Lab 9 – USB – Universal Serial Bus

Goals and limitations

A full understanding of USB is beyond the scope of a single six-hour treaty. The USB 3.2 Specification is a 518 page document and includes many supplementary documents to fully describe the complete standard.
Utilize some of USB features in the chipKIT-core.
Gain some insight into the complexities of USB.

Where does USB come from?

USB is a standard that is maintained by the "USB Implementers Form, Inc." or USB-IF. USB-IF is a non-profit organization founded by the group of companies that developed the Universal Serial Bus specification. The USB-IF was formed to provide a support organization and forum for the advancement and adoption of Universal Serial Bus technology. The Forum facilitates the development of high-quality compatible USB peripherals (devices) and promotes the benefits of USB and the quality of products that have passed compliance testing. Some of the many activities that the USB-IF supports include:

- USB Compliance Workshops
- USB Compliance Test Development
- www.usb.org Web site
- USB pavilions at Computex, IDF, and other events
- Marketing programs and collateral materials, such as retail newsletters, retail salespeople training, store end-caps, etc.
- USB Developer Conferences

At the time of this writing the USB-IF, Inc. Board of Directors is composed of the following companies and their designated representative Directors:

- Apple - Dave Conroy
- HP Inc. - Alan Berkema
- Intel Corporation - Brad Saunders
- Microsoft Corporation - Toby Nixon
- Renesas Electronics - Philip Leung
- STMicroelectronics - Joel Huloux
- Texas Instruments - Anwar Sadat

USB an Evolving Standard

Development began in 1994 by Compaq, DEC, IBM, Intel, Microsoft, NEC, and Nortel with the goal to make it fundamentally easier to connect external devices to PCs by replacing the multitude of connectors at the back of PCs, addressing the usability issues of existing interfaces, and simplifying software configuration of all devices connected to USB, as well as permitting greater data rates for external devices.

First integrated circuits supporting USB were produced by Intel in 1995.

Microsoft Windows 95, OSR 2.1 provided OEM support for the devices.

USB 1.1 was the first widely used version of USB. The 12 Mbit/s data rate was intended for higher-speed devices such as disk drives, and the lower 1.5 Mbit/s rate for low data rate devices such as joysticks.

Apple Inc.'s iMac was the first mainstream product with USB and the iMac's success popularized USB itself.

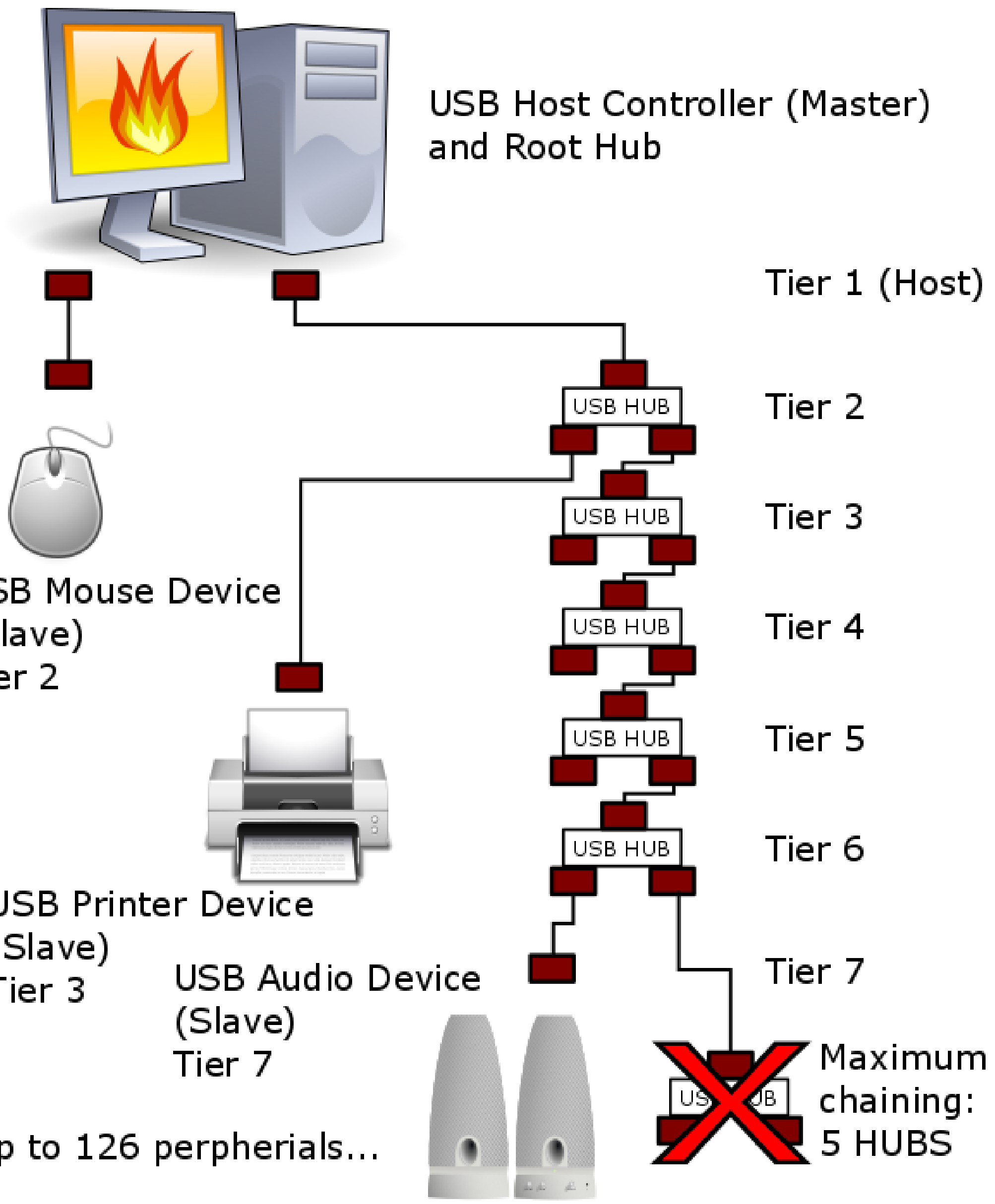
Following Apple's design decision to remove all legacy ports from the iMac, many PC manufacturers began building legacy-free PCs, which led to the broader PC market using USB as a standard.

Hewlett-Packard, Intel, Lucent Technologies (now Nokia), NEC, and Philips jointly led the initiative to develop a higher data transfer rate, with the resulting specification achieving 480 Mbit/s, 40 times as fast as the original USB 1.1 specification.

As of 2008, approximately 6 billion USB ports and interfaces were in the global marketplace, and about 2 billion were being sold each year.

Release name	Release date	Maximum transfer rate	Note
USB 0.8	Dec-94		Prerelease
USB 0.9	Apr-95		Prerelease
USB 0.99	Aug-95		Prerelease
USB 1.0-RC	Nov-95		Release Candidate
USB 1.0	Jan-96	Full Speed (12 Mbit/s)	
USB 1.1	Aug-98	Full Speed (12 Mbit/s)	
USB 2.0	Apr-00	High Speed (480 Mbit/s)	
USB 3.0	Nov-08	SuperSpeed (5 Gbit/s)	Also referred to as USB 3.1 Gen 1 and USB 3.2 Gen 1x1
USB 3.1	Jul-13	SuperSpeed+ (10 Gbit/s)	Also referred to as USB 3.1 Gen 2 and USB 3.2 Gen 2x1
USB 3.2	Sep-17	SuperSpeed+ (20 Gbit/s)	Includes new USB 3.2 Gen 1x2 and USB 3.2 Gen 2x2 multi-link modes

Elements of a USB System



Star Topology, maximum of 127 devices per host port, counting the host (126 devices)
HUB Chaining, maximum depth of 5 HUBS. (7 Tier's where the Host is Tier 1)
USB is a "Single Master + Multiple Slaves" polled bus
All Transactions are initiated by the Host
Devices cannot talk to one another
Shared USB bandwidth
Devices respond to the Host
Each Device is polled by the Host

Host – Master, considered upstream
Manages and controls the bus

Initiates all transactions

Automatically detects all device insertions and extractions

Enumerates all devices connected and matches them with drivers

Typical requirements for a host
Host is usually a PC or a Smartphone

USB Host Controller

USB drivers for identifying and enumerating USB devices

PIC32's can be USB Host (but we are not discussing this here)

Embedded Host
Embedded Host connects to a known number of USB Peripheral Devices – USB Drivers fixed in firmware

- Advantage: Smaller, less complex embedded firmware

Example: Remote Temperature Data Logger

- Download data to USB Flash Drive
- Act as Host when connected to Flash drive but …
- Not connected directly to PC Host

Device – Slave, considered downstream
Responds to Host, cannot initiate transactions

Requires drivers to be recognized by the Host

Hardware/Firmware to respond to Host

Microchip's PIC® MCUs are used in USB peripheral devices

USB Hubs
Connects Additional devices to USB Bus

Max 5 deep Hubs on Root Hub

Provides power to devices connected

Most Hubs use ASIC (Application Specific Integrated Circuit) controllers

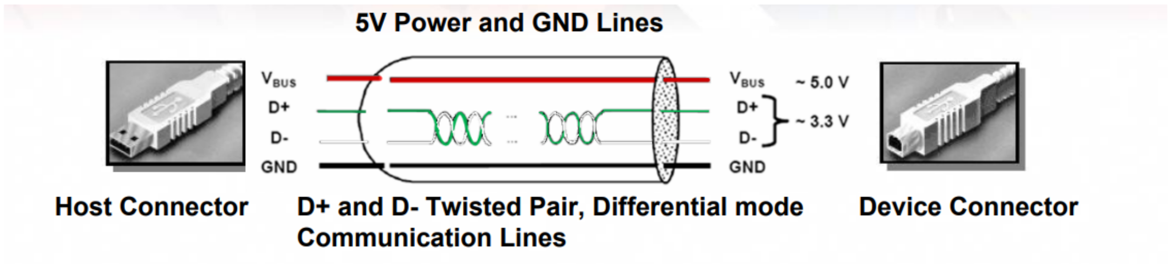
USB On-The-Go (OTG) and Dual Role
USB On-The-Go (OTG) allows application to operate as Host or Device

- Smartphone (Device) connected to PC (Host)
- Smartphone (Host) connected to Thumb drive (Device)
- Smartphone in Dual Role Mode

Smartphone connected to PDA, host and device roles can switch

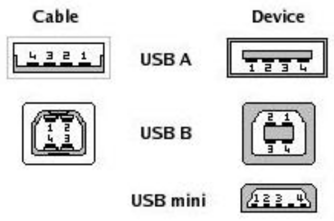
- Smartphone in OTG mode

USB Physical Interface



USB 1.1 / 2.0 Signals

Pin	Signal	Color	Description
1	VCC	RED	+5V, power lines from which USB devices can draw power
2	D-	WHITE	Data -, Twisted pair, differential mode communication lines
3	D+	GREEN	Data +, for excellent noise immunity of data
4	GND	BLACK	Ground, power lines from which USB devices can draw power



USB 3.0 Signals

Pin	Color	Signal name		Description
		A connector	B connector	
Shell	N/A	Shield		Metal housing
1	Red	VBUS		Power
2	White	D−		USB 2.0 differential pair
3	Green	D+		
4	Black	GND		Ground for power return
5	Blue	StdA_SSRX−	StdB_SSTX−	SuperSpeed transmitter differential pair
6	Yellow	StdA_SSRX+	StdB_SSTX+	
7	N/A	GND_DRAIN		Ground for signal return
8	Purple	StdA_SSTX−	StdB_SSRX−	SuperSpeed receiver differential pair
9	Orange	StdA_SSTX+	StdB_SSRX+	
The USB 3.0 Powered-B connector has two additional pins for power and ground.				
10	N/A		DPWR	Power provided to device (Powered-B only)
11			DGND	Ground for DPWR return (Powered-B only)

USB Device Power

Self-Powered

On board battery or external power source such as line voltage. Does not (should not) draw power form +5V DC on USB connection.

Bus Powered

Explicitly draws power from USB connection.

100mA at 5VDC (500mW)

Maximum power that can be drawn from USB connection without requesting more. Current draw of 100mA is consider one-unit load.

500mA at 5VDC (2.5W)

Must be requested from host and granted before allowed to draw this much power.

>500mA for USB 3.0 (maximum draw depending on host)

Must be requested from host and granted before allowed to draw this much power.

Power down mode

Must draw less than 2.5mA at 5VDC (12.5mW), preferred less than 1mA at 5VDC (5mW)

Speeds

Description	Abbreviation	Bandwidth (Speed)	USB Version Introduced
Low Speed	LS	1.5 Mbit/s (187.5 KB/s)	1.0
Full Speed	FS	12 Mbit/s (1.5 MB/s)	1.0
High Speed	HS	480 Mbit/s (60 MB/s)	2.0
Super Speed	SS	5 Gbit/s (625 MB/s)	3.0
Super Speed+	SS+	10 Gbit/s (1.25 GB/s)	3.1
Super Speed+	SS+	20 Gbit/s (2.5 GB/s)	3.2

Connectors

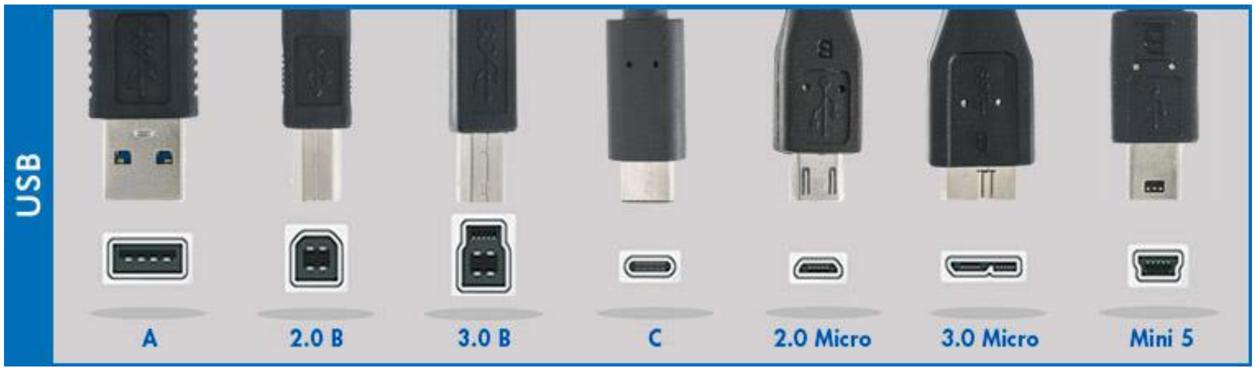
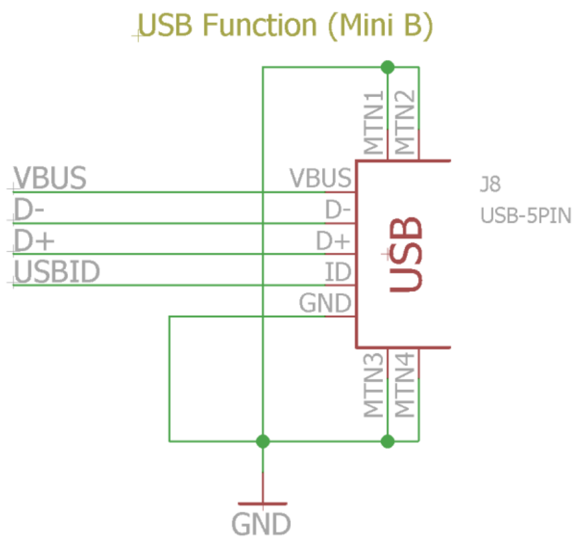


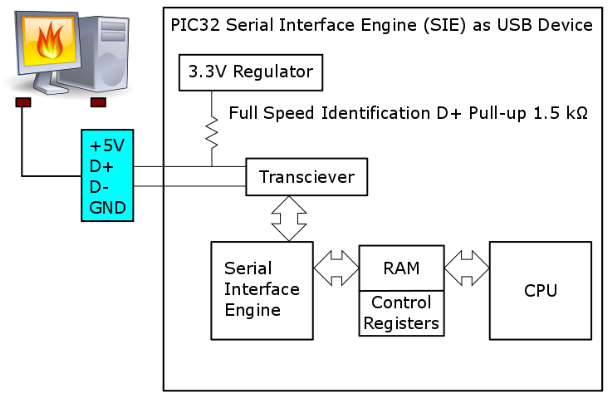
Image from cablewholesale.com (plugs on top, sockets on bottom)

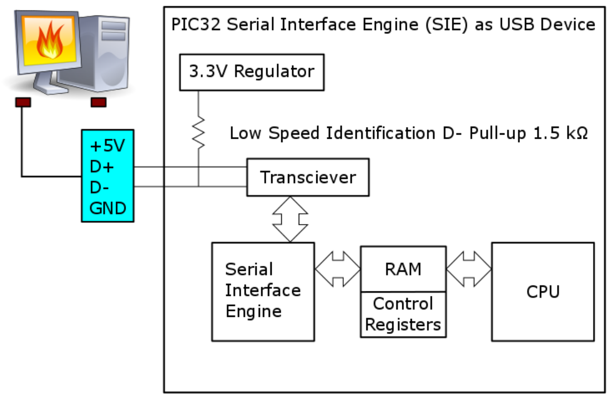
USB Connections on the Fubarino Mini / Fubarino SD



Microchip's PIC32 MCUs USB Module (Serial Interface Engine SIE)

- Support USB 2.0 Low and Full Speed
- On-chip USB/OTG (on-the-go) Transceivers, Voltage Regulator and pull-up resistors
- On chip PLL for USB clock
- USB Dual Access RAM
- Easy hardware hookup from PIC MCU to USB Port
- Serializes and deserializes USB data
- Encodes and decodes NRZI data
- Handles bit stuffing
- Checks CRC to validate data packet
- Detects bus signaling events and notifies the CPU through interrupts
- Handles USB transactions
- Handles handshaking protocal


PIC32 Serial Interface Engine (SIE) as USB Device
3.3V Regulator
Full Speed Identification D+ Pull-up 1.5 kΩ
+5V
D+
D-
GND
Transciever
Serial Interface Engine
RAM
Control Registers
CPU


PIC32 Serial Interface Engine (SIE) as USB Device
3.3V Regulator
Low Speed Identification D- Pull-up 1.5 kΩ
+5V
D+
D-
GND
Transciever
Serial Interface Engine
RAM
Control Registers
CPU

USB Peripheral Device Classes

HID – Human Interface Device
Mouse, keyboard, Joystick, Dataglove and others

MSD – Mass Storage Device
Floppy Drive, CD Drive, Thumb Drive

CDC – Communications Device Class

Ethernet Adapter

ACM – Asynchronous Communications
RS-232 / Modem replacement

Vendor Classes

Many More Classes

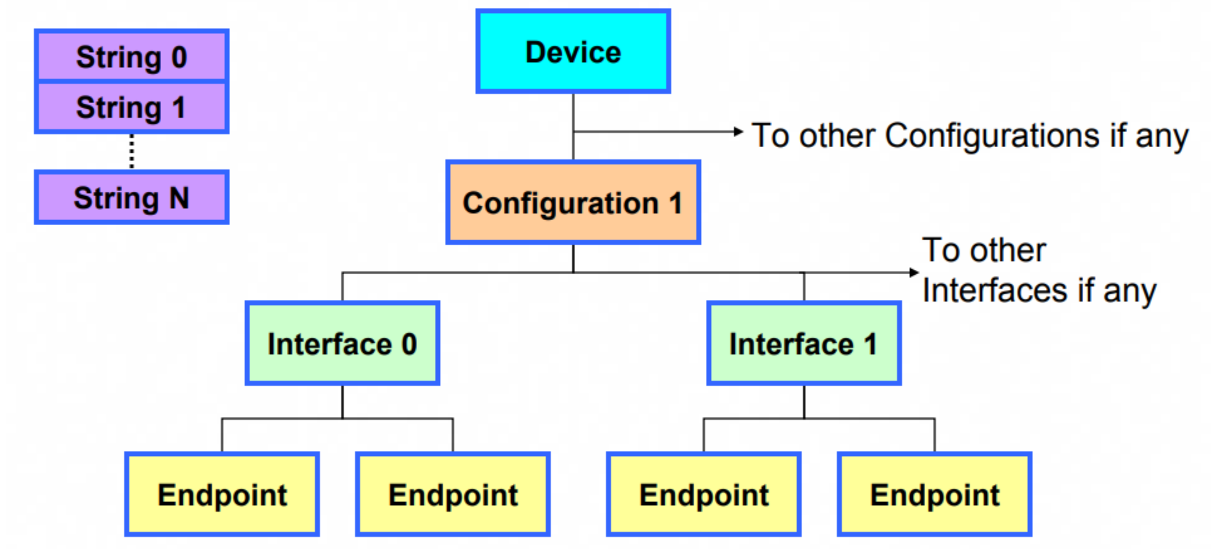
VID/PID

- VID: Vendor Identification, 16-bit number
    - Required to market your product
    - http://www.usb.org/developers/vendor
    - USD $5000 (one-time fee as of April 2018)
    - Additional annual licensing fees if you want to use the USB logos
    - Technical and legal trouble if not using an approved VID
- PID: Product Identification, 16-bit number
    - Microchip has a sub-licensing program where they will assign you a VID/PID from there VID group for use with your product (if you exceed 10,000 units you are required to get your own VID/PID from the USB-IF)
- Every product line is required to have a unique combination of VID and PID.

Protocol

Descriptors

Data sent from a device to the host to identify what its function is.



Descriptors are typically stored in non-volatile (flash) memory

Types of descriptors

Device

One per device

Product information

How many configuration descriptors

Configuration

Typically, one per device but there can be more than one.

How many interfaces

Interface

Defines the USB Classes this device has. One interface per class.

Endpoint

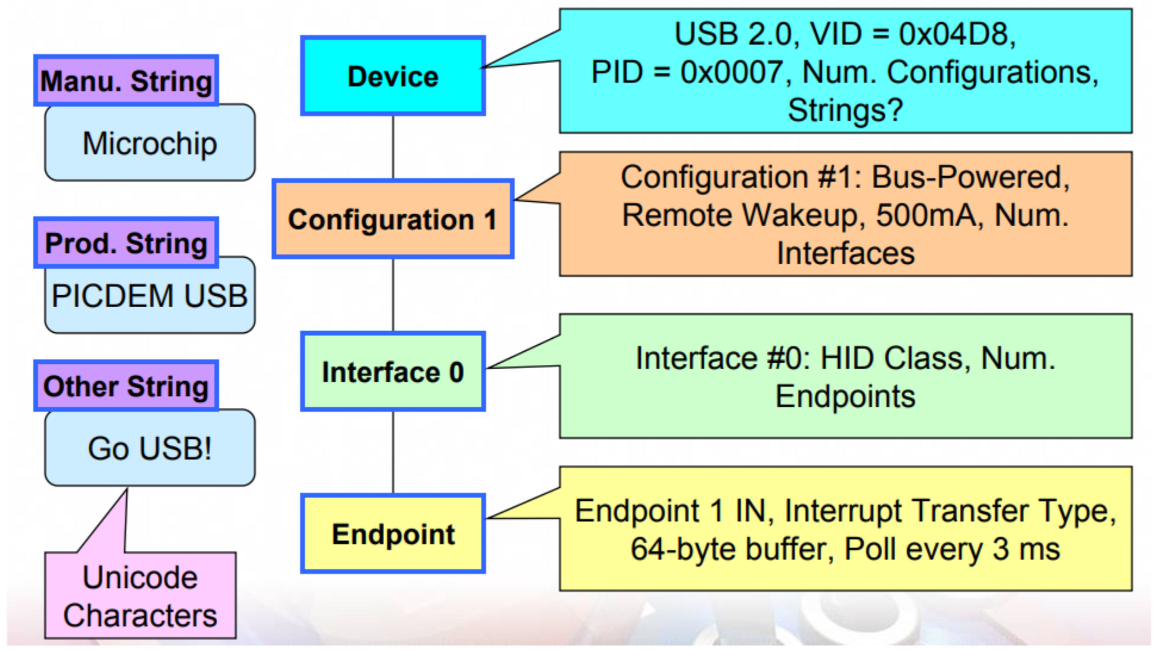
Endpoints required per class are defined in the specific USB Class specification.

String

Contains the strings referenced by other descriptors.

Arrays of two byte Unicode characters.

Descriptors Example


Manu. String
Microchip
Prod. String
PICDEM USB
Other String
Go USB!
Unicode Characters
Device
Configuration 1
Interface 0
Endpoint
USB 2.0, VID = 0x04D8, PID = 0x0007, Num. Configurations, Strings?
Configuration #1: Bus-Powered, Remote Wakeup, 500mA, Num. Interfaces
Interface #0: HID Class, Num. Endpoints
Endpoint 1 IN, Interrupt Transfer Type, 64-byte buffer, Poll every 3 ms

Transfer Types

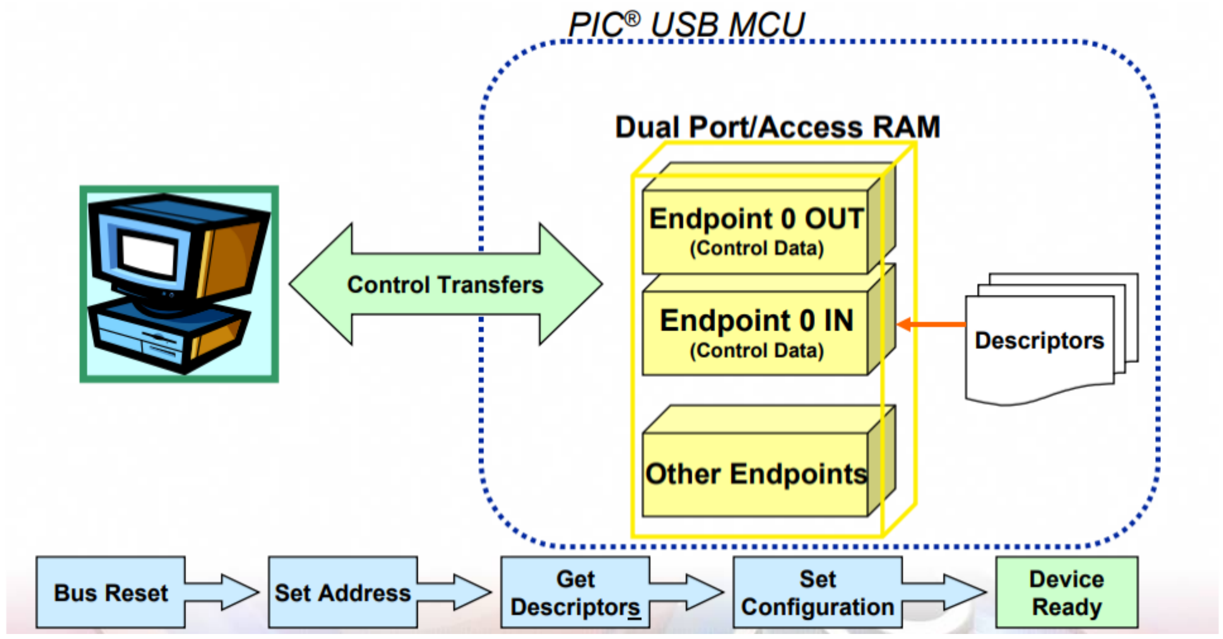
Transfer / Endpoint Type	Polling Interval	% Reserved BW/Frame for all transfers of this type	Max. # Data Bytes / Frame / Endpoint (Max# transactions per frame @ Max Ep Size)*	Data Integrity
Interrupt	Fixed, Periodic	90	64 (1 x 64)	Yes
Isochronous	Fixed, Periodic	90	1023 (1 x 1023)	No
Bulk	Variable, Uses Free Bandwidth	0	1216 (19 x 64)	Yes
Control	Variable	10	832 (13 x 64)	Yes

* Assumes transfers use maximum packet sizes allowed per Ep type

Devices classes cannot use the Control transfer type. Control is used for enumeration only.

Endpoint 0 and Enumeration

See Chapter 9 in USB 2.0 Spec for more information.



Compliance Testing

Must pass to use USB logo
It cost money probably >$1,500 (2008 price, I don't have current pricing)

Ch 9 and other USB Firmware

USB Protocol Analyzer
"USBCV"USB Command Verifier
Test device for conformance to Ch9, Hub, HID, MSD and Video Class Specifications
www.usb.org/developers/tools

Electrical Signal Quality

Power Management

For USB Compliance: Independent Test Labs

For Device 'Sanity Check': USB "Plugfest"

For USB Compliance Testing:

Must submit a compliance checklist
http://www.usb.org/developers/compliance
Download "Peripheral Checklist"
TID: Test ID
Use certified USB receptacle and cable for testing
Know the TID of your components
All USB PIC MCUs have a TID number. Get it at www.microchip.com/usb
Probably a good idea to take a look at the checklist even before starting your design!

Details we are not going to talk about

Line states

Detached

Attached

Idle

Device states / Enumeration

Attached –devices has been discovered

Powered – device is functioning

Addressed – device has give a 7-bit address

Configured – host is able to communicate with device

Suspended – device is in a minimum power draw state (mostly for bus powered devices)

Physical Interface

Voltage levels of data transfer

Half Duplex, Differential NRZI (non-return to zero inverted)

Asynchronous (with clock recovery)

Packets

8-bit SYNC sequence

0 to N 8-bit data sequence

8-bit packet identifier (PID)

2-bit end of packet identifier

Packet Types

Token

IN, OUT, SOF, SETUP

Data packets

DATA0, DATA1

Handshake packets

ACK, NACK, STALL

Special packets

ERR, PRE, PING, SPLIT

Read

IN (HOST) -> DATAx (DEVICE) -> ACK (HOST)

Write

OUT (HOST) -> DATAx (HOST) -> ACK (DEVICE)

Endpoints

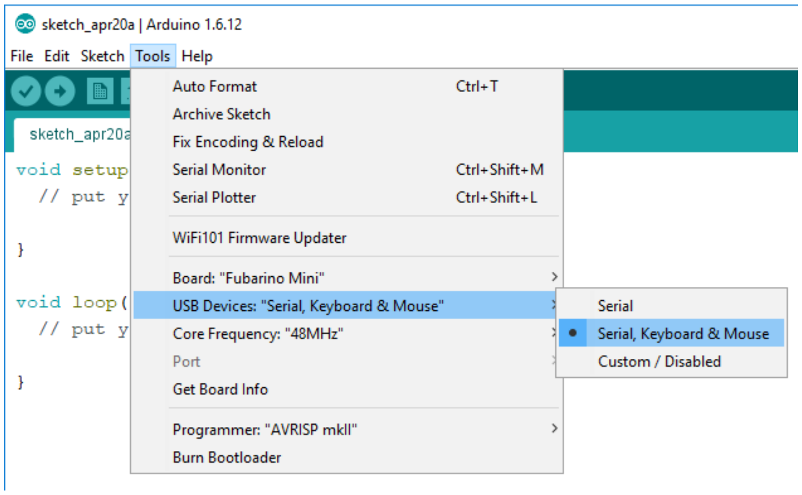
Sinks or sources of data and operate in a single direction

Host to device (16 OUT endpoints), Device to host (16 IN endpoints)

The zero endpoint controls bus management in both directions.

USB Examples Program

In order to run and test USB Example code we need to turn on the USB libraries for the chipKIT boards. The is accomplished from the tools menu by selecting the USB Devices pop out then selecting “Serial, Keyboard & Mouse” option as shown below.



Keyboard

Keyboard API

The USB Keyboard API is documented on the Arduion.cc web site here:
https://www.arduino.cc/reference/en/language/functions/usb/keyboard/

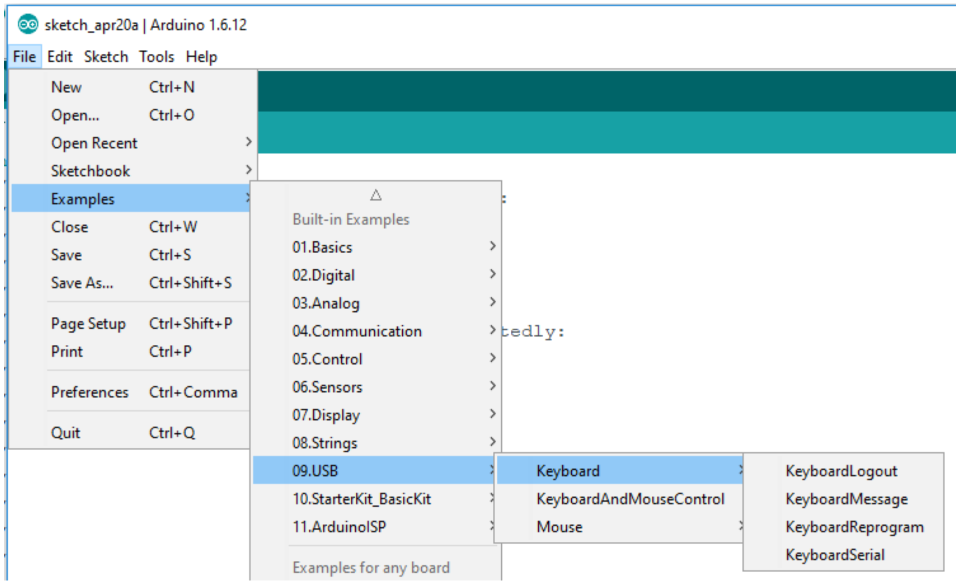
The available functions are listed below.

Keyboard.begin()
Keyboard.end()

Keyboard.press ()
Keyboard.release()
Keyboard.releaseAll()

Keyboard.print()
Keyboard.println()
Keyboard.write()

Examples



KeyboardLogout
Send the keyboard sequence to log you out of your computer.

KeyboardMessage
Send a text string to the USB host computer as a USB keyboard.

KeyboardReprogram
Reprogram an Arduino using IDE keystrokes. This sketch will not work (completely) with a Fubarino since the board need to be manually reset into to the bootloader and the COM port changed in order for a new program to be received.

KeyboardSerial
Create both a USB CDC-ACM Serial port and a USB Keyboard on the same device. Characters received over the USB CDC-ACM port are then retransmitted as keystrokes to the PC.

Keyboard Modifiers

Keyboard modifiers can be used with the Keyboard.press() to send special keystrokes to the USB Host computer by using the predefined macro that is part of the USB Keyboard API. The table below lists all the keyboard modifiers that are available for use in the sketch.

KEY	HEXADECIMAL VALUE	DECIMAL VALUE
KEY_LEFT_CTRL	0x80	128
KEY_LEFT_SHIFT	0x81	129
KEY_LEFT_ALT	0x82	130
KEY_LEFT_GUI	0x83	131
KEY_RIGHT_CTRL	0x84	132
KEY_RIGHT_SHIFT	0x85	133
KEY_RIGHT_ALT	0x86	134
KEY_RIGHT_GUI	0x87	135
KEY_UP_ARROW	0xDA	218
KEY_DOWN_ARROW	0xD9	217
KEY_LEFT_ARROW	0xD8	216
KEY_RIGHT_ARROW	0xD7	215
KEY_BACKSPACE	0xB2	178
KEY_TAB	0xB3	179
KEY_RETURN	0xB0	176
KEY_ESC	0xB1	177
KEY_INSERT	0xD1	209
KEY_DELETE	0xD4	212
KEY_PAGE_UP	0xD3	211
KEY_PAGE_DOWN	0xD6	214
KEY_HOME	0xD2	210
KEY_END	0xD5	213
KEY_CAPS_LOCK	0xC1	193
KEY_F1	0xC2	194
KEY_F2	0xC3	195
KEY_F3	0xC4	196
KEY_F4	0xC5	197
KEY_F5	0xC6	198
KEY_F6	0xC7	199
KEY_F7	0xC8	200
KEY_F8	0xC9	201
KEY_F9	0xCA	202
KEY_F10	0xCB	203
KEY_F11	0xCC	204
KEY_F12	0xCD	205

Mouse

Mouse API

The USB Mouse API is documented on the Arduion.cc web site here:
https://www.arduino.cc/reference/en/language/functions/usb/mouse/

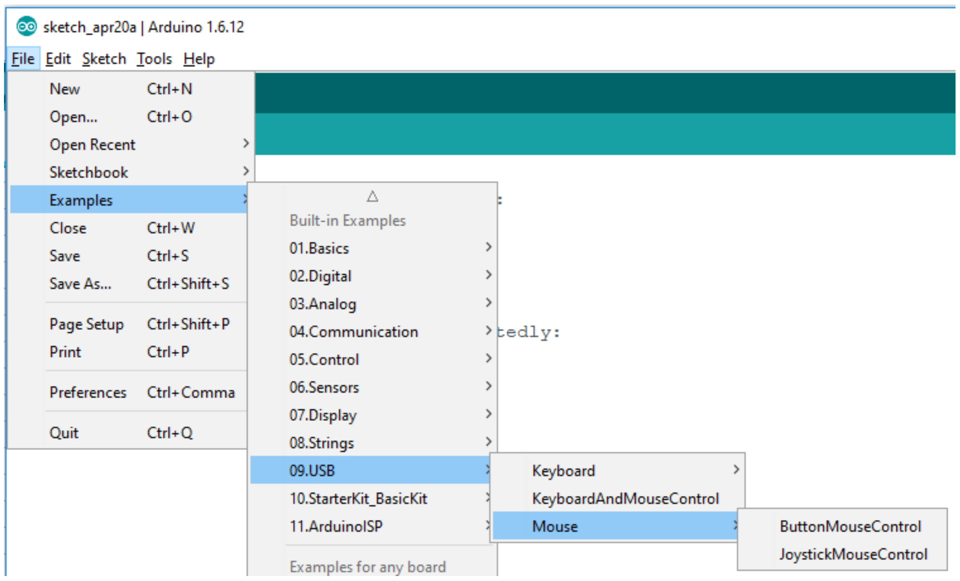
The available functions are listed below.
Mouse.begin()
Mouse.end()

Mouse.press()
Mouse.release()
Mouse.isPressed()

Mouse.click()
Mouse.move(x, y, wheel)

Parameters
button: which mouse button to press - char

MOUSE_LEFT (default)
MOUSE_RIGHT
MOUSE_MIDDLE



ButtonMouseControl
Make mouse moves with buttons presses on the control board.

JoystickMouseControl
Use analogRead() of ADC to control mouse position sent to the USB Host computer.

KeyboardAndMouseControl
Implements a combined USB Mouse and USB Keyboard in a single embedded program.

Source documentation used to for these notes
Allusb.com
en.wikipedia.org/wiki/USB (2018-04-22)
Usb.org
USB 2.0 Specification from USB.org
USB 3.0 Specification from USB.org
USB 3.1 Specification from USB.org
USB 3.2 Specification from USB.org
Microchip Masters 2008 USB Class Material

Part 1 – USB -> Keyboard -> KeyboardMessage

IMPORTANT: To use any of the USB features of the chipKIT-core other than serial communications custom descriptors need to be activated from the Tools->USB Devices: menu in the Arudino IDE. This is accomplished by selecting the Serial, Keyboard & Mouse option. Once this is selected you can write programs that tell the USB Host computer that the USB device attached can act as any combination of Serial, Keyboard or Mouse.

Open the KeyboardMessage example program found in the File -> Examples -> 09.USB -> Keyboard menu. Modify the value of the buttonPin const int variable to match the on-board program button of the Fubrino that you are using. Recall that the program button on the Fubarnio Mini is pin 16 and on the Fubrino SD is pin 23. Also, the PIN_BTN1 macro can be used for either board.

Upload the program and test it by opening notepad or any other text editor you have on your system. Click on the edit area of the text editor and press the program button on the Fubarino several times to view the functionality of the program.

Analysis

The Keyboard.begin() initializes the USB Keyboard class.

The Keyboard.print("You pressed the button ") function sends the keystrokes encapsulated by the literal string to the USB host without pressing the “Enter” key on the keyboard.
The Keyboard.print(counter) function converts the ‘counter’ variable to a decimal string and send the represented keystrokes to the USB host without pressing the “Enter” key on the keyboard.
The Keyboard.println(" times.") function sends the keystrokes encapsulated by the literal string to the USB host then send the “Enter” key on the keyboard.

Check off

Customize the message that is displayed when the button is pressed.

If you are not using a MS Windows computer connect your board running this program to a MS Windows computer and open the device manager. Next open the “Keyboards”, “Mice and other pointing devices” and “Ports” hierarchy tabs and note how many items are in each tab. Next press and hold the reset button while looking at the device manager.

Be prepared to answer the following questions:
How do these tabs change when reset is held down?
Even though Serial.Begin() is not called in this program does the USB Host see a serial port?

Demonstrate the program to the instructor.

Part 2 – USB -> Keyboard -> KeyboardMessage Customization

Modify the program in part 1 so that is sends a custom message when the button is pressed. Also have the program send the same message to the USB Serial port using the Serial object.

Check off

Demonstrate the program to the instructor.

Part 3 – USB -> Keyboard -> KeyboardLogout

Open the KeyboardLogout example program found in the File -> Examples -> 09.USB -> Keyboard menu. Modify the value of the 'platform' variable to match the operating system you are using (or closest to it) to one of the three listed operating systems defined with the preprocessor defines listed above it (OSX, WINDOWS, UBUNTU).

Modify the digitalRead() in the first line of the loop() to utilize the program button on the Fubarino (Hint: PIN_BTN1).

WARNING: This program uses keyboard shortcuts to log you out of your user session in the selected platform. Make sure you have all data saved before testing this program.

Build and upload this program to the Fubarino.

Analysis

The Keyboard.press() simulate pressing, but not releasing a key on the keyboard).
The Keyboard.releaseAll() causes all currently pressed keys to be released.
The Keyboard.write() sends a single character to the USB Host.

Checkoff

Contrast what you think are the differences between Keyboard.write() and Keyboard.print().

Demonstrate the program to the instructor.

Part 4 – USB -> Keyboard -> KeyboardSerial

Try out the KeyboardSerial example program found in the File -> Examples -> 09.USB -> Keyboard menu. This program just types back on the keyboard the characters+1 received over the USB Serial port.

Build and upload and test this program to the Fubarino.

Checkoff

Demonstrate the program to the instructor.

Part 5 – USB -> Mouse

Try out the mouse program:

```
/* Mouse Button */

#include "Mouse.h"

// set pin numbers for the five buttons:
int range = 5;              // output range of X or Y movement; affects
movement speed
int responseDelay = 10;     // response delay of the mouse, in ms

void setup() {
  // initialize the buttons' inputs:
  pinMode(PIN_BTN1, INPUT);
  // initialize mouse control:
  Mouse.begin();
}

void loop() {
  // read the buttons:
  int clickState = digitalRead(PIN_BTN1);

  // if the mouse button is pressed:
  if (clickState == LOW) {
    if (!Mouse.isPressed(MOUSE_LEFT)) { // if the mouse is not pressed, press
it:
      Mouse.press(MOUSE_LEFT);
    }
  }
  // else the mouse button is not pressed:
  else {
    if (Mouse.isPressed(MOUSE_LEFT)) {  // if the mouse is pressed, release
it:
      Mouse.release(MOUSE_LEFT);
    }
  }
  delay(responseDelay);   // a delay so the mouse doesn't move too fast:

}
```

Build and upload and test this program to the Fubarino.

Checkoff

Demonstrate the program to the instructor.

Part 6 – USB -> Mouse

Modify the program above to cause mouse to move based on a value read from an ADC port of the Fubarino.

Hint: Mouse.move(xReading, yReading, wheel);

Checkoff

Demonstrate the program to the instructor.

Part 7 – Ghost in the machine

Write a program that every 10 seconds launches notepad (Windows+R to run) then "notepad+ENTER" then waits 1 second and prints out "There is a ghost in the machine"

Hint: Use the Keyboard Modifiers table to find non-printable key board characters. KEY_LEFT_GUI is the windows key on a Windows computer or the Special key on an Apple computer.

Checkoff

Demonstrate the program to the instructor.

Part 8 – Mouse to Parser

Modify your parser to add "click" and "release": command the mouse button.

Checkoff

Demonstrate the program to the instructor.

Homework / Independent Study– USB -> Keyboard -> KeyboardReprogram

The KeyboardReprogram example uses keystrokes to cause the Arduino IDE to create a new blank sketch then it enters a custom program then compiles and uploads it to an attached board. It's an interesting idea work investigating.