RGB LED control V1.0

Developing the GPIO Driver and use it to control RGB LED on the TivaC board based using the push button.

1. Project Introduction

This project involves developing the GPIO Driver and use it to control RGB LED on the TivaC board based using the push button

2. High Level Design

2.1. System Architecture

2.1.1. Definition

Layered Architecture (*Figure 1*) describes an architectural pattern composed of several separate horizontal layers that function together as a single unit of software.

Microcontroller Abstraction Layer (*MCAL*) is a software module that directly accesses on-chip MCU peripheral modules and external devices that are mapped to memory, and makes the upper software layer independent of the MCU.

Hardware Abstraction Layer (*HAL*) is a layer of programming that allows a computer OS to interact with a hardware device at a general or abstract level rather than at a detailed hardware level.

2.1.2. Layered Architecture

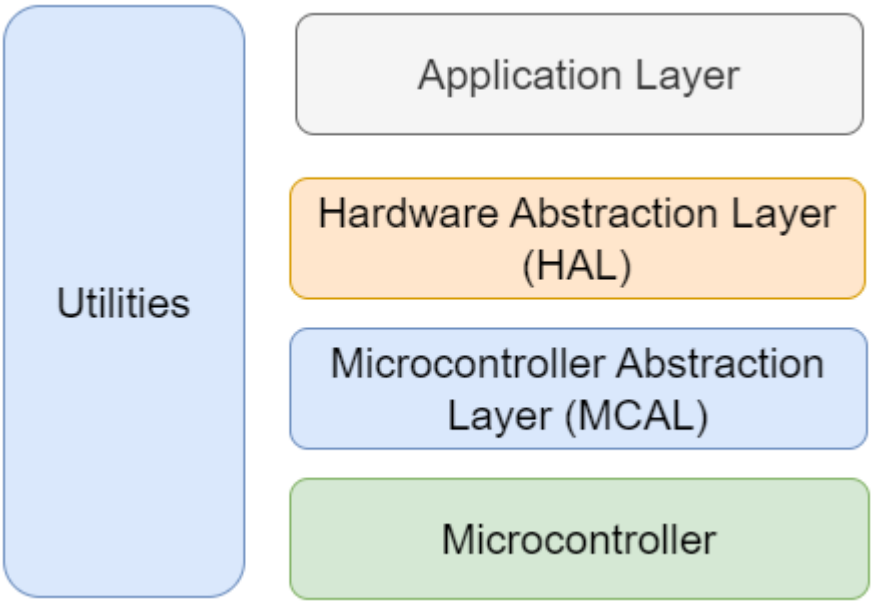


Figure 1. Layered Architecture Design

2.1.3. System Modules

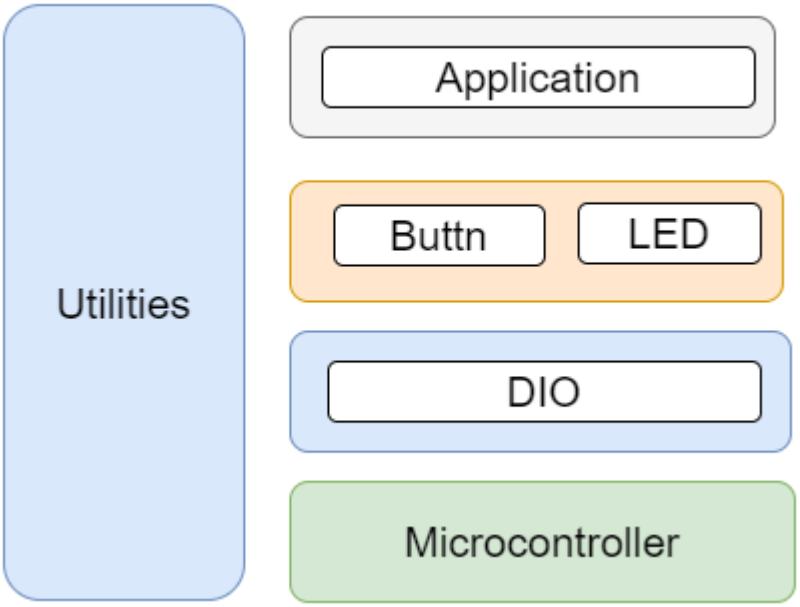


Figure 2. System Module Design

2.2. Modules Description

2.2.1. GPIO Module

The ***GPIO***, or ***General Purpose Input/Output***, is a simple form of interface used in a wide range of systems to effectively relay digital signals from sensors, transducers and mechanical equipment to other electrical circuits and devices.
Sometimes referred to as Digital Input/Output (DIO), GPIOutilizes a logic signal to transfer information.

2.2.3. Button Module

The ***Button*** can be considered the simplest input peripheral that can be connected to a microcontroller. Because of that, usually, every embedded development board is equipped with a button marked as “User Button” and this means it is actually connected to a GPIO pin you can read via software.

2.2.3. Led Module

The ***Led*** can be considered the simplest output peripheral that can be connected to a microcontroller. Because of that, usually, every embedded development board is equipped with a led marked as “Led” and this means it is actually connected to a GPIO pin you can turn it on or off or even toggle led via software.

2.3. Drivers’ Documentation (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 behavior and capabilities of the component and its inputs and outputs.
An *API* should be created so that it is generic and implementation independent. This allows for the API to be used in multiple applications with changes only to the implementation of the API and not the general interface or behavior.

2.3.2. MCAL APIs

2.3.2.1. GPIO Driver

```
|@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);
```

2.3.3. HAL APIs

2.3.3.1 LED Driver

```
| 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);
```

2.3.3.2 Button Driver

```
| Initializing the desired pin as input
|Parameters
|                   [in] uint8_port    the desired port for initializing the pin.
|                   [in] uint8_pin     the desired pin inside the port..
| Return
|    An enu_buttn_error_t value indicating the success or failure of
|                   the operation (BUTTN_OK ) if the operation succeeded, BUTTN_ERROR
|          otherwise)
|
enu_buttn_error_t  BUTTN_init(void);
```

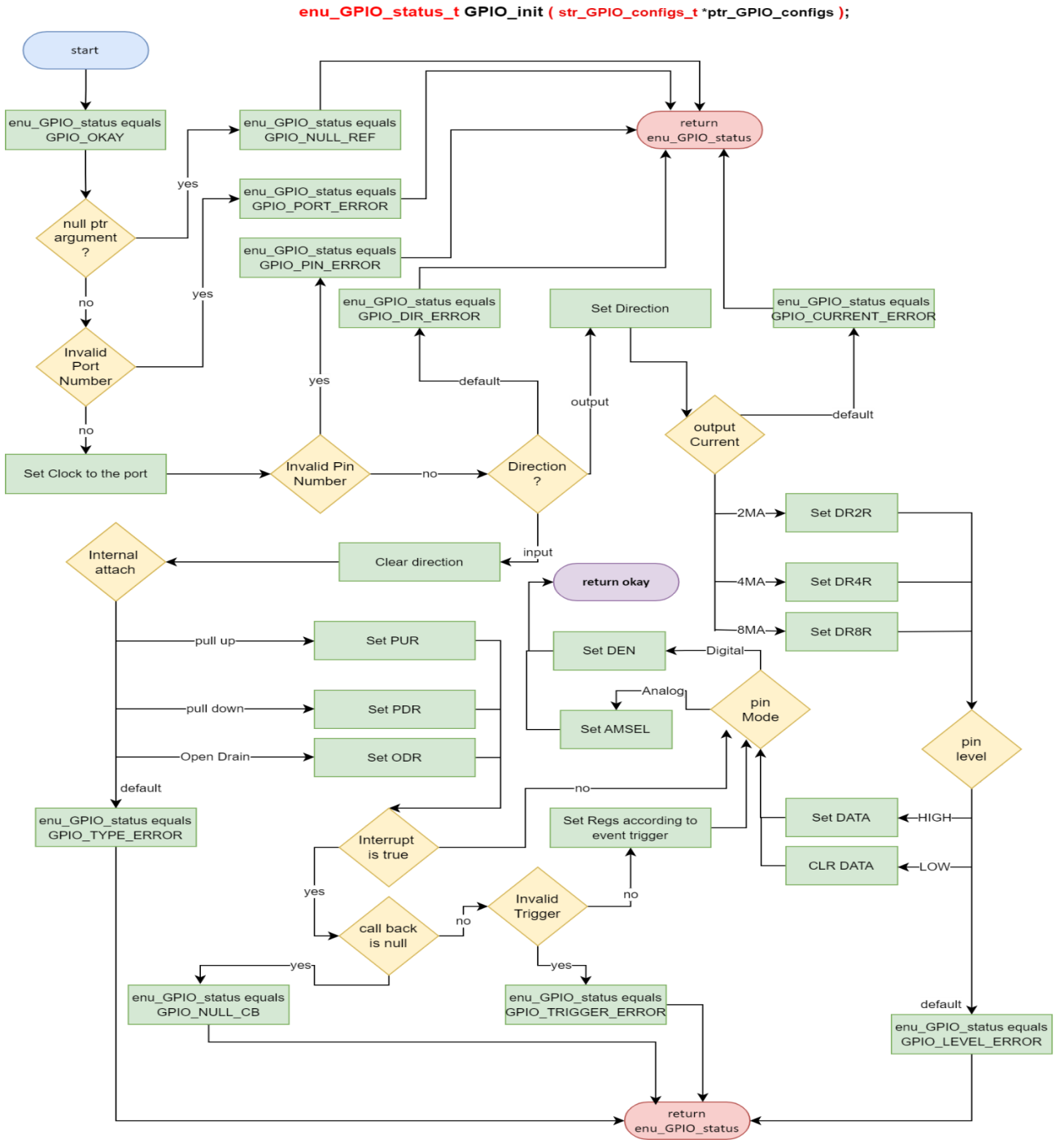
```
| Read the button status
|Parameters
|                   [in] uint8_button_id
| Return
|    An enu_buttn_error_t value indicating the success or failure of
|                   the operation (BUTTN_OK  if the operation succeeded, BUTTN_ERROR
|          otherwise)
enu_buttn_error_t BUTTN_read(uint8_t uint8_button_id);
```

3.Low Level Design

3.1. MCAL Flowcharts

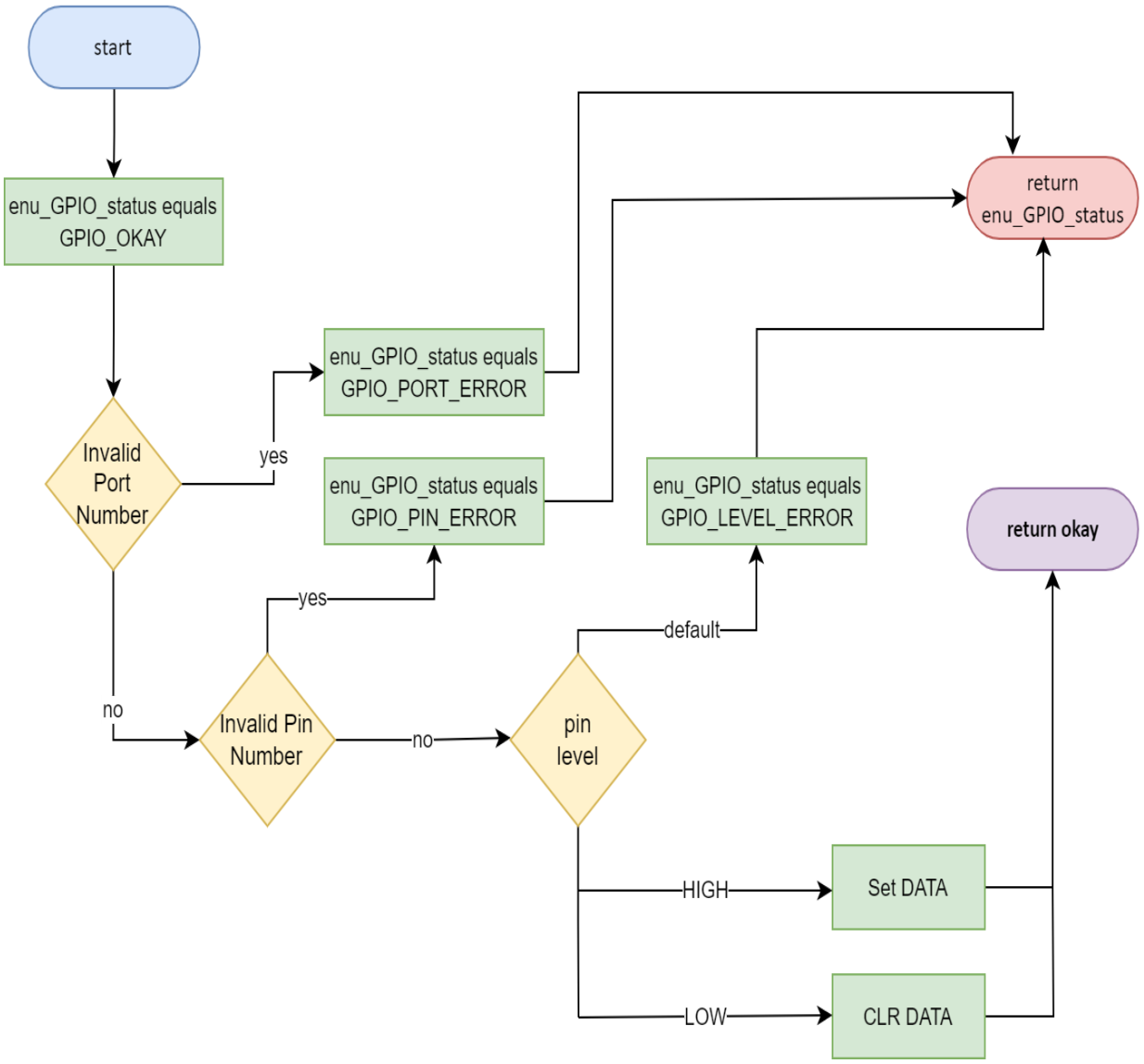
3.1.1 GPIO Module

3.1.2.1 GPIO_init

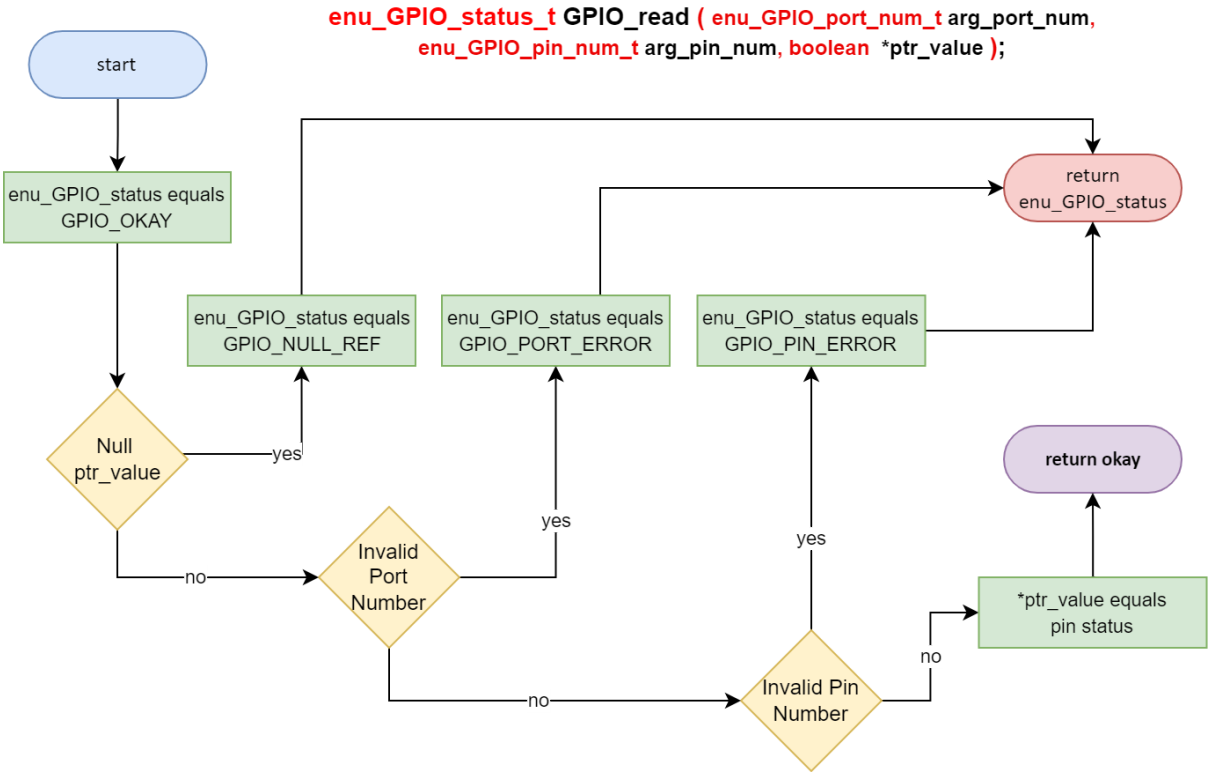


3.1.2.2 GPIO_write

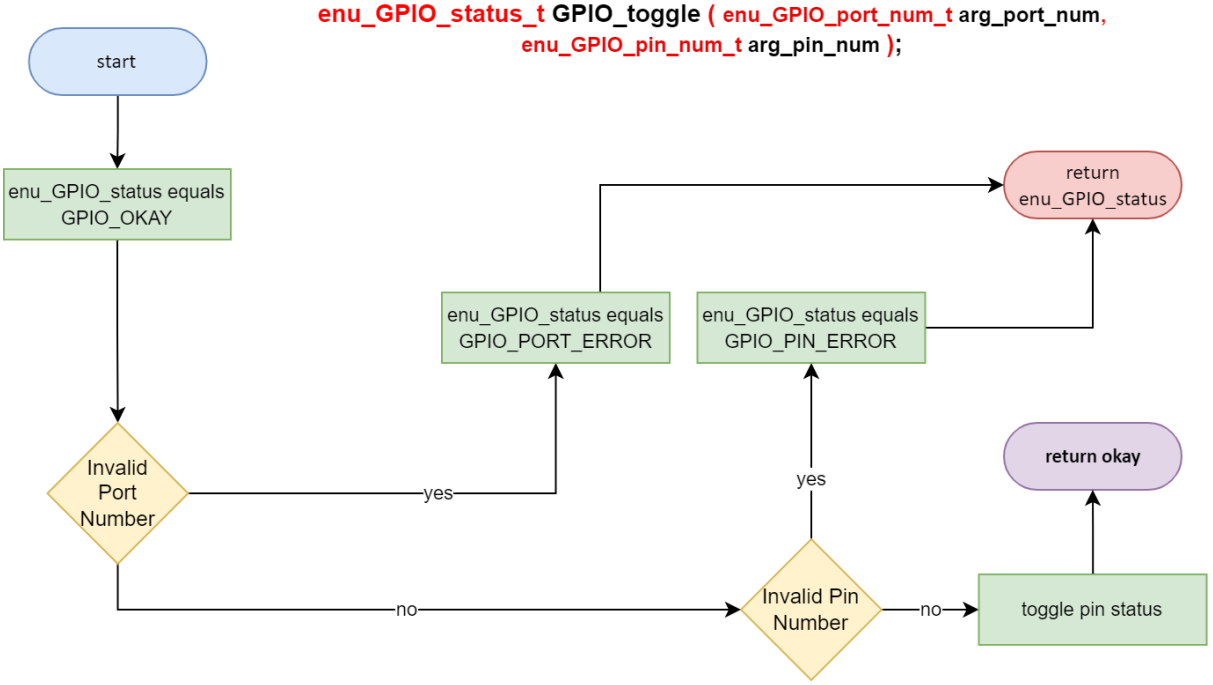
enu_GPIO_status_t GPIO_write (enu_GPIO_port_num_t arg_port_num, enu_GPIO_pin_num_t arg_pin_num, enu_GPIO_pin_level_t arg_level);



3.1.2.3 GPIO_read

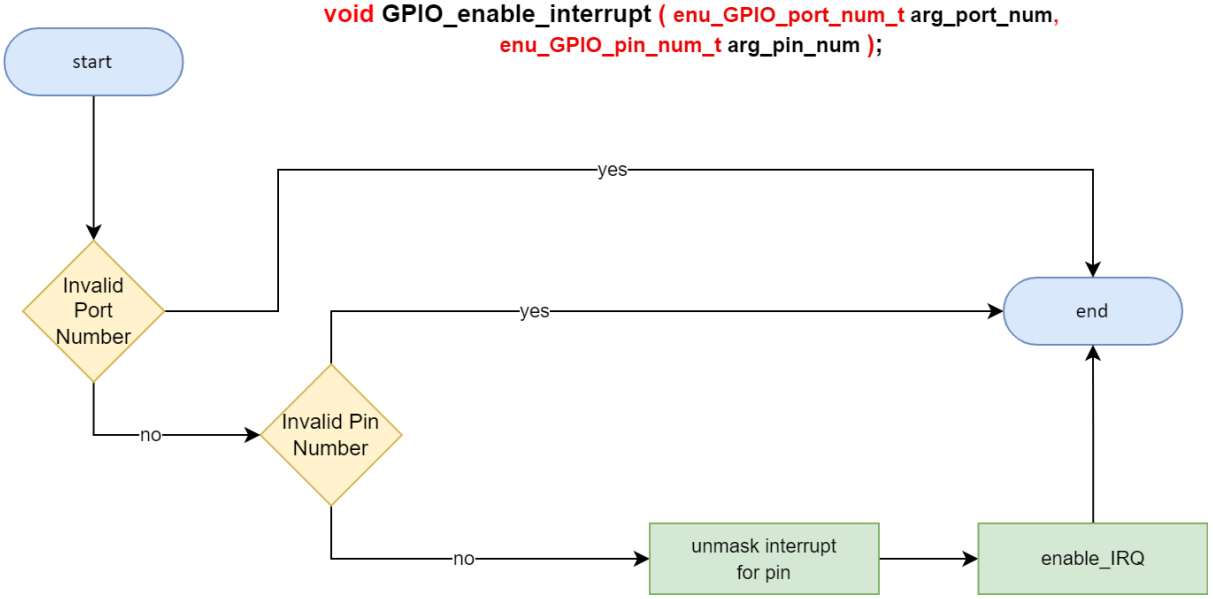


3.1.2.4 GPIO_toggle



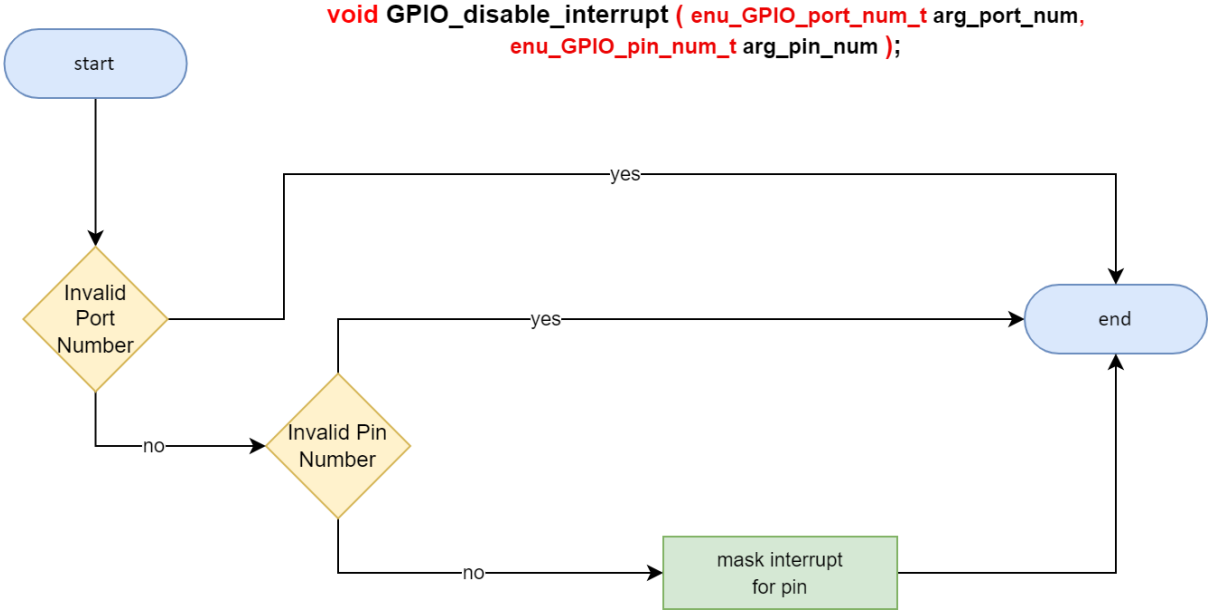
3.1.2.5 GPIO_enable_interrupr

void GPIO_enable_interrupt (enu_GPIO_port_num_t arg_port_num, enu_GPIO_pin_num_t arg_pin_num);



3.1.2.6 GPIO_desable_interrupr

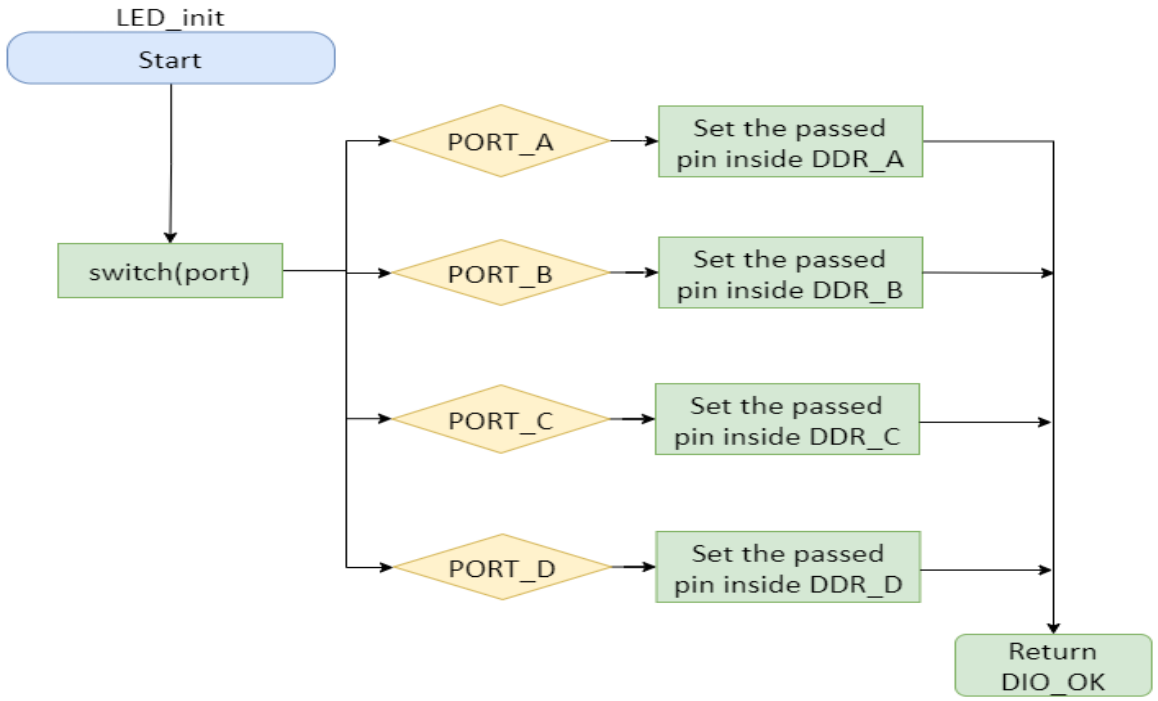
void GPIO_disable_interrupt (enu_GPIO_port_num_t arg_port_num, enu_GPIO_pin_num_t arg_pin_num);



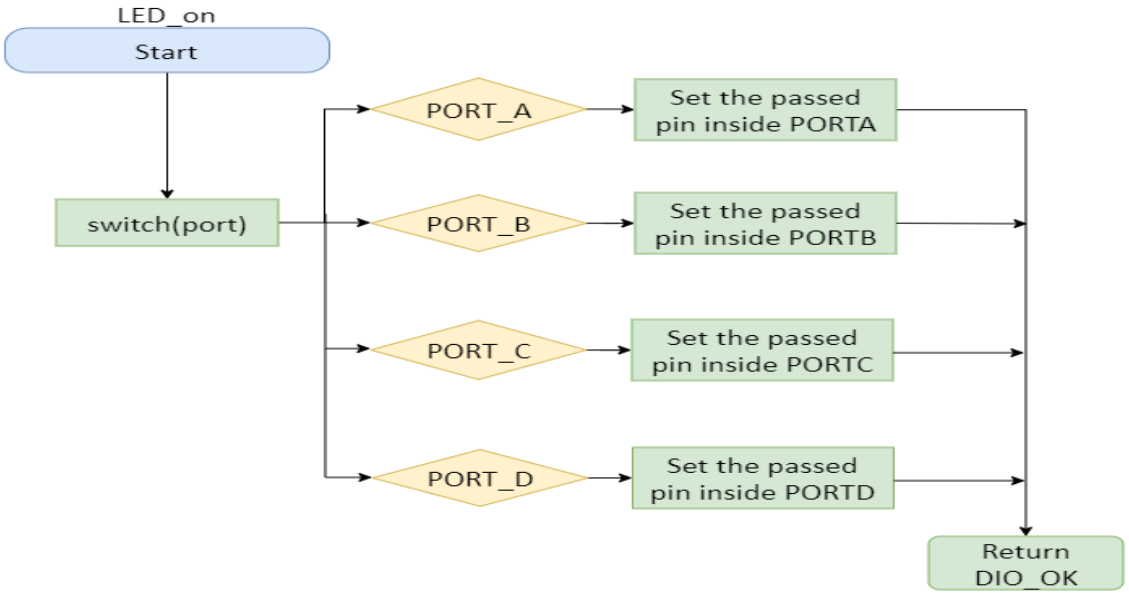
3.2. HAL Flowcharts

3.2.1 LED Module

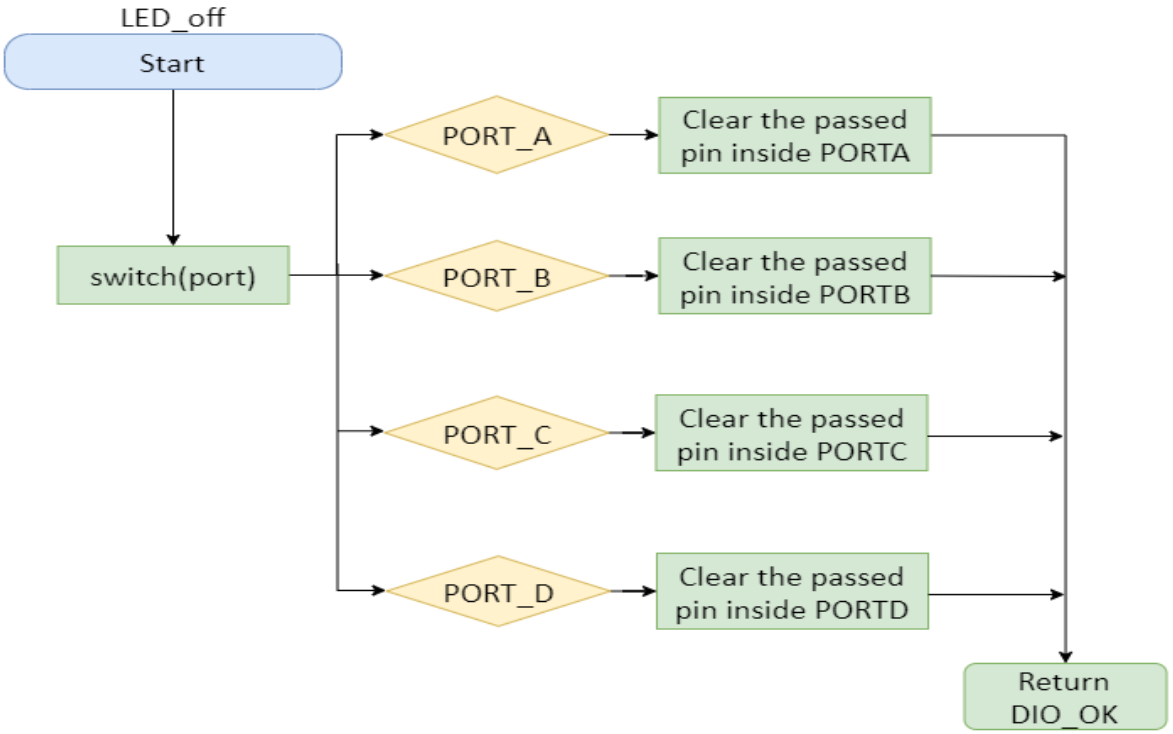
3.2.1.1 LED_init



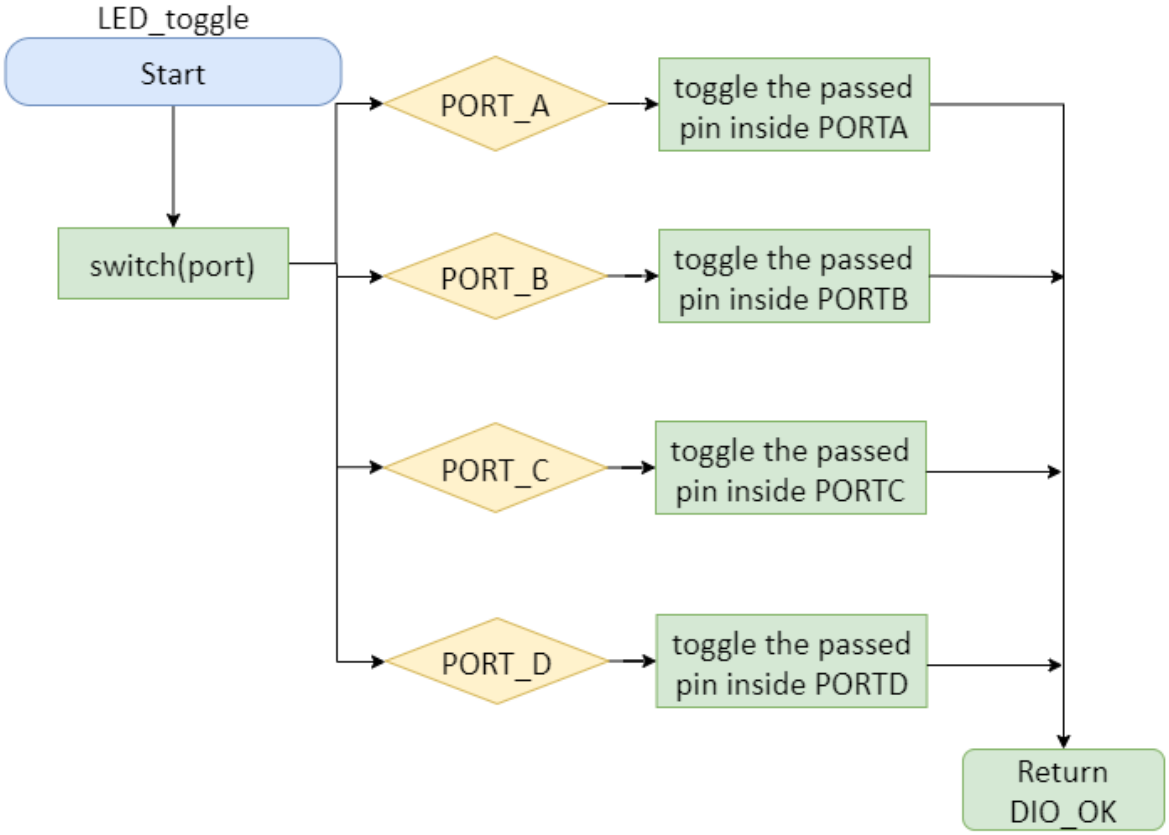
3.2.1.2 LED_on



3.2.1.3 LED_off



3.2.1.4 LED_toggle



3.2.1.5 LED_Linking Configuration

3.2.1.5.1 LED_cfg.c

```c
/*
 * led_cfg.c
 * Created: 18/6/2023 4:13:56 AM
 *  Author: Mahmoud Mowafey
 */
/** @file led_cfg.c
 * @brief The implementation for the led.
 */
/*******************************************************************************
* Includes
*******************************************************************************/
#include "led_cfg.h" /* For this modules definitions */
/*******************************************************************************
* Module Variable Definitions
*******************************************************************************/
/**
* Defines a table of structure to the configuration of the LED
*/
const str_led_confige_t LedConfig[] =
{
    { { PORTA, PIN0 , OUTPUT_PIN , DIGITAL_PIN , LOW_LEVEL ,PIN_2MA_CURRENT }, LED_0 }
};
```

```c
/*******************************************************************************
* Function Definitions
*******************************************************************************/
/*******************************************************************************
* Function : LED_Config()
*//**
* \b Description:
*
* This function is used to initialize the LED based on the configuration
* table defined in led_cfg module.
*
* PRE-CONDITION: Configuration table needs to populated (sizeof > 0)
*
* POST-CONDITION: A constant pointer to the first member of the
* configuration table will be returned.
*
* @return A pointer to the configuration table.
*
* \b Example Example:
* @code
* const str_led_confige_t *LED_Config = LED_GetConfig();
*
* @endcode
*
* @see LED_Init
* @see LED_on
* @see LED_off
* @see LED_toggle
* @see LED_status
*
*******************************************************************************/
const str_led_confige_t *const LED_ConfigGet(void)
{
/*
* The cast is performed to ensure that the address of the first element
* of configuration table is returned as a constant pointer and NOT a
* pointer that can be modified.
*/
 return (const *)LED_Config[0];
}
/***End of File**************************************************/
```

3.2.1.5.2 LED_cfg.h

```c
/*
 * led_cfg.h
 * Created: 18/6/2023 4:14:40 AM
 *  Author: Mahmoud Mowafey
 */

/** @file led_cfg.h
 * @brief This module contains interface definitions for the
 * LED configuration. This is the header file for the definition of the
 * interface for retrieving the LED configuration table.
*/

/*******************************************************************************
 * Includes
 *******************************************************************************/
#include "gpio_cfg.h" /* For this modules definitions */

/*******************************************************************************
 * Module Preprocessor Constants
 *******************************************************************************/
/*******************************************************************************
 * Module Typedefs
 *******************************************************************************/
/**
 * Defines an enumerated list of the LEDs.
 * The last element is used to specify the maximum number of
 * enumerated labels.
 */
typedef enum LED_ID {
    LED_0 = 0,
    LED_1,
    LED_2,
    LED_3,
    LED_4,
    LED_5,
    LED_6,
    LED_7,
    LED_8,
    LED_9,
    LED_10,
    LED_MAX
}enu_led_id_t;

/**
 * Defines the LED configuration table's elements that are used
 * by LED_Init to configure the LEDs.
 */
typedef struct led{
    str_GPIO_configs_t   gpio_for_led_confige;
    enu_led_id_t led_id;
}str_led_confige_t;

/*******************************************************************************
 * Module Function Prototypes
 *******************************************************************************/
const str_led_confige_t *const LED_ConfigGet(void);
```

3.2.1.6 LED_Pre-compiling Configuration

3.2.1.6.1 LED.h

```c
/*
 * led.h
 * Created: 18/6/2023 4:14:40 AM
 *  Author: Mahmoud Mowafey
 */

/** @file led.h
 * @brief This module contains interface definitions for the
 * LED APIs.
*/

#ifndef LED_H_
#define LED_H_
/*******************************************************************************
 * Includes
 *******************************************************************************/
#include "gpio_interface.h" /* For this modules definitions */

/*******************************************************************************
 * Module Typedefs
 *******************************************************************************/
// enum definition for Errors types
typedef enum LED_error {
    LED_OK = 0,
    LED_WRONG
}enu_led_error_status_t;

/**
 * Defines the possible states for a the LED pin.
 */
typedef enum
{
    LED_LOW, /** Defines digital state ground */
    LED_HIGH, /** Defines digital state power */
    LED_PIN_STATE_MAX /** Defines the maximum digital state */
}enu_led_state_t;
```

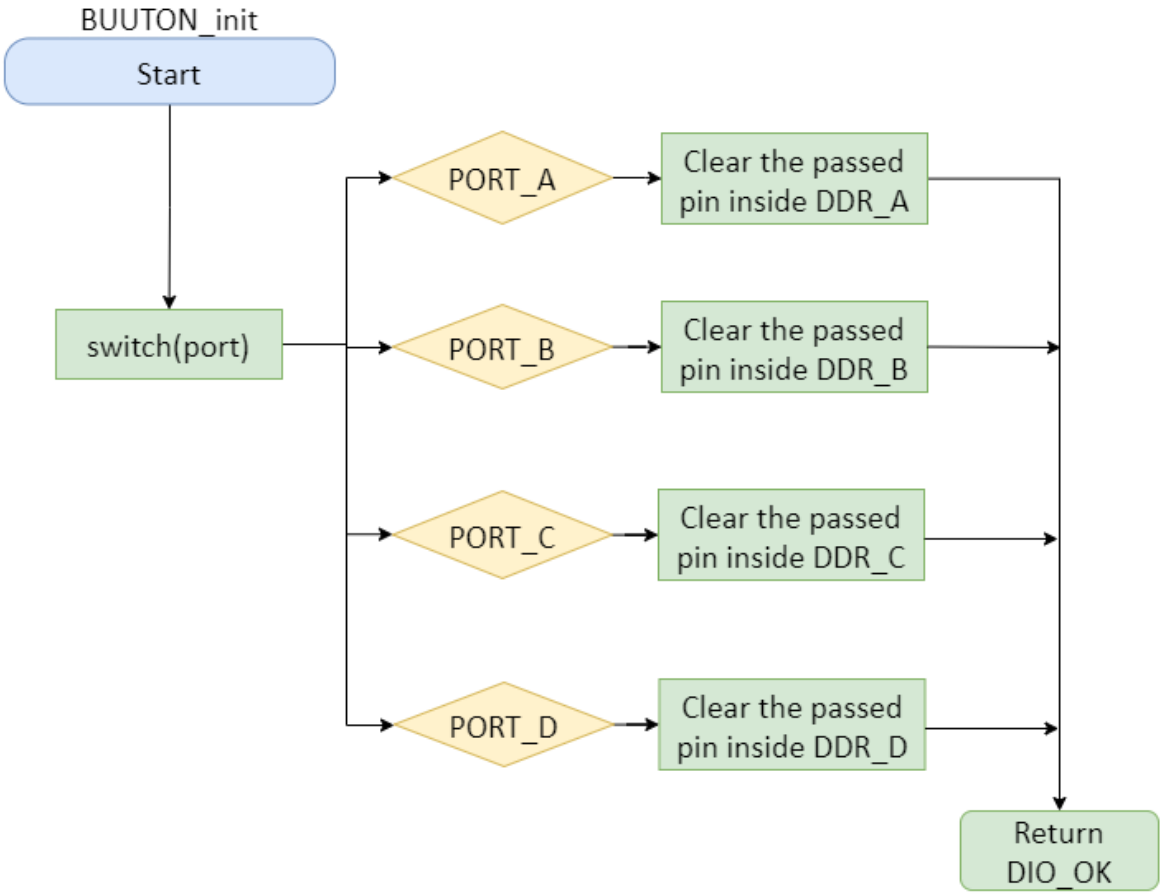
```c
/**********************************************************************
* Function Prototypes
**********************************************************************/
/**********************************************************************
* Function : LED_init()
*//**
* \b Description:
*
* This function is used to initialize the LED based on the configuration
* table defined in led_cfg module.
*
* PRE-CONDITION: Configuration table needs to populated (sizeof > 0)
*
* POST-CONDITION: A constant pointer to the first member of the
* configuration table will be returned.
*
* @return An enumeration for the LED_error status.
*
* @parameters :  [in] led_id.
*               [in] led_configuration.
* \b Example Example:
* @code
* LED_Init(str_led_confige_t *LedConfig);
*
* @endcode
*
* @see LED_Init
* @see LED_on
* @see LED_off
* @see LED_toggle
* @see LED_status
*
**********************************************************************/
enu_led_error_status_t LED_init(str_led_confige_t *LedConfig);
```

```c
/**********************************************************************
* Function : LED_on()
*//**
* \b Description:
*
* This function is used to Turn the LED on.
*
* PRE-CONDITION: Configuration table needs to populated (sizeof > 0)
* PRE-CONDITION: LED_ID needs to be passed
*
* POST-CONDITION: Output a logic high on the LED_pin.
*
* @return an enumeration for the LED_error status.
*
* @parameters :  [in] led_id.
*               [in] led_configuration.
* \b Example Example:
* @code
* LED_on(enu_led_id_t ledPin, str_led_confige_t *LedConfig);
*
* @endcode
*
* @see LED_Init
* @see LED_on
* @see LED_off
* @see LED_toggle
* @see LED_status
*
**********************************************************************/
enu_led_error_status_t LED_on(enu_led_id_t ledPin, str_led_confige_t *LedConfig);
```

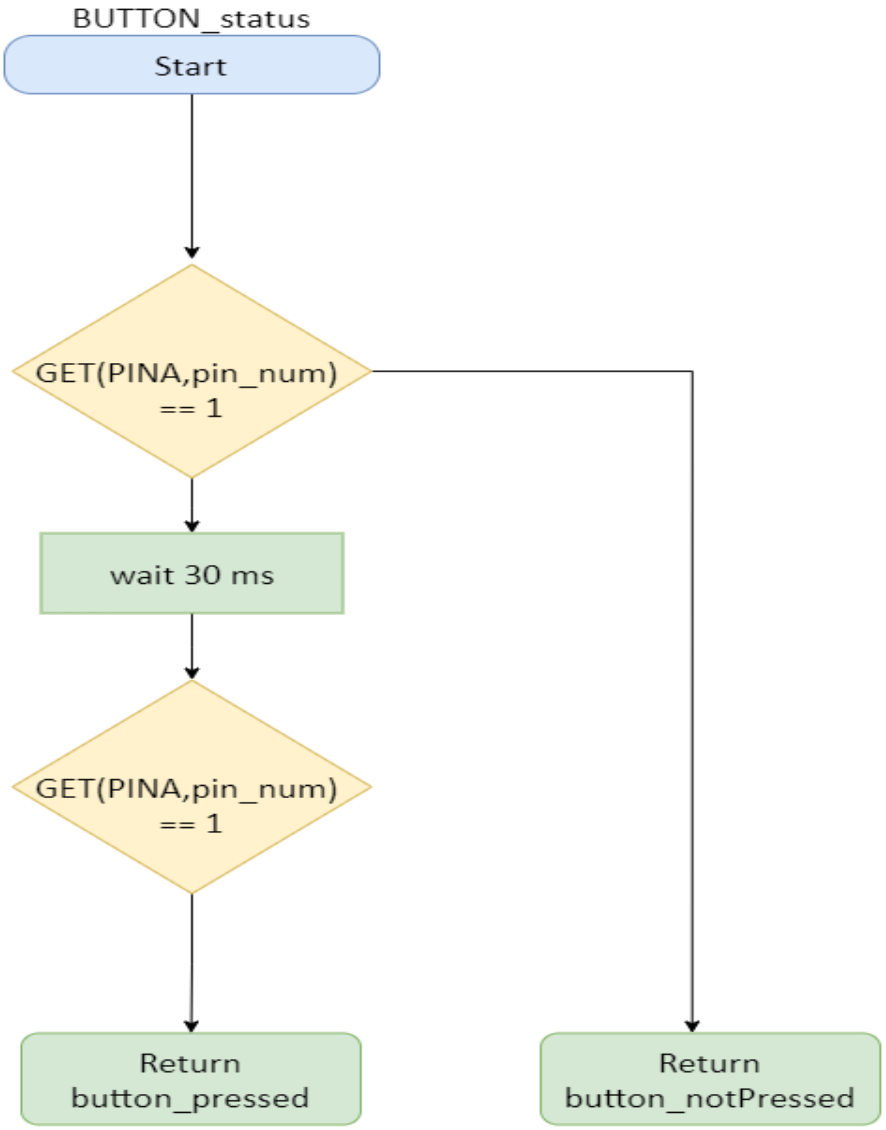
```c
/**********************************************************************
* Function : LED_off()
*//**
* \b Description:
*
* This function is used to Turn the LED on.
*
* PRE-CONDITION: Configuration table needs to populated (sizeof > 0)
* PRE-CONDITION: LED_ID needs to be passed
*
* POST-CONDITION: Output a logic low on the LED_pin.
*
* @return an enumeration for the LED_error status.
*
* @parameters :  [in] led_id.
*                [in] led_configuration.
* \b Example Example:
* @code
* LED_off(enu_led_id_t ledPin, str_led_confige_t *LedConfig);
*
* @endcode
*
* @see LED_Init
* @see LED_on
* @see LED_off
* @see LED_toggle
* @see LED_status
*
**********************************************************************/
enu_led_error_status_t LED_off(enu_led_pin_t ledPin, enu_led_port_t ledPort);

/**********************************************************************
* Function : LED_toggle()
*//**
* \b Description:
*
* This function is used to Turn the LED on.
*
* PRE-CONDITION: Configuration table needs to populated (sizeof > 0)
* PRE-CONDITION: LED_ID needs to be passed
*
* POST-CONDITION: Toggle the logic level on the LED_pin.
*
* @return an enumeration for the LED_error status.
*
* @parameters :  [in] led_id.
*                [in] led_configuration.
*
* \b Example Example:
* @code
* LED_toggle(enu_led_id_t ledPin, str_led_confige_t *LedConfig);
*
* @endcode
*
* @see LED_Init
* @see LED_on
* @see LED_off
* @see LED_toggle
* @see LED_status
*
**********************************************************************/

enu_led_error_status_t LED_toggle(enu_led_id_t ledPin, str_led_confige_t *LedConfig);

/**********************************************************************
* Function : LED_status()
*//**
* \b Description:
*
* This function is used to Turn the LED on.
*
* PRE-CONDITION: Configuration table needs to populated (sizeof > 0)
* PRE-CONDITION: LED_ID needs to be passed
*
* POST-CONDITION: Get the status of the LED.
*
* @return an enumeration for the LED_error status.
*
* @parameters :  [in] led_id.
*                [in] led_configuration.
*                [in] pointer to led_status var.
* \b Example Example:
* @code
* LED_get_status(enu_led_id_t ledPin, str_led_confige_t *LedConfig);
*
* @endcode
*
* @see LED_Init
* @see LED_on
* @see LED_off
* @see LED_toggle
* @see LED_status
*
**********************************************************************/
enu_led_error_status_t LED_get_status(enu_led_id_t ledPin, str_led_confige_t *LedConfig, uint8_t *ptr_status_var);

#endif /* LED_H_ */
/***End of File*************************************************/
```

3.2.2 BUTTON Module

3.2.2.1 BUTT_init



3.2.2.2 BUTT_status



3.2.2.3 Precompiling Configuration

```
#ifndef BUTTON_H_
#define BUTTON_H_


#include "button_cfg.h"

// enum definition for Errors types
typedef enum Button_error {
    BUTTON_OK = 0,
    BUTTON_WRONG
}enu_button_error_status_t;


enu_button_error_status_t BUTTON_init(void);
enu_button_error_status_t BUTTON_read(enu_button_id_t button_id,boolean *value);


#endif
```

3.2.2.4 Button_Linking Configuration

3.2.2.4.1 Button_cfg.c

```
/*
 * led_cfg.c
 * Created: 18/6/2023 4:13:56 AM
 *  Author: Mahmoud Mowafey
 */
/** @file led_cfg.c
 * @brief The implementation for the led.
 */
/**********************************************************************
* Includes
**********************************************************************/
#include "led_cfg.h" /* For this modules definitions */

/**********************************************************************
* Module Preprocessor Constants
**********************************************************************/
/**********************************************************************
* Module Preprocessor Macros
**********************************************************************/
/**********************************************************************
* Module Typedefs
**********************************************************************/
/**********************************************************************
* Module Variable Definitions
**********************************************************************/
/**
* Defines a table of pointers to the peripheral input register on the
* microcontroller.
*/

const str_led_confige_t LedConfig[] =
{
    { { PORTA, PIN0 , OUTPUT_PIN , DIGITAL_PIN , LOW_LEVEL ,PIN_2MA_CURRENT }, LED_0 }
};
```

```c
/**********************************************************************
 * Function Definitions
 **********************************************************************/
/**********************************************************************
 * Function : LED_Config()
 *//**
 * \b Description:
 *
 * This function is used to initialize the LED based on the configuration
 * table defined in led_cfg module.
 *
 * PRE-CONDITION: Configuration table needs to populated (sizeof > 0)
 *
 * POST-CONDITION: A constant pointer to the first member of the
 * configuration table will be returned.
 *
 * @return A pointer to the configuration table.
 *
 * \b Example Example:
 * @code
 * const str_led_confige_t *LED_Config = LED_GetConfig();
 *
 * Dio_Init(DioConfig);
 * @endcode
 *
 * @see LED_Init
 * @see LED_on
 * @see LED_off
 * @see LED_toggle
 * @see LED_status
 *
 **********************************************************************/
const str_led_confige_t *const LED_ConfigGet(void)
{
/*
 * The cast is performed to ensure that the address of the first element
 * of configuration table is returned as a constant pointer and NOT a
 * pointer that can be modified.
 */
 return (const *)LED_Config[0];
}
```

3.2.2.4.2 Button_cfg.h

```c
/*
 * led_cfg.h
 * Created: 18/6/2023 4:14:40 AM
 *  Author: Mahmoud Mowafey
 */

/** @file led_cfg.h
 * @brief This module contains interface definitions for the
 * LED configuration. This is the header file for the definition of the
 * interface for retrieving the LED configuration table.
*/

/**********************************************************************
 * Includes
 **********************************************************************/
#include "gpio_cfg.h" /* For this modules definitions */

/**********************************************************************
 * Module Preprocessor Constants
 **********************************************************************/
/**********************************************************************
 * Module Typedefs
 **********************************************************************/
typedef enum LED_ID {
    LED_0 = 0,
    LED_1,
    LED_2,
    LED_3,
    LED_4,
    LED_5,
    LED_6,
    LED_7,
    LED_8,
    LED_9,
    LED_10,
}enu_led_id_t;


typedef struct led{
    str_GPIO_configs_t   gpio_for_led_confige;
    enu_led_id_t led_id;
}str_led_confige_t;

/**********************************************************************
 * Module Function Prototypes
 **********************************************************************/
const str_led_confige_t *const LED_ConfigGet(void);
/***End of File**********************************************************/
```

3.3. APP Flowchart