ECEN 3753
Real Time Operating Systems Lab #2
GPIO, Timers, and Interrupts

Objective: Utilize GPIO and Timers to demonstrate the use of interrupts compared to polling from a main loop for code flow and power consumption.

In this lab, you will utilize the User LEDs, User Button, and Gyro on the STM32F429-DISC1. The lab consists of two parts. In the first part of the lab, you will use a main loop to sample the User Button and Gyro and to illuminate the LEDs based on the user input.

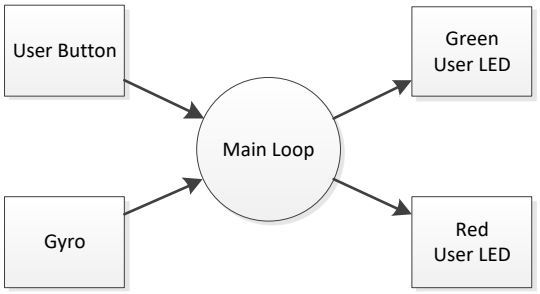

Figure 1: Main Loop Implementation

In the second part of the lab, interrupts will be used instead of polling from the main loop to sample the inputs and drive the output LEDs.

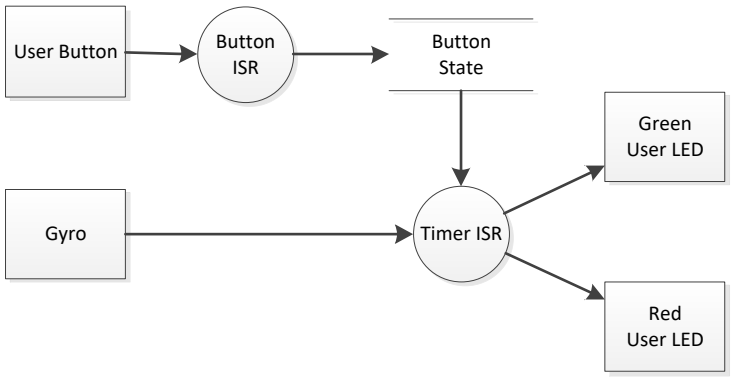

Figure 2: Interrupt-Driven Implementation

References:

1. STM32F429 Reference Manual
   https://canvas.colorado.edu/courses/133970/files?preview=82561816
2. STM32F429 Schematic
   https://canvas.colorado.edu/courses/133970/files?preview=82561806
3. I3G4250D Gyro Reference Manual
   https://www.st.com/resource/en/datasheet/i3g4250d.pdf
4. ECEN 2370 Labs on Gyro, LEDs, and/or Interrupts

Preparation:

1. Review the STM32F429 Reference Manual and/or your previous labs on GPIO, Timers, LEDs, and Gyro
2. Review the HAL functions for controlling the LEDs and for setting up a timer to generate an interrupt (remember to use crtl+space to invoke auto-complete to see what possibilities you have as you type partially guessed names of functions)
3. Review the implementation of an interrupt handler (NEED REFERENCE from 2370)
4. If this board is new to you, the following YouTube videos will help you understand the STM32 and interrupts, Timers, and the HAL.
   a. HAL: See Lab1, and/or Mitch Davis on STM32: Guide #2 and Guide #4
   b. Interrupts: review ECEN2370 Lab 3, which introduced interrupts on this board
   c. PWM and Timers: Mitch Davis on STM32 Guide #3

Procedure:

Part I:

1. Create a new project in CubeIDE. You may either create a project from scratch, or else make a copy of one of the software examples. Name the lab LastName_FirstName_RTOS_Lab2.
2. On .ioc screen for lab 2, make the following changes:
    a. in System Core → GPIO, click on PA0 (should be near the bottom). Click on the dropdown menu for GPIO Mode and change it to interrupt mode with Rising/Falling Edges.
        i. This will let us check for interrupts on the user button.
    b. in Connectivity → SPI5, change the mode to Full-Duplex Master
        i. This will be used to communicate with the gyroscope in this lab.
3. Create a source and header fileset for your application code.
4. Copy the Gyro_Driver.c and .h files into the appropriate folders.
5. Define global variables to store:
    a. The state of the User Button (pressed or not pressed)
    b. The direction of gyro rotation (left or right around the axis pointing from the end furthest from the USB ST-LINK connector towards that connector).
        i. This should be a typedef enum variable (look this up if you are unfamiliar with the concept) with one value for each range rate mentioned in step 7 below.
6. Create a function to sample the User Button.
7. Create a function to determine the rate of gyro rotation (if a new value is ready in the hardware) around the axis indicated above. At a minimum, categorize the rotation rate into 5 rate ranges—2 clockwise(+), one zero (+/- tiny amount), and 2 counter-clockwise(-), with discrimination between around nearly-zero, very-slow-but-affirmative-rotation and faster-than-slow-rotation rates. The code should determine if the rate is positive or negative (I.e. we want RotationRate bins, not |RotationRate| bins). For this lab, either of the two rates with a same sign should be treated the same, and the nearly-zero bin should be counted as a “clockwise rotation”.
8. Create a function that drives each LED based on the value of the User Button (using the global variable defined above) and the gyro rotation classification.
    a. Green User LED - Turn on while Button is pressed or while the gyro is rotating counter-clockwise; turn off otherwise.
    b. Red User LED - Turn on while Button is pressed and the gyro is rotating clockwise; turn off otherwise.
9. From the main loop, call each function to get the state of each input and update the corresponding global variable, then call the function that drives each LED. Delay 100 ms between each update.
10. Put a compile-time switch (e.g., #if !(LAB2_USE_INTERRUPT) …. #endif) around the code in the main loop that samples the inputs and drives the LEDs. Leave this code intact when developing Part II below.

Part II:

1. Use the SysTick timer to sample the inputs and to drive the LEDs at a 100 ms period.
    a. In Lab2/Core/Src/stm32f4xx_it.c, find the function Systick_Handler and add a call to HAL_SYSTICK_IRQHandler();
    b. In ApplicationCode.c, create a prototype for HAL_SYSTICK_Callback();
        i. Both of these HAL functions are predefined, so the syntax is very important.
    c. Use a static variable to determine when 100ms have passed.
        i. Hint 1: figure out how often the systick interrupt is generated.
        ii. Hint 2: a modular operation is a good way to implement this check.
    d. Every 100ms:
        i. Call the function developed in Part I above to sample the gyro and update the global variable.
        ii. Call the function to drive the LEDs.
2. Create the interrupt handler for the GPIO connected to the User Button. The interrupts should be enabled for both the rising and falling edge. From the interrupt handler, call the function defined in Part I above to get the state of the pushbutton and update the corresponding global variable.
    a. The Interrupt function you need has a specific name again. The location of this function is in the startup folder of your project.
    b. make sure you are properly handling your interrupt if it's not working (enabling the interrupt at the start of your code, clearing the interrupt, etc.)
3. In your main while loop, make a call to enter Sleep mode and wait for an interrupt.
    a. There is a HAL function to do this.
4. Put a compile-time switch (e.g., #if (LAB2_USE_INTERRUPT) …. #endif) around the code that is unique to the interrupt-driven implementation for Part II. Make any adjustments to the code as necessary such that only the code that is unique to each implementation is under a switch. The code that samples each input and drives the LEDs should be common, as well as any common initialization code.

Functional Tests:

The following functional tests should pass for both Part I and Part II and are documented in TESTING.md.

1. At reset, observe no LEDs are lit
2. Press the button and observe both LEDs are lit
    a. Release the button and observe neither LED is lit
3. Press the button while rotating clockwise, and observe both LEDs are lit
    a. Release the button and observe neither LED is lit
4. Press the button while rotating counter-clockwise, and observe only the Green LED is lit
5. Release the Button and observe that the Green LED remains lit

Additional Functional Tests:

1. **Create a test case** analogous to FT.3 above to test the LED while rotating clockwise with the button unpressed and add it to the file called "TESTING.md" at the root of your repository.
2. **Create a test case** analogous to FT.4 above to test the LED while rotating counter-clockwise with the button unpressed and include it in a file called "TESTING.md" at the root of your repository.

Grading: No credit for late submissions.

1. 28 points equals "100%" for this lab.
2. All project files and source code must be submitted so the grader can duplicate results. You will receive a ZERO if all project files and source code are not submitted.
3. Basic Construction:
    a. The project builds without warnings and successfully executes as your cited results indicate when the compile-time switch is set for "main loop" mode: 3 pts
    b. Able to build without warnings and successfully run as your cited results indicate when the compile-time switch is set for interrupt mode: 3 pts
    c. No code that is not unique to the mode is under a compile-time switch: 2 pts
4. Functional Testing:
    a. Appropriate wording of student-specified Functional Test #5: 1 pt
    b. Appropriate wording of student-specified Functional Test #6: 1 pt
    c. Each passing functional test: 1 pt each (4 pt maximum)
5. Good Coding:
    a. Header documentation has been provided for all functions: 2 pt
    b. Header documentation follows Doxygen guidelines: 2 pt
    c. Appropriate use of comments inside functions: 2 pt
6. Face-to-face demonstration to your LA that you know:
    a. How to find a particular HAL function that your LA requests: 2 pt
    b. How to configure the gyro to enable only a (particular axis, full-scale rate, and filter cutoff) provided by your LA: 3 pt
    c. How to ensure that the gyro is in polling mode as desired, and how to verify a value is ready for reading: 3 pt
7. Bonus points (will add to the above-listed 25 pts—which can push your lab#2 grade above 100%)
    a. Explain in person to your LA the basic principles by which the Gyro can detect rotational velocity
    b. Provide a report of your flash and RAM code sizes, per instruction from the TA in lab1: 3 pt

Submitting your work:

1. Commit your code to your GitHub repository.
2. Submit a link to your GitHub repository on Canvas for Lab 2.
3. Talk with an LA to verify the Face to Face requirements.