Inside the MIDI message layer

Ad van Gerven

Alten Nederland BV

Abstract

The MIDI 1.0 standard is an unambiguous but context sensitive description intended for users of electronic instruments and instrument manufacturers. This implies difficulties in specifying and designing software for MIDI. This article presents a specification for the Musical Instrument Digital Information exchange that confirms to MIDI 1.0 but is targeted to software designers. The exchange of MIDI information is modeled in 4 layers: the physical layer, network topology layer, MIDI message layer and the application layer. The focus on the article is on the software definition of the MIDI message layer. A set of requirements and a Karnaugh diagram are presented that allow a full implementation of this layer.

Keywords: MIDI, layered view, message layer, channel, system common, system real-time, system exclusive

Inside the MIDI message layer

MIDI is the abbreviation for Musical Instrument Digital Interface. It is a standard initiated by the MMA (MIDI Manufacturers Association) in 1982 (MMA MIDI 1.0). Among others it defined the physical layer for information interchange between supporting devices, and the protocol for sending and receiving MIDI information.

A layered view on the MIDI information transmission protocol

Although the MIDI standard does not define layers in terms of OSI (Open Systems Interconnection or in terms of more specific network protocol stacks it is convenient, from a software point of view, to define at least some layers as in Illustration 1. The scope and responsibilities of each layer are defined in (Table 1, p. 4).

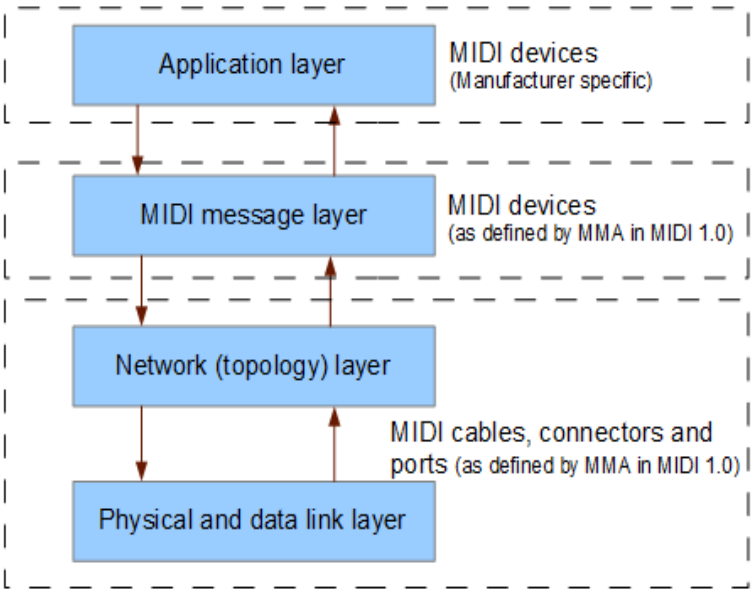

Illustration 1: Layered model view of MIDI information transmission

Layer	Scope and responsibilities
Physical and data link layer	Within MIDI 1.0 all data transmission is unidirectional, e.g. there is one transmitter and there are one or more receivers; Data transmission is serial, using a current loop and an optocoupler on the receiver side; Baud rate is fixed to 31250 bits per second; A byte (a series of 8 bits) is the smallest unit of information that is transmitted; To synchronize and safeguard the transmission a start-bit, a stop-bit and a parity bit are user for each byte. Transmission of a byte, including this overhead, thus takes 35.2 microseconds;
Network topology layer	Information is transmitted in one direction, from the MIDI out port on a transmitting device to the MIDI in port of a receiving device. A receiving device may pass on the information it receives by hardware routing a copy to its MIDI thru port.
MIDI message layer	Converts incoming byte streams to MIDI messages; Converts MIDI messages to outgoing byte streams; Supports the MIDI running status concept; Supports interrupts by MIDI system real-time messages;
Application layer	Contents of this layer are fully device and manufacturer specific. They are out of scope of the MIDI standard and out of scope for this article.

Table 1: Overview of responsibilities of the MIDI information exchange layers

The MIDI message layer

The MIDI message layer is responsible for converting incoming byte streams to MIDI messages and vice versa. These operations are typically performed by software which is located in the MIDI transmitting and receiving devices. A problem, from a software point of view, is that the definition of the MIDI information exchange protocol was, and still is, context sensitive. For example, a message is complete when:

- All expected data is received; *or*
- An “end of system exclusive” message is received *and* the previous message was a “start of system exclusive message”; *or*
- The message contains only a status and no data;

A software view on the MIDI message layer

An overview of the MIDI messages, as defined in 1982 and unchanged since, can be found in (Summary of MIDI messages, 1982). That same reference holds additional tables with all addenda made since 1982. These concern the meaning and syntax of individual data bytes in specific messages, such as controller identifiers, Registered Parameter Numbers (RPNs) and Non Registered Parameter Numbers (NRPNs), MIDI sample dump etc. We consider these out of scope for the MIDI message layer and, instead, subject for the application layer.

Functional requirements for the MIDI message layer

The table of MIDI messages (Summary of MIDI messages, 1982) and a few lines in the official MIDI 1.0 standard are actually all that is published. In order to be able to design and implement software for the MIDI message layer we need an unambiguous set of software requirements that matches the MIDI 1.0 standard, as given below.

1. A MIDI message consists of at least one byte, which is either a data byte or a status byte:
- A data byte is an unsigned integer numeric value in the range [0x00, 0x7F] ([0, 127]);
- A status byte is an unsigned integer numeric value in the range [0x80, 0xFF] ([128, 255]);
2. Each MIDI message is associated with a software status, which is one of these categories[1]:
- Channel status;
- System common status;
- System exclusive status;
- System real-time status;

Note that the MIDI 1.0 standard does not specify exception handling or "bad-weather" scenarios. To help supporting these we have added 2 more software status categories:

- Undefined status;
- Reserved status;

1 To avoid context sensitivity and to support "system exclusive" as an exception, we propose use of these software categories instead of the MIDI 1.0 message status categories (Table 2, p.7).

For a complete overview of all different status values and their properties see (Table 2, p. 7).

3. Messages associated with a channel status, system common status and system real-time status have a fixed size, ranging from 0 to 2 data bytes, depending on their associated status. See (Table 2, p. 7) for details.
4. The size of system exclusive messages associated with a "start of system exclusive" status value is unspecified in MIDI 1.0. The size of system exclusive messages associated with an "end of system exclusive" status value is defined as 0.
5. For messages, associated with any software status category, that have a size of 0 data bytes, transmission of the status byte is mandatory. The rationale behind this is understood best by considering the clock message. This message is used to synchronize timed playback of MIDI commands for multiple devices.
6. By design messages, associated with any software status category, that have a size of 0 data bytes, are considered complete immediately after transmission / reception of the status byte.
7. For messages associated with a channel status or system common status that have a size greater than 0 data bytes transmission of the status byte is mandatory only when the status associated with a new message differs from the status associated with the latest message. When the status associated with a new message is identical to the status associated with the latest message transmission of the status byte is optional[2].
8. By design messages associated with a channel status or system common status that have a size greater than 0 data bytes are considered complete immediately when the number of transmitted / received data bytes equals the message size in data bytes.
9. For messages associated with the system exclusive software status category transmission of the status byte is mandatory.
10. A message associated with a "start of system exclusive" status value is considered complete only when a message associated with an "end of system exclusive" value is received.

2 This is known as the "running status" concept, which allows for an increase in bandwidth that was especially important in first generation of MIDI devices.

11. Messages associated with any software status category that have a size greater than 0 data bytes can be interrupted by any number of system real-time messages[3].

Status value	Message has data	Message status category	Message data size	Running status allowed
0x80	TRUE	Channel	2	TRUE
0x90	TRUE	Channel	2	TRUE
0xA0	TRUE	Channel	2	TRUE
0xB0	TRUE	Channel	2	TRUE
0xC0	TRUE	Channel	1	TRUE
0xD0	TRUE	Channel	1	TRUE
0xE0	TRUE	Channel	2	TRUE
0xF0	TRUE	System exclusive	Unspecified	TRUE
0xF1	TRUE	System common	1	TRUE
0xF2	TRUE	System common	2	TRUE
0xF3	TRUE	System common	1	TRUE
0xF4	FALSE	System common	0	FALSE
0xF5	FALSE	System common	0	FALSE
0xF6	FALSE	System common	0	FALSE
0xF7	FALSE	System exclusive	0	FALSE
0xF8	FALSE	System real-time	0	FALSE
0xF9	FALSE	System real-time	0	FALSE
0xFA	FALSE	System real-time	0	FALSE
0xFB	FALSE	System real-time	0	FALSE
0xFC	FALSE	System real-time	0	FALSE
0xFD	FALSE	System real-time	0	FALSE
0xFE	FALSE	System real-time	0	FALSE
0xFF	FALSE	System real-time	0	FALSE

Table 2: Overview of status values and derived properties

Software requirements for the MIDI message layer

Software requirements must be unambiguous and context free. One obvious method of converting context sensitive requirements into usable software requirements is to specify all context dependencies and include these pre-conditions in the input parameters of the system.

3 Interruption is required due to the real-time nature of these messages, and is made possible because all system real-time message have a fixed message size of 0 data bytes (which implies no stack operations are needed to support this interrupt mechanism).

For the MIDI message layer this results in a rather large set (10) of binary input parameters. Fortunately we do not have to evaluate all 2^{10} possible combinations because the number of different status categories is limited and due to the functional requirements described earlier.

The input parameters for information reception in the MIDI message layer can be divided into 2 groups: pre-conditions that apply to the incoming byte(s), and pre-conditions that apply to the current message. These are summarized in the following tables:

Data byte	System real-time	Start of system exclusive	End of system exclusive	Status reserved	Message size greater 0	Pre-condition description
FALSE	TRUE	FALSE	FALSE	FALSE	FALSE[4]	System real-time; Non-reserved
FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	System real-time; Reserved
FALSE	FALSE	TRUE	FALSE	FALSE[5]	TRUE[6]	Start of system exclusive
FALSE	FALSE	FALSE	TRUE	FALSE[5]	FALSE[7]	End of system exclusive
FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	System common; Non-reserved; Size = 0
FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	System common; Reserved; Size = 0
FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	Channel status *OR* System common; Non-reserved; Size > 0
TRUE	X	X	X	X	X	Data byte

Table 3: Overview of pre-conditions for received MIDI bytes

4 A system real-time status always has data size zero.

5 A (start or end) of system exclusive status is never reserved (Table 2, p.7)

6 A (start of) system exclusive message always has more than zero data bytes. The actual message size is not known until an end of system exclusive status is received.

7 An end of system exclusive status always has data size zero.

Current status unknown or reserved	Current status start of system exclusive	Message size greater than 0	Message complete	Pre-condition description
X	X	X	X	Don't care
TRUE	FALSE	FALSE	TRUE	Status unknown or reserved
FALSE	FALSE	FALSE	TRUE	Known valid status; Size = 0;
FALSE	FALSE	TRUE	FALSE	Known valid status; Size > 0; Message incomplete
FALSE	FALSE	TRUE	TRUE	Known valid status; Size > 0; Message complete
FALSE	TRUE	TRUE[6]	FALSE	System exclusive message; Incomplete

Table 4: Overview of pre-conditions for the last received MIDI message

MIDI message Karnaugh diagram **© Ad van Gerven**

Incoming byte pre-conditions						Incoming byte description	Current message pre-conditions				Current message description	Action
Byte type is data	System real-time	Start of system exclusive	End of system exclusive	Status reserved	Message size > 0		Current status unknown or reserved	Current status Start of system exclusive	Message size > 0	Message complete		
FALSE	TRUE	X	X	FALSE	X	system-real time; non-reserved	X	X	X	X	Don't care	Notify subscribers of real-time message
		X	X	TRUE	X	system-real time; reserved	X	X	X	X		None
	FALSE	TRUE	FALSE	X	X	Start of system exclusive	TRUE	X	X	X	Current status unknown or reserved	Reset message to incoming status
							FALSE	FALSE	FALSE	X	Known status; Message size = 0	
							FALSE	X	TRUE	TRUE		
	FALSE	FALSE	TRUE	X	X	End of system exclusive	TRUE	X	X	X	Current status unknown or reserved	Reset message to incoming status; Notify subscribers
							FALSE	X	FALSE	X	Known status; Message size = 0	
							FALSE	X	TRUE	TRUE	Known status; Message size > 0; Message complete	
							FALSE	TRUE	X	X	System exclusive message; incomplete	Notify subscribers of completion of a system exclusive message
	FALSE	FALSE	FALSE	FALSE	FALSE	System common; non-reserved; size = 0	TRUE	X	X	X	Current status unknown or reserved	Reset message to incoming status; Notify subscribers
							FALSE	X	FALSE	X	Known status; Message size = 0	
							FALSE	X	TRUE	TRUE	Known status; Message size > 0; Message complete	
	FALSE	FALSE	FALSE	TRUE	FALSE	System common; reserved; size = 0	X	X	X	X	Don't care	None
	FALSE	FALSE	FALSE	FALSE	TRUE	Channel *or* System common; non-reserved; size > 0	TRUE	X	X	X	Current status unknown or reserved	Reset message to incoming status
							FALSE	X	FALSE	X	Known status; Message size = 0	
							FALSE	X	TRUE	TRUE	Known status; Message size > 0; Message complete	
	FALSE	X	X	X	X	Not system-real time	FALSE	X	TRUE	FALSE	Known status; Message size > 0; Message incomplete	Reset message to status "unknown"
TRUE	X	X	X	X	X	Data	TRUE	X	X	X	Current status unknown or reserved	None
							FALSE	X	FALSE	X	Known status; Message size = 0	
							FALSE	X	TRUE	TRUE	Known status; Message size > 0; Message complete	Remove current message data; Append incoming data; Notify subscribers when applicable
							FALSE	X	TRUE	FALSE	Known status; Message size > 0; Message incomplete	Append incoming data to message; Notify subscribers when applicable

Illustration 1: MIDI message layer Karnaugh diagram

References

MIDI Manufacturers Association (2006), An introduction to MIDI, Retrieved from http://www.midi.org/

MIDI Manufacturers Association (1982), Summary of MIDI messages, Retrieved from http://www.midi.org/techspecs/midimessages.php