DESIGN & IMPLEMENT A RGB LED CONTROL

Youssef Abbas

Arafa Arafa

SPRINTS

Contents

Project Introduction:

The RGB LED Control System is a hardware and software solution designed to control the behavior of an RGB LED based on user interaction with a button. The system utilizes the TivaC board as the target hardware platform and provides an intuitive interface for users to switch between different LED colors and states.

Hardware Requirements:

- TivaC board: The system is implemented on the TivaC development board, which serves as the main hardware platform.
- SW1 Button: The SW1 button is used as an input button to trigger LED state changes.
- RGB LED: The system controls an RGB LED, allowing for various color combinations and lighting effects.

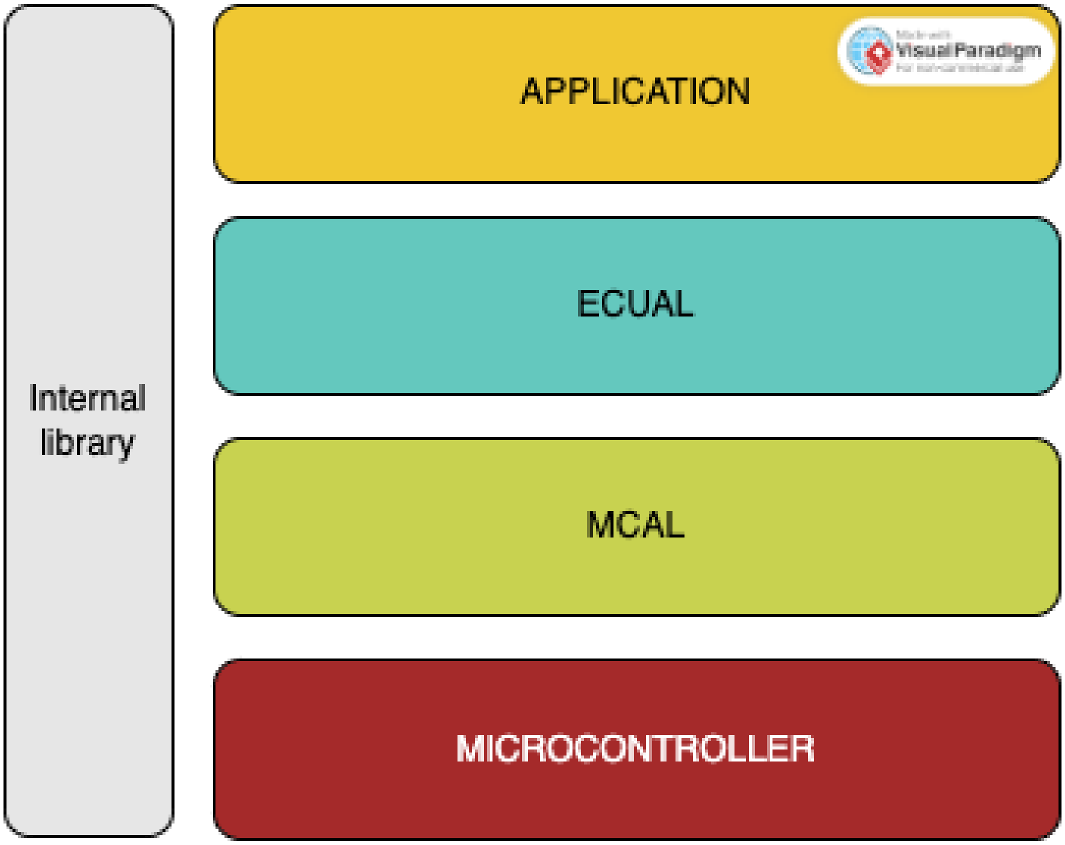
Software Requirements:

- Initialization: The system initializes by setting up the necessary configurations, including GPIO pins, interrupts, and timers.
- RGB LED Control: The RGB LED is initially turned off. Each press of the SW1 button results in a different LED state change, following a specific sequence:
    1. First press: The Red LED turns on.
    2. Second press: The Green LED turns on.
    3. Third press: The Blue LED turns on.
    4. Fourth press: All LEDs (Red, Green, and Blue) turn on simultaneously.
    5. Fifth press: All LEDs are turned off.
    6. Sixth press: The sequence repeats from the beginning.
- Button Driver: The Button driver detects button presses and triggers the corresponding LED state changes.
- GPIO Driver: The GPIO driver handles the configuration and control of GPIO pins to interface with the RGB LED.
- LED Driver: The LED driver provides functions to set the state of individual LEDs (Red, Green, and Blue) and control the RGB LED's behavior.

High Level Design:

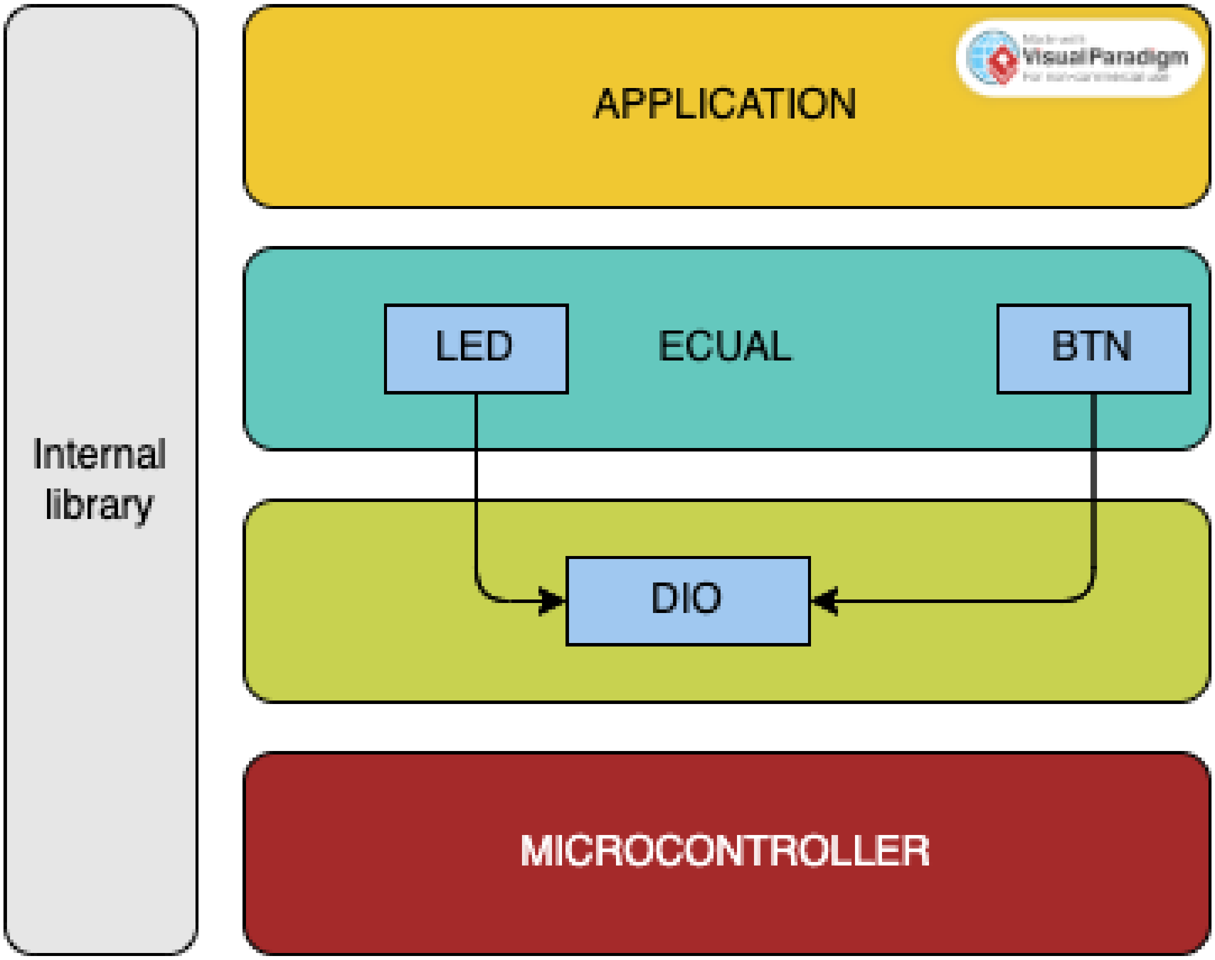
Layered architecture:

1. Application
2. ECUAL
3. MCAL
4. Microcontroller



Module Description

1. Application
2. ECUAL
    a. LED
    b. Button
3. MCAL
    a. DIO
4. Microcontroller



Driver Documentations

Button

Module Description The push button module provides functions for initializing a push button, reading its state, and configuring its properties. It utilizes the DIO (Digital Input/Output) module for pin configuration and manipulation.

Macro Declarations No macro declarations are included in the provided code.

Macro Function Declarations No macro function declarations are included in the provided code.

Data Types Declarations

- **btn_enu_btn_state_t**: An enumerated type representing the possible states of the button. It has two values: BUTTON_RELEASED and BUTTON_PRESSED.
- **btn_enu__btn_active_t**: An enumerated type representing the active state of the button. It has two values: BUTTON_ACTIVE_LOW and BUTTON_ACTIVE_HIGH.
- **button_str_btn_config_t**: A structure representing the configuration of the button. It contains the following members:
  - **port_name**: An instance of the **dio_enu_portx_t** enum, representing the port name of the button pin.
  - **pin**: An instance of the **dio_enu_pinx_t** enum, representing the pin number of the button.
  - **button_state**: An instance of the **btn_enu_btn_state_t** enum, representing the initial state of the button.
  - **button_active**: An instance of the **btn_enu__btn_active_t** enum, representing the active state of the button.

Function Declarations

- **button_initializa**: Initializes the push button by configuring its pin and setting the initial state. It takes a pointer to a **button_str_btn_config_t** structure as an argument. Returns a value of type **brn_enu_return_state_t** indicating the success or failure of the initialization.
- **button_read_state**: Reads the state of the push button and updates the **btn_enu_btn_state_t** variable pointed to by **ptr_enu_btn_state**. It takes a pointer to a **button_str_btn_config_t** structure and a pointer to a **btn_enu_btn_state_t** variable as arguments. Returns a value of type **brn_enu_return_state_t** indicating the success or failure of reading the state.

LED

The LED module provides functions for initializing an LED, turning it on and off, and toggling its state. It utilizes the DIO (Digital Input/Output) module for pin configuration and manipulation.

Macro Declarations No macro declarations are included in the provided code.

Macro Function Declarations No macro function declarations are included in the provided code.

Data Types Declarations

- **led_enu_status_t**: An enumerated type representing the possible states of the LED. It has two values: LED_OFF and LED_ON.
- **led_str_led_config_t**: A structure representing the configuration of the LED. It contains the following members:
    - **port_name**: An instance of the **dio_enu_portx_t** enum, representing the port name of the LED pin.
    - **pin**: An instance of the **dio_enu_pinx_t** enum, representing the pin number of the LED.
    - **led_status**: An instance of the **led_enu_status_t** enum, representing the initial state of the LED.

Function Declarations

- **led_initialization**: Initializes the LED by configuring its pin and setting the initial state. It takes a pointer to a **led_str_led_config_t** structure as an argument. Returns a value of type **led_enu_return_state_t** indicating the success or failure of the initialization.
- **led_turn_on**: Turns on the LED by setting the appropriate pin to the active state. It takes a pointer to a **led_str_led_config_t** structure as an argument. Returns a value of type **led_enu_return_state_t** indicating the success or failure of turning on the LED.
- **led_turn_off**: Turns off the LED by setting the appropriate pin to the inactive state. It takes a pointer to a **led_str_led_config_t** structure as an argument. Returns a value of type **led_enu_return_state_t** indicating the success or failure of turning off the LED.
- **led_toggle**: Toggles the state of the LED. If the LED is currently on, it will be turned off, and vice versa. It takes a pointer to a **led_str_led_config_t** structure as an argument. Returns a value of type **led_enu_return_state_t** indicating the success or failure of toggling the LED.

GPIO

The DIO interface module provides functions and data types for configuring and manipulating digital input/output pins. It allows you to enable clock for a specific port, initialize pins, set pin states, toggle pin states, and read pin states.

Data Types Declarations

- **dio_enu_portx_t**: An enumerated type representing the available ports. It includes values from DIO_PORTA to DIO_PORTF, and MAX_INVALID_PORT.
- **dio_enu_pinx_t**: An enumerated type representing the available pins. It includes values from DIO_PIN_0 to DIO_PIN_7, and MAX_INVALID_PIN.
- **dio_enu_pin_state_t**: An enumerated type representing the possible states of a pin. It has two values: DIO_PIN_LOW_STATE and DIO_PIN_HIGH_STATE.
- **dio_enu_pin_mode_t**: An enumerated type representing the mode of a pin. It includes values DIO_PIN_INPUT, DIO_PIN_OUTPUT, DIO_PIN_AFM, and DIO_PIN_ANALOG.
- **dio_enu_output_type_t**: An enumerated type representing the output type of a pin. It includes values DIO_PIN_OUTPUT_PUSH_PULL and DIO_PIN_OUTPUT_OPEN_DRAIN.
- **dio_enu_output_current_t**: An enumerated type representing the output current of a pin. It includes values DIO_PIN_2MA, DIO_PIN_4MA, and DIO_PIN_8MA.
- **dio_str_output_type_and_speed_and_state_t**: A structure representing the output type, speed, and state of a pin. It contains the following members:
  - **enu_output_type**: An instance of the **dio_enu_output_type_t** enum, representing the output type of the pin.
  - **enu_output_current**: An instance of the **dio_enu_output_current_t** enum, representing the output current of the pin.
  - **enu_pin_state**: An instance of the **dio_enu_pin_state_t** enum, representing the initial state of the pin.
- **dio_enu_input_type_t**: An enumerated type representing the input type of a pin. It includes values DIO_PIN_INPUT_NO_PULL_UP_NO_PULL_DOWN, DIO_PIN_INPUT_PULL_UP, and DIO_PIN_INPUT_PULL_DOWN.
- **dio_un_input_output_type_t**: A union type representing either the input or output type of a pin. It contains the following members:
  - **str_output_type_and_speed_and_state**: An instance of the **dio_str_output_type_and_speed_and_state_t** structure representing the output type, speed, and state of the pin.

  - **enu_input_type**: An instance of the **dio_enu_input_type_t** enum representing the input type of the pin.

- **dio_str_pin_Config_t**: A structure representing the configuration of a pin. It contains the following members:

  - **enu_port**: An instance of the **dio_enu_portx_t** enum, representing the port of the pin.

  - **enu_pin**: An instance of the **dio_enu_pinx_t** enum, representing the pin number.

  - **enu_pin_mode**: An instance of the **dio_enu_pin_mode_t** enum, representing the mode of the pin.

  - **un_input_output_type**: An instance of the **dio_un_input_output_type_t** union, representing either the input or output type of the pin.

- **dio_enu_return_state_t**: An enumerated type representing the return states of the DIO functions. It includes values DIO_NOT_OK, DIO_OK, DIO_NULL_PTR, and DIO_EXCEED_PORT.

Function Declarations

- **dio_enable_clock**: Enables the clock for a specific port. It takes a parameter of type **dio_enu_portx_t** representing the port number. Returns a value of type **dio_enu_return_state_t** indicating the success or failure of enabling the clock.

- **dio_init_pin**: Initializes a pin by configuring its mode, input/output type, and state. It takes a pointer to a **dio_str_pin_Config_t** structure as an argument. Returns a value of type **dio_enu_return_state_t** indicating the success or failure of pin initialization.

- **dio_set_pin**: Sets the state of a pin to either high or low. It takes a pointer to a **dio_str_pin_Config_t** structure and the desired pin state as arguments. Returns a value of type **dio_enu_return_state_t** indicating the success or failure of setting the pin state.

- **dio_toggle_pin**: Toggles the state of a pin. If the pin is currently high, it will be set to low, and vice versa. It takes a pointer to a **dio_str_pin_Config_t** structure as an argument. Returns a value of type **dio_enu_return_state_t** indicating the success or failure of toggling the pin state.

- **dio_read_pin**: Reads the current state of a pin and stores it in a variable. It takes a pointer to a **dio_str_pin_Config_t** structure and a pointer to a **dio_enu_pin_state_t** variable as arguments. Returns a value of type **dio_enu_return_state_t** indicating the success or failure of reading the pin state.

Low Level Design:

Flowchart

LED

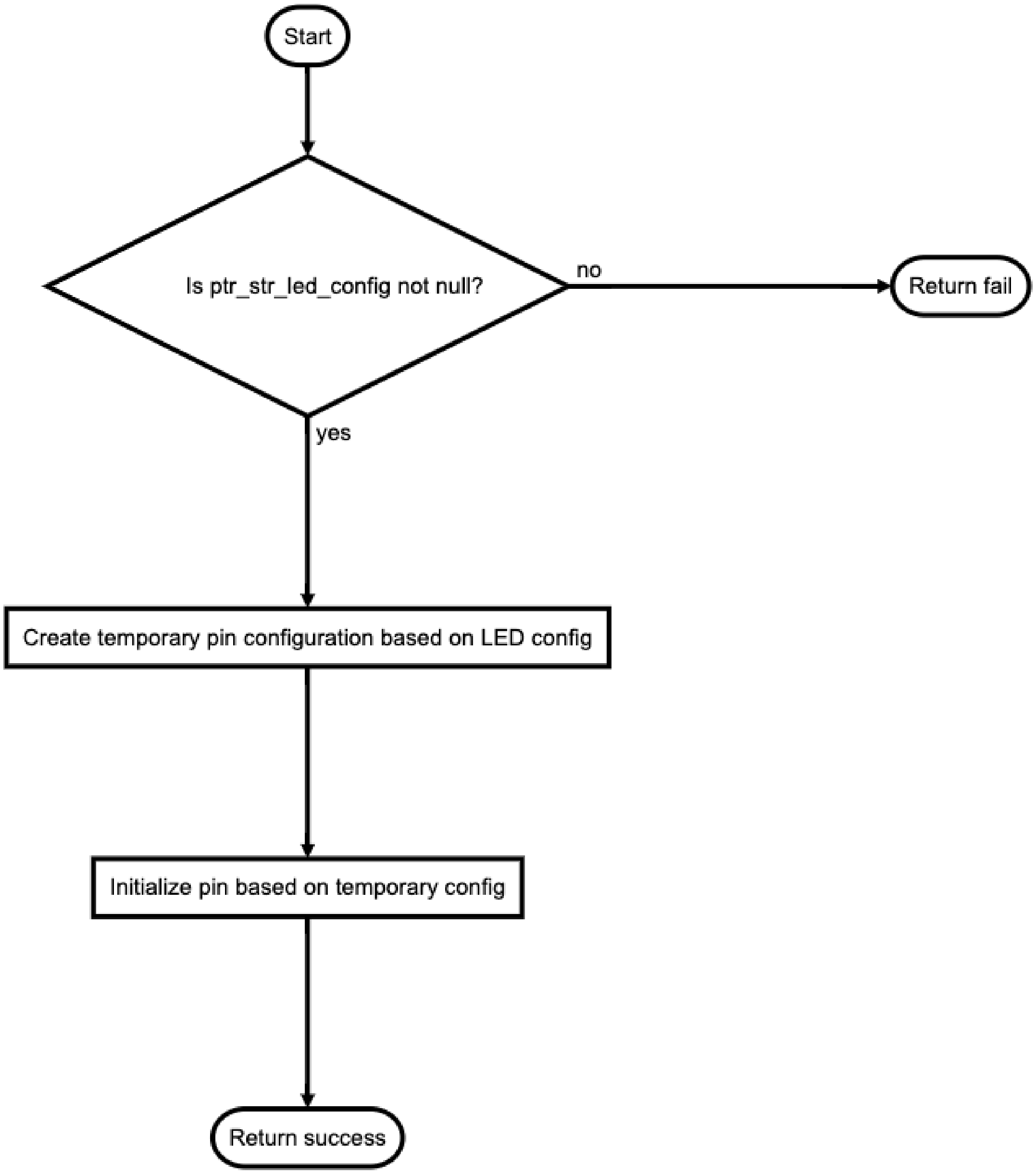

Start
Is ptr_str_led_config not null?
no
Return fail
yes
Create temporary pin configuration based on LED config
Initialize pin based on temporary config
Return success

Figure 1 led_initialization

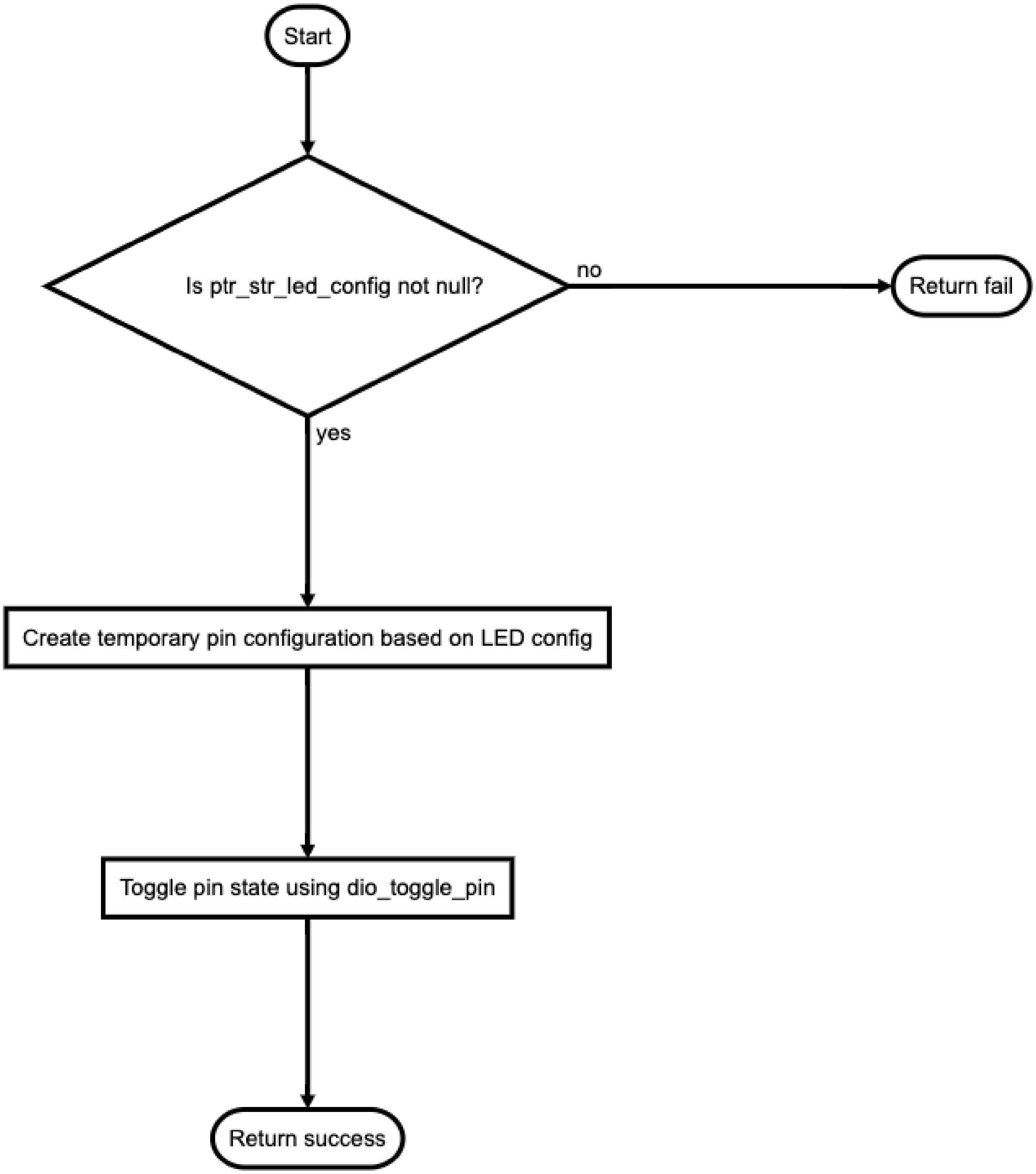

Start
Is ptr_str_led_config not null?
no
Return fail
yes
Create temporary pin configuration based on LED config
Toggle pin state using dio_toggle_pin
Return success

Figure 2 led_turn_on

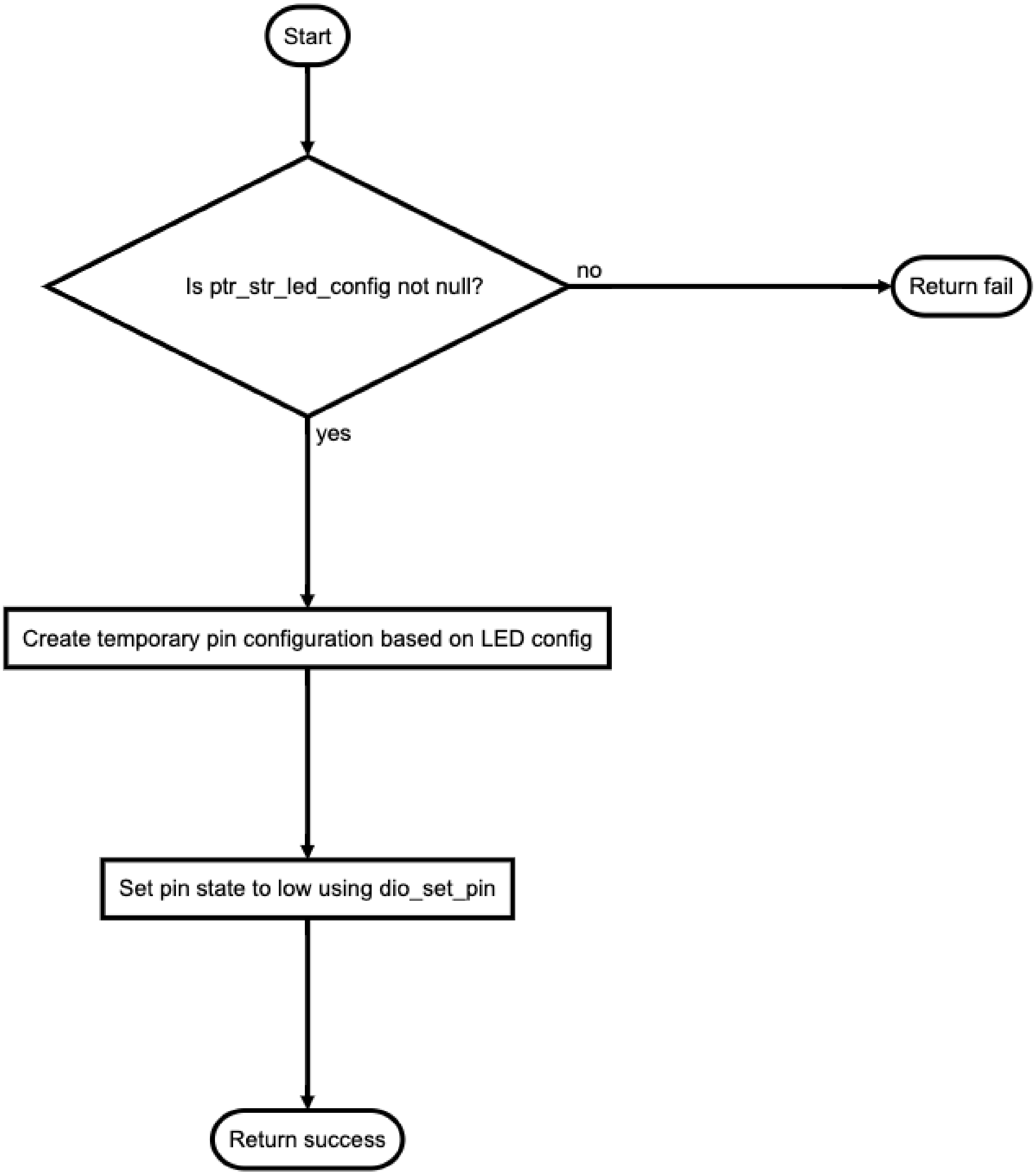

Start
Is ptr_str_led_config not null?
no
Return fail
yes
Create temporary pin configuration based on LED config
Set pin state to low using dio_set_pin
Return success

Figure 3 led_turn_off

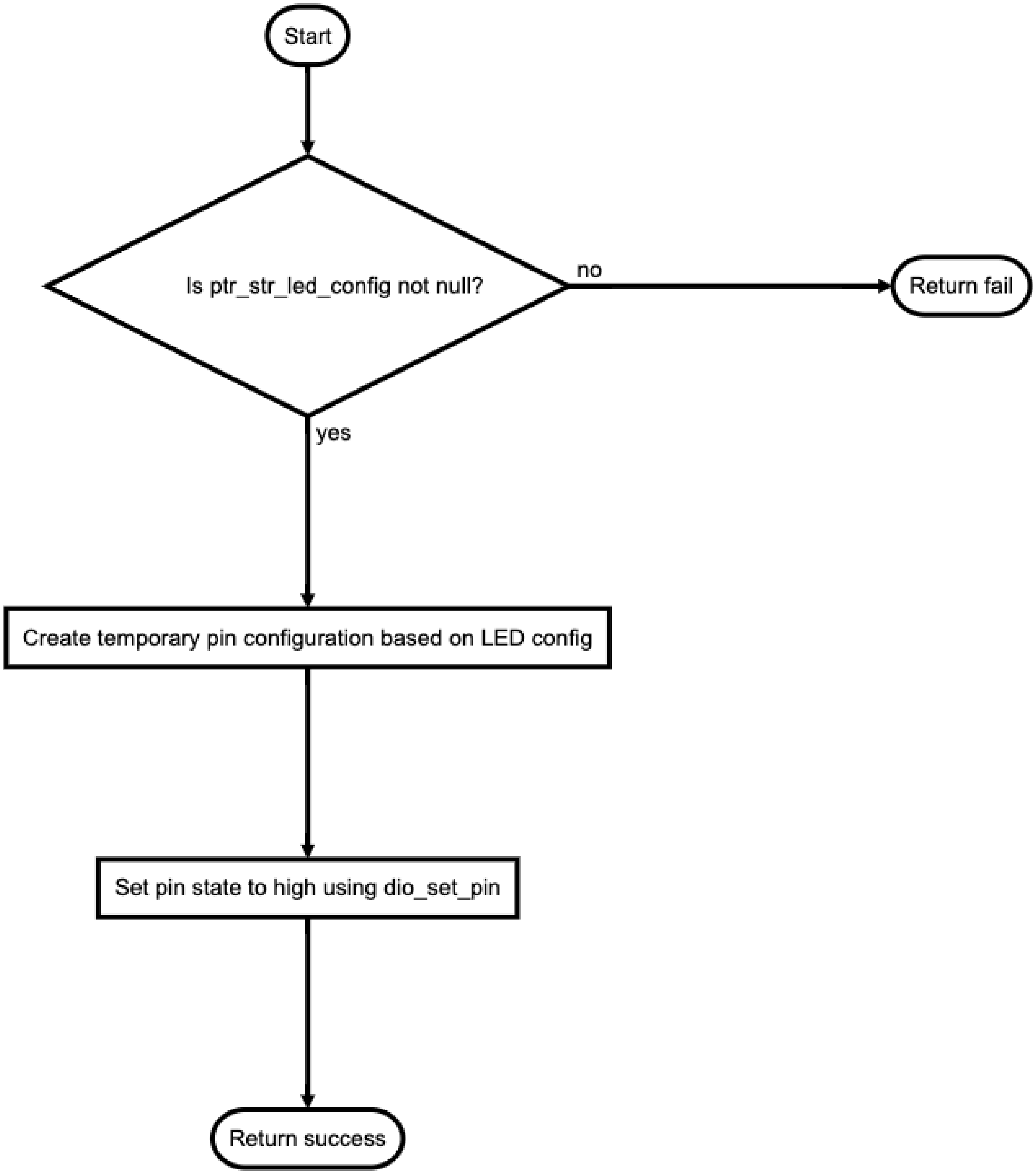

Start
Is ptr_str_led_config not null?
no
Return fail
yes
Create temporary pin configuration based on LED config
Set pin state to high using dio_set_pin
Return success

Figure 4 led_toggle

BUTTON

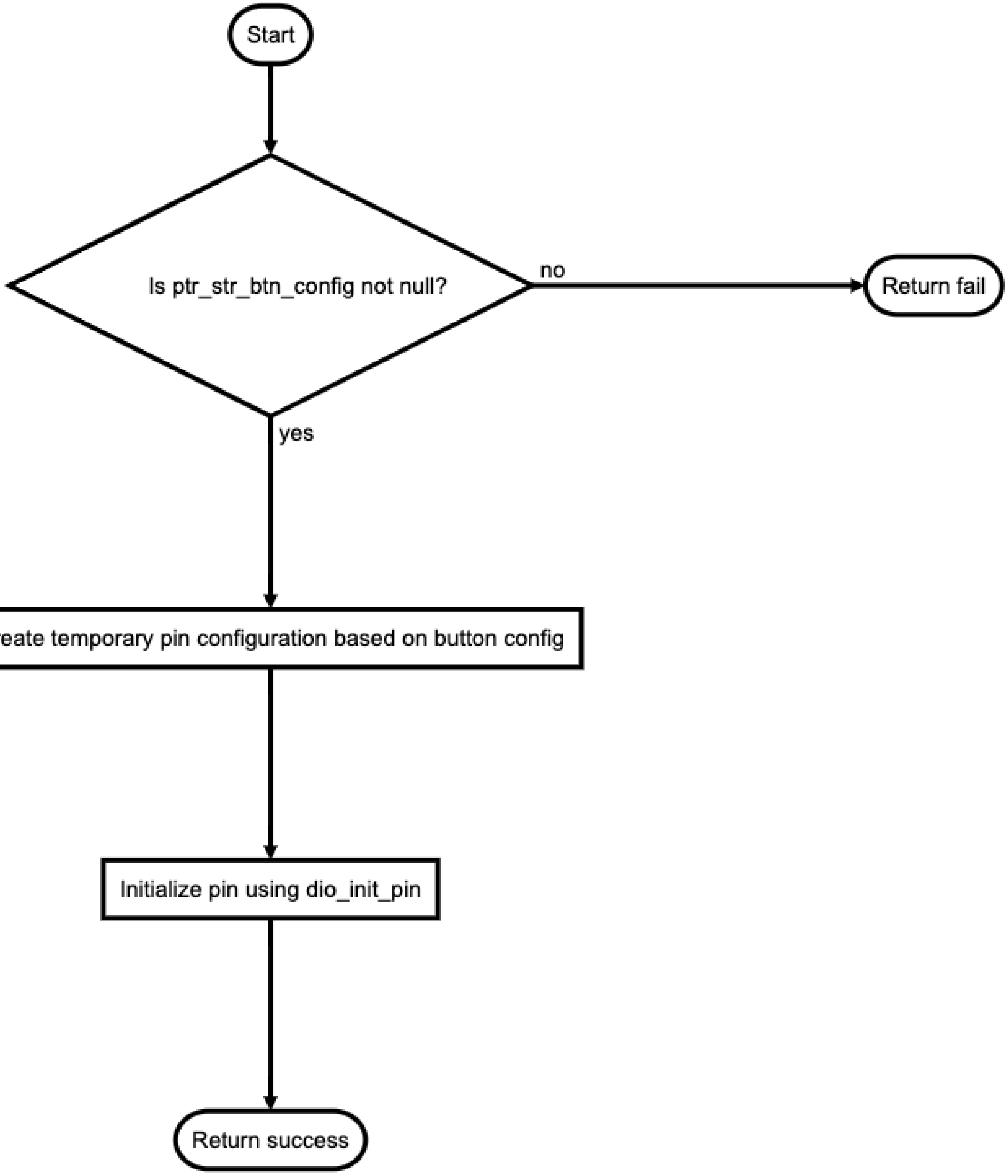

Start
Is ptr_str_btn_config not null?
no
Return fail
yes
Create temporary pin configuration based on button config
Initialize pin using dio_init_pin
Return success

Figure 5 button_initialization

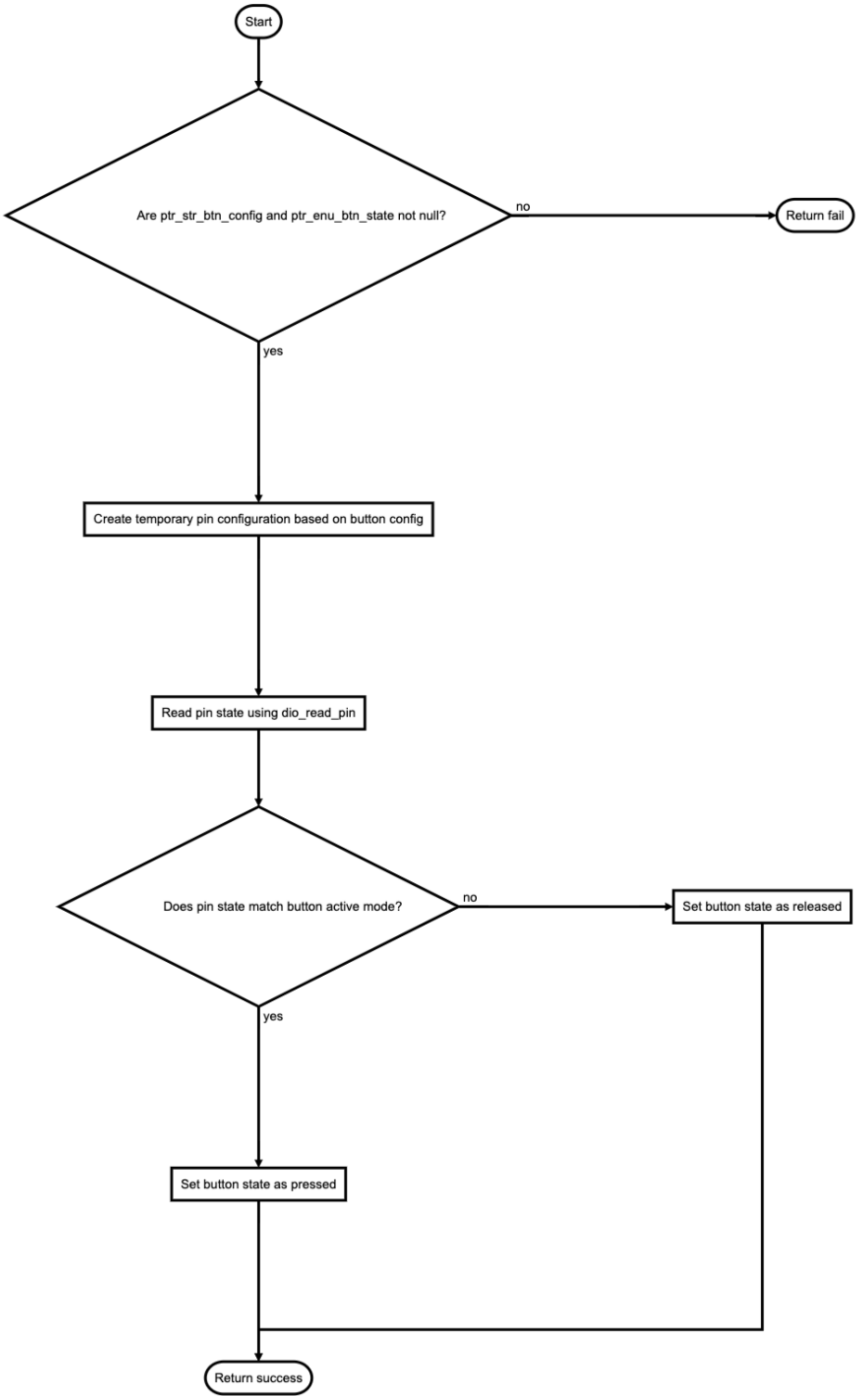

Start
Are ptr_str_btn_config and ptr_enu_btn_state not null?
no
Return fail
yes
Create temporary pin configuration based on button config
Read pin state using dio_read_pin
Does pin state match button active mode?
no
Set button state as released
yes
Set button state as pressed
Return success

Figure 6 button_read_state

GPIO

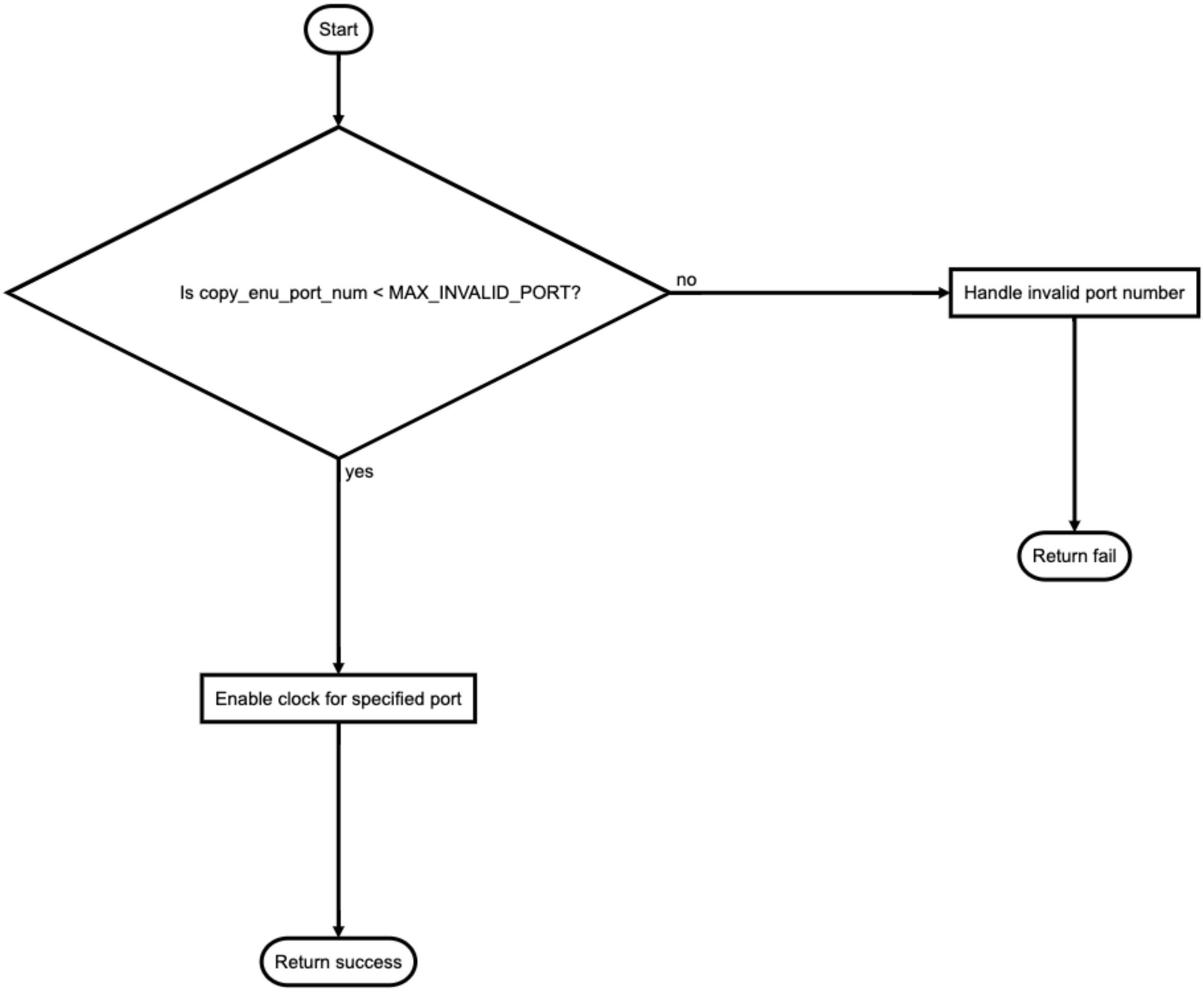

Start
Is copy_enu_port_num < MAX_INVALID_PORT?
no
Handle invalid port number
Return fail
yes
Enable clock for specified port
Return success

Figure 7 dio_enable_clock

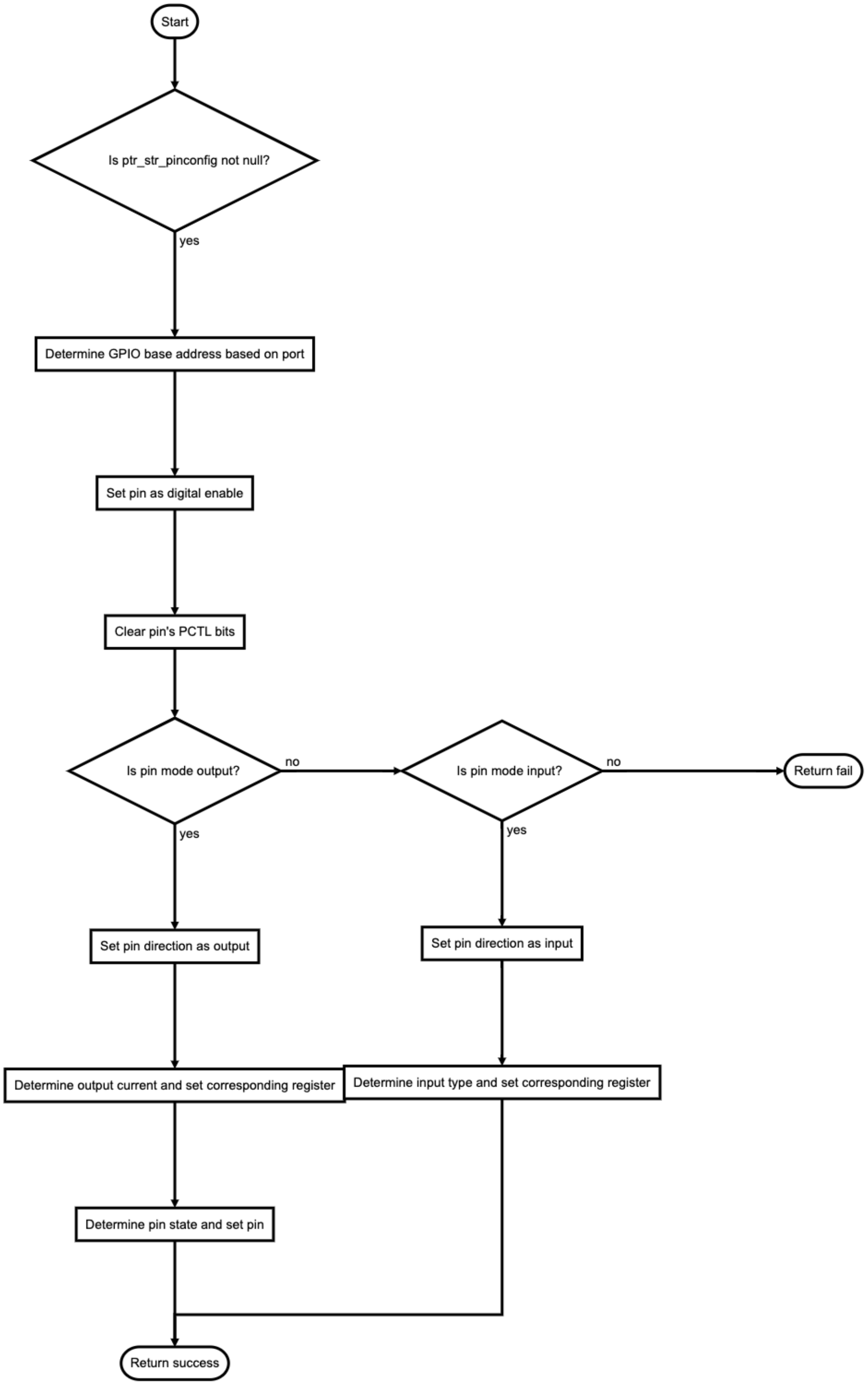

Start
Is ptr_str_pinconfig not null?
yes
Determine GPIO base address based on port
Set pin as digital enable
Clear pin's PCTL bits
Is pin mode output?
no
Is pin mode input?
no
Return fail
yes
yes
Set pin direction as output
Set pin direction as input
Determine output current and set corresponding register
Determine input type and set corresponding register
Determine pin state and set pin
Return success

Figure 8 dio_enable_clock

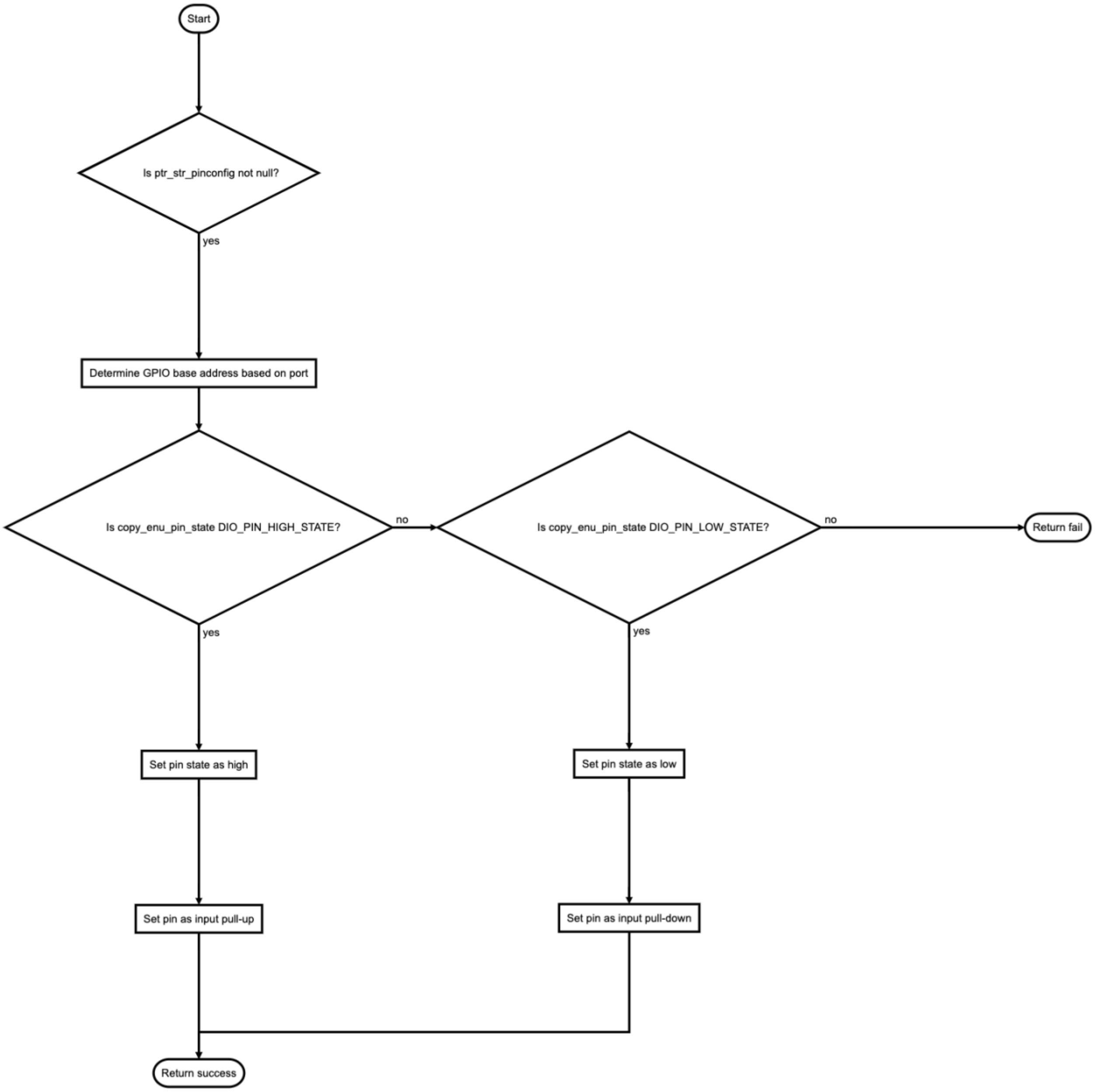

Start
Is ptr_str_pinconfig not null?
yes
Determine GPIO base address based on port
Is copy_enu_pin_state DIO_PIN_HIGH_STATE?
no
Is copy_enu_pin_state DIO_PIN_LOW_STATE?
no
Return fail
yes
yes
Set pin state as high
Set pin state as low
Set pin as input pull-up
Set pin as input pull-down
Return success

Figure 9 dio_set_pin

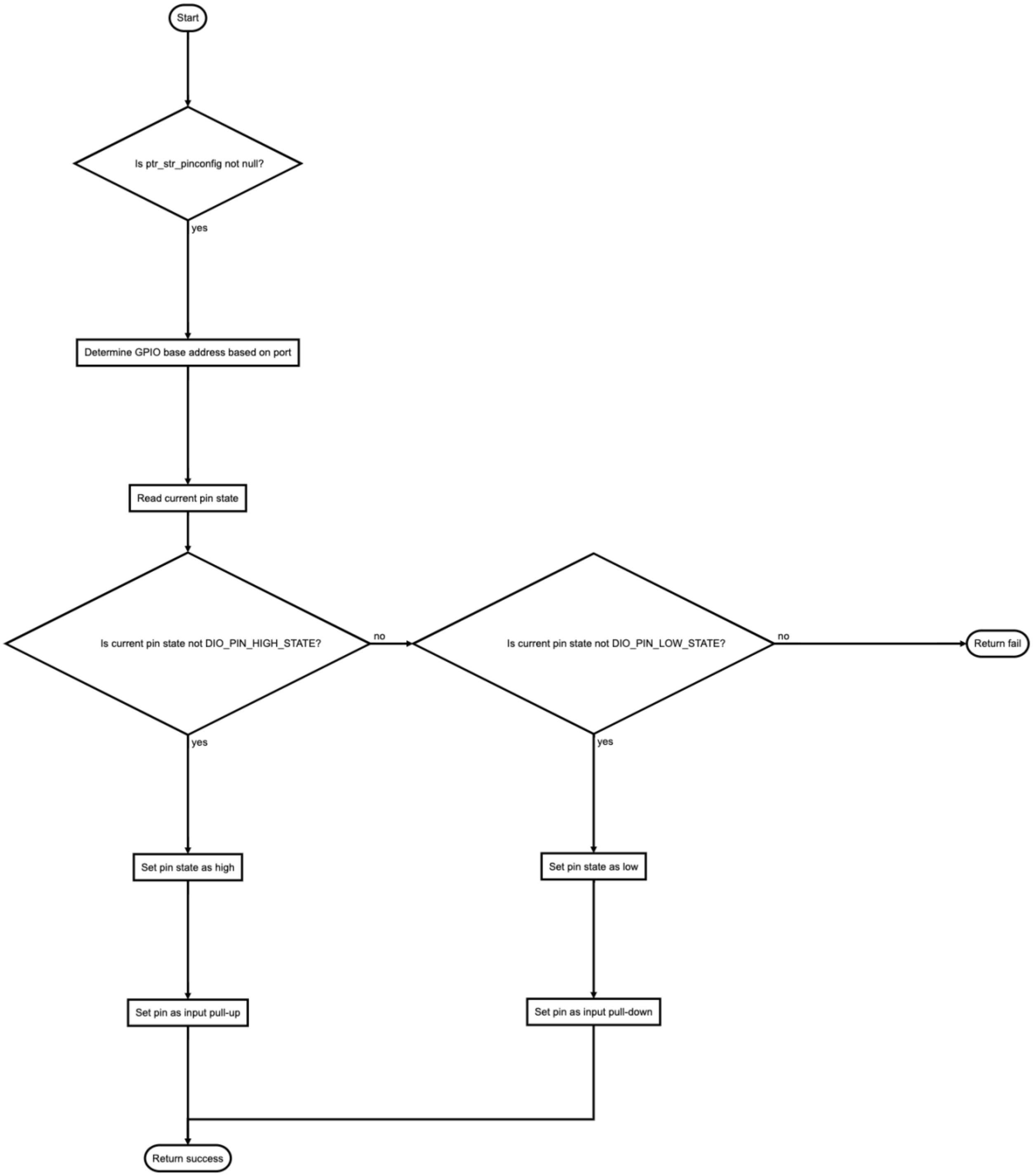

Start
Is ptr_str_pinconfig not null?
yes
Determine GPIO base address based on port
Read current pin state
Is current pin state not DIO_PIN_HIGH_STATE?
no
Is current pin state not DIO_PIN_LOW_STATE?
no
Return fail
yes
yes
Set pin state as high
Set pin state as low
Set pin as input pull-up
Set pin as input pull-down
Return success

Figure 10 dio_toggle_pin

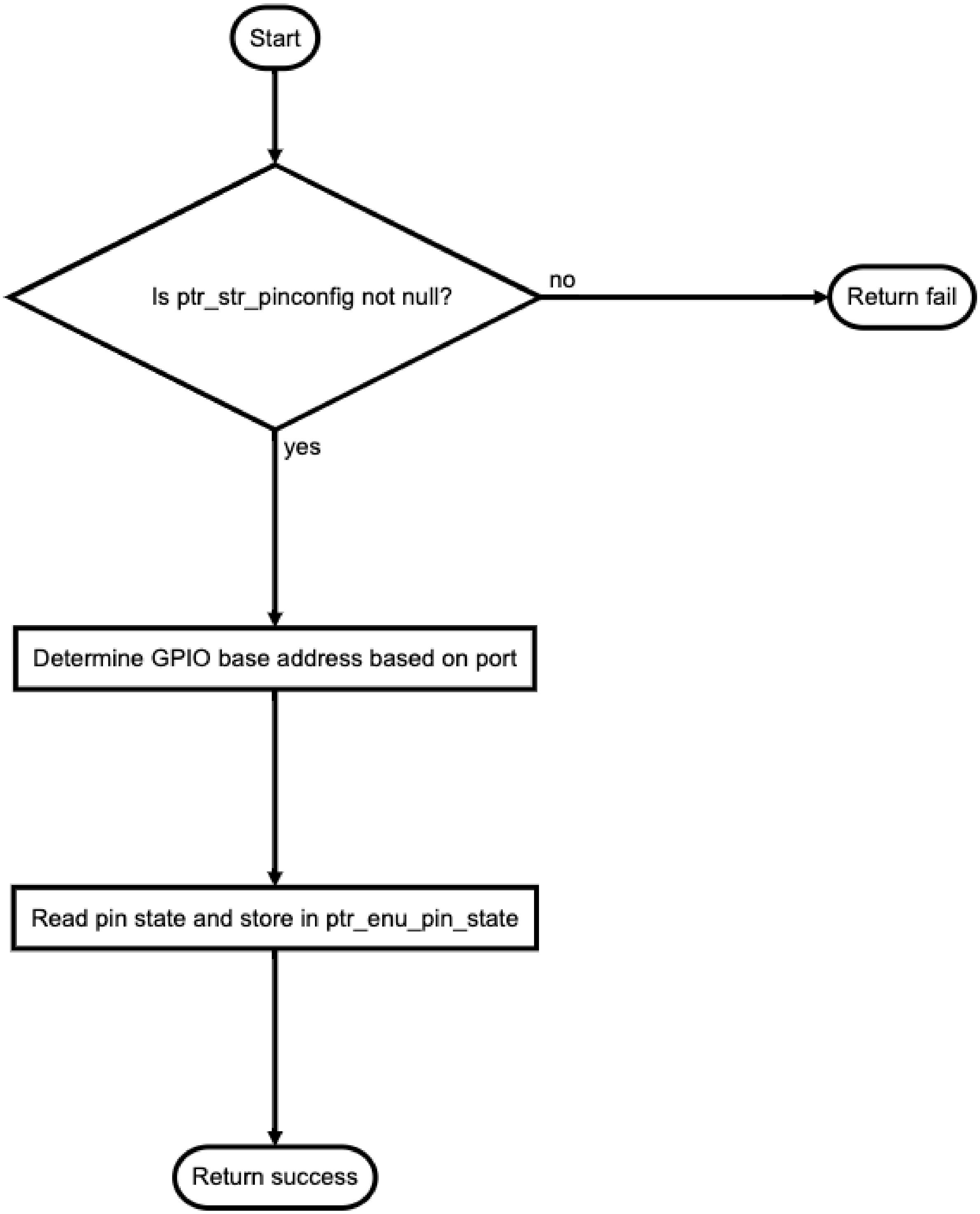
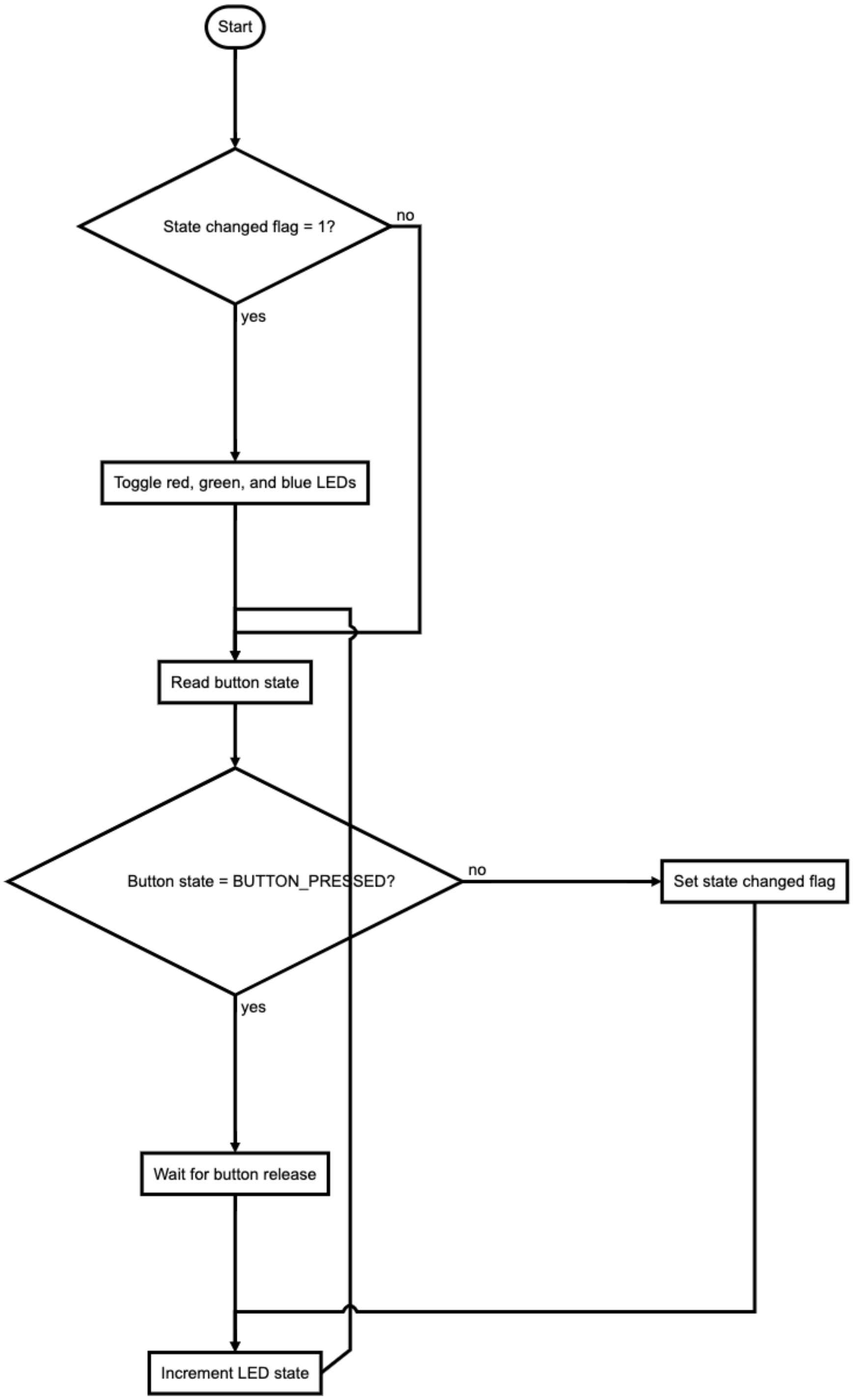

Start
Is ptr_str_pinconfig not null?
no
Return fail
yes
Determine GPIO base address based on port
Read pin state and store in ptr_enu_pin_state
Return success

Figure 11 dio_read_pin

APPLICATION


Start
State changed flag = 1?
no
yes
Toggle red, green, and blue LEDs
Read button state
Button state = BUTTON_PRESSED?
no
Set state changed flag
yes
Wait for button release
Increment LED state