VIETNAM NATIONAL UNIVERSITY
HO CHI MINH UNIVERSITY OF TECHNOLOGY
COMPUTER SCIENCE & ENGINEERING FACULTY

MICROCONTROLLER MICROPROCESSOR (CO3009)

Lab Report

Lab 3

BUTTONS AND SWITCHES

Teacher: Huynh Phuc Nghi
Student: Nguyen Thanh Hien (2111203)

HO CHI MINH CITY, OCTOBER 2023

Contents

List of Figures

List of Tables

1 EXERCISE

1.1 Overall requirement

1.1.1 Target product

Build an application of traffic lights in a four-way intersection of a crossroad, with customization features for time duration.

1.1.2 Components

- 12 single LEDs: 4 red, 4 amber, and 4 green.
- 4 seven segment LEDs
- 3 buttons

1.1.3 Modes

- **Mode 1 (Normal mode):** The traffic light running as configured
  - 12 LEDs (4 red, 4 green, and 4 amber): to simulate the traffic light indicator which corresponded to the STOP, GO, and SLOW DOWN signals blinking at 2 Hz.
  - 4 seven segment LEDs: to display time for each 2 roads which related to the time remaining for each LED indicator, in a range of 1 - 99.
  - Button 1: Switch to the next mode.
- **Mode 2, 3, 4 (Modify RED, AMBER, GREEN):** To modify the time duration of each LED color in the main road.
  - 2 seven segment LEDs: Display the mode currently in.
  - 2 seven segment LEDs: Display the time duration value, in a range of 1 - 99.
  - Button 1: Swicth to the next mode
  - Button 2: Increase the time duration value.
  - Button 3: Officially choose the value.

1.2 Exercise and Report

Question 1

Question: Sketch an FSM that describes your idea of how to solve the problem.
Answer:
In Figure 1, we illustrated that only Button 1 is used for switching between states. However, the system only detects and allows switching state once the button is released after being pressed.

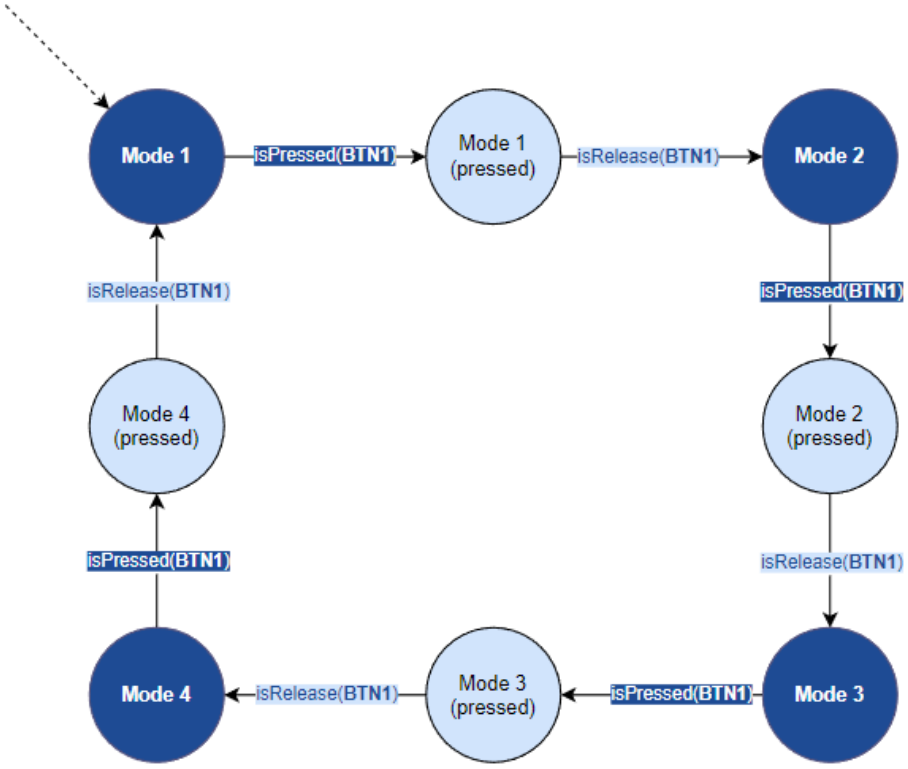


Figure 1: FSM: Button states

With the **Normal mode** ***(Mode 1)*** in Figure 2, we have 4 traffic lights each for the four-way intersection. However, there are actually only 2 separate traffic light indicator systems running since pairs in the opposite position behave precisely the same.

We have the color order of each system run in a loop: red, green, amber, and then back to red. However, each system differs by the initial state: The first one starts with red, while the second one starts with green.

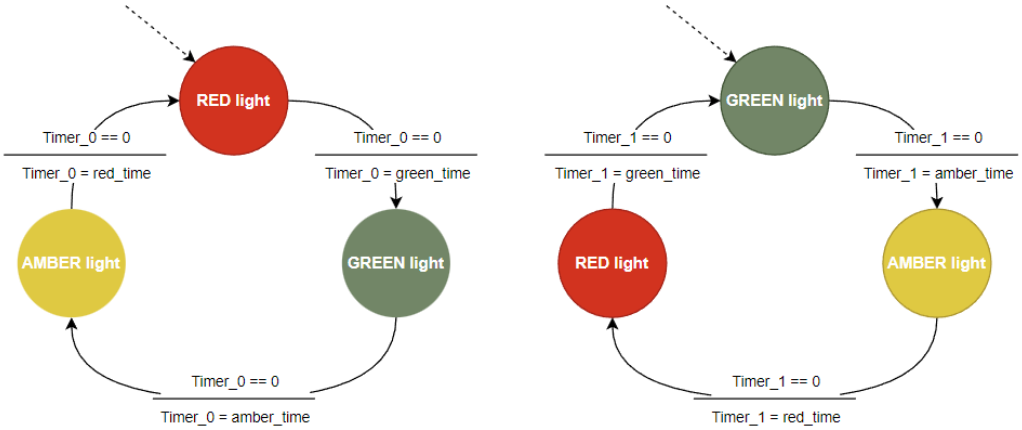


Figure 2: FSM: Traffic light indication for STOP, GO, and HALT

With the **Modifying time duration mode** ***(Mode 2, 3, 4)***, we have the initial state entered once the user presses **Button 1**. Figure 3 is a detailed version of Figure 1 specifically for modes 2 (red), 3 (amber), and 4 (green).

Users can modify the time duration by pressing **Button 2** to increase (then back to 0) in the range of 1 - 99. The time duration would increase by one for each press or gradually increase if the user chooses to not release the button but progressively presses on it.

Once entered Mode 2/3/4, the system waits for the user to press Button 2 to actually modify the time duration. If pressed once (or holding without releasing), the system would store the modified value in a temporary variable. The user must confirm the chosen value by pressing Button 3. Otherwise, if the user switches to the next mode by pressing Button 1, the system won't save the chosen value.

If the user correctly saves their preference by pressing Button 3, the temporary variable will be assigned as the max time duration for the corresponding LED color.

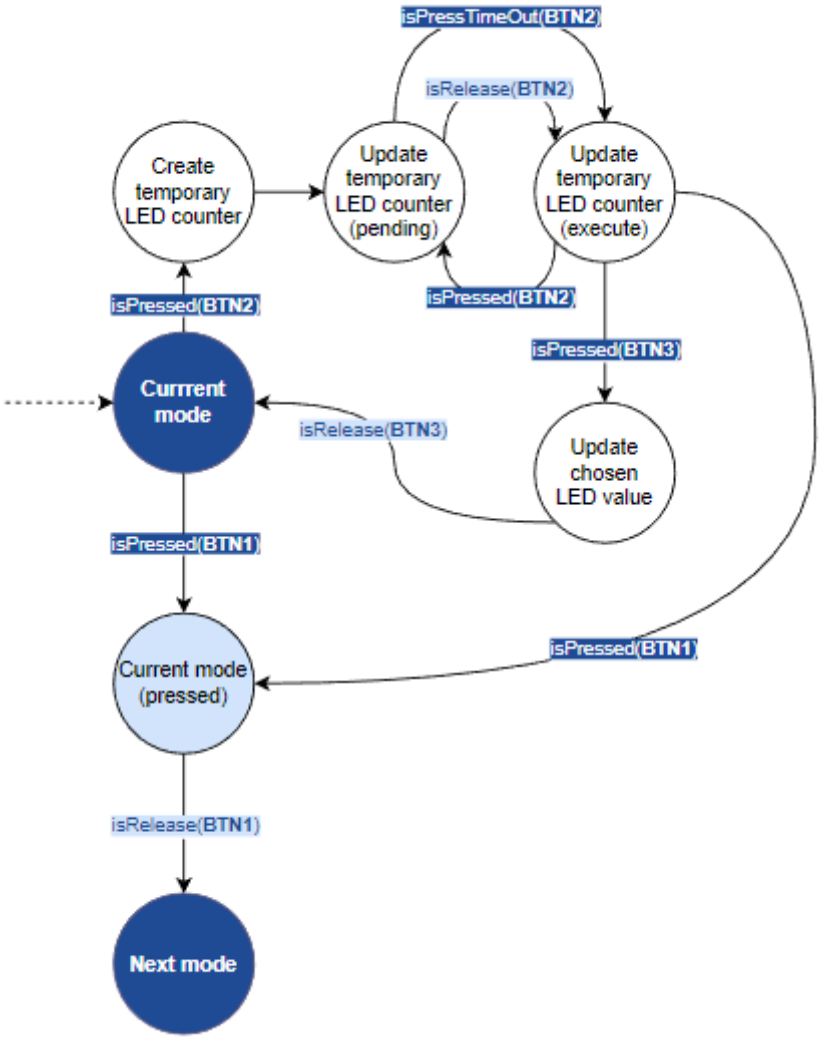


Figure 3: FSM: Modifying time duration on mode 2, 3, 4

Question 2

Question: Draw a Proteus schematic for the problem above.

Answer:

Even though the specification stated that it only requires 4 seven segment LEDs, I added 4 other LEDs on the opposite side of the original 4 LED design. The system behavior doesn't change, but the representation of a 4-way intersection is more realistic.

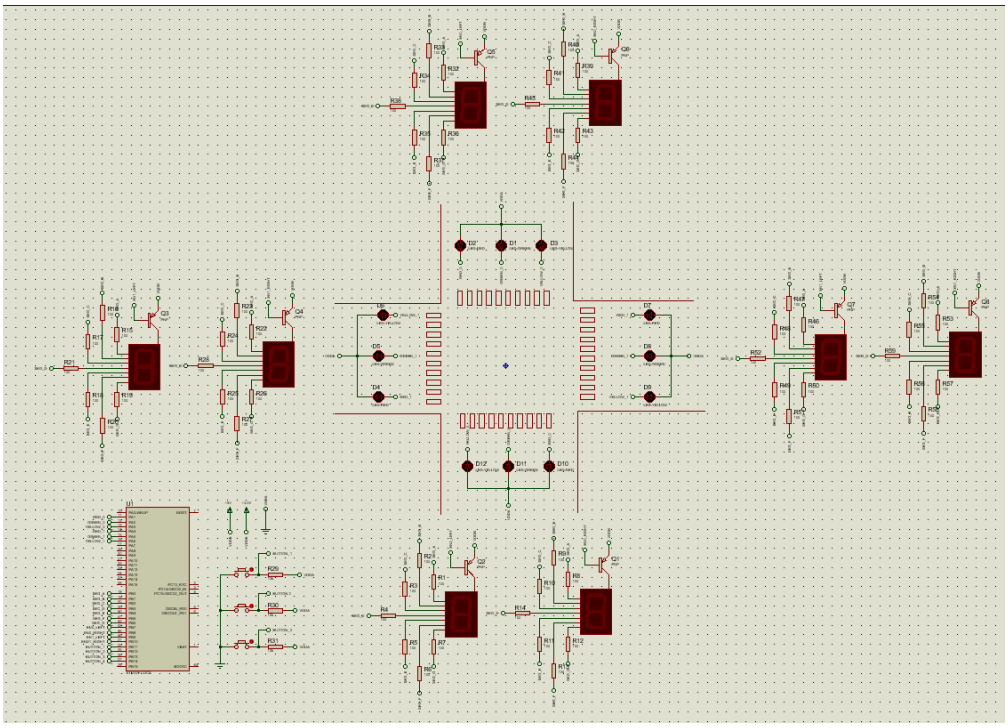
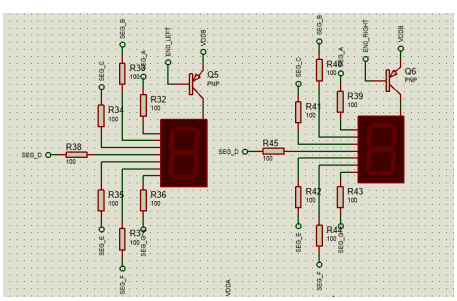
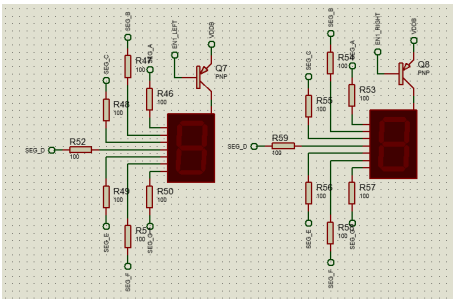


Figure 4: Schematic: Overall design



(a) Schematic: One side



(b) Schematic: Another side

Figure 5: Schematic: 2 digits of 4 segment LEDs

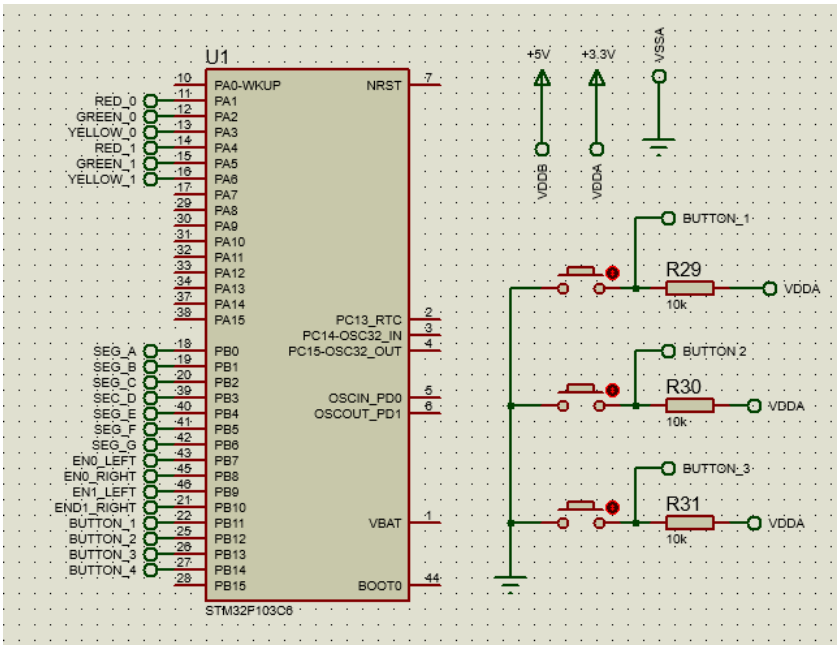


Figure 6: Schematic: MCU and Button

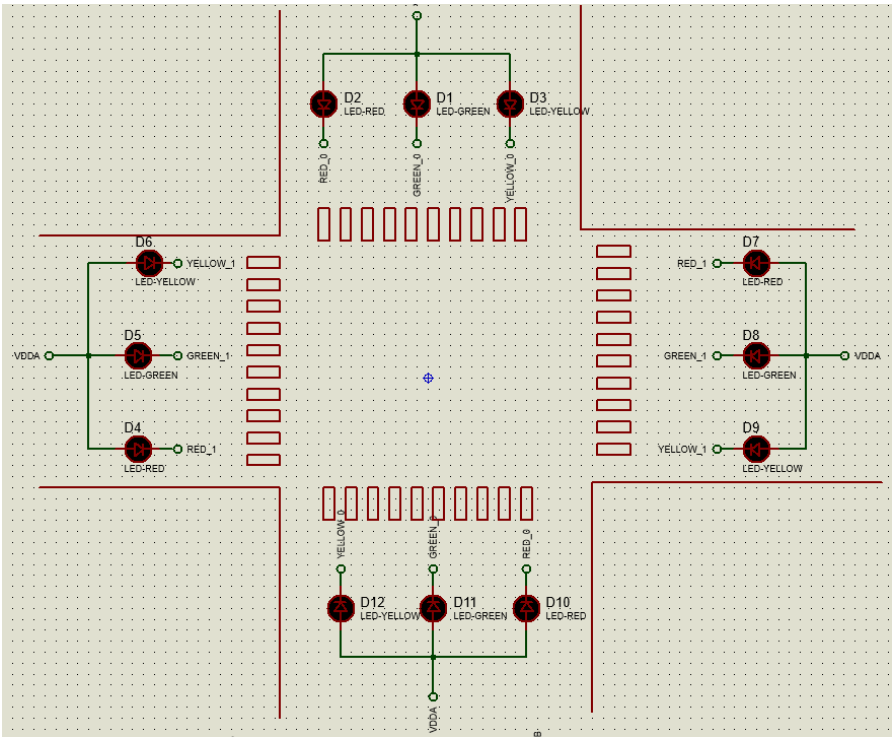


Figure 7: Schematic: 4-way intersection LED color indicator

Question 3

Question: Create a project that has a pin corresponding to the Proteus schematic that you drew in the previous section. You need to set up your timer interrupt is about 10ms.

Answer: To have a timer interrupt of 10ms, I divided the original internal clock: 8MHz by 8000 times with the prescaler. This means that every tick of the clock lasted 1ms, then to achieve 10ms, I multiplied by 10 with the counter period.

If we are going the change the timer interrupt, we must not surpass **10ms** since that was the maximum value to read a button input sample.

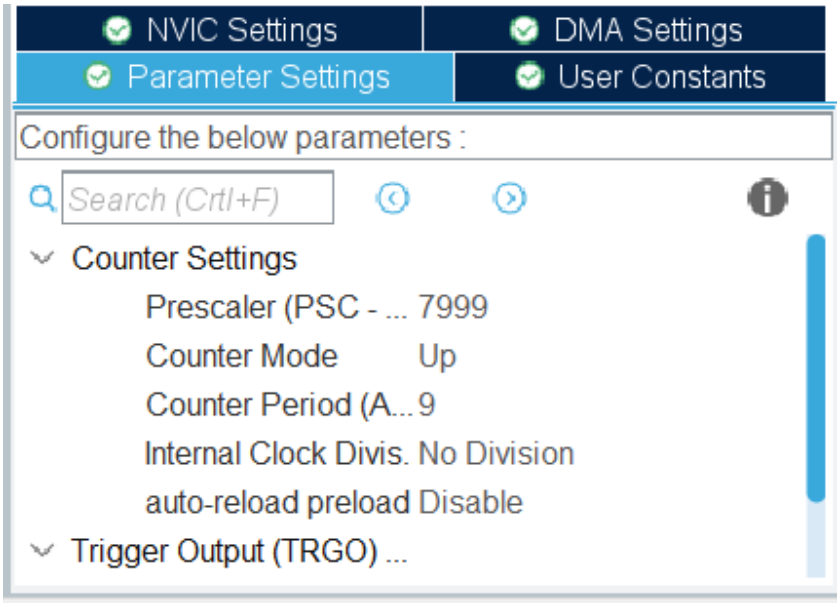


Figure 8: Timer setup (initially)

In this project, I would need to add 6 more additional header files, corresponds to the 6 additional source file:

1. `global`: Store global macros, enums that are shared with sub-modules, allows define physical port/pin as array.
   - ***Import:*** `main.h`
2. `timer`: Acts as software timer of the program.
   - ***Import:*** `global.h`
3. `led_segment`: Control the individual LED segments with scanning feature.
   - ***Import:*** `global.h`, `timer.h`
4. `led_indicator`: Control each side of the LEDs traffic color.

- ***Import:*** `global.h`, `timer.h`

5. `button`: Reads button value, determines when should the request is needed (depending on modes)
   - ***Import:*** `global.h`, `timer.h`
6. `traffic_light`: Deploy FSM for led indicator, led segment and button as a complete system
   - ***Import:*** `led_indicator.h`, `led_segment.h`, `button`,

```c
/* Import library */
#include "main.h"

/* Public define */
#define NUM_SIDE 2
#define NUM_DIGIT 2
#define NUM_COLOR 3
#define NUM_BUTTON 3
#define NUM_MODE 4
#define NUM_DEBOUNCE 3
#define TURN_OFF GPIO_PIN_SET
#define TURN_ON GPIO_PIN_RESET

/* Public variables declaration */
typedef struct _GPIO_config{
  GPIO_TypeDef* port;
  uint16_t pin;
} GPIO_config;

typedef enum {NORMAL = 0, MODIFY_RED, MODIFY_AMBER,
   MODIFY_GREEN} MODE;
typedef enum {SELECT_MODE, MODIFY_VALUE, SELECT_VALUE,
   NONE} BUTTON_PURPOSE;
typedef enum {RED = 0, AMBER, GREEN} TRAFFIC_LIGHT_COLOR
   ;
typedef enum {SIDE_A = 0, SIDE_B} TRAFFIC_LIGHT_SIDE;
typedef enum {LEFT = 0, RIGHT} LED_SEG_DIGIT;
typedef enum {TIMER_BUTTON = 0, TIMER_LED_INDICATOR,
   TIMER_LED_SEGMENT, TIMER_SECOND} TIMING;
/* Public function declaration */
```

Program 1: `global.h`

Question 4

Question: Your task in this exercise is to modify the timer settings so that when we want to change the time duration of the timer interrupt, we change it the least and it will not affect the overall system. For example, the current system we have implemented is that it can blink an LED at 2 Hz, with the timer interrupt duration is 10ms. However, when we want to change the timer interrupt duration to 1ms or 100ms, it will not affect the 2Hz blinking LED.

Answer:

We assume that the entire program depends on timer interrupt duration, we make it an unknown value. As requested, we can create 4 timers:

1. **Button with 2 counters:** With the timer limit is to determine when we should take a sample (must be < 10ms) is the timer interrupt itself, and `NUM_DEBOUNCE` is how many samples are enough to determine the result.
2. **LED segment:** We set this arbitrarily, with a duration of 50ms to light up each LED. Therefore, the frequency would be 50Hz.
3. **LED indicator:** All must blinks at 2Hz in modify mode, which means it toggles each 250ms.
4. **Clock in seconds:** To count down for traffic light number.

```
/* Import library */
#include "main.h"
#include "UDF_global.h"

/* Public variables declaration */
#define NUM_TIMER 4
#define SECOND_DURATION_MS 1000     // Clock in second
#define INTERRUPT_DURATION_MS 10    // Timer interrupt
#define LED_INDICATOR_DURATION_MS 250 // LED indicator
#define LED_SEGMENT_DURATION_MS 10    // LED segment
#define BUTTON_DURATION_MIN 10      // Min response time
#define BUTTON_DURATION_MAX 50      // Max response time

typedef struct{
  int limit;
  int counter;
  int flag;
}timer;
timer timers[NUM_TIMER];

```

```
/* Public function declaration */
void init_timer();
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *
    htim);
```

Program 2: timer.h

```
/* Import library */
#include "UDF_timer.h"

/* Public function  */
void init_timer(){
  // Set counter limit to raise a flag
  timers[TIMER_BUTTON].limit = BUTTON_DURATION_MIN /
   INTERRUPT_DURATION_MS;
  timers[TIMER_LED_INDICATOR].limit =
   LED_INDICATOR_DURATION_MS / INTERRUPT_DURATION_MS;
  timers[TIMER_LED_SEGMENT].limit =
   LED_SEGMENT_DURATION_MS / INTERRUPT_DURATION_MS;
  timers[TIMER_SECOND].limit = SECOND_DURATION_MS /
   INTERRUPT_DURATION_MS;

  // Reset counter and flag to default value
  for(uint16_t timer = 0; timer < NUM_TIMER; timer++){
    timers[timer].counter = timers[timer].limit;
    timers[timer].flag = 0;
  }
}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *
    htim){
  for(uint16_t timer = 0; timer < NUM_TIMER; timer++){
    timers[timer].counter--;
    if(timers[timer].counter <= 0){
      timers[timer].flag = TURN_ON;   // Raise the flag
      timers[timer].counter = timers[timer].limit;  //
   Reset counter
    }
  }
}
```

Program 3: time.c

Question 5

Question:
Following the example of button reading and debouncing in the previous section, your tasks in this exercise are:

- To add new files for input reading and output display.
- To add code for button debouncing.
- To add code for increasing mode when the first button is pressed.

Answer: For this exercise, I determine 4 phases for debouncing:

1. **First press:** Users first press the button, and the system debounces and accepts this action as a single press.
2. **Delay before long press:** After that single press, if users don't release the button but continue, the system will delay for a certain period.
3. **Auto increase:** After the delay ended, the system would accept another press. Then, if users continue to press without releasing the button, the system would request action periodically.
4. **Release:** The button initially released or users release it, the system would reset to its default state.

```c
/* Import library */
#include "UDF_global.h"
#include "UDF_timer.h"

/* Public function declaration */
void init_button();
void button_reading();
unsigned char is_button_pressed(uint16_t index);
unsigned char is_button_long_pressed(uint16_t index);
unsigned char is_button_response_needed(uint16_t index);
void finish_to_respond(uint16_t index);
```

Program 4: `button.h`

This interface only includes reading the button, and raising request for the system to execute (since many buttons don't react to long press but only the first press detected). The rest of this functionality would be connected in `traffic_light.c` file.

```c
#include "UDF_button.h"

/* Private define */
#define BUTTON_IS_PRESSED GPIO_PIN_RESET
#define BUTTON_IS_RELEASED GPIO_PIN_SET
#define DELAY_BEFORE_LONG_PRESS_MS (BUTTON_DURATION_MAX * 10)
#define DURATION_FOR_AUTO_INCREASE_MS (NUM_DEBOUNCE * INTERRUPT_DURATION_MS)

typedef struct _button{
  GPIO_config physical;
  GPIO_PinState state;
  int currentSampleIndex;
  int counterForLongPress; // Delay before auto increase
  uint8_t flagToResponse;      // Allow to increase
  uint8_t flagForLongPress;  // Delay state
} button;

/* Private variables declaration */
static GPIO_PinState debounceBuffer[NUM_DEBOUNCE][NUM_BUTTON];
button buttons[NUM_BUTTON];
```

Program 5: Define part in `button.c`

- `NUM_DEBOUNCE`: number of timer interrupts needed to consider accepting a set of samples in ***phase 1***.
- `DELAY_BEFORE_LONG_PRESS_MS`: the delay period (in ms) for ***phase 2***.
- `DURATION_FOR_AUTO_INCREASE_MS`: the period (in ms) for ***phase 3***.

```
void button_reading(){
  /* Trigger: Button sampling with timer interrupt, flag raised */
  for(int index = 0; index < NUM_BUTTON; index++){
    // Shift buffer & Read pin
    debounceBuffer[2][index] = debounceBuffer[1][index];
    debounceBuffer[1][index] = debounceBuffer[0][index];
    debounceBuffer[0][index] = HAL_GPIO_ReadPin(
        buttons[index].physical.port, buttons[index].physical.pin);

    // Filter 3 most recent data
    if ((debounceBuffer[0][index] == debounceBuffer[1][index])
        && (debounceBuffer[0][index] == debounceBuffer[2][index])) {

      // Detect a press/release
      if(buttons[index].state != debounceBuffer[0][index]){
        buttons[index].state = debounceBuffer[0][index];

        // Detect press: Delay & Response first time
        if(buttons[index].state == BUTTON_IS_PRESSED){
          // Count down for delay before long press
          buttons[index].counterForLongPress
          = DELAY_BEFORE_LONG_PRESS_MS / INTERRUPT_DURATION_MS;
          buttons[index].flagToResponse = TURN_ON;
        }

        // Detect release: Reset state
        if(buttons[index].state == BUTTON_IS_RELEASED){
          buttons[index].counterForLongPress = 0;
          buttons[index].flagForLongPress = TURN_OFF;
        }
      }
      // Simply sampling data
      else{
        if(buttons[index].state == BUTTON_IS_PRESSED){
```

```
            buttons[index].counterForLongPress--;
            //Start auto-increase at certain rate
            if(buttons[index].counterForLongPress <= 0){
              buttons[index].flagForLongPress = TURN_ON;
              buttons[index].flagToResponse = TURN_ON;
              buttons[index].counterForLongPress
              = DURATION_FOR_AUTO_INCREASE_MS /
    INTERRUPT_DURATION_MS;
            }
          }
        }
      }
    }
}
```

Program 6: Button reading in `colorpurplebutton.c`

```
unsigned char is_button_pressed(uint16_t index){
  if(index >= NUM_BUTTON) return 0;
  return (buttons[index].state == BUTTON_IS_PRESSED);
}

unsigned char is_button_long_pressed(uint16_t index){
  if(index >= NUM_BUTTON) return 0;
  return (buttons[index].flagForLongPress == TURN_ON);
}

unsigned char is_button_response_needed(uint16_t index){
  if(index >= NUM_BUTTON) return 0;
  return (buttons[index].flagToResponse == TURN_ON);
}

void finish_to_respond(uint16_t index){
  buttons[index].flagToResponse = TURN_OFF;
}
```

Program 7: Boolean interface of `button.c`

Since many buttons we defined, don't respond to long press but only respond to the first press (aka. phase 1). We leave a flag called `flagToResponse` for the system to catch, determine whether to execute with that request and then reset the flag by the function `finish_to_respond(uint16_t index)`.

As for increasing the mode whenever needed to increase the mode, we need to inspect traffic_light.c.

```
uint8_t respond_to_button(BUTTON_PURPOSE buttonIndex){
  uint8_t result = NO_RESPONSE_NEEDED;
  // Response to certain actions
  if(is_button_response_needed(buttonIndex)){
     finish_to_respond(buttonIndex); // Reset after
   responds
     switch(buttonIndex){
       // Response to: Single press only
       case SELECT_MODE:
       case SELECT_VALUE:
         if(!is_button_long_pressed(buttonIndex))
           result = RESPONSE_NEEDED;
         break;
       // Response to: Single/Long press
       case MODIFY_VALUE:
         result = RESPONSE_NEEDED;
         break;
     }
   }
   return result;
}

void fsm_button(){
  /* Read & Process buttons*/
  if(timers[TIMER_BUTTON].flag == TURN_ON){
    button_reading();
    timers[TIMER_BUTTON].flag = TURN_OFF;
  }

  // Catch button trigger: Prioritized (BTN1 > BTN2 >
   BTN3)
  for(int index = NUM_BUTTON - 1; index >= 0; index--){
    if(respond_to_button(index) == RESPONSE_NEEDED){
      button_to_response = index;
    }
  }
}
```

Program 8: Button functions in traffic_light.c

We can break down this implementation by functionalities:

1. `respond_to_button`: A function to catch response from a SINGLE button, output whether the system needs to respond to that trigger. We have **Button 1, 3** only accepts first press (not long press) and **Button 2** accepts just any.

2. `fsm_button`: An interface for the system entire interface to call, which would do the job ob: take a sample of ALL the buttons, determine which button the system should respond to in priority order: **Button 1 > Button 2 > Button 3** (if any button requests).

```
case SELECT_MODE:
    // Go to next mode, reset modify value
    system.mode = (system.mode + 1) % NUM_MODE;
    modifyValue = DEFAULT_MODIFY_VALUE;
    if(system.mode == NORMAL){
        // Display LEDs segment: Traffic count down
        modify_led_segment_buffer(SIDE_A, system.
  countDownTimer[SIDE_A]);
        modify_led_segment_buffer(SIDE_B, system.
  countDownTimer[SIDE_B]);
     }
     else{
        // Display LEDs segment: Mode, Modify value
        modifyValue = system.countDownLimit[system.mode
  - 1];
        modify_led_segment_buffer(SIDE_A, system.mode);
        modify_led_segment_buffer(SIDE_B, modifyValue);
     }
        display_led_indicator(SIDE_A, RED);
     break;
```

Program 9: Switch case extract from `fsm_traffic` for mode switching

In Program 9, with each button after processed, the system determines to respond to **Button 1 (SELECT MODE)**. It would go to the next mode each time it was called.

Question 6

Question: Your task in this exercise are:

- To add code to display mode on seven-segment LEDs and
- To add code for blinking LEDs depending on the mode that is selected.

Answer:

```
#include "UDF_global.h"
#include "UDF_timer.h"
/* Public variables declaration */

/* Public function declaration */
void init_led_segment();
void display_led_segment();
void modify_led_segment_buffer(TRAFFIC_LIGHT_SIDE
   trafficSide, uint16_t number);
```

Program 10: led_segment.h

```
/* Import library */
#include "UDF_global.h"
#include "UDF_timer.h"
/* Public variables declaration */

/* Public function declaration */
void init_led_indicator();
void display_led_indicator(TRAFFIC_LIGHT_SIDE
   trafficSide, TRAFFIC_LIGHT_COLOR trafficColor);
void blink_all_led();
```

Program 11: led_indicator.h

We have these files: led_segment and led_indicator to control individual elements of the system. We wouldn't go in detail of how each of these elements is executed since they have been demonstrated in the first two labs.

We would care about Program 12 and Program 13, which holds the logic for switching display value for each mode. These functions ONLY change based on **system mode** and timer interrupt.

- If we switched to **NORMAL MODE**: The system would display by default, acts as a 4-way traffic system.
- If we switched to **other mode**: The LED indicator would blink at 2Hz, and other LED segments would display the mode and the current modified value.

```c
void fsm_led_indicator(){
    if(system.mode == NORMAL){
        // Display traffic color as normal
        display_led_indicator(SIDE_A, system.colors[SIDE_A]);
        display_led_indicator(SIDE_B, system.colors[SIDE_B]);

        // Change color state each side
        for(uint8_t side = 0; side < NUM_SIDE; side++){
            if(system.countDownTimer[side] <= 0){
                switch(system.colors[side]){
                case RED:
                    system.colors[side] = GREEN;
                    system.countDownTimer[side] = system.countDownLimit[GREEN];
                    break;
                case AMBER:
                    system.colors[side] = RED;
                    system.countDownTimer[side] = system.countDownLimit[RED];
                    break;
                case GREEN:
                    system.colors[side] = AMBER;
                    system.countDownTimer[side] = system.countDownLimit[AMBER];
                }
            }
        }
    }
    else{// Blink LEDs in other modes
        if(timers[TIMER_LED_INDICATOR].flag == TURN_ON){
            blink_all_led();
            timers[TIMER_LED_INDICATOR].flag = TURN_OFF;
        }
    }
}

```

Program 12: Function `fsm_led_indicator()` in `colorpurpletraffic_light.c`

```c
void fsm_led_segment(){
  display_led_segment();
  switch(system.mode){
  // Traffic light running normally
  case NORMAL:
    if(timers[TIMER_SECOND].flag == TURN_ON){
      // Count down each second
      system.countDownTimer[SIDE_A]--;
      system.countDownTimer[SIDE_B]--;
      // Display the count down number
      modify_led_segment_buffer(SIDE_A, system.
   countDownTimer[SIDE_A]);
      modify_led_segment_buffer(SIDE_B, system.
   countDownTimer[SIDE_B]);
    }
    break;
  // Other cases: Modify mode
  case MODIFY_RED:
  case MODIFY_AMBER:
  case MODIFY_GREEN:
    modify_led_segment_buffer(SIDE_A, system.mode);
    break;
  }
  // Reset timer flag of second tracker
  timers[TIMER_SECOND].flag = TURN_OFF;
}
```

Program 13: Function `fsm_led_segment()` in `colorpurpletraffic_light.c`

Question 7

Question: Your task in this exercises are:

- To use the second button to increase the time duration value of the red LEDs.
- To use the third button to set the value for the red LEDs.

Answer:

These steps can be standardized into the procedure, such that: red/amber/green are the only parameter that needs to be anchored to execute. This code below can be used for Exercise 7-9.

```
void fsm_traffic_light_system(){
    /* Output display: led indicator, led segment*/
    fsm_led_indicator();
    fsm_led_segment();
    fsm_button();

    /* Finite state machine: Button to mode */
    switch(button_to_response){
    case SELECT_MODE:
        // Go to next mode, reset modify value
        system.mode = (system.mode + 1) % NUM_MODE;
        modifyValue = DEFAULT_MODIFY_VALUE;
        if(system.mode == NORMAL){
            // Display LEDs segment: Traffic count down
            modify_led_segment_buffer(SIDE_A, system.
countDownTimer[SIDE_A]);
            modify_led_segment_buffer(SIDE_B, system.
countDownTimer[SIDE_B]);
        }
        else{
            // Display LEDs segment: Mode, Modify value
            modifyValue = system.countDownLimit[system.
mode - 1];
            modify_led_segment_buffer(SIDE_A, system.
mode);
            modify_led_segment_buffer(SIDE_B,
modifyValue);
        }
            display_led_indicator(SIDE_A, RED);
        break;
```

```
26      case MODIFY_VALUE:
27          if(system.mode != NORMAL){
28              // Increase value
29              modifyValue = (modifyValue + 1) %
    DEFAULT_MODIFY_VALUE;
30              modify_led_segment_buffer(SIDE_B,
    modifyValue);
31          }
32          break;
33      case SELECT_VALUE:
34          if(system.mode != NORMAL){
35              // Save modified value
36              system.countDownLimit[system.mode - 1] =
    modifyValue;
37          }
38          break;
39      default: break;
40      }
41      button_to_response = NONE;// Reset response trigger
42  }
```

Program 14: FSM for traffic light in `traffic_light`

Question 8

Question: Your task in this exercises are:

- To use the second button to increase the time duration value of the amber LEDs.
- To use the third button to set the value for the amber LEDs.

Answer: The answer can be found in Program 14

Question 9

Question: Your task in this exercise are:

- To use the second button to increase the time duration value of the green LEDs.
- To use the third button to set the value for the green LEDs.

Answer:

Question 10

Question: Your task in this exercise are:

- To integrate all the previous tasks to one final project
- To create a video to show all features in the specification
- To add a report to describe your solution for each exercise
- To submit your report and code on the BKel

Answer:

- GitHub Repository: https://github.com/amyranotamirror/CO3007-2111203
- Video demo: https://clipchamp.com/watch/xehgned4zv2

2 REFERENCE