Universal serial bus digital binary values control pulse width modulation utility

DALIBOR SLOVÁK, STANISLAV PLŠEK
Department of Computer and Communication Systems
Faculty of Applied Informatics
Nad Stráněmi 4511
760 05 Zlín
CZECH REPUBLIC
http://www.fai.utb.cz, slovak@fai.utb.cz, splsek@fai.utb.cz

Abstract: This article describes develop of software that generates managing tensions for many various electrical devices. As already mentioned the name of it is the firmware that is used to control electric devices through the computer. The purpose is to create software interface for managing electrical device which are not available to communicate via digital interface protocols or another digital protocols based on binary values. USB device's firmware and their connection cooperate with most of personal computers operating system such as are Windows, MacOS and Linux. Our device is protected by CZ Patent No. 304 233 and CZ Utility Model No. 024542. Our device is registered at the European Patent Office too.

Key-Words: Control voltage, Microprocessor, single-chip, USB, USB Audio, USB Audio MIDI device, firmware.

1 Introduction

The basis for the specification of the firmware is a general standard USB 2.0, followed by the standard for USB MIDI device and it is based on the actual USB Audio standard. At the present time this is 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 at the PC case, so the external sound cards are used most often with laptops. Placement of DIN connectors in notebooks and external sound cards are precluded due to the size of DIN connectors.

The USB connects USB devices with the USB host. The USB physical interconnect is a tiered star topology. A hub is the centre of each star. Each wire segment is a point-to-point connection between the host and a hub or universal serial bus function, or a hub connected to another hub or universal serial bus function.

Dalibor Slovák Author is with the 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 (corresponding author to provide phone: +420 576 035 271; slovak@fai.utb.cz)

Stanislav Plšek is with the Tomas Bata University in Zlín, nám. T. G. Masaryka 5555, 760 01 Zlín, Czech Republic (corresponding author to provide phone: +420 576 035 274; e-mail: splsek@fai.utb.cz).

2 Solution

We have developed Universal serial bus digital binary values control pulse width modulation device for control of most of electrical devices. The paper describes firmware part of our solution. During our development we had to lay stress on maximally universal firmware for all operating systems. Our firmware is compatible with each universal serial bus devices which enable to reconfigure their firmware. Due to precise keeping of necessary norms for all suitable kinds of universal serial bus device we created Universal serial bus digital binary values control pulse width modulation utility Device Firmware compatible with most of operating systems. Important USB norms for our development were and are:

2.1. 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 most of them is always combined with another USB standards. 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 data flow is based on the relevant standard [10].

2.1.1.MIDI device USB standard

At the beginning it is necessary to say that the typical USB MIDI devices belonging to the USB Communication Device Class (CDC). It is the same as the audio devices are considered as communication interfaces. Class description for the USB MIDI device is one part of standard USB Audio. This happens when a USB device is capable of receiving, respectively send MIDI messages. You have to specify the interface on the device interface level. USB MIDI device will have two interfaces. One is an audio interface, second interface is Musical Instrument Digital Interface. This is 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 were software application Cubase SX 2, respectively Cubase SX 3 and SX4. Another software were Adobe Audition and Sony Soundforge 11. We created own solution and Graphic User Interface for our tests too.

2.2. Own test application

The application has, inter alia, one settings window. User sees all universal serial bus audio musical instruments devices at the upper combo box. User can choose one device for tests. After device choice, user sets values of the MIDI message third byte. The parameter is named Velocity. There is able to set Velocity parameter for up to eight channels in our device. When current control rod is moving, event “Note On” is generating. Exact values of Velocity parameter serve for check of device reaction due to control voltage value changes, respectively for check of whole controlled system reaction due to control voltage value changes. Control voltage value is generated via Velocity parameter. Input value of each controlled values has a form of Musical Instruments Digital Interface (MIDI) message. The second byte of MIDI message is a number identification of each MIDI note – input information for controlled device. The value of MIDI message second byte sets activity of controlled device’s outputs.

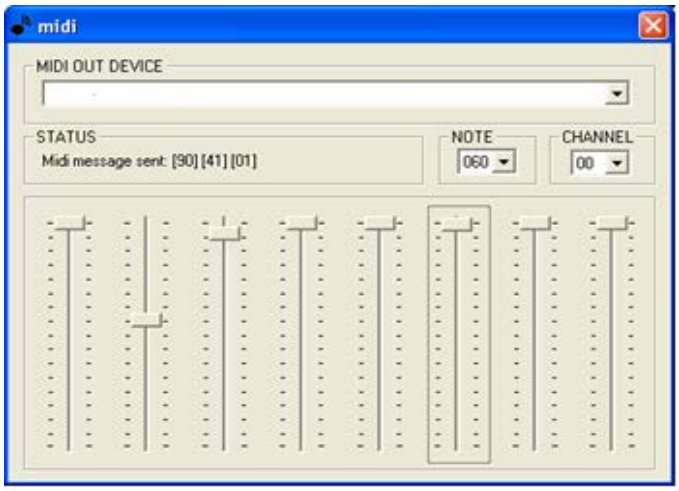

Fig. 1. test application – velocity parameter setting

3 USB Audio MIDI Endpoints and Descriptors

The specifications of endpoints are MIDI Devices descriptors well-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. Transmission of the MIDI data from the host to the MIDI device and back is following.

Information passes from Host and it is addressing to device via External MIDI OUT jack. Then information continues to Embedded MIDI IN jack to device. Now information is in device and it is treated. Treated information is send back via Embedded MIDI OUT jack of this device to External MIDI IN jack of the host.

The description is only virtual abstraction. The stave has to be programmed within the USB device. Everything is numbered via usage the descriptors and it is associated together. The relevant descriptor item shows, which connector belongs to its counterpart.

3.1.1. Universal Serial Bus Musical Instruments Digital Interface firmware descriptors

Now we describe main descriptors categories for our Universal Serial Bus Musical Instruments Digital Interface firmware.

1. Device Descriptor

The Device Descriptors items correspond to the standard CDC device class

2. Configuration Descriptor

Like the device descriptor with current configuration information.

3. Standard AC Interface Descriptor

Audio Control interface does not have any own endpoint. Default endpoint zero is 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 pointers needed to describe a group of audio interfaces in conjunction with particular audio device.

5. Standard MIDI Streaming Interface Descriptor

Standard Interface Descriptor characterizes the device as such. With this 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. This parameter is set in the bJackType variable.

8. MIDI OUT Jack Descriptor

MIDI OUT Jack Descriptor 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 each pin of the MIDI OUT connector, and his status for data transmission.

9. Element Descriptor

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 .

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 [9].

11. Class-Specific MS Bulk Data Endpoint Descriptor

The bNumEmbMIDIJack structure 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 structure contains the ID of the embedded jacks.

12. Standard MS Transfer Bulk Data Endpoint Descriptor

This descriptor also agree with descriptor description from USB specifications chapter 9.6.4., then standard Endpoint descriptor. BEndpoint Adress field designates by the help of D7 parameter, if discuss input transfer endpoint or check out transfer endpoint [3].

3.2. Universal serial bus data flows

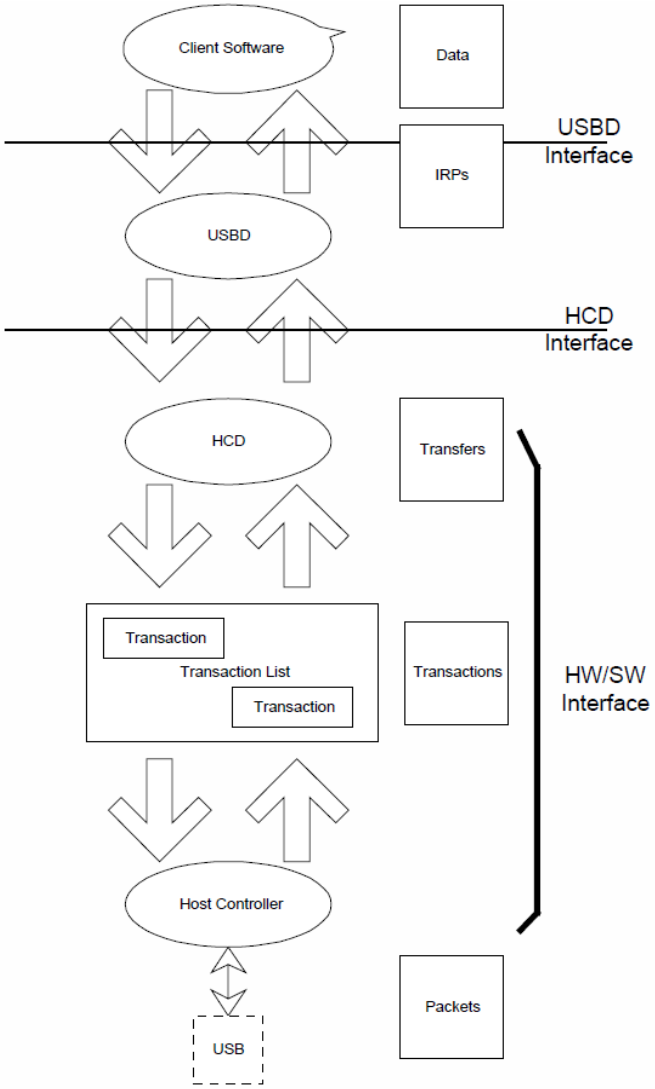

Fig. 2. USB Information Conversion From Client Software to Bus

The USB supports functional data and control exchange between the USB host and a USB device as a set of either uni-directional or bi-directional pipes. USB data transfers take place between host software and a particular endpoint on a USB device. Such associations between the host software and a USB device endpoint are called pipes. In general, data movement though one pipe is independent from the data flow in any other pipe.

A given USB device may have many pipes. As an example, a given USB device could have an endpoint that supports a pipe for transporting data to the USB device and another endpoint that supports a pipe for transporting data from the USB device. 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:

3.2.2.Isochronous transfers

Isochronous transfers transport large amounts of data (up to 1023 B) is guaranteed delivery time, but does not ensure data integrity. Isochronous data is continuous and real-time in creation, delivery, and consumption. Timing-related information is implied by the steady rate at which isochronous data is received and transferred. Isochronous data must be delivered at the rate received to maintain its timing. In addition to delivery rate, isochronous data may also be sensitive to delivery delays. For isochronous pipes, the bandwidth required is typically based upon the sampling characteristics of the associated function. The latency required is related to the buffering available at each endpoint. A typical example of isochronous data is voice. If the delivery rate of these data streams is not maintained, drop-outs in the data stream will occur due to buffer or frame underruns or overruns. Even if data is delivered at the appropriate rate by USB hardware, delivery delays introduced by software may degrade applications requiring real-time turn-around, such as telephony-based audio conferencing. The timely delivery of isochronous data is ensured at the expense of potential transient losses in the data stream. In other words, any error in electrical transmission is not corrected by hardware mechanisms such as retries. In practice, the core bit error rate of the USB is expected to be small enough not to be an issue. USB isochronous data streams are allocated a dedicated portion of USB bandwidth to ensure that data can be delivered at the desired rate. The USB is also designed for minimal delay of isochronous data transfers.

3.2.3.Bulk transfers

Bulk transfers transport of large amounts of data to ensure integrity, but is not guaranteed delivery time. Bulk data typically consists of larger amounts of data, such as that used for printers or scanners. Bulk data is sequential. Reliable exchange of data is ensured at the hardware level by using error detection in hardware and invoking a limited number of retries in hardware. Also, the bandwidth taken up by bulk data can vary, depending on other bus activities.

3.2.4.Interrupt transfers

Interrupt transfers serve a small amount of data to ensure integrity and timely delivery. A limited-latency transfer to or from a device is referred to as interrupt data. Such data may be presented for transfer by a device at any time and is delivered by the USB at a rate no slower than is specified by the device. Interrupt data typically consists of event notification, characters, or coordinates that are organized as one or more bytes. An example of interrupt data is the coordinates from a pointing device. Although an explicit timing rate is not required, interactive data may have response time bounds that the USB must support.

3.2.5.Control transfers

They are used during initial device setup (enumeration) [9].

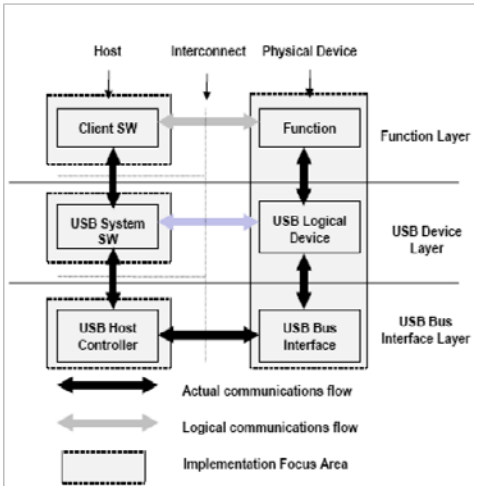

Fig. 3. Layers of universal serial bus communication

4 USB-MIDI Converter

USB-MIDI converter is the kernel 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 [11].

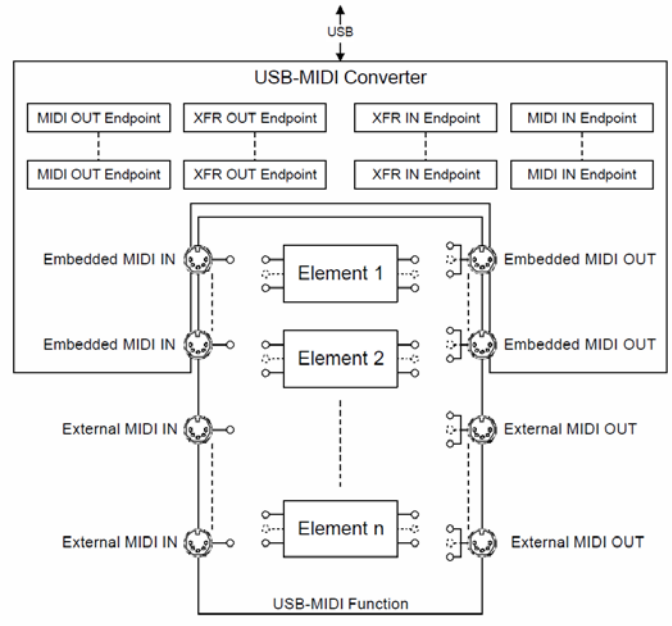

Fig. 4. USB MIDI Converter scheme

4.1. MIDI Endpoints and Embedded MIDI Jacks

The USB-MIDI Converter typically contains one or more MIDI IN and/or MIDI OUT endpoints. These endpoints use bulk transfers to exchange data with the Host. Consequently, a large quantity of USB-MIDI data can simultaneously be sent by an application without missing any MIDI events. Therefore, music applications can perform complex MIDI operations, including sending many MIDI Note On messages at the same time to more smoothly play the most complex music. The information flowing from the Host to a MIDI OUT endpoint is routed to the USB-MIDI function through one or more Embedded MIDI IN Jacks, associated with that endpoint. Information going to the Host leaves the USB-MIDI function through one or more Embedded MIDI OUT Jacks and flows through the MIDI IN endpoint to which the Embedded MIDI Out Jacks are associated. USB-MIDI converters can connect to multiple Embedded MIDI Jacks. Each MIDI Endpoint in a USB-MIDI converter can be connected to up to 16 Embedded MIDI Jacks. Each Embedded MIDI Jack connected to one MIDI Endpoint is assigned a number from 0 to 15. MIDI Data is transferred over the USB in 32-bit USB MIDI Event Packets, with the first 4 bits used to designate the appropriate Embedded MIDI Jack. A 32-bit USB-MIDI Event Packet is adopted to construct multiplexed MIDI streams (MUX MIDI) that can be sent or received by each MIDI Endpoint. At the sending end, multiple individual MIDI streams are placed into constant sized packets (with cable number) and are interleaved into a single MUX MIDI stream. At the receiving end, the multiplexed stream is properly demultiplexed, the data is extracted from the 32-bit USBMIDI Event Packets, and each original MIDI stream is routed to the indicated virtual MIDI port. In this way, one endpoint can have multiple Embedded MIDI Jacks logically assigned. This method makes economical
usage of few endpoints but requires a mux/demux process on both ends of the pipe [11].

4.2. Transfer Endpoints

The USB-MIDI Converter can contains one or more XFR IN and/or XFR OUT endpoints. These endpoints use bulk transfers to exchange data sets between the Host and any of the Elements within the USB-MIDI function. A mechanism of dynamic association is used to link a Transfer endpoint to an Element whenever that Element needs out-of-band data sets exchanged with the Host. A typical application for this endpoint type of is the transfer of down loadable Samples to a Synthesizer Element. The same technique could be used to download program code to an Element that contains a programmable DSP core [5], [10].

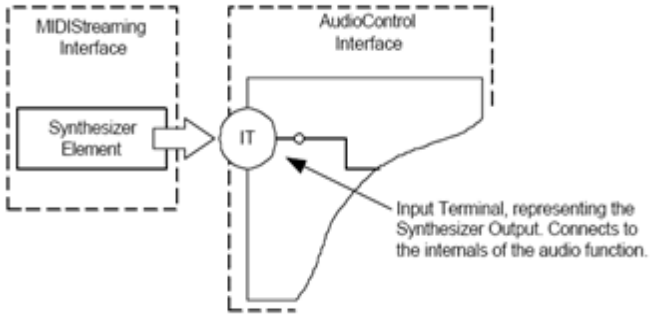

Fig. 5. Input Terminal – Output Terminal

4.3. USB Audio MIDI device GET requests

This request returns the attribute setting of a specific Control inside an Entity of the USB-MIDI function. Additionally, the memory space attribute of an Entity itself can be returned through this GET request. GET request contains following fields.

4.3.1.bmRequestType field

The bmRequestType field specifies that this is a GET request (D7=0b1). It is a class-specific request (D6..5=0b01), directed to either a MIDIStreaming interface of the USB-MIDI function (D4..0=0b00001) or a bulk endpoint of a MIDIStreaming interface (D4..0=0b00010).

The bRequest field contains a constant, identifying which attribute of the addressed Control or Entity is to be returned. Possible attributes for a Control are its:

- Current setting attribute (GET_CUR)
- Minimum setting attribute (GET_MIN)
- Maximum setting attribute (GET_MAX)

- Resolution attribute (GET_RES)

Possible attributes for an Entity are its:

- Memory space attribute (GET_MEM)

4.3.2. wValue field

The wValue field interpretation is qualified by the value in the wIndex field. Depending on what Entity is addressed, the layout of the wValue field changes. The following paragraphs describe the contents of the wValue field for each Entity separately. In most cases, the wValue field contains the Control Selector (CS) in the high byte. It is used to address a particular Control within Entities that can contain multiple Controls. If the Entity only contains a single Control, there is no need to specify a Control Selector and the wValue field can be used to pass additional parameters.

4.3.3.wIndex field

The wIndex field specifies the interface or endpoint to be addressed in the low byte and the Entity ID or zero in the high byte. In case an interface is addressed, the virtual Entity 'interface' can be addressed by specifying zero in the high byte. The values in wIndex must be appropriate to the recipient. Only existing Entities in the USB-MIDI function can be addressed and only appropriate interface or endpoint numbers may be used. If the request specifies an unknown or non-Entity ID or an unknown interface or endpoint number, the control pipe must indicate a stall.

4.3.4.wLength field

The actual parameter(s) for the Get request are returned in the data stage of the control transfer. The length of the parameter block to return is indicated in the wLength field of the request. If the parameter block is longer than what is indicated in the wLength field, only the initial bytes of the parameter block are returned. If the parameter block is shorter than what is indicated in the wLength field, the device indicates the end of the control transfer by sending a short packet when further data is requested. The layout of the parameter block is qualified by both the bRequest and wIndex fields. Refer to the following sections for a detailed description of the parameter block layout for all possible Entities [11].

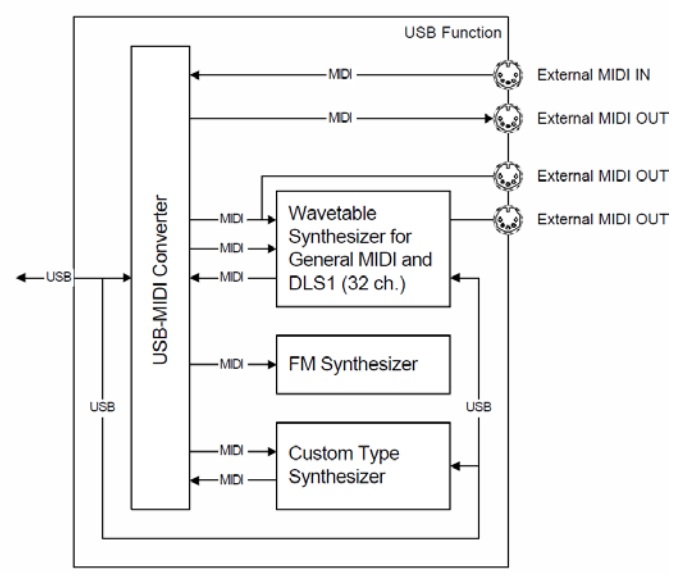

Fig. 6. Complex USB MIDI Device

5 Firmware – the most important modules and files

5.1. User.c and user.h

Fundamental module for user needs are two source text files user.c and user.h, that contains custom functions setting and macro. User makes pertinent modification necessary for given functionality of device firmware arrangement. There is module separated from of others in this text too. There 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 of switched and no switched states for individual arrangements, so setting of timer for connect or disconnect handled devices via pulse width of control voltage [3], [4].

5.2. Other important USB communication modules

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.

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 USB classes of devices (HID, MSD, CDC) do not require any intervention without a deeper study. On the contrary any changes can be detrimental.

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

usb_compile_time_validation.h – Endpoint descriptors size verification according to standard of USB. Then descriptor reach size can be either 8, 16, 32 or 64 bytes.

usbcfg.h – By the help of this file is performed endpoint configuration of device. So it is set value and default setting of endpoint zero 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.

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 Universal serial bus digital binary values control pulse width modulation utility were some interventions required and they are described in my other paper. Modules described below relate to each class. The program usage depends on what category programmed device is included.

6 Conclusion

The paper is second part for paper Universal serial bus digital binary values control pulse width modulation utility. We created device for control lightning and water fountain systems. We tested electric motors too. Our tests were successful. We have protected our device by CZ patent and CZ utility model. Our device is entered at European patent office too. Presently we develop electric voltage transducers for another forms of usage of our device. The description of hardware for the Universal serial bus digital binary values control pulse width modulation utility firmware is in next paper. This work was supported by the Ministry of Education, Youth and Sports of the Czech Republic within the National Sustainability Programme project No. LO1303 (MSMT-7778/2014).

References:

[1] D. Slovak, "General-purpose single-chip device" CZ Patent No. 304233. 15.1.2014.

[2] D. Slovak, "General-purpose single-chip device". Czech republic. Utility model No. 024542. 12.11.2012.

[3] D. Slovak, "USB MIDI Lights Device". in *Proc Recent Research in Automatic Control*. Lanzarote, Canary Islands: WSEAS Press, 2011, pp. 300-306. ISBN 978-1-61804-004-6.

[4] D. Slovak, "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.

[5] D. Slovak, "Pulse Width Modulation Control Voltage Interface," in *Proc. Recent Advanced in Communication, Circuits and Technological Innovations.* Paris, France: North Atlantic University Union, 2012, pp. 252-257.

[6] R. Tzeneva, Y. Slavtchev, N. Mastorakis, V.Mladenov, "New Design of Aluminum Bolted Busbar Connections", *Proceedings of the 13th WSEAS International Conference on CIRCUITS, WSEAS CSCC Multiconference*, Rhodos Island,Greece, July 22-24, 2009, pp. 172-177

[7] D. Slovak, "Universal serial bus musical instrument digital interface hardware," *International Journal of Mechanics. Journal* [online]. 5(4), pp. 294-301.

[8] T. Modegi, Shun-ichi Iisaku, *Proposals of MIDI Coding and its Application for Audio Authoring*, MMCS, *IEEE International Conference*, pp 305 – 314 , 1998.

[9] *Universal Serial Bus specification.pdf*, Available: http://www.usb.org

[10] *Universal Serial Bus Device Class Definition for Audio Devices.pdf* Available: http://www.usb.org/developers/docs/devclass_docs/audio10.pdf

[11] *Universal Serial Bus Device Class Definition for MIDI Devices.pdf* Available: http://www.usb.org/developers/docs/devclass_docs/midi10.pdf

[12] STORK, M., HRUSAK, J., MAYER, D.: „Nonlinearly Coupled Oscillators and State Space Energy Approach". 14th WSEAS International Conference on SYSTEMS (Part of the 14th WSEAS CSCC Multiconference), Corfu Island, Greece, 2010.