CSUN | COLLEGE OF ENGINEERING AND COMPUTER SCIENCE

ECE 528/L

Robotics and Embedded Systems with Lab

Final Project
Simon Say's Color Code
Lab Report

Leo Datta
Instructor: Aaron Nanas
Fall 2025

INTRODUCTION

This program serves as the main control logic for a color-sensing system built around the Digilent PMOD Color module and the MSP432 LaunchPad. Using the TCS34725 color sensor over the I²C interface, the application reads raw RGBC data, calibrates it, and classifies the detected color in real time. The system generates a random 4-colored sequence of 3-colors, displays that pattern to the user, and then evaluates each detected input to determine whether it matches the required sequence. Alongside color processing, the program manages onboard LEDs, motor movement, and timed events through SysTick interrupts, allowing the device to respond to correct inputs, incorrect attempts, and collision events. Overall, this code integrates sensing, actuation, and feedback into a cohesive pattern-matching and response system.

COMPONENTS USED

***Table 1**: Components used in Lab5 GPIO*

COMPONENT DESCRIPTION	Qty
MSP432 LaunchPad	1
USB-A to Micro-USB Cable	1
TI-RSLK MAX Chassis	1
PMOD Color Sensor Diligent	1

ANALYSIS RESULTS

MSP432 <--> Color Sensor Pinout Table

Signal	MSP432 Pin	Direction	Voltage	Description
SDA	P6.4	I/O	3.3V	I2C data line
SCL	P6.5	Output	3.3V	I2C clock
INT	—	Input	3.3V	Interrupt from sensor
EN	P8.3	Output	3.3V	Sensor enable
VDD	—	—	3.3V	Power
GND	—	—	0V	Ground

***Table 1:** Pinout between MSP432 board and PMOD Color Sensor*

To ensure reliable detection, the raw RGBC values from the PMOD Color sensor were closely monitored and calibrated before use. The system captured repeated samples at 50 ms intervals, allowing the calibration routine to normalize variations caused by lighting, sensor drift, or surface reflections. By observing the numerical ranges of red, green, and blue outputs in real time, we verified that each color produced a consistent and distinguishable signature. Thresholds were then tuned so that green, red, and yellow could be cleanly separated, while unknown or noisy readings were safely ignored. This validation process confirmed that the

sensor data was stable enough for pattern recognition, with minimal overlap between color clusters and predictable behavior across multiple trials.

Once the color values were verified, the detection logic implemented a structured sequence-recognition system. The program generates a random four-color pattern and displays it to the user with timed LED outputs. During operation, each measured color is passed through the detection function and then held briefly to eliminate flicker or noise. The Check Pattern routine compares the detected color to the expected position in the sequence, advancing on correct inputs, granting access on full completion, or enforcing a reset after repeated mistakes. The system provides immediate feedback through LEDs and motor movement, creating a clear cycle of showing the pattern, waiting for user input, validating each step, and responding accordingly. This process demonstrated consistent and predictable performance, confirming the effectiveness of the detection thresholds and the overall pattern-matching design.

```
260
261 void Generate_Random_Pattern(void)
262 {
263     for (int i = 0; i < PATTERN_LENGTH; i++)
264     {
265         pattern[i] = rand() % 3;    // 0 = green, 1 = red, 2 = yellow
266     }
267 }
268
269
270 void Show_Pattern(void)
271 {
272     for (int i = 0; i < PATTERN_LENGTH; i++)
273     {
274         switch(pattern[i])
275         {
276             case COLOR_GREEN:
277                 LED2_Output(RGB_LED_GREEN);
278                 break;
279
280             case COLOR_RED:
281                 LED2_Output(RGB_LED_RED);
282                 break;
283
284             case COLOR_YELLOW:
285                 LED2_Output(RGB_LED_YELLOW);
286                 break;
287         }
288
289         Clock_Delay1ms(700);  // hold the color
290         LED2_Output(RGB_LED_OFF);
291         Clock_Delay1ms(300);  // gap between colors
292     }
293 }
```

Figure 1: *Generate_Random_Pattern() & Show_Pattern() functions in main.c*

```
213
214 Color_t Detect_Color(uint16_t R, uint16_t G, uint16_t B)
215 {
216     // ---- GREEN ----
217     if (G > R + 3000 && G > B + 3000)
218     {
219         printf("GREEN\n");
220         LED2_Output(RGB_LED_GREEN);
221         return COLOR_GREEN;
222     }
223
224     // ---- YELLOW ----
225     else if (R > 0x2000 && G > 0x2000 && B < 0x3000)
226     {
227         printf("YELLOW\n");
228         LED2_Output(RGB_LED_YELLOW);
229         return COLOR_YELLOW;
230     }
231
232     // ---- RED ----
233     else if (R > G + 6000 && R > B + 6000)
234     {
235         printf("RED\n");
236         LED2_Output(RGB_LED_RED);
237         return COLOR_RED;
238     }
239     else
240     {
241         LED2_Output(RGB_LED_OFF);
242         return COLOR_UNKNOWN;
243     }
244 }
245
```

```
243
244 Color_t Hold_Color(uint16_t R, uint16_t G, uint16_t B)
245 {
246     Color_t color = Detect_Color(R, G, B);   // your existing color logic
247
248     if (color != COLOR_UNKNOWN)
249     {
250         // Hold the detected color for 1 second
251         Clock_Delay1ms(1000);
252
253         return color;  // return the locked-in color
254     }
255
256     return COLOR_UNKNOWN;
257 }
```

Figure 2: *Detect_Color() & Hold_Color() functions in main.c*

```c
int main(void)
{
    // Ensure that interrupts are disabled during initialization
    DisableInterrupts();

    // Initialize the 48 MHz Clock
    Clock_Init48MHz();

    //Initialize GPIO
    LED2_Init();
    Buttons_Init();

    //Initialize Timer & Motor
    Timer_A0_PWM_Init(TIMER_A0_PERIOD_CONSTANT, 0, 0);
    Motor_Init();

    // Initialize the SysTick timer to generate periodic interrupts every 1 ms
    SysTick_Interrupt_Init(SYSTICK_INT_NUM_CLK_CYCLES, SYSTICK_INT_PRIORITY);

    // Initialize EUSCI_A0_UART
    EUSCI_A0_UART_Init_Printf();

    // Initialize the PMOD Color module
    PMOD_Color_Init();

    // Indicate that the PMDO Color module has been initialized and powered on
    printf("PMOD COLOR has been initialized and powered on.\n");

    // Enable the interrupts used by the modules
    EnableInterrupts();

    // Display the PMOD Color Device ID
    printf("PMOD Color Device ID: 0x%02X\n", PMOD_Color_Get_Device_ID());

    // Declare structs for both raw and normalized PMOD Color data
    PMOD_Color_Data pmod_color_data;
    PMOD_Calibration_Data calibration_data;

    pmod_color_data = PMOD_Color_Get_RGBC();
    calibration_data = PMOD_Color_Init_Calibration_Data(pmod_color_data);
    Clock_Delay1us(2400);

    Generate_Random_Pattern();
    Show_Pattern();

```

Figure 3: *main() functions in main.c*

```c
while(1)
{
    // The on-board LED on the PMOD COLOR module can be controlled using the PMOD_Color_LED_Control function
    // Uncomment the line below if you'd like to see the on-board LED
    PMOD_Color_LED_Control(PMOD_COLOR_ENABLE_LED);

    // Sample the PMOD COLOR sensor every 50 ms
    pmod_color_data = PMOD_Color_Get_RGBC();
    PMOD_Color_Calibrate(pmod_color_data, &calibration_data);
    pmod_color_data = PMOD_Color_Normalize_Calibration(pmod_color_data, calibration_data);
    printf("r=%04x g=%04x b=%04x\r\n", pmod_color_data.red, pmod_color_data.green, pmod_color_data.blue);
    Clock_Delay1ms(50);


    uint16_t R = pmod_color_data.red;
    uint16_t G = pmod_color_data.green;
    uint16_t B = pmod_color_data.blue;

    Color_t detect = Hold_Color(R, G, B);
```

Figure 4: *Polling Method in main() function*

```
295 int CheckPattern(Color_t detected)
296 {
297     static int index = 0;
298     static int failCount = 0; // counts consecutive failures
299
300     if (detected == COLOR_UNKNOWN)
301         return -1;  // ignore noise completely
302
303     // ---------- CORRECT COLOR ----------
304     if (detected == pattern[index])
305     {
306         failCount = 0;    // reset failure counter
307         index++;
308
309         if (index == PATTERN_LENGTH)
310         {
311             index = 0;
312             return 2;     // full pattern matched
313         }
314         return 1; // correct so far
315     }
316
317     // ---------- WRONG COLOR ----------
318     else
319     {
320         failCount++;
321
322         if (failCount >= 2)   // only fail after 2 bad reads in a row
323         {
324             index = 0;
325             failCount = 0;
326             return 0;   // full failure: restart needed
327         }
328
329         return -1;   // mild failure → IGNORE (do not restart)
330     }
331 }
332
```

Figure 5: *CheckPattern() in main() function*

```
int result = CheckPattern(detect);

if (result == 1)
{
    printf("Correct step!\n");
    LED2_Output(RGB_LED_WHITE);
    Clock_Delay1ms(500);
    LED2_Output(RGB_LED_OFF);
```

```
175         else if (result == 2)
176         {
177             printf("ACCESS GRANTED!\n");
178             LED2_Output(RGB_LED_SKY_BLUE);
179             Clock_Delay1ms(3000);
180             LED2_Output(RGB_LED_OFF);
181
182             Motor_Forward(4500, 4500);
183             Clock_Delay1ms(2000);
184             Motor_Backward(4500, 4500);
185             Clock_Delay1ms(2000);
186             Motor_Stop();
187
188             Generate_Random_Pattern();
189             Show_Pattern();
190         }
```

```
191         else if (result == 0)
192         {
193             printf("Wrong! Restarting...\n");
194             LED2_Output(RGB_LED_PINK);
195             Clock_Delay1ms(2500);
196             LED2_Output(RGB_LED_OFF);
197 
198             Clock_Delay1ms(500);
199             Motor_Left(4500, 4500);
200             Clock_Delay1ms(2000);
201             Motor_Right(4500, 4500);
202             Clock_Delay1ms(2000);
203             Motor_Stop();
204 
205             Show_Pattern();
206         }
207 
```

Figure 6: *Result Checker in the while loop of Fig. 4*

CONCLUSIONS

Overall, the system demonstrated a reliable and responsive integration of color sensing, pattern recognition, and motor control. Through careful calibration and validation of the PMOD Color sensor, the program was able to distinguish between colors with consistent accuracy, forming the foundation for a stable detection process. The sequence-generation and pattern-checking logic functioned as intended, providing immediate visual and mechanical feedback that reinforced correct actions and handled errors gracefully. By combining sensing, decision-making, and actuation into one cohesive loop, the project proved effective both as a functional demonstration of embedded color detection and as a robust interactive system capable of responding intelligently to user input.