WHAT IS MIDI?

The first ideas for a Musical Instrument Digital Interface and communication protocol were developed by Dave Smith and Chet Wood and were presented on the 70th AES (Audio Engineering Society) convention in 1981. The name was abbreviated to MIDI by synthesizer companies who committed themselves to a normalized standard (unique on its own). In 1983 the first MIDI instruments were presented.
The detailed Midi 1.0 specification is the document that exactly describes MIDI, although many updates and extras were added and some manufacturers made their own little variations. The main part of this specification describe the numeric codes of the different "musical" events that can be send and received. The description of the hardware is only 2 pages, because it is simple and straightforward.

THE SCHEME OF A MIDI INTERFACE

The interface is a) **digital**, b) **serial** c) **unbalanced**, and d) not **bi-directional** (two-way).

In fact we are talking about communication between microprocessors and that is always with a stream of digital numbers.

Serial means numbers go bit by bit, one after the other, so principally one needs only 3 wires for transport, one ground and one send and one receive wire. Midi numbers are 8 bits wide (1 byte), so the maximum number is 256 decimal. Another way of data transport is parallel, where every bit has it's own wire and which is potentially faster (one byte in one go), but cables are more complex and vulnerable. At this moment, serial interfacing is clearly winning (USB, Firewire, Ethernet) and speed is hardly a limitation.

Unbalanced means the data transport can not profit from noise suppression that is possible with balanced data transport by means of a differential input circuit. In theory this means the maximum cable length is limited to 15 meters, although in practice lengths of 60 meters is possible under good circumstances. Special care is taken, by use of an opto coupler, to avoid ground loops.

Bi-directional means the interface is able to send and receive at the same time

Important is to be aware of the fact that Midi is only a stream of numbers that is to be interpreted by the microprocessor(s) and software at both ends. It's in fact a matter of making appointments.

In **fig.1** one sees the set-up of a midi interface. A Midi device can have a Midi-In, Midi-out and a Midi-through(Thru) connection (not always all of them are available). The Midi-in receives Midi code, Midi-through is an electronically buffered copy of the Midi-in and Midi-out delivers the Midi code that is generated by the device itself.

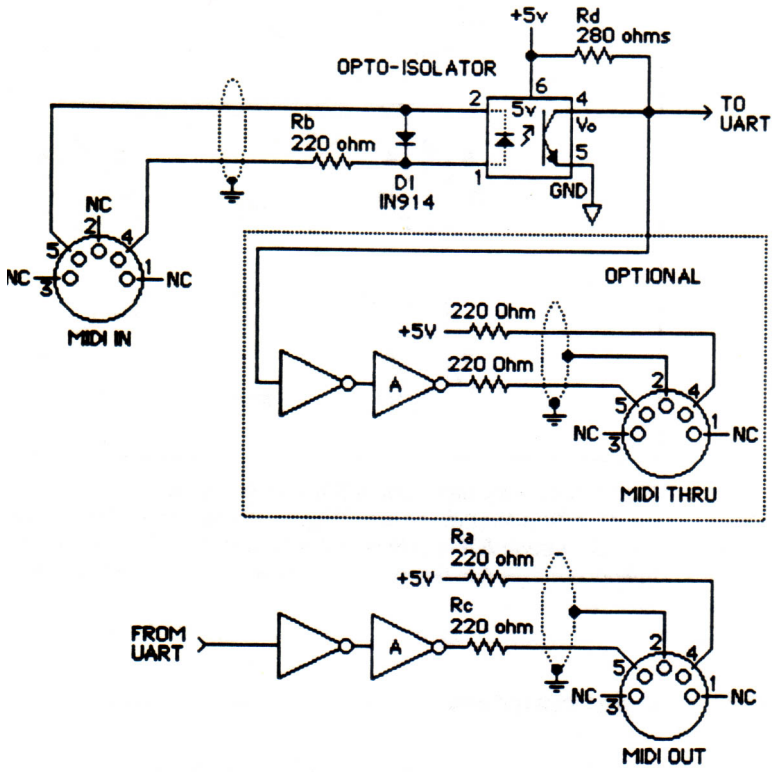


fig. 1

The connectors are (alas) of the very old fashioned female din busses type. The electrical signal is in fact a 5 mA current-loop that occurs when Midi-in and – Out are connected wrong (in to in, out to out) connections will also do no harm because of that. From the In connector the signal is fed to the opt-coupler and then on to a so called UART (serial to parallel converter) and hence fed to the microprocessor (= synthesizer or whatever). The Output has no opto-coupler (not necessary) and has a direct, buffered connection with the output of the UART. By connecting Midi-out to Midi-In and vice versa one is in fact connecting the microprocessors of two devices. These days the din busses are often replaced by a USB connector, but that is in fact not an official midi standard.
The Midi cable is a shielded, two core (twisted) cable of good quality and low capacitance with at its end a 5-pin male din plug. Although it's a 5 pin plug, pin 1 and 3 (the outer ones) are officially not used. Clearly visible is the opto-coupler.

The number of bits that are send per second over the cable is called BAUDRATE. 1 Baud means 1 bit (not byte!) per second. The BAUDRATE of midi is 31,25 kBAUD (1 MHz divided by 32) which is rather slow, but the limit those days (USB 2.0 is now 480Mbit/s, USB 3.0 will be 5Gbit/s).It is more or less acceptable as we will see later on.

THE MIDI CODES

Midi numbers consist of groups of 10 bits, the 8 bits (byte) of the Midi code and a start (always a 1) and a stop (always a 0) bit. These two bits are only there for consistency check, they have no Midi meaning. A small calculation teaches us, that Midi allows transportation of 31250 / 10 = 3125 bytes (Midi codes) per second. From now on, we will only talk about the 8 bits of the Midi code.
To make things easier we write bits (ones and zero's) of a byte one after the other, where the left most bit is called the MSB (most significant bit) because it has the most impact on the value and the right most bit the LSB (least significant bit).

In Midi there's a distinction between 2 kind of bytes: the STATUS- and the DATA byte. The STATUS byte tells the machine what has to happen (an event) and always start with an MSB = 1. A STATUS byte is followed by one, two or three (and sometimes even more in the case of system exclusives, about that later on)) DATA bytes that always have an MSB = 0 and that describe where and how much. In this way a microprocessor can easily and very fast distinguish between action and amount of action.

Example: 10110000 = statusbyte
00110000 = databyte

Together these bytes form a midi message, every new message starts hence with a STATUS byte!

Example:

MESSAGE	STATUS BYTE	DATA BYTE(S)	
Note-Off Event	1000bbbb	0kkkkkkk	0vvvvvvv
Note-On Event	1001bbbb	0kkkkkkk	0vvvvvvv

MIDI CHANNELS

An important quality of Midi is the possibility of addressing messages to 16 different channels. Channels are not to be seen as separate connections, but more like addresses for a postman: he puts a letter (message) in the mailbox that corresponds with the address (channel)on the letter. This also goes for midi messages: a midi apparatus only listens to messages that have the same channel number as the apparatus. So all messages go to (all) the connected devices, but every device picks out the message that has his channel-number. One uses the last four bits of a STATUS byte to this, so this gives us indeed 16 possibilities. A Midi device can be set up with one of the 16 Midi-channels or, as an extra, can be in omni mode, which means it listens to all channels! It looks like this:

1aaabbbb

The STATUS byte always begins with a 1, then come 3 bits (aaa) that tell is what kind of event and after that the four bits (bbbb) of the channel-number. 4 bits means, once more, 16 possible variations, so 16 channels.

e.g.: 10000000 means Key Off for channel 1
10001000 means Key Off for channel 9
10001111 means Key Off for channel 16

(N.B. counting is actually from 0 to 15)

TYPES OF MIDI MESSAGES

MIDI Messages can be divided into:

CHANNEL MESSAGES		SYSTEM MESSAGES		
Channel Voice	Channel Mode	Syst. Real Time	Syst. Common	Syst. Exclusive
Key ON	Local Contr.	Timing Clock	Song Pos.	dep. manuf.
Key Off	Omni	Start	Song sel.	
Program Ch.	Poly	Stop	Tune Req.	
Control Ch.	Mono	Continue	End	
Channel Press.	All Off	Active sens.		
Poly Press.		Reset		

(see also the Midi 1.0 specs in the appendix)

I. CHANNEL MESSAGES

I.1 Channel Voice Messages

a) NOTE ON en NOTE OFF events

A Note On event message is generated when a key on a keyboard is pressed. A Note Off event message is generated when a key on a keyboard is let loose.

MESSAGE	STATUS BYTE	DATA BYTE(S)	
Note-Off Event	1000bbbb	0kkkkkkk	0vvvvvvv
Note-On Event	1001bbbb	0kkkkkkk	0vvvvvvv

To start with, look at the STATUS byte: for key of aaa = 000 and for key on aaa = 001. After this byte always 2 data bytes follow (start with a 0, then followed by 7 data bits). In the first data bytes the 7 k's indicate 128 note numbers (see appendix for details). After this data byte is another data byte where the 7 v's express velocity = loudness. Mathematically we could say that every v-bit stands for 6 dB dynamics, so in this case a maximum of 42 dB dynamics is possible. Keyboards without "velocity" keys always give 64 as velocity.
At Key Off, vvvvvvv expresses how fast a key is let loose. Not many keyboards have this implemented. Velocity 0 actually could be seen as a key off, it is used like this by some manufacturers and it has an advantage as we see later on. All Channel Messages can be sent and received. When a synthesizer receives, fir instance, these bytes:

```
          10010001          00111100              01111111
          : :   :           :   :                 :   :
status----' :   :  Data-----'   :       Data------'   :
Note On-----'   :  Key 60 ------'       Velocity 127--'
Channel 2(1)----'
```

it means that key 60 = C will sound at its loudest.

b) PROGRAM CHANGE

A synthesizer always has a choice of many sounds (voices). They can be chosen with this Midi message:

Program Change	1100bbbb	0nnnnnnn

nnnnnnn in the data byte stands for the sound/program choice. A maximum of 127 choices is possible
Pressing a program-change button on a keyboard will generate this program change message. With Midi controllable effect devices one should be able to choose between different effect set-ups.

c) CHANNEL PRESSURE

Many synthesizers have the possibility of measuring key pressure to give the possibility of pitch, level or timbre variations in a sound. Beware, this variation is, in this case, modulated on all sounding keys. (opposed to Polyphonic Key Pressure). The MIDI message is:

Channel Pressure: 1101bbbb 0vvvvvvv

The amount of pressure can be read from vvvvvvv in 128 steps.

d) POLYPHONIC KEY PRESSURE

Some synthesizers are capable of measuring key pressure under every separate key. One can measure pressure variations polyphonically. The message is:

POLY KEY PRESSURE: 1010bbbb 0kkkkkkk 0vvvvvvv

in this case one needs a key number, so kkkkkkk expresses this and vvvvvvv expresses the amount of pressure. These messages are sent as long as the key-pressure is varied. How often per second, depends on the key-scan-rate and can cause large flows of midi data. Not many keyboards have this item implemented.

e) PITCH BEND

almost all keyboards have 1 or more continues variable control wheels, one of them is the Pitch Bend Wheel another is the Modulation Wheel. The Pitch Bend Wheel is able to alter the pitch of the played note(s) of the whole keyboard continually or in steps, up or down. Pitch Bend has its own separate status byte. In contrary the Modulation Wheel hasn't, but here another way is used to be able to read out the wheel

Pitch Bend: 1110bbbb 0xxxxxxx 0yyyyyyy

here the two data bytes form together a 14 bits number yyyyyyy is the Most Significant part, xxxxxxx the Least significant.

f) CONTROL CHANGE

To be able to use many controllers (a.o. a Modulation wheel), the Control Change Messages were introduced. They use a nice trick to be able to expand the range of "events" of a Midi device.

Control Change: 1011bbbb 0ccccccc 0vvvvvvv

The value ccccccc can vary from 0 tot 127 but now describes what controller/function is used and vvvvvvv describes the variation. The used controllers are not the same on every Midi device, but some are standard:

Standard control-codes:

Control-number	Function	
0	Pitch Bend	MSB (some manufacturers)
1	Modulation wheel	MSB
2	Breath controller	MSB
4	Foot controller	MSB
5	Portamento time	MSB
6	Data Entry	MSB
7	Volume	MSB
32	Pitch Bender	LSB (some manufacturers)
33	Modulation wheel	LSB
34	Breath controller	LSB
35	Foot controller	LSB
36	Portamento time	LSB
37	Data Entry	LSB
38	Volume	LSB
64	Sustain	
65	Portamento	
66	Sostenuto	
67	Soft pedal	
96	Data +	
97	Data -	
124	Local/Remote Keyboard Control (toggle)	
125	Omni Mode Select/ All Notes Off }	
126	Mono Mode Select/ All Notes Off } v must be 0	
127	Poly Mode Select/ All Notes Off }	

Normally controllers have a 7 bits resolution (128 steps). One uses then only the MSB(byte) of that controller, where the second data byte indicates the value of the variation (64 is the middle). When one wants to achieve a higher resolution, the second (LSB) part is also used to form a 14 bit value (more than 16000 steps)
There are some restrictions, CC 98 to 122 are not yet defined (so free) and 122 to 127 are specials and typically for keyboard. 64 to 97 are mainly used for switches, where 0 is off and 127 is on.

I.2 CHANNEL MODE MESSAGES

Until now all Midi messages had to do with sound producing or -manipulating parameters. But there is a group of messages that don't do that, but perform all kind of operational tasks. These messages are described as follows and are part of the midi controller messages:

CHANNEL MODE MESSAGE: 1011bbbb 0ccccccc 0vvvvvvv

where ccccccc goes from van 122 t/m 127 (so bits 2 to 5 are all 1).

Een lijst van de diverse modes is:

MESSAGE	STATUS BYTE	DATA BYTES	
Local Control Off	1011bbbb	01111010	00000000
Local Control On	1011bbbb	01111010	11111111
All Notes Off	1011bbbb	01111011	00000000
Omni Mode Off	1011bbbb	01111100	00000000
Omni Mode On	1011bbbb	01111101	00000000
Mono Mode On	1011bbbb	01111110	0zzzzzzz
Poly Mode On	1011bbbb	01111111	00000000

(N.B. zzzzzzz = number of active mono channels)

A rather confusing part is the voice mode (Omni, Mono and Poly)
To understand this, one has to realise that most synths are polyphonic (from 8 to 128 voices) and are able to produce more sounds at a time (multitimbral).
When a keyboard is in Omni-mode it "listens" to all the Channel Voice messages. It's not able to choose a channel.
In the Poly-mode a synthesizer only listens to messages of the channel(s) that it's tuned to and it divides it to the possible voices of a synthesizer.
In the Mono-mode, 1 voice gets information from 1 channel. Also Control Change and Program Change Channel are Voice messages which means that a Mono Mode instrument can be multitimbral, and that every voice can be multitimbrally controlled.

Also combinations are possible:

Channel Mode Combinations

Combination	Description
Omni On/Poly	The receiving side is Omni and receives all info from all channels, the sending side is Poly and sends it's info over channel n.
Omni On/Mono	Receives info from all channels en divides them one by one to the possible voices. The send side is as with poly.
Omni Off/Poly	The synthesizer receives and sends only via channel n
Omni Off/Mono	The voices are divided over a number of consecutive channels for both send and receive. Every message gets one channel.

Channel n is called the basic channel and is chosen manually (this can often only be done by hand via a menu, not with a midi message). Not all of the above mentioned combi's are implemented on every synth, but multi-timbral is these days. A special Mono mode is able to split up a synth in groups of voices that each on their own, listen to a specific channel (but never more than the maximal amount of voices).

RUNNING STATUS

In practice, long series of events of the same status are sent one after another. When one for instance plays on a keyboard mostly only key-ons and key-offs are send. So why send the status byte every time anew? Certainly when one would define key-off as a key-on with velocity 0, one would save almost 1/3 of the amount of data. So remembering the status byte until it changes can be "time-saving". This is called RUNNING STATUS.

II. SYSTEM MESSAGES

System Messages are meant for all devices in a MIDI system and therefore have no channel division possibility. The status byte always begins with 1111 (four ones). There are three types of System Messages: **Real-Time, Common en System Exclusive.**

II.1. SYSTEM REAL-TIME MESSAGES

Real Time Messages only make sense when there's a necessity for time synchronisation so when using a sequencer (a digital multi-track mid-message recorder). All Real-Time Messages are only 1 byte and have the form:

Real-Time Message: 1111dddd

where dddd is not the channel number but the 16 possible Real-Time Messages where the next 6 are mainly used:

Timing Clock	11111000
Start	11111010
Continue	11111011
Stop	11111100
Active Sensing	11111110
System Reset	11111111

The Timing Clock byte is continually generated (when switched on), more precise 24 times per quarter note.
Start begins a (recorded) sequence from the beginning, Stop stops the sequence and continue continues the sequence from stopping point.
Active Sensing is generated by some synthesizers (continually, 3 times/sec) and gives an indication that a device is on and connected.
System Reset initialises a system to a start-up status.
Midi clock should not be confused with MTC (Midi Time Code) which is the Midi version of SMPTE. About MTC later on more.

II.2. SYSTEM COMMON MESSAGES

There are four System Common Messages, two of them are used to "wind" to certain positions in a sequence.

a) Song Position Pointer: 11110010 0LLLLLLL 0hhhhhhh

b) Song Select: 11110011 0sssssss

Song Position Pointer gives the opportunity to start or continue the sequence from a certain point. hhhhhhh en LLLLLLL are respectively the MSB en LSB of a 14 bit number and represent the number of beats (max. 16384). 1 beat contains 6 MIDI clocks and 24 MIDI Clocks is 1 quarter note. So 4 in a 4/4 measure give a maximum of 16384/16 = 1024 4/4 measures per song.
Song Select gives the possibility of choosing a maximum of 128 songs from a sequencer library. sssssss gives the song number.

The other two System Common Messages are:

c) Tune Request 11110110

d) End Syst. Excl. 11110111

Tune Request initialises, only with Analog synthesizers, a routine that tunes the oscillators.
End System Exclusive is a sign, that the System Exclusive Message has ended.

II.3. SYSTEM EXCLUSIVE MESSAGES

Registered digital-music-instrument manufacturers may use System Exclusive Message to send specific device dependent information. The format is:

```
SYSTEM EXCLUSIVE:        11110000
                              0iiiiiii
                              0xxxxxxx
                                 .
                                 .
                                 .
                                 .
                                 .
                              0xxxxxxx
                         11110111
```

The first byte (as always) is the status byte 11110000 for start of system exclusive, the second (data)byte has the identification number of the manufacturer given by the MIDI Manufacturer's Association or the Japan MIDI Standards Association. Some examples are:

```
Kurzweil:  7
Roland:   65
Yamaha:   67
```

After this, an undefined amount of xxxxxxx data bytes follow. They all contain values of parameters in an order that the manufacturer determinates. To end the string, the last byte is the End Of Exclusive (System Common) byte 11110111.
These System Exclusive data bytes are often very long and rather hard to figure out. Special "Exclusive Data sheets" should help the software developers to make "editors" that make changing parameters of a Midi device easier for a "normal" user. . An example of such a datasheet is in the Appendix.
It is also possible to "dump" all parameters of a Midi device in one go, a so called Midi Data Dump. These strings can be saved as a file on a computer and recalled later.

MIDI NETWORKS

The simplest network can be found in fig. 2, a so called Master/Slave network. A master keyboard "plays" one or more sound modules. Since one man, most of the times, can't produce enormous amounts of data, this shouldn't cause large delays. Midi out of the keyboard is connected to the Midi in of device 1, then midi thru of that device goes to midi in of device 2 and midi thru of device 2 goes to midi in of 3 and so on. This is also often called a (daisy) chain network.
A bit more complex is the typical drummachine set up of fig 3. Here a drumsynth synchronises (syncs) the sequencer via midi clock. At the same time Channel Voice Messages, Real Time Messages are send. Again, when there's a lot of Channel Voice Information noticeable sound delay can be heard.

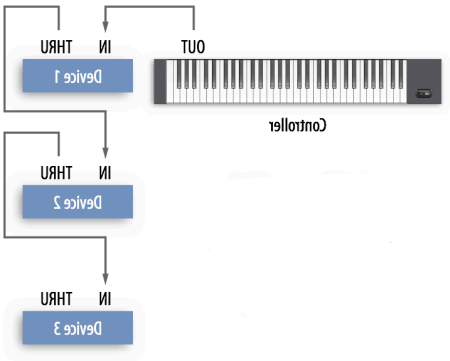


fig. 2

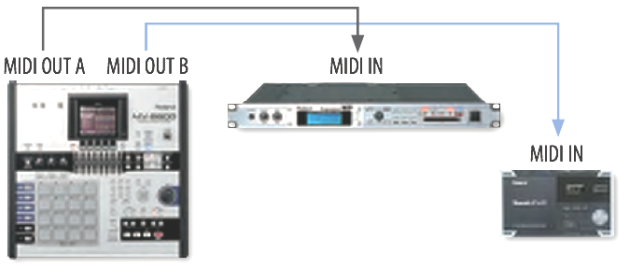


fig. 3

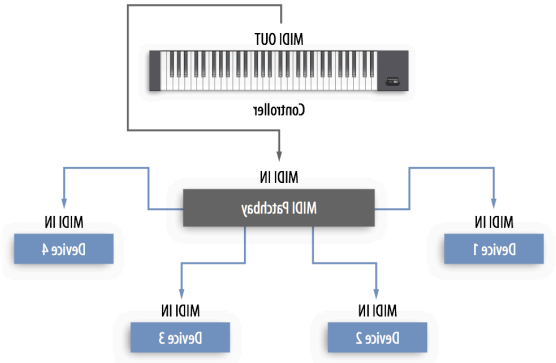


fig. 4

Another type of network can be found in fig. 4. The sequencer/keyboard is used to record and play back performance data for the used synthesizers. Because these sequencer can have many tracks, information streams can be very dense, certainly when all data is sent over one midi output. Preferably one uses in these cases, midi interfaces with more (4 - 8) outputs, actually for every synth an output. Of course data streams to the interface are as dense as before, but the speed of the transport between computer and Midi interface can be much higher and is not restricted to 31250 kBaud. Signals are divided to the different outputs at normal Midi speed by a microprocessor in the (intelligent) interface. Per Midi out the density is of course less now.
Same story goes for Midi in streams. Intelligent Midi interfaces often also have multi Midi inputs.

These data reductions can not be achieved by so called midi splitters, passive Midi out dividers that one can buy for little money and that have no microprocessor. These networks are often called star networks.
A special case is the Midi in merger. One can not just add/sum electronically, outputs of 2 or more Midi devices just like that. One needs an intelligent device (microprocessor + software) to find out which messages belong together and it has to keep them together when merging them with the data of the other device.

Combinations of above mentioned networks are always allowed without limitations, but the self-made hard-soldered splitters can cause the current loop of the Midi output of a device to be overloaded. Remember daisy-chaining is easy to cause delays.

DELAYS IN MIDI SYSTEMS

A MIDI system has unavoidable delays. Some are caused by Midi itself, while others are caused by slowness of the microprocessor and software of a device. To start with the Master Keyboard, there we already have delays caused by scanning the keys, converting the velocity into digital numbers and delays of the UART.
Then there are the transmission delays. A chord of e.g. 6 notes consists of 13 (running status) or 18 bytes. The delay will be 13/3125 = 4 mS or 18/3125 = 6 mS. A third cause of delay might be the Message priority organisation. All (almost all at the same time) incoming information has to be sorted out to give a desired priority (e.g. note numbers should have utter priority) and then to be send in the right order to the Midi device or sequencer.. A total delay of 10 mS is normal and acceptable, but when it is more than 20 mS one can hear the delay of the notes (a chord becomes an arpeggio).How much delay is acceptable depends on the type of sounds and the type of music. Phase-sync is impossible (smallest delay is 1 mS).If it's important that a certain rhythmical pattern with fast attacks, that come from different devices, special care has to be taken, so don't use daisy chains but use multi output midi interfaces instead. Sounds with slow attacks are of course not bothered too much from delay, as long as it is not more than 30 mS.
To have as little delay as possible, one should prevent to have all channel messages to come out of 1 Midi out. Use an active splitter or a multi-output Midi-interface. Also continuous Midi data generators like Pitch wheel, Key pressure and other controllers should be kept to a minimum. Often sequencer software is able to give priority to the important (note) or to quantisize Note messages to keep notes together. All in all it is sometimes amazing how well everything is kept "together" when 1 computer sends it's information to e.g. 9 synthesizer, 2 samplers and a drum synth. Software has been working hard in those cases.

MIDI DATA RESOLUTION

We already saw the item of resolution of controllers, e.g. Pitch wheel. A 7 bit resolution in 1 Data byte would give only 128 steps , 14 bits (2 Data bytes) would give 16.384 steps. So when is such a big resolution needed? In fact the Midi specs say nothing specifically about the wanted resolution of controllers, but the fact that MSB and LSB Data bytes are defined for some standard controllers, means that they also found 128 steps for continuous variations is too small. Certainly in pitch variations there would be too many pitch deviations with 128 steps, already within one octave. 14 bits give an accuracy of 1 cent within an octave. Another thing is, that in a small resolution one can hear each separate step as clicks!
Some critical parameters are the ones that have to do with frequencies of oscillators and filters, attack and decays of envelope generators. In the software of the devices one is able to interpolate the 128 steps, which means the steps are actually hardly hearable.

FUTURE CHANGES IN RESOLUTIONS

"MIDI has worked fantastically for more than 25 years, but with today's computers and embedded microprocessors we can now do much more than MIDI was originally designed to do", said MMA President and CEO Tom White. "This new protocol could encourage market growth through more expressive products, improved ease of use, and new and innovative applications. Plus new HD devices and software would be designed to be compatible with all of the great MIDI hardware and software now and for the future."
The proposed changes would increase the number of MIDI Channels and Controllers, and provide greater resolution in data values for all of the current MIDI 1.0 messages. Moreover, all of this would be accomplished with single messages, as opposed to the compound messages often used in MIDI 1.0, which means using and editing MIDI data will be far easier for both developers and users. The new protocol could also support the creation of entirely new messages that were not practical with the MIDI 1.0 protocol.
"At this point our 'HD Protocol' is still under development, but we've seen a lot of interest from both hardware and software developers," said White. "Our policy is not to discuss MMA Specifications publicly until they're officially adopted, but in this case we want to make sure that all qualified companies know what we are thinking about so they have the opportunity to participate before the first version is published."
The original MIDI 1.0 Specification, developed in 1983, has been the foundation for interoperability of digital musical instruments for 25 years. The initial "MIDI 1.0 Specification" contained the rules for remote control of keyboard devices, but over the years additional specifications were developed for file exchange, sound exchange, synthesizer design, and new applications such as stage lighting and ring-tones. Today the term "MIDI" applies to the wide variety of file formats, applications, and device specifications defined by the MIDI Manufacturers Association.
The MIDI Manufacturers Association is an industry non-profit organization that is responsible for maintaining and extending MIDI. Formed in 1985 by the original developers of the MIDI 1.0 Specification, the MMA provides a forum where companies using MIDI can cooperate and collaborate to make their equipment interoperable.

OTHER MIDI CONTROLLERS

Until now we only have been talking about synthesizers as Midi device, but by now almost everything is "midified". Pitch to midi converters make it possible to make midi data out of voice/instrument input via microphone or direct in (el. guitars). Also light-, movement- or distance measuring devices (Human Midi Controller) are made to generate midi data (a.o. by the department of electronics in the Royal Conservatory or by Steim) and can be used to control creative processes like light/video/apparatus modulation.
Any other data (text, video, pictures etc.) can be changed into midi data, so creative possibilities are almost endless. Also light mixer/dimmers are midified, although the light-world has its own midi called DMX, which uses the same principles but has, of course, different definitions and implementations.

MIDI EN EFFECT- AND OTHERS DEVICES

Midi is such a reliable communication interface, that also non (or better, less) musical devices were accommodated with Midi. Of course Note-on and off messages were not implemented, but Programchange could be handy to be able to select different set ups of e.g. a digital reverb. But mostly it gives the possibility to change/edit parameters of devices through Sysex editors so it doesn't have to be done on the front of a device through multiple button pressings and via an almost unreadable mini screen. Also real-time parameter changes are possible, which give creative people almost endless variation possibilities. The connection of parameters and midi codes is often editable in matrixes or lists.
Digital audio mixers are often also provided with midi connections

General Midi Common Misconceptions

Although the GM and GM2 specifications are dependent on the basic MIDI 1.0 protocol specification, they are separate standards from MIDI 1.0. As a result, MIDI products may legitimately implement MIDI 1.0 but not GM and/or GM2. Although GM is an important feature for MIDI content interoperability across multiple players, many important MIDI applications do not require such interoperability. For example, MIDI and the SMF format are used in professional music recording production where the MIDI file content will never be distributed and custom or specialized synthesizers are used much more commonly than GM or GM2. As a direct consequence, not all SMF content is authored for GM or GM2 synthesizers. Because playing any SMF or MIDI message stream on a different synthesizer(s) than originally intended risks the generation of unintended and incorrect sounds, it is not generally safe to merely assume that any given MIDI message stream or MIDI file is intended for GM or GM2 synthesizers. In particular it is frequently assumed, incorrectly, that all or nearly all SMF content necessarily relies on the player using a GM or GM2 synthesizer, however because there is no such dependency in the actual MMA/AMEI specifications and it is also quite legitimate for SMF content to be written for non-GM synthesizers, this assumption is not reliable. Unfortunately, there is currently no technical standard for indicating in advance what kind of synthesizer(s) a given SMF or MIDI message stream is intended to drive (with the exception of RTP MIDI and the audio/sp-midi MIME type definition).

GS and XG

To improve upon the General MIDI Standard and take advantage of the advancements in newer synthesizers, both Roland (GS) and Yamaha (XG) introduced proprietary specifications and numerous products with stricter requirements, new features, and backward compatibility with the GM specification. GS and XG are not compatible with each other, are not official MMA/AMEI MIDI standards, and adoption of each has been generally limited to the respective manufacturer.

General MIDI Level 2

Later after the success of General MIDI was firmly established, companies in Japan's Association of Musical Electronics Industry (sic) (AMEI) developed General MIDI Level 2 (GM2), incorporating and harmonizing aspects of the Yamaha XG and Roland GS formats, further extending the instrument palette, specifying more message responses in detail, and defining new messages for custom tuning scales and other new functionality, thus improving the sound editing features and the quality. For these new enhancements to be possible new messages had to be integrated into the MIDI specification; these enhancements consist of Controllers, RPNS, MIDI tuning and Universal system exclusive messages. The GM2 specs are maintained and published by the MMA and AMEI. General MIDI 2 was introduced in 1999 and is commonly implemented in some newer synthesizers.

SP-MIDI

Later still, GM2 became the basis of the instrument selection mechanism in Scalable Polyphony MIDI (SP-MIDI), a MIDI variant for mobile applications where different players may have different numbers of musical voices. SP-MIDI is a component of the 3GPP mobile phone terminal multimedia architecture, starting from release 5. GM, GM2, and SP-MIDI are also the basis for selecting player-provided instruments in several of the MMA/AMEI XMF file formats (XMF Type 0, Type 1, and Mobile XMF), which allow extending the instrument palette with custom instruments in the Downloadable Sound (DLS) formats, addressing another major GM shortcoming.

This table provides summary of comparison of various MIDI enhancement standards by various parameters.

	MT-32	GM	GS	XG level 1	XG level 2	XG level 3	GM level 2
Entry date	1987	1991	1991	1994	1997	1998	1999
Organization	Roland	MMA	Roland	Yamaha			MMA
Minimum equipment requirements							
Simultaneous voices	8 or more (up to 32 partials)	24	24	32	64	128	32
Simultaneous melodic voices	N/A	16	16	N/A XG synths have no separate limits on melodic / percussion sounds			16
Simultaneous percussion voices		8	8				16
MIDI channels allocation	8 channels, 1 rhythm channel	16 channels, #10 is fixed for percussion	16 channels, one channel can be set to play drum kits	16 channels, every channel can play drum kits with Bank Select MSB (CC#0) set to 7FH	32 channels, 2 ports, drums as in XG level 1	64 channels, 4 ports, drums as in XG level 1	16 channels, #10 and #11 are used for percussion
Channel recommendations				#1: melody; #2: melody (duet); #3: bass; #4: pad; #5: riff; #10: drums[1]			
Sounds banks available							
Melodic instruments	128	128	226	480	1074	1149	256
Drum kits	1	1	8 + 1 SFX kit	9 + 2 SFX kits	34 + 2 SFX kits	35 + 2 SFX kits	9
Drum sounds per kit	30	47	61	72			61
Controls available							
Special CC[2]	2[3]	6 (MT32+4)					6 (GM)
Parametric effect CC[4]	4[3]	5	26 (GM+11)				12 (GM+7)
RPNs	0[3]	5[5]					6 (GM+1) [5]
SysEx messages		2					14

Alternative Tunings

By convention, most MIDI synthesizers generally default to the conventional Western 12-pitch-per-octave, equal temperament tuning system. This tuning system makes many types of music inaccessible, because they depend on different intonation systems. To address this issue in a standardized manner, in 1992 the MMA ratified the MIDI Tuning Standard, or MTS. Instruments that support the MTS standard can be tuned to any desired tuning system by sending the MTS System Exclusive message (a Non-Real Time Sys Ex).
The MTS SysEx message uses a three-byte number format to specify a pitch in logarithmic form. This pitch number can be thought of as a three-digit number in base 128. To find the value of the pitch number p that encodes a given frequency f, use the following formula:

For a note in A440 equal temperament, this formula delivers the standard MIDI note number as used in the Note On and Note Off messages. Any other frequencies fill the space evenly. While support for MTS is at present not particularly widespread in commercial hardware instruments, it is nonetheless supported by some instruments and software, for example the free software programs TiMidity and Scala, as well as other microtuners.

MIDI Show Control

Main article: MIDI Show Control
The MIDI Show Control (MSC) protocol (in the Real Time System Exclusive subset) is an industry standard ratified by the MIDI Manufacturers Association in 1991 which allows all types of media control devices to talk with each other and with computers to perform show control functions in live and canned entertainment applications. Just like musical MIDI (above), MSC does not transmit the actual show media — it simply transmits digital data providing information such as the type, timing and numbering of technical cues called during a multimedia or live theatre performance.
MSC can be seen with the creation of a Halloween haunted mansion designed by Brent Ross at his Mountain View, CA home in October 2007. The haunted mansion was solely run on MIDI in which Ross converted “real-time” recordings of MIDI musical notes and converted them into electrical signals to operate and turn pneumatic valves on and off. Using Cubase software for MIDI sequencing, Ross’s MIDI entertainment haunted mansion performance could be played, recorded and edited by manually pushing a specific control button. Ross could then access his “MIDI to switch” technology, further allowing him to send MIDI messages to turn a “note on or off” while controlling the activation of the various props, movements, theatrical lighting, and sounds for entertainment.

MIDI Machine Control

MIDI Machine Control, or MMC, a subset of the MIDI specification, provides specific commands for controlling recording equipment such as multi-track recorders.
MMC messages can be sent along a standard MIDI cable for remote control of such functions as Play, Fast Forward, Rewind, Stop, Pause, and Record. These are "System Exclusive" (SysEx) messages.

MIDI Universal Real Time SysEx Commands

All numbers are in hexadecimal notation. SysEx message format:
F0, 7F, nn, sub-ID, data, F7
nn = channel number, 00 to 7F; 7F = global
sub-IDs:
01 = Long Form MTC
02 = MIDI Show Control
03 = Notation Information
04 = Device Control
05 = Real Time MTC Cueing
06 = MIDI Machine Control Command
07 = MIDI Machine Control Response
08 = Single Note Retune

MMC messages

An MMC message (that is sent to, or generated by, an MMC device) is:
F0 7F deviceID 06 command F7
The third byte is the Device ID.
The fifth byte is the command:
01 Stop
02 Play
03 Deferred Play
04 Fast Forward
05 Rewind
06 Record Strobe (Punch In)
07 Record Exit (Punch out)
08 Record Ready (Record Pause)
09 Pause
0A Eject
0B Chase
0D MMC Reset
40 Write
44 Locate/Go to
47 Shuttle

The Goto MMC message

The Goto message cues recording or playback to an SMPTE time (a specific hour, minute, second, SMPTE frame number, and subframe number):
F0 7F deviceID 06 44 06 01 hr mn sc fr ff F7
Where, following the device ID and "MIDI Machine Control Command" (0x06) bytes:
0x44 specifies the LOCATE command
0x06 specifies the data byte count
0x01 specifies the "TARGET" sub-command
And:
hr is hours - from 0 to 23 (decimal)
mn is minutes - from 0 to 59 (decimal)
sc is seconds - from 0 to 59 (decimal)
fr is frames - from 0 to 29 (decimal)
ff is sub-frames / fractional frames (leave at zero if un-sure) - from 0 to 99 (decimal)

The Shuttle MMC message

Both forward and backward shuttling share the following MMC message:
 F0 7F deviceID 06 47 03 sh sm sl F7
Note: sh, sm and sl are defined as Standard Speed in the MIDI 1.0 Recommended Practice RP-013.

The Record Ready MMC message
The Record Ready (Arm Tracks) message will record-enable tracks:
F0 7F deviceID 06 40 L1 4F L2 track bitmap F7
L1 equals the number of bytes between L1 and F7. L2 equals the number of bytes in the track bitmap. Each track is assigned a bit in the track bitmap. To set a track, you must know both the byte in which the track's bit lives, and also the bit corresponding to that track. Note that each byte can only hold 7 tracks.
 Track 1: byte 1 + 0x20
 Track 2: byte 1 + 0x40
 Track 3: byte 2 + 0x01
 Track 4: byte 2 + 0x02
 Track 5: byte 2 + 0x04
 Track 6: byte 2 + 0x08
 Track 7: byte 2 + 0x10
 Track 8: byte 2 + 0x20
 Track 9: byte 2 + 0x40
 Track 10: byte 3 + 0x01
 and so on.

Identity Request

Query an MMC device to find out its identity. Identity Request message (note that this is equal to a 'stop' message):
F0 7E channel 06 01 F7
The reply is device/manufacturer specific. For example, a tape recorder will return the following System Exclusive message:
F0, 7E, channel, 06 02 ID fc1 fc2 fn1 fn2 v1 v2 v3 v4 F7
parameters:
ID - Device's ID
fc1 fc2 - Device's family code
fn1 fn2 - Device's family number
v1 v2 v3 v4 - Software Version

MIDI time code (MTC)

or MIDI time division, embeds the same timing information as standard SMPTE time code as a series of small 'quarter-frame' MIDI messages. There is no provision for the user bits in the standard MIDI time code messages, and SysEx messages are used to carry this information instead. The quarter-frame messages are transmitted in a sequence of eight messages, thus a complete timecode value is specified every two frames. If the MIDI data stream is running close to capacity, the MTC data may arrive a little behind schedule which has the effect of introducing a small amount of jitter. In order to avoid this it is ideal to use a completely separate MIDI port for MTC data. Larger full-frame messages, which encapsulate a frame worth of timecode in a single message, are used to locate to a time while timecode is not running.
Unlike standard SMPTE timecode, MIDI timecode's quarter-frame and full-frame messages carry a two-bit flag value that identifies the rate of the timecode, specifying it as either:
24 frame/s (standard rate for film work)
25 frame/s (standard rate for PAL video)
30 frame/s (drop-frame timecode for NTSC video)

30 frame/s (non-drop timecode for NTSC video)
MTC distinguishes between film speed and video speed only by the rate at which timecode advances, not by the information contained in the timecode messages; thus, 29.97 frame/s dropframe is represented as 30 frame/s dropframe at 0.1% pulldown.
MTC allows the synchronisation of a sequencer or DAW with other devices that can synchronise to MTC or for these devices to 'slave' to a tape machine that is striped with SMPTE. For this to happen a SMPTE to MTC converter needs to be employed. Please note that it is possible for a tape machine to synchronise to an MTC signal (if converted to SMPTE), if the tape machine is able to 'slave' to incoming timecode via motor control, which is a rare feature.

Console Automation

Audio mixers can be controlled with MIDI during console automation.

Alternate hardware transports

In addition to the original 31.25 kbits/sec (baud is the signalling rate and is the reciprocal of the shortest signalling element; bits/sec is the data rate) current-loop transported on 5-pin DIN, other connectors have been used for the same electrical data, and transmission of MIDI streams in different forms over USB, IEEE 1394 a.k.a FireWire, and Ethernet is now common (see below).

USB

A standard for MIDI over USB was developed in 1999 as a joint effort between IBM, Microsoft, Altec Lansing, Roland Corporation, and Phillips[8]. To transmit MIDI over USB a Cable Number and Cable Index are added to the message, and the result is encapsulated in a USB packet. The resulting USB message can be double the size of the native MIDI message. Since USB is over 15,000 times faster than MIDI (480,000 Kbits/sec vs 31.25 Kbits/sec,) USB has the potential to be much faster. However, due to the nature of USB there is more latency and jitter introduced that is usually in the range of 2 to 10ms, or about 2 to 10 MIDI commands. Some comparisons done in the early part of the 2000s showed USB to slightly slower with higher latency [9], and this is still the case today. Despite the latency and jitter disadvantages, MIDI over USB is increasingly common on musical instruments.
[edit]
XLR3
Some early MIDI implementations used XLR3 connectors in place of the 5-pin DIN. The use of XLR3 connectors allowed the use of standard low-impedance microphone cables as MIDI cables. As the 31.25 Kbits/sec current-loop requires only three conductors, there was no problem with the loss of two pins. An example of this use is the Octave-Plateau Voyetra-8 synthesizer.

Over a computer network

Compared to USB or FireWire, the computer network implementation of MIDI provides network routing capabilities, which are extremely useful in studio or stage environments (USB and FireWire are more restrictive in the connections between computers and devices). Ethernet is moreover capable of providing the high-bandwidth channel

that earlier alternatives to MIDI (such as ZIPI) were intended to bring.
After the initial fight between different protocols (IEEE-P1639, MIDI-LAN, IETF RTP-MIDI), it appears that IETF's RTP MIDI specification for transport of MIDI streams over computer networks is now spreading faster and faster since more and more manufacturers are integrating RTP-MIDI in their products (Apple, CME, Kiss-Box, etc.). Mac OS X, Windows and Linux drivers are also available to make RTP MIDI devices appear as standard MIDI devices within these operating systems. IEEE-P1639 is now a dead project. The other proprietary MIDI/IP protocols are slowly disappearing, since most of them require expensive licensing to be implemented (while RTP MIDI is completely open), or the MIDI implementation does not bring any real advantage (apart from speed) over original MIDI protocol.

RTP-MIDI transport protocol

The RTP-MIDI protocol has been officially released in public domain by IETF in December 2006 (IETF RFC4695).[10] RTP-MIDI relies on the well-known RTP (Real Time Protocol) layer (most often running over UDP, but compatible with TCP also), widely used for real-time audio and video streaming over networks. The RTP layer is easy to implement and requires very little power from the microprocessor, while providing very useful information to the receiver (network latency, dropped packet detection, reordered packets, etc.). RTP-MIDI defines a specific payload type, that allows the receiver to identify MIDI streams.
RTP-MIDI does not alter the MIDI messages in any way (all messages defined in the MIDI norm are transported transparently over the network), but it adds additional features such as timestamping and sysex fragmentation. RTP-MIDI also adds a powerful 'journalling' mechanism that allows the receiver to detect and correct dropped MIDI messages.The first part of RTP-MIDI specification is mandatory for implementors and describes how MIDI messages are encapsulated within the RTP telegram. It also describes how the journalling system works. The journalling system is not mandatory (journalling is not very useful for LAN applications, but it is very important for WAN applications).
The second part of RTP-MIDI specification describes the session control mechanisms that allow multiple stations to synchronize across the network to exchange RTP-MIDI telegrams. This part is informational only, and it is not required.
RTP-MIDI is included in Apple's Mac OS X, as standard MIDI ports (the RTP-MIDI ports appear in Macintosh applications as any other USB or FireWire port. Thus, any MIDI application running on Mac OS X is able to use the RTP-MIDI capabilities in a transparent way). However, Apple's developers considered the session control protocol described in IETF's specification to be too complex, and they created their own session control protocol. Since the session protocol uses a UDP port different from the main RTP-MIDI stream port, the two protocols do not interfere (so the RTP-MIDI implementation in Mac OS X fully complies to the IETF specification).
Apple's implementation has been used as reference by other MIDI manufacturers. A Windows XP RTP-MIDI driver[11] for their own products only has been released by the Dutch company Kiss-Box and a Linux implementation is currently under development by the Grame association.[12] So it seems probable that the Apple's implementation will become the "de-facto" standard (and could even become the MMA reference implementation).

Converting instruments to MIDI

Some older instruments, for example electronic organs built in the 1970s and 1980s, are becoming beyond repair, due to lack of spares and/or of technicians trained on such equipment. The best candidates for upgrade are what are referred to as "Console" sized, or have at least 2x keyboards of 61 notes, and at least a 25 note (preferably 32 note concave) pedal board. Smaller "Spinet" sized organs are probably not considered worthy of conversion. In some cases, they can be modified into MIDI instruments. Terms coined from MIDI + modification are often used, such as midification or to midify.
An old electronic organ could have almost all of its discrete component electronics replaced by modern circuitry which will cause the instrument to output MIDI signals. The instrument would then become a specialised MIDI keyboard. Its MIDI output would need to be fed to a MIDI engine of some sort.
See for example: Midification of an Organ
In modern times new music keyboards have MIDI functions as standard and can be connected to the computers with a PC-to-MIDI circuit or simply via USB. Other forms of MIDI controllers include wind controllers, drums, guitars, accordion and many others.
Old synthesizers are not often modified to transmit MIDI but people sometimes modify them to receive it. The modification involves adding a circuit board that converts digital MIDI signals into analog control voltages, as well as a MIDI jack. The circuit boards are usually designed specially for one model of synthesizer and it takes some expertise to install them. This allows pre-MIDI analog synthesizers to be controlled by digital sequencers, whereas they formerly required the user to actually play them.

Beyond MIDI 1.0

Although traditional MIDI connections work well for most purposes, a number of newer message protocols and hardware transports have been proposed over the years to try to take the idea to the next level. Some of the more notable efforts include:
[edit]
OSC
The Open Sound Control (OSC) protocol was developed at CNMAT. OSC has been implemented in the well-known software synthesizer Reaktor, in other innovative projects including SuperCollider, Pure Data, Isadora, Max/MSP, Csound, vvvv, ChucK, and LuaAV as well as in many general purpose programming languages such as C (liblo), Python (pyliblo), Haskell (hosc), Scheme (sosc) and Pure (pure-liblo). The Lemur Input Device, a customizable touch panel with MIDI controller-type functions, also uses OSC. OSC differs from MIDI 1.0 over traditional 5-pin DIN in that it can run at broadband speeds when sent over Ethernet connections, however the differences are smaller compared to MIDI when run at broadband speeds over Ethernet connections. Few mainstream musical applications and no standalone instruments support the protocol so far, making whole-studio interoperability problematic. OSC is not owned by any private company, however it is also not maintained by any standards organization. Since September 2007, there is a proposal for a common namespace within OSC[13] for communication between and controllers, synthesizers and hosts, however this too would not be maintained by any standards organization.
mLAN

Yamaha has its mLAN protocol, which is based on the IEEE 1394 transport (also known as FireWire) and carries multiple MIDI 1.0 message channels and multiple audio channels. mLAN is not maintained by a standards organization as it is a proprietary protocol. mLAN is open for licensing, although covered by patents owned by Yamaha.

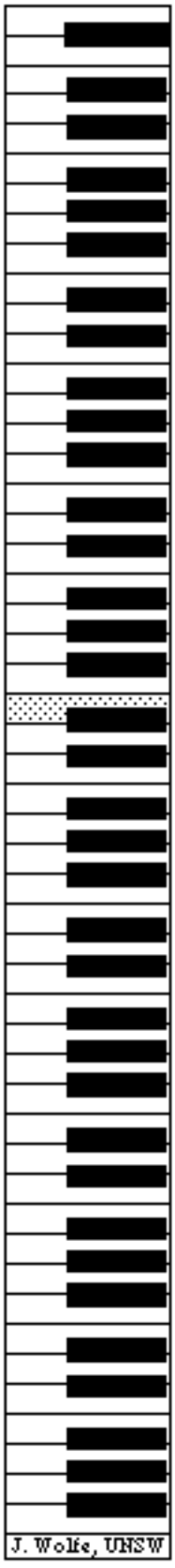
HD Protocol

Development of a version of MIDI for new products which is fully backward compatible is now under discussion in the MMA. First announced as "HD-MIDI" in 2005 and tentatively called "HD Protocol" since 2008, this new standard would support modern high-speed transports, provide greater range and/or resolution in data values, increase the number of Channels, and support the future introduction of entirely new kinds of messages. Representatives from all sizes and types of companies are involved, from the smallest speciality show control operations to the largest musical equipment manufacturers. No technical details or projected completion dates have been announced.[14][15] Various transports have been proposed for use for the HD-Protocol physical layer, including a call for ACN to be used as the sole or primary transport in show control environments. [edit]

Geraadpleegde Literatuur:

MIDI 1.0 Detailed Specification	Een uitgave van de International MIDI Association
11857 Hartsook Street,	Hollywood CA 91607, USA
MIDI for Musicians	van Craig Anderton Amsco Publications, New York
KEYBOARD USA speciaal MIDI artikel.	van januari 1986 waarin een
AES Journal door Bob Moog.	Vol. 34, No. 5 van mei 1986
SEQUENTIAL CIRCUITS	Document No. MIDI-1 januari 1983 by Stanley Jungleib
YAMAHA	Diverse gebruikers-handleidingen van o.a. TX 81z,
DMP 7 en SPX 90.	

midi association info: http://www.midi.org/techspecs/index.php
midi specs 1.0: http://www.midi.org/techspecs/midispec.php

MIDI number	Note name	Keyboard	Frequency
21	A0		27.500
22			29.135
23	B0		30.868
24	C1		32.703
25			34.648
26	D1		36.708
27			38.891
28	E1		41.203
29	F1		43.654
30			46.249
31	G1		48.999
32			51.913
33	A1		55.000
34			58.270
35	B1		61.735
36	C2		65.406
37			69.296
38	D2		73.416
39			77.782
40	E2		82.407
41	F2		87.307
42			92.499
43	G2		97.999
44			103.83
45	A2		110.00
46			116.54
47	B2		123.47
48	C3		130.81
49			138.59
50	D3		146.83
51			155.56
52	E3		164.81
53	F3		174.61
54			185.00
55	G3		196.00
56			207.65
57	A3		220.00
58			233.08
59	B3		246.94
60	**C4**		**261.63**
61			277.18
62	D4		293.67
63			311.13
64	E4		329.63
65	F4		349.23
66			369.99
67	G4		392.00
68			415.30
69	**A4**		**440.00**
70			466.16
71	B4		493.88
72	C5		523.25
73			554.37
74	D5		587.33
75			622.25
76	E5		659.26
77	F5		698.46
78			739.99
79	G5		783.99
80			830.61
81	A5		880.00
82			932.33
83	B5		987.77
84	C6		1046.5
85			1108.7
86	D6		1174.7
87			1244.5
88	E6		1318.5
89	F6		1396.9
90			1480.0
91	G6		1568.0
92			1661.2
93	A6		1760.0
94			1864.7
95	B6		1975.5
96	C7		2093.0
97			2217.5
98	D7		2349.3
99			2489.0
100	E7		2637.0
101	F7		2793.0
102			2960.0
103	G7		3136.0
104			3322.4
105	A7		3520.0
106			3729.3
107	B7		3951.1
108	C8	J. Wolfe, UNSW	4186.0