



Basic Communication Manager Design

By:
Nadeen Adel

Table Of Content:

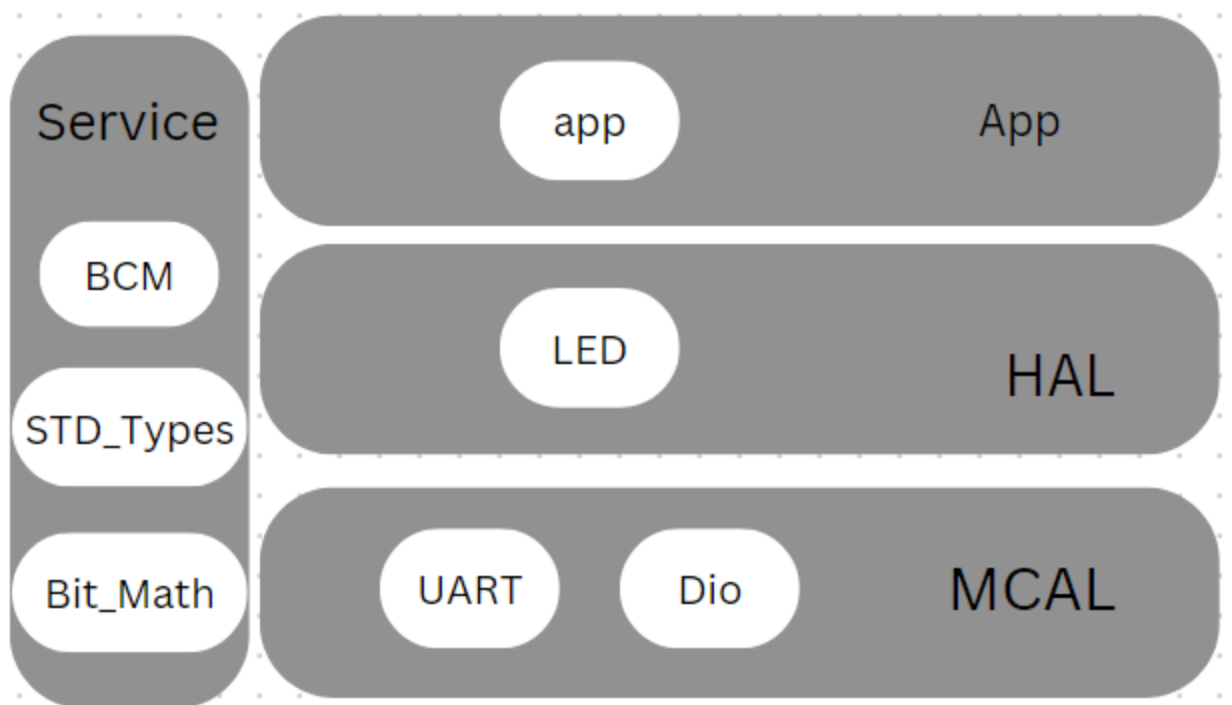
Table Of Content:	1
1. Introduction:	2
1. High Level Design:	3
2.1 Layered Architecture:.....	3
2.2 Modules Description:.....	3
2.3 Drivers' Documentation:.....	4
2.3.1 DIO:.....	4
2.3.2 UART:.....	5
2.3.3 LED:.....	5
3. Low Level Design:	6
3.1 Flowcharts:.....	6
3.2 Configurations:.....	9
3.2.1 DIO:.....	9
3.2.3 UART:.....	12
3.2.3 LED:.....	16
BCM APIs:.....	17
UML:	20
BCM State Machine:	20
BCM Sequence Diagram:	21

1. Introduction:

This project involves designing essential functions and APIs (Application Programming Interfaces) for the BCM, ensuring its initialization, proper termination, single-byte data transmission, and multi-byte data transfer capabilities. The BCM's functionalities will be managed through a series of well-defined functions that follow the specifications provided in the tables. These functions aim to make the BCM easy to integrate into a wide range of embedded systems, enabling developers to harness the full potential of this versatile communication module.

1. High Level Design:

2.1 Layered Architecture:



2.2 Modules Description:

- **DIO:**
controls GPIO pins.
- **UART:**
to send / receive (communicate) with other MCUs .

- **LED:**

controls led state in the program.

- **BCM:**

communication manager that manages communicating between different MCUs.

- **App:**

Contains main logic of the code.

2.3 Drivers' Documentation:

2.3.1 DIO:

```

/*
 * Initializes a specific digital pin based on the provided configuration.
 * @param config_ptr: Pointer to the configuration structure for the pin.
 * @return: function error state.
 */
EN_dioError_t DIO_Initpin(ST_DIO_ConfigType *config_ptr);

/*
 * Writes a digital value (HIGH or LOW) to a specific digital pin on a given port.
 * @param port: Port to which the pin belongs.
 * @param pin: Specific pin to write to.
 * @param value: Value to be written (HIGH or LOW).
 * @return: function error state.
 */
EN_dioError_t DIO_WritePin(EN_dio_port_t port, EN_dio_pin_t pin, EN_dio_value_t value);

/*
 * Reads the digital value from a specific digital pin on a given port and stores it in the specified location.
 * @param port: Port from which the pin should be read.
 * @param pin: Specific pin to read.
 * @param value: Pointer to store the read value.
 * @return: function error state.
 */
EN_dioError_t DIO_read(EN_dio_port_t port, EN_dio_pin_t pin, u8 *value);

/*
 * Toggles the state of a specific digital pin on a given port.
 * @param port: Port to which the pin belongs.
 * @param pin: Specific pin to toggle.
 * @return: function error state.
 */
EN_dioError_t DIO_toggle(EN_dio_port_t port, EN_dio_pin_t pin);

```

2.3.2 UART:

```

/*this function initailizes uart
return: error state of the UART module*/
enu_uartErrorState_t MUART_init(const ST_USART_CONFIG *config);

/*this function is used to send a byte of data via UART
return: error state of the UART module*/
enu_uartErrorState_t MUART_sendByte(u8 u8_a_data);

/*this function is used to receive a byte of data via UART
return: error state of the UART module*/
enu_uartErrorState_t MUART_receiveByte(u8* pdata);

/*this function is used to send a string of data via UART
return: error state of the UART module*/
enu_uartErrorState_t MUART_sendString(u8* pdata);

```

2.3.3 LED:

```

/*Enum for error state*/
typedef enum
{
    LED_OK,
    LED_NOK
}EN_ledError_t;

/*struct to store led attributes*/
typedef struct LEDS{
    u8 port;
    u8 pin;
    u8 state;
}LEDS;

/*initializes led according to given arguments */
EN_ledError_t HLED_init(LEDS *led);

/*function to turn the LED on*/
EN_ledError_t HLED_on(LEDS *led);

/*function to turn the LED off*/
EN_ledError_t HLED_off(LEDS *led);

/*function to toggle the LED state*/
EN_ledError_t HLED_toggle(LEDS *led);

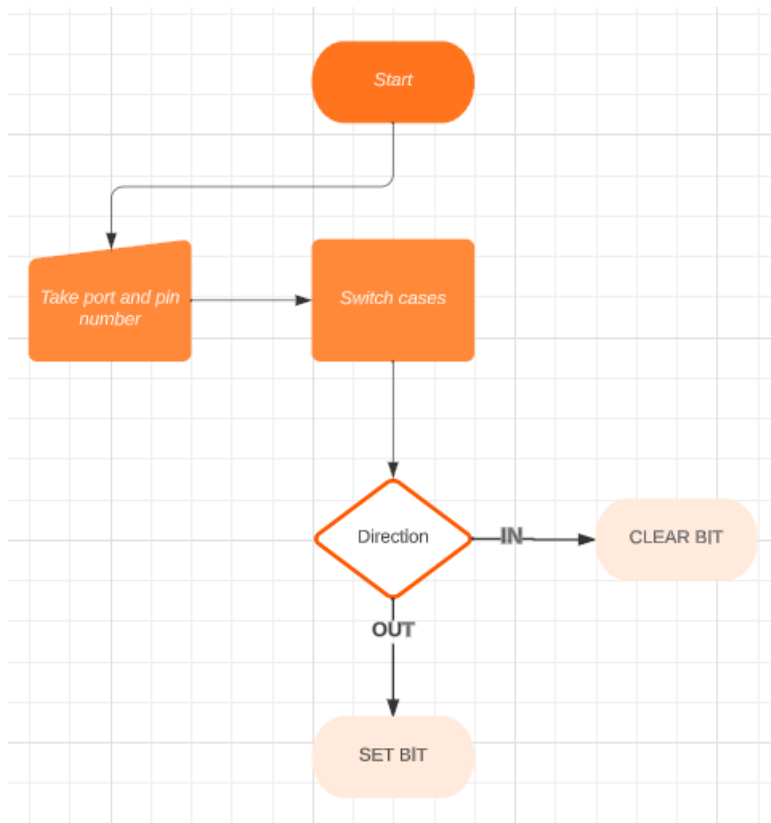
```

3. Low Level Design:

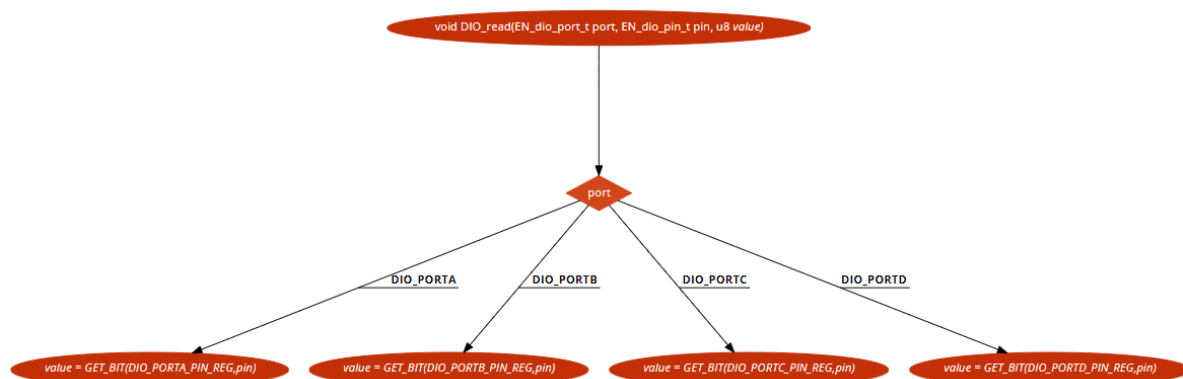
3.1 Flowcharts:

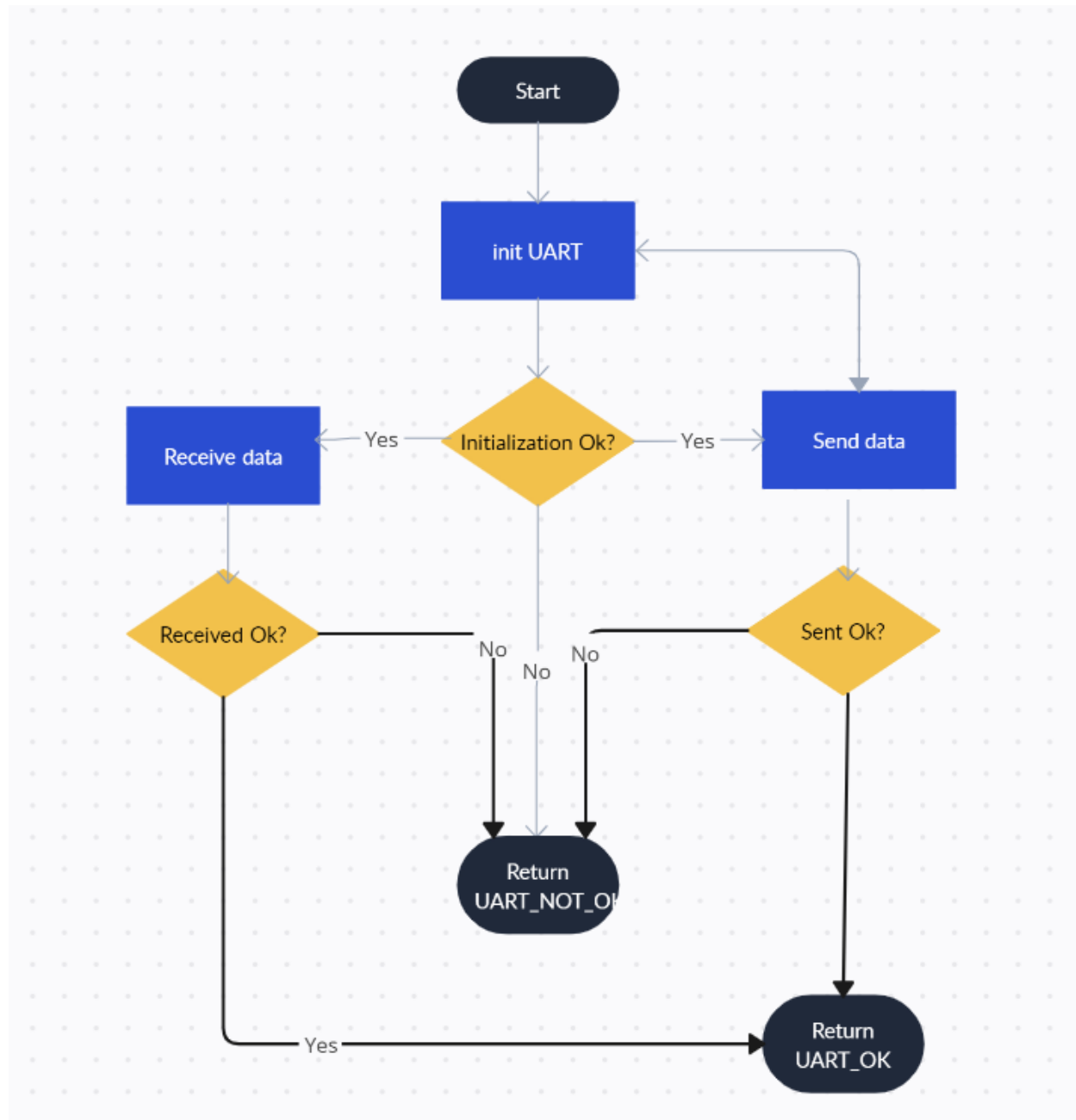
DIO:

EN_dioError_t DIO_WritePin(EN_dio_port_t port, EN_dio_pin_t pin, EN_dio_value_t value)

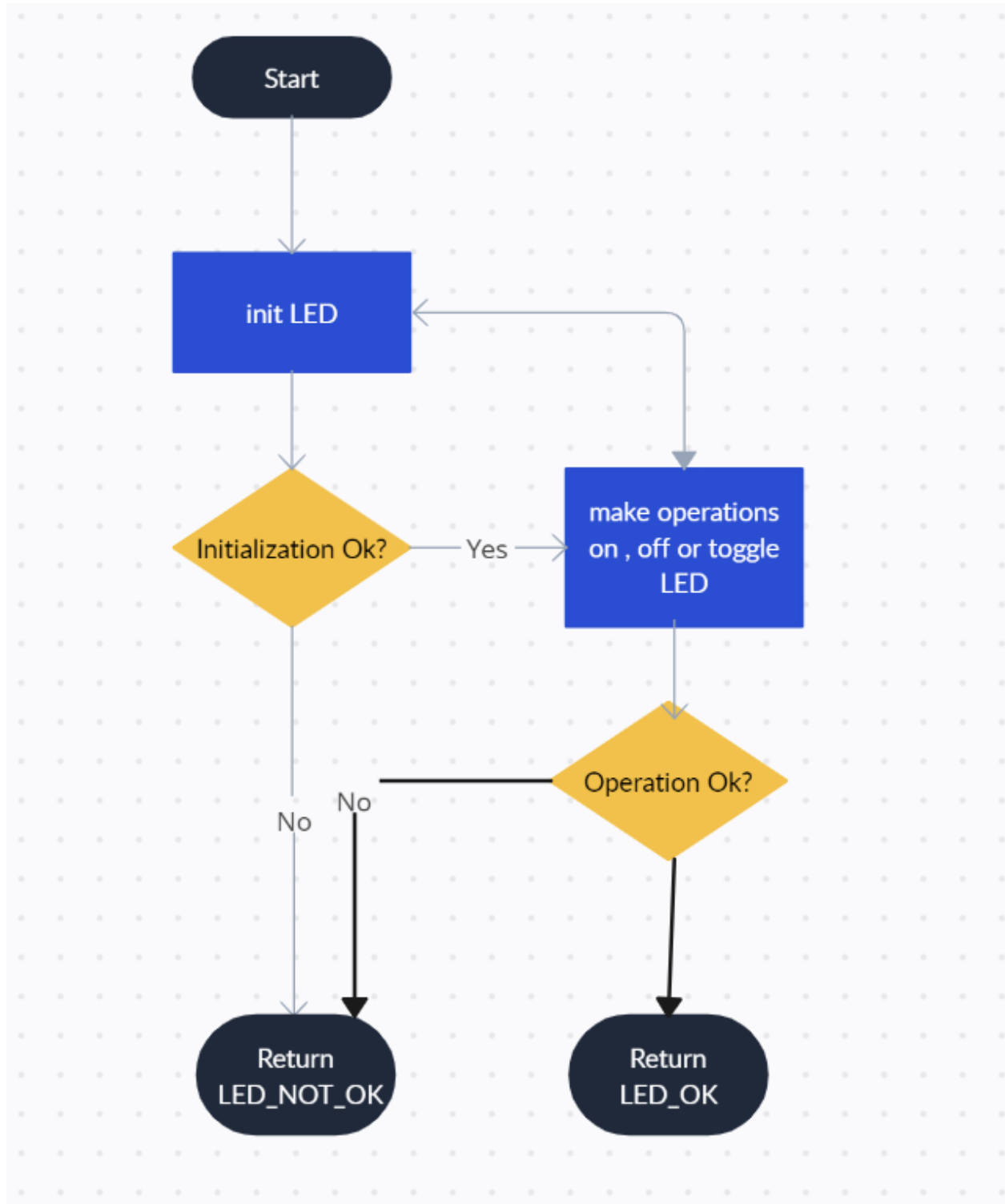


EN_dioError_t DIO_read(EN_dio_port_t port, EN_dio_pin_t pin, u8 *value)



UART:

LED:



3.2 Configurations:

3.2.1 DIO:

```

/...../
typedef struct{
    EN_dio_port_t    dio_port;
    EN_dio_pin_t     dio_pin;
    EN_dio_mode_t    dio_mode;
    EN_dio_value_t   dio_initial_value;
    EN_dio_pullup_t  dio_pullup_resistor;
}ST_DIO_ConfigType;

ST_DIO_ConfigType DIO_ConfigArray[];

/*****
/*          ENUMS DIO PRECOMPILED          */
*****/
typedef enum{
    PA=0,
    PB,
    PC,
    PD
}EN_DIO_Port_type;

typedef enum{
    OUTPUT,
    INFREE,
    INPULL
}EN_DIO_PinStatus_type;

typedef enum{
    LOW=0,
    HIGH,
}EN_DIO_PinVoltage_type;

```

```

/*****
/*          Pin modes          */
*****/
#define DIOMODE_INPUT    0
#define DIOMODE_OUTPUT   1

/*****
/*          Pin Direction Setting          */
*****/
#define DIOOUTPUT_LOW    0
#define DIOOUTPUT_HIGH   1

/*****
/*          Pin Pull Up Value          */
*****/
#define DIOINPUT_FLOATING 0
#define DIOINPUT_PULLUP   1

/*****
/*          Pin Pull Up Configuration          */
*****/
#define DIOPULLUP_DISABLED 0
#define DIOPULLUP_ENABLED  1

```

```

/
typedef enum{
    DIO_PORTA,
    DIO_PORTB,
    DIO_PORTC,
    DIO_PORTD
}EN_dio_port_t;

/*****
/*          DIO PINS          */
*****/
typedef enum{
    DIO_PIN0,
    DIO_PIN1,
    DIO_PIN2,
    DIO_PIN3,
    DIO_PIN4,
    DIO_PIN5,
    DIO_PIN6,
    DIO_PIN7
}EN_dio_pin_t;

/*****
/*          DIO PIN MODE DIRECTION          */
*****/
typedef enum{
    DIO_MODE_INPUT,
    DIO_MODE_OUTPUT
}EN_dio_mode_t;

/*****
/*          DIO PIN VALUE          */
*****/
typedef enum{
    DIO_HIGH,
    DIO_LOW
}EN_dio_value_t;

/*****
/*          DIO PIN PULL UP CONFIG          */
*****/
typedef enum{
    DIO_PULLUP_DISABLED,
    DIO_PULLUP_ENABLED
}EN_dio_pullup_t;

```

3.2.3 UART:

```

]/*USART SYNCHRONIZATION MODE OPTIONS:
  USART_ASYNC_MODE
  USART_SYNC_MODE
*/
#define USART_SET_SYNCH_MODE    USART_SYNC_MODE

]/*USART SPEED MODE OPTIONS:
  USART_NORMAL_SPEED
  USART_DOUBLE_SPEED
*/
#define USART_SET_SPEED    USART_NORMAL_SPEED

]/*USART PARITY OPTIONS :
  USART_NO_PARITY
  USART_ODD_PARITY
  USART_EVEN_PARITY
*/
#define USART_SET_PARITY_MODE    USART_EVEN_PARITY

]/*USART DATA SIZE OPTIONS :
  USART_DATA_SIZE_5
  USART_DATA_SIZE_6
  USART_DATA_SIZE_7
  USART_DATA_SIZE_8
  USART_DATA_SIZE_9
*/
#define USART_SET_DATA_SIZE    USART_DATA_SIZE_8

]/*USART STOP BITS OPTIONS :
  USART_ONE_STOP_BIT
  USART_TWO_STOP_BITS
*/
#define USART_SET_STOP_BITS    USART_TWO_STOP_BITS

]/*USART BAUD RATE OPTIONS
  BAUD_2400
  BAUD_4800
  BAUD_9600
  BAUD_14400
  BAUD_19200
  BAUD_28800
  BAUD_38400
*/
#define USART_SET_BAUD_RATE    BAUD_9600

```

```
/*USART SYNCHRONIZATION MODE OPTIONS:
USART_ASYNC_MODE
USART_SYNC_MODE
*/
typedef enum EN_USART_SET_MODE{
    USART_ASYNC_MODE=0,
    USART_SYNC_MODE
}EN_USART_SET_MODE;

/*USART SPEED MODE OPTIONS:
USART_NORMAL_SPEED
USART_DOUBLE_SPEED
*/
typedef enum EN_USART_SET_SPEED{
    USART_NORMAL_SPEED=0,
    USART_DOUBLE_SPEED
}EN_USART_SET_SPEED;

/*USART PARITY OPTIONS :
USART_NO_PARITY
USART_ODD_PARITY
USART_EVEN_PARITY
*/
typedef enum EN_USART_SET_PARITY{
    USART_NO_PARITY=0,
    USART_ODD_PARITY,
    USART_EVEN_PARITY
}EN_USART_SET_PARITY;

/*USART DATA SIZE OPTIONS :
USART_DATA_SIZE_5
USART_DATA_SIZE_6
USART_DATA_SIZE_7
USART_DATA_SIZE_8
USART_DATA_SIZE_9
*/
typedef enum EN_USART_SET_DATA_SIZE{
    USART_DATA_SIZE_5=0,
    USART_DATA_SIZE_6,
    USART_DATA_SIZE_7,
    USART_DATA_SIZE_8,
    USART_DATA_SIZE_9
}EN_USART_SET_DATA_SIZE;
```

```

typedef enum EN_USART_SET_STOP_BITS{
    USART_ONE_STOP_BIT=0,
    USART_TWO_STOP_BITS
}EN_USART_SET_STOP_BITS;

/*USART BAUD RATE OPTIONS
BAUD_2400
BAUD_4800
BAUD_9600
BAUD_14400
BAUD_19200
*/
typedef enum EN_USART_SET_BAUD_RATE{
    BAUD_2400=2400,
    BAUD_4800=4800,
    BAUD_9600=9600,
    BAUD_14400=14400,
    BAUD_19200=19200
}EN_USART_SET_BAUD_RATE;

/*CPU FREQUENCY OPTIONS
FCPU_4MHZ
FCPU_8MHZ
FCPU_16MHZ
*/
typedef enum EN_USART_SET_FCPU{
    FCPU_4MHZ=4000000,
    FCPU_8MHZ=8000000,
    FCPU_16MHZ=16000000
}EN_USART_SET_FCPU;

typedef struct ST_USART_CONFIG{
    EN_USART_SET_MODE    SYNC_MODE;
    EN_USART_SET_FCPU    FCPU;
    EN_USART_SET_BAUD_RATE    BAUD_RATE;
    EN_USART_SET_SPEED    SPEED_MODE;
    EN_USART_SET_PARITY    PARITY_MODE;
    EN_USART_SET_STOP_BITS    STOP_BIT;
    EN_USART_SET_DATA_SIZE    DATA_SIZE;
}ST_USART_CONFIG;

```



```

const ST_USART_CONFIG st_g_USARTconf = {
    .SYNC_MODE = USART_SYNC_MODE,
    .FCPU = FCPU_8MHZ,
    .BAUD_RATE = BAUD_9600,
    .SPEED_MODE = USART_NORMAL_SPEED,
    .PARITY_MODE = USART_EVEN_PARITY,
    .STOP_BIT = USART_TWO_STOP_BITS,
    .DATA_SIZE = USART_DATA_SIZE_8,
};

```

3.2.3 LED:

```

/*****
 *                               Typedefs                               *
 *****/
/*Enum for error state*/
typedef enum
{
    LED_OK,
    LED_NOK
}EN_ledError_t;

/*struct to store led attributes*/
typedef struct LEDS{
    u8 port;
    u8 pin;
    u8 state;
}LEDS;

```

BCM APIs:

BCM_init()

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 of the BCM Instance
Parameters (out):	None
Parameters (in, out):	None
Return:	BCM_STATUS_SUCCESS: In case of Successful Operation BCM_STATUS_INVALID_STATE: In case of Successful Operation

BCM_deinit()

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 of the BCM Instance
Parameters (out):	None
Parameters (in, out):	None
Return:	BCM_STATUS_SUCCESS: In case of Successful Operation BCM_STATUS_INVALID_STATE: In case of Successful Operation

BCM_send()

Function Name	BCM_send
Syntax	enu_system_status_t bcm_init (str_bcm_instance_t* ptr_str_bcm_instance,u8* ptr_u8_a_byte);
Sync/Async	Synchronous
Reentrancy	Non Reentrant
Parameters (in):	ptr_str_bcm_instance: Address of the BCM Instance ptr_u8_a_byte : Address of the sending byte
Parameters (out):	None
Parameters (in, out):	None
Return:	BCM_STATUS_SUCCESS: In case of Successful Operation BCM_STATUS_INVALID_STATE: In case of Successful Operation

BCM_send_n()

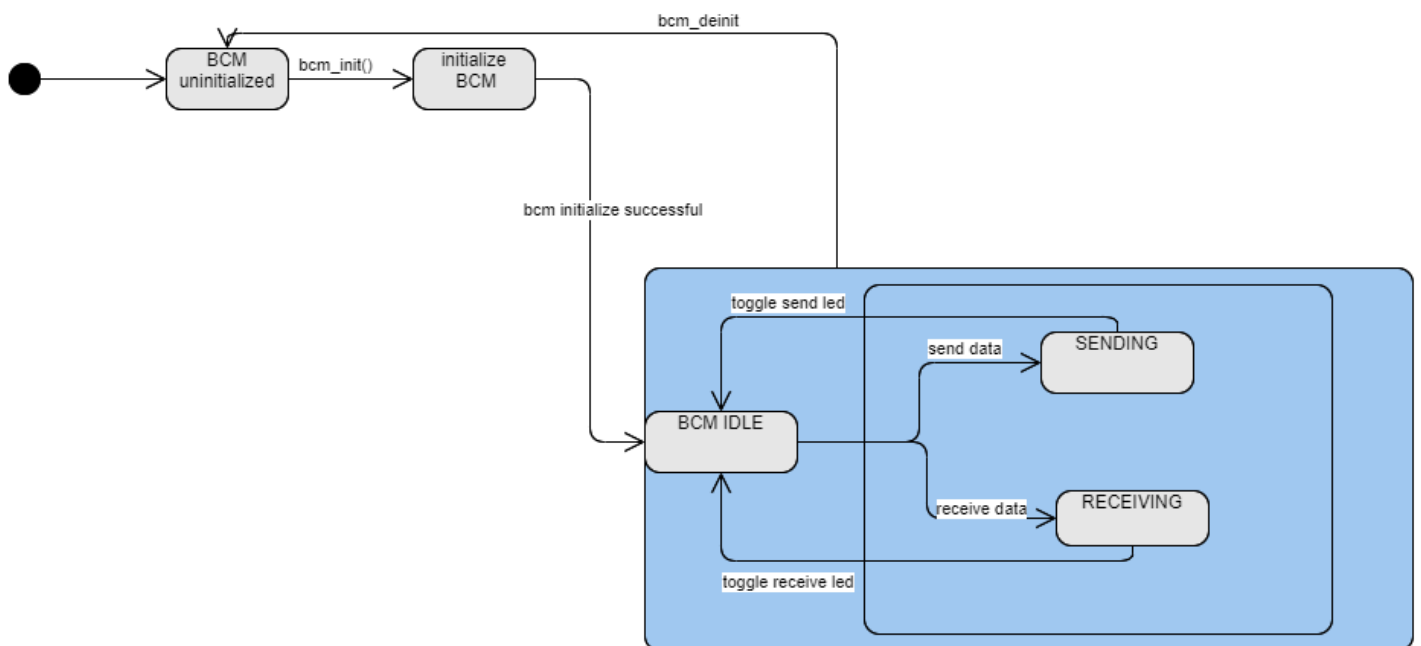
Function Name	BCM_send_n
Syntax	enu_system_status_t bcm_init (str_bcm_instance_t* ptr_str_bcm_instance,u8* ptr_u8_a_byte);
Sync/Async	Synchronous
Reentrancy	Non Reentrant
Parameters (in):	ptr_str_bcm_instance: Address of the BCM Instance ptr_u8_a_byte : Address of the sending bytes
Parameters (out):	None
Parameters (in, out):	None
Return:	BCM_STATUS_SUCCESS: In case of Successful Operation BCM_STATUS_INVALID_STATE: In case of Successful Operation

bcm_dispatcher ()

Function Name	BCM_dispatcher
Syntax	void BCM_dispatcher(void)
Sync/Async	Synchronous
Reentrancy	Non Reentrant
Parameters (in):	None
Parameters (out):	None
Parameters (in, out):	None
Return:	None

UML:

BCM State Machine:



BCM Sequence Diagram:

