. What are the names of these various waveshapes? Well, they have no names! They're just - well, new and different. Beyond the obvious recipes for typical analog waveshapes lies the real strength of the DX7II - a doorway is opening up that allows us to literally generate new timbres that may not have previously existed - ever! Remember too that we are not limited to changing the modulator pitch data input only - as we've just seen, changing this value for the carrier will also initiate new timbres for which there are no names. Also bear in mind that setting up a frequency ratio of 4:1, for example, will yield an entirely different waveshape than a frequency ratio of 1:4. With a ratio number range of 0.50 to nearly double 31.00 (61.69, to be exact), you can see that the number of possibilities, while mathematically finite, are indeed astronomical in size!

The rules governing the types of timbres that are generated when altering the frequency ratio are complicated indeed and are, for the most part, outside of the scope of this book. However, in general, we can state two of them:

a) As the modulator travels faster and faster relative to the carrier, higher overtones will increase in strength, and the gap between the fundamental and its nearest overtone will lengthen. The strength of the carrier itself will, however, remain constant. At extreme differences in speed, high inharmonic overtones will begin to predominate.

b) As the carrier travels faster and faster relative to the modulator, higher overtones will increase in strength as the fundamental decreases in strength - making the sound thinner and more nasal.

We've already experienced the phenomenon given in the first rule when we made our sawtooth and square waves. The frequency ratio for the sawtooth wave was 1:1, and the nearest overtone to the fundamental was the second harmonic. The frequency ratio for the square wave, on the other hand, was 2:1 - and the nearest overtone to the fundamental was the third harmonic. We also found that increasing the carrier's speed in this 2:1 ratio (that is, setting up a ratio of 2:3, 2:5, or 2:7) produced a thinner, more nasal type of sound, as predicted by the second rule of thumb.

Let's try running an exercise, firstly to examine some of these different timbres induced by *whole-number* frequency ratio changes (the next exercise will deal with *non- whole-number changes*). You will notice that, as the modulator and carrier get further apart from one another in frequency, rather more inharmonics than harmonics begin to make their presence felt. This, of course, will result in an apparent change in pitch as the sound becomes more dissonant, but remember that timbre is itself a frequency-dependent phenomenon, and so what we are really hearing is a timbral change, albeit one that sounds like a pitch change.

Exercise 21
Generating new complex timbres using whole-number frequency ratio changes

1) INITIALIZE your DX7II from single voice play mode, leave it in algorithm #1, TURN OFF operators 3 through 6 ("110000"), and, using the system of operators 1 and 2, GENERATE a sawtooth wave (if you don't remember how to do this, refer to Exercise 18 earlier in this chapter). Hold a note down on the keyboard and LISTEN (Audio Cue 21A).

2) Press edit switch 8 (OSCILLATOR) and press edit switch 2 in order to VIEW the Coarse parameter for operator 2. Use the cursor switch to position the blinking cursor over the Coarse parameter. While continuing to hold the note down on the keyboard, use the data entry slider to slowly increase this value through its various numbers until you reach the maximum value of 31.00. LISTEN (Audio Cue 21B). Note that you can hear this change in real time since we are not altering output level.

3) Restore the frequency ratio to 1:1 by returning the Coarse value for operator 2 to 1.00.

4) Press edit switch 1 in order to VIEW the Coarse parameter for operator 1. Hold down a note on the keyboard, and use the data entry slider to slowly increase this value through its various numbers until you reach the maximum value of 31.00. Listen (Audio Cue 21C). Note

that you are again hearing timbral change, but that it is qualitatively quite different from what you heard in step 2 above - with the sound getting considerably thinner and more nasal.

5) Experiment by altering the Coarse value for *both* operators 1 and 2 in various ways. Note, for example, that setting up a frequency ratio of 14:5 yields a very different timbre from that generated by a frequency ratio of 5:14 (Try it!).

Of course, nothing in the DX7II works by itself alone. We have seen time and time again that the various edit parameters work in concert with one another to produce the finished result of what you hear. You've certainly noticed in the preceding exercise that running a modulator at very high frequencies induces sounds that are unpleasantly dissonant and often overbright. Well, don't forget that you can alter the quantitative control as well as the qualitative one at any time. So, if the sound produced by the frequency ratio of, say, 29:1 was extremely jarring to your ears, go back and reduce the output level of operator 2. All of sudden, you'll find that that horrible 29:1 timbre becomes quite pleasant and musically usable. With this in mind, go back and repeat Exercise 22, and this time alter the modulator output level (that is, operator 2) from time to time as you go through the various frequency ratios.

We are certainly not limited to only working with whole-number frequency ratios (i.e. 19:1, 15:2, etc.). The Frequency Fine parameter will, after all, permit us to set up *non-whole-number* frequency ratios as well. Using these types of frequency ratios will allow us to generate many more inharmonics than harmonics, resulting in non-musical sounds. Let's run an exercise to try it!

Exercise 22
Generating non-musical timbres

1) INITIALIZE your DX7II from single voice play mode, leave it in algorithm #1, TURN OFF operators 3 through 6 ("110000"), and GENERATE a sawtooth wave using the system of operators 1 and 2. Listen (Audio Cue 22A).

2) Press edit switch 8, followed by edit switch 2, in order to VIEW the ratio number for operator 2. (= 1.00)

3) Use the cursor switches to position the blinking cursor over the Fine parameter. Hold a key down on the keyboard, and use the data entry slider to slowly change this value up to its maximum of 1.99, listening as you do so (Audio Cue 22B).

4) Return the Fine value to 1.00 and press the left cursor switch once in order to position the cursor over the Coarse parameter. Press the "yes" button in order to change this value to 2.00.

5) Redo steps 2 and 3, this time changing the Fine value slowly up to its maximum of 3.98, listening as you do so (Audio Cue 22C).

6) Return the Fine value back to 2.00, and then the Coarse value back to 1.00 (in other words, restore your sawtooth wave. Listen to confirm this).

7) Press edit switch 1 in order to VIEW operator 1.

8) Hold a key down on the keyboard, and slowly CHANGE the Fine parameter for operator 1 up to its maximum value of 1.99, listening as you do so (Audio Cue 22D). Note that there is a qualitative difference in the sound from that which you previously heard in step 3 (i.e., a frequency ratio of, say, 1.00:1.37 produces a very different timbre from that of a ratio of 1.37:1.00. Both timbres, however, are largely inharmonic and therefore non-musical).

9) Experiment with other non-whole number frequency ratios by altering the Fine and Coarse parameters for various modulators and carriers. Note that any time a non-whole number frequency ratio is set up, a inharmonic timbre results.

As with the previous exercise, remember that we are here only manipulating the qualitative control. Using the quantitative control - that is, changing the output level of the modulator - will help make these jarring, disharmonious sounds a lot easier to take. Learn to balance the two controls in order to establish precisely the timbre you require.

Altering the Frequency Fine parameter within a system, then, allows us to generate inharmonic timbres. But what effect can we expect from the Detune control, which, after all, is just a finer Frequency Fine control? The answer is, somewhat surprisingly, that we get very similar effects to that which we encountered when detuning lone carriers!

Try setting up a frequency ratio on your DX7II of 1.01:1.00 (for simplicity, just re-initialize, keep yourself in algorithm #1, and just use the system of operators 1 and 2. Remember to raise the output level of operator 2 from its default of 0!). Listen to the resulting timbre, and you will hear a *beating* effect very similar to that which we heard in Chapter Four. However, we are now obviously hearing only *one* wave, not two discrete sine waves. So exactly what in the sound is beating? We have learned that in order to initiate a beating effect we need to hear *two* or more frequencies that are very close to one another. Here we are only hearing *one* sound. Or are we?

Technically, we are. But the reality is that this one sound, being a complex wave, is composed of many different frequencies (the overtones), blended together. This explains why we can hear beating *within* a single complex sound - we are actually hearing the overtones beating against one another!

A frequency ratio of 1.00:1.00 generates the following set of harmonic overtones, resulting in a sawtooth waveshape. **(see figure 6-24)**

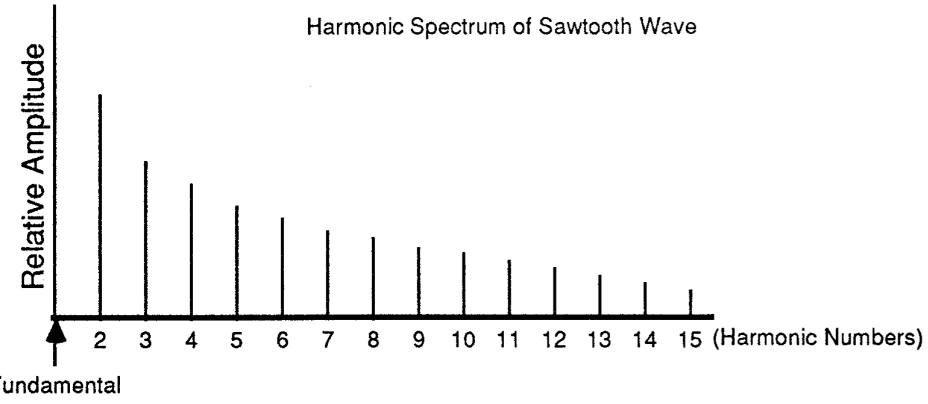


Figure 6-24

Slightly shifting the tuning of one of the two operators, resulting in a frequency ratio of 1.01:1.00, generates the same table of harmonic overtones plus a few inharmonic overtones which are *extremely close in frequency* to the existing harmonics. **(see figure 6-25)**

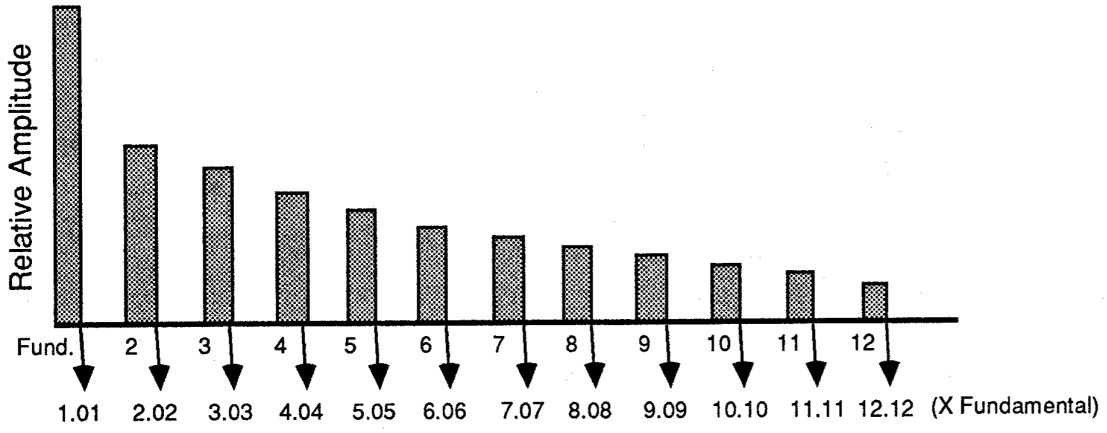


Figure 6-25

This causes a beating effect *within* the single complex sound! By using the Detune parameter, we can bring the frequency ratio even closer than 1.01:1.00, say to 1.001 : 1.000. This will result in the same effect, except that the inharmonics generated will be even closer to the harmonics than that in the previous example, hence a *slower* beating. **(see figure 6-26)**

Let's try it:

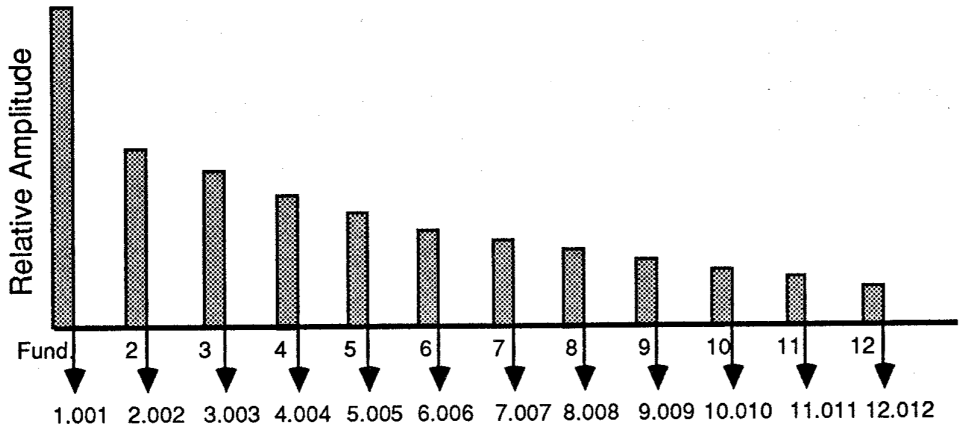


Figure 6-26

Exercise 23
Use of the Detune control within a system

1) INITIALIZE your DX7II from single voice play mode and leave it in algorithm #1. TURN OFF operators 3 through 6 ("110000") and GENERATE a sawtooth wave (refer to Exercise 18 if you don't remember how to do this).

2) Press edit switch 8 followed by edit switch 2 in order to VIEW the Detune parameter for operator 2.

3) Use the cursor switch in order to position the cursor over the Detune parameter and use the data entry slider in order to change this value from its default of 0 to its maximum value of +7.

4) Listen (Audio Cue 23A). Note the speed of the beating. Note also that this sound is very harsh. To make the sound warmer,

5) Press edit switch 10 in order to VIEW the Output Level parameter for operator 2. Change this value to 90. Listen (Audio Cue 23B).

6) Press edit switch 8 in order to go back to the Detune parameter for operator 2. Hold down a note on the keyboard, and, using the "no" button, slowly CHANGE this value back to the center point of 0, listening as you do so (Audio Cue 23C). Note that the beating effect slows down and, at the setting of 0, disappears altogether (at this setting, the frequency ratio is once again exactly 1.00:1.00, and only harmonic overtones are present).

7) Continue holding down the note and continue using the "no" button to change this value downward to its minimum setting of -7, listening as you do so (Audio Cue 23D). Note that there is no perceptible difference in the effect as from when you *increased* the Detune value up to +7. That's because the only thing that matters here is the *relative* offset, not the absolute values. Similarly, altering the Detune value for operator 1 instead of operator 2 will have the same effect. Experiment and try it!

Detuning, then, provides us with one of the most powerful *animating* techniques on the DX7II, and I should explain what I mean by this term.

Digital synthesizers have long had the largely undeserved stigma of being able to only produce sounds which are considered "cold" or "sterile". To a certain degree this is true because, as discussed in Chapter One, they produce sounds which theoretically are distortion-free. In reality, this is not actually the case. as every owner of a digital synthesizer already knows. Because of its higher-speed processing, the DX7II is much cleaner and has much higher fidelity than the original DX7. However, you will nonetheless occasionally hear some hiss or "fizz" emanating from it when you call up certain voices. This unwanted sound is called *aliasing* noise, and is essentially the result of the DAC not quite always being able to keep up with the millions of numbers per second that the microprocessor is feeding it. These "left-over" numbers are simply discarded and the aural result is a small amount of very high frequency noise. A standard analog *low-pass filter* is usually all that is necessary to remove this unwanted sound.

Because acoustic instruments and acoustic sounds all have small degrees of random distortions present, the manufacturers of digital synthesizers have had to provide us with various means of digitally simulating this randomness. The various techniques and tools available for accomplishing this on the DX7II are numerous, and I like to refer to them as "animation" techniques, since they help to animate, or humanize, the sounds we generate. Inducing beating effects, whether by beating a carrier against another carrier (as we did in the last chapter), or by beating a modulator against a carrier (or vice versa) is one of the most important animation techniques available on this instrument. My general advice is this: once you've done just about everything else you want to in creating a new sound, try some detuning effects. It's icing on the cake, but more often than not, it will add a dimensionality to your sound that makes it far more "real". We will discover as we examine the Yamaha presets, for example, that there are very few of them which do not use beating effects to some degree.

Just as we were able to beat two carriers producing simple sine waves against one another, so, too, can we beat two carriers producing complex waves against one another. This involves shifting the pitch of one of two complex waves, and results in a very beautiful and pleasing effect:

Exercise 24
Creating beating effects with complex timbres

1) INITIALIZE your DX7II from single voice play mode, leave it in algorithm #1, and TURN OFF operators 3 through 6 ("110000"). GENERATE a *square* wave using the system of operators 1 and 2 (refer to Exercise 19 if you don't remember how to do this).

2) Listen to confirm (Audio Cue 24A). 3) TURN OFF operators 1 and 2 and TURN ON operators 3 and 4 ("001100"). Observe the diagram for algorithm #1 and note that in this algorithm, operator 3 is a carrier being modulated by operator 4 (don't worry about what operators 5 and 6 are doing, since we've got them switched off anyway).

4) Using the system of operators 3 and 4, GENERATE another square wave at maximum output (operator 3 Level = 99)and listen to confirm (Audio Cue 24B). Note that there is no difference whatsoever between this square wave and the one you previously made with operators 1 and 2. Selectively TURN ON and OFF each of the two systems to confirm this.

5) Keeping all four operators ON ("111100"), press edit switch 8 followed by edit switch 3 in order to VIEW the Detune parameter for operator 3. Use the cursor switch to position the cursor over the Detune parameter and change the value to +7. Remember that in order to initiate a *pitch* change to our sound, we must also change operator 4 in the same way!! Bearing this in mind, change the Detune value for operator 4 also to +7.

6) Play a few notes on the keyboard and listen (Audio Cue 24C). This is an extremely beautiful effect!!

7) Let's take things a step further and add in yet another detuned square wave. Note that algorithm #1 will not permit us to accomplish this since neither of our remaining two operators (operators 5 and 6) are being used as carriers. Remember that you can *change algorithms at any time,* however! Examine the algorithm diagrams and note that both algorithms #5 and #6 have configurations where operators 5 and 6 act as a modulator-carrier system, and where operators 1, 2, 3, and 4 are still configured in the same way as they currently are in algorithm #1. Either algorithm #5 or algorithm #6, then, will allow us to add in another detuned square wave without affecting what we've already generated. With this in mind,

8) Press edit switch 7 (ALGORITHM) and change to algorithm #5 or #6. TURN OFF operators 1 through 4 and TURN ON operators 5 and 6 ("000011"). Using this system, GENERATE another square wave at maximum output (operator 5 Level = 99). Listen to confirm (Audio Cue 24D). Note that, again, there is no difference at all in the quality of the square wave generated by these two operators from that of the other operators.

9) VIEW the Detune parameter for operator 5 and change this value from its default of 0 to a new value of -3. Again, in order to initiate a pitch change, we will have to change operator 6 in precisely the same manner. VIEW the Detune parameter for operator 6 and change it also to -3.

10) TURN ON all operators ("111111") and listen (Audio Cue 24E). Very impressive chorusing, if I say so myself!

11) Experiment by now offsetting various carriers against their respective modulators - for example, change the Detune parameter of operator 1 to +5 (Audio Cue 24F) or of operator 4 to -6 (Audio Cue 24G). Note that as you thereby induce beatings *within* the three square waves, the sound gets even richer and begins to take on a stronger type of movement.

12) Experiment further by repeating the above exercise with different complex timbres, balancing the qualitative and quantitative controls to achieve unusual musical timbres. Also try the same with non-musical timbres generated from non-whole number frequency ratios. Finally, try it yet again with Fixed Frequency complex timbres. For example, to generate a square wave at a fixed frequency of 436.5 Hz, set your carrier to a fixed frequency value of 436.5 Hz and your modulator to a fixed frequency of 873.0 Hz (or as close as you can get), thereby maintaining a frequency ratio of 2:1! Adjust the modulator output level to 71 and you will have a square wave whose pitch doesn't change as you play different notes on the keyboard. The same detuning and beating effects can be incurred with fixed-frequency complex waves as with non-fixed timbres. Try it!

We can generate other extremely interesting and beautiful animation effects by sending a modulator's output signal to two slightly detuned carriers, or by sending two slightly detuned modulators' output signal to a single carrier. Algorithm #20 happens to conveniently provide us with both configurations in one package. **(see figure 6-27)**

The first system gives us two carriers (operators 1 and 2) which are both being modulated from the same source, operator 3. The second system gives us two modulators (operators 5 and 6) which are both equally affecting a single carrier (operator 4). Let's find out what we get by detuning operator 2 slightly relative to operator 1; and follow that by slightly detuning operator 6 relative to operator 5.

Algorithm #20:

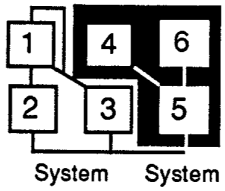


Figure 6-27

Exercise 25
Detuning with algorithm #20

1) INITIALIZE your DX7II from single play mode and, using edit switch 7, select algorithm #20.

2) TURN OFF operators 2, 4, 5, and 6. ("101000")

3) Using operators 1 and 3, GENERATE a sawtooth wave (refer to Exercise 18 if you can't remember how). Listen to confirm (Audio Cue 25A).

4) TURN OFF operator 1, and TURN ON operator 2. ("011000")

5) Using operators 2 and 3, GENERATE another sawtooth wave at maximum output (operator 2 Level = 99). Listen to confirm (Audio Cue 25B). Note that it sounds exactly the same as the one generated from the system of operators 1 and 3.

6) Now TURN ON operator 1 again ("111000") and listen (Audio Cue 25C). What you hearing is two sawtooth waves, perfectly in tune with one another, so they reinforce each other at every point and the sound is simply a bit louder. **(see figure 6-28)**

7) Press edit switch 8, followed by edit switch 2, in order to VIEW the DETUNE parameter for operator 2. Use the "yes" button to change this value to +7. Listen to the unusual beating effect (Audio Cue 25D). This is being caused because we have slightly altered the frequency ratio between operators 3 and 2, such that it is no longer 1.00:1.00 but is in fact closer to 1.00:1.008. This means that not only is there beating occurring within the wave itself (as the harmonic overtones beat

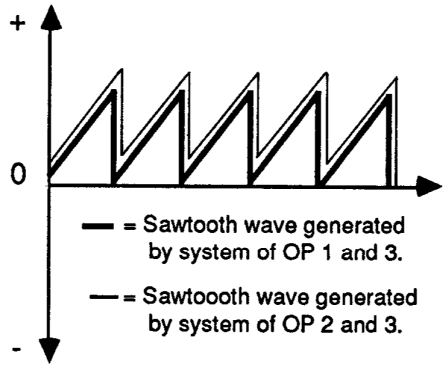


Figure 6-28

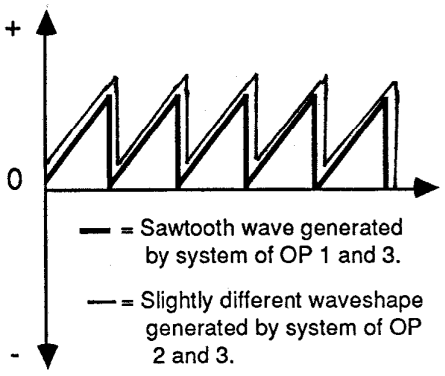


Figure 6-29

against their close inharmonic cousins) but that the overall wave generated by the combination of operators 2 and 3 is beating against the pure sawtooth wave generated by the combination of operators 1 and 3. The aural result is a very deep phase cancellation effect, as we hear these two waves periodically reinforcing and cancelling one another. The speed of this movement is determined by how closely detuned our "renegade" wave (operators 2 and 3) is. **(see figure 6-29)**

8) Continue to hold a note down and, using the "no" button, change the Detune value slowly back down to 0 and then through the negative values all the way to -7. Listen as you do so (Audio Cue 25E) and note that, as you get closer to the 0 center point, the speed of the beating slows down. Now let's listen to the other system in this algorithm:

9) TURN OFF operators 1, 2, and 3, and TURN ON operators 4 and 5. ("000110")

10) Using operators 4 and 5, GENERATE another sawtooth wave at maximum output (operator 4 Level = 99) and listen to confirm (Audio Cue 25F).

11) TURN OFF operator 5 and TURN ON operator 6. ("000101")

12) Using operators 4 and 6, GENERATE yet another sawtooth wave and listen to confirm (Audio Cue 25G).

13) Now TURN ON operator 5 once again ("000111") and listen to the composite result (Audio Cue 25H). Because operator 4 (our carrier) is being modulated to the maximum degree by two separate modulators (operators 5 and 6) the sound will be unpleasantly overbright and harsh, verging on distortion (!). We can make this much more listenable by using our front panel to reduce these modulator's output levels.

14) Press edit switch 10, followed by edit switch 5, in order to VIEW the Output Level parameter for operator 5 and CHANGE the value to 85. Now VIEW the same parameter for operator 6 and CHANGE it also to a value of 85. Listen (Audio Cue 25I) and note that our sound is much warmer (since lowering the output level of a modulator will always reduce the number of overtones).

15) Now VIEW the Detune parameter for operator 6 and change this value to +7. Listen to this very unusual effect (Audio Cue 25J) and note that it is very different from the effect we heard previously from the other system in our algorithm. What we are doing here is setting up a different frequency ratio for operators 4 and 6 than the perfect 1:1 ratio which exists for operators 4 and 5. This new frequency ratio is in fact closer to 1.008:1.00 and will cause a wave to be generated which has internal beatings due to the inharmonic overtones being very close in frequency to the harmonic overtones. When this already moving wave is output alongside a pure sawtooth wave, the movements intensify and in fact simulate a "wah-wah" type effect as we hear overtones going in and out of phase with one another. **(see figure 6-30)**

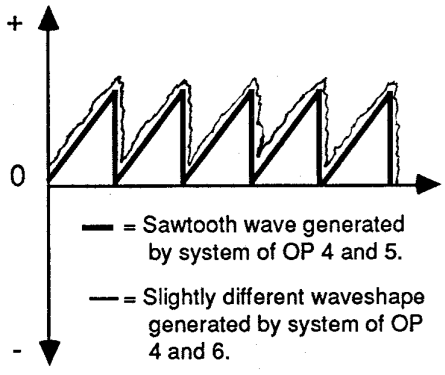


Figure 6-30

16) Continue to hold down a key and change the Detune parameter slowly back to 0 and through its negative settings, down to a minimum of -7. Listen as you do so (Audio Cue 25K) and note that the speed of the beating effect slows as you bring it nearer to the center point of 0.

17) Experiment with timbres other than the sawtooth wave, which was picked here purely as a time-saving convenience. Also try experimenting with altering detuning settings within algorithm #16, #17, or #18 (each of which offer *three* modulators sending output into a single carrier) and algorithm #22 or #24 (which offer three *carriers*, all receiving modulation data from a *single* modulator!).

We have seen that changing the Detune setting alters the speed of the effect, but how would we alter the *depth* of the effect without changing the speed? The answer is straightforward: simply lower the output level of the carrier producing the "renegade" wave. If there is less actual audio signal coming from this wave, then it will beat less severely against the unaltered wave. Try it! Redo the above exercise, and lower the output level of operator 2 in the first system, and operator 6 in the second system, in order to lessen the overall effect, with no change in speed.

Being able to conjure up these phase cancellation or "wah-wah" effects on the DX7II greatly expands the on-board power of the machine, but, perhaps more importantly, offers us yet another means of reducing the short list of available algorithms for a particular sound you have in mind. For example, if you feel strongly you will need a "wah-wah" type of effect, you will need an algorithm that provides you with at least two modulators feeding into a single carrier. Of course, these *one-into-many* or *many-into-one* systems are commonly used for many other effects as well - in combination with the *EGs* (to be covered in Chapter Nine) and *keyboard level scaling* (to be covered in Chapter Twelve). If, on the other hand, you know that you will need the sound of several complex timbres beating against (and not just within) one another, than you will need two or more carriers, each with its own attached modulator (something like what is provided by algorithm #5, which is in fact one of the most commonly used algorithms). Asking yourself these types of questions, along with the "how many carriers do I need?" question mentioned previously, will help you to ultimately decide the best algorithm to use for a particular situation.

Let's suppose, for example, that we wanted to make a sound that consisted of a sawtooth wave, a square wave, and two detuned sine waves. What algorithm would we choose for this? At first glance, it may seem as if algorithms #21, #22, #23, #29, or even #30 might work. **(see figure 6-31)**

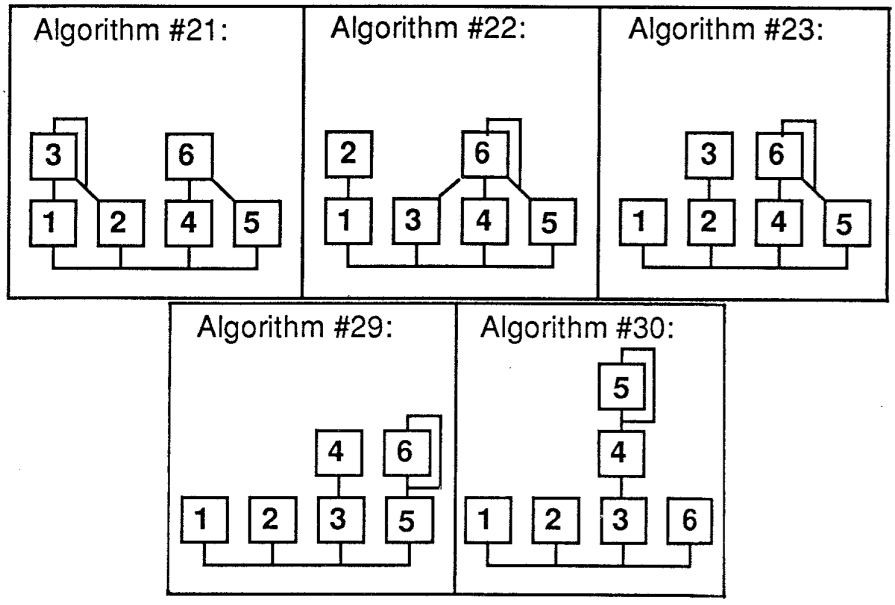


Figure 6-31

A closer look at these diagrams, however, should show you that in this particular example, only algorithm #29 will suffice. Algorithm #30 has the proper number of carriers, but only one of them (operator 3) has any modulators attached. Algorithms #21, #22, and #23 disqualify because they all have modulators feeding more than one carrier and there is no way we can direct a modulator to send its signal one place and not another. This is a purposely simple example, and often your choice will not be so straightforward, but this will give you an idea of the process involved.

Another very powerful animating technique involves the use of Fixed Frequency mode with complex timbres. We have had some experience with this already in Exercise 25, where you were asked to experiment by generating fixed-frequency complex timbres and then slightly alter their tunings so as to produce beating effects. However, we should devote some time now to more closely examining the effects of fixed frequency on such complex timbres, specifically those effects generated by *offsets*, that is, placing only one of the operators within a system into this mode while leaving the other operator in its normal ("ratio") mode.

First of all, what kind of effect will we get if we place a modulator in fixed frequency while leaving the carrier in ratio mode? Surprisingly, two different ones, depending upon whether the modulator is in a *sub-audio* or *audible range* frequency. If the modulator is outputting a sub-audio frequency, then the effect will be exactly similar to that of feeding an analog *LFO (low frequency oscillator)* controlling voltage into an audio oscillator. For those of you unfamiliar with analog terminology, we are saying in plain English that we will hear slow, repetitive pitch change from this configuration. In the case of the DX7II, since the un-modulated modulator is only capable of generating a sine wave, this periodic pitch change will be smooth, back and forth, in an up and down direction - and it will *not* change speed over the keyboard (since the modulator, in fixed frequency mode, is ignoring the keyboard). In other words, if you play "middle A" on the keyboard, you will hear the pitch slowly rise above middle A and then slowly fall below middle A. **(see figure 6-32)**

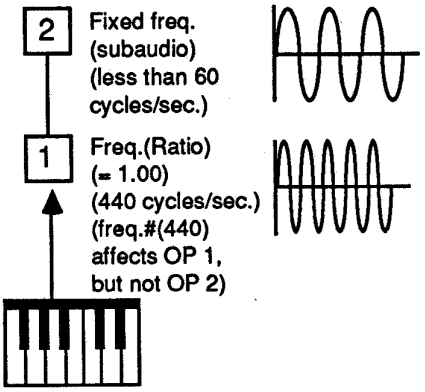


Figure 6-32

Let's try it:

Exercise 26
Generating repetitive pitch change with modulators in sub-audio fixed-frequency mode

1) INITIALIZE your DX7II from single voice play mode, leave it in algorithm #1, TURN OFF operators 3 through 6 ("110000") and, using the system of operators 1 and 2, GENERATE a sawtooth wave (refer to Exercise 18 if necessary).

2) Press edit switch 2 in order to VIEW the Mode of operator 2 and use the cursor switches to position the cursor over the Mode parameter.

3) Use the "yes" button to CHANGE this from the default of "ratio" to a new value of "fixed".

4) Press the right cursor switch once in order to position the cursor over the Coarse parameter. Change this value to 1.000 Hz.

5) Play a note on the keyboard and listen (Audio Cue 26A). You should be hearing a single sine wave, changing pitch once per second (since operator 2 is at a frequency of 1.000 Hz). Continue holding this note down for the remainder of this Exercise.

6) Press the right cursor switch once more in order to position the cursor over the Fine parameter and use the data entry slider in order to slowly increase this value to its maximum of 9.772 Hz, listening as you do so (Audio Cue 26B). Note that as you increase this value, the speed of the pitch change increases.

7) Return the Fine value to 1.000 Hz and press the left cursor switch in order to position the cursor over the Coarse parameter again. Press the "yes" switch once in order to change this value to the sub-audio frequency of 10.00 Hz.

8) Press the right cursor switch to move the cursor once again over the Fine parameter and use the data entry slider to slowly change this value up to its maximum of 97.72 Hz, listening as you do so (Audio Cue 26C). Note that as the frequency of operator 2 approaches the audio range (approximately 20 Hz), the repetitive change in pitch happens so quickly that it cannot be perceived, and instead we begin to hear dissonant overtones.

Of course, if we continue in this manner, we will eventually have operator 2 well up into the audible range and different things will start to happen. As we started to hear in step 8 above, when the modulator goes into the faster audible range, inharmonic overtones are generated and the sound becomes dissonant and non-musical. Playing a series of different notes on the keyboard will now produce *different timbres per note*. This is because each new note on the keyboard generates a different *frequency ratio:* **(see figure 6-33)**

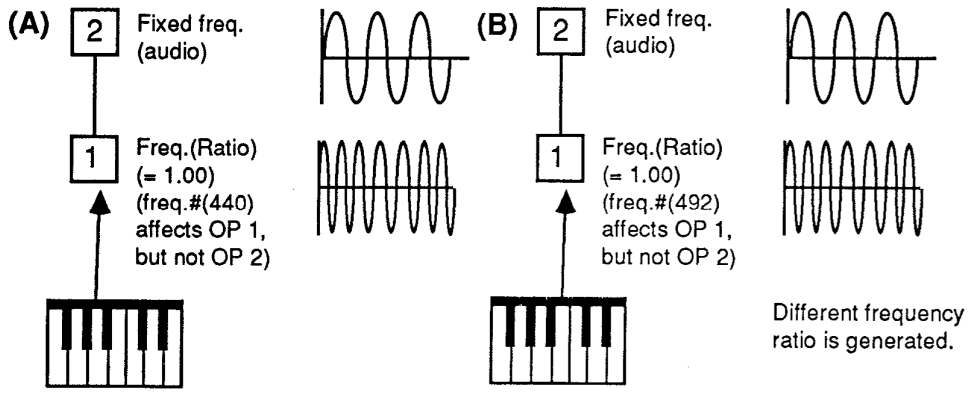


Figure 6-33

The effect generated here is very similar to that of an analog device called a *ring modulator*. This is a component that *adds* frequencies together and also *subtracts* frequencies from one another. In analog synthesizers, they are generally used to create dissonant, ringing effects, by inputting two audio signals of different frequencies (normally derived from two analog oscillators). If, for example, you feed in one wave at 300 Hz, and another at 440 Hz, the ring modulator will output a signal of 740 Hz (the sum of 300 and 440) and another of 140 Hz (the difference between 440 and 300). Neither 740 Hz nor 140 Hz have any whole-number mathematical relationship to the starting frequencies of 300 Hz or 440 Hz, and so the output signal is dissonant. Furthermore, if the input signals are complex timbres and not just sine waves, the overtones in the sound will also sum and difference and we would hear a complex output containing many inharmonic overtones. It should be easy to see why setting up constantly different non-whole number

frequency ratios on the DX7II will yield a similar dissonant effect. Try redoing Exercise 27 above, but this time put operator 2 in the audible Coarse ranges of first 100 Hz and then 1000 Hz; and then try changing the Fine parameter as before.

But what if we put the *carrier* into a fixed-frequency mode? Again, the results will be different, depending upon whether it is in an audible or sub-audio frequency. Let's try it in an audible range first. As you will see, the result is virtually the same as having the modulator in an audible fixed frequency, as we are once again setting up different frequency ratios per note. The result is the same: a dissonant, ring modulator-like effect.

Exercise 27
Generating dissonant effects by using a carrier in an audible fixed-frequency mode

1) INITIALIZE your DX7II from single voice play mode, leave it in algorithm #1, TURN OFF operators 3 through 6 ("110000"), and, using the system of operators 1 and 2, GENERATE a sawtooth wave.

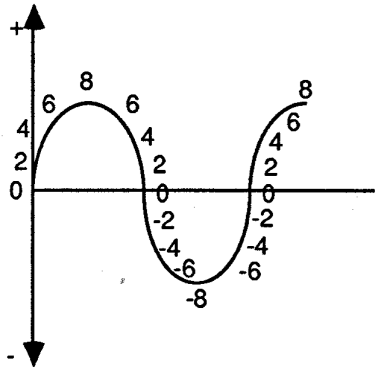
2) Press edit switch 8, followed by edit switch 1, in order to VIEW the Mode parameter for operator 1. Use the cursor switch to position the cursor over this parameter and press the "yes" button in order to change this value for operator 1 from its default of "ratio" to "fixed".

3) Press the right cursor switch in order to position the cursor over the Coarse parameter and change this value to 100.0 Hz (which is in the audible range).

4) Play a scale on the keyboard and listen (Audio Cue 27A). Note that each note played produces a different dissonant timbre.

5) Press the right cursor switch again in order to position the cursor over the Fine parameter and use the data entry slider to change this to a value of 436.5 Hz. Play a scale on the keyboard and listen (Audio Cue 27B). Note that, again, each note played produces a different dissonant timbre, dissimilar from that which you heard in step 4 above.

6) Experiment by setting operator 1 to different Coarse and Fine values, taking care to keep it in the audible range (between 20 Hz and 20,000 Hz). For now, *don't* go into sub-audio values.



Finally, what will happen if we put the carrier (operator 1 in this instance) into a sub-audio fixed frequency? The logical answer to this question might well be that we wouldn't hear *anything*, since the carrier is actually providing the audio signal, and since a sub-audio signal is by definition outside the range of human hearing. This may be the logical answer, but it is not the correct one.

In fact, what we will hear is one of the most beautiful animation techniques available on this instrument. The explanation for this comes from the fact that the digital oscillators inside the operators are in fact *number generators* (see Chapter Three if you don't remember this). In order to generate a sine wave, these oscillators must produce numbers which are both positive and negative. **(see figure 6-34)**

If we set the carrier to a sub-audio fixed frequency, then it is very slowly going to generate the same positive and negative numbers, over and over again, at a constant speed. Our modulator, which we will leave in an audible range, and in "ratio" mode, is sending into the carrier a very high-speed stream of numbers, hundreds or even thousands of times per second (depending on the key played). **(see figure 6-35)**

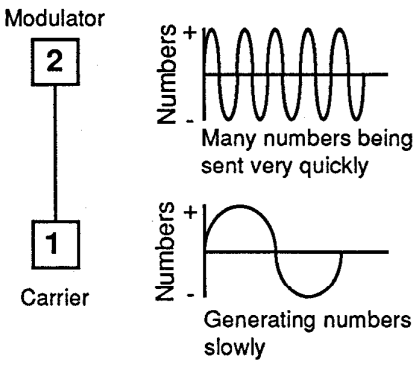


Figure 6-35

The end result is that the carrier slowly *accepts* and then *rejects* this stream of numbers, resulting in a complex timbre being generated, followed by the mirror-image of this timbre! **(see figure 6-36)**

We literally hear a complex timbre, followed by the exact inverse of this timbre, slowly and gently changing *phase* . The end result is a very beautiful movement in the sound, not dissimilar to that created by Detuning a modulator relative to a carrier (or vice-versa). The big difference here is that, unlike detuning effects, the speed of this movement WON'T change as we play different keys on the keyboard, since the carrier (in fixed mode) is ignoring the keyboard! Let's try it:

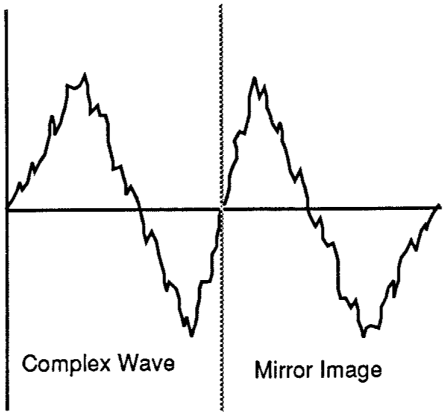


Figure 6-36

Exercise 28
Animating a sound by using a carrier in sub-audio fixed frequency

1) INITIALIZE your DX7II from single voice play mode, leave it in algorithm #1, TURN OFF operators 3 through 6 ("110000"), and, using the system of operators 1 and 2, GENERATE a sawtooth wave.

2) Press edit switch 8 followed by edit switch 1 in order to VIEW the Mode parameter for operator 1. Use the cursor switches to position the cursor over the Mode parameter and press the "yes" button in order to change this value from the default of "ratio" to "fixed".

3) Press the right cursor switch in order to position the cursor over the Coarse parameter and press the "no" button in order to change this value from 10.00 Hz to 1.00 Hz.

4) Play a note on the keyboard and listen (Audio Cue 28A). Note that the sound changes timbre once per second, since the frequency of operator 1 is exactly 1.00 Hz. Note also that the sound, being a sawtooth wave, is quite harsh. To make the sound warmer,

5) Change the Output Level (edit switch 10) for operator 2 to a value of 85. Play a note on the keyboard and listen (Audio Cue 28B) to confirm that the sound is less harsh.

6) VIEW the Fine parameter for operator 1 and, using the cursor switches and data entry slider, slowly change this upwards to its maximum of 9.772 Hz, holding a note down on the keyboard and listening as you do so (Audio Cue 28C). Note that as this value increases, the speed of the effect increases.

7) Experiment by creating different timbres and applying this effect to them.

To summarize, then, we have covered six different animation techniques, all of which help to add movement and "human-ness" to the sound, but all of which have qualitatively different effects:

a) Detune a sine wave (carrier only) relative to another sine wave (resulting in amplitude beating within the sound).

b) Detune a modulator relative to its carrier or a carrier relative to its modulator (resulting in timbral beating within the sound).

c) Detune a complex wave (generated by a system) relative to another complex wave (resulting in both amplitude and timbral beating). This, you will remember, involves changing the frequency of all operators in one system *in the same way,* in order to keep the frequency ratios within that system intact.

d) Detune one carrier relative to another, with both receiving modulation data from the *same* modulator (as with the first system in algorithm #20). This produces a characteristic deep phase cancellation.

e) Detune one modulator relative to another, with both feeding modulation data into the *same* carrier (as with the second system in algorithm #20). This produces a characteristic "wah-wah" type effect.

f) Put a carrier within a system into a sub-audio fixed frequency. This produces a characteristic subtle timbral beating, but one that remains at a constant speed no matter which notes you play. This is probably the most subtle of all the techniques covered here. In all instances except the last one, the depth of the effect can be lessened by decreasing the output level of one of the affected operators. In the last instance, you will probably want to use this effect on one system within an algorithm that offers you another system which can be set up similarly - but without the sub-audio mode. This will allow the altered system to "bounce off" an un-affected system, thereby increasing the overall effect. In that case, lowering the output level of the sub-audio carrier will induce a lessening in the depth of the effect.

Modifying presets

Congratulations to those of you who have faithfully gotten this far, initializing away and constantly creating sounds from scratch. As mentioned earlier, initializing is not a procedure you will find yourselves doing often in practice. Most of the time you will create the sounds you require by *modifying* a pre-existing sound. We now have amassed enough information to try this procedure for the first time!

When you call up a preset sound, either from the internal memory or from some outboard memory such as a cartridge, the DX7II automatically copies this data into your edit buffer. How can we view and/or change this data? Simple - we need to press only one button, and that button is the edit mode select switch.

Possibly the most remarkable thing about the DX7II is that it is absolutely incapable of keeping a secret! In fact, we can think of this instrument as being the Joan Rivers of the synthesizer world. There is also no way to protect a preset from being changed. Companies which sell sounds for the DX7II do so at great risk, because, first of all, once you get the sound, you can find out exactly how it was made, and, second of all, you can copy the data to your heart's content: to other memory storage units for your DX7II (be they cartridge or disk) and to other DX7IIs. There is absolutely nothing the author of a sound can do to prevent that. Copyright laws in this country, do, however, normally prohibit you from using or reselling a copyrighted sound without giving credit and paying a royalty. The legal entanglements of this are complicated and well beyond the scope of this book - just bear in mind that the DX7II *always* allows you to see and learn from the programming work of others!

Let's take a look at a common preset, and experiment with making some useful changes to it. Go into single play mode and call up the "Celeste" preset, which will be in your ROM cartridge, bank 2, voice 62. Now press the green "edit" *mode select* switch, and let's see what we have:

First of all, your DX7II has just called up the most recent edit display you looked at. Since I have no idea what that is, press edit switch 7 so we can start by looking at the same display. **(see figure 6-37)**

Figure 6-37

Right away, we see a lot of information about this sound: the algorithm being used (#5), the fact that Oscillator Sync is ON, and the name of the voice "Celeste". There is also something something called "Fbl" and a parameter called "Transpose", but be patient - we'll get to these in Chapters Seven and Twelve, respectively). Let's start by taking a look at what frequency ratios were used in each of the three systems available in this algorithm. Press edit switch 8 and a quick cycle through the six operators (accomplished by using the operator select switches, 1 through 6) will show you that operators 1 through 4 are in "ratio" mode, while both operators 5 and 6 are in "fixed" mode. In just a moment, we'll listen to the individual systems and see just why they are set up this way. The Coarse/Fine parameter shows you which actual ratios were used. Use the operator select switches to view the values for each operator, and you will see the following ratios. **(see figure 38)**

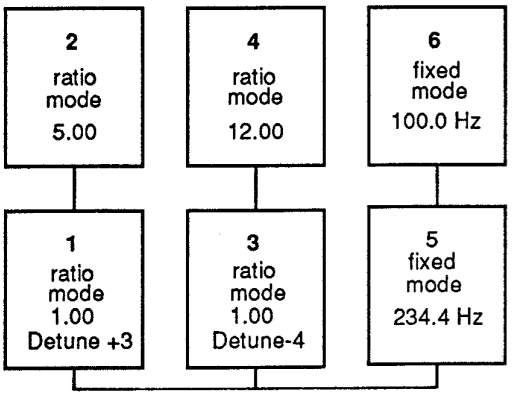


Figure 6-38

Now let's see what the output levels were for each operator. Press edit switch 10 to view this parameter, and again use the operator select switches to view this value for each operator. **(see figure 6-39)**

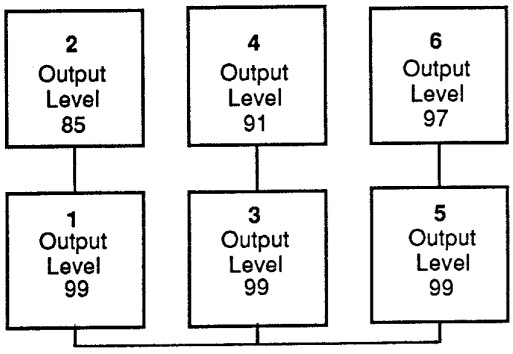


Figure 6-39

Of course, there are many factors besides these which make this sound what it is, but we will limit ourselves here to only those parameters we have already covered. So let's say you have an incorrigible urge to *change* the sound a bit - no problem! The edit buffer ensures that we can change away, without actually affecting the sound itself. Let's begin with the crudest change imaginable - changing the algorithm. Remember that you can change the algorithm at any time, and that when you do so, *no other data is changed.* So, for example, if you're curious to hear "Celeste" in algorithm #16, press edit switch 7,

use the cursor switch to position the cursor over the Algorithm parameter, and use the data entry slider to change this value to 16. Play a note on the keyboard and listen - you should be hearing a really horrible noise which doesn't sound anything like "Celeste". Try going into different algorithms and note that many of them produce enormous changes to the overall sound.

Changing the algorithm is probably the grossest change you can make to a sound. Doing this is a fairly mindless exercise and will rarely be of logical help to you in modifying a sound (of course, there is nothing inherently wrong with mindless exercises, after all 8 million MTV viewers can't be wrong! Seriously, though, often just mindlessly changing algorithms will yield new and interesting sounds and I don't mean to discourage you from doing it). However, more often, changes to more subtle parameters will yield just the kind of modifications to the sound you require.

There is a particular procedure that I recommend you follow when modifying a preset, and it goes like this:

a) Identify the algorithm being used (by pressing edit switch 7) and look up the diagram on the front panel. Take note of which operators are being used as carriers, which as modulators, and, particularly, how many *systems* are in this algorithm.

b) By using the operator on-off switches (edit switches 17 through 22), *listen to each system individually* and get a "feel" for what that system is contributing to the overall sound. If a system contains more than one modulator or more than one carrier, selectively turn them on and off in order to hear the contribution of each within that system and within the overall sound.

c) After you have listened to each system individually, *think* for a moment about precisely what it is you want to change. Remember - there are only three choices, as every alteration you make to a sound will of necessity be a change to either volume, timbre, or pitch. Cardinal Rules one, two, and three will then tell you precisely which controls you need to use in order to make the desired change.

d) In everything you do, *USE YOUR EARS!* Don't enter in data without constantly listening to the effect that change enacts to the sound. Depending upon what you are doing, you may at any given time want to listen to the total sound, or to just a system within the total sound. Remember also to keep checking the operator number in the upper left-hand corner so you don't inadvertently make the right alteration to the wrong operator.

e) Know when to stop. The mark of a good programmer is knowing when to be satisfied. Continually tweaking even after you've done what you set out to do will lead to an inevitable feeling of frustration - especially if you are in a situation when the dollars are ticking away. This does NOT mean that you should compromise with a half-baked solution to the problem you set out to conquer, either - but know where to draw the line.

Let's try this procedure out with our "Celeste" sound, and give ourselves the task of turning this pleasing bell-like sound into a dissonant, metallic steel drum sound. In doing so, we will be able to solve the mystery of why operators 5 and 6 are being used in fixed frequency mode, and also be able to hear for ourselves why these particular frequency ratios and output levels were chosen when the sound was originally created.

Exercise 29
Changing a celeste into a steel drum

1) Call up the "Celeste" sound from your ROM cartridge. You will find it in bank 2, voice # 62. (If you can't remember how to do this, refer to Exercise 2 in Chapter 2)

2) Press the green edit mode select switch, followed by edit switch 7. Observe that algorithm #5 is being used for this sound. Observe the algorithm #5 diagram on the front panel of the machine. **(see figure 6-40)**

3) Press edit switch 1, followed by edit switch 8 in order to view the mode and frequency values for operator 1. Press edit switches 19 through 22 in order to TURN OFF operators 3 through 6 ("110000"), enabling us to listen to the system of operators 1 and 2 alone (Audio Cue 29A). Note that the sound of this system is a gentle, bell-like tone. Press edit switch 2 in order to VIEW the frequency ratio in use here (5:1).

4) Press edit switches 17 and 18 in order to TURN OFF operators 1 and 2 and press edit switches 19 and 20 in order to TURN ON operators 3 and 4 ("001100"). Play a few notes and listen (Audio Cue 29B). Note that this system generates a rather bright, almost piercing tone, more like a chime than a bell. Press edit switches 3 and 4 in order to VIEW the frequency ratio in use here (12:1). The fact that the modulator in this system is traveling so much faster than the modulator in the first system explains why this is a brighter sound - since the higher frequency ratio will yield higher overtones.

5) Press edit switches 19 and 20 in order to TURN OFF operators 3 and 4 and press edit switches 21 and 22 in order to TURN ON operators 5 and 6 ("000011"). Play a few notes and listen (Audio Cue 29C). Note that this system sounds like a click - and that this click does not change pitch as you play different notes on the keyboard. This is a very common effect in *component* voicings like this (where there are more than one system in the algorithm). This click is meant to simulate the sound of the celeste's wooden hammer striking the string. By altering the output level of this system's carrier (operator 5), we can directly control just how much click is heard in the total sound. Press edit switches 5 and 6 in order to view the frequency ratio being used here (100.0 Hz : 234.4 Hz). Because the carrier is traveling at a higher speed than the modulator, this click has a relatively narrow bandwidth and resultant slight nasal quality.

6) Now that we've heard the contributions of each of the three systems, **(see figure 6-41)** we're ready to begin changing this celeste into a steel drum. First of all, we need to decide if the change we want to make will be a volume, pitch, or timbral one. Since the steel drum has a significantly different timbre than the celeste, the answer is that our change must be essentially a timbral change. How does a steel drum differ timbrally from a celeste? For one thing, it is much more dissonant - that is, inharmonic - in nature. It is also not nearly as bright as a celeste nor nearly as rich in high overtones. We will therefore want to make the frequency ratios in our systems lower *non-whole-number* ratios, and will very possibly want to reduce our modulator output levels as well. Accordingly, TURN OFF operators 3 through 6 and TURN ON operators 1 and 2 ("110000") so we can hear the sound of the first system alone. Press edit switch 2 in order to

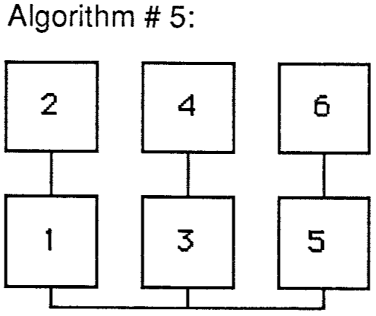


Figure 6-40

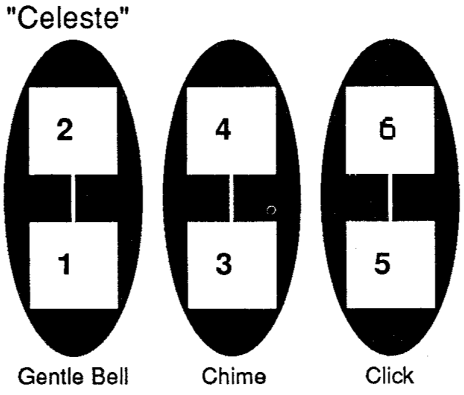


Figure 6-41

VIEW operator 2 and use the cursor switch in order to position the cursor over the Coarse parameter. Use the data entry slider or the "no" button in order to change this value to 2.00. Press the right cursor switch once in order to position the cursor over the Fine value and use the data entry slider in order to change this to a new value of 3.54.* Press edit switch 10 and position the cursor over the Level parameter. Change this to a new value of 81 in order to generate a slightly warmer sound (since less modulator output level will always result in fewer overtones). Play a few keys on the keyboard and listen (Audio Cue 29D). Note that the sound is now considerably more dissonant and metallic-sounding.

7) TURN OFF operators 1 and 2 (we're done with that system for now) and TURN ON operators 3 and 4 ("001100"). Using the same procedures outlined in step 6 above, change the ratio number for operator 3 to a new value of 1.02, and the ratio number for operator 4 to a new value of 2.28. By doing so, we not only enter in a new frequency ratio, but we make the pitch of this entire system a bit sharp relative to the first system, thus adding to the dissonance. Leave the output levels of both of these operators at their current values (99 for operator 3 and 91 for operator 4). Play a few keys on the keyboard and listen (Audio Cue 29E). Note that the sound of this system alone is also bell-like, but less bright and also a bit sharp, as compared with the first system. TURN ON operators 1 and 2 ("111100") in order to listen to both systems together (Audio Cue 29F) and note the movement in the sound, due to the Detuning values and, more importantly, due to the Fine pitch differential between the two carriers (operator 1 is at 1.00 while operator 3 is at 1.02).

8) TURN OFF operators 1 through 4 and TURN ON operators 5 and 6 ("000011"). Since the steel drum is played with soft mallets and not wooden hammers, we won't need any kind of click in our sound. Therefore, change the Mode of both operators 5 and 6 to "ratio" and enter in ratio numbers of 2.74 for operator 5 (from a Coarse setting of 2.00) and 11.55 for operator 6 (from a Coarse setting of 7.00). Play a few keys and listen (Audio Cue 29G). Note that, while the pitch changes as we play different notes, the sound is still very clicky. This is due mainly to the actions of the envelope generators - to be discussed in Chapter Nine.

9) TURN ON operators 1 through 4 ("111111") in order to listen to the entire sound (Audio Cue 29H). All that remains to be done now is to balance the blend of our three components. This change in volume is accomplished, of course, by changing the output levels of our carriers - operators 1, 3, and 5. Since there is an inordinate amount of "click" in the sound, lower operator 5's output level to a new value of 64. Because the first system is slightly more harmonic than the second (because the carrier in the first system is at a whole ratio number), reduce the contribution of that system slightly by lowering operator 1's output level to a new value of 93. Play a few keys and listen (Audio Cue 29I). Our celeste has been transformed into a steel drum!

* Since the Coarse value is 2.00, we can nearly double it with the Fine parameter, in increments of 0.02. However, note that since ".54" is also divisible by 3, we could have also arrived at this Fine value by setting a Coarse value of 3.00. If you don't believe me, try it for yourself!

Of course, the steel drum we created can be modified further still. Experiment some more with the "Celeste" preset, tweaking a bit here and bit there until you end up with the steel drum that sounds best to *your* ears. Once again, don't be frustrated by the fact that you can't make significantly more radical changes to this preset - remember that there are many more factors at work here than just output level and pitch data inputs. As we work our way through the book and learn about these other tools, we'll return to this preset and see how we can modify it further still.

In Chapter Eight, we will have an opportunity to name and store our modified sound, but first we must complete our discussion of the effects of modulators by discussing *stacked modulators* and something called the *feedback loop*.

Chapter Seven ◇ Stacked Modulators and The Feedback Loop

You've probably noticed by now that there are many algorithms that provide us with modulators modulating *other* modulators, as in algorithm #3 **(see figure 7-1)** or algorithm #16 **(see figure 7-2)** or algorithm #1, which actually provides us with a modulator (operator 6) modulating another modulator (operator 5) modulating yet *another* modulator (operator 4) before the signal finally reaches the carrier (operator 3)! **(see figure 7-3)**

These systems of more than one modulator in a row are called *stacked modulators*, or *stacks* for short. (These are also sometimes referred to as *cascades*.) We can best describe the purpose and effect of these stacks by running an exercise. What we will do here is to start ourselves in algorithm #23, which looks like **figure 7-4.**

We'll begin by using just operators 2 and 3, which comprise a single modulator-carrier system, to generate a sawtooth wave. As we've learned in the previous chapter, we can change algorithms at any time, so, having made our sawtooth wave, we will next change over to algorithm #3, which looks like **figure 7-5.**

As you can see, in algorithm #3, operator 3 is still modulating operator 2, but this time operator 2 is no longer a carrier but is instead configured as another modulator, affecting operator 1 (our carrier). When you change algorithms, remember, all of your previous data is retained, and so even though operator 2 is now a modulator instead of a carrier, it is *still* a sawtooth wave, since operator 3 is still affecting it in precisely the same way. **(see figure 7-6)**

By sending signal from the output of operator 2 to operator 1, we are now modulating our carrier with a sawtooth (complex) wave, instead of just a simple sine wave. The aural results will be strikingly different. Let's do it:

Algorithm #3:

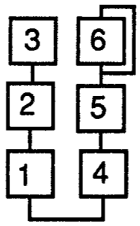


Figure 7-1

Algorithm #16:

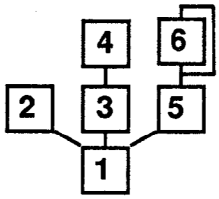


Figure 7-2

Algorithm #1:

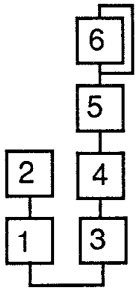


Figure 7-3

Algorithm #23:

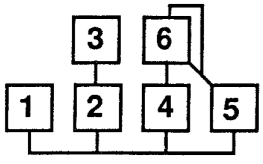


Figure 7-4

Algorithm #3:

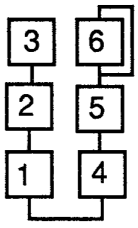


Figure 7-5

Algorithm #23 Algorithm #3

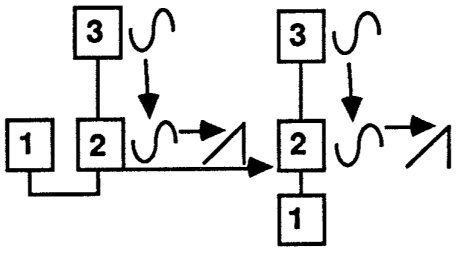


Figure 7-6

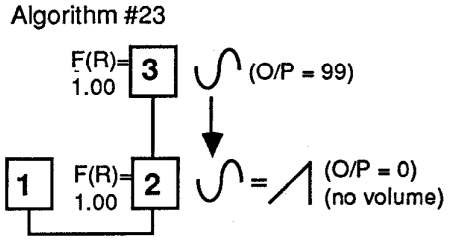


Figure 7-7

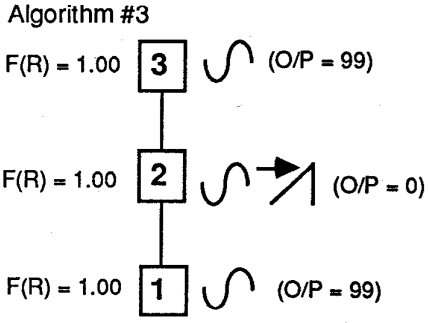


Figure 7-8

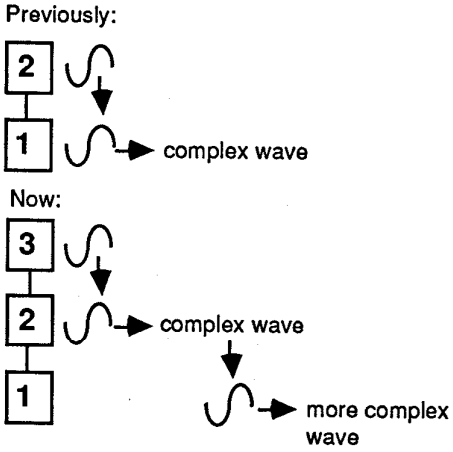


Figure 7-9

Exercise 30
Modulating with a sawtooth wave

1) INITIALIZE your DX7II from single voice play mode, select algorithm #23, press one of the *operator-specific* edit switches (that is, switches 8, 9, 10, or 11), and press edit switches 17, 20, 21, and 22 in order to TURN OFF operators 1, 4, 5, and 6 ("011000").

2) Using the system of operators 2 and 3, GENERATE a sawtooth wave at maximum output (operator 2 Level = 99)and listen to confirm (Audio Cue 30A).

3) Press edit switch 10, followed by edit switch 2 in order to VIEW the Output Level parameter of operator 2 and CHANGE this value to 0. Since operator 2 is a carrier in this algorithm, this will have the effect of lowering the volume to zero. Note that even though we can now no longer hear operator 2, it is *still* a sawtooth wave since operator 3 continues to have an output level of 99. **(see figure 7-7)**

4) Press edit switch 7 and change the current algorithm to algorithm #3.

5) Press an operator-specific edit switch and TURN ON operator 1 ("111000"). Even though operators 2 and 3 are still on, we won't hear their effect, since the output level of operator 2 has been reduced to 0. Examine the algorithm diagram and notethat there is no way that operator 3 can send any modulation data directly to operator 1. **(see figure 7-8)**

6) Play a note on the keyboard and listen (Audio Cue 30B) in order to confirm that you are only hearing the sine wave generated by the currently unmodulated operator 1.

7) Press edit switch 10, followed by edit switch 2 in order to VIEW the Output Level parameter for operator 2, and, using the data entry slider, slowly increase this value up to its maximum of 99, tapping a note on the keyboard and listening as you do so (Audio Cue 30C). Note that you cannot simply hold a note down, as we are altering output level, which does not change in real time.

What exactly do we now hear? First of all, the overtones begin appearing at a much earlier point. By the time you increase the output level of operator 2 to a value of 20 or so, several overtones already begin to make their presence felt. Secondly, the sound gets much, much brighter than anything we've heard previously, to the point where it actually overloads and distorts when the output level is close to or at maximum. Of course, there was no reason to have to start in algorithm #23; we did that here just so you could demonstrably hear that operator 2 had in fact been changed to a sawtooth wave. When generating complex timbres with stacks of modulators, you can start directly in algorithm #3, or any other algorithm you want, for that matter.

The sole purpose of these stacks, then, is to allow us to use *complex* waves as modulating sources. After all, every timbre we've previously generated has been a result of feeding a sine wave output into another sine wave. What we've accomplished here is the sending of a complex wave output (in this case a sawtooth wave) into the sine wave initially generated by the carrier. **(see figure 7-9)**

The advantage of using a complex wave as a modulating source is that it enables us to generate overbright, twangy, biting kinds of sounds in the DX7II, and, in large doses, actually *distorted* sounds! If your instrument is still set up from the end of Exercise 30, try cranking the output level of operator 2 up to 99, play a hot guitar lick on the

keyboard, and you'll hear a nastier fuzz guitar sound on your DX7II than you could ever hope to accomplish on the cheapest (and most distorted) analog synth around!

Of course, we're not limited to just using a sawtooth wave as our complex modulating source. We can in fact use any timbre we like. Just for the sake of illustration, let's run a quick exercise to hear the effect of a *square* wave as our modulating source:

Exercise 31
Modulating with a square wave

1) INITIALIZE your DX7II from single voice play mode, change the default algorithm to algorithm #3, press any operator-specific edit switch, and TURN OFF operators 4 through 6 ("111000"). Play a note on the keyboard and listen to confirm that all you are hearing is the single sine wave being generated by operator 1 (Audio Cue 31A).

2) GENERATE a square wave in operator 2 by setting the Coarse value of operator 3 to a value of 2.00, and the output level of operator 3 to a value of 71. **(see figure 7-10)** Note that even though operator 2 is now outputting a square wave, we cannot hear its effect on operator 1 since the output level of operator 2 is still 0.

Algorithm #3:

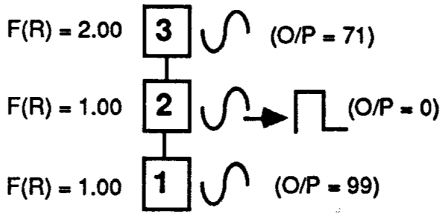


Figure 7-10

3) Press edit switch 10, followed by edit switch 2 in order to VIEW the Output Level parameter for operator 2, and, using the data entry slider, slowly increase this up to its maximum value of 99, tapping a note on the keyboard and listening as you do so (Audio Cue 31B). Note that once again, the overtones appear at an earlier stage and that the sound becomes significantly brighter than that generated by a single modulator.

4) Experiment by changing the frequency ratio between operators 2 and 3; between operators 2 and 1; between operators 3 and 1. Experiment further by changing the output level of operator 3 as well as that of operator 2. Note that as higher frequency ratios are set, more distortions appear as we increase the inharmonic content. Note also that this can be offset by reducing the output level of either operator 2 or operator 3; and that each such data change produces a qualitatively different effect.

If you plan to generate a sound on your DX7II that is harpsichord-like, clavinet-like, guitar-like, banjo-like, or any kind of twangy, sharp, biting sound, you will probably need a stack or two to accomplish this. Several algorithms, like #3 and #4, offer you two stacks, while others, like #10 or #30, offer you one stack plus several other systems. Since a stack by definition will consist of at least 3 operators, and since we only have a total of six operators at our disposal, there are no algorithms offering you more than two stacks. In any event, this is yet another question to ask yourself when making a decision as to the best algorithm to use for a specific situation. Since over half of the 32 available algorithms do *not* contain stacks, then deciding that you in fact need one will effectively eliminate over half of the potential choices. Don't waste your time trying to make an accurate electric guitar sound, for example, with a single modulator-carrier system - you won't be able to generate enough overtones to do it well. Only using a complex wave as a modulating source can accomplish that, and that's precisely why stacked modulators are available to us.

Algorithm #1:

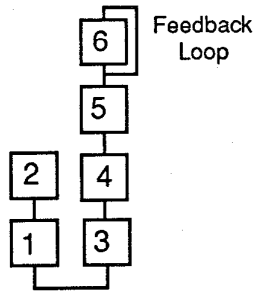


Figure 7-11

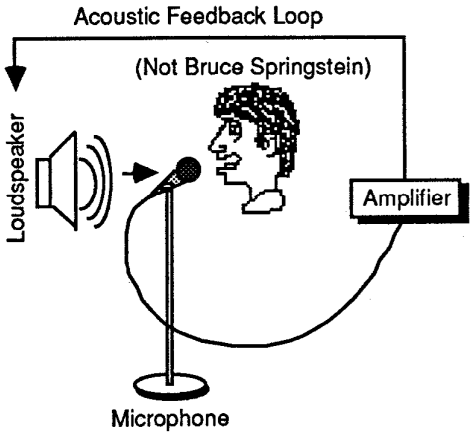


Figure 7-12

Algorithm #5:

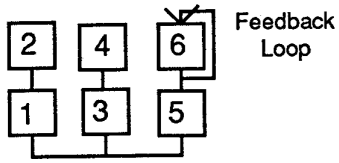


Figure 7-13

Algorithm #6:

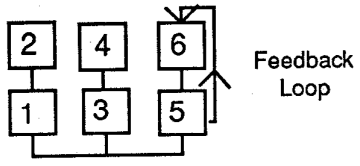


Figure 7-14

The general application and use of these stacks in presets seems to be a fairly recent phenomenon in the history of this instrument. It is worth noting that few of the original DX7 presets used algorithms with stacks, while most of the DX7II presets *do* use them. This may well help to account for the general improvement in the quality of the factory-supplied presets for the newer instrument.

But we still haven't completed our discussion of these configurations. Why would we ever need stacks of *three* modulators, as provided by algorithms #1, 2, and 18? We'll be able to better answer that question after we discuss something called *feedback*.

The Feedback Loop

In every one of the 32 algorithms in the DX7II, we are provided with something called a *feedback loop*. Those of you with keen sensory apparatus (or those of you who are just plain observant!) may have noticed these funny little boxes that are attached to at least one operator in every algorithm diagram, even though we have conveniently not said anything about them up until now. Algorithm #1, for example, has its feedback loop on operator 6. **(see figure 7-11)**

This feedback loop is an alternative means of routing an operator's output: either to its *own* modulation input, or to the input of another operator stacked above it - hence the term, "feedback".

Acoustic feedback is a phenomenon which occurs when a signal is re-routed, or "fed back" into itself, as when an open microphone is placed in front of an active loudspeaker. **(see figure 7-12)**

The output signal from the microphone goes into the loudspeaker, which reproduces it. This reproduced signal is then picked up again by the open microphone, and "fed back" into the loudspeaker; over and over again this occurs, until finally a characteristic squealing and howling sound is generated. Of course, in the DX7II we are not dealing with audio signals of any type, since all of our components are digital. The result of the DX7II feedback loop, then, will be simply to allow a modulator *to modulate itself*, or, in a few algorithms, to allow a *carrier* to actually modulate a modulator stacked above it! (Whew! Try saying *that* three times fast...)

We do not have the power to decide where the feedback loop will be: this is determined by the algorithm we choose. Usually (but not always! - consult the algorithm diagrams!!), the feedback loop is on operator 6 and it usually is set to feed back into itself, as in algorithm #5. **(see figure 7-13)**

Sometimes, however, the loop specifies that operator 6 will instead be the *recipient* of modulation input fed back by a carrier below it, as occurs in algorithm #6. **(see figure 7-14)**

In fact, the *only* difference between algorithm #5 and algorithm #6 is in the feedback loop! Obviously, we will hear qualitatively different aural effects from each configuration.

Take a moment or two to reexamine the 32 algorithms, taking mental note of where the feedback loop is placed for each one. Observe, for example, that the only algorithm that contains a *carrier* feeding back into *itself* is algorithm #32, which, after all, only *has* carriers. In every other algorithm the feedback loop terminates at a modulator.

Let's begin by examining the effect of a modulator feeding back into itself, as with algorithm #5. If operator 6 is sending its output not only to operator 5, as the algorithm indicates, but into its own modulation data input, then it will actually have the effect of modulating itself! (among some irreverent DX7II programmers, this is known as the auto-erotic mode) This means that operator 6 will be able to turn itself into a complex wave, without needing another modulator stacked above it, thereby "saving" you an operator. **(see figure 7-15)**

In the case of algorithm #6, however, operator 6 instead is the recipient of signal outputted by operator 5, sitting below it! This means that the complex wave produced as a result of operator 6 modulating operator 5 is then returned, in varying amplitudes, to the modulation input of operator 6, causing it to output a more complex wave itself. This extra-complex wave is then outputted again into operator 5, and the whole process reoccurs again and again. **(see figure 7-16)**

Exercise 32, below, will let us hear the qualitative aural difference between these two feedback routings.

We can set how much signal enters the feedback loop with the "Fbl" (for "Feedback loop") parameter, accessed by the non-operator-specific edit switch, switch 7. The LCD display looks like **figure 7-17.**

Figure 7-17

The range of the Fbl parameter is 0 - 7; a value of 0 indicates that no signal is entering this pathway, hence no feedback; a value of 7 indicates that maximum signal is in this routing and feedback is maximized. In small doses (values of 1, 2, 3, or 4), the effect of the feedback loop is very similar to that of a stacked modulator. In larger doses (values of 5, 6, or 7), so much signal will be feeding back that the sound will grossly distort and eventually break into *white noise.**

If you take all of the colors of the rainbow and mix them together, you get the color white; if you take all of the audio frequencies and randomly mix them together, you get white noise. White noise is useful for generating thoroughly non-musical sounds such as handclaps, surf, wind, thunder, engines, helicopters, and jackhammers. Using the feedback loop in its higher settings will allow us to generate white noise effects in the DX7II; however, for other reasons soon to be discussed, the DX7II is not a particularly good instrument to use for these kind of effects. In any event, let's try out the feedback loop and hear its contribution to a sound. Note that the feedback loop parameter is not operator-specific and that is why it is accessed from edit switch 7, which shows only non-operator-specific parameters (like the algorithm and the oscillator sync). This makes sense since, as we have seen, we cannot specify which operator is to have the feedback loop - it is determined for us by the algorithm we select.

Algorithm #5:

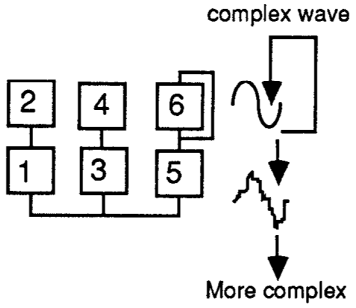


Figure 7-15

Algorithm #6:

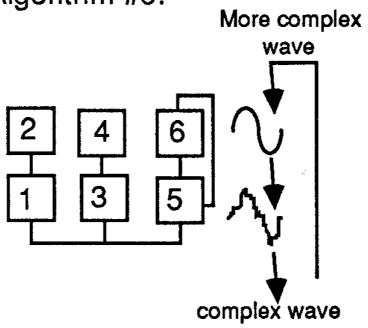


Figure 7-16

* This doesn't mean that you should assume that a high feedback value will *always* yield distorted or noise-like sounds - remember that the output level of the feedback operator will determine just how much of the feedback signal eventually reaches (and therefore effects) the carrier. For example, setting a modulator with a very low output level but high feedback value will certainly NOT yield noise or distortion.

Exercise 32
The feedback loop

1) INITIALIZE your DX7II from single voice play mode and select algorithm #5. TURN OFF operators 1 through 4 ("000011").

2) Using the system of operators 5 and 6, GENERATE a sawtooth wave at maximum output (operator 5 Level = 99) and listen to confirm (Audio Cue 32A).

3) Press edit switch 7 to VIEW the "Fbl" parameter. Note that the initialization default for this parameter is 0 (no feedback).

4) Hold down a single note on the keyboard and use the "yes" button to slowly increase this value up to its maximum of 7, listening as you do so (Audio Cue 32B). Note that the sound gets progressively brighter, simulating the effect of a stack, until, at the highest settings, it distorts and eventually degenerates into (at a value of 7) white noise.

5) Restore the Fbl value to 0 and use the cursor and data entry switches in order to change over to algorithm #6.

Algorithm #6:

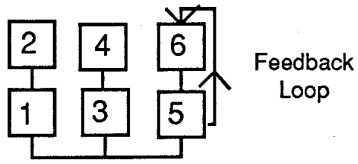


Figure 7-18

6) Repeat steps 3 and 4 above, listening as you do so (Audio Cue 32C). notethat, since in this algorithm operator 6 is not feeding back into itself but is instead the *recipient* of operator 5's output signal, **(see figure 7-18)** the effect is qualitatively different. In this configuration, the feedback overtones appear earlier and distort more quickly, producing a "cleaner" type of white noise.

7) Experiment by using timbres other than a sawtooth wave, and by trying the feedback loop in other algorithms. Try specifically using algorithm #32, since this is the only algorithm with a *carrier* feeding back into itself (notethat the aural results are particularly unspectacular). Note that regardless of the timbre selected or the algorithm used, the feedback loop always has the aural effect of brightening and/or distorting the overall sound. experiment further by calling up various presets in single voice play mode, examining their feedback loop settings, and changing them. Listen to the increase in brightness as you increase the Fbl value in all instances.

It should be obvious that the feedback loop can sometimes be used as a "phantom" seventh operator, since it can so closely simulate the effect of a stack.* For example, suppose you've created a sound on your DX7II using all six operators in the configuration of algorithm #5, and you feel that it's lacking just that extra bit of "bite". You may already have the output levels of your modulators at or near maximum, but the sound is still not cutting through enough. Well, even if you look for another algorithm - one with a stack - you're unfortunately out of luck because there are only six operators, and you're already using them all! In this example, opening up the feedback loop already present in operator 6 may well be all you need to do in order to complete the sound to your satisfaction.

Even when, in the last Exercise, we used algorithm #6 and raised the Fbl value to its maximum of 7, we didn't really come up with a very pure white noise. Noise, by definition, should be completely pitchless, and you probably noticed that quite a bit of the original carrier sine wave was still present in the sound. We can increase the purity of the noise we generate on the DX7II (that is, decrease the amount of the pitch component) by using stacks of *three* modulators, and this is the main use of such stacks.

* albeit a stack modulating with a sawtooth wave *only* , since the frequency ratio between any operator and itself can only ever be 1:1.

Here is a good method for creating pure unpitched white noise on the DX7II, using a stack of three modulators, with a carrier in sub-audio fixed-frequency mode:

Exercise 33
Generating unpitched white noise

1) INITIALIZE your DX7II from single voice play mode and leave it in algorithm #1. TURN OFF operators 1 and 2 ("001111").

2) Press edit switch 10, followed by edit switch 3, in order to VIEW the Output Level parameter for operator 3, and change it to a value of 99. listen to the pure sine wave that is coming from this single unmodulated carrier. (Audio Cue 33A)

3) Press edit switch 4 and use the data entry section to change the output level of operator 4 to a value of 99, generating a sawtooth wave in this system (remember, the frequency ratio between operators 3 and 4 is already 1:1, since these are default values). Listen to confirm (Audio Cue 33B).

4) Press edit switch 5 and change the output level of operator 5 to a value of 99, generating a distorted sound as a result of changing operator 3 to a sawtooth wave and using that sawtooth wave (instead of a sine wave) as a modulating source. Listen to confirm (Audio Cue 33C).

5) Press edit switch 6 and change the output level of operator 6 to a value of 99, increasing the distortion present in the sound to a point where it is generating noise. Listen to confirm (Audio Cue 33D). Note that this noise is still fairly pitched, and therefore not yet white noise.

6) VIEW the Fbl parameter (edit switch 7) and change it to its maximum value of 7. Note that in algorithm #1, the feedback loop is set so that operator 6 (the top modulator in the stack) is feeding back into itself. Listen (Audio Cue 33E) to confirm that the pitch component in the noise has been greatly reduced, but is still present.

7) Press edit switch 8, followed by edit switch 3, in order to VIEW the Mode parameter for operator 3 (the carrier). Use the cursor switches and "yes" button to change this to "fixed".

8) Note that the Coarse parameter for operator 3 shows that it is currently 10.00 Hz, which is a sub-audio frequency. Leave it at this setting and listen (Audio Cue 33F). Note that the pitch component has again been reduced but is still slightly present.

9) Press edit switch 4 and change operator 4 also to "fixed" mode. notethat its Coarse value is now also at the default of 10.00 Hz. Listen (Audio Cue 33G). Note that we reduced the pitch component further still.

10) Repeat step 9 above, but this time for operator 5. Listen (Audio Cue 33H) and notethat the pitch component is now entirely gone, and that we have now succeeded in generating pure white noise.

11) Experiment by changing operator 6 (the top modulator) also to a fixed frequency of 10.00 Hz. Listen (Audio Cue 33I) and notethat this produces a helicopter-like sound; interesting, but not white noise. Experiment further with other fixed frequencies for the various operators in the stack provided by algorithm #1.

The reason why the DX7II is not generally very useful as a source for white noise-based sounds is because of the action of the operator's envelope generators (see Chapter Nine for a complete discussion of these little beauties!). These have a tendency to restore the pitch component when they undergo complex changes. There is a

mathematical explanation as to why this unfortunate phenomenon occurs, but it will be beyond the scope of this book to go into it. Suffice it to say that generating typical white-noise effects, such as whipcracks, handclaps, snare drums, thunder, wind, and surf, are best left to analog synthesizers, which are usually much more successful at generating such sounds. In any event, here's a diagram of what we constructed in Exercise 33. **(see figure 7-19)**

Algorithm #1:

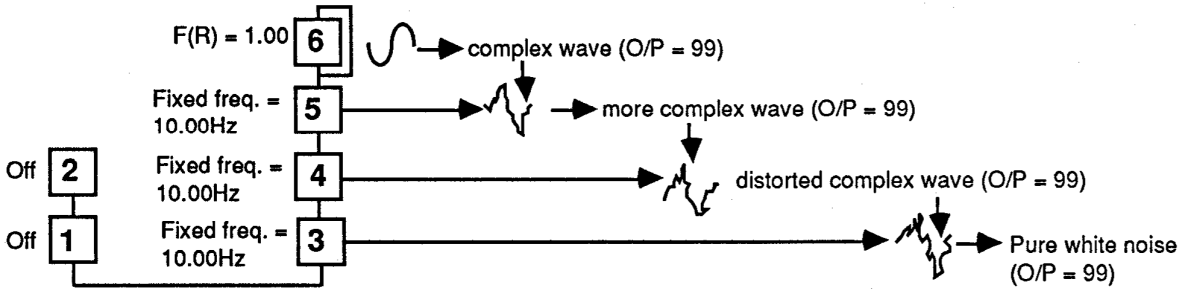


Figure 7-19

Now let's see how we can store some of the sounds we've been creating!

Chapter Eight ◇
Storing Sounds To Internal Cartridge and Disk Memories

In the overview presented in Chapter Two we spoke of both the DX7II internal memory and of the external memories - like cartridge and disk - that the instrument can use. When we discuss the MIDI interface in Chapter Fifteen, we will see that we can also access whatever memory and memory storage devices are available in virtually any computer as well. For now, however, we'll limit ourselves to just the internal and external DX7II memories.

Let's just do a quick review of the features of each kind of memory: the internal, the RAM4 cartridge, the ROM cartridge, and, if you have the "FD" model of the DX7II, disk memory:

1) Internal memory: 64 *voice* slots (plus *performance, microtuning*, and *system* slots, all to be discussed in Chapters Fourteen and Fifteen); data can be read from or written to the internal memory.

2) RAM4 cartridge memory: can be used to store precisely the same amounts and types of data as in the internal memory (or to store microtuning or *fractional level scaling* data alone - more about this in Chapter Twelve); like the internal memory, data can also be read from or written to the RAM4 cartridge.

3) ROM cartridge memory: normally four banks of data equivalent to that stored in the RAM4 cartridge. Data can *only be read* from a ROM cartridge. The information stored in these cartridges cannot be overwritten.

4) Disk memory: stores up to 44 files of data. Each file can be the equivalent of a complete internal memory bank, of a single cartridge memory bank, or may consist of generic MIDI data (when used in the *MIDI Data Recorder*, or *MDR* mode - more about this in Chapter Fifteen). Data can be read from or written to disks quickly and easily - but to access this data, it must first be stored into either the internal memory or into a RAM4 cartridge placed in the slot.

Rather than simply discuss the overall data storage procedure, it will probably be easier (and certainly more fun) to actually store a sound we create. The sound we store can be something we've built from scratch, via the *voice initialization* procedure, or it could be a sound we've modified (or even not modified) from a preexisting one. Let's work here with a modification of a preset. Take a few minutes to go back to the end of Chapter Six and redo Exercise 30. This Exercise, you may remember, modified the "Celeste" sound found in your ROM cartridge, bank 2, voice number 62 - changing it into a steel drum sound. Got the modified sound in your edit buffer? Good - let's carry on. . .

The first thing you probably noticed when simply playing back voices stored in memory is that every voice and every performance memory has a name. These names of up to ten characters (for voice names - up to twenty in the case of performance names) are for identification

purposes only and don't actually mean anything at all - you can, for example, generate a flute sound in your DX7II and name it "Tuba", or you can generate a tuba sound and name it "Fred". Normally, of course, you'll want to give a sound a name that somehow identifies it, just for the sake of your sanity, but the point is, you have the freedom to name, or rename, any sound in any way at all. When you create a new voice via the voice initialization procedure, for example, the default name for that voice will always be "INIT VOICE". Obviously, you should always rename every voice you create this way, or else you may well end up with a memory full of different sounds, all named "INIT VOICE".

The edit parameter that allows us to name or rename a sound is accessed when pressing edit switch 7 (which, you may remember, is non-operator-specific). If your DX7II isn't already in edit mode, put it there by pressing the green edit mode select switch, and then press switch 7. The right-most parameter in this display is called "Voice Name". Use the cursor switches to position the blinking cursor over this parameter, and it will automatically move over the first letter of the name, which should still be "Celeste". **(see figure 8-1)**

Figure 8-1

Underneath each of the 32 main switches as well as eight of the ten switches on the left-hand side of the machine (the exception being the cursor switches) is a brown *character*, and these are used solely for naming voices and/or performance memories. Switches 1 through 9 offer the numbers 1 through 9, and switch 10 has the number "0"; following that, switches 11 through 16 access the letters A through F, and switches 17 through 32 offer the letters G through V. **(see figure 8-2)**

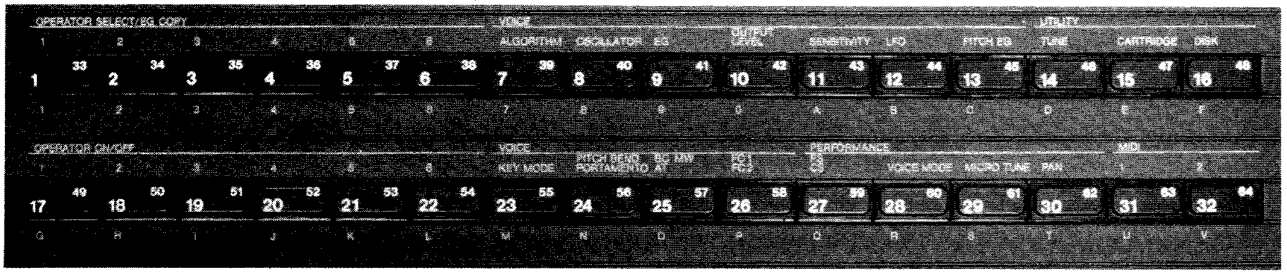


Figure 8-2

W, X, and Y are underneath the three voice play *mode select switches* (to the left of the LED display) **(see figure 8-3)** and the pink *Store* switch has the letter "Z" under it **(see figure 8-4)**while the *performance* play mode select switch doubles as a space bar, and will be used to enter blank characters (that is, spaces) **(see figure 8-5)** and, finally, the two data entry "yes-no" switches contain a hyphen and a period, respectively. **(see figure 8-6)**

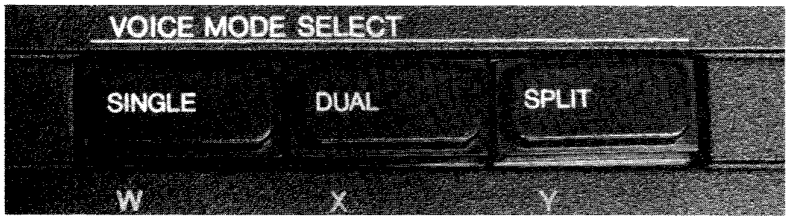


Figure 8-3

The cursor switches, as might be expected, will move the blinking cursor to the left or right, as indicated by the arrows inside them. The "internal" and "cartridge" switches, as we will see shortly, are used to select upper- or lower-case characters. This leaves us with only one remaining switch on the entire front panel, and that switch is the edit mode select switch, underneath which you will find the label, "CHARACTER".**(see figure 8-7)**

This is the switch which activates all of the previously named ones for the brown characters underneath them. When actively selecting characters in order to name a sound, you must keep this switch *held down* . This is the only way that the DX7II will know that, for example, switch 15 is to yield the character "E", and not call up parameters dealing with the cartridge memory. **(see figure 8-8)**

Note, again, the color-coding system. The word "CHARACTER" underneath the edit mode select switch is brown, linking this switch with the brown characters underneath all of the other switches. Again, if this switch is not held down while the cursor is positioned over the "Voice Name" parameter, the main switches will remain active for their other, "green", edit functions.

Unlike the original DX7, the DX7II can use both upper and lower-case characters when naming voices or performance memories. This is accomplished with the use of the "internal" and "cartridge" switches immediately to the right of the LCD. If you press the "internal" switch while holding down the character key, it will cause all subsequently pressed characters to be lower-case. Pressing the "cartridge" switch while holding down the character key will cause all subsequently selected characters to be upper-case. You can, of course, mix and match lower- and upper-case characters in the same name.

At the moment, of course, our sound already has a name, and that name is "Celeste". Why then, do we need to rename it? Well, if we simply save it in our internal memory or on a RAM4 cartridge or disk, how will we be able to distinguish it from the original "Celeste"? After all, we made some significant changes to the original sound. It would be a real pain to have to remember which one was which, especially if the original and the modification were stored in the same internal or external memory. The solution, obviously, is to give one of them a different name. Since the original is a factory preset from a ROM cartridge, and since we can't ever overwrite data in a ROM, it makes sense that we will want to rename our modified sound. Let's run an Exercise, and rename our new sound " - Joe - ".

Exercise 34
Renaming a sound

1) Your DX7II should currently have in its edit buffer the modification of the "Celeste" sound done in Exercise 30 (Chapter Six). If not, redo this Exercise now.

2) Press edit switch 7 in order to VIEW the Voice Name parameter, which should read "Celeste". Use the cursor switches to position the blinking cursor over the Voice Name parameter, and note that the cursor automatically appears over the letter "C".

3) Press and *hold down* the edit mode select switch, thereby activating all other switches for their characters. *Make sure this switch is held down before you select any characters.*

4) Since, in the name " - Joe - ", the first character is a space, press the performance mode select switch once, thereby inserting a blank character over the letter "C". note that the cursor automatically moves over the next character, which is an "e".

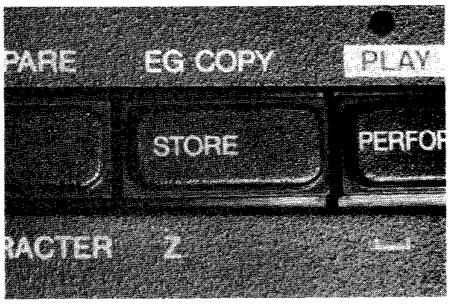


Figure 8-4

Figure 8-5

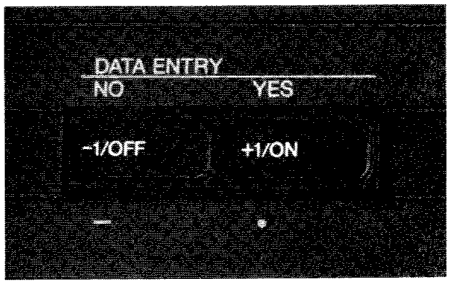


Figure 8-6

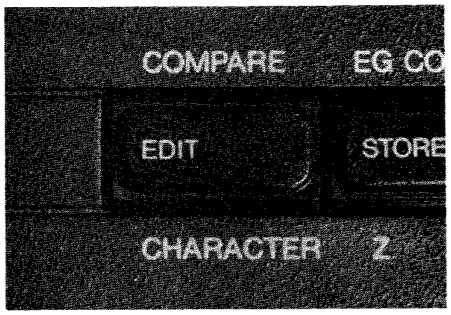


Figure 8-7

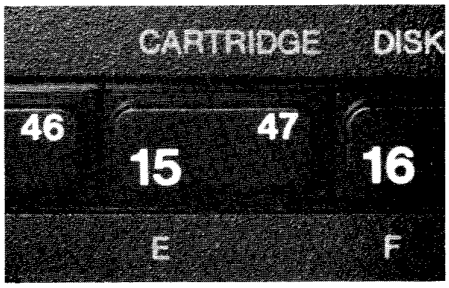


Figure 8-8

5) We wish to replace the "e" with a hyphen, so press the "no" data entry switch once to insert a "-" over the second character. note that the cursor again moves on.

6) Insert a blank space into the next character by once again pressing the space bar (performance mode select switch), as in step 4 above.

7) Press the "cartridge" switch in order to activate upper-case characters, and then press switch 20 in order to generate the character "J". Now press the "internal" switch in order to change over to lower-case characters, and follow this with switches 25 and 15 in order to enter the characters "o" and "e". Follow this with another space (performance mode select switch) and then another hyphen ("no" data entry switch).

8) Your name should now read " - Joe - ". experiment by renaming the sound " - Woe - ", " - Hoe - " or (for you Three Stooges fans) " - Moe - ". You can accomplish this easily by using the left cursor switch to move the cursor to the left, back over the letter "J", and then simply retyping the new letter. note that doing this does *not* erase any old characters. The cursor switches simply move the cursor without any erasure. To actually erase a character, use the space bar (performance mode select switch).

9) Experiment further by renaming the sound anything you like (note that the DX7II does not mind four-letter words, but that your relatives might!). After experimenting, restore the sound to its new name of " - Joe - " for identification purposes.

When naming a performance memory, as opposed to a voice, the same general procedure is followed, but we access the performance name from edit switch 29 (labeled MICROTUNE) instead of edit switch 7. We'll be doing this a bit later on, when we get to our study of performance controls in Chapter Fourteen.

Okay, now we've created our sound (by modification) and we've named it. The question is, where do we want to put it? Just knowing that we want to put it in internal, or RAM4 cartridge memory (we'll talk about disk storage a bit later) isn't enough - we need to know just *which* slot to put it in. This isn't just an arbitrary thing, either, since writing data into a memory slot ALWAYS erases whatever was previously in that slot - remember, each slot can only hold *one* set of data at a time.

We'd therefore like to be able to see what is in our internal or RAM4 cartridge memory, *without losing our current sound:* " - Joe -". This will allow us to decide what we're willing to erase in order to store " - Joe - ". At this point on most synthesizers, we would be up the proverbial creek with the proverbial paddle because most other synthesizers will not allow you to call up previously stored sounds without losing your edited work. This gives you two options - both bad: lose your modified sound, or take a deep breath and blindly store it in a slot somewhere, praying that in doing so, you aren't erasing something vitally important. In the DX7II, however, we don't have this problem, thanks to a wonderful area of memory called the *edit recall buffer.*

We know that whenever we call up or work on any voice or performance memory at all in the DX7II, the edit buffer presents us with a copy of that sound and not the original sound itself. In point of fact, the instrument has two discrete edit buffers - one for voices and the other for performance memory. For the sake of simplicity, let us discuss here only the voice edit buffer (though the performance edit buffer works exactly the same way).

Each edit buffer is actually a two-part memory. It contains within itself a little sub-section called the edit *recall buffer*: **(see figure 8-9)**

This part of the edit buffer is protected by the back-up battery and its contents can easily be recalled with a simple edit command, accessed from switch 14 (TUNE). This switch, which we have already used (when we initialized), also has an LCD display which allows you to "Recall edit".**(see figure 8-10)**

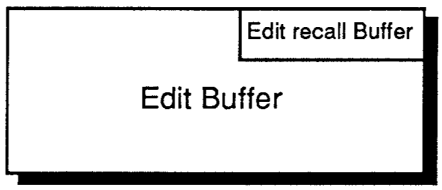


Figure 8-9

Figure 8-10

How do we get data into this special part of the edit buffer? No problem, the DX7II does it for us automatically! Take a look at your LED and you will notice a decimal point after the number. You may previously have noticed this and wondered about its significance. The decimal point appears whenever you first make any kind of change to a voice or performance memory (*except* when you initialize). That is to say, it makes its appearance the first time you enter any new data into a sound, by using the data entry section. In this particular example, we first saw the decimal point in step 6 of Exercise 30, when we first changed the ratio number of operator 2. Note that simply turning operators ON and OFF, or using the operator select switches, is not sufficient to make the decimal point appear since there is no activity in the data entry section.

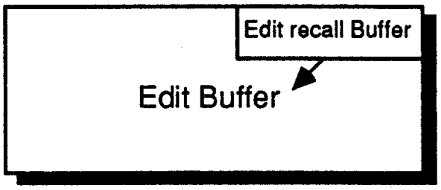
What exactly does this decimal point mean? Well, it's a reminder that a sound has somehow been changed, and it is confirmation from the DX7II that our current work is entered and stored in the edit recall buffer, where it will be protected and can later be recalled. *The edit recall buffer ALWAYS holds the most recent editing work you did.* As with all the other memory slots, it can only hold *one* set of data at a time. That means that if you then decide to stop altering "Celeste" and instead initialize and make a sawtooth wave, all of your modification work in the edit recall buffer would be lost and it would instead contain just the sawtooth wave.

This extra bit of memory is a wonderful safety net, mainly because it is protected by the back-up battery. This means that if you sit up till 5 AM constructing the most amazing DX7II sound ever heard and you then turn off the machine without remembering to store the sound, it will still be in there the next day! It also means that at any point, we can *leave* edit mode and return to voice or performance play mode in order to see just what we've got in memory and decide what we're willing to erase, *without losing our modified sound!!*

Few synthesizers have anything like the edit recall buffer - meaning that, when working with most instruments, you need to know in advance exactly which slot you plan on storing a sound in before you can even *think* about creating it. This can be very problematic unless detailed written records of the contents of the synthesizer's memory are kept - and, despite everyone's best intentions, this is a rare occurrence indeed.

We instruct the DX7II to recall the contents of the edit recall buffer by pressing edit switch 14 repeatedly until the Recall Edit display appears. At this point we have the option of recalling either the voice

edit, the performance edit, or the *microtuning* edit (more about this in Chapter Fourteen). For now, of course, we are only dealing with voice memory, so we will want to use the cursor switches to position the blinking cursor over the "Voice" parameter. At this point, pressing the "yes" data entry switch will cause an "** Are you sure?" question (just as we had when we went through the voice initialization procedure). The reason for this, again, is that recalling the edit memory is a drastic change that will normally cause most if not all of the edit parameters to be altered (and, again, this is no tragedy since we are ALWAYS in the edit buffer anyway). If we then answer "yes" to this question, then the display will happily read "** Completed!", and our edited work will be shifted from the edit recall buffer back into the edit buffer, so we can once again hear it. **(see figure 8-11)**



Data restored through Recall Edit Procedure.

Figure 8-11

With all of this information in hand, we can now Boldly Go Forward, etc. , and *leave* edit mode in order to examine the current contents of our internal memory. Let's take a deep breath and do it:

Exercise 35
Using the edit recall buffer

1) Your DX7II should currently have the modification of "Celeste" in its edit buffer, renamed " - Joe - ". If this is not the case, redo Exercise 34 above.

2) You should currently be in edit mode, with the cursor positioned over the Voice Name parameter. If this is not the case, go into edit mode, press main switch 7, and use the cursor select switch to do so. The LED should read "62. " (since the "Celeste" sound was originally stored in slot 62).

3) Turn off your DX7II - even unplug it if you like! Wait a couple of seconds and then switch it back on again. note that you are returned to single voice play mode (since the DX7II always goes to the most recent play mode on power-up) but that the voice name in the display is still " - Joe - ". PLAY a few notes on the keyboard. The modified " - Joe - " sound - not the original "Celeste" sound - is what you should be hearing.

4) Press the "single" voice play mode switch in order to *leave* edit mode and enter play mode.

5) Press the "internal" switch, followed by various main switches and/or the "32/64" switch in order to hear the voices you currently have stored in your internal memory. note the slot number of one you're willing to erase. Be careful not to make any changes to any sounds by reentering edit mode - stay in play mode!

6) Insert a cartridge (RAM4 or ROM) into the slot and enter cartridge single voice play mode by pressing the "cartridge" switch.

7) Press the various main switches and/or the "32/64" switch in order to hear the sounds you currently have stored in the cartridge memory. Once again, do not make any changes to any of these sounds!

8) Finally, select any sound from internal or cartridge memory that is significantly dissimilar from " - Joe - ". Play a few notes on the keyboard and listen.

9) Put your DX7II back into edit mode by pressing the green edit mode select switch.

10) Press edit switch 14 (TUNE) repeatedly in order to call up the "Recall edit" display. Use the cursor switches to position the cursor over the "Voice" parameter. Press the "yes" switch twice so that you see the "** Completed!" display.

11) Play a few notes on the keyboard and listen. " - Joe - " is back!!
12) Press edit switch 7 in order to VIEW the Voice Name and note that it is still " - Joe - ".

You were asked to actually turn off the power switch on your DX7II in order to illustrate that the contents of the edit recall buffer are in fact protected by the back-up battery, same as the internal memory slots. The data contained in the edit recall buffer will remain there until such time as you reintroduce the decimal point by making another change to another sound, or until the back-up battery fails (normally not for a good five years). That's why you were warned not to make any edit parameter changes to any sounds you were viewing, since doing so would have automatically erased " - Joe - " from the edit recall buffer and replaced it with your new work.

In any event, you were provided - courtesy of the edit recall buffer - with an opportunity to review the sounds currently in internal as well as cartridge memory and to decide which sound or sounds you consider expendable. If you didn't actually take note of which voice you're willing to erase first, go back and redo Exercise 35 and find one. Finding an "available" memory slot can be a lot like doing one of those 25-cent puzzles we all grew up with - the ones where you have to move pieces around to an empty slot in order to finally get them in order. **(see figure 8-12)**

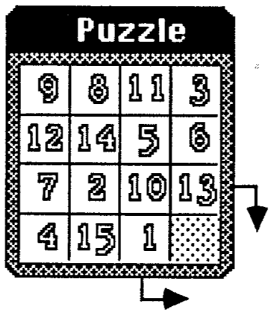


Figure 8-12

We're going to eventually be storing " - Joe - " in internal memory slot 42 (just an arbitrary number) so if you're fond of what's already in internal slot 42, go back and pick a sound somewhere else to be sacrificed. Let's suppose, for example, you decide that the sound in internal slot 3 is particularly loathsome to you - but you do quite like what's stored in slot 42. We are still going to put " - Joe - " into slot 42, not slot 3, so we're going to somehow have to *copy* the voice in slot 42 over to slot 3, so it will still exist even after we put " - Joe - " in slot 42. How do we do this?

Fortunately, just as we could move pieces of the plastic puzzle around, the data contained in each memory slot can be easily moved from place to place (the only limitation being, of course, that you cannot *write* new data into a ROM cartridge). Since I have no way of knowing what you currently have in the internal memory of your DX7II, we'll have to run an Exercise to free up internal slot 42. This will require that you be prepared to permanently erase one of the voices inside your DX7II, and you'll have to decide which one yourself. Of course, if your DX7II internal memory currently contains the so-called "Master Group", that is, a duplicate of what is in bank 1 of your ROM cartridge (this is the way that DX7IIs are normally shipped from the factory), then you have nothing to worry about. In this case, you can literally erase ANY sound in your internal memory and be certain that you already have an uneraseable copy of these sounds in the ROM cartridge itself.

The *store* procedure allows us to actually write data to a memory slot, be it internal or RAM4 cartridge. This procedure is purposely just a little complicated in order to prevent you from ever accidentally erasing a sound you wanted to keep. Actually, the description of this is more complicated than the procedure itself, but it's set up so you have to press several different switches in a particular order to make it happen.

Here is a general outline of how to store sounds:

1) Memory protect
2) Memory select
3) Slot select
4) Confirm

Both the internal memory and the RAM4 cartridge memory are normally *protected* from being written to by circuitry in the DX7II (in addition to a hardware protection in the RAM4 cartridge itself, to be discussed shortly). This protection is automatically ON whenever you turn on the instrument, and in fact is one of only a handful of things on the DX7II that actually resets itself on power-up. You can turn off this memory protection, for either the internal or cartridge memory (or both), by pressing good old edit switch 14 again - repeatedly if necessary - until you come up with the Master Tuning/ Memory Protect display, which looks like **figure 8-13.**

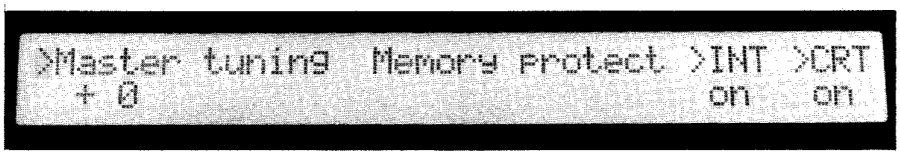


Figure 8-13

If we wish to write data into the internal memory, we will need to turn the internal memory protection off, and if we wish to write data into the cartridge memory, we will need to turn the cartridge memory protection off. This is accomplished by using the cursor switches in order to move the cursor over the appropriate parameter (*INT* or *CRT*) and simply pressing the "no" switch, which you may remember, moonlights as an "off" switch (just as the "yes" button doubles as an "on" switch).

The second step in the procedure is to tell our dumb computer exactly which memory we wish to write the data to. The DX7II does *not* necessarily assume that since you've turned the internal memory protection off that that is where you wish to send your data. You will still need to tell it the destination by pressing either the "internal" or the "cartridge" switch. You can leave out this step if you plan on writing to the currently selected memory, however. For example, if you decide to store a voice and you were most recently playing or editing an internal voice, the DX7II will assume that you want to store it into the internal memory. However, if you were most recently playing or editing a cartridge voice that you wished to store in the internal memory, you'd need to press the "internal" switch in order to tell the instrument that you wanted to access a different memory.

Now we come to one major difference between the DX7II and the original DX7. Unlike the original instrument, here *you cannot store a voice directly from edit mode.* Instead, you must return to *play* mode before you can initiate the storage procedure. As we discussed in Chapter Two, you can only go from edit mode directly back to the play mode you started from (that is, if you were editing a *single* voice, you couldn't return to *dual, split,* or *performance* mode). However, from the starting play mode, you can go to any other voice play mode and store the voice from there. In dual or split mode, you'll have to be careful to keep on eye on whether you're storing voice A or voice B (and you can, of course, change your selection with the "A/B" switch) - so in general,

it's best to store voices from single voice play mode. Therefore, even if you were editing in dual or split mode, my advice is to return to single play mode before storing. *

After you've returned to play mode, and *made sure that the voice you wish to store is currently in the edit buffer** (that is, you can hear that sound when you play notes on the keyboard), we are ready to finally store it. This involves pressing the pink *Store* button for the first time. Like the character key, this needs to be *held down* while you go through the next two steps - if you let go of it prematurely, your voice will *not* be stored. While you are holding down the store button, the LCD will prompt you, "Store data to memory?" if the memory is unprotected, or "Memory protected !" if you forgot to turn the protection OFF.

The next two steps involve telling the microprocessor which slot you want to actually store the sound in, and confirming that you want to actually do so. Therefore, while holding down the store button, you will first of all press the appropriate main switch corresponding to the slot you wish to store your sound in. Remember that if you want to store a voice into any one of slots 33 to 64, the status light above the "1-32/33-64" switch must be lit (this is accomplished simply by pressing the switch at this time). The LCD display will respond now by mimicking the slot number you selected. If, for example, you chose slot 42, the LCD would now look like **figure 8-14.**

Figure 8-14

At this point, if you've accidentally selected the wrong slot, you can simply let go of the store switch and select a different slot. But if you've picked the slot you wanted, simply press the "yes" switch to confirm this to the computer and a happy "** Completed!" will appear in the LCD. Remember, this must be done *while continuing to hold down the store switch.* This is best accomplished by using your pinky or perhaps some other appendage (perhaps we should forget I ever said that. . .). Your sound is now safely stored in the memory slot of your choice.

Finally, at the conclusion of the storage procedure, you would do well to get in the habit of turning the memory protection back ON. This is your "safety" switch - as long as it's ON, there is no way you can accidentally erase a sound in memory. This protection will reset when you turn the machine off but there's no point to actually powering down the DX7II after every storage. Turning the memory protection back on is optional, but it's an option you should use - in the immortal words of Roger Powell, "use it or lose it!"

We're going to take " - Joe - " and store him in internal slot 42. If your DX7II currently has the Master Group (that is, a copy of the contents of bank 1 of the ROM cartridge) in its internal memory, then you can skip this next Exercise, since you are free to erase ANY of the sounds currently in memory. If, on the other hand, you have a voice in

* If you were editing voice B - in dual or split mode, when you go to single mode, voice A will automatically be put in the edit buffer. How do you then restore the edited voice B? Simple - here's the edit recall buffer to the rescue again! Just go back to edit mode, use the edit recall procedure as outlined above, and the edited voice B will now be back in the (single voice) edit buffer!

internal slot 42 that you want to keep, you'll have to run this next Exercise in order to "free up" internal slot 42. As mentioned earlier, you will be asked to erase one of your sounds in order to do this, and presumably you now know which one is to be sacrificed. If not, take the time now to go through your internal memory in order to find an expendable voice, and then run this Exercise.

Exercise 36
Moving voice data within the internal memory

1) Put your DX7II in single voice play mode and press the edit mode select switch. Press edit switch 14 repeatedly until you see the "Master Tuning/Memory Protect" display. Use the cursor switches in order to position the cursor over the "INT" parameter and press the "no" button in order to turn the internal memory protection OFF.

2) Return to single voice play mode and call up the internal voice you have decided to erase. Make a written note of the slot number.

3) If its status light is not already lit, press the "1-32/33-64" switch, followed by main switch 10. This will call up the voice stored in internal slot 42 (since 10 + 32 = 42). As we know, this has the action of making a copy of voice 42 in the edit buffer.

4) Since we have just been viewing an internal voice which we wish to store into the internal memory, we don't need to press the "internal" switch. However, you might want to get in the habit of doing so anyway, since you would need to do this if you were storing a cartridge voice into the internal memory.

5) Press and hold down the pink Store switch.

6) While continuing to hold down the Store switch, with your other hand, press the main switch corresponding to the slot number of the sound you wish to *erase* (which you wrote down in step 1 above). If the sound you wish to erase is in one of the 33-64 slots, precede this by pressing the"1-32/33-64" switch so that its status light goes on.

7) While continuing to hold down the Store switch, check the LCD and make sure that the slot number of the sound you wish to *erase* is currently being displayed. If it isn't, let go of the store switch momentarily, then re-depress it and select the correct slot number. When you are absolutely sure that the slot number of the sound you wish to erase is being displayed, press the "yes" button with your pinky (continuing to keep the store switch depressed). Warning: be careful, as this step will permanently erase that sound.

8) Press the edit mode select switch once again, followed by switch 14. Follow the general procedure outlined in step 1 above but this time turn the internal memory protection back ON.

9) Return to single voice play mode. If it is not already lit, press the "1-32/3-64" switch in order to turn on the status light above that switch. Press main switch 10 (which will call up internal voice 42) in order to confirm that the voice which was in slot 42 is still there. note that copying voice data over to another slot does not in any way alter the original data. The voice you had in slot 42 is still there, as well as being in the other slot you moved it to.

What this Exercise did, of course, was to duplicate the sound you wanted to keep into somewhere other than slot 42. This means that we can now erase slot 42 (since the good sound is now safely somewhere else), effectively freeing slot 42 to receive " - Joe - " without erasing a sound you want to keep. We arefinally ready to store " - Joe - " into a memory slot in our DX7II.

Exercise 37
Recalling and storing a sound

1) Press edit switch 14 and call up the "Master tuning/Memory protect" display. Turn the internal memory protection OFF.

2) Press edit switch 14 two more times in order to call up the "Recall edit" display. Place the cursor over the "Voice" parameter and press the "yes" switch twice in order to recall " - Joe - " into the edit buffer. Return to single voice play mode and play a few notes on the keyboard to confirm that this sound is indeed present in the edit buffer. note also that the LCD should display the " - Joe - " name. If anything other than this comes up, you've probably done something you shouldn't have. Don't panic, just go back and redo Exercise 30 once again.

3) Press the "internal" switch. This step, as in the last Exercise, can actually be skipped, since we are storing a sound into the internal memory - which was in fact the last memory we viewed. Do it, anyway, as it's still a good habit to get into.

4) Press and hold down the pink Store switch.

5) Continue to hold down the Store button through this entire step. If the "1-32/33-64" status light is not lit, press that switch in order to do so. With your right hand, press main switch 10, thereby entering "42" into the "Store data into memory " LCD display. With your left pinky, press the "yes" switch in order to confirm. Observe that the LCD shows "** Completed!" We have just stored " - Joe - " into internal slot 42, knowing, in this instance, that erasing the voice which was in slot 42 is okay to do since we just copied it elsewhere in the memory (Exercise 36 above).

6) Play a few notes on the keyboard and listen: " - Joe - " should be what you are hearing.

7) Return to edit mode and press main switch 14. Turn the internal memory protection back ON again.

8) Return the DX7II to play mode and call up any voices, internal or cartridge, that you like. note that when you call up internal voice 42, however, " - Joe - " reappears. At any point, press edit switch 14 to redo the Recall Edit procedure. Note that " - Joe - " is still in the edit recall buffer *as well as* in internal memory slot 42, and will remain there until the next time you make a change to a sound in edit mode.

Voices, of course, can also be stored (along with the aforementioned performance memories, and microtuning and system data) onto a RAM4 cartridge. The procedure for storing onto a RAM4 is almost exactly the same as storing into internal memory, with a few small differences. First of all, the RAM4 cartridge has an built-in hardware memory protection switch which must be physically switched OFF (by just clicking the switch) before any data can be written to it. This *hardware* memory protection works *in addition to* the software protection accessed from edit switch 14, as discussed earlier.

Secondly, the RAM4 cartridge must be *formatted* to receive a particular type of data. Formatting is something that must be done to all forms of storage media, be they cartridge or disk, before a computer can actually use that media. The formatting procedure writes certain "pointer" information, telling the computer where various areas of data are physically stored.

As mentioned briefly in Chapter Two, the DX7II RAM4 cartridges can be used to store any one of three different types of memory - voice/performance memory, microtuning memory, or fractional level scaling memory. We'll be talking about the latter two in future chapters - for now, we want to store only voice and performance memory. The point is that you will need to specify which type of memory you want to store on the cartridge when you format it - and you will definitely have to format it before you can use it!

The cartridge format procedure, like all other cartridge functions, is accessed with edit switch 15. Pressing this switch repeatedly will give you four LCD displays. **(see figure 8-15)**

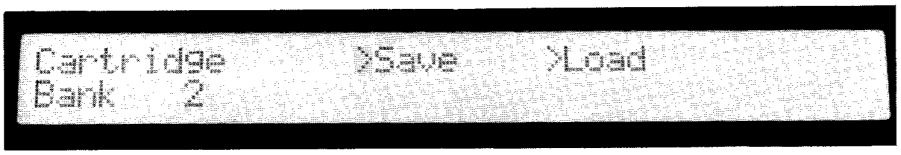


Figure 8-15(a)

Figure 8-15(b)

Figure 8-15(c)

Figure 8-15(d)

The first display (as shown in **figure 8-15(a)**, allows for the bulk transfer of data to and from the cartridge. We'll run an exercise shortly (Excercise 40, below) to try out the "save" and "load" parameters offered by this display.

The second display, which reads "Voice and Perf. " in the lower left-hand corner, is the one that we want now (the remaining two are used for formatting the cartridge to store microtuning or fractional level scaling data). If (and *only if*) you have a brand-new, never-used-before (and therefore unformatted) DX7II RAM4 cartridge, place it in the slot. DO not DO THIS IF YOU ARE WORKING WITH AN ALREADY USED RAM4 CARTRIDGE WITH DATA IN IT THAT YOU WISH TO KEEP. Press edit switch 14 and turn the software cartridge memory protection OFF and then turn the hardware protection (on the cartridge itself) OFF as well (you can do this in either order - just make sure you turn *both* memory protections OFF). Next, press edit switch 15 again. Move the cursor over the "Format" display and press "yes". The LCD display will faithfully ask, "** Are you sure?". Make sure that you're sure and press "yes" again. At the completion of the process, the LCD will tell you, "Completed!".

Now we are finally ready to store " - Joe - " into the RAM4 cartridge. As with the internal memory, we will first need to use edit switch 14 in order to turn the software cartridge memory protection OFF, in addition to turning the hardware memory protection switch to the OFF position. Let's try it:

Exercise 38:
Storing a single voice to a RAM4 cartridge

1) Insert a properly formatted RAM4 cartridge (as per the instructions above unless it already has voice and performance data stored to it) into the slot and put your DX7II into single voice play mode. Press the "cartridge" switch in order to access the cartridge memory, and press any or all of the 32 main switches in order to go through the voices currently stored on your RAM4 cartridge (if it has just been freshly formatted, all the voices will be called "INIT VOICE" and will be pure sine waves). If this is a RAM4 cartridge which has previously been used for voice storage, find a voice that you are willing to erase. Make a written note of the slot number.

2) Turn the RAM4 cartridge's hardware memory protection switch OFF. Press the edit mode select switch, followed by edit switch 14 as many times as necessary to call up the "Master tuning/Memory protect" display. Use the cursor switch to position the cursor over the "CRT" parameter and press the "no" switch in order to turn the software cartridge memory protection OFF.

3) Return to single voice play mode and press the "internal" switch. If the status light above the "1-32/33-64" switch is not currently lit, press the switch in order to do so. Press main switch 10 in order to call up internal voice 42 (yes, it's " - Joe - " again). Play a few notes on the keyboard to confirm that " - Joe - " is in fact now in the edit buffer.

4) Press the "cartridge" switch. Unlike the last two Exercises, this memory selection step is necessary here because we will be storing our sound in a memory different from the one we were last viewing (the internal memory).

5) Press and hold down the Store switch, and, with your other hand, press the main switch corresponding to the cartridge memory slot of the voice you previously decided to erase (in step 1 above). If you had decided to erase a sound in the 33 to 64 group, press the "1-32/33-64" switch first. Observe that the LCD shows the "Store voice to memory?" prompt, followed by the number of the cartridge slot you have just chosen. If you are sure you selected the correct one (the one you actually wanted to erase), press the "yes" button (if you made a mistake, let go of the Store button momentarily, then re-depress it and select the correct slot number). Observe the "Completed!" prompt in the LCD, showing you that " - Joe - " has been written to the RAM4 cartridge.

6) Press the edit switch once again and use the "yes" button to turn the cartridge memory protection back ON.

7) Go back to single voice play mode and press the "cartridge" switch in order to access the voices in the RAM4 cartridge. Go through these voices and note that " - Joe - " is now stored in the slot that you designated.

8) Experiment by redoing this Exercise, but this time attempt to store " - Joe - " on a ROM cartridge. note that when you reach step 5, the LCD will read "Memory Protected" and that you cannot continue. The ROM cartridges are *permanently write-protected*, and for that reason their contents are uneraseable.

9) Experiment further by moving data back and forth between the RAM4 cartridge and the internal memory. note that, apart from the "Under Writing !" display, both memories work the same way.

At the moment, of course, " - Joe - " is living in three separate places. We've just put him into a RAM4 cartridge slot; he's still in internal memory slot 42; and he's also still in the edit recall buffer.

Thanks to the edit buffer, we can actually move data *between* cartridges! This involves copying a voice from a cartridge into our edit buffer (done automatically by simply calling up the voice) and then writing it onto another (RAM4) cartridge. If it wasn't for the edit buffer, there would be no way of accomplishing this. Let's try storing the original "Celeste" sound in our RAM4. For the sake of simplicity, we'll overwrite " - Joe - " in the RAM4:

Exercise 39
Moving voice data from cartridge to cartridge

1) Insert your ROM cartridge into the slot. Put your DX7II into single voice play mode and press the "cartridge" switch in order to access the cartridge memory.

2) Press the edit mode select switch, followed by edit switch 15 repeatedly if necessary in order to call up the "Voice and Performance" display. **(see figure 8-16)**

Figure 8-16

3) Position the cursor over the "Bank" parameter and select bank 2. Return to single voice play mode and press the "1-32/33-64" switch if the status light above it is not already lit. Press edit switch 30 in order to call up voice 62 (since 30 + 32 = 62). "Celeste" is now, of course, in your machine's edit buffer. Play a few notes in order to confirm this.

4) Remove the ROM cartridge and insert a RAM4 cartridge into the slot. Put your DX7II back into edit mode and turn the both the software and hardware cartridge memory protections OFF.

5) Return to single voice play mode. Note that we are still accessing the cartridge memory but DO not press any of the 32 main switches.

6) Press the "cartridge" switch, simply to develop good habits - this step is not strictly necessary since we will be storing data into the same (cartridge) memory we were last viewing.

7) Press and hold down the Store switch. With your other hand, press the main switch corresponding to the RAM4 slot in which you previously stored " - Joe - ". note again that if you previously put " - Joe - " into a voice slot higher than slot 32, you will need to first press the "1-32/33-64" switch if its status light is not already lit. Observe that the LCD shows the usual "Store data to memory" prompt, followed by the slot number you have just chosen. If this slot number is correct, press the "yes" button with your pinky to confirm (if the slot number is wrong, simply let go of the Store switch momentarily, then re-depress it and choose the correct slot). Warning: This will permanently erase a sound, so be sure you select the right slot number before you press the

"yes" switch. If you can't remember which RAM4 slot held " - Joe - ", return the DX7II to cartridge single voice play mode, and go through the various slots. When you've located it, make a written note, and go back to step 1 above.

8) Return to edit mode and press edit switch 14 in order to turn the cartridge memory protection back ON.

9) Go back to cartridge single voice play mode and go through the various voices in your RAM4 cartridge. note that the slot that previously had " - Joe - " in it now contains "Celeste". Play a few notes on the keyboard to confirm.

Moving voice data around within the DX7II, then, is a fast and simple procedure once you get used to hitting the switches in precisely the right order. The contents of a memory, be it internal or cartridge, can easily be rearranged to suit your particular needs. For example, if you're doing a live performance with your DX7II and you know the first sound of the night will be ROM 20, the second sound RAM4 58, and the third sound Internal Voice 12, don't give yourself fits by having to continually swap cartridges and memories! Move the first sound into internal slot 1, the second into internal slot 2, and the third into internal slot 3. At your gig, all you'll have to do is call up your internal memory and then just increment the numbers - a much easier way of accomplishing the same thing! Your RAM4 cartridge can be used the same way.

Individual performance memories can be saved to internal or RAM4 cartridge memory with the same basic procedure, except that the "1-32/33-64" switch cannot be activated since you can only store up to 32 performance memories. However, in addition to storing individual voices one at a time, the DX7II allows for what is called *bulk data storage;* that is, the storage of an entire *bank* of memory en masse.

What is in this so-called "bank" of memory? As we explained in Chapter Two, this would contain 64 voices and 32 performance memories, as well as 2 microtuning memories (to be discussed in Chapter Fourteen) and a system memory (to be discussed in Chapter Fifteen). Bulk data can be sent to the DX7II's internal memory, to a properly formatted RAM4 cartridge, or, if you are using the "FD" model of this instrument - to the built-in floppy disk drive. In addition, such transfers can be sent, via MIDI, to any outboard computer which was been programmed to understand and use such data - or to another DX7II. Voice and performance data generated in the DX7II can not be transferred to a DX7, since so many more edit parameters have been added in the new instrument. This is what is meant by *upward compatibility* - it is a one-way street (since DX7 voices, as we mentioned earlier, CAN be sent via MIDI to the DX7II).

Since not all of you have the "FD" model, let's start with an Exercise that demonstrates how to do bulk data transfers to internal and RAM4 cartridge memory. We'll follow this with a separate Exercise to demonstrate the disk storage and retrieval functions.

Do not run the following Exercise unless you have a fresh, unused RAM4 cartridge (or a RAM4 cartridge you're willing to permanently erase) available for backup purposes.

Exercise 40
Bulk data transfers to internal and RAM4 cartridge memory

1) Insert a RAM4 cartridge into the slot. If you are working with a fresh, unused RAM4 cartridge, *format* it using the procedure outlined

earlier in this chapter. If the cartridge has previously been used for voice and performance memory storage, you won't need to format it, but be absolutely certain that you are willing to permanently erase the data currently stored in it.

2) We'll start by copying ("saving") the entire contents of your internal memory to this cartridge. This is known as *cartridge save*. In order to do this, both cartridge memory protections must be first be turned OFF: Press edit switch 14 repeatedly if necessary to call up the "Master Tuning/Memory Protect" display and turn the "CRT" memory OFF. Then turn the hardware memory protection switch on the cartridge itself to the OFF position.

3) Press edit switch 15 repeatedly if necessary in order to call up the "Bank" LCD display. **(see figure 8-17)**

Figure 8-17

4) Use the cursor switch in order to position the blinking cursor over the "Save" display and press the "yes" button. The LCD display will ask, "Are you sure?". Press the "yes" button again. When done, the LCD will read, "Completed!". The entire contents of your internal memory have now been copied and stored into your RAM4 cartridge - and the whole procedure took just seconds!

5) Let's pretend we're all from Missouri and double-check. First of all, turn both cartridge memory protections back ON. Then return your DX7II to single voice play mode and press the "cartridge" switch in order to access the memory of the RAM4 cartridge. Press any or all of the thirty-two main switches in order to confirm that all of your internal sounds are now in the cartridge. If you like, you can even go to Performance play mode and call up the thirty-two performance memories stored in the RAM4 cartridge. They, too, will be the same as the ones in your internal memory. Press the "internal" switch in order to access your internal memory once again and press any or all of the thirty-two main switches in order to confirm that nothing in your internal memory has been erased or changed in any way - all we did was to make a *copy* of the internal memory data into the RAM4 cartridge.

6) Now that we have our internal memory backed up (that is, we have made a complete copy of it into our RAM4 cartridge), let's now try *loading* bulk data into the internal memory. This will, of course, have the effect of completely erasing all the data that is currently in the internal memory - but now that's OK! Since the RAM4 cartridge has just had written to it an exact copy of the internal memory, let's be a little different and load in a bank from a *ROM* cartridge. Accordingly, remove the RAM4 cartridge from the slot and insert your ROM cartridge. First of all, we'll have to turn the internal memory protection OFF in order to store data there, so press edit switch 14 and follow the usual procedure in order to do so.

7) Let's arbitrarily load in the contents of the ROM cartridge's second bank of sounds. Press edit switch 15 and select bank 2 (from the "Bank" LCD display).

8) Next, we need to inform the DX7II that we want to do a bulk transfer from cartridge to internal memory. Press the right cursor switch twice in order to position the cursor over the "Load" parameter, and press the "yes" switch. The LCD will now ask, "Load without system?" You are being asked if you want to load in the *system* memory as well. This is information that largely has to do with MIDI parameters (to be discussed in greater detail in Chapter Fifteen), and is of little use to us right now, so we can answer the question affirmatively (if, on the other hand, we wanted to load in this data as well, we would answer "no").

9) Observethat the LCD display now asks us, "** Are you sure?". Since we are sure, press the "yes" button once more. When the task is done, the display will read, "Completed!". The entire contents of bank 2 of your ROM cartridge are now stored in your internal memory, and whatever was previously in your internal memory is gone.

10) Let's check it out to be sure. Return your DX7II to internal single voice play mode and press any or all of the thirty-two main switches. note that all of the voice data in your internal memory has been replaced with an exact copy of the data previously stored in bank 2 of your ROM cartridge. Go to Performance play mode and note that the performance memories have been overwritten as well. Because the data in a ROM cartridge is uneraseable, bank 2 of your ROM has obviously remained unchanged (if you are a third-generation Missourian, you might want to check this out, too, though I can assure you it's true!).

11) Finally, let's put the data that originally was in your internal memory back. Insert the RAM4 cartridge you used earlier in this Exercise back into the slot, and repeat steps 8 through 10. You should now be right back where you started - except that you now have a backup copy of your internal memory on your RAM4 cartridge.

Disk memory

As mentioned earlier, data can also be read from or written to the built-in disk drive if you are using the "FD" model of the DX7II. Those of you not using this model can skip ahead to just after the upcoming Exercise (Exercise 42). It is important to note several things about the use of this disk drive as a storage medium. First of all, the built-in disk memory can store any kind of data that the RAM4 cartridge can store (voice/performance, microtuning, or fractional level scaling) - or it can store externally received MIDI data of any kind (in its "MDR" - for "MIDI Data Recorder" mode). However, it can ONLY store bulk data - individual voices, performance memories, microtunings, or fractional level scalings cannot be written to disk. We will see shortly, however, a technique by which we can *update* the disk with data containing individual voice or performance memory alterations. Finally, data stored in the disk cannot be accessed immediately - in order to use any data stored on disk, you must first load it into internal or cartridge memory.

All the disk functions in the DX7IIFD are accessed with edit switch 16 (appropriately labeled "DISK"). When you repeatedly press this switch, you will see four LCD displays. **(see figure 8-18)**

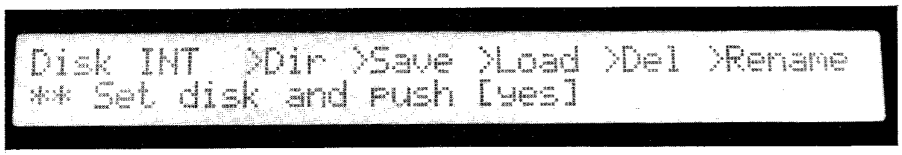


Figure 8-18(a)

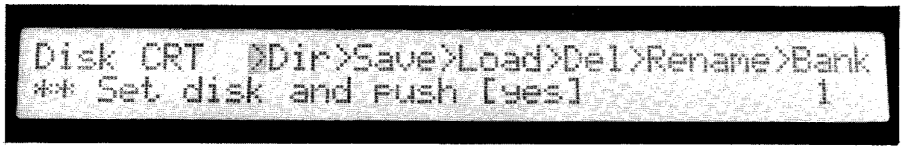


Figure 8-18(b)

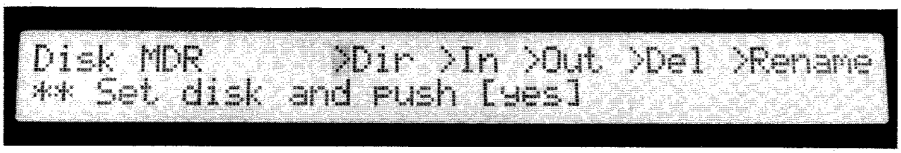


Figure 8-18(c)

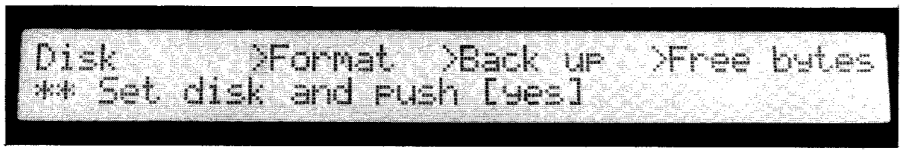


Figure 8-18(d)

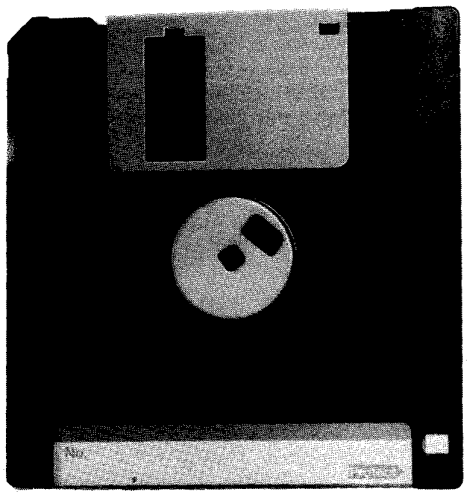


Figure 8-19

The DX7II uses 3-1/2" double-sided, double density micro floppy disks. They are relatively robust, but, as the owner's manual suggests, they should be treated with care and not stored in direct heat or sunlight nor near any magnetic signals (like speakers or TV sets, for example). Like the DX7II RAM4 cartridge, the disks have a hardware memory protection built in. **(see figure 8-19)**

This switch, obviously, must be in the OFF position whenever you write data to the disk, and should be left ON at all other times to prevent accidental erasure of data. The disk is inserted into the drive (on the left-hand side of the machine) with the label face up and the metallic shutter facing in. When inserted, a small button pops out from the drive. This is used to eject the disk when you are done accessing it. Disks should not be left in the drive for extended periods of time, and definitely not when you turn the power off. Be careful never to press the eject button when the LCD display reads "BUSY", either, or you risk doing serious damage to your disk, the drive, or both. Eject it only when the drive is idle.

Data is stored in the micro floppy disk in what are known as files. These are similar to the *banks* that we already encountered in the ROM cartridge, in that they store complete bulk data. However, the disk allows for relatively enormous amounts of data storage, as over 40 files can be stored on a single disk! A file may contain, as mentioned above, any type of data that can be stored in a RAM4 cartridge - and the data can come from either the DX7II's internal memory, its cartridge memory, or from an external source, via MIDI. This explains three of the four LCD displays given by edit switch 16. The "INT" display, (as shown in **figure 8-18(a)**) allows you to access files of data which were stored from, or are to be written to, the internal memory. The "CRT" display, (as shown in **figure 8-18(b)**) allows access to files which were stored from, or are to be written to, the cartridge memory. And, finally, the "MDR" display, (as shown in **figure 8-18(c)**)allows access to files which were obtained from, or are to be sent to, external devices (like other synthesizers, sequencers, or drum machines) via MIDI. These functions will be covered in great detail in Chapter Fifteen.

All of these displays allow you to not only read and write these kinds of files ("Save" and "Load", or, in the case of MDR files, "In" and "Out"), but they also allow you to obtain a catalog, or *directory* of files in that category (the "Dir" function), to delete unwanted files (the "Del" function), or to rename files (the "Rename" function). As we will see shortly, files are originally named when using the "Save" (or "In") function.

But before we can store any data to our disk, we need to format it - just as we needed to format our RAM4 cartridge. The only difference is that the disk is capable of storing many different kinds of files, simultaneously, so we don't need to specify how we plan on using it. The format command is found in the plain "Disk" display. (as shown in **figure 8-18(d)**)

This display also allows you to easily make backup copies of your disk (the "Back up" function, natch), and also to easily see how much memory is available on the disk, in *bytes* (with the "Free bytes" parameter). A byte is a unit of computer data. These will be expressed in terms of "k", for *kilobytes,* where one kilobyte is equal to 1,000 bytes. A blank disk will store over 700 kilobytes of data, with a single internal or cartridge file occupying about 16k, and an MDR file occupying up to 20k. We'll try this out when we run our upcoming Exercise.

As with the cartridge format procedure, formatting a disk will of necessity erase everything on that disk, so be sure to do this only with a fresh, unused disk, or one whose contents you are certain are disposable. Let's run an Exercise now to try out some of these disk functions:

Exercise 41
Reading from and writing to disk memory ("FD" model only)

1) Place a micro floppy disk in your disk drive. If this disk already has data on it that you wish to keep, go straight to step 2. Otherwise, if the disk is thus far unused, or if it contains data that you are absolutely certain you want to erase, continue with this step. Put your DX7II into edit mode and press edit switch 16 repeatedly until you see the "Disk" display, as shown in figure 8-24 above. Use the cursor switches to position the cursor over the "Format" display. Press the "yes" button. note that the LCD now asks, "Are you sure?". If you are, press the "yes" button again. The LCD now reads "Formatting", and you will hear the drive click and then buzz for about a minute. The number in the display shows you which *sector* (area of the disk) is currently being formatted. When the procedure is complete, the display will read, "Completed!".

2) If you are not already viewing the "Disk" display accessed by edit switch 16 (as shown in figure 8-24 above), press that switch repeatedly until you see that display. Use the cursor select switches to position the cursor over the "Free bytes" parameter. No, the DX7II won't byte - the drive will click and buzz and, in a second or two, the LCD will show you how many kilobytes are available on your disk and how many files are free.

3) Let's now try storing the contents of your internal memory to disk. Press edit switch 16 once in order to call up the "INT" display. Use the cursor switch to position the cursor over the "Dir" parameter. The LCD now prompts you to "Set disk and push (yes)". Respond by pressing the "yes" switch. Once again, the drive will click and buzz for a second or two and the display will then show the name of the highest

numbered internal file it was able to find. If you have just formatted this disk, of course, there will be no files on it as yet and you will instead see "File 1" and a series of eight dots, indicating that the file has no name. **(see figure 8-20)** If you are working with a previously used disk, you can press the "yes" and "no" buttons in order to view the names of the different internal memory files on your disk.

Figure 8-20

4) Press the right cursor switch once in order to position the cursor over the "Save" parameter. The LCD will now show ten dots and ask you to input a filename. **(see figure 8-21)** If you are working with a previously used disk, you must be extremely careful here. The critical thing to Observeis the file *number*, found immediately under the "INT". Whenever you save to a particular disk file number, the DX7II will ask you to rename that file. Even if you enter a completely different name, *it will still overwrite the contents of that file*. This is very different from the way that most computers work - where the only time a file gets overwritten with new data is when you use the *same* name. Here, it doesn't matter whether the new name is the same or completely different - if you tell the DX7II to write into a particular file number, it will blindly do so, regardless! Be very careful here as it is unfortunately quite easy to overwrite an important file. If you do so, there is no way to recover that data - it's gone for good. Therefore, keep your eye on the file *number*. If you don't want to overwrite any previously stored data, simply use the "Dir" command to call up a blank (or *null*) file - one with eight dots (". . . . . ") for a name. This is accomplished with the "yes" switch, by simply stepping once beyond the highest existing (named) file. Therefore, for the purposes of this Exercise, press the left cursor switch in order to go back to the Dir command. Press the "yes" switch as many times as the display will allow until eventually you call up a null file. If you have just formatted, of course, the first and only file will be null and will be as far as you can go. Now press the right cursor switch in order to return to the Save command.

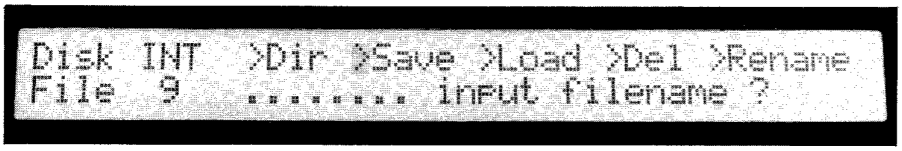


Figure 8-21

5) You are being prompted to name the null file. This is accomplished in the same way that we named a voice earlier. Press and hold down the character (edit) switch, and you will find that all the other switches on the machine are now active for their character functions. Once again, you can use upper- or lower-case letters. Unlike the voice name, you are limited here to eight characters - and you cannot use spaces or periods in the name. When you are done naming the file, let go of the character key and press the "yes" switch. If you get a "** bad filename!" prompt (grouchy machine!), then you undoubtedly included either a space or a period in the name. You'll have

to then press the "no" button in order to rename the sound. When you come up with a name that's acceptable to our finicky microprocessor, you'll be asked the usual "Are you sure?" question. Answer "yes", and you'll get a "**BUSY Now Executing!" display for a moment or two. No, no one is being done in, but the drive is spinning and your internal data is being written to disk. When this "BUSY" display is active, you must *under no circumstances* attempt to physically eject the disk by pushing the eject button. When the process is complete, you'll get a "** Completed!" display. You've now successfully stored the entire contents of your internal memory to disk!

6) Now let's store both banks of our ROM cartridge to disk. Insert your ROM cartridge into the slot and press edit switch 16 once in order to VIEW the "CRT" display. Position the cursor over the "Dir" parameter. Once again, you'll be shown the highest used file number and name. If you've just formatted the disk, you'll be viewing file 1, with the null (eight periods) name. If there are already files of cartridge memory on the disk, use the "yes" button in order to call up the first null file.

7) The "Bank" number on the far right-hand side of the display (**see figure 8-22**)will currently show the default bank number, bank 1. This is used when storing data to disk from ROM cartridge, as we are doing here, since the ROM cartridge (unlike the RAM4 cartridge) contains four banks. If you were saving data to disk from a RAM4 cartridge, the number would still be 1, but you wouldn't be able to change it. We'll begin by storing bank 1 of our ROM, so don't change it just yet. Instead, press the right cursor switch once in order to position the cursor over the "Save" parameter. As in the last step, you'll be asked to enter a valid filename. When you have done so (pressing and holding down the character key), press "yes" and then "yes" again (in answer to "Are you sure?"), and your data will be stored! Following this, you'll be automatically put back to the "Dir" parameter. Press the "yes" switch once in order to move up to a null file. Then press the right cursor switch five times in order to position the cursor over the "Bank" parameter, and press the "yes" switch once in order to be able to save bank 2 of the ROM cartridge. Repeat the same "Save" procedure in order to do so.

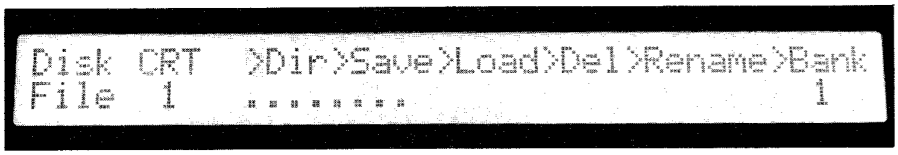


Figure 8-22

8) Now let's try *loading* data from the disk to the internal memory. Since we have already *saved* the contents of your internal memory to disk (in steps 2 through 5 above), we can do so with abandon - because remember that loading bulk data to the internal memory will of necessity *erase* the current contents of that memory. First of all, to write any data to the internal memory, we know that we have to turn the internal memory protection OFF. Therefore, press edit switch 14 and do so. Then return to switch 16 and call up the "INT" display once

again. * The cursor will now automatically be over the "Dir" display, as it always is whenever you call up a new disk memory display. You will be prompted to "Set disk and push (yes)". As long as the disk is still in the drive, simply push the "yes" switch and you will instantly see the name and number of the highest used file. If you have been faithfully working through this Exercise, you'll be able to find the file you created in step 5 above, along with any previously created files (if you're working with a previously used disk). Use the "yes"-"no" buttons to find a file that you'd like to transfer into your internal memory.

9) Press the right cursor switch twice in order to position the cursor over the "Load" display. The display will ask, "Load without system?". As when we loaded bulk data from a cartridge earlier, we are being asked if we want to load new *system* data, which, as mentioned earlier, is concerned mainly with MIDI parameters. Since we don't need to worry about that right now, simply press the "yes" switch in order to answer affirmatively. You will be asked "** Are you sure?", and, if you are, simply press the "yes" button again. A click, a buzz, and a moment or two later, your internal memory will be completely loaded up with the contents of your selected disk file.

10) Finally, if you have a new or eraseable RAM4 cartridge available, let's try loading a Cartridge disk file into the RAM4. Again, you'll need to turn both cartridge memory protections - hardware and software - OFF before you can write any data to the cartridge. Therefore, flip the switch on the cartridge for the hardware protection, and use edit switch 14 to turn off the software protection. Then press edit switch 16 once more in order to call up the "CRT" display. As in the last step, the cursor will automatically be positioned over the "Dir" parameter. If the disk is still in the drive, simply press "yes" in order to get a directory of your Cartridge disk files. The files you saved earlier in this Exercise should be there, along with any other files that might have previously been on the disk (assuming that you didn't originally format this disk back in step 1). Select the file that you wish to load to the cartridge.

11) Press the right cursor switch twice in order to position the cursor over the "Load" parameter. Press the "yes" switch. The LCD will show you the format of the RAM4 cartridge currently in the slot and will ask "OK?". The cartridge format *must* be the same as the file type in order for this to work. For example, if your Cartridge disk file contains voice and performance data, the RAM4 must be formatted to receive that data. Similarly, if your Cartridge disk file contains microtuning data, your RAM4 must be formatted for microtuning information. If the formats agree, simply press the "yes" button, and you will be asked "** Are you sure?" once again. An affirmative reply will complete the process. If the formats are mismatched, you will have to press edit switch 15 in order to correctly reformat your cartridge before you can proceed any further.

Although you cannot store individual voices or performance memories to disk, you can *update* your disk files easily by simply overwriting them *carefully and selectively*. For example, let's say you

*You can only load Internal disk files to the internal memory, just as you can only load Cartridge disk files to RAM4 cartridge memory. What if you wanted to load the contents of a Cartridge file from disk to internal memory? You couldn't do it - not directly, anyway. What you'd have to do use edit switch 15 in order to load the bulk data from the cartridge to the internal memory first (as we did in Exercise 41 above), and then from the internal memory to an Internal disk file (as we did in step 1 above).

load Internal disk file 8 into your internal memory and then tweak a couple of edit parameters in one of the voice. Simply save that voice back to the same internal memory slot and then write the internal memory back to Internal disk file 8, keeping or changing the original file name! Although ALL of the data gets rewritten to disk, you only changed a little bit of it, so the end result is, for all intents and purposes, an updated file. Alternatively, you could save your new bank to a *different* file number, which would allow you to keep and access your old file along with your updated file. Here's a useful tip: incorporate the date of the bank creation into the file name (that is, name your files "11-15" or "String3-6", etc.) so you can keep better track of what is in a particular file. Written records won't hurt, either!

The only disk functions which we didn't try out in Exercise 41 were the file Delete and Rename functions, both of which are self-explanatory. As mentioned earlier, Chapter Fifteen will contain a description of the disk MDR functions.

Now let's get back to some programming! As you work through this book, or as you work on your own, you should continually *save* your work by writing it to memory - internal, cartridge, or disk memory - your choice. Your most recent edit work will, of course, continually be held in the edit recall buffer, but remember, it will only hold the most *recent* work. Sometimes you will need to recall data that *wasn't* recently changed. Unless you saved that data to memory, you're out of luck. As every session programmer knows, the saddest words in the world are, "you know, I liked it better the way you had it two minutes ago. ". Those two minutes could be an eternity if you've changed several parameters and neglected to save as you went along.

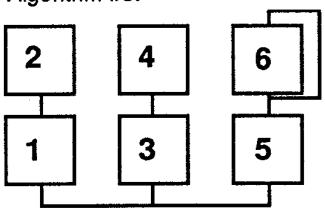
Or at least that's the way things used to be B.D. (Before DX7 - the Stone Age of synthesis). Will wonders never cease? The DX7II actually provides us with yet another amazing piece of memory, called *compare mode*, which allows us to do real-time comparisons between a modified sound and the original!

Compare mode can only be accessed from edit mode - that's logical since the only time we would ever actually need to make a comparison is when we are actually making some kind of change to a sound. If you press the edit mode select switch while *already* in edit mode, the DX7II will enter *compare mode:* the LED will begin blinking, and the original sound will temporarily be restored into the edit buffer. Furthermore, the LCD will actually display the original data for whatever parameter you call up! This is extremely useful, particularly if you've made many changes to a sound. Suppose, for example, you've modified twelve different edit parameters and you're satisfied with what you've done. Being naturally inquisitive, you decide to try one last change, perhaps to the output level of a particular operator, and it turns out awful! If you can't remember the original output level setting, you're in a lot of trouble, since it seems as if you've ruined the sound. Going back to play mode and calling up the original sound won't do you any good, since the twelve previous changes you made would be lost. Using the Recall Edit procedure won't work either, since it will restore all thirteen changes - not just the first twelve that you liked.

Here's where compare mode gallops to the rescue. Simply call up the Output Level parameter, press the edit switch again in order to enter compare mode, and the LCD will actually show you what the original output level was for that operator! Returning to edit mode by pressing the edit switch again will allow you to then restore the original data by reentering that same value with the data entry section. Let's try it out:

Original Celeste

Algorithm #5:



Modified Celeste

Algorithm #18:

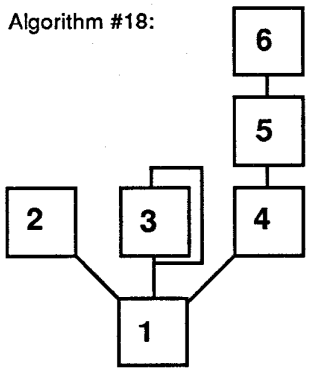


Figure 8-23

Exercise 42
Compare mode

1) Put your ROM cartridge in the slot. Place your DX7II in single voice play mode and press the "cartridge" switch in order to access the cartridge memory. Call up the "Celeste" voice from bank 2 of your ROM cartridge (it's voice 62, as if you didn't remember. . .).

2) Press the edit mode select switch and press edit switch 7 (ALGORITHM). Change over to algorithm # 18. Play a few notes and listen (Audio Cue 42A). note that the character of the sound has changed drastically, due to the new configuration. **(see figure 8-23)** Observe that a decimal point appears next to the number displayed in the LED, as this is the first actual change we are making to this sound. This modified sound and all future changes we make to this sound is now in the edit *recall* buffer.

3) Press edit switch 8, followed by edit switch 3, in order to VIEW operator 3. Change the Fine value for this operator to 1.47, yielding a highly dissonant and bell-like sound. Listen (Audio Cue 42B).

4) VIEW operator 4 and change its Coarse value to 0. 50. Move the cursor to the Fine parameter and change this to 0. 85. Play a few notes and listen (Audio Cue 42C). The sound is now extraordinarily unpleasant and almost industrial - a far cry from the delicate "Celeste" we started out with!

5) Press the edit switch in order to enter compare mode. note that the LED begins blinking rapidly.

6) Play a few notes on the keyboard and listen (Audio Cue 42D). Note that we are once again hearing "Celeste", a far cry from our modified sound!

7) While still in compare mode, observe the LCD, which is showing you that the original ratio number of operator 5 was 12.00. Press edit switch 7 and observe that the original algorithm was #5. Press the edit mode select switch once again in order to return the DX7II to edit mode and note that the LCD once again shows us the modified algorithm (#18). Press edit switch 8 and observe that we once again see the modified ratio number (0.85) for operator 5.

8) Use the data entry slider to change this ratio number back to its original value of 12.00. Play a few notes on the keyboard and listen (Audio Cue 42E). note that we have restored the sound only to the point we were at in step 4 above, and not all the way back to the original "Celeste" sound.

9) Experiment by modifying the new sound in various different ways, using compare mode to show you what the original settings were. If you're happy with the end result, SAVE the sound to a slot in your internal or cartridge memory.

The real value of compare mode, then, is that it allows you to travel down the road of modifying a sound and know that you can never stray too far. If you've made fifty changes to a sound and the fifty-first seems to ruin it, compare mode will let you restore that one parameter without having to restore the first fifty.

Notwithstanding all that, there's still a very good argument for saving your work periodically. The reason is simple - if you save your modification, then you can compare to the modification without having to compare to the original! Also, note that initializing a sound in the edit buffer and then building from that point on gives you no useful basis for comparison: the DX7 will *not* compare to the initialized default settings,

but instead just to the original sound that was in the memory slot that you happened to be accessing before initialization. If you are creating a sound using the initialization procedure, then, you will want to save an early version of it somewhere in memory so that you're not simply comparing to something totally irrelevant. Finally, remember that you cannot *change* any data while in compare mode: changes to edit parameters can only be made while in edit mode. In other words, if the LED is blinking, the data entry section will be inactive - in order to reactivate it, you will need to return your DX7II back to edit mode by pressing the edit switch again.

Now that we know how to name, store, and compare data, let's return to our discussion of the edit parameters that shape the sound in meaningful ways. Specifically, we'll move on to the devices that change our sounds dynamically over time: the *envelope generators*.

Chapter Nine ◇
The Envelope Generators

In Chapter One, we learned that all sound is composed of three parameters: volume, pitch, and timbre - and that typically each of these parameters will *change* throughout the duration of a sound. We have made great strides in unraveling the mysteries of the DX7II, but up until this point every sound we have created with this instrument has been static and unchanging. We have learned how to specify the volume (carrier output level), pitch (pitch data inputs), and timbre (modulator output level and frequency ratio) of a sound, but not yet how to *vary* them over time.

The first, and most important component in the DX7II which allows us to do this is the *envelope generator*. Remember our diagram of the operator? **(see figure 9-1)**

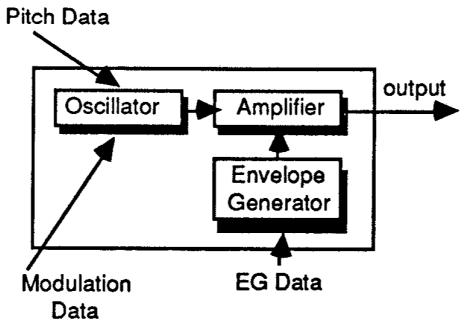


Figure 9-1

Each operator has its own, totally independent envelope generator (or EG for short), and *the actions of one EG have no bearing at all on the actions of any other.* We will be issuing instructions to each EG via the *EG data input,* in order to tell it what to tell its amplifier.

The sole purpose of the EG is to cause an *aperiodic* change of some kind to the sound over time. The word "aperiodic" simply means "non-repetitive", and the way that an envelope generator in any synthesizer works is: you press a key on the keyboard; the EG does its thing once and once only; and that is that, until you press another key on the keyboard. We'll learn in the next chapter that the DX7II also has another device, the *LFO,* that allows you to effect a *periodic,* or repetitive change over time.

Specifically, what the EG is altering is operator *output level.* **Figure 9-1** above clearly shows that the operator's EG sends its instructions directly to the amplifier, telling it how much signal from the oscillator to pass at various times, and how to vary that level. The end result of these commands will be to continuously change the amount of signal leaving the operator (that is, its *output level*). It is vital that you understand this concept - if you don't, take a moment to let it sink in before proceeding any further.

So, precisely how will these instructions from the EG affect the sound? Well, Cardinal Rules One and Two tell us that if we vary the output level of a carrier, we will get a volume change, and if we vary the output level of a modulator, we will get a timbral change. Keeping these Rules in mind, it's clear that an EG living inside a carrier will change the *volume* of that system over time, and that an EG living inside a modulator will change the *timbre* of that system over time. Of course, it's the algorithm that decides which operators are used as carriers and which as modulators, so that the EG of, say, operator 2, would alter part of the volume of a sound if algorithm #32 were being used, **(see figure 9-2)** but would affect the timbre if algorithm #5 were being used. **(see figure 9-3)**

Algorithm #32:

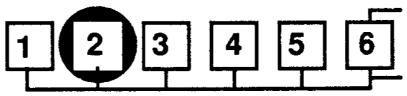


Figure 9-2

Algorithm #5:

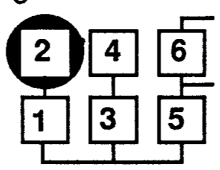


Figure 9-3

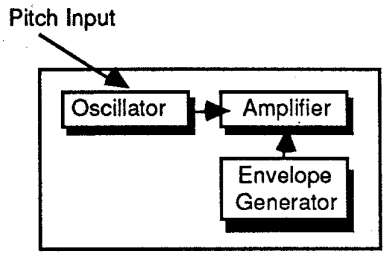


Figure 9-4

Note that neither a carrier EG nor a modulator EG can in any way affect the *pitch* of a sound since neither has any bearing at all on the pitch data inputs. **(see figure 9-4)**

We will learn shortly that the DX7II provides us with a separate mechanism (called the *pitch EG*) for varying the pitch over time. For now, though, let's concentrate on the individual operator EGs, which will aperiodically vary the volume or timbre of a sound over time.

In order to do its thing, the EG needs to know three important facts:

1) *When* you strike a key,

2) When you *let go* of the key, and

3) *How long* you held the key down.

These bits of information are transmitted within the DX7II by means of what in computerese are known as *flags*. When you strike a note on the keyboard (any note at all, it doesn't matter which), the DX7II generates a specific binary number called a *key on* flag. This is simply a "yell" to the six operator EGs, very much like someone waving a flag (hence the term), telling them that a key has been struck. As soon as the EGs receive this flag (a process which takes only a millionth of a second or so), they know to begin going through their paces. When you let go of a key on the keyboard, the DX7II generates another flag, called a *key off* flag. This lets the six EGs know that it's time to *stop* doing what they're doing and begin to return to their starting points.

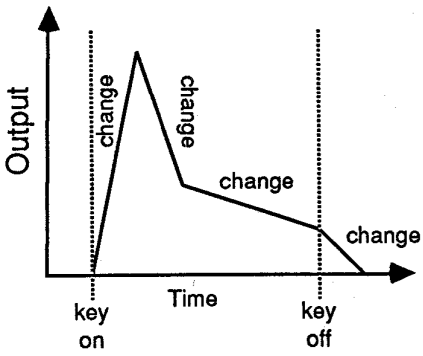


Figure 9-5

Once again, what the EGs do is change operator output level over time. We can graph these changes, as in **figure 9-5.**

The microprocessor in the DX7II has a small counter (or "clock") in it that is constantly ticking away, so it is able to calculate the amount of time that transpires between the "key on" and "key off" flags. In this way, the EGs get the three sets of information posed above, namely:

1) *When* you strike a key - "key on" flag is generated.

2) When you let go of the key - "key off" flag is generated.

3) How long you hold the key down - amount of time lapsed between "key on" and "key off" flags.

All of this happens quite automatically within the DX7II, and very quickly, so that even if you play a lot of notes very quickly, the EGs will be able to *track* what you're doing. Remember, computers are so incredibly fast that what we as humans consider "very quickly", they consider ultra-slow-motion.

The "mechanics" of how the DX7II envelopes work is really very straightforward, but this seems to be the area of the instrument that most users have problems with. The first point that must be made is that using the EGs is absolutely mandatory; you can't avoid them, so you might as well master them! The amplifiers inside the operators will not pass any signal at all (which means: no sound!) if they do not receive instructions from the EGs. The second point to be made is that since the EGs take what would otherwise be static sounds and turn them into *dynamic*, "real" sounds, their contribution is essential!

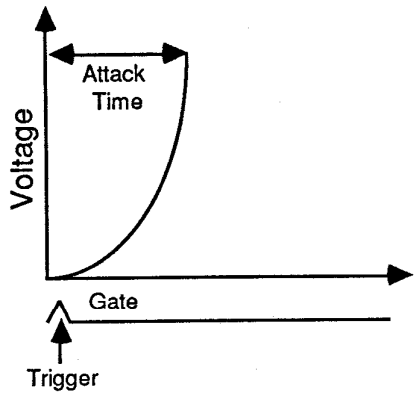


Figure 9-6

Before we examine the workings of the DX7II envelope generator, (which is, happily or unhappily, one of the most complex you'll find on any synthesizer), we might do well to take a little time to discuss the typical analog envelope found on most analog synthesizers. These envelopes are usually quite a bit simpler than the DX envelope. The analog envelope is a voltage-producing device which sends a *control voltage* (or *CV* for short), typically to a voltage-controlled amplifier (VCA) or voltage-controlled filter (VCF) in order to aperiodically vary the volume or timbre of the analog sound. Like the DX envelope, the

analog envelope needs to know when you strike a note, when you let go of the key, and how long you held the note down. This information in an analog system is generated by means of electrical signals called *triggers* and *gates*. Trigger and gate signals are in fact the analog equivalents of digital "key on" and "key off" flags. A trigger is a sharp spike in voltage which is always followed by the steady-state gate voltage. When a trigger occurs, the analog envelope generates first a rising voltage, followed by a falling voltage which then drops to a holding level. The amount of time it takes to rise is called the *attack time*: **(see figure 9-6)** and the amount of time it takes to fall is called the *decay time*: **(see figure 9-7)** and the level it drops to and holds at is called the *sustain level*. **(see figure 9-8)**

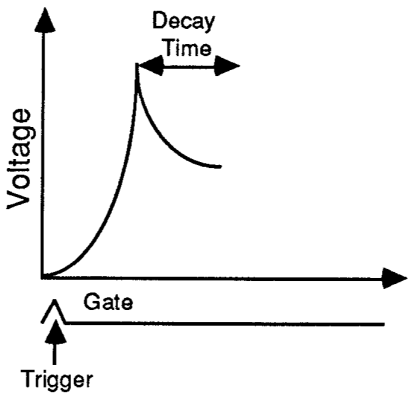


Figure 9-7

The envelope holds at its sustain level as long as your finger remains on the key - meaning that a gate voltage is present. When you let go of the key, the gate voltage disappears and the envelope then begins dropping back down to 0 volts. The amount of time it takes to return back to the 0 volt level is called the *release time*:. **(see figure 9-9)**

These four values - attack, decay, sustain, and release - are all controllable and variable by knobs or sliders on the instrument, and altering their values has the effect of "shaping" the envelope. Thus, we can build, for example, sounds that slowly rise in volume and then rapidly fall to a lower level, or sounds that quickly increase in brightness and then slowly reduce in overtone content. The four controls in the analog envelope are so standardized that the envelopes themselves are often referred to by their initials: *ADSR*.

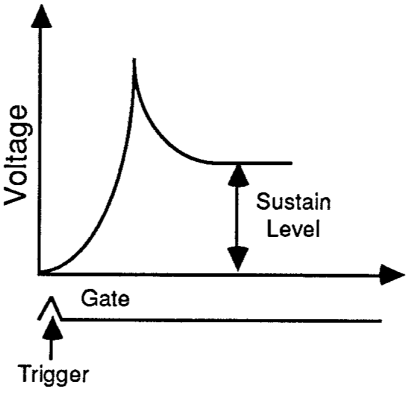


Figure 9-8

While analog envelopes have been sufficient for many years in creating interesting and useful synthesized sounds, the folks at Yamaha, with access to digital circuitry, thought that the time was right to introduce a newer, more powerful set of controls, allowing us far more flexibility in shaping sounds. We will be occasionally drawing comparisons between the DX7II EG and the standard ADSR because synthesists have become very accustomed to thinking of sounds along the lines of "attack", "decay", "sustain", and "release". Keep in mind, however, that the DX7II offers us much more than this in the way of control.

Now back to how we actually use the DX7II envelope generator: What we are essentially going to do is to draw a kind of "road-map" for the operator amplifier to follow. This map will tell the amplifier how much signal to pass at various times, and how much time to take in making changes between various output levels. We are going to be able to specify four different output levels, labeled *L1, L2, L3,* and *L4;* as well as four different rates of movement *between* the four levels. These rates will be labeled *R1, R2, R3,* and *R4*. Each Level can have a value from 0 and 99, with 0 being the lowest level (corresponding to an output level of 0) and 99 being the highest level (corresponding to the *maximum set output level.** Similarly, each Rate can have a value of 0 to 99, with 0 being the slowest possible rate of movement and 99 being the fastest.

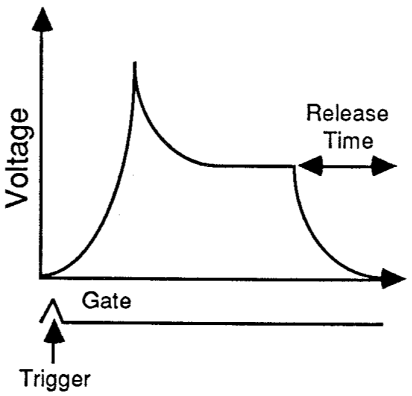


Figure 9-9

*** This will not necessarily be an output level of 99. The maximum set output level is simply the "Level" value you enter for that particular operator with edit switch 10. The EG *cannot* under any circumstances increase the "Level" value entered with edit switch 10.**

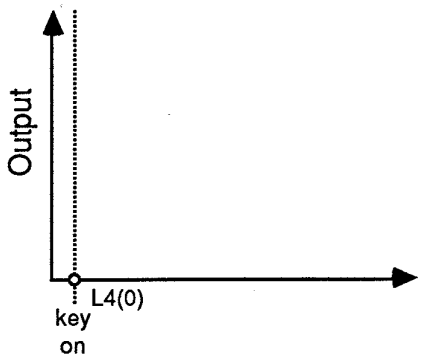


Figure 9-10

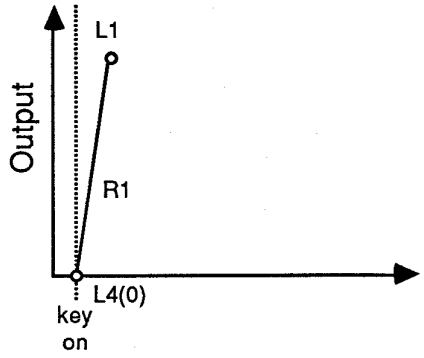


Figure 9-11

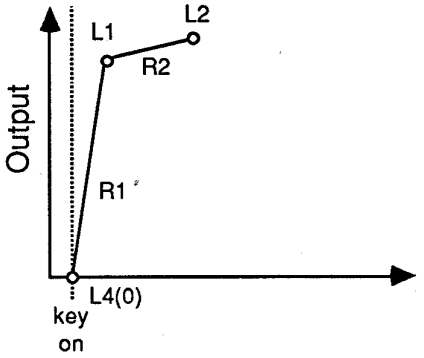


Figure 9-12

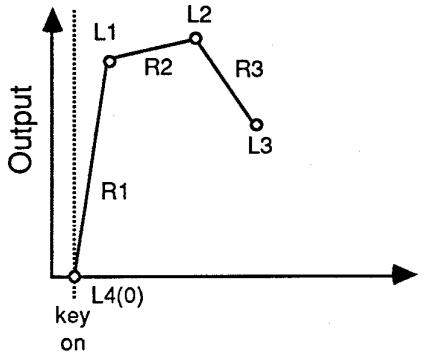


Figure 9-13

When we generate a "key on" flag by virtue of having struck a key on the keyboard, all six individual operator envelopes will begin their journey at Level 4* (*not* Level 1, as you might expect). Bear in mind, of course, that L4 might be different for each of the six operators, since their envelopes are totally independent of one another. Let's just trace the hypothetical EG of one operator, and let's suppose that we specify a Level 4 of 0 (by far the most common Level 4 value) for this operator. **(see figure 9-10)**

The first thing the EG will do is to begin moving from its starting point of Level 4 to Level 1, at a rate of speed governed by Rate 1. **(see figure 9-11)** As soon as the EG reaches Level 1, it immediately continues onward to Level 2, at a rate of speed determined by Rate 2; **(see figure 9-12)** it then continues onward to Level 3, at Rate 3. **(see figure 9-13)**

Having reached Level 3, the envelope will *hold* at this level, as long as no "key off" flag has been detected (in other words, as long as your finger stays on the note). Once a "key off" is detected, however, the envelope immediately begins returning to its ending point of Level 4 (which, you may remember, was also its starting point), at a rate of speed determined by Rate 4. **(see figure 9-14)**

Our amplifier, then, has undergone up to four separate changes in output level, resulting in up to four changes in volume or timbre over the duration of the sound. This is what we mean by a dynamic sound!

Up until this point, of course, we have not heard any dynamics whatsoever in sounds we have created from scratch. How can this be, since we know that the envelopes are *always* somehow changing the amplifiers? The answer lies in the default envelope settings that occur when we initialize.

We can view and/or enter data into an operator's EG data input via edit switch 9, which is, of course, operator-specific (we say "of course", since each operator has its *own* independent EG). Pressing this switch will call up the display in **figure 9-15.**

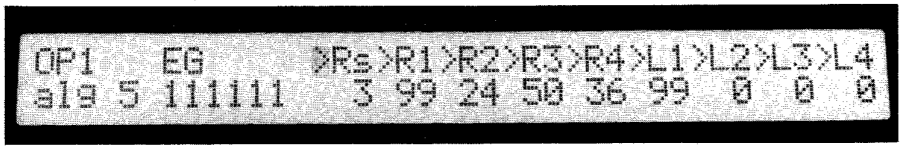


Figure 9-15

This display shows you, as usual, the operator you are viewing (upper left-hand corner), the algorithm you are working with (lower left-hand corner), and the operator on-off status (immediately to the right of the algorithm number). In addition, you are shown all four Rate and all four Level values, along with a parameter labeled "Rs" (for "Rate scaling"), which will be discussed later in this chapter.

Try initializing your instrument from single voice play mode, and then press edit switch 9, followed by edit switch 1, in order to view the default EG values for operator 1. When you initialize a DX7II voice, all six operator envelopes default the same way - that is, they each begin

*One anomaly to be aware of is that the operator EG won't know what you want Level 4 to be until it actually encounters it by reaching the *end* of an envelope cycle: in other words, the first time you strike a key upon calling up a new sound, the microprocessor will *assume* that Level 4 is 0, even if you have entered in a different value. If you are working polyphonically, you will therefore have to play 16 notes before all sixteen tone generators are fully aware of the Level 4 value you entered!

with precisely the same values for each EG Rate and Level. If we therefore observe the default EG values for operator 1, then we'll know what these values are for all six operators (or, if you are a disbeliever, you can always press edit switches 2 through 6 in order to confirm that this is the case). You should now be seeing the display in **figure 9-16.**

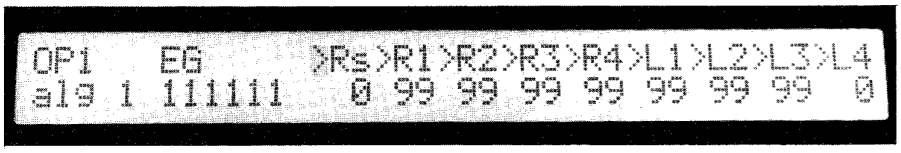


Figure 9-16

Let's graph out what this envelope would look like, again mapping output level over time. **(see figure 9-17)**

As you can see, this envelope has a square shape, and so is often referred to as a *square envelope*. Very simply, when we press a key, the output level of all six operators rises to maximum (once again, not necessarily 99, but whatever Level value has been set with edit switch 10). They then travel instantly (since R2 and R3 are at 99) to the same maximum output level (L3) and hold there until you let go of the key, at which time they instantly (R4) travel back to minimum output level (L4). This is the reason why, although we have been unknowingly using the six operator envelopes, they haven't been doing anything useful - just giving instant on and instant off - to all operators.

One point that must be made about rates, and that is that they are *rates of speed,* and do *not* reflect absolute time values. That is, they remain constant despite the distance they have to traverse. This means that the same rate value will sound *faster* when traversing lesser distances, and *slower* when traversing longer distances. Think about this: suppose you are struck with a sudden urge to visit Tibet, right now. There are probably millions of different rates of speeds that you could travel at, since Tibet is so far away (for those of you Tibetans currently reading this, substitute Brooklyn for Tibet). If you decided to go by airplane (at what would undoubtedly be a fast rate of speed), it would still take many hours to get there. On the other hand, suppose you were struck with a sudden urge to visit the door of the room you're sitting in right now. Since it's a much shorter distance to that door than it is to Tibet, there are many fewer rates of speed at which you could travel. If you decided to travel to the door at that same airplane speed, you'd obviously reach the door in much less absolute time than it would take to get to Tibet. Finally, let's suppose you were struck with a sudden urge to go nowhere at all. In that case, it doesn't matter what rate of speed you travel at, since you're not going anywhere anyway.

In the square envelope example above, different R1 and R4 values will result in many different absolute time values since they both traverse the maximum allowable distances (in the case of R1 we are traveling from an L4 of 0 to an L1 of 99; and in the case of R4 we are traveling from an L3 of 99 to an L4 of 0). **(see figure 9-18)**

On the other hand, we can see that in this particular envelope, both R2 and R3 are meaningless, since in both cases we are going from a level of 99 to a level of 99; in other words, we are going nowhere. **(see figure 9-19)**

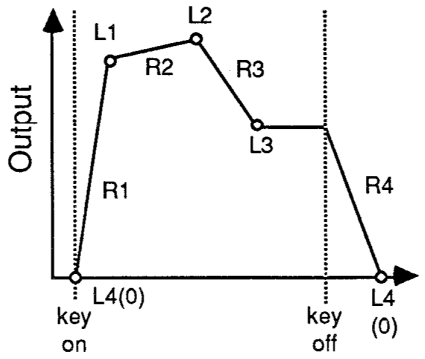


Figure 9-14

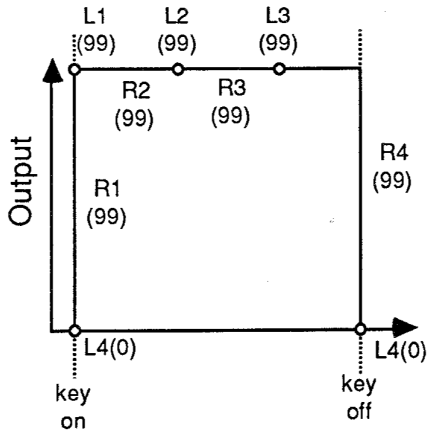


Figure 9-17

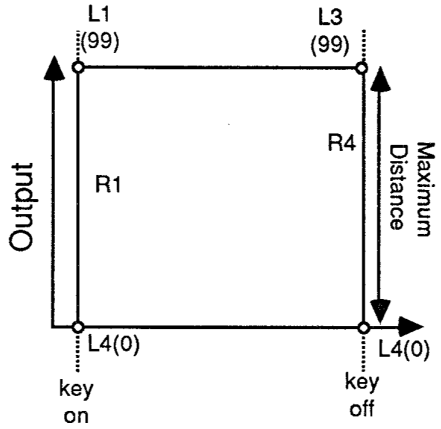


Figure 9-18

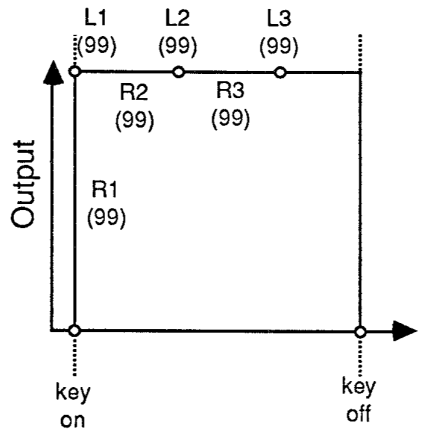


Figure 9-19

The absolute speed of the rates, then, are relative to the distances they have to travel - and the distance they have to travel is determined by the EG level settings. However, Yamaha has thrown in a couple of very unusual "rules" for the EGs to follow, rules which unfortunately can in certain circumstances negate this statement. The first of these anomalies is that, when traveling from a lower EG level to a higher EG level, the same rate will be somewhat *faster* (in absolute speed) than if it is traveling from a higher level to a lower one. In other words, an R2 of 50 will be faster if L1 is 0 and L2 is 99 than if L1 were 99 and L2 were 0. This has been done deliberately because naturally occurring "attack" times of sounds (that is, from softer to louder) are typically faster than "decay" or "release" (louder to softer) times. This "smart programming" modification to the envelope logic allows the user to accurately reflect this acoustic phenomenon in DX7II sounds. The second quirk in the DX EGs is that they have been programmed by Yamaha to have their rates respond somewhat differently (and a bit unpredictably!) when traveling from low level values to other low level values (or even, occasionally, high level values!). This is even more bizarre in the DX7II than it was in the original DX7 (for more on how this worked in the original instrument, the reader is referred to "The Complete DX7", by the same author), but can still be utilized to create *delayed*, as opposed to slow, attacks. Don't worry too much about this now, however - we'll work more with these anomalies later, in Exercise 49.

Another unusual bit of EG logic to be aware of is that the DX7II software will always automatically scale the EG levels as a function of the nominal output level of the operator (as determined by edit switch 10). This doesn't mean that the actual numbers in the LCD change, but it does mean that if you set up two different operators with different output levels, but precisely the same EG settings, the operator with the lesser output will cycle through its EG movements at a faster absolute speed than the operator with the greater output level. This is because all the EG Levels in the operator with the lesser output have been equivalently lowered (that is, *scaled*), therefore reducing the distances that the Rates in that envelope have to traverse - meaning that they will arrive at their destinations a bit sooner. We'll talk more about this phenomenon, and potential applications, in Chapter Sixteen. **(see figure 9-20)**

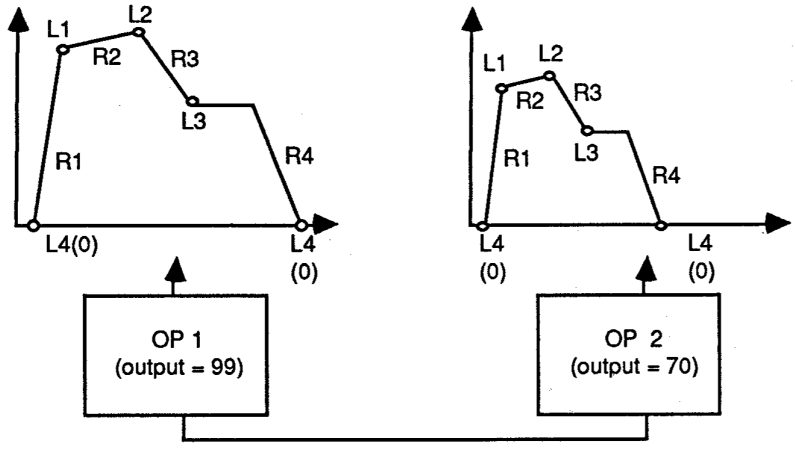


Figure 9-20

EG levels in the DX7II (unlike the DX7 EG levels) are also *real-time* controls. As you make changes to these EG levels, you will hear the resultant changes to the sound if you keep a key held down! This, you will remember, is different from the way the "Output Level" parameter works, where we had to tap the key repeatedly in order to "enter" the new data.

In any event, no one expects you to memorize all of these complicated movements, and so Yamaha has put a model EG diagram on the front panel of the machine itself. **(see figure 9-21)**

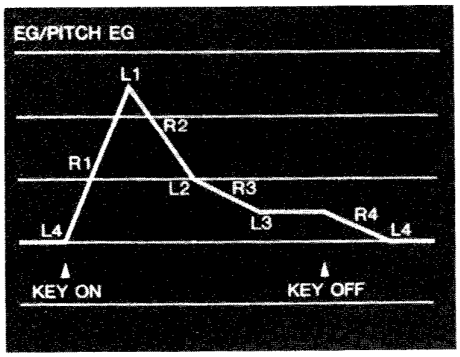


Figure 9-21

This is purely for reference purposes and should not be construed as the only shape available! If, however, we were to actually construct this envelope (and we will, in a moment), we could predict how it would affect the sound we hear, as follows: If this envelope happened to be living inside a carrier, we would hear the volume rapidly (R1) rise to maximum (L1), and then just as rapidly (R2) fall to a point about halfway (L2). It would then somewhat more slowly (R3) drop to near-minimal volume (L3) and continue to hold there as long as our finger remained on the key. Once we let go, the sound would slowly (R4, in this case about the same speed as R3) fade away to no volume (L4). **(see figure 9-22)**

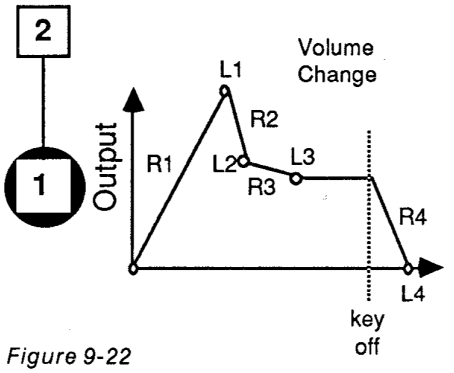


Figure 9-22

If, on the other hand, this envelope were inside a modulator, we would hear a sound whose brightness rapidly (R1) increased to maximum (L1), and then just as rapidly (R2) fell to about halfway (L2 - meaning the overtone content would quantitatively drop to about half of what it was). The sound would then somewhat more slowly (R3) decrease in brightness to a near-sine wave (L3) as long as our finger remained on the key. Once we let go, the sound would just as slowly (R4) return to a pure sine wave (L4). **(see figure 9-23)**

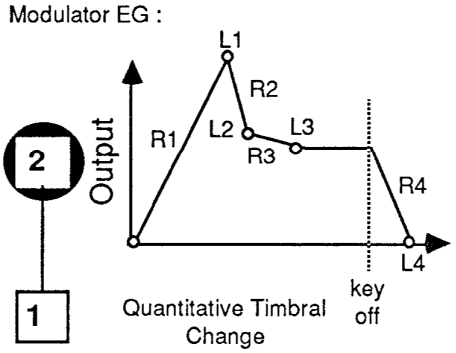


Figure 9-23

Let's run an exercise now to try and actually construct the model envelope drawn on the front panel. We'll do it first for a carrier in a single modulator-carrier system, and then for the modulator:

Exercise 43
Generating the model EG

1) INITIALIZE your DX7II from single voice play mode and leave it in algorithm #1. TURN OFF operators 3 through 6 ("110000").

2) Using the system of operators 1 and 2, GENERATE a sawtooth wave by setting the output level value for operator 2 to 99. Note that the output level for both operators is now at 99 (since operator 1 defaults to that value). Play a few notes on the keyboard and listen to confirm (Audio Cue 43A).

3) Press edit switch 9 (EG), followed by edit switch 1, in order to VIEW operator 1 (the carrier). Observe that the L4 value (the starting point of the EG) for this operator is at its default of 0 and, since that's the way the model EG diagram is represented, leave it that way. Observe also that the L1 value is also currently at its default of 99 - just where we want it to be, as well. Observe that the L2 value has also defaulted to 99. Use the cursor switches to position the cursor over the "L2" parameter, and change this to a new value of 75. Press the right cursor switch once in order to position the cursor over the "L3" parameter, and change this to a new value of 35.

4) Now we'll adjust the EG rates. Observe that R1 is currently at its default of 99. Position the cursor over the "R1" parameter and change it to a new value of 60. Press the right cursor switch again in order to position the cursor over the "R2" parameter and change it to 50.

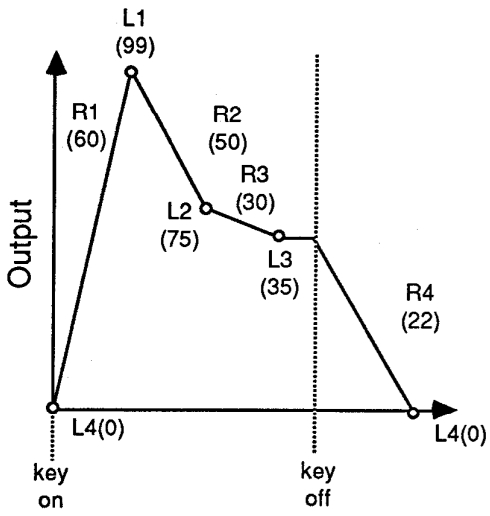


Figure 9-24

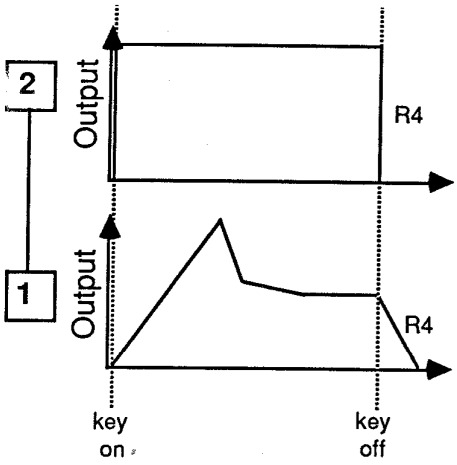


Figure 9-25

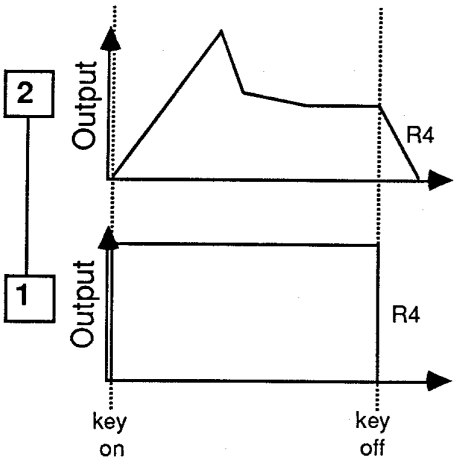


Figure 9-26

Press the right cursor switch again and change the "R3" parameter to 30. Press it one last time in order to change R4 to a new value of 22. Our newly constructed envelope for operator 1 now looks like **figure 9-24.**

5) Play a note on the keyboard and listen (Audio Cue 43B). Note that the volume quickly increases and just as quickly decreases to a lower level. This movement is followed by a slower movement (R3) down to an even lower, barely audible level. Note that when you let go of the key, the sound slowly dies away (R4) *but that a drastic timbral change occurs:* the final sound you hear is a simple sine wave. That's because the default envelope still in operator 2 has been unaffected by the changes we made to the EG of operator 1 (remember - because the EGs are totally independent of one another, altering one operator's EG will have no affect at all on any other operator's EG). Since R4 for operator 2 is still 99, *it* drops instantly (an R4 of 99) to an output level of 0 (a L4 of 0) once we let go of the key. If operator 2 sends no output to operator 1, we hear a sine wave only. **(see figure 9-25)**

6) As you play various notes, look at the model EG diagram on the front panel. You should be able to hear the volume changes that you are seeing. note that only the volume - not the timbre, and not the pitch - is altered, since we are changing only the output level of the carrier. Experiment by changing the output level of operator 1 to a lesser value. Note that while the carrier undergoes virtually the same changes, its overall volume is reduced. (As we mentioned earlier, these changes will now all be a bit faster since the EG levels have all been automatically rescaled to reflect this reduction in output level).

7) Re-INITIALIZE your DX7II from single voice play mode and repeat steps 1 through 6 above, this time making the EG changes to your modulator - operator 2. Play a few notes on the keyboard and listen (Audio Cue 43C). Note that this time the changes you hear are only timbral changes and that the volume and pitch remain constant. Note also that the R4 portion of operator 2's EG appears to be non-functioning. Why should this be? The answer lies in the fact that changing the values for operator 2's EG in no way affects operator 1. Since R4 for operator 1 is currently at its default value of 99, that means that as soon as you let go of the key, the volume of the sound instantly drops to 0 - in other words, you won't hear *anything* after you take your finger off the key. **(see figure 9-26)**

8) Experiment by changing the output level of operator 2 to a lesser value. Note that, while its EG still undergoes virtually the same changes (just a bit more quickly), the sound never gets as bright as it did before. experiment further by changing the *frequency ratio* within the system of operators 1 and 2. Note that while the EG still undergoes the same changes, the type of timbre that is generated is qualitatively different.

What happens if you take your finger off the key *before* the EG has a chance to reach its holding point of L3? The answer is simple - the envelope will immediately begin traveling from wherever it happens to be, back to L4, at R4. This means that you can set up sounds which will actually do different things depending upon whether you play notes staccato or legato (sharply or slowly). Try redoing Exercise 43 above, changing the EG values for operator 1 only, and this time try playing some notes on the keyboard quickly instead of holding the key down. If you play it so quickly that, for example, the EG only gets halfway up to

L1 before "key off", then this is what would happen: **(see figure 9-27)** and the sound would never actually reach maximum volume. Try it! The DX7II envelopes have been "trained" by Yamaha's software engineers to always automatically return to L4, at R4, whenever a "key off" flag is detected. This automatic return to the starting point is not unusual, and in fact is found even on analog ADSRs.

Let's take another look at the envelope we just created, because you may be wondering about a few of the values we selected. **(see figure 9-28)**

If we are trying to get the speed of the first drop in output level (R2) to be the same speed as the initial rise in output level (R1), why did we pick such different values for R1 and R2? The reason is, again, that these rates are constant, and so the resulting absolute times of the movements will change according to the distance to be traversed. R1 has to travel a far greater distance (0 to 99) than R2 (99 to 75); therefore, matching their absolute times of arrival means entering different constant rates. Similarly, since the angle of R3 and R4 appears to be the same on the model diagram, and since R3 and R4 have to travel different distances (R3 is going from 75 to 35 and R4 from 35 to 0), their values have to be different also.

The model diagram also appears to show L2 at a little less than half the volume of L1. Why then did we pick an L2 value of 75 instead of, say 45? The answer is that, like the output level control, the EG changes in the DX7II are *exponential,* and not *linear* (these terms will be covered in much greater detail in Chapter Twelve). Therefore, the greatest change is always at the top of the control, and an L2 of 75 will in fact be a little less than half the volume of an L1 of 99. Similarly, we had to set L3 at a value of 35, since the model diagram illustrates a volume only slightly above no volume. Levels much below 35 are virtually inaudible unless you've got your amp up really high (which you wouldn't want to do since it would make the L1 of 99 pretty nearly blow your speakers up!). The point is, the DX7II being a digital instrument, we have an enormous amount of *dynamic range* (the difference between the loudest and softest sound) available to us, even though in normal circumstances you won't make use of such a wide range.

Let's continue our examination of the EGs with step-by-step experimentation with the different levels and rates. We'll start with R1, since that's the first thing that occurs in every instance.

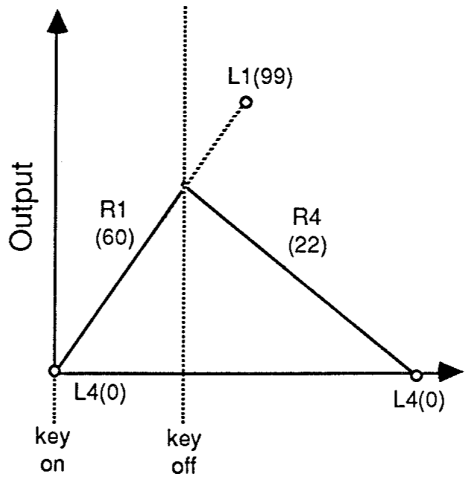


Figure 9-27

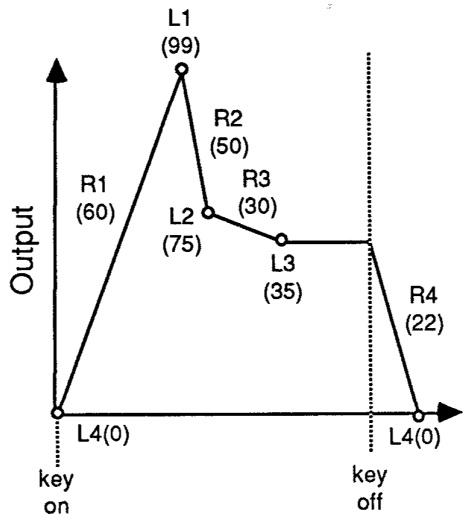


Figure 9-28

Exercise 44
Changing rate 1

1) INITIALIZE your DX7II from single voice play mode, leave it in algorithm #1, TURN OFF operators 3 through 6 ("110000"), and, using the system of operators 1 and 2, GENERATE a sawtooth wave.

2) Press edit switch 9, followed by edit switch 1, in order to VIEW the EG parameters for operator 1. Note that all four rates are currently at their default values of 99. Change the R1 value for operator 1 to a new value of 50 and listen (Audio Cue 44A). Note that the volume of the sawtooth wave fades in slowly now. Change the R1 value to 25 and listen (Audio Cue 44B). note that the volume fades in even more slowly. Hold down a key on the keyboard, and, as the sound reaches its maximum volume, play a new note (while continuing to hold down the old one). Note that each note fades in independently, since each tone generator in the DX7II is receiving information at different times (see the "Oscillator Sync" section in Chapter Three if you don't

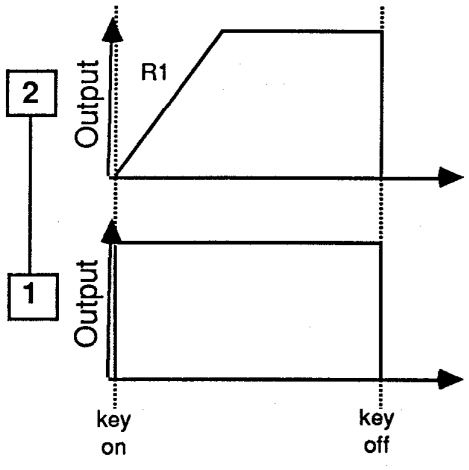


Figure 9-29

remember this concept). Note also that if you release your finger from the key before it reaches maximum volume, the volume instantly returns to 0 since R4 is still at its default of 99.

3) Experiment with different R1 values for operator 1 and note that smaller values lead to longer fade-in times and that higher values (but less than 99) just have the effect of "softening" the attack of the sound.

4) Restore the R1 value for operator 1 back to its default of 99 and press edit switch 2 in order to VIEW the EG values for operator 2.

5) Change the R1 value for operator 2 to 50, play a few notes and listen (Audio Cue 44C). Note that, while the volume remains constant, the sound now begins as a pure sine wave and quickly changes into a sawtooth wave as the output level of operator 2 is somewhat rapidly increased by its EG to maximum. **(see figure 9-29)**

6) Change the R1 value for operator 2 to 25, play a few notes and listen (Audio Cue 44D). Note that the sine-wave-to-sawtooth-wave transformation now occurs more slowly but that the overall volume of our sound remains constant. Experiment by altering the R1 value for operators 1 and/or 2 in various ways and listen to the volume and/or timbral changes that result.

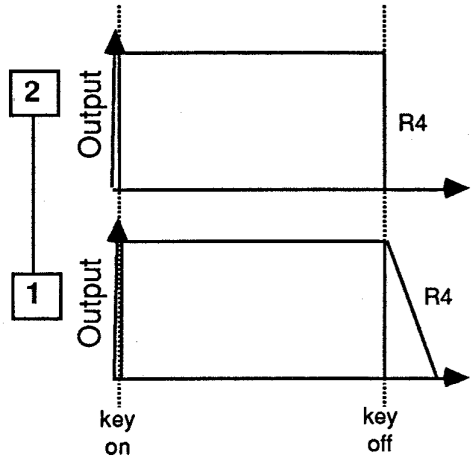


Figure 9-30

Rate 1, then, would at first glance appear to be the same as the "attack" component of the analog ADSR. This is true most of the time, but the additional flexibility of the DX7II envelope over the analog ADSR means that in special circumstances, R1 will *not* be strictly equivalent to "attack". We'll discuss these special circumstances shortly, but for now let's continue with an examination of Rate 4. This rate is responsible for any sound that will remain *after* you let go of a key (following the "key off" flag) and so it *always* emulates the "release" portion of the analog ADSR. Let's run an exercise to experiment with setting different R4 values:

Exercise 45
Changing rate 4

1) INITIALIZE your DX7II from single voice play mode, leave it in algorithm #1, TURN OFF operators 3 through 6 ("110000"), and, using the system of operators 1 and 2, GENERATE a sawtooth wave.

2) Press edit switch 9, followed by edit switch 1, in order to VIEW the EG values for operator 1 (the carrier in this system).

3) Use the cursor switches to position the cursor over the "R4" parameter and change this from its default of 99 to a new value of 30. Play a note on the keyboard, release it and listen (Audio Cue 45A). Note that while the sound lingers and slowly dies away after you release the key, the sound you hear after "key off" is a simple sine wave, not the sawtooth wave we previously generated. The reason for this is that altering R4 for operator 1 in no way affects R4 for operator 2, which is still at its default of 99. **(see figure 9-30)**

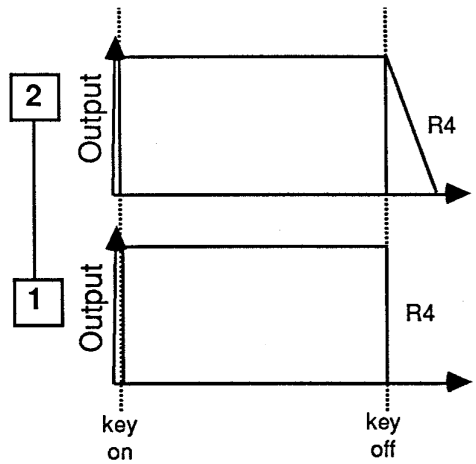


Figure 9-31

4) Now let's try the opposite: Restore R4 for operator 1 back to its default value of 99. Press edit switch 2 in order to VIEW operator 2 and change its R4 value to 30. Play a note, release the key and listen (Audio Cue 45B). note that you hear nothing at all after releasing the key. Again, this is because altering R4 for operator 2 in no way affects R4 for operator 1. Since operator 1 in this system is acting as the carrier, the total volume shuts down upon "key off". Operator 2 is continuing to fade away after "key off", but you just can't hear it! **(see figure 9-31)**

5) Now let's try setting R4 for the carrier the same as R4 for the modulator. Leave R4 for operator 2 at its current setting of 30, press edit switch 1 (in order to VIEW operator 1), and change R4 for operator 1 to the same value of 30. Play a note, release it and listen (Audio Cue 45C). Note that, as the volume fades away, so too do the overtones, at the same rate of speed. What you are hearing is a sawtooth wave slowly decreasing in volume and changing back into a sine wave at the same time. By the time the sound becomes inaudible, the overtones are practically all gone. **(see figure 9-32)**

6) What if we simply wanted to hear this sawtooth wave fade away in volume with no timbral change at all? One method would be to set the modulator's R4 to a minimal value (0). This would ensure that, as the carrier dies away, the modulator changes minimally. Try it: press edit switch 2, change R4 for operator 2 to 0, play a note, release it, and listen (Audio Cue 45D). **(see figure 9-33)**

7) Another way of accomplishing virtually the same effect would be to set R4 for the carrier to a significantly higher value than that for R4 of the modulator. This will ensure that the sound will fade away so quickly that the change in timbre is simply not heard. Try it: Change R4 for operator 1 to 50, and restore R4 for operator 2 back to 30. Play a note on the keyboard, release it, and listen (Audio Cue 45E). **(see figure 9-34)**

8) On the other hand, we may want the timbral change to occur more rapidly than the fade in volume. We can accomplish this by setting R4 for the modulator to a higher value than that of the carrier. Try it: change R4 for operator 2 to 50, and restore R4 for operator 1 back to 30. Play a note on the keyboard and listen (Audio Cue 45F). **(see figure 9-35)**

9) Experiment with different R4 values for operator 1 and operator 2. Experiment further by creating different timbres using the same system of operators 1 and 2. Note how different R4 values and offsets between the operators affect the overall sound in different ways.

10) Experiment by using algorithm #5 to create a sound with three different timbres blended together. Offset the R4 values for the different systems and note how this varies the overall sound. Don't forget that you can adjust the balance between the different timbres by altering the output levels of the individual carriers.

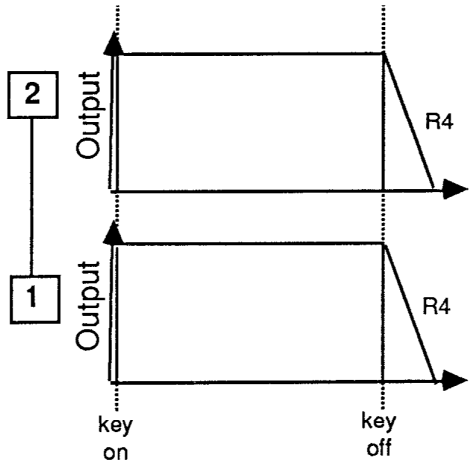


Figure 9-32

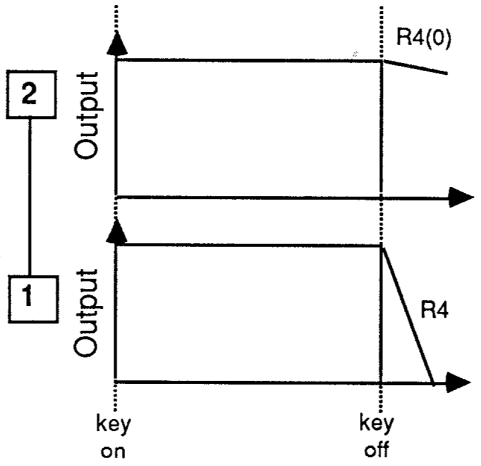


Figure 9-33

Let's continue our discussion of the EGs now with a focus on Level 3. L3 is, of course, the *sustain* level, the level that all the envelopes hold at while a "key on" situation exists. Consequently, it is of great importance because its value indicates at what output level the operator will remain while our finger stays on the key. Because L2 always immediately precedes L3, the *relative* values for these two levels also becomes important. If you are, for example, looking to create a sound that becomes either brighter or louder as you hold the key down, then you will want L3 to be *greater* than L2. **(see figure 9-36)**

If, on the other hand, you are looking to build a sound that diminishes in volume or in overtone content as you hold the key down, then you will want L3 set at a *lesser* or equivalent value to L2. **(see figure 9-37)**

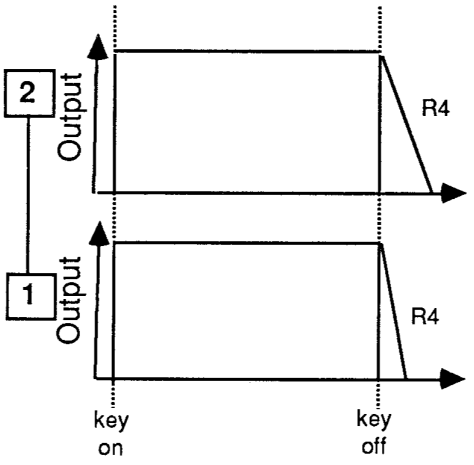


Figure 9-34

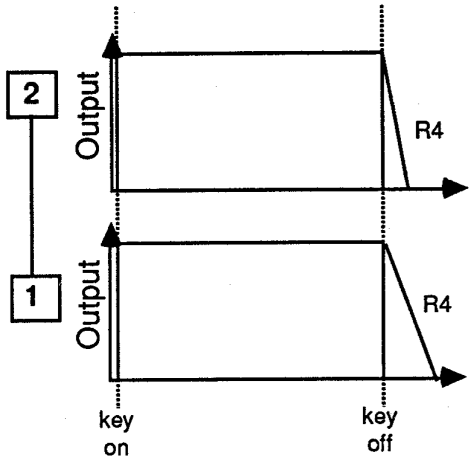


Figure 9-35

Some acoustic instruments, like the violin or tuba, are known as "sustaining" instruments, since they are theoretically capable of sustaining their sound indefinitely as long as they are played (of course, a violinist's arms will fall off eventually, and the tuba player's lungs will collapse before very long, but we are talking theoretically here). If you are trying to imitate a "sustaining" instrument on the DX7II, then you will obviously want to designate carrier L3 values greater than 0.

Other acoustic instruments, like the piano or snare drum, are "non-sustaining"; in other words, you can keep your finger on the key of a piano as long as you like, but the sound of the piano will die away sooner or later, regardless. If you are using your DX7II to imitate these types of instruments, then you will want to set your carrier L3 values to 0, thus yielding this typical fadeout in volume over time. Of course, with careful use of R3, we can have this degeneration of volume occur just as slowly or as quickly as we like.

Let's go back to "Celeste" (ROM cartridge bank 2, slot 62) - a preset which is meant to simulate the sound of a celeste, which is a non-sustaining acoustic instrument. By calling this sound up and putting the DX7II into edit mode, we can see just how the carrier envelopes are set up:

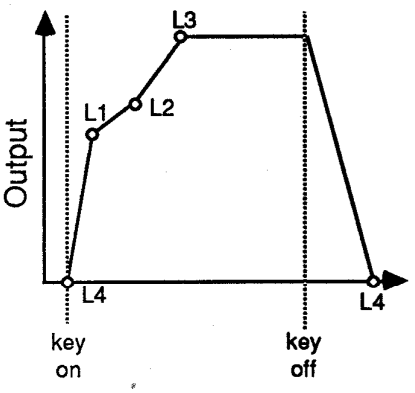


Figure 9-36

Exercise 46
Examining Level 3 and Rate 3 for "Celeste"

1) Put your DX7II in single voice play mode and call up the "Celeste" sound from your ROM cartridge (bank 2, slot 62).

2) Press the edit mode select switch, followed by edit switch 7. observe that algorithm #5 is being used for this preset. Use the operator on-off switches (edit switches 17 through 22) to listen selectively to each of the three systems. Observe that the first and second systems (operators 1 through 4) are making musical contributions to the overall sound, while the third system (operators 5 and 6) is contributing only a "knock" to the sound - so we won't deal with it here.

3) Press edit switches 18, 20, 21, and 22 in order to TURN OFF the last system and also to TURN OFF the modulators in the first two systems (operators 2 and 4). ("101000")

4) Press edit switch 9 (EG), followed by edit switch 1, in order to VIEW the EG values for operator 1. Note that L3 has been set at 0. Press edit switch 3 in order to VIEW the EG values for operator 3 and note that L3 has also been set to a value of 0 for operator 3. This confirms that both carriers will be *non-sustaining*.

5) Note the other EG values for both operators 1 and 3. They are:

Operator	L4	R1	L1	R2	L2	R3	L3	R4	L4
1	0	99	99	24	0	50	0	36	0
3	0	99	99	31	80	50	0	38	0

Note that L2 for operator 1 has been set at 0. We'll be talking more about L2 in just a little while, but the important point is that L3 is the same as L2 for this operator, meaning that the volume of this system will sustain at the same (0) level. Play a key on the keyboard in order to confirm to your own ears that the overall sound has changed. (Audio Cue 46A).

6) TURN ON operators 2 and 4. ("111100"). Note that so far we have merely observed the sound and have made no changes to it, and that is why there should currently be no decimal point in the LED. Play a chord, hold down the keys, and listen (Audio Cue 46B). Note that the

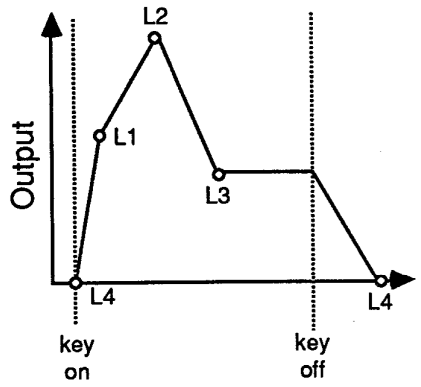


Figure 9-37

sound eventually fades away to no volume, but that it takes a long time to do so, as does a real Celeste.

7) Using the data entry slider, change the L3 values for operators 1 and 3 to a new value of 60. Change L2 for operator 1 to a new value of 60 as well. Play a chord, hold down the keys, and listen (Audio Cue 46C). Note that the sound slowly fades away, but *not* to no volume. Instead, the sound sustains at a lower level and sounds much more like a sine wave. This sound will continue as long as your fingers remain on the keys, no matter how long you do so. This is obviously not the way a real Celeste responds, but, more intriguingly, why did the timbre change? The answer lies in the EG values of the modulators in these two systems...

8) Press edit switches 2 and 4 in order to observe the EG values for operators 2 and 4. They are as follows:

Operator	L4	R1	L1	R2	L2	R3	L3	R4	L4
2	0	99	99	39	80	46	0	50	0
4	0	90	82	57	48	0	0	33	0

Note that the L3 values for these modulators is *also* 0! This means that all of the overtones will eventually die away, leaving us with only a sine wave. Try changing the L3 values for both of these modulators to the *same* values as their respective L2s (that is, change operator 2's L3 to 80, and operator 4's L3 to 48). Play a chord, hold the keys down and listen (Audio Cue 46D). Now, the volume continues to sustain *and* the timbre also remains constant - not like a real celeste at all!

9) Restore the L3 values for operators 1, 2, 3, and 4, back to their original value of 0 (if you're from that immortalized state of Missouri, use *compare* mode to prove to yourself that these were in fact the original values), and also restore L2 for operator 1 back to its original value of 0 (again, compare mode will confirm this).

10) Now let's see how we can alter the speed of the volume and timbre fadeaway. Observe the R3 values for operators 1, 2, and 3, and 4:

Operator	L2	R3	L3
1	0	50	0
2	80	46	0
3	80	50	0
4	48	0	0

What do these values tell us? Well, rate 3 represents the amount of time it takes each of these operators to reach their sustain level of 0 - in other words, it represents a *decay* time. In this particular case, of course, operator 1's R3 is meaningless, since we are going from an L2 of 0 to an L3 of 0 - in other words, we're going nowhere! For this operator, then, R2 will be our "decay" time. In the other operators, the varying R3 values show varying decay times - with modulator 4 taking many long minutes to fade away, while its carrier (operator 3) fades away quickly. What does this mean? It indicates that we will hear little or no timbral change as this system dies down in volume - just as adjusting R4 values in Exercise 46 above did. Let's try changing these "decay" times:

11) Use the data entry slider to add 20 to each of the the R3 values for operators 2, 3, and 4 to the following new values: 66, 70, and 20. Then add 20 to the R2 value for operator 1, making it 44. Play a chord on the keyboard, HOLD DOWN the keys, and listen (Audio Cue 46E). Note that the sound now takes a much shorter time to fade away altogether, and that this is equally unnatural, relative to the way a real celeste behaves.

12) Experiment with different R3 and L3 values for each of these four operators and note how these changes affect the overall sound.

While R1, R4, and L3 can each be equivalent to ADSR values, Level 4 is truly unique in that it allows the DX7II user to actually begin and end envelopes at points other that 0 - after all, there is nothing to prevent you from entering a value greater than 0 for L4. Bear in mind, however, two things: First of all, as mentioned earlier, the first time you call up a voice, the DX7II assumes that L4 for all six operator EGs is 0. Secondly, L4 - like all the other DX7II EG level controls - is a *real-time* control. This means that it changes the output level of the selected operator as you change its value. This is useful, but can be occasionally problematic when dealing specifically with L4, since all sixteen tone generators are changed at once. Therefore, if you had previously depressed keys since calling up a voice with an L4 value greater than 0, and you then subsequently changed L4 to 0 and back again to a value greater than 0, the previously played notes will actually reappear! This rather complicated description of a simple phenomenon will make more sense when we run Exercise 48, below.

If you use an L4 of greater than 0 in a carrier, you will generate a continuous sound, since the volume will never completely die away. **(see figure 9-38)**

Of course, having an L4 of greater than 0 won't automatically *start* the envelope, since it still requires a "key on" flag from the keyboard.

The relative values of L3 and L4 can also be quite important in the carrier. Having an L4 greater than 0 but equal to L3 (the sustain level) will result in a sound whose volume is unchanging from its sustain level, even after you let go of the key; in other words, you can dramatically remove your fingers from the keys and no change in the volume will occur! This can fall into the category of Amuse Your Friends At Parties, or it can have a similar effect to that of a "drone" or "hold" switch typically found on the analog synthesizer. **(see figure 9-39)**

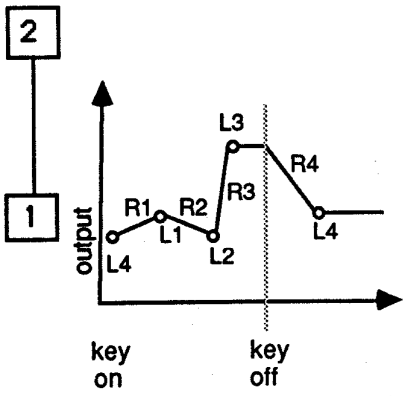


Figure 9-38

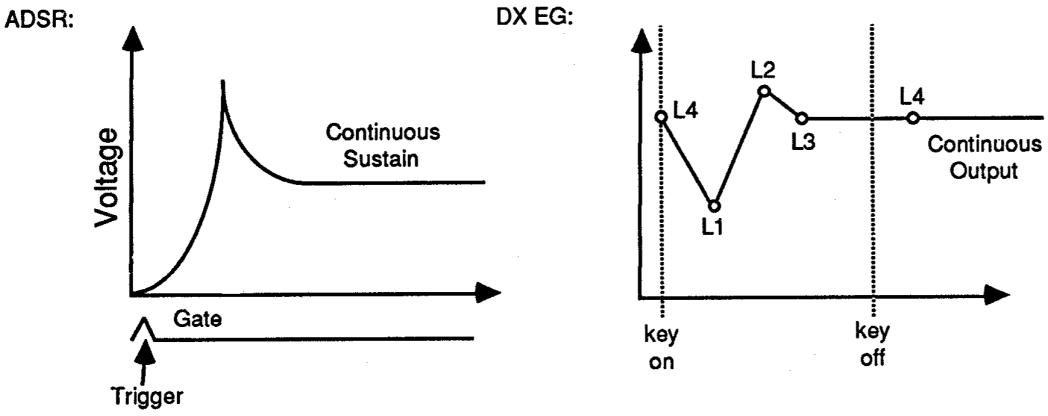


Figure 9-39

In the modulator, having an L4 greater than 0 but equal to L3 will result in a sound which undergoes no timbral change whatever after you release the key. This is actually the best method of accomplishing what we tried to do in steps 6 and 7 of Exercise 46 above.

Synthesizers are often used in recording studios for *gating* effects - a continuous white noise sound, for example, might be set up and a

device called a *noise gate* would control the output signal and perhaps only allow it to pass when triggered by a snare drum. This is a typical application, but by no means the only one. On the analog synthesizer, a commonly found *drone* switch allows the sound to be continuously heard without you having to literally keep your finger on the key continuously, or by (more commonly) holding the key down with a piece of tape. Digital FM instruments like the DX7II are often found in recording studios with remnants of pieces of tape stuck to their keys, since there is no obvious DX7II switch labeled "drone". However, L4 - as we have seen - can accomplish precisely the same effect.

On the other hand, L4 in our carrier, while still being greater than 0, can be either less or *greater* than the value for L3. An L4 value greater than L3 will result in a sound that actually gets louder after you *let go* of the key. **(see figure 9-40)** In a modulator, having an L4 greater than L3 would result in a sound that gets *brighter* after you let go of the key. Of course, the speed with which either change occurs will be determined by R4. Similarly, an L4 in our carrier which is less than L3, but still greater than 0, will result in a sound which is continuous but at a lower volume level. **(see figure 9-41)** Again, the speed with which the volume will drop is determined by R4. Remember that the aural effect caused by altering L4 for a modulator will largely depend on the L4 value for the carrier, since the carrier controls the overall volume. Let's run an Exercise to try some of these changes:

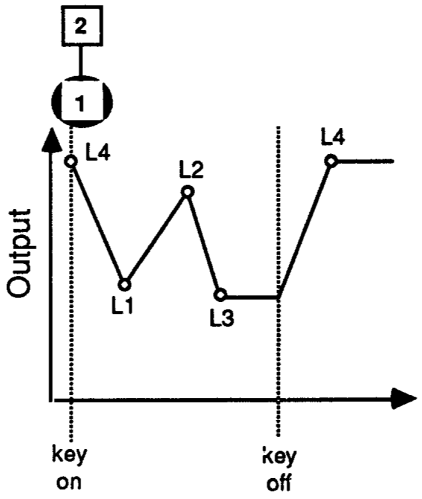


Figure 9-40

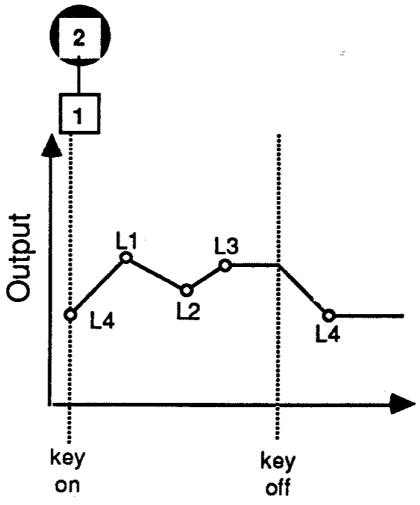


Figure 9-41

Exercise 47
Changing level 4

1) INITIALIZE your DX7II from single voice play mode, leave it in algorithm #1, TURN OFF operators 3 through 6 ("110000"), and, using the system of operators 1 and 2, GENERATE a sawtooth wave.

2) Press edit switch 9, followed by edit switch 1, in order to VIEW the EG settings for operator 1. Change the L4 value from its default of 0 to a new value of 99. Note that L3 (as well as L1 and L2) are also at 99 (their defaults).

3) Play a note on the keyboard, release the key and listen (Audio Cue 47A). Note that you hear a continuous *sine* wave when you let go of the key. This is because altering L4 for operator 1 in no way alters L4 for operator 2, which is still at its default of 0. **(see figure 9-42)**

4) Restore the L4 value for operator 1 back to its default of 0. Note that this results in an immediate cessation of the sound, since EG level changes, as noted above, are real time changes.

5) Now let's see what happens when we restore L4 back to 99. Without playing any more keys, change the L4 value back to 99. You should be hearing the last note (or notes - up to sixteen of them) you played in step 3 above. As pointed out earlier, this is one of the problems with having EG changes that are real-time changes. We will therefore have to re-initialize the DX7II if we want to try a different alteration to L4 and not hear this same note or notes reappear.

6) We now want to try to create a continuous sawtooth wave. Therefore, re-INITIALIZE your DX7II from single voice play mode, TURN OFF operators 3 through 6 ("110000"), and re-GENERATE a sawtooth wave from the system of operators 1 and 2.

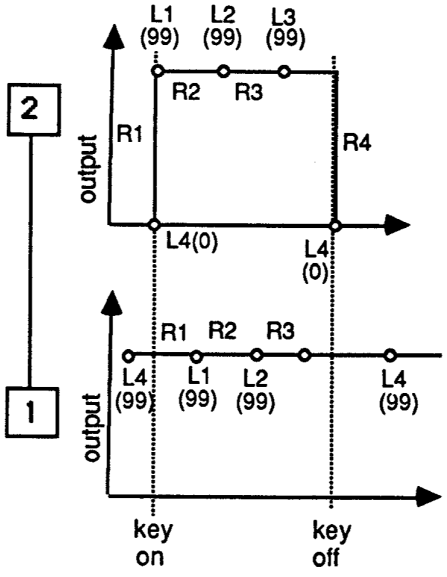


Figure 9-42

* you can also "kill" the real-time effect by pressing the edit switch twice in order to momentarily go into Compare mode and back, but this can have unusual consequences if you are comparing to a voice that has one or more carriers at L4. For this reason, you"ll be better off simply re-initializing.

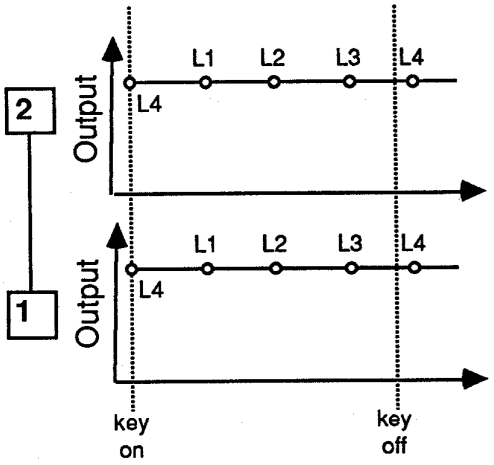


Figure 9-43

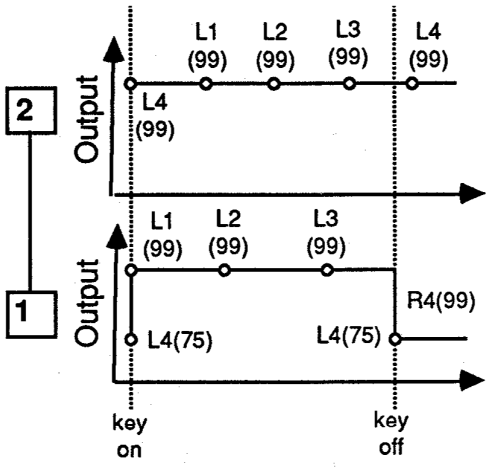


Figure 9-44

7) Press edit switch 9 and change the L4 value for *both* operator 1 and operator 2 to 99. Play a note, release the key, and listen (Audio Cue 47B). The EG graph illustrates why we are now hearing a continuous sawtooth wave. **(see figure 9-43)**

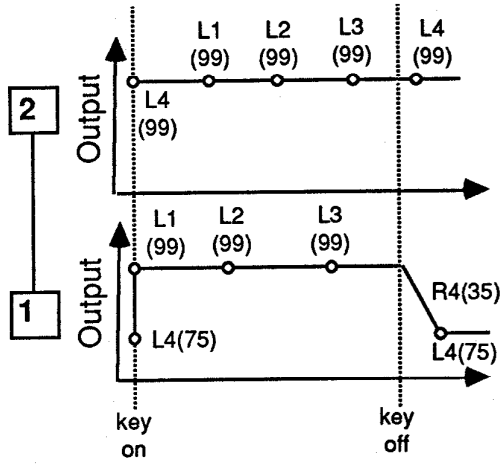


Figure 9-45

8) Let's try a different use of L4. Re-INITIALIZE your DX7II, create a sawtooth wave from the system of operators 1 and 2 as before, and then change L4 for operator 1 only to a new value of 75. Play a note on the keyboard, release it, and listen (Audio Cue 47C). You should be hearing a continuous sine wave, but at a lower level. **(see figure 9-44)** Note that the volume dropped instantly to this lower level immediately upon "key off". This is because R4 is still at its default of 99.

9) Let's see what happens if we alter R4. Repeat step 8 above (complete with re-INITIALIZING) and then change R4 for operator 1 only to a new value of 35. Play a note on the keyboard, release it, and listen (Audio Cue 47D). Note that the volume slowly dies down to its lower, continuous level. **(see figure 9-45)** Experiment with different L4 and R4 values for operator 1 - remembering to re-INITIALIZE after each experiment so as to not hear the same notes holding.

10) Let's now construct a sawtooth wave that actually gets louder when we release the key. Re-INITIALIZE your DX7II and create a sawtooth wave with the system of operators 1 and 2, as before. Change L4 for operator 2 only to a new value of 99. Change L1, L2, and L3 for operator 1 to values of 75 for each, and change L4 for operator 1 to a new value of 99. Play a note on the keyboard, release it, and listen (Audio Cue 47E). The envelopes now look like **figure 9-46.** Note that the volume *instantly* increases upon "key off". This is because R4 of operator 1 is currently at its default of 99.

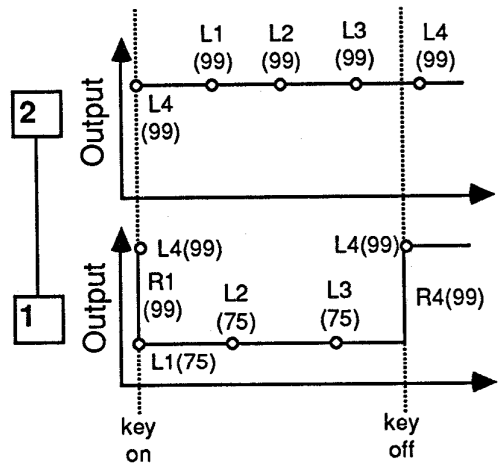


Figure 9-46

11) Once again, let's slow down operator 1's R4. Repeat step 10 above (including the re-INITIALIZATION) and change R4 for operator 1 to a new value of 35. Play a note on the keyboard, release it, and listen (Audio Cue 47F). Note that the sound now *slowly* increases in volume. **(see figure 9-47)** Experiment with different L4 and R4 values for operator 1, remembering, as usual, to re-INITIALIZE each time so that the same notes don't keep hanging on.

12) Re-INITIALIZE your DX7II one more time and, again, make a sawtooth wave with the system of operators 1 and 2. Change L4 for both operators 1 and 2 to a new value of 99, but change L1, L2, and L3 for operator 2 *only* to a new value of 30. Play a note on the keyboard, hold it down a moment, release it, and listen (Audio Cue 47G). Note

that after "key off", the sound instantly gets brighter, because our envelopes look like **figure 9-48.**

13) Of course, changing R4 to a lesser value will cause the sound to *slowly* get brighter after "key off". Try it: Re-INITIALIZE again, make that same old boring sawtooth wave (sorry, but we're nearly done with this!), change L4 for both operators 1 and 2 to a new value of 99, and change R4 for operator 2 to a new value of 99, and change R4 for operator 2 to a new value of 35. Play a note on the keyboard, hold it down a moment, release it, and listen (Audio Cue 47H). The envelope now looks like **figure 9-49.** Experiment further with different R4, and different Level values for operator 2.

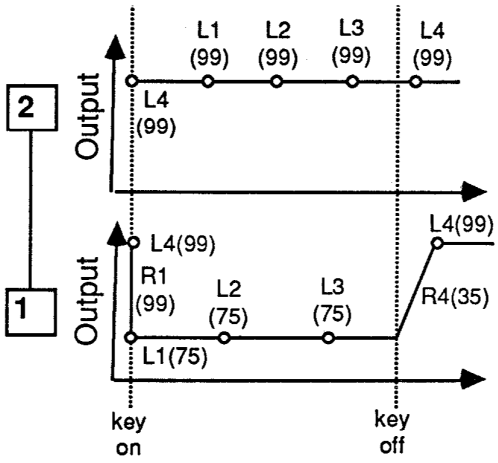


Figure 9-47

Obviously, there is a degree of interaction between the EG of our carrier and that of our modulator, even though the two are quite independent of one another. Since the carrier EG is actually controlling the amount of total sound we hear, it will tend to sometimes appear to be more in control. But we can also use that interaction to great advantage in setting up complex movements in DX7II sounds which couldn't possibly exist on any other synthesizer. Consider the "HarpsiBox" preset, which you will find in your ROM cartridge, bank 1, slot 52. In perfect mimicry of an acoustic harpsichord, when you play a note, you actually hear the "plectrum" spring back! Call up the sound, try it, and take note of the additional "after-sound" you hear after releasing the key. Any time you hear anything happen after releasing a key, you know that R4 and L4 in the EGs are at play. Call up this sound and put your DX7II into edit mode so we can examine this sound and see exactly how this effect is conjured up. Press edit switch 7 (ALGORITHM), and the first thing you will notice is that "HarpsiBox" was constructed with algorithm #9. **(see figure 9-50)** As you can see from this diagram, this algorithm offers two systems: the first one consists of a simple modulator-carrier configuration (operators 1 and 2) and the second one of a carrier (operator 3) being modulated simultaneously by a single modulator (operator 4) and a stack (operators 5 and 6). Turn off operators 3, 4, 5, and 6 for a moment and listen to the first system alone. Your ears will tell you that all you are hearing is a percussive, knocking kind of sound. This component is meant to simulate the sound of the harpsichord plectrum striking the string, and is accomplished by having both operators in fixed-frequency mode **(see figure 9-51)** with percussive EG shapes for each. **(see**

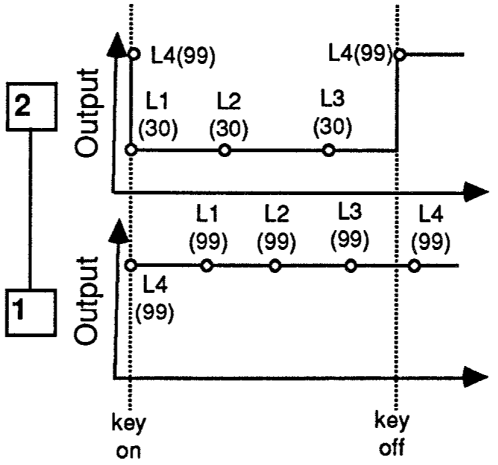


Figure 9-48

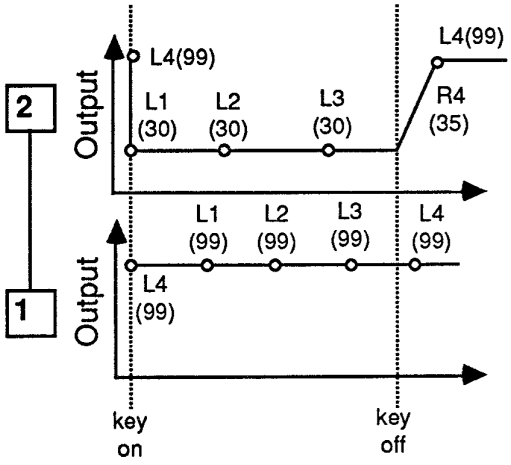


Figure 9-49

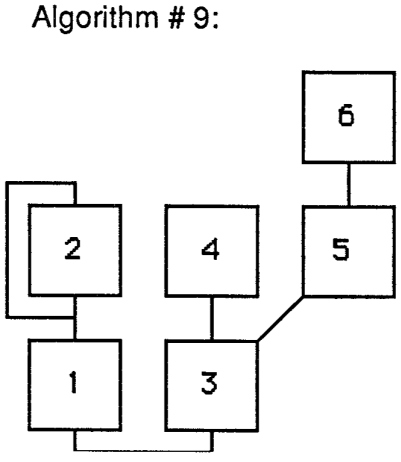


Figure 9-50

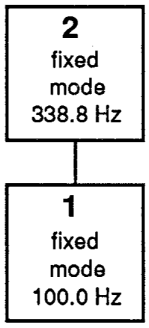


Figure 9-51

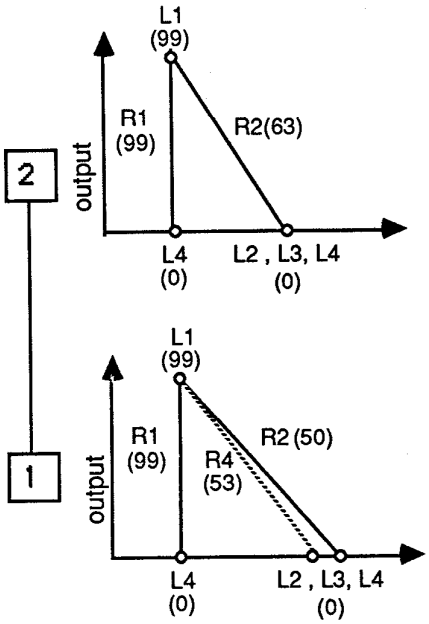


Figure 9-52

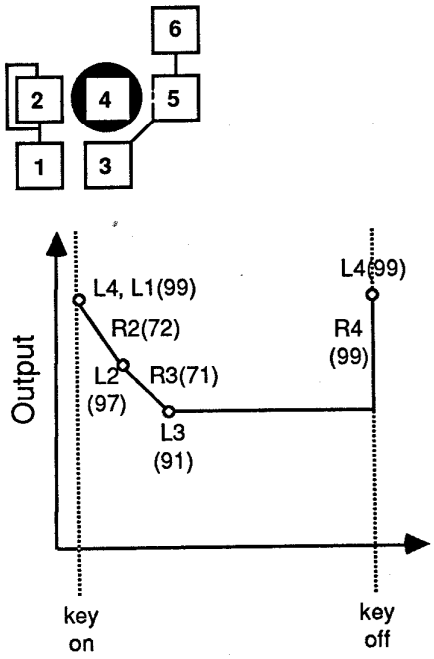


Figure 9-53

figure 9-52) You can view these values, of course, by simply pressing edit switch 9 (EG), followed by edit switches 1 and 2. In any event, this system isn't doing anything particularly exciting, and certainly is not responsible for the "spring-back" sound we referred to above. Obviously, this is being caused by the second system of operators 3, 4, 5, and 6, and a listen to this system alone will confirm that to your ears. Listen to just operators 3 and 4 alone, and then just 3, 5, and 6. Clearly, the stack of operators 5 and 6 is, as you might expect, causing the carrier to generate many more overtones (and higher ones as well) than just the single modulator (operator 4).* But all three modulators are responsible for the "spring-back". How do we know this? Simple - just turn all three of them off and listen to the carrier alone.

Doesn't sound much like a harpsichord, does it? This has a lot to do with the fact that a harpsichord is a very bright sound - meaning that we're really missing the overtones that our modulators provide - but it actually has a lot to do with the envelope shape of the carrier as well! **(see figure 9-53)** Again, you can confirm these by simply viewing the EG rate and level parameters for operator 3. Nothing very exciting here, is there? The L3 value of 0 tells us that this is a *non-sustaining* system, but the only important variable here is the R4 value of 50. This tells us that we will be able to hear some sound after we let go of the key - provided that the envelope hasn't had time to reach its sustain (L3) value of 0. The very slow R2 (28) and R3 (27) values ensure that this simply won't happen unless you keep your finger on the key for a very long time. All three modulator envelopes, on the other hand, have very similar shapes. Let's just take a look at the EG for modulator 4. **(see figure 9-54)** The interesting thing here is the L4 value of 99. Exercise 47, above, showed us that setting a modulator with an L4 greater than L3 (which in this modulator is 91) will result in a sound that gets *brighter* after the key is released. The moderate R4 in our carrier ensures that we will hear that increase in brightness. Let's overlay operator 3's EG (the carrier) with operator 4's EG (the modulator), as follows: **(see figure 9-55).** Turn off everything except operators 3 and 4. Play a note, release it, and listen as you look at this diagram. Can you "see" what you're "hearing"?** You should be able to: the carrier dies away at a moderate rate of speed (an R4 value of 50) to an L4 of 0 - in other words, the sound disappears. On the other hand, the modulator *instantly* (an R4 of 99) rises in output level to maximum (an L4 of 99) *even as the sound is dying away!* That's why the brightness of the sound increases greatly *after* "key off". The stack of modulators (operators 5 and 6) have very similar EGs and therefore contribute the same kind of "spring-back", though they cause the carrier (operator 3) to generate higher overtones than does operator 4. Examine these settings for yourself and make sure you understand this concept, because you may well find many uses for this trick of the trade in your own programming efforts. In summary, Level 4 is a powerful tool that allows us to change the sound in various different ways *after* the key is released. This is one of many DX7II envelope effects that normally cannot be duplicated on other synthesizers.

* If you listen to them on their own, you'll also find that the higher frequency ratio of the stack causes the carrier to jump to a much higher fundamental frequency than does the single modulator (operator 4). How can a carrier have two fundamental frequencies? Obviously, it can't. When all three modulators are in play, the lower of the two "fundamentals" is perceived by us as the "true" fundamental, and the higher one is perceived as yet another overtone. This comes under the heading of "aural illusions": amaze your friends at parties!

* You'll be able to hear these EG changes more clearly for low notes than for high ones. The reason for this? Another edit parameter called *keyboard rate scaling*, which will be discussed in detail shortly.

Let's quickly summarize the comparisons we have discovered between the DX7II envelope and the standard analog ADSR: 1) DX7II Rate 1 = Analog *attack* time (usually), 2) DX7II Rate 4 = Analog *release* time, and 3) DX7II Level 3 = Analog *sustain* level. The only one of the analog parameters for which we have no DX7II EG equivalent is the *decay* time. Instead, we can initiate a complicated series of movements by using R2, L2, and R3. It is in this section that we can generate very complex movements within a sound *while still holding down a key on the keyboard,* but *before* the envelope ever reaches and holds at its sustain level (L3). This will allow us to create aural effects quite unlike those typically produced by analog synthesizers. For example, as we observed in the "Celeste" preset, there's nothing that says that L2 cannot have a value of 0. If we leave the other EG levels at their initialization default values and just make this one change, we'll have an envelope that looks like **figure 9-56.**

If this EG happened to be living inside a carrier, we would actually hear a sound with *two* separate attacks! On the other hand, if we set L1 *and* L2 both at very low values, but leave all the other EG values at their defaults, we have learned that R2 will *not* operate in the usual manner but will in fact allow us to set up a *delayed* attack. **(see figure 9-57)** Let's run an Exercise to try out both of these unusual effects:

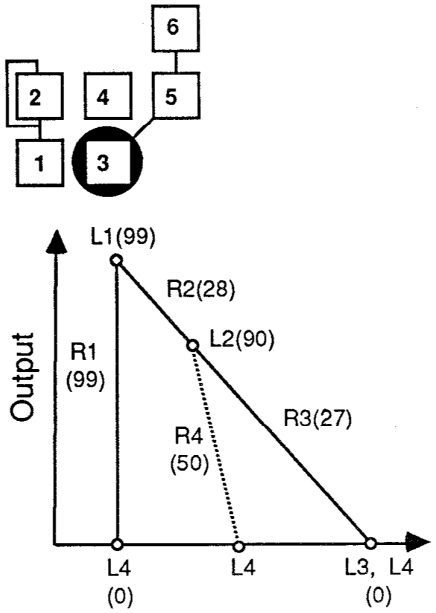


Figure 9-54

Exercise 48
Creating a delayed attack and a double attack

1) INITIALIZE your DX7II from single voice play mode, leave it in algorithm #1, TURN OFF operators 3 through 6 ("110000"), and, using the system of operators 1 and 2, GENERATE a sawtooth wave.

2) Press edit switch 9, followed by edit switch 1, in order to VIEW the EG values for operator 1. Position the cursor over the L1 value and change this to a new value of 0. Press the right cursor switch once in order to position the cursor over the L2 parameter and change this also to a new value of 2.

3) Change R2 for operator 1 to a new value of 40. Play a note on the keyboard and listen. Note that the initial attack is now delayed, but that the sound attacks instantly when it does come in. This is because R3 is currently at its default value of 99. **(see figure 9-58)** Changing R3 to a lesser value will cause the same delayed attack to occur (the amount of delay being determined by R2), but will fade in the delayed attack. Position the cursor over the R3 parameter and change R3 for operator 1 to a new value of 30. Play a note on the keyboard and listen

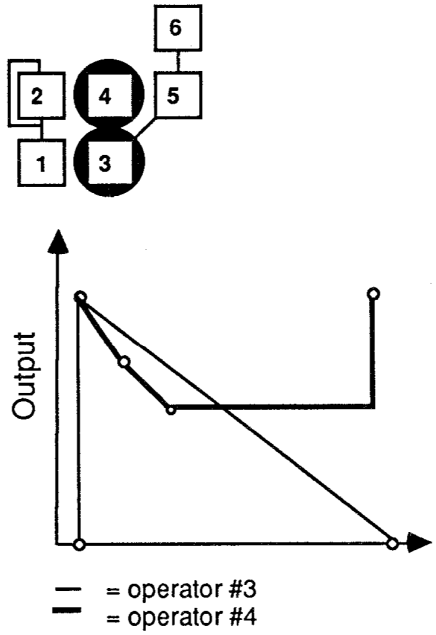


Figure 9-55

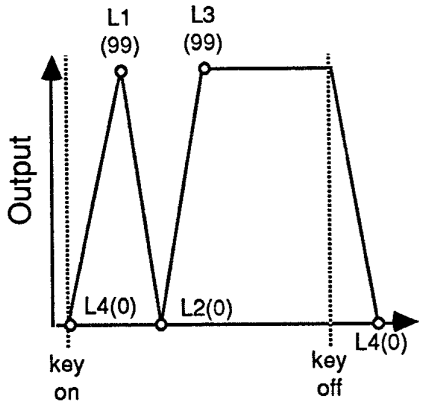


Figure 9-56

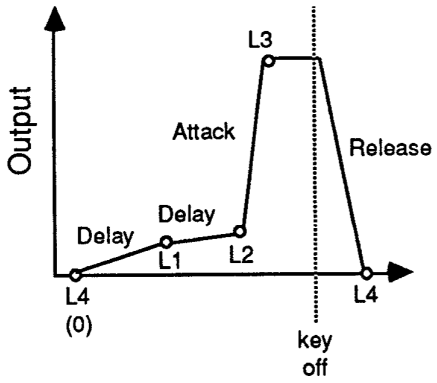


Figure 9-57

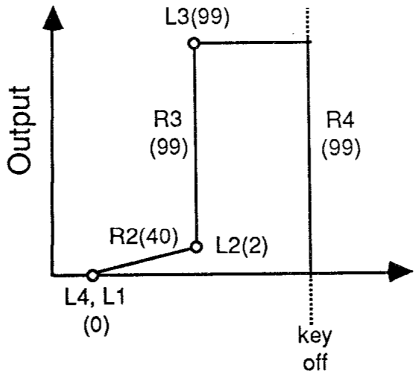


Figure 9-58

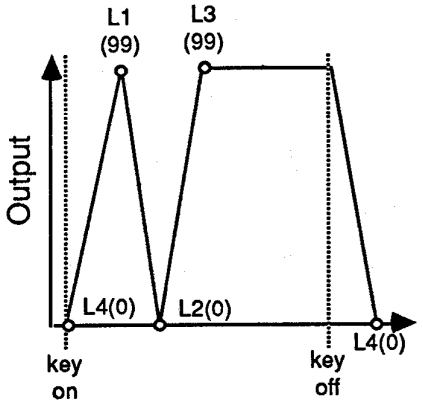


Figure 9-59

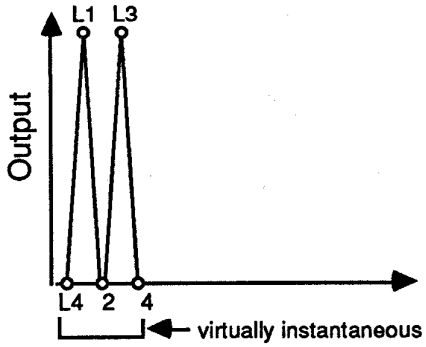


Figure 9-60

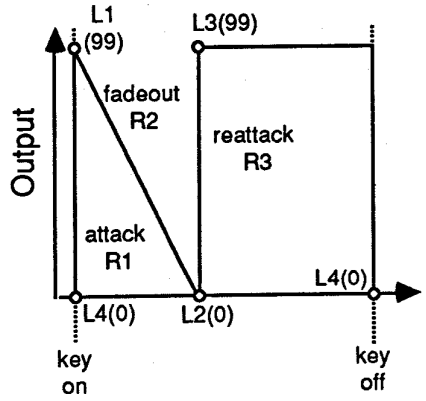


Figure 9-61

(Audio Cue 48A). Note that the attack is delayed, but is now a slow fade-in, as R3 (and *not* R1) is essentially setting the attack time. Restore R3 for operator 1 back to its default value of 99.

4) CHANGE the R2 value for operator 1 to 20. Play a key and listen. note that the sound is now delayed for a longer period of time, but that the initial attack is still instantaneous. Arpeggiate several notes, hold them down and listen (Audio Cue 48B). Note that they re-arpeggiate in the same order, but that all their appearances are delayed. Now CHANGE the R2 value for operator 1 to 15. Arpeggiate several notes, hold them down, and listen (be patient, as this will take 20 seconds or so to attack!) (Audio Cue 48C). Note that the original timing of your arpeggiation is somewhat altered, as the voice assignment operation of the DX7II becomes somewhat "confused" (this appears to happen when dealing with very slow rates of movement between very low EG levels). Finally, play a chord, hold it down and listen (Audio Cue 48D). Note that some of the notes reappear at slightly different times, again due to this "confusion".

5) Restore both L1 and L2 for operator 1 back to their default values of 99. Then restore R2 for operator 1 back to its default value of 99. You should now be back to your default square envelope.

6) Change L2 for operator 1 to a new value of 0. This causes an "M"-shaped envelope, like in **figure 9-59,** to be generated. We may now expect to hear a sound with *two* separate attacks.

7) Play a note and listen (Audio Cue 48E). Note that we do *not* hear this predicted effect - the sound is exactly the same as it was before we changed the L2 value. Why is this? The answer is that the EG *rates* are still all at their default values of 99. Even though our sound *is* changing levels twice, from 0 to 99 back to 0, back to 99 again, it is making these changes so rapidly that we, as mere humans, cannot possibly hear them. **(see figure 9-60)**

In order to hear these changes, we will have to slow down either R2 or R3, or both. R2 is the rate of speed our envelope takes to get from its first peak of 99 (L1) to its first drop to 0 (L2). Slowing this rate down, then, will have the audible effect of allowing us to hear a sawtooth wave with a rapid initial attack (R1=99) which then slowly fades away before again rapidly re-attacking. **(see figure 9-61)**

8) Change R2 for operator 1 to a new value of 45. Play a note on the keyboard and listen (Audio Cue 48F). Note that we now hear the double attack, as predicted.

9) Arpeggiate several notes on the keyboard and continue to hold the keys down and listen (Audio Cue 48G). Note that the notes re-attack in the same order. The reason for this is that the DX7II voice assignment system treats each note and each EG independently and feeds the data to consecutive tone generators only as the "key on" flags are received. Experiment with greater or lesser R2 values for operator 1 and note how they change the overall sound. If you have time to kill, try entering the minimum value of "0" and note that our sound now takes over *five full minutes* to die away completely and reattack! Because we are traveling from one extreme level (99) to the other (0), this is the longest rate of change available on the DX7II - far longer than EG values available on most other synthesizers. When you're done experimenting, restore the R2 value for operator 1 back to 45.

10) R3 is the amount of time it takes for the sound to reattack. Because R3 is currently at its default of 99, the sound is now reattacking instantly. Let's try lowering this value: change R3 for

operator 1 to a new value of 45, play a note on the keyboard and listen (Audio Cue 48H). Note that the sound now takes virtually as long to reattack as it did to die away in the first place.* Change this value further to 30, arpeggiate a few notes on the keyboard, hold the keys down and listen (Audio Cue 48I). Note that we can create some interesting "fade-in" affects with low R3 values. Restore R3 for operator 1 back to 45. **(see figure 9-62)**

11) R1, in this instance, is acting as the initial attack time. Because R1 for operator 1 is currently at its default of 99, the sound initially attacks instantaneously. Change R1 for operator 1 to a new value of 45. Play a note on the keyboard and listen (Audio Cue 48J). Note that the sawtooth wave now takes just as long in its first attack as it does in its second. **(see figure 9-63)** Experiment by inserting new values for R1, R2, and R3 and note the changes that are made to the overall sound.

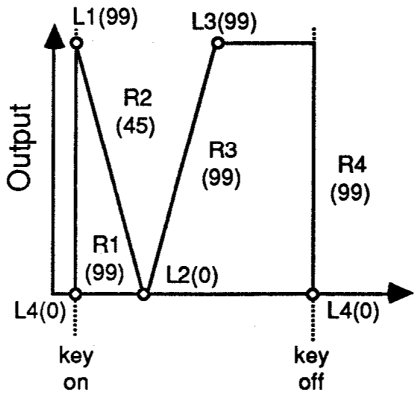


Figure 9-62

It's worth taking a moment to discuss the logic (or lack of logic) behind the process of creating delayed attacks in the DX7II. The original DX7 also had this ability, and even though it was a bit quirky, there were a few general rules that could be established: for example, you needed at least two adjacent levels of 20 or less, but it didn't matter if the level values were the same or not. If they weren't the same, the *difference* between them had a lot to do with the length of the delay engendered by the slow rate. In the DX7II, on the other hand, the only rule that we can establish is that delayed attacks can definitely *not* be generated if the adjacent values are the same. Sometimes, you'll need as slight a difference as two increments (just as we did in the last exercise) - but if the difference is greater, the delay may not necessarily take longer. Also, it doesn't seem to matter whether the first level is higher or lower than the second - only that they are both below 25 and with at least two increments of difference between them. What we may well be doing here is trying to decipher the logic of a software bug, but these delayed attacks can in fact be valuable in practice and so you might want to spend some time experimenting and seeing what rules and generalities you can come up with here.

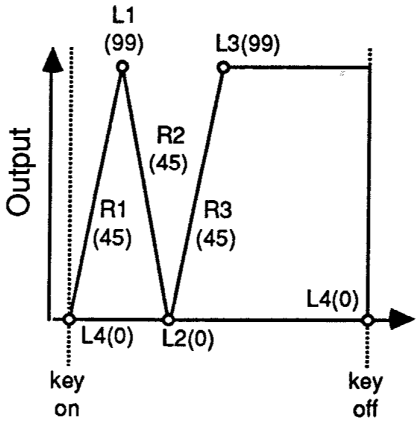


Figure 9-63

That aside, because *all* of the changes we made in the preceding exercise were to our carrier, we heard, of course, only volume changes. If, on the other hand, the same changes were made to our modulator, we would hear a similar kind of timbral change, which in this particular instance could be described as a double "wah-wah" effect. **(see figure 9-64)**

As the output level for the modulator increases from L4 (0) to L1 (99), the sound gets brighter. As it then decreases from L1 (99) to L2 (0), the sound gets duller. It then returns back to L3 (99) for a brighter effect again, hence the double "wah".

We can do this, or we can take things a step further by giving both the modulator and carrier envelopes the same settings. This, of course, would result in a sound that gets louder *and* brighter, all at the same time, followed by a similar decrease in both volume and overtones, followed by yet another increase. In just a little while, we'll discuss an easy way to set up the same envelopes in more than one operator. Before we discuss this short-cut (the *EG and Scaling Copy* function), let's digress just a bit and discuss another, important EG-related topic: *Rate scaling*.

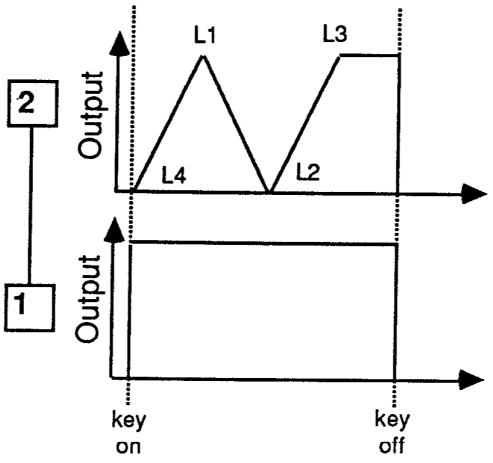


Figure 9-64

* The reason it won't take *exactly* as long is because, remember, "attack" rates in the DX7II EG are preset to always be slightly faster than equivalent "decay" rates.

The purpose of this control is to allow us to shape - or "scale" - the timing of our envelopes according to which note we actually play on the keyboard. It is therefore a geographic control, but one which does not affect output level in any way. Instead, it affects the speed - or "rate" - of movement of the operator EGs. Hence the term "Rate scaling".

If you play the lowest note of an acoustic piano and hold it down, the note will linger on for a good minute or two before it finally dies away. On the other hand, if you play and hold down the highest note, it will rapidly disappear. This is because of physical considerations within the piano - most significantly, the length of the strings. Because the low note has a much longer string, it can vibrate back and forth for a greater period of time before it loses energy and dies away. The high note, with its shorter string, cannot sustain its vibrations for as long.

There are, of course, no strings or moving parts of any kind in the DX7II, but the Rate scaling parameter allows us to digitally simulate this effect nonetheless. Specifically, what this control will do is to *increase* the four EG rate values - by the same fixed amount - for each operator as you play higher notes on the keyboard.* This will naturally cause the envelopes for those tone generators to cycle through much faster than those being triggered from the action of lower notes being played. The result, of course, is very similar to that which naturally occurs in most acoustic instruments, particularly stringed ones - where lower notes sustain for much longer periods of time than do higher notes.

The Rate scaling parameter is accessed via the EG edit switch - switch 9. This is the "RS" parameter in the LCD display, immediately to the left of the R1 value. Edit switch 9, of course, is operator-specific, and so we can set up voices which scale in different ways for different operators within the overall sound, allowing you to construct unusual (but possibly acoustically unrealistic) effects, as well as the accurate acoustic simulation we just described.

The range of this control is 0 to 7, allowing us to choose from among eight different rate scaling amounts. A value of 0, naturally, will call for no rate scaling effects, and a value of 7 will call for the maximum effect. The initialization default for this parameter, as you may have guessed, is 0 for all six operators, indicating that when you first initialize, no Rate scaling will be occurring.

Exercise 49
Keyboard rate scaling

1) INITIALIZE your DX7II from single voice play mode, leave it in algorithm #1, TURN OFF operators 3 through 6 ("110000"), GENERATE a sawtooth wave with the system of operators 1 and 2, and enter the following values into the EG of operator 1: R1 = 45; R2 = 45; R3 = 45; and L2 = 0. Leave all other EG values at their defaults. This sets up precisely the same double-attack sound that we built (and heard) in the last step (step 11) of Exercise 48 above. Play middle C on the keyboard and listen to confirm (Audio Cue 49A).

2) You should currently be viewing the EG values for operator 1. observe that the current Rate scaling value ("Rs") is at its default of 0. Use the data entry slider to change this to the maximum value of 7.

* No, the actual numbers don't increase in the LCD - but they are increased as far as the DX7II microprocessor is concerned!

3) Play C1 (the lowest note on the keyboard) and listen (Audio Cue 49B). Note that the amount of time the sound takes to attack and reattack is virtually the same as before*.

4) Play C2 and listen (Audio Cue 49C). Note that the amount of time the sound takes to both attack and reattack is slightly lessened. Continue by playing and listening to C3 (Audio Cue 49D), C4 (Audio Cue 49E), C5 (Audio Cue 49F), and C6 (the highest note on the keyboard - Audio Cue 49G). Note that each time, the envelopes take shorter and shorter time periods to cycle through to L3, and that for the highest notes of the keyboard, this becomes so short as to be virtually instantaneous. **(see figure 9-65)**

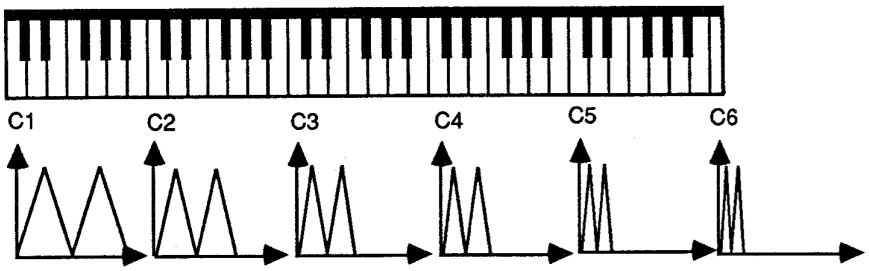


Figure 9-65

5) Experiment by setting the keyboard rate scaling value for operator 1 to a new value of 4. Repeat steps 3 and 4 above and note that the same effect occurs, but less drastically, since we have assigned operator 1 less sensitivity to this effect.

The effect of Rate scaling is very easy to hear with the "double-attack" sound, but you should try using it to greater or lesser degrees in most sounds you create - particularly if you are trying to simulate acoustic sounds, since most acoustic sounds undergo their volume and timbral changes at faster speeds when higher pitches are played.

The ability to use different Rate scaling amounts in individual operators is a very powerful tool, and one that can be used to great advantage when dealing with complex modulator-carrier systems. For example, you could have two modulators sending signal to a single carrier, as in algorithm #20 **(see figure 9-66)** and have one modulator set to a high ratio number (which will cause the carrier to generate high overtones), with the other set to a lower ratio number (which will cause the carrier to generate lower overtones). If you then set up complex envelopes in both but apply more Rate scaling to the higher frequency modulator, you will build a sound whose highest overtones fade away faster as you play higher pitches. The advantage of such a set-up? Well, for one thing, all stringed instruments - from violin to electric guitar - exhibit this kind of characteristic timbral change! For more information on this and other digital FM tips, the reader is referred to *"A Synthesist's Guide to Acoustic Instruments"*.

Algorithm#20:

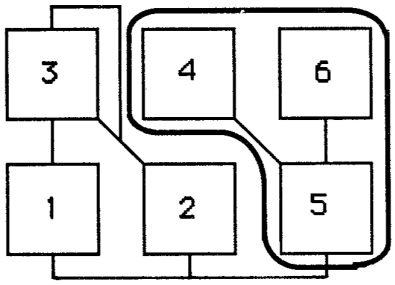


Figure 9-66

EG and Scaling Copy

As we mentioned earlier, there may well be times when you will want to have more than one operator set up with similar envelopes.

* It will actually be a little faster since C1, while the lowest physical note on the keyboard is not actually the lowest note the DX7II recognizes. This feature enables you to use the DX7II with larger external keyboards. For more information on this, see Chapter Twelve ("Keyboard Level Scaling").

For example, when creating sounds using multi-system algorithms, you may often want to set up complete systems with similar envelopes to one another - or you may want operators within a particular system to have similar envelopes. Note that I said "similar", and *not* "the same". This is because the randomness in naturally occurring sounds dictates that they will never undergo precisely the same volume and timbral changes at precisely the same time. For that reason, it will probably always be a good idea to *offset* envelopes at least to a slight degree so you avoid having any two operators with precisely the same EG settings. Failing to do so would introduce an artificial quality to the sound and would reinforce the fact that we are digitally simulating sounds. While this may be occasionally desirable, most of the time it will be preferable to disguise the DX7II's digital precision.

There are two ways to get both operator's EGs set up with the same data: the hard way and the easy way. The hard way is to do it manually. In other words, you'd press edit switch 9, followed by edit switch 1, in order to view the EG values for operator 1, make a mental (or written) note of them, then press edit switch 2 and enter the same values in for operator 2, and so on. In the best instance, this is a time-consuming and conceivably problematic way of doing things. In the worst instance, of course, it can be the proverbial pain in the proverbial you-know-what, especially if you're in a studio costing hundreds of dollars per hour with an unfriendly producer glaring at you.

For this reason, the DX7II provides us with the easy way: the *EG and Scaling Copy* command. This can be accessed only if you are in edit mode and only if you are currently viewing the LCD display of either edit switch 9 (EG) or edit switch 10 (OUTPUT LEVEL). The DX7II is given this command by the act of you pressing and *holding down* the pink store button, while in edit mode. Let's try it. Your DX7II should still be set from the last step of Exercise 50 (if it isn't, take a moment to redo it), and you should currently be viewing operator 1 (the carrier). Press and hold down the pink store button. Your LCD should look like **figure 9-67.**

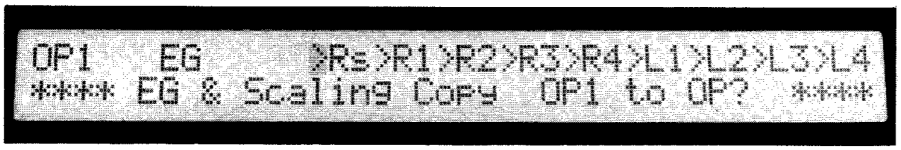


Figure 9-67

We are being asked a question by our faithful microprocessor, and it is simply asking you which operator you want to copy this EG data to. You can instruct it to copy this data to any of the remaining operators by now pressing any of the operator select switches (that is, edit switches 1 through 6 - note that these switches are also labeled "EG COPY"). It is important to realize that this control not only copies EG and Rate scaling data, it also (somewhat unfortunately) copies over the operator's set output level and also *keyboard level scaling* data for that operator (to be discussed in Chapter Twelve). This will ensure that the destination envelope is in fact exactly the same as the source envelope - but it's still a pain, simply because most of the time you probably won't want to copy output level. Still, there's nothing we can do about this, short of being aware of it!

The reason why, in this particular instance, our LCD said "from OP1 to OP?" was because we had been viewing operator 1. If we had, for example, been viewing operator 5 instead, it would have said "from OP5 to OP?". You therefore must always first view the operator whose data you wish to copy. Copying this EG, Rate scaling, and output level data in no way affects the original (source) operator; it still will retain its same values. Let's run a short Exercise to try it:

Exercise 50
EG and Scaling copy

1) We'll begin by setting up the same double-attack sound we used in Exercise 49. INITIALIZE your DX7II from single voice play mode, leave it in algorithm #1, TURN OFF operators 3 through 6, GENERATE a sawtooth wave using the system of operators 1 and 2, and enter the following values into the EG of operator 1 only: R1 = 45; R2 = 45; R3 = 45; and L2 = 0. Play a note on the keyboard and listen (Audio Cue 50A). Now change the Rs value from its default of 0 to a new value of 5. Leave all other EG values at their default settings and *don't* change any EG defaults for operator 2. Make sure you are currently viewing the EG display for operator 1.

2) Press and hold down the pink Store switch. Note that the LCD reads something like **figure 9-68.**

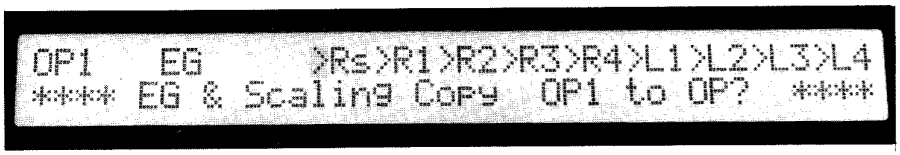


Figure 9-68

3) While still holding down the Store switch, with your other hand, press edit switch 2. Note that the LCD now reads something like **figure 9-69.**

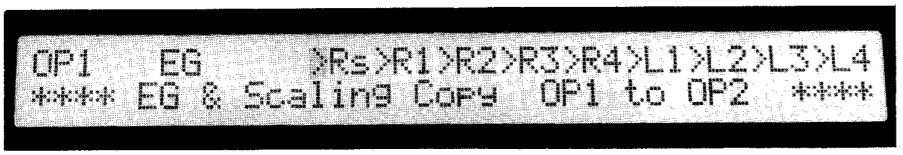


Figure 9-69

4) Release both switches. The EG data as well as the Scaling data (and output level data) from operator 1 has now been copied exactly into the EG of operator 2. Play C1 (the lowest note on the keyboard) and listen (Audio Cue 50B). Press edit switch 2 in order to VIEW operator 2. Note that the EG and Rs values for operator 2 are now the same as those for operator 1. Play C2 and listen (Audio Cue 50C). Note that the envelopes of *both* operators have increased in speed, so that both the volume *and* timbre are changing more rapidly. Play C3 (Audio Cue 50D), C4 (Audio Cue 50E), C5 (Audio Cue 50F), and C6 (Audio Cue 50G) in turn, and note that the rates of movement get faster as you play higher notes.

5) Experiment by setting up Rate scaling offsets between operators 1 and 2 and listen to the unusual effects that can be generated.

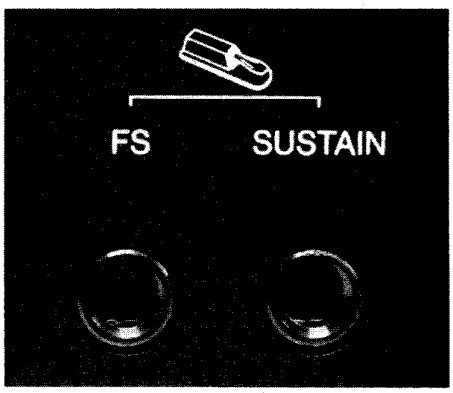


Figure 9-70

Of course, we have used the Store switch before. In the last Chapter, we saw how the Store button initiates the storage of voice data into any memory slot. However, remember that we always used the Store button in that instance from one of the DX7II play modes. Here we are not: instead, we press the Store button directly from edit mode. This is how the DX7II "knows" that we are giving an "EG and Scaling Copy" command, and not simply initiating a "Store" procedure.

The sustain footswitch

The DX7II allows for an enormous amount of external controllers. These include the use of several footpedals and footswitches. The footpedals (called *FC1* and *FC2* , for "Foot Controllers"), are normally standard push-pull type pedals and can be used for a variety of functions, all of which will be covered in Chapters Ten and Thirteen. The footswitches (called *FS1* and *FS2*) are piano-like sustain pedals. FS2 is a multi-purpose switch, which can be used for a variety of different functions, all programmed from Performance edit mode - so we'll get to it in Chapter Fourteen. However, FS1 is in fact your *sustain pedal* and, even though it is activated or deactivated in *Performance* (and not voice) edit mode, it will always default to being active whenever you are in any of the voice play modes. Its actions tie in so closely with the EG, however, we'll plunge right in and discuss it here and now.

This sustain pedal is plugged into the "sustain" input in the back of the instrument. **(see figure 9-70)**

Once it is depressed, its action will be to *block* any "key off" flags generated by the keyboard - so that the microprocessor will "think" that you are holding down any and all notes you play on the keyboard. In other words, as long as you keep your foot on the pedal, all the EGs in the DX7II will cycle through to their L3 values - even if you've taken your fingers off the keys! This is what makes the DX7II sustain pedal a *true* sustain pedal, unlike the *release* pedal used in most analog synthesizers. Having reached their respective L3s, the EGs will, of course, continue to hold at that level until you take your foot off the sustain pedal, at which time *all* the tone generators will receive a "key off" signal and all EGs will begin their long (or short - depending on R4) journeys back to their L4 ending points.

Of course, depressing the pedal alone (either by stepping on it or playing it Neil Young songs) is not enough to initiate the actions of the EGs. They still require a "key on" flag - which can *only* be generated by the keyboard - and the operators (at least those that are in "ratio" mode) still need to know which notes to play by receiving some pitch data input from the keyboard. But it should be clear that the use of the sustain pedal ensures that you won't have to hold down the key in order to get the envelopes to cycle through to their sustain (L3) levels.

The DX7II sustain pedal lends the same kind of performance dynamics and is usually used just as you would a piano sustain pedal. It also comes in very handy when you have created a voice that takes a very long time to develop because of very slow Rates in the EGs. As we've seen, it can take up to 5 whole minutes in order to get from Level to Level (at their extreme settings), and so we can literally build sounds on the DX7II that take up to 20 minutes to develop in their entirety! We won't go quite that far, but let's run a quick Exercise to create a long-developing sound and use the sustain pedal to save ourselves wrist cramps as we play it!

Exercise 51

Generating a long swelled sound

1) INITIALIZE your DX7II from single voice play mode and select algorithm #30, which looks something like **figure 9-71.** This algorithm has four systems; one stack and three simple carriers. Make sure all operators are ON ("111111").

2) Using the appropriate edit switches, set up the following frequency ratio for the stack: 0.75 : 2.00 : 1.00 and the following ratio numbers for the single carriers: operator 1 = 5.00; operator 2 = 1.01; operator 6 = 0.50. Also set the detuning control for operator 3 only to +6. The final pitch inputs to our six operators should look like **figure 9-72.**

3) Set the output levels of the six operators as follows: op 1 = 99; op 2 = 96; op 3 = 97; op 4 = 71; op 5 = 81; op 6 = 99.

4) Use the appropriate controls to enter the following EG data into the six operators:

Op	L4	R1	L1	R2	L2	R3	L3	R4	L4
1	0	5	99	31	70	33	99	35	0
2	0	65	5	6	99	22	94	22	0
3	0	36	99	79	82	99	99	23	0
4	99	11	85	99	85	99	85	10	99
5	0	7	99	99	99	99	99	27	0
6	0	13	99	20	67	84	99	23	0

A graph of these envelopes would look like **figures 9-73(a)** through **9-73(f).**

5) Leave all other edit parameters at their default values. Play a note on the keyboard and *hold it down* - this sound takes over a minute to fully develop. Listen (Audio Cue 51A). Now play a chord, hold it down, and listen. Note that the notes re-attack an octave lower some thirty seconds in. This is the envelope of operator 6 (a sine wave tuned an octave lower; i.e., F Coarse = 0.50) quickly (R3 = 84) rising back up to maximum level after slowly reaching about half-volume (L2 = 67).

6) We know that instead of holding down a note physically, we can achieve the same effect by keeping the sustain pedal depressed. Press down the sustain pedal and keep it held down *without* playing a note.

Algorithm #30:

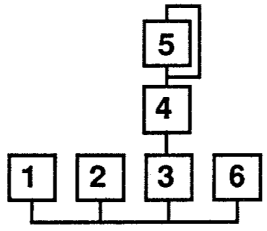


Figure 9-71

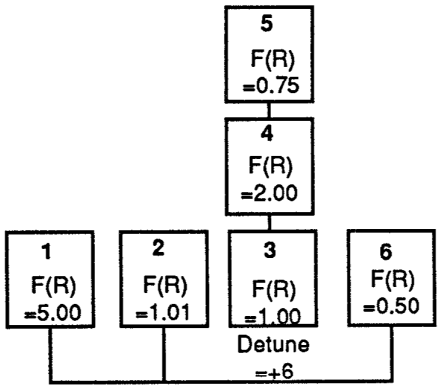


Figure 9-72

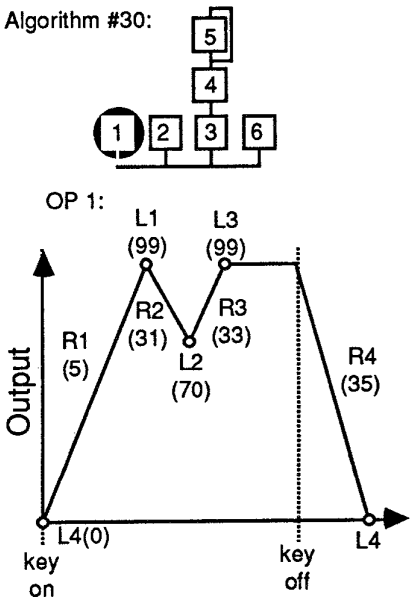


Figure 9-73(a)

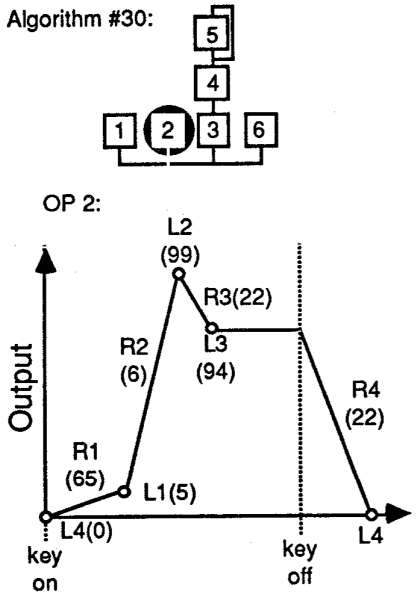


Figure 9-73(b)

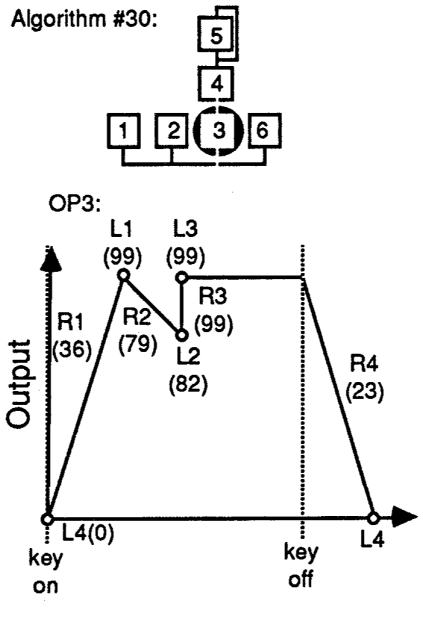


Figure 9-73(c)

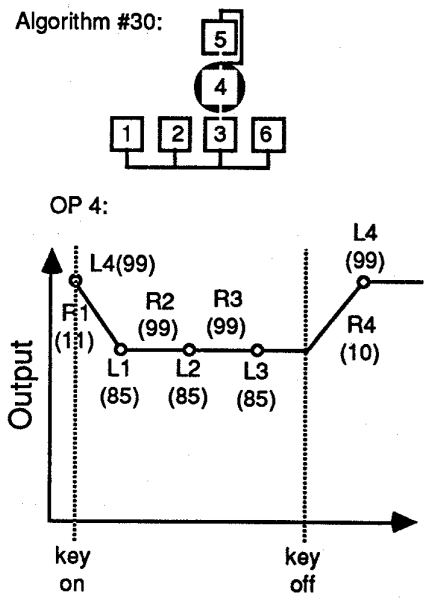


Figure 9-73(d)

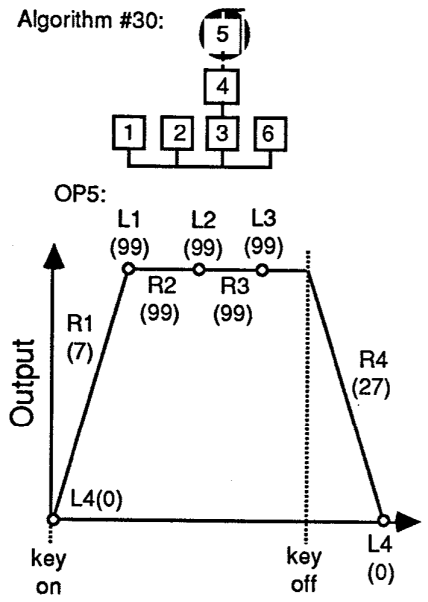


Figure 9-73(e)

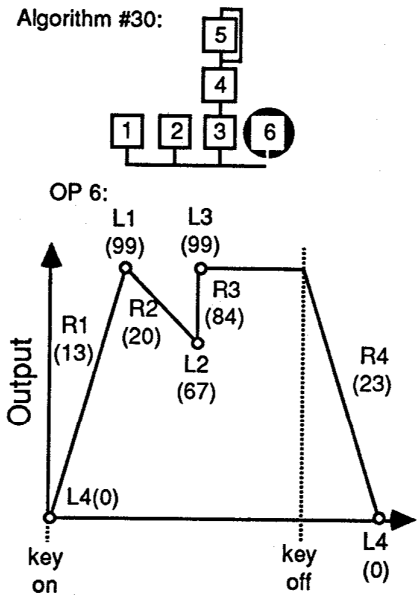


Figure 9-73(f)

Note that no sound is heard. While keeping it depressed, play a series of arpeggiated notes and listen (Audio Cue 51B) as each note develops independently. Because of the long R4 values for each operator, you can let go of the sustain pedal and the notes will hang on for several seconds (also note that the slightly different R4 values for each operator ensure that they all drop out at slightly different times, making for a very pleasant effect). Even as the previous notes are hanging on, you can re-depress the sustain pedal and add new notes to the total sound. Try it!

7) Name and store this sound in a free memory slot (RAM cartridge or internal - the choice is yours). Then experiment by listening to each system independently, and, referring to figures 9-73(a) through 9-73(f) above, visually following each envelope as it goes through its travels. Note that every operator except operator 5 (whose output to operator 3 is governed by the movements of operator 4) undergoes several output level changes over its duration. Also note that the two operators contributing beating effects (operators 2 and 3) each fade in rather slowly by virtue of their EG settings. note that the beating in this sound therefore does not occur right away, but fades in as well! This is a particularly subtle way to establish beating effects and is often more pleasing than simply introducing them right away. As you experiment with this sound, remember to store your changes (or overwrite the original) if you want to keep them. Use *compare mode* to return to these settings whenever you make a change you don't like.

This particular sound was created by me for use in an ambient composition written for a holography exhibition, and so I've named it "Hologram". You may, of course, name it anything you like, and for the sake of concerned copyright lawyers everywhere, I declare this software to be public domain freeware. This means that you're more than welcome to use it in any applications you like.

A couple of interesting things occur in this sound that some of you astute envelope-watchers may have realized. Step 5 above points out how the octave-lower re-attack occurs, so you know about that already. Another interesting observation would be that L4 for the middle modulator in the stack (operator 4) is set to 99. The other

Levels for this operator are all at 85 and R4 is quite slow, so obviously what happens here is that the brightness of the sound actually increases *after* "key off". Also, if you play this sound with extensive use of the sustain pedal (and it really lends itself to this) you will find that after you have 16 notes held down and you add a 17th, the tone generators suddenly "realize" that L4 for operator 4 is *not* 0 (which, you may remember, they all assumed beforehand), and you will begin hearing a sharper timbre for these new notes! Of course, operator 5 (the top modulator in the stack) has a more-or-less default square envelope, with a long R1 and R4, but its contribution to operator 3, as mentioned earlier, is defined by the movements of operator 4. **(see figure 9-74)**

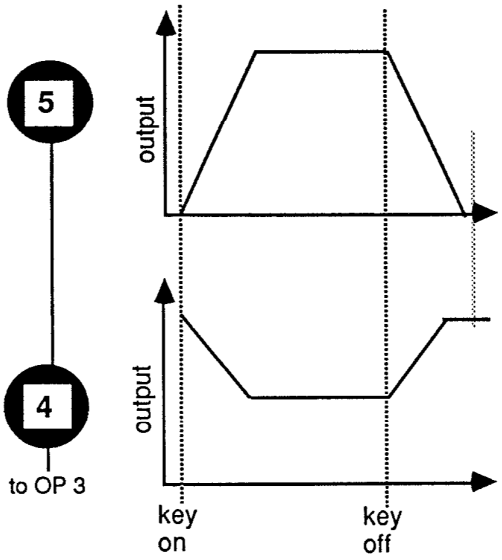


Figure 9-74

Because operator 5 is set to a non-whole number frequency (0.75) relative to operator 3 (set at 1.00), it will have the effect of inducing inharmonic overtones as it fades in. This is why about 45 seconds into the sound, you hear some low inharmonics enter for the first time. If you allow the envelopes to progress to this point and then let go of the key or sustain pedal (generating a "key off"), you'll actually hear the sound become even more inharmonic afterwards! The frequency ratio between operators 4 and 3, coupled with operator 4's output level of 71, tells us that initially a square wave is generated, and that is the case until operator 5 begins fading in and contributing its inharmonics. When we generate "key off", operator 4 (which is being continually changed as operator 5 fades in) increases its output level as it rises to an L4 of 99. **(see figure 9-75)**

Another factor that makes this sound so interesting is the several different complex movements that each operator's EG is making. This means that you can let go of the key at different points and individual notes will have developed to different degrees, making for many different timbres combining and resonating at once.

Practical usage of the operator EGs in creating a sound

We can conclude our discussion of the operator EGs by running an Exercise to construct a simulated acoustic sound. We'll work with a relatively simple one here (more complex examples are given in Chapter Sixteen) and try to make the sound of a woodblock.

There are many ways of describing the sound of a woodblock: our job here is to find those descriptions which are the most directly translatable into DX7II commands. A woodblock, of course, is a clean, hollow, wooden sound which is percussive and therefore basically pitchless (you can pick out some kind of pitch, but you couldn't, for example, tune a woodblock to an A440). It is also a sound of brief duration which does not sustain at all, but, depending upon external acoustics, may hang on a little bit afterward.

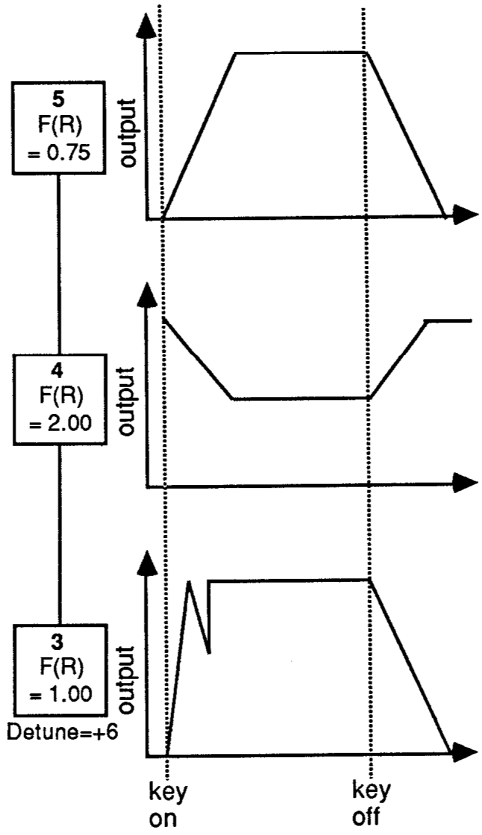


Figure 9-75

Now let's translate these plain English descriptions of the sound into "DX7II-ese": Because a woodblock is not a particularly complex sound, we won't need more than one system to generate it. Because it isn't a particularly bright or distorted sound, we won't need more than one modulator or the feedback loop. This means that we can create our woodblock with any algorithm except #32 (which doesn't provide any modulators at all). The fact that we know this sound to be "hollow" points us in the initial direction of a timbre like a square wave (which is actually missing harmonics) - and the fact that it is percussive tells us that we will need to generate some inharmonics by using a non-whole number frequency ratio. A wooden timbre, unlike a metallic timbre, is

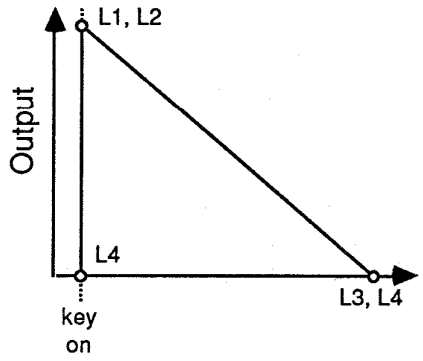


Figure 9-76

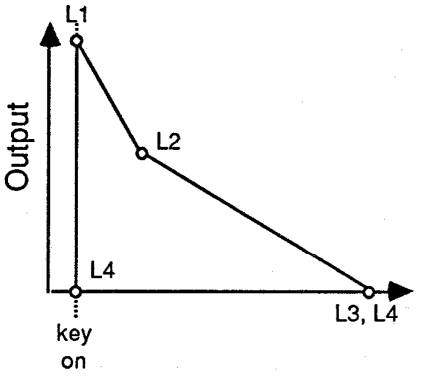


Figure 9-77

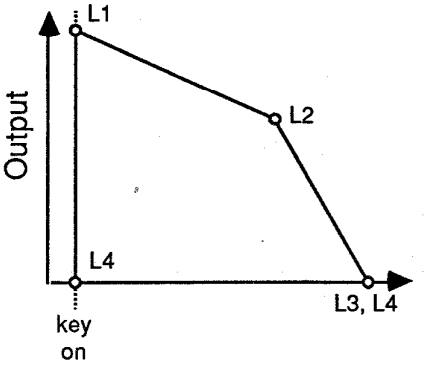


Figure 9-78

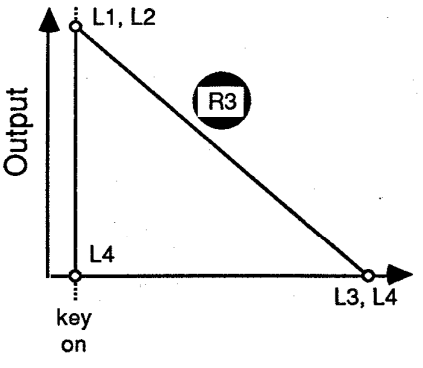


Figure 9-79

one which undergoes a great deal of timbral change over its duration (specifically, a decrease in the amount of overtones generated). Finally, the fact that the sound is of brief duration tells us a good deal about how to construct the carrier envelope. In total, we've really got enough to go on. For the sake of simplicity, we'll just use the system of operators 1 and 2 in algorithm #1 (the default algorithm), since we've determined that we don't need a stack or feedback to accurately construct this sound. Here goes nothing!

Exercise 52
Generating a woodblock sound

1) INITIALIZE your DX7II from single voice play mode, leave it in algorithm #1, TURN OFF operators 3 through 6 ("110000"), and, using the system of operators 1 and 2, GENERATE a square wave. Play a note on the keyboard and listen (Audio Cue 52A).

2) In order to generate some inharmonics within this square wave, we need to set up a non-whole number frequency ratio. Press edit switch 8, followed by edit switch 2, in order to VIEW operator 2. CHANGE operator 2's Fine value to 2.84 (just an arbitrary value I selected for demonstration), play a note and listen (Audio Cue 52B).

3) We now have created a dissonant clarinet-ish sound. The reason we are still far from the woodblock is because both operators in the system still have their default square envelope settings. Let's begin with the carrier envelope. Press edit switch 9, followed by edit switch 1, and change the values for operator 1's EG Levels as follows:

L4	L1	L2	L3	L4
0	99	99	0	0

which generates an envelope which looks like **figure 9-76.** The reasons for picking these Levels 1, 3, and 4 should be fairly obvious: Level 1 is set at maximum output because there's no reason to have any less than maximum output. In fact, whenever constructing any sound at all on the DX7II, you should always ensure that at least one carrier has at least one of its EG Levels set at 99. If not, you are going to output more noise and less signal than would be desirable (for more on this *signal-to-noise ratio*, refer back to Chapter One). Level 3 is set at 0 because a woodblock is a non-sustaining instrument, and Level 4 is similarly set to 0 because we don't want a continuous sound. The only question, in fact, is where you want to put Level 2. If you think it through, there are only three options available to you:

(a) L2 can be the same as L1; or
(b) L2 can be the same as L3; or
(c) L2 can be somewhere in-between L1 and L3.

The only reason for picking option (c) above would be if you wanted to have two *different* volume changes within the overall sound, like in **figure 9-77** or like in **figure 9-78.** A woodblock is not that complex a sound that it actually undergoes two separate changes in volume, so we can eliminate option (c) in this particular example. Well, then, should L2 be the same as L1, or should it be the same as L3? The answer is - it doesn't make the slightest bit of difference. If L2 is set the same as L1 (as we did above), then it is R3 that acts as the "decay" time. **(see figure 9-79)** On the other hand, if we set L2 the same as L3, then it is R2 that acts as the "decay" time. **(see figure 9-80)** The Levels, then, for this particular sound actually suggest themselves when you stop and analyze what it is you need to accomplish. My advice is that you

always begin setting up an EG by determining and entering the Levels first. In this way, you are delineating the points on the "road-map". Next, specify to the DX7II how quickly or slowly you wish to travel from point to point by specifying the EG Rates.

4) Enter the following EG Rates for operator 1:

L4	R1	L1	R2	L2	R3	L3	R4	L4
0	99	99	99	99	72	0	52	0

which generates an envelope which looks like **figure 9-81.** The reason we picked these particular rates should also be fairly obvious. Rate 1 has been set at its maximum value of 99, since a woodblock attacks instantaneously. Rate 2 is, of course, completely meaningless, since L1 and L2 are the same - this value could be anything at all, and I simply left it at its default of 99. Rate 3, the "decay" time in this example, has been set at a value that sounds close to the way a real woodblock decays, and Rate 4 has been set at a value close to the way a woodblock sound might typically ring after the fact.

5) Play a note on the keyboard and listen (Audio Cue 52C). Our sound is closer, but it's much more like a metallic cowbell than a wooden woodblock. The reason for this, as stated earlier, is that wooden timbres typically undergo a great deal of timbral change. Since our modulator still has the default square envelope, our sound is currently undergoing no timbral change - even though the EG of operator 1 is ensuring a significant volume change. Since we have learned that a characteristic of wooden sounds is that they undergo a rapid decrease in overtone amount, we can use the same EG we have already constructed for operator 1 (as shown in **figure 9-81**) to do the job for operator 2. Use the *EG and Scaling Copy* function to copy the EG data for operator 1 into operator 2. You'll also have to now go back and restore operator 2's output level back to 71 (since this copy function also copied operator 1's output level of 99 into operator 2). Play a note on the keyboard and listen (Audio Cue 52D). We're definitely getting closer!

6) The reason we're not quite there yet is twofold: First of all, we haven't yet offset the envelope values - even though we know that in reality there's no way the volume and the timbre of a woodblock would change precisely the same way. Second of all, when we created the timbre in the first place, we really were only guessing at what we would need. Now that we've got the envelopes closer to the way the should be, we'll have a much better idea of the frequency ratio and modulator output levels we'll need in order to accurately simulate this sound. First, let's do the offsets:

7) Change R3 and R4 for operator 2 to new values of 80 and 20, respectively, generating offset envelopes in our system that look like **figure 9-82.** The overtone content, then, dies away more rapidly than the volume, unless you strike a key staccato, in which case the overtones change little as the volume fades away. **(see figure 9-83)** Play a note and listen (Audio Cue 52E). Nearly there! Now we're ready to do the final tweaking to our timbre:

8) Change the output level of operator 2 (our quantitative control) to a new value of 78. Use the F Coarse and F Fine parameters to change the ratio number of operator 2 (thereby changing our qualitative timbral control - the frequency ratio) to 3.69 (you'll have to change the F Coarse value to 3.00 in order to enter in an F Fine value of 3.69. Pretty sneaky, huh?). Play a note on the keyboard and listen (Audio Cue 52F). Instant woodblock!

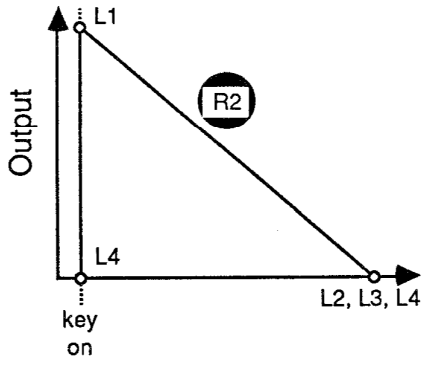
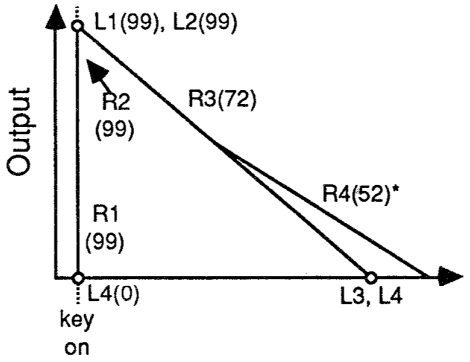


Figure 9-80



*(not used unless keyboard is played staccato.)

Figure 9-81

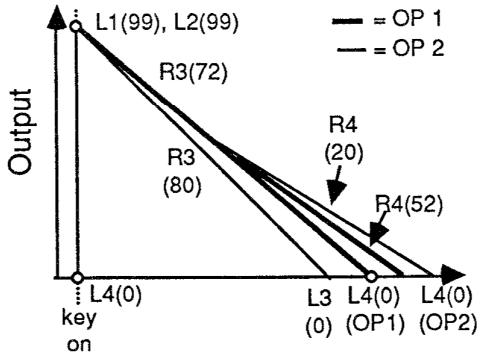


Figure 9-82

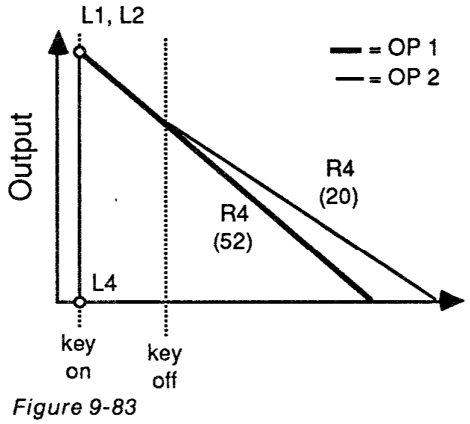


Figure 9-83

9) Name and store this sound somewhere in your DX7II's memory. Then experiment by changing different values and see how you can improve on the sound created here. Note that the sound never sounds perfect over the full five octave range, and that it is usually unrealistic to expect this with almost any one voice. Experiment further by finding one good pitch at which this sound is particularly realistic and then setting that pitch over the whole keyboard by using fixed frequency mode. Remember that in order to do this, you must keep your frequency ratio intact!

Although we took the approach in this last exercise of creating an approximate timbre first and adding in the EG movements second, you may find it more useful in certain instances to actually do things the other way around: set up an approximate EG setting - even with just a single sine wave - and then add in the modulators to create the timbre.

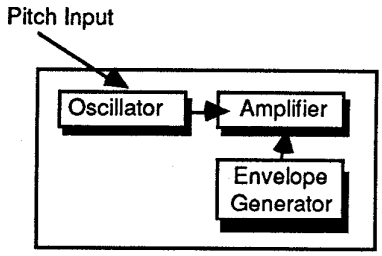


Figure 9-84

The Pitch EG

In none of the Exercises in this chapter were we able to affect the pitch of the sound in any way. A quick look at our diagram of the operator makes this clear. **(see figure 9-84)** Since there is no way that an operator EG can possibly affect the pitch data input. The pitch of most sounds, however, also undergoes change throughout its duration, and this kind of change can be made aperiodically with the use of a completely separate, seventh envelope generator called the *pitch EG*.

The pitch EG is an independent device which resides outside of the six operators and sends data to *all six* of them, *simultaneously*. There is a good reason why Yamaha set things up this way. If we are generating a sound with, say, a modulator-carrier system of operators 2 and 1, and we could use an EG to alter the pitch of operator 2 without in any way affecting operator 1, would we accomplish a pitch change? Cardinal Rule Three (Chapter Six) says we definitely would *not*. Instead, of course, we would be altering the *frequency ratio*, which would result in a *timbral* change.

Since the whole purpose of a pitch EG is to allow us to alter the *pitch* of a sound aperiodically, the wonderful software engineers at Yamaha simply made the pitch EG *non-operator-specific*. Of course, if they *had* made it operator-specific (or, better yet, if they had given each operator its *own* pitch EG), we would be able to generate even more exciting sounds on the DX7II (Dear Yamaha: hint, hint...)*

The pitch EG is a powerful tool and it is in fact rare to find a separate EG on *any* synthesizer which is dedicated to pitch change. It operates exactly the same way as the individual operator EGs in that it has four Levels and four Rates. It also initiates its activity from the same "key on" flag, and reacts the same way to "key off" (immediately returning from whatever point it's at, back to L4, at R4). However, there are a few significant differences. We access it by pressing edit switch 13 (appropriately labeled PITCH EG), which gives us the single LCD display shown in **figure 9-85.**

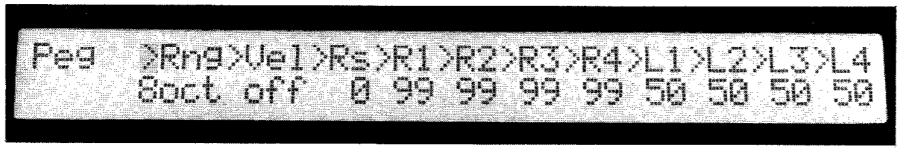


Figure 9-85

*It is perhaps worth mentioning that any operators in fixed frequency mode (see Chapter Six for details) will ignore any pitch EG data sent to them and will therefore remain at the fixed frequency you designate.

Unlike the operator EGs, changes to Levels in the pitch EG will not manifest themselves in *real time* - that is, you'll have to retrigger the key in order to hear the change. Furthermore, in addition to the "Rs" (Rate scaling control), and the usual levels and rates, there are two new controls: "Rng" (which stands for "Range"), and "Vel" (which stands for "Velocity sensitivity"). We'll be talking about velocity sensitivity, as applied here as well as to operator output level, in Chapter Eleven, so just file this last parameter away for future reference. The "Range" control, however, is a new DX7II feature - it was not available in the original DX7 (and it should also be noted that the pitch EG of the original DX7 couldn't be Rate scaled, either).

Position your cursor over the "Rng" parameter and, using the "yes-no" buttons, you will discover the following four options: "8oct", "2oct", "1oct", and "1/2oct". These determine the depth of the pitch EG effect on the six operators. In all instances, a level of 50 represents nominal pitch - that is, whatever pitch the operator takes according to the ratio number and key depressed (if in "ratio" mode), or whatever fixed frequency has been set (in "fixed" mode). Therefore, the initialization default pitch EG values are a bit different from the operator EG defaults in that all four pitch EG Levels will default at a value of 50. The four pitch EG Rates, like their individual operator cousins, default at the maximum value of 99. **(see figure 9-86)**

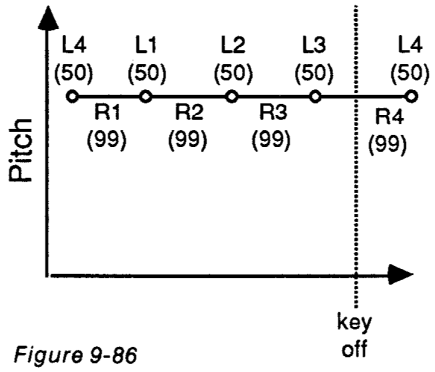


Figure 9-86

Because the pitch EG defaults this way, there is no pitch change occurring when you initialize (since all four Levels are the same value). The reason that a value of 50 was selected for the default Levels is to allow you to use the pitch EG to either sharpen *or* flatten the pitch; if 0 had been selected, you could only sharpen the pitch. **(see figure 9-87)** If 99 had been chosen, you could only flatten it. **(see figure 9-88)**

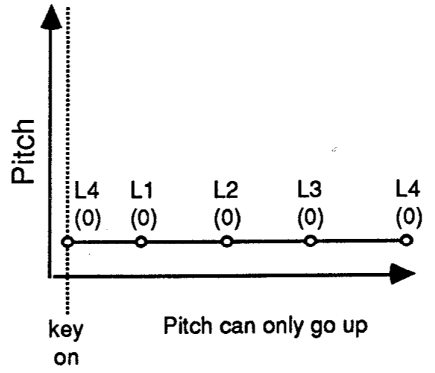


Figure 9-87

Because the four Levels default to the same value, the four Rates are completely irrelevant and could in fact have any value at all. Levels, then, which are greater than 50 will cause the pitch to rise, and Levels less than 50 will cause the pitch to fall. How much the maximum rise and fall will be is determined by the "Range" parameter. If the Range is set at 8 octaves, for example, the pitch can change over a range of four octaves sharper or four octaves flatter (since nominal pitch is a Level of 50, a Level of 0 would be four octaves flatter while a Level of 99 would be four octaves sharper). In the 2-octave range, you can go an octave in each direction; in the 1-octave range, 6 semitones (a half-octave) in each direction; and in 1/2-octave range, 3 semitones (a quarter-octave) in each direction - making this the range of choice when you're after subtle pitch change.

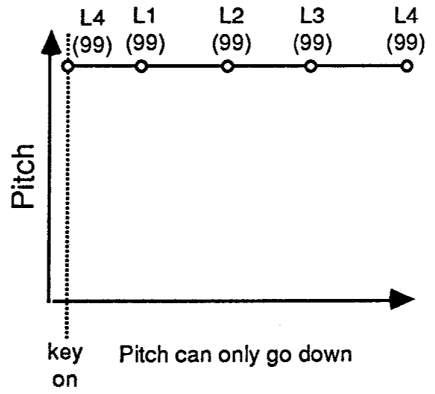


Figure 9-88

Altering the range control does to the pitch EG what changing the operator output level does to the operator EG - it "squashes" it, bringing the Levels closer together and making the absolute speed of the pitch change faster. **(see figure 9-89)**

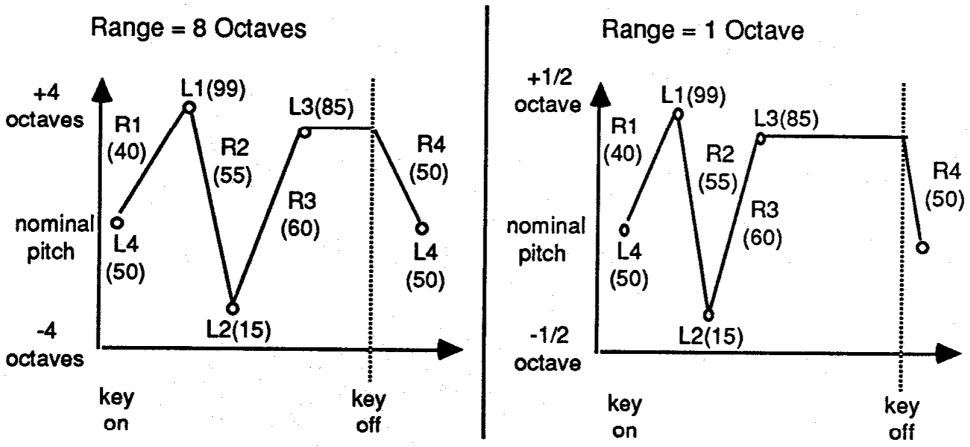


Figure 9-89

Let's run an Exercise now to try out some of our options:

Exercise 53
The pitch EG

1) INITIALIZE your DX7II from single voice play mode, leave it in algorithm #1, TURN OFF operators 3 through 6 ("110000"), and use the system of operators 1 and 2 to generate a sawtooth wave.

2) Press edit switch 13 (PITCH EG) in order to VIEW the pitch EG parameters. Observe the default values: all Levels = 50; all Rates = 99; Range = 8 oct; Rs = 0 (and Vel = off, but we'll talk about that in Chapter Eleven).

3) Change L1 to a new value of 75. Play a note on the keyboard and listen (Audio Cue 53A). Note that all you hear is a kind of "click" at the beginning of the sound. This is in fact the pitch quickly rising and falling - but it is happening so quickly (since R1 and R2 are still both at their default values of 99) that we really can't hear it.

4) Change R1 and R2 both to 35. Play a note on the keyboard and listen (Audio Cue 53B). Note that the you can now clearly hear the pitch slowly rise and fall.

5) Change the value of L1 to 25. Play a note on the keyboard and listen (Audio Cue 53C). Note that the pitch now falls instead of rising.

6) Change L1 to a new value of 0 without changing the Rates. Play a note and listen (Audio Cue 53D). Note that the pitch drops down a full four octaves and that the change in pitch occurs much more slowly than in step 5 above. The explanation for this lies in the fact that, even though the Rate settings remain the same, they have a much greater distance to travel.

7) Restore R2 back to its default value of 99. Play an arpeggiated series of notes and listen (Audio Cue 53E). Note that the pitch now drops slowly and then "springs back" to nominal quickly once it reaches bottom. Note also that the changes in pitch follow the arpeggiation, since each tone generator treats a new note as a separate event - even though there is only *one* pitch EG.

8) Change R2 back to 35. Change the "Range" parameter to 2 octaves. Play a note and listen (Audio Cue 53F). Note that the pitch now drops down only one octave, and that this change occurs more quickly than in step 7 above, since the rates have a lesser distance to traverse.

9) Change the "Range" parameter to 1 octave. Play a note and listen (Audio Cue 53G). Here the pitch only drops down 6 semitones (half an octave), and the change occurs more rapidly still.

10) Change the "Range" parameter to 1/2 octave. Play a note and listen (Audio Cue 53H). Note that you now hear only a slight pitch change (3 semitones - a quarter of an octave), and that it occurs very rapidly.

11) Restore the Range back to 8 octaves, and change L1 to a new value of 52. Play a note and listen (Audio Cue 53I). Note that we now hear a very subtle (and fast) wavering in pitch, since we are accomplishing an aperiodic pitch change of less than a quarter tone.

12) Change L1 to a new value of 48, play a note and listen (Audio Cue 53J). Note that we hear a slightly different wavering in pitch as we are now rising up to our pitch from a slightly flatter note. These kinds of subtle pitch changes bring to mind the kind of gentle pitch bends that are characteristic of much Eastern music.

13) Change L1 to a new value of 40 and experiment by changing the Range to 2 octaves (Audio Cue 53K), 1 octave (Audio Cue 53L), and 1/2 octave (Audio Cue 53M). Note that the pitch change is much more subtle, and much more rapid, as we reduce the range - to the point where it is barely noticeable in the 1/2 octave range.

14) Change L1 to a new value of 25 and restore the Range to its default of 8 octaves. Change the Rate scaling for the pitch EG to its maximum value of 7. Play C1 (the lowest note on the keyboard) and listen (Audio Cue 53N), and then play C6 (the highest note on the keyboard) and listen (Audio Cue 53O). Note that the pitch change is far more rapid for C6 than it is for C1. You can hear this even more clearly by playing C1 and C6 at the same time (Audio Cue 53P). Experiment further with the Rate scaling control and note how the pitch changes are altered in speed over different parts of the keyboard.

15) Experiment further by making various changes to different pitch EG Levels and Rates, as well as to the Range and Rate scaling controls, and by altering the pitch EG values for various presets. Note the way this affects the overall sound. Note also that no matter what sound you affect with the pitch EG, *only* a pitch change occurs because it always affects *all* operators equally.

While there are relatively few presets that utilize the gross settings of the pitch EG, there are many that utilize the subtler effects possible. Remember that most if not all acoustic sounds typically undergo some kind of pitch change - generally a subtle one - throughout their duration. Stringed instruments as well as drums, for example, undergo a rapid sharpening in pitch when they are first played. This occurs as a response to the string or skin being stretched slightly upon initial impact. The pitch EG, like beating effects, generally falls under the category of Icing On The Cake: it will usually be something you try at the end - rather than the beginning - of creating a sound, but it often will prove to be an integral part of the sound.

Having said all that, let's examine an interesting and novel preset which utilizes the pitch EG for gross pitch change. The voice we are referring to is called "LateDown", and will be found in your ROM cartridge, bank 1, slot 51. Before we examine anything in the sound, take a moment to listen to it first, and let's see what deductions we can reach as to how it was programmed.

This is a pleasing, brass-like sound that seems to deliver two different pitches - one when the note is struck, and another, two semitones lower - when the key is released. Aha! The word "released" should set off a little light bulb above your head. You should now

(correctly) be assuming that Level 4 and Rate 4 - in both the pitch EG and possibly in one or more operator EGs - are at play, since they are the only things that occur *after* "key off". The fact that we are hearing an aperiodic (once only) pitch change tells us that the pitch EG is definitely at work here.

Try playing C1 (the lowest note) and then C6 (the highest note). Do the pitch changes occur more rapidly in the upper registers? They do not - indicating that the pitch EG is *not* rate scaled. However, *something* is changing from lowest to highest note: it's the actual total duration of the sound. This tells us that the carrier envelope *is* rate scaled. Now try playing C3 (middle C) and hold the note down. After a few seconds, the sound disappears - telling us that Level 3 for the carrier or carriers is definitely at 0. Let go of the note, and - lo and behold! - there is no pitch change! Obviously, once the carrier has dropped to a sustain level of 0, there can be no more sound - unless Level 4 of the carrier is greater than 0. The fact that the sound is not continuous tells us that this cannot be the case.

Are you starting to get the picture? The ear can tell us many things about the sound - once the mind understands the meaning of the different parameters. The eye will come into play, now, as well - as we put "LateDown" into edit mode and see what some of the actual numbers are!

Algorithm #16:

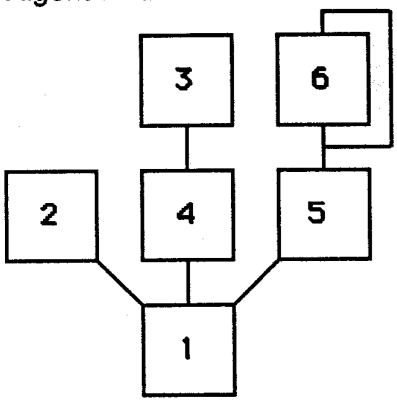


Figure 9-90

Exercise 54
Examining the "LateDown" pitch EG

1) Put your DX7II into single voice play mode and call up "LateDown" from your ROM cartridge, bank 1, slot 51. Play a few notes to confirm that the sound is in your edit buffer (Audio Cue 54A).

2) Press the edit mode select switch in order to VIEW the edit parameters for this sound.

3) Press edit switch 7 (ALGORITHM) and note that algorithm #16 was used for this sound. Observe the diagram on the front panel and note that operator 1 is the only carrier in this algorithm. **(see figure 9-90)**

4) Press edit switch 9 (EG), followed by edit switch 1, in order to VIEW the EG values for operator 1. They are as follows:

RS	L4	R1	L1	R2	L2	R3	L3	R4	L4
6	0	40	99	0	99	21	0	33	0

yielding an envelope which looks like **figure 9-91.**

5) The Rate scaling value of 6 explains why this sound is shorter for higher notes (since this is the carrier that is being scaled). However, the critical values here are L3, L4, and R4. The L3 of 0 confirms why, if we hold a note down for about 30 seconds, the sound fades away completely and we hear no pitch change after "key off" (Audio Cue 54B). The L4 value of 0 explains why the sound is not continuous, and the R4 of 33 explains why we continue to hear some sound after we let go of the key. Experiment by changing R4 for operator 1 to a new value of 99. Play a few notes and listen (Audio Cue 54C). Note that we now hear *no pitch change* whatsoever. Obviously, whatever pitch change is happening is occurring in the L3-R3-L4 portion of the pitch EG. Restore R4 for operator 1 back to 33, and let's take a look at the pitch EG settings.

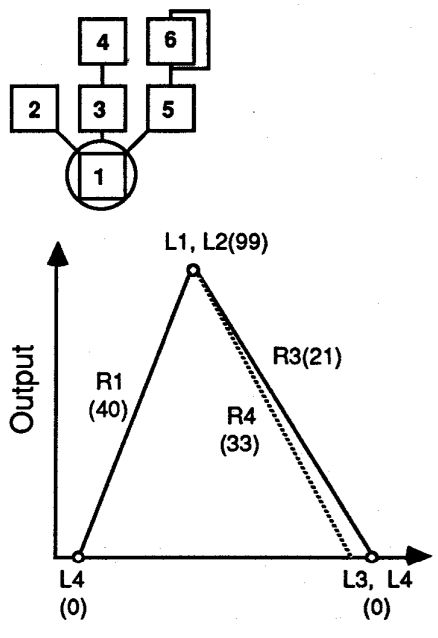


Figure 9-91

6) Press edit switch 13 (PITCH EG) and observe the pitch EG values for this preset, which are as follows:

Rng	Rs	L4	R1	L1	R2	L2	R3	L3	R4	L4
8oct	0	45	99	48	99	53	99	50	99	45

This generates an envelope which looks something like **figure 9-92.**

7) Play a note on the keyboard and listen closely. As you do so, follow the diagram above so you can correlate the pitch change you hear with the one you see. When you first generate a "key on" by depressing a key, the pitch EG begins traveling from an L4 of 45 to an L1 of 48 - in other words, it starts out *flat* of nominal pitch. But this first shift happens so rapidly (R1=99) that we can't hear it at all. From the L1of 48, it immediately (R2=99) proceeds to L2 of 53 - and then immediately (R3=99) on to the sustain L3 of 50 - which is at nominal pitch. This is why, as long as we hold our finger down on the key, the pitch undergoes no change whatsoever. However, when we let go of the key ("key off" flag is generated), two things happen: First of all, the carrier (operator 1) EG begins to slowly (R4=33) die away to an L4 of 0. Secondly, the pitch EG *immediately* (R4=99) drops down to its L4 of 45. In the 8-octave range, this pitch change of 5 increments is equivalent to two semitones.

8) Experiment by changing some of the pitch EG values and listening closely to the resultant change in the sound. Try, for example, changing the Range to 2 octaves (Audio Cue 54D), or (with the Range back to 8 octaves) slowing down R4 of the pitch EG to a value of 20 (Audio Cue 54E), or changing L1 and L2 to the nominal 50 values while changing L4 (again with the Range at 8 octaves) to a new value of 55. Note that this last alteration allows you to get automatic trilling effects! (Audio Cue 54F) Try some more changes out and see how you can improve the sound.

9) Finally, experiment by removing the effects of the pitch EG altogether from this voice. How is this accomplished? Simple - by changing all four pitch EG levels to 50 (the initialization defaults), there will be no pitch change whatsoever. Listen (Audio Cue 54G) and note how this lack of pitch change completely changes the entire sound!

Note that the DX7II tone generators are able to "recognize" a starting L4 point in the pitch EG right away - they don't automatically assume a starting L4 of 0, as they do with the operator EGs.

Bear in mind, as we saw in this last exercise, that the changes initiated by the pitch EG are in fact somewhat tied in with the actions of the individual operator EGs (although, of course, their values can be entirely different). For example, as we observed with "LateDown", if the carrier R4 is at maximum (99), you won't hear *anything* - including any kind of pitch change - after "key off". **(see figure 9-93)**

On the other hand, if your carrier EGs have very slow attack rates, then you won't be able to hear rapid initial pitch EG changes. **(see figure 9-94)**

Also note that the *EG and Scale Copy* function *cannot* be used with the pitch EG. This can only copy output level (and level scaling data) and individual operator EG (and rate scaling) data from one operator to another.

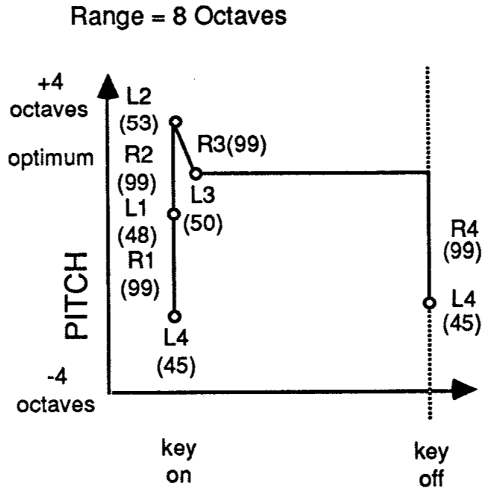


Figure 9-92

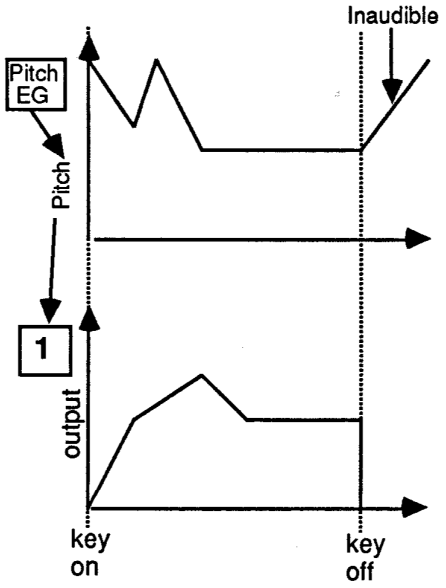


Figure 9-93

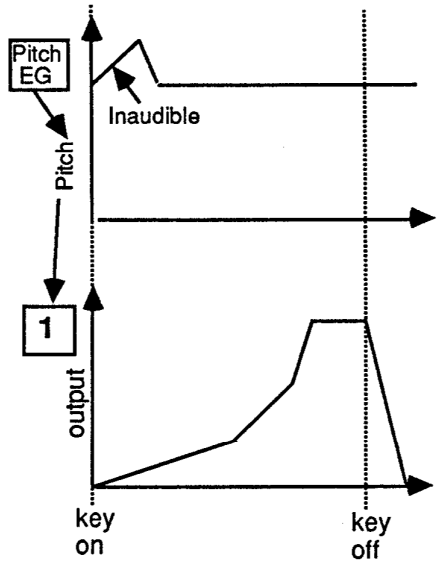


Figure 9-94

Hopefully, the exercises and examples in this chapter have demonstrated how incredibly important the use of envelopes are in synthesizing sounds. The use of the DX7II EGs is absolutely mandatory in the creation of *all* sounds with this instrument. Their very complexity allows us new freedom in audio synthesis. For these reasons, an entire volume could be written on them. The good news is that this book won't be it. It's time to move on now to the cousin of the EG, the device that allows us to make *periodic* changes to our sounds: the *LFO*.

Chapter Ten ◇
The LFO

Sounds always change over time; we have seen in the last chapter how applying an envelope to a sound causes it to become dynamic and "real". However, envelope generators by their very definition are only capable of causing an *aperiodic* change to a sound; in other words, the change caused by an EG can never be repeated without actually repeating the sound itself.

Within the duration of a single sound, however, *repetitive* changes will often be observed. For example, a *vibrato* is a repetitive pitch change; as a tremolo is a repetitive volume change. Therefore, we will require some kind of device in our synthesizer that is capable of causing a *periodic* change to a sound over time; and that device, on all synthesizers, is called a *low frequency oscillator,* or *LFO* for short.

The digital oscillators inside our six operators are capable of generating numbers at very high speeds, and so the analog voltages produced by our DAC as a result of these signals undergo high-speed changes, in the audible (20 Hz to 20 kHz) range. In other words, these oscillators are capable of producing digital signals which become audible sounds. The LFO is a different kind of digital oscillator that lives entirely outside of the six operators. There is only one LFO in a DX7II, not six,* and that lone LFO is only capable of generating signal at *low* speeds (in the 0.10 Hz to 60 Hz range). That means that the signals it generates are mostly *subsonic*. The fact that the LFO only produces inaudible signals is largely irrelevant, since there is no way of routing its signal to the DAC anyway - in other words, we can never actually hear the LFO - like the EGs, we will only be able to hear its *effect* on the operators. The LFO in the DX7II is an independent digital component, completely separate from the six operators, which will send data to some or all of those operators, causing them to periodically (repeatedly) change their *pitch* and/or *output level.* Of course, Cardinal Rule One tells us that if the output level of a carrier is periodically changed, the result will be a periodic volume change. Cardinal Rule Two tells us that if the output level of a modulator is periodically changed, the result will be a periodic quantitative timbral change.

We can send this LFO signal to either of two different destinations. It can terminate at the pitch data input of the operators **(see figure 10-1)** or it can terminate at the amplifiers of the six operators. **(see figure 10-2)**

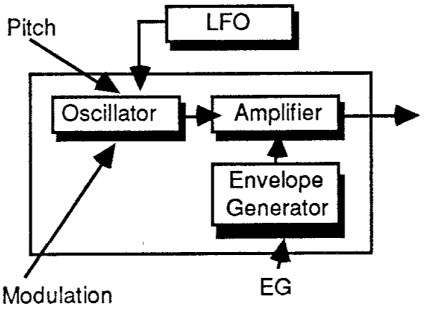


Figure 10-1

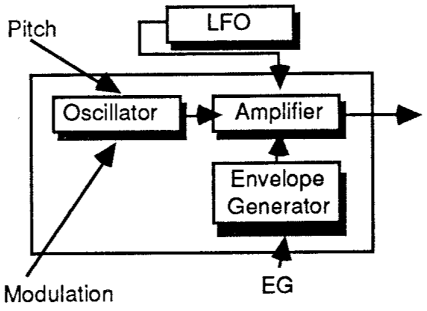


Figure 10-2

* Technically speaking, there are actually *sixteen* LFOs in the DX7II, since each tone generator has its own. However, the actions of all sixteen of these LFOs are determined by the set of commands that we issue a single *virtual* (that is, software-based) LFO. We'll talk more about this shortly, when we discuss the LFO *multi* mode.

If we decide to use the LFO as in the first example (LFO data to the pitch data inputs), then we are said to be accomplishing *pitch modulation.* Pitch modulation is always non-operator-specific; that is, this data will always be sent to all six operators simultaneously - and all six operators will always respond to this data in the same way as one another. This is exactly the same logic that is used by the aperiodic pitch control, the pitch EG. If you remember, the pitch EG also only ever affects all six operators at once - and all six operators always react to this data in the same way. Again, if we apply any kind of pitch change to certain operators and not all of them, Cardinal Rule Three tells us that we will not get a pitch change, but a qualitative timbral change instead.

On the other hand, if we decide to follow the second example and route the LFO data to the operator amplifiers, then we are said to be accomplishing *amplitude modulation.* Amplitude modulation *is* operator-specific; that is, the data is sent to all six operators, but we can specify which operators will respond to the LFO data (and to what degree they will respond) and which will not. This means that, depending upon whether we have carriers or modulators responding to the LFO commands, we will hear either a periodic volume change or a periodic quantitative timbral change.

All the LFO functions are accessed by a single switch, edit switch 12. Pressing this switch will yield the LCD display in **figure 10-3.**

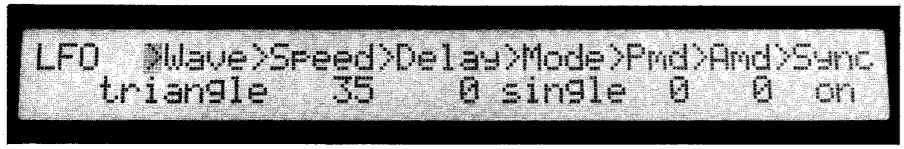


Figure 10-3

DX7II LFO Waveshapes :

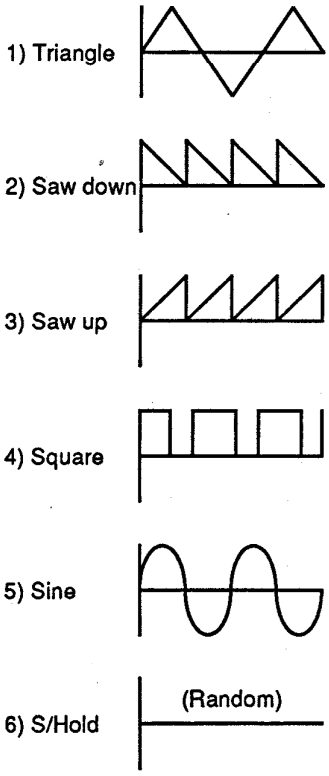


Figure 10-4

Note that this switch is *non-operator-specific* (we know this because of the absence of an "OP" number in the upper left-hand corner of the LCD). We will also be using the operator-specific edit switch 11 (SENSITIVITY) in order to determine which operators will be sensitive to amplitude modulation and how sensitive all six operators will be to pitch modulation. In addition, we'll also work with operator-specific edit switches 25 and 26 in order to route the LFO signal via some of the DX7II's *real-time controllers*: the *modulation wheel,* the *breath controller*, the keyboard *after-touch*, and the *foot controllers*. Let's begin, however, with a brief description of the seven LFO parameters shown in the edit switch 12 LCD display.

LFO Wave

When we use the EGs to change our sounds, we are able to "shape" them in various different manners by specifying different Levels and Rates. Our LFO signal cannot be "shaped" in this manner; instead, we are provided with six fixed LFO waveshapes and we are asked to select one of them. The shape of the wave will of course have a great impact on the type of periodic movement we will ultimately hear when we apply this signal. The six different waveshapes we have to choose from are in **figure 10-4.**

The *triangle, square*, and *sine* waves should already be familiar to you. So, too, should be the *sawtooth*, although here we are offered the choice of a negative-going sawtooth wave (*saw down*) or a positive-going sawtooth wave (*saw up*). The *s/hold* (sample/hold) waveshape

is a new one, but that's OK since it isn't really any kind of wave at all. Instead, when you select the s/hold shape, the DX7II generates a stream of *random* numbers. This will allow us to generate random periodic changes to our sounds, and will, in specific usages, yield quite interesting effects. Remember that since we only have one (virtual) LFO, we can only select one LFO waveshape for any given voice.

Initialize your DX7II, press edit switch 12, and position the cursor over this parameter. The voice initialization default is a triangle wave. You can use the "yes-no" buttons or the data entry slider to VIEW the six different waveshape options available. You won't hear any kind of audible change to your default sine wave just yet - in a short while, we'll run an exercise that explains why.

LFO Speed

This one's real easy - and pretty much self-explanatory. The range of this control is 0 - 99, with 0 being the slowest possible speed (about 0.10 Hz - or one wave generated every ten seconds!) and 99 being the fastest (about 60 Hz). The voice initialization default for this parameter is a speed of 35. Position the cursor over this parameter and use the data entry slider to change it through its full range. Again, you won't yet be able to hear any kind of change in the sound just yet.

LFO Delay

This is a digital delay line built into the DX7II which will serve to delay the onset of the LFO signal by a certain amount. The range of this switch is, again, 0 to 99, with 0 indicating minimum (no) delay; and 99 indicating maximum (about 3 seconds) delay. This will be useful in setting up sounds which will not undergo any periodic change upon their initial attack - for example, you could set up a sound which undergoes no change if played staccato but develops a pronounced tremolo when played legato! Unlike the LFO Delay control in the original DX7, the delay line is *not* activated for new notes if notes are currently being held down - in other words, it is only the *first* note in a chord or series of legato notes that will have its LFO signal delayed. Another anomaly of this control is that, while the delay to the onset of the LFO signal can be quite long, once the LFO signal actually begins arriving, the "fade-in" time is actually very short and so is fairly unrealistic.

Position the cursor over this parameter and use the data entry slider to step it through its possible values. The voice initialization default for this parameter is 0 (no delay). As before, there will be no audible change to the sine wave you are hearing.

Mode

This allows us to use the LFO in either *single* or *multi* mode. In single mode, all sixteen actual LFOs (see footnote above) will be synchronized and act as if they are one big LFO at all times. In multi mode, they will each act in a more independent fashion. A detailed discussion of this will follow shortly.

The initialization default for this is "single" mode. Position your cursor over this parameter and use the "yes-no" buttons or data entry slider to change this between "single" and "multi". As before, there will be no audible change to the sine wave you are hearing.

Pmd

Pmd stands for "Pitch modulation depth" and this control is used for the direct routing of LFO signal to the pitch inputs of all six operators simultaneously. The range of this control is also 0 - 99; with 0 indicating minimum (no) signal being routed and 99 indicating maximum (all) signal being routed. We will be running an Exercise shortly to try this out. For now, just position your cursor over this parameter and use the data entry slider to change it through its range of 0 to 99. The voice initialization default for this is a value of 0 (no Pmd).

Amd

Amd stands for "Amplitude modulation depth" and this control is used for the direct routing of LFO signal to the amplifiers of all six operators simultaneously. As with the Pmd control, the range is again 0 (minimum, or no routing) to 99 (maximum, or all routing). Position your cursor over this parameter and use the data entry slider to change it through its range of 0 to 99. Like the Pmd control, the voice initialization default is also 0 (no Amd).

Sync

This parameter is the LFO keyboard synchronization control. In plain English, it will allow us, if we desire, to trigger the LFO from the "key on" flag generated by the action of depressing a new key on the keyboard. We'll run an exercise later to demonstrate its use, but for now just note that this control is either "on" or "off" (according to which "on-off", that is, "yes-no" data entry button we press). Upon initialization, this parameter will default to being ON.

Let's begin by exploring the actions of the LFO being used for pitch modulation. This will, of course, give us a periodic change to the pitch of the sound we are generating. Having decided that we want to use the LFO for this purpose, the DX7II, in true computer fashion, now brings us to another fork in the road. **(see figure 10-5)**

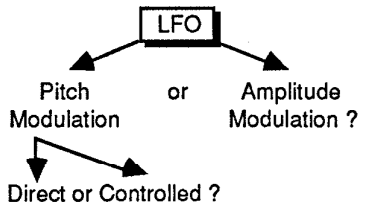


Figure 10-5

We have the option of either *directly* routing the LFO signal to the pitch inputs of our six operators, in which case we will need to use the Pmd parameter to accomplish *direct modulation,* or we can route the LFO signal indirectly, via any one of the afore-mentioned real-time controllers (the modulation wheel, the breath controller, the keyboard after-touch, or the foot controllers), in which case we will accomplish *controlled modulation.* In either event, we will need to use edit switch 11 (SENSITIVITY) in order to specify how sensitive the six operators will be to pitch change from the LFO. Let's start by setting up a *direct* pitch modulation:

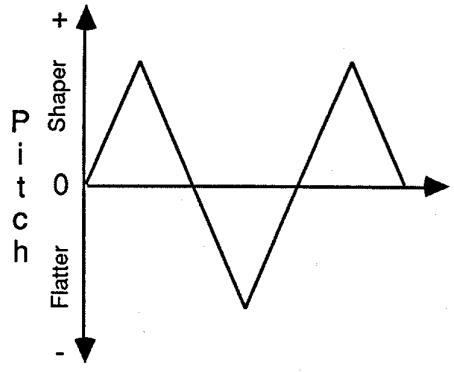


Figure 10-6

Exercise 55
Direct pitch modulation

1) Initialize your DX7II from single voice play mode, leave it in algorithm #1, TURN OFF operators 3 through 6 ("110000"), and use the system of operators 1 and 2 to generate a sawtooth wave.

2) Press edit switch 12 in order to observe the various LFO parameters. They are all currently at their defaults: triangle Wave, Speed of 35, Delay of 0 (no delay), Mode is "single", Pmd of 0, Amd of 0, and Sync ON. Press edit switch 11 in order to VIEW the sensitivity parameters. The LCD looks like **figure 10-6.**

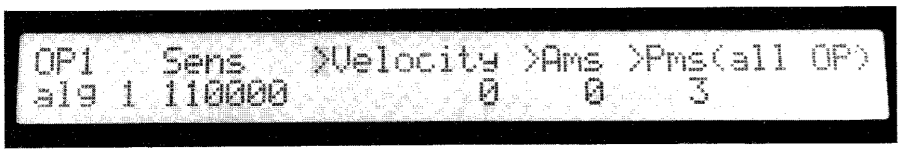


Figure 10-7

The "Velocity" sensitivity has nothing to do with the LFO and will be discussed in great detail in the next Chapter, so let's ignore that for now. "Ams" stands for "Amplitude modulation sensitivity" and refers to how sensitive a particular operator's amplifier will be to LFO signal. The "Amd" parameter in edit switch 12 showed us that currently no LFO signal is being sent to the operator amplifiers (since it defaulted to 0), so we can ignore this value for the moment as well. The "Pms" parameter refers to how sensitive *all six* operators will be to LFO pitch modulation - and this is the parameter we want to concentrate on now. Observe that there currently is a sensitivity for pitch modulation (since this parameter defaults to a value of 3).

3) Play a note on the keyboard and listen (Audio Cue 55A). Note that there is currently no periodic change occurring to the pitch of the sawtooth wave we are hearing. This is because no LFO modulation is being routed as yet to the pitch inputs of operators 1 and 2 (since Pmd = 0) even though both are already sensitive to such a modulation (since Pms = 3).

4) Press edit switch 12 again and use the cursor switches in order to position the cursor over the "Pmd" parameter. Use the data entry slider to change this to its maximum value of 99. Play a note on the keyboard and listen (Audio Cue 55B). Note that you are now hearing a rapid periodic pitch change in the sound.

5) Press the left cursor switch three times in order to position the cursor over the Speed parameter and change this to a slower speed value of 8. Play a note and listen (Audio Cue 55C). Note that the speed of the pitch change has now slowed appreciably and that you can now clearly hear the pitch gradually rising and falling, equivalent to the current triangle LFO waveshape. **(see figure 10-7)**

6) Press the left cursor switch once again in order to position the cursor over the Wave parameter and press the "yes" button once to hear the effect of a saw down LFO wave (Audio Cue 55D). Note that the pitch now instantly rises and slowly drops, as equivalent to the wave shape. **(see figure 10-8)**

7) Press the "yes" button once again to hear the effect of a saw up LFO wave (Audio Cue 55E). Note that the pitch now changes in the opposite manner, slowly rising and quickly dropping, as equivalent to the wave shape. **(see figure 10-9)**

8) Press the "yes" button once again to hear the effect of a square LFO wave (Audio Cue 55F). Note that the pitch now instantly rises, followed by an instant fall, as equivalent to the wave shape. **(see figure 10-10)**

Note that this produces a *trill* effect to the sound. Position the cursor over the Pmd parameter and use the data entry slider to slowly change this value to its minimum of 0, holding down a key and listening as you do so (Audio Cue 55G). Note that changing the Pmd value has the effect of changing the two notes you hear; so that varying the depth of the pitch modulation effectively changes the higher and lower points of the modulating square wave. **(see figure 10-11)**

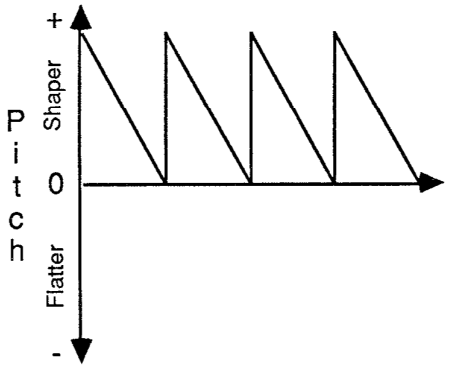


Figure 10-8

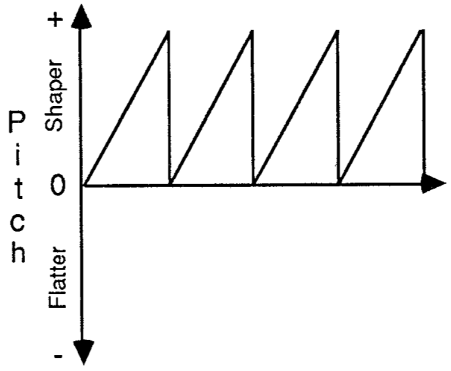


Figure 10-9

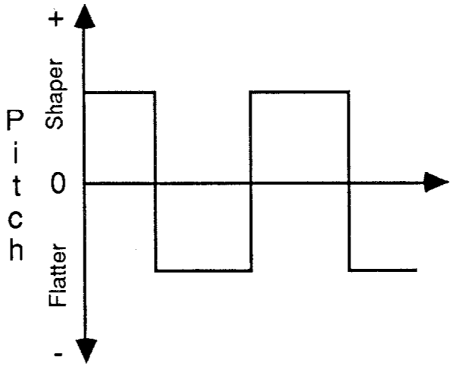


Figure 10-10

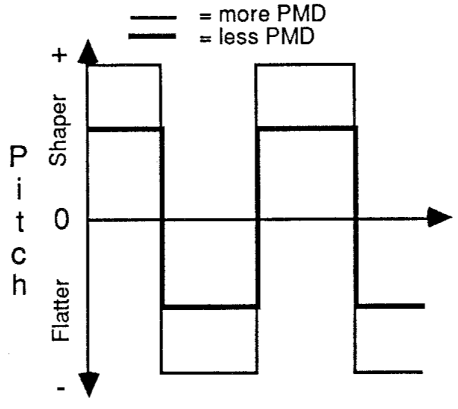


Figure 10-11

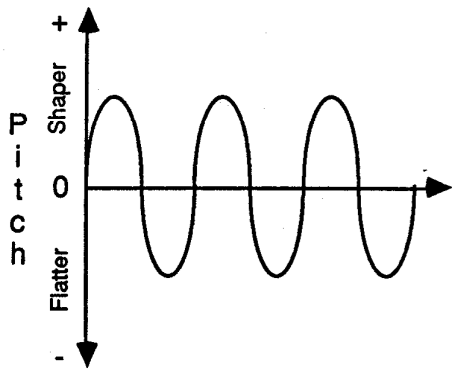


Figure 10-12

9) Restore the Pmd value to 99. Position the cursor back over the Wave parameter and then press the "yes" button once again to hear the effect of a sine LFO wave (Audio Cue 55H). Note that the pitch now gently rises and falls, in a manner equivalent to the wave shape (**see figure 10-12**) and similar to the effect of the triangle wave, but more gently, since the sine wave has a more rounded shape.

10) Press the "yes" button one last time to hear the effect of a S/hold LFO wave (Audio Cue 55I). Note that you now hear a random pitch change, since the DX7II is now periodically *sampling* from a stream of random numbers, and *holding* the number just long enough to send the data to the pitch inputs of the six operators.

11) Restore the LFO shape to that of a triangle wave. Press the right cursor switch once in order to position the cursor over the Speed parameter and, using the data entry slider, slowly change this value up to its maximum of 99, holding down a key and listening as you do so (Audio Cue 55J). Note how this change in speed does not in any way alter the *type* of pitch change you hear, nor the *amount* of pitch change. Note also that this change is recognized by the DX7II in *real time.* Now use the data entry slider to slowly change this value back to its minimum of 0, holding down a key and listening as you do so (Audio Cue 55K). Note that the speed of the pitch change now decreases, and that at the absolute minimum speed of 0, it occurs only about once every ten seconds (indicating an LFO frequency of approximately 0.10 Hz). note also that the pitch change occurs in steps, rather than smoothly, at this speed. This slight *quantization* occurs because the LFO is a software, as opposed to hardware, component.

12) Restore the LFO speed back to a value of 8 and press the right cursor switch three times in order to position the cursor over the Pmd parameter. Use the data entry slider to slowly change this value back to the minimum of 0, holding down a key and listening as you do so (Audio Cue 55L). Note that this has the effect of lessening the **depth** of the change, but in no way alters the **shape** of the change nor the speed.

13) Restore the Pmd value back to 99 and press edit switch 11 (SENSITIVITY). Position the cursor over the Pms parameter and use the "yes" button to slowly change this value to its maximum of 7, holding down a key and listening as you do so (Audio Cue 55M). Note that this has a similar effect to increasing the Pmd value. Now use the "no" button to change this slowly back to its minimum value of 0, holding down a key and listening as you do so (Audio Cue 55N). Note that this has the same audible effect as decreasing the Pmd value, and that at a value of 0 there is no pitch change whatsoever.

14) Restore the Pms parameter back to its default value of 3 and press edit switch 12 again. Position the cursor over the Delay parameter and use the data entry slider to change this to a value of 50. Play a note on the keyboard and listen (Audio Cue 55O). Note that the effect of the LFO on the pitch is now delayed by about a second, but that the ultimate pitch change is precisely the same as before. Note also that this delay is only initiated for new staccato notes, and not for new notes added to held ones or to held chords. Change the Delay value to its maximum of 99, play a note and listen (Audio Cue 55P). Note that the onset of the LFO is now delayed by some 3 seconds.

15) Experiment by setting various different LFO waveshapes with various speeds and delay times. Also Experiment with different Pmd and Pms values until you feel confident in understanding how each of these different controls alter the final modulation effect.

What have we learned from this Exercise? Clearly, the LFO shape determines the qualitative *type* of pitch modulation; the LFO speed determines the *speed* of the pitch modulation; and a combination of the Pitch modulation depth and Pitch modulation sensitivity determine the quantitative *depth* of the pitch modulation (that is, how far the pitch strays from its original setting). This last point poses an obvious question: Why are we given *two* controls for the modulation depth, and what is the difference between them?

The answer is simple: The Pmd control is used *only* when you wish to set up a *direct pitch modulation.* If you wish instead to route the LFO signal via one of the real-time controllers, then you would want the Pmd value to be 0. The Pms control, on the other hand, determines how sensitive all six operators will be to *any* kind of pitch modulation, be it direct or *controlled* (or, in certain instances, both). It seems strange, but, although we access the Pms parameter from edit switch 11 (which *is* operator-specific), we don't need to select any particular operator since the Pms parameter itself is *not* operator-specific. Confusing? I agree.

We route the LFO signal through one or more of the real-time controllers by using either edit switch 25 (for the modulation wheel, breath controller, or keyboard after-touch) or edit switch 26 (for the foot controllers or an external MIDI controller). Unlike the original DX7, these routings in the DX7II are *voice-specific*; that is, they are remembered when you store a particular voice to memory and recalled when you then call that voice up. Let's take a look first at the three LCD displays of edit switch 25. As usual, you cycle between this displays by pressing the edit switch repeatedly. **(see figure 10-13)**

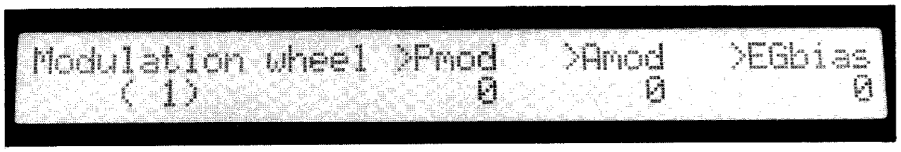


Figure 10-13(a)

Figure 10-13(b)

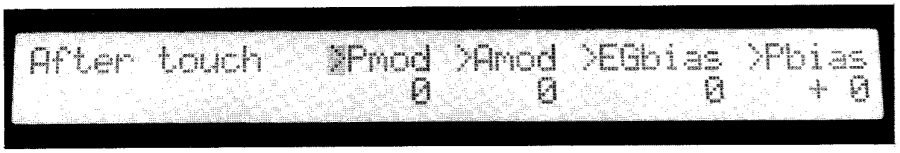


Figure 10-13(c)

The displays and so the potential actions of each of these three real-time controllers is exactly the same (that is, "Pmod", "Amod", and "EGbias"), except that the breath controller and keyboard after-touch can also route a signal called "Pbias", which stands for *pitch bias.* This has nothing to do with the LFO, and so will be discussed in detail in Chapter Thirteen ("Voice Edit Parameters"). Edit switch 26 yields three different LCD displays. **(see figure 10-14)**

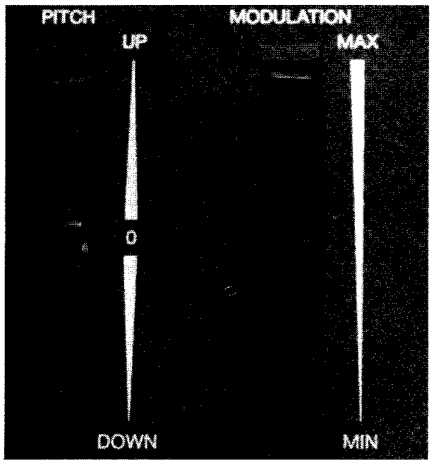


Figure 10-15

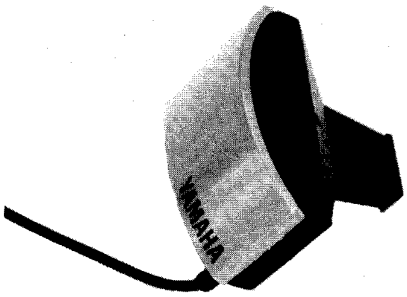


Figure 10-16

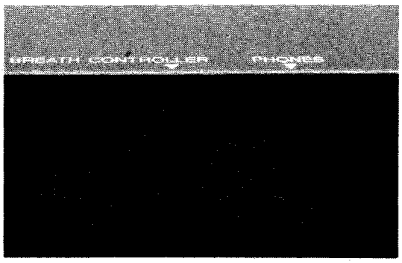


Figure 10-17

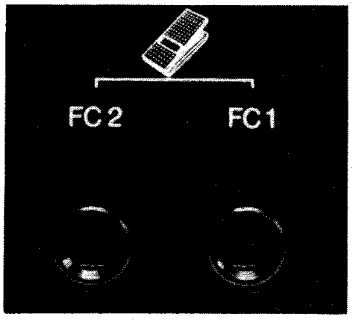


Figure 10-18

Figure 10-14(a)

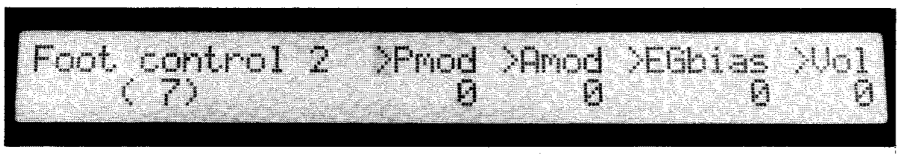


Figure 10-14(b)

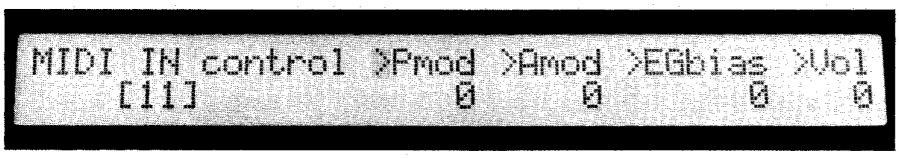


Figure 10-14(c)

Again, you cycle between these by pressing the switch repeatedly. The numbers in parentheses in the lower left-hand corner refer to the MIDI controller number. This is a concept that will be explained in greater detail in Chapter Fifteen ("MIDI"). Foot controller 1 ("FC1") can be assigned to perform the same function as Continuous Slider 1 ("CS1") (this will be covered in greater detail in Chapter Fourteen. Beyond that, it performs the same "Pmod", "Amod", and "EGbias" functions as Foot Controller 2 ("FC2").

Each controller is, of course, physically quite different. The first of these, the *Modulation Wheel*, is located directly to the left of the keyboard, and is labeled as such. This wheel has a smooth action and normally should be at its bottom-most (labeled "MIN") position. This ensures that it is not currently routing any signal. **(see figure 10-15)**

The *Breath controller* is an optional device which looks like **figure 10-16.**
It is normally held between the lips and is meant to be blown into. The breath controller is capable of reacting to wind pressure: as you blow harder or softer, it will route greater or lesser amounts of modulation signal. This must be plugged into the "BREATH CONTROLLER" mini-jack input on the front of your instrument, next to the pre-amplified PHONES jack (used for headphones). **(see figure 10-17)**

The keyboard *After touch* requires just a bit of explanation. This controller is a pressure sensor located beneath every key on the keyboard. The harder you press down on a key, the more this sensor "complains" and the more modulation signal is allowed to pass. This is a *monophonic* control (unlike the keyboard *Velocity sensitivity*, which will be covered in Chapter Eleven), which means that whichever key in a group of keys is pressed the hardest will determine the overall amount of signal routed. In other words, you cannot have varying degrees of signal routed for specific notes in a chord, just by pressing some notes harder and other notes softer.

The *Foot controls* are push-pull type foot pedals (NOT the piano-like sustain pedal that we discussed in Chapter Nine) that are plugged into the "FC1" and "FC2" input jacks on the back of your machine. **(see figure 10-18)**

They should normally be kept at the fully-back position when not in use, ensuring that they are not routing any modulation signal.

We will be concentrating here on the "Pmod" and "Amod" parameters within each of these displays, since these are the routing assignments specifically for LFO signal ("EGbias" has to do with another, non-LFO type of modulation which will be discussed in great detail in Chapter Eleven). "Pmod", as you may have guessed, determines how much LFO signal gets routed through a particular controller for pitch modulation, and "Amod" does the same for amplitude modulation. The range of both of these - for all controllers - is 0 to 99, with 0 indicating that no LFO signal is being routed to the specific (pitch or amplifier) destination, and 99 indicating that all of the LFO signal is being routed through that particular controller. Unlike many other DX7II controls, these two controls are *linear* and not *exponential*, meaning that a value of "50" literally does mean that *half* of the LFO signal is being routed. Also, when you initialize, ALL ranges for ALL routings for ALL controllers will default to 0 - meaning that NONE of the real-time controllers are routing any kind of modulation signal anywhere.

Let's run an exercise now to set up a *controlled* pitch modulation:

Exercise 56
Controlled pitch modulation

1) Initialize your DX7II from single voice play mode, leave it in algorithm #1, TURN OFF operators 3 through 6 ("110000"), and, using the system of operators 1 and 2, GENERATE a sawtooth wave.

2) Press edit switch 12 and observe that the Pmd parameter is currently at its default of 0. This means that *no* LFO signal is currently being sent to the pitch inputs of the six operators. Press edit switch 11 and observe that the Pms parameter is currently at its default value of 3. This means that all six operators are slightly sensitive to receiving LFO signal at their pitch data inputs, even though no LFO data is currently being routed. Therefore, if we can find a way of sending that data, they *will* react to some degree. We access this alternate, controlled routing, by using edit switches 25 and 26, depending on which controller(s) we wish to use.

3) Let's start by using the mod wheel. Press edit switch 25 repeatedly if necessary in order to call up the Modulation wheel LCD display. **(see figure 10-19)** Observe that all three routing parameters are currently at their initialization defaults of 0. Position the cursor over the "Pmod" parameter and use the data entry slider to enter a value of 99. This means that all of the LFO signal (which currently consists of a default triangle wave traveling at a default speed of 35) is being routed through the modulation wheel to the pitch data inputs of all six operators.

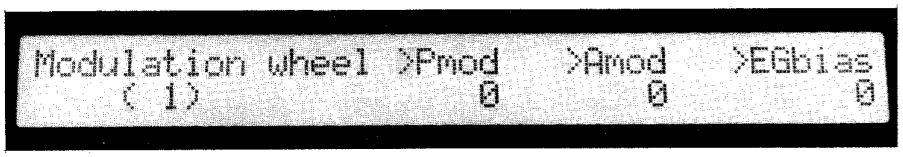


Figure 10-19

4) Begin with the mod wheel all the way at MIN (towards you). Play a note on the keyboard, hold it down, and slowly move the wheel up towards MAX as you listen (Audio Cue 56A).Note that as you raise the wheel up towards its MAX setting (furthest from you), the *depth* of the vibrato pitch modulation increases, just as it did when increasing the Pmd setting in the last Exercise. This is because, just as in the last Exercise, we are routing LFO signal to the pitch inputs of our operators, but this time that signal is traveling through the modulation wheel instead of directly getting there. Therefore, with the wheel at its MIN setting, it is as if we are routing no LFO signal in the first place. You can think of the real-time controller as acting like a valve, which can be fully closed (MIN), fully open (MAX), or anywhere in-between. The maximum allowable signal which is routed at any one time by our controller is determined by the value entered into the routing parameter - in this case, 100% of the total (since we entered "99" into the "Pmod" parameter).

5) The actions of each of the real-time controllers when routing LFO signal (either for pitch modulation or amplitude modulation) is absolutely identical - *there is nothing one controller can do that any other controller can't do*. With this in mind, slowly restore the "Pmod" value for the modulation wheel back to 0 while keeping the wheel at its MAX position. Note that the depth of the vibrato pitch change decreases - even though the wheel is fully up. Once you have reached the "Pmod" value of 0, there will be no further pitch change.

6) Put the wheel back at its MIN position. Now press edit switch 25 two more times in order to call up the After touch LCD display. **(see figure 10-20)** Position the cursor over the "Pmod" parameter and use the data entry slider in order to change this to a new value of 99. Play a note very lightly on the keyboard and listen (Audio Cue 56B). You may hear no pitch modulation at all, or you may hear just a small amount of "bleed-through" modulation.* Now press down on the same key, exert a steadily increasing pressure, and listen (Audio Cue 56C). Note that the depth of the pitch modulation increases in the same way as previously raising the modulation wheel to its MAX position did.

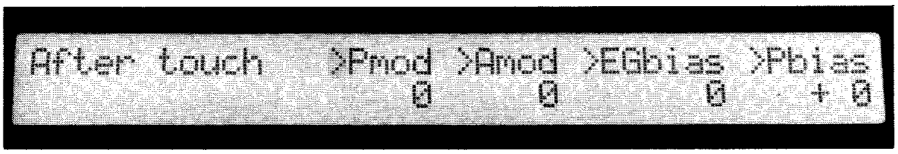


Figure 10-20

7) Play the highest note on the keyboard (C6) as lightly as you can. Continue holding this note down, and play the lowest note on the keyboard (C1) with as much pressure as you can (if your last name is Schwarzenegger, or even if it isn't - don't go crazy! These keys can and do break off if abused! You have been warned!!). Listen (Audio Cue 56D). Note that the high note begins undergoing the same vibrato pitch change as the low one. This is because After touch is a monophonic control. Experiment and note that you can control the modulation depth for a group of notes by simply exerting increasing and decreasing pressure on a single note within that group.

* This so-called "bleed-through" signal occurs simply as a result of some of the hardware circuitry in the instrument. It's nothing to worry about and, frankly, something to expect.

8) The effects of the real-time controllers are *cumulative* - if you decide to use more than one controller for a signal routing (something which admittedly is not commonly done), then the effects of one controller will add on, or accumulate, onto the effects of any other. Change the "Pmod" value for the After touch to 50. Press edit switch 25 again in order to call up the Modulation wheel display and change the "Pmod" for the mod wheel to 75. We now have *both* the keyboard after touch *and* the modulation wheel routing LFO signal to our operators' pitch inputs. Play a note on the keyboard with maximum finger pressure and listen (Audio Cue 56E). Now slowly raise the modulation wheel and listen (Audio Cue 56F). Note that the depth of the modulation increases still further as the wheel routes yet more LFO signal. These cumulative signals can in no event increase beyond the maximum of 99. Even though in this instance, the action of the two controllers appears to be greater than 99 (50 + 75 = 125, not 99), the DX7II "tops out" at its maximum. This is roughly equivalent to the action of an EG Level, which in no event can increase an operator's output level beyond its nominal setting (as determined by the Level parameter in edit switch 10).

9) Put the mod wheel back to its MIN position. Call up the After touch display and restore the "Pmod" value to 0. Call up the Modulation wheel display and restore the "Pmod" value for the wheel to 99.

10) Press edit switch 11 (SENSITIVITY), position the cursor over the Pms parameter, and use the data entry slider to change this to a new value of 0. Place the modulation wheel at its MAX (furthest from you) position, play a note and listen (Audio Cue 56G). Note that we now hear no pitch modulation at all. This is because even though we *are* routing LFO signal via the wheel to the pitch inputs of our operators, they have all been made *insensitive* (Pms=0) to the LFO's commands. **(see figure 10-21)**

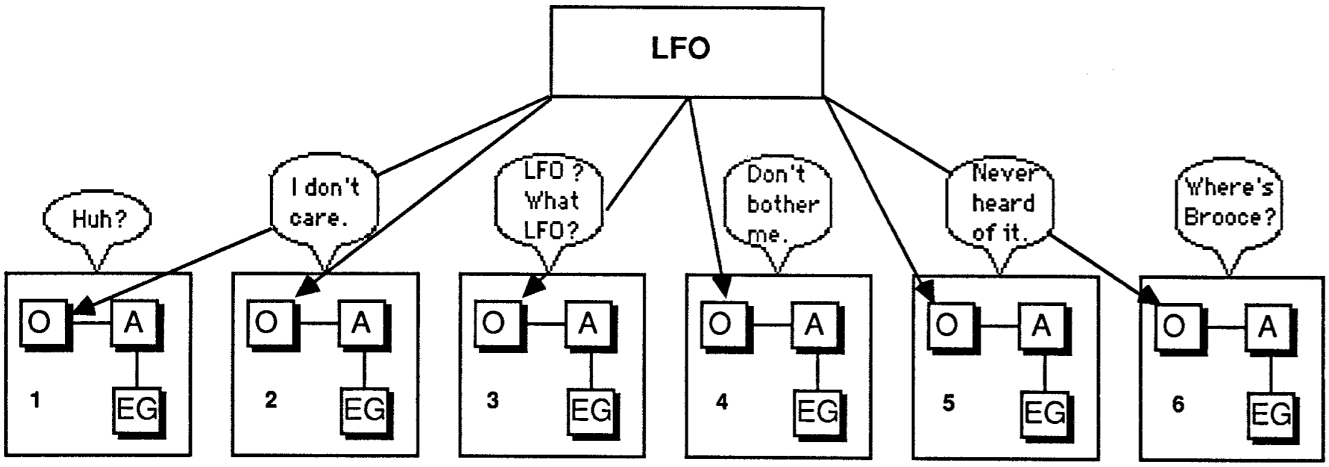


Figure 10-21

11) Use the "yes" button to slowly change the Pms value up to its maximum of 7, holding down a note and listening as you do so (Audio Cue 56H). note that as the Pms value increases, so too does the *depth* of the pitch modulation.

12) Familiarize yourself with different controlled pitch modulation routings. Press edit switch 12 and Experiment with different wave shapes and LFO speeds. Use different pitch modulation sensitivities and different controller routings to determine the maximum allowable pitch modulation depth in every instance. If you have a Breath controller or one or two Foot controllers, Experiment with using them to route LFO signal (note that if you are working with the Foot controllers, you'll have to use edit switch 26 in place of edit switch 25) and note that they affect the sound in precisely the same way as the Modulation wheel and After touch do.

When you are setting up controlled modulation routings (as opposed to direct modulation routings), the LFO Delay parameter has no effect whatsoever. The whole idea behind controlled modulation is that you yourself will, in real time, determine when, and to what degree, the LFO signal affects the sound. For that reason, the Delay parameter is only ever active for direct modulations.

Pitch modulation, then, can be used for a wide variety of effects: vibrato (with a sine or triangle LFO wave), trill (with a square LFO wave), or even random arpeggiation (with a s/hold LFO wave). Very fast LFO speeds will yield periodic changes which are so fast they can barely be perceived as individual changes and will instead sound like a growl. The Pms control, in conjunction with Pmd or the controller "Pmod" parameter, allow us to set up everything from a barely noticeable, subtle wavering, to a severe swooping in pitch. Pms, with a range of only 0 to 7, is more of a "coarse" control, since it determines in a broad sense how the six operators will respond to LFO signal - whereas Pmd and the "Pmod" parameters allow you to set precisely how much signal arrives, with a finer range of 0 to 99.

Let's turn now to the use of the LFO for *amplitude modulation*. As with pitch modulation, we again have the option of either direct or controlled routings, and they are set up pretty much the same way. The "Amd" parameter is used for the direct routing of LFO signal to the operator amplifiers. The real-time controllers can also send LFO signal to the operator amplifiers with the use of their "Amod" parameters. In either circumstance, a certain degree of Amplitude modulation sensitivity ("Ams") must be set, but here, unlike the Pms parameter, we have individual operator controls - we can specify *which* operator or operators will react to this LFO signal and which will not. Just like the Pms control, the Ams control has a range of 0 to 7, with 0 representing no sensitivity and 7 representing maximum sensitivity - but this time we set the sensitivity independently for each of the six operators.

As mentioned previously, because amplitude modulation sensitivity *is* operator-specific, we can use the LFO to either periodically change the *volume* of a sound (if applied to a carrier) and/or the *timbre* of a sound (if applied to a modulator). This also gives us the freedom to apply amplitude modulation to specific *systems* within an overall sound. Let's begin with an Exercise to try out some of the effects of amplitude modulation on a carrier:

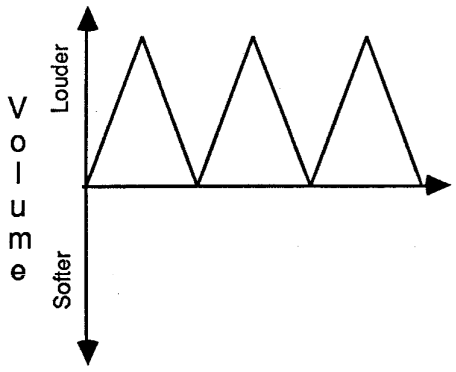


Figure 10-22

Exercise 57
Direct LFO amplitude modulation applied to a carrier

1) Initialize your DX7II from single voice play mode, leave it in algorithm #1, TURN OFF operators 3 through 6 ("110000"), and use the system of operators 1 and 2 to GENERATE a sawtooth wave.

2) Press edit switch 12 in order to VIEW the LFO parameters and observe that Amd (direct amplitude modulation depth) is currently at its default value of 0. Press edit switch 11 (SENSITIVITY) and use edit switches 1 and 2 in order to observe that there is currently no sensitivity for amplitude modulation for either of our two operators (since the initialization default is Ams = 0 for all operators).

3) Play a note on the keyboard and listen (Audio Cue 57A). Note that there is currently no periodic change occurring to either the volume or timbre of the sawtooth wave we are hearing. This is because no LFO modulation is being routed as yet to the amplifiers of either operators 1 or 2 (since Amd = 0) and that in any event neither are sensitive to such a modulation (since Ams for both operators = 0).

4) Press edit switch 11, followed by edit switch 1, and use the data entry slider to change the Ams value for operator 1 only to its maximum value of 7. Make sure that Ams for operator 2 remains at its default value of 0. Press edit switch 12 and position the cursor over the Amd value. Use the data entry slider to change this to its maximum value of 99. Play a note on the keyboard and listen (Audio Cue 57B). Note that you are now hearing a rapid change to the *volume* of the sound.

5) Press the left cursor switch four times in order to position the cursor over the Speed parameter and use the data entry slider to change this to a new value of 8. Play a note and listen (Audio Cue 57C). Note that the speed of the volume change has now slowed appreciably and that you can now clearly hear the volume gradually rising and falling, equivalent to the current Triangle LFO waveshape. **(see figure 10-22)** This has precisely the same effect as slowly moving the DX7II's volume slider up and back. At a higher LFO speed value of 40 or so, this will cause a typical *tremolo* effect. Tremolo is a steady back-and-forth change in the volume of a sound. Try it! (Audio Cue 57D).

6) Restore the LFO speed back to a value of 8. Press the left cursor switch once in order to position the cursor over the Wave parameter, and press the "yes" button once to hear the effect of a saw down LFO wave (Audio Cue 57E). note that the volume now instantly rises and slowly drops, as equivalent to the wave shape. **(see figure 10-23)**

7) Press the "yes" button once again to hear the effect of a saw up LFO wave (Audio Cue 57F). Note that the volume now changes in the opposite manner, slowly rising and quickly dropping, as equivalent to the wave shape. **(see figure 10-24)**

8) Press the "yes" button once again to hear the effect of a square LFO wave (Audio Cue 57G). Note that the volume now instantly rises, followed by an instant fall, as equivalent to the wave shape. **(see figure 10-25)** Note that this produces an on-off effect, useful in having sounds that repeat themselves periodically (yes, we were able to do this with the EG, but we were limited then to one repeat, maximum. See Exercise 49 in Chapter Nine if you don't remember how we did this).

9) Press the "yes" button once again to hear the effect of a sine LFO wave (Audio Cue 57H). Note that the volume now gently rises and falls, in a manner equivalent to the wave shape: **(see figure 10-26)** and similar to the effect of the triangle wave, but more gently, since the sine wave is a more rounded shape. The sine wave is in fact probably a better choice for most standard tremolo effects.

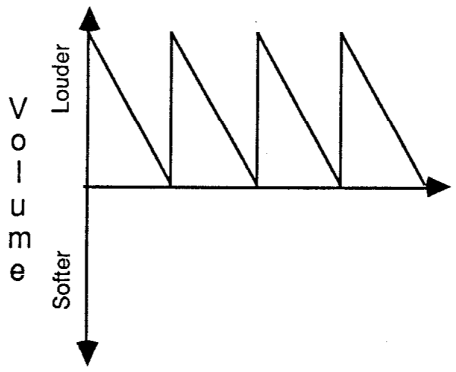


Figure 10-23

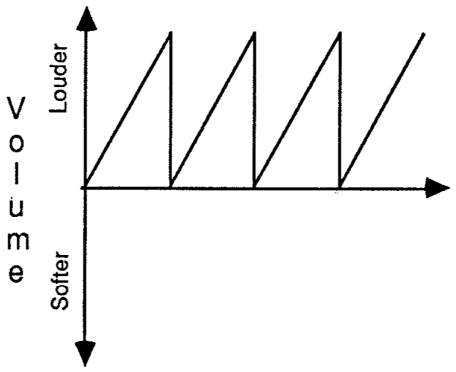


Figure 10-24

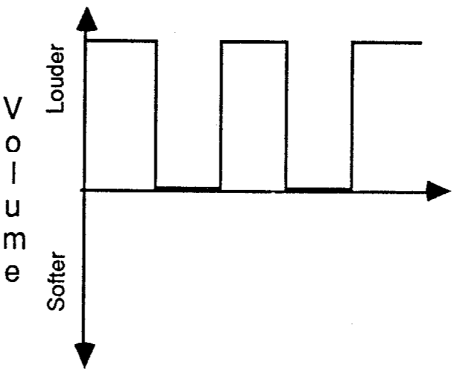


Figure 10-25

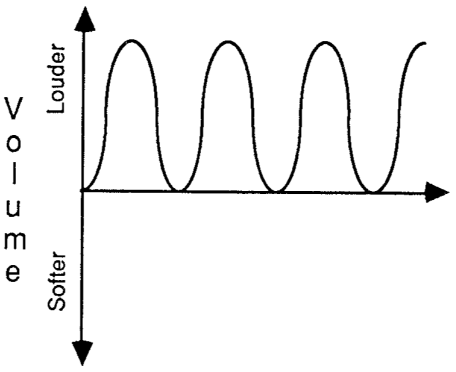


Figure 10-26

10) Press the "yes" button one last time to hear the effect of a S/hold LFO wave (Audio Cue 57I). Note that you now hear a random volume change, as the DX7II is now periodically *sampling* from a stream of random numbers, and *holding* the number just long enough to send the data to the amplifiers of the six operators.

11) Restore the LFO shape to that of a triangle wave. Press the right cursor switch once in order to position the cursor over the Speed parameter, and, using the data entry slider, slowly change this value up to its maximum of 99, holding down a key and listening as you do so (Audio Cue 57J). Note how this change in speed does *not* in any way alter the *type* of change you hear, nor the *amount* of change. Note also that this change in LFO speed is recognized by the DX7II in *real time.* Now use the data entry slider to slowly change this value back to its minimum of 0, holding down a key and listening as you do so (Audio Cue 57K). Note that the speed of the volume change now decreases, and that at the absolute minimum speed of 0, it occurs only about once every ten seconds (indicating an LFO frequency of approximately 0.10 Hz).

12) Restore the LFO speed back to a value of 8 and press the right cursor switch four times in order to position the cursor over the Amd parameter. Use the data entry slider to slowly change this value back to the minimum of 0, holding down a key and listening as you do so (Audio Cue 57L). Note that this has the effect of lessening the *depth* of the volume change but in no way alters the shape of the change nor the speed.

13) Restore the Amd value back to 99 and press edit switch 11, followed by edit switch 1, in order to VIEW the Ams parameter for operator 1. Position the cursor over this parameter and use the "no" button to slowly change this value for operator 1 to its minimum of 0, holding down a key and listening as you do so (Audio Cue 57M). Note that this has the same effect on the sound that lessening the Amd did, but because of its coarser range, it is much less subtle, and therefore less useful as a modulation depth control.

14) Restore the Ams parameter for operator 1 back to a value of 7 and press edit switch 12 again. Position the cursor over the Delay parameter and use the data entry slider to change this to a value of 50. Play a note on the keyboard and listen (Audio Cue 57N). Note that the effect of the LFO on the volume is now delayed by about a second, but that the ultimate volume change is precisely the same as before. Note also that this delay is only initiated for new staccato notes, and not for new notes added to held ones or to held chords. Change the Delay value to its maximum of 99, play a note and listen (Audio Cue 57O). Note that the onset of the LFO is now delayed by some 3 seconds.

15) Experiment by setting various different LFO waveshapes with various speeds and delay times. Also Experiment with different Amd and Ams values for operator 1 and note how changing these different controls affects the final sound. Try creating a second, different timbre within algorithm #1 by using the system of operators 3 and 4. Then apply an LFO amplitude modulation to operator 3 only. Note that we now have only one of the two timbres in our total sound (the operator 3-4 system) periodically changing in volume, while the other timbre (the operator 1-2 system) remains unchanged. **(see figure 10-27)**

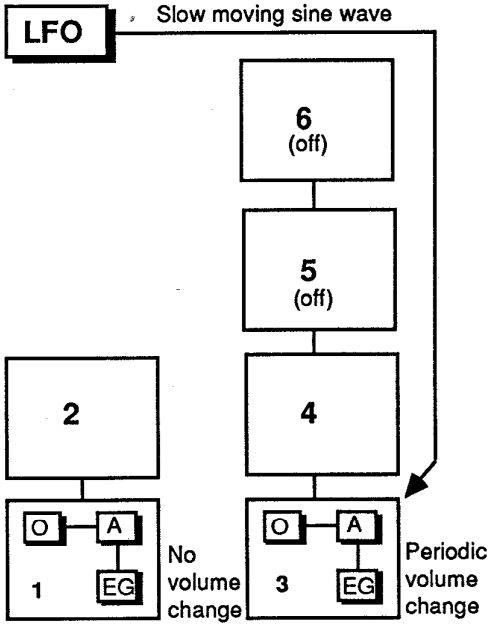


Figure 10-27

Of course, we can also accomplish the same exact modulation effects with our real-time controllers:

Exercise 58
Controlled LFO amplitude modulation applied to a carrier

1) Initialize your DX7II from single voice play mode, leave it in algorithm #1, TURN OFF operators 3 through 6 ("110000"), and, using the system of operators 1 and 2, GENERATE a sawtooth wave.

2) Press edit switch 11, followed by edit switch 1 in order to VIEW the sensitivity values. Position the cursor over the Ams parameter and enter a new value of 7 for operator 1 only. Make sure that Ams for operator 2 remains at its current default value of 0. This means that our carrier (operator 1) is now sensitive to amplitude modulation, even though no data is currently being sent to it. If we therefore route that data via one of our real-time controllers, operator 1 will react as much as possible.

3) Press edit switch 25 repeatedly if necessary in order to call up the Modulation wheel display. Position the cursor over the "Amod" parameter and use the data entry slider to enter a value of 99. This means that *all* of the LFO signal (which currently consists of a default triangle wave traveling at a default speed of 35) is now being routed through the modulation wheel. We have now instructed our DX7II to send the LFO signal to the *amplifiers* of the six operators and *not* to their pitch data inputs. **(see figure 10-28)**

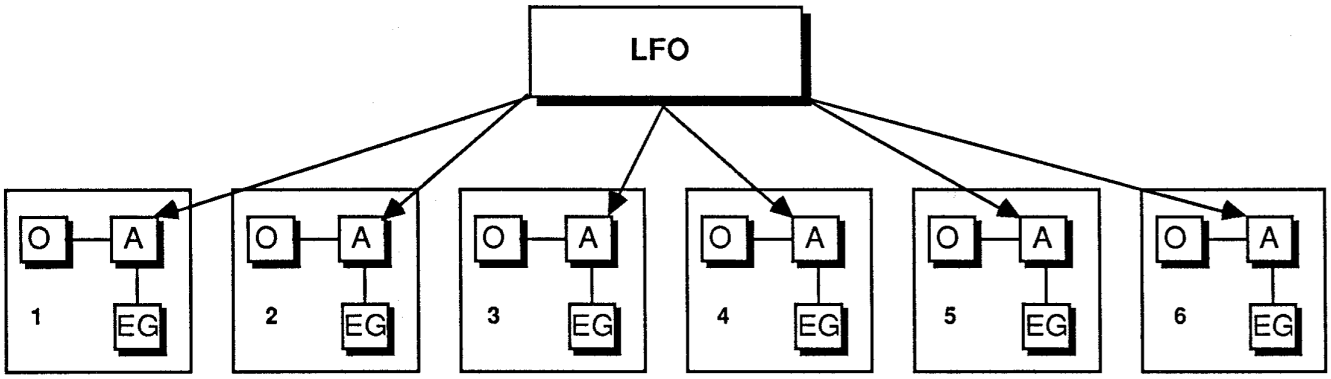


Figure 10-28

4) Begin with the modulation wheel all the way at MIN (towards you). Play a note on the keyboard, hold it down, and slowly move the wheel up towards MAX as you listen (Audio Cue 58A). Note that as you raise the wheel up towards its MAX setting (furthest from you), the *depth* of the tremolo amplitude modulation increases, just as it did when increasing the Amd setting in the last Exercise. This is because, just as in the last Exercise, we are still routing LFO signal to the amplifiers of our operators, but this time, that signal is traveling via the modulation wheel instead of directly getting there. Because currently only operator 1 is *sensitive* to amplitude modulation, we are currently only hearing a periodic volume change.

5) Return the mod wheel to its MIN position. Restore the "Amod" value for the mod wheel back to 0. Now press edit switch 25 twice more in order to call up the After touch LCD display. Position the cursor over the "Amod" parameter and change this to a new value of 99. Play a note very lightly on the keyboard and listen (Audio Cue 58B). You may hear no amplitude modulation at all, or you may hear a small amount of "bleed-through" modulation. Now press down on the same key, exert a steadily increasing pressure, and listen (Audio Cue 58C). Note that this causes the depth of the tremolo volume change to increase in the same way that previously raising the modulation wheel to MAX did.

6) Play the highest note on the keyboard (C6) as lightly as you can. Continue holding this note down, and play the lowest note on the keyboard (C1) with as much reasonable pressure as you can and listen (Audio Cue 58D). Note that the high note begins undergoing the same tremolo volume change as the low one. This is because After touch is a monophonic control. Experiment and note that you can control the modulation depth for a group of notes by simply exerting increasing and decreasing pressure on a single note within that group.

7) As with pitch modulation routings, the effects of the real-time controllers are cumulative. Change the After touch "Amod" value to 50. Press edit switch 25 once more to call up the Modulation wheel display and change the "Amod" for the mod wheel to 75. We now have *both* the After touch *and* the mod wheel routing LFO signal to our operators' amplifiers. Play a note on the keyboard with maximum finger pressure and listen (Audio Cue 58E). Now slowly raise the mod wheel and listen (Audio Cue 58F). Note that the depth of the modulated sound increases still further as the mod wheel routes yet more LFO signal. Note that these cumulative signals will in no event increase beyond the maximum of 99.

8) Restore the Modulation wheel "Amod" to 99 and restore the After touch "Amod" to 0. Press edit switch 11 and use the data entry slider to change the Ams for operator 1 only to a new value of 0. Place the mod wheel at its MAX (furthest from you) position, play a note and listen (Audio Cue 58G). Note that we now hear no amplitude modulation at all.

9) Use the "yes" button to slowly change the Ams value for operator 1 up to its maximum of 7, holding down a note and listening as you do so (Audio Cue 58H). Note that as the Ams sensitivity increases, so too does the depth of the amplitude modulation.

10) Familiarize yourself with different controlled amplitude modulation routings to carriers. Press edit switch 12 and Experiment with different wave shapes and LFO speeds. Experiment further with different Ams sensitivities for operator 1 and different controller "Amod" values to determine the maximum allowable amplitude modulation depth in every instance. If you have a Breath controller and/or one or more Foot controllers, Experiment with using them for routing LFO signal (using edit switch 26) and note that they affect the sound in precisely the same way as the mod wheel and After touch do. Get a good feel for the way the parameters in edit switches 11, 12, 25 and 26 interact. Try creating, as before, a second timbre within algorithm #1 with the system of operators 3 and 4, and apply controlled amplitude modulation to operator 3 only. note that once again, one timbre within the sound undergoes a periodic volume change while the other remains steady.

It is the operator-specificity of amplitude modulation in the DX7II that makes it such a powerful tool. With that in mind, let's try once again setting up a *direct* amplitude modulation, but this time we'll be affecting our modulator, resulting in periodic quantitative *timbral* changes:

Exercise 59
Direct LFO amplitude modulation applied to a modulator

1) Initialize your DX7II from single voice play mode, leave it in algorithm #1, TURN OFF operators 3 through 6 ("110000"), and use the system of operators 1 and 2 to GENERATE a sawtooth wave.

2) Press edit switch 12 and observe that Amd (direct amplitude modulation depth) is currently at its default value of 0. Press edit switch 11, followed by edit switches 1 and 2 in order to observe that there is currently no sensitivity for amplitude modulation for either of our two operators (since the initialization default is Ams = 0 for all operators).

3) Play a note on the keyboard and listen (Audio Cue 59A). Note that there is currently no periodic change occurring to either the amplitude or timbre of the sawtooth wave we are hearing. This is because no LFO modulation is being routed as yet to the amplifiers of either operators 1 or 2 (since Amd = 0) and that in any event neither are sensitive to such a modulation (since Ams = 0).

4) Use the data entry slider to change the Ams value for operator 2 only to its maximum value of 7. Make sure that Ams for operator 1 remains at its default value of 0. Press edit switch 12 and use the data entry slider to change the Amd to its maximum value of 99. Play a note on the keyboard and listen (Audio Cue 59B). Note that you are now hearing a rapid change to the *timbre* of the sound.

5) Use the data entry slider to change the LFO Speed value to 8. Play a note and listen (Audio Cue 59C). Note that the speed of the timbral change has now slowed appreciably and that you can now clearly hear the overtones coming in and out, as the LFO's triangle wave gradually increases and then decreases the *output level* of operator 2 (because it is controlling operator 2's amplifier). **(see figure 10-29)** At a higher LFO speed value of 40 or so this will cause a typical wah-wah effect. Try it! (Audio Cue 59D)

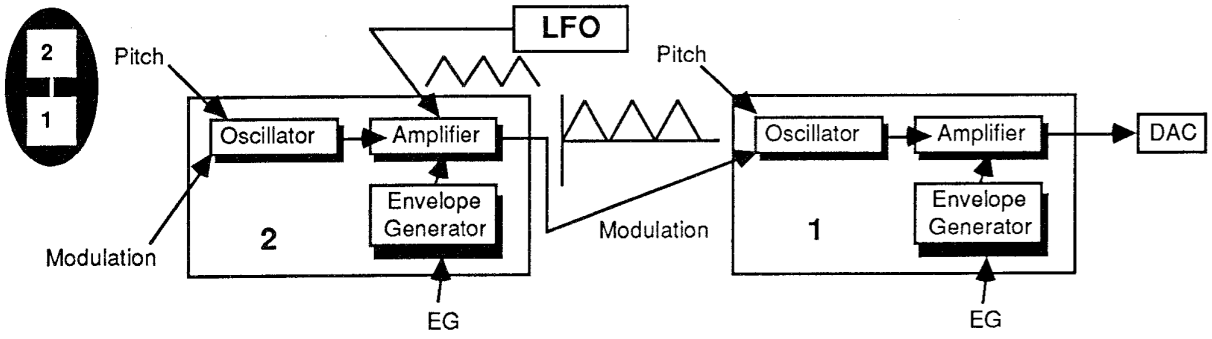


Figure 10-29

6) Restore the LFO speed back to a value of 8. Position the cursor over the Wave parameter and press the "yes" button once to hear the effect of a saw down LFO wave (Audio Cue 59E). Note that the overtone content now instantly rises and slowly drops, as equivalent to the wave shape. **(see figure 10-30)**

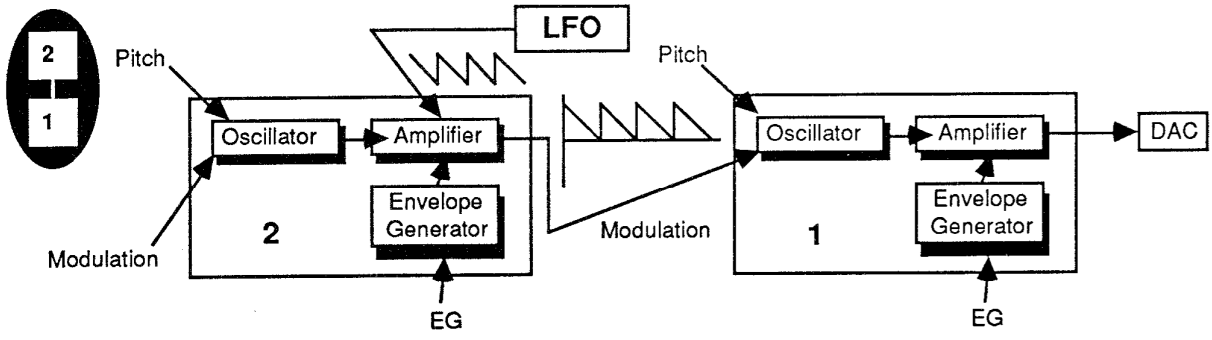


Figure 10-30

7) Press the "yes" button once again to hear the effect of a saw up LFO wave (Audio Cue 59F). Note that the timbre now changes in the opposite manner, with the overtone content slowly increasing and quickly decreasing, as equivalent to the wave shape. **(see figure 10-31)**

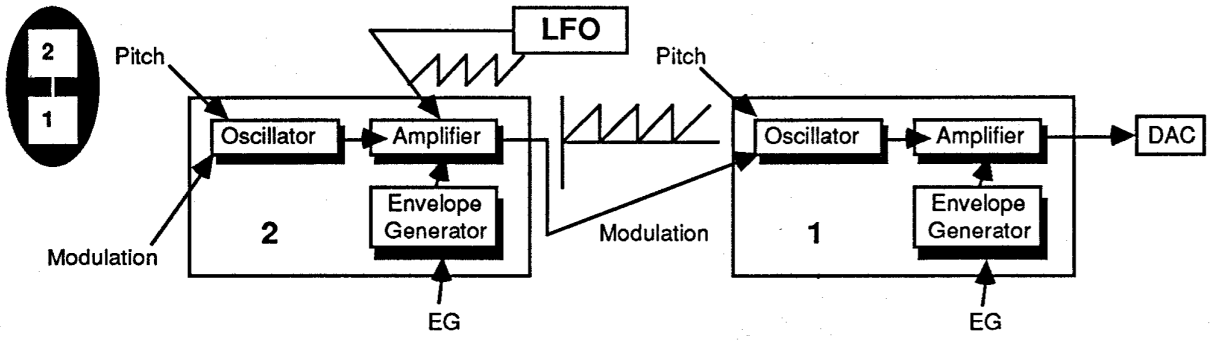


Figure 10-31

8) Press the "yes" button once again to hear the effect of a square LFO wave (Audio Cue 59G). Note that we now hear *two* different timbres - a sawtooth wave when the LFO square wave is positive **(see figure 10-32)** (and a sine wave when the LFO square wave is negative. **(see figure 10-33)**

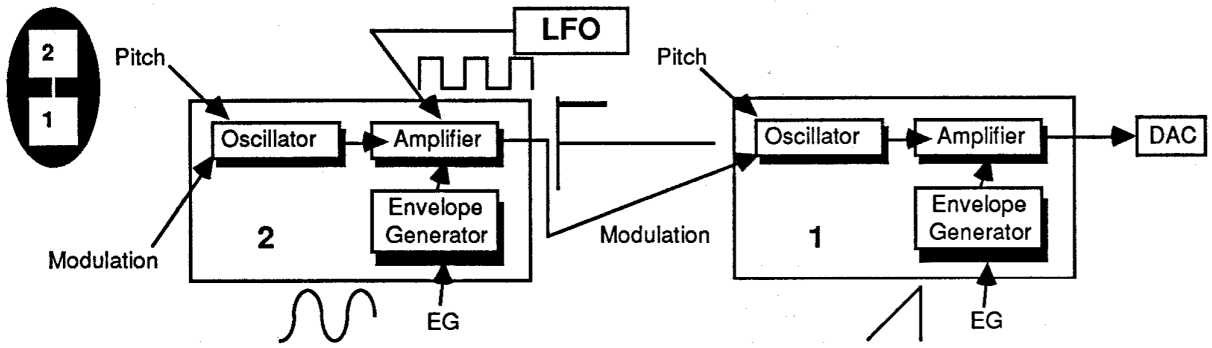


Figure 10-32

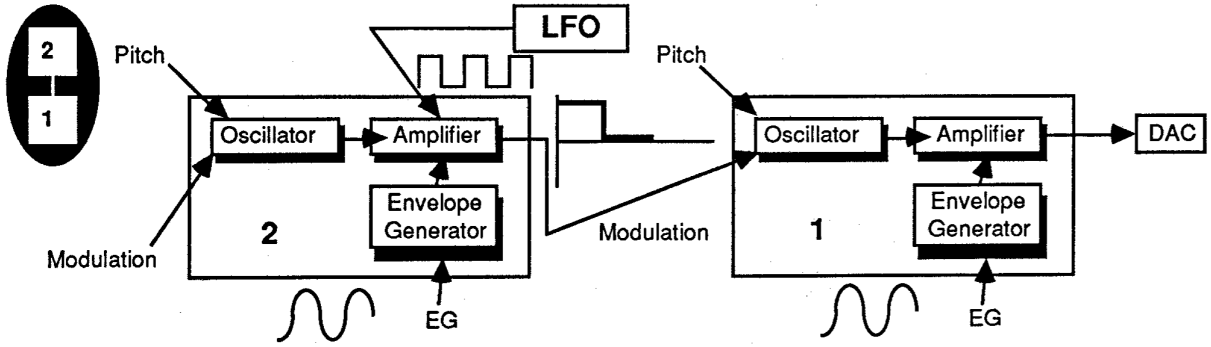


Figure 10-33

9) Press the "yes" button once again to hear the effect of a sine LFO wave (Audio Cue 59H). Note that the overtone content now gently rises and falls, producing a gentle repeating "wah-wah", in a manner equivalent to the wave shape. **(see figure 10-34)** and similar to the effect of the triangle wave, but more gently, since the sine wave is a more rounded shape.

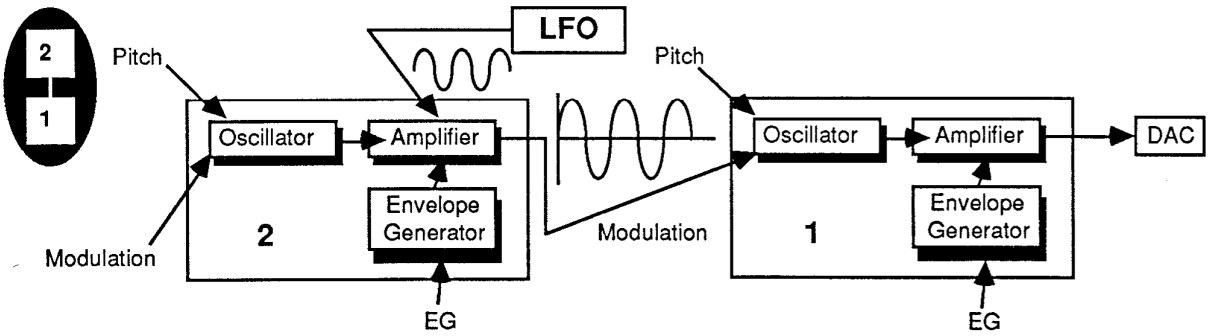


Figure 10-34

10) Press the "yes" button one last time to hear the effect of a S/hold LFO wave (Audio Cue 59I). This is probably the most interesting and typical use of the S/hold waveshape, since you now hear a *random* timbral change. Increase the LFO Speed value to about 40 or so to hear a pleasing, rhythmic random change in timbre (Audio Cue 59J).

11) Restore the LFO shape to that of a triangle wave. Position the cursor over the Speed parameter and, using the data entry slider, slowly change this value up to its maximum of 99, holding down a key and listening as you do so (Audio Cue 59K). At the highest LFO speeds you will start to hear some dissonant overtones (called *sidebands*) creeping in. This is because the LFO is now in an audible (greater than 20 Hz) range. Note, however, that these changes in speed do not in any way alter the type of change you hear, nor the amount of change. Note also that changes in LFO speed are recognized by the DX7II in real time. Now use the data entry slider to slowly change the Speed back to its minimum of 0, holding down a key and listening as you do so (Audio Cue 59L). Note that the speed of the timbral change now decreases, and that at the absolute minimum speed of 0, it occurs only about once every ten seconds. Note the quantization effects of the LFO, particularly apparent at the slowest Speed value of 0.

12) Restore the LFO Speed back to a value of 8 and position the cursor over the Amd parameter. Use the data entry slider to slowly change this value back to the minimum of 0, holding down a key and listening as you do so (Audio Cue 59M). Note that this has the effect of lessening the depth of the timbral change (we hear fewer overtones coming in and out) but in no way alters the shape of the change nor the speed.

13) Restore the Amd value back to 99 and press edit switch 11, followed by edit switch 2. Use the "no" button to slowly change the Ams value for operator 2 to its minimum of 0, holding down a key and listening as you do so (Audio Cue 59N). Note that this has the same effect as lessening the Amd did, but because of its coarser range, it is much less subtle, and so generally less useful as a modulation depth control.

14) Restore the Ams parameter for operator 2 back to a value of 7 and press edit switch 12 again. Position the cursor over the Delay parameter and use the data entry slider to change this to a value of 50. Play a note on the keyboard and listen (Audio Cue 59O). Note that the effect of the LFO on the overtone content is now delayed by about a second, but that the ultimate timbral change is precisely the same as before. Note also that this delay is only initiated for new staccato notes, and not for new notes added to held ones or to held chords.

Change the LFO Delay value to its maximum of 99, play a note and listen (Audio Cue 59P). Note that the onset of the LFO is now delayed by some 3 seconds.

15) Experiment by setting various different LFO waveshapes with various speeds and delay times. Also Experiment with different Amd and Ams values for operator 2 and note how changing these different controls affects the final sound.

16) Because amplitude modulation is operator-specific, we can apply it to *both* our modulator and carrier, in the same or different degrees, if we so desire. Experiment by redoing this entire Exercise with both operators 1 and 2 set to maximum Ams values of 7. Note how both the volume *and* the timbre are affected by the LFO in the same manner. Then Experiment by redoing this Exercise with operator 1 set to an Ams of 7 and operator 2 at an Ams of 3. note how the same LFO manipulations now affect mostly the volume, but with slight amounts of timbral change as well. Finally, Experiment by reversing the sensitivities (operator 1 = a value of 3 and operator 2 = a value of 7). Note that we now generate the opposite effect: a great deal of timbral change and only a small amount of volume change.

Again, whatever kinds of modulation effects we can set up directly, we can also route via our real-time controllers:

Exercise 60:
Controlled LFO amplitude modulation applied to a modulator

1) Initialize your DX7II from single voice play mode, leave it in algorithm #1, TURN OFF operators 3 through 6 ("110000"), and, using the system of operators 1 and 2, GENERATE a sawtooth wave.

2) Press edit switch 11, followed by edit switch 2, and change the Ams for operator 2 only to the maximum value of 7. Make sure that the Ams for operator 1 remains at its current default value of 0. This means that our modulator (operator 2) is now sensitive to amplitude modulation, even though no LFO signal is currently being sent to it. If we therefore route that signal via one of our real-time controllers, operator 2 will react as much as possible, giving us a quantitative change in the timbre of the sound.

3) Press edit switch 25 repeatedly if necessary in order to call up the Modulation wheel display and position the cursor over the "Amod" parameter. Use the data entry slider in order to enter a value of 99. This means that all of the LFO signal (which currently consists of a default triangle wave traveling at a default speed of 35) is now being routed through the modulation wheel to the amplifiers of our six operators. However, *only* operator 2 in this instance will react.

4) Begin with the modulation wheel all the way at MIN (towards you). Play a note on the keyboard, hold it down, and slowly move the wheel up towards MAX as you listen (Audio Cue 60A). Note that as you raise the wheel up towards its MAX setting (furthest from you), the depth of the amplitude modulation increases, just as it did when increasing the Amd setting in the last Exercise. This is because, just as in the last Exercise, we are routing LFO signal to the amplifiers of our operators, but this time that signal is traveling via the Modulation Wheel instead of directly getting there. Because currently only operator 2 is sensitive to amplitude modulation, we are currently only hearing a periodic *timbral* change.

5) Return the mod wheel to its MIN position. Restore the "Amod" value back to 0 and press edit switch 25 twice more in order to call up the After touch display. Change the After touch "Amod" to the

maximum value of 99. Play a note very lightly on the keyboard and listen (Audio Cue 60B). You may hear no amplitude modulation at all, or you may hear a small amount of "bleed-through" modulation. Now press down on the same key, exert a steadily increasing pressure, and listen (Audio Cue 60C). Note that the depth of the timbral change increases in the same way as previously raising the mod wheel did.

6) Play the highest note on the keyboard (C6) as lightly as you can. Continue holding this note down, and play the lowest note on the keyboard (C1) with as much reasonable pressure as you can and listen (Audio Cue 60D). Note that the high note begins undergoing the same periodic timbral change as the low one. This is because After touch is a monophonic control. Experiment and note that you can control the modulation depth for a group of notes by simply exerting increasing and decreasing pressure on a single note within that group.

7) As with pitch modulation routings, the effects of the real-time controllers are cumulative. Change the After touch "Amod" value to 50. Press edit switch 25 once more in order to call up the Modulation wheel display and change its "Amod" value to 75. We now have *both* the After touch *and* the mod wheel routing LFO signal to our operators' amplifiers. Play a note on the keyboard with maximum finger pressure and listen (Audio Cue 60E). Now slowly raise the mod wheel and listen (Audio Cue 60F). Note that the depth of the modulated sound increases still further as the mod wheel routes yet more LFO signal. Note that these cumulative signals will in no event increase beyond the maximum of 99.

8) Restore the Modulation wheel "Amod" to 99 and restore the After touch "Amod" to 0. Press edit switch 11, followed by edit switch 2, and use the data entry slider to change the Ams for operator 2 only to a new value of 0.

9) Place the mod wheel at its MAX (furthest from you) position, play a note and listen (Audio Cue 60G). Note that we now hear no amplitude modulation at all.

10) Use the "yes" button to slowly change the Ams value for operator 2 up to its maximum of 7, holding down a note and listening as you do so (Audio Cue 60H). Note that as the Ams sensitivity increases, so too does the *depth* of the amplitude modulation.

11) Familiarize yourself with different controlled amplitude modulation routings to modulators. Press edit switch 12 and Experiment with different wave shapes and LFO speeds. Use different Ams sensitivities for operator 2 and different controller "Amod" values to determine the maximum allowable amplitude modulation depth in every instance. If you have a Breath controller and/or one or more Foot Controllers, Experiment with using them to route LFO signal (using edit switch 26) and note that they affect the sound in precisely the same way as the Modulation wheel and After touch do.

Because controller routings of LFO signals are *voice-specific* in the DX7II (as opposed to the original DX7, where they weren't), it is easy to program the same voice into different memory slots with different effect set-ups. It's also very easy to customize the routings for your own particular taste - some DX7II users LOVE foot controllers, for example, while others HATE them. State your preference, and set up your voices (whether original or presets) just the way that you feel comfortable. The important thing to remember is that ALL real-time controllers will have precisely the same effect as one another when routing LFO signal - they really are completely interchangeable.

Suppose, for example, you want to use the "Pluk" voice (ROM bank 1, slot 8) for a particular song at a gig. This preset has been programmed so that raising the modulation wheel will induce a broad, screaming vibrato effect - obviously the LFO is being used for pitch modulation. Suppose you love that effect, but you'd really rather have it brought in with After touch because you know that your left hand will be occupied with power chords on a Ralond Pluto XVI during this song. Modifying this voice couldn't be simpler: simply put your DX7II into edit mode, and press edit switch 25 repeatedly to call up the Modulation wheel LCD display. **(see figure 10-35)**

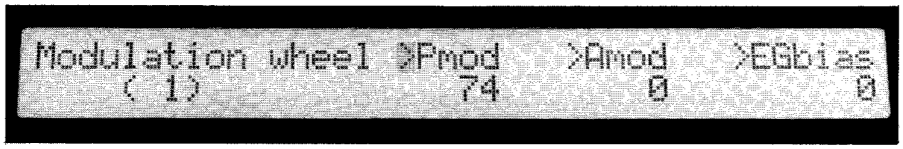


Figure 10-35

Now position the cursor over the "Pmod" parameter and use the data entry slider to change it to 0. You've now disabled the mod wheel for pitch modulation - and the appearance of a decimal point in the LED will confirm that you've done something to change this sound. Now press edit switch 25 two more times to call up the After touch LCD display. Position the cursor over the "Pmod" parameter and enter a new value of, oh, 74. **(see figure 10-36)**

Figure 10-36

The After touch is now routing LFO signal to the pitch inputs of your six operators. A quick look at the display called up by edit switch 11 **(see figure 10-37)** will confirm that there is indeed sensitivity to pitch modulation for all operators (since Pms=3). As noted earlier, even though edit switch 11 *is* operator-specific, you don't have to look at any particular operator, since Pms is a non-operator-specific parameter. That's why the display reads "(all OP)" next to the Pms value.

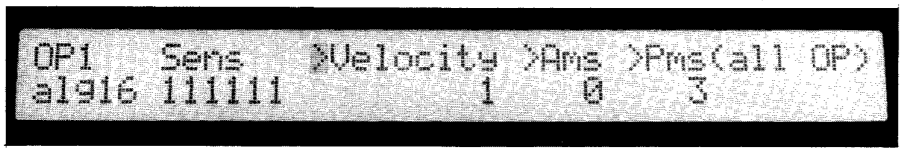


Figure 10-37

More importantly, unless you own every single kind of controller and spend many hours experimenting with all of them, you may not always be aware of what kind of complex LFO routings have been assigned to various controllers - even controllers you may not have hooked up to the machine. The only way you can be sure of what the LFO is meant to be doing in a particular voice and how its signal is being routed is to put your DX7II in edit mode and have a look. The data contained in edit switches 11, 12, 25, and 26 will tell you everything you need to know about LFO routings and modulation sensitivities within that voice.

It would probably be a good idea to devote an evening to going through all the preset sounds you have for your DX7II (ROM presets and voices you may have bought or gotten from friends (either in your internal memory or on RAM cartridge or disk) and examine each of them to see precisely what kind of modulation effects they have been set up for. The factory presets that are supplied on the ROM cartridge are fairly well-labeled when optional controllers like the Breath controller or Foot controllers are meant to be used, but third-party presets may well contain some unusual and unlabeled controller routings. If you put the time in to examine them all, you'll probably be very surprised at the number of expressive sounds you have already: many of them are probably set to deliver vibratos, tremolos, trills, bubbles, random effects, or just plain weirdness - if only you'd use the correct controllers.

LFO Sync

As mentioned earlier, this control allows you to trigger the LFO from the generation of a "key on" flag, thereby *synchronizing* it to the rhythms you play on the keyboard. When the LFO Sync is OFF, the LFO ignores you, me, and the world at large, and simply spits out its slow-moving waves, oblivious to the beauty and wonder of Nature or, for that matter, anything else. **(see figure 10-38)**

When the LFO Sync is ON, however, the LFO will be "listening" for "key on" flags, which, as we learned in Chapter Nine, are generated whenever you play a note on the keyboard. Whenever it perceives a new one, it will instantly return to the beginning of its wave cycle. In short, when the DX7II LFO Sync is ON, the act of you playing a new note (without other notes being held down) will actually cause it to *re-trigger*: **(see figure 10-39)**

This parameter, unlike (and not to be confused with) the Oscillator Sync parameter (see Chapter Three), can actually be very useful. If you are using the LFO for rhythmic effects (as with a square or saw down waveshape, for example), then having the LFO Sync ON will ensure that each new note you play will always restart the LFO wave at the top of its wavecycle. On the other hand, having it OFF will induce a sense of randomness into your playing, since you'll never know precisely where in its wavecycle the LFO will be when you depress a key: it may be at the top of its cycle, at the bottom, or anywhere in-between. Let's run an Exercise now to try it out. For the sake of a little variety, we'll work with a square wave as our audio source. We'll begin by setting up a very obvious volume change with a saw down LFO wave, and we'll see how turning the LFO Sync ON and OFF affects the sound:

LFO Sync off :

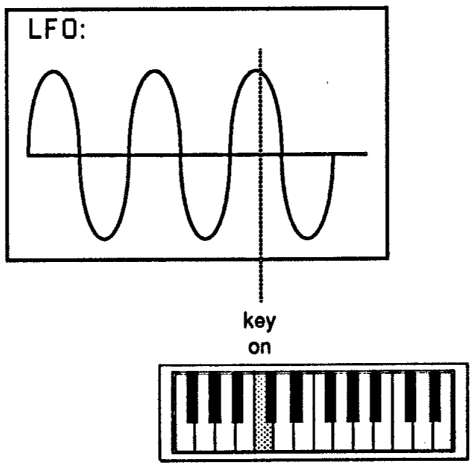


Figure 10-38

LFO Sync on:

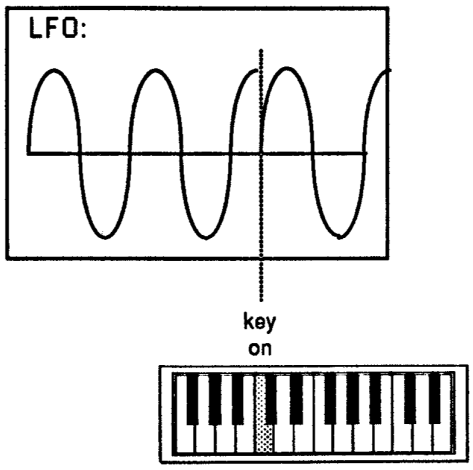


Figure 10-39

Exercise 61
LFO Key Sync

1) Initialize your DX7II from single voice play mode, leave it in algorithm #1, TURN OFF operators 3 through 6 ("110000"), and use the system of operators 1 and 2 to GENERATE a *square* wave.

2) Press edit switch 12 in order to observe the LFO default parameters. Note that the Sync parameter is currently at its default value of ON. Leave this at its default and also make sure you leave the Mode parameter at its default of "single" for this Exercise (we'll discuss the "Mode" parameter shortly).

3) Position the cursor over the Wave parameter and use the data entry slider to change this to a saw down wave. Press the right cursor switch once in order to position the cursor over the Speed parameter and change this to a new value of 11, so the effect can be heard clearly.

4) Press the right cursor switch four more times in order to position the cursor over the Amd parameter, and change this from its default of 0 to its maximum value of 99, thereby setting up a direct amplitude modulation.

5) Press edit switch 11, followed by edit switch 1 in order to VIEW the sensitivity values for operator 1. Position the cursor over the Ams parameter and change this value for operator 1 only from its current default of 0 to the maximum value of 7.

6) Play a single note on the keyboard and listen (Audio Cue 61A). Note that we are hearing a periodic volume change, as only our carrier (operator 1) is reacting to the saw down LFO wave. The shape of this wave is responsible for the full-volume-then-fadeaway effect that we are hearing, as equivalent to the waveshape. **(see figure 10-40)**

7) Play a series of staccato notes on the keyboard. Listen (Audio Cue 61B). Note that each new note *always* starts at full volume - in other words, at the *top* of the LFO saw down wavecycle.

Saw Down Wave:

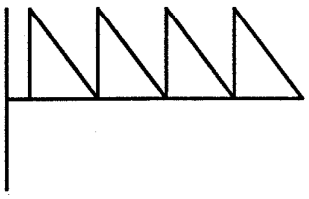


Figure 10-40

8) Now press edit switch 12 and turn the LFO Sync OFF. Play a series of staccato notes at various times and listen (Audio Cue 61C). Note that some notes fall into the beginning of the saw down wave cycle, while others fall at the end, "fade-away" portion, and that there is no way to predict where they will occur. **(see figure 10-41)**

9) Experiment with different LFO waves and speeds and different Amd or Pmd, Ams and Pms values. For each new setting, turn the LFO Sync first ON, then OFF. Note that, regardless of the LFO shape, speed, or type of modulation, the LFO Sync always has the effect of re-triggering the LFO wave at the beginning of its cycle for all new notes that are played.

LFO (Sync off):

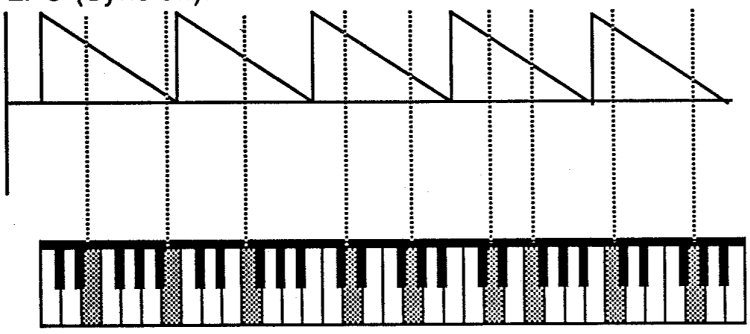


Figure 10-41

Single and Multi Mode

We referred briefly at the beginning of this chapter to the fact that the DX7II actually contains sixteen physical LFOs - though each of these is controlled by a single *virtual* LFO. What we mean by this is that, even though each of the sixteen tone generators has its own individual LFO, the settings of all sixteen must be *the same* and are determined by the values we enter into the single, virtual LFO with the parameters of edit switch 12. **(see figure 10-42)**

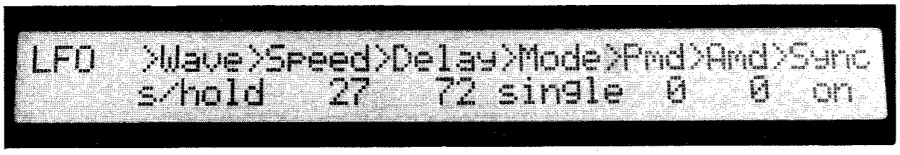


Figure 10-42

The "Mode" parameter, accessed from edit switch 12, will determine whether or not the sixteen actual LFOs - each with the same waveshape, speed, and delay time - output their signals in synchronicity *with one another*. *Single* mode causes them to work as if they are one big LFO, with all of their waves synchronized and *phase-locked* (that is, when one starts a wave cycle, they all do). Their wavecycles may or may not begin upon the reception of a "key on" flag, depending upon whether LFO Sync is OFF or ON - but in Single mode, all sixteen LFO waves will be perfectly in phase with one another. **(see figure 10-43)**

LFO Single Mode:

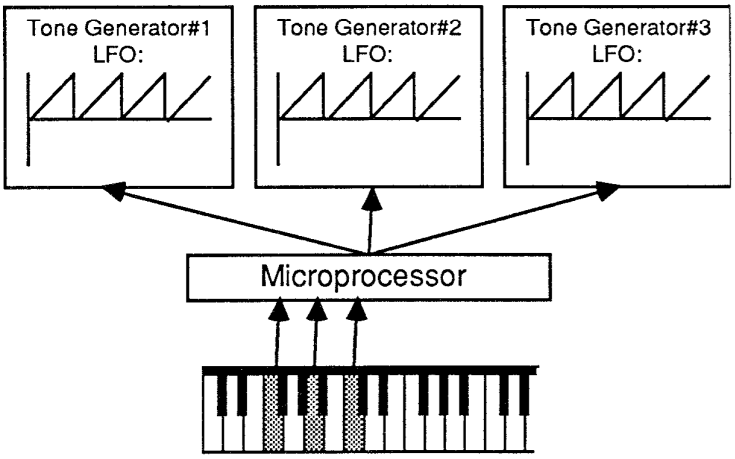


Figure 10-43

In *Multi* mode, the sixteen independent LFOs will act like - well, sixteen independent LFOs. Each key you depress will activate a different LFO - and while, as before, all sixteen will have the same waveshape, speed, and delay time - there will be no synchronization between their wavecycles, and no phase correlation between them (meaning that they may well start at different times) except that individual LFOs will start at the reception of a specific "key on" if LFO sync is ON. **(see figure 10-44)**

LFO Multi Mode:

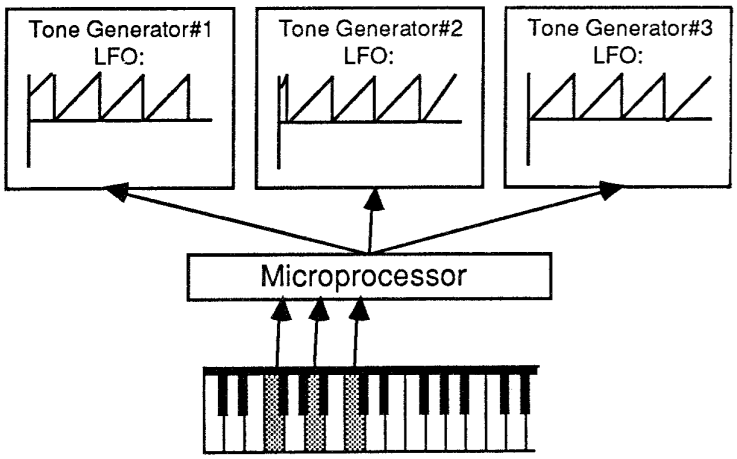


Figure 10-44

As with so many other DX7II controls, this is easier to hear than it is to describe. Let's therefore run an exercise now that lets us clearly hear the effect of Single versus Multi LFO mode:

Exercise 62
LFO Single and Multi Mode

1) Initialize your DX7II from single voice play mode and leave it in algorithm #1. For the sake of still more variety, let's make a pleasing timbre from a 14:1 frequency ratio. Therefore, TURN OFF operators 3 through 6 ("110000") and change operator 2's ratio number to 14.00. Press edit switch 10 and set operator 2's output level at a value of 46.

2) In order to clearly hear the effect of the LFO modes, let's set some "release" time with the EGs. Therefore, change R4 for *both* operators 1 *and* 2 to a new value of 22. Play a note and listen (Audio Cue 62A).

3) At the moment, of course, there is no LFO activity at all. Press edit switch 12 in order to confirm this and observe the default values in the display. **(see figure 10-45)**

Figure 10-45

4) Let's begin with a slow and exaggerated vibrato effect. Position the cursor over the Speed parameter and change this to a new value of 10. Position the cursor over the Pmd and change this to a new value of 67. Position the cursor over the Sync parameter and turn it OFF. Play an arpeggiated series of notes and listen (Audio Cue 62B). We are currently in "single" Mode, so you should be hearing a series of notes moving up and down in pitch (due to the Triangle waveshape), all at the same speed, and in a completely synchronized fashion (that is, ALL the notes go up together and then they ALL come down together). **(see figure 10-46)**

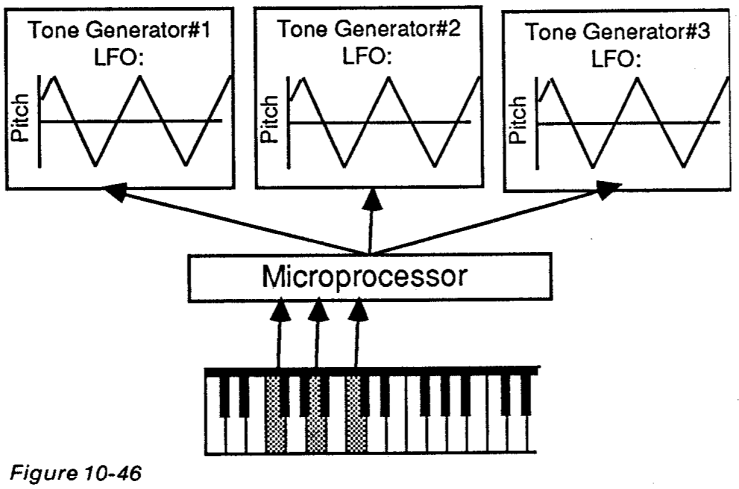
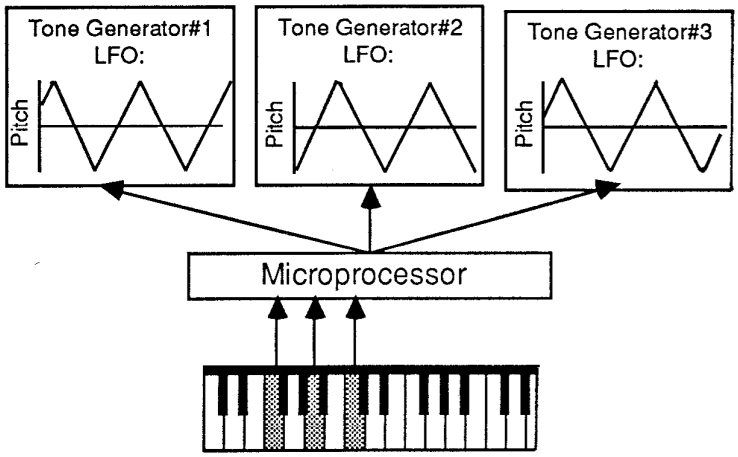


Figure 10-46

(5) Position the cursor over the Mode parameter and change this to "multi" mode. Play the same arpeggiated series of notes and listen (Audio Cue 62C). You should now be hearing the notes moving up and down in pitch at the same speed - but *not* in synchronization with one another (in other words, some notes are going up while others are coming down, and vice versa). **(see figure 10-47)**



6) Now let's try an even more esoteric LFO effect. First, turn the LFO sync back ON. Then position the cursor over the Wave parameter and change this to a square wave. Position it over the Speed parameter and change the speed value to 3. Finally, position it over the Mode parameter and change this back to "single". Play a series of arpeggiated notes and listen (Audio Cue 62D). You should be hearing a *chord* going up and down a full tone in pitch (because of the Pmd value of 67, which was sneakily chosen very carefully), *even though you originally arpeggiated the notes!* This is happening because, in "single" mode, all sixteen LFOs in the sixteen tone generators *must* all start their waves at the same time.

7) Change the Mode parameter to "multi" and play the same arpeggiated series of notes and listen (Audio Cue 62E). Now you will hear the arpeggiation repeat itself, up and down a full tone - and the notes will continue to arpeggiate at the rhythms you originally played at! This is because in "multi" mode, each of the sixteen LFOs start their wave cycles only when their own tone generator receives data from the operators. In this particular case, because LFO sync was also ON, this also had the effect of causing the LFO wave to always start at the beginning of its cycle.

8) Experiment with syncopated rhythms and listen closely (Audio Cue 62F) to the beautiful patterns that can be generated from this stunning effect.

9) Experiment further with different LFO waveshapes and speeds in both "single" and "multi" mode and make sure you clearly understand the difference between the two.

In summary, LFO modulation effects are like any other edit parameters - they can be modified at will, simply by putting your DX7II into edit mode, and entering in new data. This allows us, of course, to modify any LFO settings in any preset, and this is yet another "customizing" job that you should do to the sounds you use in order to optimize them for your particular needs.

The LFO, then, is a powerful tool that allows us to make a *periodic* change to the volume, pitch, or timbre of any sound we create or call up in the DX7II. Depending upon the LFO shape, its speed, and its destination, we can set up a wide variety of different effects - but these effects will always repeat throughout the duration of the sound. The LFO signal can be routed directly, so that its effect becomes an intrinsic part of the sound itself, or it can be routed through any of our real-time controllers, allowing the user an enormous amount of *expressiveness*. All in all, it's amazing what a slow-moving digital oscillator can do!

Chapter Eleven ◇
EG Bias Modulation and Keyboard Velocity Sensitivity

EG Bias Modulation

As we learned in the last chapter, the DX7II has provision for several *real-time controllers:* these include the Modulation wheel, keyboard After touch, the Breath controller, and two different Foot controllers (labeled "FC1" and "FC2"). Each of these can perform identical functions (although some can perform a few more functions than others: the After touch and Breath controller can also perform something called "Pitch Bias" and the Foot controllers can also act as volume pedals. These additional functions will be covered in detail in Chapter Thirteen). In the last chapter, we learned about two of these functions: routing LFO signal to the pitch inputs of our operators (*pitch modulation*), and routing LFO signal to the amplifiers of our six operators (*amplitude modulation*). Another very important type of controlled modulation, however, can be routed through all of these controllers, and this is called *EG bias modulation.* It's important to understand, first of all, that EG bias modulation has nothing whatsoever to do with the LFO, even though its associated operator-specific sensitivity control is also used by the LFO.

In order to explain this unique type of modulation, let's first return to our diagram of an operator. **(see figure 11-1)**

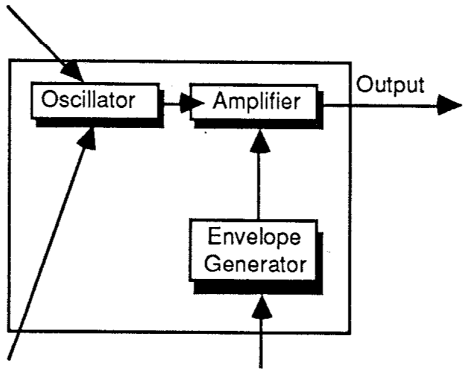


Figure 11-1

The oscillator initiates the audible signal and then passes this signal on to the amplifier, which will shape its amplitude *according to the instructions received by the EG.* The EG, of course, receives *its* instructions from *us*, via the EG data input. Finally, after the amplifier processes it according to the EG commands, the signal exits the operator as an *output.* The purpose of EG bias modulation is to control, *in real time*, the following signal flow. **(see figure 11-2)**

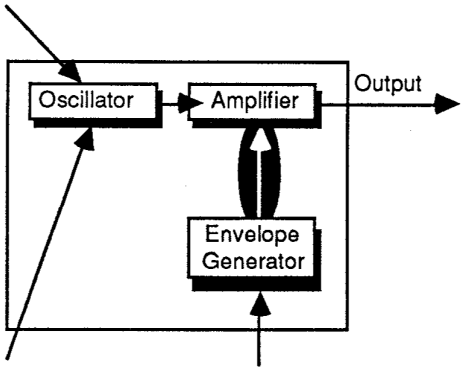


Figure 11-2

This becomes particularly useful when you realize that, for every operator, *the amount of output signal is always directly proportional to the amount of EG control signal received by the amplifier.* Taking this a step further, we can see that if an operator's amplifier receives *no* signal from its EG, then it won't pass ANY signal at all - hence no output. **(see figure 11-3)**

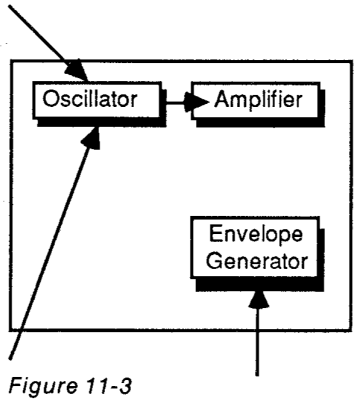


Figure 11-3

EG bias modulation, then, allows us to *directly control, in real time, the output of particular operators.* Cardinal Rules One and Two tell us that this will allow us direct, real-time control of either volume or timbre*, or both, depending upon whether we are affecting carriers or modulators. The only question remaining now is, how do we determine which of our operators will be sensitive to this type of modulation?

* Direct, real-time control of pitch is possible in the DX7II by the use of the *pitch-bend wheel.* See Chapter Thirteen for a detailed description of this device.

The answer, unfortunately, is not an obvious one. I'd love to be able to tell you that the Sensitivity switch (edit switch 11) on your DX7II has a parameter called "EG Bias Sensitivity", but a quick look at its display will tell you that this simply doesn't exist. For reasons that we can only guess at, the DX7II (unlike some of the other DX models - but not the original DX7) does not provide us with such a dedicated parameter. Instead, the operator-specific EG Bias Sensitivity is determined by the operator-specific Ams control - accessed via edit switch 11. In other words, *whichever operators are made sensitive for amplitude modulation will also automatically be sensitive for EG bias modulation.*

How do we determine how much effect a particular real-time controller will have on the EG-to-amplifier signal flow? Just as the depth of controlled LFO modulations are determined by the "Pmod" and "Amod" parameters, the depth of EG bias modulation is simply determined by the "EGbias" parameter in the LCD displays called up by edit switches 25 and 26 (again, you have to press these switches repeatedly in order to call up these different displays). **(see figure 11-4)**

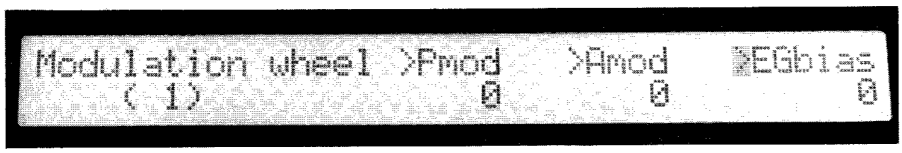


Figure 11-4(a)

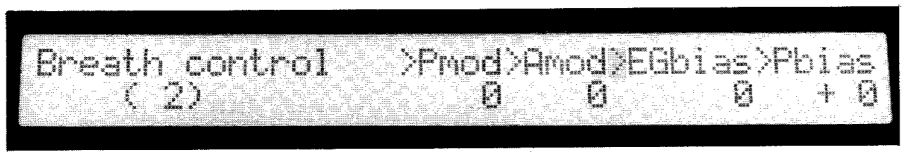


Figure 11-4(b)

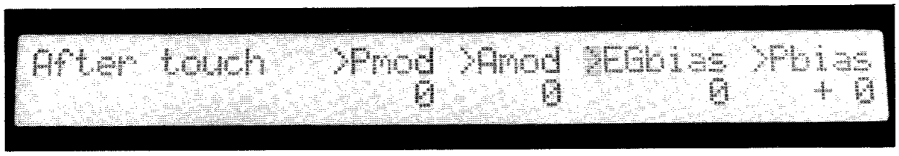


Figure 11-4(c)

Figure 11-4(d)

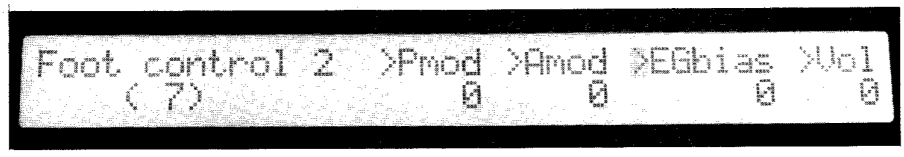


Figure 11-4(e)

Figure 11-4(f)

Therefore, the "patch" for setting up an EG bias modulation in your DX7II is as follows:

1) Decide which operator or operators you wish to have sensitive for EG bias modulation, and make them sensitive by pressing edit switch 11 (SENSITIVITY) and assigning Ams values greater than 0 for those particular operators (in most cases, you will probably want to assign the maximum value of 7 and then use the associated controller "EGbias" parameter for fine-tuning the depth).

2) Decide which controller or controllers you wish to use for EG bias modulation, and use the appropriate "EGbias" parameter in order to determine how much effect that controller will have on the signal flow. If, for example, you wish the Mod wheel to have maximum effect, you would set its "EGbias" parameter to the maximum value of 99. If you wanted the After touch to also have some, but not complete effect, you would set its "EGbias" parameter to a value greater than 0 but less than 99.

Bear in mind that even when using EG bias modulation to control the EG-to-amp signal flow, the EG is still cycling normally in response to "key on" and "key off" flags. In other words, even if you are completely attenuating the EG bias so that the operator amplifier is totally shut down, the EG is still going through whatever movements you programmed in, regardless. It doesn't know, nor does it care, that the amplifier is not receiving its instructions. Furthermore, if you only attenuate the EG bias slightly so that there is reduced output level from a particular operator, the envelope levels are *not* rescaled like they are when the nominal output level is reduced with edit switch 10. This means that the envelopes *won't* travel any faster if the output level is reduced with the EG bias control. We'll run an exercise shortly (Exercise 67), which proves this conclusively.

This means that you'll have to be aware of a few potential pitfalls when using this control: If you try, for example, applying EG bias modulation to a carrier with a short, percussive envelope, its envelope may already be at 0 sustain level (L3) by the time you get around to routing the signal via a real-time controller. Similarly, if you try applying EG bias modulation to a modulator with a very long attack time (R1), the controller may appear to have little or no effect since the output level is slowly building due to the effect of the EG. In any event, EG bias modulation is still one of the most powerful expressiveness controls in this instrument, so it pays to use it often - just be aware of exactly what it is doing, so you won't fall into these traps.

Let's run an exercise now to try out EG bias modulation, as applied to a carrier in a single-system sound:

Exercise 63
EG bias modulation applied to a carrier

1) INITIALIZE your DX7II from single voice play mode, leave it in algorithm #1, TURN OFF operators 3 through 6 ("110000"), and, using the system of operators 1 and 2, GENERATE a sawtooth wave.

2) Press edit switch 11 (SENSITIVITY), followed by edit switch 1, in order to VIEW the sensitivity parameters for operator 1. Use the cursor switches in order to position the cursor over the "Ams" parameter and change this value from its default of 0 to a new value of 7 for operator 1 only. Remember that by doing so, we are not only making

operator 1 as sensitive as it can be to LFO amplitude modulation, but we are also making it totally sensitive to EG bias modulation.

3) Press edit switch 25 repeatedly if necessary in order to call up the Modulation wheel LCD display. Position the cursor over the "EGbias" parameter and enter in the maximum value of 99. Make sure the "Pmod" and "Amod" values remain at their defaults of 0. This has the effect of "instructing" the DX7II that we wish to use the mod wheel to completely control the EG-to-AMP signal flow for any operator or operators (in this case, operator 1) sensitive to amplitude modulation (since the "EGbias" parameter is at its maximum value of 99). Because operator 1 is a carrier in algorithm #1, we can expect to now hear a volume change. Specifically, we can expect the mod wheel to *completely* control the volume of our sound, since operator 1 is the only carrier we are currently using.

4) Put the mod wheel at its MIN position, play a note on the keyboard and listen. You may hear no sound at all, or you may be hearing a slight amount of "bleed-through" signal. Now slowly raise the mod wheel up to its MAX position, holding down a note and listening as you do so (Audio Cue 63A). Note that the mod wheel is currently acting as a volume control.

5) Because the range of this modulation is currently at the maximum value of 99, we currently have *complete* control of the volume of the sound - from the maximum set output level of 99 (the initialization default for operator 1) down to the minimum of 0 (or, depending on the amount of bleed-through, near 0). Inputting a lesser "EGbias" value for the mod wheel will give us less complete control over the *dynamic range* of the sound. Therefore, enter a new "EGbias" value of 65. Put the mod wheel back to its MIN position, play a note, and listen (Audio Cue 63B). Note that, instead of hearing little or no volume, we are now hearing about half-volume. This is because the output level of operator 1 is now currently at about 65% of 99. Slowly raise the mod wheel back up to its MAX position, holding down a note and listening as you do so (Audio Cue 63C). Note that the volume still returns to its previous maximum set output level of 99. As with our other output level controls (i.e. the EGs or the LFO), the EG bias modulation *cannot* increase an operator's output level *beyond* its maximum set level (as determined by edit switch 10). This means that EG bias modulation is an *attenuating* control only.

6) As with LFO modulations, whatever effect we can accomplish with one real-time controller, we can do with any other. Change the Mod wheel "EGbias" parameter back to 0, and press edit switch 25 twice more in order to call up the After touch display. Position the cursor over the After touch "EGbias" parameter and enter in the maximum value of 99. Now press a note lightly on the keyboard and listen. Again, you should be hearing little or no sound. Slowly increase your key pressure up to maximum (don't go crazy!) and note that the volume swells proportionally. In this usage, your actual finger pressure is acting as a volume control!

7) If you have the Breath controller or one or more Foot controllers, experiment by using edit switch 26 in order to route EG bias modulation signal through them, keeping the same operator 1 AMS value. Note that each has precisely the same effect as the other. Wind players particularly will appreciate this usage of the breath controller to directly control the volume of the sound via wind pressure - the harder you blow, the louder it gets!

What if, however, we apply this EG bias modulation to our modulator instead? This will allow us to set up a real-time timbral control, of course, since altering the output level of a modulator always initiates a quantitative timbral change! Let's run an exercise to try it:

Exercise 64
EG bias modulation applied to a modulator

1) INITIALIZE your DX7II from single voice play mode, leave it in algorithm #1, TURN OFF operators 3 through 6 ("110000"), and, using the system of operators 1 and 2, GENERATE a sawtooth wave.

2) Press edit switch 11 (SENSITIVITY), followed by edit switch 2, in order to VIEW the sensitivity parameters for operator 2. Use the cursor switches in order to position the cursor over the "Ams" parameter and change this value from its default of 0 to a new value of 7 for operator 2 only. Remember that by doing so, we are not only making operator 2 as sensitive as it can be to LFO amplitude modulation, but we are also making it totally sensitive to EG bias modulation.

3) Press edit switch 25 repeatedly if necessary in order to call up the Modulation wheel LCD display. Position the cursor over the "EGbias" parameter and enter in the maximum value of 99. Make sure the "Pmod" and "Amod" values remain at their defaults of 0. This has the effect of "instructing" the DX7II that we wish to use the mod wheel to completely control the EG-to-AMP signal flow for any operator or operators (in this case, operator 2 only) that are sensitive to amplitude modulation (since the "EGbias" parameter is at its maximum value of 99). Because operator 2 is a modulator in algorithm #1, we can expect to now hear a quantitative timbral change. Specifically, we can expect the mod wheel to *completely* control the overtone content of our sound, since operator 2 is the only modulator we are currently using.

4) Put the mod wheel at its MIN position, play a note on the keyboard and listen (Audio Cue 64A). Note that you are currently hearing only a sine wave, even though operator 2 is ON, and even though you set it (in step 1) at an output level of 99! This is because the mod wheel is completely controlling its EG-to-AMP signal flow, and hence its output level! Because the mod wheel is currently at its MIN position, there is currently no output signal leaving operator 2 at all. **(see figure 11-5)**

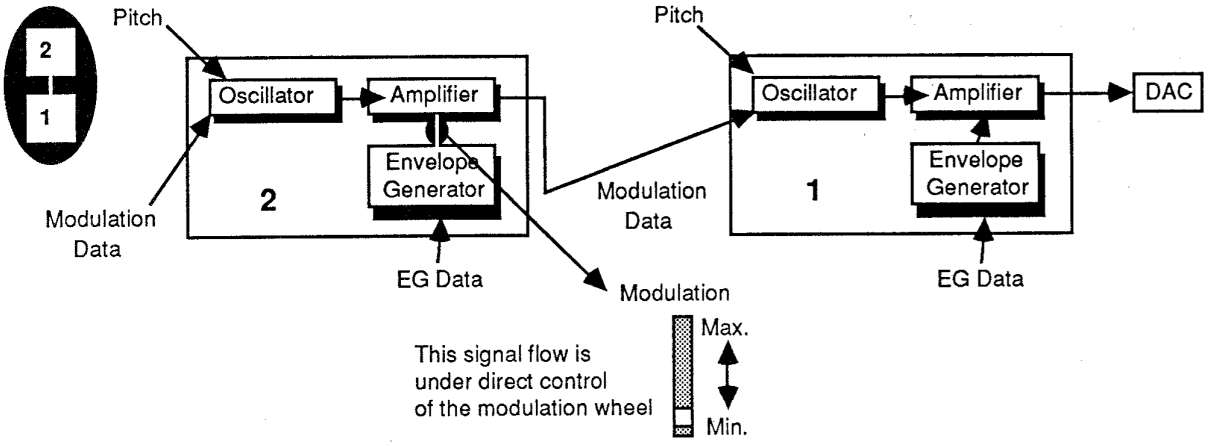


Figure 11-5

5) Now slowly raise the mod wheel up to its MAX position, holding down a note and listening as you do so (Audio Cue 64B). Note that the mod wheel is currently acting as a timbral control, and that raising it towards its MAX position has the effect of increasing the relative amplitude of the overtones you hear - therefore, it is a quantitative timbral control. Note also that **only** at its MAX position do we actually hear the sawtooth wave we originally generated.

6) Because the RANGE of the mod wheel "EGbias" is currently at a value of 99, we currently have *complete* control of the overtone content of the sound - from the maximum set output level of 99 (which gives us a sawtooth wave) down to the minimum of 0 (which gives us a pure sine wave, since there is no modulation data input to our carrier). Inputting a lesser "EGbias" value for the mod wheel will give us less complete control over the overtone content. Therefore, enter a new value of 65 into the mod wheel's "EGbias" parameter. Put the mod wheel back to its MIN position, play a note and listen (Audio Cue 64C). Note that, instead of hearing a pure sine wave, we are currently hearing a gentle timbre with few overtones. This is because the output level of operator 2 is now at approximately 65% of 99. Slowly raise the mod wheel back up to its MAX position, holding down a note and listening as you do so (Audio Cue 64D). Note that the timbre still returns to its previous maximum set output level of 99, still giving us a sawtooth wave at the MAX setting. Again, EG bias modulation *cannot* increase an operator's output level *beyond* its maximum set level (as determined by edit switch 10).

7) As with LFO modulations, whatever effect we can accomplish with one real-time controller, we can do with any other. Change the mod wheel "EGbias" back to 0, and press edit switch 25 twice more in order to call up the After touch display. Position the cursor over the "EGbias" parameter, and enter in the maximum value of 99. Now press a note lightly on the keyboard and listen. Again, you should be hearing a pure sine wave. Slowly increase your key pressure up to maximum and note that the overtone content increases proportionally. In this usage, your actual finger pressure is acting as a quantitative timbral control!

8) If you have the Breath controller or one or more Foot controllers, experiment by using edit switch 26 in order to route EG bias modulation signal through them, keeping the same operator 2 Ams value. Note that each has precisely the same effect as the other. Wind players will also appreciate this usage of the breath controller to directly control the brightness of our sound via wind pressure - the harder you blow, the brighter it gets!

9) Because EG bias modulation is operator-specific, we can use it to affect volume *and* timbre, in differing degrees, if we so desire. Press edit switch 11 (SENSITIVITY), followed by edit switch 1, and enter the maximum Ams value of 7 for operator 1 as well. Now press edit switch 25 or 26, and use one of your real-time controllers to route maximum EG bias modulation signal. As you change the real-time controller from its MIN to MAX position (either by raising the mod wheel, stepping on the Foot controller, blowing harder into the Breath controller, or pressing down harder on the keyboard (After touch), hold a note down and listen (Audio Cue 64E). Note that your controller is now acting as a volume *and* quantitative timbral control!

10) Experiment further by using *different* Ams (EG bias sensitivity) values for operators 1 and 2. Note that this allows you to change the volume of a sound a great deal, while only causing a slight timbral change, or vice-versa. Experiment also by using EG bias modulation with single timbres other than a simple sawtooth wave.

So far, so good. However, the limitations of having two different modulation sensitivities sharing the same parameter (Ams) should be apparent. For one thing, you won't be able to have some operators undergoing LFO amplitude modulation, and different ones undergoing EG bias modulation. However, we *can* have *different* controllers routing each of the two different signals, and this can allow for some elegant and complex expressiveness as we play this instrument. The exercise below demonstrates how we can set this up:

Exercise 65
Dual modulations

1) INITIALIZE your DX7II from single voice play mode, leave it in algorithm #1, TURN OFF operators 3 through 6 ("110000"), and, using the system of operators 1 and 2, GENERATE a *square* wave.

2) Press edit switch 11, followed by edit switch 1, in order to VIEW the sensitivities for operator 1. Position the cursor over the Ams parameter and change this from its default of 0 to the maximum value of 7 for operator 1 only. Operator 1 is now as sensitive as it can be to both LFO amplitude modulation *and* EG bias modulation.

3) Press edit switch 25 repeatedly if necessary in order to call up the Modulation wheel display. Position the cursor over the "Amod" parameter and enter a value of 99. The modulation wheel is now routing ALL of the LFO signal to the operator amplifiers. Step 2 above ensured that only operator 1 would be sensitive for this effect, however.

4) Press edit switch 25 twice more in order to call up the After touch display. Position the cursor over the "EGbias" parameter and enter a value of 99. The keyboard after touch is now routing ALL of the EG-to-AMP signal for all operators. Step 2 above ensured that, again, only operator 1 will be sensitive for this effect.

5) Put the mod wheel at its MIN position, play a note lightly on the keyboard and listen. Note that you are currently hearing little or no sound. Press down the key with steadily increasing pressure and listen. Note that the volume slowly increases proportionally to your finger pressure. This is occurring because the After touch is currently controlling the EG-to-AMP signal for operator 1 to the maximum degree.

6) The mod wheel, on the other hand, is not controlling this signal flow but is instead routing LFO signal (currently a default triangle wave at the default speed of 35) to the amplifiers of all six operators. At this moment, of course, only operator 1 is sensitive to this signal and so we will hear a periodic volume change, a *tremolo* effect, when we raise the mod wheel. Hold a note down with half-pressure, slowly raise the mod wheel towards its MAX position, then slowly increase your finger pressure, and listen (Audio Cue 65A). Note that the mod wheel determines the depth of the tremolo, while the After touch determines the strength of the overall volume.

7) If you have the Breath controller or one or more Foot controllers, experiment by using either of them in addition to the mod wheel or After touch to route either EG bias or amplitude modulation. Experiment further by setting up some *pitch* modulation sensitivity with edit switch 11, and using a real-time controller to route LFO signal to the pitch inputs of your operators as well. In this manner, you can actually set up *triple* modulation effects! The limitations are, firstly, that we only have the one virtual LFO, so we can only use one waveshape and speed at a time; and secondly, that we cannot have some operators respond to

Algorithm #16:

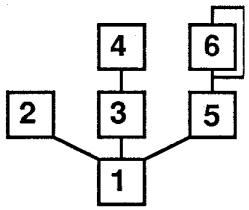


Figure 11-6

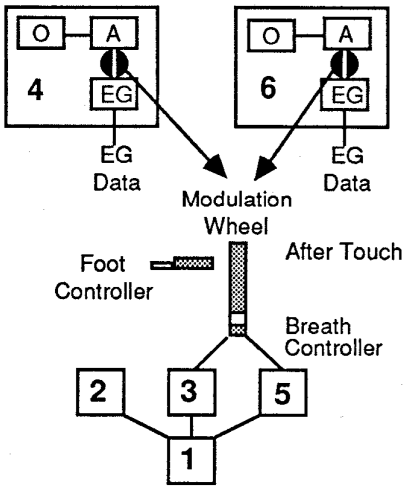


Figure 11-7

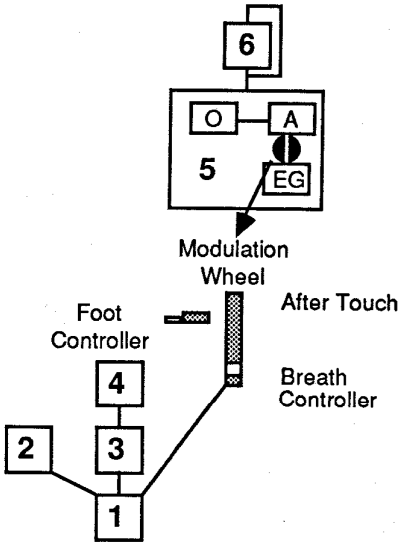


Figure 11-8

amplitude modulation and different ones respond to EG bias modulation. We can, however, use any of our controllers to route any or all of these modulation signals.

Up until now, we have only applied EG bias modulation to simple timbres; that is, sounds created with a single modulator-carrier system. The importance of this particular modulation becomes more apparent when we work with either multi-system sounds, or multi-modulator systems. For example, some of you may still be wondering about the potential use of a single-carrier algorithm like algorithm #16. **(see figure 11-6)**

By using EG bias modulation, we can, for example, create a sound with algorithm #16 where the upper modulators (operators 4 and 6) don't ever contribute to the sound at all unless a real-time controller routes their signal **(see figure 11-7)** or where, perhaps an entire *stack* (like operators 5 and 6) makes no contribution unless routed via a controller. **(see figure 11-8)**

Try setting up these two possibilities in your DX7II, and it should become apparent that this tool, plus the actions of the EGs (which allow operators to "fade in" and "fade out") make the potential applications for single-carrier algorithms such as these more obvious.

An even more interesting application for EG bias modulation, however, is selective use with one or more carriers in a multi-carrier algorithm (which, of course, is everything except algorithms #16, 17, and 18). Let's run an Exercise to try this out - and in doing so, we can, as promised, demonstrate that EG bias modulation, unlike operator output level, does *not* affect the rates of movement of the operator EGs.

Exercise 66
EG bias modulation applied to selective carriers

1) INITIALIZE your DX7II from single voice play mode, and select algorithm #3. **(see figure 11-9)** This algorithm has two systems, both of them containing modulator stacks. For this exercise, however, we won't be needing the stacks, so TURN OFF operators 3 and 6 ("110110").

2) With the system of operators 1 and 2, GENERATE a sawtooth wave. Drop the pitch of this sawtooth wave down one full octave (by changing the frequency ratio to 0.50 : 0.50). TURN OFF everything except this system ("110000"), play middle C on the keyboard, and listen (Audio Cue 66A).

3) Now TURN OFF operators 1 and 2 and TURN ON operators 4 and 5 ("000110"). With the system of operators 4 and 5, GENERATE a square wave with maximum carrier (operator 4) output level (=99). Raise the pitch of this square wave up two full octaves (by changing the frequency ratio to 8.00 : 4.00). Play middle C and listen (Audio Cue 66B).

4) TURN operators 1 and 2 back ON ("110110"), play middle C, and listen (Audio Cue 66C). Even though both systems are at full output level, the low-pitched sawtooth wave sounds louder than the higher square wave. This is simply because the sawtooth wave has a greater harmonic content.

5) Now let's set up the same exact double-attack EG in both carriers. Press edit switch 9 (EG) and enter in the following EG values for operators 1 and 4:

Rs	R1	R2	R3	R4	L1	L2	L3	L4
0	99	46	46	99	99	0	99	0

This creates an envelope shape that looks like **figure 11-10.** Note that *no* rate-scaling should be employed. If you like, you can use the *EG and Scaling* copy function (see Chapter Nine) to simply copy these values from operator 1 to operator 4, or vice versa. Make sure you leave the modulator EGs (for operators 2 and 5 at their default square settings.

6) Play a note, hold it down, and listen (Audio Cue 66D). Note that both the low-pitched sawtooth wave and the higher-pitched square wave attack and reattack at precisely the same times.

7) Now let's lower the output level of the sawtooth wave - operator 1. Press edit switch 10, followed by edit switch 1, in order to call up the Level display. **(see figure 11-11)** Position the cursor over the Level parameter and enter in a new value of 75 for operator 1 only. Play a note, hold it down and listen (Audio Cue 66E). Note that not only is the volume of this system considerably lower than that of the higher-pitched square wave, but that it fades away and reattacks significantly faster than the square wave. This is because altering the output level of an operator re-scales the EG levels, thereby causing the rates to arrive at their destinations in lesser time. **(see figure 11-12)**

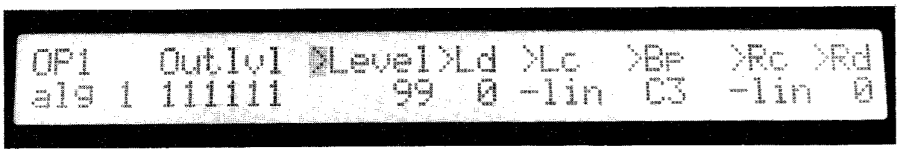


Figure 11-11

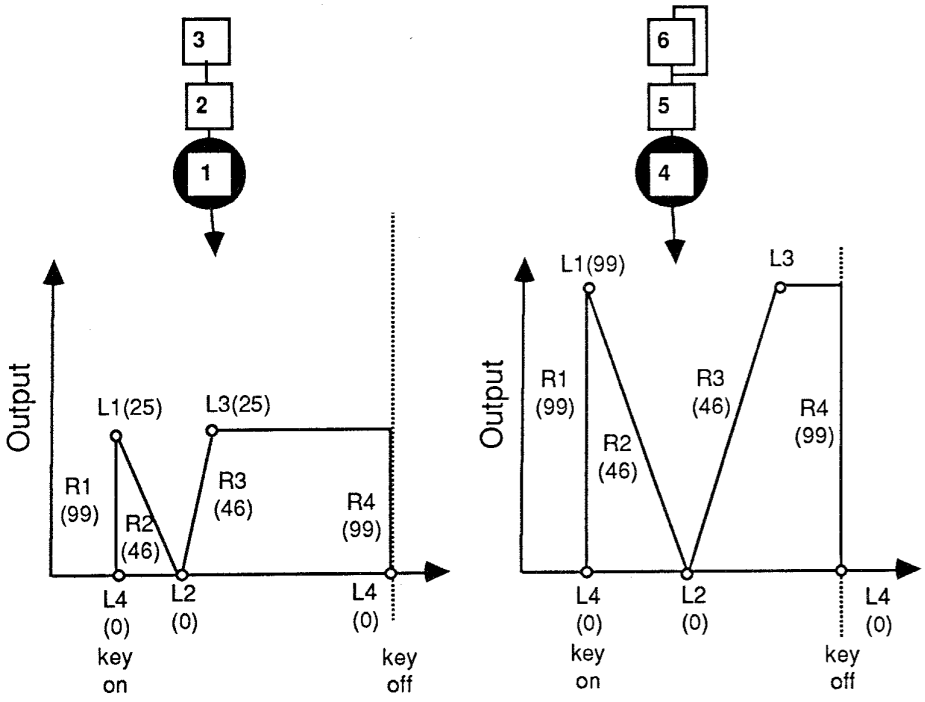


Figure 11-12

Experiment by restoring the Level of operator 1 back to 99 and changing the output level of the square wave carrier (operator 4) instead to 75. Play a note and listen (Audio Cue 66F). Note that the effect is now completely reversed, making the higher-pitched square wave not only softer, but causing it to fades away and reattack significantly faster than the lower-pitched sawtooth wave.

Algorithm #3:

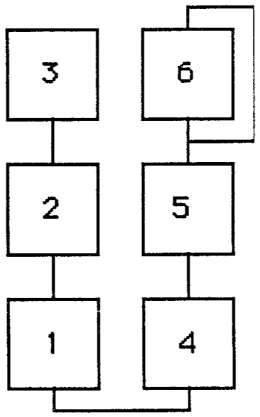


Figure 11-9

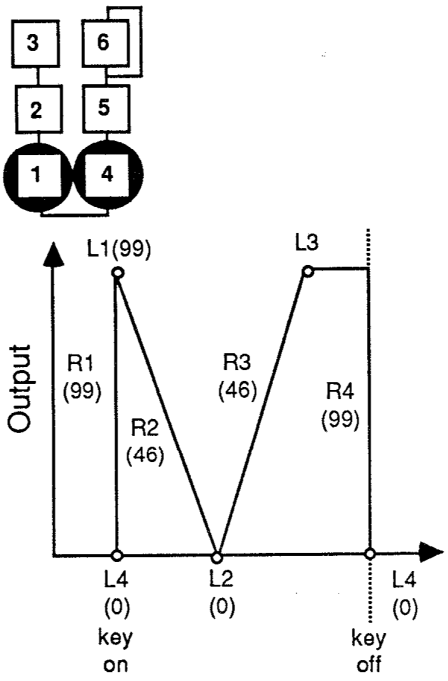


Figure 11-10

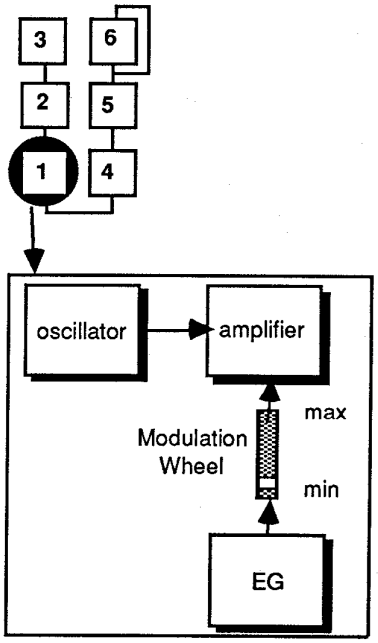


Figure 11-13

8) Restore the output level of operator 4 back to 99, so that both carriers are at precisely the same volume. Play a note and listen (Audio Cue 66G). Once again, the two components fade away and reattack at precisely the same times.

9) Now let's try attenuating the output level of one of the carriers with EG bias modulation instead. We'll start with operator 1 - the carrier responsible for the low-pitched sawtooth wave. Press edit switch 11, followed by edit switch 1, in order to VIEW the sensitivities of operator 1. Position the cursor over the Ams parameter and change this to the maximum value of 7. Operator 1 is now completely sensitive to EG bias modulation (as well as LFO amplitude modulation, which we won't be using here). Make sure all the other operators are at their Ams defaults of 0.

10) Press edit switch 25 repeatedly if necessary in order to call up the Modulation wheel display. Position the cursor over the "EGbias" parameter and enter in the maximum value of 99. The entire EG-to-AMP signal flow for operator 1 is now under the direct control of the modulation wheel. **(see figure 11-13)**

11) Place the wheel at its MIN position, play a note and listen (Audio Cue 66H). Note that we now hear the higher-pitched square wave only, since operator 1's output level is now effectively under the control of the mod wheel. Now place the wheel at the half-way point between MIN and MAX. Play a note and listen (Audio Cue 66I). Note that the lower-pitched sawtooth wave is now present at about half-volume, *but that it fades away and reattacks at precisely the same speed as the louder, higher-pitched square wave!* Continue inching the mod wheel up towards its MAX position, playing a note and listening at each small increment (Audio Cue 66J). Note that, while this has the action of increasing the volume of the sawtooth wave, that no change whatsoever occurs to the speed of operator 1's EG: the two components continue to fade away and reattack at precisely the same time.

12) Experiment by repeating steps 9 through 11 above, but this time making the carrier of the high-pitched square wave (operator 4) the object of EG bias modulation. Note that precisely the same effect occurs: the mod wheel controls the volume of this component, but the envelope remains exactly the same. This is the important concept: altering an operator's output level with EG bias will in no way change its EG movements.

The reason we didn't apply the EG bias modulation to any of the modulators in the above exercise was, of course, because we were not looking to make any kind of timbral change to our sound - but we could just as easily have done so. EG bias modulation can and often will be applied to modulators as well as carriers - and, just as in this last exercise, the movements of the envelopes will *not* be affected.

Bear in mind also that, just as with pitch or amplitude modulation, if you plan on using EG bias modulation in a sound, you will have to set it up in advance in edit mode by using the Ams parameter in edit switch 11 - and you will also have to set up your controller routings with edit switches 25 and/or 26. There are, for example, several Yamaha presets in bank 2 of your ROM cartridge: "BC Sax" (slot 18), "BC Trumpet" (slot 27), "FC Strings" (slot 51), and "FC Choir" (slot 52), which have all been pre-programmed for EG bias modulation. The "BC" in some of these names is meant to tell you that the Breath controller is routing this modulation, but, of course, you can in fact edit these sounds to use

any of the other real-time controllers. In "BC Sax" and "BC Trumpet", all the carriers have been set to be as sensitive as possible to this effect (Ams=7), and the "EGbias" for the Breath controller is set to 99, so you won't hear these voices *at all* unless you've got a Breath controller plugged in and you're puffing away! The "FC" similarly stands for "Foot controller", and you are being informed that one or both of the Foot controllers is in use for EG bias modulation. You should be able to hear these sounds even without a foot controller plugged in, however.

Many of the ROM presets also have the letter "B" at the end (like "Whisper B" in ROM bank 1, slot 53, or "Glastine B" in ROM bank 2, slot 33. These are mostly sounds which have been given extensive expressive controls via EG bias modulation - they will all get realistically louder and/or brighter when you use the routing controller to increase the EG bias signal flow. They are really meant to be used with their "A" counterparts ("Whisper A" in bank 1, slot 10 and "Glastine A" in bank 2, slot 14) in dual mode, but of course you can also use them in single or split mode. Just be aware that extensive EG bias modulation has usually been built into these voices. A quick look at the parameters of edit switches 11, 25, and 26, will tell you all you need to know about how these - and any other - sounds have been pre-programmed. It's important to realize that all of these sounds can be modified and that you yourself can change this expressiveness or add it in any way you want to any sound you create or modify.

Keyboard Velocity Sensitivity

As mentioned in the last chapter, After touch is only one oftwo keyboard sensitivities in the DX7II, the other being *velocity sensitivity*. Here, the term "velocity" refers to speed, but not horizontal speed. In other words, the keyboard velocity sensitivity has nothing to do with how quickly you play your Keith Emerson licks (God, am I showing my age!) but instead is responsive to *vertical* speed: how quickly each key travels from its resting position down to the fully depressed position. **(see figure 11-14)**

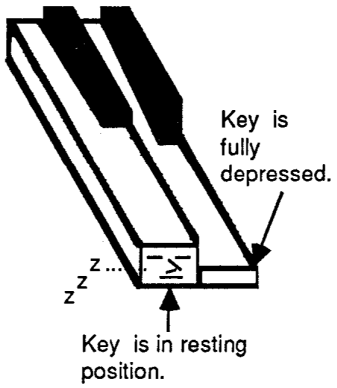


Figure 11-14

Underneath each key of the DX7II keyboard is a timing mechanism, actually a computer clock which "counts" how many clock pulses it took that particular key to drop all the way down. If the key drops very slowly, it will count many of these pulses and our microprocessor will *inversely* generate a very small number. If it drops rapidly, far fewer pulses are counted, and an inversely large number will be generated. **(see figure 11-15)**

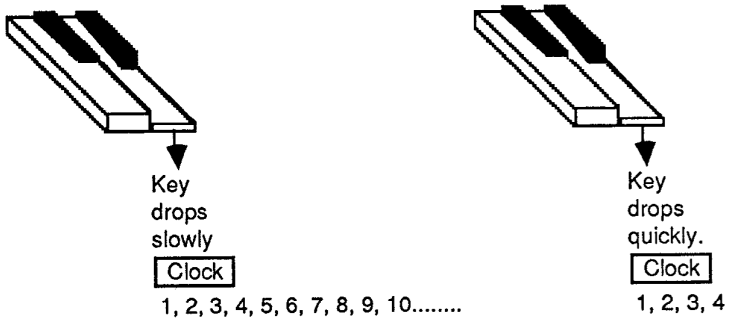


Figure 11-15

This timing mechanism, remember, is independent for each key, making keyboard velocity sensitivity a *polyphonic* control. If you play a chord, for example, but depress each of the individual notes at slightly different speeds, each of the associated keys will generate a different velocity number.

The obvious question now is, what on earth is done with that number? There are two potential applications. The more important of the two is that it will be applied to each of the six operators' *output level*. The other application is to send them to the pitch EG - but we'll talk about that function shortly.

Let's stick with the output level application for now. Applied to the output level of *all six* operators, did we say? Does this make it a non-operator-specific control? Not at all, because, just as with amplitude modulation, we can set each operator individually to a different *sensitivity* to this control. This will, of course, allow us to control the volume and/or the timbre of any sound or portion of a sound. The keyboard velocity sensitivity is determined with the operator-specific SENSITIVITY switch, edit switch 11, and is set with the "Velocity" parameter in the LCD display. **(see figure 11-16)**

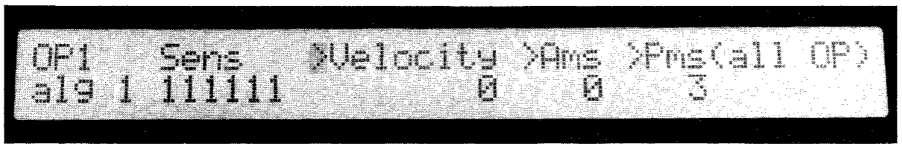


Figure 11-16

The range of this operator-specific parameter is 0 to 7. 0 represents no sensitivity, and any operator given a velocity sensitivity of 0 will simply ignore the velocity number being sent it by the key timing mechanism. A value of 7 represents maximum velocity sensitivity, and an operator assigned a value of 7 will have the full range of its maximum set output level (as determined by the Level parameter in edit switch 10) controlled by the speed of the key depressions. The in-between sensitivity values (1 through 6) simply allow different shadings of sensitivity to this control. As with the EG, LFO, and EG bias modulation controls, keyboard velocity sensitivity *cannot* ever increase an operator's output level beyond the Level value set with edit switch 10. In this way, keyboard velocity sensitivity, like the EG, LFO, and EG bias modulation, is purely an *attenuating* control. Upon initialization, the default for this parameter is 0 (no sensitivity) for all six operators.

Often, people think of this velocity sensitivity as something which measures how *hard* you strike the key, and in the interests of extending the life of your keyboard, you should avoid this erroneous assumption. Logically, the harder you strike a key, the faster the key will drop, but this control is not in fact measuring how hard the key is struck at all. With a little practice, you should be able to refine your keyboard technique so that you can depress specific notes quickly without slamming your hand down on the aching keyboard! The two keyboard sensitivities - After touch and Velocity - have been as much a curse as a blessing to certain DX7II owners who have ended up spending many unhappy hours in the lobby of their Authorized Service Center waiting for broken keys to be replaced. While the keyboard of this instrument is generally robust, it *can* succumb to over-enthusiasm! Learn to use these keyboard sensitivities without going nuts, and you'll spare yourself unnecessary headaches...

Okay, with that Stern Admonishment out of the way (cla-a-a-ss!), let's run an exercise to try it out:

Exercise 67
Keyboard velocity sensitivity

1) INITIALIZE your DX7II from single voice play mode, leave it in algorithm #1, TURN OFF operators 3 through 6 ("110000") and, using the system of operators 1 and 2, GENERATE a sawtooth wave. Play a key on the keyboard and listen (Audio Cue 67A). Note the current volume of the sound.

2) Press edit switch 11, followed by edit switch 1, in order to VIEW the sensitivity defaults for operator 1. Observe that the Velocity parameter is currently at its default of 0. Position the cursor over this and change it to the maximum value of 7 for operator 1 only.

3) Play a key on the keyboard, but depress the key *as slowly as possible*. Listen (Audio Cue 67B). Note that, because operator 1 in this system is a carrier, we hear little or no volume (again, if you are hearing a little bleed-through, not to worry). Now play the same note, with a quicker attack; attempt to get the key depressed as quickly as possible without mashing it down Neanderthal-style. Listen (Audio Cue 67C). Note that we now hear the sawtooth wave at its original, full volume.

4) Try playing the key at varying speeds and note the many different volumes available with the maximum sensitivity value of 7 (Audio Cue 67D). This is because the maximum sensitivity setting gives us control over the full dynamic range, from an operator 1 output level of 0 (note struck at slowest speed) to the current maximum set output level of 99 (note struck at fastest speed).

5) Remember that velocity sensitivity is a *polyphonic* effect. Try striking the lowest key of the keyboard as slowly as possible and simultaneously striking the highest key as quickly as possible (if you're fairly uncoordinated, or your name is Jerry Ford, you might want to have a friend help you do this). You should now be hearing the high note only. Try repeating this in reverse, striking the highest key as slowly as possible and the lowest as quickly as possible. The reverse now occurs and you should now be hearing the low note only.

6) Change the velocity sensitivity for operator 1 only to a new value of 4. Again, play a key on the keyboard as slowly as possible and listen (Audio Cue 67E). Note that, this time, we do hear some volume, but at a greatly reduced level. That's because velocity sensitivity is a *negative* sensitivity control.

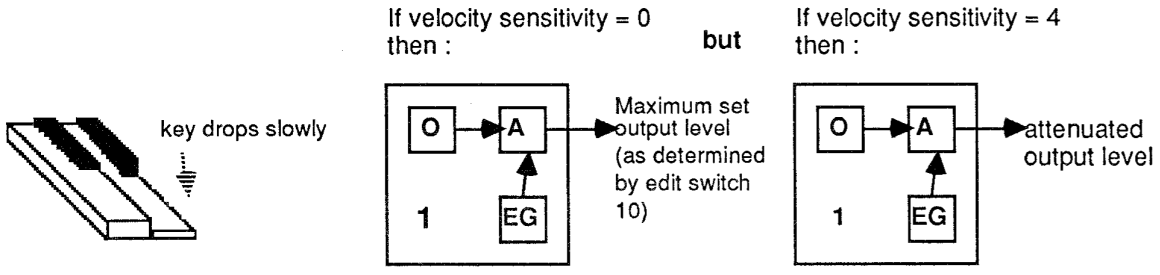


Figure 11-17

7) Now play the same key as quickly as possible and listen (Audio Cue 67F). Note that, as before, we hear the note at its original, full volume.

8) Once again, play the key at varying speeds and listen (Audio Cue 67G). Note that, while we still hear many different volumes, the full dynamic range is not at our disposal because the velocity sensitivity is only at about half. This means that we are hearing between half-volume and full volume only. Wherever this is applied to a carrier (as here), however, velocity sensitivity will act as a *volume* control, which will always result in higher volumes with faster key depressions. There is no way to invert this effect in the DX7II (in other words, you can't ever get softer volumes by pressing the key harder, or vice versa).

9) Restore operator 1's velocity sensitivity value back to 0 and press edit switch 2 in order to VIEW operator 2. Change operator 2's velocity sensitivity value from its current default of 0 to the maximum value of 7. Play a key on the keyboard as slowly as possible and listen (Audio Cue 67H). Note that we are now hearing a simple sine wave only. This is because the output level of operator 2 (the modulator in this system) is under the full control of the velocity timing mechanism. Because you depressed the key as slowly as possible, the timing mechanism generated a velocity number of 0, and this number was applied to operator 2's output level. **(see figure 11-18)** We are hearing the sine wave at full volume because operator 1 (the carrier) is currently insensitive (having a value of 0) to the keyboard velocity.

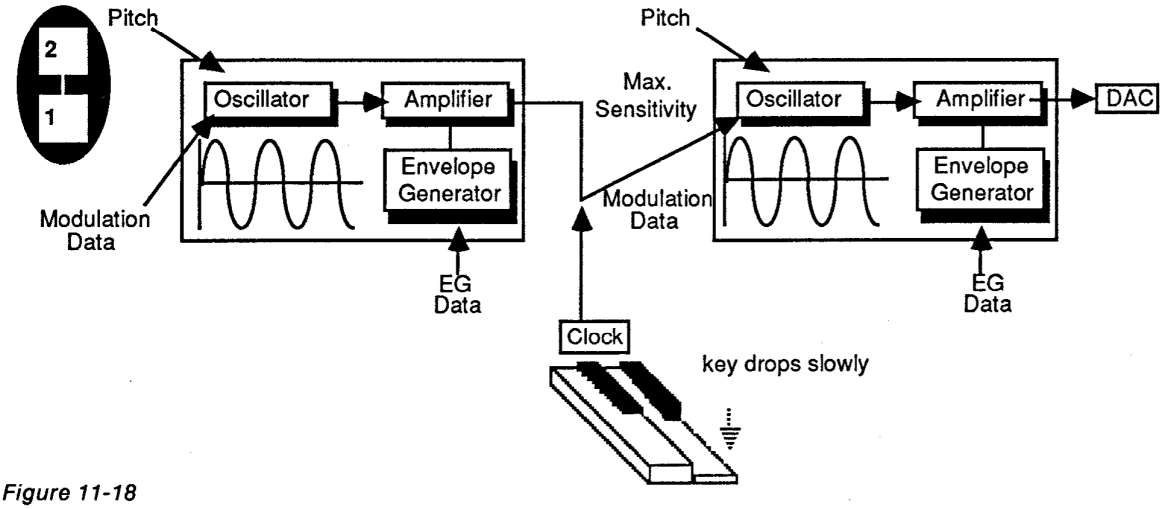


Figure 11-18

10) Play the same key as quickly as possible and listen (Audio Cue 67I). Note that we now hear the sawtooth wave, as before! Now play the key at several different speeds and note the varying timbres that result because the keyboard velocity sensitivity is controlling the output level of our modulator (which Cardinal Rule Two tells us is the quantitative timbral control). (Audio Cue 67J).

11) Again, the polyphonic aspect of this control can be demonstrated by repeating step 5 above. Note that in the first instance, you now hear the lowest note as a sine wave and the highest as a sawtooth; and in the second instance, the reverse occurs.

12) Change the velocity sensitivity for operator 2 only to a new value of 4, play a key on the keyboard as slowly as possible, and listen (Audio Cue 67K). Note that, instead of hearing a pure sine wave, we are now hearing an "in-between" timbre; a sine wave with a few overtones, but not enough of them to be called a sawtooth wave. This is because we have lowered the sensitivity of operator 2 to the key timing mechanism, giving us less dynamic range in our control. **(see figure 11-19)**

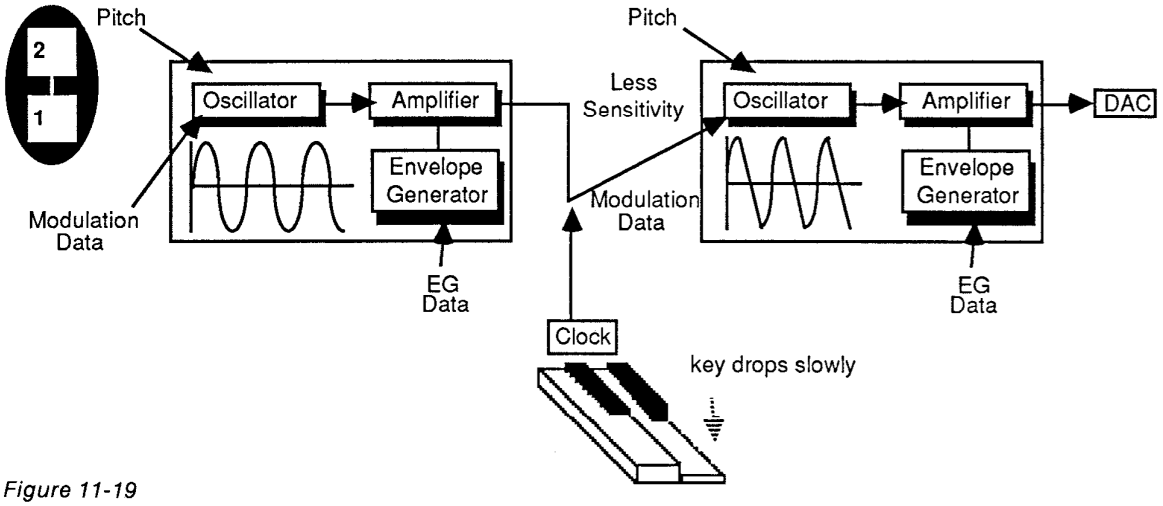


Figure 11-19

13) Now play the same key on the keyboard as quickly as possible and note that we once again hear the sawtooth wave as before (Audio Cue 67L). Play the key with varying speeds and note that, while the timbre still undergoes many different changes, we now do not have the full range of sine wave-to-sawtooth wave available (Audio Cue 67M). Whenever applying velocity sensitivity to a modulator (as we are doing here), it will act as a *brightness* control, with brighter sounds always resulting from faster key depressions. As before, there is no way to invert this control (which would give us brighter sounds as we struck the key slower).

14) Because velocity sensitivity is operator-specific, we can affect the volume *and* timbre of a sound, and to differing degrees if we so desire. Leave operator 2's velocity sensitivity at its current value of 4 but change this value for operator 1 to the maximum of 7. Play a key at varying speeds and listen (Audio Cue 67N). Note that this results in a great amount of volume change and a relatively smaller amount of timbral change as you strike the key faster and faster. Now try reversing the sensitivities - giving operator 1 a sensitivity value of 4, and operator 2 the maximum value of 7. Again, play a key at varying speeds and listen (Audio Cue 67O). Note that we now hear a great amount of timbral change, but only a small amount of volume change.

15) Experiment by creating different single timbres and applying this velocity sensitivity control in different degrees to the carrier and/or to the modulator, noting how each change affects the overall sound.

As with EG bias modulation, the real power of this control lies in the fact that it is operator-specific. This will allow us to apply it, for example, to specific systems within an overall sound. Let's try creating a complex sound, composed of two notes a fifth apart, with the root note as a square wave, and the fifth as a louder, brighter timbre, and then use the velocity sensitivity to selectively "fade in" one note or the other:

Algorithm #26:

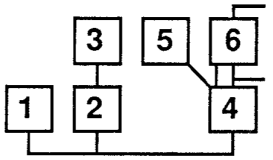


Figure 11-20

Exercise 68
Velocity sensitivity applied to a multi-system voice

1) INITIALIZE your DX7II from single voice play mode, and change it from the default of algorithm #1 to algorithm #26. (see figure 11-20)

2) TURN OFF operator 1 and operators 4 through 6 ("011000"), and, using the system of operators 2 and 3, GENERATE a square wave at maximum output (operator 4 Level = 99).

3) TURN OFF operators 2 and 3, and TURN ON operators 4 and 5 ("000110"). Using the system of operators 4 and 5, GENERATE another square wave at maximum output (operator 4 Level 99).

4) TURN OFF operator 5 and TURN ON operator 6 ("000101"). Using the system of operators 4 and 6, GENERATE yet another square wave. TURN ON operator 5 ("111000") and listen (Audio Cue 68A). Note that the sound is now considerably brighter than a square wave, since both operators 5 and 6 are now providing modulation data input to our carrier (operator 4).

5) Note that in this algorithm, operator 6 has been assigned (by Yamaha, not by us) the feedback loop. Press edit switch 7 and change the Fbl parameter to the maximum value of 7. Play a note on the keyboard and listen (Audio Cue 68B). Note that the sound is now brighter still.

6) Change the pitch of this entire system (that of operators 4, 5, and 6) so it is a musical fifth higher (if you can't remember how to do this, refer to Chapter Six).*

7) TURN ON operators 2 and 3 ("011111") and press edit switch 10, followed by edit switch 2. Change operator 2's Level parameter to a new value of 92. Play a key and listen (Audio Cue 68C). Note that you are now hearing two different timbres, a fifth apart, with the higher note being both louder and brighter than the root note.

8) Press edit switch 11, followed by edit switch 6, in order to VIEW the sensitivity values for operator 6. Change the Velocity (sensitivity) parameter to a new value of 3 for operator 6 only. Play a key at several different speeds and note that a slight change in timbre now occurs at higher key velocities (Audio Cue 68D).

9) Press edit switch 4 in order to VIEW operator 4 and change its Velocity (sensitivity) parameter to the maximum value of 7. Depress a key as slowly as possible and listen (Audio Cue 68E). Note that we now hear only the root note produced by the system of operators 2 and 3. Now play the same key at the quickest speed possible and listen (Audio Cue 68F). Note that we now hear both notes together, as in step 7 above. Play the key at several different speeds and note that you can "cross-fade" the higher note in and out as you strike the key more quickly and less quickly (Audio Cue 68G). Among other things, this will allow you to develop boring root-to-fifth bass lines by playing just a single key! (Audio Cue 68H).

10) You can make this sound far more interesting by working with the operator EGs. Experiment by changing them to sharp, percussive shapes, for example. Experiment further by bringing operator 1 (in this algorithm, an unmodulated carrier) into play: tune it to a different pitch, or perhaps route some LFO signal to it; or control its output with one of the real-time controllers using EG bias modulation. With a little work, you can turn this basic patch into a good, usable sound. Try it!

* For those of you with poor memories, or those of you who have skipped ahead to this point without ever reading Chapter Six, or those of you who are just plain too lazy to turn the pages back, here's how you do it: Cardinal Rule Three tells us we must affect the carrier and modulator the same way. Since 3 times the fundamental is an octave and a fifth higher, then 1.5 times the fundamental will be a simple fifth higher. Therefore, you'll need to set up a frequency ratio between operators 5 and 4 of 3.00 : 1.50, and the same ratio between operators 6 and 4.

The ROM preset called "Fifths" (bank 2, slot 40) is similar in principle to the sound we created from scratch in Exercise 68 above. Try calling up this voice and putting your DX7II into edit mode in order to see how the sound was created. You should find that you can create a similar kind of root-to-fifth crossfade by simply increasing the velocity sensitivity of operator 4. Try it!

The potential of this velocity sensitivity should start becoming apparent to you as you begin experimenting with it with different DX7II voices. You will find that many of the ROM presets - particularly those that emulate acoustic instruments - have a certain amount of velocity sensitivity already programmed in. If you don't care for this effect in a particular sound, or you feel there's too much or too little, change it! That's what edit mode is for, after all! Conversely, there are quite a few presets with no velocity sensitivity whatsoever. Often you can unlock new doors of expressive sounds by adding this control to specific operators within the sound. Go through your presets and try changing the Velocity parameter for various operators in each of them - you'll probably find many ways of improving the sounds you've already got. Bear in mind that whenever you *decrease* the velocity sensitivity for a particular operator, its output level will appear to increase, since this is a *negative* control. In other words, playing keys at lesser velocities will put that operator back at its often unusually high nominal output level setting. To compensate for this, you'll have to lower the nominal Level value accordingly - and let your ears be the judge of just how much it needs to be changed. It is worth pointing out that when using keyboard velocity sensitivity to attenuate an operator's output level (by striking a key slowly), the operator EGs *do* rescale themselves accordingly - just as if you had lowered the nominal output level with edit switch 10. This is unlike the effect of EG bias modulation, which, as we saw in Exercise 67 above, has no effect on the EG movements. To prove this to yourself, simply redo Exercise 67 and, instead of making a carrier sensitive for amplitude (EG bias) modulation (with the Ams control), substitute the Velocity sensitivity with the same values. You'll hear the affected system fade away and reattack more and more quickly (in addition to having less and less volume) as you strike the keys more and more slowly!

To summarize: Velocity sensitivity as applied to operator output level (via edit switch 11) is an expressiveness control, allowing us to make a sound louder or brighter when we strike a key with greater speed. By applying this control to specific systems within an overall sound, we can also set up novel effects as in the Exercise above.

The secondary application of keyboard velocity sensitivity in the DX7II is to the pitch EG. Press edit switch 13 (PITCH EG), and you will see in the LCD display **figure 11-21.**

Figure 11-21

The "Vel" parameter (second from left) is a simple "on-off" control, turned ON and OFF with the data entry "yes-no" buttons, as usual. When OFF (the initialization default), the pitch EG will undergo whatever movements you program into it, without regard to the speed

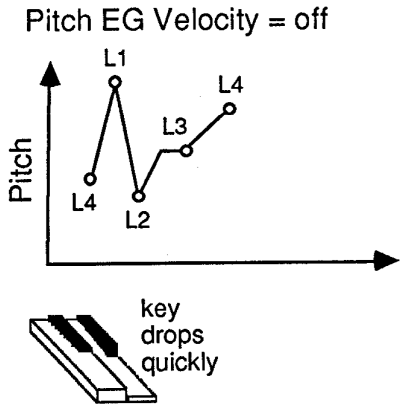


Figure 11-22

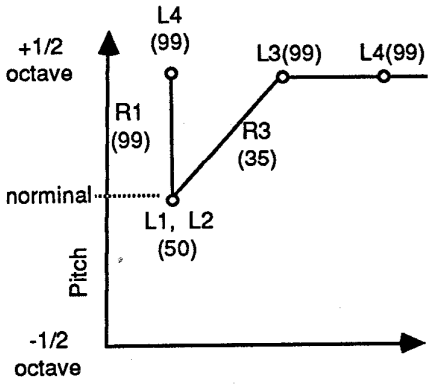


Figure 11-23

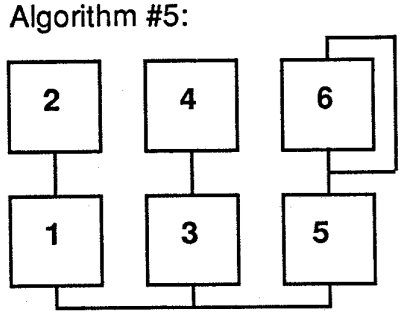


Figure 11-24

of your key depressions. When ON, however, the Levels of the pitch EG will rescale themselves according to the size of the velocity signal received from the keyboard; in other words, the faster you strike the key, the further apart different Levels will be from one another. This, naturally, will have the resultant effect of changing the speed with which the pitch EG undergoes its movements, since the Rates remain constant (see Chapter Nine for a full explanation of this phenomenon). **(see figure 11-22)**

Let's run an Exercise now to try out this effect, and in the process, create a strikingly beautiful sound:

Exercise 69
Velocity sensitivity as applied to the Pitch EG

1) INITIALIZE your DX7II from single voice play mode and leave it in algorithm #1. TURN OFF operators 2 through 6 ("100000").

2) Before we create our sound, let's just take a brief stopover at the pitch EG and hear the effect of velocity sensitivity as applied here to a single unmodulated carrier. Press edit switch 13 and change the pitch EG Range to 1 octave. Change L3 and L4 of the pitch EG both to values of 99. Change R3 to a new value of 35. These changes create an envelope which looks like **figure 11-23.** Leave the pitch EG Vel parameter at its initialization default of OFF. Play a key, hold it down, and listen (Audio Cue 69A). Note that the pitch climbs up half an octave, and that it does so regardless of how quickly or slowly you strike the key.

3) Position the cursor over the Vel parameter and press the "yes" button in order to turn this ON. Now play a key as slowly as possible and listen (Audio Cue 69B). Note that, depending on your "touch", there is little or no pitch change - and that if there is some, it occurs very quickly (since the pitch EG levels, and therefore the speed of change, is rescaled). Now play a key as quickly as possible and listen (Audio Cue 69C). Note that we now hear virtually the same pitch change - at the same speed - that we did in step 2 above (you won't hear *precisely* the same change unless you really slam down on the key with maximum brute force - which is *not* advised!). Play two notes an octave apart, but play the higher key significantly faster. Listen (Audio Cue 69D). Even though there is only one pitch EG, keyboard velocity sensitivity *is* polyphonic, so that *different keys can actually have different amounts of pitch change at different speeds!*

4) Now let's make a *real* sound - not just a sine wave - to take advantage of this phenomenon. Press edit switch 7 and select algorithm #5. **(see figure 11-24)** We'll create essentially the same timbre in each one of these systems, and use the detuning control to generate slow beating effects between them. Accordingly, TURN ON operators 2 through 6 ("111111"), and press edit switch 8, followed by edit switch 2, in order to view the pitch data input for operator 2. Next, change operator 2's ratio number to 14.00. Press edit switch 10 and change operator 2's output level to a value of 48.

5) Part of the beauty of this simple sound will be a long release time. Accordingly, press edit switch 9, followed by edit switch 1, and change R4 for operator 1 to a new value of 23. Use the EG and Scaling Copy function (see Chapter Nine if you don't remember how to do this) in order to copy this same EG to operators 3 and 5 (the other carriers in this algorithm). Next, press edit switch 2 in order to VIEW operator 2's EG values and change its R4 to a new value of 24 (offsetting

slightly). Use the EG and Scaling Copy function to copy this same EG to operators 4 and 6 (the other modulators in this algorithm).

6) The EG Scaling and Copy function also copies operator output level (so as to ensure that the EGs really are the same from operator to operator), so we won't need to readjust the output levels of operators 3 through 6, but it doesn't copy any pitch data information, so press edit switch 8 and change the ratio number for operators 4 and 6 (both modulators) to 14.00 as well.

7) Now we'll add the icing on the cake - some detuning. Leave operators 1 and 2 at their default detuning values of 0, but change the values for the other four operators as follows: operator 3 = +5; operator 4 = +3; operator 5 = -4; operator 6 = +2.

8) We're nearly ready to give it a try, but first, let's spread out the pitch EG values just a bit. Press edit switch 13 and change both R2 and R3 to nice slow values of 5. Also change R4 to 0, so that the pitch change will continue very slowly even after "key off".. Leave the Vel parameter ON. Play a note very gently and listen (Audio Cue 69E). Now play the same note quickly and listen (Audio Cue 69F). Finally, play a series of notes and chords, with some played more quickly than others and listen (Audio Cue 69G). The polyphonic ability of keyboard velocity sensitivity allows us to generate subtle pitch-bend effects, with fingertip control! The fact that this timbre has an ephemeral, shimmering quality (due to the frequency ratios of 14 : 1, the large amounts of detuning, and the very slow R4 values in the operator EGs) adds to the overall effect. experiment by altering this sound to your hearts content before naming it and storing it somewhere in memory. You might also try applying some velocity sensitivity to one or more of the operators via edit switch 11 in order to add yet more fingertip control.

Once you feel conversant with EG bias modulation and the application of keyboard velocity sensitivity to both individual operators and the pitch EG, turn the page and let's move on to the DX7II's *geographic* output level control, *keyboard level scaling*.

Chapter Twelve ◇
Keyboard Level Scaling

The DX7II provides us with many tools which allow the control of operator output level. We've already explored several of these: the EGs (which change output level aperiodically over time), the LFO (which, when used for amplitude modulation, changes output level periodically over time), EG bias modulation (which allows real-time control of output level), and keyboard velocity sensitivity (which allows timed keyboard control over output level). In addition to these, a *geographic* output level control is also provided, and this is known as *keyboard level scaling*.

In this instance, the term "geographic" does not refer to whether you are playing your DX7II in Tibet or in Brooklyn, but instead refers to *which notes* you are playing on the keyboard. By the use of this control, we will be able to decrease or *increase* the output level of any particular operator by simply playing different notes on the keyboard!

And, yes, you did read correctly: I did say "increase". This is in fact the *only* output level control in the DX7II which will allow you to increase an operator's output level beyond its maximum set level (as determined by the Level parameter in edit switch 10). However, in no event will you be able to ever increase it beyond a maximum value of 99. Because we can independently scale each operator, Cardinal Rules one and two tell us that we will be able to alter the *volume* or *timbre* of our total sound as we play notes in different parts of the keyboard. This is a powerful control, indeed!

The DX7II actually provides us with two different types of keyboard level scaling: *normal* and *fractional* . The so-called "normal" mode (called that only because it was the only type of keyboard level scaling available on the original DX7) uses a system of curves* by which individual operator's output levels are altered over different areas of the keyboard. "Fractional" level scaling, a feature first presented in this model of the "X" synthesizers, simply allows you to designate a percentage of the maximum operator output level per group of three semitones. As usual, brief descriptions of these processes are often more confusing than the processes themselves, so let's examine each of these two *scaling modes* in detail.

As we mentioned, all of these keyboard level scaling functions are accessed from edit switch 10. This switch offers two different displays (and, as usual, you flip between them by repeatedly pressing the

* These are actually not true curves since they alter output level only every three half-steps. The end result, however, is that the output level seems to change in a smooth manner over that area of the keyboard. We'll talk more about this shortly when we run an exercise.

switch). The LCD display you will see when you first press this switch will usually be like **figure 12-1.**

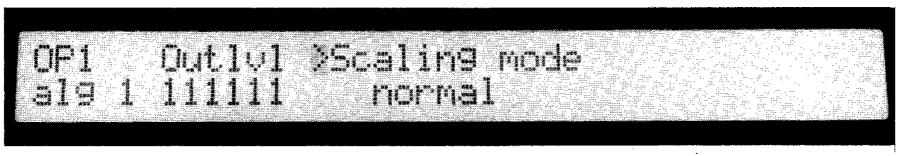


Figure 12-1

Because edit switch 10 is operator-specific (as is, of course, the entire keyboard level scaling function), you see the usual operator number, algorithm number, and operator on-off status display in the left-hand side of the LCD. The only adjustable parameter available in this display, however, is the "Scaling mode". This allows you to choose between "normal" and "fractional". A quick press of your "yes-no" switches will confirm that these are your only two options here. It's therefore logical that the DX7II will ask you which mode you want when you first press this edit switch, since the next LCD display (accessed by pressing edit switch 10 a second time) is different for each mode. Here is the display you'll get next if you've selected "normal" mode **(see figure 12-2)** and here is the display you'll see if you've selected "fractional" mode. **(see figure 12-3)**

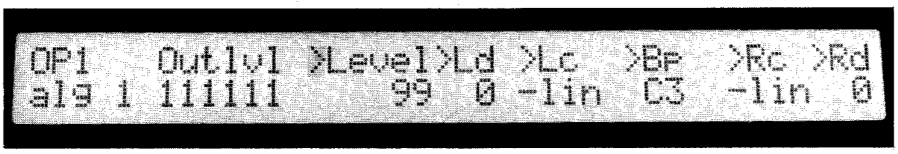


Figure 12-2

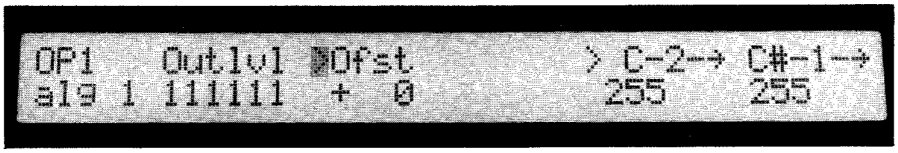


Figure 12-3

We'll start with a discussion of the initialization default: *normal* mode.

Normal level scaling

This mode should be somewhat familiar to you if you ever worked with the original DX7 - since it was in fact the only type of keyboard level scaling available on that instrument. The "mechanics" of how normal keyboard level scaling works are actually fairly straightforward. Let's go through them now, parameter by parameter.

Break point

We begin this process by specifying for each operator a note on the keyboard which is called the *Break point* , or "Bp" for short. We will then be able to either increase or decrease that operator's output level as we play notes above the break point, or below it. **(see figure 12-4)**

The break point can be any note on the keyboard, or, surprisingly, a note that doesn't even exist on the DX7II keyboard! What do we mean by this? To explain, let's pose another interesting question: How does the microprocessor in this instrument know one note from another? -

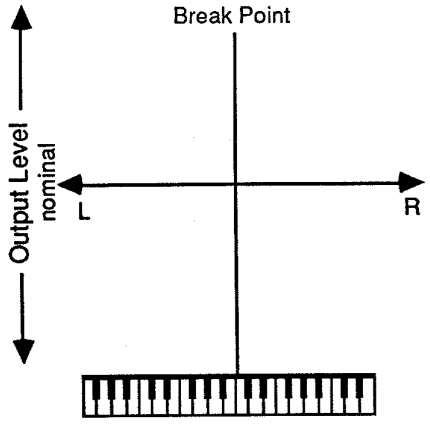


Figure 12-4

can it differentiate a C-sharp from a B-flat? It can't, at least not in the way we can - computers, after all, can only think in terms of numbers. Therefore, all of the keys on the DX7II keyboard are numbered. The lowest actual note is numbered C1 and the highest is called C6. **(see figure 12-5)**

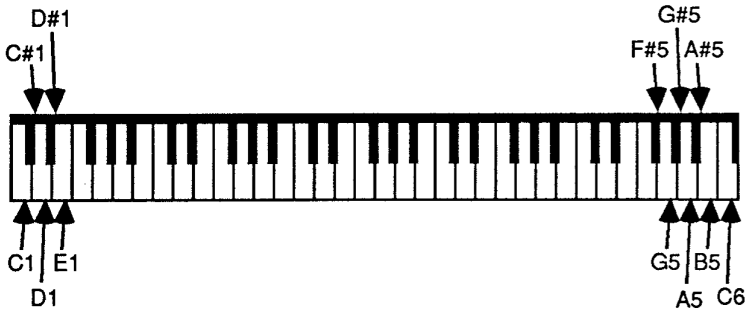


Figure 12-5

But the DX7II keyboard is only a five-octave keyboard. Through the use of the MIDI interface (see Chapter Fifteen for a complete introduction to MIDI), we can control the DX7II from an external keyboard - even one with a full 88 keys (or more!). How could we do this if our DX7II microprocessor only recognized 61 keys? We couldn't. And that's why it's been "trained" by Yamaha to recognize an additional three and a half octaves worth of notes. This ever gullible computer "believes" that there's an extra octave and a half below C1, and these notes are numbered A-1 to B0, **(see figure 12-6)** and it also believes that there are an extra two full octaves above C6, from C#6 to C8. **(see figure 12-7)**

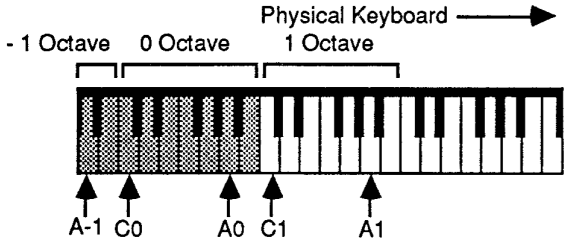


Figure 12-6

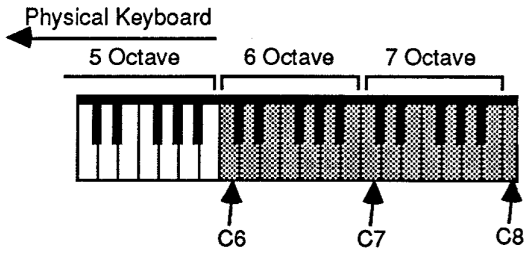


Figure 12-7

It also doesn't seem to realize (or care) that these keys aren't ever depressed - as I told you before, computers are very fast but very, very stupid.

We set the break point by calling up the "normal" LCD display from edit switch 10, and positioning the cursor over the "Bp" parameter (third from the right). Initialize your DX7II and call up this display in order to view the default values. Note that the initialization Bp default is C3 (Middle C) for all six operators. Use the data entry slider to step through all of the different possible break points, from A-1 all the way up to C8. We will see shortly why we might want to set break points that are off the scale of the physical DX7II keyboard. Remember again that each operator can have its *own* individual break point.

This will be a good time to digress briefly and talk about a function we referred to briefly earlier in this book: *key transpose.*

Key transpose

This parameter, accessed from the non-operator-specific edit switch 7 **(see figure 12-8)** allows you to theoretically "shift" the actual DX7II keyboard over the theoretical A-1 to C8 keyboard that the microprocessor believes exists. **(see figure 12-9)**

Figure 12-8

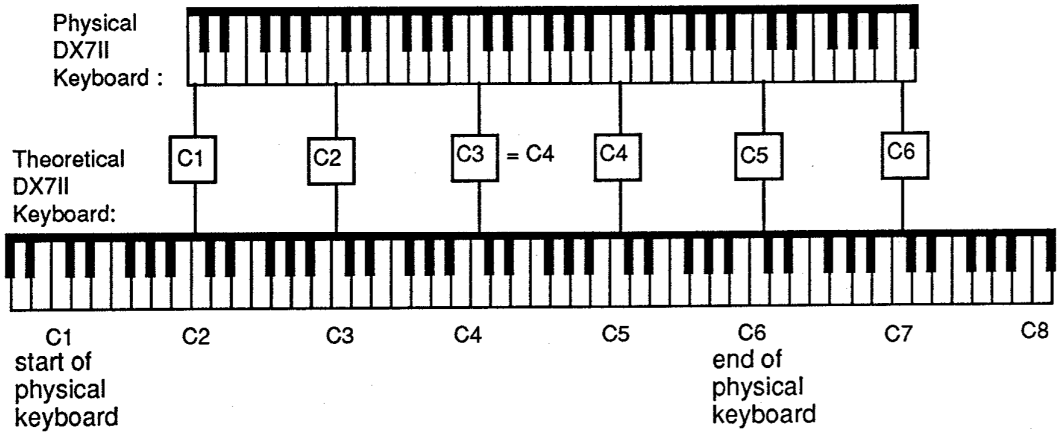


Figure 12-9

Press edit switch 7 in your instrument, call up this display, and position the cursor over the "Transpose" parameter. This parameter is slightly unusual in that, in addition to being able to change its value with the data entry slider or "yes-no" buttons, you can also enter this value in directly from the keyboard itself - but this works only when you first choose this parameter. At the moment, this display probably reads "midC=C3" (this may be different for some voices). This appears to be stating the obvious: yes, middle C is numbered C3. However, we can make middle C equal in fact to any other note on the physical keyboard, from C1 to C5 (notes above C5 will not be accepted). This is accomplished by simply playing the note you want middle C to be equal to. Try it: play D above C3. Your display should now read like **figure 12-10.**

Figure 12-10

You should have noticed that the D3 you played made no sound. That's because the first note you play after selecting this parameter is assumed to be the note you in fact want to enter into this parameter, and the voice assignment system simply ignores it. If you now play another note or two, however, you'll notice that the LCD display no longer changes; it's only the *first* note you play after selecting this parameter that is entered in. You can also change this parameter as usual, with the data entry section. Move the slider and watch this cycle through its full range of C1 to C5.

Key transpose can be a very handy function, since it allows you to play any piece of music in any key at all - as long as you can play it in the easiest (C major) key of all! You'll appreciate this function the next time you're on stage working with a singer who tells you - spur of the moment, mind you - "drop it down a half-step, my throat's sore". You can watch (with amusement, if you've got a sadistic streak) as other, less digitized musicians in your ensemble struggle to remember how to play in the new key. All *you* have to do, on the other hand, is call up the Transpose function, make middle C equal to B2 (a half-step below C3), and continue playing in the same key you've rehearsed in all along. The DX7II will take care of the transposition for you!

This same function can also create problems. Because Transpose is a voice editing parameter, it will be remembered when you store the voice, so if you've made middle C equivalent to anything other than another C, you will end up with a voice that's always out of tune with all of your other voices! If you store sounds with middle C equal to anything other than C, then, my advice is that you indicate the change in the voice name (like calling it "FluteF" or "StringD"), so at least you know how it's been transposed.

The other potential problem has to do with the use of the keyboard level scaling control. This is because the Break point mentioned above is specifically referring to the physical keyboard (albeit an unreal 111-note physical keyboard), which may be shifted by the Transpose function so that the note numbers of this keyboard are not true note numbers. For example, if you have used the Transpose function to make Middle C equivalent to C4, then selecting a Break point of C3 means in fact that you have selected C4! We'll run an exercise shortly to clarify this - hang in there until then.

Curves

As mentioned above, the point of normal keyboard level scaling is to attenuate or boost the output level of specific operators as we play notes above or below the selected break point. If we graph out the output level change initiated by keyboard level scaling, it looks like **figure 12-11.**

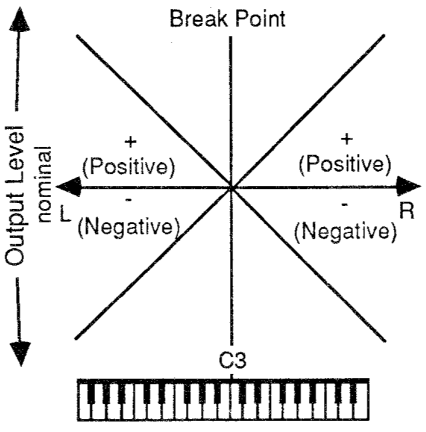


Figure 12-11

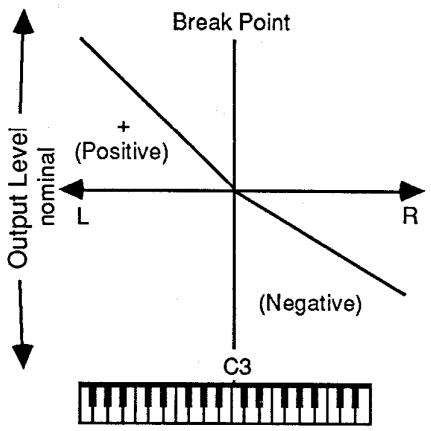


Figure 12-12

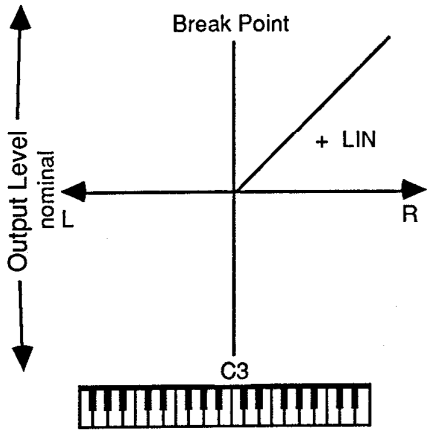


Figure 12-13

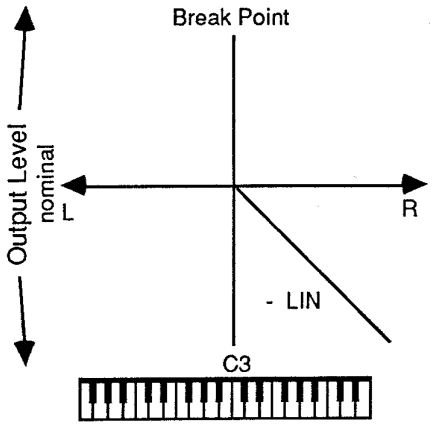


Figure 12-14

If we choose to *increase* an operator's output level above or below the break point, then we are said to be applying a *positive* curve; and if we choose to *decrease* the output level, we are said to be applying a *negative* curve. We can choose two different curves: one to the left of our break point, and another (potentially different) one to the right. **(see figure 12-12)**

Both positive and negative curves can have one of two different shapes. The type of curve we are currently displaying in this diagram is called a *linear* curve. When you use a linear curve, the change will be either additive (if you are using a *positive* linear curve) or subtractive (if you are using a *negative* linear curve). In plain English, a positive linear curve to the right of our break point will result in a greater (and smoothly changing) operator output level as we work our way up the keyboard **(see figure 12-13)** whereas a negative linear curve to the right of our breakpoint would result in a lesser (and smoothly changing) output level as we work our way up the keyboard. **(see figure 12-14)**

The second kind of curve available to us is called an *exponential* curve. With this type of curve, the microprocessor performs multiplication (for a positive exponential curve) or division (for a negative exponential curve) operations. When you multiply two positive numbers together over and over again, the results soon become astronomical:

1) 2 x 2 = 4
2) 4 x 2 = 8
3) 8 x 2 = 16
4) 16 x 2 = 32
5) 32 x 2 = 64
6) 64 x 2 = 128
7) 128 x 2 = 256
8) 356 x 2 = 512
9) 712 x 2 = 1024
10) 1424 x 2 = 2048

as opposed to a simple additive (linear) operation:

1) 2 + 2 = 4
2) 4 + 2 = 6
3) 6 + 2 = 8
4) 8 + 2 = 10
5) 10 + 2 = 12
6) 12 + 2 = 14
7) 14 + 2 = 16
8) 16 + 2 = 18
9) 18 + 2 = 20
10) 20 + 2 = 22

Therefore, the positive exponential curve will result in a much more drastic eventual increase in output level. Since we can't ever increase the output level above 99, getting to these astronomical numbers won't help us much. Instead, the computer "squeezes" the curve down by dividing the result of this operation by some fixed value, and the end result is a curve which undergoes very little change at all for some period (about two octaves on the keyboard) and then begins a very rapid increase - an increase that accelerates as you get further and further from the break point. **(see figure 12-15)**

Similarly, a series of division operations:
1) 100 / 2 = 50
2) 50 / 2 = 25
3) 25 / 2 = 12.5
4) 12.5 / 2 = 6.25
5) 6.25 / 2 = 3.125
yields a much more severe decrease in result than does a series of subtraction operations:
1) 100 - 2 = 98
2) 98 - 2 = 96
3) 96 - 2 = 94
4) 94 - 2 = 92
5) 92 - 2 = 90
So that, once again, our "squeezed" negative exponential curve yields very little change for the same two octaves, and then a sharper decrease in output level as you get further and further away from the break point. **(see figure 12-16)**

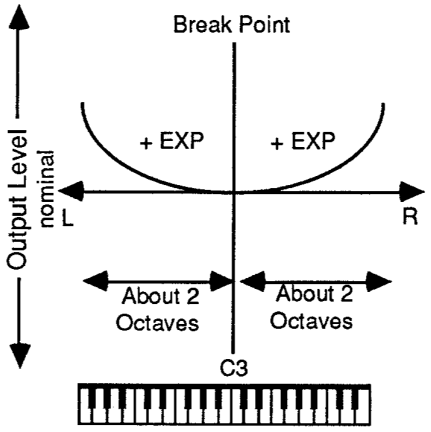


Figure 12-15

Let's summarize. The four different curves available to us are as follows:
1) *Negative linear* - each note will have slightly less output level than the preceding one.
2) *Negative exponential* - very little change at all for about two octaves, then a more drastic decrease in output level for each note played.
3) *Positive exponential* - very little change at all for about two octaves, then a more drastic increase in output level for each note played.
4) *Positive linear* - each note will have slightly greater output level than the preceding one.

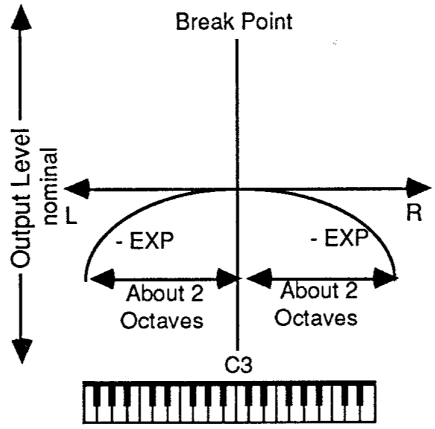


Figure 12-16

Since Yamaha doesn't expect you to memorize the effects of these curves (any more than they expect you to memorize the effects of the various EG levels and rates), there is a convenient diagram of these curves located on the front panel of the DX7II itself, on the far right-hand corner. Use it as a quick reference guide when working with keyboard level scalings.

Initialize your instrument. Press edit switch 10 twice to call up the "normal" mode LCD display. Observe that there are two parameters labeled "Lc" and "Rc". **(see figure 12-17)**

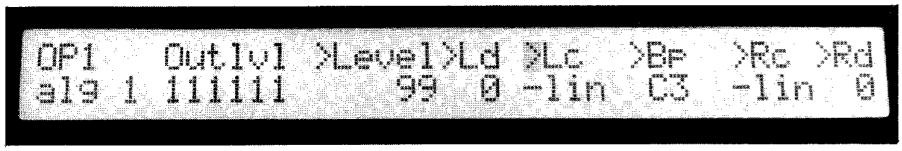


Figure 12-17

These stand for "Left curve" and "Right curve". Position the cursor over the "Lc" parameter and note that the initialization default is the negative linear ("-lin") curve. Use the "yes" button in the data entry section to scroll through the four different curve options. Press the right cursor switch twice in order to position the cursor over the "Rc" parameter, and note that it has the same default -lin curve. Whenever you initialize, both the right and left curves for all six operators will default to a negative linear curve.

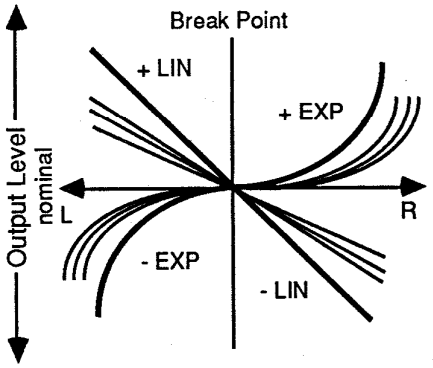


Figure 12-18

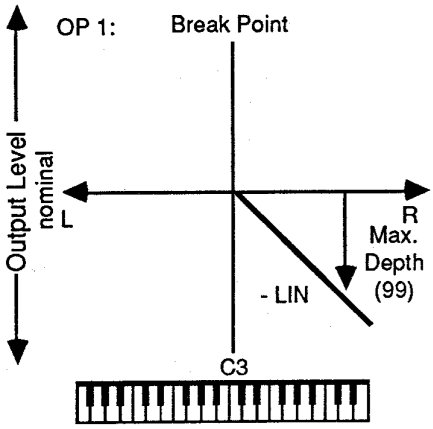


Figure 12-19

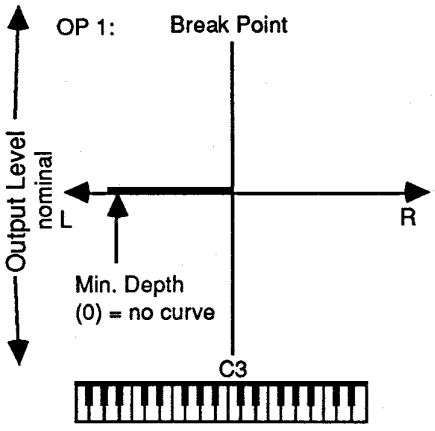


Figure 12-22

Depth

Finally, we will need to determine the *depth* of this scaling effect for each operator. This is accomplished by entering in a value for the "Ld" (Left depth) and "Rd" (Right depth) parameters in this same LCD display. Having selected a break point and a curve to the right and left of that point, you must now specify to the DX7II how much effect you wish that curve to have on the operator's output. The range of this control is 0 to 99, with 0 indicating no depth at all. In other words, no matter what curve you select, if you give it a depth value of 0, it will be as if there is no curve. On the other hand, a depth value of 99 will indicate that the curve has maximum effect. On our graph, we can represent the various depths as the steepness of the angle of the curve, as in **figure 12-18.**

As you might expect, the initialization default for this parameter is 0 for all curves, for all operators. In other words, when you initialize, there is no keyboard level scaling occurring at all. Let's run an Exercise now to hear the effect of this control on a carrier in a single-system voice:

Exercise 70

Normal level scaling applied to a carrier

1) INITIALIZE your DX7II from single voice play mode, leave it in algorithm #1, TURN OFF operators 3 through 6 ("110000"), and, using the system of operators 1 and 2, GENERATE a sawtooth wave.

2) Press edit switch 7, and observe that the Transpose function is at its initialization default - in other words, it shows that middle C is equal to C3. Do NOT change this value.

3) Press edit switch 10, followed by edit switch 1, repeatedly if necessary in order to VIEW the Scaling mode display for operator 1 (the carrier in this system). **(see figure 12-19)** Note that the Scaling mode is currently at its default of "normal". Leave it that way, and press edit switch 10 a second time in order to call up the normal mode output level display. **(see figure 12-20)** Observe that all the level scaling values are currently at their initialization defaults: Level = 99; Ld = 0; Lc = -lin; Bp = C3; Rc = -lin; and Rd = 0. Use the cursor switches to position the cursor over the Rd (Right depth) parameter) and use the data entry slider to change it to the maximum value of 99.

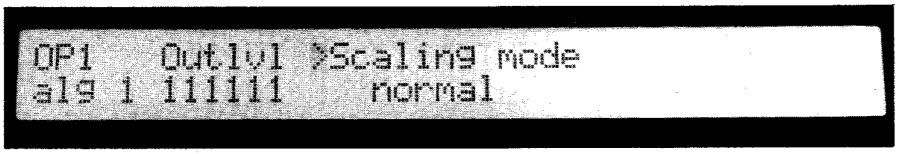


Figure 12-20

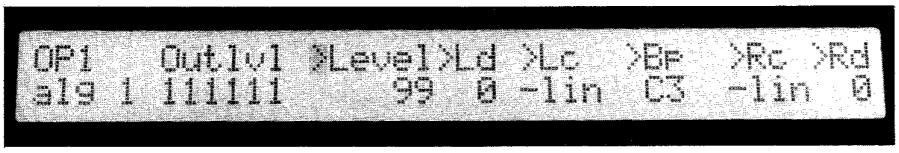


Figure 12-21

4) Play a chromatic scale (all notes) on your keyboard, starting at Middle C (C3) and going up to the highest note (C6), listening as you do so (Audio Cue 70A). Note that because we have applied a negative

linear ("-lin") curve to the right of C3, with maximum depth (since Rd now equals 99), the volume of our sawtooth wave decreases rather rapidly, disappearing altogether by the uppermost note (C6). **(see figure 12-21)**

5) Play a chromatic scale from C3 down to C1 and listen (Audio Cue 70B). Note that we now hear no volume change whatsoever as we go to the left of C3 because the Ld (Left depth) value is still at its default of 0. **(see figure 12-22)**

6) Change the Rd (Right depth) value for operator 1 to a new value of 55. Play a chromatic scale on your keyboard from C3 to C6, listening as you do so (Audio Cue 70C). Note that the volume again decreases, but not as rapidly, and that even at the uppermost note (C6), we still have a significant amount of volume. **(see figure 12-23)**

7) Press the left cursor switch four times in order to position the cursor over the Ld (Left depth) value, and use the data entry slider to change this to the maximum value of of 99. Play a chromatic scale on your keyboard, starting this time at the lowermost key (C1) all the way to the top key (C6), listening as you do so (Audio Cue 70D). Note that the volume starts low, gets louder with its maximum at C3 (in fact, your ears may tell you that A2 - which is three semitones lower than C3 - is the loudest note. This is because the change in volume actually begins at C3. For the sake of brevity, however, we won't keep repeating this - we'll pretend that a break point of C3 really means a break point of C3). The sound then gets softer again; effectively, this makes middle C the loudest note on the keyboard. **(see figure 12-24)**

8) Let's see how changing the Transpose parameter affects things. Press edit switch 7 and position the cursor over "Transpose". Play C above middle C (C4) on the keyboard in order to enter in this value, or use the data entry section in order to make "midC=C4". Play the same chromatic scale on the keyboard and listen (Audio Cue 70E). Note that the entire effect of level scaling has been shifted up a full octave, and that it is now C4 that is the softest note on the keyboard. Experiment by changing to various Transpose values and note that the C3 break point we are working with simply "shifts" to whatever note the Transpose parameter tells it to. You can go back to edit switch 10 and observe that the break point value in the display does *not* change, however. When you are done experimenting, restore the Transpose parameter back to the C3 default and press edit switch 10 again in order to return to the normal display.

9) Restore both the Rd and the Ld values for operator 1 back to their defaults of 0. This effectively removes the entire keyboard level scaling effect from this sound. Position the cursor over the Rc (Right curve) parameter and press the "yes" button once to change this for operator 1 from its default of -lin to a new value of -exp, giving us a negative exponential curve.

10) Press the right cursor switch once in order to position the cursor over the Rd parameter. Change this value for operator 1 only to its maximum of 99. Play a chromatic scale on your keyboard, from C3 to C6, and listen (Audio Cue 70F). Note that there is no apparent change in volume. Why should this be? Remember that the exponential curves induce virtually no change for a period of approximately two octaves. **(see figure 12-25)** You may have been able to detect a slight volume decrease from C5 to C6, but this would have been so subtle as to be nearly unnoticeable. However, if we shift the break point for operator 1 so that this curve begins at a lower key, we should be able to hear the full effect of this curve, so:

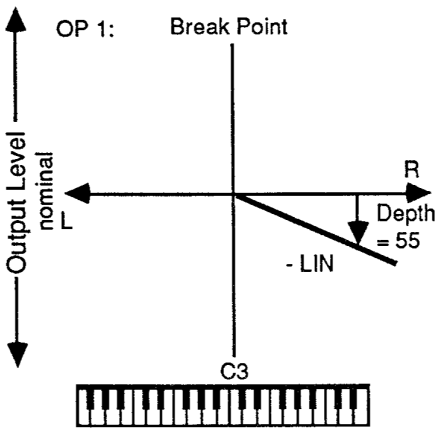


Figure 12-23

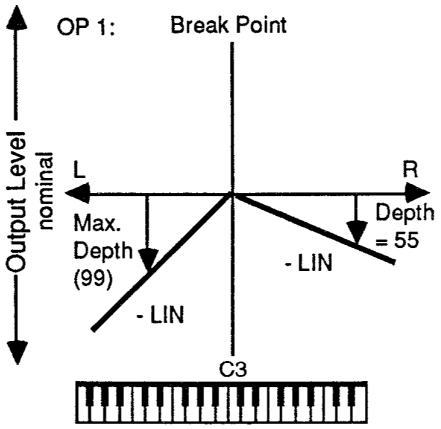


Figure 12-24

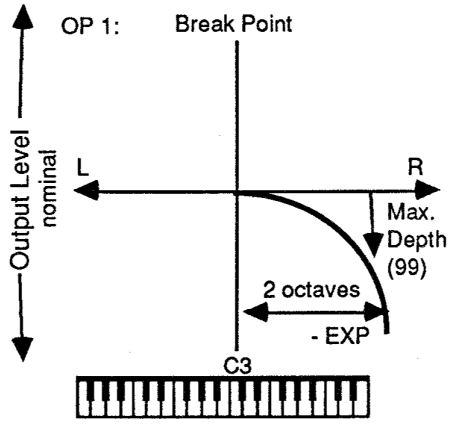


Figure 12-25

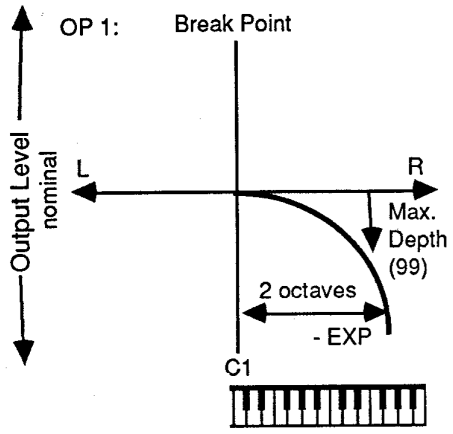


Figure 12-26

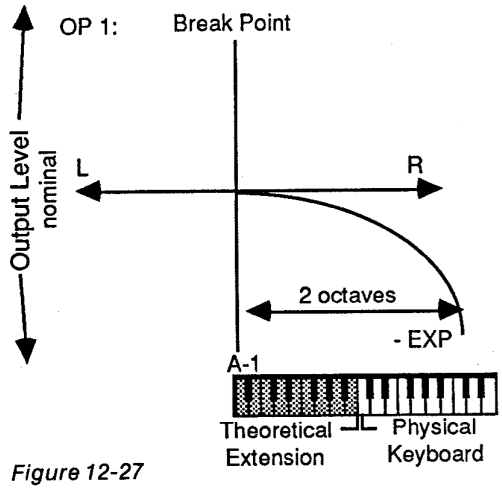


Figure 12-27

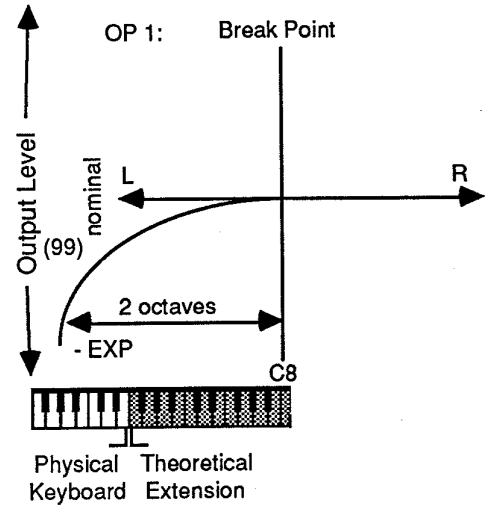


Figure 12-28

11) Press the left cursor switch twice in order to position the cursor over the Bp parameter and change this to a new value of C1 (the lowest physical note on the DX7II keyboard) for operator 1 only. Now play a chromatic scale up the length of the keyboard, starting at C1 all the way on up to C6, and listen (Audio Cue 70G). Note that we hear virtually no volume change at all from C1 to C3, a gradual decrease in volume from C#3 to about C5, and then a drastic decrease in volume from C#5 on up to C6: **(see figure 12-26)** Note that, even then, the volume does not disappear completely even at C6. In general the exponential curve will yield a more subtle effect than the linear curve.

12) If we set the break point, however, at the very lowest note possible (A-1) - right off the scale of the physical DX7II keyboard - then this sound will undergo most of its two-octave no-change period before the lowest physical note, and we'll be able to hear its effect as of the lowest actual note we play. **(see figure 12-27)** This is the main reason why the DX7II allows you to set break points right off the scale of the physical keyboard. Change the Bp for operator 1 only to the minimum value of A-1. Play a chromatic scale on the keyboard from C1 to C6 and listen (Audio Cue 70H). Note that we are now able to hear a discernable decrease in volume from nearly the very first note played, and that this change accelerates as we go higher up the keyboard. The volume is now completely gone by about D5.

13) We can reverse this effect completely by making the following changes for operator 1 only: Set the Bp to C8. Set the Lc to "-exp". Set the Ld to 99. Play a chromatic scale from the uppermost note (C6) down to the lowest note (C1) and listen (Audio Cue 70I). The effect we have created is precisely as follows. **(see figure 12-28)**

14) Now let's see how positive curves work. Restore the Bp for operator 1 only to its initialization default of C3. Restore both the Ld and the Rd values to 0, which will have the effect of making the curves inactive. Now change the Rc to a "+lin" curve by moving the data entry slider to its uppermost position (or by pressing the "yes" button), and change the Rd back to the maximum value of 99. Play a chromatic scale on the keyboard from C3 to C6 and listen (Audio Cue 70J). Note that there is currently *no* volume change. Why should this be? - especially since we've established that keyboard level scaling, unlike any of our previous output level controls, *can* in fact be used to increase an operator's output level beyond its set value? The answer lies in the fact that we can never, under any circumstances, increase that output level beyond the maximum value of 99. And that's exactly where operator 1 is already. **(see figure 12-29)** We have simply left ourselves no headroom. Position the cursor over the Level parameter and change this for operator 1 only to a new value of 50. Now play the chromatic scale again, from C3 to C6 , and listen (Audio Cue 70K). Note that we now hear the volume increase as we play higher notes on the keyboard.

15) Press the right cursor switch five times in order to position the cursor over the Rd parameter for operator 1, and change this to a new value of 75. Play a chromatic scale from C3 to C6 and listen (Audio Cue 70L). Note that we hear the same increase in loudness, but less drastically so.

16) Play a chromatic scale from C3 down to C1 and listen (Audio Cue 70M). Note that there is no volume change in this part of the keyboard, and that all the notes are at the lower output level. This is because there is currently no keyboard scaling to the left of the break point (since Ld = 0). **(see figure 12-30)**

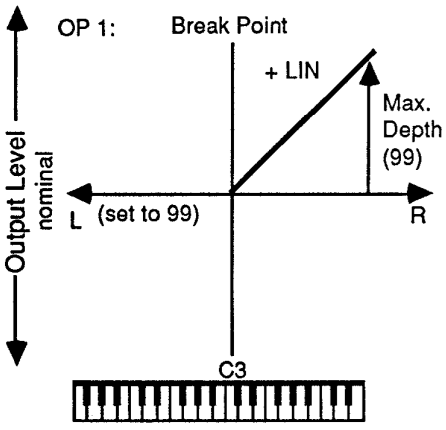


Figure 12-29

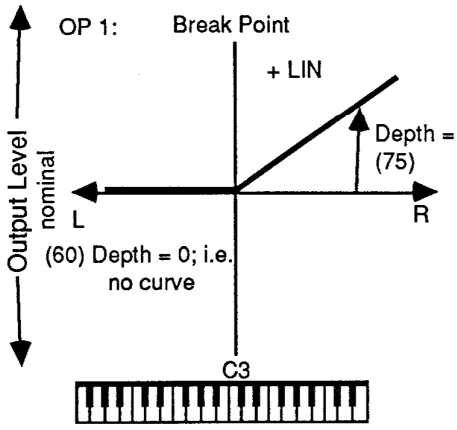


Figure 12-30

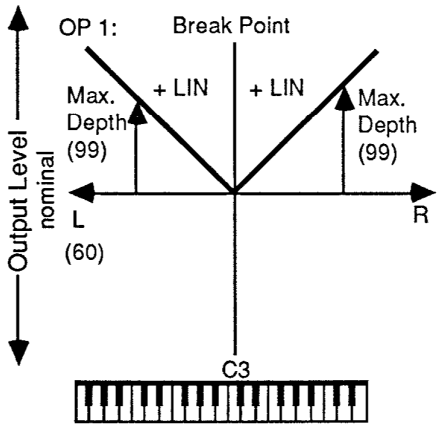


Figure 12-31

17) Position the cursor over the Lc parameter and change this for operator 1 only to "+lin". Now press the left cursor switch once and change the Ld to the maximum value of 99. Press the right cursor switch four times and restore the Rd back to the maximum value of 99. Play a chromatic scale on the keyboard from C1 all the way up to C6 and listen (Audio Cue 70N). Note that Middle C (C3) is now the quietist note on the keyboard. **(see figure 12-31)**

18) Press the left cursor switch twice and change the Bp for operator 1 only to the lowest possible value of A-1. Press the right cursor switch once and change the Rc to "+exp". Play a chromatic scale up the entire keyboard, from C1 to C6 and listen (Audio Cue 70O). Note that the volume now steadily increases from A1 to C5, with the greatest increase occurring in the range C3 to C5. Above C5, note that there is no discernible volume increase, because the output level has already reached its absolute maximum of 99 by C5. **(see figure 12-32)**

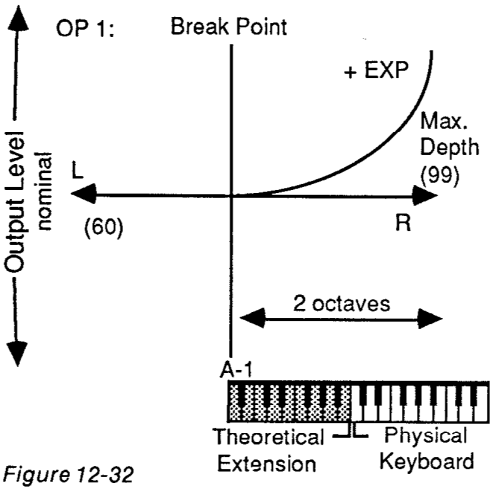


Figure 12-32

19) Let's reverse that effect by doing the following: Press the left cursor switch once and change the Bp for operator 1 only to a new value of C8. Press the left cursor switch once and change the Lc value to "+exp". The Ld should still be at 99, since that is where we set it in step 16 above. Play a chromatic scale from C6 down to C1 and listen (Audio Cue 70P). Note that the volume now steadily increases as you go down the keyboard from C6 to F2, with the greatest increase occurring in the range F4 to F2. Below F2, note that there is no discernible volume increase, because the output level of operator 1 is already at its maximum of 99 by that point. **(see figure 12-33)**

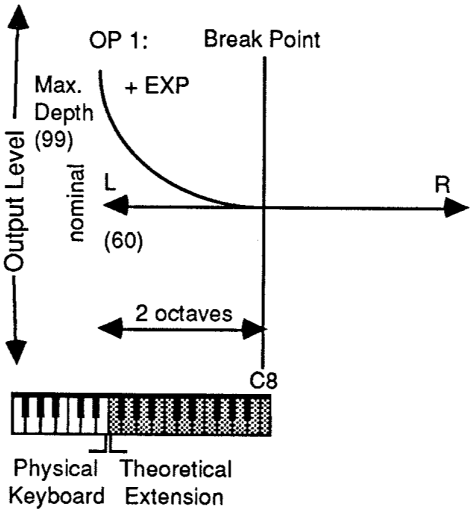


Figure 12-33

20) We can also combine positive and negative curves for the same operator, if we so desire. Change the Bp for operator 1 only to a new value of C2. Press the left cursor switch three times and change operator 1's output level (the Level parameter) to a new value of 64. Press the right cursor switch twice and set the Lc to "-lin". Press the right cursor switch twice and change the Rc to "+exp". Both the Ld and the Rd should still be at the maximum of 99. Play a chromatic scale up the keyboard from C1 to C6 and listen (Audio Cue 70Q). Note that the volume rises from near-inaudible to a low level by the time you reach C2, and then remains at that low level until about C4, at which point it begins increasing, gently at first, and then more dramatically at the upper end of the keyboard. **(see figure 12-34)**

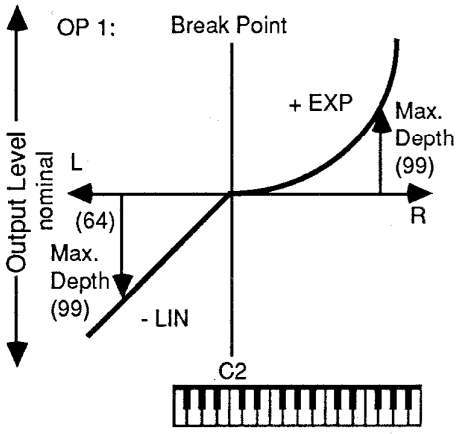


Figure 12-34

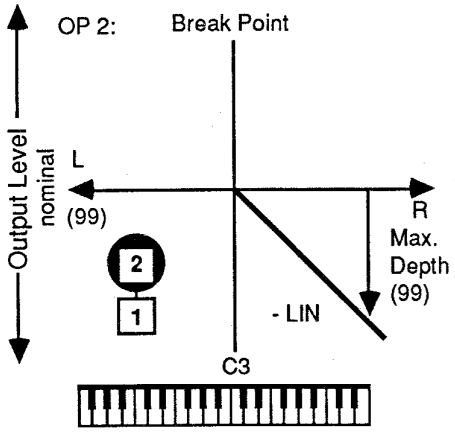


Figure 12-35

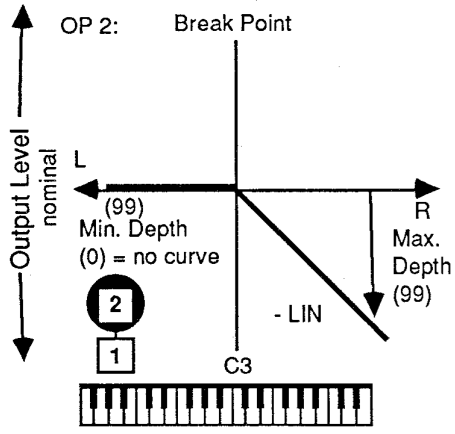


Figure 12-36

21) Experiment by changing the depth values (Ld and Rd) for either or both of the curves and note how this affects the overall sound. Experiment further with different timbres, using different break points, curves, depths, and transpositions, until you feel comfortable with the way keyboard level scaling affects carriers.

We can apply keyboard level scaling effects to a modulator as well. This will allow us to quantitatively change the overtone content of our sound, according to what note we play on the keyboard:

Exercise 71
Normal level scaling applied to a modulator

1) INITIALIZE your DX7II from single voice play mode, leave it in algorithm #1, TURN OFF operators 3 through 6 ("110000"), and, using the system of operators 1 and 2, GENERATE a sawtooth wave.

2) Press edit switch 10, followed by edit switch 2, in order to VIEW the Scaling mode display for operator 2. Note that it is currently at its default of "normal". Leave it this way and press edit switch 10 a second time in order to VIEW the normal level display. Note that all the scaling values are currently at their initialization defaults: Ld = 0; Lc = "-lin"; Bp = C3; Rc = "-lin"; and Rd = 0.

3) Position the cursor over the Rd parameter and change it for operator 2 only to the maximum value of 99.

4) Play a chromatic scale on your keyboard, starting at Middle C (C3) and going up to the highest note (C6), listening as you do so (Audio Cue 71A). Note that because we have applied a negative linear (-lin) curve to the right of C3, with maximum depth (99), the overtone content decreases rather rapidly, changing the sawtooth wave we are hearing completely back to a pure sine wave by the uppermost note (C6). **(see figure 12-35)**

5) Play a chromatic scale from C3 down to C1 and listen (Audio Cue 71B). Note that we hear no timbral change whatsoever as we go to the left of C3 because the Ld value is currently 0. **(see figure 12-36)**

6) Change the Rd for operator 2 to a new value of 55. Play a chromatic scale on your keyboard, from C3 to C6, listening as you do so (Audio Cue 71C). Note that the overtone content again diminishes, but not as rapidly, and that even at the uppermost note (C6), we still hear some overtones; we never quite revert all the way back to the sine wave. **(see figure 12-37)**

7) Press the left cursor switch four times in order to position the cursor over the Ld parameter and change this for operator 2 only to a the maximum value of 99. Play a chromatic scale on your keyboard, starting this time at the lowermost key (C1) all the way up to the top key (C6), listening as you do so (Audio Cue 71D). Note that C1 is virtually a sine wave and that the overtone content steadily increases until the break point of C3 (actually A2, as noted earlier); it then begins diminishing again, until C6. Because of the combination of these two curves, C3 (or, more accurately A2) is now the brightest note on the keyboard. **(see figure 12-38)**

8) Once again, let's see how changing the Transpose parameter affects things. Press edit switch 7 and position the cursor over "Transpose". Play C above middle C (C4) on the keyboard in order to enter in this value, or use the data entry section in order to make "midC=C4". Play the same chromatic scale on the keyboard and listen (Audio Cue 71E). Note that the entire effect of level scaling has been

shifted up a full octave, and that it is now C4 that is the least bright note on the keyboard. Experiment by changing to various Transpose values and note that the C3 break point we are working with simply "shifts" to whatever note the Transpose parameter tells it to. You can go back to edit switch 10 and observe that the break point value in the display does *not* change, however. When you are done experimenting, restore the Transpose parameter back to the C3 default and press edit switch 10 again in order to return to the normal display.

9) Restore both the Rd and the Ld for for operator 2 only back to their defaults of 0. This effectively removes the entire keyboard level scaling effect from our sound. Position the cursor over the Rc parameter, which is currently still at its default of "-lin". Press the "yes" button once to change this for operator 2 only to a new value of "-exp", giving us a negative exponential curve.

10) Press the right cursor switch once in order to position the cursor over the Rd parameter. Change this value for operator 2 only to its maximum of 99. Play a chromatic scale on your keyboard, from C3 to C6, and listen (Audio Cue 71F). Note that there is no apparent timbral change. Again, the exponential curve induces virtually no change for a period of approximately two octaves. **(see figure 12-39)** You may have been able to detect a slight overtone content decrease from C5 to C6, but this would have been so subtle as to have been virtually unnoticeable. However, if we shift the break point for operator 2 so that this curve begins at a lower key, we should be able to hear the full effect of this curve, so:

11) Press the left cursor switch twice in order to position the cursor over the Bp parameter and change this to a new value of C1 (the lowest physical note on the DX7II keyboard) for operator 2 only. Now play a chromatic scale up the length of the keyboard, starting at C1 all the way on up to C6, and listen (Audio Cue 71G). Note that we hear virtually no timbral change at all from C1 to C3, a gradual decrease in overtone content from C#3 to about C5, and then a drastic decrease in overtones from C#5 on up to C6. **(see figure 12-40)** Note that, even then, the overtones do not disappear completely even at C6.

12) If we set our break point, however, at A-1, the lowest note possible - right off the scale of the physical DX7II keyboard - then it will undergo most of its two-octave no-change period before the lowest physical key, and we'll be able to hear its effect as of the lowest actual note we play. **(see figure 12-41)**

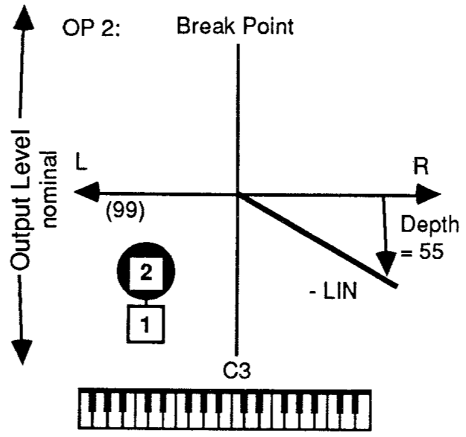


Figure 12-37

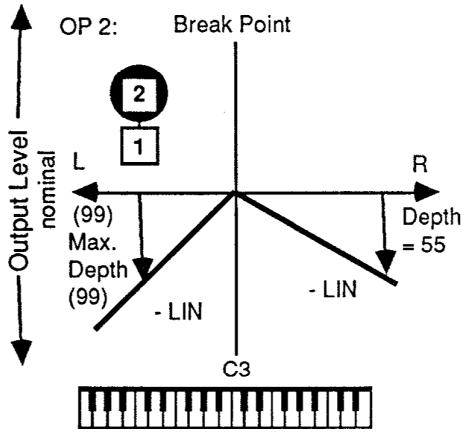


Figure 12-38

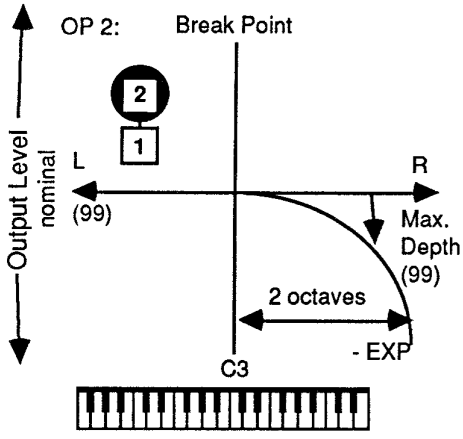


Figure 12-39

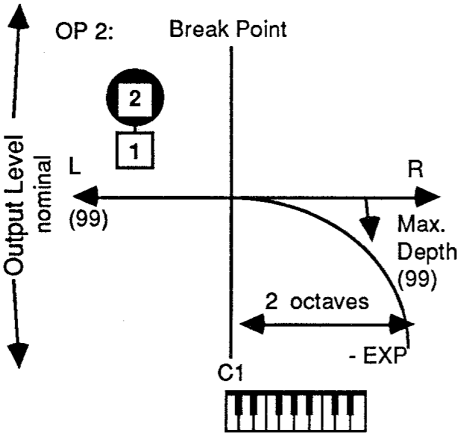


Figure 12-40

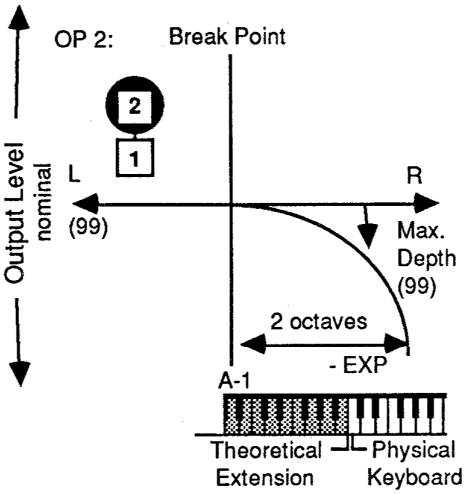


Figure 12-41

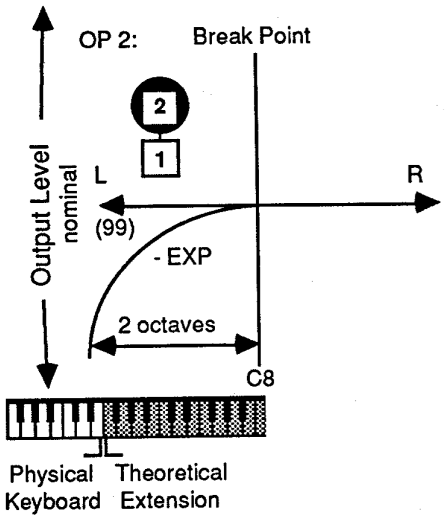


Figure 12-42

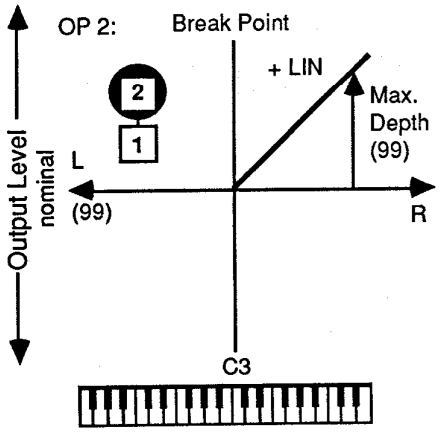


Figure 12-43

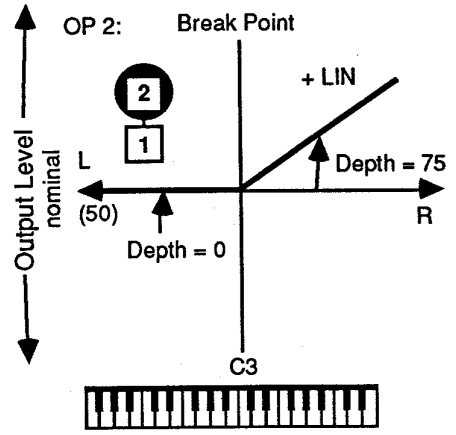


Figure 12-44

Therefore, change the break point for operator 2 only to a new value of A-1. Play a chromatic scale on the keyboard from C1 to C6 and listen (Audio Cue 71H). Note that we are now able to hear a discernable decrease in overtone content from nearly the very first note played, and that this change accelerates as we go higher up the keyboard. By the time we reach D5, we are back to a pure sine wave.

13) We can reverse this effect completely by making the following changes to the level scaling of operator 2: Set the Bp to C8. Set the Lc to "-exp". Set the Ld to 99. Play a chromatic scale from the uppermost note (C6) down to the lowest note (C1) and listen (Audio Cue 71I). The effect we have created is as shown in **figure 12-42.**

14) Now let's try a positive curve for our modulator. Restore the break point for operator 2 back to its initialization default of C3. Restore both the Ld and the Rd to 0, which will have the effect of making the curves inactive. Now change the Rc to a "+lin" curve and the Rd to 99. Play a chromatic scale on the keyboard from C3 to C6 and listen (Audio Cue 71J). Note that there is currently *no* timbral change whatever. Again, even though keyboard level scaling *can* be used to increase an operator's output level beyond its set value, we can still never increase that output level beyond the maximum value of 99. And that's exactly where operator 2 is already. (see **figure 12-43**) Once again, we have left ourselves no headroom. Position the cursor over the Level parameter and change this for operator 2 only to a new value of 50. Now play the chromatic scale again, from C3 to C6, and listen (Audio Cue 71K). Note that we now hear the overtone content increase as we play higher notes on the keyboard, so that C3 is little more than a sine wave, but C6 is a full-blown sawtooth wave.

15) Press the right cursor switch five times in order to position the cursor over the Rd parameter. Change this to a new value of 75. Play a chromatic scale from C3 to C6 and listen (Audio Cue 71L). Note that we hear the same increase in overtones, but less drastically so.

16) Play a chromatic scale from C3 down to C1 and listen (Audio Cue 71M). Note that there is no timbral change in this part of the keyboard, and that all the notes are quasi-sine waves. This is because there is currently no left depth (since Ld = 0). (see **figure 12-44**)

17) Press the left cursor switch three times in order to position the cursor over the Lc parameter and change this to a "+lin" curve. Press the left cursor switch once and change the Ld for operator 2 to the maximum value of 99. Press the right cursor switch four times and restore the Rd back to the maximum value of 99. Play a chromatic scale on the keyboard from C1 all the way up to C6 and listen (Audio Cue 71N). Note that C3 (actually A2) is now the least *bright* note on the keyboard. (see **figure 12-45**)

18) Press the left cursor switch twice and change the break point for operator 2 only to the lowest possible value of A-1. Press the right cursor switch once and change the Rc to "+exp". Play a chromatic scale up the entire keyboard, from C1 to C6 and listen (Audio Cue 71O). Note that the overtone content now steadily increases from A1 to C5, with the greatest increase occurring in the range C3 to C5. Above C5, note that there is no discernible overtone content increase, because the output level has already reached its absolute maximum of 99 by C5. (see **figure 12-46**)

19) Let's reverse that effect by doing the following: Press the left cursor switch once and change the break point for operator 2 to a new value of C8. Press the left cursor switch once more and change the Lc to "+exp". The Ld value should still be at 99 (where you set it in step 16 above). Play a chromatic scale from C6 down to C1 and listen (Audio Cue 71P). Note that the overtone content now steadily increases as you go down the keyboard from C6 to F2, with the greatest increase occurring in the range F4 to F2. Below F2, note that there is no discernible timbral change, because the output level of operator 2 is already at its maximum of 99 by that point. **(see figure 12-47)**

20) Of course, we can also combine positive and negative curves for the same operator (by having one type of curve to the left of our break point and a different type to the right). Change the break point for operator 2 only to C2. Press the left cursor switch three times and change operator 2's output level (Level parameter) to a new value of 64. Press the right cursor switch twice and set the Lc to "-lin". The Rc should still be "+exp" (where you set it in step 17 above). Both the right and left depths (Rd and Ld) should still be at the maximum of 99 (where they were set in step 16 above). Play a chromatic scale up the keyboard from C1 to C6 and listen (Audio Cue 71Q). Note that the timbre changes from a sine wave to a gentle, triangle-like wave by the time you reach C2, and then remains that way until about A3, at which point it begins increasing in overtone content, gently at first, and then more dramatically at the upper end of the keyboard. **(see figure 12-48)**

21) Experiment by changing the depth values (Ld and Rd) for either or both curves and note how this affects the overall sound. Experiment further with different timbres, using different break points, curves, depths, and transpositions, until you feel comfortable with the way keyboard level scaling affects modulators.

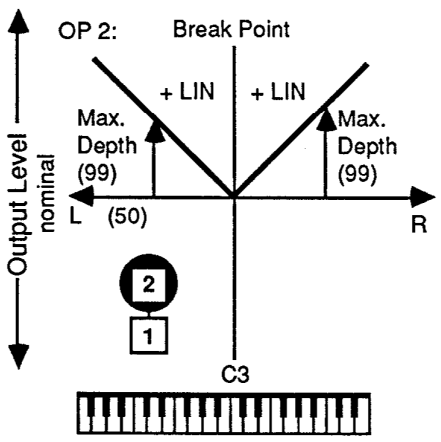


Figure 12-45

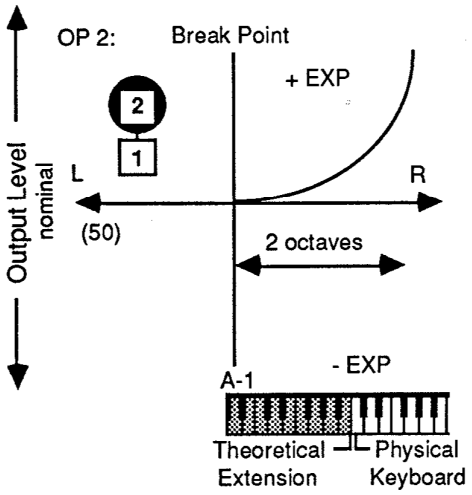


Figure 12-46

Because keyboard level scaling is operator-specific, and because each operator in our sound can have independent break points, curves, and depths, it is an enormously powerful tool in creating both "real" and "unreal" sounds. No acoustic instrument, for example, responds precisely the same way for each note you play on it: played with equal force, a low note on a piano is much louder than a high note; similarly, a high note on a trumpet is much brighter than a low note. We can simulate these and many more naturally occurring acoustic phenomena with keyboard level scaling, but we can also use it to create some very unusual effects. Take a look at a sound I programmed some time ago, a sound I refer to as my "Escher" sound, called "Playascale":

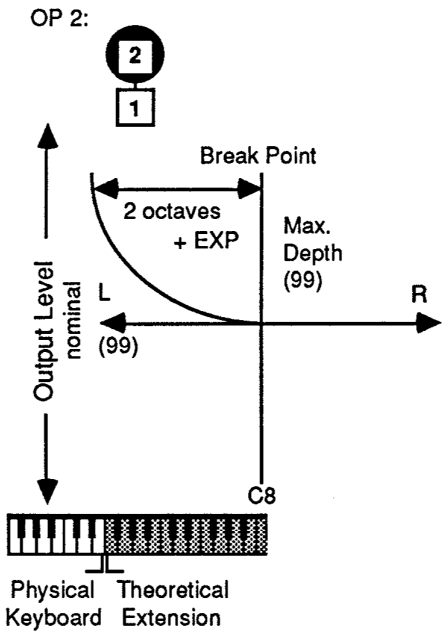


Figure 12-47

Exercise 72
Examining "Playascale": using keyboard level scaling for pitch change

1) INITIALIZE your DX7II from single voice play mode and put it into algorithm #32 (bet you never thought we'd go back to this old chestnut!).

2) Make sure all six operators are ON ("111111"). TURN OFF operator 6 ("111110") - for this sound, we will only be using operators 1 through 5. Even though our microprocessor will not be able to remember that operator 6 was off when you store this sound, its output level is currently defaulted at 0, so we won't hear it anyway.

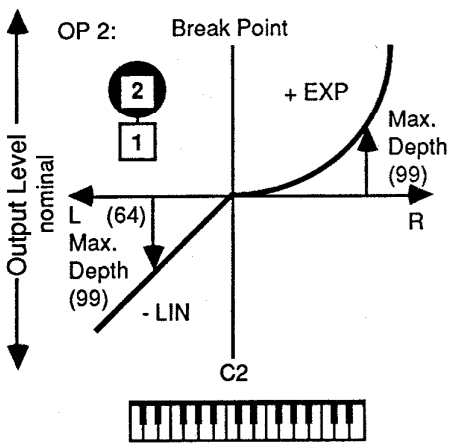


Figure 12-48

3) Press edit switch 10, followed by edit switch 2. Note that operator 2 (like all the other operators) is currently at its default of "normal" scaling mode. Leave it that way, and press edit switch 10 again. Change the output level (Level) of this operator to 99, and do the same for operators 3, 4, and 5. Because operator 1 defaulted to an output level of 99, we now have all five operators in use - operators 1 through 5 - all at the maximum output level value of 99.

4) Press edit switch 8 (OSCILLATOR) and enter in the following frequency Coarse values:

Operator #	F Coarse
1	16.00
2	8.00
3	4.00
4	2.00
5	1.00 (Rs value of 0 for all operators)

5) Press edit switch 10 again and enter in the following break points for modulators 1 through 5:

Operator #	Bp
1	G# 1
2	G# 2
3	G 3
4	G 4
5	C 6

6) Enter in the following Ld and Rd values:

Operator #	Ld	Rd
1	0	99
2	99	99
3	99	99
4	99	99
5	99	0

7) Note that all level scaling curves (Lc and Rc) should be left at their default "-lin" value. This process gives us a keyboard level scaling like in **figure 12-49.**

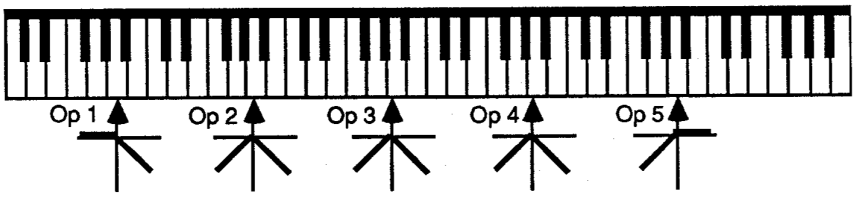


Figure 12-49

8) Now do just what the name of the sound tells you to do - play a scale - from the bottom of the keyboard to the top, and listen (Audio Cue 72A)! note the unusual effect: C1 sounds like the same note as C2, C3, C4, C5, or C6! In fact, each octave on the keyboard sounds pretty much the same as the octave below it or above it! This creates an interesting aural illusion, similar, I think, to the kind of visual illusions created by the great artist, M. C. Escher.

If you listen very closely to "Playascale", you can, in fact hear some slight differences in overtone content from octave to octave. But how can we have overtones when we're working with algorithm 32? Simple: the keyboard level scaling causes the differently pitched operators to overlap in various different ways over various parts of the keyboard. At

no time are we ever hearing purely one of the carrier sine waves to the exclusion of the others, due to this overlap. The end result is that we usually are hearing two or three differently pitched sine waves - an *additive* synthesis process. The illusion is not only that the pitches appear to be the same from octave to octave but that the sound we hear appears to contain overtones as well!

As with other original sounds presented in this book, I declare this patch to be in the public domain; and that means you're welcome to store it and use it at will. Another potential (and very powerful) use of keyboard level scaling is this: instead of scaling different carriers over different parts of the keyboard, we scale different modulators feeding a single carrier (as in a complex algorithm like #16), creating different *timbres* for each note we play:

Algorithm #16:

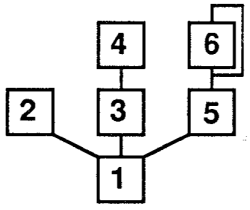


Figure 12-50

Exercise 73
Normal level scaling used for timbral change

1) INITIALIZE your DX7II from single voice play mode and select algorithm #16. **(see figure 12-50)**

2) Press edit switch 10 twice (all operators should be left in the "normal" scaling mode) and enter the following output levels (Level parameter) for the six operators:

Operator #	Level
1	99
2	99
3	71
4	99
5	55
6	99

3) Press edit switch 8 (OSCILLATOR) in order to enter the following frequencies (Coarse and Fine, where necessary) for the six operators:

Operator #	Coarse/Fine
1	1.00
2	1.00
3	2.00
4	11.20
5	14.00
6	15.00

4) Press edit switch 9 in order to enter the following EG data for the six operators:

Op #	L4	R1	L1	R2	L2	R3	L3	R4	L4
1	0	99	99	99	99	99	99	52	0
2	0	99	99	99	99	99	99	9	0
3	0	99	99	99	99	99	99	25	0
4	0	99	99	43	0	99	0	12	0
5	0	99	99	58	0	99	0	48	0
6	0	99	99	99	99	99	99	82	0

(Rs value of 0 for all operators)

5) Press edit switch 7 and enter in the maximum feedback value (Fbl parameter) of 7. Note that in this algorithm the feedback loop is on operator 6.

6) Press edit switch 10 and enter in the following break points for the six modulators:

Operator #	Bp
1	A -1
2	C 1
3	B 1
4	F# 3
5	B 4
6	D 5

7) Position the cursor over the Rc and press edit switch 3. Change the Rc for operator 3 only to "-exp". Leave all other operators at their default right and left curves of "-lin".

8) Enter in the following Ld and Rd values for the six operators:

Operator #	Ld	Rd
1	0	0
2	99	99
3	99	99
4	99	99
5	99	99
6	99	99

9) Play a chromatic scale from C1 to C6 and listen (Audio Cue 73A). Note that this sound undergoes six distinct timbral changes as the five different modulators fade in and out over the keyboard and finally disappear altogether from E5 to C6, as according to their keyboard scaling, as in **figure 12-51.** C1 is essentially just a sawtooth wave, containing a "square" envelope with a moderate release time. Because R4 for operator 2 is so slow (a value of 9), there is very little timbral change as the sound dies away. As we get to the area E1 - C2, we hear this sawtooth timbre change to a hollower square wave timbre, with a very similar type of envelope. The square wave is, of course, being generated by the action of operator 3, and because its right curve is a negative *exponential* curve, it will be present across a larger area of the keyboard than any of the other modulators. As we get towards the area C#2 - D4, we hear the steadily increasing presence of the upper modulator in this stack, operator 4. This operator has been set to a non-whole number frequency value and is at its maximum output level; furthermore, its envelope is quite percussive. The inharmonics it generates are most present at its break point of F#3 and they begin to tail off after this point. From D#4 to F#4, we begin to hear operator 5 fade in, with the characteristic 14:1 frequency ratio and slightly percussive envelope yielding a faint reminder of the classic DX7 "E.PIANO 1" Fender Rhodes preset. From G4 to E5, we hear the snare-drum like sound being contributed by the feedback modulator, operator 6. Finally, above E5, the actions of all the modulators die away and disappear, and at the upper end of the keyboard, we are left with just the single sine wave being generated by operator 1.

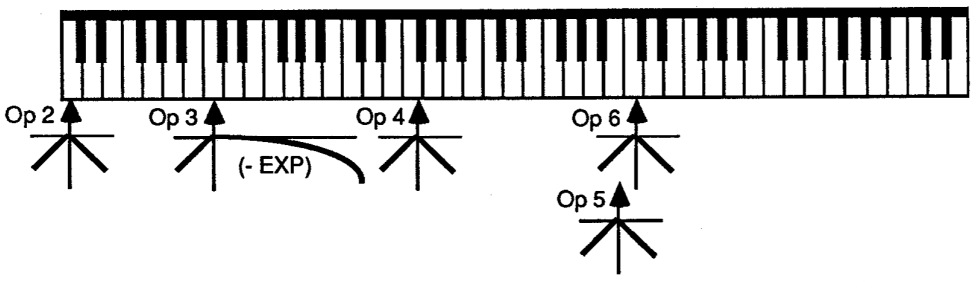


Figure 12-51

10) Experiment with altering the various scaling parameters for this sound - try, for example, reversing the entire process so that the single sine wave appears only at the bottom end of the keyboard. Experiment further by altering the various frequency ratios and operator output levels and note how these changes affect the entire sound.

Don't forget that any level scaling values you enter (in both normal and fractional modes) will be copied over when using the EG and Scaling Copy function (see Chapter Nine for more infomation about this). This ensures that the EGs will be precisely the same from note to note, since attenuating the output level of any operator over a portion of the keyboard with level scaling will also have the effect of rescaling that operator's EG levels - causing it to undergo its changes slightly more rapidly for those notes. It's important to remember that all of the DX7II editing functions are interactive - changing one parameter may actually be affecting several others! Be aware of these relationships as you are constructing or modifying your voices.

In summary, normal mode keyboard level scaling can be used for a variety of unique effects - volume change, timbral change, or even pitch change, over various areas of the keyboard. You can even use it to accomplish what are known as *keyboard splits,* although the two-sound capability and split play mode of the DX7II make this less of a need than in its *homophonous* (single-sound) predecessor, the DX7. This is accomplished simply by using a a multiple-carrier algorithm (that is, everything except #s 16, 17, and 18), setting up different sounds with the individual systems, and using negative linear curves (since linear curves generally produce more radical change per note than do exponential ones) with maximum depth to scale the various carriers over different areas of the keyboard (or alternatively, you can work with a single-carrier algorithm - as we did in Exercise 74 - and scale its various modulators over different parts of the keyboard). By choosing the break points carefully, you can minimize the necessary areas of overlap that occur because of the use of curves. **(see figure 12-52)**

Thus, the splits accomplished with this technique are not *hard splits* (where you have one sound on one key completely and a totally different sound on the next). **(see figure 12-53)**

These kind of hard splits can be generated easily on the DX7II with the use of split play mode, however, so it's unlikely that you will need to perform this operation in this second generation instrument. If you need to, though, it can be accomplished even more easily via the use of the other, newer, keyboard level scaling mode, called *fractional scaling.*

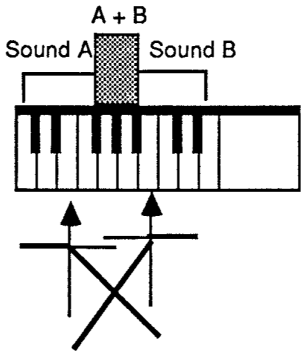


Figure 12-52

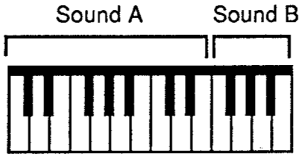


Figure 12-53

Fractional keyboard level scaling

We pointed out earlier in this chapter that the "curves" utilized by normal keyboard level scaling are not true curves but instead change the output level of various operators over a three-semitone range. For example, let's suppose we select a break point of C3 and a negative "curve" to the right of this break point for a particular operator. The notes A#2, B2, and C3 will then all cause that operator to have the same output level, while C#3, D3, and D#3 will cause a lesser output level - but all three notes will be the same as one another. E3, F3, and F#3 will effect a lesser output still. The effect would therefore be similar to that of a negative curve. **(see figure 12-54)**

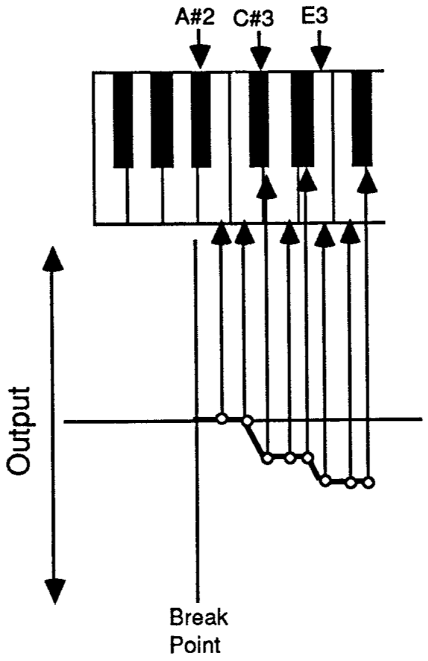


Figure 12-54

If the difference between each of these three groups were the same, then we would be observing a negative linear curve. Whereas if the difference between the first (A#2, B2, and C3) and second (C#3, D3, and D#3) groups were less than the difference between the second and third (E3, F3, and F#3) groups, then we would have a negative exponential curve, since the characteristic of that curve is that the change accelerates as you get further from the break point.
Of course, we can also increase, rather than decrease, an operator's output level - which would result in a positive curve of some kind. The use of four fixed curves by normal level scaling means that the DX7II microprocessor is working with fixed ways of changing output level - in other words, it is always looking up one of four different *tables* of data (data which is stored in its permanent memory). Fractional level scaling, however, allows us to take things one step further, and to alter the fixed tables that the microprocessor uses to generate these "curves" - or even to create our own tables of change (per group of three semitones)! This will allow for much more subtle control over operator output level through the range of the keyboard.

Initialize your DX7II from single voice play mode, and press edit switch 10, followed by edit switch 1. As usual, the LCD is asking you to specify which Scaling mode you wish to use for operator 1. The default is "normal", so that's what you should be currently seeing. The cursor will automatically be over the "mode" parameter, since this is the only adjustable parameter in this display. Press the "yes" button or move the data entry slider so as to change this to "fractional". **(see figure 12-55)**

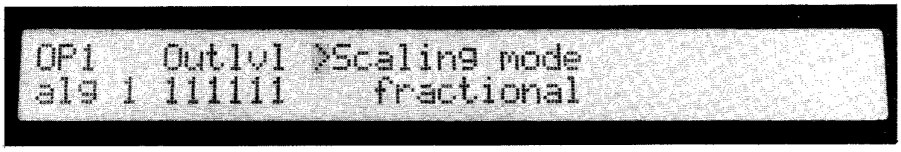


Figure 12-55

Now press edit switch 10 a second time in order to call up the fractional level scaling display. **(see figure 12-56)**

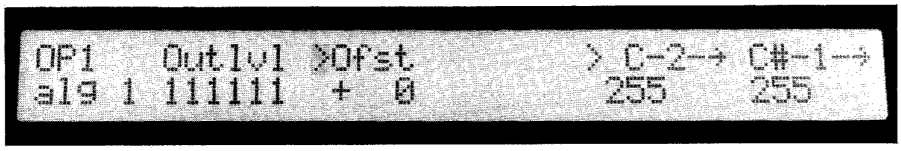


Figure 12-56

This display contains only two adjustable parameters: *Offset* and *Note Group Edit*. Position your cursor over the "offset" parameter at the left of the display (if it isn't already there). The range of this control is -127 to +127, with an initialization default of +0, which is where it should be right now. This parameter effectively takes the place of the nominal Level parameter which appears only in the "normal" display. It allows you to change, or "offset", the entire effect of fractional level scaling for a particular operator, over *the whole keyboard,* by a set amount. We'll work more with this control shortly when we run Exercise 75.

Press the right cursor switch in order to position the cursor over what Yamaha calls the Note Group Edit parameter (which, more accurately, can be referred to as the scaling value parameter). This

display usually shows you three adjacent sets of three semitones so you can view the output level of one group relative to its nearest neighbors. However, when you first initialize, you are shown only two groups - the lowest possible note (C-2), and the next group up (C#-1 to D#-1, represented only by the first note: C#-1). A group consists of three adjacent notes, though only the lowest of the three is actually displayed.

Beneath both "C-2" and "C#-1" is the number 255. This number may or may not represent maximum set output level, depending upon the *offset* parameter in this same display. Here's where things get just a bit complicated, so stick with me: In normal scaling mode, we dealt with output level ranges of 0 to 99. In fractional scaling mode, however, we work with output level ranges of 0 to 255. It's important to understand that a fractional output level of 255 is *no greater* than a normal output level of 99. In fact, if the offset value is positive (anything from +0 to +127), they are one and the same. There is simply a finer gradation to work with when you are in fractional scaling mode - but no output can ever be greater than maximum. Whether you want to call that maximum "255" or "99" is irrelevant.

If the "offset" value is a negative number, however (and remember, it can be as low as -127), then a Note Group value of 255 will *not* represent the absolute maximum output level, but some attenuated value. For example, setting a scaling value of 255 for a particular note group with an offset value of +0 means that the absolute output level of that operator for that group of notes is really maximum. However, changing the offset to -127 would in fact make the absolute output level of that operator for that group of notes equivalent to 0. Changing the offset to -50 would make their output levels greater than 0, but certainly less than maximum - and would in fact be roughly equivalent to a normal output level of 80 or so. Entering a positive offset value (of greater than +0) will *not* increase the output levels of any groups of notes having scaling values of 255, but will proportionally increase any groups having scaling values of less than 255. Once again, explaining this is more complicated than doing it. This will make more sense when we try it out in an exercise shortly.

There are two different ways of stepping through the forty groups of three semitones (from C-2 all the way up to G8!) available here. The first is to use, somewhat unusually, the "internal" and "cartridge" switches as left-right quasi-cursor switches. Written above these two switches in green is the label "FRACTIONAL/MICRO TUNE KEY SET", with a left arrow above the "internal" switch, and a right arrow above the "cartridge" switch. When working in edit mode with fractional level scaling or with micro tuning editing (to be covered in great detail in Chapter Fourteen), these switches serve this additional function. The reason is simple: the standard cursor switches are already allocated for choosing between the "offset" and "scaling value" parameters, and, of course, the "yes-no" data entry switches are used to change those values. Additional cursor switches are needed here, so the "internal" and "cartridge" switches, which are normally idle during editing operations, are pressed into service.

Therefore, press the "cartridge" (right arrow) switch, and note that the "C#-1" group is now in the center of the display, with the blinking cursor placed over it, and the "C-2" note to the left and the next group (of E-1, F-1, and F#-1 - represented only by "E-1") to the right. (see **figure 12-57)**

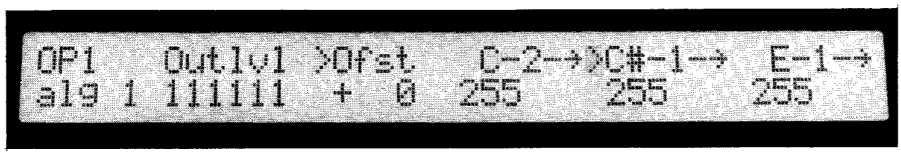


Figure 12-57

Press the "cartridge" switch once again and note that the display shifts up one more group, with "C#-1" (representing C#-1, D-1, and D#-1) shifting to the left, "E-1" (representing E-1, F-1, and F#-1) in the center, and "G-1" (representing G-1, G#-1, and A-1) on the right. As before, the blinking cursor can only be on the center group, represented here by "E-1". **(see figure 12-58)**

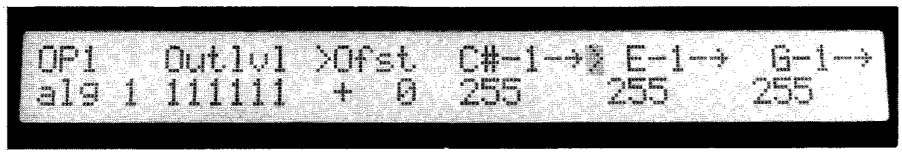


Figure 12-58

Pressing the "internal" (left arrow) switch has precisely the opposite effect, and will move the display to the left. Press it once, and you'll be back to the display shown in figure 12-59. Press it once again, and you'll be back to the original display shown in figure 12-58. Experiment by pressing the "cartridge" and "internal" switches while in this LCD display until you understand thoroughly how they shift the "active" group of three semitones into the center display (called the *active window*), while showing the adjacent groups to the left and right for comparison purposes.

There is a second, faster method for getting the particular group you want into the active window. This is accomplished simply by pressing the key you want, holding it down, and then pressing either the "internal" or the "cartridge" switch. This will automatically put the group containing that key into the center of the display. Note that the key you press will not always be precisely the one displayed, since each group is only represented by the lowest note of the three. For example, play middle C on the keyboard. Hold the key down, and press either the "internal" or the "cartridge" switch. You should now be seeing in your LCD **figure 12-59.**

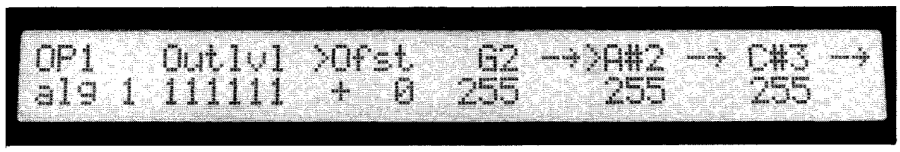


Figure 12-59

"A#2" is in fact the lowest note of the group (of A#2, B2, and C3) that contains C3. **(see figure 12-60)**

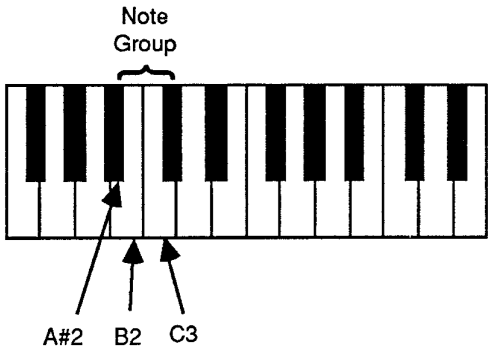


Figure 12-60

By using this latter technique, you can quickly call up any group of notes whose output level you wish to change. However, because the DX7II physical keyboard only contains the notes C1 to C6, you won't be able to use this method to access notes in the "theoretical" C-2 to B0 and C#6 to G8 extensions on either side of the keyboard. **(see figure 12-61)**

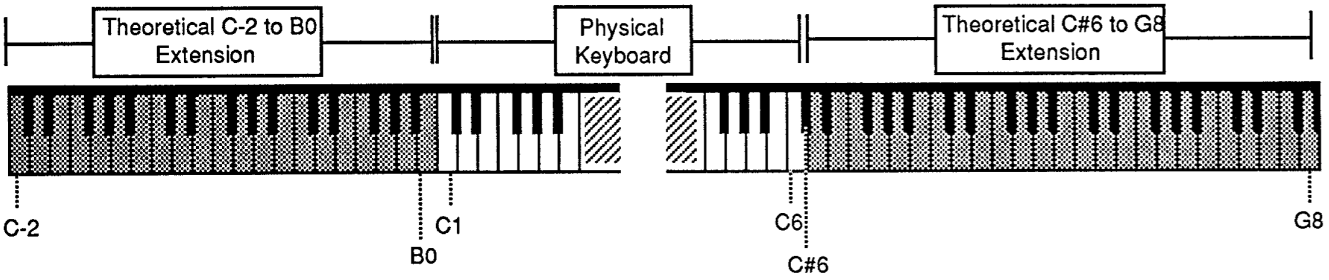


Figure 12-61

Step through all forty available groups, using a combination of the two techniques outlined above, and note that the initialization default is a Note Group Edit value of 255 for all groups for operator one *only* (don't forget, level scaling is *operator-specific* - and that operator 1 defaults to maximum output level when you initialize). Press edit switch 2, followed by edit switch 10, and select fractional Scaling mode for operator 2. Then press edit switch 10 a second time, and observe that the Note Group Edit value for all forty groups for operator 2 is 0 - in reflection of the fact that operator 2's output level defaults to minimum when you initialize. Do the same for operators 3 through 6, and you'll see precisely the same 0 values. In other words, when you initialize, there is no keyboard level scaling occurring - just as we observed when we worked with normal level scaling. Again, because the initialization offset default is 0, this represents maximum output level for operator 1, and minimum (0) output level for operators 2 through 6.

There is one very important fact to be aware of when working with fractional level scaling. Unlike normal level scaling values - which are remembered when you store a voice in internal, cartridge, or disk memory - fractional scaling data can *only* be accessed in real time in a RAM4 cartridge which has been specially formatted for this purpose! The data will then be linked to the voice - but only if the voice is called up from the internal memory. It seems as if there just isn't enough space in the DX7II memory to remember all of this data, so it has to be stored separately - and only in a RAM4 cartridge, since fractional

scaling data stored to disk (from a CRT file) cannot be accessed by the DX7II in real time - it still needs to be offloaded into a RAM4. This means that if you plan on using fractional scaling, you *must* have the voices you want to use in the internal memory, and you *must* have a specially formatted (and therefore dedicated) RAM4 cartridge sitting in the slot. If you call up a voice which was written with some fractional scaling data, therefore, you'd better make sure that that RAM4 cart is in the slot, or you may get unexpected results. If you do call up a voice or performance memory which requires fractional scaling data, and the cartridge is not in the slot, you'll see a little "f" next to the voice number(s) in the LCD. **(see figure 12-62)**

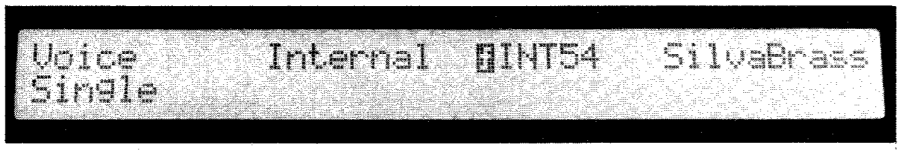


Figure 12-62

This is simply a reminder that the microprocessor is looking for fractional scaling data and not finding any. It will therefore assume no level scaling whatsoever for that particular voice or performance memory - which, as pointed out above, may severely change the sound. Poor marks here to Yamaha for inefficient and convoluted memory management - but complaining about it won't help nearly as much as learning about it will. Let's try an exercise now and see how we can apply some of this information in a simple modulator-carrier system, and how this data can be stored.

Exercise 74
Fractional level scaling applied to a simple system:

1) INITIALIZE your DX7II from single voice play mode, TURN OFF operators 3 through 6 ("110000"), and, using the system of operators 1 and 2, GENERATE a sawtooth wave.

2) We'll start by applying fractional level scaling to our carrier (operator 1 in this system). Therefore, press edit switch 10, followed by edit switch 1 and call up the Scaling mode display (you may have to press edit switch 10 twice in order to get the display shown in **figure 12-63.**

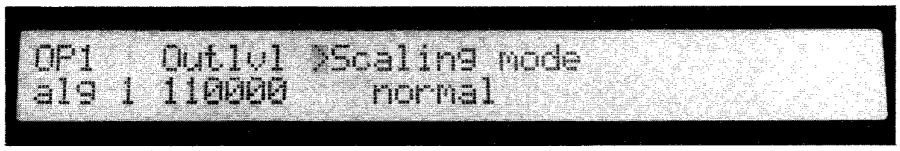


Figure 12-63

3) Note that this parameter is currently at its initialization default of "normal". Press the "yes" button in order to change this to "fractional".

4) Press edit switch 10 a second time in order to call up the fractional display. **(see figure 12-64)**

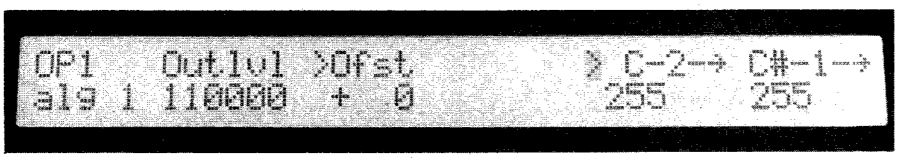


Figure 12-64

5) Note that the "offset" parameter is currently at its default of +0. Leave it there, and press the right cursor switch in order to position the cursor over the Note Group Edit parameter (which should currently have the lowest possible note - C-2 - in the active window). Press and hold down middle C on the keyboard. Listen and note its volume (Audio Cue 74A). While continuing to hold it down, press either the "internal" or "cartridge" switch. You should now see "A#2" in the active window. This represents the three notes A#2, B2, and C3. Use the data entry section to change the Note Group Edit value (scaling value) for this group to a new value of 180. Play middle C again and listen (Audio Cue 74B). Note that it is now significantly softer. Play a chromatic scale from C2 up to C4 and listen (Audio Cue 74C). Note that all these notes are at the same, maximum volume, *except for* the notes A#2, B2, and C3, which are at a significantly lower volume.

6) Press the "cartridge" switch once in order to shift the display to the right, putting the next group, represented by "C#3" (which actually means the notes C#3, D3, and D#3) into the active window. Use the data entry slider to change the scaling value for this group to a new value of 210. Press the "cartridge" switch once again, bringing the "E3" group (consisting of E3, F3, and F#3) into the active window. Change the scaling value for this group to a new value of 225. Press the "cartridge" switch one more time, bringing the "G3" group (consisting of G3, G#3, and A3) into the active window, and change the scaling value for this group to a new value of 240. Play a chromatic scale from C2 up to C4 and listen (Audio Cue 74D). Note that the notes from C2 to A2 are all at maximum volume; that the volume drops drastically for A#2; and that it slowly builds up again to maximum in the area A#2 - A#3. We have essentially built a "curve" into this one small area of the keyboard, without affecting the rest of the keyboard. **(see figure 12-65)** Using this technique, we can scale any carrier responsible for any kind of component sound over any area of the keyboard, in virtually any way at all!

7) Now let's see how the offset parameter affects this scaling. Press the left cursor switch in order to position the cursor over "Offset", and use the data entry slider to enter in a new value of -25. Play the same chromatic scale from C2 to C4 and listen (Audio Cue 74E). Note that the same volume change occurs as in step 6 above, but that the overall volume of *all* the notes is reduced. **(see figure 12-66)**

Entering a *positive* offset value will have the effect of raising the volume of the attenuated keys (A#2 to A3) but will *not* affect those keys already at maximum (255) output. Let's try it: enter in a new offset value of +50. Play the same chromatic scale from C2 to C4 and listen (Audio Cue 74F). Note that the same kind of volume change occurs in the A#2 to A3 area, but that it is far less steep **(see figure 12-67)** and also note that the notes C2 to A2 and A#3 to C4 are at precisely the same volume they were at originally. This is because, as with normal level scaling, you can never increase an operator's output level beyond maximum.

8) Let's listen to precisely what this positive offset does to the softest and loudest notes. Restore the offset value back to its initialization default of +0. With your right hand, play an alternating octave of A#2 (the softest note) and A#3 (which is at maximum output), while slowly raising the data entry slider in order to change the "offset" eventually up to its maximum of +127. Listen as you do so

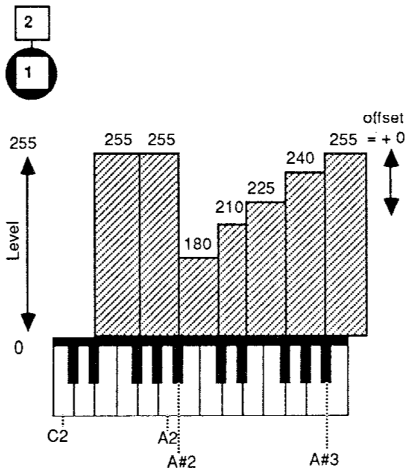


Figure 12-65

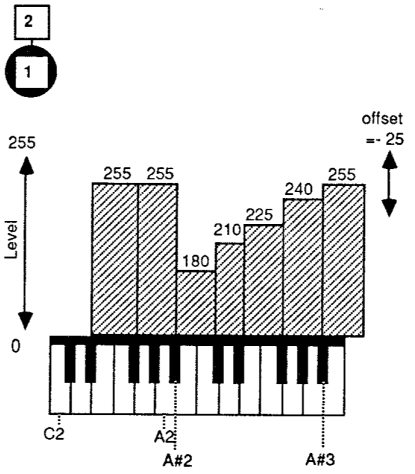


Figure 12-66

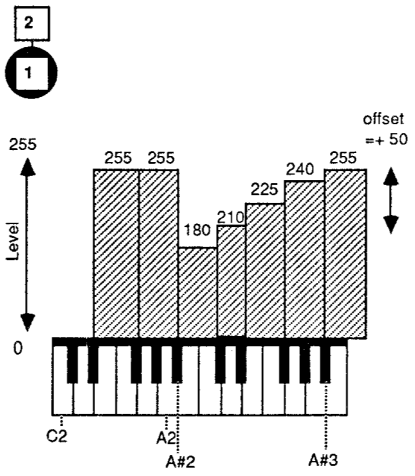


Figure 12-67

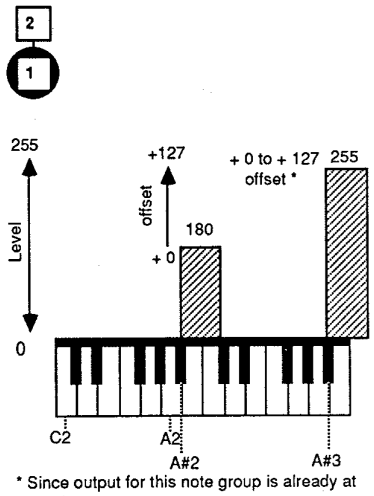


Figure 12-68

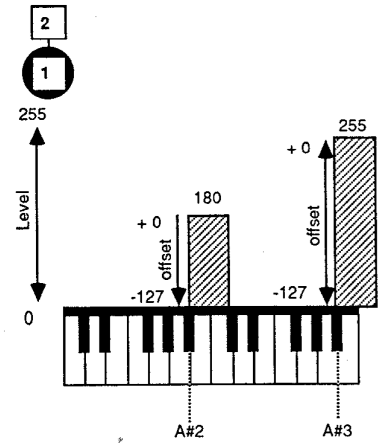


Figure 12-69

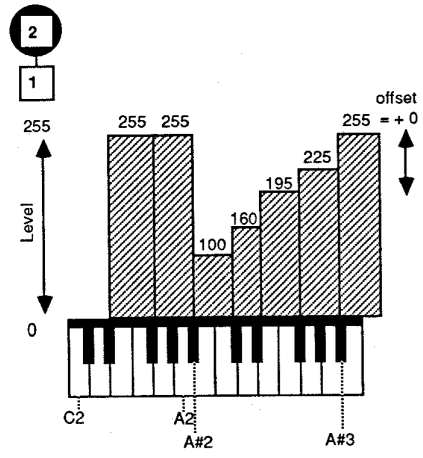


Figure 12-70

(Audio Cue 74G). You should be able to clearly hear the A#2 note get louder, while the A#3 remains the same. **(see figure 12-68)**

9) Restore the offset to +0. Now try playing the same alternating octave (A#2 and A#3) while slowly lowering the data entry slider in order to change the "offset" eventually down to its minimum of -127, listening as you do so (Audio Cue 74H). This time, the *whole sound* gets softer, though the relative difference between A#2 and A#3 remains the same. **(see figure 12-69)** Clearly, increasing the value of the offset control on a carrier has the effect of "squeezing" the available dynamic range, while decreasing it simply lowers the overall volume - while keeping the difference between the softest and loudest notes constant.

10) Experiment further by creating different timbres using a single system and by having the volume increase or decrease over different areas of the keyboard using the fractional scaling controls to adjust the output level of your carrier.

11) Now let's try these same manipulations, but this time we'll apply them to a modulator. Cardinal Rule Two tells us that these alterations of modulator output level will result in a quantitative timbral change. Let's hear it for ourselves: Re-INITIALIZE your DX7II, TURN OFF operators 3 through 6 ("110000"), and, using the system of operators 1 and 2, GENERATE a sawtooth wave yet again.

12) Press edit switch 10, followed by edit switch 2, and call up the Scaling mode display (you may have to press edit switch 10 twice in order to get this display) for operator 2. Note that this parameter is currently at its initialization default of "normal". Press the "yes" button in order to change this to "fractional".

13) Press edit switch 10 a second time in order to call up the fractional display. Note that the "offset" parameter is currently at its default of 0. Leave it there, and press the right cursor switch in order to position the cursor over the Note Group Edit parameter. Press and hold down middle C on the keyboard. Listen and note that the sound has the bright timbre of a sawtooth wave (Audio Cue 74I). While continuing to hold it down, press either the "internal" or "cartridge" switch. You should now see "A#2" in the active window. This represents the three notes A#2, B2, and C3. Use the data entry section to change the Note Group Edit value for this group to a new value of 100. Play middle C again and listen (Audio Cue 74J). Note that our sound is now no longer a sawtooth wave but is instead a pure sine wave (since the scaled output value of 100 - equivalent to a normal output level of approximately 40 - is nearly inaudible). Play a chromatic scale from C2 up to C4 and listen (Audio Cue 74K). Note that all these notes are sawtooth waves, *except for* the notes A#2, B2, and C3, which are all sine waves.

14) Press the "cartridge" switch once in order to shift the display to the right, putting the next group, represented by "C#3" (which actually means the notes C#3, D3, and D#3) into the active window. Use the data entry slider to change the Note Group Edit value (scaling value) for this group to a new value of 160. Press the "cartridge" switch once again, bringing the "E3" group (consisting of E3, F3, and F#3) into the active window. Change the scaling value for this group to a new value of 195. Press the "cartridge" switch one more time, bringing the "G3" group (consisting of G3, G#3, and A3) into the active window, and change the scaling value for this group to a new value of 225. Play a chromatic scale from C2 up to C4 and listen (Audio Cue 74L). Note

that the notes from C2 to A2 are all sawtooth wave; that A#2, B2, and C3 are all sine waves; and that notes above that point slowly build up again in overtone content to maximum in the area A#2 - A#3, where we once again hear sawtooth waves. We have essentially built a "curve" into this one small area of the keyboard, without affecting the rest of the keyboard. **(see figure 12-70)** Using this technique, we can scale any modulator, placing any kind of timbre over any area of the keyboard - in virtually any way at all!

15) Now let's see how the offset parameter affects this scaling. Press the left cursor switch in order to position the cursor over "Offset", and use the data entry slider to enter in a new value of -25. Play the same chromatic scale from C2 to C4 and listen (Audio Cue 74M). Note that the same timbral change occurs as in step 6 above, but that the overall overtone content of our sound is reduced throughout - that is, we never start with a sawtooth wave, even for non-scaled notes. **(see figure 12-71)** Entering in a *positive* offset value will have the effect of increasing the overtone content of the attenuated keys (A#2 to A3) but will *not* affect those keys already at maximum (255) output. Let's try it: enter in a new offset value of +50. Play the same chromatic scale from C2 to C4 and listen (Audio Cue 74N). Note that the same kind of timbral change occurs in the A#2 to A3 area, but that it is far less steep: in other words, A#2 is not a pure sine wave, but already has several overtones, even with its 0 scaling value. This is because the offset of +50 is essentially adding 50 increments to that value. **(see figure 12-72)** Also note that the notes C2 to A2 and A#3 to C4 are precisely the same sawtooth waves they were before. This is because, as with normal level scaling, you can never increase an operator's output level beyond maximum.

16) Let's listen to just what this positive offset does to the most affected and least affected notes. Restore the offset value back to its initialization default of +0. With your right hand, play an alternating octave of A#2 (which is a sine wave) and A#3 (which is a sawtooth wave), while slowly raising the data entry slider in order to change the "offset" eventually up to its maximum of +127. Listen as you do so (Audio Cue 74O). You should be able to clearly hear the A#2 note get brighter, while the A#3 remains the same. **(see figure 12-73)**

17) Restore the offset to +0. Now try playing the same alternating octave (A#2 and A#3) while slowly lowering the data entry slider in order to change the "offset" eventually down to its minimum of -127, listening as you do so (Audio Cue 74P). This time, the *whole sound* gets warmer (fewer overtones), though the relative difference between A#2 and A#3 remains the same. **(see figure 12-74)** Clearly, changing the offset value for a modulator has the effect of "squeezing" the available timbral range.

18) Finally, let's try storing this fractional level scaling data. In order to do this, you'll need a fresh or expendable RAM4 cartridge. DO NOT perform this operation on a RAM4 cartridge containing data you wish to keep, because the formatting operation will irretrievably destroy any existing data in the cartridge! First of all, turn both software (edit switch 14) and hardware cartridge memory protections OFF (see Chapter Eight if you can't remember how to do this). Press edit switch 15 (CARTRIDGE) repeatedly if necessary in order to call up the "Fractional SC." display. **(see figure 12-75)** Unless the cartridge you're using has previously been formatted specifically for storing fractional scaling data, the display will read "unfmtd". Position the cursor over

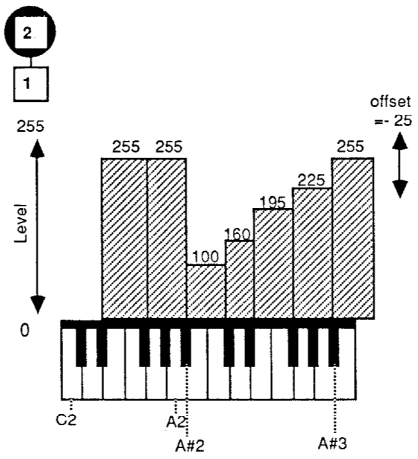


Figure 12-71

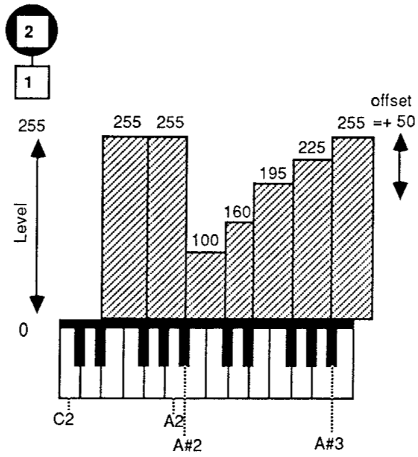
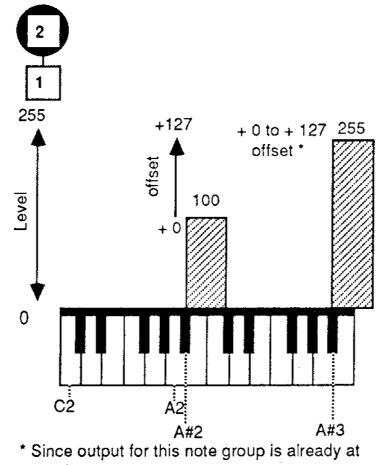


Figure 12-72



* Since output for this note group is already at maximum.

Figure 12-73

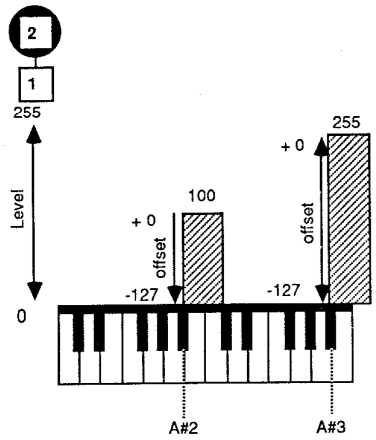


Figure 12-74

the "Format" parameter, and press the "yes" button. The display asks, "** Are you sure?". If you are, - bearing in mind the warning given above - press the "yes" button again. A blink of the eye later, the display reads, "** Completed!". We are now ready to store the data created in steps 11 through 17 above.

Figure 12-75

19) In order to store fractional scaling data, you have to store the voice involved in an internal memory slot while the properly formatted RAM4 cartridge sits in the slot. In this way, the voice data goes to the internal memory slot you select, while the appropriate fractional scaling data is automatically sent to the same slot in the RAM4 cartridge. Whenever you then call up that particular voice from the internal memory, the correct fractional scaling data will be linked to that voice - as long as the cartridge is in the slot. If the cartridge isn't in the slot, you'll get that clever little "f" in the display, and the DX7II will assume no level scaling whatsoever. If you don't know offhand of an expendable slot in your machine's internal memory, return to single voice play mode and find one. Then use the Recall Edit function (again, refer to Chapter Eight if you don't remember how to do this) to recall your voice. Turn your internal memory protection OFF and turn both the hardware and software memory protections for your RAM4 cartridge OFF.. Leave the internal memory protection OFF, since the link to this fractional scaling data will need to be written onto the voice (this is done automatically for you by the DX7II) when we store the fractional data in the next step.

20) Since we originally initialized from single voice play mode, we'll have to be in that mode in order to store thevoice and fractional scaling data. That's exactly where you should be right now. If for any reason you're not there, simply press the single voice play mode select switch. Then press and hold down the pink Store button. The display now reads, "Store data to memory ? with fractional". These last two words tell you that the DX7II recognizes fractional scaling data has been written and needs to be stored in a RAM4 if it is to be saved. If you have a properly formatted RAM4 in the slot, there will be no problem doing this - but if you have a ROM cartridge or RAM4 formatted for voice/performance or micro tuning data in the slot, you'll see a "scaling format error!" warning instead. We'll now use the numbered main switches to select the desired memory location in the RAM4 cartridge. Free up internal memory slot 1 (by moving the voice in there to an unneeded slot), and let's store this voice in internal slot, along with the fractional scaling data in the RAM4 slot 1. Therefore, while continuing to hold down the Store button, press main switch 1. The "?" in the LCD obediently changes to a "1", indicating that our microprocessor "hears" us. While continuing to keep both the Store and "1" switches held down, use your pinky to press the "yes" button. Finally, the display reads, "Completed!", and the process is finished. The voice is now safely stored in your internal memory and the fractional scaling data (which will automatically be linked to that voice whenever you call it up) is safely stored in your RAM4 cartridge.

21) Experiment further with fractional level scaling changes to a modulator or modulators within a single system, and with the process of storing this data onto a RAM4 cartridge until you feel comfortable with the entire procedure.

One of the most powerful applications for fractional level scaling is that it allows you to view and modify the curves selected with normal level scaling! In this way, we can actually see and/or change the tables of values called up by the DX7II's microprocessor when you select a "normal" mode fixed curve of a particular depth. Let's run an exercise now to try it out:

Exercise 75
Viewing normal scaling curves with fractional scaling mode

1) INITIALIZE your DX7II from single voice play mode, TURN OFF operators 3 through 6 ("110000"), and, using the system of operators 1 and 2, GENERATE a sawtooth wave.

2) Press edit switch 10, followed by edit switch 1, in order to VIEW the Scaling mode for operator 1. Observe that it is currently at its default of "normal". Leave it that way, and press edit switch 10 a second time. Observe that all the values in the display are currently at their initialization defaults: Level = 99; Lc = -lin; Ld = 0; Bp = C3; Rc = -lin; and Rd = 0.

3) Press edit switch 10 again and change the Scaling mode from "normal" to "fractional". Press edit switch 10 a second time in order to VIEW the fractional scaling display.

4) Position the cursor over the Note Group Edit parameter (scaling value parameter). Play middle C (C3) on the keyboard, and, while holding it down, press either the "internal" or the "cartridge" switch. Your LCD should now show the "A#2" group in the active window, with a scaling value of 255. **(see figure 12-76)**

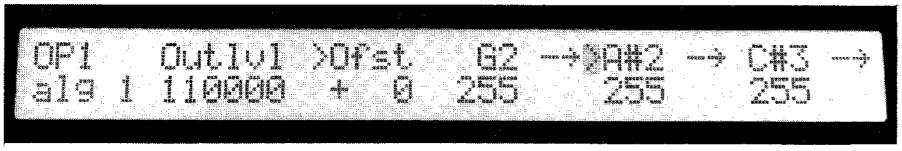


Figure 12-76

5) Press the "cartridge" switch once in order to bring up the next group (C#3) into the active window. Note that it, too, has a scaling value of 255. Continue pressing the "cartridge" switch until you step through all of the different note groups and observe that they all currently have scaling levels of 255. This is because operator 1 currently has an output level of 99, with the right curve assigned no depth - in other words, there is no keyboard level scaling to the right (or to the left, for that matter) of middle C.

6) Press edit switch 10 again in order to call up the Scaling mode display and press the "no" button in order to change this back to "normal". Press edit switch 10 again in order to call up the "normal" LCD display.

7) Position the cursor over the Rd parameter and change the Rd to a new value of 99, giving us a negative linear curve to the right of middle C, with maximum depth. Play a chromatic scale from C3 up to C6 (the highest note on the keyboard) and listen (Audio Cue 75A). Note that the volume of the sawtooth wave we are hearing slowly decreases as we play higher notes on the keyboard.

8) Press edit switch 10 again in order to return to the Scaling mode display, and press the "yes" button in order to change the Mode from "normal" to "fractional". Play the same chromatic scale as before (from C3 up to C6) and listen (Audio Cue 75B). Note that there is no change whatsoever to the sound - the sawtooth wave we are hearing is still slowly fading away as we play higher and higher notes. Whenever you change over from "normal" scaling mode to "fractional" scaling mode, *there will be no change whatsoever to the sound.**

9) Press edit switch 10 yet again in order to VIEW the fractional scaling display. Position the cursor over the Note Group edit parameter (scaling level parameter). Play middle C on the keyboard, hold it down, and, while continuing to hold it down, press either the "internal" or "cartridge" switch. You should now see "A#2" in the active window of your LCD, with a scaling value of 241, instead of the 255 we observed a moment ago. **(see figure 12-77)**

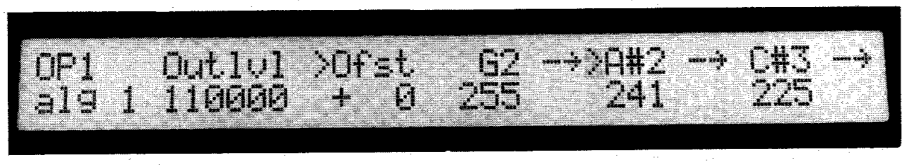


Figure 12-77

10) Press the "cartridge" switch once in order to bring up the next group (C#3) into the active window. Note that it now has a scaling value of 225, instead of the 255 we observed before. Continue pressing the "cartridge" switch until you step through all of the different note groups and observe that they all currently have different scaling levels than they did before (i.e. E3 = 209, G3 = 193, etc.) and that these levels slowly but steadily decrease as we VIEW higher and higher notes. This is because operator 1 has been scaled with a negative linear curve to the right of middle C (really A2), with maximum depth.

11) Press edit switch 10 once again in order to call up the Scaling mode display, and change this back to "normal". Let's try changing the depth of this negative linear curve, and see how this change is reflected in the fractional display. Therefore,

12) Press edit switch 10 again in order to call up the normal display, position the cursor over the "Rd" parameter, and change the right depth to a new value of 50. Play the same chromatic scale (C3 to C6) and listen (Audio Cue 75C). Note that we hear the same volume change as before, but less drastically so.

13) Press edit switch 10 once again and change the Scaling mode back to "fractional". Press it one more time in order to call up the fractional display. Play middle C on the keyboard, hold it down, and press either the "internal" or "cartridge" switch. The "A#2" note group should now be in the active window. Note that it now has a scaling value of 247, as opposed to the 241 that we observed in step 9 above. This is because its output level is increased, due to the decreased depth (steepness) of the negative linear curve. **(see figure 12-78)**

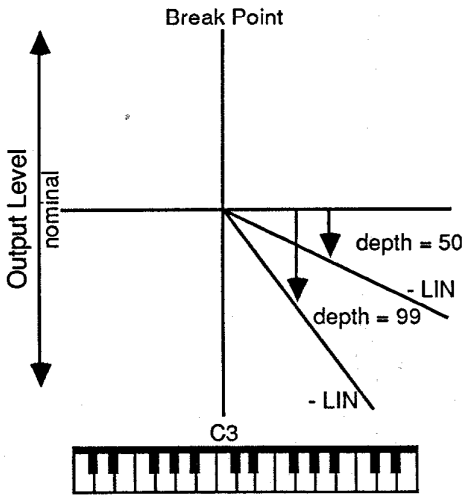


Figure 12-78

* When going from fractional mode back to normal mode, however, the DX7II will temporarily reset back to a "no-scaling" condition if and *only if* you made any changes to any fractional values. This "no-scaling" condition will not only reflect itself in the sound (with no output level change whatsoever over the keyboard) but also in the LCD (with the break point reset to C3, with -lin curves at 0 depths on both sides). Therefore, you can always freely travel from normal to fractional mode with no change to the sound, but you can only freely travel from fractional to normal mode (with no change to the sound) if you refrain from making any changes in the fractional display. This rather confusing set of circumstances is apparently meant to avoid confusion when going back and forth between these modes. Obviously, it doesn't succeed!

14) Press the "cartridge" switch once in order to bring up the next group (C#3) into the active window. Note that it now has a scaling value of 239, instead of the 225 we observed in step 10 above. Continue pressing the "cartridge" switch until you step through all of the different note groups and observe that they all currently have higher scaling levels than they did previously (i.e. E3 = 231, G3 = 223, etc.), and that these levels slowly but steadily decrease *in a less severe fashion than before* as we VIEW higher and higher notes. This is because the depth of the negative linear curve has been decreased. Play the same C3 to C6 chromatic scale on the keyboard and listen (Audio Cue 75D). Note that we are still hearing precisely the same gradual decrease in volume that we did in step 12 above.

15) Experiment further with altering the depth of this curve and observing how these changes are reflected in the fractional display.

16) Now let's VIEW the fractional values for an exponential, as opposed to linear, curve. Let's wipe the slate clean by re-INITIALIZING altogether, and re-GENERATING a sawtooth wave from the system of operators 1 and 2, as in step 1 above.

17) Press edit switch 10, followed by edit switch 1. Leave this operator in "normal" Scaling mode and press edit switch 10 a second time in order to call up the normal display. Position the cursor over the Rc parameter and change the right curve to "-exp". Press the right cursor switch once and change the Rd parameter to 99. We have now scaled the output of operator 1 to the right of middle C with a negative exponential curve with maximum depth. Play the same C3 to C6 chromatic scale on the keyboard and listen (Audio Cue 75E). Note that the volume change is quite subtle, and virtually inaudible until nearly C5, due to the two-octave "dead" area of this curve. Note also that the change becomes more and more apparent for higher notes, since a characteristic of exponential curves is that their change accelerates as you get further from the break point.

18) Press edit switch 10 again and change the Scaling mode to "fractional". Press it once more in order to call up the fractional display. Play the same C3 to C6 chromatic scale on the keyboard and listen (Audio Cue 75F). Note that we still hear the same volume change as in step 17 above. Position the cursor over the Note Group Edit (scaling value) parameter. Play middle C on the keyboard, and while continuing to hold it down, press either the "internal" or "cartridge" switch, bringing the "A#2" note group into the active window. Note that its scaling value is now at the maximum value of 255, indicating that no change whatsoever has occurred to the output level of operator 1 over this note group. Press the "cartridge" switch in order to call up the next note group (C#3) into the active window. Note that its scaling value is now 253, only very slightly (and really inaudibly) attenuated, relative to the A#2 group. Continue pressing the "cartridge" switch in order to VIEW the scaling values for higher and higher note groups. Note that the changes from group to group are far more subtle (i.e. E3 = 251, G3 = 249) than the values we observed with a negative linear curve at maximum depth. **(see figure 12-79)**

19) Observe that the change in scaling value remains quite subtle (two increments per group) until we reach the C#5 group, at which time the difference from group to group becomes wider. Eventually, at the uppermost notes, the difference from group to group reaches over *thirty* increments. This ties in with the concept of exponential curves initiating greater and greater change as they get further from the break point.

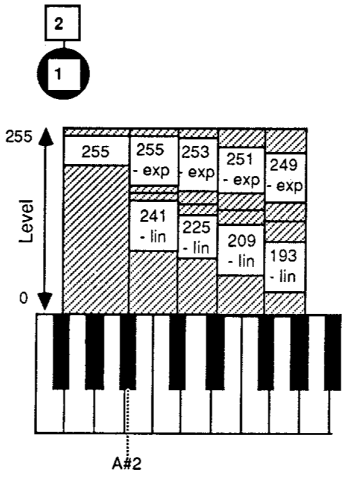


Figure 12-79

20) Experiment by returning to normal mode and changing the depth of the negative exponential curve, and observe how these alterations are reflected in the fractional display as more subtle scaling changes from group to group. experiment further by VIEWing the fractional results of using positive linear and exponential curves. Remember that the nominal output level (Level parameter) of the affected operator must be less than 99 for these positive curves to have any aural effect and for the fractional scaling values to reflect these changes (since no note group can ever have a higher scaling value than 255).

Of course, at any point, we could have made alterations to any of the scaling values we observed in fractional mode, thereby modifying the overall effect to the sound. It should go without saying (but I'll say it anyway) that these same kinds of manipulations can be performed with any of the six operators, be they carrier or modulator. Try redoing Exercise 75 above, and this time make subtle or gross changes to the scaling values you observe. Note how each of these manipulations affects the overall sound. Remember, as we pointed out in the footnote to step 8, that if you make any changes to any fractional parameters, returning to normal mode will cause the normal scaling parameters (and the sound) to be quasi-initialized to a state of "no-scaling".

Finally, let's see how we can use fractional level scaling to set up hard splits in a single voice. This can be accomplished in one of two different ways. The first way is to choose an algorithm with multiple carriers, set up different sounds with each carrier, and scale the effects of each carrier over different parts of the keyboard. The second technique, similar to the one we used in Exercise 74 above, is to just work with a single carrier, but using different modulators to cause the carrier to generate different overtones. In this case, we want the carrier over the entire keyboard - but we will scale the *modulators,* so as to induce radically different timbres over different areas of the keyboard. Let's run an exercise now and create a single voice with this latter technique that gives us a bass sound, a vibraphone (vibes) sound, and a brass sound, split over various notes.

Exercise 76
Creating a hard split using fractional level scaling

1) INITIALIZE your DX7II from single voice play mode. Select algorithm #16 **(see figure 12-80)** and TURN OFF all operators except the carrier, operator 1 ("100000"). We'll be using the system of operators 2 and 1 to create a bass sound and we'll scale it over only the bottom octave of the keyboard. The system of operators 3, 4, and 1 will be used to generate a vibraphone sound and will be placed over the next two octaves (C#2 to C4). Finally, the system of operators 5, 6, and 1 will be used for a brass sound, which will be scaled so that it appears only in the top two octaves (C#4 to C6). **(see figure 12-81)**

2) Let's start our programming with operator 1, the carrier that will be responsible for all three of these sounds. We'll leave it at its default maximum output level and at its default ratio number of 1.00. However, let's change a few of its EG settings so as to create a non-sustaining sound with a slow decay and moderate after-ring. Therefore, press edit switch 9, followed by edit switch 1, and enter in the following EG values for operator 1:

Rs	R1	R2	R3	R4	L1	L2	L3	L4
0	99	99	28	56	99	99	0	0

Algorithm#16:

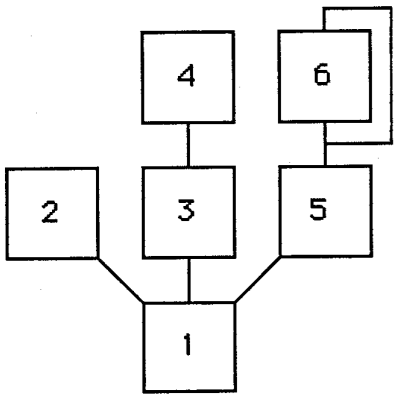


Figure 12-80

Play a few notes on the keyboard and listen (Audio Cue 76A). Note that we simply hear a basic sine wave with the slight changes (non-sustaining and short after-ring) outlined above.

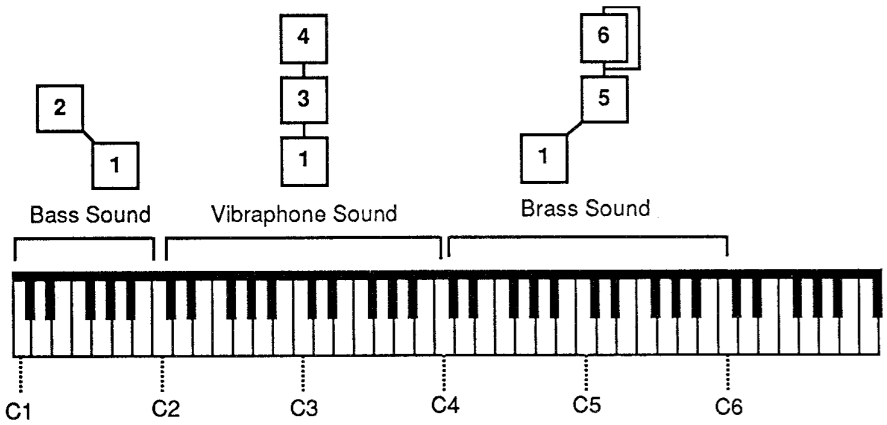


Figure 12-81

3) Now let's TURN ON operator 2 ("110000") and generate the bass sound. Start by halving operator 2's ratio number to a new value of 0.50. This will force the carrier (operator 1) to drop down in pitch by a full octave wherever operator 2 is in force (since having a modulator at a slower frequency than the carrier always causes the carrier to assume the fundamental frequency of the modulator. See Chapter Six if you don't remember this). Set operator 2's output level at 77. Now change operator 2's EG settings, as follows:

Rs	R1	R2	R3	R4	Ll	L2	L3	L4
5	99	39	75	53	99	50	0	0

This generates an EG with a rapid attack, two-tiered decay to a 0 sustain level, and short after-ring. This EG is also heavily rate-scaled, so that higher notes undergo their changes faster than lower ones do. Of course, since we will ultimately only be using this modulator in the lower-most octave of the keyboard, this will have only a very subtle effect. After you have entered in these values, play a few low keys and listen (Audio Cue 76B). Note that we have created a fairly convincing bass sound out of this simple system.

4) The next step is to use fractional scaling so as to place this sound only over the bottom-most octave of the keyboard. Therefore, press edit switch 10 (twice if necessary) in order to call up the Scaling mode display. Be sure you are currently VIEWing operator 2 (the upper left-hand corner of the LCD will tell you so). Press the "yes" button in order to change this to "fractional" mode. Press edit switch 10 again and observe the fractional display. Because the output level of operator 2 had been set at 77, the scaling value for all note groups (since we didn't change the normal scaling defaults of 0 depth for both curves) is currently at 211. Position the cursor over the Note Group Edit parameter (scaling value parameter). Play C2 (C below middle C) on the keyboard and, while holding it down, press either the "internal" or the "cartridge" switch. This will have the effect of placing the "A#1" note group (consisting of the notes A#1, B1, and C2) in the active window. We want this modulator to have 0 output level above C2 for this voice, however, so press the "cartridge" switch in order to call up the next higher note group (C#2) into the active window. Use the data entry slider to enter in a scaling value for this group of 0, and then use the "cartridge" switch in order to step through all the note groups above

this point, stopping each time to use the data entry slider to enter in a scaling value of 0. The end result of this operation should yield the following scaling values for operator 2 only:

	All note groups below A#1	A#1	C#2	All note groups above C#2
Scaling value:	211	211	0	0

Play a few notes on the keyboard and note that this bass sound is now present only in the bottom-most octave of the DX7II keyboard (or in all notes up to and including C2 if you are using a larger, external MIDI keyboard) and that above C2, you hear only the pure sine wave of the carrier alone.

5) Now let's create the vibraphone sound with modulators 3 and 4. TURN OFF operator 2 - we're done with it for now - and TURN ON operators 3 and 4 ("101100").

6) Begin by pressing edit switch 3 in order to VIEW operator 3. Set its ratio number to 14.00 and raise its output level to a value of 44. Now press edit switch 4 in order to VIEW operator 4. Set operator 4's ratio number to 17.10 (from a Coarse value of 15.00) and raise its output level to a value of 68. Now enter in the following EG values for these two operators:

Operator	Rs	R1	R2	R3	R4	L1	L2	L3	L4
3	0	99	59	38	21	99	27	0	0
4	0	99	99	99	25	99	99	0	0

The selection of these ratio numbers, output levels and EG values combine to produce a percussive sound rich in high harmonic and inharmonic overtones which undergoes relatively little timbral change throughout its duration. The end result? A strikingly accurate vibes sound, as you will discover if you play a few notes and listen (Audio Cue 76C).

7) As before, we'll now want to scale this system so that it appears only over a certain area of the keyboard - in this case over the second and third octaves (from C#2 to C4). Here, however, we won't need to scale both modulators in this stack, since operator 4 can only affect operator 1 through operator 3 - in other words, we won't need to scale operator 4 if operator 3 is already being scaled. We could scale operator 4 as well, of course, but simply entering in the same values would be redundant and we aren't looking for any further timbral changes in our sound. Therefore, press edit switch 3, followed by edit switch 10 once or twice in order to call up the Scaling mode display. Press the "yes" button in order to change this from "normal" to "fractional". Now press edit switch 10 again in order to call up the fractional display. Because operator 3 had been set at an output level of 44, the scaling value for all note groups will currently be 145. Position the cursor over the scaling value parameter. Play C2 on the keyboard, and, while holding it down, press either the "internal" or the "cartridge" switch. This will put the "A#1" group (consisting of A#1, B1, and C2) in the active window. Since this is the area of the keyboard already containing the bass sound (although you can't hear it since operator 2 is currently OFF), we want the output level of operator 3 to be 0 at this point and below. Similarly, we want operator 3's output level to be 0 from C#4 on up. Therefore, enter in the following scaling values for operator 3:

	All note groups from C-2 to A#1	All note groups from C#2 to A#3	All note groups from C#4 to G8
Scaling value:	0	145	0

Play a few notes on the keyboard and note that this vibraphone sound is now only present in the notes C#2 to C4 and that only the pure sine wave of operator 1 is present above and below these points.

8) Finally, on to the brass sound of modulators 5 and 6. TURN OFF operators 3 and 4 - we're done with them for now - and TURN ON operators 5 and 6 ("100011").

9) Begin by pressing edit switch 5 in order to VIEW operator 5. Set its ratio number to 0.50, detuned +2, and raise its output level to a value of 76. Now press edit switch 6 in order to VIEW operator 6, and give it the same ratio number of 0.50, but detuned +5. Having both of these modulators at half the frequency of the carrier (operator 1) will result in the pitch of this system being lowered an octave - which we want to do since this sound will be over the top two octaves of the keyboard and would be much too high in pitch otherwise. Raise operator 6's output level to a value of 81. Now enter in the following EG values for these two operators:

Operator	Rs	R1	R2	R3	R4	Ll	L2	L3	L4
5	3	47	32	76	7	99	51	0	0
6	3	50	99	76	20	99	99	99	0

The selection of these ratio numbers, output levels and EG values combine to produce a convincingly brass-like sound. Play a few notes and listen (Audio Cue 76D).

10) As before, we'll now want to scale this system so that it appears only over a certain area of the DX7II keyboard - in this case over the top two octaves only (from C#4 on up). Again, we won't need to scale both modulators in this stack, since operator 6 can only affect operator 1 through operator 5. Therefore, press edit switch 5, followed by edit switch 10 once or twice in order to call up the Scaling mode display. Press the "yes" button in order to change this from "normal" to "fractional". Now press edit switch 10 again in order to call up the fractional display. Because operator 5 had been set at an output level of 76, the scaling value for all note groups will currently be 209. Position the cursor over the scaling value parameter. Play C4 on the keyboard, and, while holding it down, press either the "internal" or the "cartridge" switch. This will put the "A#3" group (consisting of A#3, B3, and C4) in the active window. Since this is the area of the keyboard already containing the vibes sound (although you can't hear it since operators 3 and 4 are currently OFF), we want the output level of operator 5 to be 0 at this point and below. Therefore, enter in the following scaling values for operator 5:

	All note groups below A#3	A#3	C#4	All note groups above C#4
Scaling value:	0	0	209	209

Play a few notes on the keyboard and note that this brass sound is now only present in the upper two octaves of the keyboard and that only the pure sine wave of operator 1 is present below that point.

11) Finally, turn all operators ON ("111111"). A few moments of playing the keyboard will show that your split is complete!

Keyboard level scaling - both normal and fractional - is a powerful tool that allows you to shape your sound over the various registers of the DX7II. Spend time with these controls and get comfortable with their operation and application before moving on to our further discussion of voice edit parameters.

Chapter Thirteen ◇
The Voice Edit Parameters

In this chapter, we will discuss a group of edit parameters collectively called *Voice Edit* parameters. As mentioned way back in Chapter Two, this term is somewhat misleading in that these parameters are no different from the ones we have already examined - they are voice-specific and are stored when you store a voice. Most of these so-called Voice Edit parameters were found in the original DX7. There, they were called *function controls*, but *none* of them were voice-specific. In other words, these parameters could not be remembered by the DX7 when the voice was stored - they had to be set manually each time you wanted to change them. Additionally, these parameters did not reset themselves to default values when you initialized.

All of these problems have been addressed in the DX7II, I am happy to report. Therefore, the parameters we will be discussing here act very much like all the other edit parameters - they reset to default values when you initialize, and they are stored along with all the other voice data.

Key mode

We'll begin by examining the variables offered by edit switch 23, labeled "KEY MODE". Initialize your instrument and press this switch. The LCD display will read similar to **figure 13-1.**

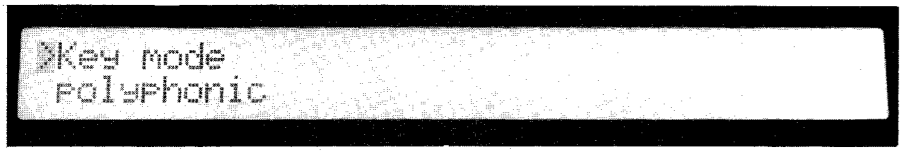


Figure 13-1

This parameter allows us to play a particular voice either *polyphonically* or *monophonically*. Furthermore, each one of these two possibilities presents you with the option of using something called *unison* mode. Therefore, there are four potential options available here: *polyphonic, monophonic, unison poly*, or *unison mono*. The "yes" - "no" buttons or the data entry slider will allow you to choose one of these four modes for every voice you create. In either polyphonic mode ("polyphonic" or "unison poly"), the DX7II allows for multiple notes to be played. In either monophonic mode ("monophonic" or "unison mono"), only one note at a time can be played.

As we've learned, the DX7II has sixteen independent tone generators. In pure "polyphonic" mode, a single voice can use all sixteen, meaning that we have sixteen-note polyphony. Engaging "unison poly" mode, however, will cause a single key depression to use

four tone generators at once. This means that the polyphony is cut by 25% - giving us a maximum of four notes that we can play at any one time. In return for this, the DX7II offers us the opportunity to *detune* three of the four tone generators by a certain small amount. As we learned in Chapter Six, such slight detuning will cause beating effects. This will be even more prominent than the detuning we invoked with edit switch 8, however, since we are detuning *all* the voice components, not just one particular operator within the voice.

In monophonic mode, fifteen of the tone generators simply go on vacation and refuse to "listen" to any voice data coming from the operators. This means that we can only play one note at a time, as the voice assignment system works overtime, constantly sending the data for the most recent note played to the single working tone generator. In unison mono mode, four tone generators are engaged in dealing with the data from a single key depression. As before, we are offered the option of detuning these tone generators from one another when working in unison mono mode - though, as with pure monophonic mode, we can still only play one note at a time.

In both unison modes, the range of the detuning is 0 to 7. A value of 0 means no detuning between the tone generators, and 7 means maximum difference - about a quarter tone, just fast enough to induce rapid beating effects. Try these modes out: initialize your instrument and generate a nice gentle timbre (like perhaps a square wave) with a single modulator-carrier system. Then press edit switch 23. The initialization default is "polyphonic" mode. Play a few licks and note that you can play up to sixteen notes at a time before any voice robbing occurs (judicious use of the sustain pedal will confirm this). Now press the "yes" data entry button in order to change this to "monophonic" mode. Play the keyboard again, and note that you can now play only one note at a time, and that the most recent note you play is always sounded. Now press the "yes" button once again in order to select "unison poly" mode. Here, the display adds another parameter, called *unison detune*: **(see figure 13-2)**

Figure 13-2

Press the right cursor switch in order to position the cursor over this parameter and move the data entry slider in order to change this through its full range of 0 - 7, listening as you do so. Note that entering a value greater than 0 causes a very pleasing beating effect, one which speeds up as you increase the value. This parameter, like operator output level, will not change in real time - you'll need to continuously re-trigger the note in order to completely "enter" the new value (though it changes immediately in the LCD, the microprocessor does not receive the new value until a new note is played). Use your sustain pedal, and note that "unison poly" cuts your total polyphony down to four notes, and that note robbing occurs with the fifth note.

Finally, move the cursor back over the Key mode parameter and press the "yes" button in order to select "unison mono" mode. Once again, changing the Unison detune value has the effect of invoking beating effects, and once again, the speed of the beating increases with

higher values. Just like "unison poly", you'll need to retrigger the notes in order to "enter" these new values. Here, as in pure "monophonic" mode, you can only play one note at a time.

All of which leads to an obvious question: why on earth would you spend over $ 2,000 on a polyphonic synthesizer and use it as a single-voice monophonic synth?

There are actually two potential reasons why. The first of these has to do with the fact that (don't get insulted now!) your keyboard technique may not be absolutely perfect. If you are, for example, trying to simulate a naturally monophonic instrument (and there are many acoustic instruments which can only ever play one note at a time - like all wind instruments) with your DX7II, then playing in polyphonic mode will tend to highlight your flaws in technique. Putting the instrument into one of the monophonic modes, however, will allow the computer to compensate for your own human shortcomings and will ensure that only one note at a time is ever heard:

Exercise 77
Exploring different key modes with the "Piccolo" preset

1) Put your DX7II into single voice play mode and call up the "Piccolo" preset from your ROM cartridge - bank 2, slot 36.

2) This voice has been preset to "polyphonic" key mode. Press edit switch 23 and VIEW the LCD in order to confirm this. Since this is a simulation of a piccolo sound, play a piccolo-like pattern on the keyboard, and try to play it sloppily - pretend that your keyboard technique is less than perfect (even if it *is* perfect!) Listen (Audio Cue 77A).

3) Now change this voice to "monophonic" mode by pressing the "yes" button. Play the same lick again, and do it just as sloppily as before. Listen (Audio Cue 77B). Note that it doesn't sound sloppy at all! Even though you may have pressed more than one note at a given time, only one note sounded, since only one of the sixteen tone generators is active in this key mode.

4) Press the "yes" button again in order to change this voice to "unison poly" mode. Note that the "Unison detune" parameter now appears on the right side of the LCD, and that the default value is 0 (no detuning). Hold down the sustain pedal and play an arpeggiated series of more than five notes. Listen (Audio Cue 77C). Note that after four notes are played, note robbing commences - that is, the fifth note you play causes the first one to drop out, the sixth one causes the second one to drop out, etc.

5) Press the right cursor switch in order to position the cursor over the Unison detune parameter. Change this to the maximum value of 7, play a few notes and listen (Audio Cue 77D). Note that the sound is now extremely detuned with itself, and the result is a very pleasing chorusing effect. Note that the total polyphony is still only four notes. Experiment by changing the Unison detune parameter through its full range of values, playing a note and listening as you do so (Audio Cue 77E). Note that as you bring this value closer to 0, the beating effects slow down and finally disappear altogether at 0. Note also that the Unison detune value does NOT change in real time - you have to re-trigger the note in order for the new value to be "entered". When you are done experimenting, restore the Unison detune value back to 0.

6) Press the left cursor switch in order to position the cursor over the Key mode parameter and press the "yes" button in order to change this to "unison mono" mode. Play a few notes and listen (Audio Cue 77F). Note that no matter how sloppily you play, you only hear one note at a time, so, just as in step 3 above, your technique sounds impeccable!

7) Press the right cursor switch in order to position the cursor over the Unison detune parameter and enter in the maximum value of 7. Play a few notes and listen (Audio Cue 77G). Note that we hear the same pleasing chorusing effect we heard in step 5 above, but that this time we can only play one note at a time. Experiment further by changing the Unison detune parameter through its full range, listening as you do so (Audio Cue 77H). Note that, just as in step 5 above, the beating effect slows down and finally disappears altogether at a value of 0.

The second reason for perhaps wanting to use a voice in either monophonic or unison monophonic mode has to do with some special *portamento* functions, which we will be discussing shortly.

The DX7II offers a second, real-time means for playing any voice or performance memory monophonically, and this is accomplished by pressing the left cursor switch (also labeled "POLY/MONO") while in any play mode. Pressing this switch in play mode will cause the LED status light above it to go on, and it will stay on until you manually press the switch again in order to turn it off. As long as this light is on, any voices or performance memories you call up will be monophonic only. Therefore, this control is *not* voice-specific; it simply allows you to quickly turn off fifteen of the sixteen tone generators for any sound you call up.

Pitch bend wheel controls

We learned in Chapter Eleven that the use of EG bias modulation allows us real-time control over the volume and/or timbre of a sound. At that time we also mentioned that real-time control over the pitch was possible elsewhere and this is precisely the purpose of a device known as the *pitch-bend wheel.*

The pitch-bend wheel dates back to one of the earliest commercially available voltage-controlled analog synthesizers, the MiniMoog. Some fifteen years later, most synth manufacturers still wouldn't dream of making an instrument without one! In an analog synthesizer, the pitch bend wheel is a device which sends either a positive or negative control voltage to the audio oscillators, depending upon whether the wheel is moved up or down from its indented center position. This will have the effect of causing the oscillators to raise or lower the speed of their oscillations, thereby "bending" the pitch, just as a guitarist can raise or lower the pitch of a string by physically bending it.

The DX7II, being a digital synthesizer, wouldn't know what to do with a control voltage if it came up and bit it! But there is a pitch bend wheel on this instrument, nonetheless. Here, the positive and negative voltages generated by the DX7II pitch bend wheel are converted by a built-in *ADC* (analog-to-digital converter; see Chapter One if this concept is unfamiliar to you) into digital data - positive or negative numbers - which are fed to the pitch inputs of the six operators, resulting in yet another form of *pitch modulation.* As with EG- or LFO-induced pitch modulations, this is non-operator specific, for the

same reasons. Therefore, the action of the pitch bend wheel will always affect all six operators simultaneously, and will thus always have the effect of causing a real-time pitch change only.

We are provided with three different pitch bend wheel controls: Range, Step, and Mode, all accessed from edit switch 24. This edit switch also allows us to access two other functions: *Portamento* and *Random pitch,* (discussions of both will follow), so there are actually three different LCD displays that are generated by pressing this switch repeatedly. **(see figures 13-3(a) through 3(c))**

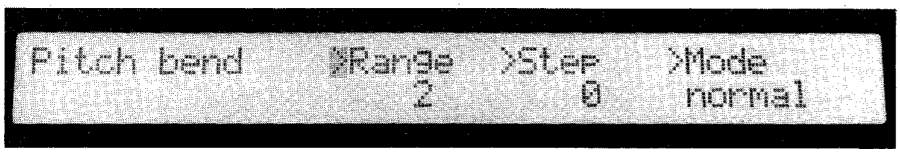


Figure 13-3(a)

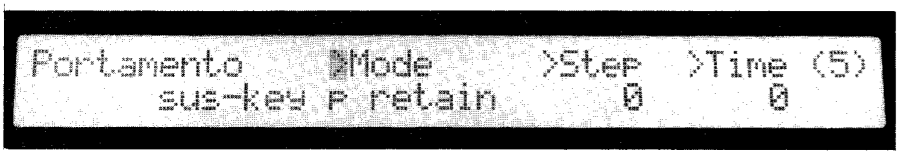


Figure 13-3(b)

Figure 13-3(c)

We'll want to work at this point with the first of these three displays (the Pitch bend display, shown in **figure 13-3(a)**), so initialize your DX7II from single voice play mode, and press edit switch 24 repeatedly until this display comes up. Note that the initialization defaults are as follows: Range = 2; Step = 0; and Mode = normal.

Let's start with the pitch bend Range control. Position your cursor over this parameter and use your data entry slider in order to step this through its potential values of 0 through 12. This number represents the maximum effect of the pitch bend wheel, in *semitones* (that is, *half-steps*). Therefore, entering a value of 1 means that the pitch bend wheel will only cause a pitch change of one semitone (one half-step) in each direction. Enter in a value of 1 here and try it! Similarly, entering a value of 2 will allow it to bend the pitch a full tone (full step) in each direction. Entering the maximum value of 12 will allow it the maximum range of a full octave (12 semitones, or half-steps) in each direction, and entering the minimum value of 0 will cause it to not work at all!

If we decide to enter in the maximum pitch bend Range value of 12, then we have the option of using the Step control. This switch will determine just how the pitch bend wheel changes pitch over its full octave range. Again, we can enter any value from 0 to 12, corresponding to semitones. Entering a value of 1 here, for example, will cause the pitch wheel to change in quantized semi-tone steps; entering a value of 2 will cause a change in full-tone steps. Similarly, entering the maximum value of 12 will cause the pitch bend wheel to jump the pitch from your starting pitch straight up to the octave, with no notes in-between **(see figure 13-4)** as entering the minimum value of 0 will allow for a smooth pitch change that is not quantized at all. **(see figure 13-5)**

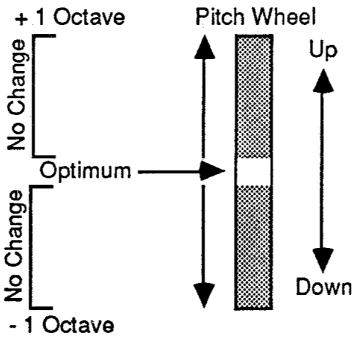


Figure 13-4

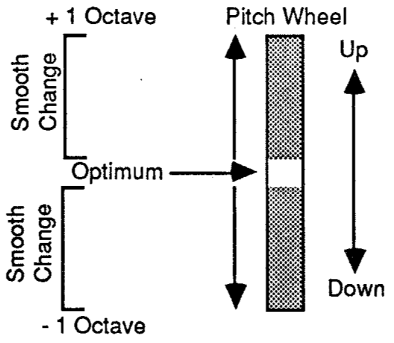


Figure 13-5

Remember that the Step control can only work if the Range control is set at the maximum value of 12. If you try to use it with a Range value of anything other than 0, the DX7II's microprocessor will automatically change the Range to the maximum of 12, and this will be reflected instantly in the LCD display. The converse is also true: if the Step parameter is set at anything other than 0, the Range control will "freeze" at 12. You won't be able to change it at all until you go back and restore the Step to 0. We'll run an Exercise soon to try out these effects, but let's discuss the Mode parameter first.

This parameter offers four potential options: "normal", "low", "high", and "key on". In normal mode, any and all notes held down while the wheel is moved will have their pitch affected. In low mode, only the lowest note in a held chord will have its pitch changed, while in high mode, only the highest note in a held chord will be affected. In key on mode, any notes currently sustaining *through the use of the sustain pedal* will be unaffected, while new notes added to the sustaining ones will have their pitch alterable by the actions of the pitch wheel. As always, the descriptions are often more complicated than the effect itself, so let's run that exercise right now and try all of these out:

Exercise 78
Pitch bend wheel controls

1) Put your DX7II into single voice play mode, and call up the "Fifths" preset from your ROM cartridge, bank 2, slot 40. This voice has two distinct systems, tuned a fifth apart, so we will be able to aurally confirm that the pitch bend wheel is affecting all operators simultaneously.

2) Press edit switch 24 repeatedly if necessary in order to call up the Pitch Bend display. Note that all of these values are at their initialization defaults (since whoever originally programmed this sound didn't change any of them). Once again, these values are: Range = 2; Step = 0; and Mode = normal. Press and hold down a key on the keyboard, and while you are doing so, raise the pitch bend wheel to its full "Up" position, listening as you do so (Audio Cue 78A). Now release the wheel (which automatically returns it to its center indent, since it is a sprung wheel) and play and hold down the same key while lowering the wheel to its fully "Down" position. Listen (Audio Cue 78B). Note that the pitch of the total sound smoothly goes up and down a full tone (that is, two semitones) as the wheel is moved.

3) Position the cursor over the Range parameter and use the data entry section to enter a new value of 1. Play a note and bend the pitch wheel up and down to its maximum and minimum settings and note that our sound now smoothly "bends" in pitch over a range of only a single semitone in each direction (Audio Cue 78C). Experiment by entering in different Range values and moving the pitch bend wheel, noting how the maximum effect of the wheel is changed each time. When you are done experimenting, restore the Range control back to its starting point of 2.

4) Now press the right cursor switch in order to position the cursor over the Step parameter. Use the data entry slider to enter a value of 4. Note that this value is accepted with no problem - but that as soon as it was raised above 0, the Range control automatically changed to 12!

5) Play a note and move the pitch bend wheel all the way to its maximum "Up" position. Listen (Audio Cue 78D). Note that the total sound now bends up a full octave, but NOT smoothly - instead, we hear three distinct changes, four semitones apart. Release the pitch bend

wheel, play the same note, and this time move the pitch bend wheel all the way to its fully "Down" position. Listen (Audio Cue 78E). Note that the same three changes in pitch, in four-semitone increments are heard, but this time they move *down* a full octave. Release the pitch bend wheel and this time play a chord, moving the pitch bend wheel up and down. Listen (Audio Cue 78F). Note that all the notes in the chord are affected equally.

6) Experiment with different Step values and note the effect they have on the sound when the pitch bend wheel is moved. Note also that only at the Step value of 0 is the Range value restored to 2: at all other points, the Range value is 12. With the Step parameter at some value other than 0, position the cursor over the Range parameter and try to change it with the data entry slider. note that, as mentioned above, the Range control "freezes" and cannot be changed if the Step value is greater than 0. When you are done experimenting, restore the Step value back to its starting point of 0.

7) Finally, let's explore the Mode parameter. Because it has remained in its starting "normal" position, so far the movements of the pitch bend wheel have affected all notes in a held chord. Position the cursor over this parameter and press the "yes" button in order to change this to "low". Play a chord and, while holding it down, move the pitch bend wheel back and forth, listening as you do so (Audio Cue 78G). Note that only the lowest note in the chord is affected.

8) Press the "yes" button again in order to change the Mode to "high". Play the same chord as in the last step, hold it down, and move the pitch bend wheel up and down, listening as you do so (Audio Cue 78H). Note that now, only the highest note in the chord is affected.

9) Press the "yes" button one more time in order to change the Mode to "key on". Play the same chord, and this time step on the sustain pedal in order to keep it sustaining. Move the pitch bend wheel, and note that there is no pitch change whatsoever initiated by this action. While keeping your foot on the sustain pedal, play a few new notes, moving the pitch bend wheel as you do so, and listen (Audio Cue 78I). Note that these new added notes *are* affected by the movement of the wheel, while the sustaining chord is not.

Portamento controls

These controls, accessed by pressing edit switch 24 again in order to call up the Portamento display (as shown in **figure 13-3**), all have to do with the portamento effects we can generate on the DX7II. The term "portamento" refers to a smooth glide between notes, similar to that generated by many acoustic instruments (of which the trombone is perhaps the best example). The DX7II also provides for a glide between notes which can be quantized in musical intervals ranging from a semitone up to a full octave. This yields an effect more commonly known as *glissando*. This display offers three parameters: "Mode", "Step", and "Time". Let's begin by examining the Time parameter first. The range of this control is 0 to 99, and whoever designed this was obviously not the person who wrote the EG software for the DX7II. In the EG rates, you may remember, 0 represents the slowest rate and 99 the fastest. With this portamento Time control, however, things are exactly the opposite: a value of 0 is the fastest portamento time and effectively means that there will be no portamento whatsoever (since the instrument will glide from note to note so rapidly, we won't be able to hear the change). A value of 99, on the other hand, is the slowest

portamento time. By the way, the "(5)" you see next to the Time parameter is simply a MIDI controller number, allowing you to alter the portamento Time from an external MIDI controller. This number can be altered with the use of edit switch 31. We'll discuss this in greater detail in Chapter Fifteen ("MIDI").

The DX7II portamento time does not exhibit what is called *constant velocity*; that is, the absolute speed of the portamento is determined solely by the distance that must be traveled. If you play a C1, for example, and then play a C6, any given portamento Time value will take far longer than if you play a C5, followed by a C6. Some other synthesizers have a so-called "constant velocity" control in which the portamento time is absolute, no matter how great the distance. In that event, even though C1 to C6 is a far greater distance than C5 to C6, the glide would take the same absolute amount of time since the microprocessor would simply compensate and make the glide from C1 to C6 faster.

Again, the DX7II does not have this control, so don't worry about it too much.

It's pretty simple to figure out whether the DX7II will glide up to or down to a particular note if you're only playing one note at a time. The instrument's voice assignment system will simply remember whether the previous note was higher or lower than the new note and will glide equivalently. **(see figure 13-6)**

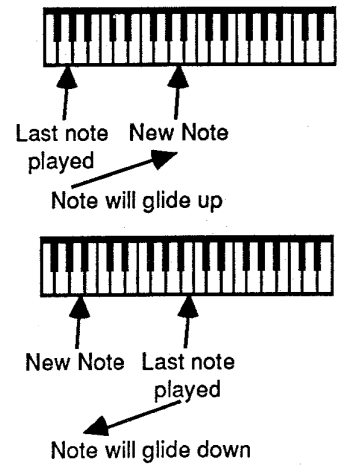


Figure 13-6

Where things get tricky, however, is when we use the glide control polyphonically. Polyphonic glide is a feature not always found on polyphonic synthesizers, and the main reason for this is that the instrument's voice assignment circuitry has to do some pretty fancy footwork in order to decide which notes to glide up to and which to glide down to.

I must confess that, even after many hours spent with this instrument, I have still not completely deciphered the polyphonic portamento logic being used. There are a couple of basic rules, however, that seem to always be followed, and these may help enlighten you:

1) Even though the DX7II thinks there are notes above C6 and below C1 (see Chapter 12), it will nonetheless always glide up to C6 and down to C1.

2) If there are notes currently being held down and you add new notes, the glide will be from the old notes, but only the first time you add these new notes. After that, it's pretty unpredictable. **(see figure 13-7)**

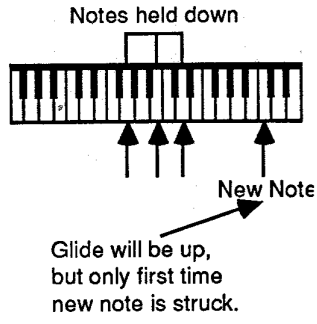


Figure 13-7

3) If you simply strike a single key repeatedly, it won't glide at all (that is, it glides from itself) after the initial key depression. However, if you play a chord repeatedly, the glides will be unpredictable and will almost certainly not be the same every time. **(see figure 13-8)**

Of course, it's entirely possible that the complete portamento process is illogical in places, and that a random "seed" has been planted by the Yamaha programmers in order to keep us honest. Even if this isn't the case, for the sake of your sanity, I advise you to regard portamento that way. If you try to predict whether a given glide will be up or down, I venture that you will be right 50% of the time and wrong the other 50%, over the long haul. So my advice is - don't bother trying.

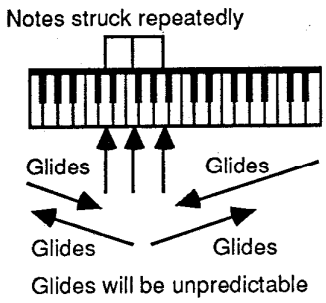


Figure 13-8

As mentioned above, a portamento Time value of 0 will have the effect of inducing no portamento at all - since the fastest portamento time is faster than we mere humans can detect. Larger Time values will give us glide effects that we can clearly hear. Initialize your instrument and press edit switch 24 repeatedly if necessary in order to call up the

Portamento display. Position the cursor over the Time parameter and enter in a value of 75. Play a couple of notes on the keyboard, and - lo and behold! - you won't hear any glide at all.

Why is this? Is the DX7II "broken"? No, not exactly... but we have uncovered a definite flaw in the logic of the instrument. The reason why this won't work with an initialized voice (or, in fact with most presets) has to do with edit switch 27 (labeled FS/CS), which is a *performance* edit switch. What we have here is an unfortunate situation where a performance edit default (which we will be discussed in detail in the next chapter) is affecting a voice default - a bad set of circumstances, indeed! Pressing edit switch 27 will yield four LCD displays. **(see figures 13-9(a) through (d))**

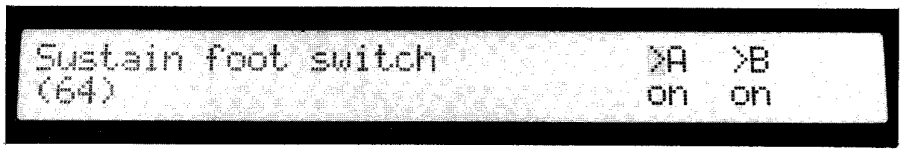


Figure 13-9(a)

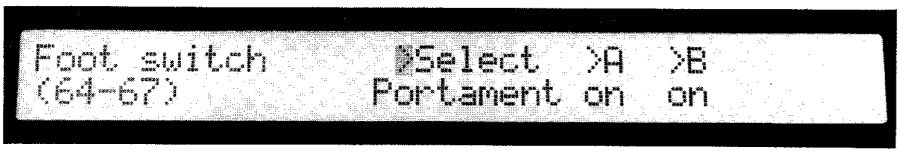


Figure 13-9(b).

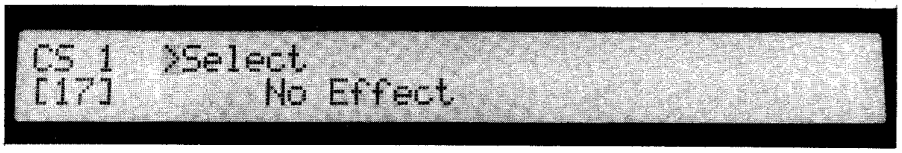


Figure 13-9(c)

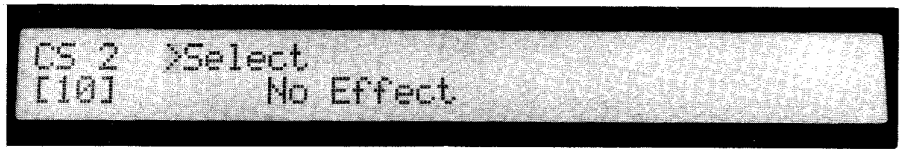


Figure 13-9(d)

The one that we're concerned with here is the display labeled Foot switch (shown in **figure 13-9(b)**). Notice that the initialization default for this is "Portament" for the Select parameter, and ON for both voices A and B. In other words, the DX7II has made this foot switch (which is similar to the sustain pedal but is plugged into the adjacent jack) active as a portamento on-off switch. This means that you won't get any portamento effects at all, no matter what you do with edit switch 24, unless you step on this foot switch. But what if you don't even have one of these foot switches plugged in? Well, this means - believe it or not - you won't *ever* get any portamento effects. That is, unless you turn this effect OFF for the voice you're working with (and in single voice mode - the mode we're in right now - that means voice A).

Therefore, in order to generate any automatic portamento effects (that is, portamento without the use of a foot switch) for any voices, you'll have to first press edit switch 27 and turn the foot switch controls OFF for either voice A, voice B or both (depending upon what play

mode you're in). It's a crazy thing to have to do, I know - but as I've said before, I don't build 'em - I only teach 'em. Even after you've accomplished this, it *won't* be saved when you store the voice because edit switch 27 is a performance edit (as opposed to voice edit) switch. There is, unfortunately, nothing you can do about this.

Let's get some temporary automatic portamento going anyway. Accordingly, turn the foot switch OFF for voice A and return to the Portamento display by pressing edit switch 24 again. Your latest entry of 75 for the portamento Time should still be there. Now play a couple of notes and listen - the portamento effect should now be present. Use the data entry slider to increase or decrease the Time value and note how smaller numbers make for faster glides, while larger numbers do the opposite. Experiment also with the three "rules" of glide outlined above. When you're done experimenting with the Time value, restore it to 75 and let's move on to a discussion of the Step parameter.

Like the Step parameter in the Pitch bend display, the range of this control is 0 to 12, representing semitones. A portamento with a Step value of 0, therefore, will result in a smooth glide - in other words, it will be true portamento. Entering in a value of 1 will cause a glide that occurs in semitones - meaning that the glide will be in the form of a chromatic scale! Entering a value of 2 here will cause a glide in full-step tones, as setting it to the maximum value of 12 will result in a glide that occurs with full octave jumps. These are the *glissando* effects referred to above, with the Step parameter acting as a quantizing control.

With the Time value still at 75, move the cursor over to the Step parameter and change this to a new value of 1. Play a few notes far apart from one another and listen closely - the glide still takes the same time to occur, but you now hear the promised chromatic scales instead of the smooth glide. Slowing down the Time control (by increasing its value) will enable you to hear this even more clearly. Now change the Step value to 2 and play the same notes. At this point, the scales are whole-tone scales. Changing the step value to 6 will result in a scale of tritones, and changing it to 12 will result in full-octave jumps - something you'll be able to hear most clearly if you play C1 followed by C6, or vice versa. Experiment with this control until you feel comfortable with what it does. The musical applications of having Step values other than 0 or 1 may not be obvious - but it's another tool that may end up inspiring new kinds of music!

Finally, we come to the last (or, actually, first) of the portamento controls, the Mode parameter. This determines the action of the *sustain pedal* (NOT the foot switch referred to above) when using portamento effects in either polyphonic or unison poly key mode, or the action of "key on" and "key off" flags, when in monophonic or unison mono key mode. There will therefore be two different Mode options (and, as usual, you cycle between them with the data entry "yes-no" buttons or slider), depending upon which Key mode you selected with edit switch 23. In either polyphonic or unison poly key mode, you'll see either this Mode **(see figure 13-10)** or this one. **(see figure 13-11)**

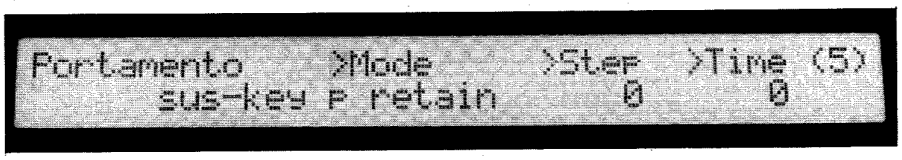


Figure 13-10

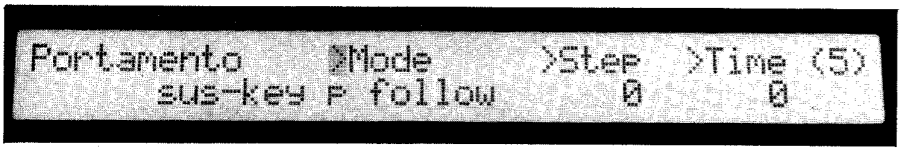


Figure 13-11

In either monophonic or unison mono Key modes, you'll see either this Mode **(see figure 13-12)** or this one.**(see figure 13-13)**

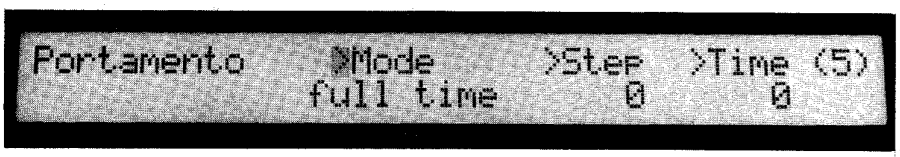


Figure 13-12

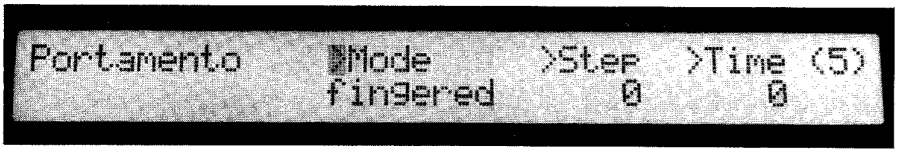


Figure 13-13

Let's discuss the polyphonic options first, since that's the way you'll probably use your DX7II most of the time. As mentioned above, this means that the Mode parameter has to do with the actions of the sustain pedal. As shown in figures 13-10 and 13-11, your two Mode options here are something called "sus-key p retain" and "sus-key p follow". In this instance, "sus" stands for *sustain,* "key" stands for *keyboard,* and "P" stands for *pitch.* So what the LCD is really saying is: "sustain keyboard pitch retain" and "sustain keyboard pitch follow". The difference between these two modes is quite simple, and can best be demonstrated with an exercise:

Exercise 79
Polyphonic portamento modes

1) Put your DX7II into single voice play mode and call up the "Warm Stg A" preset from your ROM cartridge (bank 1, slot 1). Make sure your sustain pedal is plugged into the correct jack on the back of your machine.

2) Press edit switch 27 repeatedly if necessary in order to call up the Foot switch display. **(see figure 13-14)** Position the cursor over the "voice A" parameter and press the "no" button in order to turn this OFF. Because we are in single voice play mode, this will have the effect of de-activating the Foot switch (which you may or may not actually own or be using) for portamento effects, allowing us to generate an automatic glide with edit switch 24.

Figure 13-14

3) Press edit switch 23 and observe that this voice was programmed in "polyphonic" Key mode, meaning that we have full 16-note capability.

4) Press edit switch 24 repeatedly if necessary in order to call up the Portamento display. Note that the Mode is currently at "sus-key p retain". Leave it that way and press the right cursor switch in order to position the cursor over the Step parameter (currently at 0). Press the "yes" button in order to enter a new value of 1. This will give us a quantized, glissando effect. Press the right cursor switch again in order to position the cursor over the Time parameter and enter in a new value of 83.

5) Play a three- or four-note chord in the bottom octave of your keyboard and listen (Audio Cue 79A). Note that some notes may glide up to their pitch and others may glide down - but that they all glide in half-steps, or semitones, since we have set the Step value at 1. Keep the chord held down and depress the sustain pedal. Note that the sound continues at the same volume. This is because L3 in all the carrier EGs is quite high.

6) While keeping your foot on the sustain pedal, and while still hearing the previous chord, play another three- or four-note chord, this time in the top octave of the keyboard and listen (Audio Cue 79B). Note that we continue to hear the old chord even as the new chord glides up to its pitches and sustains. The "sus-key p retain" option, then, ensures that we will always hear the most recent notes played (in this case, the 16 most recent notes played, since this voice is in polyphonic Key mode. If it were in unison poly Key mode, we'd only hear the four most recent notes played), regardless of any portamento or glissando effects.

7) Release the sustain pedal. Press the "yes" button in order to change the Mode to "sus-key p follow".

8) Repeat steps 5 and 6 above and listen (Audio Cue 79C). Note that this time, the new chord *follows* from the old one, and that previous notes are NOT retained. With this setting, we will only ever hear the most recent note or notes (if they are played simultaneously) that are played, and we can be certain that they will always glide from the previous note or notes played.

9) Experiment further by adding in new notes or two- or three-note chords to held chords in "sus-key p follow" Mode. Note that this Mode enables you to glide a single note from a multi-note held chord, or a multi-note chord from a single held note! This can make for some very striking musical effects (Audio Cue 79D).

Note that the portamento Mode controls in polyphonic mode ("sus-key p retain" and "sus-key p follow") have no effect whatsoever if you are not actually sustaining notes or chords by the act of physically stepping on the sustain pedal.

Finally, let's discuss the portamento Mode options available to us in either monophonic or unison mono Key mode. Here, we have a choice of either "full time" or "fingered" portamento. The difference between these has to do essentially with whether the DX7II microprocessor senses a new "key on" flag (see Chapter Nine for more on this) before it senses a "key off" flag. In other words, it poses the question: Are you still holding down a key when you play another key, or not? In "full time" Mode, it won't matter. You'll get whatever glide you programmed with the Time and Step parameters. In "fingered" Mode, however, it

matters a lot. If a "key off" is detected between "key ons" (in other words, if you play notes staccato), then the portamento Time is assumed to be 0, giving you no glide effect at all. On the other hand, if two "key on" flags are detected *without* a "key off" in-between (in other words, if you play notes legato), then the portamento Time and Step is whatever you programmed it to be. This is another one of these things that's easier to hear than to explain, so let's just run an exercise and try it:

Exercise 80
Monophonic portamento modes

1) As before, put your DX7II into single voice play mode and call up the "Warm Stg A" preset from your ROM cartridge (bank 1, slot 1).

2) Press edit switch 27 repeatedly if necessary in order to call up the Foot switch display, as shown in figure 13-14 above. Position the cursor over the "voice A" parameter and press the "no" button in order to turn this OFF. Because we are in single voice play mode, this will have the effect of de-activating the Foot switch (which you may or may not actually own or be using) for portamento effects, allowing us to generate an automatic glide with edit switch 24.

3) Press edit switch 23 and press the "yes" button in order to change the Key mode to "monophonic".

4) Press edit switch 24 repeatedly if necessary in order to call up the Portamento display. Note that the Mode is now "fingered". Position the cursor over the Mode parameter and press the "yes" button in order to change this to "full time".

5) Press the right cursor switch twice in order to position the cursor over the Time control and enter a value of 83. Leave the Step value at 0. This will give us a smooth, non-quantized portamento effect. Play a few notes on the keyboard and listen (Audio Cue 80A). Note that regardless of which notes you play, and whether they are played individually (staccato) or smoothly (legato), we always hear the same portamento effects.

6) Position the cursor over the Mode parameter and press the "no" button in order to change back to "fingered" Mode. Now play several different, individually articulated notes on the keyboard, taking care to release each before you play the next one, and listen (Audio Cue 80B). Note that we now have no glide whatsoever. Play the same series of notes again, but this time take care *not* to release a key before playing the next key - in other words, play this series of notes legato. Listen (Audio Cue 80C). Note that we now hear the same glide effect we heard in step 5 above. In either monophonic or unison mono Key mode, then, the portamento Mode control allows us to get portamento or glissando effects, but only if we play notes legato. This will allow you to bring these effects in and out solely by your keyboard technique. With a little practice, you'll find that you can bring them in and out at will (Audio Cue 80D).

The use of the "fingered" portamento Mode, then, is the other reason why you might want to occasionally set up a DX7II voice in a monophonic key mode, since this control is not available when working polyphonically. "Fingered" portamento is also not available when you play a voice monophonically by using the POLY/MONO switch on the left side of the instrument - in other words, the voice itself must be programmed in either monophonic or unison mono mode for this effect to be available.

Random pitch

This is a very simple, but very clever edit parameter, also available from edit switch 24 (the Random pitch display as shown above, in **figure 3(c)**). The only parameter available in this display is labeled "Depth", and the range of this parameter is 0 to 7. The purpose of this control is to allow the DX7II to more closely simulate the slight pitch differentials that occur when the same note is played repeatedly on many acoustic instruments. This effect is particularly noticeable in wind instruments or non-fretted stringed instruments, where slight imperfections in the player's intonations will result in the same note having a very slightly different pitch each time it is played. Increasing the Depth of this Random pitch control will cause the voice you are using to yield a slightly different pitch each time you play the same note, to a greater degree. A Depth value of 0 will, of course, result in no such pitch shift, and the maximum Depth of 7 will result in the pitch being shifted by nearly a semitone. For most acoustic instrument simulations, however, a depth of 1 or 2 is most realistic. This control really does help make the DX7II more "acoustic"-sounding, and hats should be doffed to Yamaha's engineers for including this subtle but important control here.

Pitch bias

In Chapters Ten and Eleven, we covered most of the parameters offered by edit switches 25 and 26. These switches allow for the programming of the DX7II's *real-time controllers*, including the breath controller, keyboard After touch, the modulation wheel, the two foot controllers (FC1 and FC2) and an external MIDI IN controller (which will be covered in greater detail in Chapter Fifteen). We saw how each of these could be used to route either LFO pitch or amplitude modulation, or EG bias modulation. However, two of these controllers - the breath controller and keyboard After touch - can also be used for another kind of modulation, called *pitch bias*.

This control, like the pitch bend wheel, allows for real time control over pitch, but allows you to apply this change by either blowing harder into the breath controller or by pressing down harder on the key (using the monophonic After touch control). Initialize your DX7II from single voice play mode and press edit switch 25 repeatedly if necessary in order to call up the Breath control display (see **figure 13-15**) and the After touch display. **(see figure 13-16)**

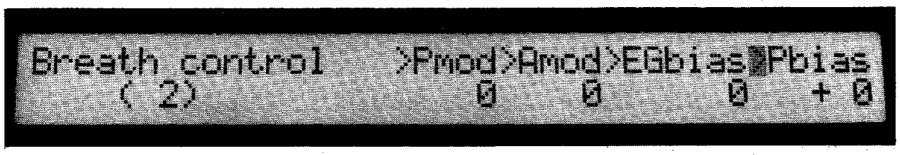


Figure 13-15

Figure 13-16

The pitch bias is set with the "Pbias" parameter on the far right of each of these displays. The initialization default, as you can see, is +0 for both the breath controller and the After touch. This indicates that no pitch change will be initiated by either controller. Try pressing down harder on a key or blowing hard into your breath controller (if you have one) and you will find this to be true. The range of the Pbias parameter is -50 to +50, with negative values yielding a flattening of pitch, and positive values yielding a sharpening. A Pbias value of -50 will actually cause a pitch change of four full octaves down if maximum signal is routed through the appropriate controller, while a value of +50 will cause the pitch to rise up to four full octaves above nominal as you blow or press harder. The tables of data that the DX7II uses to generate these pitch bias changes use exponential curves, so that a value of +25 will NOT yield a rise of two octaves - but a rise instead that is closer to one octave.

We can easily test this out. Call up the After touch display and position the cursor over the Pbias parameter. Leave it at its initialization default of +0 for the moment, but press down on a key with maximum reasonable force and keep it held down. Now press the "yes" button repeatedly in order to step through the full range of +0 to +50. Note that the pitch change per increment is very subtle at first, but increases drastically per increment as you enter in higher and higher values. The same phenomenon occurs when entering in negative Pbias values. Therefore, Pbias can be used for very subtle and slight pitch changes - well within a semitone - in its lowest values, and for very gross pitch changes in its highest values.

Many of the ROM presets (like "PickGuitar", bank 1, slot 3; or "SongFlute", bank 1, slot 16) use pitch bias effects linked to keyboard After touch. You can, of course, alter any of these so that the amount of pitch change is different, or you can link the pitch bias effects to the breath controller, if you prefer.

As with the portamento Time parameter, numbers in parentheses appear in all of the edit switch 25 and 26 displays (with the exception of After touch). These numbers, as we mentioned before, are MIDI controller numbers, allowing for external MIDI controllers to be linked to any of these on-board real-time controllers. Note that FC1 and FC2, as well as the MIDI IN controller, can be used to control overall volume (with a range of 0 - 99; 0 meaning no effect, and 99 meaning full effect over the complete dynamic range) as well as their more usual Pmod, Amod, and EGbias functions.

This concludes our discussion of all the DX7II voice editing parameters. This is probably a good time to go back and review all that we have covered, so that you'll be ready to move on now to a discussion of how to combine individual voices into performance memories and how to use the performance edit parameters.

Chapter Fourteen ◇ Performance Controls

Here's where the fun really begins! After you have used all of the information in the preceding pages to modify your DX7II voices to your absolute satisfaction, any one or two of them can be combined in a coherent way into a *performance memory*.

We briefly discussed the meaning of this term way back in Chapter Two, and in this chapter, we'll be going into much greater detail as to what a performance memory actually is. Essentially, it consists of one or two voices arranged over the keyboard in a particular voice mode (that is, *single, dual*, or *split*), plus a number of extra "icing-on-the-cake" parameters called *performance edit parameters*.

All of the performance edit parameters are accessed from edit switches 27, 28, 29, and 30, and these include foot switch and continuous slider (CS) assignments; micro tuning and stereo panning parameters; and a number of extra functions including performance memory naming, overall volume, balance and detuning between voices, split point (if in split mode), transpositions between voices (called "Note shift"), and something called *EG forced damp*, related to the actions of the operator envelope generators. We'll examine all of these parameters here in great detail, but before we do, let's discuss exactly how to get into performance edit mode.

We enter voice edit mode, as you should remember, from any one of the three voice play modes (single, split, or dual). Logic dictates that we can therefore only enter performance edit mode from performance play mode. Technically, that is correct. However, we *can* access any of the performance edit parameters (via edit switches 27 through 30) from either performance edit mode *or* voice edit mode! However, two factors make it inadvisable to change any of these parameters from voice edit mode:

1) All of the performance edit parameters automatically reset to particular initialization defaults* whenever you go to voice play mode - therefore, you will always have to work with completely initialized performance edit parameters in voice edit mode.

2) You can't *save* any changes made to performance edit parameters if they were entered while in voice edit mode. In other words, you can't ever update or overwrite any performance memories from voice edit mode.

* You can also initialize the performance memory edit functions only (that is, the parameters of edit switches 27 through 30) with the Initialization display of edit switch 14, if the cursor is positioned over the "Performance", rather than the "Voice" parameter we first used way back in Chapter Four.

The existence of a performance Recall edit function, similar to the voice Recall edit function discussed in Chapter Eight, means that if you accidentally change any performance edit parameters while in voice edit mode, you *can* ultimately retrieve these changes later on. However, the fact remains that you still won't be able to store these alterations until you get yourself into performance edit mode - and this is accomplished simply by going into performance play mode (by pressing the "PERFORMANCE" switch) and then pressing the edit button. Remember, the DX7II is very fussy as to which play mode you last used before entering edit mode - and that will be the only play mode you can return to from edit mode. All of this was covered in Chapter Two, so if any of it seems fuzzy now, this would probably be a good time to go back and reread that chapter.

You should also be aware of the fact that, if you make a change to a performance edit parameter from voice edit mode, you'll find that when you return to voice play mode, the decimal point stays in the LED no matter which voices you call up! This is a reminder from our friendly microprocessor that something has been changed that hasn't been stored. There are only two ways to get rid of the decimal point if this happens. The first is to put the DX7II into performance play mode and to then press any one of the thirty-two main switches. This "tells" the microprocessor that you really didn't want to save those changes. The second method is to actually save them - again from performance play mode. All of which brings us to the performance memory *save* function.

You save performance memories to either internal or RAM cartridge memory with virtually the same procedure used to store voice data (as discussed in Chapter Eight). The only difference is that you start out in performance play mode (as opposed to one of the three voice play modes) before pressing and holding down the pink Store button. From this point on, the procedure is identical - you press the main switch corresponding to the slot you want to store your data in, and you confirm by pressing the "yes" button with the pinky of your left hand. Here, of course, the "1-32/33-64" switch will not work, since there are only thirty-two performance memory slots. The owner's manual gives a fairly comprehensive description of this procedure, so we won't go into detail here. Suffice it to say that if you know how to store voices, you'll know how to store performance memories.

The first, and perhaps most important thing that the performance memory remembers for you is the voice mode and the particular number(s) of the voice or voices you choose. This can be any combination of one or two internal and/or cartridge voices. It's important to realize that *only* the numbers are remembered. In other words, you can call up a performance memory that, for example, recalls that you worked in split mode with internal voice 22 and cartridge voice 54. If you later change internal voice 22 substantially, or if you have a different ROM or RAM cartridge in the slot, then calling up that performance memory will *not* restore the original voices. It will instead simply call up whatever voice it finds in internal slot 22 and whatever voice it finds in cartridge slot 54. That's the main reason why performance memories are stored in the same area of internal or cartridge memory as are voices.

Whenever you enter performance play mode, you are placed into a special area of memory called the performance *edit buffer*. Like the voice edit buffer we first discovered in Chapter Two, this is a temporary "scratch-pad" area, where changes can be made to your heart's content

without anything actually being altered in stored performance memories. A special phenomenon occurs when you first enter performance mode from any of the three voice modes, however. Until you actually press one of the thirty-two main switches, the voice or voices you last worked with will be in the performance edit buffer - and all performance edit parameters (that is, those accessed with switches 27 through 30) will be at their initialization defaults. This feature allows you to immediately begin building a performance memory from the most recent work you did in voice mode. Try it out: put your DX7II in split voice play mode, and select internal voice 46 for voice A, and internal voice 3 for voice B. Now press the performance mode select switch in order to put yourself into performance play mode (the "split" LED status light will stay lit, and the "performance" status light will come on). A quick look at your display will show something similar to **figure 14-1.**

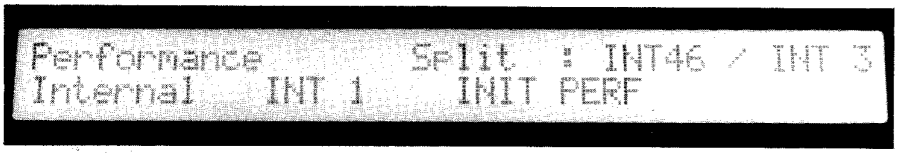


Figure 14-1

The "A" box of the LED will show either "Pi" or "Pc", indicating that you are in performance mode (the "i" stands for "internal" and the "c" stands for "cartridge". This will correspond to the most recent performance play memory you accessed). The LCD display shows "split mode", with internal voice 45 being shown as voice A, and internal voice 3 being shown as voice B. The performance name is always "INIT PERF", telling you that the performance edit parameters have been initialized. Most importantly, if you play the instrument, you will hear that nothing about the voices has changed in any way. You can experiment by entering performance play mode from any of the three voice play modes, with any voice or voices selected, and you'll see that the voice mode as well as the voice numbers are always retained when you first enter performance play mode.

If you then press any of the main thirty-two switches (and remember that the "1-32/33-64" switch won't work here, since there are only 32 available performance memory slots), then your "INIT PERF" will be replaced by whatever performance memory data is found in the internal or cartridge slot you just chose. As with voice memories, you can access performance memories from either the internal or cartridge memory, simply by pressing either the "internal" or the "cartridge" switch.

But if you *don't* press any of the thirty-two main switches and press the "edit" switch instead, then you enter performance edit mode, and you are ready to begin creating a performance memory from scratch. In summary, the technique for creating a performance memory from scratch is as follows: simply place whatever voice or voices you want to use in the edit buffer in one of the three voice play modes - and then press the "performance" switch.

One word of caution here: you unfortunately *cannot* change your mind about which voice or voices you want to use in performance edit mode. This decision can only be made in voice play mode. If you therefore enter performance edit mode, and want to hear your altered performance edit parameters applied to a different voice or voices, you are out of luck. At that point, you only have two options available:

either start from scratch, re-entering voice play mode and choosing a different voice or voices, or store the performance memory as is, and, since the performance memory remembers only the *numbers* of the voices, shift the new voices you want to use into the same voice slot numbers you selected when you first started. Both solutions are tedious, to say the least, and we can only hope that future ROM updates of the DX7II will allow you the freedom to change your voices from within performance edit mode.

Let's begin our discussion of this mode with the parameters offered by edit switch 27, labeled "FS/CS", for "Foot Switches" and "Continuous Sliders".

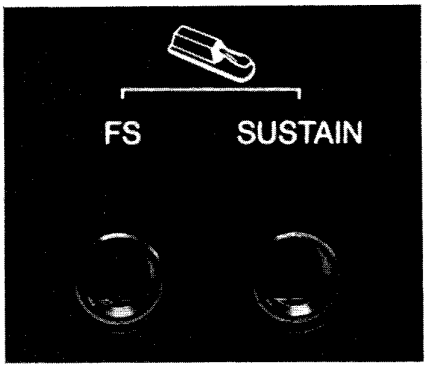


Figure 14-2

Foot switches

The DX7II provides for the use of up to two foot switches, both physically identical and looking very much like a piano-type sustain pedal. The use of each of these is determined solely by which foot switch jack you plug it into on the rear panel of the instrument. **(see figure 14-2)**

Edit switch 27 allows you to determine the functions of the foot switches, and also to specify whether voice A, voice B, or both, will be affected by each of these. We first came across this edit switch in the last chapter, in our discussion of portamento effects. As we learned, pressing this switch repeatedly will yield the four LCD displays shown in **figures 14-3(a) through 3(d).**

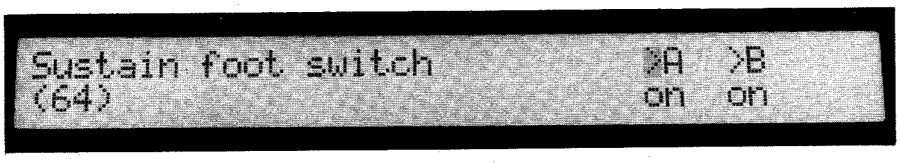


Figure 14-3(a)

Figure 14-3(b)

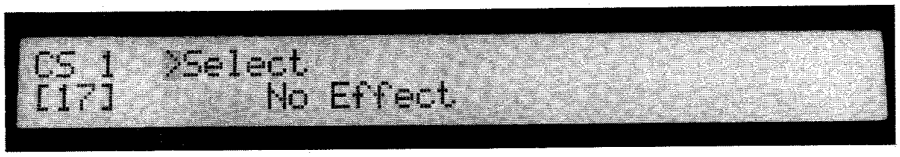


Figure 14-3(c)

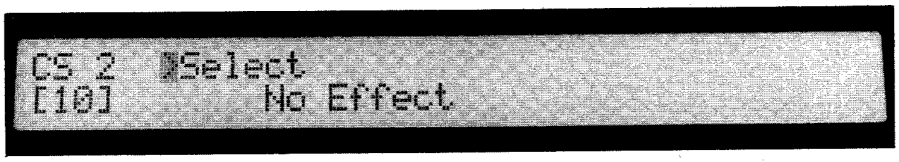


Figure 14-3(d)

Sustain foot switch (FS1)

If a foot switch is plugged into the "sustain" jack, it will always act as a sustain foot switch. We talked about the function of this controller back in Chapter Nine. The Sustain foot switch display (as shown in **figure 14-3(a)**) simply allows you to specify whether the sustain foot

switch will affect voice A, voice B, both, or neither. This is accomplished simply by positioning the cursor over the "voice A" or "voice B" parameters and using the "yes-no" buttons or data entry slider to turn the foot switch ON or OFF for each voice. Of course, if you have created a performance memory from single voice play mode, you will only be hearing voice A, so turning the sustain foot switch ON or OFF for voice B will have no effect.

Let's run a simple exercise now in order to hear the effects of the sustain foot switch.

Exercise 81
The sustain foot switch

1) Place your DX7II into dual voice play mode, and select "TouchOrgan" (ROM cartridge, bank 1, slot 12) for voice A, and "FMilters" (ROM cartridge, bank 1, slot 7) for voice B. Now press the performance mode select switch in order to enter the performance edit buffer. Your LCD should currently look like **figure 14-4.**

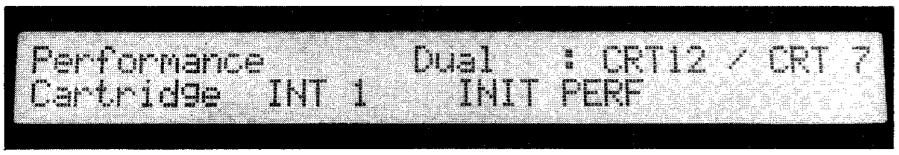


Figure 14-4

2) If you are working in stereo (that is, if you are routing the voice A and voice B outputs to different speakers via your external mixer/amplifier setup), press the "PAN" switch (so that its status light goes on) in order to separate the voice A and voice B outputs. This will help you hear the sustain pedal effect we will be generating more clearly. If you are only working monophonically, don't worry too much about this.

3) Press the edit switch in order to enter performance edit mode. The most recent edit parameter (be it a voice edit parameter or a performance edit parameter) will instantly be called up in your LCD. Press edit switch 27 repeatedly if necessary in order to call up the Sustain foot switch display. note that the initialization defaults for this parameter is voice A = ON and voice B = ON. In other words, whenever you initialize the performance memory (either from edit switch 14 or because you have entered performance play mode directly from voice play mode without pressing any of the thirty-two main switches), the sustain foot switch will be active for both voice A and voice B.

4) Step on the sustain foot switch and keep it depressed. Play a chord and listen (Audio Cue 81A). Note that both sounds sustain as long as you keep your foot on the pedal (since the carriers for both of these voices all have L3 values of greater than 0).

5) Position the cursor over the "voice A" parameter and press the "no" button in order to turn this OFF. Now only voice B - "FMilters" will sustain if you depress the foot switch, while voice A - "TouchOrgan" will not. Step on the sustain foot switch, and, while keeping it held down, play a held chord. Remove your fingers from the keys, but continue to keep the sustain pedal depressed. listen (Audio Cue 81B). Note that the "TouchOrgan" sound disappears almost instantly (since the R4 value for its carriers is very high), while the "FMilters" sound continues as long as you keep the sustain foot switch depressed.

6) Now let's try the opposite effect. Restore the voice A parameter to ON, press the right cursor switch, and change the voice B parameter to OFF. Step on the sustain foot switch, play a chord, remove your fingers from the keys and listen (Audio Cue 81C). Note that the "TouchOrgan" sound now continues as long as you keep your foot on the switch, while "FMilters" leaves instantly (since R4 for its carriers is also very high).

7) Finally, turn the voice B parameter OFF so that the sustain foot switch will have no effect on either voice. Depress it, play a chord, remove your fingers from the keys and listen (Audio Cue 81D). This is just like having no sustain foot switch at all!

This control, then, allows you to set up performance memories so that the sustain pedal may only effect one voice or the other, or may effect neither. The parentheses in the Sustain foot switch display is showing, as usual, the MIDI controller number assigned this function if an external controller is being used. We'll talk more about this in the next chapter ("MIDI").

Next, let's look at the various uses of the foot switch if it is plugged into the generic "FS" jack on the back of the instrument. Used this way, Yamaha refers to it as FS2 (as opposed to FS1, which is the sustain foot switch).

Foot switch (FS2)

If you are only using one foot switch, start by plugging it into the "FS" jack. Then select any single voice (pick a non-percussive, sustaining one) and then press the performance mode select switch in order to enter performance play mode. Next, press the edit mode select switch in order to go into initialized performance edit mode. As we learned earlier, this has the action of resetting all of the performance edit parameters to their default values. Press edit switch 27 repeatedly if necessary in order to call up the Foot switch display, as shown in **figure 14-3(b)**. Position the cursor over the "Select" parameter and note that the default application for this foot switch (as we saw in the last chapter) is for portamento (the display reads "Portament", simply because there isn't room for that last "o"!). We'll see shortly that this foot switch can also perform any one of three other functions. Press the "no" button and observe that the switch can also be used as a sustain pedal. Why this duplication of labor? Simple - if you only own one foot switch, it really makes more sense to plug it into the "FS" jack than the "SUSTAIN" jack, since you can use it for sustain functions *or* for one of the other three functions. Press the "yes" button twice and note that FS2 can also be used for a "Key hold" function (to be explained momentarily), and press the "yes" button one more time in order to view the final foot switch function, called "Soft". As with the sustain footswitch, we can also make this active for voice A, voice B, both, or neither (by positioning the cursor over the "voice A" or "voice B" parameters and using the data entry section to turn them ON or OFF.

In Chapter Thirteen, we discussed this controller's portamento functions. By simply entering in some portamento Time value greater than 0 with edit switch 24, we can bring glide effects in and out by depressing the foot switch. When used as a Sustain foot switch, this controller operates exactly the same way as the "true" sustain foot switch, so there's nothing exciting to report here, either. The next two functions, however, are indeed innovative. The "Key hold" function

allows you to use this foot switch as a pianist uses a *sostenuto* pedal. For those of you who have never used one (it's the rarely-used and often misunderstood center pedal of a grand piano), this type of pedal will allow you to sustain only those notes whose keys are physically held down when the pedal is depressed. New notes can then be played over the old ones - and they *don't* sustain. Try it! Select the "Key hold" function and play a chord. While keeping the keys depressed, step on the foot switch. Note that they continue to sustain (if you've picked a sound whose carrier L3s are greater than 0) as long as your foot remains on the switch. Now, while keeping the switch depressed, add in some new notes and you'll find that they *don't* sustain. When you release the pedal, all the notes stop sounding (as the "all keys off" command is sent).

The "Soft" function of the foot switch is also unique. Position the cursor over the Select parameter and press the "yes" button one more time in order to call up this function. Note that a new parameter, called "Range" now appears on the right side of your LCD. **(see figure 14-5)**

Figure 14-5

Increasing this value (it goes from 0 to 7) will give the pedal a greater effect. As with the "Key hold" function, this is meant to emulate the effect of another piano pedal, the left-most pedal, most commonly called the *soft pedal.* In the grand piano, this has the effect of shifting the hammers so that they strike two strings instead of three (or, for some notes, one string instead of two). This will have the effect of lessening the volume, and to a smaller degree, altering the timbre of the piano sound. The DX7II, of course, has no hammers to be moved (or if it does, I couldn't find any!), so what this pedal does is to attenuate (that is, lessen) the output level of *all the modulators* in the algorithm (or algorithms) currently in use by the voice or voices being played. It also lessens the output level of the carriers, but by a much smaller degree. The effect of the foot switch in this mode can be mapped out as follows:

Range	Attenuation of modulator output levels
7	- 21
6	- 14
5	- 11
4	- 6
3	- 5
2	- 3
1	- 2

Setting a Range of 0 will, of course, cause no attenuation of operator output level at all. Let's run a short exercise now to try out the use of FS2 as a Soft pedal.

Exercise 82
The soft pedal

1) Plug your foot switch into the "FS" jack on the rear of your DX7II. Put your instrument into split voice play mode and call up "EbonyIvory" (ROM cartridge, bank 1, slot 9) for voice A and "St.Elmo's" (ROM cartridge, bank 1, slot 20) for voice B. This puts the piano-like

"EbonyIvory" voice on the keys C3 and lower, and the more esoteric "St.Elmo's" on all the keys above C3.

2) Press the Performance mode select switch. Note that the same two sounds are still in the edit buffer, and that the LCD shows their numbers and the fact that you are still in split mode. Press the edit mode select switch, putting us into initialized performance edit mode.

3) Press edit switch 27 repeatedly if necessary in order to call up the Foot switch display. Position the cursor over the Select parameter and change this from its default of "Portament" to "Soft". Leave both voice A and voice B ON. Press the right cursor switch three times in order to position the cursor over the Range parameter and change this to its maximum of 7.

4) Without stepping on the foot switch, play a few notes on both sides of the keyboard and listen (Audio Cue 82A). Note the specific timbres of both sounds. Now step on the foot switch and, while keeping it depressed, play the same notes on both sides of the keyboard and listen (Audio Cue 82B). Note the way both sounds soften a bit in volume but lose a great deal of their brilliance, since the effect of the Soft pedal is to lower the output levels of all modulators (because, in this particular case, the range is 7, all the modulators have had their outputs attenuated by 21 increments). Note also that the relative timbral and volume difference between the two sounds remains the same: the "St.Elmo's" sound remains slightly brighter than the "EbonyIvory" one, since it was brighter to begin with.

5) Change the Range of this effect to an intermediate value of 3. Play a few notes without stepping on the foot switch and note that they are no different than the way they originally were. This is because the foot switch has absolutely no effect on the sound unless it is depressed. Now step on the foot switch, and, keeping it held down, play the same notes and listen (Audio Cue 82C). Note that the effect is much less severe - that the volume remains virtually the same but that there are a few less overtones in the sound (since here the output levels of the modulators have only been attenuated by 5 increments). Note that, once again, the relative timbral and volume difference between the two sounds remains the same.

6) Experiment by returning to performance play mode (by pressing the performance mode select switch) and then going to dual voice play mode in order to select these same two voices in dual mode. Return to performance edit mode and repeat steps 3 through 5, noting how stepping on the soft pedal changes the entire sound. Experiment further by turning this effect on for voice A only, then for voice B only, and note how this affects the overall sound.

The four numbers ("64-67") in parentheses in the lower left-hand corner of your LCD display refer, as usual, to the MIDI controller numbers assigned to FS2. In this case, there are four numbers since this controller can conceivably serve four different functions.

Let's move on now to a discussion of a new dimension in real-time control: the continuous slider controllers, labeled "CS1" and "CS2".

The Continuous Sliders

The idea of using these kinds of controllers originated with Yamaha's KX series of MIDI keyboards. Being performance edit parameters, they are obviously intended as real-time performance controls. In a nutshell, the use of these two sliders allows you to

change just about any voice editing parameter (as well as several performance edit parameters) directly from performance play mode.

The continuous sliders themselves are, as mentioned in our "guided tour" (Chapter Two), located to the right of the volume slider. We've actually been using CS2 all along, because it moonlights as the data entry slider when the DX7II is in either voice or performance edit mode. In play mode, however, it changes back to its alter ego of CS2. CS1 is the middle slider, and serves no function at all in voice edit mode, though it acts as a voice A/voice B balance control while in performance edit mode.

The CS functions, like both foot switch functions, are accessed from edit switch 27. Pressing the switch repeatedly will bring you the CS1 and CS2 displays, as shown in **figures 14-3(c)** and **14-3(d)**. Like the Foot switch display, the parameters here are simply a Select function and, for most (but not all) variables, a voice A and voice B ON/OFF function. These latter two allow us to specify that the continuous slider in question will affect either or both of the two voices you may have selected. If you have built a performance memory from a single voice, of course, there will be no voice B, so turning the CS ON or OFF for voice B will have no affect at all.

The two continuous sliders can perform identical operations: it's simply up to you to decide which one you want to assign to which function. In other words, there is nothing you can do with CS1 that you can't do with CS2, or vice versa. Remember that even though you can use these controllers to alter voice edit parameters, they will not have any effect whatsoever in voice play mode. To use them, your voice or voices must be organized into a performance memory. It's worth noting again that CS1 always acts solely as a voice A/voice B balance control whenever the DX7II is in performance edit mode, and that CS2, of course, acts as the data entry slider while in this mode. This means that you unfortunately cannot test out any of these continuous slider functions while in performance edit mode: you'll need to return to performance play mode in order to hear their effects. This doesn't mean that you necessarily have to store the sound, however - you can simply press the performance switch in order to return to a "pre-storage" performance play state, and the continuous sliders will then become active for whatever function you assigned them. At this point, if you like what you hear, you should store the performance memory - either back into the same slot (if you are simply updating the memory) or into a different one.

The following is a table of the different parameters that either continuous slider can alter in real time while the instrument is in performance play mode: these are the variables of the Select parameter:

1. No effect - you'll enter this value if you want to simply disable the CS. This is the performance initialization default for both voice A and voice B - meaning that neither continuous slider will have any effect until you assign a function with the Select parameter.
2. Total volume - this causes the CS to double as a volume slider. The only potential reason for selecting this value is if you are linking an external controller to one of the continuous sliders via MIDI (more about this in the next chapter).
3. Output balance (A/B) - here, putting the slider all the way up will give you voice A only, while having it all the way down will give you voice B only. The middle setting will give you an equal balance of both, and, of course, positions in-between will allow you to blend the

two voices together any way you want. If you have constructed a performance memory from a single voice, this use of the CS will have no effect. It's also worth noting that CS1 always automatically takes on this Balance function whenever you are in initialized performance edit mode, no matter what Select value you enter. Again, you can only hear the CS effects in performance *play* mode.

4. Pan select - steps through the four different Pan modes (to be covered in detail later in this chapter).

5. Dual detune - as the slider is moved up, the pitch of both voices will shift very slightly (to a maximum range of nearly a semitone). Voice A will go a bit sharp, while voice B will go a bit flat. This allows you to bring in very subtle beating effects if you are working in dual mode with the same or similar sounds assigned to both voice A and B. This control is nearly, but not quite a real time control - you'll need to re-trigger notes in order to "enter" in any changes you make to the position of the CS when used for this function. It's not recommended that you use this function with a performance memory built from split mode, or you'll end up with two different voices that are simply slightly out of tune with one another, and, of course, this will have no effect whatsoever on a performance memory built from a single voice.

6. Algorithm - allows you to change the algorithm of either voice A or voice B (or both) in real time. This will allow you to effectively generate brand new voices from pre-existing ones! Unfortunately, there is no way of then storing these new voices, or even seeing which algorithm(s) you ended up in, since nothing registers in the display when using continuous sliders.

7. Feedback level - like the algorithm function, this allows you to change the feedback loop value (Fbl) for either voice A, voice B, or both, in real time. This could be useful for introducing distortions or white noise into a sound.

8. LFO Wave - allows you to change the LFO waveshape for either voice A, voice B, or both.

9. LFO Speed - allows you to change the LFO speed for either voice A, voice B, or both.

10. LFO Delay time - allows you to change the LFO delay time for either voice A, voice B, or both.

11. LFO P. MOD SENS. - allows you to alter the pitch modulation sensitivity of all the operators of voice A, voice B, or both.

12. LFO P. MOD depth - allows you to change the direct pitch modulation depth for voice A, voice B, or both.

13. LFO A. MOD depth - allows you to alter the direct amplitude modulation depth for voice A, voice B, or both.

14. Pitch EG Rate 1, 2, 3, or 4 - allows you to alter any of these values for voice A, voice B, or both.

15. Pitch EG Level 1, 2, 3, or 4 - allows you to alter any of these values for voice A, voice B, or both.

16. Portamento time - allows you to change the portamento Time value for voice A, voice B, or both. Bear in mind that when you first enter performance edit mode, the Foot switch (FS2) is ON for portamento for both voices, so you won't be able to set up automatic portamento effects in any event unless you change this to OFF (see Chapter Thirteen for more on this).

17. Frequency Coarse (for operator 1, 2, 3, 4, 5, or 6) - for voice A, voice B, or both. This option, along with the next one (Frequency Fine) is perhaps the most powerful of all the continuous slider functions, since

it allows you to make *qualitative* timbral changes in real time - something you cannot do with any other DX7II programming tools. Specifically, altering the Frequency Coarse value in real time for any operator will usually induce *harmonic* timbral changes, since whole-number frequency ratios will remain whole-number ratios.

18. Frequency Fine (for operators 1, 2, 3, 4, 5, or 6) - again, for voice A, voice B, or both. The kind of qualitative timbral changes you can initiate with this option will largely be *inharmonic* ones.

19. OSC. detune (for operators 1, 2, 3, 4, 5, or 6) - this allows you to initiate relatively subtle beating effects by changing the detuning value for any operator in voice A, voice B, or both. Here, changing the position of the CS will have the effect of altering the speed (and not the depth) of the beating, since it is simply shifting the frequency of a particular operator by a certain small amount.

20. EG Rate 1 (R1) (for operators 1, 2, 3, 4, 5, or 6) - allows you to alter this EG value for voice A, voice B, or both.

21. EG Rate 2 (R2) (for operators 1, 2, 3, 4, 5, or 6) - allows you to alter this EG value for voice A, voice B, or both.

22. EG Rate 3 (R3) (for operators 1, 2, 3, 4, 5, or 6) - allows you to alter this EG value for voice A, voice B, or both.

23. EG Rate 4 (R4) (for operators 1, 2, 3, 4, 5, or 6) - allows you to alter this EG value for voice A, voice B, or both.

24. EG Level 1 (L1) (for operators 1, 2, 3, 4, 5, or 6) - allows you to alter this EG value for voice A, voice B, or both.

25. EG Level 2 (L2) (for operators 1, 2, 3, 4, 5, or 6) - allows you to alter this EG value for voice A, voice B, or both.

26. EG Level 3 (L3) (for operators 1, 2, 3, 4, 5, or 6) - allows you to alter this EG value for voice A, voice B, or both.

27. EG Level 4 (L4) (for operators 1, 2, 3, 4, 5, or 6) - allows you to alter this EG value for voice A, voice B, or both.

28. Key Velocity (for operators 1, 2, 3, 4, 5, or 6) - allows you to change the Velocity sensitivity for any operator in voice A, voice B, or both. This will have the effect of not only making that operator more sensitive to keyboard dynamics, but will also have the side effect of reducing its effective nominal output level, since Velocity sensitivity is a negative control (see Chapter Eleven for more on this).

29. AMP. MOD. SENS. (for operators 1, 2, 3, 4, 5, or 6) - this will allow you to change the amplitude modulation sensitivity for any operator in voice A, voice B, or both. If you are using a controller for EG bias, this will have the added effect of making that controller more active. Using a CS for this function would allow you to set up EG bias modulations in voices but give you the option of using them only in specific situations. For example, you may want to use the breath controller for timbral change for a particular voice, but you only want to use this effect in one particular song. By assigning a CS to the AMP. MOD. SENS. function for a particular modulator, what you are effectively doing is "sensitizing" that modulator for EG bias modulation only when the slider is moved - thus making the breath controller active only at those times. This is a powerful example of how you can actually use one real-time controller (in this case, a CS) to activate another one!

30. Total level (for operators 1, 2, 3, 4, 5, or 6) - this allows you to adjust the overall output level for any operator in voice A, voice B, or both (regardless of whether that operator is in normal or fractional scaling mode). Like the output level and offset parameters, however, this is not a true real-time control since new notes will have to be

played (that is, keys will have to be re-triggered), in order for the new value induced by the movement of the slider to be "entered". It is because of this restriction that this use of the CS is less useful than using another controller for EG bias modulation, which does allow for complete real-time control over output level or offset.

Finally, it's worth pointing out - as briefly discussed in Chapter Ten - that the whatever function you assign to CS1 may also be assigned to FC1 (foot controller 1). This is accomplished from edit switch 26 (by turning the "CS1" parameter ON). Thus, you can use a foot controller as yet another source for continuous change (although it will of necessity be making the same change that CS1 will).

Let's move on now to a discussion of the performance edit parameters offered by edit switch 28, confusingly labeled "VOICE MODE" parameters.

Voice Mode parameters (edit switch 28)

The purpose of this switch is simply to show you the voice mode that was selected when the performance memory was created, and to allow you to make just a few small overall adjustments to the that performance memory. Of course, we really don't need this switch to figure out which voice mode was in use when the performance memory was first created, since the LED status lights above the three voice mode switches will show us that. Moreover, the LCD repeats this information, as well as telling us the number(s) of the voice(s) in use. However, edit switch 28, in addition to just showing you which voice mode you are in, also allows you to change the voice mode from *within* performance edit mode. This edit switch actually calls up several different LCD displays, depending upon which voice mode you selected when creating the performance memory. As usual, you cycle through these different displays by pressing the switch repeatedly. One display that appears, however, no matter which voice mode you chose is the one in **figure 14-6.**

Figure 14-6

The only active parameter here is called "Voice mode". This allows you to change the voice mode to any of the three available options: single, dual, or split. *Note, however, that you cannot change the selected voice or voices themselves - only the voice mode.* This is, in my opinion, one of the glaring weaknesses of the memory organization of this instrument. It means that if you create a performance memory with two voices and a set of performance edit parameters, you cannot keep those parameters intact and simply call up a different combination of voices. Let's suppose you created a performance memory from dual mode, and you've set up FS2 for the "Soft" function, and CS1 for a different function. Now you decide that you'd like to see how the Soft pedal and the CS affect two different voices. The DX7II offers you no way to accomplish this. Instead, you'd have to go right back to square one - returning to voice play mode, picking the two new voices, then returning to performance edit mode and entering in the Foot switch and CS values all over again. If there were some way of actually picking

different voices from within this switch instead of simply picking different voice modes, things would be a lot easier - but you can't, and they ain't.

If you've created a performance memory from single voice mode, there will be only a "-----" under voice B, indicating that there is no voice B. Pressing edit switch 28 a second time will then yield the display in **figure 14-7.**

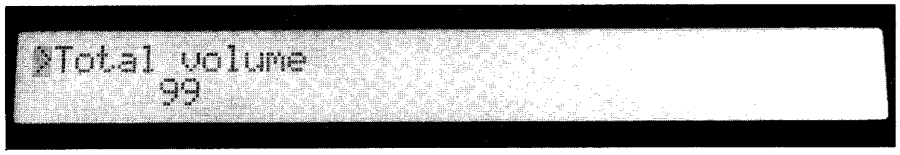


Figure 14-7

The only available parameter here, as you can see, is "Total volume", with a range of 0 - 99. This allows you to adjust the volume of the performance memory. Why would you need to ever do this? Well, you may have an internal memory (or RAM cartridge, or disk) filled with thirty-two performance memories, some of which may be significantly louder or softer than others. By attenuating the total volume of the loudest ones, you won't have to constantly deal with grabbing for the DX7II volume slider or mixing desk fader whenever you call up the louder ones; this control allows you to balance the relative volumes from performance memory to performance memory. The performance initialization default for this parameter is 99 (maximum volume); thus, it is normally used strictly as an attenuating control.

If you created your performance memory from dual mode, the second LCD display will look like **figure 14-8.**

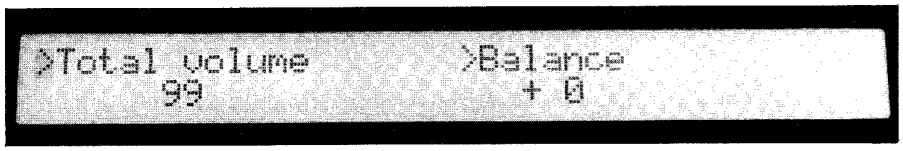


Figure 14-8

Here, in addition to being able to adjust the total volume, you can also adjust the relative Balance between the two voices. The range of this control is -50 to +50, with positive values giving you more of voice A and negative values giving you more of voice B. The maximum value of +50, then, would give you voice A only, while the minimum value of -50 will give you voice B only. The performance initialization default is a value of +0, giving you equal amounts of both voices. This control will be useful if one voice is significantly louder than the other, again allowing you to create a balance, but this time *within* the performance memory.

If you press edit switch 28 one more time, you'll see the last dual Voice mode LCD display. **(see figure 14-9)**

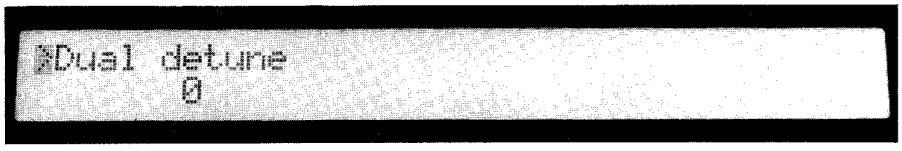


Figure 14-9

This last parameter, called "Dual detune", allows you to slightly shift (within a quarter tone) the pitch of the two voices relative to one another. The range of this control is 0 - 7, with larger values causing more of this pitch shift. It's important to realize that this parameter changes the pitch of *both* voices, shifting voice A a bit sharp, and making voice B a bit flat. Thus, *neither* of the two voices will be in tune. The purpose of this control, then, is simply to allow for beating and chorusing effects to be generated. In this way, it is similar to the "unison" key modes we examined in the last chapter - but with two important differences. First of all, the total polyphony of a performance memory created from dual voice mode is eight - not the four you get from working in "unison poly" key mode. Secondly, this is a performance edit parameter and not a voice edit parameter, so you'll have to go to performance mode in order to use this control. However, this makes a good argument for simply creating performance memories from dual voice mode with the same patch chosen for both voice A and B, and then using this Dual detune parameter instead of the "unison poly" key mode parameter - you'll end up with precisely the same effect, but with eight-note polyphony instead of four.

If you've created a performance memory from split voice mode, then pressing edit switch 28 a second time will yield the same LCD display as shown in figure 14-8 above. The Total volume and Balance parameters here work in precisely the same way. Pressing edit switch 28 once again will give you the final Split voice mode display. **(see figure 14-10)**

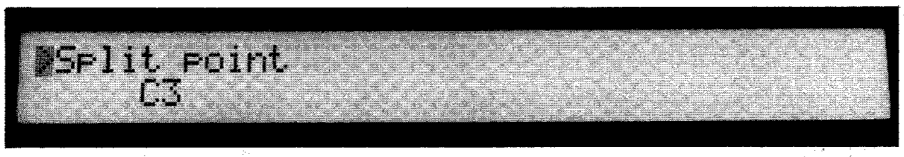


Figure 14-10

This allows you to change the Split point between the two voices from the default of C3. The range of this control is C-2 to G8, and, like the Key Transpose function (edit switch 7), data can be entered here either with the data entry section or simply by playing a note on the keyboard. The first note you play upon calling up this parameter will simply be entered in as the split point, and will not sound. Of course, you can only enter in values in the range C1 to C6 by using this latter technique. To enter in split points off the physical keyboard (which you might want to do if, for example, you have a sequencer hooked in and you want it to play one voice while you play live over it with the whole physical DX7II keyboard at your disposal), you need to use the data entry section. In summary, then, edit switch 28 allows you to view and/or change the voice mode that was used to create your performance memory, and to adjust the overall volume, balance, relative tuning, and split points between these voices. Let's move on now with a discussion of the performance edit parameters accessed from edit switch 29, the "MICRO TUNE" switch. Not all of its parameters are related to the much-heralded micro tuning functions of the DX7II, however. In fact, this switch offers four LCD displays, accessed as usual by pressing the switch repeatedly. **(see figures 14-11(a) through (d))**

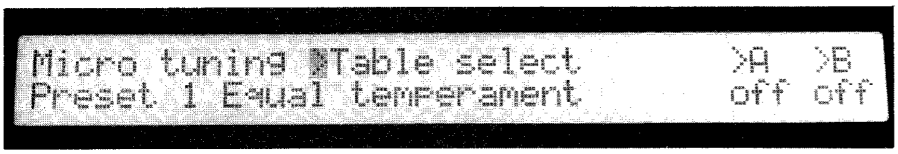


Figure 14-11(a)

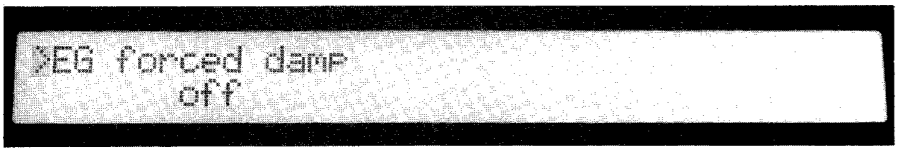


Figure 14-11(b)

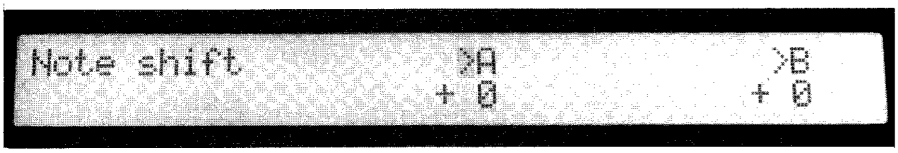


Figure 14-11(c)

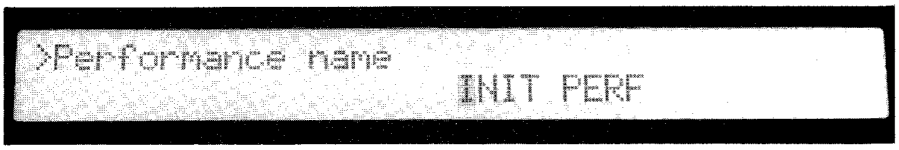


Figure 14-11(d)

Let's work our way backwards through these - in order of increasing complexity. The last of these, the Performance name display (**figure 11(d)**), is where you actually enter in a name for your performance memory. The procedure here is identical to that of naming a voice (see Chapter Eight) - with both upper and lower-case characters accepted. As usual, you must keep a finger holding down the character (edit) switch through this whole process. In fact, the only difference here is that you are allowed a maximum of 20 characters, as opposed to the 10-character limit for voice names.

The Note Shift parameter (accessed from the display shown in **figure 14-11(c)**) allows you to transpose voice A, voice B, or both, over a range of two octaves (that is, +24 semitones to -24 semitones) in each direction. As pointed out in the DX7II owners manual, the original voice transposition (accomplished with edit switch 7) is retained as part of the voice memory, and the Note Shift value is simply added to or subtracted from that voice setting when you use that voice in performance mode. One of the powerful applications for this control is that it enables you to create a performance memory with the same voice in dual mode, and to then play that voice in block chords: octaves, fifths, or whatever musical interval is useful to you. If you create a performance memory from single voice mode, then, as usual, changing the Note Shift value for voice B will have no effect on anything.

EG Forced Damping

The EG forced damp display (as shown in **figure 14-11(b)**) has only one parameter, called (surprise, surprise) EG forced damp. This is a simple ON-OFF control. What does EG forced damping do? Well, this control deals with another facet of what occurs in the DX7II when note robbing occurs. As we have learned, such note robbing occurs when we

exceed the instrument's total polyphony - a polyphony which will vary according to which voice mode and key mode we are in. For example, a single voice in polyphonic key mode will have a total polyphony of sixteen notes, while a unison poly voice will only have four. If we use two unison poly voices in dual mode, then the total polyphony of our instrument shrinks down to two voices! The table that follows will help you to see how this works:

Mode	Total polyphony
Single voice, polyphonic	16 notes
Single voice, unison poly	4 notes
Single voice, monophonic	1 note
Single voice, unison mono	1 note
Dual voice, both polyphonic	8 notes
Dual voice, both unison poly	2 notes
Dual voice, both monophonic	1 note
Dual voice, both unison mono	1 note
Split voice, both polyphonic	16 notes (eight per voice)
Split voice, both unison poly	4 notes (two per voice)
Split voice, both monophonic or both unison mono	2 notes (one per voice)
Split voice, one polyphonic and one unison poly	10 notes (eight plus two)
Split voice, one polyphonic and one monophonic or unison mono	9 notes (eight plus one)
Split voice, one unison poly and one monophonic or unison mono	5 notes (four plus one)

As you can see, a performance memory can have a total polyphony that ranges from one to sixteen voices, depending upon how it is set up. This means that note robbing may occur as early as the second note played, or as late as the seventeenth note played. What note robbing really means is that the DX7II microprocessor stops a tone generator in the middle of what it's doing and forces it to begin generating a different note. The EG forced damp parameter tells the tone generator whether it should continue its current EG movements when this interruption occurs, or whether it should actually start a new envelope. If it is OFF, the tone generator will continue with the same movements - in other words, it will start the new note within the old envelope. If it is ON, the tone generator will begin a whole new envelope with each new note played. In general, having EG forced damping OFF (which is the initialization default) will make for smoother transitions between notes when the total polyphony is exceeded and note robbing begins. There will be times, however, when it will be better for it to be ON - when working with sounds with long attack (R1) times, for example, or voices with sustain level (L3) values that are 0 or significantly low. In the first instance, new notes (after the polyphony is exceeded) won't have the long attack times unless EG damp is ON. In the second instance, new notes (after the polyphony is exceeded) may not sound at all if the old notes had a chance to drop down to a 0 or low sustain level - again, unless EG damp is ON.

Let's run an Exercise now to try out the effects of the EG Forced Damp control:

Exercise 83
EG Forced Damping

1) Put your DX7II into single voice play mode and INITIALIZE it. Leave it in algorithm #1, TURN OFF operators 3 through 6 ("110000") and, using the system of operators 1 and 2, GENERATE a square wave.

2) Make the following changes to the initialization defaults for the EG values for both operators 1 and 2: R1 = 35, and R4 = 31. Leave all the other EG values at their default settings. This will give us a sound with a moderately slow attack and a moderately long release time.

3) Press edit switch 23 and change the Key mode from its current default value of "polyphonic" to "unison poly". Change the Unison detune value to 5. Play a note and listen (Audio Cue 83A). We have created a very pleasant, chorused square wave sound.

4) Name and store this sound somewhere either in internal memory or RAM cartridge memory. Call the sound up (from single voice play mode) and press the performance mode select switch in order to put your DX7II into performance play mode. The name of the performance memory should be "INIT PERF", and it should contain your new voice in single voice play mode. Press the edit switch in order to enter performance edit mode.

5) The chart above shows that we currently have a total polyphony of four notes (since we are using a single unison poly voice). Press edit switch 29 repeatedly if necessary in order to call up the EG Forced damp display. Note that it is currently at its default of OFF. Play an arpeggiated series of five notes, taking care to hold each of the notes down (in other words, play legato) and listen (Audio Cue 83B). Note that the fifth note causes note robbing to occur and that the long attack time that we programmed in in step 2 does not occur. This is because EG forced damping is OFF, meaning that the fifth note simply started from within the existing envelope of the first note (which was simply holding at its L3 sustain level of 99).

6) Press the "yes" button in order to turn the EG forced damping ON. Play the same legato five notes and listen (Audio Cue 83C). Note that the fifth note (and any subsequently added notes) now exhibits the same slow attack time as the first four.

7) Experiment further with other voices and note how changing the EG forced damping control affects the overall sound when the maximum polyphony is exceeded.

Whenever you are in any of the voice play modes, remember that all of the performance edit parameters are reset to initialization defaults - meaning that EG forced damping will always be OFF. The only way to turn it ON is to take your voice or voices and organize them into a performance memory. It is, I agree, a rather tedious way of doing things - but it's the only way that this can be accomplished.

Let's move on now to a discussion of one of the most unique capabilities of this instrument: *micro tuning*.

Micro Tuning

Western music is based upon what is known as a *twelve-tone* system. As we learned in Chapter One, whenever you double the fundamental frequency of a sound, you are raising the pitch by an

octave; conversely, whenever you halve a sound's fundamental frequency, you are dropping the pitch by an octave. The twelve-tone system divides the octave interval into twelve increments, known as *semitones* or *half-steps*. This is the origin of the common musical scale, consisting of the notes A, A-sharp (or B-flat), B, C, C-sharp (or D-flat), D, D-sharp (or E-flat), E, F, F-sharp (or G-flat), G, and G-sharp (or A-flat).

The real question is: if we are going to use twelve increments per octave, where precisely do we place them within the octave span? An obvious answer might be that we place them at equal 1/12 of an octave points, so that the spacing between each semitone is the same. This is a valid approach, and in fact is the system used by virtually every synthesizer ever built (and by many acoustic instruments as well), called *equal temperament.*

Though equal temperament provides a simple solution to this question, it also presents some new problems of its own. First of all, there is nothing that says that we absolutely must have twelve increments per octave, or even that we have to base scales on the octave interval - there are in fact many non-Western systems of music that either use different numbers of increments per octave, or don't even base their music on the octave interval at all. Secondly, if we *do* decide to use a twelve-tone system based on the octave, we must realize that the absolute difference in frequency between octaves increases as we play higher notes: the difference between an A440 and an A880 is 440 cycles, while the difference between an A880 and an A1760 (the next octave up) is 880 cycles. While placing semitones at equally spaced intervals may provide a comfortable mathematical solution, these intervals may not provide the most musically pleasing sounds. For these reasons, many designers of acoustic instruments and acousticians have experimented with alternative tunings. The DX7II is one of only a small handful of commercially available synthesizers* that allow the synthesist to explore these options as well.

Some acoustic instruments sound much better when tuned in a way that is not equally tempered. The acoustic piano, for example, exhibits a timbral property called *inharmonicity*. Because of a number of physical factors (including the enormous tension of the strings and the stiffness of the steel used in the piano string), the overtones produced by a piano sound will tend to be slightly sharp of their expected harmonic values. This effect becomes more and more prominent with higher notes. For this reason, pianos are normally tuned with what are called *stretch tunings*, where the overtones and not the fundamental frequencies, are used as a basis for tuning adjoining strings. Because the inharmonicity causes the overtones to be slightly sharp, the end result is that higher and higher fundamentals tend to be sharper and sharper relative to where they might be if the piano were tuned in equal temperament. This yields a much more pleasing tonal quality to the piano sound. Because stretch tunings are typically used in pianos, most synthesizers will rarely if ever be in tune with an acoustic piano throughout the entire 88-note range of the piano. If you've ever tried playing a synth with an acoustic pianist, you've probably encountered this frustrating phenomenon.

*Others include the Yamaha FB-01 and TX81Z tone modules, late revisions of the Sequential Prophet-5; the Sequential Prophet T8; the Fairlight CMI; the Synclavier; and most modular analog synthesizers.

Well, there's no need to put up with these kinds of problems ever again if a DX7II is the synth you're using, because the Micro Tuning capabilities accessed from edit switches 29 and 14 will allow you to customize your own tunings - tunings which can actually be stored in memory and linked to particular performance memories!

We'll talk about the way that you can create your own tunings shortly. The process, however, can be tedious, so in an effort to make life easier for the DX7II user, eleven preset tuning schemes are provided in the instrument's permanent memory. They are as follows:

1. Equal temperament
2. Pure (Major)
3. Pure (Minor)
4. Mean tone
5. Pythagorean
6. Werckmeister
7. Kirnberger
8. Vallotti and Young
9. 1/4 tone shifted equal
10. 1/4 tone
11. 1/8 tone

These are accessed from the Table select parameter in the Micro tuning display of edit switch 29, as shown in **figure 14-11(a)** above. This display allows you to select any one of these presets (or your own user-defined microtuning table) and to use voice A, voice B, or both voices, with the selected tuning. You cannot select a different preset for each voice, though you can have one voice at the default equal temperament, and another using a different preset. Let's talk a little bit about each of the eleven presets:

1. Equal temperament - as mentioned above, this is the default preset for all DX7II voices and for any newly created performance memories. Very simply, each semitone is placed at an equal interval within the octave. The octave is normally divided into 1200 equal increments (called *cents*), and each note falls at 100-cent intervals.* This is the system of tuning you've heard for years in virtually every synthesizer and in most acoustic instruments.

2. Pure (Major) - this is called an *open tuning*, since it must always be based on a specific musical key. When you select this table, the LCD will also ask you to specify a key with the Key parameter. This tuning is derived directly from the harmonic overtones, and is based on the major triad, so that the intervals of a major third and the fifth are adjusted to be as close to beatless and as musically pleasing as possible. All the other note values are shifted accordingly, meaning that this tuning will sound best in the particular key you specify, and will sound increasingly out-of-tune as you play musical scales and chords from keys that are further away. Pure tunings such as this one allow you to hear what the true harmonic overtones of a note actually sound like!

3. Pure (Minor) - this is another open tuning, very similar to the Pure (Major), but based on the minor scale triad instead. Therefore, it is the minor third and the fifth intervals that are adjusted to be beatless, and all the other intervals shifted accordingly. Again, this scaling is

* In the DX7II, the octave is actually divided up into 1024 increments, no doubt because this is one of the magic "computer" numbers. We'll see how this works when we discuss how to create new micro tunings from scratch.

derived directly from the harmonic series of overtones - allowing you to hear what the harmonic overtones of a note actually sound like. Being an open tuning, it will be optimum only when you are playing scales and chords from the key you specify. It is also worth noting that complementary major/minor scales (like for example, C major and A minor) will sound equally good when played with these pure tunings.

4. Mean tone - In point of fact, there are many thousands of different kinds of *mean tone* tunings, and the table offered by this preset is just one of them. Mean tone tunings, developed in the Renaissance era, are open tunings (meaning that you need to specify a key) which are based more on the pure third interval than on the fifth. This will naturally lead to more errors (meaning less pleasing sounds!) as you play in keys further and further removed from the starting key - though most of the borrowed chords from closely related keys will sound just fine.

5. Pythagorean - this is, as you might expect, one of the oldest tuning systems, used extensively in Medieval music. It is an open tuning based on the interval of the fifth. The idea is to select a key and then use the "circle of fifths" to tune every interval of a fifth as perfectly (meaning as beatlessly) as possible. If, for example, you select the key of C, then the G (a musical fifth higher) is tuned to be as musically pure and as beatless as possible. From there, the next D up (which is a musical fifth higher than the G) is tuned the same way - then the next A, the next E, the next B, etc. This sounds good in theory, but if you follow this routine all the way through to the twelfth step (eighth octave), you will be tuning that last high C some 24 cents sharp of what it would be had you used equal temperament. This differential of 24 cents is called the *comma of Pythagoras*, and musicologists have struggled for centuries to deal with this inaccuracy by adjusting the spacing of various notes within the eight-octave range. In any event, the major triad will generally sound more pleasing in this tuning than in a mean tuning.

6. Werckmeister - this preset (and all the others that follow) is a *closed* tuning, meaning that it is a compromise effort designed to sound reasonably good in all musical keys. For this reason, the LCD does not ask you to specify a key when you choose this table. The Werckmeister tuning is a *well-tempered* tuning, developed in the 15th century. These kinds of tunings (used extensively in Baroque music) "split the difference" between the intervals of the third and the fifth so that they sound acceptable in all keys (but perhaps not as good as an open tuning in that key alone). The commonly accepted notion that modern musicians have that different musical keys have different qualities stems from these kinds of closed tunings, since different keys actually *do* have different tonal qualities when played within closed tunings. It is also worth pointing out that some well-tempered tunings are closer to equal temperament than are others. The Werckmeister, however, is not particularly close to equal temperament - all the semitone increments have been adjusted to some degree so that the gaps between them are all different.

7. Kirnberger - another closed tuning that represents a compromise between the intervals of a third and a fifth. The fifths are spread a little further apart here than they are in the Werckmeister.

8. Vallotti and Young - very similar to the Kirnberger (another closed tuning). The fifths are spread even more in this tuning, though the difference between this table and the two preceding ones is very slight indeed.

9. 1/4 shifted equal - this is simply a plain and ordinary equal temperament tuning, but with all notes shifted up by a quarter tone. Why? Just to show you that it can be done.

10. 1/4 tone - this tuning, and the one that follows it, is a *microtonal* closed tuning in that it includes pitches that are not part of standard diatonic intonations - in other words, it contains notes that are non-standard. This particular preset divides the octave up into twenty-four, instead of twelve, equal increments, so that the span from C1 to C3 becomes an octave - therefore changing the range of the five-octave DX7II keyboard into a mere two-and-a-half octaves! Many non-Western musical systems use quarter-tone scalings, and many experimental musicians work with these kinds of microtonal scalings. They may not be everybody's cup of tea (if you're steeped in a long tradition of Western music, intervals of less than a semitone often sound sour), but they are valid tunings nonetheless, and Yamaha is to be applauded for opening the world of microtonalities up to the "average" musician.

11. 1/8 tone - same as the 1/4 tone tuning, except that here the octave is divided into 48 equal increments, making the full pitch range of the DX7II keyboard only an octave-and-a-quarter.

It is worth noting that all of the preset tunings are based on middle A (A3) being equivalent to 440 Hz. In other words, A3 is the standard reference from which all other intervals are adjusted. If you are creating a performance memory with a voice which has been transposed (with the Key Transpose parameter) or shifted (with the Note Shift parameter), then obviously A3 will not be 440 Hz. In that case, whatever new note becomes 440 Hz acts as the point of reference. Interestingly enough, altering the Master Tuning (via edit switch 14) does not affect this reference point in any way.

Let's run an Exercise now in order to try out the eleven DX7II micro tuning presets:

Exercise 84
Using the preset micro tunings

1) INITIALIZE your DX7II from single voice play mode, leave it in algorithm #1, TURN OFF operators 3 through 6 ("110000"), and use the system of operators 1 and 2 in order to GENERATE a square wave. Name and store this voice somewhere in your internal or RAM4 cartridge memory.

2) Press the "Dual" mode select switch in order to enter dual voice play mode, and select the voice you created in step 1 for both voice A and voice B.

3) Press the "Performance" mode select switch in order to enter performance play mode. Note that dual mode, containing the voice you just created as both voice A and B, is still retained, and that the name of this performance memory is currently "INIT PERF". This tells you that all of the performance edit parameters have been initialized.

4) Without pressing any of the thirty-two main switches, press the "edit" mode select switch, putting you into initialized performance edit mode. Press edit switch 29 repeatedly if necessary in order to call up the Micro tuning display.

5) Position the cursor over the "voice A" parameter and turn it ON. Position the cursor over the "voice B" parameter and turn it ON as well. For the moment, we'll change the tuning of both voice A and voice B (both of which contain the square wave sound you made back in step 1).

6) Note that the micro tuning Table select parameter is currently at its default Equal temperament setting. It's easiest to hear the effect of micro tuning if you listen to a consistent interval, so play C3, hold it down, and add to it G3. Listen (Audio Cue 84A). Note that this is the same pleasing interval of a fifth that we have heard many, many times before from electronic and acoustic instruments alike.

7) Position the cursor over the Table select parameter and press the "yes" button in order to call up Table #2: the Pure (Major) tuning preset. Note that the LCD now adds a "Key" parameter. **(see figure 14-12)** This is because the Pure (Major) preset is an open tuning. Note that the default key is C (actually C major). Leave it at this value and play the same C3 - G3 interval as in the last step. Listen (Audio Cue 84B). Note that both the C and the G seem to be somewhat sharper than in the last step, and also that the interval between them appears to be changed somewhat. This is because they have both been *rescaled* by the DX7II microprocessor. One of the characteristics of an open tuning like this one is that other intervals, or the same interval starting at a different note, can often sound quite different. Therefore, play C# and G#, D and A, D# and A#, etc., up through a full octave and listen (Audio Cue 84C). Note that many of these (particularly the D-A and B-F# intervals) sound particularly sour.

Figure 14-12

8) Position the cursor over the Key parameter and change this to a new value of D. Repeat the same root - fifth experiment as in the last step (from C3 up to C4) and listen (Audio Cue 84D). Note that this time, C#-G# and E-B are particularly sour-sounding, while the D-A interval sounds fine. This is because we have altered the starting key for this open tuning. Experiment by repeating this root - fifth arpeggiation with the Pure (Major) micro tuning preset in different keys. Experiment further by playing different intervals, major and minor triads, and both major and minor scales. When you are done experimenting, restore the Key back to the original default of C.

9) Position the cursor over the Table select parameter and press the "yes" button in order to change this to the Pure (Minor) preset. Note that the default key for this preset is A (minor). This is because A minor is a complementary key to C major (the initialization default for the Pure (Major) preset). Play the root - fifth arpeggiation from C3 up to C4 and listen (Audio Cue 84E). Note that the C-G interval sounds fine (since, as we just pointed out, C major is a complementary key to A minor), and that the E-B and A#-F intervals sound the most sour. Experiment by changing the Key and seeing how this affects these different intervals. Experiment further by playing different intervals as well as major and minor triads and scales. When you are done experimenting, restore the Key back to its default of A.

10) Position the cursor over the Table select parameter and press the "yes" button in order to change this to the Mean Tone preset. Note that the default key for this preset is C (major). Play the same root - fifth arpeggiation from C3 up to C4 and listen (Audio Cue 84F). Note that, for the most part, all of these intervals (with the severe exception

of G#-D#) sound quite pleasing. The real purpose of Mean Tone tunings, however, is to get consonant thirds. Therefore, try playing root - third intervals (i.e. C-E, C#-F, D-F#, etc.) from C3 to C4 and listen (Audio Cue 84G). Note that virtually all of these sound extremely pleasing to the ear. As before, experiment with different Keys and by playing different intervals as well as major and minor triads and scales. When you are done experimenting, restore the Key back to its default of C.

11) Position the cursor over the Table select parameter and press the "yes" button in order to change this to the Pythagorean preset. Note that, once again, the default Key for this open tuning is C. Play the root - fifth arpeggiation from C3 to C4 and listen (Audio Cue 84H). Note that, again with the exception of the G#-D# interval, all of these sound extremely good, just as they did with the Mean Tone preset. What, then, is the difference between the two? The answer is that the Pythagorean tuning is concerned only with getting the interval of the fifth as perfect as possible, while Mean Tone tunings deal more with the interval of the third. To prove this, play the same root - third intervals from C3 to C4 that you did in the last step, and listen (Audio Cue 84I). Note that many of these, particularly F-A and A#-D, sound really sour! You can even hold down a chord and press the "no" button in order to return momentarily to Mean Tone tuning. Flip back and forth - there's really quite a difference! experiment further with changing the Key and with playing different intervals and major and minor triads and scales as well. When you are done experimenting, restore the Key back to its default of C.

12) Position the cursor over the Table select parameter and press the "yes" button in order to change this to the Werckmeister preset. Note that there is no Key parameter, since this preset and all the others to follow are closed tunings and are therefore not key-specific. Play the usual root - fifth arpeggiation from C3 to C4 and listen (Audio Cue 84J). Note that all of them sound reasonably good. Play the root - third arpeggiation from C3 to C4 and listen (Audio Cue 84K). Note that they, too, sound good - but that the spacings between them seem to change somewhat from root to root. This is because the Werckmeister is a *well-tempered* tuning, representing a compromise between "perfect fifths" and "perfect thirds". Experiment by playing other intervals as well as major and minor triads and scales.

13) Press the "yes" button again in order to call up the Kirnberger preset. Play the root - fifth intervals from C3 to C4 and listen (Audio Cue 84L). This is very close to the Werckmeister tuning. You can flip back and forth between them and listen during every interval. Note that there is little difference between the two. Experiment by playing other intervals as well as major and minor triads and scales.

14) Press the "yes" button again in order to call up the Vallotti and Young preset. Play the usual root - fifth intervals from C3 to C4 and listen (Audio Cue 84M). This preset is also very close to both the Kirnberger and the Werckmeister presets. Experiment as usual with other intervals, triads, and scales.

15) Press the "no" button seven times in order to return to Equal Temperament. Play a C-major scale and listen (Audio Cue 84N). Now press the "yes" button eight times in order to call up the 1/4 Shifted equal preset, play the same C-major scale, and listen (Audio Cue 84O). As pointed out above, this preset is simply an equal temperament, with all notes shifted a quarter-tone higher. In other words, while the pitch

of all the notes on the keyboard has gone slightly sharper, the distance between any two adjacent semitones remains the same as it was in the Equal temperament preset.

16) Press the "yes" button in order to call up the 1/4 Tone preset. As mentioned above, this is an equal-tempered tuning which yields *microtones*. Play a chromatic scale from C3 up to C5 and listen (Audio Cue 84P). Note that this two-octave range now only produces an octave worth of pitches, and that the smallest interval between adjacent keys is a quarter-tone, and not the diatonic semitone that we're used to hearing. All of the usual musical intervals that we're used to working with are now twice as far apart - if you play a C3, A3, and D4, for example, you'll hear a major triad!

17) Press the "yes" button again in order to call up the 1/8 Tone preset. This is an equal-tempered tuning, but here the smallest musical interval between adjacent notes is halved yet again to become an eighth-tone. Play a chromatic scale from C1 up to C4 and listen (Audio Cue 84Q). Note that this four-octave span now only produces an octave's worth of pitches! To get the same C-major triad we heard in the last step, you'll now have to play C2, E3, and E4. Try it!

18) Restore the Table select parameter back to Equal Temperament. Whenever a voice is OFF for micro tuning, it will automatically default to this system of tuning. Therefore, by applying micro tuning to only voice A and not voice B (or vice versa), we can have one voice in the standard equal temperament and another using a different preset. This will be most effective when using the same voice for voice A and voice B in dual mode, which is why we cleverly set things up that way back in step 1. Therefore, position the cursor over the "voice A" parameter and turn it OFF. Voice A can now only be equally tempered, while we can assign to voice B any micro tuning preset (or, as we'll see in the next exercise, any user-created micro tuning values) we like.

19) Call up the Pure (Major) preset, play a chromatic scale from C3 to C4, and listen (Audio Cue 84R). Note that *only* A3 is in tune, and that beating occurs (at different rates of speed) for all other notes. This is because, as noted above, all of the micro tuning presets are based upon A3 being equal to 440, so that A3 is the only note where the equally tempered voice A and the Pure (Major) voice B agree. Try calling up different presets for voice B only, play the same chromatic scale, and note that the beating effects change speed from note to note with all the different presets. The 1/4 Shifted Equal preset will, of course, yield beating for all notes (since every one of them, including A3, is sharpened by a quarter tone), while the microtonal presets (1/4 Tone and 1/8 Tone) will sound completely sour against the diatonic voice A.

Creating a performance memory with the same voice in dual mode, and then turning only one of them ON for an alternative tuning scheme is yet another way of generating interesting beating effects. This is particularly useful since the speed of the beating will change from note to note - and in a way that is largely unpredictable. This is another "trick of the trade" that can help to make the DX7II sound more "natural".

Beyond these eleven permanently programmed tables (that you can't possibly erase), the DX7II also allows you to modify these tuning presets, or even to create your *own* microtuning schemes! This is

accomplished with edit switch 14, used in conjunction with edit switch 29. Edit switch 14, as we have seen before, offers four separate LCD displays, accessed as usual by pressing the switch repeatedly. **(see figures 14-13(a) through (d))**

Figure 14-13(a)

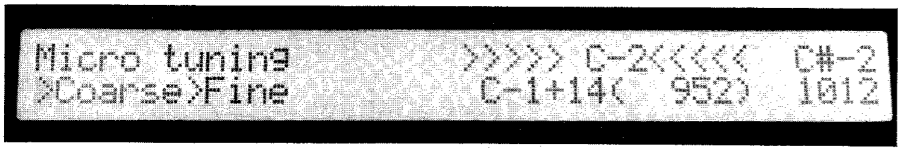


Figure 14-13(b)

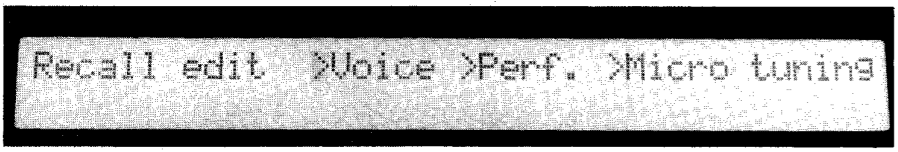


Figure 14-13(c)

Figure 14-13(d)

The display that we are interested in here is the Micro tuning display, shown in **figure 14-13(b)**. This display is similar in many ways to the Fractional scaling display we worked with in Chapter Twelve, except that single notes (as opposed to Note Groups) are shown. However, like the Fractional display, a series of three notes are displayed, though you have the ability only to alter the center one (said to be in the *active window*). Also like the fractional display, we step through these notes by using the "internal" and "cartridge" switches as quasi-cursor switches (active for their "FRACTIONAL/MICRO TUNE SET" function, as labeled in green above these switches). We can also bring a particular note into the active window by pressing and holding down the key we want and then pressing either the "internal" or the "cartridge" switch.

The "Coarse" and "Fine" parameters allow you to individually adjust the tuning of *each note* that the DX7II can access. That means that we are not limited to the C1 - C6 range of the physical keyboard, but that we can actually change the tuning of any or all notes through the full MIDI note range of C-2 to G8! The Coarse parameter will simply increase or decrease the note value by a full semitone - the smallest "coarse" increment we're used to hearing in Western music. The Fine parameter, on the other hand, allows you to increment the tuning of a particular note by units which are *not* cents, but are actually equivalent to approximately 1.17 cents. This is because, as pointed out earlier, the DX7II software divides the octave up into 1024 increments

instead of the more usual 1200 cents. By placing the cursor over the Fine parameter, you can therefore use the data entry section to alter the tuning of a particular note by these 1.17 cent increments.

Let's examine the Micro tuning display in more detail. Initialize your DX7II from single voice play mode, and press edit switch 14 repeatedly in order to call up this display. You will see **figure 14-14.**

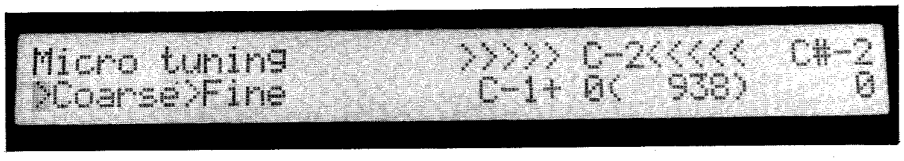


Figure 14-14

The note that will be in the active window (shown by the arrow pointers in the center) will initially be C-2, the lowest possible note the DX7II will recognize and well off the scale of the physical keyboard. Let's take a look at middle C instead. Play middle C on the keyboard and, while keeping the note held down, use your other hand to press either the "internal" or the "cartridge" switch. The display should now look like **figure 14-15.**

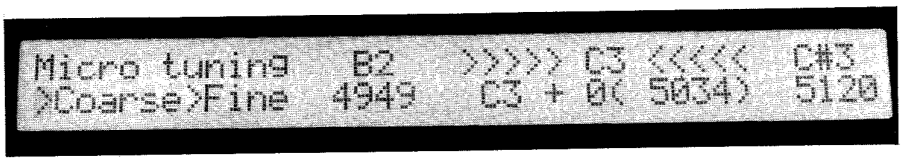


Figure 14-15

Because we are in a voice play mode (as opposed to performance play mode), all of the performance edit functions - including micro tuning - are currently at their default values. If you press edit switch 29 and look at the Micro tuning display, you will see that the initialization default sets both voice A and voice B OFF for micro tuning; in other words, they are both at equal temperament. Press edit switch 14 again to return to the Micro tuning edit display, and note that "C3" appears on both the top and the bottom line of the display. The top "C3" refers to the actual key on the keyboard and will not be changed by the data entry slider. The "C3" below the line shows us which Coarse value C3 is currently equivalent to. In this case, since we've just initialized, C3 is equal to C3 - meaning that when we play C3 on the keyboard, C3 is the note we hear. The Coarse parameter will let us change things, however, so that C3 - or any other note, for that matter - becomes equivalent to any other note. We'll be running an exercise shortly to try this out.

Getting back to the display, take a look now at the "+0" next to "C3". This indicates that C3 has not been shifted at all relative to equal temperament. Press the "cartridge" switch in order to put the next note higher (C#3, in this case) into the active window, and note that it, too, has "+0" next to its name. You can continue pressing the "cartridge" switch in order to view higher and higher notes (up to G8), or you can press the "internal" switch to view lower and lower notes (down to C-2), but you'll find that they are all at +0. This is proof positive that all notes are currently equally tempered. This number will change when we use the Fine parameter, and will show us by how many increments up (+ increments) or down (- increments) the tuning of a particular note has been changed (relative to equal temperament).

The number in parentheses just to the right of +0 tells us the absolute tuning of this note. Specifically, it shows us how many Fine increments above 0 this note is, with 0 being the normal value for C-2, the lowest MIDI note. As we change the relative "+0" value with the Fine control, the number in parentheses will change as well. In this way, the display always shows us both the relative and the absolute changes that we make (or any that have been made) to any note. Remember that the Fine increments are not cents, but are in fact slightly larger (each one being approximately 1/85 of a semitone). Thus, there are 85 (or, occasionally, 86) Fine increments to one Coarse increment. When you increment the Fine value for a note above +42 (since the note itself starts, in equal temperament, at the center point of +0), the display will therefore go to the next note higher at -42 Fine increments. Similarly, when you decrement the Fine value for a note below -42, the display will go to the next note lower at +42. Think about this, and it will start to make sense.

On either side of the active window are the surrounding notes (there won't be any to the left when you first initialize, since there are no notes lower than C-2) for reference purposes. In this way, you can see just how much you've changed a note relative to its nearest neighbors (just as the fractional scaling display allows us to see how the output level of a note group has been changed relative to adjacent note groups).

Beyond using this display to create new micro tunings, we can also use it to view and/or modify micro tuning presets in much the same way that we were able to use the fractional scaling display to view and/or modify normal curves. Press edit switch 29 in order to call up the Micro tuning display, and turn voice A (our only voice, since we started out in single voice play mode) ON. Now position the cursor over the Table select parameter and press the "yes" button in order to call up the Pure (Major) table. Press edit switch 14 again. The LCD should now look like **figure 14-16.**

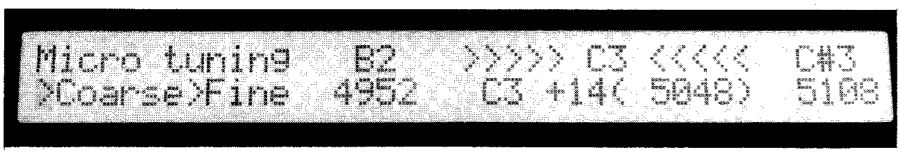


Figure 14-16

In this preset, C3 has been shifted up by 14 increments ("+14") to an absolute value of 5048 - as opposed to the figure of 5034 we saw in figure 14-15 above. Press the "cartridge" switch repeatedly and you'll see that C#3 has been shifted down by 12 increments (to 5108), that D3 has been shifted up 17 increments (to 5222), and that in fact virtually all notes - except A3, which is the point of reference - have been changed somewhat from the +0 values they exhibited when in equal temperament. If you now return to edit switch 29 and call up a different preset, then edit switch 14 will show you different shifts per note. This remarkable display literally lets you see what your ears tell you you are hearing. For example, if the interval of a fifth sound a little flat with a particular preset, the display will undoubtedly show a negative pitch shift for that note.

Furthermore, you can position the cursor over the Coarse or Fine parameter and *change* any of these values as well. Bring C3 back into the window by playing it on the keyboard, holding it down, and then pressing either the "internal" or "cartridge" switch. Press edit switch

29 and restore the Table select parameter to the Equal Temperament preset. Press edit switch 14 to return to the Micro Tuning Edit display. Position the cursor over the Coarse parameter (if it isn't already there) and press the "yes" button in order to make C3 now equivalent to C#3. Play middle C on the keyboard and note that it sounds exactly the same as C#3! Press edit switch 29 again and press the "yes" button in order to call up the Pure (Major) tuning preset. Return to edit switch 14, and make C3 once again equivalent to C#3 with the use of the Coarse control. Play C3 on the keyboard again and note that it now actually sounds a little bit *sharper* than C#3! This is because "C3" (which we've just changed to C#3) has been shifted up 14 increments by the Pure (Major) tuning preset, while "C#3" (which is still equivalent to C#3) has been shifted by the Pure (Major) preset *down* 12 increments. If you think about this for a moment, it should make sense. All we are doing is simply changing Coarse note values - but the preset is still imposing its Fine pitch shifts, which remain active even though the note itself may be changed.

Spend some time calling up each of the eleven tuning presets and then examining their values in the Micro Tuning Edit display offered by edit switch 14. Experiment by changing various key values until you get comfortable with this exciting control. User-created tunings usually begin with one of these presets, so it makes sense to have a good handle on the actual values that each of these presets represents. For your reference, Appendix D presents the tables of values used by each of the eleven permanent presets.

The DX7II offers us two internal memory slots (in addition the the edit buffer, which is where we are right now) in which to store microtuning data. These are simply designated as "User 1" and "User 2" from edit switch 29, and they can be accessed by simply pressing the "yes" button after calling up the last permanent preset (1/8 tone). **(see figure 14-17)**

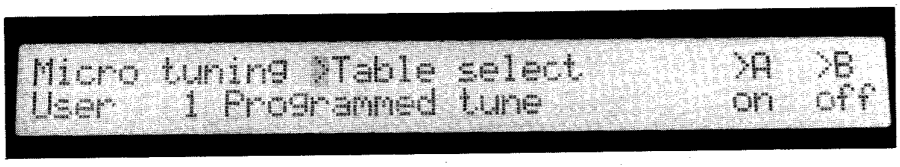


Figure 14-17

These two microtunings will be stored to disk as well (if you are working with the "FD" model) when voice and performance data are stored. Up to 63 additional micro tunings can also be stored onto a specially formatted RAM4 cartridge. These can be accessed if such a cartridge is in the slot by continuing to press the "yes" button after the "User 2" Table select value. If no cartridge is in the slot (or if one that is not formatted for micro tuning is in the slot), you will instead see an unfriendly "Cartridge format err!" message in the LCD. Note that micro tuning data can only be stored if the Store button is pressed while you are viewing the Micro Tuning Edit display of edit switch 14.

There is also a Recall Edit procedure for micro tuning data. The Recall edit function of edit switch 14 will allow you to recall micro tuning data alone from the edit recall buffer if you select the "Micro tuning" parameter. In this case, no other voice or performance edit parameters will be recalled - allowing you to link up an earlier tuning scheme to a new voice or performance memory you may have created.

Let's run an exercise now to create a completely inverted DX7II keyboard, and see just how we can store this new micro tuning in both internal and cartridge memory:

Exercise 85
Using micro tuning to create an inverted keyboard

1) Put your DX7II into performance play mode and call up the "Good Licks" performance memory from your ROM cartridge (bank 2, slot 22). Play a chromatic scale from C1 to C6 and listen (Audio Cue 85A).

2) This performance memory consists of two ROM cartridge voices ("ClaviPluck" and "Plukatan") being used in dual mode, as the lit status LED above the "dual" switch indicates. Press the edit mode select switch in order to enter performance edit mode. Press edit switch 29 repeatedly if necessary in order to call up the Note Shift display. **(see figure 14-18)** As you can see, both voice A and voice B have been shifted down by 12 semitones (-12). This shift (as well as the Key Transpose voice edit parameter, which for these two voices is normal) will affect our micro tuning, so use the data entry slider to change both voice A and voice B to new Note Shift values of +0.

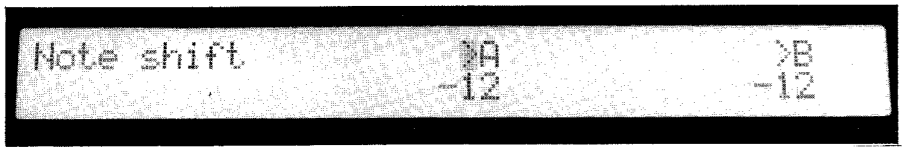


Figure 14-18

3) Press edit switch 29 twice more in order to call up the Micro Tuning display. **(see figure 14-19)** As the display shows, "Good Licks" was created with no micro tuning (that is, both voices are in equal temperament), since both voice A and voice B are OFF.

Figure 14-19

4) Position the cursor over "Voice A" and press the "yes" button in order to turn it ON. Do the same with voice B. Both voices are still equally tempered (since that is the preset currently in the Table select parameter), but now we will be able to hear both voices respond to any changes we make in the Micro Tuning Edit display of edit switch 14.

5) Accordingly, press edit switch 14 repeatedly if necessary in order to call up the Micro Tuning Edit display. Press the "cartridge" switch to step through all the keys and note that they all have Fine increment values of +0, reflecting the fact that this performance memory is equally-tempered. For this exercise, we'll simply re-tune only the notes of the physical DX7II keyboard (that is, C1 to C6). Therefore, press C1 and, while it is being held down, use your other hand to press either the "internal" or the "cartridge" switch. This brings C1 into the active window. The LCD now looks like **figure 14-20.**

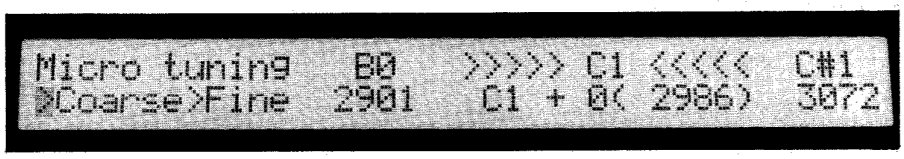


Figure 14-20

6) Position the cursor over the Coarse parameter and use the data entry slider in order to change the Coarse value of "C1" from C1 to C6. **(see figure 14-21)** Play C1 on the keyboard and listen (Audio Cue 85B). Note that it is now precisely the same note as C6!

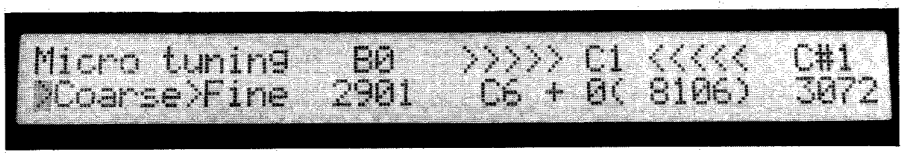


Figure 14-21

7) Press the "cartridge" switch in order to bring C#1 into the active window. The cursor should still be positioned over the Coarse parameter. Use the data entry slider to change the Coarse value of "C#1" from C#1 to B5. **(see figure 14-22)** Play C#1 on the keyboard and listen (Audio Cue 85C). Note that it is now precisely the same note as B5 (the B below C6).

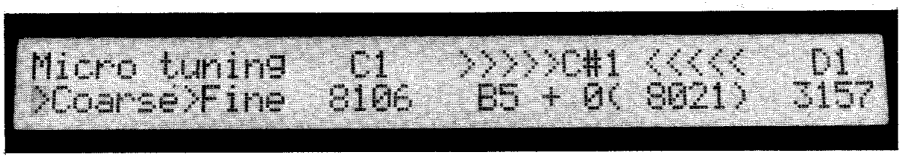


Figure 14-22

8) Continue this process throughout the entire keyboard, making D1 equal to A#5, D#1 equal to A5, E1 equal to G#5, etc., etc., until you have changed all the notes from C1 to C6 (at the end, C6 should be equal to C1). Play a chromatic scale from C1 to C6 and listen (Audio Cue 85D). We have completely inverted the entire keyboard, so that "higher" notes have lower pitch, and "lower" ones have higher pitch!

9) Now let's store this micro tuning into the internal memory. Press edit switch 14 three times in order to call up the Master Tuning display (be careful not to press any other edit switch or you'll temporarily lose this micro tuning. If this happens, don't panic - you can use the Micro tuning Recall edit procedure to get it back). Use the "no" button in order to turn the internal memory protection OFF. Press edit switch 14 again to return to the Micro Tuning Edit display. Now press and hold down the Store button. The LCD will look like **figure 14-23.**

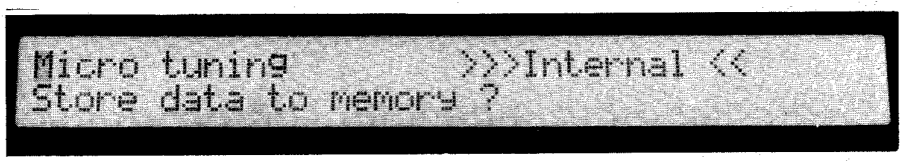


Figure 14-23

10) While keeping the Store button held down, press either main switch 1 or main switch 2, depending upon which of the two internal micro tuning slots you wish to store this data into. The LCD will echo your choice, and if it is correct, simply press the "yes" button with your pinky to complete the process. The LCD will read "Completed!" and you're in business! note that all we have done here is to store the micro tuning data *only*. If you want this performance memory to always come up with this inverted keyboard setup, you will need to re-save the performance memory using the Store procedure outlined at the beginning of this chapter.

11) If you have a blank or expendable RAM4 cartridge, you can try storing this micro tuning data to it as well. DO NOT DO THE FOLLOWING if your RAM4 cartridge contains data you wish to keep, since the formatting process will irretrievably erase all of its data. Turn the hardware cartridge memory protection OFF and place it in the slot. Press edit switch 14 three times in order to call up the software memory protection display and turn it OFF for the cartridge memory. Press edit switch 15 repeatedly if necessary in order to call up the Micro tuning cartridge display. Position the cursor over the Format parameter and press the "yes" button. If you are sure you want to format (and thereby erase) this cartridge, answer "yes" to the "Are you sure?" query in the display. A "Completed!" prompt will appear almost immediately if the format is successful.

12) Press edit switch 14 twice in order to get back to the Micro Tuning Edit display. Press and hold down the store button, and then press the "cartridge" button in order to tell the microprocessor that this data is to be sent to the RAM4 cartridge. The display will read like **figure 14-24.**

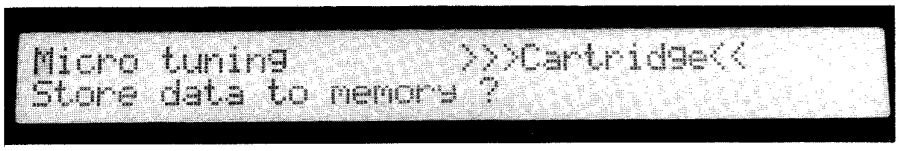


Figure 14-24

13) While continuing to hold down the Store button, press any one of the thirty-two main switches (plus the "1-32/33-64" switch if you want to store this data to a slot higher than 32), and, with your pinky, press the "yes" button. The display will read "Completed!" and this inverted keyboard micro tuning data is now stored in the equivalent RAM4 slot. You'll find that you cannot store micro tuning data into slot #64, however, for reasons that are known only to Yamaha.

14) As usual, turn all memory protections back ON. In order to now use this inverted keyboard with other performance memories, all you have to do is to return to performance play mode (by pressing the performance mode select switch) and call up any other performance memory, either from internal or cartridge memory. Then simply press edit switch 29 repeatedly if necessary in order to call up the Micro tuning display, turn voice A, voice B or both ON for microtuning, and position the cursor over the Table select parameter. Then use the data entry section to step through the eleven permanent presets and on to either "User 1" or "User 2" - whichever internal memory micro tuning slot you stored the data in. If you've stored this micro tuning in a RAM4 cartridge and that cartridge is currently in the slot, step beyond "User 2" to the 63 cartridge microtunings. Stop at the slot that contains

this inverted keyboard data, and - lo and behold! - this new performance memory will have an inverted keyboard as well. If you want to, you can store the performance memory as well so that whenever you call it up, it will automatically call up this micro tuning. If you instruct it to call up a cartridge micro tuning and the proper RAM4 cartridge isn't in the slot, however, you'll get a small "t" in the LCD, **(see figure 14-25)** indicating that the DX7II is looking for micro tuning data and not finding any. In this case, the performance memory will simply default to both voices OFF - in other words, equal temperament.

Figure 14-25

All of the adjustments we made to the scaling of the keyboard in this last exercise were Coarse adjustments - meaning that we changed the actual note value for each key. A more esoteric use of the Micro Tuning Edit function is to use the Fine control to change the gaps between the notes - meaning that we create new scales with existing notes. If we go far enough with this control as to create microtonal or macrotonal tunings (*macrotonal* tunings are those where each note is *greater* than a semitone), we'll even create brand new notes - notes which can be radically different to the ones that we are used to hearing with the Western musical system. Appendix E contains an example of this: an equally-tempered *fifteen-tone* scale created by Howard Sandroff of the University of Chicago, along with an example of how to use this new scale in a musical context.

Micro tuning is an area of the machine that few of you may explore in detail - but take my advice, it's worth it! This very powerful control not only can add an extra dimension to your sounds and can be musically inspiring - it can also be a lot of fun.

PAN controls

As we've learned, the DX7II is essentially two synthesizers in one. In either Dual or Split modes, you have two discrete voices available. These voices can be (and often are) totally different from one another in terms of how their edit parameters are set - meaning that they can sound quite different from one another. What's more, the audio signal generated by each of these voices can be routed independently - to the "A" and "B" output jacks on the rear of the instrument. This can occur even if you are working with a Dual performance memory or in Dual voice mode, so that even though both voices are triggered by the same key depression, the sound that each one produces can be completely isolated and separately processed after it leaves the instrument.

In order to achieve this total separation of the audio signals produced by voices A and B, you must do three things: First of all, you need to plug separate audio cables into each of the "A" and "B" jacks. Secondly, these cables must be plugged at the other end into separate audio channels - either two channels of a mixer, or the left-right inputs of your stereo amplifier. Last, but by no means least, you must turn the Pan control ON by pressing the "PAN" switch while in either voice or performance play mode. When ON, the status light above this switch will remain lit. The use of this dedicated play mode switch means that

you can pre-program in panning effects (we'll see how, shortly) and then dramatically introduce them with the touch of a single button while in performance play mode. The PAN switch moonlights as the right cursor switch when in edit mode, so you can only turn Pan ON in one of the play modes. Here's what such a setup will look like in **figure 14-26.**

Bear in mind that you can, of course, take these two output signals and route them any way you like *after* they leave the DX7II. The most common routing, however, is to simply send voice A to the left speaker, and voice B to the right (or vice-versa, if you prefer). This kind of audio routing will yield the widest *stereo image* (more about this shortly). Also note that when you first turn on your DX7II, the Pan light will be OFF. This is a power-up reset default, so if you want the audio outputs separated, you'll have to manually press this switch each time you turn the synthesizer on. It will then stay on until such time as you manually turn it off - or until you turn the power off.

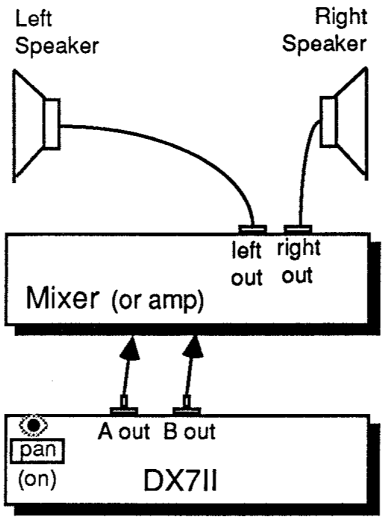


Figure 14-26

Panning is a term used by audio engineers to describe the act of moving a signal within a stereo left-right image. This is normally accomplished either manually by turning *panpots* (typically knobs which shift the signal from left to right or right to left), or automatically by *auto-panners* (electronic devices which are programmed to shift an audio signal from left to right or vice versa in a particular way at a set speed). The DX7II is one of only a handful of synthesizers that not only allows for separation of the audio outputs, but also allows for user-programmed auto-panning of these signals.

It should be pointed out that this whole area of the instrument is completely optional, and that you can instead opt to take a single (*mono*) output from the DX7II. This is accomplished by simply plugging a single audio cable into the "A" jack (also labeled "MIX") and leaving the Pan control OFF (status light unlit). You will be missing a *lot*, however, by not programming in panning effects for your performance memories - so if you possibly can, use both outputs in order to utilize this instrument as the true *stereo* device that it is.

Let's just take a moment to define the term "stereo". If your last name is not Van Gogh, we can assume that you listen to sounds with two ears, and not one. This means that the typical human being's perception of a sound not only includes pitch, timbral, and amplitude perception, but spatial perception as well. In other words, we can also tell "where" a sound is coming from. This is accomplished through a very complicated series of events within the nervous system, but essentially our brain is cross-checking to see whether more signal entered through the right ear than the left, or vice versa. In this way, the very complicated computer we all carry around with us permanently (you know, the one above your neck) is able to calculate the approximate position of the source signal. A stereo image, then, is one which uses two discrete loudspeakers. Within these two speakers, an engineer is able to place signals either more in one or the other, thus producing panning effects. This left-right imaging is only one of many ways that audio engineers shape the total signal. Good engineers tend to think in three planes when mixing audio signals together. **(see figure 14-27)**

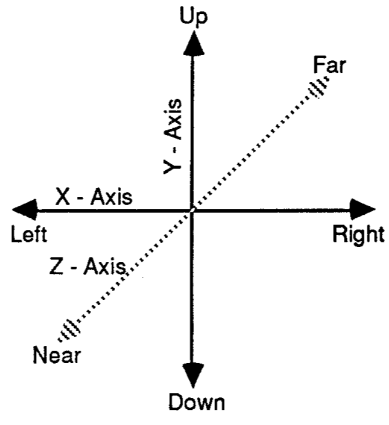


Figure 14-27

The positioning of a signal within the vertical plane (y-axis) is largely frequency-dependent. For example, bass drums tend to sound closer to the floor, while high-hats and cymbals tend to sound closer to the ceiling when you are listening to a good, well-separated stereo signal over quality loudspeakers. The positioning of a signal within the

near-far plane (z-axis) has to do with the amount of "depth" a signal is given through the use of reverberation and echo effects. Sounds which are very dry tend to exhibit a great deal of "presence" and are perceived as very close to us, while sounds which are reverberant and echoey sound more distant. Combine these with left-right panning (the x-axis) and you can see that the audio engineer has the tools to make any sound truly three-dimensional.

But these are audio treatments which are normally accomplished *after* the signal leaves the instrument. What makes the panning control of the DX7II so special is that it allows us to do auto-panning of its stereo signal directly from the front panel of the instrument, and to do so in a way that can be interactive with our real-time playing of this instrument! Of course, you can still add more externally-controlled panning as well, and you still have the ability to shape the sound within the y- and z-axes. But the DX7II will allow you to place the output of either or both voices anywhere along the x-axis, and to then *move* that signal during the course of playing.

All of these marvelous effects are set up with the use of edit switch 30, labeled "PAN". This switch offers the three LCD displays, accessed as usual by pressing the switch repeatedly, shown in **figures 14-28(a) through (c).**

Figure 14-28(a)

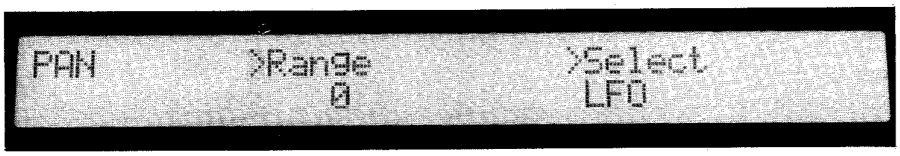


Figure 14-28(b)

Figure 14-28(c)

Let's begin by discussing the Mode parameter, as shown in **figure 14-28(a)**. There are four different Pan modes, called mode 0, mode 1, mode 2, and mode 3. You select one of these modes simply by using the data entry section while viewing this display. These modes are as follows:

Mode 0

In this mode, the audio outputs of both voice A and voice B is combined internally and sent to either or both audio output jacks according to the panning effects you select with the other Pan parameters. In this way, the total signal will move from left to right, or vice versa. It's worth noting that this is the only Pan mode that can affect a performance memory consisting of a Single voice; all the other modes are available only if the performance memory is made up of two

Pan Mode 0:

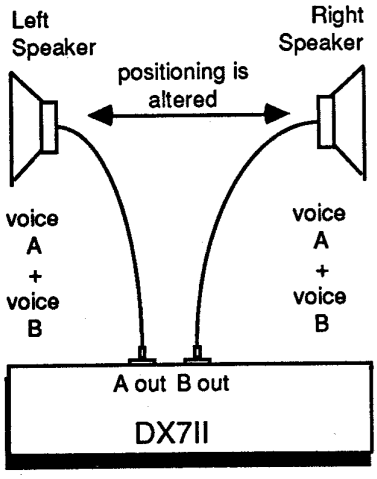


Figure 14-29

voices in Dual or Split mode. If you create a performance memory from Single voice mode, the "Mode" parameter will have a "-" under it, indicating that it has defaulted to mode 0 and that no other choices are available. A block diagram of what mode 0 does to the audio signal is shown in **figure 14-29.**

Mode 1

This mode is available only if the performance memory is created from Dual or Split mode, and is, incidentally, the initialization default when in those voice modes. When in Pan mode 1, the audio outputs of the two voices remains separated. The relative level (that is, volume) of the two voices, however, will be affected by the other Pan parameters so that as one increases, the other decreases. In this way, voice A may get relatively louder while voice B simultaneously gets softer (or vice versa), causing the illusion of left-right movement - which not coincindentally is the way that mixing desk panpots and auto-panners work. When you select this Pan mode (or when you first enter initialized performance edit mode with Dual or Split voices), the display simply shows you that both voice A and voice B are ON. **(see figure 14-30)**

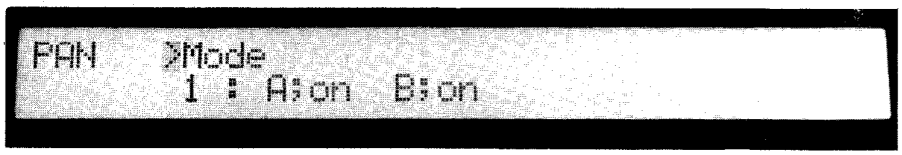


Figure 14-30

A block diagram of what mode 1 does to the audio signal is shown in **figure 14-31.**

Mode 2

This mode works much like mode 1 in that both voice outputs remain separate. However, only the level of voice A will be affected by the other Pan parameters; voice B will remain at a stable output level - a fact which is reflected in the LCD display ("voice A = ON" while "voice B = OFF"). This will create the effect of one signal remaining stable while the other one changes. **(see figure 14-32)**

This mode will be particularly useful when you are working with a performance memory made from Split mode, since it will allow one of the two voices to be panned while the other stays at the same level. Interesting effects can also be garnered from using Mode 2 in Dual mode, as well.

Mode 3

This is exactly the same as mode 2, except that only voice B is affected, while voice A remains stable. **(see figure 14-33)**

When in mode 3, the LCD will show voice A as being OFF, while voice B is ON.

Having selected the Pan mode (and, of course, you are limited to one Pan mode per performance memory), press edit switch 30 a second time in order to call up the Range/Select display, as shown in **figure 14-28** above. The Range parameter allows you to specify how much effect panning will have; this is your depth control, with potential values of 0 through 99. A Range value of 0 will indicate no panning whatsoever from any of the sources offered by the Select parameter (and these will

Pan Mode 1:

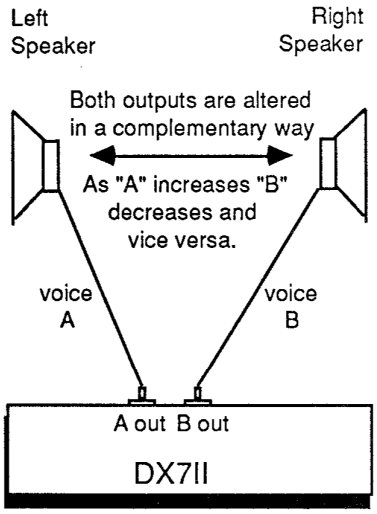


Figure 14-31

Pan Mode 2:

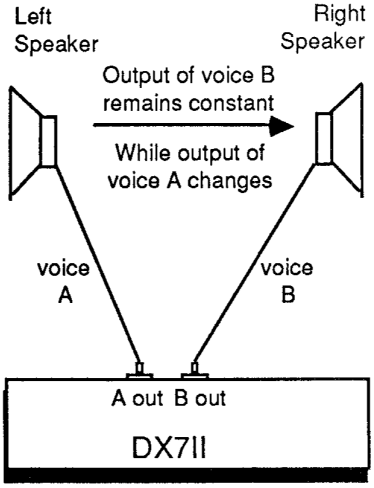


Figure 14-32

Pan Mode 3:

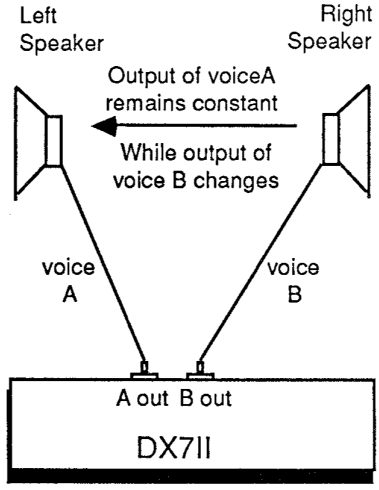


Figure 14-33

be discussed shortly). Therefore, if you're certain that you don't want any panning at all from these sources, set the Range to 0. 0 is also the performance initialization default for this parameter, so that there will be no panning from the Select sources when you first enter initialized performance edit mode. A Range value of 99 indicates maximum depth, making the panning movements as severe as they can be. Intermediate values will simply cause less radical panning changes.

The Select parameter allows you to choose one of the following three options:

LFO

Here, the LFO of voice A will alter either the positioning of the combined audio signals (if in mode 0) or the output level of either voice A (in mode 2), voice B (in mode 3), or both (in mode 1). The speed of this change will simply be that LFO's Speed, as set with edit switch 12, and, more importantly, the *shape* of the panning movement will be determined by the LFO Wave, also set with edit switch 12. For example, a sine LFO Wave will cause a smooth change in either positioning or level, while a s/hold LFO Wave will cause a random change. Other LFO parameters, such as Delay and Sync, will also be active when using the LFO for the Pan function. It is worth pointing out that using the LFO in this way does *not* preclude any other uses you may have programmed for it in voice edit mode - they will occur simultaneously. This can actually be problematic. For example, if you've set your LFO up for a particular vibrato effect but you want your performance memory panning at a slower speed than the vibrato, you are unfortunately out of luck. I believe it would have been better design if Yamaha had included a separate LFO, dedicated exclusively to panning - but, as usual, we've got to learn to live with what is here, not what we wish was here. We'll run an exercise shortly to try out the various LFO panning effects.

Velocity

Here, the positioning or output level of one or both voices (depending upon the Pan mode) will be affected by the speed of the key depression, as described in the "Keyboard Velocity" section of Chapter Eleven. In mode 0, slower key depressions (softer touches) will place more audio signal in output jack A, while faster key depressions (harder touches) will position it more in output jack B. By altering your keyboard touch, then, you can actually change the position of the sound in the left-right axis! In the other Pan modes, the output level of the affected voice or voices will be controlled by this keyboard velocity. One very clever application of this source is to use it in Pan mode 1, while taking both audio outputs of a Dual performance memory and putting them in mono at your external mixer or amplifier. This will allow you to do *velocity cross-fades,* between the two voices, so that harder-struck notes will give you more of voice A, and softer-struck notes more of voice B. We'll examine the general use of the Velocity source in an exercise very shortly.

Note number

This is a unique control which allows you to control the positioning or output level of one or both voices (again, depending upon the mode selected) according to which key you depress. For example, in mode 0, lower notes will send more of the combined audio signal to output jack

A, while higher notes will send more to output jack B. The permanently set "split point" for this control is middle C (C3); in other words, playing C3 will cause equal amounts of signal to be sent to both jack A and jack B - thus positioning the signal in dead center, if one output is externally routed to hard right and the other to hard left. In modes 1, 2, or 3, playing different keys will change the output level of one or both voices (depending upon the mode selected). We'll try this out also in the upcoming exercise.

Again, the depth of the effect of whichever one of these three sources you choose will be determined by the Range control. The initialization default Select value is the LFO, which really is completely irrelevant since the default Range is 0. You'll therefore have to enter in a new Range value before the Select value has any meaning.

As if all of this wasn't enough, the good folks at Yamaha thought that you might want some *aperiodic* control over the panning as well as the *periodic* (LFO) and *real-time* (Velocity and Note number) controls. Therefore, if you press edit switch 30 yet again, you'll see (as shown in **figure 14-28(c)** above) the parameters for the dedicated *Pan EG*.

The good news here is that the "mechanics" of the Pan EG are exactly the same as those of the operator EGs or the pitch EG: there are four Levels, and four Rates of movement between them. They are tied in, as per usual, with the "key on" and "key off" flags. In order to avoid redundancy, then, the reader is advised to refer to Chapter Nine for a detailed discussion of how the DX7II EG does its thing. The only difference between the Pan EG and the other seven EGs we have studied is that it is (unlike the LFO) completely dedicated to panning effects only. It therefore controls either the positioning of the combined audio signal between the two output jacks (if you are in mode 0), or it controls the total output level of voice A, voice B, or both (in modes 2, 3, and 1, respectively). The default values for the Pan EG are the same as those for the Pitch EG: all Levels are at 50, and all Rates at 99. A Pan EG level of 50 will have the effect of either positioning the signal equally in jack A and jack B (in mode 0) or having no effect on the output level (in the other Pan modes). Levels less than 50 will shift the combined signal more to jack A (in mode 0) or attenuate the output level of the affected voice or voices (in the other Pan modes). Levels greater than 50 will shift the combined signal more to jack B (in mode 0) or boost the output level of the affected voice or voices (in the other Pan modes). A diagram that shows how it works in mode 0 appears in **figure 14-34.** Here's how it works in modes 1, 2, or 3. **(see figure 14-35)**

All of these panning effects are *global*, meaning that Pan is essentially a monophonic control. In other words, *all* the voices will be repositioned or have their output levels adjusted simultaneously, according to the latest information received by the microprocessor. Let's suppose, for example, that you are in mode 0. We'll assume that the Pan EG is idle (with all Levels at 50), and that Note number is being used as the sole panning control, with maximum range. If you play C1 on the keyboard, all of the signal will be routed to jack A. But if you keep C1 held down and then play C6, *both* notes will jump over to jack B. If you play rapid chords, the *most recent* note values (as determined by the super-fast microprocessor) will cause the entire signal to shift position. This will cause wonderful pseudo-random movements in the total sound!

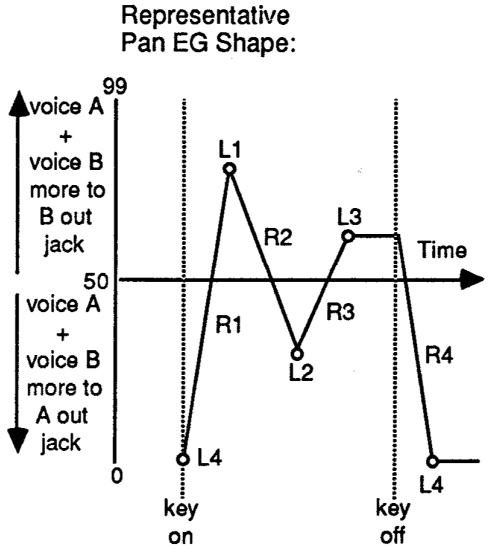


Figure 14-34

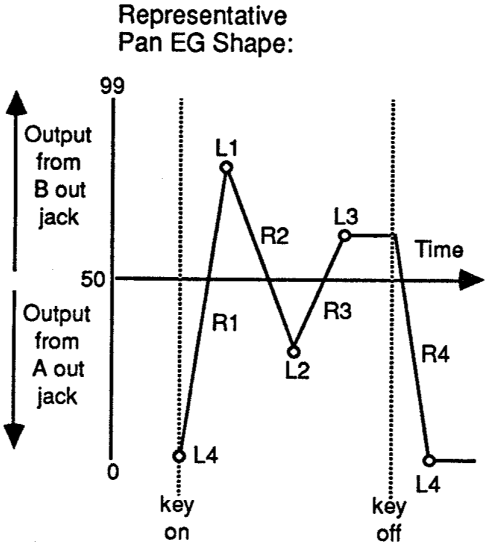


Figure 14-35

The other complicating factor is that the Pan EG works *alongside* whatever other Pan control you designate with the Select parameter in the Range/Select display, and in a complementary fashion. For example, the LFO may be altering the positioning or output level - but this will happen along with the actions of the Pan EG. Alternatively, positioning or output level may be changing according to Note number - but, again, this will be happening along with the actions of the Pan EG. If the LFO, Note number, or Velocity is telling the DX7II to "shift the output more to jack A", and the Pan EG is simultaneously telling it to "shift the output to jack B", the end result will be output sent to *both* jacks. In this way, you can set up constant "push-pull" situations within the panning controls. All of these tend to make the effects you can generate with these controls much more complex - and therefore much more interesting.

Let's run an Exercise now in order to get more familiar with some of the Pan controls:

Note: The audio cues presented in this Exercise are in stereo. The reader is therefore advised to play back the soundsheet (or tape recording of the soundsheet) in the widest possible stereo image, with the left signal being routed completely to one speaker and the right signal being routed completely to another. For best results, we recommend listening to these cues (as well as to the output of your DX7II) with a pair of good quality stereo headphones while running this exercise.

Exercise 86
Using the Pan controls

1) Plug separate audio cables into the "A" and "B" output jacks on the rear of your DX7II and route them to your two speakers in a hard-left (voice A) and hard-right (voice B) configuration, as shown in figure 14-26 above.

2) Put your DX7II into Dual or Split voice play mode and press the "A/B" button repeatedly if necessary in order to turn its status light OFF. This step is necessary if we want to be able to VIEW voice A parameters while in performance play mode (since the light will stay off when we change modes). Now go to performance play mode and call up the "St. Elmos StringBell" performance memory preset from your ROM cartridge (bank 1, slot 15). Note that this cartridge performance memory uses *internal* voices 1 ("Warm Stg A") and 20 ("St. Elmo's"), so you'll have to make sure that these particular voices are currently in your internal memory. The easiest way to do this is to simply call up bank 1 from your ROM cartridge and then use the "Load" procedure to load all of its voices into your internal memory. Take care, however, to first *save* your internal voices somewhere on a RAM4 cartridge or on disk first, if you have created original voices or performance memories. If your Pan switch is OFF (status light unlit), leave it there. If it is currently ON (status light lit), turn it OFF. Play a few notes and listen (Audio Cue 86A). Note that both the "Warm Stg A" voice and the "St. Elmo's" voice presets (these are the two voices used in this performance memory preset) are coming out of both audio jacks - they are present in both your speakers and therefore sound in mono (as opposed to stereo) when both speakers are on.

3) Press the "Pan" switch so that its status light is lit. Play the same few notes and listen (Audio Cue 86B). Note that the sound is

now in stereo, with "Warm Stg A" (voice A) coming from your left speaker alone and "St. Elmo's" (voice B) coming from your right speaker alone.

4) Press the edit mode select switch in order to enter performance edit mode. We are not entering an initialized performance edit mode, since we preceded this step by pressing one of the thirty-two main switches (this was done in step 1 when we first called up this performance memory). Press edit switch 30 repeatedly if necessary in order to call up the Mode display, as shown in **figure 14-28(a)**. Observe that the current Mode is Mode 1. Press edit switch 30 again in order to call up the Range/Select display, as shown in **figure 14-28(b)**. Observe that the Range is currently at a value of 0, indicating that Velocity (the Select value) is currently having no effect on panning. Press edit switch 30 one more time in order to call up the Pan EG display (as shown in **figure 14-28(c)**. Observe that all four Pan EG levels are currently at values of 50, indicating that the Pan EG is having no effect on panning either.

5) Press edit switch 30 once again in order to call up the Mode display, and press the "no" button in order to change this to Mode 0. Play a few notes and listen (Audio Cue 86C). Note that, just as in step 2 above, both voices are now coming out of both speakers. Remember that in Pan mode 0, the two outputs are mixed together in equal proportions. Because neither the Pan EG nor any other control is altering the panning (since the Range value is 0), the signal is currently simply being sent to both speakers, with no movement.

6) Now we'll get things moving in the stereo image! Press edit switch 30 again in order to call up the Range/Select display and position the cursor over the Range control. Use the data entry slider in order to change this to the maximum value of 99. Leave the Select parameter at its current value of "Velocity". Play a few notes as slowly as possible and listen (Audio Cue 86D). note that we now hear a much softer combined sound (this is because most of the carriers in both sounds are sensitive to keyboard velocity), and, more importantly, that it only comes from the left speaker! Now play the same few notes, but depress the keys as quickly as possible (you'll need to use some reasonable force here) and listen (Audio Cue 86E). Note that we now hear both sounds quite a bit louder, and that they both come from the right speaker only!

7) The explanation as to why these two sounds have migrated from speaker to speaker is simple: we have applied maximum Range (99) to the Velocity parameter. Therefore, notes struck with lesser speed will cause the mixed output (since we are in mode 0) to be routed more to jack A, while notes struck with greater speed will cause the mixed output to shift to jack B. Try playing a series of staccato notes at differing velocities and listen (Audio Cue 86F). Note that you can actually shift the position of the sound with the force of your key depressions. As mentioned above, panning controls are monophonic - the most recently played note will shift all previously played ones together. To prove this, play a chord very lightly and hold it down. While continuing to hold it down, play a series of forceful staccato notes with your other hand. listen (Audio Cue 86G). Note how the addition of new notes shifts even the old, held chord, within the stereo image.

8) Position the cursor over the Select parameter and press the "yes" button in order to change this to Note Number. Play a chromatic scale from C1 up to C6 and listen (Audio Cue 86H). Note that lower notes

appear more from the left speaker, while higher ones appear more from the right. Hold down a low chord and, while keeping it held down, add new notes - higher and lower. Listen (Audio Cue 86I). Note that the new notes shift the entire signal - even the old, held notes - accordingly. Try playing some chords and note that the image moves in seemingly unpredictable ways. What's happening here is that the microprocessor - faster than we can imagine - is "seeing" which key in the chord was depressed most recently and it is shifting the position of the sound accordingly.

9) Press the "no" button twice in order to change the Select parameter to LFO. Play a few notes and listen (Audio Cue 86J). Note that the mixed signal is now sweeping rapidly back and forth between the two speakers. Press edit switch 12 in order to VIEW the LFO parameters for voice A (the "A/B" status light should currently be OFF, indicating that we are viewing voice edit parameters for voice A only - if you carefully followed the instructions in step 2 above). Although we cannot store any voice edit changes from performance edit mode, we can still VIEW voice edit parameters and temporarily change them if we wish. Observe that the LFO Speed is currently at a value of 30. Position the cursor over this parameter and use the "no" button to slowly change this down to the minimum of 0, holding down a chord and listening as you do so (Audio Cue 86K). Note that the speed of the signal shift between speakers slows down appreciably and becomes very subtle and pleasant at the lowest LFO values. Use the data entry slider in order to change the LFO Speed value to 4.

10) The LFO Delay parameter also functions as a panning delay when the LFO is responsible for panning movements (as it is right now). Note that it is currently at a value of 15, indicating very little delay in the onset of the LFO effect. Position the cursor over this parameter and use the data entry slider to change this to its maximum value of 99. Play a held chord and listen (Audio Cue 86L). Note that the shift in position only occurs after about three seconds, and that until that time, the audio signal simply remains in the center position. Experiment by trying different Delay values, and when you are done, change the value to 0, indicating no LFO delay.

11) The LFO Wave parameter will dictate the "shape" of the panning movement. Observe that it is currently a sine wave, accounting for the smooth and regular movement we are hearing. Position the cursor over the Wave parameter and press the "yes" button in order to change this to a s/hold waveshape. Play a held chord and listen (Audio Cue 86M). Note that the sound now randomly and abruptly travels from point to point in the stereo image, with no smooth changes in position. Press the "no" button five times in order to change the Wave to a triangle wave. Play a held chord and listen (Audio Cue 86N). Note that the sound now moves smoothly back and forth between the two speakers, in a manner similar to (but not quite as gentle as) the sine wave we heard originally. The LFO key Sync function also affects the panning movement. This control will enable us to hear the effects of the remaining LFO waves (saw down, saw up, and square) more clearly, so position the cursor over the Sync parameter and press the "yes" button in order to turn it ON. Now reposition the cursor over the Wave parameter and press the "yes" button in order to change this value to a saw down Wave. Play a held chord and listen (Audio Cue 86O). note that the mixed audio signal now starts instantly in the right speaker (since the LFO key Sync is ON), and then slowly moves towards the

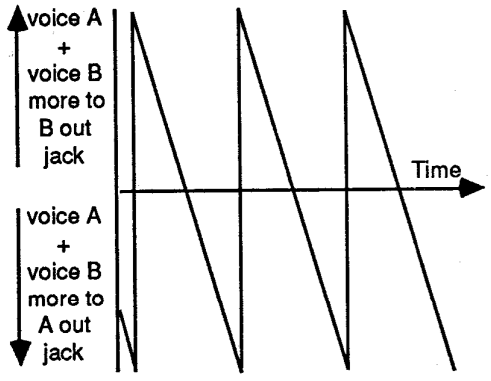


Figure 14-36

left speaker. Once it reaches the far left point in the stereo image, it then immediately "springs back" to the right-most point. **(see figure 14-36)** Press the "yes" button in order to change the LFO Wave to a saw up. Play the same held chord and listen (Audio Cue 86P). Note that precisely the reverse effect is now generated. **(see figure 14-37)** Finally, press the "yes" button once again in order to change the LFO Wave to a square wave. Play the held chord and listen (Audio Cue 86Q). Note that the mixed signal now jumps back and forth between the hard-left and hard-right points, with no stops in-between. **(see figure 14-38)**

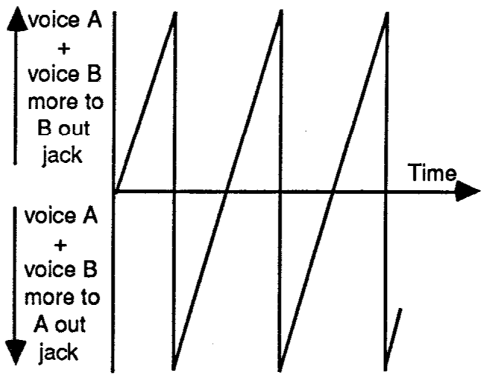


Figure 14-37

12) Because the Range value has been 99 up till now, we have heard all of these panning effects at their most extreme - the signal has been traveling through the full 180 degrees of the left-right plane. Press edit switch 30 repeatedly if necessary in order to call up the Range/Select display, and change the Range value to 50. Now repeat steps 6 through 11. Note that every change you make - to Select, to LFO Speed, Delay, Sync, and Wave - has less radical impact upon the movement of the sound. When you are clear on how the Range control affects things, restore it to 0, restore all LFO settings back to their preset values (use Compare mode to VIEW these prior settings), and press edit switch 30 once more in order to call up the Pan EG display.

13) Currently, as noted above, the Pan EG is having no effect on the sound, since all of its Levels are at 50. Let's change a few of these so that we can hear the effects of the Pan EG. Change L2 to 0 and L3 to 99, and change both R2 and R3 to 45. This gives us an EG which looks like **figure 14-39.** In panning terms, this will cause our mixed signal (since we are still in Mode 0) to move from center (L1) slowly to hard left (L2), then slowly to hard right (L3), where it will stay (since L3 is the sustaining panning point) until we release the key, at which time it will spring back to the center position (L4). The only reason we will be able to actually hear this last spring-back is because R4 is significantly slow in the carriers of both voices in this performance memory. Try it! Enter in these Pan EG values, play a held note, and listen (Audio Cue 86R). Note that the total sound undergoes precisely these movements, as predicted.

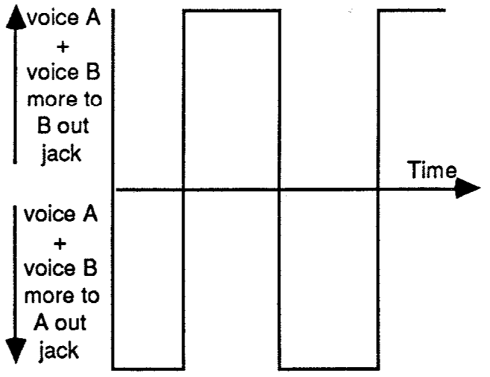


Figure 14-38

14) As with the other Pan controls, the effect of the Pan EG is monophonic. Each new note you play will therefore re-trigger its effect. Try it! Play a note and hold it down long enough for it to reach the far right-hand point of L3. Then add a new note and listen carefully (Audio Cue 86S). Note that the addition of the new note pulls the entire signal back into the center of the image.

15) Setting Level values of 0 and 99 give us full hard left-right movements, but setting Level values closer to the center point of 50 will give us movements that stay closer to the center image. Try changing L2 to 20 and L3 to 80. Play the same held note and listen (Audio Cue 86T). Note that the signal now never gets all the way to the left or all the way to the right. experiment further with different EG values and note that pretty much all of the rules that apply to the operator and pitch EGs apply here.

16) Finally, let's see how the Pan EG movements can interact with another Pan control. Restore the Pan EG values to those given in the beginning of step 13 above: L2 to 0, L3 to 99, and R2 and R3 both to 45. Press edit switch 30 twice more in order to return to the Range/Select display. Position the cursor over the Range parameter and change this to its maximum value of 99. Position the cursor over the Select

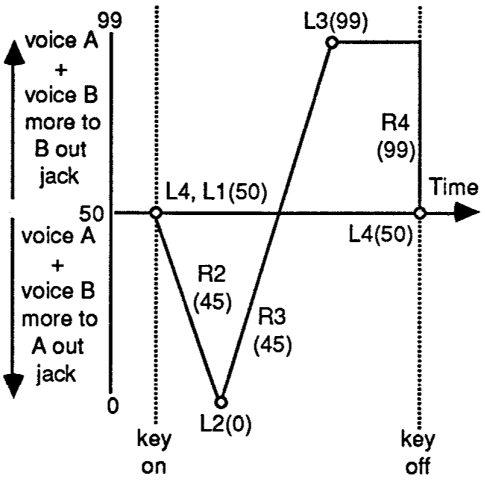


Figure 14-39

parameter and change this to Note Number. Now play a series of slow, arpeggiated notes and listen (Audio Cue 86U). Note that the two panning effects interact in a subtle and interesting way: the Note Number determines the starting point of the Pan EG, which then imposes its slow hard-left to hard-right movements. When you release the key, the signal does *not* necessarily return to the center point, but instead returns to whatever starting point was dictated by its Note Number. This is only one of many, many different ways that the Pan EG can interact with another Pan control. experiment with others until you feel comfortable with the ways that these interactions manifest themselves.

17) Up until this point, we have solely been working with Pan mode 0. This has had the effect of shifting the mixed output of both voice A and voice B through the left-right axis. Change the Mode to mode 1, and repeat steps 6 through 16, listening carefully to the effect as you do so. In mode 1, the output signals of both voice A and voice B remain separated, and their levels are modulated by the Velocity, Note Number, LFO, and/or Pan EG. Experiment further by changing to mode 2 and repeating steps 6 through 16, again listening carefully as you do so. Here, only the "Warm Stg A" voice will have its output level affected, while "St. Elmo's" will remain at a constant output level, and will be routed to output jack B only. Finally, change the Mode to mode 3, and listen to the effects of steps 6 through 16 as "St. Elmo's" output level changes, while "Warm Stg A" remains constant.

In summary, the Pan controls allow you to take a synthetic and highly directional sound and to change it into an ambient one. The controls provided here - while having limitations - are extraordinarily powerful. This is one area of the machine that I strongly advise you to spend a good deal of time with, as it really does put the DX7II a step ahead of most other synthesizers.

This concludes our discussion of all of the DX7II edit parameters, except for the MIDI controls and "MDR" function of the disk drive, both to be covered in the next chapter. It's been a long journey, but hopefully it's been a pleasant one, too. Keep up the good programming, people - this synthesizer would, like all others, be pretty useless without good sounds for it.

Let's move on now with a discussion of the ways in which the DX7II can communicate with other synthesizers and with computers, as we talk about a small revolution called MIDI.

Chapter Fifteen ◇
MIDI

Lack of standardization has been a problem which has plagued every consumer industry almost since the day the first consumer appeared on this earth - one sometimes wonders if the Neanderthals had trouble getting spare parts for their clubs! In no industry, however, has this created more headaches and gray hairs than in the computer industry.

MIDI was born in a valiant attempt to solve this problem, at least in so far as computerized synthesizers were concerned. To a large degree it has succeeded. But one could make a good case that MIDI has caused as many problems as it has solved. Confused? Let's start at the beginning.

First of all, the word *MIDI* is an acronym for the Musical Instrument Digital Interface. The term "interface" is common computerese for any hardware or software device that allows two pieces of equipment to work together. For example, between a computer and a printer, one needs an interface, which may consist of anything from a simple cable to a complex logic circuit board. In a similar manner, the interface between a record player and your ears is a complicated system of equipment, including an amplifier, loudspeakers, and several interconnecting wires (see Chapter One for a thorough description of this process). In the instance of MIDI, the "interface" is two things: a standardized type of required hardware (here we are talking about the MIDI transmitter, receiver, and connecting cables), and a common software *language* which will allow synthesizers of all varieties - even those made by different manufacturers - to communicate and work with one another.

Quite simply, the key to understanding MIDI is to think of it as being a language. Assuming you are currently reading the original English version of this book, the reason you are able to understand these words and work with the information contained in these pages is because these words are written in the English language, and because you understand the English language. Similarly, if we take two synthesizers which both have been programmed to "speak" and "understand" MIDI, they will be able to communicate with one another.

How does this work? Well, we have discussed the fact that today, all synthesizers of all types have one very important thing in common: they all contain *microprocessors*. In the case of the DX7II, this wonderful circuit is actually responsible for creating the ultimate sounds we hear. In the case of analog synthesizers, the microprocessor will be performing much simpler "housekeeping" tasks, like calling up different voltage settings and storing them in memory slots, or keeping track of which notes are depressed on the keyboard and when (again, refer to Chapter One for a detailed discussion of this). The point is, no matter what different tasks the digital circuitry is responsible for, the important

fact remains that a microprocessor is present in all these different machines.

Another thing we have learned is that microprocessors "think" only in terms of numbers - specifically, ones and zeroes. It is this feature which allows data stored in computer memories to be transferred to other computers. We've all encountered this when we've gone to our bank and the teller has turned to a computer in order to find out our balance. What the teller is actually doing is quite interesting: He or she is using a computer terminal to telephone (via a device called a *modem*) a large central computer in which all the bank's customers current balances are stored. Similarly, when you go to a travel agent and their computer is able to instantly determine if a seat is available on a flight you wish to take, a central computer is also being "interrogated" via standard telephone lines. This central computer contains up-to-the-minute data regarding all major airline flight bookings. The modem (which is short for "MOdulator-DEModulator") takes the digital commands sent out by the computer terminal and routes them through a DAC, converting them to electrical signal, which is then modulated (sound familiar?) by a steady audio tone. This modulated audio signal is then passed down standard telephone wiring where it is de-modulated at the other end: the steady audio signal is filtered out and the changing electrical signal is fed to an ADC, which converts it back into the original ones and zeroes.

Ah, the wonders of modern technology! All this so they can *still* screw up your bank statement at the end of the month! Anyway, I digress...

The point here is that computer data is very highly standardized: when you get down to it, all it consists of is a stream of ones and zeroes. Therefore, in order to use this data, we need to pre-program the different microprocessors (the logic circuits) within different computers so that they can consistently make sense of all these ones and zeroes. In order to do that, we need a language. If the microprocessor in synthesizer A can send out its stored data in a particular format that the microprocessor in synthesizer B can understand, and vice versa - well, then, we have COMMUNICATION. And that's precisely what MIDI is - it's a standardized format for communicating synthesizer data, which allows all these different instruments to interface with (that is, work with) one another.

The concept of MIDI was actually born in a large polyphonic analog synthesizer called the Prophet-10, made by Sequential Circuits. This cumbersome machine (originally introduced in 1981 and manufactured until 1983) was essentially two Prophet-5s in one box. Each had its own keyboard, and the instrument also contained a polyphonic digital *sequencer* which could act as a master controller. A digital sequencer is a device which performs a function similar to that of a tape recorder (even though it doesn't use tape, but digital memory instead). It remembers notes played and timing values between the notes played. In any event, the interesting idea behind the Prophet-10 was that each of the two independent synthesizers in it, as well as the sequencer, could interact with one another in various different ways. What this instrument contained, then, were three independent microprocessors that needed to communicate with one another and to be able to send data stored in each of the three independent memories back and forth to one another. **(see figure 15-1)**

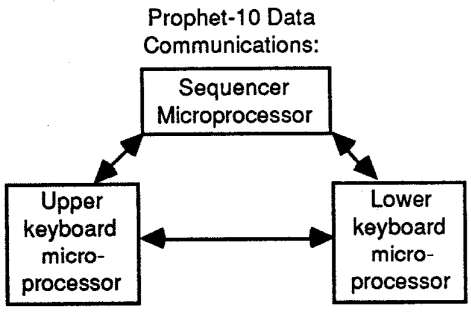


Figure 15-1

Since at that time there was no standardized means of accomplishing this, the software engineers at Sequential simply went ahead and invented their own *protocol* (a set of rules for doing things) for achieving these internal data transfers. Every language has its own particular protocol - for example, in the English language we know that every sentence must begin with a capital letter and must end with either a period, question mark, or exclamation point. In the spoken English language, it is understood that when asking a question, one raises pitch at the end of sentence, and when making a statement, one lowers pitch at the end of a sentence (try it - you'll find that you sound awful funny if you consciously break these rules). Similarly, the Sequential engineers had to not only organize the internal synthesizer data in such a way that all three microprocessors could use it, but also had to invent rules for transmitting and receiving such data. For example, there had to be a particular number which meant "beginning of statement" and a different number which meant "end of statement". There had to be identification numbers which, for example, let microprocessor 1 know that it was hearing from microprocessor 2 and not microprocessor 3. There had to be numbers which signified requests for specific data, and others which signified acknowledgements of data or data errors received. In short, an entire new language had to be invented from scratch!

At the time when all this was being developed (the early '80s), it was becoming apparent that soon all synthesizers of all types would have microprocessors onboard - and there were hints that the affordable digital synthesizers of today were just around the corner. The forward-thinking executives of Sequential realized that this provided the potential for some kind of universal interface which would allow all these disparate instruments to communicate data back and forth, and, perhaps more importantly, to allow them all to communicate with standard personal computers. The rise of the computer industry in the last two decades has unquestionably been the most dramatic of any industry in human history - and one that has had perhaps the greatest impact on our everyday lives. Several different synthesizer manufacturers - Sequential, Oberheim, and Roland, among others - saw the great opportunity being presented: that of tying in the relatively small synthesizer industry with the massively greater computer industry - and, to their credit, they did not keep the idea to themselves.

Instead, at the 1981 Audio Engineering Society convention in New York, the president of Sequential Circuits, Dave Smith, presented a paper outlining Sequential's work with the Prophet-10 and proposing a universal standard interface (at that time, actually called the USI!) that could be adopted by ALL synthesizer manufacturers. This original proposal was then passed on to the large Japanese synthesizer manufacturers - Roland, Yamaha, and Kawai. After nearly two years of back-and-forth, what became the final MIDI 1.0 specification was finally born. Within a year of its inception, every major synthesizer manufacturer in the world - even those who had originally voiced the most strenuous objections - put MIDI in their instruments.

And that's a curious statement unto itself - how do you "put" MIDI in an instrument? Well, this means two things. Number one, it means that a digital transmitter/receiver circuit (called a "UART", for Universal Asynchronous Receiver Transmitter) which can "understand" the MIDI protocol (and send and receive MIDI data to and from the instrument's own microprocessor) is installed and, secondly, that the appropriate

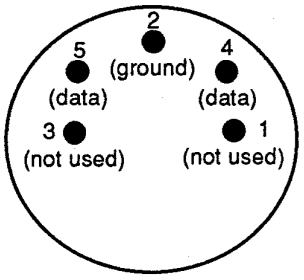


Figure 15-2

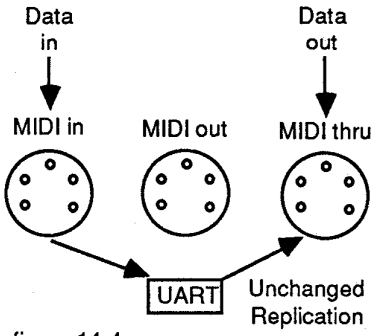


figure 14-4

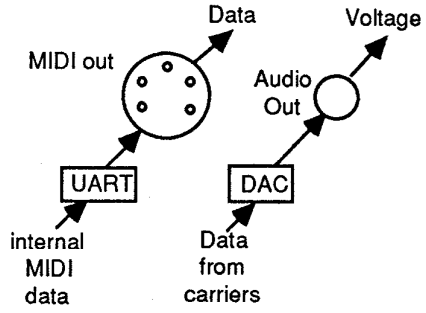


Figure 15-5

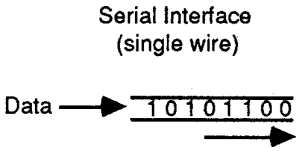


Figure 15-6

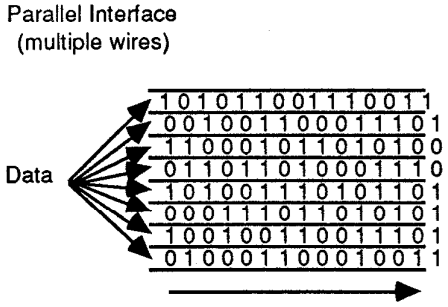


Figure 15-7

hardware connections - the MIDI jacks - are provided. These jacks are standardized, as are the cables which are meant to interconnect different machines. So as not to confuse the MIDI signal with audio signal, the MIDI spec calls for 5-pin DIN connectors (which are rarely if ever used in audio cables) to be utilized, even though only three of the five pins are actually wired up. **(see figure 15-2)**

A synthesizer that speaks MIDI will typically have three such jacks, labeled "MIDI IN", "MIDI OUT", and "MIDI THRU", as you will find on the back of your DX7II. **(see figure 15-3)**

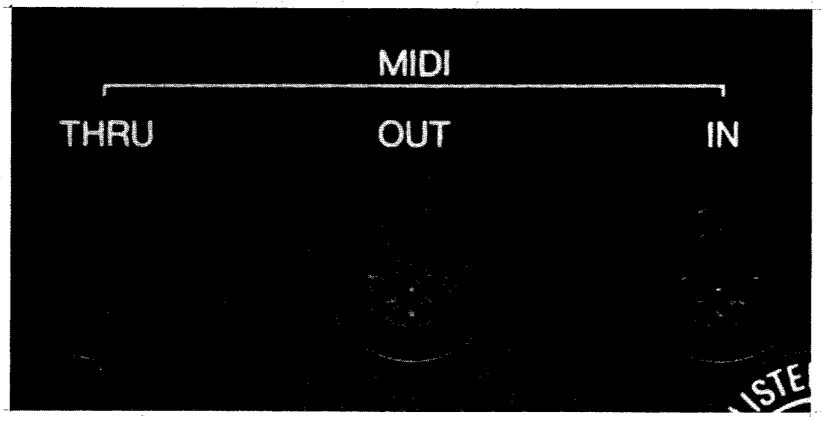


Figure 15-3

The purpose of the "MIDI IN" and "MIDI OUT" jacks is pretty much self-explanatory: this is where MIDI data enters and leaves the machine. The "MIDI THRU" jack is an *output,* out of which is instantaneously sent an exact copy of the data coming into the "MIDI IN" jack. In effect, the UART replicates the incoming MIDI signal and passes a copy of it right back out of the MIDI THRU port* while the original goes on its merry way to the receiving instrument's own microprocessor. **(see figure 15-4)**

Therefore, the data leaving the "MIDI THRU" jack does *not* contain any data generated by the host machine - but only that original data *sent* to the host machine. We'll talk more about uses for "MIDI THRU" later in this chapter.

First, let's talk a bit more about the way that MIDI data is organized, and how the interface itself is set up. The very first point which must be clearly understood is that it is digital data *only* (in the form of ones and zeroes), that travels down the MIDI cables - *no audio signal is sent.* So what goes into the MIDI cable via the "MIDI OUT" port is entirely different from what goes into the audio cables you plug into the instrument's output jacks. **(see figure 15-5)**

Secondly, it's important to understand what kind of interface MIDI is. In the computer industry, there are two different ways of transmitting and receiving data - via a *serial* interface and via a *parallel* interface. The difference between them is as follows: in a serial interface, the ones and zeroes all travel down a single wire, one after another, like soldiers in a single file. **(see figure 15-6)**

On the other hand, in a parallel interface, the stream of ones and zeroes are sent down several wires simultaneously, usually eight at a time. **(see figure 15-7)**

* "port" is computerese for any physical device - like a jack - which allows data to enter or leave a computer.

The parallel interface, then, would appear to be the interface of choice. However, MIDI is a serial interface.

Surprised? When this was announced, a lot of people were. The arguments against using a parallel interface were twofold: firstly, the parallel connectors and cables themselves were far more expensive and fragile than serial hardware; and secondly, data being transmitted via a parallel interface can only reliably travel ten feet or so before bits of information start dropping by the wayside. Serially transmitted data can travel further since it is typically sent at much greater speeds. So, in the interest of consumer cost and reliability, the serial interface was chosen for the MIDI standard.

Of course, speed is of paramount importance with MIDI since we will want events to be transmitted pretty much as they occur, in *real time*. If we are playing a Cm chord on the DX7II, for example, we want another synthesizer to which we are transmitting MIDI data to be aware of that Cm chord *as we play it*. Consequently, a lot of thought and debate went into the question of how fast the MIDI transmission rate was to be. In the end, the manufacturers settled on a rate of 31.25 kBaud. A Baud is a unit of measurement which indicates one bit of information (that is, one one or one zero) being transmitted each second. Therefore, the MIDI transmission rate of 31.25 kBaud tells us that we will be transmitting data at the incredible rate of 31,250 bits of information per second!

If that doesn't seem sufficient to you, consider this: the standard rate at which your bank teller or travel agent sends and receives data is 1200 Baud - about 1/30 the speed of MIDI. It all seems instantaneous to us mere human beings, but bear in mind that far more information is sent via MIDI than from your travel agent to an airline. In fact, there is a body of dissenters who argue that the MIDI transmission rate is too slow! It is possible that a future version of MIDI will have a considerably higher transmission rate, but in any event, this won't negate the effectiveness of current synthesizers which work at the "slow" 31.25 kBaud rate. It also should be noted that the current computer technology is far advanced from what it was in 1983, when 31.25 kBaud was considered extremely fast. Today, the Apple Macintosh computer, for example, can actually transmit data at a rate of a *million* bits per second!

So now we know how MIDI data is sent, and at what speed, but the real question is, what *is* MIDI data? Exactly what is one synthesizer telling another? The answer to this question is actually quite complex. There could, in fact, be many things that the DX7II, for example, could be telling the world. It could be passing along commands it's receiving from its own keyboard, such as "key on" and "key off" flags, which key(s) are being depressed, velocity sensitivities, etc. Or it could be passing along data it is receiving from its controllers - the current status of the pitch bend wheel, mod wheel, breath controller, etc. Or it could be telling the world which program number is currently being used, and when we call up a different program, it could divulge that information, too. Or, it could actually be sending specific information about the sound currently in its internal or cartridge memory or edit buffer - information about the operator EGs, frequency ratios, LFO settings, etc.

In fact, at any given time, we might require all of the above information. At other times, we'd need only small portions of it. For example, if you "MIDI up" a DX7II and a Prophet (by hooking up MIDI

OUT of one to MIDI IN of the other, and vice versa), and our DX7II starts telling the Prophet about its operator output levels, the Prophet will, in computer terms, shrug its digital shoulders and, in effect, say "Huh?" **(see figure 15-8)** since it doesn't have any operators. Similarly, if the Prophet starts gabbing to the DX7II about its VCF resonance setting, the DX7II will scratch its digital forehead and a blank expression will prevail **(see figure 15-9)** since it wouldn't know a VCF if it came up and bit it! In fact, when one instrument tells a dissimilar instrument information which is exclusive to that brand and model only, the receiving instrument simply ignores that data. For this reason, you cannot possibly use MIDI to, say, store Prophet sounds in your DX7II, or DX7II sounds in your Prophet.

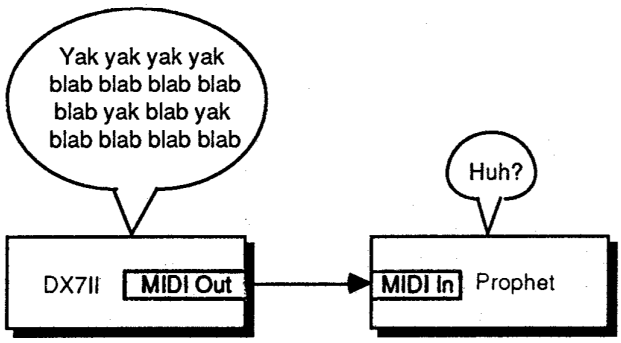


Figure 15-8

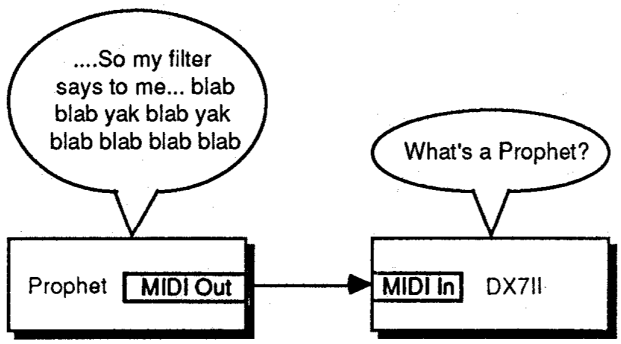


Figure 15-9

Since speed is of paramount importance here, and since MIDI data travels only one bit at a time, we really don't want to gum up the lines with lots of useless information. For this reason, MIDI data has been divided up into four major categories:

1) Channel Messages: These include: a) Common keyboard information, such as which note is being played, "key on" and "key off" flags, and velocity sensitivities. b) Controller information, including pitch bend, mod wheel, after touch, breath controller, and any foot pedals or switches that may be attached. c) Program information, specifically the *number* of the program that is currently in use - and *not* any information about the sound itself.

When a synthesizer receives MIDI *channel messages*, it will normally respond by playing the same notes at the same times. This was in fact the original concept of MIDI - it would be an interface that would allow for instantaneous "layering" of sounds - and this is still by far its most common use. Bear in mind that the receiving synthesizer does NOT have to have the same sound in order for this to occur, nor

does it have to be tuned the same as the transmitting synthesizer. This can create some unique and interesting effects. If set properly, the receiving synthesizer may also apply the same controller information - pitch bends, velocity, etc., and may even call up the same program number. It is also important to realize that the reception of MIDI channel data can in no way cause the receiving synthesizer to perform beyond its own limits. For example, sending a ten-note chord to an eight-voice polyphonic synthesizer cannot magically permit it to play any more than eight of those notes. Similarly, sending velocity sensitivities to a non-velocity-sensitive instrument will not cause it to suddenly develop that characteristic.

Channel messages sent to a sequencer or computer sequencer program will simply be stored (along with a timing "clock" signal) for later recall and playback.

2) System exclusive data: This is the information about the sound currently in use, or even about ALL the sounds in the instrument's internal or cartridge memory. By performing "data dumps" - that is, sending system-exclusive data to another synthesizer of the same model (or to a computer programmed to work with that data) - we can cause that synthesizer (or computer) to load voice and performance parameters into its own edit buffer or internal memory. Every major synthesizer manufacturer has been given a specific MIDI ID number, and many have created sub-IDs for specific models they make as well. In this way, a DX7II (which has a MIDI ID number of --) can only recognize system exclusive messages from another DX7II. For all of you hackers out there, the specific system exclusive code for the DX7II is presented in Appendix F.

3) System Real-Time Data: This includes timing information (sometimes called the *MIDI clock*), which is normally only transmitted and used by MIDI sequencers and drum machines. The MIDI clock is a standardized 24-digital-flags (or "pulses")-per-quarter-note timing signal, which obviously changes with any sequencer or drum machine tempo changes. Newer revisions of MIDI will also include something called *MIDI Time Code,* which will be an adapted version of SMPTE time code (more about this shortly).

4) System Common Data: This is a kind of catch-all "housekeeping" category which includes such minor things as an auto-tuning command for those synthesizers whose microprocessors can perform this task.

By breaking down all of the possible MIDI data into these smaller subcategories, we are able to ensure that we aren't sending or receiving any more information than we can actually use. For example, system exclusive data leaving a Kurzweil is only going to be comprehensible to another Kurzweil or to a computer which has been programmed to understand and perhaps store this information - so there's no point in sending this data to an Emulator. System real-time data is only usable by a MIDI drum machine, or by a dedicated sequencer or computer which has been programmed with sequencing capabilities - so there's no point in sending this kind of information to a DX7II.

On the other hand, channel information is usable by virtually every synthesizer ever manufactured - as long as the instrument can recognize a keyboard, it can utilize most if not all of this data. Why is this then referred to as "channel" data? The answer lies in the fact that MIDI has been structured as a multi-channel system. The MIDI protocol actually allows for up to sixteen independent sets of channel messages - each containing up to sixteen notes at a time - to be simultaneously (more or less) transmitted.

How can we send sixteen separate sets of data at once down a single wire? We can't. But at the high-speed transmission rate of 31,250 bits of information each second, we can do it fast enough that most humans in most situations will never hear the fact that it isn't completely simultaneous. Of course, in order to separate out the sixteen different sets of information, we'll have to mark each batch with identifying flags. And that's precisely the way MIDI channel information is sent. A special number - called an "address" flag - identifies a particular set of MIDI data as belonging to a particular MIDI channel. A good way to understand the concept of MIDI channels is to hold in your mind the now-familiar picture of various TV stations transmitting the 7 o'clock news, and your TV set at home being able to receive any of these stations. If you want to watch Dan Rather's sweaters at 7:00, you will have to tune your TV set to receive CBS. On the other hand, if Peter Jennings' ties are more to your liking, you'll have to tune to ABC. In no event could you hear Dan Rather by tuning to any channel other than CBS; nor could you see Peter Jennings if you're tuned to anything other than ABC. Each TV station *transmits* on a particular channel, and if you want to tune in that program, your set will have to be *receiving* that particular channel. **(see figure 15-10)**

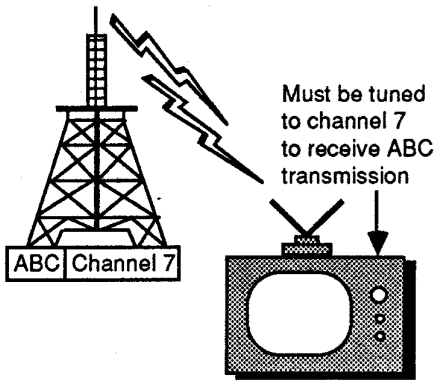


Figure 15-10

MIDI works in a very similar manner. Instead of getting the 7 o'clock news, however, our synthesizer or computer can get the latest MIDI data - but it will have to be "tuned" to the appropriate channel in order to receive the correct channel messages. **(see figure 15-11)**

How do we determine which channel our synthesizer or computer is "tuned" to? Again, we have to give a fairly complex answer to a fairly simple question: first, we must know our instrument's *reception mode.*

Originally, there were three different ways that MIDI data could be received: Omni, Poly, and Mono. *Omni* mode is a kind of "super-reception" whereby a receiving instrument accepts MIDI data coming in on *all* MIDI channels - the TV equivalent of you somehow being able to tune in CBS, ABC, and (heavens, no!) NBC, not to mention all other local stations, all at the same time (very important, if you want to see how Peter Jennings' ties match with Dan Rather's sweaters). What you'd see on your screen would be a mess, but you'd be getting all the information at once.* *Poly* mode sets up the situation whereby the receiving machine simply "listens" to only one MIDI channel at a time. And *Mono* mode was intended for multi-timbral synthesizers (those synthesizers, which, like the DX7II, can produce more than one sound at a time) whereby they could listen to several specified MIDI channels

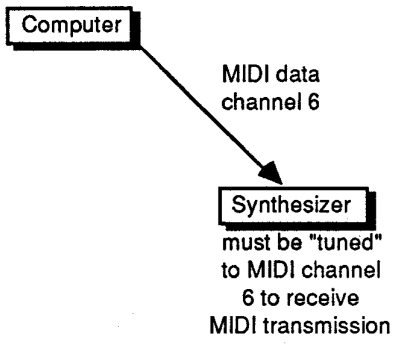


Figure 15-11

* Why would we ever want to listen to all sixteen MIDI channels at once? One reason might be for quick scannings of complicated multi-timbral sequences. Rather than having to set up ten, twelve, or sixteen different synthesizers in order to hear all the different musical parts, you could simply plug in one machine, set it to omni mode, and hear all the different parts at once. Of course, that one instrument may only be capable of giving you one timbre, but at least you'll be able to quickly hear all the parts.

at once, and then internally assign the incoming data to various different voices within the machine. The DX7II is able to receive MIDI data in any one of these three modes, as we will see shortly.

Unfortunately, so the story goes, someone at a major synthesizer manufacturer (who will remain unnamed, but the first initial is a "Y"), misinterpreted the original concept of "Mono" mode, and so today these three modes have mutated into the following *four* MIDI reception modes:

Mode 1: Omni-on Poly In this mode, the receiving machine listens and responds to all incoming MIDI channel data, and plays its own notes in response polyphonically, according to its own limits (which in the DX7II will be anywhere from one to sixteen voices, depending upon the voice mode and the key mode selected - see Chapter Fourteen for more on this).

Mode 2: Omni-on mono In this mode, the receiving machine again listens and responds to all incoming MIDI channel data, but will play only the last note received, no matter what channel it is on. Although the DX7II cannot be specifically put into this mode, use of the DX7II in its own monophonic or unison mono key mode (see Chapter Thirteen) can simulate this effect when the DX7II is listening to MIDI data input in Omni mode.

Mode 3: Omni-off poly In this, the most "normal" of all MIDI modes, the receiving instrument listens and responds to only one MIDI channel at a time, and plays its own notes in response, polyphonically (up to its internal limits - again, from one to sixteen notes in the DX7II). Remember that *each* MIDI channel can be sending up to sixteen notes simultaneously. It's a handy "coincidence", therefore, that the DX7II is itself capable of playing up to 16 notes at a time!

Mode 4: Omni-off mono In this mode, the receiving instrument again only listens and responds to one MIDI channel at a time, but will play only the last note heard on that channel, monophonically. By using the DX7II in monophonic or unison mono key mode (see Chapter Thirteen), we can simulate this when the DX7II is listening to MIDI data input in Poly mode.

Note that system information (that is, system exclusive, system real-time, and system common) is not marked with an identifying channel address flag, and so is received by all devices regardless of the receive mode they are in.

So far, we have explored why MIDI exists, how MIDI data is transmitted and received, and what kind of information is actually sent. The obvious remaining question is, what kind of things can we actually *do* with MIDI?

MIDI Applications

1) First and foremost, as mentioned earlier, is the simple hook-up of two synthesizers via a MIDI cable in order for the "slave" to receive channel messages from the "master" and to respond in kind. **(see figure 15-12)**

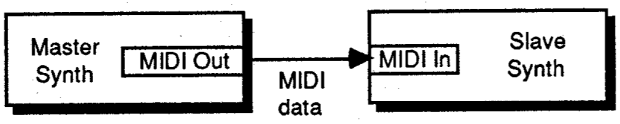


Figure 15-12

In this instance, any notes played on the keyboard of the master instrument will result in the same notes being played simultaneously on the slave, regardless of the current sound in the slave. This will allow for the generation of layered, multi-timbral effects. Although the slave will play the same keys on its own keyboard, bear in mind once again that its keyboard need not be tuned to the same notes. For example, playing A3 on a Prophet (tuned to A440) being used as a master will result in a DX7II slave playing A3 on its keyboard - but if the DX7II's keyboard has been transposed (using the Transpose parameter of edit switch 7 - see Chapter Twelve) so that C3 = G3, then the DX7II's A3 will in fact be playing an E4 (since we have transposed all notes up by an interval of a musical fifth)! **(see figure 15-13)**

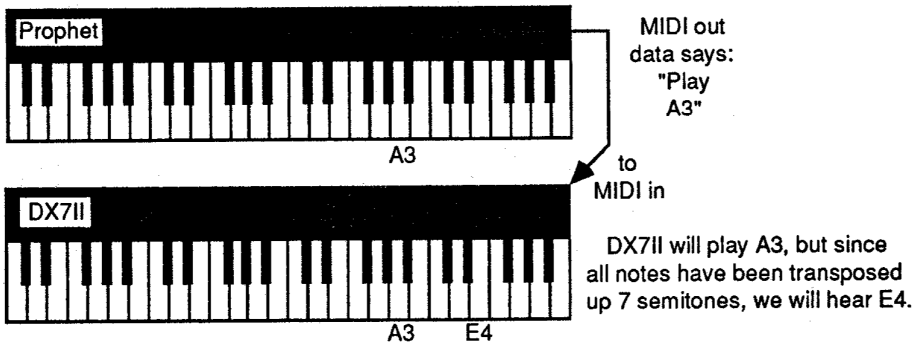


Figure 15-13

2) The use of a dedicated sequencer or computer programmed to act as a sequencer allows for the storage of channel messages. This information can then be retrieved - "played back" - at a later time, and the original performance recreated exactly, with all expressive controls as well as the original note and timing values. Additionally, the data can be edited to various degrees, allowing you to change note values, timing values, controller changes, or even program numbers! Moreover, the speed of the playback can be adjusted without affecting the pitch (unlike the simple act of speeding up or slowing down a tape playback, where pitch changes as a function of speed), or the pitch can be transposed by some fixed interval without altering the speed. Because MIDI is a multi-channel system, one central computer sequencer can output up to sixteen different sets of data at once! Each channel can be playing up to sixteen-note chords simultaneously, and, of course, any number of synthesizers can be listening and responding at any given time to any (in Poly mode) or all (in Omni mode) of the sixteen channels. Instruments receiving in Mono mode will listen to more than one specified MIDI channels. This allows for complete orchestrations **(see figure 15-14)** and also to a large degree eliminates the need for multi-track tape recorders, since each receiving synthesizer can have its audio output processed independently. **(see figure 15-15)**

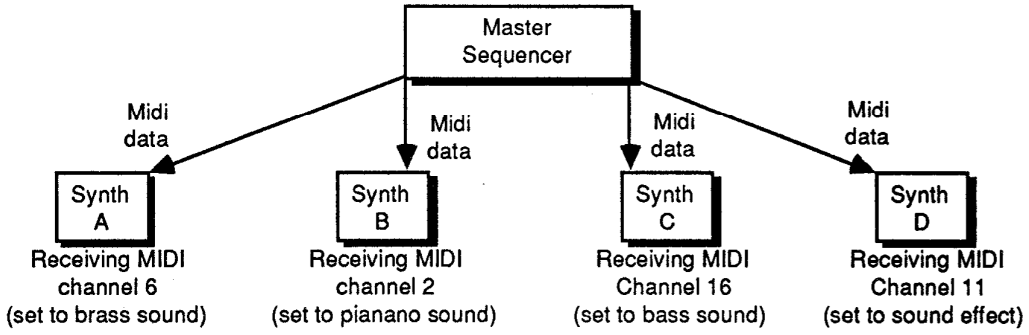


Figure 15-14

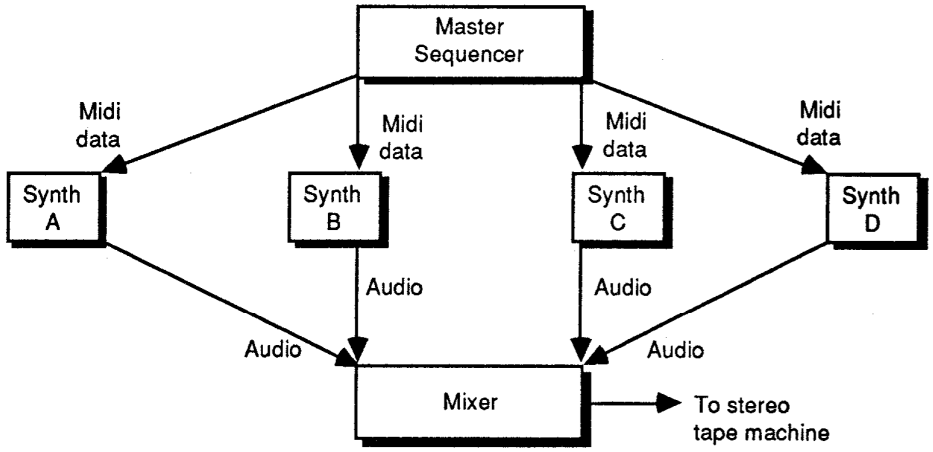


Figure 15-15

In this manner, if you physically have enough synthesizers in the room at once (and the new concept of *modular* synthesizer systems is making this more of an affordable reality), there is really no need to ever do multi-tracking on tape. Just start the sequencer playing back, get all the different machines sounding the way you want, and send their audio signals straight to your stereo mixdown machine! Synchronization of all the different parts is accomplished by the MIDI clock. Because the MIDI clock flags are tied to the overall tempo, MIDI sequencers can easily adjust to any tempo changes within a song. Moreover, they can readily be synchronized to other standard sync tones, such as FSK (Frequency Shift Keying - an analog pulse tone), or SMPTE time code (used extensively in video applications, as a means of numbering individual video frames). As mentioned above, the MIDI protocol will also include the implementation of something called *MIDI time code,* which essentially incorporates a form of SMPTE into the MIDI data stream.

Of course, the fact that MIDI is a serial interface means that the more data you send down that one wire, the longer will be the time delays between an event being transmitted and it being received. In point of fact, MIDI is fast enough to permit more than 65 16-note chords each second! But delays - more often than not engendered by the transmitting and receiving instrument's own internal microprocessors - do build up. If these delays exceed a thousandth of a second or so, they can begin to become perceptible to us mere mortals. One way of minimizing these delays, however, is to use the MIDI THRU port. The data leaving this port, remember, is exactly the same data received by the MIDI IN port - but it doesn't have to go through the receiving

instrument's microprocessor first. By setting up a MIDI "chain" like this, the original data sent out by the master sequencer is simply passed on to all the other receiving instruments in bucket-brigade fashion without each synth in the chain having to wait for the preceding one to have scanned and processed that data. **(see figure 15-16)**

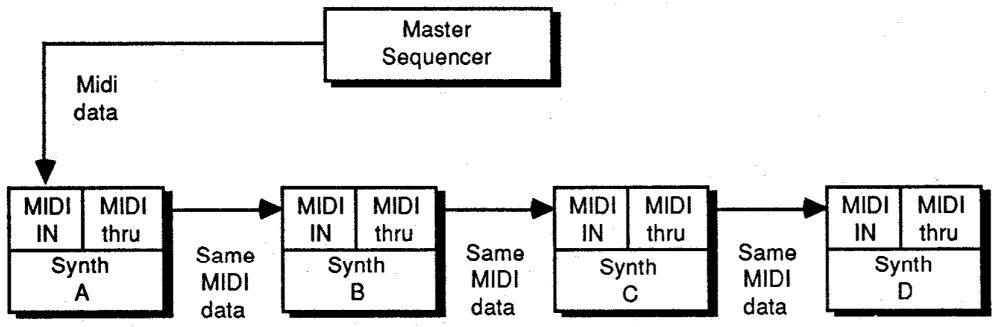


Figure 15-16

An even better method of distributing the data is to use a device called a *MIDI THRU box* - which is pretty much like an analog signal splitter. A typical MIDI THRU box will contain one MIDI Input and four or more MIDI THRU ports. A good way of setting up a multi-instrument MIDI sequencing system would be to use a MIDI thru box like **figure 15-17.**

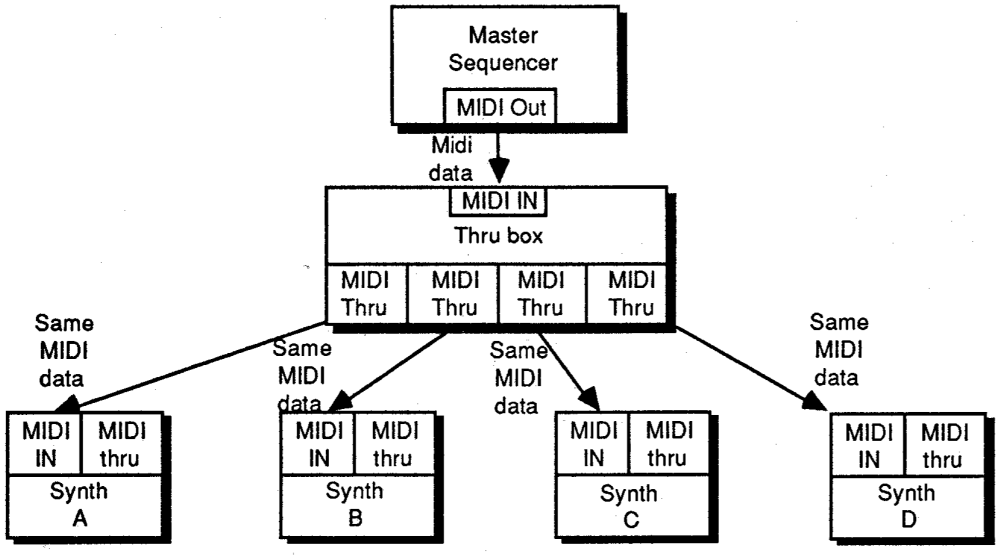


Figure 15-17

As if all this wasn't enough, some of the newer generation of computer sequencing programs, combined with the newer high-speed, graphics-oriented personal computers (such as the Macintosh and Amiga), even allow you to generate note transcriptions of the sequencer data! This allows musicians who normally work with "dots on a line" instant access to work which need only be performed correctly once - and even then quite possibly in sections - by someone hooked up via MIDI into a computer.

3) Many synthesizers, like the DX7II, exhibit only rudimentary data displays. Two lines of forty characters each on a small LCD does not exactly allow for scintillating conversations with your instrument. However, sending system exclusive data via MIDI to a computer hooked into a video display (like your TV) will allow for the

simultaneous display of a great deal of data at once. Many commercial MIDI *patch editor* programs even allow graphic display of this data. These programs will typically do things like draw envelopes, diagrams of algorithms, and even the various keyboard level scalings for each sound in your DX7II. Moreover, you can make changes to this data directly from the computer (or from the DX7II's own data entry section) and watch these diagrams change in real time! This is an exciting educational concept which really helps to explain the inner workings of a synthesizer, and also acts as a real time-saver when editing voices.

4) Once you've got the system exclusive data into your computer, it's a simple matter for the computer to store that data via its own internal data storage routines. This typically involves a disk drive of some type, and floppy disks are extremely inexpensive, especially when compared with something like the DX7II RAM4 cartridge, at over $100 a shot! Using such "off-line" data storage allows you to inexpensively build huge libraries of sounds for your DX7II - just as the disk drive of the "FD" model allows you to do this in the instrument itself. As the need presents itself, you simply download voices into either your DX7II internal memory or a RAM4 cartridge (just as you can from the FD's disk drive), and you can always have your most-needed voices at your fingertips. Most patch editor programs, and even several sequencing programs, allow for this "librarian" function.

5) It's not only synthesizers and computers that can use MIDI data. Many other devices exist which contain microprocessors, and, as we've seen, virtually anything that has a microprocessor can theoretically "speak" MIDI. Digital drum machines capable of sending out MIDI system real-time data (in the form of the "MIDI clock") and/or MIDI channel messages ("Hey, I'm playing the bass drum on 1 and 3, and the snare on 2 and 4, guys!") can be used as MIDI controllers or sequencers. Most digital audio signal processing devices, such as digital delay lines and digital reverb units have the potential for being "MIDI'd" to synthesizers. Several of these units (like the Lexicon PCM70 and the Yamaha SPX90) allow the synthesist to directly control variables such as delay time, feedback, output filtering, etc, as well as offering the ability to change pre-programmed settings in the outboard device's own memory. Performance accessories such as lighting rigs can be microprocessor-controlled and issued instructions via MIDI commands, directly from the synthesizer or from a master computer. If no one has done it yet, you should soon be able to turn your microprocessor-controlled dishwasher and microwave on and off from your synthesizer keyboard! (Why? Why not??) **(see figure 15-18)**

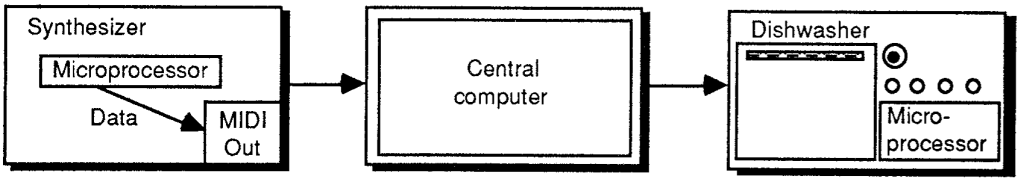


Figure 15-18

6) Last, but by no means least, MIDI data manipulations can be used as a compositional tool. Got writer's block? Need inspiration? Send in some MIDI data to your computer and use a compositional program (like Dr. T's "Algorithmic Composer" or Intelligent Music's "M") to "roll the dice" for you. Computers are very good at randomly pulling out data from a table of numbers that you enter in. These types of MIDI programs utilize these random functions in order to allow the

computer to actually do composing for you - and generally in a semi-intelligent fashion! This MIDI application is possibly the most esoteric, yet it is an exciting one that is currently being explored by many software manufacturers.

The really exciting thing about MIDI is that it is software-based and therefore open-ended. This means that the potential is literally limitless. Just as we could theoretically use the English language (or Swahili, or whatever) to communicate any idea to anyone who understands it, so too can we use MIDI to theoretically send controlling information to any device of any kind - by using that device's own microprocessor, or (if it doesn't have one) by simply inserting one before it. Because virtually any analog signal can be converted back and forth to digital code (via our old friends, the ADC and DAC), there is virtually no limit as to what can be accomplished with MIDI.

If I sound excited, you read me correctly. Yes, MIDI has caused nearly as many problems as it has solved. But these problems are mostly those of misunderstanding and misinterpreting what the MIDI protocol allows us to do. It *has* forced musicians everywhere to perhaps learn more about computers than they ever wanted to. But that in itself certainly isn't a bad thing, given the increasing influence computers are having on our everyday lives. In summary, there probably isn't a creative musician anywhere who doesn't feel that the advantages presented by MIDI far outweigh any potential disadvantages. It's a powerful creative tool - learn how to use it!

For further, and much greater detailed information on MIDI, the reader is referred to Craig Anderton's excellent book, "MIDI For Musicians" (Amsco Publictions), as well as to the many fine publications and articles on the subject put out by KEYBOARD magazine. You can also contact the International MIDI Association at 11857 Hartsook St., North Hollywood, CA 91607.

Now let's take an in-depth look at how MIDI is implemented in the DX7II:

The DX7II MIDI controls (edit switches 31 and 32)

Unlike its predecessor (the DX7), MIDI is implemented here in a sophisticated fashion, and the DX7II is in fact quite suitable for use as a master keyboard controller. This is because such features as *local on-off* and programmable program change transmissions are offered. In addition, the DX7II keyboard transmits the full range of 127 velocity increments (even though it internally only makes use of 99), instead of the 99 transmitted by the DX7. Last, but not least, the keyboard of the DX7II has an improved "feel", relative to the DX7 keyboard.

Virtually all of the DX7II MIDI functions are accessed from edit switches 31 and 32. Let's begin by taking a look at the four LCD displays offered by edit switch 31. **(see figure 15-19(a) through (d))**

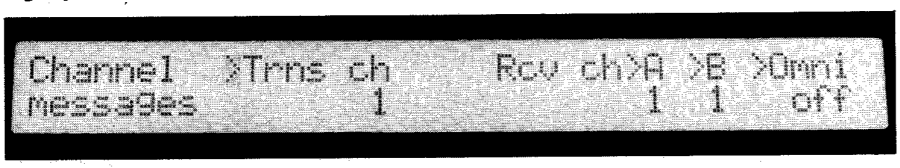


Figure 15-19(a)

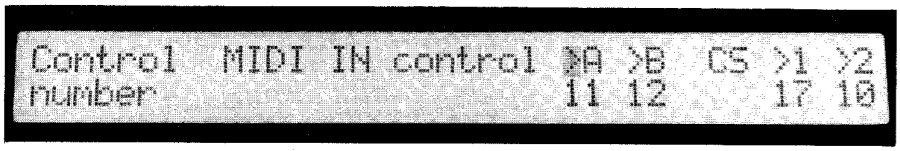


Figure 15-19(b)

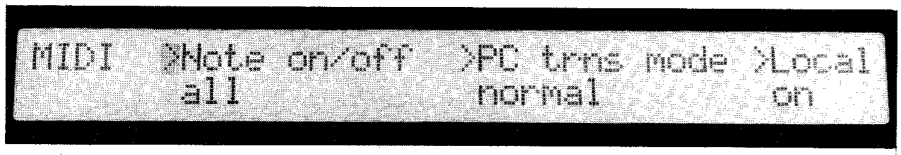


Figure 15-19(c)

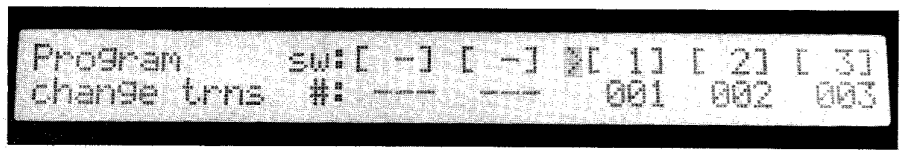


Figure 15-19 (d)

As usual, you cycle through these different displays by simply pressing edit switch 31 repeatedly. All of the parameters offered by edit switch 31 relate to the transmission and reception of MIDI channel messages only. The first of these displays (**figure 15-19(a)**) allows you to set the transmit and receive channels, and also to set the DX7II receive mode. Even though this instrument can operate internally in a split mode, it can only transmit over the whole keyboard via one MIDI channel (unlike many dedicated MIDI keyboard controllers that allow you to transmit over two or more channels, with programmable split points built in). This means that if you want to use the DX7II keyboard to control multiple synthesizers, they will have to all be set to listen to the same MIDI channel and you'll have to set low-note high-note boundaries for each. Most of the Yamaha modular synthesizers (i.e. the TX816, TX81Z, and FB-01) are capable of this kind of setting. In any event, the range of the transmit parameter ("Trns ch") is quite obviously 1 - 16. The value is changed like all other edit parameters - by using the data entry section. Additionally, you can choose a value of "off", where the DX7II will do no MIDI transmissions via its MIDI OUT jack at all. This will save you the hassle of having to unplug the MIDI OUT cable manually if you ever want to cease transmissions.

Let's skip ahead momentarily to the parameter on the far right of the display, called "Omni". This is a simple on-off control, allowing you to determine if the DX7II will listen to ALL MIDI data (Omni ON) or SOME MIDI data (Omni OFF). If Omni is OFF, then the last two controls ("Rcv ch") have some meaning. If it is ON, then both voices in the DX7II will respond promiscuously to any and all data coming in through the MIDI IN jack.

The "Rcv ch" parameter allows you to set the receive channels independently for voice A and voice B - or to turn MIDI reception "off" for either or both voices. The range here is the same as that of the transmit channel parameter: 1 to 16. If you are using an external controller (a keyboard or a sequencer, for example) to "play" the DX7II remotely, you will find that this offers you some interesting possibilities. First of all, if the DX7II is in Single voice mode (or if it's set to a performance memory made with a single voice), then, obviously, altering the receive channel for voice B will have no meaning.

If the DX7II is set to Dual voice mode (or if it's set to a performance memory made with dual voices), then you will find that the voice B setting still has no meaning - the DX7II "listens" through voice A only. Therefore, whatever MIDI channel you set voice A to receive on will be the channel that *both* voices respond to. You cannot use the two voices separately from a remote controller in this situation.

In Split mode (or when using a performance memory created with split voices), however, new and interesting things start to happen. If both voices are set to the *same* MIDI channel, the remote controller will act just like the DX7II's own keyboard - voice A will appear below the split point, and voice B above it. However, if the two voices are set to receive on *different* MIDI channels, then you will be able to access each of the two voices through the full C-2 to G8 MIDI note range! In other words, a sequencer or remote keyboard controller will be able to play voice A *above* as well as below the set split point, and will similarly be able to play voice B *below* as well as above that point. With this setup, the DX7II will respond in a manner similar to most other multitimbral synthesizer modules - each voice is available through the complete MIDI note range.

The next display (the Control number display, as shown in **figure 15-19(b)**) allows you to link external MIDI controllers (like those from a remote keyboard, for example) to either the MIDI IN control shown in edit switch 26 (as discussed briefly in Chapter Ten), or to the whatever functions you have assigned the two continuous sliders (as discussed in Chapter Thirteen). This same display also allows the continuous sliders to *transmit* controller change data via MIDI, so that they can alter parameters in other synthesizers. Let's start with the MIDI IN control. Press edit switch 26 repeatedly if necessary in order to call up its display. **(see figure 15-20)**

Figure 15-20

(As you can see, the external MIDI controller can route LFO signal for pitch modulation or amplitude modulation; it can route EG bias modulation signal; or it can act as an overall volume control. If you are currently viewing voice A (if the "A/B" status light is off), the number in parentheses will be "11". If you are currently viewing voice B (status light on), the number will be "12". This number is the *MIDI controller number*. External keyboards will often have a series of sliders, wheels, and footswitches (if they are Yamaha keyboards, they will often have breath controllers, as well) that can be utilized to remotely change some facet of the sound you are hearing. The only way that a receiving synthesizer can know just which one of these real-time controllers has had its position changed is by numbering each of them. Some of them may have fixed numbers only, while other MIDI keyboard controllers will let you specify the numbering. In any event, we'll need to tell each DX7II voice just which controller number to listen to in order to make the changes assigned to the MIDI IN control. The initialization default for the MIDI IN control of voice A is MIDI controller #11, while the default for voice B is #12. Any signal received from external controller #11 will perform whatever function you specify for the MIDI IN

controller with edit switch 26 for voice A, and any signal coming in from external controller #12 will perform the programmed MIDI IN function for voice B.

The purpose of the "MIDI IN control" parameter of edit switch 31 is to allow you to change this number. Try it: put your DX7II into single voice play mode and press edit switch 31 repeatedly if necessary in order to call up the Control Number display (as shown in figure 15-19(b)). Position the cursor over the MIDI IN control parameter for voice A and move the data entry slider. The range here is 11 - 31, reflecting most of the established controller numbers, as determined by the MIDI spec 1.0. Alter this to the maximum value of 31, and then press edit switch 26 repeatedly if necessary in order to call up the MIDI IN display. The "(11)" has changed to a "(31)"! You can try the same thing with voice B (you'll have to, naturally, change over to Dual or Split mode in order to do so). All you are doing here is *mapping* an external controller to an internal function. The important thing is that the DX7II makes provision for external controllers taking over some real-time functions from the onboard real-time controllers.

In a similar manner, the "CS" parameters in edit switch 31 (again, see **figure 15-19(b)**) allow you to link any external MIDI controller to the DX7II's onboard continuous sliders - or to use the sliders to make changes to parameters in external synthesizers. In the first instance, whatever external controller you have assigned will take on whatever function you programmed for the continuous slider in question (there are separate links for CS1 and CS2) with edit switch 27. As with the MIDI in function, changing the value of the external controller number in edit switch 31 will cause a change that is reflected in edit switch 27. The range here is 5 - 31, meaning that more different external controllers can take on CS functions than MIDI IN functions. If you are planning on using external controllers to take on the CS functions, you can only use controller numbers 9 - 31, however. Let's try it: press edit switch 31 repeatedly if necessary in order to call up the Control number display, as shown in figure 15-19(b)). Note that the default value for CS1 is "8" (the CS2 external controller default is "10"). Now press edit switch 27 repeatedly if necessary in order to call up the CS1 display. **(see figure 15-21)**

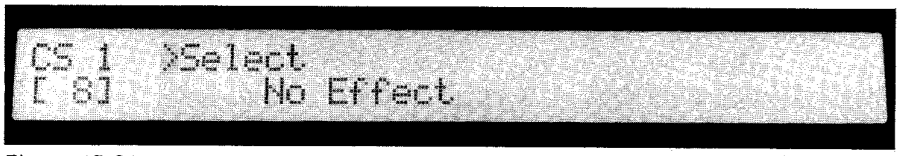


Figure 15-21

Note that the number in parentheses in the lower-left hand corner also reads "8", reflecting the default value we just saw in the Control number display. Now press edit switch 31 again in order to return to this display. Position the cursor over the "CS1" parameter in this display, and use the data entry slider to change this through its full range of 5 - 31, and enter in the minimum value of 5. Press edit switch 27 again. Note that the "8" has now changed to a "5", reflecting the change we just made. At this point, if you have an external controller which is labeled controller number 5 sending MIDI data IN to the DX7II, changes made to it will result in whatever kind of parameter change you assign to CS1 with this display. You can alter the external controller number assigned to CS2 in the same way.

As mentioned above, there is a two-way street available for the CS functions. Not only can an external controller be mapped to it, but the CS can also transmit controller change data via the MIDI OUT jack. In this use, changing the controller number here will tell the receiving synthesizer on the other end of the MIDI cable just which controller number the CS is assuming. Here, you can use the full range of controller numbers from 5 to 31 (you aren't limited to 9 - 31, as you are when the continuous slider functions are recipients of external controller data only).

Pressing edit switch 31 yet again will yield the generic "MIDI" display, as shown in **figure 15-19(c)**. The first parameter here, called "Note on/off", allows you to specify whether the DX7II will respond to all "key on" and "key off" commands it receives (if the value is "all"), to odd-numbered notes only (if set to "odd"), or to even-numbered notes only (if set to "even"). If it were set to "even", for example, the DX7II would respond to C3 and D3 played from an external keyboard, but would not respond to B2 or C#3. If set to "odd", it would respond to B2 and C#3 but not C3 and D3. The "odd" and "even" settings can allow you to generate some interesting stereo effects - such as if the DX7II is set to "odd", for example, and another receiving synth were set to "even". This feature affects MIDI reception only; the DX7II keyboard always transmits ALL "key on" and "key off" flags.

The "Local" parameter on the far right-hand side of the display is your Local on/off control. When "on", the DX7II keyboard not only transmits MIDI data, but it also sends data to its own microprocessor, causing sounds to be created and signal outputted from the DX7II audio jacks. When local control is "off", however, the DX7II keyboard is no longer sending data to the internal microprocessor; it is acting *solely* as a MIDI transmitter. The DX7II microprocessor will now respond only to external data received from the MIDI IN port. What advantage does this pose? Well, it means that you can use the DX7II keyboard as a master controller, sending MIDI channel information to remote slave synthesizers, without actually hearing the DX7II voices. In the meantime, DX7II voices can be responding to incoming MIDI data from a sequencer or even a *different* MIDI keyboard. Remember also that most MIDI delays are not a function of the MIDI transmission rate, but are actually incurred by the various instruments' own internal microprocessors. If you are using the DX7II as a master keyboard controller for many synths with local control "on", you might therefore find that the DX7II sounds are generated a fraction of a second sooner or later than the other synths, thus incurring a MIDI delay. By disconnecting the keyboard-to-internal-microprocessor routing (in other words, turning local control "off"), you at least eliminate one potential problem. Of course, DX7II sounds can still be used in such a setup - but, just as if you were addressing them from a dedicated keyboard controller that does *not* generate any sound of its own, they will appear - along with the sounds of any other receiving synthesizers - with slight delays which will at least be more or less consistent. One word of caution - be very careful with this control, since turning it OFF will have the effect of seemingly making your DX7II dead as a doornail. Before you go racing off to an Authorized Service Center, do yourself a favor and just check that Local Control has not inadvertantly been turned OFF.

Last, but by no means least, is the "PC trns mode" parameter, in the center of your display. "PC" stands for Program Change. As discussed

earlier, the MIDI spec provides for the transmission of program numbers, so that, for example, when you call up program #14 on your master synth, the slave synth responds by calling up *its* program #14 (which, remember, does *not* have to be the same sound, or even in the same tuning!). This parameter allows you to specify whether you want this to occur or not. If you don't, simply set it to "off". If you do, however, you have two options at your disposal. Either you can have things occur in the "normal" manner described above - calling up voice #14 will cause the receiving synth to call up its voice #14 - or you can set things up so that calling up one particular voice number on the DX7II causes the receiving synth to call up a *different* program number! In the former case, you would enter in the "normal" value here. In the latter, you would enter in the "programmable" value, and then press edit switch 31 yet again in order to call up the "Program change trns" display, as shown in **figure 15-19(d)** above.

This display is somewhat similar to the Fractional Scaling and Micro Tuning Edit displays in that there is a central active window. The top line shows the program number of the DX7II, while the bottom line shows the program number that will be transmitted via MIDI when that program is called up. On either side of the active window are the two nearest neighbors, allowing you to potentially see up to five such values, although, as usual, you can only ever change the one in the center active window. At the moment, there are simply hyphens ("---") in the two values to the left. This is simply because there are no program numbers lower than 1, which is in the active center window. You cycle through to higher numbers by using the right cursor switch, and to lower ones with the left cursor switch. You change the value in the lower line of the active window with the data entry section. Acceptable values here are 001 through 128, since the MIDI spec provides for up to 128 different program numbers. If you therefore enter in the number "12" here, then the receiving synth will call up its program #12 whenever you call up voice or performance memory #1 in the DX7II. What happens if you enter in a value that is higher than the receiving instrument's highest program number? Normally, the receiving synth will deal with this by simply multiplying its highest program number by a factor of 8, 4, 2, or 1 (since most synthesizers have either 16, 32, 64, or 128 onboard memory slots), and then calling up the program number corresponding to the left-over value. If you, for example, set things up here so that whenever you call up DX7II program #3, program #122 is transmitted to a receiving DX7, the following calculation will occur: the DX7's microprocessor (knowing full well that the instrument only holds 32 voices) will multiply 32 by 4 (since 32 x 4 = 128). It will then subtract 122 from 128, leaving 6 as the remainder, and finally subtract 6 from 32 (since it only has 32 voices). The final result? The DX7 will call up voice #26. It may not be pretty, but it works!

The potential use of this programmable program transmit function is that it allows you to call up whatever voice you want from any external synth without having to worry about arranging the internal memory of either machine to suit. Without such a control, if you wanted your voice #12 to call up another synth's voice #27, you'd have to either copy your voice #12 into slot #27, or shift the either synth's voice #27 into its slot #12. This can be a time-consuming and tedious process - so be glad the DX7II offers this control!

You can also quickly and easily send out a program change command from the DX7II while in any play mode. This function is, not surprisingly, called *quick program change* by Yamaha. To do this,

simply press and hold down the mode select switch corresponding to the play mode you are currently in (so that if you are, for example, in Dual voice play mode, you would press and hold down the "Dual" switch). The display will read: "Sending program change No. ---". While keeping the mode select switch held down, use your other hand to type in three digits from the numeric characters shown in brown below edit switches 1 through 10. These numbers will be reflected in the display. Once you type in the third number, the program change number will automatically be transmitted out of the MIDI OUT jack.

The settings of all the parameters offered by edit switch 31, along with the Master tuning specified by you with edit switch 14 and the cartridge Bank number set with edit switch 15, are automatically remembered by the DX7II in a special area of memory called the *System Setup memory*, and is saved along with voice and performance data whenever you store bulk data to either internal, cartridge, or disk memory (for more detail on this, see Chapter Eight). When loading bulk data (again, as explained in Chapter Eight) to either internal, cartridge, or disk memory, you have the option of loading in this System Setup data, or of retaining the System Setup currently stored (the LCD will ask you, "Load without System?" and you answer either "yes" or "no"). In this way, you can load in new voices and performance memories, complete with MIDI setup, or you can load in the voice and performance data without disturbing your current MIDI setup.

Edit switch 32 offers another five displays, all related to the transmission and reception of MIDI system exclusive data. They are accessed as usual by simply pressing the switch repeatedly. **(see figure 15-22(a) through (e))**

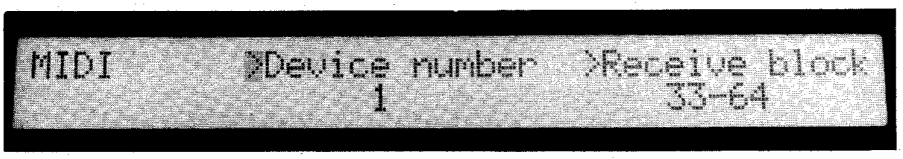


Figure 15-22(a)

Figure 15-22(b)

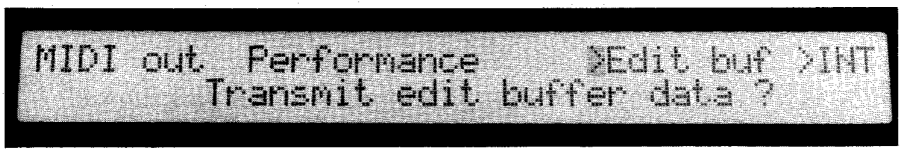


Figure 15-22(c)

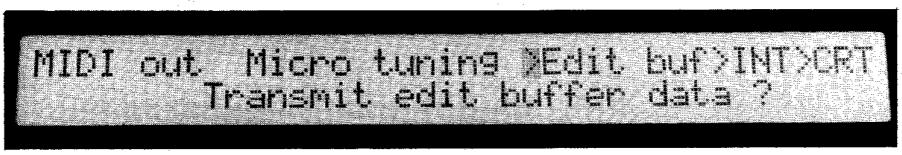


Figure 15-22(d)

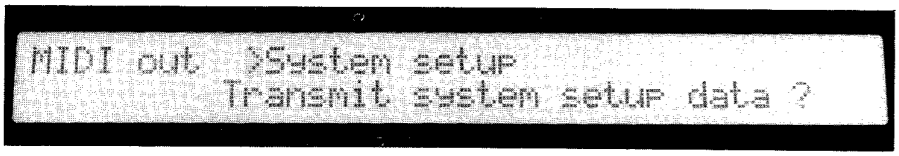


Figure 15-22(e)

The first of these (as shown in **figure 15-22(a)**) is another generic "MIDI" display, offering two parameters. The first parameter, called "Device number", is used to specify a Yamaha ID number for the transmission and reception of system exclusive data. Every Yamaha product has such a number, and in this manner the DX7II can be "told" which Yamaha instrument it is listening or speaking to via MIDI. The range of this control is 1 - 16, with "1" set as the default value, since both the DX7II and the original DX7 have this ID number. This will really only need to be changed when sending or receiving bulk data from an instrument other than a DX7 or DX7II, or when sending or transmitting such data from the "FD" disk drive in its MDR mode (to be discussed below).

The "receive block" parameter allows you to specify whether bulk voice data being received via MIDI (and, remember, this can only be DX7II or DX7 bulk data) should be stored in voice slots 1 through 32 or 33 through 64. This is useful when transferring voices from a DX7 into a DX7II (which can be done, since the DX7II is *upwardly compatible* with the DX7), since it allows you to put the incoming voices into either slots 1-32 or slots 33-64. In this way, you can actually store two banks of DX7 voices in your one DX7II. Appendix C will explain the slight incompatibilities between the two instruments - but, for the most part, DX7 voices can be successfully used in the DX7II, allowing for the higher fidelity of this more recent instrument to be applied, and also allowing for DX7 voices to be organized into DX7II performance memories.

Pressing edit switch 32 a second time will yield the MIDI out Voice display, as shown in **figure 15-22(b)**. This allows you to transmit the system exclusive data currently in your instrument's edit buffer (if you position the cursor over the "Edit buf" parameter and press the "yes" button), or to transmit the system exclusive data for the first thirty-two voices (by positioning the cursor over the "1-32" parameter and pressing "yes") or the second thirty-two voices (by positioning the cursor over the "33-64" parameter and pressing "yes"). In both instances, the display will ask the usual "Are you sure?" question and will then show "Completed!" when the process is complete. As was explained previously, such data will be totally ignored by anything other than another DX7II or a computer "trained" to use this data by means of a DX7II patch librarian or patch editor software program.

Pressing edit switch 32 again will call up the MIDI out Performance display, as shown in **figure 15-22(c)**. This allows you to either transmit the system exclusive data currently residing in your performance edit buffer (by positioning the cursor over the "Edit buf" parameter and pressing "yes") or the system exclusive data for all thirty-two performance memories currently stored in your internal memory (by positioning the cursor over the "INT" parameter and pressing "yes"). As with the last display, you will get an "Are you sure?" question and a "Completed!" confirmation when the process is done. Again, this data will be ignored by anything other than another DX7II or a computer specifically programmed to use DX7II system exclusive information (as with a patch librarian or editor program).

The MIDI out Micro tuning display (as shown in **figure 15-22(d)**) allows you to transmit micro tuning system exclusive data from either your edit buffer, internal memory, or cartridge memory via MIDI. This operates just like the last two displays, and sends out information that can only be used by another DX7II or by a computer programmed with the proper software.

Finally, the MIDI out System setup display(as shown in **figure 15-22(e)**) allows you to transmit the System Setup memory, as explained above, in the form of MIDI system exclusive data.

With the parameters offered by edit switches 31 and 32, then, you can specify a wide variety of controls that allow the DX7II to communicate with the outside world via MIDI in many ways. The DX7II, developed in 1986, offers far more of these MIDI controls than did the DX7, developed in the years before 1983 - making it a much more suitable instrument for use in a complex MIDI setup.

The last topic of discussion here will be the use of the DX7IIFD disk drive for generic MIDI data storage. This is its "MDR" (for "MIDI Data Recorder") function, alluded to briefly earlier in this chapter, and in Chapter Eight. The purpose of this mode is to allow for MIDI data of any type - channel or system - to be stored in the onboard disk drive. Thus, the DX7IIFD can be used as a central storage area for all data needed in a complex MIDI setup. This will allow you, for example, to remotely load data to external sequencers, drum machines, and other synthesizers directly from the front panel of your DX7IIFD - a wonderful time saver in performance! In order to do so, you will first of all have to have the external instrument (or instruments) hooked up correctly to the DX7II's MIDI ports. **(see figure 15-23)**

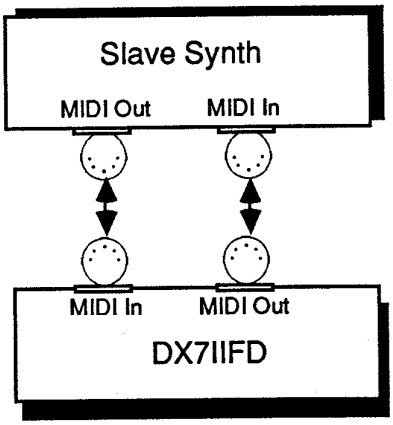


Figure 15-23

Now press edit switch 16 repeatedly if necessary in order to call up the Disk MDR display. **(see figure 15-24)**

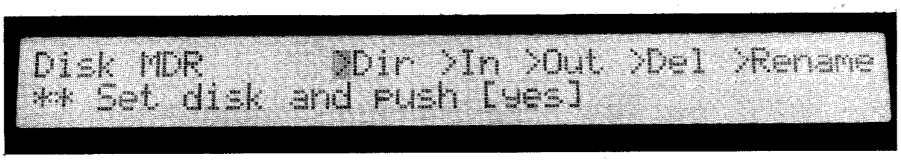


Figure 15-24

As with the Disk INT or CRT files (see Chapter Eight), you can get a directory of such generic MDR files, or you can delete ("Del") or rename ("Rename") such files. Additionally, you can specify that you want to receive MIDI data and store it into such a file ("In") or that you want to transmit such a file out to a receiving instrument ("Out"). There is literally no restriction as to what kinds of instruments, drum machines, or sequencers, you can use the MDR function with - as long as the instrument is MIDI-capable and can transmit or receive the data you want to store. The only restriction, in fact, is that the MDR file can only be up to 20 kilobytes (20k) long. This will preclude the use of the MDR function to store long, data-intensive files such as sampler files. In general, though, 20k will be sufficient for many sequencer and drum machine files, and for bulk data transfers from most synthesizers. Remember, of course, that any MDR data you transmit can *only* be understood by the type of instrument it came from. Don't expect that a QX sequencer will be able to understand an RX drum machine MDR file, or that an Oberheim will be able to understand the voice data from a Roland MDR file. MIDI is somewhat standardized - but system exclusive data certainly isn't!

Let's talk about how we would use the MDR function to specifically store and transmit another instrument's bulk voice data, since that is the way that many of you will initially need to use this function. Position the cursor over the "Dir" function to find a null file (one with "....." for a name). If you've never used this function before, you will only be shown file 1, which will automatically be a null file. In order to store an external instrument's system exclusive data on the disk, we'll have to bring the data in, so position the cursor over the "In" parameter. You'll be asked to "input filename?", and you then hold down the character key and enter a filename of up to eight characters. As with the INT and CRT files, the DX7II is very fussy here - no spaces or periods are allowed. You can, however, mix upper- and lower-case letters, as with any other naming process.

After you've come up with the name you want, press the "yes" button. If your filename is acceptable, you'll get an "Are you sure?" request. If the filename is not, you'll get a "** bad filename!" reprimand, and you'll have to try it again. Once you get to the "Are you sure?" stage, simply press "yes" if you are. The display will read "** Now waiting MIDI bulk data!" At this point, you have to go to the external synth and do whatever you need to do to get it to transmit its system exclusive data - the procedure will be found in its owner's manual. If you run into problems, you can abort this entire procedure at any time by simply pressing the "no" button. However, if everything goes smoothly, and your external synth starts sending MIDI data, the DX7II display will briefly read "Busy - now executing!" as it assimilates this incoming information. It will then immediately return to the "** Now waiting display", giving you the opportunity to append new data into the file. When you have sent as much as you want to send into the DX7II, simply press the "no" button. You'll get the "Busy - now executing!" display for a little bit longer this time, and you'll hear the disk drive start to buzz and click as the data is actually written to disk. A few seconds later, the display will happily report, "Completed!", and your data will safely be stored. You'll find also that the cursor is automatically and thoughtfully moved to the Dir parameter at the conclusion of this procedure, getting you ready to receive another MDR file from another device in your MIDI setup.

To transmit MDR data, simply make sure the receiving synth is prepared to receive and assimilate the data you're about to send it (instructions as to how to prepare a synth for externally receiving voice data via MIDI will be found in that instrument's owner's manual). Then select the MDR file you wish to transmit, and position the cursor over the "Out" parameter. The display will ask you, "transmit?". If you're all set at the other end, simply press the "yes" button. You'll have to confirm the following "Are you sure?" question with another "yes", and then you'll see the reassuring "Busy - now executing!" display, as the drive spins and the data is read off disk and transmitted through the Out port. When done, the DX7II will read "Completed!" and your receiving synth may show some kind of "MIDI received" display. Following the "Out" procedure, the DX7II will automatically position the cursor over the innocuous Dir parameter, as it did at the conclusion of the "In" procedure. This allows you to immediately prepare to transmit another MDR file to yet another device in your MIDI setup.

Be extra careful when using the MDR Out function, as bulk data receptions usually have the effect of completely erasing all data in the receiving synthesizer's internal memory. The use of MDR functions will in no way affect anything about the DX7II's memory, however.

The next chapter will show you how to put all of this information into a coherent whole, as we wrap things up and describe some advanced programming techniques.

Chapter Sixteen ◇ Advanced Programming Techniques

Now that we have discussed in detail all of the various controls and parameters available to us in the DX7II, the time of reckoning is at hand. How can we actually make use of all this information in order to create the sounds we want to get from our instrument?

Working randomly with a DX7II may sometimes yield interesting results (as the proverbial billion monkeys in front of a billion typewriters may eventually yield "Hamlet"), but more usually will result in a waste of time. This book has attempted to clarify the inner workings of digital FM in order to point you logically in the direction you need to go in order to create any particular sound. Let's review some of those directions, as applied to voice and performance edit parameters.

Selecting the best algorithm

First of all, you should keep in mind that the decision you make is *not* irreversible and that you can change your algorithm at any time, without losing any of the data you have previously entered. Of course, it's entirely possible that a carrier you were working with is now a modulator, or vice versa, but nonetheless, the changes you made remain. If you are in doubt as to what you had been doing, *compare mode* will instantly recall your past work, *if it has been stored somewhere in memory*. Remember that compare mode can only compare back to the original sound stored in memory, whether internal or RAM4 cartridge.

In terms of selecting the best algorithm in a particular context, you obviously need to have some idea about how you want to approach the sound you'll be building. Books like "A Synthesist's Guide to Acoustic Instruments" (Amsco Publications,) can help by pointing out features of common acoustic sounds, and analyzing these features in terms of synthesizer processes (including digital FM). This book has presented you with several questions that need to be asked in order to eliminate obviously unsuitable algorithms. The first and most important of these questions is: how many sound sources (carriers) are needed? The second question is, how many modulators are needed, and how should they be plugged into these carriers? If one or more of these carriers are to be simple sine waves only, then you won't need modulators at all for those particular carriers. On the other hand, if you are trying to create a sound or part of a sound that must be harmonically very rich, overbright, or even distorted, then you will require at least one *stack* of modulators. Algorithms 16, 17, and 18, which only contain one carrier, will be useful only if you are attempting to create a single, non-chorused, non-detuned, harmonically rich sound - like the sound of a single bowed violin or single trumpet. They will not usually be a good choice for most ensemble sounds, where the interharmonic beatings caused by physical

resonances are extremely important (these kinds of movements can most successfully be simulated in the DX7II with detuned carriers beating against one another). On the other hand, algorithms 5 and 6, with three modulator-carrier systems, seem to provide the most scope for creating rich and complex sounds.

Algorithm 32, with no modulators, will probably be of very limited use to you unless you are looking to create organ sounds or are otherwise heavily into sine waves. In fact, algorithms 23 through 32 all contain at least one unmodulated carrier, and will therefore usually be of value only if a pure sine wave is needed somewhere in the sound (One tip: bass sounds are often helped greatly by the addition of a low-pitched sine wave, since sine waves carry the greatest low frequency content. In fact, you can produce quite usable - and recordable - bass lines from just a single unmodulated carrier, given the appropriate EG settings).

Algorithms 1, 2, and 18 also provide limited service, since they each contain stacks of *three* modulators - something you won't need too often unless you are creating either highly complex waveshapes, white noise, or intricate keyboard scaling patterns. Algorithms 5, 6, 12, 13, 20, 21, and 22 are all somewhat similar in that they provide no unmodulated carriers, but no stacks either. Each has varying numbers of carriers, and their feedback loops are in different places, so you should be able to choose between them fairly easily. Apart from these seven, all the algorithms with numbers lower than 23 have stacks in them and will useful when creating timbrally complex sounds (that is, sounds which are unusually bright).

Furthermore, algorithms 7 through 27 all contain systems of either two or more modulators into one carrier, or one modulator into two or more carriers. These are useful for setting up interesting animation techniques, or for setting up complex keyboard scalings where the timbre changes over the keyboard. **(see figure 16-1)**

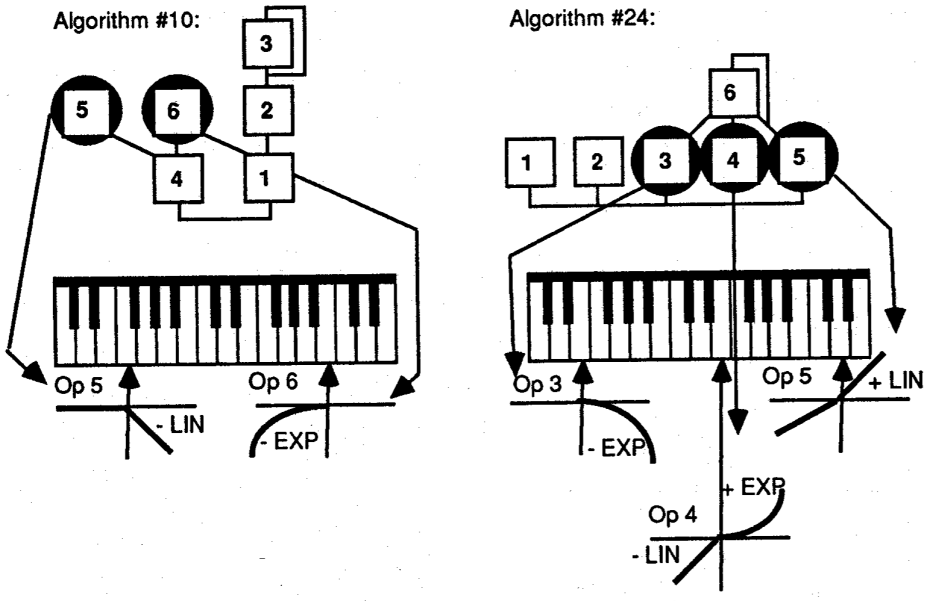


Figure 16-1

They're also useful for setting up complex timbral changes over time, using the EGs. **(see figure 16-2)**

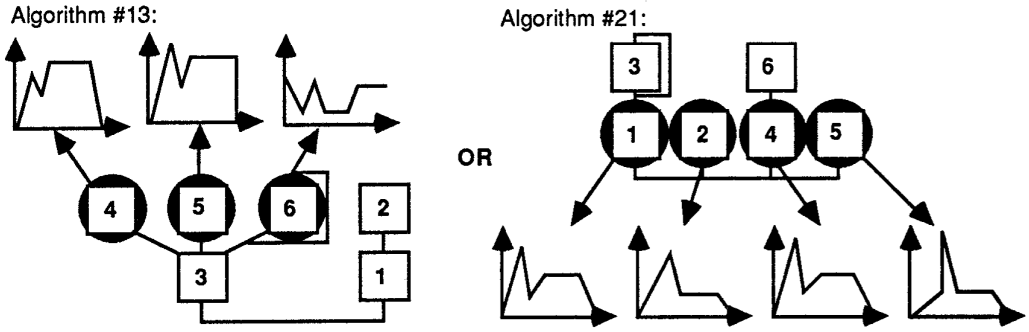


Figure 16-2

Here's a quick reference chart which may be of some use to you in selecting algorithms:

Algorithms with one carrier	Algorithms with two carriers	Algorithms with three carriers	Algorithms with more than three carriers
16, 17, 18	1 - 4, 7 - 15	5, 6, 19, 20, 26, 27, 28	21 - 25, 29 - 32

Algorithms with unmodulated carriers	Algorithms with modulator stacks	Algorithms with one-into-two or two-into-one systems
23 - 32*	1 - 4, 7 - 11, 14 - 19, 28, 30	7 - 27

The Envelope Generators

These are probably the most important tools in the DX7II, and also probably the most misunderstood. Often, the EG settings should be the first things you concern yourself with when creating or modifying a voice. *The envelope generators cause aperiodic change to a sound.* The operator EGs cause aperiodic change to operator output level, the pitch EG causes aperiodic change to pitch, and the Pan EG causes aperiodic change to either positioning or total voice level. Keep these sentences firmly implanted in your mind, along with Cardinal Rules One, Two, and Three, and a lot of the mystery will disappear.

Books such as "A Synthesist's Guide to Acoustic Instruments" show us that the changes in a sound are often as important as the actual components that make up the sound. Remember that the timbre, in particular, of almost every sound in existence changes throughout that sound's duration, and also that it rarely changes in precisely the same way as the volume. Translation: use the EGs in the modulators for complex timbral changes, but don't just copy carrier EG data to them and expect that you'll end up with a realistic sound. Think of the EG levels as a road map, on which you are plotting how the sound should change, and think of the EG rates as setting the speed with which it gets from point to point. Remember that the EG rates are absolute speeds and not absolute times: an R2 of 50 will mean different things, depending upon where L1 is and where L2 is, and also on what the nominal output level of the operator is. The ability to set L4 anywhere

* don't forget that you can always set a modulator's output level to 0 if you need an unmodulated carrier in a *different* algorithm!

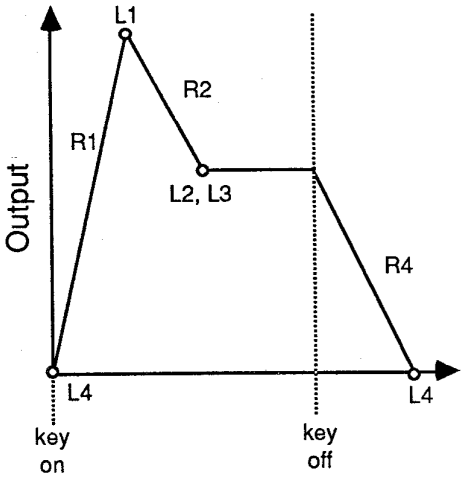


Figure 16-3

we want allows us, among other things, to simulate a "hold" or "drone" switch. Finally, keep in mind that the good folks at Yamaha have provided us with an envelope generator that can do far more than an analog ADSR: take advantage of that fact - you're not limited to just a simple how-quickly-does-it-die-away-and-maybe-release-time EG, as you are with many other synthesizers. Many beginning students of the DX7II create EGs that always look something like **figure 16-3.**

A sure giveaway of a DX7II novice! Become an Expert in the Privacy of Your Own Home: use the capabilities of the DX7II envelope generators to their fullest extent - and be thankful you don't own a Kurzweil, with its 255-stage EGs!

Brightness or lack thereof

Remember the relationship of modulators to carriers - modulator output level is the *quantitative* timbral control, while frequency ratio is the *qualitative* timbral control. Translation: if you want a brighter sound, a modulator somewhere will have to somehow increase its output level. This will not always be as simple as just turning to edit switch 10. Remember that there are often many factors controlling the output level of a particular operator - the operator's EG, keyboard velocity sensitivity (which, you'll remember, is a negative sensitivity control: reducing a modulator's sensitivity will *increase* the softly-struck key's brightness), keyboard level scaling, and EG bias sensitivity. Adjusting these parameters will often have a more natural effect than simply cranking up the nominal output level with edit switch 10. If you're seeking a more subtle change in brightness, you might think about changing the frequency ratio instead. Introducing higher harmonics or inharmonics (don't forget Frequency Fine!) into the sound will usually create the illusion of a sharper edge - and this may well be all you need to do.

Movement in a sound

The LFO provides periodic volume, pitch, or timbral changes, and can be very effective in generating subtle movements within a sound, particularly when used in *multi* mode. In *single* mode, on the other hand, the changes it induces are always very regular. You will therefore want to choose the LFO mode carefully, depending upon whether you are looking for a subtle, barely noticeable movement (as in a held violin note), or whether you are looking for a clearly heard periodic movement (as when you are trying to imitate a rotating-speaker effect in an organ sound).

Another very subtle way of generating movement in a sound involves the technique of putting a modulated carrier into a subaudio fixed frequency. Like the LFO in single mode, this also creates a movement which is unchanging in phase and speed from key to key. Often, the best choice for creating the most natural movements in a voice is the careful use of the Detune control, which adds a very natural beating effect to the sound - and one whose speed changes per note. Once a voice is organized into a performance memory, use of the Pan controls will help to "spread" the sound, thus also adding subtle movement. Detuning and Panning are two of the most powerful tools in the DX7II, because they really help to negate the digital perfection of the instrument, adding a "human" quality to the sounds generated. Be careful of overusing them, though - abuse of the Detune control in particular can backfire on you and just turn your sound into a muddy

mess. In general, it is recommended that you always at least try to add detuning to any voice you create, but do it as a last step, and in small and careful doses. Keep in mind the extraordinary effects that can be generated by applying detuning within a two (or more)-into-one or one-into-two (or more) system. **(see figure 16-4)**

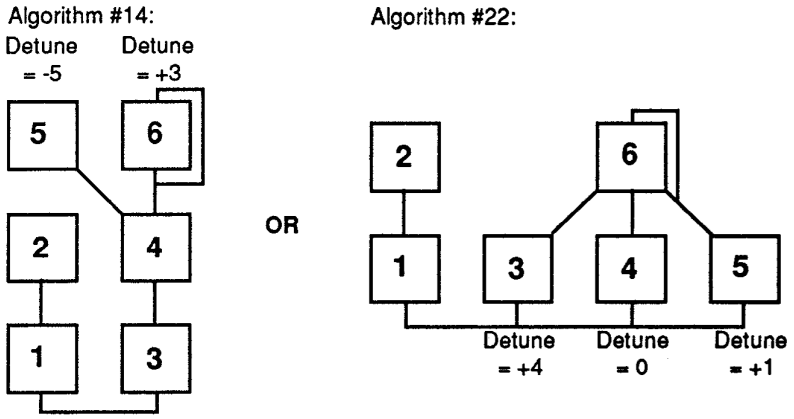


Figure 16-4

Conversely, if you are trying to remove a movement from a preset sound, don't always look to the LFO as the culprit. If you set both the pitch and amplitude modulation sensitivities to 0 and you still hear the movement, look to the Frequency Fine or Detune controls as the most likely source.

Keyboard scalings

One of the most amazing things we can do on the DX7II is to actually scale the keyboard so that operator output levels change (via the keyboard level scaling controls) or so that EG rates change (via the keyboard rate scaling controls). Remember that real sounds rarely maintain the same volume or timbre as they change pitch, and that in most acoustic instruments, higher notes undergo their changes over shorter periods of time than do lower notes. The abovementioned controls allow us to simulate these naturally occurring acoustic phenomena in the DX7II. Fractional level scaling, in particular, allows you to set up very subtle changes from note group to note group, and frees you from the restrictions of set curves.

You can also use the Pan control in a novel way to set up what in many samplers is known as a *positional cross-fade*. This is a technique whereby two similar sounds are placed over an area of the keyboard, and one subtly changes into another as you play higher or lower notes within that keyboard span. This can be accomplished in the DX7II by working in Pan Mode 0 or 1, using Note number as your panning control, and working in Dual mode with voices that are routed externally to the same (mono) output. Here, lower notes will give you more of voice A, while higher notes will give you more of voice B, with the two sounds cross-fading between one another over the center range of the keyboard. **(see figure 16-5)**

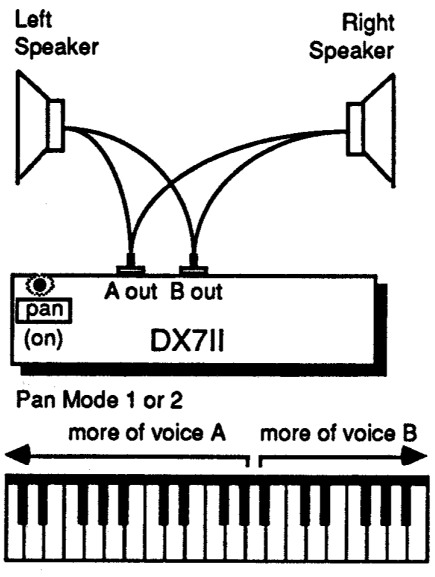


Figure 16-5

Keyboard scalings in general - whether they be normal or fractional level scalings, rate scalings, or creative pannings - are an area that should not be overlooked. Like the Detune control, they serve the important function of hiding the digital precision of this instrument and help to impart a more "real" quality to the sounds we create. On the

other hand, using these controls in unconventional ways allows you to set up unbelievably "unreal" sounds - and to generate audio effects that acoustic instruments could never accomplish. Either way, they're worth devoting time to.

Pitch changes

There are several ways of accomplishing this, depending upon whether you are trying to alter the pitch of a system within a voice, the voice itself, or the overall performance memory - and also whether your intention is to make fine or coarse, semitone-incremented changes to the pitch.

Cardinal Rule Three tells us how to alter the pitch of a system within a voice: make whatever change you want to the pitch data inputs of *all* operators in the system in the same way. The key Transpose function allows us to shift the entire keyboard, thus allowing for gross pitch changes to a particular voice. The Note Shift function allows us to alter the pitch of a voice within a particular performance memory in a gross manner, while the Dual Detune allows you to do it in a subtle manner when working with a specific performance memory made of Dual voices. The powerful micro tuning editing function not only lets you alter the pitch of a voice, it actually allows you to change the *gaps* between keys! The Master Tuning control is, however, the DX7II's only global tuning control, allowing for slight pitch changes for any and all data - voice or performance - called up into the edit buffer. The Random pitch control, the pitch bend wheel, and the pitch bias functions of the breath controller and keyboard after touch, are other ways of altering the pitch of a voice or performance memory in real time.

The pitch EG can also be used to shift the pitch of a voice without necessarily inducing a pitch *change* throughout the duration of the sound. Remember that setting pitch EG Levels of 50 will maintain the nominal pitch, so that setting all four Levels to, say, 48, will slightly flatten the overall sound without causing any actual fluctuation in the pitch from beginning to end. You can even use the LFO for this purpose. How? Simple - sync the LFO to the keyboard (using the LFO Sync control), select a square Wave in LFO single mode, and set the LFO Speed to 0. Now set the PMD to some value greater than 0. Surprise! Since the LFO sync causes the square wave to always start at the bottom of its cycle whenever you press a key down, the overall pitch will drop down and will stay down for a good minute or so. Every time you strike another note, the whole process starts all over again. Therefore, this technique will work fine as long as you don't need to hold a note or chord down for longer than a minute. Adjusting the Pmd will give you control over just how low you go. This is, admittedly, an unusual LFO application, but the point is that solutions to programming problems can often come from unexpected sources.

Expressiveness

The DX7II provides us with an inordinate number of expressive controls: the keyboard velocity and after-touch, the foot switches and controllers, the breath controller, the continuous sliders, the modulation wheel, the pitch bend wheel... the list goes on and on. These all allow you fingertip (or foot-tip, or ... lip-tip?) control over volume, brightness vibrato, tremolo, or any one of dozens of other periodic and real-time effects. One of the hallmarks of conventional acoustic instruments is that they can be played *expressively*. Up until very recently, the *lack* of

expressiveness was the hallmark of synthesizers. This is certainly not true of the DX7II. Don't allow a piano purist to accuse you of just pressing keys down on a "toy". Dazzle them with expressiveness! Shut them up for another day longer!!

Ambience

The extraordinary Pan control, discussed in detail in Chapter Fourteen, allows you to "spread" the sound created by the DX7II in exciting and constantly changing ways. In addition to using this control extensively and imaginatively, you can add a great deal of "depth" to your voices with the use of Rate 4 in your operator envelopes. This rate, you may remember, determines the amount of time it takes a sound to fade away after the key is released. By setting up offsets within the algorithm, and offsets in R4 values between voices in Dual mode, you can essentially create very convincing reverb effects! Such reverberation, whether artificial or not, will help push the sound back further in the near-far z-axis discussed in Chapter Fourteen, thus adding to the illusion of ambience.

You should also always consider the addition of outboard signal processing to your DX7II sounds. Devices such as digital reverbs, echos, chorusing, flanging, harmonizing, and phase-shifting units can help the sound immeasurably in certain contexts, although they always will add a certain amount of unwanted noise. The great synthesizer sounds you hear on record or on radio have almost always been treated with some amount of outboard signal processing, so you shouldn't expect to be able to completely duplicate these sounds without some of these kinds of effects added to your own sounds. It may be "cheating", but if it works...

Creating composite sounds

Perhaps the greatest strength of the DX7II is its ability to output two separate sounds simultaneously in Dual mode. This allows you to effectively create a single sound from the composite sum of two independent voices; allowing you to graft the scrape of a bow onto the sound of a violin, and the "mallet" onto the body of a vibraphone. Or - the scrape of a bow onto the body of a vibe and the "mallet" onto the sound of a violin! Because you can actually hear the changes being made in edit mode to one voice while listening to both (a capability that was absent in the DX7II's multitimbral predecessor, the DX5), you can even create complete sounds that, in effect, use twelve operators within one giant algorithm! These composite voicings require a good deal of skill and patience, but the results are really worth it. Advanced DX7II programmers tend to shun Split mode - you can always get two different sounds from any two synthesizers and "split" them with a master keyboard controller. The real coup de grace of this instrument is Dual mode, used to create component voicings. Exercise 90, below, will explore this exciting area.

Some final sage advice

If this entire book could be summed up in just three words, it would be these three: USE YOUR EARS! Many DX7II users seem to become inordinately fascinated with the numbers being constantly presented in the LCD by the microprocessor. But numbers, in and of themselves, are meaningless. This is an easy trap to fall into: don't become yet another victim of the dread Numbers Game. Stop constantly and listen to how

the sound has changed as you enter in these new digits. If it sounds good to you, it's good, no matter what the number in the LCD says. USE YOUR EARS!!

Creating generic voices

In an effort to sum up all of the above, let's now run a short series of exercises and see just how we can apply these principles to creating a generic brass sound, a generic string sound, and a generic vox humana sound in the DX7II.

Following each exercise is something called a "voice data chart" of the patch. These charts are provided by Yamaha and are useful for keeping alternative records of the sounds you create - just in case your machine's back-up battery ups and dies in the middle of the night. It's probably a good idea to get in the habit of filling out these charts for voices that are exceptionally important to you - Murphy's Law assures us that accidents can and do happen.

Let's start with a generic, all-purpose brass sound. This started out as an attempt to create a life-like solo trumpet but this also works pretty well as a brass ensemble sound, if you intonate chords on the keyboard in a realistic manner. Treat this sound, and the two others that follow it, not as an end product, but as a means to an end - in other words, use it as a stepping-stone to your own customized voices. After all, what my ears subjectively tell me sounds "good", your ears may say "expletive deleted", so let your *own* ears be the final judge of any sound you synthesize.

PLEASE note: If you have skipped ahead to this point without reading the preceding 200-odd pages, welcome aboard! Please be aware that a convention in doing these exercises has been somewhat painstakingly established, so if the instructions below confuse you at all, you'll probably have to go back and read many of those 200-odd pages! Sorry, but I never promised you a rose garden...

Note: In this exercise and all succeeding ones, wherever a parameter is not specifically mentioned, it is meant to be left at its initialized default settings.

Algorithm #18:

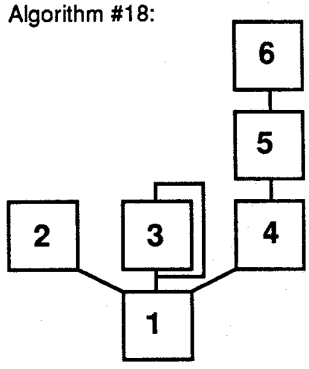


Figure 16-6

Exercise 87
Creating a generic brass voice

1) INITIALIZE your DX7II from single voice play mode and select algorithm #18. Because we are setting out to create a highly complex sound (since brass instruments are very bright - exhibiting many harmonic and inharmonic overtones in great strength), and because this was originally meant to be a solo sound, algorithm #18, with its single carrier and stack of three modulators, was the obvious choice. **(see figure 16-6)**

2) TURN OFF operators 2, 4, 5, and 6 ("101000") and change operator 3's output level to 78, leaving the default frequency ratio of 1.00 : 1.00. Since most brass sounds have characteristic sawtooth-like waveshapes, we'll start with this ratio. Because we'll be using four other modulators to induce more overtones, we'll start with a relatively subdued sound, hence the lowered output level. Enter the following EG data for operators 1 and 3:

	Op 1	Op 3
L4	0	0
R1	59	46
L1	99	99
R2	24	35
L2	86	86
R3	99	99
L3	86	86
R4	55	50

Play a note and listen (Audio Cue 87A). It's starting to sound distinctly brass-like, as the modulator envelope. **(see figure 16-7)** is giving the distinctive "wah" sound to the timbre. But it's still not quite bright enough. Fortunately, in this algorithm, you will note that the feedback loop is assigned to operator 3. Change the feedback value to 7, play a note and listen (Audio Cue 87B). Note that even though feedback is now at maximum, we're not hearing appreciable distortion or white noise, since the output level of operator 3 is still quite low (at a value of 78 - see step 2 above). Experiment by increasing the output level of operator 3 and note that you hear the distinctive distortion and white noise. When you're done, restore operator 3's output level back to 78.

3) TURN OFF operator 3 and TURN ON operator 2 ("110000"). We're going to use operator 2 only for a gentle addition of a few upper harmonics, so change its Coarse frequency to a new value of 23.00 and just raise its output level to a value of 10. This will ensure that these upper harmonics don't cut through too much - we just want to add a little edge to the sound. Enter the following EG values for operator 2: L1 = 80, L2, L3, and L4 = 0; R1 = 41, R2 = 34, R4 = 70. R3 is, of course, irrelevant, and can be left at the default of 99. This gives us a very simple envelope which looks like **figure 16-8.** Play a note on the keyboard and listen (Audio Cue 87C). This will be the total contribution of operator 2 to this sound - a small one, but an important one.

4) TURN OFF operator 2 and TURN ON the stack of operators 4, 5, and 6 ("100111"). Let's leave the bottom modulator, operator 4, at the default frequency, and set its output level to 79, so that we are once again working with a quasi-sawtooth wave. We're going to use the stack above it to modify it and add in the characteristic distortions and inharmonics that come from blowing hard into a brass tube like a trumpet. Therefore, change operator 5's frequency value to 4.76, and operator 6's to 7.63 - and set their output levels to 75 and 47 respectively. Detune operator 5 a little, to a value of -1. This will give us a very slow beating effect which will only be audible on held notes. Enter the following EG data for operators 4, 5, and 6:

	Op 4	Op 5	Op 6
L4	0	0	0
R1	66	48	77
L1	53	98	99
R2	92	55	56
L2	61	61	0
R3	22	22	99
L3	62	62	0
R4	50	50	70

Play a note and listen (Audio Cue 87D). This will be the total contribution of the stack. Now TURN ON operators 2 and 3 ("111111") and listen (Audio Cue 87E). We're getting there!

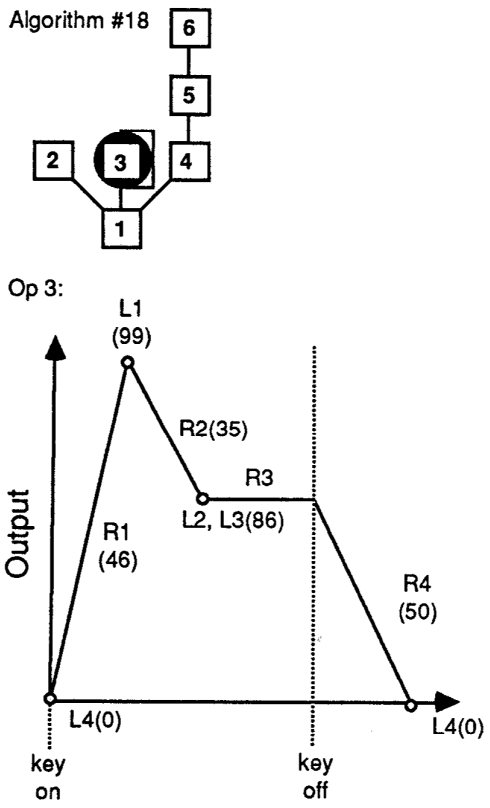


Figure 16-7

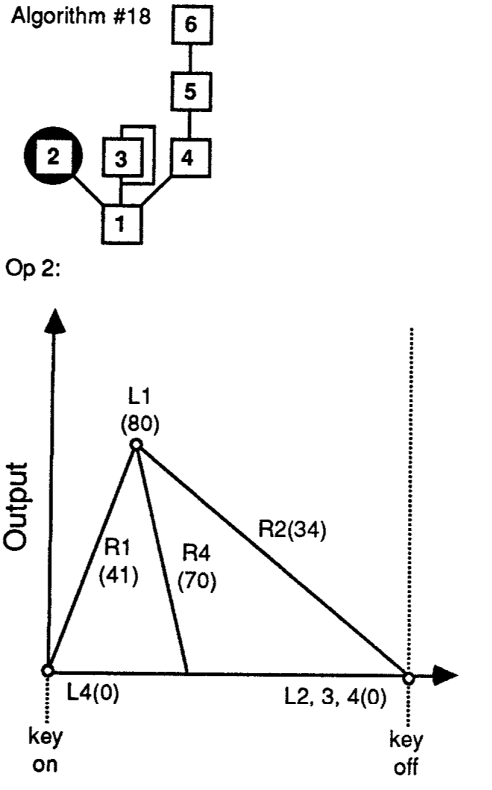


Figure 16-8

5) One thing that may be apparent as you try out this sound over the full range of the keyboard is that it's a little unrealistically overbright at this point. We can start to deal with that by adding some keyboard velocity sensitivity so that only keys which are struck with substantial force will have this characteristic - after all, when a horn player really blows hard, these kinds of distortions begin to occur in the sound. Therefore, set the velocity sensitivities for operators 2 and 3 to a value of 1. Also, when a horn player blows hard, the volume increases somewhat. In order to simulate that, set the velocity sensitivity for operator 1 to a value of 2. So far, so good, but it's still a little overbright, particularly in the highest notes. This calls for some keyboard level scaling. Set a break point for operator 3 of A#4, and set a -lin curve to the right of that break point. A right depth of 10 will do nicely, just enough to roll off some of those unpleasant distortions at the upper end of the keyboard. **(see figure 16-9)**

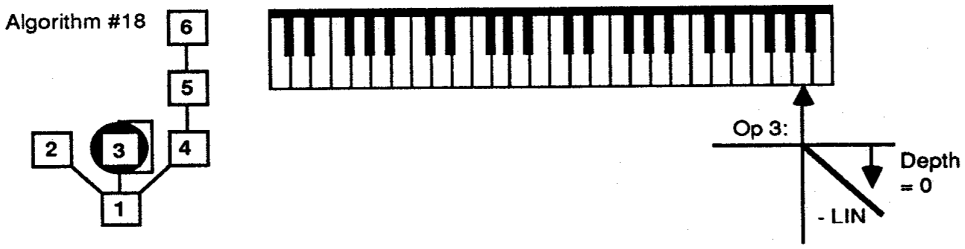


Figure 16-9

6) While a horn player can theoretically sustain a high note just as long as a low note, the timbral changes that occur within that high note happen somewhat faster than in lower ones. Therefore, set the keyboard rate scalings for some of our modulators as follows: operator 2 = 2; operator 3 = 1; and operator 6 = 7. This will accurately simulate this acoustic effect. Because there is a fair amount of release time in the carrier (since operator 1's R4 = 55), we'll also have to scale the carrier a bit so it doesn't ring out for unnaturally long periods of time when playing high notes. Therefore, set the keyboard rate scaling for operator 1 to a value of 2. Play a note and listen (Audio Cue 87F).

7) Finally, let's set up the LFO for a nice slow vibrato, to be routed through a real-time controller. Change the LFO Wave to a sine wave, set the LFO Speed to a value of 26, and turn the LFO key Sync OFF. Now set the pitch modulation sensitivity to a very slight value of 1, and, by turning any of your real-time controllers ON for the pitch destination ("Pmod"), you'll be able to bring a realistic slow vibrato effect in and out at will.

8) That's all there is to it! Play a note and listen (Audio Cue 87G). Instant generic brass... try a few well-intonated chords and you'll find that this sound works well as a brass ensemble also.

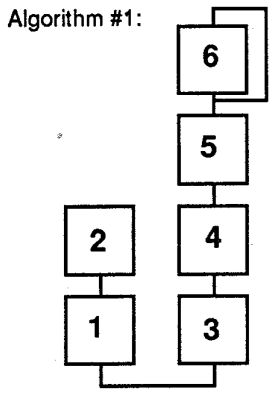


Figure 16-10

DX7 II·FD/D

Voice name : Generic brass

Date : / /

ALGORITHM	
ALG	18
FBL	7
OSC.Sync	on
Transpose	C3
L F O	
Wave	sine
Speed	26
Delay	0
Mode	multi
PMS	1
PMD	0
AMD	0
Sync	off
Pitch E G	
Range	8 oct
Velocity	off
RS	0
R1	99
R2	99
R3	99
R4	99
L1	50
L2	50
L3	50
L4	50

OSCILLATOR OP	1	2	3	4	5	6
Mode	r	r	r	r	r	r
Coarse·Fine	1.00	23.00	1.00	1.00	4.76	7.63
Detune	+0	+0	+0	+0	-1	+0
E G OP	1	2	3	4	5	6
RS	2	2	1	0	0	7
R1	59	41	46	66	48	77
R2	24	34	35	92	55	56
R3	99	99	99	22	22	99
R4	55	70	50	50	50	70
L1	99	80	99	53	98	99
L2	86	0	86	61	61	0
L3	86	0	86	62	62	0
L4	0	0	0	0	0	0
Output Level OP	1	2	3	4	5	6
Scaling mode	n	n	n	n	n	n
Output Level	99	10	78	79	75	47
LD	0	0	0	0	0	0
LC	-lin	-lin	-lin	-lin	-lin	-lin
BP	C3	C3	A#4	C3	C3	C3
RC	-lin	-lin	-lin	-lin	-lin	-lin
RD	0	0	10	0	0	0
Sensitivity OP	1	2	3	4	5	6
Velocity	2	1	1	0	0	0
AMS	0	0	0	0	0	0

Key mode	
Key assign mode	poly
Unison detune	-
Pitch Bend	
Range	2
Step	0
Mode	normal
Portamento	
Mode	retain
Step	0
Time	0
Random pitch S.	2
Modulation Wheel	
P. MOD	50
A. MOD	0
EG. B	0
Breath Control	
P. MOD	0
A. MOD	0
EG. B	0
P. Bias	+0
After Touch	
P. MOD	0
A. MOD	0
EG. B	0
P. Bias	+0

Foot control 1	
P. MOD	0
A. MOD	0
EG. B	0
P. Bias	+0
Foot control 2	
P. MOD	0
A. MOD	0
EG. B	0
P. Bias	+0
MIDI IN control	
P.MOD	0
A. MOD	0
EG. B	0
P. Bias	0

Now let's take a look at how to create a generic string voice. While this is admittedly not one of the DX7II's strengths, we can nonetheless simulate a fairly good bowed violin/viola/cello/bass sound. One of the problems, of course, is that each of these sound somewhat similar but are nonetheless different instruments, and synthesists have become accustomed to having all four at their fingertips over the whole keyboard. Fortunately, the sophisticated keyboard level scaling controls provided by the DX7II help us to get around this problem:

Exercise 88
Creating a generic string voice

1) INITIALIZE your DX7II from single voice play mode and select algorithm #2. We've selected this algorithm because the complex harmonics induced by bowing a stringed instrument will require a stack of three modulators, and the complex beating and detuning effects we will need tend to point us in the direction of two carriers. Since both carriers will need to have overtones (a pure sine wave being of very little use here), we are essentially given the choice of algorithm #1 **(see figure 16-10)** or algorithm #2 **(see figure 16-11).** The sole difference between these two is in their feedback loop. Because we're going to need the feedback loop on the single modulator-carrier system, rather than on the stack, we've selected algorithm #2.

Algorithm #2:

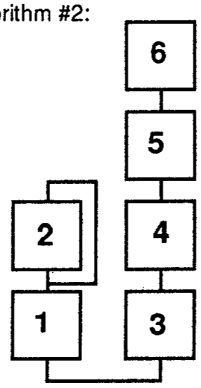


Figure 16-11

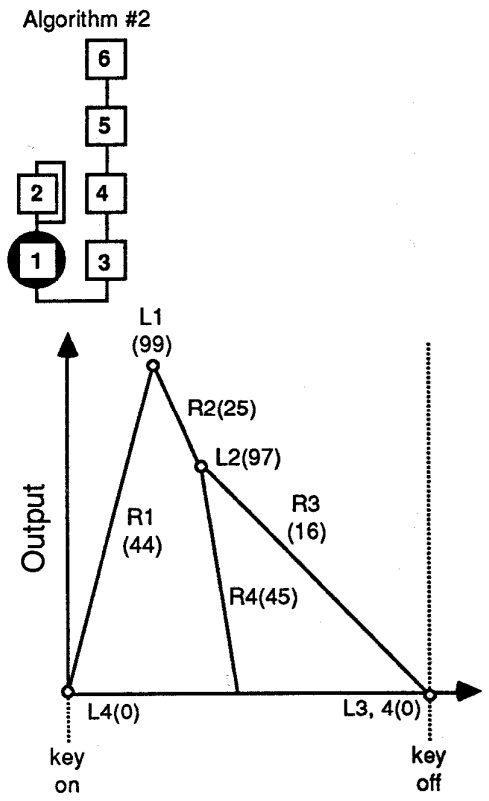


Figure 16-12

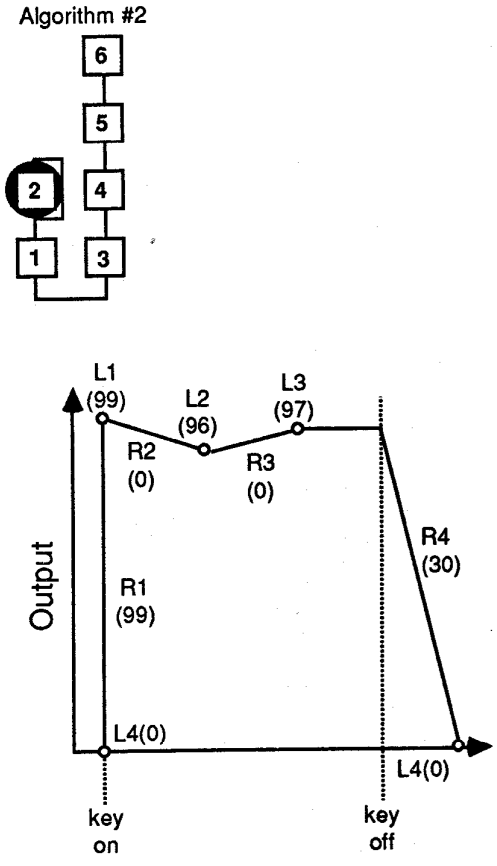


Figure 16-13

2) We'll start as usual by working with the individual systems inside the algorithm. We're going to use the stack for the "bowing" part of the sound, and then blend it with a smoother string sound we'll create with the other system. Let's program the simpler system first. TURN OFF operators 3 through 6 ("110000"). Because stringed instruments produce a characteristic pulse-waveish shape, we'll begin by changing the frequency of operator 2 to 2.00, and by setting its output level to 76 (thereby yielding a significant amplitude of the odd-numbered harmonics which are normally present in a square wave). This still isn't nearly bright enough, so we'll also use the feedback loop that's conveniently on operator 2 and change it to the maximum value of 7. Play a note on the keyboard and listen (Audio Cue 88A). Note the characteristically overbright overtones. As with the generic brass sound, even though the feedback loop is all the way open, we're still not hearing distortion or white noise since the output level of operator 2 is fairly low.

3) In a string section, of course, there is always a great deal of movement. As a first step in animating the sound further, let's detune operator 1 to a value of +1, and operator 2 to -7. Play a note and listen (Audio Cue 88B). That's better, but we'll need to animate the sound even more. Therefore, change operator 1 to a fixed frequency, and set that frequency at 1.259 Hz. Play a note and listen (Audio Cue 88C). There's plenty of movement, up one full octave, since when a carrier travels at a sub-audio frequency, it takes as its fundamental the pitch of the nearest modulator (operator 2, in this case). There's plenty of movement, alright, but it still doesn't sound much like a string section. Obviously, this is because our EGs are still at their static default states. Enter the following EG values for operators 1 and 2:

	Op 1	Op 2
L4	0	0
R1	44	99
L1	99	99
R2	25	0
L2	97	96
R3	16	0
L3	0	97
R4	45	30

Play a note and listen (Audio Cue 88D). The carrier is now rising to maximum volume at a moderate rate, dropping slowly to a slightly lesser volume, and then very slowly dying back to 0, **(see figure 16-12)** whereas operator 2 is in full force right away, but then very slowly drops to a slightly lesser output, and then very slowly increases output slightly. **(see figure 16-13)**

4) Okay, that's enough manipulation of this system for the time being - now let's work with the other system. TURN OFF operators 1 and 2 and TURN ON operators 3 through 6 ("001111"). Start by setting maximum output (99) for operator 3 (the carrier). Let's begin with a sawtooth waveshape this time, but raise it up an octave. Therefore, set the frequency of both operator 3 and operator 4 to 2.00, and set operator 4's output level to 87. Now let's use operator 5 to change operator 4 into a complex waveshape with much higher overtones. Set operator 5's frequency to 8.00 and change its output level to 77. Finally, we're going to use the top modulator - operator 6 - to induce some inharmonics, but we want these inharmonics to *change* with each new note we play on the keyboard. How do we accomplish this? Simple -

we put operator 6 into an audible range fixed frequency* - let's set it at 2042 Hz, give it just a bit of output level (44), and see what happens.

5) If you have a strong stomach and a good imagination, play a note and listen (Audio Cue 88E). Of course, this weird timbre doesn't sound much like a string section, but, again, the operator EGs in this system are all still at their square default settings. Therefore, enter the following EG values for operators 3, 4, 5, and 6:

	Op 3	Op 4	Op 5	Op 6
L4	0	0	0	0
R1	53	92	99	97
L1	99	99	99	99
R2	18	30	49	99
L2	95	98	90	99
R3	17	0	55	99
L3	0	90	80	99
R4	56	35	46	59

Play a note and listen (Audio Cue 88F). The carrier in this system is behaving similarly to the carrier in system one (operator 1) but with enough slight differences that they are offsetting one another. The modulators are also following somewhat similar, offsetting patterns, with operator 6 actually pretty much remaining with the default square envelope. It is the quick attack of operators 4, 5, and 6 that is primarily responsible for the bowing sound that we hear. TURN ON operators 1 and 2 again ("111111") in order to hear the total effect. Two other "bonuses" are derived from the EGs we've set up here. First of all, the moderate R4s in all the operators imparts a sort of digital reverb effect to the total sound. Secondly, the L3 values for the carriers are both 0, and the R3 values for these same operators are slow - meaning that this sound will not only be sustaining, but will also perform a nice, natural "fadeout" (like a violinist's arm getting tired!) if we hold a key down. Try holding a chord for some time and listen (Audio Cue 88G). Note the natural fadeout that occurs - one which is similar to, but a bit longer than, the one which occurs when playing notes staccato.

6) Of course, we'll need to add some movement into this system too, so detune operator 5 to +3 and operator 6 to +5. Next, let's do something fairly unusual and detune this system's carrier also: set operator 3 to the same detune value as operator 1, that is, +1. We wouldn't ordinarily detune both carriers in two-carrier algorithm, since that will have the effect of making the overall sound slightly out of tune, but the reason here is to allow us to somewhat offset an effect we're going to set up with the Pitch EG:

7) Studies of string instruments show that they undergo a subtle pitch change through their duration, as well as the more apparent volume and timbral changes. We can use the Pitch EG to simulate this. Press edit switch 13 and change the following Pitch EG values from their initialized defaults: Range = 1 octave, L1 = 53, L2 = 49, R1 = 87, R2 = 94, and R3 = 0, yielding a Pitch EG shape like **figure 16-14.** In other words, the pitch will quickly (R1 = 87) rise up nearly a quarter tone upon "key on" (L4 = 50 but L1 = 53) and will even more quickly (R2 = 94) drop down to go a bit flat (L2 = 49) and then ever so slowly (R3 = 0) sneak back to the initial pitch (L3 = 50). This extremely subtle use of the Pitch EG really helps to add realism to the sound. Play a note and listen (Audio Cue 88H).

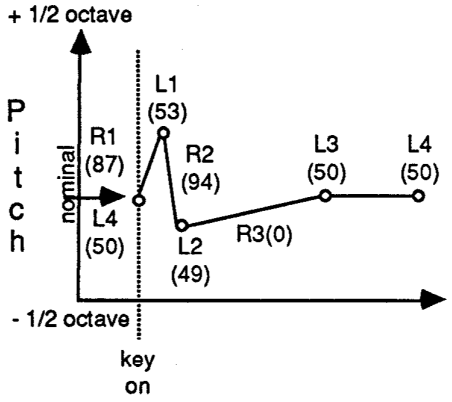


Figure 16-14

* this will have the effect of giving us a different frequency ratio for each note on the keyboard. See Chapter Six if you don't remember this trick.

8) We're starting to get there! However, this sound is still unpleasantly overbright. Because a string section exhibits a great deal of dynamic range and also is capable of undergoing a great deal of timbral change (depending on how the instruments are played), we will want some kind of real-time control over both volume and timbre. Keyboard velocity sensitivity, being a negative sensitivity control, is the answer, since it will also serve to compensate for the current overbrightness of our sound, and make it far more realistic than it currently is. Therefore, enter the following keyboard velocity sensitivity values: operator 1 = 2; operator 3 = 7; operator 5 = 2. The reason we have assigned so much sensitivity to operator 3 (the carrier in the second system) is so that we will hear much more of the bowing sound when we strike keys with greater force. Play a few notes at various different velocities and listen (Audio Cue 88I). Since the keyboard is currently transposed at a range in which the highest violin sounds will be too high, and the lowest bass notes not low enough, use the key Transpose parameter to drop the keyboard down one octave (so that Middle C = C2).

9) Now for the scalings. First of all, violins and violas undergo both volume and timbral changes somewhat more quickly than cellos or basses, so we'll need to do a certain amount of keyboard rate scaling. Enter the following values: operators 1 and 2 = 1; operators 3 through 6 = 2. Play a note and listen (Audio Cue 88J). Secondly, we'll need to adjust the relative levels of the various modulators over the keyboard in order to simulate the fact that we are attempting to recreate three or four different instruments over the one keyboard. Therefore, enter the following normal keyboard level scaling values for operators 2, 4, 5, and 6:

	Operator 2	Operator 4	Operator 5	Operator 6
Bp	B 1	G 3	B 2	F# 2
Rc	- exp	- lin	- LIN	- LIN
Rd	35	47	22	45
Lc		+ lin		
Ld		4		

As you can see, operator 2 is being gently attenuated from about F3 on up (since the exponential curve has virtually no effect for about two octaves), with the most drastic decrease in the upper (violin range) notes. Operator 4, whose output is actually determining the effects of operators 5 and 6, is having its effect slightly increased below G3 and more drastically decreased above that point (again, affecting the violin sounds). This is the reason why the bowing component will be most prominent in the bass and cello range notes. Finally, the effects of operators 5 and 6 are pretty much being attenuated over the upper three octaves. These complicated keyboard level scalings are actually vital to our being able to somewhat accurately simulate the differences between each of these instruments. Play several notes and chords through the full range of the keyboard and listen (Audio Cue 88K). Note how the timbre of this sound now changes realistically as we play in the different ranges.

10) All that's left to do now is to set up the LFO for a subtle vibrato effect which can be added to by any of our real-time controllers. Leave the LFO Wave at its default triangle setting, and change the Speed value to 24. Set the Pmd to a very subtle 17, and change the Pms value to 1. Now set up any of your real-time controllers to route pitch modulation, and listen (Audio Cue 88L). Instant generic strings! Once again, treat these as a building block from which to program your own customized string sounds.

DX7 II·FD/D

Voice name : Generic string
Date : / /

ALGORITHM	
ALG	2
FBL	7
OSC.Sync	off
Transpose	C2
L F O	
Wave	triangle
Speed	24
Delay	0
Mode	multi
PMS	1
PMD	17
AMD	0
Sync	off
Pitch E G	
Range	1 oct
Velocity	off
RS	0
R1	87
R2	94
R3	0
R4	99
L1	53
L2	49
L3	50
L4	50

OSCILLATOR OP	1	2	3	4	5	6
Mode	f	r	r	r	r	f
Coarse·Fine	1.259	2.00	2.00	2.00	8.00	2042
Detune	+1	-7	+1	+0	+3	+5
E G OP	1	2	3	4	5	6
RS	1	1	2	2	2	2
R1	44	99	53	92	99	97
R2	25	0	18	30	49	99
R3	16	0	17	0	55	99
R4	45	30	56	35	46	59
L1	99	99	99	99	99	99
L2	97	96	95	98	90	99
L3	0	97	0	90	80	99
L4	0	0	0	0	0	0
Output Level OP	1	2	3	4	5	6
Scaling mode	n	n	n	n	n	n
Output Level	99	76	99	87	77	44
LD	0	0	0	4	0	0
LC	-lin	-lin	-lin	+lin	-lin	-lin
BP	C3	B1	C3	G3	B2	F#2
RC	-lin	-exp	-lin	-lin	-lin	-lin
RD	0	35	0	47	22	45
Sensitivity OP	1	2	3	4	5	6
Velocity	2	0	7	0	2	0
AMS	0	0	0	0	0	0

Key mode	
Key assign mode	poly
Unison detune	-
Pitch Bend	
Range	2
Step	0
Mode	normal
Portamento	
Mode	retain
Step	0
Time	0
Random pitch S.	1
Modulation Wheel	
P. MOD	0
A. MOD	0
EG. B	0
Breath Control	
P. MOD	0
A. MOD	0
EG. B	0
P. Bias	+0
After Touch	
P. MOD	0
A. MOD	0
EG. B	0
P. Bias	+0

Foot control 1	
P. MOD	0
A. MOD	0
EG. B	0
P. Bias	+0
Foot control 2	
P. MOD	0
A. MOD	0
EG. B	0
P. Bias	+0
MIDI IN control	
P.MOD	0
A. MOD	0
EG. B	0
P. Bias	+0

Next, let's generate a generic human voice sound - the trickiest assignment of the three. The only natural acoustic sound known which is believed to be timbrally more complex than the human voice is the acoustic piano - and I have yet to create or hear a DX7II do a convincing imitation of an acoustic piano. This is, more than anything, a limitation of the number of operators available to us in the DX7II - an instrument like the Yamaha TX816, with its *forty-eight* (!) operators, can actually do a pretty good job of it. However, with that disclaimer in hand, let's get ourselves at least into the ballpark of vox humana sounds (in this instance with the sound of a choir voicing the syllable "ah") with the following exercise:

Algorithm #14:

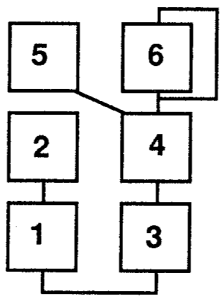
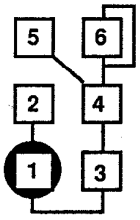


Figure 16-15

Exercise 89
Creating a generic vox humana voice

1) INITIALIZE your DX7II from single voice play mode and select algorithm #14. We have selected this algorithm **(see figure 16-15)** for much the same reasons as we had for selecting algorithm #2 in the last exercise. We know that we'll need a stack, but since the human voice is far less bright than a bowed cello, we won't need three modulators in the stack. However, we want the option of having different timbres over the keyboard in order to simulate some of the differences between soprano, alto, tenor, and bass voices, and the two-into-one system of operators 6, 5, and 4 in this algorithm will lend itself nicely to that.

Algorithm #14



Op 1:

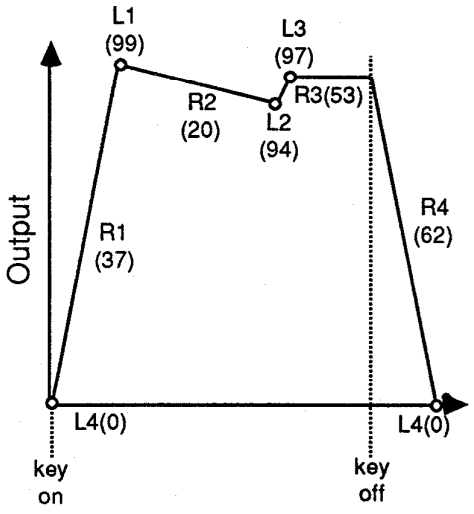
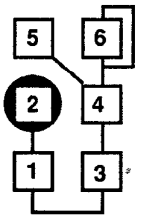


Figure 16-16

Algorithm #14



Op 2:

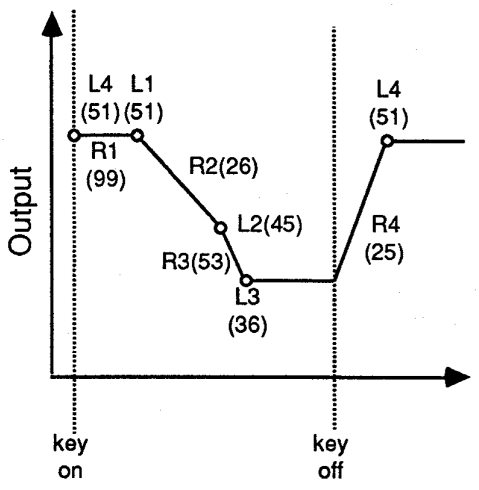


Figure 16-17

Algorithms 14 and 15 vary only in the placement of their feedback loop, but, for this voice, we'll need the feedback loop in the stacked system.

2) As usual, let's work with one system at a time. TURN OFF operators 3 through 6 ("110000") and set up a frequency ratio between operators 2 and 1 of 1.00 : 2.00. This rather unusual ratio will have the effect of raising the pitch an octave and will also yield a very unusual, slightly nasal, gently harmonic timbre. (As we learned in Chapter Six, whenever the carrier is moving faster than the modulator, this type of resonance will occur). Raise the output level of operator 2 to 99. We will shortly be using the EG in operator 2 to attenuate this output level by having none of the EG levels anywhere near 99, so if you listen to the sound at this point, it will have many more overtones than we're going to end up with. Remember that the rates in the EGs vary slightly as a function of the operator output level (as discussed in Chapter Nine), and so this is a valid alternative to simply always setting whatever nominal output Level you need (with edit switch 10) and then setting at least one of the EG levels to a value of 99. The technique we are proposing here will accomplish much the same effect, but the EG rates will be scaled slightly differently.

3) Once again, the fact that the EGs are currently at their square default shapes is keeping us from hearing this sound at anything close to reality. Therefore, enter the following EG values for operators 1 and 2:

	Op 1	Op 2
L4	0	51
R1	37	99
L1	99	51
R2	20	26
L2	94	45
R3	53	53
L3	97	36
R4	62	25

Play a note and listen (Audio Cue 89A). The carrier is undergoing a slow series of movements, fading in rather gently, then dropping a bit in volume, and then somewhat more quickly rising again in volume. **(see figure 16-16)** while the modulator, as promised, is never getting beyond about a quarter of its nominal output level (remember that most DX7II controls are exponential and not linear; this means that an EG level of 51 translates to about a quarter of the output that a level of 99 would yield). The fact that L4 is the same as L1 (also 51), means that the first timbral change we hear is a slight drop to L2 (45), followed by a faster drop again (to L3). After "key off", as the carrier is moderately fading away, operator 2 will actually increase its output level back to the starting point of L4. **(see figure 16-17)** If you listen carefully to this system alone (Audio Cue 89B) you should just about be able to hear the slight timbral increase after "key off".

4) The first system in this algorithm is only going to be contributing a very slight upper edge to the total sound: the meat of it will come from the stack provided by the other system. TURN OFF operators 1 and 2 and TURN ON operators 3 through 6 ("001111"). Begin by giving the carrier (operator 3) maximum output level (=99). Then set operator 4's output level also to 99. As with the first system, we'll be using operator 4's EG to greatly attenuate this output level. Set up a frequency ratio between operators 4 and 3 of 1.01 : 1.00, which causes a rapid detuning effect. Now open operator 5's output level to a value of

53, and leave it at its default frequency of 1.00. Raise operator 6's output level to a value of 45, and set its frequency to 5.10. We'll be using operator 6 to provide the highest harmonics found in the singing voice, and also the slight upper inharmonics which are inevitably present as a result of breathiness. In order to add a little more "edge" to the sound, open the feedback loop (which in this algorithm is on operator 6) to a value of 4. Now enter the following EG values for operators 3, 4, 5, and 6:

	Op 3	Op 4	Op 5	Op 6
L4	0	56	0	0
R1	42	72	35	99
L1	99	48	99	99
R2	20	19	21	99
L2	32	58	90	99
R3	53	41	36	99
L3	97	20	85	99
R4	53	12	63	17

Play a note and listen (Audio Cue 89C). The carrier in this system is undergoing a moderate attack, followed by a slow (R2) decrease in volume (L2), followed by a more rapid increase back up to nearly maximum (L3). This "roller-coaster" effect is meant to simulate the characteristic change in volume that choral singers typically exhibit when holding a single note. **(see figure 16-18)** (PLEASE note once again that although the human voice is capable of generating literally millions of complex sounds, but that we are here attempting only one of the simplest of them - the choir "ah".) As promised, the EG of operator 4 is serving to attenuate its output level in much the same way that operator 2 was affected by its EG. This time, however, we have set up a slight *increase* in output level, followed by a decrease, as opposed to the reverse effect engendered by operator 2's EG. **(see figure 16-19)** Operator 5 is only undergoing a very slight decrease in output level throughout its duration, while operator 6 is almost entirely at its default settings, with the exception of having a very slow R4. The fact that R4 is so slow for both operator 6 and operator 4 simply means that the overtone content they are causing to be present in the carrier will remain virtually unchanged even after "key off". **(see figure 16-20)**

5) TURN ON operators 1 and 2 ("111111") in order to hear the total sound and listen (Audio Cue 89D). The human voice is potentially capable of extreme dynamics, but we won't need to incorporate a whole lot of dynamic range into the simple "ah" sound we're attempting here. Therefore, set keyboard velocity sensitivity values as follows: operator 2 = 2; operator 3 = 3; operators 5 and 6 = 1. This will give us slight timbral changes as we play keys harder, and also a slight increase in the component of the sound coming from the stack.

6) Because both operator 4 and operator 6 are generating inharmonics with a fast beating effect, we won't need to use detuning to add more movement to this sound. However, in this instance we'll use the regularity of the LFO to add a bit of delayed vibrato, so change the LFO Wave to a sine, turn the LFO key Sync OFF, leave the Speed at the default of 35, but add in a Delay time of 35. Now change the Pmd to 3 and the Pms to 4, and listen (Audio Cue 89E). Change the Amd value to 3 and enter the following Ams values: operator 2 = 7; operator 4 = 2. This will induce a very slight "wah-wah" to the overall sound since we are affecting modulators only, and will also allow us, by using EG bias modulation, to use any of our real-time controllers as a subtle brightness control. Assign one of your controllers to EG bias modulation (with a range of about 50), and try it out (Audio Cue 89F). It's very subtle, but it works!

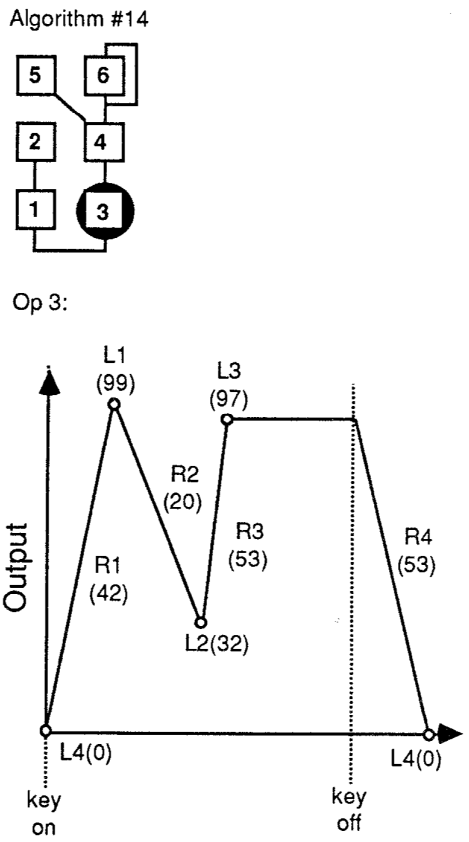


Figure 16-18

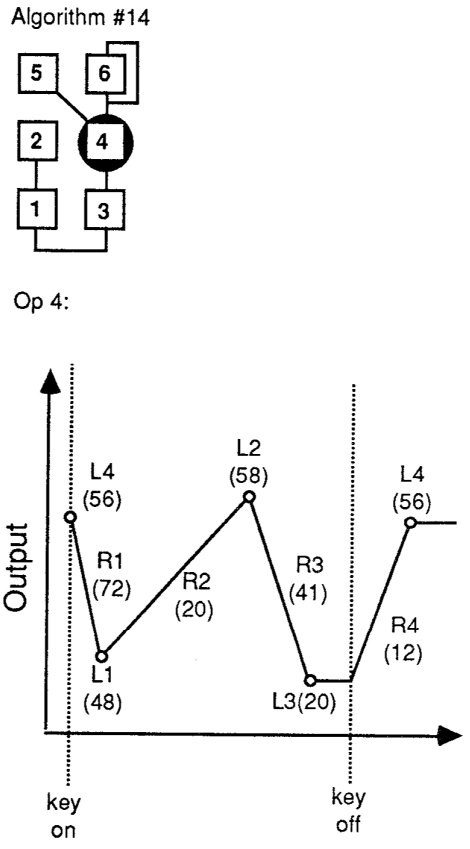
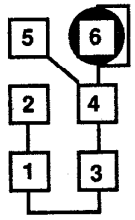


Figure 16-19

Algorithm #14



Op 6:

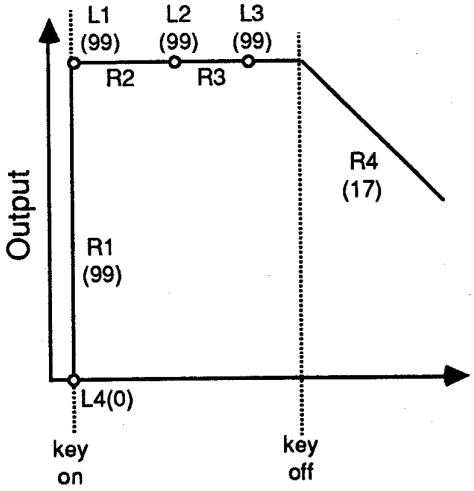


Figure 16-20

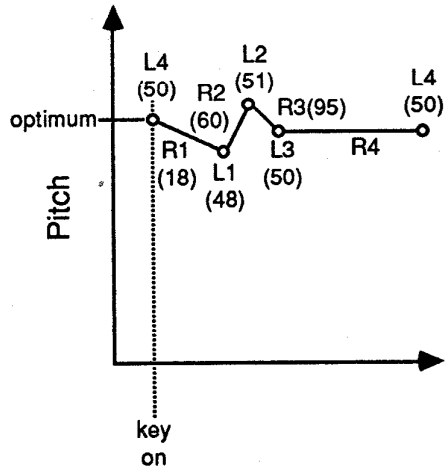


Figure 16-21

7) We only have to do a very small amount of scaling in this sound. Because human vocal chords don't quite react the same as inanimate strings, keyboard rate scaling appears to be inappropriate for this particular sound, so leave it out (it's already defaulted to 0 for all operators, anyway). In terms of keyboard level scaling, you've probably found that the sound is a bit dull at the lower end of the keyboard. Therefore, enter a Bp for operator 5 of C#6, enter a +exp curve to the left of that break point, and enter an Ld value of 99. This will serve to make notes below A#4 brighter and brighter, with the strongest effect at the very lowest notes.

8) Finally, to complete the picture, let's add in a slight pitch shift that's also characteristic of choir sounds. Enter in the following Pitch EG values: Range = 1 octave; L1 = 48; L2 = 51; R1 = 18; R2 = 60; R3 = 95. This will give us a Pitch EG that looks something like **figure 16-21** and yields a very slight wavering in pitch, all of which helps to "humanize" the sound. You might also try turning the Velocity control ON for the Pitch EG, so that you can control the amount of pitch shift with the force with which you play various keys. And there we have it: a generic human choir sound, which, once again, you are encouraged to use as a starting point to build your own customized sounds.

As we mentioned earlier in this chapter, perhaps the main strength of the DX7II is its multitimbral capability. Specifically, when working with dual voices, we can create a single sound which is a composite of two independent voices - thus creating a whole which can be greater than the sum of its parts. Let's run an exercise now which explores this fascinating ability. First of all, we'll create a "bowing" effect which can then be grafted onto and blended in with "Warm Stg A" - a beautiful preset sound, but one with such a long attack that it has no bowing component at all. Secondly, we'll create a sound which emulates that of a mallet hitting a metallic object. By building a component sound such as this one in isolation, we'll be able to graft it onto the preset "VibraPhone" sound - or onto other sounds (even "Warm Stg A"!) which you might not expect to have this component! What really makes this entire concept of component voicing possible in the DX7II is the fact that we can hear editing changes made to one of two voices while in dual mode - thus hearing how those changes will affect the total composite sound.

DX7 II·FD/D

Voice name : Generic voice
Date : / /

ALGORITHM	
ALG	14
FBL	4
OSC.Sync	off
Transpose	C3
L F O	
Wave	sine
Speed	35
Delay	35
Mode	multi
PMS	4
PMD	3
AMD	3
Sync	off
Pitch E G	
Range	1 oct
Velocity	off
RS	0
R1	18
R2	60
R3	95
R4	99
L1	48
L2	51
L3	50
L4	50

OSCILLATOR OP	1	2	3	4	5	6
Mode	r	r	r	r	r	r
Coarse·Fine	2.00	1.00	1.00	1.01	1.00	5.10
Detune	+0	+0	+0	+0	+0	+0
E G OP	1	2	3	4	5	6
RS	0	0	0	0	0	0
R1	37	99	42	72	35	99
R2	20	26	20	19	21	99
R3	53	53	53	41	36	99
R4	62	25	53	12	63	17
L1	99	51	99	48	99	99
L2	94	45	32	58	90	99
L3	97	36	97	20	85	99
L4	0	51	0	56	0	0
Output Level OP	1	2	3	4	5	6
Scaling mode	n	n	n	n	n	n
Output Level	99	99	99	99	53	45
LD	0	0	0	0	99	0
LC	-lin	-lin	-lin	-lin	+exp	-lin
BP	C3	C3	C3	C3	C#6	C3
RC	-lin	-lin	-lin	-lin	-lin	-lin
RD	0	0	0	0	0	0
Sensitivity OP	1	2	3	4	5	6
Velocity	0	2	3	0	1	1
AMS	0	7	0	2	0	0

Key mode	
Key assign mode	uni poly
Unison detune	1
Pitch Bend	
Range	2
Step	0
Mode	normal
Portamento	
Mode	retain
Step	0
Time	0
Random pitch S.	1
Modulation Wheel	
P. MOD	20
A. MOD	0
EG. B	0
Breath Control	
P. MOD	0
A. MOD	0
EG. B	0
P. Bias	+0
After Touch	
P. MOD	0
A. MOD	0
EG. B	50
P. Bias	+0

Foot control 1	
P. MOD	0
A. MOD	0
EG. B	0
P. Bias	+0
Foot control 2	
P. MOD	0
A. MOD	0
EG. B	0
P. Bias	+0
MIDI IN control	
P.MOD	0
A. MOD	0
EG. B	0
P. Bias	+0

Exercise 90
Creating composite sounds

1) INITIALIZE your DX7II from single voice play mode - this will always automatically mean that we are working with voice A only. We'll begin by creating a generic "bowing" component, and, when the sound is nearly complete, we'll "graft" it onto the "Warm Stg A" preset in order to make some final adjustments.

2) The first step, as usual, consists of selecting the best algorithm. We won't be needing more than one carrier, but we will need at least a single modulator plus a modulator stack in order to produce the large numbers of inharmonics necessary to this largely unpitched sound. Several algorithms give us this option - we chose algorithm #8 because of the placement of the feedback loop on the single modulator. (see **figure 16-22)**

Algorithm #8:

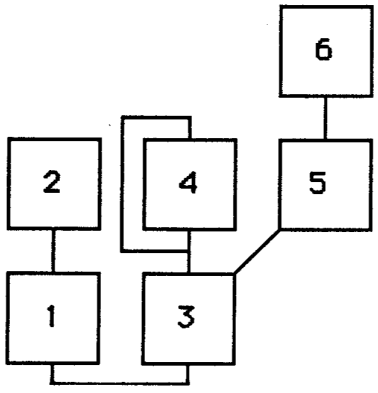


Figure 16-22

3) TURN OFF operators 1 and 2 - we won't be needing them at all. In order to make sure that operator 1 does not contribute at all to the final stored sound, set its output level to 0 (remember, operator 1 defaults to an output level of 99 when you initialize). The first, and perhaps most important characteristic of the bowing sound is its transient nature. Therefore, raise operator 3's output level (it is the only carrier we will be using) to 99, and enter in the following EG values for operator 3:

Rs	R1	R2	R3	R4	L1	L2	L3	L4
0	53	57	58	60	99	47	0	0

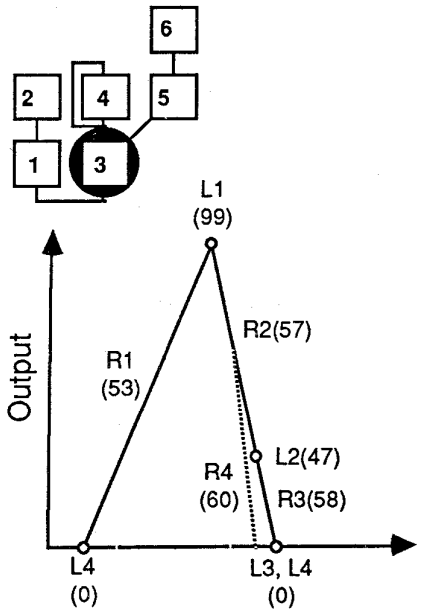


Figure 16-23

This creates an envelope which looks like **figure 16-23.** The moderate R4 time will be needed in order to help blend between the end of this component and the beginning of "Warm Stg A", which, as noted above, has a fairly long attack time.

4) Play a note and listen (Audio Cue 90A). You are hearing a very transient sound, but one which hardly sounds anything like bowing. There are two main reasons why: first of all, the bowing sound is largely pitchless (operator 3 is still tracking the keyboard), and secondly, since we haven't added any modulators yet, our sound still contains no overtones. Therefore, raise the output level of operator 4 to a new value of 49 and open its feedback loop to maximum (7). Change its frequency mode to Fixed. Enter in a value of 436.5 Hz for operator 4. Slow down R1 in operator 4's EG to a new value of 82, and slow down its R4 to a new value of 57, thus yielding little timbral change as the sound rapidly fades away.

5) We also don't want the bowing itself to change pitch as we play different notes on the keyboard, so change operator 3 also to Fixed frequency mode and enter in a frequency value of 1.905 Hz - meaning that there will be a slight timbral beating (important for emulating the timbral instability caused by the initial application of the bow to the string). Play a note and listen (Audio Cue 90B). It's a little closer, but not there yet.

6) We're at the point where it will be useful to hear this component in context with the overall sound we want to be creating. Therefore, name and store this voice somewhere in your internal or RAM4 cartridge memory (if you don't know what slot you want to store it in, use the Recall edit feature to allow you to VIEW the contents of your internal or RAM4 cartridge memory, and then store the voice). When you have successfully completed the process (check to make sure!), go to Dual voice play mode, and select this voice for voice A. Press the "A/B" switch (the status light will go on) and select "Warm Stg A" from your ROM cartridge (bank 1, slot 1). Note that even if you stored your original voice onto a RAM4 cartridge, you can still remove it in order to insert the ROM cartridge, thanks to the edit buffer!

7) Press the "A/B" switch in order to turn its status light OFF. This is a crucial step: because it is off, when we now return to edit mode, we will be able to VIEW the parameters for voice A (our newly created, nearly-finished bowing sound), while listening to this voice along with "Warm Stg A". Play a few notes and listen (Audio Cue 90C).

8) We're on the right track, but there are several problems. First of all, the bowing effect is much too loud. Therefore, press the edit switch in order to return to edit mode (as described in the step above, we'll be able to VIEW and change edit parameters for the new voice A, while hearing it in context along with the "Warm Stg A" assigned to voice B). While listening to both voices, adjust the output level of operator 3 (the carrier) until the blend seems about right (Audio Cue 90D). You should find that an output level of 80 works pretty well.

9) The other problem is that there seems to be a real gap between the end of this percussive, unpitched sound, and the beginning of the highly pitched voice B sound. In order to help bridge that gap, let's add a slight pitch component to voice A. Therefore, raise the output levels of operators 5 and 6 to new values of 74 and 82, respectively. Leave them in ratio mode, and at their default frequency values of 1.00. Leave operator 6 (the top modulator in this stack) at its default EG settings,

but enter in the following new EG values for operator 5 (the modulator that is carrying operator 6's signal):

Rs	R1	R2	R3	R4	L1	L2	L3	L4
0	99	99	35	47	99	99	0	0

This creates an envelope which looks like **figure 16-24.**

10) Play a note and listen (Audio Cue 90E). It's nearly there, but there are a couple more things we can do to more closely simulate reality. When a bow crosses a string, as we learned in Exercise 88 above), a subtle but important pitch change occurs, as the string is momentarily stretched. Now that we've added some pitch to voice A - and especially since this voice has a much faster attack than voice B - we'll need to duplicate this pitch change. The obvious tool that will allow us to do this is the pitch EG. Therefore, change the following pitch EG values from their defaults: L1=84, and R2=92. Set the pitch EG range at 1 octave, and turn the Velocity ON so that notes struck with harder force will induce more of this change.

11) We'll also want some dynamic control over this component, so enter in a Velocity sensitivity value of 3 for the carrier (operator 3), and also make operator 4 (the Fixed frequency modulator) slightly sensitive by entering in a value of 1. Play a few notes at different degrees of force and listen (Audio Cue 90F).

12) If you've played this up and down the full range of the keyboard, you've probably noticed that the sound is a little more like a trombone in the lowest registers. This is because operator 5 (modulator carrying the pitched signal from operator 6) is having an undue effect on low-pitched notes. Therefore, we'll use keyboard rate scaling to attenuate its effects: set a Bp of C6 and put a -exp curve to the left of that break point, with a left depth of 30. This will serve to smoothly roll off the effects of this modulator (and hence the one above it) as you play lower and lower notes. Play a few notes and listen (Audio Cue 90G) to the composite sound. Then store this modified bowing sound back into the same memory slot and return to Single voice play mode. Play a few notes in order to listen (Audio Cue 90H) to the bowing component alone. It may not sound like much on its own - but there's no question that it helps add "attack" in a realistic way to the "Warm Stg A" preset! experiment by further tweaking and fine-tuning the bowing component until it sounds good to your ears, in combination with "Warm Stg A" and then create and store a performance memory out of these two voices, using the procedure outlined in Chapter Fourteen.

13) Now let's try the considerably simpler task of creating the sound of a mallet hitting a metal bar. Re-INITIALIZE your DX7II from single voice play mode. We'll only need a single carrier for this sound, but we'll want to have about three single modulators feeding data into this one carrier so as to create a complex (albeit transient) series of high inharmonics, as is characteristic of this sound. Algorithms 16, 17, and 18 all provide this configuration, but we chose algorithm #16. **(see figure 16-25)**

14) As before, we'll start by entering in EG values so as to approximate the contour of the sound we wish to create. Therefore, enter in the following EG values for operator 1:

Rs	R1	R2	R3	R4	L1	L2	L3	L4
1	96	99	64	99	99	99	0	0

This creates a very percussive envelope which looks like **figure 16-26.**

15) The mallet sound is basically unpitched, but is composed mostly of inharmonic frequencies. Therefore, raise the frequency value of

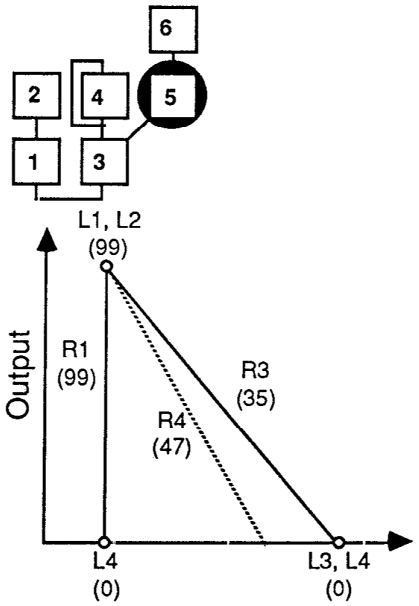


Figure 16-24

Algorithm #16:

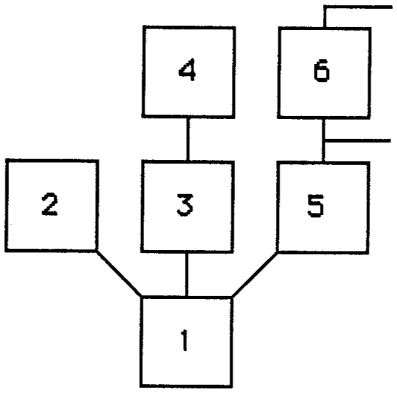


Figure 16-25

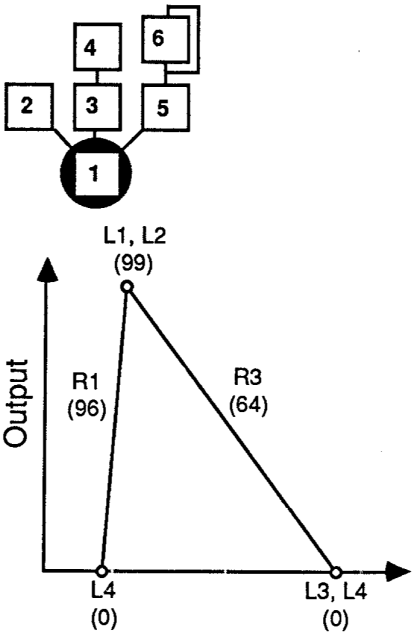


Figure 16-26

operator 1 (leave it in ratio mode) to 5.00. Now let's add in the contributions from modulators 2, 3, and 5. Set their output levels at 41, 56, and 64, respectively. Leave them all also in ratio mode, but change their frequency numbers as follows: operator 2 = 0.87 (producing a few low inharmonics); operator 3 = 4.17 (for some mid-range ones); and operator 5 = 12.87 (for some very high ones). We won't be needing any stacks in this sound, so leave the output levels of operators 4 and 6 at their default values of 0.

16) Play a few notes and listen (Audio Cue 90I). This sound is nearly there - the only thing it's lacking, in fact, is some dynamic control over both volume and timbre. This will be an important component since striking a metal bar with greater or lesser force will induce very different volumes and timbres. Therefore, set the Velocity sensitivity for operator 1 to a new value of 7; that of operators 2 and 3 to 2; and that of operator 5 to 5. Play a few notes at different degrees of force and listen (Audio Cue 90J).

17) We're once again at the point where we really need to hear this component along with the sound we intend to blend it with. Therefore, name and store this sound somewhere in your internal or RAM4 cartridge memory (again, use the Recall Edit procedure if you're not sure where you want to put it), and then put your DX7II into Dual voice mode. Assign this newly created voice to voice A, and assign the "VibraPhone" preset (ROM cartridge, bank 1, slot 18) to voice B. Make sure the "A/B" status light is off, indicating that we will be viewing voice A only (the newly created mallet sound) when we return to edit mode.

18) Play a few notes at varying degrees of force and listen (Audio Cue 90K). Note that the mallet sound is much too loud, relative to the body of the sound being produced by "VibraPhone". Therefore, press edit switch 10, followed by edit switch 1, and adjust the output level of operator 1 by ear until it blends in a way that sounds right, listening carefully as you do so (Audio Cue 90L). You should find that an operator 1 output level of 84 sounds about right.

19) Experiment by continuing to tweak and fine-tune this component until it sounds good to your ears and produces a satisfying blend with voice B. When you have adjusted it to your satisfaction, store it back to the same memory slot, thus updating your data. Then create a performance memory out of this voice and "VibraPhone", and save it using the performance storage procedure outlined in the beginning of Chapter Fourteen.

20) Finally, experiment by calling up the mallet sound in dual mode in combination with "Warm Stg A" and listen (Audio Cue 90M). Call up the bowing sound you created in steps 1 through 12 in combination with "VibraPhone" (you may have to raise its output level somewhat by adjusting the output of operator 3, since the "VibraPhone" preset is considerably louder than "Warm Stg A") and listen (Audio Cue 90N). Experiment further by "grafting" these components onto other preset voices, and by creating other component voicings.

DX7 II·FD/D

Voice name : Bowing
Date : / /

ALGORITHM	
ALG	8
FBL	7
OSC.Sync	on
Transpose	C3

L F O	
Wave	–
Speed	–
Delay	–
Mode	–
PMS	–
PMD	–
AMD	–
Sync	–

Pitch E G	
Range	1 oct
Velocity	on
R1	99
R2	92
R3	99
R4	99
L1	84
L2	50
L3	50
L4	50

OSCILLATOR OP	1	2	3	4	5	6
Mode	–	–	f	f	r	r
Coarse·Fine	–	–	1.905	436.5	1.00	1.00
Detune	–	–	+0	+0	+0	+0

E G OP	1	2	3	4	5	6
RS	–	–	0	0	0	0
R1	–	–	53	82	99	99
R2	–	–	57	99	99	99
R3	–	–	58	99	35	99
R4	–	–	60	57	47	99
L1	–	–	99	99	99	99
L2	–	–	47	99	99	99
L3	–	–	0	99	0	99
L4	–	–	0	0	0	0

Output Level OP	1	2	3	4	5	6
Scaling mode	n	n	n	n	n	n
Output Level	0	0	80	49	74	82
LD	–	–	0	0	30	0
LC	–	–	-lin	-lin	-exp	-lin
BP	–	–	C3	C3	C6	C3
RC	–	–	-lin	-lin	-lin	-lin
RD	–	–	0	0	0	0

Sensitivity OP	1	2	3	4	5	6
Velocity	–	–	3	1	0	0
AMS	–	–	0	0	0	0

Key mode	
Key assign mode	poly
Unison detune	–

Pitch Bend	
Range	2
Step	0
Mode	normal

Portamento	
Mode	retain
Step	0
Time	0
Random pitch S.	2

Modulation Wheel	
P. MOD	0
A. MOD	0
EG. B	0

Breath Control	
P. MOD	0
A. MOD	0
EG. B	0
P. Bias	+0

After Touch	
P. MOD	0
A. MOD	0
EG. B	0
P. Bias	+0

Foot control 1	
P. MOD	0
A. MOD	0
EG. B	0
P. Bias	+0

Foot control 2	
P. MOD	0
A. MOD	0
EG. B	0
P. Bias	+0

MIDI IN control	
P.MOD	0
A. MOD	0
EG. B	0
P. Bias	+0

DX7 II-FD/D

Voice name : Mallet
Date : / /

ALGORITHM	
ALG	16
FBL	0
OSC.Sync	on
Transpose	C3
L F O	
Wave	-
Speed	-
Delay	-
Mode	-
PMS	-
PMD	-
AMD	-
Sync	-
Pitch E G	
Range	-
Velocity	-
RS	-
R1	-
R2	-
R3	-
R4	-
L1	-
L2	-
L3	-
L4	-

OSCILLATOR OP	1	2	3	4	5	6
Mode	r	r	r	-	r	-
Coarse·Fine	5.00	0.87	4.17	-	12.87	-
Detune	+0	+0	+0	-	+0	-
E G OP	1	2	3	4	5	6
RS	1	0	0	-	0	-
R1	96	99	99	-	99	-
R2	99	99	99	-	99	-
R3	64	99	99	-	99	-
R4	99	99	99	-	99	-
L1	99	99	99	-	99	-
L2	99	99	99	-	99	-
L3	99	99	99	-	99	-
L4	0	0	0	-	0	-
Output Level OP	1	2	3	4	5	6
Scaling mode	n	n	n	-	n	-
Output Level	84	41	56	-	64	-
LD	0	0	0	-	0	-
LC	-lin	-lin	-lin	-	-lin	-
BP	C3	C3	C3	-	C3	-
RC	-lin	-lin	-lin	-	-lin	-
RD	0	0	0	-	0	-
Sensitivity OP	1	2	3	4	5	6
Velocity	7	2	2	-	5	-
AMS	0	0	0	0	0	0

Key mode	
Key assign mode	poly
Unison detune	-
Pitch Bend	
Range	2
Step	0
Mode	normal
Portamento	
Mode	retain
Step	0
Time	0
Random pitch S.	2
Modulation Wheel	
P. MOD	0
A. MOD	0
EG. B	0
Breath Control	
P. MOD	0
A. MOD	0
EG. B	0
P. Bias	+0
After Touch	
P. MOD	0
A. MOD	0
EG. B	0
P. Bias	+0

Foot control 1	
P. MOD	0
A. MOD	0
EG. B	0
P. Bias	+0
Foot control 2	
P. MOD	0
A. MOD	0
EG. B	0
P. Bias	+0
MIDI IN control	
P.MOD	0
A. MOD	0
EG. B	0
P. Bias	+0

Finally, let me leave you now with three original performance memories, along with the patch charts for the voices in use. This one's called "Two-Note Power", and, although the total polyphony is only two notes at a time, I think you'll agree that the incredibly strong analogish sound you'll hear more than compensates!

DX7 II·FD/D

Performance name : Two-Note Power

	A		B	
Voice mode	dual			
Voice No(name)	INT 2 ana-power		INT 2 ana-power	
Total volume	92			
Balance	+0			
Dual detune	2			
Split point	-			
Sustain foot switch	on		on	
Foot switch (portamento)RNG	on		on	
Continuous slider 1 (Dual detune)	-		-	
Continuous slider 2 (op 1 osc. detune)	on		on	
Micro tuning table select ( ) Key=	off		off	
EG forced damping	off			
Note shift	+0		+0	
PAN mode	1			
PAN range	20			
PAN select	Velocity			
PAN EG	R1 99	R2 99	R3 99	R4 99
	L1 50	L2 50	L3 50	L4 50

DX7 II-FD/D

Voice name : INT 2 ana-power
Date : / /

ALGORITHM	
ALG	23
FBL	6
OSC.Sync	on
Transpose	C3
L F O	
Wave	sine
Speed	39
Delay	78
Mode	multi
PMS	1
PMD	6
AMD	0
Sync	off
Pitch E G	
Range	8 oct
Velocity	off
RS	0
R1	99
R2	99
R3	99
R4	99
L1	50
L2	50
L3	50
L4	50

OSCILLATOR OP	1	2	3	4	5	6
Mode	r	r	r	r	r	r
Coarse•Fine	1.00	0.50	0.50	0.50	0.50	0.50
Detune	+0	-1	+0	+1	+0	+7
E G OP	1	2	3	4	5	6
RS	0	0	0	0	0	7
R1	99	99	99	99	99	99
R2	39	39	39	39	39	39
R3	32	32	32	32	32	32
R4	49	42	35	38	39	13
L1	99	99	99	99	99	99
L2	98	98	98	98	98	98
L3	99	99	99	99	99	99
L4	0	0	0	0	0	0
Output Level OP	1	2	3	4	5	6
Scaling mode	n	n	n	n	n	n
Output Level	99	84	61	99	99	88
LD	0	0	0	0	0	0
LC	-lin	-lin	-lin	-lin	-lin	-lin
BP	A-1	B3	C#4	B3	A#3	C3
RC	-exp	+lin	+lin	+lin	+lin	-lin
RD	38	38	38	38	38	0
Sensitivity OP	1	2	3	4	5	6
Velocity	0	0	0	0	0	0
AMS	0	0	0	0	0	0

Key mode	
Key assign mode	uni poly
Unison detune	3
Pitch Bend	
Range	2
Step	0
Mode	normal
Portamento	
Mode	retain
Step	0
Time	0
Random pitch S.	0
Modulation Wheel	
P. MOD	0
A. MOD	0
EG. B	0
Breath Control	
P. MOD	0
A. MOD	0
EG. B	0
P. Bias	+0
After Touch	
P. MOD	0
A. MOD	0
EG. B	0
P. Bias	+0

Foot control 1	
P. MOD	0
A. MOD	0
EG. B	0
P. Bias	+0
Foot control 2	
P. MOD	0
A. MOD	0
EG. B	0
P. Bias	+0
MIDI IN control	
P.MOD	0
A. MOD	0
EG. B	0
P. Bias	+0

DX7 II·FD/D

Performance name : Flying Saucer

	A		B	
Voice mode	single			
Voice No(name)	INT 18 U--F--0			
Total volume	99			
Balance	-			
Dual detune	-			
Split point	-			
Sustain foot switch	on			
Foot switch (portamento)RNG	on			
Continuous slider 1 (No effect)	-			
Continuous slider 2 (No effect)	-			
Micro tuning table select ( ) Key=	off			
EG forced damping	off			
Note shift	+0			
PAN mode	-			
PAN range	0			
PAN select	-			
PAN EG	R1 14	R2 94	R3 8	R4 44
	L1 99	L2 50	L3 50	L4 0

DX7 II-FD/D

Voice name : INT 18 U--F--0
Date : / /

ALGORITHM	
ALG	1
FBL	7
OSC.Sync	on
Transpose	C3
L F O	
Wave	-
Speed	-
Delay	-
Mode	-
PMS	-
PMD	-
AMD	-
Sync	-
Pitch E G	
Range	-
Velocity	-
RS	-
R1	-
R2	-
R3	-
R4	-
L1	-
L2	-
L3	-
L4	-

OSCILLATOR	OP	1	2	3	4	5	6
Mode		r	-	f	f	f	r
Coarse·Fine		9.00	-	10.00H	10.00H	1000	26.00
Detune		+0	-	+0	+0	+0	+0
E G	OP	1	2	3	4	5	6
RS		0	-	0	0	0	0
R1		14	-	30	99	12	10
R2		99	-	47	99	99	99
R3		99	-	3	6	57	57
R4		27	-	32	7	3	7
L1		99	-	99	99	99	99
L2		99	-	99	99	99	99
L3		99	-	0	0	0	0
L4		0	-	0	0	0	0
Output Level	OP	1	2	3	4	5	6
Scaling mode		n	-	n	n	n	n
Output Level		99	0	99	99	99	99
LD		0	-	0	0	0	0
LC		-lin	-	-lin	-lin	-lin	-lin
BP		C3	-	C3	C3	C3	C3
RC		-lin	-	-lin	-lin	-lin	-lin
RD		0	-	0	0	0	0
Sensitivity	OP	1	2	3	4	5	6
Velocity		0	-	0	0	0	0
AMS		0	-	0	0	0	0

Key mode	
Key assign mode	poly
Unison detune	-
Pitch Bend	
Range	12
Step	0
Mode	normal
Portamento	
Mode	retain
Step	0
Time	0
Random pitch S.	0
Modulation Wheel	
P. MOD	0
A. MOD	0
EG. B	0
Breath Control	
P. MOD	0
A. MOD	0
EG. B	0
P. Bias	+0
After Touch	
P. MOD	0
A. MOD	0
EG. B	0
P. Bias	+0

Foot control 1	
P. MOD	0
A. MOD	0
EG. B	0
P. Bias	+0
Foot control 2	
P. MOD	0
A. MOD	0
EG. B	0
P. Bias	+0
MIDI IN control	
P.MOD	0
A. MOD	0
EG. B	0
P. Bias	+0

And, last but not least, this performance memory creates a brilliant sound for use with rapid arpeggiations - and, as its name implies - with the use of rapid, sequenced patterns. Of particular note here is the fact that one of the two voices in use is essentially just the initialized default, with only a few changes made. It serves to heighten the effect of the other voice greatly, though - especially when played in stereo (with the PAN switch on).

DX7 II-FD/D

Performance name : Plug A Sequencer In

	A		B	
Voice mode	dual			
Voice No(name)	INT 54 sequenceme		INT 63 nearlyINIT	
Total volume	99			
Balance	+0			
Dual detune	0			
Split point	-			
Sustain foot switch	on		on	
Foot switch 6 (soft)RNG	on		on	
Continuous slider 1 (No effect)	-		-	
Continuous slider 2 (No effect)	-		-	
Micro tuning table select ( ) Key=	off		off	
EG forced damping	off			
Note shift	+0		+0	
PAN mode	1			
PAN range	0			
PAN select	-			
PAN EG	R1 99	R2 99	R3 99	R4 99
	L1 50	L2 50	L3 50	L4 50

DX7 II·FD/D

Voice name : INT 54 sequenceme
Date : / /

ALGORITHM	
ALG	15
FBL	5
OSC.Sync	on
Transpose	C3
L F O	
Wave	sine
Speed	25
Delay	0
Mode	single
PMS	2
PMD	10
AMD	0
Sync	off
Pitch E G	
Range	8 oct
Velocity	off
R S	0
R 1	99
R2	99
R3	99
R4	99
L 1	50
L2	50
L3	50
L4	50

OSCILLATOR OP	1	2	3	4	5	6
Mode	r	r	r	r	r	f
Coarse·Fine	2.00	1.50	1.00	8.00	0.62	309.0H
Detune	+0	+0	+0	+7	+0	+0
E G OP	1	2	3	4	5	6
RS	4	2	3	3	3	3
R1	99	99	99	99	99	99
R2	40	19	30	44	40	49
R3	33	20	35	50	38	28
R4	38	9	42	21	0	12
L1	99	99	99	91	91	91
L2	92	87	92	82	82	82
L3	0	0	0	0	0	0
L4	0	0	0	0	0	0
Output Level OP	1	2	3	4	5	6
Scaling mode	n	n	n	n	n	n
Output Level	99	64	99	88	64	49
LD	0	0	0	0	0	0
LC	-lin	-lin	-lin	-lin	-lin	-lin
BP	C3	F4	C3	C3	C3	C3
RC	-lin	-lin	-lin	-lin	-lin	-lin
RD	0	33	0	0	0	0
Sensitivity OP	1	2	3	4	5	6
Velocity	0	2	3	1	0	0
AMS	0	0	0	0	0	0

Key mode	
Key assign mode	poly
Unison detune	-
Pitch Bend	
Range	2
Step	0
Mode	normal
Portamento	
Mode	retain
Step	0
Time	0
Random pitch S.	0
Modulation Wheel	
P. MOD	0
A. MOD	0
EG. B	0
Breath Control	
P. MOD	0
A. MOD	0
EG. B	0
P. Bias	+0
After Touch	
P. MOD	0
A. MOD	0
EG. B	0
P. Bias	+0

Foot control 1	
P. MOD	0
A. MOD	0
EG. B	0
P. Bias	+0
Foot control 2	
P. MOD	0
A. MOD	0
EG. B	0
P. Bias	+0
MIDI IN control	
P.MOD	0
A. MOD	0
EG. B	0
P. Bias	+0

DX7 II-FD/D

Voice name : INT 63 nearlyINIT
Date : / /

ALGORITHM	
ALG	1
FBL	0
OSC.Sync	on
Transpose	C3
L F O	
Wave	-
Speed	-
Delay	-
Mode	-
PMS	-
PMD	-
AMD	-
Sync	-
Pitch E G	
Range	-
Velocity	-
RS	-
R1	-
R2	-
R3	-
R4	-
L1	-
L2	-
L3	-
L4	-

OSCILLATOR	OP	1	2	3	4	5	6
Mode		f	-	-	-	-	-
Coarse·Fine		1.00	-	-	-	-	-
Detune		+0	-	-	-	-	-
E G	OP	1	2	3	4	5	6
RS		0	-	-	-	-	-
R1		99	-	-	-	-	-
R2		99	-	-	-	-	-
R3		99	-	-	-	-	-
R4		30	-	-	-	-	-
L1		99	-	-	-	-	-
L2		99	-	-	-	-	-
L3		0	-	-	-	-	-
L4		0	-	-	-	-	-
Output Level	OP	1	2	3	4	5	6
Scaling mode		n	-	-	-	-	-
Output Level		99	0	0	0	0	0
LD		0	-	-	-	-	-
LC		-lin	-	-	-	-	-
BP		C3	-	-	-	-	-
RC		-lin	-	-	-	-	-
RD		0	-	-	-	-	-
Sensitivity	OP	1	2	3	4	5	6
Velocity		0	-	-	-	-	-
AMS		0	-	-	-	-	-

Key mode	
Key assign mode	poly
Unison detune	-
Pitch Bend	
Range	2
Step	0
Mode	normal
Portamento	
Mode	retain
Step	0
Time	0
Random pitch S.	0
Modulation Wheel	
P. MOD	0
A. MOD	0
EG. B	0
Breath Control	
P. MOD	0
A. MOD	0
EG. B	0
P. Bias	+0
After Touch	
P. MOD	0
A. MOD	0
EG. B	0
P. Bias	+0

Foot control 1	
P. MOD	0
A. MOD	0
EG. B	0
P. Bias	+0
Foot control 2	
P. MOD	0
A. MOD	0
EG. B	0
P. Bias	+0
MIDI IN control	
P.MOD	0
A. MOD	0
EG. B	0
P. Bias	+0

Appendix A ◇
Quick Reference Guide To The DX7II Switches

SWITCH	PARAMETER	RANGE	CHAPTER #	PAGES
Poly/mono			Thirteen	268
(as left cursor)			Two	42
Pan			Fourteen	312-313
(as right cursor)			Two	42
Data entry section			Two	42
Single voice mode select			Two	42
Dual voice mode select			Two	42
Split voice mode select			Two	42
Edit mode select			Two	41
(as Compare mode)			Eight	139-141
(as Character key)			Eight	119
Store			Eight	125
(as EG and Scaling copy)			Nine	166-167
Performance play mode select			Two	35, 37
LED/LCD display			Two	30, 37-38
A/B			Two	36
1-32/33-64			Two	36
Internal memory select			Two	35
(as fractional scaling key set)			Twelve	249
(as micro tuning key set)			Fourteen	305
Cartridge memory select			Two	36
(as fractional scaling key set)			Twelve	249
(as micro tuning key set)			Fourteen	305
Edit 1 - 6				
(as op select)			Four	59
(OP SELECT/COPY)				
(as op copy)			Nine	166-167
Edit 7	Algorithm	1 - 32	Four	56-57
(ALGORITHM)	Fbl	1 - 7	Seven	112-114
	Osc. sync	on - off	Three	50-52, 75-76
	Transpose	C1 - C5	Twelve	232-233

	Voice name		Eight	118-120
Edit 8	Mode	ratio - fixed	Five	64-65
(OSCILLATOR)	Coarse	0.50 - 31.00 (ratio mode);	Five	65-66
		1.000 Hz - 1000 Hz (fixed mode)	Five	73-75
	Fine	0.50 - 61.69 (ratio mode);	Five	68-69
		1.000 Hz 9772 Hz (fixed mode)	Five	73-75
	Detune	-7 - +7	Five	69-73
Edit 9	Rs	0 - 7	Nine	164-165
(EG)	R1	0 - 99	Nine	151-157
	R2	0 - 99	Nine	161-163
	R3	0 - 99	Nine	153-156
	R4	0 - 99	Nine	152-153
	L1	0 - 99	Nine	151-152
	L2	0 - 99	Nine	172-173
	L3	0 - 99	Nine	153-156
	L4	0 - 99	Nine	156-160
Edit 10	Scaling mode	normal or fractional	Twelve	229-230
(OUTPUT LEVEL)	Level	0 - 99	Four	59-61
	Ld	0 - 99	Twelve	236
	Lc	-lin; -exp; +exp; +lin	Twelve	235
	Bp	A-1 - C8	Twelve	230- 232
	Rc	-lin; -exp; +exp; +lin	Twelve	235
	Rd	0 - 99	Twelve	236
	Ofst	-127 - +127	Twelve	249
	Note group level	0 - 255	Twelve	248-253
Edit 11	Velocity	0 - 7	Eleven	220
(SENSITIVITY)	Ams	0 - 7	Ten	192
	Pms	0 - 7	Ten	184-186
Edit 12	LFO Wave	triangle; saw down; saw up; square; sine; sample/hold	Ten	182- 183
(LFO)	LFO Speed	0 - 99	Ten	183
	LFO Delay	0 - 99	Ten	183
	Mode	single - multi	Ten	204-207
	Pmd	0 - 99	Ten	184

	Amd	0 - 99	Ten	184
	Sync	on - off	Ten	203-204
Edit 13	Rng	1/2 - 8 octs	Nine	175- 177
	Vel	on - off	Eleven	225
	Rs	0 - 7	Nine	175-177
	R1	0 - 99	Nine	175-177
	R2	0 - 99	Nine	175-177
	R3	0 - 99	Nine	175-177
	R4	0 - 99	Nine	175-177
	L1	0 - 99	Nine	175-177
	L2	0 - 99	Nine	175-177
	L3	0 - 99	Nine	175-177
	L4	0 - 99	Nine	175-177
Edit 14	Master tuning	-64 - +63	Four	54
(TUNE)	INT memory protect	on - off	Eight	124
	CRT memory protect	on - off	Eight	124
	Coarse micro tuning	C#-2 - G8	Fourteen	305-308
	Fine micro tuning	-42 - +42	Fourteen	305-308
	Note edit	C-2 - G8	Fourteen	305-308
	Recall edit Voice		Eight	120-123
	Recall edit Perf.		Fourteen	282
	Recall edit Micro tuning		Fourteen	310
	Initialize Voice		Four	54
	Initialize Perf.		Fourteen	281
Edit 15	Save		Eight	131-133
(CARTRIDGE)	Load		Eight	131-133
	Voice and Perf. bank	1 - 4	Two	42-43
	Voice and Perf. format		Eight	127-128
	Fractional SC. bank	1 - 4	Twelve	255- 256
	Fractional SC. format		Twelve	255- 256
	Micro tuning bank	1 - 4	Fourteen	311
	Micro tuning format		Fourteen	311
Edit 16	Format		Eight	135
(DISK)	Back up		Eight	135
	Free bytes	0 - 713k	Eight	135
	INT Dir		Eight	135-136
	INT Save		Eight	136-137
	INT Load		Eight	137-138
	INT Del		Eight	139
	INT Rename		Eight	139
	CRT Dir		Eight	137
	CRT Save		Eight	137

	CRT Load		Eight	138
	CRT Del		Eight	139
	CRT Rename		Eight	139
	MDR Dir		Fifteen	344
	MDR In		Fifteen	344-345
	MDR Out		Fifteen	344-346
	MDR Del		Fifteen	344
	MDR Rename		Fifteen	344
Edit 17 - 22 (ON/OFF)	Operator on/off		Four	57-58
Edit 23 (KEY MODE)	Key mode	polyphonic; monophonic; unison poly; unison mono	Thirteen	265-268
	Unison detune	0 - 7	Thirteen	265-268
Edit 24 (PITCH BEND/ PORTAMENTO)	Pitch bend range	0 - 12	Thirteen	269
	Pitch bend step	0 - 12	Thirteen	269-270
	Pitch bend mode	normal; low; high; key on	Thirteen	270
	Portamento mode	sus-key p retain or sus-key p follow in Poly key modes; full-time or fingered in Mono key modes	Thirteen	274-277
			Thirteen	276-277
	Portamento step	0 - 12	Thirteen	274
	Portamento time	0 - 99	Thirteen	271-272
	Random pitch	0 - 7	Thirteen	278
Edit 25 (BC/MW/AT)	BC Pmod	0 - 99	Ten	187
	BC Amod	0 - 99	Ten	187
	BC EG bias	0 - 99	Eleven	210-211
	BC Pitch bias	-50 - +50	Thirteen	278-279
	AT Pmod	0 - 99	Ten	187
	AT Amod	0 - 99	Ten	187
	AT EG bias	0 - 99	Eleven	210-211
	AT Pitch bias	-50 - +50	Thirteen	278-279
	MW Pmod	0 - 99	Ten	187
	MW Amod	0 - 99	Ten	187
	MW EG bias	0 - 99	Eleven	210-211
Edit 26 (FC1/FC2)	FC1 CS1	on - off	Fourteen	292
	FC1 Pmod	0 - 99	Ten	187-188
	FC1 Amod	0 - 99	Ten	187-188
	FC1 EG bias	0 - 99	Eleven	210-211
	FC1 Vol	0 - 99	Thirteen	279
	FC2 Pmod	0 - 99	Ten	187-188
	FC2 Amod	0 - 99	Ten	187-188

	FC2 EG bias	0 - 99	Eleven	210-211
	FC2 Vol	0 - 99	Thirteen	279
	MIDI IN Pmod	0 - 99	Fifteen	338
	MIDI IN Amod	0 - 99	Fifteen	338
	MIDI IN EG bias	0 - 99	Fifteen	338
	MIDI IN Vol	0 - 99	Fifteen	338
Edit 27 (FS/CS)	Sustain foot switch (A or B)	on - off	Fourteen	284-286
	FS Select	sustain; portamento; key hold; soft	Thirteen Fourteen	273-275 286-288
	FS voice A	on - off	Fourteen	286
	FS voice B	on - off	Fourteen	286
	CS1 Select	various	Fourteen	289-292
	CS1 voice A	on - off	Fourteen	289
	CS1 voice B	on - off	Fourteen	289
	CS2 Select	various	Fourteen	289-292
	CS2 voice A	on - off	Fourteen	289
	CS2 voice B	on - off	Fourteen	289
Edit 28 (VOICE MODE)	Voice mode	single; dual; split	Fourteen	292-293
	Total volume	0 - 99	Fourteen	293
	Balance	-50 - +50	Fourteen	293
	Dual detune	0 - 7	Fourteen	293-294
	Split point	C-2 - G8	Fourteen	294
Edit 29 (MICRO TUNING)	Table select	various	Fourteen	299- 304
	Voice A	on - off	Fourteen	299
	Voice B	on - off	Fourteen	299
	EG forced damp	on - off	Fourteen	295-297
	Note shift A	-24 - +24	Fourteen	295
	Note shift B	-24 - +24	Fourteen	295
	Performance name		Fourteen	295
Edit 30 (PAN)	Pan Mode	0 - 3	Fourteen	314-316
	Pan Range	0 - 99	Fourteen	317
	Pan Select	LFO;Velocity; Note Number	Fourteen	316-317
	Pan EG R1	0 - 99	Fourteen	317, 321-322
	Pan EG R2	0 - 99	Fourteen	317, 321-322
	Pan EG R3	0 - 99	Fourteen	317, 321-322
	Pan EG R4	0 - 99	Fourteen	317, 321-322
	Pan EG L1	0 - 99	Fourteen	317, 321-322
	Pan EG L2	0 - 99	Fourteen	317, 321-322
	Pan EG L3	0 - 99	Fourteen	317, 321-322
	Pan EG L4	0 - 99	Fourteen	317, 321-322
Edit 31 (MIDI 1)	Trns ch	1 - 16	Fifteen	337
	Rcv ch A	1 - 16	Fifteen	337-338
	Rcv ch B	1 - 16	Fifteen	337-338
	Omni	on - off	Fifteen	337

	MIDI IN A	11 - 31	Fifteen	338- 339
	MIDI IN B	11 - 31	Fifteen	338- 339
	CS1	5 - 31	Fifteen	339-340
	CS2	5 - 31	Fifteen	339-340
	Note on/off	on - off	Fifteen	340
	PC trns mode	off; normal; programmable	Fifteen	341
	Local	on - off	Fifteen	340
	Program change trns	001 - 128	Fifteen	341
Edit 32 343	Device number	1 - 16, off	Fifteen	
(MIDI 2)	Receive block	1-32; 33-64	Fifteen	343
	Voice edit buf		Fifteen	343
	Voice 1-32		Fifteen	343
	Voice 33-64		Fifteen	343
	Perf. edit buf		Fifteen	343
	Perf. INT		Fifteen	343
	Micro tuning edit buf		Fifteen	344
	Micro tuning INT		Fifteen	344
	Micro tuning CRT		Fifteen	344
	System setup		Fifteen	344

Appendix B ◇ Voice and Performance Initialization Defaults For The DX7II

The following is a complete summary of all the DX7II default values that are inserted upon entering the voice and/or performance memory initialization procedure (see Chapters Four and Fourteen for more details):

I. Voice initialization defaults:

SWITCH #	PARAMETER	DEFAULT VALUE
Edit 7 (ALGORITHM)	Algorithm (Alg)	1
	Feedback loop (Fbl)	0
	Osc. sync	on
	Transpose	mid C = C3
	Voice name	INIT VOICE
Edit 8 (OSCILLATOR)	Mode (all operators)	ratio
	Coarse/Fine (all operators)	1.00
	Detune (all operators)	+0
Edit 9 (EG)	Rate scaling (Rs) (all operators)	0
	Rate 1 (R1) (all operators)	99
	Rate 2 (R2) (all operators)	99
	Rate 3 (R3) (all operators)	99
	Rate 4 (R4) (all operators)	99
	Level 1 (L1) (all operators)	99
	Level 2 (L2) (all operators)	99
	Level 3 (L3) (all operators)	99
	Level 4 (L4) (all operators)	0
Edit 10 (OUTPUT LEVEL)	Scaling mode (all operators)	normal
	Level (operator 1)	99
	Level (operators 2 through 6)	0
	Left depth (Ld) (all operators)	0
	Left curve (Lc) (all operators)	- lin
	Break point (Bp) (all operators)	C3
	Right curve (Rc) (all operators)	- lin
	Right depth (Rd) (all operators)	0
Edit 11 (SENSITIVITY)	Velocity (all operators)	0
	Ams (all operators)	0
	Pms	3
Edit 12 (LFO)	Wave	triangle
	Speed	35
	Delay	0

	Mode	single
	Pitch modulation depth (Pmd)	0
	Amplitude modulation depth (Amd)	0
	Sync	on
Edit 13 (PITCH EG)	Range (Rng)	8 oct
	Velocity (Vel)	off
	Rate scaling (Rs)	0
	Rate 1 (R1)	99
	Rate 2 (R2)	99
	Rate 3 (R3)	99
	Rate 4 (R4)	99
	Level 1 (L1)	50
	Level 2 (L2)	50
	Level 3 (L3)	50
	Level 4 (L4)	50
Edit 23 (KEY MODE)	Key mode	polyphonic
Edit 24 (PITCH BEND/ PORTAMENTO)	Pitch bend Range	2
	Pitch bend Step	0
	Pitch bend Mode	normal
	Portamento Mode	sus-ke p retain
	Portamento Step	0
	Portamento Time	0
	Random pitch Depth	0
Edit 25 (BC/MW/AT)	Breath control Pmod	0
	Breath control Amod	0
	Breath control EGbias	0
	Breath control Pbias	+0
	After touch Pmod	0
	After touch Amod	0
	After touch EGbias	0
	After touch Pbias	+0
	Modulation wheel Pmod	0
	Modulation wheel Amod	0
	Modulation wheel EGbias	0
Edit 26 (FC1/FC2)	Foot control 1 CS1	off
	Foot control 1 Pmod	0
	Foot control 1 Amod	0
	Foot control 1 EGbias	0
	Foot control 1 Volume	0
	Foot control 2 Pmod	0
	Foot control 2 Amod	0
	Foot control 2 EGbias	0
	Foot control 2 Volume	0
	MIDI IN control Pmod	0
	MIDI IN control Amod	0
	MIDI IN control EGbias	0
	MIDI IN control Volume	0

II. Performance memory initialization defaults:

SWITCH #	PARAMETER	DEFAULT VALUE
Edit 27 (FS/CS)	Sustain foot switch voice A	on
	Sustain foot switch voice B	on
	Foot switch Select	portamento
	Foot switch voice A	on
	Foot switch voice B	on
	CS1 Select	No Effect
	CS2 Select	No Effect
Edit 28 (VOICE MODE)	Voice mode	dual
	Voice A	INT 1
	Voice B	INT 1
	Total volume	99
	Balance	+0
	Dual detune	0
Edit 29 (MICRO TUNE)	Micro tuning Table select	Equal temperament
	Voice A	off
	Voice B	off
	EG forced damp	off
	Note shift voice A	+0
	Note shift voice B	+0
	Performance name	INIT PERF
Edit 30 (PAN)	Pan Mode	1
	Pan Range	0
	Pan Select	LFO
	Pan EG Rate 1 (R1)	99
	Pan EG Rate 2 (R2)	99
	Pan EG Rate 3 (R3)	99
	Pan EG Rate 4 (R4)	99
	Pan EG Level 1 (L1)	50
	Pan EG Level 2 (L2)	50
	Pan EG Level 3 (L3)	50
	Pan EG Level 4 (L4)	50

Note that the parameters of edit switches 14, 15, 16, 31, and 32 do not change following the initialization process; they will be altered only when loading in new system setup data from cartridge or disk (see Chapter Fifteen for more details). The following is a summary of DX7II power-up defaults; these are the only two parameters that reset themselves whenever turning on the power to the DX7II:

SWITCH #	PARAMETER	DEFAULT VALUE
Edit 14	Internal (INT) memory protection	on
Edit 14	Cartridge (CRT) memory protection	on

Appendix C ◇ DX7II Compatibility With The DX7

The DX7II was designed to be completely upwardly compatible with the original DX7. This means that all existing DX7 data - in the form of voices and banks of voices - can be transferred to the DX7II, either via MIDI or via the special ADP-1 cartridge adapter, and played successfully by the DX7II. The key word here, however is "upwardly". This means that DX7II voice data CANNOT be transferred to a DX7, or to any DX7 data storage devices such as RAM cartridges - meaning that your DX7 RAM cartridges become, in effect, ROM cartridges when used with the DX7II. In general, Yamaha has succeeded in preserving compatibility, and the DX7 owner will benefit from transferring his or her voices to a DX7II, since the higher-speed processing and increased bit resolution of the DX7II will ensure that these older voices will be reproduced with higher fidelity and with much less digital "fizz", or aliasing noise. There are, however, a few minor differences between the two instruments that the veteran DX7 user should be aware of. Several of these simply have to do with slightly different responses when in edit mode, while others affect voices themselves when transferred from a DX7 to the DX7II.

The following is a summary of the different responses found when using the DX7II in edit mode:

1) Cursor controls: the major difference, of course, is the use of cursor controls to select one of a number of parameters typically shown in the LCD. Whenever selecting a new edit switch, the cursor will automatically be positioned over the last parameter selected the last time that edit switch was pressed. This feature will not be bypassed, even after initializing or using the Recall edit procedure. As with the original DX7, the most recent edit switch called up when last in edit mode will be the first one called up when re-entering edit mode.

2) Real-time operator envelope changes: when making changes to the levels of operator EGs in the DX7II, the voice you are hearing will change accordingly as you are making these alterations. This is particularly noticeable when working with Level 3 or Level 4.

3) Voice recall edit procedure: when recalling the contents of the edit recall buffer, the LED voice number will not change to reflect the voice number of the data that was originally altered. However, a decimal point will appear after the current voice number in order to indicate that new data has been placed into the edit buffer.

4) Voice initialization procedure: unlike the original DX7, initializing a DX7II voice will not result in a decimal point automatically appearing next to the LED voice number. The general rule here seems to be that if the decimal point was there at the time you initialize, it stays there, and if it wasn't, it still won't be there. Also, the six operators will not automatically all be turned on following the initialization procedure: whichever operators were on prior to the procedure will stay on, and whichever were off will stay off.

5) EG copy - as noted in Chapter Nine, this procedure now not only copies EG data, but rate scaling, output level, and keyboard level scaling as well, thus ensuring that the envelopes will be precisely the same from operator to operator. This command is available from either edit switch 9 (EG) or edit switch 10 (OUTPUT LEVEL).

6) Data storage - unlike the DX7, the storage procedure is not completed with the act of pressing the main switch corresponding to the internal or cartridge slot in which you wish to store your voice or performance memory; instead, you must hold the "store" switch down and press the "yes" button in order to complete the procedure. The LCD will prompt you with the data slot number you have selected, and will read "Completed!" when the process is done. If you do not see the "Completed!" prompt, you can assume that your data has NOT been stored correctly. The storage procedure can be aborted simply by releasing the "store" switch.

The following is a summary of the parameters that may need to be re-adjusted when using DX7 voices in the DX7II:

1) Lack of function controls: the DX7II has no function controls, as they existed in the DX7, anyway. Instead, all of these parameters - which in the DX7 were not voice specific and could not be stored with the voice data - are in the DX7II ordinary voice edit parameters - meaning that they are voice-specific and are in fact stored along with all the other voice data in the usual way. No function data is ever transmitted by a DX7 via MIDI, nor can it be stored in a RAM cartridge. Therefore, when shipping DX7 voices to a DX7II, you will find none of these parameters are transferred. Instead, the equivalent DX7II voice edit parameters will instead simply revert to their initialization defaults (see Appendix B). This means that you will have to manually set up any controller routings, pitch bend and/or portamento settings, and poly/mono settings in the DX7II. It means a bit more work, but the bonus is that these settings will actually get stored along with the voice when you finally save it as a DX7II voice.

2) DX7II edit parameters which don't exist in the DX7: these include things like Pitch EG range and velocity sensitivity, fractional scalings, LFO modes, key modes, pitch bend mode, random pitch, pitch bias, MIDI IN controls, and foot controller 2. All of these will simply revert to their initialization defaults (as shown in Appendix B) when transferring DX7 voices to the DX7II, and will need to be adjusted manually if you wish to use any of them.

3) Amplitude modulation sensitivity: in the DX7, this parameter has a range of 0 to 3, with "3" representing maximum sensitivity to both LFO and EG bias modulations. In the DX7II, the range of this parameter has been increased to 0 to 7. When transferring DX7 voices to the DX7II, any operators having an amplitude modulation sensitivity (Ams) of 3, will still be given a value of 3, but "3" no longer represents maximum sensitivity - "7" does. In general, this will be important when working with DX7 sounds that use the EG bias modulation control. As a general rule of thumb, any operators which had in the DX7 an Ams of

"3" should have this value manually changed in the DX7II to "7". Similarly, operators with Ams values of "2" should have this value changed in the DX7II to "4", and any with Ams values of "1" should be changed in the DX7II to "2". Operators having Ams values of 0 can be left as is, since they will have no sensitivity in either instrument.

4) Pitch EG differences: this is perhaps the most significant area of incompatibility, but will be of interest to you only if you are transferring DX7 voices which use very slow pitch EG rates of movement. The problem stems from the new "Range" parameter in the DX7II pitch EG, which allows you to determine the maximum amount of pitch change that the envelope will generate at any given time. The default "Range" value is "8 octaves", and it is this default which will be assumed when DX7 voices are transferred into the DX7II. The DX7 pitch envelope is fixed at a range of 8 octaves, anyway, so at first glance it would seem as if this should cause no problem. However, the absolute speeds of movement within the pitch EG change (even though the rate values remain the same) when the Range is altered - just as the operator EG speeds change as output level is altered (as the envelope "squeezes down", it takes less time to get from level to level - see Chapter Nine). The problem is that a DX7 pitch EG rate of, say, 50, is the same speed as compared with a DX7II pitch EG rate of 50 only when the range is *one octave*, not eight. Therefore, you will find that all DX7 pitch EG rates are considerably faster when the sound is exported to the DX7II. This will have little audible effect in most instances, but will be quite apparent, as noted above, for pitch EG rates which are slow. The solution to this problem is to simply alter the pitch EG Range value to either 1 octave or 2 octaves after the DX7 voice is in the DX7II, and to then simply increase the rate values until your ear tells you that the absolute speeds of pitch change sound the same as they did in the DX7.

5) Delay effects incurred by operator envelope generators: In Chapter Nine, a technique for introducing delayed attacks into sounds by using operator EG levels below 20 is explored. The logic for doing this exists in the DX7 as well, but is slightly different. In the DX7, this effect can be generated by using adjacent level values which are both below 20 but equal to one another. In that case, the rate of movement between these two levels - a rate of movement which is normally meaningless when traveling between two levels which are the same - acts as a digital delay line. In the DX7II, this "flaw" in the logic has been "repaired", and you will find that no delays are incurred when traveling between two EG levels which are identical. Instead, there must be a difference between two levels - even one as slight as a single increment! - in order for this effect to be generated. Several DX7 presets (including "WATERGARDEN" from the original DX7 ROM cartridge) use the technique of adjacent and equal EG levels. These sounds will have no delays - and consequently sound very different - when exported to the DX7II. In order to restore these voices back to their original effect, all you need to do is to alter at least one of the levels - even very slightly.

Appendix D ◇
The Micro Tuning Presets

The following tables, prepared by Howard Sandroff, Director of the Electronic Music Studio at the University of Chicago, show a typical octave extracted from the preset micro tunings available in the DX7II. Each key is shown, followed by its absolute Fine value. This is followed by the size of the interval itself, shown two ways. The first of these is called "YAN", which is an abbreviation for "Yamaha Absolute Number" and represents the difference in Fine increments between adjacent semitones. The second interval size is shown in cents, hundredths of a semitone. Also included in non-equally tempered tunings is the deviation from equal temperament, in "YAN" and in cents.

The author is indebted to Mr. Sandroff and to the Yamaha Music Corporation for their permission to reproduce these charts in their entirety.

I. Equal temperament

NOTE NUMBER	ABSOLUTE FINE VALUE	INTERVAL SIZE	
		YAN	CENTS
C3	5034	86	100.781250
C#3	5120	85	99.609375
D3	5205	85	99.609375
D#3	5290	86	100.781250
E3	5376	85	99.609375
F3	5461	85	99.609375
F#3	5546	86	100.781250
G3	5632	85	99.609375
G#3	5717	85	99.609375
A3	5802	86	100.781250
A#3	5888	85	99.609375
B3	5973	85	99.609375
C4	6058		

II. Pure Major, Key of C:

NOTE NUMBER	ABSOLUTE FINE VALUE	INTERVAL SIZE		DEVIATION	
		YAN	CENTS	YAN	CENTS
C3	5048	60	70.312500	+14	+16.406250
C#3	5108	114	133.593750	-12	-14.062500
D3	5222	95	111.328125	+17	+19.921875
D#3	5317	60	70.312500	+27	+31.640625
E3	5377	96	112.500000	+ 1	+ 1.171875
F3	5473	60	70.312500	+12	+14.062500
F#3	5533	114	133.593750	-13	-15.234375
G3	5647	60	70.312500	+15	+17.578125
G#3	5707	95	111.328125	-10	-11.718750
A3	5802	114	133.593750	0	0.000000
A#3	5916	60	70.312500	+28	+32.812500
B3	5976	96	112.500000	+ 3	+ 3.515625
C4	6072			+14	+16.406250

III. Pure Minor, Key of C:

NOTE NUMBER	ABSOLUTE FINE VALUE	INTERVAL SIZE		DEVIATION	
		YAN	CENTS	YAN	CENTS
C3	5048	114	133.593750	+14	+16.406250
C#3	5162	60	70.312500	+42	+49.218750
D3	5222	96	112.500000	+17	+19.921875
D#3	5318	60	70.312500	+28	+32.812500
E3	5378	95	111.328125	+ 2	+2.343750
F3	5473	114	133.593750	+12	+14.062500
F#3	5587	60	70.312500	+41	+48.046875
G3	5647	96	112.500000	+15	+17.578125
G#3	5743	60	70.312500	-26	-11.718750
A3	5802	114	133.593750	0	0.000000
A#3	5917	60	70.312500	+29	+33.984375
B3	5977	95	111.328125	+ 4	+4.687500
C4	6072			+14	+16.406250

IV. Mean Tone, Key of C:

NOTE NUMBER	ABSOLUTE FINE VALUE	INTERVAL SIZE		DEVIATION	
		YAN	CENTS	YAN	CENTS
C3	5043	65	76.171875	+ 9	+10.546875
C#3	5108	100	117.187500	-12	-14.062500
D3	5208	100	117.187500	+ 3	+ 3.515625
D#3	5308	65	76.171875	+18	+21.093750
E3	5373	100	117.187500	- 3	- 3.515625
F3	5473	64	75.000000	+12	+14.062500
F#3	5537	100	117.187500	- 9	-15.234375
G3	5637	65	76.171875	+ 5	+17.578125
G#3	5702	100	117.187500	-15	-11.718750
A3	5802	100	117.187500	0	0.000000
A#3	5902	65	76.171875	+14	+32.812500
B3	5967	100	117.187500	- 6	+ 3.515625
C4	6067			+ 9	+16.406250

V. Pythagorean, Key of C:

NOTE NUMBER	ABSOLUTE FINE VALUE	INTERVAL SIZE		DEVIATION	
		YAN	CENTS	YAN	CENTS
C3	5029	96	112.500000	- 5	- 5.859375
C#3	5125	78	91.406250	+ 5	+49.218750
D3	5203	77	90.234375	- 2	- 2.343750
D#3	5280	97	113.671875	-10	-11.718750
E3	5377	77	90.234375	+ 1	+ 2.343750
F3	5454	97	113.671875	- 7	+14.062500
F#3	5551	77	90.234375	+ 5	+48.046875
G3	5628	97	113.671875	- 4	+17.578125
G#3	5725	77	90.234375	- 8	-11.718750
A3	5802	77	90.234375	0	0.000000
A#3	5879	97	113.671875	- 9	-10.546875
B3	5976	77	90.234375	+ 3	+ 3.515625
C4	6053			- 5	+16.406250

VI. Werckmeister:

NOTE NUMBER	ABSOLUTE FINE VALUE	INTERVAL SIZE		DEVIATION	
		YAN	CENTS	YAN	CENTS
C3	5044	76	89.062500	+10	+11.718750
C#3	5120	88	103.125000	0	0.000000
D3	5208	87	101.953125	+ 3	+ 3.515625
D#3	5295	82	96.093750	+ 5	+ 5.859375
E3	5377	92	107.812500	+ 1	+ 1.171875
F3	5469	77	90.234375	+ 8	+ 9.375000
F#3	5546	92	107.812500	0	-15.234375
G3	5638	82	96.093750	+ 6	+ 7.031250
G#3	5720	82	96.093750	+ 3	+ 3.515625
A3	5802	92	107.812500	0	0.000000
A#3	5894	82	96.093750	+ 6	+ 7.031250
B3	5976	84	98.437500	+ 3	+ 3.515625
C4	6060			+ 2	+ 2.343750

VII. Kirnberger:

NOTE NUMBER	ABSOLUTE FINE VALUE	INTERVAL SIZE		DEVIATION	
		YAN	CENTS	YAN	CENTS
C3	5043	77	90.234375	+ 9	+10.546875
C#3	5120	88	103.125000	0	0.000000
D3	5208	86	100.781250	+ 3	+ 3.515625
D#3	5294	79	92.578125	+ 4	+ 4.687500
E3	5373	95	111.328125	- 3	- 3.515625
F3	5468	79	92.578125	+ 7	+ 8.203125
F#3	5547	90	105.468750	+ 1	+ 1.171875
G3	5637	82	96.093750	+ 5	+ 7.031250
G#3	5719	83	97.265625	+ 2	+ 3.515625
A3	5802	91	106.640625	0	0.000000
A#3	5893	79	92.578125	+ 5	+ 5.859375
B3	5972	95	111.328125	- 1	- 1.171875
C4	6067			+ 9	+10.546875

VIII. Vallotti and Young:

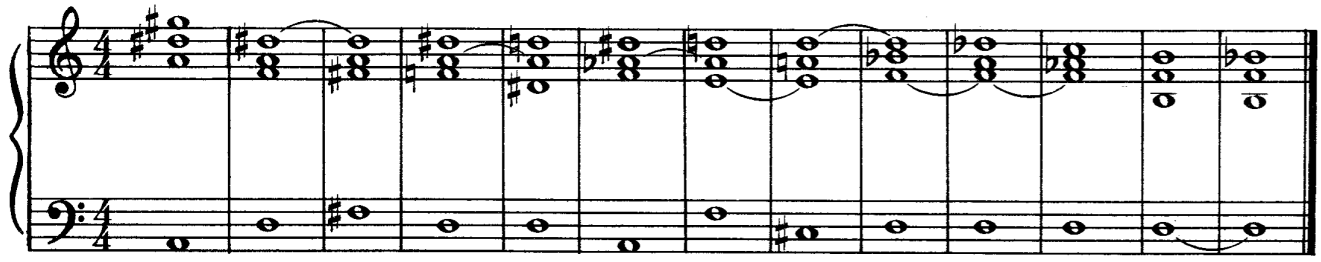
NOTE NUMBER	ABSOLUTE FINE VALUE	INTERVAL SIZE		DEVIATION	
		YAN	CENTS	YAN	CENTS
C3	5039	81	94.921875	+ 5	+ 5.859375
C#3	5120	87	101.953125	0	0.000000
D3	5207	87	101.953125	+ 2	+ 2.343750
D#3	5294	80	93.750000	+ 4	+ 4.687500
E3	5374	94	110.156250	- 2	- 2.343750
F3	5468	76	89.062500	+ 7	+ 8.203125
F#3	5544	91	106.640625	- 2	+ 0.000000
G3	5635	84	98.437500	+ 3	+ 7.031250
G#3	5719	83	97.265625	+ 2	+ 3.515625
A3	5802	91	106.640625	0	0.000000
A#3	5893	76	89.062500	+ 5	+ 7.031250
B3	5969	94	110.156250	- 4	+ 3.515625
C4	6063			+ 5	+ 2.343750

Appendix E ◇
A Fifteen-Tone Scale

The following table presents the Fine micro tuning values for an equally-tempered fifteen-tone scale (as opposed to the twelve tones we are used to hearing), developed by Mr. Howard Sandroff, Director of the Electronic Music Studio at the University of Chicago. The author is grateful to Mr. Sandroff for his kind permission to reprint this table:

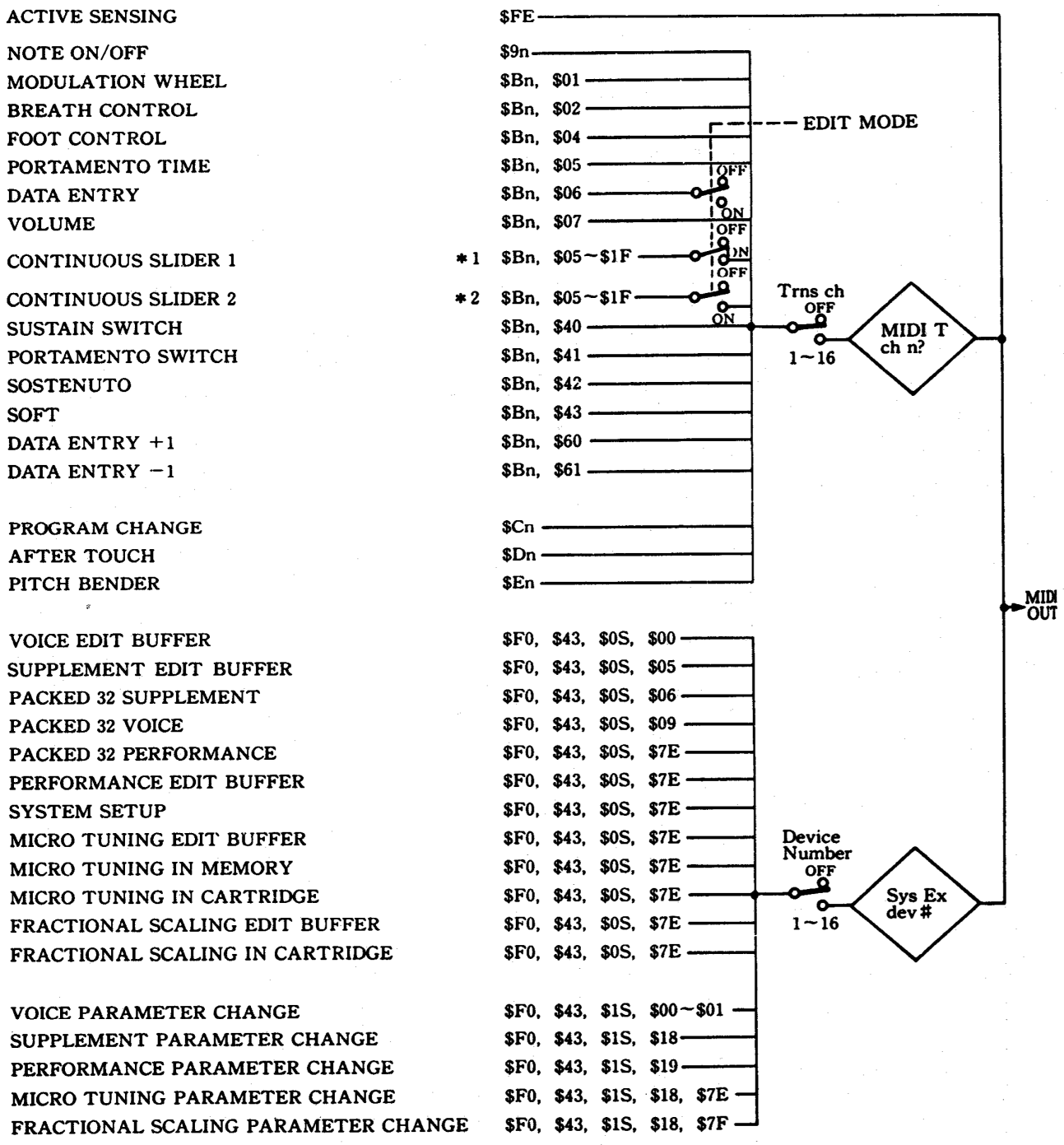
PITCH #	KEY #	VALUE	KEY #	VALUE	KEY #	VALUE	KEY #	VALUE
ONE	D#-1	2116	F#0	3140	A1	4164	C3	5188
gap (YAN)	68		68		68		68	
TWO	E-1	2184	G0	3208	A#1	4232	C#3	5256
gap (YAN)	68		68		68		68	
THREE	F-1	2252	G#0	3276	B1	4300	D3	5324
gap (YAN)	69		69		69		69	
FOUR	F#-1	2321	A0	3345	C2	4369	D#3	5393
gap (YAN)	68		68		68		68	
FIVE	G-1	2389	A#0	3413	C#2	4437	E3	5461
gap (YAN)	68		68		68		68	
SIX	G#-1	2457	B0	3481	D2	4505	F3	5529
gap (YAN)	68		68		68		68	
SEVEN	A-1	2525	C1	3549	D#2	4573	F#3	5597
gap (YAN)	69		69		69		69	
EIGHT	A#-1	2594	C#1	3618	E2	4642	G3	5666
gap (YAN)	68		68		68		68	
NINE	B-1	2662	D1	3686	F2	4710	G#3	5734
gap (YAN)	68		68		68		68	
TEN	C0	2730	D#1	3754	F#2	4778	A3	5802
gap (YAN)	68		68		68		68	
ELEVEN	C#0	2798	E1	3822	G2	4846	A#3	5870
gap (YAN)	69		69		69		69	
TWELVE	D0	2867	F1	3891	G#2	4915	B3	5939
gap (YAN)	68		68		68		68	
THIRTEEN	D#0	2935	F#1	3959	A2	4983	C4	6007
gap (YAN)	68		68		68		68	
FOURTEEN	E0	3003	G1	4027	A#2	5051	C#4	6075
gap (YAN)	68		68		68		68	
FIFTEEN	F0	3071	G#1	4095	B2	5119	D4	6143
gap (YAN)	69		69		69		69	

Mr. Sandroff recommends trying the following short musical phrase with this altered scale in order to hear some of its potential:



Appendix F ◇
The DX7II MIDI System Exclusive Code

[1] Transmission Requirements



*1 BALANCE $Bn, $08 in EDIT MODE
*2 DATA ENTRY $Bn, $06 in EDIT MODE

[2] Transmission Data

[2]-1 Channel Information

Transmission is possible only when 1~ 16 is specified as the transmission channel.

1) Channel voice message

1 Key ON/OFF

Status	1 0 0 1 n n n n	(9n)	n = channel No.
Note No.	0 k k k k k k k		k = 36(C_1)~96(C_6)
Velocity	0 v v v v v v v	(v = 0)	Key ON
	0 0 0 0 0 0 0 0	(v = 0)	Key OFF

2 Control change

Status	1 0 1 1 n n n n	(Bn)	n = channel No.
Control No.	0 c c c c c c c		
Control Value	0 v v v v v v v		

Control No.

c = 1	Modulation wheel	v = 0~127
c = 2	Breath control	v = 0~127
c = 4	Foot control	v = 0~127
c = 5	Portamento time	v = 0~127
c = 6	Data entry slider	v = 0~127
c = 7	Volume	v = 0~127
c = 5~ c = 31	Continuous slider	v = 0~127
c = 64	Sustain SW	v = 0: OFF, 127: ON
c = 65	Portamento SW	v = 0: OFF, 127: ON
c = 66	Sostenuto	v = 0: OFF, 127: ON
c = 67	Soft	v = 0: OFF, 127: ON
c = 96	Data entry +1	
c = 97	Data entry −1	

3 Program change

Status	1 1 0 0 n n n n	(Cn)	n = channel No.
Program No.	0 p p p p p p p		p = 0~63: INTERNAL p = 64~127: CARTRIDGE

4 After touch

Status	1 1 0 1 n n n n	(Dn)	n = channel No.
Value	0 v v v v v v v		v = 0~127

5 Pitch bender

Status	1 1 1 0 n n n n	(En)	n = channel No.
Value (LSB)	0 u u u u u u u		
Value (MSB)	0 v v v v v v v		
Resolution 7bit			

The transmission data are as follows:

MSB		LSB		
00000000	(00)	00000000	(00)	Min.
01000000	(40)	00000000	(00)	Mid.
01111111	(7F)	01111110	(7E)	Max.

[2]-2 System Information

1) System real time message

Active sensing

Status	1 1 1 1 1 1 1 0	(FE)

2) System exclusive message

Transmission is possible only when the device No. is set to 1~16.

1 Parameter change

Status	1 1 1 1 0 0 0 0	(F0)	
ID No.	0 1 0 0 0 0 1 1	(43)	
Substatus/ device No.	0 0 0 1 n n n n	(1n)	
Parameter group No.	0 g g g g g h h		
Parameter No.	0 p p p p p p p		
Data	0 d d d d d d d 0 d d d d d d d		Single or multiple bytes
EOX	1 1 1 1 0 1 1 1	(F7)	

There are seven parameter group Nos. and parameter Nos.

Parameter	g	h	p	No. of data byte
Voice	0	0	0~127	1
	0	1	0~28	1
Supplement Note 3)	6	0	0~73	1
Performance	6	1	0~52	1
System set-up	6	1	64~	1
Micro tuning	6	0	126	3 Note 1)
Fractional scaling	6	0	127	4 Note 2)

NOTE 1

Data bytes

0kkkkkkk	key number		total of 3 bytes
0hhhhhhh	data (high)	0-84 binary	
01111111	data (low)	0-127 binary	

NOTE 2

Data bytes

00000ppp	operator number		total of 4 byte
00kkkkkk	key group number		
0hhhhhhh	data (high)	0-1 binary	
01111111	data (low)	0-127 binary	

NOTE 3

Under the Supplement parameter change, DX7 function parameter change will be transmitted along with the above.

● **Fractional Scaling Parameter Change**

Operator number

P	Operator
0	op 6
1	op 5
2	op 4
3	op 3
4	op 2
5	op 1

Key group number

K	Key	Data
0	offset	−128 ~ 127
1	C#-2~ C-1	0 ~256
2	C#-1~D#-1	
3	E-1 ~F#-1	
4	G-1 ~ A-1	
5	A#-1~ C0	
6	C#0 ~ D#0	
7	E0 ~ F#0	
8	G0 ~ A0	
9	A#0 ~ C1	
10	C#1 ~ D#1	
11	E1 ~ F#1	
12	G1 ~ A1	
13	A#1 ~ C2	
14	C#2 ~ D#2	
15	E2 ~ F#2	
16	G2 ~ A2	
17	A#2 ~ C3	
18	C#3 ~ D#3	
19	E3 ~ F#3	
20	G3 ~ A3	
21	A#3 ~ C4	
22	C#4 ~ D#4	
23	E4 ~ F#4	
24	G4 ~ A4	
25	A#4 ~ C5	
26	C#5 ~ D#5	
27	E5 ~ F#5	
28	G5 ~ A5	
29	A#5 ~ C6	
30	C#6 ~ D#6	
31	E6 ~ F#6	
32	G6 ~ A6	
33	A#6 ~ C7	
34	C#7 ~ D#7	
35	E7 ~ F#7	
36	G7 ~ A7	
37	A#7 ~ C8	
38	C#8 ~ D#8	
39	E8 ~ F#8	
40	G8	

(Complement of 2)
(Binary)

2 Bulk data

For
- Voice edit buffer
- Supplement edit buffer
- Packed 32 supplement
- Packed 32 voice

Status	1 1 1 1 0 0 0 0	(F0)
ID No.	0 1 0 0 0 0 1 1	(43)
Substatus/ device No.	0 0 0 0 n n n n	(0n)
Format No.	0 f f f f f f f	
Byte count (MSB)	0 b b b b b b b	
Byte count (LSB)	0 b b b b b b b	
Data	0 d d d d d d d	
	↓	
	0 d d d d d d d	
Checksum	0 e e e e e e e	
EOX	1 1 1 1 0 1 1 1	(F7)

Format No.	Data	Byte count
0	Voice edit buffer	155
5	Supplement edit buffer	49
6	Packed 32 supplement	1120
9	Packed 32 voice	4096

● **When using universal Bulk Damp**

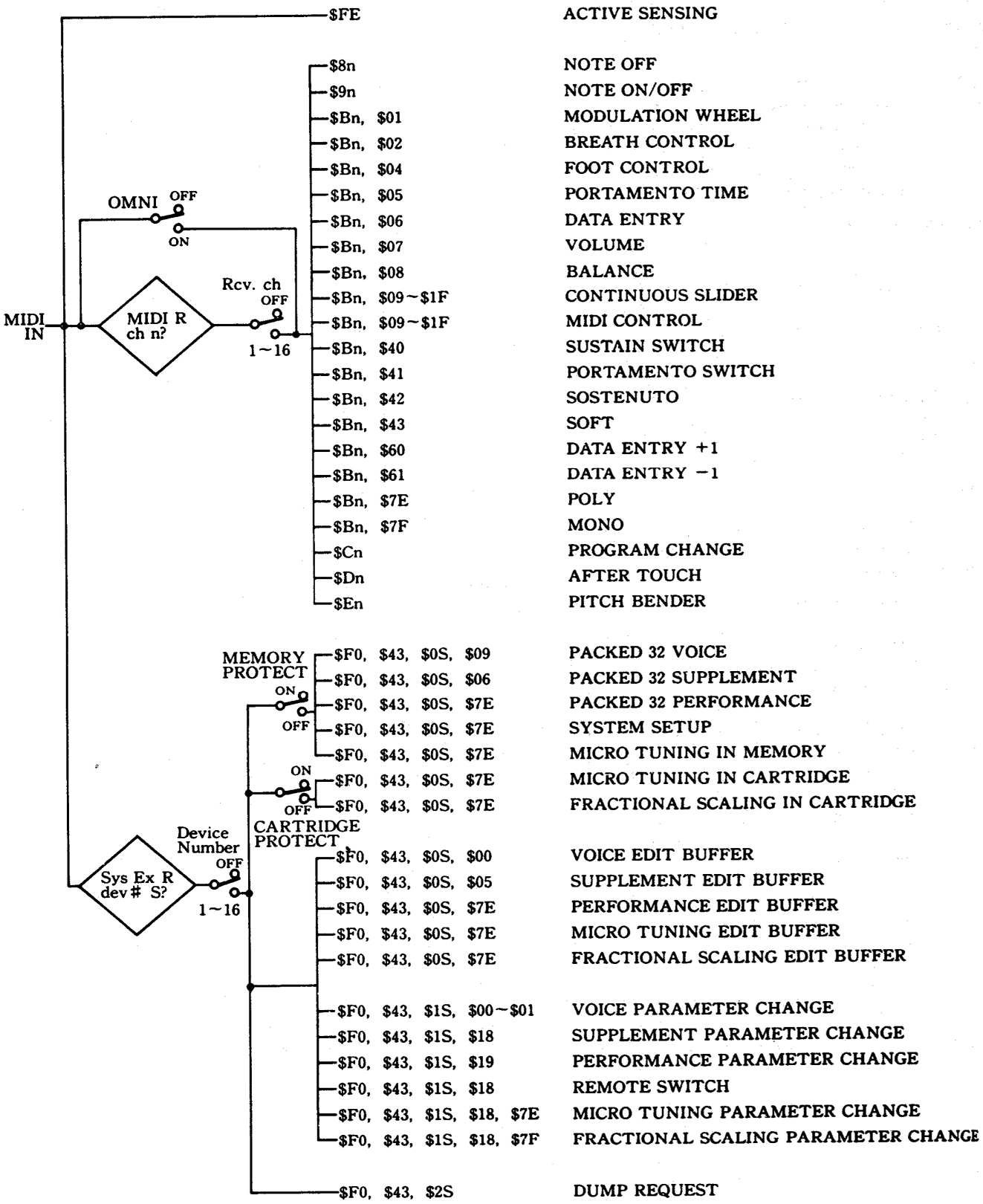
Status	1 1 1 1 0 0 0 0	(F0)
ID No.	0 1 0 0 0 0 1 1	(43)
Substatus/ device No.	0 0 0 0 n n n n	(0n)
Format No.	0 1 1 1 1 1 1 0	(7E)
Byte count (MSB)	0 b b b b b b b	
Byte count (LSB)	0 b b b b b b b	
Classification	0 a a a a a a a	ASCII 'L
name	0 a a a a a a a	'M
(4 bytes)	0 a a a a a a a	'␣
	0 a a a a a a a	'␣
Data format	0 m m m m m m m	ASCII
name (6 bytes)	↓	
	0 m m m m m m m	
Data	0 d d d d d d d	
	↓	
	0 d d d d d d d	
Checksum	0 e e e e e e e	
EOX	1 1 1 1 0 1 1 1	(F7)

Repeat group

Data	Byte count	Classification name	Data format name	No. of repeats
DX7 II Performance Edit Buffer	61	LM␣␣	8973P E	1
DX7 II Packed 32 Performance	1642	LM␣␣	9873P M	1
DX7 II System Set-up	112	LM␣␣	8973 S␣	1
Micro Tuning Edit Buffer	266	LM␣␣	MCRYE	1
Micro Tuning with Memory #x	266	LM␣␣	MCRYMx	2
Micro Tuning Cartridge	266	LM␣␣	MCRYC␣	64
Fractional Scaling Edit Buffer	502	LM␣␣	FKSYE␣	1
Fractional Scaling in Cartridge with Memory #	502	LM␣␣	FKSYC␣	32

Note 1) The x of MCRYMx is a memory No. expressed in binary form, 0 or 1.
Note 2) When the number of repeats is 64, the data group from byte count to checksum will be transmitted 64 times.

[3] Reception Requirements



[4] Reception Data

[4]-1 Channel Information

There are two types of MIDI reception channels for channel messages: A and B.

Single mode : Only A is effective
Dual mode : Only A is effective
Split mode : A, B independent
The split point function is effective when A = B, assigning A to the lower half and B to the upper half.

1) Channel voice message

1 Key OFF

Status	1 0 0 0 n n n n	(8n)	n = channel No.
Note No.	0 k k k k k k k		k = 0(C-2) ~ 127(G8)
Velocity	0 v v v v v v v		Ignore vs

2 Key ON/OFF

Status	1 0 0 1 n n n n	(9n)	n = channel No.
Note No.	0 k k k k k k k		k = 0(C-2) ~ 127(G8)
Velocity	0 v v v v v v v		v = 1 ~ 127 Key ON
	0 0 0 0 0 0 0 0		Key OFF

3 Control change

Status	1 0 1 1 n n n n	(Bn)
Control No.	0 c c c c c c c	
Control Value	0 v v v v v v v	

c = 1	Modulation wheel	v = 0 ~ 127
c = 2	Breath control	v = 0 ~ 127
c = 4	Foot control	v = 0 ~ 127
c = 5	Portamento time	v = 0 ~ 127
c = 6	Data entry slider	v = 0 ~ 127
c = 8	Balance	v = 0 ~ 127
c = 9-31	Continuous slider	v = 0 ~ 127
c = 9-31	MIDI control	v = 0 ~ 127
c = 64	Sustain SW	v = 0 ~ 63: OFF, 64 ~ 127: ON
c = 65	Portamento SW	v = 0 ~ 63: OFF, 64 ~ 127: ON
c = 66	Sosutenuto	v = 0 ~ 63: OFF, 64 ~ 127: ON
c = 67	Soft	v = 0 ~ 63: OFF, 64 ~ 127: ON
c = 96	Date entry +1	
c = 97	Data entry −1	

The continuous sliders can be assigned to certain internal effects.
MIDI control can be assigned in the same way as foot control.

4 Program change

Status	1 1 0 0 n n n n	(Cn)	n = channel No.
Program No.	0 p p p p p p p		p = 0 ~ 127

0 ~ 31 select internal PERFORMANCE combinations in PERFORMANCE mode.
32 ~ 63 select cartridge PERFORMANCE combinations. Values over 63 repeat this order of selection (INT 1 ~ 32 → CRT 1 ~ 32).
In Single, Dual or Split mode, 0 ~ 63 select INT voices, 64 ~ 127 CRT voices.

5 After touch

Status	1 0 1 1 n n n n	(Dn)	n = channel No.
Value	0 v v v v v v v		v = 0 ~ 127

6 Pitch bender

Status	1 1 1 0 n n n n	(En)	n = channel No.
Value (LSB)	0 u u u u u u u		
Value (MSB)	0 v v v v v v v		

Operates with only the MSB data.

MSB

00000000	Min.
01000000	Mid.
01111111	Max.

2) Channel mode message

1 Poly/All note off

1 0 1 1 n n n n	(Bn)	
0 1 1 1 1 1 1 0	(7E)	Poly/All note off
0 0 0 0 0 0 0 0		

2 Mono/All note off

1 0 1 1 n n n n	(Bn)	
0 1 1 1 1 1 1 1	(7F)	Mono/All note off
0 m m m m m m m		Set to the Mono mode with only m = 1 recognized. Ignore when m = 1.

[4]-2 System Information

1) System real time messages

Active sensing

Status	1 1 1 1 1 1 1 0	(FE)

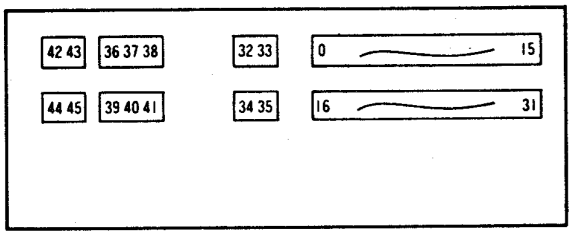
Upon reception of the code, sensing will start. When there is no status byte or data for 300 msec, the MIDI reception buffer is cleared and the on-going sound turned OFF.

2) System exclusive messages

1 Parameter change (Switch remote)

Status	1 1 1 1 0 0 0 0	(F0)
ID No.	0 1 0 0 0 0 1 1	(43)
Substatus/ device No.	0 0 0 1 n n n n	(1n)
Parameter group No.	0 0 0 1 1 0 1 1	(1B)
Switch No.	0 mmmmmmm	
Data	0 d d d d d d d	d = 0: OFF d = 127: ON
EOX	1 1 1 1 0 1 1 1	(F7)

All the panel switches are controlled.
The switch numbers are follows:



2 Parameter change
Same as for transmission

3 Bulk data
Same as for transmission

4 Dump request

For		
	Voice edit buffer	(f = 0)
	Supplement edit buffer	(f = 5)
	Packed 32 supplement	(f = 6)
	Packed 32 voice	(f = 9)

Status	1 1 1 1 0 0 0 0	(F0)
ID No.	0 1 0 0 0 0 1 1	(43)
Substatus/ device No.	0 0 1 0 n n n n	(2n)
Format No.	0 f f f f f f f	f = 0, 5, 6, 9
EOX	1 1 1 1 0 1 1 1	(F7)

- **Universal bulk dump**

Status	1 1 1 1 0 0 0 0	(F0)
ID No.	0 1 0 0 0 0 1 1	(43)
Substatus/ device No.	0 0 1 0 n n n n	(2n)
Format No.	0 1 1 1 1 1 1 0	(7E)
Classification name (ASCII 4 letters)	0 a a a a a a a ↓ 0 a a a a a a a	
Data format name (ASCII 6 letters)	0 mmmmmmm ↓ 0 mmmmmmm	
EOX	1 1 1 1 0 1 1 1	

Classification name and data format name are same as for transmission.

88·05·27
035 musique

The Complete DX7II.

In **The Complete DX7II** you will find the most comprehensive guide to this new generation of DX instruments. You don't need any previous knowledge of, or experience with, the instrument, to work through this book – it takes you from A to Z, covering every facet of the DX7II from basic audio theory to advanced programming techniques. There are stops along the way for discussions of algorithms, carriers, modulators, feedback, envelope generators, LFOs, velocity sensitivity, level scaling, rate scaling, micro tuning, stereo panning, and MIDI implementation. By the time you have completed the 90 hands-on exercises – asssisted by the three full-length soundsheets – you'll be making the DX7II sound like the pros do. In addition, there are over 500 illustrations, diagrams, and photographs to guide you through every nuance of programming this incredible synthesizer. If you own or use a DX7II – or if you have ever thought about owning one – this book is an essential addition to your library.

Howard Massey has been professionally involved in the world of synthesizers for over a decade. He is a composer, record producer, and audio engineer with credits that include work for Kraftwerk and Spandau Ballet. He is also the executive director of the Center for Electronic Music, a nonprofit corporation based in New York City and offering educational and production services in music technology. He has been teaching digital FM synthesis to both professionals and novices since 1983, and he has also organized and taught numerous seminars and workshops on MIDI and programmable analog synthesis.

3 SOUND SHEETS INSIDE

Amsco Publications
UK ISBN 0.7119.1239.4
US ISBN 0.8256.1119.9
Order No. AM 67109