USB Capabilities and Bootability of Portable Devices

Siddharth Kaul, Kamakshi Kaul, Parth Maheta

Abstract— Almost every portable devices uses Universal Serial Bus(USB) for PC connectivity and to enhance their capabilities. Such capabilities include interfacing and connection of features like charging, Audio, Host and Video. This paper implements and explains the basic USB functionality of USB Audio, USB Mass Storage and Secondary Bootloading through USB channel. This paper provides the basic information one requires to implement these device classes using available stack library by clearly explaining the important aspect in implementation. It is also shown that a portable device can be read, write and reprogrammed using another portable device using the USB interconnection. By performing the implementation, readers can take the first steps towards non PC centric development environment.

Index Terms— Universal Serial Bus, USB Audio class, USB Mass Storage class, Secondary Bootloader.

1 INTRODUCTION

This paper demonstrates use of USB stack by implementing USB Audio and Mass Storage Classes on a NXPLPC1768 board and develops the idea of booting a portable device using an OTG enabled portable device.

USB has enough bandwidth for sound, even high-quality audio. Many applications relating to audio playback and audio recording can take advantage of this large bandwidth of USB. A versatile bus specification like USB requires standardized audio transport mechanism to keep software drivers as generic as possible. A major issue in audio is synchronization of data streams. A robust synchronization scheme on isochronous transfer has been developed and incorporated in USB specification. Audio Device class definition adheres to robust synchronization scheme to transport data reliably over the bus.

USB Mass Storage Class specifications are supported by USB Mass Storage Class Working Group. The focus of this paper is only on USB Mass Storage Class Bulk Only Transport specification and USB Mass Storage Class Bootability Specification. Bulk Only Transport is the transport of command, data and status occurring solely through via bulk endpoints (not via interrupt and control endpoints). Bootability specification defines a set of commands and associated data sufficient to allow the loading of a program or an operating system stored on a USB Mass Storage Class Device.

USB OTG is supplement to USB2.0 specification that augments the capability of existing portable devices and USB peripherals by adding host functionality for connection to USB peripherals. This supplement enables point to point connections and ability of a portable device to play dual role of either host or peripheral and allowing dynamic switching between these two roles.

All vendors provide their own custom stack for their independent chips to be used as stack or apis for USB. The experimental board used landtiger is a custom designed derivative of Keil's MCB1700 board. So Keil custom stack for MCB1700 has been implied in this experimentation. The core files are used as it is and the descriptor files has been modified to suit interface and endpoint descriptors are defined by developer according to guidelines provided in the Audio Device Class the board and rearranged for better comparison with the specification. This paper will help the developer understand and decipher the randomness that is the USB stack.

2 IMPLEMENTATION OF USB AUDIO

2.1 Introduction

USB Audio is basically a completely different protocol written over the base USB specification. Audio function is located at the interface level of device class hierarchy. Audio functions are addressed through their audio interfaces. Each audio function has a single AudioControl interface and can have several AudioStreaming and MIDIStreaming interfaces. The AudioControl (AC) interface is used for the control of audio functions whereas the AudioStreaming (AS) interfaces and MIDIStreaming interfaces (MS) are used for transport audio streams and MIDI data streams into and out of the audio function respectively. The collection of the single AudioControl interface and the AudioStreaming and MIDIStreaming interfaces belonging to the same audio function is called the Audio Interface Collection (AIC). A device can have multiple Audio Interface Collections active at the same time. These Collections are used to control multiple independent audio functions located in the same composite device.

2.2 Implementation

The audio stream is mono channel and sampled at 32khz. Timer is set to generate interrupts periodically at 31.25usecs. A service routine to serve this interrupt is developed that not only buffers the audio stream that's being received but also performs the volume adjustments in the output stream. Similarly the plugging of USB generates a interrupt USB_IRQn that again is added to Nested Vector Interrupt Control and service routine to the same is developed. The USB Interrupt service routine reads device status, interrupts status and calls the routines based on the error the interrupt was triggered. The interrupt in USB is triggered by standard USB events like power up event, device read event etc. The Device, configuration, Specification. The standard request and Audio device Class specific request handling is done by the core USB library pro-

vided in the stack. The Endpoint requests are also handled by the core libraries but a developer can do changes to required endpoint by creating independent endpoint handlers.

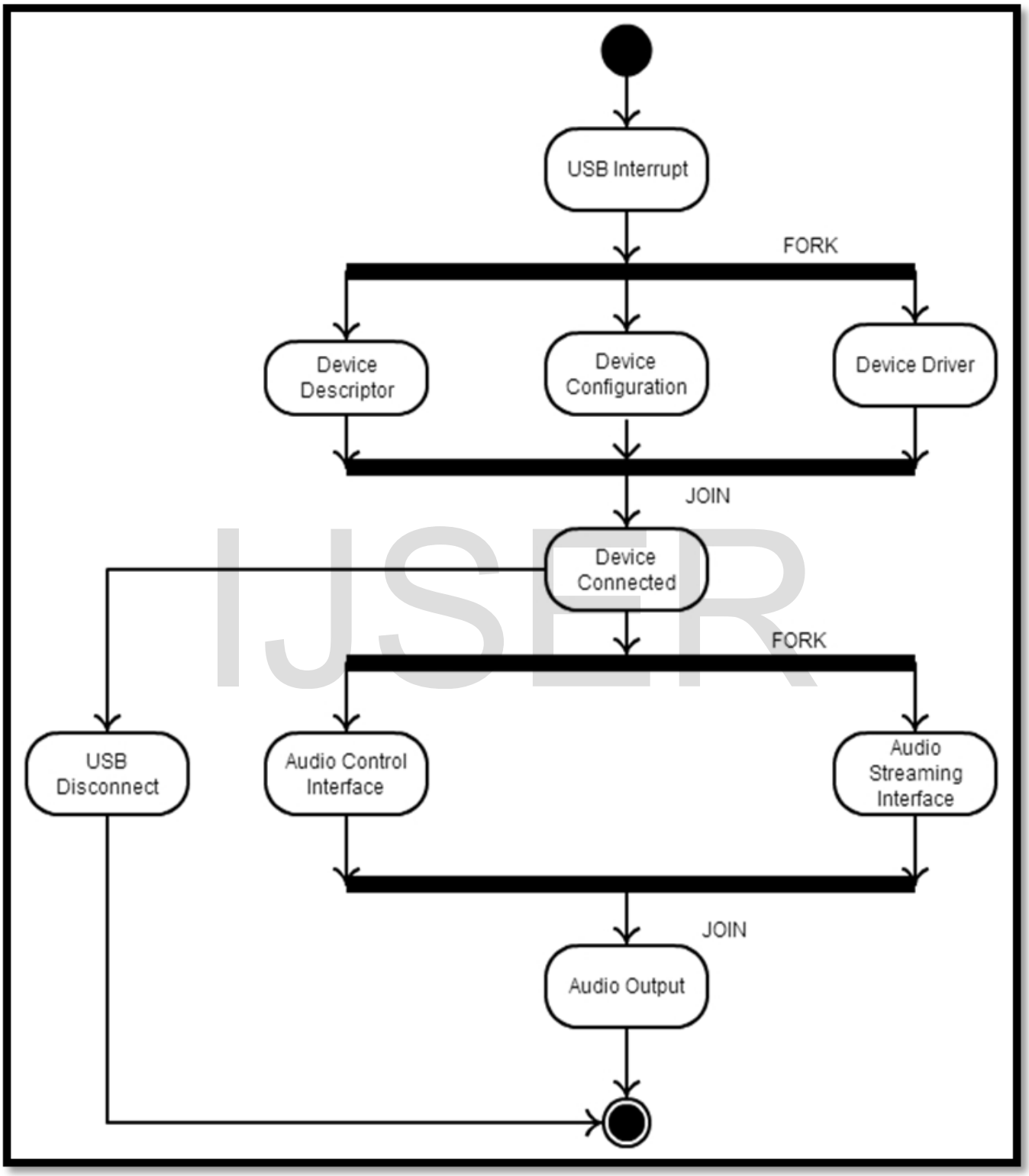

Fig.1. Flow Chart Describing implementation of USB Audio Class

3 IMPLEMENTATION OF USB MASS STORAGE

3.1 Introduction

USB Mass Storage Class specifications are supported by the USB Mass Storage Class Working Group (CWG). The titles of these specifications are:

1. USB Mass Storage Class Control/Bulk/Interrupt (CBI) Transport
2. USB Mass Storage Class Bulk-Only (BBB) Transport
3. USB Mass Storage Class UFI (UFI) Command Specification
4. USB Mass Storage Class Bootability Specification
5. USB Mass Storage Class Compliance Test Specification
6. USB Lockable Storage Devices Feature Specification (LSD FS)
7. USB Mass Storage Class USB Attached SCSI Protocol (UASP)

This paper deals with implementation of Bulk only transport and bootability.

3.2 Implementation

A Bulk-Only Protocol requires a host to send a CBW to the device and then attempt to make the appropriate data transfer (In, Out or none). The device receives the Control Block Wrap-

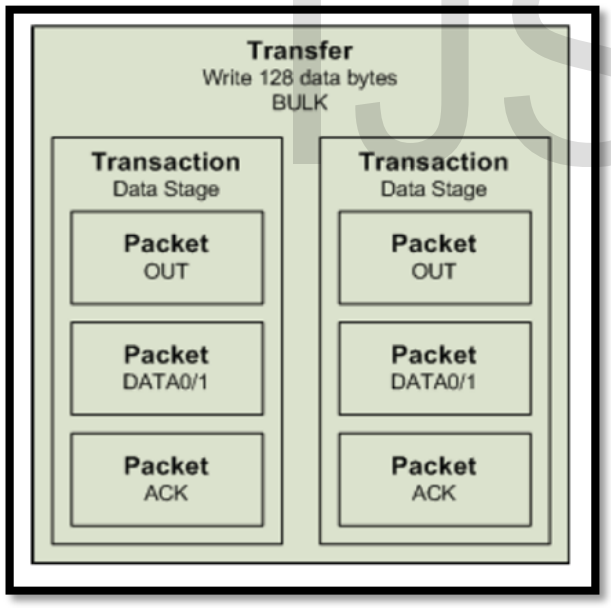

Fig. 2. Two stage transaction of 128 data bytes

-per (CBW), checks and interprets it, attempts to satisfy the host's request, and returns status via a Control Status Wrapper (CSW). With bulk transfers, a maximum of 64 bytes may be transferred in a single data stage transaction. A bulk transfer that writes 128 data bytes requires two data stage transactions, and has the structure as shown Fig.2.

There are four major tasks our device needs to perform to be a mass storage device.

1. First is to get control block wrapper (CBW) which is nothing but a command block and its associated information.
2. Second and third is to either get the bulk data input or provide with a bulk data output.
3. Fourth is to set the control status wrapper (CSW) which is a packet containing status of a command block.

This CSW will inform the host about the receipt and validation of CBW and that its ready for another CBW. The host cannot transfer another CBW to the device until the host has received the CSW for any outstanding CBW. If the host issues two consecutive CBWs without an intervening CSW or reset, the device response to the second CBW cannot be determined. The mass storage class header files provide definitions of function that is going to perform the get CBW function, set CSW function and perform bulk data transfer. This file also provides with all the necessary functions require by device to be a mass storage device. These include memory read, write and verify function, stall endpoint routine and bulk transfer routines. These also comply to SCSI protocol for bulk transfer enabling them to be made into bootable disk drives. Except for use of DMA for data transfer rest of all the core and hardware libraries remains the same as that used in USB Audio Device Class. Developer again needs to provide standard USB and Mass Storage Class Device, Configuration, interface, endpoint and vendor descriptors. Fig. 3. Describes implementation of USB Mass Storage class as a sequence diagram.

4 SECONDARY BOOTLOADING THROUGH USB CHANNEL

4.1 Introduction

Almost all modern 32 bit microcontrollers have two methods for bug fixes and product updates.

- ISP (In System Programming) is programming or reprogramming the on-chip flash memory, using the boot loader software and UART0 serial port. This can be done when the part resides in the end-user board.
- IAP (In Application Programming) is performing erase and write operation on the on-chip flash memory, as directed by the end-user application code.

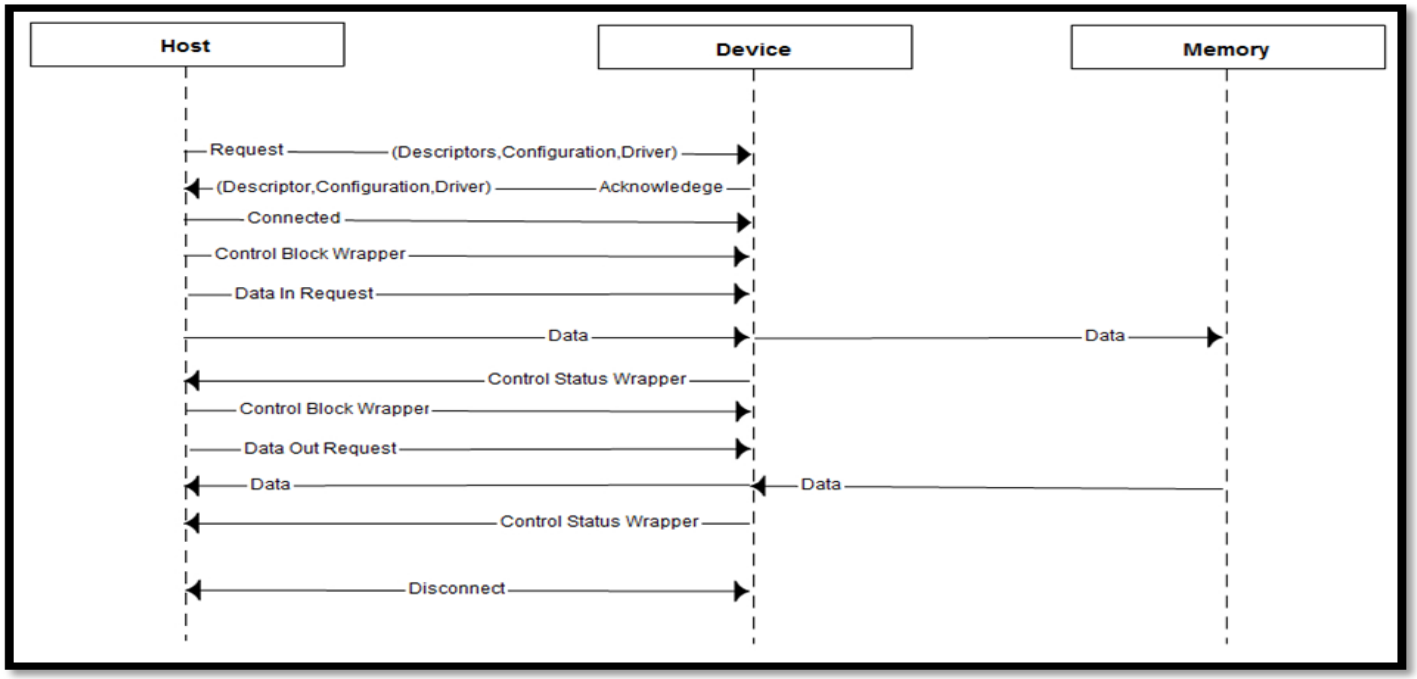

Fig. 3. Sequence diagram describing implementation of USB Mass Storage class

There is primary bootloader and a secondary bootloader. A primary bootloader is a firmware that resides in a microcontroller's boot ROM block and is executed on power-up and resets. The secondary bootloader is a piece of code which allows a user application code to be downloaded using alternative channels other than the standard methods. After boot the secondary bootloader will be executed that in turn will execute the end user program or application. The concept of this is the basis of our theory that says that using secondary bootloader in target microcontroller and an OTG enabled device or network based device we can reprogram our targeted microcontroller through other than standard channels like USB or LAN (Local Area Network). This will give end user freedom to apply a single hardware for multiple control and acquisition tasks by just reprogramming the target hardware.

4.2 Description

A standard Mass Storage Class Device provides separate segments for SD Card and on chip memory. The Mass Storage Class with USB secondary bootloader provides segment only for flash memory. This segment is displayed as extended disk on a computer. Now on a portable device say for example a smartphone or tablet, the file format reading capabilities are limited by their phone OS and most of them are able to read only FAT32 system with few phones supporting all FAT12, FAT32, exFAT and NTFS file system. Now as most of the microcontrollers this paper has dealt with have less than 32MB of on chip memory, the file system of choice obviously becomes FAT12. FAT12 is the preferred choice as FAT12 doesn't require segmentation and can have a stable segment of upto 32MB (Maximum volume of drive). Moreover for simplification purpose the flash code will appear as one single entity, meaning a single file, solving any defragmentation problem. The access to flash memory of target device can be developed in either FAT12 or FAT32 file system depending on size of on chip memory. The experiment performed was pretty simple; a laptop was used to reprogram NXP 1768 with just simple file transfer. A standard NXP secondary USB bootloader was used with hardware entry into bootloader, modified in the code. Hardware entry is provided to prevent accidental entry into the secondary bootloader.

5 Conclusion and Future Work

The implementation of device class and interface has confirmed the theory that a USB enabled portable device can be reprogrammed by just a simple file transfer. Hence this can be the first step to move away from our PC-centric environment and become completely mobile, both in connectivity and development.

The future work includes to be able to perform reprogramming using an OTG enabled phone, to implement FAT32 file system for the bootloader and to check the fault tolerance and behavior of both bootloader and the application that will be loaded. Moreover to find the limitation of FAT32 to be used as a bootloader file system and finding a better way to make bootloader compatible to maximum number of available OTG enabled devices.

References

[1] Gal Ashour, Billy Brackenridge, Oren Tirosh, Craig Todd, Remy Zimmermann, Geert Knapen Universal Serial Bus Device Class Definition for Audio Devices Release 1.0 March 18, 1998.

[2] Gal Ashour, Billy Brackenridge, Oren Tirosh, Craig Todd, Remy Zimmermann, Geert Knapen Universal Serial Bus Device Class Definition for Audio Data Formats Release 1.0 March 18, 1998.

[3] Gal Ashour, Billy Brackenridge, Oren Tirosh, Craig Todd, Remy Zimmer-

mann, Geert Knapen Universal Serial Bus Device Class Definition for Terminal Types Release 1.0 March 18, 1998.

[4] Al Rickey, Alan Haffner, Bill Stanley, Calaimany Bhoopathi, Curtis E. Stevens, Darrell Redford Universal Serial Bus Mass Storage Class Bulk Only Transport Revision 1.0 September 31,1999.

[5] John Garney, Ken Stufflebeam, David Wooten, Matt Nieberger, John Howard, Steve McGowan Universal Serial Bus Revision 2.0 April 27, 2000.

[6] Doug Azzarito, Fred Bhesania, Jim Blackson, Mark Bohm, Robert Chang, Jason Chien Universal Serial Bus Mass Storage Specification for Bootability Revision 1.0 October 25, 2004.

[7] Geert Knapen, Dan Ellis, Jim Koser, Yagal Blum, Dave Podsiadlo, Cristial Chis Universal Serial Bus Device Class Definition for Basic Audio Devices Release 1.0 November 24, 2009.

[8] Amit Nanda, Hans van Antwerpen, Chuck Trefts, Mario Pasquali, Yuji Oishi, Steve McGowan Universal Serial Bus Mass Storage Class Specification Overview Revision 1.4 February 19, 2010

[9] Jing Wang, Kenneth Ma, Thomas Hackett, Pawel Eichler, Brad Saunders, Steve McGowan Universal Serial Bus Mass Storage Class Specification For UASP Bootability Revision 1.0 March 04, 2013.