BCM Project

Arafa Arafa Abd Elmawgod Ali

BCM project with protocols documentation

Sprints

Contents

Project Introduction:

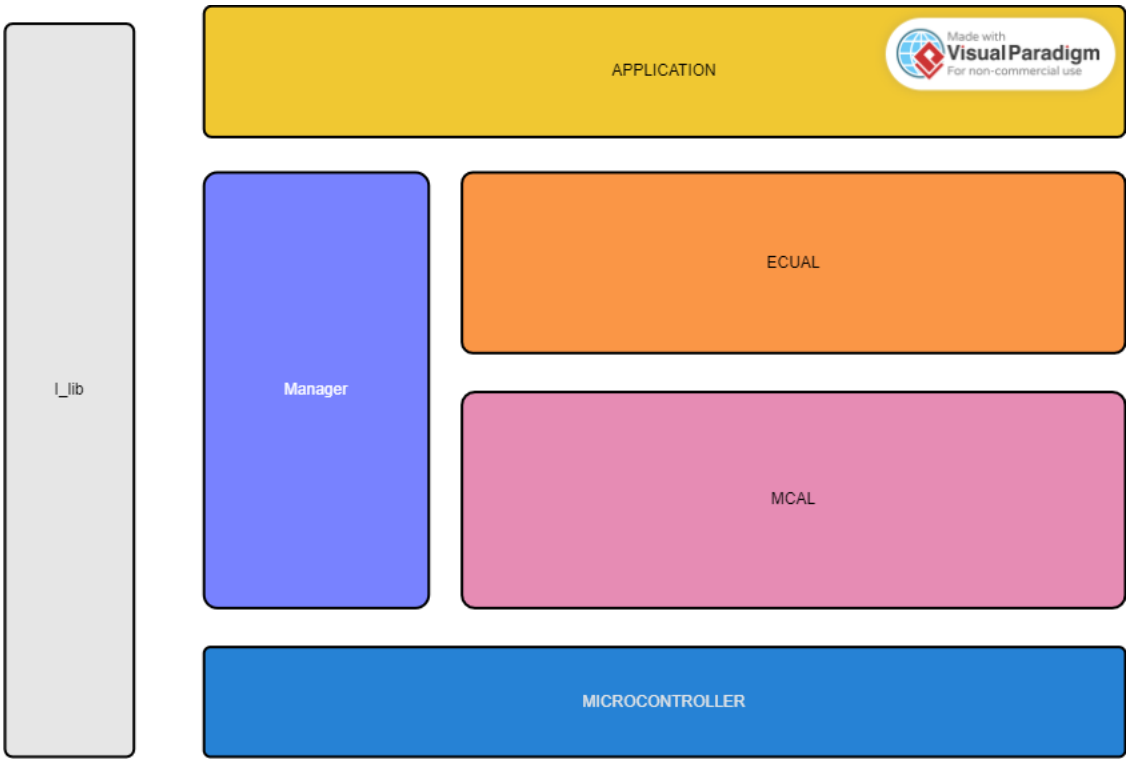
This project aims to implement a Communication Module (BCM) using the BCM Framework. The BCM is designed to facilitate data transmission and reception between different components or systems within a larger software application. It provides a flexible and efficient communication mechanism, supporting various communication protocols and data lengths up to 65535 bytes.

The implementation will be done using the C programming language, which offers low-level control and efficiency. Standard libraries and data structures will be utilized to ensure compatibility and optimal performance. The project will be developed and tested on an appropriate development environment, such as an Integrated Development Environment (IDE) or a text editor along with a compiler.

High Level Design:

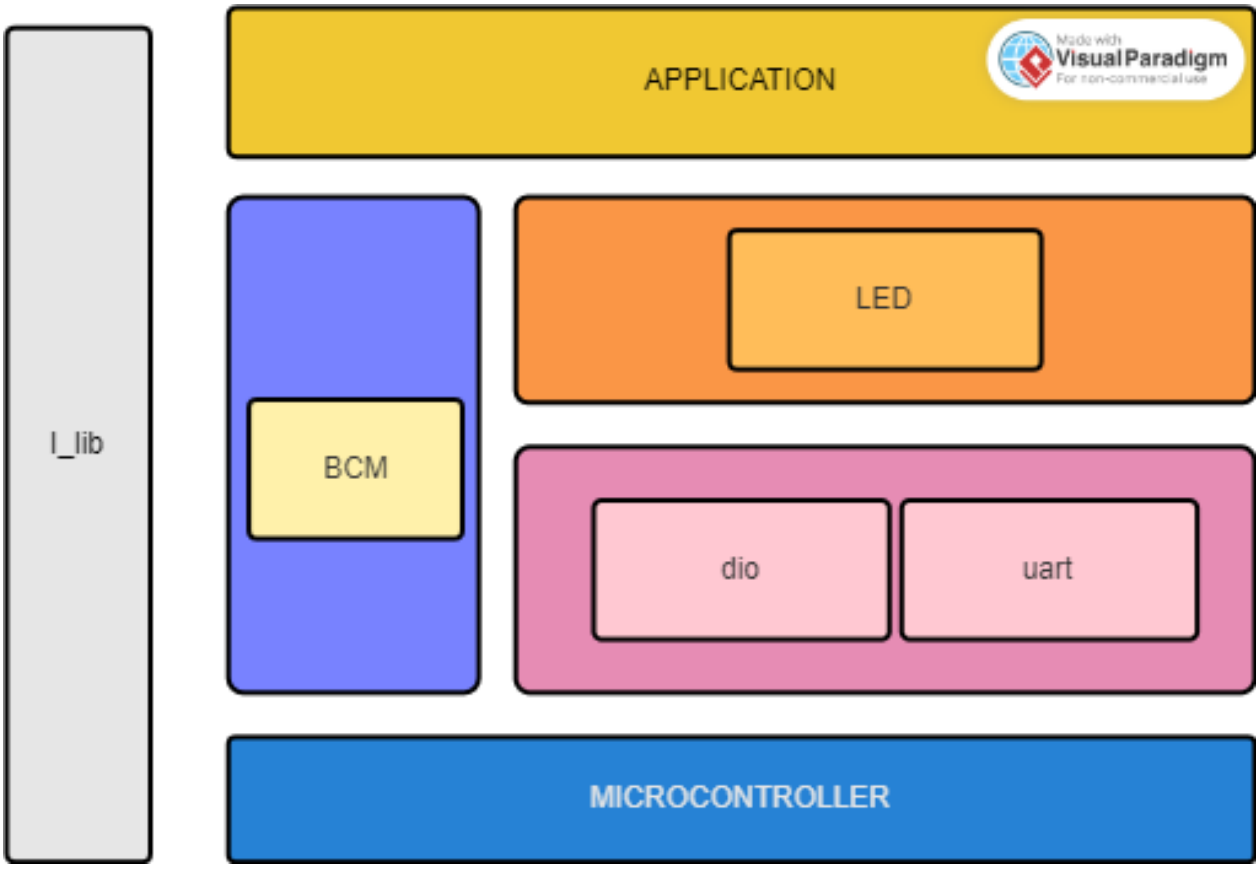
Layered architecture:

1. Application
2. Manager
3. ECUAL
4. MCAL
5. Microcontroller



Module Description

1. Application
2. ECUAL
    a. LED
3. Manager
    a. BCM
4. MCAL
    a. Dio
    b. Uart
5. Microcontroller



LED:

The module contains functions for initializing the LED, turning it on and off, and toggling its state.

To use this module, the **dio_interface.h** header file must be included. Additionally, the **str_dio_t** structure is used to configure the underlying digital input/output (DIO) pins associated with the LED.

Dependencies

- **dio_interface.h**: This header file defines the functions and data structures related to digital input/output (DIO) operations.

Data Types

enm_led_status_t

This enumerated type defines the possible states of the LED. It has the following values:

- **LED_ON**: Represents the LED being turned on (value: 1).
- **LED_OFF**: Represents the LED being turned off (value: 0).

str_led_t

This structure represents the LED and its associated properties. It contains the following members:

- **str_dio**: An instance of the **str_dio_t** structure that configures the DIO pins associated with the LED.
- **enm_led_status**: The current status of the LED, which can be either **LED_ON** or **LED_OFF**.

Functions

void LED_init(str_led_t* led)

This function initializes the LED by configuring the DIO pins and setting the initial LED status.

- **led**: A pointer to the **str_led_t** structure representing the LED to be initialized.

void LED_on(str_led_t* led)

This function turns the LED on by setting the appropriate DIO pin(s) to the active state.

- **led**: A pointer to the **str_led_t** structure representing the LED to be turned on.

void LED_off(str_led_t* led)

This function turns the LED off by setting the appropriate DIO pin(s) to the inactive state.

- **led**: A pointer to the **str_led_t** structure representing the LED to be turned off.

void LED_toggle(str_led_t* led)

This function toggles the state of the LED. If the LED is currently on, it will be turned off, and vice versa.

- **led**: A pointer to the **str_led_t** structure representing the LED to be toggled.

BCM Manager:

BCM (Communication Module) interface, which facilitates communication using different protocols such as UART, SPI, and I2C. The module includes functions for initializing and deinitializing the BCM, sending and receiving data, and executing periodic actions.

To use this module, the **std_types.h** header file must be included. The BCM operates on instances of the **str_bcm_instance_t** structure, which holds information about the communication protocol, instance ID, and specific protocol instance.

Dependencies

- **std_types.h**: This header file provides standard types used throughout the module.

Data Types

enm_cpo_t

This enumerated type defines the communication protocol options supported by the BCM. It has the following values:

- **BCM_PROTOCOL_UART**: Represents UART communication protocol (value: 0).
- **BCM_PROTOCOL_SPI**: Represents SPI communication protocol.
- **BCM_PROTOCOL_I2C**: Represents I2C communication protocol.
- **BCM_MAX_PROTOCOL**: Represents the maximum number of communication protocols supported.

enm_transiver_state_t

This enumerated type defines the states of the transceiver. It has the following values:

- **BCM_BUSY_FLAG**: Represents the transceiver being busy.
- **BCM_IDEL_FLAG**: Represents the transceiver being idle.

str_data_packet_t

This structure represents a data packet to be sent. It contains the following members:

- **ptr_data**: A pointer to the data buffer.
- **data_length**: The length of the data in the buffer.

str_rdata_packet_t

This structure represents a received data packet. It contains the following members:

- **ptr_data**: A pointer to the data buffer for storing received data.
- **data_length**: A pointer to a variable storing the length of the received data.

str_bcm_instance_t

This structure represents a BCM instance and its associated properties. It contains the following members:

- **bcm_instance_id**: The ID of the BCM instance.
- **protocol**: The communication protocol used by the instance (e.g., UART, SPI, I2C).
- **protocolInstance**: A pointer to the specific protocol instance.

Functions

enu_system_status_t bcm_init(str_bcm_instance_t* ptr_str_bcm_instance)

This function initializes the BCM module for a specific BCM instance.

- **ptr_str_bcm_instance**: A pointer to the **str_bcm_instance_t** structure representing the BCM instance to be initialized.

enu_system_status_t bcm_deinit(str_bcm_instance_t* ptr_str_bcm_instance)

This function deinitializes the BCM module for a specific BCM instance.

- **ptr_str_bcm_instance**: A pointer to the **str_bcm_instance_t** structure representing the BCM instance to be deinitialized.

enu_system_status_t bcm_send(str_bcm_instance_t* ptr_str_bcm_instance, uint8 *data)

This function sends a single byte of data over a specific BCM instance.

- **ptr_str_bcm_instance**: A pointer to the **str_bcm_instance_t** structure representing the BCM instance.
- **data**: A pointer to the data byte to be sent.

enu_system_status_t bcm_send_n(str_bcm_instance_t* ptr_str_bcm_instance, uint8* data, uint16 length)

This function sends multiple bytes of data over a specific BCM instance.

- **ptr_str_bcm_instance**: A pointer to the **str_bcm_instance_t** structure representing the BCM instance.
- **data**: A pointer to the data buffer to be sent.
- **length**: The length of the data buffer.

enu_system_status_t bcm_recive_n(str_bcm_instance_t* ptr_str_bcm_instance, uint8* data, uint16 *length)

This function receives multiple bytes of data over a specific BCM instance.

- **ptr_str_bcm_instance**: A pointer to the **str_bcm_instance_t** structure representing the BCM instance.
- **data**: A pointer to the buffer for storing received data.
- **length**: A pointer to a variable storing the maximum length of the received data. Upon completion, it will be updated with the actual length of the received data.

enu_system_status_t bcm_dispatcher(str_bcm_instance_t* ptr_str_bcm_instance,enm_transiver_state_t * state)

This function is a dispatcher that executes periodic actions and notifies events related to the BCM instance.

- **ptr_str_bcm_instance**: A pointer to the **str_bcm_instance_t** structure representing the BCM instance.
- **state**: A pointer to a variable storing the current state of the transceiver.

DIO:

Documentation: DIO (Digital Input/Output) Interface

Overview

DIO (Digital Input/Output) interface, which facilitates controlling and reading digital signals on specific pins and ports. The module includes functions for initializing pins, writing values to pins and ports, reading values from pins and ports, and toggling pin states.

To use this module, the **std_types.h** and **dio_private.h** header files must be included. The module defines enums for ports, pin values, pin directions, and DIO errors. It also includes a structure **str_dio_t** for representing a DIO pin.

Dependencies

- **std_types.h**: This header file provides standard types used throughout the module.
- **dio_private.h**: This header file provides private definitions and declarations for the DIO module.

Enums

enm_dio_port_t

This enumerated type defines the available ports for DIO pins. It has the following values:

- **PORT_A**: Represents Port A.
- **PORT_B**: Represents Port B.
- **PORT_C**: Represents Port C.
- **PORT_D**: Represents Port D.

enm_dio_value_t

This enumerated type defines the possible values for a DIO pin. It has the following values:

- **DIO_LOW**: Represents a low logic level (value: 0).
- **DIO_HIGH**: Represents a high logic level.

enm_dio_dir_t

This enumerated type defines the possible directions for a DIO pin. It has the following values:

- **DIO_IN**: Represents the input direction (value: 0).
- **DIO_OUT**: Represents the output direction.

enm_dio_error_t

This enumerated type defines the possible errors that can occur during DIO operations. It has the following values:

- **DIO_FAIL**: Represents a failure or error (value: 0).
- **DIO_SUCCESS**: Represents a successful operation.

Structures

str_dio_t

This structure represents a DIO pin and its associated properties. It contains the following members:

- **port**: The port to which the pin belongs (**enm_dio_port_t**).

- **pin**: The number of the pin within the port.

Functions

enm_dio_error_t dio_init(str_dio_t dio_pin, enm_dio_dir_t dir)

This function initializes a DIO pin with the specified direction.

- **dio_pin**: The **str_dio_t** structure representing the DIO pin to be initialized.
- **dir**: The desired direction for the pin (**DIO_IN** or **DIO_OUT**).

enm_dio_error_t dio_write_pin(str_dio_t dio_pin, enm_dio_value_t value)

This function writes a value to a DIO pin.

- **dio_pin**: The **str_dio_t** structure representing the DIO pin to be written to.
- **value**: The value to be written to the pin (**DIO_LOW** or **DIO_HIGH**).

enm_dio_error_t dio_toggle(str_dio_t dio_pin)

This function toggles the state of a DIO pin. If the pin is currently high, it will be set to low, and vice versa.

- **dio_pin**: The **str_dio_t** structure representing the DIO pin to be toggled.

enm_dio_error_t dio_read_pin(str_dio_t dio_pin, uint8 *value)

This function reads the value of a DIO pin and stores it in the provided variable.

- **dio_pin**: The **str_dio_t** structure representing the DIO pin to be read.
- **value**: A pointer to a variable where the pin value will be stored (**DIO_LOW** or **DIO_HIGH**).

enm_dio_error_t dio_write_port(enm_dio_port_t port, enm_dio_value_t value)

This function writes a value to the specified DIO port. The value will be applied to all pins of the port.

- **port**: The port to which the value will be written (**PORT_A**, **PORT_B**, **PORT_C**, or **PORT_D**).
- **value**: The value to be written to the port (**DIO_LOW** or **DIO_HIGH**).

enm_dio_error_t dio_read_port(enm_dio_port_t port, uint8 *data)

This function reads the value of a DIO port and stores it in the provided variable. The value represents the combined state of all pins in the port.

- **port**: The port to be read (**PORT_A**, **PORT_B**, **PORT_C**, or **PORT_D**).

- **data**: A pointer to a variable where the port value will be stored.

UART:

UART (Universal Asynchronous Receiver Transmitter) interface, which enables serial communication between devices. The module includes enums for various UART configurations and a structure **uart_config_t** to represent the UART configuration settings. Additionally, it defines functions for initializing the UART, writing and reading data, and enabling/disabling UART interrupts.

To use this module, the **std_types.h** header file must be included.

Enums

uart_receive_mode_t

This enumerated type defines the receive mode options for UART. It has the following values:

- **UART_RECEIVE_DISABLE**: Disable receive.
- **UART_RECEIVE_ENABLE**: Enable receive.

uart_transmit_mode_t

This enumerated type defines the transmit mode options for UART. It has the following values:

- **UART_TRANSMIT_DISABLE**: Disable transmit.
- **UART_TRANSMIT_ENABLE**: Enable transmit.

uart_udre_interrupt_mode_t

This enumerated type defines the interrupt mode options for UART's Data Register Empty (UDRE) interrupt. It has the following values:

- **UART_UDRE_INTERRUPT_DISABLE**: Disable the interrupt.
- **UART_UDRE_INTERRUPT_ENABLE**: Enable the interrupt.

uart_rxc_interrupt_mode_t

This enumerated type defines the interrupt mode options for UART's Receive Complete (RXC) interrupt. It has the following values:

- **UART_RXC_INTERRUPT_DISABLE**: Disable the interrupt.
- **UART_RXC_INTERRUPT_ENABLE**: Enable the interrupt.

uart_txc_interrupt_mode_t

This enumerated type defines the interrupt mode options for UART's Transmit Complete (TXC) interrupt. It has the following values:

- **UART_TXC_INTERRUPT_DISABLE**: Disable the interrupt.

- **UART_TXC_INTERRUPT_ENABLE**: Enable the interrupt.

uart_rx_mode_t

This enumerated type defines the receive mode options for UART. It has the following values:

- **UART_RX_DISABLE**: Disable receive.
- **UART_RX_ENABLE**: Enable receive.

uart_tx_mode_t

This enumerated type defines the transmit mode options for UART. It has the following values:

- **UART_TX_DISABLE**: Disable transmit.
- **UART_TX_ENABLE**: Enable transmit.

uart_speed_mode_t

This enumerated type defines the speed mode options for UART. It has the following values:

- **UART_SYNC_SPEED_MODE**: Synchronous mode.
- **UART_NORMAL_MODE**: Normal mode.
- **UART_DOUBLE_MODE**: Double speed mode.

uart_clock_polarity_t

This enumerated type defines the clock polarity options for UART. It has the following values:

- **UART_NO_CLOCK**: No clock in asynchronous mode.
- **UART_TXR_RXF**: Transmit rising, receive falling.
- **UART_TXF_RXR**: Transmit falling, receive rising.

uart_stop_mode_t

This enumerated type defines the stop bit options for UART. It has the following values:

- **UART_STOP_1_BIT**: One stop bit.
- **UART_STOP_2_BIT**: Two stop bits.

uart_parity_mode_t

This enumerated type defines the parity mode options for UART. It has the following values:

- **UART_PARITY_DISABLED**: Parity disabled.
- **UART_PARITY_EVEN**: Even parity mode.

- **UART_PARITY_ODD**: Odd parity mode.

uart_operating_mode_t

This enumerated type defines the operating mode options for UART. It has the following values:

- **UART_ASYNC_MODE**: Asynchronous mode.
- **UART_SYNC_MODE**: Synchronous mode.

uart_data_size_t

This enumerated type defines the data size options for UART. It has the following values:

- **UART_CS_5**: 5 bits length.
- **UART_CS_6**: 6 bits length.
- **UART_CS_7**: 7 bits length.
- **UART_CS_8**: 8 bits length.
- **UART_CS_9**: 9 bits length.

Structures

uart_config_t

This structure represents the configuration settings for the UART module. It contains the following members:

- **uart_mode**: The operating mode of the UART (asynchronous or synchronous).
- **uart_data_size**: The number of bits in a data frame.
- **uart_parity_mode**: The parity mode for error detection.
- **uart_stop_mode**: The number of stop bits.
- **uart_clock_polarity**: The clock polarity in asynchronous mode.
- **uart_speed_mode**: The speed mode (normal, double, or synchronous).
- **uart_receive_mode**: The receive mode (enable or disable).
- **uart_transmit_mode**: The transmit mode (enable or disable).
- **uart_udre_interrupt_mode**: The interrupt mode for Data Register Empty (enable or disable).
- **uart_rx_mode**: The receive mode (enable or disable).
- **uart_tx_mode**: The transmit mode (enable or disable).

- **uart_rxc_interrupt_mode**: The interrupt mode for Receive Complete (enable or disable).
- **uart_txc_interrupt_mode**: The interrupt mode for Transmit Complete (enable or disable).
- **uart_baudrate**: The desired baud rate for communication.

Function Prototypes

void uart_init(uart_config_t *uart_config)

This function initializes the UART module with the specified configuration.

- **uart_config**: A pointer to a **uart_config_t** structure containing the desired UART configuration settings.

void uart_write(uint16 *data)

This function writes a single character of data to the UART for transmission.

- **data**: A pointer to the data to be transmitted.

void uart_read(uint16 *data)

This function reads a single character of data from the UART.

- **data**: A pointer to a variable where the received data will be stored.

void uart_write_INT(void(*callback)(void))

This function enables interrupt-driven UART transmission. The provided callback function will be called when the UART is ready to transmit data.

- **callback**: A function pointer to the callback function that will be executed when the UART is ready to transmit data.

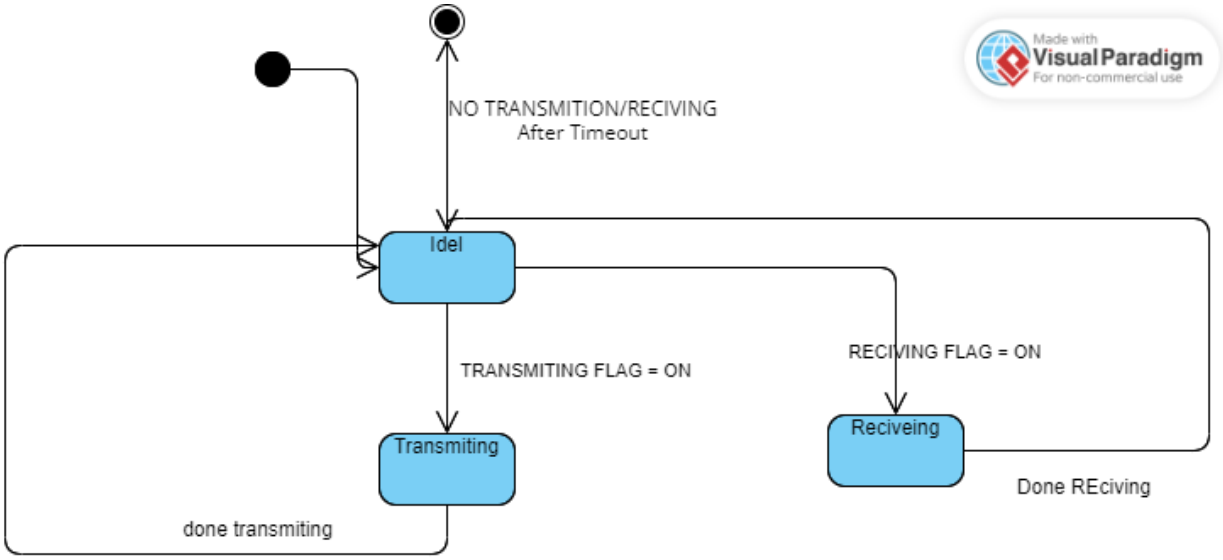
void uart_read_INT(void(*callback)(void))

This function enables interrupt-driven UART reception. The provided callback function will be called when data is received.

- **callback**: A function pointer to the callback function that will be executed when data is received.

UML:

State Machine:



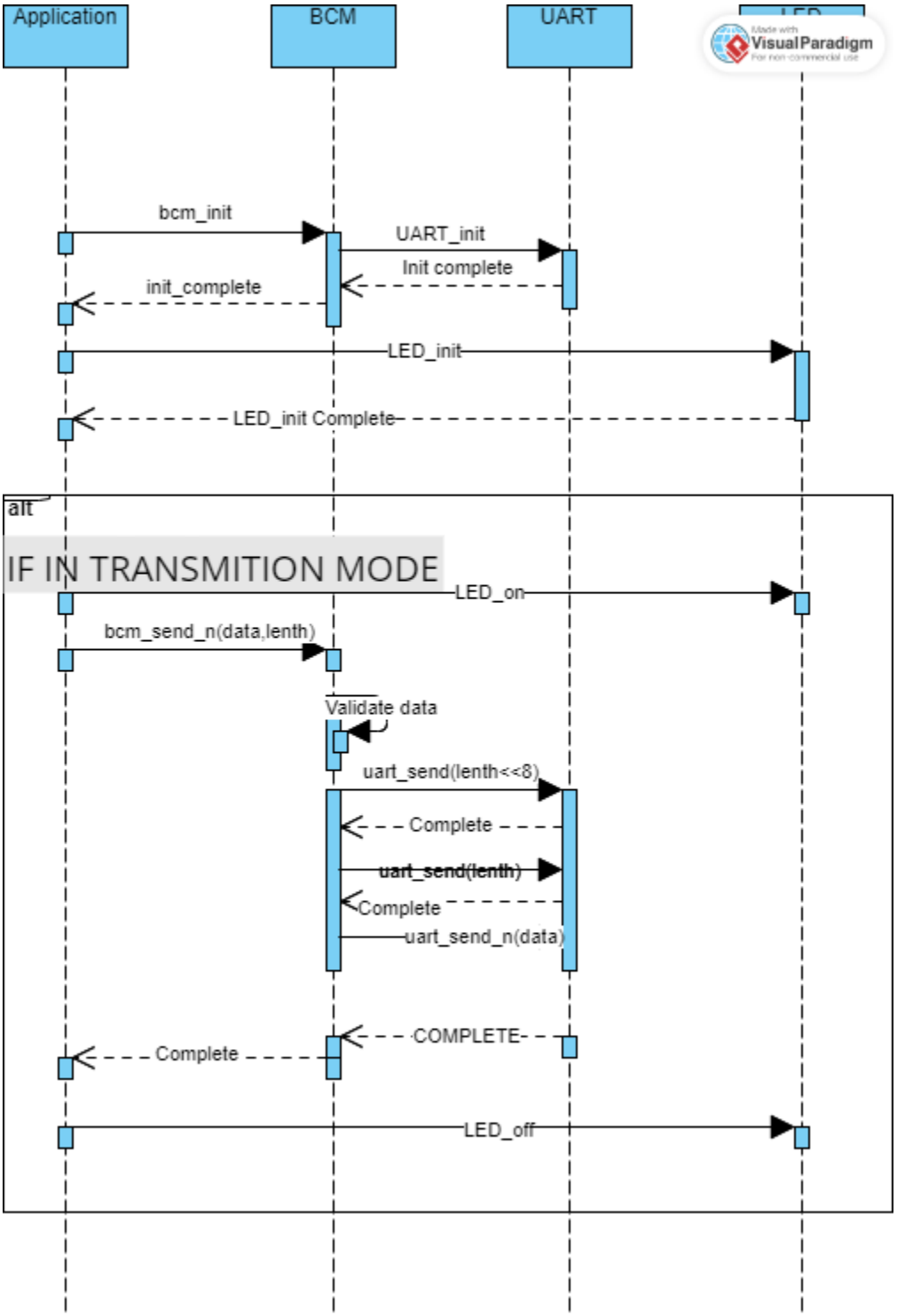
The state machine for the application consists of three states:

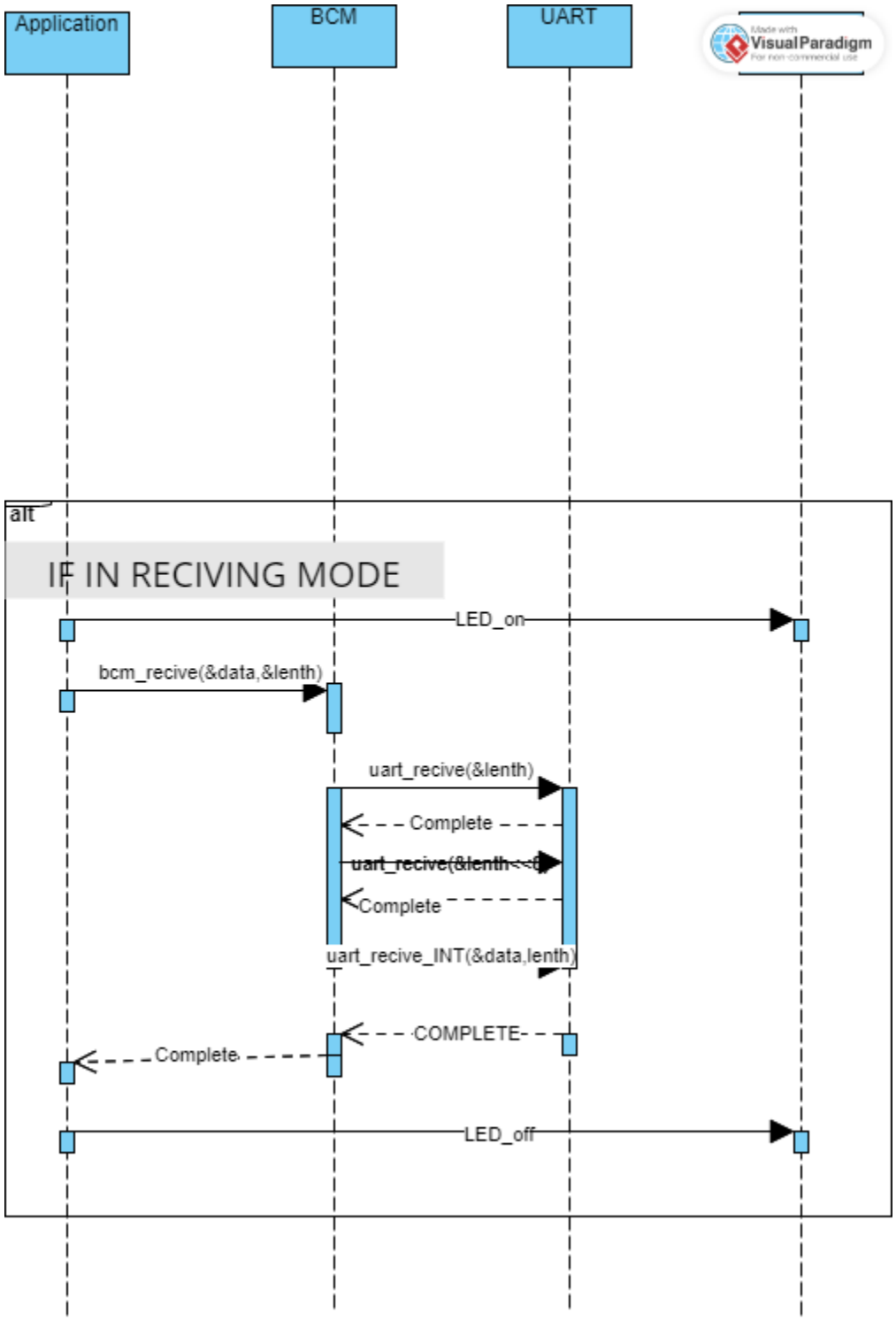
1. STATE_IDLE:
    - This state represents the idle state of the system.
    - When the state machine is in this state, it waits for the EVENT_START event to occur.
    - Upon receiving the EVENT_START event, it transitions to the STATE_TRANSMIT state.
2. STATE_TRANSMIT:
    - This state represents the transmit state of the system.
    - When the state machine is in this state, it performs the transmission operation using the BCM interface.
    - After completing the transmission, it waits for the EVENT_TRANSMIT_COMPLETE event to occur.
    - Upon receiving the EVENT_TRANSMIT_COMPLETE event, it transitions to the STATE_RECEIVE state.
3. STATE_RECEIVE:

- This state represents the receive state of the system.
- When the state machine is in this state, it performs the receive operation using the BCM interface.
- After completing the receive operation, it waits for the EVENT_RECEIVE_COMPLETE event to occur.
- Upon receiving the EVENT_RECEIVE_COMPLETE event, it transitions back to the STATE_TRANSMIT state.

Sprints

Sequence Diagram




Application
BCM
UART
alt
IF IN RECIVING MODE
LED_on
bcm_recive(&data,&lenth)
uart_recive(&lenth)
Complete
Complete
uart_recive_INT(&data,lenth)
COMPLETE
Complete
LED_off

Sprints

Low Level Design:

Flowchart

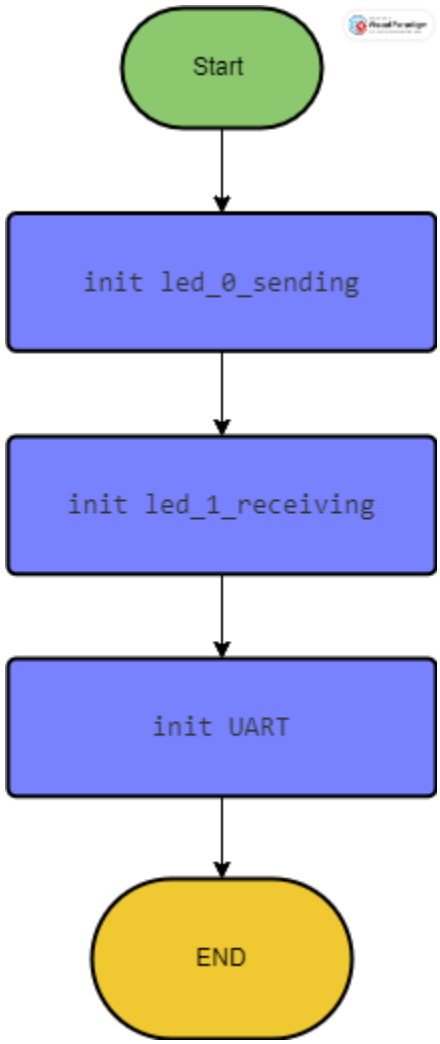


Figure 1app_init

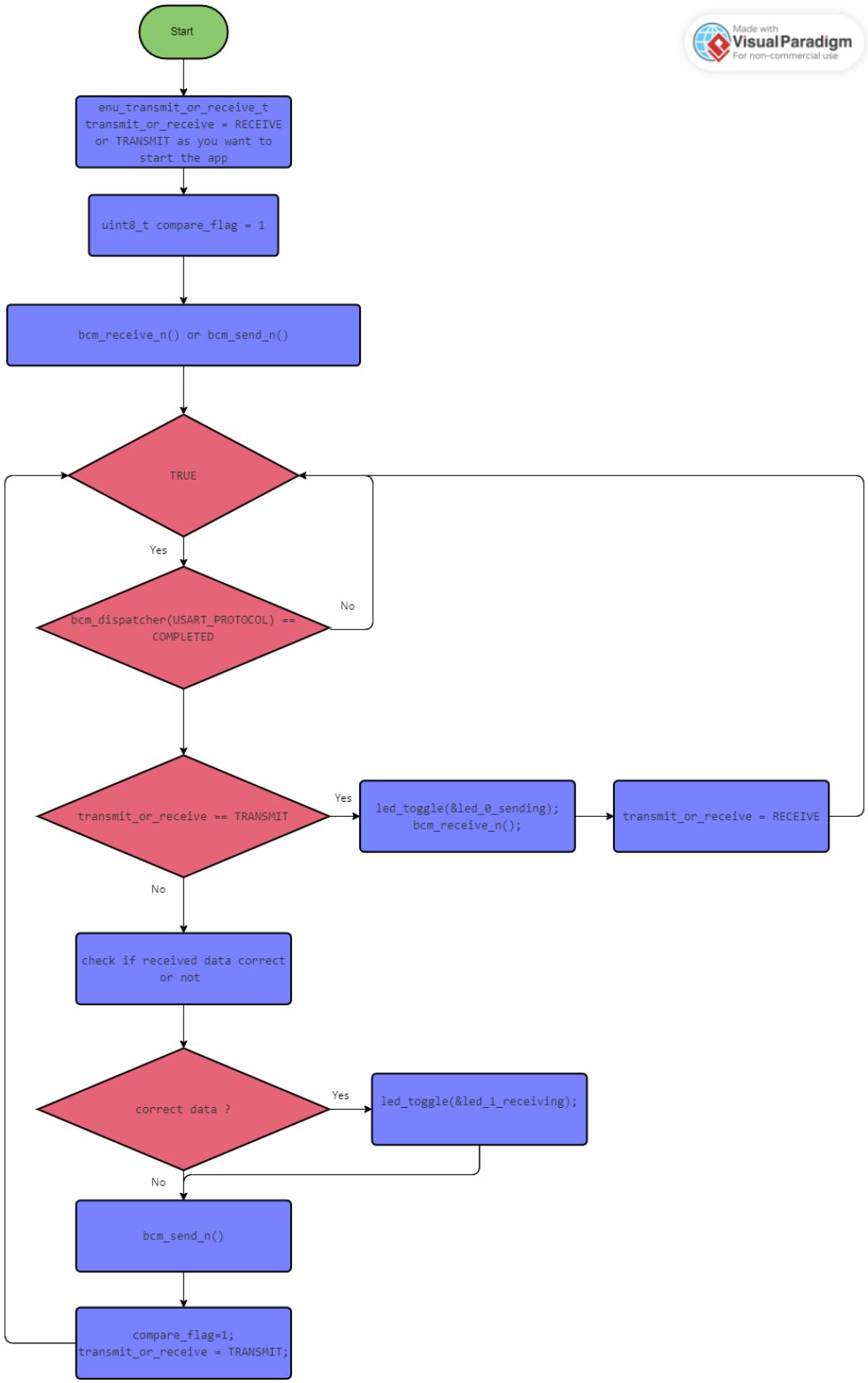


Figure 2app_run

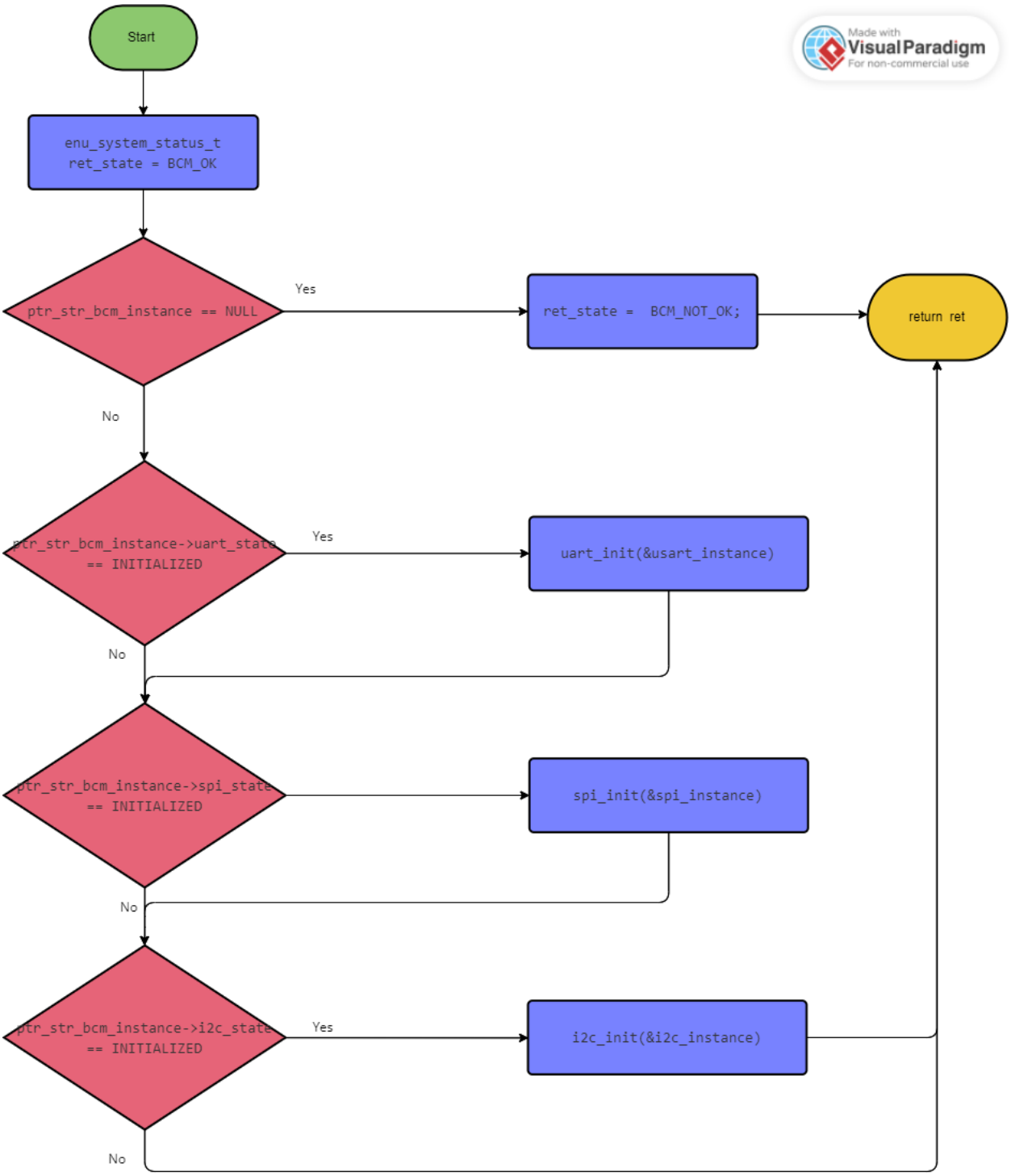


Figure 3 bcm_init.vpd

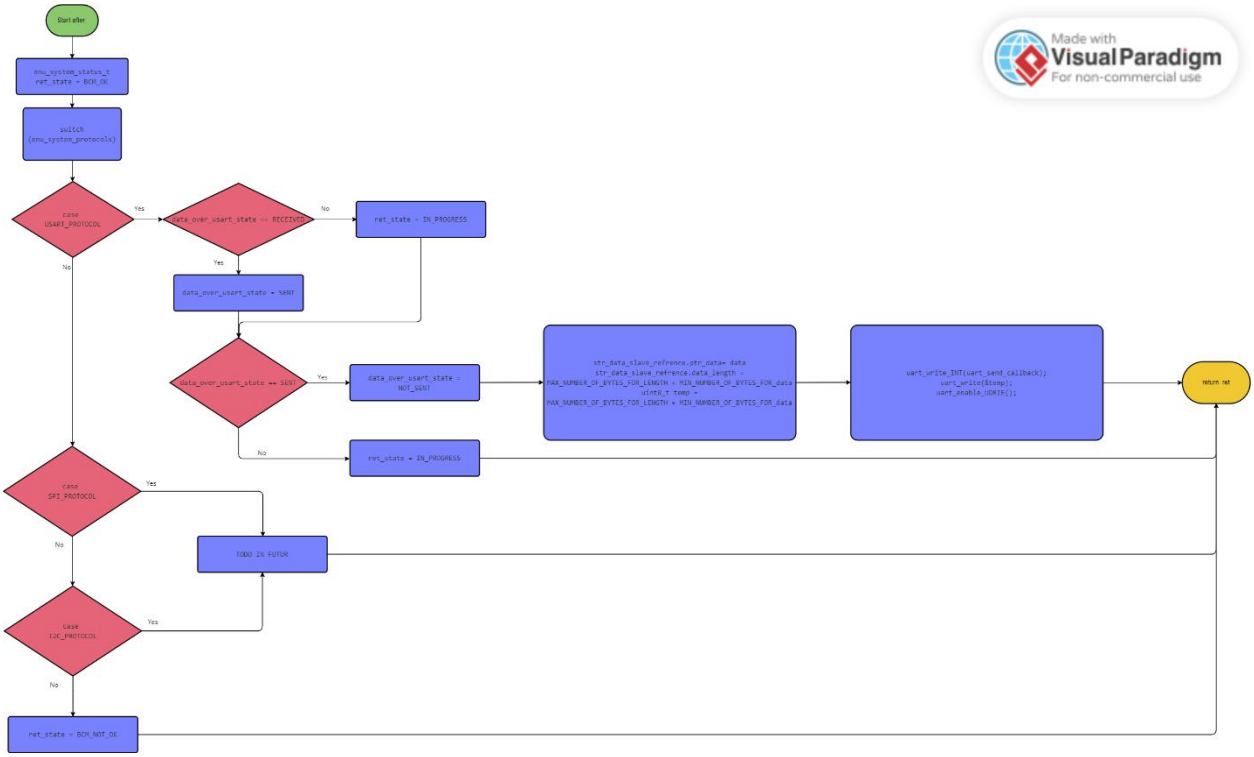


Figure 4 bcm_send.vpd

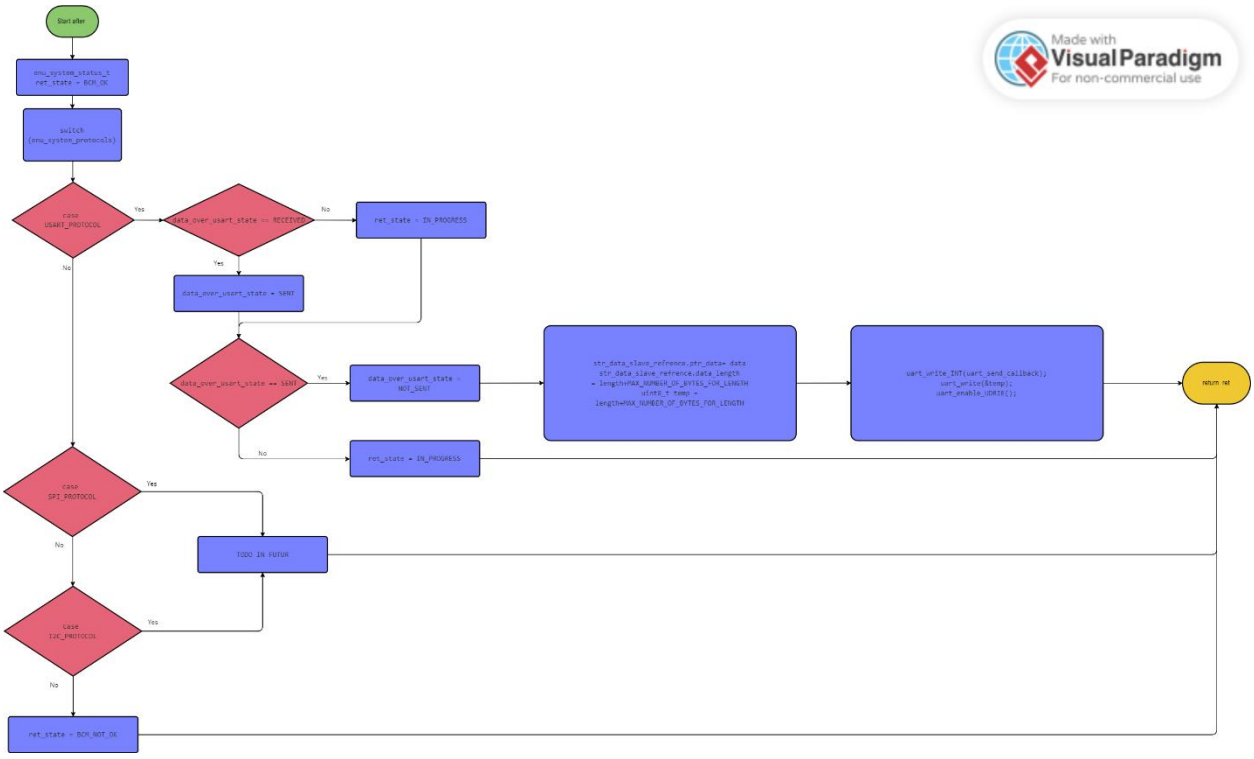


Figure 5 bcm_send_n.vpd

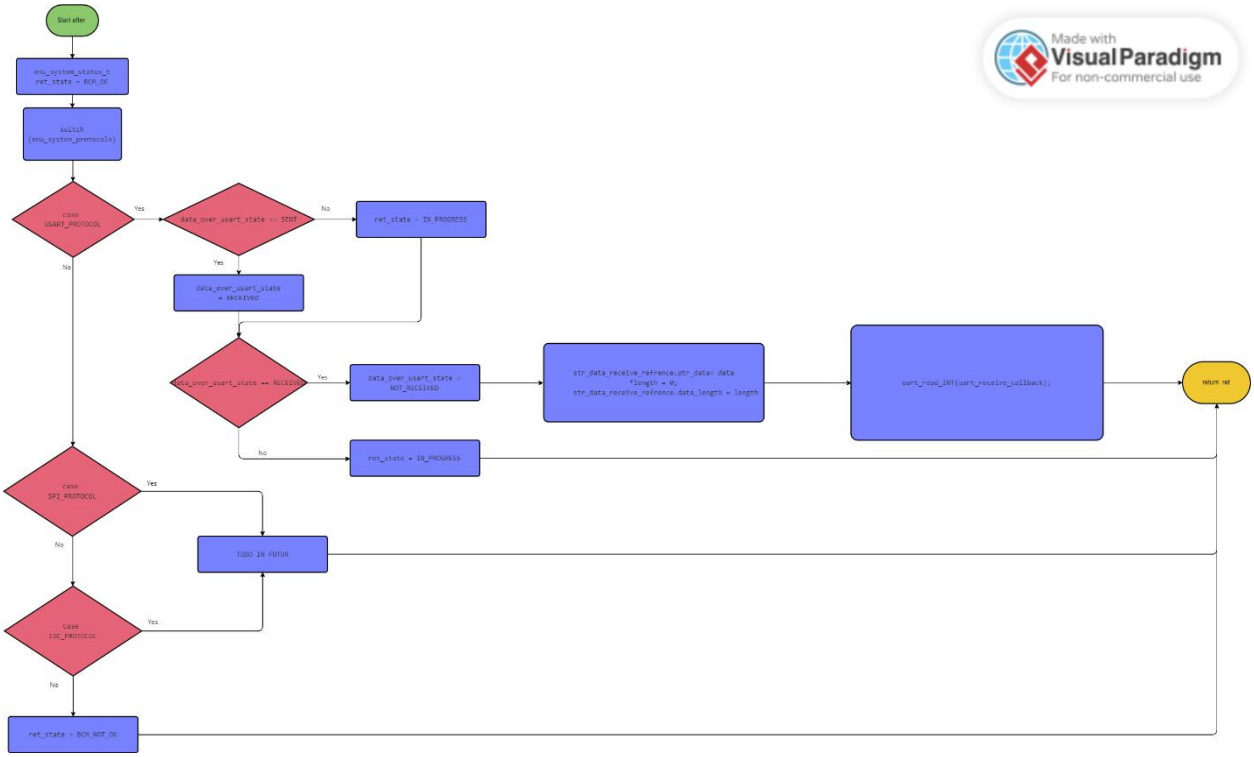


Figure 6 bcm_receive_n.vpd

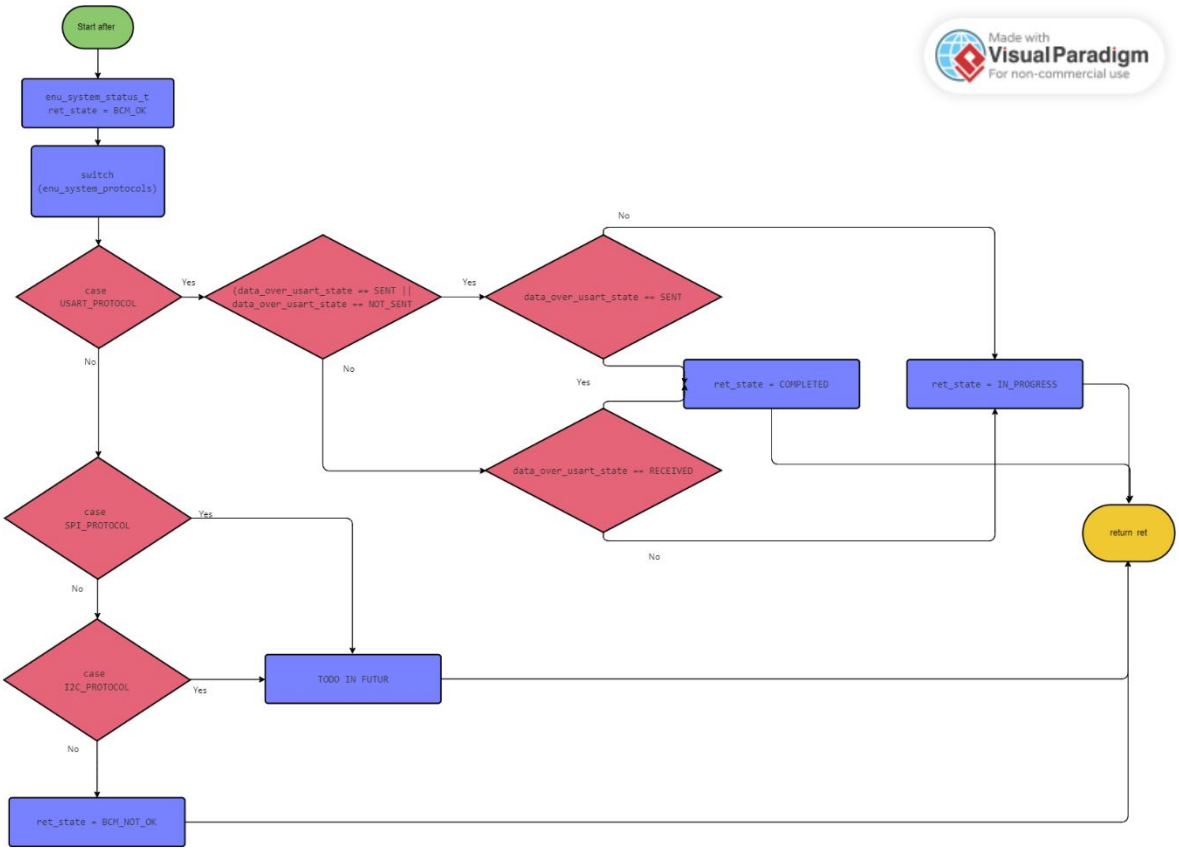


Figure 7 bcm_dispatcher.vpd

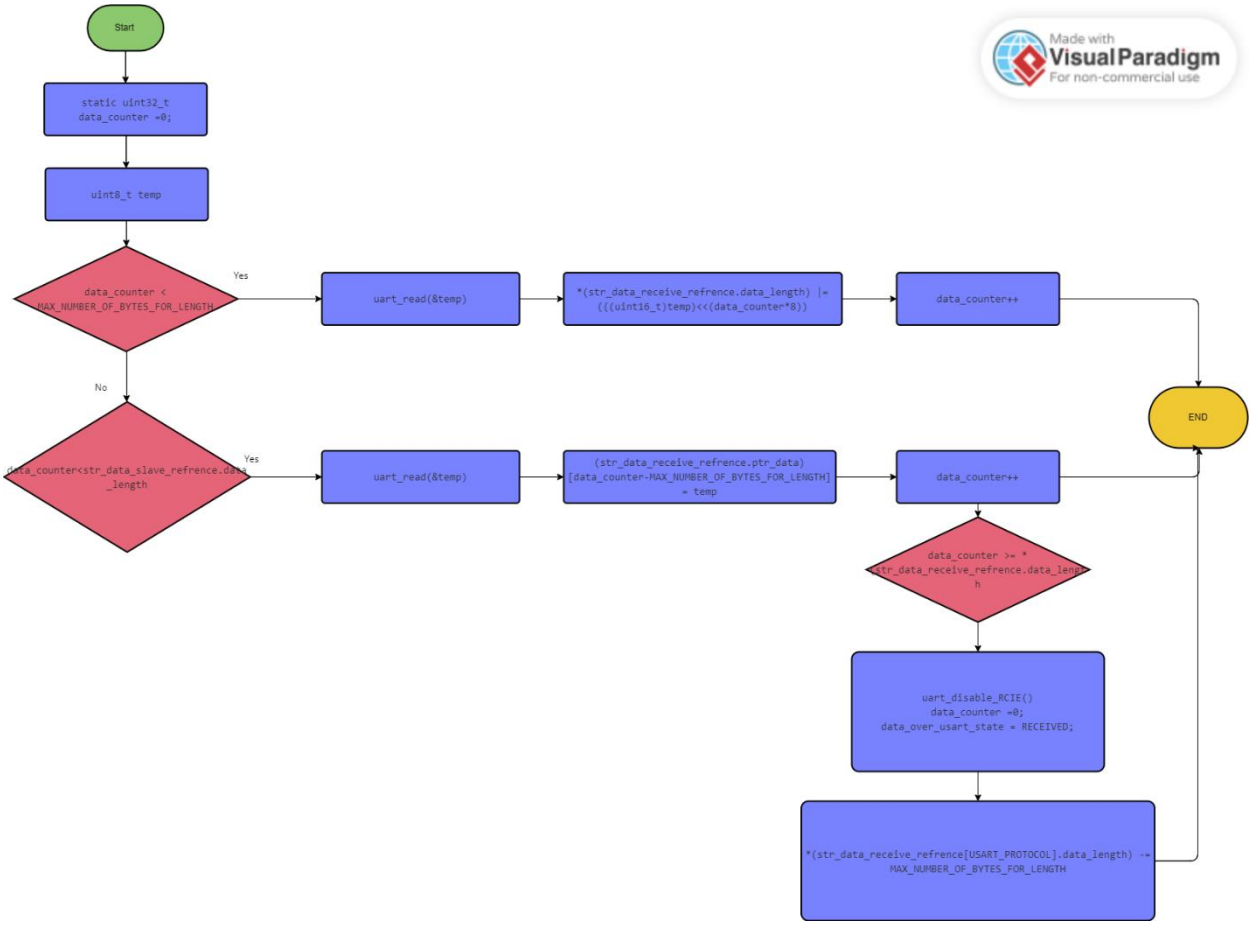


Figure 8 uart_receive_callback.vpd

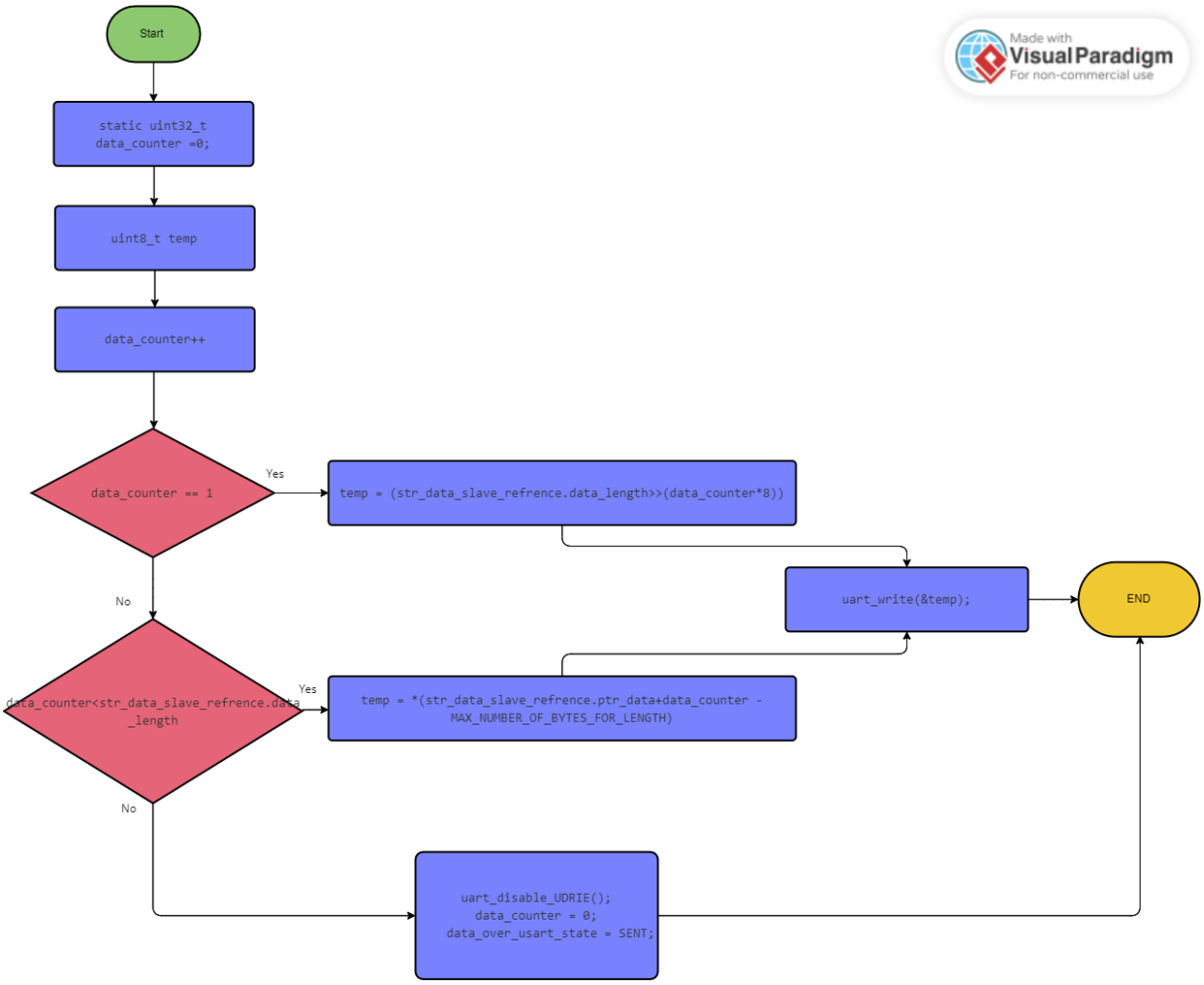


Figure 9 uart_send_callback.vpd

Pre-compline

Application

```
/* to define the maximum size of array*/
#define  BUFFER_MAX_SIZE 50
```

BCM

```
/************the maximum number of protocols that can be used with BCM**********/
#define MAX_PROTOCOL_COUNTER 3
/********the maximum number of bytes to describe the length of data will send*********/
#define MAX_NUMBER_OF_BYTES_FOR_LENGTH 2
/************ the minimum number of bytes of data will send **********/
#define MIN_NUMBER_OF_BYTES_FOR_data 1
```

Linking configuration

BCM

```
/*
*
*
* -enu_protocol_state_t datatype enum has all initialization state
*   -Members-
*
* -1-INITIALIZED
* -2-NOT_REQUIRED
*
*/
typedef enum{
	INITIALIZED = 0,
	NOT_REQUIRED

}enu_protocol_state_t;
/*
*
*
* -str_data_and_slave_instance_t datatype hold the data to transmit and the wanted
slave if wanted for protocol
*   -Members-
* -1-  (uint8_t * ptr_data)  pointer to the data
* -2-  (uint16_t data_length) the length of data
*
*/
typedef struct
{
	uint8_t * ptr_data;
	uint16_t data_length;

}str_data_and_slave_instance_t;

/*
*
*
* -str_data_instance_t datatype hold the address of data queue to receive and the
length of data
*   -Members-
* -1-  (uint8_t * ptr_data)  pointer to the data
* -2-  (uint16_t *data_length) pointer to the length of data
* -3-  (void *slave_id) void pointer to hold slave info address
*
*/
typedef struct
{
	uint8_t *ptr_data;
	uint16_t *data_length;
```

```
}str_data_instance_t;

/*
*
*
* -enu_system_status_t datatype enum has all system return states
*   -Members-
*
* -1-BCM_NOT_OK
* -2-BCM_OK
* -3-IN_PROGRESS
* -4-COMPLETED
*
*/
typedef enum{
       BCM_NOT_OK =0,
       BCM_OK,
       IN_PROGRESS,
       COMPLETED
}enu_system_status_t;

/*
*
*
* -enu_system_protocols_t datatype enum has all types of protocol that can be used with
BCM
*   -Members-
*
* -1-USART_PROTOCOL
* -2-SPI_PROTOCOL
* -3-I2C_PROTOCOL
*
*/
typedef enum{
       USART_PROTOCOL =0,
       SPI_PROTOCOL,
       I2C_PROTOCOL
}enu_system_protocols_t;

/*
*
*
* -enu_data_on_bus_state_t datatype enum has all states of data on the bus
*   -Members-
*
* -1-SENT
* -2-NOT_SENT
* -3-RECEIVED
* -4-NOT_RECEIVED
*
*/
typedef enum{
       SENT =0,
       NOT_SENT,
       RECEIVED,
       NOT_RECEIVED
}enu_data_on_bus_state_t;
```

```
/*
*
*
* -str_slave_protocol_selection_t datatype hold the address of data queue to receive
and the length of data
*   -Members-
* -1-  (enu_system_protocols_t enu_system_protocols)  protocol that can be used with
BCM
* -2-  (void *slave_id) void pointer to hold slave info address
*
*/
typedef struct{
	void *slave_id;
	enu_system_protocols_t enu_system_protocols;

}str_slave_protocol_selection_t;


/*
*
*
* -str_bcm_instance_t datatype hold all configuration for wanted protocols to
initialization
*   -Members-
* -1-  (enu_protocol_state_t uart_state)  initialized or not for uart
* -2-  (uart_config_t usart_instance) configuration for usart
*
*/
typedef struct
{
	enu_protocol_state_t uart_state;
	uart_config_t usart_instance;
/*
	enu_protocol_state_t spi_state;
	str_spi_instance_t spi_instance;

	enu_protocol_state_t i2c_state;
	str_i2c_instance_t i2c_instance;
*/
}str_bcm_instance_t;
```

UART

```
/*
*
*
* -uart_rcie_mode_t datatype enum has all RC interrupt mode states
*   -Members-
*
* -1-UART_RCIE_DISABLE
* -2-UART_RCIE_ENABLE
*
*/
typedef enum{
       UART_RCIE_DISABLE = 0,
       UART_RCIE_ENABLE
}uart_rcie_mode_t;
/*
*
*
* -uart_tcie_mode_t datatype enum has all TC interrupt mode states
*   -Members-
*
* -1-UART_TCIE_DISABLE
* -2-UART_TCIE_ENABLE
*
*/
typedef enum{
       UART_TCIE_DISABLE = 0,
       UART_TCIE_ENABLE
}uart_tcie_mode_t;


/*
*
*
* -uart_urie_mode_t datatype enum has all UDR empty interrupt mode states
*   -Members-
*
* -1-UART_UDRIE_DISABLE
* -2-UART_UDRIE_ENABLE
*
*/
typedef enum{
       UART_UDRIE_DISABLE = 0,
       UART_UDRIE_ENABLE
}uart_urie_mode_t;

/*
*
*
* -uart_rx_mode_t datatype enum has all RX mode states
*   -Members-
*
* -1-UART_RX_DISABLE
* -2-UART_RX_ENABLE
*
```

```
*/
typedef enum{
	UART_RX_DISABLE = 0,
	UART_RX_ENABLE
}uart_rx_mode_t;

/*
*
*
* -uart_tx_mode_t datatype enum has all TX mode states
*   -Members-
*
* -1-UART_TX_DISABLE
* -2-UART_TX_ENABLE
*
*/
typedef enum{
	UART_TX_DISABLE = 0,
	UART_TX_ENABLE
}uart_tx_mode_t;

/*
*
*
* -uart_speed_mode_t datatype enum has all speed mode states
*   -Members-
*
* -1-UART_SYNC_SPEED_MODE
* -2-UART_NORMAL_MODE
* -3-UART_DOUBLE_MODE
*
*/
typedef enum{
	UART_SYNC_SPEED_MODE = 0,
	UART_NORMAL_MODE = 0,
	UART_DOUBLE_MODE
}uart_speed_mode_t;


/*
*
*
* -uart_clock_polarity_t datatype enum has all clock polarity mode states
*   -Members-
*
* -1-UART_NO_CLOCK
* -2-UART_TXR_RXF
* -3-UART_TXF_RXR
*
*/
typedef enum{
	UART_NO_CLOCK = 0,
	UART_TXR_RXF = 0,
	UART_TXF_RXR
}uart_clock_polarity_t;

/*
*
```

```
*
* -uart_stop_mode_t datatype enum has all number of stop bits
*   -Members-
*
* -1-UART_STOP_1_BIT
* -2-UART_STOP_2_BIT
*
*/
typedef enum{
       UART_STOP_1_BIT =0,
       UART_STOP_2_BIT
}uart_stop_mode_t;

/*
*
*
* -uart_parity_mode_t datatype enum has all modes of parity bits
*   -Members-
*
* -1-UART_PARITY_DIABLED
* -2-UART_PARITY_EVEN
* -3-UART_PARITY_ODD
*
*/
typedef enum{
       UART_PARITY_DIABLED =0,
       UART_PARITY_EVEN =2,
       UART_PARITY_ODD
}uart_parity_mode_t;

/*
*
*
* -uart_mode_t datatype enum has all modes of usart
*   -Members-
*
* -1-UART_ASYNC_MODE
* -2-UART_SYNC_MODE
*
*/
typedef enum{
       UART_ASYNC_MODE =0,
       UART_SYNC_MODE
}uart_mode_t;

/*
*
*
* -uart_mode_t datatype enum has all modes of character sizes
*   -Members-
*
* -1-UART_CS_5
* -2-UART_CS_6
* -3-UART_CS_7
* -4-UART_CS_8
* -5-UART_CS_9
*
*/
```

```c
typedef enum{
	UART_CS_5 =0,
	UART_CS_6,
	UART_CS_7,
	UART_CS_8,
	UART_CS_9 =7
}uart_cs_mode_t;


/*
*
*
* -uart_config_t datatype hold all configuration of usart
*
*
*/
typedef struct{
	uart_mode_t uart_mode;
	uart_cs_mode_t uart_cs_mode;
	uart_parity_mode_t uart_parity_mode;
	uart_stop_mode_t uart_stop_mode;
	uart_clock_polarity_t uart_clock_polarity;
	uart_speed_mode_t uart_speed_mode;
	uart_rcie_mode_t uart_rcie_mode;
	uart_tcie_mode_t uart_tcie_mode;
	uart_urie_mode_t uart_urie_mode;
	uart_rx_mode_t	uart_rx_mode;
	uart_tx_mode_t	uart_tx_mode;
	uint32_t usart_buadrate;
	/*void (*RXC_func)(void);
	void (*TXC_func)(void);
	void (*UDRE_func)(void);*/

}uart_config_t;
```