Trichromy Saturation

This is a lab guide for creating a sample circuit for the TCS34725 RGB sensor. The sample will use the sensor to read an RGB color placed on top of it, and then blink the corresponding LEDs at a rate relative to the overall saturation of each color component.

Background:

The Young-Helmholtz theory of color vision, also known as the trichromatic theory, states that there are three different types of cone receptors in the human eye to perceive color. One of these is receptive to short wavelength (blue) light, one medium wavelength (green) light, and the third long wavelength (red) light.

Approximate Time (Excluding Preparation Work): 1 Hour

Prerequisites:

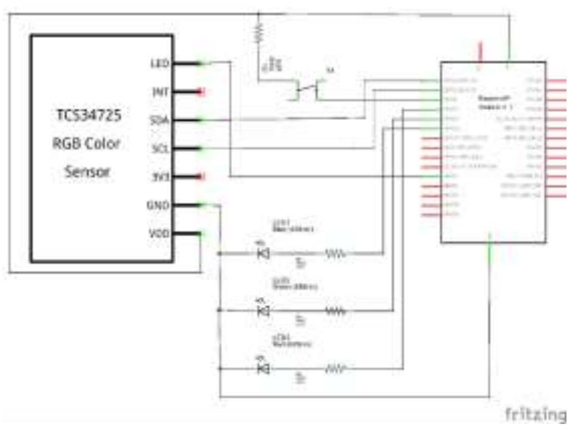
- The latest public build of Windows 10
- Visual Studio 2015 with Update 1
- Windows IoT Core Templates (https://visualstudiogallery.msdn.microsoft.com/55b357e1-a533-43ad-82a5-a88ac4b01dec)
- Windows IoT Core Dashboard (http://go.microsoft.com/fwlink/?LinkID=708576)
- Download sketch code from (http://github.com/SkoonStudios/TrichromySaturation)

Part List:

This lab was designed to use the parts contained within the Adafruit Windows 10 IoT Starter Kit.

- Raspberry Pi 2
- Full Size Breadboard
- TCS34725 RGB Color Sensor
- Pushbutton
- (3x) LEDs (red, blue, and green)
- (3x) 560 Ohm resistor
- 10k Ohm resistor
- Connector Wires

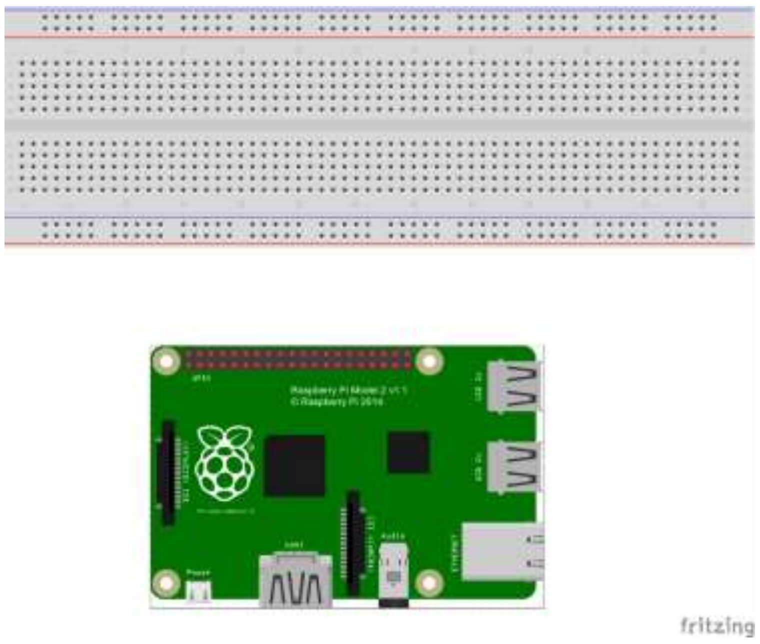
WIRING SCHEMATIC:



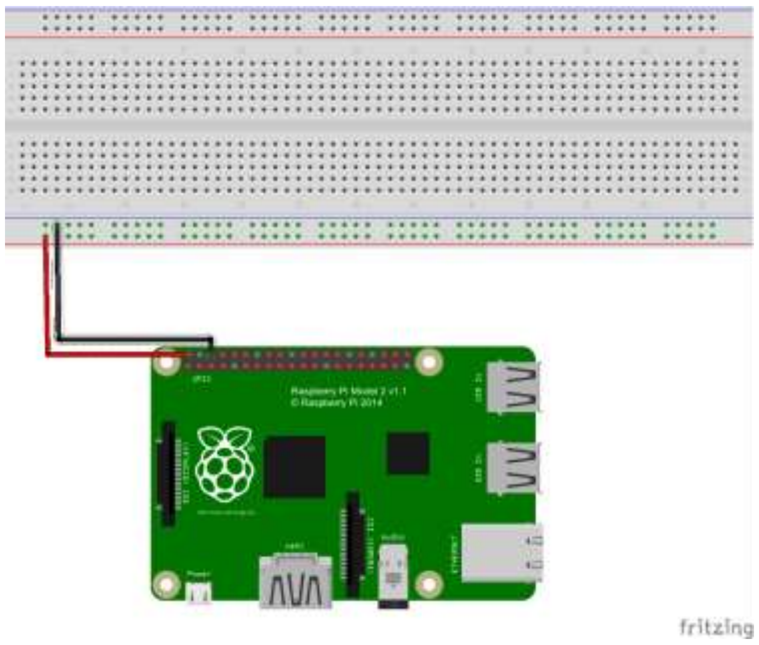
INSTRUCTIONS:

A) Building The Circuit

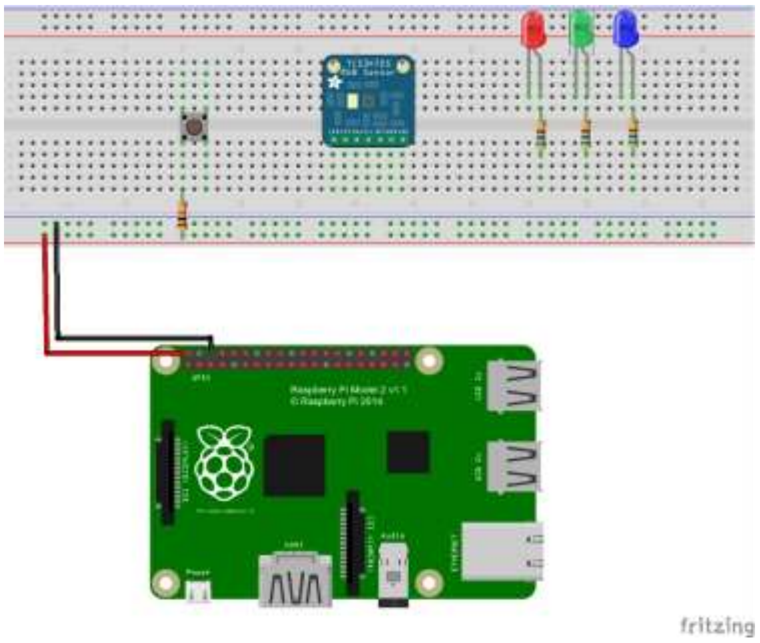
1. Lay out the Raspberry Pi and the breadboard



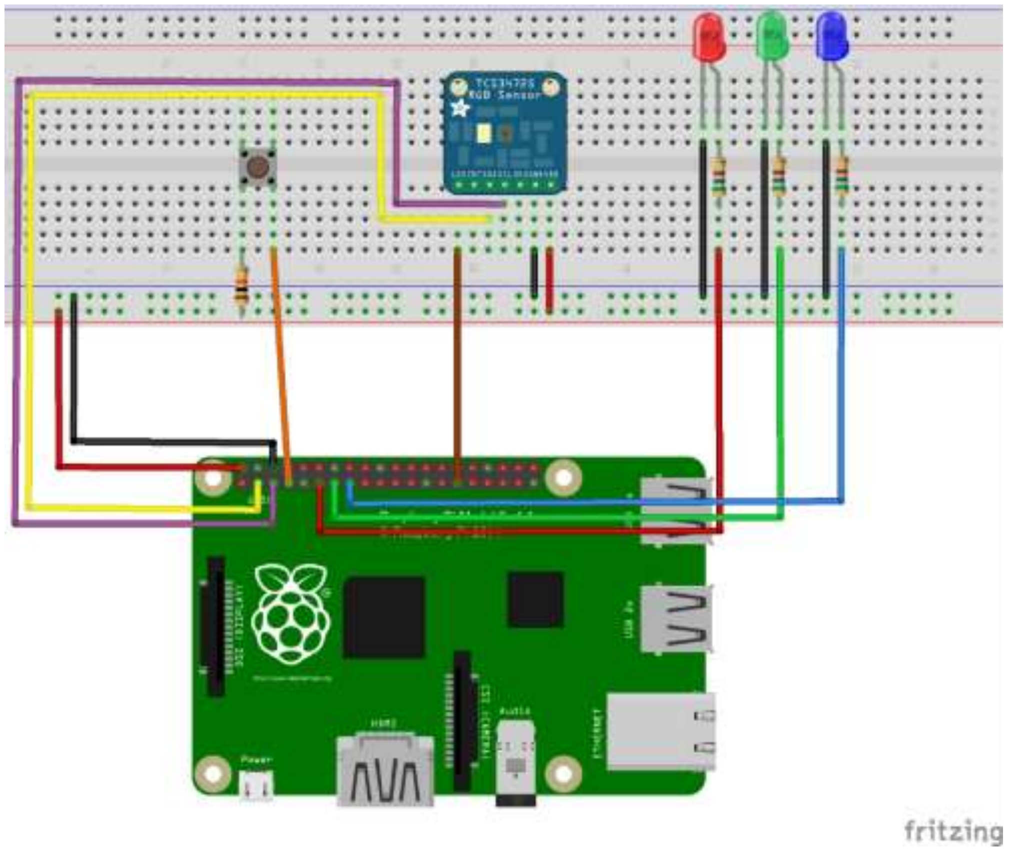
2. Wire the power rails



3. Place the components in the breadboard



4. Wire the components to the Raspberry Pi. Note that it is important to wire the GPIO pins on the Raspberry Pi exactly as shown below as the code is programmed to use those pins.



B) The Code

After downloading the code from the gihub repository referenced above, open up the SLN file to load the project into Visual Studio.

1. TCS34725.cs

   This is the main class wrapper that was generated from the data sheet for the sensor. The methods contained within will set the appropriate addresses to read the data.

   - The class wrapper is created by instantiating a new instance of the ***TCS34725*** object. The default constructor takes an optional parameter for the pin that the LED light embedded on the sensor is hooked up onto. The wiring diagram above uses GPIO Pin 5 for the sensor LED light.
   - The LED on the sensor is controlled via the ***LedState*** property.
   - The class exposes two main asynchronous functions that will read the data from the sensor.
     - ***GetRgbData()*** returns normalized RGB data retrieved from the sensor with a value of 0 – 255.
     - ***GetRawData()*** returns raw RGB data retrieved from the sensor.

2. MainPage.xaml
This is the primary page that is navigated to once the app is launched. The app is designed to run headless (without needing a monitor or mouse / keyboard) or headed. When hooked up to a display, when the sensor data is read, a color sample will display on the screen. Additionally you can manually simulate a sensor read to see what the resultant LED light pattern would be. The following code will add the UI elements to the XAML page.

```
<Grid Background="Black">

    <Slider x:Name="sldRed" VerticalAlignment="Top" Margin="20,35,0,0"
HorizontalAlignment="Left" Width="175" Maximum="255" TickFrequency="10"
Background="#66FF0000" Foreground="#FFFF0000" BorderBrush="Red"
ValueChanged="sld_ValueChanged"/>

    <Slider x:Name="sldGreen" HorizontalAlignment="Left" Margin="20,75,0,0"
VerticalAlignment="Top" Width="175" Maximum="255" TickFrequency="10" Background="#6600FF00"
Foreground="#FF00FF00" BorderBrush="Green" ValueChanged="sld_ValueChanged"/>

    <Slider x:Name="sldBlue" HorizontalAlignment="Left" Margin="20,115,0,0"
VerticalAlignment="Top" Width="175" Maximum="255" TickFrequency="10" Background="#660000FF"
Foreground="#FF0000FF" BorderBrush="Blue" ValueChanged="sld_ValueChanged"/>

    <Button x:Name="cmdSet" Content="Set" HorizontalAlignment="Left" VerticalAlignment="Top"
FontSize="12" Margin="105,155,0,0" Click="cmdSet_Click" BorderBrush="White"
Foreground="White"/>

    <TextBlock x:Name="txtRed" HorizontalAlignment="Left" Margin="205,46,0,0"
TextWrapping="Wrap" Text="0" VerticalAlignment="Top" Foreground="Red" FontSize="9"/>

    <TextBlock x:Name="txtGreen" HorizontalAlignment="Left" Margin="205,86,0,0"
TextWrapping="Wrap" Text="0" VerticalAlignment="Top" Foreground="#8800FF00" FontSize="9"/>

    <TextBlock x:Name="txtBlue" HorizontalAlignment="Left" Margin="205,126,0,0"
TextWrapping="Wrap" Text="0" VerticalAlignment="Top" Foreground="Blue" FontSize="9"/>

    <Rectangle x:Name="rectSample" HorizontalAlignment="Left" Height="68" Margin="231,61,0,0"
VerticalAlignment="Top" Width="102" Fill="Black" Stroke="White"/>

</Grid>
```

3. MainPage.Xaml.cs
   This is the code behind page for the main UI. It is broken up into the following sections
   - Variable Declaration

```
// Constant pin assignments
private const int PUSH_BUTTON_PIN = 4;
private const int RED_LED_PIN = 17;
private const int GREEN_LED_PIN = 27;
private const int BLUE_LED_PIN = 22;
private const int RGB_LED_PIN = 5;

// Class wrapper for the color sensor
private TCS34725 _colorSensor = null;

// GPIO pins for the pushbutton and LEDs
private GpioPin _gpioPushbutton = null;
private GpioPin _gpioRedLED = null;
private GpioPin _gpioGreenLED = null;
private GpioPin _gpioBlueLED = null;
private GpioPinValue _pinValRed;
private GpioPinValue _pinValGreen;
private GpioPinValue _pinValBlue;

// Other variables
private readonly TimeSpan LED_OFF_DURATION_MSEC = TimeSpan.FromMilliseconds(100); // 100 msec
private TimeSpan _redLEDDurationMsec;
private TimeSpan _greenLEDDurationMsec;
private TimeSpan _blueLEDDurationMsec;

// Timers for blinking LEDs
private DispatcherTimer _timerRed;
private DispatcherTimer _timerGreen;
private DispatcherTimer _timerBlue;

private bool  isInitialized = false;
```

- GPIO and Variable Initialization

```
private async void InitializeGPIO()
{
    GpioController controller = GpioController.GetDefault();

    if (null != controller)
    {
        // Create and initialize color sensor instance
        _colorSensor = new TCS34725(RGB_LED_PIN);
        await _colorSensor.Initialize();
        _colorSensor.LedState = TCS34725.eLedState.Off;

        // Setup button pin
        _gpioPushbutton = controller.OpenPin(PUSH_BUTTON_PIN);
        _gpioPushbutton.DebounceTimeout = TimeSpan.FromMilliseconds(100); // 100 ms
        _gpioPushbutton.SetDriveMode(GpioPinDriveMode.Input);
        _gpioPushbutton.ValueChanged += gpioPushbutton_ValueChanged;

        // Setup LEDs
        // Red
        _gpioRedLED = controller.OpenPin(RED_LED_PIN);
        _gpioRedLED.SetDriveMode(GpioPinDriveMode.Output);
        _pinValRed = SetGpioPinState(_gpioRedLED, GpioPinValue.High); // HIGH == ON
        _timerRed = new DispatcherTimer();
        _timerRed.Tick += timerRed_Tick;

       // Setup other colors
       ...
    }
}
```

- Calculating the LED blink rates.
  This code simply looks at the value of the color passed into the function. The higher the color value, the smaller the timer value which will cause the LED to blink faster. The code breaks the blink rate into 5 thresholds.

```
private TimeSpan GetLEDDurationForValue(int colorValue)
{
    if ((colorValue < 0) || (colorValue > 255))
    {
        throw new ArgumentOutOfRangeException("Invalid Value");
    }

    // This sample only has 5 blink rates.  Subtract out the off duration
    // so that the blink will remain in sync across all possible values
    if(colorValue < 51) {
                return (TimeSpan.FromMilliseconds(1600) - LED_OFF_DURATION_MSEC); }
    else if (colorValue < 102) {
                return (TimeSpan.FromMilliseconds(1300) - LED_OFF_DURATION_MSEC); }
    else if (colorValue < 153) {
                return (TimeSpan.FromMilliseconds(1000) - LED_OFF_DURATION_MSEC); }
    else if (colorValue < 204) {
                return (TimeSpan.FromMilliseconds(700) - LED_OFF_DURATION_MSEC); }
    else {
                return (TimeSpan.FromMilliseconds(400) - LED_OFF_DURATION_MSEC); }

}
```

- Handling Events
  The code in the button event will fire when the button is released. The LED light is temporarily turned back on and then the data is read before the light is turned back off.

  On the timer tick event, the state of the LED will be toggled and the timer will be reset for the next event.

```
private async void gpioPushbutton_ValueChanged(GpioPin sender, GpioPinValueChangedEventArgs
args)
{
    //Only read the sensor value when the button is released
    if (args.Edge == GpioPinEdge.RisingEdge)
    {
        //Read the approximate color from the sensor
        _colorSensor.LedState = TCS34725.eLedState.On;
        await System.Threading.Tasks.Task.Delay(1000);

        RGBData colorData = await _colorSensor.GetRgbData();
        _colorSensor.LedState = TCS34725.eLedState.Off;

        ...
    }
}

private void timerRed_Tick(object sender, object e)
{
    _timerRed.Stop();
    _pinValRed = SetGpioPinState(_gpioRedLED, (_pinValRed == GpioPinValue.High ?
GpioPinValue.Low : GpioPinValue.High));
    Debug.WriteLine(String.Format("{0}: Red Pin Val: {1}", DateTime.Now.Ticks, _pinValRed));

    _timerRed.Interval = (_pinValRed == GpioPinValue.Low ? LED_OFF_DURATION_MSEC :
_redLEDDurationMsec);
    _timerRed.Start();
}
```

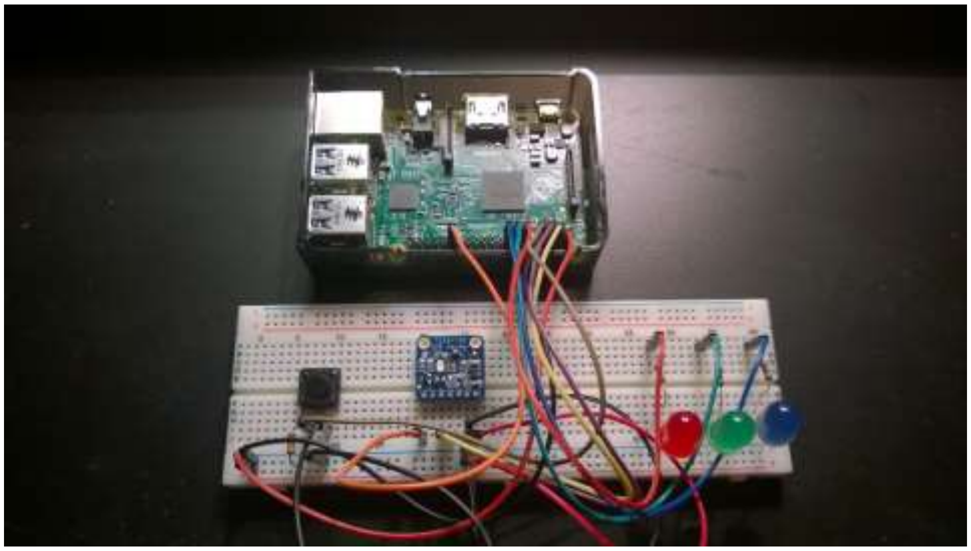
Deploying The Application

1) You can use the IoT Dashboard app to deploy the IoT core image to the SD Card. After inserting the SD Card into the computer, launch the Dashboard app and follow the directions for "Set Up A New Device"
2) Once the image is created, put the SD Card back into the Raspberry Pi and plug in the power. It will take ~5 minutes for the first boot as the device goes through initial configuration.
3) After the device has booted up, it should appear under the "My Devices" section of the dashboard app. Take note of the device name and IP address.
4) In Visual Studio, set the architecture to ARM and change the target from "Local Machine" to "Remote Machine"
5) The device may automatically show up in the Remote Deployment dialog, but if not, enter the device name or IP Address in the "Address" field, leave the "Authentication Mode" as "Universal"
6) Press F5 to deploy the application to the device.

7) You will be able to set breakpoints and debug the code running remotely on the device via the Visual Studio IDE.

SUMMARY / ACKNOWLEDGEMENTS

This project was created as part of a hack night for the Charlotte IoT Meetup Group (http://www.charlotteiot.com). The sample is loosely based on the "What Color Is It?" (https://www.hackster.io/windows-iot/what-color-is-it-578fdb) sample provided as part of the Adafruit Starter Pack for Windows 10 IoT Core on Raspberry Pi 2. We encourage you to innovate and put your own creative touches on this project.



http://www.skoonstudios.com

Last Revised: February 6, 2016

* Skoon Studios, LLC has no affiliation or specific endorsement with Adafruit Industries but in the past have bought components and recommended the detailed tutorials and instructions available on the site.