BASIC COMMUNICATION MANAGER DESIGN

Presented to: Sprints

Prepared by: Ahmed Hesham

Contents

1. 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

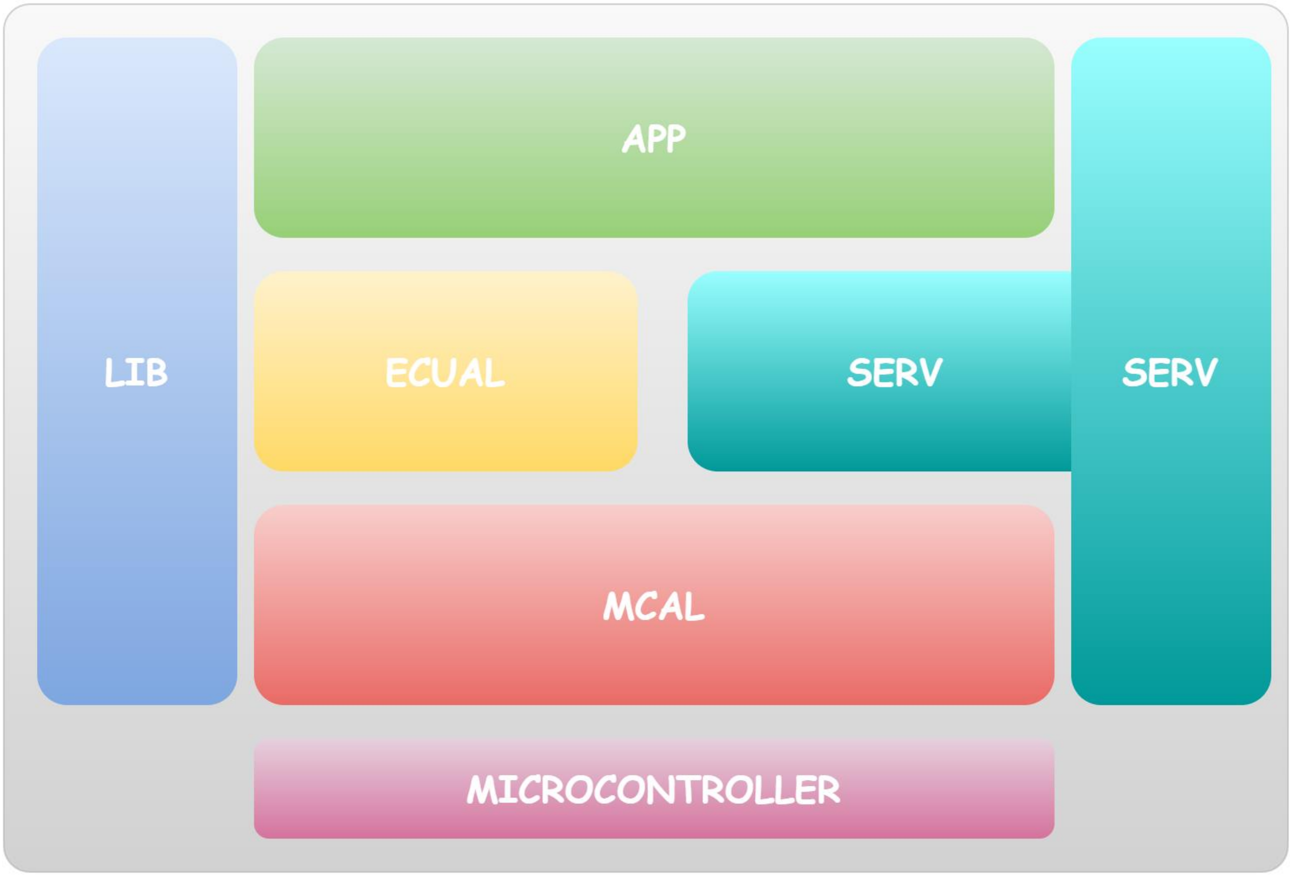
1.1. Project Components

- 2 ATmega32 microcontroller
- 2 LEDS for each microcontroller

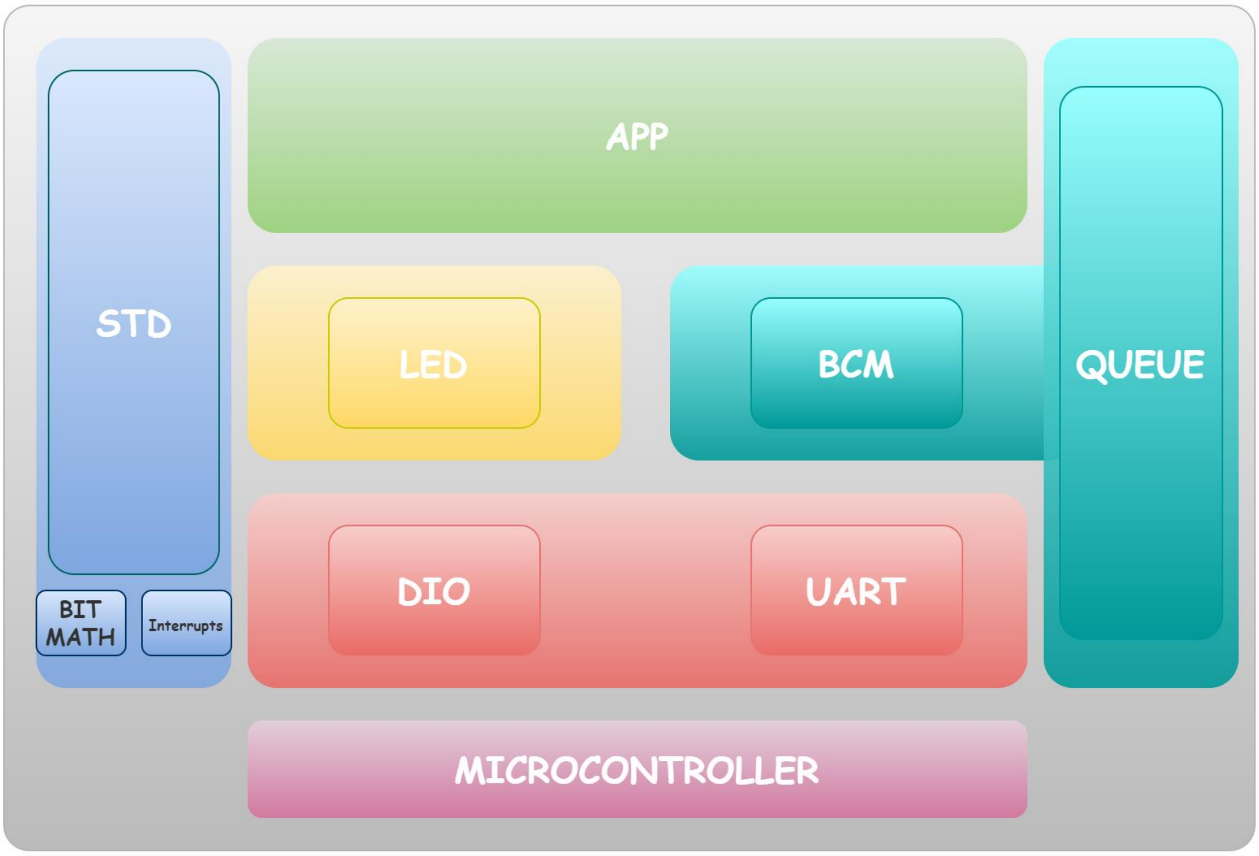
2. High Level Design

2.1. System Architecture

2.1.1. Layered Architecture

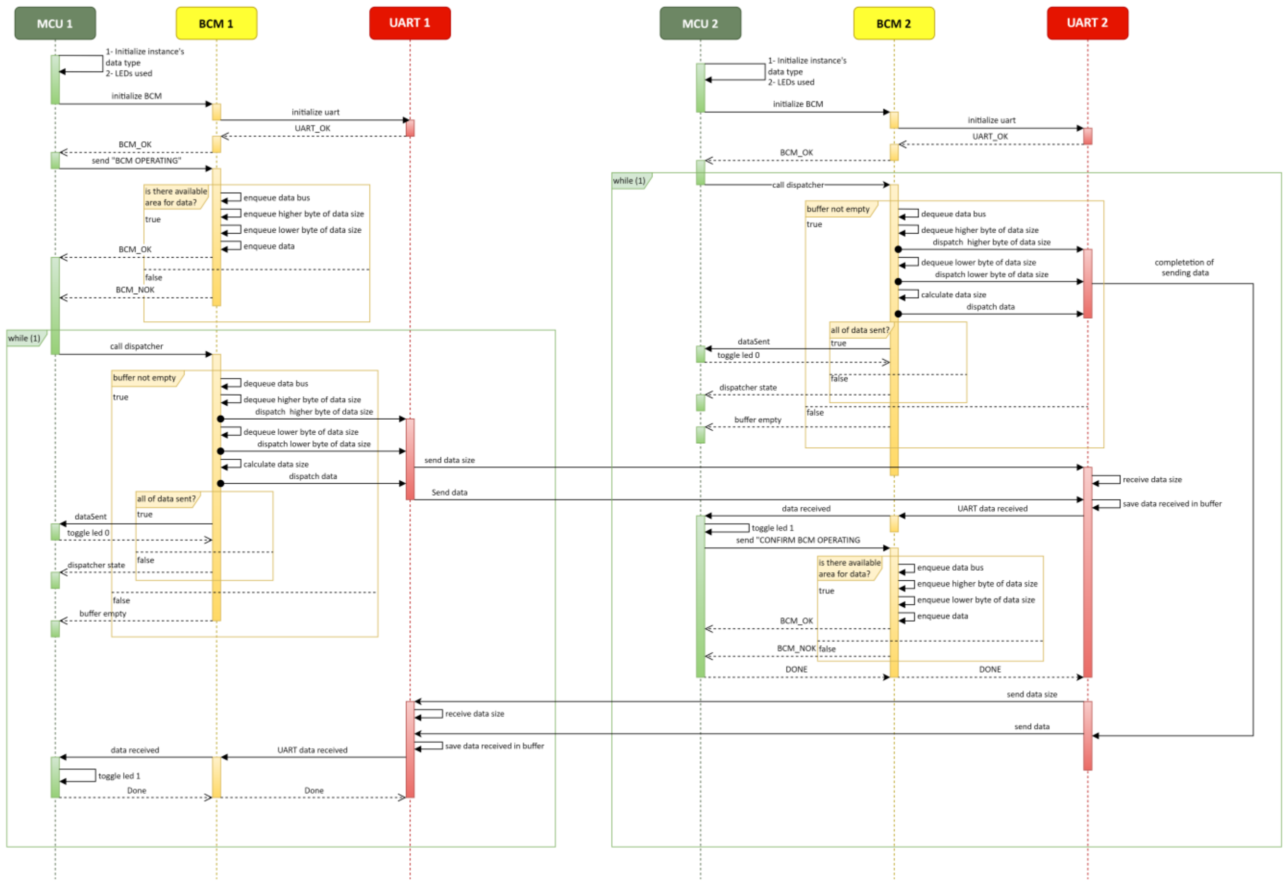


2.1.2. Design

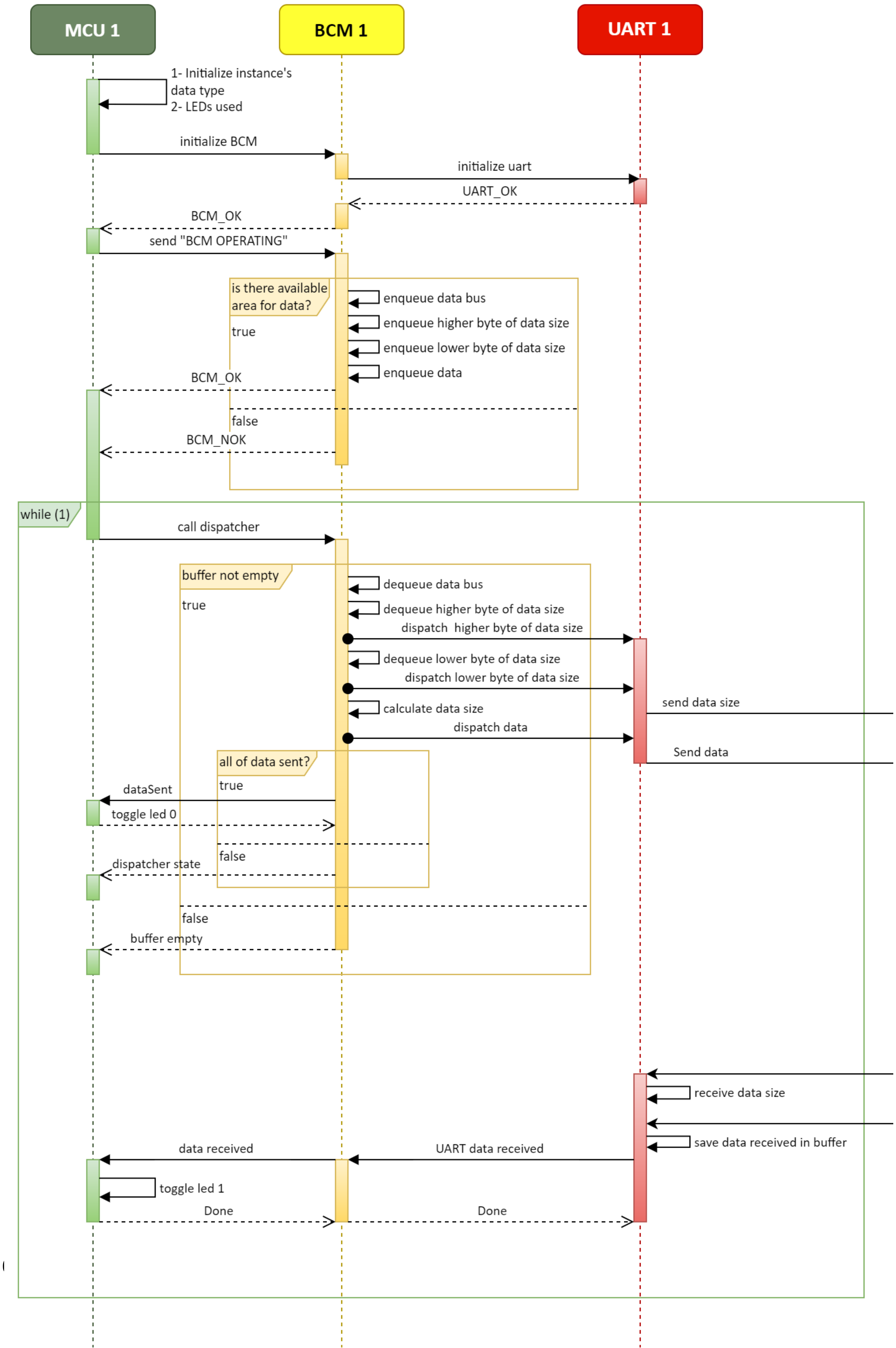

STD
BIT MATH
Interrupts
APP
LED
BCM
QUEUE
DIO
UART
MICROCONTROLLER

2.2. Sequence Diagram

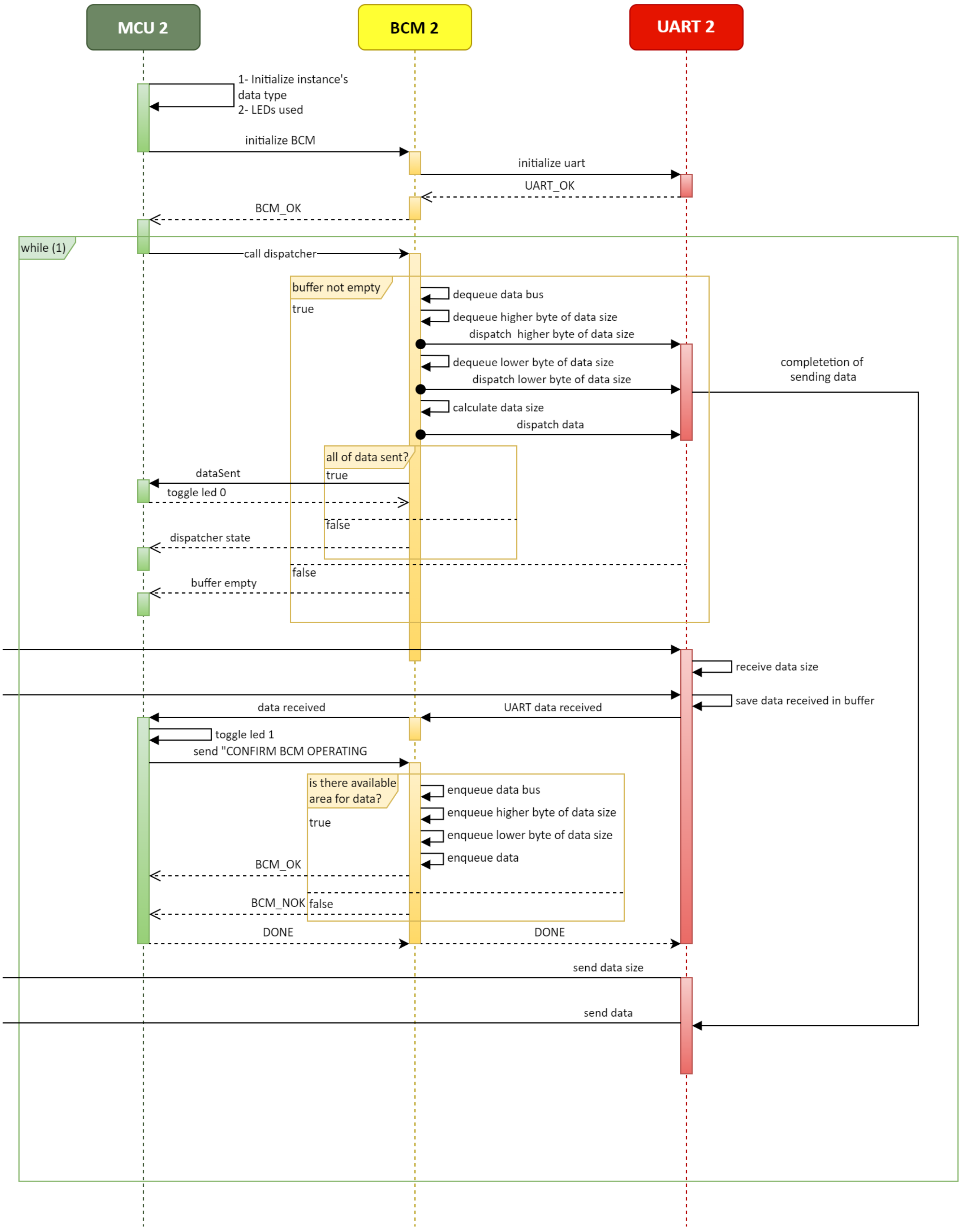
This whole of the sequence, you can find a zoomed parts in the next pages.



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

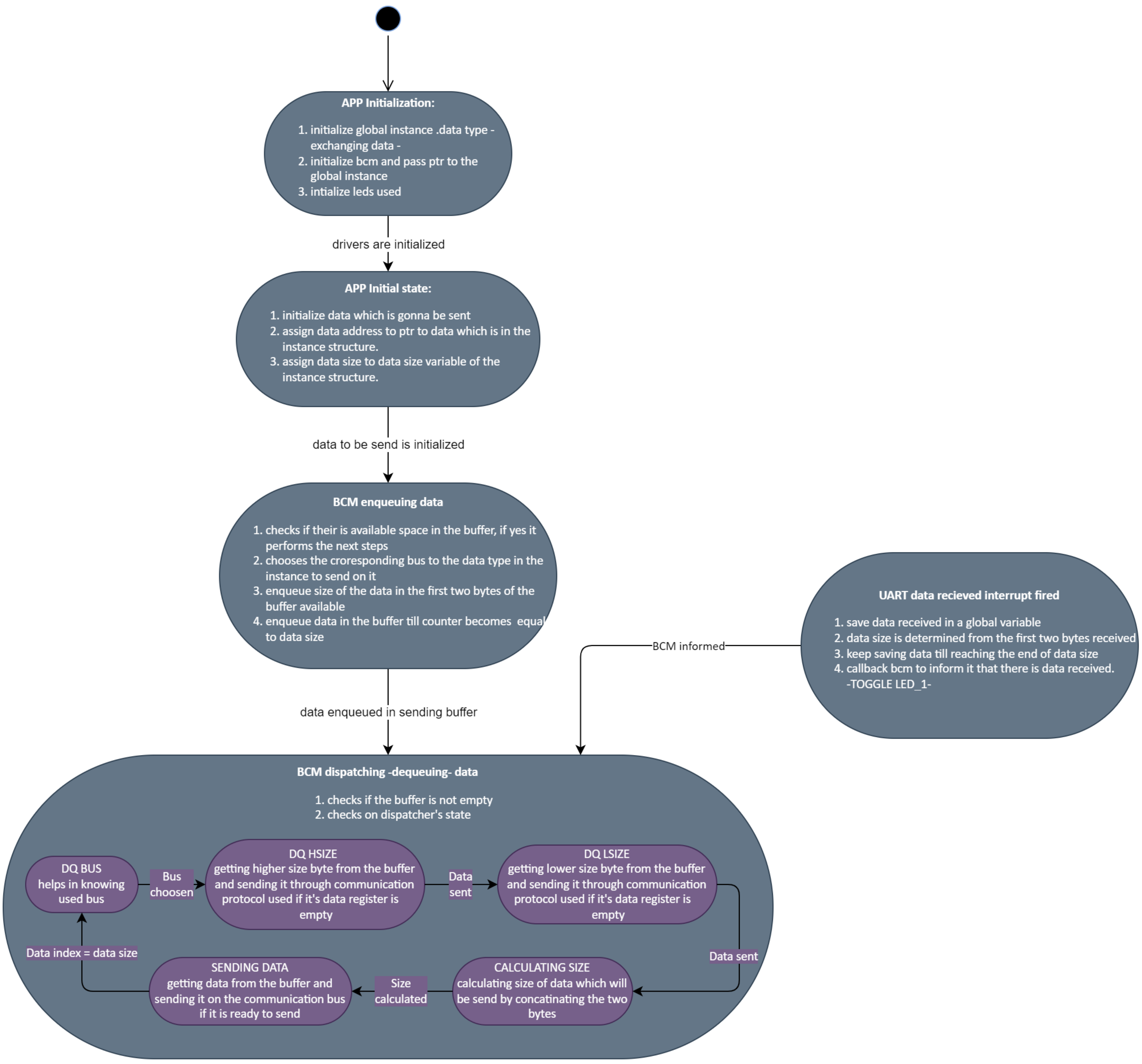


MCU 2 - please zoom to 150-200% to see a better quality -



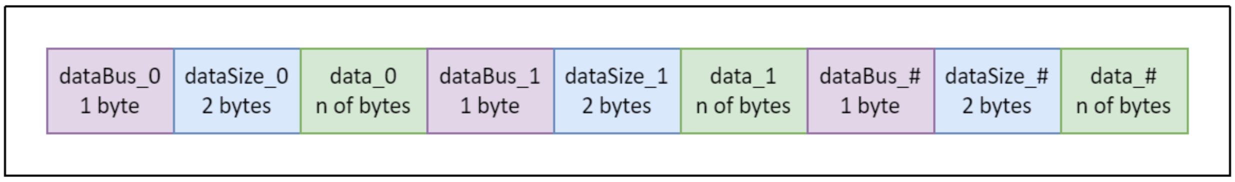
2.3. State Machine Diagram

Zoom in for a better quality.


APP Initialization:
1. initialize global instance .data type - exchanging data -
2. initialize bcm and pass ptr to the global instance
3. intialize leds used
drivers are initialized
APP Initial state:
1. initialize data which is gonna be sent
2. assign data address to ptr to data which is in the instance structure.
3. assign data size to data size variable of the instance structure.
data to be send is initialized
BCM enqueuing data
1. checks if their is available space in the buffer, if yes it performs the next steps
2. chooses the coresponding bus to the data type in the instance to send on it
3. enqueue size of the data in the first two bytes of the buffer available
4. enqueue data in the buffer till counter becomes equal to data size
data enqueued in sending buffer
BCM informed
UART data recieved interrupt fired
1. save data received in a global variable
2. data size is determined from the first two bytes received
3. keep saving data till reaching the end of data size
4. callback bcm to inform it that there is data received. -TOGGLE LED_1-
BCM dispatching -dequeuing- data
1. checks if the buffer is not empty
2. checks on dispatcher's state
DQ BUS helps in knowing used bus
Bus choosen
DQ HSIZE getting higher size byte from the buffer and sending it through communication protocol used if it's data register is empty
Data sent
DQ LSIZE getting lower size byte from the buffer and sending it through communication protocol used if it's data register is empty
Data sent
CALCULATING SIZE calculating size of data which will be send by concatinating the two bytes
Size calculated
SENDING DATA getting data from the buffer and sending it on the communication bus if it is ready to send
Data index = data size

2.4. BCM Data Buffer

BCM DATA BUFFER



dataBus_0	dataSize_0	data_0	dataBus_1	dataSize_1	data_1	dataBus_#	dataSize_#	data_#
1 byte	2 bytes	n of bytes	1 byte	2 bytes	n of bytes	1 byte	2 bytes	n of bytes

2.5. Modules Description

2.5.1. 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.

2.5.2. 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.

2.5.3. 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.

2.5.4. BCM

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

2.6. Drivers’ Documentation

2.6.1. MCAL Drivers’ Functions

2.6.1.1. 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 uint8_pinNumber uint8_direction	port number used pin number used 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 uint8_pinNumber uint8_value	port number used pin number used 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 uint8_pinNumber	port number used 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 uint8_pinNumber	port number used 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 uint8_pinNumber uint8_pullUpState	port number used pin number used pullup state
Parameters (out)	None	
Return value	enu_DIO_status_t	DIO_OK WRONG_PIN_NUMBER WRONG_PORT_NUMBER WRONG_VALUE

2.6.1.2. 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

2.6.2. HAL Drivers' Functions

2.6.2.1. 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

2.6.3. SERVER Drivers' functions

2.6.3.1. 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

2.6.3.2. 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

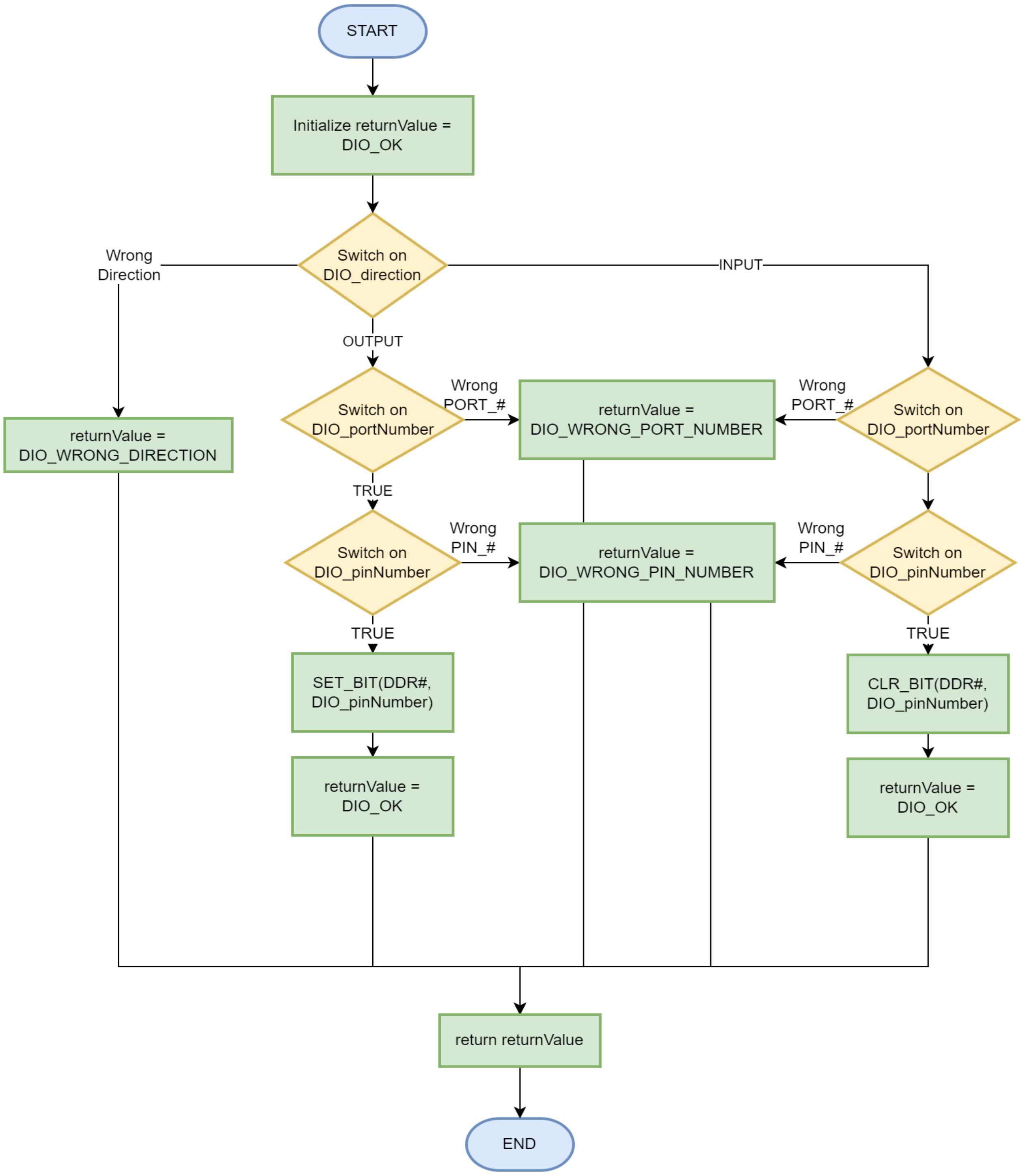
Syntax	QUEUE_deQueue(sint16_t* ptr_sint16_front, sint16_t* ptr_sint16_rear, uint8_t* ptr_uint8_queue, uint16_t uint16_queueSize, uint8_t* ptr_uint8_element)
Description	Dequeue an element from the queue
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	ptr_uint8_queue uint16_queueSize
Parameters (out)	ptr_sint16_front ptr_sint16_rear ptr_uint8_element
Return value	enu_BCM_status_t QUEUE_NEMPTY, QUEUE_EMPTY

3. Low Level Design

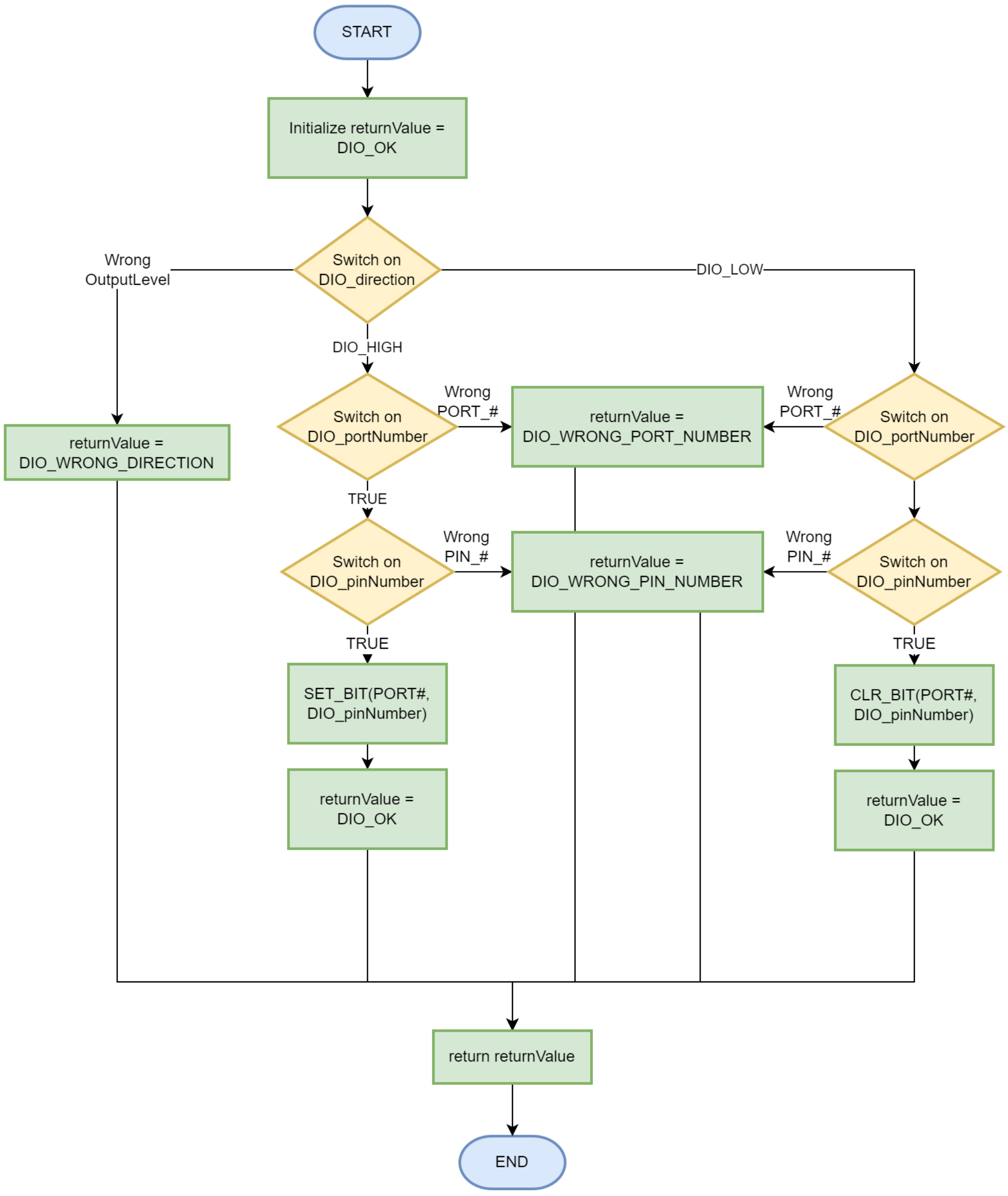
3.1. MCAL Layer

3.1.1. DIO Module

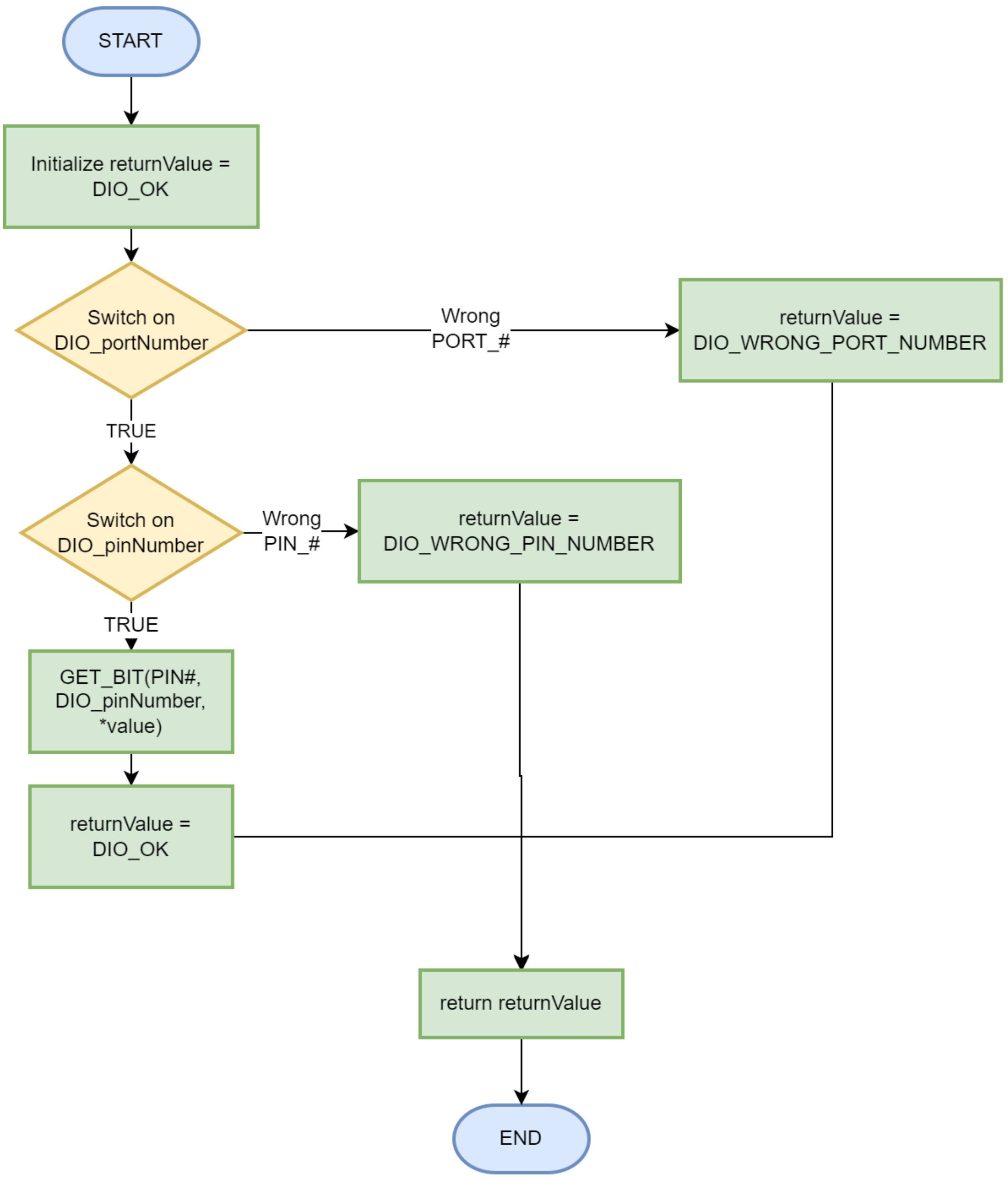
3.1.1.1. DIO_init



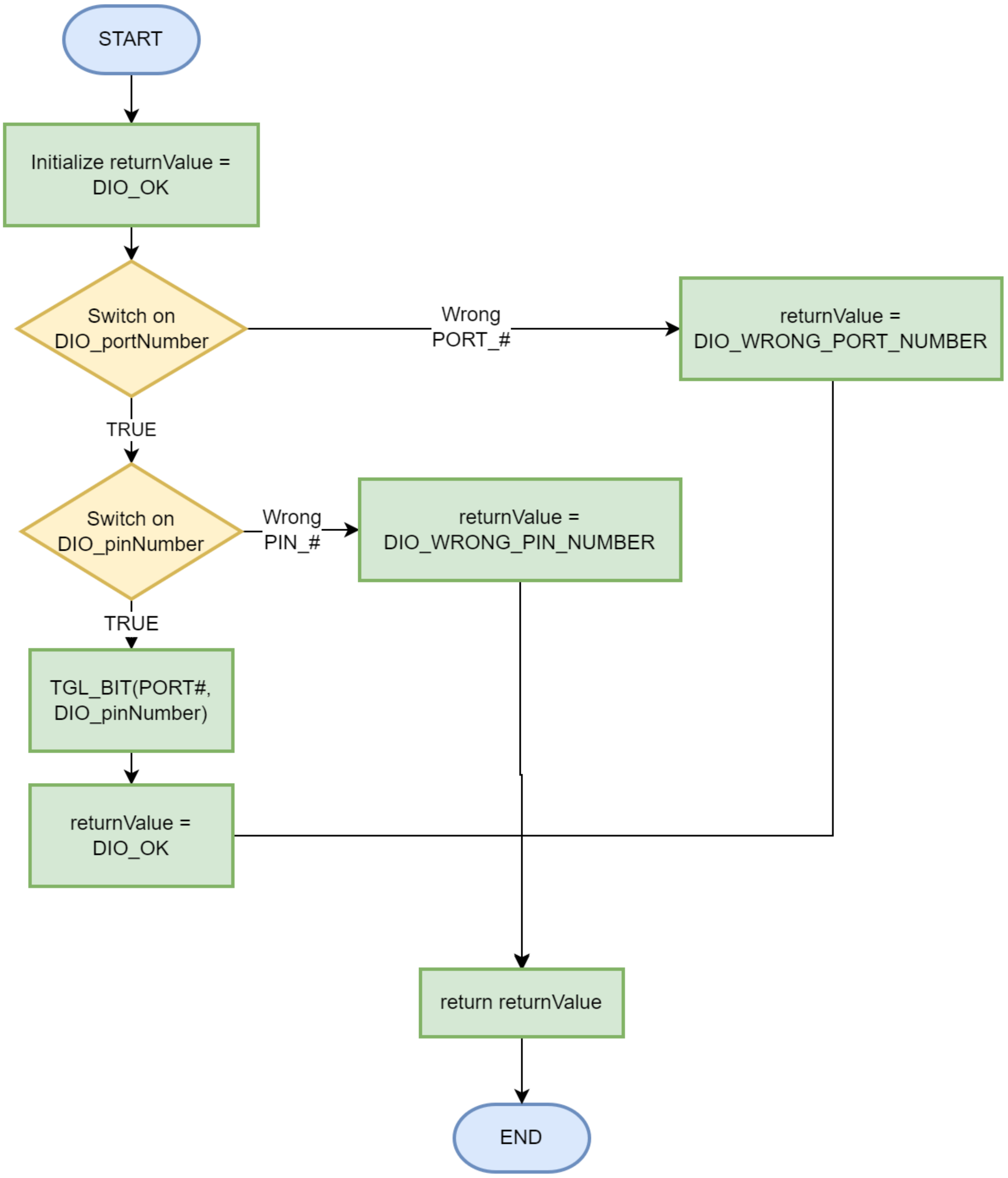
3.1.1.2. DIO_write



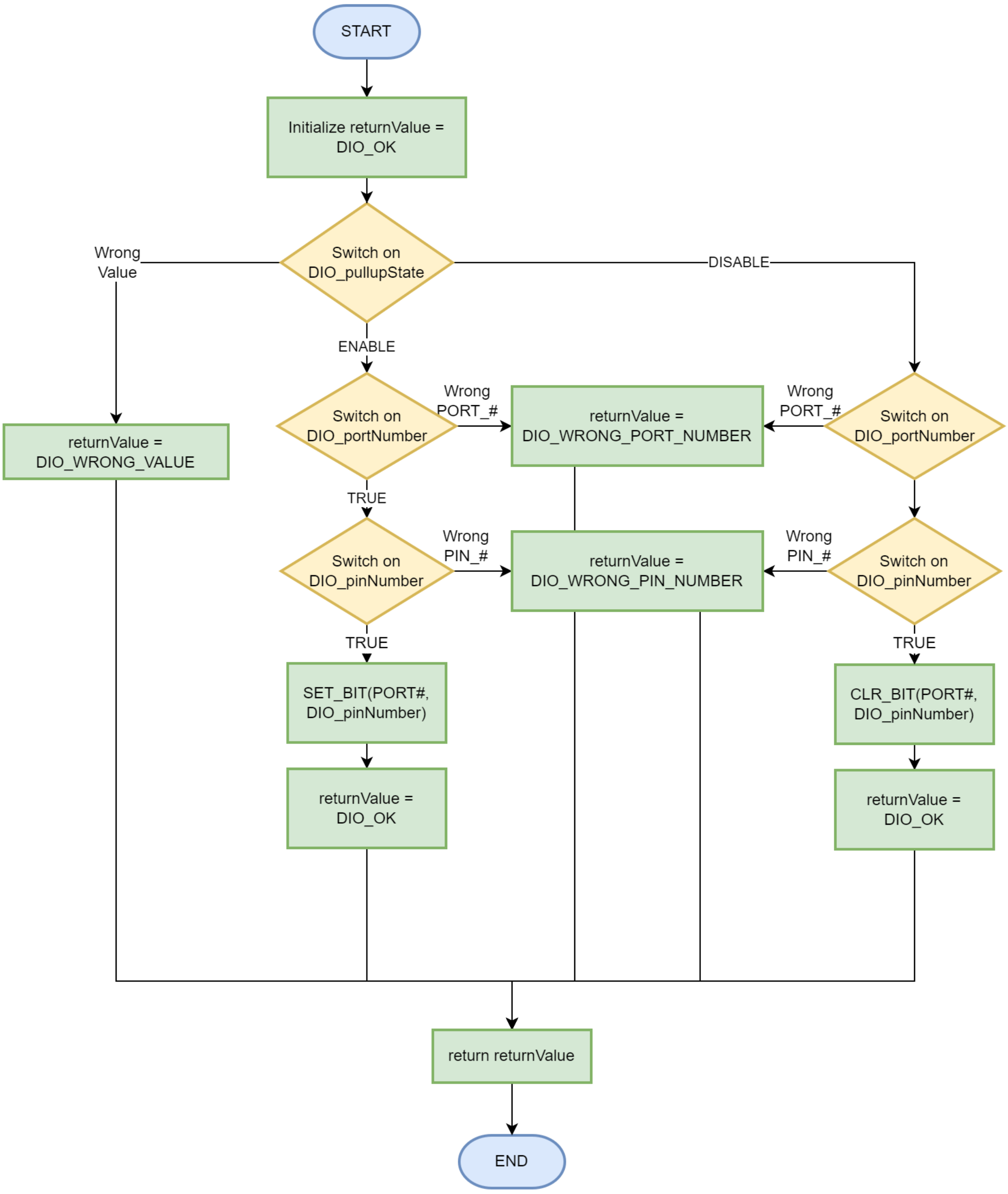
3.1.1.3. DIO_read


START
Initialize returnValue = DIO_OK
Switch on DIO_portNumber
Wrong PORT_#
returnValue = DIO_WRONG_PORT_NUMBER
TRUE
Switch on DIO_pinNumber
Wrong PIN_#
returnValue = DIO_WRONG_PIN_NUMBER
TRUE
GET_BIT(PIN#, DIO_pinNumber, *value)
returnValue = DIO_OK
return returnValue
END

3.1.1.4. DIO_toggle

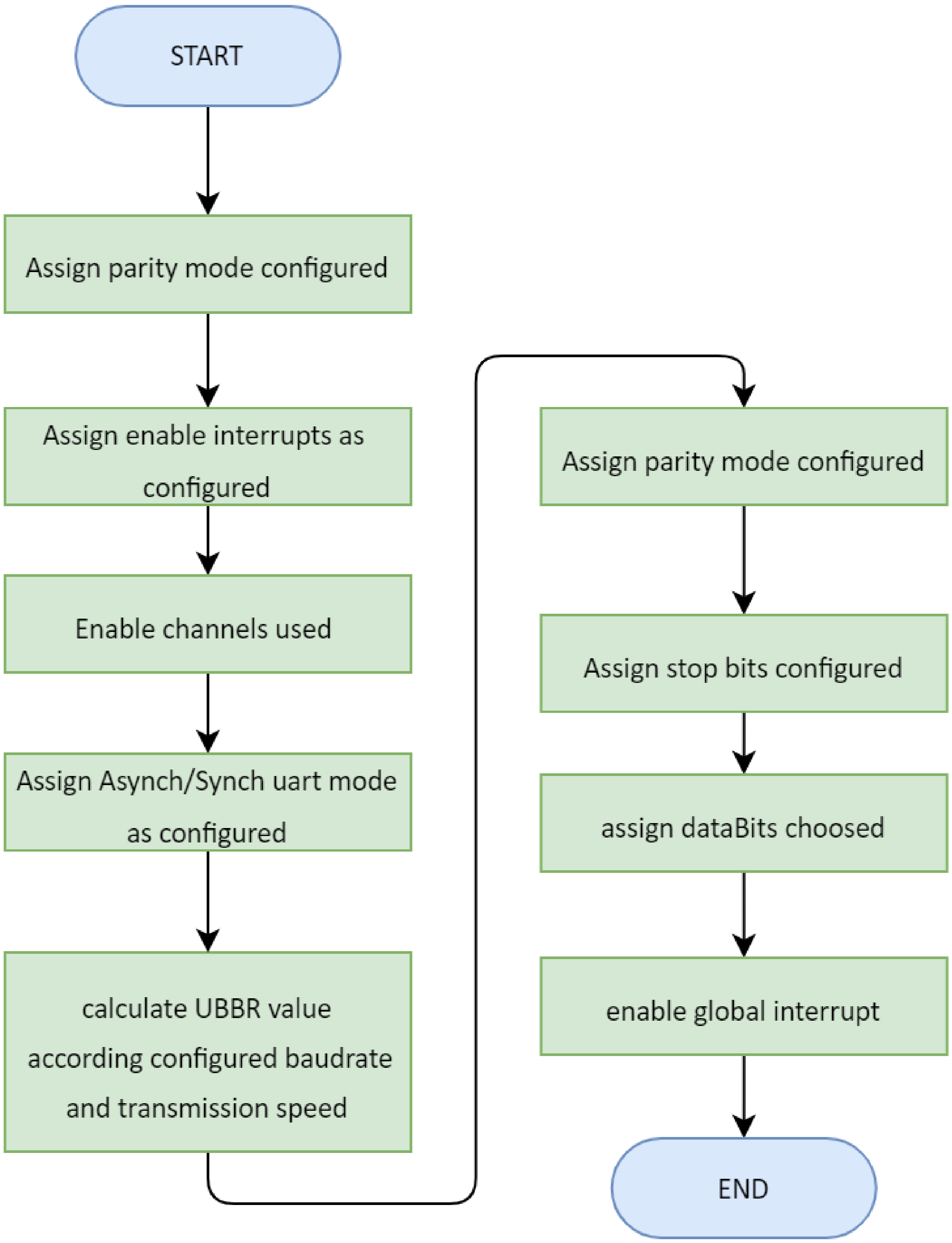

START
Initialize returnValue = DIO_OK
Switch on DIO_portNumber
Wrong PORT_#
returnValue = DIO_WRONG_PORT_NUMBER
TRUE
Switch on DIO_pinNumber
Wrong PIN_#
returnValue = DIO_WRONG_PIN_NUMBER
TRUE
TGL_BIT(PORT#, DIO_pinNumber)
returnValue = DIO_OK
return returnValue
END

3.1.1.5. DIO_pinPullUp

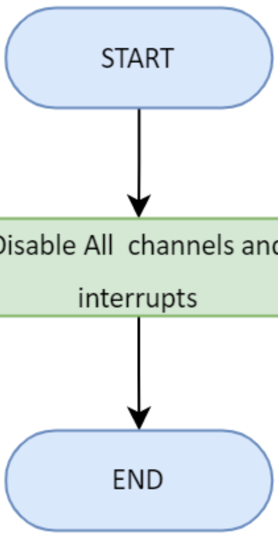


3.1.2. UART

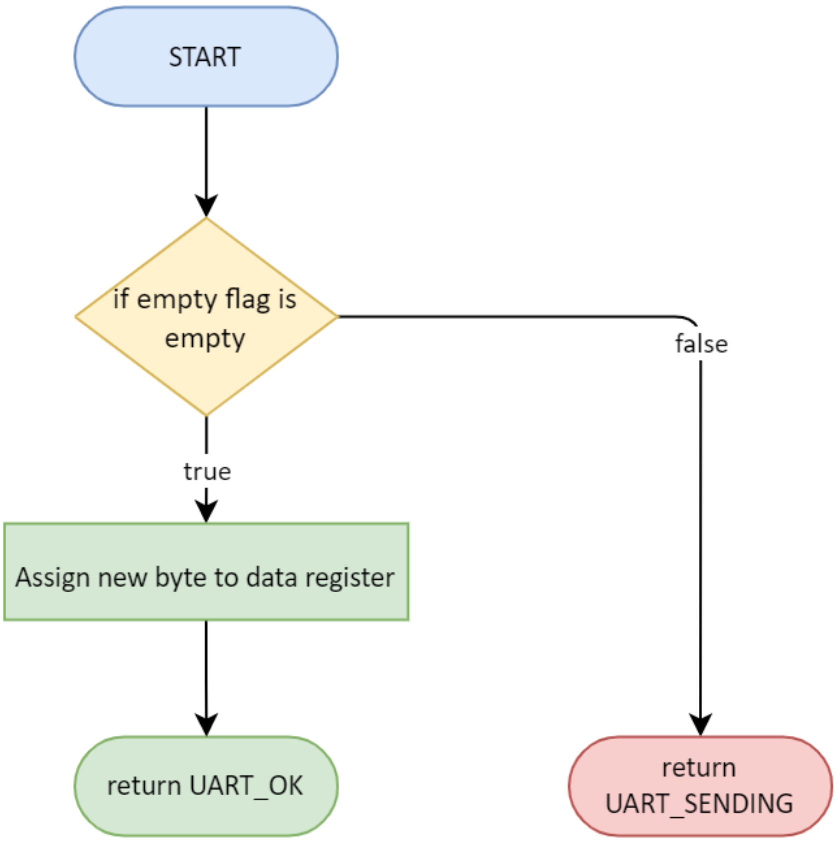
3.1.2.1. UART_init


START
Assign parity mode configured
Assign enable interrupts as configured
Enable channels used
Assign Asynch/Synch uart mode as configured
calculate UBBR value according configured baudrate and transmission speed
Assign parity mode configured
Assign stop bits configured
assign dataBits choosed
enable global interrupt
END

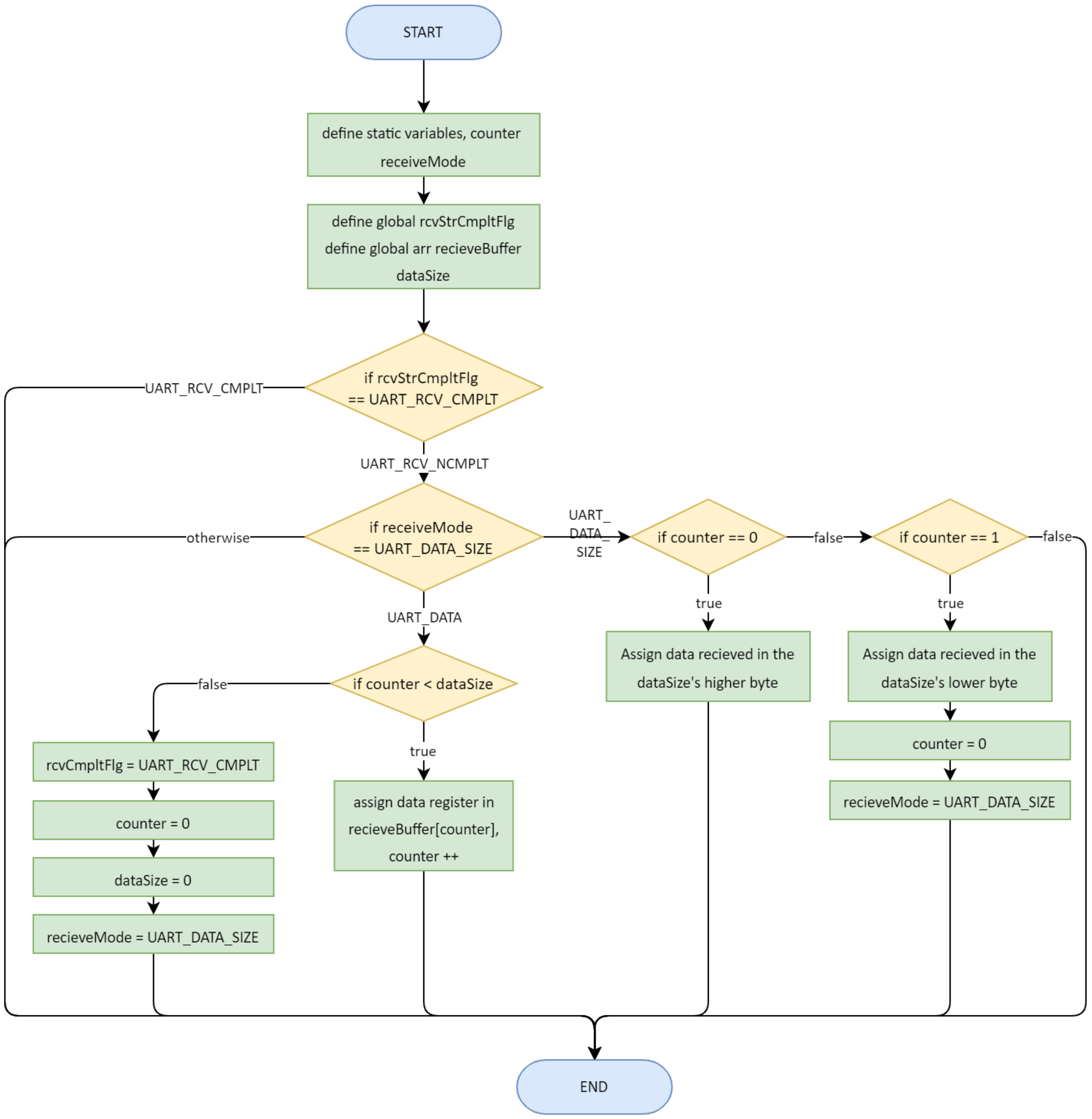
3.1.2.2. UART_deinit



3.1.2.3. UART_sendByte



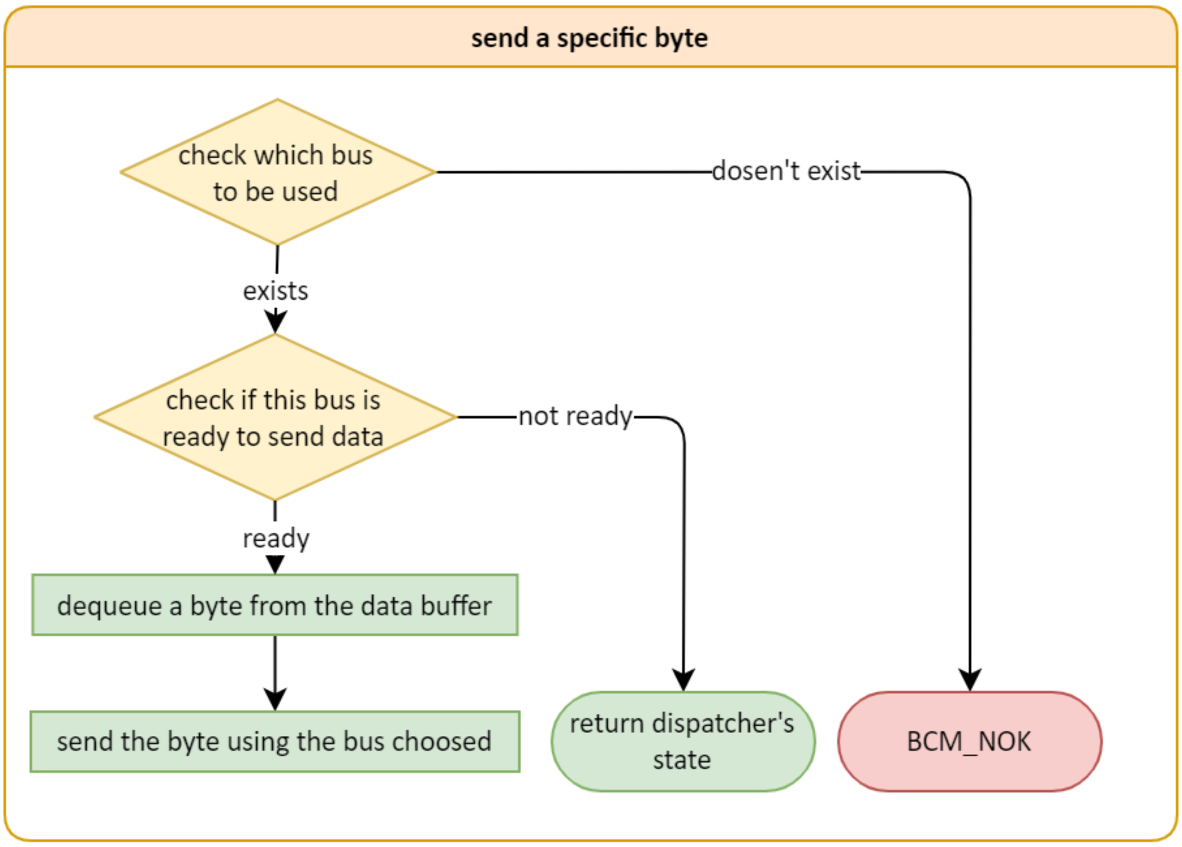
3.1.2.4. ISR(USART_RXC_INT)


START
define static variables, counter receiveMode
define global rcvStrCmpltFlg
define global arr recieveBuffer
dataSize
if rcvStrCmpltFlg == UART_RCV_CMPLT
UART_RCV_CMPLT
UART_RCV_NCMPLT
if receiveMode == UART_DATA_SIZE
otherwise
UART_DATA_SIZE
if counter == 0
false
if counter == 1
false
true
Assign data recieved in the dataSize's higher byte
true
Assign data recieved in the dataSize's lower byte
counter = 0
recieveMode = UART_DATA_SIZE
UART_DATA
if counter < dataSize
false
rcvCmpltFlg = UART_RCV_CMPLT
counter = 0
dataSize = 0
recieveMode = UART_DATA_SIZE
true
assign data register in recieveBuffer[counter], counter ++
END

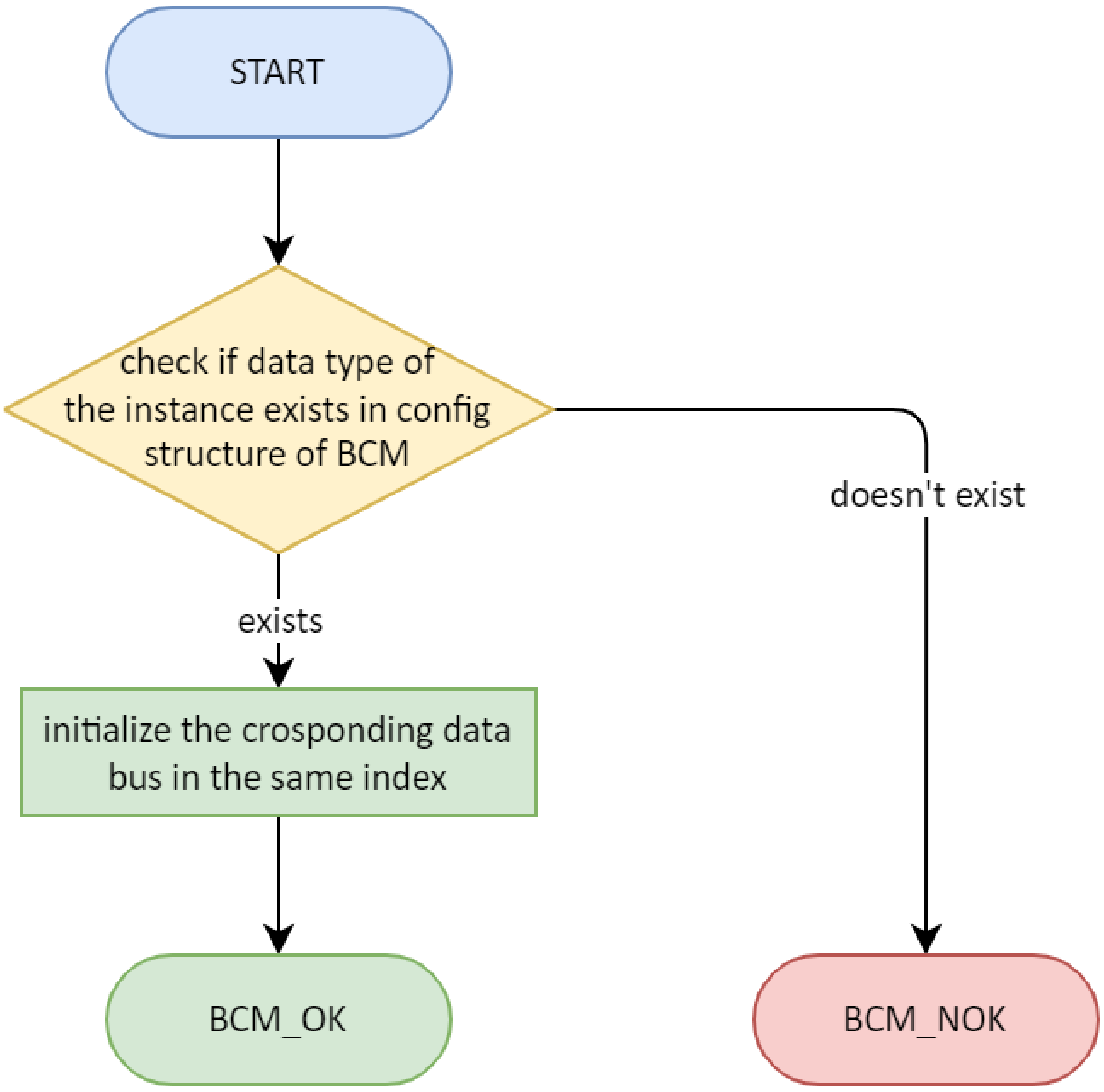
3.2. Server Layer

3.2.1. BCM

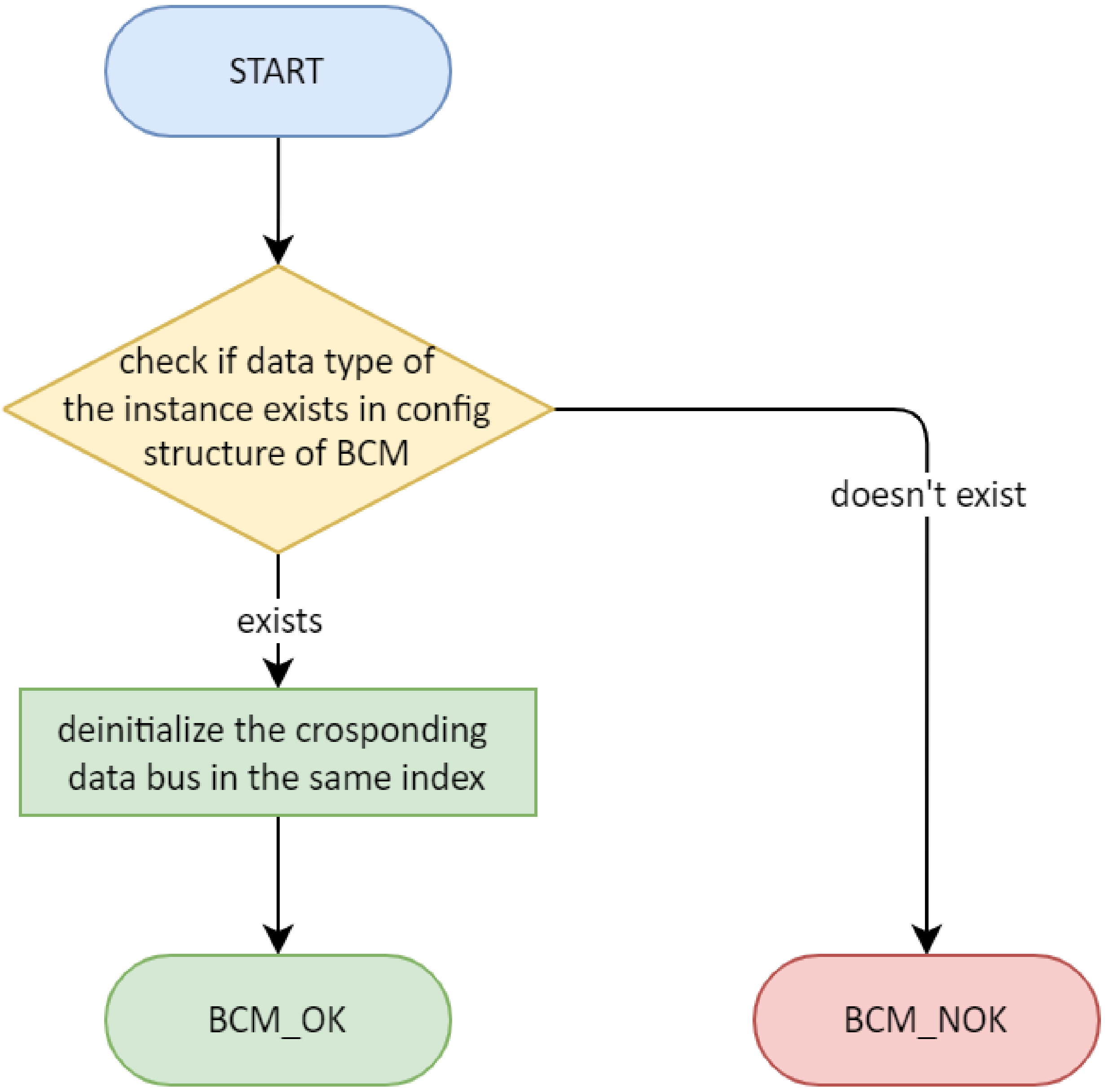
3.2.1.1. Send a specific byte process


send a specific byte
check which bus to be used
dosen't exist
exists
check if this bus is ready to send data
not ready
ready
dequeue a byte from the data buffer
send the byte using the bus choosed
return dispatcher's state
BCM_NOK

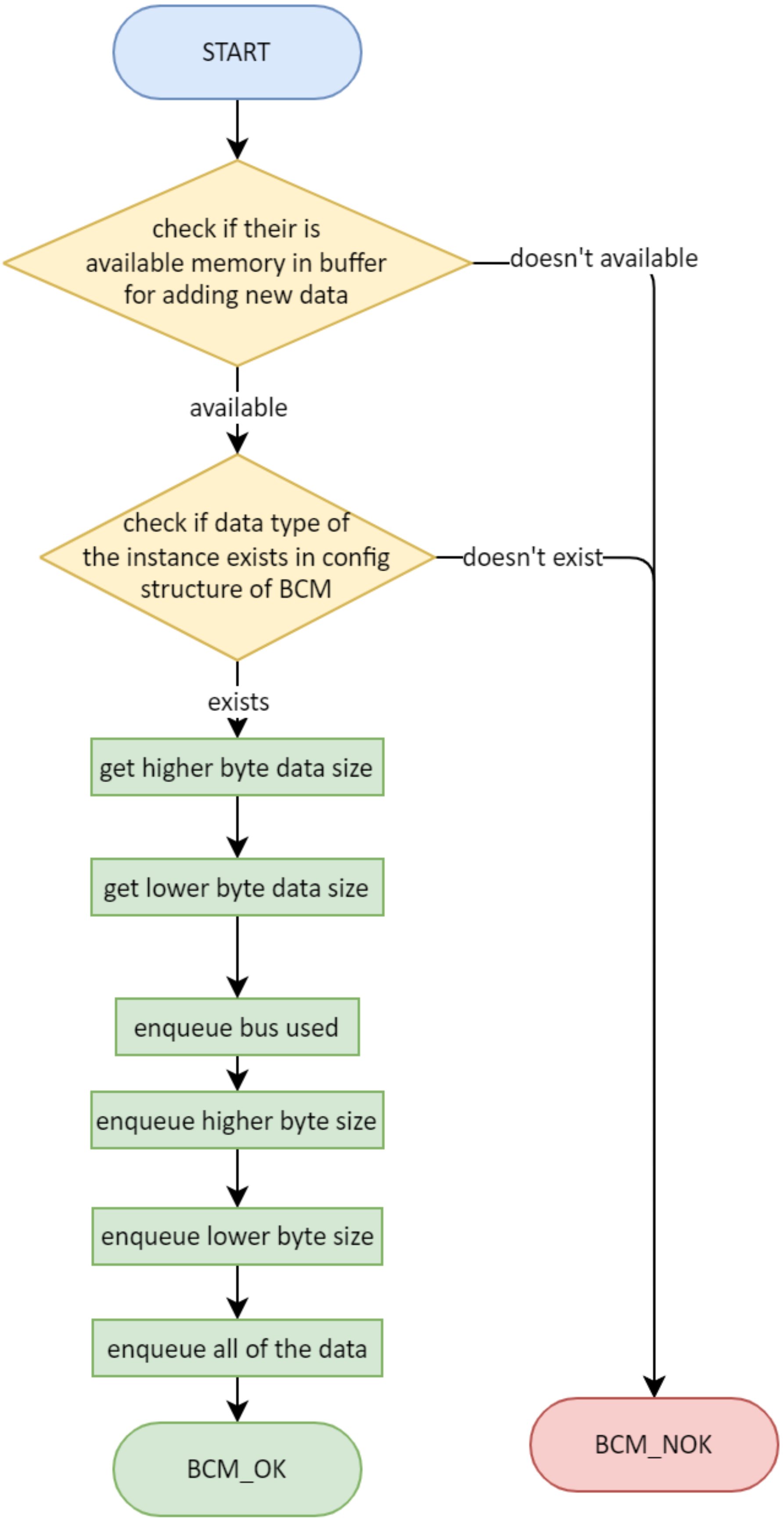
3.2.1.2. BCM_init


START
check if data type of the instance exists in config structure of BCM
doesn't exist
exists
initialize the crosponding data bus in the same index
BCM_OK
BCM_NOK

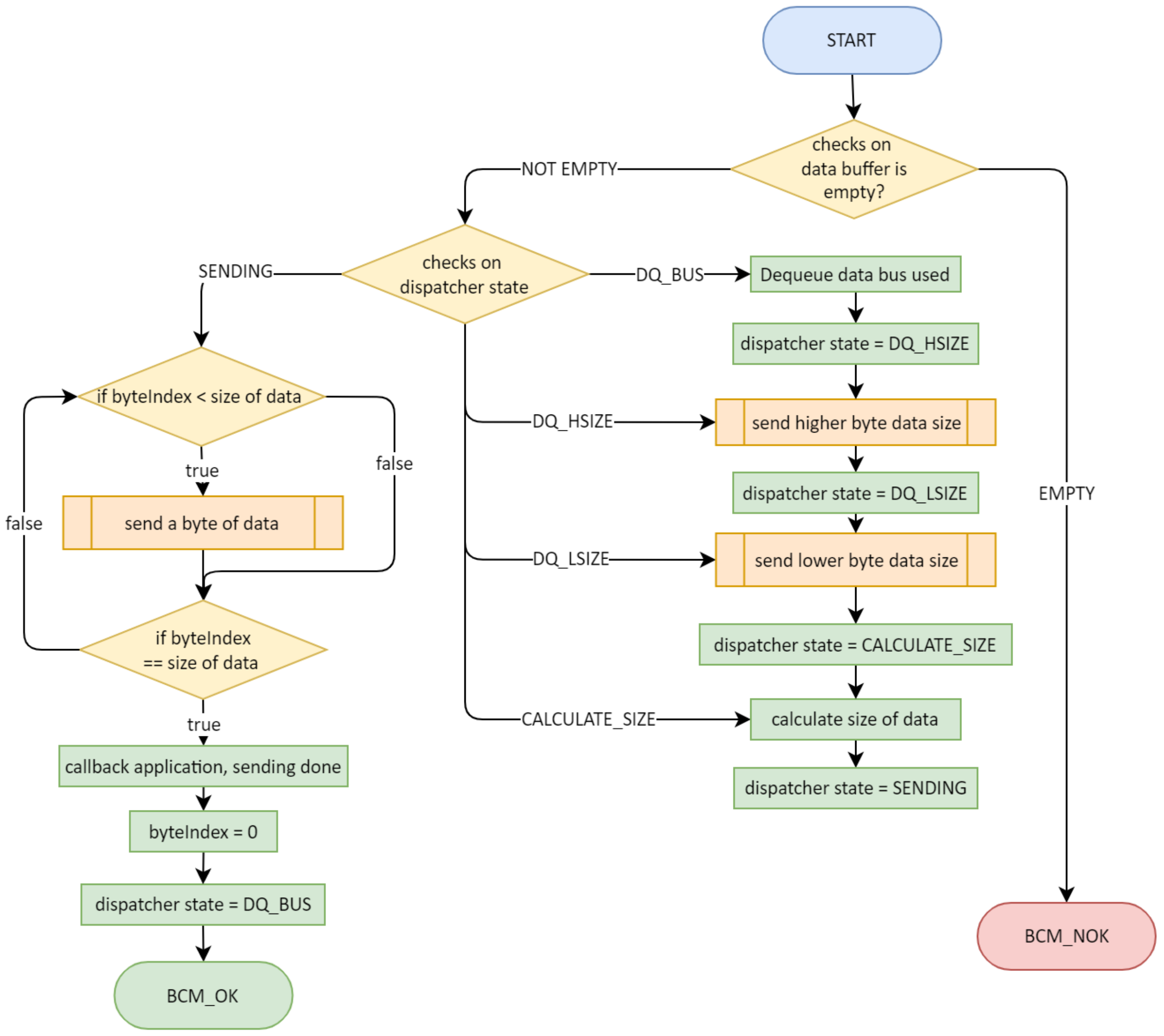
3.2.1.3. BCM_deinit


START
check if data type of the instance exists in config structure of BCM
doesn't exist
exists
deinitialize the crosponding data bus in the same index
BCM_OK
BCM_NOK

3.2.1.4. BCM_send_n



3.2.1.5. BCM_dispatcher


START
checks on data buffer is empty?
NOT EMPTY
EMPTY
checks on dispatcher state
SENDING
DQ_BUS
DQ_HSIZE
DQ_LSIZE
CALCULATE_SIZE
Dequeue data bus used
dispatcher state = DQ_HSIZE
send higher byte data size
dispatcher state = DQ_LSIZE
send lower byte data size
dispatcher state = CALCULATE_SIZE
calculate size of data
dispatcher state = SENDING
if byteIndex < size of data
true
false
send a byte of data
false
if byteIndex == size of data
true
callback application, sending done
byteIndex = 0
dispatcher state = DQ_BUS
BCM_OK
BCM_NOK