微處理機系統實習　　Lab6

班級：資訊三乙 學號：D1210799 姓名：王建葦

一、【實驗目的】：

What was your design? What were the concepts you have used for your design?

本次實驗包含兩個部分，分別著重於 LCD 顯示控制、隨機數生成、LED／Buzzer 整合 與交通號誌模擬：

- Lab6.1 — 數字選擇與加總系統
  使用 LCD、Keypad、LED 及蜂鳴器設計互動式加總系統。
  以亂數產生 4 個兩位數（10~99）顯示於 LCD。
  使用方向鍵（↑、↓）移動游標選擇數字，按下 S 鍵 可將該數字加入總和。
  LED 會依已選數量亮起；B 鍵可回刪上一次選擇，C 鍵清除總和。
  R 鍵重新生成亂數並重設狀態。
  主要概念為 手動亂數種子生成 (Linear Congruential Generator)、LCD 資料動態更新、Keypad 事件觸發與防抖動 (debounce) 以及 多輸入狀態管理。

- Lab6.2 — 智慧交通號誌模擬系統
  利用 LED、蜂鳴器、七段顯示器及 LCD 模擬紅綠燈運作：
  系統開機時黃燈閃爍、LCD 顯示「STOP」圖示；
  按下 5 鍵（GO） 啟動號誌循環：
  依序進行「車綠→車黃→全紅→行人綠→全紅」，LCD 與七段顯示器同步顯示狀態與倒數秒數。
  每次進入新階段時會更新 LED、LCD 圖案，並搭配蜂鳴器提示。
  實驗重點為 多狀態機控制 (Finite State Machine)、時間倒數顯示 (Timer Counter)、影像緩衝繪製 (Frame Buffer) 與 LCD 位元圖操作 (Bitmap Rendering)。

二、【遭遇的問題】：

What problems you faced during design and implementation?

1. 在 Lab6.1 中，若多次按鍵未釋放，會造成蜂鳴器重複觸發或 LCD 顯示閃爍。
2. 亂數生成初期使用 rand() 結果固定，導致四個數字重複。
3. 在加總功能中，游標移動超過範圍時顯示會錯位。
4. Lab6.2 的號誌系統中，初期黃燈閃爍與序列倒數同時運作時畫面閃爍嚴重。
5. LCD 上顯示的 STOP／GO 圖示若未同步更新，會殘留前一幀圖形。

三、【解決方法】：

How did you solve the problems?

1. 使用「按鍵釋放偵測」（Key-release Detection）機制，只在 放開按鍵時 執行動作，避免蜂鳴器重複響應。
2. 實作自製亂數函式 my_rand() 搭配 my_srand(count)，以系統迴圈變數為種子，確保每次產生不同亂數。
3. 對游標移動範圍設定上限（0 - 3）並以 view_offset 控制 LCD 顯示視窗。

4. 在 Lab6.2 中以 sequence_active 與 blink_state 分離兩種模式（閃爍狀態與運作狀態），避免 LED 與 LCD 同步衝突。
5. 將 LCD 更新封裝為 UpdateLCDDisplay()，每次更新前清除畫面 (clear_LCD())，再以 copy_bitmap_to_buffer() 將圖像繪入暫存緩衝區後再一次性顯示，確保畫面穩定。
6. 七段顯示器在倒數時只於每秒更新一次，減少 CPU 負載並同步顯示數值。

在找尋這些問題的解決方法與問題點時，我有使用 ChatGPT 協助我找尋與解決問題。包含 實驗結報的內容修改與潤飾都有使用 ChatGPT 協助。

四、【未能解決的問題】：

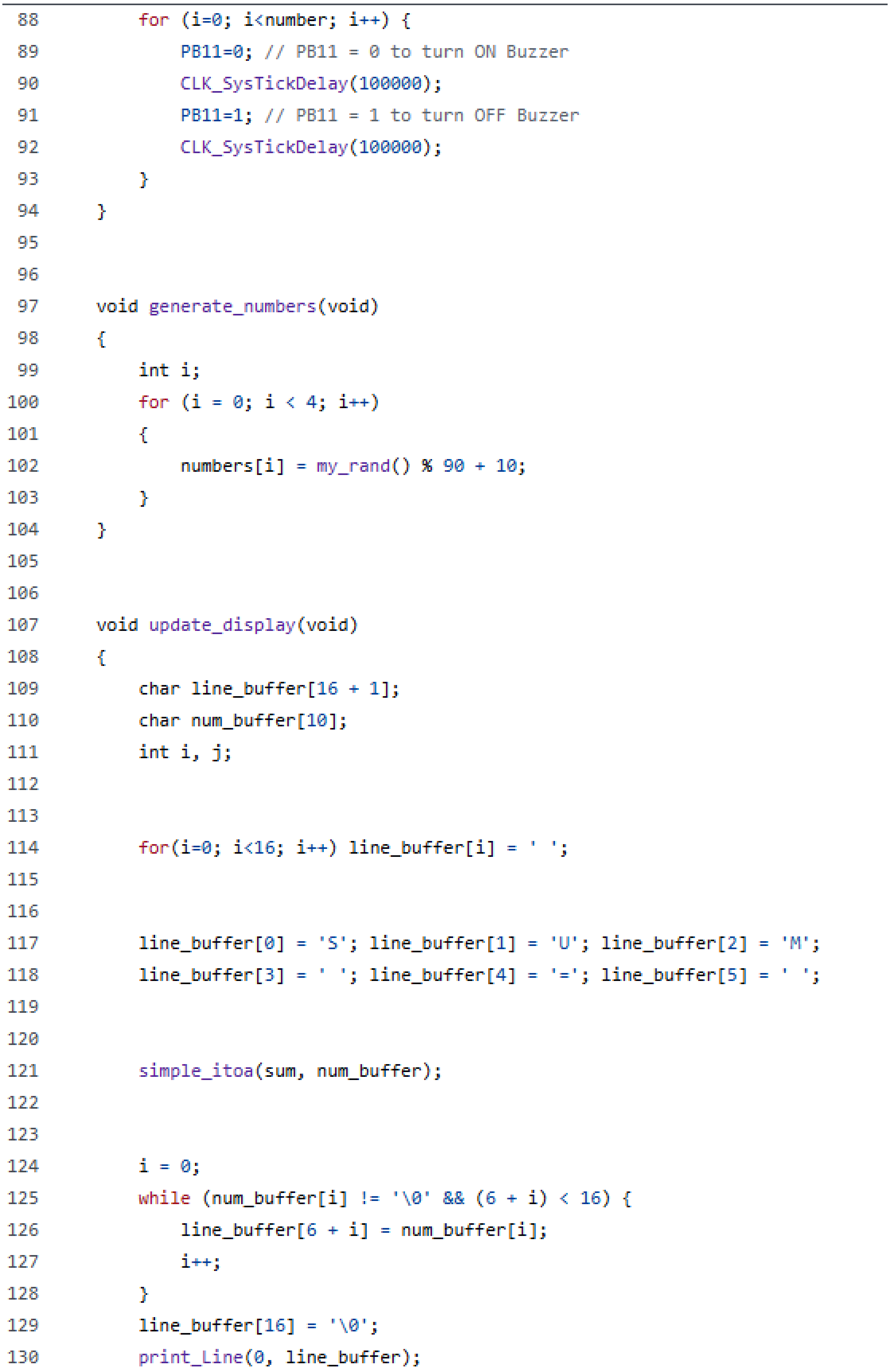
Was there any problem that you were unable to solve? Why was it unsolvable?

1. LCD 的畫面更新仍有極輕微閃爍，推測與 draw_LCD() 傳輸時間與主迴圈更新頻率不同步有關。
2. 在長時間運作下，Lab6.1 的亂數有時仍出現重複值，因為自製亂數產生式受限於簡化線性同餘公式。
3. Traffic sequence 若中途按鍵被干擾，可能會提早或延後狀態切換，尚未實作中斷式定時器改善。

五、【程式碼】：

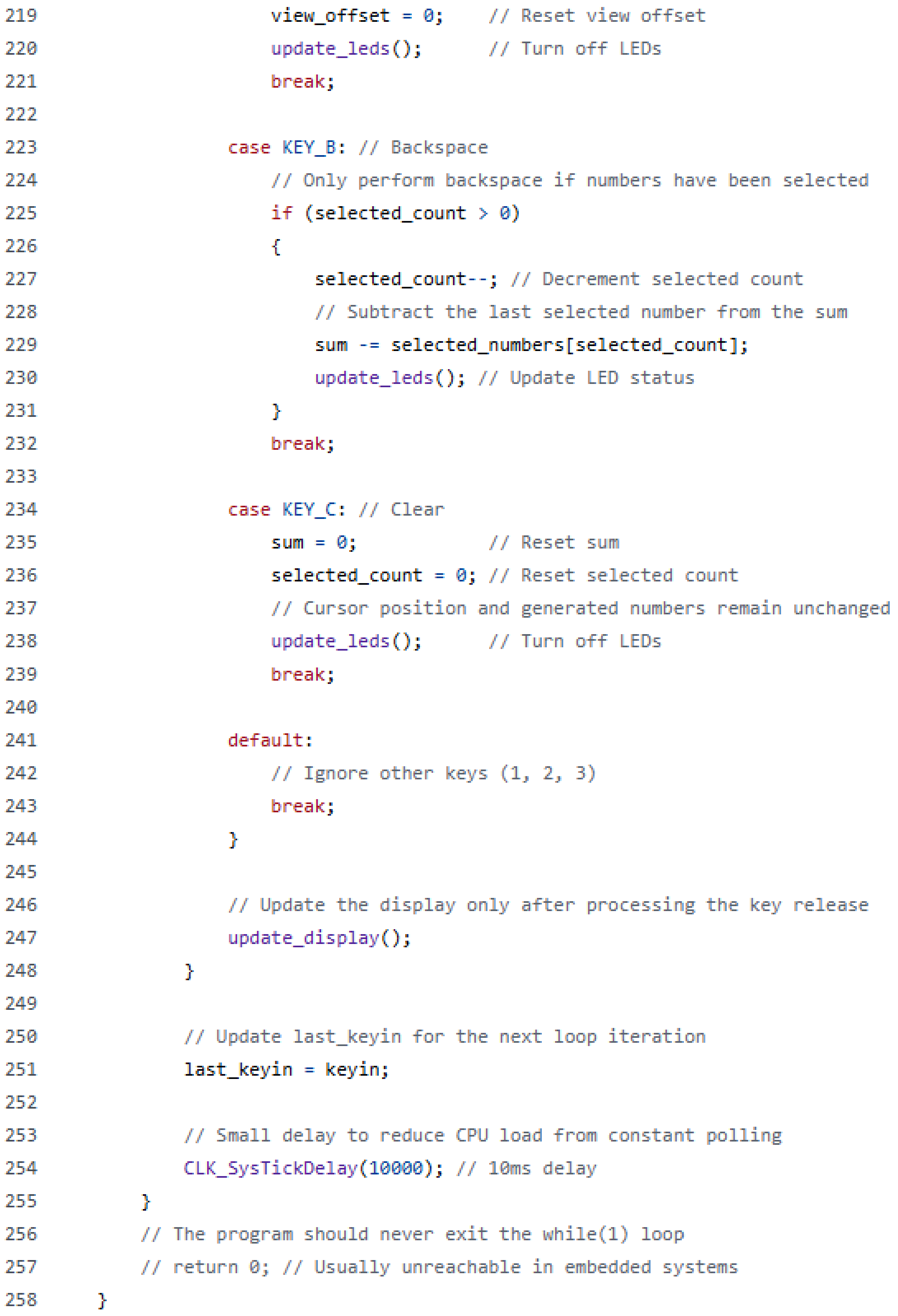
Lab 6.1:

```c
#include <string.h>
#include "NUC100Series.h"
#include "MCU_init.h"
#include "SYS_init.h"
#include "LCD.h"
#include "Scankey.h"
#include "clk.h"

#define KEY_UP 4
#define KEY_DOWN 6
#define KEY_S 5
#define KEY_R 7  // R?
#define KEY_B 8
#define KEY_C 9

static uint32_t g_seed; // ???????
void my_srand(uint32_t seed) { g_seed = seed; }
int my_rand(void)
{
    // ?? g_seed ? 0 (?????),??????????
    if (g_seed == 0) g_seed = 1;

    g_seed = (1103515245 * g_seed + 12345) & 0x7FFFFFFF; // Standard LCG values
    return (int)g_seed;
}

void simple_itoa(int val, char *buf)
{
    int i = 0;
    char temp[10];
    int j = 0;

    // Handle 0 explicitly
    if (val == 0) {
        buf[0] = '0';
        buf[1] = '\0';
        return;
    }

    while (val > 0) {
        temp[j++] = (val % 10) + '0';
        val /= 10;
    }
    while (j > 0) {
        buf[i++] = temp[--j];
    }

    // Null-terminate the string
    buf[i] = '\0';
}

int numbers[4];
int sum = 0;
int cursor_pos = 0;
int view_offset = 0;
int selected_numbers[4];
int selected_count = 0;


void init_leds(void)
{
    GPIO_SetMode(PC, BIT12 | BIT13 | BIT14 | BIT15, GPIO_PMD_OUTPUT);
    PC12 = 1; PC13 = 1; PC14 = 1; PC15 = 1; // LEDs off (assuming active-low)
}


void update_leds(void)
{
    // LEDs turn on based on the number of selected items (active-low)
    PC12 = (selected_count > 0) ? 0 : 1;
    PC13 = (selected_count > 1) ? 0 : 1;
    PC14 = (selected_count > 2) ? 0 : 1;
    PC15 = (selected_count > 3) ? 0 : 1;
}


void init_buzzer(void)
{
    GPIO_SetMode(PB, BIT11, GPIO_PMD_OUTPUT);
    PB11 = 1;
}


void Buzz(int number)
{
    int i;
```



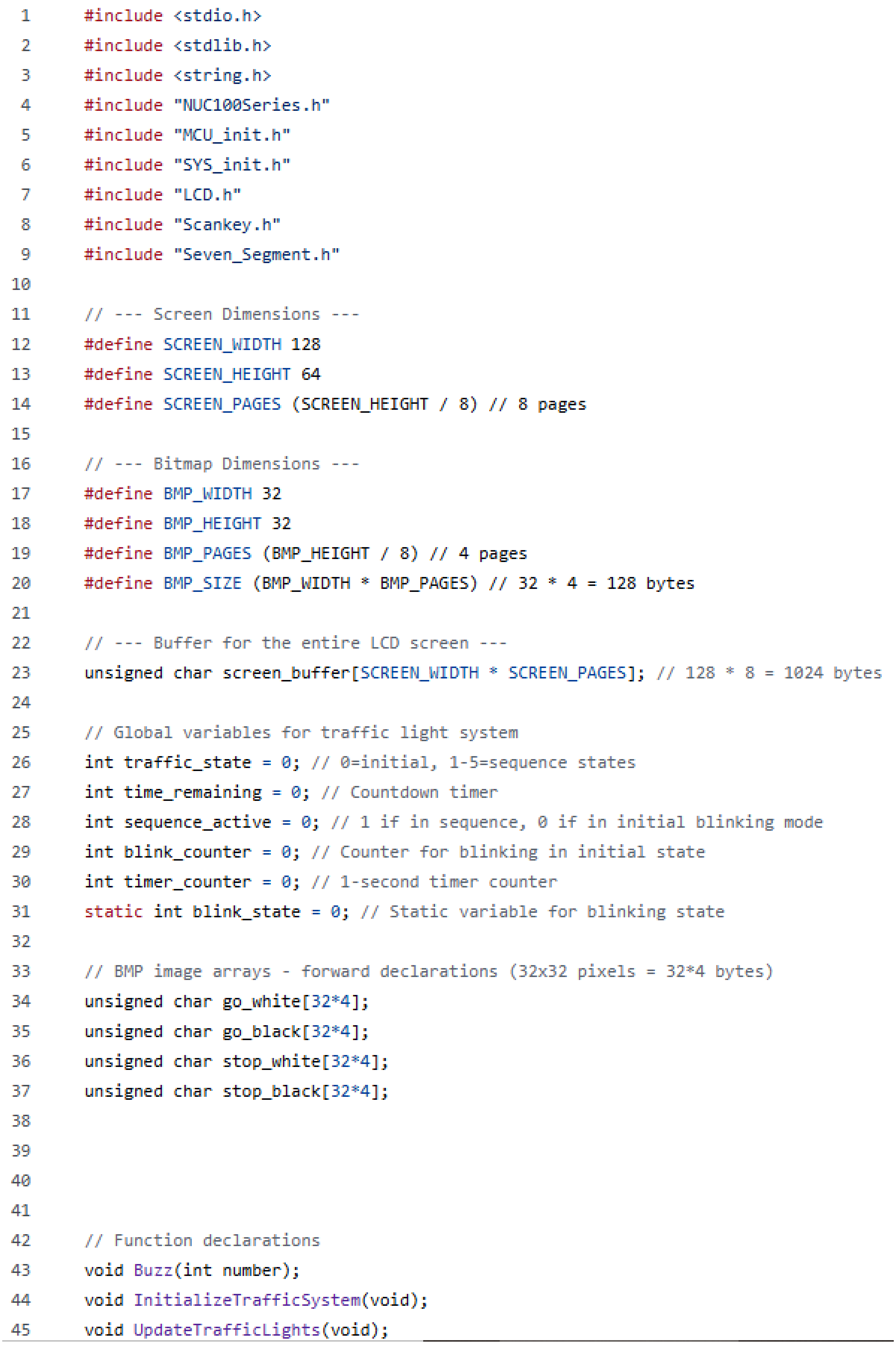
```
    for (i=0; i<number; i++) {
        PB11=0; // PB11 = 0 to turn ON Buzzer
        CLK_SysTickDelay(100000);
        PB11=1; // PB11 = 1 to turn OFF Buzzer
        CLK_SysTickDelay(100000);
    }
}


void generate_numbers(void)
{
    int i;
    for (i = 0; i < 4; i++)
    {
        numbers[i] = my_rand() % 90 + 10;
    }
}


void update_display(void)
{
    char line_buffer[16 + 1];
    char num_buffer[10];
    int i, j;


    for(i=0; i<16; i++) line_buffer[i] = ' ';


    line_buffer[0] = 'S'; line_buffer[1] = 'U'; line_buffer[2] = 'M';
    line_buffer[3] = ' '; line_buffer[4] = '='; line_buffer[5] = ' ';


    simple_itoa(sum, num_buffer);


    i = 0;
    while (num_buffer[i] != '\0' && (6 + i) < 16) {
        line_buffer[6 + i] = num_buffer[i];
        i++;
    }
    line_buffer[16] = '\0';
    print_Line(0, line_buffer);
    for (j = 0; j < 3; j++)
    {
        int num_index = j + view_offset;
        int has_cursor = (num_index == cursor_pos);

        for(i=0; i<16; i++) line_buffer[i] = ' ';

        line_buffer[0] = (has_cursor ? '>' : ' ');
        line_buffer[1] = ' '; // Space after cursor

        simple_itoa(numbers[num_index], num_buffer);

        i = 0;
        while (num_buffer[i] != '\0' && (2 + i) < 16) {
            line_buffer[2 + i] = num_buffer[i];
            i++;
        }
        line_buffer[16] = '\0';
        print_Line(j + 1, line_buffer);
    }
}

int main(void)
{
    uint8_t keyin;
    uint8_t last_keyin = 0;
    uint32_t count = 0;

    SYS_Init();
    init_LCD();
    clear_LCD();
    OpenKeyPad();
    init_leds();
    init_buzzer();

    generate_numbers();
    update_display();

    while (1)
    {
        count++;
        keyin = ScanKey();
```

```
            if (keyin == 0 && last_keyin != 0)
            {
                // Process the key that was just released (stored in last_keyin)
                switch (last_keyin)
                {
                case KEY_UP: // Up arrow
                    if (cursor_pos > 0)
                    {
                        cursor_pos--;
                        // Adjust view offset if cursor moves out of the 3 visible lines
                        view_offset = (cursor_pos == 3) ? 1 : 0;
                    }
                    break;

                case KEY_DOWN: // Down arrow
                    if (cursor_pos < 3)
                    {
                        cursor_pos++;
                        // Adjust view offset if cursor moves to the 4th item
                        view_offset = (cursor_pos == 3) ? 1 : 0;
                    }
                    break;

                case KEY_S: // Select
                    // Only add if less than 4 numbers are selected
                    if (selected_count < 4)
                    {
                        int selected_val = numbers[cursor_pos];
                        sum += selected_val;
                        // Store the selected number for Backspace functionality
                        selected_numbers[selected_count] = selected_val;
                        selected_count++;
                        update_leds(); // Update LED status
                        Buzz(1);       // Beep once on selection (when key is released)
                    }
                    break;

                case KEY_R: // Reset
                    my_srand(count); // Use the current count as the random seed
                    generate_numbers(); // Generate new random numbers
                    sum = 0;            // Reset sum
                    selected_count = 0; // Reset selected count
                    cursor_pos = 0;     // Reset cursor position
```



```
                view_offset = 0;    // Reset view offset
                update_leds();      // Turn off LEDs
                break;

            case KEY_B: // Backspace
                // Only perform backspace if numbers have been selected
                if (selected_count > 0)
                {
                    selected_count--; // Decrement selected count
                    // Subtract the last selected number from the sum
                    sum -= selected_numbers[selected_count];
                    update_leds(); // Update LED status
                }
                break;

            case KEY_C: // Clear
                sum = 0;            // Reset sum
                selected_count = 0; // Reset selected count
                // Cursor position and generated numbers remain unchanged
                update_leds();      // Turn off LEDs
                break;

            default:
                // Ignore other keys (1, 2, 3)
                break;
            }

            // Update the display only after processing the key release
            update_display();
        }

        // Update last_keyin for the next loop iteration
        last_keyin = keyin;

        // Small delay to reduce CPU load from constant polling
        CLK_SysTickDelay(10000); // 10ms delay
    }
    // The program should never exit the while(1) loop
    // return 0; // Usually unreachable in embedded systems
}
```

Lab 6.2:



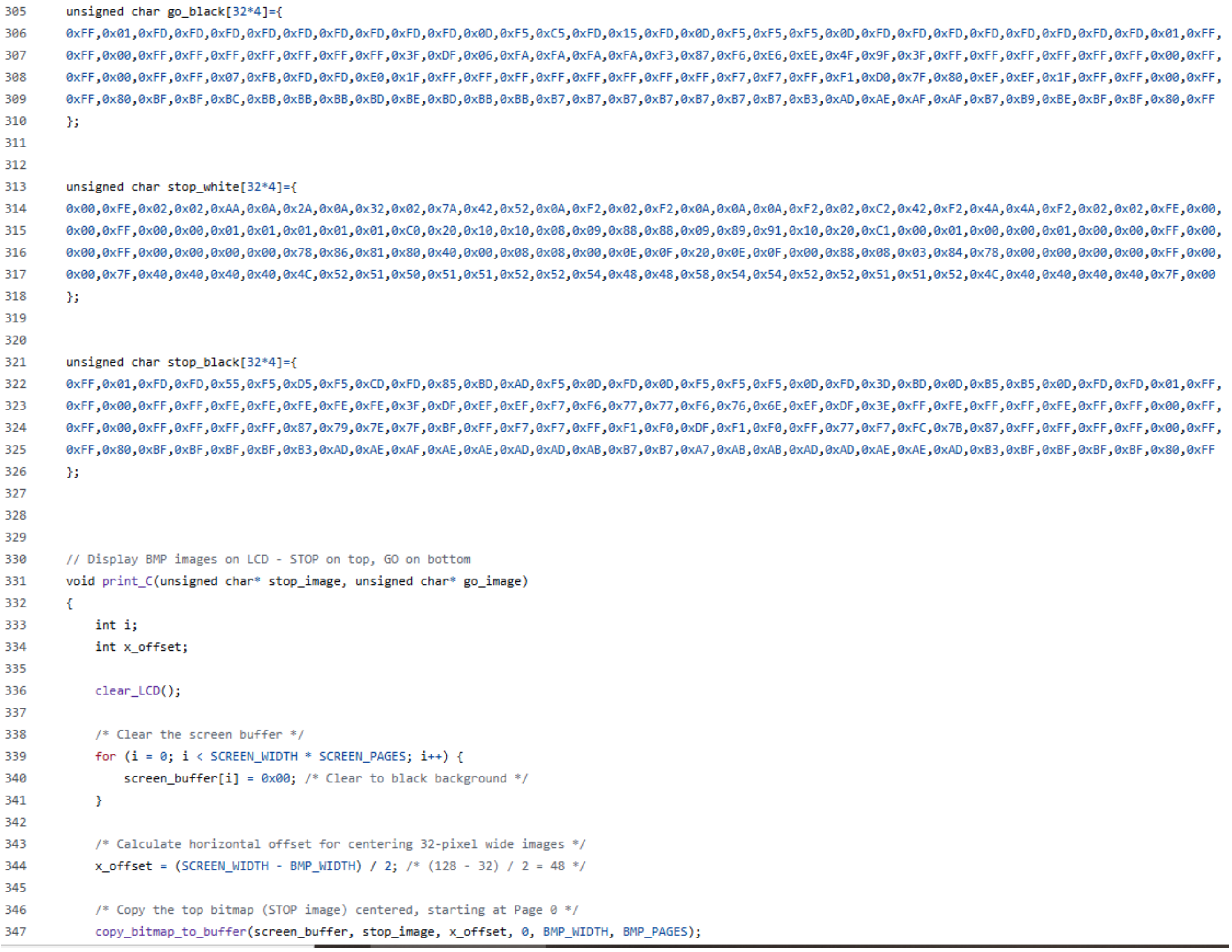
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "NUC100Series.h"
#include "MCU_init.h"
#include "SYS_init.h"
#include "LCD.h"
#include "Scankey.h"
#include "Seven_Segment.h"

// --- Screen Dimensions ---
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define SCREEN_PAGES (SCREEN_HEIGHT / 8) // 8 pages

// --- Bitmap Dimensions ---
#define BMP_WIDTH 32
#define BMP_HEIGHT 32
#define BMP_PAGES (BMP_HEIGHT / 8) // 4 pages
#define BMP_SIZE (BMP_WIDTH * BMP_PAGES) // 32 * 4 = 128 bytes

// --- Buffer for the entire LCD screen ---
unsigned char screen_buffer[SCREEN_WIDTH * SCREEN_PAGES]; // 128 * 8 = 1024 bytes

// Global variables for traffic light system
int traffic_state = 0; // 0=initial, 1-5=sequence states
int time_remaining = 0; // Countdown timer
int sequence_active = 0; // 1 if in sequence, 0 if in initial blinking mode
int blink_counter = 0; // Counter for blinking in initial state
int timer_counter = 0; // 1-second timer counter
static int blink_state = 0; // Static variable for blinking state

// BMP image arrays - forward declarations (32x32 pixels = 32*4 bytes)
unsigned char go_white[32*4];
unsigned char go_black[32*4];
unsigned char stop_white[32*4];
unsigned char stop_black[32*4];




// Function declarations
void Buzz(int number);
void InitializeTrafficSystem(void);
void UpdateTrafficLights(void);
void UpdateSevenSegment(void);
void UpdateLCDDisplay(void);
void StartTrafficSequence(void);
void ProcessTrafficTimer(void);
void SetVehicleLights(int red, int yellow, int green);
void SetPedestrianLights(int red, int green);
void print_C(unsigned char* stop_image, unsigned char* go_image);
void print_C_at_position(unsigned char* image, int start_page, int start_col);
void copy_bitmap_to_buffer(unsigned char* dest_buffer, const unsigned char* src_bitmap, int dest_x, int dest_y_page, int src_width, int src_height_pages);

void copy_bitmap_to_buffer(unsigned char* dest_buffer, const unsigned char* src_bitmap, int dest_x, int dest_y_page, int src_width, int src_height_pages) {
    int src_byte_idx = 0;
    int p, x; /* Declare loop variables for C89 */

    /* Loop through pages of source bitmap */
    for (p = 0; p < src_height_pages; p++) {
        int target_page = dest_y_page + p;
        if (target_page >= SCREEN_PAGES) continue; /* Boundary check */

        /* Loop through columns of source bitmap for this page */
        for (x = 0; x < src_width; x++) {
            int target_x = dest_x + x;
            if (target_x >= SCREEN_WIDTH) continue; /* Boundary check */

            /* Copy byte from source to destination buffer */
            dest_buffer[target_page * SCREEN_WIDTH + target_x] = src_bitmap[src_byte_idx];
            src_byte_idx++;
        }
    }
}

void Buzz(int number)
{
   int i;
   for (i=0; i<number; i++) {
     PB11=0; // PB11 = 0 to turn on Buzzer
     CLK_SysTickDelay(100000);  // Delay
     PB11=1; // PB11 = 1 to turn off Buzzer
     CLK_SysTickDelay(100000);  // Delay
   }
}
```

```c
void InitializeTrafficSystem(void)
{
    traffic_state = 0;
    time_remaining = 0;
    sequence_active = 0;
    blink_counter = 0;
    timer_counter = 0;

    // Turn off all LEDs initially
    PA12 = 1; // Blue off
    PA13 = 1; // Green off
    PA14 = 1; // Red off
}

// Set vehicle traffic lights (Red=PA14, Yellow=PA13+PA14, Green=PA13)
void SetVehicleLights(int red, int yellow, int green)
{
    if(red) {
        PA14 = 0; // Red on
        PA13 = 1; // Green off
    } else if(yellow) {
        PA14 = 0; // Red on
        PA13 = 0; // Green on (Red+Green=Yellow)
    } else if(green) {
        PA14 = 1; // Red off
        PA13 = 0; // Green on
    } else {
        PA14 = 1; // All off
        PA13 = 1;
    }
}

// Set pedestrian lights - using same PA pins for simplicity
void SetPedestrianLights(int red, int green)
{
    // For this lab, pedestrian lights can use the same logic as vehicle lights
    // or be controlled separately if needed
    (void)red;   // Suppress unused parameter warning
    (void)green; // Suppress unused parameter warning
}
```

```c
void StartTrafficSequence(void)
{
    if(!sequence_active) {
        sequence_active = 1;
        traffic_state = 1;
        time_remaining = 5; // State 1: 5 seconds
        timer_counter = 0;
        UpdateTrafficLights();
        UpdateSevenSegment();
        UpdateLCDDisplay();
    }
}

// Process 1-second timer for traffic sequence
void ProcessTrafficTimer(void)
{
    timer_counter++;
    if(timer_counter >= 1000) { // 1 second passed (assuming 1ms system tick)
        timer_counter = 0;

        if(sequence_active) {
            time_remaining--;

            if(time_remaining <= 0) {
                // Move to next state
                traffic_state++;

                switch(traffic_state) {
                    case 2: time_remaining = 3; break;  // 3 seconds
                    case 3: time_remaining = 3; break;  // 3 seconds
                    case 4: time_remaining = 10; break; // 10 seconds
                    case 5: time_remaining = 3; break;  // 3 seconds
                    default:
                        // Sequence complete, return to initial state
                        sequence_active = 0;
                        traffic_state = 0;
                        time_remaining = 0;
                        blink_counter = 0;
                        break;
                }
                UpdateTrafficLights();
                UpdateLCDDisplay(); // Only update LCD when state changes
            }
            UpdateSevenSegment(); // Update countdown display every second
        }
    }
}




// Update traffic lights based on current state
void UpdateTrafficLights(void)
{
    static int last_blink_state = -1; // Track last blink state to detect changes

    if(!sequence_active) {
        // Initial blinking state
        blink_counter++;
        if(blink_counter >= 500) { // Blink every 500ms
            blink_counter = 0;
            blink_state = !blink_state;

            // Vehicle: Yellow blink, Pedestrian: Red blink
            SetVehicleLights(0, blink_state, 0);  // Yellow blink
            SetPedestrianLights(blink_state, 0);  // Red blink

            // Only update LCD when blink state changes
            if(last_blink_state != blink_state) {
                UpdateLCDDisplay();
                last_blink_state = blink_state;
            }
        }
    } else {
        // Sequence states - LED control only, LCD updated separately
        switch(traffic_state) {
            case 1: // Vehicle: Green, Pedestrian: Red
                SetVehicleLights(0, 0, 1);
                SetPedestrianLights(1, 0);
                break;
            case 2: // Vehicle: Yellow, Pedestrian: Red
                SetVehicleLights(0, 1, 0);
                SetPedestrianLights(1, 0);
                break;
            case 3: // Vehicle: Red, Pedestrian: Red
```

```
                SetVehicleLights(1, 0, 0);
                SetPedestrianLights(1, 0);
                break;
            case 4: // Vehicle: Red, Pedestrian: Green
                SetVehicleLights(1, 0, 0);
                SetPedestrianLights(0, 1);
                break;
            case 5: // Vehicle: Red, Pedestrian: Red
                SetVehicleLights(1, 0, 0);
                SetPedestrianLights(1, 0);
                break;
        }
    }
}




// Update seven segment display
void UpdateSevenSegment(void)
{
    CloseSevenSegment();

    if(sequence_active && time_remaining > 0) {
        if(time_remaining < 10) {
            // Single digit, show on rightmost position
            ShowSevenSegment(0, time_remaining);
        } else {
            // Two digits
            ShowSevenSegment(1, time_remaining / 10);    // Tens
            ShowSevenSegment(0, time_remaining % 10);    // Units
        }
    } else {
        // Show 0 in initial state
        ShowSevenSegment(0, 0);
    }
}
```

```c
void UpdateLCDDisplay(void)
{
    if(!sequence_active) {
        // Initial state - display based on blink state
        if(blink_state) {
            // When yellow light is on, show both images dimmed
            print_C(stop_black, go_black);
        } else {
            // When lights are off, show STOP highlighted
            print_C(stop_white, go_black);
        }
        return;
    }

    // Display appropriate images based on current state (for pedestrians)
    switch(traffic_state) {
        case 1: // Vehicle Green - pedestrians must STOP
            print_C(stop_white, go_black);
            break;
        case 2: // Vehicle Yellow - pedestrians must STOP
        case 3: // Vehicle Red - pedestrians still must wait
            print_C(stop_white, go_black);
            break;
        case 4: // Vehicle Red, Pedestrian Green - pedestrians can GO
            print_C(stop_black, go_white);
            break;
        case 5: // Vehicle Red, Pedestrian Red - pedestrians must STOP again
            print_C(stop_white, go_black);
            break;
        default:
            print_C(stop_black, go_black);
            break;
    }
}

// BMP image data - 32x32 pixels (4 pages x 32 columns)
unsigned char go_white[32*4]={
0x00,0xFE,0x02,0x02,0x02,0x02,0x02,0x02,0x02,0x02,0x02,0xF2,0x0A,0x3A,0x02,0xEA,0x02,0xF2,0x0A,0x0A,0x0A,0xF2,0x02,0x02,0x02,0x02,0x02,0x02,0x02,0x02,0xFE,0x00,
0x00,0xFF,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xC0,0x20,0xF9,0x05,0x05,0x05,0x05,0x0C,0x78,0x09,0x19,0x11,0xB0,0x60,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0xFF,0x00,
0x00,0xFF,0x00,0x00,0xF8,0x04,0x02,0x02,0x1F,0xE0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x08,0x08,0x00,0x0E,0x2F,0x80,0x7F,0x10,0x10,0xE0,0x00,0x00,0xFF,0x00,
0x00,0x7F,0x40,0x40,0x43,0x44,0x44,0x44,0x42,0x41,0x42,0x44,0x44,0x48,0x48,0x48,0x48,0x48,0x48,0x48,0x4C,0x52,0x51,0x50,0x50,0x48,0x46,0x41,0x40,0x40,0x7F,0x00
};
```



```c
unsigned char go_black[32*4]={
0xFF,0x01,0xFD,0xFD,0xFD,0xFD,0xFD,0xFD,0xFD,0xFD,0xFD,0x0D,0xF5,0xC5,0xFD,0x15,0xFD,0x0D,0xF5,0xF5,0xF5,0x0D,0xFD,0xFD,0xFD,0xFD,0xFD,0xFD,0xFD,0xFD,0x01,0xFF,
0xFF,0x00,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x3F,0xDF,0x06,0xFA,0xFA,0xFA,0xFA,0xF3,0x87,0xF6,0xE6,0xEE,0x4F,0x9F,0x3F,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x00,0xFF,
0xFF,0x00,0xFF,0xFF,0x07,0xFB,0xFD,0xFD,0xE0,0x1F,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xF7,0xF7,0xFF,0xF1,0xD0,0x7F,0x80,0xEF,0xEF,0x1F,0xFF,0xFF,0x00,0xFF,
0xFF,0x80,0xBF,0xBF,0xBC,0xBB,0xBB,0xBB,0xBD,0xBE,0xBD,0xBB,0xBB,0xB7,0xB7,0xB7,0xB7,0xB7,0xB7,0xB7,0xB7,0xB3,0xAD,0xAE,0xAF,0xAF,0xB7,0xB9,0xBE,0xBF,0xBF,0x80,0xFF
};


unsigned char stop_white[32*4]={
0x00,0xFE,0x02,0x02,0xAA,0x0A,0x2A,0x0A,0x32,0x02,0x7A,0x42,0x52,0x0A,0xF2,0x02,0xF2,0x0A,0x0A,0x0A,0xF2,0x02,0xC2,0x42,0xF2,0x4A,0x4A,0xF2,0x02,0x02,0xFE,0x00,
0x00,0xFF,0x00,0x00,0x01,0x01,0x01,0x01,0x01,0xC0,0x20,0x10,0x10,0x08,0x09,0x88,0x88,0x09,0x89,0x91,0x10,0x20,0xC1,0x00,0x01,0x00,0x00,0x01,0x00,0x00,0xFF,0x00,
0x00,0xFF,0x00,0x00,0x00,0x00,0x78,0x86,0x81,0x80,0x40,0x00,0x08,0x08,0x00,0x0E,0x0F,0x20,0x0E,0x0F,0x00,0x88,0x08,0x03,0x84,0x78,0x00,0x00,0x00,0x00,0xFF,0x00,
0x00,0x7F,0x40,0x40,0x40,0x40,0x4C,0x52,0x51,0x50,0x51,0x51,0x52,0x52,0x54,0x48,0x48,0x58,0x54,0x54,0x52,0x52,0x51,0x51,0x52,0x4C,0x40,0x40,0x40,0x40,0x7F,0x00
};


unsigned char stop_black[32*4]={
0xFF,0x01,0xFD,0xFD,0x55,0xF5,0xD5,0xF5,0xCD,0xFD,0x85,0xBD,0xAD,0xF5,0x0D,0xFD,0x0D,0xF5,0xF5,0xF5,0x0D,0xFD,0x3D,0xBD,0x0D,0xB5,0xB5,0x0D,0xFD,0xFD,0x01,0xFF,
0xFF,0x00,0xFF,0xFF,0xFE,0xFE,0xFE,0xFE,0xFE,0x3F,0xDF,0xEF,0xEF,0xF7,0xF6,0x77,0x77,0xF6,0x76,0x6E,0xEF,0xDF,0x3E,0xFF,0xFE,0xFF,0xFF,0xFE,0xFF,0xFF,0x00,0xFF,
0xFF,0x00,0xFF,0xFF,0xFF,0xFF,0x87,0x79,0x7E,0x7F,0xBF,0xFF,0xF7,0xF7,0xFF,0xF1,0xF0,0xDF,0xF1,0xF0,0xFF,0x77,0xF7,0xFC,0x7B,0x87,0xFF,0xFF,0xFF,0xFF,0x00,0xFF,
0xFF,0x80,0xBF,0xBF,0xBF,0xBF,0xB3,0xAD,0xAE,0xAF,0xAE,0xAE,0xAD,0xAD,0xAB,0xB7,0xB7,0xA7,0xAB,0xAB,0xAD,0xAD,0xAE,0xAE,0xAD,0xB3,0xBF,0xBF,0xBF,0xBF,0x80,0xFF
};



// Display BMP images on LCD - STOP on top, GO on bottom
void print_C(unsigned char* stop_image, unsigned char* go_image)
{
    int i;
    int x_offset;

    clear_LCD();

    /* Clear the screen buffer */
    for (i = 0; i < SCREEN_WIDTH * SCREEN_PAGES; i++) {
        screen_buffer[i] = 0x00; /* Clear to black background */
    }

    /* Calculate horizontal offset for centering 32-pixel wide images */
    x_offset = (SCREEN_WIDTH - BMP_WIDTH) / 2; /* (128 - 32) / 2 = 48 */

    /* Copy the top bitmap (STOP image) centered, starting at Page 0 */
    copy_bitmap_to_buffer(screen_buffer, stop_image, x_offset, 0, BMP_WIDTH, BMP_PAGES);
    copy_bitmap_to_buffer(screen_buffer, go_image, x_offset, 4, BMP_WIDTH, BMP_PAGES);

    /* Draw the combined buffer to the LCD */
    draw_LCD(screen_buffer);
}

// LCD BMP display functions are provided by LCD.h
// Using draw_LCD() function as shown in tutorial materials

// Display single BMP image at specific position
void print_C_at_position(unsigned char* image, int start_page, int start_col)
{
    int i;

    /* Clear the screen buffer */
    for (i = 0; i < SCREEN_WIDTH * SCREEN_PAGES; i++) {
        screen_buffer[i] = 0x00;
    }

    /* Copy the bitmap to specified position */
    copy_bitmap_to_buffer(screen_buffer, image, start_col, start_page, BMP_WIDTH, BMP_PAGES);

    /* Draw to LCD */
    draw_LCD(screen_buffer);
}

int main(void)
{
    uint8_t keyin, last_key = 0;
    uint32_t debounce_counter = 0;
    uint8_t key_processed = 0;

    SYS_Init();

    // Setup GPIO for traffic lights
    GPIO_SetMode(PA, BIT12, GPIO_PMD_OUTPUT); // Blue LED
    GPIO_SetMode(PA, BIT13, GPIO_PMD_OUTPUT); // Green LED
    GPIO_SetMode(PA, BIT14, GPIO_PMD_OUTPUT); // Red LED
    GPIO_SetMode(PB, BIT11, GPIO_PMD_OUTPUT); // Buzzer control

    // Initialize system
    InitializeTrafficSystem();
    PB11 = 1; // Turn off Buzzer
    init_LCD();
    clear_LCD();

    // Initialize 7-segment display
    OpenSevenSegment();
    UpdateSevenSegment();

    OpenKeyPad(); // initialize 3x3 keypad

    // Initial LCD update
    UpdateLCDDisplay();

    while(1)
    {
        // Process timer for 1-second intervals
        ProcessTrafficTimer();

        // Update traffic lights (handles blinking in initial state)
        UpdateTrafficLights();

        keyin = ScanKey(); // scan keypad to input

        // Debounce and key processing logic
        if(keyin != 0) {
            // Key is pressed
            if(keyin == last_key) {
                // Same key, increment debounce counter
                debounce_counter++;
                if(debounce_counter > 50 && !key_processed) {  // Reduced from 1000 to 50
                    // Key is stable and not yet processed
                    key_processed = 1;

                    // Process the key - only when NOT in sequence
                    if(!sequence_active) {
                        switch(keyin) {
                            case 5: // GO key - Start traffic sequence
                                StartTrafficSequence();
                                Buzz(1); // Give audio feedback only when starting sequence
                                break;
                            // Other keys are inactive in traffic light system
                        }
                    }
                }
            } else {
                // Different key pressed, reset debounce
                debounce_counter = 0;
                key_processed = 0;
            }
        } else {
            // No key pressed, reset all
            debounce_counter = 0;
            key_processed = 0;
        }

        last_key = keyin;

        // Small delay to prevent excessive CPU usage
        CLK_SysTickDelay(1000); // 1ms delay
    }
}
```