Lab 5: SPI and the Gyro

Due Date: Friday**,** November 17th, 2023, at 5 p.m.

Prerequisites:

- Know how to read and understand documentation
- Solid understanding of the SPI protocol
- Somewhat comfortable with the idea of integrating code that you did not write with code that you did write
- Strong ability in reading code you did not write
- The ability to “trust” that code that you are supposed to leverage is doing what it is saying it does

Background information:

Congratulations, we will no longer have entire labs dedicated for you to write drivers from the ground up. Now, the next critical skill for being a programmer is integrating and leveraging code you did not write and had no say in how it was written. In industry, code integration will be an essential skill and ability you need in your toolbox. This lab will use the HAL for the STM for some drivers. The generated HAL code looks terrifying initially, but if you take it line by line, you will notice you’ve written similar code yourself! **Build your code periodically to resolve build errors as you go.**

Lab Instruction:

1. Verify you understand the prelab and can meet the prerequisites
2. For this lab, you should use lab 3 to build off of, as we will not be utilizing the timer in this lab
3. Create a new project to bring in the HAL Driver for SPI and GPIO
    a. *Refer to prelab if needed*
4. Integrate your source and header files (from lab 3) into the project
    a. You will include all filesets but the GPIO_Driver fileset
    b. DO NOT include your stm32429i.h file, the HAL uses its own version of that file, and it can be found in Drivers\CMSIS\Device\ST\STM32F4xx\Include\stm32f429xx.h
5. Change appropriate function calls
    a. We need to call the correct functions since we are using the HAL GPIO driver.
        i. This may also require you to change certain data types for them to be compliant with the HAL GPIO functions
    b. This includes but is not limited to Init functions and clock enable/disable functions
    c. Be mindful of macro uses, our macros were created for our drivers, they may not map 1 to 1 when using the HAL Driver
        i. For example, notice what we used for GPIO_PIN_0 in prior labs and what STM uses for GPIO_PIN_0 in their labs
6. Update includes as needed
    a. If you don’t want 200+ errors due to “redefinitions” and such, use the general header file

        i. Examine that file and see why this file suffices as the only file to include to interact with the HAL.
7. Consider building your code and resolving the build errors if you can
8. Create and populate the ErrorHandling fileset
    a. The header should contain the function named APPLICATION_ASSERT, this should not return any but takes in a boolean argument
    b. The source should contain the APPLICATION_ASSERT function, which goes into an infinite loop if the input argument is false
        i. It will be very similar to the 'Error Handler' function in the main.c generated by the HAL
9. Create and populate the Gyro fileset
    a. The header should contain the following:
        i. Macros for the Gyro
            1. These should be the addresses of needed registers on the Gyro
        ii. Macros for the Pin and Port information for the Gyro
        iii. The following prototypes:
            1. NOTE- All Gyro prototypes in the Gyro fileset should have "Gyro_" prepended to the function name.
            2. A prototype to Initialize the Gyro
            3. A prototype to get the device ID and print it
            4. A prototype to power on the Gyro
            5. A prototype to reboot the Gyro
            6. A prototype to get the temperature and print it
            7. A prototype to configure the registers on the Gyro
            8. A prototype to verify the HAL status for the SPI is okay
            9. A prototype to manually enable slave communication
            10. A prototype to manually disable slave communication
            11. Once you start writing the source code, there may be repeated code that will make you want to create prototypes.
        iv. You may need some macros later on, it will be your responsibility to add them later on
            1. Remember, we should not have magic numbers in the source code. Create Macros as appropriate.
    b. The source files should contain definitions for the prototypes mentioned above
        i. Read the reference manual and other relevant documentation to determine which SPI configurations are needed
        ii. When configuring the registers for the Gyro
            1. For the Control Register 1
                a. We want to enable the X, Y, and Z axis
                b. We want to have the power down mode to be in normal mode
                c. The bandwidth bit field *should* be okay to be left at 00
            2. For the Control Register 4
                a. Configure the full-scale selection to be 500 dps

3. For the Control Register 5
    a. We want the FIFO to be enabled
    b. We want the Rebot memory content to be enabled
4. For the FIFO Control Register
    a. Configure it such that the FIFO mode is FIFO mode

iii. Create a static variable to keep track of the HAL status
1. That said, any HAL function that returns the status should be assigned to this variable
2. The function responsible for verifying the HAL status is OK should be called regularly (when it is possible for the HAL status to change) and should fall into the application assert infinite loop if the HAL is not okay

iv. You may want to create static variables for other things, and that is okay as long as coding hierarchies are not being violated.

10. Update the scheduler code
    a. You should add the following event flags
        i. Event flag to get the temperature from the Gyro
        ii. Event flag to get the Gyro Axis Data
        iii. Event flag to issue the reboot CMD to the Gyro
    b. You do not need any other eventsbesides the one above
11. Populate the Application Code
    a. Create the prototypes and functions needed to prevent main.c from calling into interfacing with the Gyro_Driver code directly
    b. When adjusting your button interrupt, remember, the interrupt should add the reboot CMD to the scheduled events
12. Clean up and populate main.c
    a. You can keep main.h, and it's include if you want
    b. You may leave the SystemClock_Config and its call in main.c
    c. You may also leave the HAL_Init call in main.c
    d. You may want to use HAL_Delay() right after your call to power on the Gyro to give the Gyro time to power on before trying to interact with it
        i. 100 ms should be plenty of time
    e. Delete the calls to MX_GPIO_Init() and MX_SPI_Init()
    f. Add necessary functions/libraries to be able to use printf
        i. Refer to Lab 1 if needed.
    g. Powering on the Gyro, receiving and printing the Device ID, and configuring the registers for the Gyro should all happen before the super-loop
    h. Integrate the scheduler and call appropriate functions when the appropriate event flags are set
13. Build and debug
14. Verify that all acceptance criteria are met if time permits
15. Zip up your project and turn it into Canvas

Acceptance Criteria:

1. You receive and print the **correct** Device ID from the Gyro
2. You are receiving and printing consistent and *reasonable* temperatures from the Gyro
3. You are pulling and printing *reasonable* raw data from the Gyro Axis registers
4. The reset command is sent when the button is pushed
    a. This doesn't have to be immediate due to the expected implementation

The term *reasonable* is put like that as we are getting raw data and we could make an entire lab on how to parse and interpret that data. As far as temperature selection, the temperature module within the Gyro seems to be cost-efficient so the performance is not super ideal.

Grading rubric:
This lab will be worth 100 points. The breakdown of grading will be given below.

1. Code Compilation (20 points)
    a. Full credit - Code Compiles with 0 errors and 0 code warnings
    b. Partial Credit - Code contains warnings (-3 points for each warning)
    c. No credit - Code does not compile (Student has at least one error)
2. Code Standards and Hierarchy (15 points)
    a. Proper naming of functions/files
    b. Proper layering of files
    c. For each violation, 5 points will get subtracted from the 15 points possible
3. Code functionality (55 points)
    a. 1 (10 points)
    b. 2 & 3 (20 points each)
    c. 4 (5 points)
4. Project Exporting (5 points)
    a. Was the project exported right with the appropriate naming?
5. Lab Attendance (5 points)
    a. Did the student attend lab sessions appropriately and consistently?