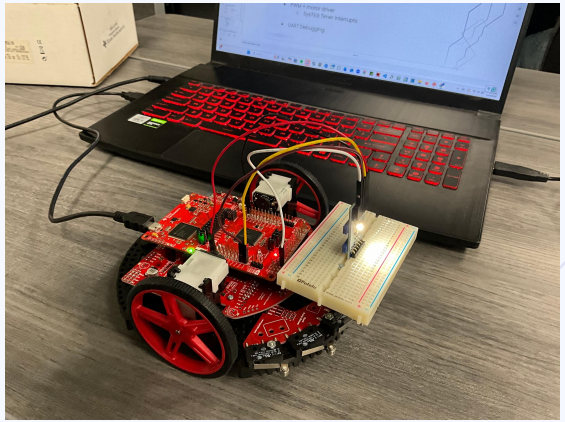
ECE 528 Final Project

Leo Datta
Professor Nanas

Simon Says Color Code Game

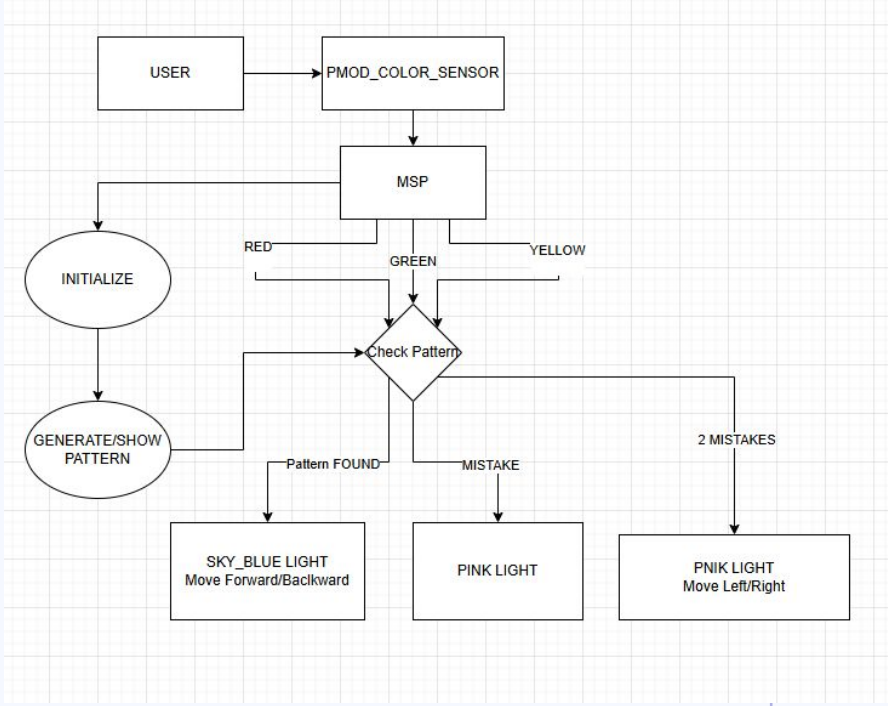
- It shows a random 4-color sequence, and the user must scan the same colors in the same order using real objects and the PMOD Color Sensor.

- The MSP432 checks each input and gives success or failure feedback through LEDs and motors



Main Aspects Overview

- I2C Color Sensing and Calibration
- Random Pattern Generation
- GPIO for LEDs and buttons
- PWM + motor driver
    - SysTick Timer Interrupts
- UART Debugging



PMOD_Color_Init()

- Initializes the I2C hardware and powers on the color sensor
- Controls the sensor's onboard LED
- Reads individual registers or full 8-byte RGBC data
- Provides calibration tools (tracks min/max)
- Normalizes RGB values so detection is stable in different lighting

```
95
96 PMOD_Color_Data PMOD_Color_Get_RGBC()
97 {
98     PMOD_Color_Data data;
99     uint8_t color_buffer[8];
100
101     EUSCI_B1_I2C_Send_A_Byte(PMOD_COLOR_ADDRESS, PMOD_COLOR_AUTO_INC | PMOD_COLOR_CDATA_L_REG);
102
103     EUSCI_B1_I2C_Receive_Multiple_Bytes(PMOD_COLOR_ADDRESS, color_buffer, 8);
104
105     data.clear = (color_buffer[1] << 8) | color_buffer[0];
106     data.red = (color_buffer[3] << 8) | color_buffer[2];
107     data.green = (color_buffer[5] << 8) | color_buffer[4];
108     data.blue = (color_buffer[7] << 8) | color_buffer[6];
109
110     return data;
111 }
112
113 uint8_t PMOD_Color_Read_Raw_Color_Data(uint8_t register_address)
114 {
115     uint8_t PMOD_Color_Byte = PMOD_Color_Read_Register(PMOD_COLOR_AUTO_INC | register_address);
116     return PMOD_Color_Byte;
117 }
118
119 PMOD_Calibration_Data PMOD_Color_Init_Calibration_Data(PMOD_Color_Data first_sample)
120 {
121     PMOD_Calibration_Data calibration_data;
122
123     calibration_data.min = first_sample;
124     calibration_data.max = first_sample;
125
126     return calibration_data;
127 }
128
```

Random Pattern Generation

- Generate_Random_Pattern()
  - Fills pattern array with random values 0-2
  - Made an enum where GREEN =0, RED = 1, YELLOW = 2, UNKNOWN = 3 ,
- Show_Pattern()
  - Plays Back Sequence one color at a time
  - Hold its for 0.7 ms for the user then switches after 0.3 ms

```
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 }
```

Detecting/Holding Color

- Real-world light readings are not perfect.
  - Red objects sometimes produced high Green values
  - Yellow was too close to bright Green
  - Green readings varied depending on distance from sensor
- Needed to tune thresholds
- Holding to prevent flickering between colors

```
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 }
```

Main() Inits & Polling

```
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();
```

```
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);
```

Checking Pattern

- Takes the detected color and compares it to the next color in the pattern

- If it matches → move to next spot in the code
    - If all colors match → return 2 (sequence completed)
    - If the color is wrong → count the mistake

- Two mistakes in a row → return 0 (restart)

```
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
```

Correct, Success & Failure Response

```
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         }
```

THANK YOU