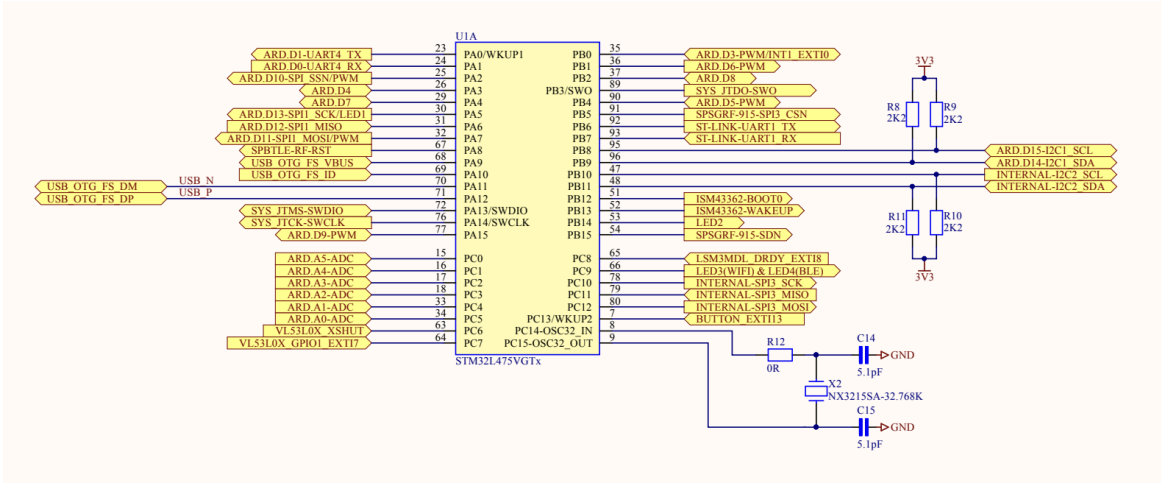
CSE 190 Project 2 - Interfacing with peripherals

Due Friday Feb 7th, 10:00pm. This is a group assignment, same as the previous project. Please submit your code using Gradescope, and use the same starter repository. You will be building on the same codebase that you started with for Project 1, there is no skeleton code provided for this project.

Overview

In the second project you will use serial buses to communicate between your microcontroller and external devices. Then you will build I2C drivers to read the data from your onboard accelerometer. Finally, you will build your first prototype of the PrivTag product that we will be developing in this course.

1. **Implement the I2C driver -** This driver will provide an abstraction of the I2C controller. In this project you will use it to read and write data to/from the accelerometer.
2. **Implement the accelerometer driver -** This driver will provide an abstraction of the key functions of the accelerometer on the STM32 board so you can use it to detect if the tag has been lost (not moved).
3. **Implement the PrivTag application -** This application will sense if the tag has been lost. If it has, it will begin outputting your ID number and how many minutes have passed since it was dropped on the LEDs.



Serial bus wiring diagram for the STM32 including I2C, UART, and SPI (from board's schematic)

Resources

For this project the most important documents are:

- LSM6DSL (accelerometer/gyroscope) - Datasheet
- LSM6DSL(accelerometer/gyroscope) - Application Note
- Schematic - STM32 Development board
- Reference Manual - STM32L Microcontroller
- Datasheet - STM32L475 Microcontroller

Grading Criteria

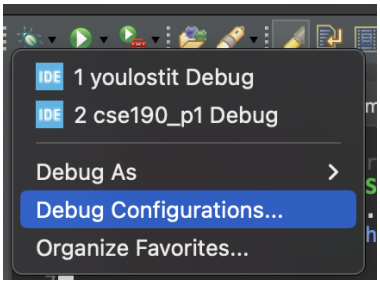
- 20% I2C driver complies to its interface
- 20% Accelerometer driver complies to its interface
- 50% PrivTag application works as described
- 10 % Code readability and commenting. In particular, explain non-obvious things like how you set up your baud rate etc.

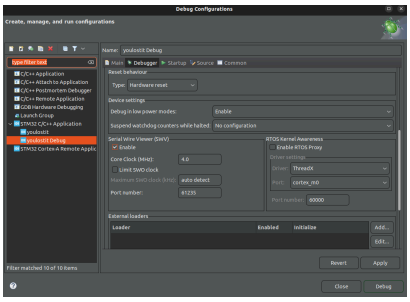
Step 0: Implement printf() debugging

Please add the following code to your `main.c` file so you can debug your code with the `printf()` function. This will be an essential debugging tool so you can see if you are receiving the correct information from your peripheral device (i.e., accelerometer). Adding this function will make the C standard library (`libc`) write characters over the STM32's UART-like character printing bus (called SMV ITM) over USB, which will appear on your STM32CubeIDE.

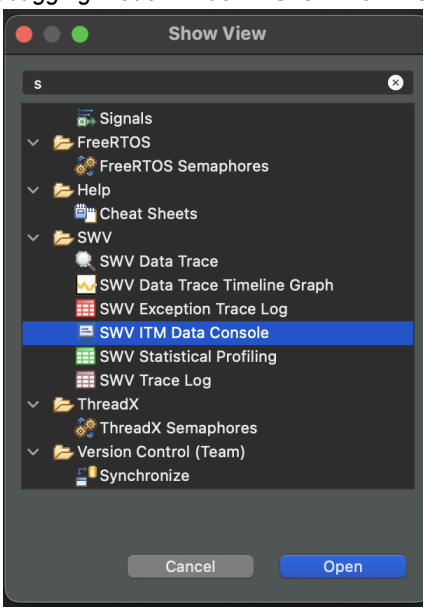
```
// Redefine the libc _write() function so you can use printf in your code
int _write(int file, char *ptr, int len) {
    int i = 0;
    for (i = 0; i < len; i++) {
        ITM_SendChar(*ptr++);
    }
    return len;
}
```

Then you need to enable the SMV debugging console by following the steps below (*right click on the bug to get the config pane)*.

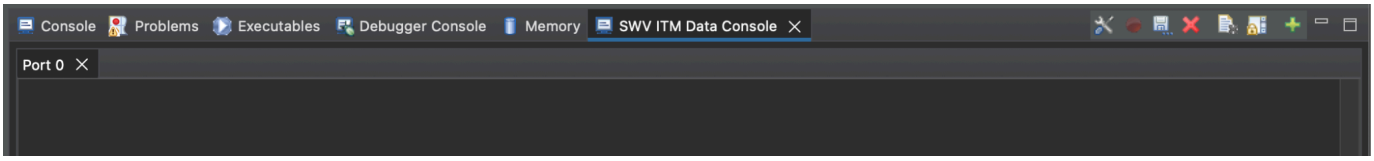




In debugging mode Window->Show View->Other

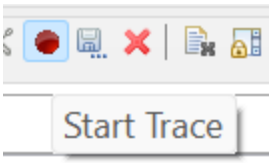


Then, when you run your project in Debug mode, you will see a “SMV ITM Data Console” pane at the bottom. You will see all of your printf statements logged into this console live as you run your project.



Please post any questions you have to the Piazza forum so we can share the knowledge with all the other students.

Click on the start trace button (this may default to being on; if you do not see console output, however, this may be the cause).



Step 1: Implement the I2C driver

The I2C hardware driver will provide a basic abstraction of the STM32’s MCU’s I2C peripheral so it can be easily used to transmit and receive data between the MCU and the STM32 development board’s onboard accelerometer. You will need to implement the following two functions:

Relevant datasheets: STM32 MCU datasheet - Chapter 39 (I2C Peripheral)
Suggested Reading: Programming Embedded Systems (Oriley UCSD login required) Ch. 7
Files that will be graded: i2c.h, i2c.c

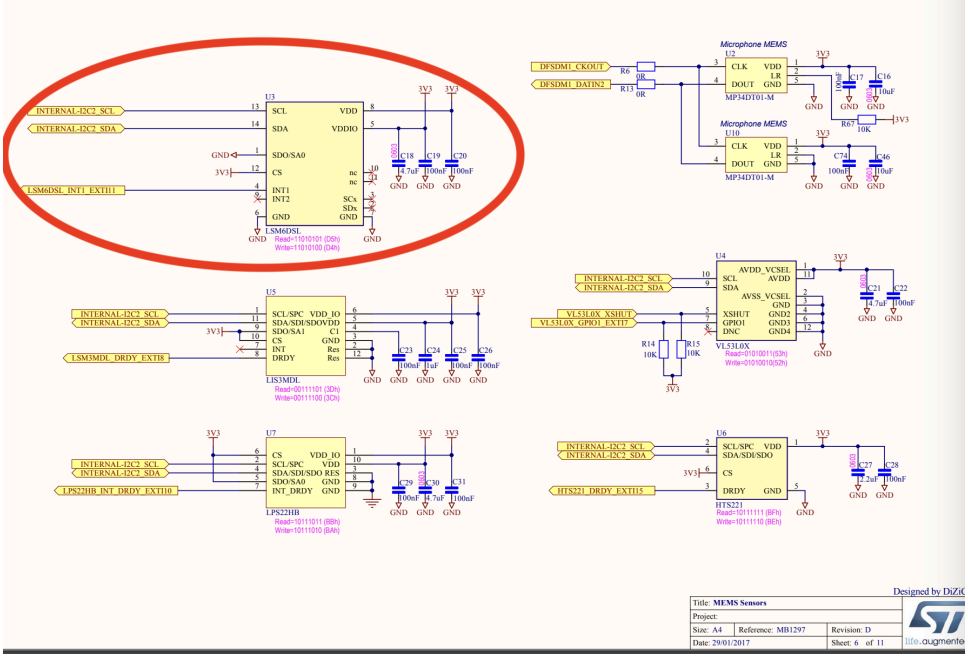
`void i2c_init();`
Configure and enable the I2C2 peripheral (it is connected to pins `PB10` and `PB11`) to communicate in master mode, at a standard BAUD RATE i.e., 400 kHz or less (You will need to read the I2C timing characteristics (Section 39.4.3 & 39.4.5) to determine how to set the baud rate). This function also should configure the appropriate pins on the MCU so they are connected to the correct I2C peripheral rather than operating as GPIO pins using the I/O Alternate function GPIO peripheral config (*STM32 Datasheet* - *Table 17*).

`uint8_t i2c_transaction(uint8_t address, uint8_t dir, uint8_t* data, uint8_t len);`
Perform a transaction to Write/Read bytes to/from the microcontroller (primary) to any I2C peripheral (secondary) with the secondary `address` parameter. If the `dir` parameter is `0` it is *writing* to the secondary, if it is `1` it is *reading* from the secondary. This is a blocking function: it should return only when the entire transaction has been completed.

Step 2: Implement the accelerometer driver

Relevant datasheets: LSM6DSL datasheet (particularly Section 6 - I2C)
Files that will be graded: `lsm6dsl.h, lsm6dsl.c`

Debugging support: We will provide access to a *digital logic analyzer* in the lab so you can debug your I2C bus to see if the transactions are working properly. More details about that will be provided in class.



`void lsm6dsl_init();`
Configure and enable the LSM6DSL. Refer to section 4.1 of the LSM6DSL application note.

`void lsm6dsl_read_xyz(int16_t* x, int16_t* y, int16_t* z);`
Read the current X,Y, and Z acceleration data from the accelerometer. This data will reflect the orientation and movement of the hardware. Refer to the data sheet for the units that these values are reported in.

Step 3: Implement the PrivTag application

The PrivTag application will incorporate both the code that you wrote for Project 1 and the drivers that you have written for this project. The PrivTag application will detect if the user has lost the tag if it is not moving for 1 minute. If it has, it will begin blinking the LEDs to indicate what bit has been lost. If the device is moving, it should immediately exit lost mode.

Lost device detection should be implemented in C by reading the raw X,Y, and Z raw acceleration values from the accelerometer, and processing this data to determine when the device has not moved for at least one minute. There are many algorithms you could use to detect if the device has stopped moving. We encourage you to experiment and see which works well. We will not grade you on what detection algorithm that you use, just that it activates when we leave your device still for one minute, and not when the device is moving.