Sprints

BASIC COMMUNICATION MANAGER

BCM

Design and Implementation for basic communication manager (BCM), This module has the capability to work with different serial communication protocols using ISR with the highest possible throughput.

Prepared By

Hossam Elwahsh

Sprints

1. Project Introduction....4
1.1. Project Components....4
1.2. System Requirements....5
1.3. Assumptions....8
2. High Level Design....9
2.1. System Architecture....9
2.1.1. Definition....9
2.1.2. Layered Architecture....9
2.1.3. Project Schematic....10
2.2. Modules Description....11
2.2.1. DIO (Digital Input/Output) Module....11
2.2.2. UART Module....11
2.2.3. LED Module....11
2.2.4. BCM Module....11
2.2.5. CIRCULAR_QUEUE Module....12
2.3. Drivers' Documentation (APIs)....13
2.3.1 Definition....13
2.3.2. MCAL APIs....13
2.3.2.1. DIO Driver....13
2.3.2.2. UART Driver....16
2.3.3. HAL APIs....18
2.3.3.1. LED APIs....18
2.3.4. SERV APIs....19
2.3.4.1. BCM APIs....19
2.3.4. APP APIs....20
3. Low Level Design....21
3.1. MCAL Layer....21
3.1.1. DIO Module....21
3.1.1.a. Pin/Port Check sub process....21
3.1.1.1. DIO_init....22
3.1.1.2. DIO_read....22
3.1.1.3. DIO_write....22
3.1.1.4. DIO_toggle....24
3.2. HAL Layer....25
3.2.1. LED Module....25
3.2.1.1. LED_init....25
3.2.1.2. LED_on....25
3.2.1.3. LED_off....26
3.2.1.4. LED_toggle....26
3.3. SERV Layer....27
3.3.1. BCM Module....27
3.3.1.1. bcm_init....27

3.3.1.2. bcm_deinit....28
3.3.1.3. bcm_send....29
3.3.1.4. bcm_send_n....30
3.3.1.5. bcm_receive....31
3.3.1.6. bcm_dispatcher....32
3.3.1.7. bcm_setCallback....33
3.4. APP Layer....34
3.4.1. MCU1....34
3.4.1.1. app_initialize....34
3.4.1.2. app_start (click for HQ)....35
3.4.2. MCU2....36
3.4.2.1. app_initialize....36
3.4.2.2. app_start (click for HQ)....37
4. Configurations....38
4.1. UART Driver....38
4.1.1. Preconfiguration....38
4.1.2. Linking Configuration....39
4.2. BCM Driver - Linking Configuration....39
4.3. Circular Queue (UTILITY) - Preconfiguration....39
4.4. App preconfiguration....40

Basic Communication Manager Design and Implementation

1. Project Introduction

Introducing the Basic Communication Manager (BCM) project, where our goal is to design a highly efficient module for handling various serial communication protocols. We aim to achieve the maximum possible data throughput. Our project focuses on developing a flexible and robust BCM solution, ensuring seamless integration with diverse systems and meeting industry standards. Through rigorous research, planning, and execution, we aim to deliver a reliable and optimized BCM module for efficient and rapid data transmission.

1.1. Project Components

- 2x ATmega32 microcontroller
- 4x LEDs (2 for each µController)
    - Green LED: Transmit Complete
    - Blue LED: Receive Complete
- DEBUG TOOLS:
    - Virtual Terminal (U1 Simulator)
        - Simulates MCU 1 UART Module
    - Virtual Terminal (U2 Simulator)
        - Simulates MCU 2 UART Module

1.2. System Requirements

System 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** using the below table. This function will initialize the corresponding serial communication protocol

Function Name	bcm_init
Syntax	`Enu_system_status_t bcm_init(str_bcm_instance_t * ptr_str_bcm_instance)`
Sync/Async	Synchronous
Reentrancy	Non reentrant
Parameters (in)	Ptr_str_bcm_instance: address for bcm instance
Parameters (out)	None
Parameters (in, out)	None
Return	`typedef enum { BCM_SYSTEM_OK = 0, // config errors BCM_ERROR_BAD_CFG, // operational errors BCM_INVALID_DATA_LENGTH, BCM_ERROR_OPR_NULL_PTR_GIVEN, BCM_ERROR_OPR_SYN_FAILED, /* Still Sending or Receiving Data */ BCM_ERROR_OPR_BUSY, /* Trying to read received data while it's empty */ BCM_ERROR_OPR_NO_DATA_TO_READ, BCM_ERROR_INTERNAL_SYS_FAILURE, }enu_system_status_t;`

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

Function Name	**bcm_deinit**
Syntax	Enu_system_status_t bcm_init(str_bcm_instance_t * ptr_str_bcm_instance)
Sync/Async	Synchronous
Reentrancy	Non reentrant
Parameters (in)	Ptr_str_bcm_instance: address for bcm instance
Parameters (out)	None
Parameters (in, out)	None
Return	Enu_system_status_t (same as above)

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

Function Name	**bcm_send**
Syntax	`enu_system_status_t bcm_send(const str_bcm_instance_t * ptr_str_bcm_instance, uint8_t_ uint8_dataByte);`
Sync/Async	Synchronous/Async (depends on comm protocol config)
Reentrancy	Re-entrant

Parameters (in)	`ptr_str_bcm_instance: address for bcm instance`
	`str_send_queue: address to send queue`
	`uint8_dataByte: data byte to send`

Parameters (out)	None
Parameters (in, out)	None
Return	`Enu_system_status_t (same as above)`

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_system_status_t bcm_send_n(const str_bcm_instance_t * ptr_str_bcm_instance,`

	`str_circularqueue_t_ ** str_send_queue, uint16_t_ uint16_dataLength);`
Sync/Async	Synchronous/Async (depends on comm protocol config)
Reentrancy	Re-entrant

Parameters (in)	`ptr_str_bcm_instance: address for bcm instance`
	`str_send_queue: address to send queue`
	`uint16_dataLength: data length`

Parameters (out)	None
Parameters (in, out)	None
Return	`Enu_system_status_t (same as above)`

7. Implement **bcm_receive** will receive more one or more bytes with a length n over a specific BCM instance

Function Name	**bcm_receive**
Syntax	`enu_system_status_t bcm_receive(const str_bcm_instance_t * ptr_str_bcm_instance, str_circularqueue_t_ ** str_rec_queue, uint16_t_ * uint16_received_data_length)`
Sync/Async	Synchronous/Async (depends on comm protocol config)
Reentrancy	Re-entrant
Parameters (in)	`ptr_str_bcm_instance: address for bcm instance`
Parameters (out)	None
Parameters (in, out)	`str_rec_queue: address for receive queue` `uint16_received_data_length: data received length`
Return	`Enu_system_status_t (same as above)`

8. 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	`bcm_dispatcher(const str_bcm_instance_t * str_bcm_instance)`

Sync/Async	Synchronous
Reentrancy	Non-reentrant
Parameters (in)	`ptr_str_bcm_instance: address for bcm instance`
Parameters (out)	None
Parameters (in, out)	`None`
Return	`Enu_system_status_t (same as above)`

9. **bcm_setCallback** Sets callback pointer to which BCM will route events that BCM finds important to be processed outside

Function Name	**bcm_setCallback**
Syntax	`enu_system_status_t bcm_setCallback(void (*ptr_callback)(uint8_t_ uint8_instance_id, str_operation_info_t_ str_operation_info))`
Sync/Async	Synchronous
Reentrancy	Reentrant

Parameters (in)	`uint8_instance_id: address for bcm instance`
	`str_operation_info: info on operation`
	
	`TX_COMPLETE = 0,`
	`TX_FAIL = 1,`
	`RX_COMPLETE = 2`

Parameters (out)	None
Parameters (in, out)	`None`
Return	`Enu_system_status_t (same as above)`

2. High Level Design

2.1. System Architecture

2.1.1. Definition

Layered Architecture (*Figure 1*) describes an architectural pattern composed of several separate horizontal layers that function together as a single unit of software.

2.1.2. Layered Architecture

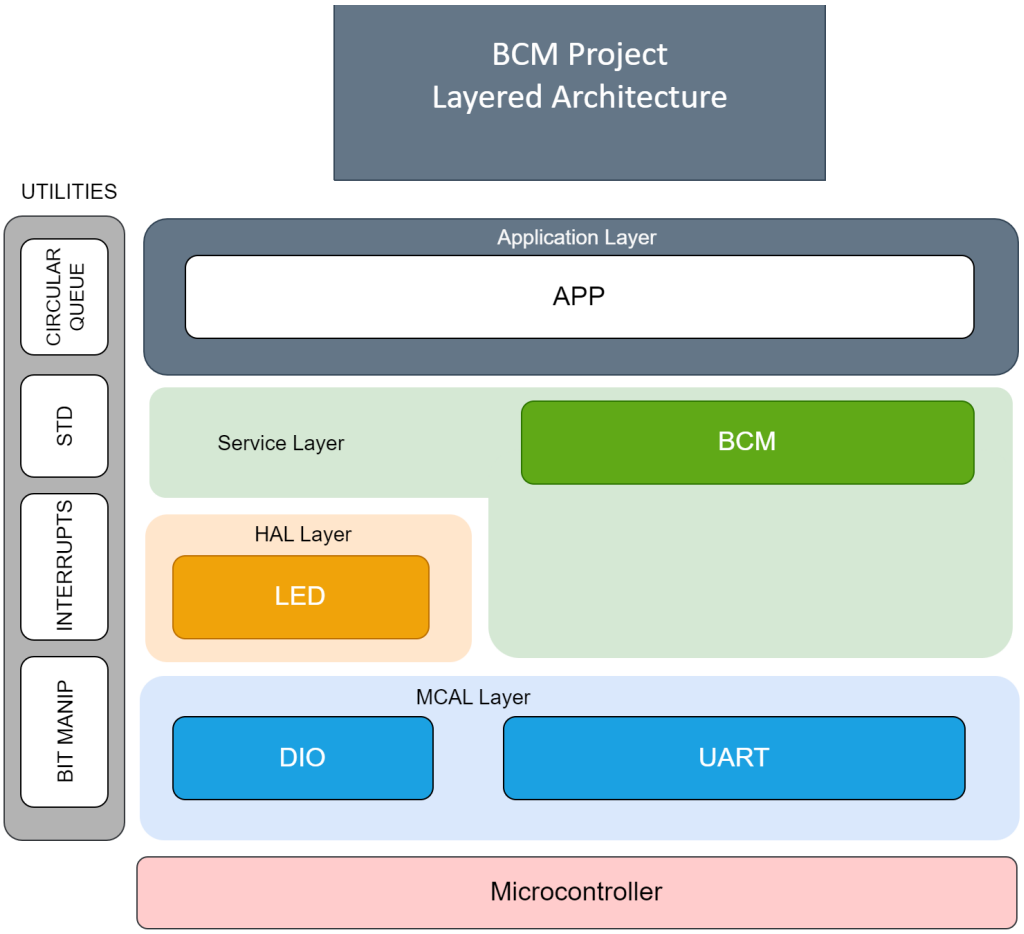


Figure 1. Layered Architecture Design

2.1.3. Project Schematic

Click for HQ

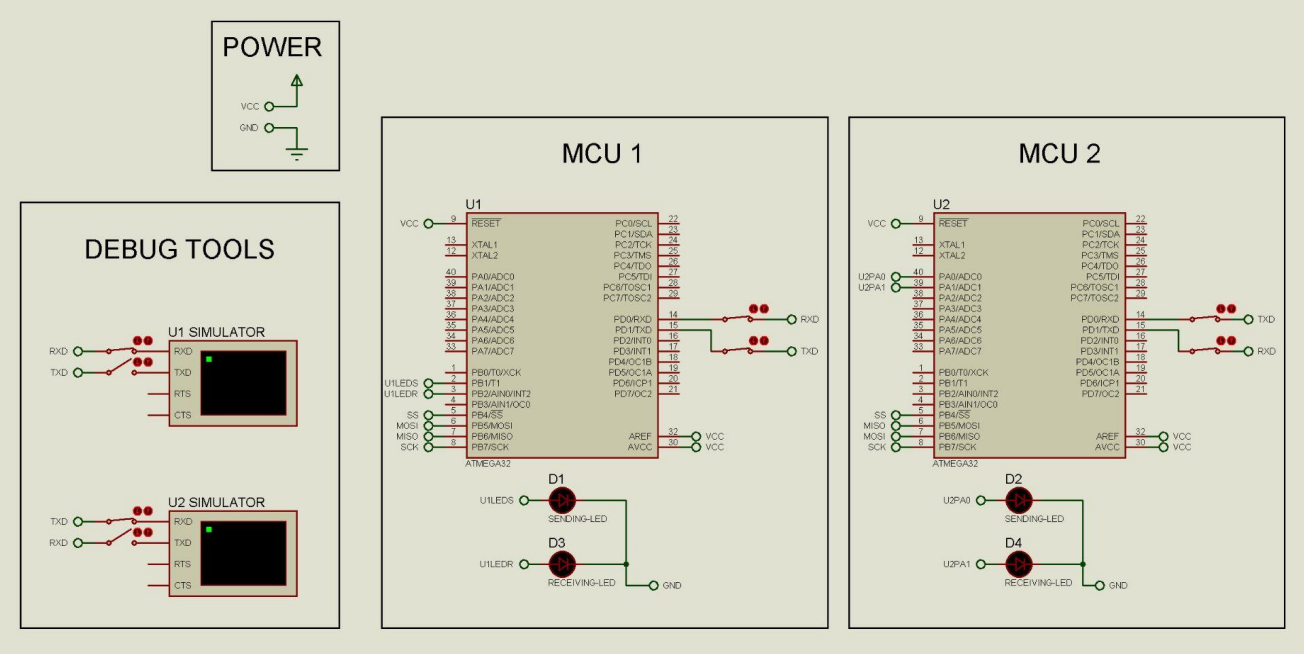


Figure 2. Project Schematic

2.2. Modules Description

2.2.1. DIO (Digital Input/Output) Module

The *DIO* module is responsible for reading input signals from the system's sensors (such as buttons) and driving output signals to the system's actuators (such as *LEDs*). It provides a set of APIs to configure the direction and mode of each pin (input/output, pull-up/down resistor), read the state of an input pin, and set the state of an output pin.

2.2.2. UART Module

The *UART* Module (driver) for ATmega32 is a comprehensive solution for serial communication. It enables seamless integration of ATmega32 microcontrollers with external devices through the UART protocol. The module provides efficient and reliable data transmission, supporting both asynchronous and synchronous modes. With configurable baud rates and data formats, it accommodates various communication requirements. Its compact design and optimized code ensure minimal resource usage, allowing for efficient data transfer. Whether for interfacing with sensors, modules, or other microcontrollers, the UART Module for ATmega32 offers a robust and flexible solution for seamless serial communication.

2.2.3. LED Module

The *LED* Module (driver) for ATmega32 is a compact and versatile solution designed to control LEDs in various applications. With its support for ATmega32 microcontroller, it offers seamless integration and efficient LED management. The module provides easy-to-use functions for controlling individual LEDs, allowing for dynamic lighting effects and customization. Its compact design and optimized code ensure minimal resource utilization while delivering reliable and precise LED control.

2.2.4. BCM Module

The BCM Module (driver) for ATmega32 is a high-performance solution designed to handle diverse serial communication protocols. With its advanced capabilities, it seamlessly integrates with ATmega32 microcontroller, offering efficient and reliable data transfer. The module leverages Interrupt Service Routines (ISRs) to achieve maximum throughput, ensuring rapid and seamless communication. Its versatile design supports various serial communication protocols, making it adaptable to different applications. With meticulous hardware integration and software development.

2.2.5. CIRCULAR_QUEUE Module

The Circular Queue Module (driver) for ATmega32 is a versatile solution for efficient data storage and retrieval. It provides a circular buffer structure that maximizes memory utilization and minimizes overhead. The module supports dynamic insertion and removal of elements, making it ideal for applications with varying data sizes. With its optimized code, it ensures fast and reliable operations, enabling seamless data management. The Circular Queue Module for ATmega32 offers a flexible and efficient solution for implementing queue-based algorithms, data buffering, and event handling, making it a valuable tool for ATmega32-based systems.

2.3. Drivers' Documentation (APIs)

2.3.1 Definition

An *API* is an *Application Programming Interface* that defines a set of *routines*, *protocols* and *tools* for creating an application. An *API* defines the high level interface of the behavior and capabilities of the component and its inputs and outputs.

An *API* should be created so that it is generic and implementation independent. This allows for the API to be used in multiple applications with changes only to the implementation of the API and not the general interface or behavior.

2.3.2. MCAL APIs

2.3.2.1. DIO Driver

```
|
|  Function to set the direction of a given port
|
| This function takes an 8-bit value and sets the direction of each
| pin in the given port according to the corresponding bit value
|
| Parameters
|          [in] en_a_port    The port to set the direction of
|          [in] u8_a_portDir The desired port direction
|
|
| Return
|       en_DIO_error_t value that indicates operation success/failure
|               (DIO_OK in case of success or DIO_ERROR in case of failure)
|
en_DIO_error_t DIO_setPortDir(en_DIO_port_t en_a_port,  u8 u8_a_portDir);
|
|  Function to set the value of a given port
|
| This function takes an 8-bit value and sets the value of each
| pin in the given port according to the corresponding bit value
|
| Parameters
|          [in] en_a_port    The port to set the value of
|          [in] u8_a_portVal The desired port value
|
| Return
|       en_DIO_error_t value that indicates operation success/failure
|               (DIO_OK in case of success or DIO_ERROR in case of failure)
|
en_DIO_error_t DIO_setPortVal(en_DIO_port_t en_a_port,  u8 u8_a_portVal);
```

```
|
|  Function to set the direction of a given pin
|
| This function takes an en_DIO_pinDir_t value and sets the direction
| of the given pin accordingly
|
| Parameters
|          [in] en_a_port   The port of the desired pin
|          [in] en_a_pin    The desired pin to set direction of
|          [in] en_a_pinDir The desired pin direction (INPUT/OUTPUT)
|
|
| Return
|       en_DIO_error_t value that indicates operation success/failure
|               (DIO_OK in case of success or DIO_ERROR in case of failure)
|
en_DIO_error_t DIO_setPinDir (en_DIO_port_t en_a_port, en_DIO_pin_t en_a_pin,
en_DIO_pinDir_t en_a_pinDir);

|
|  Function to set the value of a given pin
|
| This function takes an en_DIO_level_t value and sets the value
| of the given pin accordingly
|
| Parameters
|          [in] en_a_port   The port of the desired pin
|          [in] en_a_pin    The desired pin to set value of
|          [in] en_a_pinDir The desired pin value (HIGH/LOW)
|
|
| Return
|       en_DIO_error_t value that indicates operation success/failure
|               (DIO_OK in case of success or DIO_ERROR in case of failure)
|
en_DIO_error_t DIO_setPinVal (en_DIO_port_t en_a_port, en_DIO_pin_t en_a_pin,
en_DIO_level_t en_a_pinVal);
```

```
|
|  Function to toggle the value of a given pin
|
| If the pin value is high, this function sets it to low
| and if it is low it sets it to high
|
| Parameters
|          [in] en_a_port    The port of the desired pin
|          [in] en_a_pin     The desired pin to toggle value of
|
|
| Return
|       en_DIO_error_t value that indicates operation success/failure
|               (DIO_OK in case of success or DIO_ERROR in case of failure)
|
en_DIO_error_t DIO_togPinVal (en_DIO_port_t en_a_port, en_DIO_pin_t en_a_pin);

|
|  Function to get the value of a given pin
|
| This function reads the value of the given pin and
| returns the value in the given address
|
| Parameters
|          [in] en_a_port    The port of the desired pin
|          [in] en_a_pin     The desired pin to read value of
|          [out] pu8_a_Val   address to return the pin value into
|
|
| Return
|       en_DIO_error_t value that indicates operation success/failure
|               (DIO_OK in case of success or DIO_ERROR in case of failure)
|
en_DIO_error_t DIO_getPinVal (en_DIO_port_t en_a_port, en_DIO_pin_t en_a_pin,
u8* pu8_a_Val);
```

2.3.2.2. UART Driver

```
| Initializes UART Module with a specific given config
| Parameters
|           [in]ptr_uart_config pointer to UART configuration
|
| Return
|       [enum] enu_uart_error
|
enu_uart_error_t_ uart_init(const str_uart_config_t_ * ptr_uart_config);


| Sends a byte of data
| Parameters
|           [in]uint8_data data byte to be sent
|
| Return
|       [enum] enu_uart_error
|
enu_uart_error_t_ uart_send(uint8_t_ uint8_data);



| Sends multiple bytes up to 65,535 or queue MAX over UART
| Parameters
|           [in]str_send_queue send queue
|           [in]uint16_dataLength data length to send
|
| Return
|       [enum] enu_uart_error
|
enu_uart_error_t_ uart_send_n(str_circularqueue_t_ ** str_send_queue,
uint16_t_ uint16_dataLength);



| Receives multiple bytes up to 65,535 or queue MAX over UART
| Parameters
|           str_circularqueue_queue receive queue
|           uint16_received_data_length received data length
| Return
|       [enum] enu_uart_error
|
enu_uart_error_t_ uart_receive(str_circularqueue_t_ **
str_circularqueue_queue, uint16_t_ * uint16_received_data_length);
```

```
| Sets callback ptr to notify events on
| Parameters
|           ptr_callback ptr to notify function
|
| Return
|       [enum] enu_uart_error
|
enu_uart_error_t_ uart_setCallback(void (* ptr_callback)(uint8_t_
uint8_instance_id, str_operation_info_t_ str_operation_info));

|
| Handles flow of events for UART
|
| Return
|       [enum] enu_uart_error
|
enu_uart_error_t_ uart_dispatcher(void);
```

2.3.3. HAL APIs

2.3.3.1. LED APIs

```
| Initializes LED on given port & pin
|
| Parameters
|           ledPort [in] LED Port
|           ledPin  [in] LED Pin number
|
EN_LED_ERROR_t LED_init(EN_DIO_PORT_T ledPort, uint8_t ledPin);


| Turns on LED at given port/pin
|
| Parameters
|            ledPort [in] LED Port
|            ledPin  [in] LED Pin number
|
EN_LED_ERROR_t LED_on(EN_DIO_PORT_T ledPort, uint8_t ledPin);


| Turns on LED at given port/pin
|
| Parameters
|            ledPort [in] LED Port
|            ledPin [in] LED Pin number in ledPort
|
EN_LED_ERROR_t LED_off(EN_DIO_PORT_T ledPort, uint8_t ledPin);

| Toggles LED at given port/pin
|
| Parameters
|            ledPort [in] LED Port
|            ledPin [in] LED Pin number in ledPort
|
EN_LED_ERROR_t LED_toggle(EN_DIO_PORT_T ledPort, uint8_t ledPin);
```

2.3.4. SERV APIs

2.3.4.1. BCM APIs

```
| Initializes BCM Instance
| Parameters
|           ptr_str_bcm_instance BCM Instance
|
| Return
|       enu_system_status
|
enu_system_status_t bcm_init(const str_bcm_instance_t * ptr_str_bcm_instance);

| De-initializes BCM Instance
| Parameters
|           ptr_str_bcm_instance BCM Instance
|
| Return
|       enu_system_status
|
enu_system_status_t bcm_deInit(const str_bcm_instance_t *
ptr_str_bcm_instance);

| Sends one byte over a certain BCM instance
| Parameters
|           [in]ptr_str_bcm_instance BCM instance
|           [in]uint8_dataByte byte to send
|
| Return
|       [enum] enu_system_status_t
|
enu_system_status_t bcm_send(const str_bcm_instance_t * ptr_str_bcm_instance,
uint8_t_ uint8_dataByte);

| Sends multiple bytes over a certain BCM instance
| Parameters
|           [in]ptr_str_bcm_instance BCM instance address
|           [in]ptr_str_send_queue ptr to queue holding data to be sent
|           [in]uint16_dataLength data length to send
|
| Return
|       [enum] enu_system_status
|
enu_system_status_t bcm_send_n(const str_bcm_instance_t *
ptr_str_bcm_instance, str_circularqueue_t_ ** str_send_queue, uint16_t_
uint16_dataLength);
```

```
| Used to receive data from a certain BCM instance
| Parameters
|           [in]ptr_str_bcm_instance BCM instance address
|           [out]str_rec_queue receiving queue
|           [out]uint16_received_data_length data length to read
| Return
|       [enum] enu_system_status_t
|
enu_system_status_t bcm_receive(const str_bcm_instance_t *
ptr_str_bcm_instance, str_circularqueue_t_ ** str_rec_queue, uint16_t_ *
uint16_received_data_length);

| Handles flow of events for a certain bcm instance
| Parameters
|           str_bcm_instance
|
| Return
|       [enum] enu_system_status_t
|
enu_system_status_t bcm_dispatcher(const str_bcm_instance_t *
str_bcm_instance);

| Sets callback pointer to which BCM will route events that BCM finds
|  important to be processed outside
|
enu_system_status_t bcm_setCallback(void (*ptr_callback)(uint8_t_
uint8_instance_id, str_operation_info_t_ str_operation_info));

/** BCM Instances Available To Use */
extern const str_bcm_instance_t gl_cst_str_data_bus;
```

2.3.4. APP APIs

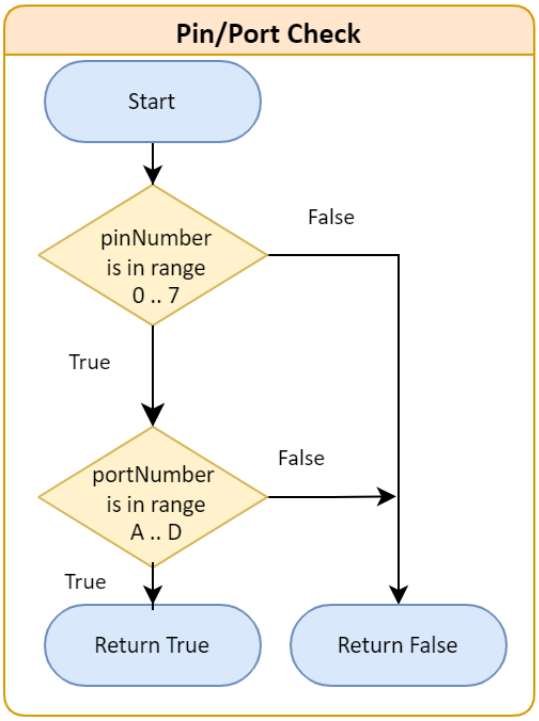
```
| Initializes Application
void app_initialize();

| Starts Application Super Loop
void app_start();
```

3. Low Level Design

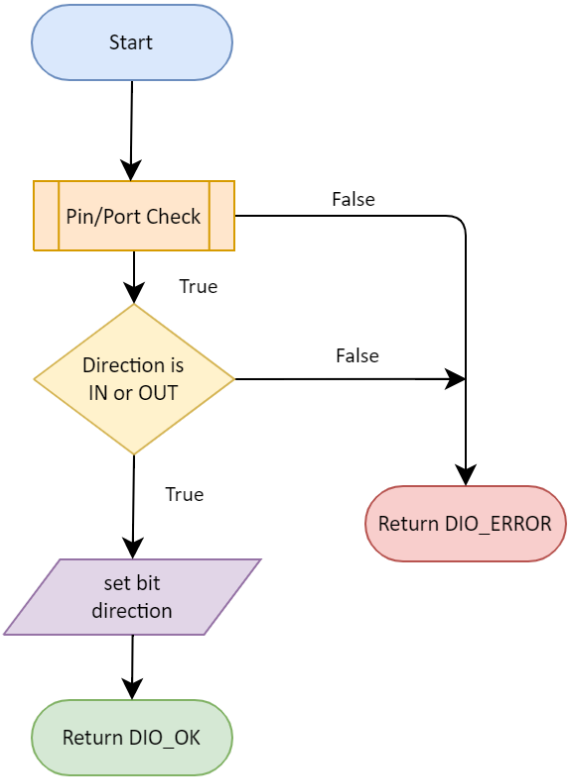
3.1. MCAL Layer

3.1.1. DIO Module

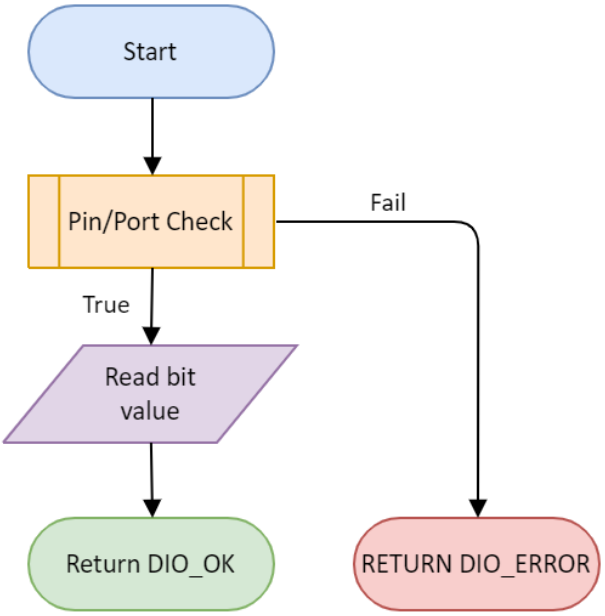
3.1.1.a. Pin/Port Check sub process



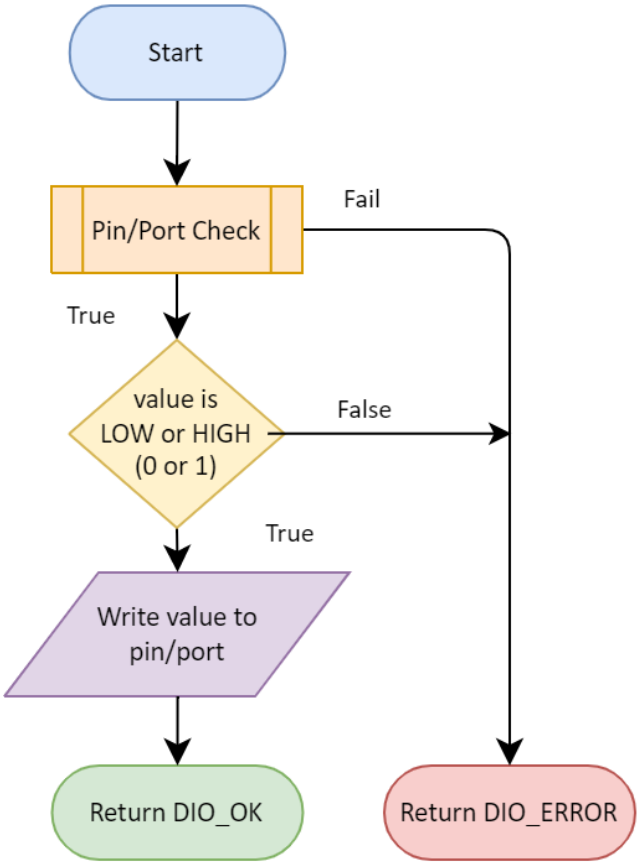
3.1.1.1. DIO_init



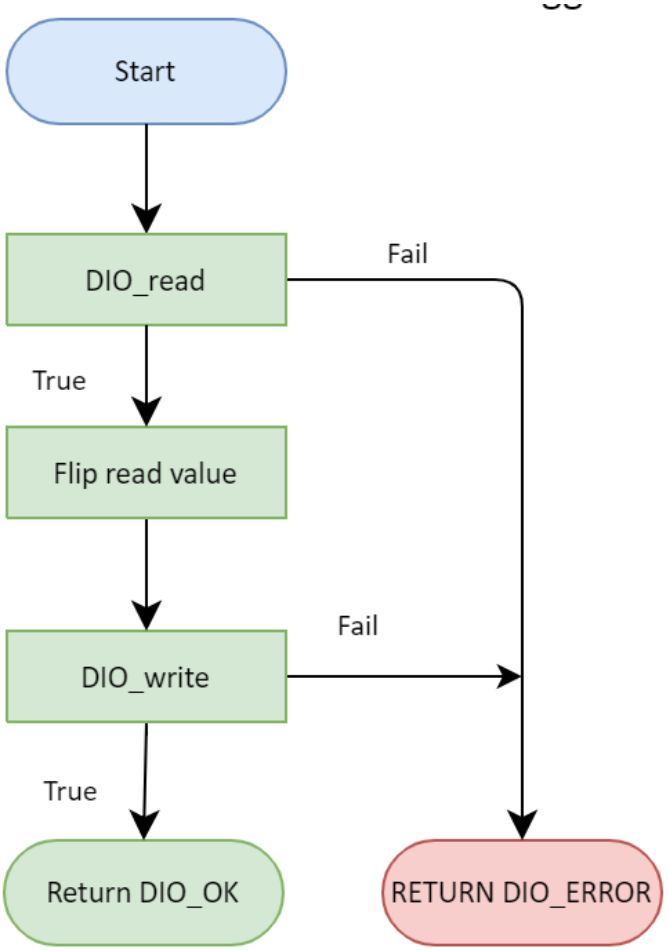
3.1.1.2. DIO_read



3.1.1.3. DIO_write



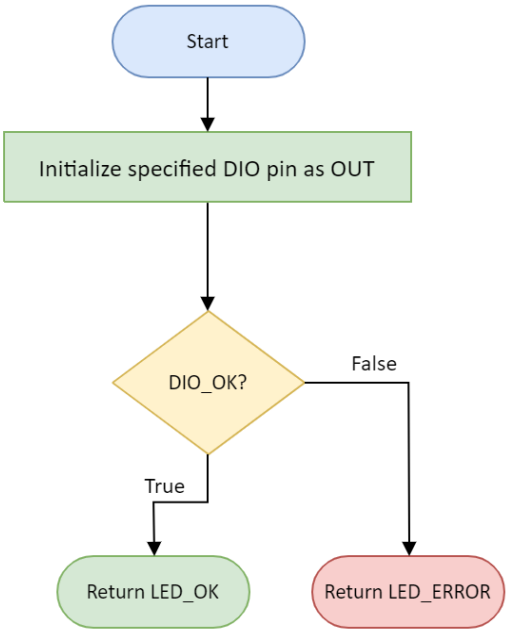
3.1.1.4. DIO_toggle



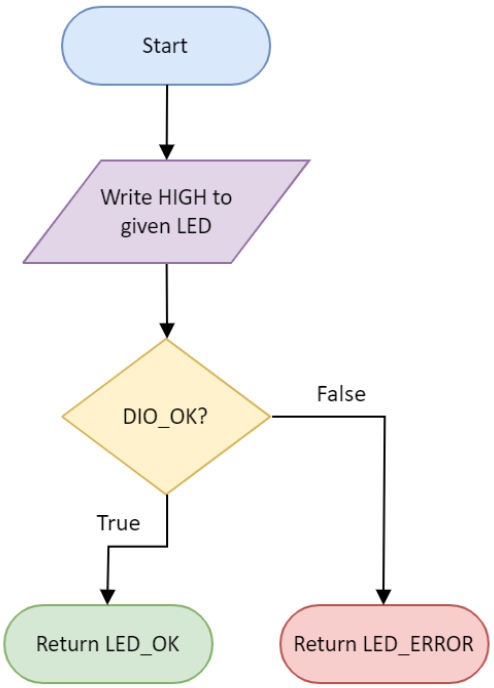
3.2. HAL Layer

3.2.1. LED Module

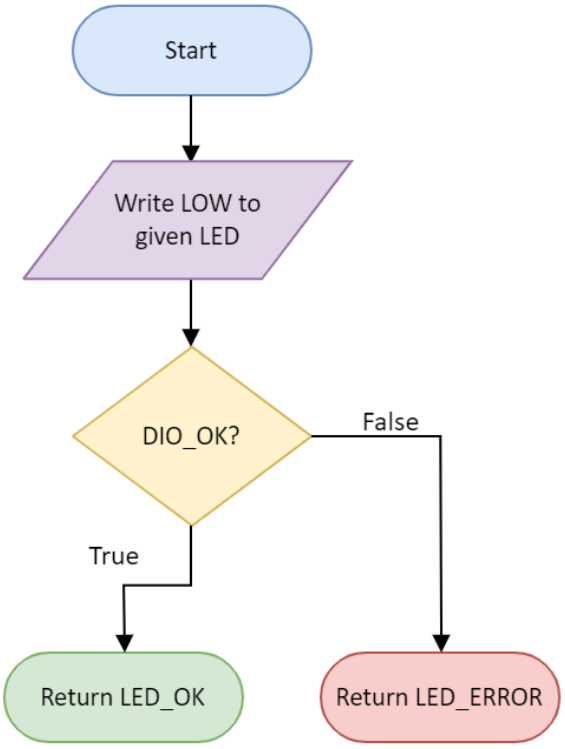
3.2.1.1. LED_init



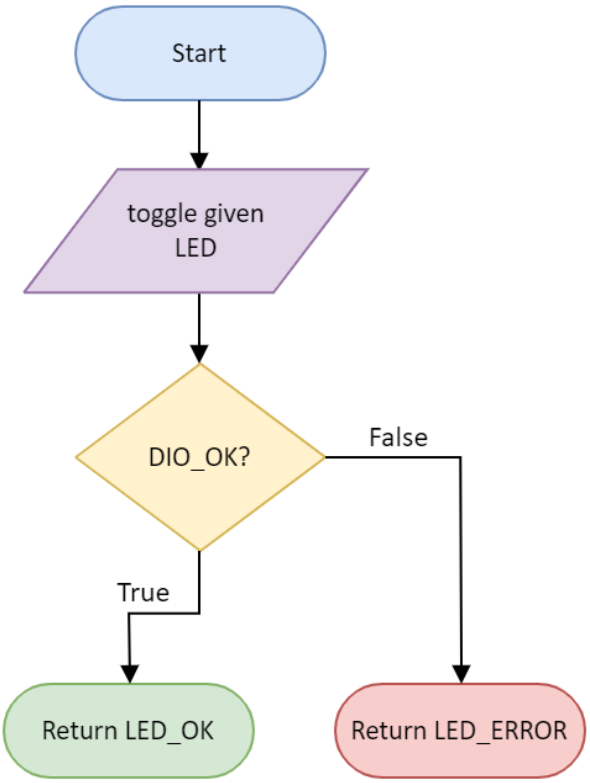
3.2.1.2. LED_on



3.2.1.3. LED_off



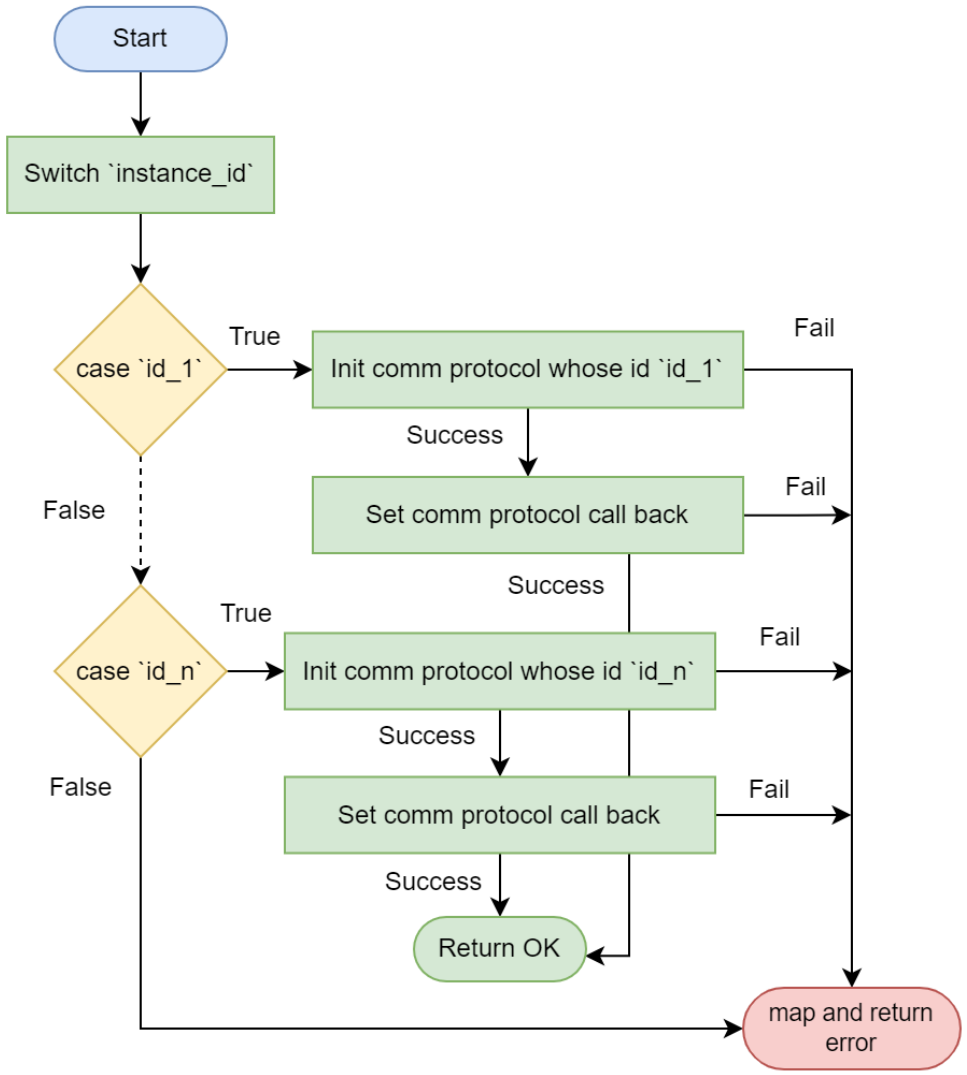
3.2.1.4. LED_toggle



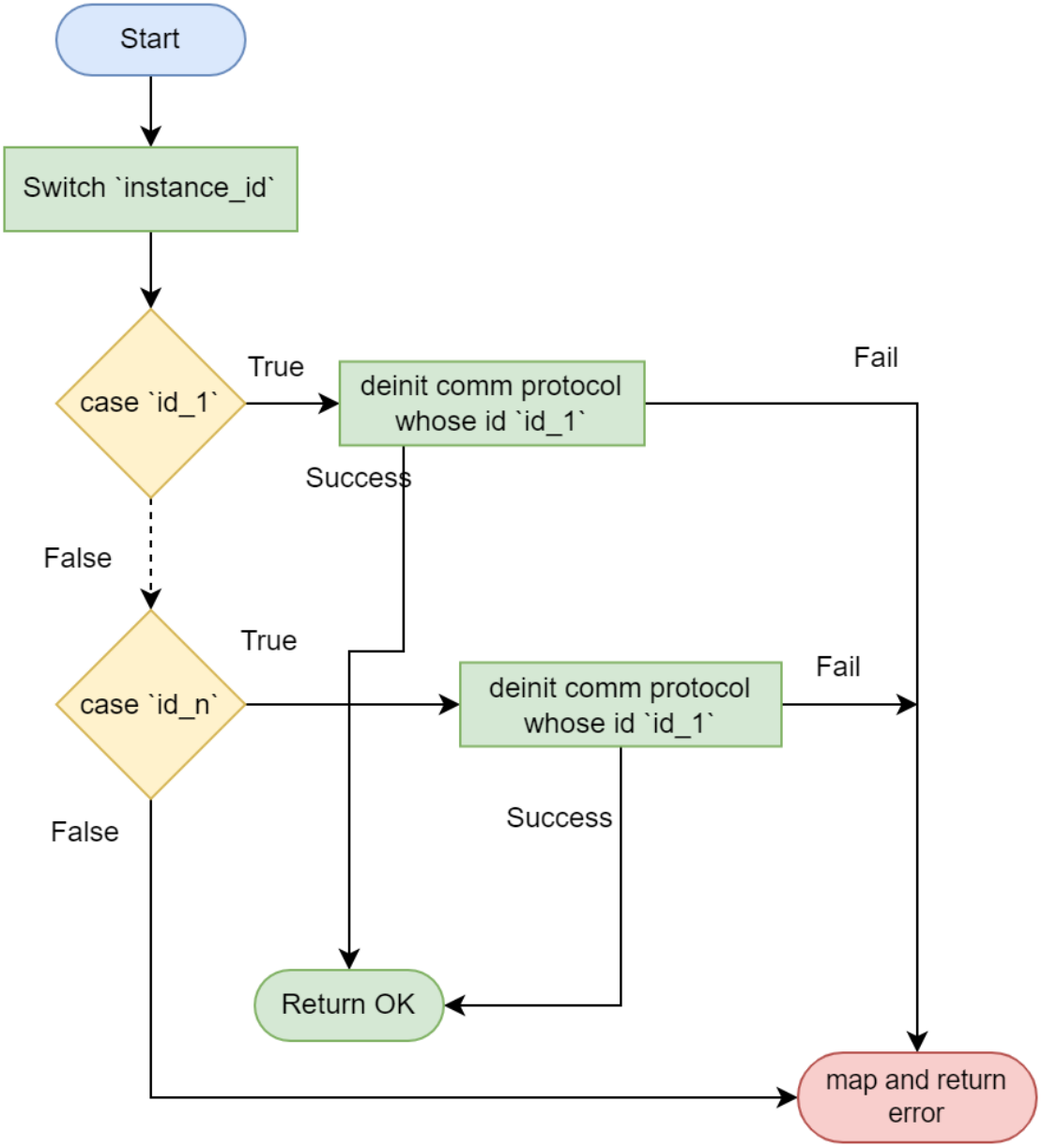
3.3. SERV Layer

3.3.1. BCM Module

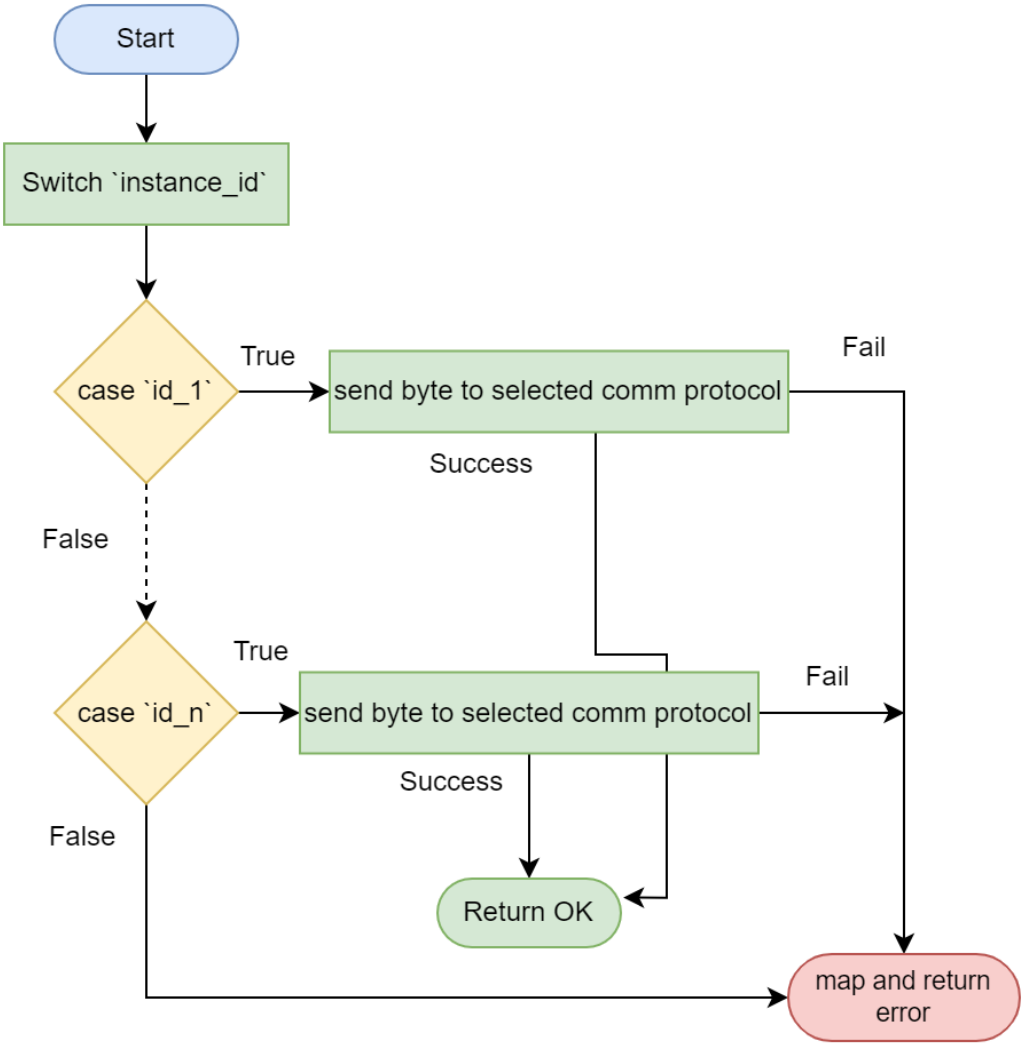
3.3.1.1. bcm_init



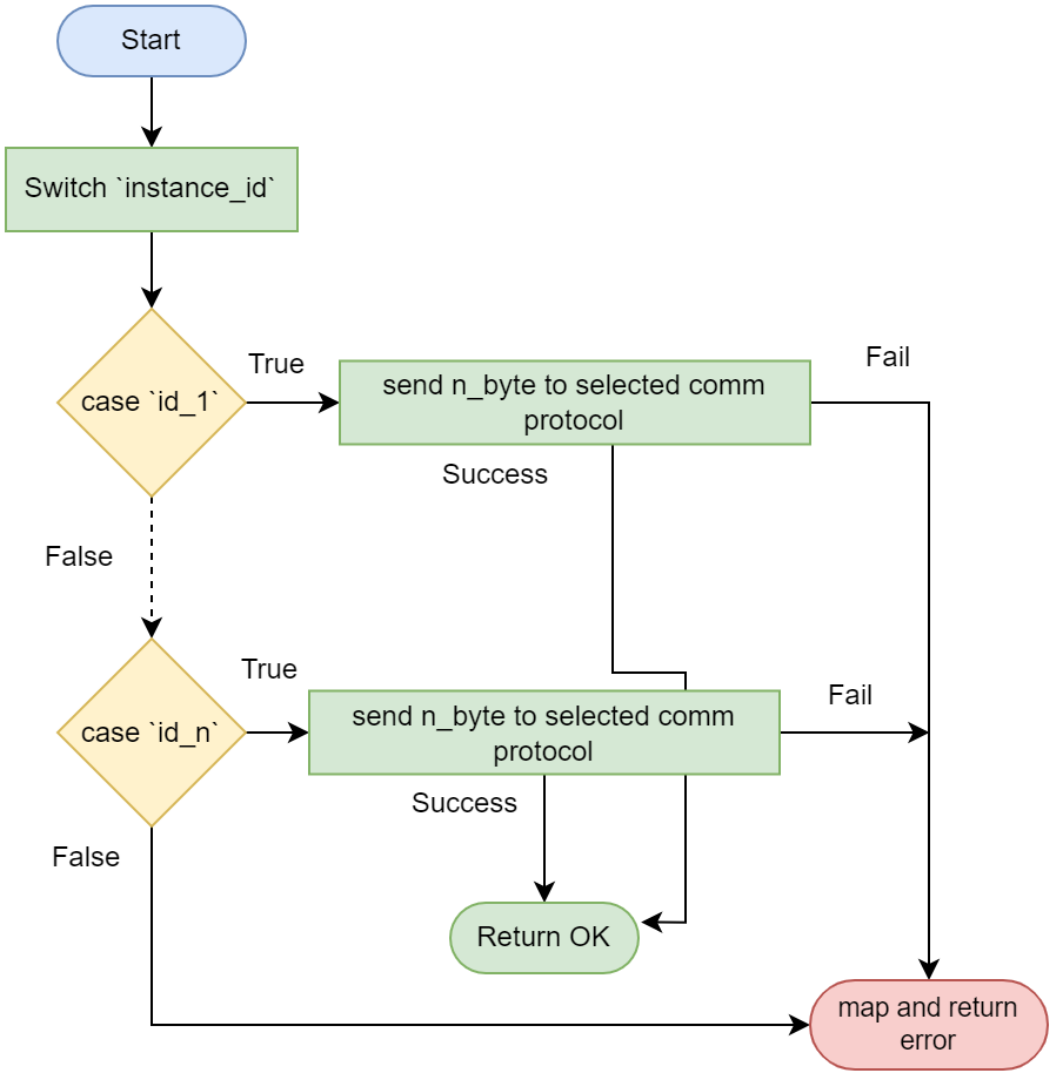
3.3.1.2. bcm_deinit



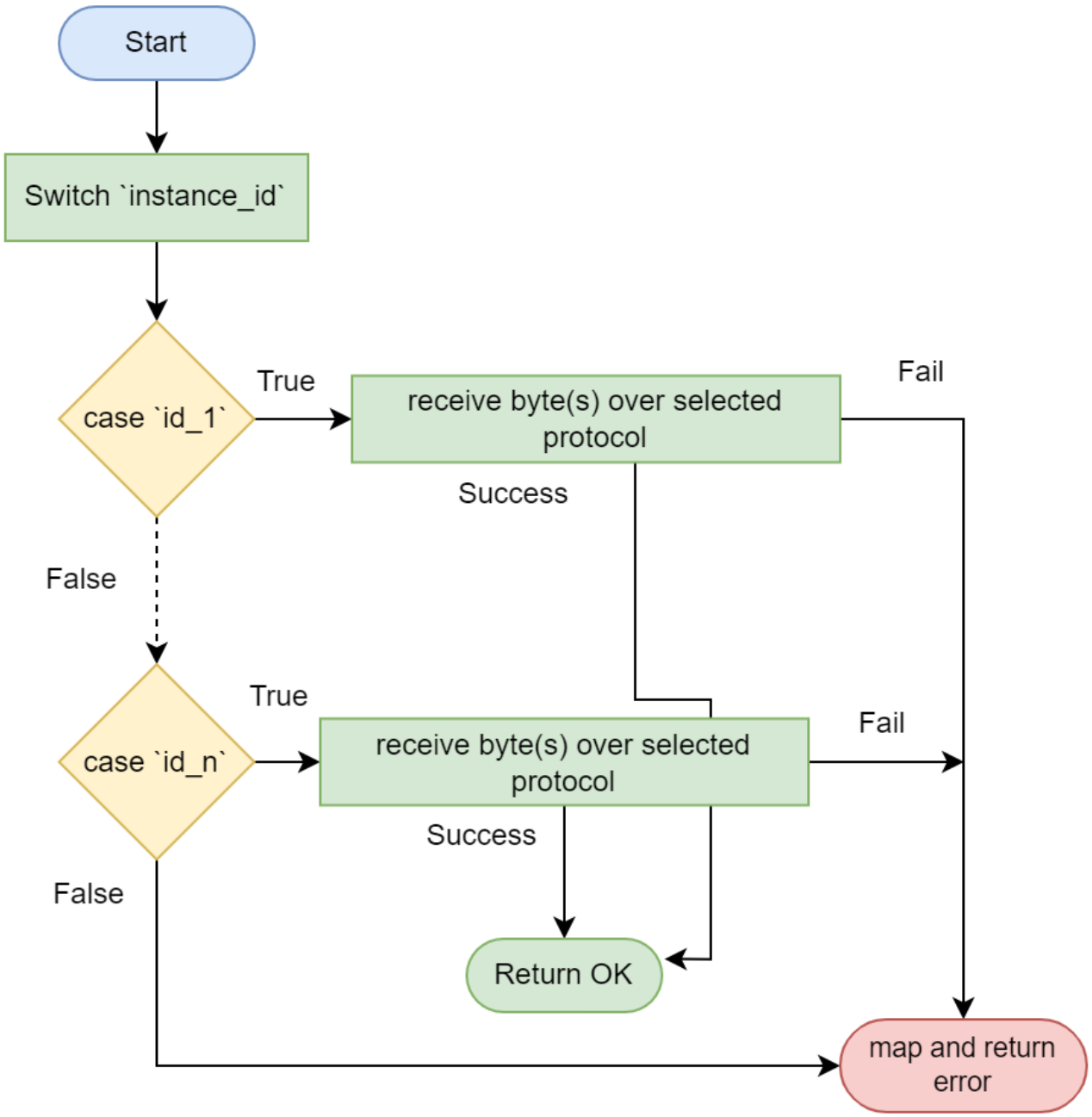
3.3.1.3. bcm_send



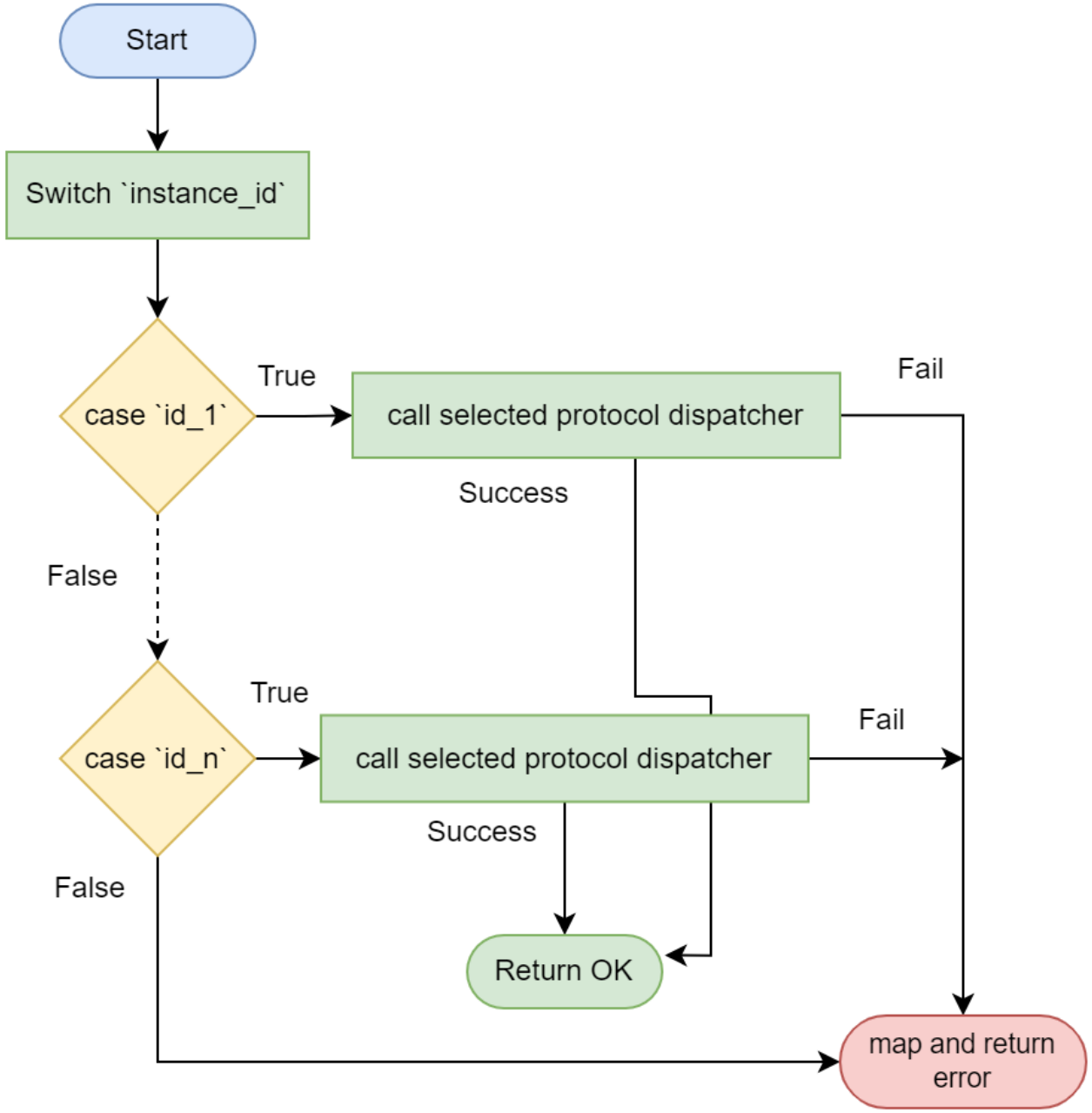
3.3.1.4. bcm_send_n



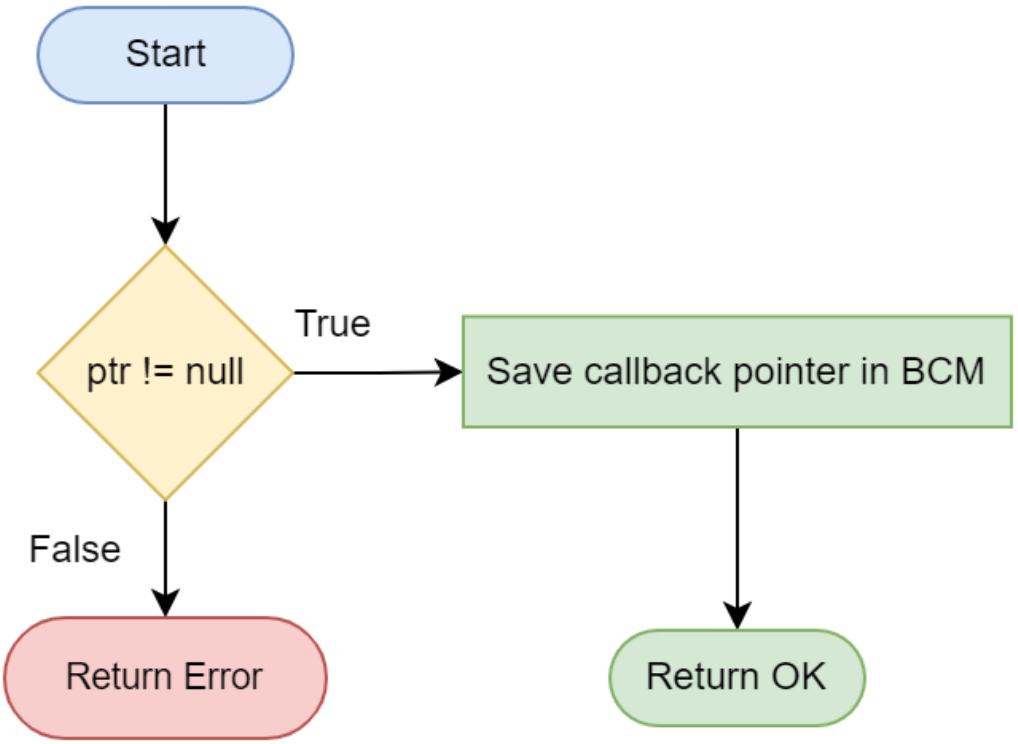
3.3.1.5. bcm_receive



3.3.1.6. bcm_dispatcher



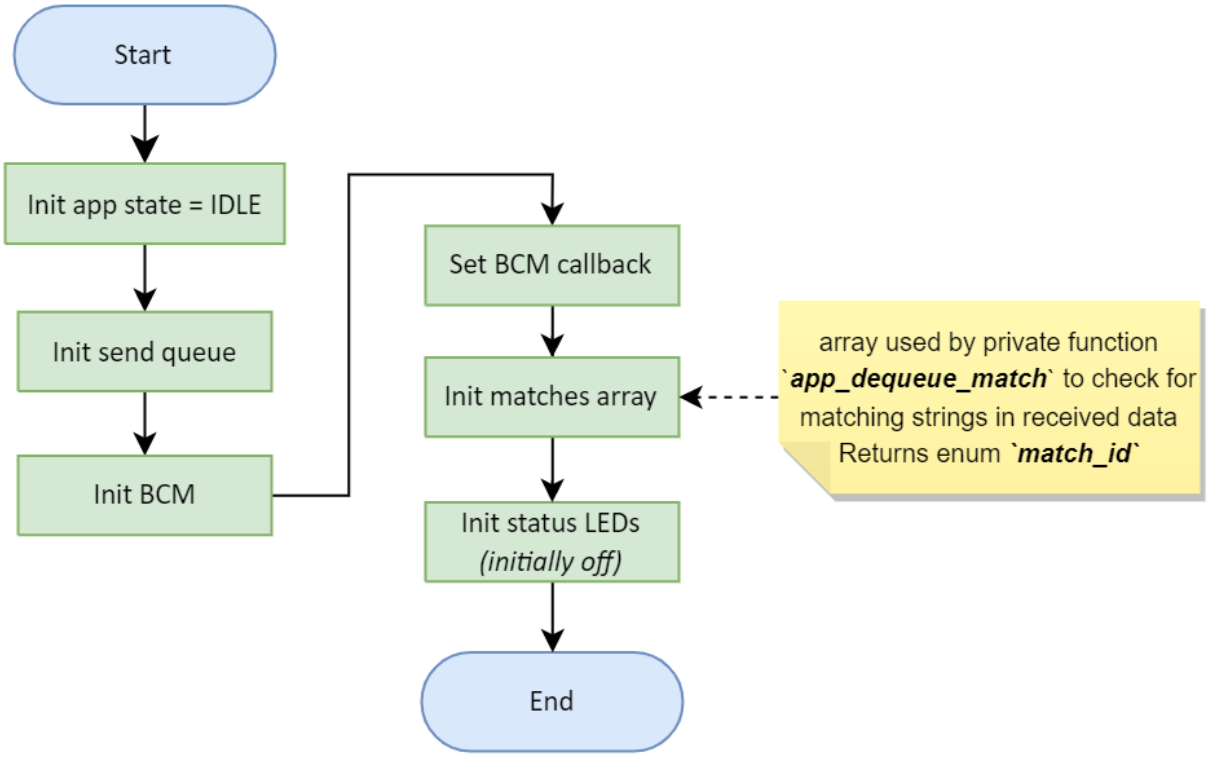
3.3.1.7. bcm_setCallback



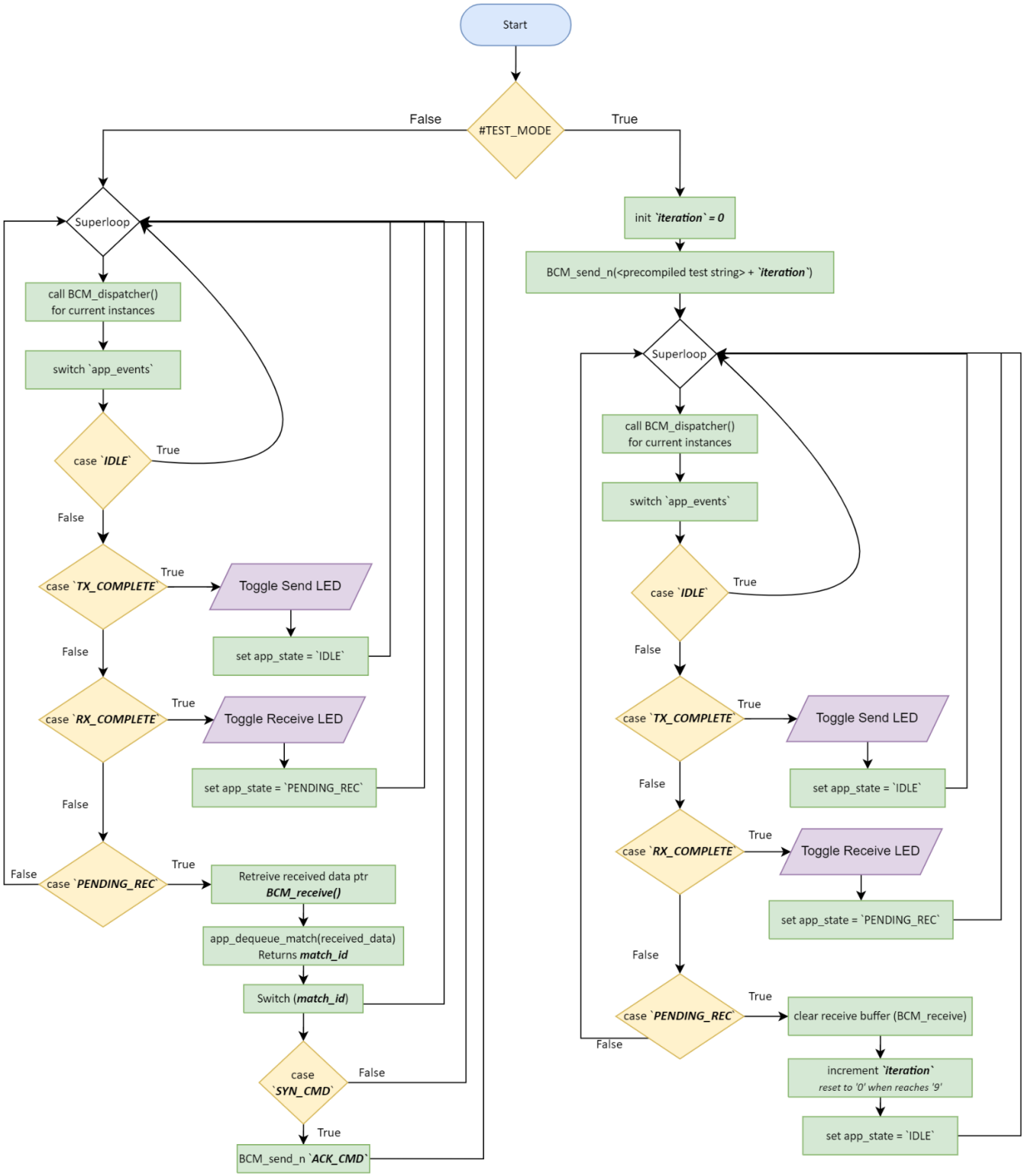
3.4. APP Layer

3.4.1. MCU1

3.4.1.1. app_initialize

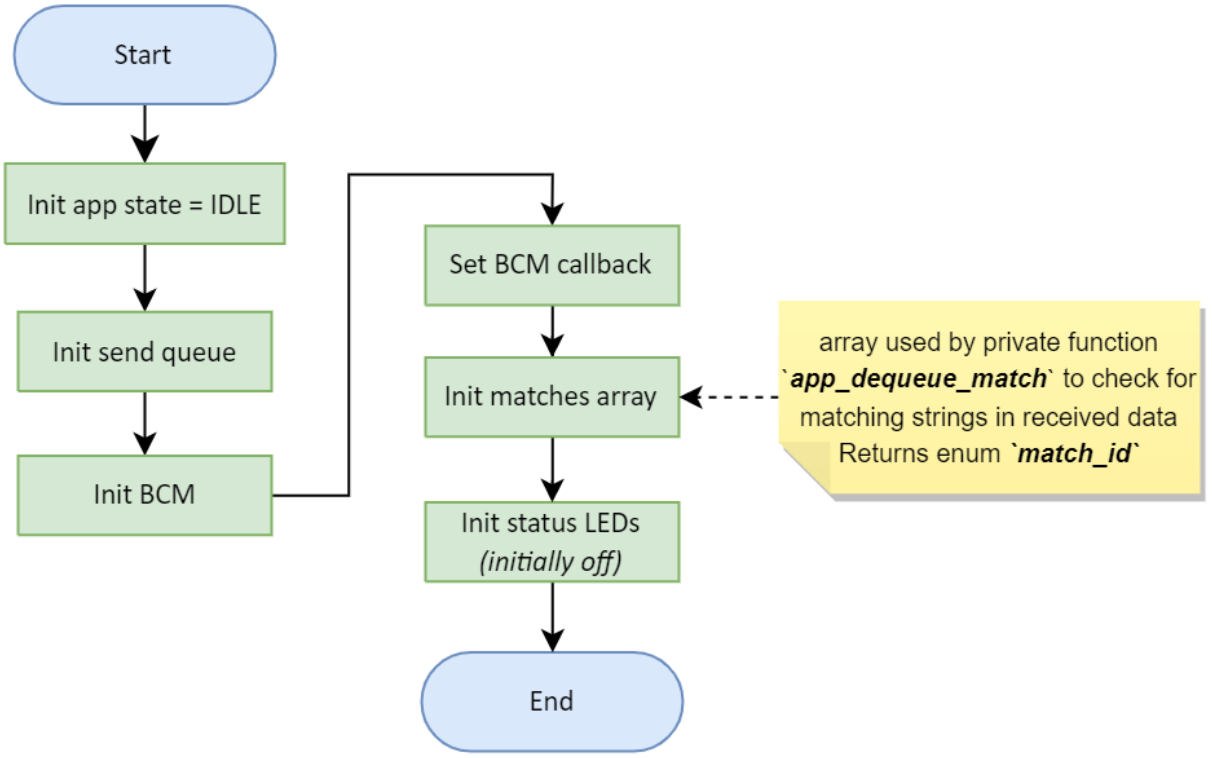


3.4.1.2. app_start (click for HQ)

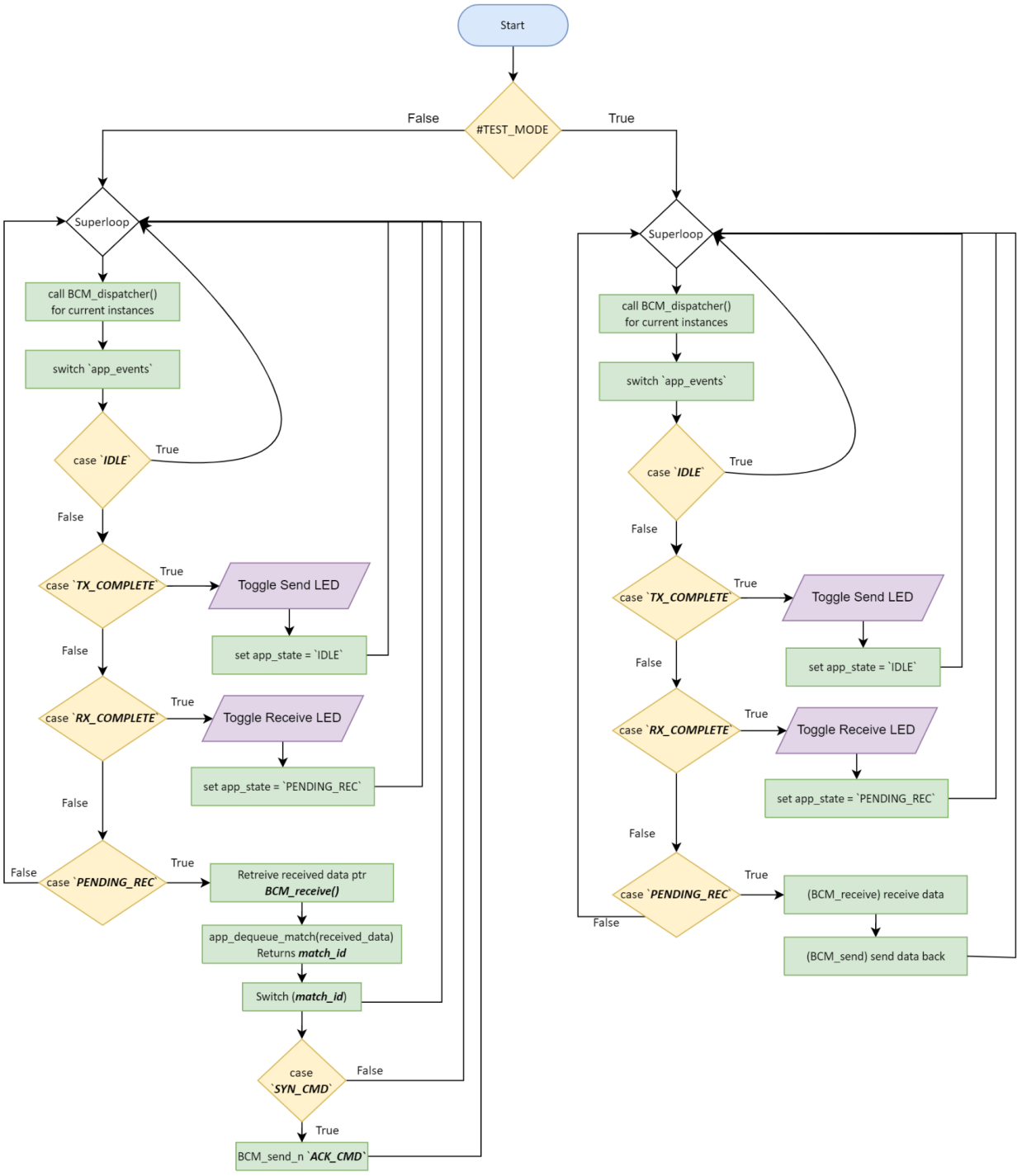


3.4.2. MCU2

3.4.2.1. app_initialize

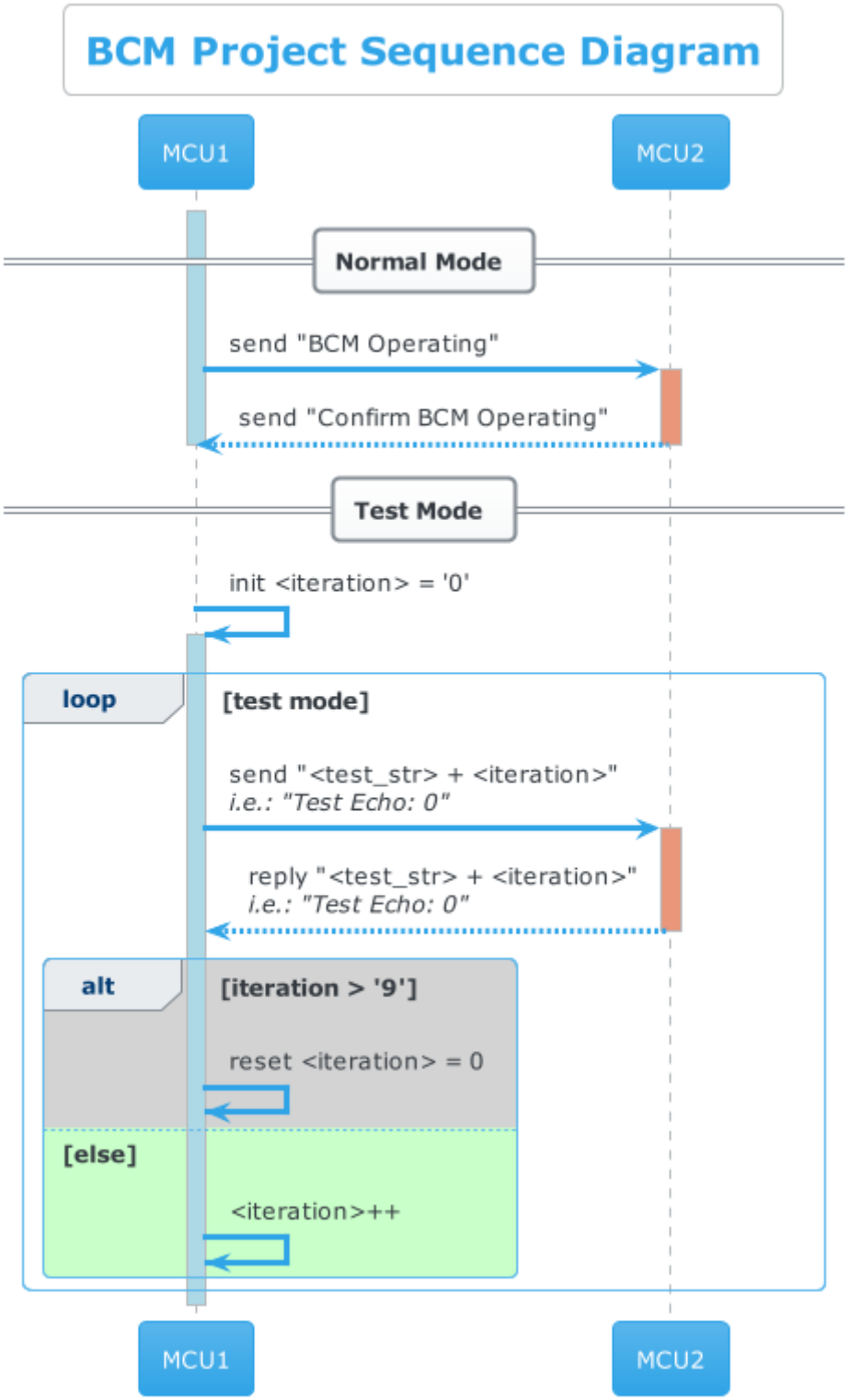


3.4.2.2. app_start (click for HQ)



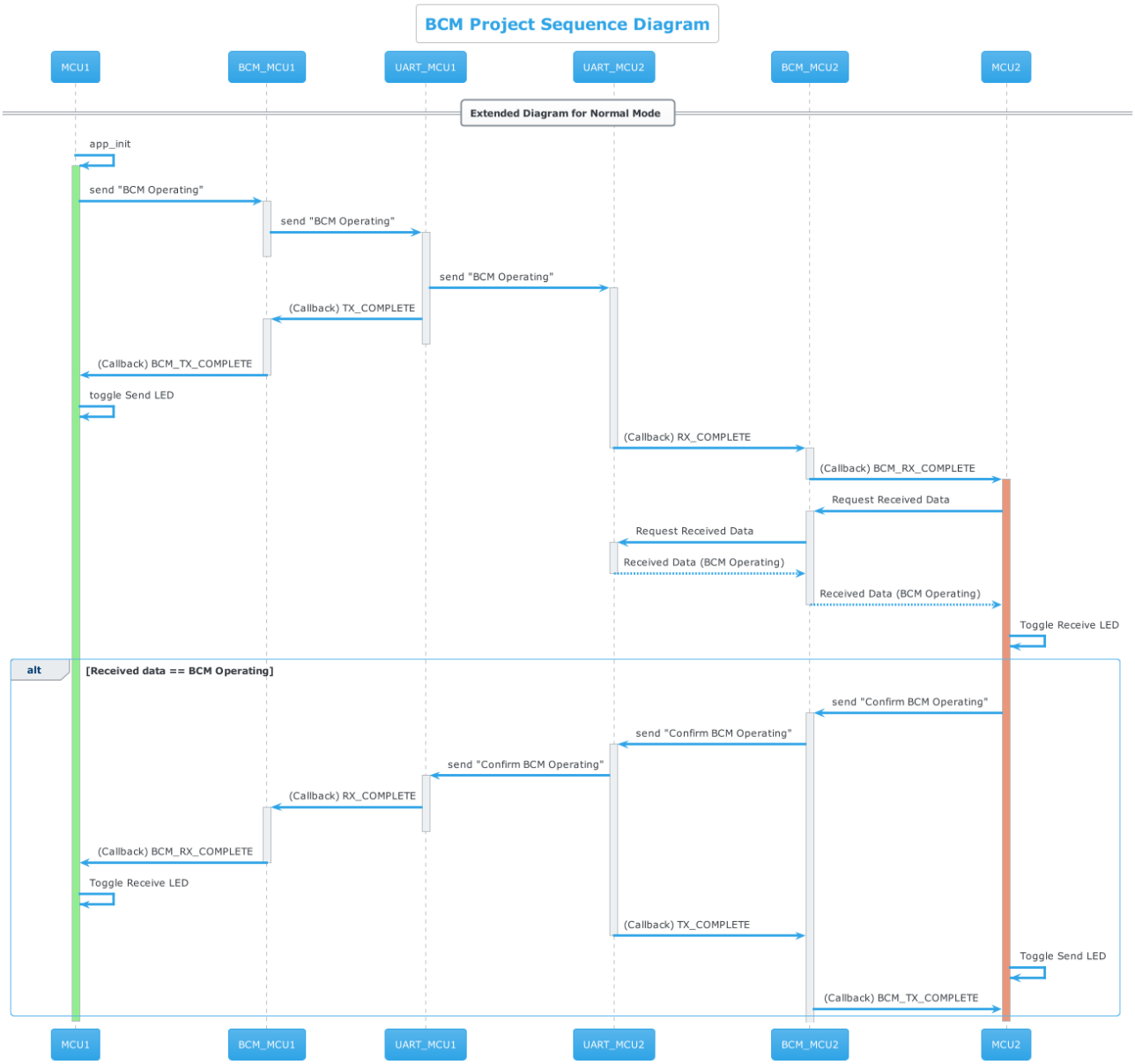
4. Sequence Diagrams

4.1. Brief Sequence Diagram



4.2. Detailed Sequence Diagram

(Click for HQ)



5. Configurations

5.1. UART Driver

5.1.1. Preconfiguration

```
#ifndef UART_PRECONFIG_H_
#define UART_PRECONFIG_H_

#ifndef F_OSC
    // System frequency Oscillator in Hz
    #define F_OSC 8000000UL // 8 MHz
#endif


/* TURN OFF FOR PRODUCTION USE */
//#define UART_DEBUG 1
#define UART_DEBUG 0

#define UART_EOT_CHAR '\0'

#define UART_CARRIAGE_RETURN_CHAR 0x13  // Enter Key
#define UART_NEW_LINE_CHAR 0x0D         // \r

#define UART_TX_RETRY_COUNT 3 // Retries to transmit before failing

#endif /* UART_PRECONFIG_H_ */
```

5.1.2. Linking Configuration

```
const str_uart_config_t_ glcststr_uart_config_ch_0 = {
        .enu_uart_mode = TRANSCEIVER,
        .enu_uart_clock_mode = ASYNC,
        .uni_uart_mode_options.str_async_options.enu_uart_baud_rate = BAUD_RATE_9600,
        .enu_uart_parity_mode = NO_PARITY,
        .enu_uart_speed_mode = SPEED_NORMAL,
        .enu_uart_data_length = DATA_8_BITS,
        .enu_uart_stop_bits_count = ONE_STOP_BIT,
        .enu_uart_operating_mode = UART_ASYNC,
};

extern const str_uart_config_t_ glcststr_uart_config_ch_0;
```

5.2. BCM Driver - Linking Configuration

```
/****************** BCM LINKING CONFIGURATION ******************/

const str_bcm_instance_t gl_cst_str_data_bus = {
        .uint8_instance_id = BCM_PROTOCOL_UART,
        {.uartConfig =  &glcststr_uart_config_ch_0 }
};
```

5.3. Circular Queue (UTILITY) - Preconfiguration

```
#ifndef CQUEUE_PRECONFIG_H_
#define CQUEUE_PRECONFIG_H_

#define QUEUE_SIZE 500

#endif /* CQUEUE_PRECONFIG_H_ */
```

5.4. App preconfiguration

```
#define SEND_LED_PORT   DIO_PORT_B
#define SEND_LED_PIN    DIO_PIN_1

#define RECEIVE_LED_PORT   DIO_PORT_B
#define RECEIVE_LED_PIN    DIO_PIN_2

#define TEST_STR "\r\rLorem ipsum dolor sit amet, test echo (0-9): "

/* ECHO MODE */
//#define APP_TEST_ENABLE 1
#define APP_TEST_ENABLE 0

/* Size(count) of commands to cross-match on receive */
#define MATCH_SIZE 2

/* Commands */
#define SYN_STR "BCM Operating"
#define ACK_STR "Confirm BCM Operating"
```