USB MIDI pulse width modulation software

Ing. Dalibor Slovák
Department of Computer and Communication systems
Tomas Bata University in Zlin, Faculty of Applied Informatics
nám. T.G.Masaryka 5555, 760 01 Zlín
Czech republic
slovak@fai.utb.cz

ABSTRACT

This article describes software application, that generates voltage pulses. The default commands are generated in the MIDI protocol. MIDI - Musical Instrument Digital Interface. As already mentioned the name of it is the interface that is used primarily to control musical instruments through the computer. The intention is for applications that make controlling other devices that do not communicate via this protocol. The result is the control voltage, which enables control the device via this voltage. The application is primarily intended for users who use the MIDI protocol as part of a live audio and audio-visual production. The application is primarily intended for devices that are not equipped with an interface for communicating via MIDI protocol.

Keywords: MIDI, USB, software, USB Audio, USB Audio MIDI, descriptor

1 Introduction

Today is MIDI protokol part of efects units for all musical instruments. For example effect units for electric guitars, keyboards for singers or acoustic instruments too. MIDI protocol is part of other stage equipment. E.g. MIDI protocol is part of stage lights or artificials smokes. Henceforth exist devices that do not have the capacity to understand commands generated by the MIDI protocol. Based on this requirement was initiated research and development of software that would be able to operate and manage such facilities. As part of this development appeared to be the most effective combination of MIDI and USB protocol.

The aim of this research was to design as simple and cost-optimized software for control devices that are not equipped with interfaces communicating via MIDI protocol. Platform and system-independent software that can be used on all available, hardware platforms, which enable communication via USB.

2 Description of system equipment and technology

2.1 Universal serial bus

The basis for this specification of this software is of course a general standard USB 2.0, followed by the standard USB Audio device and it is based on the actual USB MIDI standard. This is now widely used because MIDI transmission of information via the USB is more efficient than transmission of information using standard MIDI DIN jacks. DIN occupies too much space as the PC case, so the external sound cards that are used most often in combination with laptops. Placement DIN connectors on notebooks is due to the size precluded.

The USB connection is defined by the so-called Endpoints, which are in any USB devices, its functionality, there may be to 15 Endpoint zero - the default endpoint is the default pipe (channel connecting the USB client and host), which connects, when the USB device identified in the hardware configuration. They demand signal IRP (I / O Request Packet), which are performed in serial order. If an error occurs in the implementation

ISBN: 978-1-61804-004-6

of an IRP, the IRP others are removed from the queue and is then to correct errors and reload the IRP requirements of the USB device. The pipes are of two kinds - Messages and Streams. Each endpoint has its own descriptor. It is a structure that describes the components of the USB device and its identification in the system. Classification and description of the descriptors is given below.

1. **Device descriptor** – descriptor of the first level - the kind of program-independent component that is used during enumeration (the introduction of devices into the system) for device identification to the system.
2. **Configuration descriptor** – structure that describes device identification to the lowest possible levels within the endpoints. This structure contains all other sub-structures (descriptors endpoints) and it determines all the functionality of the USB device.
3. **Interface descriptor** – interface descriptor may not always be the only one. In most cases, number is given enlistment arrangement after its functionality. The number is given to the inclusion of a USB device class. The software described in this article is primarily intended for devices that have a total of two interface descriptors. Descriptors for USB Audio Interfaces and USB Audio and MIDI interface.
4. **Endpoint descriptors** – these descriptors always be several. Their number is based on the functionality of USB devices. Their name is directly dependent on the type of USB devices.

All communication on the bus is time multiplexed into frames 1 millisecond long. Each frame can contain many transactions of different devices and different endpoints. Data transfer over the bus can be divided into 4 types:

1. **Isochronous transfers** - transport large amounts of data (up to 1023 B) is guaranteed delivery time, but does not ensure data integrity
2. **Bulk transfers** – Transport of large amounts of data to ensure integrity, but is not guaranteed delivery time
3. **Interrupt transfers** – For a small amount of data to ensure integrity and timely delivery
4. **Control transfers** – Used during initial device setup (enumeration)

At the moment when the device is connected to the bus, the USB host will start a process called enumeration. During this process identifies the host device receives configuration information needed by requesting a device descriptor, configuration, interfaces and endpoints, and finally sets the desired device configuration. Course enumeration might look like this:

1. USB reset
2. Device descriptor request – only basic part of Device identificaton
3. USB reset
4. Unique device adress setting
5. Device descriptor request – this time whole descriptor including all basic descriptors
6. Configuration descriptors request
7. Other descriptors request
8. Setting the desired configuration

In terms of device USB specification defines specific conditions in which the device may be located during enumeration, and allowed transitions between them (DETACHED _ATTACHED _ POWERED _ DEFAULT _ ADDRESS_PENDING _ ADDRESSED_ CONFIGURED _ READY) [4].

Equipment connected to the bus can be quite specific and require specific drivers, or they may belong to the class of USB devices (HID – human interface device class, MSD – mass storage class, CDC – communication device class). These classes define some of the properties and behavior of the device, so for a variety of devices claiming to be the same class, the system can use the same driver.

Messages are accurately specified in advance and are determined by the size and requirement of device or guest. Pipes messages comply with a data structure that allows the identification of orders and requests. The pipes are two-way messaging and always use two endpoints with the same number.

Streams are the pipes that contain actual data transfer. They are one way and the data in Streams are transmitted subsequently, via packets. If you do not fill in the whole packet, it needs not to add full size [5].

ISBN: 978-1-61804-004-6

2.2 Audio device USB standard

As is clear from the chosen theme, USB is entirely sufficient transmission capacity for transmission of audio data, similarly for MIDI information too. Audio equipment to the USB protocol specifications and their own appropriate set of descriptors required endpoints for transferring audio data. In most cases, it is one of several specifications of the equipment because they are always combined with devices other than the standard USB Audio. A somewhat different situation is in the case of MIDI devices. MIDI USB standard is an extension of standard USB Audio. Description of the audio course is based on the relevant standard [6].

2.3 MIDI device USB standard

At the beginning we must say that the typical USB MIDI devices belonging to the USB CDC class. It is the same as the audio devices are considered communication interfaces. Class description for the USB MIDI device is one part of standard USB Audio. This happens when a USB device capable of receiving, respectively send MIDI messages. As already mentioned in chapter 2.2, you must specify the interface on the device interface level. USB MIDI device will then have two interfaces (interface). One is an audio interface, MIDI, then the second. This is, of course, described using the descriptors, so that the device is easily identified in the system and was particularly visible for the applications that are capable to communicate via MIDI protocol. In our case, they are software application Cubase SX 2, respectively. Cubase SX 3.

The specifications of endpoints are descriptors for MIDI devices known as endpoints for MIDI input and output jacks. These jacks are of two kinds. Some are known as External MIDI In, respectively OUT jacks. The second groups are then Embedded MIDI In, respectively Embedded OUT jacks. Passage of the MIDI data from the host to the MIDI device and then back is following.

Information walking from Host is addressing to device via External MIDI OUT jack then continous to Embedded MIDI IN jack to device. Now is information in device and it is treated. So treated infromation is send back via Embedded MIDI OUT jack of this device to External MIDI IN jack of the host.

This description is only virtual abstraction. This must be programmed within the USB device. Everything is numbered using the descriptors and associated together. The relevant descriptor item shows, which connector belongs to its opposite counterpart.

Furthermore, the association is also intended connections between External MIDI IN (OUT) jacks with the corresponding Embedded MIDI OUT (IN) jacks. It is always connected Embedded IN jack with External OUT jack and vice versa. Similarly, the individual pins on the MIDI jacks have their serial numbers and are set with numeric information, which is how many pins is, and to which devices are connected [6]. Within these standards are often a multi-functional devices. Very often is such instruments is keyboard. Keyboard instrument is falling into the category of workstation.

2.3.1 Element

It is one of the intended logic levels. This is a basic building block for USB MIDI communication. This is a specification for the device, which may be a:

- Synthetizer
- External MIDI time synchronization
- Efects units handled by MIDI
- Etc.

Element is usually connected to one or a few Embedded or External MIDI connectors. If it is a synthesizer, usually produces audio output data generated by the MIDI information that is sent over the USB MIDI interface. Audio is then transmitted through the audio input terminal through input terminal of synthesizer. [6].

2.3.2 Input and Output terminal

Input terminal represents the output of the device (eg, synthesizer), the input terminal is connected to the device and through it to get a USB MIDI transfer to the device's configuration.

The communication between MIDI devices and audio equipment is via the input and output terminals. These terminals are logically created by the descriptors primarily used for transmission of information important for

ISBN: 978-1-61804-004-6

correct processing of audio. Such information includes, for example transmission setup information effect units or equalizers and last but not least, e.g. the volume of output. [6].

2.3.3 USB-MIDI Converter

USB-MIDI converter is the heart of every MIDI device, it provides a connection between the host and USB-MIDI interface. It is the fundamental building block. On one hand, it interfaces with the USB pipes, which are used to exchange MIDI data between the host and USB-MIDI endpoints of the device. On the other hand, there is presented an appropriate number of embedded MIDI jacks. These embedded jacks are logical interface presenting the true connectivity within a MIDI device. USB MIDI converter provides a connection between the MIDI OUT endpoint and relevant Embedded MIDI IN jack. Similarly, it provides a link between the Embedded MIDI OUT jack and the corresponding MIDI IN endpoint.

2.3.4 USB MIDI device descriptors description

Following the previous text and standards then will have the typical MIDI device communicating via USB these descriptors:

1. **Device Descriptor**

   The items correspond to the standard CDC device class

2. **Configuration Descriptor**
   Like the device descriptor
3. **Standard AC Interface Descriptor**

   Audio Control interface has no its endpoint. Default endpoint zero used for communication. Class-specific Audio Control requests are sent out using default channel. It does not provide any endpoints for settings USB device interrupt.

4. **Class-specific AC Interface Descriptor**

   It is always connected with Standard (header) descriptor, which contains basic information about audio interfaces. It contains all the pointers needed to describe a group of audio interfaces in conjunction with a given audio device.

5. **Standard MIDI Streaming Interface Descriptor**

   Standard Interface Descriptor characterizes the device as such. With this descriptor is specified by the internal structure of the USB MIDI device, and further detailed description is contained in descriptors, which are part of the configuration structure.

6. **Class-specific MIDI Streaming Interface Header Descriptor**

   It provides more (precise) information relating to the internal structure of the device.

7. **MIDI IN Jack Descriptor**
   Describes MIDI IN jacks, no matter what already discusses Embedded or about External jacks. This parameter is set in e bJackType variable.

8. **MIDI OUT Jack Descriptor**

   It describes the MIDI OUT jacks, as well as MIDI IN descriptor. Its structure is added to other items that are necessary for accurate specification of the corresponding External links, respectively Embedded MIDI IN descriptor. These additional items specified with each pin of the MIDI OUT connector, and his status for data transmission.

9. **Element Descriptor**
   Extends structure MIDI OUT descriptor about sum input and check out data station further about setting of pertinent other ability USB MIDI arrangement[6].
10. **Standard MIDI Streaming Bulk Data Endpoint Descriptor**

    The content of this descriptor is consistent with a standard endpoint descriptor as described in chapter 9.6.4 USB specification [4].

11. **Class-Specific MS Bulk Data Endpoint Descriptor**

    The bNumEmbMIDIJack field contains the number Embedded MIDI Jacks associated with this endpoint. In the event that it is an input endpoint, then embedded jack should be the MIDI OUT. If this is the final endpoint, should be the Embedded MIDI IN jack. BaAssocJacks field contains the ID then embedded jacks.

ISBN: 978-1-61804-004-6

12. Standard MS Transfer Bulk Data Endpoint Descriptor

This descriptor also agree with description descriptor from chappter 9.6.4. USB specifications, then standard Endpoint descriptor. BEndpointAdress field designates by the help of D7 parameter, if discuss input transfer endpoint or check out transfer endpoint.

13. Class-Specific MS Transfer Bulk Data Endpoint Descriptor

The specification of this device is not present this Descriptor type.

2.4 Development environmets and used firmware and software

2.4.1 MPLAB IDE

To edit and develop the source code was used development environment, which Microchip adds and also offers as a shareware to application development on their boards. Reason using this environment was facilitation development of a case study, which is a separate hardware platform USB MIDI Device Lights, in which was implemented above-mentioned software.

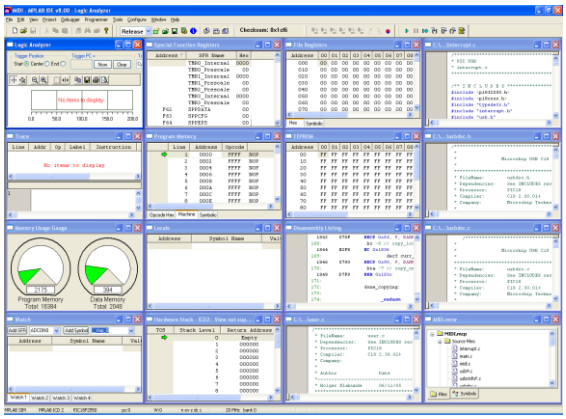

Fig. 1. MPLAB IDE

2.4.2 MCC C18

Compiling MPLAB IDE is part of the environment. The exact name is of the MPLAB ® C18 C Compiler. This compiler is based on ANSI C '89.

2.4.3 Firmware – modules and files

Software, that the was used to operating USB MIDI light Device, coming-out from specimen instance CDC RS – 232, then software that the over USB emulates serial port. This software was forced and modified so, in order to tend lighting on the basis taken MIDI orders. Basic and indispensable files are:

main.c - Function *main()* includes infinite program central loop *while(1)*. In this loop are valet through procedures *USBTASKS(void)* and *void ProcessIO(void)* all requisite tasks in programmatic succession that the is given instructions source text code.

typedefs.h - In this file are defineds individual data type.

interrupt.c a .h - This module contains an implementation of interrupt handling both, high and low priority.

usb.h – This file interlocks nesting needed header file to the whole program. Is here possibly remark, that the initialization isn't quite full and is necessary is complete as need may be single arrangement and their functionality.

usbdefs_ep0_buff.h – In set is included definition textures Ctrl_TRF_SETUP and data - faggot for answer Ctrl_TR_DATA for Controll transfer.

usbdefs_std_dsc.h – Content of file is description of descriptors arrangement as well as definition values for input and check out endpoints. This file includes all variables for full structures of endpoints.

usb_compile_time_validation.h– Verification sizes endpoint descriptor according to standard of USB. Then reach size hereof descriptor can be either 8, 16, 32 or 64 bytes.

usbcfg.h – By the help of hereof file is effected endpoint configuration of arrangement. Then is intended value and starting set for endpointzero and further then endpoint assignment for configuration descriptor and further also values for interface descriptors and their endpoints.

usbdsc.h a .c - This modulus includes information about USB descriptors. In set *usbdsc.h* are included definition structures of configuration and globalize descriptors here for visibility and in of others modulus through key word *external*.

usbmap.h a .c - This module presents USB memory manager. Allocation of USB endpoint and their buffer descriptors proceeds dynamically at compile time with usage some parameters defined in *usbcfg.h*.

usbdrv.c a .h - This module is in charge of USB communication and functional integration of other modules.

ISBN: 978-1-61804-004-6

usb9.c a .h - This module handles standard USB requests coming through EP0 under Chapter 9 of the USB 2.0 specification [3]. **usbctrltrf.h a .c -** The heart of this module is routine USBCtrlEPService (void), which serves only the following three operations - EP0 SETUP EP0 OUT EP0 IN and calls the appropriate routines.

In the case of an ordinary programming of any of the classes of devices (HID, MSD, CDC) do not require any intervention. Without a deeper study on the contrary any changes can be detrimental. In the case of programming a USB MIDI Device Lights were some interventions required and they are described in my other paper. Modules described below relate to each class. The program use depends on what category programmed device is included.

hid.c a .h, msd.c a .h,cdc.c a .h – modules specifying individual classes arrangement in terms of USB protocol.

2.4.4 user.c a h.

Fundamental module for user needs are two source text files *user.c and user.h*, that contains custom functions setting and macro. Here user does pertinent modification necessary for given functionality of arrangement. Therefore is this module separated from of others in this text too. Here are the adjustments made in the case of this software, which is described in this paper. All functional treatment for needs device control by the help of pulse – width modulation. Starting setting is treatment switched and no switched states for individual arrangements, that will be handled. So, setting of timer for connect or disconnect handled devices via pulse width of control voltage.

3 Conclusion

Aim proposal was create moneywise optimized, programmatic robust software interface for purposes specified in introduction of this paper. Among basic benefits software then belongs to its system catholicity. Next advantage is easy implementation to the lights applications.

From arrangement will accessible only USB connector and eventually jumper commutator for easy application reprogramming in the event of changes in lights configuration. An important benefit is the ability to exploit any effect devices, regardless of it, which protocol these devices have or not have in their software toolkit.

References:

[1] *MIDI standard review*. [available online]. <http://www.midi.cz>

[2] *DMX512 protocol*. [available online] <http://www.soundlight.de/techtips/dmx512/dmx512.htm>

[3] *Universal Serial Bus specification.pdf* [available online] <http://www.usb.org>

[4] Mach, J. *Firmware for USB devices with micro computers PIC:* Czech Technical University in Prague, 2006, 62 p. Diploma thesis at Faculty of Electrical Engineering, Thesis supervisor Ing. Miroslav Skrbek, PhD.

[5] *Universal Serial Bus Device Class Definition for MIDI Devices.pdf* [available online] <http://www.usb.org>

[6] *Universal Serial Bus Device Class Definition for Audio Devices.pdf* [available online] <http://www.usb.org>

[7] *Universal Serial Bus Class Definition for Communication Devices.pdf* [online]. [cit. 2008-04-01]. Dostupný z WWW: <http://www.usb.org>

[8] *MPLAB C18 C Compiler User's guide.pdf* [available online] <http://www.microchip.com>

[9] *MPLAB C18 C Compiler Libraries.pdf* [available online]. <http://www.microchip.com>

[10] Slovák, D. *Protocol MIDI and stage equipment: Tomas Bata University*, Faculty of Applied Informatics, Department of Applied Infomatics, 2008, 64 p. Thesis supervisor prof. Ing. Vladimír Vašek, CSc