Capturing, Analyzing, and Displaying Visible Light

CSCI 6907: Embedded Systems

Samuel Bressi

Table of Contents

Project Abstract

The goal of this project is to utilize a color light sensor to capture analog wavelengths of red, green, and blue light and visually display them on a computer monitor. In addition, a mechanism is in place to analyze the color and perform complex color theory mathematics to calculate analogous, complementary, monochromatic, split complementary, and triadic color groups.

A Zilog® ZNEO™ Z16F microcontroller interfaces with the I2C color light sensor[1] and communicates to a graphical user interface on a PC through a serial connection.

Status

Originally, this project was to include a 128x128 pixel LCD screen[2] (as shown at right), to display the captured color as well as toggle between color modes. The project proposal initially stated that the two pushbuttons on the LCD screen would toggle between complementary and harmonious (triadic) color groups. During the development of this embedded system, it was determined that completion of the project using the LCD screen would not be feasible and as such, the scope was modified to include a graphical user interface (GUI) on a PC to display the colors.

To compensate for the lessened scope, I increased the complexity by including other color groups – analogous, monochromatic, and split complementary – in addition to the proposed ones. Aside from this change, the remainder of the project was developed as initially planned.

Hardware

Microcontroller

The Zilgo® ZNEO™ Z16F series (contest kit) flash microcontroller (MCU) is used to control the embedded software and hardware. The MCU is configured to run at approximately 5.5 megahertz. For this project, I utilized the following hardware features of the MCU:

- The four 5x7 LED arrays: D1, D2, D3, and D4
- The I2C interface controller: U2

---

[1] Color light sensor purchased from http://www.sparkfun.com/products/8663. Additional product information is available on page.

[2] LCD screen purchased from http://www.sparkfun.com/products/8600. Additional product information is available on page.

- The console port: P1, using UART0

- The three pushbuttons: SW1, SW2, and SW3

Light Sensor

External to the MCU, I purchased an Avago Technologies ADJD-S371-QR999 light sensor. The sensor is a mere 3.9 x 4.5 x 1.8 millimeters in size but is affixed to a 0.6 x 0.9 inch breakout board for easier access. The breakout board is available at SparkFun Electronics (see footnote 1 on previous page). This breakout board also includes the necessary circuitry to utilize the sensor; however, not all features on the sensor are used. The features that were not used are the white LED for reflective sensing (i.e. sensing the color of reflected light against a surface as opposed to projected light) and the ability to use an external clock source.

The sensor operates at 3.3V pulled from the VDD pin on the MCU. The I²C pins on the sensor are wired to the PA7 (SDA/data) and PA6 (SCL/clock) header pins. The sleep pin on the sensor is not used and is wired to ground to maintain an operational state. The schematic for the hardware components is shown below in Figure 1.
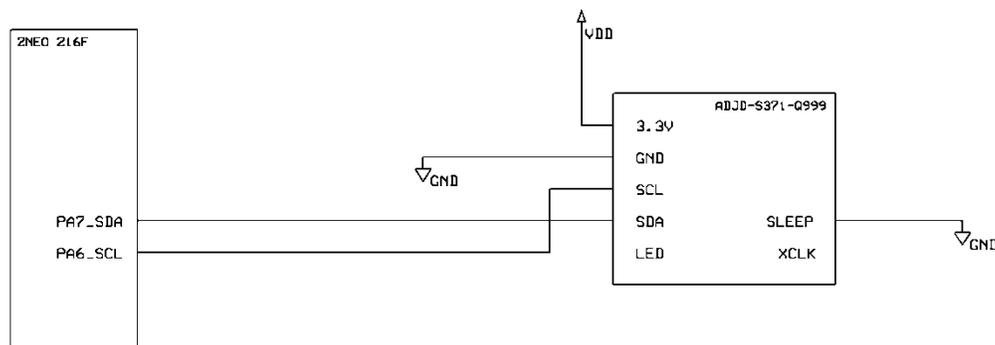

Figure 1: Hardware Schematic

The sensor recognizes four channels: red, green, blue, and clear (white/luminosity). Each channel is captured by 16 capacitors (64 in total arranged in a checkerboard pattern for consistent sensing) and passed through an analog-to-digital converter (ADC). The ADC converts the analog signals into 10-bits per channel digital representations. These digital representations are stored in two 8-bit registers where the **DATA_color_LO** register contains the eight least significant bits (LSB) and the **DATA_color_HI** contains the two most significant bits (MSB), as shown below where *color* can be **RED**, **GREEN**, **BLUE**, or **CLEAR**.

DATA_*color*_HI Register

B7	B6	B5	B4	B3	B2	B1	B0
UNUSED						DATA_*color*[9:8]	

DATA_*color*_LO Register

B7	B6	B5	B4	B3	B2	B1	B0
DATA_*color*[7:0]							

Sensor Operation

The sensor is operated by first initializing the number of capacitors (0x0 to 0xF) to use as well as the number of integration time slots (0x0000 to 0xFFFF) per channel. These values are stored in the **CAP_*color*** and **INT_*color*** registers, respectively. Every sensor is slightly different and the values must be manually calibrated and set. After initialization, the sensor can be instructed to read by writing the 0x01 instruction to register 0x00 and reading the **DATA_*color*_HI**/**LO** registers using standard I²C communication protocols with a slave address of 0x74 (MSB) and a data direction bit (0 to write, 1 to read) in the LSB.

The timing diagram for this sensor is shown below in Figure 2, courtesy of Avago Technologies.

Serial Interface Timing Information

| Parameter	Symbol	Minimum	Maximum	Units
SCL Clock Frequency	f_{scl}	0	100	kHz
(Repeated) START Condition Hold Time	$t_{HD:STA}$	4	-	µs
Data Hold Time	$t_{HD:CAT}$	0	3.45	µs
SCL Clock Low Period	t_{LOW}	4.7	-	µs
SCL Clock High Period	t_{HIGH}	4.0	-	µs
Repeated START Condition Setup Time	$t_{SU:STA}$	4.7	-	µs
Data Setup Time	$t_{SU:DAT}$	250	-	µs
STOP Condition Setup Time	$t_{SU:STD}$	4.0	-	µs
Bus Free Time Between START and STOP Conditions	t_{BUF}	4.7	-	µs
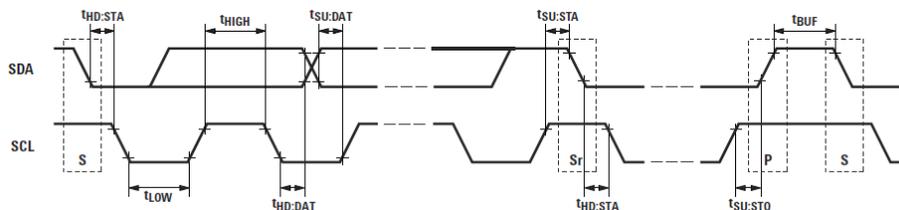
Figure 2 - Color Light Sensor I2C Timing Diagram

Software

The software for this project is divided into two portions: the embedded microcontroller software and the GUI.

Embedded Software

The embedded software is low-level application code that resides on the MCU. The source code was written in C using the ZDS II integrated development environment. This code is responsible for the operation of the MCU and color light sensor. The light sensor source code is organized in a three-tier structure depicted in Figure 1.
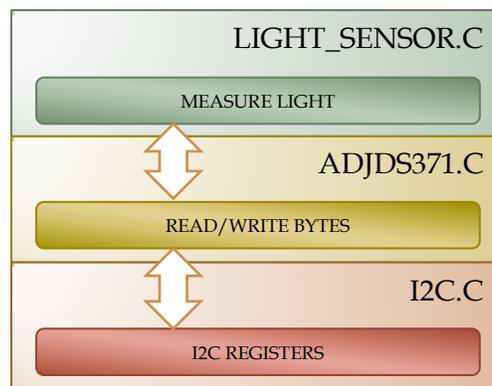
The following files make up the embedded application:

- **ADJDS371.C, ADJDS371.H** – The device driver software that controls the ADJD-S371-Q999 color light sensor. The source file performs all I2C communication and abstracts the functionality for use in other pieces of code. The header file defines the I2C slave address, operation modes, and the sensor control registers/instructions. The header file also exposes three core functions:
  - **void adjds371_initialize(void)** – Initialize the ADJD-S371-Q999 sensor
  - **void adjds371_write_byte(unsigned char register_address, unsigned char register_value)** – Write the **register_value** byte to the **register_address** register on the sensor
  - **unsigned char adjds371_read_byte(unsigned char register_address)** – Return the byte stored in the **register_address** register on the sensor
- **ASCII.H** – The header file that defines the mapping for each alphanumeric character in the ASCII character set for use on the 5x7 LEDs.
- **BUTTONS.C, BUTTONS.H** – The source code to initialize and poll the MCU pushbuttons (SW1, SW2, and SW3). Pressing each button performs the following:

- o SW1 – Increases the number of red capacitors used by the sensor and displays feedback to the LEDs.

- o SW2 – Increases the number of green capacitors used by the sensor and displays feedback to the LEDs

- o SW3 – Increases the number of blue capacitors used by the sensor and displays feedback to the LEDs

- **CLOCK.C, CLOCK.H** – The source code to initialize the internal 5.5 MHz clock. The header file defines the clock speed and LED refresh rate (in milliseconds) constants.

- **I2C.C, I2C.H** – The lowest level source code used by the light sensor. The functions contained within the I2C files interact directly with the MCU registers for I2C data, status, and control.

- **LEDS.C, LEDS.H** – The source code used to initialize and operate the LED arrays on the MCU. This includes the high-level functions for setting the text that appears on the LEDs as well as the low-level functions for operating the D1, D2, D3, and D4 anodes and cathodes, scanning the individual characters, or scrolling the text (if used).

- **LIGHT_SENSOR.C, LIGHT_SENSOR.H** – The highest level source code used by the light sensor. The functions contained in the **LIGHT_SENSOR** files are used to increase the red, green, and blue capacitors (see **BUTTONS**), set the number of integration time slots, and take readings from the sensor. The header file contains a constant for configuring the number of samples for each reading.

- **MAIN.C** – The main thread and initialization of the embedded software.

- **SYSTEM.C, SYSTEM.H** – The source code used to perform supplementary system-wide functions, such as waiting a specific number of milliseconds.

- **TIMERS.C, TIMERS.H** – The source code used to initialize and manage the timer. The files contain the interrupt service routine (ISR) attached to the **TIMER0** vector for scanning and scrolling the LED characters.

- **TYPES.H** – The header file used to define some data type shortcut aliases, such as **byte** and **boolean**.

- **UART.C, UART.H** – The source code used to initialize the **UART0** receiver/transmitter.
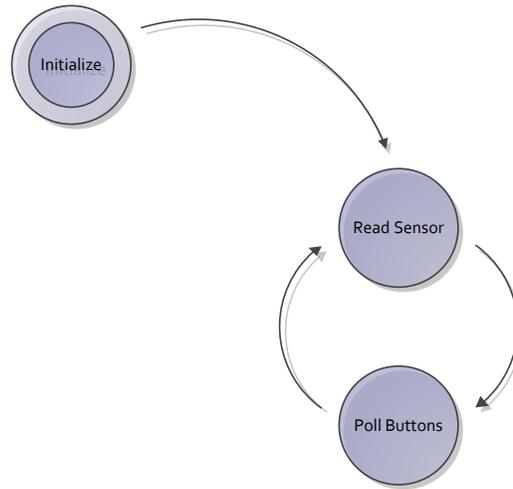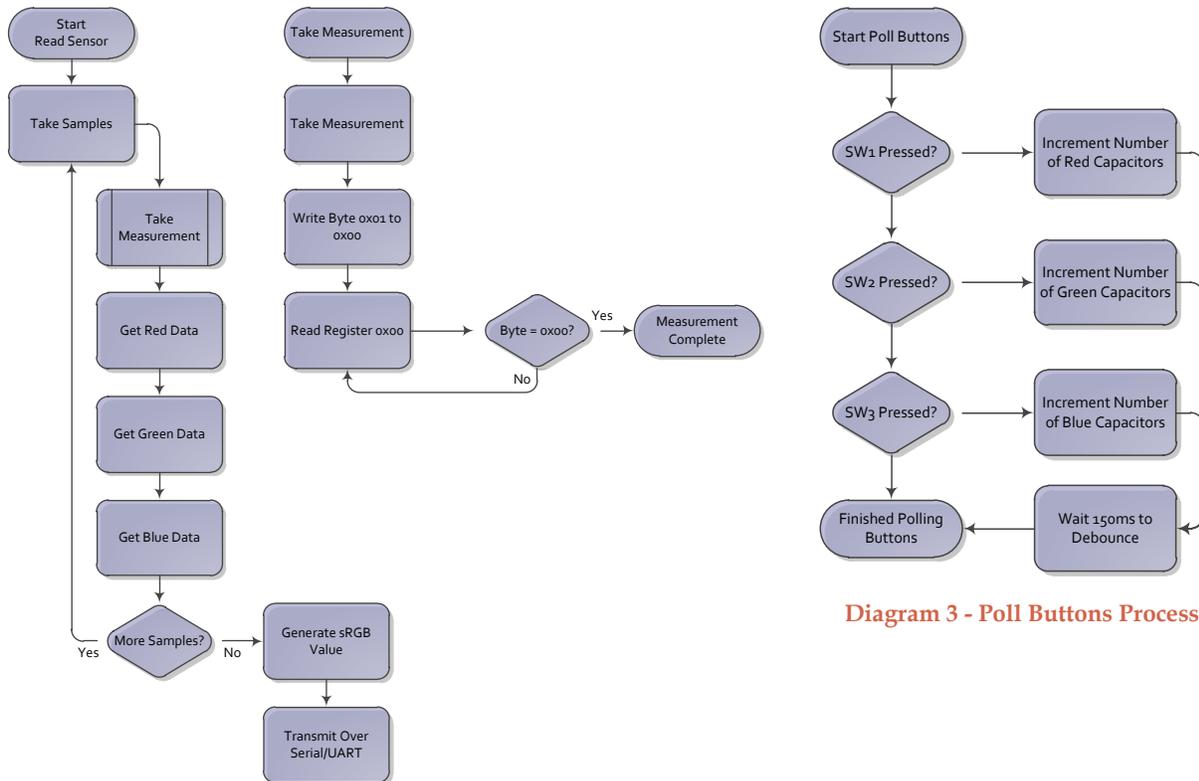
State and Process Diagrams

Initialize

Read Sensor

Poll Buttons

Diagram 1 - Main Thread State Diagram

Start
Read Sensor

Take Samples

Take Measurement

Get Red Data

Get Green Data

Get Blue Data

More Samples?

Yes

No

Generate sRGB Value

Transmit Over Serial/UART

Take Measurement

Take Measurement

Write Byte 0x01 to 0x00

Read Register 0x00

Byte = 0x00?

Yes

No

Measurement Complete

Diagram 2 - Read Sensor Process

Start Poll Buttons

SW1 Pressed?

Increment Number of Red Capacitors

SW2 Pressed?

Increment Number of Green Capacitors

SW3 Pressed?

Increment Number of Blue Capacitors

Finished Polling Buttons

Wait 150ms to Debounce

Diagram 3 - Poll Buttons Process

Graphical User Interface

To analyze and represent the colors, a GUI is used on a client PC. This GUI was developed using Java 1.6 with the Eclipse integrated development environment. The GUI is compiled into an executable JAR file that accepts a single parameter for the COM port that connects to the MCU. This PC must be connected to the MCU via a serial cable/COM port, or a virtual COM port using a serial-to-USB converter. The rate for the communication must be 57600 baud. The GUI is broken into three tabs: Color Monitor, Serial Output, and Color Mode.

- **Color Monitor** – A real-time view of the color that is captured by the light sensor. The hexadecimal representation is shown in bold for the sensor color. The other colors that appear on this tab are the analogous, complementary, monochromatic, split complementary, or triadic colors (mode can be viewed under the Color Mode tab).

    o **Lock/Unlock Color Sensor** – This button can be used to freeze the light sensor so the light source can be removed.
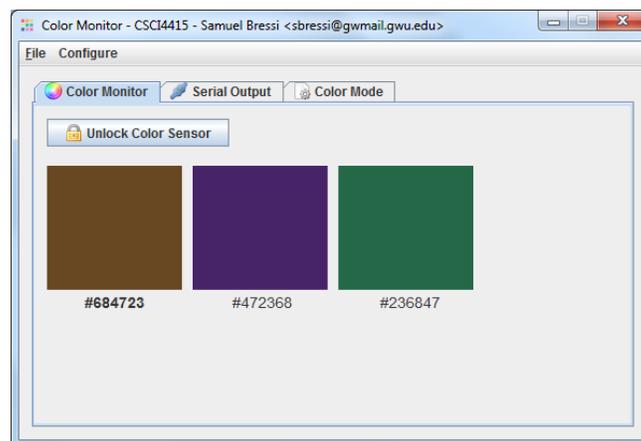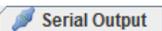
- **Serial Output** – A real-time view of the data coming from the MCU through the serial port. The data that is transmitted is the 4-byte long integer representation of the sRGB color. The MCU captures 10 bits per channel (red, green, blue) and reduces it to an 8-bit value for sRGB representation.

    o sRGB contains one byte for the alpha channel (transparency), one byte for the red channel, one byte for the green channel, and one byte for the blue channel.
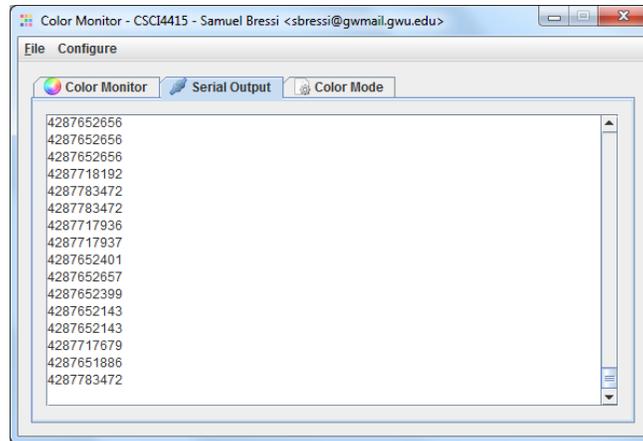
- **Color Mode** – The current method for determining the second, third, and fourth colors on the Color Monitor tab. (Note: not all modes use all four color canvases)

  o **Analogous** – Two colors are considered to be analogous to the base color if their hues are within a specified number of degrees from the base hue on a 360° color wheel. The saturation levels and lightness remain the same for all colors. This application uses 45° above and below the base color.

  o **Complementary** – One color is considered to be the complement to the base color if its hue is exactly 180° away from the base hue on a 360° color wheel. The saturation level and lightness is the same as the base color.

  o **Monochromatic** – One or more colors are considered to be monochromatic if their lightness varies but maintains the same hue and saturation level.

  o **Split Complementary** – Two colors are considered to be the split complements to the base color if they are analogous to the complement of the base color.

  o **Triadic** – Two colors and the base color are considered to be triadic (harmonious) if all three colors maintain the same color depth on differing channels. That is, the color that is represented as RGB(128, 200, 100) is harmonious to the color represented as RGB(200, 100 , 128).
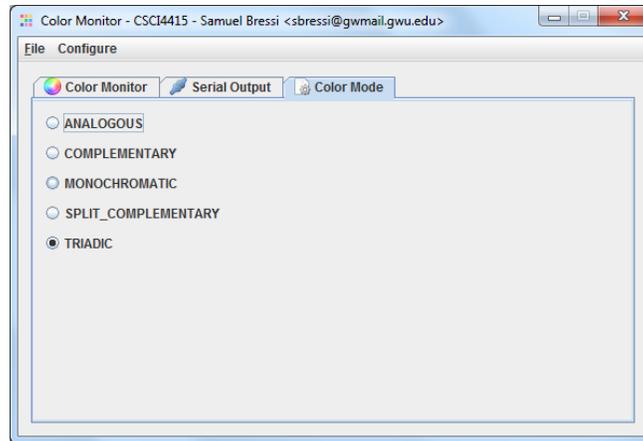
Practical Applications

The system that I developed is useful for applications that require aesthetic color matching. While I did not use the onboard LED for reflective sensing, it would be quite simply to implement. Using this feature, you would be able to reflect light off a paint swatch, for example, to find colors that are aesthetically pleasing. This would be ideal for an application to find accent colors while painting the interior of your house.

Retrospective

This project primarily incorporates serial communication over a UART and through I²C. While these two tasks are generally considered to be trivial; the underlying details of the color light sensor added a lot of complexity. Specifically, understanding the gain and integration times as well as converting the three 10-bit channels to a 32-bit sRGB value with four 8-bit channels.

I would have enjoyed getting the LCD to function but I am happy with the results as they are. The LCD would have given me the ability to swap between two modes while the GUI provides the ability to swap between as many as desired. I also believe that the UART communication adds more to the project than just reading a set of bits from one register (the sensor's) and moving them to another (the LCD's). Using the UART required string manipulation, buffering, and processing that required the MCU and the GUI to maintain the communication channel.

While the 10-bit to 8-bit channel degradation does not really result in too much of a noticeable difference, I would have liked to procure a color sensor that reads in 8-bit channels

instead of the one that I purchased. This would have allowed me to focus more time getting the LCD screen working.

After development, I added the pushbutton functionality. When I was testing the sensor, it was picking up too much saturation in the red channel. After reading the data sheet, I realized that each sensor had to be optimized manually. Rather than hard coding the gain, I added functions to increase the gain by pressing one of the three buttons (corresponding to the red, green, and blue channels). This allowed me to watch the sensor through the GUI while I calibrated it in real time without the need to recompile and reload the firmware. Additionally, I realized that the sensor was calibrated for the ambient lighting in my house and not for the classroom where the demonstration will take place. These buttons will allow me to calibrate it for that lighting the day of the demonstration.

Diffusion and intensity of light remains a variable with using this system. I tested the system with two flashlights: a bright white LED Mini Maglite® and a generic warm color flashlight. There is a significant and visible different with the results from the two light sources. The Mini Maglite® is incredibly luminous and washes out color causing the sensor to max out the capacitors (resulting in #FFFFFF white). I was able to limit this to an extent by putting a translucent diffuser (such as a facial tissue, piece of parchment paper, or regular computer printer paper) between the light source and the sensor and received mixed results. The sensor is the type typically found inside cell phones or small handheld devices where light is channeled through a very small opening. Using the sensor with an open space seems to overwhelm it.

Timelines

3/22/2011
Provide Project Status Report

4/15/2011
Class Demonstration

4/26/2011
Final Report Due

Planned Timeline

| 3/5/2011 - 3/16/2011 Preparation | 3/16/2011 - 4/14/2011 Software Development and Testing | 4/14/2011 - 4/26/2011 Project Wrap-Up |

4/1/2011

3/5/2011

4/26/2011

3/20/2011
Complete Sensor
Integration and Testing

4/12/2011
Complete LCD
Integration and Testing

3/22/2011
Provide Project Status Report

4/15/2011
Class Demonstration

4/26/2011
Final Report Due

Actual Timeline

| 3/5/2011 - 3/16/2011 Preparation | 3/16/2011 - 4/14/2011 Software Development and Testing | 4/14/2011 - 4/26/2011 Project Wrap-Up |

4/1/2011

3/5/2011

4/26/2011

3/24/2011
Complete Sensor
Integration and Testing

4/14/2011
Complete GUI
Development and Testing