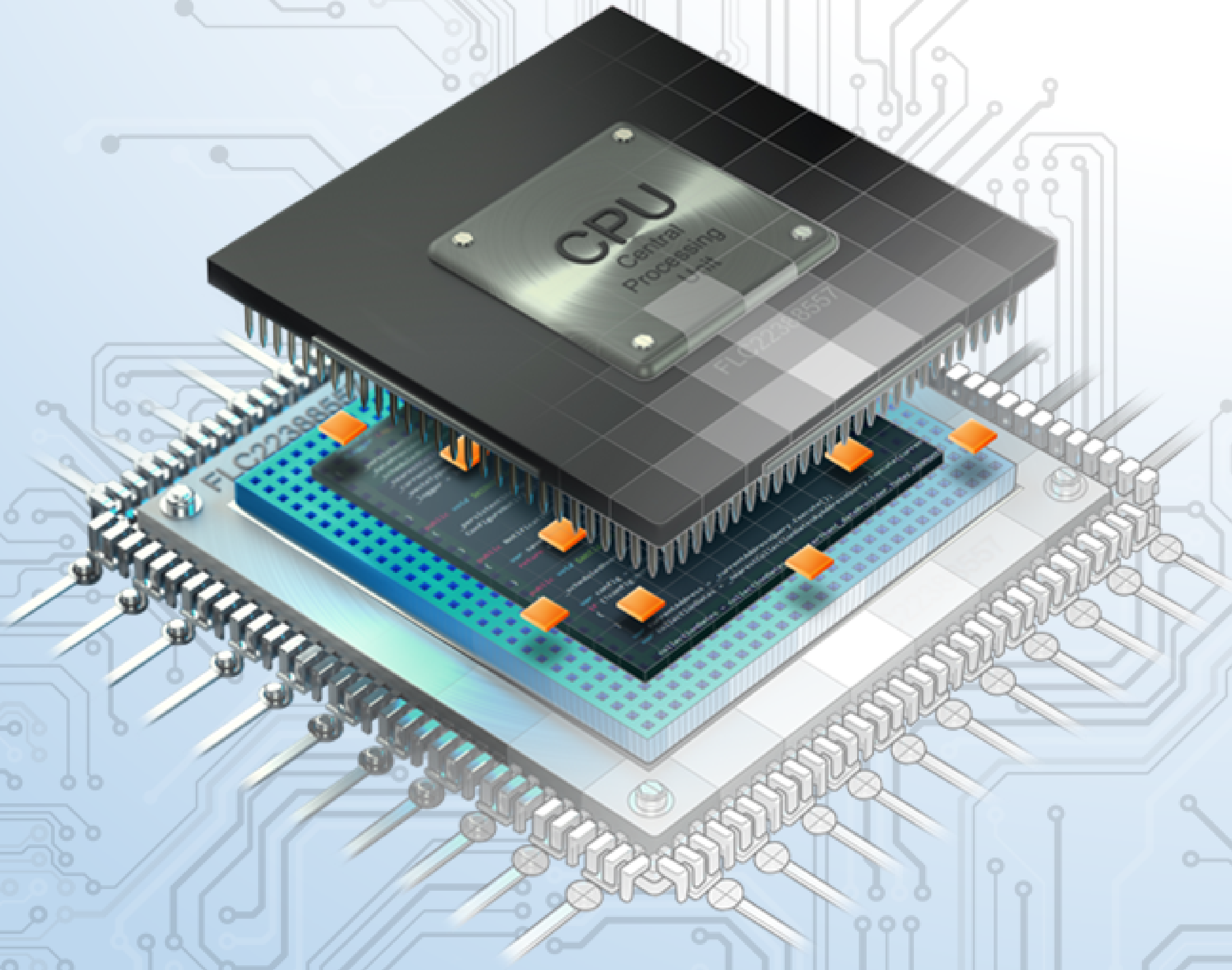
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
COMPUTER ENGINEERING

Microcontroller

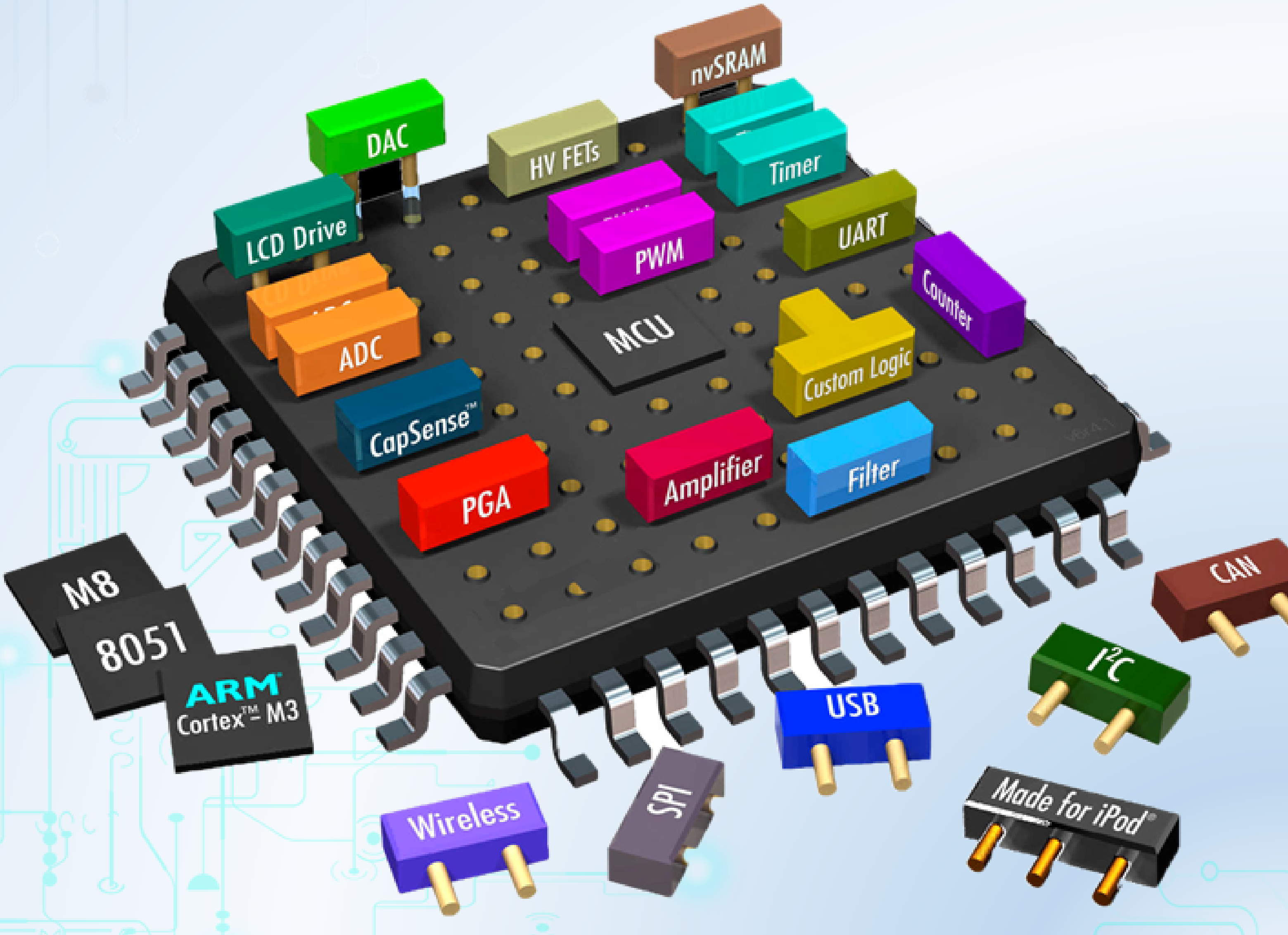


Dr. Le Trong Nhan

Contents

CHAPTER 1

Buttons/Switches



1 Objectives

In this lab, you will

- Learn how to add new C source files and C header files in an STM32 project,
- Learn how to read digital inputs and display values to LEDs using a timer interrupt of a microcontroller (MCU).
- Learn how to debounce when reading a button.
- Learn how to create an FSM and implement an FSM in an MCU.

2 Introduction

Embedded systems usually use buttons (or keys, or switches, or any form of mechanical contacts) as part of their user interface. This general rule applies from the most basic remote-control system for opening a garage door, right up to the most sophisticated aircraft autopilot system. Whatever the system you create, you need to be able to create a reliable button interface.
A button is generally hooked up to an MCU so as to generate a certain logic level when pushed or closed or “active" and the opposite logic level when unpushed or open or “inactive." The active logic level can be either ‘0’ or ‘1’, but for reasons both historical and electrical, an active level of ’0’ is more common.
We can use a button if we want to perform operations such as:

- Drive a motor while a switch is pressed.
- Switch on a light while a switch is pressed.
- Activate a pump while a switch is pressed.

These operations could be implemented using an electrical button without using an MCU; however, use of an MCU may well be appropriate if we require more complex behaviours. For example:

- Drive a motor while a switch is pressed.

  Condition: If the safety guard is not in place, don’t turn the motor. Instead sound a buzzer for 2 seconds.

- Switch on a light while a switch is pressed.

  Condition: To save power, ignore requests to turn on the light during daylight hours.

- Activate a pump while a switch is pressed

  Condition: If the main water reservoir is below 300 litres, do not start the main pump: instead, start the reserve pump and draw the water from the emergency tank.

In this lab, we consider how you read inputs from mechanical buttons in your embedded application using an MCU.

3 Basic techniques for reading from port pins

3.1 The need for pull-up resistors

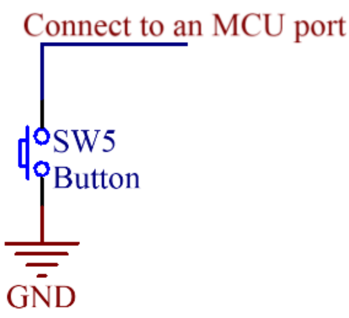


Figure 1.1: Connecting a button to an MCU

Figure 1.1 shows a way to connect a button to an MCU. This hardware operates as follows:

- When the switch is open, it has no impact on the port pin. An internal resistor on the port “pulls up" the pin to the supply voltage of the MCU (typically 3.3V for STM32F103). If we read the pin, we will see the value ‘1’.
- When the switch is closed (pressed), the pin voltage will be 0V. If we read the pin, we will see the value ‘0’.

However, if the MCU does not have a pull-up resistor inside, when the button is pressed, the read value will be ‘0’, but even we release the button, the read value is still ‘0’ as shown in Figure 1.2.

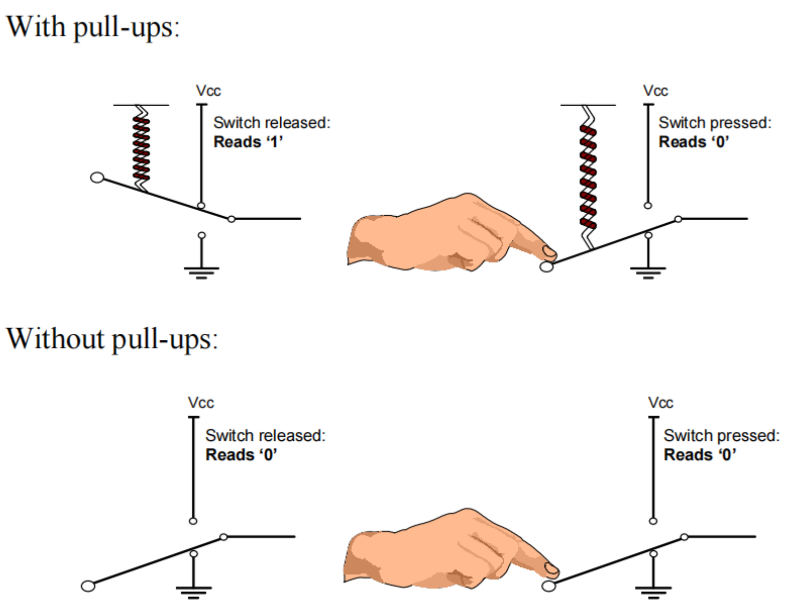


Figure 1.2: The need of pull up resistors

So a reliable way to connect a button/switch to an MCU is that we explicitly use an external pull-up resistor as shown in Figure 1.3.

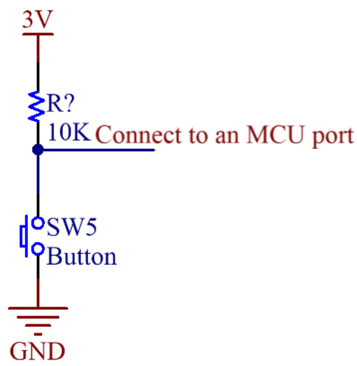


Figure 1.3: A reliable way to connect a button to an MCU

3.2 Dealing with switch bounces

In practice, all mechanical switch contacts bounce (that is, turn on and off, repeatedly, for short period of time) after the switch is closed or opened as shown in Figure 1.4.

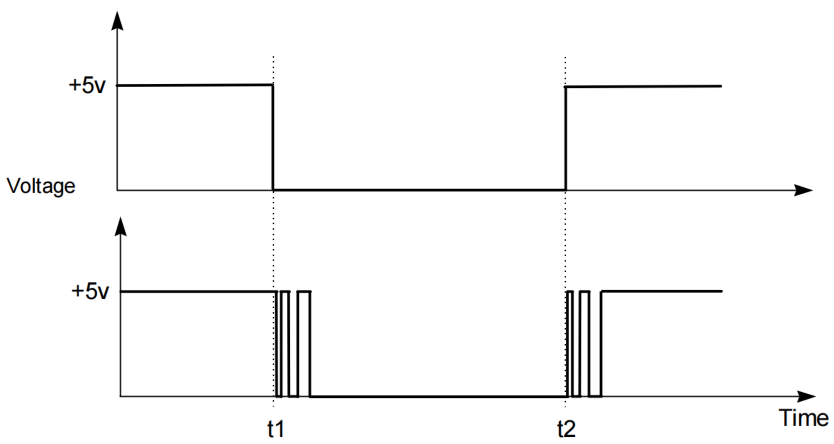


Figure 1.4: Switch bounces

Every system that uses any kind of mechanical switch must deal with the issue of debouncing. The key task is to make sure that one mechanical switch or button action is only read as one action by the MCU, even though the MCU will typically be fast enough to detect the unwanted switch bounces and treat them as separate events. Bouncing can be eliminated by special ICs or by RC circuitry, but in most cases debouncing is done in software because software is “free".
As far as the MCU concerns, each “bounce" is equivalent to one press and release of an “ideal” switch. Without appropriate software design, this can give several problems:

- Rather than reading ‘A’ from a keypad, we may read ‘AAAAA’
- Counting the number of times that a switch is pressed becomes extremely difficult
- If a switch is depressed once, and then released some time later, the ‘bounce’ may make it appear as if the switch has been pressed again (at the time of release).

The key to debouncing is to establish a minimum criterion for a valid button push, one that can be implemented in software. This criterion must involve differences in time - two button presses in 20ms must be treated as one button event, while two button presses in 2 seconds must be treated as two button events. So what are the relevant times we need to consider? They are these:

- Bounce time: most buttons seem to stop bouncing within 10ms
- Button press time: the shortest time a user can press and release a button seems to be between 50 and 100ms
- Response time: a user notices if the system response is 100ms after the button press, but not if it is 50ms after

Combining all of these times, we can set a few goals

- Ignore all bouncing within 10ms
- Provide a response within 50ms of detecting a button push (or release)
- Be able to detect a 50ms push and a 50ms release

The simplest debouncing method is to examine the keys (or buttons or switches) every N milliseconds, where N > 10ms (our specified button bounce upper limit) and N <= 50ms (our specified response time). We then have three possible outcomes every time we read a button:

- We read the button in the solid '0' state
- We read the button in the solid '1' state
- We read the button while it is bouncing (so we will get either a '0' or a '1')

Outcomes 1 and 2 pose no problems, as they are what we would always like to happen. Outcome 3 also poses no problem because during a bounce either state is acceptable. If we have just pressed an active-low button and we read a '1' as it bounces, the next time through we are guaranteed to read a '0' (remember, the next time through all bouncing will have ceased), so we will just detect the button push a bit later. Otherwise, if we read a '0' as the button bounces, it will still be '0' the next time after all bouncing has stopped, so we are just detecting the button push a bit earlier. The same applies to releasing a button. Reading a single bounce (with all bouncing over by the time of the next read) will never give us an invalid button state. It's only reading multiple bounces (multiple reads while bouncing is occurring) that can give invalid button states such as repeated push signals from one physical push.
So if we guarantee that all bouncing is done by the time we next read the button, we're good. Well, almost good, if we're lucky...
MCUs often live among high-energy beasts, and often control the beasts. High energy devices make electrical noise, sometimes great amounts of electrical noise. This noise can, at the worst possible moment, get into your delicate button-and-high-value-pullup circuit and act like a real button push. Oops, missile launched, sorry!
If the noise is too intense we cannot filter it out using only software, but will need hardware of some sort (or even a redesign). But if the noise is only occasional, we can filter it out in software without too much bother. The trick is that instead of regarding a single button 'make' or 'break' as valid, we insist on N contiguous makes or breaks to mark a valid button event. N will be a factor of your button scanning rate and the amount of

filtering you want to add. Bigger N gives more filtering. The simplest filter (but still a big improvement over no filtering) is just an N of 2, which means compare the current button state with the last button state, and only if both are the same is the output valid.
Note that now we have not two but three button states: active (or pressed), inactive (or released), and indeterminate or invalid (in the middle of filtering, not yet filtered). In most cases we can treat the invalid state the same as the inactive state, since we care in most cases only about when we go active (from whatever state) and when we cease being active (to inactive or invalid). With that simplification we can look at simple N = 2 filtering reading a button wired to STM32 MCU:

```c
void button_reading(void){
    static unsigned char last_button;
    unsigned char raw_button;
    unsigned char filtered_button;
    last_button = raw_button;
    raw_button = HAL_GPIO_ReadPin(BUTTON_1_GPIO_Port,
   BUTTON_1_Pin);
    if(last_button == raw_button){
        filtered_button = raw_button;
    }
}
```

Program 1.1: Read port pin and deboucing

The function button_reading() must be called no more often than our debounce time (10ms).
To expand to greater filtering (larger N), keep in mind that the filtering technique essentially involves reading the current button state and then either counting or reseting the counter. We count if the current button state is the same as the last button state, and if our count reaches N we then report a valid new button state. We reset the counter if the current button state is different than the last button state, and we then save the current button state as the new button state to compare against the next time. Also note that the larger our value of N the more often our filtering routine must be called, so that we get a filtered response within our specified 50ms deadline. So for example with an N of 8 we should be calling our filtering routine every 2 - 5ms, giving a response time of 16 - 40ms (>10ms and <50ms).

4 Reading switch input (basic code) using STM32

To demonstrate the use of buttons/switches in STM32, we use an example which requires to write a program that

- Has a timer which has an interrupt in every 10 milliseconds.
- Reads values of button PB0 every 10 milliseconds.
- Increases the value of LEDs connected to PORTA by one unit when the button PB0 is pressed.
- Increases the value of PORTA automatically in every 0.5 second, if the button PB0 is pressed in more than 1 second.

4.1 Input Output Processing Patterns

For both input and output processing, we have a similar pattern to work with. Normally, we have a module named driver which works directly to the hardware. We also have a buffer to store temporarily values. In the case of input processing, the driver will store the value of the hardware status to the buffer for further processing. In the case of output processing, the driver uses the buffer data to output to the hardware.

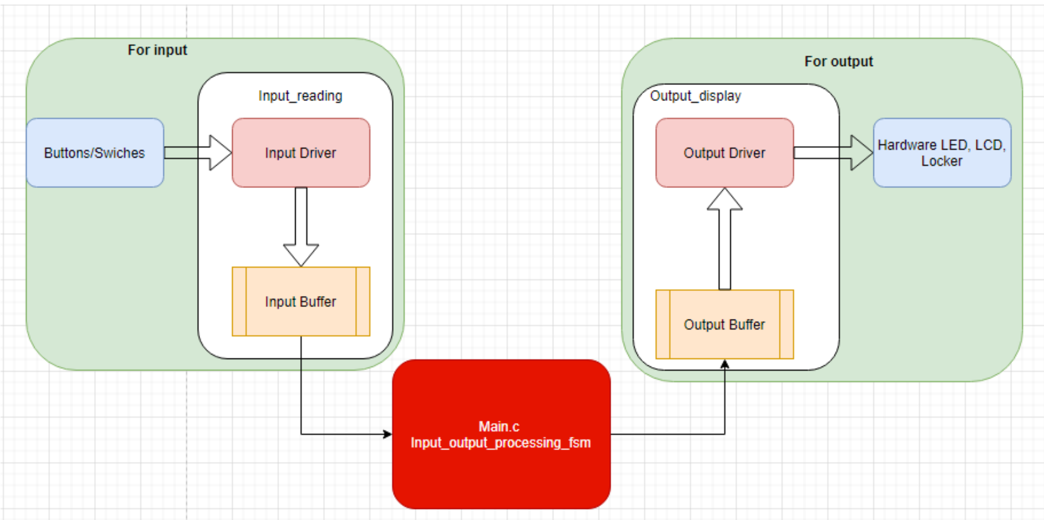


Figure 1.5: Input Output Processing Patterns

Figure 1.5 shows that we should have an *input_reading* module to processing the buttons, then store the processed data to the buffer. Then a module of *input_output_processing_fsm* will process the input data, and update the output buffer. The output driver gets the value from the output buffer to transfer to the hardware.

4.2 Setting up

4.2.1 Create a project

Please follow the instruction in Labs 1 and 2 to create a project that includes:

- PB0 as an input port pin,
- PA0-PA7 as output port pins, and
- Timer 2 10ms interrupt

4.2.2 Create a file C source file and header file for input reading

We are expected to have files for button processing and led display as shown in Figure 1.6.

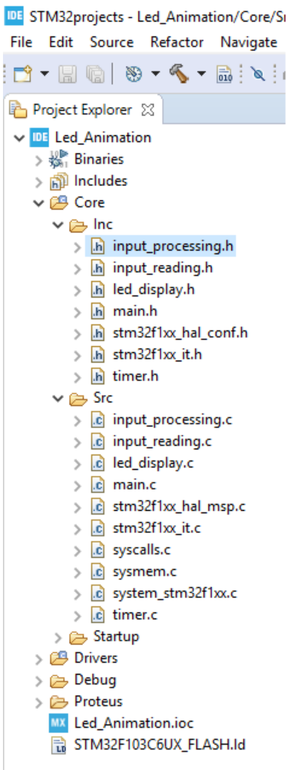


Figure 1.6: File Organization

Steps 1 (Figure 1.7): Right click to the folder **Src**, select **New**, then select **Source File**. There will be a pop-up. Please type the file name, then click **Finish**.
Step 2 (Figure 1.8): Do the same for the C header file in the folder **Inc**.

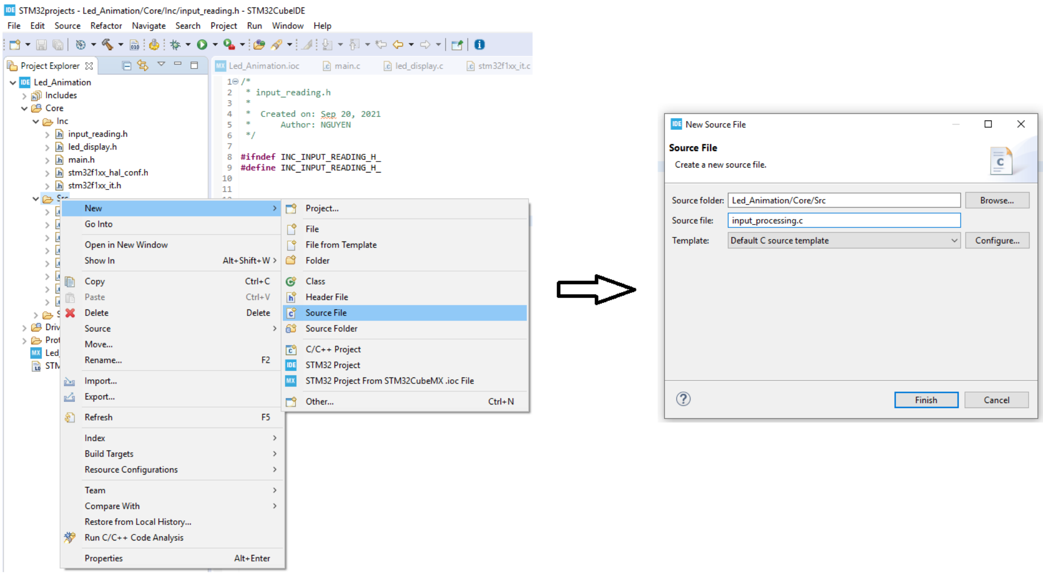


Figure 1.7: Step 1: Create a C source file for input reading

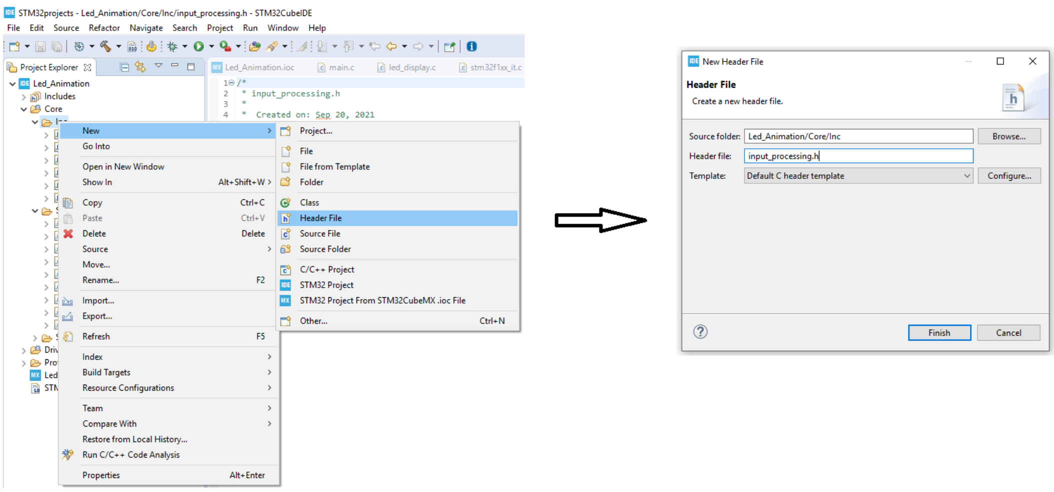


Figure 1.8: Step 2: Create a C header file for input processing

4.3 Code For Read Port Pin and Debouncing

4.3.1 The code in the input_reading.c file

```
#include "main.h"
//we aim to work with more than one buttons
#define NO_OF_BUTTONS                1
//timer interrupt duration is 10ms, so to pass 1 second,
//we need to jump to the interrupt service routine 100 time
#define DURATION_FOR_AUTO_INCREASING     100
#define BUTTON_IS_PRESSED                  GPIO_PIN_RESET
#define BUTTON_IS_RELEASED                 GPIO_PIN_SET
//the buffer that the final result is stored after
//debouncing
static GPIO_PinState buttonBuffer[NO_OF_BUTTONS];
//we define two buffers for debouncing
static GPIO_PinState debounceButtonBuffer1[NO_OF_BUTTONS];
static GPIO_PinState debounceButtonBuffer2[NO_OF_BUTTONS];
//we define a flag for a button pressed more than 1 second.
static uint8_t flagForButtonPress1s[NO_OF_BUTTONS];
//we define counter for automatically increasing the value
//after the button is pressed more than 1 second.
static uint16_t counterForButtonPress1s[NO_OF_BUTTONS];
void button_reading(void){
  for(char i = 0; i < NO_OF_BUTTONS; i ++){
    debounceButtonBuffer2[i] =debounceButtonBuffer1[i];
    debounceButtonBuffer1[i] = HAL_GPIO_ReadPin(
   BUTTON_1_GPIO_Port, BUTTON_1_Pin);
    if(debounceButtonBuffer1[i] == debounceButtonBuffer2[i
   ])
      buttonBuffer[i] = debounceButtonBuffer1[i];
      if(buttonBuffer[i] == BUTTON_IS_PRESSED){
      //if a button is pressed, we start counting
        if(counterForButtonPress1s[i] <
   DURATION_FOR_AUTO_INCREASING){
          counterForButtonPress1s[i]++;
        } else {
        //the flag is turned on when 1 second has passed
        //since the button is pressed.
          flagForButtonPress1s[i] = 1;
          //todo
        }
      } else {
        counterForButtonPress1s[i] = 0;
        flagForButtonPress1s[i] = 0;
      }
  }
}
```

Program 1.2: Define constants buffers and button_reading function

```
unsigned char is_button_pressed(uint8_t index){
  if(index >= NO_OF_BUTTONS) return 0;
  return (buttonBuffer[index] == BUTTON_IS_PRESSED);
}
```

Program 1.3: Checking a button is pressed or not

```
unsigned char is_button_pressed_1s(unsigned char index){
  if(index >= NO_OF_BUTTONS) return 0xff;
  return (flagForButtonPress1s[index] == 1);
}
```

Program 1.4: Checking a button is pressed more than a second or not

4.3.2 The code in the input_reading.h file

```
#ifndef INC_INPUT_READING_H_
#define INC_INPUT_READING_H_
void button_reading(void);
unsigned char is_button_pressed(unsigned char index);
unsigned char is_button_pressed_1s(unsigned char index);
#endif /* INC_INPUT_READING_H_ */
```

Program 1.5: Prototype in input_reading.h file

4.3.3 The code in the timer.c file

```
#include "main.h"
#include "input_reading.h"

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  if(htim->Instance == TIM2){
    button_reading();
  }
}
```

Program 1.6: Timer interrupt callback function

4.4 Button State Processing

4.4.1 Finite State Machine

To solve the example problem, we define 3 states as follows:

- State 0: The button is released or the button is in the initial state.
- State 1: When the button is pressed, the FSM will change to State 1 that is increasing the values of PORTA by one value. If the button is released, the FSM goes back to State 0.
- State 2: while the FSM is in State 1, the button is kept pressing more than 1 second, the state of FSM will change from 1 to 2. In this state, if the button is kept pressing, the value of PORTA will be increased automatically in every 500ms. If the button is released, the FSM goes back to State 0.

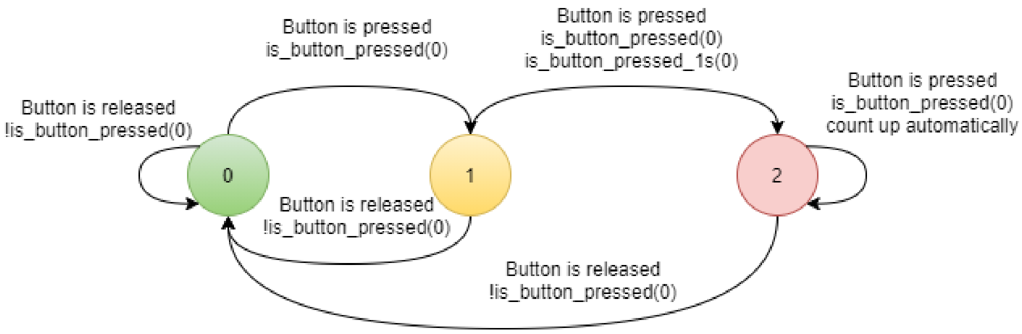


Figure 1.9: An FSM for processing a button

4.4.2 The code for the FSM in the input_processing.c file

Please note that *fsm_for_input_processing* function should be called inside the super loop of the main functin.

```c
#include "main.h"
#include "input_reading.h"

enum ButtonState{BUTTON_RELEASED, BUTTON_PRESSED,
  BUTTON_PRESSED_MORE_THAN_1_SECOND} ;
enum ButtonState buttonState = BUTTON_RELEASED;
void fsm_for_input_processing(void){
  switch(buttonState){
  case BUTTON_RELEASED:
    if(is_button_pressed(0)){
      buttonState = BUTTON_PRESSED;
      //INCREASE VALUE OF PORT A BY ONE UNIT
    }
    break;
  case BUTTON_PRESSED:
    if(!is_button_pressed(0)){
      buttonState = BUTTON_RELEASED;
    } else {
      if(is_button_pressed_1s(0)){
        buttonState = BUTTON_PRESSED_MORE_THAN_1_SECOND;
      }
    }
    break;
  case BUTTON_PRESSED_MORE_THAN_1_SECOND:
    if(!is_button_pressed(0)){
      buttonState = BUTTON_RELEASED;
    }
    //todo
    break;
  }
}
```

Program 1.7: The code in the input_processing.c file

4.4.3 The code in the input_processing.h

```c
#ifndef INC_INPUT_PROCESSING_H_
#define INC_INPUT_PROCESSING_H_

void fsm_for_input_processing(void);

#endif /* INC_INPUT_PROCESSING_H_ */
```

Program 1.8: Code in the input_processing.h file

4.4.4 The code in the main.c file

```c
#include "main.h"
#include "input_processing.h"
//don't modify this part
int main(void){
    HAL_Init();
    /* Configure the system clock */
    SystemClock_Config();
    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_TIM2_Init();
    while (1)
    {
        //you only need to add the fsm function here
        fsm_for_input_processing();
    }
}
```

Program 1.9: The code in the main.c file

5 Exercises and Report

5.1 Specifications

You are required to build an application of a traffic light in a cross road which includes some features as described below:

- The application has 12 LEDs including 4 red LEDs, 4 amber LEDs, 4 green LEDs.
- The application has 4 seven segment LEDs to display time with 2 for each road. The 2 seven segment LEDs will show time for each color LED corresponding to each road.
- The application has three buttons which are used

  - to select modes,

  - to modify the time for each color led on the fly, and

  - to set the chosen value.
- The application has at least 4 modes which is controlled by the first button. Mode 1 is a normal mode, while modes 2 3 4 are modification modes. You can press the first button to change the mode. Modes will change from 1 to 4 and back to 1 again.

  Mode 1 - Normal mode:

  - The traffic light application is running normally.

  Mode 2 - Modify time duration for the red LEDs: This mode allows you to change the time duration of the red LED in the main road. The expected behaviours of this mode include:

  - All single red LEDs are blinking in 2 Hz.

  - Use two seven-segment LEDs to display the value.

  - Use the other two seven-segment LEDs to display the mode.

  - The second button is used to increase the time duration value for the red LEDs.

  - The value of time duration is in a range of 1 - 99.

  - The third button is used to set the value.

  Mode 3 - Modify time duration for the amber LEDs: Similar for the red LEDs described above with the amber LEDs.

  Mode 4 - Modify time duration for the green LEDs: Similar for the red LEDs described above with the green LEDs.

5.2 Exercise 1: Sketch an FSM

Your task in this exercise is to sketch an FSM that describes your idea of how to solve the problem.
Please add your report here.

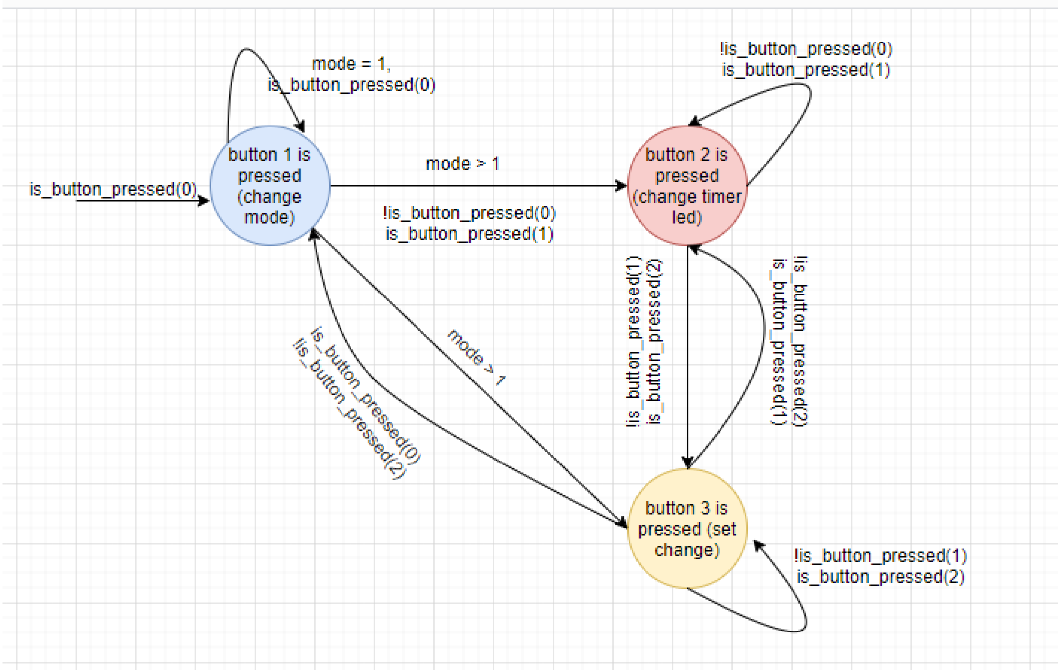


Figure 1.10: FSM to solve problem.

5.3 Exercise 2: Proteus Schematic

Your task in this exercise is to draw a Proteus schematic for the problem above.

PA1 - PA7 is output port pin - connect first LED 7SEGMENT
PA8 - PA14 is output port pin - connect third LED 7SEGMENT
PA 15 is output port pin - connect Led red.
PB0 - PB6 is output port pin - connect second and fourth LED 7 SEGMENT.
PB7 - PB12 is output port pin - connect in turn horizontal's led red,green, amber and vertical's led red, green, amber.
PB13 - PB15 is input port pin - connect in turn each button. Please add your report here.

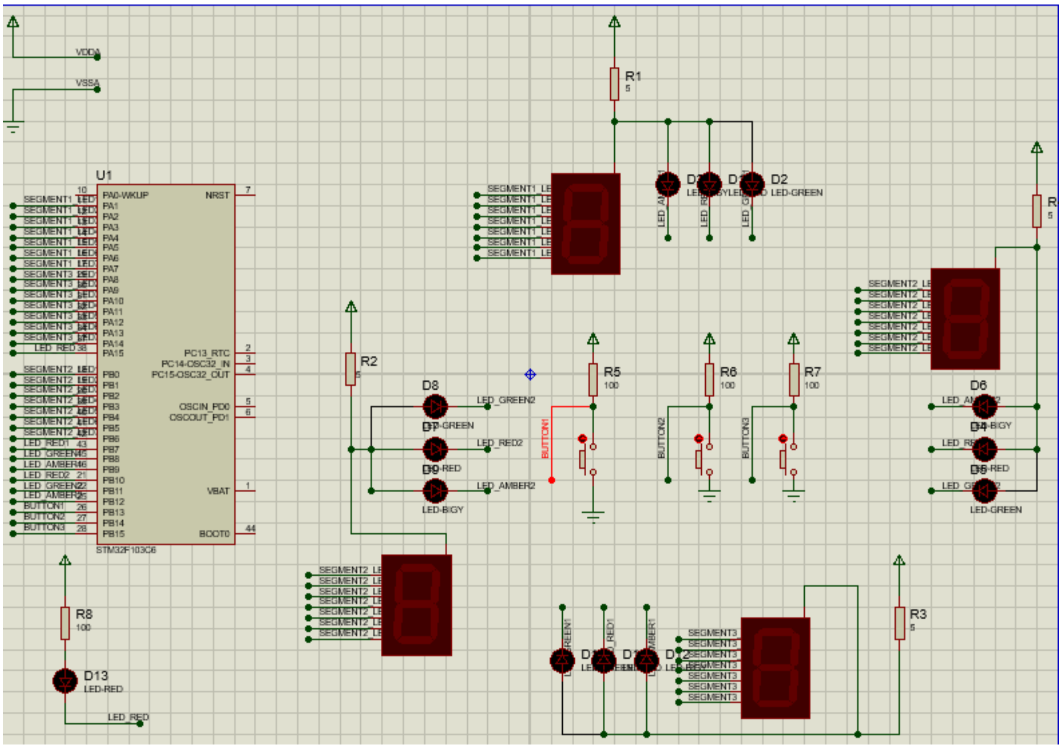


Figure 1.11: Schematic for the problem.

5.4 Exercise 3: Create STM32 Project

Your task in this exercise is to create a project that has pin corresponding to the Proteus schematic that you draw in previous section. You need to set up your timer interrupt is about 10ms.
Please add your report here.

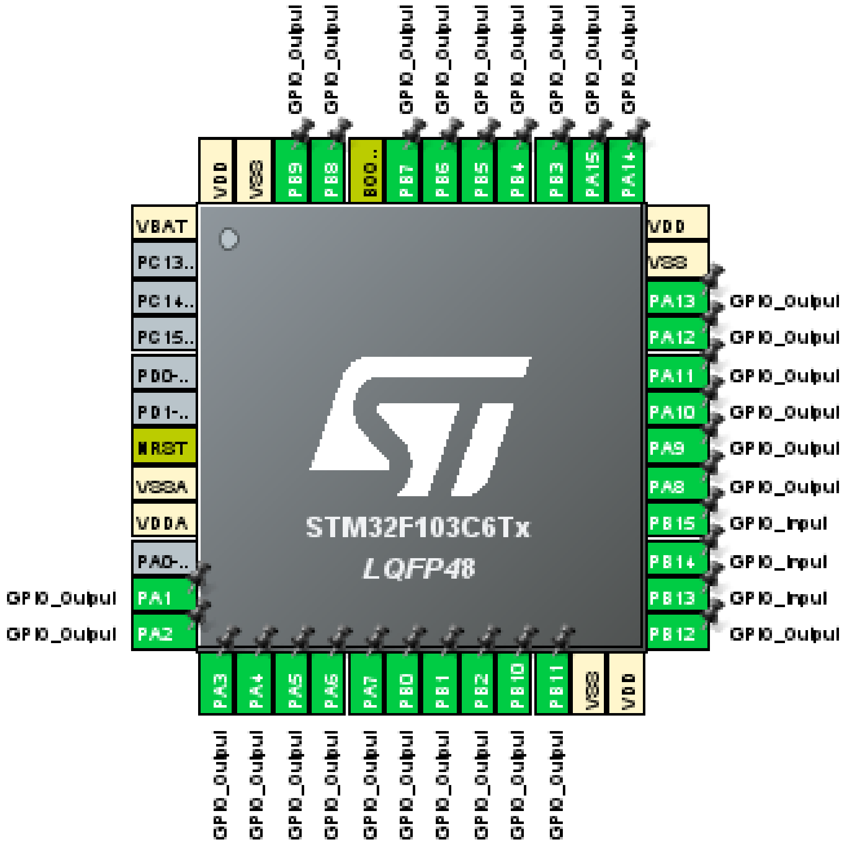


Figure 1.12: Set up port corresponding to Proteus schematic in Exercise 2.

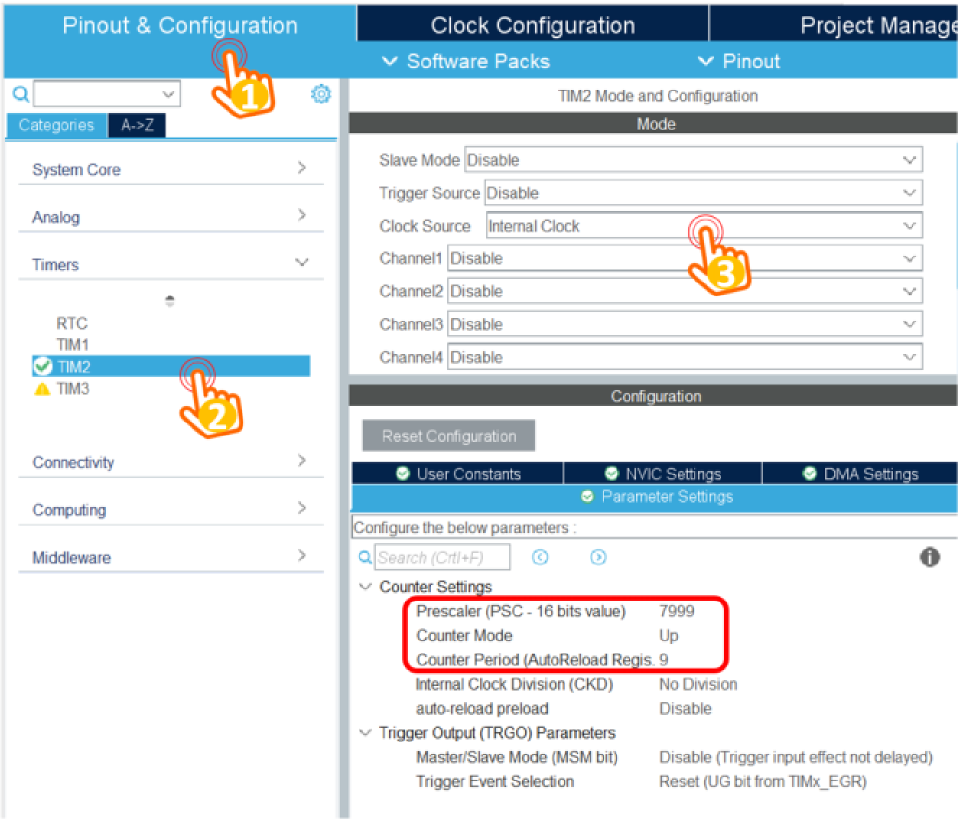


Figure 1.13: Step to set up timer interrupt is about 10ms.

5.5 Exercise 4: Modify Timer Parameters

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 in 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.
When changing the timer interrupt later, we can avoid opening code and just changing it by setting some parameter.

```
#ifndef INC_TIMER_H_
#define INC_TIMER_H_
#include "main.h"
#define TIMER_INTERUPT     10   // 10 miliseconds

#define TIMER_1s          1000 //1000 miliseconds = 1seconds
static int count_1s = TIMER_1s / TIMER_INTERUPT;

#define TIMER_50ms        50 //50 miliseconds
static int count_50ms = TIMER_50ms / TIMER_INTERUPT;

#define TIMER_blinking      500 //500 miliseconds = 0.5
   seconds
static int count_500ms = TIMER_blinking / TIMER_INTERUPT;
#endif /* INC_TIMER_H_ */
```

Program 1.10: the code in the timer.h

5.6 Exercise 5: Adding code for button debouncing

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.

Ideas: Create 4 file include: **input_reading.c**, **input_reading.h** (two file will handle the state of button), **input_processing.c** and **input_processing.h** (two file will handle for example turn on led, and increase time).

5.6.1 The code in the input_reading.c file

Ideas: Firstly, create a function **preprocessing(void)** to set default properties before process run. **button_reading(void)** to set state for button. After that, declare each function we create in **input_reading.c** to file **input_reading.h**

```
#include "main.h"
#include "input_reading.h"
#include "output_display.h"
```

```
GPIO_PinState buttonBuffer [ NO_OF_BUTTONS ];
// we define two buffers for
GPIO_PinState debounceButtonBuffer1 [ NO_OF_BUTTONS ];
GPIO_PinState debounceButtonBuffer2 [ NO_OF_BUTTONS ];

uint16_t pinBuffer[NO_OF_BUTTONS];

// we define a flag for a button pressed more than 1 second
static uint8_t flagForButtonPress1s [ NO_OF_BUTTONS ];
// we define counter for automatically increasing the value
// after the button is pressed more than 1 second .
static uint16_t counterForButtonPress1s [ NO_OF_BUTTONS ];
//Default set when process run
void preprocessing(void){
  pinBuffer[0] = GPIO_PIN_13;
  pinBuffer[1] = GPIO_PIN_14;
  pinBuffer[2] = GPIO_PIN_15;
  default_setting();
  for(int i = 0 ; i < NO_OF_BUTTONS ; i++){
    debounceButtonBuffer1[i] = BUTTON_IS_RELEASED;
    debounceButtonBuffer2[i]= BUTTON_IS_RELEASED;
    buttonBuffer[i]= BUTTON_IS_RELEASED;
  }
}
void button_reading(void){
  for(char i = 0; i < NO_OF_BUTTONS; i ++){
    debounceButtonBuffer2[i] =debounceButtonBuffer1[i];
    debounceButtonBuffer1 [i] = HAL_GPIO_ReadPin (GPIOB, pinBuffer[i]);
    if(debounceButtonBuffer1[i] == debounceButtonBuffer2[i])
      buttonBuffer[i] = debounceButtonBuffer1[i];
      if(buttonBuffer[i] == BUTTON_IS_PRESSED){
      //if a button is pressed, we start counting
        if(counterForButtonPress1s[i] < DURATION_FOR_AUTO_INCREASING){
          counterForButtonPress1s[i]++;
        } else {
        //the flag is turned on when 1 second has passed
        //since the button is pressed.
          flagForButtonPress1s[i] = 1;
          //todo
        }
      } else {
        counterForButtonPress1s[i] = 0;
        flagForButtonPress1s[i] = 0;
      }
  }
}
```

```c
unsigned char is_button_pressed(uint8_t index){
  if(index >= NO_OF_BUTTONS) return 0;
  return (buttonBuffer[index] == BUTTON_IS_PRESSED);
}
unsigned char is_button_pressed_1s(unsigned char index){
  if(index >= NO_OF_BUTTONS) return 0xff;
  return (flagForButtonPress1s[index] == 1);
}
```

Program 1.11: Define constants buffers and button_reading function

5.6.2 The code in the input_reading.h file

```
//the buffer that the final result is stored after
# define NO_OF_BUTTONS 3

# define DURATION_FOR_AUTO_INCREASING 100
# define BUTTON_IS_PRESSED     GPIO_PIN_RESET
# define BUTTON_IS_RELEASED    GPIO_PIN_SET

void preprocessing(void);
void button_reading ( void );
int getButton (int index);
unsigned char is_button_pressed ( unsigned char index );
unsigned char is_button_pressed_1s ( unsigned char index );
```

Program 1.12: Prototype in input_reading.h file

When the first button is pressed mode will be transform. **Ideas:** In file input_processing.c,

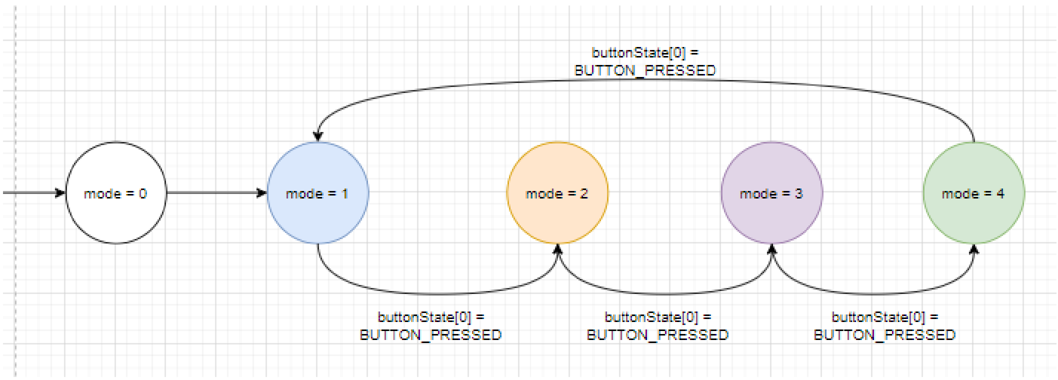


Figure 1.14: Step to set up timer interrupt is about 10ms.

create a variable mode to point out the MODE we want to performance. Beside that, create a function **void increase_mode()** to transform mode when the first button is pressed , increase a function **int MODE()** to take value mode (we will use this function in next Exercises).In **void fsm_for_input_processing (void)**,create a for loop for three button, when the first button (i = 0) is pressed the Led connected to PA15 will turn on (to show the button was pressed), call **void increase_mode()** to transform mode, call function **void reset_mode()** to guaranty when we suddenly transform Mode the time duration of led will be reset with original values.

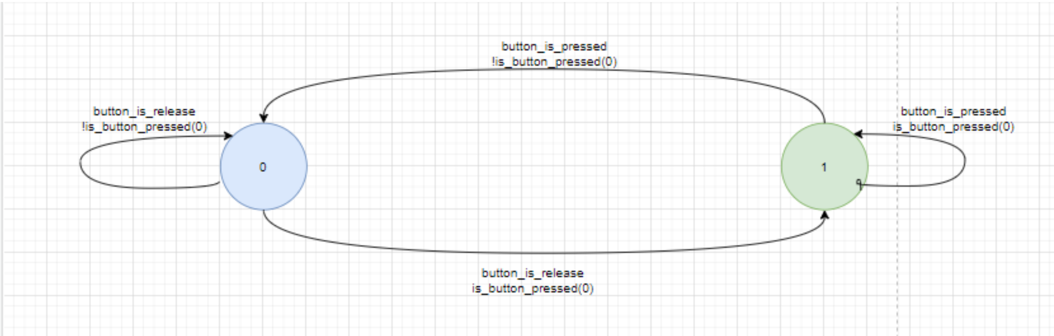


Figure 1.15: Step to set up timer interrupt is about 10ms.

5.6.3 The code in input_processing.c

```
void increase_mode(){
  if(mode > 4 ) mode = 1;
  else mode++;
}
```

Program 1.13: Source code in function increase_mode()

```
void reset_mode(){ //reset time_duration after transform to a mode and reset timer each when we didn't complete time's run cycle at mode 1
  for(int i = 0; i < 3 ; i++){
    horizontal_arr[i] = arr2[i];
    vertical_arr[i] = arr2[i];
  }
  time_duration = 0;
}
```

Program 1.14: Source code in function reset_mode()

```
#include "main.h"
#include "input_processing.h"
#include "input_reading.h"
#include "output_display.h"

enum ButtonState { BUTTON_RELEASED , BUTTON_PRESSED , BUTTON_PRESSED_MORE_THAN_1_SECOND } ;
enum ButtonState buttonState[3] = {BUTTON_RELEASED, BUTTON_RELEASED, BUTTON_RELEASED};
int time_duration = 0;
int mode = 0;
void fsm_for_input_processing ( void ){
  for(int i = 0 ; i < NO_OF_BUTTONS; i++){
    switch ( buttonState[i] ){
      case BUTTON_RELEASED :
        if( is_button_pressed (i)){
          buttonState[i] = BUTTON_PRESSED ;
          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, RESET);
```

```
            // INCREASE VALUE OF PORT A BY ONE UNIT
            if(i == 0){ //first button.
              increase_mode();
              if(mode > 1)reset_mode();
            }
          }
          break ;
        case BUTTON_PRESSED :
          if (! is_button_pressed (i)){
            buttonState[i] = BUTTON_RELEASED ;
             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, SET);
          } else {
            if( is_button_pressed_1s (i) )
              buttonState[i] =
   BUTTON_PRESSED_MORE_THAN_1_SECOND ;
          }
          break ;
        case BUTTON_PRESSED_MORE_THAN_1_SECOND :
          if (! is_button_pressed (i)){
            buttonState[i] = BUTTON_RELEASED;
             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, SET);
          }
          // todo
          break ;
        }
  }
  Mode();
}
```

Program 1.15: Source code in function fsm_for_input_processing (void)

5.6.4 The code in input_processing.h

```
#ifndef INC_INPUT_PROCESSING_H_
#define INC_INPUT_PROCESSING_H_

void fsm_for_input_processing ( void );
int Mode();

#endif /* INC_INPUT_PROCESSING_H_ */
```

Program 1.16: source code in file input_processing.h

5.7 Exercise 6: Adding code for displaying modes

Your tasks in this exercise are:

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

Ideas: Create three array vertical_arr[3], horizontal_arr[3], arr2[3] to hold time duration Leds, create a function **default_setting(void)** to get value of vertical_arr[3].Create sequence 4 functions **mode_1()**,**mode_2()**,**mode_3()**,**mode_4()** to display four Mode. In mode 1, display normal reuse code in Lab1.In mode 2,3,4 create a global variable counter_mode with value = 1 (the counter_mode = 1 will turn on, counter_mode = 2 will turn off) and turn off remains four Leds.

5.7.1 The code in output_display.c

```c
#include "main.h"
#include "output_display.h"
int vertical_arr[3] = {5,2,3};
int horizontal_arr[3] = {5,2,3};
int arr2[3] = {0,0,0}; //temp arr
void default_setting(void){
  for(int i = 0; i < 3 ; i++){
    arr2[i] = vertical_arr[i];
  }
}

void display7SEG(int num,GPIO_TypeDef* GPIO1, uint16_t GPIO_PIN0,
         uint16_t GPIO_PIN1, uint16_t GPIO_PIN2, uint16_t GPIO_PIN3,
         uint16_t GPIO_PIN4, uint16_t GPIO_PIN5, uint16_t GPIO_PIN6)
{
  switch(num){
  case 0:
     HAL_GPIO_WritePin(GPIO1, GPIO_PIN0, RESET);
     HAL_GPIO_WritePin(GPIO1, GPIO_PIN1, RESET);
     HAL_GPIO_WritePin(GPIO1, GPIO_PIN2, RESET);
     HAL_GPIO_WritePin(GPIO1, GPIO_PIN3, RESET);
     HAL_GPIO_WritePin(GPIO1, GPIO_PIN4, RESET);
     HAL_GPIO_WritePin(GPIO1, GPIO_PIN5, RESET);
     HAL_GPIO_WritePin(GPIO1, GPIO_PIN6, SET);
    break;
  case 1:
     HAL_GPIO_WritePin(GPIO1, GPIO_PIN0, SET);
     HAL_GPIO_WritePin(GPIO1, GPIO_PIN1, RESET);
     HAL_GPIO_WritePin(GPIO1, GPIO_PIN2, RESET);
     HAL_GPIO_WritePin(GPIO1, GPIO_PIN3, SET);
     HAL_GPIO_WritePin(GPIO1, GPIO_PIN4, SET);
```

```
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN5, SET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN6, SET);
    break;
  case 2:
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN0, RESET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN1, RESET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN2, SET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN3, RESET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN4, RESET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN5, SET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN6, RESET);
    break;
  case 3:
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN0, RESET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN1, RESET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN2, RESET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN3, RESET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN4, SET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN5, SET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN6, RESET);
    break;
  case 4:
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN0, SET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN1, RESET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN2, RESET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN3, SET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN4, SET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN5, RESET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN6, RESET);
    break;
  case 5:
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN0, RESET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN1, SET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN2, RESET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN3, RESET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN4, SET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN5, RESET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN6, RESET);
    break;
  case 6:
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN0, RESET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN1, SET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN2, RESET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN3, RESET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN4, RESET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN5, RESET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN6, RESET);
    break;
  case 7:
```

```
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN0, RESET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN1, RESET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN2, RESET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN3, SET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN4, SET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN5, SET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN6, SET);
    break;
  case 8:
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN0, RESET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN1, RESET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN2, RESET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN3, RESET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN4, RESET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN5, RESET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN6, RESET);
    break;
  case 9:
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN0, RESET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN1, RESET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN2, RESET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN3, RESET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN4, SET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN5, RESET);
      HAL_GPIO_WritePin(GPIO1, GPIO_PIN6, RESET);
    break;
  }
}

void traffic_light(uint16_t GPIO_Pin1,uint16_t GPIO_Pin2,
   uint16_t GPIO_Pin3,int count, int arr[]){
  switch(count){
  case 0:
      HAL_GPIO_WritePin(GPIOB,GPIO_Pin1, RESET);
      HAL_GPIO_WritePin(GPIOB,GPIO_Pin2, SET);
      HAL_GPIO_WritePin(GPIOB,GPIO_Pin3, SET);
    break;
  case 1:
       HAL_GPIO_WritePin(GPIOB,GPIO_Pin1, SET);
       HAL_GPIO_WritePin(GPIOB,GPIO_Pin2, SET);
       HAL_GPIO_WritePin(GPIOB,GPIO_Pin3, RESET);
       break;
  case 2:
      HAL_GPIO_WritePin(GPIOB,GPIO_Pin1, SET);
      HAL_GPIO_WritePin(GPIOB,GPIO_Pin2, RESET);
      HAL_GPIO_WritePin(GPIOB,GPIO_Pin3, SET);
      break;
  }
  arr[count]--;
```

```
}
void traffic_light_4(int count_vled,int count_hled){
    ///////////////////////////////HORIZONTAL LEDS /////////////////////////////
    traffic_light(GPIO_PIN_7,GPIO_PIN_8, GPIO_PIN_9, count_hled, horizontal_arr);

    ////////////////////////////VERTICAL LEDS //////////////////////////
    traffic_light(GPIO_PIN_10, GPIO_PIN_11, GPIO_PIN_12, count_vled,vertical_arr);
}

/////////////////////////////////MODE 1///////////////////////////////////////////
int count_vled = 0;
int count_hled = 2;
void mode_1(){
  if(count_vled >= 3) count_vled = 0;
  if(count_hled < 0) count_hled = 2;
    traffic_light_4(count_vled, count_hled);
    display7SEG(vertical_arr[count_vled], GPIOB, GPIO_PIN_0 ,GPIO_PIN_1, GPIO_PIN_2, GPIO_PIN_3, GPIO_PIN_4, GPIO_PIN_5, GPIO_PIN_6);
    display7SEG(horizontal_arr[count_hled], GPIOA, GPIO_PIN_1,GPIO_PIN_2, GPIO_PIN_3, GPIO_PIN_4, GPIO_PIN_5, GPIO_PIN_6, GPIO_PIN_7);
    display7SEG(horizontal_arr[count_hled], GPIOA, GPIO_PIN_8, GPIO_PIN_9, GPIO_PIN_10, GPIO_PIN_11, GPIO_PIN_12, GPIO_PIN_13, GPIO_PIN_14);
    if(vertical_arr[count_vled] <= 1){
      vertical_arr[count_vled] = arr2[count_vled];
      count_vled++;
    }
    if(horizontal_arr[count_hled] <=  1){
      horizontal_arr[count_hled] = arr2[count_hled];
      count_hled--;
    }
}
```

Program 1.17: source code in output_display.c

Create a function **void mode_2()** to display led red and display value 2 in two vertical LED 7 SEGMENT. To guaranty timer turn on of the led RED is 500ms, create a **int counter_mode2** , when **int counter_mode2 = 1** ,led RED is connect PB7 and PB10 will turn all,when **int counter_mode2 = 2** led RED is connect PB7 and PB10 will turn off.

```
////////////////////////// MODE 2 ////////////////////////////////
int counter_mode2 = 1;
```

```c
void mode_2(){
  if(counter_mode2 == 1){
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7| GPIO_PIN_10, RESET);
    counter_mode2 = 2;
  }
  else {
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7| GPIO_PIN_10, SET);
    counter_mode2 = 1;
  }
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8| GPIO_PIN_9
              | GPIO_PIN_11| GPIO_PIN_12, SET);
  display7SEG(2, GPIOB, GPIO_PIN_0, GPIO_PIN_1, GPIO_PIN_2, GPIO_PIN_3, GPIO_PIN_4, GPIO_PIN_5, GPIO_PIN_6);
}
```

Program 1.18: Source code of function mode_2()

Similary, create a function **void mode_3()** with implementation function like **void mode_2()**.But, instead of led RED is connected port PB7 and PB10, we replace by led amber is connect PB9 and PB12.

```c
//////////////MODE 3//////////////////
int counter_mode3 = 1;
void mode_3(){
  if(counter_mode3 == 1){
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9| GPIO_PIN_12, RESET);
    counter_mode3 = 2;
  }
  else {
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9| GPIO_PIN_12, SET);
    counter_mode3 = 1;
  }
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7| GPIO_PIN_8
              | GPIO_PIN_10| GPIO_PIN_11, SET);
  display7SEG(3, GPIOB, GPIO_PIN_0, GPIO_PIN_1, GPIO_PIN_2, GPIO_PIN_3, GPIO_PIN_4, GPIO_PIN_5, GPIO_PIN_6);
}
```

Program 1.19: Source code of function mode_3()

Similary, create a function **void mode_4()** with implementation function like **void mode_2()**.But, instead of led RED is connected port PB7 and PB10, we replace by led amber is connect PB8 and PB11.

```c
///////////////MODE 4////////////////
int counter_mode4 = 1;
void mode_4(){
  if(counter_mode4 == 1){
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8| GPIO_PIN_11, RESET);
    counter_mode4 = 2;
```

```
  }
  else {
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8| GPIO_PIN_11, SET);
    counter_mode4 = 1;
  }
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7| GPIO_PIN_9
              | GPIO_PIN_10| GPIO_PIN_12, SET);
  display7SEG(4, GPIOB, GPIO_PIN_0, GPIO_PIN_1, GPIO_PIN_2,
    GPIO_PIN_3, GPIO_PIN_4, GPIO_PIN_5, GPIO_PIN_6);
}
```

Program 1.20: Source code of function mode_4()

5.7.2 The code in output_display.h

```

#ifndef INC_OUTPUT_DISPLAY_H_
#define INC_OUTPUT_DISPLAY_H_

int vertical_arr[3];
int horizontal_arr[3];
int arr2[3];
void display7SEG(int num,GPIO_TypeDef* GPIO1, uint16_t
   GPIO_PIN0,
         uint16_t GPIO_PIN1, uint16_t GPIO_PIN2, uint16_t
   GPIO_PIN3,
         uint16_t GPIO_PIN4, uint16_t GPIO_PIN5, uint16_t
   GPIO_PIN6);
void traffic_light(uint16_t GPIO_Pin1,uint16_t GPIO_Pin2,
   uint16_t GPIO_Pin3,int count, int arr[]);
void traffic_light_4(int count_vled,int count_hled);
void mode_1();
void mode_2();
void mode_3();
void mode_4();
void default_setting(void);
#endif /* INC_OUTPUT_DISPLAY_H_ */
```

Program 1.21: source code in output_display.h

5.7.3 The code in timer.c

```
#include "main.h"
#include "timer.h"
#include "output_display.h"
#include "input_reading.h"
int count = 0;
int mode_button;
void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef *
   htim ){
```

```
    count--;
    button_reading();
    mode_button = Mode();
    if(count <= 0){
      if(mode_button == 1){
        mode_1();
        count = count_1s;
      }
      if(mode_button == 2){
        mode_2();
        count = count_500ms;
      }
      if(mode_button == 3){
        mode_3();
        count = count_500ms;
      }
      if(mode_button == 4){
        mode_4();
        count = count_500ms;
      }
    }
}
```

Program 1.22: Source code in timer.c

5.8 Exercise 7: Adding code for increasing time duration value for the red LEDs

Your tasks in this exercise 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.

Please add your report here. **Ideas:** In function **preprocessing(void)**, set N0_OF_BUTTONS with value 3 and set pin_Buffer with port PB14, PB15. Create a global variable time_duration with value = 0, when the sencond button (i = 1) time_duration will increase 1 unit. The

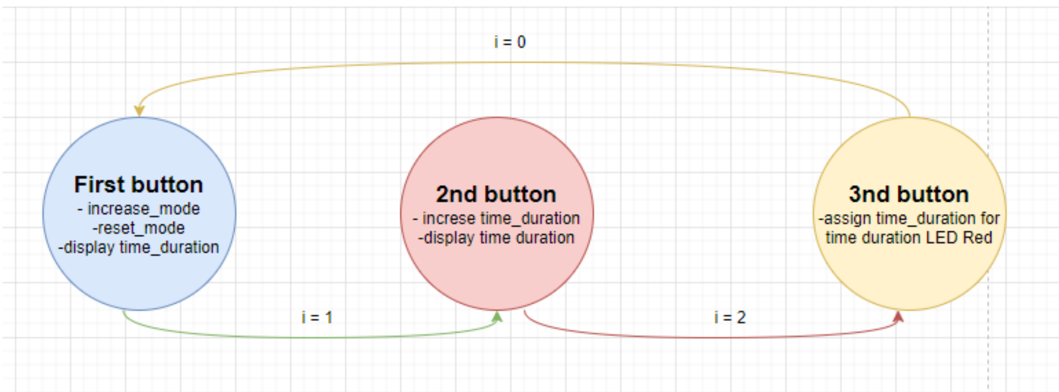


Figure 1.16: Summary of the code's operating cycle.

source code will be use in **Exercise 7,8,9**.

```c
#include "main.h"
#include "input_processing.h"
#include "input_reading.h"
#include "output_display.h"

enum ButtonState { BUTTON_RELEASED , BUTTON_PRESSED , BUTTON_PRESSED_MORE_THAN_1_SECOND } ;
enum ButtonState buttonState[3] = {BUTTON_RELEASED, BUTTON_RELEASED, BUTTON_RELEASED};
int time_duration = 0;
int mode = 0;
int Mode() {
  return mode;
}
void increase_mode(){
  if(mode > 4 ) mode = 1;
  else mode++;
}
void reset_mode(){ //reset time_duration after transform to a mode and reset timer each when we didn't complete time's run cycle at mode 1
  for(int i = 0; i < 3 ; i++){
    horizontal_arr[i] = arr2[i] + 1;
    vertical_arr[i] = arr2[i] + 1;
  }
  time_duration = 0;
}
void fsm_for_input_processing ( void ){
  for(int i = 0 ; i < NO_OF_BUTTONS; i++){
    switch ( buttonState[i] ){
      case BUTTON_RELEASED :
        if( is_button_pressed (i)){
          buttonState[i] = BUTTON_PRESSED ;
          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, RESET);
          // INCREASE VALUE OF PORT A BY ONE UNIT
          if(i == 0){ //first button.
            increase_mode();
            reset_mode();
            if(mode > 0){
              display7SEG(time_duration%10, GPIOA, GPIO_PIN_8, GPIO_PIN_9, GPIO_PIN_10, GPIO_PIN_11, GPIO_PIN_12, GPIO_PIN_13, GPIO_PIN_14);
              display7SEG(time_duration/10, GPIOA, GPIO_PIN_1, GPIO_PIN_2, GPIO_PIN_3, GPIO_PIN_4, GPIO_PIN_5, GPIO_PIN_6, GPIO_PIN_7);
            }
          }
```

```
          else if(i == 1){ //second button.
            if(time_duration < 100)time_duration++;
            else time_duration = 0;
            display7SEG(time_duration%10, GPIOA, GPIO_PIN_8
   , GPIO_PIN_9, GPIO_PIN_10, GPIO_PIN_11, GPIO_PIN_12,
   GPIO_PIN_13, GPIO_PIN_14);
            display7SEG(time_duration/10, GPIOA, GPIO_PIN_1
   , GPIO_PIN_2, GPIO_PIN_3, GPIO_PIN_4, GPIO_PIN_5,
   GPIO_PIN_6, GPIO_PIN_7);
          }
          else if(i == 2){ //third button.
            if(mode > 1)arr2[mode - 2] = time_duration;
          }
        }
        break;
      case BUTTON_PRESSED:
        if (! is_button_pressed(i)){
          buttonState[i] = BUTTON_RELEASED;
           HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, SET);
        } else {
          if( is_button_pressed_1s(i) )
            buttonState[i] =
   BUTTON_PRESSED_MORE_THAN_1_SECOND;
        }
        break;
      case BUTTON_PRESSED_MORE_THAN_1_SECOND:
        if (! is_button_pressed(i)){
          buttonState[i] = BUTTON_RELEASED;
           HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, SET);
        }
        // todo
        break;
      }
  }
  Mode();
}
```

Program 1.23: source code input_processing.c after add time duration value for each LEDS.

5.9 Exercise 8: Adding code for increasing time duration value for the amber LEDs

Your tasks in this exercise 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.

Please add your report here. **Ideas:** In function **preprocessing(void)**, set N0_OF_BUTTONS with value 3 and set pin_Buffer with port PB14, PB15. Create a global variable time_duration with value = 0, when the sencond button (i = 1) time_duration will increase 1 unit. Source

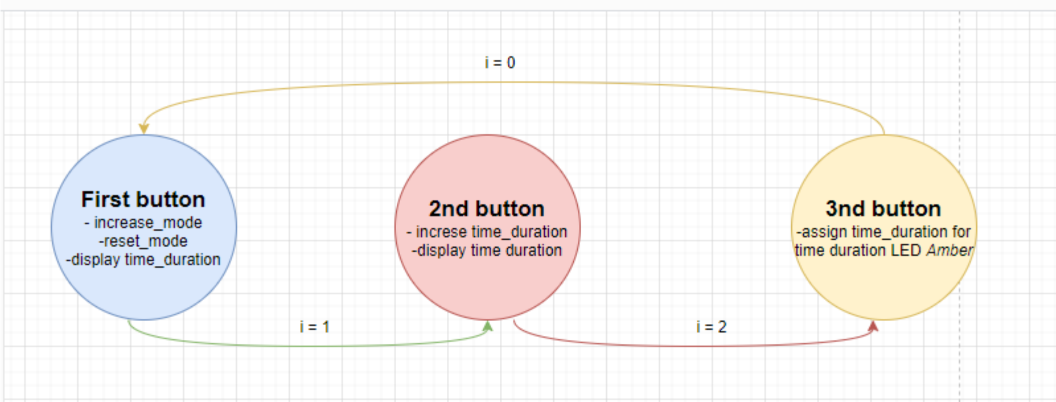


Figure 1.17: Summary of the code's operating cycle.

code was show in **Exercise 7**.

5.10 Exercise 9: Adding code for increasing time duration value for the green LEDs

Your tasks 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.

Please add your report here. **Ideas:** In function **preprocessing(void)**, set N0_OF_BUTTONS with value 3 and set pin_Buffer with port PB14, PB15. Create a global variable time_duration with value = 0, when the sencond button (i = 1) time_duration will increase 1 unit. Source

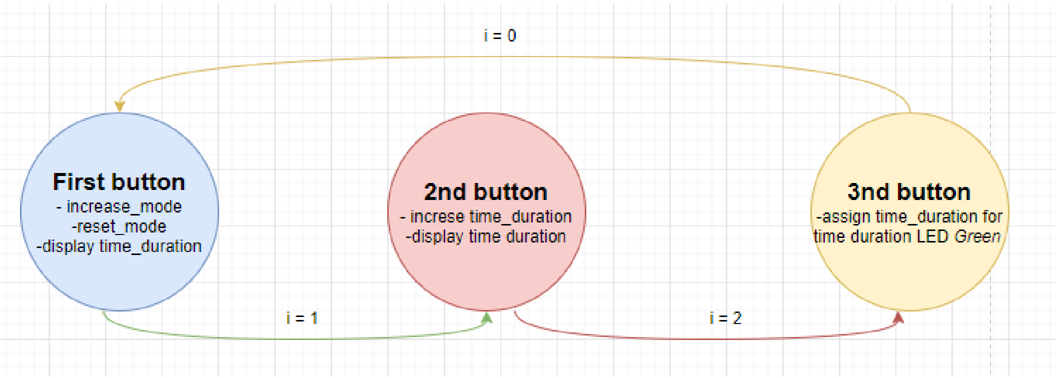


Figure 1.18: Summary of the code's operating cycle.

code was show in **Exercise 7**.

5.11 Exercise 10: To finish the project

Your tasks 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