Renovation of Spectronic-20 by Integrating a Raspberry Pi Computing System

Tyler A. McGaughey '18 and C. William Anderson

Department of Chemistry, Hampden-Sydney College, Hampden-Sydney, VA 23943

INTRODUCTION

The purpose of this project was to renovate a Spectronic 20. The Spectronic-20 when it originally came out in the 1950's was a revolutionary instrument. It opened many doors because of its low cost. Over the course of production approximately 600,000 were made. Due to the low cost and relatively large production numbers many high schools and college were able to acquire one. However, as technologies advanced they fell by the wayside. The Spectronic 20 relies primarily on vacuum tubes, which can take up to half an hour to warm up. This caused them to be put in storage closets and forgot about. The instrument also takes a lot of skill to use. This project's goal was to turn a functioning antiquated instrument into a user friendly. modern instrument.

HARDWARE

The first portion of this project focused on the hardware or the actual instrument its self. The Spectronic 20 is a spectrophotometer. Which means it uses light to analyze a sample. The optics inside the instrument are still viable. Figure 1 explains the light's path through the instrument.

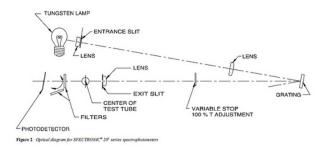


Figure 1 - Optics Diagram²

Light is first emitted from the light source. It then passes through a lens for focusing then a prism for grating. The technician has a dial to adjust the wavelength of light that goes to the sample. This is useful because samples absorb different wavelengths of light based on color. After the light passes through the sample it hits the photo detector. The signal then goes from the photo detector to the electronics, which translate it into a percent absorbance. That information is then displayed on a needle display. Figure 2 shows the current exterior of the instrument.



Figure 2 – Exterior

First the instrument was completely eviscerated except the optics. Figure 3 shows the internal components prior to overhaul.



Figure 3 - Internal Mechanisms

After overhaul, new components were put into the shell of the instrument starting with the photo detector. It is responsible for quantifying the photons of photons its sees and transferring that information to a computing system. Figure 4 shows the old photo detector.



Figure 4 – Old Photo Detector

This project utilized the TSL2591 high dynamic range digital light sensor. This sensor is rated for both infrared and visible light spectra. The sensor is rated for light from 420nm to 1050nm.³ This sensor is perfect for the project for a number of reasons. The first is its low cost, under ten dollars.⁴ Second are its wide spectra abilities. Lastly, is the easy interface via i2c with the Raspberry Pi. Figure 5 shows the TSL2591 sensor.



Figure 5 - TSL2591 Sensor '

This sensor was mounted in the light beam path behind the sample and connected with wires to the general-purpose input/output (GPIO) pins on the Raspberry Pi.

A Raspberry Pi 3 was utilized in this project for a number of reasons. The first is the low cost, under forty dollars. 5 Keeping in line with the goal of the project to get these instruments back into the classroom easily and most importantly at a low cost. The next reason is the great versatility of the system. There are endless programing opportunities since the Pi comes from the manufacturer blank. It is up to the consumer to write and program an operating system or code. The third reason is the number of input types. The Raspberry Pi 3 comes with built in Bluetooth, Wi-Fi, HDMI port, four USB ports, Ethernet port, and forty GPIO. It also comes with a 1.2 GHz Quad Core Broadcom BCM2837 64-bit ARMv8 processor giving it a respectable amount of processing power.5



Figure 6 – Raspberry Pi 3 5

The aforementioned features of the Raspberry Pi allow for easy use of the instrument. The Wi-Fi capabilities provide the technician operating the renovated instrument with a number of new capabilities, for instance the ability to email data to a professor or wireless interfacing for diagnostics. The Bluetooth capabilities allow for the use of a wireless mouse and keyboard. Lastly, the HDMI port allows for the incorporation of a LCD (liquid crystal display). After the hardware was in place the software was developed.

SOFTWARE

As stated earlier all Raspberry Pi's come from the manufacturer blank. It is up to the user install their own operating system. The Raspberry Pi foundation has several open source operating systems designed specifically for the Raspberry Pi. The most commonly used one is called Raspbian. This software is downloaded onto a micro-SD card and then loaded into the slot on the Raspberry Pi. Raspbian provides a simple desktop and a terminal for python coding. All coding for this project was done using the python language.

The initial python code to run the TSL2591 sensor was derived from an open source repository. That code is listed in the code appendices. That code was then transformed and reworked to make it more user-friendly. The original script only outputs one lux value. This project developed a code that is much easier for the technician to use. The python script opens a blank document then creates a header consisting of Hampden Sydney College, the current time, and the wavelength of the current test in progress. The script then prompts the researcher to remove all cuvettes so it can take a background light reading. It then prompts the researcher to insert a cuvette filled with deionized water. This allows the

instrument to take a max light reading. The instrument then prompts the researcher to insert the cuvette with the sample. It then outputs three light values in Lux. See the second entry in the code appendices for the full code. After both the hardware and software were in place it was time to test the newly renovated instrument.

TESTING

As stated earlier the Spectronic 20 is a spectrophotometer. This means that it analyzes samples based on absorbed light. In order to test the instrument a dilute sample of Nickel Acetate was prepared. The solution was then run through a UV-Vis spectrometer. This produced an absorption spectra. The solution was then tested on the instrument. Three values were gathered at every wavelength; a background light value, a max light value and a sample light value. The background light value was obtained by measuring the amount of light present with no cuvette in the instrument. A max light value was then obtained by inserting a cuvette filled with deionized water. Then the sample light value was obtained by inserting a cuvette filled with the sample liquid. Then using Beer's law the percent of light absorbed or percent transmittance was calculated. Figure 7 shows the graph comparing the absorption spectra of the renovated Spectronic 20 versus the UV-Vis spectrometer.

Absoption Spectra of Ni Acetate of UV Spectrophotometer Versus a Spectronic 20 with a TSL2591 Detector and a Raspberry Pi

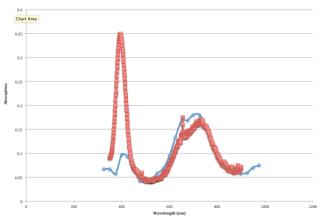


Figure 7 - Spec 20 V.S. UV-Vis Spectrometer

The blue line shows the values for the renovated Spectronic 20 while the orange shows the UV-Vis. The peak around 400nm is not as distinct because the TSL2591 sensor is not rated below 420nm. However, the renovated Spectronic 20 provides comparable results for a fraction of the cost of the UV-Vis Spectrometer.

FUTURE IMPLICATIONS

Like any project further research and work could be done to improve this project. These efforts could be focused on developing a better graphical interface and producing a series of instructional how to videos.

CONCLUSIONS

Overall the project was successful. A Spectronic 20 was gutted and renovated with modern electronics. These electronics allow for easier use of the instruments. Also the entire project was done for two to three hundred dollars. Therefore, this is a viable option to get these instruments back into the classroom nationwide.

ACKNOWLEDGEMENTS

I would like to personally thank Dr. Anderson for his supervision and advice for the duration of the project. I would also like to thank Dr. Sipe, Dr. Deifel, and Mrs. Hines for their assistance and support.

REFERENCES

- 1) https://en.wikipedia.org/wiki/Spectronic_20
- Thermo Spectronic. (2001). Operators manual for Spectronic 20 Series Spectrophotometers, NY: Thermo Electronic.
 - http://www.cienytec.com/PDFS/Espec_SPEC20_ OpMan_ing.pdf
- AMS AG. (2013). TSL2591 Datasheet, Austria. https://www.adafruit.com/datasheets/TSL25911_D atasheet_EN_v1.pdf
- 4) https://www.adafruit.com/product/1980
- 5) https://www.adafruit.com/products/3055
- 6) https://www.raspberrypi.org/downloads/raspbian/
- 7) https://github.com/maxlklaxl/python-tsl2591/blob/master/tsl2591/read_tsl.py last update on 3/6/2106

CODE APPENDICES

Entry 1: original source code from https://github.com/maxlklaxl/python-tsl2591/blob/master/tsl2591/read_tsl.py

This code is basically an adaptation of the Arduino_TSL2591 library from adafruit:

https://github.com/adafruit/Adafruit_TSL2591_Library

for configuring I2C in a raspberry https://learn.adafruit.com/adafruits-raspberry-pilesson-4-gpio-setup/configuring-i2c

```
GAIN MAX = 0x30 \# max gain (9876x)
datasheet:
http://ams.com/eng/Products/Light-Sensors/Light-to-
Digital-Sensors/TSL25911
                                                     class Tsl2591(object):
                                                        def __init__(
                                                               self,
                                                               i2c_bus=1,
import smbus
                                                               sensor address=0x29,
import time
                                                               integration=INTEGRATIONTIME 100MS,
VISIBLE = 2 # channel 0 - channel 1
                                                               gain=GAIN LOW
INFRARED = 1 # channel 1
                                                               ):
FULLSPECTRUM = 0 # channel 0
                                                          self.bus = smbus.SMBus(i2c bus)
                                                          self.sendor address = sensor address
ADDR = 0x29
                                                          self.integration time = integration
READBIT = 0x01
                                                          self.gain = gain
COMMAND BIT = 0xA0 # bits 7 and 5 for 'command
                                                          self.set_timing(self.integration time)
                                                          self.set gain(self.gain)
CLEAR BIT = 0x40 # Clears any pending interrupt
                                                          self.disable() # to be sure
(write 1 to clear)
WORD BIT = 0x20 \# 1 = read/write word (rather)
                                                        def set timing(self, integration):
than byte)
                                                          self.enable()
BLOCK_BIT = 0x10 # 1 = using block read/write
                                                          self.integration time = integration
ENABLE POWERON = 0x01
                                                          self.bus.write byte data(
ENABLE_POWEROFF = 0x00
                                                                 self.sendor address,
ENABLE AEN = 0x02
                                                                 COMMAND BIT |
ENABLE AIEN = 0x10
                                                     REGISTER CONTROL,
CONTROL RESET = 0x80
                                                                 self.integration_time | self.gain
LUX_DF = 408.0
LUX COEFB = 1.64 # CH0 coefficient
                                                          self.disable()
LUX COEFC = 0.59 # CH1 coefficient A
                                                        def get timing(self):
LUX COEFD = 0.86 # CH2 coefficient B
                                                          return self.integration time
REGISTER ENABLE = 0x00
REGISTER_CONTROL = 0x01
                                                        def set_gain(self, gain):
REGISTER THRESHHOLDL LOW = 0x02
                                                          self.enable()
REGISTER THRESHHOLDL HIGH = 0x03
                                                          self.gain = gain
REGISTER_THRESHHOLDH_LOW = 0x04
                                                          self.bus.write_byte_data(
REGISTER THRESHHOLDH HIGH = 0x05
                                                                 self.sendor address,
REGISTER_INTERRUPT = 0x06
                                                                 COMMAND BIT |
                                                     REGISTER CONTROL,
REGISTER CRC = 0x08
REGISTER ID = 0x0A
                                                                 self.integration time | self.gain
REGISTER CHAN0 LOW = 0x14
REGISTER CHAN0 HIGH = 0x15
                                                          self.disable()
REGISTER CHAN1 LOW = 0x16
REGISTER_CHAN1_HIGH = 0x17
                                                        def get gain(self):
INTEGRATIONTIME_100MS = 0x00
                                                          return self.gain
INTEGRATIONTIME 200MS = 0x01
INTEGRATIONTIME 300MS = 0x02
                                                        def calculate lux(self, full, ir):
INTEGRATIONTIME 400MS = 0x03
                                                          # Check for overflow conditions first
INTEGRATIONTIME 500MS = 0x04
                                                          if (full == 0xFFFF) | (ir == 0xFFFF):
INTEGRATIONTIME 600MS = 0x05
                                                            return 0
GAIN LOW = 0x00 \# low gain (1x)
                                                          case integ = {
GAIN MED = 0x10 \# medium gain (25x)
                                                            INTEGRATIONTIME 100MS: 100.,
GAIN HIGH = 0x20 \# medium gain (428x)
                                                            INTEGRATIONTIME 200MS: 200.,
```

```
INTEGRATIONTIME_300MS: 300.,
       INTEGRATIONTIME_400MS: 400.,
                                                              self.disable()
                                                              return full, ir
       INTEGRATIONTIME 500MS: 500.,
       INTEGRATIONTIME 600MS: 600.,
                                                            def get luminosity(self, channel):
                                                              full, ir = self.get full luminosity()
    if self.integration time in case integ.keys():
       atime = case_integ[self.integration_time]
                                                              if channel == FULLSPECTRUM:
                                                                # Reads two byte value from channel 0
    else:
                                                         (visible + infrared)
       atime = 100.
                                                                return full
                                                              elif channel == INFRARED:
    case gain = {
       GAIN LOW: 1.,
                                                                # Reads two byte value from channel 1
                                                         (infrared)
       GAIN_MED: 25.,
       GAIN HIGH: 428.,
                                                                return ir
       GAIN MAX: 9876.,
                                                              elif channel == VISIBLE:
      }
                                                                # Reads all and subtracts out ir to give just the
                                                         visible!
    if self.gain in case gain.keys():
                                                                return full - ir
       again = case gain[self.gain]
                                                              else: # unknown channel!
    else:
                                                                return 0
       again = 1.
    # cpl = (ATIME * AGAIN) / DF
                                                         if name == ' main ':
    cpl = (atime * again) / LUX DF
    lux1 = (full - (LUX_COEFB * ir)) / cpl
                                                           tsl = Tsl2591() # initialize
                                                            full, ir = tsl.get full luminosity() # read raw values
    lux2 = ((LUX_COEFC * full) - (LUX_COEFD * ir))
                                                         (full spectrum and ir spectrum)
                                                            lux = tsl.calculate_lux(full, ir) # convert raw values
/ cpl
                                                         to lux
    # The highest value is the approximate lux
                                                            print (lux, full, ir)
equivalent
                                                            print ()
    return max([lux1, lux2])
                                                            def test(int_time=INTEGRATIONTIME_100MS,
                                                         gain=GAIN LOW):
  def enable(self):
    self.bus.write_byte_data(
                                                              tsl.set gain(gain)
            self.sendor address,
                                                              tsl.set timing(int time)
            COMMAND BIT | REGISTER ENABLE,
                                                              full test, ir test = tsl.get full luminosity()
            ENABLE POWERON | ENABLE AEN |
                                                              lux test = tsl.calculate lux(full test, ir test)
                                                              print ('Lux = %f full = %i ir = %i' % (lux_test,
ENABLE AIEN
            ) # Enable
                                                         full test, ir test))
                                                              print("integration time = %i" % tsl.get_timing())
                                                              print("gain = %i \n" % tsl.get gain())
  def disable(self):
    self.bus.write byte data(
                                                            for i in [INTEGRATIONTIME 100MS,
            self.sendor address,
            COMMAND_BIT | REGISTER_ENABLE,
                                                                  INTEGRATIONTIME 200MS.
            ENABLE POWEROFF
                                                                  INTEGRATIONTIME_300MS,
                                                                  INTEGRATIONTIME_400MS,
            )
                                                                  INTEGRATIONTIME 500MS,
  def get_full_luminosity(self):
                                                                  INTEGRATIONTIME_600MS]:
    self.enable()
                                                              test(i, GAIN_LOW)
    time.sleep(0.120*self.integration time+1) # not
sure if we need it "// Wait x ms for ADC to complete"
                                                            for i in [GAIN LOW,
    full = self.bus.read word data(
                                                                  GAIN MED,
            self.sendor address, COMMAND BIT I
                                                                  GAIN HIGH,
REGISTER CHAN0 LOW
                                                                  GAIN MAX1:
                                                              test(INTEGRATIONTIME 100MS, i)
                                                         Entry 2: This projects python code
    ir = self.bus.read word data(
            self.sendor_address, COMMAND_BIT |
                                                         print "Hampden Sydney College"
REGISTER_CHAN1_LOW
                                                         import datetime
```

```
REGISTER CRC = 0x08
now = datetime.datetime.now()
print now.strftime("%Y-%m-%d %H:%M")
                                                    REGISTER ID = 0x0A
                                                    REGISTER CHAN0 LOW = 0x14
fob=open('/home/pi/data.txt','a')
                                                    REGISTER_CHAN0_HIGH = 0x15
fob.write(" \n")
                                                    REGISTER_CHAN1_LOW = 0x16
fob.write(" \n")
                                                    REGISTER_CHAN1_HIGH = 0x17
fob.write('Hampden Sydney College\n')
                                                    INTEGRATIONTIME 100MS = 0x00
fob.write(now.strftime("%Y-%m-%d %H:%M\n"))
                                                    INTEGRATIONTIME 200MS = 0x01
                                                    INTEGRATIONTIME_300MS = 0x02
                                                    INTEGRATIONTIME 400MS = 0x03
wavelength = raw input("What wavelength is this test
preformed at? ")
                                                    INTEGRATIONTIME_500MS = 0x04
print "Wavelength = %r" % (wavelength)
                                                    INTEGRATIONTIME 600MS = 0x05
fob.write("Wavelength = %r" % (wavelength))
fob.write(" \n")
                                                    GAIN HIGH = 0x20 \# medium gain (428x)
fob.close()
                                                    class Tsl2591(object):
                                                       def init (
self,
##########
                                                              i2c bus=1,
blank = raw input("Remove cuvette to run
                                                              sensor address=0x29,
background, press ENTER to continue.")
                                                              integration=INTEGRATIONTIME 100MS,
print 'Measuring Background Readings, Please wait.'
                                                              gain=GAIN HIGH
import smbus
                                                              ):
                                                         self.bus = smbus.SMBus(i2c bus)
import time
                                                         self.sendor address = sensor address
VISIBLE = 2 # channel 0 - channel 1
                                                         self.integration time = integration
                                                         self.gain = gain
INFRARED = 1 # channel 1
FULLSPECTRUM = 0 # channel 0
                                                         self.set_timing(self.integration_time)
                                                         self.set gain(self.gain)
ADDR = 0x29
                                                         self.disable() # to be sure
READBIT = 0x01
COMMAND BIT = 0xA0 # bits 7 and 5 for 'command
                                                       def set timing(self, integration):
                                                         self.enable()
CLEAR BIT = 0x40 # Clears any pending interrupt
                                                         self.integration_time = integration
(write 1 to clear)
                                                         self.bus.write byte data(
WORD BIT = 0x20 \# 1 = read/write word (rather)
                                                                self.sendor address,
                                                                COMMAND_BIT |
than byte)
BLOCK BIT = 0x10 # 1 = using block read/write
                                                    REGISTER CONTROL,
ENABLE POWERON = 0x01
                                                                self.integration_time | self.gain
ENABLE POWEROFF = 0x00
                                                         self.disable()
ENABLE AEN = 0x02
ENABLE AIEN = 0x10
CONTROL RESET = 0x80
                                                       def get timing(self):
LUX DF = 408.0
                                                         return self.integration time
LUX COEFB = 1.64 # CH0 coefficient
LUX COEFC = 0.59 # CH1 coefficient A
                                                       def set_gain(self, gain):
LUX COEFD = 0.86 # CH2 coefficient B
                                                         self.enable()
                                                         self.gain = gain
REGISTER_ENABLE = 0x00
                                                         self.bus.write_byte_data(
REGISTER CONTROL = 0x01
                                                                self.sendor address,
REGISTER THRESHHOLDL LOW = 0x02
                                                                COMMAND BIT |
REGISTER THRESHHOLDL HIGH = 0x03
                                                    REGISTER CONTROL,
REGISTER THRESHHOLDH LOW = 0x04
                                                                self.integration time | self.gain
REGISTER THRESHHOLDH HIGH = 0x05
REGISTER INTERRUPT = 0x06
                                                         self.disable()
```

```
full = self.bus.read_word_data(
                                                                      self.sendor address, COMMAND BIT |
  def get gain(self):
                                                          REGISTER CHAN0 LOW
    return self.gain
  def calculate lux(self, full, ir):
                                                              ir = self.bus.read_word_data(
                                                                      self.sendor address, COMMAND BIT |
    # Check for overflow conditions first
    if (full == 0xFFFF) | (ir == 0xFFFF):
                                                          REGISTER_CHAN1_LOW
       return 0
                                                              self.disable()
                                                              return full, ir
    case integ = {
       INTEGRATIONTIME_100MS: 100.,
       INTEGRATIONTIME_200MS: 200.,
                                                            def get luminosity(self, channel):
       INTEGRATIONTIME_300MS: 300.,
                                                              full, ir = self.get_full_luminosity()
       INTEGRATIONTIME 400MS: 400.,
                                                              if channel == FULLSPECTRUM:
       INTEGRATIONTIME 500MS: 500.,
                                                                 # Reads two byte value from channel 0
       INTEGRATIONTIME_600MS: 600.,
                                                          (visible + infrared)
                                                                 return full
    if self.integration time in case integ.keys():
                                                              elif channel == INFRARED:
       atime = case integ[self.integration time]
                                                                 # Reads two byte value from channel 1
    else:
                                                          (infrared)
       atime = 100.
                                                                 return ir
                                                              elif channel == VISIBLE:
    case gain = {GAIN HIGH: 428.}
                                                                 # Reads all and subtracts out ir to give just the
                                                          visible!
    if self.gain in case_gain.keys():
                                                                 return full - ir
       again = case gain[self.gain]
                                                              else: # unknown channel!
    else:
                                                                 return 0
       again = 1.
                                                         if _name__ == '__main__':
    # cpl = (ATIME * AGAIN) / DF
    cpl = (atime * again) / LUX DF
                                                            tsl = Tsl2591() # initialize
    lux1 = (full - (LUX COEFB * ir)) / cpl
                                                            full, ir = tsl.get full luminosity() # read raw values
                                                          (full spectrum and ir spectrum)
    lux2 = ((LUX COEFC * full) - (LUX COEFD * ir))
                                                            lux = tsl.calculate lux(full, ir) # convert raw values
lgo \
                                                          to lux
                                                            def test(int_time=INTEGRATIONTIME_100MS,
    # The highest value is the approximate lux
                                                          gain=GAIN HIGH):
equivalent
                                                              tsl.set gain(gain)
    return max([lux1, lux2])
                                                              tsl.set timing(int time)
  def enable(self):
                                                              full_test, ir_test = tsl.get_full_luminosity()
                                                              lux test = tsl.calculate lux(full test, ir test)
    self.bus.write byte data(
                                                              fob=open('/home/pi/data.txt','a')
            self.sendor address.
            COMMAND_BIT | REGISTER_ENABLE,
                                                                 fob.write(" \n")
                                                                 fob.write('Background Reading\n')
            ENABLE POWERON | ENABLE AEN |
ENABLE AIEN
                                                                 fob.write('full = %i\n' % (full test))
            ) # Enable
                                                                 fob.write("integration time = %i\n" %
                                                          tsl.get timing())
  def disable(self):
                                                                 fob.write("gain = %i \n" % tsl.get_gain())
    self.bus.write byte data(
                                                                 fob.close()
                                                                 print ('full = %i' % (full test))
            self.sendor address,
            COMMAND BIT | REGISTER ENABLE,
                                                              print("integration time = %i" % tsl.get timing())
            ENABLE POWEROFF
                                                              print("gain = %i \n" % tsl.get_gain())
                                                            for i in [INTEGRATIONTIME 200MS]:
  def get full luminosity(self):
                                                              test(i, GAIN_HIGH)
    self.enable()
    time.sleep(0.120*self.integration_time+1) # not
                                                          sure if we need it "// Wait x ms for ADC to complete"
                                                          ###
```

```
upper = raw input("Insert cuvette with DiH2O to
                                                               i2c bus=1,
establish max light value, press ENTER to continue.")
                                                               sensor address=0x29.
print 'Measuring Upper Bound Readings, Please
                                                               integration=INTEGRATIONTIME_100MS,
                                                               gain=GAIN_HIGH
wait.'
import smbus
                                                               ):
import time
                                                          self.bus = smbus.SMBus(i2c_bus)
                                                          self.sendor address = sensor address
VISIBLE = 2 # channel 0 - channel 1
                                                          self.integration time = integration
INFRARED = 1 # channel 1
                                                          self.gain = gain
FULLSPECTRUM = 0 # channel 0
                                                          self.set timing(self.integration time)
                                                          self.set gain(self.gain)
ADDR = 0x29
                                                          self.disable() # to be sure
READBIT = 0x01
COMMAND BIT = 0xA0 # bits 7 and 5 for 'command
                                                        def set timing(self, integration):
                                                          self.enable()
CLEAR BIT = 0x40 # Clears any pending interrupt
                                                          self.integration time = integration
(write 1 to clear)
                                                          self.bus.write byte data(
WORD BIT = 0x20 # 1 = read/write word (rather
                                                                 self.sendor address,
than byte)
                                                                 COMMAND BIT |
BLOCK BIT = 0x10 # 1 = using block read/write
                                                     REGISTER CONTROL,
ENABLE POWERON = 0x01
                                                                 self.integration time | self.gain
ENABLE_POWEROFF = 0x00
                                                          self.disable()
ENABLE AEN = 0x02
ENABLE AIEN = 0x10
CONTROL RESET = 0x80
                                                        def get timing(self):
LUX DF = 408.0
                                                          return self.integration time
LUX COEFB = 1.64 # CH0 coefficient
LUX_COEFC = 0.59 # CH1 coefficient A
                                                        def set_gain(self, gain):
LUX COEFD = 0.86 # CH2 coefficient B
                                                          self.enable()
                                                          self.gain = gain
REGISTER ENABLE = 0x00
                                                          self.bus.write byte data(
REGISTER CONTROL = 0x01
                                                                 self.sendor address,
REGISTER_THRESHHOLDL_LOW = 0x02
                                                                 COMMAND BIT I
REGISTER_THRESHHOLDL_HIGH = 0x03
                                                     REGISTER_CONTROL,
REGISTER THRESHHOLDH LOW = 0x04
                                                                 self.integration time | self.gain
REGISTER THRESHHOLDH HIGH = 0x05
REGISTER_INTERRUPT = 0x06
                                                          self.disable()
REGISTER CRC = 0x08
REGISTER_ID = 0x0A
                                                        def get_gain(self):
REGISTER CHAN0 LOW = 0x14
                                                          return self.gain
REGISTER CHANO HIGH = 0x15
REGISTER CHAN1 LOW = 0x16
                                                        def calculate lux(self, full, ir):
REGISTER CHAN1 HIGH = 0x17
                                                          # Check for overflow conditions first
INTEGRATIONTIME_100MS = 0x00
                                                          if (full == 0xFFFF) | (ir == 0xFFFF):
INTEGRATIONTIME_200MS = 0x01
                                                            return 0
INTEGRATIONTIME_300MS = 0x02
INTEGRATIONTIME 400MS = 0x03
                                                          case integ = {
INTEGRATIONTIME 500MS = 0x04
                                                            INTEGRATIONTIME_100MS: 100.,
INTEGRATIONTIME_600MS = 0x05
                                                            INTEGRATIONTIME_200MS: 200.,
                                                            INTEGRATIONTIME_300MS: 300.,
GAIN HIGH = 0x20 # medium gain (428x)
                                                            INTEGRATIONTIME 400MS: 400.,
                                                            INTEGRATIONTIME 500MS: 500.,
class Tsl2591(object):
                                                            INTEGRATIONTIME 600MS: 600...
  def __init__(
                                                          if self.integration time in case integ.keys():
         self.
```

```
atime = case_integ[self.integration_time]
                                                                # Reads two byte value from channel 1
                                                         (infrared)
       atime = 100.
                                                                return ir
                                                              elif channel == VISIBLE:
                                                                # Reads all and subtracts out ir to give just the
    case_gain = {GAIN_HIGH: 428.}
                                                         visible!
    if self.gain in case_gain.keys():
                                                                return full - ir
       again = case_gain[self.gain]
                                                              else: # unknown channel!
    else:
                                                                return 0
       again = 1.
                                                         if __name__ == '__main__':
    # cpl = (ATIME * AGAIN) / DF
    cpl = (atime * again) / LUX_DF
                                                            tsl = Tsl2591() # initialize
    lux1 = (full - (LUX COEFB * ir)) / cpl
                                                            full, ir = tsl.get full luminosity() # read raw values
                                                         (full spectrum and ir spectrum)
    lux2 = ((LUX_COEFC * full) - (LUX_COEFD * ir))
                                                            lux = tsl.calculate_lux(full, ir) # convert raw values
/ cpl
                                                         to lux
    # The highest value is the approximate lux
                                                            def test(int_time=INTEGRATIONTIME_100MS,
                                                         gain=GAIN HIGH):
equivalent
    return max([lux1, lux2])
                                                              tsl.set gain(gain)
                                                              tsl.set timing(int time)
                                                              full test, ir test = tsl.get full luminosity()
  def enable(self):
    self.bus.write byte data(
                                                              lux test = tsl.calculate lux(full test, ir test)
                                                              fob=open('/home/pi/data.txt','a')
            self.sendor address,
            COMMAND BIT | REGISTER ENABLE.
                                                                 fob.write(" \n")
            ENABLE POWERON | ENABLE AEN |
                                                                 fob.write('Max Reading\n')
                                                                 fob.write('full = %i\n' % (full_test))
ENABLE AIEN
                                                                 fob.write("integration time = %i\n" %
            ) # Enable
                                                         tsl.get timing())
  def disable(self):
                                                                 fob.write("gain = %i \n" % tsl.get gain())
    self.bus.write byte data(
                                                                 fob.close()
                                                                 print ('full = %i' % (full_test))
            self.sendor address,
            COMMAND BIT | REGISTER ENABLE,
                                                              print("integration time = %i" % tsl.get timing())
            ENABLE POWEROFF
                                                              print("gain = %i \n" % tsl.get_gain())
            )
  def get full luminosity(self):
                                                            for i in [INTEGRATIONTIME 200MS]:
                                                              test(i, GAIN_HIGH)
    self.enable()
    time.sleep(0.120*self.integration time+1) # not
sure if we need it "// Wait x ms for ADC to complete"
    full = self.bus.read word data(
                                                         self.sendor address, COMMAND BIT |
                                                         REGISTER CHAN0 LOW
                                                         sample = raw input("Insert cuvette with sample to run
                                                         sample, press ENTER to continue.")
    ir = self.bus.read word data(
                                                         print 'Measuring Sample Readings, Please wait.'
            self.sendor_address, COMMAND_BIT |
                                                         import smbus
REGISTER CHAN1 LOW
                                                         import time
    self.disable()
                                                         VISIBLE = 2 # channel 0 - channel 1
    return full, ir
                                                         INFRARED = 1 # channel 1
                                                         FULLSPECTRUM = 0 # channel 0
  def get luminosity(self, channel):
    full, ir = self.get full luminosity()
                                                         ADDR = 0x29
    if channel == FULLSPECTRUM:
                                                         READBIT = 0x01
       # Reads two byte value from channel 0
                                                         COMMAND BIT = 0xA0 # bits 7 and 5 for 'command
(visible + infrared)
                                                         CLEAR BIT = 0x40 # Clears any pending interrupt
       return full
                                                         (write 1 to clear)
    elif channel == INFRARED:
```

```
WORD BIT = 0x20 \# 1 = read/write word (rather)
                                                                 self.sendor address,
than byte)
                                                                 COMMAND BIT I
BLOCK_BIT = 0x10 # 1 = using block read/write
                                                      REGISTER_CONTROL,
ENABLE POWERON = 0x01
                                                                 self.integration_time | self.gain
ENABLE POWEROFF = 0x00
                                                          self.disable()
ENABLE_AEN = 0x02
ENABLE AIEN = 0x10
CONTROL RESET = 0x80
                                                        def get timing(self):
LUX DF = 408.0
                                                          return self.integration time
LUX COEFB = 1.64 # CH0 coefficient
LUX_COEFC = 0.59 # CH1 coefficient A
                                                        def set gain(self, gain):
LUX COEFD = 0.86 # CH2 coefficient B
                                                          self.enable()
                                                          self.gain = gain
REGISTER ENABLE = 0x00
                                                          self.bus.write byte data(
REGISTER CONTROL = 0x01
                                                                 self.sendor_address,
REGISTER_THRESHHOLDL_LOW = 0x02
                                                                 COMMAND_BIT |
REGISTER THRESHHOLDL HIGH = 0x03
                                                      REGISTER CONTROL,
REGISTER THRESHHOLDH LOW = 0x04
                                                                 self.integration time | self.gain
REGISTER THRESHHOLDH HIGH = 0x05
REGISTER INTERRUPT = 0x06
                                                          self.disable()
REGISTER CRC = 0x08
REGISTER ID = 0x0A
                                                        def get gain(self):
REGISTER CHAN0 LOW = 0x14
                                                          return self.gain
REGISTER CHANO HIGH = 0x15
REGISTER CHAN1 LOW = 0x16
                                                        def calculate lux(self, full, ir):
REGISTER CHAN1 HIGH = 0x17
                                                          # Check for overflow conditions first
INTEGRATIONTIME_100MS = 0x00
                                                          if (full == 0xFFFF) | (ir == 0xFFFF):
INTEGRATIONTIME_200MS = 0x01
                                                             return 0
INTEGRATIONTIME 300MS = 0x02
INTEGRATIONTIME 400MS = 0x03
                                                          case integ = {
INTEGRATIONTIME 500MS = 0x04
                                                             INTEGRATIONTIME 100MS: 100.,
INTEGRATIONTIME_600MS = 0x05
                                                             INTEGRATIONTIME 200MS: 200.,
                                                             INTEGRATIONTIME_300MS: 300.,
GAIN_HIGH = 0x20 \# medium gain (428x)
                                                             INTEGRATIONTIME_400MS: 400.,
                                                             INTEGRATIONTIME 500MS: 500.,
class Tsl2591(object):
                                                             INTEGRATIONTIME 600MS: 600.,
  def __init__(
         self.
                                                          if self.integration time in case integ.keys():
                                                             atime = case_integ[self.integration_time]
         i2c bus=1,
                                                          else:
         sensor address=0x29,
         integration=INTEGRATIONTIME 100MS.
                                                             atime = 100.
         gain=GAIN HIGH
                                                          case gain = {GAIN HIGH: 428.}
    self.bus = smbus.SMBus(i2c bus)
    self.sendor address = sensor address
                                                          if self.gain in case gain.keys():
    self.integration_time = integration
                                                             again = case_gain[self.gain]
    self.gain = gain
    self.set timing(self.integration time)
                                                             again = 1.
    self.set_gain(self.gain)
    self.disable() # to be sure
                                                          # cpl = (ATIME * AGAIN) / DF
                                                          cpl = (atime * again) / LUX DF
  def set timing(self, integration):
                                                          lux1 = (full - (LUX COEFB * ir)) / cpl
    self.enable()
    self.integration time = integration
                                                          lux2 = ((LUX_COEFC * full) - (LUX_COEFD * ir))
    self.bus.write byte data(
                                                      / cpl
```

```
# The highest value is the approximate lux
equivalent
    return max([lux1, lux2])
  def enable(self):
    self.bus.write_byte_data(
            self.sendor_address,
            COMMAND BIT | REGISTER ENABLE.
            ENABLE POWERON | ENABLE AEN |
ENABLE AIEN
            ) # Enable
  def disable(self):
    self.bus.write byte data(
            self.sendor_address,
            COMMAND_BIT | REGISTER_ENABLE,
            ENABLE POWEROFF
  def get full luminosity(self):
    self.enable()
    time.sleep(0.120*self.integration time+1) # not
sure if we need it "// Wait x ms for ADC to complete"
    full = self.bus.read_word_data(
            self.sendor address, COMMAND BIT |
REGISTER_CHAN0_LOW
    ir = self.bus.read_word_data(
            self.sendor address, COMMAND BIT |
REGISTER CHAN1 LOW
    self.disable()
    return full, ir
  def get luminosity(self, channel):
    full, ir = self.get full luminosity()
    if channel == FULLSPECTRUM:
       # Reads two byte value from channel 0
(visible + infrared)
       return full
    elif channel == INFRARED:
       # Reads two byte value from channel 1
(infrared)
       return ir
    elif channel == VISIBLE:
       # Reads all and subtracts out ir to give just the
visible!
       return full - ir
    else: # unknown channel!
       return 0
if name == ' main ':
  tsl = Tsl2591() # initialize
  full, ir = tsl.get full luminosity() # read raw values
(full spectrum and ir spectrum)
  lux = tsl.calculate lux(full, ir) # convert raw values
to lux
```

```
def test(int_time=INTEGRATIONTIME_100MS,
gain=GAIN HIGH):
     tsl.set gain(gain)
     tsl.set timing(int time)
     full test, ir test = tsl.get full luminosity()
     lux test = tsl.calculate lux(full test, ir test)
     fob=open('/home/pi/data.txt','a')
        fob.write("\n")
        fob.write('Sample Reading\n')
        fob.write('full = %i\n' % (full test))
        fob.write("integration time = %i\n" %
tsl.get_timing())
        fob.write("gain = %i \n" % tsl.get gain())
        fob.close()
        print ('full = %i' % (full_test))
     print("integration time = %i" % tsl.get_timing())
     print("gain = %i \n" % tsl.get_gain())
  for i in [INTEGRATIONTIME 200MS]:
     test(i, GAIN HIGH)
```