Generic HID USB Interface for the NXP FRDM-KL43Z Demo Board for EE342

October 12, 2019 John B. Gilmer Jr.

Background:

EE342 Microcontroller Based Design has evolved considerably over the past three decades. When I began teaching EE342 in 1992, it was very hardware oriented, with students building computers on large solderless breadboards using the M68000. Software was done in machine / assembly language. This limited the complexity of what could be done with software, since hardware issues required considerable attention. Later the course moved to the use of microcontrollers, with the "DEMOJM" prototyping board, featuring a Coldfire processor. The hardware complexity was greatly reduced, and focused more on interfaces to sensors and actuation. This allowed software (programmed in C) to be developed including serial communications with a terminal emulator on a PC, command line processing, and eventually interactions with a PC application over USB, allowing a "point and click" interface.

When the course was offered Fall 2018, it was discovered that the Freescale / NXP Coldfire processor family had been discontinued, and the "DEMOJM" demo boards were no longer available. Even the simpler S08JM processors and their support was gone. This required a shift to a current microcontroller family. The NXP 9S08H8 (previously used in EE247) was used for the first part of EE342, then a shift was made to the NXP/Freescale Kinetis family, specifically the FRDM-KL43Z demo board. The new boards proved quite capable, but are considerably more complex. It was not possible to develop ways to do all that had been done with the Coldfire series earlier. Specifically, the USB interface to an application running on a PC was beyond the scope of what could be achieved. There just wasn't time.

The FRDM-KL43Z and related processors have several advantages over the S08 and Coldfire processors used earlier in EE342. It's a current and widely used processor family. The development context and tools are more sophisticated. This requires students to master software abstractions more thoroughly than was necessary before. While that adds to the scope of instruction, it does prepare students better for doing microcontroller based applications. Finally, the cost of hardware is reduced. The DEMOJM boards cost $100, the KL43Z boards are only $20. (However, the DEMOJM had more basic features like LED's and pushbuttons. The FRDM-KL43Z has only two pushbuttons and two LED's. The FRDM-KL-43Z does have an LCD display, but using the display requires considerable effort. It also comes with no connectors soldered on. So labor and extra parts are needed to do that.)

The biggest problem has been the lack of a USB interface to a PC application. This was an important feature of the course, since students would be simultaneously developing the microcontroller and the PC applications so that they worked with each other, as well as the hardware connected to the microcontroller. This gave students an introduction to programming in a sophisticated operating system environment with a Graphics User Interface (GUI), Windows, something quite a bit more common than a text interface these days.

This document serves to report a success in achieving a USB microcontroller to PC interface, similar to what was used with the Coldfire processor. This can be used as a departure point for students to develop more fully capable interfaces for their projects. The methodology will be described, and attached files and discussion are intended to help a student get started.

One caution: This was done with the Kinetis Development System software. We may need to use the MCUXpresso Integrated development Environment (IDE) in the future. KDS seems to have been discontinued or at least no longer supported. That should not be difficult.

Approach:

The intent was to develop something very similar to the USB demonstration application that was used to illustrate this capability for the Coldfire processor on the DEMOJM platform. CMX, the company that developed the prototyping tools, had relatively simple applications for both the PC and microcontroller ends that were packaged in a demonstration product. The PC application presented a GUI to the user as shown in Figure 1 below. The user could click the LED buttons to make the different LED's come on. The application also showed which pushbuttons on the demo board were pushed. One of the projects for students was to extend the demo to include all four pushbuttons and all eight LED's on the DEMOJM board. That required modifying the original CMX demo code at both ends, helping the students gain understanding of how the software was structured, and how it worked together.

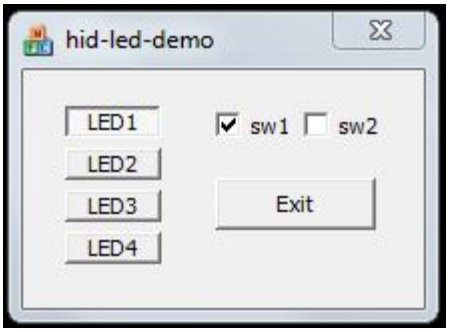

Figure 1 HID-LED-Demo Graphics User Interface

The Kinetis Development System (KDS) also included a wide variety of demo packages. However, nothing was similar to the CMX Human interface device (HID) led demo. (More common "human interface devices" included mice, keyboards, printers, displays and such. The variety used in the Coldfire demo was a "generic" version, useful for interfacing a custom scientific instrument to a computer, and adaptable to a wide range of uses.) The KDS support did include a "generic" HID demo, but it didn't interface to anything available and I was unable to do anything useful or illustrative with it. Nevertheless, it did serve as a starting point for something that was useful, as described here. The existing CMS PC application would serve as the PC end of the USB connection; it required very little modification.

So, the approach taken was to modify the KDS "dev-hid-generic-lite-bm-frdmkl43z" demonstration project to work with the CMX "hid-led-demo" Windows application.

The PC End:

This was relatively simple and straightforward. Two modifications were needed. First, the PC application looks for a matching USB device by examining the manufacturer's "Vendor ID" (assigned by the some authority, perhaps Microsoft) and a "Product ID" for the particular device that is to interface to the PC. Adapting this to the FRDMKL43Z (in place of the DEMOJM) could be accomplished by changing these ID's in the source file "hid_dev.cpp", the HID_Open (void) function, as shown in the line of code modified below. (These ID's were found in the KDS demo project "usb_device_config.h" file, where various information is defined that is to be passed to the PC when the USB link is opened or in message exchanges.

```
const int vid=0x15a2, pid=0x007f;     /* Vendor and product ID of the device. */
```

The second issue is that the original hid-led-demo application used only one byte messages. Flags for the various LED's (outbound from the PC) and pushbuttons (inbound) were packed into that one byte. (One of the things students needed to do later was extend the size of the messages, and define what data would be sent and how they would convey that data.) The sample generic hid demo for the KL43Z used eight byte messages. This required changing the code in the functions in the PC application dialog window file that sent and received data via USB. The code fragment below from the dialog initialization function sends all zeros. An array of eight bytes was substituted for the initial single character (byte).

```
//Turn off all LEDs
     /* Modify for 8 bytes for Freescale KL43Z JBG 10-12-19 */
     /* unsigned c=0;*/
     unsigned char cc[8]={0,0,0,0,0,0,0,0};
     /*HIDWrite(&c);*/
     HIDWrite(cc);
```

Similarly, the previously used single characters were inserted into or extracted from eight byte arrays elsewhere. In the "update_leds" function the change made is shown below.

```
/* Modify for 8 bytes for Freescale KL43Z JBG 10-12-19 */
/*HIDWrite(&lstate);*/
cc[0]=lstate; /* added JBG */
HIDWrite(cc); /* added JBG */
```

And, in the "update_sws function, a similar modification was made to data received back from the microcontroller:

```
static void update_sws(void)
{
  unsigned char lstate=0;
  /* Modify for 8 bytes for Freescale KL43Z JBG 10-12-19 */
  unsigned char cc[8];
  /*if (HIDRead(&lstate))*/
  if (HIDRead(cc)) /* added JBG */
  {
       lstate=cc[0]; /* added JBG */
```

With these changes, the application worked well. Of note, the FRDM_KL43Z has only two LED's, so the bottom two buttons send data to initiate those LED's but they don't exist. The two switches on the KL43Z board are known as "SW1" and "SW3". ("SW2" is a reset.) It didn't seem worthwhile to modify the application to identify the second switch as SW3.

The Microcontroller End:

The starting point was the "dev-hid-generic-lite-bm-frdmkl43z" KDS demonstration project. Changes needed included adding facility for manipulating the LED's and monitoring the switches. This was more complicated than expected due to the structure of virtual folders in the project. Some needed include files were not "visible" to the files that would normally need them. As an expedient default, these initializations were made to the main program (in the "hid_generic.c" file). The modifications needed for switches and pushbuttons included the addition of four include files:

```
/* Added JBG 10-12-19 4 lines below:*/
#include "clock_config.h" //These four files are needed for configuring I/O (LEDs,Sws)
#include "fsl_gpio.h"
#include "fsl_port.h"
#include "board.h"
```

Initialization code for the two LED's and two switches were added to main() after clock initialization. This may seem like a lot of code for just four pins. It is. It is. Not shown is all the additional code in needed include files. The corresponding code for the Coldfire in CodeWarrior would have needed maybe 4 lines of code. Isn't progress wonderful!

```
    BOARD_BootClockRUN();
    /*Added JBG 10-12-19 Would have been in board.c but can't get to needed includes*/
    CLOCK_EnableClock(kCLOCK_PortD);
    CLOCK_EnableClock(kCLOCK_PortE);
    PORT_SetPinMux(PORTD, 5U,kPORT_MuxAsGpio); //green LED
    PORT_SetPinMux(PORTE, 31U,kPORT_MuxAsGpio); //red LED
    gpio_pin_config_t LED_GREEN_config ={
                    .pinDirection =kGPIO_DigitalOutput,
                    .outputLogic = 1U};
    gpio_pin_config_t LED_RED_config ={
                    .pinDirection =kGPIO_DigitalOutput,
                    .outputLogic = 1U};
    GPIO_PinInit(BOARD_LED_GREEN_GPIO, BOARD_LED_GREEN_GPIO_PIN, &LED_GREEN_config);
    GPIO_PinInit(BOARD_LED_RED_GPIO, BOARD_LED_RED_GPIO_PIN, &LED_RED_config);
    const port_pin_config_t LED_GREEN
                  ={kPORT_PullDisable,kPORT_SlowSlewRate,kPORT_PassiveFilterDisable,
                   kPORT_LowDriveStrength, kPORT_MuxAsGpio};
    PORT_SetPinConfig(BOARD_LED_GREEN_GPIO_PORT,BOARD_LED_GREEN_GPIO_PIN, &LED_GREEN);
    const port_pin_config_t LED_RED
                  ={kPORT_PullDisable,kPORT_SlowSlewRate,kPORT_PassiveFilterDisable,
                   kPORT_LowDriveStrength, kPORT_MuxAsGpio};
    PORT_SetPinConfig(BOARD_LED_RED_GPIO_PORT, BOARD_LED_RED_GPIO_PIN, &LED_RED);

    CLOCK_EnableClock(kCLOCK_PortA); //Here and following for the two switches SW1,SW3
    CLOCK_EnableClock(kCLOCK_PortC);
    gpio_pin_config_t SW1_config ={
              .pinDirection =kGPIO_DigitalInput,
                    .outputLogic = 0U};
    GPIO_PinInit(BOARD_SW1_GPIO, BOARD_SW1_GPIO_PIN, &SW1_config);
    const port_pin_config_t SW1
             ={kPORT_PullUp,kPORT_FastSlewRate,kPORT_PassiveFilterDisable,
             kPORT_LowDriveStrength, kPORT_MuxAsGpio};
    PORT_SetPinConfig(BOARD_SW1_PORT, BOARD_SW1_GPIO_PIN, &SW1);
    gpio_pin_config_t SW3_config ={
              .pinDirection =kGPIO_DigitalInput,
                    .outputLogic = 0U};
    GPIO_PinInit(BOARD_SW3_GPIO, BOARD_SW3_GPIO_PIN, &SW3_config);
    const port_pin_config_t SW3
             ={kPORT_PullUp,kPORT_FastSlewRate,kPORT_PassiveFilterDisable,
             kPORT_LowDriveStrength, kPORT_MuxAsGpio};
    PORT_SetPinConfig(BOARD_SW3_PORT, BOARD_SW3_GPIO_PIN, &SW3);
```

The next issue is how to get the switch data to the needed USB HID outputs. This was done by "polling" in the main loop to keep things simple. main() was modified to detect changes in the switches and send in "IN" message to the PC when a change occurred. The function "USB_DeviceSendRequest()" stores the message pending the device being polled by the PC.

```
int main(void){
    unsigned char sw1=1,sw2=1,s1,s2;
       …… (lots of other stuff)…..
       ..............................
    /* MAIN LOOP CHECKS SWITCHES */
    while (1U)
    {
       s1=GPIO_ReadPinInput(BOARD_SW1_GPIO, BOARD_SW1_GPIO_PIN);
       s2=GPIO_ReadPinInput(BOARD_SW3_GPIO, BOARD_SW3_GPIO_PIN);
       if(s1!=sw1||s2!=sw2){  /* If switch has changed */
              sw1=s1;
              sw2=s2;
              g_UsbDeviceHidGeneric.buffer[g_UsbDeviceHidGeneric.bufferIndex][0]=
                             (1-s1)|((1-s2)<<1);
          USB_DeviceSendRequest(g_UsbDeviceHidGeneric.deviceHandle,
                         USB_HID_GENERIC_ENDPOINT_IN,(uint8_t *)
             &g_UsbDeviceHidGeneric.buffer[g_UsbDeviceHidGeneric.bufferIndex][0],
                         USB_HID_GENERIC_IN_BUFFER_LENGTH);
       } /* End of if for switches*/
    }/* End while for main loop */
}/* end of main() */
```

Getting "OUT" messages in to the microcontroller is handled by interrupt. (For the Coldfire, and interrupt was used by the USB device but the data waited to be polled by the main program. Here the processing was put in the interrupt handler. The interrupt handler was already in the demonstration application. It needed to be modified to do something useful, that is, accept and act on the LED data. The code to do that is shown as the modified interrupt handler below: (Appendix A contains the full hid.generic.c file, the only file that had to be modified, which was very convenient, and will be helpful in simplifying things for the students.

```
/* The hid generic interrupt OUT endpoint callback */  // This one is busy.
static usb_status_t USB_DeviceHidGenericInterruptOut(usb_device_handle handle,
                     usb_device_endpoint_callback_message_struct_t *message,
                                                        void *callbackParam){
       unsigned char ldata, swdata;
       if (g_UsbDeviceHidGeneric.attach) {
        USB_DeviceSendRequest(g_UsbDeviceHidGeneric.deviceHandle,
              USB_HID_GENERIC_ENDPOINT_OUT, (uint8_t *)
             &g_UsbDeviceHidGeneric.buffer[g_UsbDeviceHidGeneric.bufferIndex][0],
             USB_HID_GENERIC_OUT_BUFFER_LENGTH);
        /* Content added to the demo here JBG 10-12-19 */
        ldata=g_UsbDeviceHidGeneric.buffer[g_UsbDeviceHidGeneric.bufferIndex][0];
        if(ldata&1)LED_RED_ON();
        else LED_RED_OFF();
        if(ldata&2)LED_GREEN_ON();
        else LED_GREEN_OFF();
        g_UsbDeviceHidGeneric.bufferIndex ^= 1U;
        return USB_DeviceRecvRequest(g_UsbDeviceHidGeneric.deviceHandle,
             USB_HID_GENERIC_ENDPOINT_OUT, (uint8_t *)
              &g_UsbDeviceHidGeneric.buffer[g_UsbDeviceHidGeneric.bufferIndex][0],
               USB_HID_GENERIC_OUT_BUFFER_LENGTH);}
    return kStatus_USB_Error;}
```

Conclusion:

It works! This can serve as a departure point for EE342 projects going forward.

Appendix A File “hid_generic.c” as modified.

```
/*
 * Copyright (c) 2015, Freescale Semiconductor, Inc.
 * All rights reserved.
 * Modified J.Gilmer Oct 12 2019 to work with HID_LED_Demo from CMX Coldfire
 * This is based on the KDS example "dev_hid_generic_lite_bm_frdmkl43z example.
 * I have deleted the stuff in this file that was not active (C++ and ECHI)
 *
 * Redistribution and use in source and binary forms, with or without modification,
 * are permitted provided that the following conditions are met:
 *
 * o Redistributions of source code must retain the above copyright notice, this list
 *   of conditions and the following disclaimer.
 *
 * o Redistributions in binary form must reproduce the above copyright notice, this
 *   list of conditions and the following disclaimer in the documentation and/or
 *   other materials provided with the distribution.
 *
 * o Neither the name of Freescale Semiconductor, Inc. nor the names of its
 *   contributors may be used to endorse or promote products derived from this
 *   software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
 * ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
 * ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */
#include "usb_device_config.h"  // This defines the USB_DEVICE_CONFIG constants.
#include "usb.h"                            // Lots of stuff in here.
#include "usb_device.h"                     // Many of the USB things we need are in here.
#include "usb_device_hid.h"           // USB_DEVICE_HID_REQUEST definitions
#include "usb_device_ch9.h"           // Don't know why this is here.
#include "usb_device_descriptor.h" //IMPORTANT: This is what the uC reports to the PC.
#include "hid_generic.h"
#include "fsl_device_registers.h"
#include "clock_config.h"
#include "board.h"
#include "fsl_debug_console.h"
/* Added JBG 10-12-19 4 lines below:*/
#include "clock_config.h" //These four files are needed for configuing I/O (LEDs, Sws)
#include "fsl_gpio.h"
#include "fsl_port.h"
#include "board.h"
#include <stdio.h>
#include <stdlib.h>
#include "fsl_common.h"
#include "pin_mux.h"  // Also needed for configuring pins.
/*******************************************************************************
 * Prototypes
 ******************************************************************************/
void BOARD_InitHardware(void);  /* The board.c file ought to configure the pins. Doesn't.*/
/* Below are ISR's for incoming (OUT from PC) and outgoing (IN to PC) messages */
static usb_status_t USB_DeviceHidGenericInterruptIn(usb_device_handle handle,
                      usb_device_endpoint_callback_message_struct_t *message,
                                                    void *callbackParam);
static usb_status_t USB_DeviceHidGenericInterruptOut(usb_device_handle handle,
                       usb_device_endpoint_callback_message_struct_t *message,
                                                     void *callbackParam);
static void USB_DeviceApplicationInit(void);  // Starts up USB
/*******************************************************************************
 * Variables
 ******************************************************************************/
```

```
usb_hid_generic_struct_t g_UsbDeviceHidGeneric;  /* this is the HID device */
extern uint8_t g_UsbDeviceCurrentConfigure;
extern uint8_t g_UsbDeviceInterface[USB_HID_GENERIC_INTERFACE_COUNT];
/**************************************************************************
 * Code
 *************************************************************************/
/* The hid generic interrupt IN endpoint callback */  // This basically does nothing.
static usb_status_t USB_DeviceHidGenericInterruptIn(usb_device_handle handle,
                     usb_device_endpoint_callback_message_struct_t *message,
                                                 void *callbackParam){
    if (g_UsbDeviceHidGeneric.attach){}
    return kStatus_USB_Error;}

/* The hid generic interrupt OUT endpoint callback */  // This one is busy.
static usb_status_t USB_DeviceHidGenericInterruptOut(usb_device_handle handle,
                     usb_device_endpoint_callback_message_struct_t *message,
                                                  void *callbackParam){
        unsigned char ldata, swdata;
    if (g_UsbDeviceHidGeneric.attach) {
        USB_DeviceSendRequest(g_UsbDeviceHidGeneric.deviceHandle, USB_HID_GENERIC_ENDPOINT_OUT,
             (uint8_t *)&g_UsbDeviceHidGeneric.buffer[g_UsbDeviceHidGeneric.bufferIndex][0],
             USB_HID_GENERIC_OUT_BUFFER_LENGTH);
        /* Content added to the demo here JBG 10-12-19 */
        usb_echo("Got OUT endpoint int\n\r");
        ldata=g_UsbDeviceHidGeneric.buffer[g_UsbDeviceHidGeneric.bufferIndex][0];
        if(ldata&1)LED_RED_ON();
        else LED_RED_OFF();
        if(ldata&2)LED_GREEN_ON();
        else LED_GREEN_OFF();
        g_UsbDeviceHidGeneric.bufferIndex ^= 1U;
        return USB_DeviceRecvRequest(g_UsbDeviceHidGeneric.deviceHandle,
USB_HID_GENERIC_ENDPOINT_OUT,
              (uint8_t *)&g_UsbDeviceHidGeneric.buffer[g_UsbDeviceHidGeneric.bufferIndex][0],
              USB_HID_GENERIC_OUT_BUFFER_LENGTH);}
    return kStatus_USB_Error;}

/* The device callback */  //This & following is concerned with setting up the USB-HID
usb_status_t USB_DeviceCallback(usb_device_handle handle, uint32_t event, void *param){
    usb_status_t error = kStatus_USB_Success;
    uint8_t *temp8 = (uint8_t *)param;
    switch (event) {
        case kUSB_DeviceEventBusReset:{
            /* USB bus reset signal detected */
            /* Initialize the control pipes */
            USB_DeviceControlPipeInit(g_UsbDeviceHidGeneric.deviceHandle);
            g_UsbDeviceHidGeneric.attach = 0U;}
        break;
        case kUSB_DeviceEventSetConfiguration:
            if (USB_HID_GENERIC_CONFIGURE_INDEX == (*temp8)){
                /* If the confguration is valid, initliaze the HID generic interrupt IN pipe */
                usb_device_endpoint_init_struct_t epInitStruct;
                usb_device_endpoint_callback_struct_t endpointCallback;

                endpointCallback.callbackFn = USB_DeviceHidGenericInterruptIn;
                endpointCallback.callbackParam = handle;

                epInitStruct.zlt = 0U;
                epInitStruct.transferType = USB_ENDPOINT_INTERRUPT;
                epInitStruct.endpointAddress =
                    USB_HID_GENERIC_ENDPOINT_IN | (USB_IN <<
                              USB_DESCRIPTOR_ENDPOINT_ADDRESS_DIRECTION_SHIFT);
                if (USB_SPEED_HIGH == g_UsbDeviceHidGeneric.speed){
                    epInitStruct.maxPacketSize = HS_HID_GENERIC_INTERRUPT_IN_PACKET_SIZE;}
                else {
                    epInitStruct.maxPacketSize = FS_HID_GENERIC_INTERRUPT_IN_PACKET_SIZE;}
                USB_DeviceInitEndpoint(handle, &epInitStruct, &endpointCallback);
                endpointCallback.callbackFn = USB_DeviceHidGenericInterruptOut;
                endpointCallback.callbackParam = handle;
                epInitStruct.zlt = 0U;
```

```
                epInitStruct.transferType = USB_ENDPOINT_INTERRUPT;
                epInitStruct.endpointAddress =
                    USB_HID_GENERIC_ENDPOINT_OUT | (USB_OUT <<
                            USB_DESCRIPTOR_ENDPOINT_ADDRESS_DIRECTION_SHIFT);
                if (USB_SPEED_HIGH == g_UsbDeviceHidGeneric.speed){
                    epInitStruct.maxPacketSize = HS_HID_GENERIC_INTERRUPT_OUT_PACKET_SIZE;}
                else{
                    epInitStruct.maxPacketSize = FS_HID_GENERIC_INTERRUPT_OUT_PACKET_SIZE;}
                USB_DeviceInitEndpoint(handle, &epInitStruct, &endpointCallback);
                g_UsbDeviceHidGeneric.attach = 1U;
                error = USB_DeviceRecvRequest(
                    g_UsbDeviceHidGeneric.deviceHandle, USB_HID_GENERIC_ENDPOINT_OUT,
                    (uint8_t
*)&g_UsbDeviceHidGeneric.buffer[g_UsbDeviceHidGeneric.bufferIndex][0],
                    USB_HID_GENERIC_OUT_BUFFER_LENGTH);}
            break;
        default:
            break;
    }
    return error;}
/* Get setup buffer */
usb_status_t USB_DeviceGetSetupBuffer(usb_device_handle handle, usb_setup_struct_t
**setupBuffer){
    /* Keep the setup is 4-byte aligned */
    static uint32_t hid_generic_setup[2];
    if (NULL == setupBuffer){
        return kStatus_USB_InvalidParameter;}
    *setupBuffer = (usb_setup_struct_t *)&hid_generic_setup;
    return kStatus_USB_Success;}
/* Configure device remote wakeup */
usb_status_t USB_DevcieConfigureRemoteWakeup(usb_device_handle handle, uint8_t enable){
    return kStatus_USB_InvalidRequest;}
/* Configure the endpoint status (idle or stall) */
usb_status_t USB_DevcieConfigureEndpointStatus(usb_device_handle handle, uint8_t ep, uint8_t
status){
    if (status) {
        if (((USB_HID_GENERIC_ENDPOINT_IN == (ep & USB_ENDPOINT_NUMBER_MASK)) &&
             (ep & USB_DESCRIPTOR_ENDPOINT_ADDRESS_DIRECTION_MASK)) ||
            ((USB_HID_GENERIC_ENDPOINT_OUT == (ep & USB_ENDPOINT_NUMBER_MASK)) &&
             (!(ep & USB_DESCRIPTOR_ENDPOINT_ADDRESS_DIRECTION_MASK)))){
            return USB_DeviceStallEndpoint(handle, ep);}
        else{}}
    else{
        if (((USB_HID_GENERIC_ENDPOINT_IN == (ep & USB_ENDPOINT_NUMBER_MASK)) &&
             (ep & USB_DESCRIPTOR_ENDPOINT_ADDRESS_DIRECTION_MASK)) ||
            ((USB_HID_GENERIC_ENDPOINT_OUT == (ep & USB_ENDPOINT_NUMBER_MASK)) &&
             (!(ep & USB_DESCRIPTOR_ENDPOINT_ADDRESS_DIRECTION_MASK)))) {
            return USB_DeviceUnstallEndpoint(handle, ep);}
        else{}}
    return kStatus_USB_InvalidRequest;}
/* Get class-specific request buffer */
usb_status_t USB_DeviceGetClassReceiveBuffer(usb_device_handle handle,
        usb_setup_struct_t *setup,uint32_t *length, uint8_t **buffer){
    static uint8_t setupOut[8];
    if ((NULL == buffer) || ((*length) > sizeof(setupOut))){
        return kStatus_USB_InvalidRequest;}
    *buffer = setupOut;
    return kStatus_USB_Success;}
/* Handle class-specific request */
usb_status_t USB_DeviceProcessClassRequest(usb_device_handle handle,
       usb_setup_struct_t *setup,uint32_t *length,uint8_t **buffer){
    usb_status_t error = kStatus_USB_InvalidRequest;
    if (setup->wIndex != USB_HID_GENERIC_INTERFACE_INDEX){
        return error;}
    switch (setup->bRequest){
        case USB_DEVICE_HID_REQUEST_GET_REPORT:
            break;
        case USB_DEVICE_HID_REQUEST_GET_IDLE:
            break;
```

```
        case USB_DEVICE_HID_REQUEST_GET_PROTOCOL:
            break;
        case USB_DEVICE_HID_REQUEST_SET_REPORT:
            break;
        case USB_DEVICE_HID_REQUEST_SET_IDLE:
            break;
        case USB_DEVICE_HID_REQUEST_SET_PROTOCOL:
            break;
        default:
            break;}
    return error;}
void USB0_IRQHandler(void){
    USB_DeviceKhciIsrFunction(g_UsbDeviceHidGeneric.deviceHandle);}
static void USB_DeviceApplicationInit(void){
    uint8_t irqNumber;
    uint8_t usbDeviceKhciIrq[] = USB_IRQS;
    irqNumber = usbDeviceKhciIrq[CONTROLLER_ID - kUSB_ControllerKhci0];
    SystemCoreClockUpdate();
    CLOCK_EnableUsbfs0Clock(kCLOCK_UsbSrcIrc48M, 48000000U);
    /* Set HID generic default state */
    g_UsbDeviceHidGeneric.speed = USB_SPEED_FULL;
    g_UsbDeviceHidGeneric.attach = 0U;
    g_UsbDeviceHidGeneric.deviceHandle = NULL;
    /* Initialize the usb stack and class drivers */
    if (kStatus_USB_Success != USB_DeviceInit(CONTROLLER_ID, USB_DeviceCallback,
                &g_UsbDeviceHidGeneric.deviceHandle)){
        usb_echo("USB device generic failed\r\n");
        return;}
    else{
        usb_echo("USB device HID generic demo\r\n");}
    /* Install isr, set priority, and enable IRQ. */
    NVIC_SetPriority((IRQn_Type)irqNumber, USB_DEVICE_INTERRUPT_PRIORITY);
    NVIC_EnableIRQ((IRQn_Type)irqNumber);
    /* Start USB device HID generic */
    USB_DeviceRun(g_UsbDeviceHidGeneric.deviceHandle);}

/* The MAIN program, including a lot of initialization */
int main(void){
    unsigned char sw1=1,sw2=1,s1,s2;
    BOARD_InitPins();
    BOARD_BootClockRUN();
    /* Added JBG 10-12-19  Would have been in board.c but can't get to needed includes*/
    CLOCK_EnableClock(kCLOCK_PortD);
    CLOCK_EnableClock(kCLOCK_PortE);
    PORT_SetPinMux(PORTD, 5U,kPORT_MuxAsGpio); //green LED
    PORT_SetPinMux(PORTE, 31U,kPORT_MuxAsGpio); //red LED
    gpio_pin_config_t LED_GREEN_config ={
                      .pinDirection =kGPIO_DigitalOutput,
                      .outputLogic = 1U};
    gpio_pin_config_t LED_RED_config ={
                      .pinDirection =kGPIO_DigitalOutput,
                      .outputLogic = 1U};
    GPIO_PinInit(BOARD_LED_GREEN_GPIO, BOARD_LED_GREEN_GPIO_PIN, &LED_GREEN_config);
    GPIO_PinInit(BOARD_LED_RED_GPIO, BOARD_LED_RED_GPIO_PIN, &LED_RED_config);
    const port_pin_config_t LED_GREEN
={kPORT_PullDisable,kPORT_SlowSlewRate,kPORT_PassiveFilterDisable,
                     kPORT_LowDriveStrength, kPORT_MuxAsGpio};
    PORT_SetPinConfig(BOARD_LED_GREEN_GPIO_PORT, BOARD_LED_GREEN_GPIO_PIN, &LED_GREEN);
    const port_pin_config_t LED_RED
={kPORT_PullDisable,kPORT_SlowSlewRate,kPORT_PassiveFilterDisable,
                     kPORT_LowDriveStrength, kPORT_MuxAsGpio};
    PORT_SetPinConfig(BOARD_LED_RED_GPIO_PORT, BOARD_LED_RED_GPIO_PIN, &LED_RED);

    CLOCK_EnableClock(kCLOCK_PortA);  // Here and following for the two switches SW1, SW3
    CLOCK_EnableClock(kCLOCK_PortC);
    gpio_pin_config_t SW1_config ={
                .pinDirection =kGPIO_DigitalInput,
                      .outputLogic = 0U};
    GPIO_PinInit(BOARD_SW1_GPIO, BOARD_SW1_GPIO_PIN, &SW1_config);
```

```
    const port_pin_config_t SW1 ={kPORT_PullUp,kPORT_FastSlewRate,kPORT_PassiveFilterDisable,
              kPORT_LowDriveStrength, kPORT_MuxAsGpio};
    PORT_SetPinConfig(BOARD_SW1_PORT, BOARD_SW1_GPIO_PIN, &SW1);
    gpio_pin_config_t SW3_config ={
                .pinDirection =kGPIO_DigitalInput,
                      .outputLogic = 0U};
    GPIO_PinInit(BOARD_SW3_GPIO, BOARD_SW3_GPIO_PIN, &SW3_config);
    const port_pin_config_t SW3 ={kPORT_PullUp,kPORT_FastSlewRate,kPORT_PassiveFilterDisable,
              kPORT_LowDriveStrength, kPORT_MuxAsGpio};
    PORT_SetPinConfig(BOARD_SW3_PORT, BOARD_SW3_GPIO_PIN, &SW3);

    BOARD_InitDebugConsole();
    USB_DeviceApplicationInit(); // This starts up USB

    /* mAIN LOOP CHECKS SWITCHES */
    while (1U)
    {
        s1=GPIO_ReadPinInput(BOARD_SW1_GPIO, BOARD_SW1_GPIO_PIN);
        s2=GPIO_ReadPinInput(BOARD_SW3_GPIO, BOARD_SW3_GPIO_PIN);
        if(s1!=sw1||s2!=sw2){  /* If switch has changed */
               sw1=s1;
               sw2=s2;
               g_UsbDeviceHidGeneric.buffer[g_UsbDeviceHidGeneric.bufferIndex][0]=(1-s1)|((1-
s2)<<1);
           USB_DeviceSendRequest(g_UsbDeviceHidGeneric.deviceHandle,
USB_HID_GENERIC_ENDPOINT_IN,
                 (uint8_t *)&g_UsbDeviceHidGeneric.buffer[g_UsbDeviceHidGeneric.bufferIndex][0],
                                              USB_HID_GENERIC_IN_BUFFER_LENGTH);
        } /* End of if for switches*/
    }/* End if for main loop */
}/* end of main() */
```

(Note: In earlier fragments a bit of prettying up was done because of inadequate margins.)