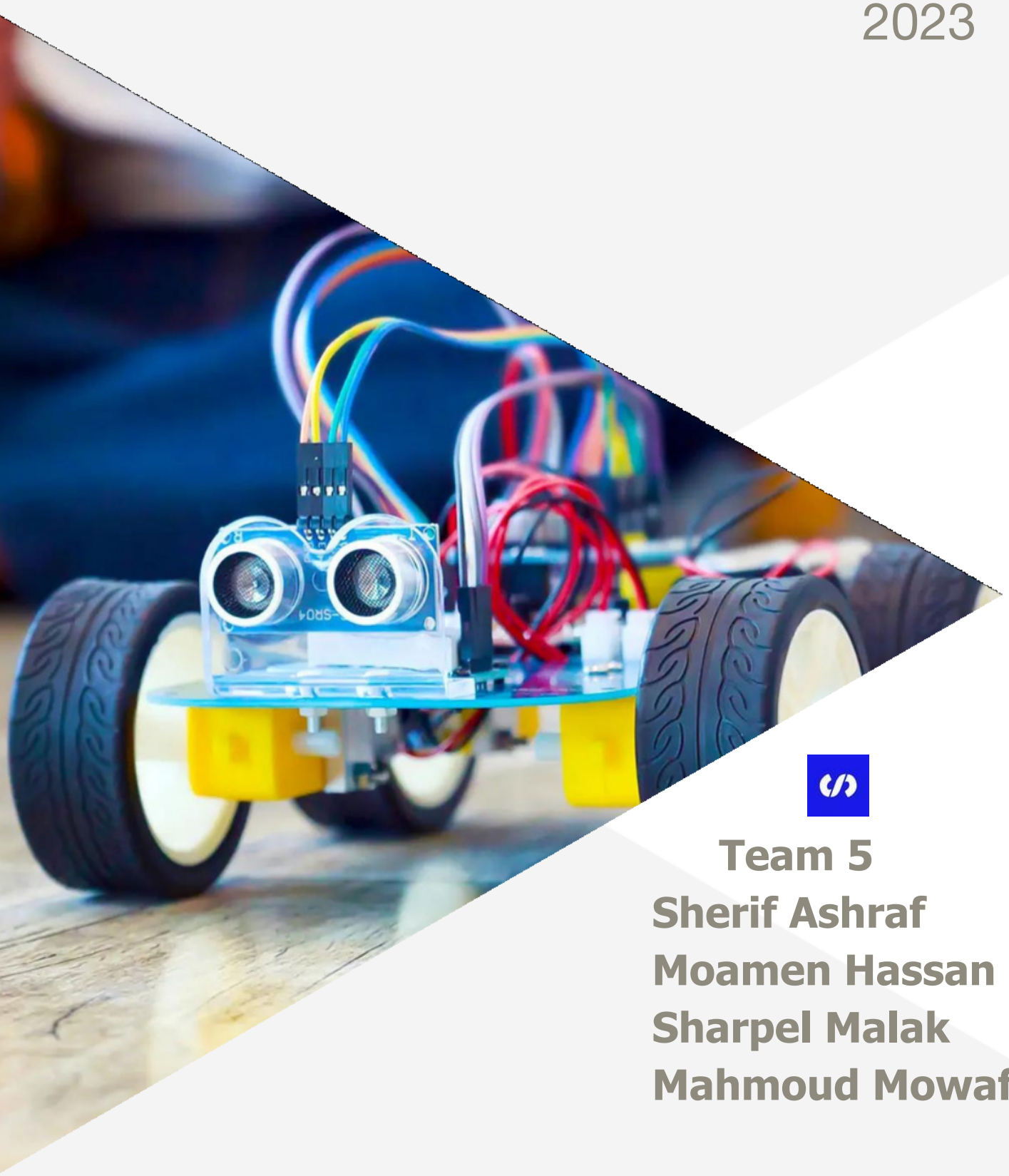
Four-driving wheel Robot

2023



Team 5

Sherif Ashraf

Moamen Hassan

Sharpel Malak

Mahmoud Mowafey

1. Project Introduction 4
1.1. Car Components 4
1.2. System Requirements 4
2. High Level Design 5
2.1. System Architecture 5
2.1.1. Definition 5
2.1.1.1. MCAL 5
2.1.1.1. APP Layer 6
2.1.2. Layered Architecture 6
2.2.1. Digital Input Output Module(DIO/GPIO) 7
2.2.2. T_Handler Module 7
2.2.3. GPT Module 7
2.2.4. Push-Button Module 7
2.2.5. PWM Module 7
2.2.6. DC Motor Module 7
2.2.7. SystemModules 8
2.3. APIs 8
2.3.1. Definition 8
2.3.2. MCAL_APIs 8
2.3.2.1. DIO 8
2.3.2.2. GPT 10
2.3.3. ECUAL_APIs 12
2.3.3.1. Push-BUTTON 12
2.3.3.2. T_Handler 12
2.3.3.4. PWM 14
2.3.3.6 LED 17
2.3.4. Application_APIs 18
2.3.4.1.APP 18
3.Low Level Design 19
3.1. MCAL 19
3.1.1. GPIO 19
3.1.1.1. GPIO_init 19
3.1.1.2. DIO_writepinn 20
3.1.1.3. DIO_readpin 20
3.1.1.4. DIO_togglepin 21
3.1.1.5. DIO_handler 21
3.1.1.6. DIO_enable_int 22
3.1.2. GPT 23
3.1.2.1 GPT_init 23
3.1.2.2 GPT_start_timer 24

3.1.2.3 GPT_enable_interrupt....25
3.1.2.4 GPT_disable_interrupt....26
3.1.2.5 GPT_stop_timer....26
3.1.2.6 GPT_get_elapsed_time....27
3.1.2.7 GPT_get_remaining_time....28
3.1.2.8 GPT_set_pwm....29
3.2. ECUAL....30
3.3.1. Push-BUTTON....30
3.3.1.1 PUSH_BTN_intialize....30
3.3.1.2 PUSH_BTN_read_state....31
3.3.2. DCM....32
3.3.2.1.DCM_motorInit....32
3.3.2.2.DCM_changeDirection....33
3.3.2.3.DCM_vdStop....33
3.3.2.4.DCM_setDutyCycle....34
3.3.2.5.DCM_rotate....35
3.3.2.6. DCM_MoveForward / DCM_MoveBackward....36
3.3.3. T_Handler....37
3.3.3.1 HANDLER_init....37
3.3.3.2 HANDLER_start_timer....38
3.3.3.3 HANDLER_enable_interrupt....39
3.3.3.4 HANDLER_disable_interrupt....40
3.3.3.5 HANDLER_stop_timer....41
3.3.3.6 HANDLER_get_elapsed_time....42
3.3.3.7 HANDLER_get_remaining_time....43
3.3.3.8 HANDLER_set_pwm....44
3.3.4. PWM....44
3.3.5. LED....44
3.3.5.1.LED_init....44
3.3.5.2.LED_on....45
3.3.5.3.LED_off....45
4. Precompiling and linking configuration....46
4.1.GPIO....46
4.1.1.linking Configuration....46
4.1.2.Precompiling Configuration....46
4.2.GPT....46
4.2.1 linking Configuration....46
4.2.2 Precompiling Configuration....46
4.3. LED....46
4.3.1 linking Configuration....46
4.3.2 Precompiling Configuration....46
4.4. DCM....46

4.4.1 Linking Configuration.....46
4.4.1 Precompiling Configuration..... 46
4.5. PWM..... 47
4.5.1 Linking Configuration.....47
4.5.2 Precompiling Configuration..... 47
4.6 Push Button..... 48
4.6.1 Precompiling Configuration..... 48

1. Project Introduction

1.1. Car Components

- Use Sprints Kit with TivaC launchpad plugged in
- You will develop your application on the ARM microcontroller
- Four motors (M1, M2, M3, M4)
- One button to start (PB1)
- One button for stop (PB2)
- Four LEDs (LED1, LED2, LED3, LED4)

1.2. System Requirements

- The car starts initially from 0 speed
- When PB1 is pressed, the car will move forward after 1 second
- The car will move forward to create the longest side of the rectangle for 3 seconds with 50% of its maximum speed
- After finishing the first longest side the car will stop for 0.5 seconds, rotate 90 degrees to the right, and stop for 0.5 second
- The car will move to create the short side of the rectangle at 30% of its speed for 2 seconds
- After finishing the shortest side, the car will stop for 0.5 seconds, rotate 90 degrees to the right, and stop for 0.5 second
- Steps 3 to 6 will be repeated infinitely until you press the stop button (PB2)
- PB2 acts as a sudden break, and it has the highest priority
- LEDs Operations
    a. LED1: On means moving forward on the long side
    b. LED2: On means moving forward on the short side
    c. LED3: On means stop
    d. LED4: On means Rotating

2. High Level Design

2.1. System Architecture

- Layered architecture in embedded systems organises software components into distinct layers, each with a specific responsibility and level of abstraction.
- The layers are arranged hierarchically, starting from the hardware layer, followed by MCAL, HAL and ending with the application layer.
- Layering provides benefits like modularity, scalability, and maintainability by separating software components into distinct layers with clear separation of concerns.

2.1.1. Definition

2.1.1.1. MCAL

- MCAL is a layer in embedded systems that provides a standardised interface to the microcontroller hardware.
- It includes functions/drivers for controlling hardware peripherals and is critical for enabling higher-level software layers to be developed independently of the hardware platform.

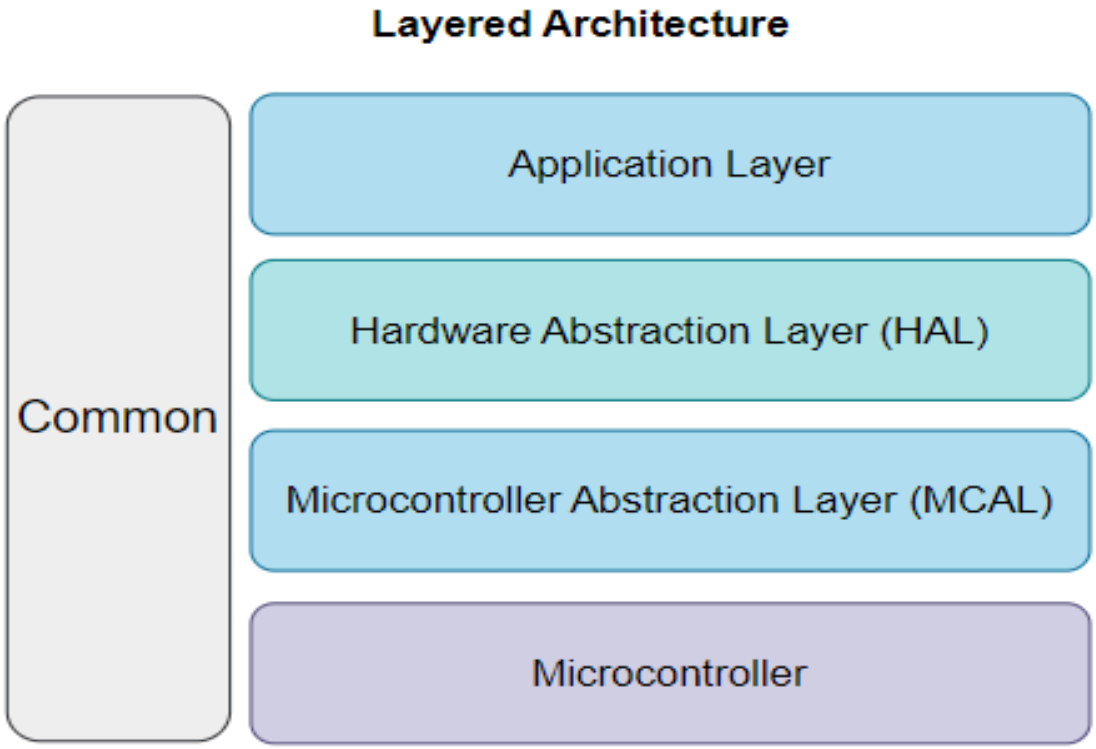
2.1.1.1. ECUAL

- In a layered architecture for embedded systems, the ECU (Electronic Control Unit) layer sits above the MCAL layer and provides a higher-level, more application-specific interface for controlling the system's functions.
- It implements the system's primary functionality and interacts with the MCAL layer through well-defined APIs and interfaces. The ECU layer may also include functionality for managing system diagnostics, error handling, and communication with other ECUs or external systems

2.1.1.1. APP Layer

- In a layered architecture for embedded systems, the APP (Application) layer is responsible for providing the system's primary functionality and interacts with the lower layers through well-defined APIs and interfaces. It may include software components for implementing specific functions, managing system diagnostics, error handling, and communication with other systems.

2.1.2. Layered Architecture



2.2. Modules Descriptions

2.2.1. Digital Input Output Module(DIO/GPIO).

- This module deals with the digital input and output operations, such as reading and writing to digital pins of a microcontroller or a microprocessor. It may include functions for setting pin direction, reading and writing digital values, and handling interrupts related to digital pins.

2.2.2. T_Handler Module

- A ***T_Handler*** (***Timer Handle***r) module is a programmable handler that can be used to control the GPT driver from the HAL layer without calling the GPT in the APP Layer.

2.2.3. GPT Module.

- The *GPT* module is responsible for generating timing events that are used by other modules in the system. It provides a set of APIs to configure the timer clock source and prescaler, set the timer mode (count up/down), set the timer period, enable/disable timer interrupts, and define an ISR that will be executed when the timer event occurs .

2.2.4. Push-Button Module.

- Is a feature in microcontrollers that allows an external signal to interrupt the normal program execution and perform a specific task. It is a hardware feature that is triggered by a change in the state of an external signal, such as a button press or a sensor trigger.

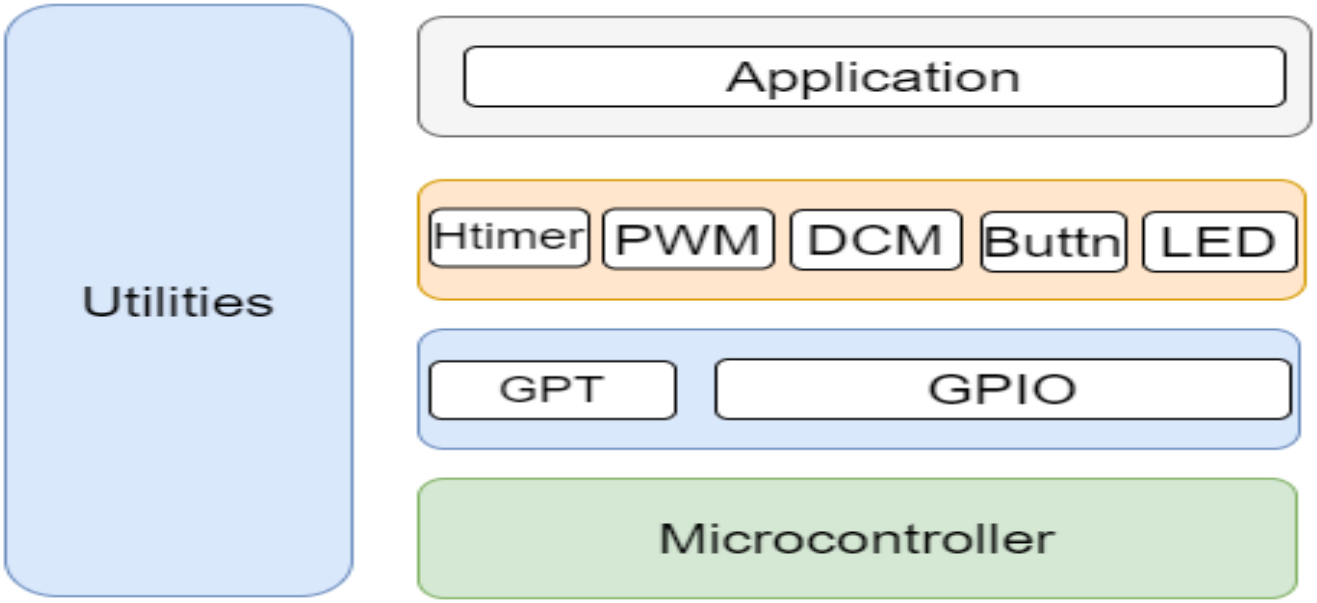
2.2.5. PWM Module.

- The pulse width modulation (PWM) module enables the generation of pulse width modulated signals on GPIO. The module implements an up or up-and-down counter with four PWM channels that drive assigned GPIOs .

2.2.6. DC Motor Module.

- DC Motor is a device which transforms electrical energy into mechanical energy. Specifically, a DC motor uses DC current to convert electrical energy into mechanical energy. The basic principle of a motor is the interaction between the magnetic field and current to produce a force within the motor which helps the motor to rotate. So when the electric current is passed through a coil in a magnetic field, a magnetic force is generated which produces a torque resulting in the movement of the motor. The direction of the motor is controlled by reversing the current .

2.2.7. SystemModules.



2.3. APIs

2.3.1. Definition

- An *API* is an *Application Programming Interface* that defines a set of *routines*, *protocols* and *tools* for creating an application. An *API* defines the high level interface of the behaviour and capabilities of the component and its inputs and outputs.

2.3.2. MCAL_APIs

2.3.2.1. DIO

```
|@name       : GPIO_init
|@berif      : this function initialies GPIO pin as ( OUTPUT , INPUT OR INTERRUPT )
|@param[in]  : pointer to str_GPIO_configs_t type with desired option
              [REQUIRED OPTOINS]
               - enu_port_num          : Select Port Number
               - enu_pin_num           : Select Pin Number
               - enu_pin_direction     : Select Pin Direction
               - enu_pin_mode          : Select Pin Mode
              [Case Output]
               - enu_pin_level         : Select Output level
               - enu_pin_out_current   : Select output current
              [Case Input]
               - enu_pin_internal_type         : Select Internal attach type
               - bool_use_interrupt            : Select if it is interrupt or not
               - enu_GPIO_pin_event_trigger : Select sense trigger
               - ptr_GPIO_cb                   : Set call back to upper layer

|@return    : GPIO_OKAY              (In case of success initialization)
              GPIO_NULL_REF          (In case of Null pointer argument )
              GPIO_PORT_ERROR        (In case of Invalid port number   )
              GPIO_PIN_ERROR         (In case of Invalid pin nimber    )
```

GPIO_DIRECTION_ERROR (In case of Invalid pin direction)
GPIO_MODE_ERROR (In case of Invalid mode selection)
GPIO_OUT_CURRENT_ERROR (In case of Invalid output current)
GPIO_INTERNAL_TYPE_ERROR (In case of Invalid internal type)
GPIO_LEVEL_ERROR (In case of Invalid output level)
GPIO_EVENT_TRIGGER_ERROR (In case of Invalid sense trigger)
GPIO_NULL_CB_REF (In case of Null pointer to cbf)

```
enu_GPIO_status_t GPIO_init(str_GPIO_configs_t  *ptr_GPIO_configs);
```

| write the desired digital logic on the pin
|Parameters
| [in] arg_port_num the desired port for initializing the port.
| [in] arg_pin_num the desired pin inside the pin..
| [in] ptr__value apply the desired logic level..
| Return
| An enu_GPIO_status_t value indicating the success or failure of
| the operation *(GPIO_OK if the operation succeeded, GPIO_ERROR*
| *otherwise)*
|

```
enu_GPIO_status_t GPIO_write( enu_GPIO_port_num_t  arg_port_num,
                              enu_GPIO_pin_num_t  arg_pin_num,
                              boolean  *ptr_value);
```

| Read the applied digital logic on the pin
|Parameters
| [in] arg_port_num the desired port for initializing the port.
| [in] arg_pin_num the desired pin inside the pin..
| [in] ptr__value stores the pin current logic..
| Return
| An enu_GPIO_status_t value indicating the success or failure of
| the operation *(GPIO_OK if the operation succeeded, GPIO_ERROR otherwise)*
|

```
enu_GPIO_status_t GPIO_read( enu_GPIO_port_num_t  arg_port_num,
                             enu_GPIO_pin_num_t  arg_pin_num,
                             boolean  *ptr_value);
```

| toggle digital logic on the pin
|Parameters
| [in] arg_port_num the desired port for initializing the port.
| [in] arg_pin_num the desired pin inside the pin..
|
| Return
| An enu_GPIO_status_t value indicating the success or failure of
| the operation *(GPIO_OK if the operation succeeded, GPIO_ERROR*
| *otherwise)*
|

```
enu_GPIO_status_t GPIO_toggle( enu_GPIO_port_num_t  arg_port_num,
                               enu_GPIO_pin_num_t  arg_pin_num);
```

| Enable the desired interrupt on the pin
|Parameters
| [in] arg_port_num the desired port for initializing the port.
| [in] arg_pin_num the desired pin inside the pin..
| Return **void**
|

```
void GPIO_enable_interrupt( enu_GPIO_port_num_t  arg_port_num,
                            enu_GPIO_pin_num_t  arg_pin_num);
```

| Enable the desired interrupt on the pin

```
|Parameters
|                  [in] arg_port_num    the desired port for initializing the port.
|                  [in] arg_pin_num     the desired pin inside the pin..
| Return           void
|
void GPIO_disable_interrupt( enu_GPIO_port_num_t  arg_port_num,
                             enu_GPIO_pin_num_t  arg_pin_num);
```

2.3.2.2. GPT

```
| Initializing the GPT Module
|Parameters
|                  none
| Return
|                   An enu_GPT_status_t value indicating the success or failure of
|                  the operation (GPT_OK if the operation succeeded, GPT_ERROR
|                  otherwise)
enu_GPT_status_t GPT_init(str_GPT_configs_t * ptr_GPT_configs );
|Start the timer
|Parameters
|                  [in] enu_arg_GPT_timer_select    the desired GPT channel.
|                  [in] u32_arg_time     the desired delay .
|                  [in] enu_arg_time_unit     the desired delay unit.
|
| Return
|                   An enu_GPT_status_t value indicating the success or failure of
|                  the operation (GPT_OK if the operation succeeded, GPT_ERROR
|                  otherwise)
|
enu_GPT_status_t GPT_start_timer(enu_GPT_timer_select_t  enu_arg_GPT_timer_select,
                                 unit32_t              u32_arg_time,
                                 enu_time_unit_t enu_arg_time_unit );


| get the elapsed time
|Parameters
|                  [in] enu_arg_GPT_timer_select    the desired GPT channel.
|                  [in] u32_ptr_time                pointer to variable to store the remaining time.
| Return
|                   An enu_GPT_status_t value indicating the success or failure of
|                  the operation (GPT_OK if the operation succeeded, GPT_ERROR
|                  otherwise)
|
enu_GPT_status_t GPT_get_elapsed_time(
                                      enu_GPT_timer_select_t enu_arg_GPT_timer_select,
                                       unit32_t                    *u32_ptr_time  );

|get the remaining timer
|Parameters
|                  [in] enu_arg_GPT_timer_select    the desired GPT channel.
|
| Return
|                   An enu_GPT_status_t value indicating the success or failure of
|                  the operation (GPT_OK if the operation succeeded, GPT_ERROR
|                  otherwise)
```

```
|
enu_GPT_status_t GPT_get_remaining_time(
                                        enu_GPT_timer_select_t enu_arg_GPT_timer_select,
                                         unit32_t                    *u32_ptr_time  );


| enable the GPT interrupt
|Parameters
|                  [in] enu_arg_GPT_timer_select    the desired GPT channel.
| Return
|                   An enu_GPT_status_t value indicating the success or failure of
|                  the operation (GPT_OK if the operation succeeded, GPT_ERROR
|                  otherwise)
|
enu_GPT_status_t  GPT_enable_interrupt(
                                        enu_GPT_timer_select_t enu_arg_GPT_timer_select   );
| disable the GPT interrupt
|Parameters
|                  [in] enu_arg_GPT_timer_select    the desired GPT channel.
|                   An enu_GPT_status_t value indicating the success or failure of
|                  the operation (GPT_OK if the operation succeeded, GPT_ERROR
|                  otherwise)
enu_GPT_status_t  GPT_disable_interrupt(
                                        enu_GPT_timer_select_t enu_arg_GPT_timer_select   );
|stop the timer
|Parameters
|                  [in] enu_arg_GPT_timer_select    the desired GPT channel.
|
| Return
|                  [in] enu_arg_GPT_timer_select    the desired GPT channel.
|                   An enu_GPT_status_t value indicating the success or failure of
|                  the operation (GPT_OK if the operation succeeded, GPT_ERROR
|                  otherwise)
enu_GPT_status_t GPT_stop_timer(
                                        enu_GPT_timer_select_t enu_arg_GPT_timer_select,
                                         );
```

2.3.3. ECUAL_APIs

2.3.3.1. Push-BUTTON

```
/*
*Function: PUSH_BTN_intialize
*
*Description: Initialises a push button Pins based on the configuration settings.
*
*Parameters: Void.
*
*Return Type: Void.
*/
```

```
void PUSH_BTN_intialize(void);



/*
*Function     : PUSH_BTN_read_state
*
*Description  : Reads the current en_g_state of a push button and returns its value.
*
*Parameters:
*btn          : A pointer to an ST_PUSH_BTN_t struct that contains the configuration settings
*and current en_g_state information for the push button.
*btn_state : A pointer to an EN_PUSH_BTN_state_t enum where the current en_g_state of the push
*button will be stored.
*Return Type  : Void.
*
*Overall, the PUSH_BTN_read_state function provides a way to read the current en_g_state of a
*push button and return its value. By using this function, the software can determine whether
*the push button is currently pressed or released and take appropriate action based on its
*en_g_state.
*/
void PUSH_BTN_read_state(Uchar8_t btnNumber , EN_PUSH_BTN_state_t);
```

2.3.3.2. T_Handler

```
| Initializing the GPT Module
|Parameters
|                 none
| Return
|                  An enu_handler_error_status_tvalue indicating the success or failure of
|                 the operation (Handler_OK if the operation succeeded, Handler_ERROR
|                 otherwise)
|
enu_handler_error_status_t HANDLER_init(str_GPT_configs_t * ptr_GPT_configs );

|Start the handler
|Parameters
|                 [in] enu_arg_GPT_timer_select    the desired GPT channel.
|                 [in] u32_arg_time     the desired delay .
|                 [in] enu_arg_time_unit     the desired delay unit.
|
| Return
|                  An enu_handler_error_status_tvalue indicating the success or failure of
|                 the operation (Handler_OK if the operation succeeded, Handler_ERROR
|                 otherwise)
|
enu_handler_error_status_t HANDLER_start_timer(
                            enu_GPT_timer_select_t  enu_arg_GPT_timer_select,
                              unit32_t              u32_arg_time,
                               enu_time_unit_t enu_arg_time_unit );

|stop the timer
```

| **Parameters**
| [in] enu_arg_GPT_timer_select the desired GPT channel.
|
| **Return**
| An enu_handler_error_status_tvalue indicating the success or failure of
| the operation *(Handler_OK if the operation succeeded, Handler_ERROR*
| *otherwise)*
|

```
enu_handler_error_status_t HANDLER_stop_timer(
                                    enu_GPT_timer_select_t enu_arg_GPT_timer_select,
                                     );
```

| get the elapsed time
| **Parameters**
| [in] enu_arg_GPT_timer_select the desired GPT channel.
| [in] u32_ptr_time pointer to variable to store the remaining time.
| **Return**
| An enu_handler_error_status_tvalue indicating the success or failure of
| the operation *(Handler_OK if the operation succeeded, Handler_ERROR*
| *otherwise)*
|

```
enu_handler_error_status_t HANDLER_get_elapsed_time(
                                    enu_GPT_timer_select_t enu_arg_GPT_timer_select,
                                     unit32_t                    *u32_ptr_time  );
```

|get the remaining timer
| **Parameters**
| [in] enu_arg_GPT_timer_select the desired GPT channel.
|
| **Return**
| An enu_handler_error_status_tvalue indicating the success or failure of
| the operation *(Handler_OK if the operation succeeded, Handler_ERROR*
| *otherwise)*
|

```
enu_handler_error_status_t HANDLER_get_remaining_time(
                                    enu_GPT_timer_select_t enu_arg_GPT_timer_select,
                                     unit32_t                    *u32_ptr_time  );
```

| enable the GPT interrupt
| **Parameters**
| [in] enu_arg_GPT_timer_select the desired GPT channel.
| **Return**
| An enu_handler_error_status_tvalue indicating the success or failure of
| the operation *(Handler_OK if the operation succeeded, Handler_ERROR*
| *otherwise)*
|

```
enu_handler_error_status_t  HANDLER_enable_interrupt(
                                    enu_GPT_timer_select_t enu_arg_GPT_timer_select   );
```

| disable the GPT interrupt
| **Parameters**
| An enu_handler_error_status_tvalue indicating the success or failure of
| the operation *(Handler_OK if the operation succeeded, Handler_ERROR*
| *otherwise)*
|

```
enu_handler_error_status_t HANDLER_disable_interrupt(
                                    enu_GPT_timer_select_t enu_arg_GPT_timer_select   );
```

2.3.3.4. PWM

```
/*
* Function     : PWM_init
* Description : this function initializes only one pwm pin as output and sets high on it.
* Args         : enu_arg_pwm_channel
* return       : enu_pwm_error_status_t
*/
enu_pwm_error_status_t PWM_init_channel(enu_pwm_channels_t enu_arg_pwm_channel);



/*
* Function     : PWM_init
* Description : this function initializes all the pwm pins as output and sets high on them.
* Args         : enu_arg_pwm_channel
* return       : enu_pwm_error_status_t
*/
enu_pwm_error_status_t PWM_init_all_channels(void);

/*
* Function     : PWM_set_Duty
* Description : this function sets the desired duty cycle and the initial signal duration.
* Args         : enu_arg_pwm_channel
* return       : enu_pwm_error_status_t
*/
enu_pwm_error_status_t PWM_set_Duty(Uint16_t  u16_arg_signal_duration_ms
                                    Uint8_t   u8_arg_duty_cycle );



/*
* Function     : PWM_stop
* Description : this function provides 0 speed on the motor
* Args         : void
* return       : enu_pwm_error_status_t
*/
enu_pwm_error_status_t PWM_stop(void);
```

2.3.3.5. DCM

```
/*
* DCM_motorInint
*
* This function initialises/selects the desired pins for the two motors
*  using the passed configuration
```

```
* Parameters
*            [in] *DCM_a_ptrToConfig pointer to the configuration struct
* Return
*            [out] EN_DCM_ERROR_s error statue for DC motor.
*/
EN_DCM_ERROR_T DCM_motorInit(ST_DCM_config_t *DCM_a_ptrToConfig);


/*
* DCM_changeDiretion
*
* This function changes the motor direction
* Parameters
*            [in] *DCM_a_ptrToConfig pointer to the configuration struct
*            [in]  DCM_a_motorNum enum selection for the motor side.
* Return
*            [out] EN_DCM_ERROR_s error statue for DC motor.
*/
EN_DCM_ERROR_T DCM_changeDiretion(ST_DCM_config_t *DCM_a_ptrToConfig, EN_DCM_MOTORSIDE
DCM_a_motorNum);


/*
* Stop the motor
*
* Parameters
*            [void
* Return
*            void
*/
void DCM_vdStopDCM(void);


/*
*
* This function set the desired PWM.
* Parameters
*            [in] DCM_a_dutyCycleValuevalue set the PWM.
* Return
*            [out] EN_DCM_ERROR_s error statue for DC motor.
*/
EN_TIMER_ERROR_T DCM_setDutyCycle(Uchart8_t DCM_a_dutyCycleValue);

/*
* This function rotates the desired motor.
* Parameters
*            [in] DCM_a_motorNum enum selection for the motor side.
*            [in] DCM_a_dutyCycleValuevalue set the PWM.
* Return
*            [out] EN_DCM_ERROR_s error statue for DC motor.
*/
```

```
EN_TIMER_ERROR_T DCM_rotateDCM(EN_DCM_MOTORSIDE DCM_a_motorNum,Uchart8_t
DCM_a_rotateSpeed );

/*
* Move the robot forward
*
* Parameters
*             [void]
* Return
*             [void]
*/
void DCM_MoveForward(void);

/*
* Move the robot backward
*
* Parameters
*             [void]
* Return
*             [void]
*/
void DCM_MoveBackward(void);
```

2.3.3.6 LED

```
| Initializing the desired led_pin as output
|Parameters
|                 none
| Return
|    An enu_led_error_t value indicating the success or failure of
|                 the operation (LED_OK if the operation succeeded, LED_ERROR
|        otherwise)
|
enu_led_error_t LED_init(void);

| Turn the LED on
|Parameters
|                 [in] uint8_t_led_id
| Return
|    An enu_led_error_t value indicating the success or failure of
|                 the operation (LED_OK if the operation succeeded, LED_ERROR
|        otherwise)
|
enu_led_error_t LED_on(uint8_t uint8_led_id);

| Turn the LED off
|Parameters
|                 [in] uint8_t_led_id
| Return
|    An enu_led_error_t value indicating the success or failure of
|                 the operation (LED_OK if the operation succeeded, LED_ERROR
```

```
|          otherwise)
|
enu_led_error_t LED_off(uint8_t uint8_led_id);

| Toggle the LED
|Parameters
|                  [in] uint8_t_led_id
| Return
|    An enu_led_error_t value indicating the success or failure of
|                  the operation (LED_OK if the operation succeeded, LED_ERROR
|          otherwise)
|
enu_led_error_t LED_toggle(uint8_t uint8_led_id);
```

2.3.4. Application_APIs

2.3.4.1.APP

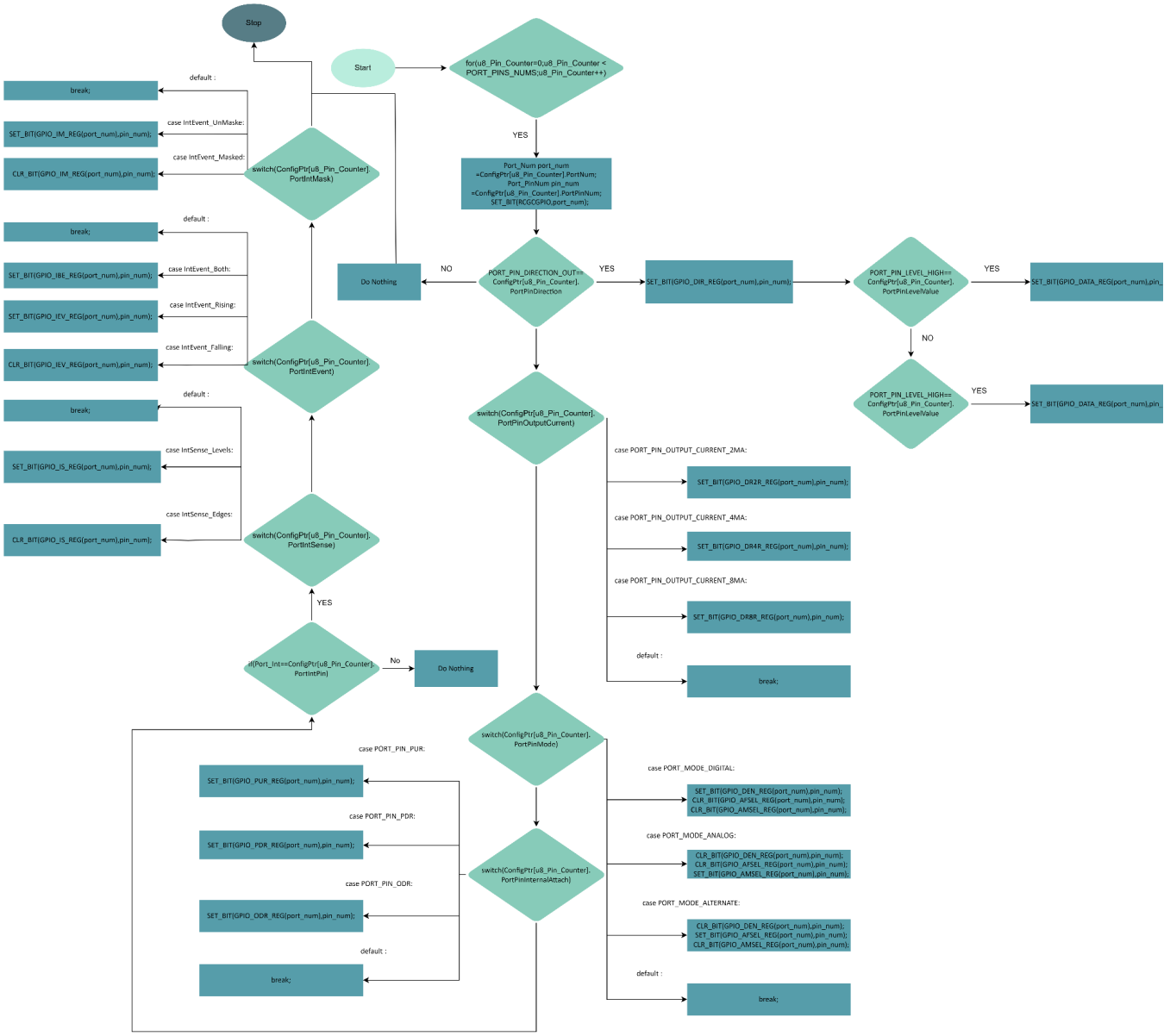
```
/*
* Function    : app_init
* Description: this function initialize all required drivers in hal layer
* Args        : void
* return      : void
*/

void app_init(void);

/*
* Function    : app_main
* Description: this function contain all app logic, must called in super loop
* Args        : void
* return      : void
*/

void app_main(void);
```

3.Low Level Design

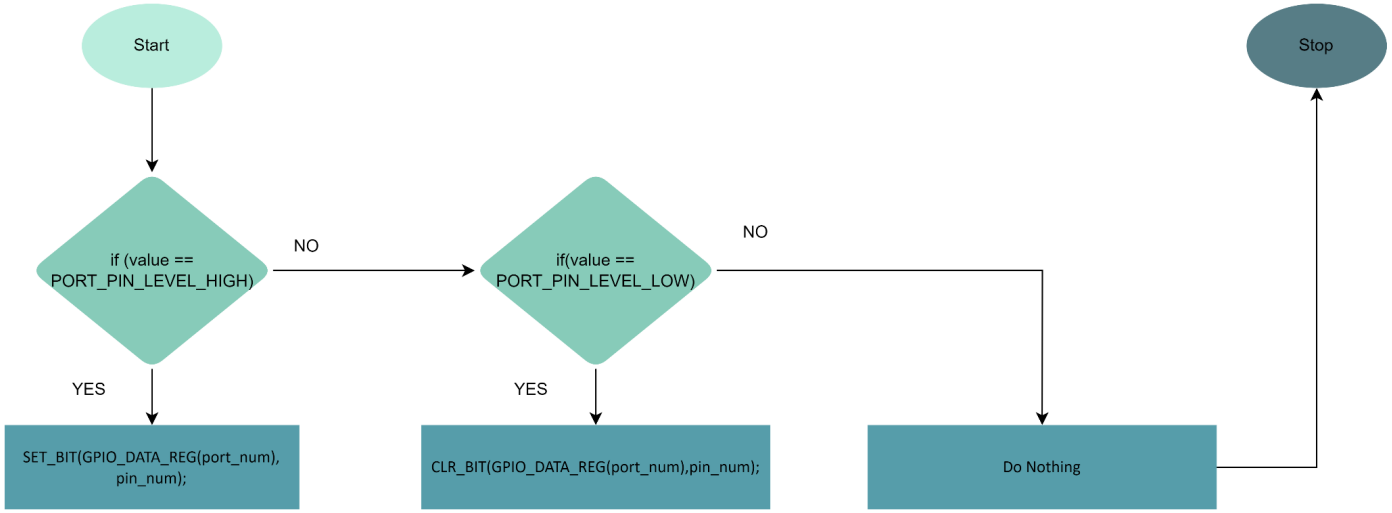
3.1. MCAL

3.1.1. GPIO

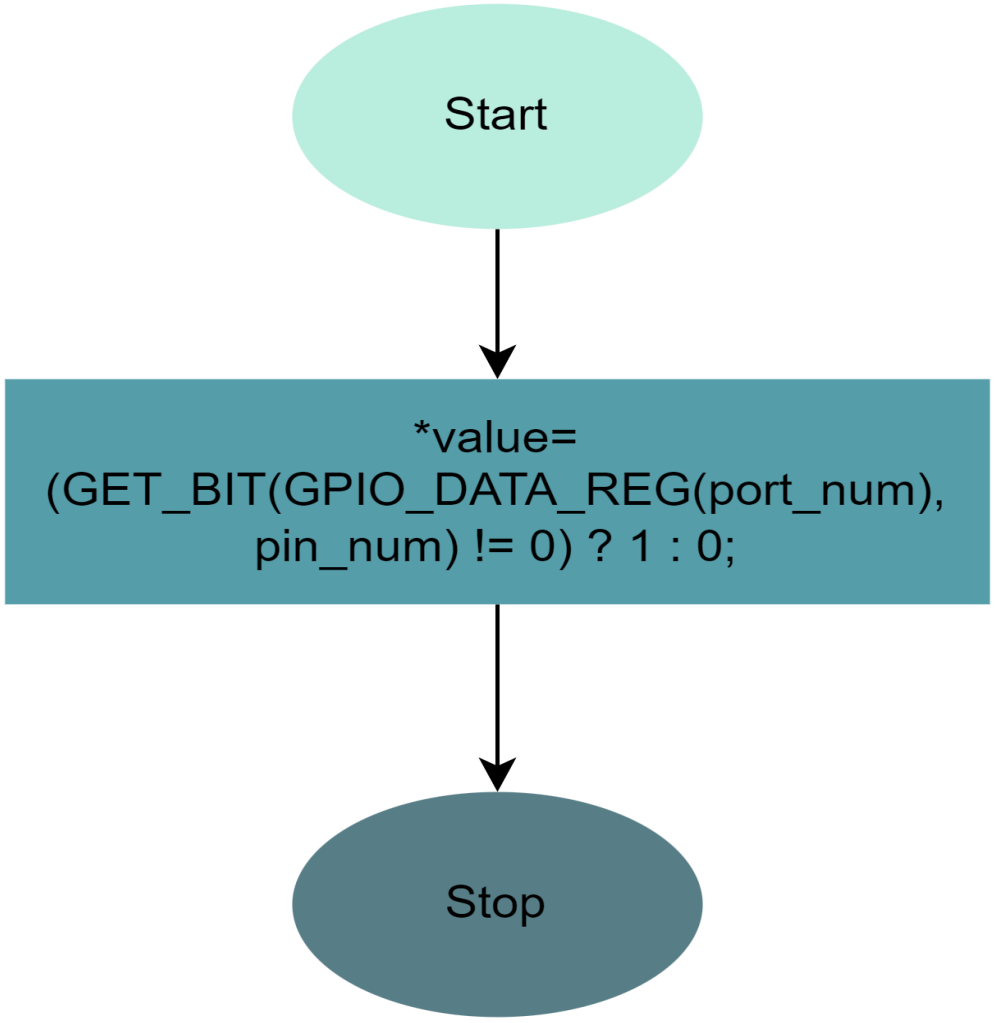
3.1.1.1. GPIO_init



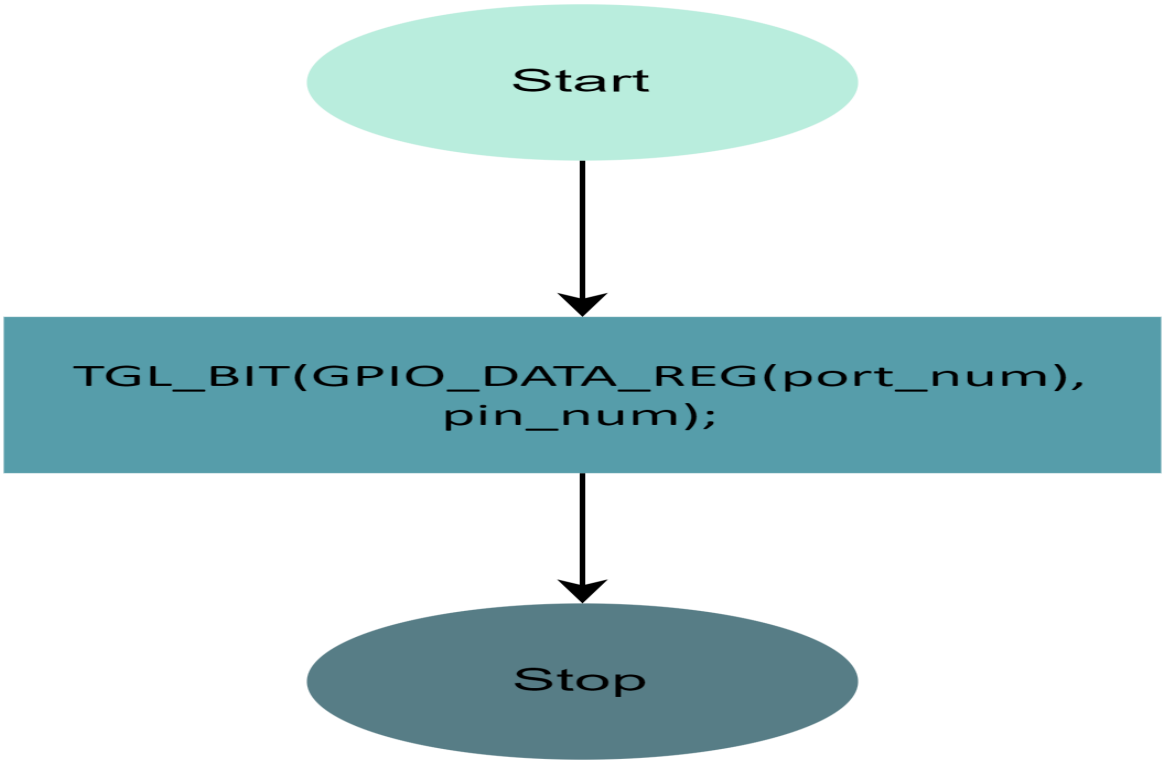
3.1.1.2. DIO_writepinn



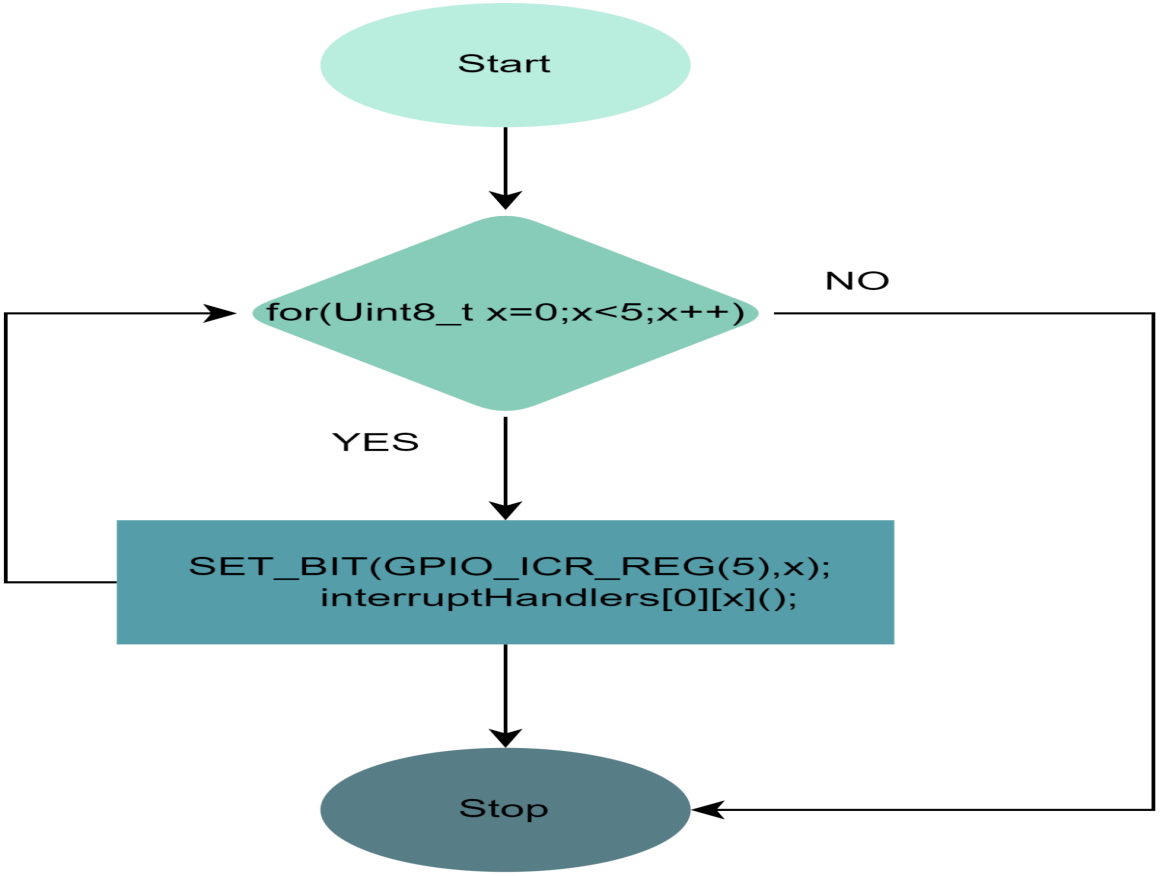
3.1.1.3. DIO_readpin



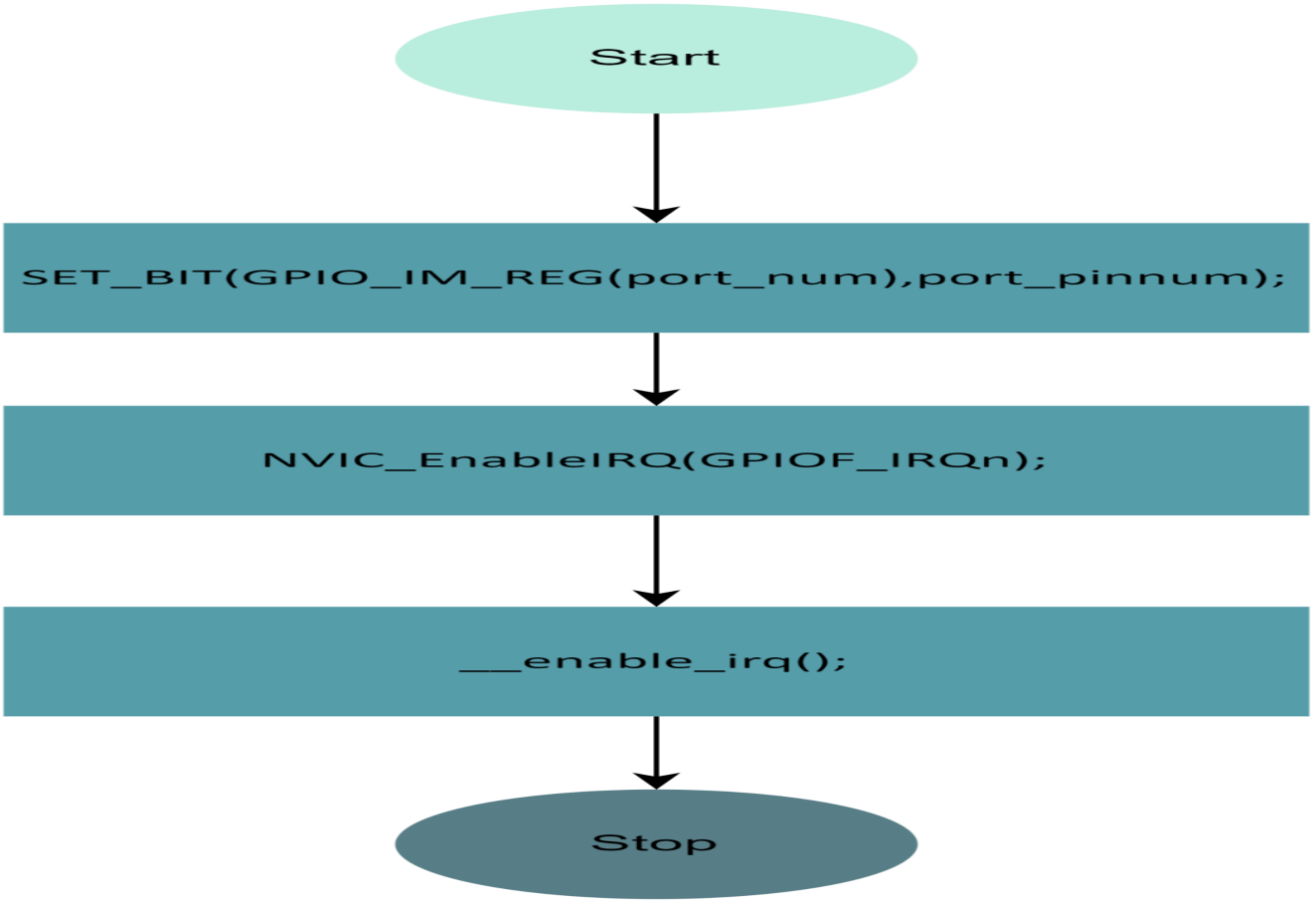
3.1.1.4. DIO_togglepin



3.1.1.5. DIO_handler

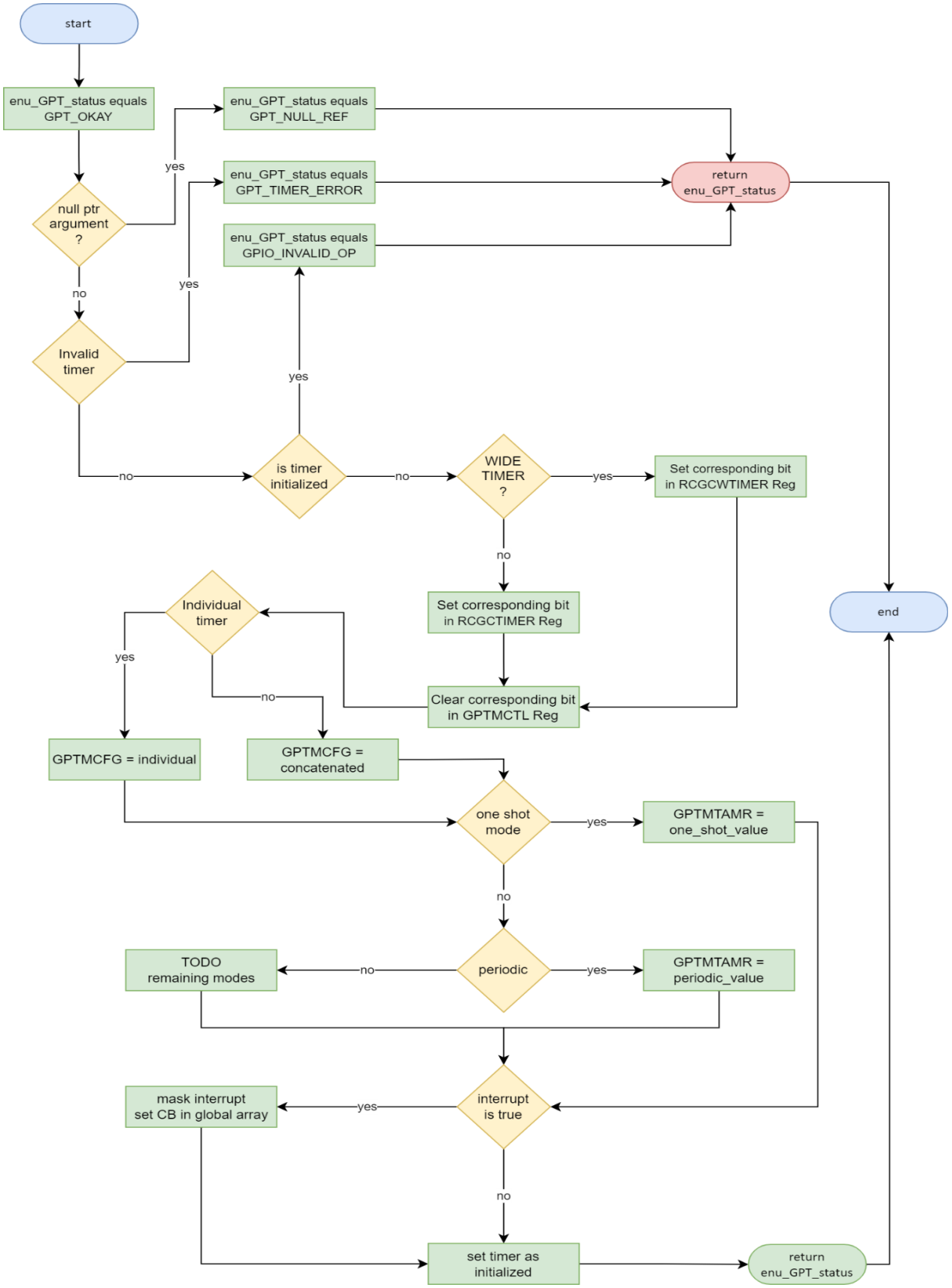


3.1.1.6. DIO_enable_int

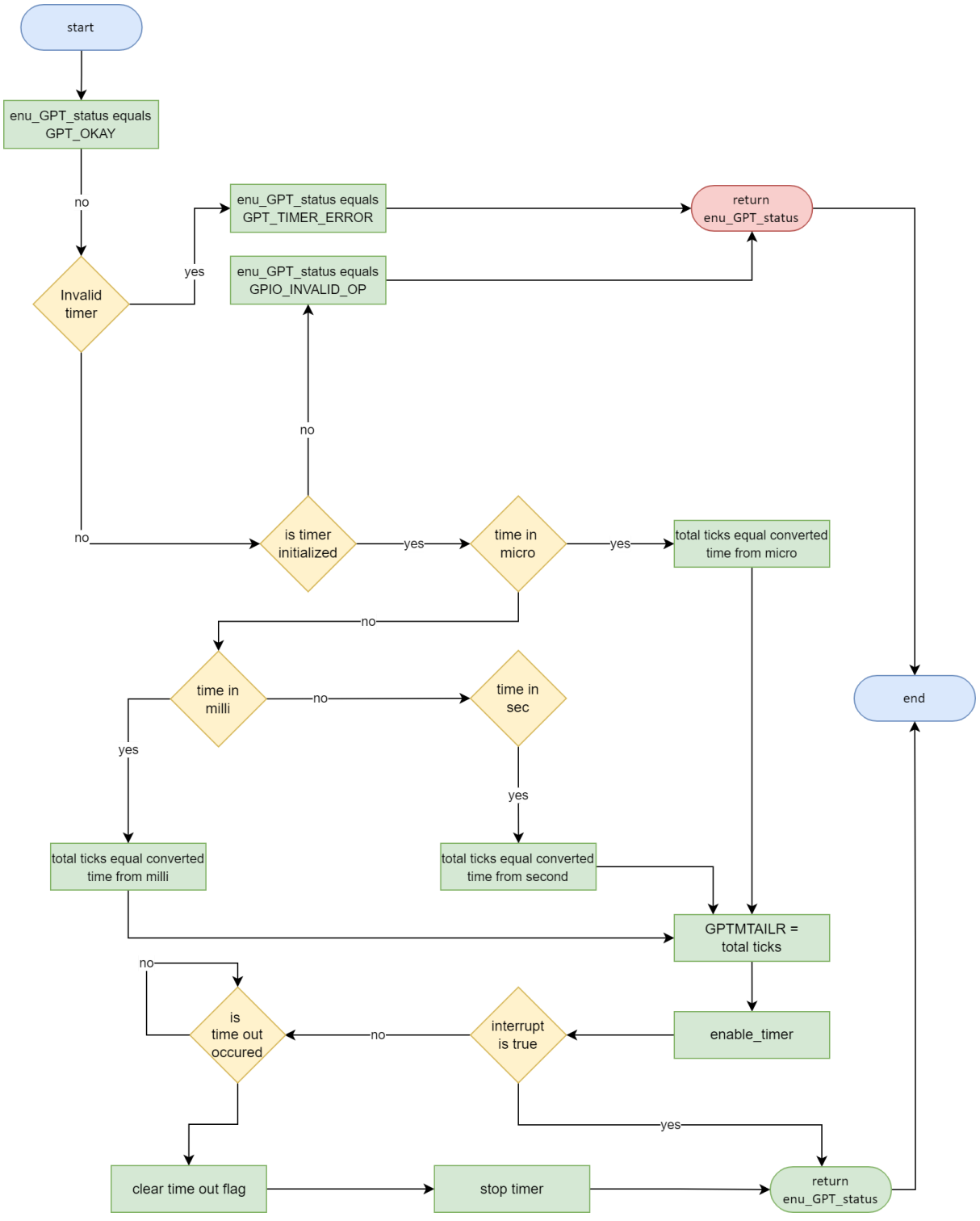


3.1.2. GPT

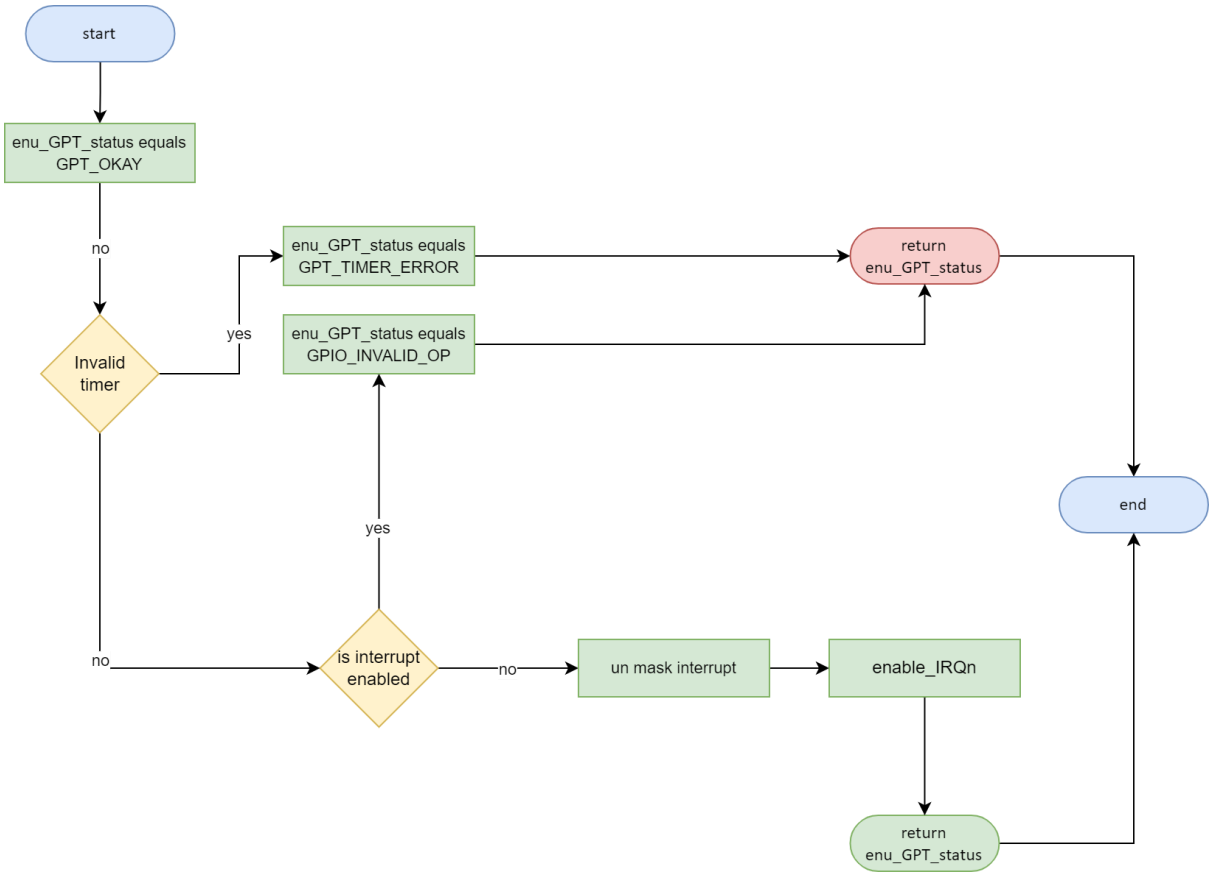
3.1.2.1 GPT_init



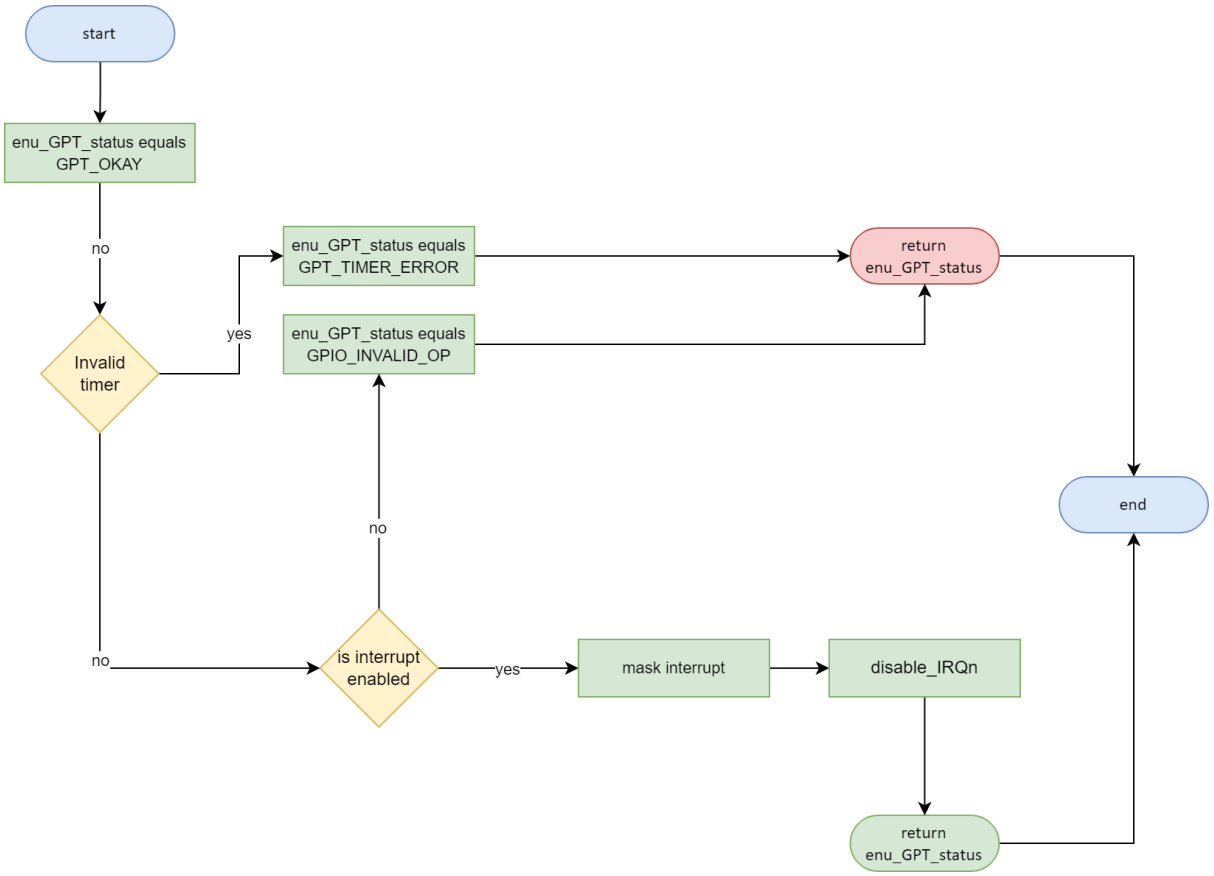
3.1.2.2 GPT_start_timer



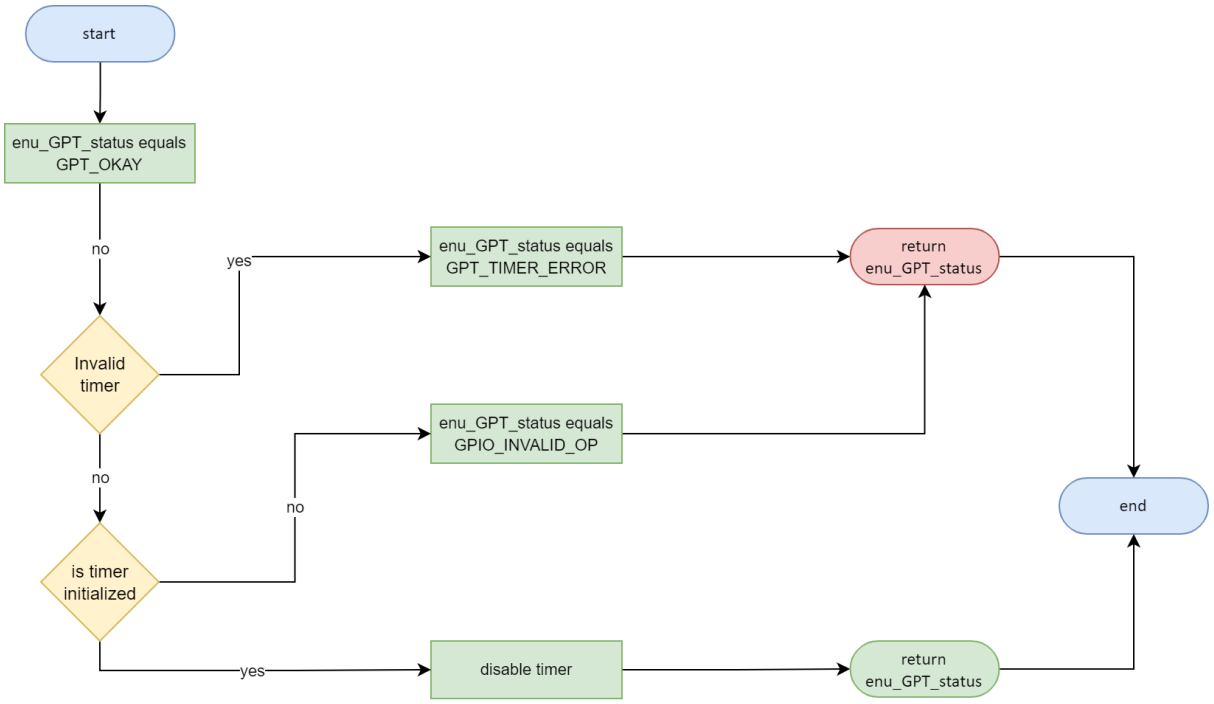
3.1.2.3 GPT_enable_interrupt



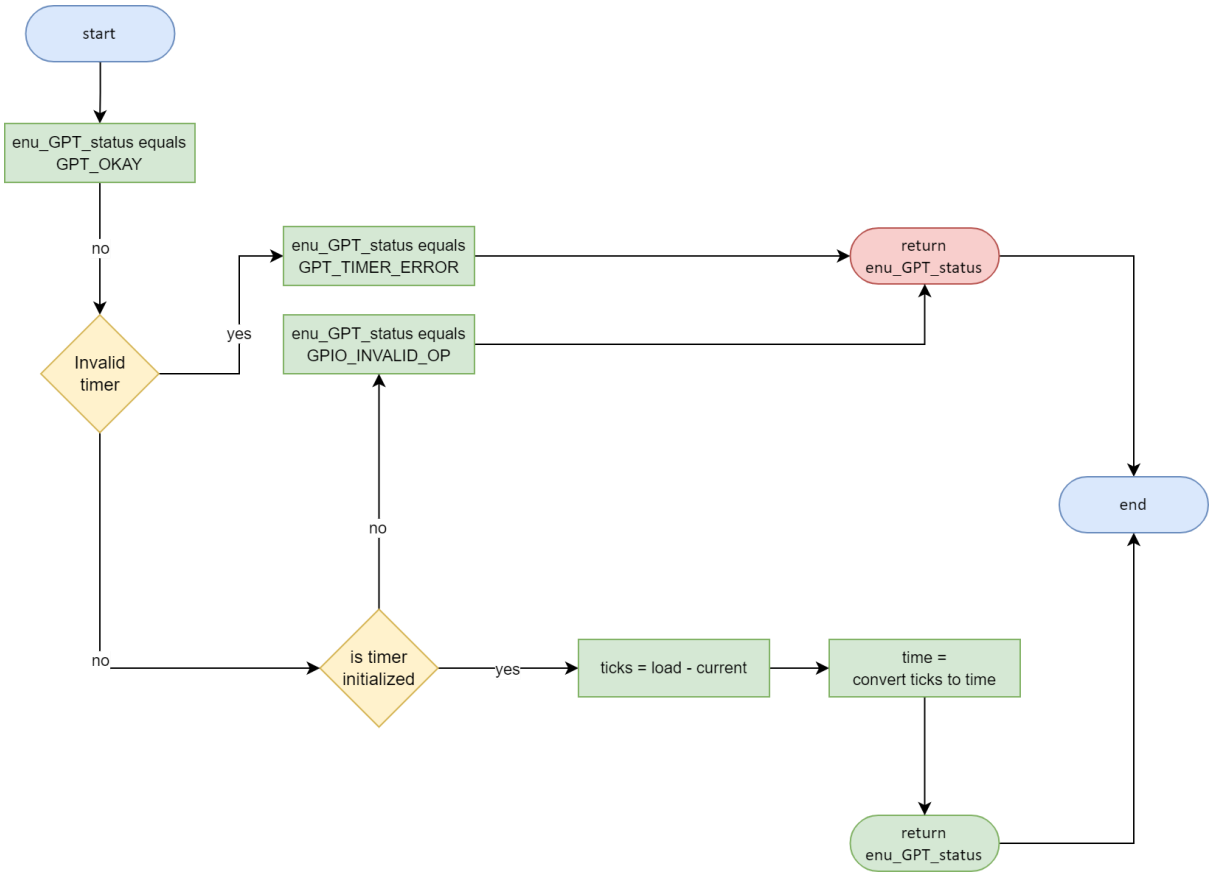
3.1.2.4 GPT_disable_interrupt



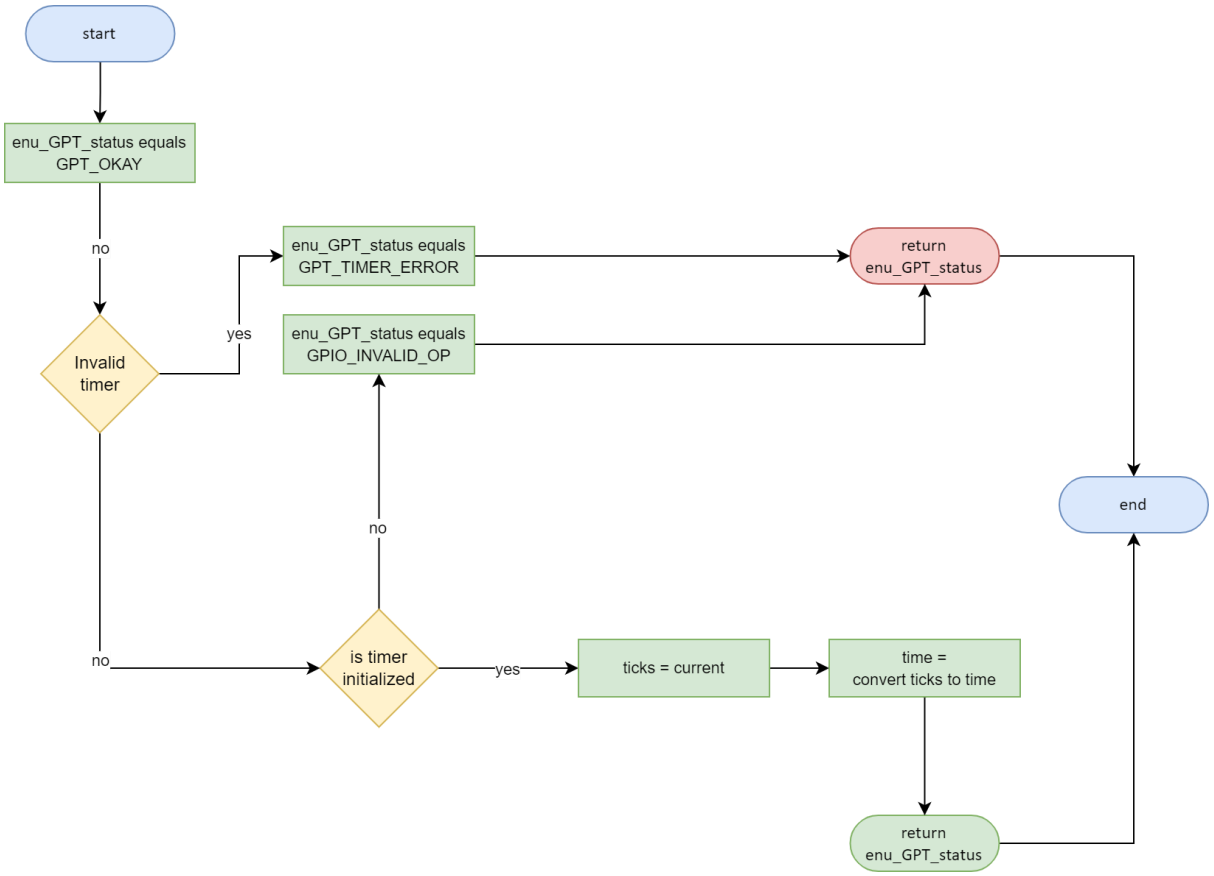
3.1.2.5 GPT_stop_timer



3.1.2.6 GPT_get_elapsed_time



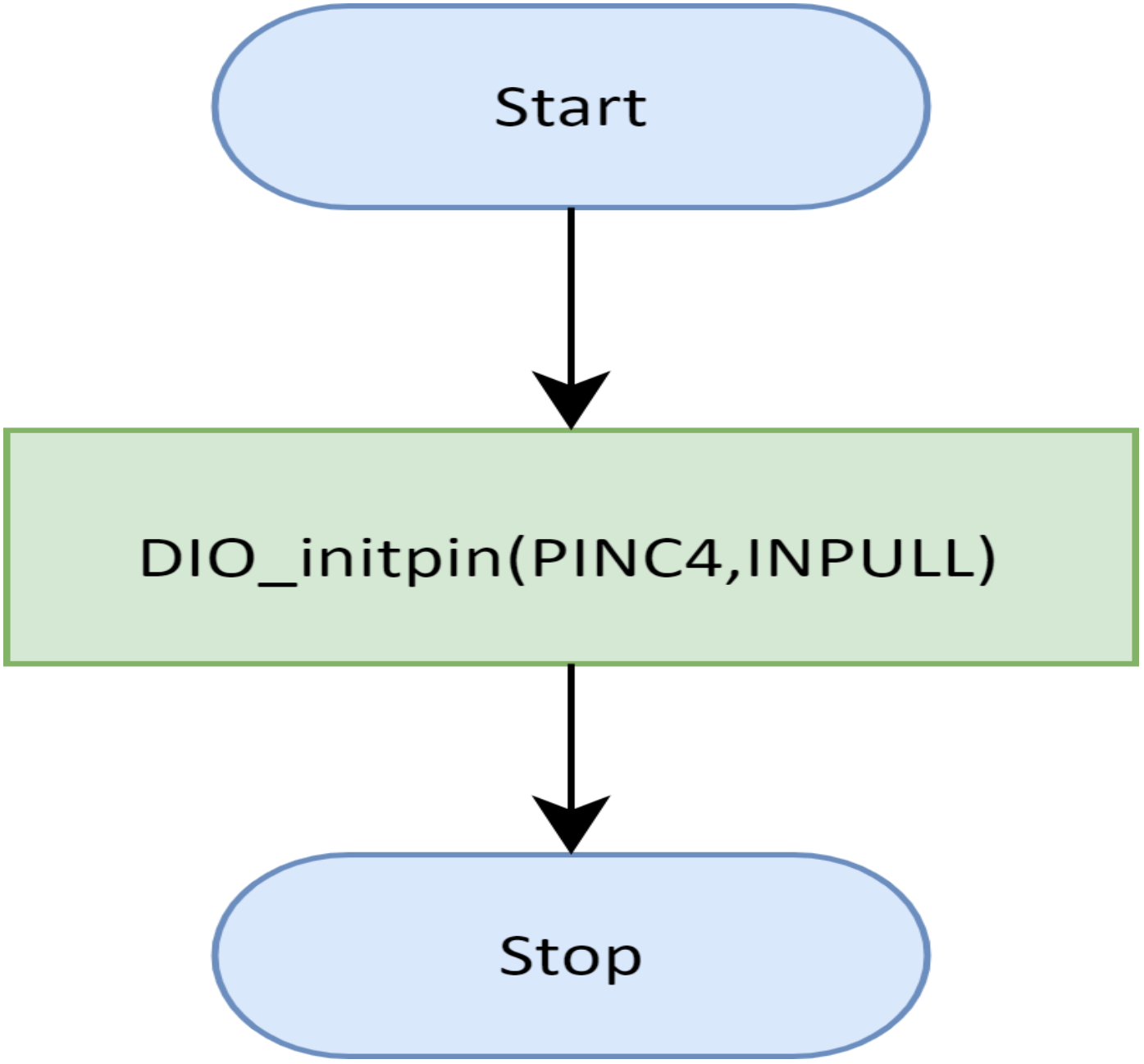
3.1.2.7 GPT_get_remaining_time



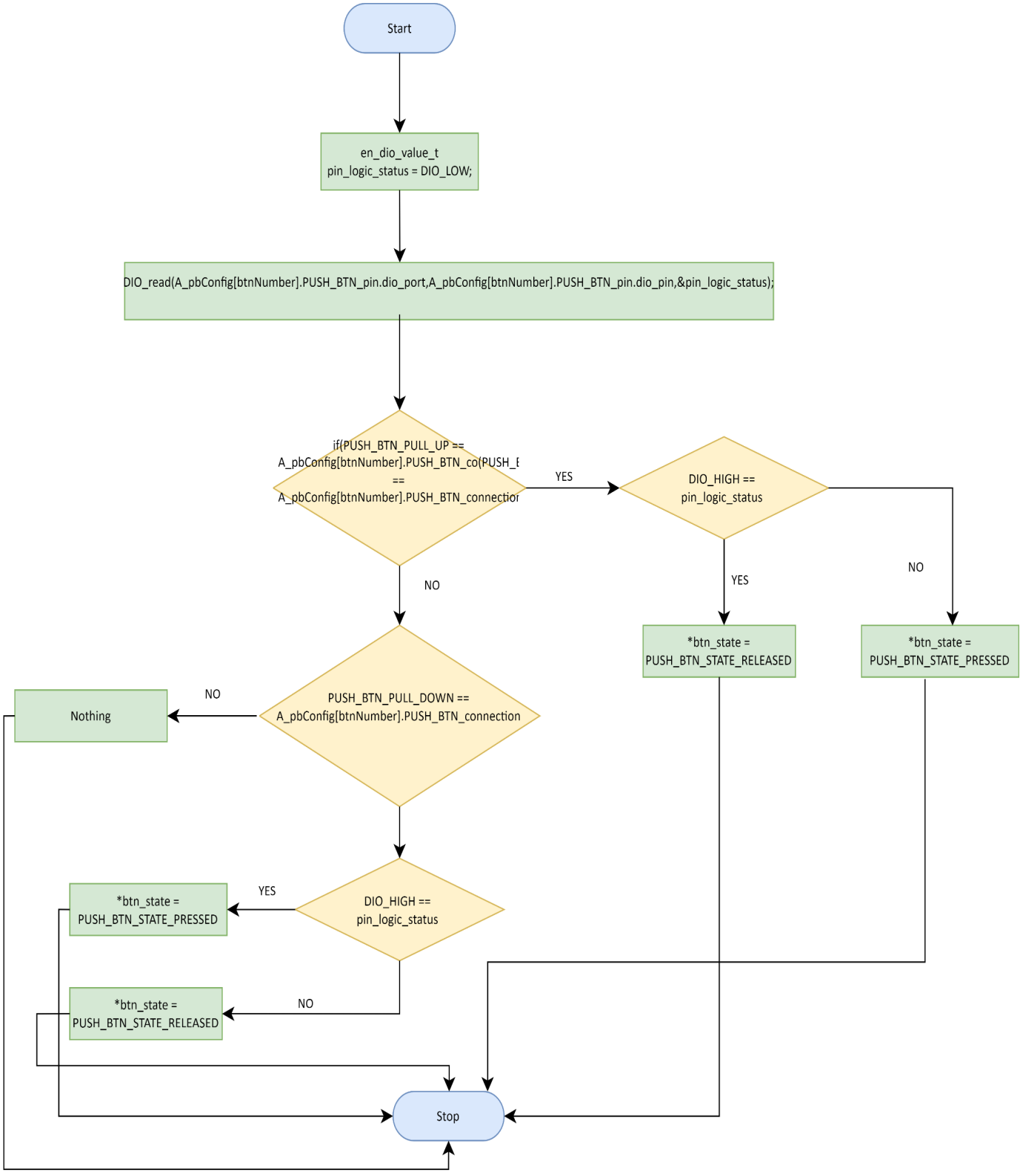
3.2. ECUAL

3.3.1. Push-BUTTON

3.3.1.1 PUSH_BTN_intialize

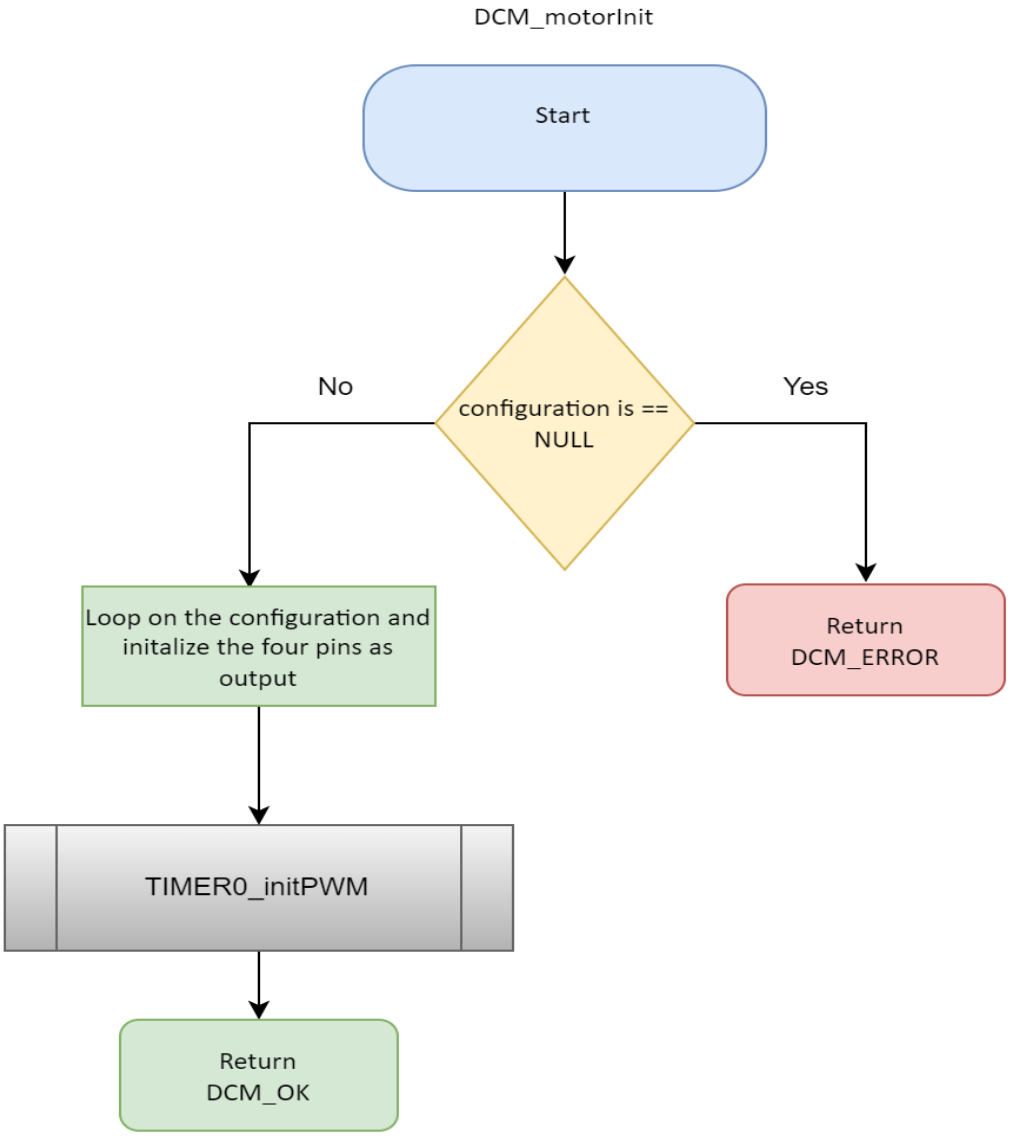


3.3.1.2 PUSH_BTN_read_state

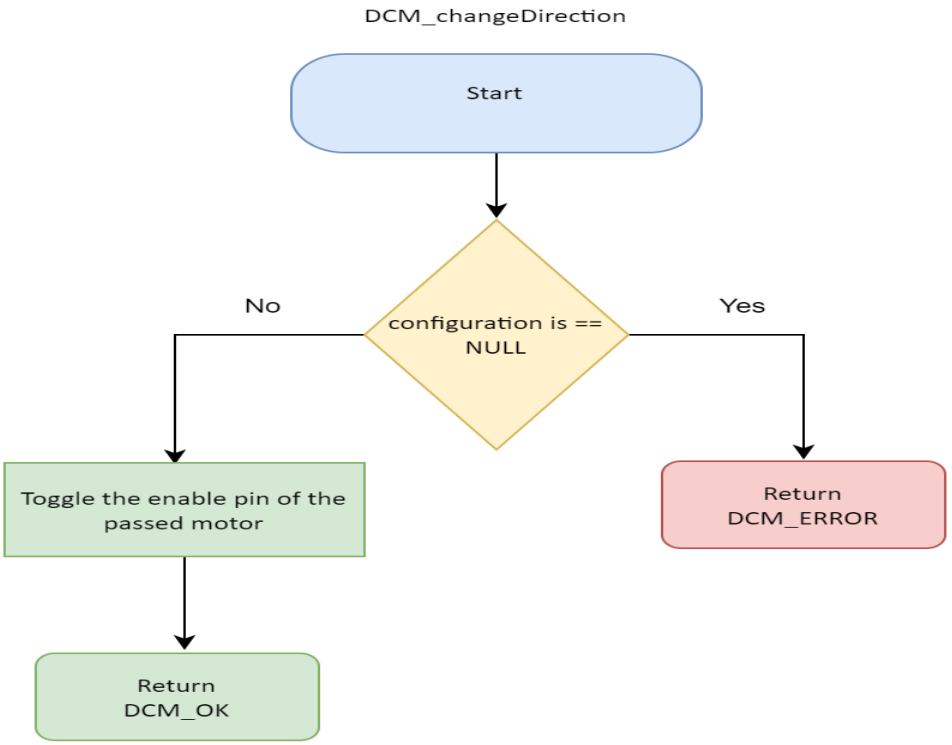


3.3.2. DCM

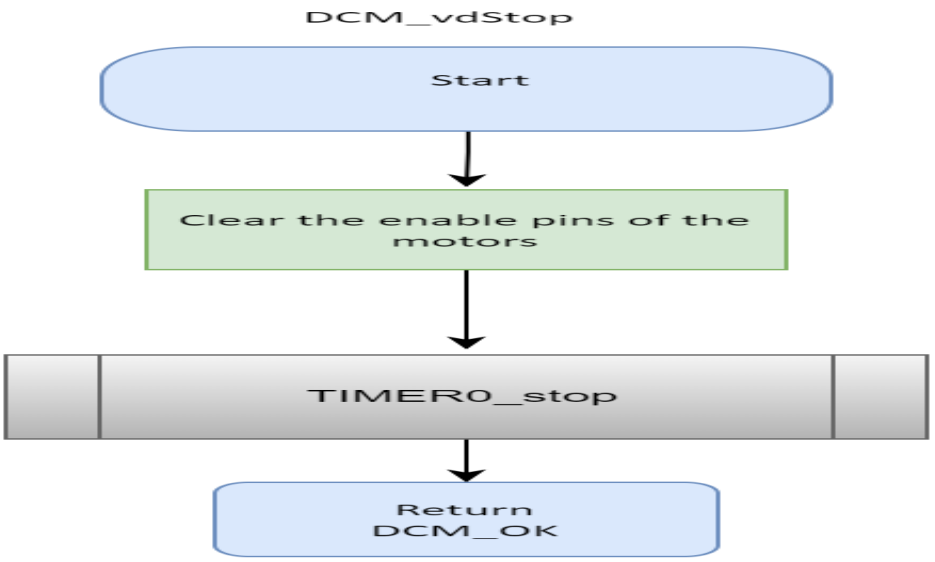
3.3.2.1.DCM_motorInit



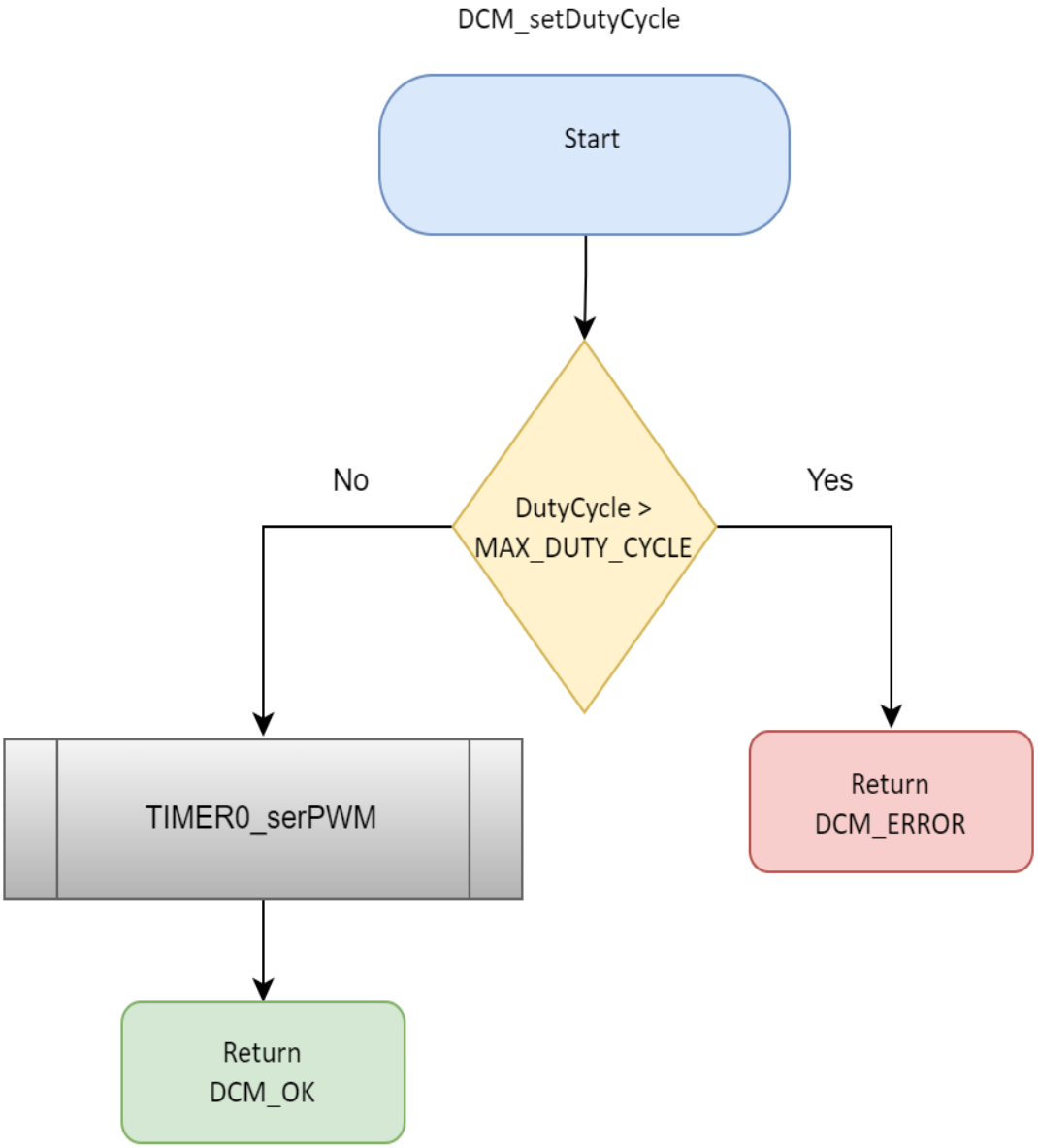
3.3.2.2.DCM_changeDirection



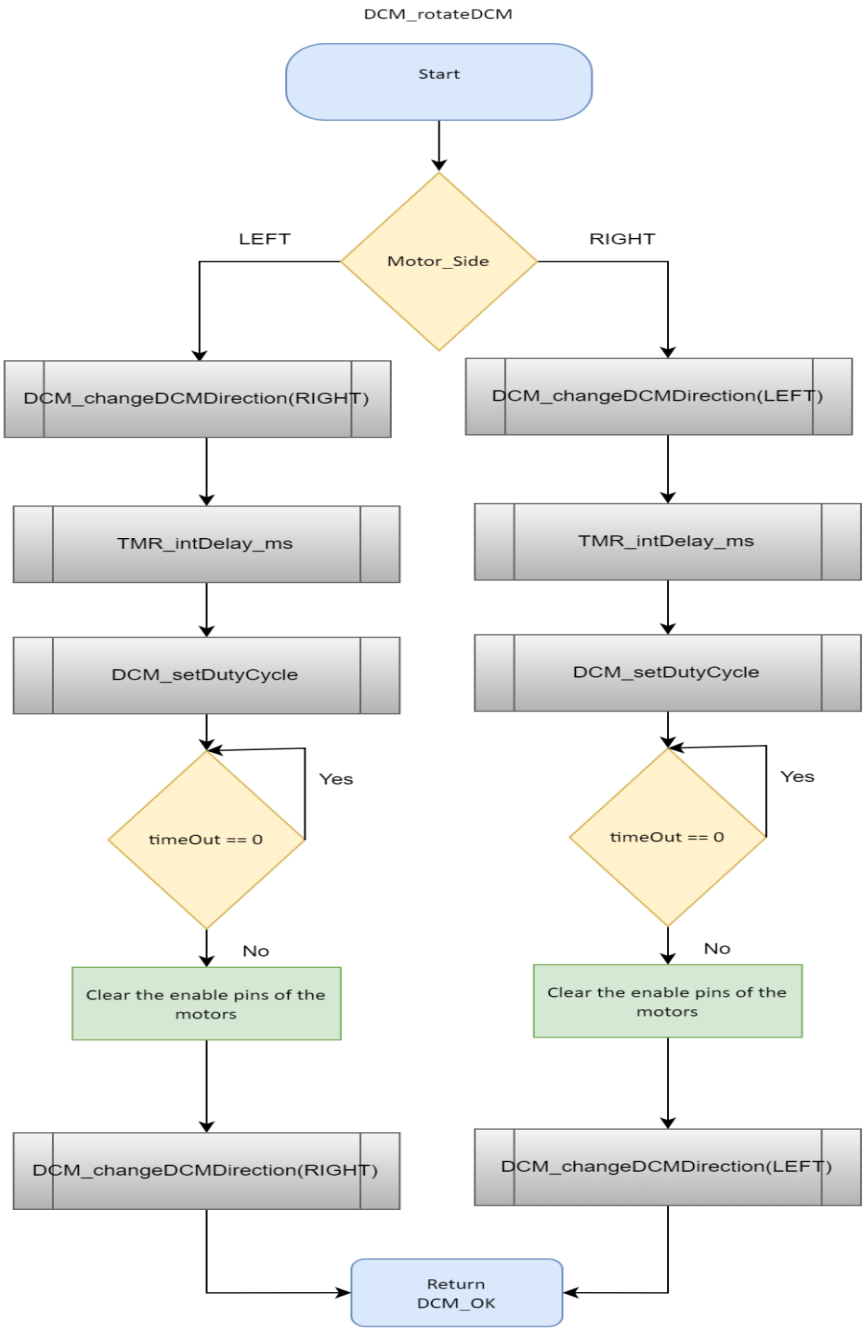
3.3.2.3.DCM_vdStop



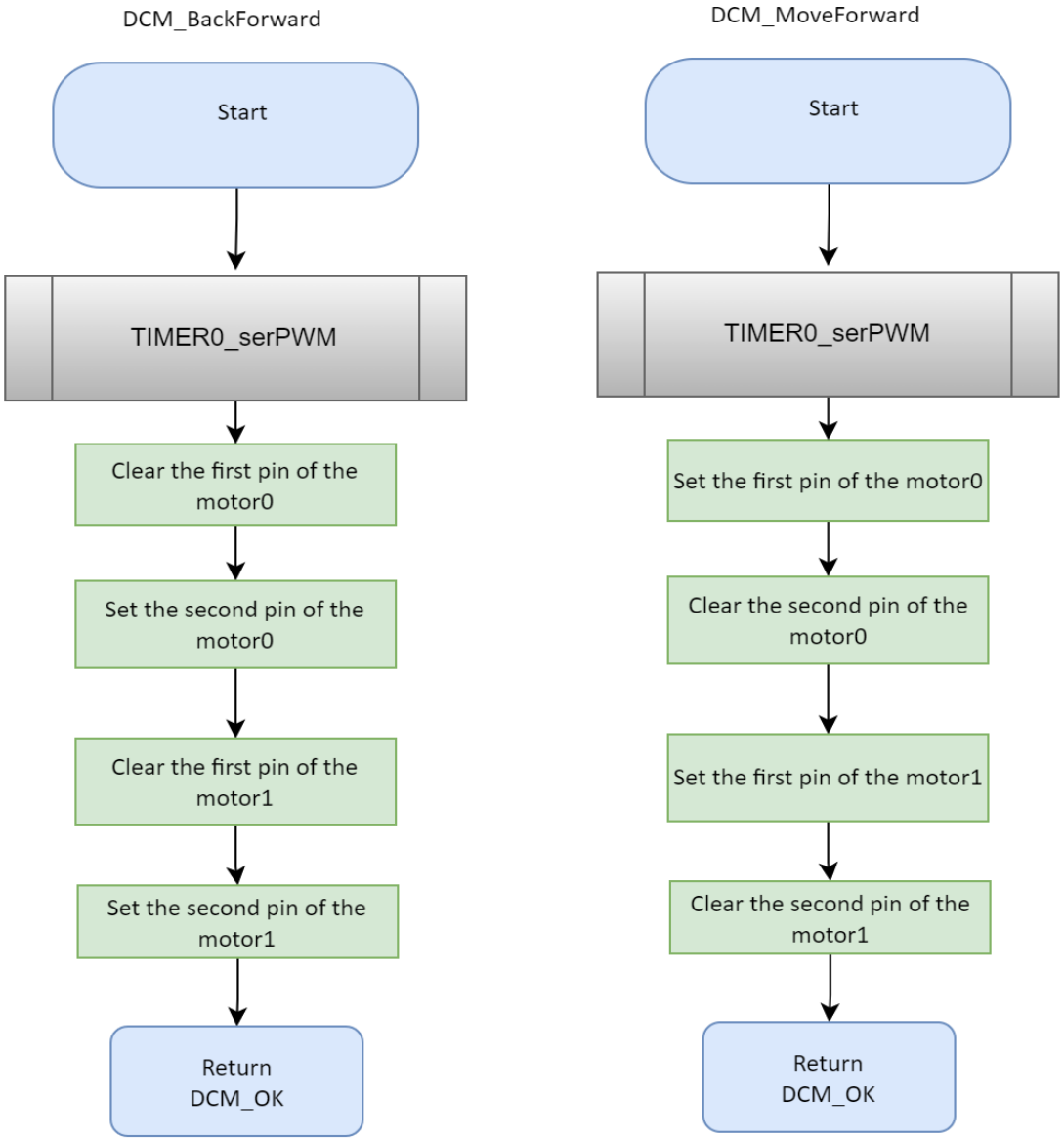
3.3.2.4.DCM_setDutyCycle



3.3.2.5.DCM_rotate

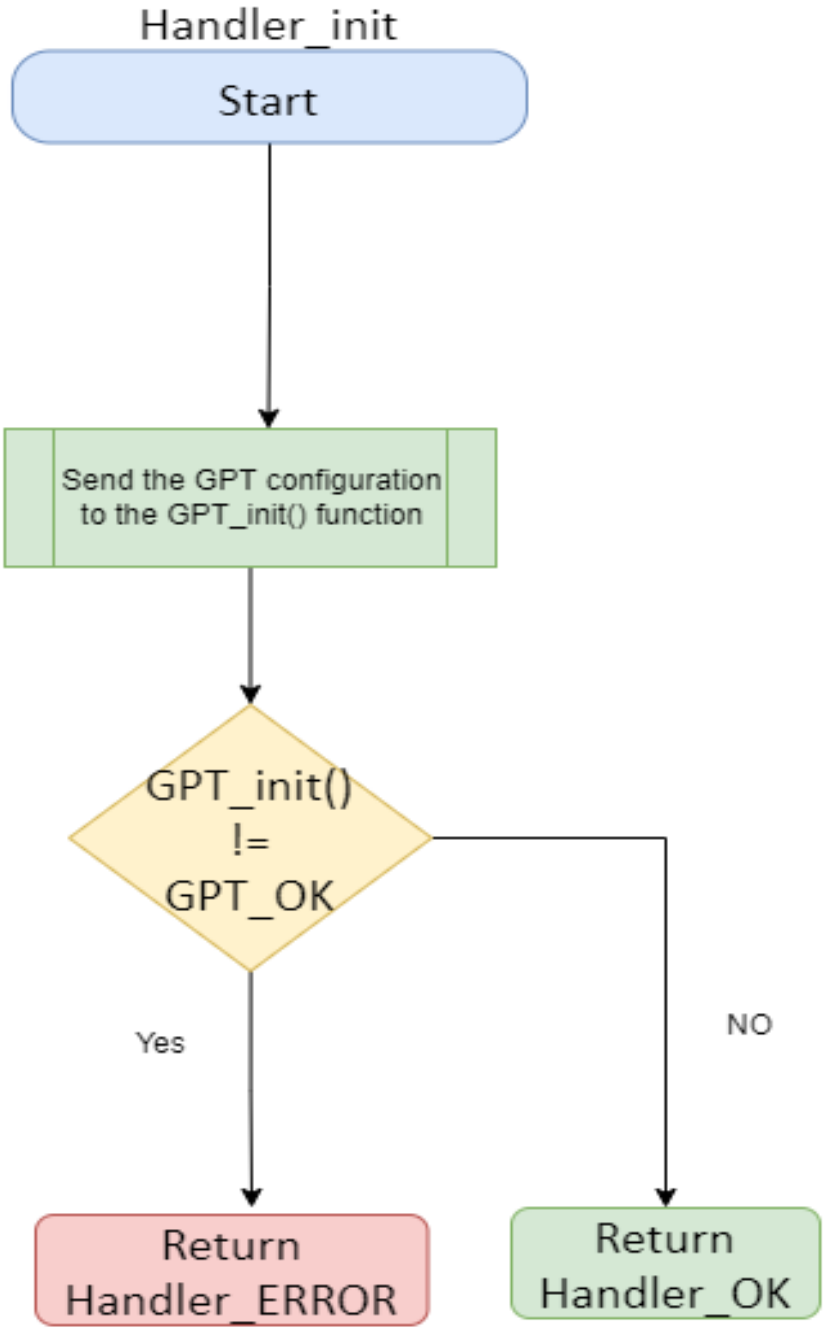


3.3.2.6. DCM_MoveForward / DCM_MoveBackward

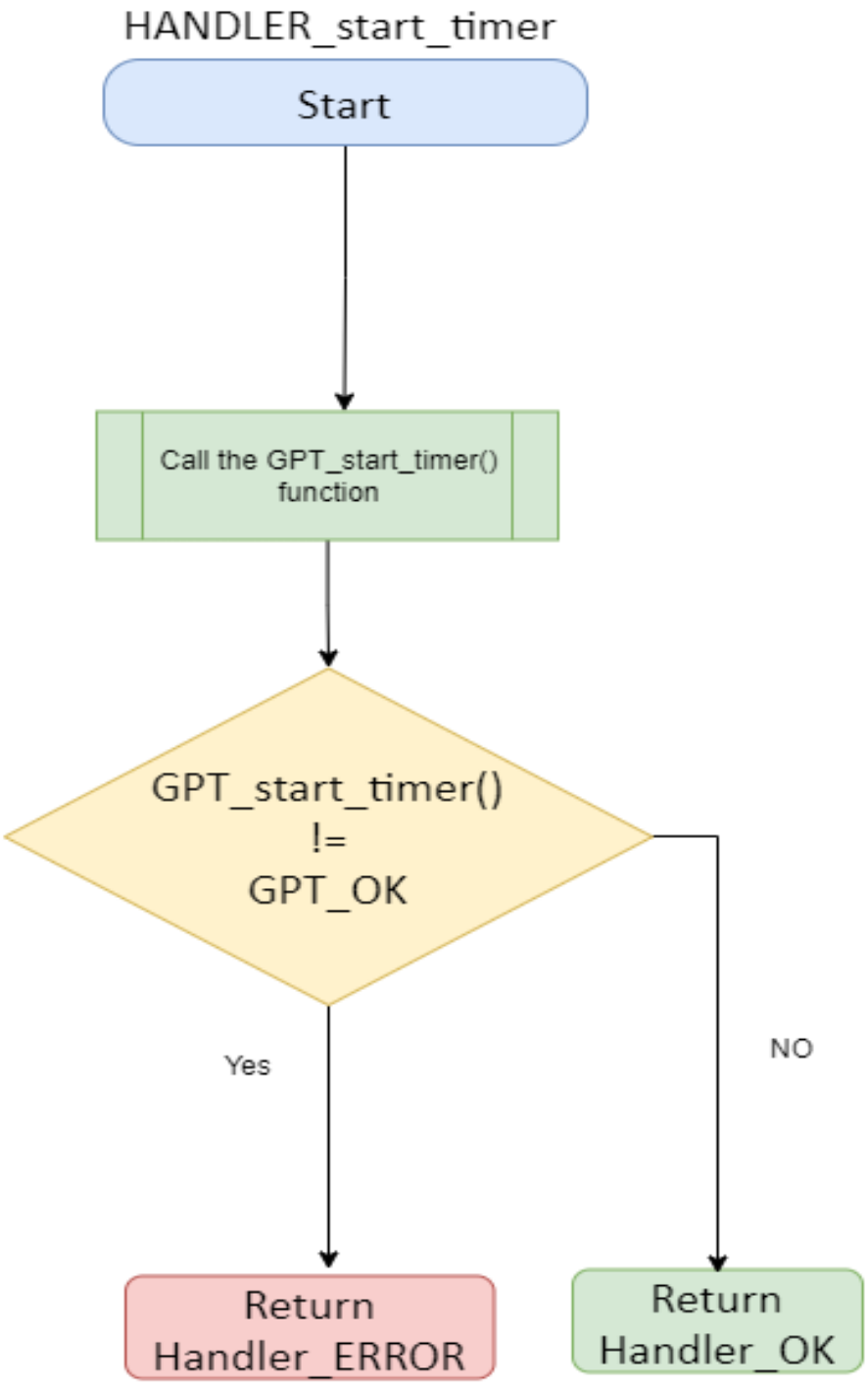


3.3.3. T_Handler

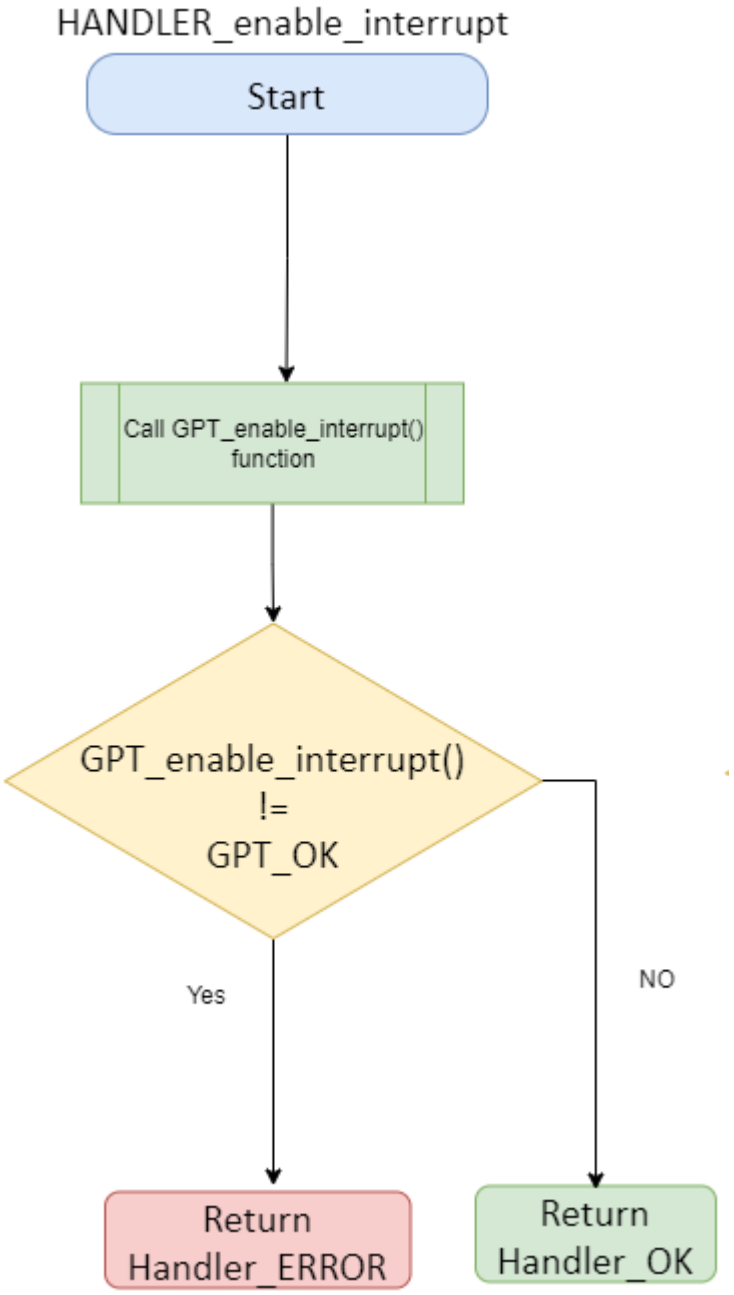
3.3.3.1 HANDLER_init



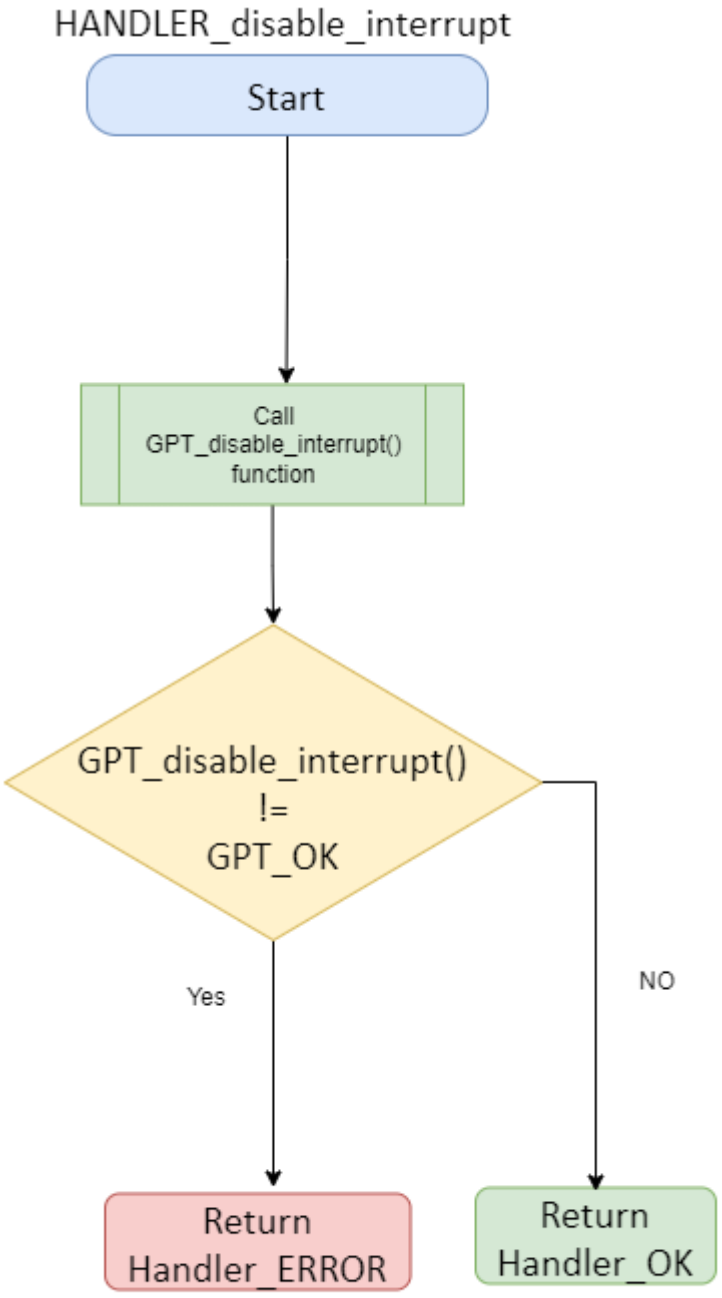
3.3.3.2 HANDLER_start_timer



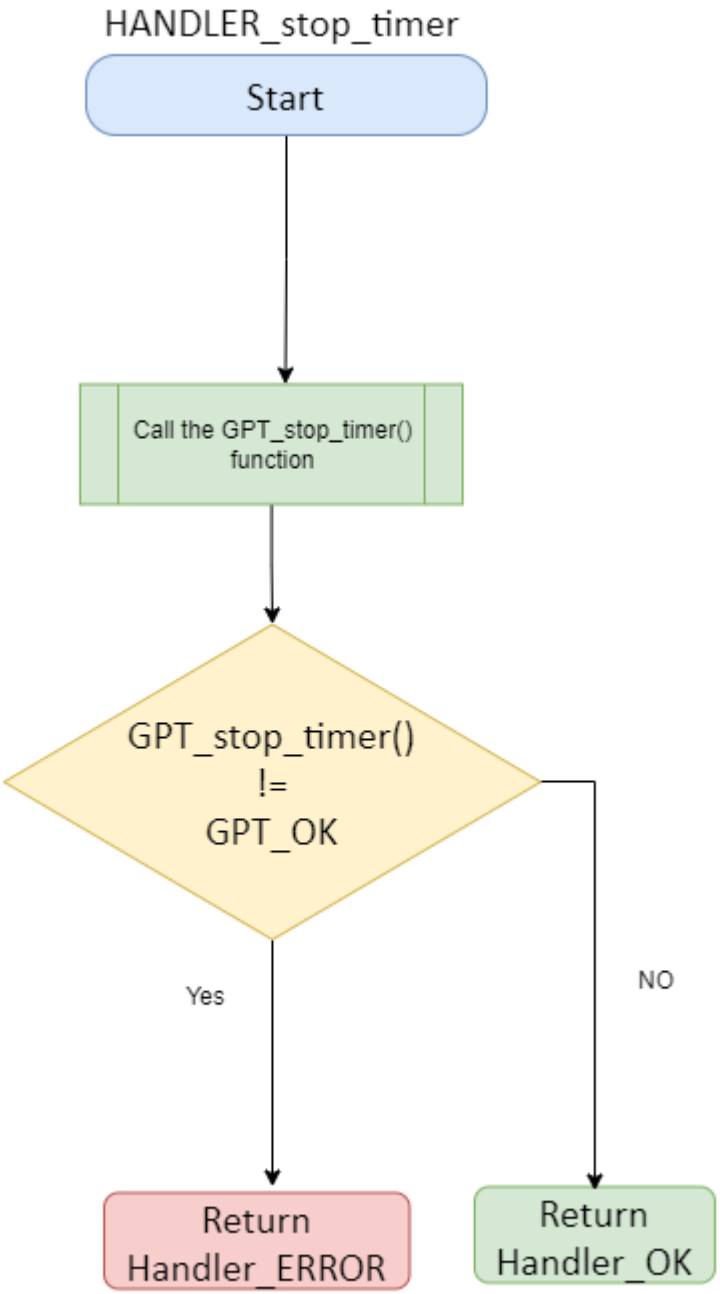
3.3.3.3 HANDLER_enable_interrupt



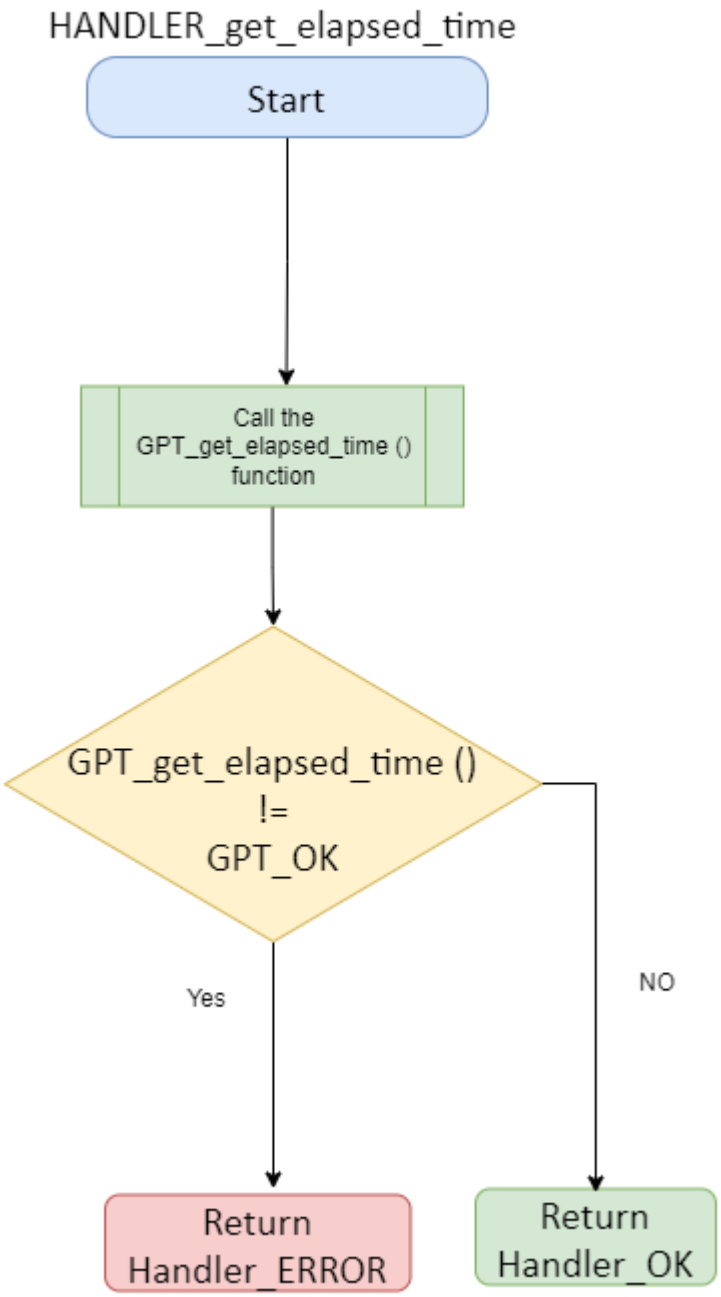
3.3.3.4 HANDLER_disable_interrupt



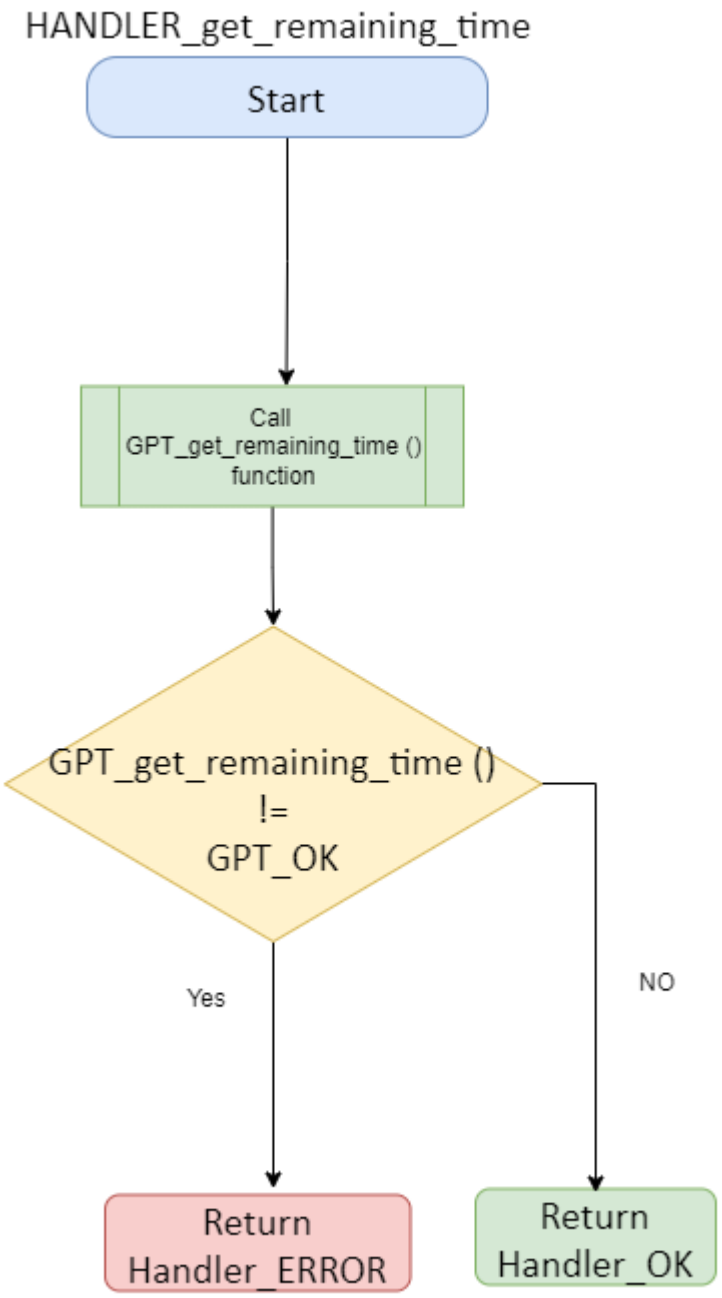
3.3.3.5 HANDLER_stop_timer



3.3.3.6 HANDLER_get_elapsed_time

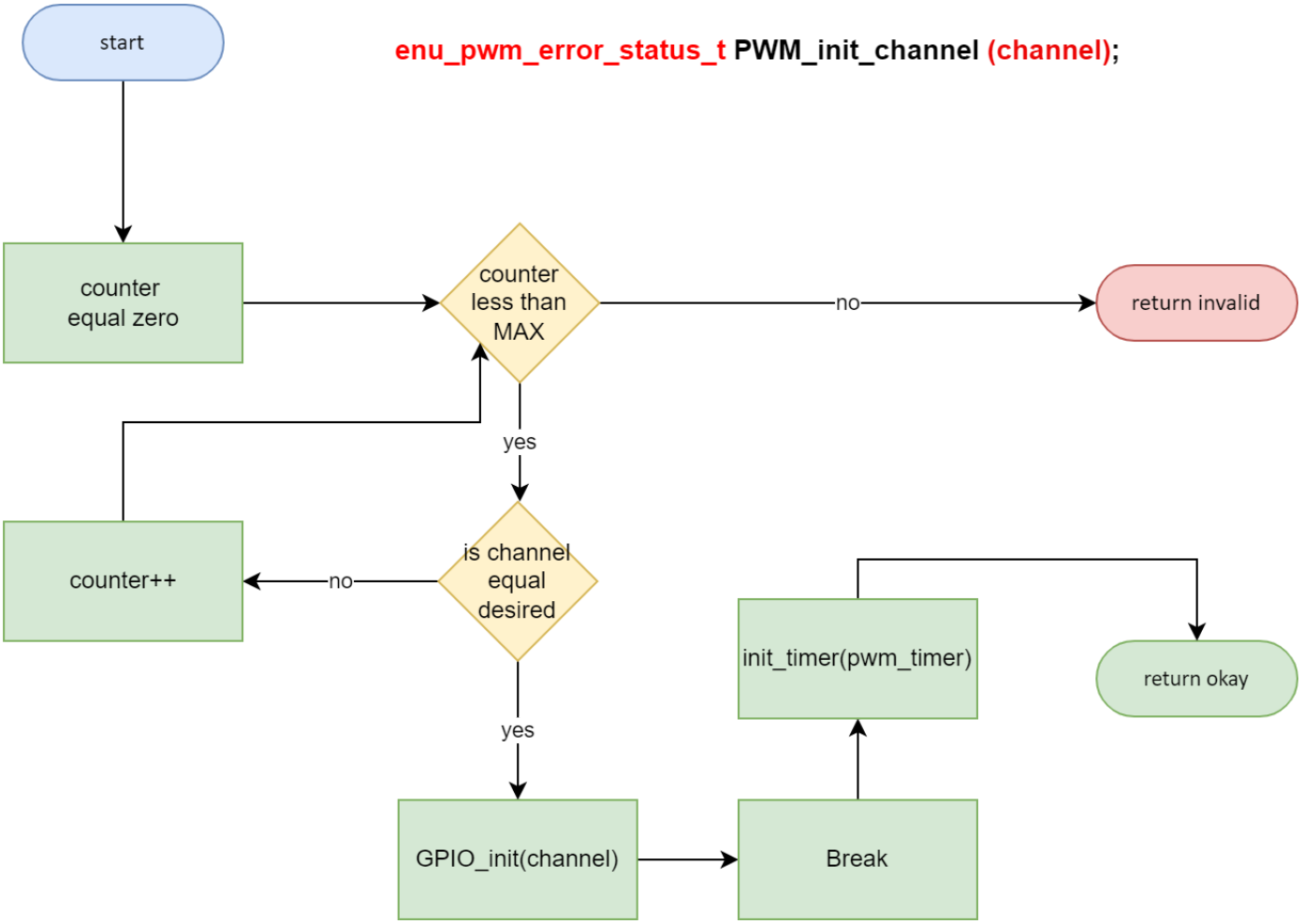


3.3.3.7 HANDLER_get_remaining_time

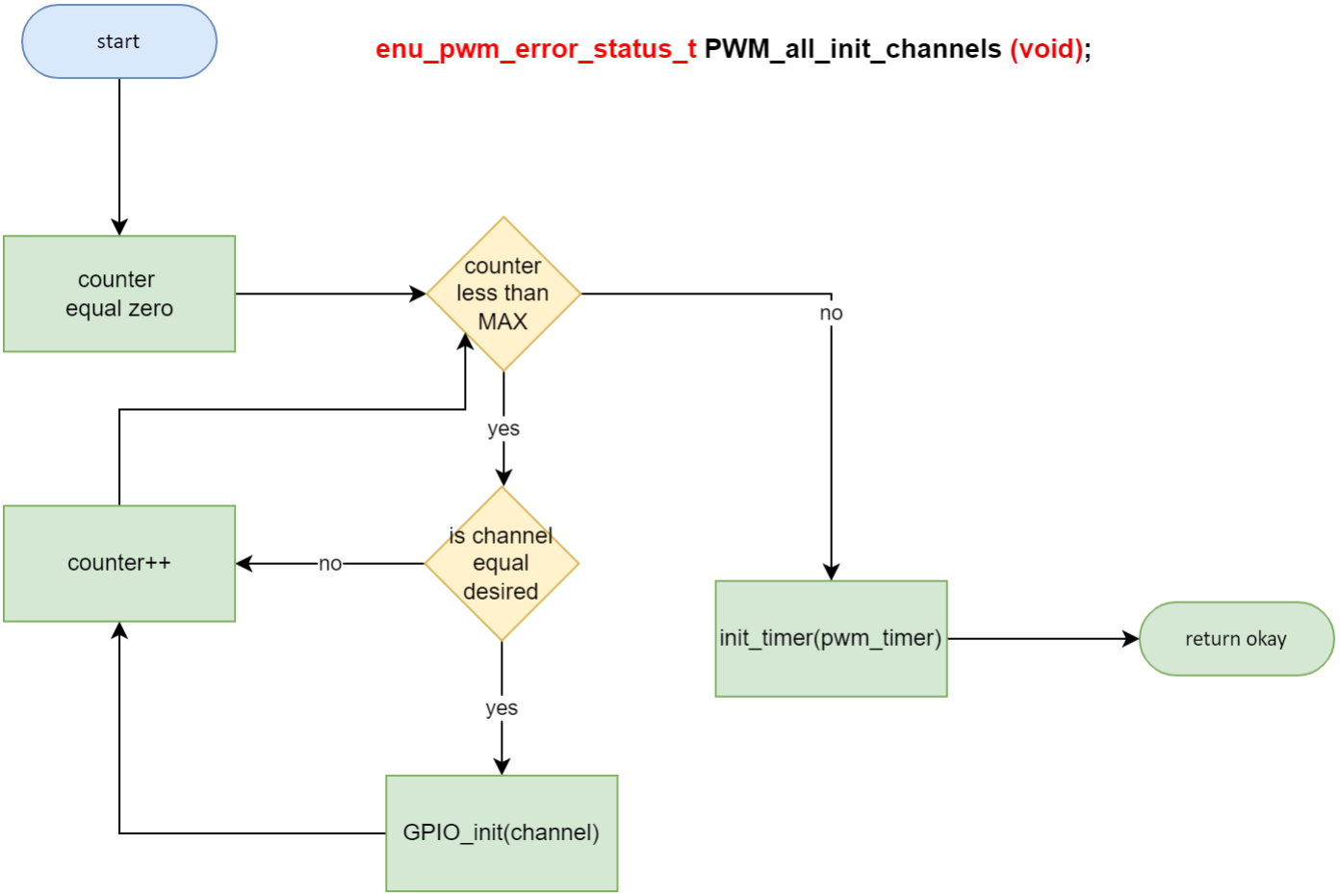


3.3.4. PWM

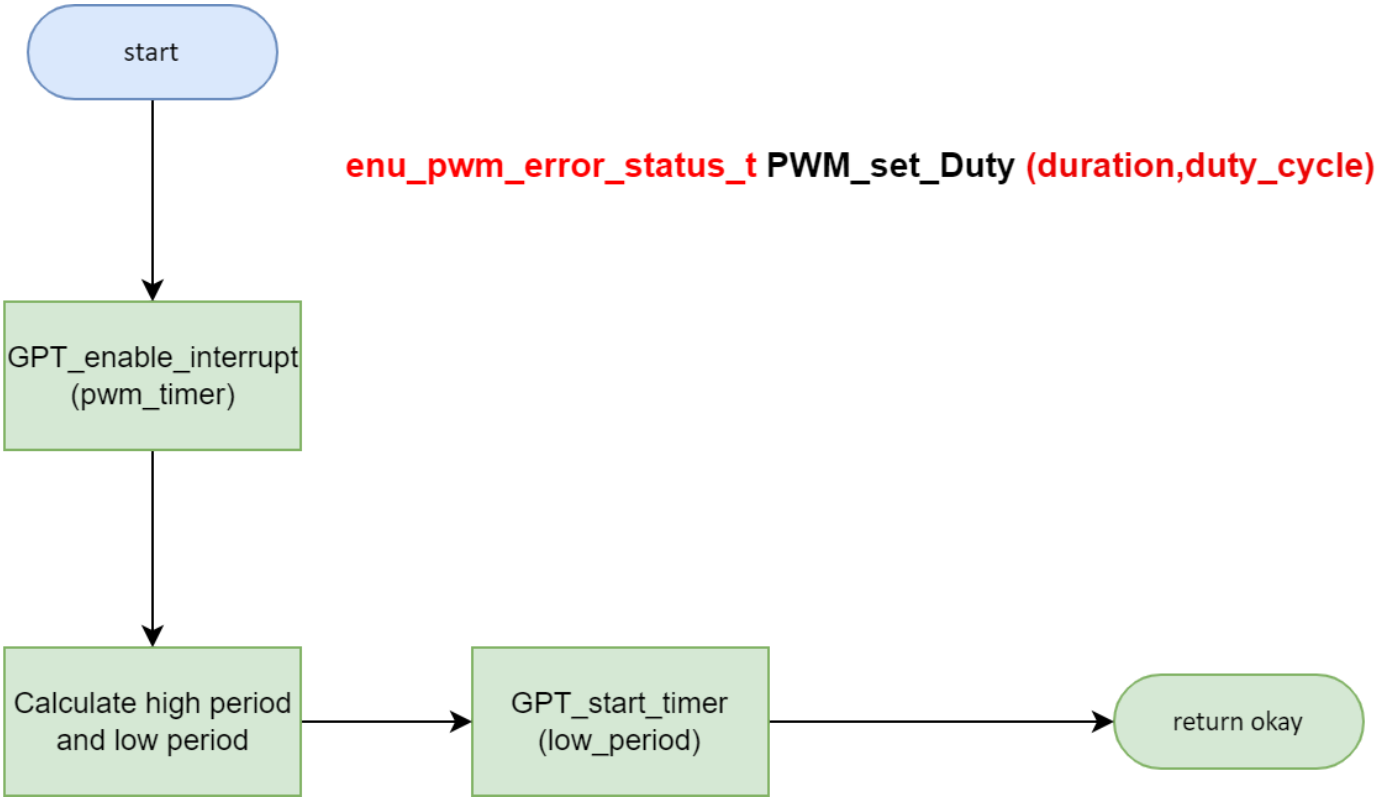
3.3.4.1.PWM_init_channel



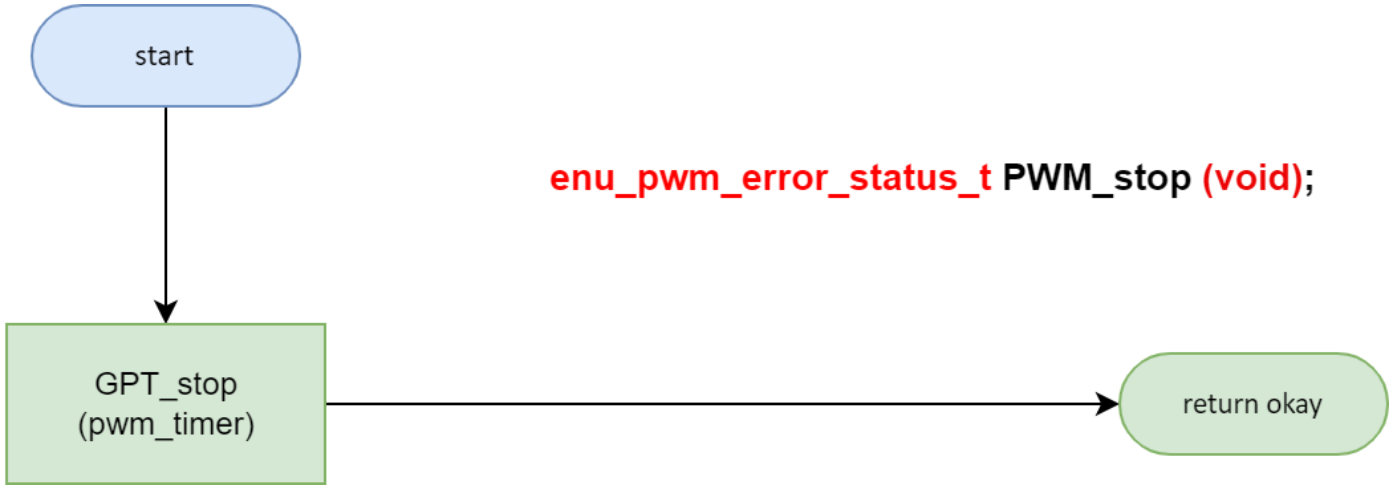
3.3.4.2.PWM_init_all_channels



3.3.4.2.PWM_set_Duty

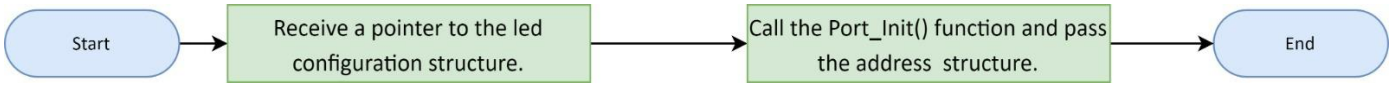


3.3.4.2.PWM_stop



3.3.5. LED

3.3.5.1.LED_init



3.3.5.2.LED_on

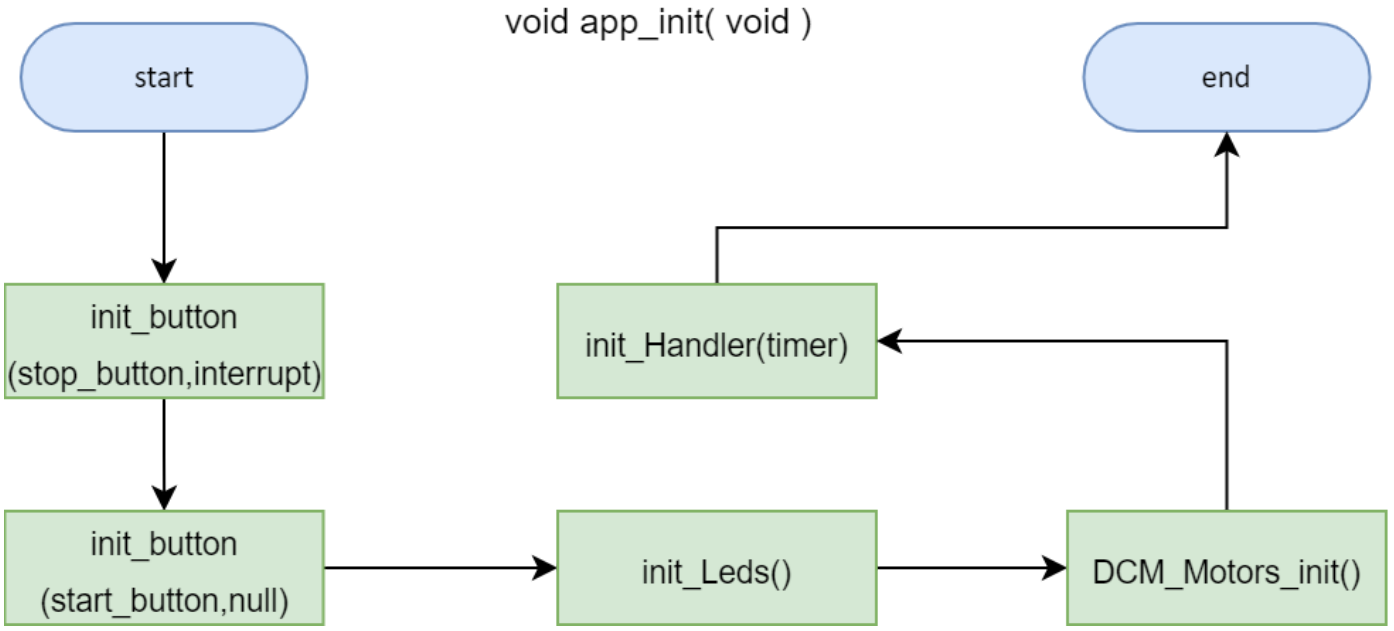
3.3.5.3.LED_off

3.3.5.4.LED_toggle

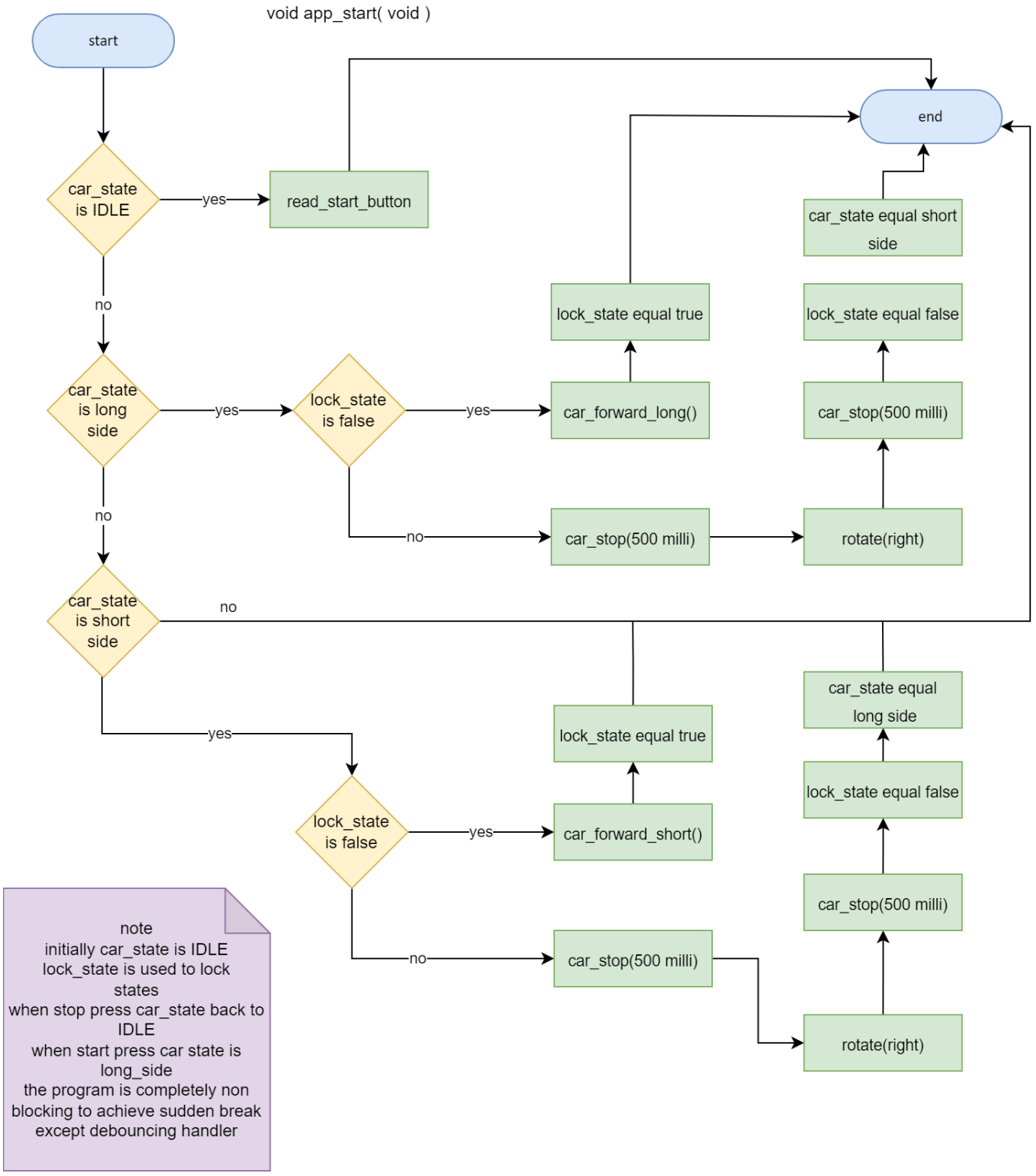
3.3. App

3.3.1. App

3.3.1.1.app_init



3.3.1.2.app_main



4. Precompiling and linking configuration

4.1.GPIO

4.1.1.linking Configuration

```
extern void callback(void);

ST_dio_pinCfg_t a_cfgPins[PINS_CFG_ARRAY_SIZE] =
{
  {
      .pinMode          =    GPIO_MODE_DIGITAL,
      .currentLevel     =    GPIO_CUR_8MA,
      .pinDirection     =    GPIO_DIR_OUTPUT,
      .pinLogic         =    GPIO_LOGIC_HIGH,
      .portNumber       =    GPIO_PORTE ,
      .pinNumber        =    GPIO_PIN2  ,
      .interruptStatus        = GPIO_INTERRUPT_DISABLE        ,
       .interruptSenseControl  = GPIO_INTERRUPT_SENSE_DISABLED  ,
       .interruptEvent         = GPIO_INTERRUPT_EVENT_DISABLED  ,
       .triggerStatus          = GPIO_INTERRUPT_TRIGGER_DISABLED,
       .ptrFucnPinIrqCallback  = PTR_NULL
    },
    {
      .pinMode        =    GPIO_MODE_DIGITAL,
      .currentLevel     =    GPIO_CUR_8MA,
      .pinDirection     =    GPIO_DIR_OUTPUT,
      .pinLogic       =    GPIO_LOGIC_LOW,
      .portNumber     =    GPIO_PORTE ,
      .pinNumber      =    GPIO_PIN3  ,
      .interruptStatus        = GPIO_INTERRUPT_DISABLE        ,
       .interruptSenseControl  = GPIO_INTERRUPT_SENSE_DISABLED  ,
       .interruptEvent         = GPIO_INTERRUPT_EVENT_DISABLED  ,
       .triggerStatus          = GPIO_INTERRUPT_TRIGGER_DISABLED,
       .ptrFucnPinIrqCallback  = PTR_NULL
```

```c
typedef enum
{
    GPIO_INTERRUPT_SENSE_LEVELS = 0,
    GPIO_INTERRUPT_SENSE_EDGES,
    GPIO_INTERRUPT_SENSE_DISABLED
}ENU_GPIO_interruptSenseControl_t;

typedef enum
{
    GPIO_INTERRUPT_EVENT = 0,
    GPIO_INTERRUPT_BOTH_EDGES,
    GPIO_INTERRUPT_EVENT_DISABLED
}ENU_GPIO_interruptEventStatus_t;

typedef enum
{
    GPIO_TRIGGER_LOW_EDGE_LOW_LEVEL = 0,
    GPIO_TRIGGER_HIGH_EDGE_HIGH_LEVEL,
    GPIO_INTERRUPT_TRIGGER_DISABLED
}ENU_GPIO_interruptEventTriggerStatus_t;


typedef struct{

    ENU_GPIO_pinMode_t                          pinMode;
    ENU_GPIO_currentLevel_t                     currentLevel;
    ENU_GPIO_direction_t                        pinDirection;
    ENU_GPIO_logic_t                            pinLogic;
    ENU_GPIO_pinInternalAttach_t                pinInternalAttach;
    ENU_GPIO_interruptStatus_t                  interruptStatus;
    ENU_GPIO_interruptSenseControl_t            interruptSenseControl;
    ENU_GPIO_interruptEventStatus_t             interruptEvent;
    ENU_GPIO_interruptEventTriggerStatus_t      triggerStatus;
    void(*ptrFucnPinIrqCallback)(void);
    ENU_GPIO_port_t                             portNumber;
    ENU_GPIO_pin_t                              pinNumber;

}ST_dio_pinCfg_t;
```

4.1.2.Precompiling Configuration

```c
#ifndef MCAL_GPIO_GPIO_INTERFACE_H_
#define MCAL_GPIO_GPIO_INTERFACE_H_

#include "TM4C123.h"
#include "std_types.h"
#include "bit_math.h"
#include "gpio_config.h"
#include "gpio_private.h"

extern void callback(void);


ENU_GPIO_systemState_t GPIO_init(ST_dio_pinCfg_t  *arg_pincfg);

ENU_GPIO_systemState_t GPIO_portDeinit(ST_dio_pinCfg_t  *arg_pincfg);

ENU_GPIO_systemState_t GPIO_interruptPortEnable(ST_dio_pinCfg_t  *arg_pincfg);

ENU_GPIO_systemState_t GPIO_interruptPortDisable(ST_dio_pinCfg_t  *arg_pincfg);


ENU_GPIO_systemState_t GPIO_writeLogic(ST_dio_pinCfg_t  *arg_pincfg , ENU_GPIO_logic_t arg_logicValue);

ENU_GPIO_systemState_t GPIO_readLogic(ST_dio_pinCfg_t  *arg_pincfg , ENU_GPIO_logic_t *arg_logicValue);

ENU_GPIO_systemState_t GPIO_toggleLogic(ST_dio_pinCfg_t  *arg_pincfg);

ENU_GPIO_systemState_t GPIO_setDirection(ST_dio_pinCfg_t  *arg_pincfg , ENU_GPIO_direction_t arg_diractionValue);

ENU_GPIO_systemState_t GPIO_setPinCallbackHandelr(ST_dio_pinCfg_t  *arg_pincfg);

#endif /* MCAL_GPIO_GPIO_INTERFACE_H_ */
```

```
 *      Author: 20101
 */

#ifndef MCAL_GPIO_GPIO_PRIVATE_H_
#define MCAL_GPIO_GPIO_PRIVATE_H_

#define RCGCGPIO_REG            *((VUint32_t *) 0x400FE608)
/*#define SYSCTL_RCGC2_R          (*((volatile uint32_t *)0x400FE108))*/

#define GPIO_OFFSET(x)          (x<4?((0x40004000) + ((x)*0x1000)) : ((0x40024000) + ((x-4) *0x1000)))

#define GPIODATA_REG(x)          *((VUint32_t *)(GPIO_OFFSET(x) + 0x3FC))
#define GPIODIR_REG(x)           *((VUint32_t *)(GPIO_OFFSET(x) + 0x0400))
#define GPIOIS_REG(x)            *((VUint32_t *)(GPIO_OFFSET(x) + 0x0404))
#define GPIOIBE_REG(x)           *((VUint32_t *)(GPIO_OFFSET(x) + 0x0408))
#define GPIOIEV_REG(x)           *((VUint32_t *)(GPIO_OFFSET(x) + 0x040C))
#define GPIOIM_REG(x)            *((VUint32_t *)(GPIO_OFFSET(x) + 0x0410))
#define GPIORIS_REG(x)           *((VUint32_t *)(GPIO_OFFSET(x) + 0x0414))
#define GPIOMIS_REG(x)           *((VUint32_t *)(GPIO_OFFSET(x) + 0x0418))
#define GPIOICR_REG(x)           *((VUint32_t *)(GPIO_OFFSET(x) + 0x041C))
#define GPIOAFSEL_REG(x)         *((VUint32_t *)(GPIO_OFFSET(x) + 0x0420))
#define GPIODR2R_REG(x)          *((VUint32_t *)(GPIO_OFFSET(x) + 0x0500))
#define GPIODR4R_REG(x)          *((VUint32_t *)(GPIO_OFFSET(x) + 0x0504))
#define GPIODR8R_REG(x)          *((VUint32_t *)(GPIO_OFFSET(x) + 0x0508))
#define GPIOODR_REG(x)           *((VUint32_t *)(GPIO_OFFSET(x) + 0x050C))
#define GPIOPUR_REG(x)           *((VUint32_t *)(GPIO_OFFSET(x) + 0x0510))
#define GPIOPDR_REG(x)           *((VUint32_t *)(GPIO_OFFSET(x) + 0x0514))
#define GPIOSLR_REG(x)           *((VUint32_t *)(GPIO_OFFSET(x) + 0x0518))
#define GPIODEN_REG(x)           *((VUint32_t *)(GPIO_OFFSET(x) + 0x051C))
#define GPIOLOCK_REG(x)          *((VUint32_t *)(GPIO_OFFSET(x) + 0x0520))
#define GPIOCR_REG(x)            *((VUint32_t *)(GPIO_OFFSET(x) + 0x0524))
#define GPIOAMSEL_REG(x)         *((VUint32_t *)(GPIO_OFFSET(x) + 0x0528))
#define GPIO_LOCK_KEY            0x4C4F434B
#define GPIO_DUMP_LOCK_KEY       0x03
```

4.2.GPT

4.2.1 linking Configuration

```
typedef struct
{
   enu_GPT_timer_select_t      enu_GPT_timer_select;
   enu_GPT_mode_t              enu_GPT_mode;
   enu_GPT_type_t              enu_GPT_type;
   boolean                     bool_use_interrupt;
   ptrf_cb_t                   ptrf_call_back;    //In case interrupt is true this field is mandatory

   /*TODO SUPPORT INDIVIDUAL TIMER FOR TIMERB*/
}str_GPT_configs_t;


// time unit selection
typedef enum
{
   TIME_IN_MICROSECONDS  = 0 ,
   TIME_IN_MILLIOSECONDS     ,
   TIME_IN_SECONDS           ,
   TIME_UNIT_INVALID
}enu_time_unit_t;

// GPT error types
typedef enum
{
   GPT_OKAY      = 0           ,
   GPT_TIMER_SELECT_ERROR      ,
   GPT_MODE_SELECT_ERROR       ,
   GPT_TYPE_SELECT_ERROR       ,
   GPT_NULL_REF_ERROR          ,
   GPT_INVALID_OPERATION_ERROR ,
   GPT_NULL_REF_CB_ERROR       ,
   GPT_INVALID_UNIT_ERROR


}enu_GPT_status_t;
```

4.2.2 Precompiling Configuration

```
#define GPT_OFFSET(x)           (x<8?((0x40030000)+((x)*0x1000)):((0x4004C000)+((x-8)*0x1000)))


#define GPTMCFG(x)              (*(VUint32_t*)(GPT_OFFSET(x)+(0x000)))
#define GPTMTAMR(x)             (*(VUint32_t*)(GPT_OFFSET(x)+(0x004)))
#define GPTMTBMR(x)             (*(VUint32_t*)(GPT_OFFSET(x)+(0x008)))
#define GPTMCTL(x)              (*(VUint32_t*)(GPT_OFFSET(x)+(0x00C)))
#define GPTMSYNC(x)             (*(VUint32_t*)(GPT_OFFSET(x)+(0x010)))
#define GPTMIMR(x)              (*(VUint32_t*)(GPT_OFFSET(x)+(0x018)))
#define GPTMRIS(x)              (*(VUint32_t*)(GPT_OFFSET(x)+(0x01C)))
#define GPTMMIS(x)              (*(VUint32_t*)(GPT_OFFSET(x)+(0x020)))
#define GPTMICR(x)              (*(VUint32_t*)(GPT_OFFSET(x)+(0x024)))
#define GPTMTAILR(x)            (*(VUint32_t*)(GPT_OFFSET(x)+(0x028)))
#define GPTMTBILR(x)            (*(VUint32_t*)(GPT_OFFSET(x)+(0x02C)))
#define GPTMTAMATCHR(x)         (*(VUint32_t*)(GPT_OFFSET(x)+(0x030)))
#define GPTMTBMATCHR(x)         (*(VUint32_t*)(GPT_OFFSET(x)+(0x034)))
#define GPTMTAPR(x)             (*(VUint32_t*)(GPT_OFFSET(x)+(0x038)))
#define GPTMTBPR(x)             (*(VUint32_t*)(GPT_OFFSET(x)+(0x03C)))
#define GPTMTAPMR(x)            (*(VUint32_t*)(GPT_OFFSET(x)+(0x040)))
#define GPTMTBPMR(x)            (*(VUint32_t*)(GPT_OFFSET(x)+(0x044)))
#define GPTMTAR(x)              (*(VUint32_t*)(GPT_OFFSET(x)+(0x048)))
#define GPTMTBR(x)              (*(VUint32_t*)(GPT_OFFSET(x)+(0x04C)))
#define GPTMTAV(x)              (*(VUint32_t*)(GPT_OFFSET(x)+(0x050)))
#define GPTMTBV(x)              (*(VUint32_t*)(GPT_OFFSET(x)+(0x054)))
#define GPTMRTCPD(x)            (*(VUint32_t*)(GPT_OFFSET(x)+(0x058)))
#define GPTMTAPS(x)             (*(VUint32_t*)(GPT_OFFSET(x)+(0x05C)))
#define GPTMTBPS(x)             (*(VUint32_t*)(GPT_OFFSET(x)+(0x060)))
#define GPTMTAPV(x)             (*(VUint32_t*)(GPT_OFFSET(x)+(0x064)))
#define GPTMTBPV(x)             (*(VUint32_t*)(GPT_OFFSET(x)+(0x068)))
#define GPTMPP(x)               (*(VUint32_t*)(GPT_OFFSET(x)+(0xFC0)))
```

4.3. LED

4.3.1 linking Configuration

```
/*
 * led_config.h
 *
 *  Created on: 17 Jun 2023
 *      Author: 20101
 */

#ifndef ECUAL_LED_LED_CONFIG_H_
#define ECUAL_LED_LED_CONFIG_H_

#include "gpio_interface.h"


#define  LED_PIN_CFG_ARRAY_SIZE    3


typedef enum
{
  LED_INIT_OK = 0,
  LED_INIT_NOK,
  LED_NULL_PTR

}ENU_LED_systemState_t;


typedef struct
{
    ENU_GPIO_port_t    portNumber;
    ENU_GPIO_pin_t     pinNumber;
    ENU_GPIO_logic_t   led_status;
}ST_led_pinCfg_t;


#endif /* ECUAL_LED_LED_CONFIG_H_ */
```

```c
/*
 * led_config.c
 *
 *  Created on: 17 Jun 2023
 *      Author: 20101
 */

#include "led_interface.h"

ST_led_pinCfg_t a_ledCfgPins[LED_PIN_CFG_ARRAY_SIZE] =
{
  {GPIO_PORTF , GPIO_PIN1 , GPIO_LOGIC_LOW},
  {GPIO_PORTF , GPIO_PIN2 , GPIO_LOGIC_LOW},
  {GPIO_PORTF , GPIO_PIN3 , GPIO_LOGIC_LOW}
};
```

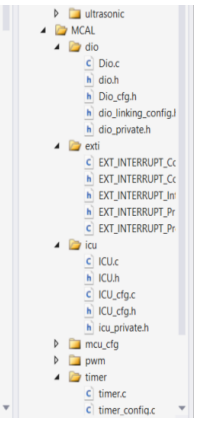
4.3.2 Precompiling Configuration

```c
/*
 * led_interface.h
 *
 *  Created on: 17 Jun 2023
 *      Author: 20101
 */

#ifndef ECUAL_LED_LED_INTERFACE_H_
#define ECUAL_LED_LED_INTERFACE_H_

#include "bit_math.h"
#include "std_types.h"
#include "gpio_interface.h"
#include "led_config.h"


ENU_LED_systemState_t LED_initialize(const ST_led_pinCfg_t *led);
ENU_LED_systemState_t LED_turnOn(const ST_led_pinCfg_t *led);
ENU_LED_systemState_t LED_turnOff(const ST_led_pinCfg_t *led);
ENU_LED_systemState_t LED_toggle(const ST_led_pinCfg_t *led);


#endif /* ECUAL_LED_LED_INTERFACE_H_ */
```

4.4. DCM

4.4.1 Linking Configuration

```c
 /* HAL */
 #include "../../MCAL/dio/dio_linking_config.h"
#include "DCM_private.h"
#include "dcm_cfg.h"

/*************************************************************************************************************************************************
/* Declaration and Initialization */


ST_DCM_g_Config_t ST_g_carMotors[2]=
{
    {    MOT0_EN_PIN_NUMBER_0 ,  MOT0_EN_PIN_NUMBER_1, MOT0_EN_PORT_NUMBER},
    {    MOT1_EN_PIN_NUMBER_0 ,  MOT1_EN_PIN_NUMBER_1, MOT1_EN_PORT_NUMBER}
};
```



4.4.1 Precompiling Configuration

```c

extern ST_DCM_g_Config_t ST_g_carMotors[2];
/*************************************************************************************************************************************************
/* DCM Functions' Prototypes */

EN_DCM_ERROR_T DCM_rotateDCM(EN_DCM_MOTORSIDE DCM_l_motorNumber, u16 DCM_a_rotateSpeed);

EN_DCM_ERROR_T DCM_changeDCMDirection(ST_DCM_g_Config_t* DCM_a_ptrToConfig, EN_DCM_MOTORSIDE DCM_a_motorNum);

EN_DCM_ERROR_T DCM_u8SetDutyCycleOfPWM(u8 DCM_a_dutyCycleValue);

void DCM_vdStopDCM(void);

EN_DCM_ERROR_T DCM_motorInit(ST_DCM_g_Config_t* DCM_a_ptrToConfig);

void DCM_updateStopFlag(void);
void DCM_MoveForward(u8 u8_a_speed);
void DCM_MoveBackward(u8 u8_a_speed);

//u8 DCM_u8GetDutyCycleOfPWM(u8* Cpy_pu8ReturnedDutyCycleValue);
/*************************************************************************************************************************************************

/*************************************************************************************************************************************************

```

4.5. PWM

4.5.1 Linking Configuration

```
str_pwm_congigs_t arr_str_gconst_pwm_configs[MAX_PWM_PINS] =
{
  {
     .enu_pwm_channel = PWM_CHANNEL_0 ,
     .str_dio_cfg     =
     {
       .pinMode                = GPIO_MODE_DIGITAL             ,
       .currentLevel           = GPIO_CUR_8MA                  ,
       .pinDirection           = GPIO_DIR_OUTPUT               ,
       .pinLogic               = GPIO_LOGIC_HIGH               ,
       .pinInternalAttach      = GPIO_PULL_UP                  ,
       .interruptStatus        = GPIO_INTERRUPT_DISABLE        ,
       .interruptSenseControl  = GPIO_INTERRUPT_SENSE_DISABLED ,
       .interruptEvent         = GPIO_INTERRUPT_EVENT_DISABLED ,
       .triggerStatus          = GPIO_INTERRUPT_TRIGGER_DISABLED,
       .portNumber             = GPIO_PORTB                    ,
       .pinNumber              = GPIO_PIN7                     ,
      .ptrFucnPinIrqCallback   = PTR_NULL
     }
  },
  {
     .enu_pwm_channel = PWM_CHANNEL_1 ,
     .str_dio_cfg     =
     {
       .pinMode                = GPIO_MODE_DIGITAL             ,
       .currentLevel           = GPIO_CUR_8MA                  ,
       .pinDirection           = GPIO_DIR_OUTPUT               ,
       .pinLogic               = GPIO_LOGIC_HIGH               ,
       .pinInternalAttach      = GPIO_PULL_UP                  ,
       .interruptStatus        = GPIO_INTERRUPT_DISABLE        ,
       .interruptSenseControl  = GPIO_INTERRUPT_SENSE_DISABLED ,
       .interruptEvent         = GPIO_INTERRUPT_EVENT_DISABLED ,
       .triggerStatus          = GPIO_INTERRUPT_TRIGGER_DISABLED,
       .portNumber             = GPIO_PORTF                    ,
       .pinNumber              = GPIO_PIN0                     ,
       .ptrFucnPinIrqCallback  = PTR_NULL
     }
  },
```

```
**@file     : pwm_cfg.h
**@brief    : this is header file for pwm configuration
**@author   : sharpel
**@date     : 25 june 2023
**@version  : 0.1
*/


#ifndef PWM_CFG_H_
#define PWM_CFG_H_


#include "gpio_interface.h"

#define MAX_PWM_PINS    2

typedef enum
{
  PWM_CHANNEL_0  = 0 ,
  PWM_CHANNEL_1      ,
  PWM_CHANNEL_2      ,
  PWM_CHANNEL_3      ,

}enu_pwm_channels_t;

typedef struct
{
  enu_pwm_channels_t enu_pwm_channel;
  ST_dio_pinCfg_t    str_dio_cfg;
}str_pwm_congigs_t;

#endif
```

4.5.2 Precompiling Configuration

```
/*
**@file     : pwm_interface.h
**@brief    : this is header file to interface with pwm
**@author   : sharpel
**@date     : 25 june 2023
**@version  : 0.1
*/


#ifndef PWM_INTERFACE_H_
#define PWM_INTERFACE_H_

#include "pwm_cfg.h"

typedef enum
{
  PWM_OK = 0,
  PWM_ERROR
}enu_pwm_error_status_t;


enu_pwm_error_status_t PWM_init_channel     (enu_pwm_channels_t enu_arg_pwm_channel);
enu_pwm_error_status_t PWM_init_all_channels(void);
enu_pwm_error_status_t PWM_set_Duty         (Uint16_t u16_arg_signal_duration_ms ,Uint8_t u8_arg_duty_cycle);
enu_pwm_error_status_t PWM_stop             (void);


#endif
```

4.6 Push Button

4.6.1 Precompiling Configuration

```
#include "push_button_cfg.h"

ST_PUSH_BTN_pinCfg_t a_pushBtnCfgPins[PUSH_BTN_PIN_CFG_ARRAY_SIZE] =
{
   {GPIO_PORTB , GPIO_PIN6 , GPIO_PULL_DOWN},
   {GPIO_PORTF , GPIO_PIN4 , GPIO_PULL_UP}
};
```

```c
#ifndef ECUAL_PUSH_BTN_PUSH_BTN_CONFIG_H_
#define ECUAL_PUSH_BTN_PUSH_BTN_CONFIG_H_

#include "gpio_interface.h"
#define  PUSH_BTN_PIN_CFG_ARRAY_SIZE    2


typedef enum
{
  PUSH_BTN_INIT_OK = 0,
  PUSH_BTN_INIT_NOK,
  PUSH_BTN_NULL_PTR,
  PUSH_BTN_INVALD_CONNECTION,
  PUSH_BTN_READ_OK,
  PUSH_BTN_READ_NOK

}ENU_PUSH_BTN_systemState_t;

typedef enum
{
    PUSH_BTN_STATE_PRESSED = 0,
    PUSH_BTN_STATE_RELEASED
}ENU_PUSH_BTN_state_t;

typedef struct
{
     ENU_GPIO_port_t                portNumber;
     ENU_GPIO_pin_t                 pinNumber;
     ENU_GPIO_pinInternalAttach_t   pushBtnConnection;
}ST_PUSH_BTN_pinCfg_t;


#endif /* ECUAL_PUSH_BTN_PUSH_BTN_CONFIG_H_ */
```

4.6.2 Linking Configuration

```c
#ifndef ECUAL_PUSH_BTN_PUSH_BTN_INTERFACE_H_
#define ECUAL_PUSH_BTN_PUSH_BTN_INTERFACE_H_


#include "STD_TYPES.h"
#include "BIT_MATH.h"
#include "gpio_interface.h"
#include "push_button_cfg.h"

ENU_PUSH_BTN_systemState_t PUSH_BTN_intialize (const ST_PUSH_BTN_pinCfg_t *btn,void (*ptrf)(void));
ENU_PUSH_BTN_systemState_t PUSH_BTN_read_state(const ST_PUSH_BTN_pinCfg_t *btn , ENU_PUSH_BTN_state_t *btn_state);


#endif /* ECUAL_PUSH_BTN_PUSH_BTN_INTERFACE_H_ */
```