QUANTIFICATION OF BLOOD FLOW VELOCITY USING COLOR SENSING

A Thesis

presented to

the Faculty of California Polytechnic State University

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Electrical Engineering

by

Aditya Deepak Sanghani

October 2015

COMMITTEE MEMBERSHIP

TITLE:	Quantification of Blood Flow Velocity Using Color Sensing
AUTHOR:	Aditya Deepak Sanghani
DATE SUBMITTED:	October 2015
COMMITTEE CHAIR:	Tina Smilkstein, Ph.D. Associate Professor of Electrical Engineering
COMMITTEE MEMBER:	Dennis Derickson, Ph.D. Department Chair and Professor of Electrical Engineering
COMMITTEE MEMBER:	James Eason, Ph.D. Instructor of Biomedical Engineering

ABSTRACT

Quantification of Blood Flow Velocity Using Color Sensing

Aditya Deepak Sanghani

Blood flow velocity is an important parameter that can give information on several pathologies including atherosclerosis, glaucoma, Raynaud's phenomenon, and ischemic stroke [2,5,6,10]. Present techniques of measuring blood flow velocity involve expensive procedures such as Doppler echocardiography, Doppler ultrasound, and magnetic resonance imaging [11,12]. They cost from $8500-$20000. It is desired to find a low-cost yet equally effective solution for measuring blood flow velocity. This thesis has a goal of creating a proof of concept device for measuring blood flow velocity.

Finger blood flow velocity is investigated in this project. The close proximity to the skin of the finger's arteries makes it a practical selection. A Red Green Blue (RGB) color sensor is integrated with an Arduino Uno microcontroller to analyze color on skin. The initial analysis involved utilization of red RGB values to measure heart rate; this was performed to validate the sensor. This test achieved similar results to an experimental control as the measurements had error ranging from 0% to 6.67%.

The main analysis was to measure blood flow velocity using 2 RGB color sensors. The range of velocity found was 5.20cm/s to 12.22cm/s with an average of 7.44cm/s. This compared well with the ranges found in published data that varied from 4cm/s to 19cm/s. However, there is an error associated with the device that affects the accuracy of the results. The apparatus has the limitation of collecting data between sensors every 102-107ms, so there is a maximum error of 107ms. The average finger blood flow velocity of 7.44cm/s may actually be between 6.17cm/s and 9.39cm/s due to the sampling error. In addition, mean squared error analysis found that the most likely time difference between pulses among those found is 739ms, which corresponds to 5.21cm/s.

Although there is error in the system, the tests for heart rate along with the obtained range and average for finger blood velocity data provided a method for analyzing blood flow velocity. Finger blood velocity was examined in a much more economical manner than its traditional methods that cost between $8500-$20000. The cost for this entire thesis was $99.66, which is a maximum of 1.17% of the cost.

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Smilkstein for all of the help and support throughout the past year. Her passion for biomedical devices led us to work together and it was inspiring to be guided by a professor with such a strongly innovative mentality and diverse background.

I would like to also thank my other committee members Dr. Eason and Dr. Derickson. I would like to especially thank Dr. Eason for the initial help with the physiology.

I would like to thank Dr. Carlton for his help with the statistical error analysis.

I would like to thank my friend Jonathan Fish for his help connecting the multiplexer to the breakout board. I would also like to thank my friend Michael Harriman for some assistance with experiments and my friend Bijan Shirazi-Wu for giving me direction in searching for a multiplexer.

I would like to dedicate this thesis to my family who has been there for me throughout my life. I would not be where I am today without their incredible support and teachings.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1: Introduction

Blood flow characteristics carry vital information about any person's health. Blood flow velocity is one parameter that is not given enough attention but is important to monitor. Blood flow velocity is influenced by blood vessel size, shape, and wall texture, blood viscosity, cardiac cycle phase, and blood flow rate [1]. The velocity is affected during stenosis, regurgitation, and in other cases. Stenosis is the blockage of blood vessels. One example is stenosis is atherosclerosis, which is a pathology that involves the buildup in plaque in arteries [2]. Regurgitation is when blood flows in the opposite direction through heart values from what is normal [3]. In general, when blood flow is turbulent, this can be suggestive of disease.

Generally, aortic, femoral, or carotid blood flow velocity is measured and analyzed in order to find pathologies. However, blood flow velocity in distal extremities can provide useful information about a person's cardiovascular health [4]. Analysis of smaller arteries can show arteriosclerosis and other problems and this can indicate pathologies at a systemic level. Lower finger blood flow velocities from what is normal can be indicative of glaucoma [5]. Additionally, low blood flow velocity can be due to Raynaud's phenomenon, especially at lower temperatures such as 19 °C [6]. Raynaud's phenomenon is a condition during which blood flow is decreased for a certain period of time due to vasospasm. For all of the aforementioned reasons, finger blood flow velocity is examined in this thesis. Specifically, an inexpensive method for its measurement is investigated.

One published study found that the range for finger blood flow velocity (arterial) is about 4cm/s to 10cm/s (see Figure 1.1) [7]:

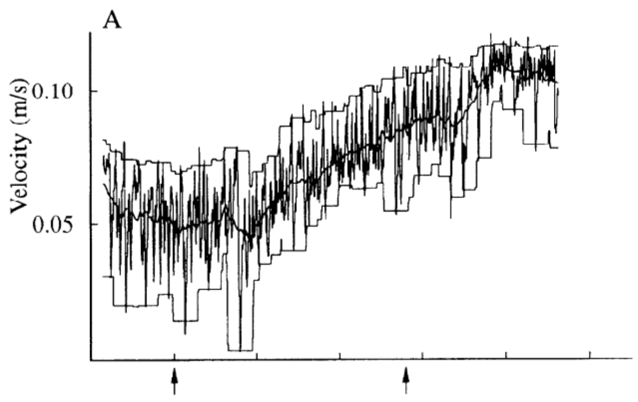


Figure 1.1: Published data for range of finger blood flow velocity

Another study found that the finger blood flow velocity for arteries ranges from 4.9cm/s to 19cm/s [4]. It is desired to create an apparatus to achieve the same range of data as these studies found.

A separate study found that ischemic stroke affects values for blood flow velocity in the carotid artery [10]. The study showed that patients without ischemic stroke experienced an average blood flow velocity of 20.8cm/s whereas those with a stable form of ischemic stroke experienced an average of 16.8cm/s. This is a decrease of 19.2%, and this kind of decrease can be expected in the finger as well. The new apparatus has to be able to sense a decrease of this magnitude.

The artery analyzed in this paper is the digital artery, as shown in Figure 1.2:

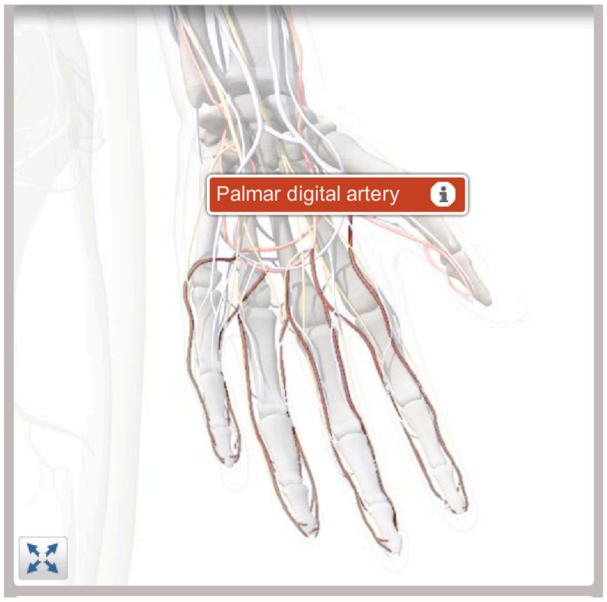


Figure 1.2: Palmar digital artery illustration [8]

The digital artery is straight and runs through the fingers as shown in Figure 1.2, which makes it possible to measure blood flow velocity from two points. Figure 1.3 shows markings of the location of one of the digital arteries from the outside [9]: These arteries will be directly monitored in finding blood flow velocity, and this is explained further in this thesis.

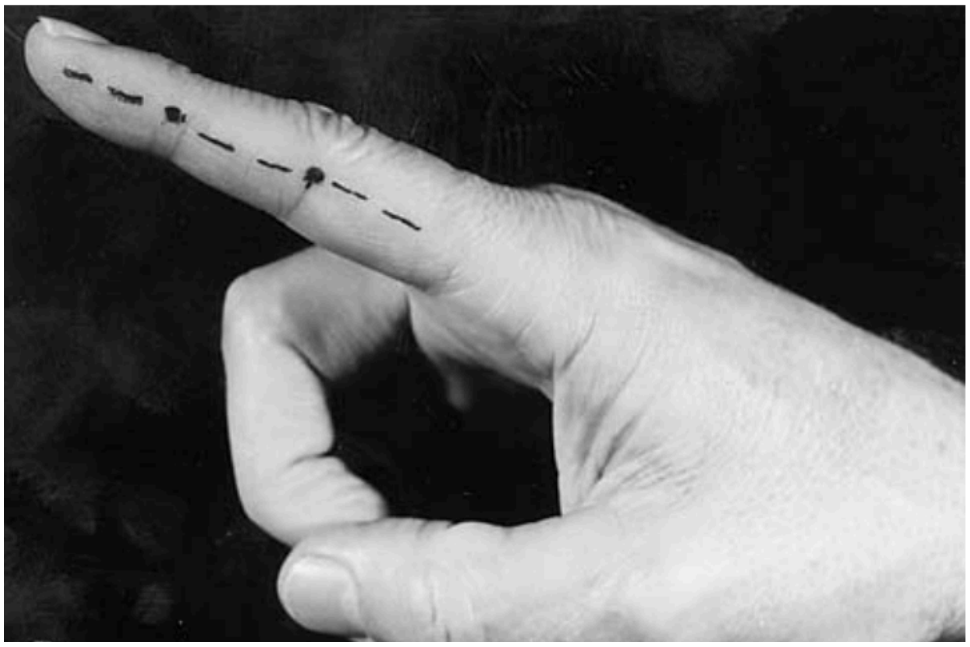


Figure 1.3: Marking of digital artery location to show straightness

Figure 1.3 shows that the digital artery has a straight shape. It is also known that it is close to the surface of the skin. Color sensors are used over the skin's surface in the experiments in this thesis, and this is explained in Chapter 2.

Current techniques to measure blood flow velocity are very expensive. Examples include Doppler echocardiography, Doppler ultrasound, and magnetic resonance imaging (MRI) [11,12]. Doppler ultrasound is a technique that analyzes how sound waves bounce off of blood cells and how they are reflected [13]. A transducer is used to transmit and receive sound waves and its analyses give information on the blood flow. Figure 1.4 shows data from Doppler ultrasound [14]:

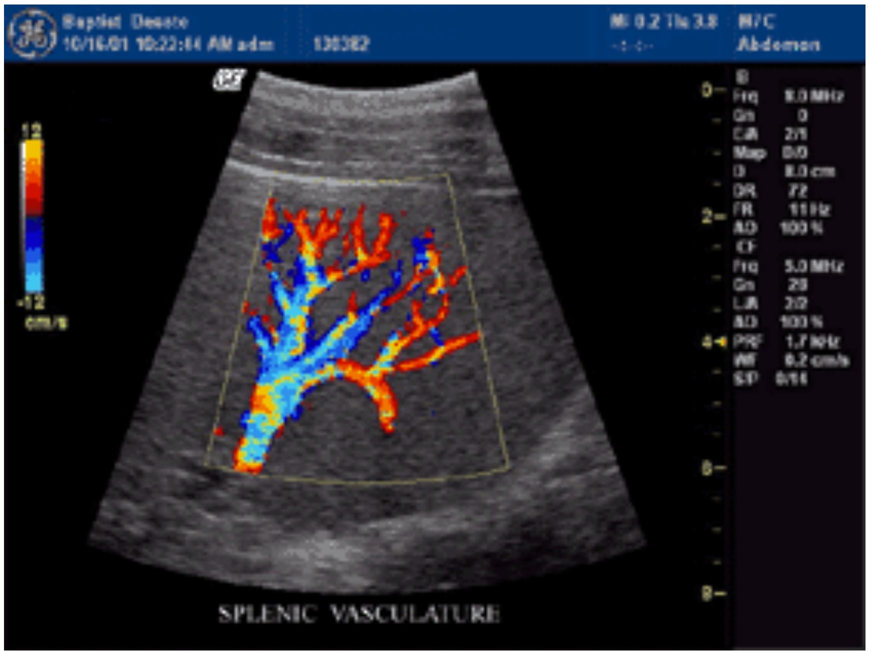


Figure 1.4: Measurement of blood flow using Doppler ultrasound

Doppler echocardiography is when ultrasound techniques are used to measure blood flow activity in the heart; it provides detailed images [15]. Though these seem to be promising techniques, it has been found that Doppler can produce errors in flow information. A study found that the error in mean velocity can be as high as 20-32% [16]. Sources show that the cost of these procedures range from \$8500-\$17000 [17]. MRI uses a magnetic field and radio waves in order to generate images [18]. Phase shift with the MRI readings of blood in vessels in used to find blood flow velocity. This technique produces error as well, and its cost is well over \$20000 [19].

It is desired to investigate a more economical technique to accurately measure blood flow velocity−color characterization using color sensors is used.

Chapter 2 discusses the theory behind the design and it includes the experimental setup. Chapter 3 illustrates the data collection and the corresponding analyses. Chapter 4 concludes and gives future considerations.

Chapter 2: Theory

2.1. Theory and Background

The hypothesis is that blood flow velocity can be found through color characterization of the skin. Specifically, two Red Green Blue (RGB) color sensors are placed a small distance apart on an index finger and the data are communicated through an Arduino Uno microcontroller. The RGB color sensor chosen is TCS34725 by Texas Advanced Optoelectronic Solutions (TAOS) that runs on I^2C protocol [20]. The color sensor comes with an infrared (IR) blocking filter. The breadth of different wavelengths and subtleties of color attainable through this sensor are not visible with the human eye. With the filtering coupled with analysis of the color readings, a characterization of periodic color changes over a small time interval (10-20 seconds) is attained. Specifically, red wavelengths are analyzed in order to track blood. These wavelengths illustrate pulse events at their graphical output, which is explained further in this thesis. The physical distance between the sensors is divided by the time between these pulse events in order to obtain the velocity of the blood. The assessed values for blood flow velocity are compared with published data to confirm the validity of the measurements.

2.2. Hardware

The TCS34725 color sensor is a slave device that needs to have its communications controlled by a master device. This relationship is very customary in the I^2C protocol. The code utilized on the Arduino for communicating with the sensors is derived from sample code produced by

Adafruit Industries for use in conjunction with an Arduino microcontroller. Thus, the Arduino is a master in this apparatus that controls the color sensor communications; it asks the sensor for data and these get subsequently sent to the Arduino. The TCS34725 color sensor uses a 3x4 photodiode array with red-filtered, green-filtered, blue filtered, and unfiltered (clear) photodiodes. They each have an infrared blocking filter and the output is a single value each for R, G, B, and C (clear) values. Analog to digital converters (ADCs) are included for each of these four parameters and they run in parallel in the data collection. The R, G, and B values that correspond to 0 to 255 are found by dividing each individual color value by the value for clear and then multiplying that new amount by 255. Red is the important color for blood characterization, and that is why red RGB values are analyzed in this application.

Adafruit has provided a breakout board along with some circuit elements such as pull-up resistors in order to facilitate the task of placing the TCS34725 color sensor onto a breadboard. Additionally, a 4150K LED is included to the left of the physical sensor in order to illuminate what is being analyzed. This is helpful to reading the colors correctly because it makes it easier for the sensor to gauge the colors and it reduces effects of noise from external light sources. Figure 2.1 is an image of what is included in the Adafruit package:

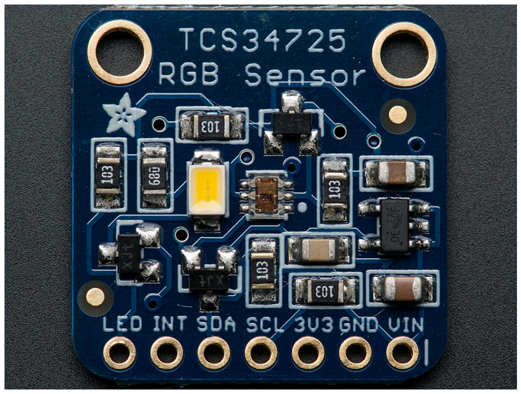


Figure 2.1: TCS34725 RGB color sensor with breakout board [21]

Since two sensors with the same fixed address are used, an I^2C multiplexer has to be employed. I^2C is a protocol for which a master selects a device address for a slave in order to communicate with it, but there must be only one slave with a given address to avoid conflict [22]. This multiplexer will have to quickly switch between the two sensors in the data collection in order to allow for a sufficient amount of data to be taken. This is because the multiplexer has to switch significantly faster than speed of blood in order for approximation of pulse shape. The difference between the pulse event peaks from the two sensors is what signifies the time taken for the blood to travel the given distance on the finger; 3.85cm is chosen for this study. This distance gives a maximum of about one second to for the blood to pass from one point to the other; this corresponds to the minimum amount of published blood flow velocity in the finger (4cm/s). Additionally, the analysis in Figure 2.3 shows that choosing this distance ensures that pulse events will not be missed. The time taken for blood to travel 3.85cm is found and the blood flow velocity is easily found by dividing the distance by the time. There is certain error associated with the device, and this is explained later in the thesis in sections 3.3 and 3.4.

2.3. Timing

The multiplexer selected is the Texas Instruments TCA9544A, which allows up to 4 sensors to be connected at once. In this project, two sensors are used and transmissions are alternated between them. There is a time delay of approximately 100ms between data points of the two sensors and a delay of approximately 200ms between data points of a single sensor. The exact time figures are explained in section 3.3. These time figures are due to the limitation of the TCS34725 color sensor that about 50ms are necessary to read data and that it is recommended to give a 50ms delay for fidelity. This additional time gives the sensor enough time to collect data. Shown in Figure 2.2 is a communication timing diagram when one sensor is connected.

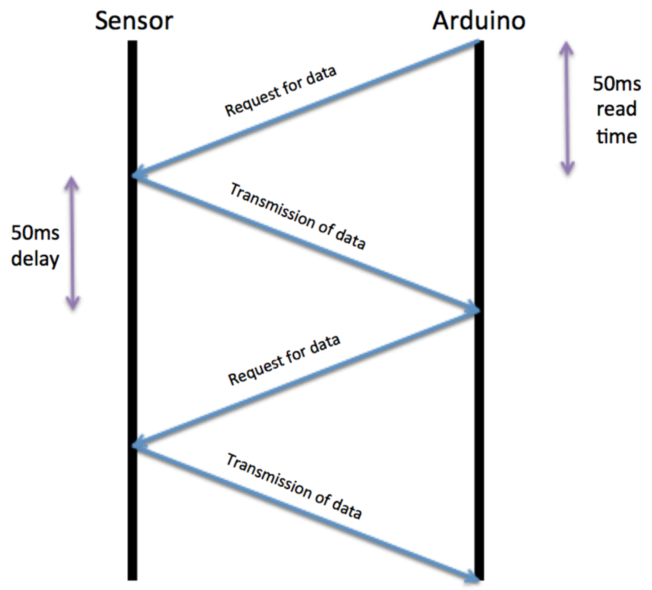


Figure 2.2: Communication timing diagram for TCS34725 and Arduino Uno

Given from published data that the finger blood flow velocity should not exceed 19cm/s, a 100ms delay will not cause the second sensor to miss the pulse event from the first sensor since the distance is 3.85cm and the minimum time to travel that distance is 203ms (see Figure 2.3):

$$time = \frac{distance}{velocity} = \frac{3.85cm}{19cm/s} = 203ms$$

Figure 2.3: Minimum time for pulse event to travel 3.85cm on finger; found using maximum published finger blood flow velocity

It is evident that pulse events cannot be missed entirely at the second sensor due to the 100ms delay. However, there will be some error in collecting data because of this 100ms delay and thus the 200ms delay between data points in a single sensor; the exact location of the peaks of the pulses may not be accurate. The graphical output is not smooth because straight lines are drawn to connect the discrete data points. A peak can occur between two data points and its exact location would not be caught. Figure 2.4 illustrates the connection of discrete data points with blue circles.

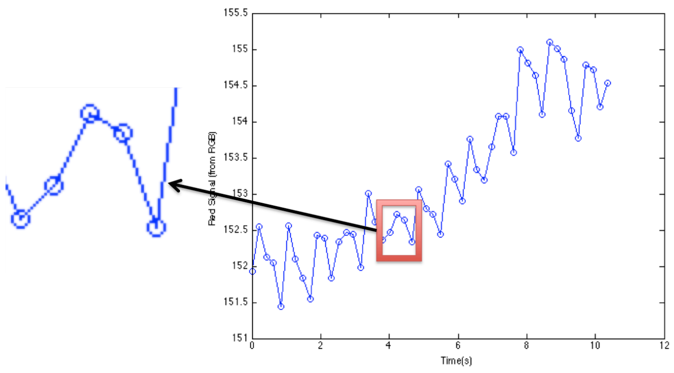


Figure 2.4: Connection of discrete data points

These lines may not be a good approximation of the actual blood movement. This means that the correct interval between pulse events may be off due to the limitations arisen with discrete data points. There is a maximum error of about 100ms since the blood flow is in a single direction; the possibilities of time separation between pulse events are about 100ms, 300ms, 500ms, and so on. The error is explained further later in this thesis in sections 3.3 and 3.4.

The datasheet for the TCS34725 states that the ADC has an integration time that is based on integration cycles. Increasing the number of cycles increases precision but consequently increases the integration time and thus the time between valid sensor readings. The minimum amount of cycles is 1 or 2.4ms integration time, and this corresponds to a maximum count of 1024, which is precise enough to capture pulse events. This is shown in this thesis. Thus, this amount of cycles (1) is selected in this application and 1024 different variations are possible in the data for each color reading. Thus, each iteration requires 2.4ms for the ADC; this is coded in the program with the appropriate hexadecimal value of FF. The value of 2.4ms is ideal because it is small compared to the 100ms separating the transmissions from the sensors (2.4%). This is not an ADC that varies steps based on the color changes; the 2.4ms is fixed. Increasing the integration time would only increase precision and this is unnecessary in this application as previously mentioned.

2.4. Block Diagram and Other Illustrations

The following is the block diagram of the system (see Figure 2.5):

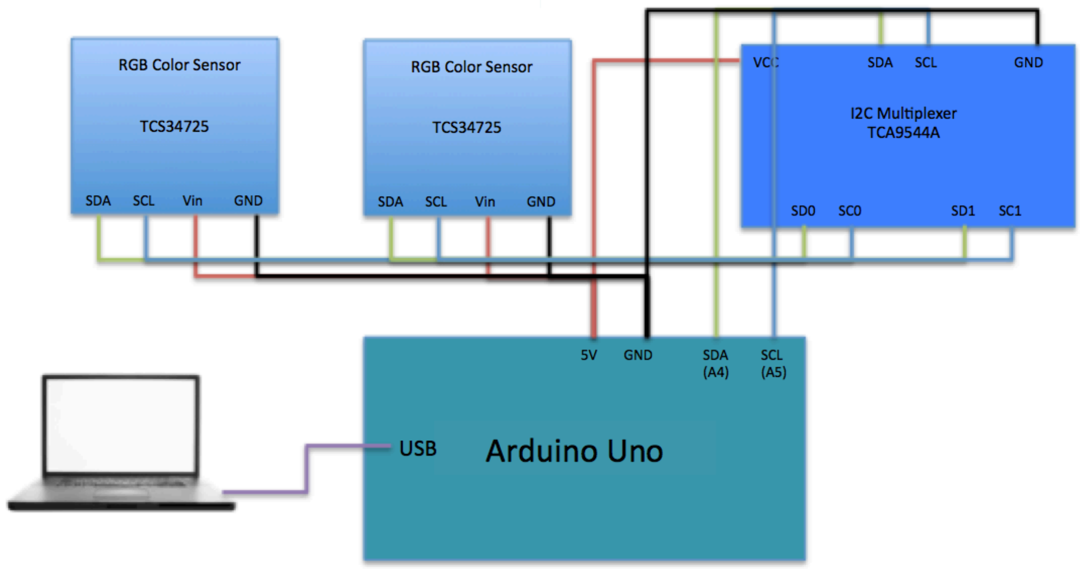


Figure 2.5: System block diagram for full apparatus with multiplexer. This shows the entire construction needed to measure blood flow velocity

The voltage level is 5V for this application; the sensor package provided by Adafruit allows 3.3-5V. Serial data line (SDA) is the line of the I^{2}C protocol that sends and receives data [22]. Serial clock line (SCL) is the line that controls the clock. The TCS34725 sensors have SDA and SCL pins, and the Figure 2.6 illustrates the timing diagram for the TCS34725:

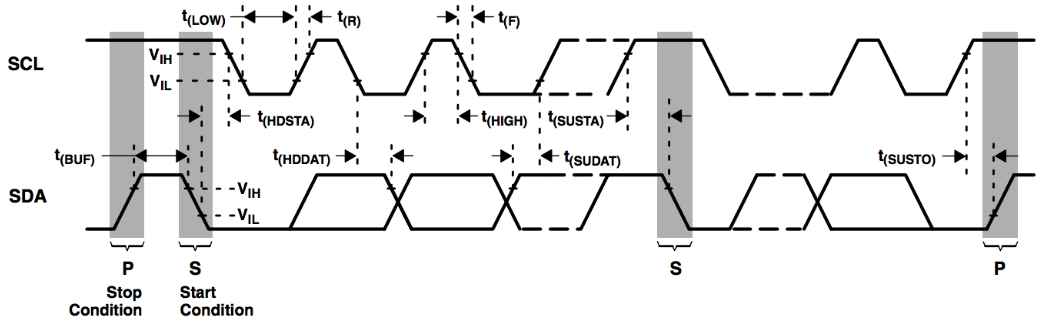


Figure 2.6: TCS34725 SDA and SCL timing diagram to show how data are collected

The SCL and SDA lines have to coordinate with each other in order to send and receive data. The SCL line is high and SDA is sloped down for the start condition. The SCL line is high but the SDA line is sloped up for the stop condition. The information is passed in between the start and stop conditions. On the color sensor package, the LED, INT, and 3v3 pins are unused. The LED pin can be connected to ground when it is desired to turn off the LED. The INT pin can be connected to the LED pin in order to control when the LED is on and off. The 3v3 pin can be used to power the sensor with 3.3V.

For the Arduino Uno, the analog A4 and A5 pins correspond to SDA and SCL, respectively. Figure 2.7 contains an illustration to display where the SDA and SCL pins are located on an Arduino Uno:

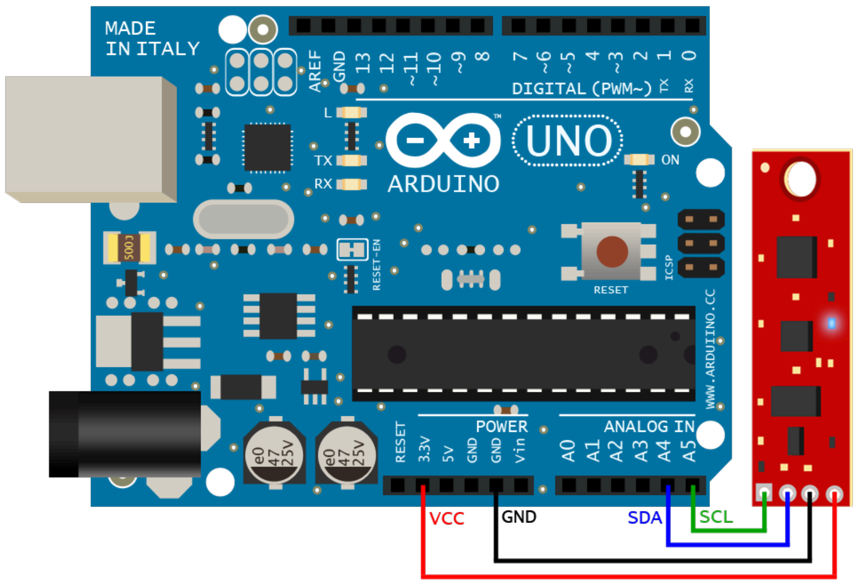


Figure 2.7: Arduino image to show SDA and SCL pins [23] (5V is actually used in this thesis)

The TCA9544A multiplexer has separate SDA and SCL inputs for each color sensor and a single set of SDA and SCL as the output and these are to be connected to the Arduino Uno analog A4 and A5 pins. Figure 2.8 is a schematic of the multiplexer [24]:

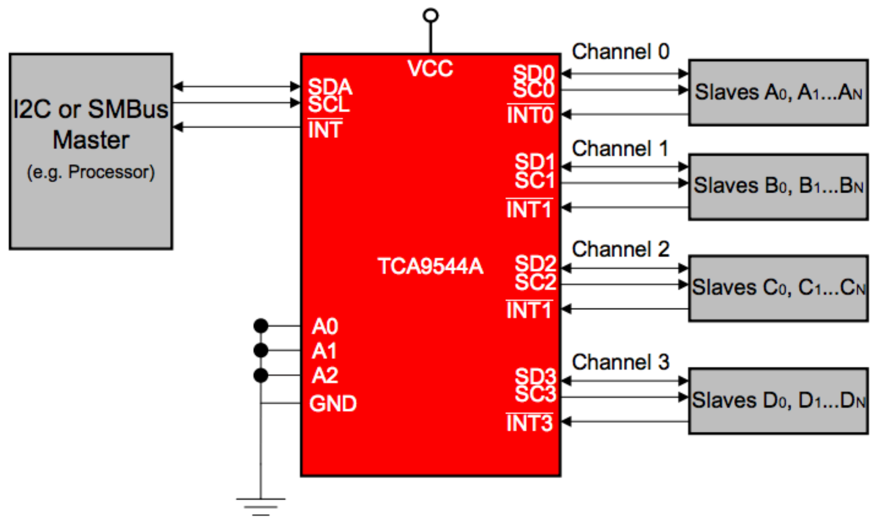


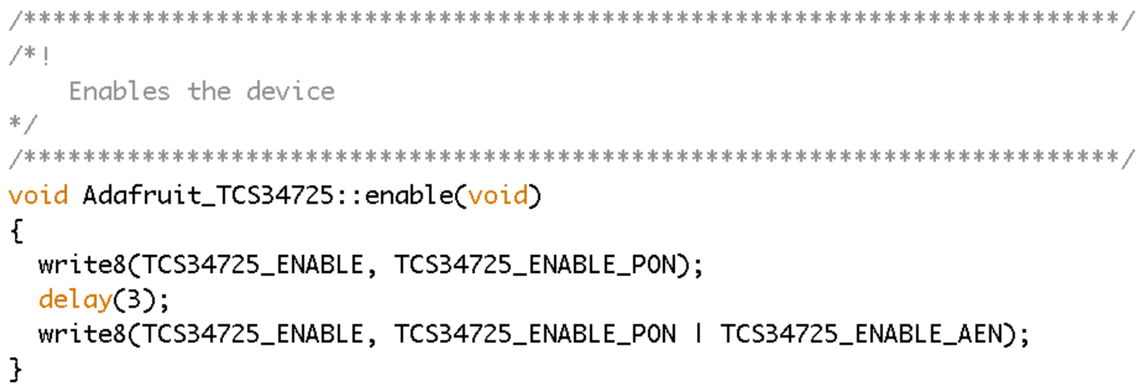
Figure 2.8: Schematic for TCA9544A multiplexer to show how the channels lead to the output and where the analog pins are located

The Arduino Uno is connected to the computer using universal serial bus (USB). From there, the data analysis is performed using the aforementioned Arduino code provided by Adafruit with customized edits and a Python code to show graphical data. The post-processing is then done in MATLAB to find blood velocity.

Chapter 3: Data Collection and Analyses

3.1. Starting the TCS34725 RGB Color Sensor

The Arduino code provided by Adafruit is written to drive one TCS34725 color sensor and it involves three different files: “colorview.ino,” “Adafruit_TCS34725.cpp,” and “Adafruit_TCS34725.h.” The latter two files provide the driver for the TCS34725 and the corresponding header file. The driver includes several functions. First, a “powf” function is written. This function takes two float values and returns a float of the power of the first number raised to the second number. The next functions are written to be able to write a register and read 8 and 16 bit values over I^{2}C. The next function is written for enabling the TCS34725, and this is shown in Figure 3.1:



```
/**************************************************************************/
/*!
    Enables the device
*/
/**************************************************************************/
void Adafruit_TCS34725::enable(void)
{
  write8(TCS34725_ENABLE, TCS34725_ENABLE_PON);
  delay(3);
  write8(TCS34725_ENABLE, TCS34725_ENABLE_PON | TCS34725_ENABLE_AEN);
}
```

Figure 3.1: Function in driver to enable TCS34725

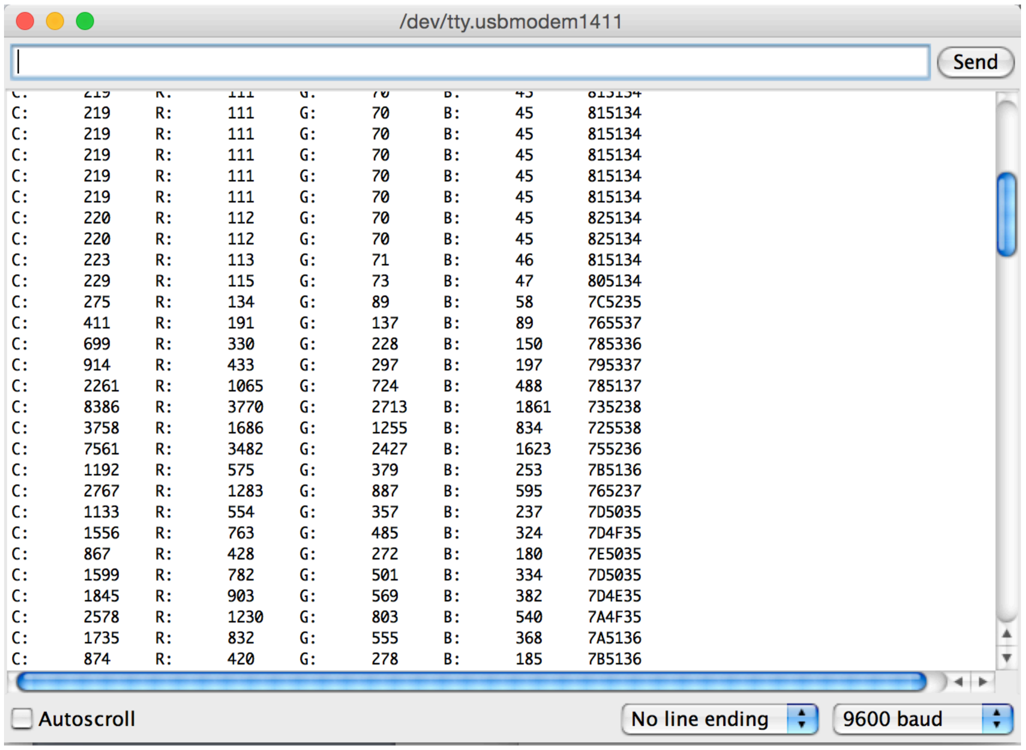
PON stands for “power on” and it allows the timers to start and the ADCs to work. AEN enables the RGBC (C is clear). The next function is written in the event the device needs to be disabled. The 3ms delay only occurs on initialization and not on every write. Thereafter, constructors are written. Then, a function is written to initialize I^{2}C and configure the device. The remaining functions set integration time, adjust gain, reads RGBC data, and converts RGBC data to color temperature in Kelvins. Color temperature corresponds to various hues.

In the “colorview.ino” file, the data processing is performed in order to output the red, green, blue, and clear values along with hexadecimal RGB value. Figure 3.2 displays the main loop that writes the values to serial:

```
void loop() {
  uint16_t clear, red, green, blue;

  tcs.setInterrupt(false);      // turn on LED

  delay(60);  // takes 50ms to read

  tcs.getRawData(&red, &green, &blue, &clear);

  tcs.setInterrupt(true);  // turn off LED

  Serial.print("C:\t"); Serial.print(clear);
  Serial.print("\tR:\t"); Serial.print(red);
  Serial.print("\tG:\t"); Serial.print(green);
  Serial.print("\tB:\t"); Serial.print(blue);

  // Figure out some basic hex code for visualization
  uint32_t sum = clear;
  float r, g, b;
  r = red; r /= sum;
  g = green; g /= sum;
  b = blue; b /= sum;
  r *= 256; g *= 256; b *= 256;
  Serial.print("\t");
  Serial.print((int)r, HEX); Serial.print((int)g, HEX); Serial.print((int)b, HEX);
  Serial.println();


  analogWrite(redpin, gammatable[(int)r]);
  analogWrite(greenpin, gammatable[(int)g]);
  analogWrite(bluepin, gammatable[(int)b]);
}
```

Figure 3.2: Main loop to output data for TCS34725

The values get output to the serial port as follows in Figure 3.3:



```
/dev/tty.usbmodem1411
Send
C:   219    R:   111    G:   70     B:   45     815134
C:   219    R:   111    G:   70     B:   45     815134
C:   219    R:   111    G:   70     B:   45     815134
C:   219    R:   111    G:   70     B:   45     815134
C:   219    R:   111    G:   70     B:   45     815134
C:   220    R:   112    G:   70     B:   45     825134
C:   220    R:   112    G:   70     B:   45     825134
C:   223    R:   113    G:   71     B:   46     815134
C:   229    R:   115    G:   73     B:   47     805134
C:   275    R:   134    G:   89     B:   58     7C5235
C:   411    R:   191    G:   137    B:   89     765537
C:   699    R:   330    G:   228    B:   150    785336
C:   914    R:   433    G:   297    B:   197    795337
C:   2261   R:   1065   G:   724    B:   488    785137
C:   8386   R:   3770   G:   2713   B:   1861   735238
C:   3758   R:   1686   G:   1255   B:   834    725538
C:   7561   R:   3482   G:   2427   B:   1623   755236
C:   1192   R:   575    G:   379    B:   253    7B5136
C:   2767   R:   1283   G:   887    B:   595    765237
C:   1133   R:   554    G:   357    B:   237    7D5035
C:   1556   R:   763    G:   485    B:   324    7D4F35
C:   867    R:   428    G:   272    B:   180    7E5035
C:   1599   R:   782    G:   501    B:   334    7D5035
C:   1845   R:   903    G:   569    B:   382    7D4E35
C:   2578   R:   1230   G:   803    B:   540    7A4F35
C:   1735   R:   832    G:   555    B:   368    7A5136
C:   874    R:   420    G:   278    B:   185    7B5136
Autoscroll    No line ending    9600 baud
```

Figure 3.3: Example output for system with one TCS34725 sensor. Includes red, green, blue, and clear values and overall hexadecimal value

In addition, logic is included in this file to drive a diffused RGB LED to match the color placed in front of the sensor. The following is an example of how it works; an orange is placed over the sensor and the color is matched by the diffused LED, as shown in Figure 3.4:

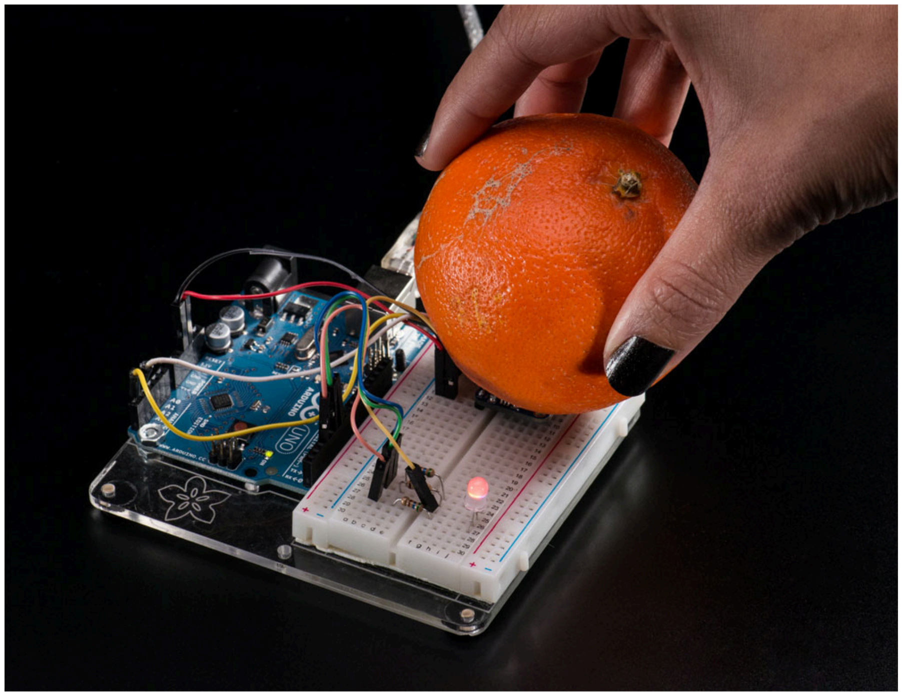


Figure 3.4: Adafruit example for a diffused RGB LED [25]

In the application in this thesis, this setup is used to make some initial tests on colors of random objects. Also, the tests are compared to the RGB values returned from the sensor. The following the comparison of a blue folder placed over the sensor with its output from the TCS34725 (see Figure 3.5):

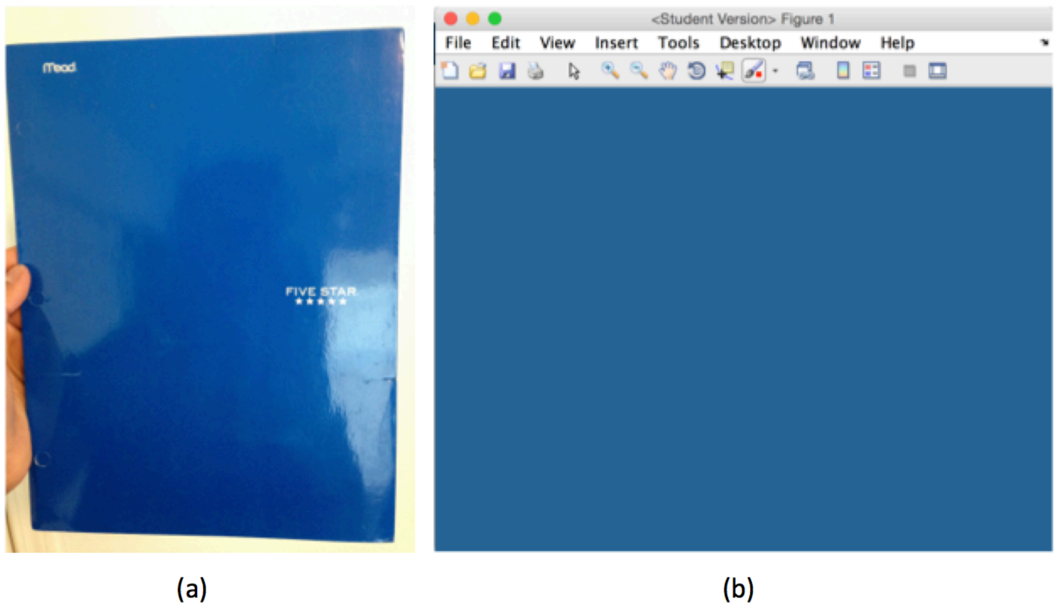

Mead
FIVE STAR
<Student Version> Figure 1
File Edit View Insert Tools Desktop Window Help
(a)
(b)

Figure 3.5: (a) Actual blue folder (b) TCS34725 reading shown in MATLAB

The values for R, G, B and clear are used. The calculation mentioned previously to divide R, G, and B individually by clear is utilized. Multiplication by 255 is not necessary because MATLAB normalizes the RGB values to between 0 and 1. After similar tests are performed to validate the overall color sensor, skin measurements commence.

3.2. Heart Rate Analysis

The initial measurements involved testing the validity of the sensor itself and whether or not it can detect a pulse. It is required that subtle changes in red wavelength get recorded; pulses of around 1Hz should be detectable. For these measurements, analysis on the finger is selected since it is not difficult to isolate arterial color information over the skin on a finger. Only one sensor is used (the other sensor and the multiplexer are omitted in this portion), and the setup is shown in Figure 3.6:

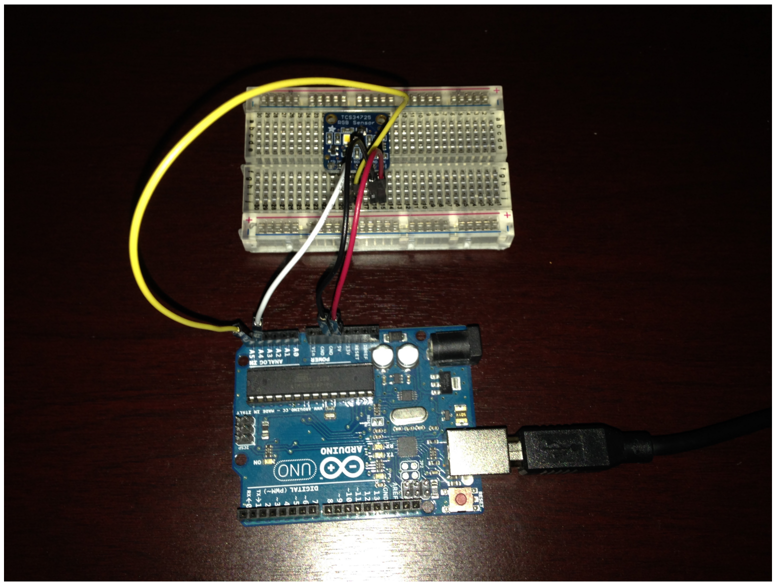


Figure 3.6: Apparatus with one TCS34725 color sensor

The initial measurements are conducted at rest and are thus seeking resting heart rate. The red value is analyzed and graphed over a 22.5 second period; the post-processing is performed in MATLAB. The red value is normalized in RGB (divided by 255), and the output graph is shown in Figure 3.7:

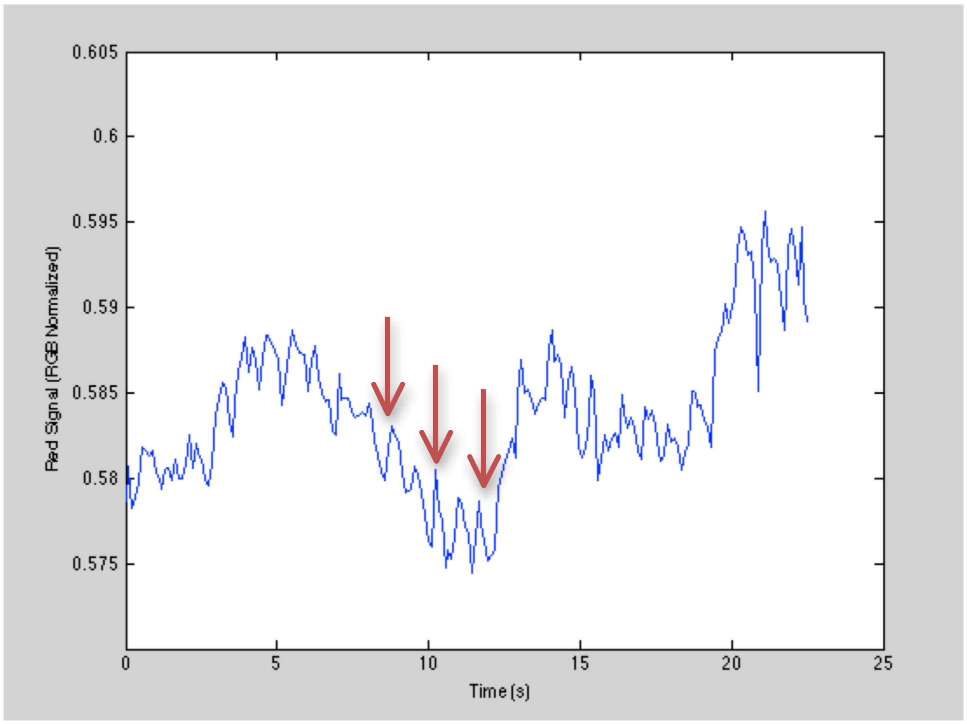


Figure 3.7: Red wavelength using right index finger over 22.5 seconds (at rest). Red arrows are heartbeats

Red arrows in Figure 3.7 correspond to heartbeats. There is a function in MATLAB called ***minpeakdistance***. This function allows the user to find the amount of peaks given a minimum peak distance (5 indices was the selected amount). This means that the peaks must be at least 5 indices apart; this is done to prevent noise from causing errors. The heart rate found is 69 beats per minute (bpm). This is found by taking the number of peaks after using ***minpeakdistance*** and normalizing that amount to 60 seconds.

There is an application available on mobile phones (an iPhone was used) known as "Heart Rate" and it uses the flash and camera to find the heart rate. Its functionality is not explained, but it can be assumed that changes in color due to blood arrival is analyzed. The procedure is simple: place a finger in front of the camera and wait. The heart rate comes after a few seconds of waiting. Azumio is the company that develops this application. Figure 3.8 is an example screen from the application:

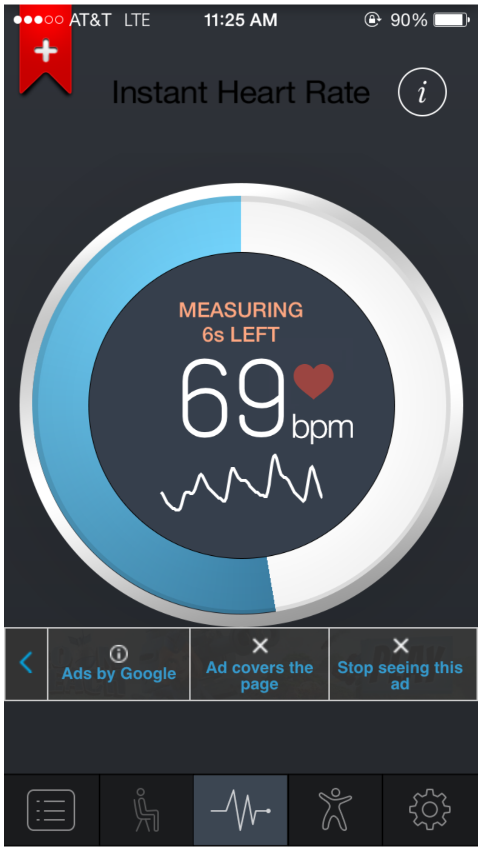


Figure 3.8: Azumio Heart Rate phone application sample screen

The application finds 66 bpm immediately following the test on the TCS34725 from Figure 3.7. It is conceivable that the heart rate would change slightly between the readings. The second analysis for heart rate is performed after a workout of performing 35 push-ups. The test utilizing the TCS34725 is conducted immediately after the workout and the results are shown in the following graph (see Figure 3.9):

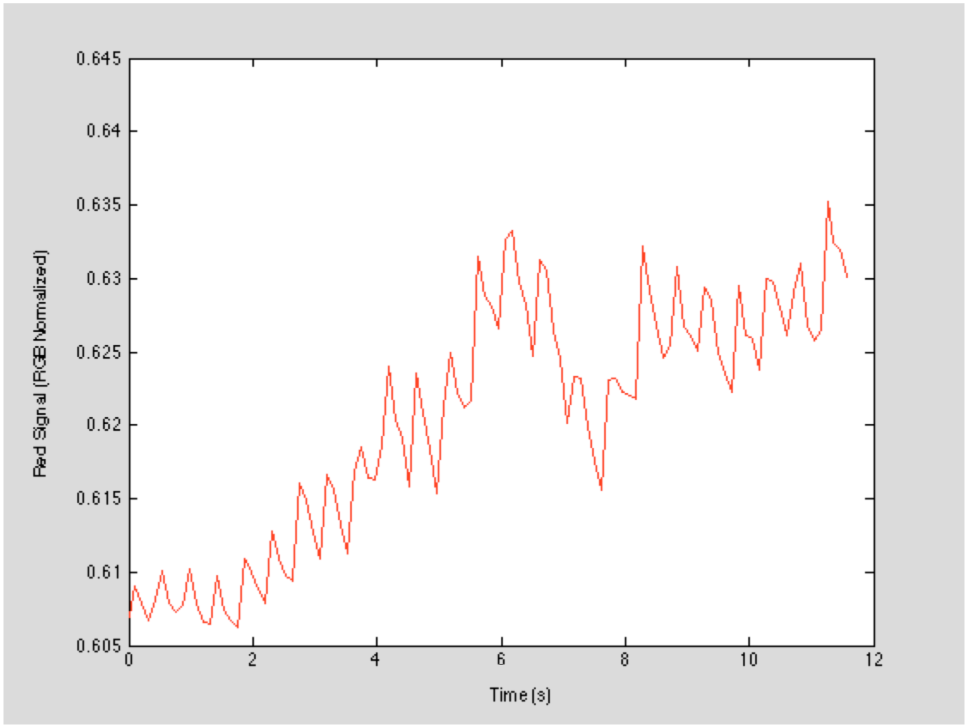
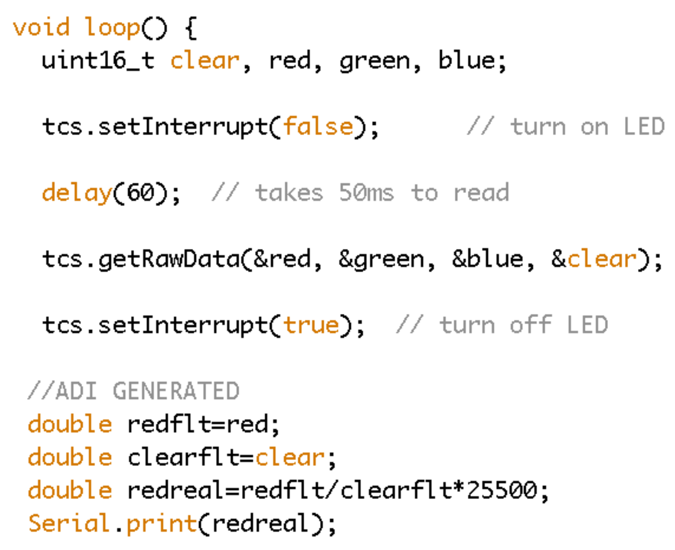


Figure 3.9: Red wavelength using right index finger for 11.6 seconds (after workout)

Since the graph is less noisy than the one from Figure 3.7, it is only required to use the ***findpeaks*** function in MATLAB. The heart rate is 124bpm from the graph. The mobile phone application generated the identical 124bpm immediately afterward. The two tests performed give validity to the TCS34725 sensor.

The next step is to make the data collection process fully automated in Python. To complete this step, the pySerial library in Python needs to be utilized [26]. This library allows a user to communicate with the serial port through Python. Anaconda, which is a free Python distribution by Continuum Analytics with many built-in libraries, is used to make graphing easier [27]. The matplotlib library is used for the graphical analysis. It is desired to output just the red RGB value in Python in order to simplify the code. Thus, the code from Figure 3.2 is modified to leave just the red RGB value for the output (see Figure 3.10):



```
void loop() {
  uint16_t clear, red, green, blue;

  tcs.setInterrupt(false);      // turn on LED

  delay(60);  // takes 50ms to read

  tcs.getRawData(&red, &green, &blue, &clear);

  tcs.setInterrupt(true);  // turn off LED

 //ADI GENERATED
 double redflt=red;
 double clearflt=clear;
 double redreal=redflt/clearflt*25500;
 Serial.print(redreal);
```

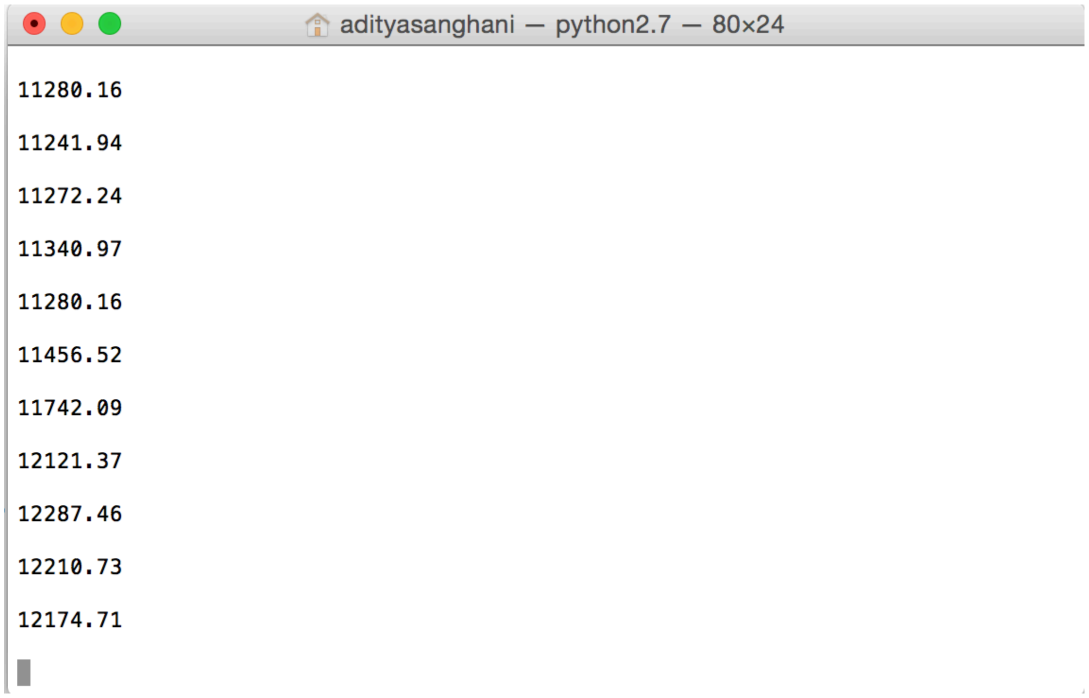
Figure 3.10: Code modification to isolate red RGB value

The RGB value for red is found by dividing the red amount by clear. This value is multiplied by 100 in Arduino to improve precision; the output gives two decimal places. A sample output (without the multiplication) is 107.62 for the red RGB value. With the multiplication, it is 10762.32 and is thus more precise when imported into Python. The values will later get divided by 100 in Python. Figure 3.11 is the equation for the red RGB value multiplied by 100:

$$red\ RGB = \frac{red}{clear} * 255 * 100$$

Figure 3.11: Equation for red RGB value multiplied by 100

Shown in Figure 3.12 is a sample of the output from the serial monitor viewed in Python (red value is multiplied by 100):



```
11280.16

11241.94

11272.24

11340.97

11280.16

11456.52

11742.09

12121.37

12287.46

12210.73

12174.71
```

Figure 3.12: Python output of just red RGB value (multiplied by 100)

In Python, these values are divided by 100. Then, an array of the red RGB value is created in order to store values to be graphed. A loop is created to store several values into the array, and it is shown in Figure 3.13:

```
while i<101:

        a=ser.readline()
        print a
        b=float(a)
        print b

        redarray[i-1]=b/100

        i=i+1
```

Figure 3.13: Loop in Python to store red RGB values into array

Figure 3.14 is output from Python in the fully automated test at rest involving 100 iterations (at 9.2 iterations/second):

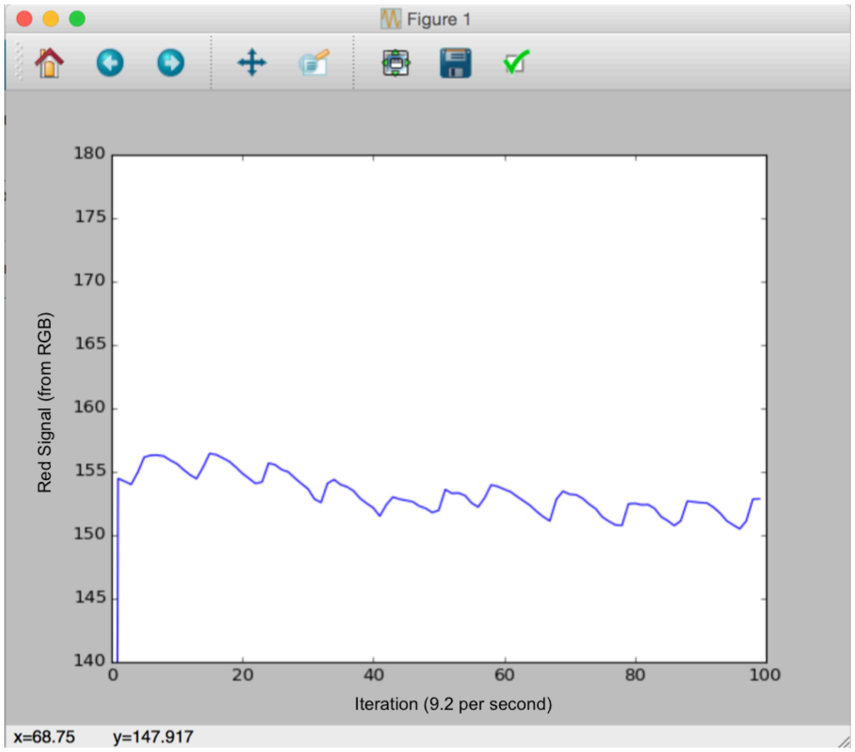


Figure 3.14: Right index finger output analyzed in Python

This analysis shows a heart rate of 57.6bpm. The iPhone application produced 54bpm. Thus, the error for the sensor to heart rate mobile application ranged from 0% to 6.67%. The error for the iPhone application is not noted in its specifications.

3.3. Full Apparatus with Multiplexer

The next step is to successfully implement the TCA9544A multiplexer with two TCS34725 sensors. The multiplexer was ordered from Digikey Corporation in a TSSOP (20) package (see Figure 3.15):

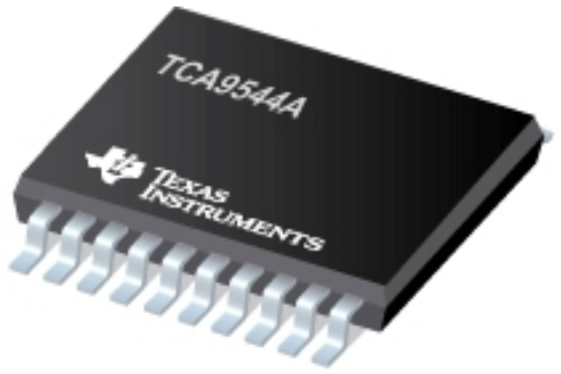


Figure 3.15: Texas Instruments TCA9544A multiplexer [28]

Since it is required to place the multiplexer onto a breadboard, a breakout board compatible with TSSOP (20) is required. This breakout board, shown in Figure 3.16, was purchased from Adafruit Industries:

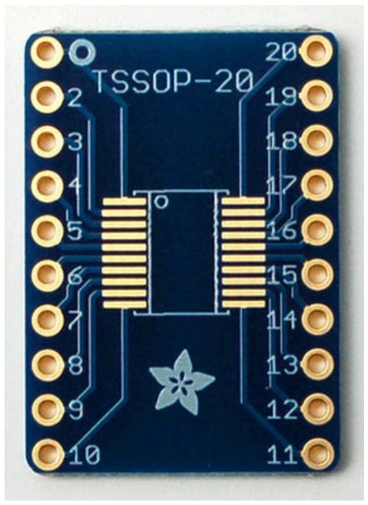


Figure 3.16: TSSOP (20) breakout board [29]

The TCA9544A is attached to the breakout board using solder, flux, and a rework tool. Figure 3.17 is a photo of the two put together:

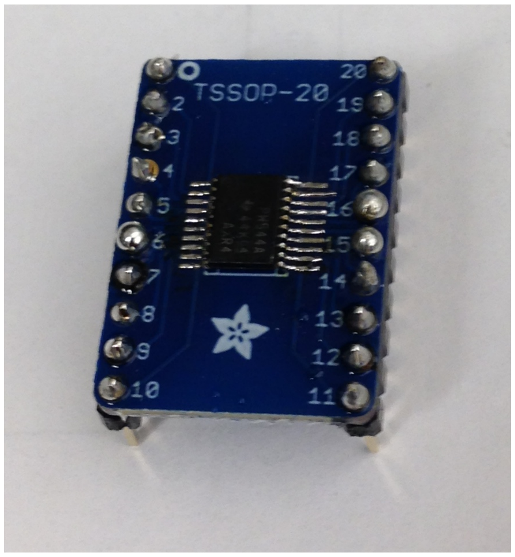


Figure 3.17: TCA9544A multiplexer on breakout board

Header pins are soldered on the sides as shown in Figure 3.17. The next step is to put together the entire to match the block diagram in Figure 2.5. The typical configuration for the multiplexer is shown in the datasheet for the TCA9544A (see Figure 3.18):

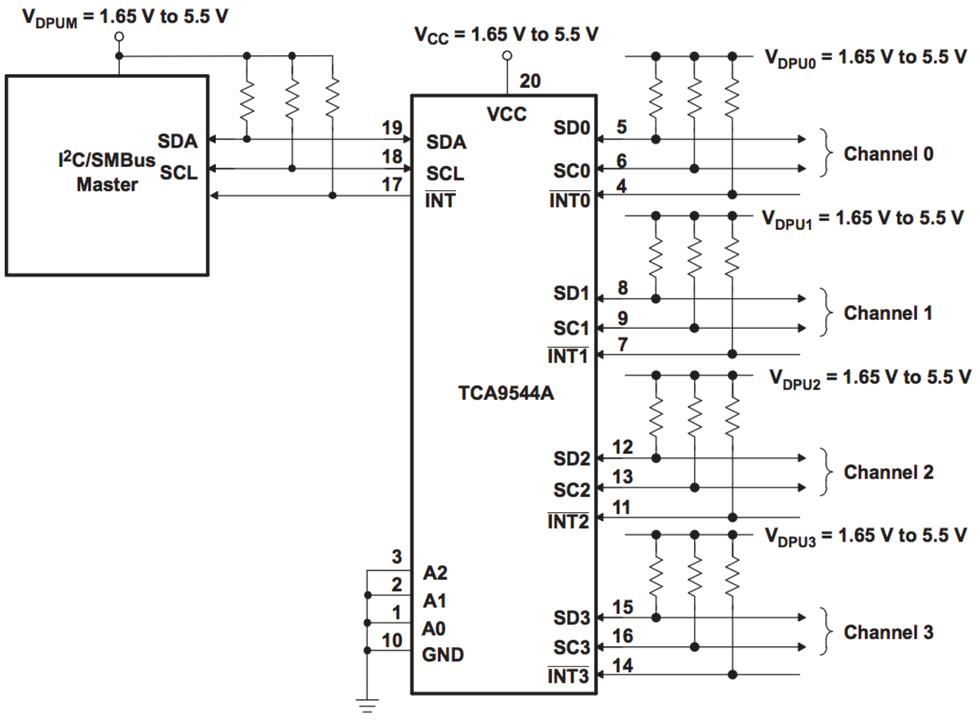


Figure 3.18: Typical configuration for TCA9544A to show full setup

The resistors shown in Figure 3.18 correspond to pull-up resistors; 18k ohm resistors are used. The following is a picture of the complete apparatus (see Figure 3.19):

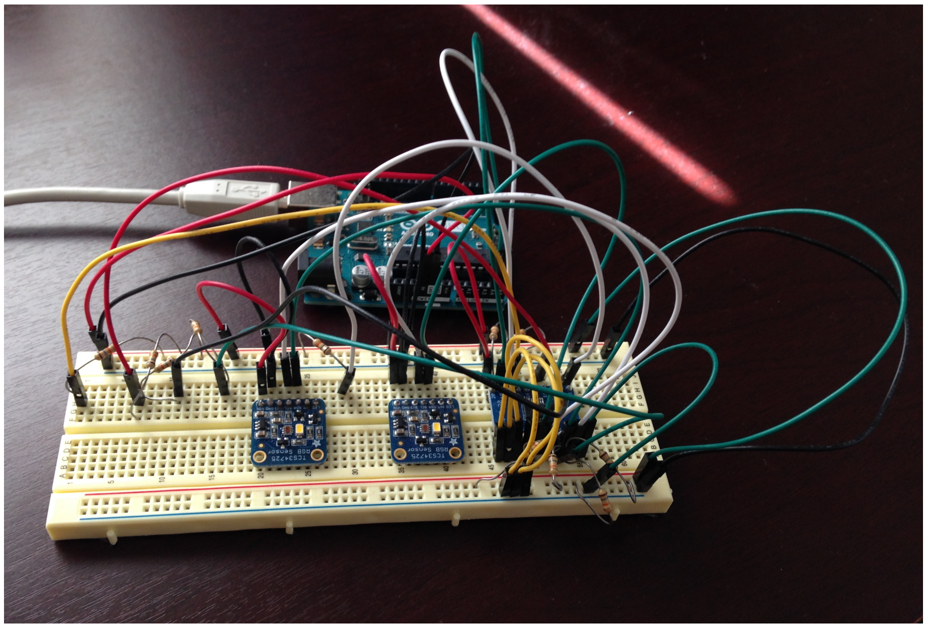


Figure 3.19: Complete apparatus to measure blood flow velocity

The pins A0, A1, and A2 are used in order to select the specific TCA9544A multiplexer. Even though only one multiplexer is used in this application, it is necessary perform this selection. The device address of the TCA9544A is set as follows in Figure 3.20:

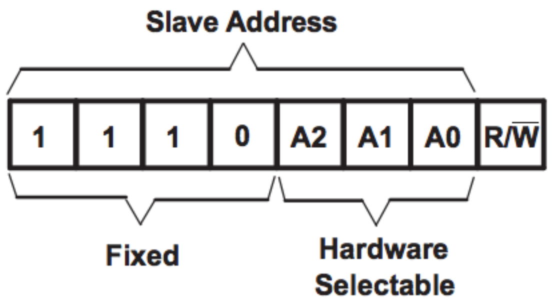
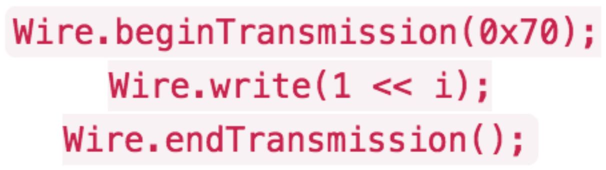


Figure 3.20: Device address for TCA9544A to select individual multiplexer

The values selected for A0, A1, and A2 are 0 for this application (connected to ground), and this corresponds to 1110000, which is a hexadecimal value of 70. The R/W value is not part of the slave address, as indicated in Figure 3.20. The INT pins (See Figure 3.18) correspond to interrupts; these are unused. They are connected to the 5V source through the 18k pull-up resistors because the datasheet mentions not to leave them floating. Keeping them high leaves them off. The SCL and SDA pins for the potential 3rd and 4th sensors are unused and are left floating.

For performing the implementation of the multiplexer, sample code from netopyaplanet.com is used [30]. In the application on this website, a TCA9548A multiplexer is used with 6 TCS34725 sensors. This is an 8 to 1 multiplexer that has virtually the same functionality as the TCA9544A but without interrupt logic (which is unused in the application of this thesis anyway). The sample code provided (see Figure 3.21) has 3 lines of code that switches between sensors:



```
Wire.beginTransmission(0x70);
Wire.write(1 << i);
Wire.endTransmission();
```

Figure 3.21: Code to switch between sensors

The wire.h library was used; this is the Arduino library that allows communication with I^{2}C devices [31]. The first line from Figure 3.21 selects the hardware. The hexadecimal value of 70 is selected because of the previous explanation; the device address from Figure 3.20 is used and the A0, A1, and A2 pins are 0. No change is necessary with this line because the addresses are the same for TCA9548A and TCA9544A. The second line from Figure 3.21 selects the specific sensor. The control register is responsible for the selection of channels that correspond to a sensor. Figure 3.22 is the control register for the TCA9548A [32]:

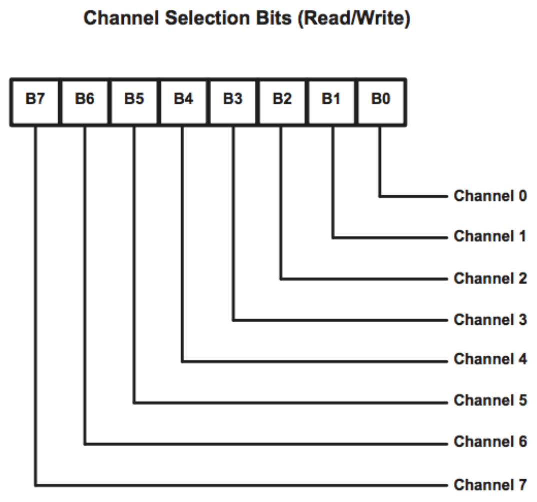


Figure 3.22: Control Register TCA9548A

The TCA9548A has the following truth table (see Table 3.1):

Table 3.1: Truth table to select device on TCA9548A

CONTROL REGISTER BITS								COMMAND
B7	B6	B5	B4	B3	B2	B1	B0	
X	X	X	X	X	X	X	0	Channel 0 disabled
							1	Channel 0 enabled
X	X	X	X	X	X	0	X	Channel 1 disabled
						1		Channel 1 enabled
X	X	X	X	X	0	X	X	Channel 2 disabled
					1			Channel 2 enabled
X	X	X	X	0	X	X	X	Channel 3 disabled
				1				Channel 3 enabled
X	X	X	0	X	X	X	X	Channel 4 disabled
			1					Channel 4 enabled
X	X	0	X	X	X	X	X	Channel 5 disabled
		1						Channel 5 enabled
X	0	X	X	X	X	X	X	Channel 6 disabled
	1							Channel 6 enabled
0	X	X	X	X	X	X	X	Channel 7 disabled
1								Channel 7 enabled
0	0	0	0	0	0	0	0	No channel selected, power-up/reset default state

Given the control register and truth table, it is logical to utilize a bit shift to move through the sensors. That is why Wire.write(1 << i) is used in a ***for loop***; the value is bit shifted one to the left every iteration of the loop. Figure 3.23 is the ***for loop*** to match utilization of 2 sensors using the TCA9548A:

```
void loop() {
  //loop through the color sensors
  for(int i = 0; i<2;i++)
  {
    //Set the i2c switch to the desired output
    Wire.beginTransmission(0x70);
    Wire.write(1 << i);
    Wire.endTransmission();
    //wait for the switch to switch
    delay(1);
    uint16_t clear, red, green, blue;
    tcs[i].setInterrupt(false);      // turn on LED

    delay(50);  // takes 50ms to read

    tcs[i].getRawData(&red, &green, &blue, &clear);

    tcs[i].setInterrupt(true);  // turn off LED

    //Print the data out over serial
                       Serial.print(clear);
    Serial.print(":"); Serial.print(red);
    Serial.print(":"); Serial.print(green);
    Serial.print(":"); Serial.print(blue);

    if(i != 1)
    {
      Serial.print(";");
    }
    else
    {
      Serial.print("\n");
    }
  }
}
```

Figure 3.23: *For loop* to use 2 sensors with the TCA9548A

The TCA9544A has different logic and that is why this loop needs to change. Figure 3.24 shows the control register and truth table for the TCA9544A, and Table 3.2 shows its truth table to select channels:

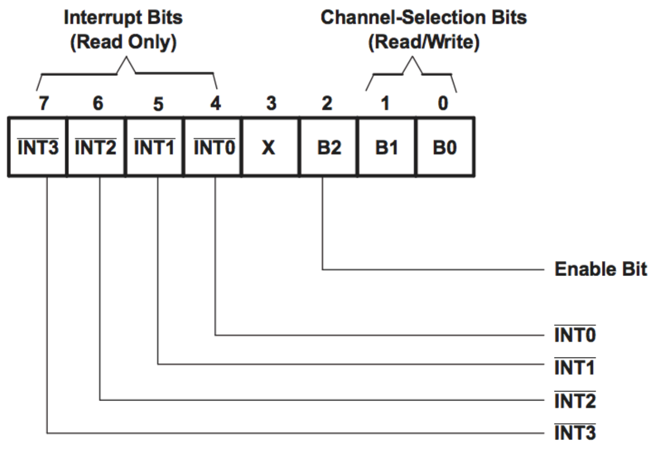


Figure 3.24: Control register TCA9544A

Table 3.2: Truth table to select device on TCA9544A

$\overline{\text{INT3}}$	$\overline{\text{INT2}}$	$\overline{\text{INT1}}$	$\overline{\text{INT0}}$	D3	B2	B1	B0	COMMAND
X	X	X	X	X	0	X	X	No channel selected
X	X	X	X	X	1	0	0	Channel 0 enabled
X	X	X	X	X	1	0	1	Channel 1 enabled
X	X	X	X	X	1	1	0	Channel 2 enabled
X	X	X	X	X	1	1	1	Channel 3 enabled
0	0	0	0	0	0	0	0	No channel selected, power-up default state

Table 3.2 shows that the bit shift will not work. The values run from 4 to 7 in binary, and the ***for loop*** can thus run for those values. This application is adapted to receive data from two color sensors; the loop runs from 4 to 5. The interrupt bits are left as 1. In the loop, the output also needs to be modified; it needs to only send red RGB values to the serial port. Figure 3.25 is the overall corrected ***for loop*** for the TCA9544A:

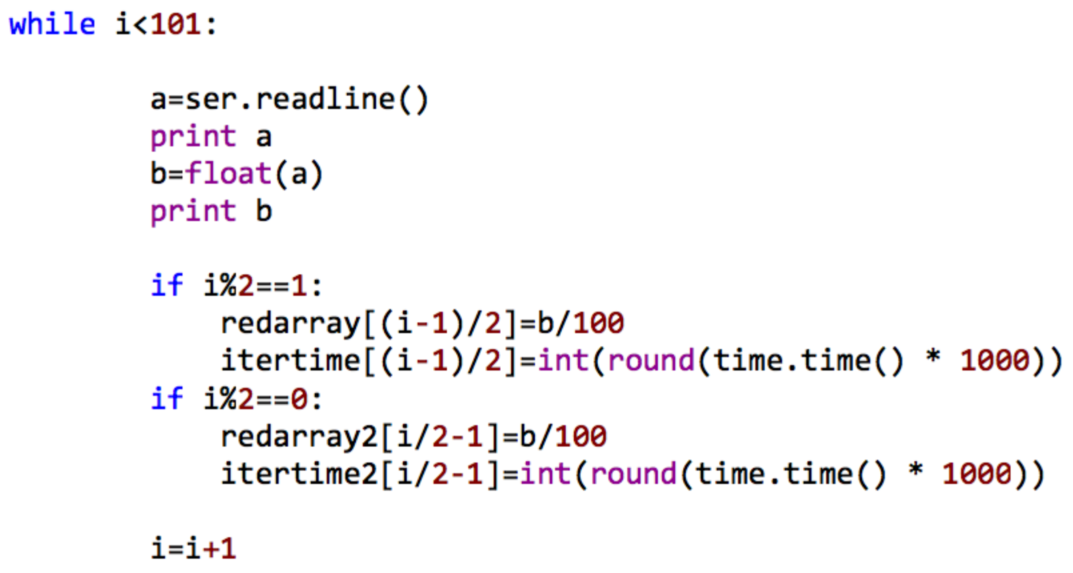
```
void loop() {
  //loop through the color sensors
  for(int i = 0; i<2;i++)
  {
    //Set the i2c switch to the desired output
    Wire.beginTransmission(0x70);
    Wire.write(i+4);
    Wire.endTransmission();
    //wait for the switch to switch
    delay(1);
    uint16_t clear, red, green, blue;
    tcs[i].setInterrupt(false);      // turn on LED
    delay(50);  // takes 50ms to read

    tcs[i].getRawData(&red, &green, &blue, &clear);

    tcs[i].setInterrupt(true);  // turn off LED

    double redflt=red;
    double clearflt=clear;
    double redreal=redflt/clearflt*25500;
    Serial.print(redreal);
    //

    if(i != 1)
    {
      //Serial.print(";");
      Serial.print("\n");
    }
    else
    {
      Serial.print("\n");
    }
  }
}
```

Figure 3.25: Corrected *for loop* to switch between sensors using TCA9544A

The third line from Figure 3.21 does not change. This stops the transmission from the present sensor in question; this is necessary to avoid conflict due to the color sensors having the same address.

The Python code has to be modified in order to accommodate the two sensors. A second array has to be made. Additionally, the timestamps for the received data need to be recorded in order to keep track of the time lag between the two sensors. This time lag remains approximately 100ms between transmissions of the two sensors (102ms-107ms as shown later in this section).

Since it is necessary to find differences in the millisecond level, the epoch time is taken in milliseconds. Figure 3.26 is the loop modified from Figure 3.13:



```
while i<101:

        a=ser.readline()
        print a
        b=float(a)
        print b

        if i%2==1:
            redarray[(i-1)/2]=b/100
            itertime[(i-1)/2]=int(round(time.time() * 1000))
        if i%2==0:
            redarray2[i/2-1]=b/100
            itertime2[i/2-1]=int(round(time.time() * 1000))

        i=i+1
```

Figure 3.26: Loop to collect and store data from 2 color sensors

The round function is used to get the epoch time in milliseconds and the int function makes it an integer value without decimal places. Epoch time here gives the number of milliseconds since January 1, 1970. Figure 3.27 is the at rest recording from an index finger while keeping the two sensors 3.85cm apart:

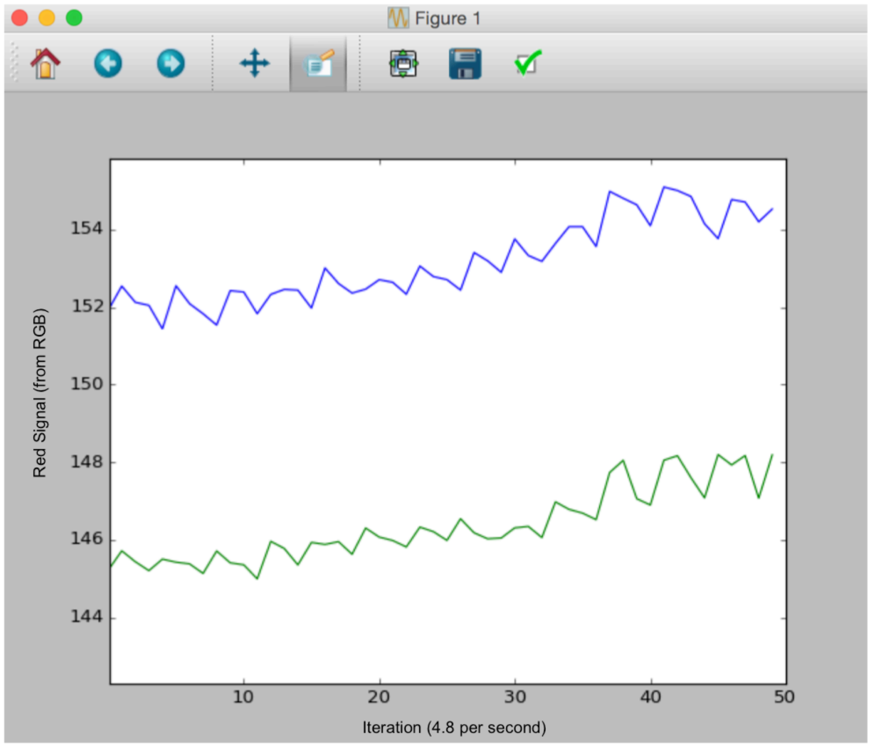


Figure 3.27: Right index finger readings with full apparatus

The array data are then taken into MATLAB for post-processing in order to determine blood flow velocity. Figure 3.28 is the equation used to find blood flow velocity:

$$velocity = \frac{\Delta x}{\Delta t}$$

Figure 3.28: Velocity calculation to obtain blood flow velocity

In this application, Δx is the distance between the sensors (3.85cm) and Δt is the time between events. The value for Δt is found by subtracting the epoch time from the event in question on one sensor from the epoch time from the event in question from the other sensor. The delay between transmissions from the two sensors, as mentioned previously, remains approximately 100ms. The actual range is 102ms-107ms due to the 2.4ms integration time and additional time to carry out the functions in the codes. This gives 107ms for the maximum error because the maximum time between sensor readings is 107ms, 321ms, 535ms... and the actual value could be directly in between those points, or 107ms away from the received value from the code. The reason why these are the possible values is because the blood flows in a single direction, there is a range of 102ms-107ms between sensor readings(between different sensors, let this be X), and there is a range of 204ms-214ms between sensor readings of an individual sensor (let this be Y). So, the following equation in Figure 3.29 can be used to characterize possible time differences between sensor readings (for use in calculation of velocity):

$$\Delta t = X + nY$$

Figure 3.29: Equation to characterize possible time difference between pulse events

n=0,1,2,3... in this equation. It changes based on how fast the blood is moving.

Shown in Figure 3.30 is the method by which the value for Δt is found:

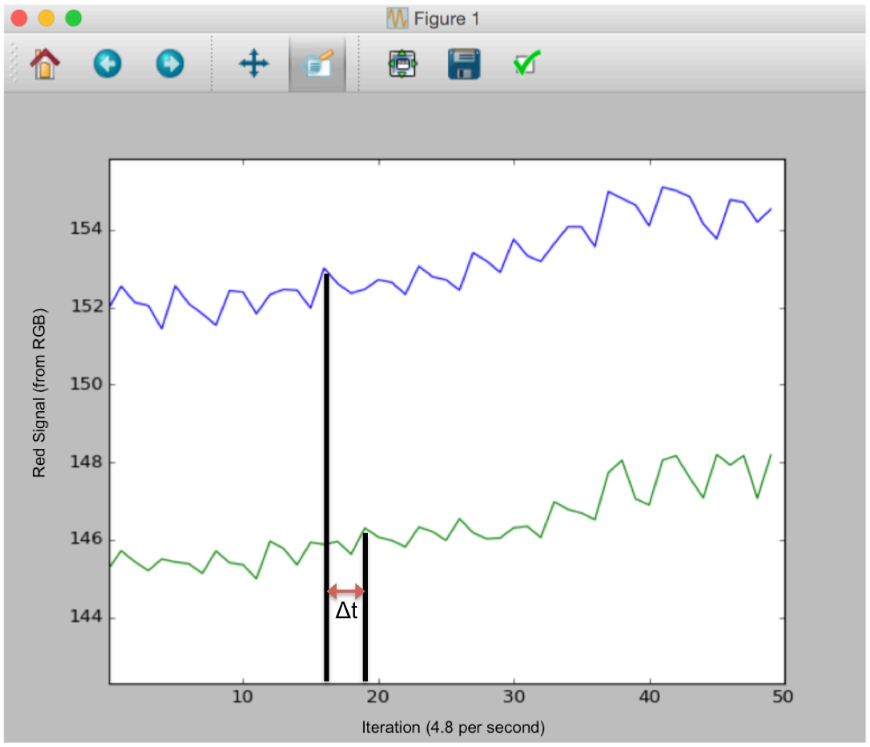


Figure 3.30: Example time difference between events

The peaks are identified and the difference between the peaks is found. As mentioned previously, taking the difference between the epoch times of the data points gives Δt. After that, the value for Δx is divided by the Δt value to get the average velocity in that time. These individual values are averaged throughout the sample to find an overall blood flow velocity for the entire sample. Shown in Figure 3.31 is a sample calculation for velocity using the points from Figure 3.30:

$$velocity = \frac{\Delta x}{\Delta t} = \frac{3.85cm}{0.741s} = 5.20cm/s$$

Figure 3.31: Sample blood flow velocity calculation

Five samples from the above graph are taken and used for analysis. Evidently similar waveforms are taken and compared such as the example shown in Figure 3.30. They are taken throughout the graph as indicated by the indices. Table 3.3 shows the extracted data:

Table 3.3: Blood flow velocity information for right index finger

Blood Flow Velocity Values (cm/s)	distance between sensors (cm)	pulse 2 index	pulse 1 index	time between events (s)
5.2	3.85	20	17	0.741
7.29	3.85	24	22	0.528
5.2	3.85	34	31	0.741
12.22	3.85	39	38	0.315
7.28	3.85	16	14	0.529
AVERAGE Blood Flow Velocity (cm/s)				
7.44				

The blood flow velocity values range from 5.20cm/s to 12.22cm/s. The one high value of 12.22cm/s is an outlier and it may be due to the finger vibrating or being pressed harder in the test. The average value is 7.44cm/s. Since the maximum error associated with the time between sensor readings is 107ms, the actual average velocity is between 6.17cm/s and 9.39cm/s:

$$avg\ time = \frac{3.85cm}{7.44cm/s} = 0.517s$$

Figure 3.32: Average time for blood to travel 3.85cm on finger

$$lowest\ avg\ velocity = \frac{3.85cm}{0.517s + .107s} = 6.17cm/s$$

Figure 3.33: Lowest possible average velocity (using maximum error)

$$highest\ avg\ velocity = \frac{3.85cm}{0.517s - .107s} = 9.39cm/s$$

Figure 3.34: Highest possible average velocity (using maximum error)

The overall range of values compares well to those from published data (4cm/s-10cm/s and 4.9cm/s-19cm/s), thus further giving validity to the sensor and to the calculation for blood flow velocity in the finger.

3.4. Mean Squared Error

In order to find the most reliable time difference between events in the data, the concept of mean squared error is employed. The graphs from Figure 3.27 are brought to be centered at zero by subtracting them by their respective averages. The reason why they are off vertically is because there is a difference in red wavelength based on how hard a person presses on each sensor. This does not affect the data for the pulses, but the centering at zero had to be performed to compare the data, as shown in Figure 3.35:

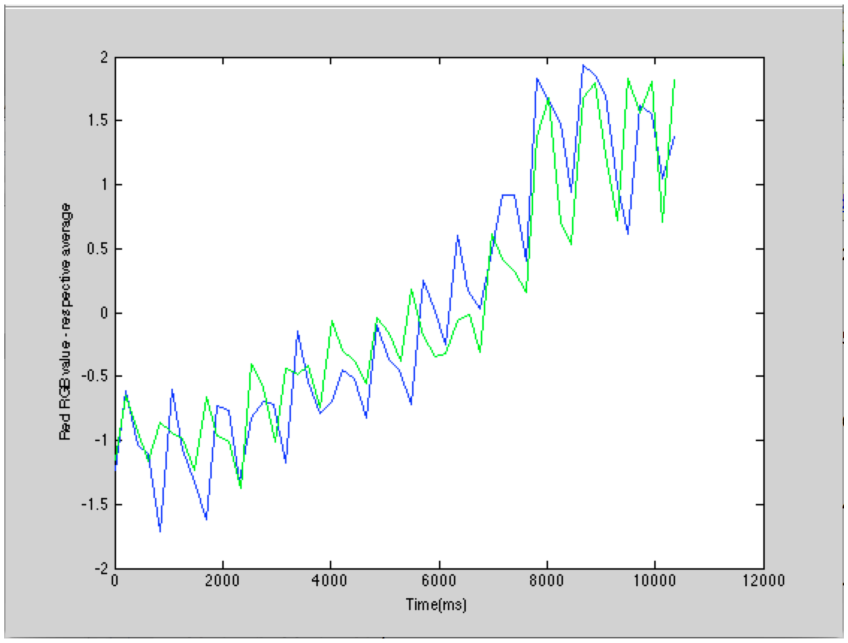


Figure 3.35: Graphs subtracted by their respective means

After this, one graph had to be translated by different time amounts in order to make comparisons. A MATLAB function was generated to perform the following calculation for mean squared error (see Figure 3.36):

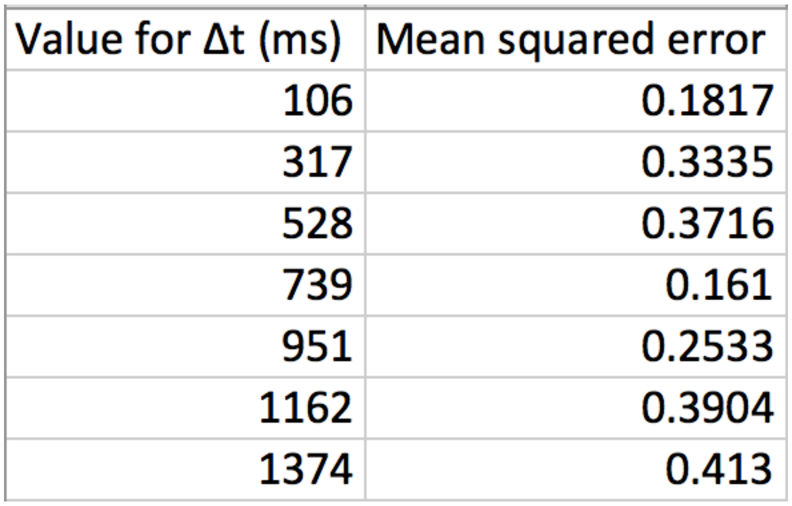
$$mean\ squared\ error = \frac{1}{N}\sum_{i=1}^{N}[X_2(i) - X_1(i + \Delta t)]^2$$

Figure 3.36: Equation for mean squared error

In this equation, N is the number of iterations in the data, i is an iteration, and X_2 and X_1 represent the pulses.

Table 3.4 contains the results for the different values of Δt:

Table 3.4: Mean squared error for different values of delta t



Value for Δt (ms)	Mean squared error
106	0.1817
317	0.3335
528	0.3716
739	0.161
951	0.2533
1162	0.3904
1374	0.413

With this analysis, it is clear that 739ms is the most reasonable value for Δt and thus the most likely time separation among all of the values. This is because its mean squared error is the least; the corresponding blood flow velocity value is 5.21cm/s. The value of 106ms corresponds to a blood flow velocity of 36.32 cm/s, which is too large to be reasonable as indicated by the aforementioned published studies.

Chapter 4: Conclusion and Future Work

4.1. Conclusion

The TCS34725 RGB color sensor functions for color characterization of the skin. The initial readings to measure heart rate illustrated that the color sensor was capable of keeping up with pulses and thus the movement of blood in the finger. This test was successfully verified with the Azumio “Heart Rate” mobile application. Next, the TCA9544A I^2C multiplexer was implemented and coded. There was a 102ms-107ms delay between alternating sensors, and this introduced potential error into the system. The two sensors were still able to collect enough data to see two clear waveforms, and the waveforms were compared at peaks to obtain blood flow velocity. The velocity values were analyzed and matched with ranges found from published data. The range found for finger blood flow velocity in the experiments ranged from 5.20cm/s to 12.22cm/s. The average value was 7.44cm/s. With taking into maximum error of 107ms, the actual average blood flow velocity is between 6.17cm/s and 9.39cm/s as mentioned previously. Utilization of mean squared error found that the most likely time separation between pulse events from those available from the setup is 739ms, which corresponds to a blood flow velocity of 5.21cm/s. The values obtained for blood flow velocity are realistic because the range found from published data was 4cm/s to 19cm/s. The apparatus can easily detect velocity changes of 19.2%; this was the figure from the introduction for the decrease in blood flow velocity from normal patients to that of those experiencing ischemic stroke. The total cost for the components was $99.66, which is significantly cheaper than existing methods ($8500-$20000). Overall, the project investigated a method to measure blood flow velocity in the finger in a far more economical manner than current methods.

4.2. Future Work

There are certain future considerations for this project. First, it would be beneficial to utilize a sensor that has less than 50ms of a read time or can have its address changed. This would reduce the overall error in the system. If using the same sensor, it would be useful to test the system without the additional 50ms delay used for fidelity. This can be done with a function generator with LEDs. If successful, this 50ms delay can be removed and all of the data collection can be redone with the overall time between transmissions reduced from 102ms-107ms to 52ms-57ms. Next, it would be helpful to create an algorithm to make the measurement process fully automated. It is already very automated by collecting two waveforms for a given amount of time. The next step is to automatically develop a matrix of velocity values and to average that value. This can be performed if the graphs are smooth enough to have accurate peaks and the difference in phase can be calculated very easily using the ***findpeaks*** function in MATLAB. Next, it would be advantageous to make a full casing to make the measurement process more comfortable and to make the readings more uniform from test to test. Another future consideration would be to add variables into the system and test differences. For example, heating and cooling of the finger prior to the test could provide interesting results. After that, various other parts of the body could be tested such as the femoral and carotid arteries.

This project can move further and become a prototype in the future.

REFERENCES

[1] Bulwer, Bernard E., and Jose M. Rivero. *Echocardiography Pocket Guide: The Transthoracic Examination*. Sudbury, MA: Jones and Bartlett, 2011. Print.

[2] "Atherosclerosis: Causes, Symptoms, Tests, and Treatment - WebMD." *WebMD*. WebMD, 2005-2015. Web. 05 June 2015.

[3] "Problem: Heart Valve Regurgitation." *Problem: Heart Valve Regurgitation*. N.p., 2015. Web. 05 June 2015.

[4] KlarhFer, M., B. Csapo, Cs. Balassy, J.c. Szeles, and E. Moser. "High-resolution Blood Flow Velocity Measurements in the Human Finger." *Magn. Reson. Med. Magnetic Resonance in Medicine* 45.4 (2001): 716-19. Web.

[5] Rojanapongpun, Prin, and Stephen M. Drance. "The Response of Blood Flow Velocity in the Ophthalmic Artery and Blood Flow of the Finger to Warm and Cold Stimuli in Glaucomatous Patients." *Graefe's Archive for Clinical and Experimental Ophthalmology Graefe's Arch Clin Exp Ophthalmol* 231.7 (1993): 375-77. Web.

[6] *Raynaud's Phenomenon: A Guide to Pathogenesis and Treatment*. New York: Springer, 2015. Print.

[7] Bergersen, T. K., L. Eriksen M Fau - Walloe, and L. Walloe. "Effect of Local Warming on Hand and Finger Artery Blood Velocities."0002-9513 (Print). Print.

[8] "Palmar Digital Artery." *Palmar Digital Artery Anatomy, Function & Diagram*.

N.p., n.d. Web. 05 June 2015.

[9] *Practical Plastic Surgery* (n.d.): n. pag. *NERVE AND VASCULAR INJURIES OF THE HAND*. Web. 22 June 2015.

[10] Bai, Chyi-Huey, et al. "Lower Blood Flow Velocity, Higher Resistance Index, and Larger Diameter of Extracranial Carotid Arteries Are Associated with Ischemic Stroke Independently of Carotid Atherosclerosis and Cardiovascular Risk Factors." *Journal of Clinical Ultrasound* 35.6 (2007): 322-30. Print.

[11] Rådegran, G. "Ultrasound Doppler Estimates of Femoral Artery Blood Flow during Dynamic Knee Extensor Exercise in Humans." (1997): n. pag. The American Physiological Society. Web. <http://www.harvardapparatusregen.com/media/harvard/pdf/PP54UL~1.PDF>.

[12] Powell, Andrew J. "Evaluation of Blood Flow by Evaluation of Blood Flow by Magnetic Resonance Imaging." *Non-Invasive Cardiac Imaging Seminars* (n.d.): n. pag. Children's Hospital Boston, MA. Web. 22 June 2015.

[13] "Doppler Ultrasound." *WebMD*. WebMD, n.d. Web. 23 June 2015.

[14] *What Is Ultrasound?* Diason Ultrasound Center, n.d. Web. 23 June 2015.

[15] "Echocardiogram." *HeartSite.com*. N.p., n.d. Web. 22 June 2015.

[16] Hoskins, P. R. "Measurement of Arterial Blood Flow by Doppler Ultrasound."0143-0815 (Print). Print.

[17] "Doppler Echocardiography Machine." *Google*. N.p., n.d. Web. 21 June

2015.

[18] "MRI." Mayo Clinic, n.d. Web. 21 June 2015.

[19] "MRI Machine." *Google*. N.p., n.d. Web. 22 June 2015.

[20] 20, Taos135 − August. *TCS3472 COLOR LIGHT-TO-DIGITAL CONVERTER with IR FILTER*. Plano, TX: Texas Advanced Optoelectronic Solutions Inc., 2012. PDF.

[21] "RGB Color Sensor with IR Filter and White LED - TCS34725." *Adafruit Industries Blog RSS*. N.p., n.d. Web. 05 June 2015. <http://www.adafruit.com/products/1334?gclid=CNCSuqns6MUCFcEkgQodlVQACA>.

[22] "Using the I2C Bus." *I2C Tutorial*. N.p., n.d. Web. 05 June 2015. <http://www.robot-electronics.co.uk/acatalog/I2C_Tutorial.html>.

[23] *9DOF IMU Stick Connecting*. Digital image. N.p., n.d. Web. 5 June 2015. <http://1.bp.blogspot.com/-xUCQu2NV8Fs/T3YUaMhhuyI/AAAAAAAABWE/YahHuVnHHA0/s1600/9DOF+IMU+Stick+Connecting.png>.

[24] "Multiplexer Channel." *SpringerReference* (2011): n. pag. Texas Instruments, May 2014. Web. 5 June 2015. <http://www.ti.com/lit/ds/scps209a/scps209a.pdf>.

[25] *Light_1334orange_LRG*. Digital image. N.p., n.d. Web. 5 June 2015. <https://learn.adafruit.com/system/assets/assets/000/008/659/original/light_1334orange_LRG.jpg?1396869943>.

[26] "PySerial." *PySerial — PySerial 2.7 Documentation*. N.p., n.d. Web. 05 June

2015. <http://pyserial.sourceforge.net/pyserial.html>.

[27] "Anaconda." *Scientific Python Distribution*. N.p., n.d. Web. 05 June 2015. <https://store.continuum.io/cshop/anaconda/>.

[28] *TCA9544A*. Digital image. N.p., n.d. Web. 5 June 2015. <http://www.ti.com/graphics/folders/partimages/TCA9544A.jpg>.

[29] "SMT Breakout PCB for SOIC-20 or TSSOP-20 - 3 Pack!" *Adafruit Industries Blog RSS*. N.p., n.d. Web. 05 June 2015. <https://www.adafruit.com/products/1206>.

[30] "Controlling Multiple I2C Devices with Arduino." *Netopya Planet*. N.p., n.d. Web. 05 June 2015. <http://www.netopyaplanet.com/article.php?id=6>.

[31] "Arduino - Wire." *Arduino - Wire*. N.p., n.d. Web. 05 June 2015. <http://www.arduino.cc/en/Reference/Wire>.

[32] *TCA9548A Low-Voltage 8-Channel I 2C Switch With Reset* (n.d.): n. pag. Texas Instruments. Web. 10 June 2015.

APPENDICES

“colorview.ino” – code used to operate one TCS34725 color sensor. Its output includes red, green, blue, and clear values

```
 #include <Wire.h>
#include "Adafruit_TCS34725.h"

// Pick analog outputs, for the UNO these three work well
// use ~560  ohm resistor between Red & Blue, ~1K for green (its brighter)
#define redpin 3
#define greenpin 5
#define bluepin 6
// for a common anode LED, connect the common pin to +5V
// for common cathode, connect the common to ground

// set to false if using a common cathode LED
#define commonAnode true

// our RGB -> eye-recognized gamma color
byte gammatable[256];


Adafruit_TCS34725 tcs =
Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_50MS,
TCS34725_GAIN_4X);

void setup() {
  Serial.begin(9600);
  Serial.println("Color View Test!");

  if (tcs.begin()) {
    Serial.println("Found sensor");
  } else {
    Serial.println("No TCS34725 found ... check your connections");
    while (1); // halt!
  }

  // use these three pins to drive an LED
  pinMode(redpin, OUTPUT);
  pinMode(greenpin, OUTPUT);
  pinMode(bluepin, OUTPUT);

  // thanks PhilB for this gamma table!
  // it helps convert RGB colors to what humans see
  for (int i=0; i<256; i++) {
    float x = i;
    x /= 255;
    x = pow(x, 2.5);
    x *= 255;

    if (commonAnode) {
      gammatable[i] = 255 - x;
    } else {
      gammatable[i] = x;
    }
    //Serial.println(gammatable[i]);
  }
}


void loop() {
  uint16_t clear, red, green, blue;

  tcs.setInterrupt(false);      // turn on LED

  delay(60);  // takes 50ms to read

  tcs.getRawData(&red, &green, &blue, &clear);

  tcs.setInterrupt(true);  // turn off LED

  Serial.print("C:\t"); Serial.print(clear);
  Serial.print("\tR:\t"); Serial.print(red);
  Serial.print("\tG:\t"); Serial.print(green);
  Serial.print("\tB:\t"); Serial.print(blue);

  // Figure out some basic hex code for visualization
  uint32_t sum = clear;
  float r, g, b;
  r = red; r /= sum;
  g = green; g /= sum;
  b = blue; b /= sum;
  r *= 256; g *= 256; b *= 256;
  Serial.print("\t");
  Serial.print((int)r, HEX); Serial.print((int)g, HEX); Serial.print((int)b, HEX);
  Serial.println();

  //Serial.print((int)r ); Serial.print(" "); Serial.print((int)g);Serial.print(" ");
Serial.println((int)b );

  analogWrite(redpin, gammatable[(int)r]);
  analogWrite(greenpin, gammatable[(int)g]);
  analogWrite(bluepin, gammatable[(int)b]);
}
```

“Adafruit_TCS34725.cpp” – Driver for TCS34725

```
/**************************************************************************/
/*!
   @file     Adafruit_TCS34725.cpp
   @author   KTOWN (Adafruit Industries)
   @license  BSD (see license.txt)

   Driver for the TCS34725 digital color sensors.

   Adafruit invests time and resources providing this open source code,
   please support Adafruit and open-source hardware by purchasing
   products from Adafruit!

   @section  HISTORY

   v1.0 - First release
*/
/**************************************************************************/
#include <avr/pgmspace.h>
#include <util/delay.h>
#include <stdlib.h>
#include <math.h>

#include "Adafruit_TCS34725.h"

/*========================================================================
============*/
/*                        PRIVATE FUNCTIONS                            */
/*========================================================================
============*/

/**************************************************************************/
/*!
   @brief  Implements missing powf function
*/
/**************************************************************************/
float powf(const float x, const float y)
{
  return (float)(pow((double)x, (double)y));
}

/**************************************************************************/
/*!
   @brief  Writes a register and an 8 bit value over I2C
*/
```

```
/**************************************************************************/
void Adafruit_TCS34725::write8 (uint8_t reg, uint32_t value)
{
  Wire.beginTransmission(TCS34725_ADDRESS);
  #if ARDUINO >= 100
  Wire.write(TCS34725_COMMAND_BIT | reg);
  Wire.write(value & 0xFF);
  #else
  Wire.send(TCS34725_COMMAND_BIT | reg);
  Wire.send(value & 0xFF);
  #endif
  Wire.endTransmission();
}

/**************************************************************************/
/*!
    @brief  Reads an 8 bit value over I2C
*/
/**************************************************************************/
uint8_t Adafruit_TCS34725::read8(uint8_t reg)
{
  Wire.beginTransmission(TCS34725_ADDRESS);
  #if ARDUINO >= 100
  Wire.write(TCS34725_COMMAND_BIT | reg);
  #else
  Wire.send(TCS34725_COMMAND_BIT | reg);
  #endif
  Wire.endTransmission();

  Wire.requestFrom(TCS34725_ADDRESS, 1);
  #if ARDUINO >= 100
  return Wire.read();
  #else
  return Wire.receive();
  #endif
}

/**************************************************************************/
/*!
    @brief  Reads a 16 bit values over I2C
*/
/**************************************************************************/
uint16_t Adafruit_TCS34725::read16(uint8_t reg)
{
  uint16_t x; uint16_t t;
  Wire.beginTransmission(TCS34725_ADDRESS);
  #if ARDUINO >= 100
  Wire.write(TCS34725_COMMAND_BIT | reg);
  #else
  Wire.send(TCS34725_COMMAND_BIT | reg);
  #endif
  Wire.endTransmission();

  Wire.requestFrom(TCS34725_ADDRESS, 2);
  #if ARDUINO >= 100
  t = Wire.read();
  x = Wire.read();
  #else
  t = Wire.receive();
  x = Wire.receive();
  #endif
  x <<= 8;
  x |= t;
  return x;
}

/**************************************************************************/
/*!
    Enables the device
*/
/**************************************************************************/
void Adafruit_TCS34725::enable(void)
{
  write8(TCS34725_ENABLE, TCS34725_ENABLE_PON);
  delay(3);
  write8(TCS34725_ENABLE, TCS34725_ENABLE_PON |
TCS34725_ENABLE_AEN);
}

/**************************************************************************/
/*!
    Disables the device (putting it in lower power sleep mode)
*/
/**************************************************************************/
void Adafruit_TCS34725::disable(void)
{
  /* Turn the device off to save power */
  uint8_t reg = 0;
  reg = read8(TCS34725_ENABLE);
  write8(TCS34725_ENABLE, reg & ~(TCS34725_ENABLE_PON |
TCS34725_ENABLE_AEN));
```

```
}

/*=========================================================================*/
/*                            CONSTRUCTORS                               */
/*=========================================================================*/

/**************************************************************************/
/*!
    Constructor
*/
/**************************************************************************/
Adafruit_TCS34725::Adafruit_TCS34725(tcs34725IntegrationTime_t it,
tcs34725Gain_t gain)
{
  _tcs34725Initialised = false;
  _tcs34725IntegrationTime = it;
  _tcs34725Gain = gain;
}

/*=========================================================================*/
/*                           PUBLIC FUNCTIONS                            */
/*=========================================================================*/

/**************************************************************************/
/*!
    Initializes I2C and configures the sensor (call this function before
    doing anything else)
*/
/**************************************************************************/
boolean Adafruit_TCS34725::begin(void)
{
  Wire.begin();

  /* Make sure we're actually connected */
  uint8_t x = read8(TCS34725_ID);
  Serial.println(x, HEX); //(COMMENT THIS LINE IN FULL MUX
IMPLEMENTATION
  if (x != 0x44)
  {
    return false;
  }
  _tcs34725Initialised = true;
  /* Set default integration time and gain */
  setIntegrationTime(_tcs34725IntegrationTime);
  setGain(_tcs34725Gain);

  /* Note: by default, the device is in power down mode on bootup */
  enable();

  return true;
}

/**************************************************************************/
/*!
    Sets the integration time for the TC34725
*/
/**************************************************************************/
void Adafruit_TCS34725::setIntegrationTime(tcs34725IntegrationTime_t it)
{
  if (!_tcs34725Initialised) begin();

  /* Update the timing register */
  write8(TCS34725_ATIME, it);

  /* Update value placeholders */
  _tcs34725IntegrationTime = it;
}

/**************************************************************************/
/*!
    Adjusts the gain on the TCS34725 (adjusts the sensitivity to light)
*/
/**************************************************************************/
void Adafruit_TCS34725::setGain(tcs34725Gain_t gain)
{
  if (!_tcs34725Initialised) begin();

  /* Update the timing register */
  write8(TCS34725_CONTROL, gain);

  /* Update value placeholders */
  _tcs34725Gain = gain;
}

/**************************************************************************/
/*!
    @brief  Reads the raw red, green, blue and clear channel values
```

```
*/
/**************************************************************************/
void Adafruit_TCS34725::getRawData (uint16_t *r, uint16_t *g, uint16_t *b,
uint16_t *c)
{
  if (!_tcs34725Initialised) begin();

  *c = read16(TCS34725_CDATAL);
  *r = read16(TCS34725_RDATAL);
  *g = read16(TCS34725_GDATAL);
  *b = read16(TCS34725_BDATAL);

  /* Set a delay for the integration time */
  switch (_tcs34725IntegrationTime)
  {
    case TCS34725_INTEGRATIONTIME_2_4MS:
      delay(3);
      break;
    case TCS34725_INTEGRATIONTIME_24MS:
      delay(24);
      break;
    case TCS34725_INTEGRATIONTIME_50MS:
      delay(50);
      break;
    case TCS34725_INTEGRATIONTIME_101MS:
      delay(101);
      break;
    case TCS34725_INTEGRATIONTIME_154MS:
      delay(154);
      break;
    case TCS34725_INTEGRATIONTIME_700MS:
      delay(700);
      break;
  }
}

/**************************************************************************/
/*!
    @brief  Converts the raw R/G/B values to color temperature in degrees
            Kelvin
*/
/**************************************************************************/
uint16_t Adafruit_TCS34725::calculateColorTemperature(uint16_t r, uint16_t g,
uint16_t b)
{
  float X, Y, Z;      /* RGB to XYZ correlation      */
  float xc, yc;       /* Chromaticity co-ordinates   */
  float n;            /* McCamy's formula            */
  float cct;

  /* 1. Map RGB values to their XYZ counterparts.    */
  /* Based on 6500K fluorescent, 3000K fluorescent   */
  /* and 60W incandescent values for a wide range.   */
  /* Note: Y = Illuminance or lux                    */
  X = (-0.14282F * r) + (1.54924F * g) + (-0.95641F * b);
  Y = (-0.32466F * r) + (1.57837F * g) + (-0.73191F * b);
  Z = (-0.68202F * r) + (0.77073F * g) + ( 0.56332F * b);

  /* 2. Calculate the chromaticity co-ordinates      */
  xc = (X) / (X + Y + Z);
  yc = (Y) / (X + Y + Z);

  /* 3. Use McCamy's formula to determine the CCT    */
  n = (xc - 0.3320F) / (0.1858F - yc);

  /* Calculate the final CCT */
  cct = (449.0F * powf(n, 3)) + (3525.0F * powf(n, 2)) + (6823.3F * n) + 5520.33F;

  /* Return the results in degrees Kelvin */
  return (uint16_t)cct;
}

/**************************************************************************/
/*!
    @brief  Converts the raw R/G/B values to color temperature in degrees
            Kelvin
*/
/**************************************************************************/
uint16_t Adafruit_TCS34725::calculateLux(uint16_t r, uint16_t g, uint16_t b)
{
  float illuminance;

  /* This only uses RGB ... how can we integrate clear or calculate lux */
  /* based exclusively on clear since this might be more reliable?      */
  illuminance = (-0.32466F * r) + (1.57837F * g) + (-0.73191F * b);

  return (uint16_t)illuminance;
}


void Adafruit_TCS34725::setInterrupt(boolean i) {
  uint8_t r = read8(TCS34725_ENABLE);
  if (i) {
    r |= TCS34725_ENABLE_AIEN;
  } else {
    r &= ~TCS34725_ENABLE_AIEN;
  }
  write8(TCS34725_ENABLE, r);
}

void Adafruit_TCS34725::clearInterrupt(void) {
  Wire.beginTransmission(TCS34725_ADDRESS);
  #if ARDUINO >= 100
  Wire.write(0x66);
  #else
  Wire.send(0x66);
  #endif
  Wire.endTransmission();
}


void Adafruit_TCS34725::setIntLimits(uint16_t low, uint16_t high) {
   write8(0x04, low & 0xFF);
   write8(0x05, low >> 8);
   write8(0x06, high & 0xFF);
   write8(0x07, high >> 8);
}
```

“Adafruit_TCS34725.h” – Header file for TCS34725

```
/**************************************************************************/
/*!
   @file     Adafruit_TCS34725.h
   @author   KTOWN (Adafruit Industries)

   @section LICENSE

   Software License Agreement (BSD License)

   Copyright (c) 2013, Adafruit Industries
   All rights reserved.

   Redistribution and use in source and binary forms, with or without
   modification, are permitted provided that the following conditions are met:
   1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
   2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.
   3. Neither the name of the copyright holders nor the
   names of its contributors may be used to endorse or promote products
   derived from this software without specific prior written permission.

   THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS ''AS IS''
AND ANY
   EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED
   WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE ARE
   DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER BE LIABLE
FOR ANY
   DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES
   (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS OR SERVICES;
   LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND
   ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT
   (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
OF THE USE OF THIS
   SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/
/**************************************************************************/
```

```
#ifndef _TCS34725_H_
#define _TCS34725_H_

#if ARDUINO >= 100
 #include <Arduino.h>
#else
 #include <WProgram.h>
#endif

#include <Wire.h>

#define TCS34725_ADDRESS          (0x29)

#define TCS34725_COMMAND_BIT      (0x80)

#define TCS34725_ENABLE           (0x00)
#define TCS34725_ENABLE_AIEN      (0x10)    /* RGBC Interrupt Enable */
#define TCS34725_ENABLE_WEN       (0x08)    /* Wait enable - Writing 1
activates the wait timer */
#define TCS34725_ENABLE_AEN       (0x02)    /* RGBC Enable - Writing 1
actives the ADC, 0 disables it */
#define TCS34725_ENABLE_PON       (0x01)    /* Power on - Writing 1 activates
the internal oscillator, 0 disables it */
#define TCS34725_ATIME            (0x01)    /* Integration time */
#define TCS34725_WTIME            (0x03)    /* Wait time (if
TCS34725_ENABLE_WEN is asserted) */
#define TCS34725_WTIME_2_4MS      (0xFF)    /* WLONG0 = 2.4ms   WLONG1
= 0.029s */
#define TCS34725_WTIME_204MS      (0xAB)    /* WLONG0 = 204ms
WLONG1 = 2.45s  */
#define TCS34725_WTIME_614MS      (0x00)    /* WLONG0 = 614ms
WLONG1 = 7.4s   */
#define TCS34725_AILTL            (0x04)    /* Clear channel lower interrupt
threshold */
#define TCS34725_AILTH            (0x05)
#define TCS34725_AIHTL            (0x06)    /* Clear channel upper interrupt
threshold */
#define TCS34725_AIHTH            (0x07)
#define TCS34725_PERS             (0x0C)    /* Persistence register - basic SW
filtering mechanism for interrupts */
#define TCS34725_PERS_NONE        (0b0000)  /* Every RGBC cycle generates
an interrupt                              */
#define TCS34725_PERS_1_CYCLE     (0b0001)  /* 1 clean channel value
outside threshold range generates an interrupt   */
#define TCS34725_PERS_2_CYCLE     (0b0010)  /* 2 clean channel values
outside threshold range generates an interrupt  */
```

```
#define TCS34725_PERS_3_CYCLE     (0b0011)  /* 3 clean channel values
outside threshold range generates an interrupt  */
#define TCS34725_PERS_5_CYCLE     (0b0100)  /* 5 clean channel values
outside threshold range generates an interrupt  */
#define TCS34725_PERS_10_CYCLE    (0b0101)  /* 10 clean channel values
outside threshold range generates an interrupt */
#define TCS34725_PERS_15_CYCLE    (0b0110)  /* 15 clean channel values
outside threshold range generates an interrupt */
#define TCS34725_PERS_20_CYCLE    (0b0111)  /* 20 clean channel values
outside threshold range generates an interrupt */
#define TCS34725_PERS_25_CYCLE    (0b1000)  /* 25 clean channel values
outside threshold range generates an interrupt */
#define TCS34725_PERS_30_CYCLE    (0b1001)  /* 30 clean channel values
outside threshold range generates an interrupt */
#define TCS34725_PERS_35_CYCLE    (0b1010)  /* 35 clean channel values
outside threshold range generates an interrupt */
#define TCS34725_PERS_40_CYCLE    (0b1011)  /* 40 clean channel values
outside threshold range generates an interrupt */
#define TCS34725_PERS_45_CYCLE    (0b1100)  /* 45 clean channel values
outside threshold range generates an interrupt */
#define TCS34725_PERS_50_CYCLE    (0b1101)  /* 50 clean channel values
outside threshold range generates an interrupt */
#define TCS34725_PERS_55_CYCLE    (0b1110)  /* 55 clean channel values
outside threshold range generates an interrupt */
#define TCS34725_PERS_60_CYCLE    (0b1111)  /* 60 clean channel values
outside threshold range generates an interrupt */
#define TCS34725_CONFIG          (0x0D)
#define TCS34725_CONFIG_WLONG     (0x02)    /* Choose between short and
long (12x) wait times via TCS34725_WTIME */
#define TCS34725_CONTROL          (0x0F)    /* Set the gain level for the sensor
*/
#define TCS34725_ID              (0x12)    /* 0x44 = TCS34721/TCS34725, 0x4D =
TCS34723/TCS34727 */
#define TCS34725_STATUS          (0x13)
#define TCS34725_STATUS_AINT      (0x10)    /* RGBC Clean channel interrupt
*/
#define TCS34725_STATUS_AVALID    (0x01)    /* Indicates that the RGBC
channels have completed an integration cycle */
#define TCS34725_CDATAL          (0x14)    /* Clear channel data */
#define TCS34725_CDATAH          (0x15)
#define TCS34725_RDATAL          (0x16)    /* Red channel data */
#define TCS34725_RDATAH          (0x17)
#define TCS34725_GDATAL          (0x18)    /* Green channel data */
#define TCS34725_GDATAH          (0x19)
#define TCS34725_BDATAL          (0x1A)    /* Blue channel data */
#define TCS34725_BDATAH          (0x1B)
```

```
typedef enum
{
  TCS34725_INTEGRATIONTIME_2_4MS  = 0xFF,   /**<  2.4ms - 1 cycle    - 
Max Count: 1024  */
  TCS34725_INTEGRATIONTIME_24MS   = 0xF6,   /**<  24ms  - 10 cycles  - 
Max Count: 10240 */
  TCS34725_INTEGRATIONTIME_50MS   = 0xEB,   /**<  50ms  - 20 cycles  - 
Max Count: 20480 */
  TCS34725_INTEGRATIONTIME_101MS  = 0xD5,   /**<  101ms - 42 cycles  - 
Max Count: 43008 */
  TCS34725_INTEGRATIONTIME_154MS  = 0xC0,   /**<  154ms - 64 cycles  - 
Max Count: 65535 */
  TCS34725_INTEGRATIONTIME_700MS  = 0x00    /**<  700ms - 256 cycles - 
Max Count: 65535 */
}
tcs34725IntegrationTime_t;

typedef enum
{
  TCS34725_GAIN_1X                = 0x00,   /**<  No gain  */
  TCS34725_GAIN_4X                = 0x01,   /**<  2x gain  */
  TCS34725_GAIN_16X               = 0x02,   /**<  16x gain */
  TCS34725_GAIN_60X               = 0x03    /**<  60x gain */
}
tcs34725Gain_t;

class Adafruit_TCS34725 {
 public:
  Adafruit_TCS34725(tcs34725IntegrationTime_t = 
TCS34725_INTEGRATIONTIME_2_4MS, tcs34725Gain_t = 
TCS34725_GAIN_1X);

  boolean  begin(void);
  void     setIntegrationTime(tcs34725IntegrationTime_t it);
  void     setGain(tcs34725Gain_t gain);
  void     getRawData(uint16_t *r, uint16_t *g, uint16_t *b, uint16_t *c);
  uint16_t calculateColorTemperature(uint16_t r, uint16_t g, uint16_t b);
  uint16_t calculateLux(uint16_t r, uint16_t g, uint16_t b);
  void     write8 (uint8_t reg, uint32_t value);
  uint8_t  read8 (uint8_t reg);
  uint16_t read16 (uint8_t reg);
  void setInterrupt(boolean flag);
  void clearInterrupt(void);
  void setIntLimits(uint16_t l, uint16_t h);
  void     enable(void);
 private:
  boolean _tcs34725Initialised;
  tcs34725Gain_t _tcs34725Gain;
  tcs34725IntegrationTime_t _tcs34725IntegrationTime;

  void     disable(void);
};

#endif
```

“colorview.ino” – This is the updated version; it outputs just red RGB value multiplied by 100

```
#include <Wire.h>
#include "Adafruit_TCS34725.h"

// Pick analog outputs, for the UNO these three work well
// use ~560  ohm resistor between Red & Blue, ~1K for green (its brighter)
#define redpin 3
#define greenpin 5
#define bluepin 6
// for a common anode LED, connect the common pin to +5V
// for common cathode, connect the common to ground

// set to false if using a common cathode LED
#define commonAnode true

// our RGB -> eye-recognized gamma color
byte gammatable[256];


Adafruit_TCS34725 tcs =
Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_50MS,
TCS34725_GAIN_4X);

void setup() {
  Serial.begin(9600);
  //Serial.println("Color View Test!");

  if (tcs.begin()) {
    //Serial.println("Found sensor");
  } else {
    Serial.println("No TCS34725 found ... check your connections");
    while (1); // halt!
  }

  // use these three pins to drive an LED
  pinMode(redpin, OUTPUT);
  pinMode(greenpin, OUTPUT);
  pinMode(bluepin, OUTPUT);

  // thanks PhilB for this gamma table!
  // it helps convert RGB colors to what humans see
  for (int i=0; i<256; i++) {
    float x = i;
    x /= 255;
    x = pow(x, 2.5);
    x *= 255;

    if (commonAnode) {
      gammatable[i] = 255 - x;
    } else {
      gammatable[i] = x;
    }
    //Serial.println(gammatable[i]);
  }
}


void loop() {
  uint16_t clear, red, green, blue;

  tcs.setInterrupt(false);      // turn on LED

  delay(60);  // takes 50ms to read

  tcs.getRawData(&red, &green, &blue, &clear);

  tcs.setInterrupt(true);  // turn off LED

  //Serial.print("C:\t"); Serial.print(clear);
 // Serial.print("\tR:\t");

 //ADI GENERATED
 double redflt=red;
 double clearflt=clear;
 double redreal=redflt/clearflt*25500;
 Serial.print(redreal);
 //Serial.print(red);
 //Serial.println();
 //Serial.print(clear);

 // Serial.print("\tG:\t"); Serial.print(green);
 // Serial.print("\tB:\t"); Serial.print(blue);

  // Figure out some basic hex code for visualization
  uint32_t sum = clear;
  float r, g, b;
  r = red; r /= sum;
  g = green; g /= sum;
  b = blue; b /= sum;
  r *= 256; g *= 256; b *= 256;
  //Serial.print("\t");
  //Serial.print((int)r, HEX); Serial.print((int)g, HEX); Serial.print((int)b, HEX);

  //PRINTS A NEW LINE AT END
  Serial.println();

  //Serial.print((int)r ); Serial.print(" "); Serial.print((int)g);Serial.print(" ");
Serial.println((int)b );

  analogWrite(redpin, gammatable[(int)r]);
  analogWrite(greenpin, gammatable[(int)g]);
  analogWrite(bluepin, gammatable[(int)b]);
}
```

Python code for one TCS34725 color sensor

```
# -*- coding: utf-8 -*-
"""
Created on Mon Apr 13 16:45:26 2015

@author: adityasanghani
"""

import serial
import matplotlib.pyplot as plt
ser = serial.Serial('/dev/tty.usbmodem1411', 9600)
i=1
redarray=range(100)

while i<101:

a=ser.readline()
print a
b=float(a)
print b

redarray[i-1]=b/100

i=i+1


print redarray
plt.plot(redarray)
plt.ylim(110,180)
```

“colorviewgoodtca9544a.ino” – Code for full multiplexer implementation

```
#include <Wire.h>
#include "Adafruit_TCS34725.h"

//initialize color sensors objects
Adafruit_TCS34725 tcs[2] =
Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_50MS,
TCS34725_GAIN_4X);

void setup() {

  byte ok = 1;

  Serial.begin(9600);
  Wire.begin();
  delay(10);

  //initialize all the sensors
  for(int i = 0; i<2;i++)
  {
    //switch i2c multiplexor
    Wire.beginTransmission(0x70);
    Wire.write(i+4);
    Wire.endTransmission();

    delay(100);

    //attempt to connect the sensor
    if (tcs[i].begin())
    {
       //Sensor is connected!
    } else {
      //something went wrong
      Serial.println("No TCS34725 found at " + (String)i + " ... check your
connections");
      ok = 0;
    }

  }

  if(!ok)
  { //sensor not connected, halt the program
    Serial.println("Halting!");
    while (1); // halt!
  }
}


void loop() {
  //loop through the color sensors
  for(int i = 0; i<2;i++)
  {
    //Set the i2c switch to the desired output
    Wire.beginTransmission(0x70);
    Wire.write(i+4);
    Wire.endTransmission();
    //wait for the switch to switch
    delay(1);
    uint16_t clear, red, green, blue;
    tcs[i].setInterrupt(false);      // turn on LED
    delay(50);  // takes 50ms to read

    tcs[i].getRawData(&red, &green, &blue, &clear);

    tcs[i].setInterrupt(true);  // turn off LED

    double redflt=red;
    double clearflt=clear;
    double redreal=(redflt/clearflt)*25500;
    Serial.print(redreal);
    //

    if(i != 1)
    {
      //Serial.print(";");
      Serial.print("\n");
    }
    else
    {
      Serial.print("\n");
    }
  }
}
```

Python code for full multiplexer implementation

```
# -*- coding: utf-8 -*-
"""
Created on Mon Apr 13 16:45:26 2015

@author: adityasanghani
"""

import serial
import matplotlib.pyplot as plt

import time
ser = serial.Serial('/dev/tty.usbmodem1411', 9600)
i=1
redarray=range(50)
redarray2=range(50)

itertime=range(50)
itertime2=range(50)

while i<101:

a=ser.readline()
print a
b=float(a)
print b

if i%2==1:
redarray[(i-1)/2]=b/100
itertime[(i-1)/2]=int(round(time.time() * 1000))
if i%2==0:
redarray2[i/2-1]=b/100
itertime2[i/2-1]=int(round(time.time() * 1000))

i=i+1


print redarray
print redarray2
print itertime
print itertime2

plt.plot(redarray)
plt.plot(redarray2)
plt.ylim(115,185)
```