Introduction

In this lab, you will write and test code on the ESP32 that communicates with the MPU-6050 9-axis Motion Processing Unit (MPU) using the I2C protocol.

The MPU-6050 has the following features:

- supports I2C communications at up to 400 KHz (2.5 us/bit),
- contains a solid-state micro-electrical-mechanical system (MEMS)-based accelerometer, gyroscope, and magnetometer, as well as a temperature sensor,
- the gyroscope offers a maximum sample rate of 8 KHz (when not using the built-in low pass filter) and the accelerometer offers a maximum sample rate of 1 KHz,
- its accelerometer can measure acceleration in the X-, Y-, and Z-axes over ranges of +/-2g, +/-4g, +/-8g, and +/- 16g, and
- its gyroscope can measure rotational speeds in X-, Y-, and Z-axes over ranges of +/-250, +/-500, +/-1000, and +/-2000 degrees per second,
- conveys all sensor measurements in 16-bit precision, and
- contains 82 8-bit I2C registers for configuration and sensing.

Although the MPU-6050 has multiple sensors, we will only use the accelerometer for this lab. The requirements are to read the X-, Y-, and Z- acceleration and compute the angle at which the sensor is oriented with respect to the pull of gravity. Your code must print the computed values to the serial terminal at a rate of one per second.

Accessing MPU-6050 Registers

The MPU-6050 contains 82 registers. All of these registers are accessed through a single I2C device address, which is 0x68. For this reason, we make a distinction between an *device address* and an *MPU-6050 register address*.

Writing to the *device address* sets the MPU-6050's address register, which allows the user to specify the target *MPU-6050 register address* from which subsequent reads and writes are performed.

Thus, accessing a register on the MPU-6050 requires a two-step process:

(1) initiate an I2C write transaction with device address 0x68 and value corresponding to the address of a targeted MPU-6050 register,
(2) to change the value of the MPU-6050 register, transmit an additional byte with the desired value,
(3) to read the contents of the MPU-6050 register, end the write transaction from step 1 and then initiate a new I2C read transaction at device address 0x68, to which the slave will respond by

sending the value of the requested MPU-6050 register. During this transaction the master read additional registers by pulsing the SCL for additional sets of eight clock pulses.

For example, to write the value of 0 into MPU-6050 register 0x6B, use the following transaction:

cycles 0-7	cycle 8	cycles 9-16	cycle 17	cycles 18-25	cycle 26
master sends 0x68 << 1	slave sends ACK	master sends 0x6B	slave sends ACK	master sends 0	slave sends ACK

To read register 0x6B:

Transaction 1:

cycles 0-7	cycle 8	cycles 9-16	cycle 17
master sends 0x68 << 1	slave sends ACK	master sends 0x6B	slave sends ACK

Transaction 2:

cycles 0-7	cycle 8	cycles 9-16	cycle 17
master sends 0x68 << 1	slave sends ACK	slave sends contents of register 0x6B	master sends ACK

When reading, the master can read multiple consecutive registers starting with the register address written during the initial I2C write (in transaction 1). This is performed by continuing to pulse the SCK after the first register is received. This ends when the master sends a not acknowledge (NACK) signal and a stop signal.

Relevant Registers

For this lab you will need to access the registers listed in Table 1.

Table 4: MPU-6050 accelerometer data registers.

register address	purpose
0x3B	bits 15:8 of X acceleration
0x3C	bits 7:0 of X acceleration
0x3D	bits 15:8 of Y acceleration
0x3E	bits 7:0 of Y acceleration
0x3F	bits 15:8 of Z acceleration
0x40	bits 7:0 of Z acceleration

Register 0x6B (Power Management Register):

bit 7	bit 6	bit 5	bit 4	bit 3	bits 2:0
device_reset	sleep	cycle	-	temp_dis	clksel

For this lab, you must set this register to 0 on startup.

Accessing the SCL and SDA pins

In this lab we will use the Arduino IDE's I/O abstraction, i.e. the `pinMode()` and `digitalWrite()` functions, as opposed to low-level programming of the esp32 GPIO matrix and IO MUX. Also, assign constants to assign the pins we use for the I2C channel to the MPU-6050, pins 21 and 22.

```
1 #define SCL 1
2 #define SDA 2
```

Next, set the pin mode of `SCL` to `OUTPUT_OPEN_DRAIN`:

```
1 pinMode(SCL, OUTPUT_OPEN_DRAIN);
```

You can pulse the clock on this pin using the following code:

```
1 digitalWrite(SCL, HIGH);
2 delayMicroseconds(10);
3 digitalWrite(SCL, LOW);
```


The Arduino library does not support an `INPUT_OPEN_DRAIN` mode, so you will need to change the mode of `SDA` to `INPUT_PULLUP` when receiving on this pin and to `OUTPUT_OPEN_DRAIN` when transmitting on this pin.

```
// write example
pinMode(SDA, OUTPUT_OPEN_DRAIN);
digitalWrite(SDA, LOW);

// read example
pinMode(SDA, INPUT_PULLUP);
digitalRead(SDA);
```

Conversion from Acceleration to Angle

The acceleration measurements are conveyed in 16-bit twos complement format, giving a dynamic range of -32768 to 32767. This range corresponds to the range of measurable accelerations under the current setting, which is +/- 2 g by default. This means that the measurement resolution is `4/65536 = 1/16384 g/ulp` (ulp = unit in the last place).

Acceleration is measured in the X-, Y-, and Z- axes. When the board is not moving, you can expect the norm of all accelerations to equal 1 g, since the portion of gravity's influence will be distributed across all three axes in proportions that depend on the tilt angle of the sensor.

In this lab, your objective is to use these measurements to determine the tilt angle of the sensor relative to its X-Y, X-Z, and Y-Z planes, as shown below.

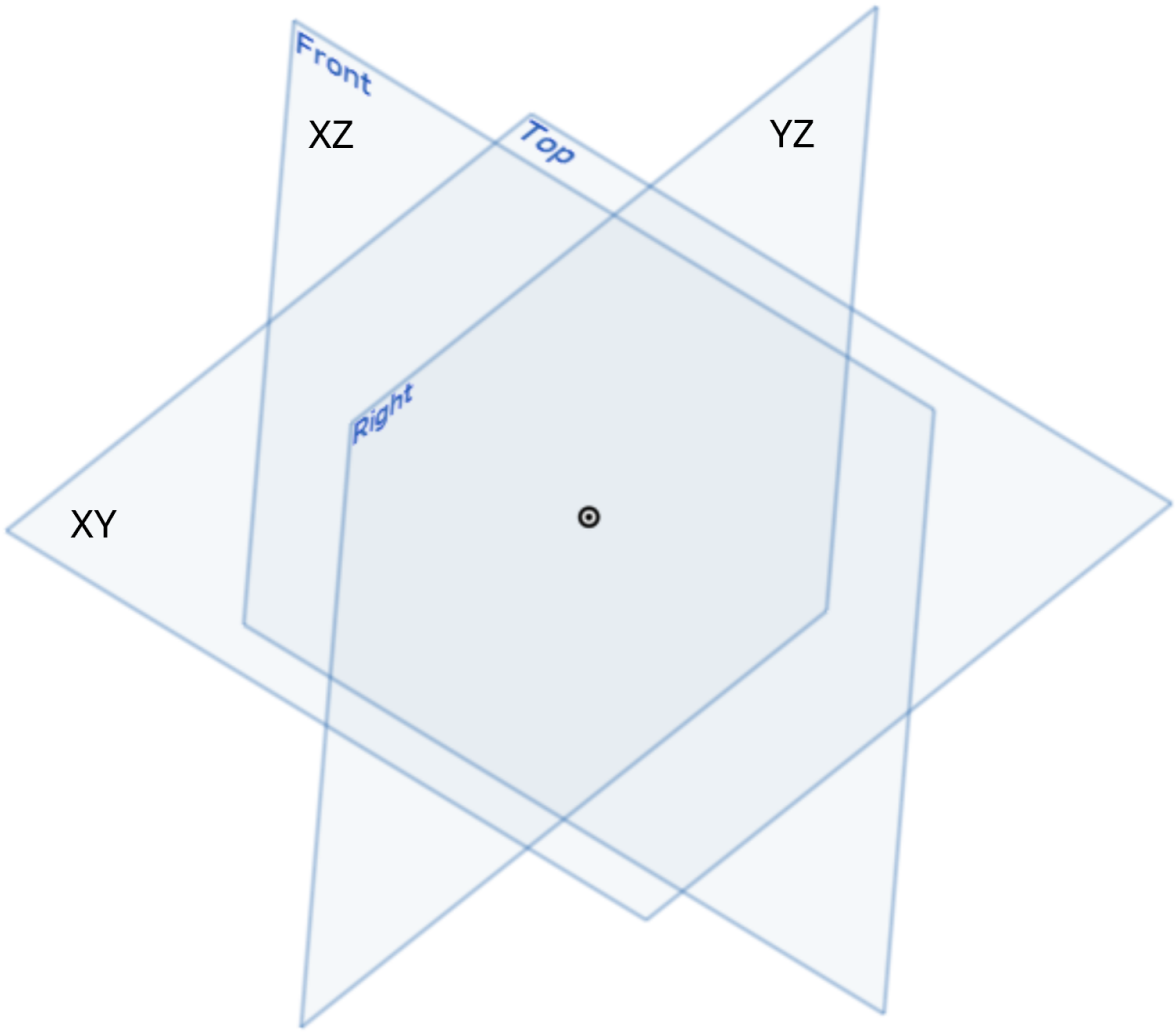

Figure 1: Planes in 3D space

To calculate the angle of the sensor in each of the 2D planes, you need only use the `atan2()` function, which converts two orthogonal magnitudes into an angle. For example, use `atan2(accel_x, accel_y)` to compute the angle in radians that the X-Y plan is tiled, relative to vertical (pointing up). You can multiply this value by the built-in constant `RAD_TO_DEG` to convert to degrees.

If sitting on a desk, the XY-plane angle would register as -90 degrees, since it is 90 degrees rotated relative to the up direction, while the XZ- and YZ- planes should be 0 degrees since they are pointed up.

Printing

Your code should print the angular value of the XY, XZ, and YZ planes to the console every second. To enabling printing, add the following to your `setup()` function:

```
1 Serial.begin(115200);
```

To print a message, use the following function:

```
1 Serial.printf("XY angle: %0.2f degrees\n",angle*RAD_TO_DEG);
```

Submitting Your Code

When you are ready to turn in your code for grading, upload your INO file to Blackboard. If you have multiple files, ZIP them together.

Rubric

To receive credit, you will need to demo your project to the TA. Partial credit will be assigned using the following rubric:

- I2C write to MPU-6050 (40 points):
- (5) start bit: pull SDA low while SCL is high
- (5) change SDA while SCL is low
- (4) pulse SCL with each address bit in 7 cycles (0x68)
- (4) pulse SCL with R/W bit set to 0
- (4) read acknowledgement after address and print message on NACK
- (4) send register address in 8 cycles (0x6B)
- (4) read acknowledgement after address and print message on NACK
- (5) send data to write in 8 cycles
- (5) stop bit: pull SDA high while SCL is high
- I2C read from MPU-6050 (35 points):
- transaction 1:

- (0) start bit: pull SDA low while SCL is high
- (0) change SDA while SCL is low
- (0) pulse SCL with each address bit in 7 cycles (0x68)
- (0) pulse SCL with R/W bit set to 0
- (0) read acknowledgement after address and print message on NACK
- (0) send register address in 8 cycles (0x6B)
- (0) read acknowledgement after address and print message on NACK
- (5) stop bit: pull SDA high while SCL is high
- transaction 2:
- (5) start bit: pull SDA low while SCL is high
- (5) change SDA while SCL is low
- (5) pulse SCL with each address bit (0x68)
- (5) set R/W to 1
- (5) read acknowledgement after address and print message on NACK
- (5) pulse SCL while reading data bits
- Measuring angle (25 points)
- (10) Read and convert accel_x, accel_y, and accel_z to XY, XZ, and YZ tilt
- (5) Repeat every 1 second
- (10) Demo

