**MPS Universal Serial Bus Lab Exercise**

**Universal Serial Bus**

Student's name & ID (1): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Partner's name & ID (2): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Your Section number & TA's name \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Notes:**

You must work on this assignment with your partner. Hand in a printed copy of your software listings for the team. Hand in a neat copy of your circuit schematics for the team.

These will be returned to you so that they may be used for reference.

------------------------------- do not write below this line -----------------------------

|  |  |  |  |
| --- | --- | --- | --- |
|  | POINTS (1) (2) | | TA init. |
| Grade for performance verification (50% max.) |
| USB Part 1 (20% max.) |  | |  |
| USB Part 2 (30% max.) |  | |  |
| Grade for answers to TA's questions (20% max.) |  |  |  |
| Enhancement (5% max.) |  | |  |
| Grade for documentation and appearance (25% max.) |  | |  |
|  |  |  | TOTAL |

Grader's signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Universal Serial Bus**

**GOAL**

By doing this lab assignment, you will learn:

1. Universal Serial Bus (USB)

**PREPARATION**

• References: *769 Reference Manual (Register Map).pdf*

Skim Ch. 41 (USB)

*STM32 USB Host Library.pdf*

All

*STM32 FATFs Library.pdf*

All

*769 Description of HAL Drivers.pdf*

Ch. 30 (USB HCD)

(Reference only; “stm32f7xx\_ll\_usb.h/.c” with “stm32f7xx\_hal\_hcd.h/.c” contain more useful info in their comments)

*USB3320 Datasheet.pdf*

Skim

**UNIVERSAL SERIAL BUS (USB)**

**1. Introduction to USB**

USB devices are everywhere. In fact, chances are the device you’re reading this document on has some form of USB connector or two in it. Even seemingly oddball interfaces like iPhone/iPad Lightning ports, many proprietary camera connectors, Microsoft Surface dock connectors, and some car interconnections are actually just glorified USB ports, as they incorporate USB to some capacity but have different connector shapes and [sometimes] extra pins for other functions. There are also toasters[[1]](#footnote-1) and refrigerators[[2]](#footnote-2) that come with USB ports these days, so the interface certainly isn’t going anywhere anytime soon (especially with the popularity of USB Type-C connectors on laptops and tablets).[[3]](#footnote-3)

There are a couple different flavors of USB devices, namely hosts, peripherals, and hubs. As the names imply, hosts are devices like laptops, desktops, and tablets that have the USB female connectors, peripherals are endpoint devices like memory sticks, keyboards, printers, STM32F769I-DISCOs, etc., and hubs are those things you plug into a host that gives you more ports.

USB ports also have capabilities that depend upon their specification. For example, USB 1.0 tops at 1.5 Mb/s (192 kB/s). The first major revision, USB 1.1, supports devices up to 12Mb/s (1.5 MB/s). USB 2.0, by contrast, goes up to 480Mb/s (60MB/s), and USB 3.0 goes up to 5 Gb/s (625MB/s). More recently, there have been USB 3.1 Gen 1 and USB 3.1 Gen 2 specifications, where USB 3.1 gen 1 is just the new name for USB 3.0 while USB 3.1 gen 2 goes up to 10Gb/s (1.25 GB/s).

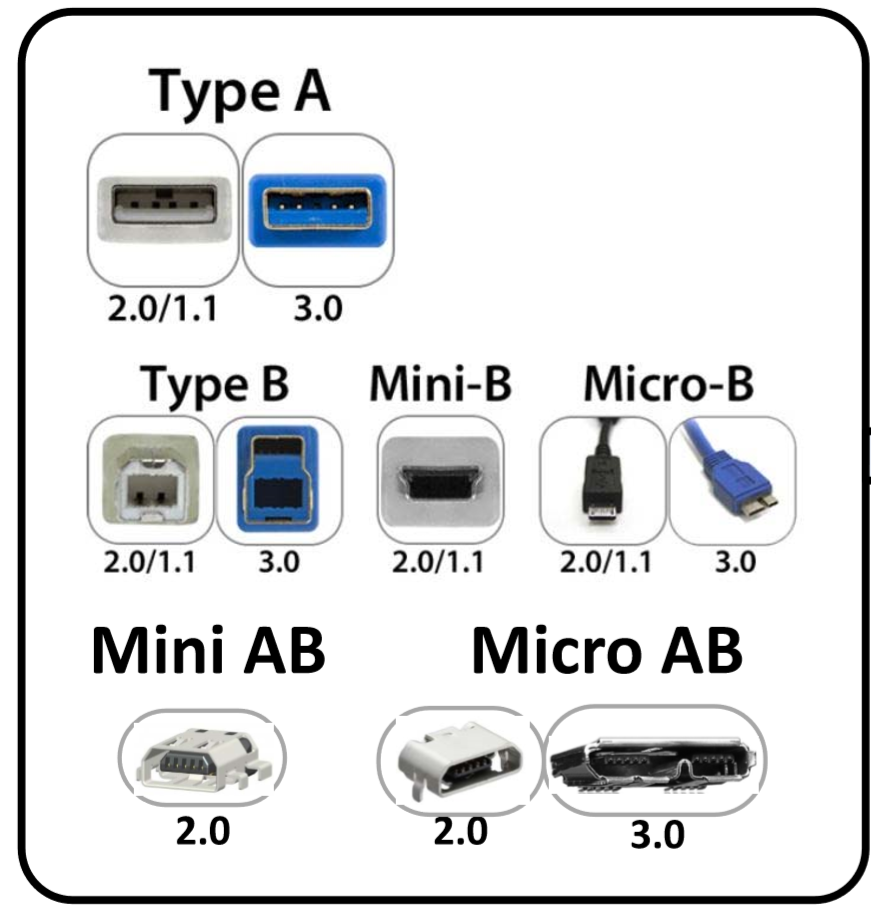
Additionally, because USB uses differential signaling, USB 1.0, 1.1, and 2.0 use only 4 pins (VDD, D+, D-, GND). USB 3.0 bumped this up to 9 pins, adding RX-, RX+, TX-, TX+, and another GND to allow for full-duplex serial data transmission.

The following table organizes the various elements of the major USB specifications:

**Table 1: USB Specification Properties**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **USB Specification** | **Speed** | **Typical Port Color**  **(non-Type-C)** | **Common Port Types**  **(no combo ports)** | **Spec Name** | **Max Current** |
| 1.0 | 1.5 Mb/s  (192 kB/s) | Black | Type-A  Type-B | Low Speed (LS) | 500 mA |
| 1.1 | 12 Mb/s  (1.5 MB/s) | Black | Type-A  Type-B | Full Speed (FS) | 500mA |
| 2.0 | 480 Mb/s  (60MB/s) | Black  Red (eSATA combo) | Type-A  Type-B  Mini-A  Mini-B  Micro-A (uncommon)  Micro-B | High Speed (HS)  or  Hi-Speed | 500mA |
| 3.0/3.1 Gen 1  or  3.1 Rev 1 | 5 Gb/s  (625 MB/s) | Blue | Type-A  Type-B  Micro-B  Type-C | SuperSpeed (SS) | 900mA |
| 3.1 Gen 2  or  3.1 Rev 2 | 10 Gb/s  (1.25 GB/s) | Blue | Type-A  Type-C | SuperSpeed+ (SS+)  or  SuperSpeed 10 Gb/s (SS10) | 900mA |
| Charging | N/A | Yellow  Orange  Red | Type-A  Type-C | Charging  or  Power Delivery (USB-PD) | 1.5A |

In addition, the USB specification requires that all newer specs be backwards compatible, so an old USB 1.1 keyboard will still work in a USB 3.1 port just fine (albeit only at USB 1.1 speeds). Similarly, hosts, hubs, and peripherals can be mixed ‘n’ matched of any revision, but the bus will only operate as fast as the slowest link in the chain. Also, while we won’t officially be dealing with hubs in this lab, an interesting thing to note is that USB spec certifies daisy-chains of up to 127 devices through hubs.[[4]](#footnote-4)



**Figure 1: Common USB Connector Types from (Random) Converting USB 2 to Type-C.pdf**

*A note about USB port types:*

Generally, type-A plugs and connectors are meant for the host end of a USB cable, and type-B for the peripheral end. Type-C is reversible, and both hosts and devices can have them. There exist combination plugs like the Micro-AB port on the DISCO board (CN15) that signify the device can act as either host or peripheral, though this is not very common. It is more common to see dual-role Micro-B (looks like CN16) connectors instead—a good example is most smartphones. Have you ever tried to hook up a USB drive or USB ethernet adapter to your phone?

Technically, dual-role Micro-USB devices are referred to as USB On-the-Go (USB OTG), which is an actual specification defining devices that can act as USB hosts but are predominantly meant to be USB peripherals. This includes smartphones, TI-84s, the DISCO boards, and other devices, and looking for the USB OTG label on a device is an easy way to tell if the device supports USB host capability.

**2. Configuring a USB OTG Host**

From *769 Reference Manual (Register Map).pdf*, Ch. 41.16:

**41.16.1 Core initialization**

The application must perform the core initialization sequence. If the cable is connected during power-up, the current mode of operation bit in the OTG\_GINTSTS (CMOD bit in OTG\_GINTSTS) reflects the mode. The OTG\_FS/OTG\_HS controller enters host mode when an “A” plug is connected or device mode when a “B” plug is connected.

This section explains the initialization of the OTG\_FS/OTG\_HS controller after power-on. The application must follow the initialization sequence irrespective of host or device mode operation. All core global registers are initialized according to the core’s configuration:

1. Program the following fields in the OTG\_GAHBCFG register:

– Global interrupt mask bit GINTMSK = 1

– Rx FIFO non-empty (RXFLVL bit in OTG\_GINTSTS)

– Periodic Tx FIFO empty level

1. Program the following fields in the OTG\_GUSBCFG register:

– HNP capable bit

– SRP capable bit

– OTG\_FS/OTG\_HS timeout calibration field

– USB turnaround time field

1. The software must unmask the following bits in the OTG\_GINTMSK register:

OTG interrupt mask

Mode mismatch interrupt mask

1. The software can read the CMOD bit in OTG\_GINTSTS to determine whether the OTG\_FS/OTG\_HS controller is operating in host or device mode.

**41.16.2 Host initialization**

To initialize the core as host, the application must perform the following steps:

1. Program the HPRTINT in the OTG\_GINTMSK register to unmask
2. Program the OTG\_HCFG register to select full-speed host
3. Program the PPWR bit in OTG\_HPRT to 1. This drives VBUS on the USB.
4. Wait for the PCDET interrupt in OTG\_HPRT0. This indicates that a device is connecting to the port.
5. Program the PRST bit in OTG\_HPRT to 1. This starts the reset process.
6. Wait at least 10 ms for the reset process to complete.
7. Program the PRST bit in OTG\_HPRT to 0.
8. Wait for the PENCHNG interrupt in OTG\_HPRT.
9. Read the PSPD bit in OTG\_HPRT to get the enumerated speed.
10. Program the HFIR register with a value corresponding to the selected PHY clock 1
11. Program the FSLSPCS field in the OTG\_HCFG register following the speed of the device detected in step 9. If FSLSPCS has been changed a port reset must be performed.
12. Program the OTG\_GRXFSIZ register to select the size of the receive FIFO.
13. Program the OTG\_HNPTXFSIZ register to select the size and the start address of the Non-periodic transmit FIFO for non-periodic transactions.
14. Program the OTG\_HPTXFSIZ register to select the size and start address of the periodic transmit FIFO for periodic transactions.

To communicate with devices, the system software must initialize and enable at least one channel.

See Ch. 41.16.5 in *769 Reference Manual (Register Map).pdf* for channel initialization and channel halting, in addition to USB behavior during operation.

**3. USB on the STM32F769I-DISCO**

The STM32F769NI has USB 2.0 host capabilities on CN15— the square-ish Micro USB port next to the STLINK connector[[5]](#footnote-5)—so it can interface with devices such as USB mice and memory sticks conforming to USB 1.0/1.1/2.0 specifications. It can also act like a peripheral on that port, enabling multiple devices to connect to each other… Or letting you turn one into a totally overkill 64MB USB stick. You will only be required to use the host capabilities for this lab, however if you additionally want to try out peripheral mode, go for it!

Note that the STM32 DISCO board has two different kinds of USB ports on it: A Micro-AB port (CN15) and a Micro-B port (CN16). You are using CN16 for the virtual COM port (it’s a peripheral-only port connected through an onboard debugger chip), though only CN15 is directly connected to the STM32F769NI. That means that we can’t really use CN16 for much more than what we are currently using it for (debugging, uploading programs, and UART), but we have full access to CN15.

Unfortunately, it appears that STMicroelectronics did not adequately document their USB host controller driver (HCD) module’s HAL implementation, so you’ll have to reference stm32f7xx\_ll\_usb.h/.c and stm32f7xx\_hal\_hcd.h/.c for configuration information about the HAL HCD.

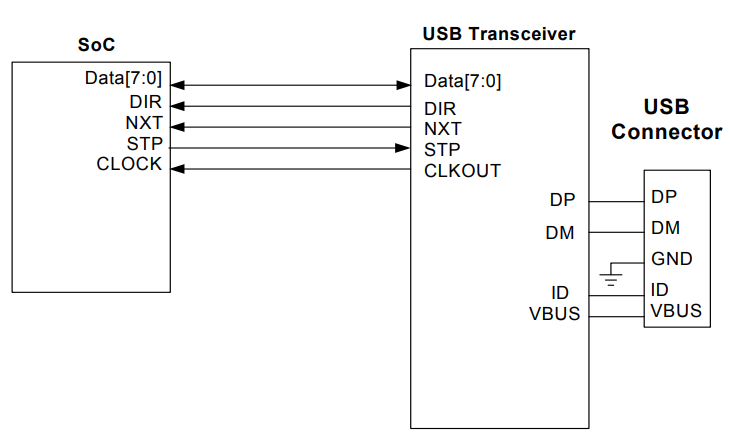
When setting up USB OTG High-Speed, you will need to configure the following GPIO to use the appropriate alternate function GPIO\_AF*x*\_OTG\_HS (see Table 13 in *769 Datasheet (Alternate Functions etc).pdf* to fill in the *x* based on Table 3):

**Table 3: USB HS Port Pins**

|  |  |
| --- | --- |
| **Port** | **Function** |
| PA3 | USB\_OTG\_HS\_ULPI\_D0 |
| PA5 | USB\_OTG\_HS\_ULPI\_CK |
| PB0 | USB\_OTG\_HS\_ULPI\_D1 |
| PB1 | USB\_OTG\_HS\_ULPI\_D2 |
| PB5 | USB\_OTG\_HS\_ULPI\_D7 |
| PB10 | USB\_OTG\_HS\_ULPI\_D3 |
| PB11 | USB\_OTG\_HS\_ULPI\_D4 |
| PB12 | USB\_OTG\_HS\_ULPI\_D5 |
| PB13 | USB\_OTG\_HS\_ULPI\_D6 |
| PC0 | USB\_OTG\_HS\_ULPI\_STP |
| PH4 | USB\_OTG\_HS\_ULPI\_NXT |
| PI11 | USB\_OTG\_HS\_ULPI\_DIR |

You should use mode AF\_PP (USB’s all-digital), no pull, and high speed.

The reason we have 12 pins instead of 2 for high-speed is because we are configuring the STM32F7’s connection to an external USB PHY, which is a physical chip (USB3320C-EZK) that takes data over a 4- or 8-bit data bus with 4 control signals, converts it to 2-pin differential signaling, and sends it to a transceiver on the target device. Essentially, the PHY is a lot like a multiplexer, but it is this whole system that is under the umbrella of the term “USB.” Technically this 12-pin link is called ULPI, which means UTMI+ Low Pin Count,[[6]](#footnote-6) and is shown in Figure 2:



**Figure 2: 12-pin ULPI interface Between SoC and Transceiver (PHY) from SMSC USB ULPI Design Guide.pdf**

Something worth noting is that we are NOT using the native OTG FS module (STMicroelectronics integrated a USB 1.x PHY directly into the 769 design) for low-speed transfers. This is simply because the port pins it uses conflicts with our virtual UART.

**USB PROGRAMMING TASKS**

In the following programming exercises, use the USB OTG host functions. You should also operate the host in DMA mode, bearing in mind the constraint of 32-bit aligned addresses. (*Hint: For Part 2, there is a certain FATFs setting that helps with this*.)

In the lab folder, there is a USB host library that you are encouraged to use, and a sample usb\_conf.c and usb\_conf.h have been provided (you need to make your own usb\_conf or incorporate the configuration into your main program, but you can use these as a guide as well as the template in the host library). If you use any of the files in the host library folder, you’ll need to put them in the appropriate src and inc directories. The only files you are not permitted to use directly are the sample usb\_conf.c/.h files—your program simply won’t work if you try to use them.

*Helpful hint from stm32f7xx\_hal\_hcd.h:*

typedef USB\_OTG\_GlobalTypeDef HCD\_TypeDef;

typedef USB\_OTG\_CfgTypeDef HCD\_InitTypeDef;

typedef USB\_OTG\_HCTypeDef HCD\_HCTypeDef ;

typedef USB\_OTG\_URBStateTypeDef HCD\_URBStateTypeDef ;

typedef USB\_OTG\_HCStateTypeDef HCD\_HCStateTypeDef ;

**PART I – Mouse**

Using a MicroUSB to USB OTG adapter, detect a USB mouse and display its connection status on the terminal. This program should be written so that it uses the USB OTG HS module, even though mice/keyboards are USB 1.x devices. It should also detect when a device has been plugged or unplugged, and behave accordingly (i.e. don’t poll a mouse when it’s not there!). Be sure to use the USB PHY’s internal DMA mode for maximum throughput.

Note: A mouse is considered a “Human Interface Device” (HID). So is a keyboard and a touchscreen.

You may be tempted to use the USBH\_HID\_GetMouseInfo() function ST provides in the host library, but be aware that in order for it to work you need to have USBH\_Process() called on a regular basis. This function, according to ST, “increments the state machine,” which basically means it is a pseudo-background process that needs to be manually triggered (so it’s not *truly* a background process). It does certain tasks like poll the mouse for interrupts and calls USBH\_HID\_EventCallback(), which isn’t actually a real interrupt callback due to being called by the pseudo-background process.[[7]](#footnote-7)

To get raw data from your mouse, you can use the generic USBH\_HID\_GetReport() function:

USBH\_StatusTypeDef USBH\_HID\_GetReport (USBH\_HandleTypeDef \*phost, // Your USB Handle

uint8\_t reportType, // Type 1 in, 2 out, 3 feature, 4 to ff reserved

uint8\_t reportId, // Id, generally 0 for one device on the bus

uint8\_t\* reportBuff, // A receive buffer for incoming data

uint8\_t reportLen) // Length of incoming data

*Possible Enhancements:*

-Mouse wheel support (if you used the USB HID library, you need to add it to the library)

-Move the terminal cursor based on mouse movement, and respond to various clicks in different ways

-Keyboard support

**PART 2 – USB Mass Storage**

Using a MicroUSB to USB OTG adapter, read a FAT/FAT32 USB 2.0 (HS) drive and display its contents on the serial terminal. This program should add on to your part 1 program, and be able to automatically differentiate between HID and MSC (Mass Storage Class) devices. It should also be able to differentiate between USB 1.0/1.1/2.0 devices and change speed accordingly. Furthermore, plug/unplug detection is required, as is using DMA mode (recall USB DMA differs from the main system DMA!).

The FATFs filesystem library has been provided (as well as its supporting documentation), and you should use this to work with properly FAT or FAT32 formatted USB drives. Remember that you’ll need to put the files you need form the library into the proper src/inc folders.

*Possible Enhancements:*

-Basic file I/O like copy/paste files from one folder on a USB drive to another folder on the same drive

-A small text editor that can read and write text files via the serial terminal

*For Parts 3 and 4, the primary reference document is* STM32 USB Device Library.pdf*.*

**PART 3 – UART CDC USB Peripheral {OPTIONAL}[[8]](#footnote-8)**

Put the DISCO board into peripheral mode and configure it to have a second virtual COM port so that plugging into CN15 can also work to send serial data. You can use the provided USB device library for this, and, as with the host samples, you should not use the provided sample peripheral usb\_conf files directly in your program.

Also, a USB virtual COM port is technically a “Communications Device Class” (CDC) device.

Note: You should not use DMA for this, as explained in Ch. 6.5.7 of *STM32 USB Device Library.pdf*.

Note: There are a lot of typos in Table 8 on page 24, e.g. usbh→usbd, and usbd\_cdc is not an Audio Speaker, it’s a virtual RS232 port. Table 12 is better.

*Possible Enhancements:*

-Receive characters

**PART 4 – SD Reader USB Peripheral {OPTIONAL}**

Put the DISCO board into peripheral mode and configure it such that it is seen by another device as an SD card reader. The other device should be a machine with a Micro-USB cable plugged into CN15.

Unlike Part 3, you should use DMA for this, especially if you want to read/write from/to a MicroSD card.

Note that MicroSD card readers are considered Mass Storage devices.

*Possible Enhancements:*

-Well, there is a touchscreen on the DISCO board…

(If you want to use the screen be sure to clock the system at 200MHz max due to SDRAM constraints.)

**FAQ:**

Q: *It compiles, but nothing’s happening!*

A: Did you remember your IRQ Handlers?

Q: *Why is my mouse packet delayed by one?*

A: *Hint*: If you’re using the USB host library, examine the fifo\_init() function.

Q: *I keep getting hard faults!*

A: You need to link drivers with libraries properly. Like how using DMA required \_\_HAL\_LINKDMA(), using mass storage drivers with libraries also requires linking. Read *STM32 FATFs Library.pdf*.

1. <http://www.vexal.us/Projects/USBToaster/USBToaster.html> [↑](#footnote-ref-1)
2. <https://www.cnet.com/news/kitchenaids-new-fridge-is-more-than-a-pretty-face/> [↑](#footnote-ref-2)
3. In addition to USB, some type-C connectors also carry Thunderbolt 3, which is an external, hot-pluggable PCI-Express & DisplayPort hybrid interface. It has data transfer rates up to 40 Gb/s (5 GB/s) while also driving 2x 4K external displays, all over one cable. Neat! [↑](#footnote-ref-3)
4. See it in action here: <http://plugable.com/products/usb3c-hub97xxx/> [↑](#footnote-ref-4)
5. You may have accidentally plugged into this port and wondered why it wasn’t working when setting up for previous labs. [↑](#footnote-ref-5)
6. UTMI+ itself an acronym for USB Transmit Macrocell Interface +. [↑](#footnote-ref-6)
7. Oy vey. [↑](#footnote-ref-7)
8. This is actually a really cool exercise to try, and the only reason this isn’t required is because ST’s documentation, as well as their sample code, for USB peripheral device mode is *utterly atrocious.* If you thought the host documentation needed improvement, well, you might want to put this one off until after the course is over. [↑](#footnote-ref-8)