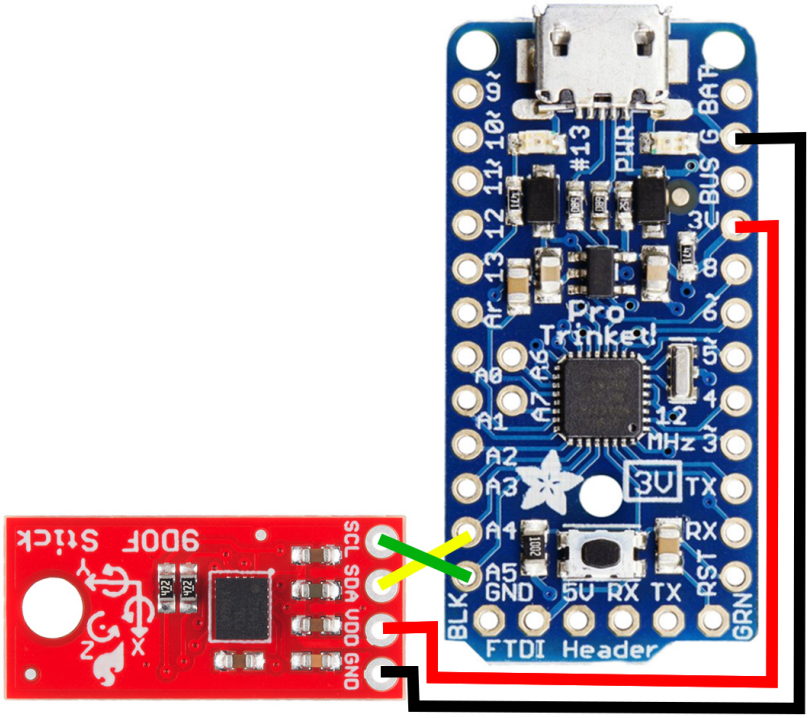
9DoF Integration

In this document, we will connect the 9 Degrees of Freedom (9DoF) breakout to our Arduino by means of a standard I^2C connection. We will then read some values from the device and print them to the serial line.

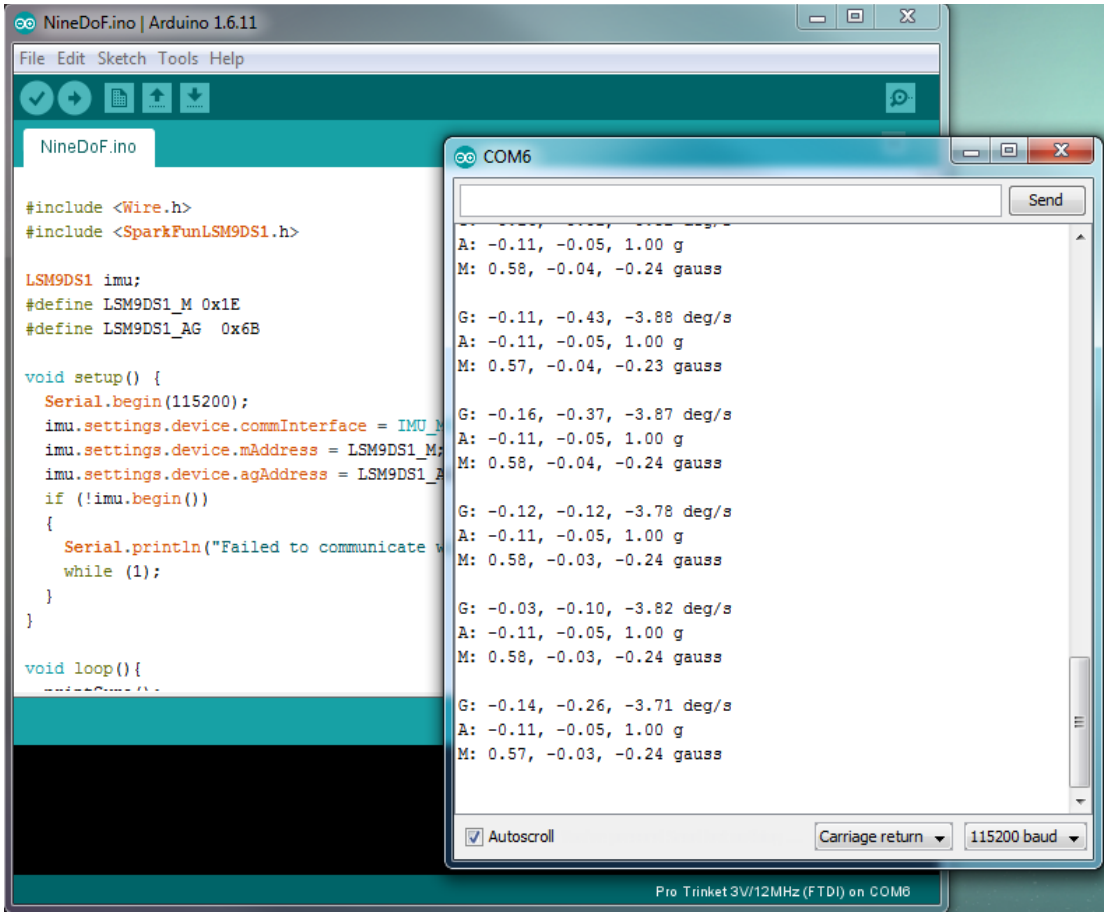
Part 1 Wiring
The 9DoF board communicates with the Arduino over an Inter Integrated Circuit (I^2C) connection. The SPI Bus has just two lines for data, it does not use a clock or chip select line.

Wire the 9DoF board to the Arduino as shown in the schematic below. Remember that your prototyping boards provide power and ground rails, so you don't need to bring power and ground all the way back to the pins on the Arduino to connect power and ground.



Part 2 9DoF Software

Next, open the sketch *RisingData/src/NineDoF/NineDoF.ino.* You can upload the sketch as written and then open the serial monitor to see the output of the 9DoF device. Feel free to move the device around and see how what effect that has on the sensor readings.



Appendix: Code Tour

```
#include <Wire.h>
#include <SparkFunLSM9DS1.h>
```

These lines import the I^2C and 9DoF libraries respectively. The 9DoF library in particular hides a great deal of the complexity of reading data from that device.

```
LSM9DS1 NineDoF;
#define MAG_ADDRESS 0x1E
#define AG_ADDRESS 0x6B
```

The first line define some objects we will reference later. LM9DS31 is the part number of the 9DoF device we are using. The second two lines are the address of the Magnetometer and Acceleration/Gryo Sensor used by the I^2C bus.

9DoF I²C:
This bus expects each connected device to have an address. This allows the bus to not require a chip select for each device and simplifies wiring. The 9DoF board is actually two devices connected together over an integrated I²C connection. This is why we reference the Magnetometer and Accelerometer/Gyroscope with different addresses.

```
void setup() {
  Serial.begin(115200);
  NineDoF.settings.device.commInterface = IMU_MODE_I2C;
  NineDoF.settings.device.mAddress = MAG_ADDRESS;
  NineDoF.settings.device.agAddress = AG_ADDRESS;
  if (!NineDoF.begin())
  {
    Serial.println("Failed to communicate with 9DOF.");
    while (1);
  }
}
```

Our setup loop first initializes the serial port for debugging. Next we set the 9DoF to be communicating over the correct interface. Next we set specify the addresses of the Magnetometer and the Accelerometer/Gyroscope. Finally, we attempt to start the device. If the device does not start properly, we enter an infinite loop to stop the program from continuing.

```
void loop(){
  printGyro();
  printAccel();
  printMag();
  Serial.println();
  delay(250);
}
```

Our main program loop print the values from each of the sensor, then a blank line, then waits 250 milliseconds before repeating. Most of the complexity here is in the print functions that are being called, which all have the same structures.

```
void printGyro(){
  NineDoF.readGyro();
  Serial.print("G: ");
  Serial.print(NineDoF.calcGyro(NineDoF.gx), 2);
  Serial.print(", ");
  Serial.print(NineDoF.calcGyro(NineDoF.gy), 2);
  Serial.print(", ");
  Serial.print(NineDoF.calcGyro(NineDoF.gz), 2);
  Serial.println(" deg/s");
}
```

First, we ask the sensor to take a reading. Then we print a little text of which sensor this is. Next we call a set of nested functions, let's deconstruct that line.

```
Serial.print(NineDoF.calcGyro(NineDoF.gx), 2);
```

The first function here references the raw sensor reading for gyroscope's x-axis.

```
Serial.print(NineDoF.calcGyro(NineDoF.gx), 2);
```

The next function takes that raw value, and converts it into a calibrated value with real units.

```
Serial.print(NineDoF.calcGyro(NineDoF.gx), 2);
```

The final function prints that calibrated value with 2 decimal places of accuracy.

This pattern is repeated for each axis of each sensor, until we finally end up printout out all 9 values that this sensor is capable of measuring:

Gyroscope(x,y,z) measured in **degrees/second**
Acceleration (x,y,z) measured in **g**
Magnetometer (x,y,z) measured in **gauss**