

Rising Data

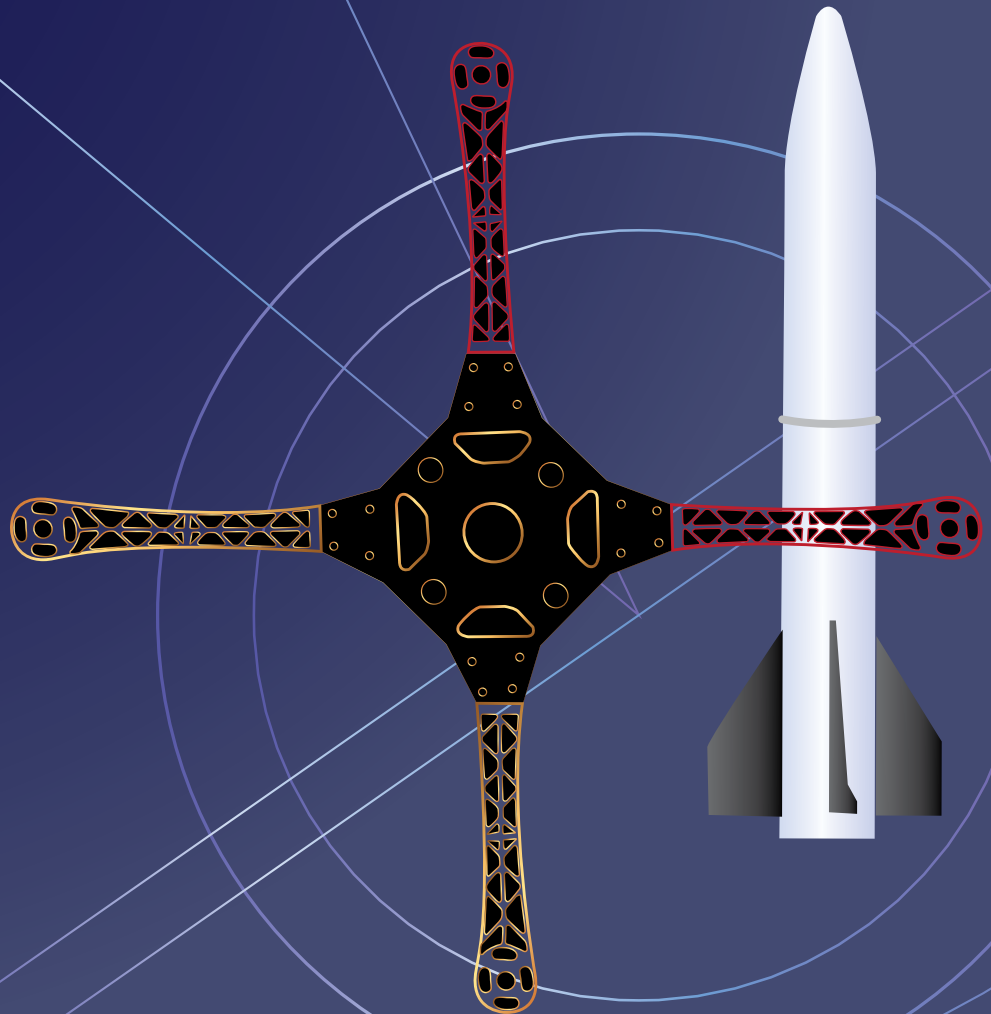
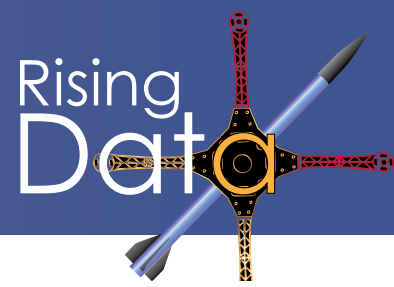


Table of Contents



Chapters	Pages
Parts and Assembly Guide	1
1. Rising Data Hardware Setup	1-1
2. Arduino Integration	2-1
3. SD Card Integration	3-1
4-A. 9DoF Integration	4-1
4-B. BME280 Integration	4-5
5. TMP-36 Integration	5-1
6. Battery Board Integration	6-1
7. Building the Payload	7-1
8. Building the Rocket	8-1
Additional Payload Photos	9-1

Acknowledgement:

Funding for the Rising Data program has been provided by NASA's Minority University Research and Education Program #NNX15AV98A. This guide was developed by the Education and Public Outreach group at Sonoma State University, under the direction of Prof. Lynn Cominsky. Questions about this guide should be sent to lynnc@universe.sonoma.edu.

Reference:

All videos and guides used in Rising Data are linked here:
<http://lbym.sonoma.edu/RisingData/node/5>

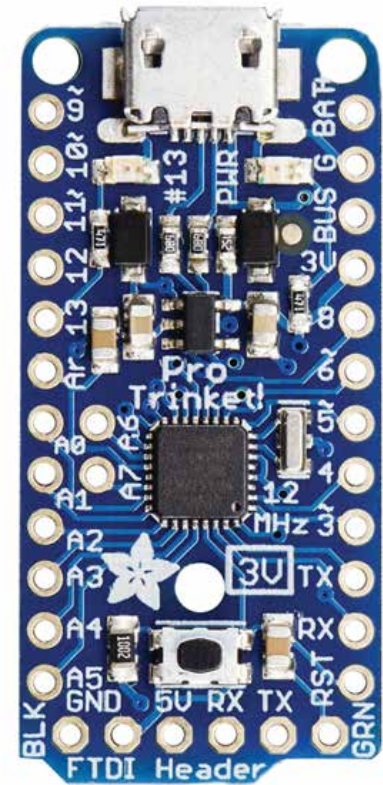
Parts

Arduino Pro Trinket 3V

This device provides the processing power for the payload. It uses an ATmega328 Processor that runs regular Arduino C/Processing Code written with the Arduino IDE (Integrated Development Environment). While this device has a USB Port capable of powering the device and loading code, it does NOT have an on-board FTDI chip with which many other Arduinos come equipped. This means that in order to have serial communication with the device, it is necessary to use an FTDI cable instead of a regular USB cable.

There are two versions of the Pro Trinket that use 3.3V and 5V for logic levels. For Rising Data, we are using 3.3V, so be careful if ordering additional boards. The 5V has (almost) identical pins, and all the documentation provided by the manufacturer is for the 5V version. Don't let this confuse you.

Instructions for testing the Arduino are in sections 1 and 2.

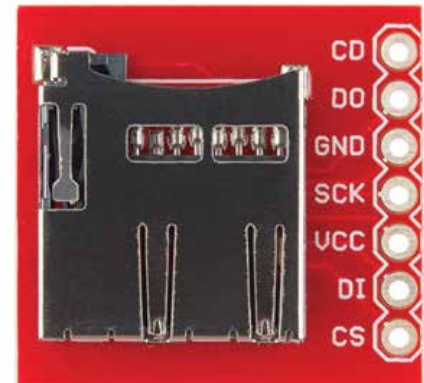


MicroSD Card Device

This device provides mechanical and electrical connections to the pins on a micro SD card. Unlike other SD breakout boards, this does not provide any kind of integrated processor or logic level conversion. It only provides a mechanical socket for the SD card, and brings the SD card pins out into a DIP mountable format.

This device connects to the Arduino over an SPI bus. It allows us to quickly log data as we collect it. The software we provide allows us to write to an SD card formatted FAT16/32, so it can be removed and read as a text file from a regular computer.

Instructions for wiring and testing the SD Card are in section 3.



9DoF (Nine Degrees of Freedom) Device

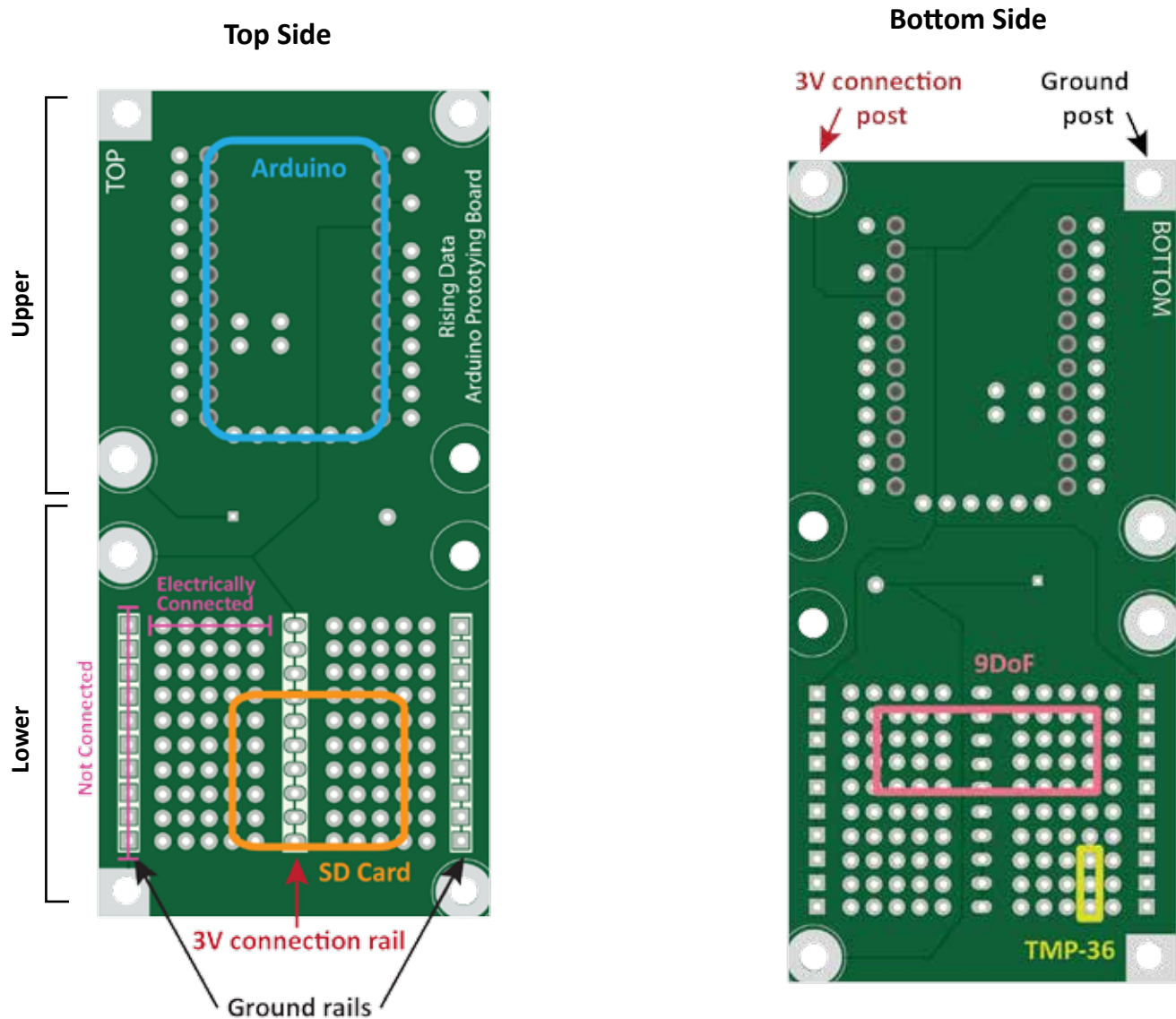
This device incorporates three sub-sensors: a 3-axis accelerometer, a 3-axis gyro, and a 3-axis magnetometer. This provides telemetry to analyze after flights. It communicates with the Arduino over an I²C interface.

Instructions for wiring and testing the 9DoF are in section 4.



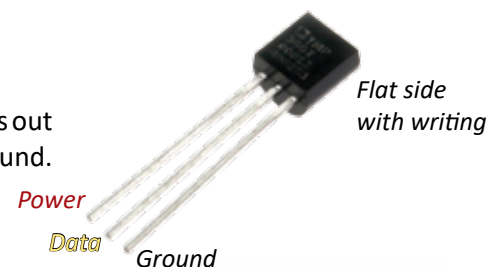
Protoboard

These boards have been custom designed for the Rising Data project. On the top side of the board, there are special areas for soldering the socket that holds the Arduino (upper) and SD card (lower). On the bottom side of the board, the 9DoF and TMP-36 sockets can be soldered in the lower area. There is also room for additional devices in this lower area. Note that there are traces built into the board that connect the power and ground pins for the Arduino to the inside and outside rails, respectively. There are also holes in all four corners of the board that connect to posts to allow the board to become part of the stack that comprises the payload. Some of the posts are connected to ground, and others to 3V power supplied by the battery board.



TMP-36

The TMP-36 is a relatively linear analog device, which means that the voltage it puts out is proportional to the ambient temperature. It has three pins: Power, data and ground.

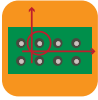


Female Header Socket

We will be attaching all our parts using rows of female header sockets so that parts can be easily attached and removed from the protoboard. These will be soldered to the protoboard (instead of the parts).

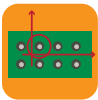
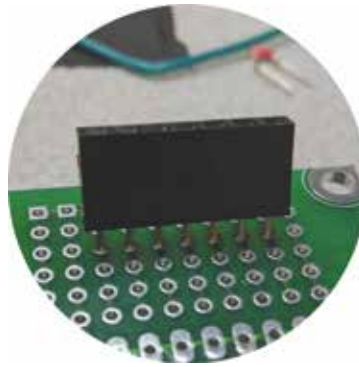


Assembly Information



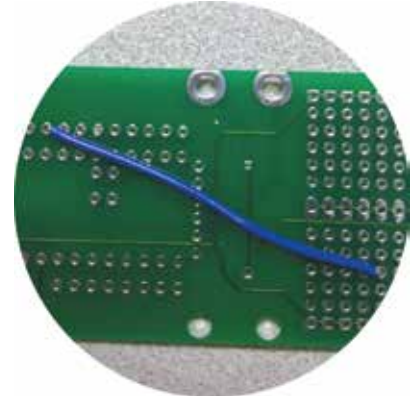
Place Socket

Put the part through the holes you intend to solder it into. You may also decide to tape the part in place so it doesn't fall out when you pass it off to have it checked or when you move it around to solder it.

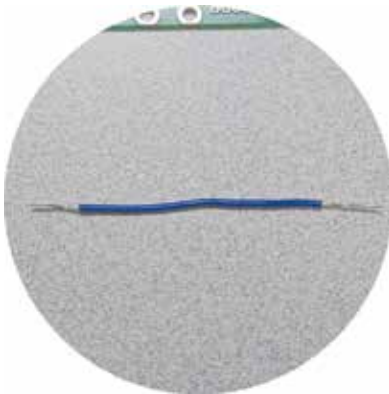


Wiring Placement

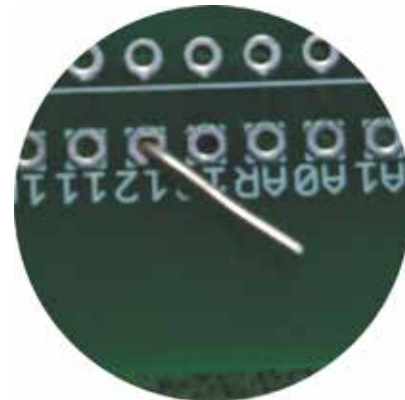
First cut the wires to the desired length. You want the wire to fit comfortably between the two pins it connects, but you don't want enough extra slack that it could get snagged when being handled. Be sure to leave about an extra inch past what you need for the next step (stripping the ends).



Strip about ½ inch off each end of the wire so you can connect it to the board with solder.



After putting the wires through the appropriate holes, fold them over as shown. This will keep them from falling out in the time between placing them and soldering them.



For the very short jumper wires used to connect power and ground to the devices, you may find it difficult to strip the wires and leave a tiny bit of insulation in place. Consult the wire stripping help guide on page 4 if you are having trouble.



Check

Pass the payload to a partner to have them confirm your placement. This step is critical to avoid mistakes that are very difficult to reverse. You should be checking the placement against the images in the document. You should be ensuring that parts are placed one row over from the edge, to allow the ground jumper pins to be connected as shown in the diagrams.

You should also be checking to insure that the part is being placed through the correct face (top or bottom) of the board. The top face of the board is the one with writing on it, the bottom face has no writing. Placing the part through the wrong face will prevent the payload from operating.



Solder

This is the step in which you will solder parts in place.



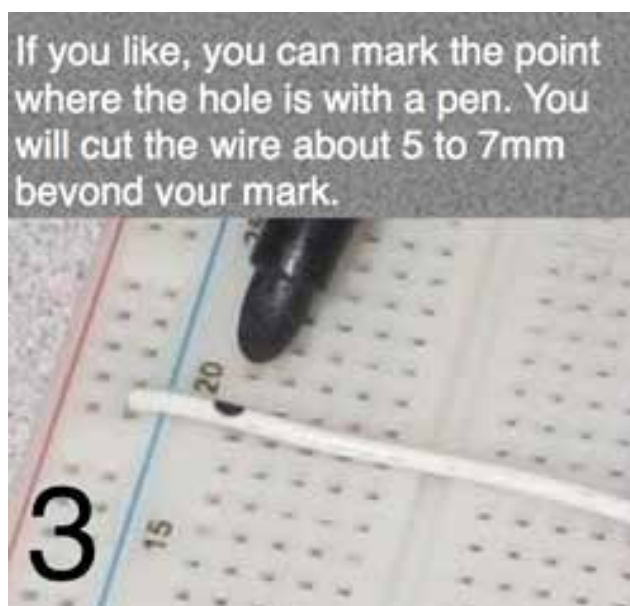
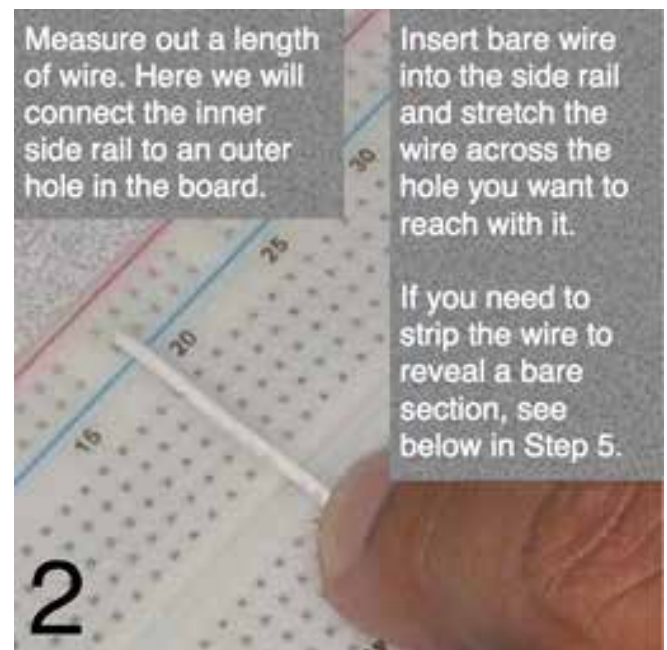
Test Software

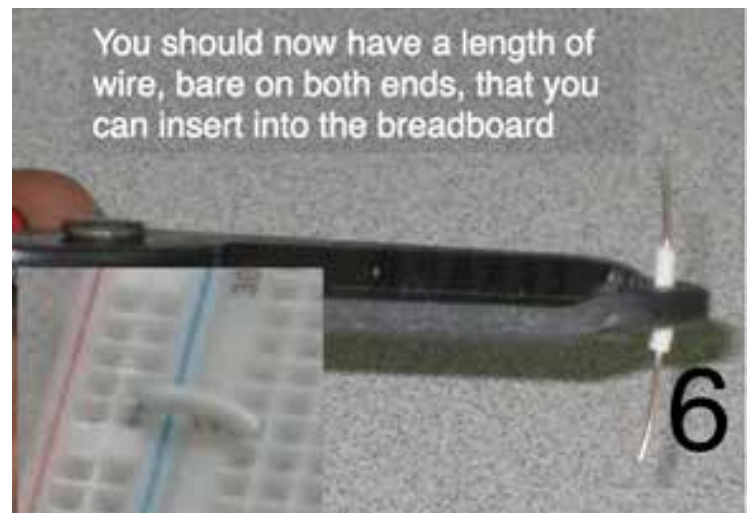
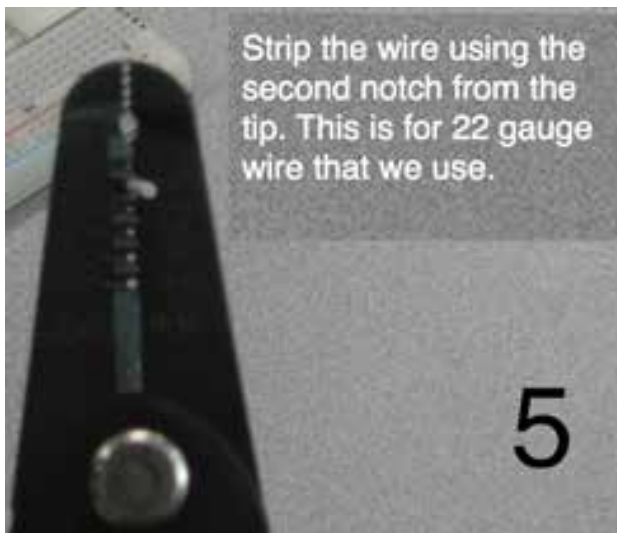
The final step for each new part is to check the software. You should test both the new software for each part and the software that previously functioned correctly.



Cutting and Stripping Wires

Cutting wires to length and stripping their insulation for use on a breadboard or protoboard is a basic skill needed for building test and experimental circuits. This brief guide shows how this can be done.





The following link also has information about stripping wires:

<https://learn.sparkfun.com/tutorials/working-with-wire/how-to-strip-a-wire>



Soldering Basics

You should be able to do this soldering with proper supervision. You will need the following tools, which are shown to the right:

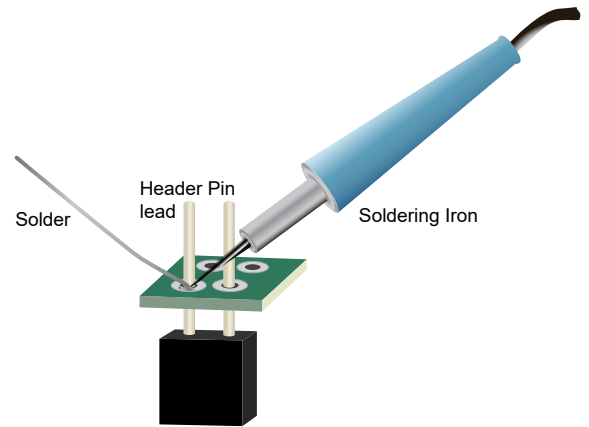
- A pair of safety goggles
- Electrical Tape
- Helping hands (something to hold the board still while you are soldering). You can also use vises or clamps
- Solder wire
- Brass sponge
- A soldering iron
- Regular Pliers
- Needle nose pliers
- Small flat head screwdriver
- Wire cutters



When soldering, remember to always wear safety goggles. Additionally, make sure you've read the user's manual for your soldering iron for specific information related to its use.

Before you begin, be sure to turn the soldering iron on so that it is hot enough when you wish to begin soldering.

1. Take the soldering wire and unravel about 6-10 inches of it so that you have plenty of slack.
2. Take the soldering iron out of its holder and touch the tip of it on the brass sponge to be sure it is clean and ready.
3. Hold the soldering iron with one hand up against the side of the lead as close to the board as you can. You will want to hold it at an angle as shown to the right.
4. On the other side of the lead you will hold the tip of the soldering wire. As you do this, you will see the wire starting to melt and it will form a small pyramid shape of solder around your lead. As soon as you see this happen, remove the wire first and then the iron.
5. Place the iron back in its holder and look closely to be sure you've made a good connection.

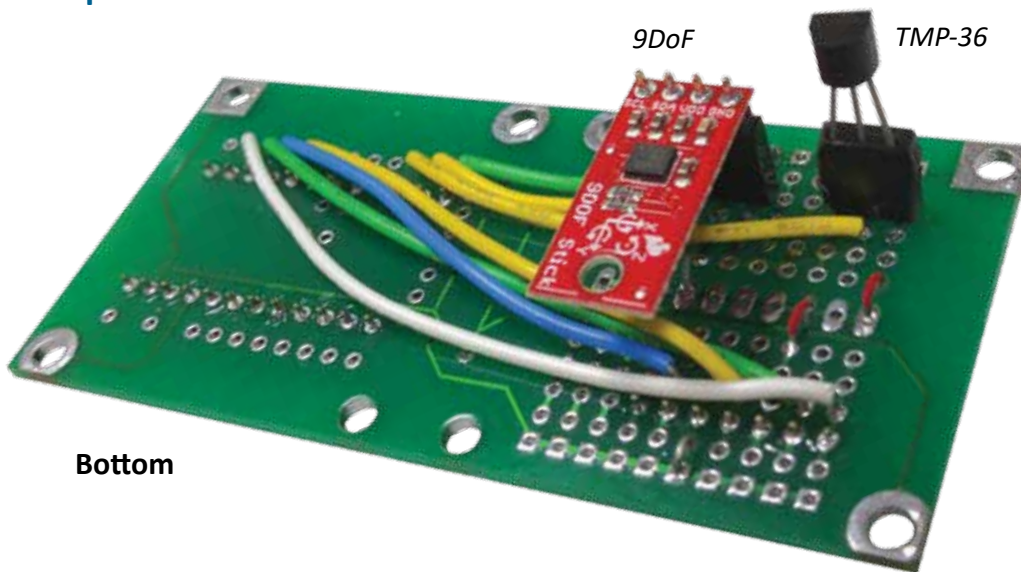


5 Soldering Video:

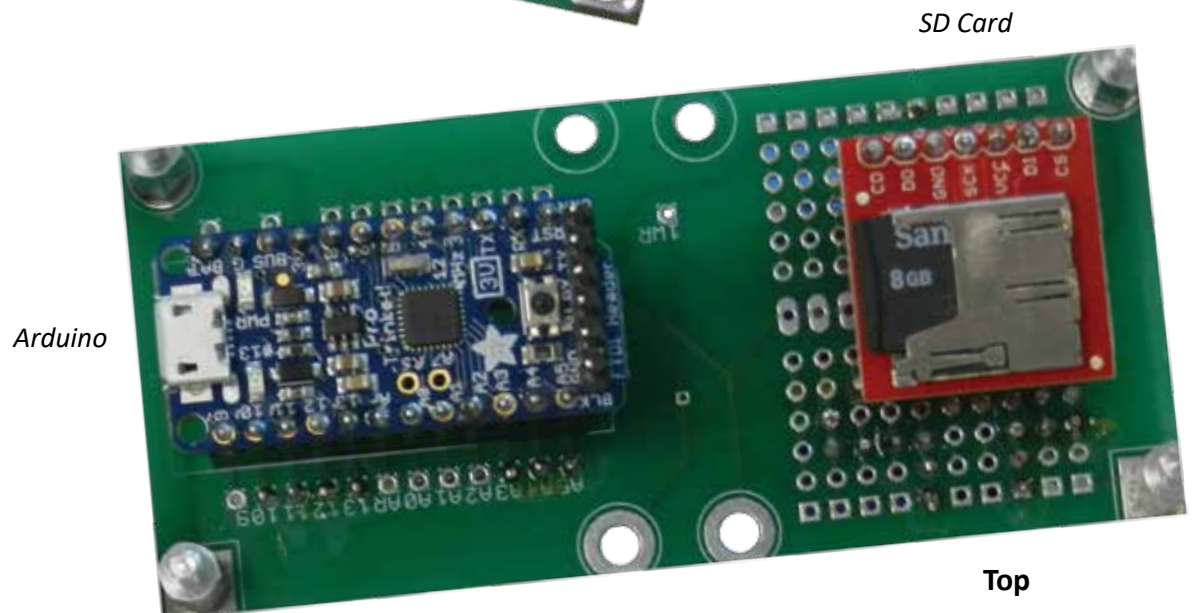
For more information about soldering see the video at: <http://lbym.sonoma.edu/RisingData/node/5>



Completed Protoboard:



Bottom

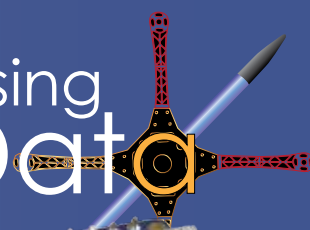


Arduino

Top

1. Rising Data Hardware Setup

Rising
Data



Part 1 – Hardware Preparation: The Arduino Pro Trinket and FTDI cables are prepared to prepared in order to talk to the payload from the computer. Header pins are soldered on to the Arduino as well as onto the 9DoF and SD card “sticks.”

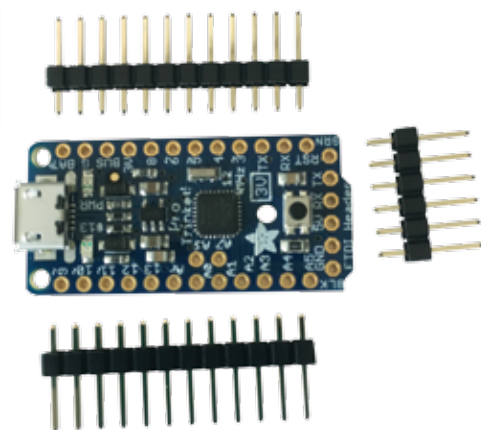
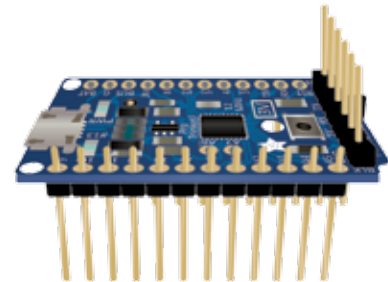
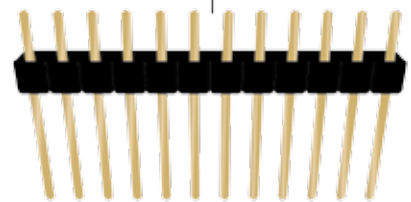
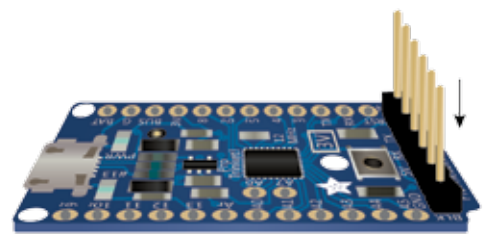
Part 2 – Integration Testing: The Arduino is connected to the computer and data are sent back and forth to verify that the serial connection is working.

Note: There are two version of the Arduino Pro Trinket being sold by Adafruit, the 5V and 3.3V variants. In many instances throughout the documentation we inherited from Adafruit, the 5V version is referenced or imaged. We will always be using the 3.3V version, even when the documentation might suggest otherwise.

Part 1 – Hardware Preparation:

1) Solder Header Pins onto Arduino:

The Arduino needs two 12-pin and one six-pin header pins. Break off the lengths from the 40-pin sticks that are provided in your kit. The longer pin side of each stick will eventually be plugged into sockets soldered onto the green protoboard, and the shorter pin side will be soldered to the Arduino itself.

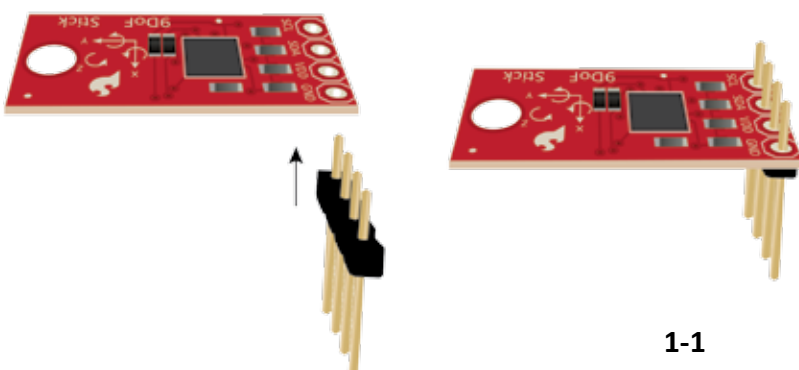
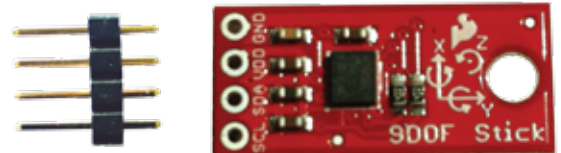


Solder each of the two 12-piece header pins onto the sides of the Arduino facing downwards.

Solder one of the 6-piece header pins along the bottom of the Arduino facing upwards.

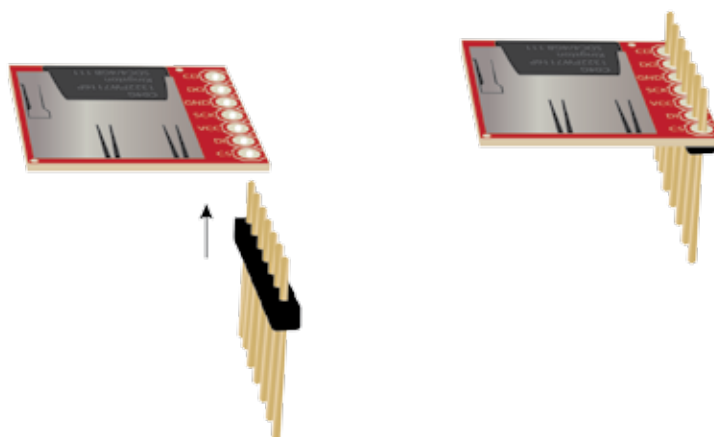
2) Solder Header Pins onto the 9DoF:

For the 9DoF, break off a piece with four header pins. Again, the shorter side will be soldered to the 9DoF itself, while the longer side will eventually be plugged into sockets on the green protoboard.



3) Solder Header Pins onto the SD Card:

For the SD card, break off a piece with 7 header pins. Again, the shorter side will be soldered to the SD card itself, while the longer side will eventually be plugged into sockets on the green protoboard.



4) Pin Back Unused Lines on the FTDI cable:

We will use the black, brown, red, orange, yellow, and green wires from the FTDI cable. The unused cables can be secured back on the base using a cable tie or small zip tie to keep them out of the way.



Part 2 – Integration Testing for the Arduino:

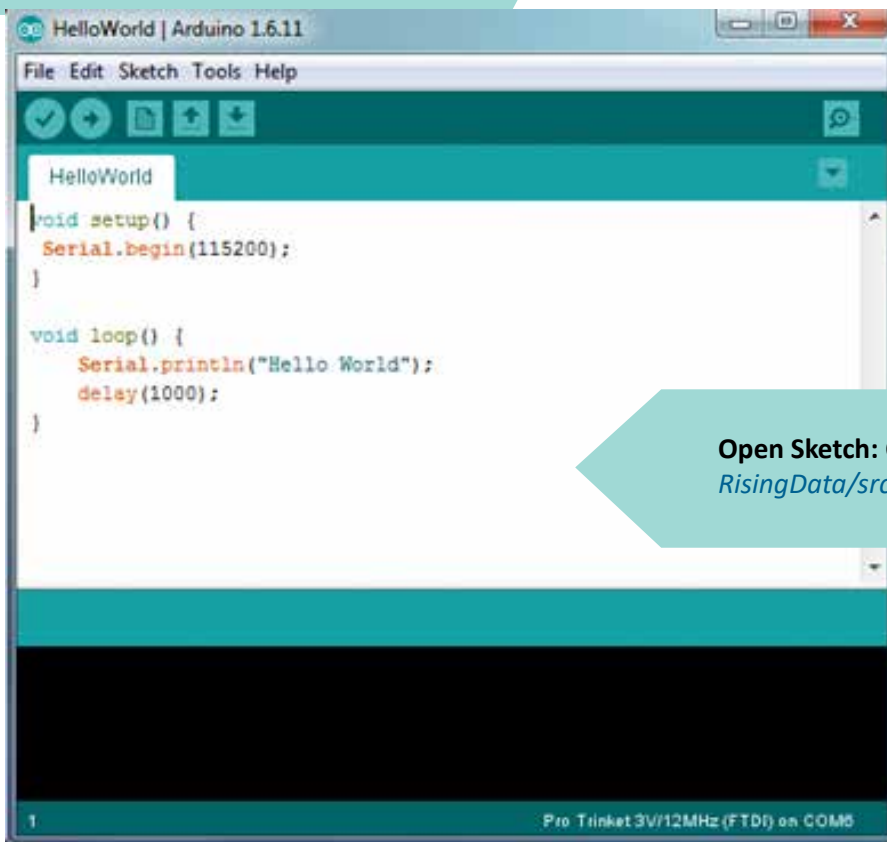
FTDI Pins: Connect FTDI Pins to header pins on the Arduino as shown in the image:

Note: Physicists might find it easy to remember the order as Black, Brown, [R,O,Y,G] when going from left to right.



FTDI USB: Connect the USB side of the cable to your computer's USB port.

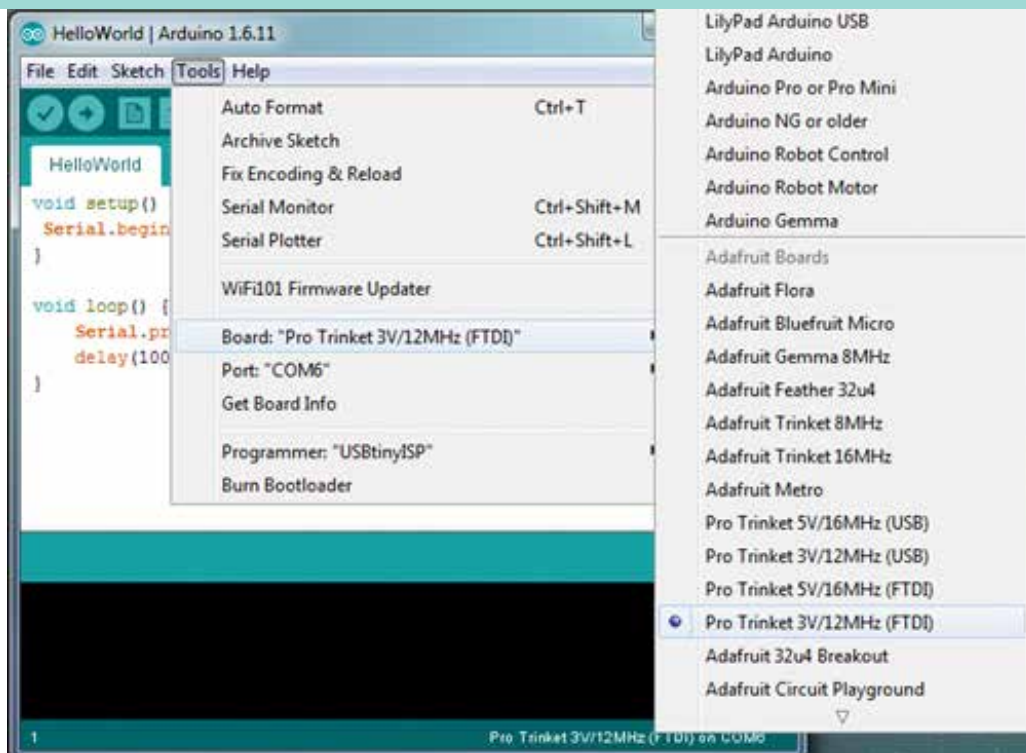
IDE Software Testing “Hello World”:

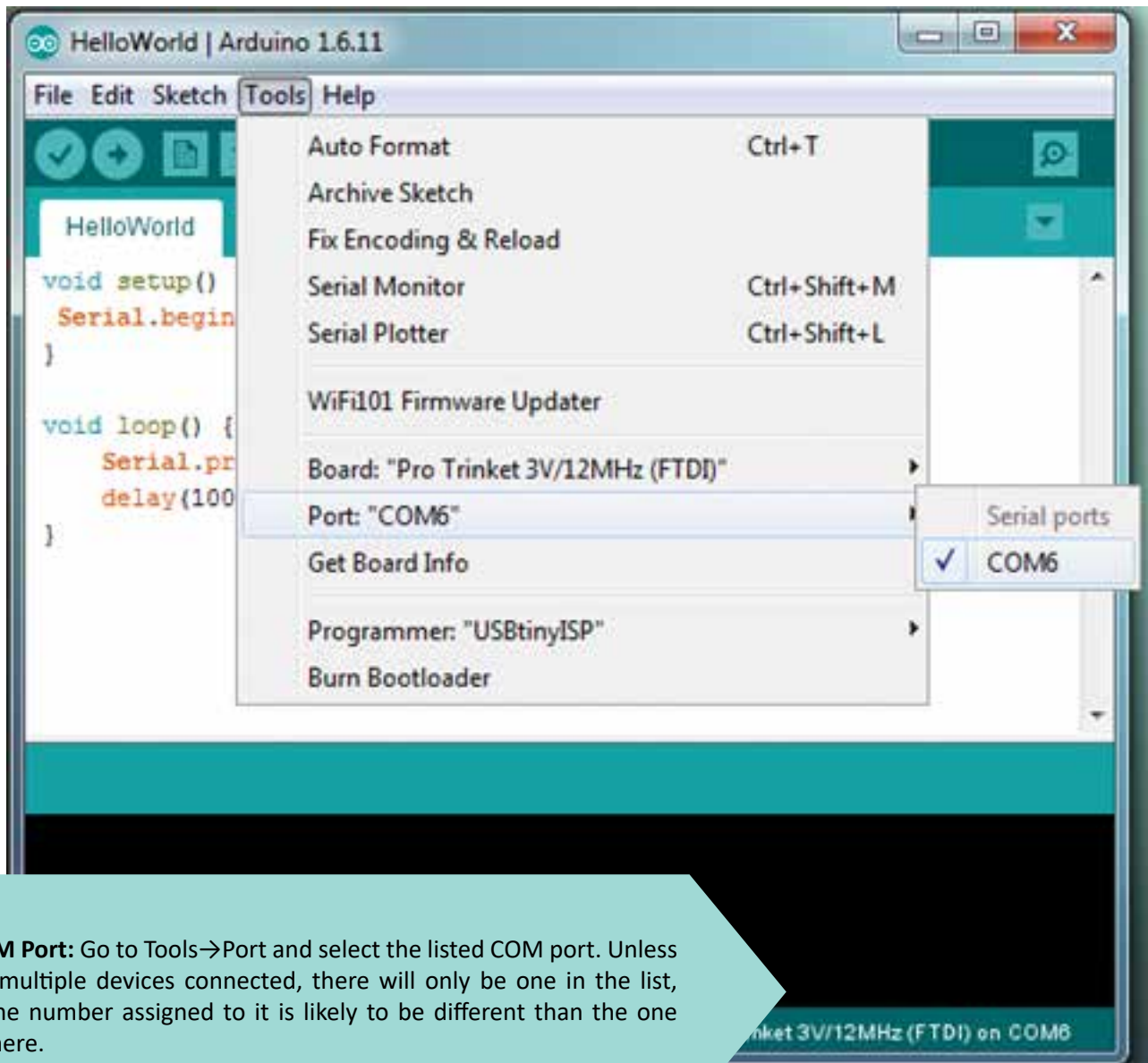


Open Sketch: Open the sketch found at [RisingData/src/HelloWorld/HelloWorld.ini](#)

This very simple program is designed to communicate a simple message over the serial connection. Compiling, loading, and running this sketch will show us that the serial communications are working correctly.

Select Device: Go to **Tools→Board** and select the *Arduino Pro Trinket 3V/12Mhz* (FTDI) option.





Select COM Port: Go to Tools→Port and select the listed COM port. Unless you have multiple devices connected, there will only be one in the list, though the number assigned to it is likely to be different than the one depicted here.

Compile and Upload: Compile and upload the code to your device by pressing the upload button (a right-pointing arrow). When finished, you should see the following at the bottom of your window:



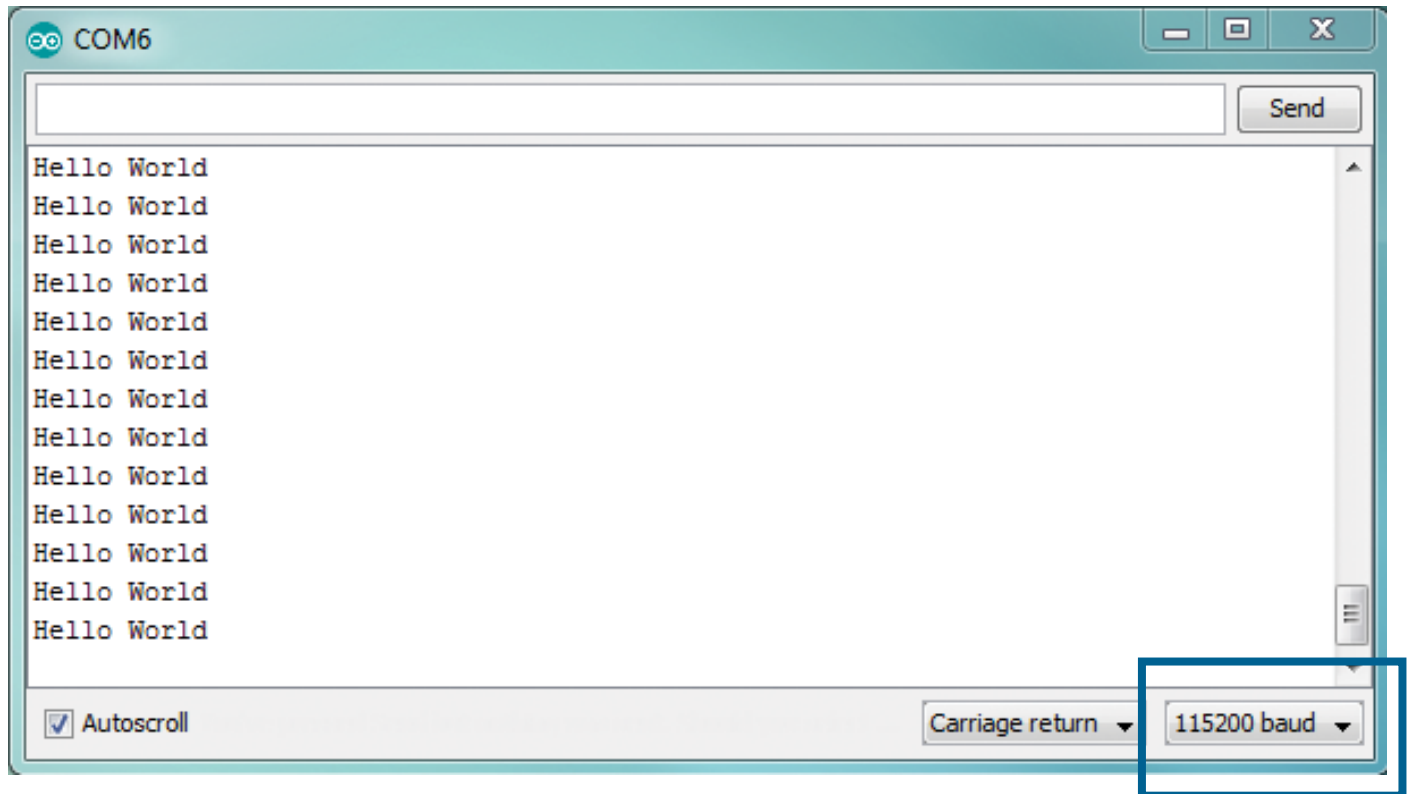
Done uploading.

```
Sketch uses 1,638 bytes (5%) of program storage space. Maximum is 28,672
Global variables use 198 bytes of dynamic memory.
```

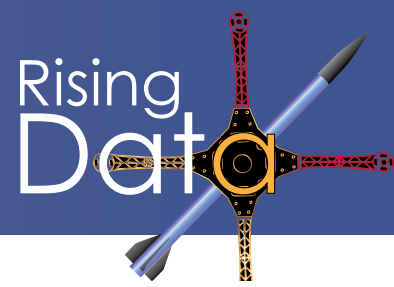

Open the Serial Monitor: Open the serial monitor by pressing



Make sure you change the baud rate (bottom right corner) from the default value of 9600 to the new value of 115200. You should then see “Hello World” written to the screen at a rate of once per second. This means everything is working and we are ready to start testing the other chips once they are plugged and wired into the protoboard.



2. Arduino Integration

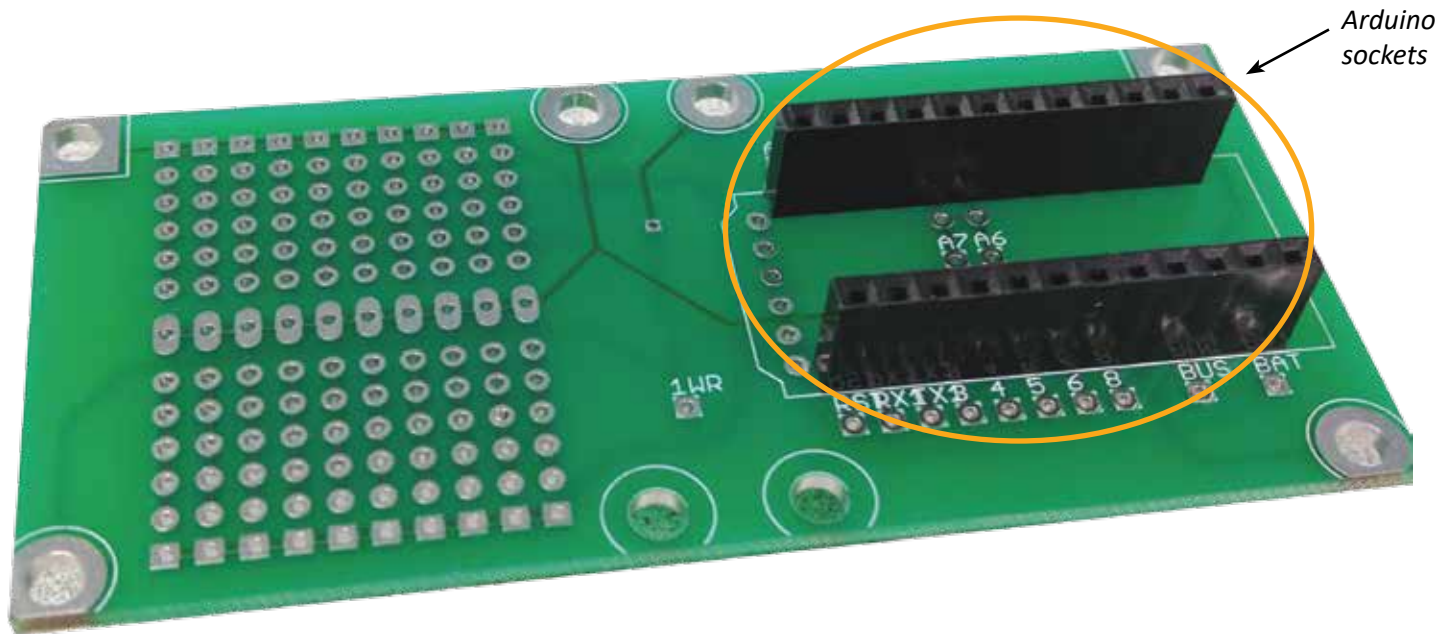


In this section, we will connect the Arduino to the protoboard. If you have not already done so, unplug the Arduino Pro Trinket from the FTDI cable. You will first connect the header socket rows ("sockets") to the protoboard, then you will plug the Arduino in and repeat the software test that you ran in the previous section ("Hello world").



Place sockets

Insert the two 12-pin female sockets into the protoboard as shown. Note that they should go through the innermost pair of hole rows. Also note that the sockets go into the top side of the protoboard (the side with writing on it)



Check

Have a partner check your work by plugging the Arduino into the sockets to ensure it fits.



Solder

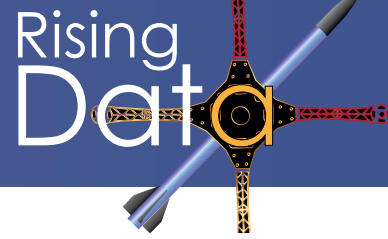
Solder both sockets in place.



Test Software

Connect the FTDI cable to the Arduino and repeat the Hello World software test described in the previous section (Rising Data Hardware Setup).

3. SD Card Integration

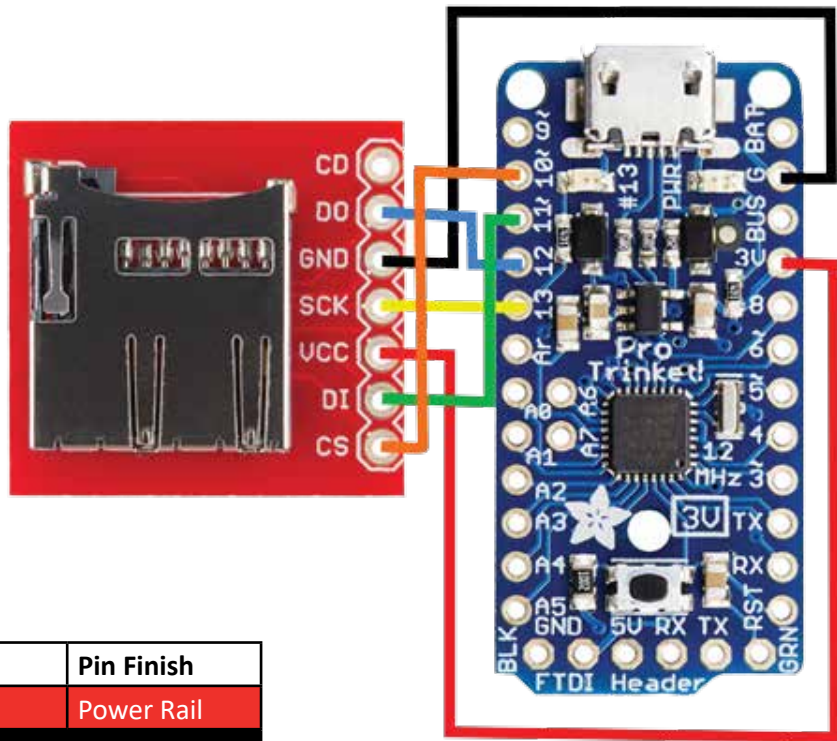


In this section, we will connect the SD card device to the Arduino using a standard Serial Interface connection. We will then insert an SD card, write some text into a file on that SD card, and finally insert the SD card into a computer card reader so that we can view the text file on the computer.

Part 1 - Wiring and Soldering

The SD Card communicates over an interface called Serial Peripheral Interface or SPI. The SPI bus has three lines for data going from the Arduino to the SD card (green), from the SD card to the Arduino (blue), a shared clock line to ensure synchronous operations (yellow), and a Chip Select so the Arduino can specify which device on the SPI bus it wants to talk to (orange). Of course, the device also requires a 3.3V power line (red) and a ground line (Black).

The goal is to wire the SD card to the Arduino as shown in the schematic. Note that the top pin (CD) on the SD card is not used in this configuration. Also remember that your protoboards provide power and ground rails, so you don't need to connect the SD card pins directly to the Arduino pins. Instead, connect the power (VCC) and ground (GND) pins on the SD card device to the correct rails on the protoboard.



The SDI connections are:

Color	Pin Start	Pin Finish
Red	VCC	Power Rail
Black	GND	Ground Rail
Blue	DO	12
Orange on schematics	CS	10
Yellow	SCK	13
Green	DI	11

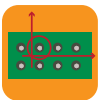
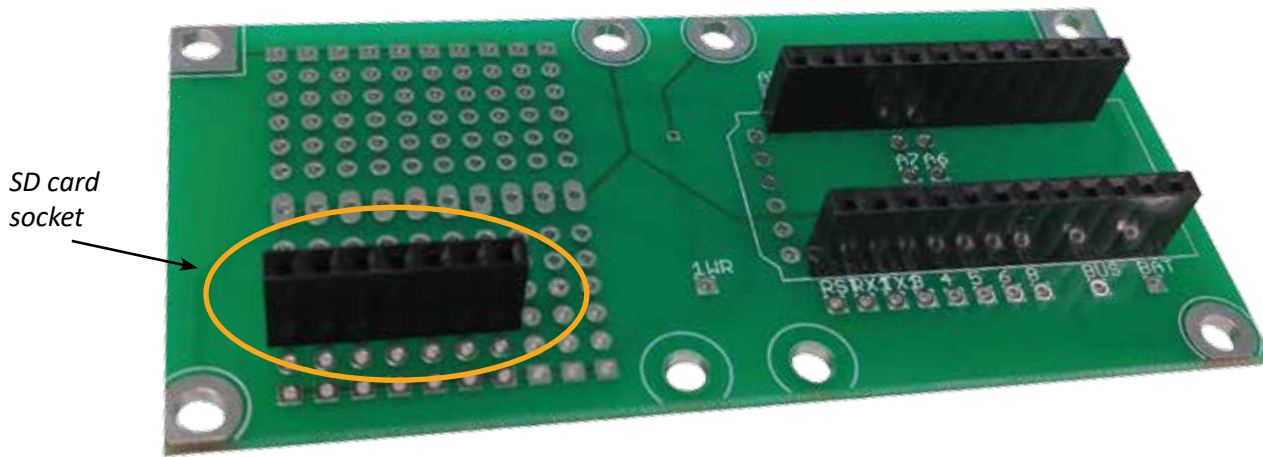
Note: we use orange for readability in the diagrams to represent the white wire used on the device

The SD card device will be plugged into a socket on the top of the protoboard after some connecting wires are attached on the bottom of the board.



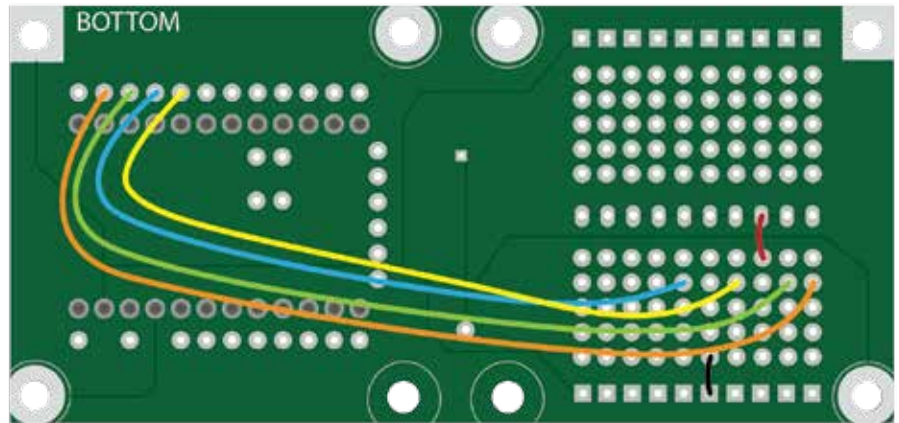
Place Socket

Insert the 7-pin socket into the protoboard as shown. Note the placement from the edge. The outside row is the power rail, then skip a row, then place the socket. Also note that the socket goes into the top of the protoboard (the side with writing on it).



Place Wires

The wires are placed on the bottom side of the protoboard, including the red and black jumpers for power and ground. The wiring placement assumes that the SD card device will be located on the top side, as shown above:



Check

Have a partner check your work.



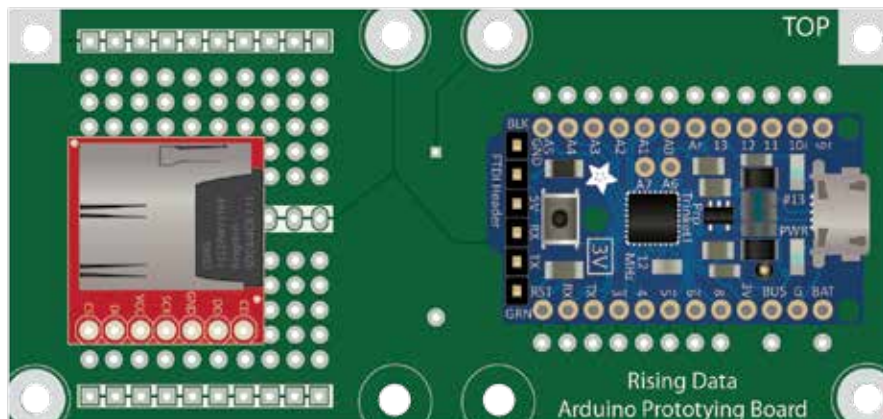
Solder

If your partner agrees that you have placed the wires and the socket correctly, you can solder these parts in place.



Test Software

Plug the SD card reader into the socket on the top side of the board. Proceed to test the SD card using the software described on page 3-3.



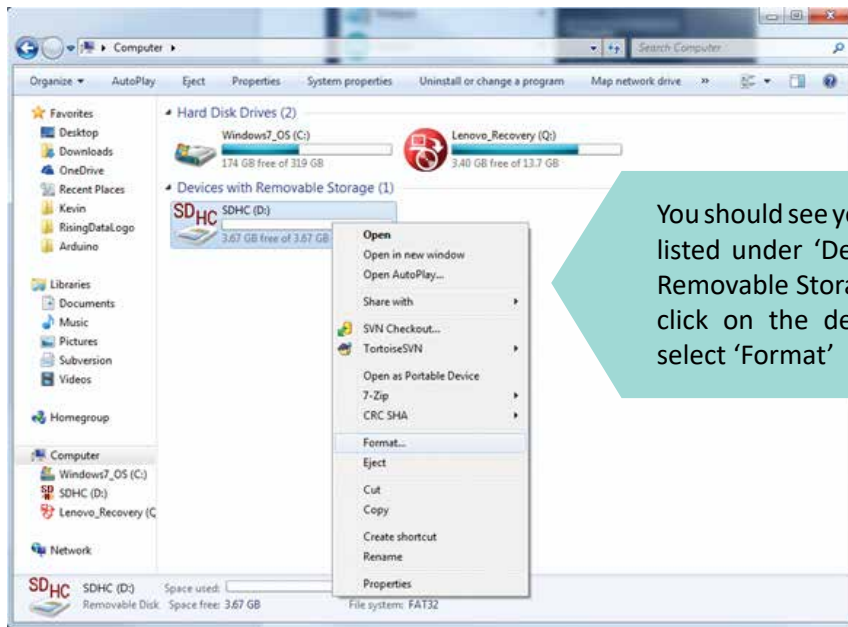
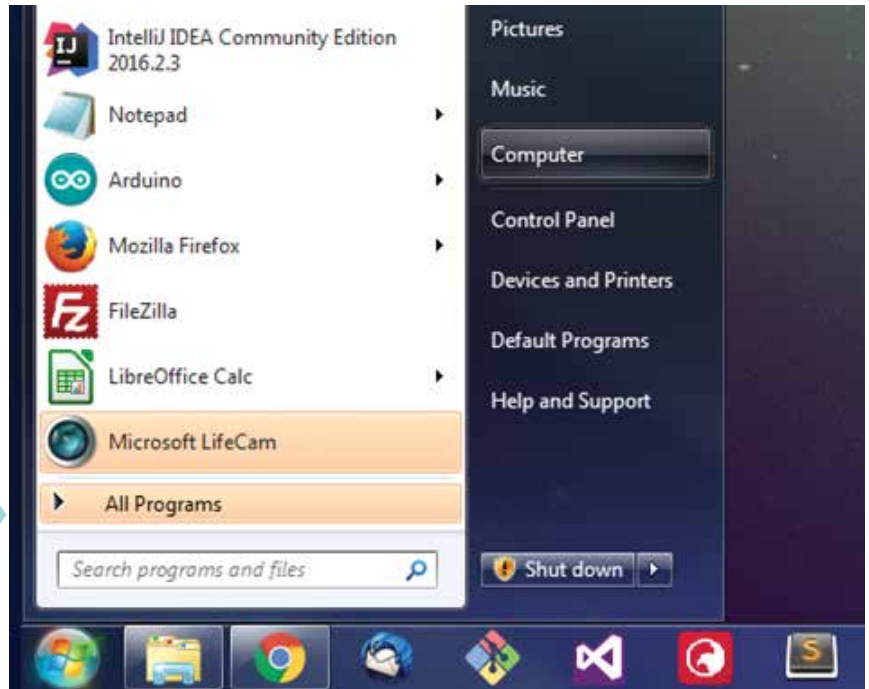
Part 2 - SD Card Software

We will use a Windows 7 or 10 computer to format the SD card as FAT32.

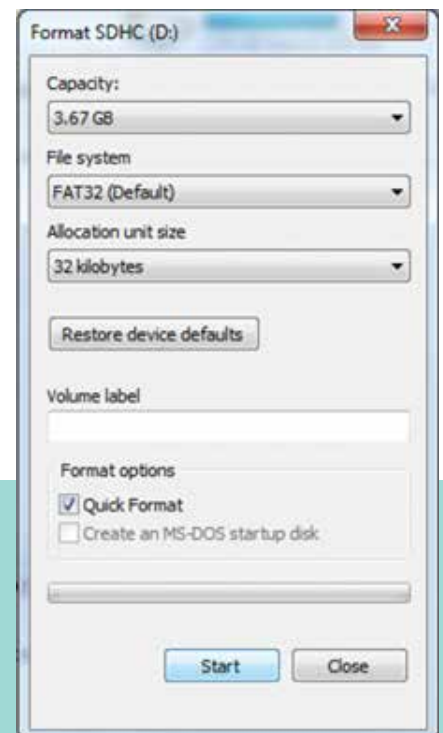


To keep the payload small, we are using a micro SD card. This is the same type of small form factor storage device you might see in your cell phones or tablets. However, most card readers are not equipped to read a device this small. With your SD card, you will also have received a micro to standard SD adapter as pictured in the upper right corner. This lets the micro SD card fit into your computer's card reader.

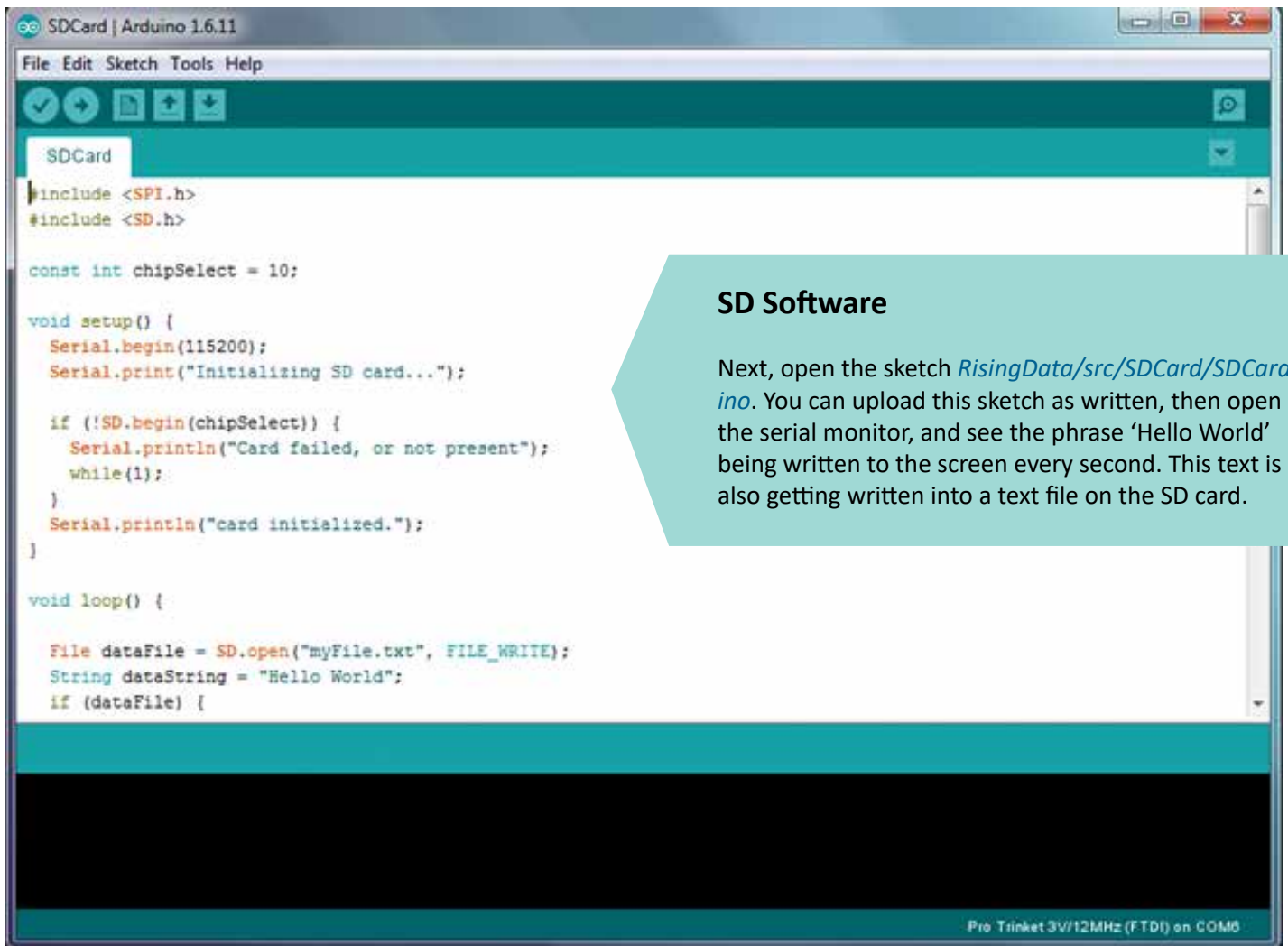
Put the card into your computer's card reader and it should be recognized by windows. Next, go to **Start→ Computer**



You should see your device listed under 'Devices and Removable Storage.' Right click on the device, and select 'Format'



You will be presented with a dialogue window like the one to the right. Make sure the File System shows as FAT32 and then press start. You will be warned (and rightly so) that this process will delete all the information on the drive. If you have any data on this drive, be sure to save it somewhere safe before continuing. When complete, you will get a process complete message. At this point you may remove the SD card from the computer and place it in the SD card device on your payload.



The screenshot shows the Arduino IDE interface with the 'SDCard' sketch open. The code includes SPI and SD libraries, defines a chipSelect pin, and contains setup and loop functions. The setup function initializes the serial port and the SD card, printing status messages. The loop function opens a file named 'myFile.txt' and writes 'Hello World' to it.

```
SDCard
#include <SPI.h>
#include <SD.h>

const int chipSelect = 10;

void setup() {
  Serial.begin(115200);
  Serial.print("Initializing SD card...");

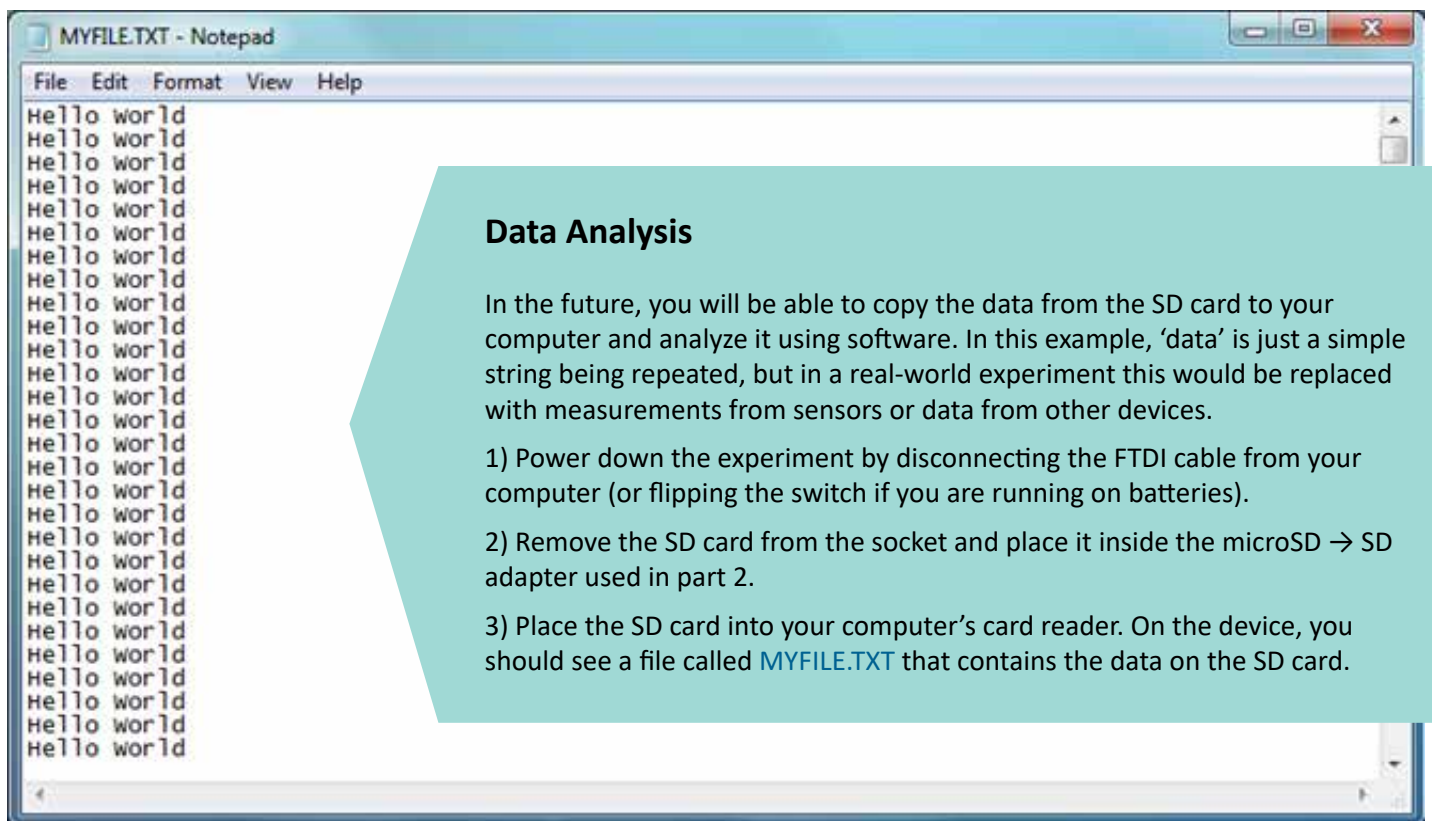
  if (!SD.begin(chipSelect)) {
    Serial.println("Card failed, or not present");
    while(1);
  }
  Serial.println("card initialized.");
}

void loop() {

  File dataFile = SD.open("myFile.txt", FILE_WRITE);
  String dataString = "Hello World";
  if (dataFile) {
```

SD Software

Next, open the sketch [RisingData/src/SDCard/SDCard.ino](#). You can upload this sketch as written, then open the serial monitor, and see the phrase 'Hello World' being written to the screen every second. This text is also getting written into a text file on the SD card.



The screenshot shows a Notepad window titled 'MYFILE.TXT - Notepad'. The file contains 20 lines of the text 'Hello world'.

```
MYFILE.TXT - Notepad
File Edit Format View Help
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
```

Data Analysis

In the future, you will be able to copy the data from the SD card to your computer and analyze it using software. In this example, 'data' is just a simple string being repeated, but in a real-world experiment this would be replaced with measurements from sensors or data from other devices.

- 1) Power down the experiment by disconnecting the FTDI cable from your computer (or flipping the switch if you are running on batteries).
- 2) Remove the SD card from the socket and place it inside the microSD → SD adapter used in part 2.
- 3) Place the SD card into your computer's card reader. On the device, you should see a file called [MYFILE.TXT](#) that contains the data on the SD card.

Appendix: Code Tour

```
#include <SPI.h>
#include <SD.h>
```

These lines import two libraries. The first, SPI, is the library that implements the SPI interface. The second, SD, is the library that contains the functions which read and write from the SD card using a FAT32 file system.

```
const int chipSelect = 10;
```

The SPI interface requires a dedicated chip select line to be assigned to each and every device on the bus. This line is driven high to put the desired device(s) into a listening state. In our case, we are only using one device so we only have one chip select line. We assigned pin 10, because it happens to be next to the pins for the SPI bus.

```
void setup() {
  Serial.begin(115200);
  Serial.print("Initializing SD card...");

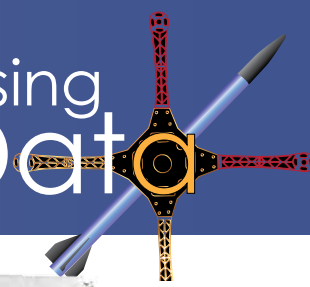
  if (!SD.begin(chipSelect)) {
    Serial.println("Card failed, or not present");
    while(1);
  }
  Serial.println("card initialized.");
}
```

The setup loop first initializes the serial port for debugging. We use a higher than default baud rate of 115200, as it will be useful later in the project. Next we try and initialize the SD card. In our logic, we specify that if the SD card location on the chip select line does not report that it has initialized successfully, we go into an infinite loop to stop program execution. Otherwise, we print that the card initialized and we move on.

```
void loop() {

  File dataFile = SD.open("myFile.txt", FILE_WRITE);
  String dataString = "Hello World";
  if (dataFile) {
    dataFile.println(dataString);
    dataFile.close();
    Serial.println(dataString);
  }
  else {
    Serial.println("error opening file");
  }
  delay(100);
}
```

4-A 9DoF Integration



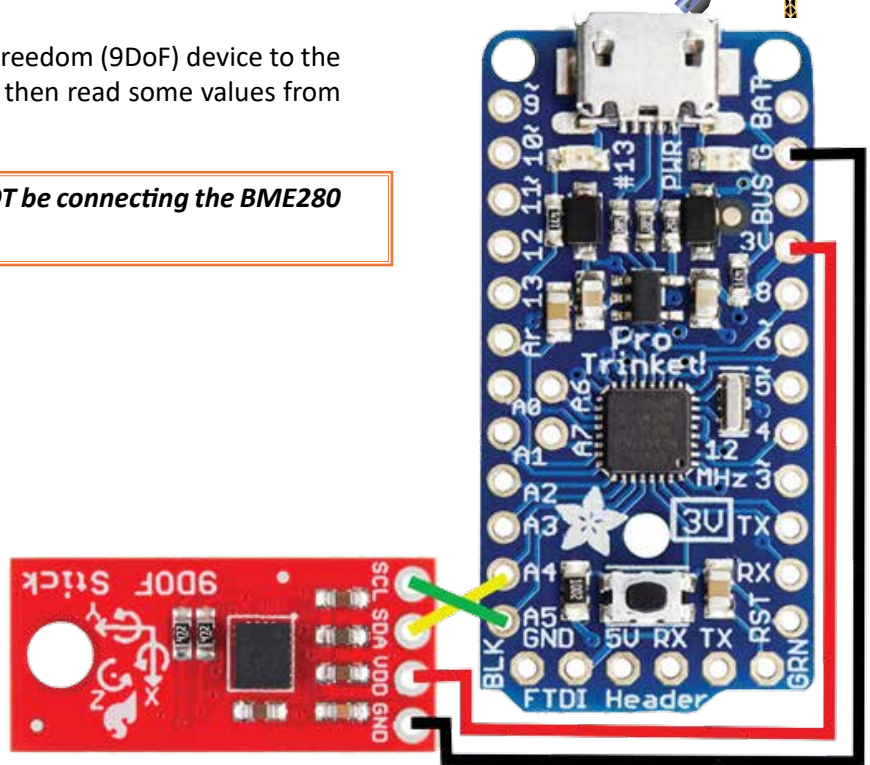
In this section, we will connect the 9 Degrees of Freedom (9DoF) device to the Arduino using a standard I²C connection. We will then read some values from the device and print them to the serial line.

Note: These instructions assume that you will **NOT** be connecting the BME280 sensor described in Section 4-B (p.4-5).

Part 1 - Wiring and Soldering

The 9DoF board communicates with the Arduino over an Inter Integrated Circuit (I²C) connection. The Serial Bus has two lines for data and does not use a clock or chip select line.

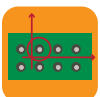
The goal is to wire the 9DoF board to the Arduino as shown in the schematic. Remember that the protoboards provide power and ground rails, so you don't need to use wires to bring power and ground to the pins on the protoboard.



The IC connections are:

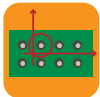
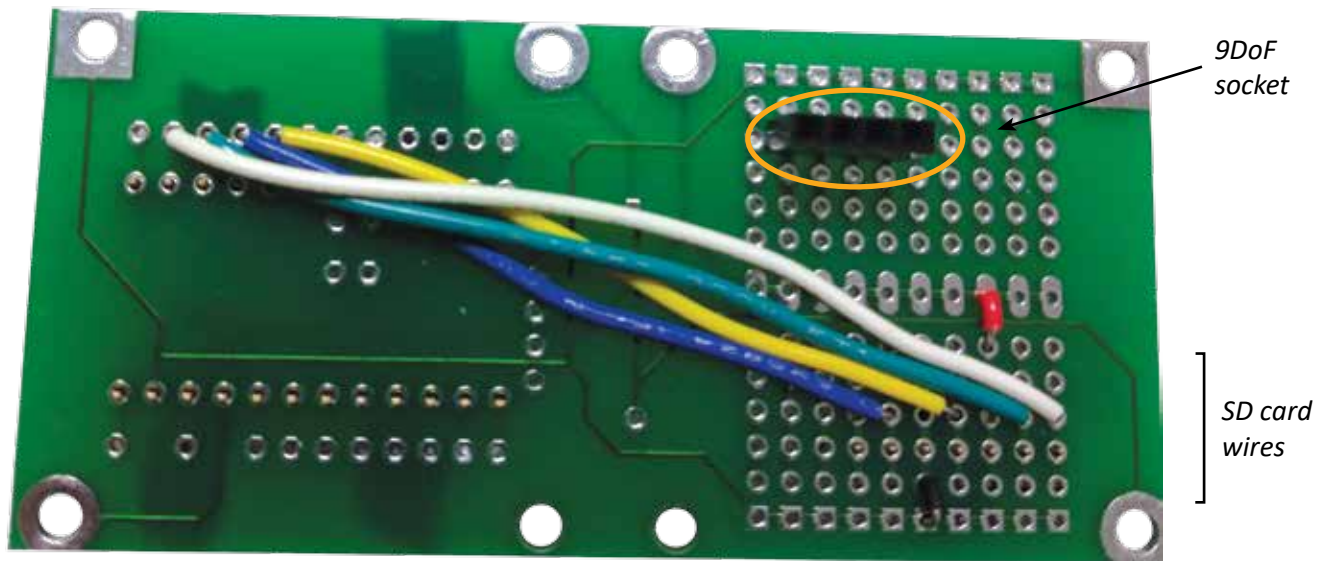
Color	Pin Start	Pin Finish
Red	VDD	Power Rail
Black	GND	Ground Rail
Yellow	SDA	A4
Green	SCL	A5

The 9DoF board will be plugged into a socket on the protoboard after some connecting wires are attached. Both the 9DoF and the wires will be attached to the bottom side of the protoboard. In the drawings below, we show one possible location for the 9DoF and the correct wiring for this location.



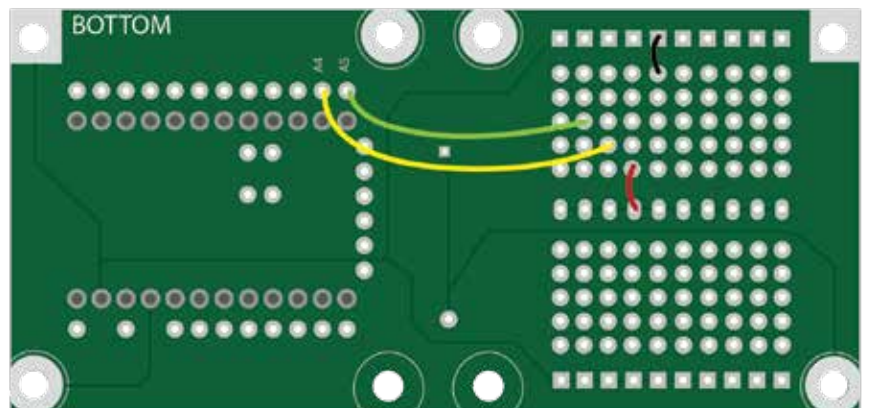
Place Socket

Insert the 4-pin socket into the protoboard as shown. Note the placement from the edge. The outside row is the power rail, then skip a row, then place the socket. Also note that the socket goes into the bottom of the protoboard (the side without writing.)



Place Wires

Attach power and ground jumpers to the rails and the yellow and green wires that connect the 9DoF to the Arduino. Note: SD card wires have been omitted for simplicity.



Check

Have a partner check your work.



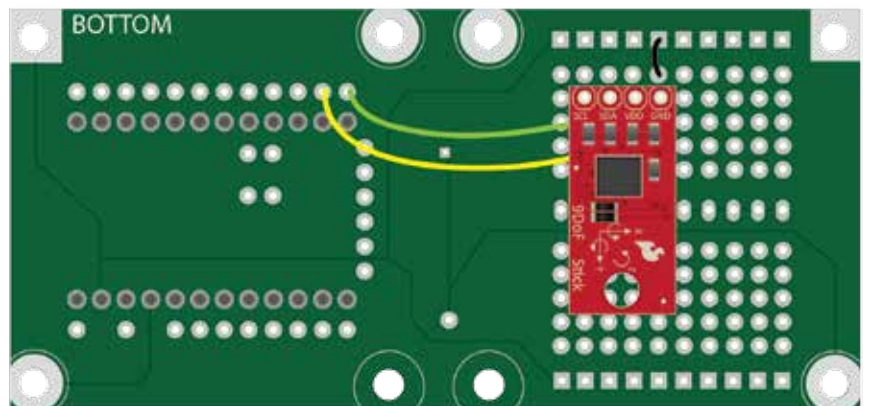
Solder

If your partner agrees that you have placed the wires and the socket correctly, you can solder these parts in place.



Test Software

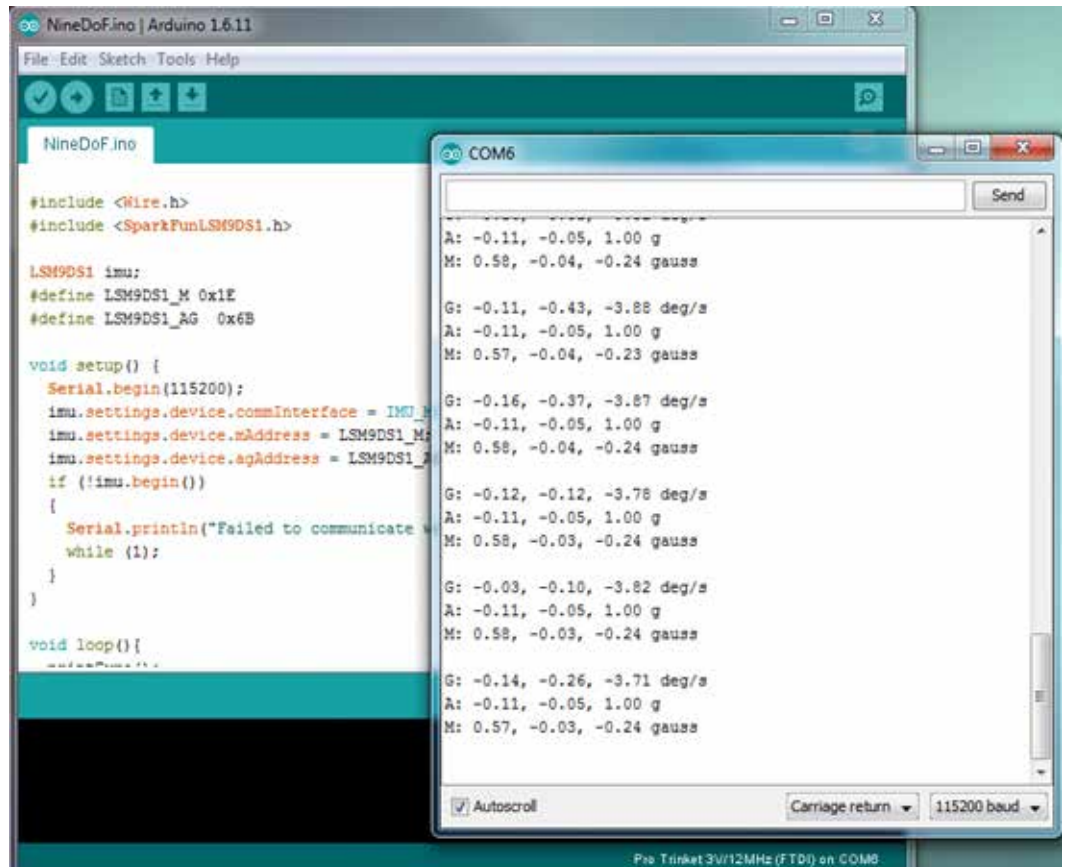
Plug in the 9Dof into the socket and proceed to the software test in Part 2.



Part 2 - 9DoF Software

Once the 9DoF is plugged into the board, 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 what effect that has on the sensor readings.

Also repeat the tests for the SD card and check to make sure the 9DoF data have been written to the SD card by copying the data onto your computer.



Appendix: Code Tour

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

These lines import the I²C 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 defines some objects we will reference later. LM9DS1 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 I²C bus.

9DoF I²C:

The I²C 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.

Our setup loop first initializes the serial port for debugging.

```
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);
  }
}
```

Next we set the 9DoF to communicate over the correct interface. Then, we 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.

```
printMag();
Serial.println();
delay(250);
}
```

Our main program loop prints the values from each of the sensors, 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");
}
```

This line of code does three things:

1. asks the sensor to take a reading;
2. prints text with the name of the sensor;
3. calls a set of nested functions.

Let's deconstruct this line of code:

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.

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

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

Gyroscope(x,y,z) measured in **degrees/second**

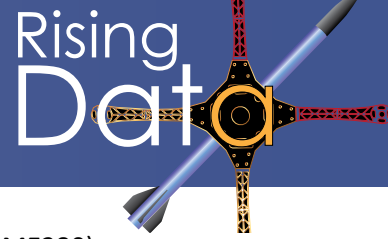
Acceleration (x,y,z) measured in **g**

Magnetometer (x,y,z) measured in **gauss**

Reality Check:

Look at the three values for the (x, y, z) components of acceleration. If the device is motionless, how should these 3 values relate to each other?

4-B BME280 Integration



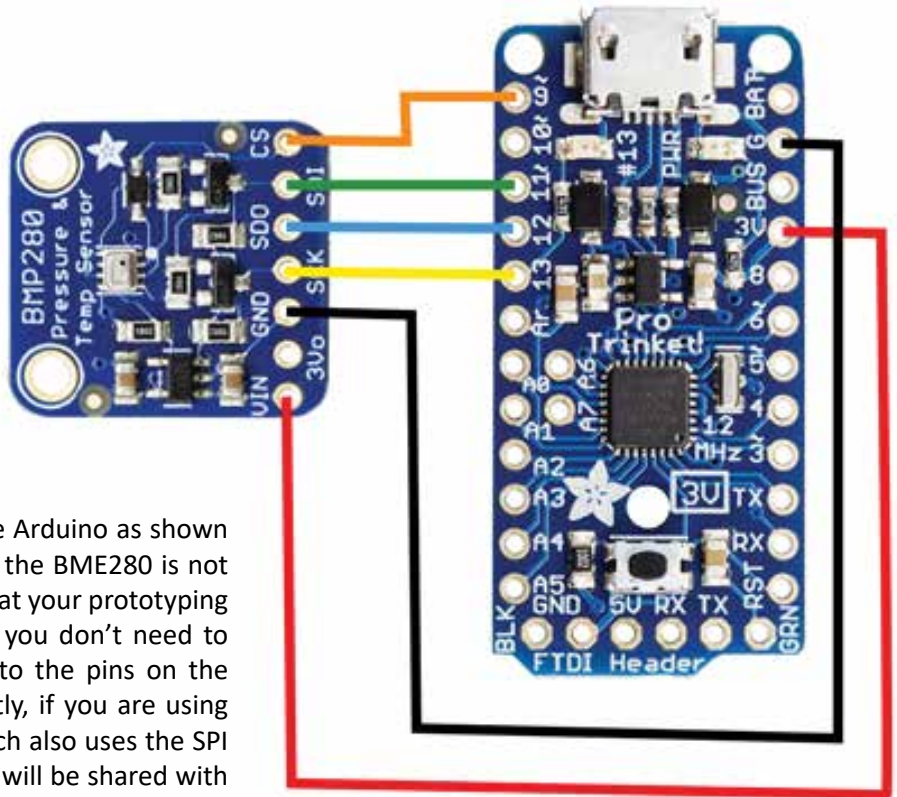
In this section, we will connect the Barometric Pressure, Temperature, and Humidity sensor (BME280) breakout to our Arduino by means of a standard SPI connection. We will then read some values from the device and print them to the serial line.

Note: These instructions assume that you will NOT be connecting the 9 Degrees of Freedom sensor described in Section 4-A (p.4-1).

Part 1 - Wiring and Soldering

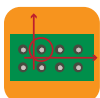
The BME280 board communicates with the Arduino over a Serial Peripheral Interface (SPI) connection. The SPI bus has three lines for data going from the Arduino to the BME280 (green), from the BME280 to the Arduino (blue), a shared clock line to ensure synchronous operations (yellow), and a Chip Select so the Arduino can specify which device on the SPI bus it wants to talk to (orange). Of course, the device also requires a 3.3V power line (red) and a ground line (black).

The goal is to wire the BME280 board to the Arduino as shown in the schematic. Note that that 3Vo pin on the BME280 is not used in this configuration. Also 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. Lastly, if you are using this device in concert with the SD Card (which also uses the SPI bus) the lines for the Clock, MOSI and MISO will be shared with the SD card, but the Chip Select (CS) line will not be shared.



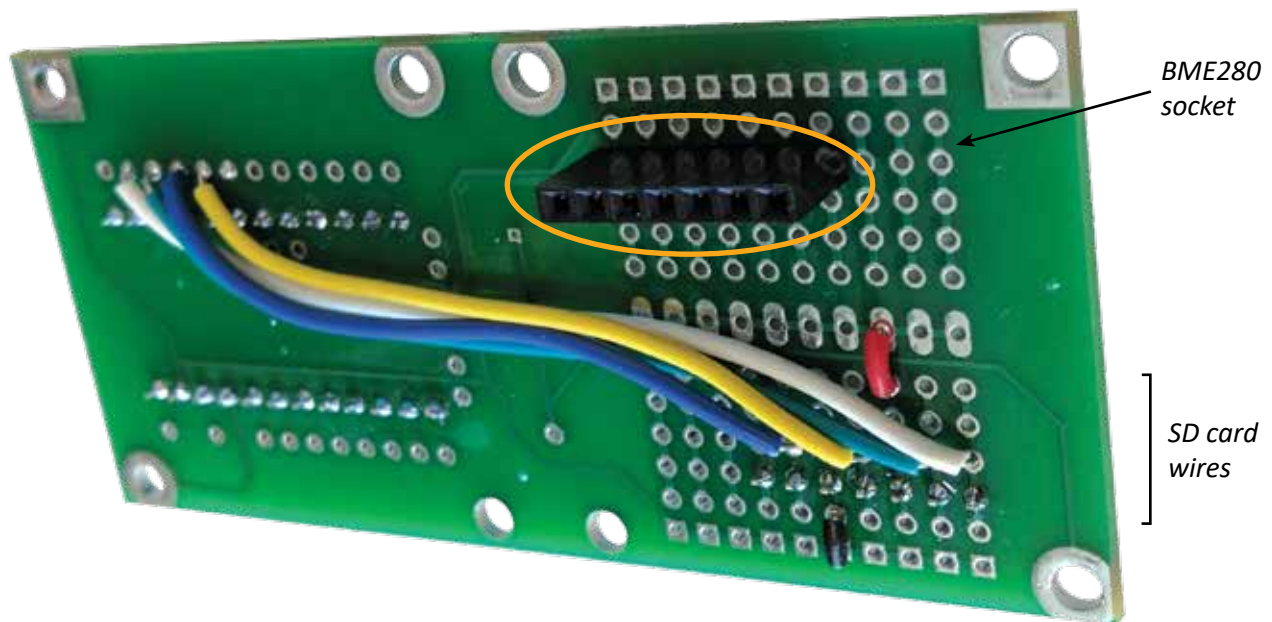
The SPI connections are:

Color	Function	Pin Start	Pin Finish
Red	Power	Vin	3V
Black	Ground	GND	G
Yellow	Clock	SCK	13
Blue	Master In/Slave Out	SDO	12
Green	Master Out/Slave In	SDI	11
Green	Select	CS	9



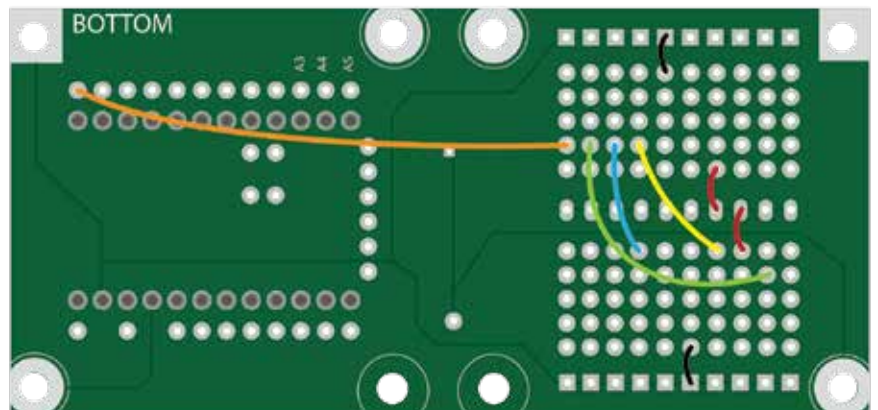
Place Socket

Insert the 7-pin socket into the protoboard as shown. Note the placement from the edge. The outside row is the power rail, then skip a row, then place the socket. Also note that the socket goes into the bottom of the protoboard (the side without writing.)



Place Wires

Attach power and ground jumpers to the rails and the yellow, blue and green wires that connect the BME280 to the SPI bus. Then connect the orange wire to the Arduino. Note: SD card wires have been omitted for simplicity.



Check

Have a partner check your work.



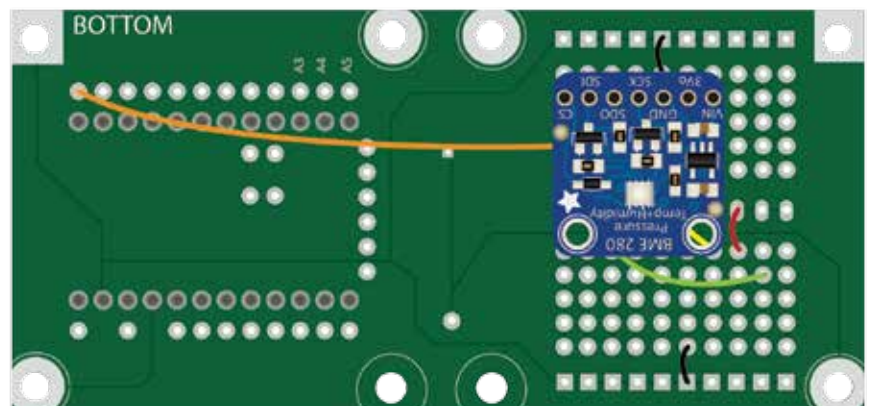
Solder

If your partner agrees that you have placed the wires and the socket correctly, you can solder these parts in place.



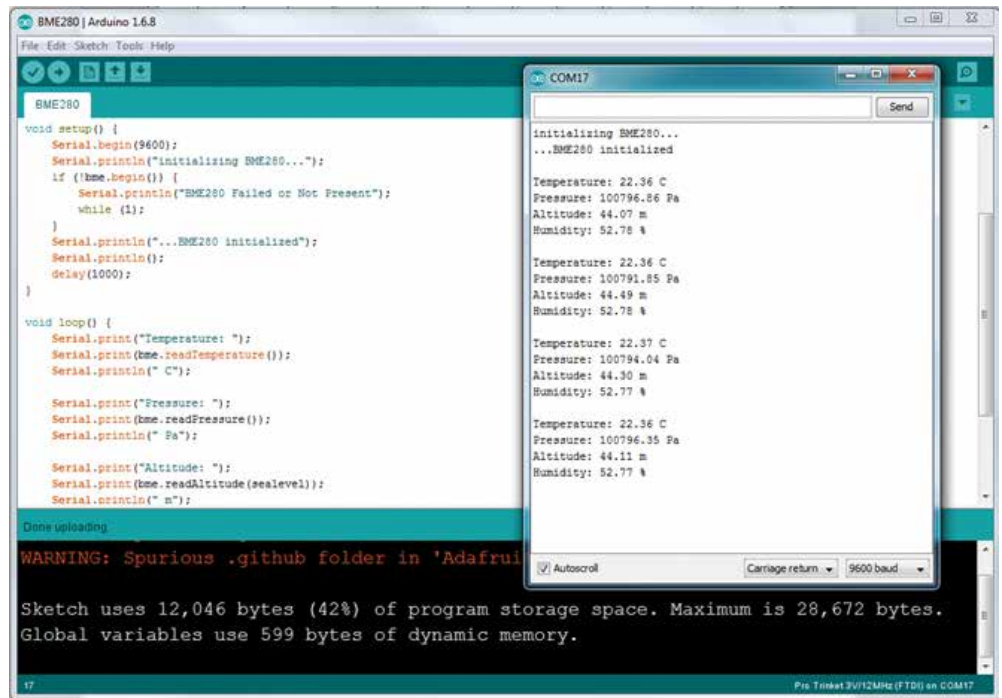
Test Software

Plug in the BME280 into the socket and proceed to the software test in Part 2.



Part 2 - BME280 Software

Once the BME280 is plugged into the board, open the sketch [RisingData/src/BME280/BME280.ino](#). You can upload the sketch as written and then open the serial monitor to see the output of the BME280 device. Feel free to move the device to areas of higher or lower temperature, humidity, or pressure to see how the output changes.



Appendix: Code Tour

```
#include <SPI.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>
```

The first line of code imports the SPI library. The second line of code imports the common libraries for all the sensors Adafruit makes. The third line imports the library specific to the BME280.

```
int chipSelect = 9;
float sealevel = 1013.25;

Adafruit_BME280 bme(chipSelect);
```

The first line specifies the chip select pin for this device is 9. The next line specifies the pressure at sea level: this is used by the chip to determine altitude. Finally, a new variable is created that represents the BME280.

BME280 SPI:

The SPI bus expects each connected device to have a dedicated select line. This means the devices don't need software addresses.

This makes the SPI bus somewhat more difficult to wire, but simpler in software. The devices on the SPI bus share the same lines for Clock, MISO and MOSI, so to choose a device (SD Card or BME280), we drive the select line low.

```
void setup() {
  Serial.begin(9600);
  Serial.println("initializing BME280...");
  if (!bme.begin()) {
    Serial.println("BME280 Failed or Not Present");
    while (1);
  }
  Serial.println("...BME280 initialized");
  Serial.println();
  delay(1000);
}
```

In the setup function the serial interface is initialized at 9600 baud. A message is printed indicating we are beginning the initialization process, and then we call the *bme.begin()* function to start the device. Depending on if we fail or succeed, a message is printed indicating the status of the initialization. We print a blank line and wait one second before continuing on success.

```
void loop() {  
  Serial.print("Temperature: ");  
  Serial.print(bme.readTemperature());  
  Serial.println(" C");  
}
```

In the main loop, we start by printing out the current value of the temperature. We use the *bme.readTemperature()* function to return the current measurement of temperature expressed in degrees C.

```
Serial.print("Pressure: ");  
Serial.print(bme.readPressure());  
Serial.println(" Pa");
```

Next we print the current value of the pressure. We use the *bme.readPressure()* function to return the current measurement of pressure expressed in pascals.

```
Serial.print("Altitude: ");  
Serial.print(bme.readAltitude(sealevel));  
Serial.println(" m");
```

Next we print the current value of altitude. We use the *bme.readAltitude()* function, and pass the pressure at sea level as an argument. It should be noted that the device is using the pressure to calculate the altitude, but you could calculate it yourself based on the pressure reading if you so choose.

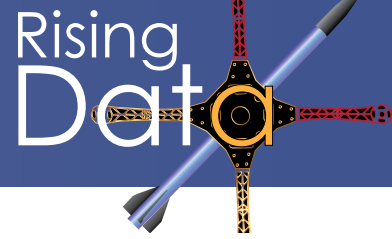
```
Serial.print("Humidity: ");  
Serial.print(bme.readHumidity());  
Serial.println(" %");  
  
Serial.println();  
delay(1000);  
}
```

Finally, we print the current value of humidity. We use the *bme.readHumidity()* function to return the current % humidity. We then print a blank line and wait 1 second before repeating the loop.

Reality Check:

Do the values for temperature, pressure, humidity and altitude make sense? If possible, compare them against known values.

5. TMP-36 Integration



In this section we will connect the TMP-36 analog temperature sensor to the Arduino. We will then read some values from the device and print them to the serial line.

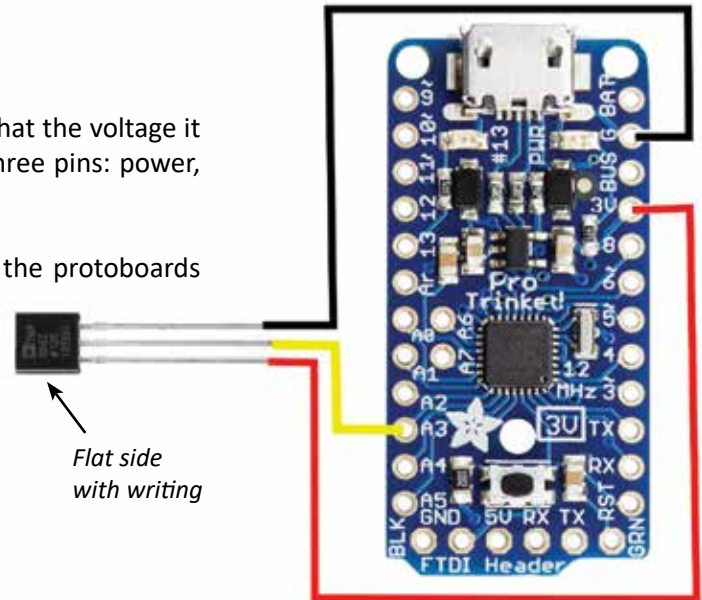
Part 1 - Wiring and Soldering:

The TMP-36 is a relatively linear analog device, which means that the voltage it puts out is proportional to the ambient temperature. It has three pins: power, data and ground.

Wire the TMP-36 as shown in the schematic. Remember that the protoboards provide power and ground rails, so you don't need to use wires to bring power and ground to the pins on the Arduino.

The connections are:

Color	Pin Start	Pin Finish
Red	Power	Power Rail
Black	Ground	Ground Rail
Yellow	Data	A3

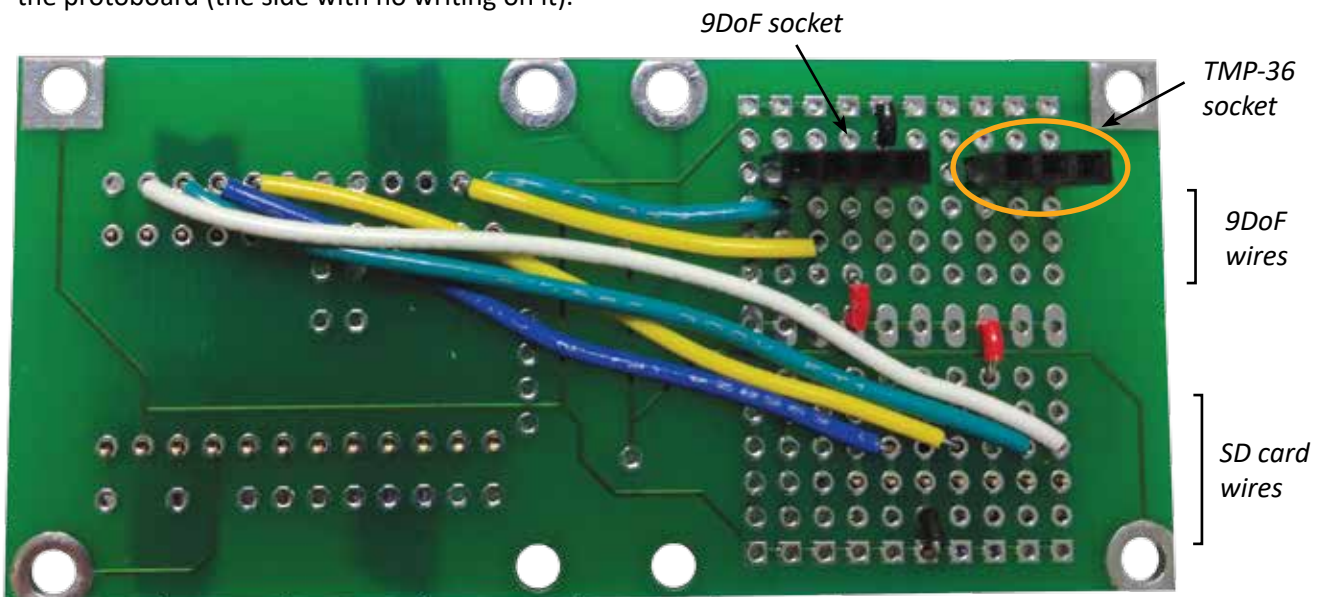


The TMP-36 sensor will be plugged into a socket on the bottom of the protoboard. It only needs one new wire to connect to the Arduino, and this wire will also be placed on the bottom side on the board.



Place Sockets

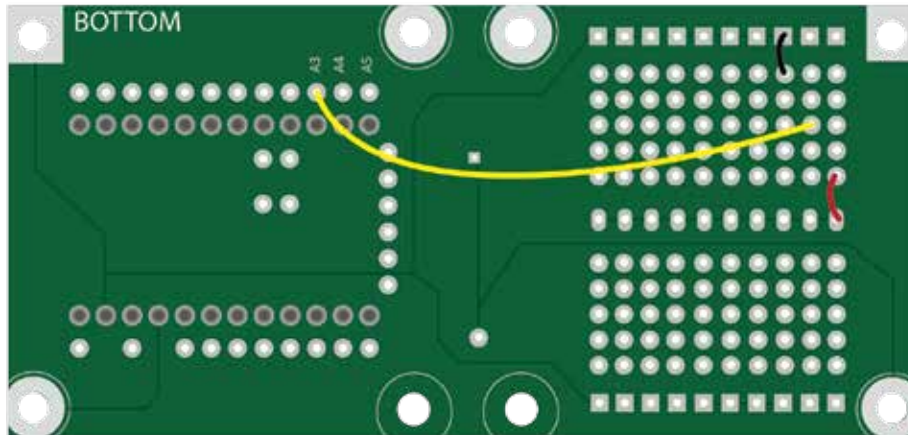
Insert the 3-pin socket into the protoboard as shown. Note the placement from the edge. The outside row is the power rail, then skip a row and place the socket. Also note that the socket goes into the bottom of the protoboard (the side with no writing on it).





Place Wires

Attach power and ground jumpers to the rails and the yellow wire from the data pin of the TMP-36 to the Arduino. Note: the 9Dof and SD card wires have been omitted for simplicity.



Check

Have a partner check your work.



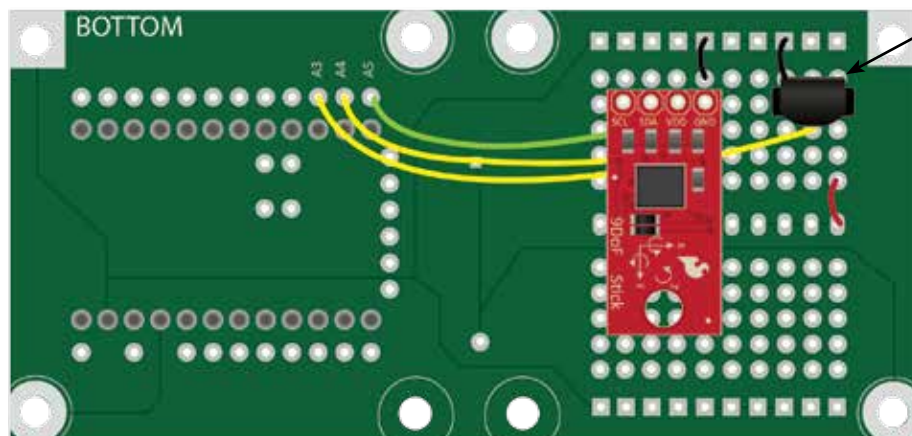
Solder

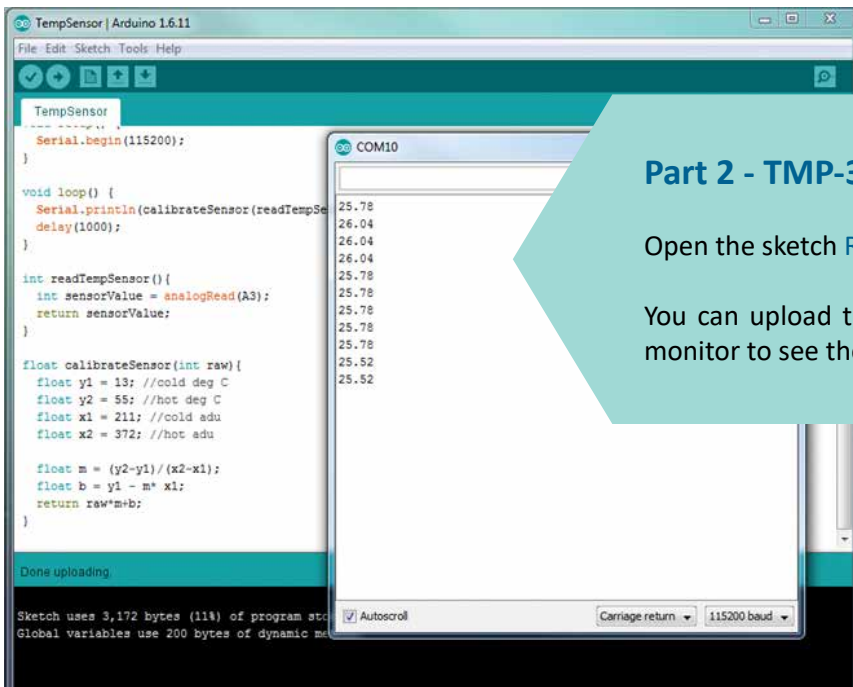
If your partner agrees that you have placed the wires and the socket correctly, you can solder these parts in place.



Test Software

Plug the TMP-36 into the socket and proceed to the software test. Be careful to plug the temperature sensor into the socket so that the flat side with the writing on it is facing out. If you plug it in backwards, it will rapidly heat up and destroy the sensor.





Part 2 - TMP-36 Software:

Open the sketch [RisingData/src/TempSensor/TempSensor.ino](#).

You can upload the sketch as written and then open the serial monitor to see the output of the TMP-36 device.

Also repeat the tests for the SD card and check to make sure the TMP-36 data have been written to the SD card by copying the data onto your computer.

Appendix: Code Tour

```
void setup() {
  Serial.begin(115200);
}
```

The only thing done in the setup is to start up the serial connection for debugging.

```
void loop() {
  Serial.println(calibrateSensor(readTempSensor()));
  delay(1000);
}
```

The main program loop reads the raw value from the temp sensor, calibrates it, and prints it. This is accomplished by using the following functions.

```
int readTempSensor() {
  int sensorValue = analogRead(A3);
  return sensorValue;
}
```

This function polls analog sensor 3 to which the temperature sensor is attached and returns that value as an integer.

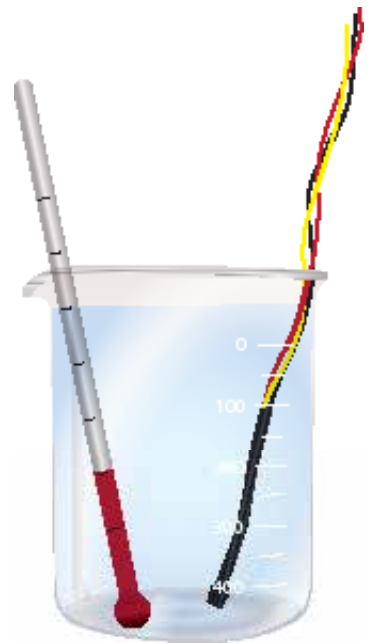
```
float calibrateSensor(int raw) {
  float y1 = 13; //cold deg C
  float y2 = 55; //hot deg C
  float x1 = 211; //cold adu
  float x2 = 372; //hot adu

  float m = (y2-y1)/(x2-x1);
  float b = y1 - m* x1;
  return raw*m+b;
}
```

This function expects an integer as input and uses a linear interpolation to calculate the appropriate value.

Calibration:

The Arduino TMP-36 code includes instructions that convert the Analog to Digital units (ADUs) returned from the Arduino into temperature in degrees Celsius. The numerical values that do this conversion were determined for a representative TMP-36 sensor, but you may wish to do your own calibration of the sensor to better understand this process. In order to do so, we have provided you with a special TMP-36 sensor that is on a long wire “leash” and that has been rubberized to allow it to be dunked into water of various temperatures. Here is one way to do the calibration:



1. Remove the TMP-36 from the payload and set it aside. Replace the TMP-36 with the specially created TMP-36 on a leash. This is an identical sensor that has been attached to long leads and protected with a layer of rubber so it can be safely submerged in water.
2. Prepare two cups of water, one boiling (or very hot) and one that contains many ice cubes. Be sure to use well insulated cups if possible to minimize the change in water temperature over the course of the calibration. Stick a normal thermometer into the hot water to accurately record the temperature in degrees Celsius. What should the temperature of the ice-cube water be?
3. Comment out the code that currently prints the temperature, and replace it with the following code that will print out the uncalibrated ADC value in ADUs:

```
void loop() {  
  //Serial.println(calibrateSensor(readTempSensor()));  
  Serial.println(readTempSensor());  
  delay(1000);  
}
```

4. Place both the leashed TMP-36 sensor and a known working thermometer in the first of the two cups. Run the TMP-36 software and monitor the serial data until you see the ADC reading reach some kind of equilibrium. Record the ADC number and the temperature.
5. Repeat step 4 in the cup at the other temperature.
6. Create a plot of Temperature vs ADU values that includes the two values you measured. (If you want, you can use additional cups at different water temperatures to add additional points to the fitting process, but two points will work pretty well if the points are significantly colder and hotter than room temperature.)

Fit a line that goes through your measured values that is of the form $y = mx + b$ where m is the slope and b is the y-intercept. In this equation, the variable x represents the ADU values, and the linear fit returns the temperature in degrees Celsius for y .

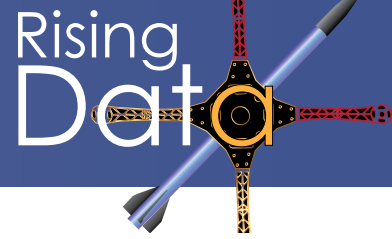
Note about ADUs: The Analog to digital converter on the Arduino has 10 bits, that span a range of 3.3 Volts. The Arduino converts the voltage read from the data line of the TMP-36 into ADUs using a linear relationship (so a reading of 3.3 V will return an ADU value of $2^{10}-1 = 1023$). We then further convert the ADU value into a temperature using a second linear relationship.

7. Replace the value in the `calibrateSensor` word for m (slope) and b (intercept) with the values you have empirically calculated.
8. Delete the line we added to display the uncalibrated ADC and uncomment the original code.

```
void loop() {  
  Serial.println(calibrateSensor(readTempSensor()));  
  delay(1000);  
}
```

9. Replace the leashed TMP-36 with your original one and run the program. You should now see reasonable and more accurate numbers for the temperature being output to the serial monitor.

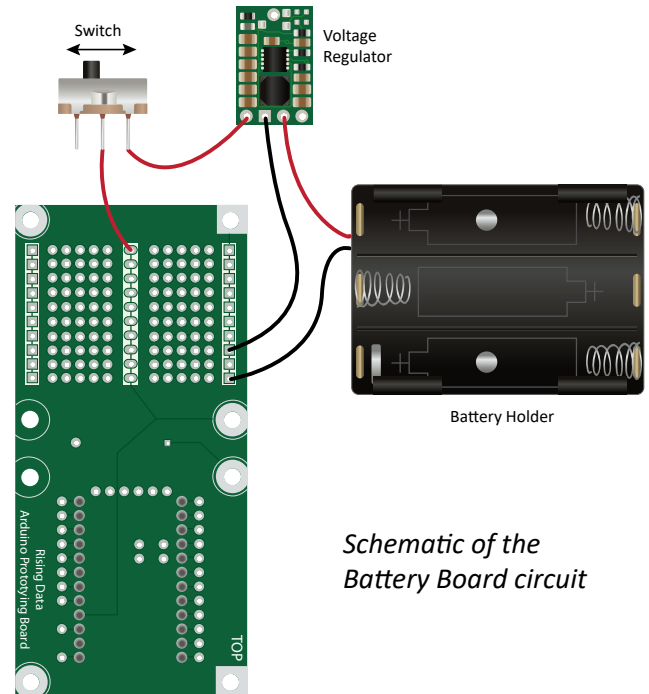
6. Battery Board Integration



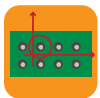
In this document we will connect the components of the Battery Board to deliver power from the batteries to the experiment. We will then use multi-meters to test that we can switch on and off a 3.3V power source at the exterior rails of the proto-board. The battery board does not include any software.

Part 1 - Placement and Wiring

If you follow the path of the power lines, in the schematic drawing to the right, you should see that power comes out of the battery and goes into the V_{in} pin on the voltage regulator. The input voltage to the pin can fluctuate, but the regulator will ensure that the output pin always gives us exactly 3.3V. The output of the regulator goes through a manual switch that allows us to open or close the circuit to control power to the entire experiment. Note that the switch has 3 pins; moving the switch from side to side controls which two pins are electrically connected.



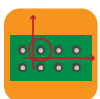
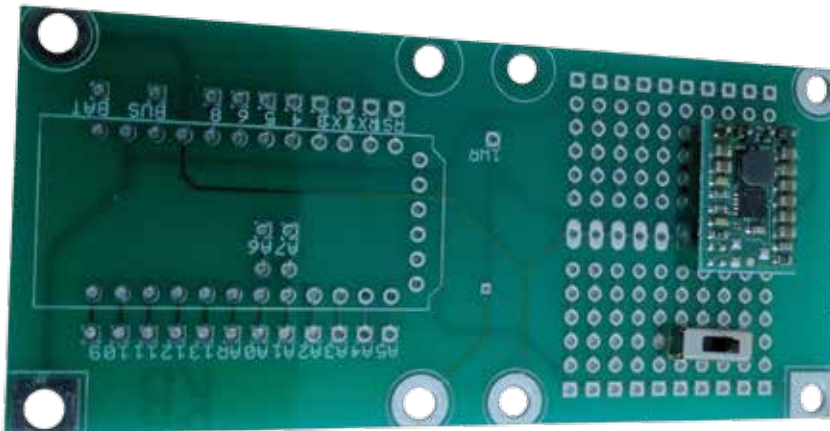
Schematic of the Battery Board circuit



Place Regulator and Power Switch

Place the regulator and power switch into the board on the top side. Note the alignment to the top of the board.

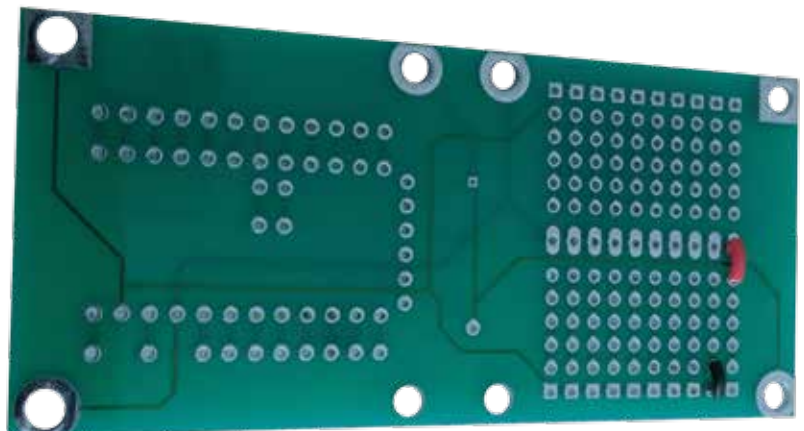
Make note of which pins align with which rows. Remove the switch and regulator when done.



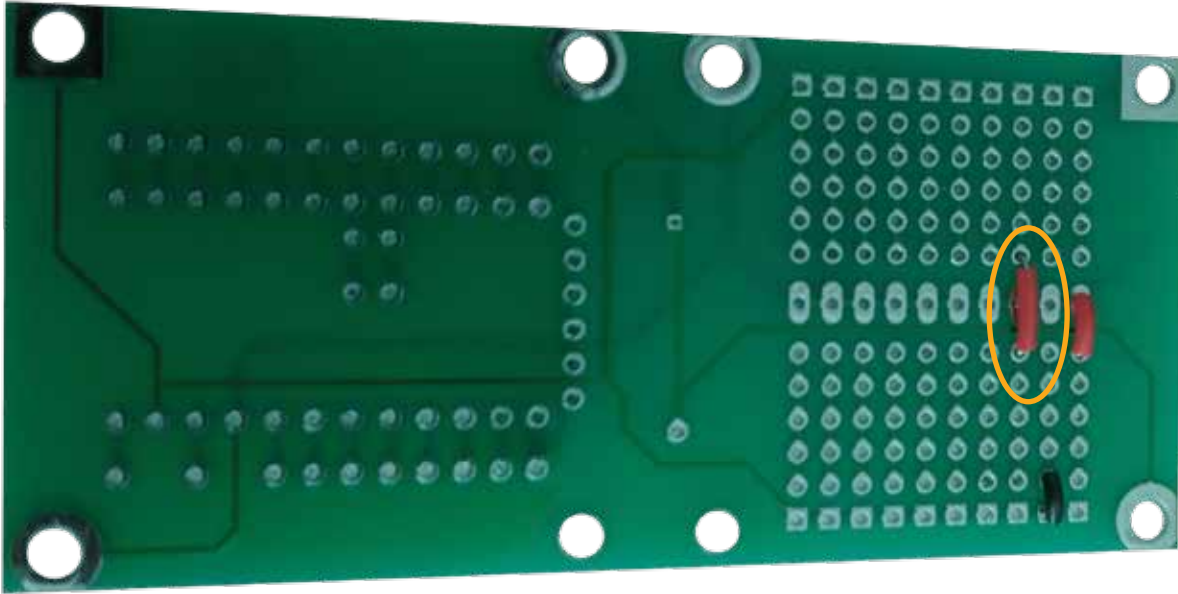
Place Wires

Strip and place wires on the bottom side of the board to connect the ground and power for the regulator.

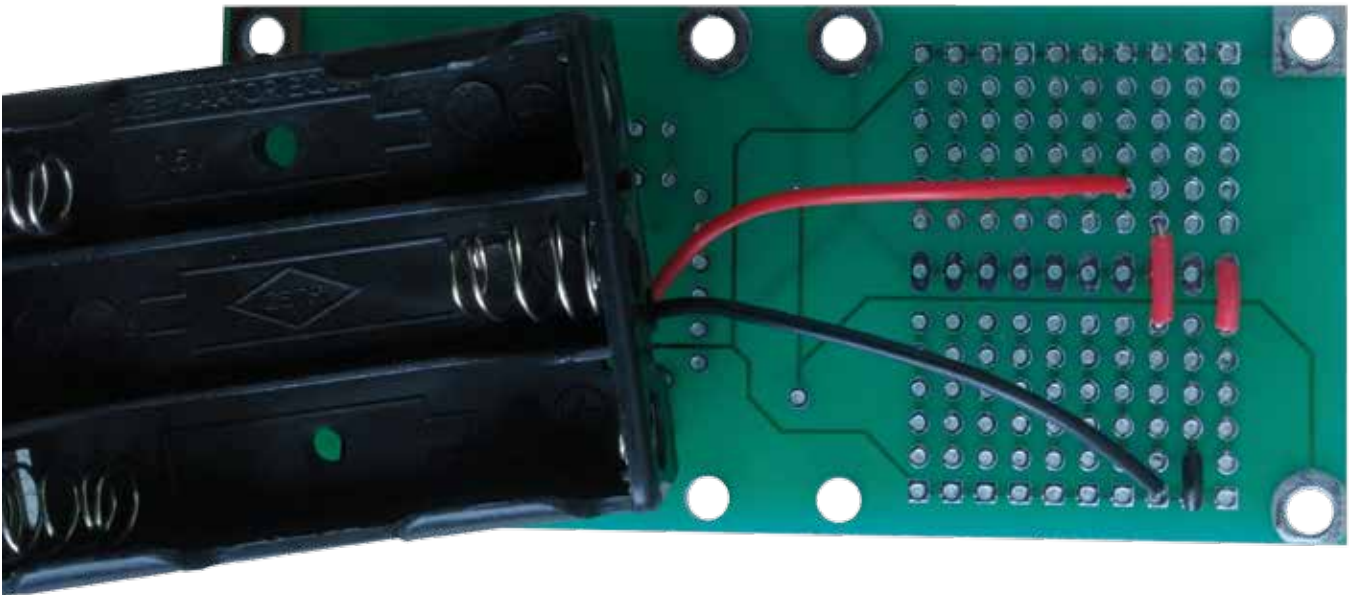
Cut wires to fit as cleanly as possible.



Place wire connecting the output of the switch to the input of the regulator.



Place the battery holder over the center of the full protoboard.
Cut and place the power and ground wires to the input on the switch and ground respectively.



Check the placement of your regulator and switch and ensure that you are not blocking the pins needed by those parts.



Check

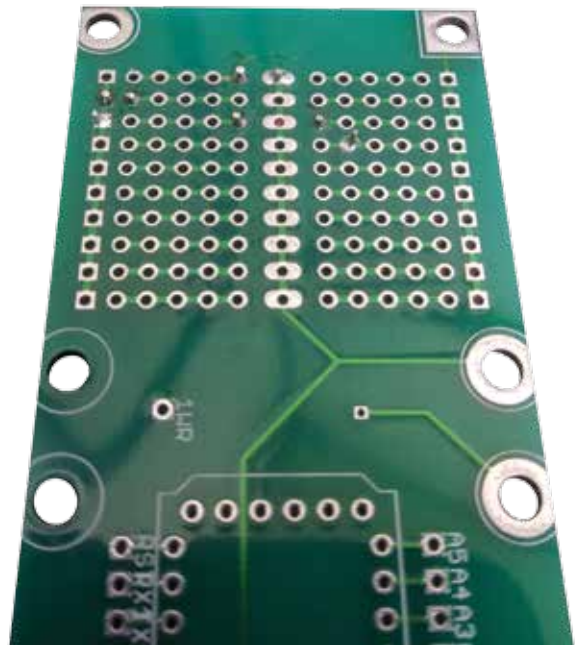
Have a partner check your work.

Part 2 - Soldering and Securing



Solder Wires

If your partner agrees that you have placed the wires correctly, you can solder these parts in place.
Clip leads after soldering.



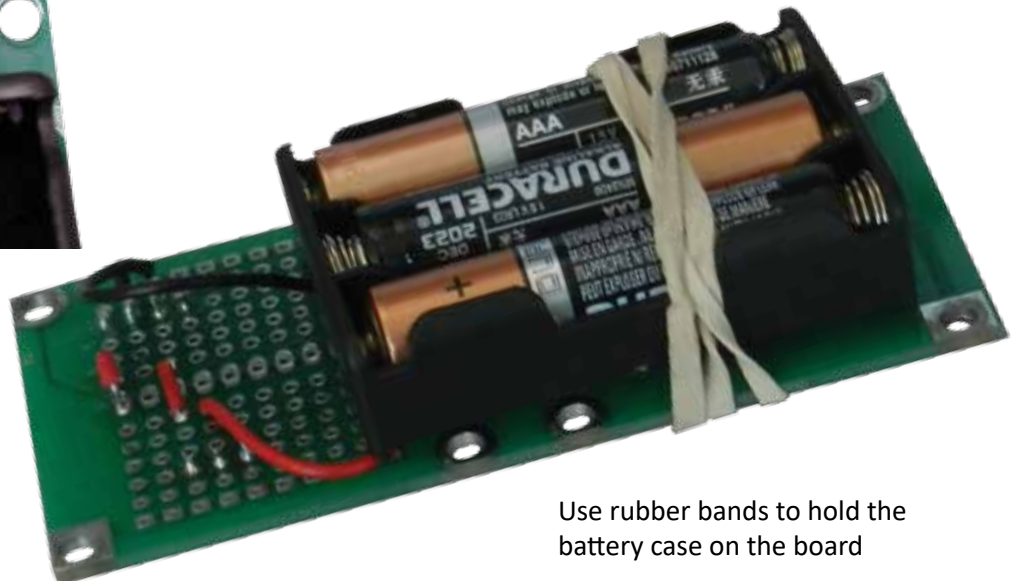
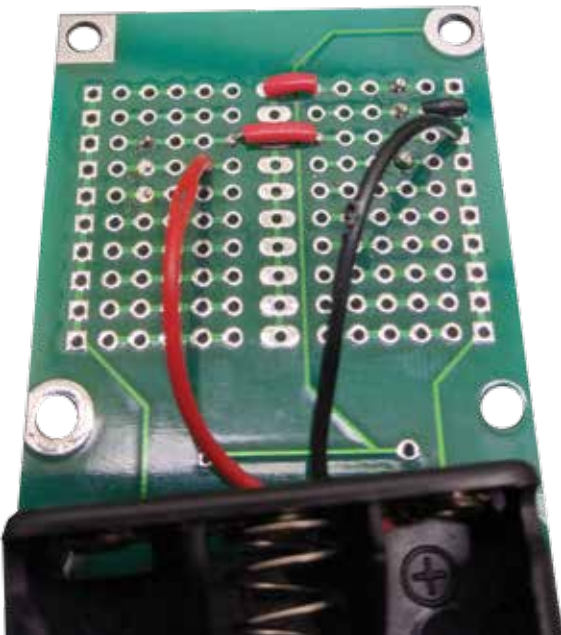
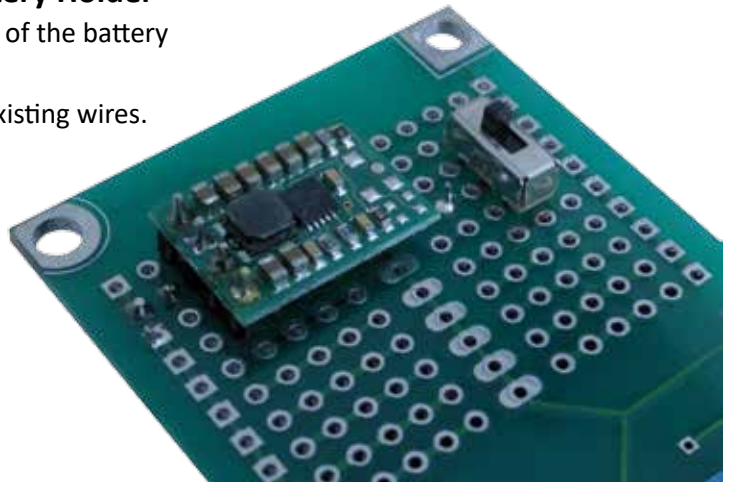
Solder/Glue Regulator, Switch and Battery Holder

Place the regulator and switch through the top of the battery board.

Double check that the placement aligns with existing wires.

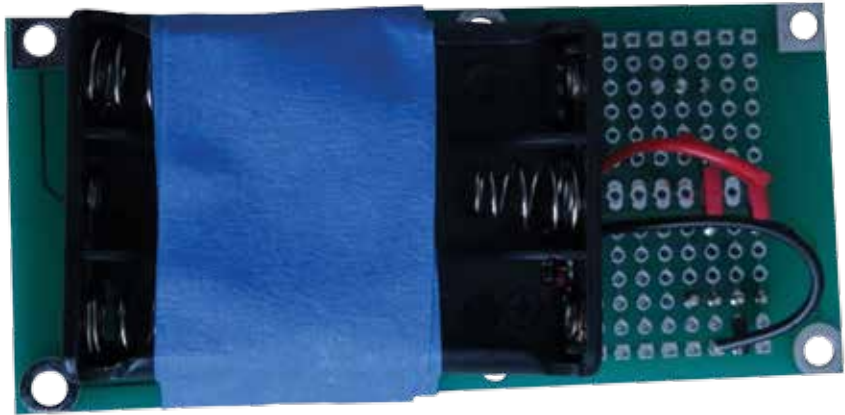
Solder parts in place.

Clip leads after soldering.



Use rubber bands to hold the battery case on the board

Make sure the battery holder has ample room around the corners for the threaded standoffs to pass through. Use tape to hold the battery holder in place while the epoxy dries (~ 1 hr)



Part 3 - Testing

Insert three AA batteries into the battery holder. This board should provide a 3.3v source to the power and ground rails when switched on, and no voltage when switched off.

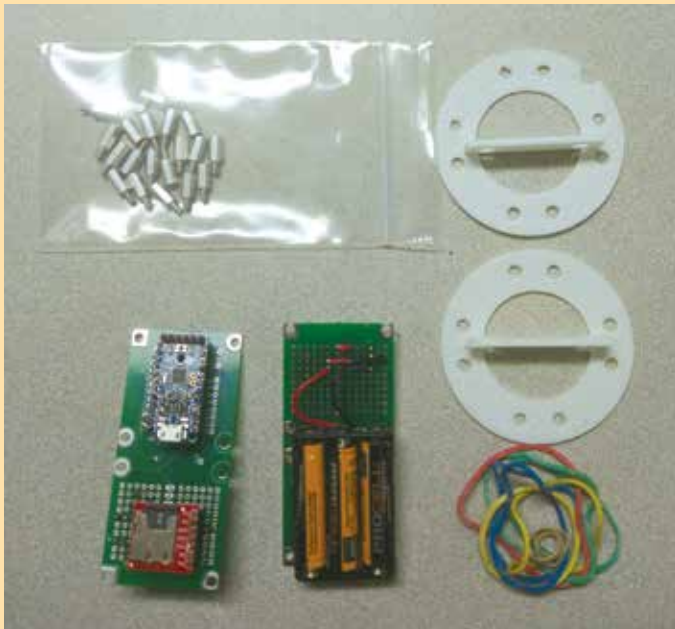


Test 1: With the device in the off position (switch towards the battery cradle) place the probes of your multi-meter through the power and ground rail holes. You should get a value very close to zero.



Test 2: Switch the device to the on position (away from the battery cradle) and you should see the voltage jump up to around 3.3V.

7. Building the Payload



Material Needed:

- 20 $\frac{3}{8}$ " Hex Standoffs
- Two 3D printed Sleds
- Eight Rubber Bands
- Battery Board
- Payload Board

Assembling the Payload

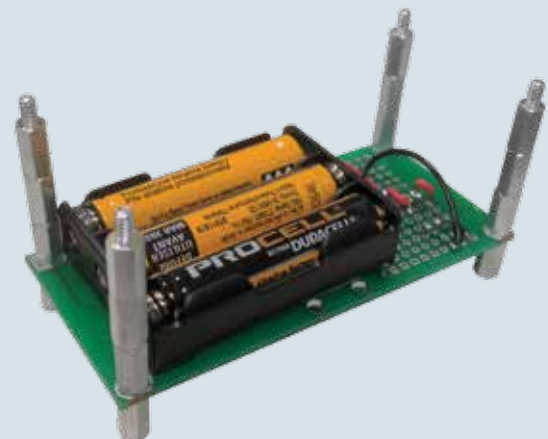
1. Using 12 of your $\frac{3}{8}$ " hex standoffs, create 4 $1\frac{1}{8}$ " standoffs by connecting 3 standoffs together as shown:



2. Place one $\frac{3}{8}$ " hex standoff through each of the outside holes on the battery board. Note that the battery side is facing up.



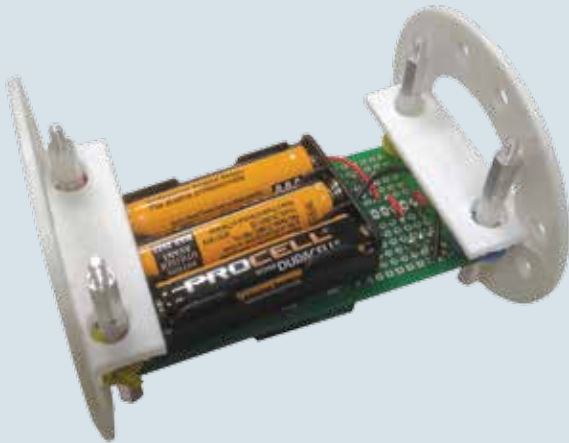
3. screw the $1\frac{1}{8}$ " rails onto the threaded standoffs as shown.



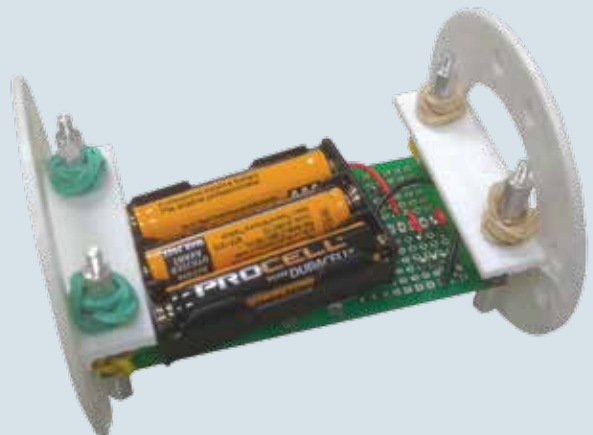
4. wrap rubber bands around the rails as shown



5. Place the 3D printed sleds over the rails as shown:



6. Wrap another set of rubber bands around the rails as shown:

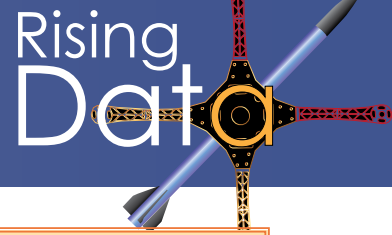


7. Place the protoboard as shown and secure it in place with the last four $\frac{3}{8}$ " hex standoffs



See additional photos of the finished payload assembly at the end of the guide.

8. Building the Rocket



Introduction

The following are steps and methods to make building and modifying the Estes Big Daddy rocket easier.

The modifications are necessary to be able to fly the Rising Data payloads.

There is a Youtube video available: “[SSU Rising Data](#)” that shows the construction of the engine mount (the hardest part of the build) and the modifications to the rocket.

Do Not Attach The Rubber Shock Cord!

Refer to the shock cord, parachute and recovery system section covered in the modifications segment and in the video.

Note: you will not need the yellow and the orange tubes provided in the kit.

Material, Tools and Supplies Needed:

- Estes Big Daddy Rocket Kit and modification parts bag

Needed in addition to the Rocket Kit:

- | | |
|-----------------------|------------------------------|
| • Pencil | • Metric ruler |
| • Drill with 9/64 bit | • Aluminum Scrubbing Pad |
| • Glue | • Sandpaper |
| • Hacksaw | • Scissors |
| • Hobby knife | • Small Phillips screwdriver |
| • Masking tape | |
| • Ruler | |

Assembling the Engine Mount

Remove the following from the small plastic bag in the rocket kit:

- B Engine mount tube: Light brown tube
- D Engine hook: Flat bent metal piece
- C Green engine block: Short green tube

1. Using a ruler measure from one end of the engine mount tube and make marks at 6 mm, 25 mm, 8.9 cm and 9.8 cm.



2. Make a small incision with a sharp knife on the third mark (8.9 cm from the bottom).



3. Place the end of the engine hook with the 90° bend into the incision you just made. Lay the rest of the engine hook against the tube.



4. Use a piece of masking tape around the engine mount tube at the 25 mm mark to hold the engine hook in place.



5. At the end of the tube where your 9.8 cm mark is, place a light ring of glue *inside* the tube. You should be able to see the engine hook sticking through the tube.



6. Now take the green engine block tube and place it into the glue. Push it down until it contacts the protruding engine hook inside the engine mount tube.



7. Remove the centering rings out of the white cardboard and then remove the center pieces.



8. Place a ring of glue on the first mark (5mm) and the fourth mark (9.8 cm) of the engine motor tube.



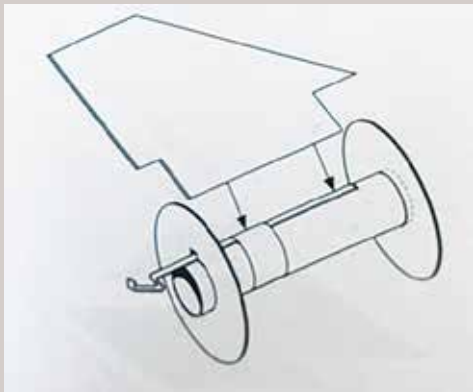
9. Insert the brown tube through the centering rings lining them up with the marks on the tube. The notched ring goes on the bottom, and the un-notched ring goes on the top (the end with the green ring). Reinforce with some glue and let dry COMPLETELY.



You are now done building the engine mount.

Preparing the Fins

1. Punch out the fins from the balsa wood panels
2. Sand the fins as desired.
3. Take the rectangular fin inserts and test to see if they fit perpendicularly between the two centering rings of the engine mount. Do not glue!



Mark the Main Tube

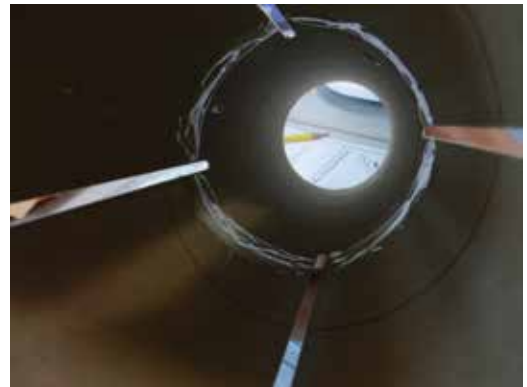
1. Draw a straight line from top to bottom of the main body tube between two of the fin slots.



2. Make a mark 13 mm from the top of the fin slots.



3. Put a ring of glue on the inside of the body tube at the mark.



4. Insert the entire engine mount into the bottom of the body tube. It is in the correct position when the top centering ring of the engine mount is aligned with the previously made mark (13 mm from the top of the fin slots).



5. NOTE: the engine hook must line up with the straight line made in Step One of this section.



Do Not Install The Shock Cord at this time.

Attaching the Fins

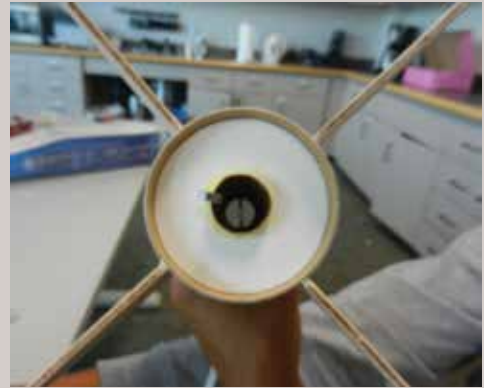
1. Apply a thin layer of glue on the edge of the fin with the rectangular insert.



2. Insert the fin into a fin slot of the body tube. Glue both sides of the fin where it meets the body tube.



3. Repeat for all four fins. Make sure that each fin is perpendicular to the rocket body.



Attaching the Launch Lug

1. Make a mark 11.4 cm from the bottom of the Rocket Body. Make sure that this mark is along the “top to bottom” lengthwise mark you made in a previous section.



2. Apply a dab of glue on the launch lug, and place it on the lengthwise mark, going up from the mark from step one (11.4 cm from the bottom). Use a piece of masking tape to hold this until the glue dries.



Let the glue dry...

Modification as shown on *YouTube*

The instructions below are steps selected from the video. It is strongly recommended to follow along: "[SSU Rising Data](#)"

1. The masking tape marks where to cut the base off the nose cone.



2. Insert the disc to one end of the black shock cord. Tie four or more overhand knots to secure it.

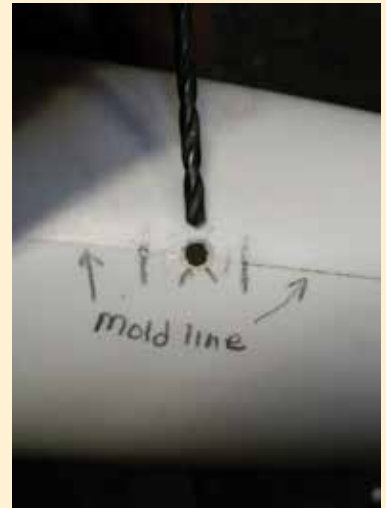


- 3a. Place the white disc in the nose cone. Hold up the cone to the light to see the disc's edges and mark two lines on either side by the cone's mold line. Repeat on the opposite side of the cone.



- 3b. Remove the disc. Then use a 9/64 bit and drill the nose cone on the mold line. Repeat on the opposite side of the cone.

Then use the screws to secure the disc.



4. Tie the orange streamer to the shock cord 16 cm from the nose cone.



5. Attach the baffle (scrubbing pad) to the other end of the black shock cord at about 30 cm from the end.



Final result of the modification assembly



Attaching the Shock Cord

1. Cut out the strip of paper (which will serve as the shock cord mount) with a pair of scissors.



2. Place a dab of glue on the second section of the piece of paper, and place one end of the black shock cord on the glue.



3. Fold the first section on the second and let dry.



4. When dried, place a dab of glue on the third section and fold the folded first and second sections on the third section on the third section.



5. Next, place a dab of glue on the inside of the white tube, 8.9 cm from the top.



6. Place the paper shock cord mount onto the dab of glue on the inside of the rocket and let dry.

Attaching the Parachute

1. Cut the rubber shock cord in half. Attach one end to the cross bar on the 3D printed end of the payload "sled".



2. Find the centers of the parachute's shroud lines. Tie them into a loop as a group. Tie the remaining end of the rubber shock cord to the loop you just formed.



3. Test the fit of the nose cone on the body. It should not be too tight or too loose, but just right!

Final Assembly

1. Finally, fold the parachute, and place it inside the body of the rocket.

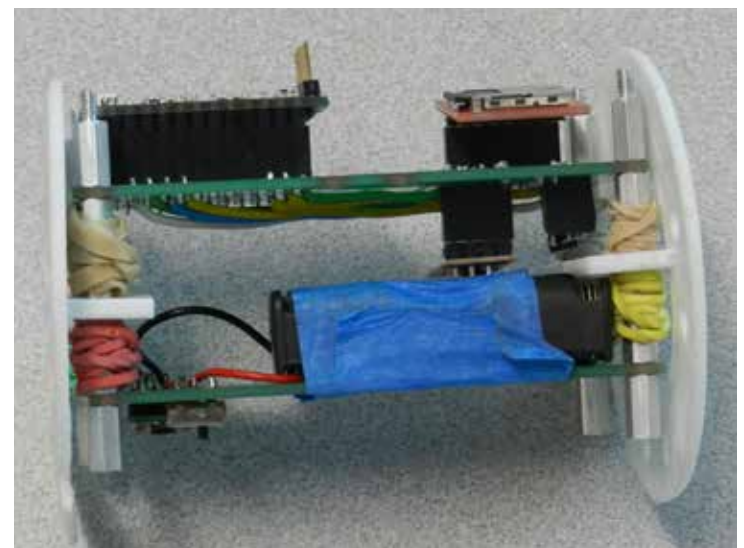
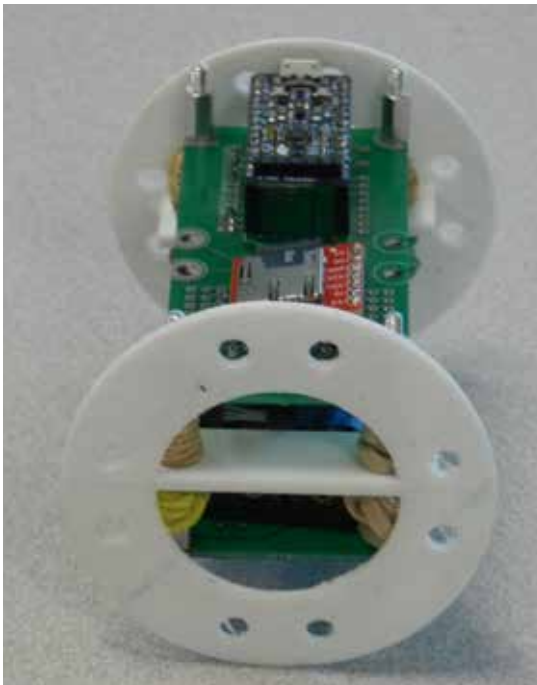
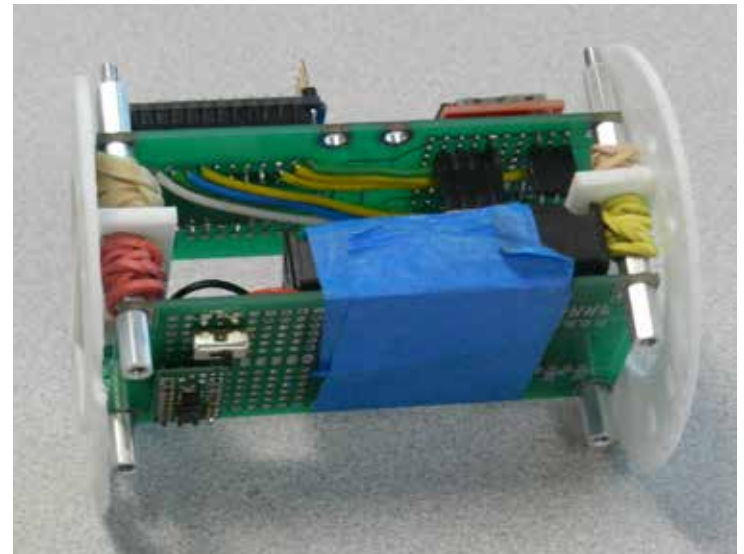
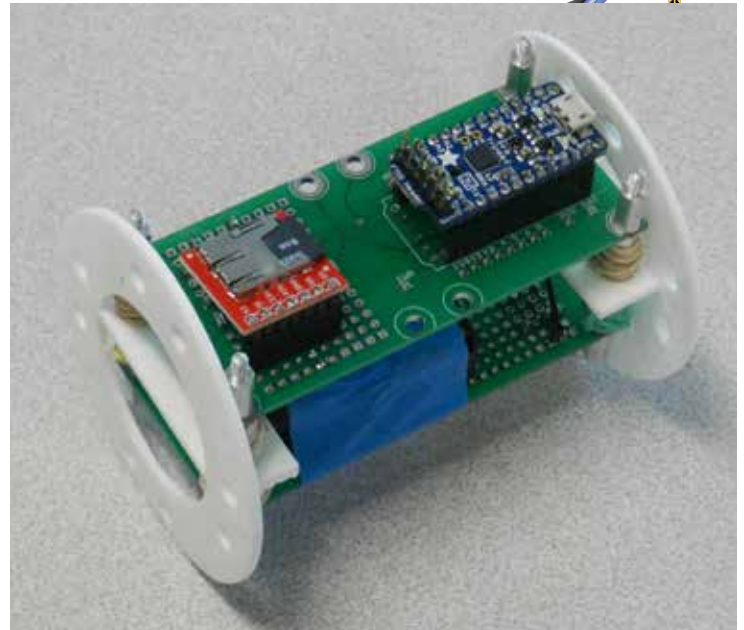
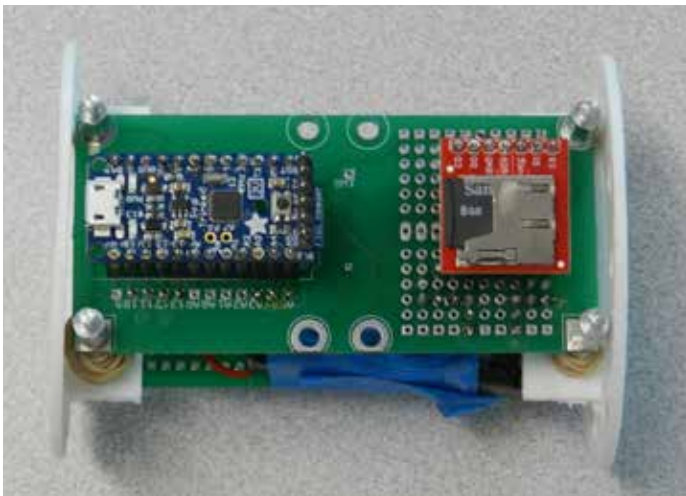


2. Afterward, close the body cavity by placing the nose cone inside the top end of the rocket body.

Congratulations, you have built a Big Daddy Rocket!



Finished Payload Photos





lbym.sonoma.edu/RisingData