Logic Fruit Technologies
White Paper

806, 8th Floor,
BPTP Park Centra,
Sector – 30, Gurgaon.
Pin: 122001
T: +91-124-4117336
W: http://www.logic-fruit.com

USB Host

By:
Abhinav saxena
R&D Engineer

Abstract

Implementing an embedded system with a USB host support can be a challenging task. This paper will provide software guidelines for such integrated solutions, and the factors to be considered/taken care while building these type of systems.

Index

Introduction

Motivation

USB aims to simplify things by extending the trend of "user friendliness" to the hardware level. To the average computer user, it is a system where you can simply plug a device into any available socket and that device will instantly be available for use by the computer. Up to 127 devices can be connected, and since it is a high speed system supporting up to 12 Megabits per second, it can accommodate the needs of a wide variety of peripherals. Other advantages include the ability to safely disconnect and reconnect items without switching off the computer, and the ability to use a USB device on any computer supporting the USB system.

What is USB Host?

Description

The USB is based on a so-called 'tiered star topology' in which there is a single host controller and up to 127 'slave' devices. The host controller is connected to a hub, integrated within the PC, which allows a number of attachment points (often loosely referred to as ports). A further hub may be plugged into each of these attachment points, and so on. However there are limitations on this expansion.

As stated above a maximum of 127 devices (including hubs) may be connected. This is because the address field in a packet is 7 bits long, and the address 0 cannot be used as it has special significance. (In most systems the bus would be running out of bandwidth, or other resources, long before the 127 devices was reached.)

A device can be plugged into a hub, and that hub can be plugged into another hub and so on. However the maximum number of tiers permitted is six.

The length of any cable is limited to 5 metres. This limitation is expressed in the specification in terms of cable delays etc, but 5 metres can be taken as the practical consequence of the specification. This means that a device cannot be further than 30 metres from the PC, and even to achieve that will involve 5 external hubs, of which at least 2 will need to be self-powered.

So the USB is intended as a bus for devices near to the PC. For applications requiring distance from the PC, another form of connection is needed, such as Ethernet.

Class Codes

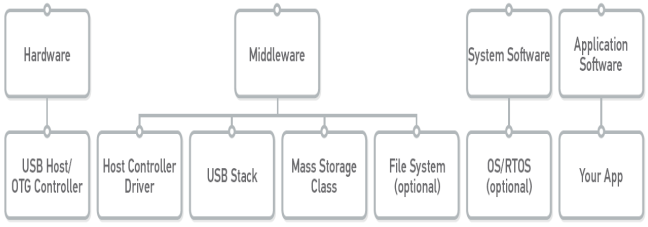
Base Class	Descriptor Usage	Description
00h	Device	Use class information in the Interface Descriptors
01h	Interface	Audio
02h	Both	Communications and CDC Control
03h	Interface	HID (Human Interface Device)
05h	Interface	Physical
06h	Interface	Image
07h	Interface	Printer
08h	Interface	Mass Storage
09h	Device	Hub
0Ah	Interface	CDC-Data
0Bh	Interface	Smart Card
0Dh	Interface	Content Security
0Eh	Interface	Video

0Fh	Interface	Personal Healthcare
10h	Interface	Audio/Video Devices
11h	Device	Billboard Device Class
12h	Interface	USB Type-C Bridge Class
DCh	Both	Diagnostic Device
E0h	Interface	Wireless Controller
EFh	Both	Miscellaneous
FEh	Interface	Application Specific
FFh	Both	Vendor Specific

USB Host Model

Embedded host for a USB flash drive is a complex combination of hardware and software that works together as a system.

A key component of this system is software that we call 'middleware.' Middleware is a collection of software consisting of drivers, protocol stacks and class drivers that make it possible to communicate with the USB devices via the USB host controller.



Types of Host Controller

There are three commonly encountered types of USB host controller, each with its own history and characteristics.

OHCI (Open Host Controller Interface)

Compaq, Microsoft and National Semiconductors cooperated to produce this standard host controller specification for USB 1.0 and USB 1.1. It is a more hardware oriented version than UHCI. Low speed and full speed.

UHCI (Universal Host Controller Interface)

Intel's more software-oriented version of a controller for USB 1.0 and USB 1.1. Requires a license from Intel. Low speed and full speed.

EHCI (Extended Host Controller Interface)

When USB 2.0 appeared with its new high speed functionality, the USB-IF insisted on there being a single host controller specification, to keep device development costs down. The EHCI handles high speed transfers, and hands off low and full speed transfers to either OHCI or UHCI companion controllers.

Existing controllers having USB Host feature

Supplier	Family	Architecture	USB OTG or Host Support
NTI	Tiva	ARM	FS/HS Device & OTG
TI	Hercules	ARM	LS/FS OHCI, FS Device
TI	Sitara	ARM	FS/HS Host & Device
TI	OMAP	ARM	FS/HS Host & Device
Freescale	Kinetis	ARM	FS/HS Host & Device
Freescale	Vybrid	ARM	FS/HS Host & Device
Freescale	i.MX	ARM	FS/HS Host & Device
Atmel	AVR	AVR	FS OTG & Device
Atmel	SAM	ARM	FS Host & Device
NXP	LPC	ARM	FS/HS Host & Device
Microchip	PIC	PIC	FS/HS Host & Device

Functional Description

USB Host Flow Control

- When USB device is plugged in, the host becomes aware (because of the pullup resistor on one data line), that a device has been plugged in.
- The host now signals a USB Reset to the device, in order that it should start in a known state at the end of the reset. In this state the device responds to the default address 0. Until the device has been reset the host prevents data from being sent downstream from the port. It will only reset one device at a time, so there is no danger of two devices responding to address 0.
- The host will now send a request to endpoint 0 of device address 0 to find out its maximum packet size. It can discover this by using the Get Descriptor (Device) command. This request is one which the device must respond to even on address 0.
- Typically (i.e. with Windows) the host will now reset the device again. It then sends a Set Address request, with a unique address to the device at address 0. After the request is completed, the device assumes the new address. (And at this point the host is now free to reset other recently plugged-in devices.)
- Typically the host will now begin to quiz the device for as many details as it feels it needs. Some requests involved here are:
  - Get Device Descriptor
  - Get Configuration Descriptor
  - Get String Descriptor
- At the moment the device is in an addressed but unconfigured state, and is only allowed to respond to standard requests.
- Once the host feels it has a clear enough picture of what the device is, it will load a suitable device driver.
- The device driver will then select a configuration for the device, by sending a Set Configuration request to the device.
- The device is now in the configured state, and can start working as the device it was designed to be. From now on it may respond to device specific requests, in addition to the standard requests which it must continue to support.
- We can now see that there is a set of requests which a device must respond to, and need to look at the detailed means by which the requests are conveyed.
- The only transfer type available before the device has been configured is the Control Transfer. The only endpoint available at this time is the bidirectional Endpoint 0.

Implementation

- When a device is attached to the USB system, it gets assigned a number called its ***address***. The address is uniquely used by that device while it is connected and, unlike the traditional system, this number is likely to be different to the address given to that device the last time it was used. Each device also contains a number of ***endpoints***, which are a collection of sources and destinations for communications between the host and the device.

Endpoints

- Each USB device has a number of endpoints. Each endpoint is a source or sink of data. A device can have up to 16 OUT and 16 IN endpoints.
- OUT always means from host to device.
- IN always means from device to host.
- Endpoint 0 is a special case which is a combination of endpoint 0 OUT and endpoint 0 IN, and is used for controlling the device.

Pipe

A logical data connection between the host and a particular endpoint, in which we ignore the lower level mechanisms for actually achieving the data transfers.

- The combination of the address, endpoint number and direction are what is used by the host and software to determine along which pipe data is traveling.
- USB, sends a block of data called an I/O Request Packet (IRP) to the appropriate pipe, and the software is later notified when this request is completed successfully or terminated by error.
- Data transmission in the bus occurs in a serial form. Bytes of data are broken up and sent along the bus one bit at a time, with the least significant bit first

Types of Packets

To cope with the various communications that must occur to establish a data flow, there needs to be a variety of types of packets, and these are as follows:

- ***Token Packets*** : These are used to query the device and are issued by the host. They consist of PID, address and endpoint fields, along with a 5 bit CRC check.

- ***Start-of-Frame Packets*** : The USB host controls the processing of data in 1ms units called frames. During each frame, it examines what requests are outstanding and allocates each pipe bandwidth depending on its requirements and type of transfer that it uses. Each frame is marked by a Start of Frame (SOF) packet, consisting of an appropriate PID, an 11-bit counter and a 5 bit CRC. The counter is incremented once per frame, allowing devices to determine whether they missed a frame due to an error and adjust their timing appropriately.

  - At high speed the 1 ms frame is divided into 8 microframes of 125 us. A SOF is sent at the start of each of these 8 microframes, each having the same frame number, which then increments every 1 ms frame.

- ***Data Packets*** : Consists of all of the above field types, and is protected by a 16 bit CRC

  - DATA0 and DATA1 PIDs are used in Low and Full speed links as part of an error-checking system. When used, all data packets on a particular endpoint use an alternating DATA0 / DATA1 so that the endpoint knows if a received packet is the one it is expecting. If it is not it will still acknowledge (ACK) the packet as it is correctly received, but will then discard the data, assuming that it has been re-sent because the host missed seeing the ACK the first time it sent the data packet.

  - DATA2 and MDATA are only used for high speed links.

- ***Handshake Packets*** : These are used for returning the status of data transfers and consist only of a PID. These come in three types and are named ACK (ACKnowledgement of receipt with no errors), NAK (the function is not ready to communicate data) and STALL (function is busy or some other error occurred)

  - ACK : Receiver acknowledges receiving error free packet.
  - NAK : Receiving device cannot accept data or transmitting device cannot send data.
  - STALL : Endpoint is halted, or control pipe request is not supported.
  - NYET : No response yet from receiver

(high speed only)

Packet Format

- An eight bit "***SYNC***" synchronisation field used by inputs to correct their timing for accepting data. Part of this field is a special symbol used to mark the start of a packet.

- The 8 bit Packet Identifier (***PID***) which uses 4 bits to determine the type, and hence format, of the packet data. The remaining 4 bits are a 1's complement of this, acting as check bits. Part of this field determines which of the four groups (token, data, handshake, and special) that the packet belongs to, and also specifies an input, output or setup instruction.

PID Type	PID Name	PID<3:0>*
Token	OUT	0001b
	IN	1001b
	SOF	0101b
	SETUP	1101b
Data	DATA0	0011b
	DATA1	1011b
	DATA2	0111b
	MDATA	1111b
Handshake	ACK	0010b
	NAK	1010b
	STALL	1110b
	NYET	0110b
Special	PRE	1100b
	ERR	1100b
	SPLIT	1000b
	PING	0100b
	Reserved	0000b

- An ***address field*** which gives the address of the function on the end of the pipe to be used

- The 4 bit ***endpoint field***, giving the appropriate endpoint which sends or receives the packet.

- A ***data field*** consisting of 0-1023 bytes

Sync (8)	PID (8)	Address	Endpoint (4)	Data (0-1023 bytes)

Figure : A typical data packet. Numbers represent size of field in bits, unless otherwise indicated.

Types of Data Transfers -

- ***Control Transfers*** : These differ from the other types in that they are intended for use in configuring, controlling, and checking the status of a USB device. A request is sent to the device from the host, and appropriate data transfers follow in the appropriate pipes. At some later stage, a status indicator is returned to the host. The pipe used for this type of data may be bidirectional, but uses the same numbered endpoint for each direction. In addition, a device only handles one control request at a time, with the host withholding outstanding requests until a status is returned on the one in progress. For example, the Default Control Pipe uses Control Transfers and accomplishes such tasks as initializing the device, and telling the host of the requirements of each of its endpoints. This type of pipe might also be used to control the operation of other pipes.

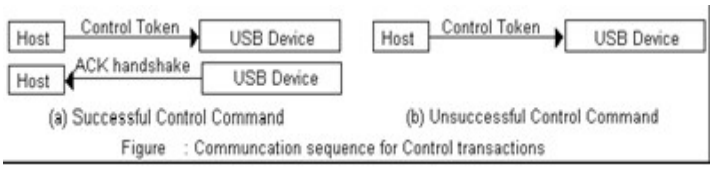


Figure : Communcation sequence for Control transactions

- ***Isochronous Transfers*** : These involve data whose accuracy is not critical and which is sent at a rate corresponding to some timing mechanism. For example, 44100KHz audio fits into this category since it doesn't have to be perfectly accurate and every 44100 samples indicates one second of audio. USB provides a special type of transfer for this data, giving it preference to guarantee a constant transmission rate with the required bandwidth. To ensure that the USB has enough time to handle the maximum data flow (1023 bytes) in each frame, a check is made during the initial configuration and the pipes will only be configured if this check is successful. This transfer method uses unidirectional pipes with no error handling procedures. Even though an error may be indicated in the status reply to a request, the pipe will not be halted and it is up to the software to decide what to do.

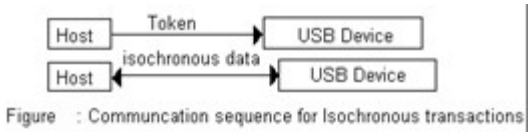


Figure : Communcation sequence for Isochronous transactions

- ***Interrupt Transfers*** : These are used for small, infrequent transfers which require priority over other requests. As with Isochronous transfers, pipe configuration is granted on whether or not the system can handle the maximum packet size within the required time, with a further restriction that stops Interrupt and Isochronous Transfers from using more than 90% of any frame (discussed later) and stopping other transfers from occurring. The endpoint tells the host during configuration how often it should be polled for interrupt requests, and upon each polling returns a NAK signal if there is nothing to send. The use of this type of pipe is in some ways similar in purpose to the IRQ lines of the traditional peripheral system used in computers.

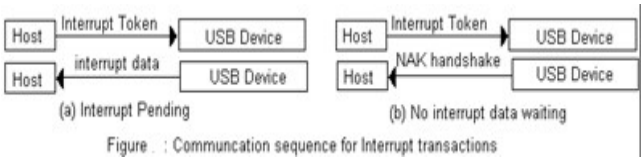


Figure : Communcation sequence for Interrupt transactions

- ***Bulk Transfers*** : As the name suggests, the intended purpose is for transmitting large amounts of data. This type of transfer gets the lowest priority, so pipes using this method are only allowed to transmit when there is available bandwidth. This means that a heavily loaded USB may have relatively slow bulk transfers compared to one with is servicing few devices. This transfer type would be useful for sending data from devices like digital scanners.

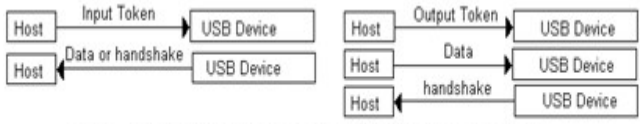
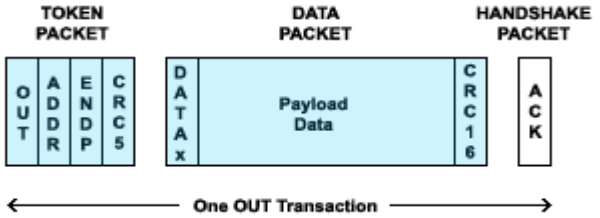
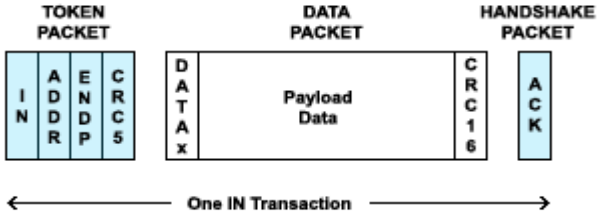


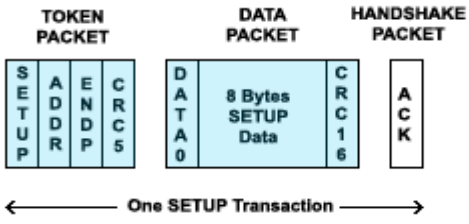
Figure : Communcation sequence for input and output bulk transactions

Token Transaction

- Simple transfers of data called 'Transactions' are built up using packets.
- A successful transaction is a sequence of three packets which performs a simple but secure transfer of data.
- For IN and OUT transactions used for isochronous transfers, there are only 2 packets; the handshake packet on the end is omitted. This is because error-checking is not required.
- There are three types of transaction. In each of the illustrations below, the packets from the host are shaded, and the packets from the device are not.







Setup Packet

- The Standard requests are all conveyed using control transfers to endpoint 0. Remember that a control transfer starts with a SETUP transaction which conveys 8 bytes. These 8 bytes define the request from the host.
- The structure of bmRequestType makes it easy to use it to switch on when our firmware is trying to interpret the setup request. Essentially, when the

SETUP arrives, you need to branch to the handler for the particular request, so for example bits 6:5 allow you to distinguish the mandatory standard commands, from any class or vendor commands you may have implemeted for you particular device.

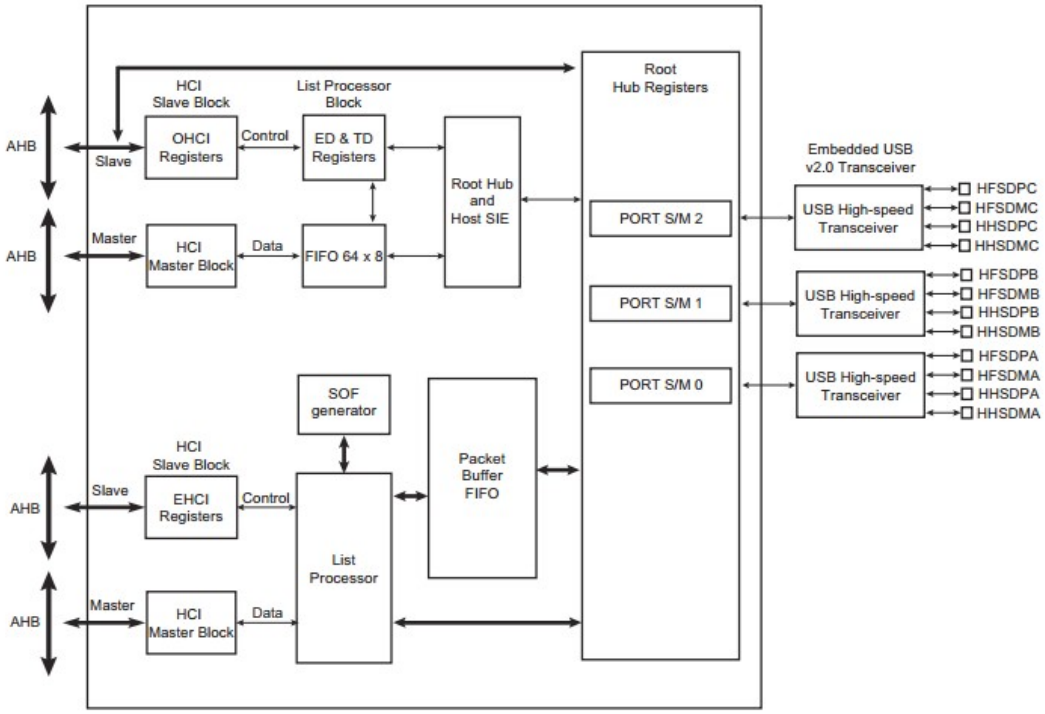
- Switching on bit 7 allows you to deal with IN and OUT direction requests in separate areas of the code.

Offset	Field	Size	Value	Description
0	bmRequestType	1	Bitmap	D7 Data direction 0 - Host-to-device 1 - Device-to-host D6:5 Type 0 = Standard 1 = Class 2 = Vendor 3 = Reserved D4:0 Recipient 0 = Device 1 = Interface 2 = Endpoint 3 = Other 4-31 = Reserved
1	bRequest	1	Value	Specific Request
2	wValue	2	Value	Use varies according to request
4	wIndex	2	Index or Offset	Use varies according to request
6	wLength	2	Count	Number of bytes to transfer if there is a data stage

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000b 00000001b 00000010b	CLEAR_FEATURE [4]	Feature Selector	Zero Interface Endpoint	Zero	None
10000000b	GET_CONFIGURATION [4]	Zero	Zero	One	Configuration Value
10000000b	GET_DESCRIPTOR [4]	Descriptor Type (H) and Descriptor Index (L)	Zero or Language ID	Descriptor Length	Descriptor
10000001b	GET_INTERFACE [4]	Zero	Interface	One	Alternate Interface
10000000b 10000001b 10000010b	GET_STATUS [4]	Zero	Zero Interface Endpoint	Two	Device, Interface or Endpoint Status
00000000b	SET_ADDRESS [4]	Device Address	Zero	Zero	None
00000000b	SET_CONFIGURATION [4]	Configuration Value	Zero	Zero	None
00000000b	SET_DESCRIPTOR [4]	Descriptor Type (H) and Descriptor Index (L)	Zero or Language ID	Descriptor Length	Descriptor
00000000b 00000001b 00000010b	SET_FEATURE [4]	Feature Selector	Zero Interface Endpoint	Zero	None
00000001b	SET_INTERFACE [4]	Alternate Setting	Interface	Zero	None
10000010b	SYNCH_FRAME	Zero	Endpoint	Two	Frame Number

Example Atmel-SAMBA5D36 Host Controller

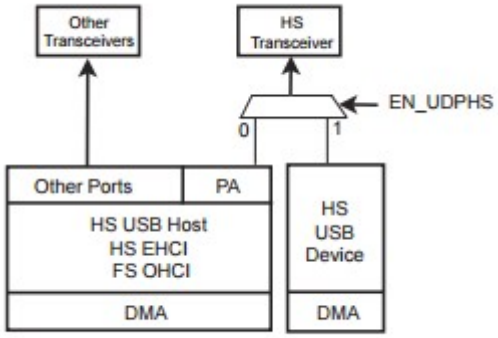
Block Diagram



HHSDPA - USB Host Port A High Speed Data +
HHSDMA - USB Host Port A High Speed Data -
HHSDPB - USB Host Port B High Speed Data +
HHSDMB - USB Host Port B High Speed Data -
HHSDPC - USB Host Port C High Speed Data +
HHSDMC - USB Host Port C High Speed Data -

Access to the USB host operational registers is achieved through the AHB bus slave interface. The Open HCI host controller and Enhanced HCI host controller initialize master DMA transfers through the AHB bus master interface as follows:

- Fetches endpoint descriptors and transfer descriptors
- Access to endpoint data from system memory
- Access to the HC communication area
- Write status and retire transfer descriptor



Implementation flow

USB Selection

TBD

Results

TBD

Challenges faced

- Non OS Implementation
- No existing USB host driver for SAMA5d3x controllers
- Interfacing custom CDC device(LPC controller)

Solutions/Observations/Findings

TBD

References

[1] http://www.geoffknagge.com/uni/elec101/essay.shtml#Ch7

[2] http://www.usbmadesimple.co.uk/ums_3.htm

[3] http://datakey.com/resources/whitepapers/embedded-systems-design-guide-for-removable-usb-flash-drives/success

[4] http://www.usbmadesimple.co.uk/ums_4.htm

[5] http://www.usb.org/developers/defined_class

[6] http://www.atmel.com/Images/Atmel-11121-32-bit-Cortex-A5-Microcontroller-SAMA5D3_Datasheet.pdf