Interfacing TCS34725 Color Sensor with PSoC Microcontroller for RGB Color Detection and display on MATLAB GUI

Embedded Systems Lab Report

Submitted by:

Name: Aditya Krishna Jaiswal
Roll Number: 2201011

Department of Electronics and Communication Engineering
Indian Institute of Information Technology Guwahati

Date of submisson: April 22, 2025

1. Aim

To interface the TCS34725 color sensor with a PSoC microcontroller using the I2C protocol, read and display RGB values, transmit the RGB data using USBUART and display the RGB values and detected colour using MATLAB GUI.

2. Apparatus Required

- PSoC 5LP Development Kit
- TCS34725 RGB Color Sensor Module
- 16x2 LCD Display
- USB Cable
- Jumper Wires and Breadboard
- PC with Serial Terminal (like TeraTerm or PuTTY)

3. Theory

3.1. TCS34725 Color Sensor

The TCS34725 is an RGB color light-to-digital converter with an IR blocking filter. It provides precise color measurements in the form of 16-bit digital values for the red, green, blue, and clear light components. It includes:

- A 3-channel ADC
- Adjustable integration time and gain
- Built-in IR filter to remove infrared light for accurate color sensing
- I2C digital interface

3.2. I2C Protocol

I2C (Inter-Integrated Circuit) is a synchronous, multi-master, multi-slave, packet-switched, single-ended, serial communication bus. It uses only two lines:

- SDA (Serial Data Line)
- SCL (Serial Clock Line)

The I2C communication is initiated by the master (in this case, PSoC). Each slave device has a unique 7-bit address. The TCS34725 sensor responds to address 0x29 (7-bit).

Key terms:

- **Start Condition:** Initiated by the master to begin communication.
- **Stop Condition:** Indicates the end of communication.

- **ACK/NACK:** Acknowledge/Not acknowledge signals sent after each byte.
- **Register Addressing:** Commands start with 0x80 as a command bit.

4. Top Design and Pin-Out

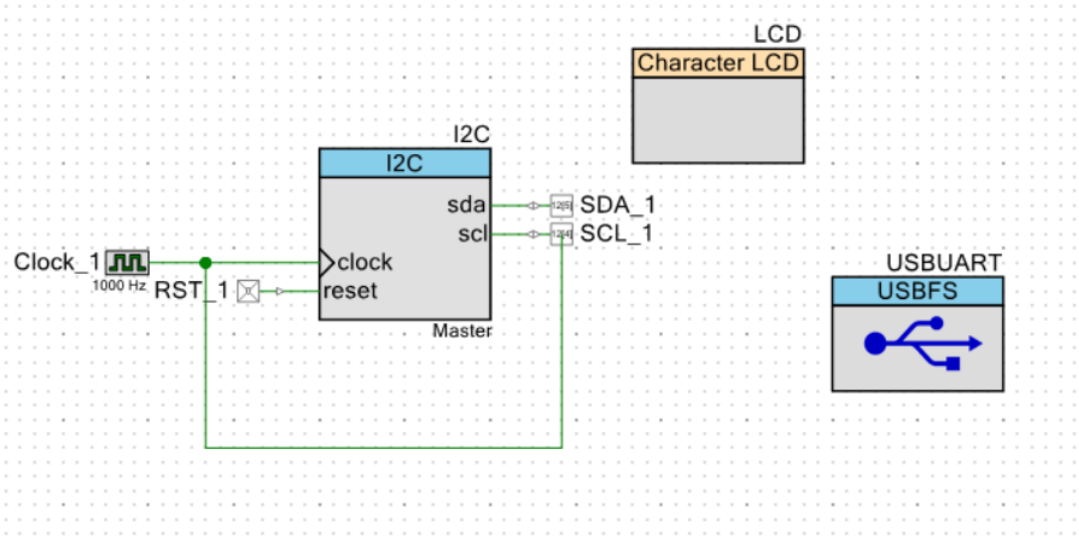


Figure 1: Top Design for Interfacing TCS34725 Color Sensor with PSoC Microcontroller

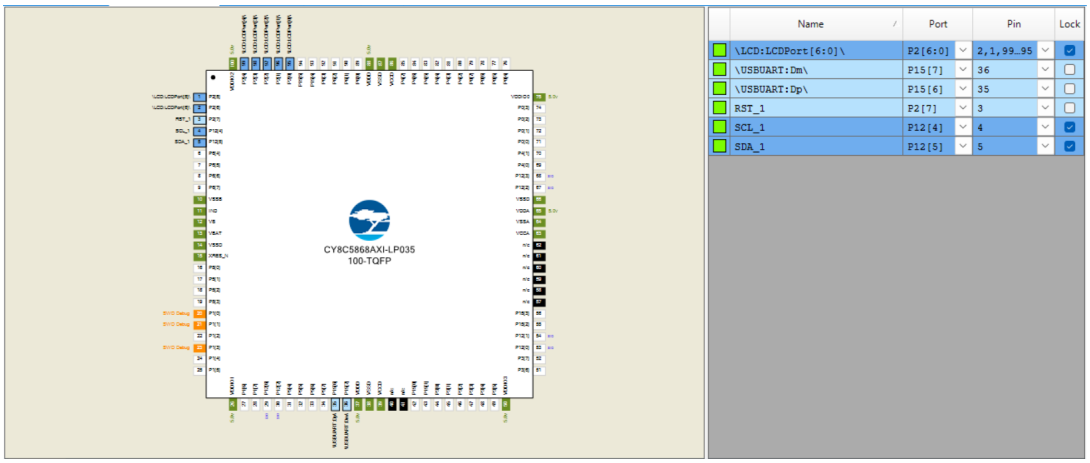


Figure 2: Pin Out for Interfacing TCS34725 Color Sensor with PSoC Microcontroller

5. Procedure

1. Setup I2C communication between PSoC and the TCS34725 sensor.
2. Initialize sensor by enabling power and ADC.
3. Configure integration time and gain.
4. Continuously read red, green, blue, and clear data registers.

5. Convert raw values to identify the dominant color.

6. Display values on an LCD and send them to a USB terminal.

6. Code Snippet

```
#include "project.h"
#include <stdio.h>

#define TCS34725_ADDRESS  0x29  // 7-bit I2C address of the TCS34725 sensor
#define TCS34725_COMMAND_BIT 0x80

// Register addresses
#define TCS34725_ENABLE    0x00
#define TCS34725_ATIME     0x01
#define TCS34725_CONTROL   0x0F
#define TCS34725_ID        0x12
#define TCS34725_CDATA     0x14

void I2C_WriteByte(uint8_t reg, uint8_t value) {
    uint8_t buffer[2] = {TCS34725_COMMAND_BIT | reg, value};
    I2C_MasterWriteBuf(TCS34725_ADDRESS, buffer, 2, I2C_MODE_COMPLETE_XFER);
    while ((I2C_MasterStatus() & I2C_MSTAT_WR_CMPLT) == 0);
}

uint16_t I2C_ReadWord(uint8_t reg) {
    uint8_t buffer[2];
    uint8_t regAddr = TCS34725_COMMAND_BIT | reg;
    I2C_MasterWriteBuf(TCS34725_ADDRESS, &regAddr, 1, I2C_MODE_COMPLETE_XFER);
    while ((I2C_MasterStatus() & I2C_MSTAT_WR_CMPLT) == 0);

    I2C_MasterReadBuf(TCS34725_ADDRESS, buffer, 2, I2C_MODE_COMPLETE_XFER);
    while ((I2C_MasterStatus() & I2C_MSTAT_RD_CMPLT) == 0);

    return (buffer[1] << 8) | buffer[0];
}

void TCS34725_Init() {
    I2C_Start();
    CyDelay(3);
    I2C_WriteByte(TCS34725_ENABLE, 0x03);  // Power ON and enable ADC
    I2C_WriteByte(TCS34725_ATIME, 0xFF);   // Maximum integration time
    I2C_WriteByte(TCS34725_CONTROL, 0x01); // Set gain to 4x
}

void Read_RGB_Values(uint16_t *r, uint16_t *g, uint16_t *b) {
    *r = I2C_ReadWord(TCS34725_CDATA + 2);
```

```
    *g = I2C_ReadWord(TCS34725_CDATA + 4);
    *b = I2C_ReadWord(TCS34725_CDATA + 6);
}

void Display_RGB_Values(uint16_t r, uint16_t g, uint16_t b) {
    char buffer[16];
    LCD_ClearDisplay();

    snprintf(buffer, sizeof(buffer), "R:%u G:%u", r, g);
    LCD_Position(0, 0);
    LCD_PrintString(buffer);

    snprintf(buffer, sizeof(buffer), "B:%u", b);
    LCD_Position(1, 0);
    LCD_PrintString(buffer);
}

void USB_Send_RGB_Values(uint16_t r, uint16_t g, uint16_t b) {
    char usbBuffer[64];
    snprintf(usbBuffer, sizeof(usbBuffer), "R:%u, G:%u, B:%u\r\n", r, g, b);

    if (USBUART_GetConfiguration()) {
        USBUART_PutString(usbBuffer);
    }
}



int main(void) {
    CyGlobalIntEnable;
    I2C_Start();
    LCD_Start();
    USBUART_Start(0, USBUART_5V_OPERATION);
    TCS34725_Init();

    uint16_t red, green, blue;

    for (;;) {
        if (USBUART_IsConfigurationChanged()) {
            USBUART_CDC_Init();
        }

        Read_RGB_Values(&red, &green, &blue);
        Display_RGB_Values(red, green, blue);
        USB_Send_RGB_Values(red, green, blue);


        CyDelay(500);
```

```
    }
}
```

7. MATLAB Code

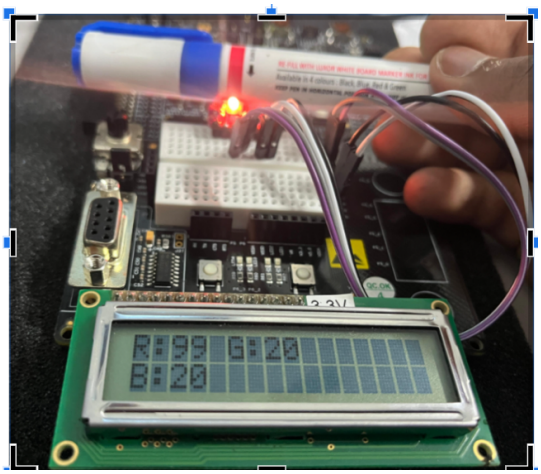
```
clc;
clear;

% --- Serial Port Configuration ---

% Define the serial port (adjust 'COM8' to match your actual port)
serialPort = 'COM8';
baudRate = 115200;  % Set the baud rate to match the transmitting device

% Open and configure the serial port
s = serialport(serialPort, baudRate);
s.Timeout = 10;  % Set timeout duration in seconds to avoid read issues

% --- Plot Setup ---

% Create a figure with 4 vertically stacked plots
figure;
t = tiledlayout(4, 1);  % 4 rows, 1 column layout

% Prepare individual tiles for plotting
tile1 = nexttile;
tile2 = nexttile;
tile3 = nexttile;
tile4 = nexttile;

% Keep plots on each tile active for dynamic updates
hold(tile1, 'on');
hold(tile2, 'on');
hold(tile3, 'on');
hold(tile4, 'on');

% Initialize data buffers for live plotting
timeWindow = 100;  % Number of samples to keep/display in the sliding window
r_data = zeros(1, timeWindow);  % Red channel data buffer
g_data = zeros(1, timeWindow);  % Green channel data buffer
b_data = zeros(1, timeWindow);  % Blue channel data buffer
x = 1:timeWindow;  % X-axis representing sample index

% Plot Red channel
h1 = plot(tile1, x, r_data, 'r', 'LineWidth', 2);
```

```
title(tile1, 'Red Channel');
ylabel(tile1, 'Intensity');
ylim(tile1, [0, 255]);

% Plot Green channel
h2 = plot(tile2, x, g_data, 'g', 'LineWidth', 2);
title(tile2, 'Green Channel');
ylabel(tile2, 'Intensity');
ylim(tile2, [0, 255]);

% Plot Blue channel
h3 = plot(tile3, x, b_data, 'b', 'LineWidth', 2);
title(tile3, 'Blue Channel');
ylabel(tile3, 'Intensity');
xlabel(tile3, 'Time');
ylim(tile3, [0, 255]);

% Plot for visualizing the resulting color
h4 = fill(tile4, [0 1 1 0], [0 0 1 1], [0 0 0], 'EdgeColor', 'none');
title(tile4, 'Current Color');
axis(tile4, [0 1 0 1]);

disp('Reading RGB values from serial port...');

% --- Main Loop ---

while true
    if s.NumBytesAvailable > 0
        try
            % Read a full line of data from the serial port
            data = readline(s);

            % Extract numeric RGB values using regular expressions
            tokens = regexp(data, 'R:(\d+), G:(\d+), B:(\d+)', 'tokens');
            if ~isempty(tokens)
                values = str2double(tokens{1});  % Convert string tokens to num value

                % Display the received RGB values in the console
                fprintf('R: %d, G: %d, B: %d\n', values(1), values(2), values(3));

                % Update the sliding data window with new values
                r_data = [r_data(2:end), values(1)];
                g_data = [g_data(2:end), values(2)];
                b_data = [b_data(2:end), values(3)];

                % Update the plots with the latest data
                set(h1, 'YData', r_data);
                set(h2, 'YData', g_data);
```

```
                set(h3, 'YData', b_data);

                % Update the color display rectangle with current RGB color
                set(h4, 'FaceColor', [values(1)/255, values(2)/255, values(3)/255]);
                drawnow;  % Force plot update
            end
        catch ME
            % Print error message in case of any exception during read/parse
            fprintf('Error reading data: %s\n', ME.message);
        end
    else
        pause(0.1);  % Brief pause to avoid excessive CPU usage
    end
end
```

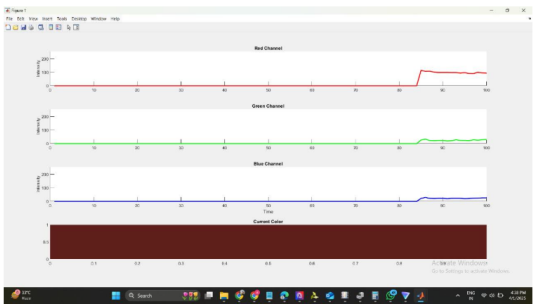
8. Observations

- Sensor was initialized successfully with proper ID verification.
- RGB values change under different lighting and colored surfaces.
- LCD displayed real-time color readings.
- Dominant color detection logic worked as expected.
- USBUART terminal received formatted color readings.

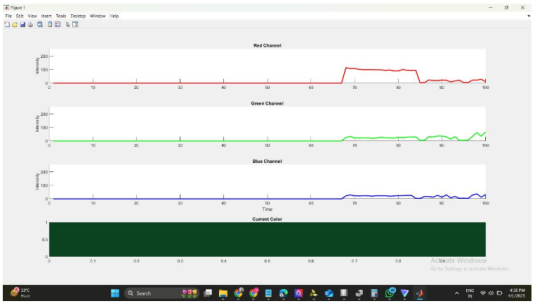


(a) Red Object detection

(b) Green Object detection



(c) MATLAB Plot for red object



(d) MATLAB Plot for green object

Figure 3: Comparison of real objects and corresponding MATLAB plots

9. Results

The TCS34725 sensor was successfully interfaced with the PSoC using I2C. RGB and clear light intensities were read, displayed, and transmitted over USB and displayed on MATLAB. Color detection was implemented and tested successfully.

10. Applications

- Color-based object sorting systems
- Ambient light monitoring
- Color quality control in manufacturing
- Smart farming and agricultural diagnosis

- Interactive art installations

11. Future Work

- Implement real-time color graphing on a PC interface.
- Integrate with IoT platforms for cloud-based logging.
- Combine with proximity sensors to adjust reading distance.
- Expand to full spectrum sensors for wider applications.

12. Conclusion

This lab demonstrated successful I2C interfacing between the TCS34725 color sensor and a PSoC microcontroller. The implementation covered sensor initialization, data acquisition, display, and communication. This system forms the foundation for many practical color sensing applications in the fields of robotics, automation, and IoT.