SPRINTS

19 JUNE

# 2023

# RGB LED CONTROL V1.0 REPORT

RGB LED
CONTROL V1.0

Created By :
Momen Hassan
Sherif Ashraf Khadr

# 1 - Project Introduction

## 1.1 - Project Description

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

## 1.2 - Project Components

### 1.2.2 Hardware Requirements

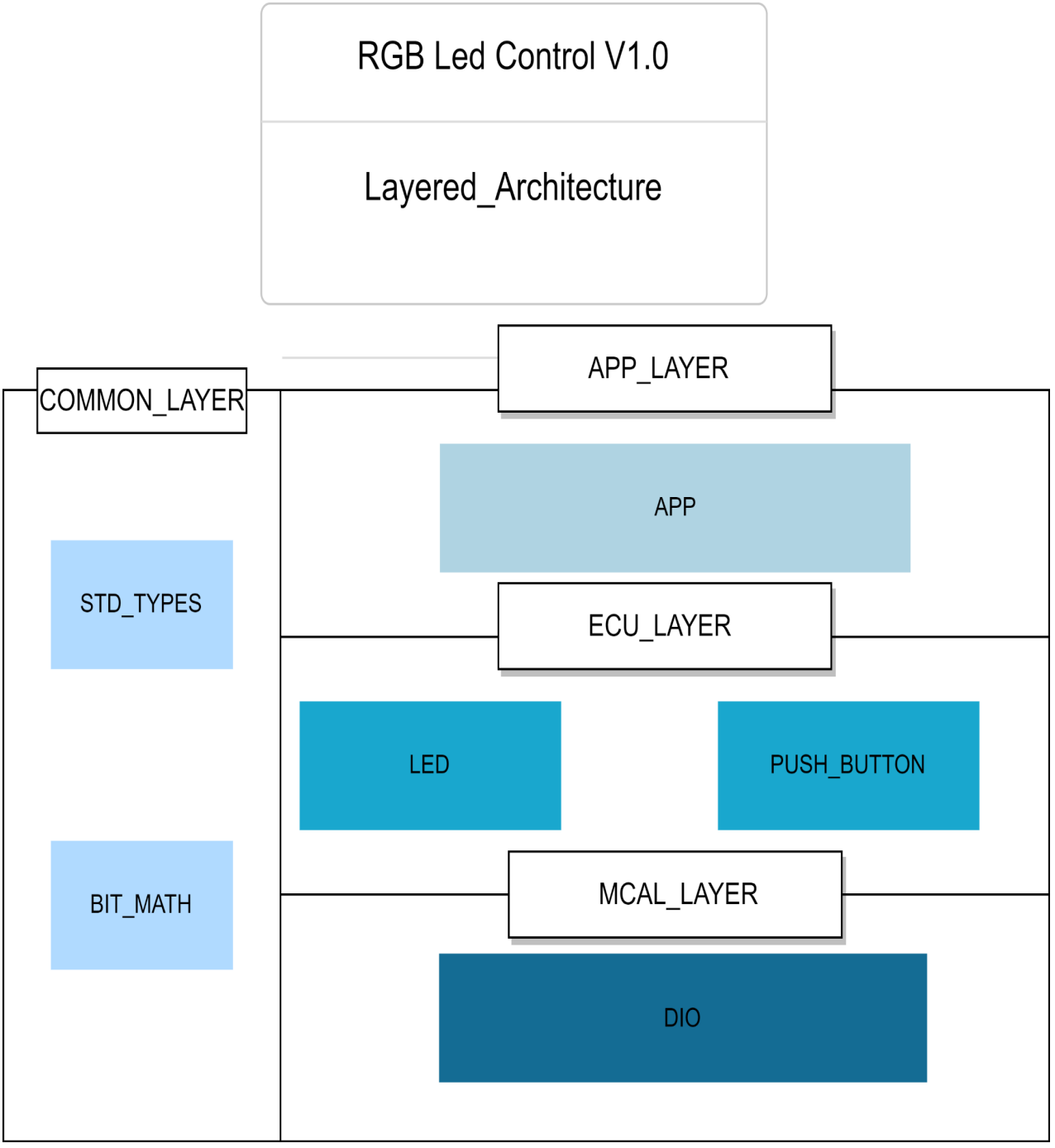
- Use the TivaC board
- Use SW1 as an input button
- Use the RGB LED

### 1.2.3 Software Requirements

- The RGB LED is OFF initially
- Pressing SW1:
- After the first press, the Red led is on
- After the second press, the Green Led is on
- After the third press, the Blue led is on
- After the fourth press, all LEDs are on
- After the fifth press, should disable all LEDs
- After the sixth press, repeat steps from 1 to 6

# 2 - High Level Design

## 2.1 - Layered Architecture



## 2.2 - Modules Description

### 2.2.1 - GPIO Module

A DIO (Digital Input/Output) module is a hardware component that provides digital input and output capabilities to a system. It can be used to interface with digital sensors, switches, and actuators, and typically includes features such as interrupt capability, programmable resistors, overvoltage/current protection, and isolation. Additionally, some DIO modules may include advanced features such as counter/timer functionality or PWM.

### 2.2.2 - LED Module

A LED (Light Emitting Diode) module is a hardware component that provides visual output to a system. It can be controlled by software running on a microcontroller and typically includes features such as brightness control, colour selection, and blinking patterns. The LED module is useful for providing status indicators, displaying data, or as a user interface, and can be interfaced with the microcontroller through different protocols such as I2C, SPI, or digital I/O.

### 2.2.3 - PUSH_BUTTON Module

A push button is a simple switch mechanism that controls some aspect of a machine or a process. It is usually made of plastic or metal and has a flat or shaped surface that can be easily pressed or pushed. A push button can be either momentary or latching, meaning that it returns to its original state when released or stays in the pushed state until pressed again. Push buttons are used for various purposes, such as turning on or off devices, performing calculations and controlling games.

### 2.2.4 - APP Module

An app module can use ECU driver modules to handle the flow of a specific application within a system. The ECU driver module provides low-level access to the hardware components of the system, such as sensors and actuators. The app module uses these driver modules to interact with the system and perform its specific tasks, resulting in more efficient development and better code organization.

## 2.3 - Drivers’ documentation

### 2.3.1 - GPIO Driver

```
/*
 * @brief Initializes a pin with the given configuration.
 * @param ConfigPtr A pointer to the configuration structure.
 * @return void
 *
 * This function takes a pointer to a Port_ConfigType structure and
```

```
 * sets the pin registers according to the configuration parameters.
 * The function should be called before using any other pin functions.
 */
```

**void Port_Init( Port_ConfigType* ConfigPtr);**

```
/*
 * @brief Writes a value to a specific pin of a port.
 * @param port_num The port number to write to.
 * @param pin_num The pin number to write to.
 * @param value The value to write (0 or 1).
 * @return void
 *
 * This function writes a logical value (0 or 1) to a specific pin of a port.
 * The port and pin numbers should be valid and configured as output pins.
 */
```

**void GPIO_Write(Port_Num port_num, Port_PinNum pin_num, uint8_t value);**

```
/*
 * @brief Reads a value from a specific pin of a port.
 * @param port_num The port number to read from.
 * @param pin_num The pin number to read from.
 * @param value A pointer to store the read value (0 or 1).
 * @return void
 *
 * This function reads a logical value (0 or 1) from a specific pin of a port
 * and stores it in the memory location pointed by value.
 * The port and pin numbers should be valid and configured as input pins.
 */
```

**void GPIO_Read(Port_Num port_num, Port_PinNum pin_num,uint8_t* value);**

```
/*
 * @brief Toggles the value of a specific pin of a port.
 * @param port_num The port number to toggle.
 * @param pin_num The pin number to toggle.
 * @return void
```

```
 *
 * This function toggles the logical value (0 or 1) of a specific pin of a port.
 * The port and pin numbers should be valid and configured as output pins.
 */
 void GPIO_Toggle(Port_Num port_num, Port_PinNum pin_num);

/**
 * @brief Enables the interrupt for a specific pin of a port.
 * @param port_num The port number to enable the interrupt for.
 * @param port_pinnum The pin number to enable the interrupt for.
 * @return void
 *
 * This function enables the interrupt for a specific pin of a port,
 * which means that an interrupt will be triggered when the pin changes its state.
 * The port and pin numbers should be valid and configured as input pins with
interrupt capability.
 */
void GPIO_Enable_Interrupt(Port_Num port_num,Port_Pin Num port_pinnum);

/*
 * @brief Disables the interrupt for a specific pin of a port.
 * @param port_num The port number to disable the interrupt for.
 * @param port_pinnum The pin number to disable the interrupt for.
 * @return void
 *
 * This function disables the interrupt for a specific pin of a port,
 * which means that no interrupt will be triggered when the pin changes its state.
 * The port and pin numbers should be valid and configured as input pins with
interrupt capability.
 */
 void GPIO_Disable_Interrupt(Port_Num,Port_PinNum);
```

### 2.3.2 - LED Driver

```
/**
 * @brief Initializes a LED with the given configuration.
 * @param led A pointer to the configuration structure for the LED.
 * @return void
```

```
 *
 * This function initializes a LED with the given configuration structure,
 * which specifies the port and pin numbers for the LED and its active state (0 or 1).
 * The function should be called before using any other LED functions.
 */
 void LED_initialize(const ST_led_pinCfg_t *led);

/**
 * @brief Turns on a LED with the given configuration.
 * @param led A pointer to the configuration structure for the LED.
 * @return void
 *
 * This function turns on a LED with the given configuration structure,
 * which specifies the port and pin numbers for the LED and its active state (0 or 1).
 * The function sets the pin value to the active state of the LED.
 * The LED should be initialized before calling this function.
 */
 void LED_turnOn(const ST_led_pinCfg_t *led);

/**
 * @brief Turns off a LED with the given configuration.
 * @param led A pointer to the configuration structure for the LED.
 * @return void
 *
 * This function turns off a LED with the given configuration structure,
 * which specifies the port and pin numbers for the LED and its active state (0 or 1).
 * The function sets the pin value to the opposite of the active state of the LED.
 * The LED should be initialized before calling this function.
 */
 void LED_turnOff(const ST_led_pinCfg_t *led);

/**
 * @brief Toggles a LED with the given configuration.
 * @param led A pointer to the configuration structure for the LED.
 * @return void
 *
 * This function toggles a LED with the given configuration structure,
 * which specifies the port and pin numbers for the LED and its active state (0 or 1).
 * The function inverts the pin value of the LED.
 * The LED should be initialized before calling this function.
 */
 void LED_toggle(const ST_led_pinCfg_t *led);
```

### 2.3.3 - PUSH_BOTTON Driver

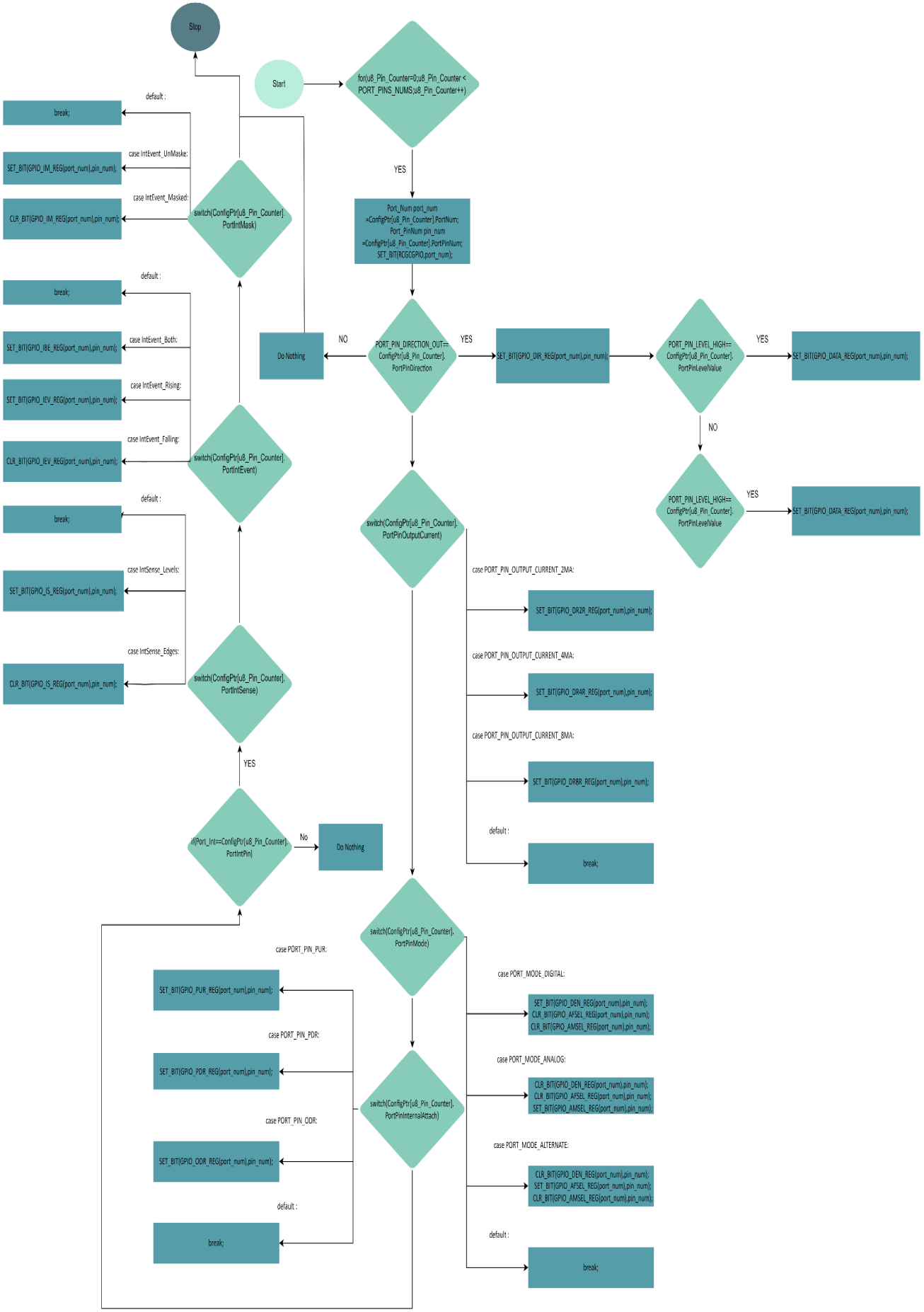
```
/**
 * @brief Initializes a push button with the given configuration.
 * @param btn A pointer to the configuration structure for the push button.
 * @return void
 *
 * This function initializes a push button with the given configuration structure,
 * which specifies the port and pin numbers for the push button and its active state (0
*or 1).
 * The function should be called before using any other push button functions.
 */
 void PUSH_BTN_intialize(const ST_PUSH_BTN_pinCfg_t *btn);

/**
 * @brief Reads the state of a push button with the given configuration.
 * @param btn A pointer to the configuration structure for the push button.
 * @param btn_state A pointer to store the state of the push button
(PUSH_BTN_PRESSED or PUSH_BTN_RELEASED).
 * @return void
 *
 * This function reads the state of a push button with the given configuratio structure,
 * which specifies the port and pin numbers for the push button and its active state (0
*or 1).
 * The function compares the pin value with the active state of the push button and
*stores
 * the result in the memory location pointed by btn_state.
 * The push button should be initialized before calling this function.
 */
void PUSH_BTN_read_state(const ST_PUSH_BTN_pinCfg_t *btn ,
ENU_PUSH_BTN_state_t *btn_state);
```

# 3 - Low-Level Design

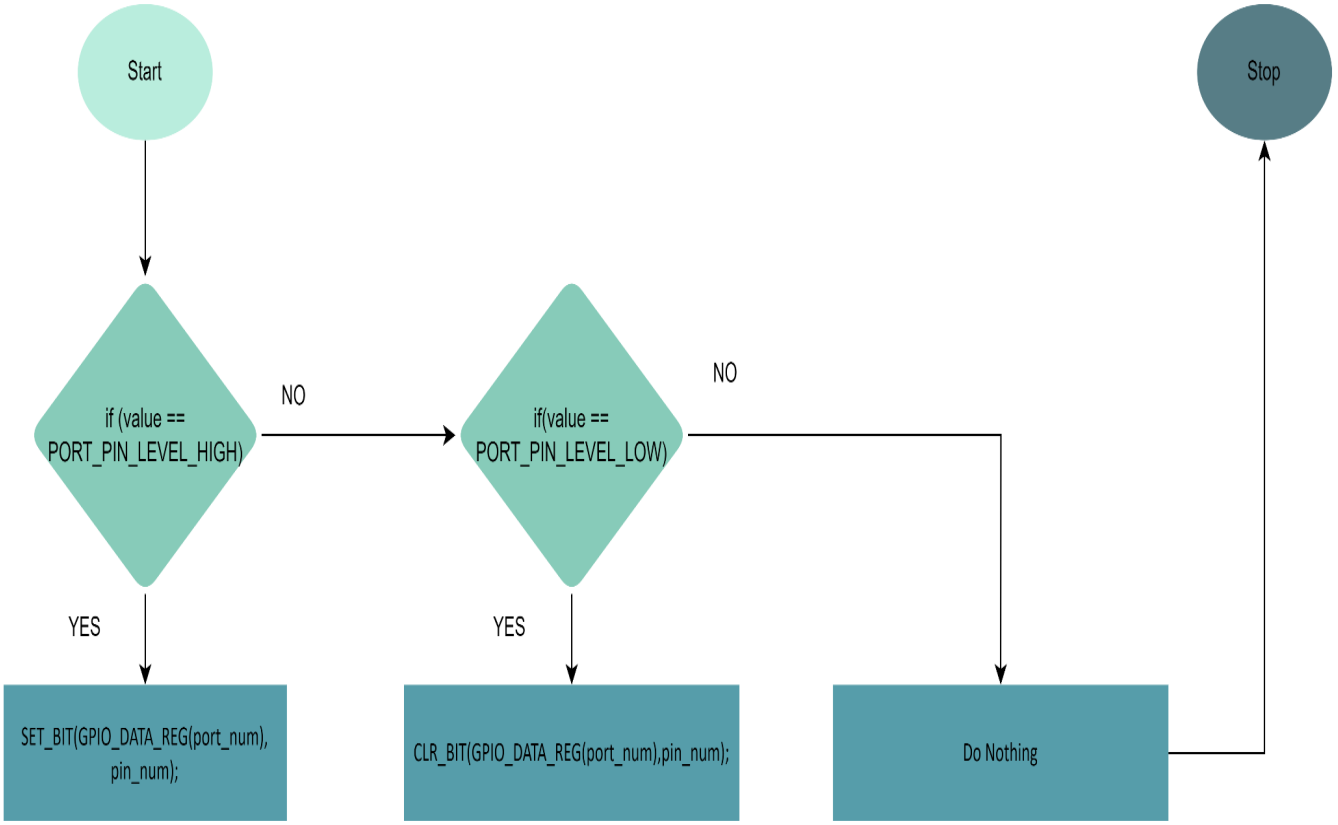
## 3.1 - Module Flow Charts

### 3.1.1 - GPIO Flow Charts

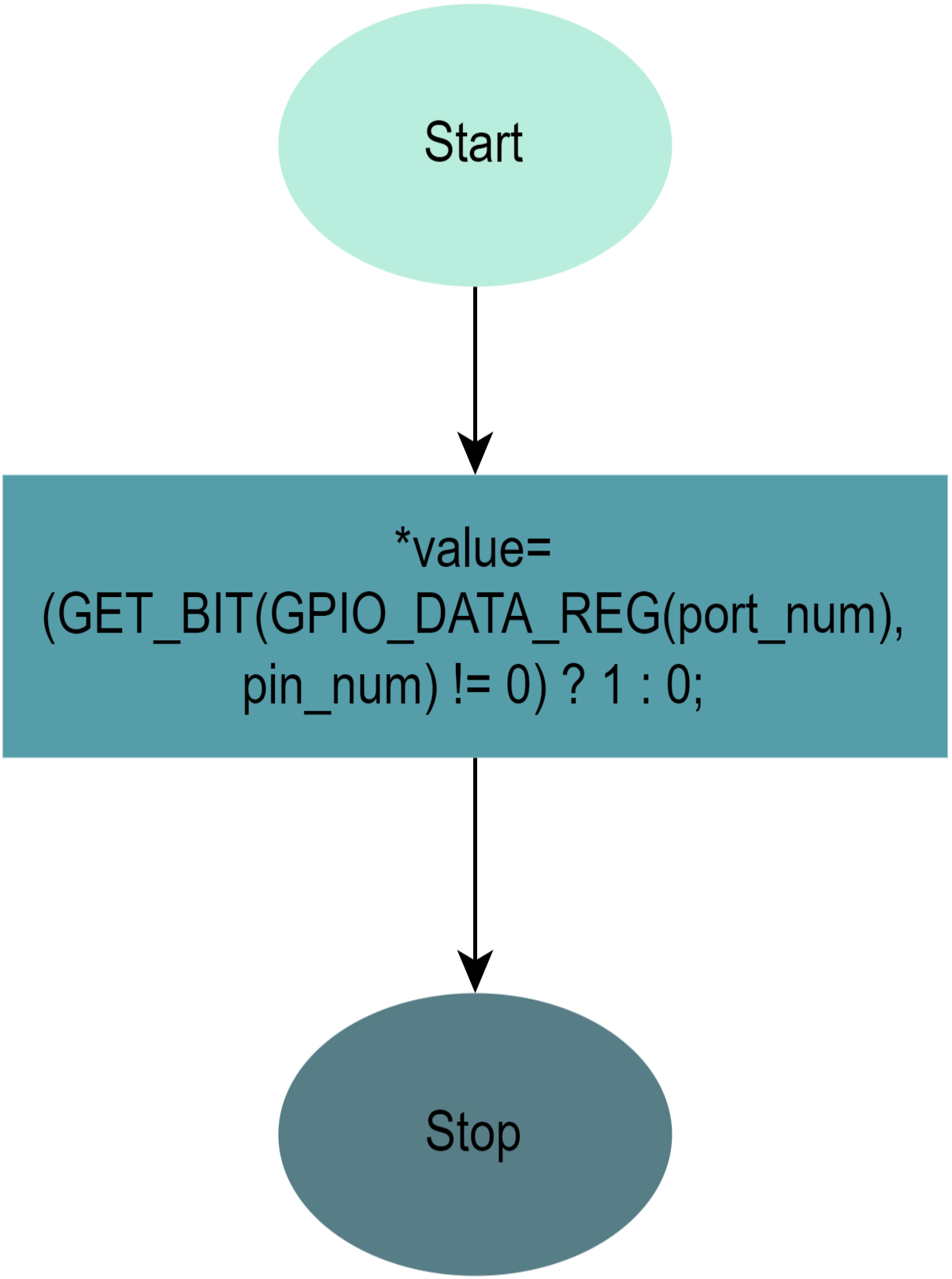
#### 3.1.1.1 - GPIO_INIT


Stop
Start
for(u8_Pin_Counter=0;u8_Pin_Counter < PORT_PINS_NUMS;u8_Pin_Counter++)
YES
Port_Num port_num =ConfigPtr[u8_Pin_Counter].PortNum;
Port_PinNum pin_num =ConfigPtr[u8_Pin_Counter].PortPinNum;
SET_BIT(RCGCGPIO,port_num);
PORT_PIN_DIRECTION_OUT== ConfigPtr[u8_Pin_Counter]. PortPinDirection
NO
Do Nothing
YES
SET_BIT(GPIO_DIR_REG(port_num),pin_num);
PORT_PIN_LEVEL_HIGH== ConfigPtr[u8_Pin_Counter]. PortPinLevelValue
YES
SET_BIT(GPIO_DATA_REG(port_num),pin_num);
NO
PORT_PIN_LEVEL_HIGH== ConfigPtr[u8_Pin_Counter]. PortPinLevelValue
YES
SET_BIT(GPIO_DATA_REG(port_num),pin_num);
switch(ConfigPtr[u8_Pin_Counter]. PortPinOutputCurrent)
case PORT_PIN_OUTPUT_CURRENT_2MA:
SET_BIT(GPIO_DR2R_REG(port_num),pin_num);
case PORT_PIN_OUTPUT_CURRENT_4MA:
SET_BIT(GPIO_DR4R_REG(port_num),pin_num);
case PORT_PIN_OUTPUT_CURRENT_8MA:
SET_BIT(GPIO_DR8R_REG(port_num),pin_num);
default :
break;
switch(ConfigPtr[u8_Pin_Counter]. PortPinMode)
case PORT_MODE_DIGITAL:
SET_BIT(GPIO_DEN_REG(port_num),pin_num);
CLR_BIT(GPIO_AFSEL_REG(port_num),pin_num);
CLR_BIT(GPIO_AMSEL_REG(port_num),pin_num);
case PORT_MODE_ANALOG:
CLR_BIT(GPIO_DEN_REG(port_num),pin_num);
CLR_BIT(GPIO_AFSEL_REG(port_num),pin_num);
SET_BIT(GPIO_AMSEL_REG(port_num),pin_num);
case PORT_MODE_ALTERNATE:
CLR_BIT(GPIO_DEN_REG(port_num),pin_num);
SET_BIT(GPIO_AFSEL_REG(port_num),pin_num);
CLR_BIT(GPIO_AMSEL_REG(port_num),pin_num);
default :
break;
switch(ConfigPtr[u8_Pin_Counter]. PortPinInternalAttach)
case PORT_PIN_PUR:
SET_BIT(GPIO_PUR_REG(port_num),pin_num);
case PORT_PIN_PDR:
SET_BIT(GPIO_PDR_REG(port_num),pin_num);
case PORT_PIN_ODR:
SET_BIT(GPIO_ODR_REG(port_num),pin_num);
default :
break;
if(Port_Int==ConfigPtr[u8_Pin_Counter]. PortIntPin)
No
Do Nothing
YES
switch(ConfigPtr[u8_Pin_Counter]. PortIntSense)
case IntSense_Levels:
SET_BIT(GPIO_IS_REG(port_num),pin_num);
case IntSense_Edges:
CLR_BIT(GPIO_IS_REG(port_num),pin_num);
default :
break;
switch(ConfigPtr[u8_Pin_Counter]. PortIntEvent)
case IntEvent_Both:
SET_BIT(GPIO_IBE_REG(port_num),pin_num);
case IntEvent_Rising:
SET_BIT(GPIO_IEV_REG(port_num),pin_num);
case IntEvent_Falling:
CLR_BIT(GPIO_IEV_REG(port_num),pin_num);
default :
break;
switch(ConfigPtr[u8_Pin_Counter]. PortIntMask)
case IntEvent_UnMaske:
SET_BIT(GPIO_IM_REG(port_num),pin_num);
case IntEvent_Masked:
CLR_BIT(GPIO_IM_REG(port_num),pin_num);
default :
break;

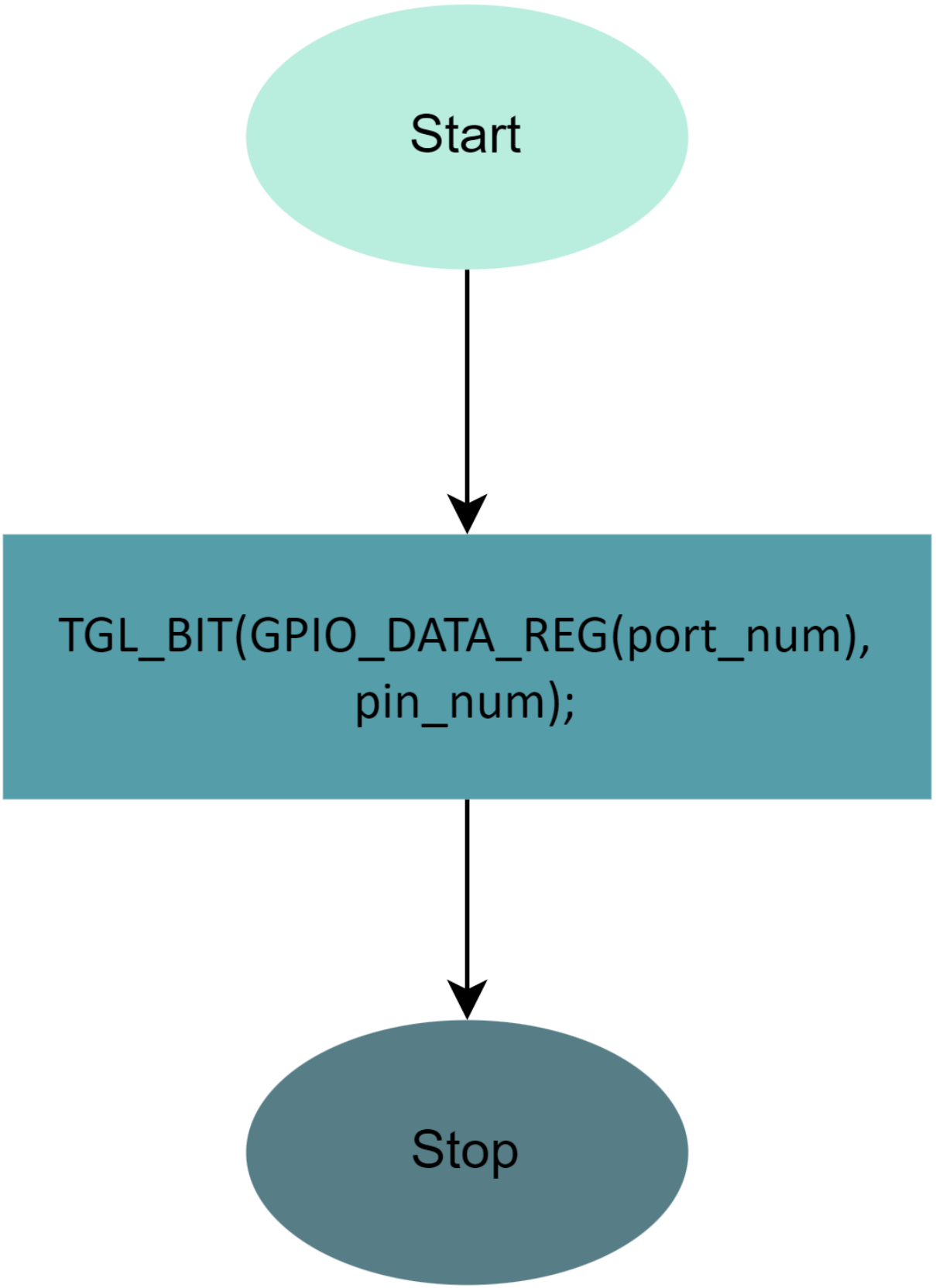
### 3.1.1.2 - GPIO_WRITE



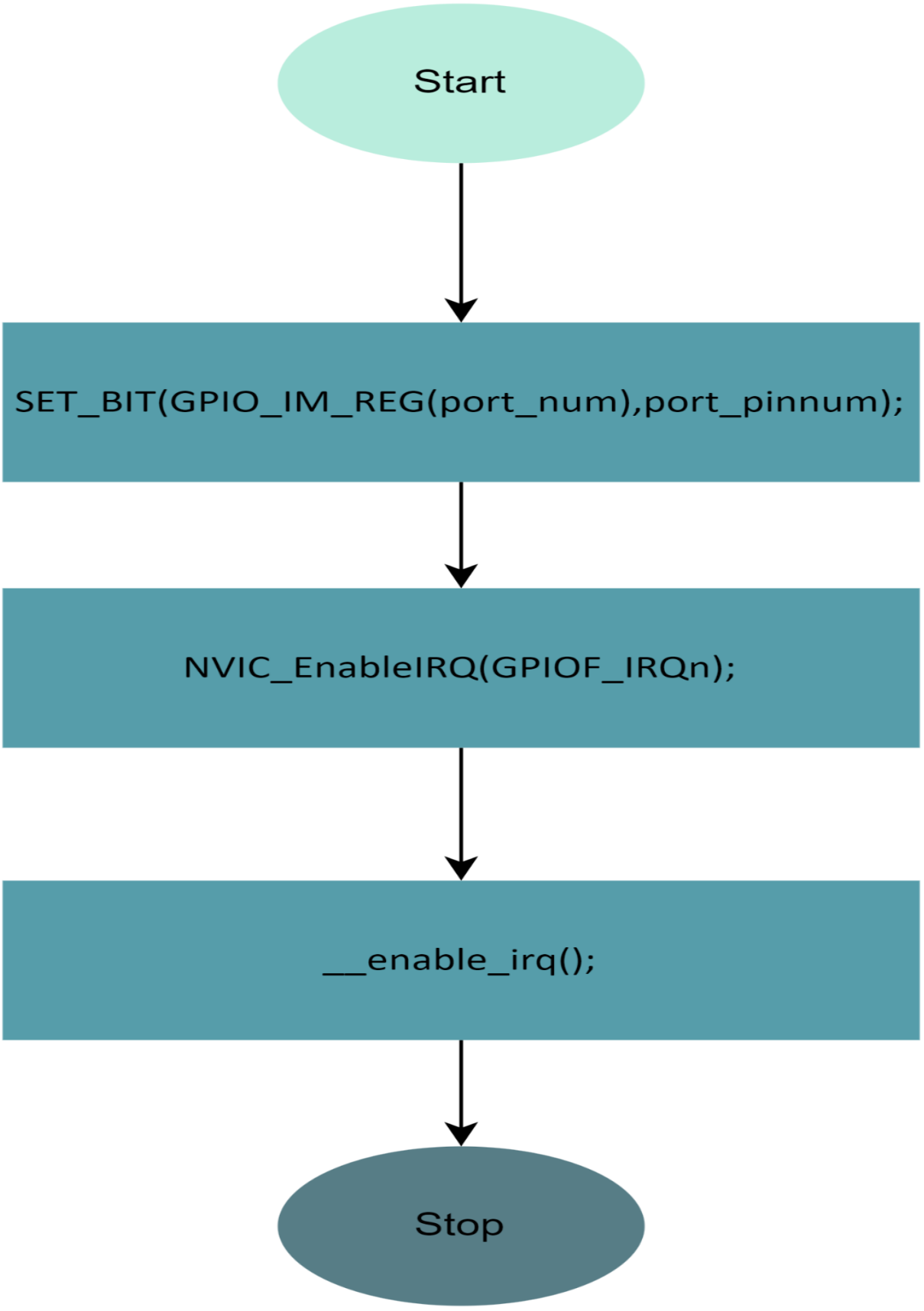
### 3.1.1.3 - GPIO_READ



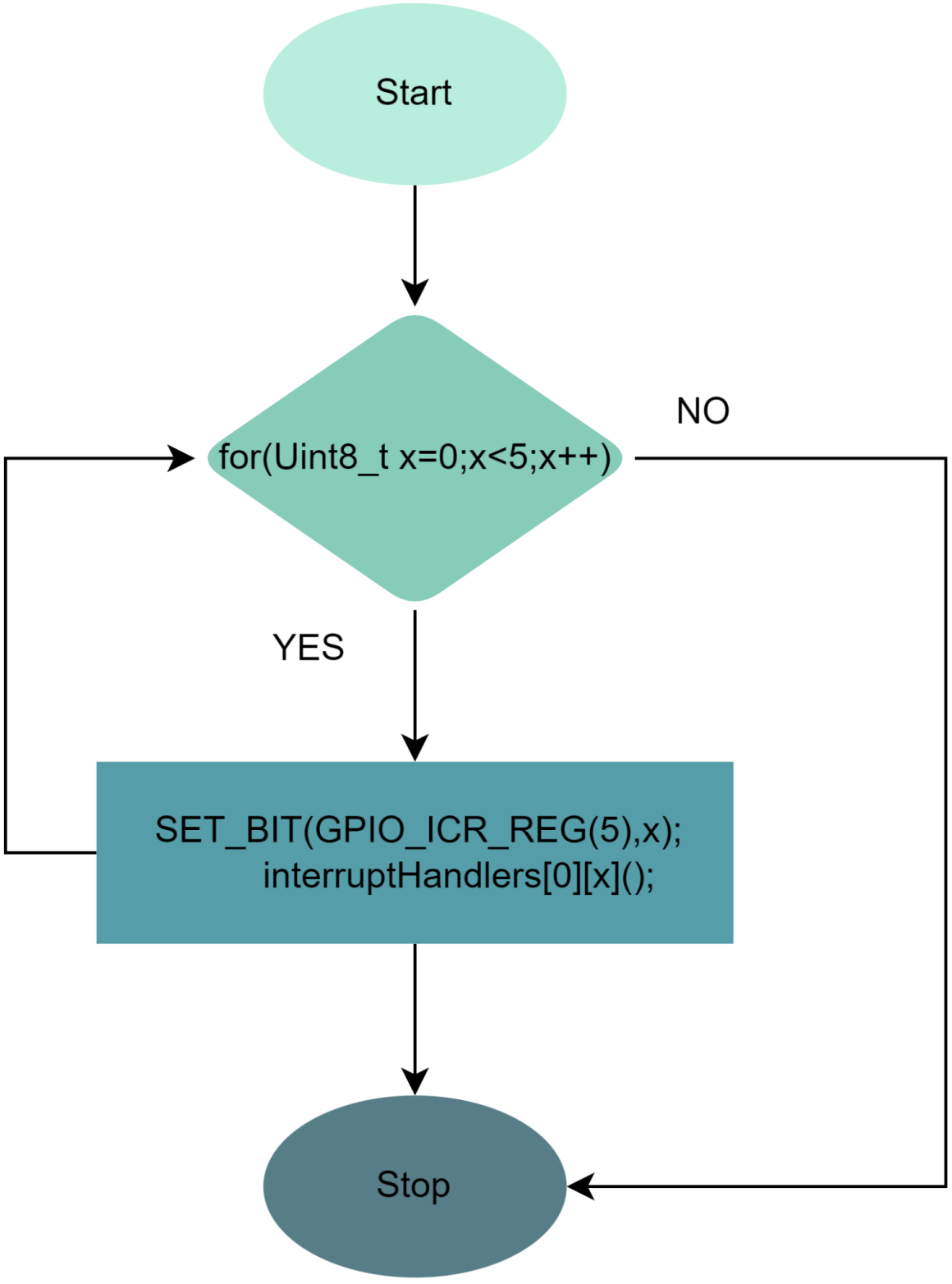
### 3.1.1.4 - GPIO_TOGGLE



### 3.1.1.5 - GPIO_ENABLE_INTERRUPT

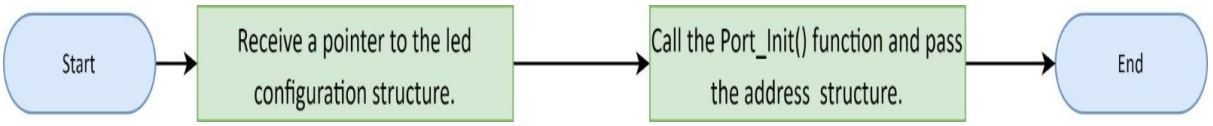


### 3.1.1.6 - GPIO_HANDLER

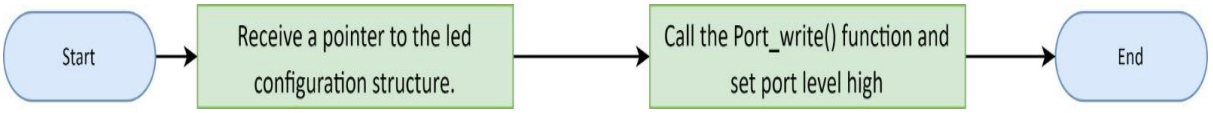


## 3.1.2 - LED FLOW Charts

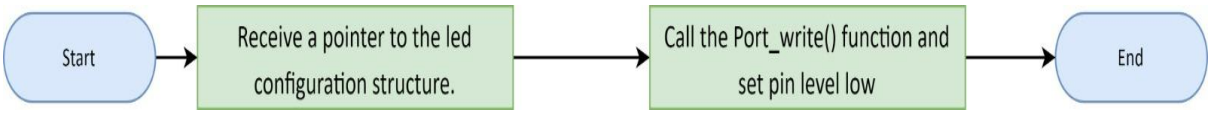
### 3.1.2.1 - LED_INIT



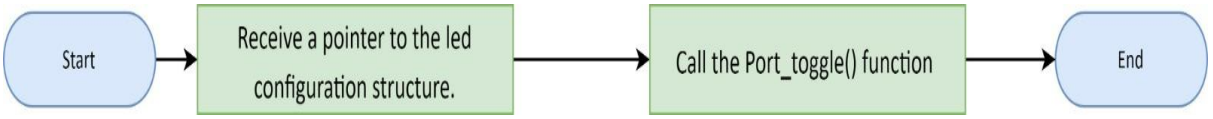

### 3.1.2.2 - LED_TURNON




### 3.1.2.3 - LED_TURNOFF

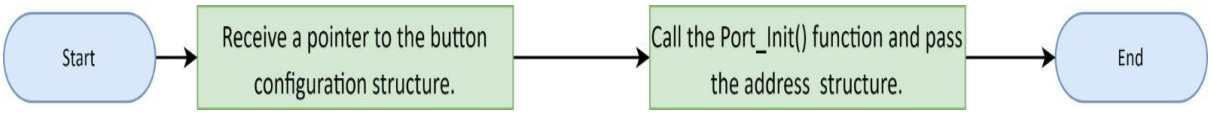



### 3.1.2.4 - LED_TOGGLE

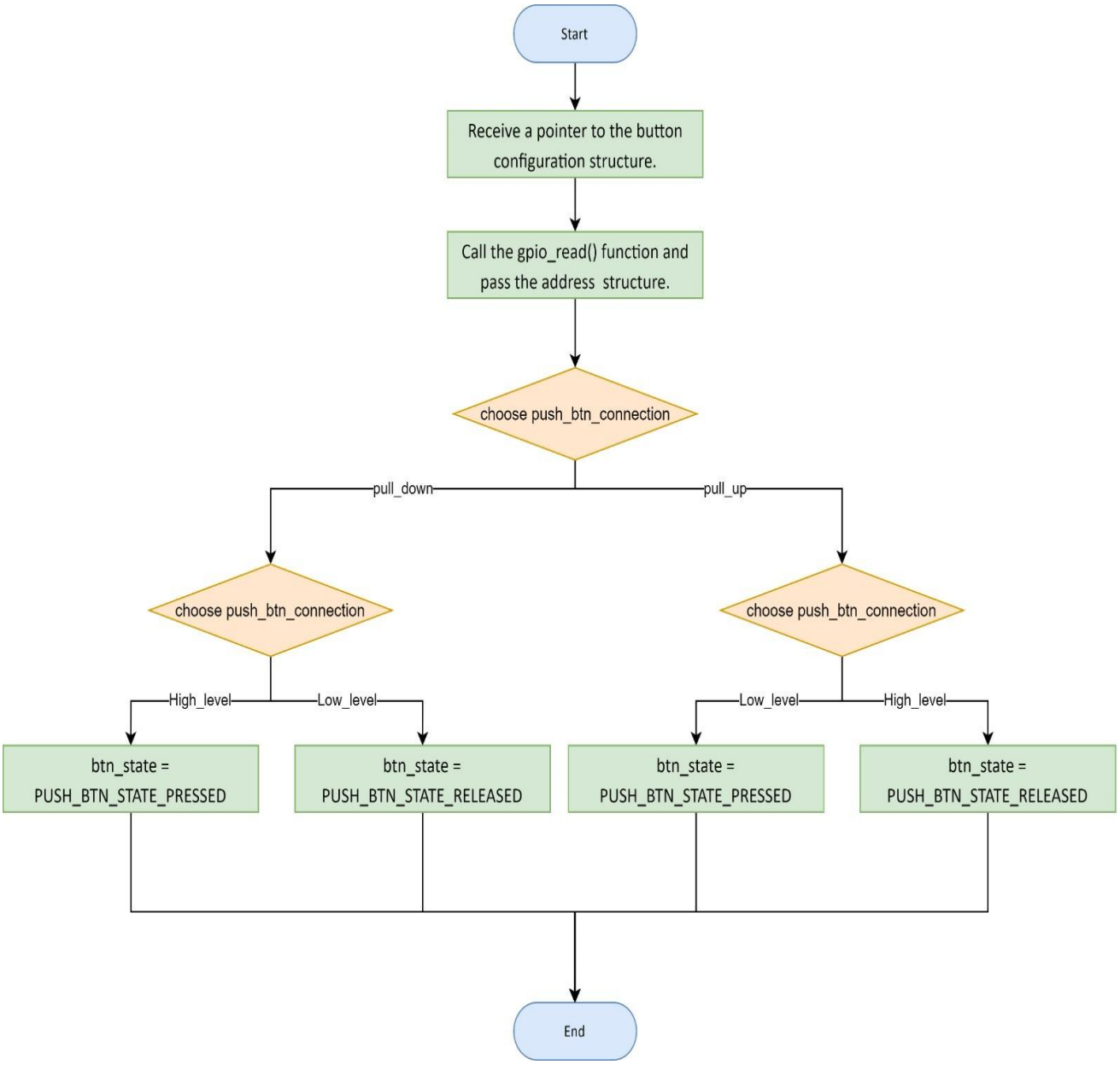


## 3.1.3 - PUSH_BUTTON Flow Charts

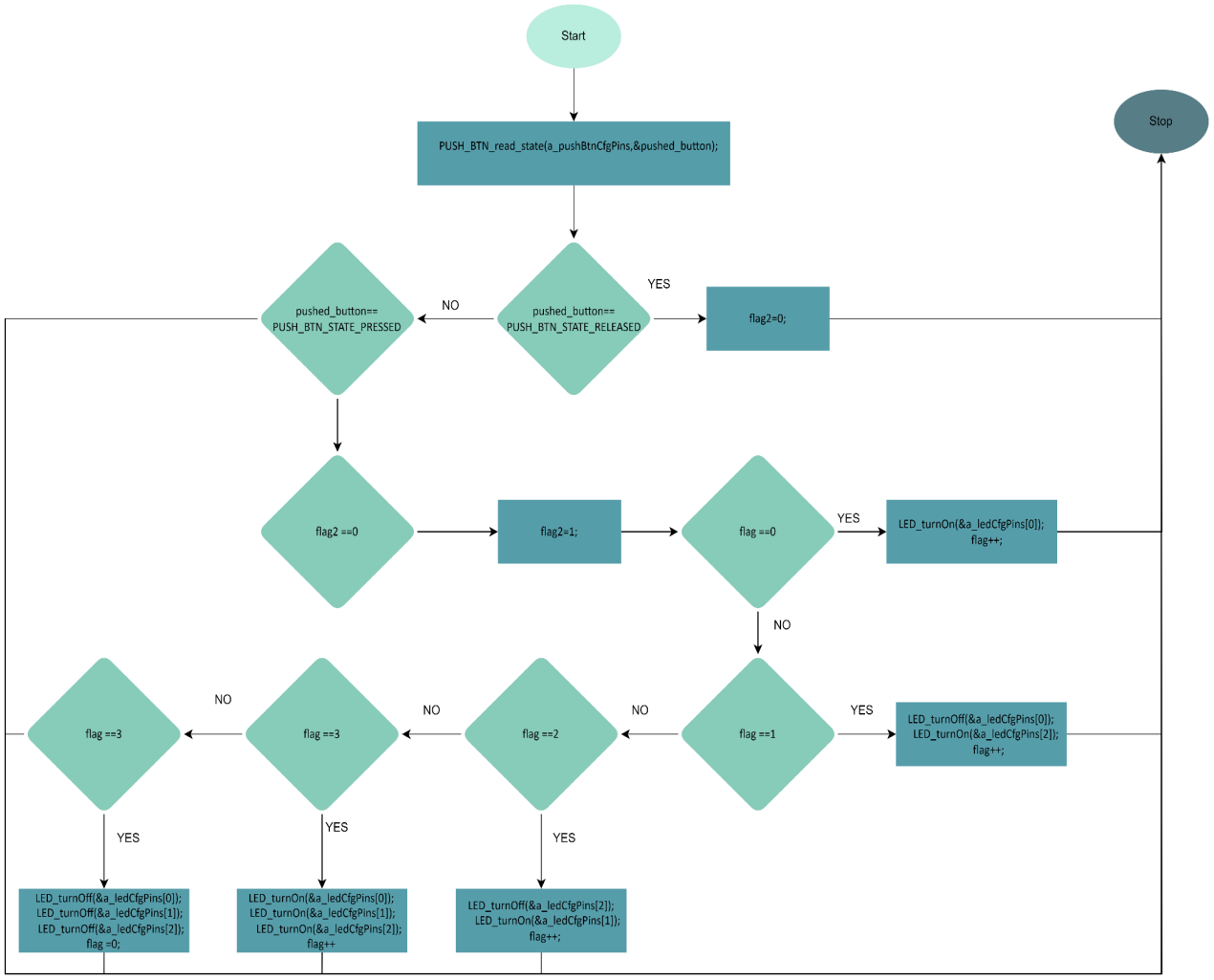
### 3.1.3.1 - PUSH_BUTTON_INIT




### 3.1.3.2 - PUSH_BUTTON_READ



## 3.1.4 - MAIN Flow Charts



## 3.2 - Pre Compiling Files

### 3.2.1 - GPIO Driver

```
#define PORT_PINS_NUMS 1
```

### 3.2.2 - LED Driver

```
#define  LED_PIN_CFG_ARRAY_SIZE    3
```

### 3.2.3 - PUSH_BUTTON Driver

```
#define  PUSH_BTN_PIN_CFG_ARRAY_SIZE    1
```

## 3.3 - Pre Linking Configuration

### 3.3.1 - GPIO Driver

None

### 3.3.2 - LED Driver

```
ST_led_pinCfg_t a_ledCfgPins[LED_PIN_CFG_ARRAY_SIZE] =
{
                {PORTF , PIN_1 , PORT_PIN_LEVEL_LOW},
   {PORTF , PIN_2 , PORT_PIN_LEVEL_LOW},
                {PORTF , PIN_3 , PORT_PIN_LEVEL_LOW}

};
```

### 3.3.3 - PUSH_BUTTON Driver

```
ST_PUSH_BTN_pinCfg_t a_pushBtnCfgPins[PUSH_BTN_PIN_CFG_ARRAY_SIZE] =
{
  {PORTF , PIN_4 , PORT_PIN_PUR}
};
```

# FOR FLOW CHART WITH HIGH QUALITY IT IS ON GITHUB