

Contents

[1. Project Introduction 3](#_Toc136044258)

[1.1. Project Components 3](#_Toc136044259)

[2. High Level Design 3](#_Toc136044260)

[2.1. System Architecture 3](#_Toc136044261)

[2.1.1. Layered Architecture 3](#_Toc136044262)

[2.1.2. Design 4](#_Toc136044263)

[2.2. Sequence Diagram 5](#_Toc136044264)

[2.3. State Machine Diagram 8](#_Toc136044265)

[2.4. BCM Data Buffer 9](#_Toc136044266)

[2.5. Modules Description 9](#_Toc136044267)

[2.5.1. DIO (Digital Input/Output) Module 9](#_Toc136044268)

[2.5.2. UART 9](#_Toc136044269)

[2.5.3. LED 9](#_Toc136044270)

[2.5.4. BCM 9](#_Toc136044271)

[2.6. Drivers’ Documentation 10](#_Toc136044272)

[2.6.1. MCAL Drivers’ Functions 10](#_Toc136044273)

[2.6.2. HAL Drivers’ Functions 14](#_Toc136044274)

[2.6.3. SERVER Drivers’ functions 16](#_Toc136044275)

[Low Level Design 21](#_Toc136044276)

[2.7. MCAL Layer 21](#_Toc136044277)

[2.7.1. DIO Module 21](#_Toc136044278)

[2.7.2. UART 26](#_Toc136044279)

[2.8. Server Layer 29](#_Toc136044280)

[2.8.1. BCM 29](#_Toc136044281)

# Project Introduction

This project is aiming to deliver a BCM -Basic Communication Manager- which will manage the data which the user wants to transmit/receive to make a specific task. I implemented the project to send data using UART protocol which is up to 512 byte and it can be from 0 to 255

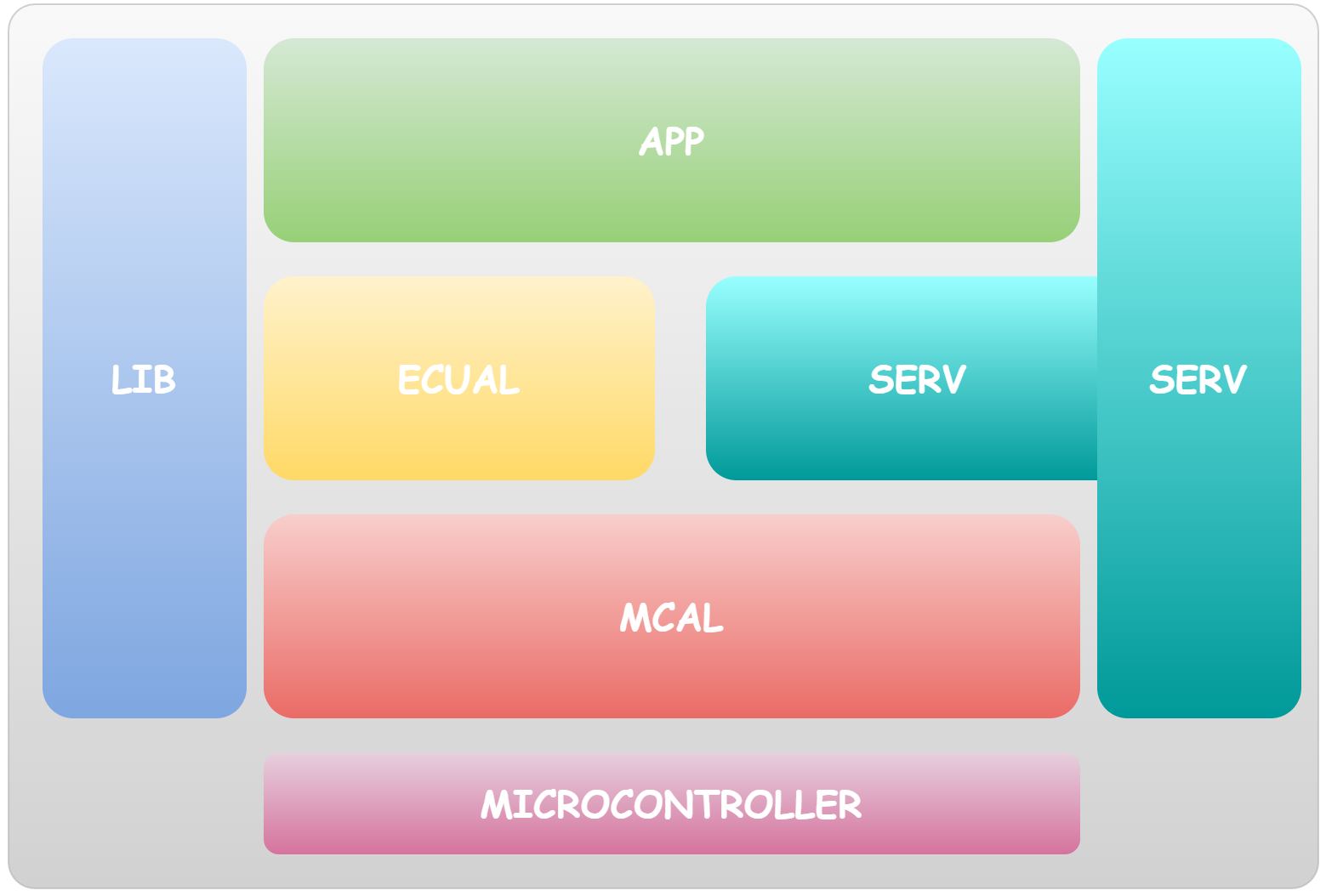
## Project Components

* 2 ATmega32 microcontroller
* 2 LEDS for each microcontroller

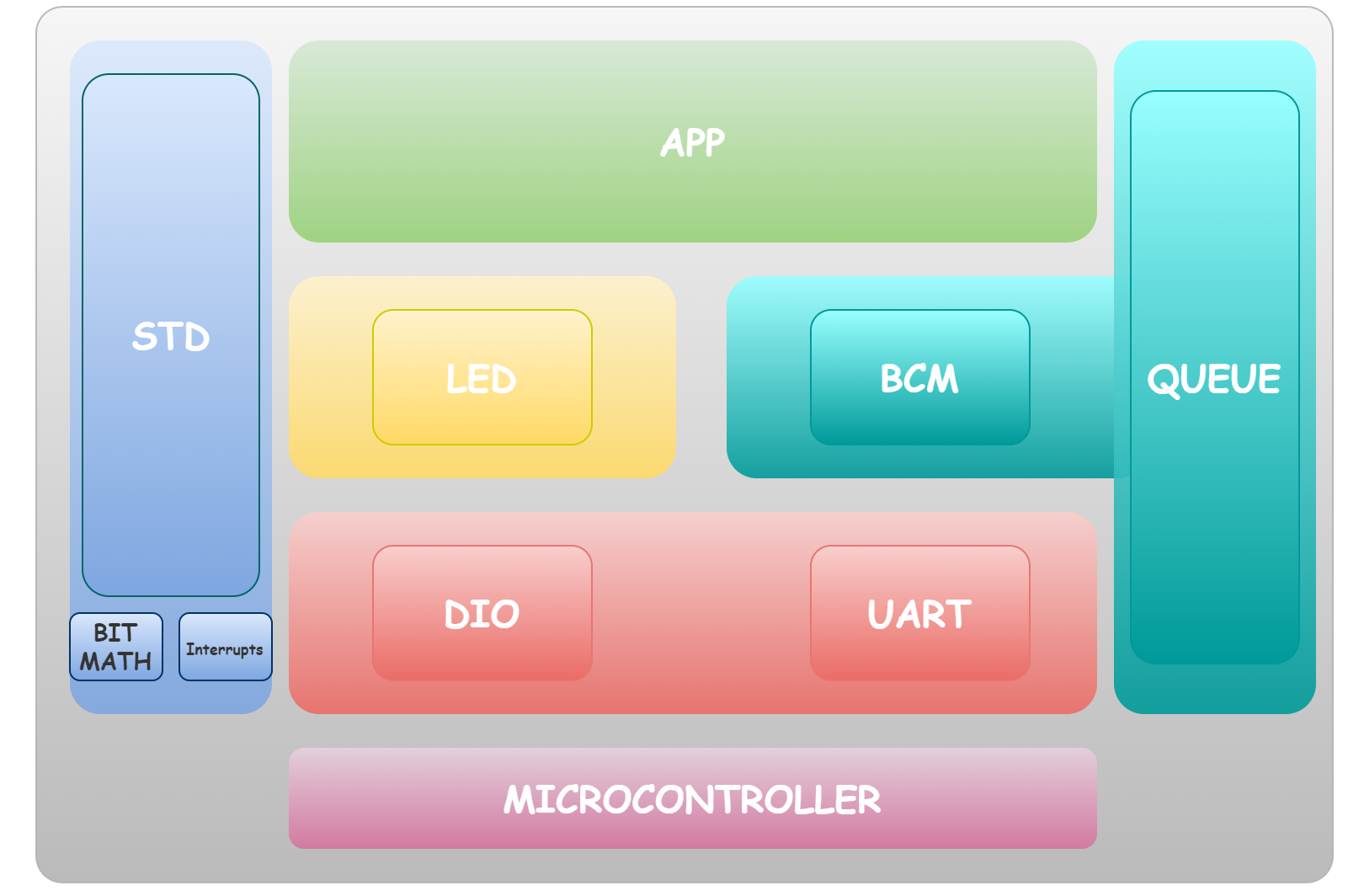
# High Level Design

## System Architecture

### Layered Architecture

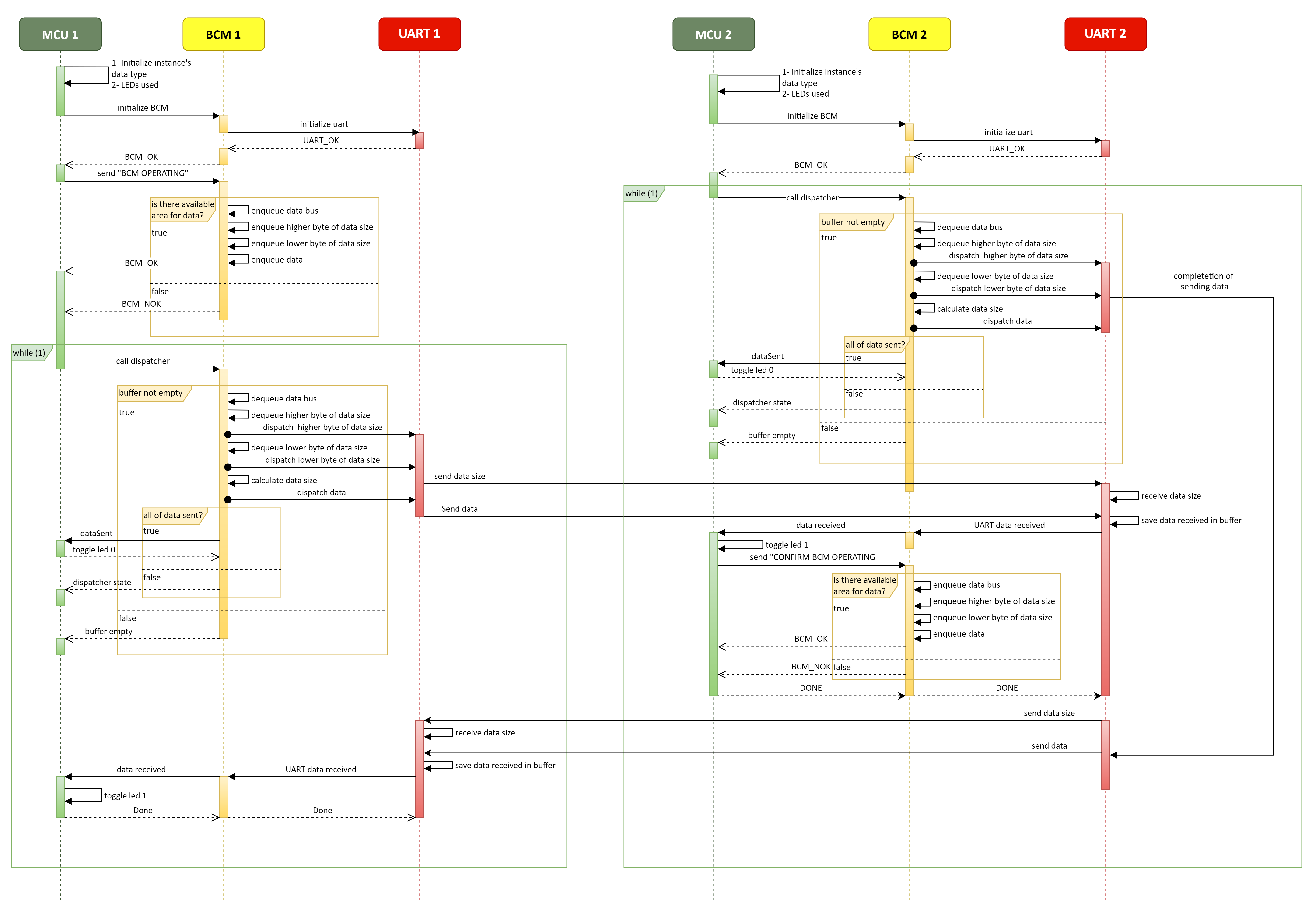


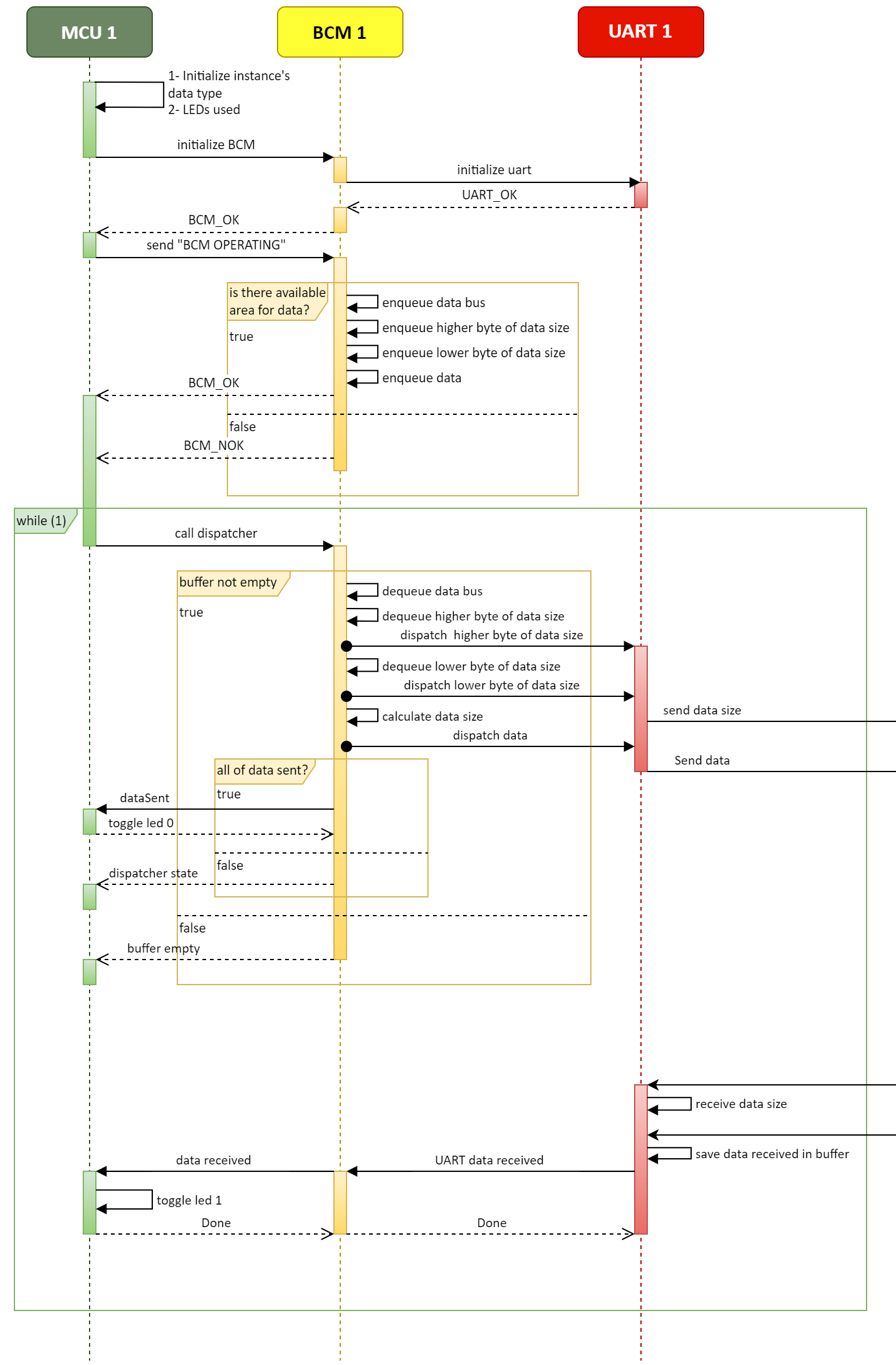
### Design



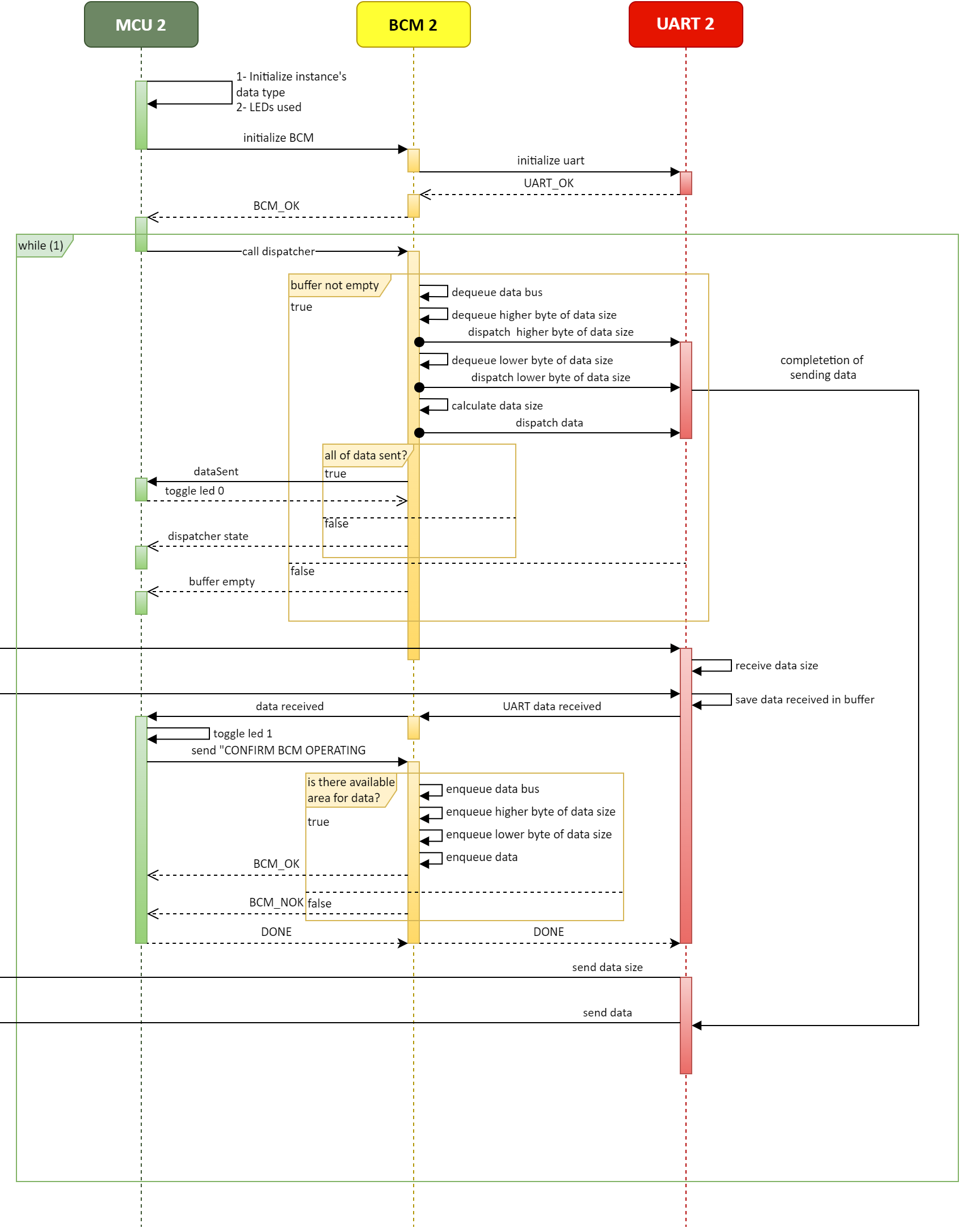
## Sequence Diagram

This whole of the sequence, you can find a zoomed parts in the next pages.



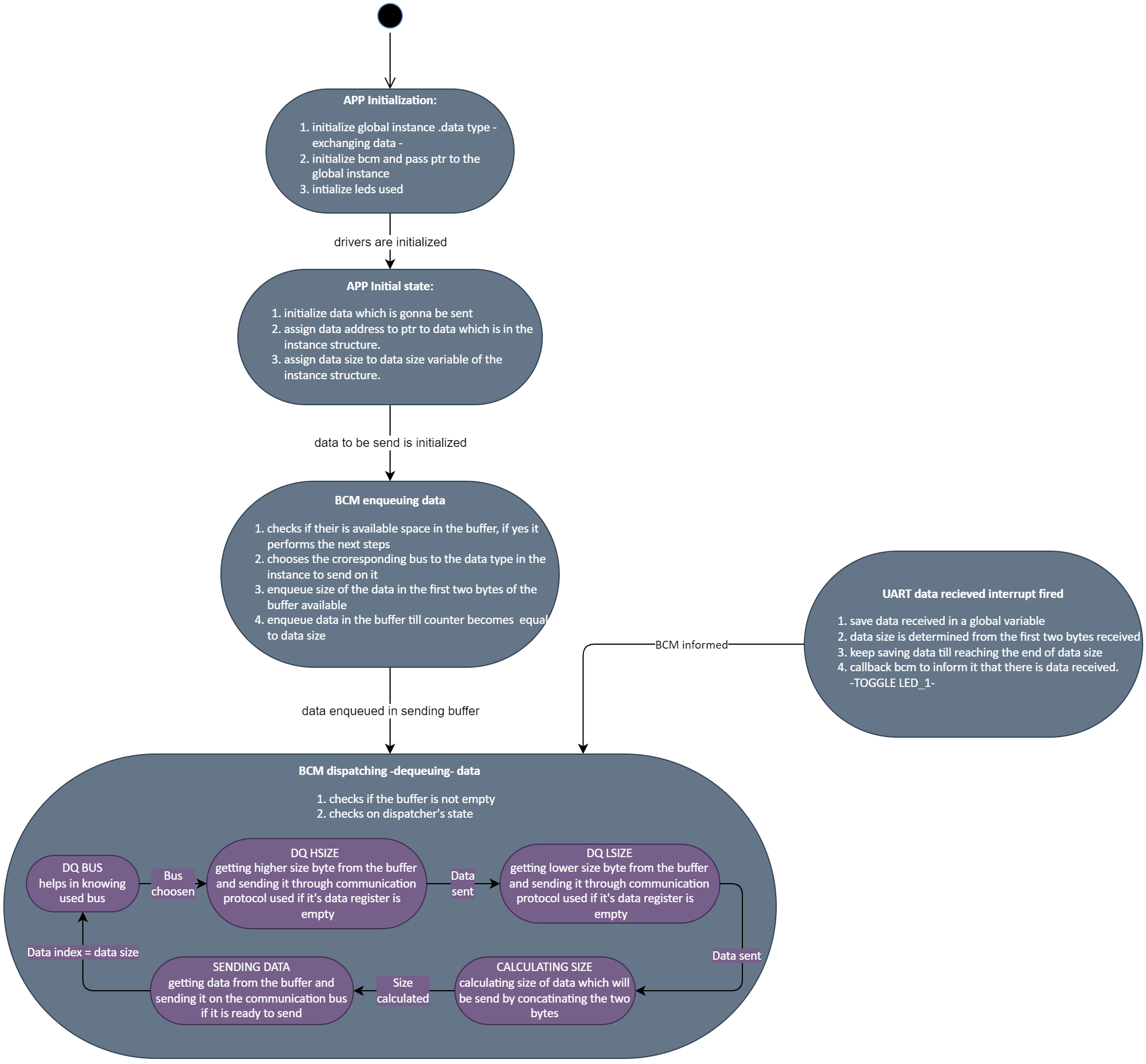
MCU 1 - please zoom to 150-200% to see a better quality -

MCU 2 - please zoom to 150-200% to see a better quality -

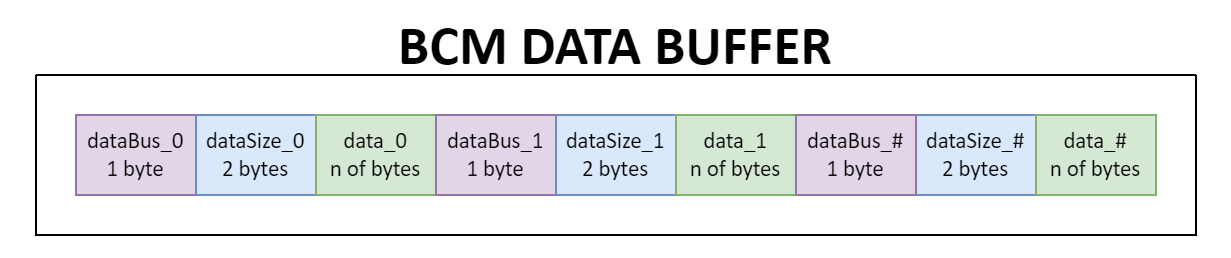


## State Machine Diagram

Zoom in for a better quality.



## BCM Data Buffer



## Modules Description

### DIO (Digital Input/Output) Module

The DIO driver 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.

### UART

The UART driver is responsible for initializing UART channel to configured specifications, it is used to send data through it and receive data too. All of its tasks are non-blocking.

### LED

The LED driver is used to initialize LEDs used as output and control them as it can turn them on or off or toggle them.

### BCM

The BCM driver is used to initialize communication protocols used to send/receive specific data types, its responsible to manage the communication process.

## Drivers’ Documentation

### MCAL Drivers’ Functions

#### DIO Driver

DIO\_init

|  |  |
| --- | --- |
| Syntax | DIO\_init(uint8\_t uint8\_portNumber, uint8\_t uint8\_pinNumber, uint8\_t uint8\_direction) |
| Description | Initializes DIO pins’ direction, output current. |
| Sync\Async | Synchronous |
| Reentrancy | Reentrant |
| Parameters (in) | uint8\_portNumber port number used  uint8\_pinNumber pin number used  uint8\_direction direction of the pin |
| Parameters (out) | None |
| Return value | enu\_DIO\_status\_t DIO\_OK  WRONG\_PIN\_NUMBER  WRONG\_PORT\_NUMBER  WRONG\_DIRECTION |

DIO\_write

|  |  |
| --- | --- |
| Syntax | DIO\_write(uint8\_t uint8\_portNumber, uint8\_t uint8\_pinNumber, uint8\_t uint8\_value) |
| Description | Write on DIO pins’ a specific output High or Low. |
| Sync\Async | Synchronous |
| Reentrancy | Reentrant |
| Parameters (in) | uint8\_portNumber port number used  uint8\_pinNumber pin number used  uint8\_value level of the pin |
| Parameters (out) | None |
| Return value | enu\_DIO\_status\_t DIO\_OK  WRONG\_PIN\_NUMBER  WRONG\_PORT\_NUMBER  WRONG\_VALUE |

DIO\_toggle

|  |  |
| --- | --- |
| Syntax | DIO\_toggle(uint8\_t uint8\_portNumber, uint8\_t uint8\_pinNumber) |
| Description | Toggle the output of a specific pin. |
| Sync\Async | Synchronous |
| Reentrancy | Reentrant |
| Parameters (in) | uint8\_portNumber port number used  uint8\_pinNumber pin number used |
| Parameters (out) | None |
| Return value | enu\_DIO\_status\_t DIO\_OK  WRONG\_PIN\_NUMBER  WRONG\_PORT\_NUMBER |

DIO\_read

|  |  |
| --- | --- |
| Syntax | DIO\_read(uint8\_t uint8\_portNumber, uint8\_t uint8\_pinNumber, uint8\_t \*uint8\_value) |
| Description | Read input from a pin and send it back in a pointer to uint8\_t |
| Sync\Async | Synchronous |
| Reentrancy | Reentrant |
| Parameters (in) | uint8\_portNumber port number used  uint8\_pinNumber pin number used |
| Parameters (out) | uint8\_value input value will be returned in that parameter |
| Return value | enu\_DIO\_status\_t DIO\_OK  WRONG\_PIN\_NUMBER  WRONG\_PORT\_NUMBER |

DIO\_pinPullUp

|  |  |
| --- | --- |
| Syntax | DIO\_pinPullUp(uint8\_t uint8\_portNumber, uint8\_t uint8\_pinNumber, uint8\_t uint8\_pullUpState) |
| Description | Disables/enables a pull up resistor to a specific input pin |
| Sync\Async | Synchronous |
| Reentrancy | Reentrant |
| Parameters (in) | uint8\_portNumber port number used  uint8\_pinNumber pin number used  uint8\_pullUpState pullup state |
| Parameters (out) | None |
| Return value | enu\_DIO\_status\_t DIO\_OK  WRONG\_PIN\_NUMBER  WRONG\_PORT\_NUMBER  WRONG\_VALUE |

#### UART Driver

UART\_init

|  |  |
| --- | --- |
| Syntax | UART\_init (void) |
| Description | Initializes UART pins’ direction, and specifications |
| Sync\Async | Synchronous |
| Reentrancy | Reentrant |
| Parameters (in) | None |
| Parameters (out) | None |
| Return value | enu\_UART\_status\_t UART\_OK = 0,  UART\_NOK |

UART\_deinit

|  |  |
| --- | --- |
| Syntax | UART\_deinit (void) |
| Description | Deinitializes UART |
| Sync\Async | Synchronous |
| Reentrancy | Reentrant |
| Parameters (in) | None |
| Parameters (out) | None |
| Return value | enu\_UART\_status\_t UART\_OK = 0,  UART\_NOK |

UART\_sendByte

|  |  |
| --- | --- |
| Syntax | UART\_sendByte (uint8\_t byte) |
| Description | Sending one byte through UART channel |
| Sync\Async | Asynchronous |
| Reentrancy | Reentrant |
| Parameters (in) | uint8\_byte |
| Parameters (out) | None |
| Return value | enu\_UART\_status\_t UART\_OK = 0,  UART\_SENDING = 1,  UART\_NOK |

UART\_setCallBack

|  |  |
| --- | --- |
| Syntax | UART\_setCallBack (void (\*ptr\_func)(void)) |
| Description | Set callback function |
| Sync\Async | Synchronous |
| Reentrancy | Reentrant |
| Parameters (in) | void (\*ptr\_func)(void) |
| Parameters (out) | None |
| Return value | enu\_UART\_status\_t UART\_OK = 0,  UART\_NOK |

UART\_receiveData

|  |  |
| --- | --- |
| Syntax | UART\_receiveData(uint8\_t\*\* ptr\_uint8\_receivedData, uint16\_t\* uint16\_dataSize) |
| Description | Receives data buffer |
| Sync\Async | Synchronous |
| Reentrancy | Reentrant |
| Parameters (in) | None |
| Parameters (out) | ptr\_ptr\_uint8\_receivedData  ptr\_uint16\_dataSize |
| Return value | enu\_UART\_status\_t UART\_OK = 0,  UART\_NOK |

UART\_isEmpty

|  |  |
| --- | --- |
| Syntax | UART\_isEmpty (void) |
| Description | Checks if data register is empty |
| Sync\Async | Synchronous |
| Reentrancy | Reentrant |
| Parameters (in) | None |
| Parameters (out) | ptr\_uint8\_receivedData  ptr\_uint16\_dataSize |
| Return value | enu\_UART\_bufferStatus\_t UART\_BUFFER\_NEMPTY  UART\_BUFFER\_EMPTY |

### HAL Drivers’ Functions

#### LED

LED\_init

|  |  |
| --- | --- |
| Syntax | LED\_init(void) |
| Description | Initializes LED pins’ direction as output |
| Sync\Async | Synchronous |
| Reentrancy | Reentrant |
| Parameters (in) | None |
| Parameters (out) | None |
| Return value | enu\_LED\_status\_t LED\_OK= 0,  LED\_WRONG\_LED\_PORT  LED\_WRONG\_LED\_PIN |

LED\_on

|  |  |
| --- | --- |
| Syntax | LED\_on(uint8\_t uint8\_ledID) |
| Description | Turns on a specific LED |
| Sync\Async | Synchronous |
| Reentrancy | Reentrant |
| Parameters (in) | uint8\_ledID |
| Parameters (out) | None |
| Return value | enu\_LED\_status\_t LED\_OK= 0,  LED\_WRONG\_LED\_PORT  LED\_WRONG\_LED\_PIN |

LED\_off

|  |  |
| --- | --- |
| Syntax | LED\_off(uint8\_t uint8\_ledID) |
| Description | Turns off a specific LED |
| Sync\Async | Synchronous |
| Reentrancy | Reentrant |
| Parameters (in) | uint8\_ledID |
| Parameters (out) | None |
| Return value | enu\_LED\_status\_t LED\_OK= 0,  LED\_WRONG\_LED\_PORT  LED\_WRONG\_LED\_PIN |

LED\_toggle

|  |  |
| --- | --- |
| Syntax | LED\_toggle(uint8\_t uint8\_ledID) |
| Description | Toggles a specific LED |
| Sync\Async | Synchronous |
| Reentrancy | Reentrant |
| Parameters (in) | uint8\_ledID |
| Parameters (out) | None |
| Return value | enu\_LED\_status\_t LED\_OK= 0,  LED\_WRONG\_LED\_PORT  LED\_WRONG\_LED\_PIN |

### SERVER Drivers’ functions

#### BCM

BCM\_init

|  |  |
| --- | --- |
| Syntax | BCM\_init (str\_BCM\_instance\_t\* ptr\_str\_BCM\_instance) |
| Description | Initializes a specific communication protocol |
| Sync\Async | Synchronous |
| Reentrancy | Reentrant |
| Parameters (in) | ptr\_str\_BCM\_instance |
| Parameters (out) | None |
| Return value | enu\_BCM\_status\_t BCM\_OK = 0,  BCM\_NOK |

BCM\_deinit

|  |  |
| --- | --- |
| Syntax | BCM\_deinit (str\_BCM\_instance\_t\* ptr\_str\_BCM\_instance) |
| Description | Deinitializes a specific communication protocol |
| Sync\Async | Synchronous |
| Reentrancy | Reentrant |
| Parameters (in) | ptr\_str\_BCM\_instance |
| Parameters (out) | None |
| Return value | enu\_BCM\_status\_t BCM\_OK = 0,  BCM\_NOK, |

BCM\_send\_n

|  |  |
| --- | --- |
| Syntax | BCM\_send\_n (str\_BCM\_instance\_t\* ptr\_str\_BCM\_instance) |
| Description | Enqueue any amount of data in data buffer to be send through a specific communication protocol using dispatcher |
| Sync\Async | Synchronous |
| Reentrancy | Reentrant |
| Parameters (in) | ptr\_str\_BCM\_instance |
| Parameters (out) | None |
| Return value | enu\_BCM\_status\_t BCM\_OK = 0,  BCM\_NOK,  BCM\_FULL |

BCM\_dispatcher

|  |  |
| --- | --- |
| Syntax | BCM\_dispatcher (void) |
| Description | Dequeue the data from data buffer and send it through a specific communication protocol |
| Sync\Async | Asynchronous |
| Reentrancy | Reentrant |
| Parameters (in) | ptr\_str\_BCM\_instance |
| Parameters (out) | None |
| Return value | enu\_BCM\_status\_t BCM\_OK = 0,  BCM\_NOK,  BCM\_EMPTY |

BCM\_receiveData

|  |  |
| --- | --- |
| Syntax | BCM\_receiveData (uint8\_t\*\*ptr\_ptr\_uint8\_receivedData, uint16\_t\* ptr\_uint16\_dataSize) |
| Description | Receives address of data received to a specific communication protocol and the size of it |
| Sync\Async | Synchronous |
| Reentrancy | Reentrant |
| Parameters (in) | None |
| Parameters (out) | ptr\_ptr\_uint8\_receivedData  ptr\_uint16\_dataSize |
| Return value | enu\_BCM\_status\_t BCM\_OK = 0,  BCM\_NOK, |

BCM\_setCallBack

|  |  |
| --- | --- |
| Syntax | BCM\_setCallBack(str\_BCM\_instance\_t\* ptr\_str\_BCM\_instance, void(\*ptr\_func) (void)) |
| Description | Set callback function, which is called when there is data received through any communication protocol |
| Sync\Async | Synchronous |
| Reentrancy | Reentrant |
| Parameters (in) | ptr\_str\_BCM\_instance  void(\*ptr\_func) (void) |
| Parameters (out) | None |
| Return value | enu\_BCM\_status\_t BCM\_OK = 0,  BCM\_NOK, |

BCM\_dataSentCallBack

|  |  |
| --- | --- |
| Syntax | BCM\_ dataSentCallBack (void(\*ptr\_func) (void)) |
| Description | Set callback function, which is called when there is data transmitted through any communication protocol |
| Sync\Async | Synchronous |
| Reentrancy | Reentrant |
| Parameters (in) | void(\*ptr\_func) (void) |
| Parameters (out) | None |
| Return value | enu\_BCM\_status\_t BCM\_OK = 0,  BCM\_NOK |

#### Queue

QUEUE\_isFull

|  |  |
| --- | --- |
| Syntax | QUEUE\_isFull(sint16\_t sint16\_front, sint16\_t sint16\_rear, uint16\_t uint16\_queueSize) |
| Description | Checks if the queue is full or not |
| Sync\Async | Synchronous |
| Reentrancy | Reentrant |
| Parameters (in) | sint16\_front  sint16\_rear  uint16\_queueSize |
| Parameters (out) | None |
| Return value | enu\_BCM\_status\_t QUEUE\_NFULL = 0,  QUEUE\_FULL, |

QUEUE\_isEmpty

|  |  |
| --- | --- |
| Syntax | QUEUE\_isEmpty(sint16\_t sint16\_front) |
| Description | Checks if the queue is empty or not |
| Sync\Async | Synchronous |
| Reentrancy | Reentrant |
| Parameters (in) | sint16\_front |
| Parameters (out) | None |
| Return value | enu\_BCM\_status\_t QUEUE\_NEMPTY,  QUEUE\_EMPTY |

QUEUE\_enQueue

|  |  |
| --- | --- |
| Syntax | QUEUE\_enQueue( sint16\_t\* ptr\_sint16\_front, sint16\_t\* ptr\_sint16\_rear, uint8\_t\* ptr\_uint8\_queue, uint16\_t uint16\_queueSize, uint8\_t uint8\_element) |
| Description | Enqueue an element in the queue |
| Sync\Async | Synchronous |
| Reentrancy | Reentrant |
| Parameters (in) | ptr\_uint8\_queue  uint8\_element  uint16\_queueSize |
| Parameters (out) | ptr\_sint16\_front  ptr\_sint16\_rear |
| Return value | enu\_BCM\_status\_t QUEUE\_NFULL = 0,  QUEUE\_FULL |

QUEUE\_deQueue

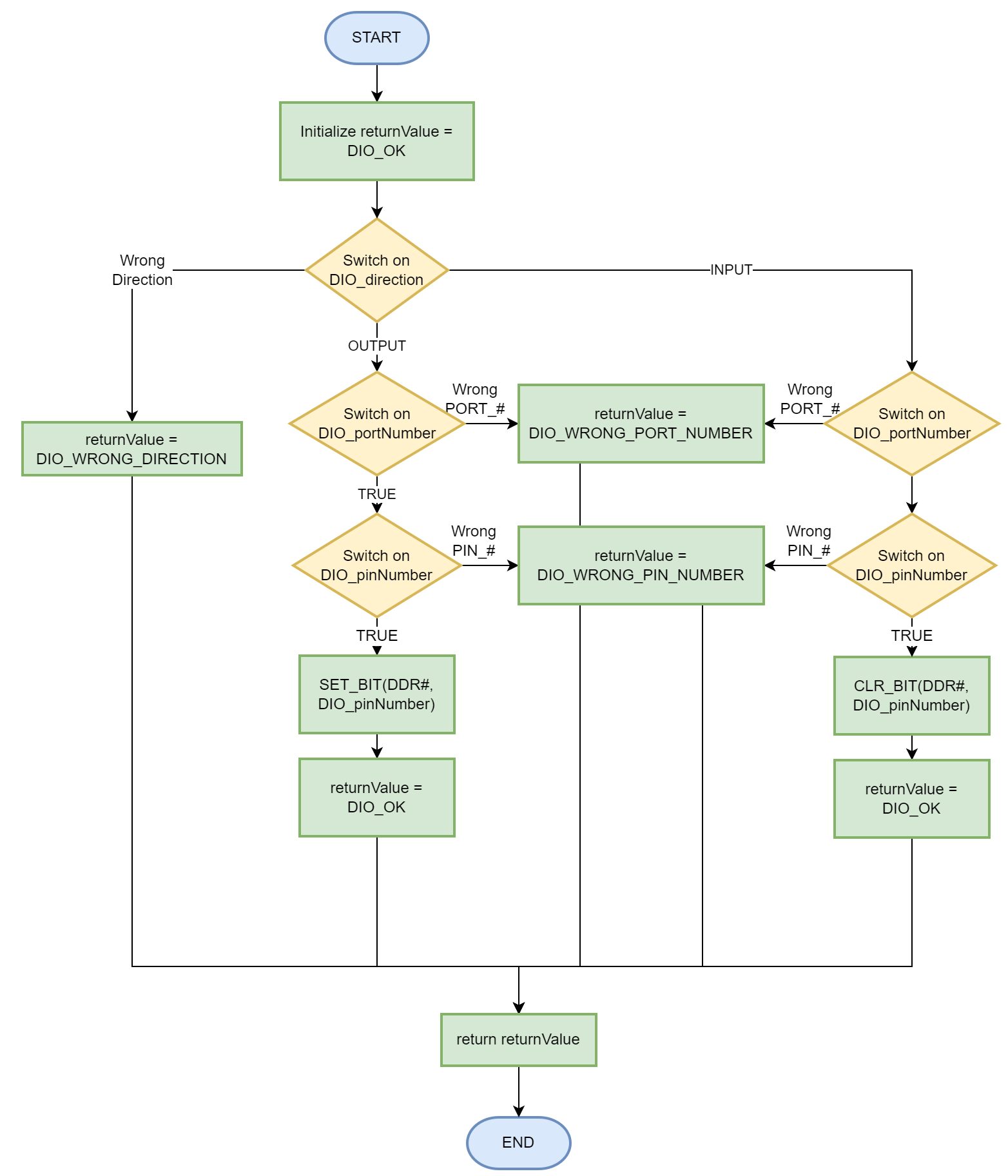
|  |  |
| --- | --- |
| Syntax | QUEUE\_deQueue(sint16\_t\* ptr\_sint16\_front, sint16\_t\* ptr\_sint16\_rear, uint8\_t\* ptr\_uint8\_queue, uint16\_t uint16\_queueSize, uint8\_t\* ptr\_uint8\_element) |
| Description | Dequeue an element from the queue |
| Sync\Async | Synchronous |
| Reentrancy | Reentrant |
| Parameters (in) | ptr\_uint8\_queue  uint16\_queueSize |
| Parameters (out) | ptr\_sint16\_front  ptr\_sint16\_rear  ptr\_uint8\_element |
| Return value | enu\_BCM\_status\_t QUEUE\_NEMPTY,  QUEUE\_EMPTY |

# Low Level Design

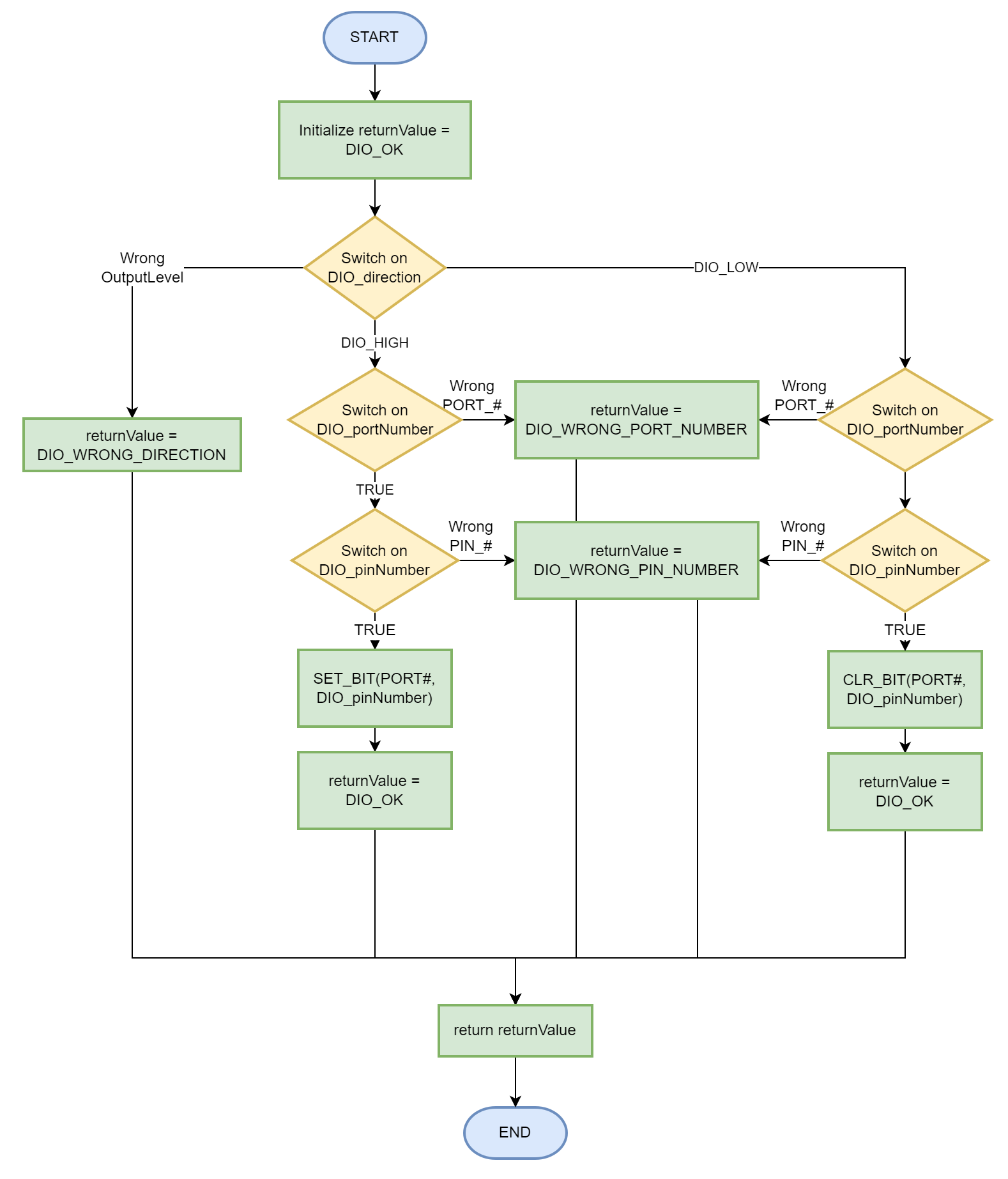
## MCAL Layer

### DIO Module

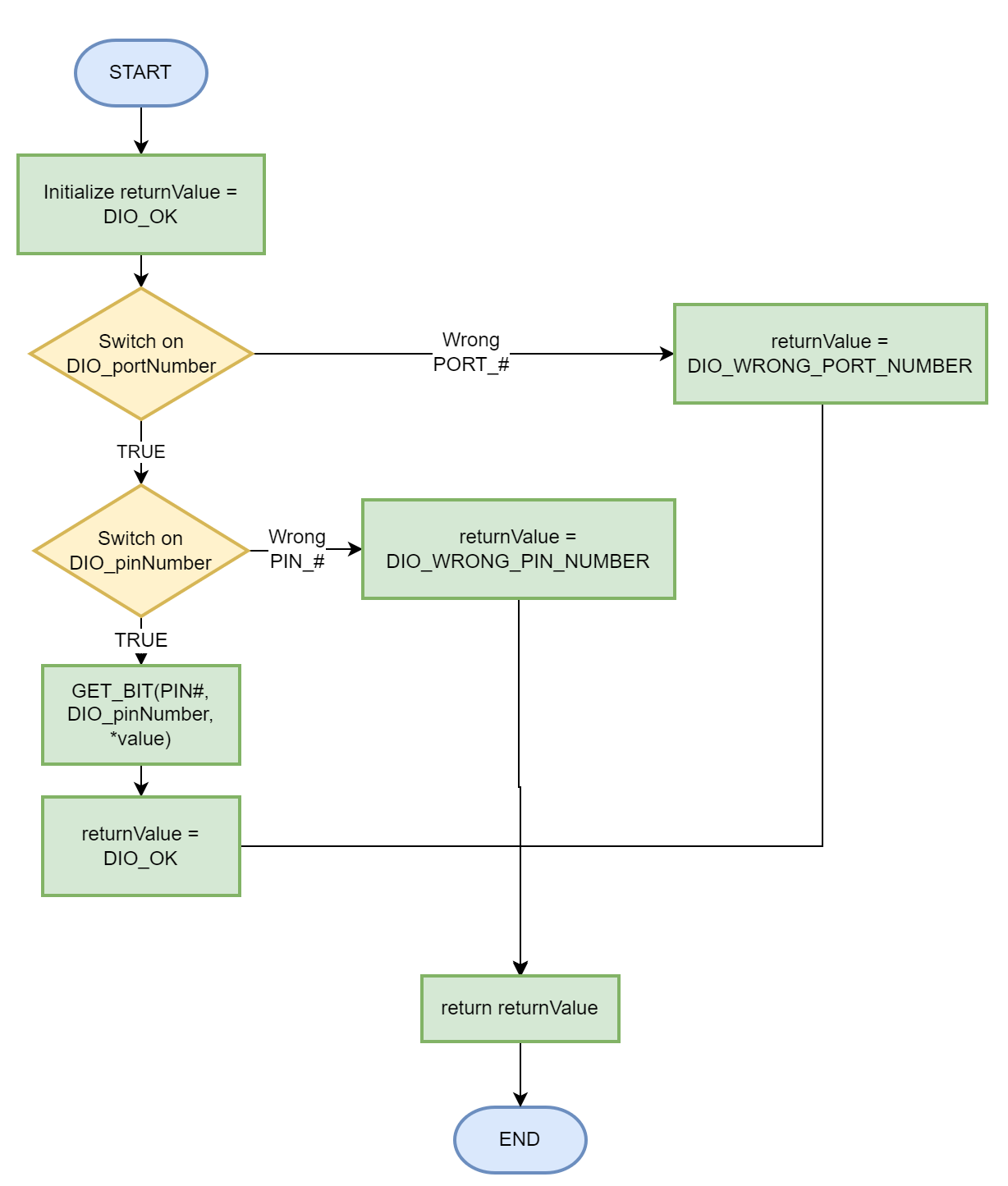
#### DIO\_init



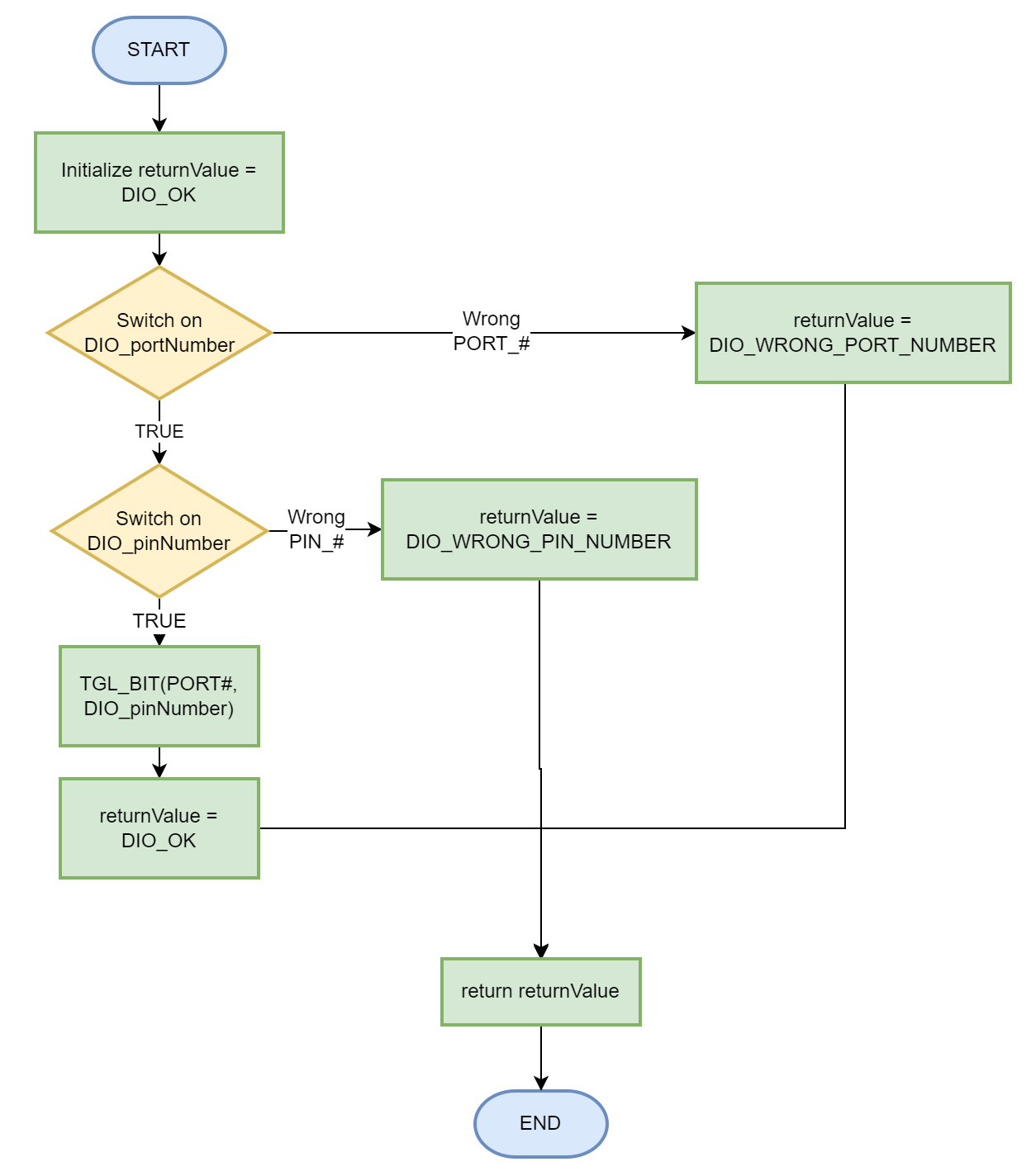
#### DIO\_write



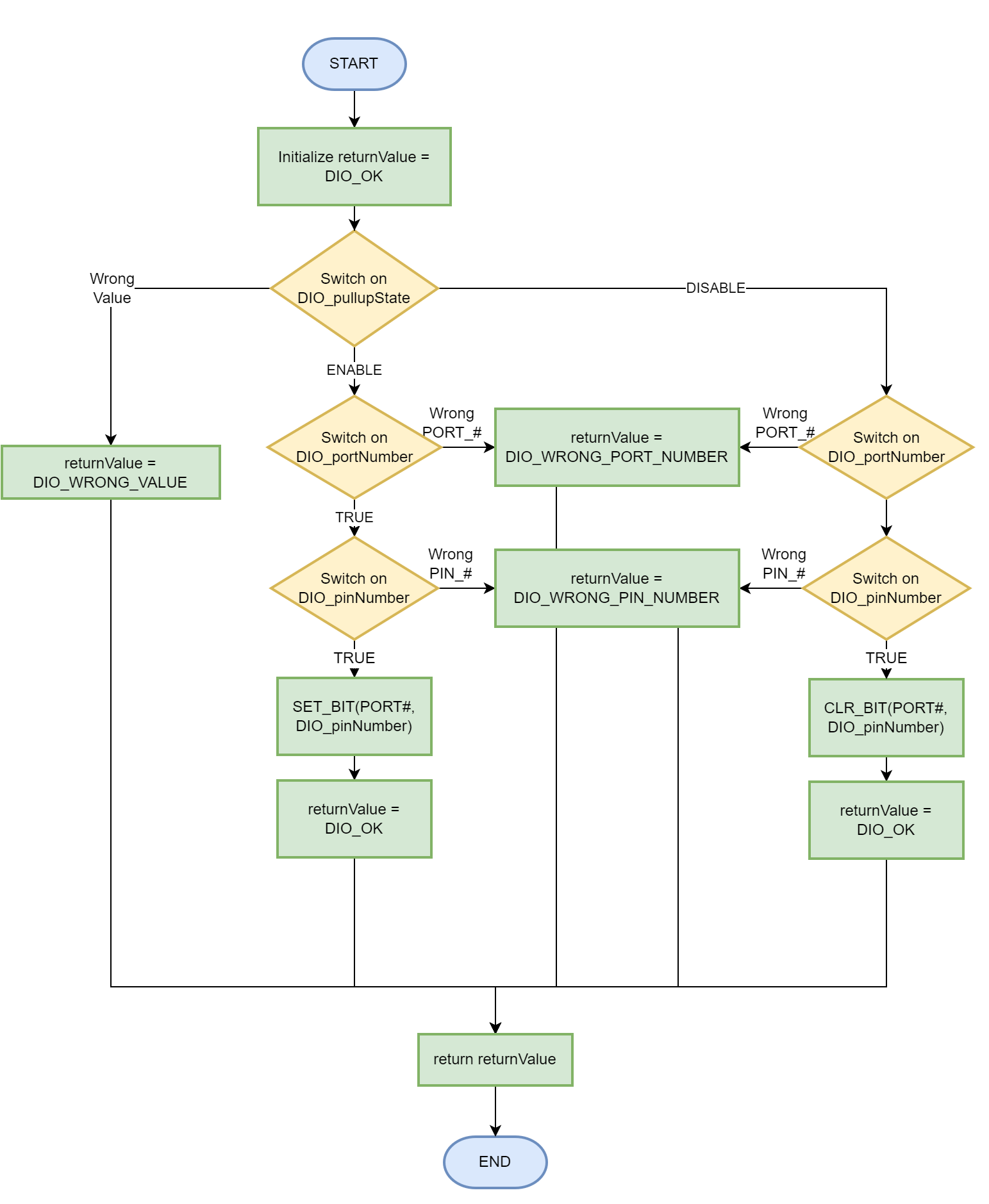
#### DIO\_read



#### DIO\_toggle

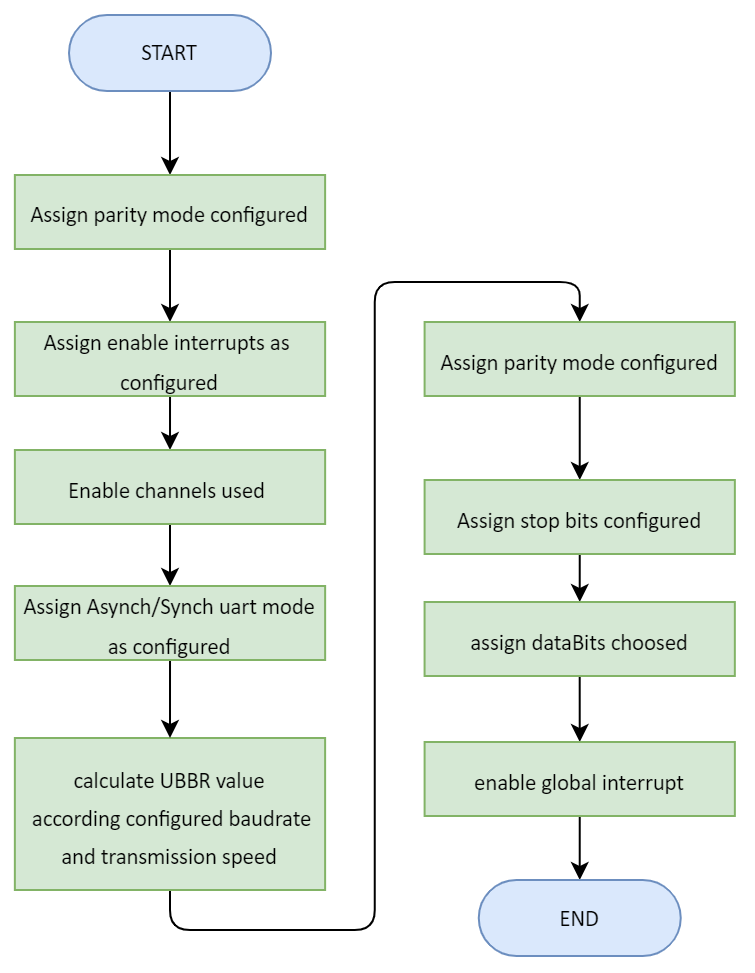


#### DIO\_pinPullUp

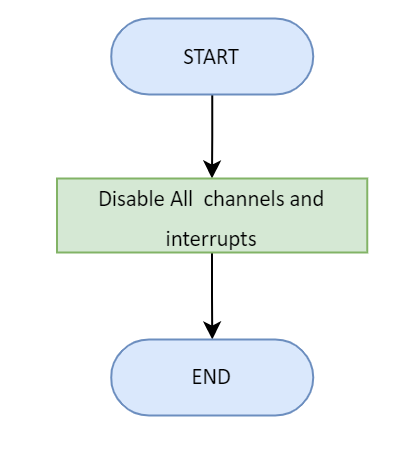


### UART

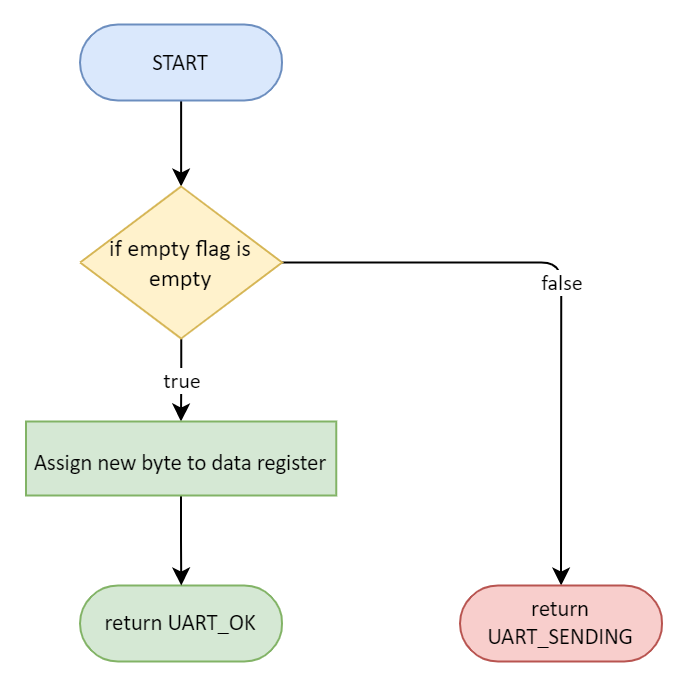
#### UART\_init



#### UART\_deinit



##### UART\_sendByte

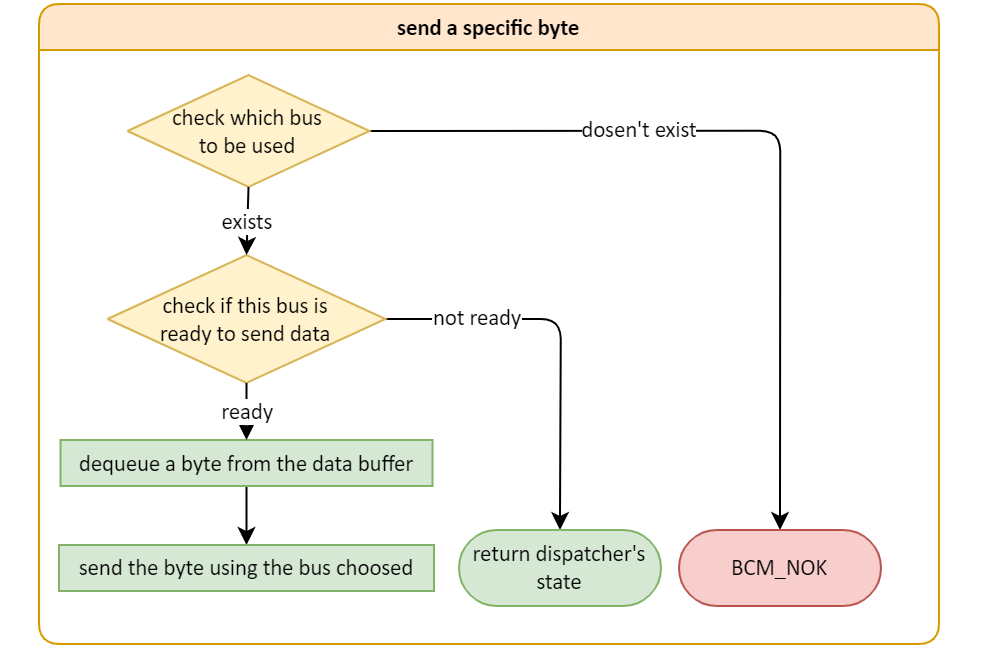


#### ISR(USART\_RXC\_INT)

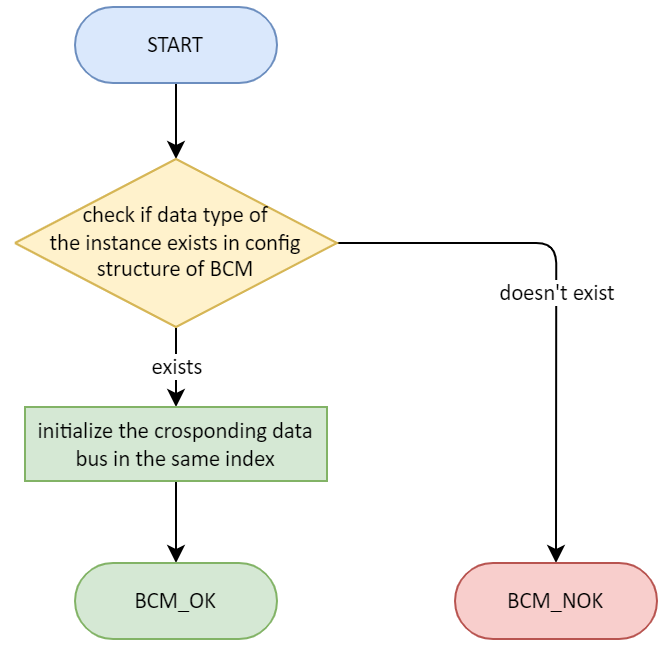
## Server Layer

### BCM

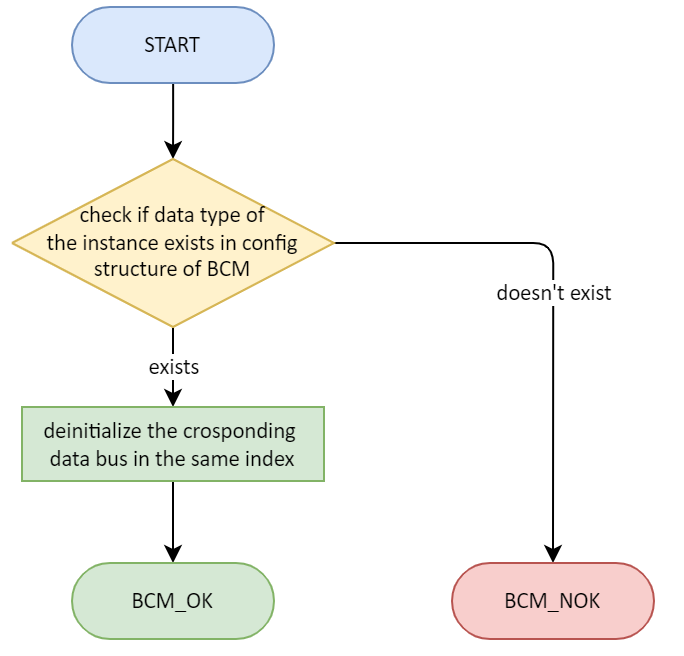
#### Send a specific byte process



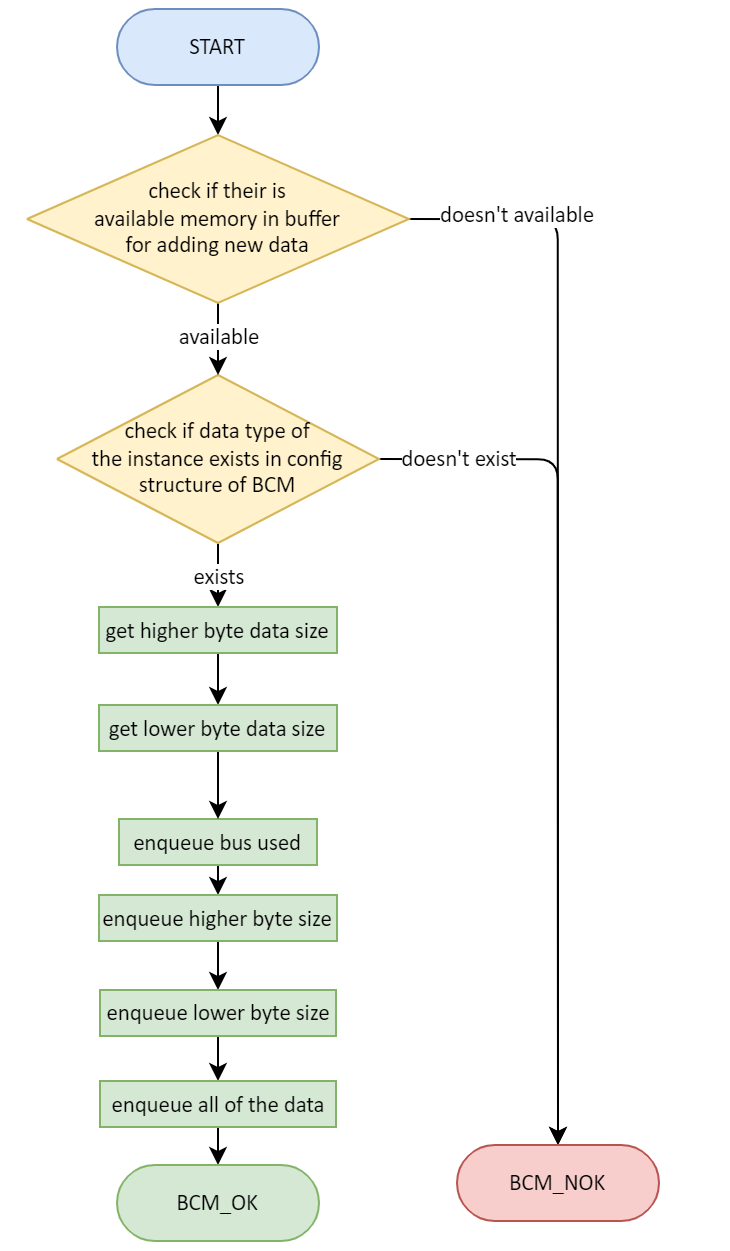
#### BCM\_init



#### BCM\_deinit



#### BCM\_send\_n



#### BCM\_dispatcher

