Charlie Nguyen

ECE 480

Application Note

Friday, Nov 4

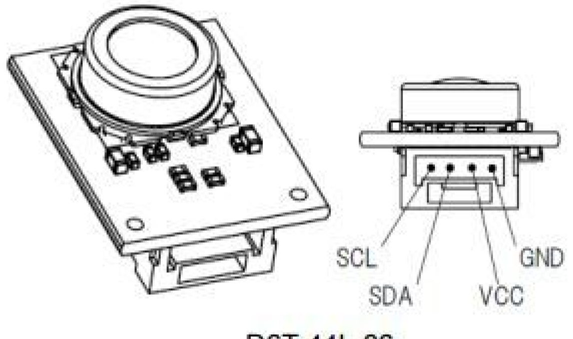
How to integrate an Omron D6T thermal sensor with an Arduino

Table of Contents

1. Introduction

Hello! This tutorial will teach you how to control a D6T thermal sensor with an Arduino microcontroller; then, use that information to output a graphical display using Processing software. If you are reading this because it is interesting, mandatory, or out of sheer boredom then you have come to the right place! If you are confused about any of this right now then do not worry, everything will be explained clearly with ample references: even for the most inexperienced.

2. What is a Omron D6T-44L-06 Thermal Sensor?



D6T-44L-06

This is a very sensitive infrared temperature sensor. Instead of measuring temperature on a single point, it can detect it in an array of pixels. This one in particular has a 4x4 pixel; Omron also has a 8x1 pixel array and is coming out with a 16x16 in the near future. This device uses an I2C, or communicates through a serial port, connection to exchange information. In this case, we will be using an Arduino. If you want more details on the specification and datasheet on the D6T then look at reference I. Also, this guide is not intended to teach you how to completely use I2C, so if you are interested in learning more about that then the information on I2C communication is provided in reference II.

3. What is an Arduino?

An Arduino is an open-source microcontroller; this was chosen because it is easy to learn, has a ton of documentation, and is very cheap. This microcontroller can take input from any digital or analog device and process that information and output it digitally. We can use the Arduino to take in information, in the form of bytes, from the D6T and have it process and output something desirable. In my group project for MSU, we are using it to detect human presence from temperature and making an alarm sound. Reference III and IV shows more information on the Arduino and tutorials on how to use an Arduino if you are new at this.

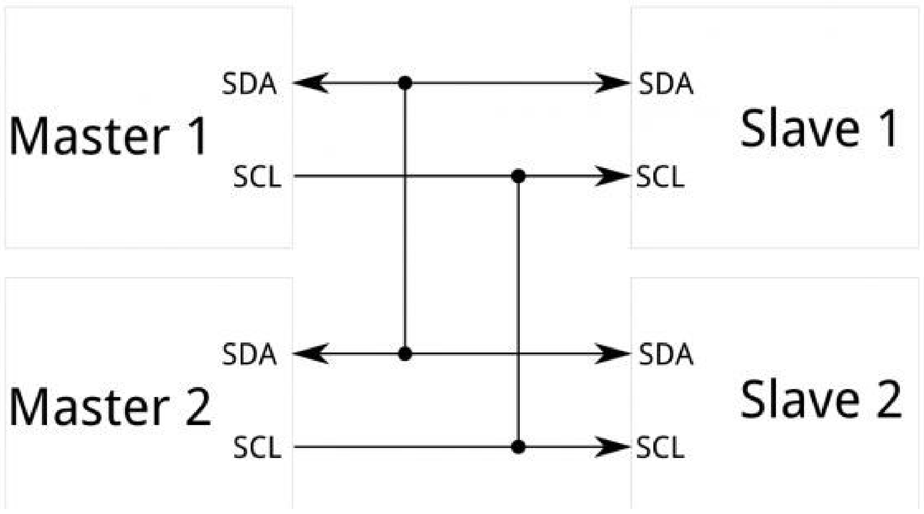
4. What can we use these things for?

There are many applications where this can be useful. Like I mentioned before, we are using this to detect a human presence in a steel mill and using that information to turn on an alarm to alert the operator of any danger. The D6T is intended for human detection but can be used for anything that requires temperature as a reading: fire safety, power conservation, and security comes to mind first.

5. What do you need to get started?

1. Any Arduino board that has a RX and TX pins
2. An Omron D6T Temperature Sensor
3. A breadboard or some type of perf board to join connections
4. 2x 10k Ohm resistors (used to pull up)
5. A few jumper wires to make the connections
6. Arduino IDE environment (www.arduino.cc)
7. Processing IDE environment (www.processing.org)

6. Understanding how the D6T communicates

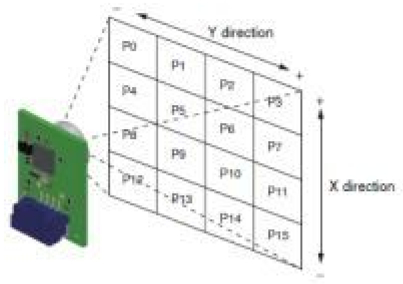


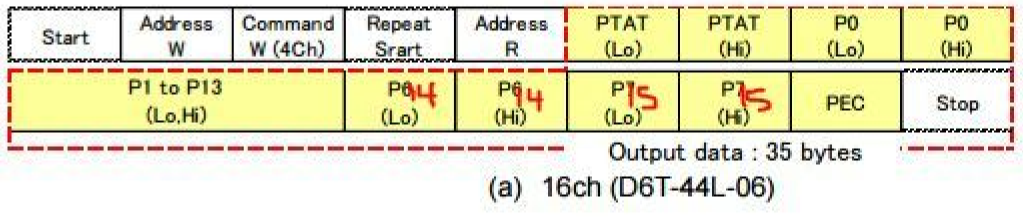
The Omron D6T thermal sensor uses I2C to communicate between itself and a micro controller. This protocol allows serial “master” circuits, like the Arduino, to talk to several “slave” circuits; slave circuits can be the D6T and any other circuits that use the same I2C protocols. This way, multiple D6Ts or other I2C chipsets can be connected in parallel and communicate with the arduino. Above shows a simplified diagram on how that works.

To establish a connection, the master will send a signal to a specific slave ID to say “hey, I want to communicate with you and not the other I2C in the circuit.” Thus, the D6T has a specific ID, which is 0x0A, you must connect to. After the connection has been established, it is time to talk to the D6T. We can write a command to ask it to do something. In this case, we will ask it to send the temperature data through the serial line; the command is 0x4C. After the command has been sent, the transmission must be ended. This way, we can wait for the data to go through on the serial line and then we can read the buffer.

The sensor sends 35 bytes of data. Each temperature data has two parts, the Low and High significant bytes of the data. The full temperature data uses a 16 bit signed that is 10 times the value of the celsius temperature. For example: the high byte data 0x00 and low byte data of 0xFA will translate to 0x00FA which is equivalent to 250. Dividing that by 10 will be 25.0 C or 77 F.

However, if you have counted 16 pixels with 2 data bytes each, you have probably asked why is is only 32 bytes of data. What are the other 3 bytes doing? Below is a picture of how the data bytes are organized that comes directly from the datasheet.





Start	Address W	Command W (4Ch)	Repeat Srart	Address R	PTAT (Lo)	PTAT (Hi)	P0 (Lo)	P0 (Hi)
P1 to P13 (Lo,Hi)			P6 14 (Lo)	P6 14 (Hi)	P7 15 (Lo)	P7 15 (Hi)	PEC	Stop

Output data : 35 bytes

(a) 16ch (D6T-44L-06)

- Bytes 1-2 are the PTAT data
  - The internal reference temperature
  - This is the temperature around the device
- Bytes 3-34 are the temperature data of each pixel respectively
  - E.g. Bytes 3-4 are Pixel 0's low and high byte
  - E.g. Bytes 4-5 are Pixel 1's low and high byte
  - E.g. Bytes 6-7 are Pixel 2's low and high byte
- Byte 35, the final byte, is the PEC
  - Packet error check byte

Once the thermal sensor has sent the information into the Arduino's buffer, the information can then be read and processed to do what we want. In this case, have it graphically output into the monitor screen. The full understanding of the I2C protocol goes beyond the scope of this article, but the reference V has a link to a more in depth tutorial on how it works.

7. Connecting the Arduino to the D6T

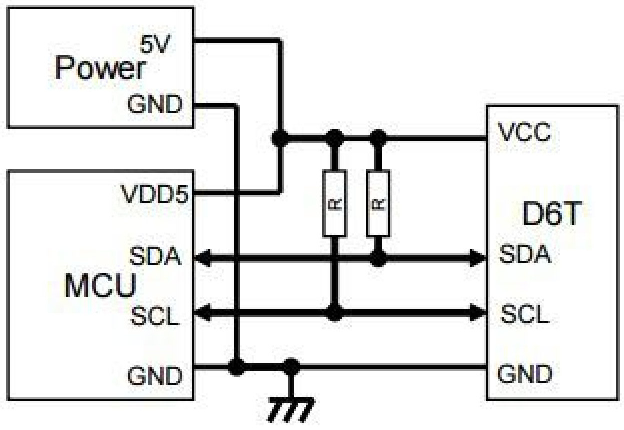


Fig.7 (a) Direct connection

The thermal sensors input for power and signals use 5V. Most Arduino boards have 5V in the the analog and digital output pins. So, making the connection just requires the data line, clock line, and also a 5V line to be directly connected to the Arduino like above. Please note the resistors that are used as a pull up resistor to the 5V rail. You can look up "pull-up resistors" if you are curious to why they are there. On another last note, some Arduinos, usually any mini version, are 3.3V outputs so you will have to have the D6T's data and clock lines linked with a pull-up resistor directly to a 5V power source instead.

8. Setting up your Arduino IDE

Previously, it was mentioned that the D6T will send 35 bytes of data into the internal buffer for the Arduino to process; however, the Arduino's default max size for the internal buffer is only 32 bytes. Any more data than that then it is simply ignored. Arduino has done this because they want to make it universal for all of their boards. Some of their boards are very tiny, and thus have much less memory to work with. To get around this, we have to increase the internal buffer max size. Below I have described how to do that using different OS systems. Note: if you are using linux, then it should be very similar to the MAC OS X.

MAC OS X:

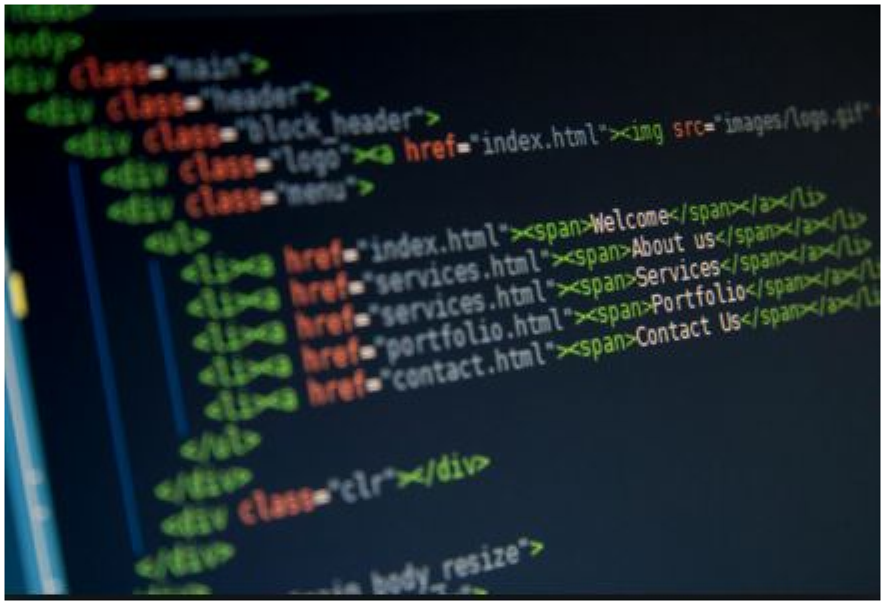
- Locate your Arduino program.
    - Most likely it is located in your Applications folder
- Right click and "Show Package Contents"
- Go to Java->hardware->arduino->avr->libraries->Wire
- You will be modifying wire.h and the twi.h
    - wire.h is located in the Wire folder
    - twi.h is located, one folder in Wire, in the folder utility
- Make a backup of those files
    - Right Click -> Duplicate
    - This way, in case anything goes wrong, you can change the name back to the original and replace the modified file
- Open up the wire.h file in a text editor
    - Change "BUFFER_LENGTH 32" to "BUFFER_LENGTH 64"
- Open up the twi.h file in a text editor
    - Change "TWI_BUFFER_LENGTH 32" to "TWI_BUFFER_LENGTH 64"
- You have changed the internal buffers to 64 byte max

Windows:

- Go to <install directory>\arduino-0022\libraries\Wire
- Locate the wire.h inside the folder wire
- Locate the twi.h inside the folder utility inside the Wire folder
- Make a backup of those files by copying and changing the name
    - This way, in case anything goes wrong, you can change the name back to the original and replace the modified file
- Open up the wire.h file in a text editor
    - Change "BUFFER_LENGTH 32" to "BUFFER_LENGTH 64"
- Open up the twi.h file in a text editor
    - Change "TWI_BUFFER_LENGTH 32" to "TWI_BUFFER_LENGTH 64"
- You have changed the internal buffers to 64 byte max

The Arduino's Libraries have just been modified to change the internal buffer to 64 bytes instead of the default 32 bytes. If you need to change it back for any reason, then just change the "64" values back to "32."

9. Arduino Coding



I will be using the Arduino's IDE to code, compile, and upload the file into the Arduino. I will not go in depth on the full features of the Arduino because there is a lot to learn, but reference III is the link to the website directly. The Arduino uses C and C++ and is relatively simple to use compared to many other microcontrollers.

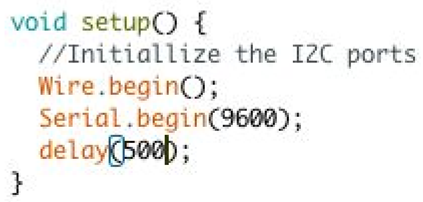
To start, we must define variables that we are going to use in our code. This will help make it easier to read and change the code.

```
#define D6T_ID 0x0A //Id address for the D6T
#define D6T_CMD 0x4C //Command to get information

//The D6T will return 35 bytes of data to be processed
//These vars will store them
int ReadBuffer[35]; //D6T Buffer
float ptat; // reference temperature (inside sensor)
float tdata[16]; // temporary temperature data for 16 pixels (4x4)
//float tpec; // packet error check ( may implement in future)
```

The above code shows the variables used and what they mean in the comments. One example to see why it is useful is by looking at the "D6T_ID," it is assigned the value of the I2C ID of "0x0A". Instead of having to remember different addresses for multiple devices, a simple name will make it a lot easier to use.

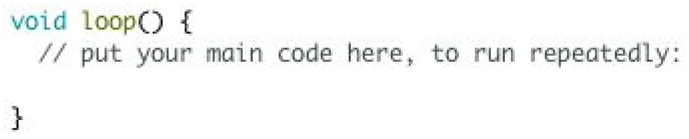
```
#include <Wire.h>
```

The library that the IDE uses for simple I2C communication is <wire.h>. The next step is to set up the initial setup for the I2C communication.



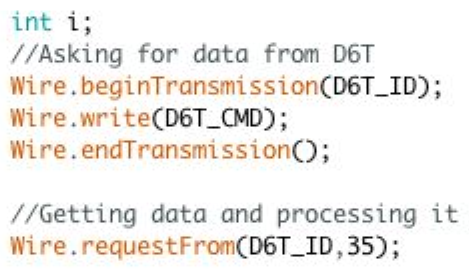
```
void setup() {
  //Initiallize the I2C ports
  Wire.begin();
  Serial.begin(9600);
  delay(500);
}
```

The “void setup()” is to initialized everything from the beginning. The first line in the setup says that a I2C connection is going to be used. The Serial.begin() starts a serial connection and the delay will give everything some time initially to start.



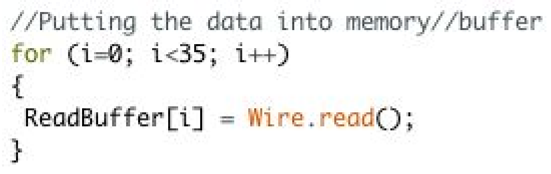
```
void loop() {
  // put your main code here, to run repeatedly:

}
```

The next part of the code is the loop of the code. Everything in this section will be repeated.



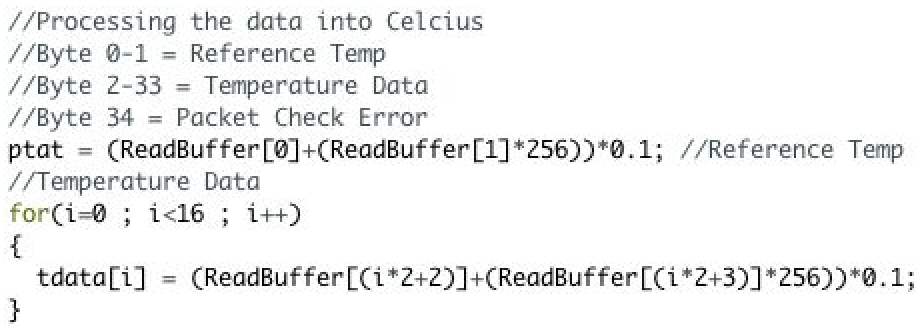
```
int i;
//Asking for data from D6T
Wire.beginTransmission(D6T_ID);
Wire.write(D6T_CMD);
Wire.endTransmission();

//Getting data and processing it
Wire.requestFrom(D6T_ID,35);
```

In section 6, I explained how the D6T communicates. The Arduino needs to begin a communication line with thermal sensor, write a command, then end the transmission to wait for the information to come into the internal buffer. Then, it must get the information from the internal buffer. The above code achieves that. The only thing that will probably a little confusing is the Wire.requestfrom() function. The ID and number of bytes to receive are the two inputs for that function.



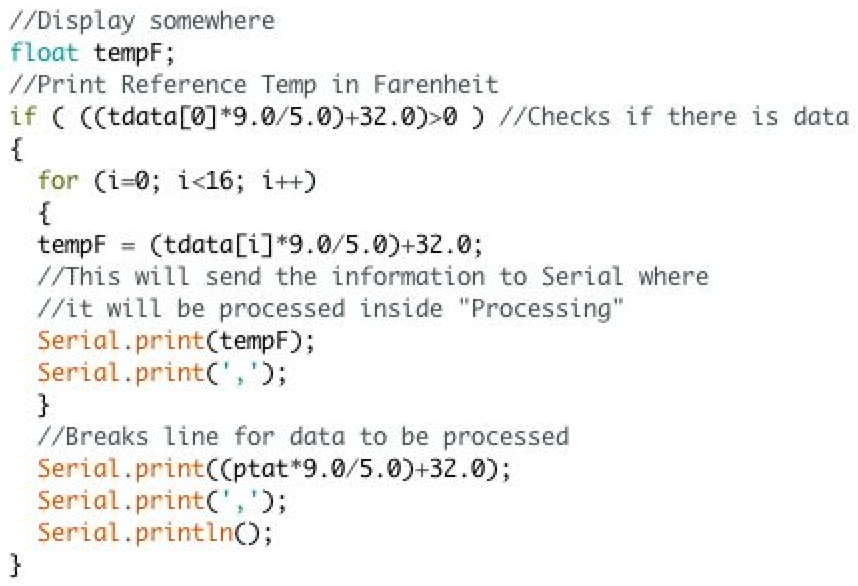
```
//Putting the data into memory//buffer
for (i=0; i<35; i++)
{
 ReadBuffer[i] = Wire.read();
}
```

The next part reads the data in the buffer and assigns it into the ReadBuffer[] variable to be used to process.



```
//Processing the data into Celcius
//Byte 0-1 = Reference Temp
//Byte 2-33 = Temperature Data
//Byte 34 = Packet Check Error
ptat = (ReadBuffer[0]+(ReadBuffer[1]*256))*0.1; //Reference Temp
//Temperature Data
for(i=0 ; i<16 ; i++)
{
  tdata[i] = (ReadBuffer[(i*2+2)]+(ReadBuffer[(i*2+3)]*256))*0.1;
}
```

Now that the information is ready, it must be processed into information that can be understood easily. This part of the code puts the low and high bytes of the temperature, processes it into celsius, then puts them into a temperature array. The “tdata” stores the pixels in order from 0-15 in the array.



```
//Display somewhere
float tempF;
//Print Reference Temp in Farenheit
if ( ((tdata[0]*9.0/5.0)+32.0)>0 ) //Checks if there is data
{
  for (i=0; i<16; i++)
  {
  tempF = (tdata[i]*9.0/5.0)+32.0;
  //This will send the information to Serial where
  //it will be processed inside "Processing"
  Serial.print(tempF);
  Serial.print(',');
  }
  //Breaks line for data to be processed
  Serial.print((ptat*9.0/5.0)+32.0);
  Serial.print(',');
  Serial.println();
}
```

Lastly, all the processed information will be send over the serial port. Later on, we will use “Processing” to use this information and make a nice graphical display. For now, we can display all the information on a serial monitor.

The “if” statement in this part will check if the data in the “tdata” array is greater than 0. The reason for this is because the D6T will send a value of -14 if there is no data available yet. This is a simple check to see if there is usable data, if there is then continue.

The next part converts the celsius into fahrenheit of each “tdata” and outputs it in the format of “x,x,x,x,x,x” where each of the x represents a temperature reading in order from 1-16 pixels. Then, there is a line break: which will let Processing know that the information for this segment is complete.

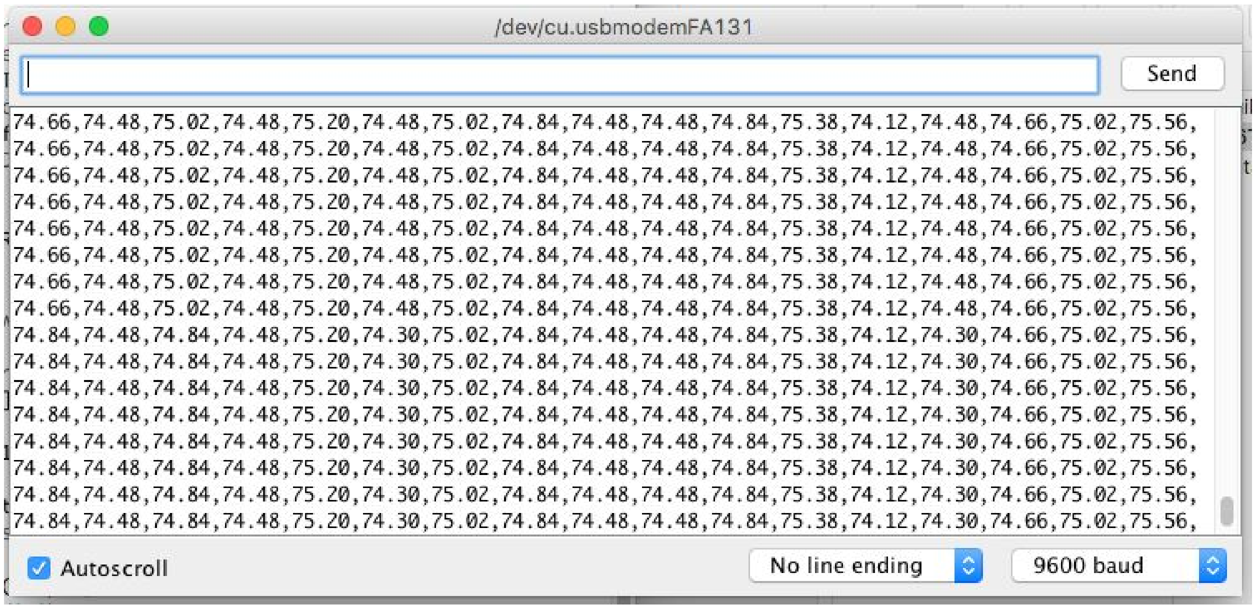
```
//I2C Library
#include <Wire.h>

#define D6T_ID 0x0A //Id address for the D6T
#define D6T_CMD 0x4C //Command to get information

//The D6T will return 35 bytes of data to be processed
//These vars will store them
int ReadBuffer[35]; //D6T Buffer
float ptat; // reference temperature (inside sensor)
float tdata[16]; // temporary temperature data for 16 pixels (4x4)
//float tpec; // packet error check ( may implement in future)


void setup() {
  //Initiallize the I2C ports
  Wire.begin();
  Serial.begin(9600);
  delay(500);
}


void loop() {

  int i;
  //Asking for data from D6T
  Wire.beginTransmission(D6T_ID);
  Wire.write(D6T_CMD);
  Wire.endTransmission();

  //Getting data and processing it
  Wire.requestFrom(D6T_ID,35);

  //Putting the data into memory//buffer
  for (i=0; i<35; i++)
  {
   ReadBuffer[i] = Wire.read();
  }

  //Processing the data into Celcius
  //Byte 0-1 = Reference Temp
  //Byte 2-33 = Temperature Data
  //Byte 34 = Packet Check Error
  ptat = (ReadBuffer[0]+(ReadBuffer[1]*256))*0.1; //Reference Temp
  //Temperature Data
  for(i=0 ; i<16 ; i++)
  {
    tdata[i] = (ReadBuffer[(i*2+2)]+(ReadBuffer[(i*2+3)]*256))*0.1;
  }
```

```
//Display somewhere
float tempF;
//Print Reference Temp in Farenheit
if ( ((tdata[0]*9.0/5.0)+32.0)>0 ) //Checks if there is data
{
  for (i=0; i<16; i++)
  {
  tempF = (tdata[i]*9.0/5.0)+32.0;
  //This will send the information to Serial where
  //it will be processed inside "Processing"
  Serial.print(tempF);
  Serial.print(',');
  }
  //Breaks line for data to be processed
  Serial.print((ptat*9.0/5.0)+32.0);
  Serial.print(',');
  Serial.println();
}
```

The final code should look like above. To compile this and upload this to the Arduino: go to Tools->Board and choose the correct Arduino board, then choose the port that the Arduino is connected to your computer. Then, press the “play,” or upload, button.

10. Testing that the code works

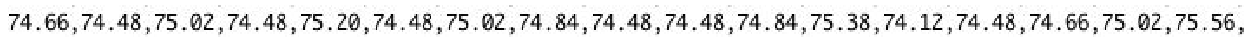
So, all that work was coded but how can you check if everything is working as intended? The Arduino IDE has a nice feature that can allow you see what information is being outputted through the serial connection from the Arduino. To use this: go to Tools->Serial Monitor while the arduino is connected to the port with the code uploaded.



```
74.66,74.48,75.02,74.48,75.20,74.48,75.02,74.84,74.48,74.48,74.84,75.38,74.12,74.48,74.66,75.02,75.56,
74.66,74.48,75.02,74.48,75.20,74.48,75.02,74.84,74.48,74.48,74.84,75.38,74.12,74.48,74.66,75.02,75.56,
74.66,74.48,75.02,74.48,75.20,74.48,75.02,74.84,74.48,74.48,74.84,75.38,74.12,74.48,74.66,75.02,75.56,
74.66,74.48,75.02,74.48,75.20,74.48,75.02,74.84,74.48,74.48,74.84,75.38,74.12,74.48,74.66,75.02,75.56,
74.66,74.48,75.02,74.48,75.20,74.48,75.02,74.84,74.48,74.48,74.84,75.38,74.12,74.48,74.66,75.02,75.56,
74.66,74.48,75.02,74.48,75.20,74.48,75.02,74.84,74.48,74.48,74.84,75.38,74.12,74.48,74.66,75.02,75.56,
74.66,74.48,75.02,74.48,75.20,74.48,75.02,74.84,74.48,74.48,74.84,75.38,74.12,74.48,74.66,75.02,75.56,
74.66,74.48,75.02,74.48,75.20,74.48,75.02,74.84,74.48,74.48,74.84,75.38,74.12,74.48,74.66,75.02,75.56,
74.84,74.48,74.84,74.48,75.20,74.30,75.02,74.84,74.48,74.48,74.84,75.38,74.12,74.30,74.66,75.02,75.56,
74.84,74.48,74.84,74.48,75.20,74.30,75.02,74.84,74.48,74.48,74.84,75.38,74.12,74.30,74.66,75.02,75.56,
74.84,74.48,74.84,74.48,75.20,74.30,75.02,74.84,74.48,74.48,74.84,75.38,74.12,74.30,74.66,75.02,75.56,
74.84,74.48,74.84,74.48,75.20,74.30,75.02,74.84,74.48,74.48,74.84,75.38,74.12,74.30,74.66,75.02,75.56,
74.84,74.48,74.84,74.48,75.20,74.30,75.02,74.84,74.48,74.48,74.84,75.38,74.12,74.30,74.66,75.02,75.56,
74.84,74.48,74.84,74.48,75.20,74.30,75.02,74.84,74.48,74.48,74.84,75.38,74.12,74.30,74.66,75.02,75.56,
74.84,74.48,74.84,74.48,75.20,74.30,75.02,74.84,74.48,74.48,74.84,75.38,74.12,74.30,74.66,75.02,75.56,
74.84,74.48,74.84,74.48,75.20,74.30,75.02,74.84,74.48,74.48,74.84,75.38,74.12,74.30,74.66,75.02,75.56,
```

No line ending
9600 baud

The information above should be displayed on the serial monitor. The screen will fill a lot faster than we can read individually but it can be broken down. It looks like a lot of data but if you take it line by line you can see it more clearly.



```
74.66,74.48,75.02,74.48,75.20,74.48,75.02,74.84,74.48,74.48,74.84,75.38,74.12,74.48,74.66,75.02,75.56,
```

Looking at the line above, you can see the data a lot easier. The first part of the line is 74.66 F and has a comma for every temperature reading. Each temperature reading is in respect to each pixel of the D6T.

11. Using "Processing" to output a graphical display

Once again, Processing is a very in depth program that can do wonders when it comes to outputting your information into a very beautiful graphical display, but I can not cover everything because that is beyond the scope of this article. However, if you are still interested in learning more about this program, you can look at reference VI for more tutorials and examples.

When you open the program, you will notice it is very similar to the Arduino IDE; that is because it the Arduino IDE was based on processing.

Okay, now it is time to see how it will work. First, we will take the information from the serial that is inputted from the Arduino and take the steps below:

- Take the string and split it into our own temperature data
- Build a boxed graphical display that will output the temperature data
    - Output will be a color relative to the temperature

Processing has three main functions:

- setup()
    - Used to initially set up processing to do what we want
- draw()
    - Equivalent to the loop() function in the Arduino IDE
    - Loops and draws the shapes we want every "Draw()" tick
- serialEvent()
    - This function gets called in complementary with a different bufferUntil() function
    - The bufferUnit() function will take data from the serial until a certain character or word; in our case, we are using a line break to differentiate between different data sets of temperature readings.

To start, we will import the serial library. This will activate the using of the serial line to input data for a graphical display. Also, just like the other Arduino program, we can add the variables too.

```
import processing.serial.*;

//Serial Info
String portName = "/dev/cu.usbmodemFA131"; // Port name for Serial connection (COM or DEV)
Serial port; // Port object main
int BaudRate = 9600;
String inString; //String to process

//Temperature Variables
float[] tdata = new float[17]; //temperature data goes here + reference

//Misc Vars
String buff; //Buffer for everything
PFont font;
float colorGreen; //Changes green in relation to temperature
int lbTemp = 60; //lowest relative temp
int hbTemp = 90; //highest relative temp
```

The variables include the serial initial names, speed, and the string buffer. It also includes the array for the temperature data, relative temperature data points, font, and colors for the temperature output. Everything will be explained in more details later on.

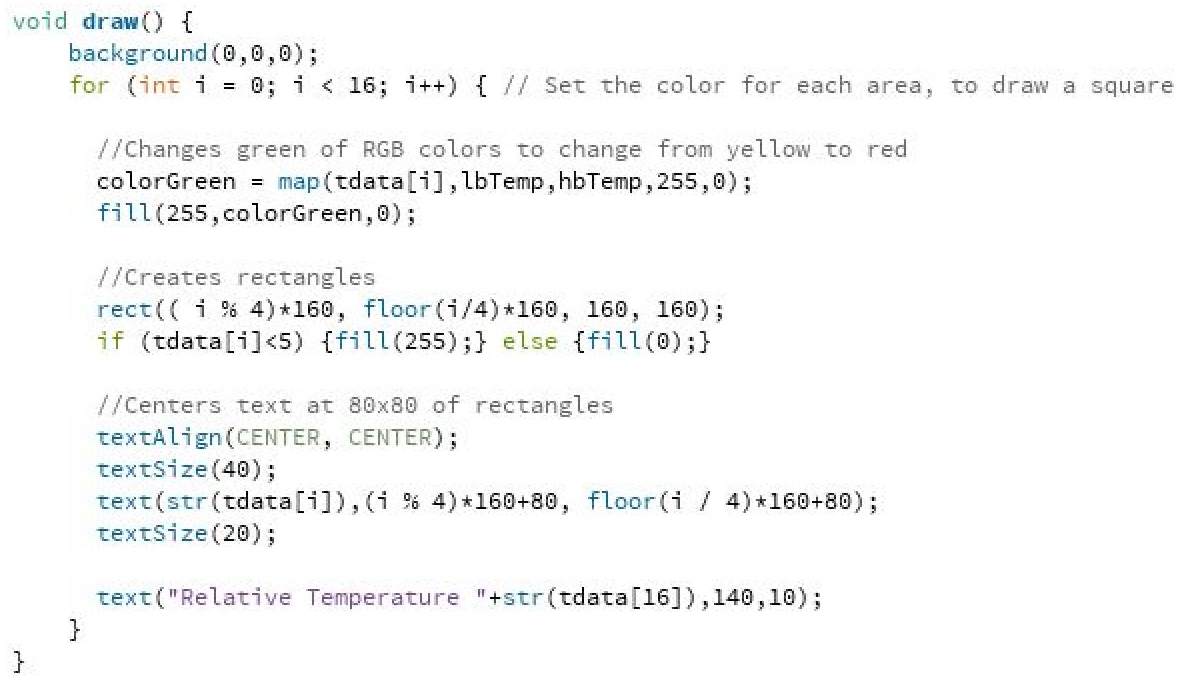
```
//Inital Setup
void setup() {
  size(640,640); //<----------------Sets Size of Window

  //Initiallizing the Serial port and look for line breaks for processing
  port = new Serial(this, portName, BaudRate);
  port.bufferUntil('\n');

  //Setting font for display

  font = loadFont("FZLTZHB--B51-0-48.vlw");
  textFont(font,40);

}
```

To initially set everything up, I will start with the first portion of the code. The size() function sets up the window size to be displayed.

The second portion sets up the serial port and where to stop collecting data and processes it. In the serial() function, the “portName” is the usb port that the Arduino is plugged in. The best way to find the name is by plugging the Arduino into the plug and go to the Arduino IDE Tools->Port-> and there should be a few of the ports showing. If you have a Windows then it should likely be “COMx” and if you have a OS X then it should be “/dev/cu/xxx.” The next part of this portion is the port.bufferUntil() function; the Processing will look for this character and call the serialEvent. In this case, ‘\n’ is used. This is a the special character for line break.

To sum it up again, Processing will take all the information in the serial up until '\n,' or line break, then call the serialEvent().

The last portion of the section is the font initialization. The graphical display will have the relative temperature and the pixel temperatures displayed along with the colors.



```
void draw() {
    background(0,0,0);
    for (int i = 0; i < 16; i++) { // Set the color for each area, to draw a square

      //Changes green of RGB colors to change from yellow to red
      colorGreen = map(tdata[i],lbTemp,hbTemp,255,0);
      fill(255,colorGreen,0);

      //Creates rectangles
      rect(( i % 4)*160, floor(i/4)*160, 160, 160);
      if (tdata[i]<5) {fill(255);} else {fill(0);}

      //Centers text at 80x80 of rectangles
      textAlign(CENTER, CENTER);
      textSize(40);
      text(str(tdata[i]),(i % 4)*160+80, floor(i / 4)*160+80);
      textSize(20);

      text("Relative Temperature "+str(tdata[16]),140,10);
    }
}
```

This is the draw function, it is equivalent to the loop() function in the arduino IDE. As you can see, everything is in the for() function that has 16 iteration: one for each pixel.

The beginning of the for() loop has to do with the color of that iteration of the temperature data. The color is picked using the RGB scale and by keeping the red at 255, blue at 0 , and changing the green between 0-255, the color can be change between yellow and red. To get a better visual display, reference VII is a link to a color picker. If you set those values referenced previously, you can see it changes between those colors. By using the map() function, we can map between low and high temperature data to the values between 0-255.

That leads me to my next clue, the "lbTemp" and "hbTemp" stands for "low bound temperature" and "high bound temperature"; to make this more clear, the temperature we are taking might not need to change between 0-200 fahrenheit. By changing to a lower bound, we can have a display that can change more relative to the temperature we are using. In this example, I am using 60 degrees as the low bound and 90 degrees as the high bound. This means, if it is higher than 90 degrees then it will be red and lower than 60 it will be yellow.

The next part of the for() loop deals with creating rectangles that will be filled with the color that relates to which pixel we are working on. It uses the format:

rect(a,b,c,d):

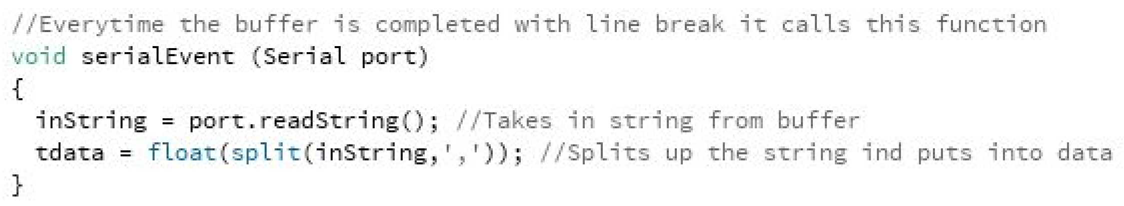
- a = x position
- b = y position
- c = width
- d = height of the rectangle

It will be placed within the 640x640 window. You can prove my calculated by breaking it down.

The last parts deals with the text size and placement. It uses the format:

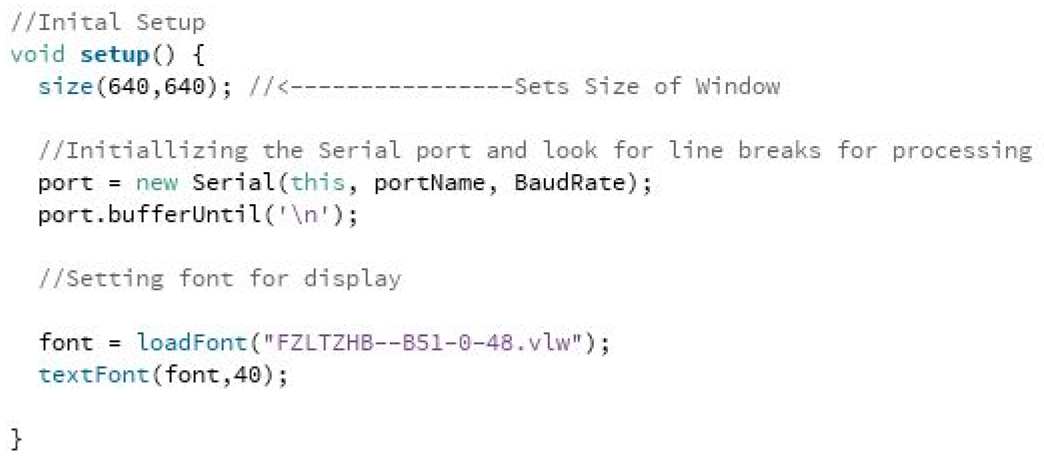
text(e,f,g):

- e = string to be displayed
- f = x position
- g = y position.

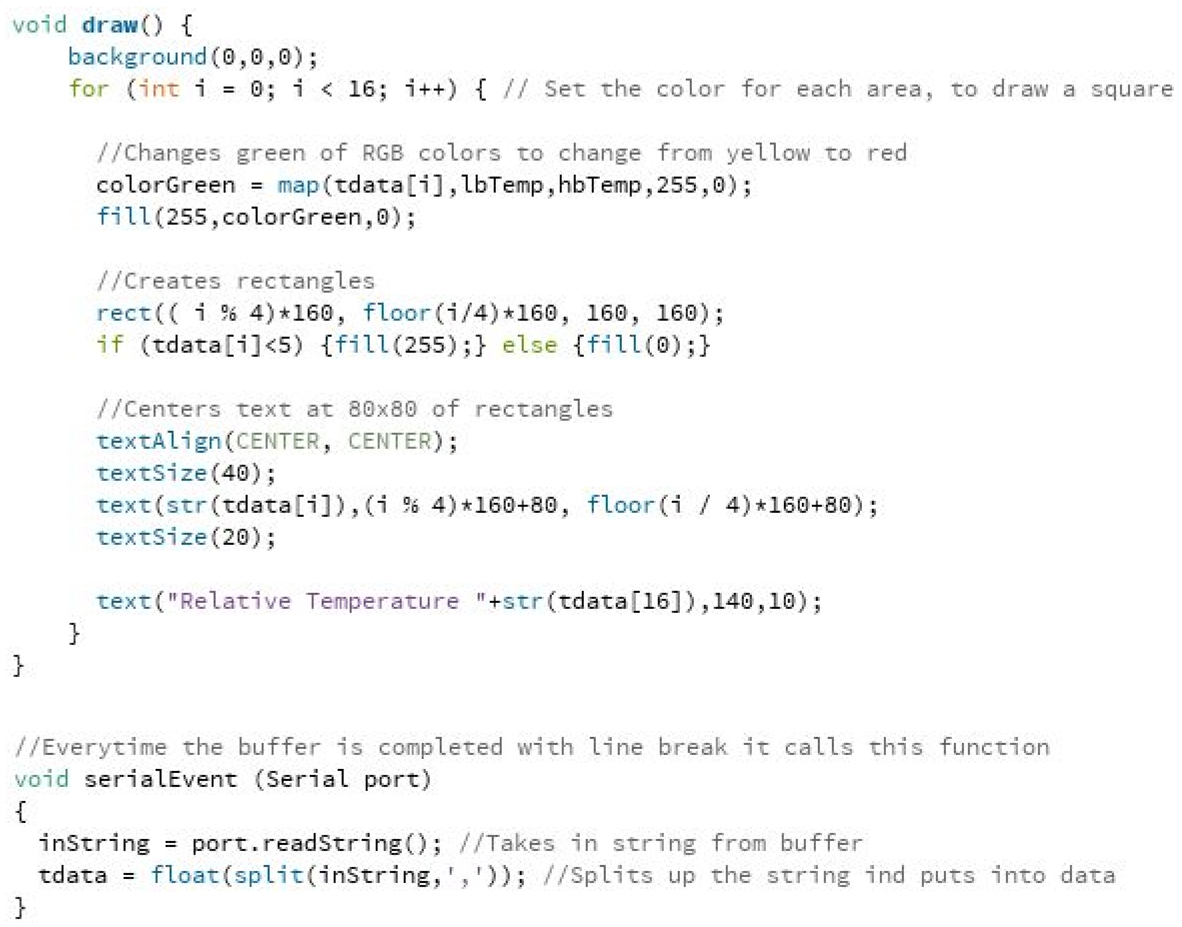


```
//Everytime the buffer is completed with line break it calls this function
void serialEvent (Serial port)
{
  inString = port.readString(); //Takes in string from buffer
  tdata = float(split(inString,',')); //Splits up the string ind puts into data
}
```

Finally, this is the serialEvent() function. It takes in the string that was captured by the bufferUntil() and splits it into the tdata array to be processed. Since the draw() function constant repeats itself, as the serialEvent() updates, then the draw() will also.



```
//Inital Setup
void setup() {
  size(640,640); //<----------------Sets Size of Window

  //Initiallizing the Serial port and look for line breaks for processing
  port = new Serial(this, portName, BaudRate);
  port.bufferUntil('\n');

  //Setting font for display

  font = loadFont("FZLTZHB--B51-0-48.vlw");
  textFont(font,40);

}
```

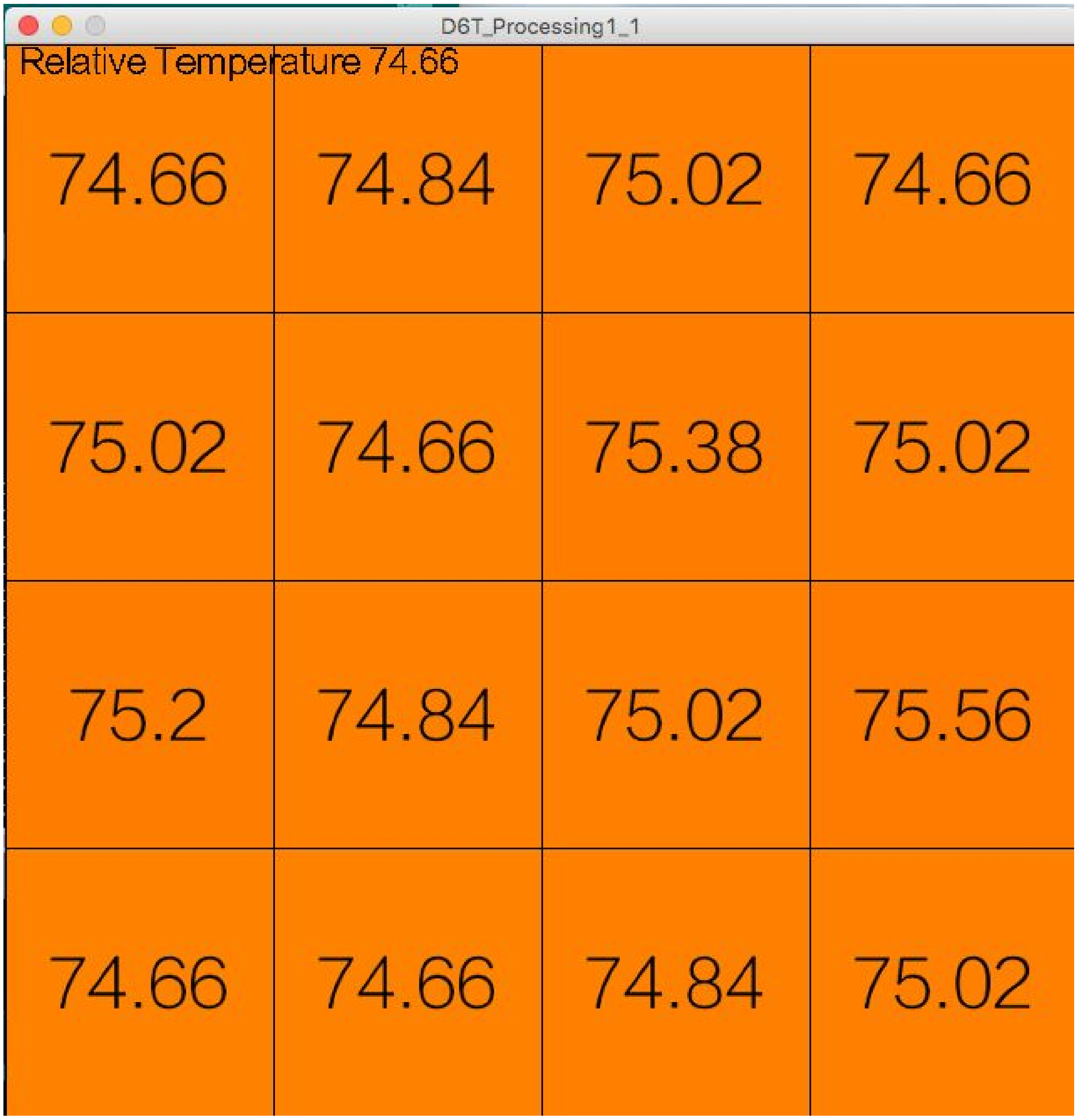


```
void draw() {
    background(0,0,0);
    for (int i = 0; i < 16; i++) { // Set the color for each area, to draw a square

      //Changes green of RGB colors to change from yellow to red
      colorGreen = map(tdata[i],lbTemp,hbTemp,255,0);
      fill(255,colorGreen,0);

      //Creates rectangles
      rect(( i % 4)*160, floor(i/4)*160, 160, 160);
      if (tdata[i]<5) {fill(255);} else {fill(0);}

      //Centers text at 80x80 of rectangles
      textAlign(CENTER, CENTER);
      textSize(40);
      text(str(tdata[i]),(i % 4)*160+80, floor(i / 4)*160+80);
      textSize(20);

      text("Relative Temperature "+str(tdata[16]),140,10);
    }
}


//Everytime the buffer is completed with line break it calls this function
void serialEvent (Serial port)
{
  inString = port.readString(); //Takes in string from buffer
  tdata = float(split(inString,',')); //Splits up the string ind puts into data
}
```

This is how the final code should look like.

12. Results

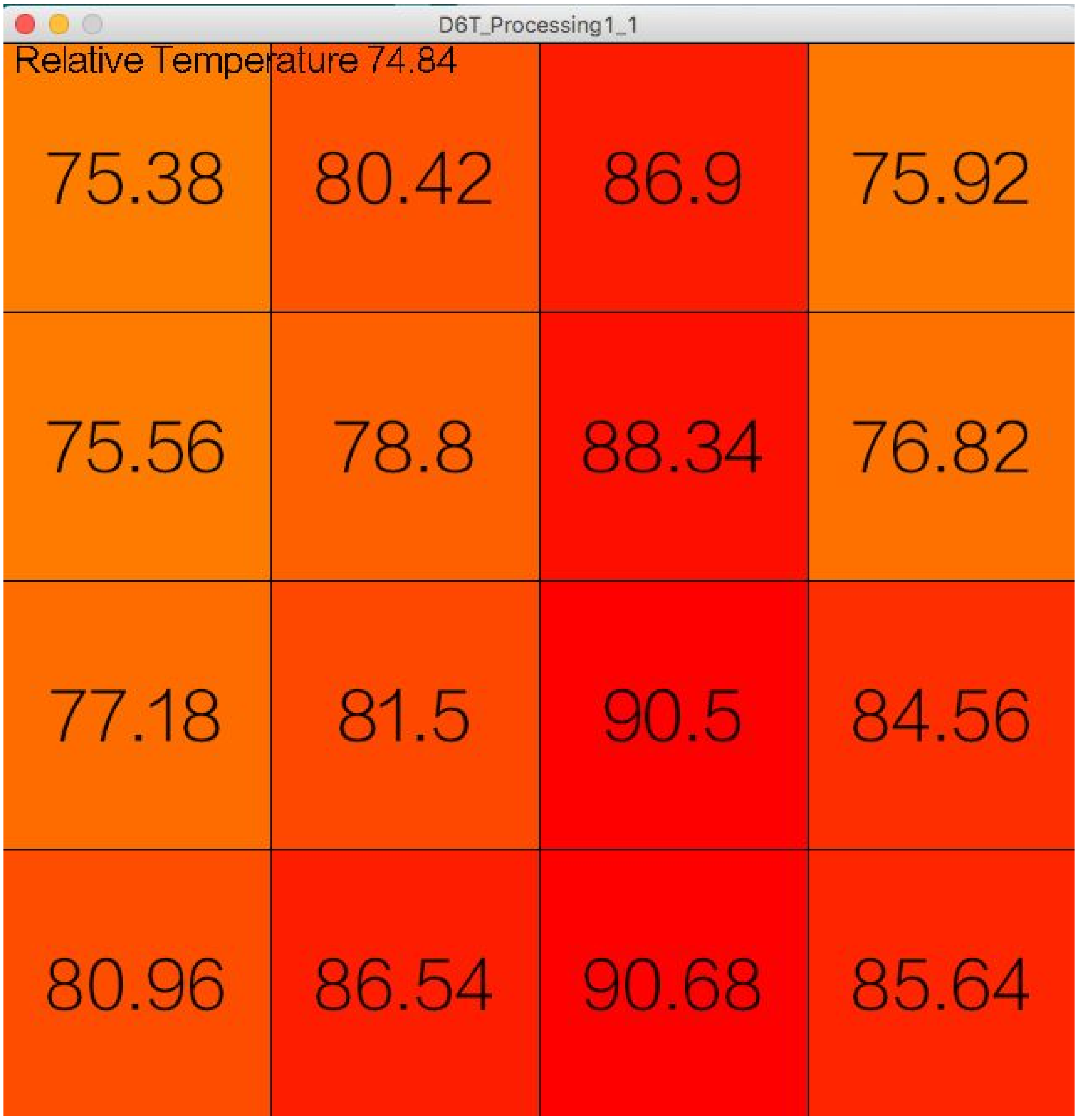
Now that the code is finished, you can graphically display everything. If you plug in your Arduino and press the “play” button on the Processing IDE then you should see the display. Below, you can see the temperature changing as I put my hand in front of the sensor.


D6T_Processing1_1

Relative Temperature 74.66

74.66	74.84	75.02	74.66
75.02	74.66	75.38	75.02
75.2	74.84	75.02	75.56
74.66	74.66	74.84	75.02

Above shows the graphical display initially


D6T_Processing1_1

Relative Temperature 74.84

75.38	80.42	86.9	75.92
75.56	78.8	88.34	76.82
77.18	81.5	90.5	84.56
80.96	86.54	90.68	85.64

Above shows the temperature change as I put my hand in front

13. Conclusions

I hope you enjoyed this tutorial on how to integrate an Arduino with a D6T temperature sensor and then displaying it graphically using Processing. I've learn a lot about how everything works the first time I did this and thoroughly enjoyed this experience.

14. References

I. https://www.omron.com/ecb/products/sensor/11/d6t.html
II. https://learn.sparkfun.com/tutorials/i2c
III. https://www.arduino.cc/
IV. https://www.arduino.cc/en/Tutorial/HomePage
V. https://learn.sparkfun.com/tutorials/i2c
VI. https://processing.org/tutorials/
VII. http://www.colorpicker.com/