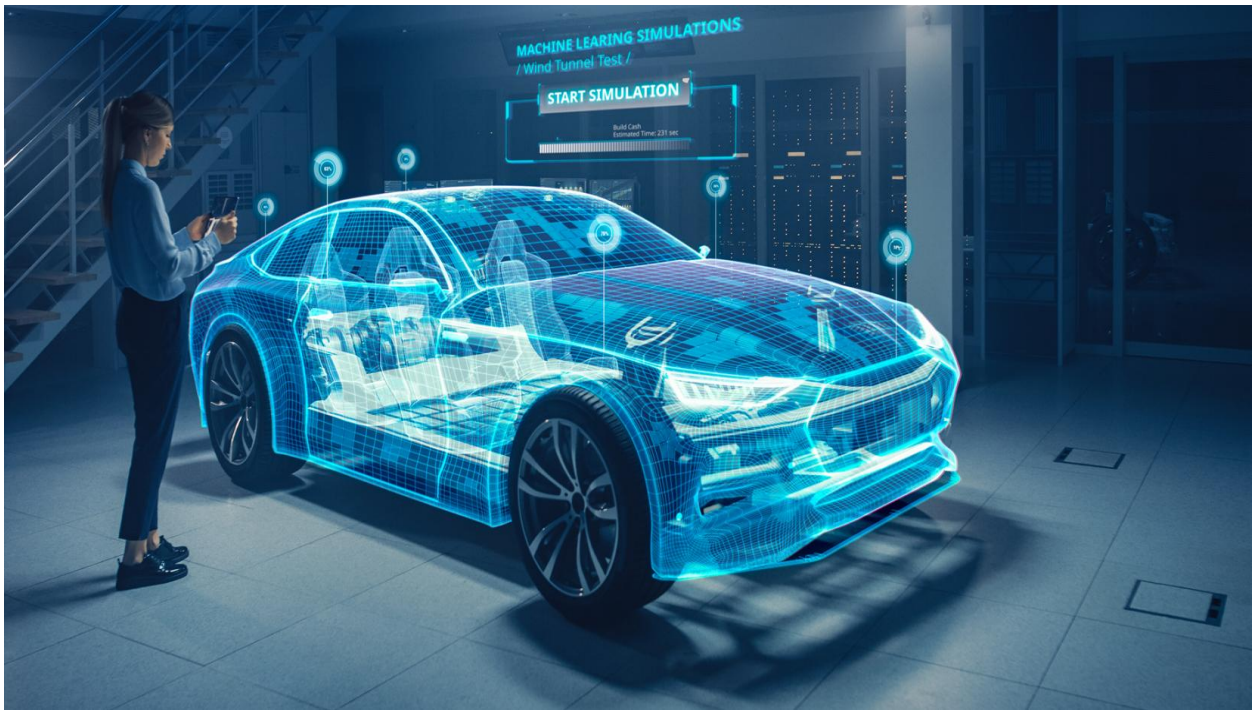


Basic Communication Manager Design

Peter Essam
Brightskies Company

Table of Content :

Introduction

BCM (Basic Communication Manager) This module has the capability to work with different serial communication protocols using ISR with the highest possible throughput.

Detailed Requirements

1. Specifications
    - The BCM has the capability to send and receive any data with maximum length of 65535 bytes (Maximum of unsigned two bytes variable).
    - It can use any communication protocol with the support of Send, Receive or both.
    - Implement bcm_Init use the below table. This function will initialize the corresponding serial communication protocol.
    - Implement bcm_deinit use the below table. This function will uninitialize the corresponding BCM instance, (instance: is the communication channel).
    - Implement bcm_send that will send only 1 byte of data over a specific BCM instance
    - Implement bcm_send_n will send more than one byte with a length n over a specific BCM instance
    - Implement bcm_dispatcher will execute the periodic actions and notifies the user with the needed events over a specific BCM instance

1.bcm_init

Function Name	bcm_init
Syntax	enu_bcm_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 of the BCM Instance
Parameters (out):	None
Parameters (in, out):	None
Return:	2 (NULL_POINTER) 1 (CHANNEL_ERROR) 0 (BCM_OKAY)

2.bcm_deinit

Function Name	bcm_deinit
Syntax	enu_bcm_status_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:	2 (NULL_POINTER) 0 (BCM_OKAY)

3. bcm_send

Function Name	bcm_send
Syntax	enu_bcm_status_t bcm_send(str_bcm_instance_t* ptr_str_bcm_instance, u8_t u8_arg_byte);
Sync/Async	Asynchronous
Reentrancy	Reentrant
Parameters (in):	ptr_str_bcm_instance: Address of the BCM Instance u8_arg_byte : byte
Parameters (out):	None
Parameters (in, out):	None
Return:	2 (NULL_POINTER) 1 (CHANNEL_ERROR) 0 (BCM_OKAY)

4. bcm_send_n

Function Name	bcm_send_n
Syntax	enu_bcm_status_t bcm_send_n (str_bcm_instance_t* ptr_str_bcm_instance, u8_t *ptr_arg_bytes, u8_t u8_arg_size);
Sync/Async	Asynchronous
Reentrancy	Reentrant
Parameters (in):	ptr_str_bcm_instance: Address of the BCM Instance ptr_arg_byte : Address of the array of bytes u8_arg_size : size of array
Parameters (out):	None
Parameters (in, out):	None
Return:	2 (NULL_POINTER) 1 (CHANNEL_ERROR) 0 (BCM_OKAY)

5. bcm_dispatcher

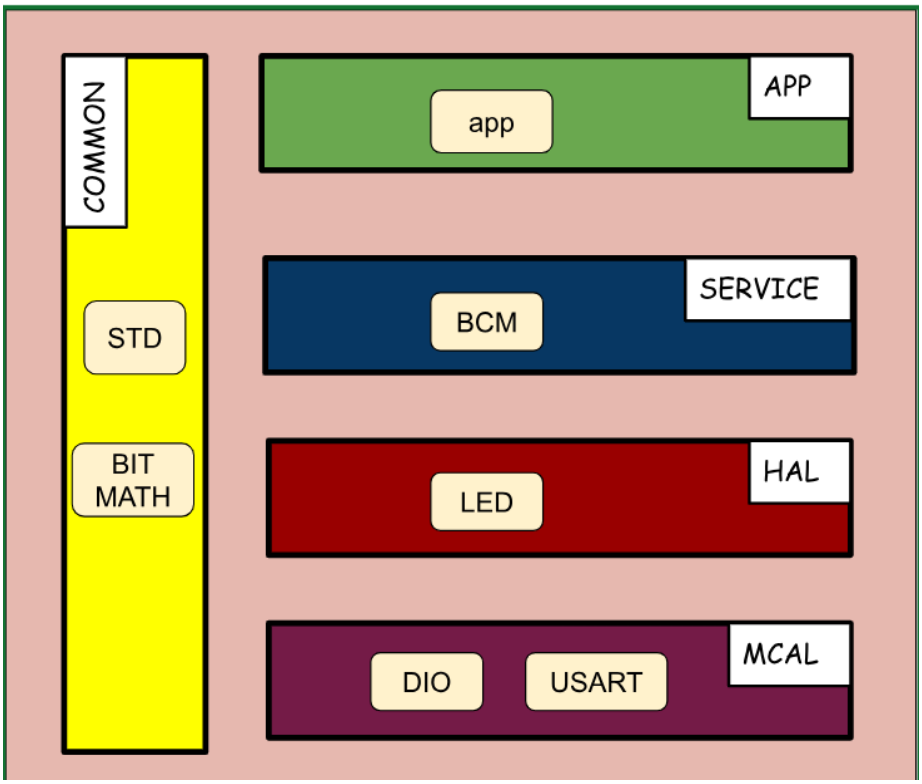
Function Name	bcm_dispatcher
Syntax	enu_bcm_status_t bcm_dispatcher (str_bcm_instance_t* ptr_str_bcm_instance);
Sync/Async	Synchronous
Reentrancy	Non Reentrant
Parameters (in):	ptr_str_bcm_instance: Address of the BCM Instance
Parameters (out):	None
Parameters (in, out):	None
Return:	3(SEND_OPERATION_DONE) 4 (REC_OPERATION_DONE) 0 (BCM_OKAY)

2. Module Testing
    - Send [BCM Operating] string from MCU_1 to MCU_2.
    - When MCU_1 finishes sending, LED_0 in MCU_1 will be toggled.
    - When MCU_2 finishes receiving the [BCM Operating] string, LED_1 in MCU_2 will be toggled.
    - When MCU_2 finishes sending, LED_0 in MCU_2 will be toggled.
    - When MCU_1 finishes receiving the [BCM Operating] string, LED_1 in MCU_1 will be toggled.

High Level Design

1. Layered Architecture

    I use Five Layers (MCAL , HAL , SERVICE , APP , COMMON)



2. Modules Descriptions

    - **Dio :**

Stands for Digital Input/Output. It is an interface component that allows the system to send digital signals to devices. Also read signals from others.

- **Usart :**

The Universal Synchronous Asynchronous Receiver Transmitter (USART) module is one of the serial I/O modules for communication interfacing functions with other devices/units.

- **Led :**

This Module Controls Leds state in the program

- **Bcm :**

Manages Communication between program and different communication channels.

- **App :**

Contain Applicatoin Logic.

3. Drivers Documentations

- **Dio**

Description : This function initialize PIN and set direction

ARGS : take PIN Number and PORT Number and Direction (INPUT,OUTPUT)

return : return DIO_OK if the PIN initializes correctly, DIO_NOT_OK otherwise

EN_DIO_ERROR DIO_init (EN_DIO_PINS pinNumber,EN_DIO_PORTS portNumber,EN_DIO_DIRECTION direction);

Description : This function write on PIN and set it's level

ARGS : take PIN Number and PORT Number and level (LOW,HIGH)

return : return DIO_OK if the PIN level sets correctly, DIO_NOT_OK otherwise

EN_DIO_ERROR DIO_write(EN_DIO_PINS pinNumber,EN_DIO_PORTS portNumber,EN_DIO_LEVEL level);

Description : This function toggles PIN level

ARGS : take PIN Number and PORT Number

return : return DIO_OK if the PIN toggles correctly, DIO_NOT_OK otherwise

EN_DIO_ERROR DIO_toggle (EN_DIO_PINS pinNumber,EN_DIO_PORTS portNumber);

Description : This function reads PIN level and store it in the variable

ARGS : take PIN Number and PORT Number and pointer to the variable

return : return DIO_OK if the PIN value stored correctly , DIO_NOT_OK otherwise

EN_DIO_ERROR DIO_read (EN_DIO_PINS pinNumber,EN_DIO_PORTS portNumber, u8_t * value);

- **Usart**

Description : This function inits Usart to operate on specfic mode look at usart.configs

ARGS : channel id

return : return STATUS_OK if the module initialized correctly , CONFIG_ERROR , CHANNEL_NOT_FOUND otherwise

en_usart_error_code_t USART_init (u8_t u8_arg_channel_id);

Description : This function set byte in the queue to be sent

ARGS : byte to be send

return : return STATUS_OK if the byte sent to queue correctly , QUEUE_OVERFLOW otherwise

en_usart_error_code_t USART_send_byte(u8_t u8_arg_byte);

Description : This function set n of bytes in the queue to be sent

ARGS : pointer to array of bytes

return : return STATUS_OK if the bytes sent to queue correctly , QUEUE_OVERFLOW otherwise

en_usart_error_code_t USART_send_n_bytes(u8_t *u8_arg_arr_bytes,u8_t u8_arg_arr_size);

Description : This function set call back function to specific pointer

ARGS : pointer to function and state(send/receive)

return : return STATUS_OK if the bytes sent to queue correctly , CALL_BACK_ERROR otherwise

En_usart_error_code_t USART_setCallBack(en_usart_operating_state_t en_usart_operating_state , void(*ptr_func)(void));

- **Led**

Description : This function inits led as output

ARGS : pointer to struct (pin/port)

return : return LED_OK if the Led initialized correctly , LED_NOT_OKAY otherwise

enu_led_error_t LED_init(str_led_config_t *str_ptr_led_config);

Description : This function sent High to pin

ARGS : pointer to struct (pin/port)

return : return LED_OK if the Led turns high correctly , LED_NOT_OKAY otherwise

enu_led_error_t LED_on(str_led_config_t *str_ptr_led_config);

Description : This function sent Low to pin

ARGS : pointer to struct (pin/port)

return : return LED_OK if the Led turns Low correctly , LED_NOT_OKAY otherwise

enu_led_error_t LED_off(str_led_config_t *str_ptr_led_config);

Description : This function toggle pin state

ARGS : pointer to struct (pin/port)

return : return LED_OK if the Led toggled correctly , LED_NOT_OKAY otherwise

enu_led_error_t LED_toggle(str_led_config_t *str_ptr_led_config);

- **Bcm**

Description : checks if the en_bcm_channel member of the input ptr_str_bcm_instance matches the en_bcm_channel member of one of the str_bcm_instance structures at the current index uint8_loc_counter. If there is a match, it calls three functions using function pointers stored in the str_bcm_functions_pointer structure of the matching str_bcm_instance. These functions are ptr_func_init , ptr_func_setCall with USART_SEND_STATE , and ptr_func_setCall with USART_RECEIVE_STATE

ARGS : pointer to struct

return : enu_bcm_status_t status code.

enu_bcm_status_t bcm_init (str_bcm_instance_t * ptr_str_bcm_instance)

Description : This function seems to be deinitializing a BCM (Broadcom) instance by setting its en_bcm_channel to NULL, provided that the input pointer is valid and the en_bcm_channel member is not already NULL. It returns a status code to indicate whether the deinitialization was successful or if there were any errors, such as a null pointer or an already NULL en_bcm_channel.T

ARGS : pointer to struct

return : enu_bcm_status_t status code.

enu_bcm_status_t bcm_deinit (str_bcm_instance_t * ptr_str_bcm_instance)

Description : is meant to send data using a BCM instance, and it takes an instance pointer (ptr_str_bcm_instance) and a byte of data (u8_arg_byte) as input.

The function will perform some operations specific to the BCM instance, likely related to sending data.

ARGS : pointer to struct , u8 byte

return : enu_bcm_status_t status code.

enu_bcm_status_t bcm_send (str_bcm_instance_t * ptr_str_bcm_instance , u8_t u8_arg_byte)

ARGS : pointer to struct

return : enu_bcm_status_t status code.

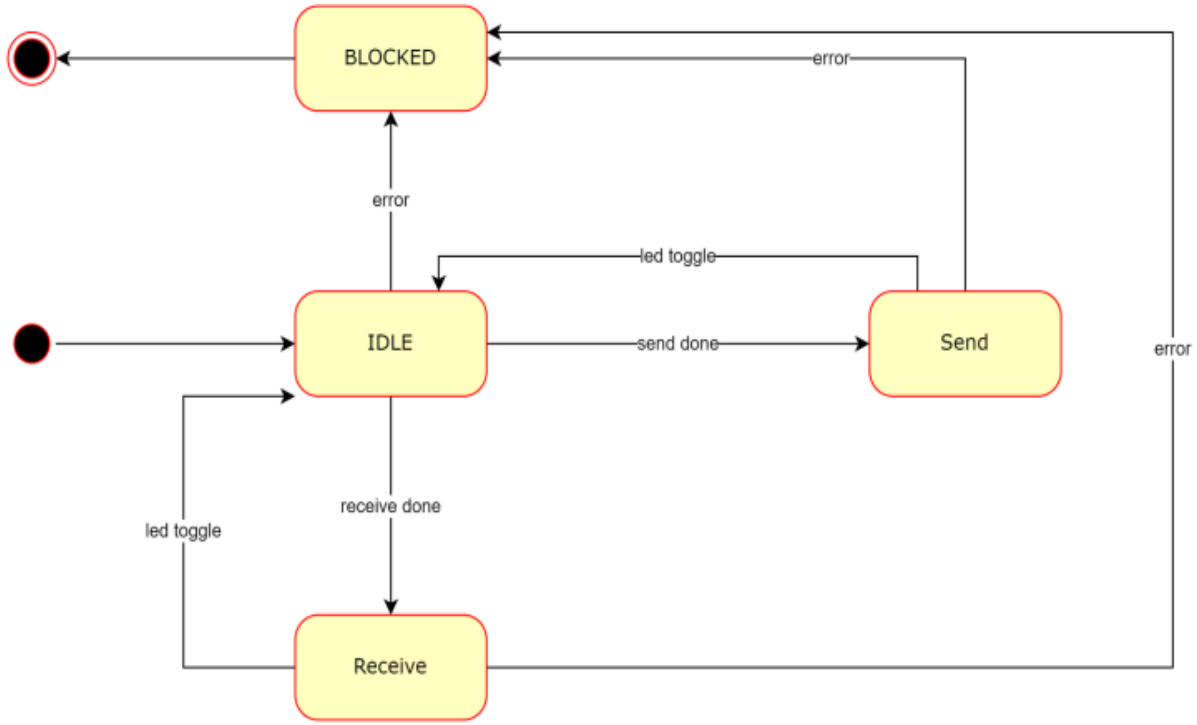
enu_bcm_status_t bcm_dispatcher (str_bcm_instance_t * ptr_str_bcm_instance_t)

To look more details on BCM look Pages : 2 , 3 , 4

---

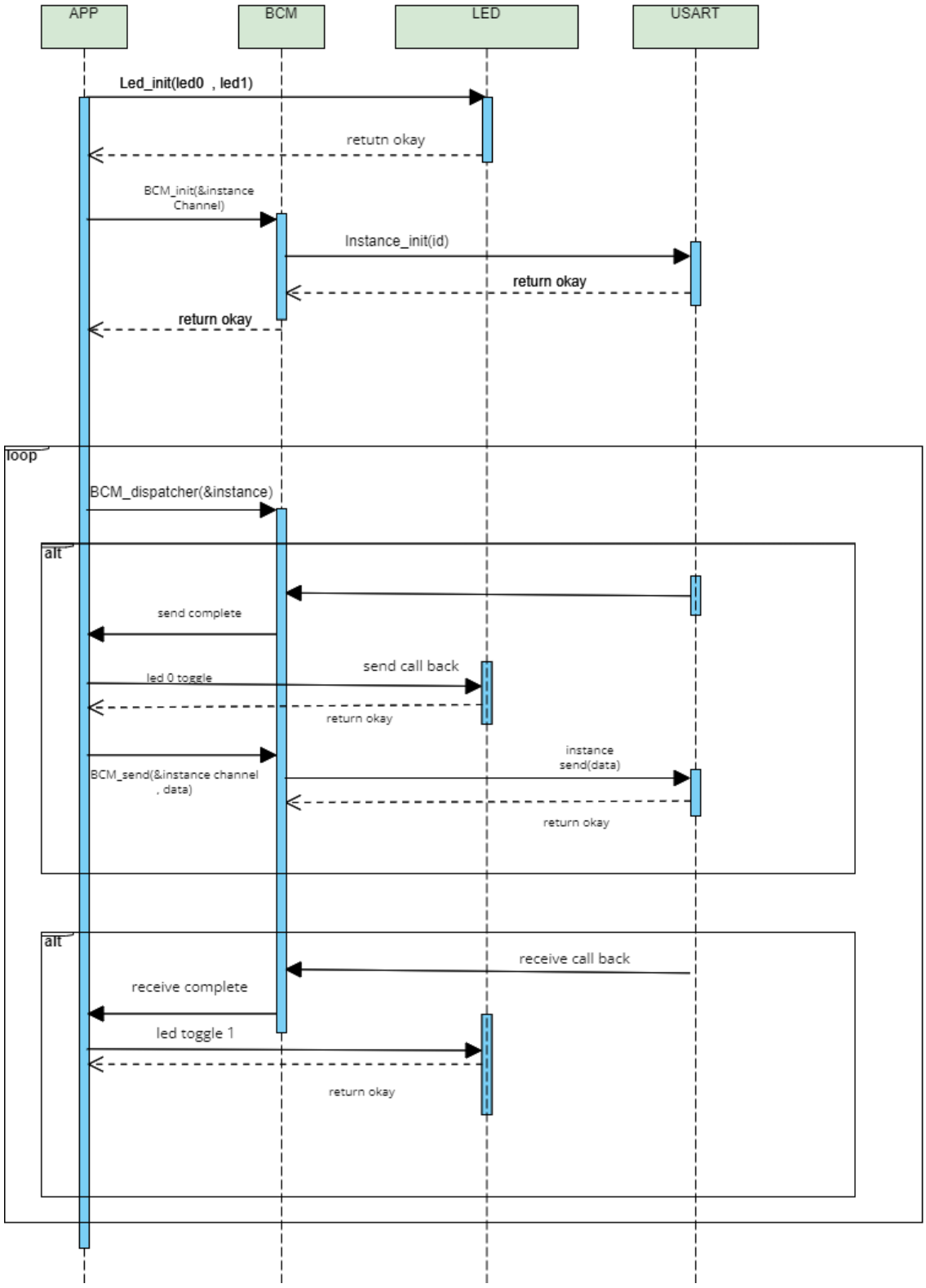
4. UML

- **State Machine**

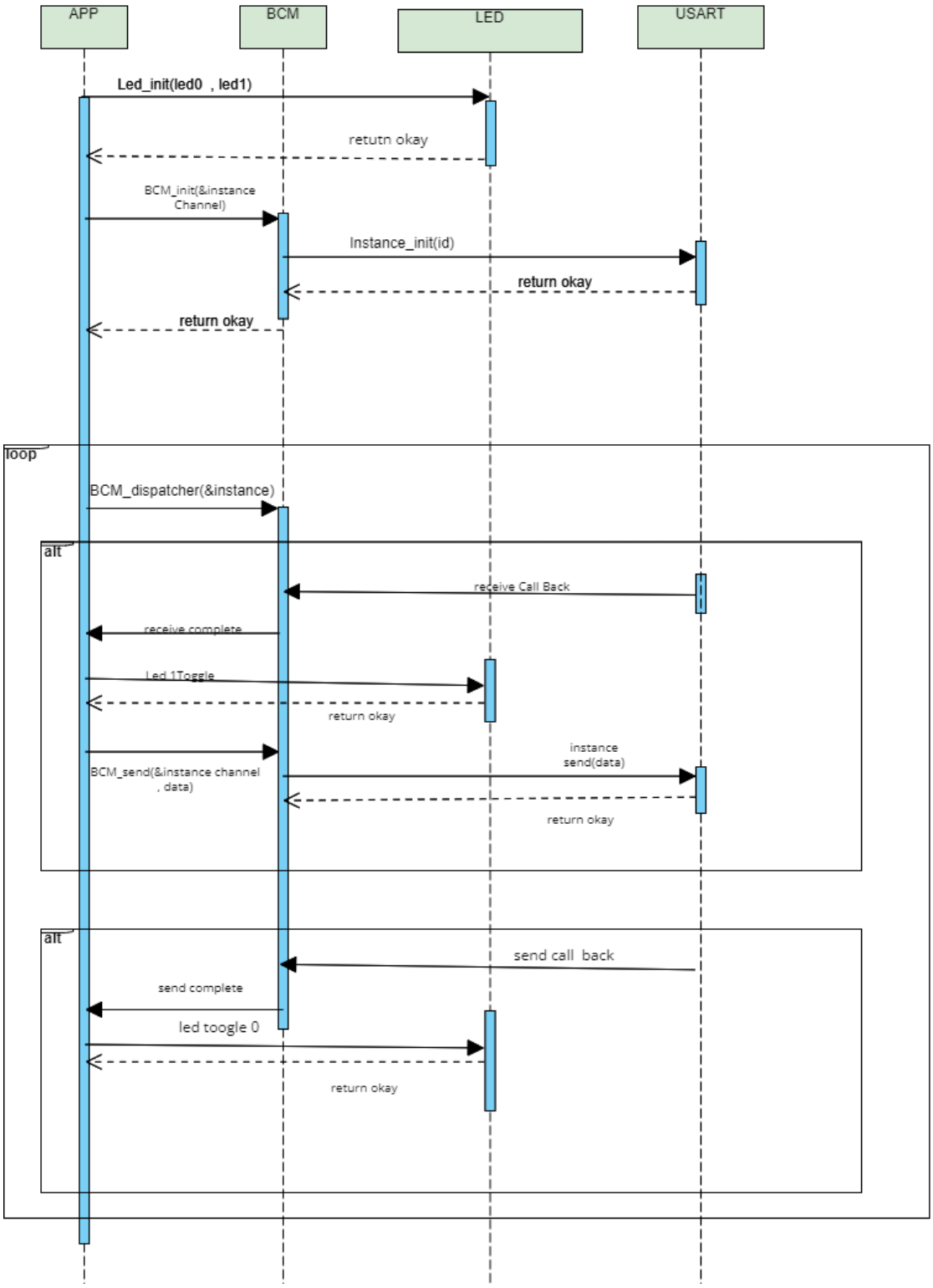


5. Sequence Diagram

- **MCU _ 1**

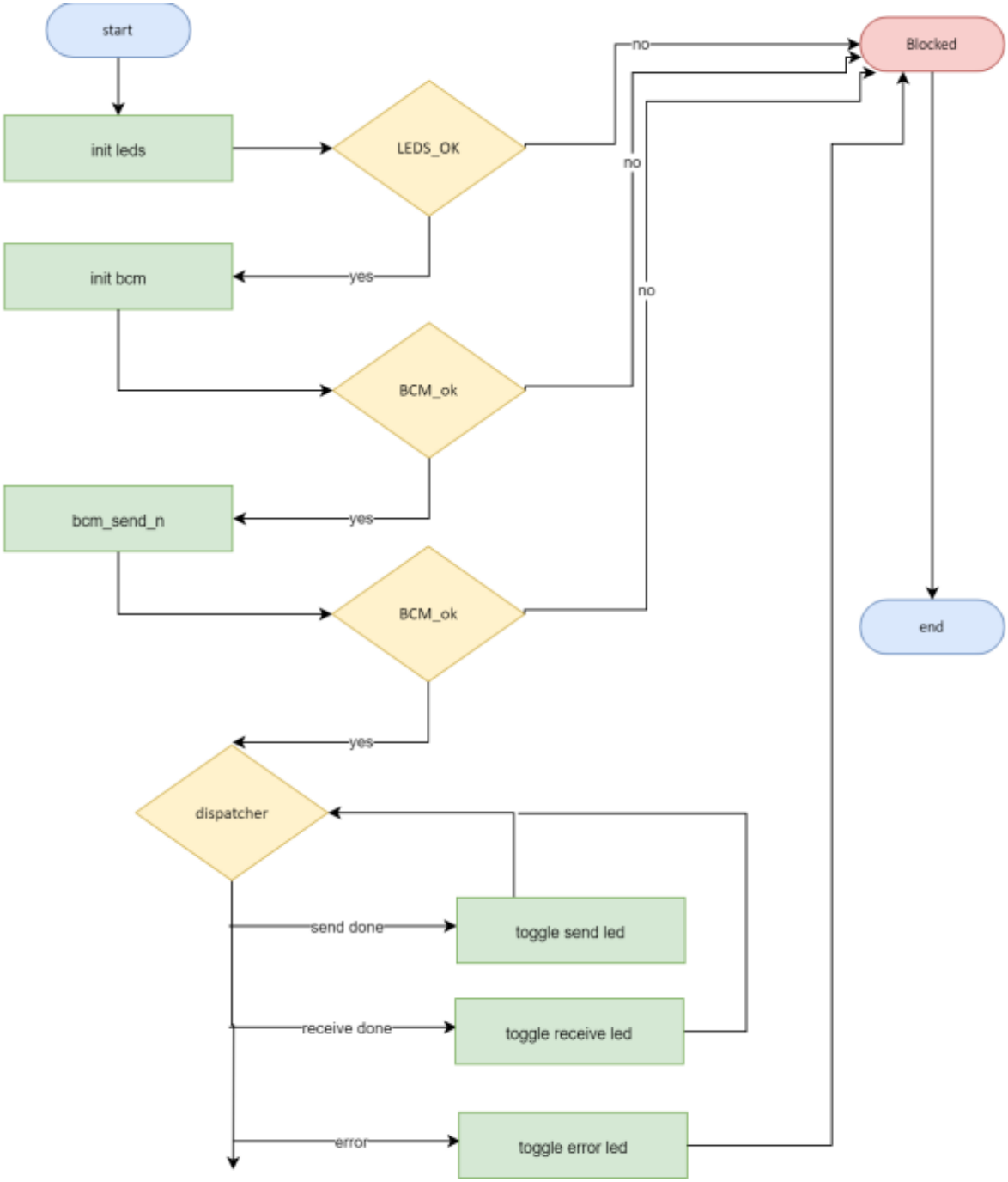

APP
BCM
LED
USART
Led_init(led0 , led1)
retutn okay
BCM_init(&instance Channel)
Instance_init(id)
return okay
return okay
loop
BCM_dispatcher(&instance)
alt
send complete
send call back
led 0 toggle
return okay
instance send(data)
BCM_send(&instance channel , data)
return okay
alt
receive call back
receive complete
led toggle 1
return okay

- **MCU _ 2**


APP
BCM
LED
USART
Led_init(led0 , led1)
retutn okay
BCM_init(&instance Channel)
Instance_init(id)
return okay
return okay
loop
BCM_dispatcher(&instance)
alt
receive Call Back
receive complete
Led 1Toggle
return okay
instance send(data)
BCM_send(&instance channel , data)
return okay
alt
send call back
send complete
led toogle 0
return okay

Low Level Design Design

1. Flowchart App



2. Pre_Compiling Configuration

- **Usart**

```
#define F_CPU                                8000000UL
#define BAUDRATE                             9600
#define BAUD_PRESCALLER                      ((F_CPU/(16UL*BAUDRATE))-1)
#define BAUD_PRESCALLER_DOUBLE_SPEED ((F_CPU/(8UL*BAUDRATE))-1)
#define USART_NORMAL_SPEED                   0
#define USART_DOUBLE_SPEED                   1
#define USART_ENABLE_INTERRUPT               0
#define USART_DISABLE_INTERRUPT              1
#define USART_CHANNELS                       2
#define TASKS_MAX_SIZE                       200
#define USART_SPEED_SELECT                   USART_NORMAL_SPEED
#define USART_INTERRUPT_OPTION               USART_ENABLE_INTERRUPT
```

3. Linking Configuration

- **USART**

```
const str_usart_configs_t str_gl_usart_arr_configs[USART_CHANNELS] =
{
    {
        .uint8_channel_id = 0,
        .en_usart_set_mode = USART_ASYNC_MODE,
        .en_usart_operating_state = USART_FULL_DUBLEX_STATE,
        .en_usart_parity_select = USART_DIS_PARITY,
        .en_usart_stop_bit_select = USART_ONE_STOP_BIT,
        .en_usart_data_size_select = USART_DATA_SIZE_8,
    },
    {
        .uint8_channel_id = 1,
```

```
        .en_usart_set_mode = USART_ASYNC_MODE,
        .en_usart_operating_state = USART_SEND_STATE,
        .en_usart_parity_select = USART_EVEN_PARITY,
        .en_usart_stop_bit_select = USART_TWO_STOP_BITS,
       .en_usart_data_size_select = USART_DATA_SIZE_8,
    }
};
```

- **Bcm**

```
const str_bcm_instance_t str_bcm_instance[BCM_INSTANCES] =
{
        {
       .en_bcm_comm_type = BCM_USART,
        .en_bcm_channel = CHANNEL_0,
        .str_bcm_functions_pointer.ptr_func_init = USART_init,
       .str_bcm_functions_pointer.ptr_func_send = USART_send_byte,
       .str_bcm_functions_pointer.ptr_func_send_n = USART_send_n_bytes,
       .str_bcm_functions_pointer.ptr_func_setCall = USART_setCallBack
        },
        {
        .en_bcm_comm_type = BCM_SPI
       , .en_bcm_channel = CHANNEL_1,
       //.str_bcm_functions_pointer.ptr_func_init = SPI_init;
       }
 };
```

Thanks