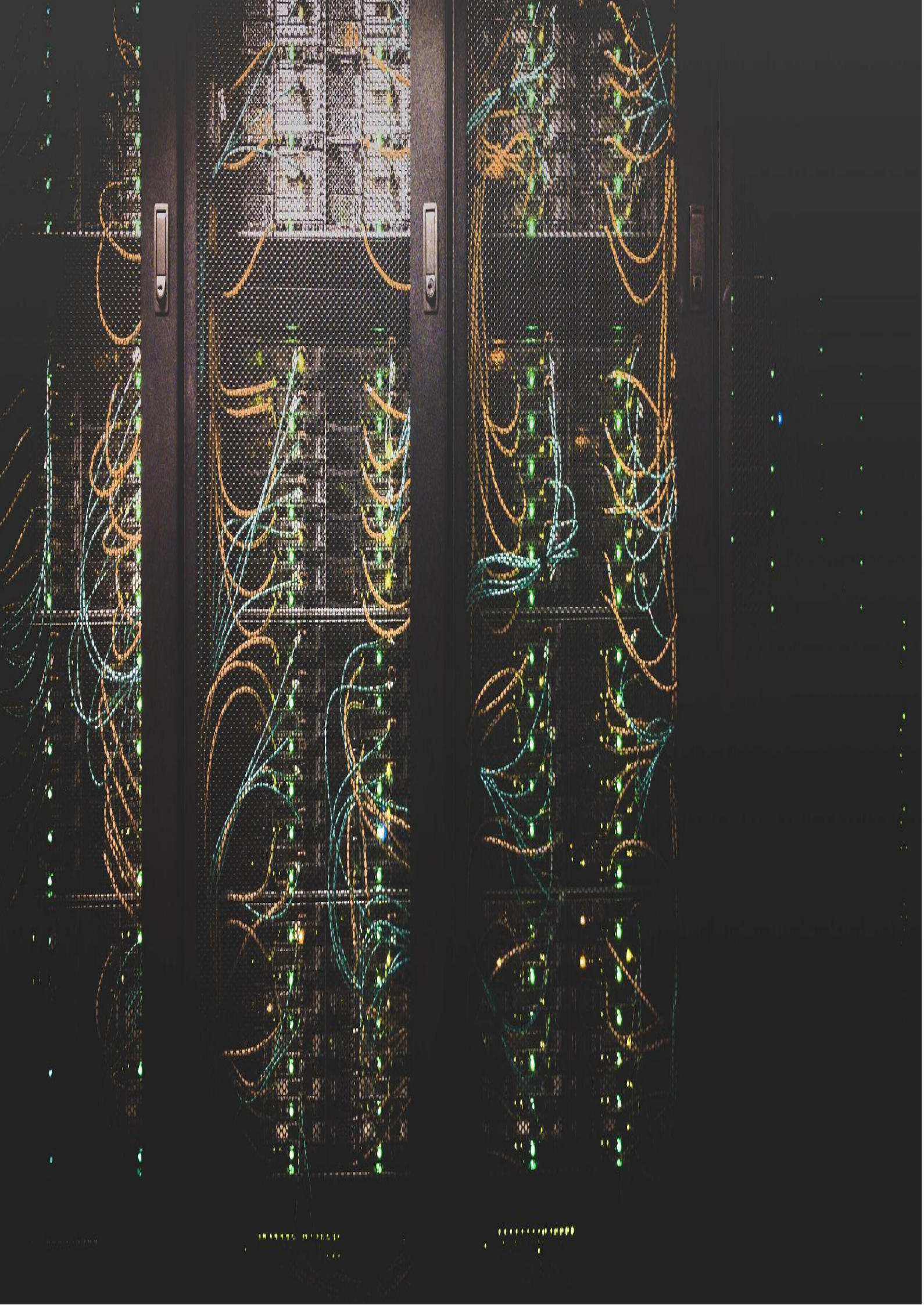


Basic Communication Manager
By Sherif Ashraf Khadr

1 - Project Introduction

1.1 - Project Description

The communication manager module is capable of working with different serial communication protocols using ISR (Interrupt Service Routine) with the highest possible throughput. This means that it can handle multiple serial communication protocols, such as UART, SPI, and I2C, and can efficiently manage data transfer between devices using the ISR method, which allows for faster and more efficient data transfer. In this project, we also prioritise the dynamic design laws. This means that we take into account the principles of dynamic design, such as minimizing delay, optimizing data transfer rate, and ensuring data integrity, while developing the communication manager module. By adhering to these laws, we can ensure that the module is efficient, reliable, and responsive, and can meet the requirements of different applications that use serial communication protocols.

1.2 - Project Components

- 2 Atmega32 MCU
- 4 Led

1.3 - Detailed Requirements

1. The BCM has the capability to send and receive any data with a maximum length of 65535 bytes (Maximum of unsigned two bytes variable)
2. It can use any communication protocol with the support of Send, Receive or both.
3. Implement bcm_Init use the below table. This function will initialize the corresponding serial communication protocol

Function Name	bcm_init
Syntax	enu_bcm_states_code_t bcm_init(str_bcm_instance_t* ptr_str_bcm_instance)
Sync / Async	Synchronous
Reentrancy	Non Reentrant

Parameters (in)	Ptr_str_bcm_instance : Address of the bcm instance
Parameters (out)	None
Parameters (in , out)	None
Return	typedef enum { BCM_NOK = 0, BCM_OK, BCM_NULL_POINTER, INVALID_SEND_DATA, INVALID_NUMBER_OF_BYTES, ALL_BYTES_SENDED }enu_bcm_states_code_t;

4. Implement bcm_deinit use the below table. This function will uninitialize the corresponding BCM instance, (instance: is the communication channel)

Function Name	bcm_deinit
Syntax	enu_bcm_states_code_t bcm_deinit(str_bcm_instance_t* ptr_str_bcm_instance)
Sync / Async	Synchronous
Reentrancy	Non Reentrant
Parameters (in)	Ptr_str_bcm_instance : Address of the bcm instance
Parameters (out)	None
Parameters (in , out)	None
Return	typedef enum { BCM_NOK = 0,

	BCM_OK, BCM_NULL_POINTER, INVALID_SEND_DATA, INVALID_NUMBER_OF_BYTES, ALL_BYTES_SENDED }enu_bcm_states_code_t;

5. Implement bcm_send that will send only 1 byte of data over a specific bcm instance

Function Name	bcm_send
Syntax	enu_bcm_states_code_t bcm_send(str_bcm_instance_t* ptr_str_bcm_instance , Uchar8_t *copy_ptr_u8_data_to_send)
Sync / Async	Asynchronous
Reentrancy	Reentrant
Parameters (in)	Ptr_str_bcm_instance : Address of the bcm instance Uchar8_t *copy_ptr_u8_data_to_send : Data Will Be Send
Parameters (out)	None
Parameters (in , out)	None
Return	typedef enum { BCM_NOK = 0, BCM_OK, BCM_NULL_POINTER, INVALID_SEND_DATA, INVALID_NUMBER_OF_BYTES, ALL_BYTES_SENDED }enu_bcm_states_code_t;

6. Implement bcm_send_n will send more than one byte with a length n over a specific BCM instance

Function Name	bcm_send_n
Syntax	enu_bcm_states_code_t bcm_send_n(str_bcm_instance_t* ptr_str_bcm_instance , Uchar8_t *copy_ptr_u8_data_to_send , Uchar8_t copy_u8_number_of_bytes)
Sync / Async	Asynchronous
Reentrancy	Reentrant
Parameters (in)	Ptr_str_bcm_instance : Address of the bcm instance
Parameters (out)	None
Parameters (in , out)	None
Return	typedef enum { BCM_NOK = 0, BCM_OK, BCM_NULL_POINTER, INVALID_SEND_DATA, INVALID_NUMBER_OF_BYTES, ALL_BYTES_SENDED }enu_bcm_states_code_t;

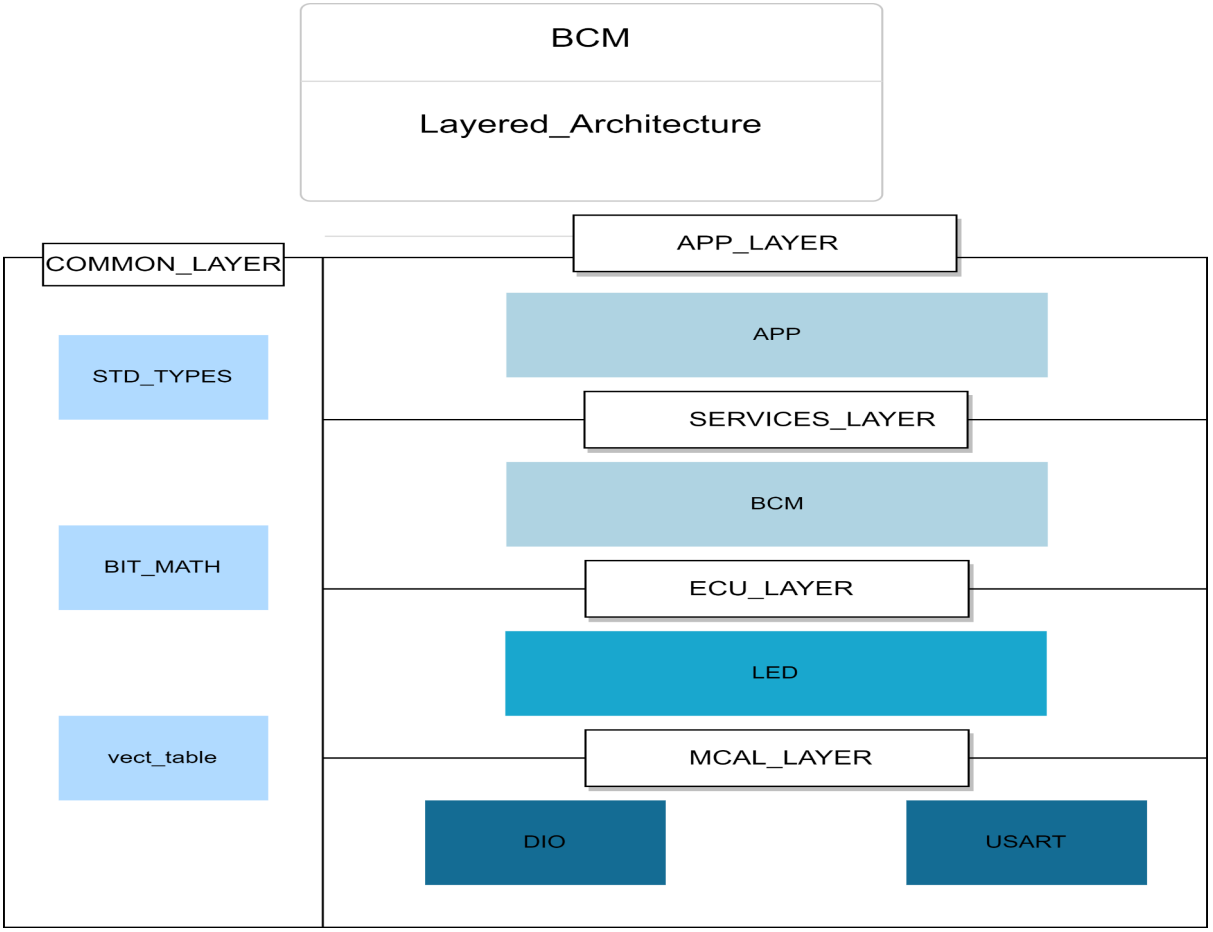
7. Implement bcm_dispatcher will execute the periodic actions and notifies the user with the needed events over a specific BCM instance

Function Name	bcm_dispatcher
Syntax	enu_bcm_states_code_t bcm_dispatchert(str_bcm_instance_t* ptr_str_bcm_instance)
Sync / Async	Synchronous
Reentrancy	Non Reentrant

Parameters (in)	Ptr_str_bcm_instance : Address of the bcm instance
Parameters (out)	None
Parameters (in , out)	None
Return	typedef enum { BCM_NOK = 0, BCM_OK, BCM_NULL_POINTER, INVALID_SEND_DATA, INVALID_NUMBER_OF_BYTES, ALL_BYTES_SENDED }enu_bcm_states_code_t;

2 - High Level Design

2.1 - Layered Architecture



2.2 - Modules Description

2.2.1 - DIO 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 - USART Module

A USART (Universal Synchronous/Asynchronous Receiver/Transmitter) module is a hardware component that provides serial communication capabilities to a microcontroller or other embedded system. It can interface with a variety of external devices, and typically includes features such as programmable baud rate, parity checking, and flow control. The module supports both synchronous and asynchronous data transfer, and is essential for systems that require reliable and flexible serial communication.

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, color 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 BCM Module

The communication manager module handles multiple serial communication protocols using ISR for fast and efficient data transfer. We prioritize dynamic design laws, such as minimizing delay, optimizing data transfer rate, and ensuring data integrity, to ensure the module is efficient, reliable, and meets different application requirements.

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

2.3.2 - MCAL Drivers

2.3.2.1 - DIO Driver

Function: GPIO_pin_direction_intialize

Description: Initializes the direction of a GPIO pin on a micro-controller based on the pin configuration specified in the input parameter.

Parameters: _pin_config: A pointer to an instance of the ST_pin_config_t struct that contains the pin number, port number, pin logic on the pin and desired direction of the GPIO pin.

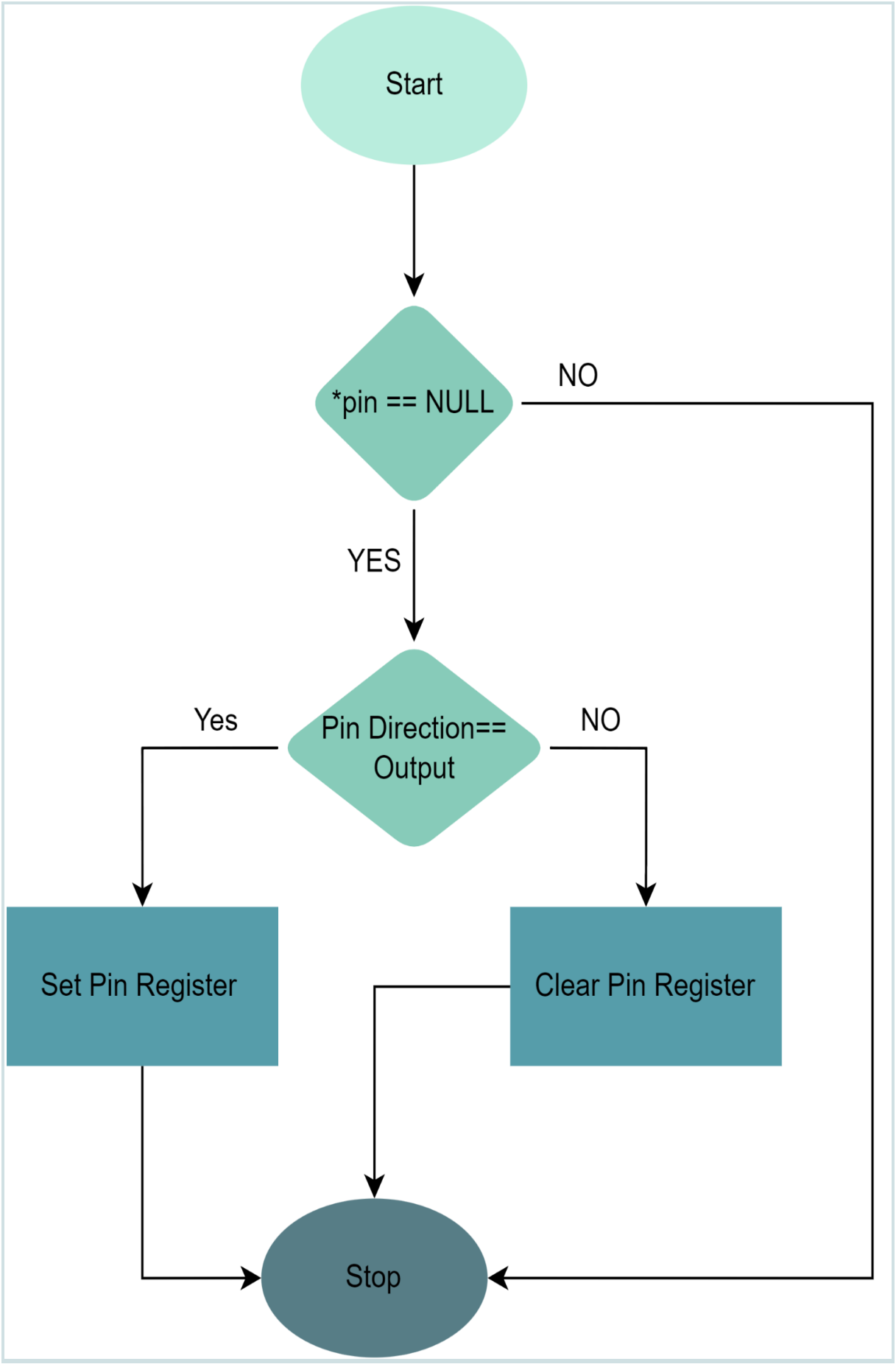
Return Type: Std_ReturnType. This is a standard type used in AUTOSAR (Automotive Open System Architecture) software development to indicate the success or failure of a function call.

The possible return values for this function are:

E_OK : The function has completed successfully.
E_NOT_OK: The function has encountered an error and could not complete successfully.

Overall, the GPIO_pin_direction_intialize function provides a way to initialize the direction of a GPIO pin based on its configuration, allowing the software to set the direction of the pin as needed for its specific functionality.
*/

Std_ReturnType GPIO_pin_direction_intialize(const ST_pin_config_t *_pin_config);


Start
*pin == NULL
NO
YES
Pin Direction== Output
Yes
NO
Set Pin Register
Clear Pin Register
Stop

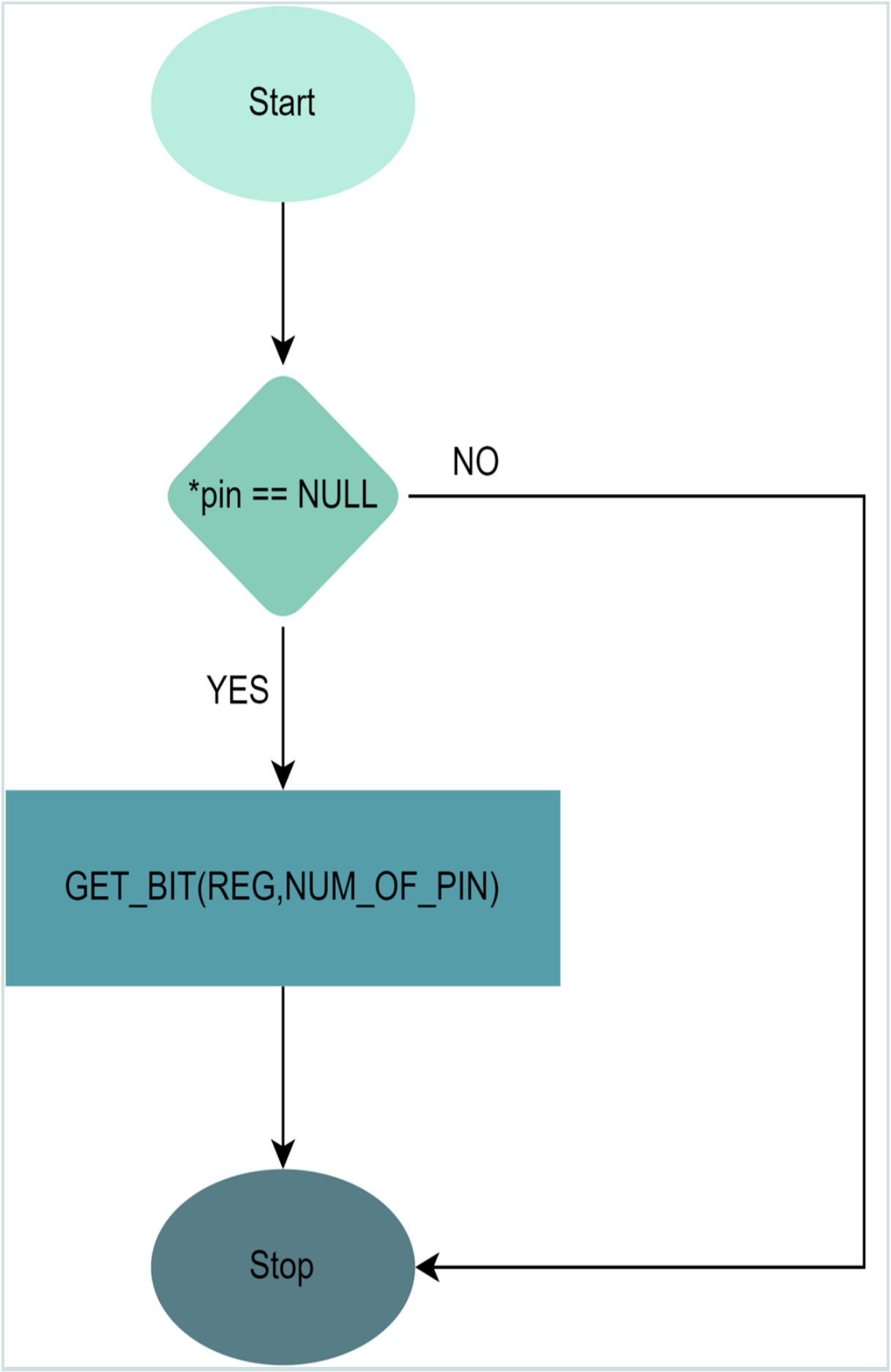
/*Function: GPIO_pin_get_direction_status
Description: Gets the current direction status of a GPIO (General Purpose Input/Output) pin on a micro-controller based on the pin configuration specified in the input parameter.

Parameters:-
_pin_config : A pointer to an instance of the ST_pin_config_t struct that contains the pin number, port number,pin logic and desired direction of the GPIO pin.
- direction_status : A pointer to a variable of type EN_direction_t where the current direction status of the GPIO pin will be stored.

Return Type : Std_ReturnType. This is a standard type used in AUTOSAR (Automotive Open System Architecture) software development to indicate the success or failure of a function call.
The possible return values for this function are:
- E_OK : The function has completed successfully.

- E_NOT_OK : The function has encountered an error and could not complete successfully.
Overall, the GPIO_pin_get_direction_status function provides a way to retrieve the current direction status of a GPIO pin based on its configuration, allowing the software to check the pin's direction as needed.*/

Std_ReturnType GPIO_pin_get_direction_status(const ST_pin_config_t *_pin_config , EN_direction_t *direction_status);


Start
*pin == NULL
NO
YES
GET_BIT(REG,NUM_OF_PIN)
Stop

/*Function: GPIO_pin_write_logic
Description : Writes the logic level of a GPIO (General Purpose Input/Output) pin on a micro-controller based on the pin configuration and desired logic level specified in the input parameters.

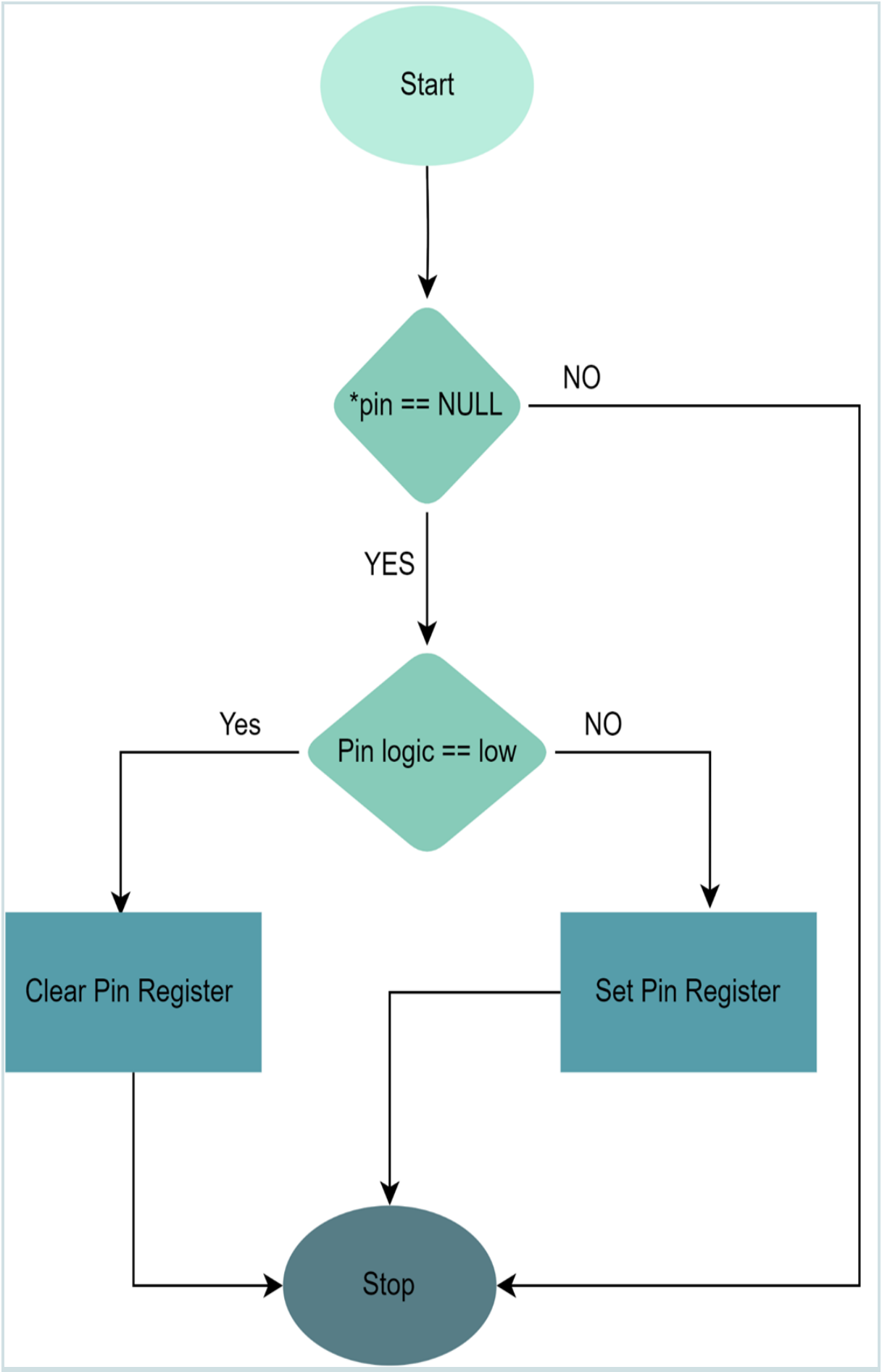
Parameters:
- _pin_config : A pointer to an instance of the ST_pin_config_t struct that contains the pin number, port number, pin logic and current direction of the GPIO pin.
- logic : The desired logic level to be written to the GPIO pin, either GPIO_LOGIC_LOW or GPIO_LOGIC_HIGH.

Return Type : Std_ReturnType. This is a standard type used in AUTOSAR (Automotive Open System Architecture) software development to indicate the success or failure of a function call.

The possible return values for this function are:
- E_OK : The function has completed successfully.
- E_NOT_OK : The function has encountered an error and could not complete successfully.

Overall, the GPIO_pin_write_logic function provides a way to write a logic level to a GPIO pin based on
its configuration, allowing the software to set the output of the pin as needed for its specific functionality.*/

Std_ReturnType GPIO_pin_write_logic(const ST_pin_config_t *_pin_config , EN_logic_t logic);


Start
*pin == NULL
NO
YES
Pin logic == low
Yes
NO
Clear Pin Register
Set Pin Register
Stop

/*Function: GPIO_pin_read_logic
Description: Reads the current logic level of a GPIO (General Purpose Input/Output) pin on a micro-controller based on the pin configuration specified in the input parameter.

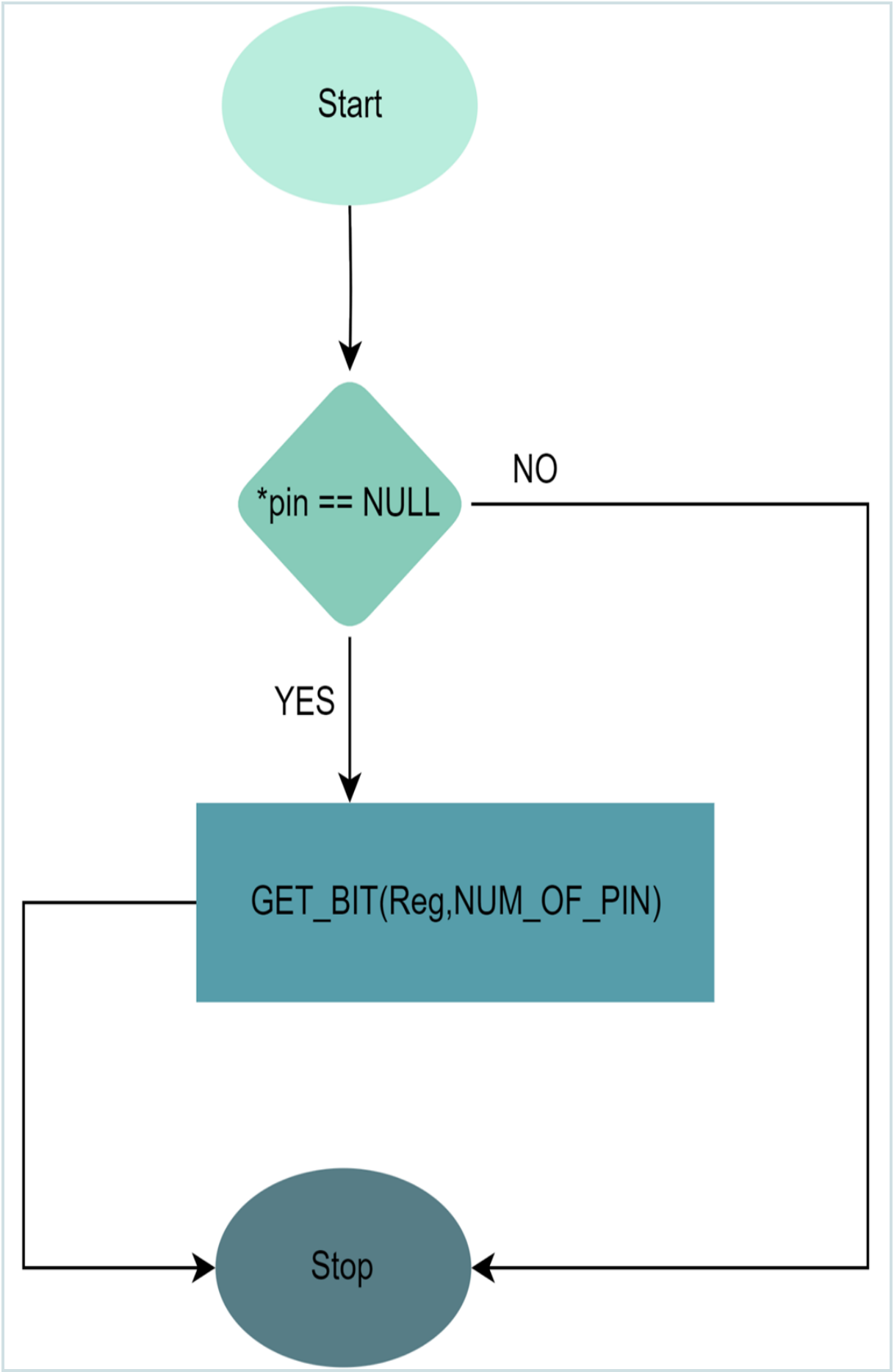
Parameters:
- _pin_config : A pointer to an instance of the ST_pin_config_t struct that contains the pin number,
port number, pin logic and current direction of the GPIO pin.
- logic_status : A pointer to a variable of type EN_logic_t where the current logic level of the GPIO pin will be stored.

Return Type: Std_ReturnType. This is a standard type used in AUTOSAR (Automotive Open System Architecture) software development to indicate the success or failure of a function call.

The possible return values for this function are:
- E_OK : The function has completed successfully.
- E_NOT_OK : The function has encountered an error and could not complete successfully.

Overall, the GPIO_pin_read_logic function provides a way to read the current logic level of a GPIO pin based on its configuration, allowing the software to check the input of the pin as needed for its specific functionality The current logic level is stored in the logic_status parameter.*/

Std_ReturnType GPIO_pin_read_logic(const ST_pin_config_t *_pin_config , EN_logic_t *logic_status);


Start
*pin == NULL
NO
YES
GET_BIT(Reg,NUM_OF_PIN)
Stop

/*Function: GPIO_pin_toggle_logic
Description: Toggles the logic level of a GPIO (General Purpose Input/Output) pin on a micro-controller based on the pin configuration specified in the input parameter.

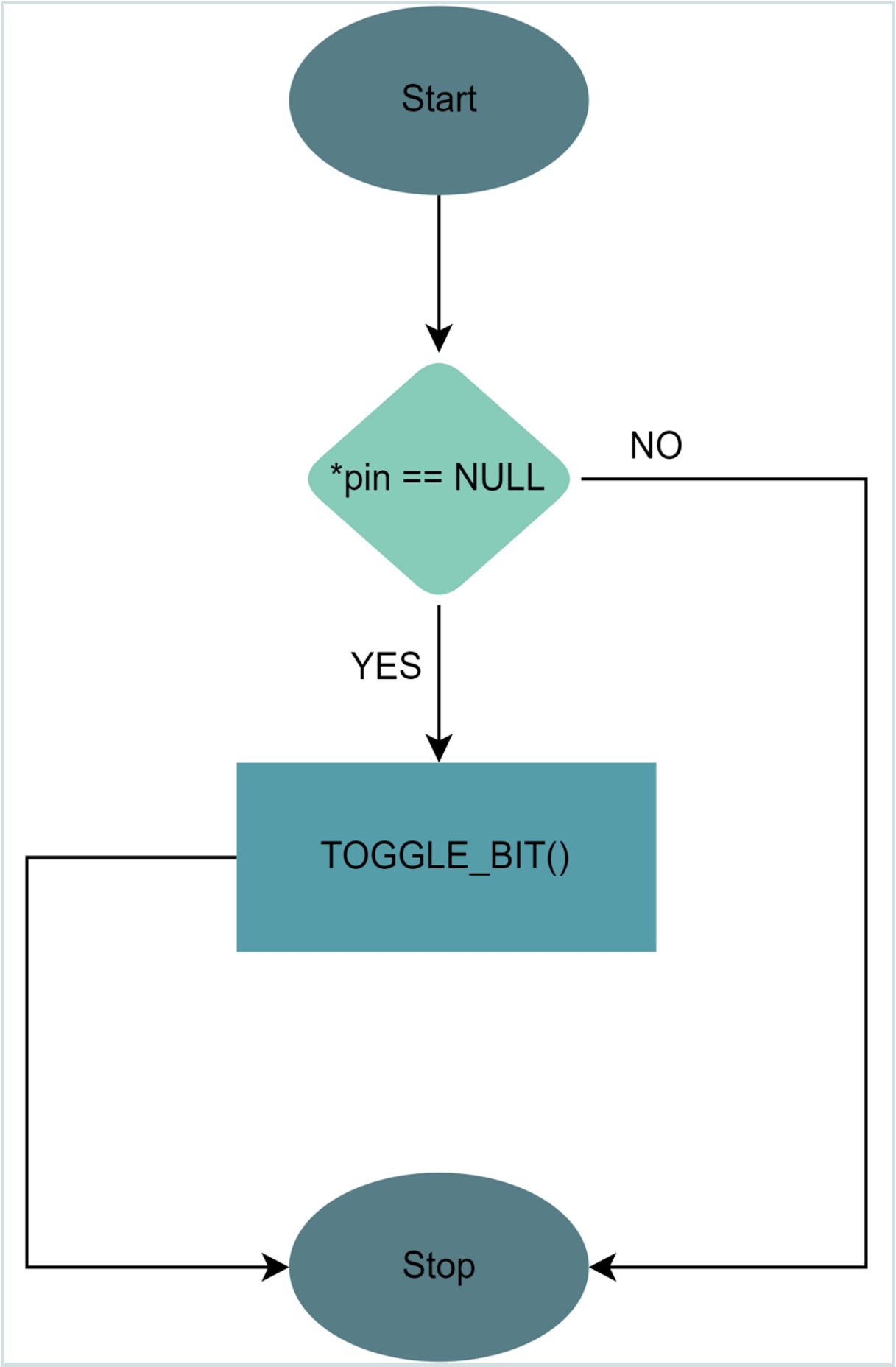
Parameters:
- _pin_config : A pointer to an instance of the ST_pin_config_t struct that contains the pin number,port number, pin logic and current direction of the GPIO pin.

Return Type : Std_ReturnType. This is a standard type used in AUTOSAR (Automotive Open System Architecture) software development to indicate the success or failure of a function call.

The possible return values for this function are:
- E_OK : The function has completed successfully.
- E_NOT_OK : The function has encountered an error and could not complete successfully.

Overall, the GPIO_pin_toggle_logic function provides a way to toggle the logic level of a GPIO pin based on its configuration, allowing the software to change the output of the pin between high and low states as needed for its specific functionality.*/

Std_ReturnType GPIO_pin_toggle_logic(const ST_pin_config_t *_pin_config);


Start
*pin == NULL
NO
YES
TOGGLE_BIT()
Stop

/* Function: GPIO_pin_intialize
Description : Initializes a GPIO (General Purpose Input/Output) pin on a micro-controller based on the pin configuration specified in the input parameter.

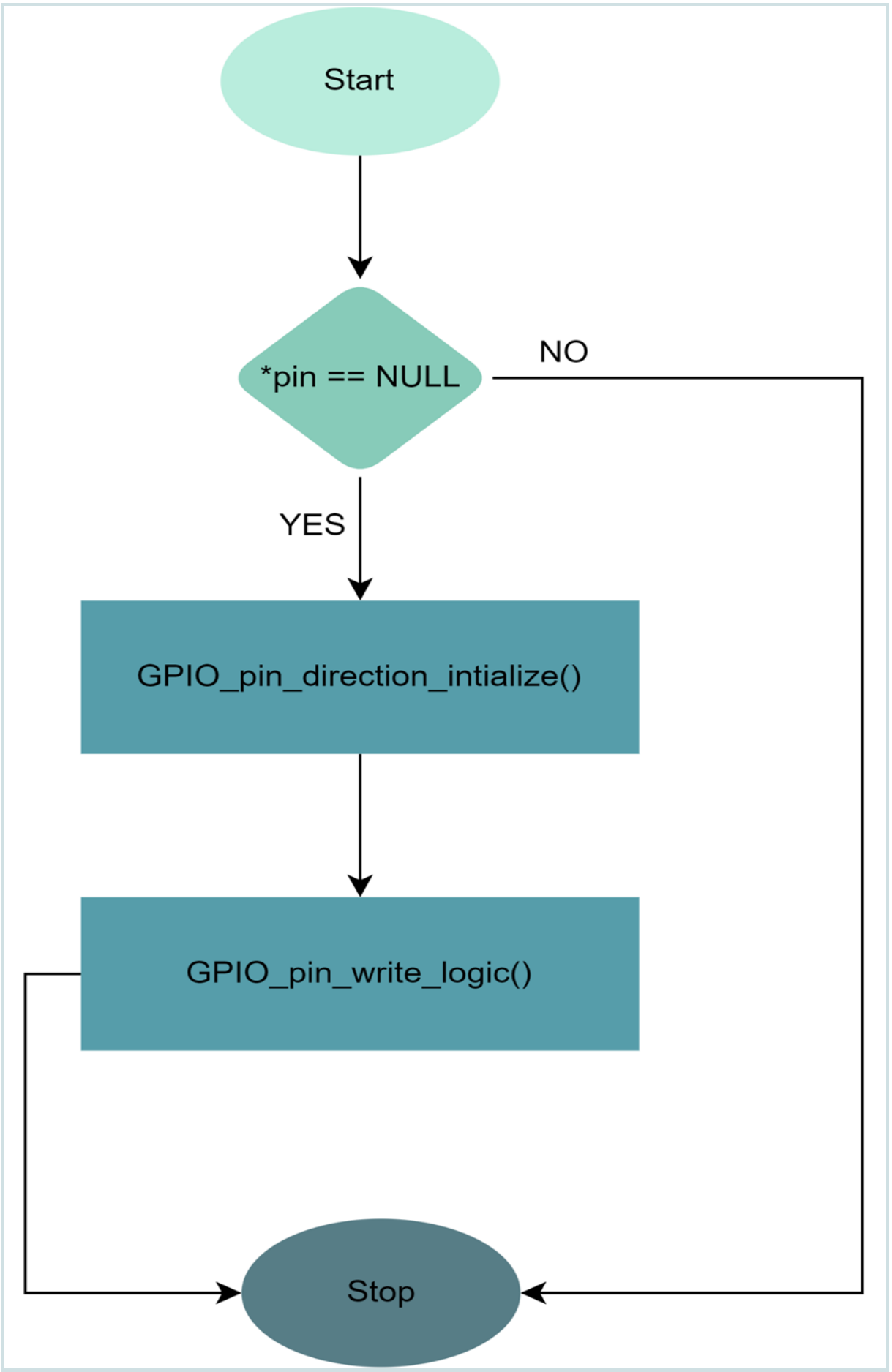
Parameters:
- _pin_config : A pointer to an instance of the ST_pin_config_t struct that contains the pin number,port number, direction, and initial logic level of the GPIO pin.

Return Type : Std_ReturnType. This is a standard type used in AUTOSAR (Automotive Open System Architecture) software development to indicate the success or failure of a function call.

The possible return values for this function are:
- E_OK : The function has completed successfully.
- E_NOT_OK : The function has encountered an error and could not complete successfully.

Overall, the GPIO_pin_intialize function provides a way to initialize a GPIO pin based on its configuration, allowing the software to set the initial direction and logic level of the pin as needed for its specific functionality.*/

Std_ReturnType GPIO_pin_intialize(const ST_pin_config_t *_pin_config);


Start
*pin == NULL
NO
YES
GPIO_pin_direction_intialize()
GPIO_pin_write_logic()
Stop

2.3.2.1 - USART Driver

Function Name: enu_uart_error_status_t uart_enable()

Parameters: None

Return Type: enu_uart_error_status_t

Description:
The uart_enable() function enables the UART module by setting the RXEN and TXEN bits in the UCSR B register. It does not take any input parameters and returns nothing. This function is typically called during initialization of the UART module in an embedded system.

Function Name: enu_uart_error_status_t uart_deinit()

Parameters: None

Return Type: enu_uart_error_status_t

Description:
The uart_deinit() function disables the UART module by clearing the RXEN and TXEN bits in the UCSR B register. It does not take any input parameters and returns nothing. This function is typically called during the shutdown or reset of the UART module in an embedded system, or when switching to a different communication protocol.

Function Name: enu_uart_error_status_t uart_init(const str_uart_instance_cfg_t *ptr_str_instance)

Parameters: const str_uart_instance_cfg_t *ptr_str_instance

Return Type: enu_uart_error_status_t

Description:
This Function To Config The Configuration Of The Uart Like Stop bit , Operation Mode , Char Size , Enable Or Disable Interrupt , Parity bit And Baud Rate .

Function Name: enu_uart_error_status_t uart_send(Uchar8_t *data_to_send)

Short Description:
The uart_send() function sends the data contained in the specified buffer over the UART module, one byte at a time. It does not return any data, but it does not return until all the data has been transmitted.

Parameters:
- data_to_send: a pointer to a buffer that contains the data to be sent over the UART module.

Return Type: enu_uart_error_status_t

Function Name: enu_uart_error_status_t uart_send_n(Uchar8_t *data_to_send , Uint16_t data_length)

Short Description:
The function sends the data contained in the specified buffer over the UART module, one byte at a time. It does not return any data, but it does not return until all the data has been transmitted.

Parameters:
- data_to_send: a pointer to a buffer that contains the data to be sent over the UART module.
- data_length : This Is The Length Of The Data Buffer

Return Type: enu_uart_error_status_t

Function Name: enu_uart_error_status_t uart_recive(Uchar8_t *data_to_send , Uint16_t data_length)

Short Description:
The function recive the data contained in the specified buffer over the UART module, one byte at a time. It does not return any data, but it does not return until all the data has been transmitted.

Parameters:
- data_to_send: a pointer to a buffer that contains the data to be revice over the UART module.
- data_length : This Is The Length Of The Data Buffer

Return Type: enu_uart_error_status_t

2.3.3 - ECUAL Drivers

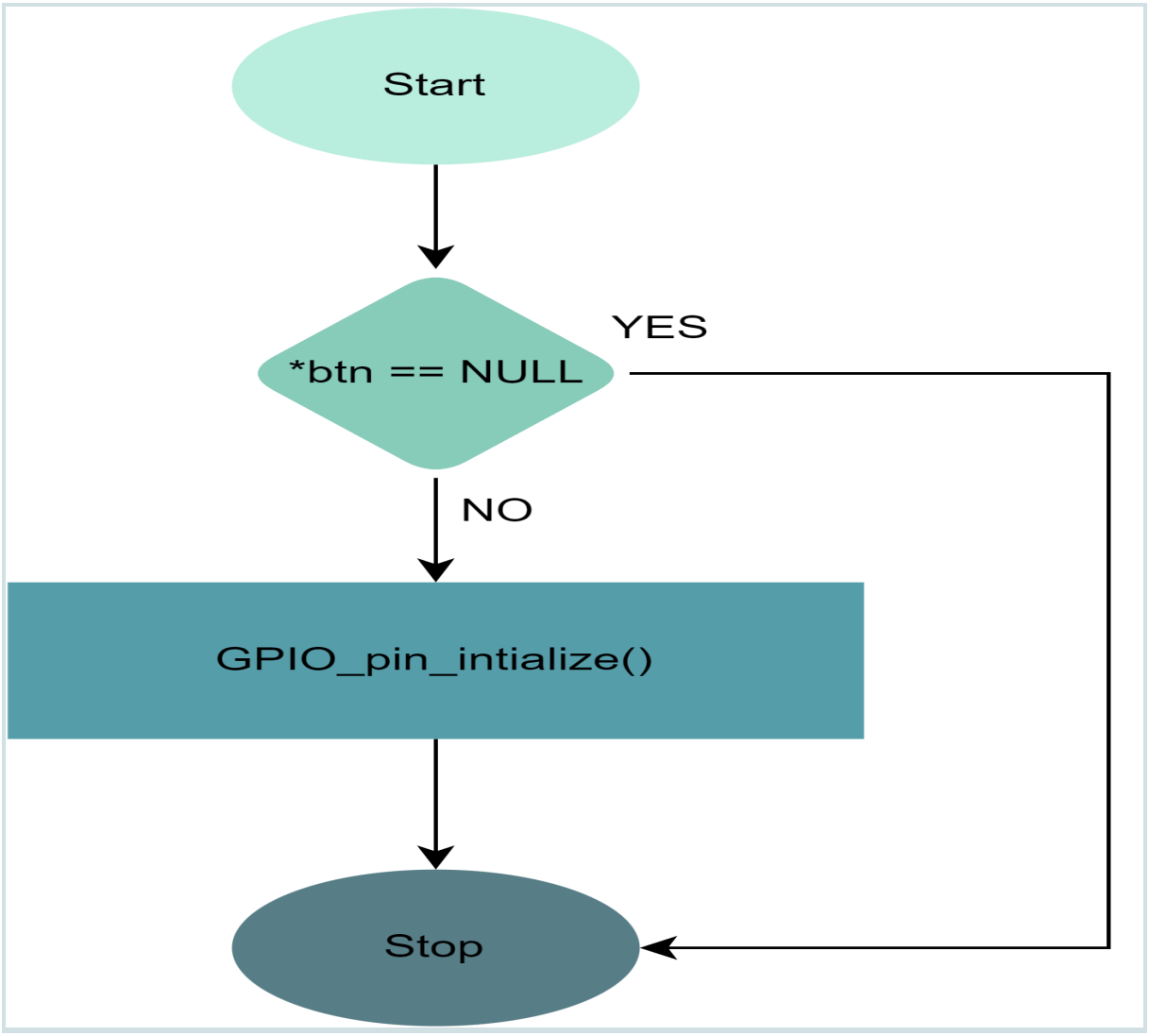
2.3.3.1 - LED Driver

Function Name: LED_initialize()

Parameters:
- led: a pointer to a structure of type ST_led_t that contains the configuration settings for the LED module.

Return Type: Std_ReturnType, a standardized enumerated data type that indicates the status of the LED initialization.

Description:
The LED_initialize() function initializes the LED module with the settings specified in the ST_led_t structure. It configures the LED pin as an output, sets the initial state of the LED to OFF, and returns a standardized enumerated data type that indicates whether the initialization was successful or if an error occurred.

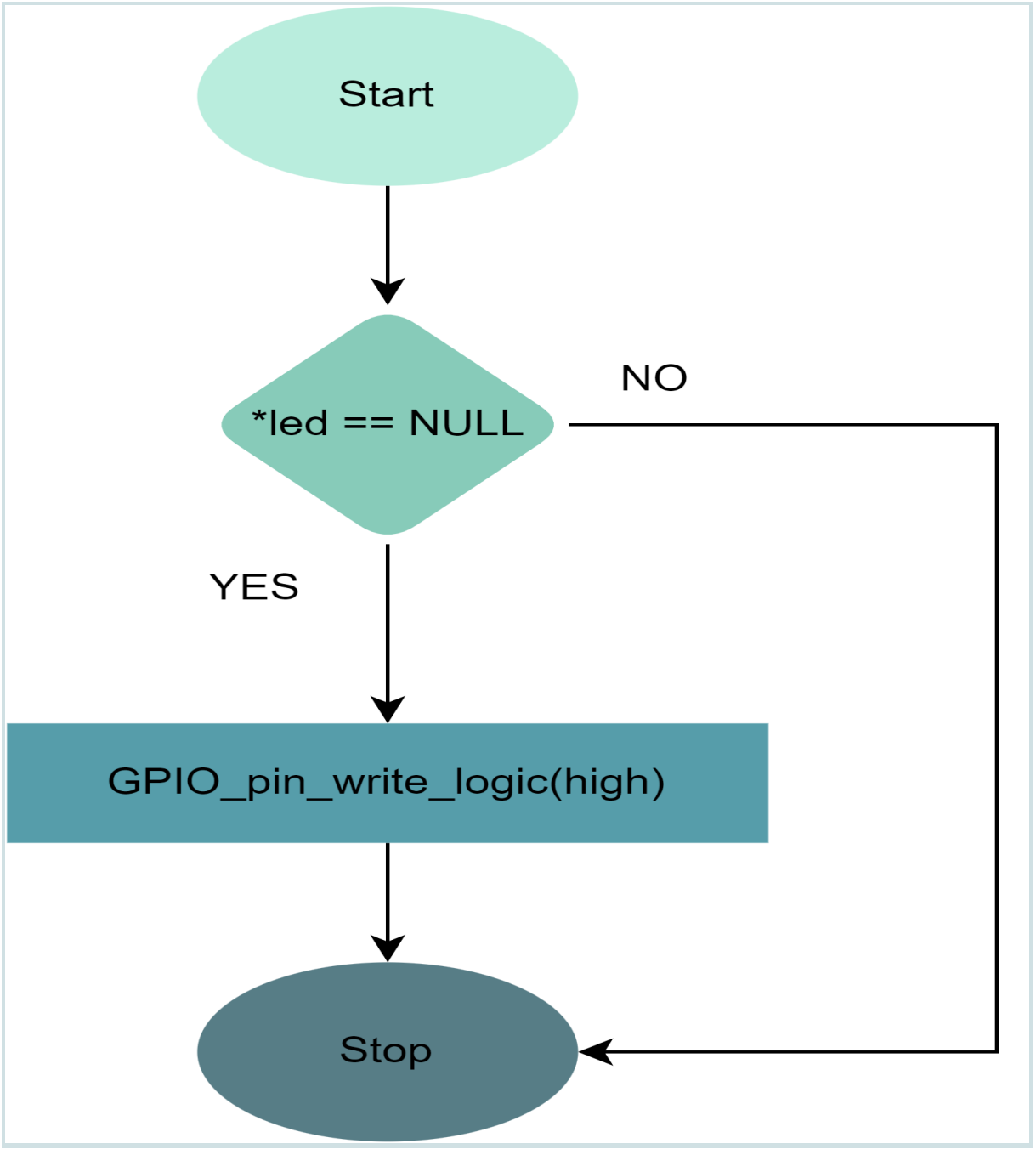


Function Name: LED_turn_on()

Parameters:

led: a pointer to a structure of type ST_led_t that contains the configuration settings for the LED module.
Return Type: Std_ReturnType, a standardized enumerated data type that indicates the status of the LED turn on operation.

Description:
The LED_turn_on() function turns on the LED specified by the ST_led_t structure. It sets the LED pin to HIGH, which turns on the LED. The function returns a standardized enumerated data type that indicates whether the operation was successful or if an error occurred.

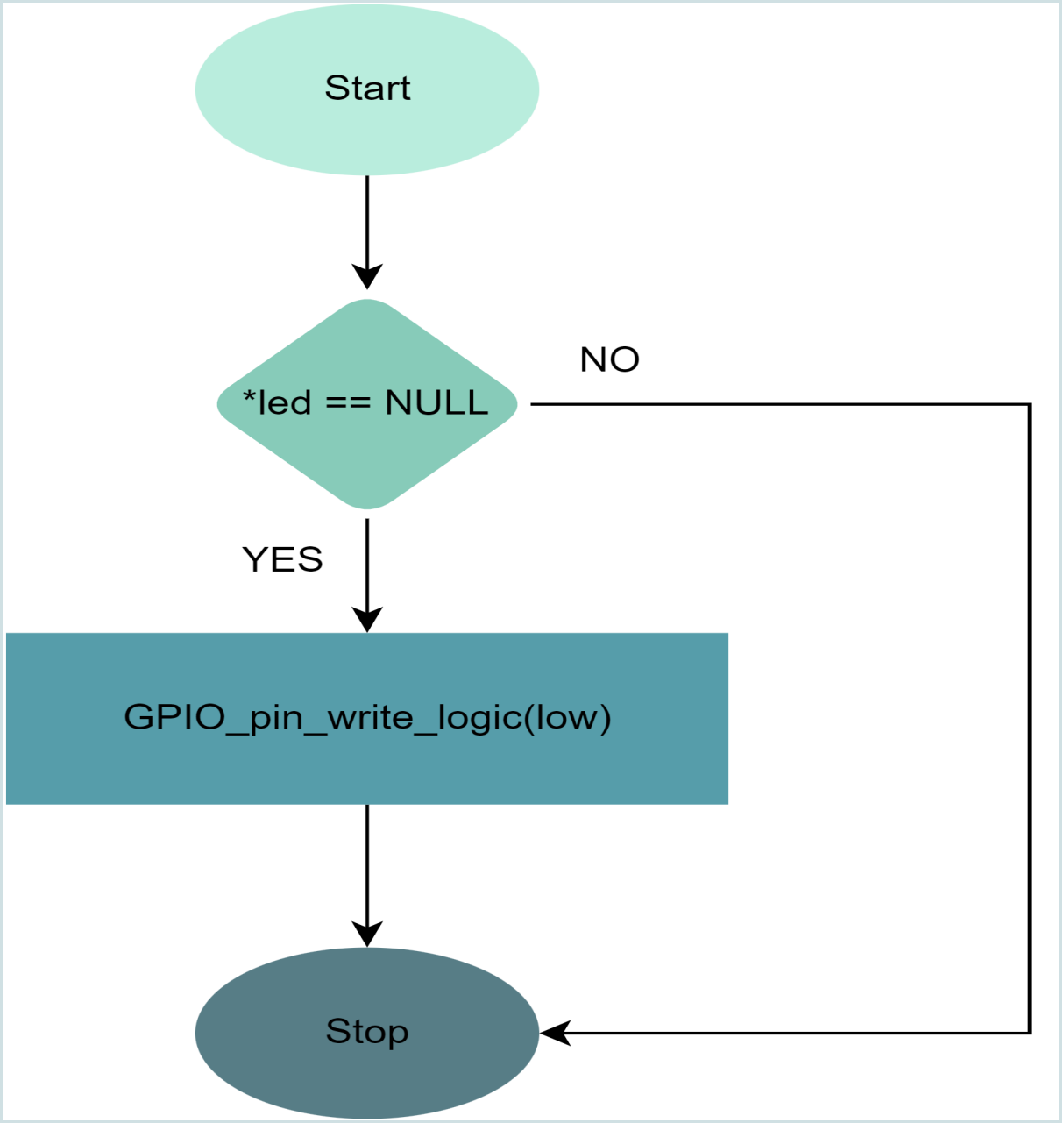


Function Name: LED_turn_off()

Parameters:

led: a pointer to a structure of type ST_led_t that contains the configuration settings for the LED module.
Return Type: Std_ReturnType, a standardized enumerated data type that indicates the status of the LED turn off operation.

Description:
The LED_turn_off() function turns off the LED specified by the ST_led_t structure. It sets the LED pin to LOW, which turns off the LED. The function returns a standardized enumerated data type that indicates whether the operation was successful or if an error occurred.

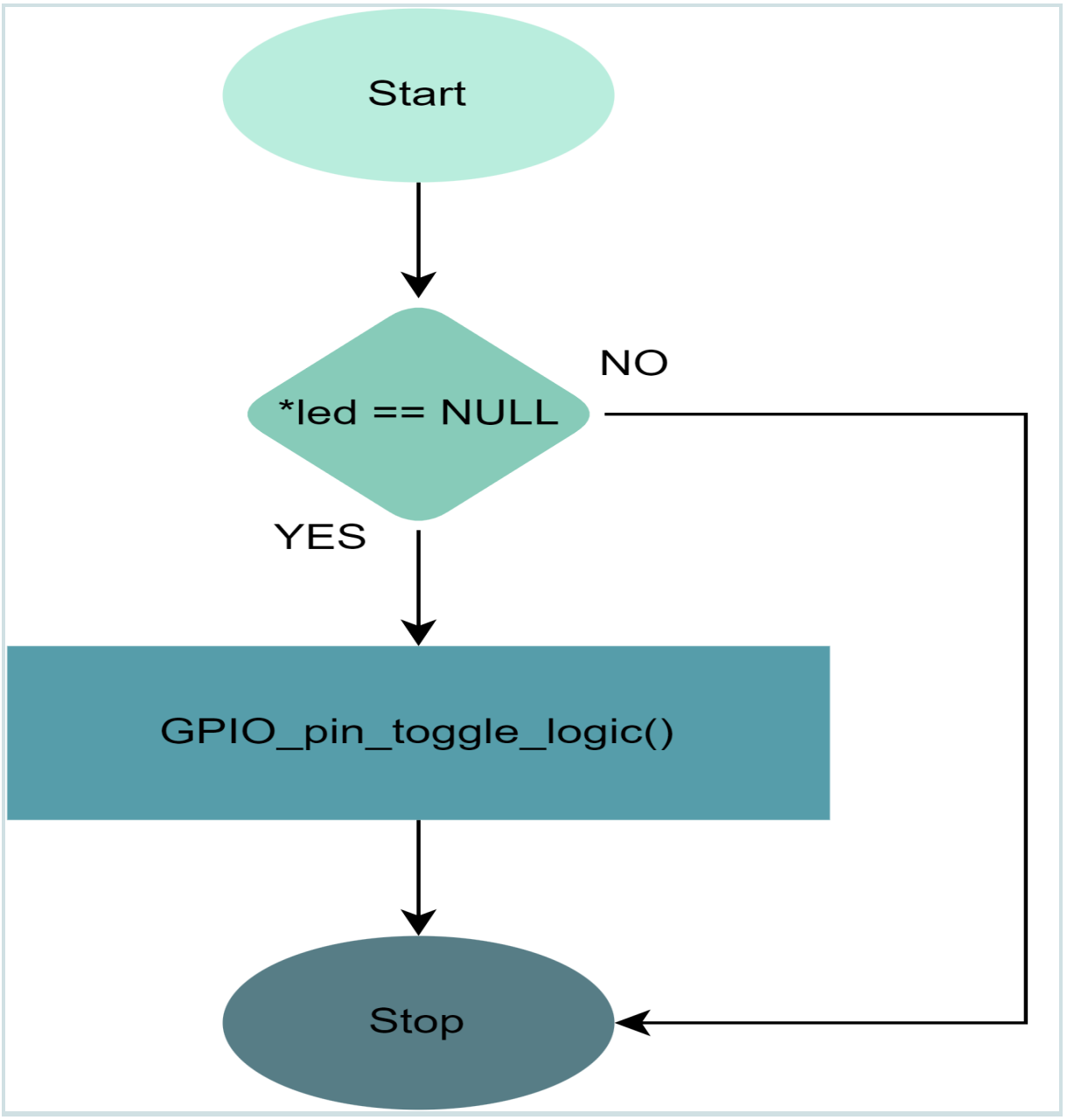


Function Name: LED_toggle()

Parameters:

led: a pointer to a structure of type ST_led_t that contains the configuration settings for the LED module.
Return Type: Std_ReturnType, a standardized enumerated data type that indicates the status of the LED toggle operation.

Description:
The LED_toggle() function toggles the state of the LED specified by the ST_led_t structure. If the LED is currently on, the function turns it off, and if it is off, the function turns it on. The function returns a standardized enumerated data type that indicates whether the operation was successful or if an error occurred.



2.3.4 - Services Layer

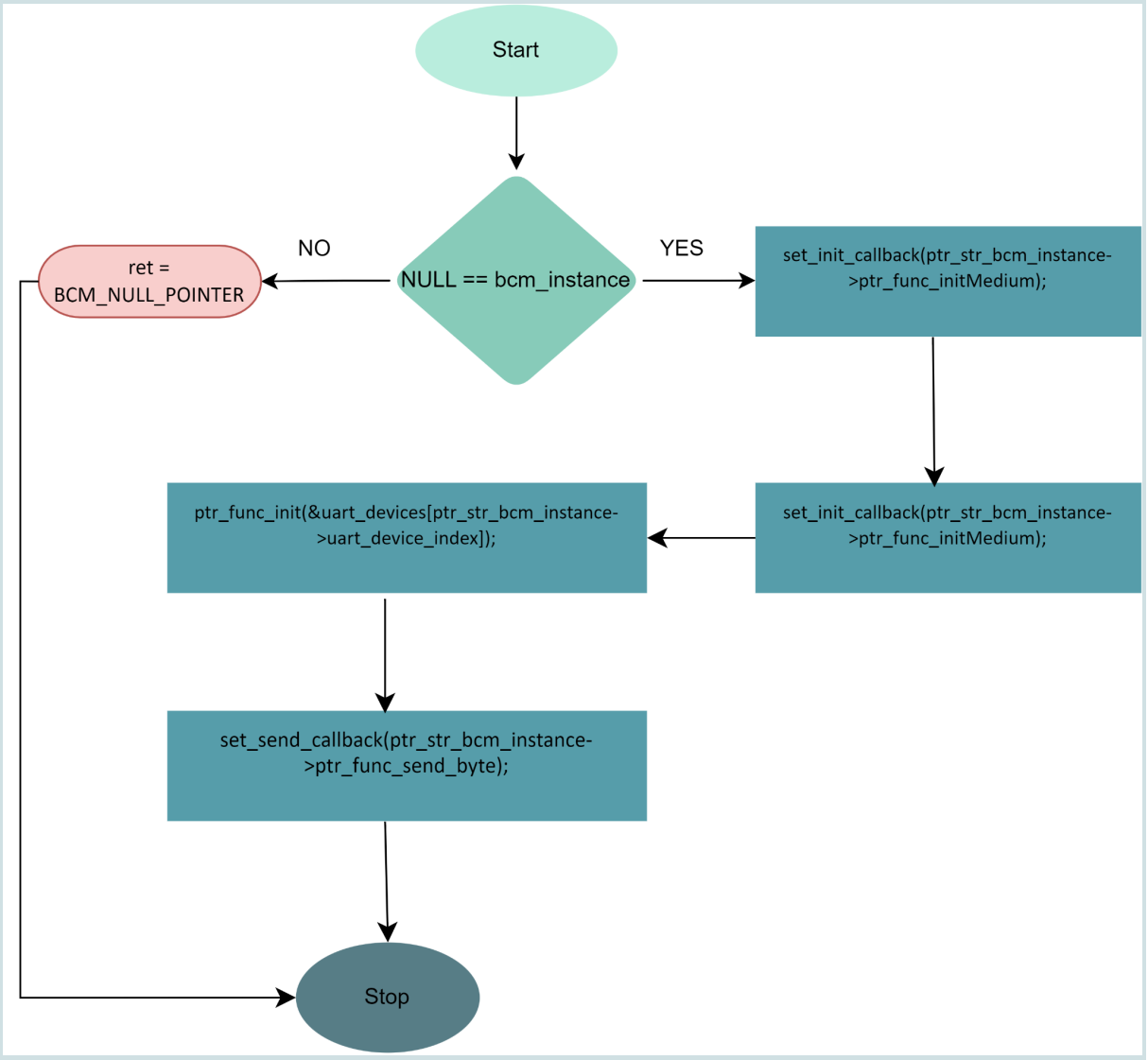
2.3.4.1 - BCM Module

Function Name: enu_bcm_states_code_t bcm_init(str_bcm_instance_t * ptr_str_bcm_instance)

Parameters:

ptr_str_bcm_instance: a pointer to a structure of type str_bcm_instance_t that contains the configuration settings for the BCM.
Return Type: enu_bcm_states_code_t, an enumerated data type that indicates the status of the BCM initialization.

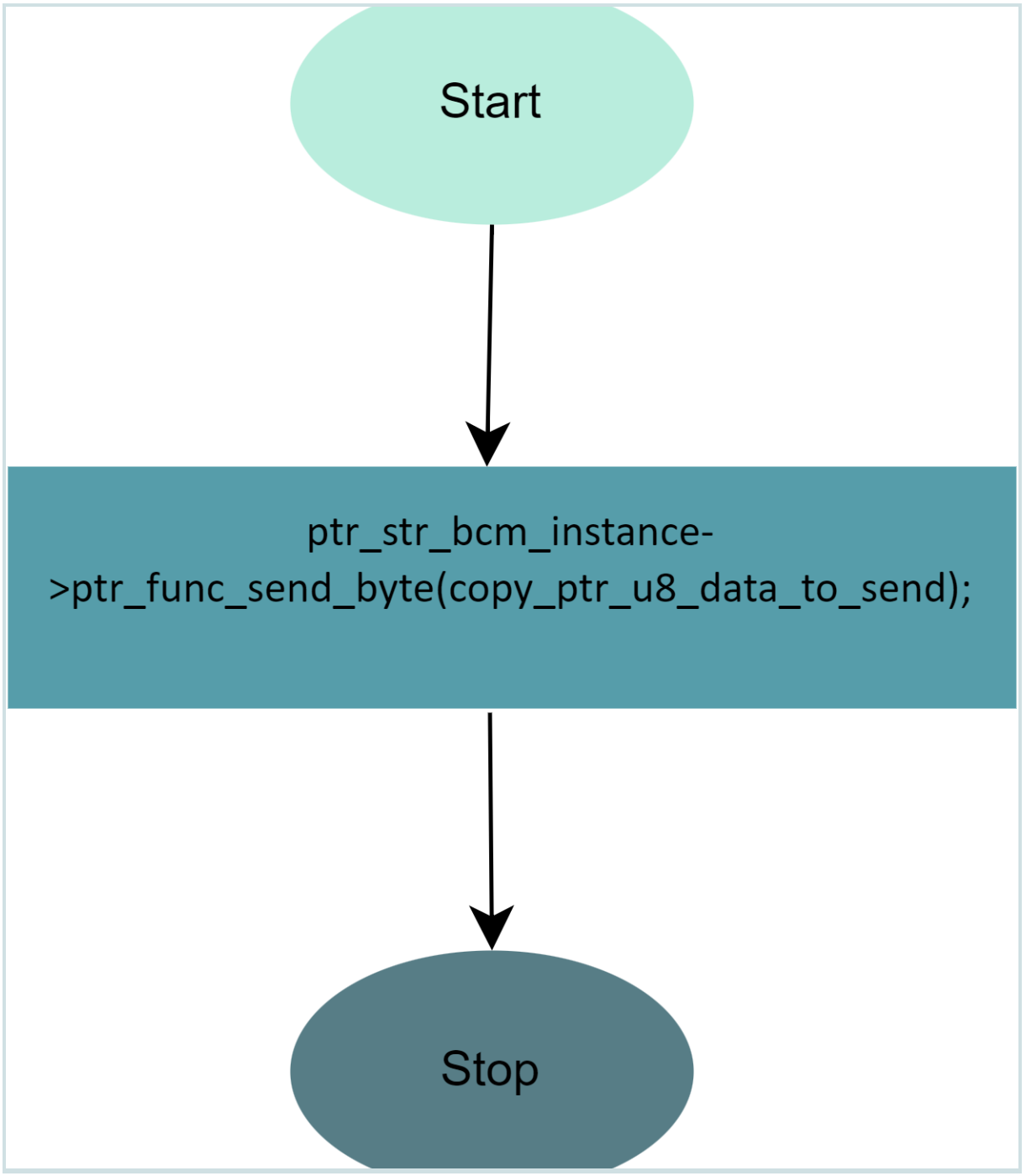
Description:This Function Use To Call Back The Init Function Of The Using Communication Protocol



Function Name: enu_bcm_states_code_t bcm_send(str_bcm_instance_t * ptr_str_bcm_instance , Uchar8_t *copy_ptr_u8_data_to_send)

Parameters:
ptr_str_bcm_instance: a pointer to a structure of type str_bcm_instance_t that contains the configuration settings for the BCM.
Return Type: enu_bcm_states_code_t, an enumerated data type that indicates the status of the BCM initialization.

Description:This Function Use To Call Back The Send Function Of The Using Communication Protocol

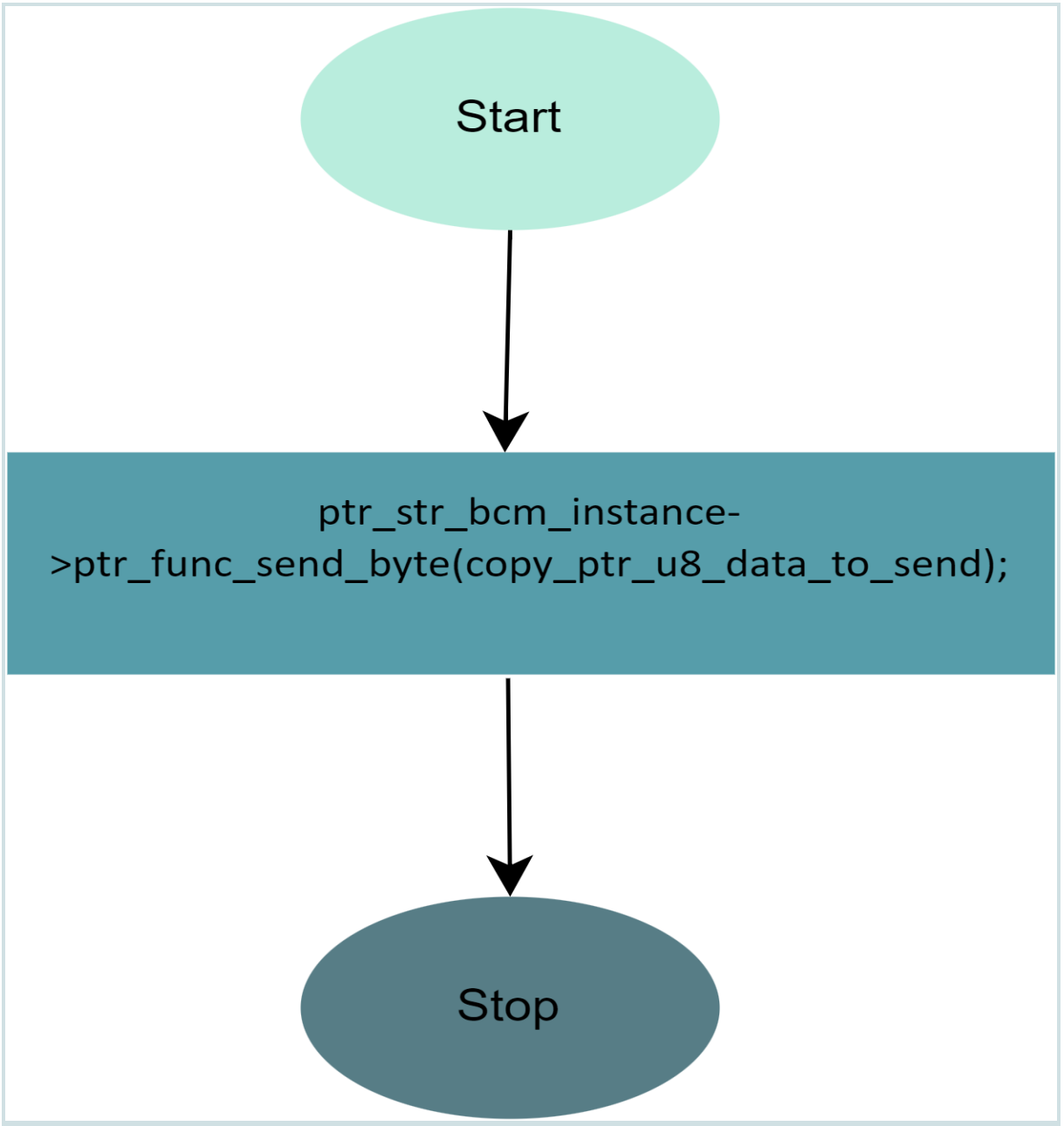


Function Name: enu_bcm_states_code_t bcm_deinit(str_bcm_instance_t * ptr_str_bcm_instance)

Parameters:

ptr_str_bcm_instance: a pointer to a structure of type str_bcm_instance_t that contains the configuration settings for the BCM.
Return Type: enu_bcm_states_code_t, an enumerated data type that indicates the status of the BCM initialization.

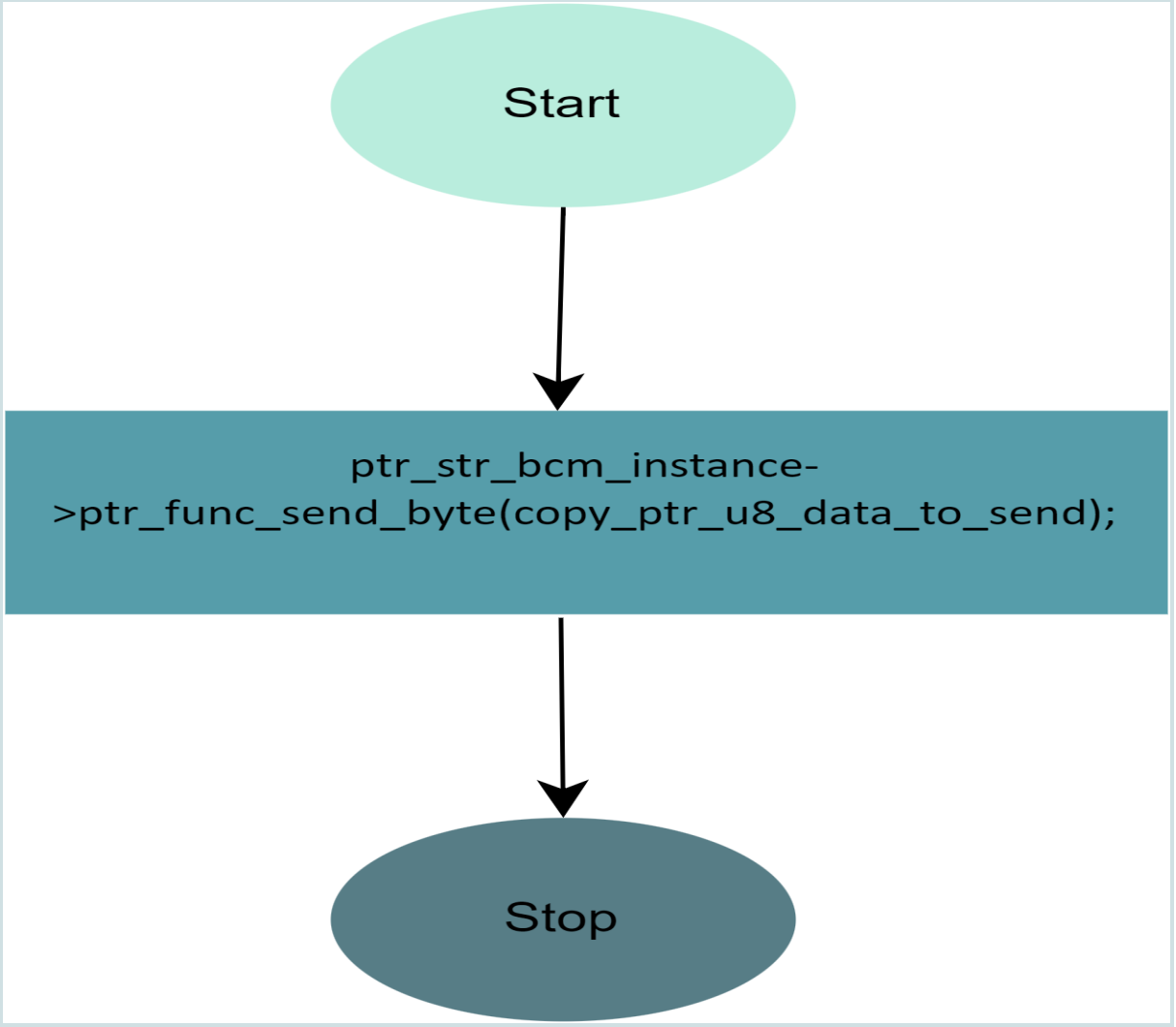
Description:This Function Use To Call Back The deinit Function Of The Using Communication Protocol



Function Name: enu_bcm_states_code_t bcm_send(str_bcm_instance_t * ptr_str_bcm_instance , Uchar8_t *copy_ptr_u8_data_to_send)

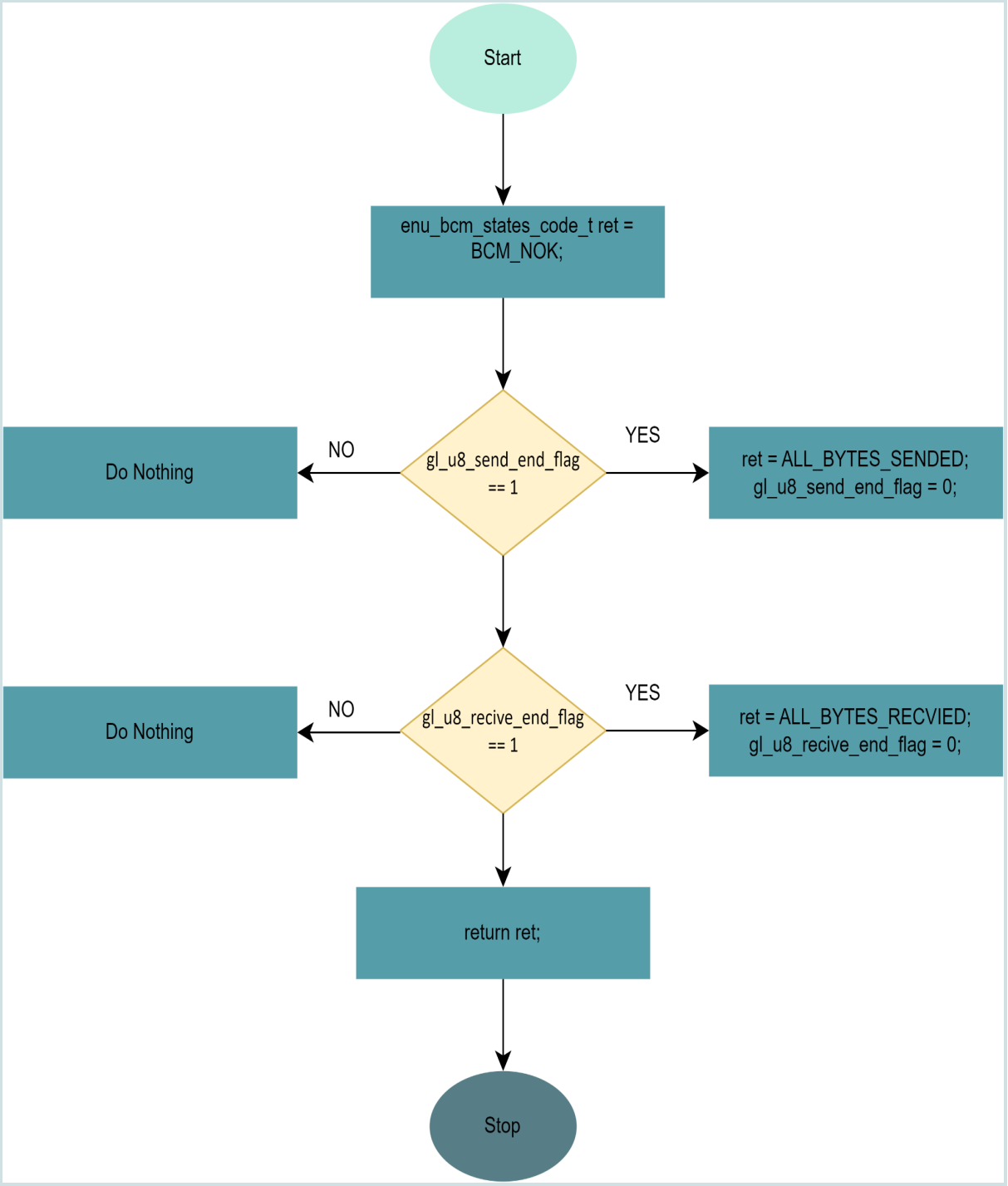
Parameters:
ptr_str_bcm_instance: a pointer to a structure of type str_bcm_instance_t that contains the configuration settings for the BCM.
Return Type: enu_bcm_states_code_t, an enumerated data type that indicates the status of the BCM initialization.

Description:This Function Use To Call Back The Send Function Of The Using Communication Protocol



Function Name: enu_bcm_states_code_t bcm_dispatcher(void);

Parameters:
ptr_str_bcm_instance: void
Return Type: enu_bcm_states_code_t, an enumerated data type that indicates the status of the BCM initialization.
Description:bcm_dispatcher_sender will execute the periodic actions and notifies the user with the needed events over a specific BCM instance

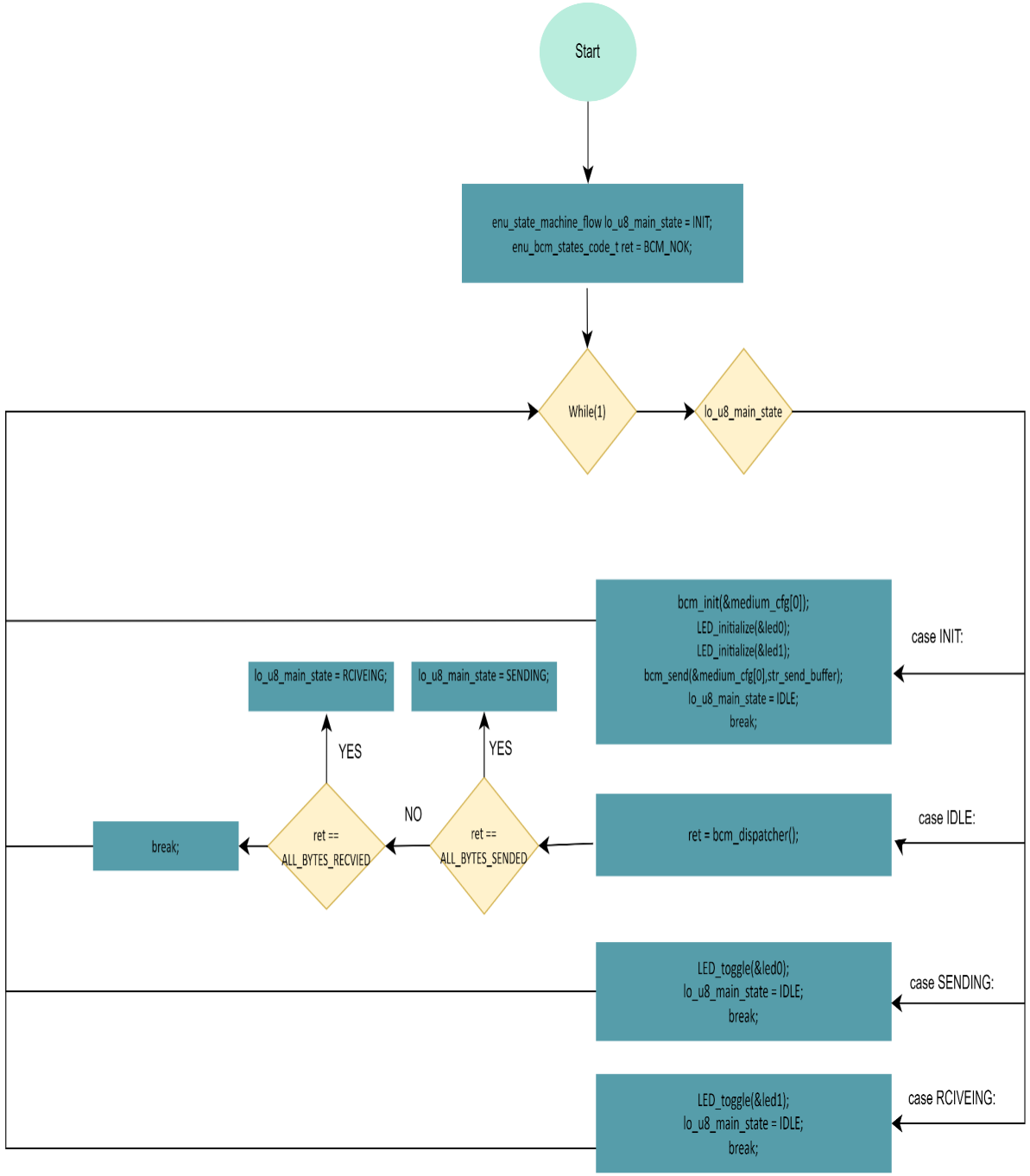


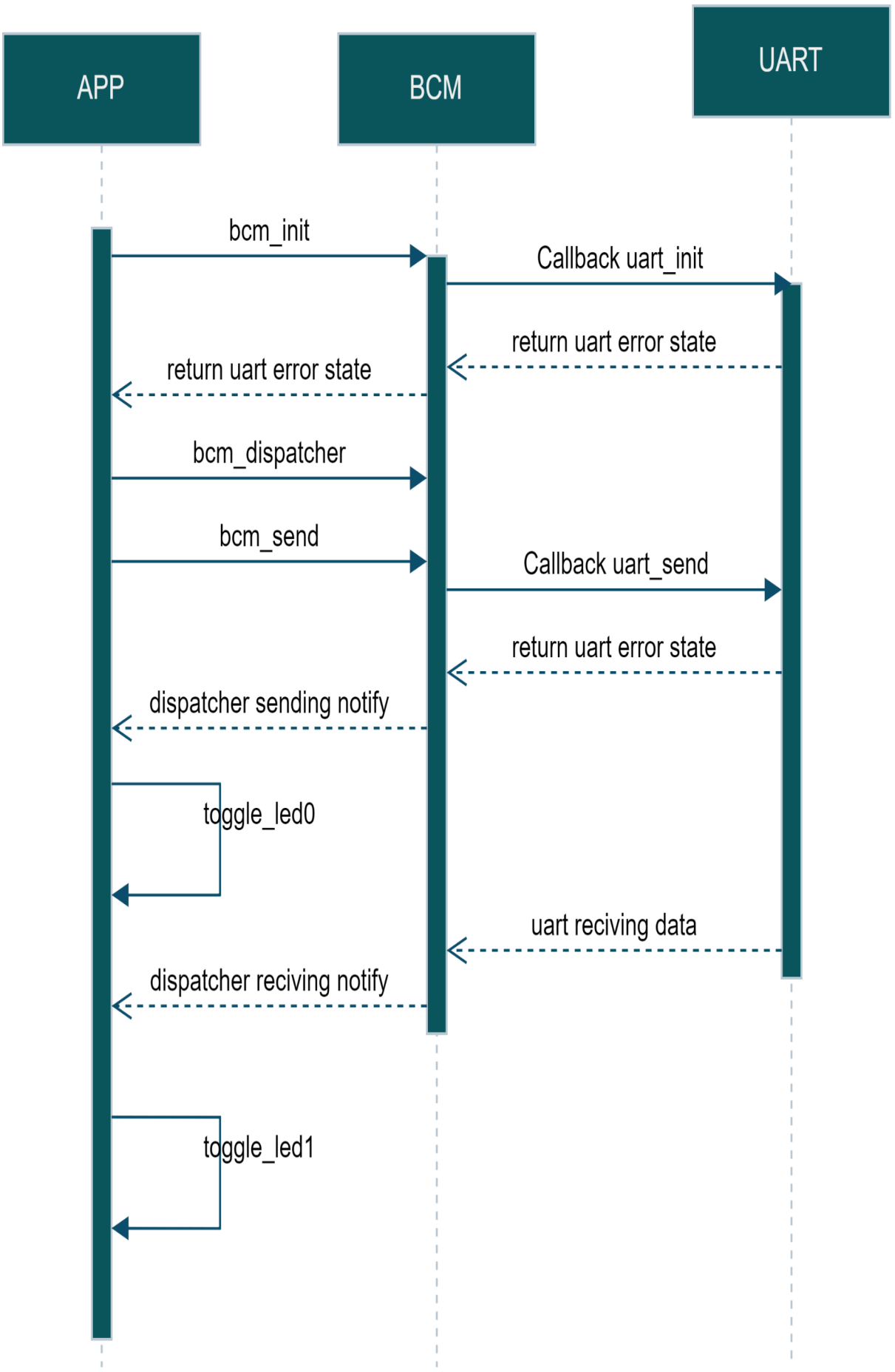
2.3.5 - APP Layer

2.3.5.1 - APP Module

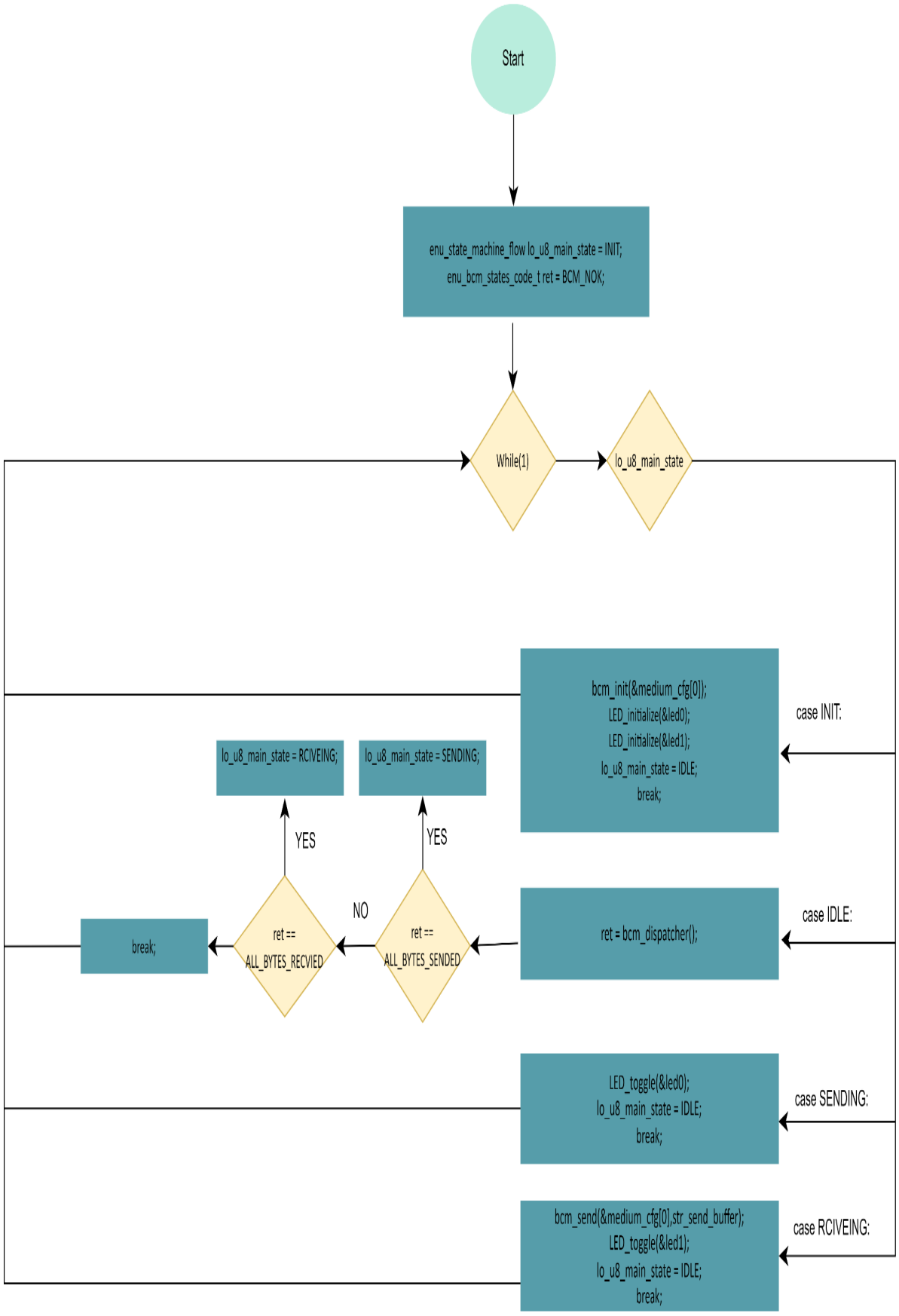
This App Will Handle The Flow Of The Application And What DecisionTo Take When The Dispatcher Notify It With A Operation Is Done From The BCM Module

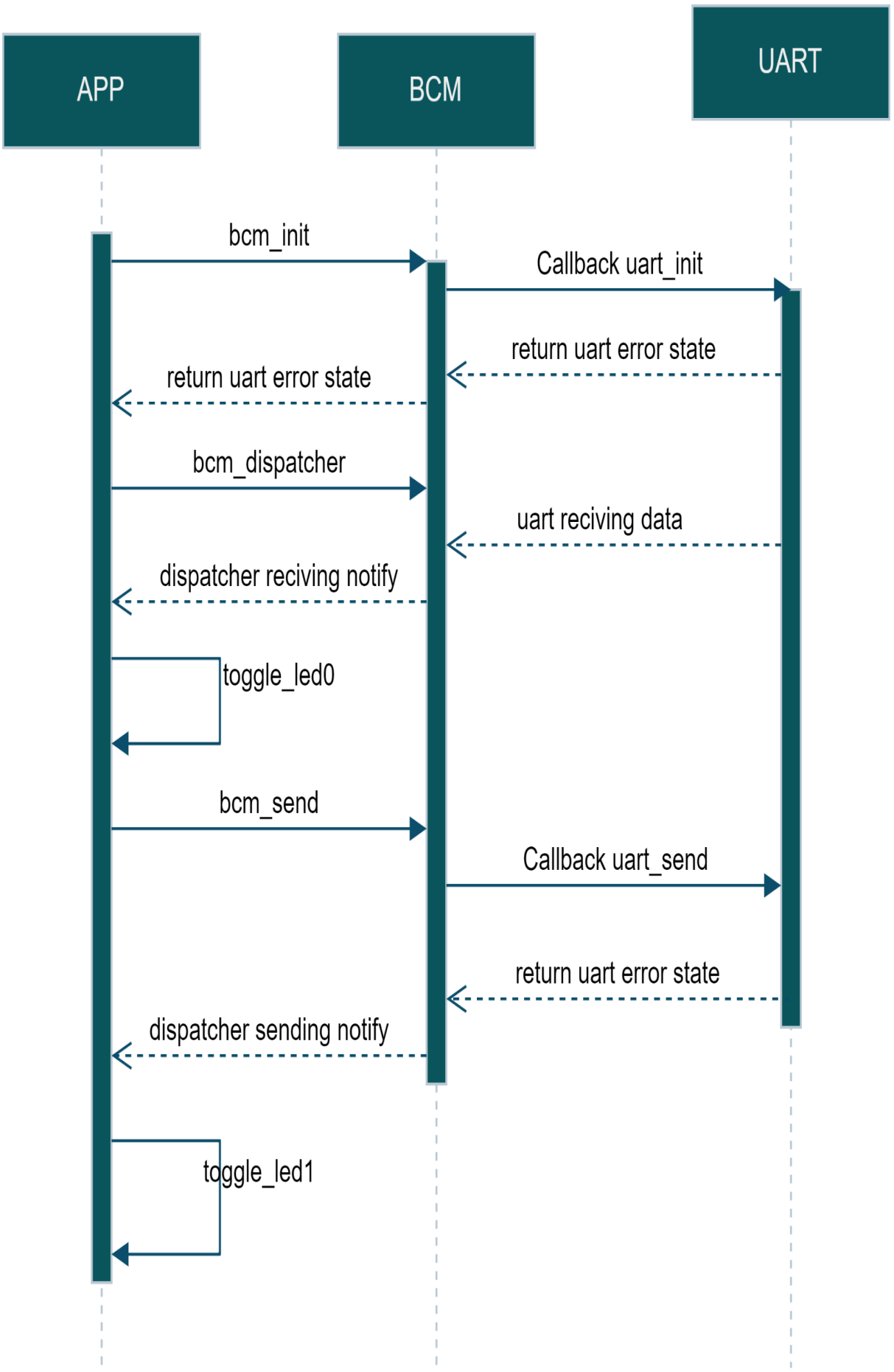
2.3.5.1.1 - Sender Mcu




APP
BCM
UART
bcm_init
Callback uart_init
return uart error state
return uart error state
bcm_dispatcher
bcm_send
Callback uart_send
return uart error state
dispatcher sending notify
toggle_led0
uart reciving data
dispatcher reciving notify
toggle_led1

2.3.5.1.2 - Receiver Mcu




APP
BCM
UART
bcm_init
Callback uart_init
return uart error state
return uart error state
bcm_dispatcher
uart reciving data
dispatcher reciving notify
toggle_led0
bcm_send
Callback uart_send
return uart error state
dispatcher sending notify
toggle_led1

3 - Low Level Design

3.1 - Prelinking Configuration

3.1.2 - USART Module Prelinking configuration

```
typedef enum
{
	ONE_STOP_BIT = 0,
	TWO_STOP_BIT
}enu_stop_bit_cfg_t;


typedef enum
{
	PARITY_DISABLED = 0,
	PARITY_ENABLED_EVEN,
	PARITY_ENABLED_ODD
}enu_parity_bit_cfg_t;

typedef enum
{
	ASYNCHRONOUS_OPERATION = 0,
	SYNCHRONOUS_OPERATION
}enu_operation_mode_cfg_t;

typedef enum
{
	CHARACTER_SIZE_5_BIT = 0,
	CHARACTER_SIZE_6_BIT,
	CHARACTER_SIZE_7_BIT,
	CHARACTER_SIZE_8_BIT,
	CHARACTER_SIZE_9_BIT
}enu_character_size_cfg_t;

typedef enum
{
	ENABLE_INTERRUPT = 0,
	DISABLE_INTERRUPT
}enu_interrupt_cfg_t;

typedef enum
{
	SPEED_U1X = 0,
	SPEED_U2X
}enu_transmission_speed_cfg_t;
```

```
typedef enum
{
	UART_BR1X_2400 = 207,
	UART_BR1X_4800 = 103,
	UART_BR1X_9600 = 51,
	UART_BR1X_38400 = 12
}enu_baud_rate_speed_u1x_cfg_t;

typedef enum{
	UART_BR2X_2400 = 416,
	UART_BR2X_4800 = 207,
	UART_BR2X_9600 = 103,
	UART_BR2X_38400 = 25
}enu_baud_rate_speed_u2x_cfg_t;


typedef enum
{
	UART_INIT_OK = 0,
	UART_INIT_NOK,
	UART_SEND_OK,
	UART_SEND_NOK,
	UART_RECIVE_OK,
	UART_RECIVE_NOK,
	UART_NULL_PTR,
	UART_HANDLLER_SET_OK
}enu_uart_error_status_t;

const str_uart_instance_cfg_t uart_devices[1] =
{
	{
		.enu_baud_rate_2x = UART_BR2X_9600,
		.enu_baud_rate_1x = UART_BR1X_9600,
		.enu_char_size = CHARACTER_SIZE_8_BIT,
		.enu_interrupt_status = ENABLE_INTERRUPT,
		.enu_operation_mode = ASYNCHRONOUS_OPERATION,
		.enu_parity_bit = PARITY_DISABLED,
		.enu_stop_bit = ONE_STOP_BIT,
		.enu_transmission_speed = SPEED_U1X,
		.ptr_func_tx_callback = uart_send_n,
		.ptr_func_rx_callback = uart_recive
	}
};
```

4 - High Quality Of Diagrams Link