BASIC COMMUNICATION MANGER (BCM)

PREPARED BY

MOHAMMED ABDULWAHAB HEGAZY

Sprints

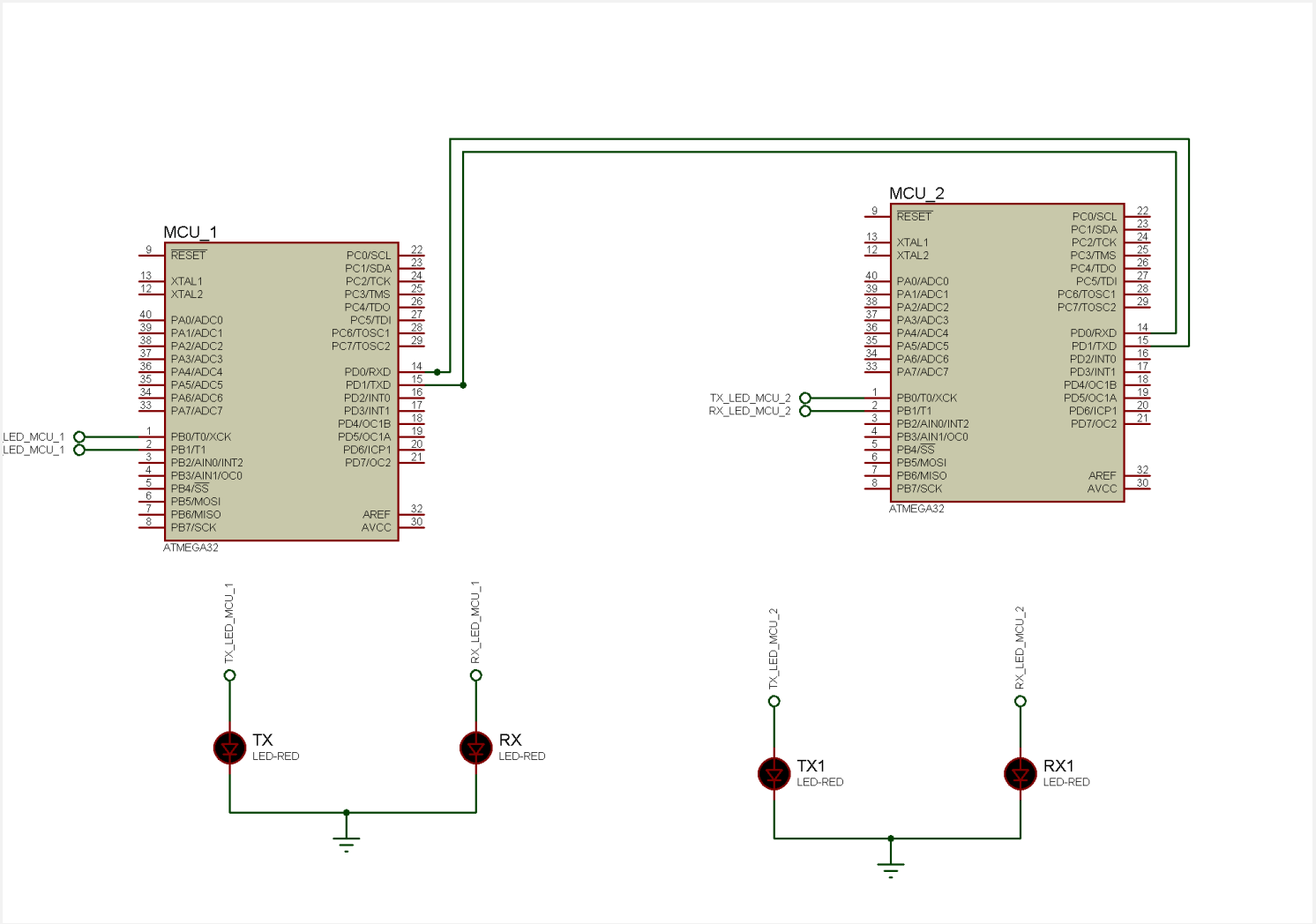
1. Project Introduction

Basic communication manager is responsible for all the communication done by the application layer with different communication protocols but with more abstracted way.

1.1. Project Components

- ATMEGA32
- LEDS

1.1.1. Circuit Schematic



1.2. SYSTEM REQUIRMENTS

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_inctance_t *str_ptr_a_bcm_inctance);`
Sync/Async	Synchronous
Reentrancy	Non reentrant
Parameters (in)	Ptr_str_bcm_instance :the refrence of a bcm structure object
Parameters (out)	None
Parameters (in, out)	None
Return	`typedef enum { BCM_E_OK=0, BCM_E_NOK=2, }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_deInit(str_bcm_inctance_t *str_ptr_a_bcm_inctance);`
Sync/Async	Synchronous
Reentrancy	Non reentrant
Parameters (in)	Ptr_str_bcm_instance :the refrence of a bcm structure object
Parameters (out)	None
Parameters (in, out)	None
Return	`typedef enum { BCM_E_OK=0, BCM_E_NOK=2, }enu_system_status_t;`

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(str_bcm_inctance_t *str_ptr_a_bcm_inctance , uint8_t u8_a_data);`
Sync/Async	Asynchronous(cause using non-blocking)
Reentrancy	Non reentrant
Parameters (in)	Ptr_str_bcm_instance :the refrence of a bcm structure object `u8_a_data: data to be send`
Parameters (out)	None
Parameters (in, out)	None

Return	`typedef enum {`
	`BCM_E_OK=0,`
	`BCM_E_NOK=2,`
	`}enu_system_status_t;`

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(str_bcm_inctance_t *str_ptr_a_bcm_inctance , uint8_t *u8Arr_a_stringData, uint16_t u16_a_stringSize);`
Sync/Async	Asynchronous(cause using non-blocking)
Reentrancy	Re-entrant

Parameters (in)	Ptr_str_bcm_instance :the refrence of a bcm structure object
	`u8Arr_a_stringData: string to be send`
	`u16_a_stringSize) : string size`

Parameters (out)	None
Parameters (in, out)	None

Return	`typedef enum {`
	`BCM_E_OK=0,`
	`BCM_E_NOK=2,`
	`}enu_system_status_t;`

6. Implement **bcm_receive** will receive only 1 byte of data over a specific BCM instance

Function Name	**bcm_receive**
Syntax	`enu_system_status_t BCM_recieve(str_bcm_inctance_t *str_ptr_a_bcm_inctance , uint8_t *u8_a_data);`
Sync/Async	Asynchronous(cause using non-blocking)
Reentrancy	Re-entrant
Parameters (in)	Ptr_str_bcm_instance :the reference of a bcm structure object
Parameters (out)	None
Parameters (in, out)	`u8Arr_a_stringData`:refrenc to the data be stored at
Return	`typedef enum { BCM_E_OK=0, BCM_E_NOK=2, }enu_system_status_t;`

7. Implement **bcm_ receive_n** will receive more than one byte with a length n over a specific BCM instance

Function Name	**bcm_send_n**
Syntax	`enu_system_status_t BCM_recieve_n(str_bcm_inctance_t *str_ptr_a_bcm_inctance , uint8_t *u8Arr_a_stringData, uint16_t u16_a_stringSize);`
Sync/Async	Asynchronous(cause using non-blocking)
Reentrancy	Re-entrant
Parameters (in)	Ptr_str_bcm_instance :the refrence of a bcm structure object `u16_a_stringSize) : string size`
Parameters (out)	None
Parameters (in, out)	`u8Arr_a_stringData`: reference to the string which will store the data at
Return	`typedef enum { BCM_E_OK=0, BCM_E_NOK=2, }enu_system_status_t;`

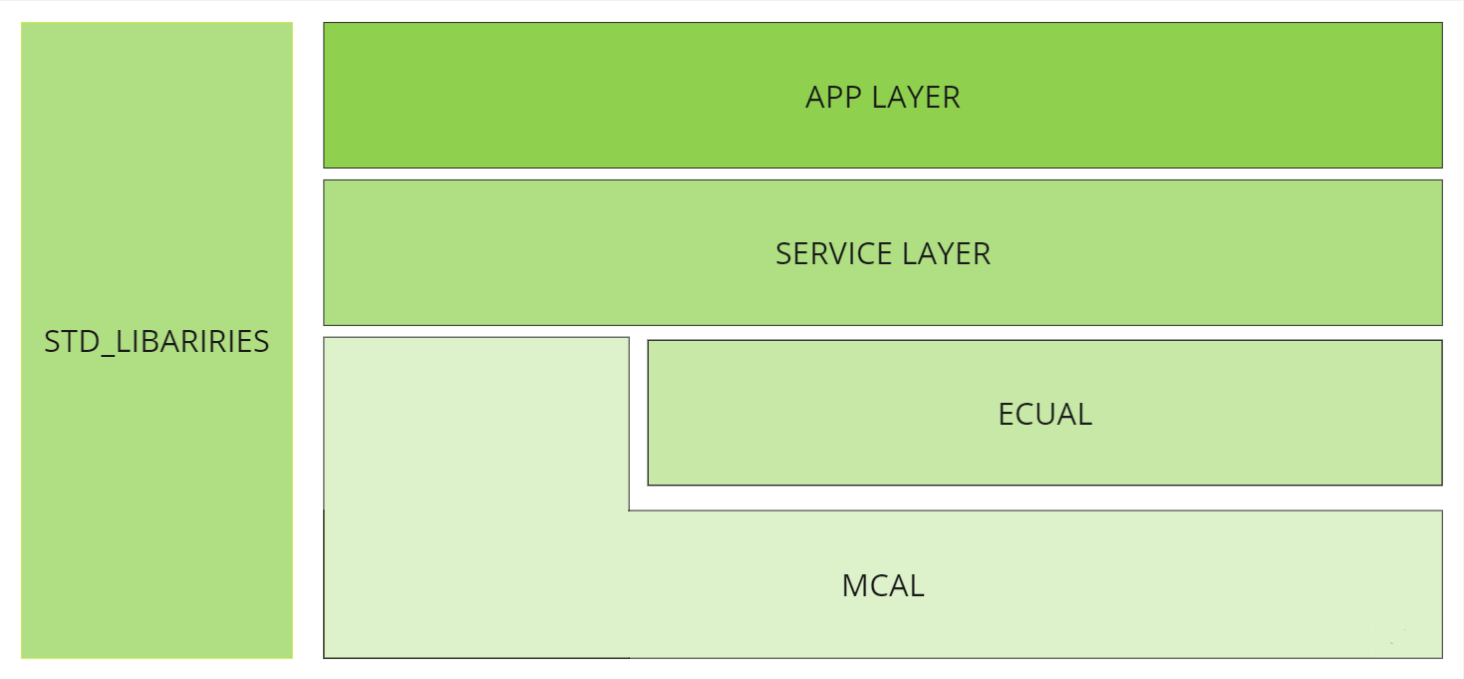
8. Implement **bcm_dispatcher** will execute the periodic actions and notifies the userwith the needed events over a specific BCM instance

Function Name	**bcm_send_n**
Syntax	`enu_system_status_t BCM_dispatcher(str_bcm_inctance_t *str_ptr_a_bcm_inctance );`
Sync/Async	Synchronous
Reentrancy	Re-entrant
Parameters (in)	Ptr_str_bcm_instance :the refrence of a bcm structure object
Parameters (out)	None
Parameters (in, out)	None
Return	`typedef enum { BCM_E_OK=0, BCM_E_NOK=2, }enu_system_status_t;`

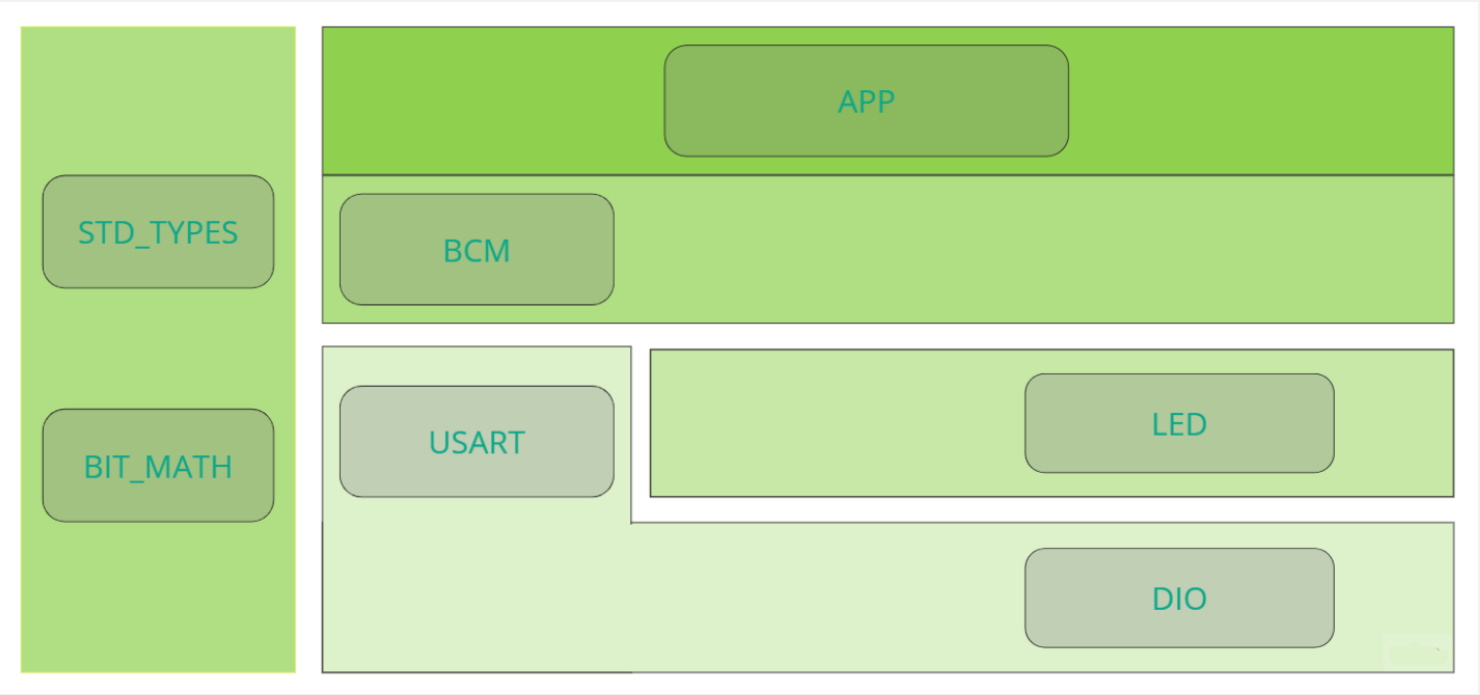
2. High Level Design

2.1.System Architecture

2.1.1. Layered Architecture



2.1.2. System modules



2.2. Modules Description

2.2.1. DIO Module

DIO module: Digital Input/output module is used to drive an output digital logic or read digital logic from external devices.

2.2.2. USART Module

A USART (universal synchronous/asynchronous receiver/transmitter) is hardware that enables a device to communicate using serial protocols. It can function in a slower asynchronous mode, like a universal asynchronous receiver/transmitter (UART), or in a faster synchronous mode with a clock signal.

2.2.3. LED Module

LED 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

Basic communication manager is responsible for all the communication done by the application layer with different communication protocols but with more abstracted way.

2.3. Drivers API's

2.3.1 MCAL API's

2.3.1.1 DIO API's

```
/*********************************  Function Declarations ************************/
/**
* @brief Initialize the direction of specific pin @ref direction_t
* @param _pin_config A Reference of the pin configuration @pin_config_t
* @return status of the function
* E_OK :the function done successfully
* E_NOT_OK :the function has issues performing the function
*/
Std_ReturnType DIO_pin_direction_intialize(const pin_config_t *pin_config_ptr,direction_t
a_direction);
/**
* @brief Write the logic of specific pin @ref logic_t
* @param _pin_config A Reference of the pin configuration @pin_config_t
* @param logic
* @return status of the function
* E_OK :the function done successfully
* E_NOT_OK :the function has issues performing the function
*/
Std_ReturnType DIO_pin_write_logic(const pin_config_t *pin_config_ptr,const logic_t
a_logic);
/**
* @brief Read the logic of specific pin @ref logic_t
* @param _pin_config A Reference of the pin configuration @pin_config_t
* @param logic
* @return status of the function
* E_OK :the function done successfully
* E_NOT_OK :the function has issues performing the function
*/
Std_ReturnType DIO_pin_read_logic(const pin_config_t *pin_config_ptr, logic_t
*logic_ptr);

/**
* @brief Toggle the logic of specific pin @ref logic_t
* @param _pin_config A Reference of the pin configuration @pin_config_t
* @return status of the function
* E_OK :the function done successfully
* E_NOT_OK :the function has issues performing the function
*/
Std_ReturnType DIO_pin_toggle_logic(const pin_config_t *pin_config_ptr);
```

2.3.1.2 USART API's

```
/*
* Description : initialize the USART driver
* @param A Reference of the USART driver's configuration structure
* @return Std_ReturnType: status of the function
* USART_E_OK :the function done successfully
* USART_E_NOK :the function has issues performing the function
*/
u8_usartErorrState_t USART_init(const st_usart_config_t *stPtr_a_usartConfig);

/*
* Description : De-initialize the USART driver
* @param A Reference of the USART driver's configuration structure
* @return Std_ReturnType: status of the function
* USART_E_OK :the function done successfully
* USART_E_NOK :the function has issues performing the function
*/
u8_usartErorrState_t USART_DeInit(const st_usart_config_t *stPtr_a_usartConfig);
/*
* Description : Send one byte via USART bus
* @param u8_a_data : The data to be send
*              stPtr_a_usartConfig : A Reference of the USART driver's configuration
structure
* @return Std_ReturnType: status of the function
* USART_E_OK :the function done successfully
* USART_E_NOK :the function has issues performing the function
*/
u8_usartErorrState_t USART_sendData(const st_usart_config_t *stPtr_a_usartConfig ,
uint8_t u8_a_data);
/*
* Description : Receive one byte via USART bus
* @param u8Ptr_a_data: A Reference of the container of the received data
*                     stPtr_a_usartConfig:A Reference of the USART driver's configuration
structure
* @return Std_ReturnType: status of the function
* USART_E_OK :the function done successfully
* USART_E_NOK :the function has issues performing the function
*/
u8_usartErorrState_t USART_reciveData(const st_usart_config_t *stPtr_a_usartConfig ,
uint8_t *const u8Ptr_a_data);
/*
* Description : Send string via USART bus
* @param The data string -array of characters- to be send
*                     stPtr_a_usartConfig:A Reference of the USART driver's configuration
structure
* @return Std_ReturnType: status of the function
* USART_E_OK :the function done successfully
* USART_E_NOK :the function has issues performing the function
*/
u8_usartErorrState_t USART_sendString(const st_usart_config_t *stPtr_a_usartConfig ,
uint8_t *u8Arr_a_stringOfData , uint16_t u16_a_stringSize);
/*
* Description : Receive string via USART bus
* @param The data string -array of characters- to store the received data string
*                     stPtr_a_usartConfig:A Reference of the USART driver's configuration
structure
```

```
* @return Std_ReturnType: status of the function
* USART_E_OK :the function done successfully
* USART_E_NOK :the function has issues performing the function
*/
u8_usartErorrState_t USART_reciveString(const st_usart_config_t *stPtr_a_usartConfig
,uint8_t *const u8Arr_a_stringOfData , uint16_t u16_a_stringSize);


/*
* Description : Call the Call Back function in the application after transmissions did
its job
* @param A pointer to function
* @return status of the function
* USART_E_OK :the function done successfully
* USART_E_NOK :the function has issues performing the function
*/
u8_usartErorrState_t USART_setCallBackTx( Fptr_usartCallBack_t Fptr_a_TxCallBack);

/*
* Description : Call the Call Back function in the application after Reception did its
job
* @param A pointer to function
* @return status of the function
* USART_E_OK :the function done successfully
* USART_E_NOK :the function has issues performing the function
*/
u8_usartErorrState_t USART_setCallBackRx( Fptr_usartCallBack_t Fptr_a_RxCallBack);
```

2.3.2. HAL API's

2.3.1.1. LED API's

```
/**
 * @breif  Initialize The led by configuring the pin as output and write low
 * @param Led  The reference of the led module configuration
 * @return status of the function
 *          E_OK :the function done successfully
 *          E_NOT_OK :the function has issues performing the function
 */
Std_ReturnType LED_initialize(const led_t *led_ptr);

/**
 * @breif Turn the led on
 * @param led  The reference of the led module configuration
 * @return status of the function
 *          E_OK :the function done successfully
 *          E_NOT_OK :the function has issues performing the function
 */

Std_ReturnType LED_turn_on(const led_t *led_ptr);

/**
 * @breif Turn the led off
 * @param led  The reference of the led module configuration
```

```
 * @return status of the function
 *          E_OK :the function done successfully
 *          E_NOT_OK :the function has issues performing the function
 */

Std_ReturnType LED_turn_off (const led_t *led_ptr);

/**
 * @breif  Toggle the led
 * @param led  The reference of the led module configuration
 * @return status of the function
 *          E_OK :the function done successfully
 *          E_NOT_OK :the function has issues performing the function
 */
Std_ReturnType LED_turn_toggle (const led_t *led_ptr);
```

2.3.3. SERVICE API's

2.3.3.1 BCM API's

```
/*
* Description : initialize the BCM communication unit
* @param A Reference of the BCM driver's configuration structure
* @return enu_system_status_t: status of the function
* BCM_E_OK :the function done successfully
* BCM_E_NOK :the function has issues performing the function
*/
enu_system_status_t BCM_init(str_bcm_inctance_t *str_ptr_a_bcm_inctance);
/*
* Description : De-initialize the BCM communication unit
* @param A Reference of the BCM driver's configuration structure
* @return enu_system_status_t: status of the function
* BCM_E_OK :the function done successfully
* BCM_E_NOK :the function has issues performing the function
*/
enu_system_status_t BCM_deinit(str_bcm_inctance_t *str_ptr_a_bcm_inctance);
/*
* Description : Send one byte via BCM
* @param
*              str_ptr_a_bcm_inctance : A Reference of the BCM driver's configuration
structure
*              u8_a_data : The data to be send
* @return enu_system_status_t: status of the function
* BCM_E_OK :the function done successfully
* BCM_E_NOK :the function has issues performing the function
*/
enu_system_status_t BCM_send(str_bcm_inctance_t *str_ptr_a_bcm_inctance , uint8_t
u8_a_data);
/*
* Description : Receive one byte via BCM
* @param
*              str_ptr_a_bcm_inctance : A Reference of the BCM driver's configuration
structure
```

```
*              u8_a_data :  A Reference of The variable which will be store the recived
byte at
* @return enu_system_status_t: status of the function
* BCM_E_OK :the function done successfully
* BCM_E_NOK :the function has issues performing the function
*/
enu_system_status_t BCM_recieve(str_bcm_inctance_t *str_ptr_a_bcm_inctance , uint8_t
*u8_a_data);
/*
* Description : Send Multiple bytes via BCM
* @param
*              str_ptr_a_bcm_inctance : A Reference of the BCM driver's configuration
structure
*              u8_a_data : The string to be send
*              u16_a_stringSize : The string size
* @return enu_system_status_t: status of the function
* BCM_E_OK :the function done successfully
* BCM_E_NOK :the function has issues performing the function
*/
enu_system_status_t BCM_send_n(str_bcm_inctance_t *str_ptr_a_bcm_inctance , uint8_t
*u8Arr_a_stringData, uint16_t u16_a_stringSize);
/*
* Description : receive Multiple bytes via BCM
* @param
*              str_ptr_a_bcm_inctance : A Reference of the BCM driver's configuration
structure
*              u8_a_data : The string to stored the string at
*              u16_a_stringSize : The string size
* @return enu_system_status_t: status of the function
* BCM_E_OK :the function done successfully
* BCM_E_NOK :the function has issues performing the function
*/
enu_system_status_t BCM_recieve_n(str_bcm_inctance_t *str_ptr_a_bcm_inctance , uint8_t
*u8Arr_a_stringData, uint16_t u16_a_stringSize);
/*
* Description : Call the Call Back function in the application after transmissions or
reception is done
* @paramstr_ptr_a_bcm_inctance : A Reference of the BCM driver's configuration structure
* @return enu_system_status_t: status of the function
* BCM_E_OK :the function done successfully
* BCM_E_NOK :the function has issues performing the function
*/
enu_system_status_t BCM_setCallBack(str_bcm_inctance_t *str_ptr_a_bcm_inctance );
/*
* Description :Will execute the periodic actions and notifies the user with the needed
events over a specific BCM instance
* @paramstr_ptr_a_bcm_inctance : A Reference of the BCM driver's configuration structure
* @return enu_system_status_t: status of the function
* BCM_E_OK :the function done successfully
* BCM_E_NOK :the function has issues performing the function
*/
enu_system_status_t BCM_dispatcher(str_bcm_inctance_t *str_ptr_a_bcm_inctance );
```

2.3.3. APP API's

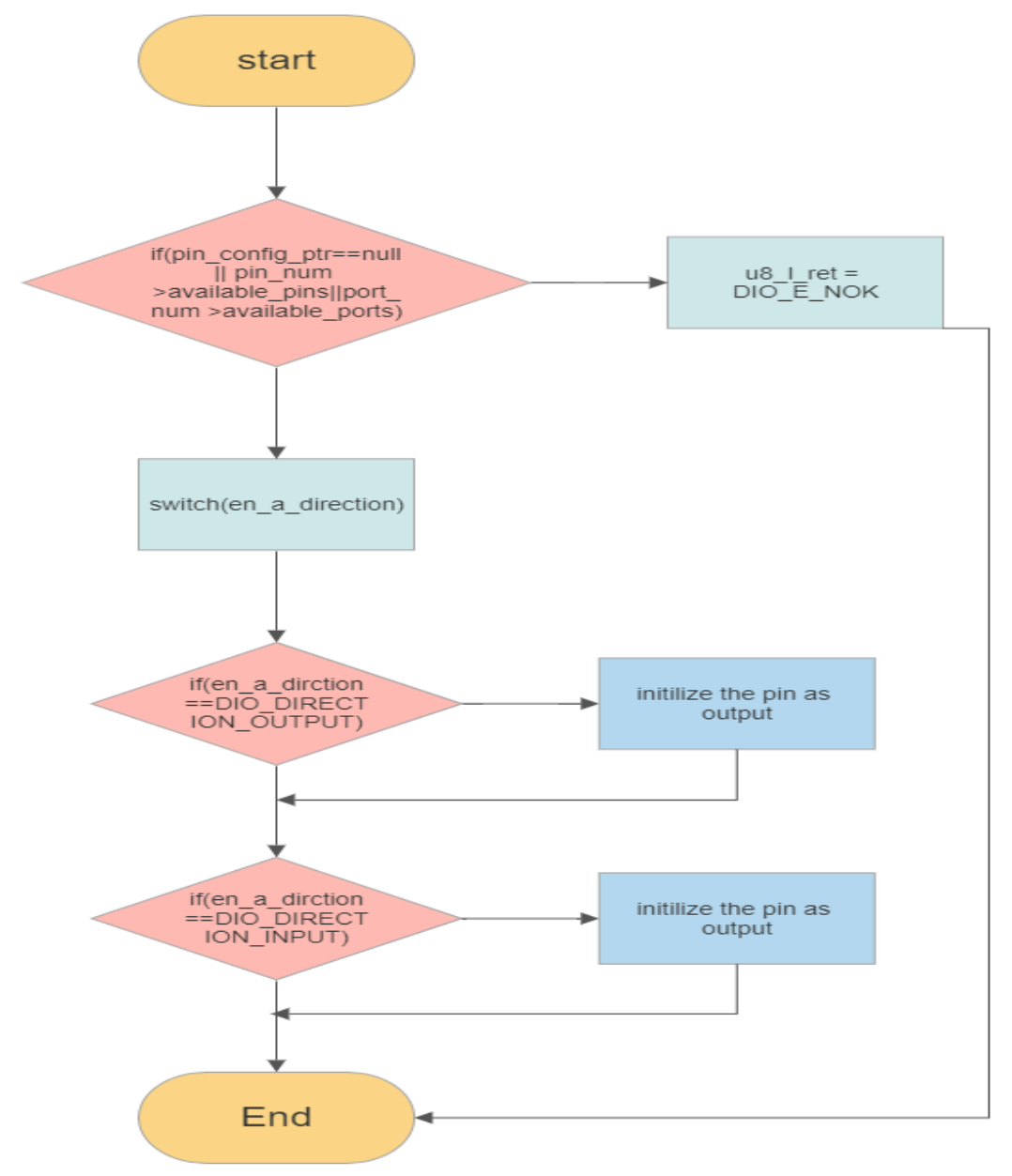
```
void APP_init(void);

void APP_start(void);
```

3. Low Level Design

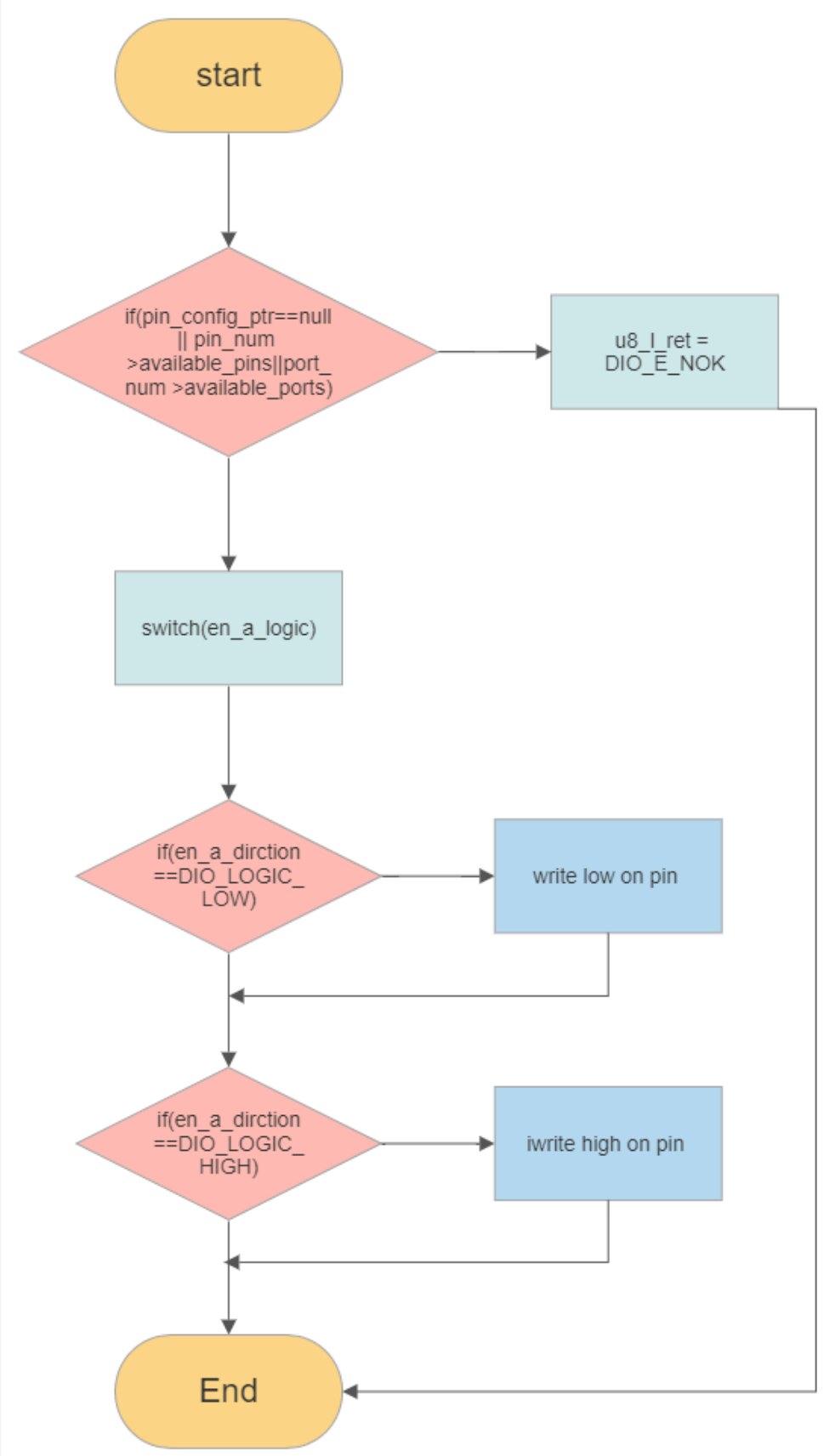
3.1. MCAL

3.1.1 DIO'S FLOWCHARTS

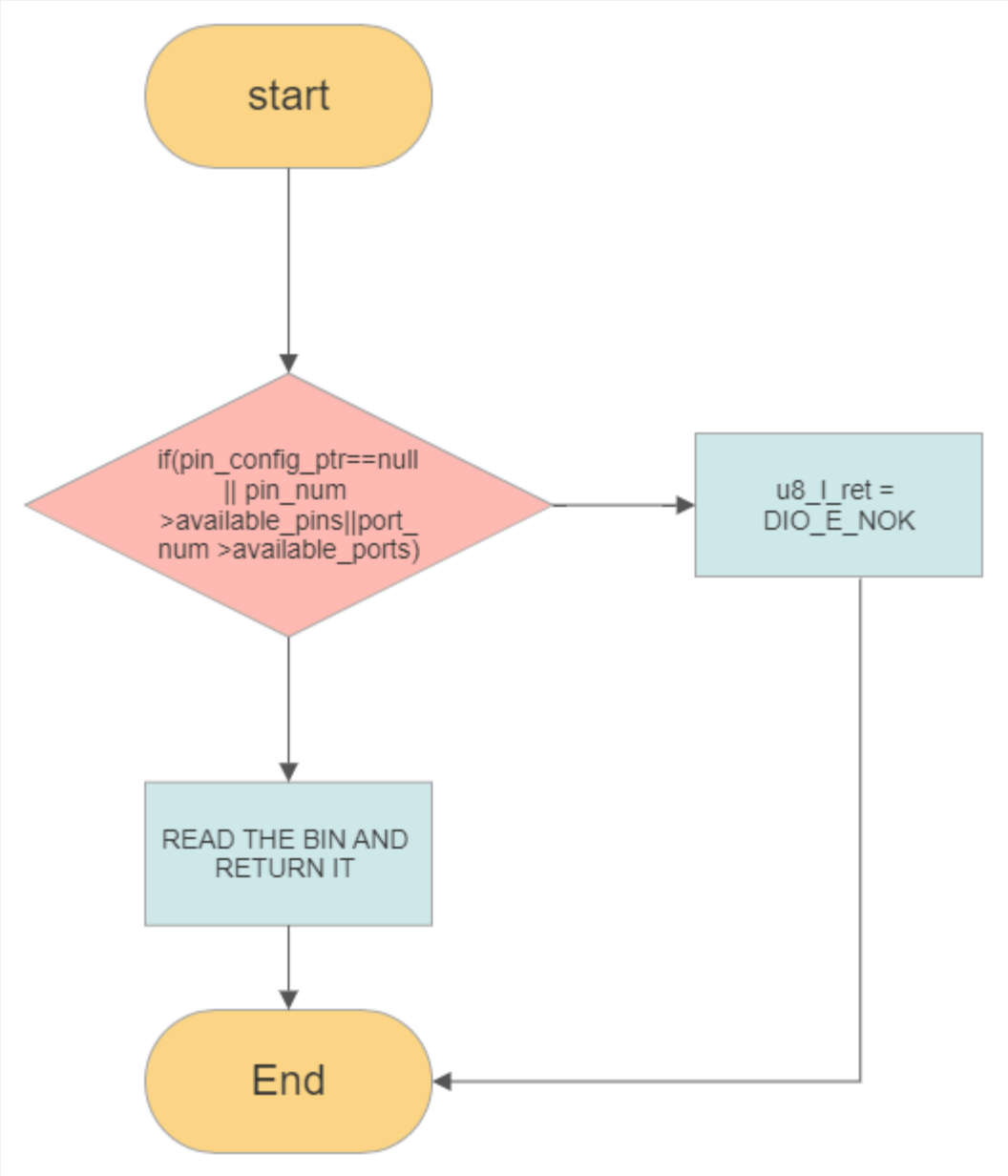
```
Std_ReturnType DIO_pin_direction_intialize(const pin_config_t *pin_config_ptr,direction_t a_direction);
```



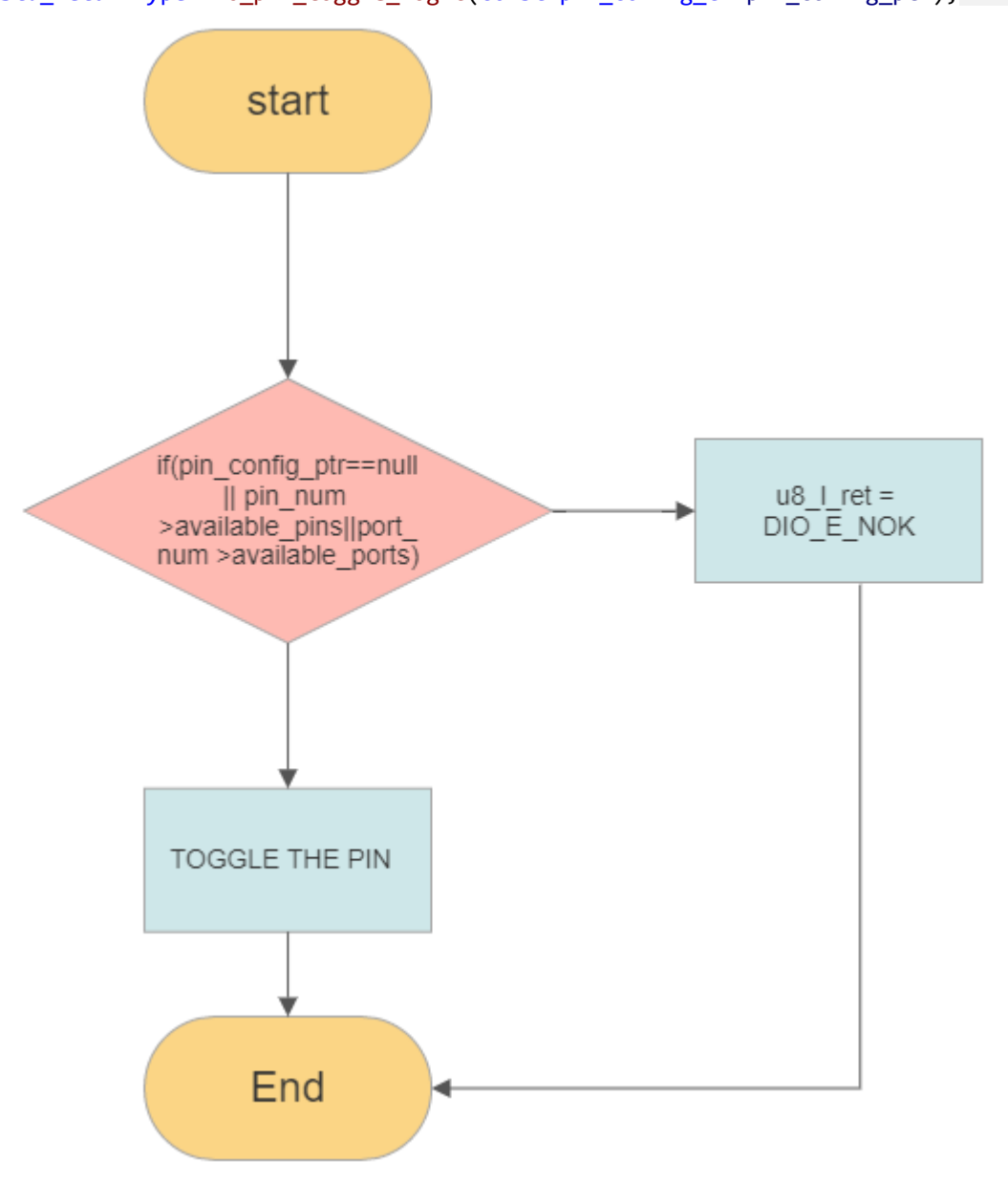
```
Std_ReturnType DIO_pin_write_logic(const pin_config_t *pin_config_ptr,const logic_t
a_logic);
```



```
Std_ReturnType DIO_pin_read_logic(const pin_config_t *pin_config_ptr, logic_t
*logic_ptr);
```



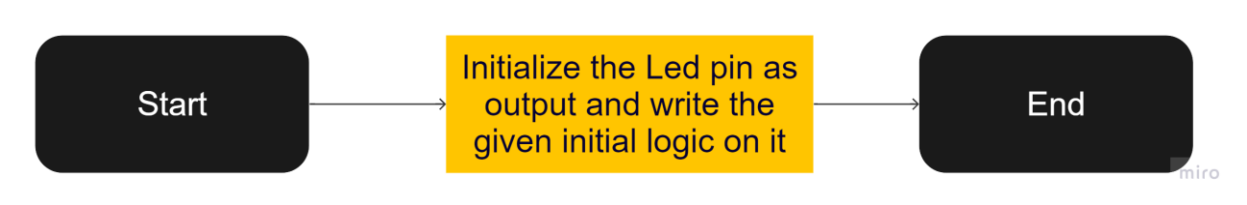
```
Std_ReturnType DIO_pin_toggle_logic(const pin_config_t *pin_config_ptr);
```



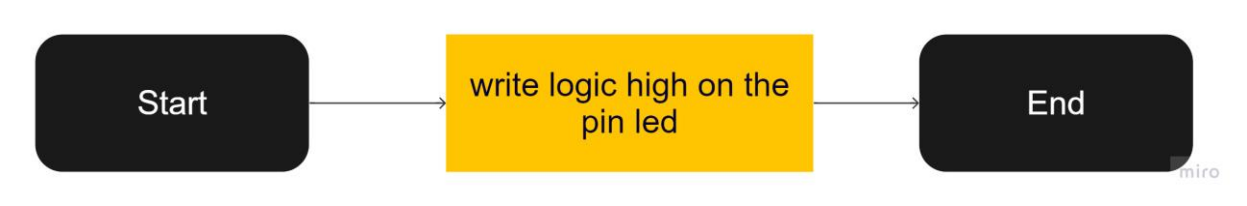
3.2. HAL

3.2.1. LED'S FLOWCHARTS

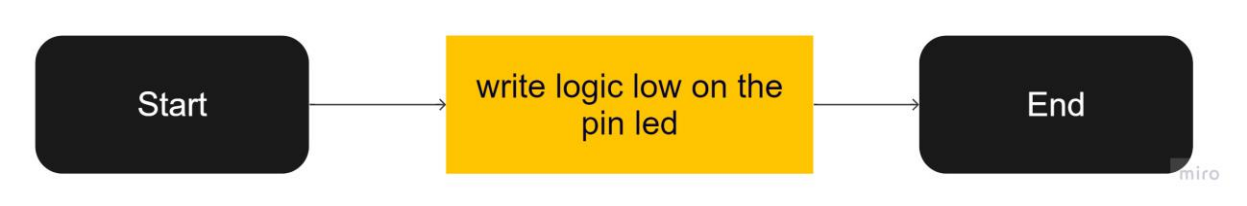
```
Std_ReturnType LED_initialize(const led_t *led_ptr);
```



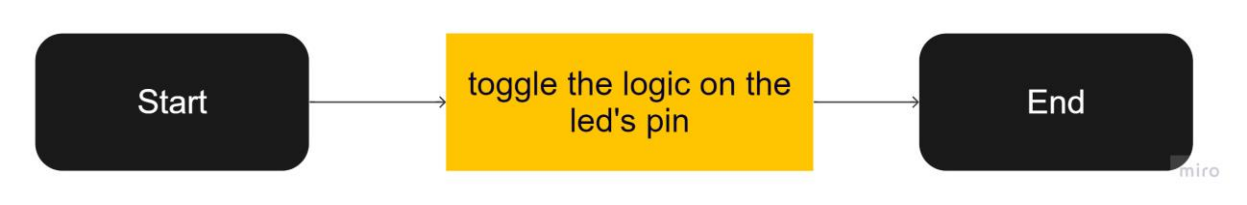

```
Std_ReturnType LED_turn_on(const led_t *led_ptr);
```




```
Std_ReturnType LED_turn_off (const led_t *led_ptr);
```



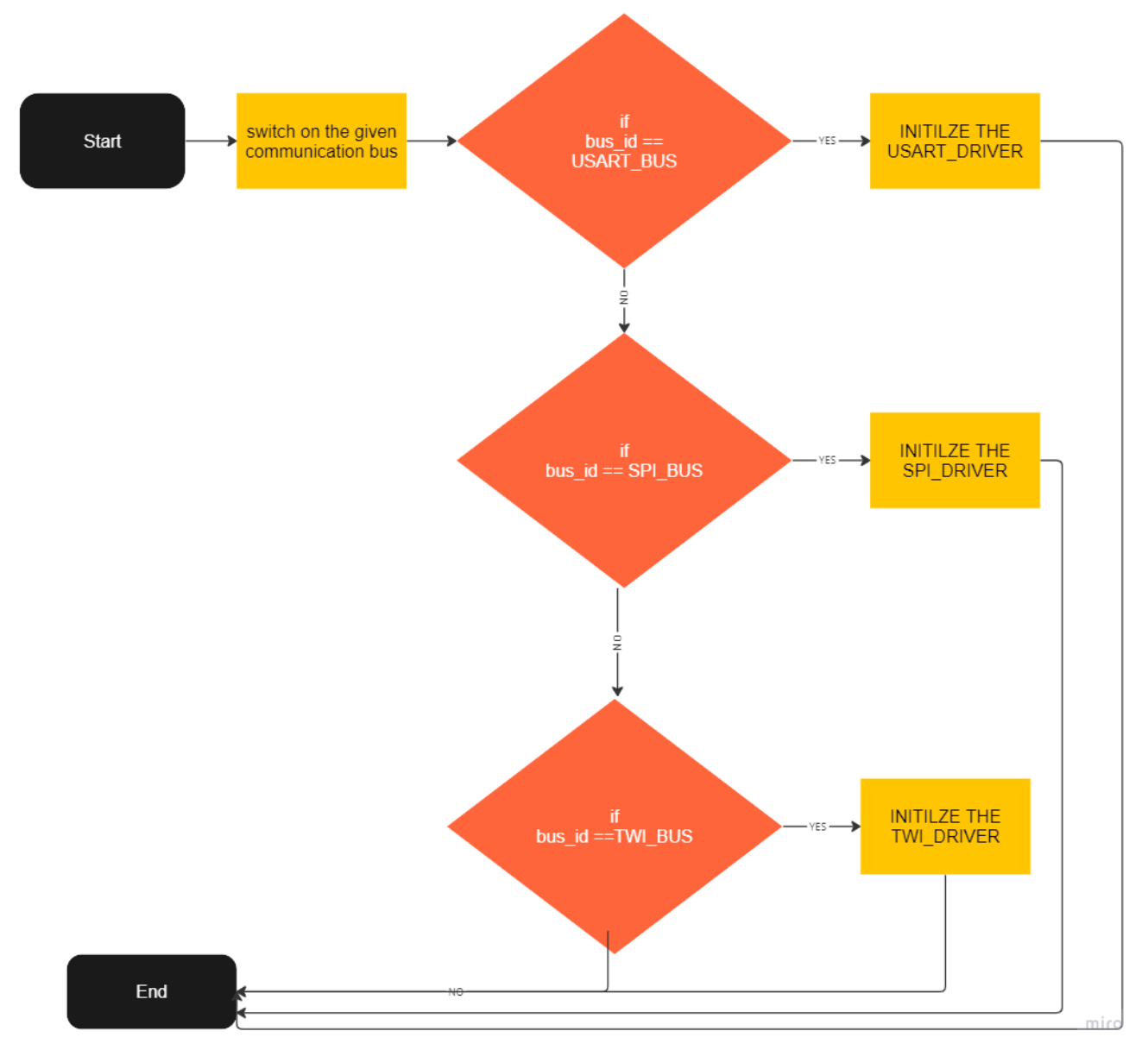

```
Std_ReturnType LED_turn_toggle (const led_t *led_ptr);
```



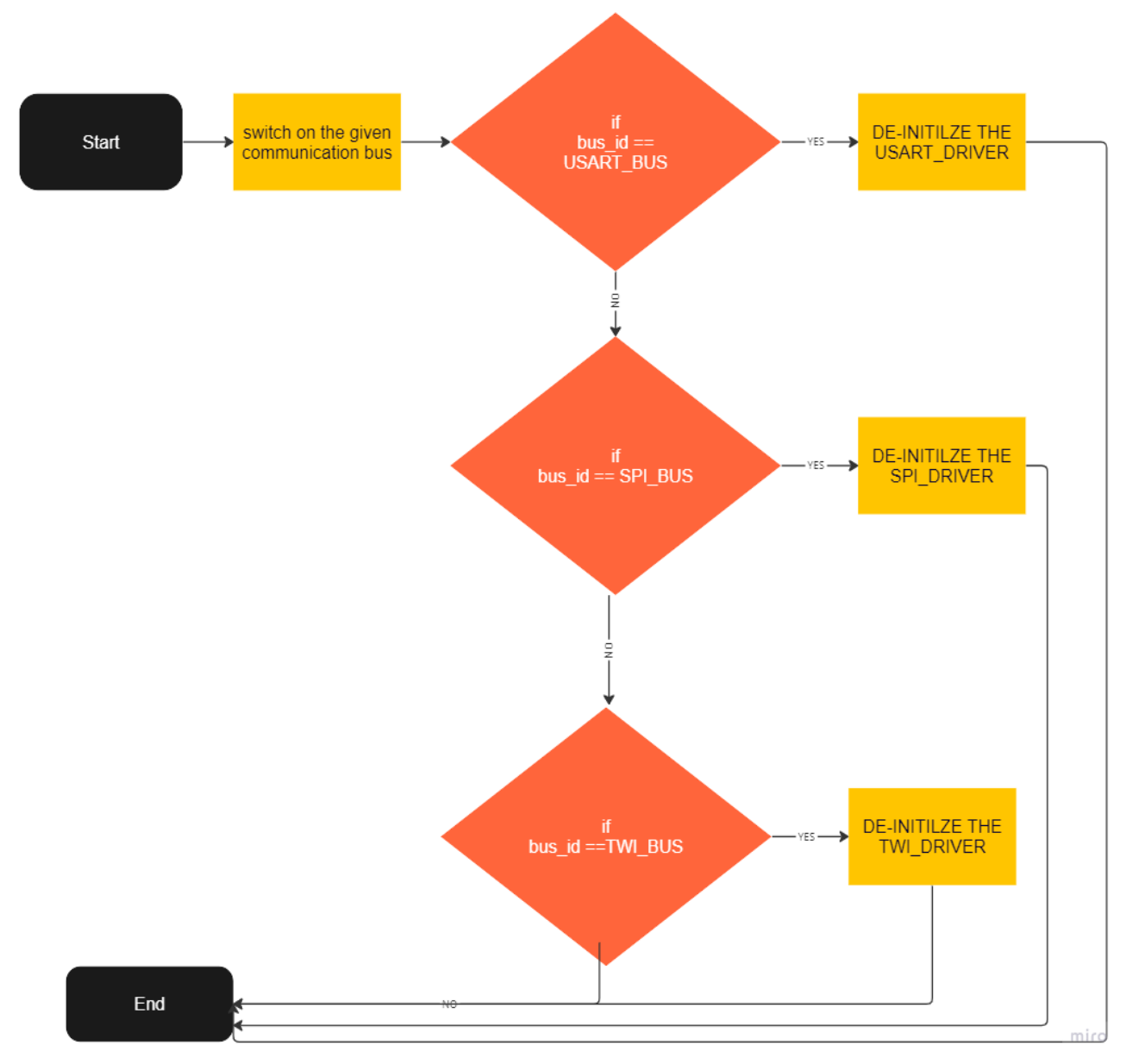
3.2. SERVICE LAYER

3.2.1. BCM'S FLOWCHARTS

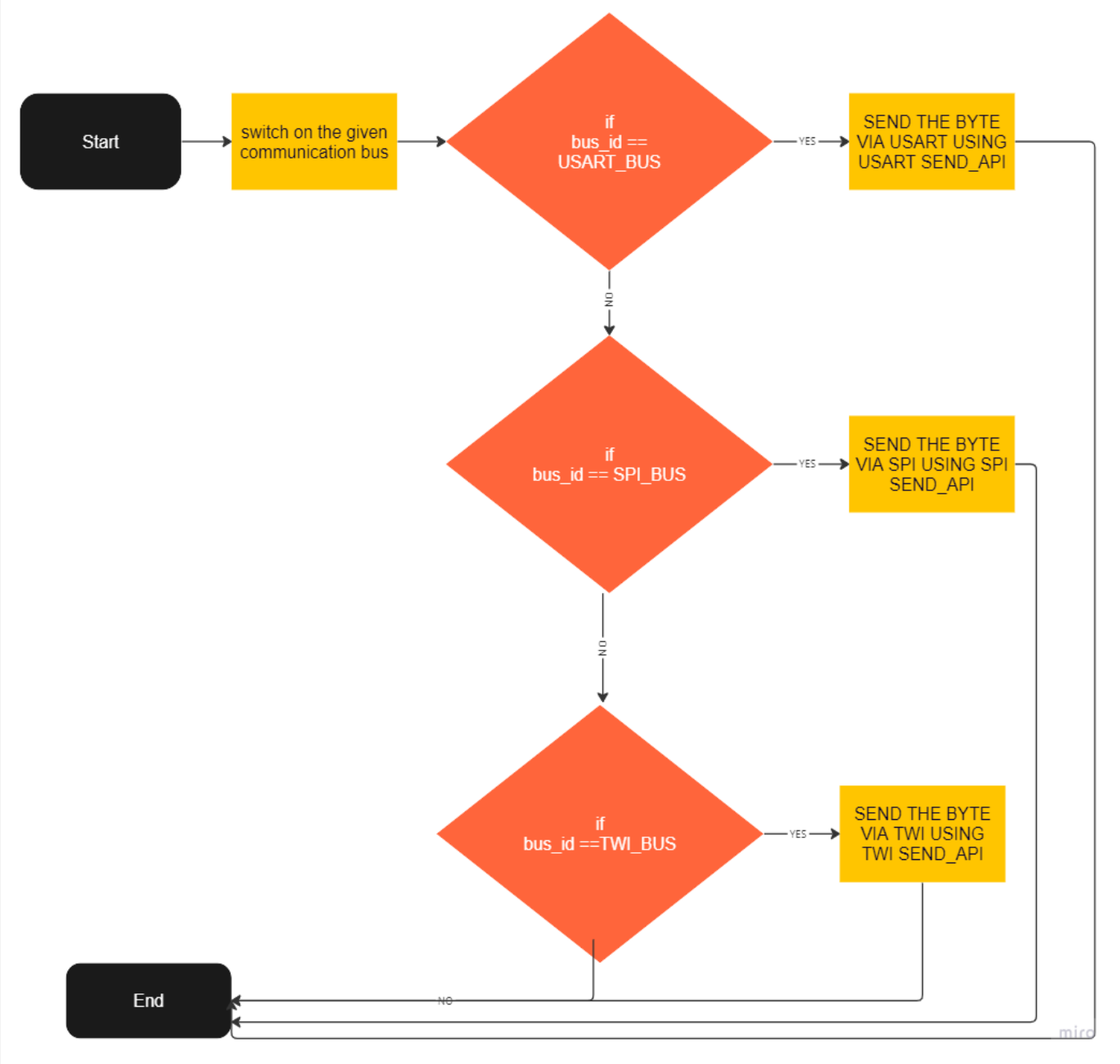
```
enu_system_status_t BCM_init(str_bcm_inctance_t *str_ptr_a_bcm_inctance);
```



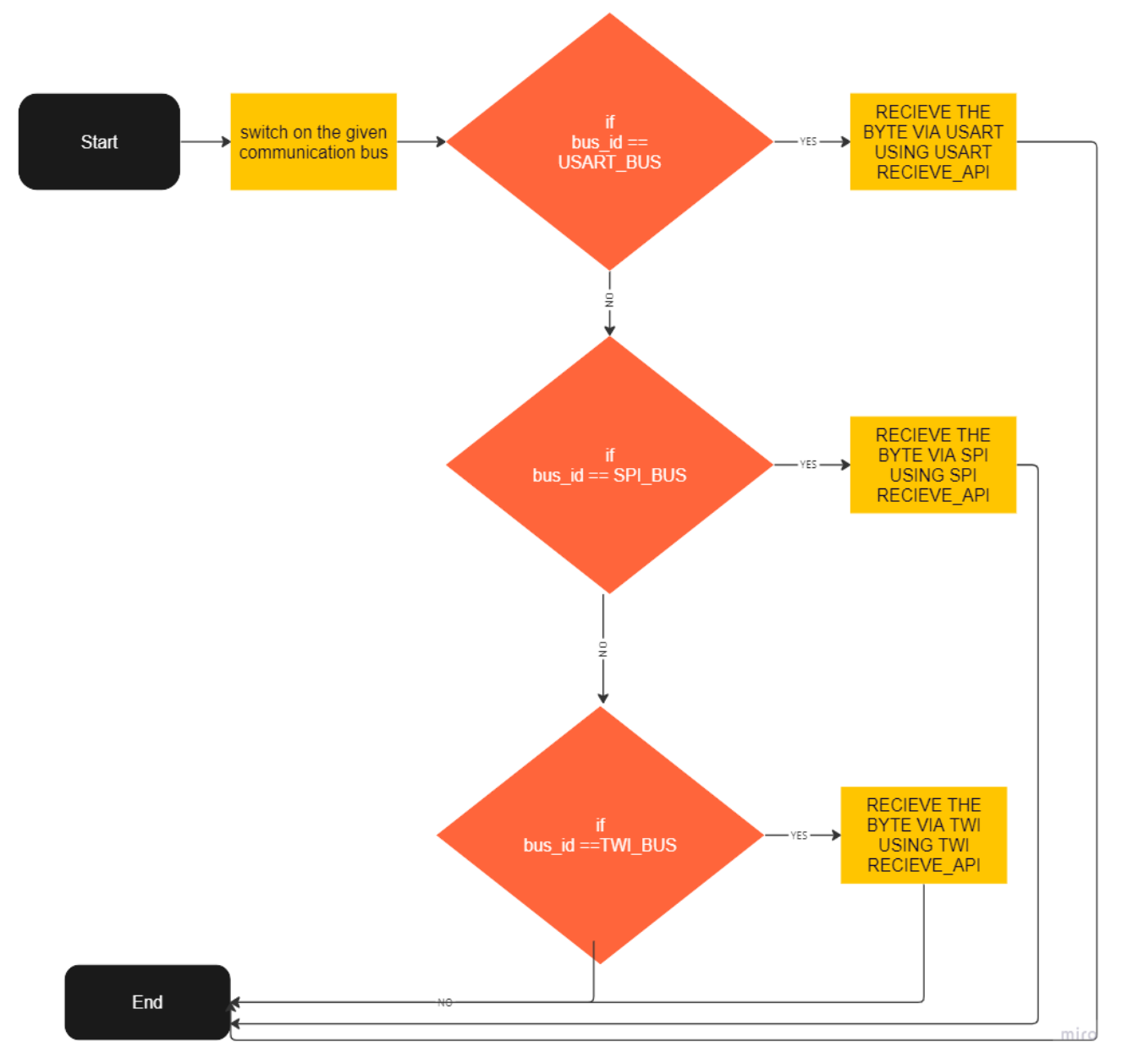
```
enu_system_status_t BCM_deinit(str_bcm_inctance_t *str_ptr_a_bcm_inctance);
```



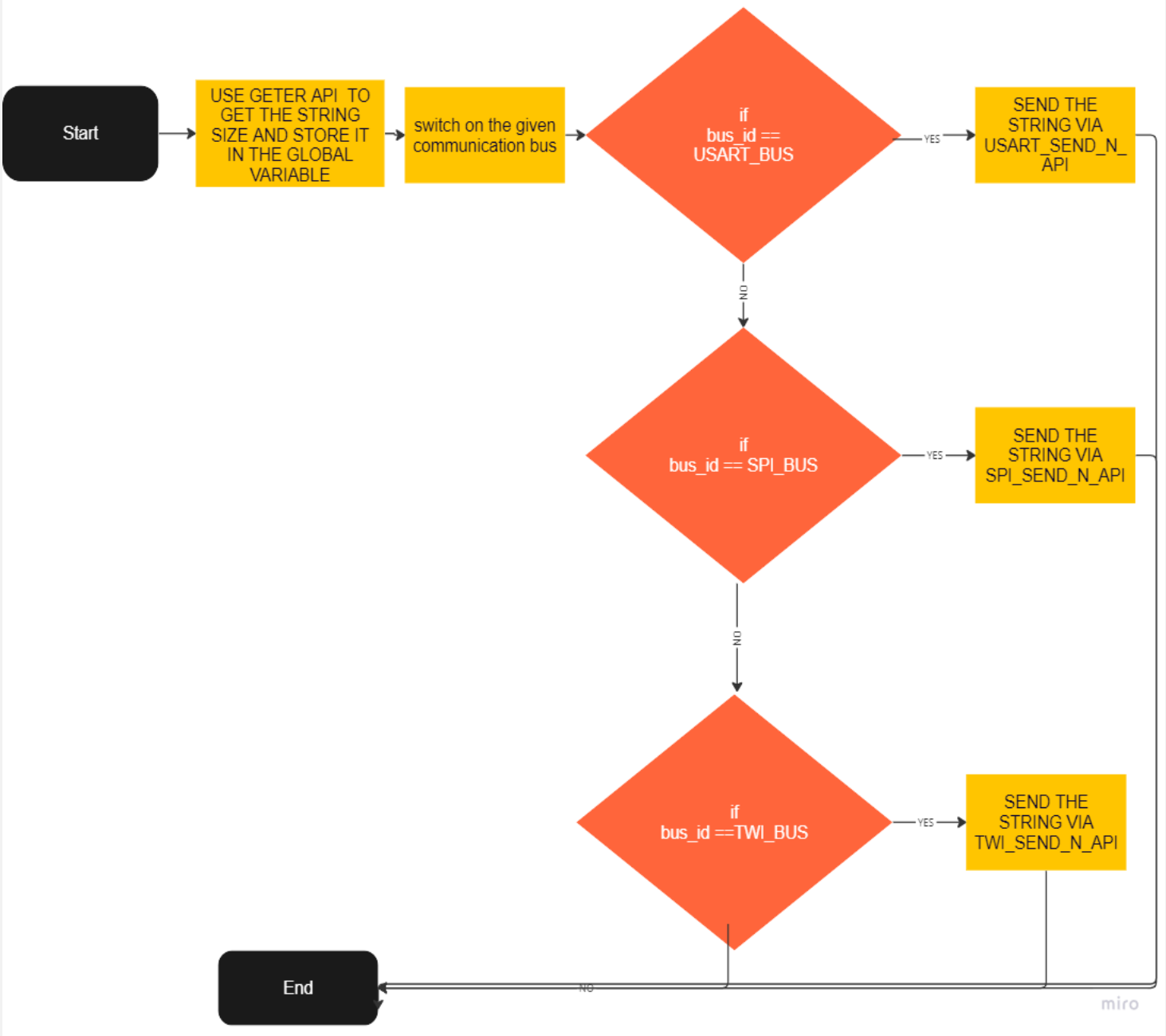
```
enu_system_status_t BCM_send(str_bcm_inctance_t *str_ptr_a_bcm_inctance , uint8_t
u8_a_data);
```



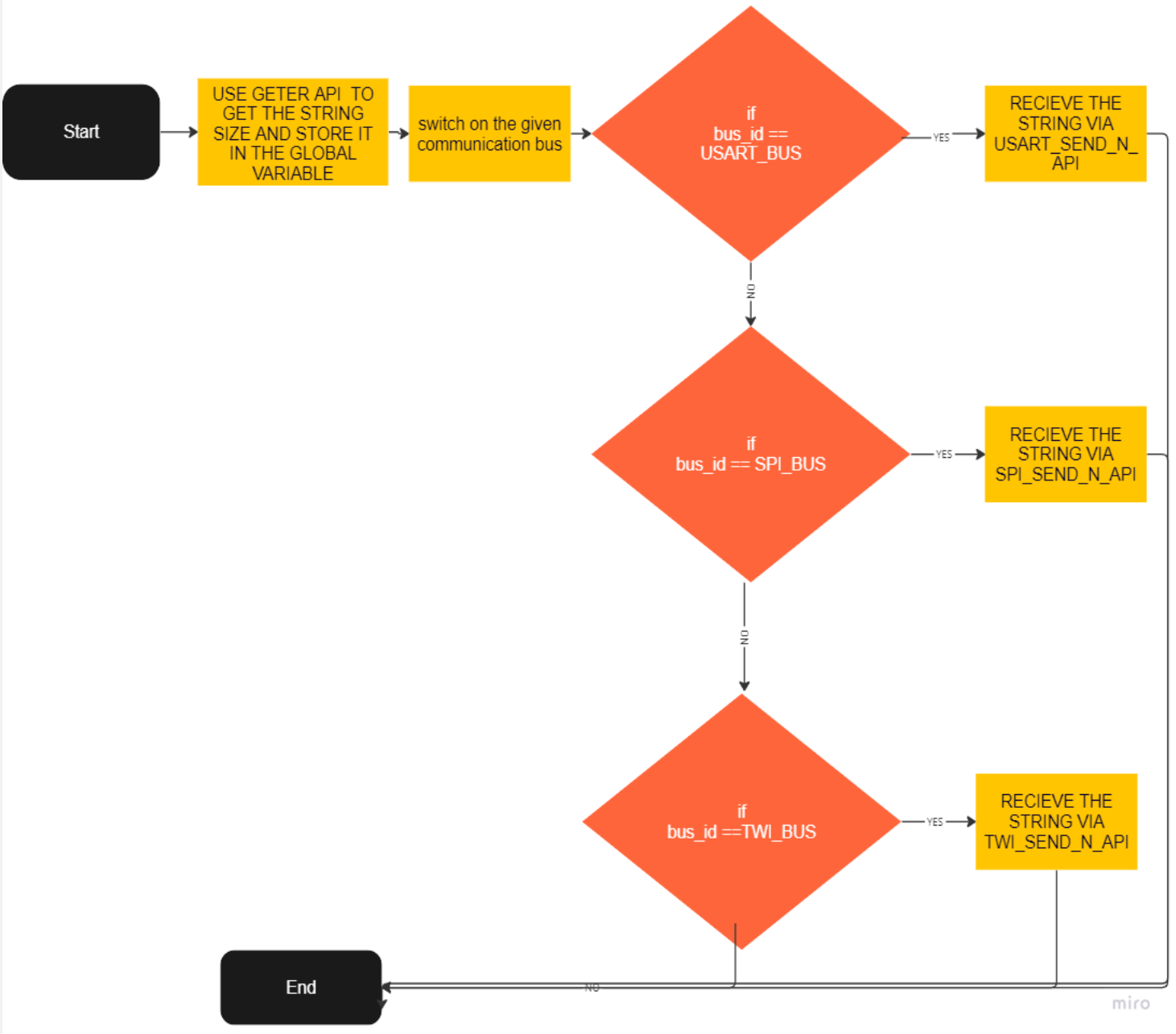
```
enu_system_status_t BCM_recieve(str_bcm_inctance_t *str_ptr_a_bcm_inctance , uint8_t
*u8_a_data);
```



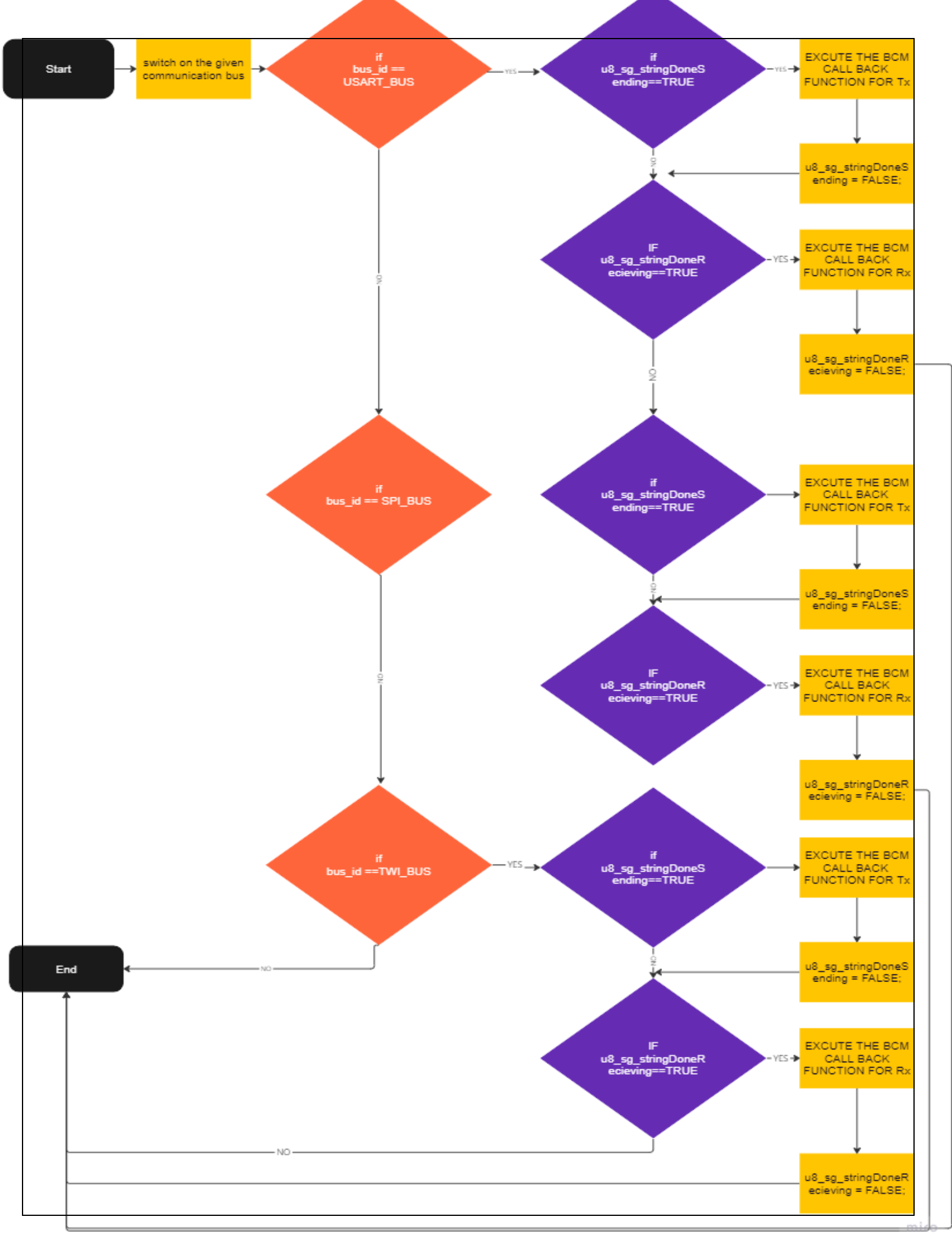
```
enu_system_status_t BCM_send_n(str_bcm_inctance_t *str_ptr_a_bcm_inctance , uint8_t
*u8Arr_a_stringData, uint16_t u16_a_stringSize);
```



```
enu_system_status_t BCM_recieve_n(str_bcm_inctance_t *str_ptr_a_bcm_inctance , uint8_t
*u8Arr_a_stringData, uint16_t u16_a_stringSize);
```




```
enu_system_status_t BCM_dispatcher(str_bcm_inctance_t *str_ptr_a_bcm_inctance );
```


Start
switch on the given communication bus
if bus_id == USART_BUS
YES
if u8_sg_stringDoneSending==TRUE
YES
EXCUTE THE BCM CALL BACK FUNCTION FOR Tx
u8_sg_stringDoneSending = FALSE;
NO
IF u8_sg_stringDoneRecieving==TRUE
YES
EXCUTE THE BCM CALL BACK FUNCTION FOR Rx
u8_sg_stringDoneRecieving = FALSE;
NO
NO
if bus_id == SPI_BUS
if u8_sg_stringDoneSending==TRUE
YES
EXCUTE THE BCM CALL BACK FUNCTION FOR Tx
u8_sg_stringDoneSending = FALSE;
NO
IF u8_sg_stringDoneRecieving==TRUE
YES
EXCUTE THE BCM CALL BACK FUNCTION FOR Rx
u8_sg_stringDoneRecieving = FALSE;
NO
if bus_id ==TWI_BUS
YES
if u8_sg_stringDoneSending==TRUE
YES
EXCUTE THE BCM CALL BACK FUNCTION FOR Tx
u8_sg_stringDoneSending = FALSE;
NO
IF u8_sg_stringDoneRecieving==TRUE
YES
EXCUTE THE BCM CALL BACK FUNCTION FOR Rx
u8_sg_stringDoneRecieving = FALSE;
NO
End

4. CONFIGURATION

4.1. USART CONFIGURATION

```
/* ----------------------------------------------------Data Type Declarations ---------
-------------------------------------------------*/

typedef  void (*Fptr_usartCallBack_t) (void);

typedef uint8_t u8_usartMode_t;

typedef uint8_t u8_usartTx_enable_t;
typedef uint8_t u8_usartRx_enable_t;

typedef uint8_t u8_usartTx_interruptMask_t;
typedef uint8_t u8_usartRx_interruptMask_t;

typedef uint8_t u8_usartStopBit_t;
typedef uint8_t u8_usartParityBit_t;
typedef uint8_t u8_usartDataSize_t;

typedef uint8_t u8_usartTxClkPolarity_t;
typedef uint8_t u8_usartRxClkPolarity_t;



typedef uint8_t u8_usartErorrState_t;

typedef struct{
	u8_usartMode_t usartMode;   /*@ref u8_usartMode_t*/
	u8_usartTx_enable_t usartTxEnable;/*@ref u8_usartTx_enable_t*/
	u8_usartTx_interruptMask_t usartTxInterrupt;    /*@ref
u8_usartTx_interruptMask_t*/
	u8_usartRx_enable_t usartRxEnable;/*@ref u8_usartRx_enable_t*/
	u8_usartRx_interruptMask_t usartRxInterrupt;    /*@ref
u8_usartRx_interruptMask_t*/
	u8_usartStopBit_t usartStopBitNum;/*@ref u8_usartStopBit_t*/
	u8_usartParityBit_t usartParityBit;      /*@ref u8_usartParityBit_t*/
	u8_usartDataSize_t usartDataSize;  /*@ref u8_usartDataSize_t*/
	u8_usartTxClkPolarity_t usartTxClkPolarity;            /*@ref
u8_usartTxClkPolarity_t*/
	u8_usartRxClkPolarity_t usartRxClkPolarity;            /*@ref
u8_usartRxClkPolarity_t*/
	uint32_t usartBaudRate; /*define the Buadrate value*/
}st_usart_config_t;

/* ------------------------------------------------- Macro Declarations ------------
-------------------------------------------------*/

/* USART Working Mode */
#define USART_ASYNCHRONOUS_NORMAL_SPEED_MODE            ((u8_usartMode_t)0x00)
#define USART_ASYNCHRONOUS_DOUBLE_SPEED_MODE            ((u8_usartMode_t)0x01)
#define USART_SYNCHRONOUS_MODE
((u8_usartMode_t)0x02)
```

```
#define USART_INVALID_MODE
((u8_usartMode_t)0x03)
/* USART Transmit Enable */
#define USART_TX_DISABLE            ((u8_usartTx_enable_t)0x00)
#define USART_TX_ENABLE                   ((u8_usartTx_enable_t)0x01)
/* USART Receiver Enable */
#define USART_RX_DISABLE            ((u8_usartRx_enable_t)0x00)
#define USART_RX_ENABLE                   ((u8_usartRx_enable_t)0x01)
/* USART Transmit Interrupt Enable Feature */
#define USART_TX_INTERRUPT_DISABLE                ((u8_usartTx_interruptMask_t)0x00)
#define USART_TX_INTERRUPT_ENABLE                 ((u8_usartTx_interruptMask_t)0x01)
/* EUSART Receiver Interrupt Enable Feature*/
#define USART_RX_INTERRUPT_DISABLE                ((u8_usartRx_interruptMask_t)0x00)
#define USART_RX_INTERRUPT_ENABLE                 ((u8_usartRx_interruptMask_t)0x01)
/*Select Number of stop-bit either one or two */
#define USART_ONE_STOP_BIT                ((u8_usartStopBit_t)0x00)
#define USART_TWO_STOP_BITS               ((u8_usartStopBit_t)0x01)
#define USART_INVALID_STOP_BITS           ((u8_usartStopBit_t)0x02)
/*Select Parity mode or disabled parity*/
#define USART_DISABLED_PARITY_BIT   ((u8_usartParityBit_t)0x00)
#define USART_EVEN_PARITY_BIT             ((u8_usartParityBit_t)0x01)
#define USART_ODD_PARITY_BIT              ((u8_usartParityBit_t)0x02)
#define USART_INVALID_PARITY_BIT    ((u8_usartParityBit_t)0x03)
/*Select the data-bit number*/
#define USART_FIVE_BIT_DATA               ((u8_usartDataSize_t)0x00)
#define USART_SIX_BIT_DATA                ((u8_usartDataSize_t)0x01)
#define USART_SEVEN_BIT_DATA              ((u8_usartDataSize_t)0x02)
#define USART_EIGHT_BIT_DATA              ((u8_usartDataSize_t)0x03)
#define USART_NINE_BIT_DATA               ((u8_usartDataSize_t)0x04)
#define USART_INVALID_BIT_DATA            ((u8_usartDataSize_t)0x05)
/*SELCET THE CLOCK POLARITY IN CASE OF SYNCHRONOUS MODE ONLY */
#define USART_SYNCHRONOUS_TX_RISING_XCK_EDGE            ((u8_usartTxClkPolarity_t)0x00)
#define USART_SYNCHRONOUS_TX_FALLING_XCK_EDGE           ((u8_usartTxClkPolarity_t)0x01)

#define USART_SYNCHRONOUS_RX_RISING_XCK_EDGE            ((u8_usartRxClkPolarity_t)0x00)
#define USART_SYNCHRONOUS_RX_FALLING_XCK_EDGE           ((u8_usartRxClkPolarity_t)0x01)

/*The Error state of The USART*/
#define USART_E_OK          ((u8_usartErorrState_t)0x00)
#define USART_E_NOK      ((u8_usartErorrState_t)0x01)


/*AN INDICATION TO TERMINATE RECIEVING BYTES AND STORE IT IN THE CHARCTER ARRAY (ASCII OF
ENTER)*/
#define END_OF_STRING_SYMPOL ((uint8_t)0x0D)
/* ------------------------------------------------- Macro Like Functions
Declarations ------------------------------------------------*/



#define  ENABLE_TX_INTERRUPT()                                  (SET_BIT(UCSRB,TXCIE))
#define  DISABLE_TX_INTERRUPT()                                 (CLEAR_BIT(UCSRB,TXCIE))

#define  ENABLE_EMPTY_DATA_REG_INTERRUPT()              (SET_BIT(UCSRB,UDRIE))
#define  DISABLE_EMPTY_DATA_REG_INTERRUPT()             (CLEAR_BIT(UCSRB,UDRIE))

#define  ENABLE_RX_INTERRUPT()                                  (SET_BIT(UCSRB,RXCIE))
#define  DISABLE_RX_INTERRUPT()                                 (CLEAR_BIT(UCSRB,RXCIE))
```

```c
/* ------------------------------------------------ Software Interfaces Declarations ---
-------------------------------------------------*/



st_usart_config_t st_g_usartObjForBcm = {
        .usartBaudRate=USART_CONFIG_BAUDRATE,
        .usartDataSize = USART_EIGHT_BIT_DATA,
        .usartMode = USART_ASYNCHRONOUS_NORMAL_SPEED_MODE,
        .usartParityBit = USART_DISABLED_PARITY_BIT,
        .usartRxEnable = USART_RX_ENABLE,
        .usartTxEnable = USART_TX_ENABLE,
        .usartRxInterrupt = USART_RX_INTERRUPT_ENABLE,
        .usartTxInterrupt = USART_TX_INTERRUPT_ENABLE,
        .usartStopBitNum = USART_ONE_STOP_BIT,
 };



#define USART_PRE_COMPILE_CONFIG_H_

#ifndef        F_CPU
# define F_CPU 8000000UL
#endif
#ifndef        USART_CONFIG_BAUDRATE
# define USART_CONFIG_BAUDRATE            9600UL
#endif


#ifndef        DATA_TO_SEND_MAX_BUFFER
# define DATA_TO_SEND_MAX_BUFFER   250U
#endif
#ifndef        DATA_TO_RECIVE_MAX_BUFFER
# define DATA_TO_RECIVE_MAX_BUFFER 250U
#endif



#endif /* USART_PRE_COMPILE_CONFIG_H_ */
```

4.2. LED CONFIGURATION

```
/*********************************  section 4: Data Type Declarations
*****************************************/
typedef enum{
       LED_OFF=0,
       LED_ON
}led_status_t;


typedef struct{
       pin_config_t led_pin;
       led_status_t led_status;
    }led_t;



led_t st_g_led0_instance = {
       .led_pin.pin=PIN0,
       .led_pin.port=PORTB_INDEX,
       .led_status = LED_OFF
};


led_t st_g_led1_instance = {
       .led_pin.pin=PIN1,
       .led_pin.port=PORTB_INDEX,
       .led_status = LED_OFF
    };
```

4.3. BCM CONFIGURATION

```
// typedef  void (*Fptr_bcmtCallBack_t) (uint16_t u16_a_sizeOfData);
typedef  void (*Fptr_bcmtCallBack_t) (void);


typedef enum {
	BCM_E_OK=0,
	BCM_E_NOK=2,
	}enu_system_status_t;
typedef enum {
	BCM_USART_BUS=0,
	BCM_SPI_BUS,
	BCM_TWI_BUS,
	BCM_INVALID_BUS_ID
}enu_bcm_busID_t;

 typedef enum {
	BCM_TRANSIMTER=0,
	BCM_RECIEVER,
	BCM_TRANSIMTER_RECIEVER,
	BCM_INVALID_OBERATION
 }enu_bcm_operation_t;

typedef struct{
	enu_bcm_busID_t bcm_busID; //@ref enu_bcm_busID_t
	enu_bcm_operation_t bcm_operation;//@ref enu_bcm_operation_t
	Fptr_bcmtCallBack_t  Fptr_bcmtTxCallBack;  //@ref Fptr_bcmtCallBack_t
	Fptr_bcmtCallBack_t  Fptr_bcmtRxCallBack;  //@ref Fptr_bcmtCallBack_t
	}str_bcm_inctance_t;



str_bcm_inctance_t str_g_bcm_inctance = {
	.bcm_busID = BCM_USART_BUS,
	.bcm_operation = BCM_TRANSIMTER_RECIEVER
};
```