

design & Implement a BCM

Youssef Ahmed Abbas Mohamed



Sprints

Contents

[Project Introduction: 1](#_Toc136091548)

[High Level Design: 1](#_Toc136091549)

[Layered architecture: 1](#_Toc136091550)

[Module Description 2](#_Toc136091551)

[Driver Documentations 3](#_Toc136091552)

[LED: 3](#_Toc136091553)

[BCM Manager: 4](#_Toc136091554)

[DIO: 6](#_Toc136091555)

[UART: 10](#_Toc136091556)

[UML: 14](#_Toc136091557)

[State Machine: 14](#_Toc136091558)

[Sequence Diagram 16](#_Toc136091559)

[Low Level Design: 18](#_Toc136091560)

[Flowchart 18](#_Toc136091561)

[Pre-compline 22](#_Toc136091562)

[Application 22](#_Toc136091563)

[BCM 22](#_Toc136091564)

[STD 22](#_Toc136091565)

[BIT\_MATH 23](#_Toc136091566)

[Linking configuration 23](#_Toc136091567)

[LED 23](#_Toc136091568)

[BCM 23](#_Toc136091569)

[DIO 24](#_Toc136091570)

[UART 25](#_Toc136091571)

# 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

A screenshot of a computer screen

Description automatically generated with low confidence

## Module Description

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

A screenshot of a computer

Description automatically generated with medium confidence

## Driver Documentations

### 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.

**void uart\_udrei\_enable(void)**

This function enables the UART Data Register Empty (UDRE) interrupt.

**void uart\_udrei\_disable(void)**

This function disables the UART Data Register Empty (UDRE) interrupt.

**void uart\_rxci\_enable(void)**

This function enables the UART Receive Complete (RXC) interrupt.

**void uart\_rxci\_disable(void)**

This function disables the UART Receive Complete (RXC) interrupt.

**void uart\_txci\_enable(void)**

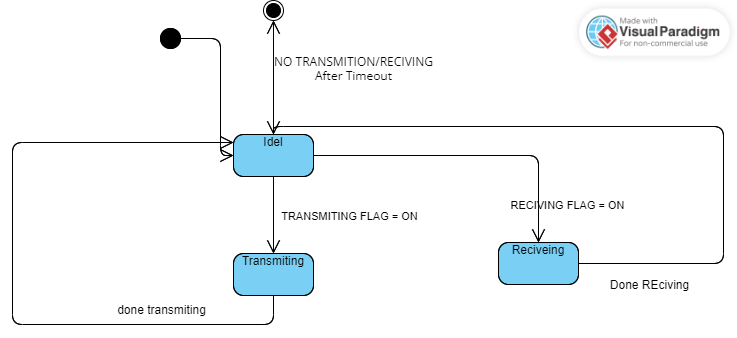
This function enables the UART Transmit Complete (TXC) interrupt.

**void uart\_txci\_disable(void)**

This function disables the UART Transmit Complete (TXC) interrupt.

## UML:

### State Machine:



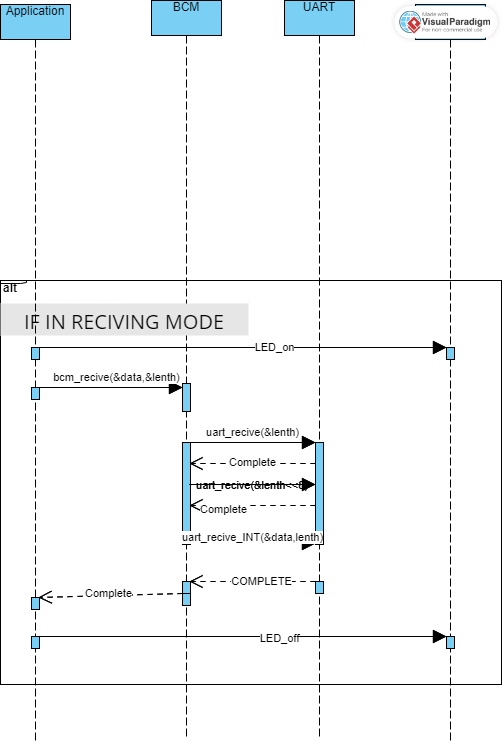
The state machine for the application consists of three states:

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.

### Sequence Diagram

A picture containing text, screenshot, parallel, diagram

Description automatically generated



# Low Level Design:

## Flowchart

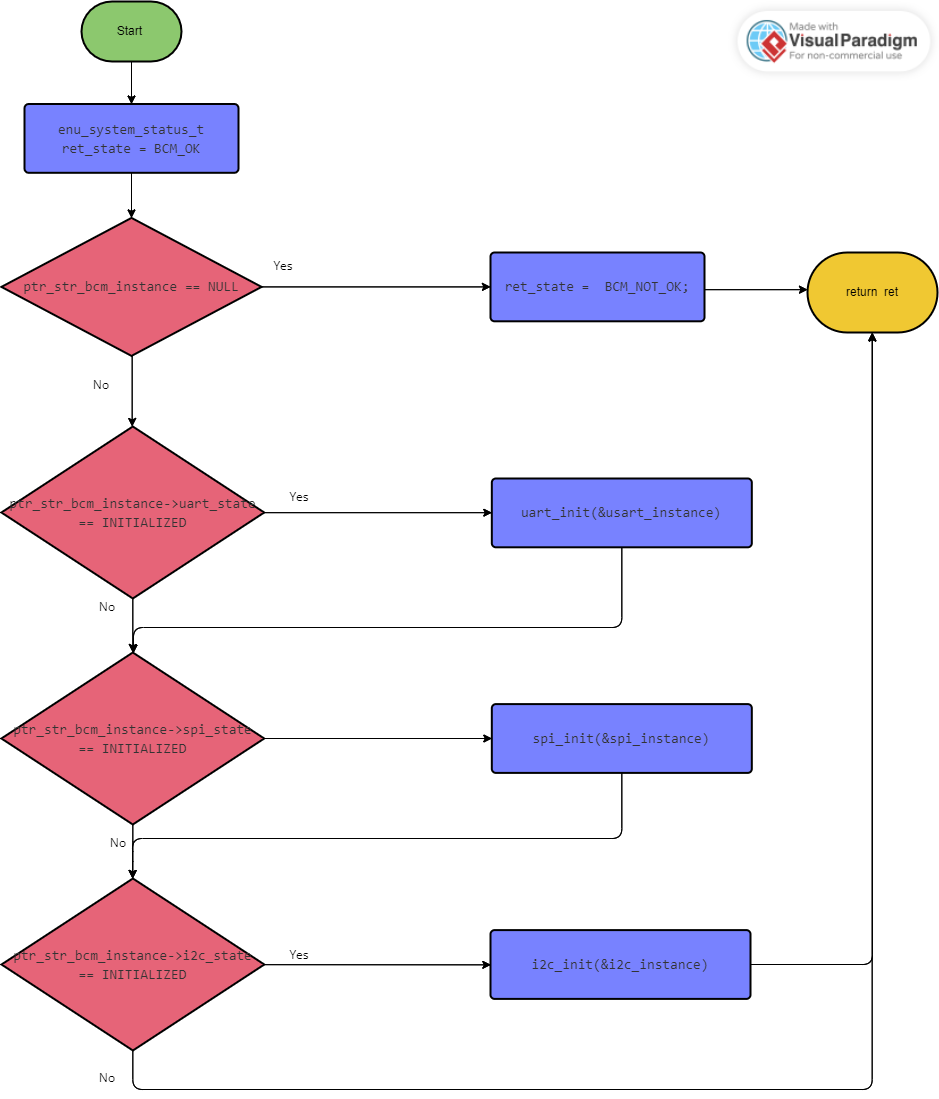


Figure 1 bcm\_init.vpd

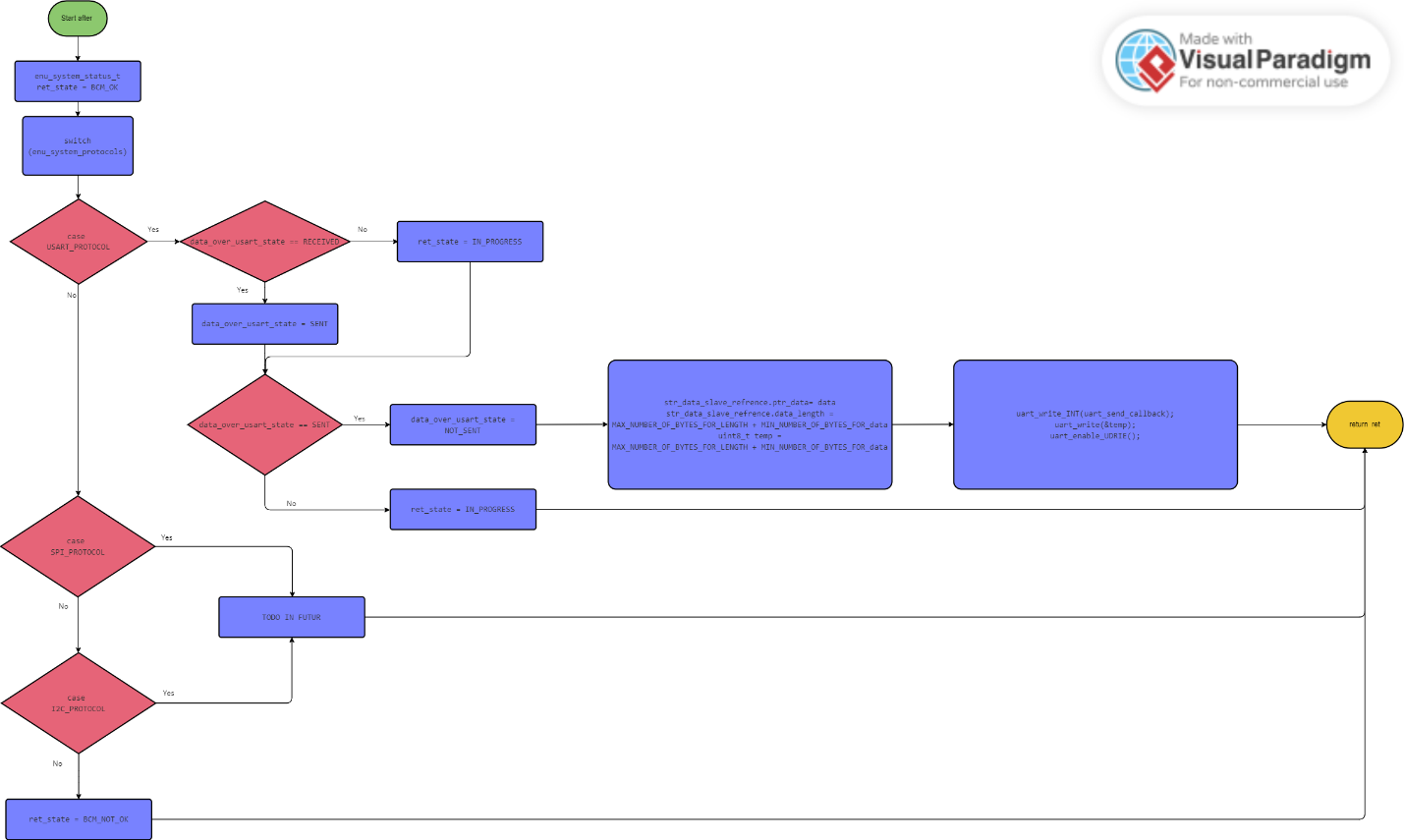


Figure 2 bcm\_send.vpd

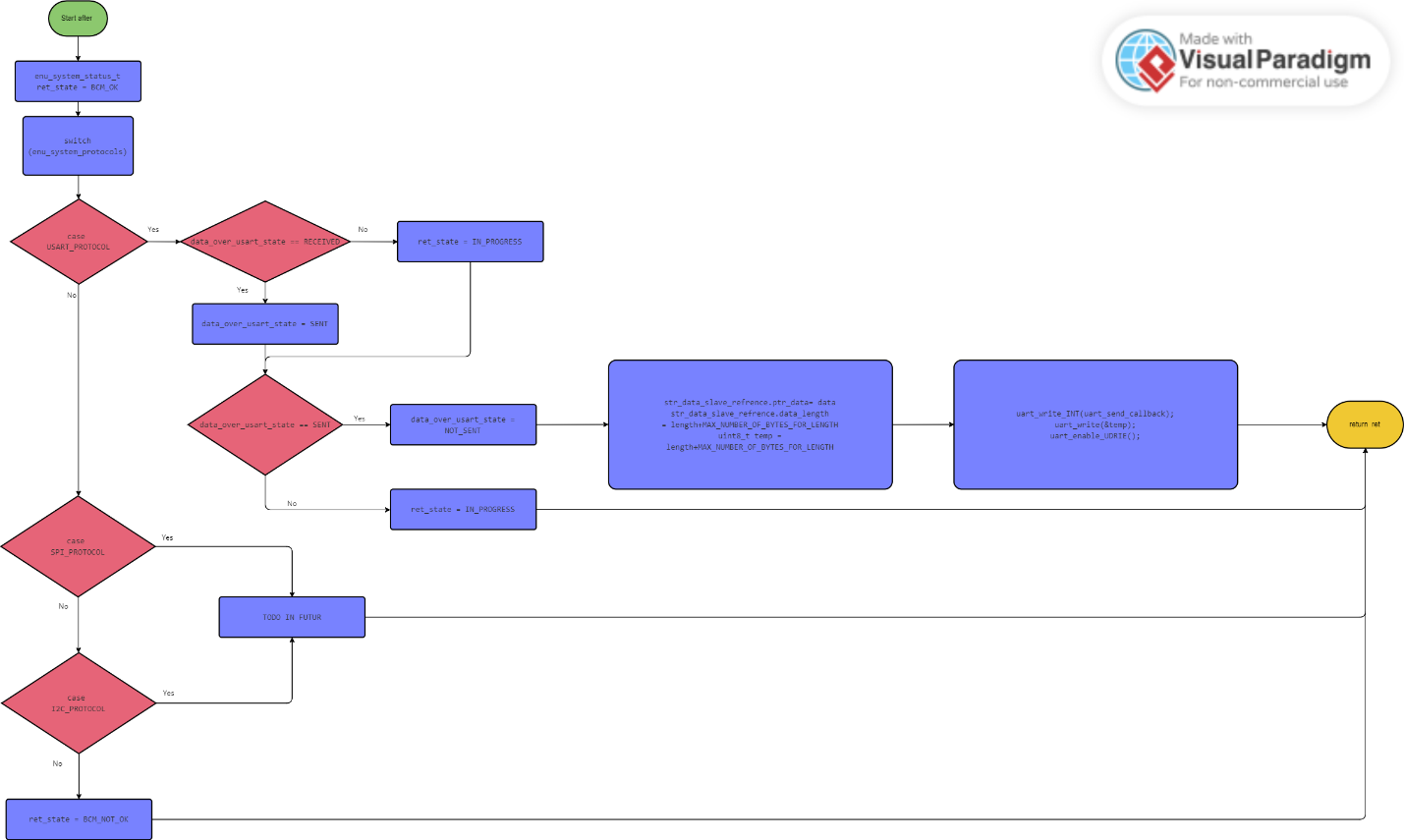


Figure 3 bcm\_send\_n.vpd

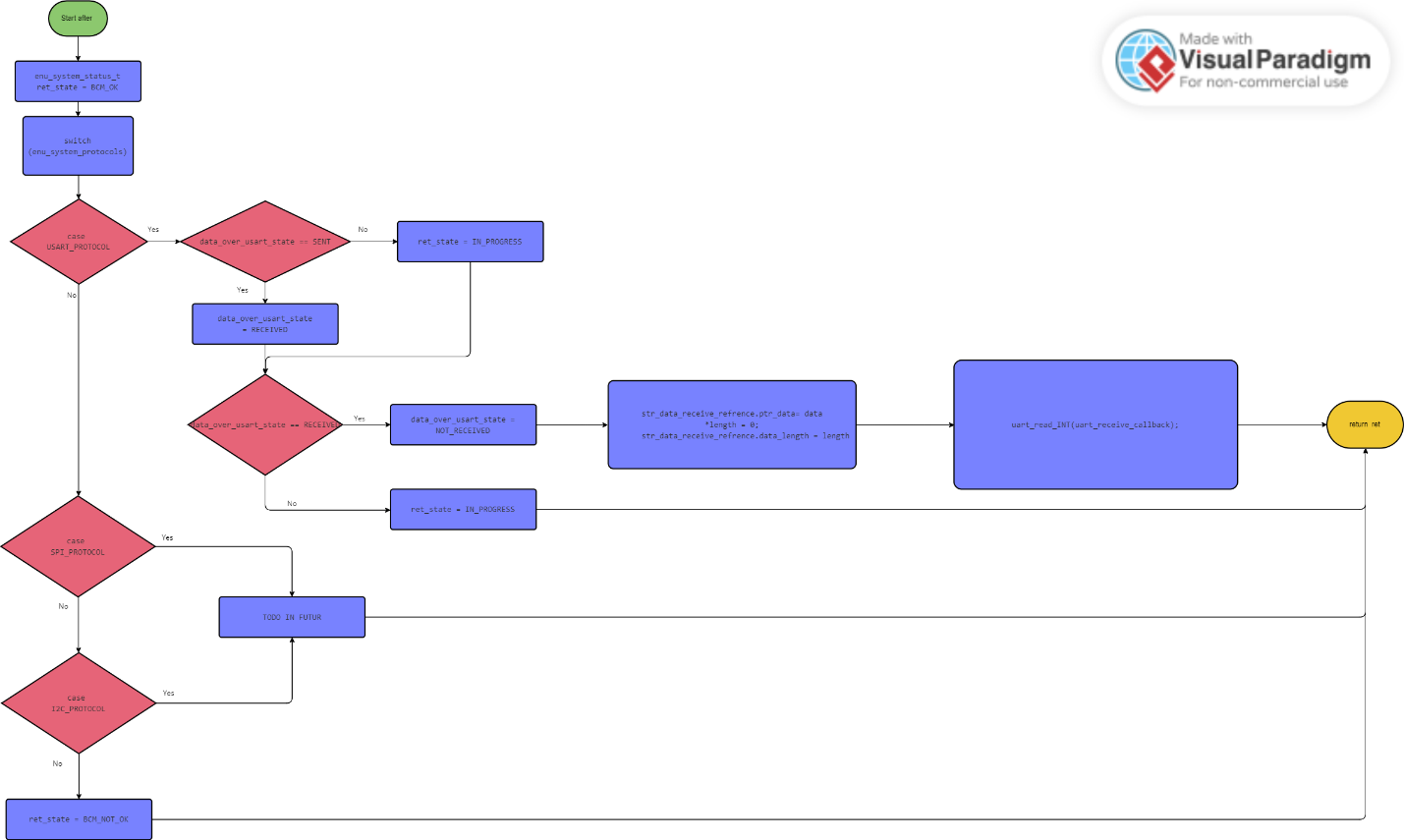


Figure 4 bcm\_receive\_n.vpd

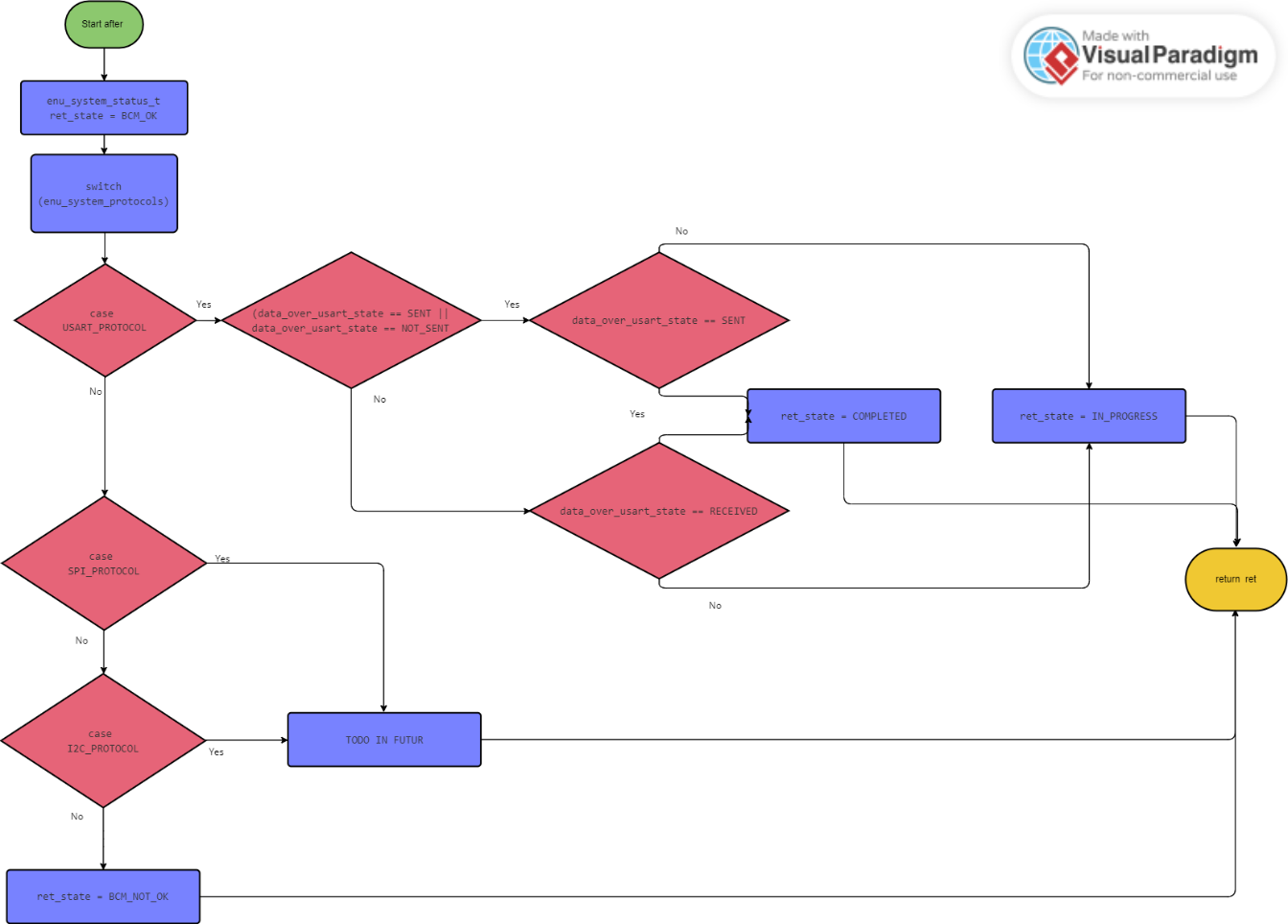


Figure 5 bcm\_dispatcher.vpd

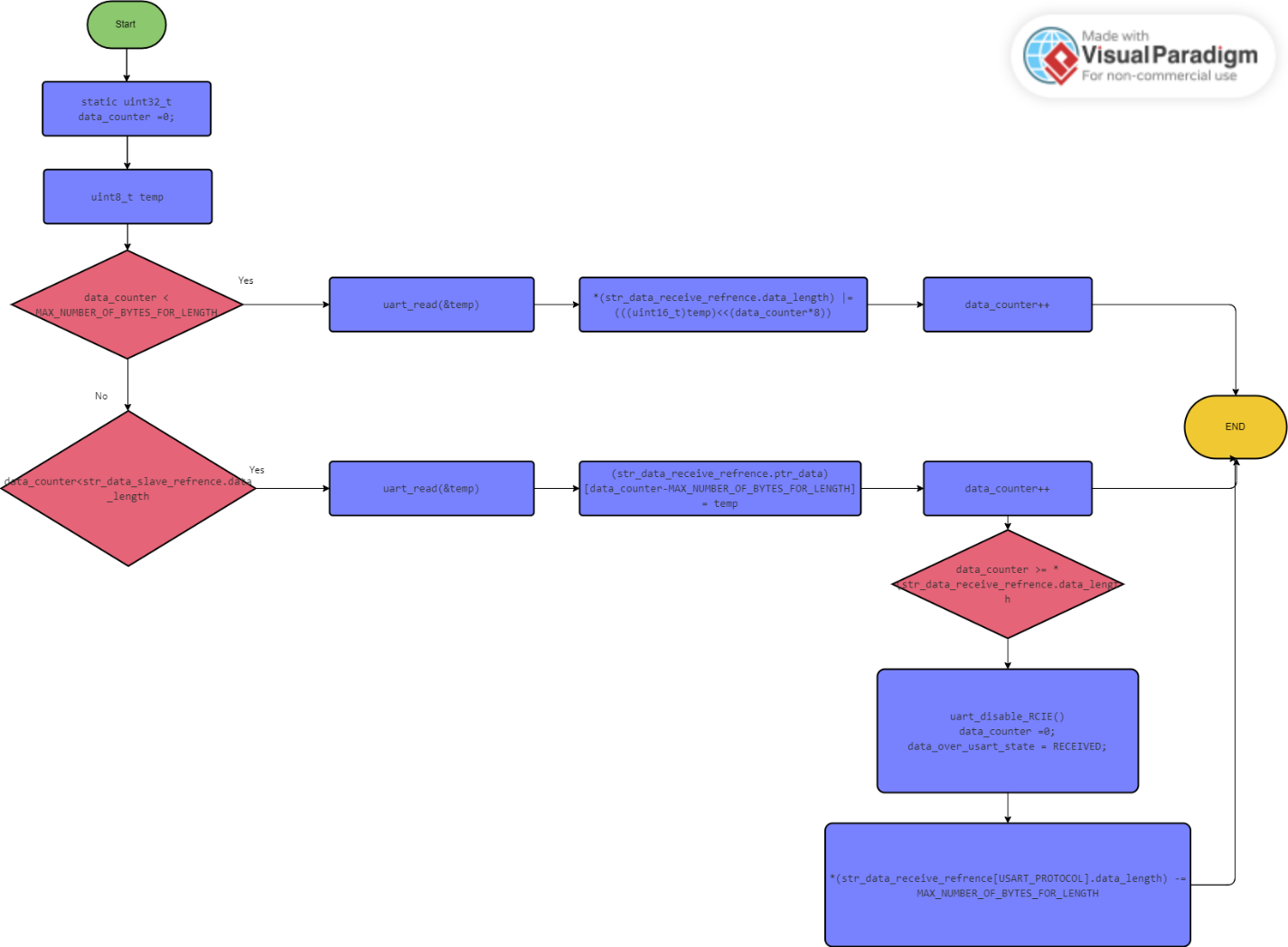


Figure 6 uart\_receive\_callback.vpd

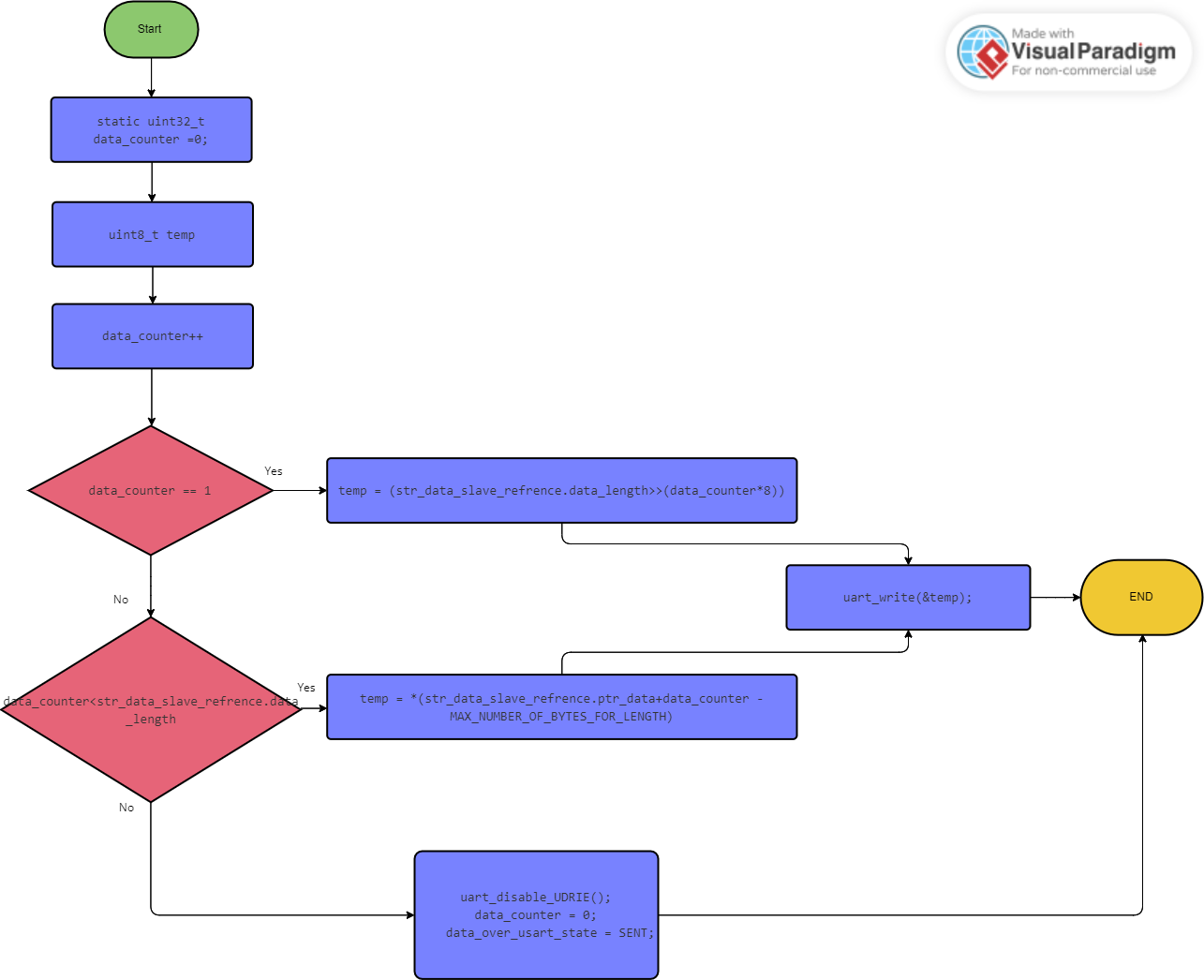


Figure 7 uart\_send\_callback.vpd

## Pre-compline

### Application

|  |
| --- |
| #define BUFFER\_SIZE 50 |

### BCM

|  |
| --- |
| #define NUM\_BCM\_INSTANCES 3  #define LENGTH\_BYTE\_SIZE 2  #define DATA\_BYTE\_SIZE 1 |

### STD

|  |
| --- |
| typedef unsigned char uint8; /\* 1-BYTE UNSIGNED DATA (0 - 255) \*/  typedef signed char sint8; /\* 1-BYTE SIGNED DATA (0 - 127) or (-1 - -128) \*/  typedef unsigned short int uint16; /\* 2-BYTES UNSIGNED DATA \*/  typedef signed short int sint16; /\* 2-BYTES SIGNED DATA \*/  typedef unsigned long int uint32; /\* 4-BYTES UNSIGNED DATA \*/  typedef signed long int sint32; /\* 4-BYTES SIGNED DATA \*/  typedef float float32; /\* 4-BYTES FLOATING DATA \*/  typedef double double64; /\* 8-BYTES FLOATING DATA \*/  #define NULL '\0' |

### BIT\_MATH

|  |
| --- |
| /\*macro to check if a bit is set\*/  #define BIT\_IS\_SET(byte,bit\_num) (byte & (1<<bit\_num))  /\*macro to check if bit is cleared\*/  #define BIT\_IS\_CLEAR(byte,bit\_num) (!(byte & (1<<bit\_num)))  #define BIT\_MASK 0x01  #define CLEAR\_BIT(REG, BIT\_POSN) (REG &= (~(BIT\_MASK << BIT\_POSN))) /\*clear a specific bit in reg\*/  #define SET\_BIT(REG, BIT\_POSN) (REG |= (BIT\_MASK << BIT\_POSN)) /\*set a specific bit in reg\*/  #define TOGGLE\_BIT(REG, BIT\_POSN) (REG ^= (BIT\_MASK << BIT\_POSN)) /\*toggle specific bit in reg\*/  #define READ\_BIT(REG, BIT\_POSN) (((REG >> BIT\_POSN) & BIT\_MASK)) /\*read a specific bit in reg\*/ |

## Linking configuration

### LED

|  |
| --- |
| typedef enum {  LED\_ON=1,  LED\_OFF = 0  }enm\_led\_status\_t;  typedef struct{  str\_dio\_t str\_dio;  enm\_led\_status\_t enm\_led\_status;  }str\_led\_t; |

### BCM

|  |
| --- |
| // Communication protocol options  typedef enum {  BCM\_PROTOCOL\_UART =0,  BCM\_PROTOCOL\_SPI ,  BCM\_PROTOCOL\_I2C,  BCM\_MAX\_PROTOCOL  } enm\_cpo\_t;  typedef enum {  BCM\_BUSY\_FLAG=0,  BCM\_IDEL\_FLAG  }enm\_transiver\_state\_t;  typedef struct {  uint8 \* ptr\_data;  uint16 data\_length;  }str\_data\_packet\_t;  typedef struct {  uint8 \* ptr\_data;  uint16 \* data\_length;  }str\_rdata\_packet\_t;  // BCM instance structure  typedef struct {  uint8 bcm\_instance\_id; // BCM instance ID  enm\_cpo\_t protocol; // Communication protocol (e.g., UART, SPI, I2C)  void\* protocolInstance; // Pointer to the specific protocol instance      } str\_bcm\_instance\_t;  // System status enumeration  typedef enum {  BCM\_OK = 0, // Operation successful  BCM\_INVALID\_INSTANCE, // Invalid BCM instance ID  BCM\_ALREADY\_INITIALIZED, // BCM instance already initialized  BCM\_NOT\_INITIALIZED, // BCM instance not initialized  BCM\_INVALID\_PROTOCOL, // Invalid communication protocol  BCM\_INVALID\_PARAMETER  } enu\_system\_status\_t; |

### DIO

|  |
| --- |
| // define ports  typedef enum{  PORT\_A,  PORT\_B,  PORT\_C,  PORT\_D  }enm\_dio\_port\_t;  // dio value  typedef enum {  DIO\_LOW = 0,  DIO\_HIGH  }enm\_dio\_value\_t;  // dio direction  typedef enum {  DIO\_IN = 0,  DIO\_OUT  }enm\_dio\_dir\_t;  // DIO Errors  typedef enum {  DIO\_FAIL=0,  DIO\_SUCCESS  }enm\_dio\_error\_t;  typedef struct {  enm\_dio\_port\_t port;  uint8 pin;  }str\_dio\_t; |

### UART

|  |
| --- |
| // Enums  typedef enum {  UART\_RECEIVE\_DISABLE = 0, // Disable receive  UART\_RECEIVE\_ENABLE // Enable receive  } uart\_receive\_mode\_t;  typedef enum {  UART\_TRANSMIT\_DISABLE = 0, // Disable transmit  UART\_TRANSMIT\_ENABLE // Enable transmit  } uart\_transmit\_mode\_t;  typedef enum {  UART\_UDRE\_INTERRUPT\_DISABLE = 0, // Disable interrupt  UART\_UDRE\_INTERRUPT\_ENABLE // Enable interrupt  } uart\_udre\_interrupt\_mode\_t;  typedef enum {  UART\_RXC\_INTERRUPT\_DISABLE = 0, // Disable RX  UART\_RXC\_INTERRUPT\_ENABLE // Enable RX  } uart\_rxc\_interrupt\_mode\_t;  typedef enum {  UART\_TXC\_INTERRUPT\_DISABLE = 0, // Disable TX  UART\_TXC\_INTERRUPT\_ENABLE // Enable TX  } uart\_txc\_interrupt\_mode\_t;  typedef enum {  UART\_RX\_DISABLE = 0, // Disable RX  UART\_RX\_ENABLE // Enable RX  } uart\_rx\_mode\_t;  typedef enum {  UART\_TX\_DISABLE = 0, // Disable TX  UART\_TX\_ENABLE // Enable TX  } uart\_tx\_mode\_t;  typedef enum {  UART\_SYNC\_SPEED\_MODE = 0, // Sync mode  UART\_NORMAL\_MODE=0, // Normal mode  UART\_DOUBLE\_MODE // Double speed  } uart\_speed\_mode\_t;  typedef enum {  UART\_NO\_CLOCK = 0, // No clock in async mode  UART\_TXR\_RXF,  UART\_TXF\_RXR  } uart\_clock\_polarity\_t;  typedef enum {  UART\_STOP\_1\_BIT = 0, // One bit  UART\_STOP\_2\_BIT // Two bits  } uart\_stop\_mode\_t;  typedef enum {  UART\_PARITY\_DISABLED = 0, // No parity mode  UART\_PARITY\_EVEN, // Even parity mode  UART\_PARITY\_ODD // Odd parity mode  } uart\_parity\_mode\_t;  typedef enum {  UART\_ASYNC\_MODE = 0, // Async mode  UART\_SYNC\_MODE // Sync mode  } uart\_operating\_mode\_t;  typedef enum {  UART\_CS\_5 = 0, // 5 bits length  UART\_CS\_6, // 6 bits length  UART\_CS\_7, // 7 bits length  UART\_CS\_8, // 8 bits length  UART\_CS\_9= 7 // 9 bits length  } uart\_data\_size\_t;  // Structures  typedef struct {  uart\_operating\_mode\_t uart\_mode;  uart\_data\_size\_t uart\_data\_size;  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\_receive\_mode\_t uart\_receive\_mode;  uart\_transmit\_mode\_t uart\_transmit\_mode;  uart\_udre\_interrupt\_mode\_t uart\_udre\_interrupt\_mode;  uart\_rx\_mode\_t uart\_rx\_mode;  uart\_tx\_mode\_t uart\_tx\_mode;  uart\_rxc\_interrupt\_mode\_t uart\_rxc\_interrupt\_mode;  uart\_txc\_interrupt\_mode\_t uart\_txc\_interrupt\_mode;  uint16 uart\_baudrate;  } uart\_config\_t; |