

**RGB LED Brightness Control V1.0 Design**

Peter Essam | ARM | 26/11/2023

**Table of Content**

1. Introduction
2. High Level Design

* Layered Architecture
* Modules Descriptions
* Drivers Documentations
* GPIO
* GPT(Timer)
* TIMER MANAGER
* LED
* BUTTON

1. Low Level Design

* Flow Chart
* Precompiling Configurations
* Linking Configurations

**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

**COMMON**

**APP**

**SERVICE (Timer Manager)**

**HAL (led, Button)**

**MCAL (DIO , GPT)**

* 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**

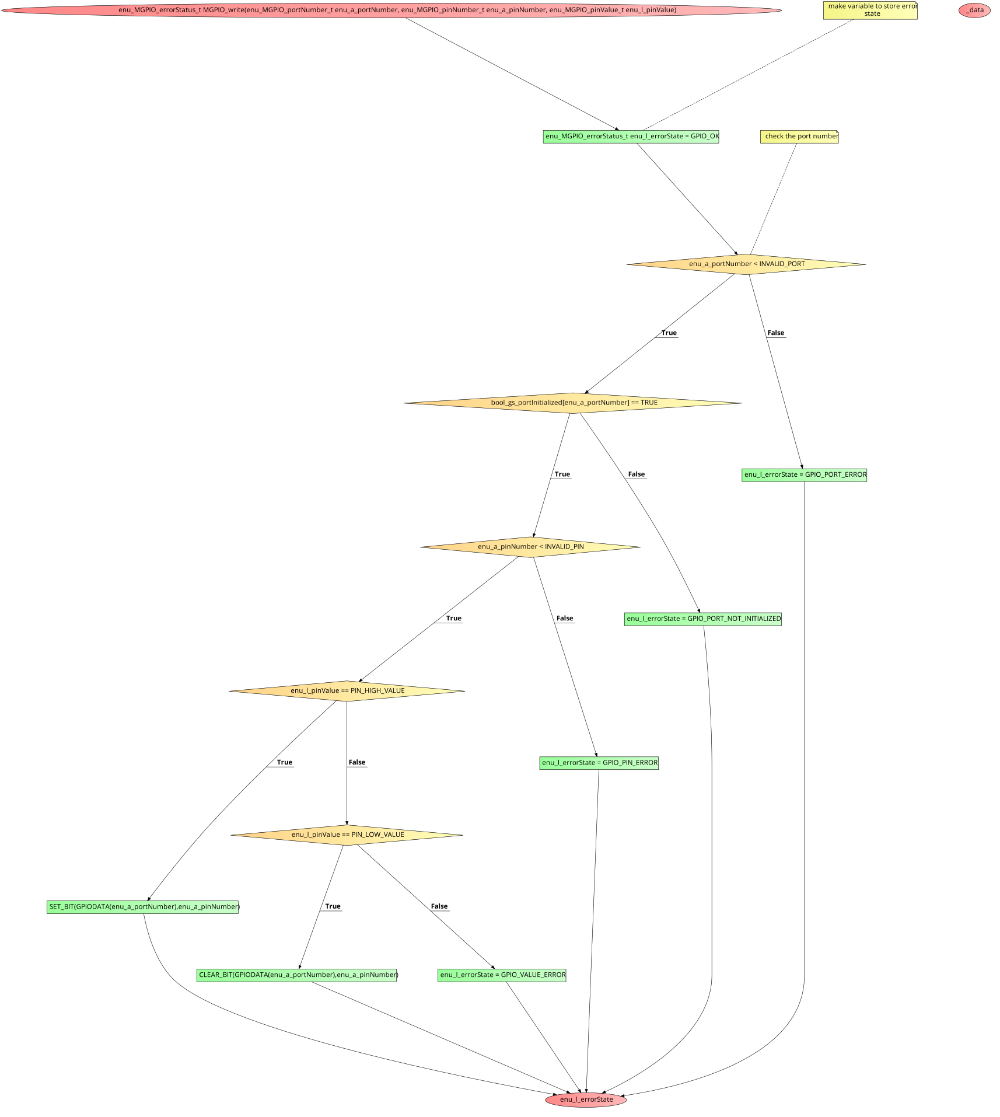
**Return**

**enu\_timerHandlerErrorStatus\_t Error status after setting PWM configuration in the timer handler.**

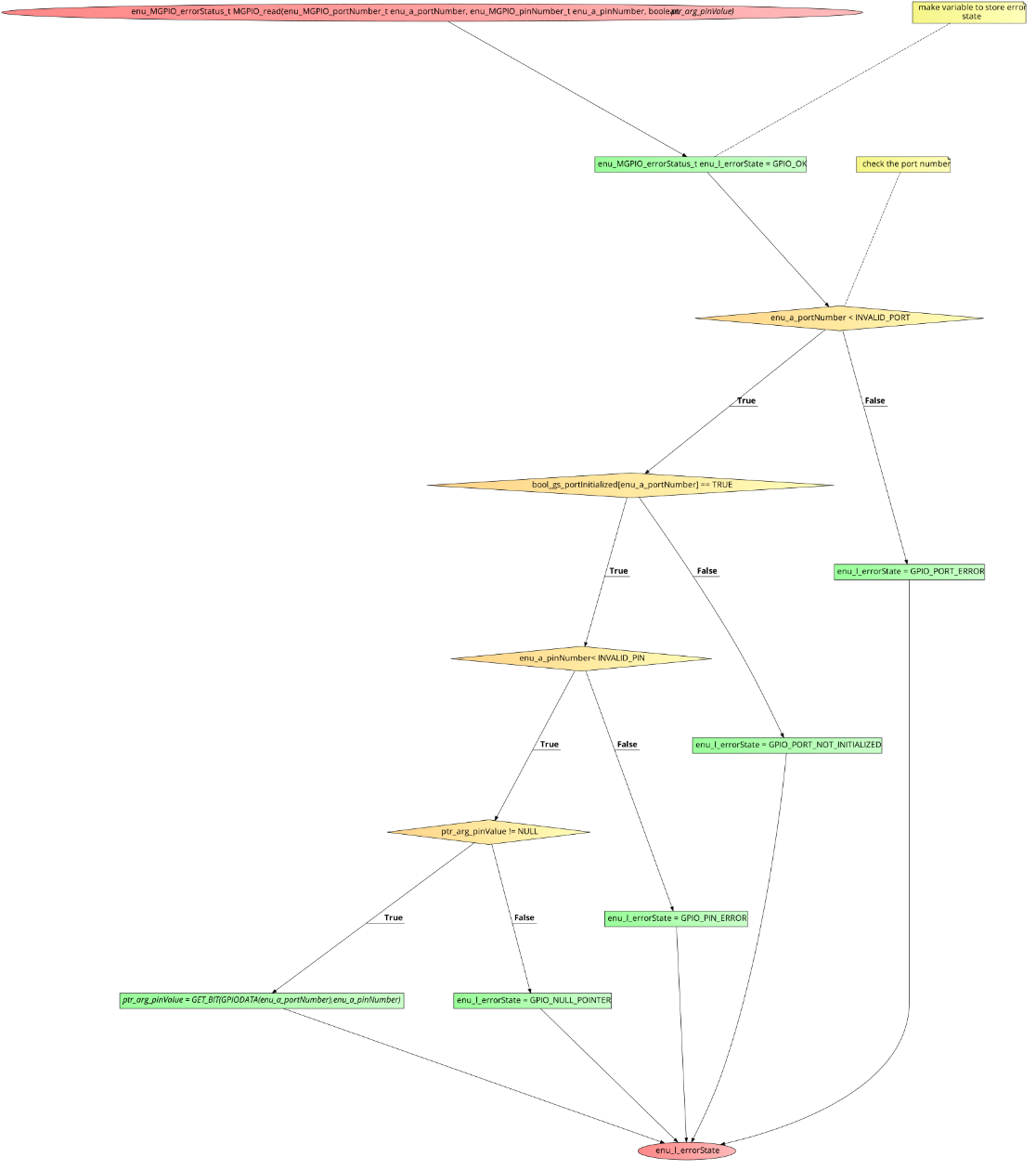
**Low Level Design**

* Flow Chart
* GPIO

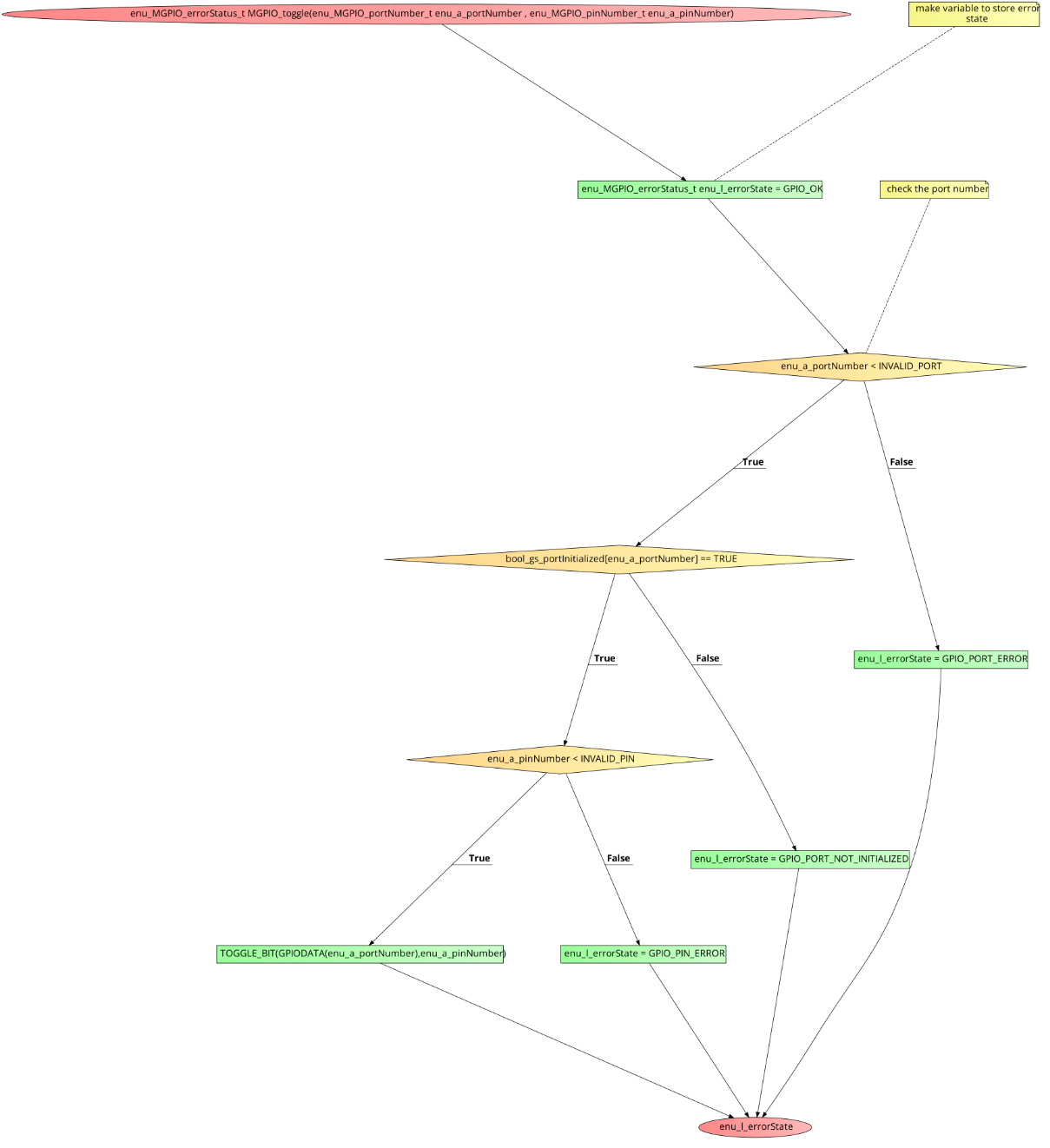
enu\_MGPIO\_errorStatus\_t MGPIO\_write(enu\_MGPIO\_portNumber\_t enu\_a\_portNumber, enu\_MGPIO\_pinNumber\_t enu\_a\_pinNumber, enu\_MGPIO\_pinValue\_t enu\_l\_pinValue)



enu\_MGPIO\_errorStatus\_t MGPIO\_read(enu\_MGPIO\_portNumber\_t enu\_a\_portNumber, enu\_MGPIO\_pinNumber\_t enu\_a\_pinNumber, boolean \*ptr\_arg\_pinValue)

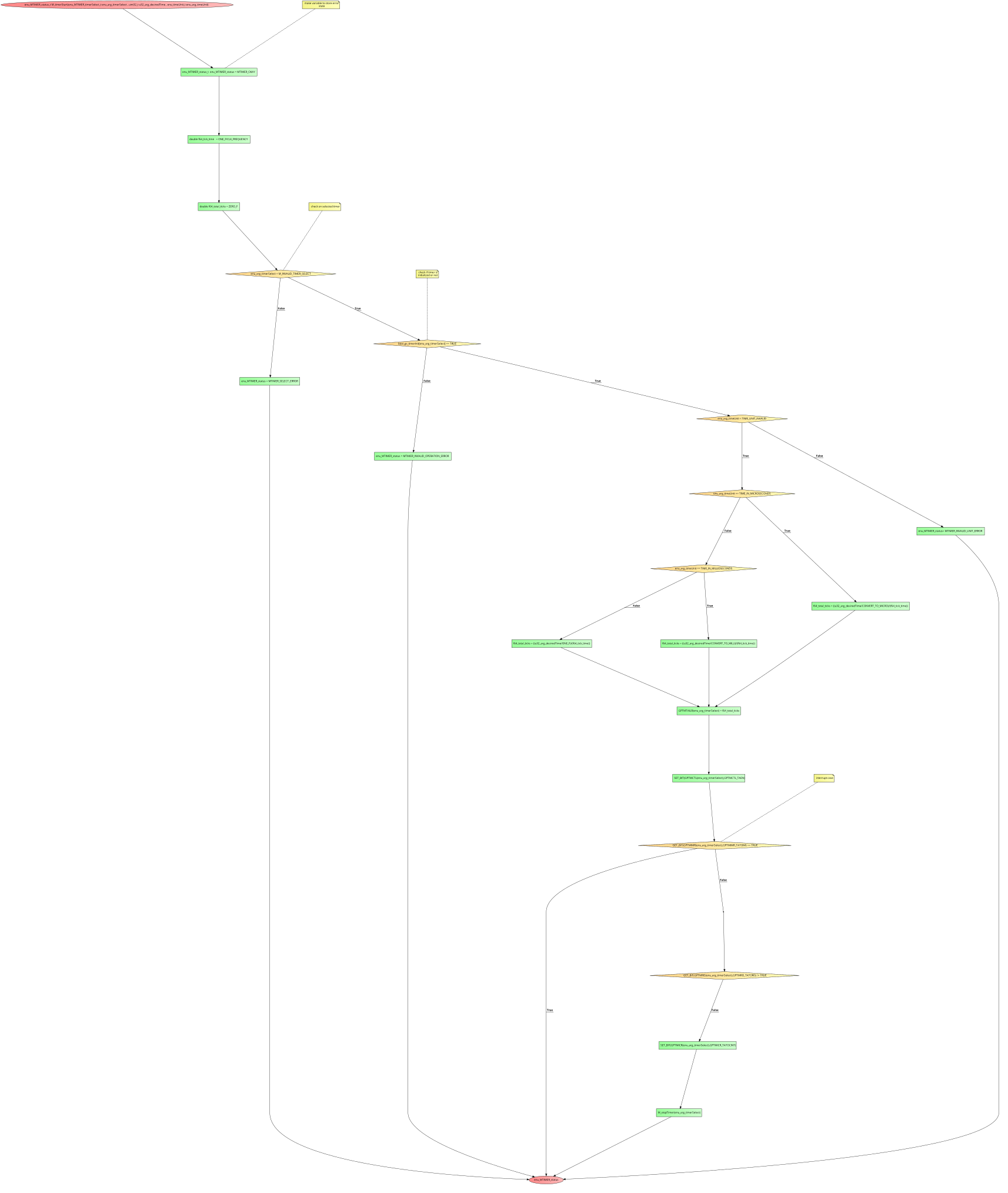


enu\_MGPIO\_errorStatus\_t MGPIO\_toggle(enu\_MGPIO\_portNumber\_t enu\_a\_portNumber , enu\_MGPIO\_pinNumber\_t enu\_a\_pinNumber)

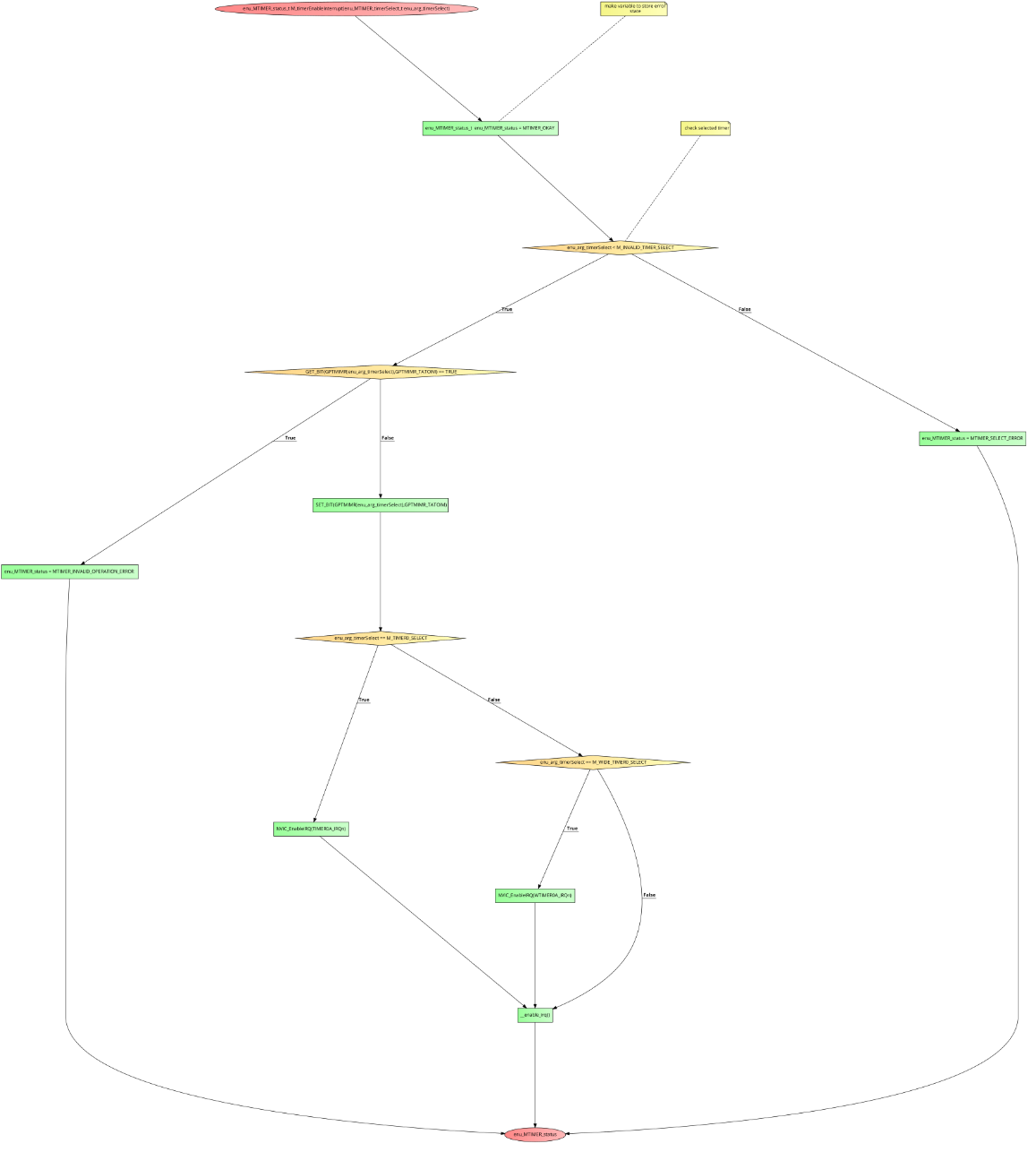


* GPT(Timer)

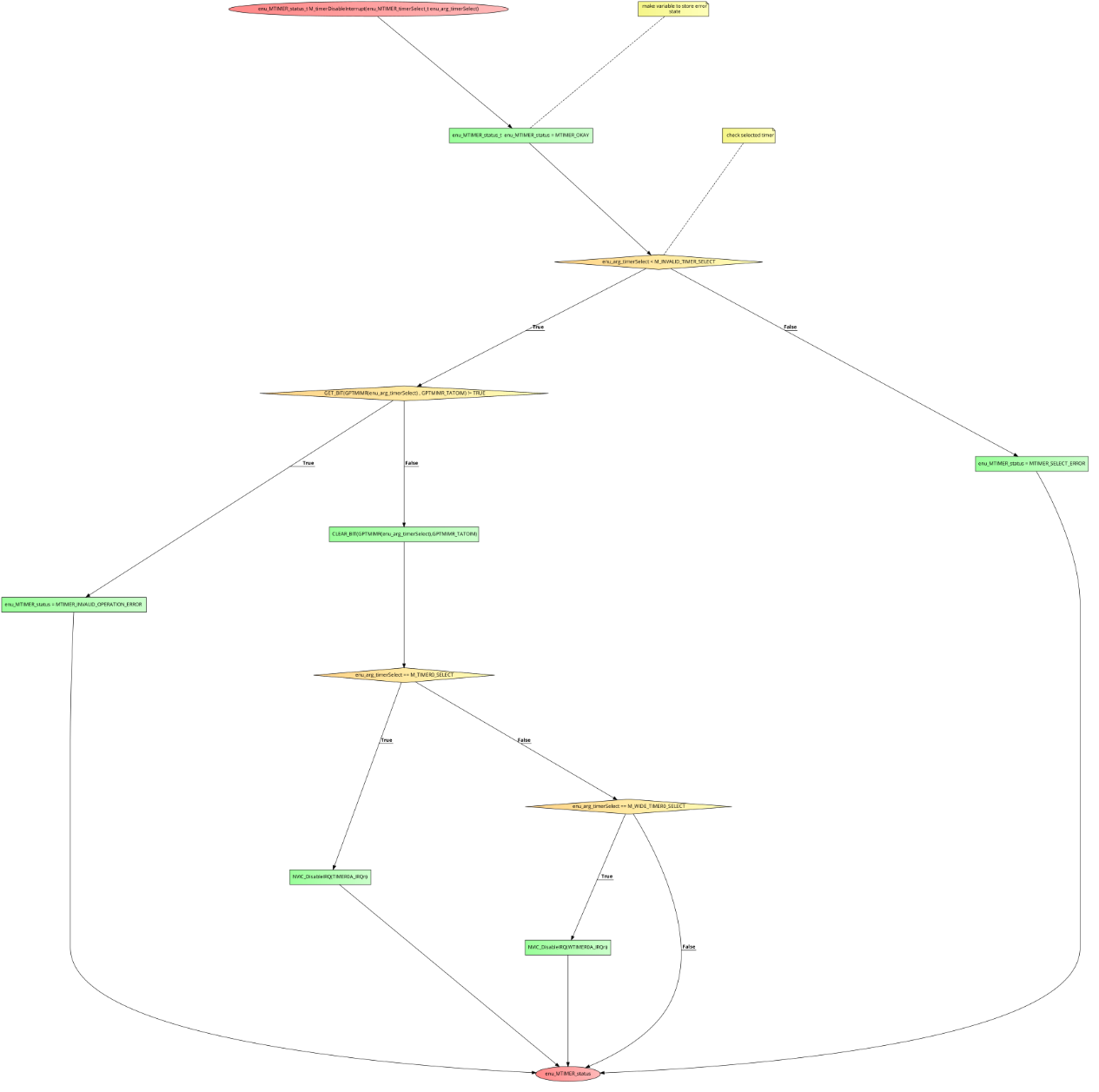
enu\_MTIMER\_status\_t M\_timerStart(enu\_MTIMER\_timerSelect\_t enu\_arg\_timerSelect , uint32\_t u32\_arg\_desiredTime , enu\_timeUnit\_t enu\_arg\_timeUnit)



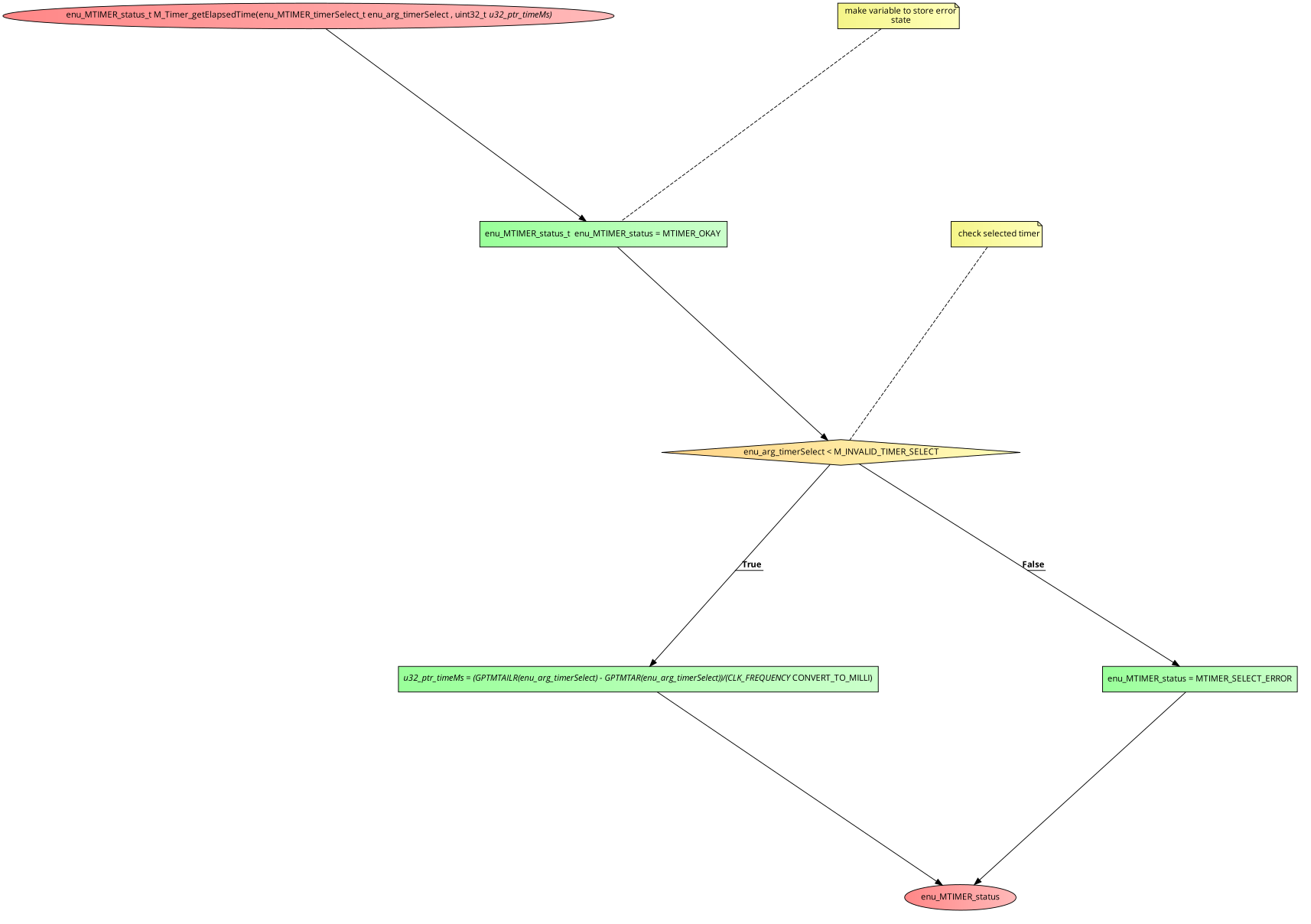
enu\_MTIMER\_status\_t M\_timerEnableInterrupt(enu\_MTIMER\_timerSelect\_t enu\_arg\_timerSelec



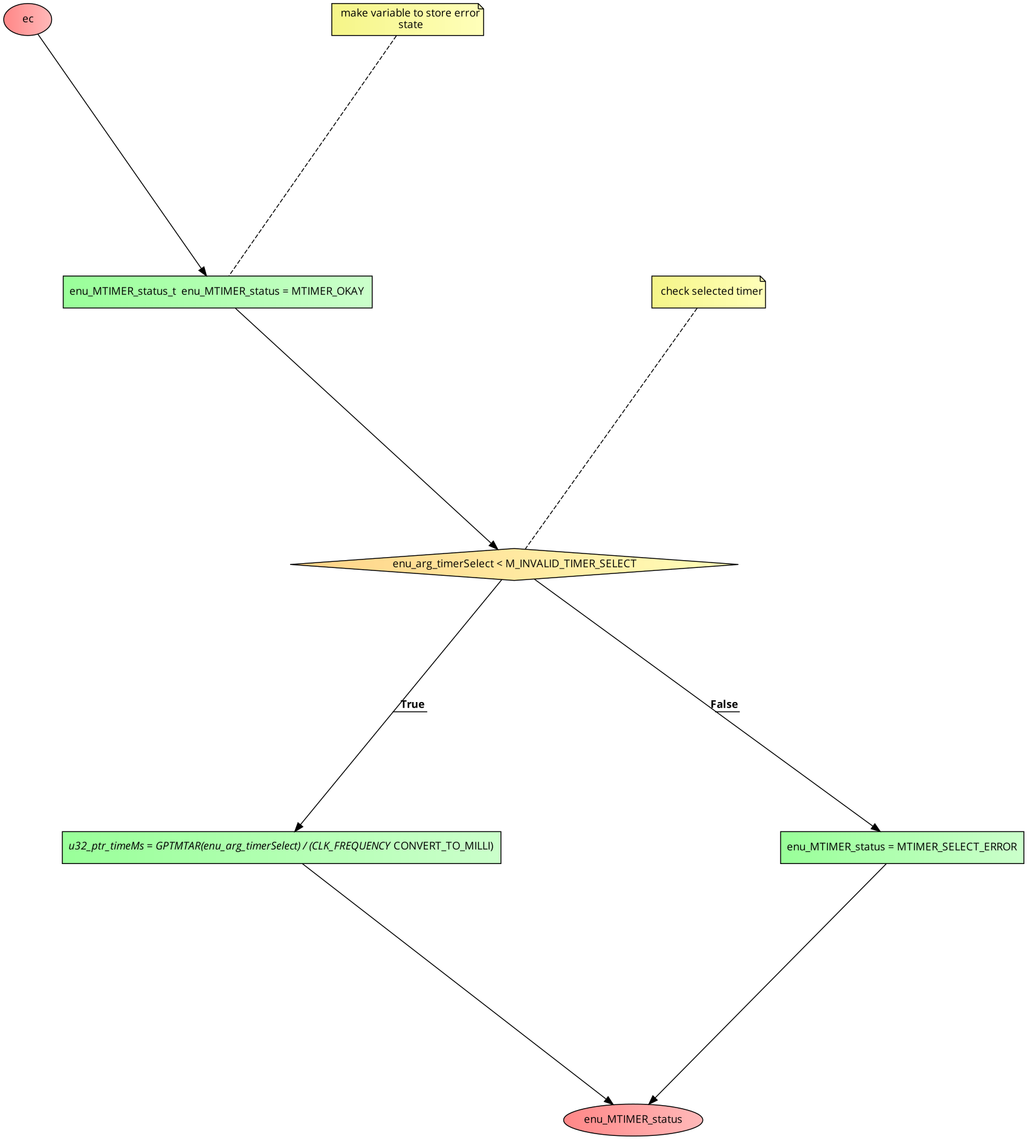
enu\_MTIMER\_status\_t M\_timerDisableInterrupt(enu\_MTIMER\_timerSelect\_t enu\_arg\_timerSelect)



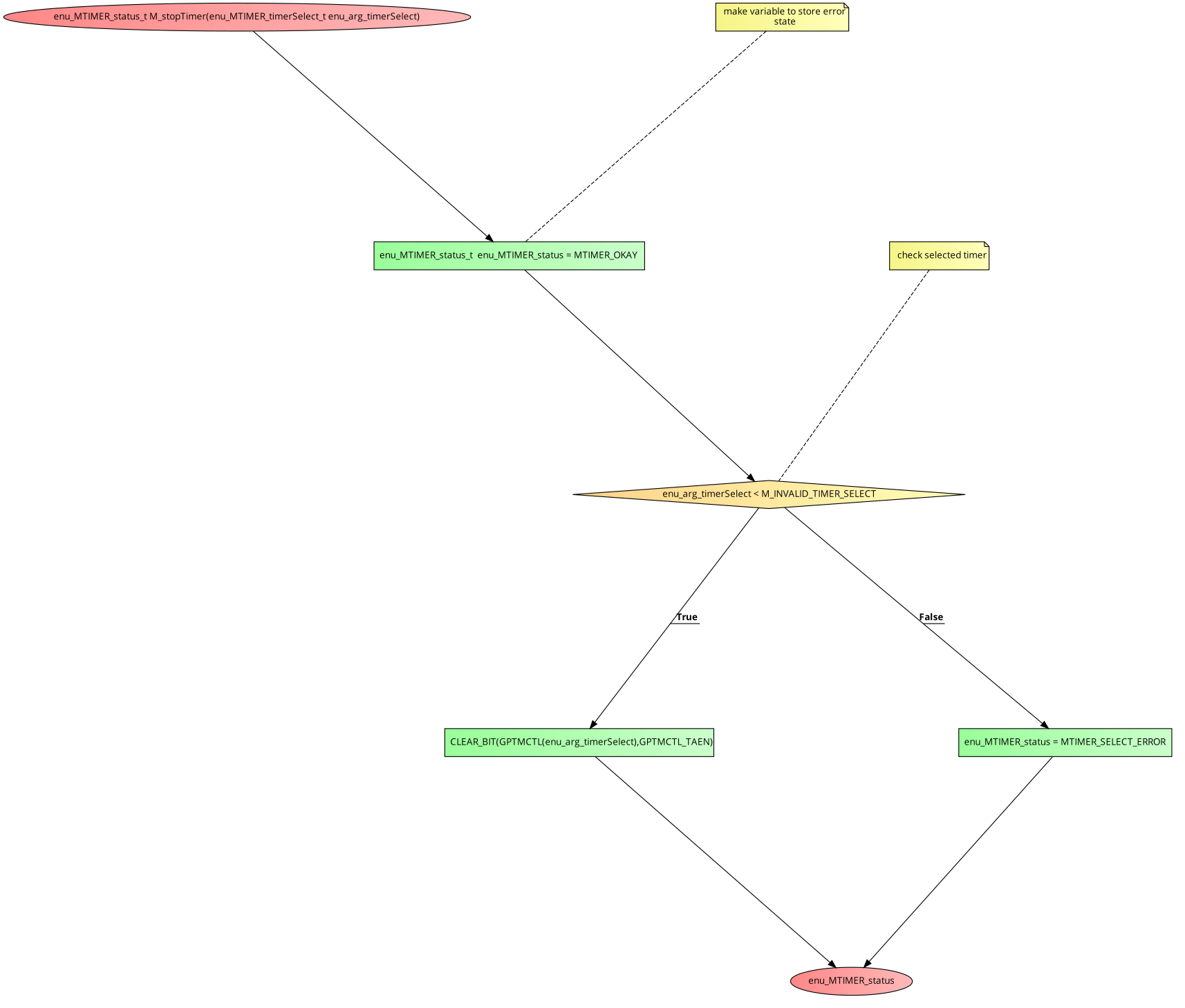
enu\_MTIMER\_status\_t M\_Timer\_getElapsedTime(enu\_MTIMER\_timerSelect\_t enu\_arg\_timerSelect , uint32\_t \*u32\_ptr\_timeMs)



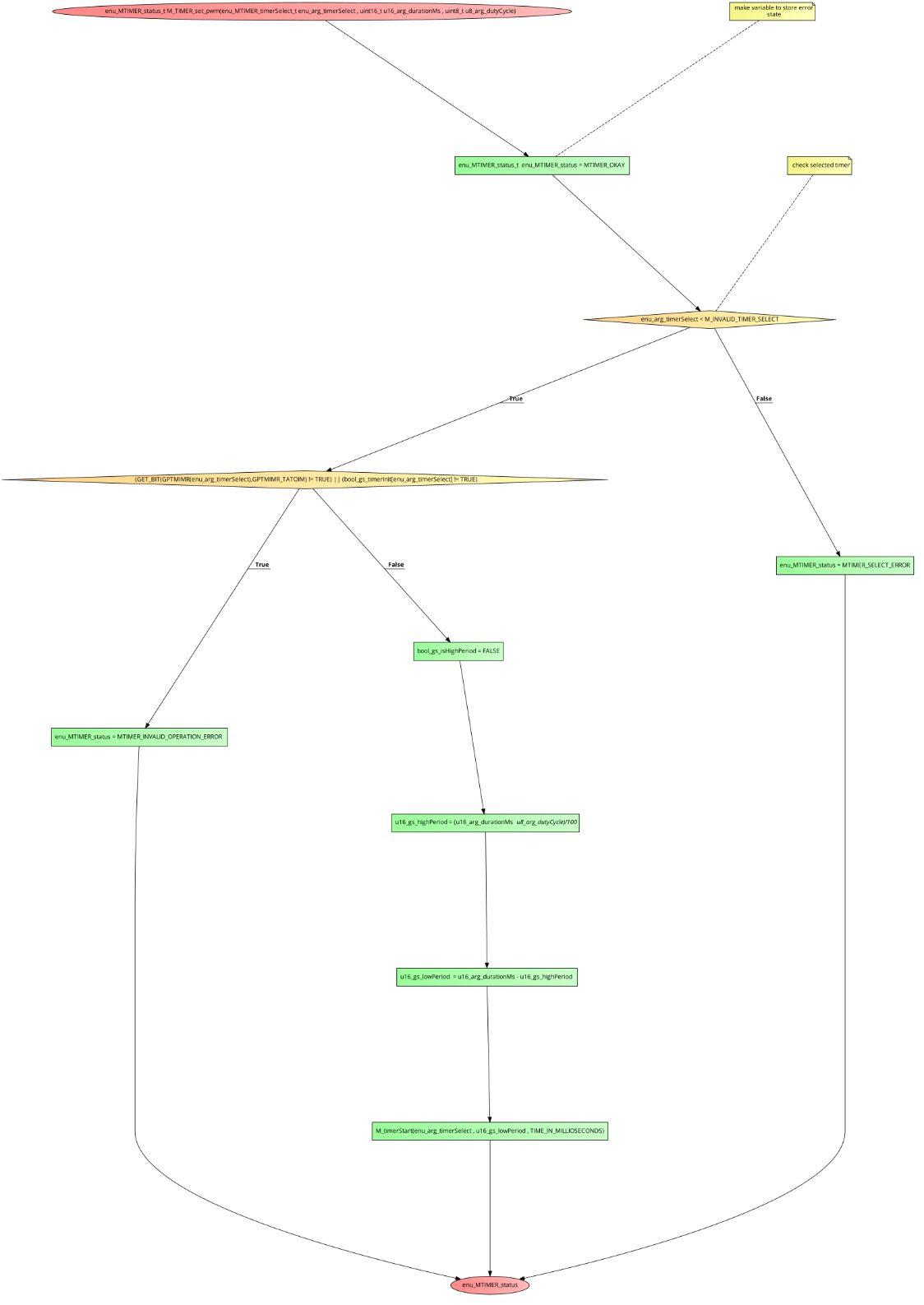
enu\_MTIMER\_status\_t M\_TIMER\_getRemainingTime (enu\_MTIMER\_timerSelect\_t enu\_arg\_timerSelect , uint32\_t \*u32\_ptr\_timeMs)



enu\_MTIMER\_status\_t M\_stopTimer(enu\_MTIMER\_timerSelect\_t enu\_arg\_timerSelect)



enu\_MTIMER\_status\_t M\_TIMER\_set\_pwm(enu\_MTIMER\_timerSelect\_t enu\_arg\_timerSelect , uint16\_t u16\_arg\_durationMs , uint8\_t u8\_arg\_dutyCycle)

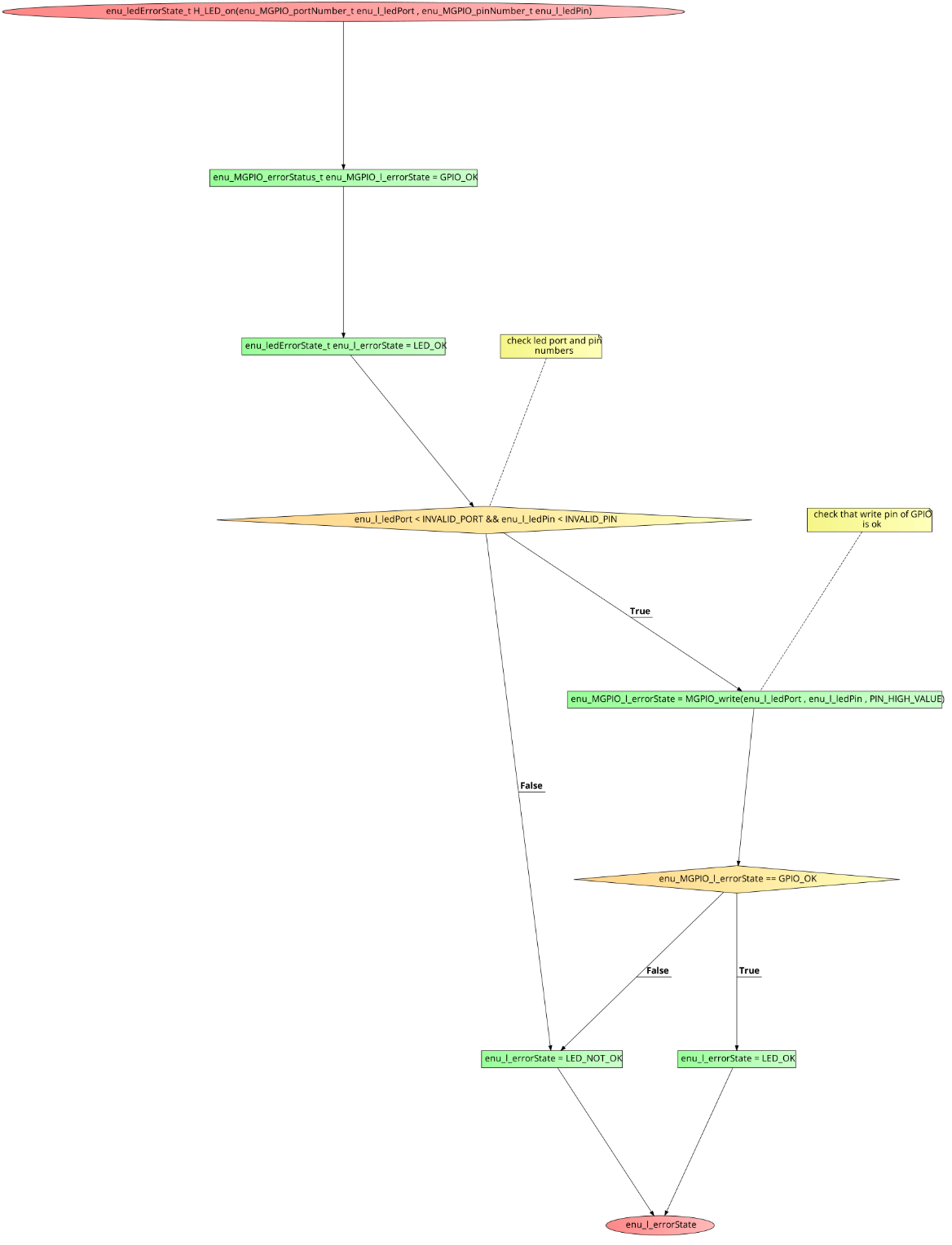


* LED

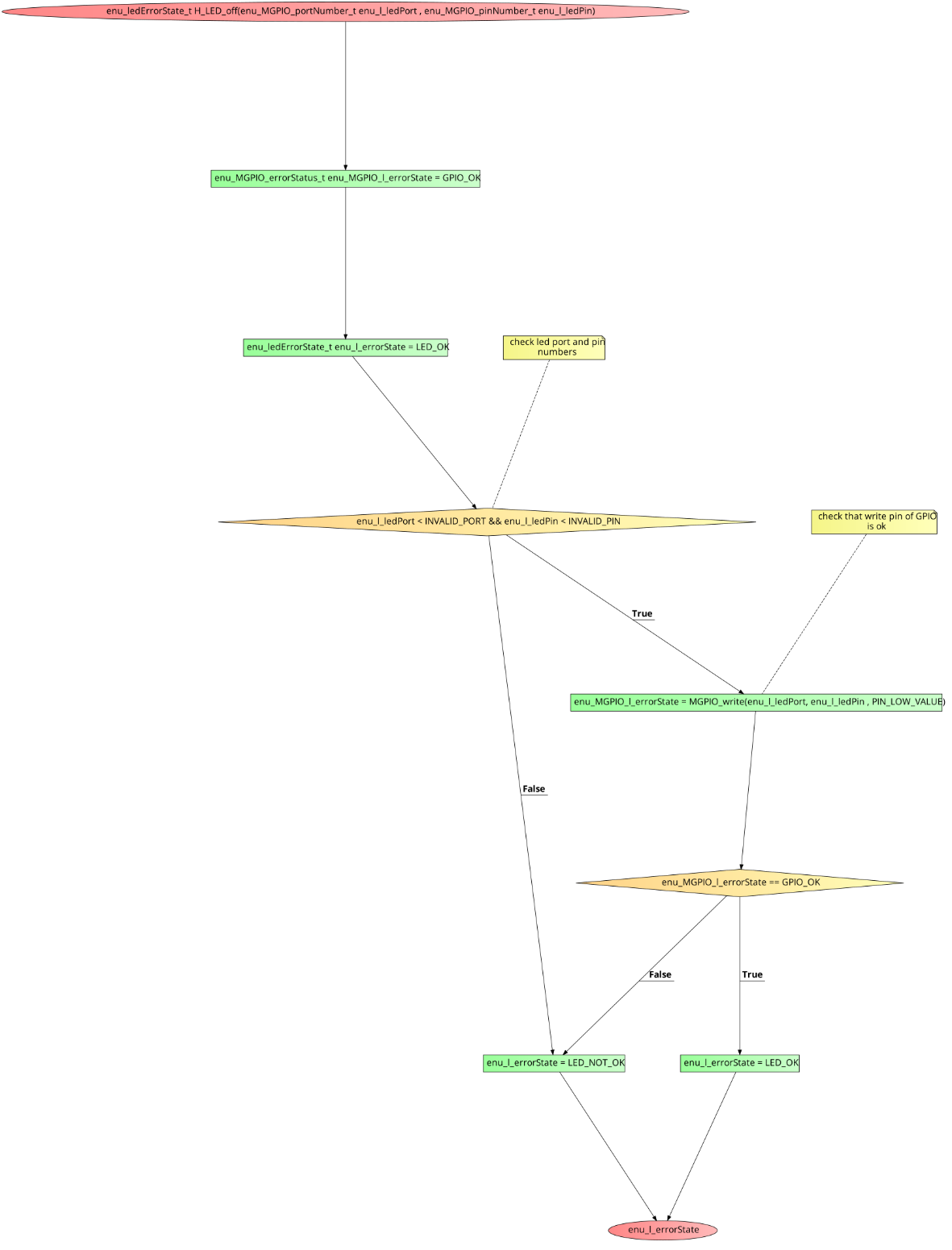
enu\_ledErrorState\_t H\_LED\_init(enu\_MGPIO\_portNumber\_t enu\_l\_ledPort , enu\_MGPIO\_pinNumber\_t enu\_l\_ledPin)



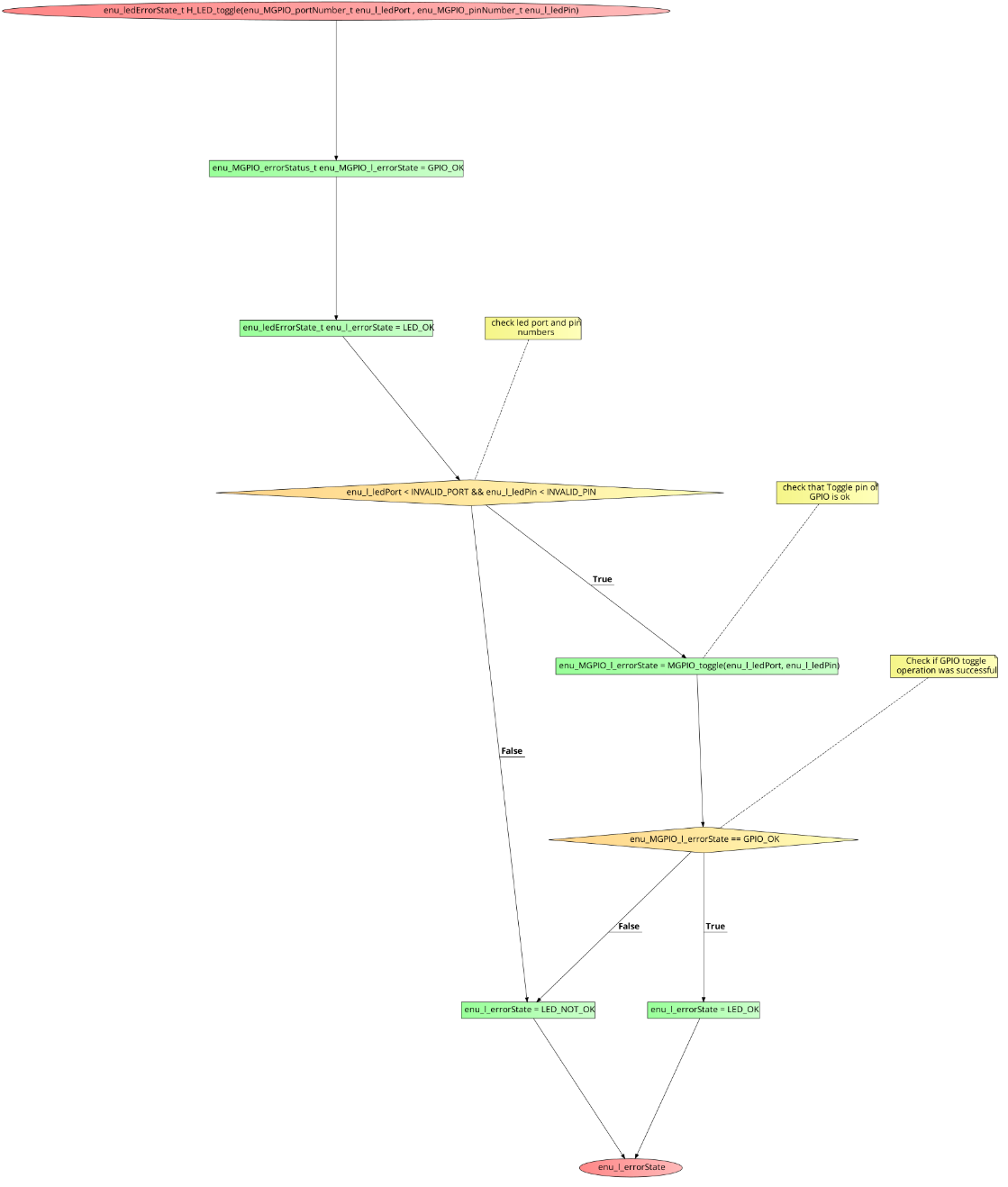
enu\_ledErrorState\_t H\_LED\_on(enu\_MGPIO\_portNumber\_t enu\_l\_ledPort , enu\_MGPIO\_pinNumber\_t enu\_l\_ledPin)



enu\_ledErrorState\_t H\_LED\_off(enu\_MGPIO\_portNumber\_t enu\_l\_ledPort , enu\_MGPIO\_pinNumber\_t enu\_l\_ledPin)

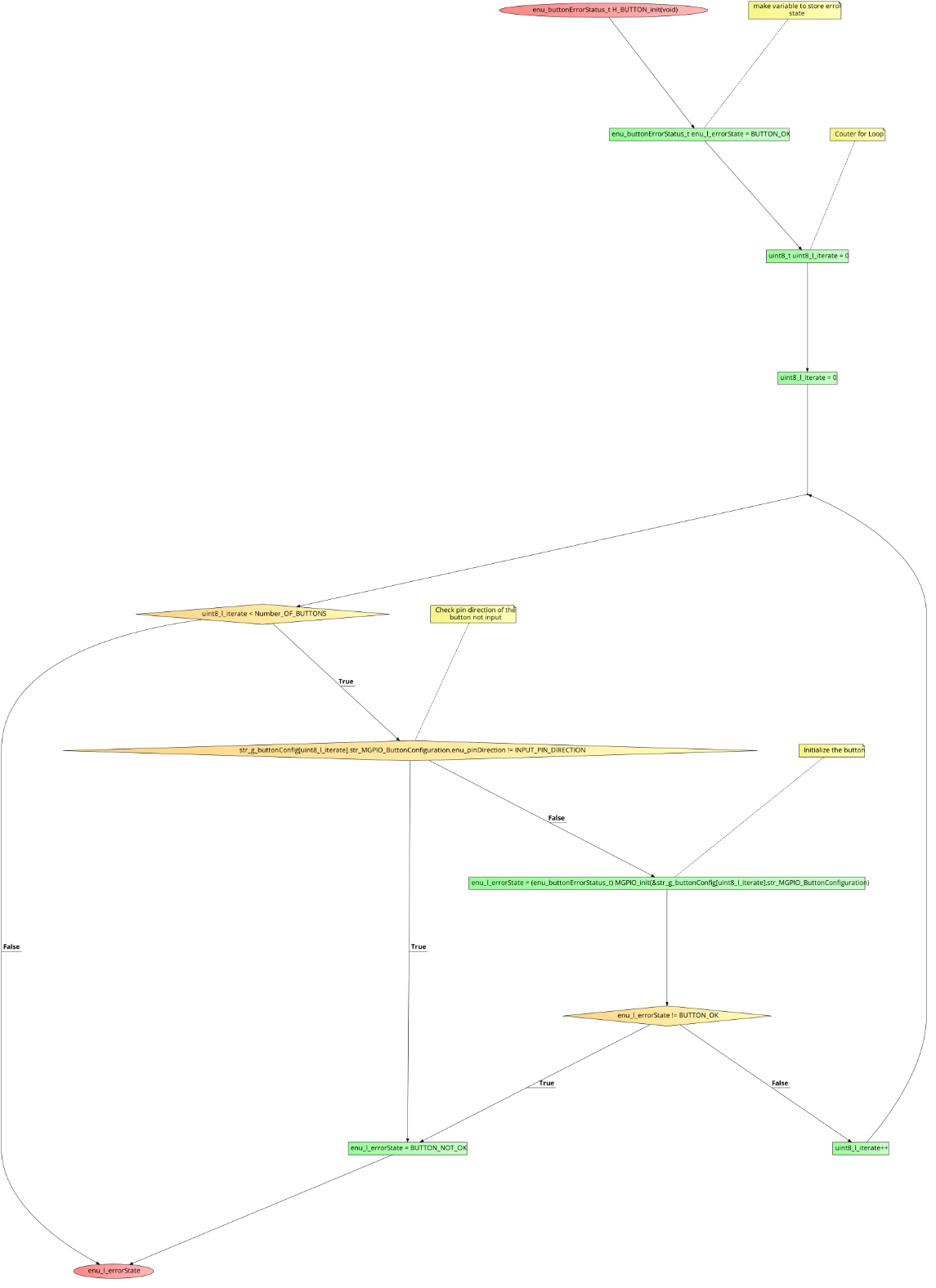


enu\_ledErrorState\_t H\_LED\_toggle(enu\_MGPIO\_portNumber\_t enu\_l\_ledPort , enu\_MGPIO\_pinNumber\_t enu\_l\_ledPin)

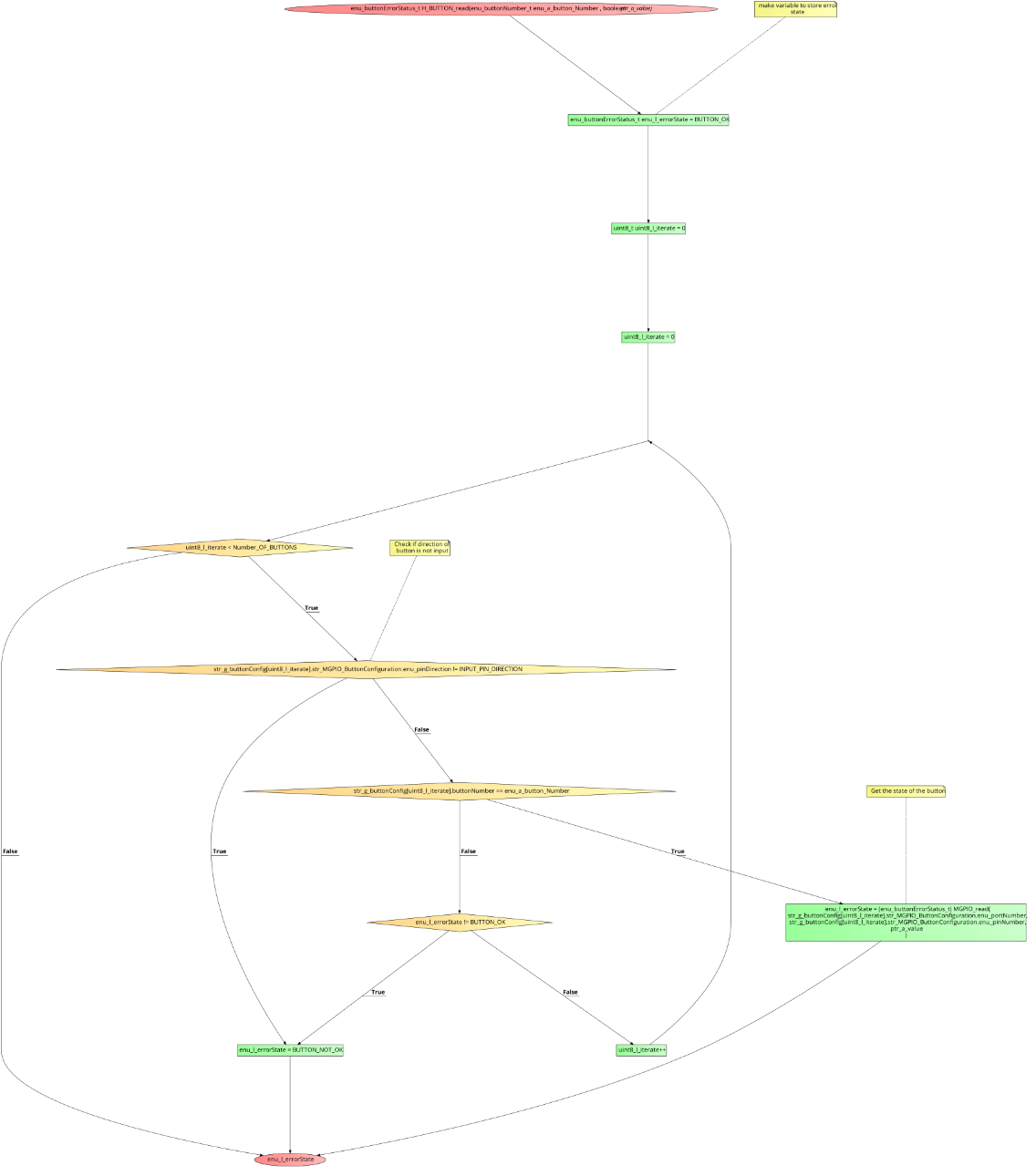


* BUTTON

enu\_buttonErrorStatus\_t H\_BUTTON\_init(void)

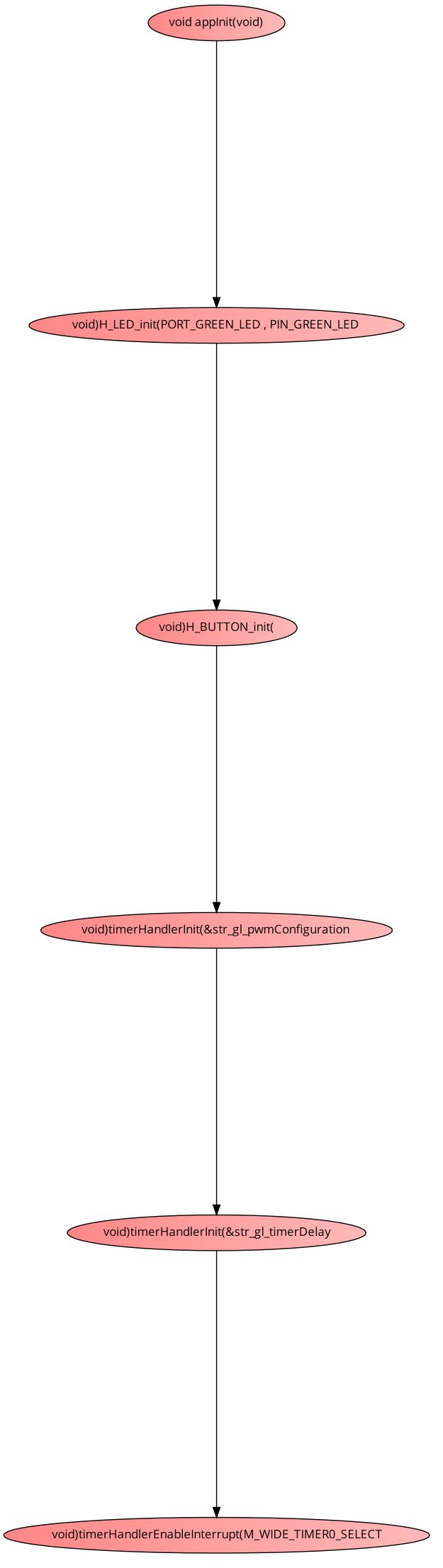


enu\_buttonErrorStatus\_t H\_BUTTON\_read(enu\_buttonNumber\_t enu\_a\_button\_Number , boolean \*ptr\_a\_value)



* APP

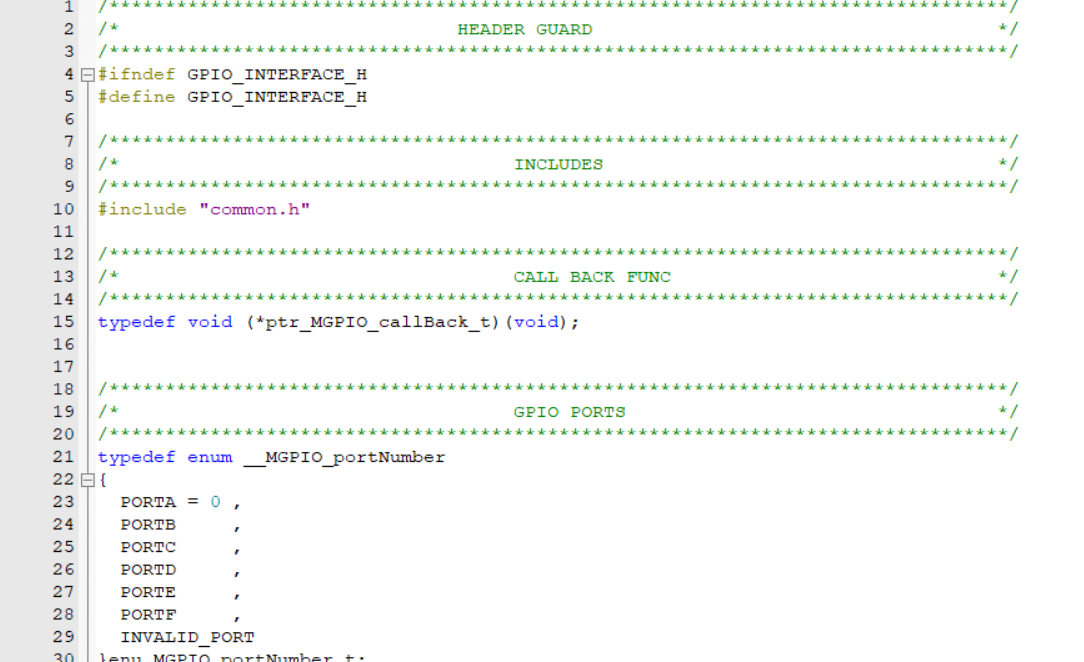
void appInit(void)

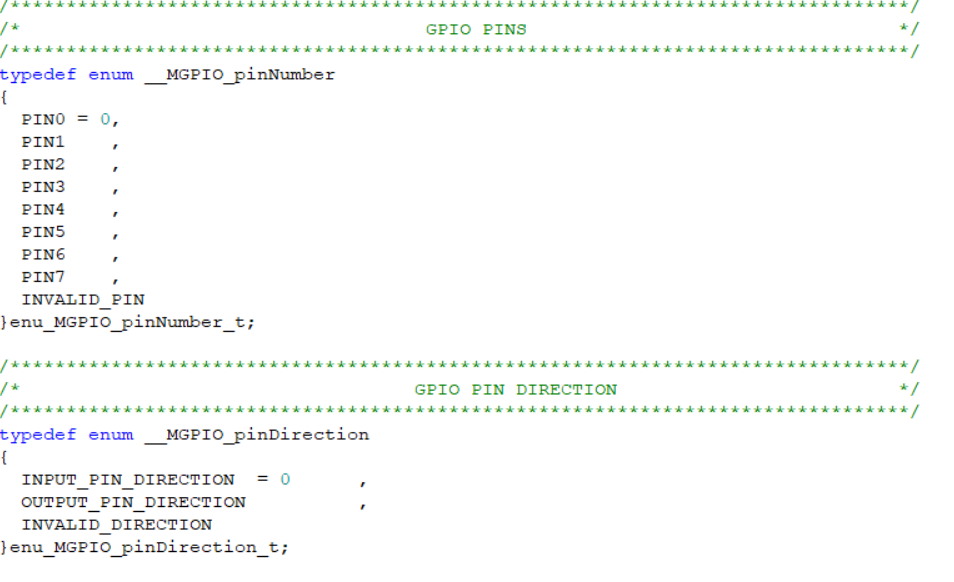


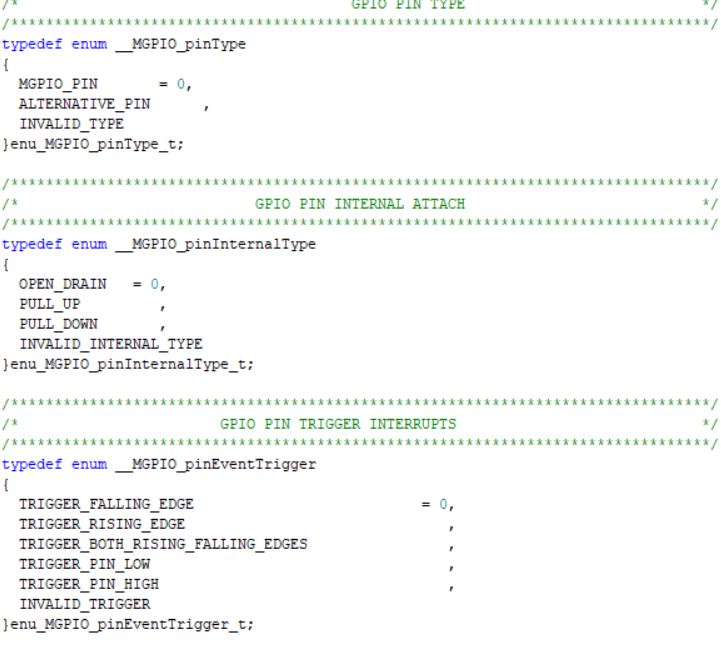
void appStart(void)

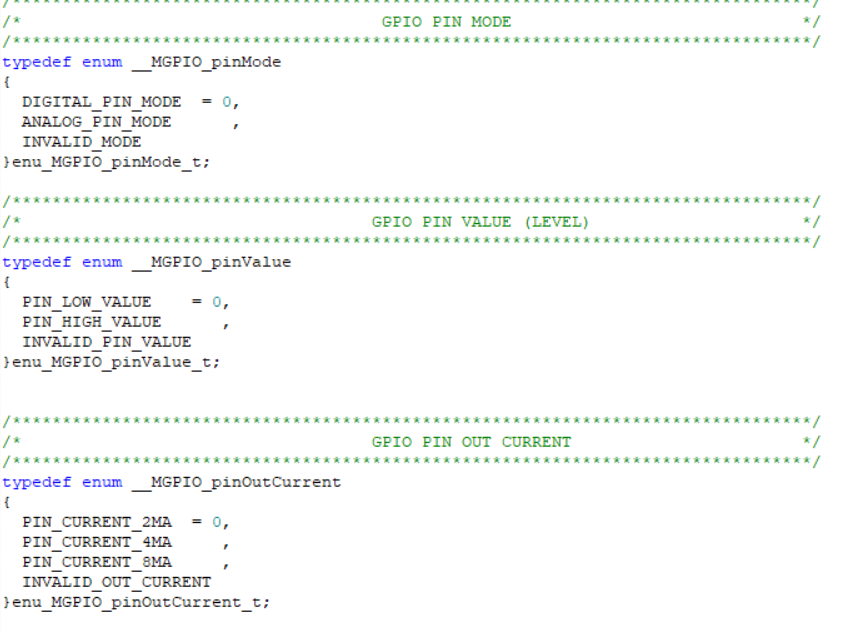


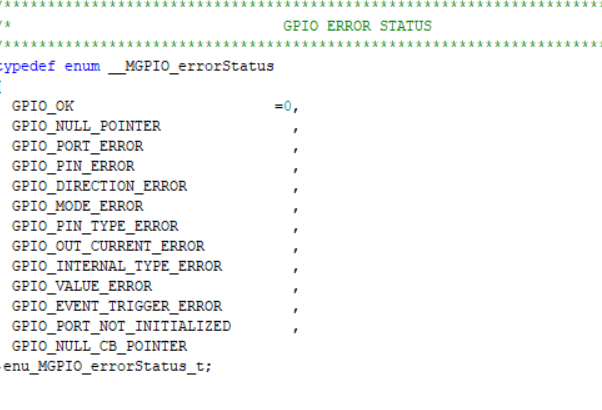
* Precompiling & Linking Configurations
* GPIO

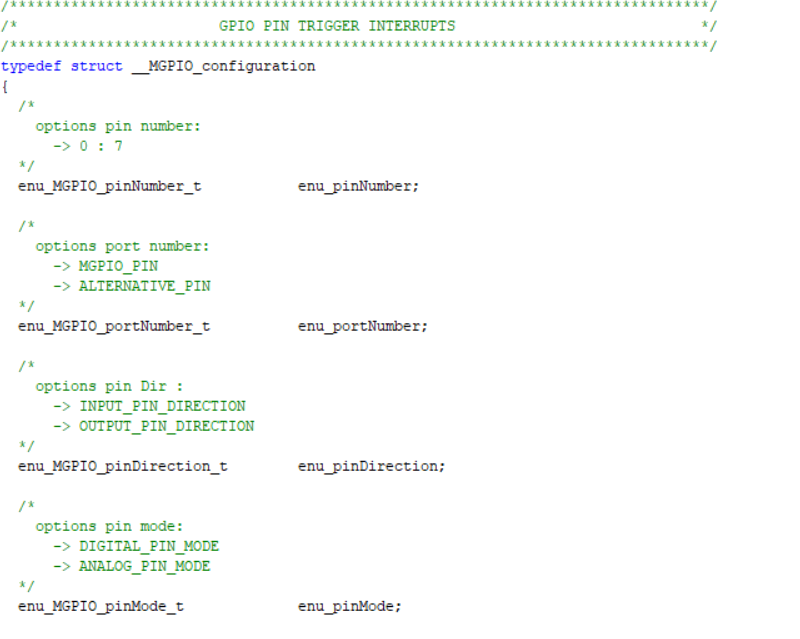


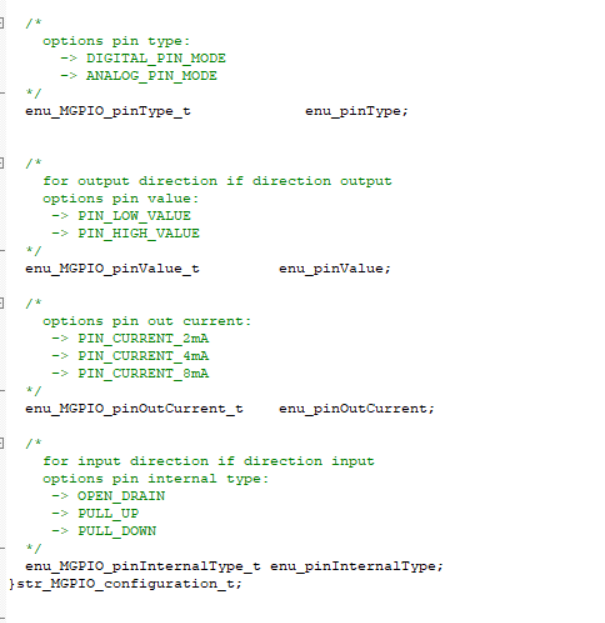


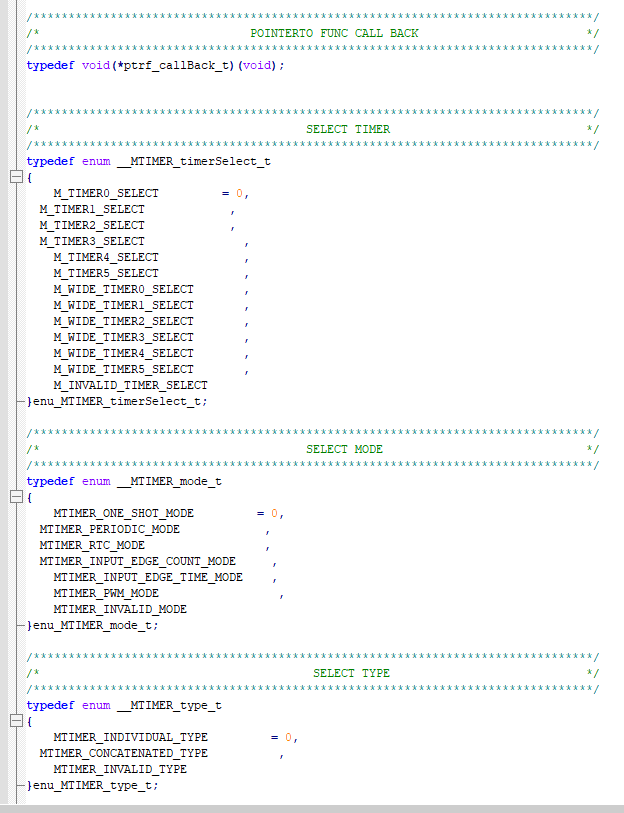


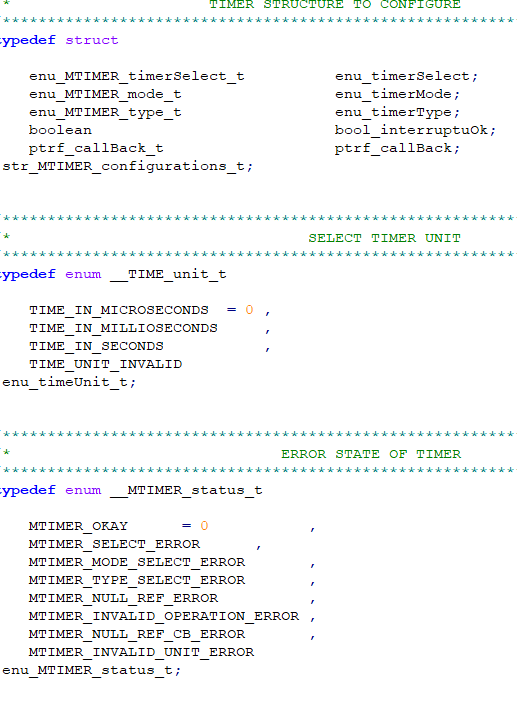




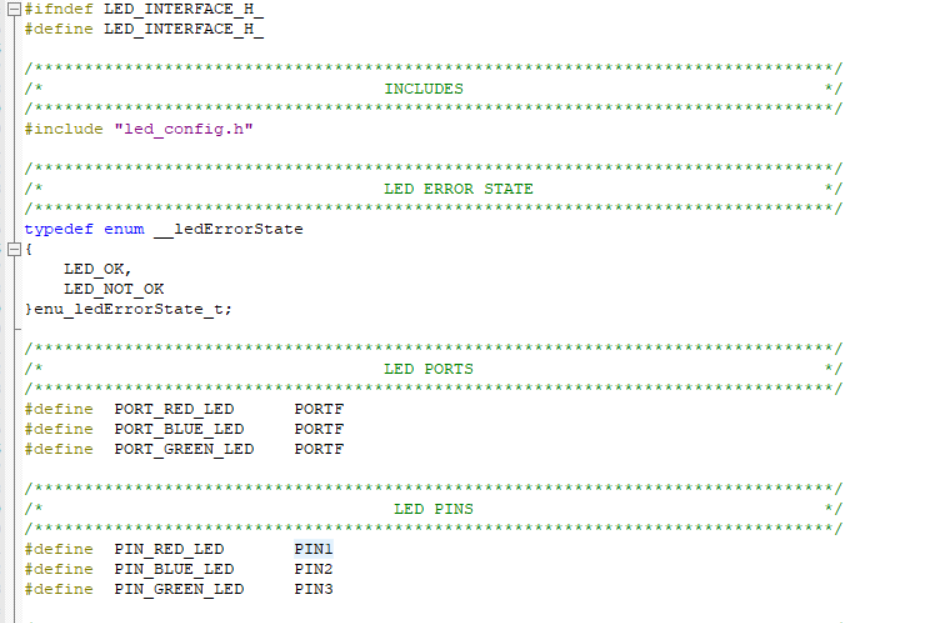


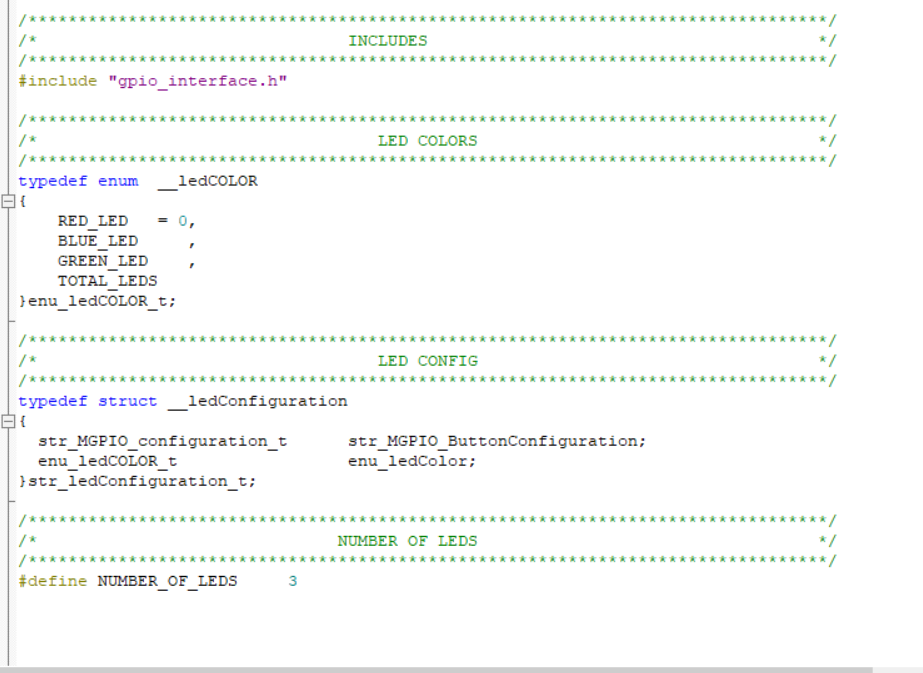


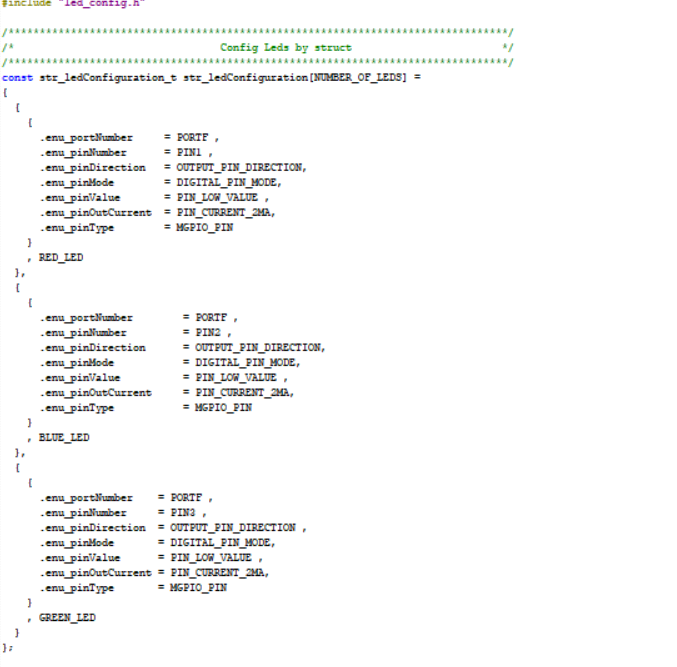
* GPT



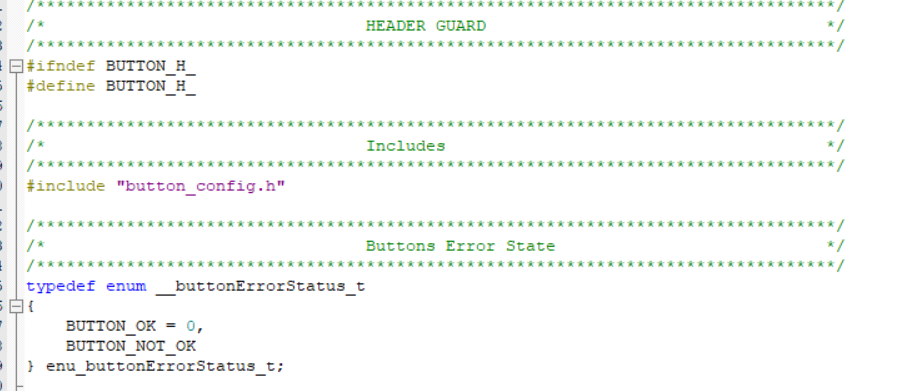
* LED



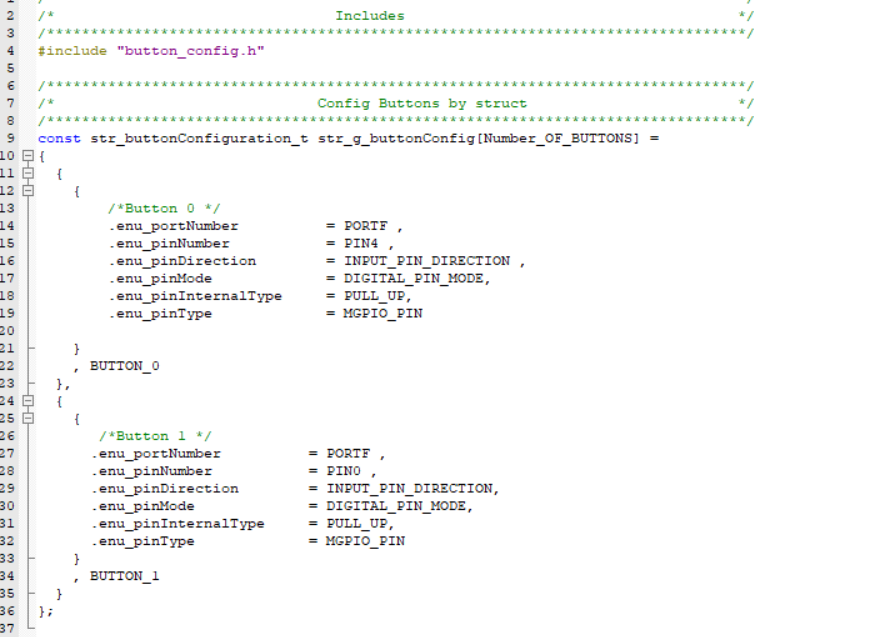




* BUTTON







That is all requirements

Thanks