

design & Implement   
A rgb led control

Youssef Abbas

Arafa Arafa



Sprints

Contents

[Project Introduction: 1](#_Toc137926563)

[Hardware Requirements: 1](#_Toc137926564)

[Software Requirements: 1](#_Toc137926565)

[High Level Design: 2](#_Toc137926566)

[Layered architecture: 2](#_Toc137926567)

[Module Description 3](#_Toc137926568)

[Driver Documentations 4](#_Toc137926569)

[Button 4](#_Toc137926570)

[LED 5](#_Toc137926571)

[GPIO 6](#_Toc137926572)

[Low Level Design: 8](#_Toc137926573)

[Flowchart 8](#_Toc137926574)

[LED 8](#_Toc137926575)

[BUTTON 12](#_Toc137926576)

[GPIO 14](#_Toc137926577)

[APPLICATION 19](#_Toc137926578)

[Figure 1 led\_initialization 8](#_Toc137926594)

[Figure 2 led\_turn\_on 9](#_Toc137926595)

[Figure 3 led\_turn\_off 10](#_Toc137926596)

[Figure 4 led\_toggle 11](#_Toc137926597)

[Figure 5 button\_initialization 12](#_Toc137926598)

[Figure 6 button\_read\_state 13](#_Toc137926599)

[Figure 7 dio\_enable\_clock 14](#_Toc137926600)

[Figure 8 dio\_enable\_clock 15](#_Toc137926601)

[Figure 9 dio\_set\_pin 16](#_Toc137926602)

[Figure 10 dio\_toggle\_pin 17](#_Toc137926603)

[Figure 11 dio\_read\_pin 18](#_Toc137926604)

# 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.

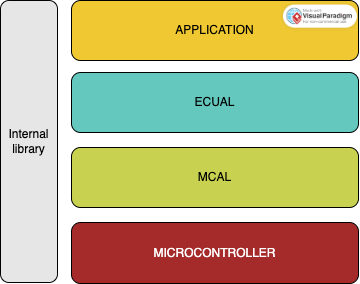
## Software Requirements:

* 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
   1. LED
   2. Button
3. MCAL
   1. DIO
   2. SYSTICK
4. Microcontroller

A picture containing text, screenshot, diagram, rectangle

Description automatically generated

## 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.

## systick

The SysTick interface provides functions for interacting with the SysTick timer. It allows configuring the timer, setting timeouts, enabling/disabling the counter, and retrieving the state of the timer. This module is designed to work with the SysTick timer available on microcontrollers.

Data Types:

* ptr\_void\_func: Function pointer type for callback functions.
* systick\_enu\_reach\_zero\_flag\_t: Enumeration representing the reach-zero flag of the SysTick timer.
  + SYSTICK\_NOT\_REACH\_FLAG: SysTick timer has not reached zero.
  + SYSTICK\_REACH\_FLAG: SysTick timer has reached zero.
* systick\_enu\_clock\_source\_t: Enumeration representing the clock source for the SysTick timer.
  + SYSTICK\_PIOSC\_CLOCK\_PRE\_4: PIOSC (Precision Internal Oscillator) clock divided by 4.
  + SYSTICK\_SYSCLK\_CLOCK: System clock.
* systick\_enu\_int\_state\_t: Enumeration representing the interrupt state for the SysTick timer.
  + SYSTICK\_DISABLE\_INT: Disable interrupts.
  + SYSTICK\_ENABLE\_INT: Enable interrupts.
* systick\_enu\_counter\_state\_t: Enumeration representing the counter state for the SysTick timer.
  + SYSTICK\_DISABLE\_COUNTER: Disable the SysTick counter.
  + SYSTICK\_ENABLE\_COUNTER: Enable the SysTick counter.
* systick\_str\_systick\_config\_t: Structure representing the configuration settings for the SysTick timer.
  + enu\_clock\_source: Clock source for the SysTick timer.
  + enu\_int\_state: Interrupt state for the SysTick timer.
  + enu\_counter\_state: Counter state for the SysTick timer.
* systick\_enu\_return\_state\_t: Enumeration representing the return status of the SysTick functions.
  + SYSTICK\_NOT\_OK: General error or failure.
  + SYSTICK\_OK: Successful operation.
  + SYSTICK\_NULL\_PTR: Null pointer error.
  + SYSTICK\_MAX\_INVALID\_LOAD\_VALUE: Invalid maximum load value error.

Function Prototypes:

* systick\_enu\_return\_state\_t systick\_init(systick\_str\_systick\_config\_t \*ptr\_str\_systick\_config): Initializes the SysTick timer.
* systick\_enu\_return\_state\_t systick\_set\_timeout\_ms(uint32\_t time\_out): Sets the timeout value in milliseconds for the SysTick timer.
* systick\_enu\_return\_state\_t systick\_set\_callback(ptr\_void\_func ptr\_callback\_func): Sets the callback function to be called when the SysTick timer interrupt occurs.
* systick\_enu\_return\_state\_t systick\_counter\_enable(systick\_str\_systick\_config\_t \*ptr\_str\_systick\_config): Enables the SysTick counter with the specified configuration.
* systick\_enu\_return\_state\_t systick\_counter\_disable(systick\_str\_systick\_config\_t \*ptr\_str\_systick\_config): Disables the SysTick counter with the specified configuration.
* systick\_enu\_return\_state\_t systick\_get\_state(systick\_enu\_reach\_zero\_flag\_t \*ptr\_enu\_reach\_zero\_flag): Retrieves the state of the SysTick timer (reach-zero flag).

Note: This interface assumes the presence of additional header files, such as "STD\_TYPES.h", for data type definitions.

Dependencies:

* "../../STD\_LIBRARIES/STD\_TYPES.h": Header file defining standard data types used by the interface.

Usage:

1. Include the "systick\_interface.h" header file in your code.
2. Configure the systick\_str\_systick\_config\_t structure with the desired settings for the SysTick timer.
3. Initialize the SysTick timer using the systick\_init function with the configuration structure.
4. Set the timeout value using the systick\_set\_timeout\_ms

# Low Level Design:

## Flowchart

## LED

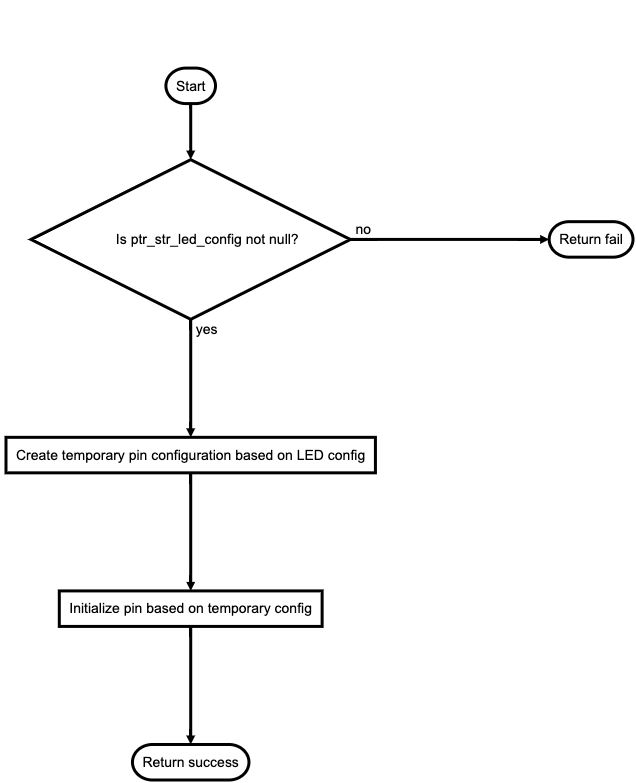


Figure led\_initialization

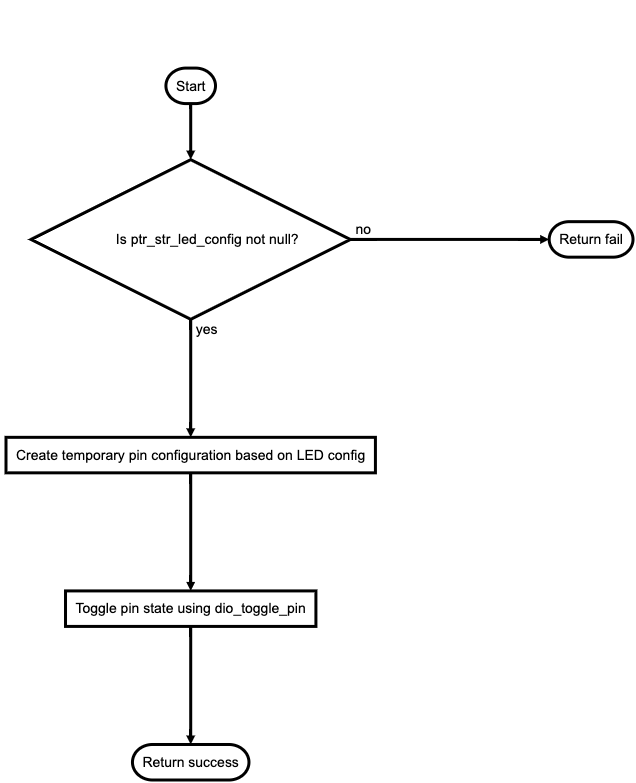


Figure led\_turn\_on

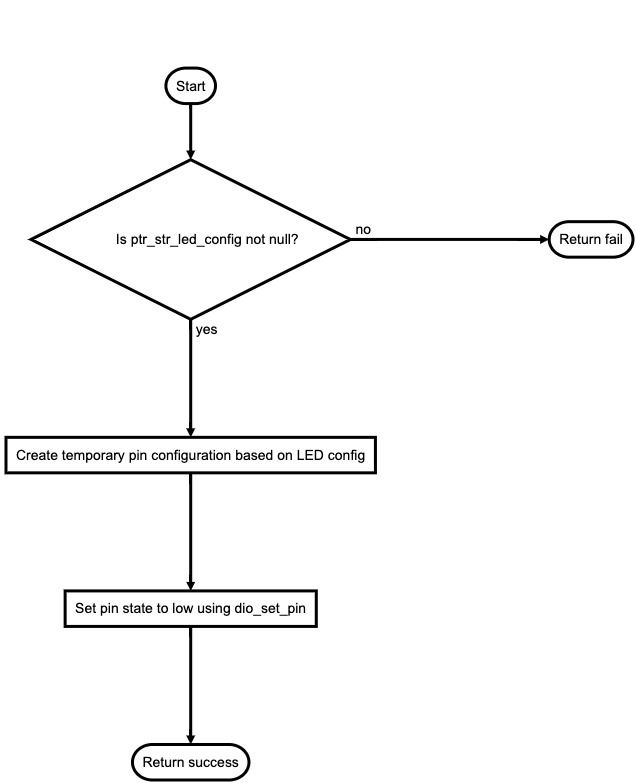


Figure led\_turn\_off

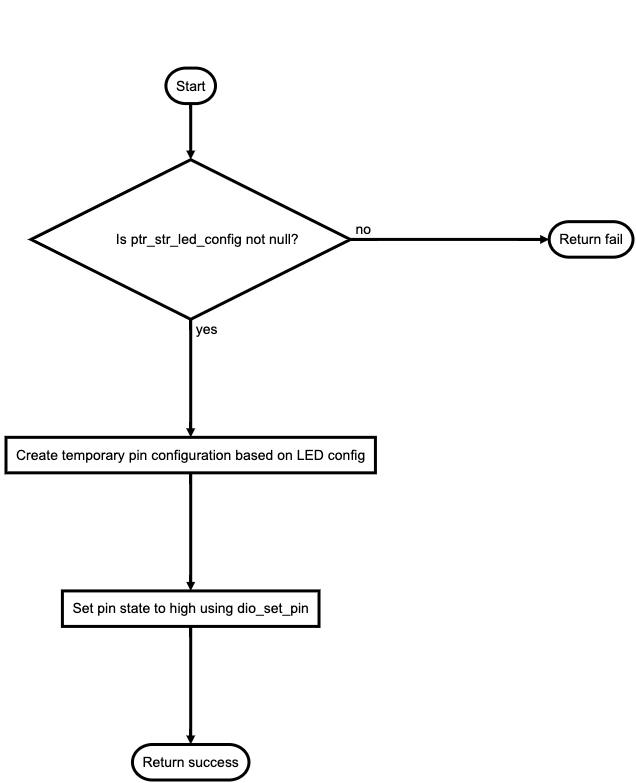


Figure led\_toggle

## BUTTON

A screenshot of a computer

Description automatically generated with medium confidence

Figure button\_initialization

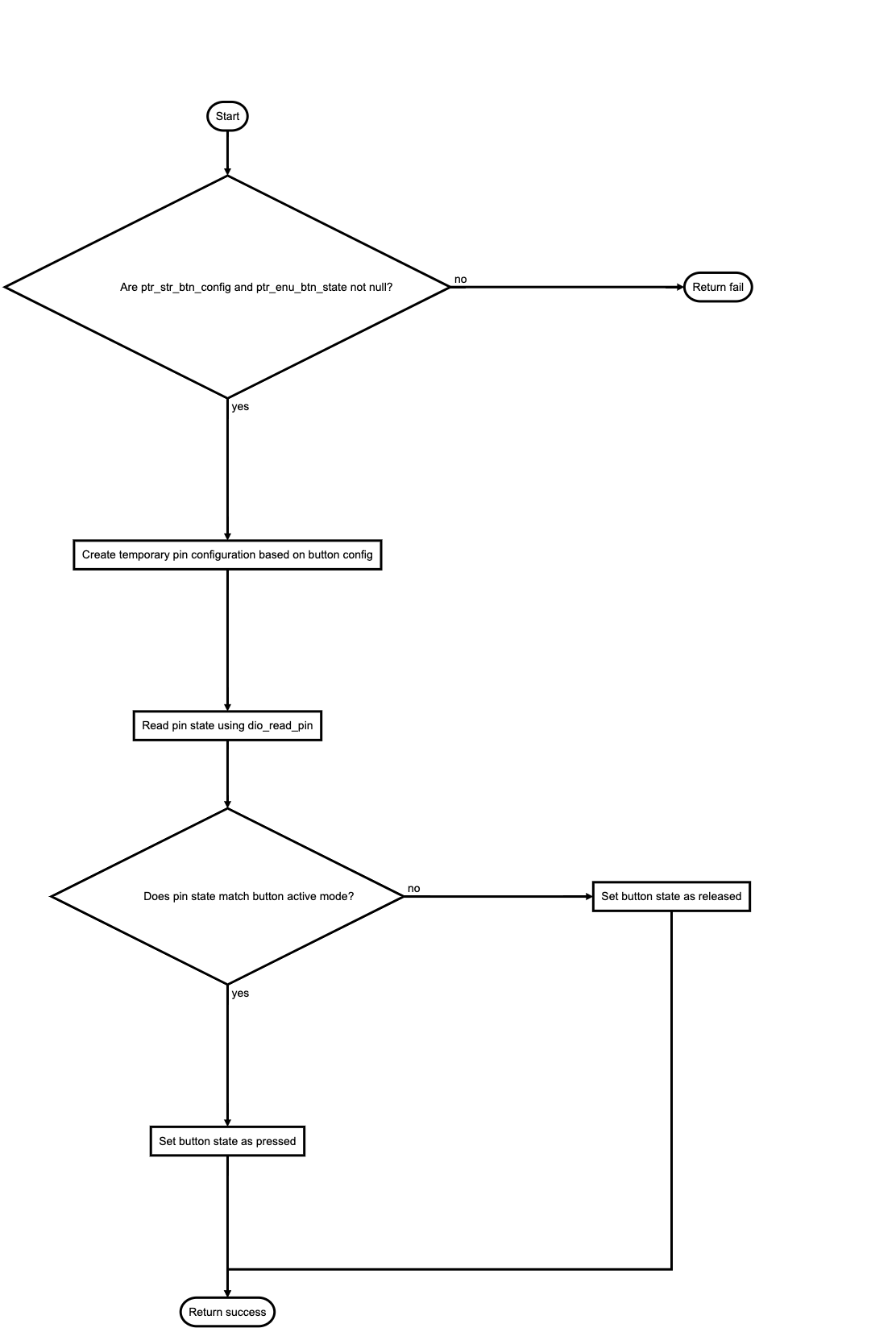


Figure button\_read\_state

## GPIO

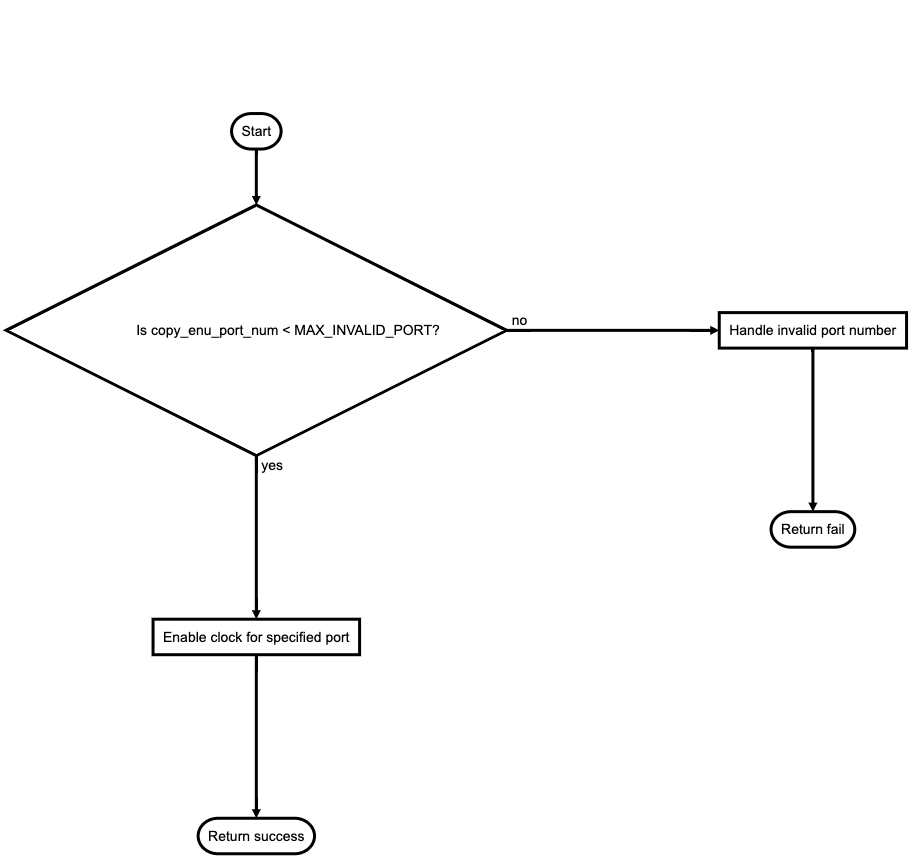


Figure dio\_enable\_clock

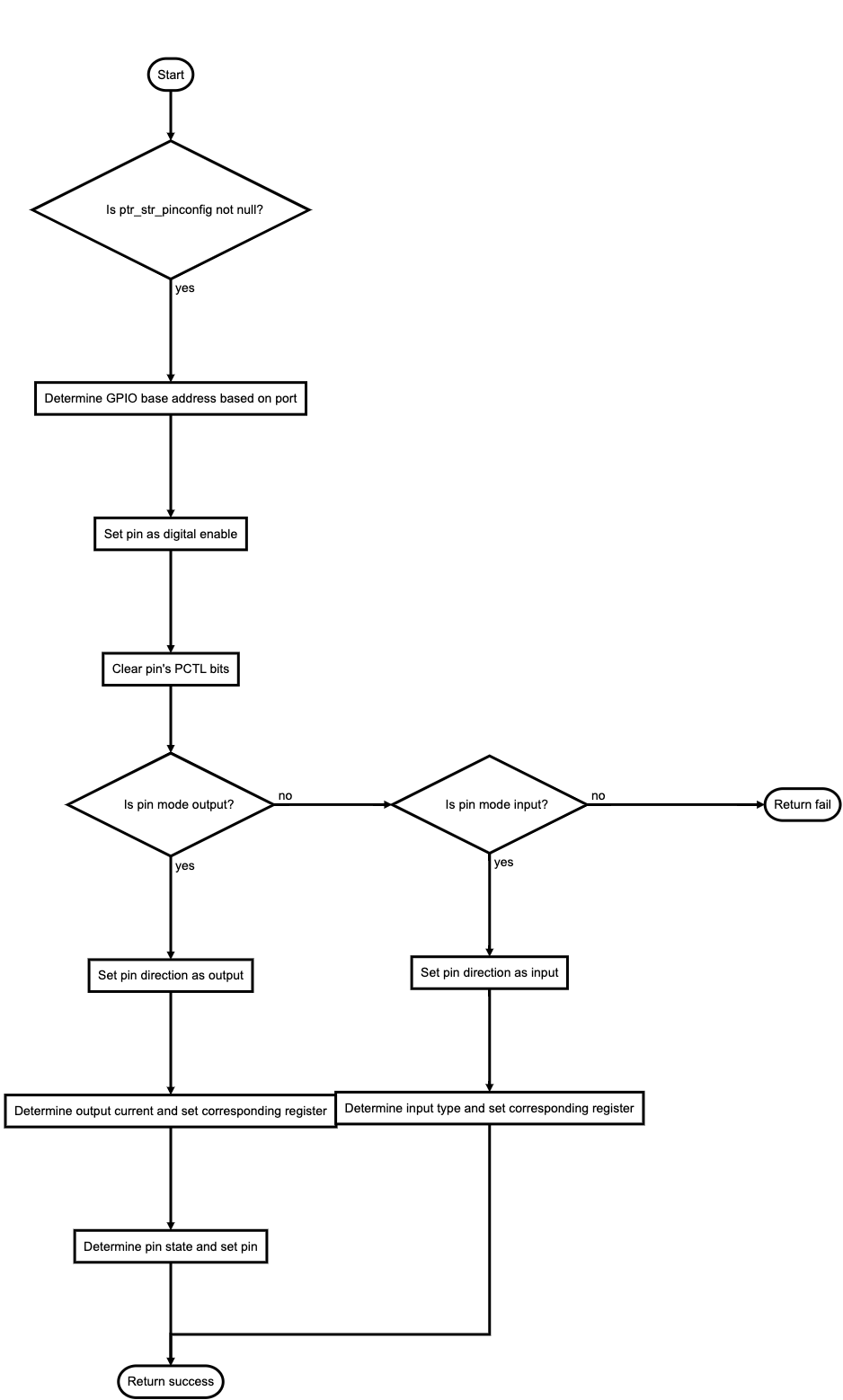


Figure dio\_enable\_clock

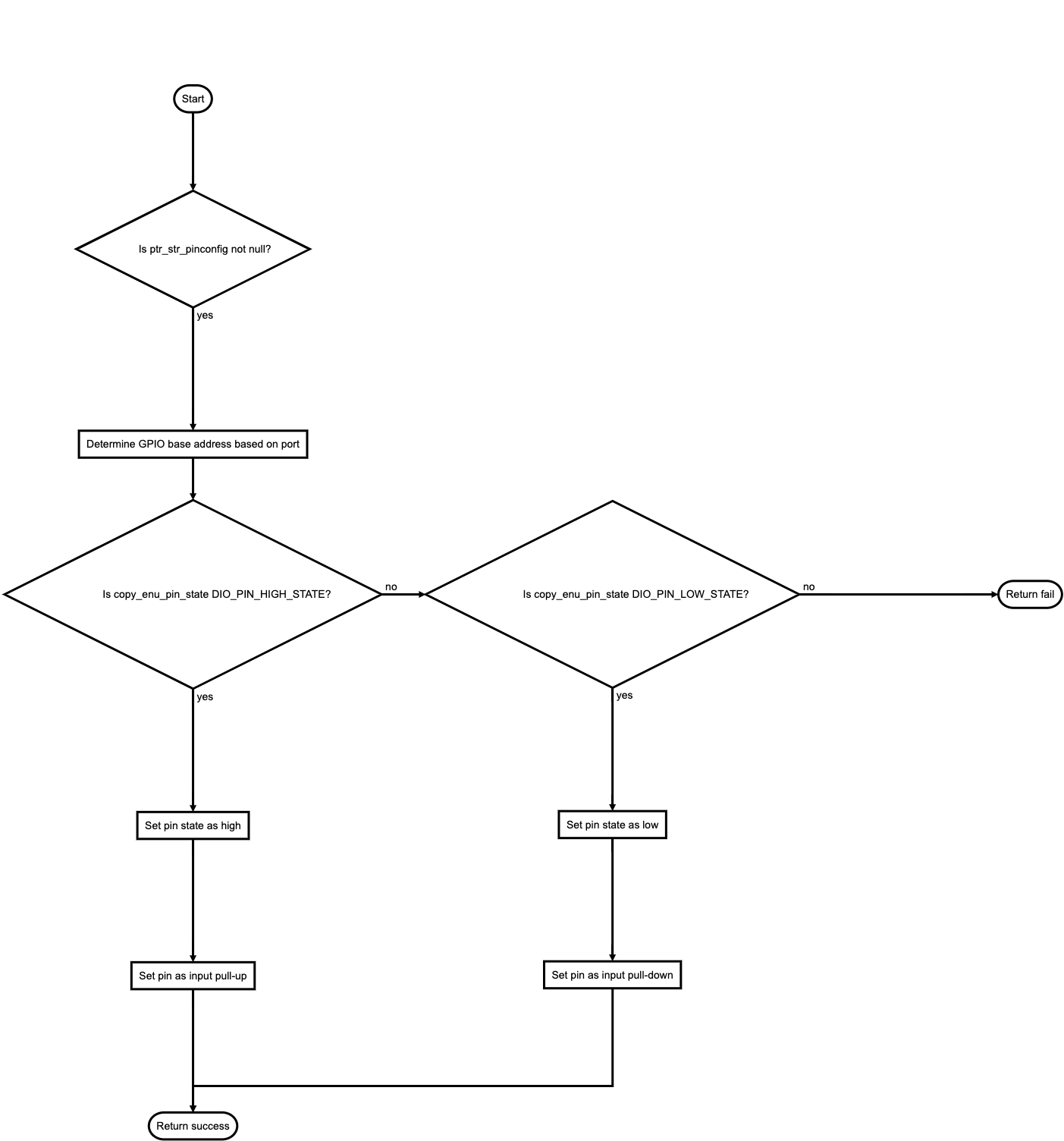


Figure dio\_set\_pin

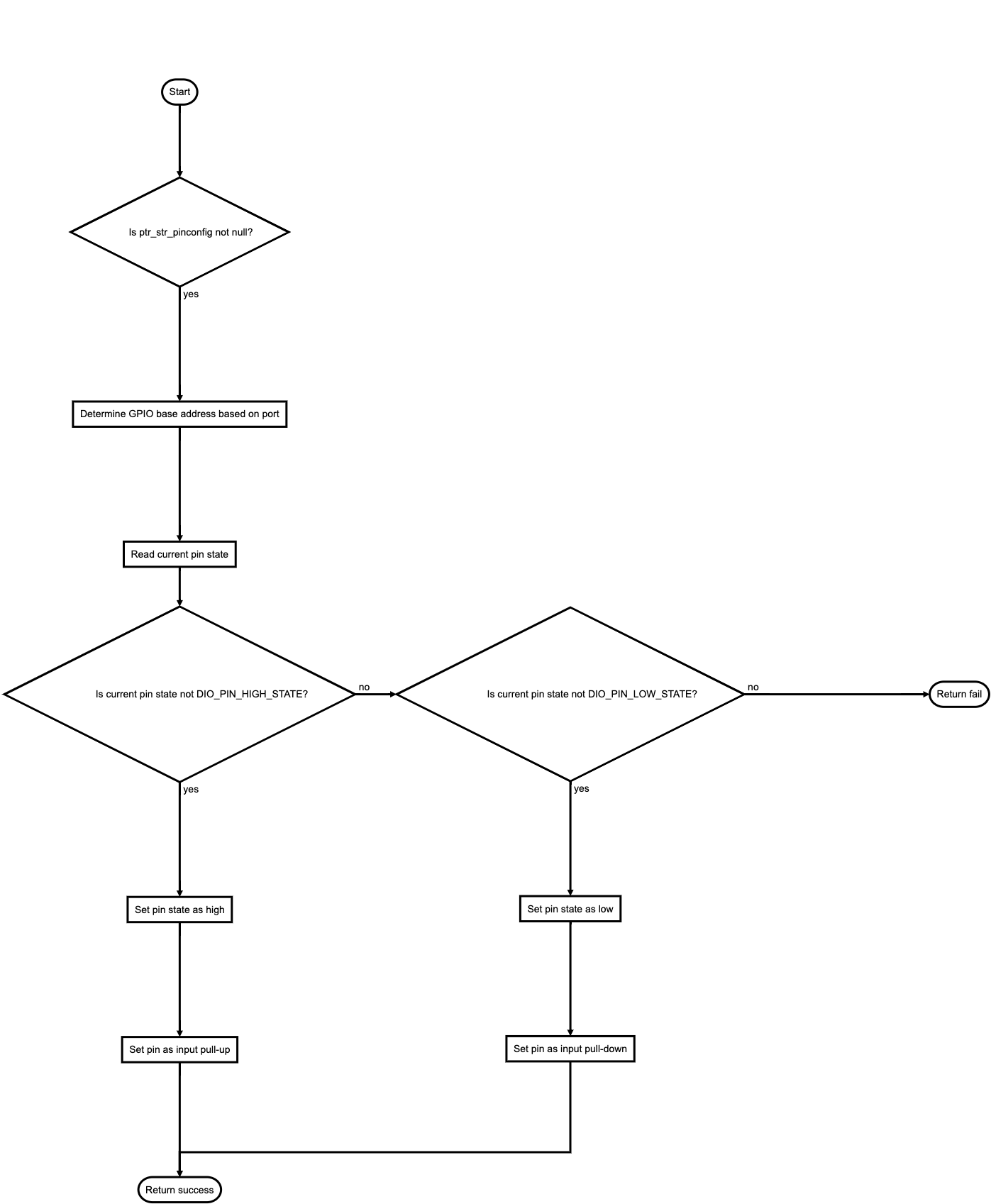


Figure dio\_toggle\_pin

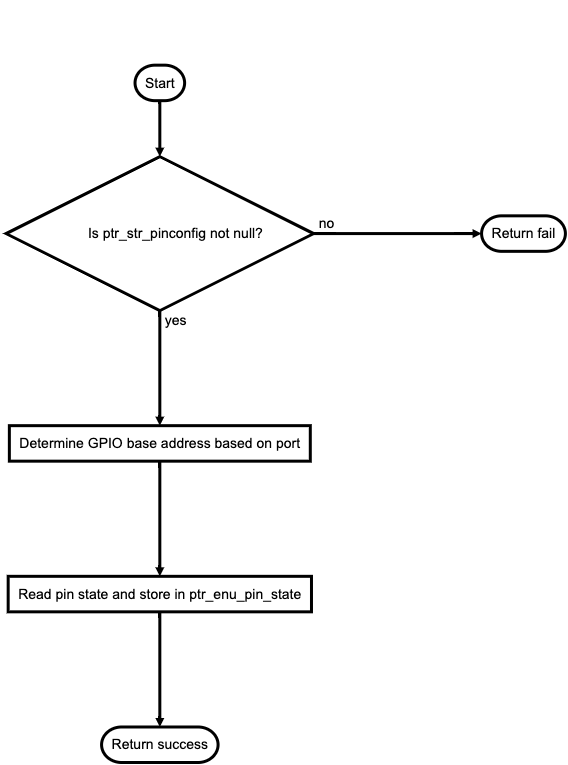


Figure dio\_read\_pin

## APPLICATION