C:\microchip\mla\v2016_04_27_3\apps\usb\device\cdc_basic\firmware\pic18f46j50_pim.x\app_device_cdc_basic.c

```c
/*******************************************************************************
Copyright 2016 Microchip Technology Inc. (www.microchip.com)

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

To request to license the code under the MLA license (www.microchip.com/mla_license),
please contact mla_licensing@microchip.com
*******************************************************************************/

/** INCLUDES *******************************************************/
#include "system.h"

#include <stdint.h>
#include <string.h>
#include <stddef.h>

#include "usb.h"
#include "drv_spi.h"

#include "app_led_usb_status.h"
#include "app_device_cdc_basic.h"
#include "usb_config.h"

/** VARIABLES ******************************************************/

static bool buttonPressed;
static char buttonMessage[] = "Button pressed.\r\n";
static uint8_t readBuffer[CDC_DATA_OUT_EP_SIZE];
static uint8_t writeBuffer[CDC_DATA_IN_EP_SIZE];

static char Latch[3];

/********************************************************************
* Function: void enable_latch();
*
* Overview: Brings the pin related to the LE signal at a high level in
* order to generate a pulse.
*
* PreCondition: None
*
* Input: None
*
* Output: None
*
*******************************************************************/
void enable_latch(){
    LATDbits.LATD7=1; //LE=1
    LATDbits.LATD7=1;
```

```
}

/*****************************************************************************
* Function: void disable_latch();
*
* Overview: Brings the pin related to the LE signal at a low level.
*
* PreCondition: None
*
* Input: None
*
* Output: None
*
****************************************************************************/
void disable_latch(){
   LATDbits.LATD7=0; //LE=0
   LATDbits.LATD7=0;
}

/*****************************************************************************
* Function: void initialize_SPI();
*
* Overview: Initializes the SPI communication to send data to the
* PLL (signals LE, DATA and CLK).
*
* PreCondition: None
*
* Input: None
*
* Output: None
*
****************************************************************************/
void initialize_SPI(){
   TRISBbits.TRISB4=0; //Output. CLK
   TRISCbits.TRISC7=0; //Output. DATA
   TRISDbits.TRISD7=0; //Output. LE (Latch Enable)

   disable_latch();   //Default low state for the LE signal

   DRV_SPI_INIT_DATA  spiInitData = {1, 0, 1, SPI_BUS_MODE_0, 0};  //The structure that defines
                                                   //the SPI channel's operation

   DRV_SPI_Initialize(&spiInitData);   //Initializes the SPI instance specified by the channel
                                 //of the initialization structure.
}

/*****************************************************************************
* Function: void APP_DeviceCDCBasicDemoInitialize(void);
*
* Overview: Initializes the demo code.
*
* PreCondition: None
*
* Input: None
*
* Output: None
*
****************************************************************************/
void APP_DeviceCDCBasicDemoInitialize()
{
   line_coding.bCharFormat = 0;
```

```
    line_coding.bDataBits = 8;
    line_coding.bParityType = 0;
    line_coding.dwDTERate = 9600;

    buttonPressed = false;

    initialize_SPI();
}

/*********************************************************************
* Function: void APP_DeviceCDCBasicDemoTasks(void);
*
* Overview: Keeps the demo running.
*
* PreCondition: The demo should have been initialized and started via
* the APP_DeviceCDCBasicDemoInitialize() and APP_DeviceCDCBasicDemoStart() demos
* respectively.
*
* Input: None
*
* Output: None
*
********************************************************************/

void APP_DeviceCDCBasicDemoTasks()
{
    /* If the USB device isn't configured yet, we can't really do anything
     * else since we don't have a host to talk to.  So jump back to the
     * top of the while loop. */
    if( USBGetDeviceState() < CONFIGURED_STATE )
    {
        return;
    }

    /* If we are currently suspended, then we need to see if we need to
     * issue a remote wakeup.  In either case, we shouldn't process any
     * keyboard commands since we aren't currently communicating to the host
     * thus just continue back to the start of the while loop. */
    if( USBIsDeviceSuspended()== true )
    {
        return;
    }

    /* If the user has pressed the button associated with this demo, then we
     * are going to send a "Button Pressed" message to the terminal.
     */
    if(BUTTON_IsPressed(BUTTON_DEVICE_CDC_BASIC_DEMO) == true)
    {
        /* Make sure that we only send the message once per button press and
         * not continuously as the button is held.
         */
        if(buttonPressed == false)
        {
            /* Make sure that the CDC driver is ready for a transmission.
             */
            if(mUSBUSARTIsTxTrfReady() == true)
            {
                putrsUSBUSART(buttonMessage);
                buttonPressed = true;
            }
        }
```

```
}
else
{
    /* If the button is released, we can then allow a new message to be
     * sent the next time the button is pressed.
     */
    buttonPressed = false;
}

/* Check to see if there is a transmission in progress, if there isn't, then
 * we can see about performing an echo response to data received.
 */
if( USBUSARTIsTxTrfReady() == true)
{
    uint8_t i;
    uint8_t numBytesRead;

    /* We retrieve the 3 bytes sent on the USB bus.
     * readBuffer : buffer containing the data read on the USBUSART bus.
     * numBytesRead  : number of bytes read.
     */
    numBytesRead = getsUSBUSART(readBuffer, 3);

    /* PLL initialization : Initialization Latch Method
     * 1. Apply VDD.
     * 2. Program the Initialization Latch (with COUNTER_RESET=0).
     * 3. Do a Function Latch load (with COUNTER_RESET=0).
     * 4. Do an R Latch load.
     * 5. Do an AB Latch load.
     */

    disable_latch();    //Make sure that the LE signal is low before starting
                    //a transfer.

    /* As soon as we receive something ... */
    if(numBytesRead >0){
        for(i=0;i<3;i++){
            Latch[i] = readBuffer[i];       //Store the 3 bytes read in an array
        }
        disable_latch();                    //LE in low state during data transfer
        DRV_SPI_PutBuffer (1, Latch, 3);    //Send data buffer on SPI bus channel 1
        enable_latch();                     //LE in high state to finalize transfer
        disable_latch();                    //Make sure LE goes low again
    }

    /* For every byte that was read... */
    for(i=0; i<numBytesRead; i++)
    {
        switch(readBuffer[i])
        {
            /* If we receive new line or line feed commands, just echo
             * them direct.
             */
            case 0x0A:
            case 0x0D:
                writeBuffer[i] = readBuffer[i];
                break;

            /* If we receive something else, then echo it plus one
             * so that if we receive 'a', we echo 'b' so that the
             * user knows that it isn't the echo enabled on their
```

```
                 * terminal program.
                 */
                default:
                    writeBuffer[i] = readBuffer[i] + 1;
                    break;
            }
        }

        if(numBytesRead > 0)
        {
            /* After processing all of the received data, we need to send out
             * the "echo" data now.
             */
            putUSBUSART(writeBuffer,numBytesRead);
        }
    }

    CDCTxService();
}
```