SPRINTS

19 JUNE

2023

RGB LED CONTROL V2.0 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

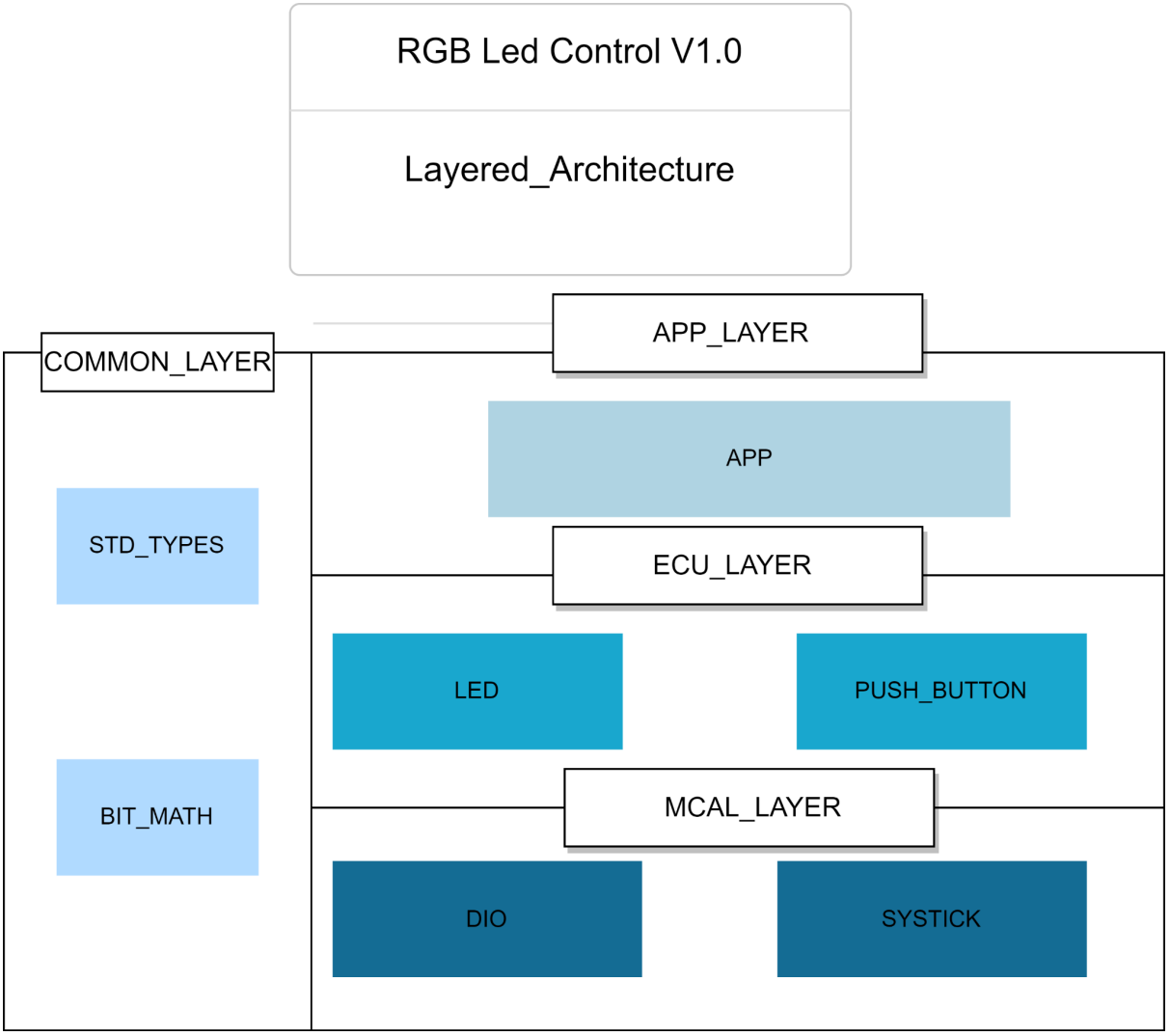
- Use the TivaC board
- Use SW1 as an input button
- Use the RGB LED

1.2.3 Software Requirements

- After the first press, the Red led is on for 1 second only
- After the second press, the Green Led is on for 1 second only
- After the third press, the Blue led is on for 1 second only
- After the fourth press, all LEDs are on for 1 second only
- 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 - SYSTICK TIMER Module

The systick timer is a 24-bit down counter that is integrated as part of the NVIC (Nested Vector Interrupt Controller) on all ARM Cortex-M devices. It can be used to generate periodic interrupts for operating systems, enabling task and context switching. The systick timer can be configured by setting the reload value, the clock source and the enable bit in the SYST_CSR register.

2.2.3 - 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.

2.2.4 - 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.

2.2.5 - 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 the GPIO pin with the given configuration parameters.

* @param arg_pincfg A pointer to a structure that contains the configuration parameters for the GPIO pin.

* @return An enumeration value that indicates the system state after the initialization.

*/

ENU_GPIO_systemState_t GPIO_init(ST_dio_pinCfg_t *arg_pincfg);

/**

* @brief Enables the interrupt for the GPIO port that contains the given pin.

* @param arg_pincfg A pointer to a structure that contains the configuration parameters for the GPIO pin.

* @return An enumeration value that indicates the system state after enabling the interrupt.

*/

ENU_GPIO_systemState_t GPIO_interruptPortEnable(ST_dio_pinCfg_t *arg_pincfg);

/**

* @brief Disables the interrupt for the GPIO port that contains the given pin.

* @param arg_pincfg A pointer to a structure that contains the configuration parameters for the GPIO pin.

* @return An enumeration value that indicates the system state after disabling the interrupt.

*/

ENU_GPIO_systemState_t GPIO_interruptPortDisable(ST_dio_pinCfg_t *arg_pincfg);

/**

* @brief Writes a logic value (high or low) to the GPIO pin.

* @param arg_pincfg A pointer to a structure that contains the configuration parameters for the GPIO pin.

* @param arg_logicValue An enumeration value that indicates the logic value to be written to the GPIO pin.

* @return An enumeration value that indicates the system state after writing the logic value.

*/

ENU_GPIO_systemState_t GPIO_writeLogic(ST_dio_pinCfg_t *arg_pincfg , ENU_GPIO_logic_t arg_logicValue);

/**

* @brief Reads a logic value (high or low) from the GPIO pin.

* @param arg_pincfg A pointer to a structure that contains the configuration parameters for the GPIO pin.

* @param arg_logicValue A pointer to an enumeration value that stores the logic value read from the GPIO pin.
* @return An enumeration value that indicates the system state after reading the logic value.
*/

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

/**
* @brief Toggles a logic value (high or low) of the GPIO pin.
* @param arg_pincfg A pointer to a structure that contains the configuration parameters for the GPIO pin.
* @return An enumeration value that indicates the system state after toggling the logic value.
*/

ENU_GPIO_systemState_t GPIO_toggleLogic(ST_dio_pinCfg_t *arg_pincfg);

2.3.2 - SYSTICK TIMER Driver

/**
* @brief Initializes the systick timer with the given configuration parameters.
* @param systickCfg A pointer to a structure that contains the configuration parameters for the systick timer.
* @return An enumeration value that indicates the system state after the initialization.
*/

ENU_SYSTICK_systemState_t SYSTICK_init(STR_SYSTICK_cfg_t *systickCfg);

/**
* @brief Enables the systick timer interrupt.
* @return An enumeration value that indicates the system state after enabling the interrupt.
*/

ENU_SYSTICK_systemState_t SYSTICK_enableInterrupt(void);

/**
* @brief Disables the systick timer interrupt.
* @return An enumeration value that indicates the system state after disabling the interrupt.
*/

ENU_SYSTICK_systemState_t SYSTICK_disableInterrupt(void);

/**
* @brief Sets the delay time for the systick timer in milliseconds.
* @param arg_delayInMs The delay time in milliseconds.

```
 * @return An enumeration value that indicates the system state after setting the delay time.
 */
ENU_SYSTICK_systemState_t SYSTICK_setDelayMs(Uint32_t arg_delayInMs);
```

2.3.3 - LED Driver

```
/**
 * @brief Initializes the LED with the given pin configuration parameters.
 * @param led A pointer to a structure that contains the pin configuration parameters for the LED.
 * @return An enumeration value that indicates the system state after the initialization.
 */
ENU_LED_systemState_t LED_initialize(const ST_led_pinCfg_t *led);

/**
 * @brief Turns on the LED by writing a high logic value to the pin.
 * @param led A pointer to a structure that contains the pin configuration parameters for the LED.
 * @return An enumeration value that indicates the system state after turning on the LED.
 */
ENU_LED_systemState_t LED_turnOn(const ST_led_pinCfg_t *led);

/**
 * @brief Turns off the LED by writing a low logic value to the pin.
 * @param led A pointer to a structure that contains the pin configuration parameters for the LED.
 * @return An enumeration value that indicates the system state after turning off the LED.
 */
ENU_LED_systemState_t LED_turnOff(const ST_led_pinCfg_t *led);

/**
 * @brief Toggles the LED by writing the opposite logic value to the pin.
 * @param led A pointer to a structure that contains the pin configuration parameters for the LED.
```

```
 * @return An enumeration value that indicates the system state after toggling the LED.
 */
ENU_LED_systemState_t LED_toggle(const ST_led_pinCfg_t *led);
```

2.3.4 - PUSH_BOTTON Driver

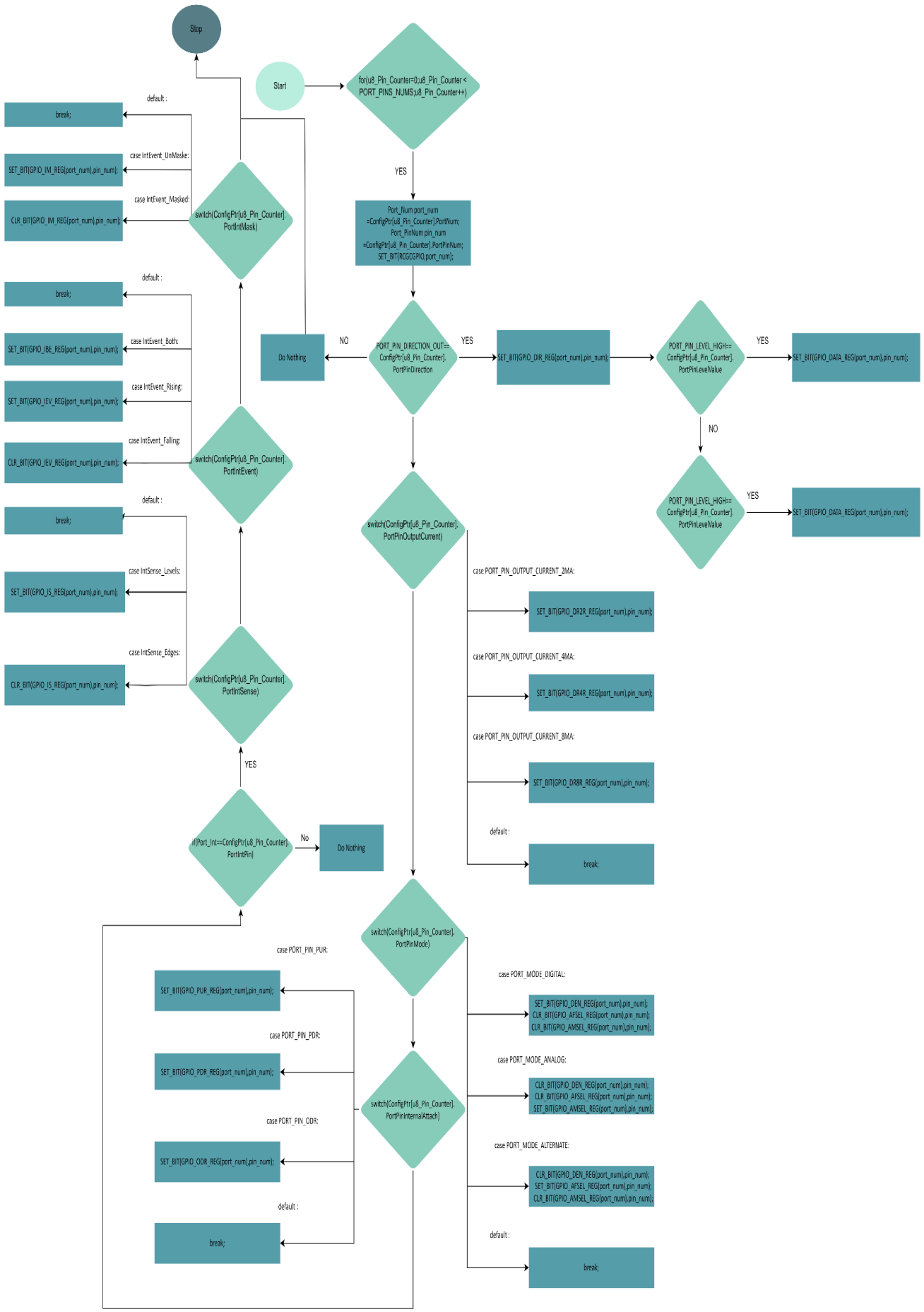
```
/**
 * @brief Initializes the push button with the given pin configuration parameters.
 * @param btn A pointer to a structure that contains the pin configuration parameters for the push button.
 * @return An enumeration value that indicates the system state after the initialization.
 */
ENU_PUSH_BTN_systemState_t PUSH_BTN_intialize(const ST_PUSH_BTN_pinCfg_t *btn);

/**
 * @brief Reads the current state of the push button (pressed or released).
 * @param btn A pointer to a structure that contains the pin configuration parameters for the push button.
 * @param btn_state A pointer to an enumeration value that stores the current state of the push button.
 * @return An enumeration value that indicates the system state after reading the push button state.
 */
ENU_PUSH_BTN_systemState_t PUSH_BTN_read_state(const ST_PUSH_BTN_pinCfg_t *btn , ENU_PUSH_BTN_state_t *btn_state);
```

3 - Low-Level Design

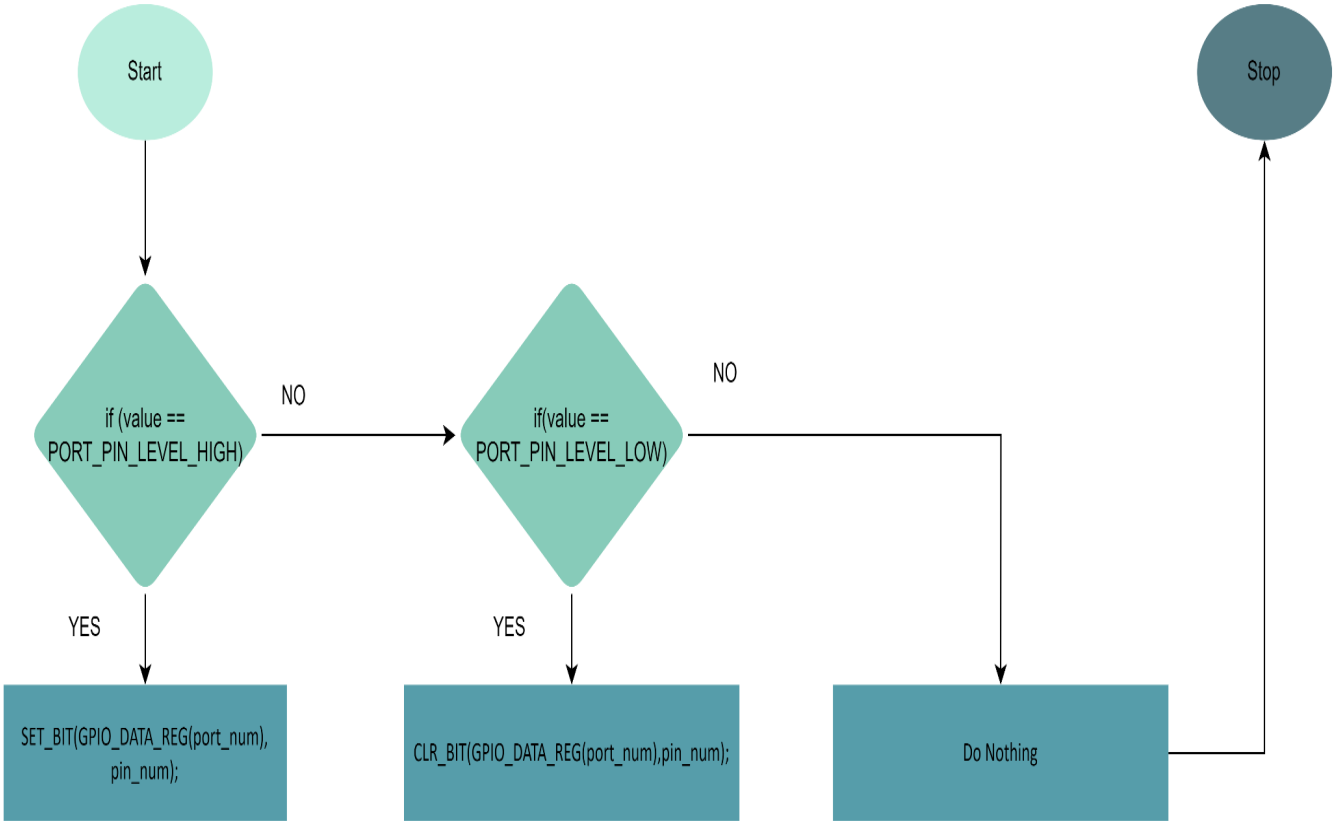
3.1 - Module Flow Charts

3.1.1 - GPIO Flow Charts

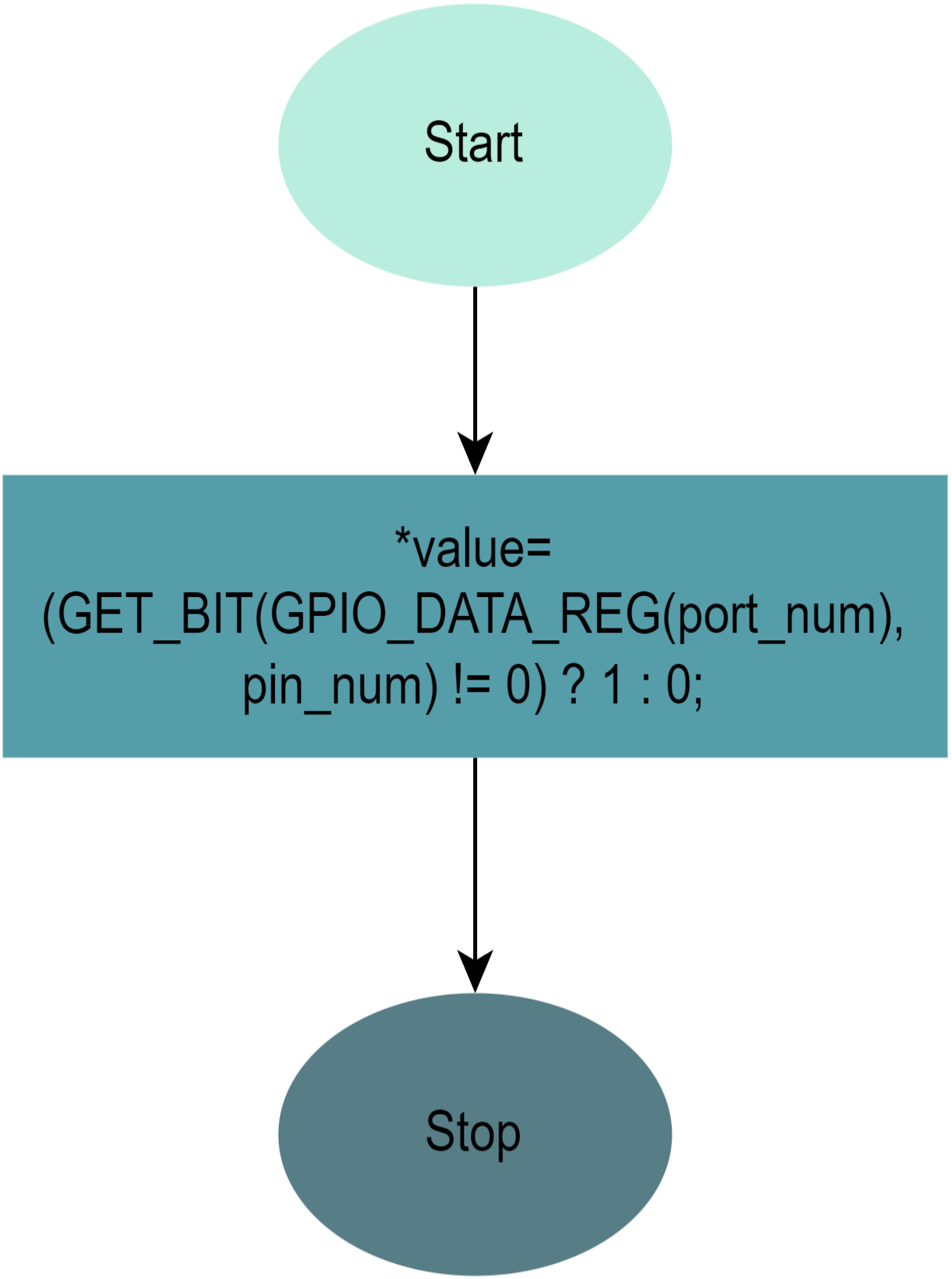
3.1.1.1 - GPIO_INIT


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;

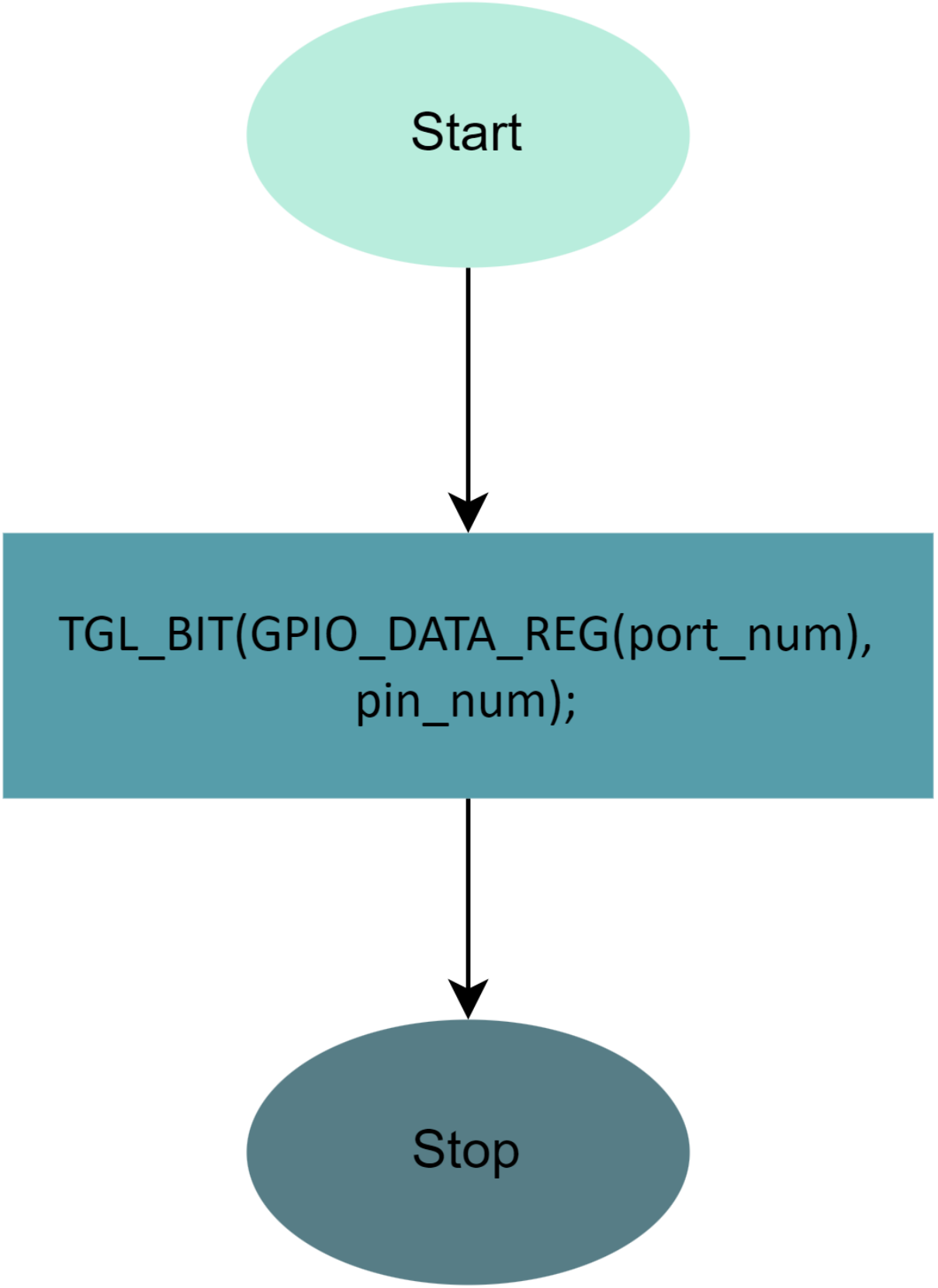
3.1.1.2 - GPIO_WRITE



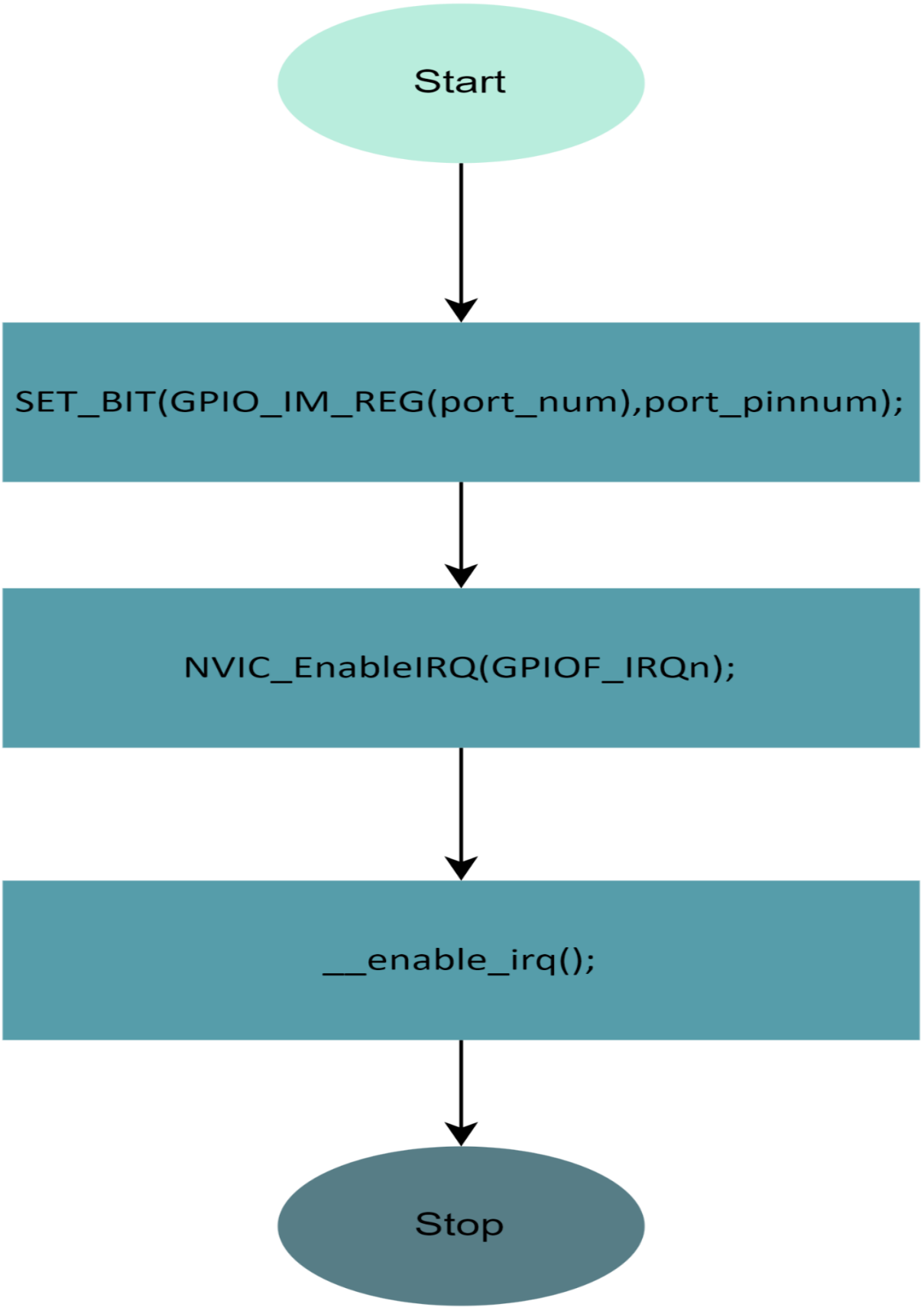
3.1.1.3 - GPIO_READ



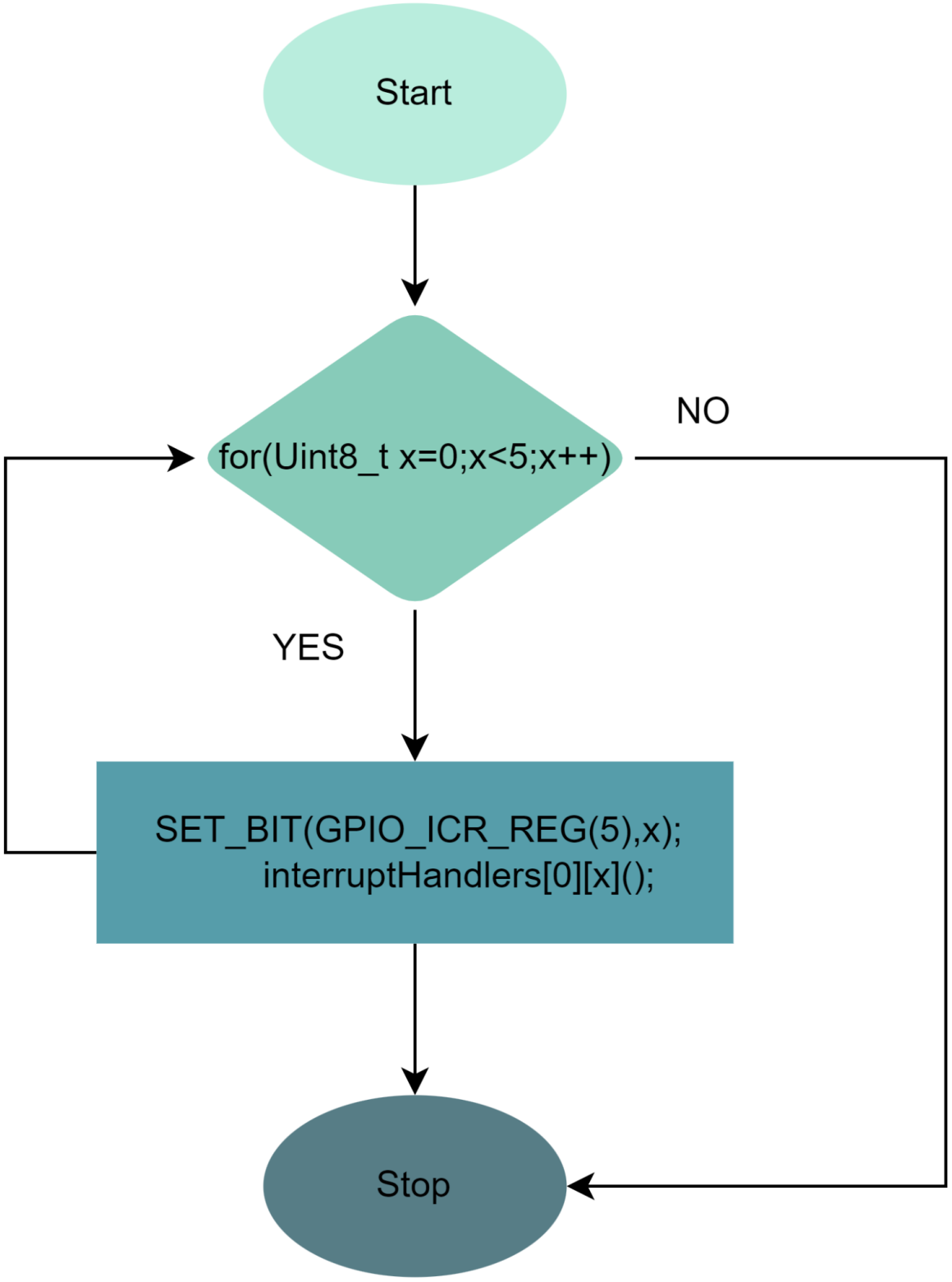
3.1.1.4 - GPIO_TOGGLE



3.1.1.5 - GPIO_ENABLE_INTERRUPT

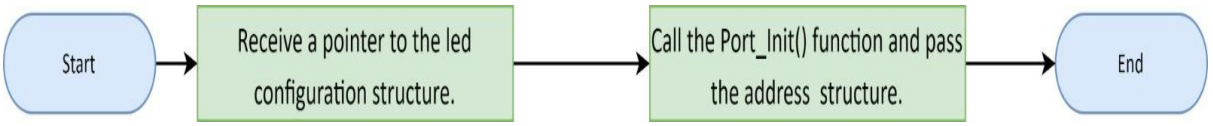


3.1.1.6 - GPIO_HANDLER

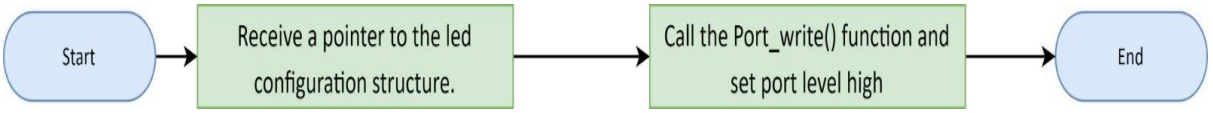


3.1.2 - LED FLOW Charts

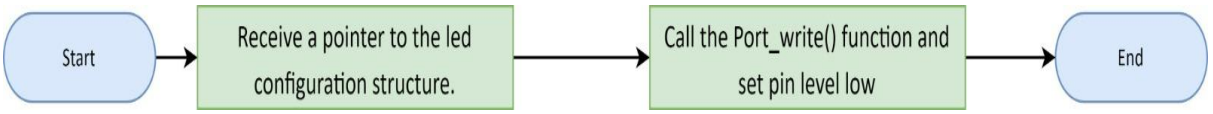
3.1.2.1 - LED_INIT



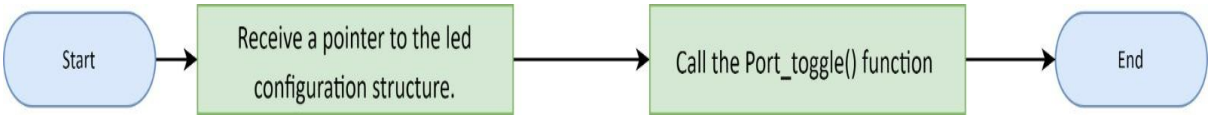
3.1.2.2 - LED_TURNON



3.1.2.3 - LED_TURNOFF

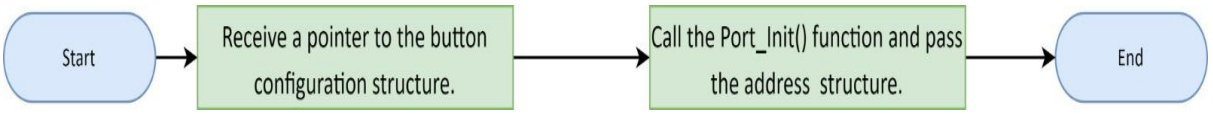


3.1.2.4 - LED_TOGGLE

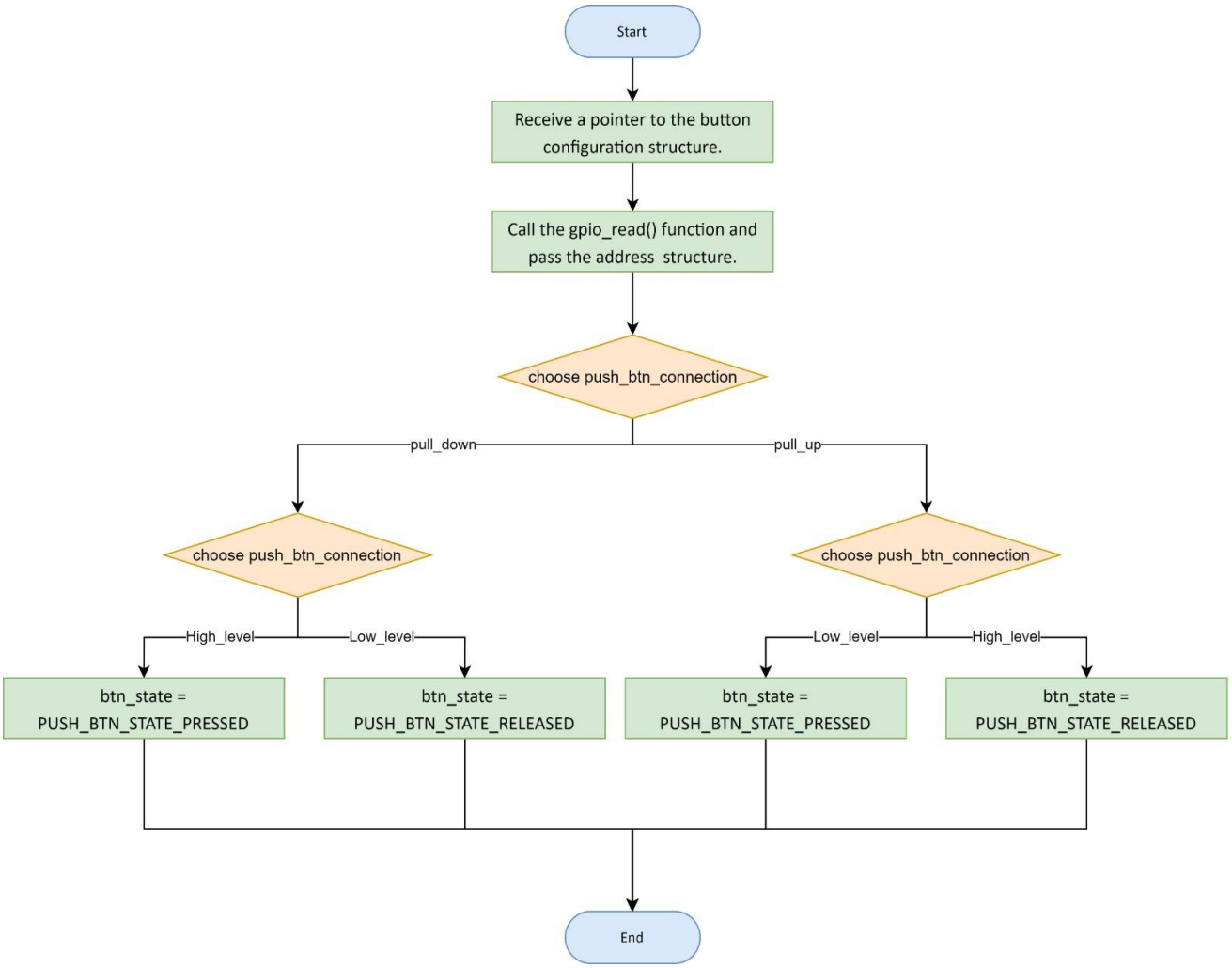


3.1.3 - PUSH_BUTTON Flow Charts

3.1.3.1 - PUSH_BUTTON_INIT

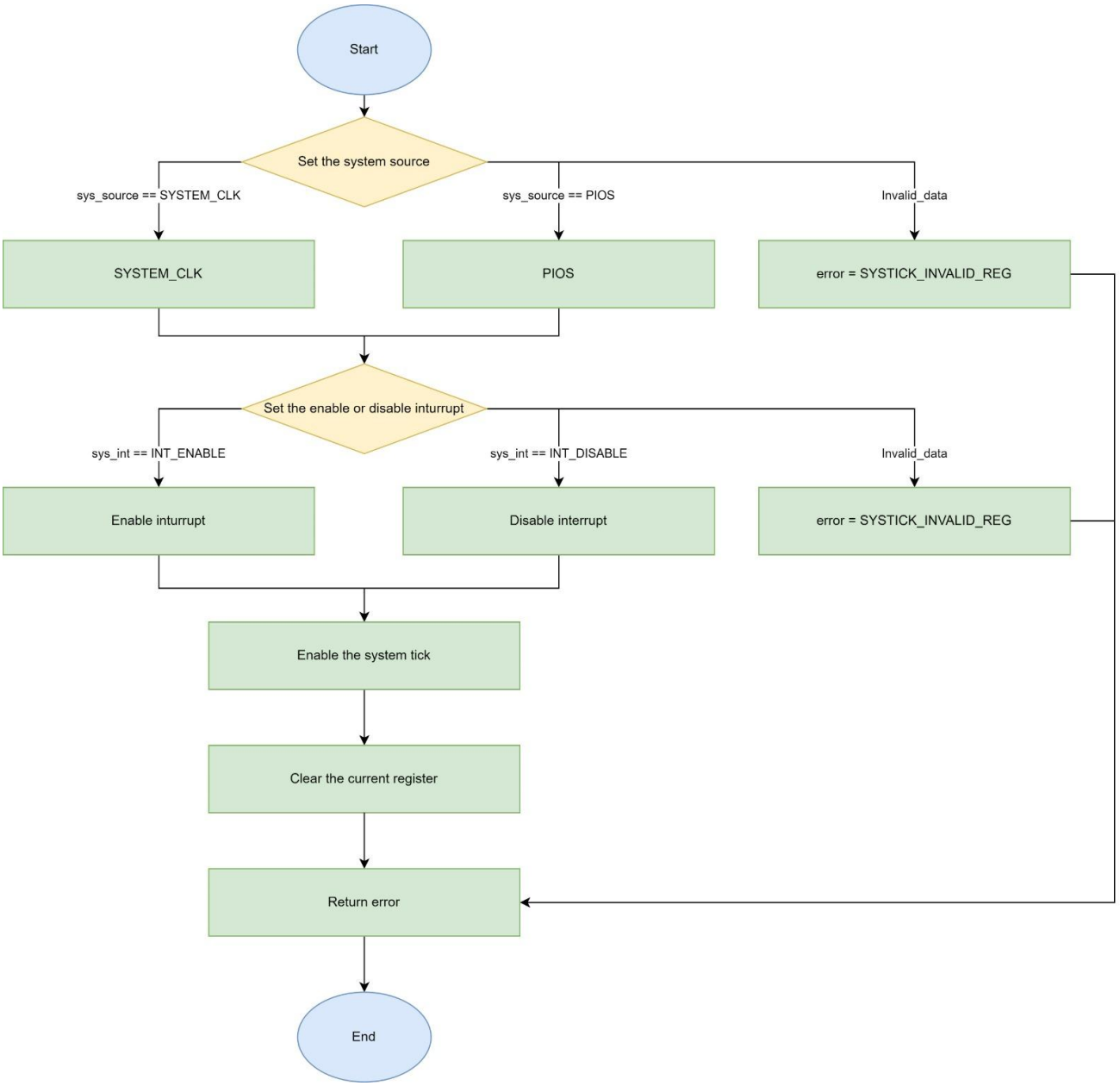


3.1.3.2 - PUSH_BUTTON_READ

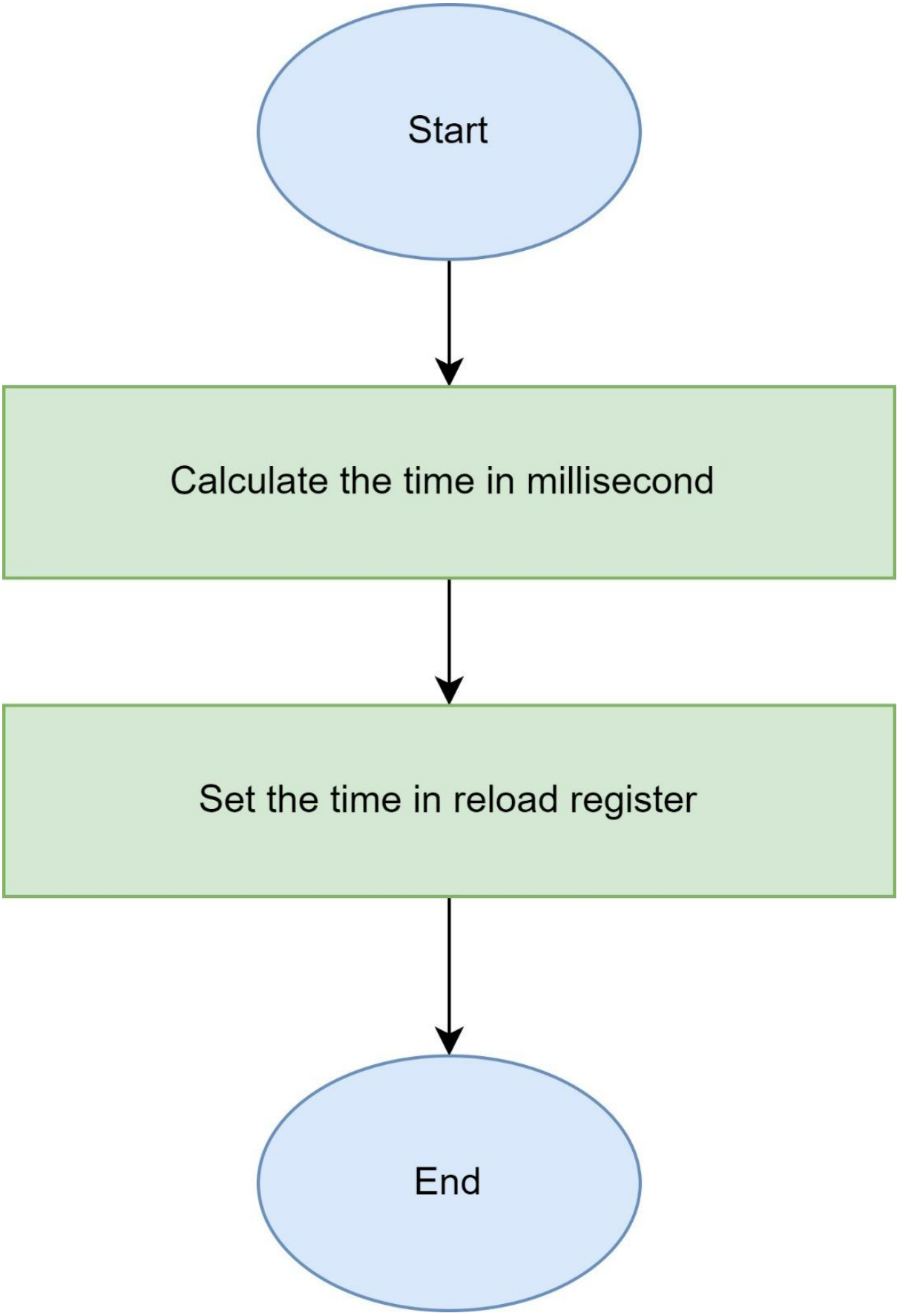


3.1.4 - SYSTICK TIMER Flow Chart

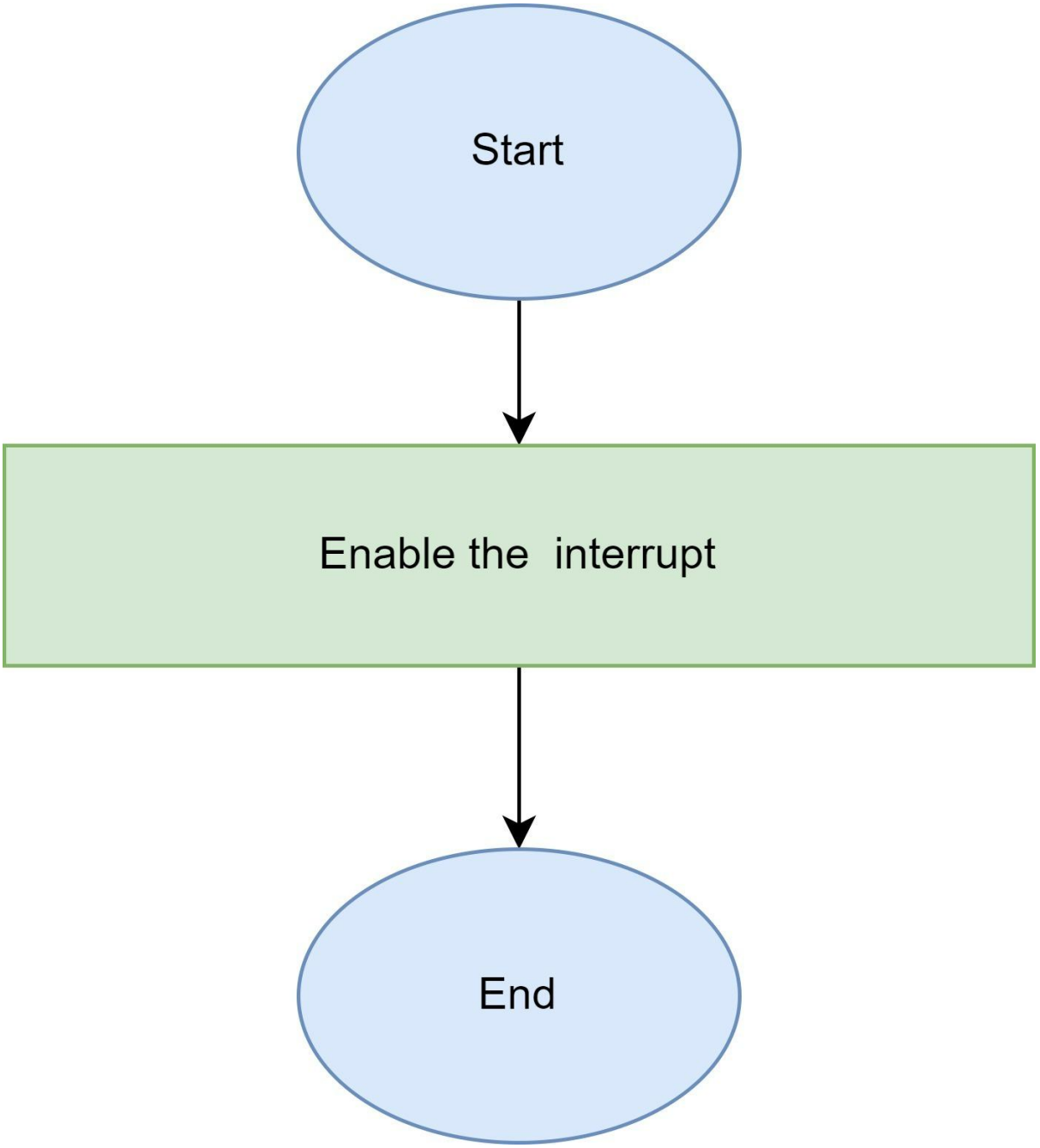
3.1.4.1 - SYSTICK_init



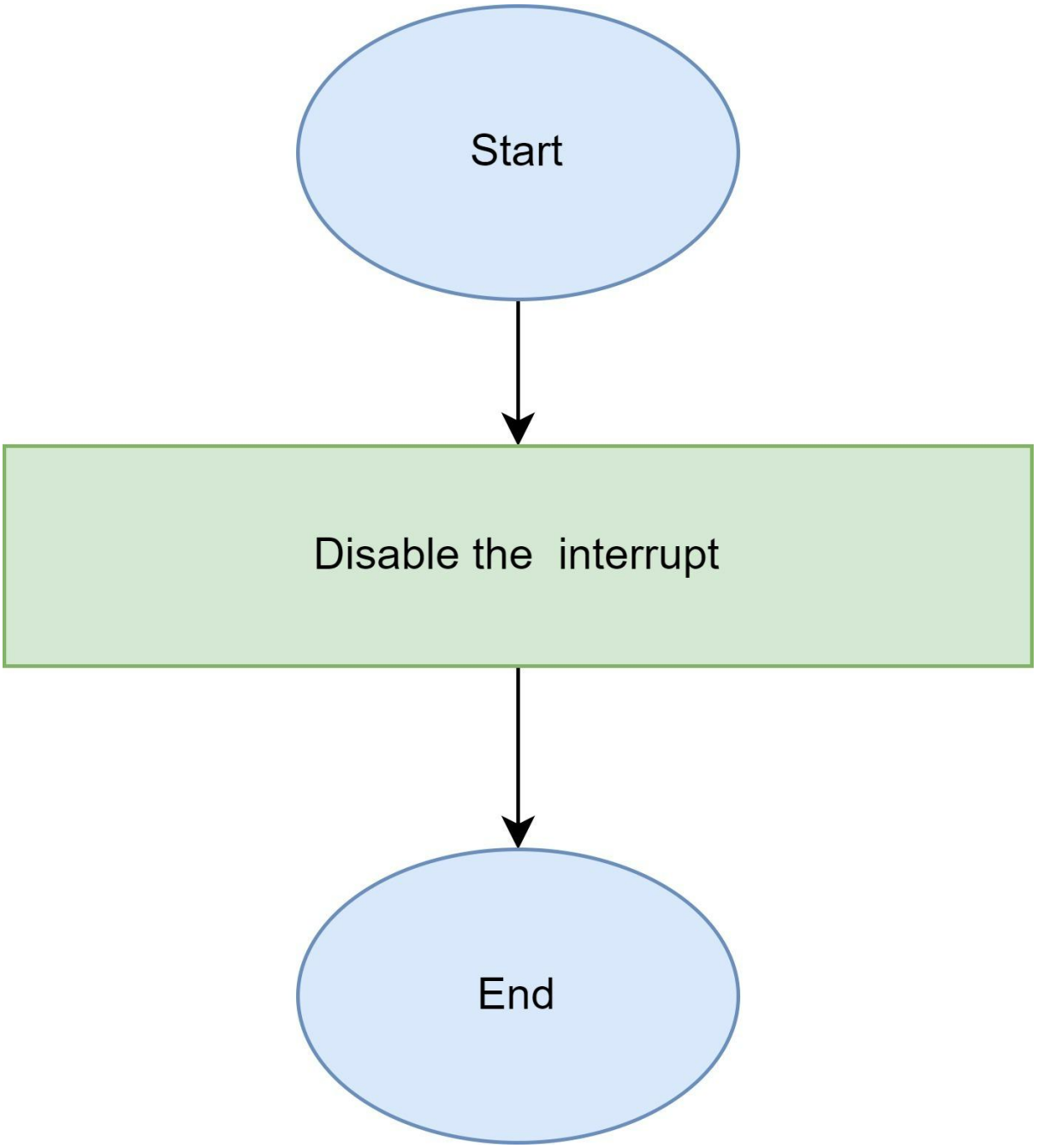
3.1.4.2 - SYSTICK_setDelayInMs



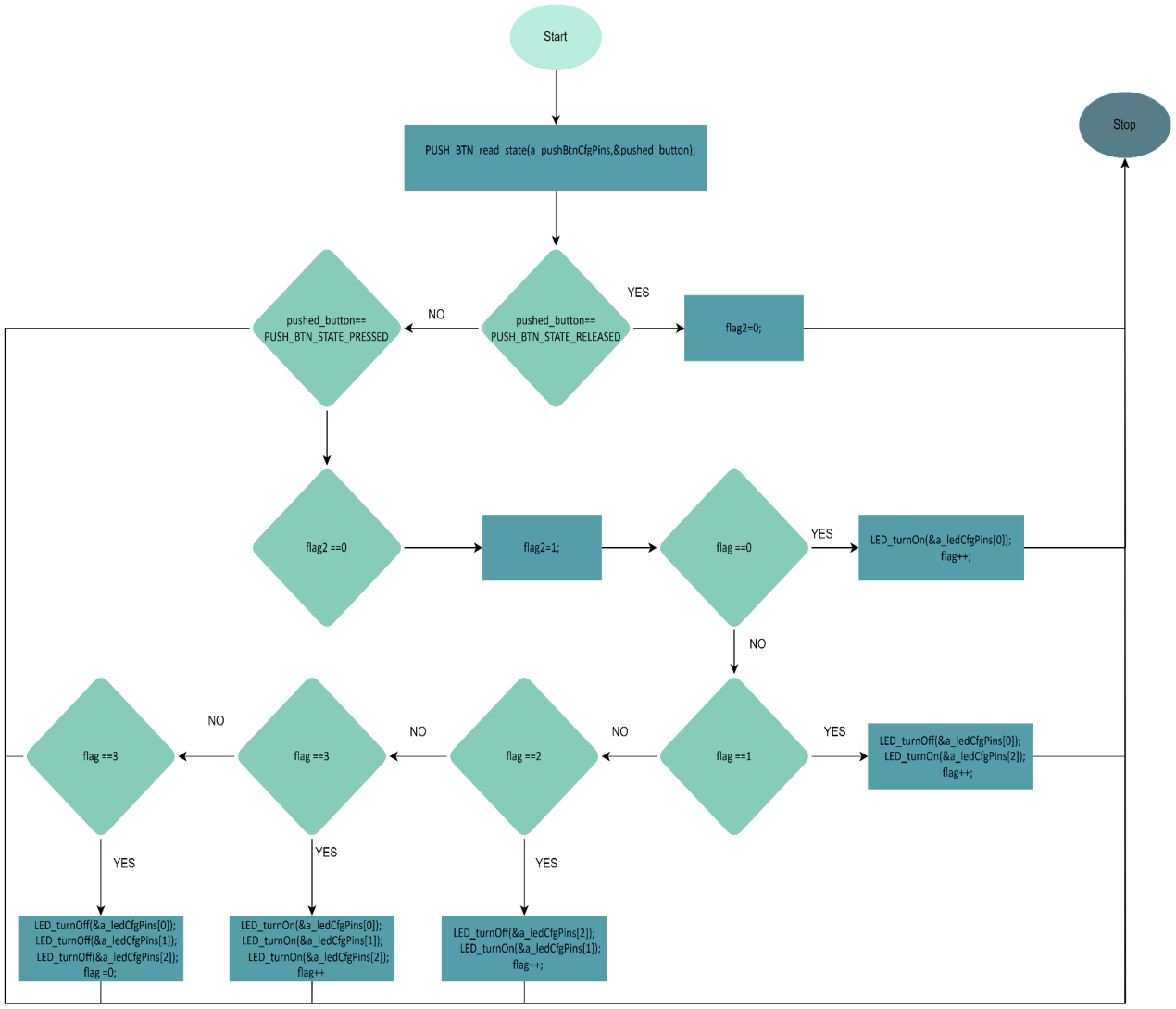
3.1.4.3 - SYSTICK_interruptEnable



3.1.4.4 - SYSTICK_interruptDisable



3.1.5 - 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
```

3.2.4 - SYSTICK_TIMER Driver

```
#define  SYSTICK_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}
};
```

3.3.4 - SYSTICK_TIMER Driver

```
STR_SYSTICK_cfg_t a_systickCfg[SYSTICK_CFG_ARRAY_SIZE] =
{
{SYSTICK_SYSTEM_CLOCK , SYSTICK_INTERRUPT_ENABLED ,  SYSTICK_ENABLED
, systickCallback}
};
```

FOR FLOW CHART WITH HIGH QUALITY IT IS ON GITHUB