RoboJackets Firmware Training Week 4 Lab Guide

Varun Madabushi, Joe Spall, edited by Andrew Roach

August 11, 2021
v1.1

Contents

1 Background

1.1 Topics

The important topics being discussed this week in lab include communication protocols (specifically I2C), register mapping on datasheets, and sensors.

1.2 Premise

The lab premise is to use a training board with a mounted MPU-6050 sensor to detect g-force. This g-force can be used to determine what angle the training board is held. This angle will then be used to determine which of the 5 controllable LEDs will light up on-board, acting like a bubble level indicator.

1.3 Inertial Measurement

Inertial Measurement Units (IMUs) are a type of sensor that respond to changes in force applied to them. These consist of Accelerometers (which measure linear force) and Gyroscopes (which measure rotational velocity). These sensors are made up of Micro Electromechanical Systems (MEMS) which are precisely tuned vibrating microstructures in the chip. Forces change the frequency of these vibrations, and that information is translated into electrical signals. These sensors are commonly used in robotics for measuring the pose (position and orientation) and motion of the robot in space.

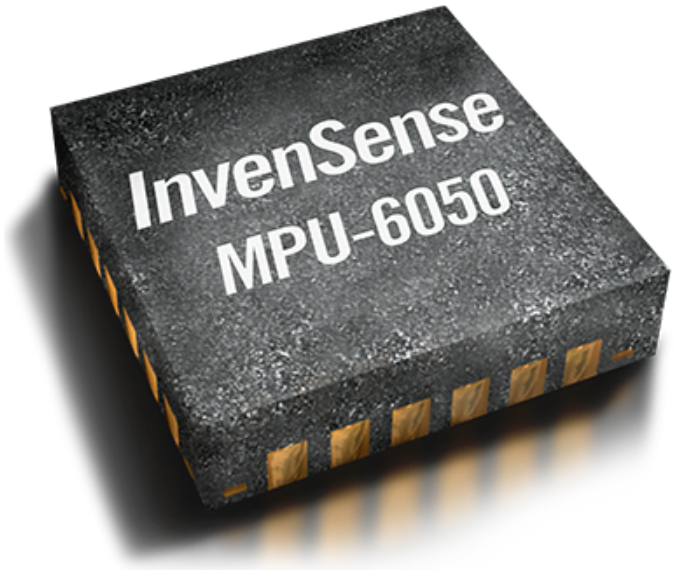

Figure 1: The MPU-6050 from TDK

For this lab we will be interfacing with the MPU-6050 Inertial Measurement Unit. Further information about the section can be found in Section 3.1.

2 Materials

- Firmware training board
- Mini usb cable
- MPU-6050 breakout board

3 Relevant Information

3.1 MPU-6050

The MPU-6050 is an Inertial Measurement Unit (IMU) which is mounted to the training board by means of a breakout board (the smaller PCB). In order to control it, first familiarize yourself with the register map found here (click on Section 3 from the Table of Contents).

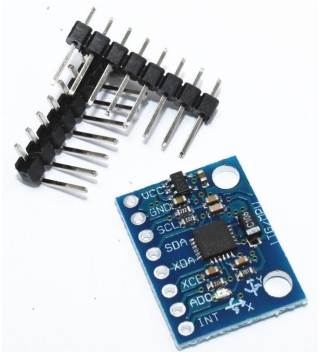

Figure 2: Picture of the MPU-6050

As per the datasheet, communicating with this involves interacting with the internal read/write pointer of the microcontroller.

If you wish to write to a particular register, the following steps must be taken:

1. Begin a transmission (`Wire.beginTransmission(DEV_ADDR)`).
2. Send the address of the register you wish to write to (`Wire.write(REG_ADDR)`).
3. Send the value you wish to write to the register (`Wire.write(VALUE)`).
4. End the transmission (`Wire.endTransmission()`).

A similar process can be followed for reading from registers:

1. Begin a transmission (`Wire.beginTransmission(DEV_ADDR)`).
2. Send the address of the register you wish to start reading from (`Wire.write(REG_ADDR)`).
3. End the transmission (`Wire.endTransmission()`).
4. Request a number of bytes you want to receive (`Wire.requestFrom(DEV_ADDR, BYTES, true)`).
5. Read in the bytes, repeating for the number of bytes to be read (`Wire.read()`)

3.2 Important MPU-6050 Registers

The particular registers we use in this lab are the following:

- `PWR_MGMT_1` - Turns on device
- `WHO_AM_I` - Identifies device
- `ACCEL_XOUT_H` - Accelerometer X, high 8 bits
- `ACCEL_XOUT_L` - Accelerometer X, low 8 bits
- `ACCEL_YOUT_H` - Accelerometer Y, high 8 bits
- `ACCEL_YOUT_L` - Accelerometer Y, low 8 bits
- `ACCEL_ZOUT_H` - Accelerometer Z, high 8 bits
- `ACCEL_ZOUT_L` - Accelerometer Z, low 8 bits
- `ACCEL_CONFIG` - contains scaling between bits and g-force

A good practice for readability is to reference these registers by name in the code. To do this, first fill out `#define` statements which will represent the hexadecimal values of the registers as readable words. For example, the statement `#define PWR_MGMT_1 0x6b` gives the text `PWR_MGMT_1` a value of `0x6b` in the code.

3.3 `Wire` Library

The library we will be using for I2C communication is the `Wire` Library. You can refer to the reference page here for understanding the necessary functions along with examples.

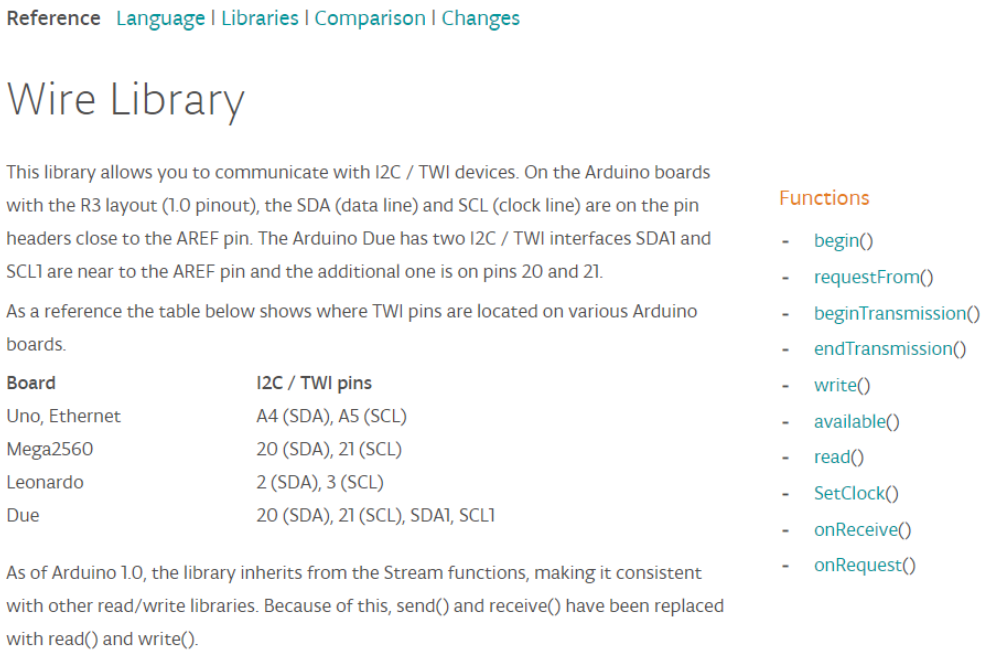

Figure 3: `Wire` Library on the Arduino website

4 Lab Objectives

Go to the RoboJackets Github here. Open up `code` > `Week4` > `Week_ 4_Template` > `Week_4_Template.ino` in the Arudino IDE.

4.1 Task 1 - Get the IMU to turn on

In `Week_4_Template.ino`, look at the end of the `setup()` function.

1. Set the IMU to 'Awake' by writing 0 to `PWR_MGMT_1`
   - The register names for a few relevant registers have been predefined in the code template. Please take a moment to look through these at the top of the code file. They can be accessed directly by name.
   - Page 41-42 of the MPU-6050 datasheet shows the various fields of this register.
   - Writing 0 to this register sets all bits such that the device is not reset, the sleep mode is disabled, the chip does not cycle on-off, the temperature sensor is enabled, and the internal oscillator is used.
   - A refresher on reading and writing from external devices with I2C can be found in Section 3.1.
2. Run a "Who Am I" test to ensure communication.
   - A "Who Am I" test asks the chip to return a known value so we can ensure data is transferring smoothly.
   - Read from the `WHO_AM_I` register and use Arduino's `Serial.print()` to print the result to the console. This should return the value specified in the register map.

4.2 Task 2 - Read raw data

In `Week_4_Template.ino`, look at the beginning of the `loop()` function.

1. We would like to read the X, Y, and Z accelerations from the IMU, so we must request these values from the sensor with `Wire.requestFrom()`.
   - Each acceleration value is represented by 16 bits spread across 2 8-bit (1 byte) registers.
   - We want to read from 6 registers starting from `ACCEL_XOUT_H` (`0x38`).
   - Refer to Page 30 of the MPU-6050 datasheet for more information.
2. Now, receive the individual bytes and reconstruct them into sensor readings.
   - The High byte (e.g. `ACCEL_XOUT_H`) corresponds to the first 8 bits, while the low byte (e.g. `ACCEL_XOUT_L`) corresponds to the last 8 bits.
   - These must be combined into a 16-bit data type (such as `int16_t`).
   - Shift the high byte over by 8 bits and bitwise-OR it with the low byte.
3. Lastly, we must convert the `int16_t` values into decimal accelerations in Gs.
   - Data usually comes in units of least significant bits (LSBs), which are divisions of the Full-Scale Range into 2^{BIT_DEPTH} number of divisions.
   - The `AFS_SEL` register changes the Full-Scale range of the sensor and thus sets the conversion factor between LSBs and Gs.
4. Print the resulting G-Force values to the console to make sure they are reasonable. If you have the board, place it stationary and flat on the table, `x_g` and `y_g` should be close to 0, and `z_g` should be close to 1.

4.3 Task 3 - Use acceleration data to light up LEDs

In `Week_4_Template.ino`, look at the end of the `loop()` function.

1. Instantiate a struct to hold the acceleration vector.
   - Construct an instance of the vector struct by creating a new `tb::Vector3D`.
   - Assign each of the fields `x_g, y_g, z_g` of your new `Vector3D` variable using the values calculated in Section 4.2 step 3.
2. Light up LEDs corresponding to value in the struct.
   - A method, `setBubbleIndicator(tb::Vector3D vector)`, is also provided for you.
   - Pass the `tb::Vector3D` struct into this method, and your LEDs should light up corresponding to the angle.
   - NOTE: If you are using the simulator, this step will only work if you set the y-axis acceleration to 0 and set the z axis acceleration to 1G (about half the positive range).

5 Troubleshooting

- Remember to use the `#define` macros, especially when referring to device addresses. Don't re-invent the wheel!
- If you don't see any input/output from the IMU, remember to use `Wire.beginTransmission()` and `Wire.endTransmission()` when sending and `Wire.requestFrom()` when receiving.
- By default, the `AFS_SEL` register has a value of 0; you can see what value this corresponds to on page 30 of the IMU documentation. You can use the `SCALE_FACTOR_2G` macro to scale the IMU outputs by the proper amount.

6 Egg

There is a virtual egg in this document. If you find it, contact the authors of this document with proof of what you think the virtual egg is and you will receive one (1) real egg in return.