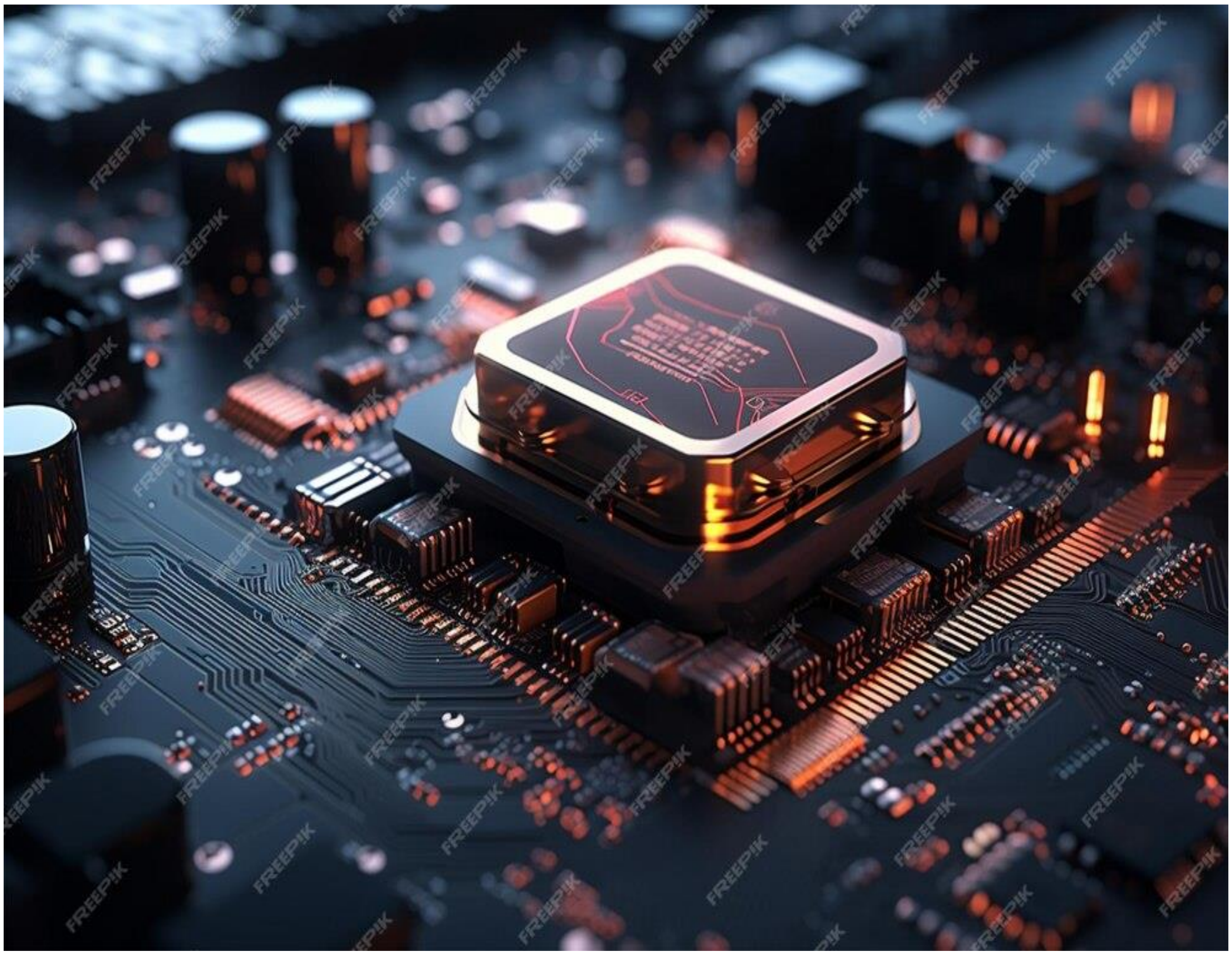


RGB LED Control V2.0 Design

Peter Essam | ARM | 21/11/2023

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.

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.
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

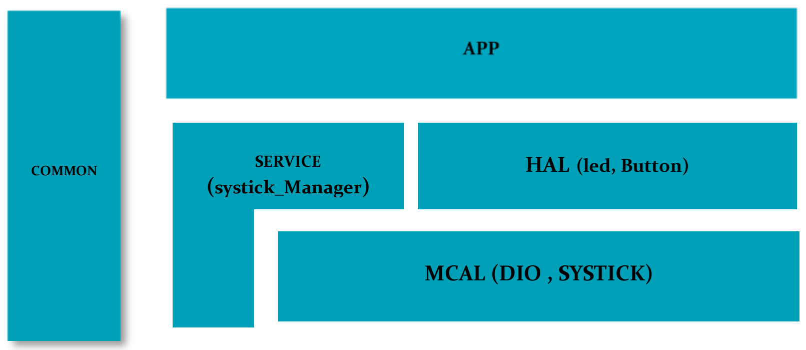
Standard library names in the common layer ensure consistent coding practices.

- ## 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

- SYSTICK

(This Driver Located in MCAL Layer)

1-
enu_SysTick_Error_t SysTick_Init(uint32_t reload_value, enu_SysTick_ClockSource_t clk_source)

Description:
Initialize the SysTick timer.
Arguments:
reload_value: The value to load into the SysTick Reload register.
param clk_source: The clock source for SysTick
Return:
enu_SysTick_Error_t: Error status after initialization.

2-
enu_SysTick_Error_t SysTick_Start(void)

Description:
Start the SysTick timer.
Return:
enu_SysTick_Error_t: Error status after start the Systick.

3-
enu_SysTick_Error_t SysTick_Stop(void)
Description:
Stop the SysTick timer.
Return:
enu_SysTick_Error_t: Error status after stop the Systick.

4-
enu_SysTick_Error_t SysTick_DelayMs(uint32_t delay_ms)
Description:
Delay the execution for a specified number of milliseconds using SysTick
Arguments:
The delay time in milliseconds.
Return:
enu_SysTick_Error_t: Error status after delay.

5-
enu_SysTick_Error_t SysTick_DelayUs(uint32_t delay_us)

Description:
Delay the execution for a specified number of microseconds using SysTick

Arguments:
The delay time in microseconds.

Return:
enu_SysTick_Error_t: Error status after delay.

6-
uint8_t SysTick_CheckTimeOut(void)

Description:
Check Systick timer reached Zero

Return:
1 if the COUNTFLAG is set, indicating a timeout; otherwise, 0.

- ## 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.

➢ SYSTICK_MANAGER
(This Driver Located in Service Layer)

1-
enu_SysTickManager_Error_t SysTickManager_Init(uint32_t reload_value, enu_SysTick_ClockSource_t clk_source)

Description:
Initialize the SysTick timer .
Arguments:
reload_value: The reload value for the SysTick timer.
clk_source: The clock source for the SysTick timer.
Return
enu_SysTickManager_Error_t: Error status.

2-
enu_SysTickManager_Error_t SysTickManager_Start(void)

Description:
Start the SysTick timer .
Return
enu_SysTickManager_Error_t: Error status.

3-
enu_SysTickManager_Error_t SysTickManager_Stop(void)

Description:
Stop the SysTick timer .
Return
enu_SysTickManager_Error_t: Error status.

4-
enu_SysTickManager_Error_t SysTickManager_DelayMs(uint32_t delay_ms)
Description:
Dleay the SysTick timer .
Arguments:
delay_ms: The value in millisec for the SysTick timer want delay.

Return
enu_SysTickManager_Error_t: Error status.

5-
enu_SysTickManager_Error_t SysTickManager_DelayUs(uint32_t delay_us)

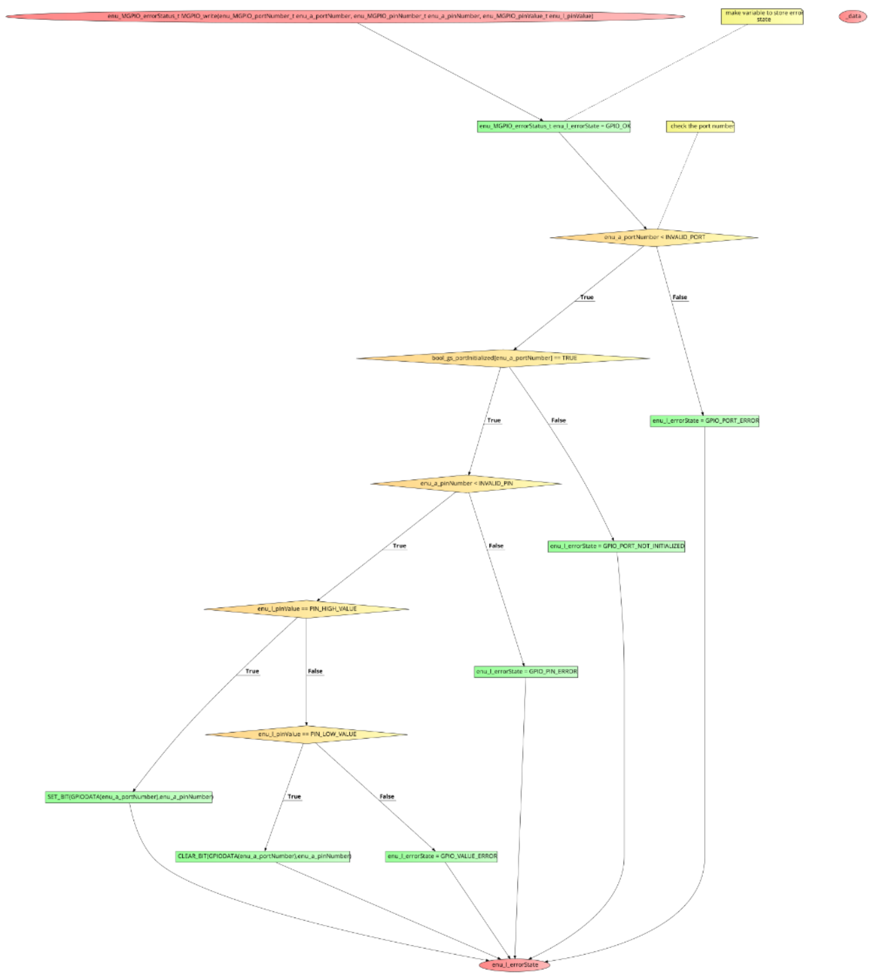
Description:
Dleay the SysTick timer .
Arguments:
delay_us: The value in microsec for the SysTick timer want delay.

Return
enu_SysTickManager_Error_t: Error status.

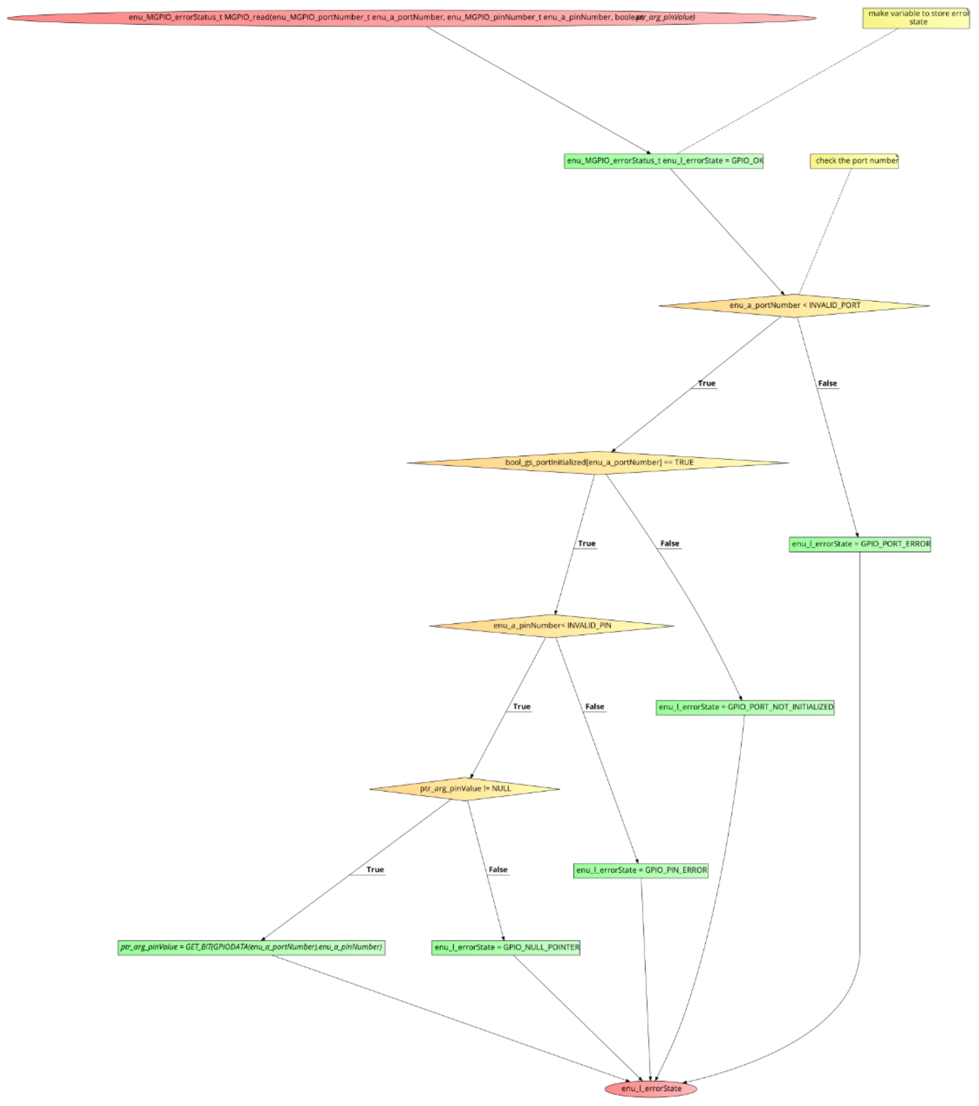
Low Level Design

- Flow Chart
  - GPIO

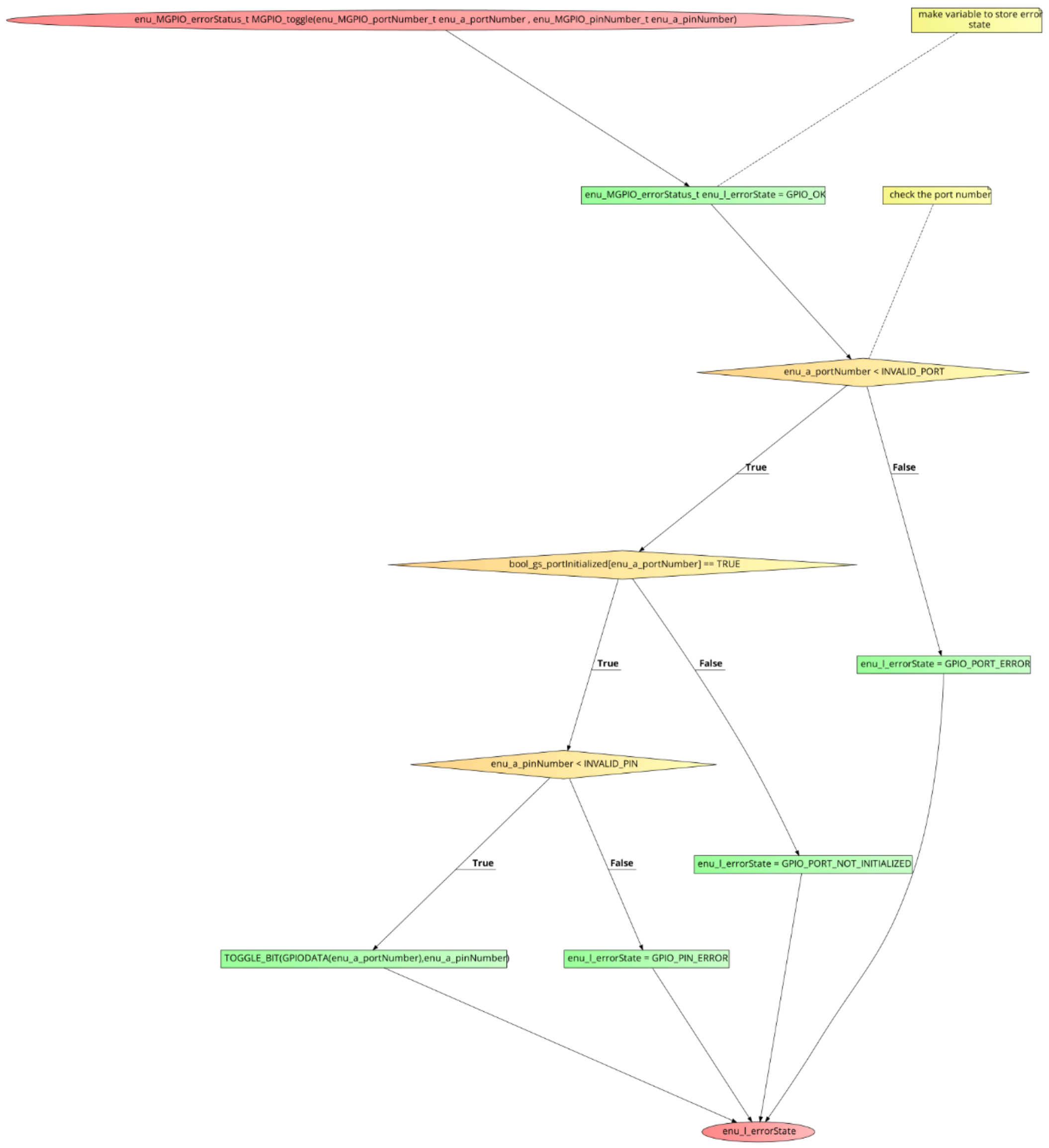
enu_MGPIO_errorStatus_t MGPIO_write(enu_MGPIO_portNumber_t enu_a_portNumber, enu_MGPIO_pinNumber_t enu_a_pinNumber, enu_MGPIO_pinValue_t enu_l_pinValue)


enu_MGPIO_errorStatus_t MGPIO_write(enu_MGPIO_portNumber_t enu_a_portNumber, enu_MGPIO_pinNumber_t enu_a_pinNumber, enu_MGPIO_pinValue_t enu_l_pinValue)
make variable to store error state
_data
enu_MGPIO_errorStatus_t enu_l_errorState = GPIO_OK
check the port number
enu_a_portNumber < INVALID_PORT
True
False
bool_gs_portInitialized[enu_a_portNumber] == TRUE
True
False
enu_l_errorState = GPIO_PORT_ERROR
enu_a_pinNumber < INVALID_PIN
True
False
enu_l_errorState = GPIO_PORT_NOT_INITIALIZED
enu_l_pinValue == PIN_HIGH_VALUE
True
False
enu_l_errorState = GPIO_PIN_ERROR
enu_l_pinValue == PIN_LOW_VALUE
SET_BIT(GPIODATA[enu_a_portNumber],enu_a_pinNumber)
True
False
CLEAR_BIT(GPIODATA(enu_a_portNumber),enu_a_pinNumber)
enu_l_errorState = GPIO_VALUE_ERROR
enu_l_errorState

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

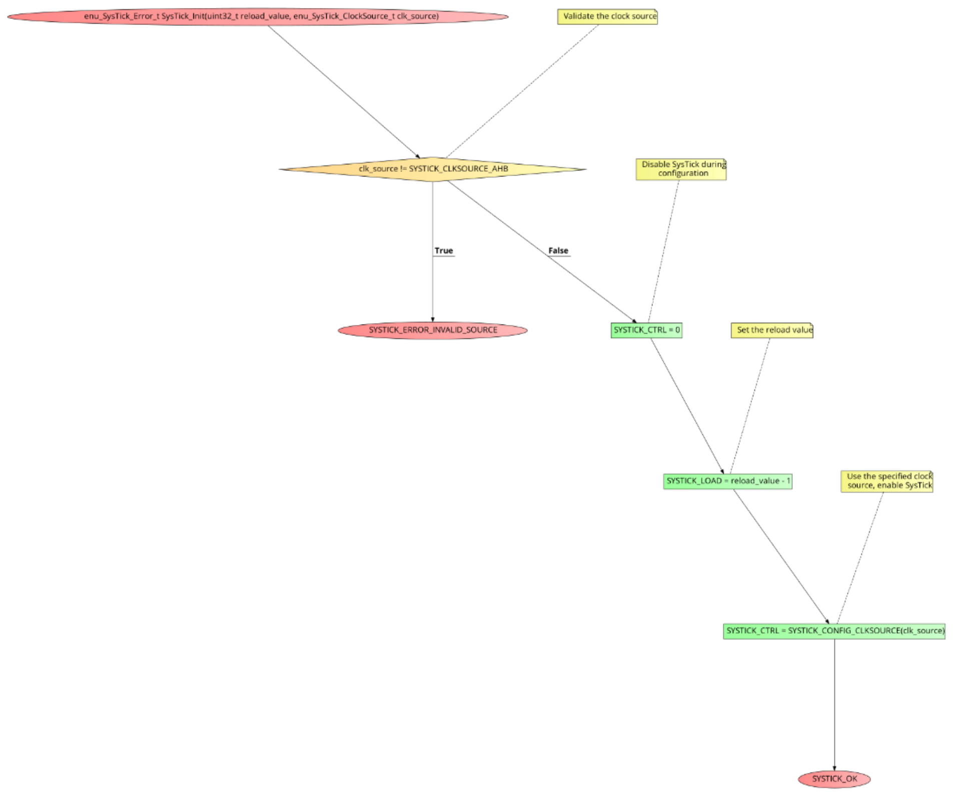

enu_MGPIO_errorStatus_t MGPIO_read(enu_MGPIO_portNumber_t enu_a_portNumber, enu_MGPIO_pinNumber_t enu_a_pinNumber, boolean ptr_arg_pinValue)
make variable to store error state
enu_MGPIO_errorStatus_t enu_l_errorState = GPIO_OK
check the port number
enu_a_portNumber < INVALID_PORT
True
False
bool_gs_portInitialized[enu_a_portNumber] == TRUE
True
False
enu_l_errorState = GPIO_PORT_ERROR
enu_a_pinNumber< INVALID_PIN
True
False
enu_l_errorState = GPIO_PORT_NOT_INITIALIZED
ptr_arg_pinValue != NULL
True
False
enu_l_errorState = GPIO_PIN_ERROR
ptr_arg_pinValue = GET_BIT(GPIODATA(enu_a_portNumber),enu_a_pinNumber)
enu_l_errorState = GPIO_NULL_POINTER
enu_l_errorState

enu_MGPIO_errorStatus_t MGPIO_toggle(enu_MGPIO_portNumber_t enu_a_portNumber , enu_MGPIO_pinNumber_t enu_a_pinNumber)


enu_MGPIO_errorStatus_t MGPIO_toggle(enu_MGPIO_portNumber_t enu_a_portNumber , enu_MGPIO_pinNumber_t enu_a_pinNumber)
make variable to store error state
enu_MGPIO_errorStatus_t enu_l_errorState = GPIO_OK
check the port number
enu_a_portNumber < INVALID_PORT
True
False
bool_gs_portInitialized[enu_a_portNumber] == TRUE
True
False
enu_l_errorState = GPIO_PORT_ERROR
enu_a_pinNumber < INVALID_PIN
True
False
enu_l_errorState = GPIO_PORT_NOT_INITIALIZED
TOGGLE_BIT(GPIODATA(enu_a_portNumber),enu_a_pinNumber)
enu_l_errorState = GPIO_PIN_ERROR
enu_l_errorState

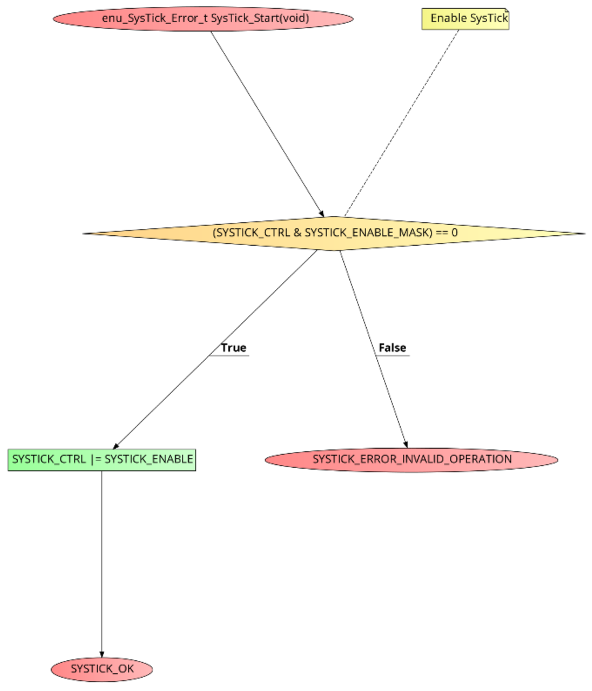
➢ SYSTICK

enu_SysTick_Error_t SysTick_Init(uint32_t reload_value, enu_SysTick_ClockSource_t clk_source)



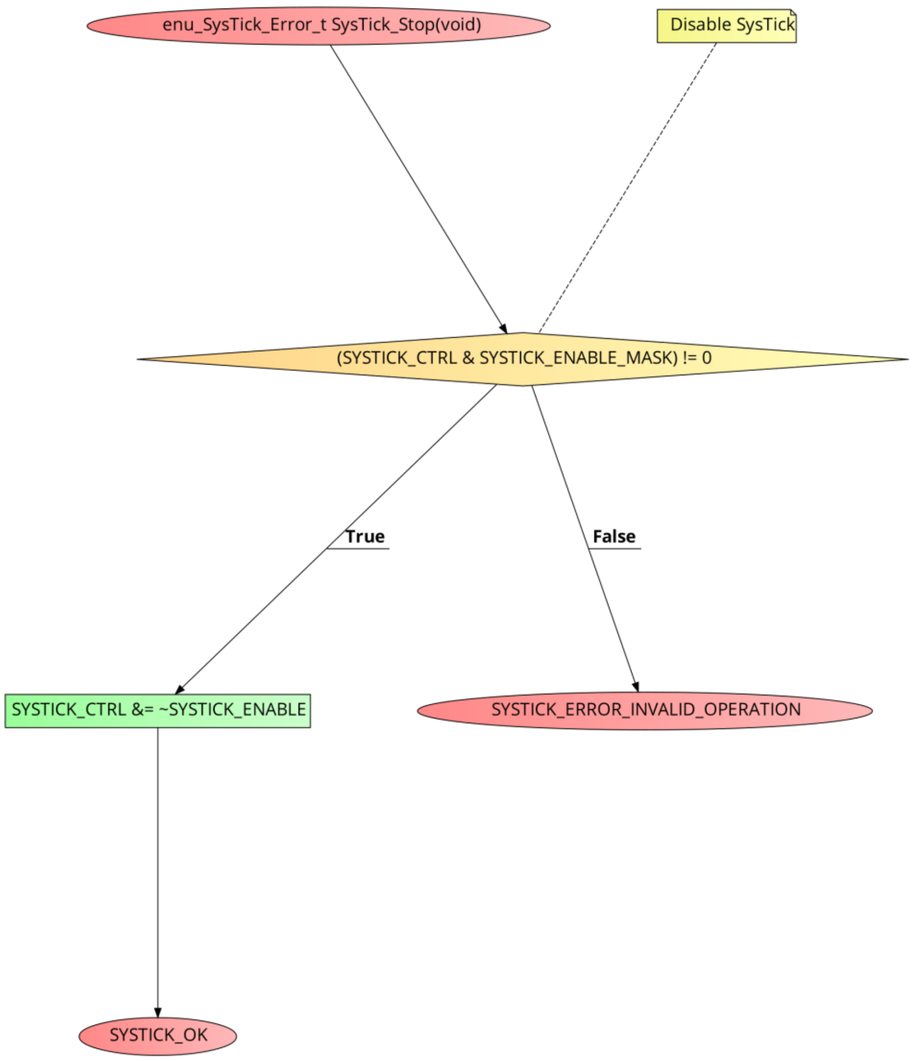
enu_SysTick_Error_t SysTick_Init(uint32_t reload_value, enu_SysTick_ClockSource_t clk_source)
Validate the clock source
clk_source != SYSTICK_CLKSOURCE_AHB
Disable SysTick during configuration
True
False
SYSTICK_ERROR_INVALID_SOURCE
SYSTICK_CTRL = 0
Set the reload value
SYSTICK_LOAD = reload_value - 1
Use the specified clock source, enable SysTick
SYSTICK_CTRL = SYSTICK_CONFIG_CLKSOURCE(clk_source)
SYSTICK_OK

enu_SysTick_Error_t SysTick_Start(void)

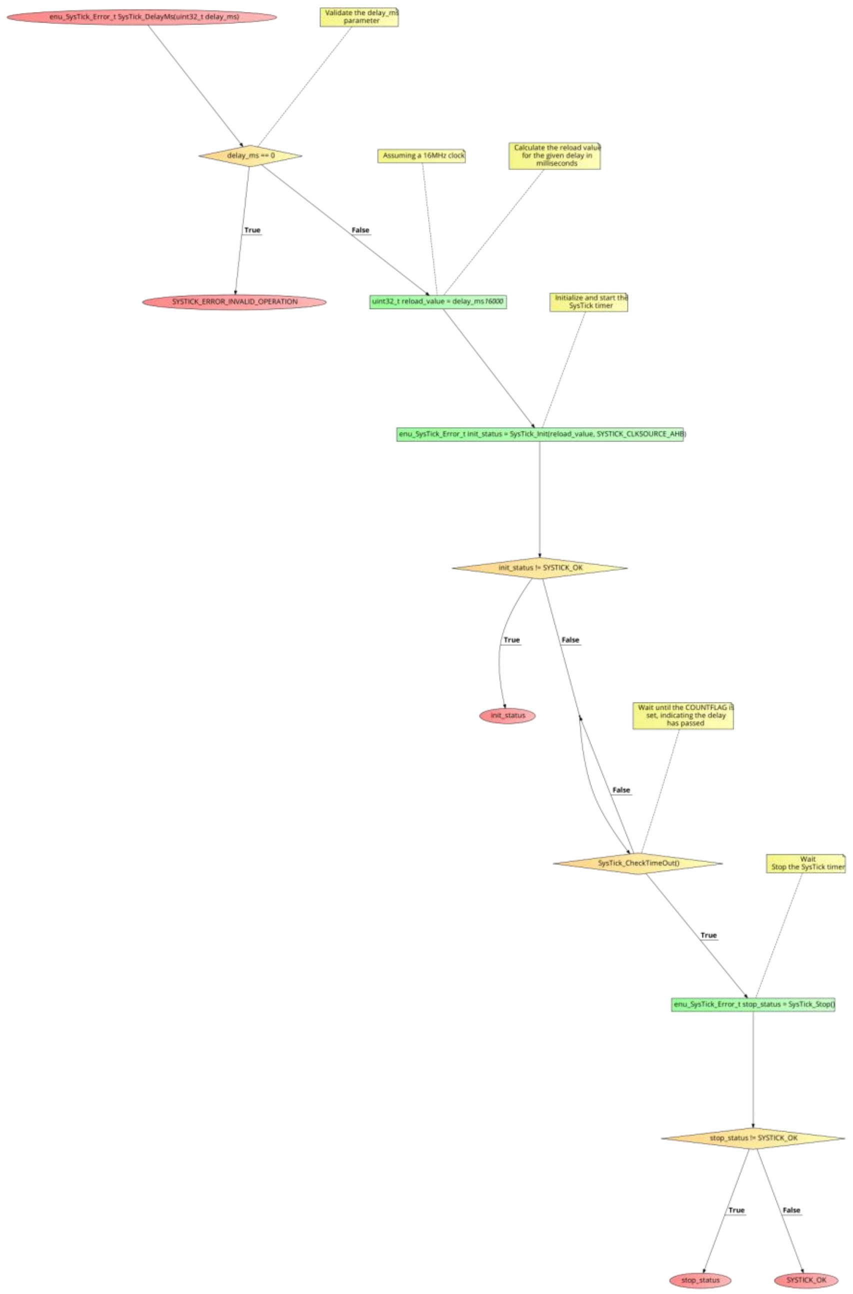


enu_SysTick_Error_t SysTick_Start(void)
Enable SysTick
(SYSTICK_CTRL & SYSTICK_ENABLE_MASK) == 0
True
False
SYSTICK_CTRL |= SYSTICK_ENABLE
SYSTICK_ERROR_INVALID_OPERATION
SYSTICK_OK

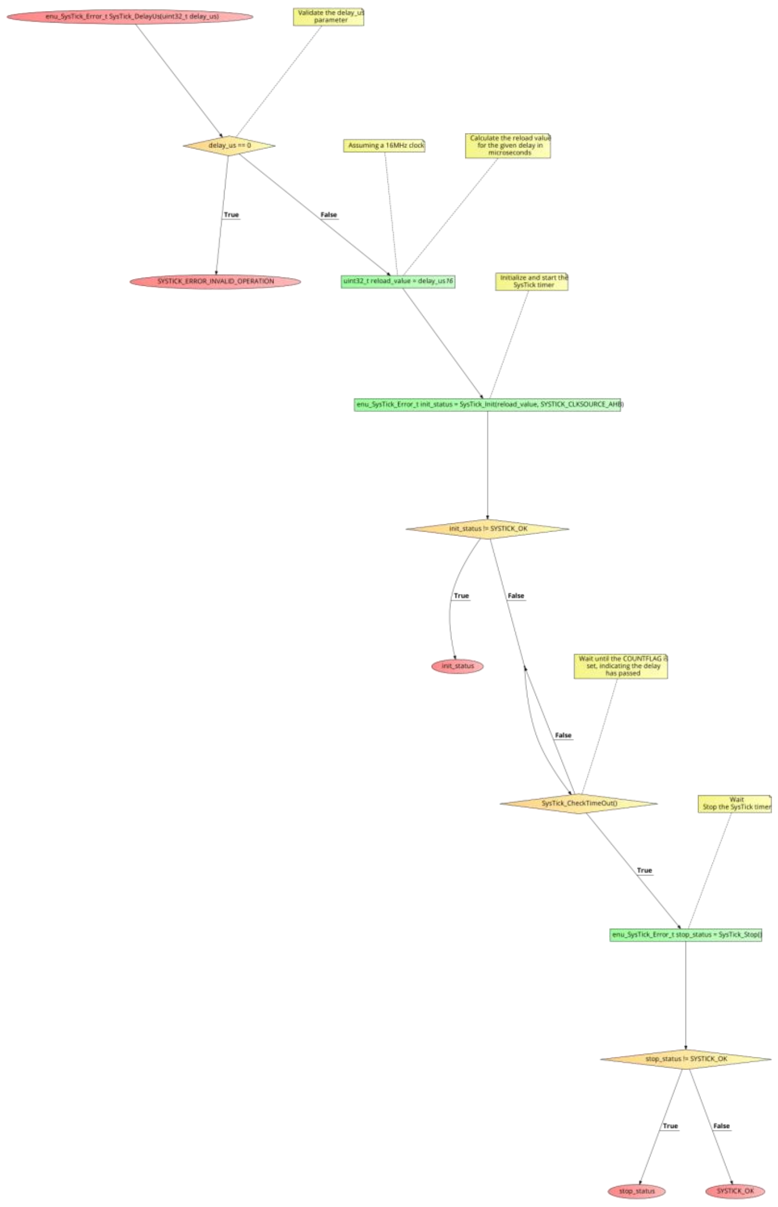
enu_SysTick_Error_t SysTick_Stop(void)


enu_SysTick_Error_t SysTick_Stop(void)
Disable SysTick
(SYSTICK_CTRL & SYSTICK_ENABLE_MASK) != 0
True
False
SYSTICK_CTRL &= ~SYSTICK_ENABLE
SYSTICK_ERROR_INVALID_OPERATION
SYSTICK_OK

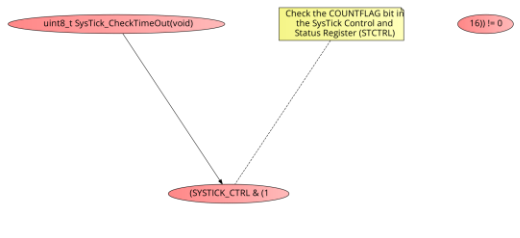
enu_SysTick_Error_t SysTick_DelayMs(uint32_t delay_ms)


enu_SysTick_Error_t SysTick_DelayMs(uint32_t delay_ms)
Validate the delay_ms parameter
delay_ms == 0
True
False
SYSTICK_ERROR_INVALID_OPERATION
Assuming a 16MHz clock
Calculate the reload value for the given delay in milliseconds
uint32_t reload_value = delay_ms*16000
Initialize and start the SysTick timer
enu_SysTick_Error_t init_status = SysTick_Init(reload_value, SYSTICK_CLKSOURCE_AHB)
init_status != SYSTICK_OK
True
False
init_status
Wait until the COUNTFLAG is set, indicating the delay has passed
False
SysTick_CheckTimeOut()
Wait
Stop the SysTick timer
True
enu_SysTick_Error_t stop_status = SysTick_Stop()
stop_status != SYSTICK_OK
True
False
stop_status
SYSTICK_OK

enu_SysTick_Error_t SysTick_DelayUs(uint32_t delay_us)

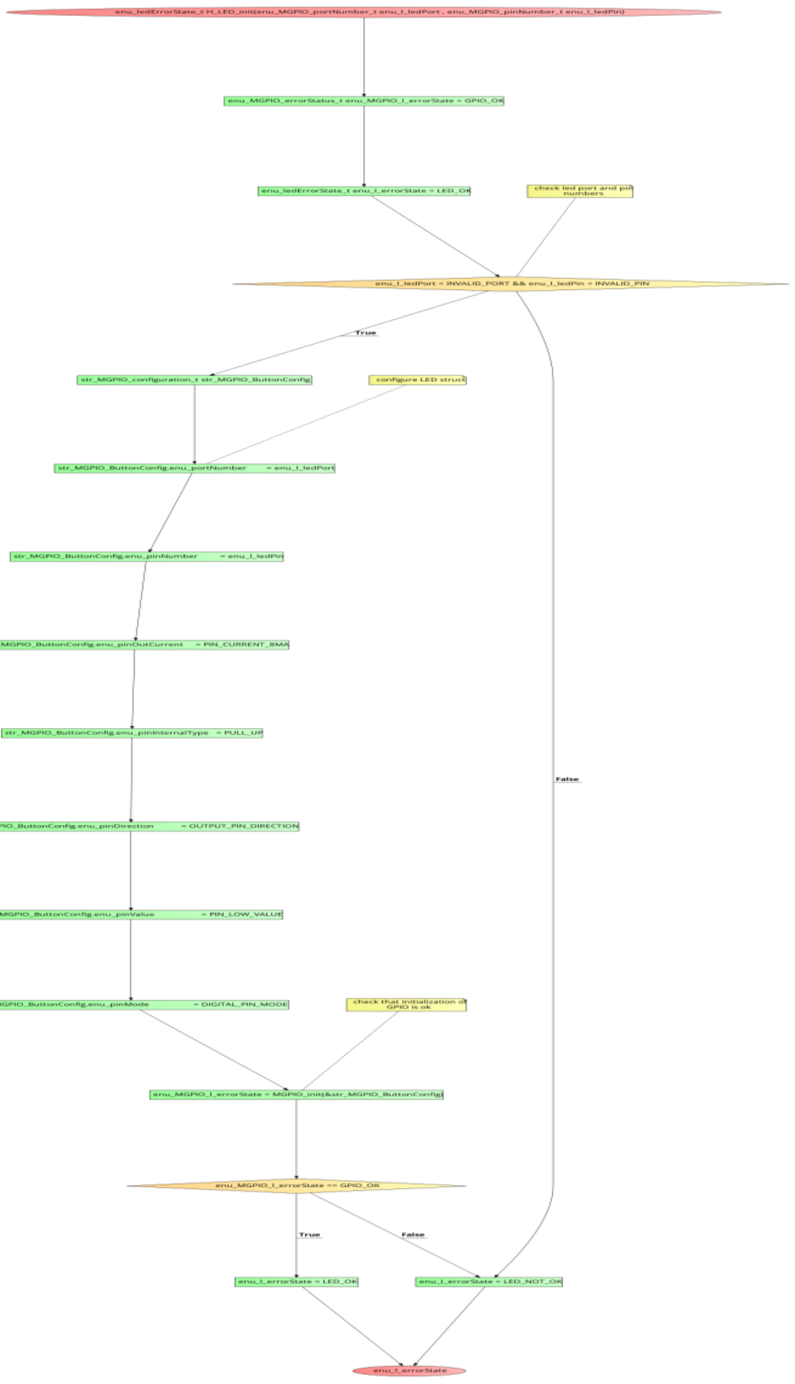

enu_SysTick_Error_t SysTick_DelayUs(uint32_t delay_us)
Validate the delay_us parameter
delay_us == 0
True
False
SYSTICK_ERROR_INVALID_OPERATION
Assuming a 16MHz clock
Calculate the reload value for the given delay in microseconds
uint32_t reload_value = delay_us?6
Initialize and start the SysTick timer
enu_SysTick_Error_t init_status = SysTick_Init(reload_value, SYSTICK_CLKSOURCE_AHB)
init_status != SYSTICK_OK
True
False
init_status
Wait until the COUNTFLAG is set, indicating the delay has passed
False
SysTick_CheckTimeOut()
Wait
Stop the SysTick timer
True
enu_SysTick_Error_t stop_status = SysTick_Stop()
stop_status != SYSTICK_OK
True
False
stop_status
SYSTICK_OK

uint8_t SysTick_CheckTimeOut(void)

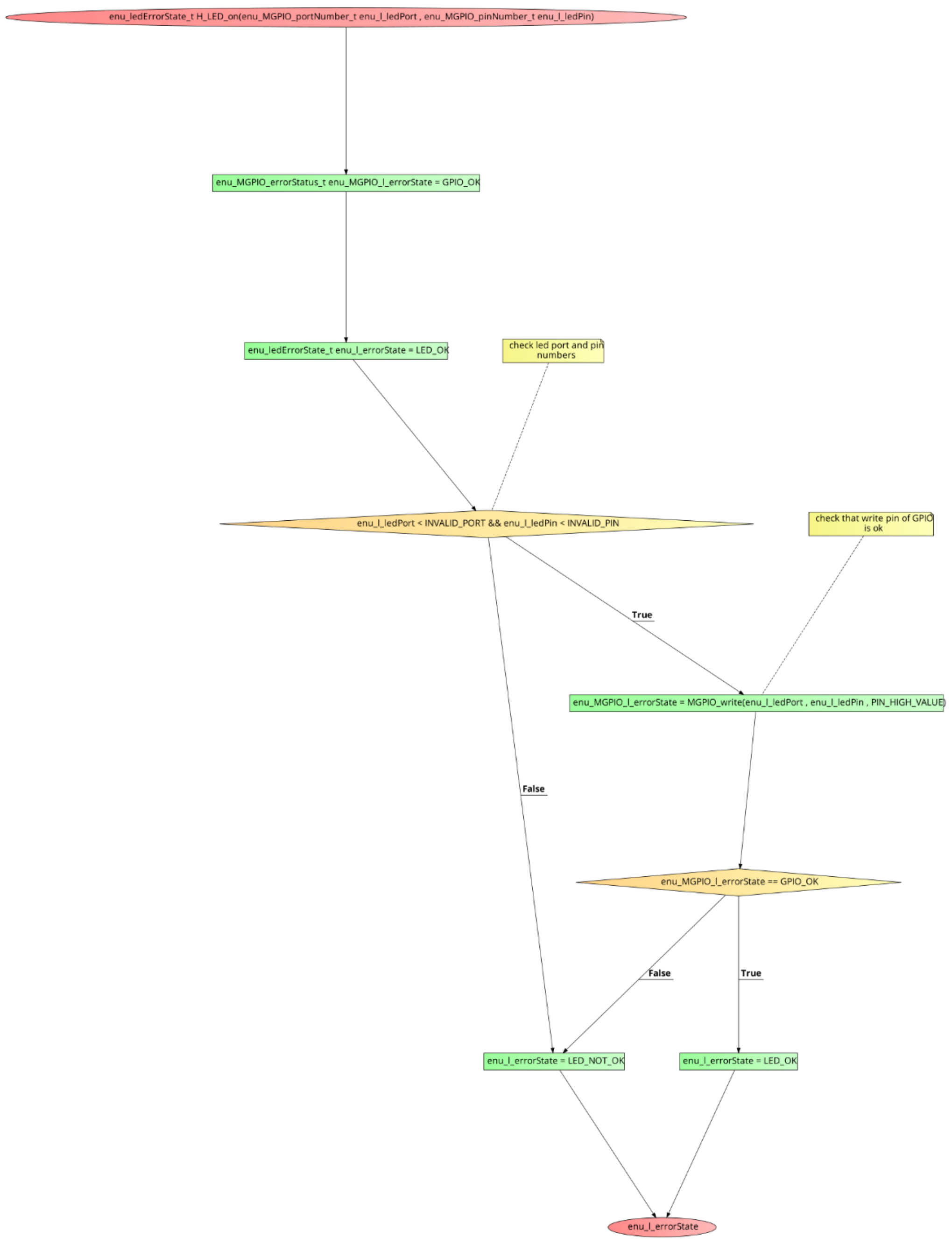

uint8_t SysTick_CheckTimeOut(void)
Check the COUNTFLAG bit in the SysTick Control and Status Register (STCTRL)
16)) != 0
(SYSTICK_CTRL & (1

➢ LED

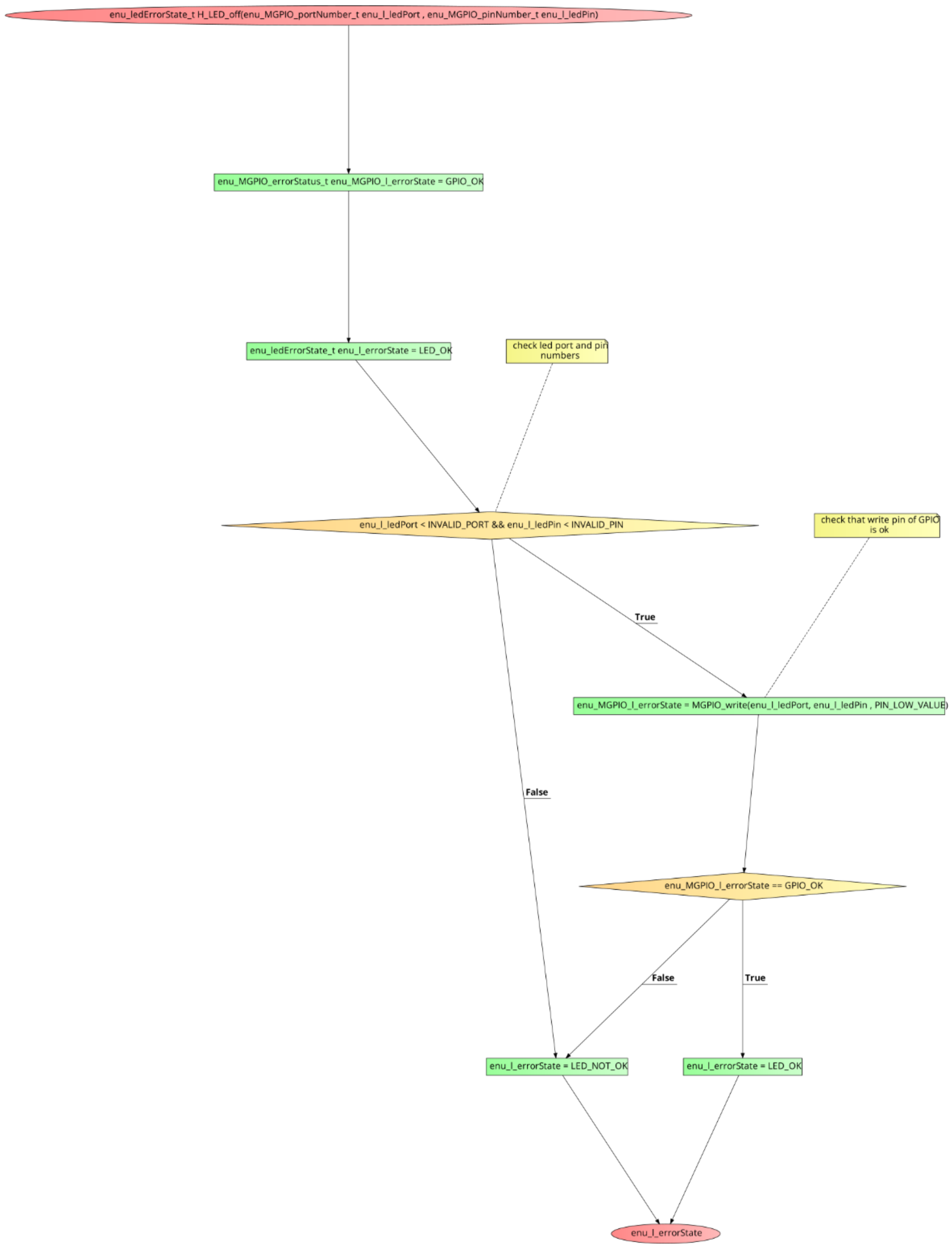
enu_ledErrorState_t H_LED_init(enu_MGPIO_portNumber_t enu_l_ledPort , enu_MGPIO_pinNumber_t enu_l_ledPin)


enu_ledErrorState_t H_LED_init(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
True
False
str_MGPIO_configuration_t str_MGPIO_ButtonConfig
configure LED struct
str_MGPIO_ButtonConfig.enu_portNumber = enu_l_ledPort
str_MGPIO_ButtonConfig.enu_pinNumber = enu_l_ledPin
str_MGPIO_ButtonConfig.enu_pinOutCurrent = PIN_CURRENT_8MA
str_MGPIO_ButtonConfig.enu_pinInternalType = PULL_UP
str_MGPIO_ButtonConfig.enu_pinDirection = OUTPUT_PIN_DIRECTION
str_MGPIO_ButtonConfig.enu_pinValue = PIN_LOW_VALUE
str_MGPIO_ButtonConfig.enu_pinMode = DIGITAL_PIN_MODE
check that initialization of GPIO is ok
enu_MGPIO_l_errorState = MGPIO_init(&str_MGPIO_ButtonConfig)
enu_MGPIO_l_errorState == GPIO_OK
True
False
enu_l_errorState = LED_OK
enu_l_errorState = LED_NOT_OK
enu_l_errorState

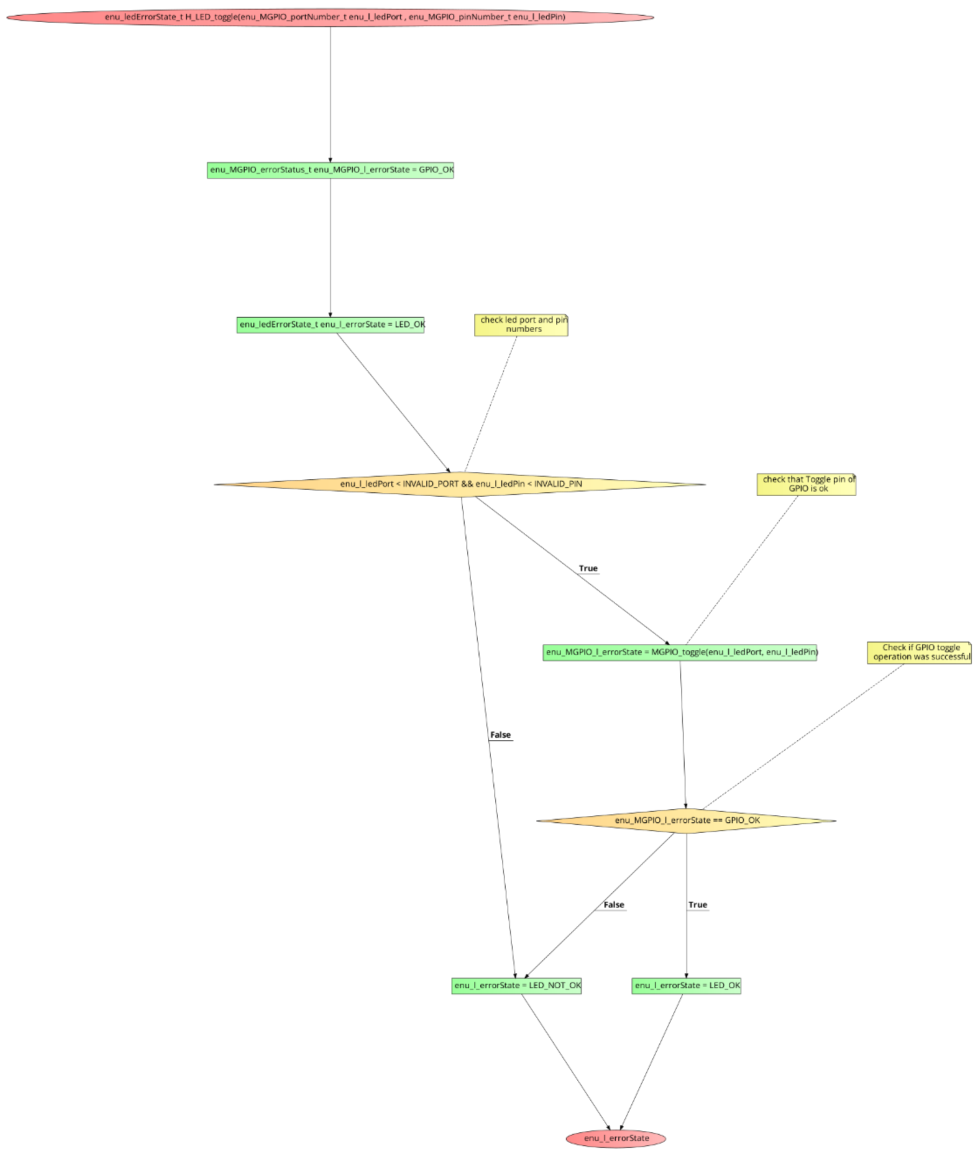
enu_ledErrorState_t H_LED_on(enu_MGPIO_portNumber_t enu_l_ledPort , enu_MGPIO_pinNumber_t enu_l_ledPin)


enu_ledErrorState_t H_LED_on(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 write pin of GPIO is ok
True
enu_MGPIO_l_errorState = MGPIO_write(enu_l_ledPort , enu_l_ledPin , PIN_HIGH_VALUE)
False
enu_MGPIO_l_errorState == GPIO_OK
False
True
enu_l_errorState = LED_NOT_OK
enu_l_errorState = LED_OK
enu_l_errorState

enu_ledErrorState_t H_LED_off(enu_MGPIO_portNumber_t enu_l_ledPort , enu_MGPIO_pinNumber_t enu_l_ledPin)

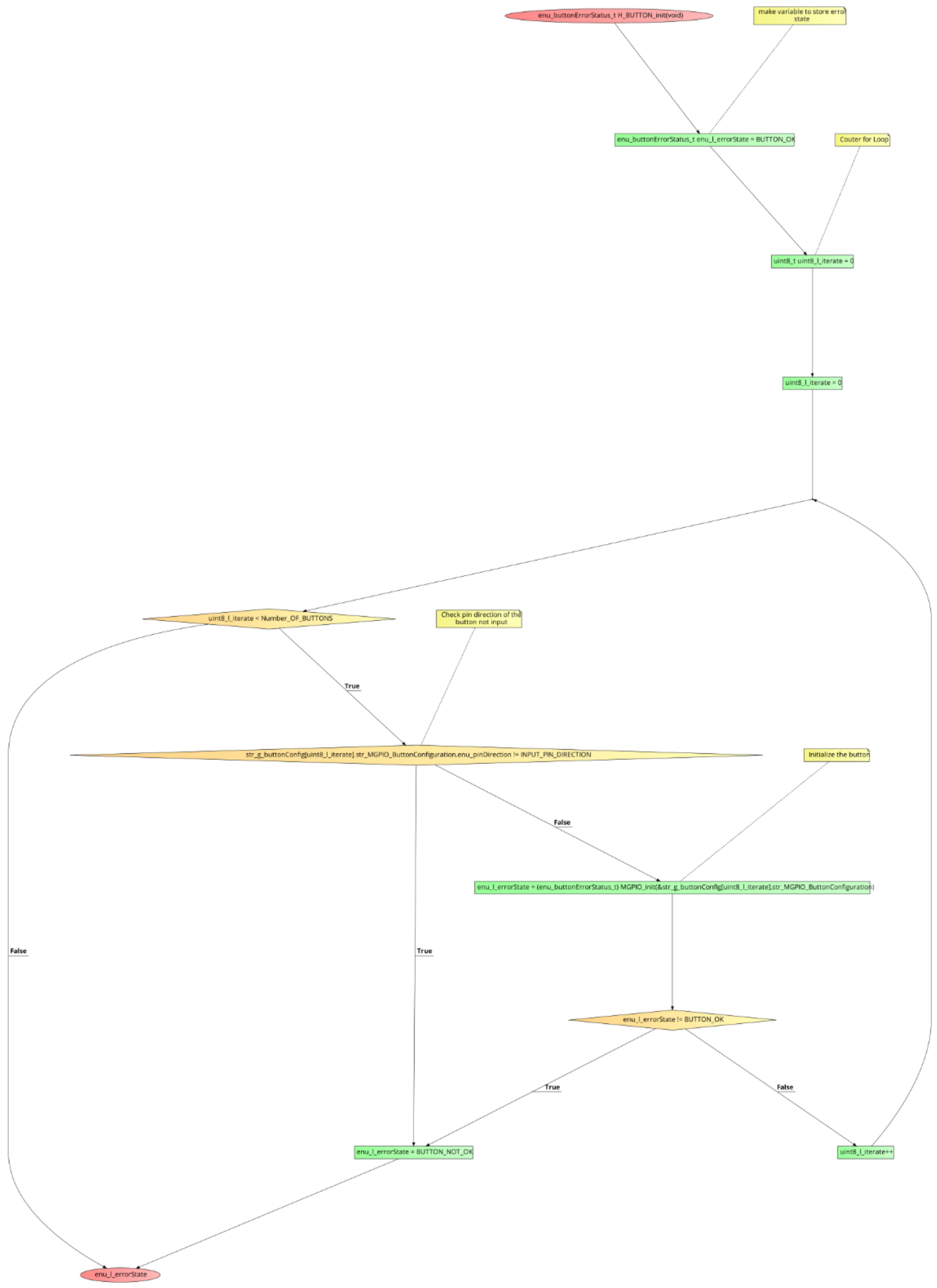

enu_ledErrorState_t H_LED_off(enu_MGPIO_portNumber_t enu_l_ledPort , enu_MGPIO_pinNumber_t enu_l_ledPin)
enu_MGPIO_errorStatus_t enu_MGPIO_l_errorState = GPIO_OK
enu_ledErrorState_t enu_l_errorState = LED_OK
check led port and pin numbers
enu_l_ledPort < INVALID_PORT && enu_l_ledPin < INVALID_PIN
check that write pin of GPIO is ok
True
enu_MGPIO_l_errorState = MGPIO_write(enu_l_ledPort, enu_l_ledPin , PIN_LOW_VALUE)
False
enu_MGPIO_l_errorState == GPIO_OK
False
True
enu_l_errorState = LED_NOT_OK
enu_l_errorState = LED_OK
enu_l_errorState

enu_ledErrorState_t H_LED_toggle(enu_MGPIO_portNumber_t enu_l_ledPort , enu_MGPIO_pinNumber_t enu_l_ledPin)

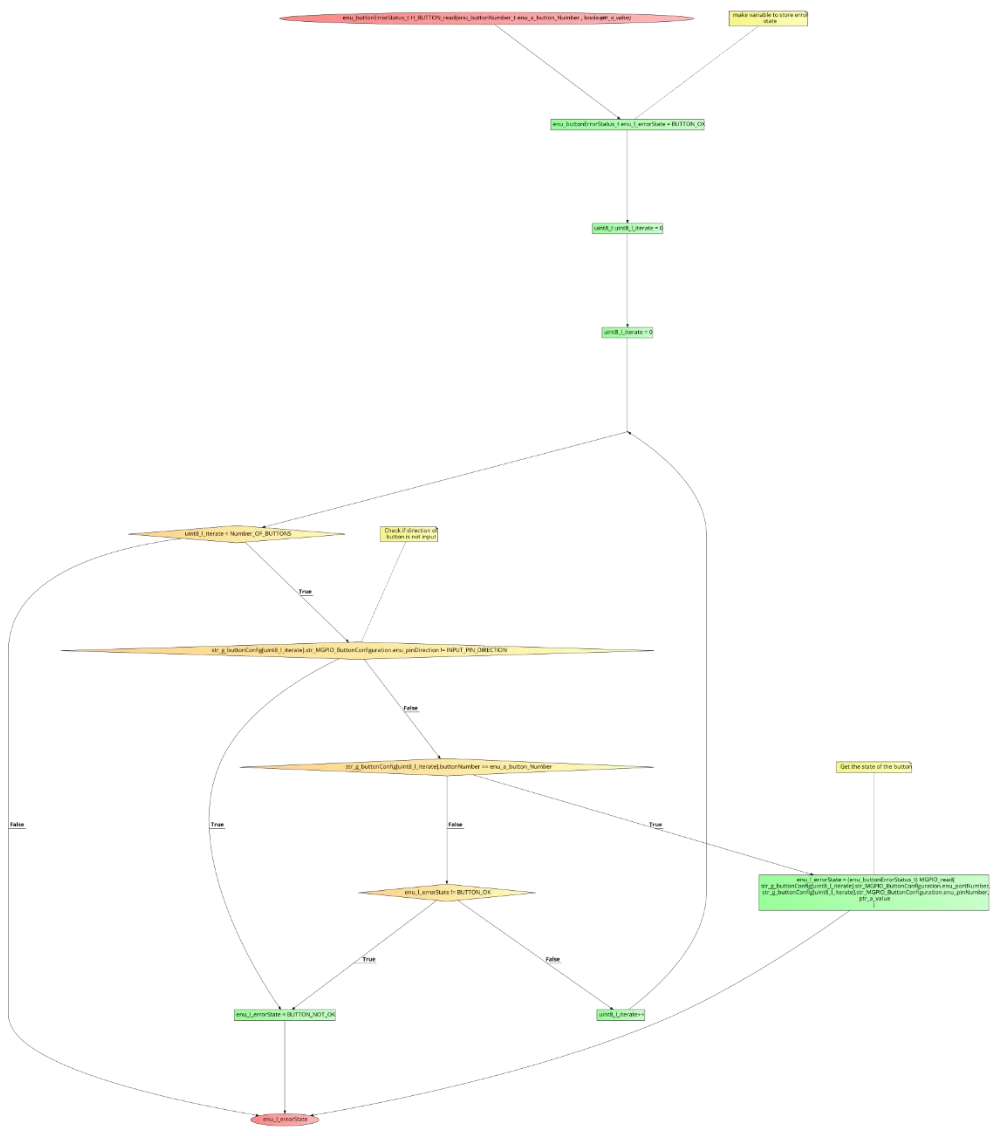

enu_ledErrorState_t H_LED_toggle(enu_MGPIO_portNumber_t enu_l_ledPort , enu_MGPIO_pinNumber_t enu_l_ledPin)
enu_MGPIO_errorStatus_t enu_MGPIO_l_errorState = GPIO_OK
enu_ledErrorState_t enu_l_errorState = LED_OK
check led port and pin numbers
enu_l_ledPort < INVALID_PORT && enu_l_ledPin < INVALID_PIN
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)

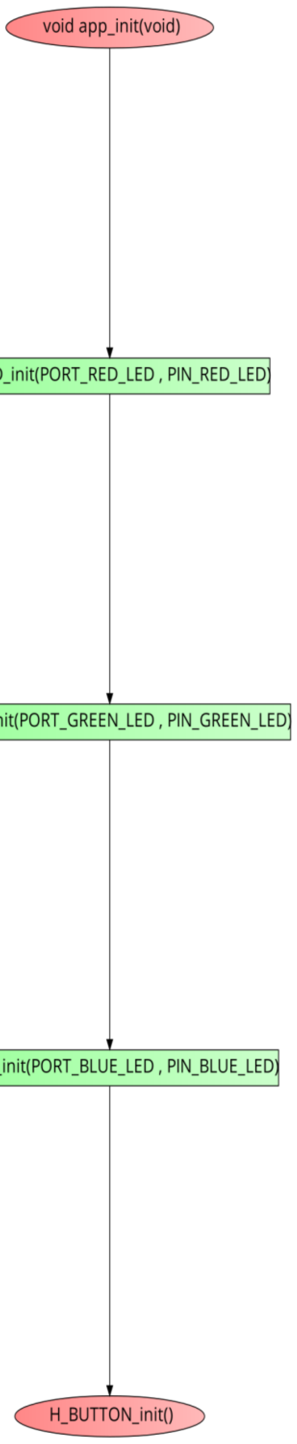

enu_buttonErrorStatus_t H_BUTTON_init(void)
make variable to store error state
enu_buttonErrorStatus_t enu_l_errorState = BUTTON_OK
Couter for Loop
uint8_t uint8_l_iterate = 0
uint8_l_iterate = 0
uint8_l_iterate < Number_OF_BUTTONS
Check pin direction of the button not input
True
str_g_buttonConfig[uint8_l_iterate].str_MGPIO_ButtonConfiguration.enu_pinDirection != INPUT_PIN_DIRECTION
Initialize the button
False
enu_l_errorState = (enu_buttonErrorStatus_t) MGPIO_init(&str_g_buttonConfig[uint8_l_iterate].str_MGPIO_ButtonConfiguration)
False
True
enu_l_errorState != BUTTON_OK
True
False
enu_l_errorState = BUTTON_NOT_OK
uint8_l_iterate++
enu_l_errorState

enu_buttonErrorStatus_t H_BUTTON_read(enu_buttonNumber_t enu_a_button_Number , boolean *ptr_a_value)

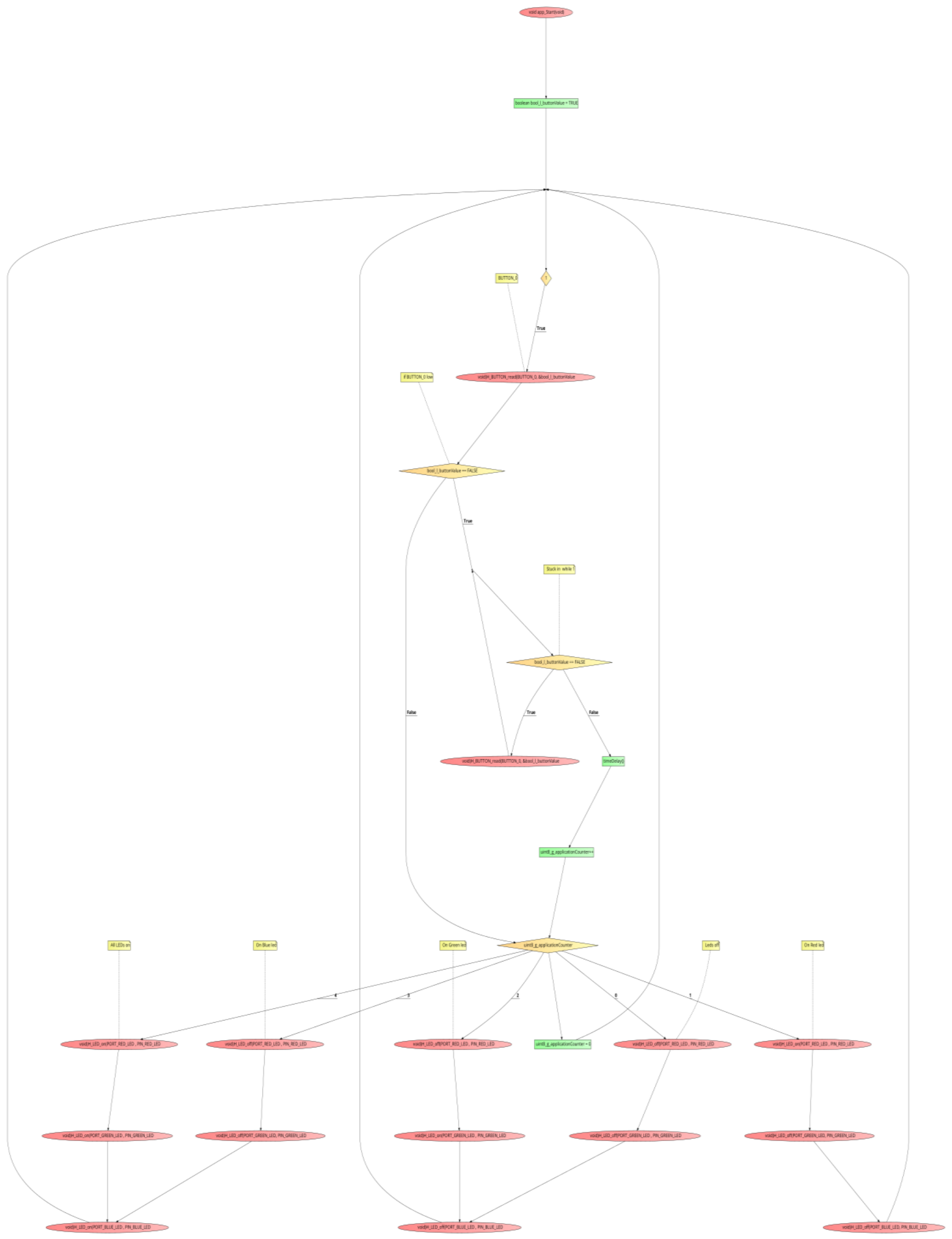

enu_buttonErrorStatus_t H_BUTTON_read(enu_buttonNumber_t enu_a_button_Number , boolean ptr_a_value)
make variable to store error state
enu_buttonErrorStatus_t enu_l_errorState = BUTTON_OK
uint8_t uint8_l_iterate = 0
uint8_l_iterate = 0
uint8_l_iterate < Number_OF_BUTTONS
Check if direction of button is not input
True
str_g_buttonConfig[uint8_l_iterate].str_MGPIO_ButtonConfiguration.enu_pinDirection != INPUT_PIN_DIRECTION
False
str_g_buttonConfig[uint8_l_iterate].buttonNumber == enu_a_button_Number
Get the state of the button
False
True
False
True
enu_l_errorState != BUTTON_OK
enu_l_errorState = (enu_buttonErrorStatus_t) MGPIO_read(str_g_buttonConfig[uint8_l_iterate].str_MGPIO_ButtonConfiguration.enu_portNumber, str_g_buttonConfig[uint8_l_iterate].str_MGPIO_ButtonConfiguration.enu_pinNumber, ptr_a_value)
True
False
enu_l_errorState = BUTTON_NOT_OK
uint8_l_iterate++
enu_l_errorState

➢ APP

void app_init(void)

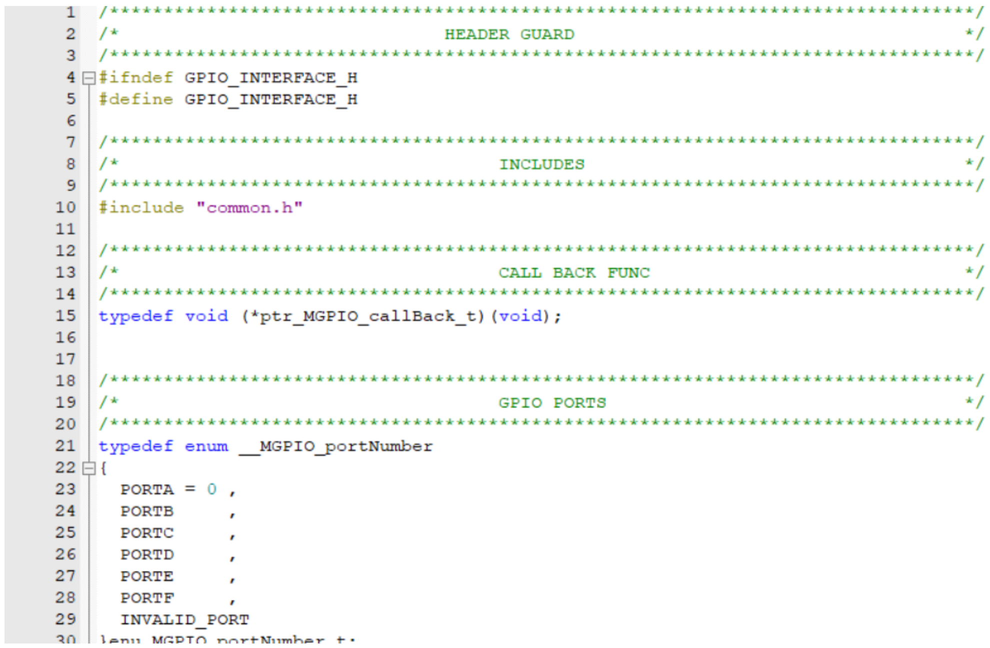

void app_init(void)
H_LED_init(PORT_RED_LED , PIN_RED_LED)
H_LED_init(PORT_GREEN_LED , PIN_GREEN_LED)
H_LED_init(PORT_BLUE_LED , PIN_BLUE_LED)
H_BUTTON_init()

void app_Start(void)

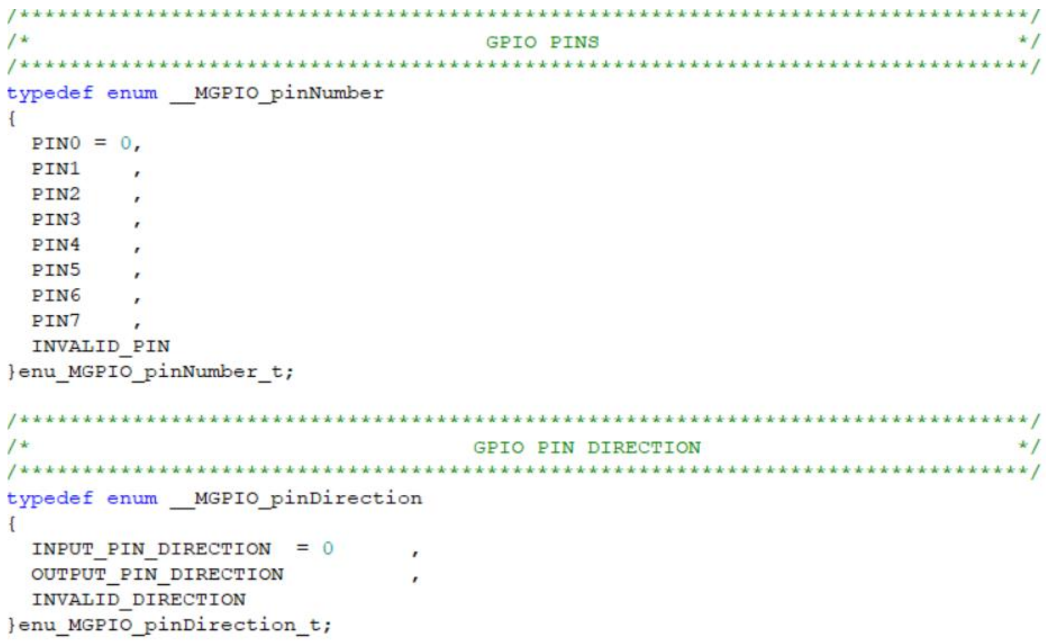

void app_Start(void)
boolean bool_l_buttonValue = TRUE
BUTTON_0
1
True
if BUTTON_0 low
void(H_BUTTON_read(BUTTON_0, &bool_l_buttonValue
bool_l_buttonValue == FALSE
True
Stuck in while 1
bool_l_buttonValue == FALSE
False
True
False
void(H_BUTTON_read(BUTTON_0, &bool_l_buttonValue
timeDelay()
uint8_g_applicationCounter++
All LEDs on
On Blue led
On Green led
uint8_g_applicationCounter
Leds off
On Red led
4
3
2
0
1
void(H_LED_on(PORT_RED_LED , PIN_RED_LED
void(H_LED_off(PORT_RED_LED , PIN_RED_LED
void(H_LED_off(PORT_RED_LED , PIN_RED_LED
uint8_g_applicationCounter = 0
void(H_LED_off(PORT_RED_LED , PIN_RED_LED
void(H_LED_on(PORT_RED_LED , PIN_RED_LED
void(H_LED_on(PORT_GREEN_LED , PIN_GREEN_LED
void(H_LED_off(PORT_GREEN_LED, PIN_GREEN_LED
void(H_LED_on(PORT_GREEN_LED , PIN_GREEN_LED
void(H_LED_off(PORT_GREEN_LED , PIN_GREEN_LED
void(H_LED_off(PORT_GREEN_LED, PIN_GREEN_LED
void(H_LED_on(PORT_BLUE_LED , PIN_BLUE_LED
void(H_LED_off(PORT_BLUE_LED , PIN_BLUE_LED
void(H_LED_off(PORT_BLUE_LED, PIN_BLUE_LED

- ## Precompiling & Linking Configurations

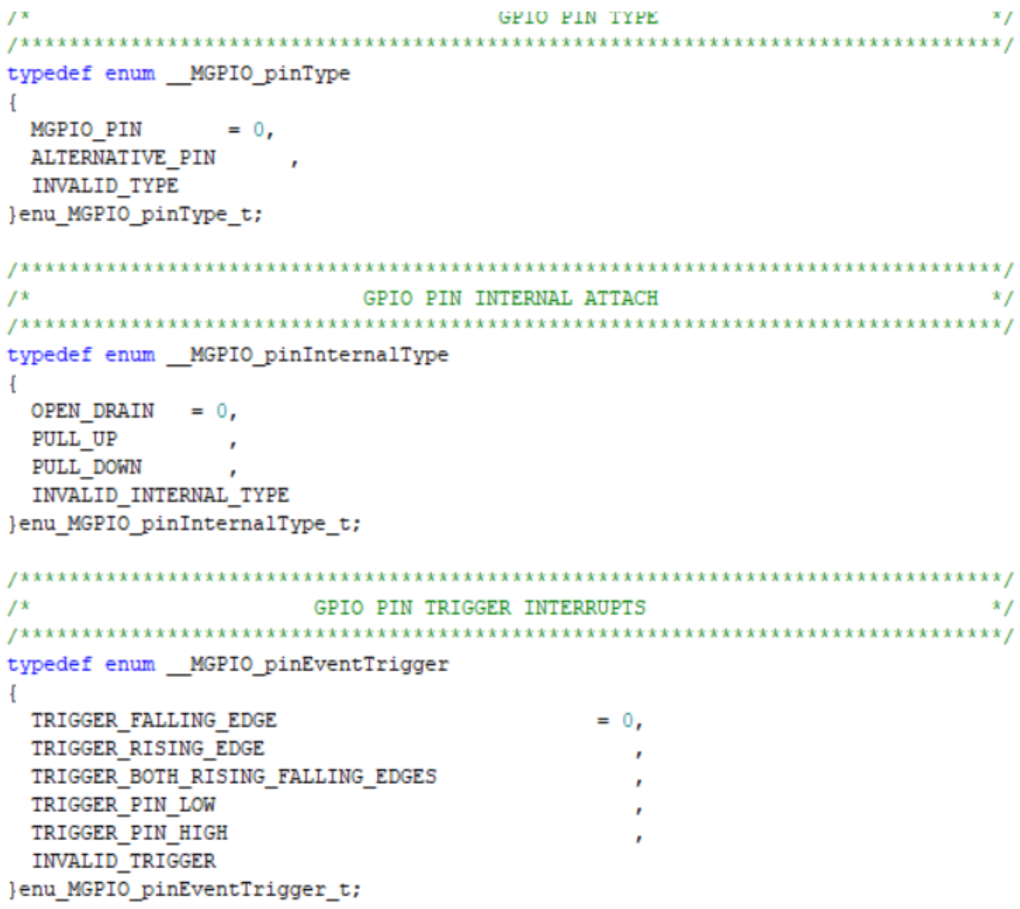
➢ GPIO



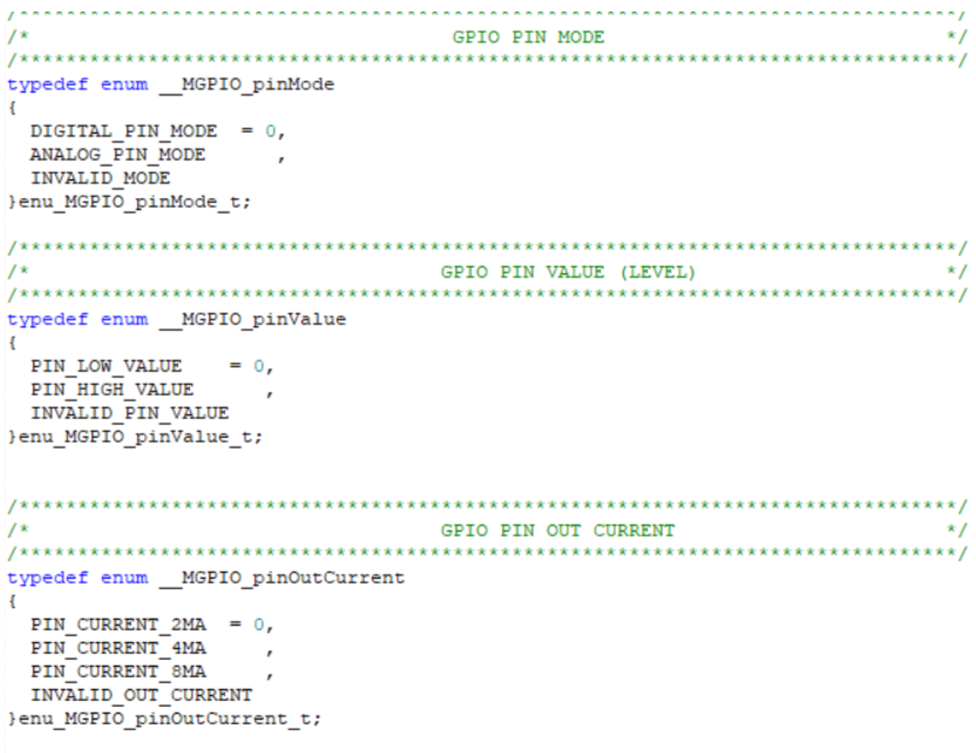
```c
/*******************************************************************************/
/*                                HEADER GUARD                                 */
/*******************************************************************************/
#ifndef GPIO_INTERFACE_H
#define GPIO_INTERFACE_H

/*******************************************************************************/
/*                                  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;
```



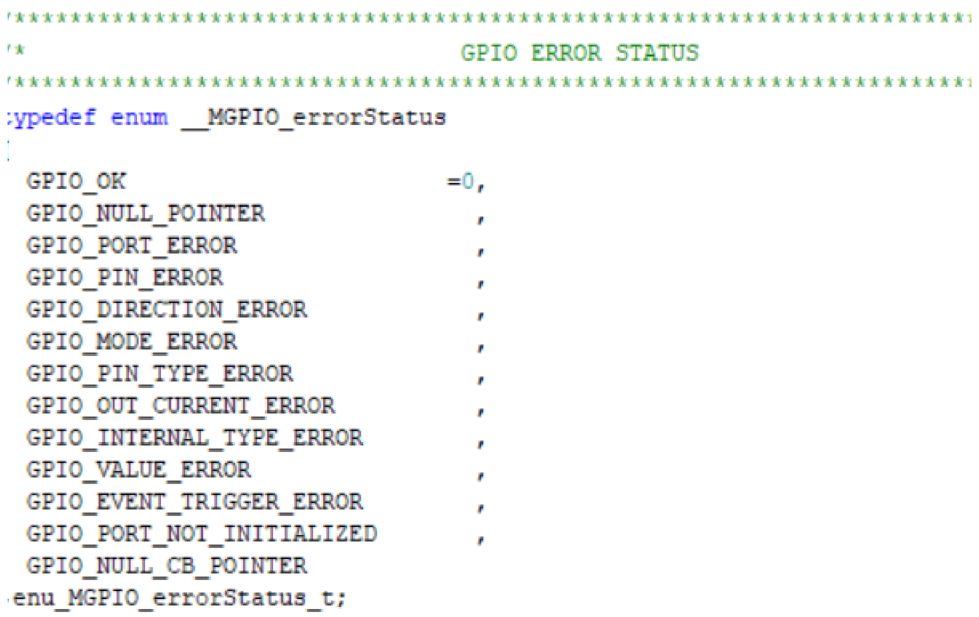
```c
/*******************************************************************************/
/*                                 GPIO PINS                                   */
/*******************************************************************************/
typedef enum __MGPIO_pinNumber
{
  PIN0 = 0,
  PIN1    ,
  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;
```



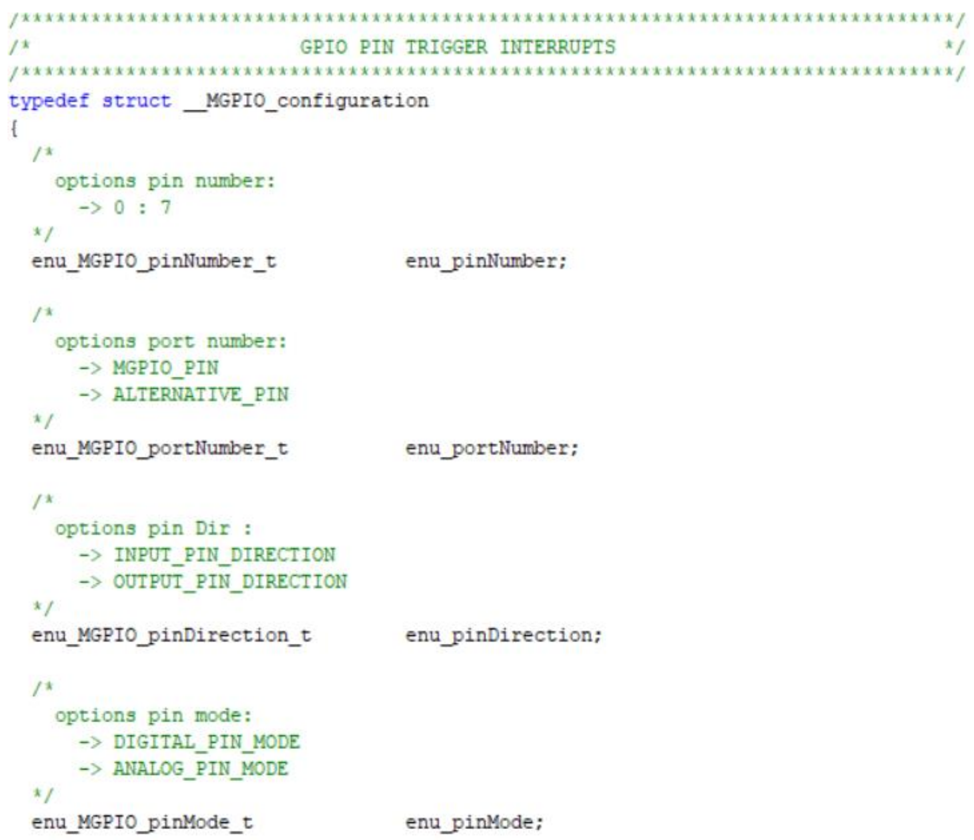
```
/*                                GPIO PIN TYPE                                */
/*****************************************************************************/
typedef enum __MGPIO_pinType
{
  MGPIO_PIN       = 0,
  ALTERNATIVE_PIN      ,
  INVALID_TYPE
}enu_MGPIO_pinType_t;

/*****************************************************************************/
/*                         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;
```



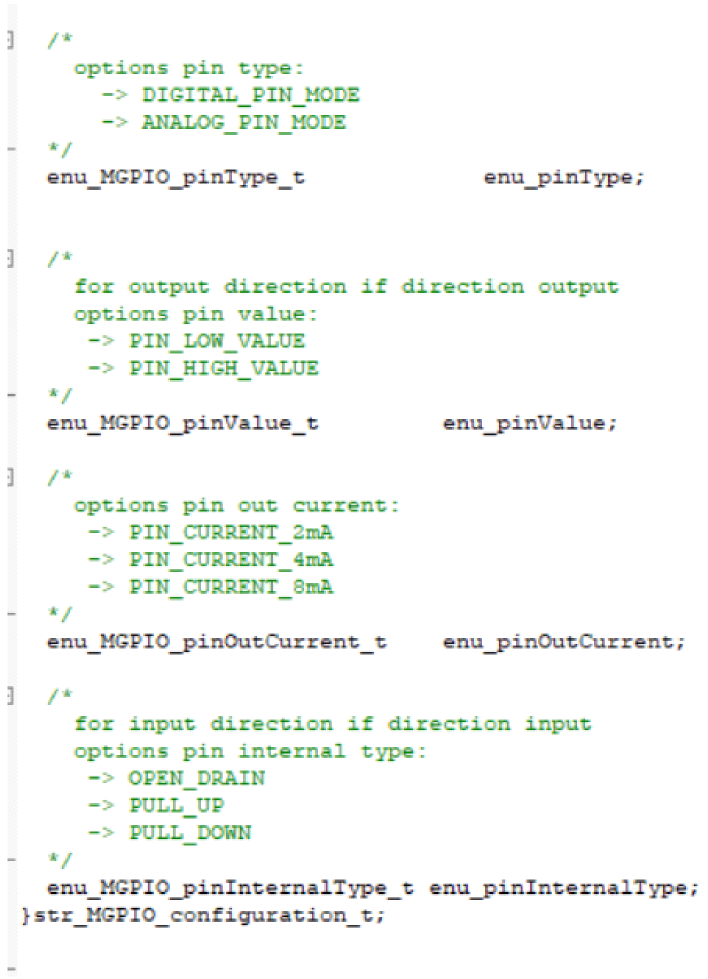
```
/*****************************************************************************/
/*                               GPIO PIN MODE                               */
/*****************************************************************************/
typedef enum __MGPIO_pinMode
{
  DIGITAL_PIN_MODE  = 0,
  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
************************************************************************
ypedef 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;
```

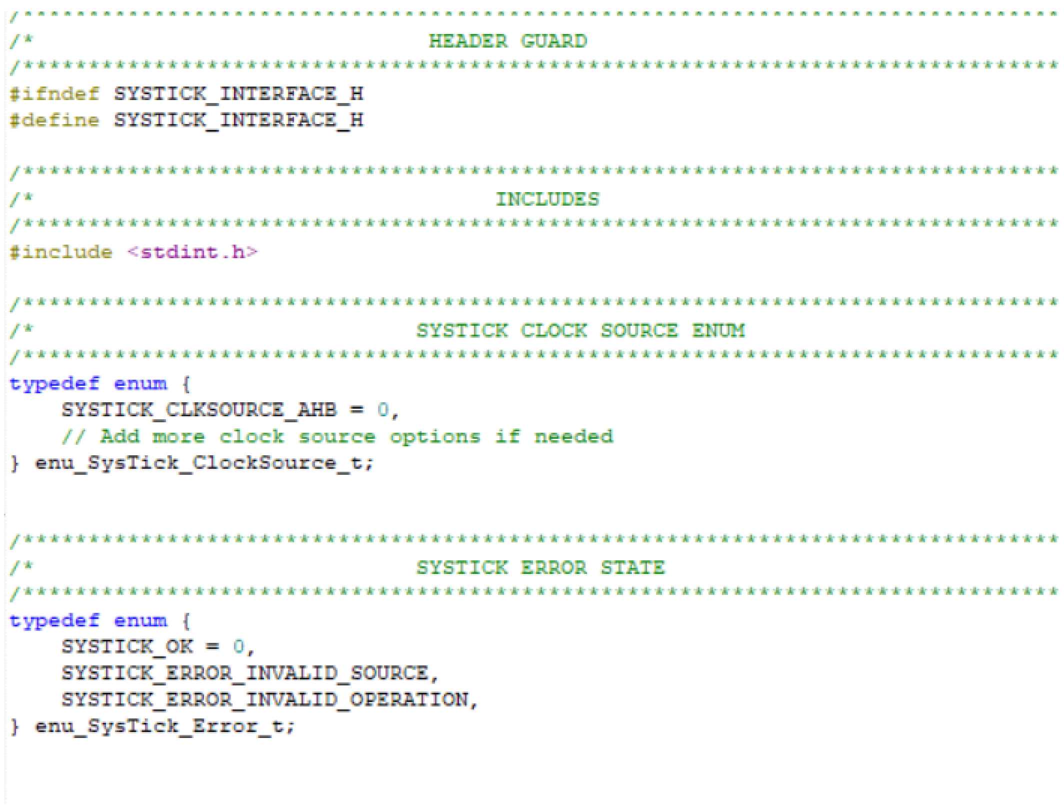


```
/*******************************************************************************/
/*                         GPIO PIN TRIGGER INTERRUPTS                          */
/*******************************************************************************/
typedef struct __MGPIO_configuration
{
  /*
    options 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;
```

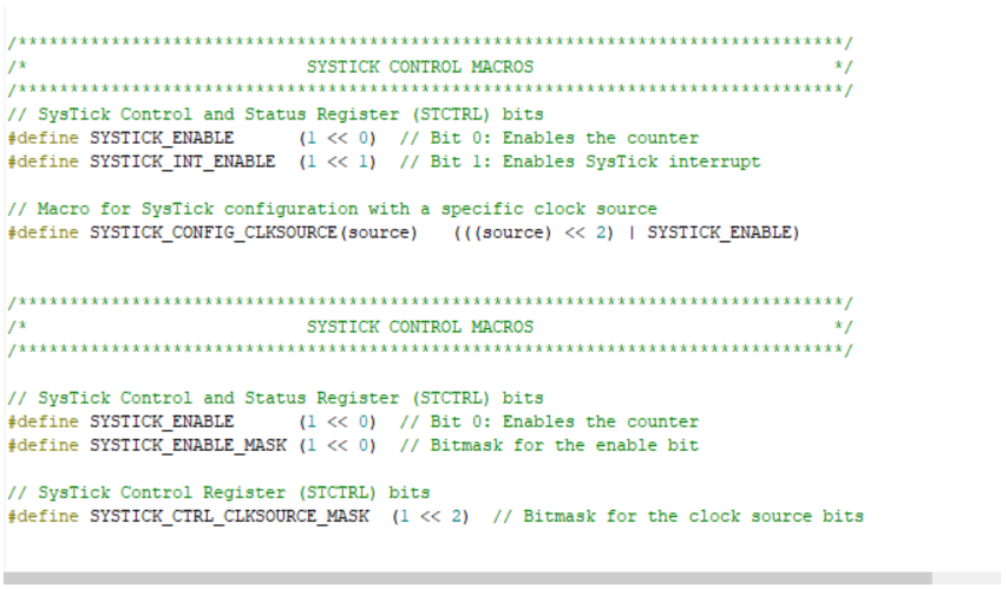


```
  /*
    options pin type:
      -> DIGITAL_PIN_MODE
      -> ANALOG_PIN_MODE
  */
  enu_MGPIO_pinType_t               enu_pinType;


  /*
    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;
```

➢ SYSTICK

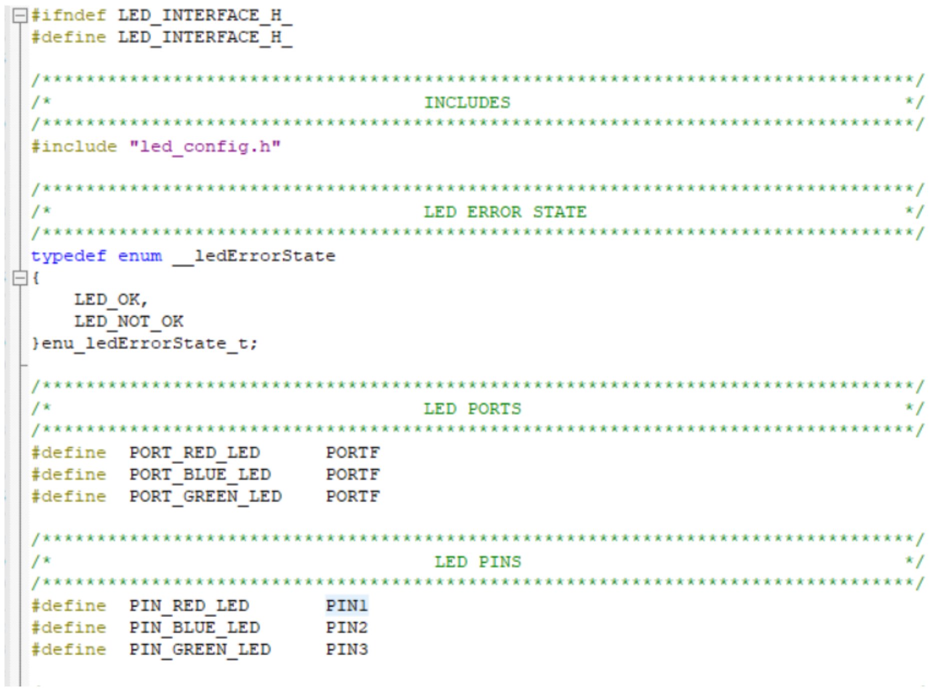


```c
/*****************************************************************************
/*                                   HEADER GUARD
/*****************************************************************************
#ifndef SYSTICK_INTERFACE_H
#define SYSTICK_INTERFACE_H

/*****************************************************************************
/*                                        INCLUDES
/*****************************************************************************
#include <stdint.h>

/*****************************************************************************
/*                              SYSTICK CLOCK SOURCE ENUM
/*****************************************************************************
typedef enum {
    SYSTICK_CLKSOURCE_AHB = 0,
    // Add more clock source options if needed
} enu_SysTick_ClockSource_t;


/*****************************************************************************
/*                              SYSTICK ERROR STATE
/*****************************************************************************
typedef enum {
    SYSTICK_OK = 0,
    SYSTICK_ERROR_INVALID_SOURCE,
    SYSTICK_ERROR_INVALID_OPERATION,
} enu_SysTick_Error_t;
```

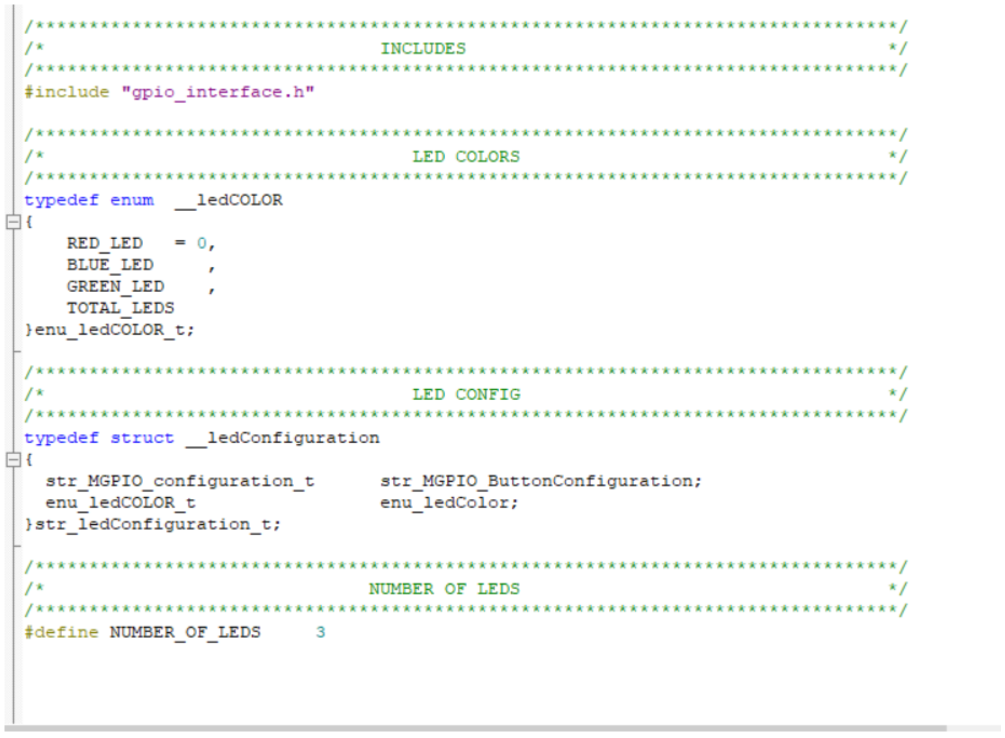


```c
/*****************************************************************************/
/*                         SYSTICK CONTROL MACROS                            */
/*****************************************************************************/
// SysTick Control and Status Register (STCTRL) bits
#define SYSTICK_ENABLE      (1 << 0)  // Bit 0: Enables the counter
#define SYSTICK_INT_ENABLE  (1 << 1)  // Bit 1: Enables SysTick interrupt

// Macro for SysTick configuration with a specific clock source
#define SYSTICK_CONFIG_CLKSOURCE(source)   (((source) << 2) | SYSTICK_ENABLE)


/*****************************************************************************/
/*                         SYSTICK CONTROL MACROS                            */
/*****************************************************************************/

// SysTick Control and Status Register (STCTRL) bits
#define SYSTICK_ENABLE      (1 << 0)  // Bit 0: Enables the counter
#define SYSTICK_ENABLE_MASK (1 << 0)  // Bitmask for the enable bit

// SysTick Control Register (STCTRL) bits
#define SYSTICK_CTRL_CLKSOURCE_MASK  (1 << 2)  // Bitmask for the clock source bits
```

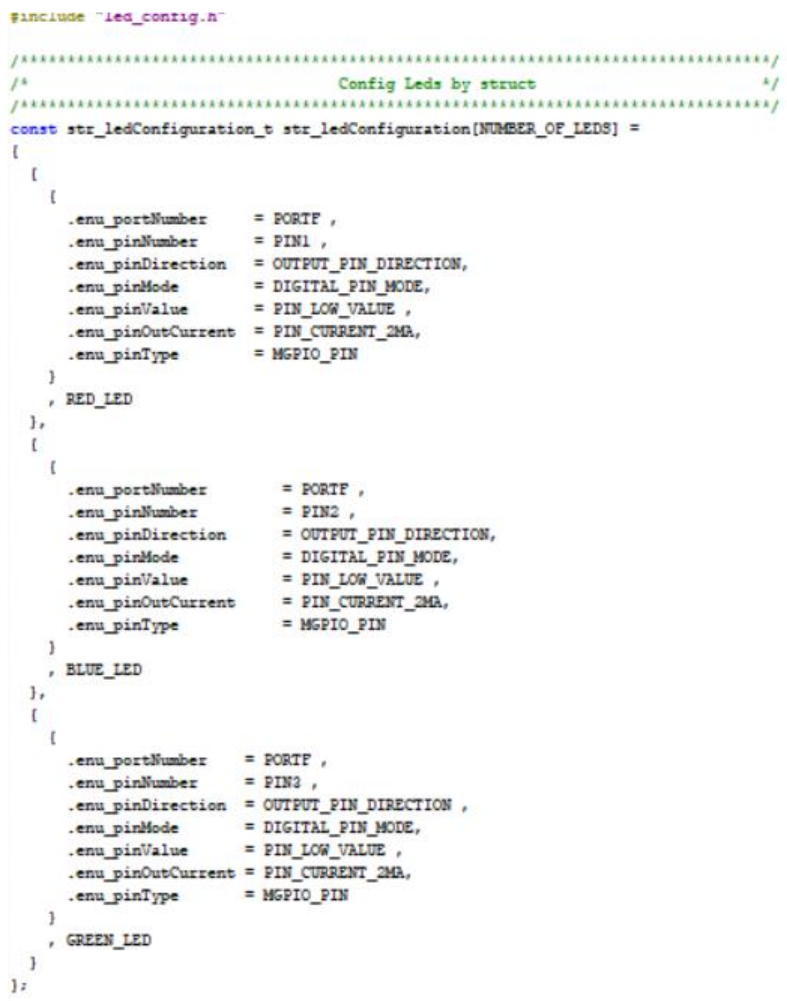
➢ LED



```
#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
```

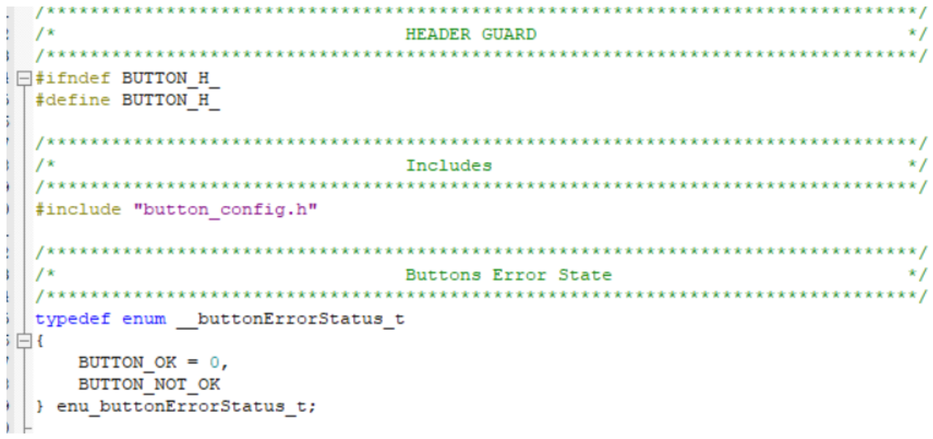


```
/*******************************************************************************/
/*                                   INCLUDES                                  */
/*******************************************************************************/
#include "gpio_interface.h"

/*******************************************************************************/
/*                                  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
```

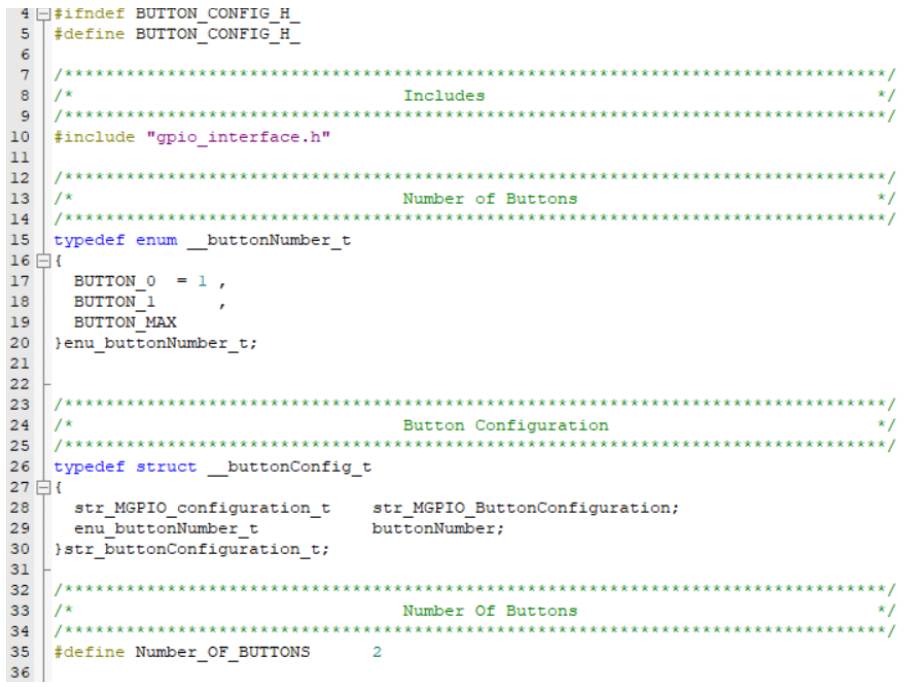


```
#include "led_config.h"

/*********************************************************************************/
/*                              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
  }
};
```

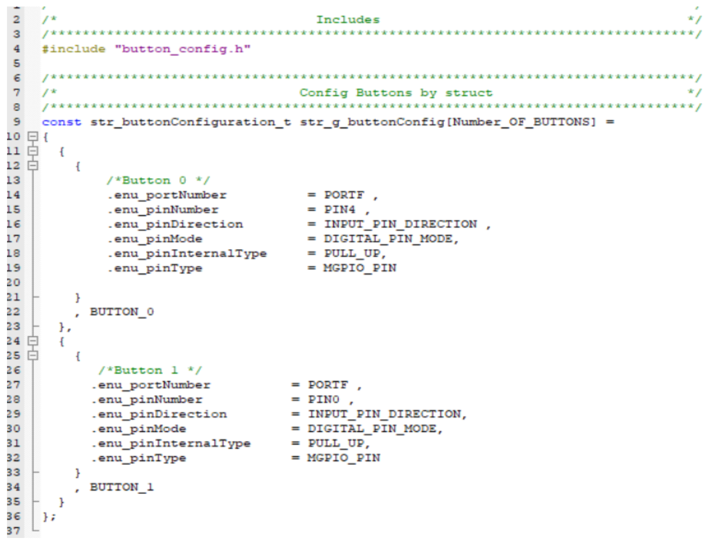
➢ BUTTON



```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