SPRINTS

# Basic Communication Manager Design

May 2023

Presented by
**Abdelrhman Walaa**

Presented to
**Sprints**

# Table Of Content

## Basic Communication Manager Design

# 1. Project Introduction

This project involves developing Basic Communication Manager software.

## 1.1. System Requirements

### 1.1.1. Hardware Requirements

1. **ATmega32** microcontroller
2. **Two LEDs** to be toggled

### 1.1.2. Software Requirements

1. Send "*BCM Operating*" string from **MCU_1** to **MCU_2**.
2. When MCU_1 finishes sending, **LED_0** in **MCU_ 1** will be toggled.
3. When MCU_2 finishes receiving the "*BCM Operating*" string, **LED_1** in **MCU_2** will be toggled.
4. **MCU_2** will respond with a "*Confirm BCM Operating*" string to **MCU_1**.
5. When **MCU_2** finishes sending, **LED_0** in **MCU_2** will be toggled.
6. When **MCU_1** finishes receiving the "*Confirm BCM Operating*" string, **LED_1** in **MCU_1** will be toggled.

# 2. High Level Design

## 2.1. System Architecture

### 2.1.1. Definition

*Layered Architecture* (*Figure 1*) describes an architectural pattern composed of several separate horizontal layers that function together as a single unit of software.

*Microcontroller Abstraction Layer* (*MCAL*) is a software module that directly accesses on-chip MCU peripheral modules and external devices that are mapped to memory, and makes the upper software layer independent of the MCU.

*Hardware Abstraction Layer* (*HAL*) is a layer of programming that allows a computer OS to interact with a hardware device at a general or abstract level rather than at a detailed hardware level.

*Service Layer* (*SRVL*) is usually the software layer that mediates between application software and device driver software.

### 2.1.2. Layered Architecture

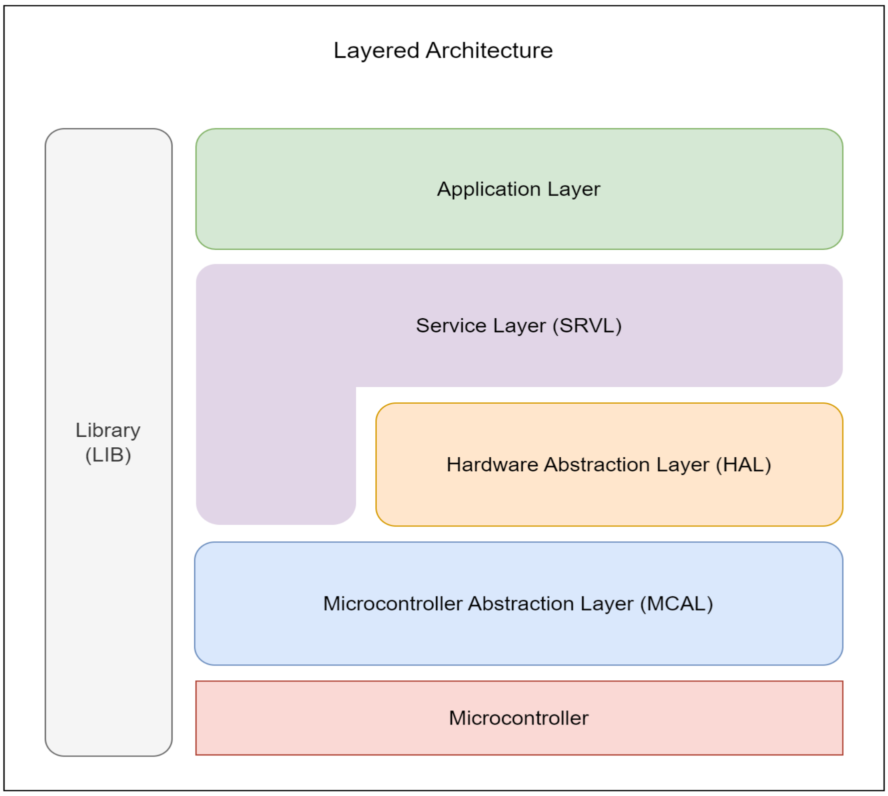



*Figure 1. Layered Architecture Design*

## 2.2. System Modules

### 2.2.1. Definition

A *Module* is a distinct assembly of components that can be easily added, removed or replaced in a larger system. Generally, a *Module* is not functional on its own.

In computer hardware, a *Module* is a component that is designed for easy replacement.

### 2.2.2. Design

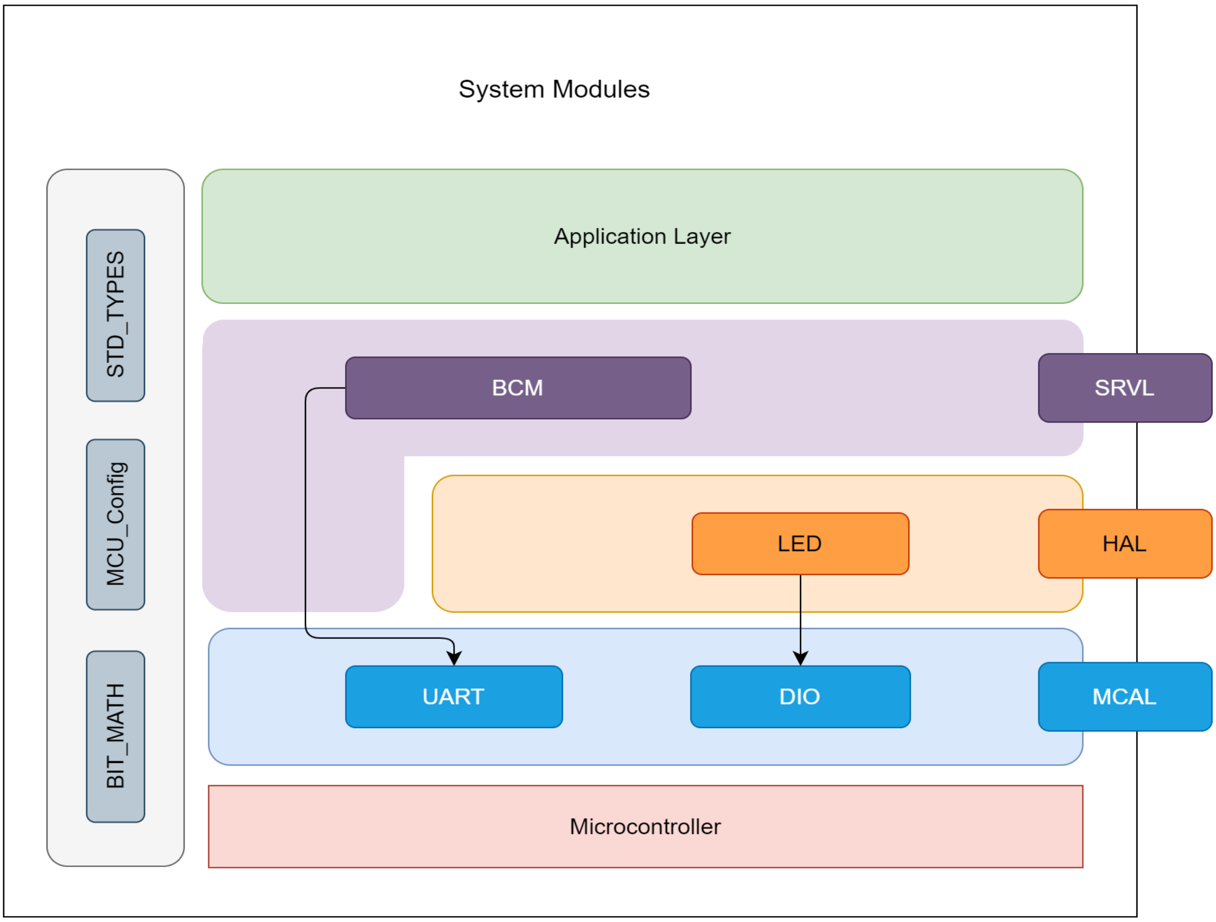



*Figure 2. System Modules Design*

## 2.3. Modules Description

### 2.3.1. DIO Module

The *DIO* (Digital Input/Output) module is responsible for reading input signals from the system's sensors (such as buttons) and driving output signals to the system's actuators (such as *LEDs*). It provides a set of APIs to configure the direction and mode of each pin (input/output, pull-up/down resistor), read the state of an input pin, and set the state of an output pin.

### 2.3.2. UART Module

The Universal Asynchronous Receiver-Transmitter (*UART*) is a popular asynchronous communication protocol. The *UART* driver is responsible for managing the data transfer between the two devices. The *UART* protocol allows for two-way data transfer, making it ideal for communication with external devices.

### 2.3.3. LED Module

*LED* (Light Emitting Diode) is responsible for controlling the state of the systems' LEDs.
It provides a set of APIs to turn off, to turn on, or to toggle the state of each led.

### 2.3.4. BCM Module

The Basic Communication Manager (*BCM*) module is a crucial component in embedded systems that helps mediate interactions between the *Application* Layer and the underlying layers involved in communication protocols. Its primary function is to manage and facilitate communication between the application running on top and the lower-level communication layers, such as the *HAL* and *MCAL* layers.

## 2.4. Drivers' Documentation (APIs)

### 2.4.1 Definition

An *API* is an *Application Programming Interface* that defines a set of *routines*, *protocols* and *tools* for creating an application. An *API* defines the high level interface of the behavior and capabilities of the component and its inputs and outputs.

An *API* should be created so that it is generic and implementation independent. This allows for the API to be used in multiple applications with changes only to the implementation of the API and not the general interface or behavior.

### 2.4.2. MCAL APIs

#### 2.4.2.1. DIO Driver APIs

```
Precompile Configurations Snippet:

/* Initial Directions of Port A Pins*/
/* Options: DIO_U8_INITIAL_INPUT
 *          DIO_U8_INITIAL_OUTPUT
 */

/* PORTA */
#define DIO_U8_PA0_INITIAL_DIRECTION    DIO_U8_INITIAL_INPUT
#define DIO_U8_PA1_INITIAL_DIRECTION    DIO_U8_INITIAL_INPUT
                •
                •
#define DIO_U8_PA7_INITIAL_DIRECTION    DIO_U8_INITIAL_INPUT

/* Initial Values of Port A Pins */
/* Options:  DIO_U8_INPUT_FLOATING
 *           DIO_U8_INPUT_PULLUP_RESISTOR
 *           DIO_U8_OUTPUT_LOW
 *           DIO_U8_OUTPUT_HIGH
 */

/* PORTA */
#define DIO_U8_PA0_INITIAL_VALUE       DIO_U8_INPUT_FLOATING
#define DIO_U8_PA1_INITIAL_VALUE       DIO_U8_INPUT_FLOATING
                •
                •
#define DIO_U8_PA7_INITIAL_VALUE       DIO_U8_INPUT_FLOATING
```

```
Linking Configurations Snippet:

There are no Linking Configurations as the defined APIs below could change the
DIO peripheral Configurations during the Runtime.

| Name: DIO_initialization
| Input: void
| Output: void
| Description: Function to initialize DIO peripheral.
|
void DIO_initialization (void)

| Name: DIO_setPinDirection
| Input: u8 PortId, u8 PinId, and u8 PinDirection
| Output: u8 Error or No Error
| Description: Function to set Pin direction.
|
u8 DIO_setPinDirection (u8 u8_a_portId, u8 u8_a_pinId, u8
u8_a_pinDirection)

| Name: DIO_setPinValue
| Input: u8 PortId, u8 PinId, and u8 PinValue
| Output: u8 Error or No Error
| Description: Function to set Pin value.
|
u8 DIO_setPinValue (u8 u8_a_portId, u8 u8_a_pinId, u8 u8_a_pinValue)

| Name: DIO_getPinValue
| Input: u8 PortId, u8 PinId, and Pointer to u8 ReturnedPinValue
| Output: u8 Error or No Error
| Description: Function to get Pin value.
|
u8 DIO_getPinValue (u8 u8_a_portId, u8 u8_a_pinId, u8
*pu8_a_returnedPinValue)

| Name: DIO_togglePinValue
| Input: u8 PortId and u8 PinId
| Output: u8 Error or No Error
| Description: Function to toggle Pin value.
|
u8 DIO_togglePinValue (u8 u8_a_portId, u8 u8_a_pinId)
```

#### 2.4.2.2. UART Driver APIs

```
Precompile and Linking Configurations Snippet:

/* UART Interrupt Ids */
typedef enum
{
  UART_EN_RXC_INT = 0,
  UART_EN_UDRE_INT,
  UART_EN_TXC_INT,
  UART_EN_INVALID_INT_ID
} UART_enInterruptId_t;

/* UART Reception/Transmission Blocking Modes */
typedef enum
{
  UART_EN_BLOCKING_MODE = 0,
  UART_EN_NON_BLOCKING_MODE,
  UART_EN_INVALID_BLOCK_MODE
} UART_enBlockMode_t;

/* UART Error States */
typedef enum
{
  UART_EN_NOK = 0,
  UART_EN_OK
} UART_enErrorState_t;

/* UART Modes */
typedef enum
{
  UART_EN_ASYNC_MODE,
  UART_EN_SYNC_MODE
} UART_enMode_t;

/* UART Transmission Speeds */
typedef enum
{
  UART_EN_NORMAL_SPEED,
  UART_EN_DOUBLE_SPEED
} UART_enSpeed_t;

/* UART Multi-processor Communication Mode */
typedef enum
{
  UART_EN_MPCM_DISABLED,
  UART_EN_MPCM_ENABLED
} UART_enMPCM_t;
```

```c
/* UART Receiver RX */
typedef enum
{
  UART_EN_RX_DISABLED,
  UART_EN_RX_ENABLED
} UART_enRXEnable_t;

/* UART Transmitter TX */
typedef enum
{
  UART_EN_TX_DISABLED,
  UART_EN_TX_ENABLED
} UART_enTXEnable_t;

/* UART RX Complete Interrupt */
typedef enum
{
  UART_EN_RX_INT_DISABLED,
  UART_EN_RX_INT_ENABLED
} UART_enRXInterruptEnable_t;

/* TX Complete Interrupt */
typedef enum
{
  UART_EN_TX_INT_DISABLED,
  UART_EN_TX_INT_ENABLED
} UART_enTXInterruptEnable_t;

/* UART Data Register Empty Interrupt */
typedef enum
{
  UART_EN_UDRE_INT_DISABLED,
  UART_EN_UDRE_INT_ENABLED
} UART_enUDREInterruptEnable_t;

/* UART Parity Modes */
typedef enum
{
  UART_EN_PARITY_MODE_DISABLED,
  UART_EN_EVEN_PARITY_MODE,
  UART_EN_ODD_PARITY_MODE
} UART_enParityMode_t;

/* UART Stop Bit(s) */
typedef enum
{
  UART_EN_ONE_STOP_BIT,
  UART_EN_TWO_STOP_BIT
} UART_enStopBitsSelect_t;
```

```c
/* UART Data Bits */
typedef enum
{
  UART_EN_5_DATA_BITS,
  UART_EN_6_DATA_BITS,
  UART_EN_7_DATA_BITS,
  UART_EN_8_DATA_BITS,
  UART_EN_9_DATA_BITS
} UART_enDataBitsSelect_t;

/* UART Baud Rates (Symbol Per Second -> Bit Per Second (bps)) */
typedef enum
{
  UART_EN_BAUD_RATE_2400,
  UART_EN_BAUD_RATE_4800,
  UART_EN_BAUD_RATE_9600,
  UART_EN_BAUD_RATE_14400,
  UART_EN_BAUD_RATE_19200,
  UART_EN_BAUD_RATE_28800,
  UART_EN_BAUD_RATE_38400,
  UART_EN_BAUD_RATE_57600
} UART_enBaudRateSelect_t;

/* UART Linking Configurations Structure */
typedef struct
{
  UART_enMode_t                en_g_mode;
  UART_enSpeed_t               en_g_speed;
  UART_enMPCM_t                en_g_MPCM;
  UART_enRXEnable_t            en_g_RXEnable;
  UART_enTXEnable_t            en_g_TXEnable;
  UART_enRXInterruptEnable_t   en_g_RXInterruptEnable;
  UART_enTXInterruptEnable_t   en_g_TXInterruptEnable;
  UART_enUDREInterruptEnable_t en_g_UDREInterruptEnable;
  UART_enParityMode_t          en_g_parityMode;
  UART_enStopBitsSelect_t      en_g_stopBit;
  UART_enDataBitsSelect_t      en_g_dataBits;
  UART_enBaudRateSelect_t      en_g_baudRate;
} UART_stLinkConfig_t;

| Name: UART_initialization
| Input: void
| Output: void
| Description: Function to initialize UART peripheral using Pre-compile
|              Configurations.
|
void UART_initialization (void)
```

```
| Name: UART_linkConfigInitialization
| Input: Pointer to stLinkConfig
| Output: en Error or No Error
| Description: Function to initialize UART peripheral using Linking Configurations.
|
UART_enErrorState_t UART_linkConfigInitialization (const UART_stLinkConfig_t
*pst_a_linkConfig)

| Name: UART_receiveByte
| Input: en BlockMode and Pointer to u8 ReturnedReceiveByte
| Output: en Error or No Error
| Description: Function to Receive Byte using both Blocking and Non-blocking Modes,
|              with a Timeout mechanism.
|
UART_enErrorState_t UART_receiveByte (UART_enBlockMode_t en_a_blockMode, u8
*pu8_a_returnedReceiveByte)

| Name: UART_transmitByte
| Input: en BlockMode and u8 TransmitByte
| Output: en Error or No Error
| Description: Function to Transmit Byte using both Blocking and Non-blocking Modes,
|              with a Timeout mechanism.
|
UART_enErrorState_t UART_transmitByte (UART_enBlockMode_t u8_a_blockMode, u8
u8_a_transmitByte)

| Name: UART_enableInterrupt
| Input: en InterruptId
| Output: en Error or No Error
| Description: Function to enable UART different interrupts.
|
UART_enErrorState_t UART_enableInterrupt (UART_enInterruptId_t
en_a_interruptId)

| Name: UART_disableInterrupt
| Input: en InterruptId
| Output: en Error or No Error
| Description: Function to disable UART different interrupts.
|
UART_enErrorState_t UART_disableInterrupt (UART_enInterruptId_t
en_a_interruptId)
```

```
| Name: UART_RXCSetCallback
| Input: Pointer to Function that takes void and returns void
| Output: en Error or No Error
| Description: Function to receive an address of a function ( in Upper Layer ) to be
|              called back in ISR function, the address is passed through a pointer
|              to function ( RXCInterruptAction ), and then pass this address to the
|              ISR function.
|
UART_enErrorState_t UART_RXCSetCallback
(void(*vpf_a_RXCInterruptAction)(void))

| Name: UART_UDRESetCallback
| Input: Pointer to Function that takes void and returns void
| Output: en Error or No Error
| Description: Function to receive an address of a function ( in Upper Layer ) to be
|              called back in ISR function, the address is passed through a pointer
|              to function ( UDREInterruptAction ), and then pass this address to the
|              ISR function.
|
UART_enErrorState_t UART_UDRESetCallback
(void(*vpf_a_UDREInterruptAction)(void))

| Name: UART_TXCSetCallback
| Input: Pointer to Function that takes void and returns void
| Output: en Error or No Error
| Description: Function to receive an address of a function ( in Upper Layer ) to be
|              called back in ISR function, the address is passed through a pointer
|              to function ( TXCInterruptAction ), and then pass this address to the
|              ISR function.
|
UART_enErrorState_t UART_TXCSetCallback
(void(*vpf_a_TXCInterruptAction)(void))
```

## 2.4.3. HAL APIs

### 2.4.3.1. LED Driver APIs

```
| Name: LED_initialization
| Input: u8 LedId
| Output: u8 Error or No Error
| Description: Function to initialize LED peripheral.
|
u8 LED_initialization (u8 u8_a_ledId)

| Name: LED_setLEDPin
| Input: u8 LedId and u8 Operation
| Output: u8 Error or No Error
| Description: Function to switch LED on, off, or toggle.
|
u8 LED_setLEDPin (u8 u8_a_ledId, u8 u8_a_operation)
```

## 2.4.4. SRVL APIs

### 2.4.4.1. BCM Driver APIs

```
Precompile and Linking Configurations Snippet:

/* BCM End of String Character */
#define BCM_U8_END_OF_STRING     '&'

/* BCM Protocols Ids */
typedef enum
{
  BCM_EN_PROTOCOL_0 = 0,  // UART Protocol
  BCM_EN_PROTOCOL_1,      // SPI Protocol
  BCM_EN_PROTOCOL_2,      // TWI Protocol
  BCM_EN_INVALID_PROTOCOL
} BCM_enProtocolId_t;

/* BCM States */
typedef enum
{
  BCM_EN_IDLE = 0,
  /* UART Protocol */
  BCM_EN_UART_READY_TO_RECEIVE,
  BCM_EN_UART_RECEIVE_COMPLETE,
  BCM_EN_UART_READY_TO_TRANSMIT,
  BCM_EN_UART_TRANSMIT_COMPLETE,
  /* SPI Protocol */
  BCM_EN_SPI_READY_TO_RECEIVE,
  BCM_EN_SPI_RECEIVE_COMPLETE,
  BCM_EN_SPI_READY_TO_TRANSMIT,
  BCM_EN_SPI_TRANSMIT_COMPLETE,
  /* TWI Protocol */
  BCM_EN_TWI_READY_TO_RECEIVE,
  BCM_EN_TWI_RECEIVE_COMPLETE,
  BCM_EN_TWI_READY_TO_TRANSMIT,
  BCM_EN_TWI_TRANSMIT_COMPLETE,
  BCM_EN_INVALID_STATE
} BCM_enState_t;

/* BCM Error States */
typedef enum
{
  BCM_EN_NOK = 0,
  BCM_EN_OK
} BCM_enErrorState_t;
```

```
| Name: BCM_initialization
| Input: en ProtocolId
| Output: en Error or No Error
| Description: Function to initialize BCM.
|
BCM_enErrorState_t BCM_initialization (BCM_enProtocolId_t en_a_protocolId)

| Name: BCM_deinitialization
| Input: en ProtocolId
| Output: en Error or No Error
| Description: Function to deinitialize BCM.
|
BCM_enErrorState_t BCM_deinitialization (BCM_enProtocolId_t en_a_protocolId)

| Name: BCM_receiveByte
| Input: en ProtocolId and Pointer to u8 ReturnedReceiveByte
| Output: en Error or No Error
| Description: Function to Transmit Byte.
|
BCM_enErrorState_t BCM_receiveByte (BCM_enProtocolId_t en_a_protocolId, u8
*pu8_a_returnedReceiveByte)

| Name: BCM_receiveString
| Input: en ProtocolId and Pointer to u8 ReturnedReceiveString
| Output: en Error or No Error
| Description: Function to Receive Byte.
|
BCM_enErrorState_t BCM_receiveString (BCM_enProtocolId_t en_a_protocolId, u8
*pu8_a_returnedReceiveString)

| Name: BCM_transmitByte
| Input: en ProtocolId and u8 TransmitByte
| Output: en Error or No Error
| Description: Function to Transmit Byte.
|
BCM_enErrorState_t BCM_transmitByte (BCM_enProtocolId_t en_a_protocolId, u8
u8_a_transmitByte)

| Name: BCM_transmitString
| Input: en ProtocolId and Pointer to u8 TransmitString
| Output: en Error or No Error
| Description: Function to Transmit String.
|
BCM_enErrorState_t BCM_transmitString (BCM_enProtocolId_t en_a_protocolId, u8
*pu8_a_transmitString)
```

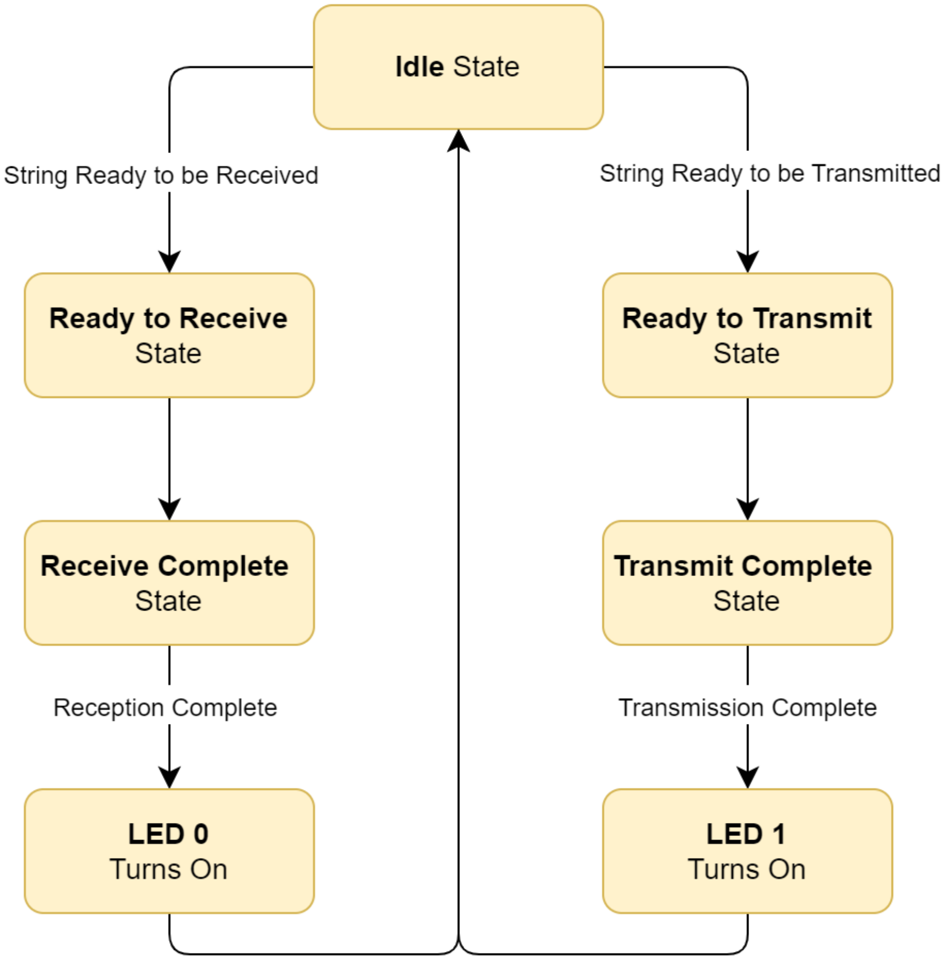
```
| Name: BCM_receiveDispatcher
| Input: en ProtocolId
| Output: en Error or No Error
| Description: Function to execute the periodic actions and notifies the user with
|             the needed events over a specific BCM instance.
|
BCM_enErrorState_t BCM_receiveDispatcher (BCM_enProtocolId_t en_a_protocolId)

| Name: BCM_transmitDispatcher
| Input: en ProtocolId
| Output: en Error or No Error
| Description: Function to execute the periodic actions and notifies the user with
|              the needed events over a specific BCM instance.
|
BCM_enErrorState_t BCM_transmitDispatcher (BCM_enProtocolId_t en_a_protocolId)

| Name: BCM_receiveCompleteSetCallback
| Input: en ProtocolId and Pointer to Function that takes void and returns void
| Output: en Error or No Error
| Description: Function to receive an address of a function ( in APP Layer ) to be
|              called back in ISR function, the address is passed through a pointer
|              to function ( ReceiveCompleteInterruptAction ), and then pass this
|              address to the ISR function.
|
BCM_enErrorState_t BCM_receiveCompleteSetCallback (BCM_enProtocolId_t
en_a_protocolId, void(*vpf_a_receiveCompleteInterruptAction)(void))

| Name: BCM_transmitCompleteSetCallback
| Input: en ProtocolId and Pointer to Function that takes void and returns void
| Output: en Error or No Error
| Description: Function to receive an address of a function ( in APP Layer ) to be
|              called back in ISR function, the address is passed through a pointer
|              to function ( TransmitCompleteInterruptAction ), and then pass this
|              address to the ISR function.
|
BCM_enErrorState_t BCM_transmitCompleteSetCallback (BCM_enProtocolId_t
en_a_protocolId, void(*vpf_a_transmitCompleteInterruptAction)(void))
```

### 2.4.5. APP APIs

```
| Name: APP_vdInitialization
| Input: void
| Output: void
| Description: Function to Initialize the Application.
|
void APP_vdInitialization (void)

| Name: APP_vdStartProgram
| Input: void
| Output: void
| Description: Function to Start the basic flow of the Application.
|
void APP_vdStartProgram (void)

| Name: APP_receiveComplete
| Input: void
| Output: void
| Description: Function to be called back when reception is completed.
|
void APP_receiveComplete (void)

| Name: APP_transmitComplete
| Input: void
| Output: void
| Description: Function to be called back when transmission is completed.
|
void APP_transmitComplete (void)
```

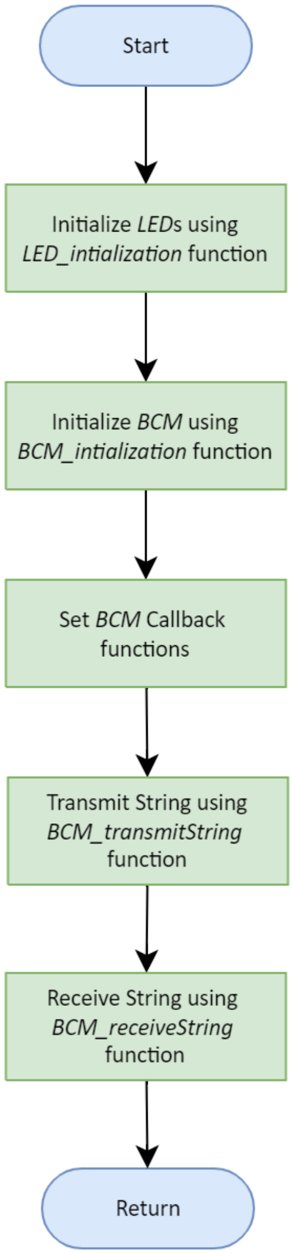
## 2.5. UML

### 2.5.1. State Machine Diagram

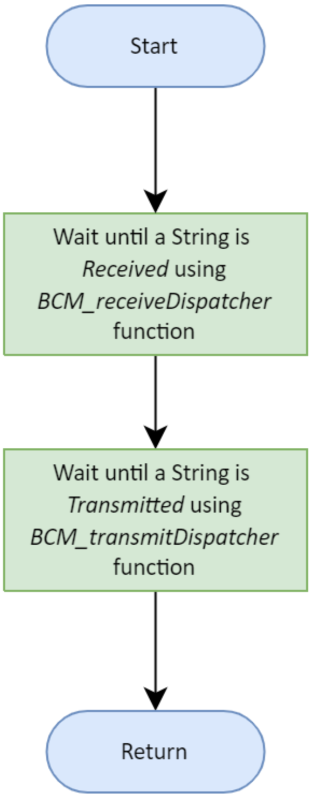


### 2.5.2. Flowchart Diagram

*A. APP_initialization*



*B. APP_startProgram*



## 4. References


1. *Draw IO*
2. *Layered Architecture | Baeldung on Computer Science*
3. *Block Diagram Maker | Free Block Diagram Online | Lucidchart*
4. *Microcontroller Abstraction Layer (MCAL) | Renesas*
5. *Hardware Abstraction Layer - an overview | ScienceDirect Topics*
6. *Top Five Differences Between Layers And Tiers | Skill-Lync*
7. *What is a module in software, hardware and programming?*
8. *Embedded Basics – API's vs HAL's*