Part 1: Push Buttons Using Interrupts

In a previous lab, you interacted with push buttons using polling, in this lab will leverage your previous labs code to interact with the push buttons using interrupts.

In order to use interrupts, you will need to configure interrupts on the GPIO port. Similar to how you utilized the initialization and configuration section of the datasheet in the previous lab, you will need to revisit this and complete the initialization function in button.c for configuring the interrupts.

In addition to setting up the GPIO port to handle interrupts, you will also need to set up the NVIC to tell the CPU which interrupt to use. Use **page 104** in the datasheet to find the interrupt number that corresponds to the port that you are using for interrupts. This information will help you determine which bits to set in an NVIC Enable register **NVIC_ENx_R**. The vector number indicates a group of 32 interrupts that will be decoded as x, a number between 0-4. The bit you will set in the register is the interrupt number. Visit **page 142** in the datasheet to see the register diagram and instructions on how to decode the interrupt and vector numbers. Note the priority field in the NVIC is 0 by default if it is not assigned any priority. You will assign interrupt priority in future labs. **Sections 5.1 – 5.3 of the textbook has further details on interrupts.**

When an interrupt is triggered, your program will jump to the interrupt handler to execute code to take care of the interrupt. A GPIO interrupt is triggered at a port level. Since all the buttons are on the same port, regardless of which button is pressed, the same interrupt will be triggered, and thus the same interrupt handler will be executed for all the buttons. Because of this, when the interrupt is triggered you may need to utilize some parts of your code written in the previous lab to determine which button has been pressed. Print the pressed button number to the LCD screen. Button press functionality from the previous lab should be retained, meaning that if multiple buttons are pressed only the rightmost button is returned. When no button has been pressed, 0 should be displayed.

To begin, complete the function **init_button_interrupts()**. This function will enable the NVIC for detecting when a button connected to Port E is pressed.

**Note: Look up the keyword 'volatile". What does the key word volatile do, and why is it necessary when working with ISR that share a variable with main()?

Once you have completed the function to initialize GPIO interrupts on Port E, you will need to call the function in main similar to your other initialization functions.

Next, complete the interrupt handler **gpioe_handler()**. Specify the code to execute when a push-button event occurs. Remember to clear the Interrupt status at the end of your handler.
WARNING: You should never call **gpioe_handler().**

Checkpoint: Demonstrate the push buttons using interrupts to your mentor

Part 2: Setting up GPIO module for PuTTy to Cybot UART Communication

In Labs 2, 3 & 4, you made use of a UART device to send information between PuTTy and the CyBot. You were given the function called `cyBot_uart_init()`. For this part, you will develop the first half of this initialization process, which involves setting up the GPIO portion of the microcontroller. We provide you with the last half, called: `cyBot_uart_init_last_half`. You will replace `cyBot_uart_init()` with the following code that you need to complete.

```
cyBot_uart_init_clean(); //Clean UART initialization

// Complete code for configuring the(GPIO) part of UART init
SYSCTL_RCGCGPIO_R |= FIXME;
timer_waitMillis(1);          // Small delay

GPIO_PORTB_AFSEL_R |= FIXME;
GPIO_PORTB_PCTL_R &= FIXME; //Force 0's at desired locations
GPIO_PORTB_PCTL_R |= FIXME; // Force 1's at desired locations
GPIO_PORTB_DEN_R |= FIXME;
GPIO_PORTB_DIR_R &= FIXME;  // Force 0's at desired locations
GPIO_PORTB_DIR_R |= FIXME;  // Force 1's at desired locations

cyBot_uart_init_last_half(); // Last half of UART configuration
```

The above will replace the call to `cyBot_uart_init` you made use of in labs 2, 3&4. Your job is to replace the FIXME's with proper values. **Make sure to copy the new libcybotUART.lib** into your project (Note: the previous labs versions do not have the `cyBot_uart_init_last_half` function call.

Note: Make use of the last part of Lab 4 to help you complete the code for this part of the lab.

Checkpoint: Demonstrate to your mentor that you can send and receive data to PuTTy over the UART like in Lab 4, using the code you developed for initializing the first half of the UART interface.

Part 3: Calibrating RAW IR Measurements

In this part of the lab you will convert Raw IR values provided by the CyBot Scan function into calibrated distances.

Summary:

Due to the nonlinear nature of the IR sensor, there is not a simple linear transfer function between the distance being measured and the analog voltage output of the sensor (i.e., Raw IR value). See the IR sensor datasheet to see this nonlinearity graphically: IR datasheet . Thus, you will need to implement a technique to map quantization values (i.e. Raw IR measurement values) to distance values. To do this, you will need to take some measurements. Your task is to accurately display a distance value for an object 9 – 50 cm away.

In order to calibrate the sensor, you will need to collect several data points consisting of the known distance and quantized values (i.e., Raw IR measurement) at that distance. Based on these data points, you can find an equation for a best fit line/curve, or use a lookup table based approach. Your method will return an estimated distance value for a given quantization value. For full credit, the estimated distance calculated by your program must be within 2 cm of the actual distance. Print to the LCD both the quantization value and the estimated distance in cm.

a) <u>Log Data</u>: Set up Putty to send the information it receives to a file.

b) <u>Analyze Data</u>: Use your favorite analysis tool (e.g., Matlab, Excel) to make a Raw IR value vs. PING (or meter stick measured) distance plot of the data in the file. Then use your chosen tool to find an equation that closely fits the measured data. This is called curve fitting.

c) <u>Implement Calibration</u>: Implement your "best-fit" conversion function onto your CyBot.

<u>Note 1:</u> Reduce the variability in the IR raw values by averaging multiple measurements. For example, collect and average N samples to get a more stable sensor value.

<u>Note 2:</u> You may want to use the PING Distance from the CyBOT Scan function sensor to record the distance associated with a given Raw IR sensor value, or just use a meter/yard stick to collect this information.

Checkpoint: Demonstrate the distance measurement. Print to the LCD both the quantization value (i.e., Raw IR value) and the estimated distance in cm (computed from your conversion function). Your distance readings should be within 2 cm of actual values (for distances from 9 – 50 cm). Additionally, you must implement an averaging mechanism in your program. Explain and justify your calibration methods with a detailed description of how your choices were made or implemented

Part 4: Driving the Cybot using a simple Socket program

In Labs 2, 3, and 4, you have made use of PuTTy to communicate between your Desktop PC and the CyBot. PuTTy is an amazingly useful program, especially for debugging a communication link. However, it is not easily extendable for developing solutions for specific tasks.

In this part of lab, you will use a simple Socket program to send messages to the Cybot, and receive a message back. While this program does not have many of the features of PuTTy, it will give you a nice starting point for adding your own application specific features in the future.

Review the starter Socket program:
i) The code: Review the provided starter Socket programs (**simple-Socket-or-UART-client.py** & **simple-mock-Cybot-Socket-server.py** - found on the resource section of the course website) with your lab partner. Give a brief summary of the steps the start Socket programs are taking.
ii) Network Sockets: Review the following Network Socket overview link given within the program (https://realpython.com/python-sockets/) with your lab partner. Based on this overview link briefly summarize what sockets are, and how they may be useful.

Running the socket program without a CyBot:
i) Server: Start the Server Program (**simple-mock-Cybot-Socket-server.py**).
ii) Client: Start the Client Program (**simple-Socket-or-UART-client.py**).
iii) Echo: Within the Client Program's prompt type messages to the server. Verify the Server sends the message back to you.
iv) Modify Server: Review the Server program. Modify it to be a "Chat" server, instead of an "Echo" server. This should only require changing 1-2 lines of the Server program.
v) Chat: Repeat steps i) and ii), using your updated Server program. You should be able to chat back and forth between the Client and Server Socket program.

Use Socket program to drive the Cybot: Instead of having the Client communicate with the given Server program, instead have it communicate with your CyBot.
i) Client: Update your client program to use the **IP address** and **Port number** that you have been using to set up PuTTy communication
ii) Cybot: Use your Lab 2 Cybot program that allows you to manually drive the Cybot to get commands from the Client program for driving the Cybot.

- **Note:** Your Lab 2 code needs to be compatible with the Client (or you need to modify the Client). So first review, and use the **Simple-CyBot-echo.c** C program (found on the resource section of the course website) to communicate between the Client and Cybot. Then modify your Lab 2 code based on this example code to make it compatible with the Client program.

iii) Drive the Cybot: Start your Client code, and use it to manually drive your Cybot. In other words, drive the CyBot using your Client program instead of PuTTy.

Checkpoint: Give a brief explain of the Client code to your Mentor, and given them a brief summary of what Network Sockets are, and why they are useful. Demonstrate that you can drive your CyBot using the Client Socket program.