ROBOTICS CLUB
TRIBHUVAN UNIVERSITY
PULCHOWK CAMPUS

LOCUS 2021
18th National Technological Festival

Hardware Fellowship

DAY 5 AND 6

Overview

- Data Communication
- Communication Protocols

I2C

- Interfacing Bluetooth Module with Arduino
- Interfacing MPU6050 with Arduino

Data Communication

- Communication between two devices
- Transfer data from one device to another

Things to consider....

- Representation of Data in device's memory
  - Int is stored 2 or 4 bytes
  - Strings are sequency of bytes
- Error Detection and Recovery
  - Parity Bits, CRC etc.
- Flow Control
  - Acknowledgements
- Line Coding
  - Convert 1's and 0's to appropriate voltage levels

Some More Things to Consider

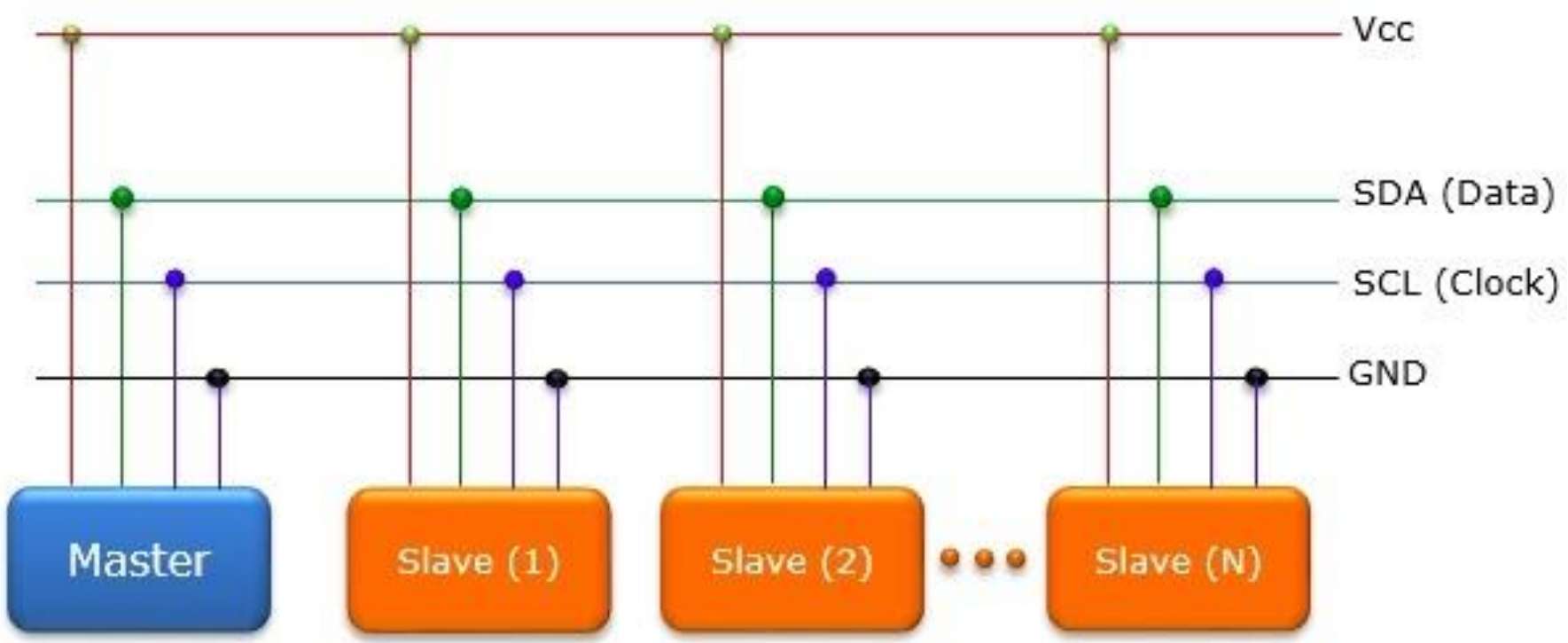
- Serial vs Parallel Communication
- Bus vs. Point to Point
- Speed of Data Transfer
- Synchronous vs. Asynchronous

I2C PROTOCOL

I2C Protocol

- Stands for **Inter-Integrated Circuit**.
- Widely Used Protocol for Short Distance Communication.
- Also known as a Two Wire Interface
- Uses 2 bi-directional open-drain lines, pulled up with resistors.
  - **SDA (Serial Data):** Transfer of data through this pin.
  - **SCL (Serial Clock):** Carries the clock signal.

- There are many speed modes of I2C available where the most commonly used mode has a speed of 100kBit/sec.

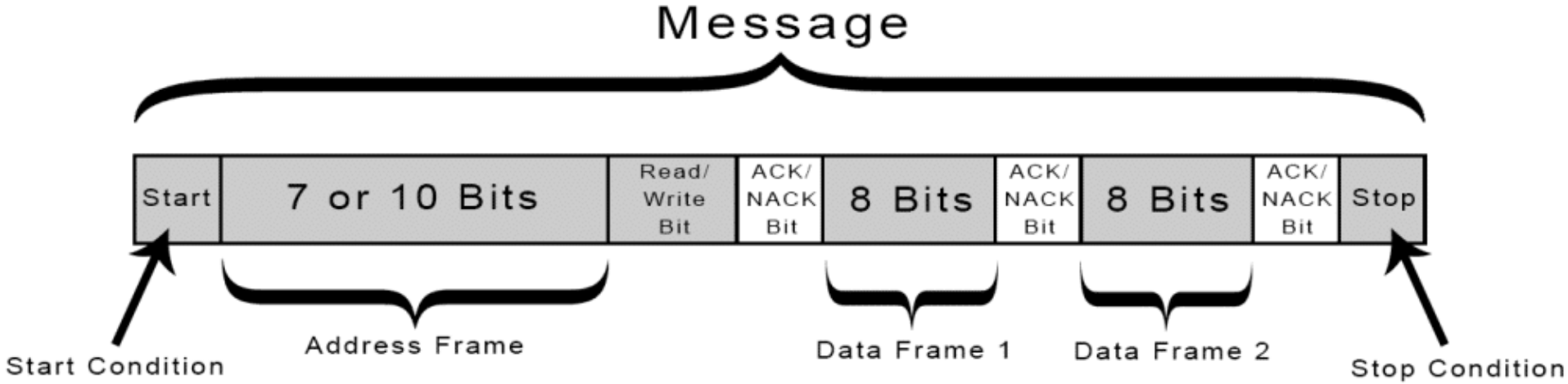

Vcc
SDA (Data)
SCL (Clock)
GND
Master
Slave (1)
Slave (2)
Slave (N)


VCC
GND
SCL
SDA
XDA
XCL
ADO
INT


UNO

Examples

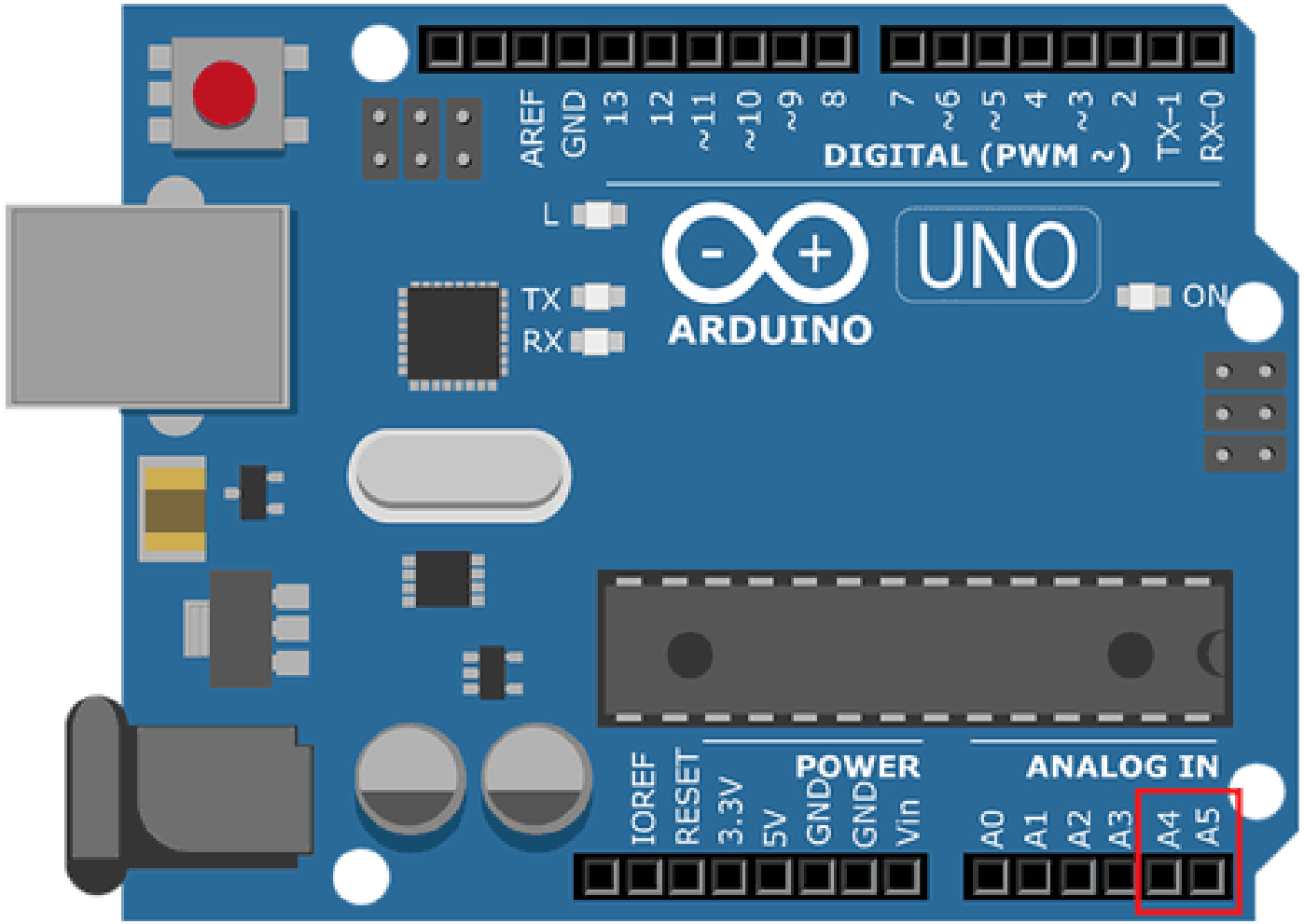
How I2C works?



Start Condition: The SDA line switches from a high voltage level to a low voltage level *before* the SCL line switches from high to low.
Stop Condition: The SDA line switches from a low voltage level to a high voltage level *after* the SCL line switches from low to high.
Address Frame: A 7 or 10 bit sequence unique to each slave that identifies the slave when the master wants to talk to it.
Read/Write Bit: A single bit specifying whether the master is sending data to the slave (low voltage level) or requesting data from it (high voltage level).
ACK/NACK Bit: Each frame in a message is followed by an acknowledge/no-acknowledge bit. If an address frame or data frame was successfully received, an ACK bit is returned to the sender from the receiving device.
Data frame: After the master detects the ACK bit from the slave, the first data frame is ready to be sent. The data frame is always 8 bits long, and sent with the most significant bit first. Each data frame is immediately followed by an ACK/NACK bit to verify that the frame has been received successfully.

I2C pins in Arduino Uno

A4 -> SDA pin
A5 -> SCL pin



Note : We will be using I2C protocol in next module

Inertial Measurement Unit (IMU)

IMUs can measures acceleration, specific force, angular position and rate of angular change & magnetic fields surrounding the device(in presence of Magnetometer).

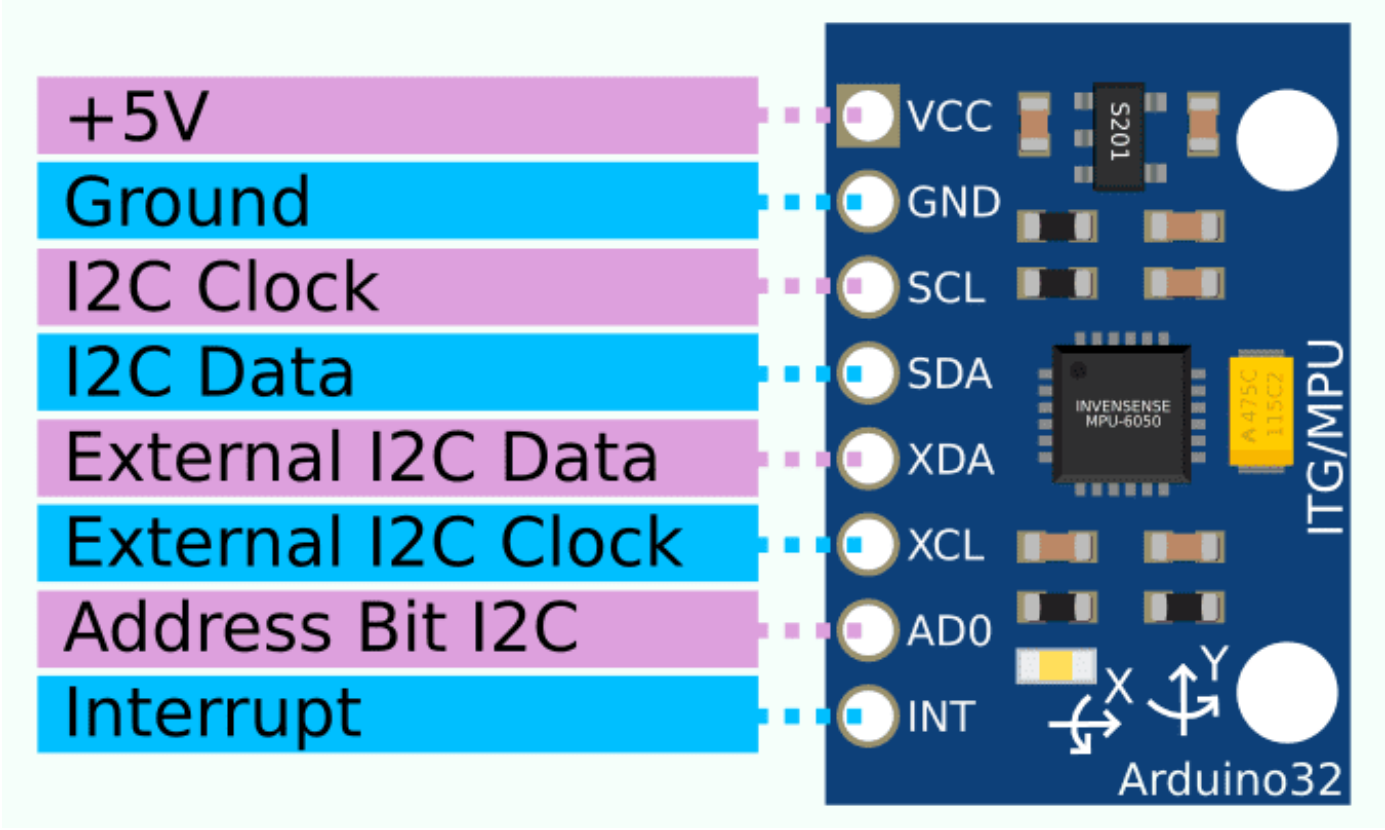
Each tool in an IMU is used to capture different data types:

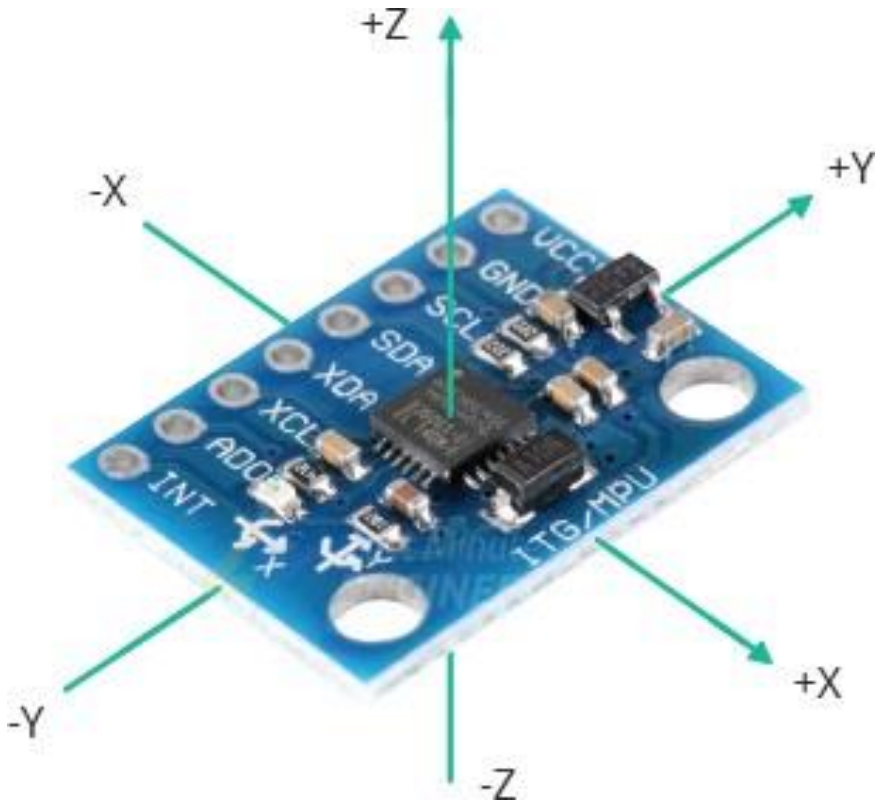
- **Accelerometer**: measures velocity and acceleration
- **Gyroscope**: measures rotation and rotational rate
- **Magnetometer**:(Optional) measures the strength and sometimes the direction of magnetic fields

Example of IMU is **MPU-6050.**

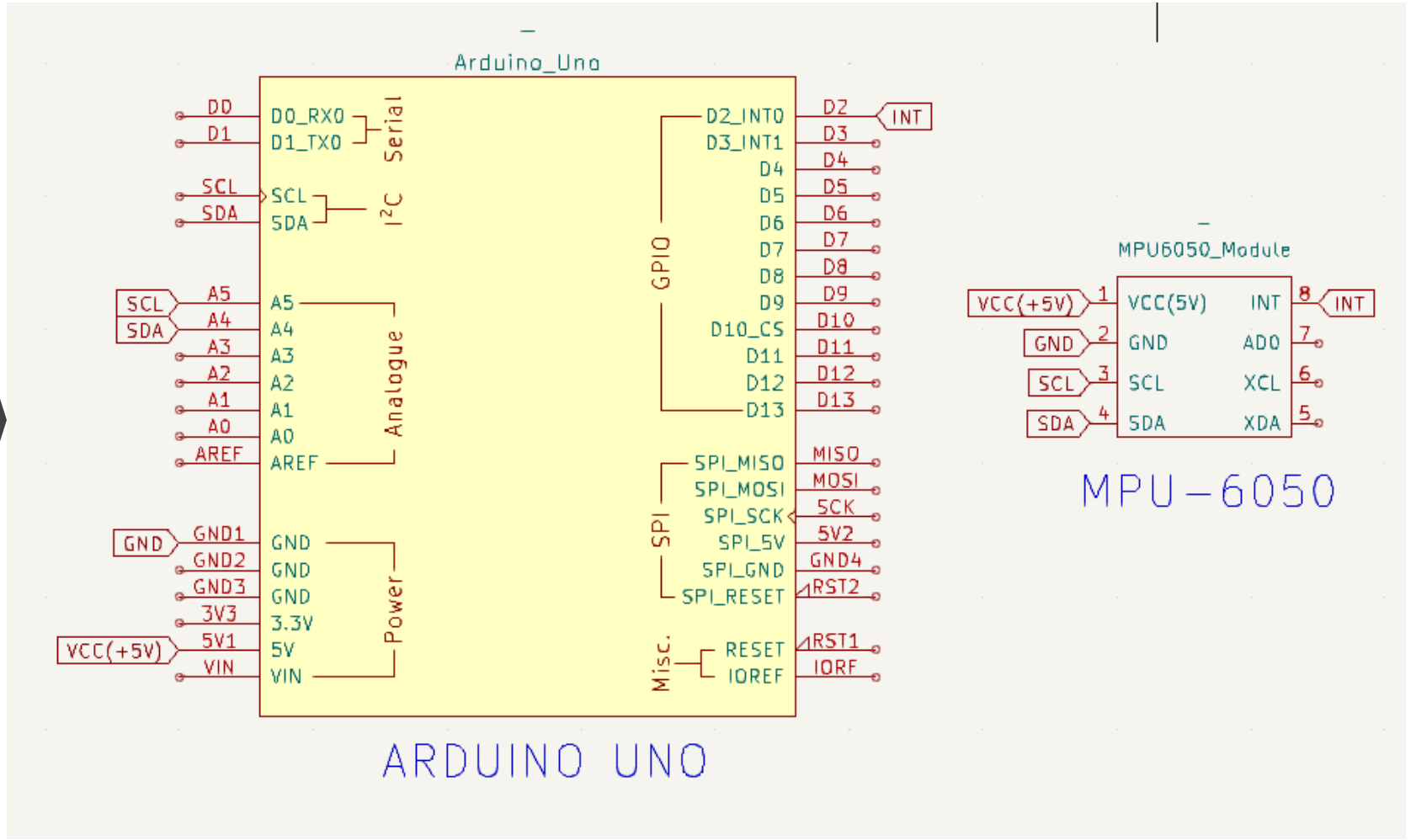
MPU 6050

- MPU-6050 is a low cost IMU device(Inertial Measurement Unit).
- MPU-6050 IMU consists both Accelerometer and Gyroscope.
- It is used in Robots and UAVs.




+Z
-X
+Y
VCC
GND
SCL
SDA
XDA
XCL
ADO
INT
ITG/MPU
-Y
+X
-Z

Interfacing MPU-6050 with Arduino UNO



```ino
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <Wire.h>

Adafruit_MPU6050 mpu;

void setup(void) {
  Serial.begin(115200);
  Serial.println("Adafruit MPU6050 test!");

  // Try to initialize!
  if (!mpu.begin()) {
    Serial.println("Failed to find MPU6050 chip");
    while (1) {
      delay(10);
    }
  }
  Serial.println("MPU6050 Found!");

  mpu.setAccelerometerRange(MPU6050_RANGE_2_G);
  mpu.setGyroRange(MPU6050_RANGE_250_DEG);
  mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);
  delay(100);
}

void loop() {

  /* Get new sensor events with the readings */
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);

  /* Print out the values */
  Serial.print("Acceleration X: ");
  Serial.print(a.acceleration.x);
  Serial.print(", Y: ");
  Serial.print(a.acceleration.y);
  Serial.print(", Z: ");
  Serial.print(a.acceleration.z);
  Serial.println(" m/s^2");

  Serial.print("Rotation X: ");
  Serial.print(g.gyro.x);
  Serial.print(", Y: ");
  Serial.print(g.gyro.y);
  Serial.print(", Z: ");
  Serial.print(g.gyro.z);
  Serial.println(" rad/s");

  Serial.print("Temperature: ");
  Serial.print(temp.temperature);
  Serial.println(" degC");

  Serial.println("");
  delay(500);
}
```

Find the Offset of Rotation Speed

- From the gyroscope readings, read the angular velocities
- Ideally, these values should be all zeros.
- Any such offset must be removed prior to using the velocity data
- How to remove such offset?

Getting the Angular Position from Velocity

- Recall: Integration!

$$\Delta\theta = \omega dt$$

$$\theta_{new} = \theta_{old} + \Delta\theta$$

Calculating the Position

- Need two more variables
  - One to keep track of time
  - One to keep track of our position till the current time

```
#include <Adafruit_Sensor.h>
#include <Wire.h>
#include <Adafruit_MPU6050.h>
#include <Servo.h>

Adafruit_MPU6050 mpu;

unsigned long prev = 0;
float position = 0;
```

Integration

```
/* Get new sensor events with the readings */
sensors_event_t a, g, temp;
mpu.getEvent(&a, &g, &temp);

unsigned long now = millis();
float dt = now - prev;
float velocity = g.gyro.z - 0.03;
Serial.print("Velocity: ");
Serial.println(velocity);
float dPosition =  dt * 1e-3 * velocity;

position = position + toDegrees(dPosition);
Serial.println(position);

prev = now;
delay(100);
```

Gesture Controlled Servo

- Control the servo angle by using the position data from the IMU
- Recall using the Servo with Arduino!

```
#include <Servo.h>

Servo myservo;
int pos = 0;

void setup() {
  myservo.attach(9);
}

void loop() {
  for (pos = 0; pos <= 180; pos += 1) {
    myservo.write(pos);
    delay(100);
  }
  for (pos = 180; pos >= 0; pos -= 1) {
    myservo.write(pos);
    delay(100);
  }
}
```

```
/* Get new sensor events with the readings */
sensors_event_t a, g, temp;
mpu.getEvent(&a, &g, &temp);

unsigned long now = millis();
float dt = now - prev;
float velocity = g.gyro.z - 0.03;
Serial.print("Velocity: ");
Serial.println(velocity);
float dPosition =  dt * 1e-3 * velocity;

position = position + toDegrees(dPosition);
Serial.println(position);
if ( position > 0 ){
  myservo.write((int)position);
}

prev = now;
delay(100);
```

Controlling Sensitivity: Mapping Values

- IMU gives position data from a wide range, e.g from –90 to 90 degrees
- Servo needs angle from 0 to 180 degrees
- We need to convert from [-90,90] -> [0,180], I.e, -90 degrees should be 0 and +90 should be 180
- Think of a function such that:

$$f(-90) = 0 \qquad f(90) = 180$$

Linear Mapping

- Recall: Two point form of equation of line (x1,y1) and (x2,y2)
- The x's are our input, the y's are our desired output at that input

$$y - y_1 = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1)$$

Linear Mapping

```
float toServoAngle(float x){
  float x1 = -90, y1 = 0, x2 = 90, y2 = 180;
  float slope = (y2 - y1)/(x2-x1);
  return y1 + slope * ( x - x1 );
}
```

Change the values of x1 and x2 to vary the sensitivity and the servo angles.

The Final Code

```
/* Get new sensor events with the readings */
sensors_event_t a, g, temp;
mpu.getEvent(&a, &g, &temp);

unsigned long now = millis();
float dt = now - prev;
float velocity = g.gyro.z - 0.03;
Serial.print("Velocity: ");
Serial.println(velocity);
float dPosition =  dt * 1e-3 * velocity;

position = position + toDegrees(dPosition);
Serial.println(position);

float servoAngle = toServoAngle(position);
Serial.println(servoAngle);
if ( servoAngle > 0 ){
  myservo.write(servoAngle);
}

prev = now;
delay(100);
```