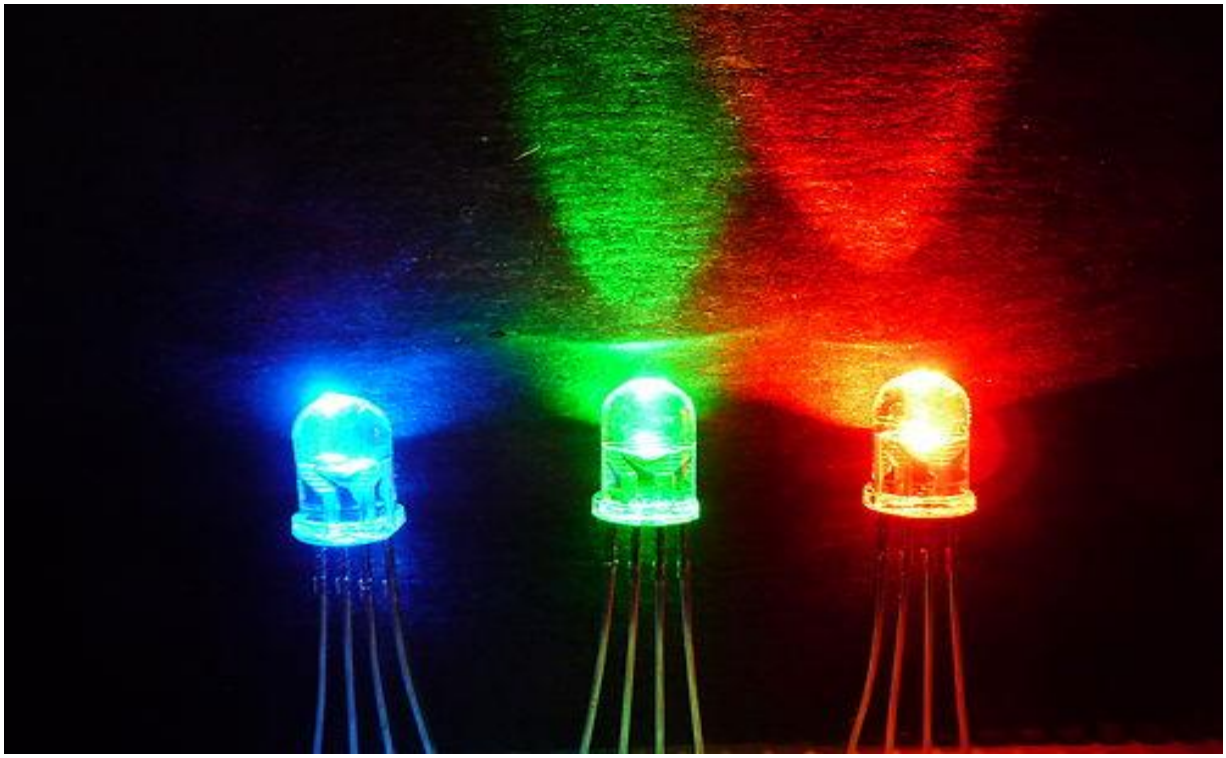
RGB LED Control V3.0 Design



Anas Mahmoud	Team 2
Khaled Mustafa	Team 2

Table of Contents

1-Description

1.1 Hardware components

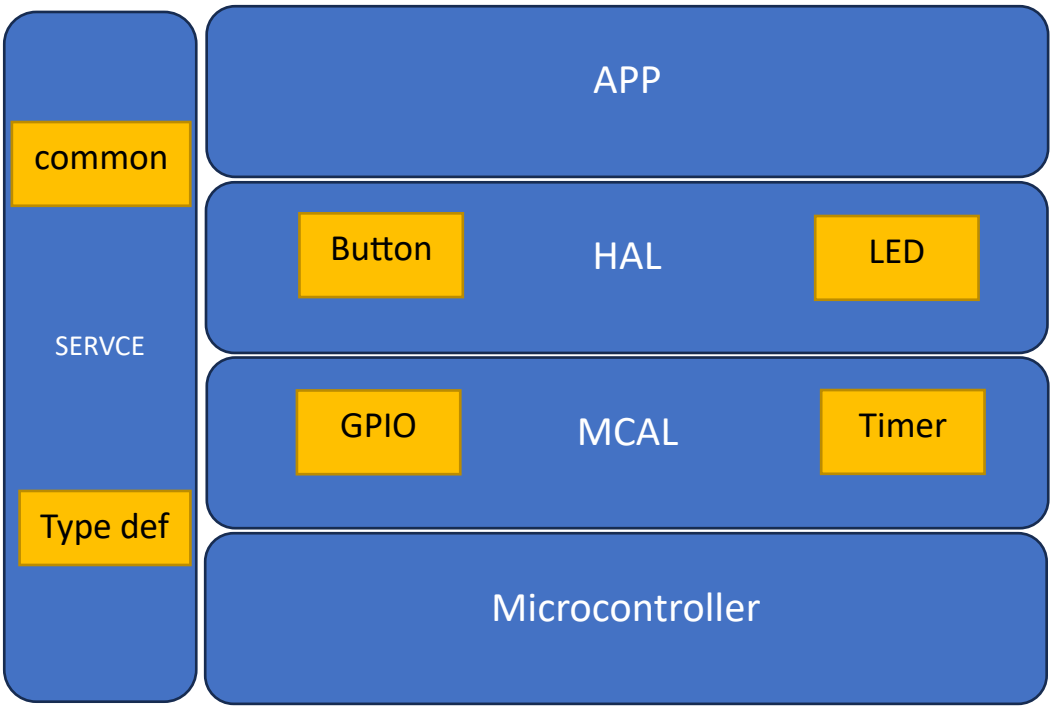
1. Use the TivaC board.
2. Use SW1 as an input button.
3. Use the RGB LED

1.2 Software Requirements

1. The RGB LED is OFF initially
2. The PWM signal has a 500ms duration
3. The system has four states
    1. SW1 - First press
        1. The Green LED will be on with a 30% duty cycle
    2. SW1 - Second press
        1. The Green LED will be on with a 60% duty cycle
    3. SW1 -Third press
        1. The Green LED will be on with a 90% duty cycle
    4. SW1 - Fourth press will be off
        1. The Green LED will be off
    5. On the fifth press, system state will return to state 1

2-High Level Design

2.1 Layered Architecture



2.2 Drivers Descriptions

2.2.1 GPIO Driver

Function: MCAL

Function: used to set pin direction (input or output), pin value (high or low) or read a value from a pin or toggle a pin

2.2.2 Button Driver

Location: HAL

Function: used to initialize the button, check the button status pressed or not

2.2.3 LED Driver

Location: HAL

Function: used to initialize the LEDs, describe the way of working the LEDs

2.2.4 TIMER Driver

Location: MCAL

Function: used to make a time delay

2.2.5 Application Driver

Location: App

Function: Combine between the driver's API's to meet the requirement

2.3 Modules API's

2.3.1 GPIO Module

```
/**
 * @brief Initializes a set of pins with the specified configuration.
 * @param[in] ptr_st_GPIO_config    Address of the array of the specified pins.
 * @return en_GPIO_error_t
 */
en_GPIO_error_t GPIO_Init(const st_GPIO_config_t *ptr_st_GPIO_config);
/**
 * @brief Reads the state of a single pin.
 * @param[in] ptr_st_GPIO_config    Address of the specified pin configuration strcuture.
 * @param[in] ptr_pinValue          Address of the value where we stores the state of the pin.
 * @return en_GPIO_error_t
 */
en_GPIO_error_t GPIO_ReadPin(const st_GPIO_config_t *ptr_st_GPIO_config, uint8_t *ptr_pinValue);

/**
 * @brief Reads the state of a single pin.
 * @param[in] ptr_st_GPIO_config    Address of the specified pin configuration strcuture.
 * @param[in] pinValue              The value to be written on the specified pin.
 * @return en_GPIO_error_t
 */
en_GPIO_error_t GPIO_WritePin(const st_GPIO_config_t *ptr_st_GPIO_config, uint8 pinValue);
```

2.3.2 Button Module

```
/**
 * @brief Initializes the button pin.
 * @param[in] Button_config     Address of the configuration array.
 */
void BUTTON_Init(st_GPIO_config_t *Button_config);

/**
 * @brief Initializes the button pin.
 * @param[in] usr_buttonConfig Address of the configuration array.
 * @param[in] value        Address of the variables to store the state of push button.
 */
void BUTTON_IsPressed( st_GPIO_config_t *usr_buttonConfig, en_button_state_t *value);
```

2.3.3 LED Module

```
/**
 * @brief Initializes the LED pin using the configuration array.
 * @param[in] ptr_st_LED_config     Address of the configuration array.
 * @return en_GPIO_error_t
 */
void Led_Init(st_GPIO_config_t *ptr_st_LED_config);

/**
 * @brief Turns on the red LED and turns off all the other LEDs.
 * @param[in] ptr_st_LED_config     Address of the configuration array.
 * @return en_GPIO_error_t
 */
void LED_GreenOn(st_GPIO_config_t *ptr_st_LED_config);
```

```
/**
 * @brief Turns on the blue LED and turns off all the other LEDs.
 * @param[in] ptr_st_LED_config     Address of the configuration array.
 * @return en_GPIO_error_t
 */
void LED_BlueOn(st_GPIO_config_t *ptr_st_LED_config);

/**
 * @brief Turns on the green LED and turns off all the other LEDs.
 * @param[in] ptr_st_LED_config     Address of the configuration array.
 * @return en_GPIO_error_t
 */
void LED_RedOn(st_GPIO_config_t *ptr_st_LED_config);

/**
 * @brief Turns on all the LEDs.
 * @param[in] ptr_st_LED_config     Address of the configuration array.
 * @return en_GPIO_error_t
 */
void LED_AllOn(st_GPIO_config_t *ptr_st_LED_config);

/**
 * @brief Turns off all the LEDs.
 * @param[in] ptr_st_LED_config     Address of the configuration array.
 * @return en_GPIO_error_t
 */
void LED_AllOff(st_GPIO_config_t *ptr_st_LED_config);
```

2.3.4 Timer Module

```
void TIMER_Init(st_TIMER_config_t *ptr_st_TIMER_config);

en_TIMER_Status_t TIMER_GetStatus(en_TIMER_TimerID_t en_TIMER_TimerID);

void TIMER_ClearTimeoutFlag(en_TIMER_TimerID_t en_TIMER_TimerID);

void TIMER_DelayMS(st_TIMER_config_t *ptr_st_TIMER_config, uint32 Period);
```

2.3.5 App Module

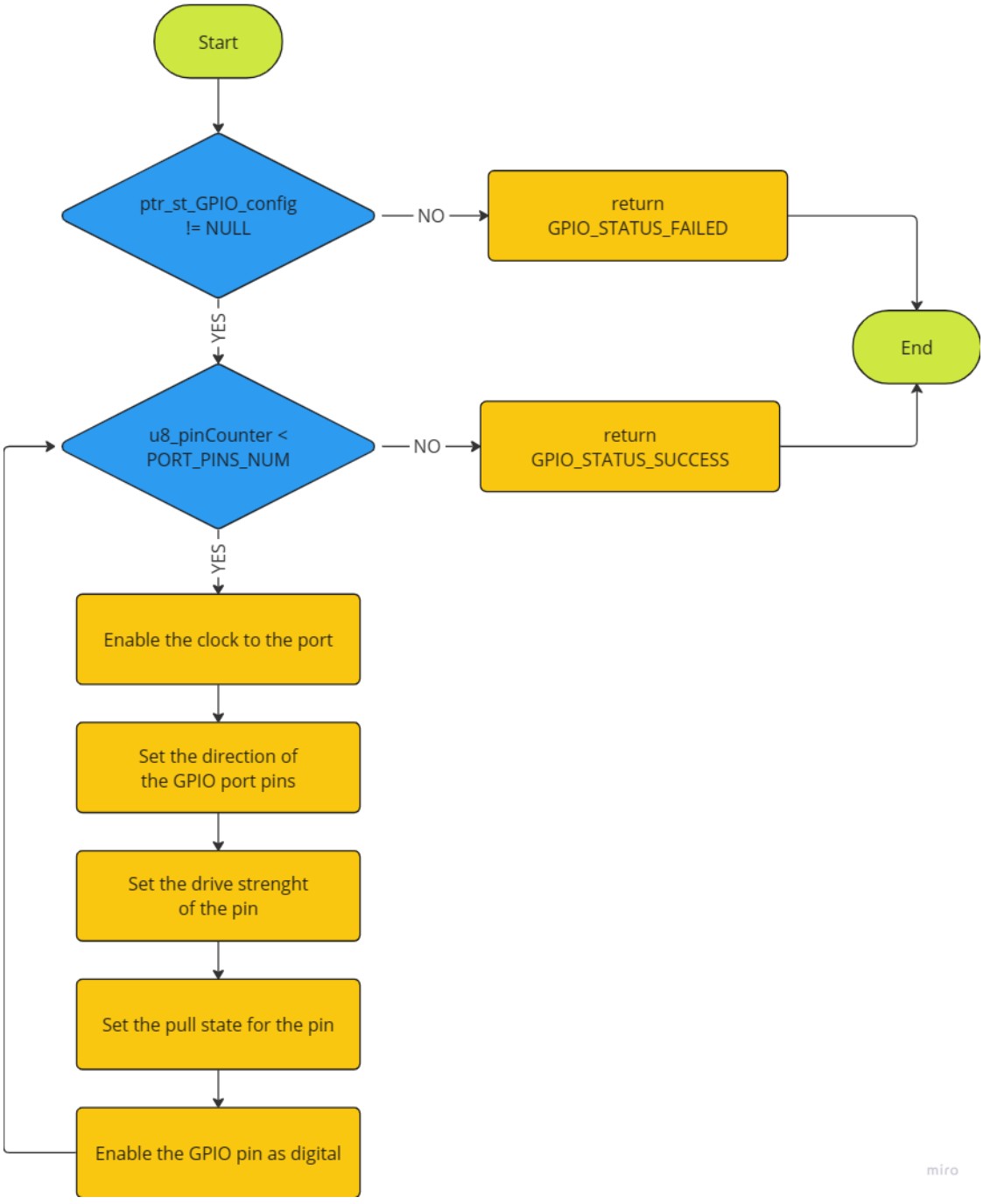
```
/**
 * @brief Initializes all the ECUAL.
 */
void APP_Init(void);

/**
 * @brief Starts the application logic.
 */
void APP_Start(void);
```

3-Low Level Design

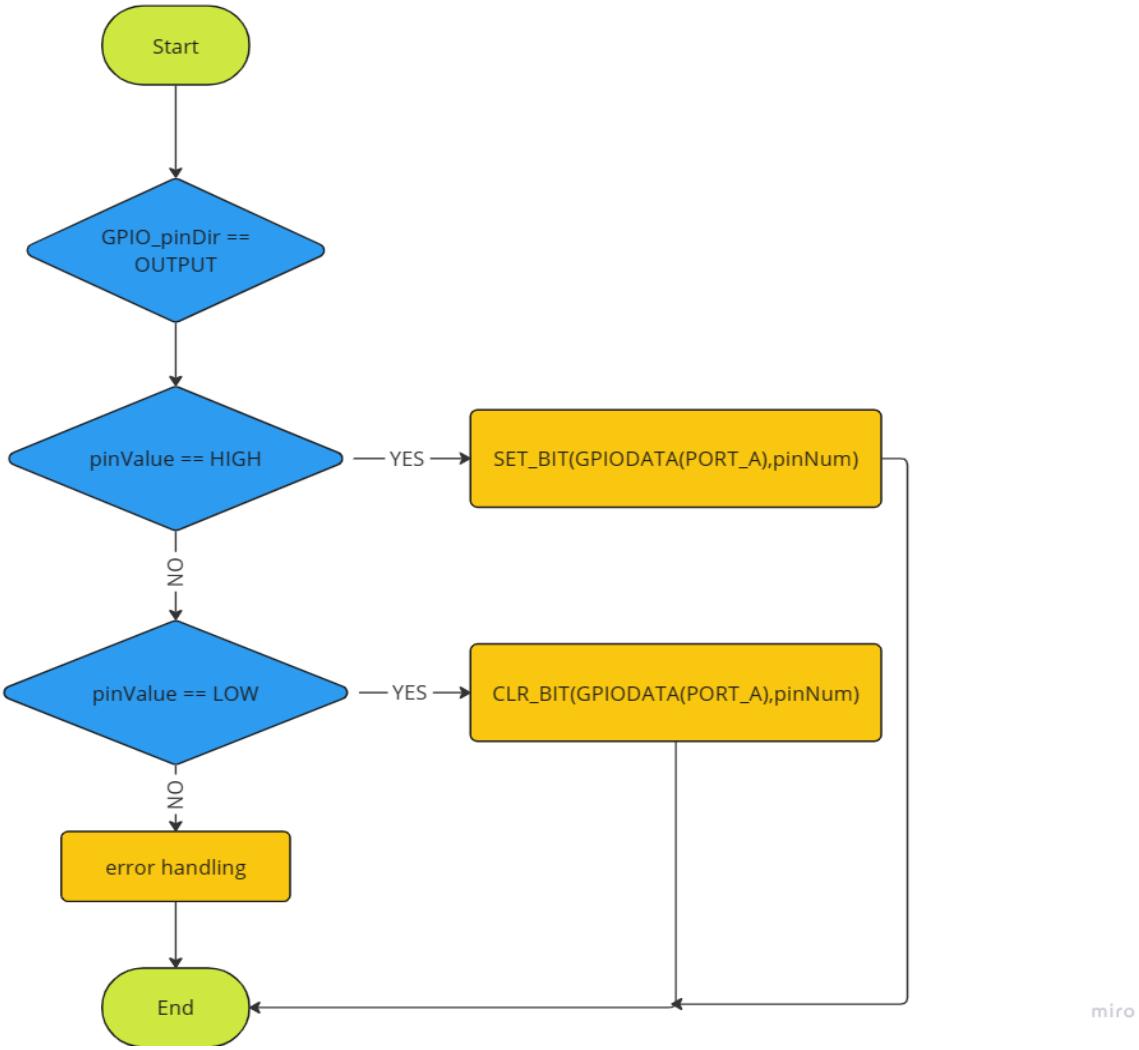
3.1 APIs Flow Chart

3.1.1 GPIO Module

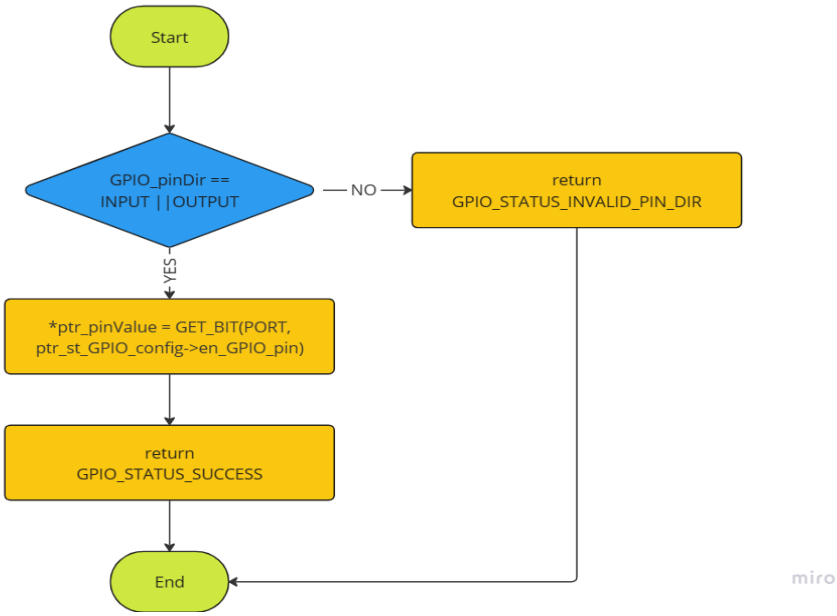
en_GPIO_error_t GPIO_Init(const st_GPIO_config_t *ptr_st_GPIO_config)



en_GPIO_error_t GPIO_WritePin(const st_GPIO_config_t *ptr_st_GPIO_config, uint8 pinValue)

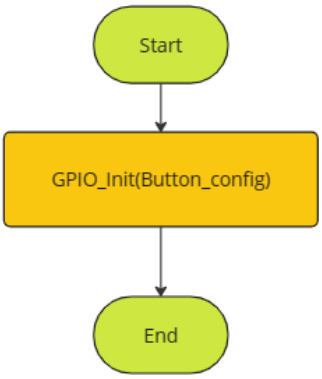


en_GPIO_error_t GPIO_ReadPin(const st_GPIO_config_t *ptr_st_GPIO_config, uint8_t *ptr_pinValue)

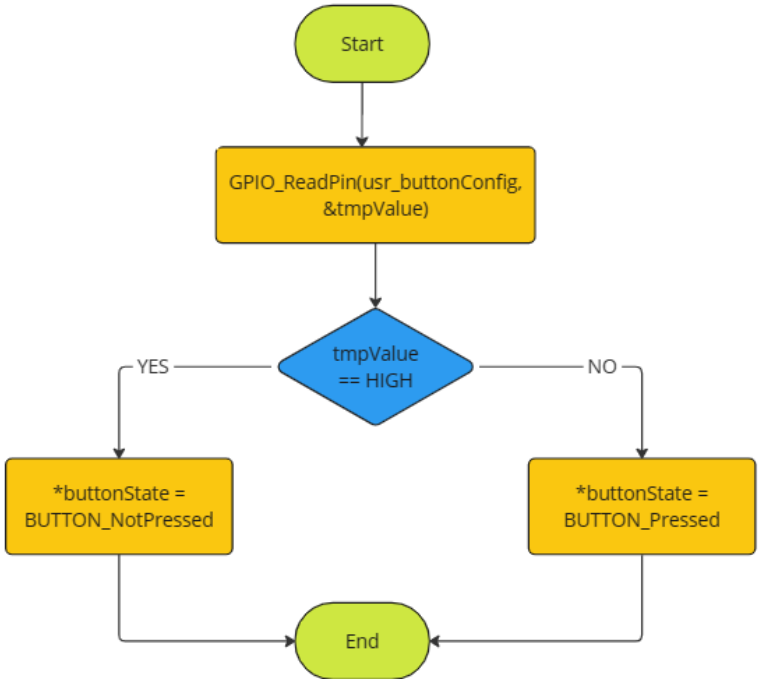


3.1.2 Button Module

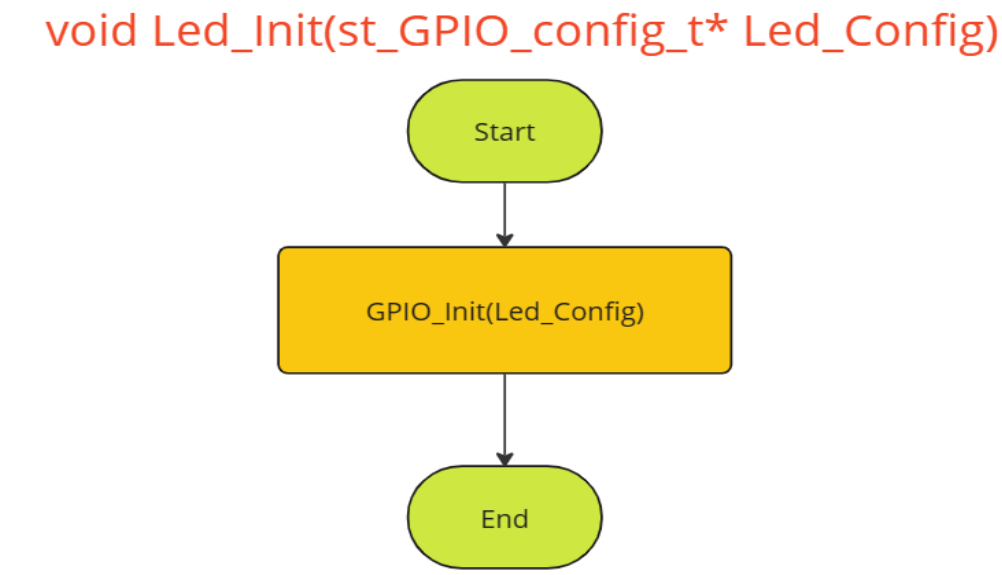
void button_init(st_GPIO_config_t* Button_config)

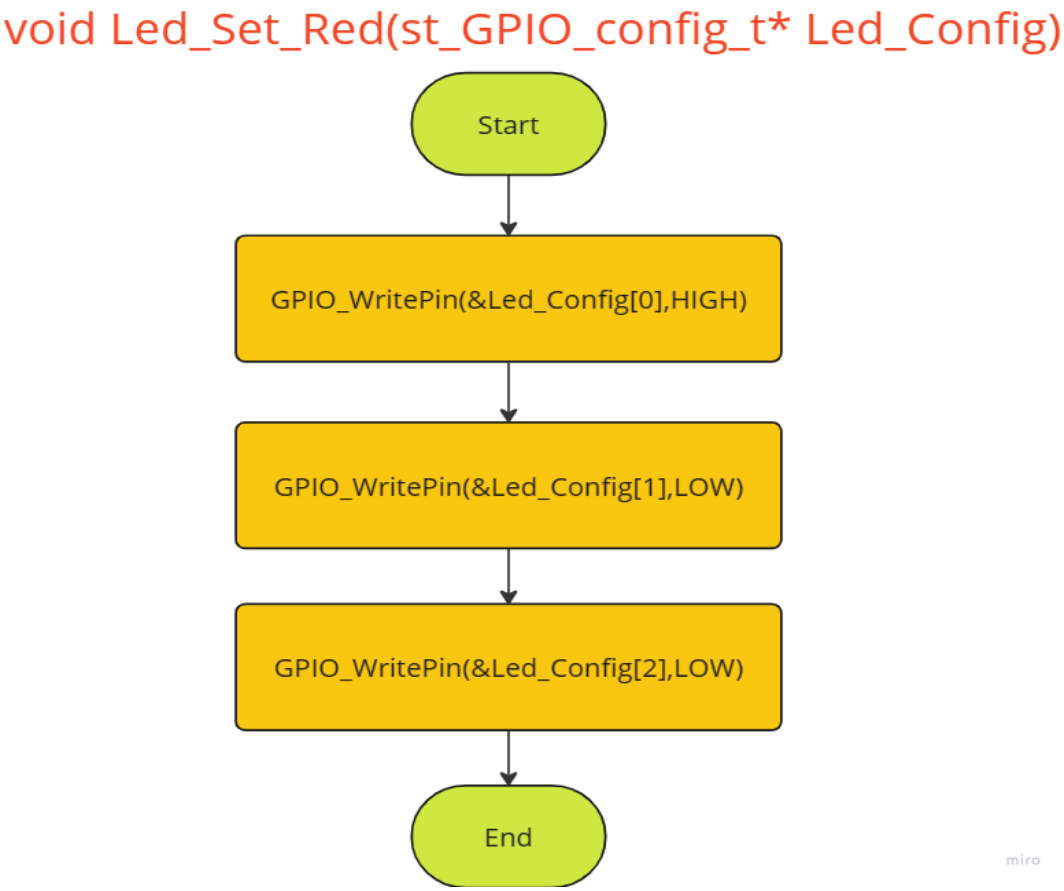


void BUTTON_IsPressed(st_GPIO_config_t *usr_buttonConfig, en_button_state_t *buttonState)

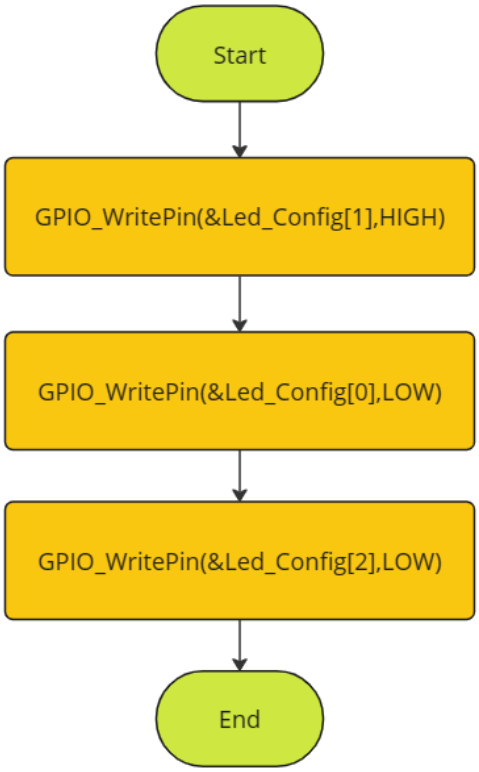


3.1.3 LED Module

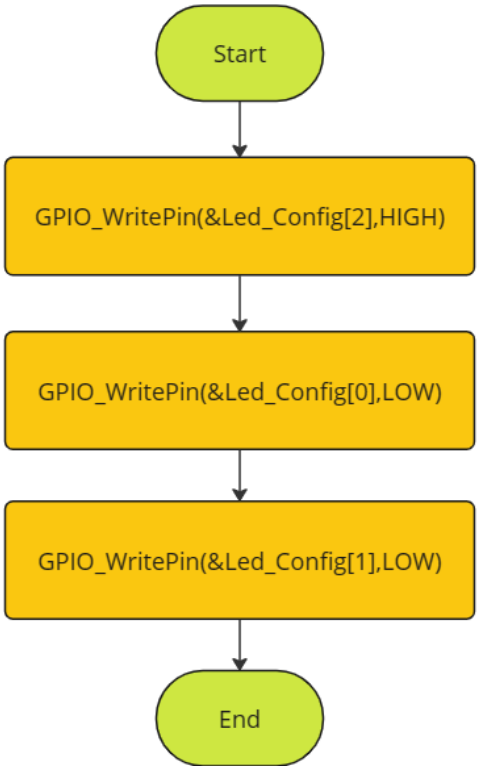




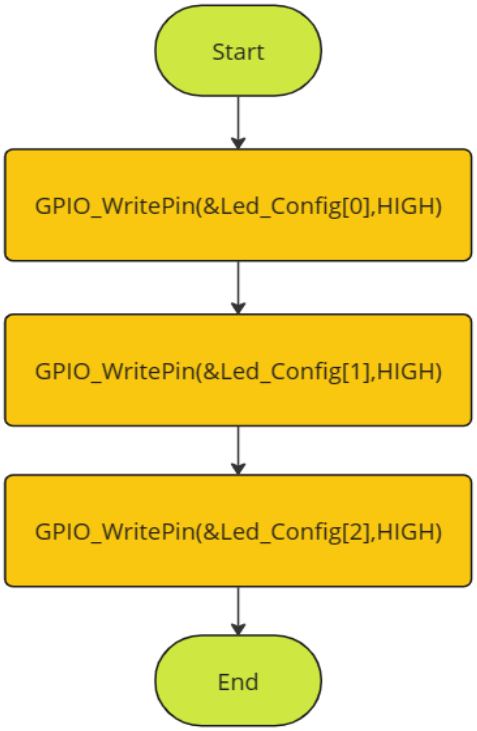
void Led_Set_Blue(st_GPIO_config_t* Led_Config)



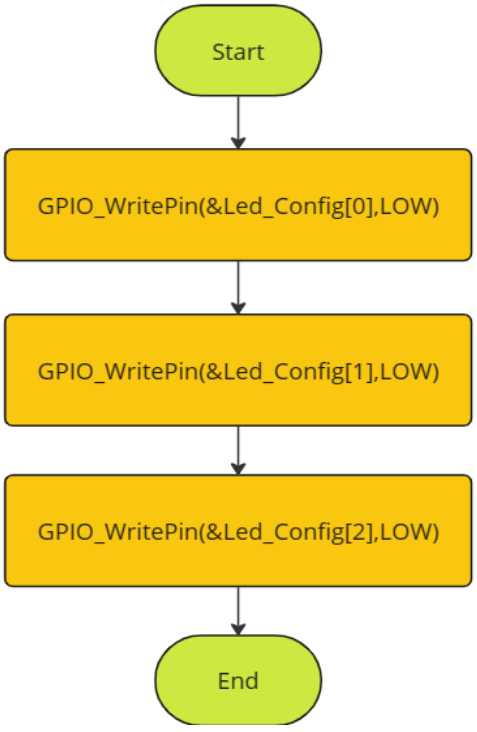
void Led_Set_Green(st_GPIO_config_t* Led_Config)



void Leds_on(st_GPIO_config_t* Led_Config)

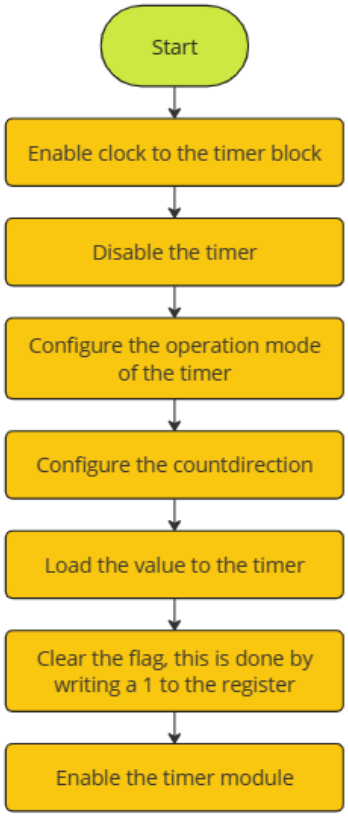


void Leds_off(st_GPIO_config_t* Led_Config)

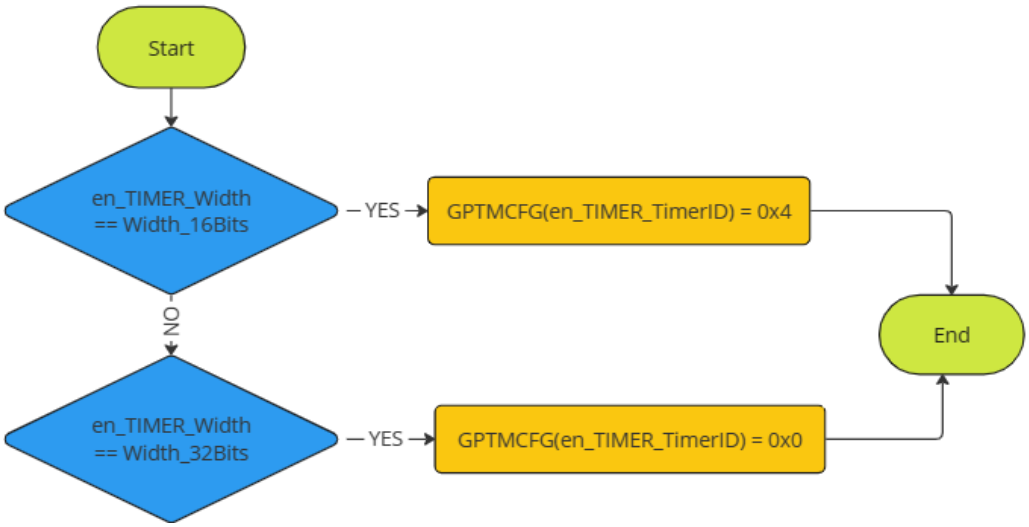


3.1.4 Timer Module

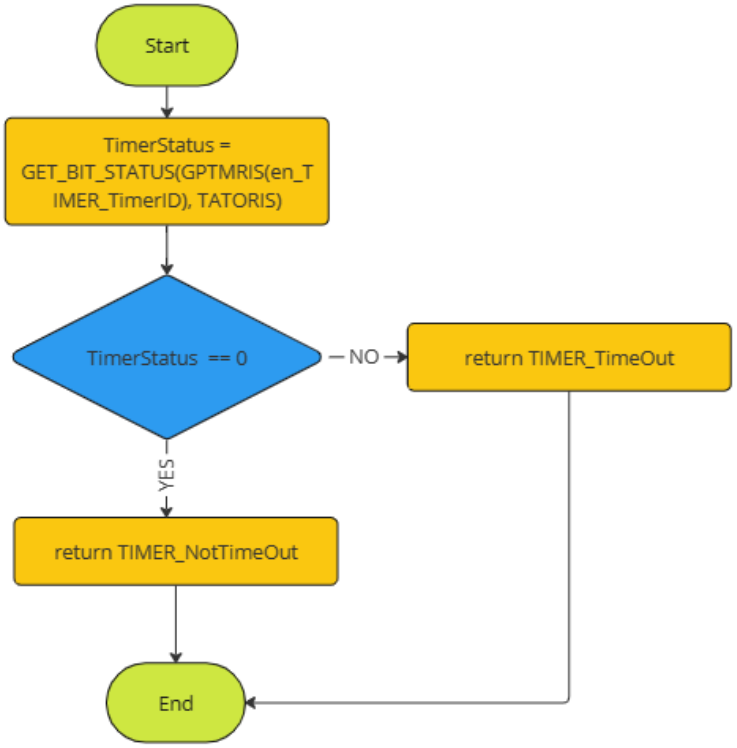
void TIMER_Init(st_TIMER_config_t *ptr_st_TIMER_config)



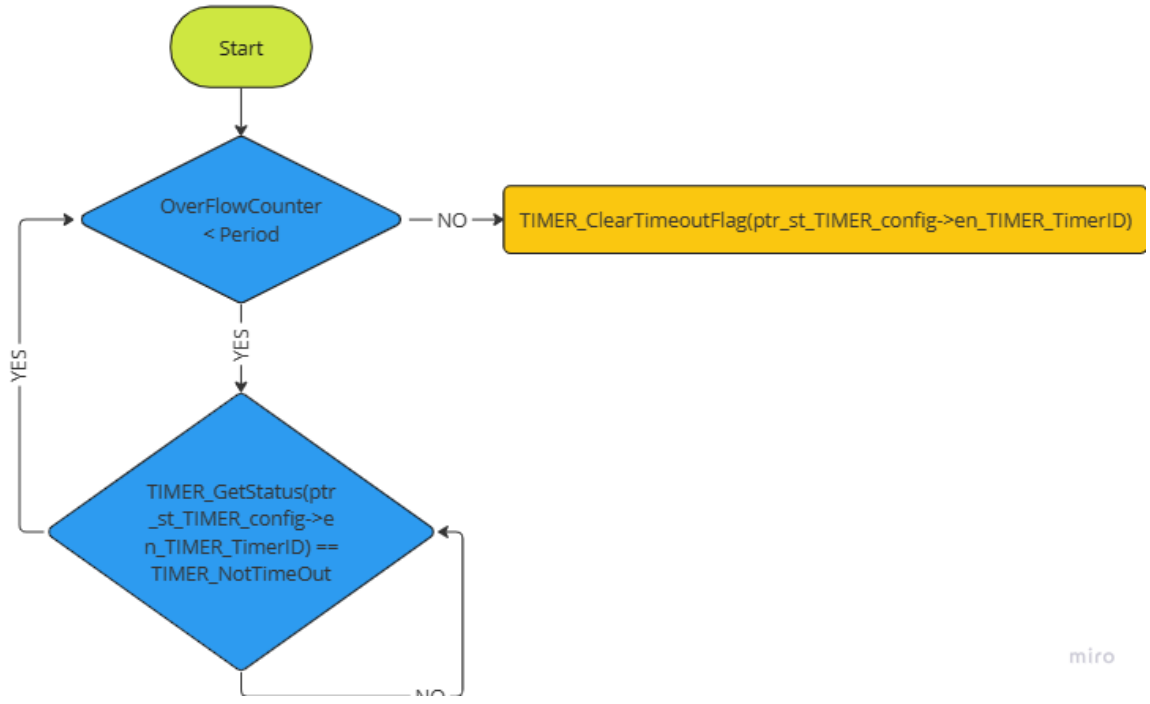
static void TIMER_WidthSelect(en_TIMER_TimerID_t en_TIMER_TimerID , en_TIMER_Width_t en_TIMER_Width)



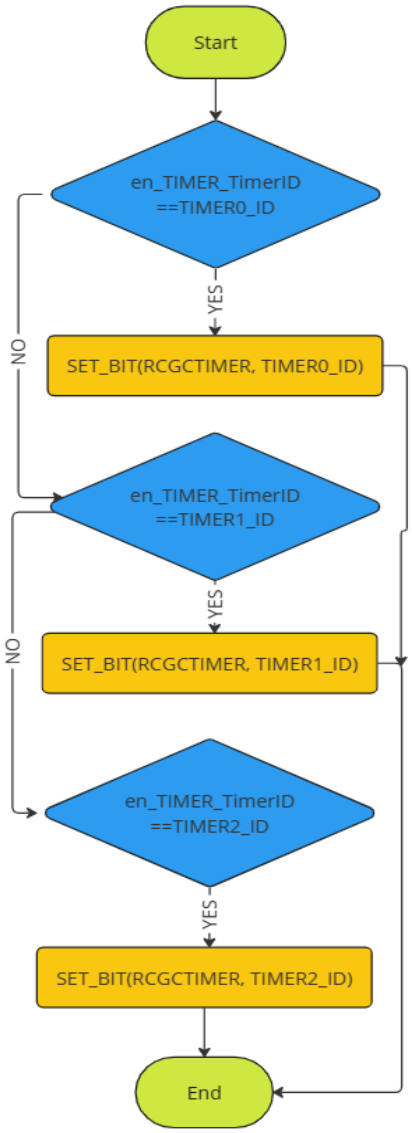
en_TIMER_Status_t TIMER_GetStatus(en_TIMER_TimerID_t en_TIMER_TimerID)



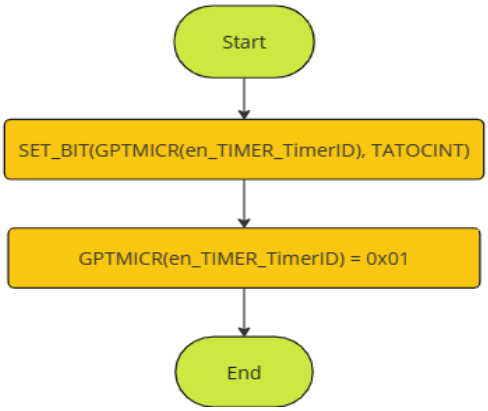
void TIMER_DelayMS(st_TIMER_config_t *ptr_st_TIMER_config, uint32 Period)



static void TIMER_EnableClock(en_TIMER_TimerID_t en_TIMER_TimerID)



void TIMER_ClearTimeoutFlag(en_TIMER_TimerID_t en_TIMER_TimerID)



3.2 Precompiling & Linking Configurations

3.2.1GPIO

```
    * @enum en_GPIO_pinDir_t
    * Specifies the mode of operation of the pin.
    */
typedef enum {
        INPUT = 0, OUTPUT = 1
}en_GPIO_pinDir_t;

/**
    * @enum en_GPIO_driveCurrent_t
    * Specifies the drive current of the pin.
    */
typedef enum {
        DRIVE_2mA = 0, DRIVE_4mA = 1, DRIVE_8mA = 2
}en_GPIO_driveCurrent_t;

/**
    * @enum en_GPIO_pull_t
    * Specifies the pull state of the pin.
    */
typedef enum {
        PULL_UP = 0, PULL_DOWN = 1, OPEN_DRAIN = 2
}en_GPIO_pull_t;

typedef enum {
        GPIO_STATUS_SUCCESS                             =    100,
        GPIO_STATUS_CLOCK_FAILED                        =    101,
        GPIO_STATUS_SET_DIRECTION_FAILED                =    102,
        GPIO_STATUS_SET_DRIVE_CURRENT_FAILED            =    103,
        GPIO_STATUS_SET_PULL_FAILED                     =    104,
        GPIO_STATUS_INVALID_PIN_DIR,
        GPIO_STATUS_INVALID_PORT_NUM,
        GPIO_STATUS_INVALID_CONFIG_ARRAY                =    105
}en_GPIO_error_t;
```

```
typedef struct {
    en_GPIO_port_t          en_GPIO_port;
    en_GPIO_pin_t           en_GPIO_pin;
    en_GPIO_pinDir_t        en_GPIO_pinDir;
    en_GPIO_driveCurrent_t  en_GPIO_driveCurrent;
    en_GPIO_pull_t          en_GPIO_pull;
}st_GPIO_config_t;

/**
 * @brief Initializes a set of pins with the specified configuration.
 * @param[in] ptr_st_GPIO_config    Address of the array of the specified pins.
 * @return en_GPIO_error_t
 */
en_GPIO_error_t GPIO_Init(const st_GPIO_config_t *ptr_st_GPIO_config);


en_GPIO_error_t GPIO_DeInit(const st_GPIO_config_t *ptr_st_GPIO_config);

/**
 * @brief Reads the state of a single pin.
 * @param[in] ptr_st_GPIO_config    Address of the specified pin configuration strcuture.
 * @param[in] ptr_pinValue          Address of the value where we stores the state of the pin.
 * @return en_GPIO_error_t
 */
en_GPIO_error_t GPIO_ReadPin(const st_GPIO_config_t *ptr_st_GPIO_config, uint8_t *ptr_pinValue);

/**
 * @brief Reads the state of a single pin.
 * @param[in] ptr_st_GPIO_config    Address of the specified pin configuration strcuture.
 * @param[in] pinValue              The value to be written on the specified pin.
 * @return en_GPIO_error_t
 */
en_GPIO_error_t GPIO_WritePin(const st_GPIO_config_t *ptr_st_GPIO_config, uint8 pinValue);
void GPIO_TogglePin(const st_GPIO_config_t *ptr_st_GPIO_config);
```

3.2.2Button

```c
#define DEBOUNCE_THRESHOLD 9000

typedef enum {
    BUTTON_NotPressed = 0, BUTTON_Pressed = 1
}en_button_state_t;

/**
 * @brief Initializes the button pin.
 * @param[in] Button_config     Address of the configuration array.
 */
void BUTTON_Init(st_GPIO_config_t *Button_config);

/**
 * @brief Initializes the button pin.
 * @param[in] usr_buttonConfig Address of the configuration array.
 * @param[in] value         Address of the variables to store the state of push button.
 */
void BUTTON_IsPressed( st_GPIO_config_t *usr_buttonConfig, en_button_state_t *value);
```

```c
#include "button_config.h"

/* User configuration array. */
st_GPIO_config_t usr_button_config [PUSH_BUTTON_NUM] = {
    {PORT_F , PIN4 , INPUT , DRIVE_4mA , PULL_UP}
};
```

3.2.3LED

```
 * @brief Initializes the LED pin using the configuration array.
 * @param[in] ptr_st_LED_config     Address of the configuration array.
 * @return en_GPIO_error_t
 */
void Led_Init(st_GPIO_config_t *ptr_st_LED_config);

/**
 * @brief Turns on the red LED and turns off all the other LEDs.
 * @param[in] ptr_st_LED_config     Address of the configuration array.
 * @return en_GPIO_error_t
 */
void LED_GreenOn(st_GPIO_config_t *ptr_st_LED_config);

/**
 * @brief Turns on the blue LED and turns off all the other LEDs.
 * @param[in] ptr_st_LED_config     Address of the configuration array.
 * @return en_GPIO_error_t
 */
void LED_BlueOn(st_GPIO_config_t *ptr_st_LED_config);

/**
 * @brief Turns on the green LED and turns off all the other LEDs.
 * @param[in] ptr_st_LED_config     Address of the configuration array.
 * @return en_GPIO_error_t
 */
void LED_RedOn(st_GPIO_config_t *ptr_st_LED_config);

/**
 * @brief Turns on all the LEDs.
 * @param[in] ptr_st_LED_config     Address of the configuration array.
 * @return en_GPIO_error_t
 */
void LED_AllOn(st_GPIO_config_t *ptr_st_LED_config);
```

```
#include "led_config.h"

/* User configuration array. */
st_GPIO_config_t usr_led_config [PORT_PINS_NUM] = {
    {PORT_F, PIN1, OUTPUT,DRIVE_2mA, PULL_DOWN},   /* Red Led */
    {PORT_F, PIN2, OUTPUT,DRIVE_4mA, PULL_DOWN},   /* Blue Led */
    {PORT_F, PIN3, OUTPUT,DRIVE_8mA, PULL_DOWN}   /* Geen Led */
};
```

3.2.4 Timer

```
typedef enum {
    Width_16Bits = 0, Width_32Bits = 1
}en_TIMER_Width_t;

typedef enum {
    OneShotMode = 1, PeriodicMode = 2, CaptureMode = 3
}en_TIMER_OperationMode_t;

typedef enum {
    CountDown = 0, CountUp = 1
}en_TIMER_CountDirection_t;

typedef enum {
    TIMER0_ID = 0, TIMER1_ID = 1, TIMER2_ID = 2, TIMER3_ID = 3, TIMER4_ID = 4, TIMER5_ID = 5
}en_TIMER_TimerID_t;

typedef enum {
    TIMER_NotTimeOut = 0, TIMER_TimeOut = 1
}en_TIMER_Status_t;

typedef struct {
    en_TIMER_TimerID_t          en_TIMER_TimerID;
    en_TIMER_Width_t            en_TIMER_Width;
    en_TIMER_OperationMode_t    en_TIMER_OperationMode;
    uint32                      Period;
    en_TIMER_CountDirection_t   en_TIMER_CountDirection;
}st_TIMER_config_t;
```

```
#define PERIOD_1_MS (16000 - 1)

st_TIMER_config_t myTimerConfig = {
    .en_TIMER_TimerID           = TIMER0_ID,
    .en_TIMER_Width             = Width_16Bits,
    .en_TIMER_OperationMode     = PeriodicMode,
    .Period                     = PERIOD_1_MS,
    .en_TIMER_CountDirection    = CountUp,
};
```