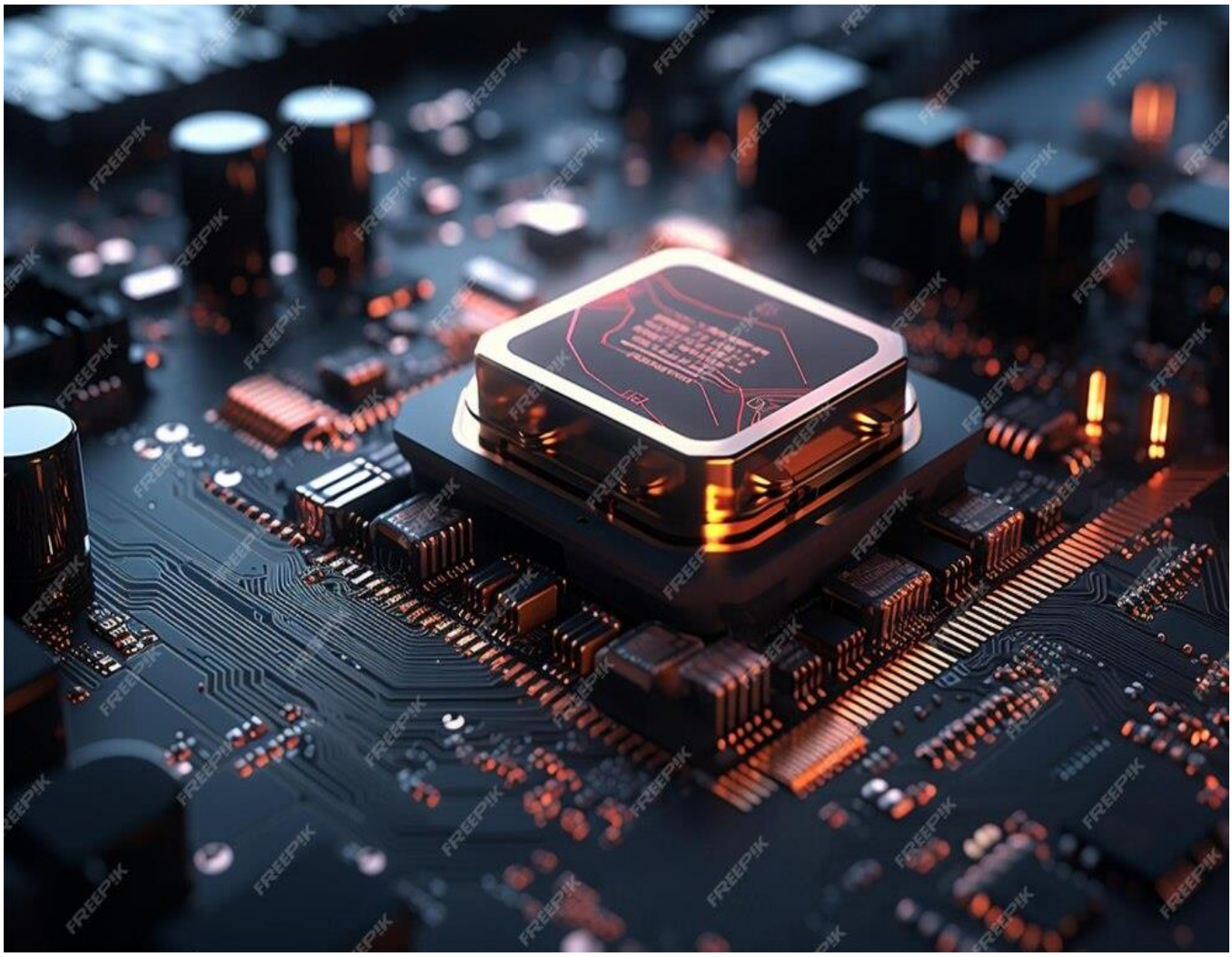


RGB LED BRIGHTNESS CONTROL V1.0 DESIGN

Peter Essam | ARM | 26/11/2023

Table of Content

Introduction

- Overview:

  This project is designed with a layered architecture, separating concerns into different layers for better maintainability and scalability. The project focuses on controlling an RGB LED using GPIO (General Purpose Input/Output) pins. The Microcontroller Abstraction Layer (MCAL) handles low-level hardware interactions, the Hardware Abstraction Layer (HAL) manages LED and button functionality, the Service Layer manage the drivers in MCAL to can included in application and the Common Layer provides standard library names for consistency.

- Layers:

  MCAL (Microcontroller Abstraction Layer)

  Responsible for low-level hardware interactions.

  Utilizes GPIO to control hardware-level features

  And GPT to calculate time and generate pwm.

  Abstracts microcontroller-specific details.

  HAL (Hardware Abstraction Layer)

  Manages higher-level functionalities for LEDs and buttons.

  Uses MCAL services to control GPIO pins.

Provides an abstraction for RGB LED control and button input.

Service Layer

Manage the drivers in MCAL layer to can included in App layer

Common Layer

Hosts standard library names and common services.

App (Application Layer)

This is the Application

- ## Project Functionality:

The main objective of this project is to control an RGB LED based on button presses and time calculated. The RGB LED is connected to specific GPIO pins on the microcontroller. When a button is pressed, the program detects the button press through the HAL layer, and the RGB LED changes its state accordingly and when the time finish the LED is off .

- ## Key Components :

MCAL Layer

GPIO driver: Provides low-level functions for GPIO pin initialization, reading, and writing.

HAL Layer

LED Interface: functions to control the RGB LED (e.g., turning on, turning off , ...).
Button Interface: Handles button-related operations (e.g., detecting button presses).

Service Layer

Facilitation control any driver in MCAL want to included in App layer.

Common Layer

Standard Library Names: Ensures consistent naming conventions and library usage across the project.

- ## Workflow:

Initialization:

MCAL initializes GPIO pins for the RGB LED
And initialize GPT(Timer) to generate time and pwm.
HAL initializes LED and button components.

Button Press Detection:

HAL layer monitors the button state and detects button presses.

RGB LED Control:

Based on button presses, the HAL layer controls the RGB LED through the MCAL GPIO driver.
Possible actions: turn on, turn off, change color.

- ## Benefits:

Modularity

Each layer is modular, making it easier to modify or extend functionalities.

Abstraction

Higher layers abstract hardware details, promoting code readability.

Consistency

Standard library names in the common layer ensure consistent coding practices.

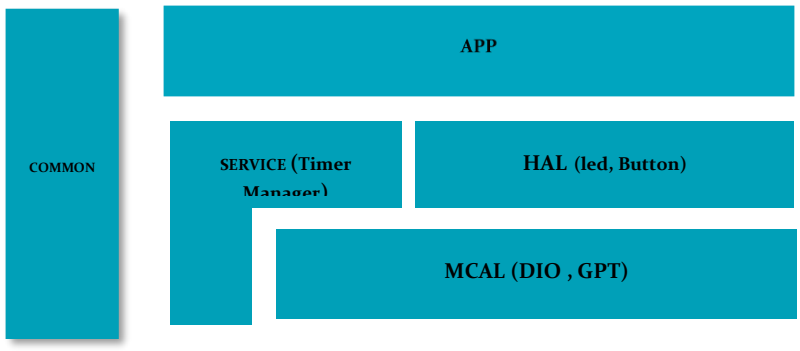
- ## Conclusion:

This project showcases a well-organized architecture with separate layers, each serving a specific purpose. The use of MCAL, HAL, Service Layer and a Common Layer contributes to

code clarity, maintainability, and scalability, making it easier to manage and expand the functionality of the RGB LED control system.

High Level Design

- Layered Architecture



- Modules Descriptions

MCAL Layer

GPIO driver : Provides low-level functions for GPIO pin initialization, reading, and writing to control RGB Leds signals.

HAL Layer

LED driver: control the RGB LED (e.g., led initialization , led on, led off , led toggle)
Button driver: Handles button-related operations (e.g., button initialization, detecting button presses).\

Service Layer

Control MCAL drivers in App layer

Common Layer

Standard Library Names: to serve all project layers

App Layer

This is the application i want to do

- Drivers Documentations
  - GPIO

    (This Driver Located in MCAL Layer)

1-

enu_MGPIO_errorStatus_t MGPIO_init(str_MGPIO_configuration_t *ptr_str_MGPIO_config)

Description:

Initializes a GPIO pin based on the provided configuration.

Arguments:

ptr_str_MGPIO_config Pointer to a structure contain GPIO config

Return:

GPIO_OK: Successful initialization.
GPIO_NULL_POINTER: Null pointer argument.
GPIO_PORT_ERROR: Invalid port number.
GPIO_PIN_ERROR: Invalid pin number.
GPIO_DIRECTION_ERROR: Invalid pin direction.
GPIO_MODE_ERROR: Invalid mode selection.
GPIO_OUT_CURRENT_ERROR: Invalid output current.
GPIO_INTERNAL_TYPE_ERROR: Invalid internal type.
GPIO_VALUE_ERROR: Invalid output level.

2-

enu_MGPIO_errorStatus_t MGPIO_write(enu_MGPIO_portNumber_t enu_a_portNumber, enu_MGPIO_pinNumber_t enu_a_pinNumber, enu_MGPIO_pinValue_t enu_l_pinValue)

Description:

Write a value to a specific GPIO pin.

Arguments:

enu_a_portNumber Select the GPIO port number.

enu_a_pinNumber Select the GPIO pin number.

enu_l_pinValue Select the value to be written to the pin (PIN_HIGH_VALUE or PIN_LOW_VALUE).

Return:

GPIO_OK	Success operation.
GPIO_PORT_ERROR	Invalid port number.
GPIO_PIN_ERROR	Invalid pin number.
GPIO_VALUE_ERROR	Invalid pin value.
GPIO_PORT_NOT_INITIALIZED	Port not initialized.

3-

enu_MGPIO_errorStatus_t MGPIO_read(enu_MGPIO_portNumber_t enu_a_portNumber, enu_MGPIO_pinNumber_t enu_a_pinNumber, boolean *ptr_arg_pinValue)

Description:

Read the value of a specific GPIO pin.

Arguments:

enu_a_portNumber	Select the GPIO port number.
enu_a_pinNumber	Select the GPIO pin number.
ptr_arg_pinValue	Pointer to a boolean variable to store the read value.

Return

GPIO_OK	Success operation.
GPIO_PORT_ERROR	Invalid port number.
GPIO_PIN_ERROR	Invalid pin number.
GPIO_NULL_POINTER	Null pointer argument.
GPIO_PORT_NOT_INITIALIZED	Port not initialized.

4-

enu_MGPIO_errorStatus_t MGPIO_read(enu_MGPIO_portNumber_t enu_a_portNumber, enu_MGPIO_pinNumber_t enu_a_pinNumber, boolean *ptr_arg_pinValue)

Description:

Toggle the value of a specific GPIO pin.

Arguments:

enu_a_portNumber	Select the GPIO port number.
enu_a_pinNumber	Select the GPIO pin number.

Return

GPIO_OK	Success operation.
GPIO_PORT_ERROR	Invalid port number.
GPIO_PIN_ERROR	Invalid pin number.
GPIO_PORT_NOT_INITIALIZED	Port not initialized

➢ GPT (Timer)

(This Driver Located in MCAL Layer)

1-

enu_MTIMER_status_t M_timerInit(str_MTIMER_configurations_t *ptr_timerConfigurations)

Description:

Initializes the specified timer based on the provided configurations.

Arguments:

ptr_timerConfigurations Pointer to a structure containing timer configurations.

Return:

enu_MTIMER_status_t Error status after timer initialization.

2-

enu_MTIMER_status_t M_timerStart(enu_MTIMER_timerSelect_t enu_arg_timerSelect , uint32_t u32_arg_desiredTime , enu_timeUnit_t enu_arg_timeUnit)

Description:

Starts the specified timer with the desired time and time unit

Arguments:

enu_arg_timerSelect The selected timer to start.

u32_arg_desiredTime The desired time to run the timer.

enu_arg_timeUnit The time unit of the desired time (microseconds, milliseconds, seconds).

Return:

enu_MTIMER_status_t Error status after starting the timer.

3-

enu_MTIMER_status_t M_timerEnableInterrupt(enu_MTIMER_timerSelect_t enu_arg_timerSelect)

Description:

Enables interrupts for the specified timer.

Arguments:
enu_arg_timerSelect The selected timer to enable interrupts
Return:
enu_MTIMER_status_t Error status after enabling interrupts.

4-
enu_MTIMER_status_t M_timerDisableInterrupt(enu_MTIMER_timerSelect_t enu_arg_timerSelect)
Description:
Disable interrupts for the specified timer.
Arguments:
enu_arg_timerSelect The selected timer to disable interrupts
Return:
enu_MTIMER_status_t Error status after disabling interrupts.

5-
enu_MTIMER_status_t M_Timer_getElapsedTime(enu_MTIMER_timerSelect_t enu_arg_timerSelect , uint32_t *u32_ptr_timeMs)
Description:
Retrieves the elapsed time from the specified timer.
Arguments:
enu_arg_timerSelect The selected timer to retrieve elapsed time.
u32_ptr_timeMs Pointer to store the elapsed time in milliseconds.
Return:
enu_MTIMER_status_t Error status after retrieving elapsed time.

6-
enu_MTIMER_status_t M_TIMER_getRemainingTime (enu_MTIMER_timerSelect_t enu_arg_timerSelect , uint32_t *u32_ptr_timeMs)
Description:
Retrieves the remaining time from the specified timer.
Arguments:
enu_arg_timerSelect The selected timer to retrieve remaining time.
u32_ptr_timeMs Pointer to store the remaining time in milliseconds.
Return:
enu_MTIMER_status_t Error status after retrieving remaining time.

7-
enu_MTIMER_status_t M_stopTimer(enu_MTIMER_timerSelect_t enu_arg_timerSelect)
Description:
Stop the specified timer.

Arguments:
enu_arg_timerSelect The selected timer stop time.
Return:
enu_MTIMER_status_t Error status stop time.

8-
enu_MTIMER_status_t M_TIMER_set_pwm(enu_MTIMER_timerSelect_t enu_arg_timerSelect , uint16_t u16_arg_durationMs , uint8_t u8_arg_dutyCycle)
Description:
Configure timer for set PWM operation.
Arguments:
enu_arg_timerSelect The selected timer to configure for PWM.
u16_arg_durationMs The total duration of the PWM signal in milliseconds.
u8_arg_dutyCycle The duty cycle of the PWM signal as a percentage.
Return:
enu_MTIMER_status_t Error status after configuring the timer for PWM.

- LED

(This Driver Located in HAL Layer)

1-
enu_ledErrorState_t H_LED_init(enu_MGPIO_portNumber_t enu_l_ledPort, enu_MGPIO_pinNumber_t enu_l_ledPin)

Description:
Initialize a LED on a specific GPIO port and pin
Arguments:
enu_l_ledPort Select the GPIO port number for the LED.
enu_l_ledPin Select the GPIO pin number for the LED.
Return
LED_OK Success initialization.
LED_NOT_OK LED initialization not successful.

2-

enu_ledErrorState_t H_LED_on(enu_MGPIO_portNumber_t enu_l_ledPort, enu_MGPIO_pinNumber_t enu_l_ledPin)

Description:
Turn on a LED connected to a specific GPIO port and pin.
Arguments:
enu_l_ledPort Select the GPIO port number for the LED.
enu_l_ledPin Select the GPIO pin number for the LED.
Return
LED_OK Success initialization.
LED_NOT_OK LED initialization not successful.

3-

enu_ledErrorState_t H_LED_off(enu_MGPIO_portNumber_t enu_l_ledPort, enu_MGPIO_pinNumber_t enu_l_ledPin)

Description:
Turn off a LED connected to a specific GPIO port and pin.
Arguments:
enu_l_ledPort Select the GPIO port number for the LED.
enu_l_ledPin Select the GPIO pin number for the LED.
Return
LED_OK Success initialization.
LED_NOT_OK LED initialization not successful.

4-

enu_ledErrorState_t H_LED_toggle(enu_MGPIO_portNumber_t enu_l_ledPort, enu_MGPIO_pinNumber_t enu_l_ledPin)

Description:
Toggle the state of an LED connected to a specific GPIO port and pin.
Arguments:
enu_l_ledPort Select the GPIO port number for the LED.
enu_l_ledPin Select the GPIO pin number for the LED.
Return
LED_OK Success initialization.
LED_NOT_OK LED initialization not successful.

- BUTTON

(This Driver Located in HAL Layer)

1-

enu_buttonErrorStatus_t H_BUTTON_init(void)

Description:

Initialize the configuration of all buttons.

Return:

BUTTON_OK	Success initializing all buttons.
BUTTON_NOT_OK	Failed to initialize buttons.

2-

enu_buttonErrorStatus_t H_BUTTON_read(enu_buttonNumber_t enu_a_button_Number, boolean *ptr_a_value)

Description:

Read the state of a specific button.

Arguments:

enu_a_button_Number	The button number to read.
ptr_a_value	Pointer to a boolean variable to store the button state.

Return:

BUTTON_OK	Success initializing all buttons.
BUTTON_NOT_OK	Failed to initialize buttons.

- TIMER_MANAGER

(This Driver Located in Service Layer)

1-

enu_timerHandlerErrorStatus_t timerHandlerInit(str_MTIMER_configurations_t *str_TIMER_configs)

Description:

Initializes a timer handler based on the provided timer configurations..

Arguments:

str_TIMER_configs Pointer to a structure containing timer configurations.

Return

enu_timerHandlerErrorStatus_t Error status after initializing the timer handler.

2-

enu_timerHandlerErrorStatus_t timerHandlerStartTimer(enu_MTIMER_timerSelect_t enu_arg_timerSelect ,uint32_t u32_arg_time , enu_timeUnit_t enu_arg_timeUnit)

Description:

Start a timer handler based on the provided timer configurations..

Arguments:

enu_arg_timerSelect The selected timer to start.
u32_arg_time The duration of the timer.
enu_arg_timeUnit The time unit for the duration (e.g., microseconds, milliseconds).

Return

enu_timerHandlerErrorStatus_t Error status after start the timer handler.

3-

enu_timerHandlerErrorStatus_t timerHandlerEnableInterrupt(enu_MTIMER_timerSelect_t enu_arg_timerSelect)

Description:

Enable a timer interrupt handler based on the provided timer configurations..

Arguments:

enu_arg_timerSelect The selected timer to start.

Return

enu_timerHandlerErrorStatus_t Error status after disable the timer handler.

4-

enu_timerHandlerErrorStatus_t timerHandlerDisableInterrupt(enu_MTIMER_timerSelect_t enu_arg_timerSelect)

Description:

Disable a timer interrupt handler based on the provided timer configurations..

Arguments:

enu_arg_timerSelect The selected timer to start.

Return

enu_timerHandlerErrorStatus_t Error status after disable the timer handler.

5-
enu_timerHandlerErrorStatus_t timerHandlergetElapsedTime (enu_MTIMER_timerSelect_t enu_arg_timerSelect , uint32_t *u32_ptr_time)

Description:

Retrieves the elapsed time from the specified timer in the timer handler

Arguments:

Retrieves the elapsed time from the specified timer in the timer handler

Return

enu_timerHandlerErrorStatus_t Error status after retrieving the elapsed time from the timer handler.

6-
enu_timerHandlerErrorStatus_t timerHandlerRemainingTime (enu_MTIMER_timerSelect_t enu_arg_timerSelect , uint32_t *u32_ptr_time)

Description:

Retrieves the remaining time from the specified timer in the timer handler

Arguments:

Retrieves the remaining time from the specified timer in the timer handler

Return

enu_timerHandlerErrorStatus_t Error status after retrieving the remaining time from the timer handler.

7-
enu_timerHandlerErrorStatus_t timerHandlerStopTimer(enu_MTIMER_timerSelect_t enu_arg_timerSelect)

Description:

Stop timer in timer handler

Arguments:

enu_arg_timerSelect The selected timer to stop.

Return

enu_timerHandlerErrorStatus_t Error status after stop the time from the timer handler.

8-
enu_timerHandlerErrorStatus_t timerHandlerSetPwm(enu_MTIMER_timerSelect_t enu_arg_timerSelect , uint16_t u16_arg_durationMs ,uint8_t u8_arg_dutyCycle)

Description:

Sets the PWM configuration for the specified timer in the timer handler

Arguments:

enu_arg_timerSelect The selected timer to set PWM configuration.
u16_arg_durationMs The duration of the PWM signal in milliseconds.
u8_arg_dutyCycle The duty cycle of the PWM signal as a percentage

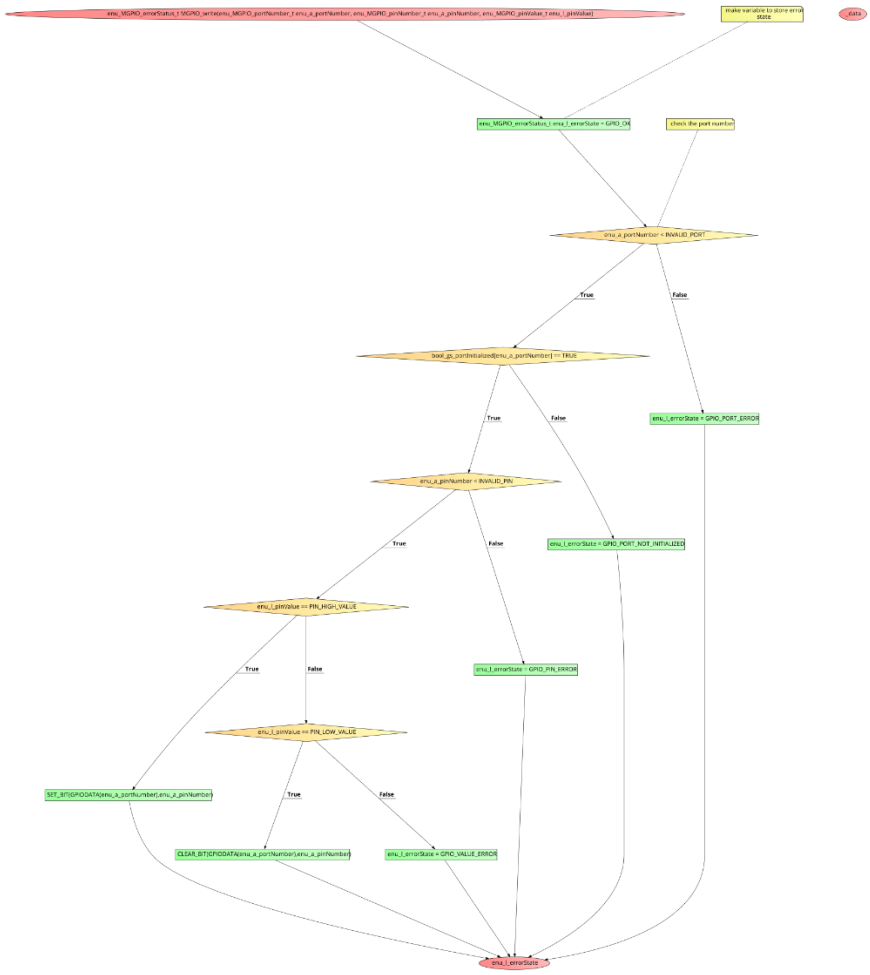
Return

enu_timerHandlerErrorStatus_t Error status after setting PWM configuration in the timer handler.

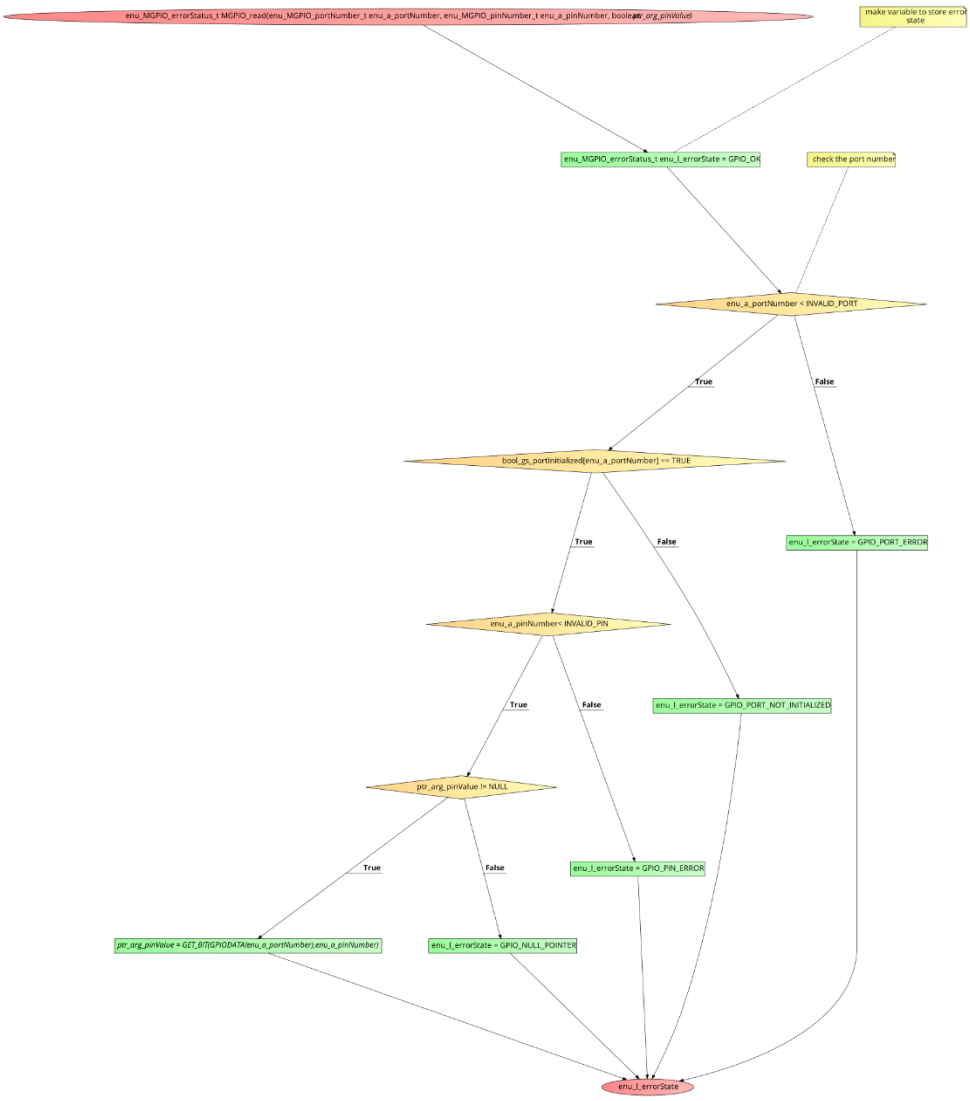
Low Level Design

- Flow Chart
  - GPIO

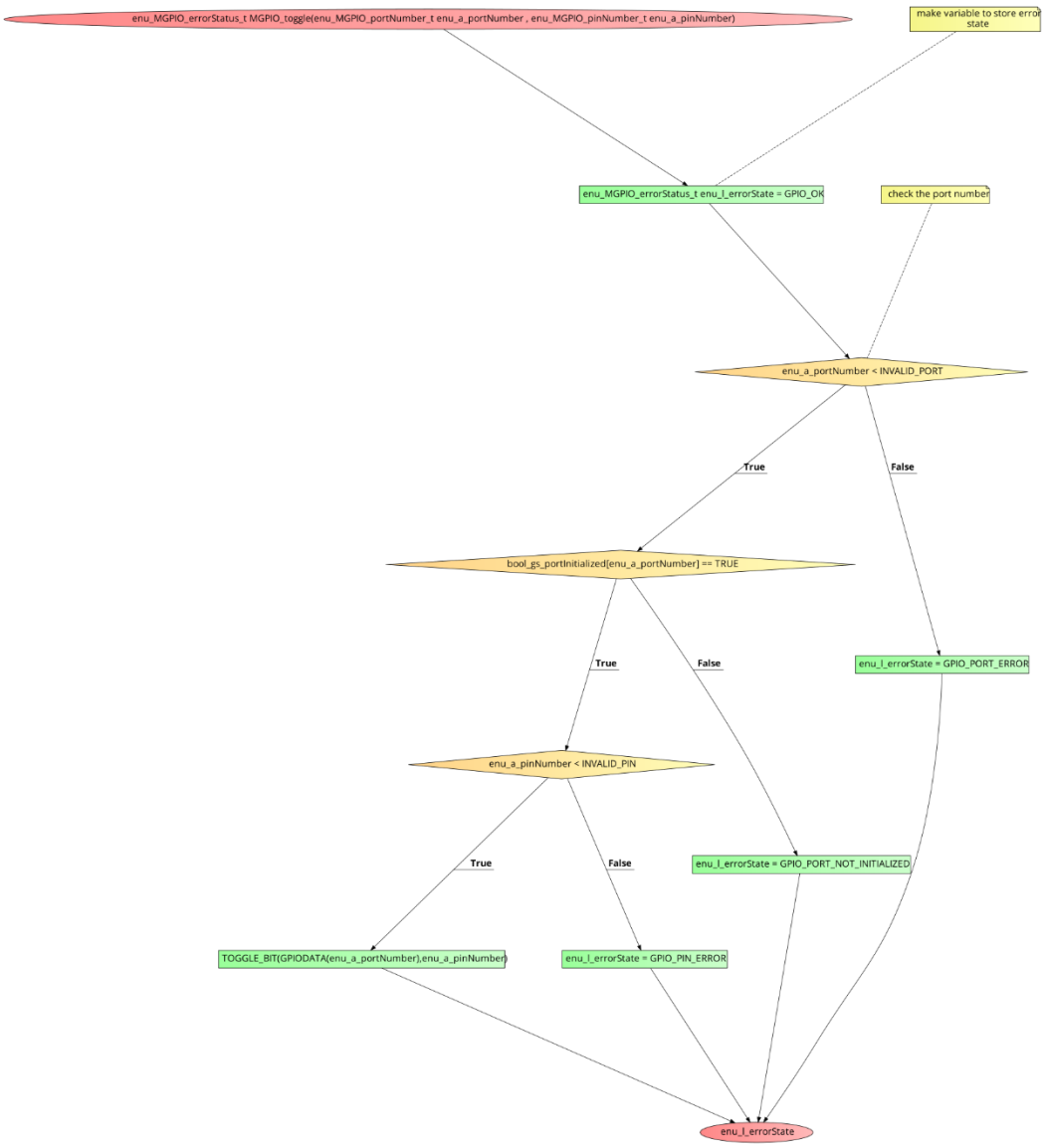
enu_MGPIO_errorStatus_t MGPIO_write(enu_MGPIO_portNumber_t enu_a_portNumber, enu_MGPIO_pinNumber_t enu_a_pinNumber, enu_MGPIO_pinValue_t enu_l_pinValue)



enu_MGPIO_errorStatus_t MGPIO_read(enu_MGPIO_portNumber_t enu_a_portNumber, enu_MGPIO_pinNumber_t enu_a_pinNumber, boolean *ptr_arg_pinValue)

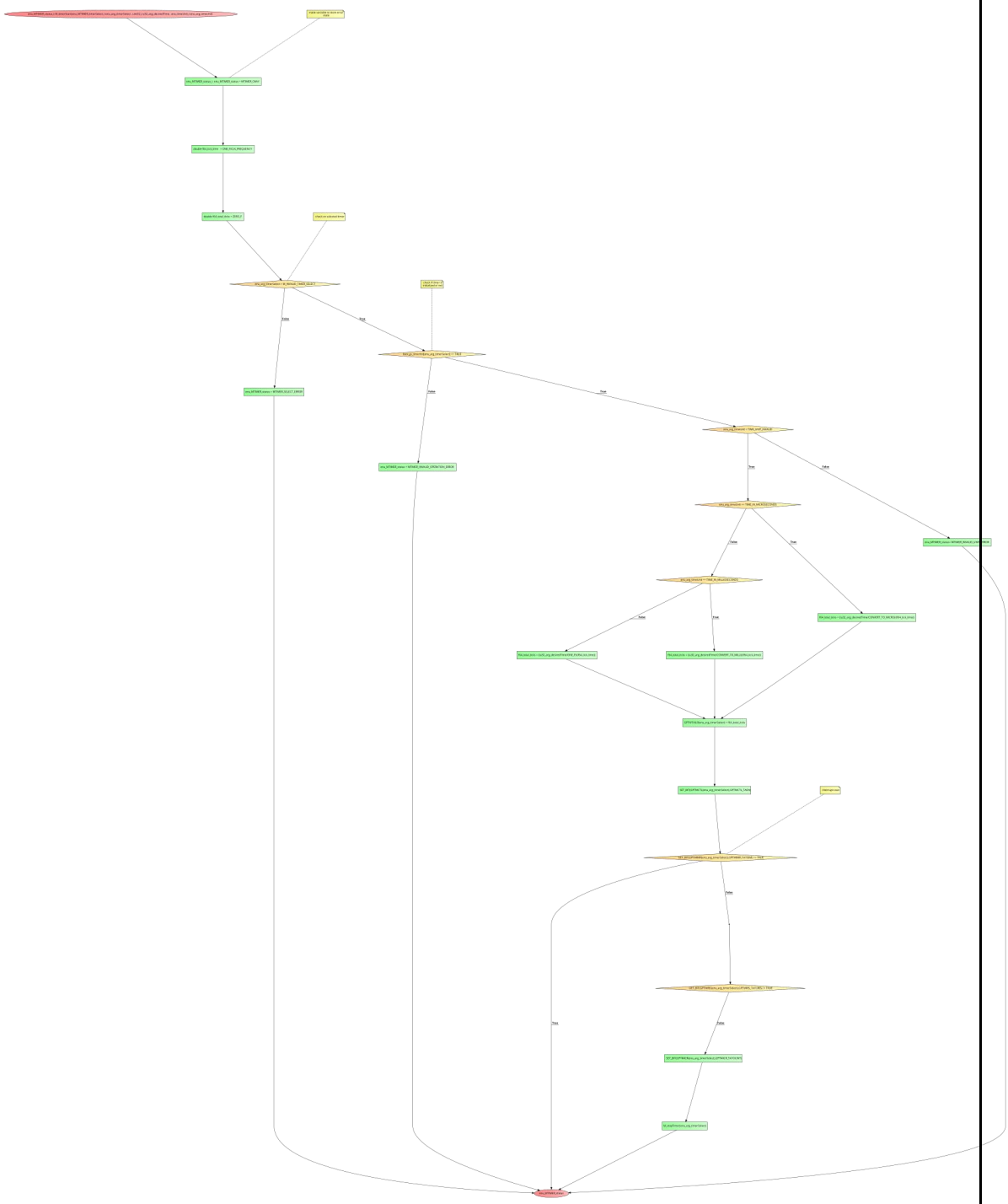


enu_MGPIO_errorStatus_t MGPIO_toggle(enu_MGPIO_portNumber_t enu_a_portNumber , enu_MGPIO_pinNumber_t enu_a_pinNumber)



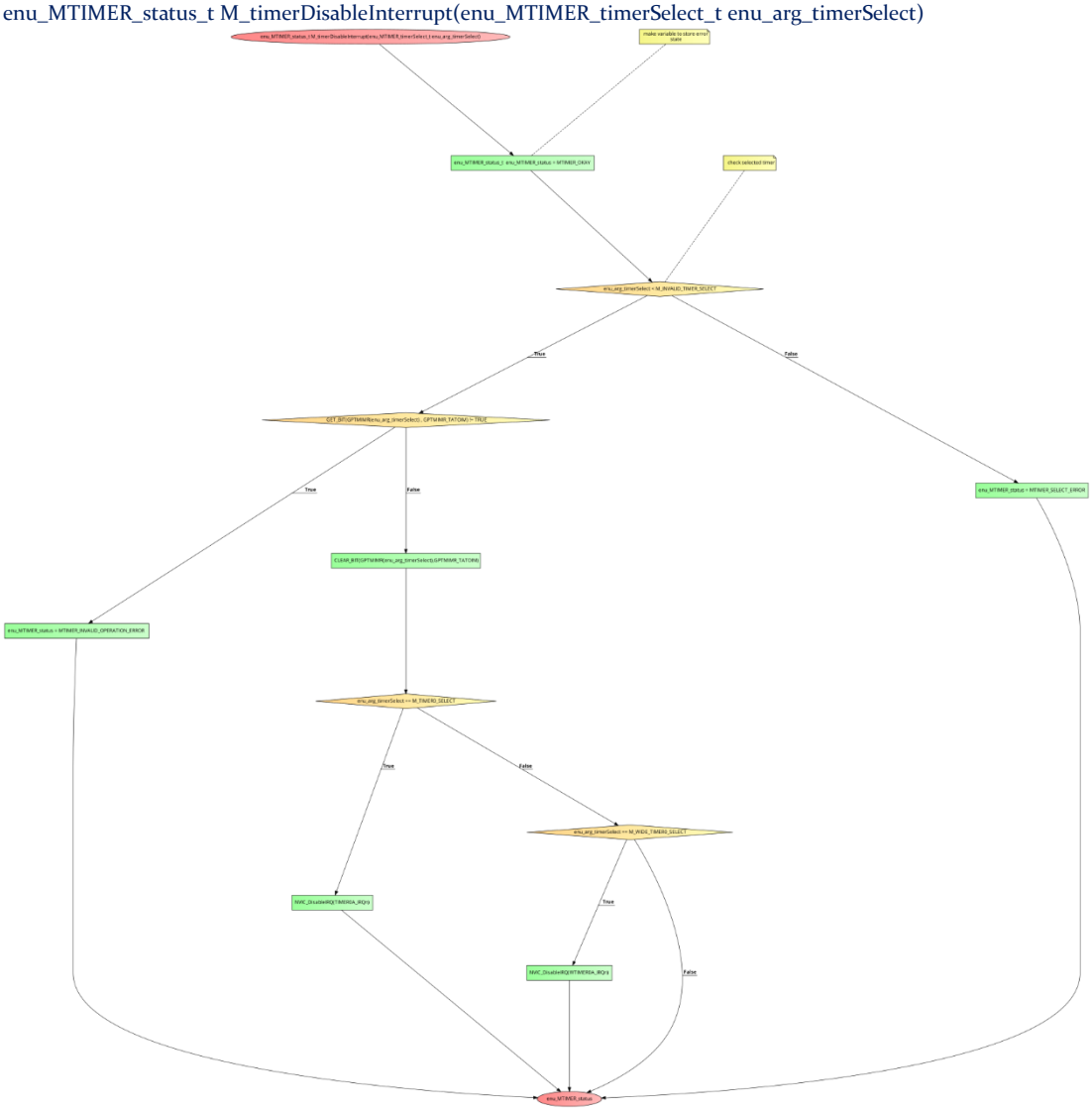
➢ GPT(Timer)

enu_MTIMER_status_t M_timerStart(enu_MTIMER_timerSelect_t enu_arg_timerSelect , uint32_t u32_arg_desiredTime , enu_timeUnit_t enu_arg_timeUnit)

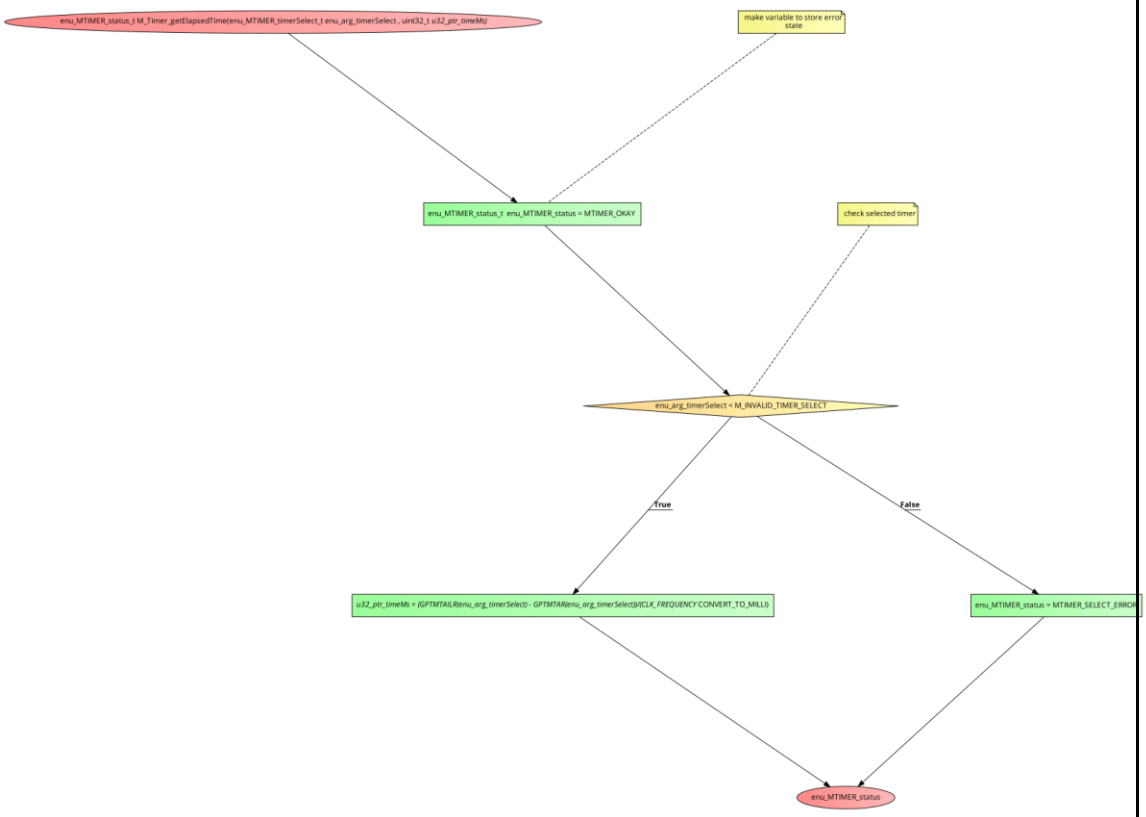


enu_MTIMER_status_t M_timerEnableInterrupt(enu_MTIMER_timerSelect_t enu_arg_timerSelec

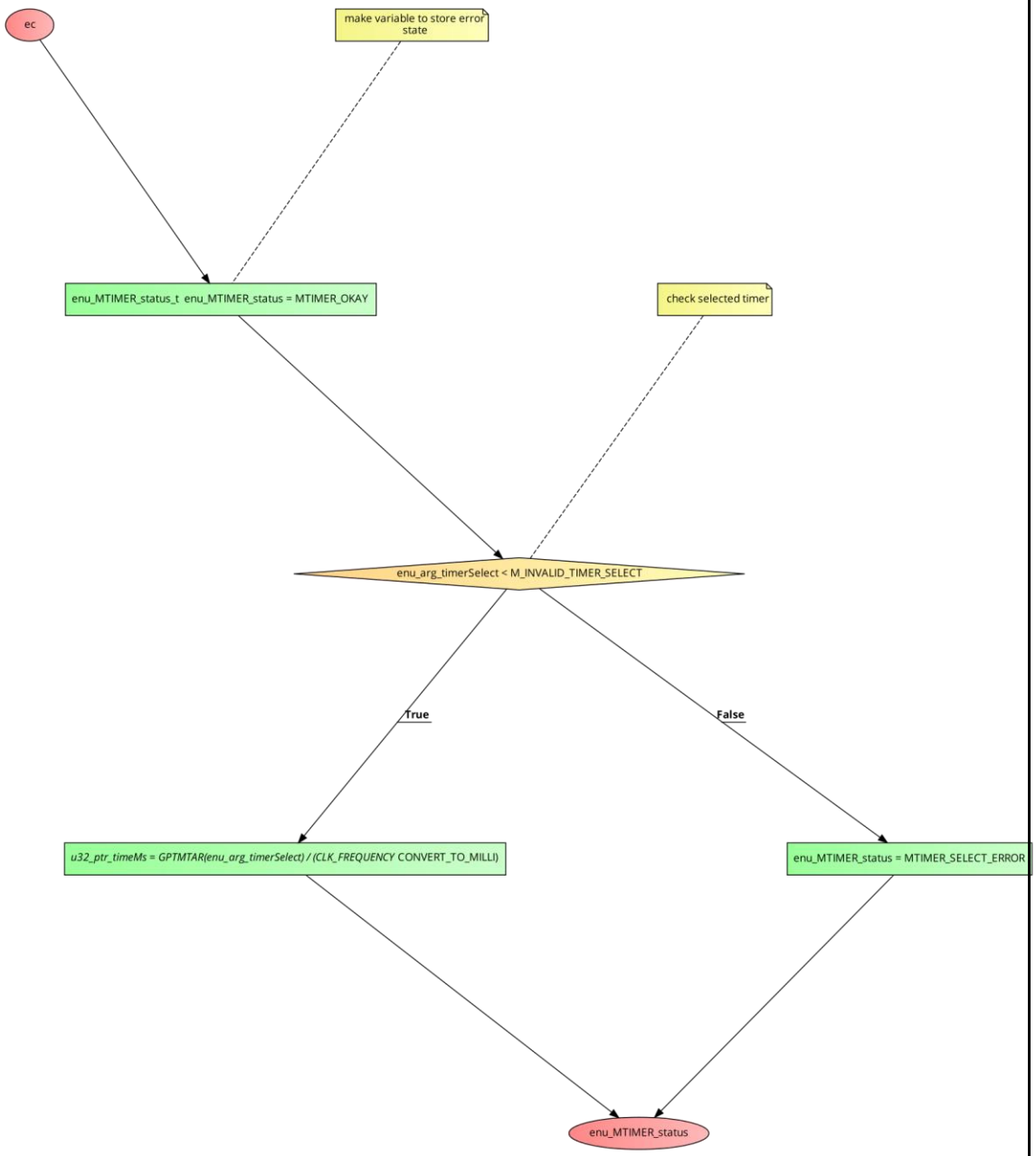
enu_MTIMER_status_t M_timerDisableInterrupt(enu_MTIMER_timerSelect_t enu_arg_timerSelect)



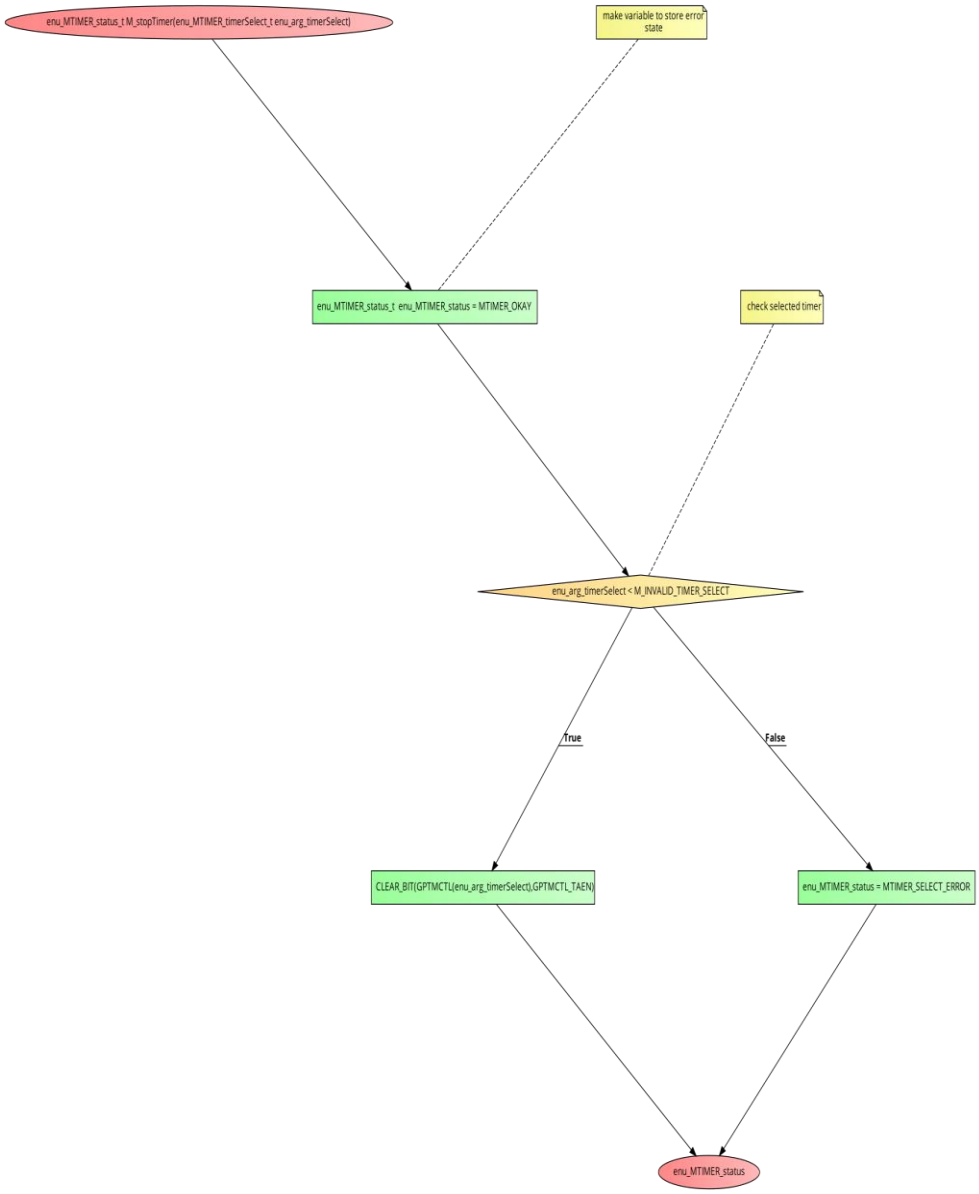
enu_MTIMER_status_t M_Timer_getElapsedTime(enu_MTIMER_timerSelect_t enu_arg_timerSelect , uint32_t *u32_ptr_timeMs)



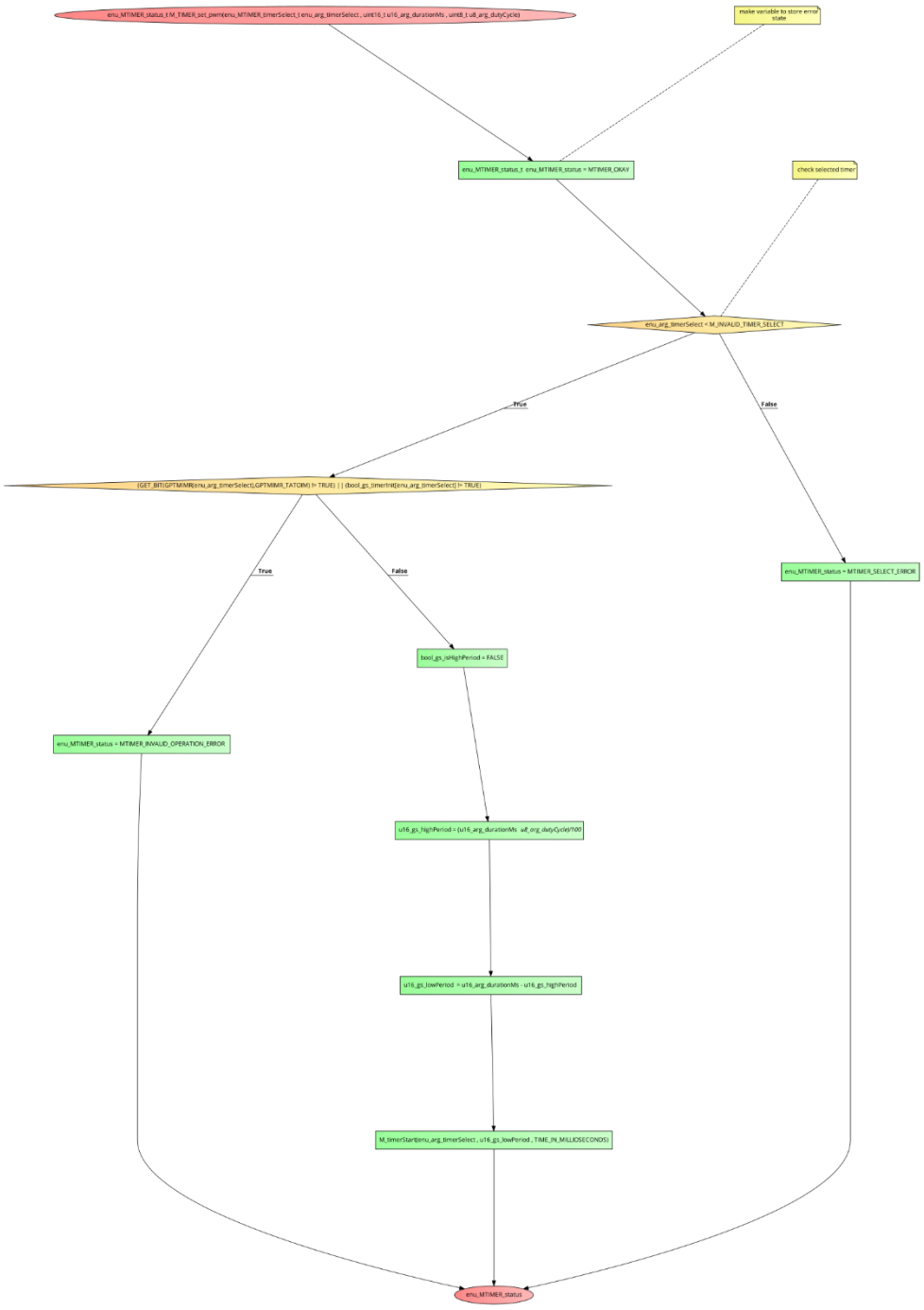
enu_MTIMER_status_t M_TIMER_getRemainingTime (enu_MTIMER_timerSelect_t enu_arg_timerSelect , uint32_t *u32_ptr_timeMs)



enu_MTIMER_status_t M_stopTimer(enu_MTIMER_timerSelect_t enu_arg_timerSelect)

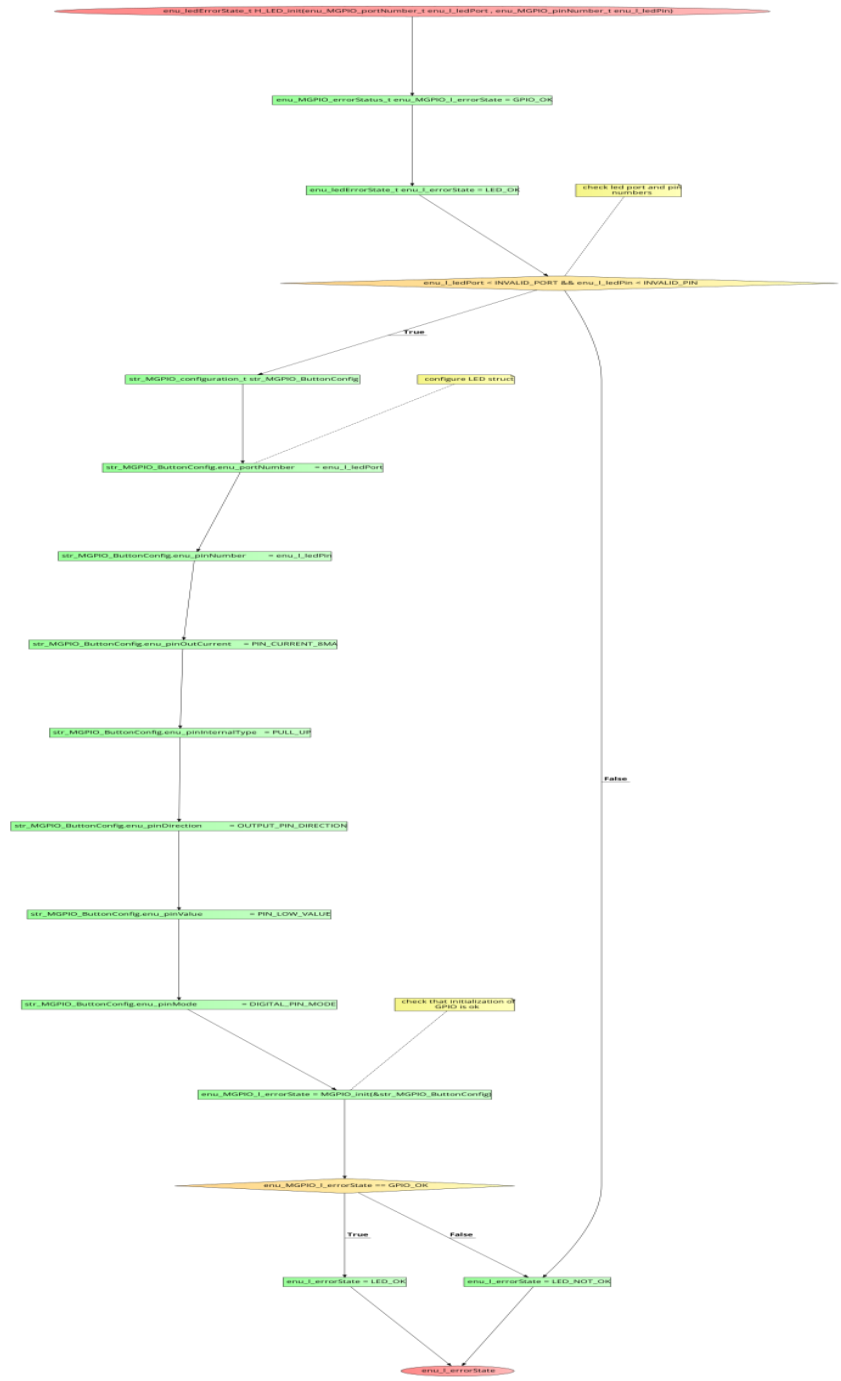

enu_MTIMER_status_t M_stopTimer(enu_MTIMER_timerSelect_t enu_arg_timerSelect)
make variable to store error state
enu_MTIMER_status_t enu_MTIMER_status = MTIMER_OKAY
check selected timer
enu_arg_timerSelect < M_INVALID_TIMER_SELECT
True
False
CLEAR_BIT(GPTMCTL(enu_arg_timerSelect),GPTMCTL_TAEN)
enu_MTIMER_status = MTIMER_SELECT_ERROR
enu_MTIMER_status

enu_MTIMER_status_t M_TIMER_set_pwm(enu_MTIMER_timerSelect_t enu_arg_timerSelect , uint16_t u16_arg_durationMs , uint8_t u8_arg_dutyCycle)

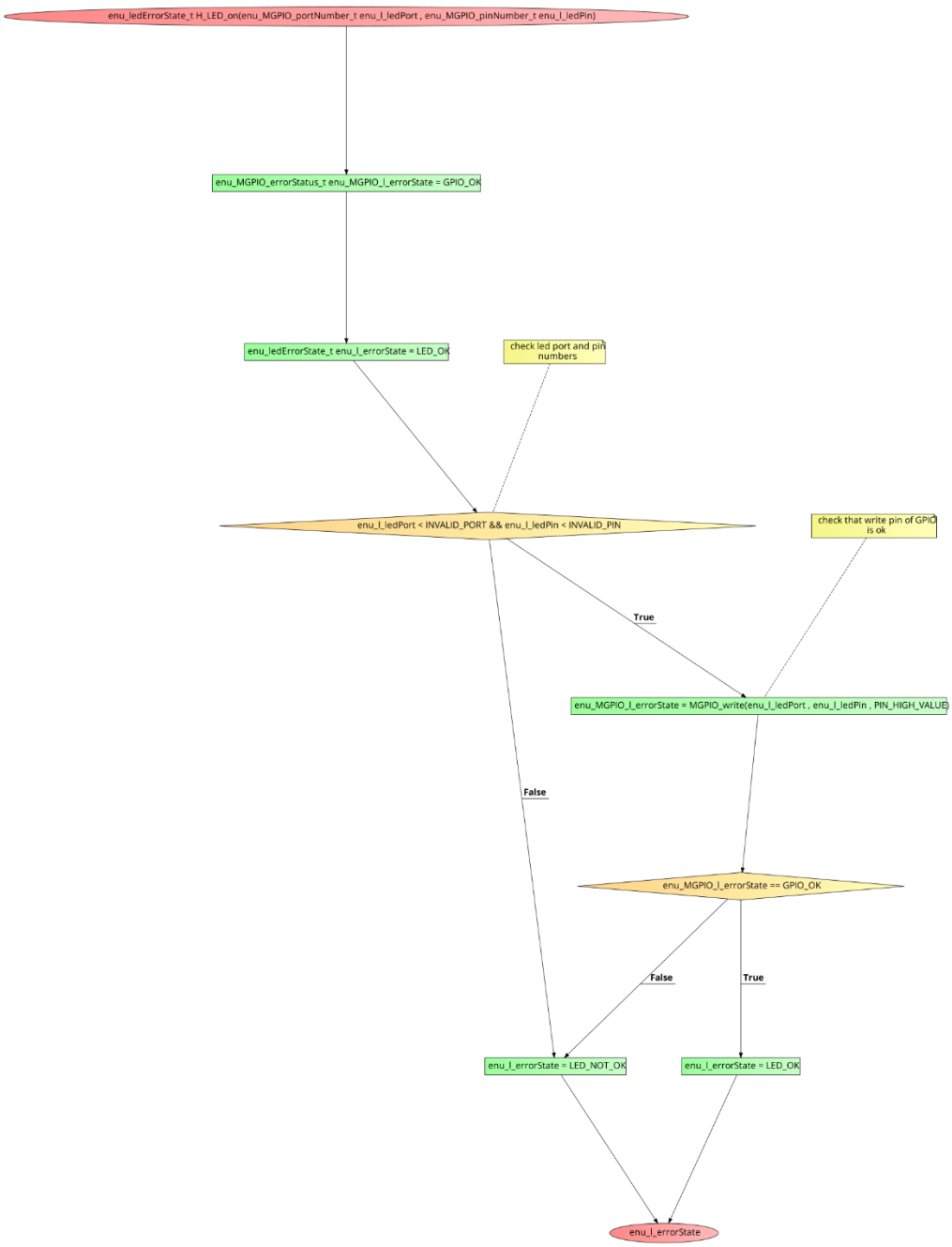


- LED

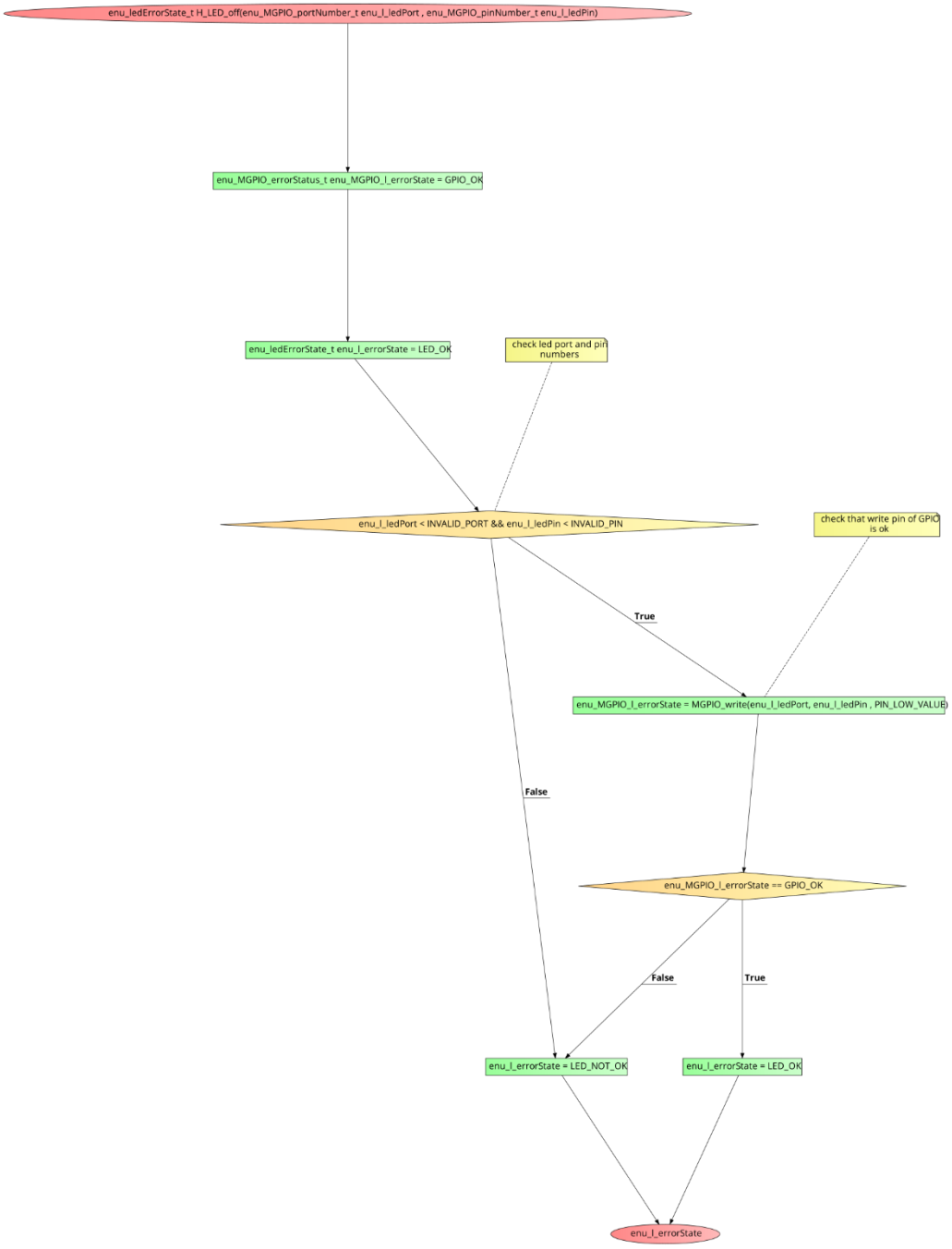
enu_ledErrorState_t H_LED_init(enu_MGPIO_portNumber_t enu_l_ledPort ,
enu_MGPIO_pinNumber_t enu_l_ledPin)



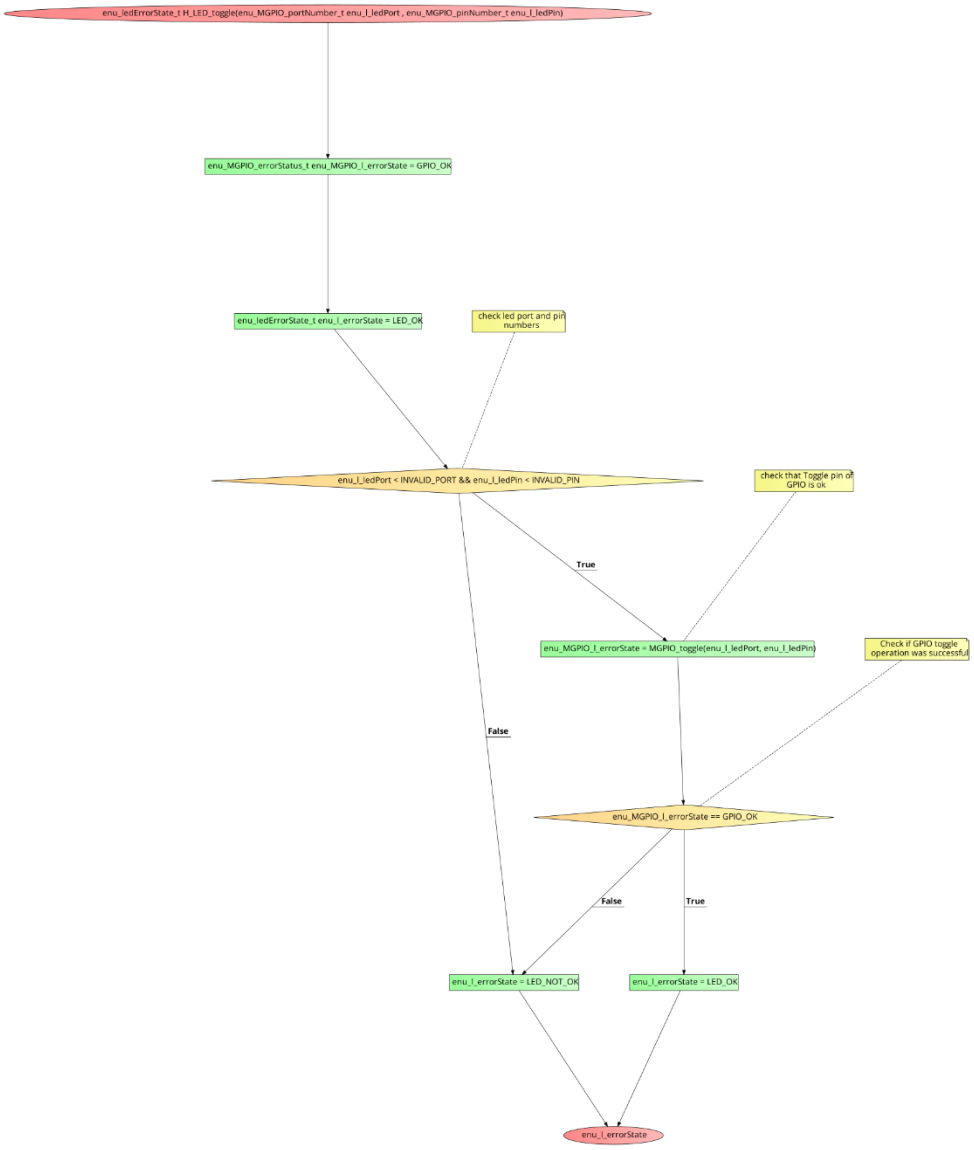
enu_ledErrorState_t H_LED_on(enu_MGPIO_portNumber_t enu_l_ledPort ,
enu_MGPIO_pinNumber_t enu_l_ledPin)



enu_ledErrorState_t H_LED_off(enu_MGPIO_portNumber_t enu_l_ledPort ,
enu_MGPIO_pinNumber_t enu_l_ledPin)

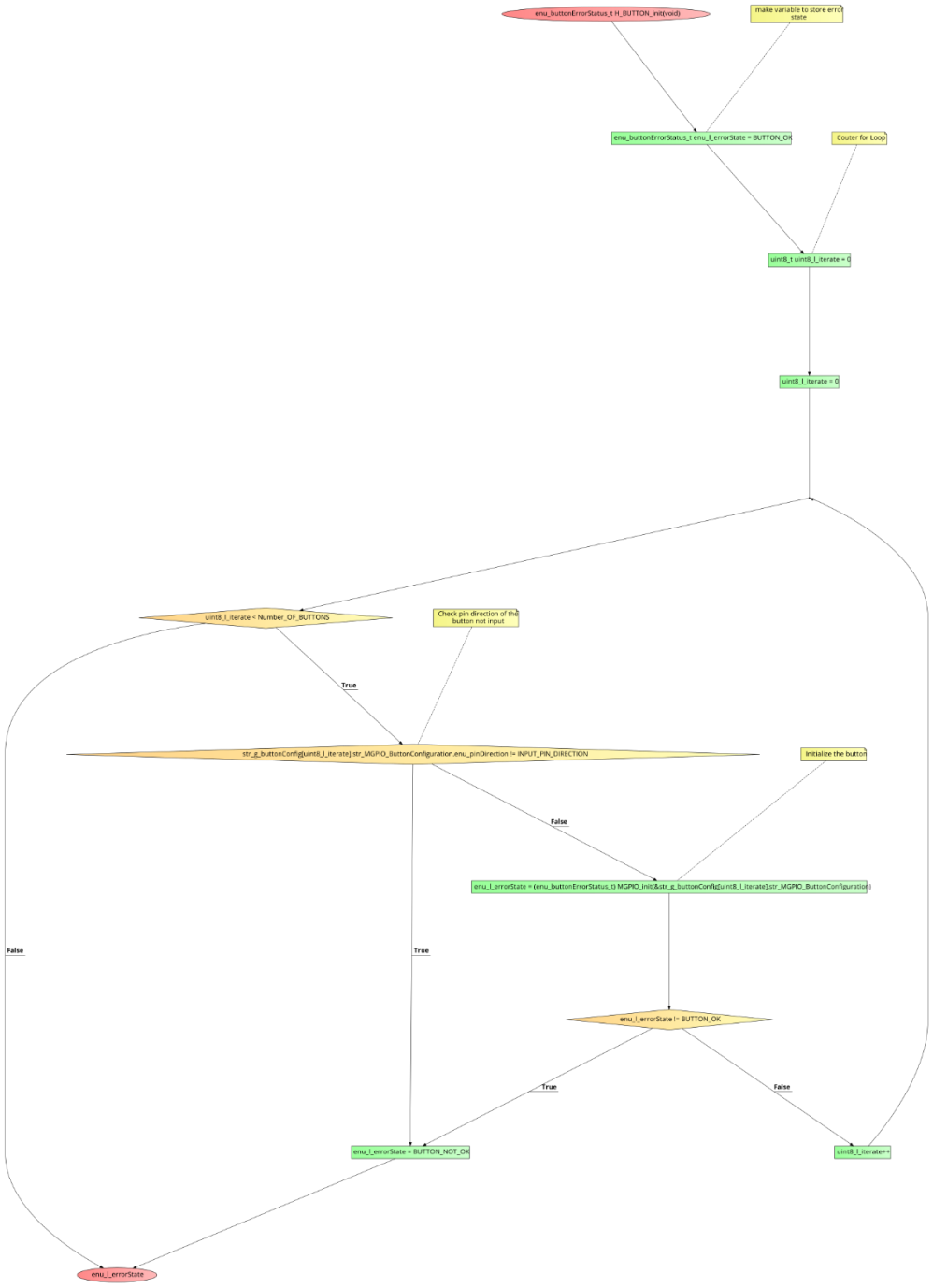


enu_ledErrorState_t H_LED_toggle(enu_MGPIO_portNumber_t enu_l_ledPort , enu_MGPIO_pinNumber_t enu_l_ledPin)

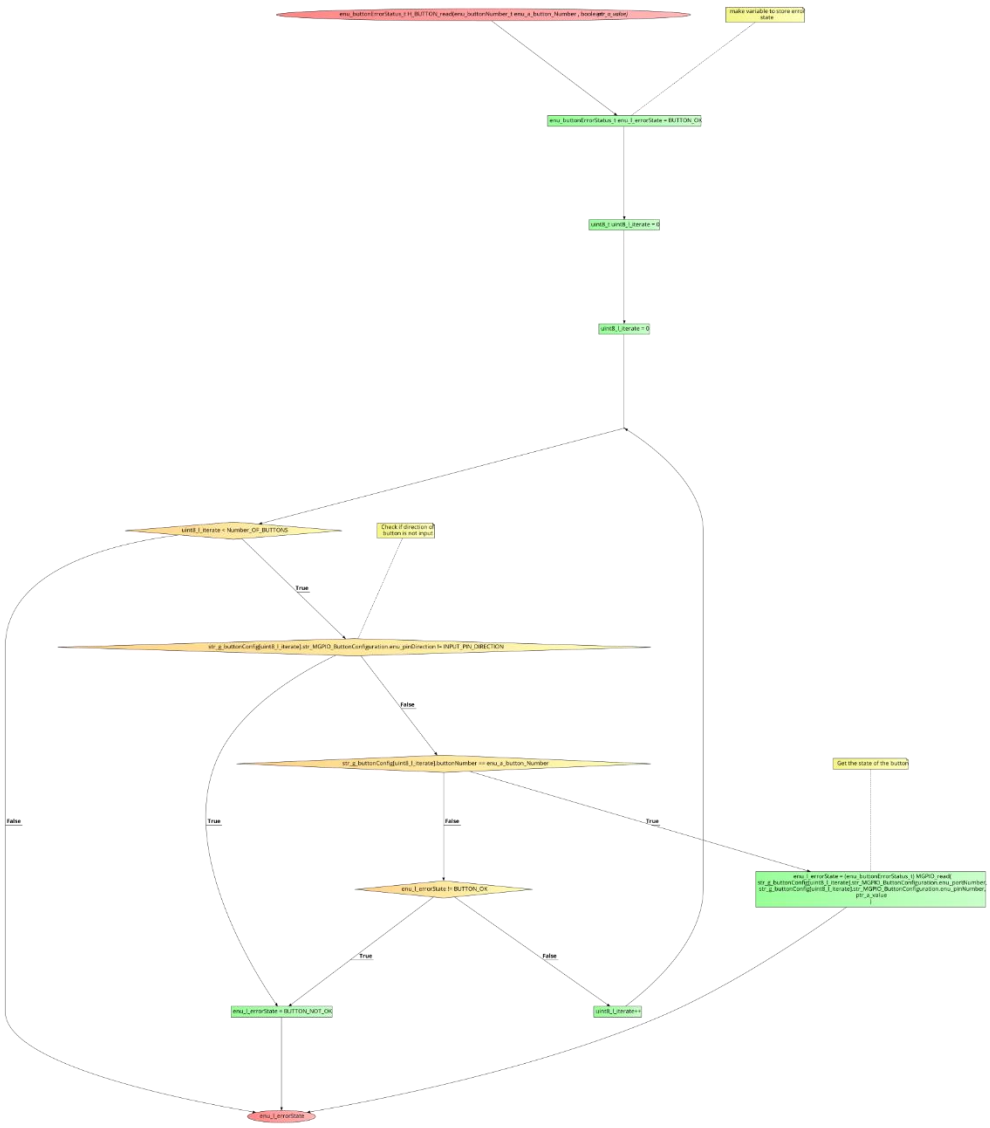

enu_ledErrorState_t H_LED_toggle(enu_MGPIO_portNumber_t enu_l_ledPort , enu_MGPIO_pinNumber_t enu_l_ledPin)
enu_MGPIO_errorStatus_t enu_MGPIO_l_errorState = GPIO_OK
enu_ledErrorState_t enu_l_errorState = LED_OK
check led port and pin numbers
enu_l_ledPort < INVALID_PORT && enu_l_ledPin < INVALID_PIN
check that Toggle pin of GPIO is ok
True
enu_MGPIO_l_errorState = MGPIO_toggle(enu_l_ledPort, enu_l_ledPin)
Check if GPIO toggle operation was successful
False
enu_MGPIO_l_errorState == GPIO_OK
False
True
enu_l_errorState = LED_NOT_OK
enu_l_errorState = LED_OK
enu_l_errorState

➢ BUTTON

enu_buttonErrorStatus_t H_BUTTON_init(void)

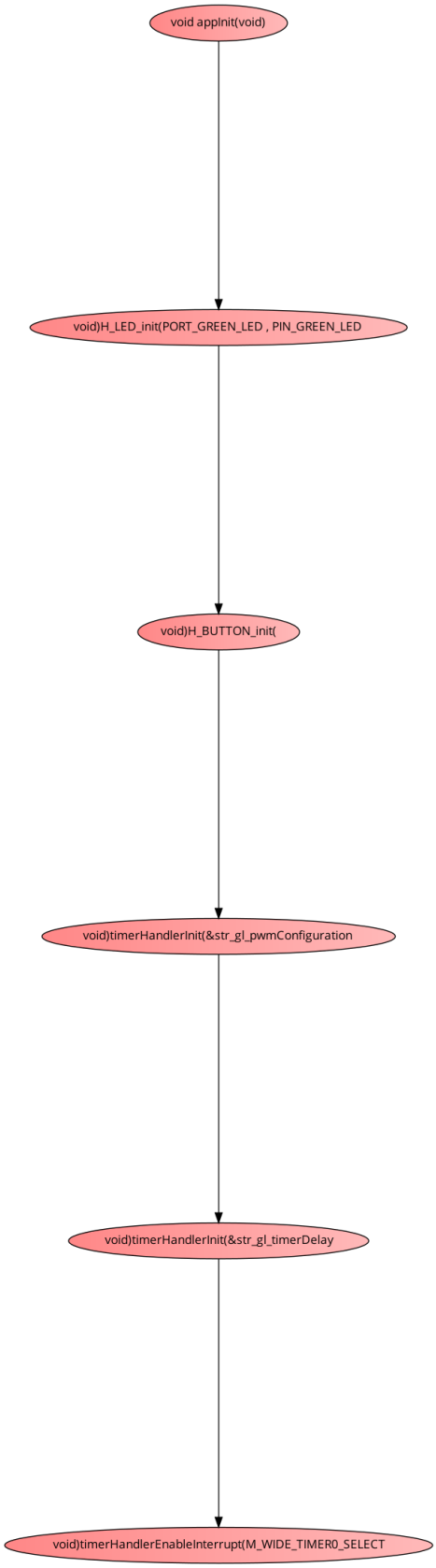


enu_buttonErrorStatus_t H_BUTTON_read(enu_buttonNumber_t enu_a_button_Number , boolean *ptr_a_value)

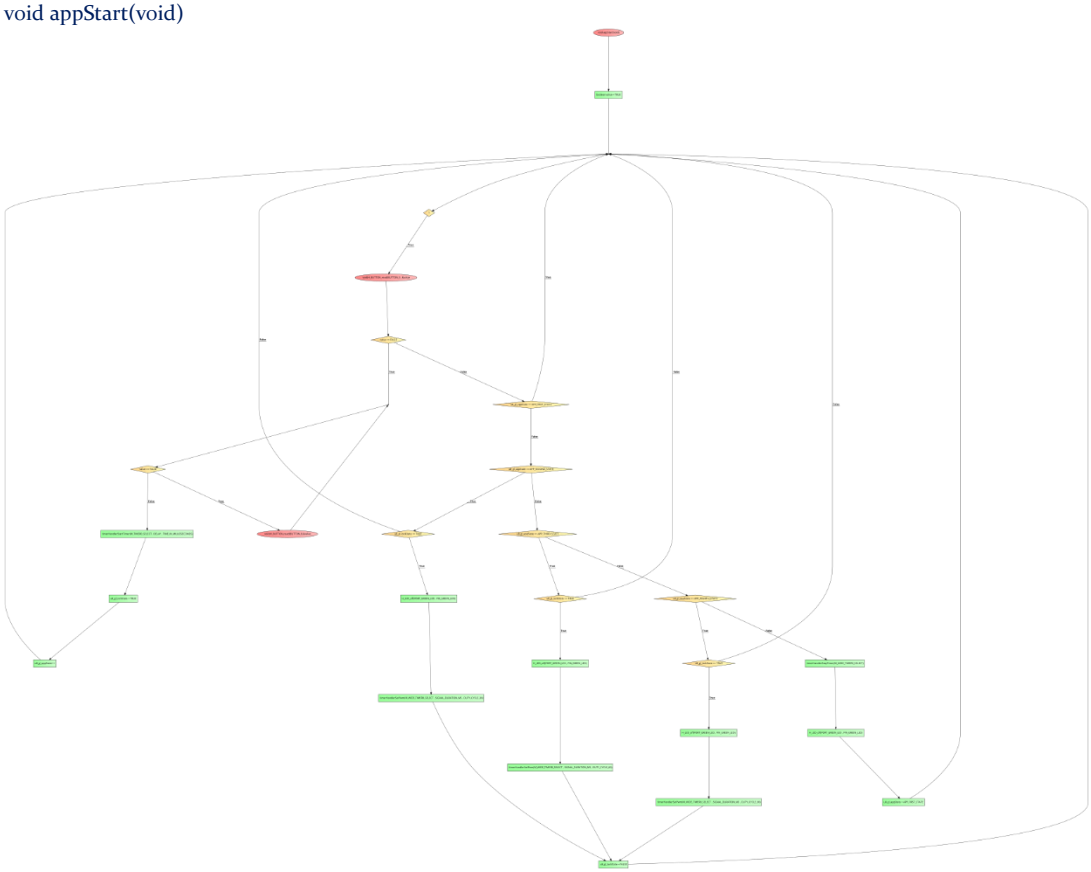


➢ APP

void appInit(void)


void appInit(void)
void)H_LED_init(PORT_GREEN_LED , PIN_GREEN_LED
void)H_BUTTON_init(
void)timerHandlerInit(&str_gl_pwmConfiguration
void)timerHandlerInit(&str_gl_timerDelay
void)timerHandlerEnableInterrupt(M_WIDE_TIMER0_SELECT

void appStart(void)



- Precompiling & Linking Configurations
  - GPIO

```c
/*********************************************************************************/
/*                                HEADER GUARD                                   */
/*********************************************************************************/
#ifndef GPIO_INTERFACE_H
#define GPIO_INTERFACE_H

/*********************************************************************************/
/*                                  INCLUDES                                     */
/*********************************************************************************/
#include "common.h"

/*********************************************************************************/
/*                                CALL BACK FUNC                                 */
/*********************************************************************************/
typedef void (*ptr_MGPIO_callBack_t)(void);


/*********************************************************************************/
/*                                  GPIO PORTS                                   */
/*********************************************************************************/
typedef enum __MGPIO_portNumber
{
  PORTA = 0 ,
  PORTB     ,
  PORTC     ,
  PORTD     ,
  PORTE     ,
  PORTF     ,
  INVALID_PORT
}enu_MGPIO_portNumber_t;
```

```c
/*********************************************************************************/
/*                                  GPIO PINS                                    */
/*********************************************************************************/
typedef enum __MGPIO_pinNumber
{
  PIN0 = 0,
  PIN1    ,
  PIN2    ,
  PIN3    ,
  PIN4    ,
  PIN5    ,
  PIN6    ,
  PIN7    ,
  INVALID_PIN
}enu_MGPIO_pinNumber_t;

/*********************************************************************************/
/*                                GPIO PIN DIRECTION                             */
/*********************************************************************************/
typedef enum __MGPIO_pinDirection
{
  INPUT_PIN_DIRECTION  = 0      ,
  OUTPUT_PIN_DIRECTION          ,
  INVALID_DIRECTION
}enu_MGPIO_pinDirection_t;
```

```
/*                                        GPIO PIN TYPE                                        */
/*************************************************************************************************/
typedef enum __MGPIO_pinType
{
  MGPIO_PIN       = 0,
  ALTERNATIVE_PIN      ,
  INVALID_TYPE
}enu_MGPIO_pinType_t;

/*************************************************************************************************/
/*                              GPIO PIN INTERNAL ATTACH                                         */
/*************************************************************************************************/
typedef enum __MGPIO_pinInternalType
{
  OPEN_DRAIN   = 0,
  PULL_UP         ,
  PULL_DOWN       ,
  INVALID_INTERNAL_TYPE
}enu_MGPIO_pinInternalType_t;

/*************************************************************************************************/
/*                          GPIO PIN TRIGGER INTERRUPTS                                          */
/*************************************************************************************************/
typedef enum __MGPIO_pinEventTrigger
{
  TRIGGER_FALLING_EDGE                          = 0,
  TRIGGER_RISING_EDGE                             ,
  TRIGGER_BOTH_RISING_FALLING_EDGES               ,
  TRIGGER_PIN_LOW                                 ,
  TRIGGER_PIN_HIGH                                ,
  INVALID_TRIGGER
}enu_MGPIO_pinEventTrigger_t;

/*-----------------------------------------------------------------------------------------------*/
/*                                      GPIO PIN MODE                                            */
/*************************************************************************************************/
typedef enum __MGPIO_pinMode
{
  DIGITAL_PIN_MODE  = 0,
  ANALOG_PIN_MODE      ,
  INVALID_MODE
}enu_MGPIO_pinMode_t;

/*************************************************************************************************/
/*                                    GPIO PIN VALUE (LEVEL)                                     */
/*************************************************************************************************/
typedef enum __MGPIO_pinValue
{
  PIN_LOW_VALUE    = 0,
  PIN_HIGH_VALUE      ,
  INVALID_PIN_VALUE
}enu_MGPIO_pinValue_t;


/*************************************************************************************************/
/*                                    GPIO PIN OUT CURRENT                                       */
/*************************************************************************************************/
typedef enum __MGPIO_pinOutCurrent
{
  PIN_CURRENT_2MA  = 0,
  PIN_CURRENT_4MA     ,
  PIN_CURRENT_8MA     ,
  INVALID_OUT_CURRENT
}enu_MGPIO_pinOutCurrent_t;
```

```
/*********************************************************************************
*                                  GPIO ERROR STATUS
*********************************************************************************/
typedef enum __MGPIO_errorStatus
{
  GPIO_OK                          =0,
  GPIO_NULL_POINTER                 ,
  GPIO_PORT_ERROR                   ,
  GPIO_PIN_ERROR                    ,
  GPIO_DIRECTION_ERROR              ,
  GPIO_MODE_ERROR                   ,
  GPIO_PIN_TYPE_ERROR               ,
  GPIO_OUT_CURRENT_ERROR            ,
  GPIO_INTERNAL_TYPE_ERROR          ,
  GPIO_VALUE_ERROR                  ,
  GPIO_EVENT_TRIGGER_ERROR          ,
  GPIO_PORT_NOT_INITIALIZED         ,
  GPIO_NULL_CB_POINTER
}enu_MGPIO_errorStatus_t;


/*********************************************************************************/
/*                        GPIO PIN TRIGGER INTERRUPTS                            */
/*********************************************************************************/
typedef struct __MGPIO_configuration
{
  /*
    options pin number:
      -> 0 : 7
  */
  enu_MGPIO_pinNumber_t           enu_pinNumber;

  /*
    options port number:
      -> MGPIO_PIN
      -> ALTERNATIVE_PIN
  */
  enu_MGPIO_portNumber_t          enu_portNumber;

  /*
    options pin Dir :
      -> INPUT_PIN_DIRECTION
      -> OUTPUT_PIN_DIRECTION
  */
  enu_MGPIO_pinDirection_t        enu_pinDirection;

  /*
    options pin mode:
      -> DIGITAL_PIN_MODE
      -> ANALOG_PIN_MODE
  */
  enu_MGPIO_pinMode_t             enu_pinMode;
```

```
  /*
    options pin type:
      -> DIGITAL_PIN_MODE
      -> ANALOG_PIN_MODE
  */
  enu_MGPIO_pinType_t             enu_pinType;


  /*
    for output direction if direction output
    options pin value:
     -> PIN_LOW_VALUE
     -> PIN_HIGH_VALUE
  */
  enu_MGPIO_pinValue_t          enu_pinValue;

  /*
    options pin out current:
     -> PIN_CURRENT_2mA
     -> PIN_CURRENT_4mA
     -> PIN_CURRENT_8mA
  */
  enu_MGPIO_pinOutCurrent_t     enu_pinOutCurrent;

  /*
    for input direction if direction input
    options pin internal type:
     -> OPEN_DRAIN
     -> PULL_UP
     -> PULL_DOWN
  */
  enu_MGPIO_pinInternalType_t enu_pinInternalType;
}str_MGPIO_configuration_t;
```

➢ GPT

```c
/*********************************************************************************************/
/*                                POINTERTO FUNC CALL BACK                                   */
/*********************************************************************************************/
typedef void(*ptrf_callBack_t)(void);


/*********************************************************************************************/
/*                                      SELECT TIMER                                         */
/*********************************************************************************************/
typedef enum __MTIMER_timerSelect_t
{
    M_TIMER0_SELECT         = 0,
  M_TIMER1_SELECT            ,
  M_TIMER2_SELECT            ,
  M_TIMER3_SELECT              ,
    M_TIMER4_SELECT            ,
    M_TIMER5_SELECT            ,
    M_WIDE_TIMER0_SELECT       ,
    M_WIDE_TIMER1_SELECT       ,
    M_WIDE_TIMER2_SELECT       ,
    M_WIDE_TIMER3_SELECT       ,
    M_WIDE_TIMER4_SELECT       ,
    M_WIDE_TIMER5_SELECT       ,
    M_INVALID_TIMER_SELECT
}enu_MTIMER_timerSelect_t;

/*********************************************************************************************/
/*                                      SELECT MODE                                          */
/*********************************************************************************************/
typedef enum __MTIMER_mode_t
{
    MTIMER_ONE_SHOT_MODE         = 0,
  MTIMER_PERIODIC_MODE           ,
  MTIMER_RTC_MODE                ,
  MTIMER_INPUT_EDGE_COUNT_MODE     ,
    MTIMER_INPUT_EDGE_TIME_MODE    ,
    MTIMER_PWM_MODE                 ,
    MTIMER_INVALID_MODE
}enu_MTIMER_mode_t;

/*********************************************************************************************/
/*                                      SELECT TYPE                                          */
/*********************************************************************************************/
typedef enum __MTIMER_type_t
{
    MTIMER_INDIVIDUAL_TYPE         = 0,
  MTIMER_CONCATENATED_TYPE          ,
    MTIMER_INVALID_TYPE
}enu_MTIMER_type_t;
```

```
*                        TIMER STRUCTURE TO CONFIGURE
*******************************************************************
ypedef struct

    enu_MTIMER_timerSelect_t          enu_timerSelect;
    enu_MTIMER_mode_t                 enu_timerMode;
    enu_MTIMER_type_t                 enu_timerType;
    boolean                           bool_interruptuOk;
    ptrf_callBack_t                   ptrf_callBack;
str_MTIMER_configurations_t;


*******************************************************************
*                                    SELECT TIMER UNIT
*******************************************************************
ypedef enum __TIME_unit_t

    TIME_IN_MICROSECONDS  = 0 ,
    TIME_IN_MILLIOSECONDS     ,
    TIME_IN_SECONDS           ,
    TIME_UNIT_INVALID
enu_timeUnit_t;


*******************************************************************
*                                  ERROR STATE OF TIMER
*******************************************************************
ypedef enum __MTIMER_status_t

    MTIMER_OKAY      = 0            ,
    MTIMER_SELECT_ERROR       ,
    MTIMER_MODE_SELECT_ERROR       ,
    MTIMER_TYPE_SELECT_ERROR       ,
    MTIMER_NULL_REF_ERROR          ,
    MTIMER_INVALID_OPERATION_ERROR ,
    MTIMER_NULL_REF_CB_ERROR       ,
    MTIMER_INVALID_UNIT_ERROR
enu_MTIMER_status_t;
```

- LED

```c
#ifndef LED_INTERFACE_H_
#define LED_INTERFACE_H_

/*******************************************************************************/
/*                                  INCLUDES                                   */
/*******************************************************************************/
#include "led_config.h"

/*******************************************************************************/
/*                                LED ERROR STATE                              */
/*******************************************************************************/
typedef enum __ledErrorState
{
    LED_OK,
    LED_NOT_OK
}enu_ledErrorState_t;

/*******************************************************************************/
/*                                  LED PORTS                                  */
/*******************************************************************************/
#define  PORT_RED_LED      PORTF
#define  PORT_BLUE_LED     PORTF
#define  PORT_GREEN_LED    PORTF

/*******************************************************************************/
/*                                  LED PINS                                   */
/*******************************************************************************/
#define  PIN_RED_LED       PIN1
#define  PIN_BLUE_LED      PIN2
#define  PIN_GREEN_LED     PIN3
```

```c
/*******************************************************************************/
/*                                INCLUDES                                     */
/*******************************************************************************/
#include "gpio_interface.h"

/*******************************************************************************/
/*                                LED COLORS                                   */
/*******************************************************************************/
typedef enum  __ledCOLOR
{
    RED_LED   = 0,
    BLUE_LED     ,
    GREEN_LED    ,
    TOTAL_LEDS
}enu_ledCOLOR_t;

/*******************************************************************************/
/*                                LED CONFIG                                   */
/*******************************************************************************/
typedef struct __ledConfiguration
{
  str_MGPIO_configuration_t      str_MGPIO_ButtonConfiguration;
  enu_ledCOLOR_t                 enu_ledColor;
}str_ledConfiguration_t;

/*******************************************************************************/
/*                              NUMBER OF LEDS                                 */
/*******************************************************************************/
#define NUMBER_OF_LEDS     3
```

```c
#include "led_config.h"

/*******************************************************************************/
/*                                Config Leds by struct                        */
/*******************************************************************************/
const str_ledConfiguration_t str_ledConfiguration[NUMBER_OF_LEDS] =
{
  {
    {
      .enu_portNumber     = PORTF ,
      .enu_pinNumber      = PIN1 ,
      .enu_pinDirection   = OUTPUT_PIN_DIRECTION,
      .enu_pinMode        = DIGITAL_PIN_MODE,
      .enu_pinValue       = PIN_LOW_VALUE ,
      .enu_pinOutCurrent  = PIN_CURRENT_2MA,
      .enu_pinType        = MGPIO_PIN
    }
    , RED_LED
  },
  {
    {
      .enu_portNumber        = PORTF ,
      .enu_pinNumber         = PIN2 ,
      .enu_pinDirection      = OUTPUT_PIN_DIRECTION,
      .enu_pinMode           = DIGITAL_PIN_MODE,
      .enu_pinValue          = PIN_LOW_VALUE ,
      .enu_pinOutCurrent     = PIN_CURRENT_2MA,
      .enu_pinType           = MGPIO_PIN
    }
    , BLUE_LED
  },
  {
    {
      .enu_portNumber    = PORTF ,
      .enu_pinNumber     = PIN3 ,
      .enu_pinDirection  = OUTPUT_PIN_DIRECTION ,
      .enu_pinMode       = DIGITAL_PIN_MODE,
      .enu_pinValue      = PIN_LOW_VALUE ,
      .enu_pinOutCurrent = PIN_CURRENT_2MA,
      .enu_pinType       = MGPIO_PIN
    }
    , GREEN_LED
  }
};
```

➢ BUTTON

```c
/*******************************************************************************/
/*                                HEADER GUARD                                 */
/*******************************************************************************/
#ifndef BUTTON_H_
#define BUTTON_H_

/*******************************************************************************/
/*                                Includes                                     */
/*******************************************************************************/
#include "button_config.h"

/*******************************************************************************/
/*                                Buttons Error State                          */
/*******************************************************************************/
typedef enum __buttonErrorStatus_t
{
    BUTTON_OK = 0,
    BUTTON_NOT_OK
} enu_buttonErrorStatus_t;
```

```c
#ifndef BUTTON_CONFIG_H_
#define BUTTON_CONFIG_H_

/*******************************************************************************/
/*                                Includes                                     */
/*******************************************************************************/
#include "gpio_interface.h"

/*******************************************************************************/
/*                                Number of Buttons                            */
/*******************************************************************************/
typedef enum __buttonNumber_t
{
  BUTTON_0  = 1 ,
  BUTTON_1      ,
  BUTTON_MAX
}enu_buttonNumber_t;


/*******************************************************************************/
/*                                Button Configuration                         */
/*******************************************************************************/
typedef struct __buttonConfig_t
{
  str_MGPIO_configuration_t    str_MGPIO_ButtonConfiguration;
  enu_buttonNumber_t           buttonNumber;
}str_buttonConfiguration_t;

/*******************************************************************************/
/*                                Number Of Buttons                            */
/*******************************************************************************/
#define Number_OF_BUTTONS      2

```

```c
/*                                  Includes                                  */
/****************************************************************************/
#include "button_config.h"

/****************************************************************************/
/*                              Config Buttons by struct                      */
/****************************************************************************/
const str_buttonConfiguration_t str_g_buttonConfig[Number_OF_BUTTONS] =
{
  {
    {
        /*Button 0 */
        .enu_portNumber          = PORTF ,
        .enu_pinNumber           = PIN4 ,
        .enu_pinDirection        = INPUT_PIN_DIRECTION ,
        .enu_pinMode             = DIGITAL_PIN_MODE,
        .enu_pinInternalType     = PULL_UP,
        .enu_pinType             = MGPIO_PIN

    }
    , BUTTON_0
  },
  {
    {
       /*Button 1 */
      .enu_portNumber          = PORTF ,
      .enu_pinNumber           = PIN0 ,
      .enu_pinDirection        = INPUT_PIN_DIRECTION,
      .enu_pinMode             = DIGITAL_PIN_MODE,
      .enu_pinInternalType     = PULL_UP,
      .enu_pinType             = MGPIO_PIN
    }
    , BUTTON_1
  }
};
```

That is all requirements

Thanks