Application Note

A TUTORIAL ON MIDI AND WAVETABLE MUSIC SYNTHESIS

By Jim Heckroth

INTRODUCTION

The Musical Instrument Digital Interface (MIDI) protocol has been widely accepted and utilized by musicians and composers since its conception in the 1982/1983 time frame. MIDI data is a very efficient method of representing musical performance information, and this makes MIDI an attractive protocol for computer applications which produce sound, such as multimedia presentations or computer games. However, the lack of standardization of synthesizer capabilities hindered applications developers and presented MIDI users with a rather steep learning curve to overcome. Fortunately, thanks to the publication of the General MIDI System specification, wide acceptance of the most common PC/MIDI interfaces, support for MIDI in Microsoft WINDOWS, and the evolution of low-cost high-quality wavetable music synthesizers, the MIDI protocol is now seeing widespread use in a growing number of applications. This paper gives a brief overview of the standards and terminology associated with the generation of sound using the MIDI protocol and wavetable music synthesizers.

USE OF MIDI IN MULTIMEDIA APPLICATIONS

Originally developed to allow musicians to connect synthesizers together, the MIDI protocol is now finding widespread use in the generation of sound for games and multimedia applications. There are several advantages to generating sound with a MIDI synthesizer rather than using sampled audio from disk or CD-ROM. The first advantage is storage space. Data files used to store digitally sampled audio in PCM format (such as .WAV files) tend to be quite large. This is especially true for lengthy musical pieces captured in stereo using high sampling rates. MIDI data files, on the other hand, are extremely small when compared with sampled audio files. For instance, files containing high quality stereo sampled audio require about 10 MBytes of data per minute of sound, while a typical MIDI sequence might consume less than 10 KBytes of data per minute of sound. This is because the MIDI file does not contain the sampled audio data, it contains only the instructions needed by a synthesizer to play the sounds. These instructions are in the form of MIDI messages, which instruct the synthesizer which sounds to use, which notes to play, and how loud to play each note. The actual sounds are then generated by the synthesizer.

The smaller file size also means that less of the PCs bandwidth is utilized in spooling this data out to the peripheral which is generating sound. Other advantages of utilizing MIDI to generate sounds include the ability to easily edit the music, and the ability to change the playback speed and the pitch or key of the sounds independently. This last point is particularly important in synthesis applications such as karaoke equipment, where the musical key and tempo of a song may be selected by the user.

Cirrus Logic, Inc.
Crystal Semiconductor Products Division
P.O. Box 17847, Austin, Texas 78760
(512) 445 7222 FAX: (512) 445 7581
http://www.crystal.com

MIDI SYSTEMS

The Musical Instrument Digital Interface (MIDI) protocol provides a standardized and efficient means of conveying musical performance information as electronic data. MIDI information is transmitted in "MIDI messages", which can be thought of as instructions which tell a music synthesizer how to play a piece of music. The Synthesizer receiving the MIDI data must generate the actual sounds. The MIDI 1.0 Detailed Specification, published by the International MIDI Association, provides a complete description of the MIDI protocol.

The MIDI data stream is a unidirectional asynchronous bit stream at 31.25 kbits/sec. with 10 bits transmitted per byte (a start bit, 8 data bits, and one stop bit). The MIDI interface on a MIDI instrument will generally include three different MIDI connectors, labeled IN, OUT, and THRU. The MIDI data stream is usually originated by a MIDI controller, such as a musical instrument keyboard, or by a MIDI sequencer. A MIDI controller is a device which is played as an instrument, and it translates the performance into a MIDI data stream in real time (as it is played). A MIDI sequencer is a device which allows MIDI data sequences to be captured, stored, edited, combined, and replayed. The MIDI data output from a MIDI controller or sequencer is transmitted via the devices' MIDI OUT connector.

The recipient of this MIDI data stream is commonly a MIDI sound generator or sound module, which will receive MIDI messages at its MIDI IN connector, and respond to these messages by playing sounds. Figure 1 shows a simple MIDI system, consisting of a MIDI keyboard controller and a MIDI sound module. Note that many MIDI keyboard instruments include both the keyboard controller and the MIDI sound module functions within the same unit. In these units, there is an internal link between the keyboard and the sound module which may be enabled or disabled by setting the "local control" function of the instrument to ON or OFF respectively.

The single physical MIDI channel is divided into 16 logical channels by the inclusion of a 4-bit channel number within many of the MIDI messages. A musical instrument keyboard can generally be set to transmit on any one of the sixteen MIDI channels. A MIDI sound source, or sound module, can be set to receive on specific MIDI channel(s). In the system depicted in Figure 1, the sound module would have to be set to receive the channel which the keyboard controller is transmitting on in order to play sounds.

Information received on the MIDI IN connector of a MIDI device is transmitted back out (repeated) at the devices' MIDI THRU connector. Several MIDI sound modules can be daisy-chained by connecting the THRU output of one device to the IN connector of the next device downstream in the chain.

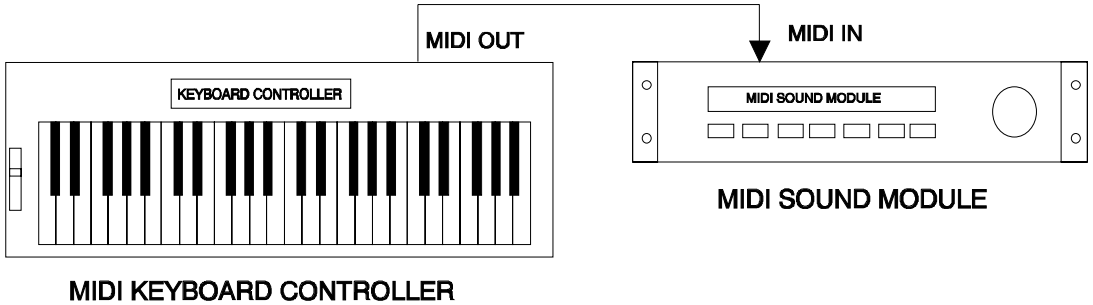

Figure 1. A Simple MIDI System

Figure 2 shows a more elaborate MIDI system. In this case, a MIDI keyboard controller is used as an input device to a MIDI sequencer, and there are several sound modules connected to the sequencer's MIDI OUT port. A composer might utilize a system like this to write a piece of music consisting of several different parts, where each part is written for a different instrument. The composer would play the individual parts on the keyboard one at a time, and these individual parts would be captured by the sequencer. The sequencer would then play the parts back together through the sound modules. Each part would be played on a different MIDI channel, and the sound modules would be set to receive different channels. For example, Sound module number 1 might be set to play the part received on channel 1 using a piano sound, while module 2 plays the information received on channel 5 using an acoustic bass sound, and the drum machine plays the percussion part received on MIDI channel 10.

In the last example, a different sound module is used to play each part. However, sound modules which are "multi-timbral" are capable of playing several different parts simultaneously. A single multi-timbral sound module might be configured to receive the piano part on channel 1, the bass part on channel 5, and the drum part on channel 10, and would play all three parts simultaneously.

Figure 3 depicts a PC-based MIDI system. In this system, the PC is equipped with an internal MIDI interface card which sends MIDI data to an external multi-timbral MIDI synthesizer module. Application software, such as Multimedia presentation packages, educational software, or games, send information to the MIDI interface card over the PC bus. The MIDI interface converts this information into MIDI messages which are sent to the sound module. Since this is a multi-timbral module, it can play many different musical parts, such as piano, bass and drums, at the same time. Sophisticated MIDI sequencer software packages are also available for the PC. With this software running on the PC, a user could connect a MIDI keyboard controller to the MIDI IN port of the MIDI interface card, and have the same music composition capabilities discussed in the last paragraph.

There are a number of different configurations of PC-based MIDI systems possible. For instance, the MIDI interface and the MIDI sound module might be combined on the PC add-in card. In fact, the Microsoft Multimedia PC (MPC) Specification states

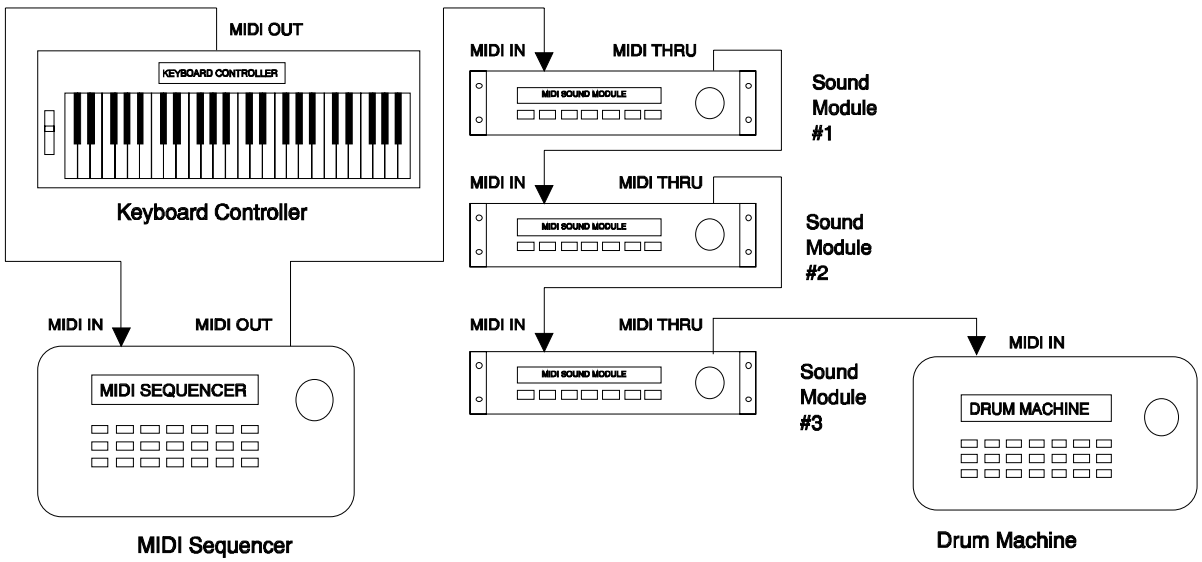

Figure 2. An Expanded MIDI System

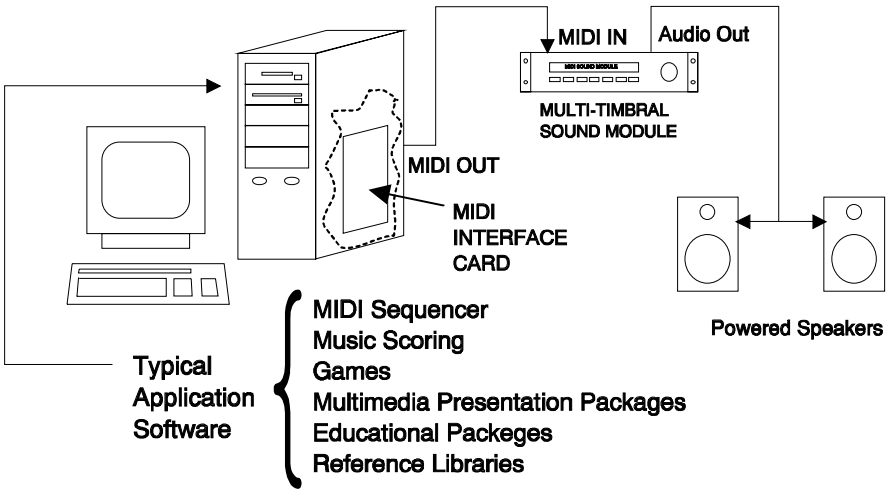

Figure 3. PC-Based MIDI System

that a PC add-in sound card must have an on-board synthesizer in order to be MPC compliant. Until recently, most MPC compliant sound cards included FM synthesizers with limited capabilities and marginal sound quality. With these systems, an external wavetable synthesizer module might be added to get better sound quality. Recently, more advanced sound cards have been appearing which include high quality wavetable music synthesizers on-board, or as a daughter-card options. With the increasing use of the MIDI protocol in PC applications, this trend is sure to continue.

MIDI MESSAGES

A MIDI message is made up of an eight bit status byte which is generally followed by one or two data bytes. There are a number of different types of MIDI messages. At the highest level, MIDI messages are classified as being either Channel Messages or System Messages. Channel messages are those which apply to a specific channel, and the channel number is included in the status byte for these messages. System messages are not channel specific, and no channel number is indicated in their status bytes. Channel Messages may be further classified as being either Channel Voice Messages, or Mode Messages. Channel Voice Messages carry musical performance data, and these messages comprise most of the traffic in a typical MIDI data stream. Channel Mode messages affect the way a receiving instrument will respond to the Channel Voice messages. MIDI System Messages are classified as being System Common Messages, System Real Time Messages, or System Exclusive Messages. System Common messages are intended for all receivers in the system. System Real Time messages are used for synchronization between clock-based MIDI components. System Exclusive messages include a Manufacturer's Identification (ID) code, and are used to transfer any number of data bytes in a format specified by the referenced manufacturer. The various classes of MIDI messages are discussed in more detail in the following paragraphs.

Channel Voice Messages

Channel Voice Messages are used to send musical performance information. The messages in this category are the Note On, Note Off, Polyphonic Key Pressure, Channel Pressure, Pitch Bend Change, Program Change, and the Control Change message.

In MIDI systems, the activation of a particular note and the release of the same note are considered as two separate events. When a key is pressed on a

MIDI keyboard instrument or MIDI keyboard controller, the keyboard sends a Note On message on the MIDI OUT port. The keyboard may be set to transmit on any one of the sixteen logical MIDI channels, and the status byte for the Note On message will indicate the selected channel number. The Note On status byte is followed by two data bytes, which specify key number (indicating which key was pressed) and velocity (how hard the key was pressed). The key number is used in the receiving synthesizer to select which note should be played, and the velocity is normally used to control the amplitude of the note. When the key is released, the keyboard instrument or controller will send a Note Off message. The Note Off message also includes data bytes for the key number and for the velocity with which the key was released. The Note Off velocity information is normally ignored.

Some MIDI keyboard instruments have the ability to sense the amount of pressure which is being applied to the keys while they are depressed. This pressure information, commonly called "aftertouch", may be used to control some aspects of the sound produced by the synthesizer (vibrato, for example). If the keyboard has a pressure sensor for each key, then the resulting "polyphonic aftertouch" information would be sent in the form of Polyphonic Key Pressure messages. These messages include separate data bytes for key number and pressure amount. It is currently more common for keyboard instruments to sense only a single pressure level for the entire keyboard. This "channel aftertouch" information is sent using the Channel Pressure message, which needs only one data byte to specify the pressure value.

The Pitch Bend Change message is normally sent from a keyboard instrument in response to changes in position of the pitch bend wheel. The pitch bend information is used to modify the pitch of sounds being played on a given channel. The Pitch Bend message includes two data bytes to specify the pitch bend value. Two bytes are required to allow fine enough resolution to make pitch changes resulting from movement of the pitch bend wheel seem to occur in a continuous manner rather than in steps.

The Program Change message is used to specify the type of instrument which should be used to play sounds on a given channel. This message needs only one data byte which specifies the new program number.

MIDI Control Change messages are used to control a wide variety of functions in a synthesizer. Control Change messages, like other MIDI channel messages, should only affect the channel number indicated in the status byte. The control change status byte is followed by one data byte indicating the "controller number", and a second byte which specifies the "control value". The controller number identifies which function of the synthesizer is to be controlled by the message.

Controller Numbers 0 - 31 are generally used for sending data from switches, wheels, faders, or pedals on a MIDI controller device such as a musical instrument keyboard. Control numbers 32 - 63 are used to send an optional Least Significant Byte (LSB) for control numbers 0 through 31, respectively. Some examples of synthesizer functions which may be controlled are modulation (controller number 1), volume (controller number 7), and pan (controller number 10). Controller numbers 64 through 67 are used for switched functions. these are the sustain/damper pedal (controller number 64), portamento (controller number 65), sostenuto pedal (controller number 66), and soft pedal (controller number 67). Controller numbers 16-19 and 80-83 are defined to be general purpose controllers, and controller numbers 48-51 may be used to send an optional LSB for controller numbers 16-19. Several of the MIDI controllers merit more detailed descriptions, and these controllers are described in the following paragraphs.

Controller number zero is defined as the bank select. The bank select function is used in some synthesizers in conjunction with the MIDI Program Change message to expand the number of different instrument sounds which may be specified (the Program Change message alone allows selection of one of 128 possible program numbers). The additional sounds are commonly organized as "variations" of the 128 addressed by the Program Change message. Variations are selected by preceding the Program Change message with a Control Change message which specifies a new value for controller zero (see the Roland General Synthesizer Standard topic covered later in this paper).

Controller numbers 91 through 95 may be used to control the depth or level of special effects, such as reverb or chorus, in synthesizers which have these capabilities.

Controller number 6 (Data Entry), in conjunction with Controller numbers 96 (Data Increment), 97 (Data Decrement), 98 (Registered Parameter Number LSB), 99 (Registered Parameter Number MSB), 100 (Non-Registered Parameter Number LSB), and 101 (Non-Registered Parameter Number MSB), may be used to send parameter data to a synthesizer in order to edit sound patches. Registered parameters are those which have been assigned some particular function by the MIDI Manufacturers Association (MMA) and the Japan MIDI Standards Committee (JMSC). For example, there are Registered Parameter numbers assigned to control pitch bend sensitivity and master tuning for a synthesizer. Non-Registered parameters have not been assigned specific functions, and may be used for different functions by different manufacturers. Parameter data is transferred by first selecting the parameter number to be edited using controllers 98 and 99 or 100 and 101, and then adjusting the data value for that parameter using controller number 6, 96, or 97.

Controller Numbers 121 through 127 are used to implement the MIDI "Channel Mode Messages". These messages are covered in the next section.

Channel Mode Messages

Channel Mode messages (MIDI controller numbers 121 through 127) affect the way a synthesizer responds to MIDI data. Controller number 121 is used to reset all controllers. Controller number 122 is used to enable or disable Local Control (In a MIDI synthesizer which has it's own keyboard, the functions of the keyboard controller and the synthesizer can be isolated by turning Local Control off). Controller numbers 124 through 127 are used to select between Omni Mode On or Off, and to select between the Mono Mode or Poly Mode of operation.

When Omni mode is On, the synthesizer will respond to incoming MIDI data on all channels. When Omni mode is Off, the synthesizer will only respond to MIDI messages on one channel. When Poly mode is selected, incoming Note On messages are played polyphonically. This means that when multiple Note On messages are received, each note is assigned its own voice (subject to the number of voices available in the synthesizer). The result is that multiple notes are played at the same time. When Mono mode is selected, a single voice is assigned per MIDI channel. This means that only one note can be played on a given channel at a given time. Most modern MIDI synthesizers will default to Omni On/Poly mode of operation. In this mode, the synthesizer will play note messages received on any MIDI channel, and notes received on each channel are played polyphonically. In the Omni Off/Poly mode of operation, the synthesizer will receive on a single channel and play the notes received on this channel polyphonically. This mode is useful when several synthesizers are daisy-chained using MIDI THRU. In this case each synthesizer in the chain can be set to play one part (the

MIDI data on one channel), and ignore the information related to the other parts.

Note that a MIDI instrument has one MIDI channel which is designated as its "Basic Channel". The Basic Channel assignment may be hard-wired, or it may be selectable. Mode messages can only be received by an instrument on the Basic Channel.

System Common Messages

The System Common Messages which are currently defined include MTC Quarter Frame, Song Select, Song Position Pointer, Tune Request, and End Of Exclusive (EOX). The MTC Quarter Frame message is part of the MIDI Time Code information used for synchronization of MIDI equipment and other equipment, such as audio or video tape machines.

The Song Select message is used with MIDI equipment, such as sequencers or drum machines, which can store and recall a number of different songs. The Song Position Pointer is used to set a sequencer to start playback of a song at some point other than at the beginning. The Song Position Pointer value is related to the number of MIDI clocks which would have elapsed between the beginning of the song and the desired point in the song. This message can only be used with equipment which recognizes MIDI System Real Time Messages (MIDI Sync).

The Tune Request message is generally used to request an analog synthesizer to retune its' internal oscillators. This message is generally not needed with digital synthesizers.

The EOX message is used to flag the end of a System Exclusive message, which can include a variable number of data bytes.

System Real Time Messages

The MIDI System Real Time messages are used to synchronize all of the MIDI clock-based equipment within a system, such as sequencers and drum machines. Most of the System Real Time messages are normally ignored by keyboard instruments and synthesizers. To help ensure accurate timing, System Real Time messages are given priority over other messages, and these single-byte messages may occur anywhere in the data stream (a Real Time message may appear between the status byte and data byte of some other MIDI message). The System Real Time messages are the Timing Clock, Start, Continue, Stop, Active Sensing, and the System Reset message. The Timing Clock message is the master clock which sets the tempo for playback of a sequence. The Timing Clock message is sent 24 times per quarter note. The Start, Continue, and Stop messages are used to control playback of the sequence.

The Active Sensing signal is used to help eliminate "stuck notes" which may occur if a MIDI cable is disconnected during playback of a MIDI sequence. Without Active Sensing, if a cable is disconnected during playback, then some notes may be left playing indefinitely because they have been activated by a Note On message, but will never receive the Note Off. In transmitters which utilize Active Sensing, the Active Sensing message is sent once every 300 ms by the transmitting device when this device has no other MIDI data to send. If a receiver who is monitoring Active Sensing does not receive any type of MIDI messages for a period of time exceeding 300 ms, the receiver may assume that the MIDI cable has been disconnected, and it should therefore turn off all of its' active notes. Use of Active Sensing in MIDI transmitters and receivers is optional.

The System Reset message, as the name implies, is used to reset and initialize any equipment which receives the message. This message is generally not sent automatically by transmitting devices, and must be initiated manually by a user.

System Exclusive Messages

System Exclusive messages may be used to send data such as patch parameters or sample data between MIDI devices. Manufacturers of MIDI equipment may define their own formats for System Exclusive data. Manufacturers are granted unique identification (ID) numbers by the MMA or the JMSC, and the manufacturer ID number is included as the second byte of the System Exclusive message. The manufacturers ID byte is followed by any number of data bytes, and the data transmission is terminated with the EOX message. Manufacturers are required to publish the details of their System Exclusive data formats, and other manufacturers may freely utilize these formats, provided that they do not alter or utilize the format in a way which conflicts with the original manufacturers specifications.

There is also a MIDI Sample Dump Standard, which is a System Exclusive data format defined in the MIDI specification for the transmission of sample data between MIDI devices.

Running Status

MIDI data is transmitted serially. Musical events which originally occurred at the same time must be sent one at a time in the MIDI data stream, and therefore these events will not actually be played at exactly the same time. However, the resulting delays are generally short enough that the events are perceived as having occurred simultaneously. The MIDI data transmission rate is 31.35 kbit/s with 10 bits transmitted per byte of MIDI data. Thus, a 3 byte Note On or Note Off message takes about 1 ms to be sent. For a person playing a MIDI instrument keyboard, the time skew between playback of notes when 10 keys are pressed simultaneously should not exceed 10 ms, and this would not be perceptible. However, MIDI data being sent from a sequencer can include a number of different parts. On a given beat, there may be a large number of musical events which should occur simultaneously, and the delays introduced by serialization of this information might be noticeable.

To help reduce the amount of data transmitted in the MIDI data stream, a technique called "running status" may be employed. It is very common for a string of consecutive messages to be of the same message type. For instance, when a chord is played on a keyboard, 10 successive Note On messages may be generated, followed by 10 Note Off messages. When running status is used, a status byte is sent for a message only when the message is not of the same type as the last message sent on the same channel. The status byte for subsequent messages of the same type may be omitted (only the data bytes are sent for these subsequent messages). The effectiveness of running status can be enhanced by sending Note On messages with a velocity of zero in place of Note Off messages. In this case, long strings of Note On messages will often occur. Changes in some of the the MIDI controllers or movement of the pitch bend wheel on a musical instrument can produce a staggering number of MIDI channel voice messages, and running status can also help a great deal in these instances.

MIDI SEQUENCERS AND STANDARD MIDI FILES

MIDI messages are received and processed by a MIDI synthesizer in real time. When the synthesizer receives a MIDI "note on" message it plays the appropriate sound. When the corresponding "note off" message is received, the synthesizer turns the note off. If the source of the MIDI data is a musical instrument keyboard, then this data is being generated in real time. When a key is pressed on the keyboard, a "note on" message is generated in real time. In these real time applications, there is no need for timing information to be sent along with the MIDI messages. However, if the MIDI data is to be stored as a data file, and/or edited using a sequencer, then some form of "time-stamping" for the MIDI messages is required.

The International MIDI Association publishes a Standard MIDI Files specification, which provides a standardized method for handling time-stamped MIDI data. This standardized file format for time-stamped MIDI data allows different applications, such as sequencers, scoring packages, and multi-media presentation software, to share MIDI data files.

The specification for Standard MIDI Files defines three formats for MIDI files. MIDI sequencers can generally manage multiple MIDI data streams, or "tracks". MIDI files having Format 0 must store all of the MIDI sequence data on a single track. This is generally useful only for simple "single track" devices. Format 1 files, which are the most commonly used, store data as a collection of tracks. Format 2 files can store several independent patterns.

SYNTHESIZER POLYPHONY AND TIMBRES

The polyphony of a sound generator refers to its ability to play more than one note at a time. Polyphony is generally measured or specified as a number of notes or voices. Most of the early music synthesizers were monophonic, meaning that they could only play one note at a time. If you pressed five keys simultaneously on the keyboard of a monophonic synthesizer, you would only hear one note. Pressing five keys on the keyboard of a synthesizer which was polyphonic with four voices of polyphony would, in general, produce four notes. If the keyboard had more voices (many modern sound modules have 16, 24, or 32 note polyphony), then you would hear all five of the notes.

The different sounds that a synthesizer or sound generator can produce are often referred to as "patches", "programs", "algorithms", sounds, or "timbres". Modern synthesizers commonly use program numbers to represent different sounds they produce. Sounds may then be selected by specifying the program numbers (or patch numbers) for the desired sound. For instance, a sound module might use patch number 1 for its acoustic piano sound, and patch number 36 for its fretless bass sound. The association of patch numbers to sounds is often referred to as a patch map. A MIDI Program Change message is used to tell a device receiving on a given channel to change the instrument sound being used. For example, a sequencer could set up devices on channel 4 to play fretless bass sounds by sending a Program Change message for channel four with a data byte value of 36 (this is the General MIDI program number for the fretless bass patch).

A synthesizer or sound generator is said to be multi-timbral if it is capable of producing two or more different instrument sounds simultaneously. Again, if a synthesizer can play five notes simultaneously, then it is polyphonic. If it can produce a piano sound and an acoustic bass sound at the same time, then it is also multi-timbral. A synthesizer or sound module which has 24 notes of polyphony and which is 6 part multi-timbral (capable of producing 6 different timbres simultaneously) could synthesize the sound of a 6 piece band or orchestra. A sequencer could send MIDI messages for a piano part on channel 1, bass on channel 2, saxophone on channel 3, drums on channel 10, etc. A 16 part multi-timbral synthesizer could receive a different part on each of MIDI's 16 logical channels.

The polyphony of a multi-timbral synthesizer is usually allocated dynamically among the different parts (timbres) being used. In our example, at a given instant five voices might be used for the piano part, two voices for the bass, one for the saxophone, and 6 voices for the drums, leaving 10 voices free. Note that some sounds utilize more than one voice, so the number of notes which may be produced simultaneously may be less than the stated polyphony of the synthesizer, depending on which sounds are being utilized.

THE GENERAL MIDI (GM) SYSTEM

At the beginning of a MIDI sequence, a Program Change message is usually sent on each channel used in the piece in order to set up the appropriate instrument sound for each part. The Program Change message tells the synthesizer which patch number should be used for a particular MIDI channel. If the synthesizer receiving the MIDI sequence uses the same patch map (the assignment of patch numbers to sounds) that was used in the composition of the sequence, then the sounds will be assigned as intended. Unfortunately, prior to General MIDI, there was no standard for the relationship of patch numbers to specific sounds for synthesizers. Thus, a MIDI sequence might produce different sounds when played on different synthesizers, even though the synthesizers had comparable types of sounds. For example, if the composer had selected patch number 5 for channel 1, intending this to be an electric piano sound, but the synthesizer playing the MIDI data had a tuba sound mapped at patch number 5, then the notes intended for the piano would be played on the tuba when using this synthesizer (even though this synthesizer may have a fine electric piano sound available at some other patch number).

The General MIDI (GM) Specification, published by the International MIDI Association, defines a set of general capabilities for General MIDI Instruments. The General MIDI Specification includes the definition of a General MIDI Sound Set (a patch map), a General MIDI Percussion map (mapping of percussion sounds to note numbers), and a set of General MIDI Performance capabilities (number of voices, types of MIDI messages recognized, etc.). A MIDI sequence which has been generated for use on a General MIDI Instrument should play correctly on any General MIDI synthesizer or sound module.

The General MIDI system utilizes MIDI channels 1-9 and 11-16 for chromatic instrument sounds, while channel number 10 is utilized for "key-based" percussion sounds. The General MIDI Sound set for channels 1-9 and 11-16 is given in Table 1. These instrument sounds are grouped into "sets" of related sounds. For example, program numbers 1-8 are piano sounds, 6-16 are chromatic percussion sounds, 17-24 are organ sounds, 25-32 are guitar sounds, etc.

For the instrument sounds on channels 1-9 and 11-16, the note number in a Note On message is used to select the pitch of the sound which will be played. For example if the Vibraphone instrument (program number 12) has been selected on channel 3, then playing note number 60 on channel 3 would play the middle C note (this would be the default note to pitch assignment on most instruments), and note number 59 on channel 3 would play B below middle C. Both notes would be played using the Vibraphone sound.

The General MIDI percussion map used for channel 10 is given in Table 2. For these "key-based" sounds, the note number data in a Note On message is used differently. Note numbers on channel 10 are used to select which drum sound will be played. For example, a Note On message on channel 10 with note number 60 will play a Hi Bongo drum sound. Note number 59 on channel 10 will play the Ride Cymbal 2 sound.

It should be noted that the General MIDI system specifies sounds using program numbers 1 through 128. The MIDI Program Change message used to select these sounds uses an 8-bit byte, which corresponds to decimal numbering from 0 through 127, to specify the desired program number. Thus, to select GM sound number 10, the Glockenspiel, the Program Change message will have a data byte with the decimal value 9.

The General MIDI system specifies which instrument or sound corresponds with each program/patch number, but General MIDI does not specify how these sounds are produced. Thus, pro-

Prog #	Instrument Name	Prog #	Instrument Name	Prog #	Instrument Name
1	Acoustic Grand Piano	44	Contrabass	87	Lead 7 (fifths)
2	Bright Acoustic Piano	45	Tremolo Strings	88	Lead 8 (bass + lead)
3	Electric Grand Piano	46	Pizzicato Strings	89	Pad 1 (new age)
4	Honky-tonk Piano	47	Orchestral Harp	90	Pad 2 (warm)
5	Electric Piano 1	48	Timpani	91	Pad 3 (polysynth)
6	Electric Piano 2	49	String Ensemble 1	92	Pad 4 (choir)
7	Harpsichord	50	String Ensemble 2	93	Pad 5 (bowed)
8	Clavi	51	SynthStrings 1	94	Pad 6 (metallic)
9	Celesta	52	SynthStrings 2	95	Pad 7 (halo)
10	Glockenspiel	53	Choir Aahs	96	Pad 8 (sweep)
11	Music Box	54	Voice Oohs	97	FX 1 (rain)
12	Vibraphone	55	Synth Voice	98	FX 2 (soundtrack)
13	Marimba	56	Orchestra Hit	99	FX 3 (crystal)
14	Xylophone	57	Trumpet	100	FX 4 (atmosphere)
15	Tubular Bells	58	Trombone	101	FX 5 (brightness)
16	Dulcimer	59	Tuba	102	FX 6 (goblins)
17	Drawbar Organ	60	Muted Trumpet	103	FX 7 (echoes)
18	Percussive Organ	61	French Horn	104	FX 8 (sci-fi)
19	Rock Organ	62	Brass Section	105	Sita
20	Church Organ	63	SynthBrass 1	106	Banjo
21	Reed Organ	64	SynthBrass 2	107	Shamisen
22	Accordion	65	Soprano Sax	108	Koto
23	Harmonica	66	Alto Sax	109	Kalimba
24	Tango Accordion	67	Tenor Sax	110	Bag pipe
25	Acoustic Guitar (nylon)	68	Baritone Sax	111	Fiddle
26	Acoustic Guitar (steel)	69	Oboe	112	Shanai
27	Electric Guitar (jazz)	70	English Horn	113	Tinkle Bell
28	Electric Guitar (clean)	71	Bassoon	114	Agogo
29	Electric Guitar (muted)	72	Clarinet	115	Steel Drums
30	Overdriven Guitar	73	Piccolo	116	Woodblock
31	Distortion Guitar	74	Flute	117	Taiko Drum
32	Guitar harmonics	75	Recorder	118	Melodic Tom
33	Acoustic Bass	76	Pan Flute	119	Synth Drum
34	Electric Bass (finger)	77	Blown Bottle	120	Reverse Cymbal
35	Electric Bass (pick)	78	Shakuhachi	121	Guitar Fret Noise
36	Fretless Bass	79	Whistle	122	Breath Noise
37	Slap Bass 1	80	Ocarina	123	Seashore
38	Slap Bass 2	81	Lead 1 (square)	124	Bird Tweet
39	Synth Bass 1	82	Lead 2 (sawtooth)	125	Telephone Ring
40	Synth Bass 2	83	Lead 3 (calliope)	126	Helicopter
41	Violin	84	Lead 4 (chiff)	127	Applause
42	Viola	85	Lead 5 (charang)	128	Gunshot
43	Cello	86	Lead 6 (voice)		

Table 1. General MIDI Sound Set (All Channels Except 10)

Note #	Drum Sound	Note #	Drum Sound	Note #	Drum Sound
35	Acoustic Bass Drum	51	Ride Cymbal 1	67	High Agogo
36	Bass Drum 1	52	Chinese Cymbal	68	Low Agogo
37	Side Stick	53	Ride Bell	69	Cabasa
38	Acoustic Snare	54	Tambourine	70	Maracas
39	Hand Clap	55	Splash Cymbal	71	Short Whistle
40	Electric Snare	56	Cowbell	72	Long Whistle
41	Low Floor Tom	57	Crash Cymbal 2	73	Short Guiro
42	Closed Hi-Hat	58	Vibraslap	74	Long Guiro
43	High Floor Tom	59	Ride Cymbal 2	75	Claves
44	Pedal Hi-Hat	60	Hi Bongo	76	Hi Wood Block
45	Low Tom	61	Low Bongo	77	Low Wood Block
46	Open Hi-Hat	62	Mute Hi Conga	78	Mute Cuica
47	Low Mid Tom	63	Open Hi Conga	79	Open Cuica
48	Hi Mid Tom	64	Low Conga	80	Mute Triangle
49	Crash Cymbal 1	65	High Timbale	81	Open Triangle
50	High Tom	66	Low Timbale		

Table 2. General MIDI Percussion Map (Channel 10)

gram number 1 should select the Acoustic Grand Piano sound on any General MIDI instrument. However, the Acoustic Grand Piano sound on two General MIDI synthesizers which use different synthesis techniques may sound quite different.

THE ROLAND GENERAL SYNTHESIZER (GS) STANDARD

The Roland General Synthesizer (GS) functions are a superset of those specified for General MIDI. The GS system includes all of the GM sounds (which are referred to as "capital instrument" sounds), and adds new sounds which are organized as variations of the capital instruments.

Variations are selected using the MIDI Control Change message in conjunction with the Program Change message. The Control Change message is sent first, and it is used to set controller number 0 to some specified nonzero value indicating the desired variation (some capital sounds have several different variations). The Control Change message is followed by a MIDI Program Change message which indicates the program number of the related capital instrument. For example, Capital instrument number 25 is the Nylon String Guitar. The Ukulele is a variation of this instrument. The Ukulele is selected by sending a Control Change message which sets controller number 0 to a value of 8, followed by a program change message on the same channel which selects program number 25. Sending the Program change message alone would select the capital instrument, the Nylon String Guitar. Note also that a Control Change of controller number 0 to a value of 0 followed by a Program Change message would also select the capital instrument.

The GS system also includes adjustable reverberation and chorus effects. The effects depth for both reverb and chorus may be adjusted on an individual MIDI channel basis using Control Change messages. The type of reverb and chorus sounds employed may also be selected using System Exclusive messages.

SYNTHESIZER IMPLEMENTATIONS: FM VS. WAVETABLE

There are a number of different technologies or algorithms used to create sounds in music synthesizers. Two widely used techniques are Frequency Modulation (FM) synthesis and Wavetable synthesis. FM synthesis techniques generally use one periodic signal (the modulator) to modulate the frequency of another signal (the carrier). If the modulating signal is in the audible range, then the result will be a significant change in the timbre of the carrier signal. Each FM voice requires a minimum of two signal generators. These generators are commonly referred to as "operators", and different FM synthesis implementations have varying degrees of control over the operator parameters. Sophisticated FM systems may use 4 or 6 operators per voice, and the operators may have adjustable envelopes which allow adjustment of the attack and decay rates of the signal. Although FM systems were implemented in the analog domain on early synthesizer keyboards, modern FM synthesis implementations are done digitally.

FM synthesis techniques are very useful for creating expressive new synthesized sounds. However, if the goal of the synthesis system is to recreate the sound of some existing instrument, this can generally be done more accurately with digital sample-based techniques. Digital sampling systems store high quality sound samples digitally, and then replay these sounds on demand. Digital sample-based synthesis systems may employ a variety of special techniques, such as sample looping, pitch shifting, mathematical interpolation, and polyphonic digital filtering, in order to reduce the amount of memory required to store the sound samples (or to get more types of sounds from a given amount of memory). These sample-based synthesis systems are often called "wavetable" synthesizers (the sample memory in these systems contains a large number of sampled sound segments, and can be thought of as a "table" of sound waveforms which may be looked up and utilized when needed). A number of the special techniques employed in this type of synthesis are discussed in the following paragraphs.

WAVETABLE SYNTHESIS TECHNIQUES

Looping and Envelope Generation

One of the primary techniques used in wavetable synthesizers to conserve sample memory space is the looping of sampled sound segments. For a large number of instrument sounds, the sound can be modeled as consisting of two major sections, the attack section and the sustain section. The attack section is the initial part of the sound, where the amplitude and the spectral characteristics of the sound may be changing very rapidly. The sustain section of the sound is that part of the sound following the attack, where the characteristics of the sound are changing less dynamically. Figure 4 shows a waveform with portions which could be considered the attack and the sustain sections indicated. In this example, the spectral characteristics of the waveform remain constant throughout the sustain section, while the amplitude is decreasing at a fairly constant rate. This is an exaggerated example, in most natural instrument sounds, both the spectral characteristics and the amplitude continue to change through the duration of the sound. The sustain section, if one can be identified, is that section for which the characteristics of the sound are relatively constant.

A great deal of memory can be saved in wave-table synthesis systems by storing only a short segment of the sustain section of the waveform, and then looping this segment during playback. Figure 5 shows a two period segment of the sustain section from the waveform in Figure 4, which has been looped to create a steady state signal. If the original sound had a fairly constant spectral content and amplitude during the sustained section, then the sound resulting from this looping operation should

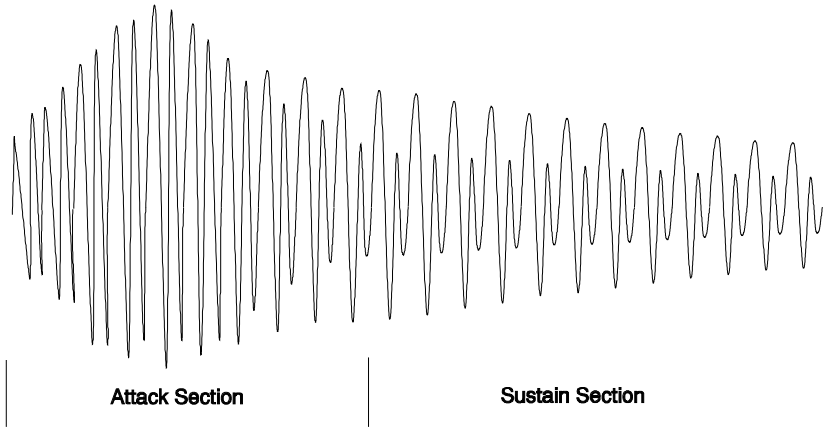

Figure 4. Attack and Sustain Portions of a waveform

be a good approximation of the sustained section of the original.

For many acoustic string instruments, the spectral characteristics of the sound remain fairly constant during the sustain section, while the amplitude of the signal decays. This can be simulated with a looped segment by multiplying the looped samples by a decreasing gain factor during playback to get the desired shape or envelope. The amplitude envelope of a sound is commonly modeled as consisting of some number of linear segments. An example is the commonly used four part piecewise-linear Attack-Decay-Sustain-Release (ADSR) envelope model. Figure 6 depicts a typical ADSR envelope shape, and Figure 7 shows the result of applying this envelope to the looped waveform from Figure 5.

A typical wavetable synthesis system would store separate sample segments for the attack section and the looped section of an instrument. These sample segments might be referred to as the initial sound and the loop sound. The initial sound is played once through, and then the loop sound is played repetitively until the note ends. An envelope generator function is used to create an envelope which is appropriate for the particular instrument, and this envelope is applied to the output samples during playback. Playback of the initial wave (with the the Attack portion of the envelope applied) begins when a Note On message is received. The length of the initial sound segment is fixed by the number of samples in the segment, and the length of the Attack and Decay sections of the envelope are gener-

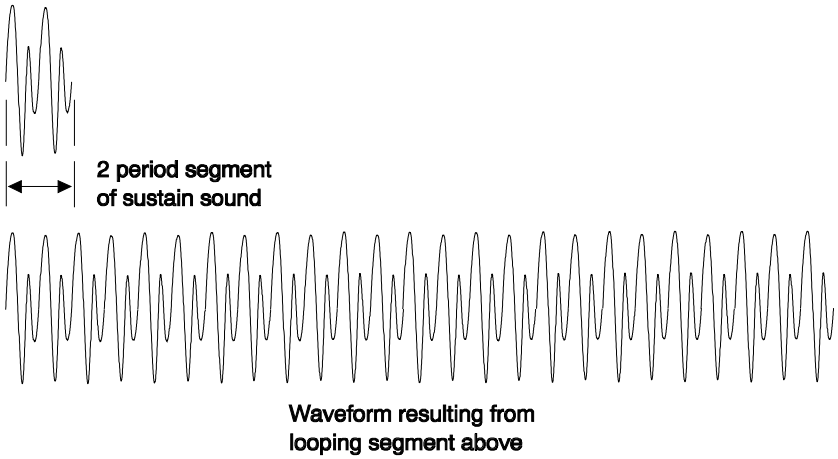

Figure 5. Looping a Sound Segment

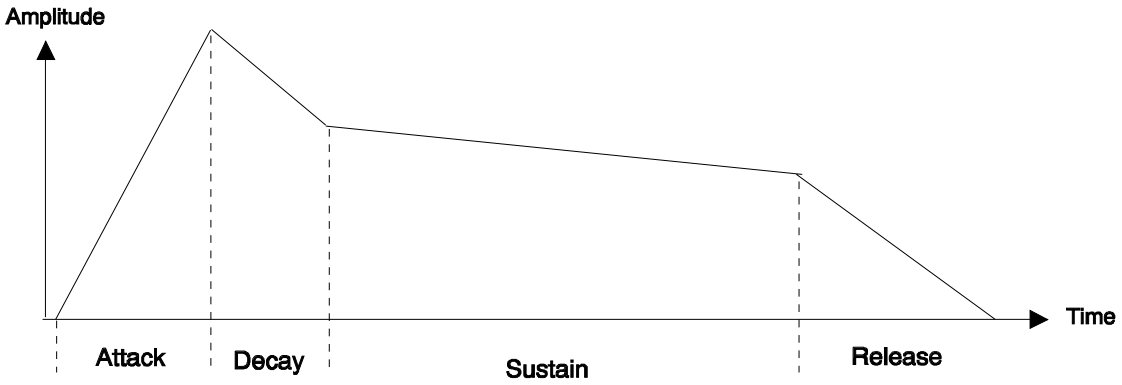

Figure 6. A Typical ADSR Amplitufe Envelope

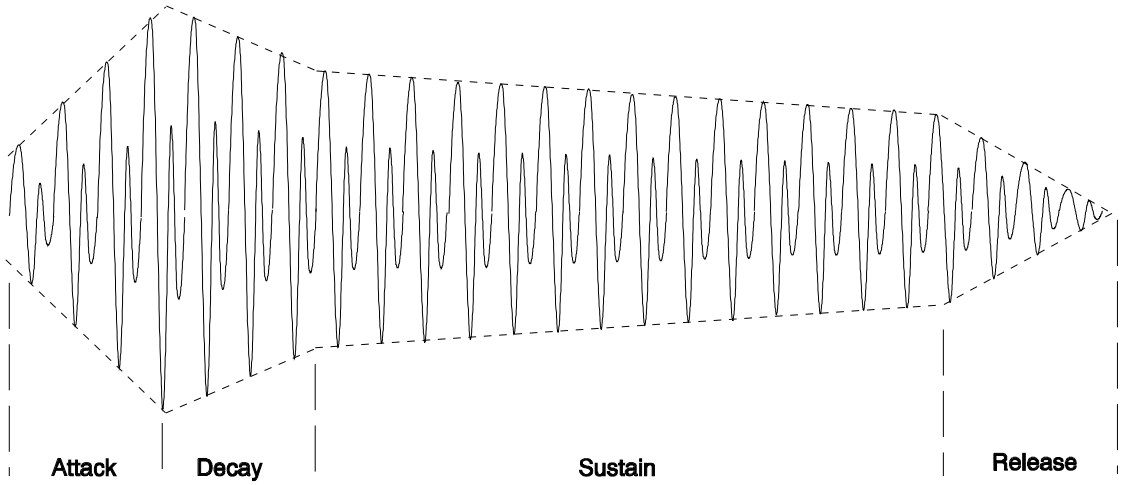

Figure 7. ADSR Envelope Applied to Looped Sound Segment

ally also fixed for a given instrument sound. The sustain section will continue to repeat the loop samples while applying the Sustain envelope slope (which decays slowly in our examples), until a Note Off message is applied. The Note Off message triggers the beginning of the Release portion of the envelope.

Loop Length

The loop length is measured as a number of samples, and the length of the loop should be equal to an integral number of periods of the fundamental pitch of the sound being played (if this is not true, then an undesirable "pitch shift" will occur during playback when the looping begins). Of course, the length of the pitch period of a sampled instrument sound will generally not work out to be an integral number of sample periods. Therefore, it is common to perform a "resampling" process on the original sampled sound, to get new a new sound sample for which the pitch period is an integral number of sample periods.

In practice, the length of the loop segment for an acoustic instrument sample may be many periods with respect to the fundamental pitch of the sound. If the sound has a natural vibrato or chorus effect, then it is generally desirable to have the loop segment length be an integral multiple of the period of the vibrato or chorus.

One-Shot Sounds

The previous paragraphs discussed dividing a sampled sound into an attack section and a sustain section, and then using looping techniques to minimize the storage requirements for the sustain portion. However, some sounds, particularly sounds of short duration or sounds whose characteristics change dynamically throughout their duration, are not suitable for looped playback techniques. Short drum sounds often fit this description. These sounds are stored as a single sample segment which is played once through with no looping. This class of sounds are referred to as "one-shot" sounds.

Sample Editing and Processing

There are a number of sample editing and processing steps involved in preparing sampled sounds for use in a wave-table synthesis system. The requirements for editing the original sample data to identify and extract the initial and loop segments, and for resampling the data to get a pitch period length which is an integer multiple of the sampling period, have already been mentioned.

Editing may also be required to make the endpoints of the loop segment compatible. If the amplitude and the slope of the waveform at the beginning of the loop segment do not match those at the end of the loop, then a repetitive "glitch" will be heard during playback of the looped section. Additional processing may be performed to "compress" the dynamic range of the sound to improve the signal/quantizing noise ratio or to conserve sample memory. This topic is addressed next.

When all of the sample processing has been completed, the resulting sampled sound segments for the various instruments are tabulated to form the sample memory for the synthesizer.

Sample Data Compression

The signal-to-quantizing noise ratio for a digitally sampled signal is limited by sample word size (the number of bits per sample), and by the amplitude of the digitized signal. Most acoustic instrument sounds reach their peak amplitude very quickly, and the amplitude then slowly decays from this peak. The ear's sensitivity dynamically adjusts to signal level. Even in systems utilizing a relatively small sample word size, the quantizing noise level is generally not perceptible when the signal is near maximum amplitude. However, as the signal level decays, the ear becomes more sensitive, and the noise level will appear to increase. Of course, using a larger word size will reduce the quantizing noise, but there is a considerable price penalty paid if the number of samples is large.

Compression techniques may be used to improve the signal-to-quantizing noise ratio for some sampled sounds. These techniques reduce the dynamic range of the sound samples stored in the sample memory. The sample data is decompressed during playback to restore the dynamic range of the signal. This allows the use of sample memory with a smaller word size (smaller dynamic range) than is utilized in the rest of the system. There are a number of different compression techniques which may be used to compress the dynamic range of a signal.

For signals which begin at a high amplitude and decay in a fairly linear fashion, a simple compression technique can be effective. If the slope of the decay envelope of the signal is estimated, then an envelope with the complementary slope (the negative of the decay slope) can be constructed and applied to the original sample data. The resulting sample data, which now has a flat envelope, can be stored in the sample memory, utilizing the full dynamic range of the memory. The decay envelope can then be applied to the stored sample data during sound playback to restore the envelope of the original sound.

Note that there is some compression effect inherent in the looping techniques described earlier. If the loop segment is stored at an amplitude level which makes full use of the dynamic range available in the sample memory, and the processor and D/A

converters used for playback have a wider dynamic range than the sample memory, then the application of a decay envelope during playback will have a decompression effect similar to that described in the previous paragraph.

Pitch Shifting

In order to minimize sample memory requirements, wavetable synthesis systems utilize pitch shifting, or pitch transposition techniques, to generate a number of different notes from a single sound sample of a given instrument. For example, if the sample memory contains a sample of a middle C note on the acoustic piano, then this same sample data could be used to generate the C# note or D note above middle C using pitch shifting.

Pitch shifting is accomplished by accessing the stored sample data at different rates during playback. For example, if a pointer is used to address the sample memory for a sound, and the pointer is incremented by one after each access, then the samples for this sound would be accessed sequentially, resulting in some particular pitch. If the pointer increment was two rather than one, then only every second sample would be played, and the resulting pitch would be shifted up by one octave (the frequency would be doubled).

Frequency Accuracy

In the previous example, the sample memory address pointer was incremented by an integer number of samples. This allows only a limited set of pitch shifts. In a more general case, the memory pointer would consist of an integer part and a fractional part, and the increment value could be a fractional number of samples. The integer part of the address pointer is used to address the sample memory, the fractional part is used to maintain frequency accuracy. For example if the increment value was equivalent to 1/2, then the pitch would be shifted down by one octave (the frequency would be halved). When non-integer increment values are utilized, the frequency resolution for playback is determined by the number of bits used to represent the fractional part of the address pointer and the address increment parameter.

Interpolation

When the fractional part of the address pointer is non-zero, then the "desired value" falls between available data samples. Figure 8 depicts a simplified addressing scheme wherein the Address Pointer and the increment parameter each have a 4-bit integer part and a 4-bit fractional part. In this case, the increment value is equal to 1 1/2 samples. Very simple systems might simply ignore the fractional part of the address when determining the sample value to be sent to the D/A converter. The data values sent to the D/A converter when using this approach are indicated in the Figure 8, case I. A slightly better approach would be to use the nearest available sample value. More sophisticated systems would perform some type of mathematical interpolation between available data points in order to get a value to be used for playback. Values which might be sent to the D/A when interpolation is employed are shown as case II. Note that the overall frequency accuracy would be the same for both cases indicated, but the output is severely distorted in the case where interpolation is not used.

There are a number of different algorithms used for interpolation between sample values. The simplest is linear interpolation. With linear interpolation, interpolated value is simply the weighted average of the two nearest samples, with the fractional address used as a weighting constant. For example, if the address pointer indicated an address of (n+K), where n is the integer part of the address and K is the fractional part, than the interpolated value can be calculated as $s(n+K) = (1-K)s(n) + (K)s(n+1)$, where s(n) is the sample data value at address n. More sophisticated interpolation techniques can can be utilized to further reduce distortion, but these techniques are computationally expensive.

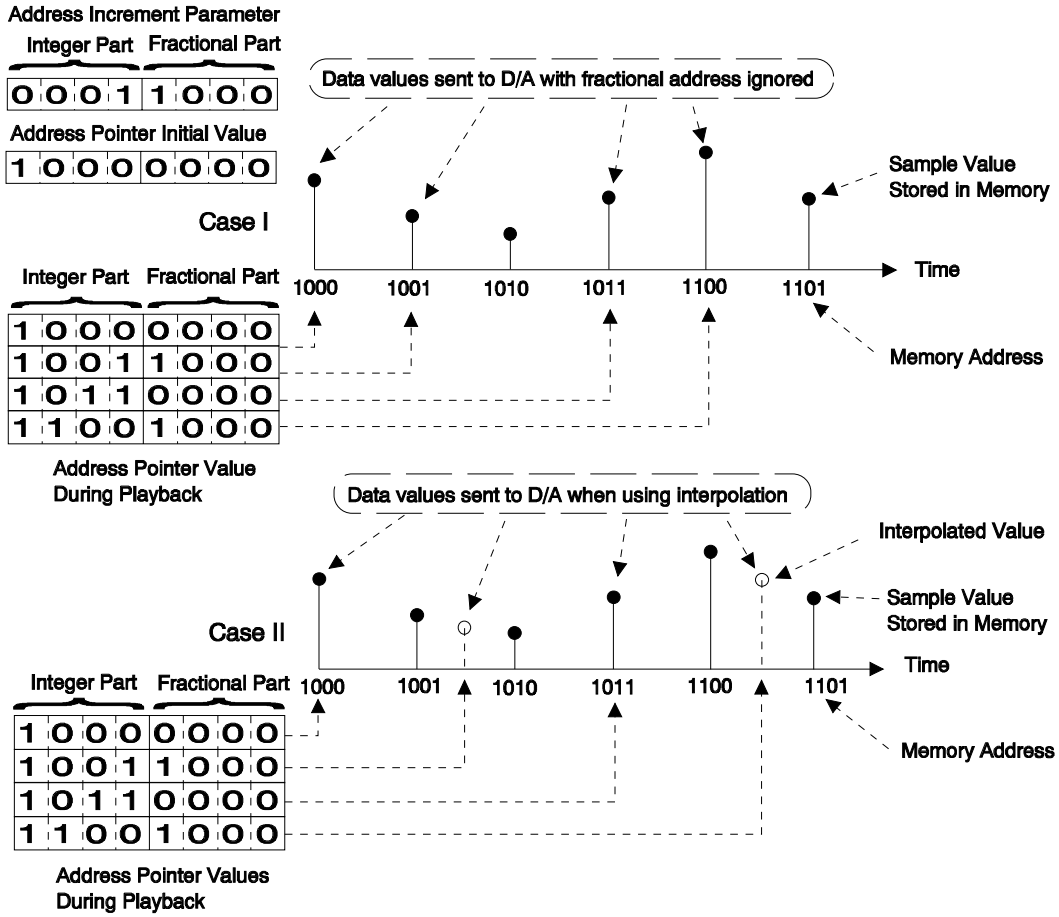

Figure 8. Sample Memory Addressing and Interpolation

Oversampling

Oversampling of the sound samples may also be used to improve distortion in wavetable synthesis systems. For example, if 4X oversampling were utilized for a particular instrument sound sample, then an address increment value of 4 would be used for playback with no pitch shift. The data points chosen during playback will be closer to the "desired values", on the average, than they would be if no oversampling were utilized because of the increased number of data points used to represent the waveform. Of course, oversampling has a high cost in terms of sample memory requirements.

In many cases, the best approach may be to utilize linear interpolation combined with varying degrees of oversampling where needed. The linear interpolation technique provides reasonable accuracy for many sounds, without the high penalty in terms of processing power required for more sophisticated interpolation methods. For those sounds which need better accuracy, oversampling is employed. With this approach, the additional memory required for oversampling is only utilized where it is most needed. The combined effect of linear interpolation and selective oversampling can produce excellent results.

Splits

When the pitch of a sampled sound is changed during playback, the timbre of the sound is changed somewhat also. For small changes in pitch (up to a few semitones), the timbre change is generally not noticed. However, if a large pitch shift is used, the

resulting note will sound unnatural. Thus, a particular sample of an instrument sound will be useful for recreating a limited range of notes using pitch shifting techniques. To get coverage of the entire instrument range, a number of different samples of the instrument are used, and each of these samples is used to synthesize a limited range of notes. This technique can be thought of as splitting a musical instrument keyboard into a number of ranges of notes, with a different sound sample used for each range. Each of these ranges is referred to as a split, or key split.

Velocity splits refer to the use of different samples for different note velocities. Using velocity splits, one sample might be utilized if a particular note is played softly, where a different sample would be utilized for the same note of the same instrument when played with a higher velocity.

Note that the explanations above refer to the use of key splits and velocity splits in the sound synthesis process. In this case, the different splits utilize different samples of the same instrument sound. Key splitting and velocity splitting techniques are also utilized in a performance context. In the performance context, different splits generally produce different instrument sounds. For instance, a keyboard performer might want to set up a key split which would play a fretless bass sound from the lower octaves of his keyboard, while the upper octaves play the vibraphone. Similarly, a velocity split might be set up to play the acoustic piano sound when keys are played with soft to moderate velocity, but an orchestral string sound plays when the keys are pressed with higher velocity.

Aliasing Noise

The previous paragraph discussed the timbre changes which result from pitch shifting. The resampling techniques used to shift the pitch of a stored sound sample can also result in the introduction of aliasing noise into an instrument sound. The generation of aliasing noise can also limit the amount of pitch shifting which may be effectively applied to a sound sample. Sounds which are rich in upper harmonic content will generally have more of a problem with aliasing noise. Low-pass filtering applied after interpolation can help eliminate the undesirable effect of aliasing noise. The use of oversampling also helps eliminate aliasing noise.

LFOs for vibrato and tremolo

Vibrato and tremolo are effects which are often produced by musicians playing acoustic instruments. Vibrato is basically a low-frequency modulation of the pitch of a note, while tremolo is modulation of the amplitude of the sound. These effects are simulated in synthesizers by implementing low-frequency oscillators (LFOs) which are used to modulate the pitch or amplitude of the synthesized sound being produced. Natural vibrato and tremolo effects tend to increase in strength as a note is sustained. This is accomplished in synthesizers by applying an envelope generator to the LFO. For example, a flute sound might have a tremolo effect which begins at some point after the note has sounded, and the tremolo effect gradually increases to some maximum level, where it remains until the note stops sounding.

Layering

Layering refers to a technique in which multiple sounds are utilized for each note played. This technique can be used to generate very rich sounds, and may also be useful for increasing the number of instrument patches which can be created from a limited sample set. Note that layered sounds generally utilize more than one voice of polyphony for each note played, and thus the number of voices available is effectively reduced when these sounds are being used.

Polyphonic Digital Filtering for Timbre Enhancement

It was mentioned earlier that low-pass filtering may be used to help eliminate noise which may be generated during the pitch shifting process. There are also a number of ways in which digital filtering is used in the timbre generation process to improve the resulting instrument sound. In these applications, the digital filter implementation is polyphonic, meaning that a separate filter is implemented for each voice being generated, and the filter implementation should have dynamically adjustable cutoff frequency and/or Q.

For many acoustic instruments, the character of the tone which is produced changes dramatically as a function of the amplitude level at which the instrument is played. For example, the tone of an acoustic piano may be very bright when the instrument is played forcefully, but much more mellow when it is played softly. Velocity splits, which utilize different sample segments for different note velocities, can be implemented to simulate this phenomena. Another very powerful technique is to implement a digital low-pass filter for each note with a cutoff frequency which varies as a function of the note velocity. This polyphonic digital filter dynamically adjusts the output frequency spectrum of the synthesized sound as a function of note velocity, allowing a very effective recreation of the acoustic instrument timbre.

Another important application of polyphonic digital filtering is in smoothing out the transitions between samples in key-based splits. At the border between two splits, there will be two adjacent notes which are based on different samples. Normally, one of these samples will have been pitch shifted up to create the required note, while the other will have been shifted down in pitch. As a result, the timbre of these two adjacent notes may be significantly different, making the split obvious. This problem may be alleviated by employing a polyphonic digital filter which uses the note number to control the filter characteristics. A table may be constructed containing the filter characteristics for each note number of a given instrument. The filter characteristics are chosen to compensate for the pitch shifting associated with the key splits used for that instrument.

It is also common to control the characteristics of the digital filter using an envelope generator or an LFO. The result is an instrument timbre which has a spectrum which changes as a function of time. For example, It is often desirable to generate a timbre which is very bright at the onset, but which gradually becomes more mellow as the note decays. This can easily be done using a polyphonic digital filter which is controlled by an envelope generator.

THE PC TO MIDI INTERFACE AND THE MPU-401

To use MIDI with a personal computer, a PC to MIDI interface product is generally required (there are a few personal computers which come equipped with built-in MIDI interfaces). There are a number of MIDI interface products for PCs. The most common types of MIDI interfaces for IBM compatibles are add-in cards which plug into an expansion slot on the PC bus, but there are also serial port MIDI interfaces (connects to a serial port on the PC) and parallel port MIDI interfaces (connects to the PC printer port). The fundamental function of a MIDI interface for the PC is to convert parallel data bytes from the PC data bus into the serial MIDI data format and vice versa (a UART function). However, "smart" MIDI interfaces may provide a number of more sophisticated functions, such as generation of MIDI timing data, MIDI data buffering, MIDI message filtering, synchronization to external tape machines, and more.

The defacto standard for MIDI interface add-in cards for the PC is the Roland MPU-401 interface. The MPU-401 is a smart MIDI interface, which

also supports a dumb mode of operation (often referred to as "pass-through mode" or "UART mode"). There are a number of MPU-401 compatible MIDI interfaces on the market. In addition, many add-in sound cards include built-in MIDI interfaces which implement the UART mode functions of the MPU-401.

COMPATIBILITY CONSIDERATIONS FOR MIDI APPLICATIONS ON THE PC

There are two levels of compatibility which must be considered for MIDI applications running on the PC. First is the compatibility of the application with the MIDI interface being used. The second is the compatibility of the application with the MIDI synthesizer. Compatibility considerations under DOS and the Microsoft Windows operating system are discussed in the following paragraphs.

DOS Applications

DOS applications which utilize MIDI synthesizers include MIDI sequencing software, music scoring applications, and a variety of games. In terms of MIDI interface compatibility, virtually all of these applications support the MPU-401 interface, and most utilize only the UART mode. These applications should work correctly if the PC is equipped with a MPU-401, a full-featured MPU-401 compatible, or a sound card with a MPU-401 UART-mode capability. Other MIDI interfaces, such as serial port or parallel port MIDI adapters, will only work if the application provides support for that particular model of MIDI interface.

A particular application may provide support for a number of different models of synthesizers or sound modules. Prior to the General MIDI standard, there was no widely accepted standard patch set for synthesizers, so applications generally needed to provide support for each of the most popular synthesizers at the time. If the application did not support the particular model of synthesizer or sound module that was attached to the PC, then the sounds produced by the application might not be the sounds which were intended. Modern applications can provide support for a General MIDI (GM) synthesizer, and any GM-compatible sound source should produce the correct sounds. Some other models which are commonly supported are the Roland MT-32, the Roland LAPC-1, and the Roland Sound Canvas. The Roland MT-32 was an external MIDI sound module which utilized Roland's Linear Additive (LA) synthesis, and the MT-32 combined with an MPU-401 interface became a popular MIDI synthesis platform for the PC. The LAPC-1 was a PC add-in card which combined the MT-32 synthesis function with the MPU-401 MIDI interface. The Sound Canvas is Roland's General Synthesizer (GS) sound module, and this unit has become an industry standard.

Microsoft Windows and the Multimedia PC (MPC)

The number of applications for high quality audio functions on the PC (including music synthesis) grew explosively after the introduction of Microsoft Windows 3.0 with Multimedia Extensions ("Windows with Multimedia") in 1991. The Multimedia PC (MPC) specification, originally published by Microsoft in 1991 and now published by the Multimedia PC Marketing Council (a subsidiary of the Software Publishers Association), specifies minimum requirements for multimedia-capable Personal Computers. A system which meets these requirements will be able to take full advantage of Windows with Multimedia. Note that many of the functions originally included in the Multimedia Extensions have been incorporated into the Windows 3.1 operating system.

The audio capabilities utilized by Windows 3.1 or Windows with Multimedia include audio recording and playback (linear PCM sampling), music synthesis, and audio mixing. In order to support the required music synthesis functions, MPC-compliant

audio adapter cards must have on-board music synthesizers.

The MPC specification defines two types of synthesizers; a "Base Multitimbral Synthesizer", and an "Extended Multitimbral Synthesizer". Both the Base and the Extended synthesizer must support the General MIDI patch set. The difference between the Base and the Extended synthesizer requirements is in the minimum number of notes of polyphony, and the minimum number of simultaneous timbres which can be produced. Base Multitimbral Synthesizers must be capable of playing 6 "melodic notes" and "2 percussive" notes simultaneously, using 3 "melodic timbres" and 2 "percussive timbres". The formal requirements for an Extended Multitimbral Synthesizer are only that it must have capabilities which exceed those specified for a Base Multitimbral Synthesizer. However, the "goals" for an Extended synthesizer include the ability to play 16 melodic notes and 8 percussive notes simultaneously, using 9 melodic timbres and 8 percussive timbres.

The MPC specification also includes an authoring standard for MIDI composition. This standard requires that each MIDI file contain two arrangements of the same song, one for Base synthesizers and one for Extended synthesizers. The MIDI data for the Base synthesizer arrangement is sent on MIDI channels 13 - 16 (with the percussion track on channel 16), and the Extended synthesizer arrangement utilizes channels 1 - 10 (percussion is on channel 10). This technique allows a single MIDI file to play on either type of synthesizer.

Windows applications generally address hardware devices such as MIDI interfaces or synthesizers through the use of drivers. The drivers provide applications software with a common interface through which hardware may be accessed, and this simplifies the hardware compatibility issue. Before a synthesizer is used, a suitable driver must be installed using the Windows Driver applet within the Control Panel. The device drivers supplied with Windows 3.1 include a driver for the MPU-401/LAPC-1 MIDI interface, and a driver for the original AdLib FM synthesizer card. Most other MIDI interfaces and/or synthesizers are shipped with their own Windows drivers.

When a MIDI interface or synthesizer is installed in the PC and a suitable device driver has been loaded, the Windows MIDI Mapper applet will appear within the Control Panel. MIDI messages are sent from an application to the MIDI Mapper, which then routes the messages to the appropriate device driver. The MIDI Mapper may be set to perform some filtering or translations of the MIDI messages in route from the application to the driver. The processing to be performed by the MIDI Mapper is defined in the MIDI Mapper Setups, Patch Maps, and Key Maps.

MIDI Mapper Setups are used to assign MIDI channels to device drivers. For instance, If you have an MPU-401 interface with a General MIDI synthesizer and you also have a Creative Labs Soundblaster card in your system, you might wish to assign channels 13 to 16 to the Ad Lib driver (which will drive the Base-level FM synthesizer on the Soundblaster), and assign channels 1 - 10 to the MPU-401 driver. In this case, MPC compatible MIDI files will play on both the General MIDI synthesizer and the FM synthesizer at the same time. The General MIDI synthesizer will play the Extended arrangement on MIDI channels 1 - 10, and the FM synthesizer will play the Base arrangement on channels 13-16. The MIDI Mapper Setups can also be used to change the channel number of MIDI messages. If you have MIDI files which were composed for a General MIDI instrument, and you are playing them on a Base Multitimbral Synthesizer, you would probably want to take the MIDI percussion data coming from your application on channel 10 and send this information to the device driver on channel 16.

The MIDI Mapper patch maps are used to translate patch numbers when playing MPC or General MIDI files on synthesizers which do not use the General MIDI patch numbers. Patch maps can also be used to play MIDI files which were arranged for non-GM synthesizers on GM synthesizers. For example, the Windows-supplied MT-32 patch map can be used when playing GM-compatible .MID files on the Roland MT-32 sound module or LAPC-1 sound card.

The MIDI Mapper key maps perform a similar function, translating the key numbers contained in MIDI Note On and Note Off messages. This capability is useful for translating GM-compatible percussion parts for playback on non-GM synthesizers or vice-versa. The Windows-supplied MT-32 key map changes the key-to-drum sound assignments used for General MIDI to those used by the MT-32 and LAPC-1.

Some MIDI applications, such as MIDI sequencer software packages, can be set to make use of the MIDI Mapper, or to address the device driver directly (bypassing the MIDI Mapper). Other Windows applications always utilize the MIDI Mapper.

SUMMARY

The MIDI protocol provides an efficient format for conveying musical performance data, and the Standard MIDI Files specification ensures that different applications can share time-stamped MIDI data. The storage efficiency of the MIDI file format makes MIDI an attractive vehicle for generation of sounds in multimedia applications, computer games, or high-end karaoke equipment. The General MIDI system provides a common set of capabilities and a common patch map for high polyphony, multi-timbral synthesizers. General MIDI-compatible Synthesizers employing high quality wavetable synthesis techniques provide an ideal MIDI sound generation facility for multimedia applications.


SMART
Analog
™