TivaC Lab 7 - USB

CPE 403

**Checklist for Lab 7**

* A text/word document of the initial code with comments
* In the document, for each task submit the modified or included code (only) with highlights and justifications of the modifications. Also include the comments.
* Provide a permanent link to all main and dependent source code files only (name them as LabXX-TYY, XX-Lab# and YY-task#)Screenshots of debugging process along with pictures of actual circuit
* Video link of demonstration.

**Code for Experiment**

**Task 1:**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// usb\_dev\_bulk.c - Main routines for the generic bulk device example.

//

// Copyright (c) 2012-2015 Texas Instruments Incorporated. All rights reserved.

// Software License Agreement

//

// Texas Instruments (TI) is supplying this software for use solely and

// exclusively on TI's microcontroller products. The software is owned by

// TI and/or its suppliers, and is protected under applicable copyright

// laws. You may not combine this software with "viral" open-source

// software in order to form a larger program.

//

// THIS SOFTWARE IS PROVIDED "AS IS" AND WITH ALL FAULTS.

// NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT

// NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR

// A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER ANY

// CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL

// DAMAGES, FOR ANY REASON WHATSOEVER.

//

// This is part of revision 2.1.1.71 of the EK-TM4C123GXL Firmware Package.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**#include** <stdbool.h>

**#include** <stdint.h>

**#include** "inc/hw\_ints.h"

**#include** "inc/hw\_memmap.h"

**#include** "inc/hw\_types.h"

**#include** "driverlib/debug.h"

**#include** "driverlib/fpu.h"

**#include** "driverlib/gpio.h"

**#include** "driverlib/interrupt.h"

**#include** "driverlib/pin\_map.h"

**#include** "driverlib/sysctl.h"

**#include** "driverlib/systick.h"

**#include** "driverlib/timer.h"

**#include** "driverlib/uart.h"

**#include** "driverlib/rom.h"

**#include** "usblib/usblib.h"

**#include** "usblib/usb-ids.h"

**#include** "usblib/device/usbdevice.h"

**#include** "usblib/device/usbdbulk.h"

**#include** "utils/uartstdio.h"

**#include** "utils/ustdlib.h"

**#include** "usb\_bulk\_structs.h"

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! \addtogroup example\_list

//! <h1>USB Generic Bulk Device (usb\_dev\_bulk)</h1>

//!

//! This example provides a generic USB device offering simple bulk data

//! transfer to and from the host. The device uses a vendor-specific class ID

//! and supports a single bulk IN endpoint and a single bulk OUT endpoint.

//! Data received from the host is assumed to be ASCII text and it is

//! echoed back with the case of all alphabetic characters swapped.

//!

//! A Windows INF file for the device is provided on the installation CD and

//! in the C:/ti/TivaWare-for-C-Series/windows\_drivers directory of TivaWare C

//! series releases. This INF contains information required to install the

//! WinUSB subsystem on Windowi16XP and Vista PCs. WinUSB is a Windows

//! subsystem allowing user mode applications to access the USB device without

//! the need for a vendor-specific kernel mode driver.

//!

//! A sample Windows command-line application, usb\_bulk\_example, illustrating

//! how to connect to and communicate with the bulk device is also provided.

//! The application binary is installed as part of the ''Windows-side examples

//! for USB kits'' package (SW-USB-win) on the installation CD or via download

//! from http://www.ti.com/tivaware . Project files are included to allow

//! the examples to be built using Microsoft VisualStudio 2008. Source code

//! for this application can be found in directory

//! TivaWare-for-C-Series/tools/usb\_bulk\_example.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// The system tick rate expressed both as ticks per second and a millisecond

// period.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**#define** SYSTICKS\_PER\_SECOND 100

**#define** SYSTICK\_PERIOD\_MS (1000 / SYSTICKS\_PER\_SECOND)

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// The global system tick counter.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**volatile** uint32\_t g\_ui32SysTickCount = 0;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Variables tracking transmit and receive counts.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**volatile** uint32\_t g\_ui32TxCount = 0;

**volatile** uint32\_t g\_ui32RxCount = 0;

**#ifdef** DEBUG

uint32\_t g\_ui32UARTRxErrors = 0;

**#endif**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Debug-related definitions and declarations.

//

// Debug output is available via UART0 if DEBUG is defined during build.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**#ifdef** DEBUG

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Map all debug print calls to UARTprintf in debug builds.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**#define** DEBUG\_PRINT UARTprintf

**#else**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Compile out all debug print calls in release builds.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**#define** DEBUG\_PRINT **while**(0) ((**int** (\*)(**char** \*, ...))0)

**#endif**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Flags used to pass commands from interrupt context to the main loop.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**#define** COMMAND\_PACKET\_RECEIVED 0x00000001

**#define** COMMAND\_STATUS\_UPDATE 0x00000002

**volatile** uint32\_t g\_ui32Flags = 0;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Global flag indicating that a USB configuration has been set.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**static** **volatile** bool g\_bUSBConfigured = false;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// The error routine that is called if the driver library encounters an error.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**#ifdef** DEBUG

**void**

\_\_error\_\_(**char** \*pcFilename, uint32\_t ui32Line)

{

UARTprintf("Error at line %d of %s\n", ui32Line, pcFilename);

**while**(1)

{

}

}

**#endif**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Interrupt handler for the system tick counter.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**void**

**SysTickIntHandler**(**void**)

{

//

// Update our system tick counter.

//

g\_ui32SysTickCount++;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Receive new data and echo it back to the host.

//

// \param psDevice points to the instance data for the device whose data is to

// be processed.

// \param pui8Data points to the newly received data in the USB receive buffer.

// \param ui32NumBytes is the number of bytes of data available to be processed.

//

// This function is called whenever we receive a notification that data is

// available from the host. We read the data, byte-by-byte and swap the case

// of any alphabetical characters found then write it back out to be

// transmitted back to the host.

//

// \return Returns the number of bytes of data processed.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**static** uint32\_t

**EchoNewDataToHost**(tUSBDBulkDevice \*psDevice, uint8\_t \*pui8Data,

uint32\_t ui32NumBytes)

{

uint32\_t ui32Loop, ui32Space, ui32Count;

uint32\_t ui32ReadIndex;

uint32\_t ui32WriteIndex;

tUSBRingBufObject sTxRing;

//

// Get the current buffer information to allow us to write directly to

// the transmit buffer (we already have enough information from the

// parameters to access the receive buffer directly).

//

**USBBufferInfoGet**(&g\_sTxBuffer, &sTxRing);

//

// How much space is there in the transmit buffer?

//

ui32Space = **USBBufferSpaceAvailable**(&g\_sTxBuffer);

//

// How many characters can we process this time round?

//

ui32Loop = (ui32Space < ui32NumBytes) ? ui32Space : ui32NumBytes;

ui32Count = ui32Loop;

//

// Update our receive counter.

//

g\_ui32RxCount += ui32NumBytes;

//

// Dump a debug message.

//

DEBUG\_PRINT("Received %d bytes\n", ui32NumBytes);

//

// Set up to process the characters by directly accessing the USB buffers.

//

ui32ReadIndex = (uint32\_t)(pui8Data - g\_pui8USBRxBuffer);

ui32WriteIndex = sTxRing.ui32WriteIndex;

**while**(ui32Loop)

{

//

// Copy from the receive buffer to the transmit buffer converting

// character case on the way.

//

//

// Is this a lower case character?

//

**if**((g\_pui8USBRxBuffer[ui32ReadIndex] >= 'a') &&

(g\_pui8USBRxBuffer[ui32ReadIndex] <= 'z'))

{

//

// Convert to upper case and write to the transmit buffer.

//

g\_pui8USBTxBuffer[ui32WriteIndex] =

(g\_pui8USBRxBuffer[ui32ReadIndex] - 'a') + 'A';

}

**else**

{

//

// Is this an upper case character?

//

**if**((g\_pui8USBRxBuffer[ui32ReadIndex] >= 'A') &&

(g\_pui8USBRxBuffer[ui32ReadIndex] <= 'Z'))

{

//

// Convert to lower case and write to the transmit buffer.

//

g\_pui8USBTxBuffer[ui32WriteIndex] =

(g\_pui8USBRxBuffer[ui32ReadIndex] - 'Z') + 'z';

}

**else**

{

//

// Copy the received character to the transmit buffer.

//

g\_pui8USBTxBuffer[ui32WriteIndex] =

g\_pui8USBRxBuffer[ui32ReadIndex];

}

}

//

// Move to the next character taking care to adjust the pointer for

// the buffer wrap if necessary.

//

ui32WriteIndex++;

ui32WriteIndex = (ui32WriteIndex == BULK\_BUFFER\_SIZE) ?

0 : ui32WriteIndex;

ui32ReadIndex++;

ui32ReadIndex = (ui32ReadIndex == BULK\_BUFFER\_SIZE) ?

0 : ui32ReadIndex;

ui32Loop--;

}

//

// We've processed the data in place so now send the processed data

// back to the host.

//

**USBBufferDataWritten**(&g\_sTxBuffer, ui32Count);

DEBUG\_PRINT("Wrote %d bytes\n", ui32Count);

//

// We processed as much data as we can directly from the receive buffer so

// we need to return the number of bytes to allow the lower layer to

// update its read pointer appropriately.

//

**return**(ui32Count);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Handles bulk driver notifications related to the transmit channel (data to

// the USB host).

//

// \param pvCBData is the client-supplied callback pointer for this channel.

// \param ui32Event identifies the event we are being notified about.

// \param ui32MsgValue is an event-specific value.

// \param pvMsgData is an event-specific pointer.

//

// This function is called by the bulk driver to notify us of any events

// related to operation of the transmit data channel (the IN channel carrying

// data to the USB host).

//

// \return The return value is event-specific.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

uint32\_t

**TxHandler**(**void** \*pvCBData, uint32\_t ui32Event, uint32\_t ui32MsgValue,

**void** \*pvMsgData)

{

//

// We are not required to do anything in response to any transmit event

// in this example. All we do is update our transmit counter.

//

**if**(ui32Event == USB\_EVENT\_TX\_COMPLETE)

{

g\_ui32TxCount += ui32MsgValue;

}

//

// Dump a debug message.

//

DEBUG\_PRINT("TX complete %d\n", ui32MsgValue);

**return**(0);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Handles bulk driver notifications related to the receive channel (data from

// the USB host).

//

// \param pvCBData is the client-supplied callback pointer for this channel.

// \param ui32Event identifies the event we are being notified about.

// \param ui32MsgValue is an event-specific value.

// \param pvMsgData is an event-specific pointer.

//

// This function is called by the bulk driver to notify us of any events

// related to operation of the receive data channel (the OUT channel carrying

// data from the USB host).

//

// \return The return value is event-specific.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

uint32\_t

**RxHandler**(**void** \*pvCBData, uint32\_t ui32Event,

uint32\_t ui32MsgValue, **void** \*pvMsgData)

{

//

// Which event are we being sent?

//

**switch**(ui32Event)

{

//

// We are connected to a host and communication is now possible.

//

**case** USB\_EVENT\_CONNECTED:

{

g\_bUSBConfigured = true;

**UARTprintf**("Host connected.\n");

//

// Flush our buffers.

//

**USBBufferFlush**(&g\_sTxBuffer);

**USBBufferFlush**(&g\_sRxBuffer);

**break**;

}

//

// The host has disconnected.

//

**case** USB\_EVENT\_DISCONNECTED:

{

g\_bUSBConfigured = false;

**UARTprintf**("Host disconnected.\n");

**break**;

}

//

// A new packet has been received.

//

**case** USB\_EVENT\_RX\_AVAILABLE:

{

tUSBDBulkDevice \*psDevice;

//

// Get a pointer to our instance data from the callback data

// parameter.

//

psDevice = (tUSBDBulkDevice \*)pvCBData;

//

// Read the new packet and echo it back to the host.

//

**return**(EchoNewDataToHost(psDevice, pvMsgData, ui32MsgValue));

}

//

// Ignore SUSPEND and RESUME for now.

//

**case** USB\_EVENT\_SUSPEND:

**case** USB\_EVENT\_RESUME:

{

**break**;

}

//

// Ignore all other events and return 0.

//

**default**:

{

**break**;

}

}

**return**(0);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Configure the UART and its pins. This must be called before UARTprintf().

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**void**

**ConfigureUART**(**void**)

{

//

// Enable the GPIO Peripheral used by the UART.

//

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA);

//

// Enable UART0

//

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UART0);

//

// Configure GPIO Pins for UART mode.

//

ROM\_GPIOPinConfigure(GPIO\_PA0\_U0RX);

ROM\_GPIOPinConfigure(GPIO\_PA1\_U0TX);

ROM\_GPIOPinTypeUART(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1);

//

// Use the internal 16MHz oscillator as the UART clock source.

//

**UARTClockSourceSet**(UART0\_BASE, UART\_CLOCK\_PIOSC);

//

// Initialize the UART for console I/O.

//

**UARTStdioConfig**(0, 115200, 16000000);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// This is the main application entry function.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**int**

**main**(**void**)

{

**volatile** uint32\_t ui32Loop;

uint32\_t ui32TxCount;

uint32\_t ui32RxCount;

//

// Enable lazy stacking for interrupt handlers. This allows floating-point

// instructions to be used within interrupt handlers, but at the expense of

// extra stack usage.

//

ROM\_FPULazyStackingEnable();

//

// Set the clocking to run from the PLL at 50MHz

//

ROM\_SysCtlClockSet(SYSCTL\_SYSDIV\_4 | SYSCTL\_USE\_PLL | SYSCTL\_OSC\_MAIN |

SYSCTL\_XTAL\_16MHZ);

//

// Enable the GPIO port that is used for the on-board LED.

//

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOF);

//

// Enable the GPIO pins for the LED (PF2 & PF3).

//

ROM\_GPIOPinTypeGPIOOutput(GPIO\_PORTF\_BASE, GPIO\_PIN\_3 | GPIO\_PIN\_2);

//

// Open UART0 and show the application name on the UART.

//

ConfigureUART();

**UARTprintf**("\033[2JTiva C Series USB bulk device example\n");

**UARTprintf**("---------------------------------\n\n");

//

// Not configured initially.

//

g\_bUSBConfigured = false;

//

// Enable the GPIO peripheral used for USB, and configure the USB

// pins.

//

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOD);

ROM\_GPIOPinTypeUSBAnalog(GPIO\_PORTD\_BASE, GPIO\_PIN\_4 | GPIO\_PIN\_5);

//

// Enable the system tick.

//

ROM\_SysTickPeriodSet(ROM\_SysCtlClockGet() / SYSTICKS\_PER\_SECOND);

ROM\_SysTickIntEnable();

ROM\_SysTickEnable();

//

// Tell the user what we are up to.

//

**UARTprintf**("Configuring USB\n");

//

// Initialize the transmit and receive buffers.

//

**USBBufferInit**(&g\_sTxBuffer);

**USBBufferInit**(&g\_sRxBuffer);

//

// Set the USB stack mode to Device mode with VBUS monitoring.

//

**USBStackModeSet**(0, *eUSBModeForceDevice*, 0);

//

// Pass our device information to the USB library and place the device

// on the bus.

//

**USBDBulkInit**(0, &g\_sBulkDevice);

//

// Wait for initial configuration to complete.

//

**UARTprintf**("Waiting for host...\n");

//

// Clear our local byte counters.

//

ui32RxCount = 0;

ui32TxCount = 0;

//

// Main application loop.

//

**while**(1)

{

//

// See if any data has been transferred.

//

**if**((ui32TxCount != g\_ui32TxCount) || (ui32RxCount != g\_ui32RxCount))

{

//

// Has there been any transmit traffic since we last checked?

//

**if**(ui32TxCount != g\_ui32TxCount)

{

//

// Turn on the Green LED.

//

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_3, GPIO\_PIN\_3);

//

// Delay for a bit.

//

**for**(ui32Loop = 0; ui32Loop < 150000; ui32Loop++)

{

}

//

// Turn off the Green LED.

//

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_3, 0);

//

// Take a snapshot of the latest transmit count.

//

ui32TxCount = g\_ui32TxCount;

}

//

// Has there been any receive traffic since we last checked?

//

**if**(ui32RxCount != g\_ui32RxCount)

{

//

// Turn on the Blue LED.

//

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_2, GPIO\_PIN\_2);

//

// Delay for a bit.

//

**for**(ui32Loop = 0; ui32Loop < 150000; ui32Loop++)

{

}

//

// Turn off the Blue LED.

//

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_2, 0);

//

// Take a snapshot of the latest receive count.

//

ui32RxCount = g\_ui32RxCount;

}

//

// Update the display of bytes transferred.

//

**UARTprintf**("\rTx: %d Rx: %d", ui32TxCount, ui32RxCount);

}

}

}

**Video Link to Demo**

NONE