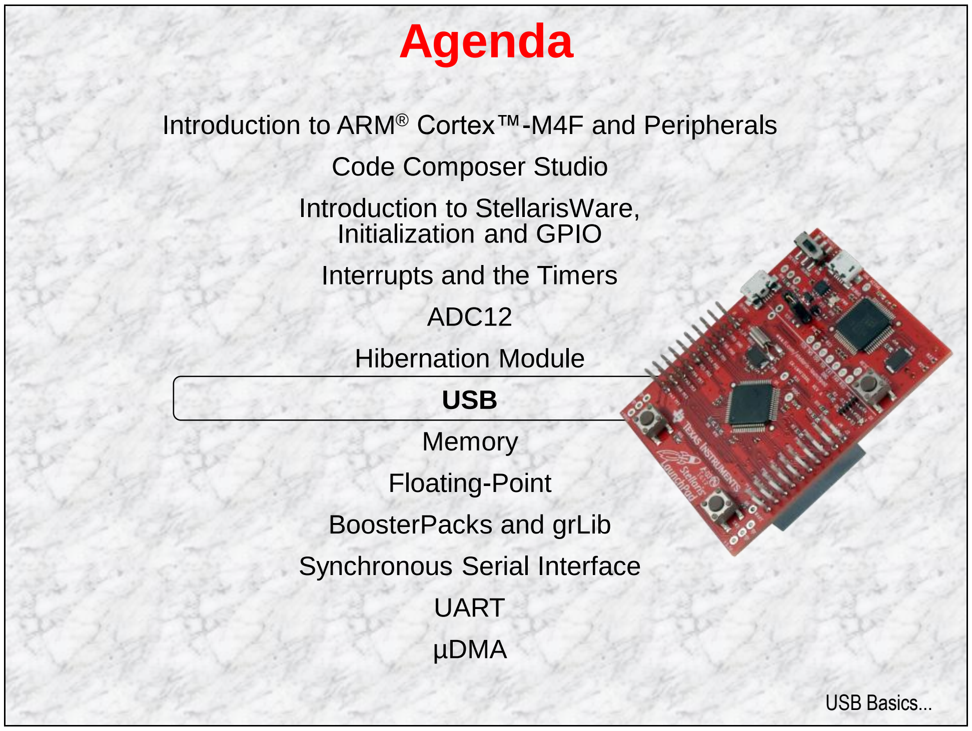
Introduction

This chapter will introduce you to the basics of USB and the implementation of a USB port on Stellaris devices. In the lab you will experiment with sending data back and forth across a bulk transfer-mode USB connection.



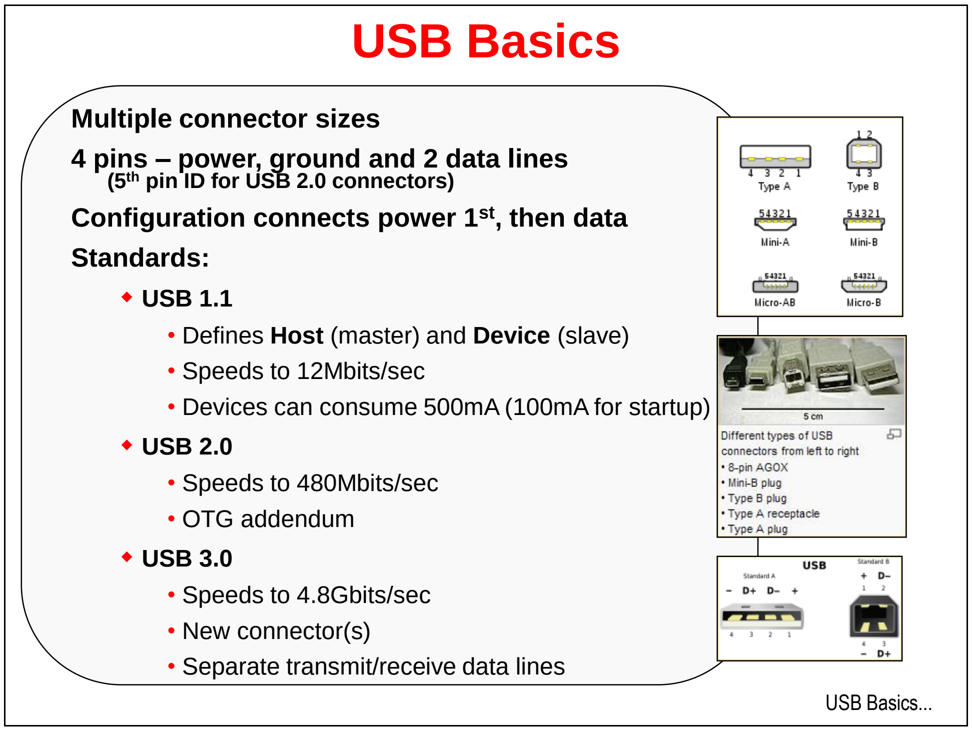
Agenda

Introduction to ARM® Cortex™-M4F and Peripherals
Code Composer Studio
Introduction to StellarisWare, Initialization and GPIO
Interrupts and the Timers
ADC12
Hibernation Module
USB
Memory
Floating-Point
BoosterPacks and grLib
Synchronous Serial Interface
UART
µDMA

USB Basics...

Chapter Topics

USB Basics



USB Basics

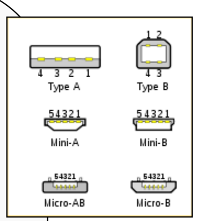
Multiple connector sizes

4 pins – power, ground and 2 data lines
(5th pin ID for USB 2.0 connectors)

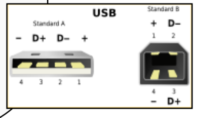
Configuration connects power 1st, then data

Standards:

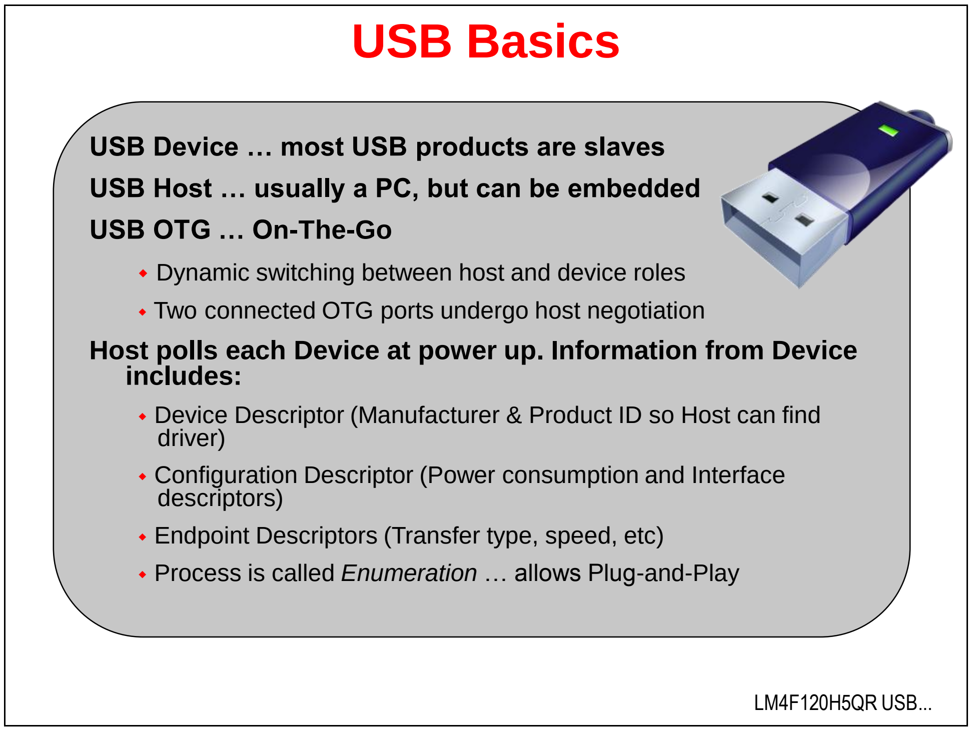
- **USB 1.1**
  - Defines **Host** (master) and **Device** (slave)
  - Speeds to 12Mbits/sec
  - Devices can consume 500mA (100mA for startup)
- **USB 2.0**
  - Speeds to 480Mbits/sec
  - OTG addendum
- **USB 3.0**
  - Speeds to 4.8Gbits/sec
  - New connector(s)
  - Separate transmit/receive data lines



Different types of USB connectors from left to right
- 8-pin AGOX
- Mini-B plug
- Type B plug
- Type A receptacle
- Type A plug



USB Basics...



USB Basics

USB Device … most USB products are slaves

USB Host … usually a PC, but can be embedded

USB OTG … On-The-Go

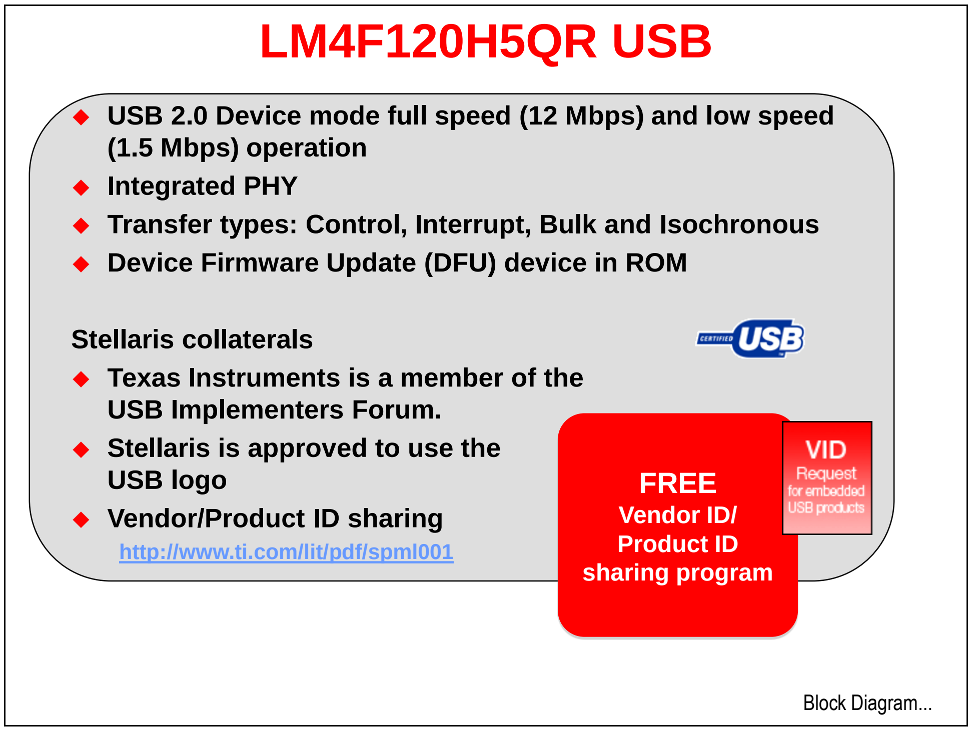
- Dynamic switching between host and device roles
- Two connected OTG ports undergo host negotiation

Host polls each Device at power up. Information from Device includes:

- Device Descriptor (Manufacturer & Product ID so Host can find driver)
- Configuration Descriptor (Power consumption and Interface descriptors)
- Endpoint Descriptors (Transfer type, speed, etc)
- Process is called *Enumeration* … allows Plug-and-Play

LM4F120H5QR USB...

LM4F120H5QR USB



LM4F120H5QR USB

- USB 2.0 Device mode full speed (12 Mbps) and low speed (1.5 Mbps) operation
- Integrated PHY
- Transfer types: Control, Interrupt, Bulk and Isochronous
- Device Firmware Update (DFU) device in ROM

Stellaris collaterals

- Texas Instruments is a member of the USB Implementers Forum.
- Stellaris is approved to use the USB logo
- Vendor/Product ID sharing
  http://www.ti.com/lit/pdf/spml001

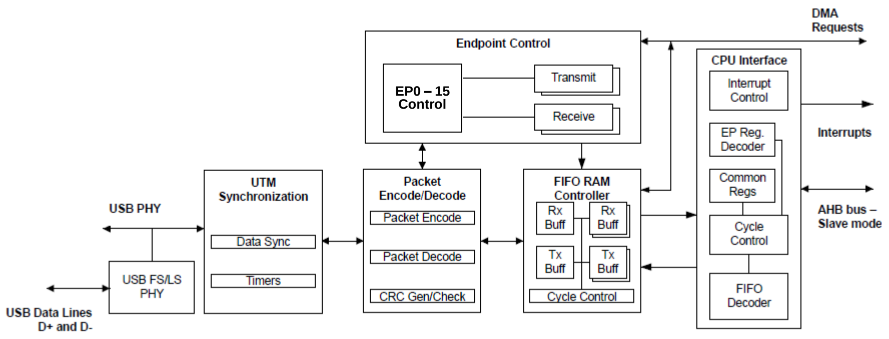
CERTIFIED USB
FREE Vendor ID/ Product ID sharing program
VID Request for embedded USB products

Block Diagram...

Sublicense application: http://www.ti.com/lit/pdf/spml001

USB Hardware and Library

USB Peripheral Block Diagram



Integrated USB Controller and PHY with up to 16 Endpoints

- 1 dedicated control IN endpoint and 1 dedicated control OUT endpoint
- Up to 7 configurable IN endpoints and 7 configurable OUT endpoints
- 4 KB dedicated endpoint memory (not part of device SRAM)
- Separate DMA channels (up to three IN Endpoints and three OUT Endpoints)
- 1 endpoint may be defined for double-buffered 1023-bytes isochronous packet size

USBLib...

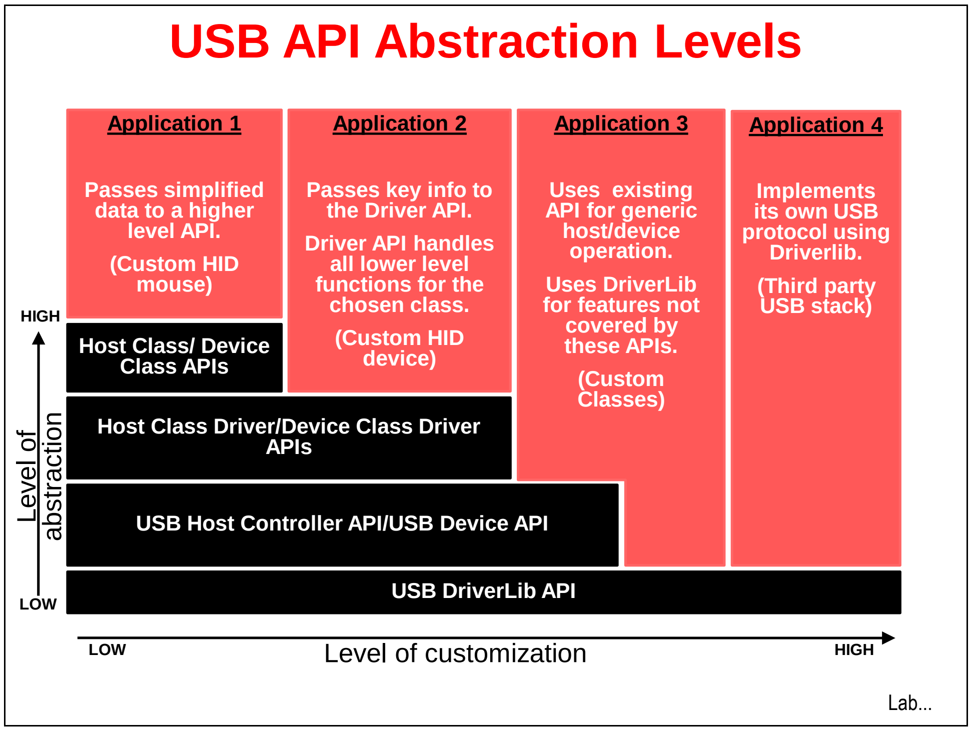
StellarisWare USBLib

- **License-free & royalty-free drivers, stack and example applications for Stellaris MCUs**
- **USBLib supports Host/Device and OTG, but the LM4F120H5QR USB port is Device only**
- **Builds on DriverLib API**
  - Adds framework for generic Host and Device functionality
  - Includes implementations of common USB classes
- **Layered structure**
- **Drivers and .inf files included where appropriate**
- **Stellaris MCUs have passed USB Device and Embedded Host compliance testing**

- Device Examples
  - HID Keyboard
  - HID Mouse
  - CDC Serial
  - Mass Storage
  - Generic Bulk
  - Audio
  - Device Firmware Upgrade
  - Oscilloscope
- Windows INF for supported devices
  - Points to base Windows drivers
  - Sets config string
  - Sets PID/VID
  - Precompiled DLL saves development time
- Device framework integrated into USBLib

CERTIFIED USB™

Abstraction Levels...



USB API Abstraction Levels

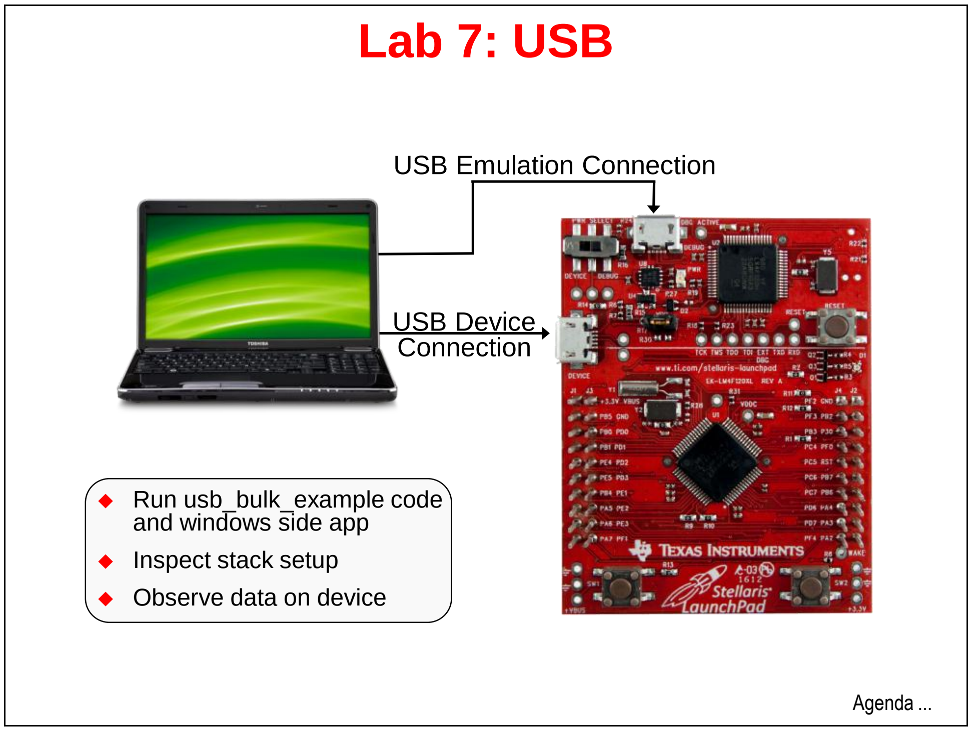
Application 1
Passes simplified data to a higher level API.
(Custom HID mouse)
Application 2
Passes key info to the Driver API.
Driver API handles all lower level functions for the chosen class.
(Custom HID device)
Application 3
Uses existing API for generic host/device operation.
Uses DriverLib for features not covered by these APIs.
(Custom Classes)
Application 4
Implements its own USB protocol using Driverlib.
(Third party USB stack)
Host Class/ Device Class APIs
Host Class Driver/Device Class Driver APIs
USB Host Controller API/USB Device API
USB DriverLib API
HIGH
Level of abstraction
LOW
LOW
Level of customization
HIGH

Lab...

Lab 7: USB

Objective

In this lab you will experiment with sending data back and forth across a bulk transfer-mode USB connection.



Lab 7: USB

USB Emulation Connection
USB Device Connection

- Run usb_bulk_example code and windows side app
- Inspect stack setup
- Observe data on device

Agenda ...

Procedure

Example Code

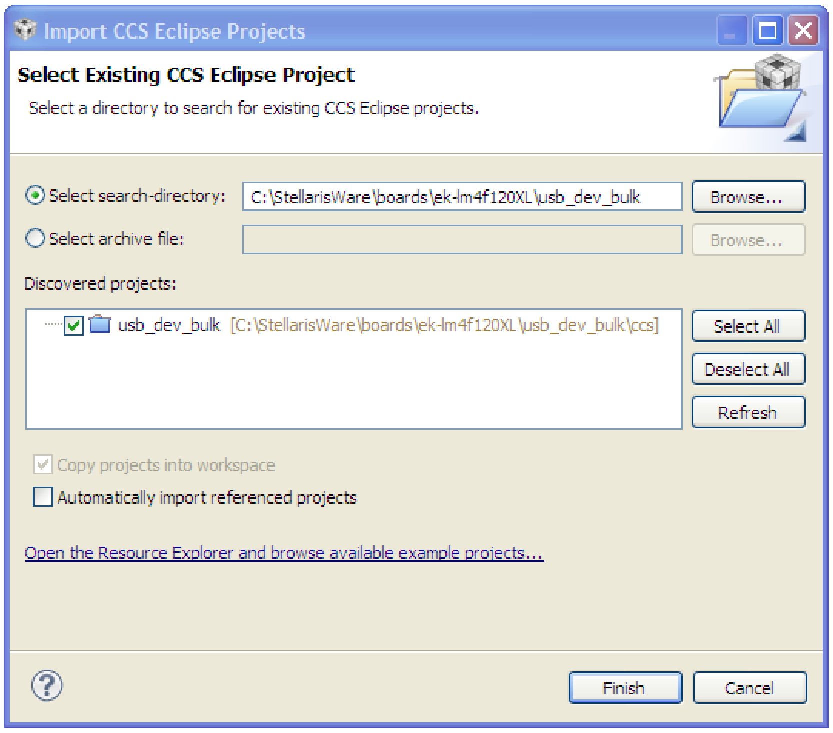
1. There are four types of transfer/endpoint types in the USB specification: Control transfers (for command and status operations), Interrupt transfers (to quickly get the attention of the host), Isochronous transfers (continuous and periodic transfers of data) and Bulk transfers (to transfer large, bursty data).

   Before we start poking around in the code, let's take the `usb_bulk_example` for a test drive. We'll be using a Windows host command line application to transfer strings over the USB connection to the LaunchPad board. The program there will change upper-case to lower-case and vice-versa, then transfer the data back to the host.

Import The Project

2. The `usb_bulk_example` project is one of the StellarisWare examples. When you import the project, note that it will be automatically copied into your workspace, preserving the original files. If you want to access your project files through Windows Explorer, the files you are working on are in your workspace, <u>not</u> StellarisWare. If you delete the project in CCS, the imported project will still be in your workspace unless you tell the dialog to delete the files from the disk.

   Click Project → Import Existing CCS Eclipse Project. Make the settings shown below and click Finish


Import CCS Eclipse Projects

Select Existing CCS Eclipse Project

Select a directory to search for existing CCS Eclipse projects.

Select search-directory: C:\StellarisWare\boards\ek-lm4f120XL\usb_dev_bulk

Select archive file:

Discovered projects:

✓ usb_dev_bulk [C:\StellarisWare\boards\ek-lm4f120XL\usb_dev_bulk\ccs]

✓ Copy projects into workspace

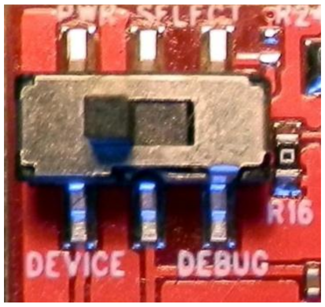
☐ Automatically import referenced projects

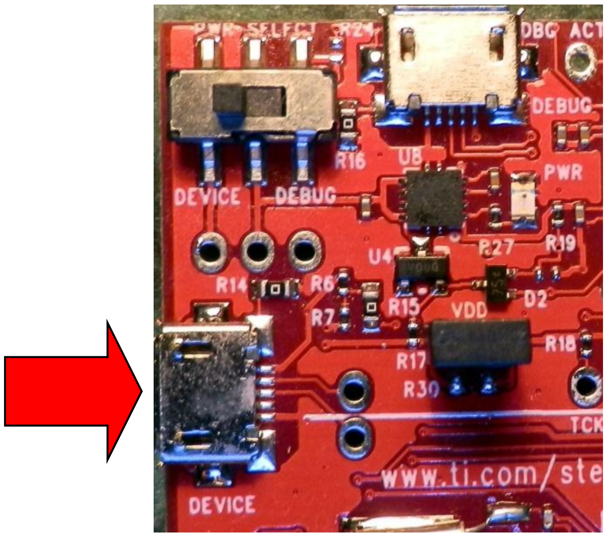
Open the Resource Explorer and browse available example projects...

Build, Download and Run The Code

3. Make sure your evaluation board's USB DEBUG port is connected to your PC. Build and download your application by clicking the Debug button on the menu bar (make sure your device is awake if you still have hibernate code programmed in the flash).

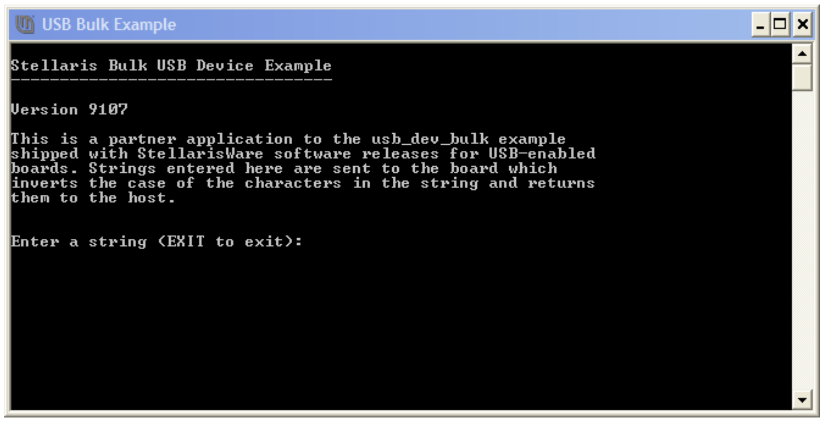
4. Click the Terminate button , and when CCS returns to the CCS Edit perspective, unplug the USB cable from the LaunchPad's DEBUG port. Move the PWR SELECT switch on the board to the DEVICE position (nearest the outside of the board). Plug your USB cable into the USB DEVICE connector on the side of the LaunchPad board. The green LED in the emulator section of the LaunchPad should be lit, verifying that the board is powered.





5. In a few moments, your computer will detect that a generic bulk device has been plugged into the USB port. Similar to the driver installation process in module 1, install the driver for this device from `C:\StellarisWare\windows_drivers`. In your Windows Device Manager, you should verify that the Stellaris Bulk Device is correctly installed.

6. Make sure that you installed the StellarisWare Windows-side USB examples from www.ti.com/sw-usb-win as shown in module one. In Windows, click Start → All Programs → Texas Instruments → Stellaris → USB Examples → USB Bulk Example. The window below should appear:



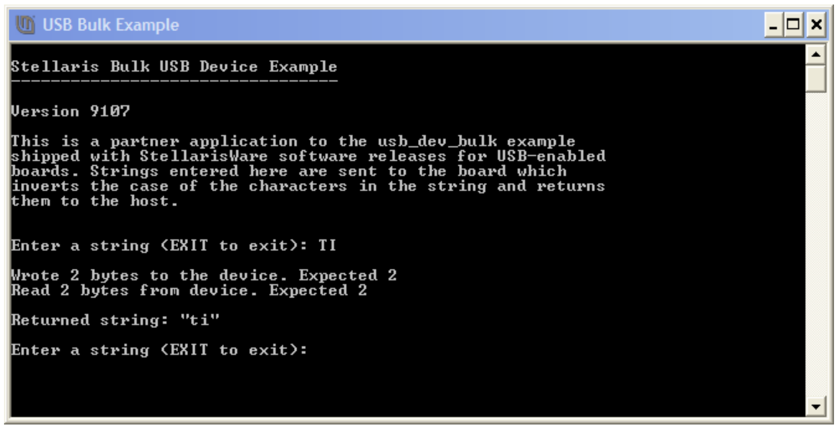
```
Stellaris Bulk USB Device Example
---------------------------------

Version 9107

This is a partner application to the usb_dev_bulk example
shipped with StellarisWare software releases for USB-enabled
boards. Strings entered here are sent to the board which
inverts the case of the characters in the string and returns
them to the host.


Enter a string (EXIT to exit):
```

7. Type something in the window and press Enter. For instance “TI” as shown below:



```
Stellaris Bulk USB Device Example
---------------------------------

Version 9107

This is a partner application to the usb_dev_bulk example
shipped with StellarisWare software releases for USB-enabled
boards. Strings entered here are sent to the board which
inverts the case of the characters in the string and returns
them to the host.

Enter a string (EXIT to exit): TI

Wrote 2 bytes to the device. Expected 2
Read 2 bytes from device. Expected 2

Returned string: "ti"

Enter a string (EXIT to exit):
```

The host application will send the two bytes representing TI over the USB port to the LaunchPad board. The code there will change uppercase to lowercase and echo the transmission. Then the host application will display the returned string. Feel free to experiment. Now that we’re assured that our data is traveling across the DEVICE USB port, we can look into the code more.

Digging a Little Deeper

8. Type EXIT to terminate the `USB Bulk Example` program on your laptop.

   Connect your other USB cable from your PC to the DEBUG USB port the on the LaunchPad and move the PWR SELECT switch on the board to the DEBUG position . The green LED in the emulator section of the LaunchPad should be lit, verifying that the board is powered. You should now have both ports connected to your PC.

9. In CCS, if `usb_dev_bulk`.c is not open, expand the `usb_dev_bulk` project in the Project Explorer pane and double-click on `usb_dev_bulk.c`.

10. The program is made up of five sections:

    `SysTickIntHandler` – an ISR that handles interrupts from the SysTick timer to keep track of “time”.

    `EchoNewDataToHost` – a routine that takes the received data from a buffer, flips the case and sends it to the USB port for transmission.

    `TxHandler` – an ISR that will report when the USB transmit process is complete.

    `RxHandler` – an ISR that handles the interaction with the incoming data, then calls the `EchoNewDataHost` routine.

    `main()` – primarily initialization, but a while loop keeps an eye on the number of bytes transferred

11. Note the `UARTprintf()` APIs sprinkled throughout the code. This technique “instru-ments” the code, allowing us to monitor its status.

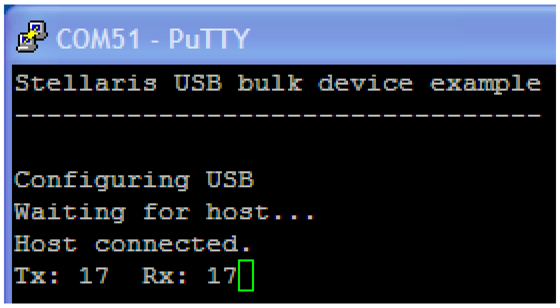
Watching the Instrumentation

12. As shown earlier in module 1, start your terminal program and connect it to the Stellaris Virtual Serial Port. Arrange the terminal window so that it takes up no more than a quarter of your screen. Position it in the upper left of your screen.

13. Resize CCS so that it takes up the lower half of your screen. Click the Debug button to build and download the code and reconnect to your LaunchPad. Run the code by clicking the Resume button.

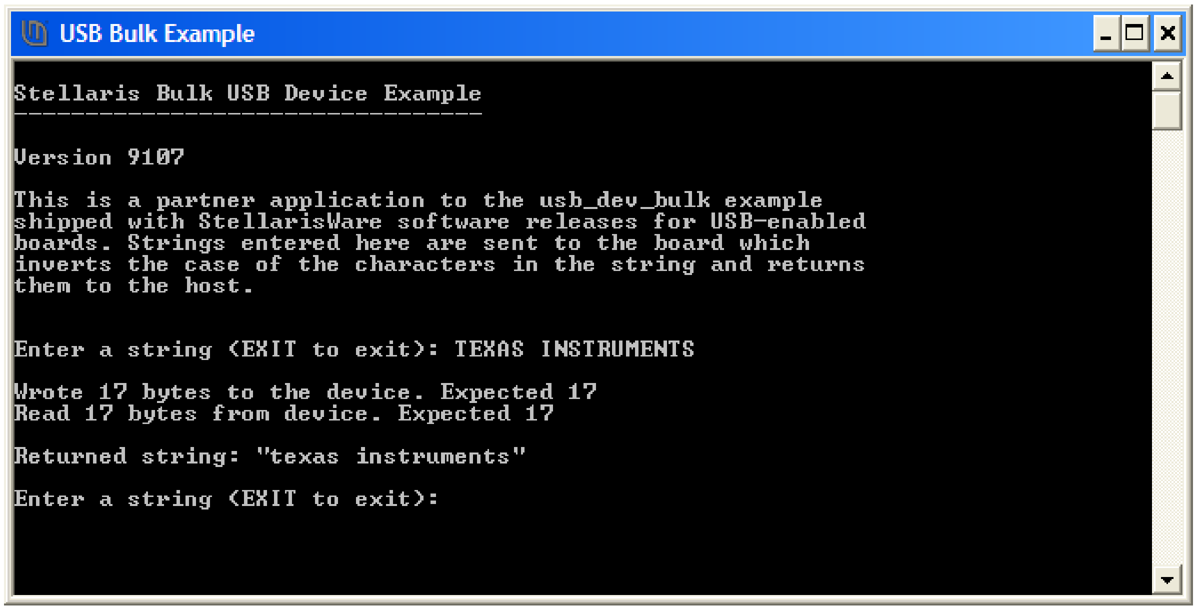
14. Start the USB Bulk Example Windows application as shown in step 6. Place the window in the upper right corner of your screen. This would all be so much easier with multiple screens, wouldn't it?

15. Note the status on your terminal display and type something, like TEXAS INSTRUMENTS into the USB Bulk Example Windows application and press Enter. Note that the terminal program will display



```
Stellaris USB bulk device example
---------------------------------

Configuring USB
Waiting for host...
Host connected.
Tx: 17  Rx: 17
```



```
Stellaris Bulk USB Device Example
---------------------------------

Version 9107

This is a partner application to the usb_dev_bulk example
shipped with StellarisWare software releases for USB-enabled
boards. Strings entered here are sent to the board which
inverts the case of the characters in the string and returns
them to the host.


Enter a string (EXIT to exit): TEXAS INSTRUMENTS

Wrote 17 bytes to the device. Expected 17
Read 17 bytes from device. Expected 17

Returned string: "texas instruments"

Enter a string (EXIT to exit):
```

16. Click the Suspend button in CCS to halt the program.

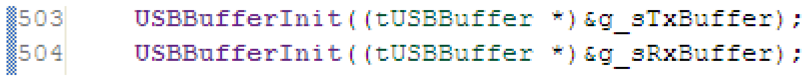
    As a summary, we're sending bulk data across the DEVICE USB connection. At the same time we are performing emulation control and sending UART serial data across the DEBUG USB connection.

    If you get things out of sync here and find that the USB Bulk Example won't run, remember that it must be started <u>after</u> the code on the LaunchPad is running.

Watch the Buffers

17. Remove all expressions (if there are any) from the Expressions pane by right-clicking inside the pane and selecting Remove All.

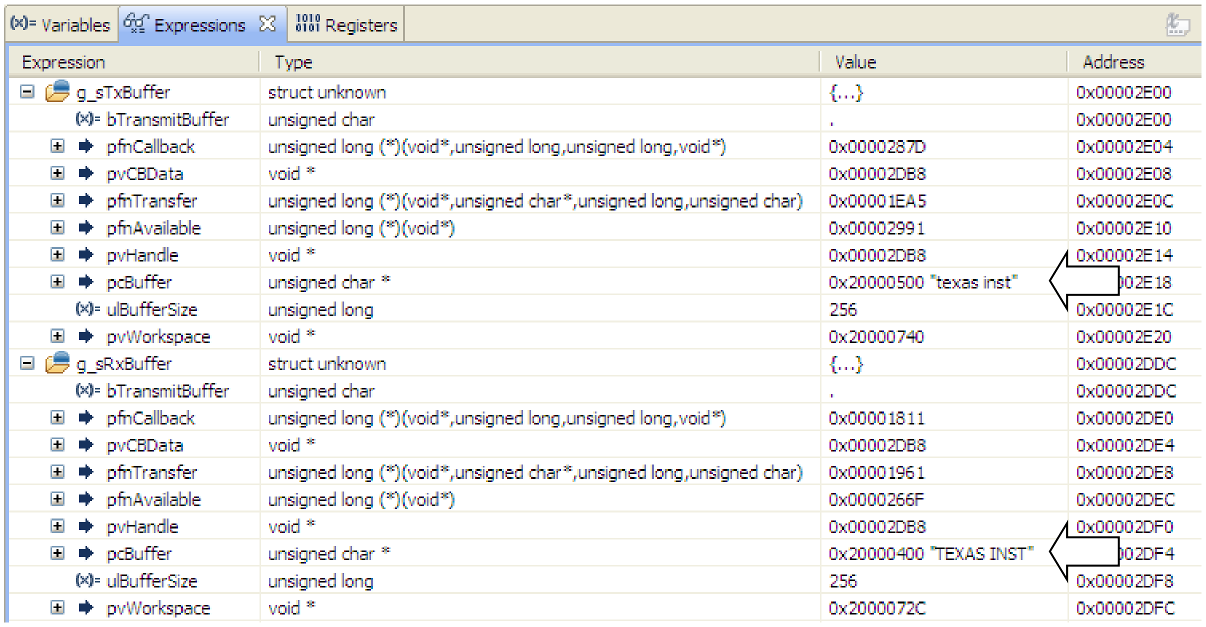
18. At about line 503 in the code, find the following:



```
503     USBBufferInit((tUSBBuffer *)&g_sTxBuffer);
504     USBBufferInit((tUSBBuffer *)&g_sRxBuffer);
```

    One at the time, highlight `g_sTxBuffer` and `g_sRxBuffer` and add them as watch expressions by right-clicking on them, selecting Add Watch Expression … and then OK (by the way, we could have watched the buffers in the Memory Browser, but this method is more elegant).

19. Expand both buffers as shown below:

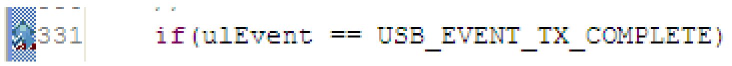


Expression	Type	Value	Address
g_sTxBuffer	struct unknown	{...}	0x00002E00
bTransmitBuffer	unsigned char	.	0x00002E00
pfnCallback	unsigned long (*)(void*,unsigned long,unsigned long,void*)	0x0000287D	0x00002E04
pvCBData	void *	0x00002DB8	0x00002E08
pfnTransfer	unsigned long (*)(void*,unsigned char*,unsigned long,unsigned char)	0x00001EA5	0x00002E0C
pfnAvailable	unsigned long (*)(void*)	0x00002991	0x00002E10
pvHandle	void *	0x00002DB8	0x00002E14
pcBuffer	unsigned char *	0x20000500 "texas inst"	02E18
ulBufferSize	unsigned long	256	0x00002E1C
pvWorkspace	void *	0x20000740	0x00002E20
g_sRxBuffer	struct unknown	{...}	0x00002DDC
bTransmitBuffer	unsigned char	.	0x00002DDC
pfnCallback	unsigned long (*)(void*,unsigned long,unsigned long,void*)	0x00001811	0x00002DE0
pvCBData	void *	0x00002DB8	0x00002DE4
pfnTransfer	unsigned long (*)(void*,unsigned char*,unsigned long,unsigned char)	0x00001961	0x00002DE8
pfnAvailable	unsigned long (*)(void*)	0x0000266F	0x00002DEC
pvHandle	void *	0x00002DB8	0x00002DF0
pcBuffer	unsigned char *	0x20000400 "TEXAS INST"	02DF4
ulBufferSize	unsigned long	256	0x00002DF8
pvWorkspace	void *	0x2000072C	0x00002DFC

The arrows above point out the memory addresses of the buffers as well as the contents. Note that the Expressions window only shows the first 10 bytes in the buffer.

The LM4F120H5QR code uses both buffers as "circular" buffers … rather than clearing out the buffer each time data is received, the code appends the new data after the previous data in the buffer. When the end of the buffer is reached, the code starts again from the beginning. You can use the Memory Browser to view the rest of the buffers, if you like.

20. Resize the code window in the Debug Perspective so you can see a few lines of code. Around line 331 in `usb_dev_bulk.c`, find the line containing `if(ulEvent` . This is the first line in the `TxHandler` ISR. At this point the buffers hold the last received and transmitted values. Double-click in the gray area to the left on the line number to set a breakpoint. Resize the windows again so you can see the entire Expressions pane.



```
331        if(ulEvent == USB_EVENT_TX_COMPLETE)
```

Right-click on the breakpoint and select Breakpoint Properties … Click on the Action property value Remain Halted and change it to Update View. Click OK.

21. Click the Core Reset button to reset the LM4F120H5QR. Make sure your buffers are expanded in the Expressions pane and click the Resume button to run the code. The previous contents of the buffers shown in the Expressions pane will be erased when the code runs for the first time.

22. Restart your USB Bulk example Windows application so that it can reconnect with our device.

23. Since the Expressions view will only display 10 characters, type something short into the USB Bulk Example window like “TI”.

24. When the code reaches the breakpoint, the Expressions pane will update with the contents of the buffer. Try typing “IS” and “AWESOME”. Notice that the “E” is the 11th character and will not be displayed in the Expressions pane.

25. When you are done, close the USB Bulk Example and Terminal program windows. Click the Terminate button in CCS to return to the CCS Edit perspective. Close the `usb_dev_bulk` project in the Project Explorer pane. Minimize Code Composer Studio.

26. Disconnect and store the USB cable connected to the DEVICE USB port.

You’re done.