Multiplexer Setup

Dan Zilinskas

ECE 480 Team 8

Motion Capture For Runners

Contents

Introduction

A multiplexer, or MUX, is a device that selects one of several analog or digital input signals and then forwards the data or signal along that line into a single input. This means that multiple sensors can share the same data-line simultaneously and the multiplexer chooses which device to listen to based on the inputs of the selector bits on the MUX. Multiplexers are mainly used to increase the amount of data that can be sent over a network within a certain amount of time.

In its simplest form, a multiplexer will have two signal inputs, one control input, and one output. An example of this would be a home stereo unit that allows the user to switch between the audio of the CD player, the DVD player, or the cable television line. Multiplexers are also used in devices such as central processing units, as well as graphics controllers, both which are essential for personal computing. In these types of applications, the number of inputs on the multiplexer is usually a multiple of two, and the number of outputs is usually relatively small, usually one or two.

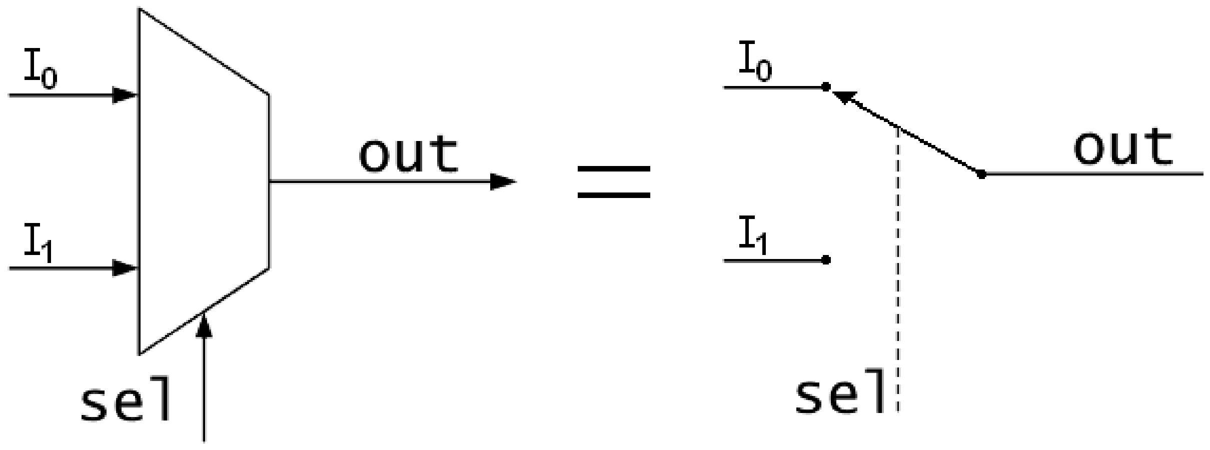


Figure 1) Basic Overview of how a Multiplexer works

Objective

This tutorial assumes that the user had at least fundamental knowledge of electronic circuit design, as well as basic knowledge of digital logic. This guide will give the user basic knowledge in the function and setup of a multiplexer, along with Arduino programming software, and IMU (Inertial Measurement Unit) setup.

- Give and explain the basic schematic of a Multiplexer
- Provide instructions on creating wiring connections between multiple IMUs and a multiplexer

- Provide information on connections between IMU, Multiplexer, and Arduino Uno
- Supply basic programming code to allow for functioning of Multiplexer with IMU and Arduino Uno

Multiplexer Schematic and Information

To start, it is important that the user understand the connections on a multiplexer. The multiplexer that we will be using in this tutorial is the 16-Channel Analog/Digital MUX Breakout board with the CD74HC4067 Multiplexer chip. This product is supplied by Sparkfun, and the board allows for easier configuration and connecting of devices for quicker assembly and setup.

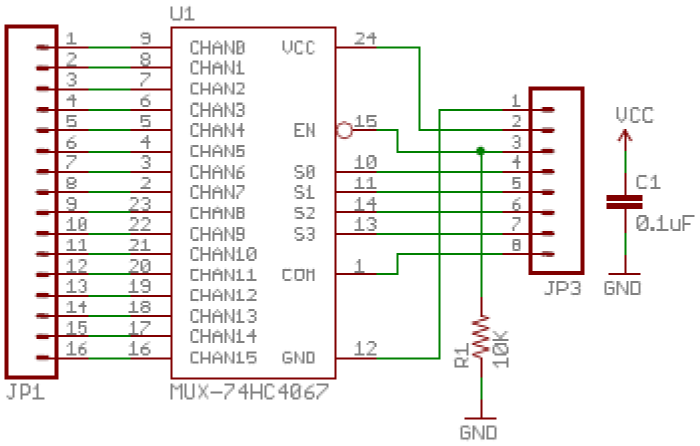


Figure 2) Multiplexer Schematic for Sparkfun's Multiplexer Breakout Board

As you can see in the figure above, the channel ports are all on the left side of the schematic. These ports, (C0…C15) are what allow you to attach multiple sensors/signals to your system. On the right side, you can see the selector ports (S0…S3) which are the bits that are adjusted to choose which channel the Multiplexer is listening to on the input side of the board. Pin 24 is VCC, which is the voltage input for the board, and it requires a voltage of 2V to 6V to operate successfully. Pin 12 is GND, which is the ground pin, and pin 15 is the EN pin, which is the enable on the multiplexer board.

IMU and Arduino

In order to actually get the Multiplexer working, it's important to have a sensor to connect, as well as a microcontroller. The sensor that we will be using in this tutorial is the MPU-9150 IMU, and the Arduino Uno microcontroller. Two sensors are necessary in order to ensure that the multiplexer is working correctly, and the Arduino will be the base programming language that will be used to write code to the Multiplexer.

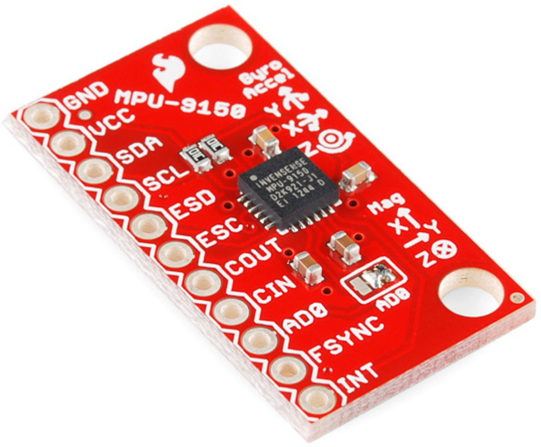


Figure 3) The MPU-9150 IMU

The figure above gives you a visual of what the IMU looks like and the connections that are available on the breakout board provided. The first thing necessary is to provide power and ground to the IMU, supplied to the VCC pin and GND pins respectively. The SDA pins and SCL pins are the data and clock lines respectively, and are necessary to provide a synchronized clock, as well as a data line for the IMU to communicate with the Arduino.

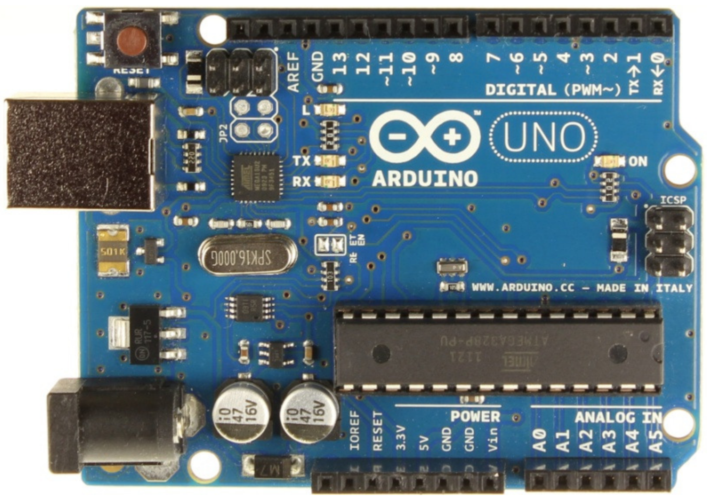


Figure 4) The Arduino Uno

Figure 4 gives an overview of the microcontroller that will be used in this tutorial. The important pins to keep in mind for this tutorial include the digital input pints, analog pins A4 and A5, which are the data and clock lines respectively.

Getting Started (Initial Setup)

To start with, we need to create our connector that will allow us to physically connect our multiplexer, our IMU, and our Arduino together. The circuit below gives a basic overview of the circuit necessary to connect two IMU sensors, along with the multiplexer and Arduino.

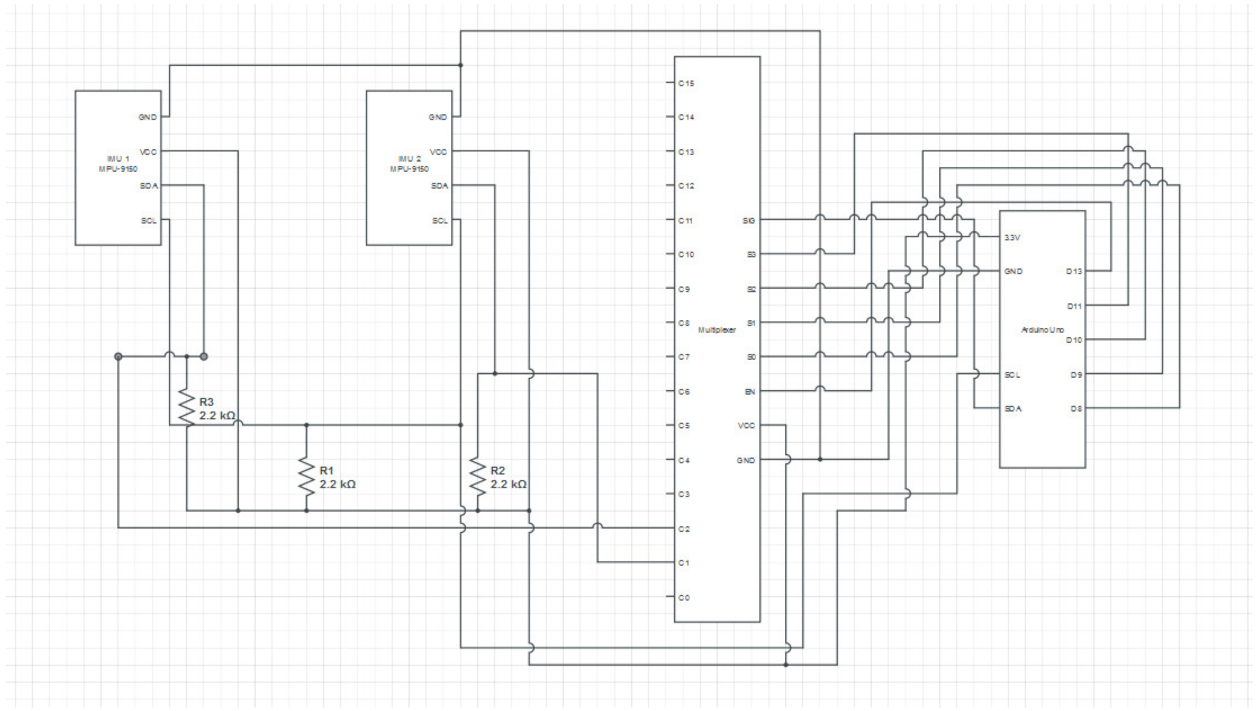


Figure 5) Multiplexer connection circuit

First, the connections between the IMUs include wires connecting VCC's on both IMUs as well as the multiplexer to the 3.3V output of the Arduino. Three 2.2K resistors are required in order to pull down the digital values of the data and clock lines of the IMUs. These resistors are placed between the SDA line and the VCC line of both IMUs and one of the resistors is placed in between the SCL line and the VCC line of the IMU. Two resistors are used for the data lines because two separate data lines are going to the multiplexer, while on the clock lines, the two clock lines coming from the IMU sensors are synchronized. The data lines from the IMUs are connected to the multiplexer channels, C1 and C2. The selector bits on the right side of the multiplexer (S0, S1, S2, and S3) are connected to the digital pins of the Arduino (D8, D9, D10, and D11) respectively. The En (Enable) pin of the multiplexer is also set to digital pin D13 on the Arduino. The SIG channel on the multiplexer is also connected to the SDA line of the arduino, which is pin A4 on the arduino. The clock lines of both IMUs are connected together and then sent to pin A5 on the arduino, otherwise known as SCL.

Code for the Multiplexer

In this tutorial, we are planning on using the Arduino microcontroller to connect all of our devices, including the IMUs and multiplexer. This also means that we are going to be using the Arduino programming software in order to code our multiplexer. First, we need to download the software off of the arduino website. First go to your internet browser, and then type in Arduino in your search bar. The arduino homepage should be the first result to appear on the search. Once on the website, click on the downloads tab near the top of the page. Depending on the computer or laptop operating system that you are using, download the version suited for your computer. Once downloaded, you may need to extract the files to a folder. I'd advise making the folder something easily accessible and easy to remember. The Arduino software is a stand-alone application, meaning it doesn't need to be installed in order to be run. Once everything is downloaded and extracted, start up the Arduino software.

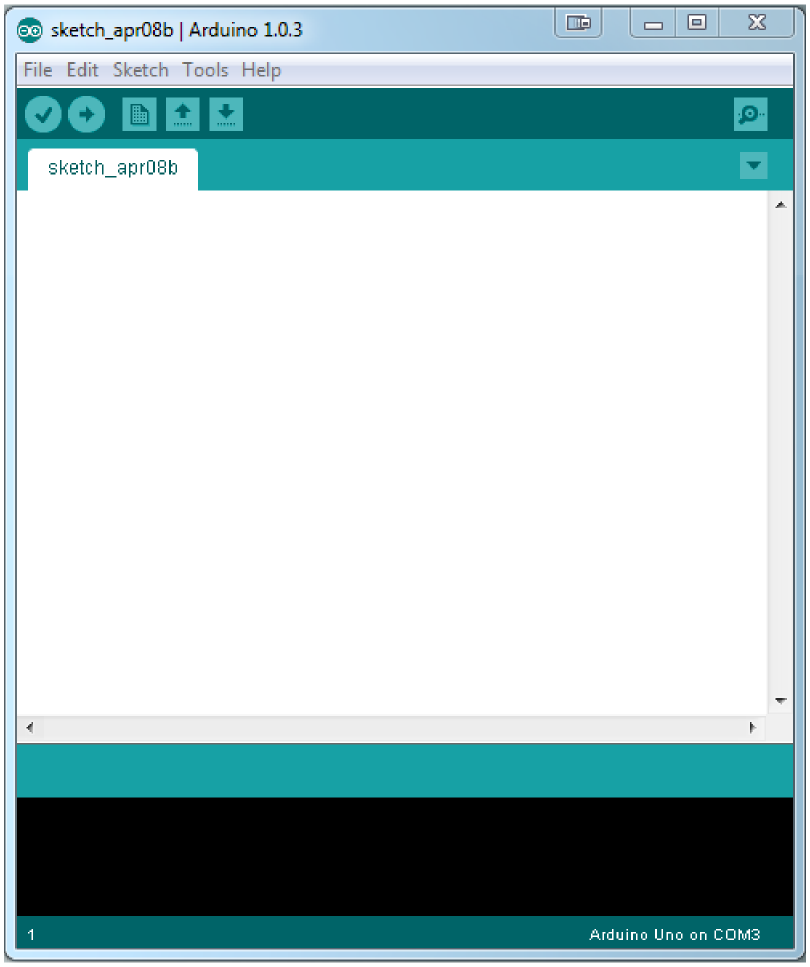


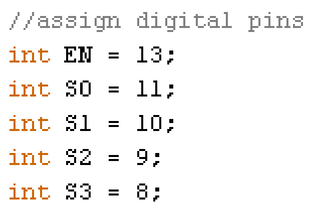
Figure 6) The starting window for Arduino

The opening window for Arduino should look like the window in the figure above. We will now begin coding for using multiple IMUs with the multiplexer.

Before writing our own code, we need to get the base IMU-9150 library code in order to acquire data from the IMU. This library can be found on the Sparkfun website by searching for MPU-9150 in their search bar, and then clicking on example code near the middle of the page, under the device description. It will then take you to a website called “github” and there should be a button that allows you to zip the IMU library and save it. We then need to take the library files of the IMU and put them in the library folder of the Arduino software. This is why I said to make the folder for Arduino easily accessible because we usually need to add many libraries that are not normally supported by Arduino. Once the code is successfully in the Arduino library folder, we can now open the MPU-6050 example code. The MPU-6050 code is essentially the same as the MPU-9150 code, except for the fact that we don’t have magnetometer functionality, but that is easily fixed with a few lines of code, and it is not necessary to get both sensors working. The figure below shows you what the example code should look like in the Arudino window, but remember this is only the first few lines of the code.

```
#include <SoftwareSerial.h>

#include <SPI.h>

// I2C device class (I2Cdev) demonstration Arduino sketch for MPU9150
// 1/4/2013 original by Jeff Rowberg <jeff@rowberg.net> at https://github.com/jrowberg/i2cdevl
//          modified by Aaron Weiss <aaron@sparkfun.com>
//
// Changelog:
//     2011-10-07 - initial release
//     2013-1-4 - added raw magnetometer output

/* ============================================
I2Cdev device library code is placed under the MIT license

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
===============================================
*/

// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE implementation
// is used in I2Cdev.h
#include "Wire.h"

// I2Cdev and MPU6050 must be installed as libraries, or else the .cpp/.h files
// for both classes must be in the include path of your project
#include "I2Cdev.h"
#include "MPU6050.h"
```

First, we need to assign digital pins that will be connected between the Arduino and the multiplexer. These pins will be the bits that control which channel the multiplexer is listening on. This is done with the code shown below in figure 7.

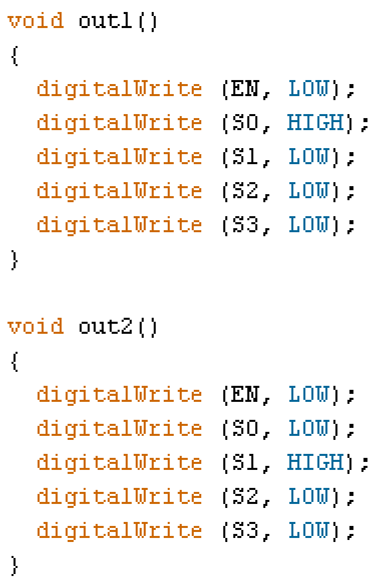


```
//assign digital pins
int EN = 13;
int S0 = 11;
int S1 = 10;
int S2 = 9;
int S3 = 8;
```

Figure 7) The initial digital pin setup for the selector bits

This figure shows that the enable pin is set to pin 13 on the arduino, and that selector bits 0, 1, 2, and 3 are connected to pins 11, 10, 9, and 8 respectively, on the Arduino. The pins that select as your selector pins don’t really matter, as long as the pins you select on the Arduino are the digital pins.

Next we need to create functions that will change whether each selector bit is set to high or low, in order to change the address that the Arduino is listening to. For example, a byte of “0000” would communicate with channel C0, and a byte of “0100” would communicate with channel C8 on the multiplexer. Figure 8 below shows how to set up the initial functions for each of the two IMU channels.

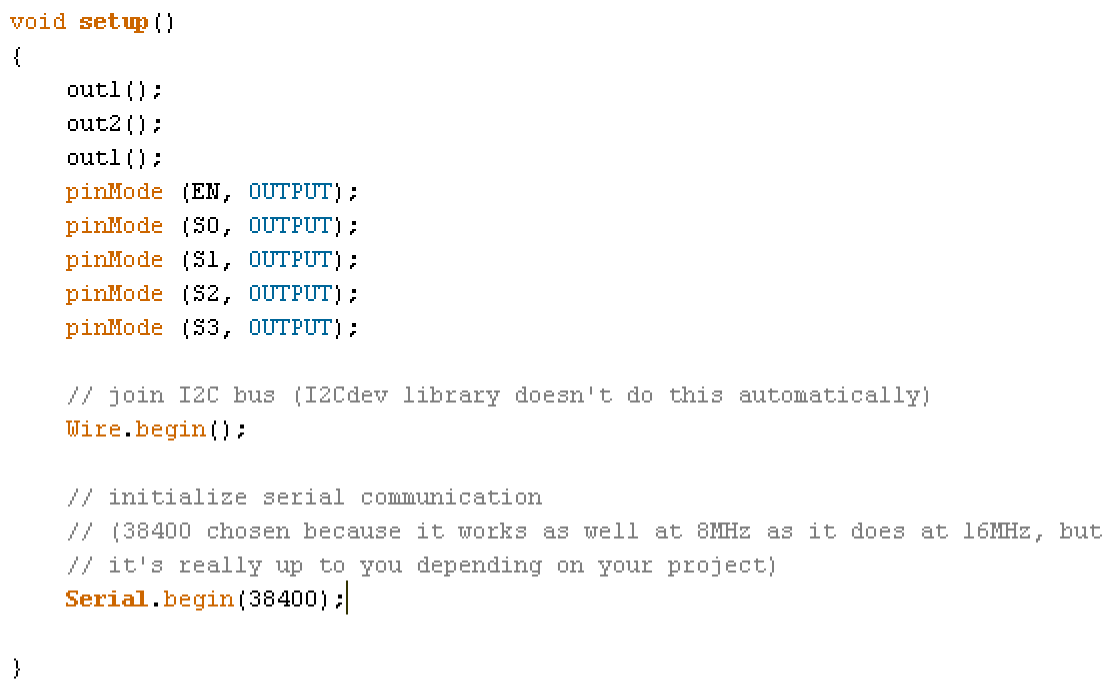


```
void out1()
{
  digitalWrite (EN, LOW);
  digitalWrite (S0, HIGH);
  digitalWrite (S1, LOW);
  digitalWrite (S2, LOW);
  digitalWrite (S3, LOW);
}

void out2()
{
  digitalWrite (EN, LOW);
  digitalWrite (S0, LOW);
  digitalWrite (S1, HIGH);
  digitalWrite (S2, LOW);
  digitalWrite (S3, LOW);
}
```

Figure 8) Functions that determine the channels being used on the multiplexer

For t his tutorial, we are going to be using channels C1 and C2, which was shown on the initial schematic above. The function “digitalWrite” serves as our way to communicate with the Arduino by setting each selector bit high or low, depending on which IMU we wish to talk to.

The next step requires us to initialize the pins on the multiplexer as outputs so that the data being received on the IMUs can be sent over the channels to the Arduino. This is done by initializing the Arduino loop known as “void setup()”. This loop is the initialization stage of the Arduino programming code where the initialization of variables, outputs, and things that only need to be run once, occur. Figure 9 shows how this code should look.

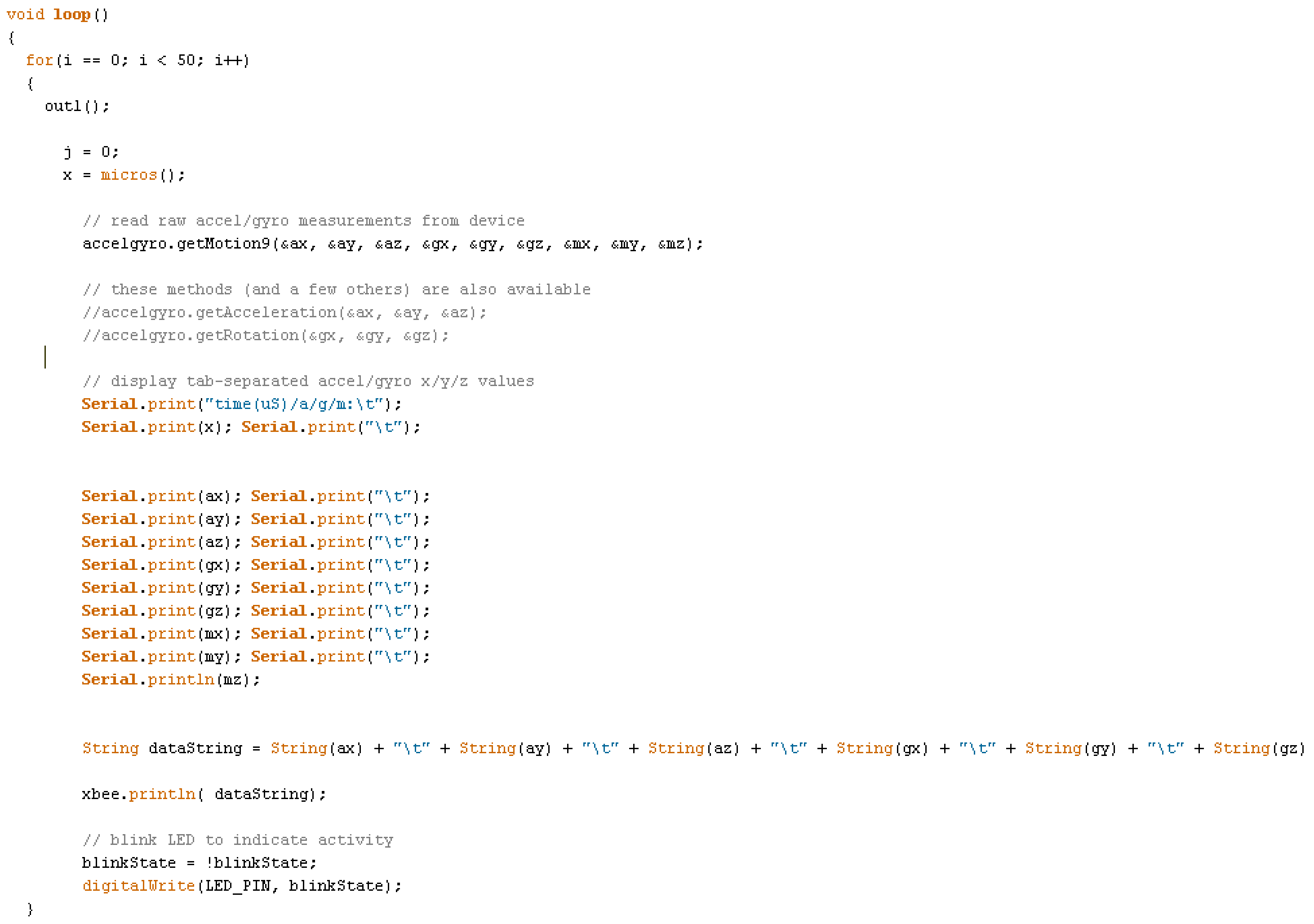


```
void setup()
{
    out1();
    out2();
    out1();
    pinMode (EN, OUTPUT);
    pinMode (S0, OUTPUT);
    pinMode (S1, OUTPUT);
    pinMode (S2, OUTPUT);
    pinMode (S3, OUTPUT);

    // join I2C bus (I2Cdev library doesn't do this automatically)
    Wire.begin();

    // initialize serial communication
    // (38400 chosen because it works as well at 8MHz as it does at 16MHz, but
    // it's really up to you depending on your project)
    Serial.begin(38400);

}
```

Figure 9) The setup loop of the multiplexer code

The figure above includes the function “pinMode” which is an Arduino function that enables us to determine whether we want specific pins to act as outputs or inputs. For this particular example, we would like the selector bits to act as outputs to allow for data transfer. Other functions in this loop include “Wire.begin()” which is a function that activates the “wire.h” library for initializing the I2C bus for the Arduino. “Serial.begin()” is another function that activates the baudrate of the Arduino software’s serial monitor. This is necessary if we want to check that our code is working properly, as data will be streaming to this monitor. Lastly, the 2 functions, “out1();” and “out2();” you should recognize because they were the two functions we just wrote to control the selector bits of the multiplexer. We need this code to make sure that both channels are set as outputs so that data can be read.

Next comes the loop that will allow us to constantly read data from both IMUs. This tutorial will have each IMU read values for 50 iterations and then switch to the other IMU. This process will repeat as long as the Arduino has power from your PC. This code is shown in the figure below, Figure 10.



```
void loop()
{
  for(i == 0; i < 50; i++)
  {
    outl();

      j = 0;
      x = micros();

        // read raw accel/gyro measurements from device
        accelgyro.getMotion9(&ax, &ay, &az, &gx, &gy, &gz, &mx, &my, &mz);

        // these methods (and a few others) are also available
        //accelgyro.getAcceleration(&ax, &ay, &az);
        //accelgyro.getRotation(&gx, &gy, &gz);

        // display tab-separated accel/gyro x/y/z values
        Serial.print("time(uS)/a/g/m:\t");
        Serial.print(x); Serial.print("\t");


        Serial.print(ax); Serial.print("\t");
        Serial.print(ay); Serial.print("\t");
        Serial.print(az); Serial.print("\t");
        Serial.print(gx); Serial.print("\t");
        Serial.print(gy); Serial.print("\t");
        Serial.print(gz); Serial.print("\t");
        Serial.print(mx); Serial.print("\t");
        Serial.print(my); Serial.print("\t");
        Serial.println(mz);


        String dataString = String(ax) + "\t" + String(ay) + "\t" + String(az) + "\t" + String(gx) + "\t" + String(gy) + "\t" + String(gz)

        xbee.println( dataString);

        // blink LED to indicate activity
        blinkState = !blinkState;
        digitalWrite(LED_PIN, blinkState);
  }
```

Figure 10) The void loop() for the multiplexer code

The figure above beings with the statement "void loop()" which is the initialization of a loop that will continue as long as the Arduino is powered. This is where all the data acquisition and also the switching between the two sensors occurs. The code is mainly from the base IMU-6050 library, but there are a few things added in order to provide functionality as well as verification that the devices are working. We need to make a for loop that will run for 50 iterations, increasing the intial variable by 1 for each loop. After the loop is complete, we then need to change which sensor we are listening to by creating another for loop below this one with the same exact code, just with another variable. For example, this loop uses the variable i in order to loop for 50 iterations, and it starts at the value 0 and keeps increasing for each loop. Once the value of i hits 50 it stops repeating this loop and goes on to the next part of the code. Remember though, that the function "void loop()" will be running constantly and this code will continue to repeat as long as the Arduino is powered. Notice also that in the code, j is set to 0 at the beginning of the loop, and the function out1() is also stated. This means that the channel that is being listened to at the moment is C1 of the multiplexer, and the variable j corresponds to the next loop after the loop i. The only difference between this loop and the next loop is that we need to set i=0 and we need to use the function out2() in order to read from sensor 2. Lastly, save your code, and then click on the checkmark near the upper left corner. The check mark verifies your code to make sure there are no errors or syntax errors. The verify button will tell you if there are any errors in your coding, and you need to fix them before moving on. Lastly we then click on the button next to it, which is the upload button, which will upload your code to your Arduino, as long as your Arduino is connected to your PC. Then, in order to view the serial monitor, click on tools on the menu bar, and click serial monitor. This is all shown in the figure below, figure 11.

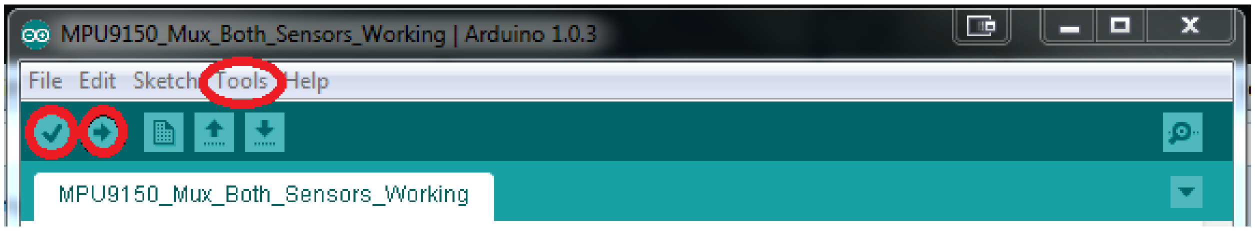
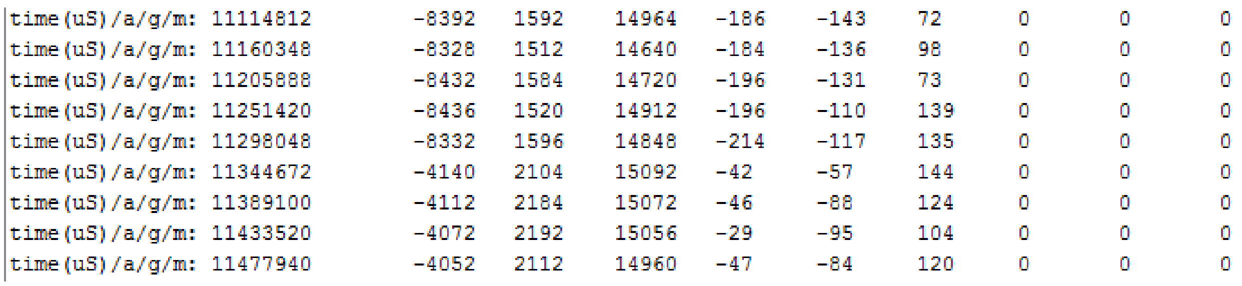

MPU9150_Mux_Both_Sensors_Working | Arduino 1.0.3
File Edit Sketch Tools Help
MPU9150_Mux_Both_Sensors_Working

Figure 11) Menu bar and buttons of the Arduino Software

After that, the code should be complete and we should have a working multiplexer. Below is figure 12, which should be similar to what is currently outputting on your serial monitor.



```
time(uS)/a/g/m: 11114812        -8392   1592    14964   -186    -143    72      0       0       0
time(uS)/a/g/m: 11160348        -8328   1512    14640   -184    -136    98      0       0       0
time(uS)/a/g/m: 11205888        -8432   1584    14720   -196    -131    73      0       0       0
time(uS)/a/g/m: 11251420        -8436   1520    14912   -196    -110    139     0       0       0
time(uS)/a/g/m: 11298048        -8332   1596    14848   -214    -117    135     0       0       0
time(uS)/a/g/m: 11344672        -4140   2104    15092   -42     -57     144     0       0       0
time(uS)/a/g/m: 11389100        -4112   2184    15072   -46     -88     124     0       0       0
time(uS)/a/g/m: 11433520        -4072   2192    15056   -29     -95     104     0       0       0
time(uS)/a/g/m: 11477940        -4052   2112    14960   -47     -84     120     0       0       0
```

Figure 12) The serial monitor output with both sensors working

As you can see from the figure above, the two sensors are being read after 50 iterations, and you can see the change in values between the sensors as they switch between the two IMUs.