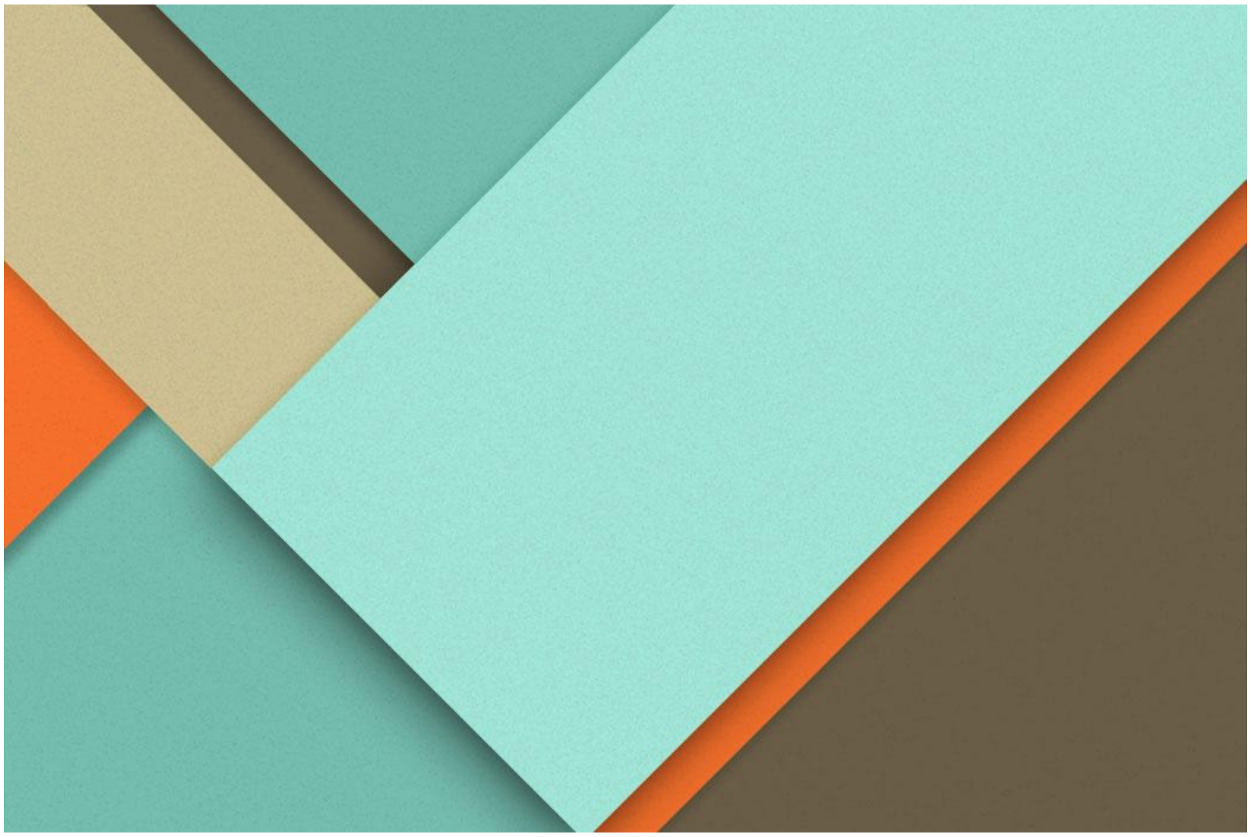


Basic Communication Manger Design

Prepared By:

Moustafa Abdelrahim

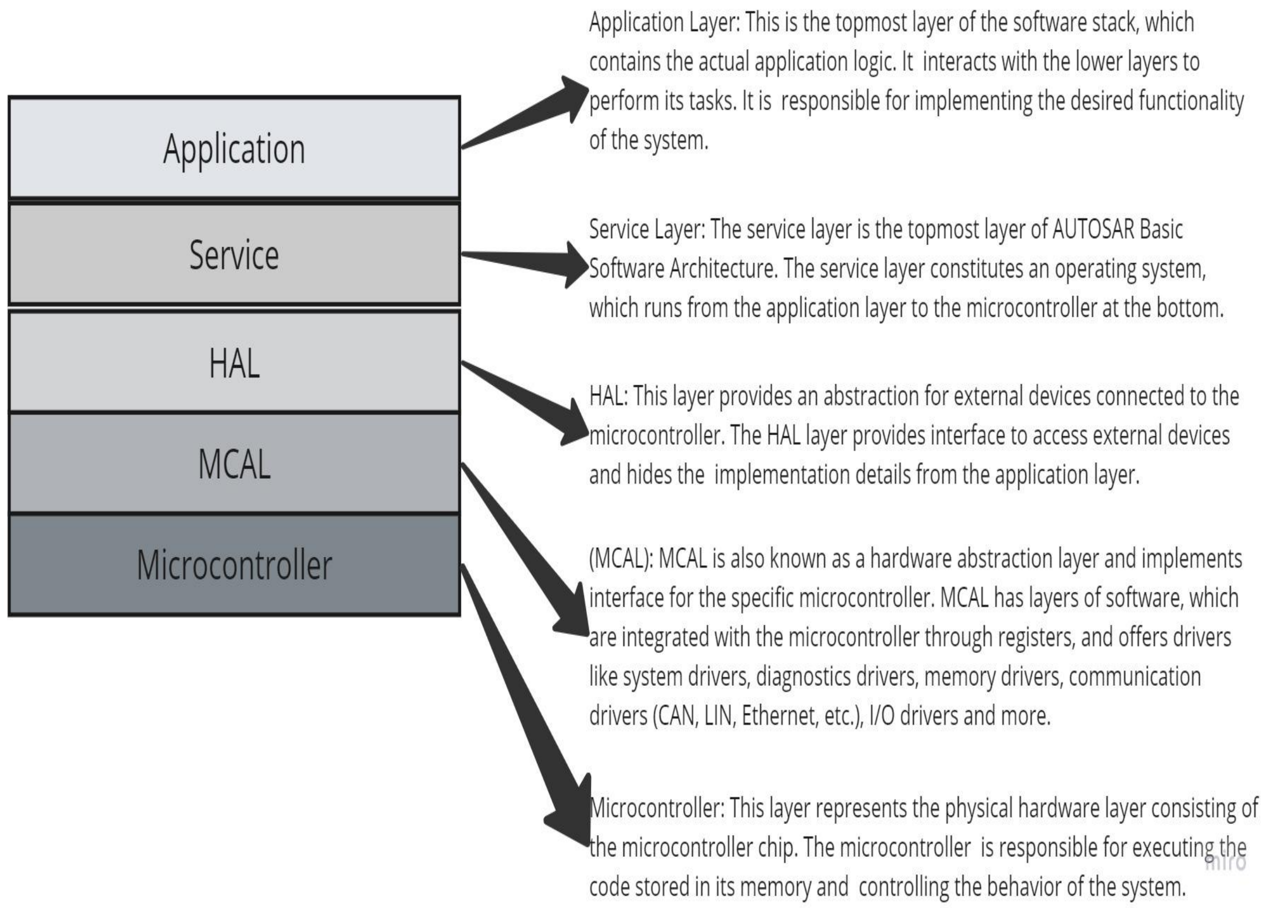
Table Of Content:

1. Introduction:

Basic Communication Manager Module acts as a resource manager, it is responsible for managing the underlying comm services. Functions of comm manager including of necessary resources for the he requested communication mode, switches to a communication mode as requested by the user, implement channel state machine for every channel to control more than one communication channel of an ECU etc.

2. High Level Design

2.1. Layered Architecture:



Application Layer: This is the topmost layer of the software stack, which contains the actual application logic. It interacts with the lower layers to perform its tasks. It is responsible for implementing the desired functionality of the system.

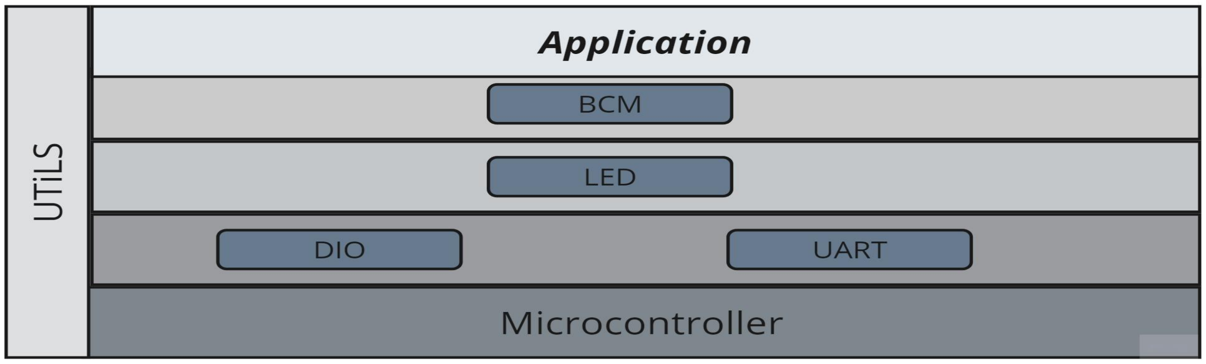
Service Layer: The service layer is the topmost layer of AUTOSAR Basic Software Architecture. The service layer constitutes an operating system, which runs from the application layer to the microcontroller at the bottom.

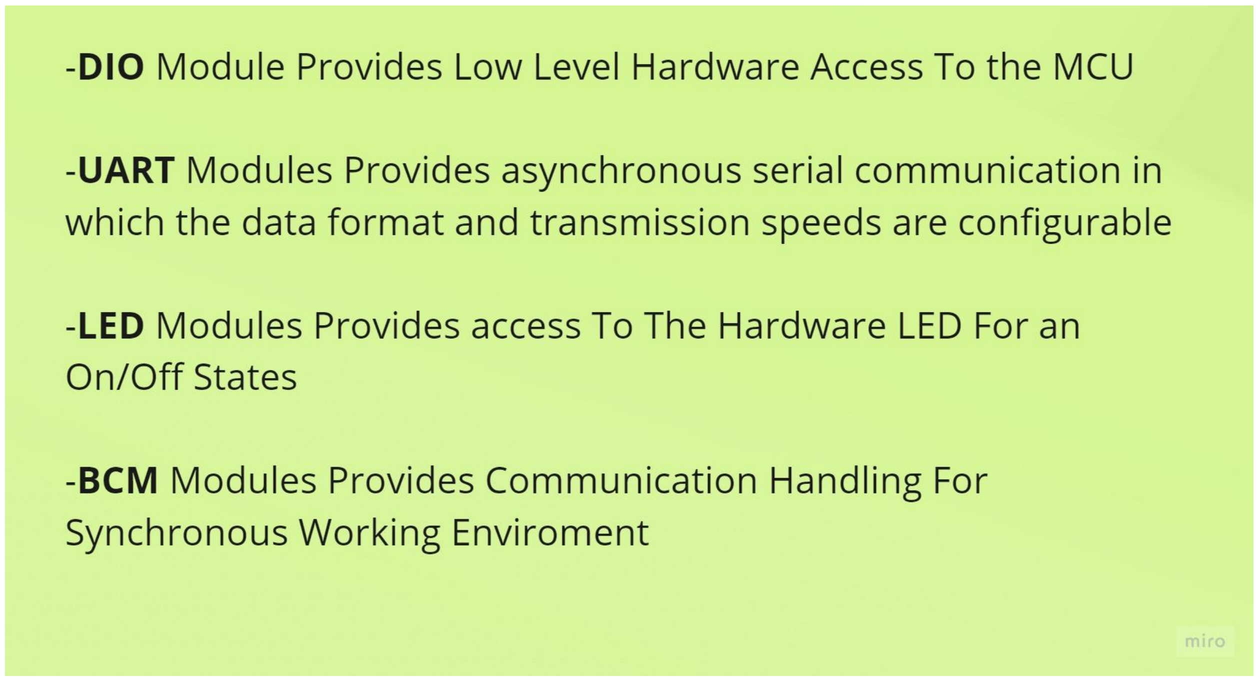
HAL: This layer provides an abstraction for external devices connected to the microcontroller. The HAL layer provides interface to access external devices and hides the implementation details from the application layer.

(MCAL): MCAL is also known as a hardware abstraction layer and implements interface for the specific microcontroller. MCAL has layers of software, which are integrated with the microcontroller through registers, and offers drivers like system drivers, diagnostics drivers, memory drivers, communication drivers (CAN, LIN, Ethernet, etc.), I/O drivers and more.

Microcontroller: This layer represents the physical hardware layer consisting of the microcontroller chip. The microcontroller is responsible for executing the code stored in its memory and controlling the behavior of the system.

2.2. Modules Description:





-**DIO** Module Provides Low Level Hardware Access To the MCU

-**UART** Modules Provides asynchronous serial communication in which the data format and transmission speeds are configurable

-**LED** Modules Provides access To The Hardware LED For an On/Off States

-**BCM** Modules Provides Communication Handling For Synchronous Working Enviroment

2.3. Drivers Documentation:

2.3.1. DIO:

Description: *The DIO (Digital Input Output) driver is responsible for setting up the digital pins of the microcontroller to either input or output mode. This driver will be used to control the buttons and LEDs.*

Functions:

```
DIO_ERROR_TYPE DIO_INITPIN(DIO_PIN_TYPE PIN,DIO_PINSTATUS_TYPE STATUS);

DIO_ERROR_TYPE DIO_WRITEPIN(DIO_PIN_TYPE PIN,DIO_VOLTAGE_TYPE VOLTAGE);

DIO_ERROR_TYPE DIO_READPIN(DIO_PIN_TYPE PIN,DIO_VOLTAGE_TYPE* VOLT);

void DIO_TogglePin(DIO_PIN_TYPE pin);
```

2.3.2. USART:

Description: *This driver enables the microcontroller to establish and control serial communication with peripheral devices using the UART protocol. It handles tasks such as configuring the UART interface, setting baud rates, and managing data transmission and reception. This driver enables the microcontroller to communicate with UART-compatible devices, such as wireless modules, GPS receivers, and Bluetooth devices, allowing for reliable and efficient data transfer.*

Functions:

```
void UART_init(enu_id_num_t ID_num);

uint8_t UART_receive_NoBlock(void);

void UART_transmit_NoBlock(uint8_t data);

void UART_receive_String_NoBlock(uint8_t*str);

void UART_SetCallBack(enu_Callback_t enu_Callback,void(*FPTR)(void));
```

2.3.3. LED:

Description: The LED driver is responsible for setting up and controlling the LEDs of the microcontroller.

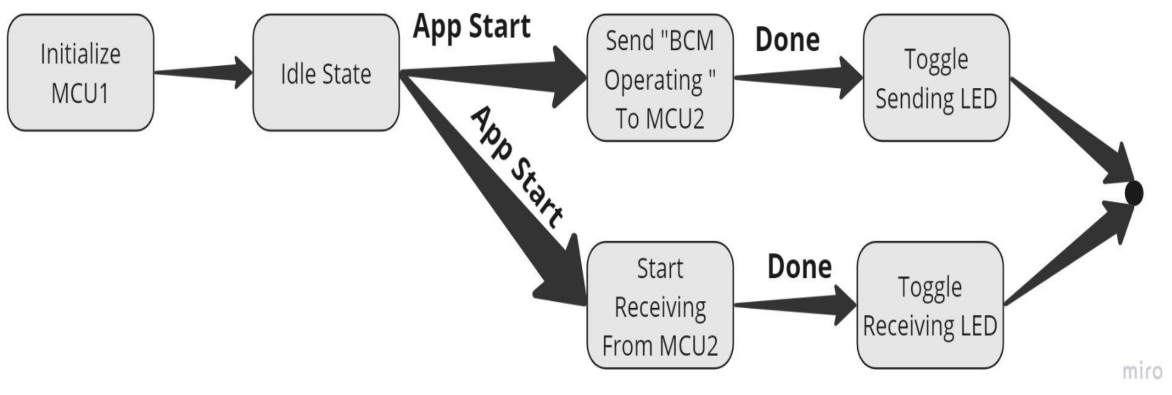
Functions:

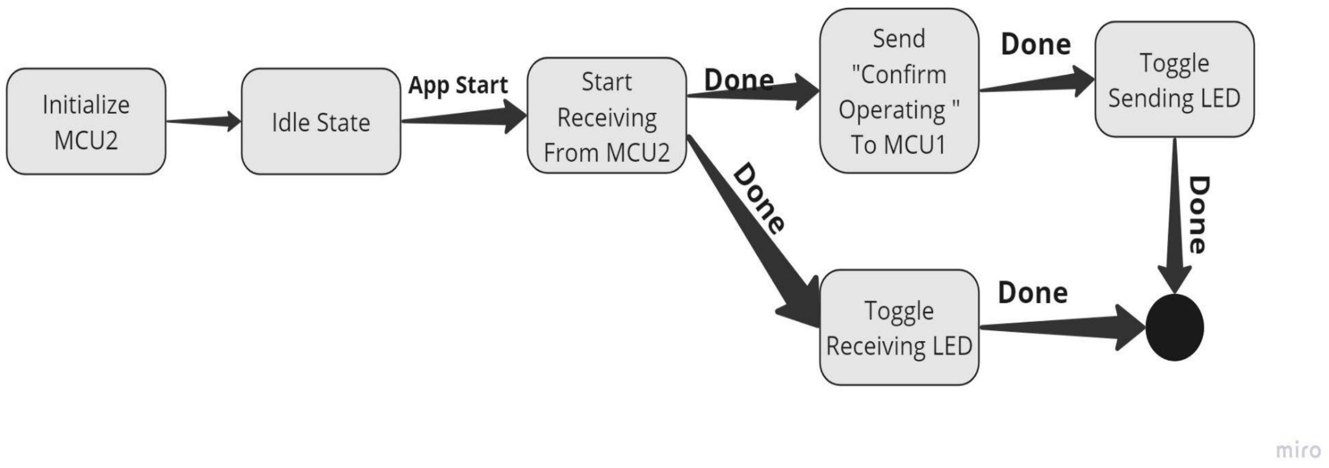
```
LED_ERROR_TYPE LED_INIT(DIO_PIN_TYPE PIN);

LED_ERROR_TYPE LED_ON(DIO_PIN_TYPE PIN);

LED_ERROR_TYPE LED_OFF(DIO_PIN_TYPE PIN);
```

2.4. State Machine:

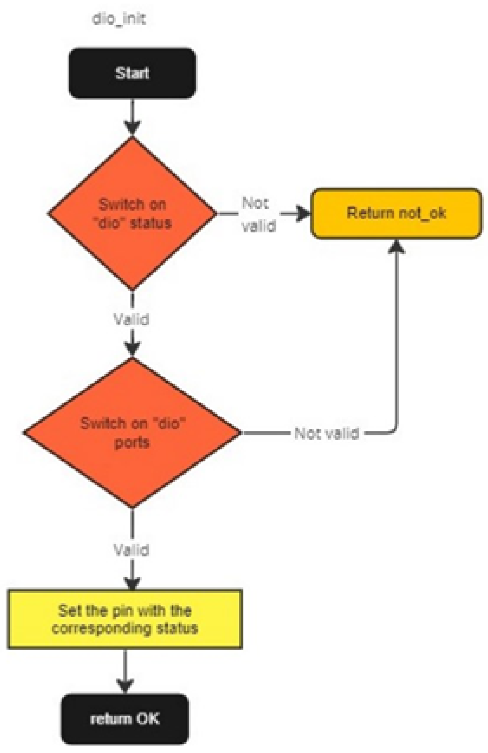


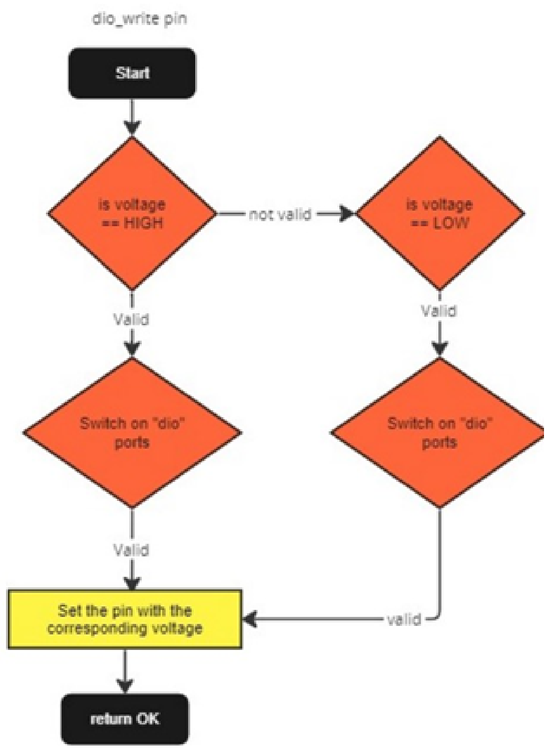


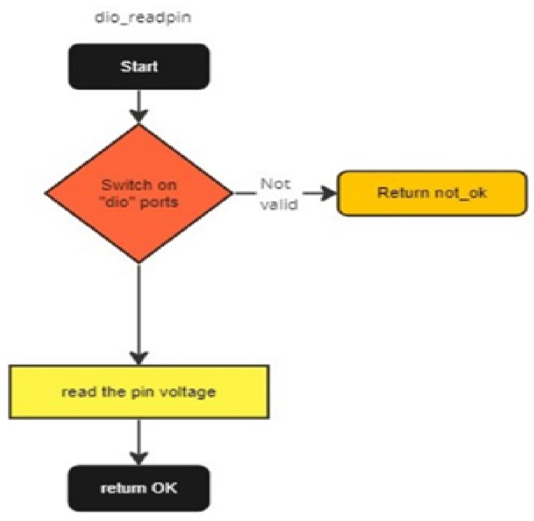
3. Low Level Design

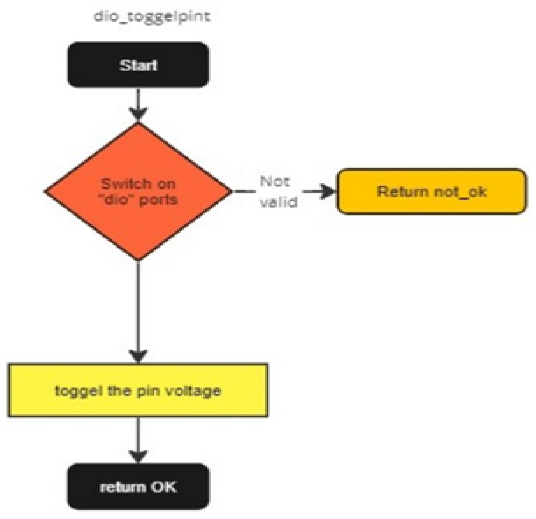
3.1. Functions FlowCharts

3.1.1. Dio:

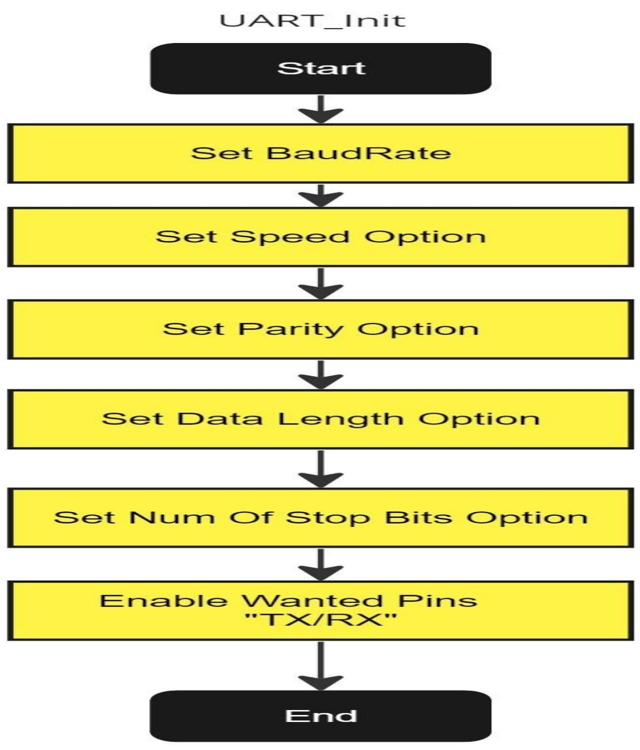


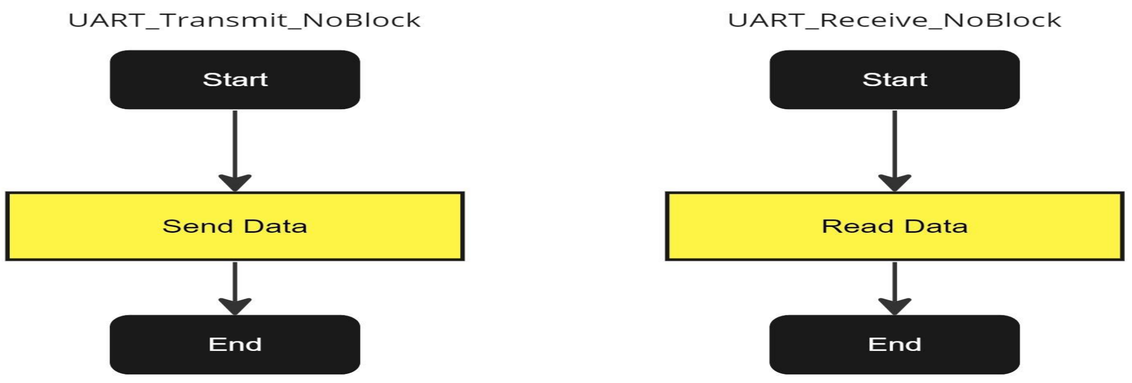


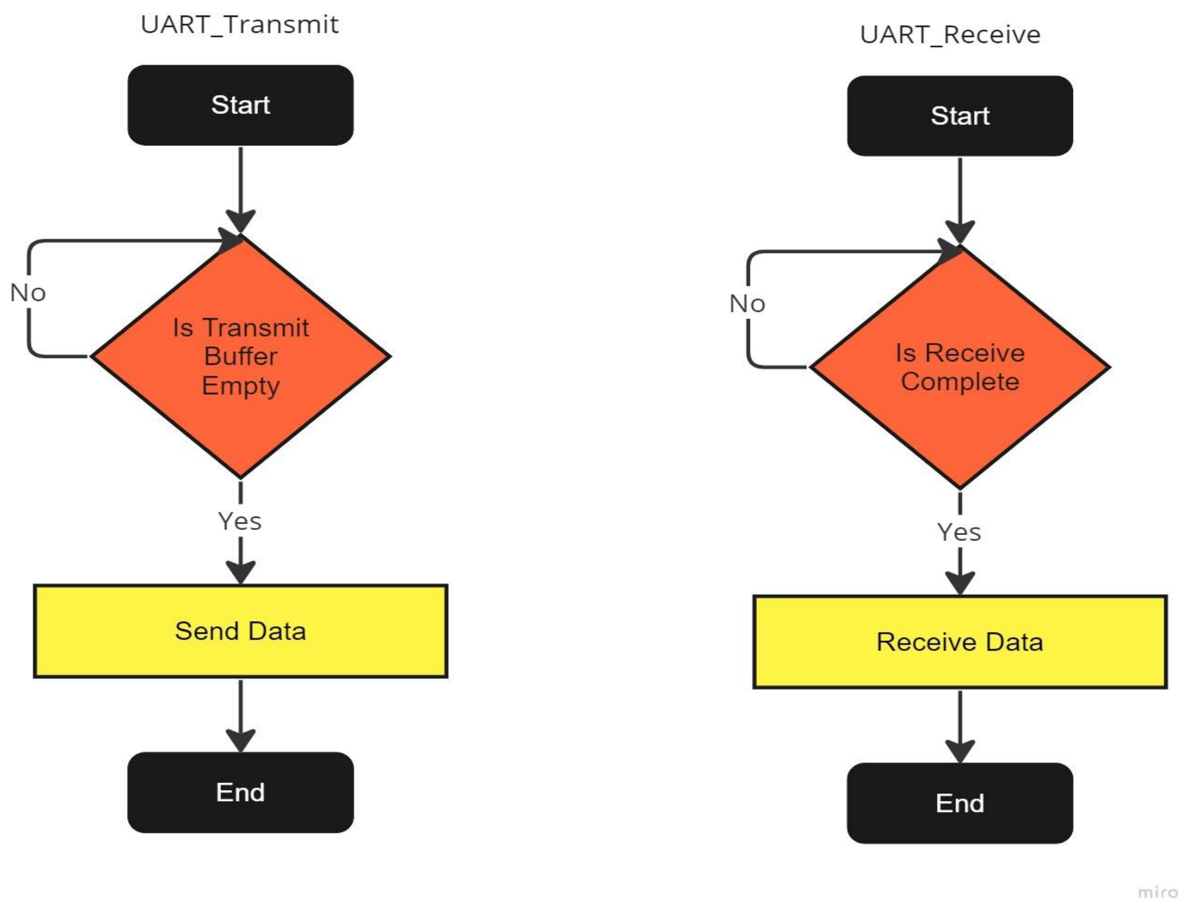




3.1.2. UART:

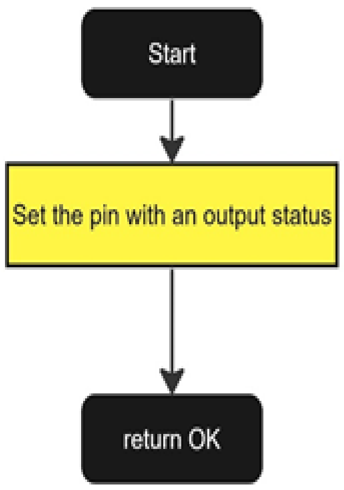

UART_Init
Start
Set BaudRate
Set Speed Option
Set Parity Option
Set Data Length Option
Set Num Of Stop Bits Option
Enable Wanted Pins "TX/RX"
End


UART_Transmit_NoBlock
Start
Send Data
End
UART_Receive_NoBlock
Start
Read Data
End

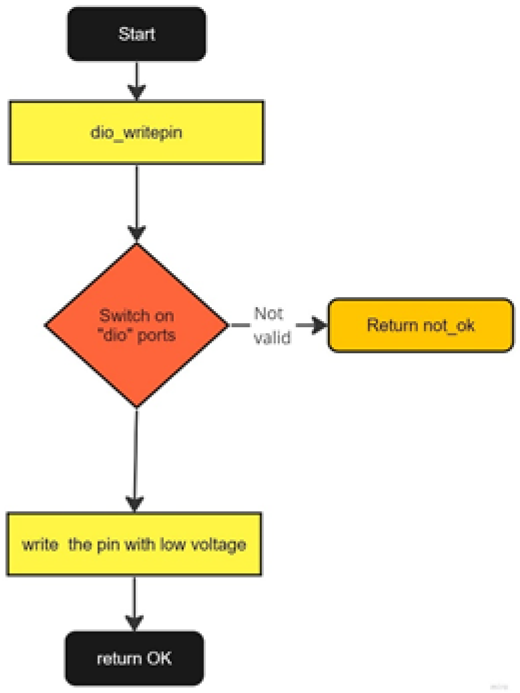

UART_Transmit
Start
No
Is Transmit Buffer Empty
Yes
Send Data
End
UART_Receive
Start
No
Is Receive Complete
Yes
Receive Data
End
miro

3.1.3. LED:

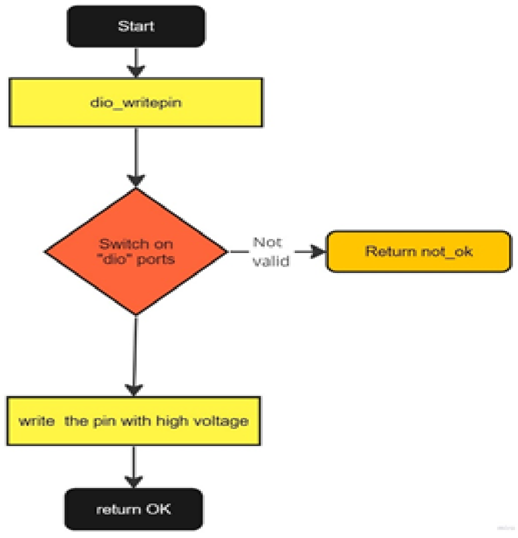
led_init



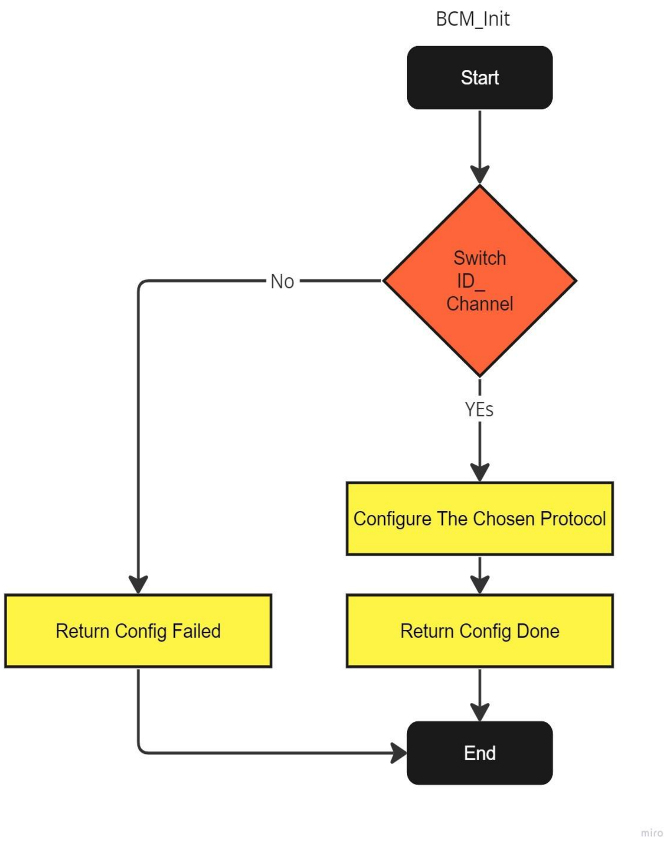
led_off

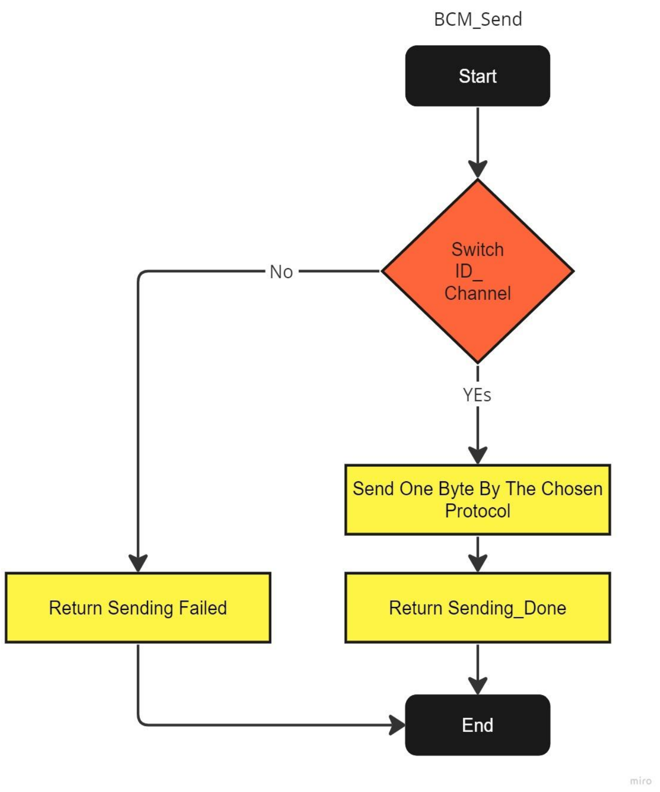


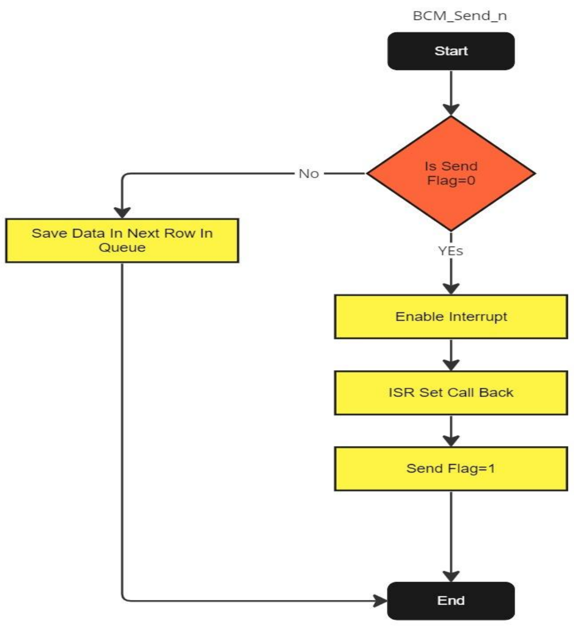
led_on

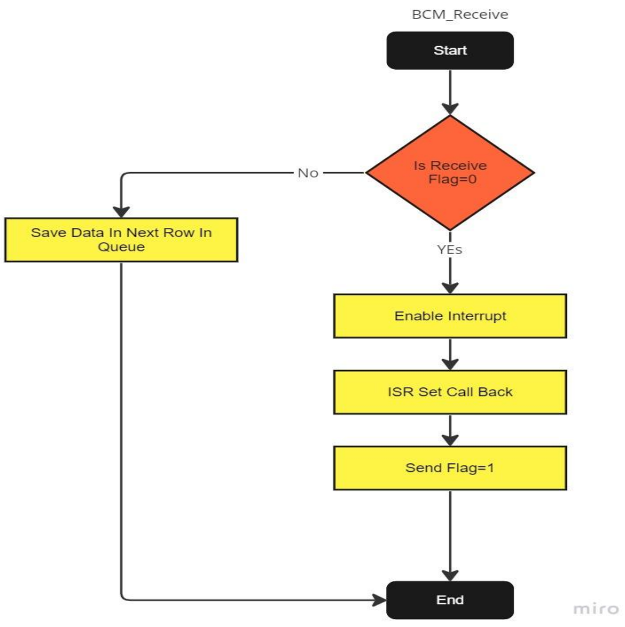


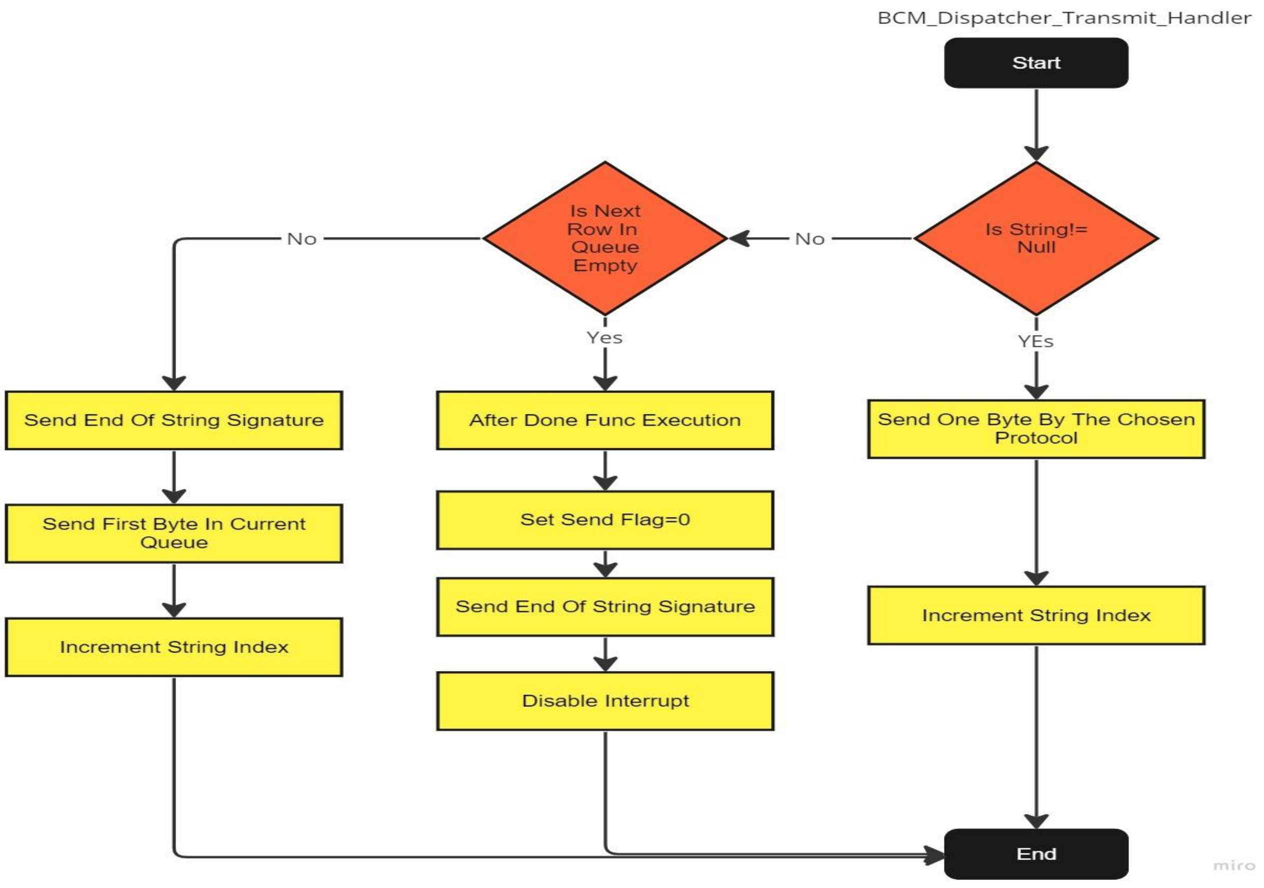
3.1.4. BCM:


BCM_Init
Start
Switch ID_ Channel
No
YEs
Configure The Chosen Protocol
Return Config Failed
Return Config Done
End
miro

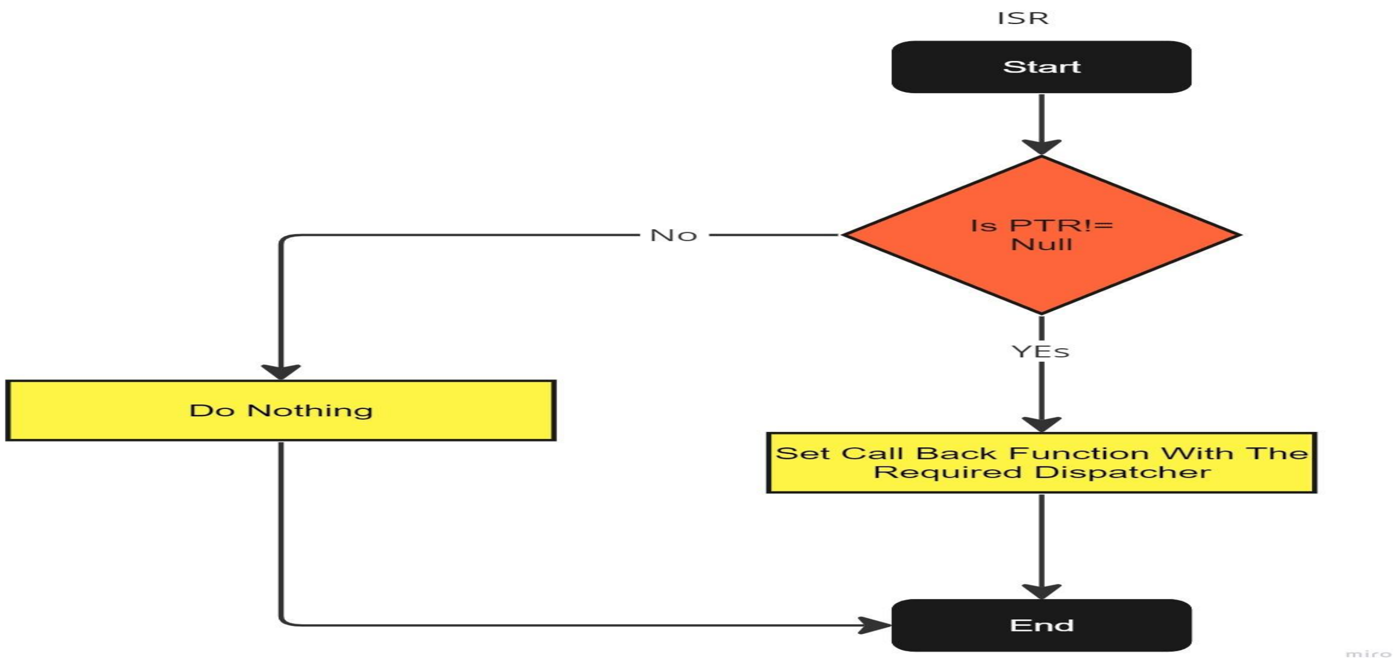

BCM_Send
Start
Switch ID_ Channel
No
YEs
Send One Byte By The Chosen Protocol
Return Sending Failed
Return Sending_Done
End
miro


BCM_Send_n
Start
Is Send Flag=0
No
YEs
Save Data In Next Row In Queue
Enable Interrupt
ISR Set Call Back
Send Flag=1
End

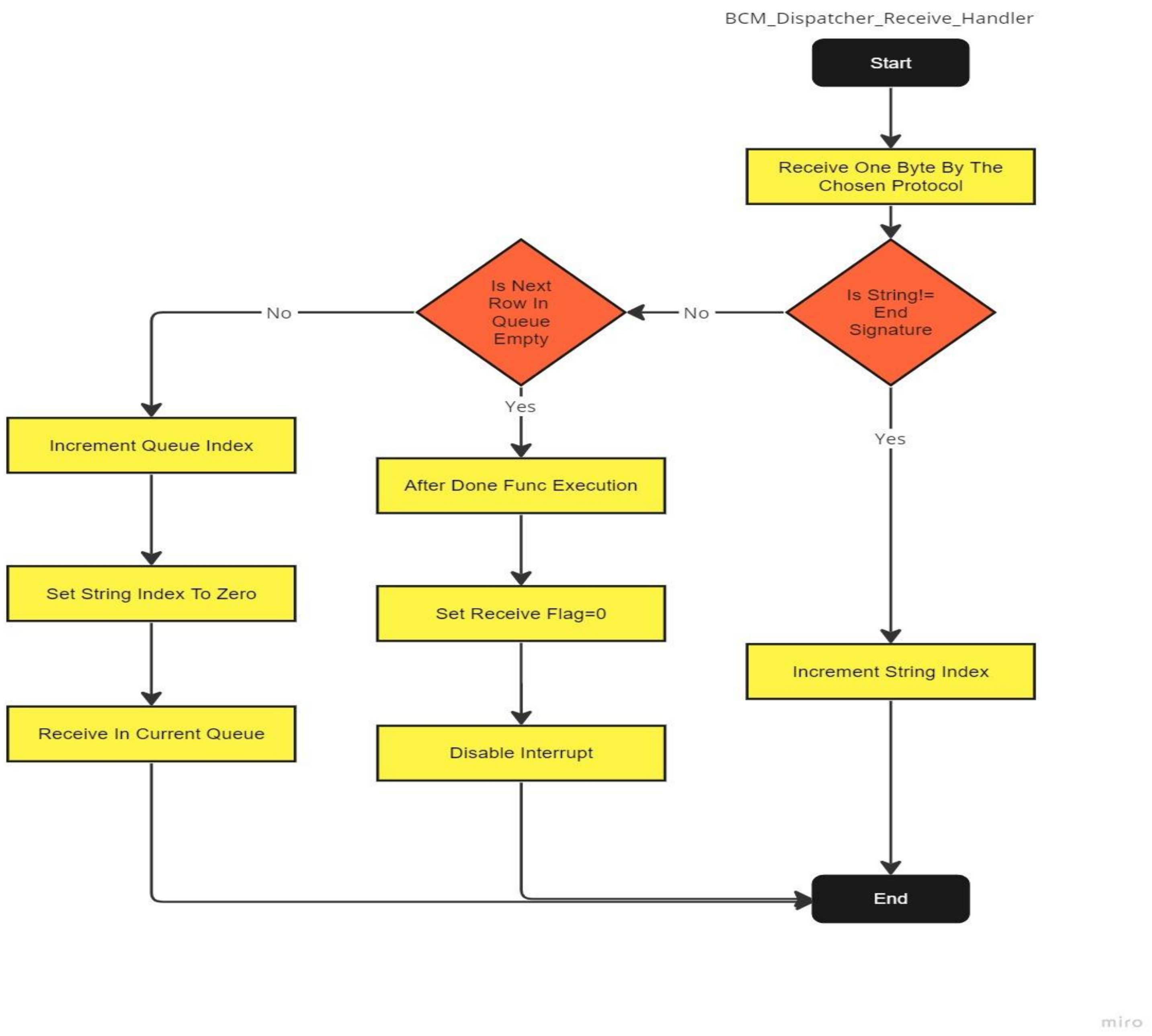

BCM_Receive
Start
Is Receive Flag=0
No
YEs
Save Data In Next Row In Queue
Enable Interrupt
ISR Set Call Back
Send Flag=1
End
miro



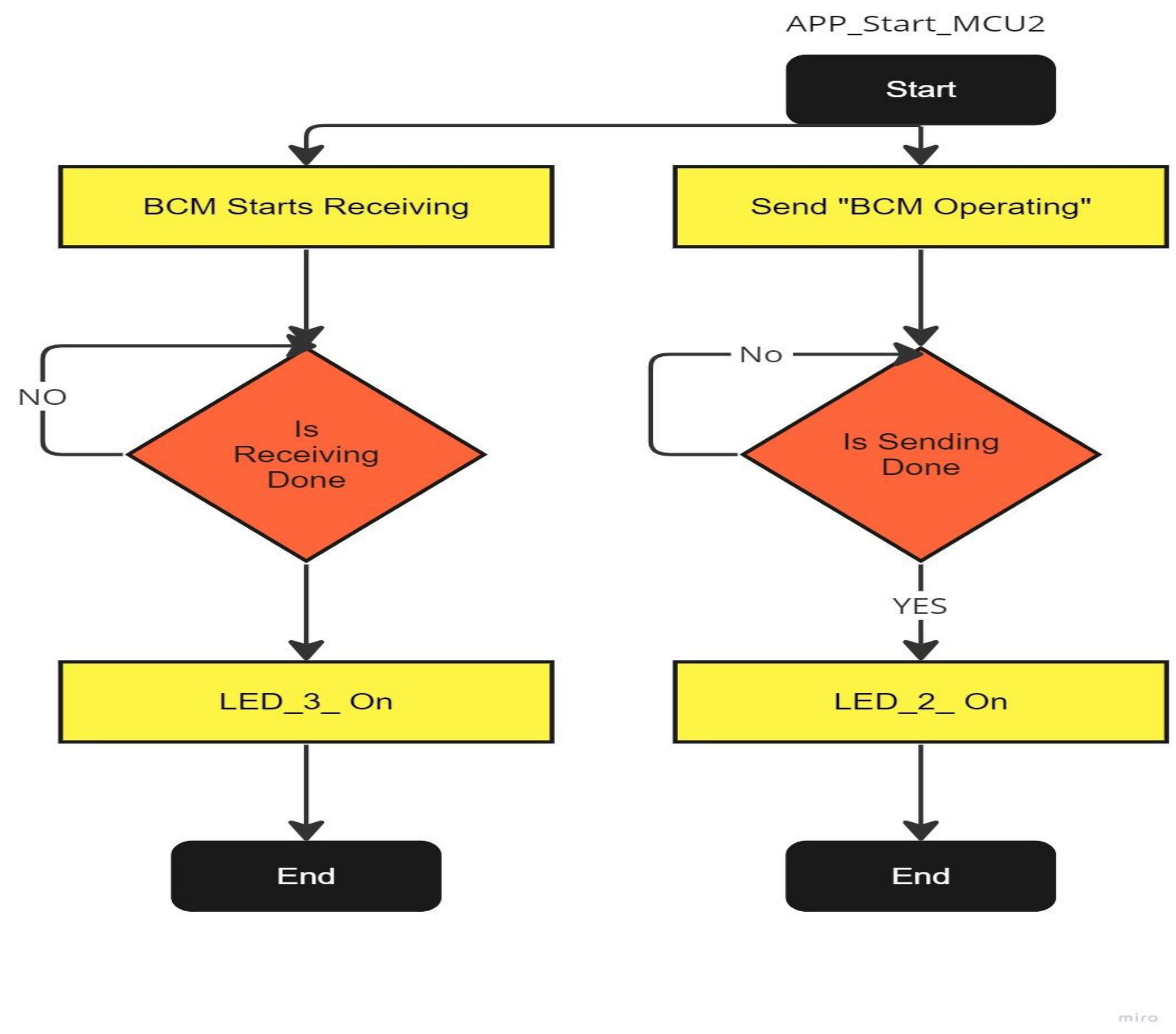
BCM_Dispatcher_Transmit_Handler
Start
Is String!= Null
No
YEs
Is Next Row In Queue Empty
No
Yes
Send End Of String Signature
Send First Byte In Current Queue
Increment String Index
After Done Func Execution
Set Send Flag=0
Send End Of String Signature
Disable Interrupt
Send One Byte By The Chosen Protocol
Increment String Index
End
miro



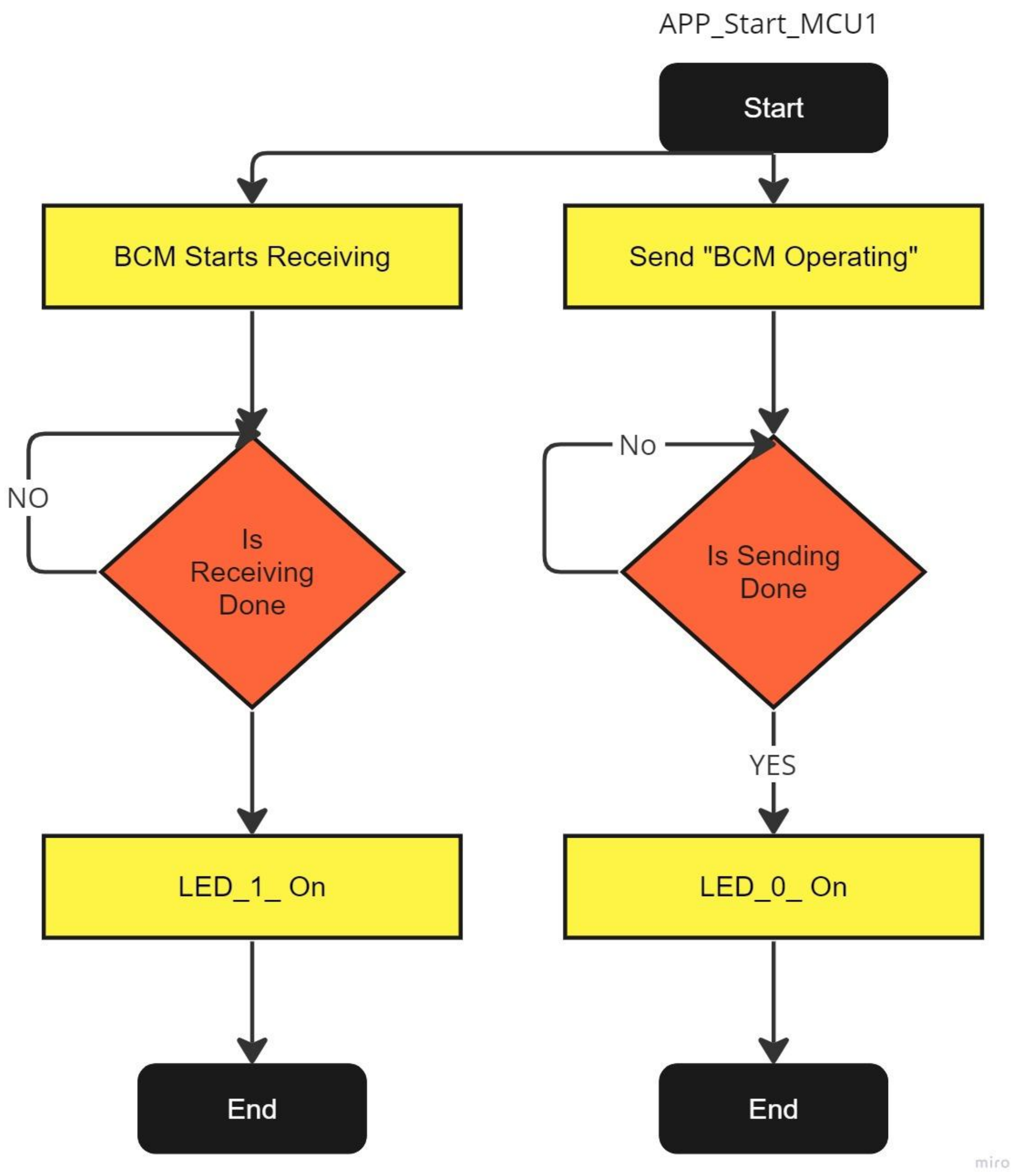
ISR
Start
Is PTR!= Null
No
YEs
Do Nothing
Set Call Back Function With The Required Dispatcher
End
miro


BCM_Dispatcher_Receive_Handler
Start
Receive One Byte By The Chosen Protocol
Is String!= End Signature
No
Is Next Row In Queue Empty
No
Yes
Yes
Increment Queue Index
Set String Index To Zero
Receive In Current Queue
After Done Func Execution
Set Receive Flag=0
Disable Interrupt
Increment String Index
End
miro

3.1.5. APP:

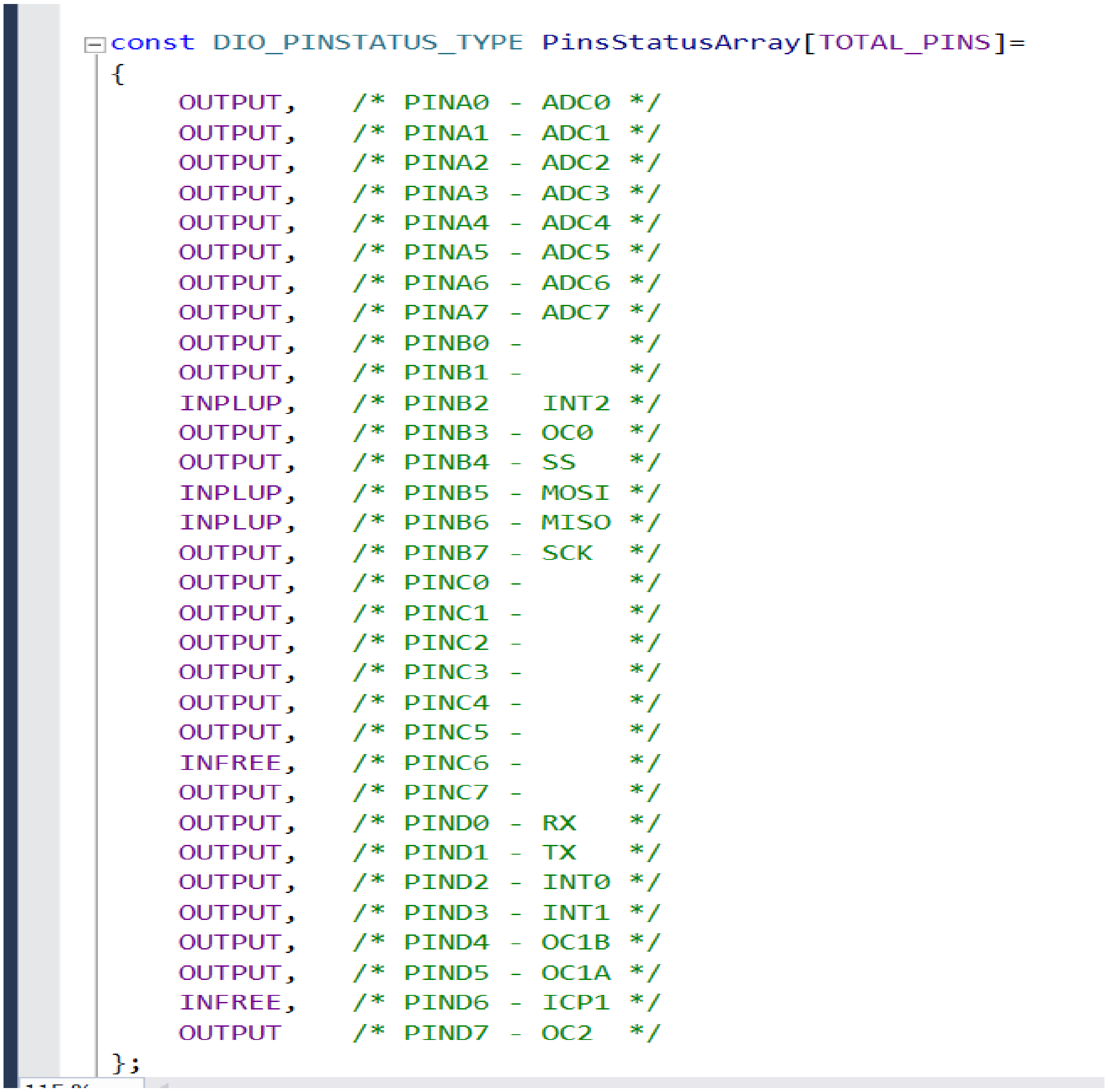


APP_Start_MCU2
Start
BCM Starts Receiving
Send "BCM Operating"
NO
Is Receiving Done
No
Is Sending Done
YES
LED_3_ On
LED_2_ On
End
End
miro


APP_Start_MCU1
Start
BCM Starts Receiving
Send "BCM Operating"
NO
Is Receiving Done
No
Is Sending Done
YES
LED_1_ On
LED_0_ On
End
End
miro

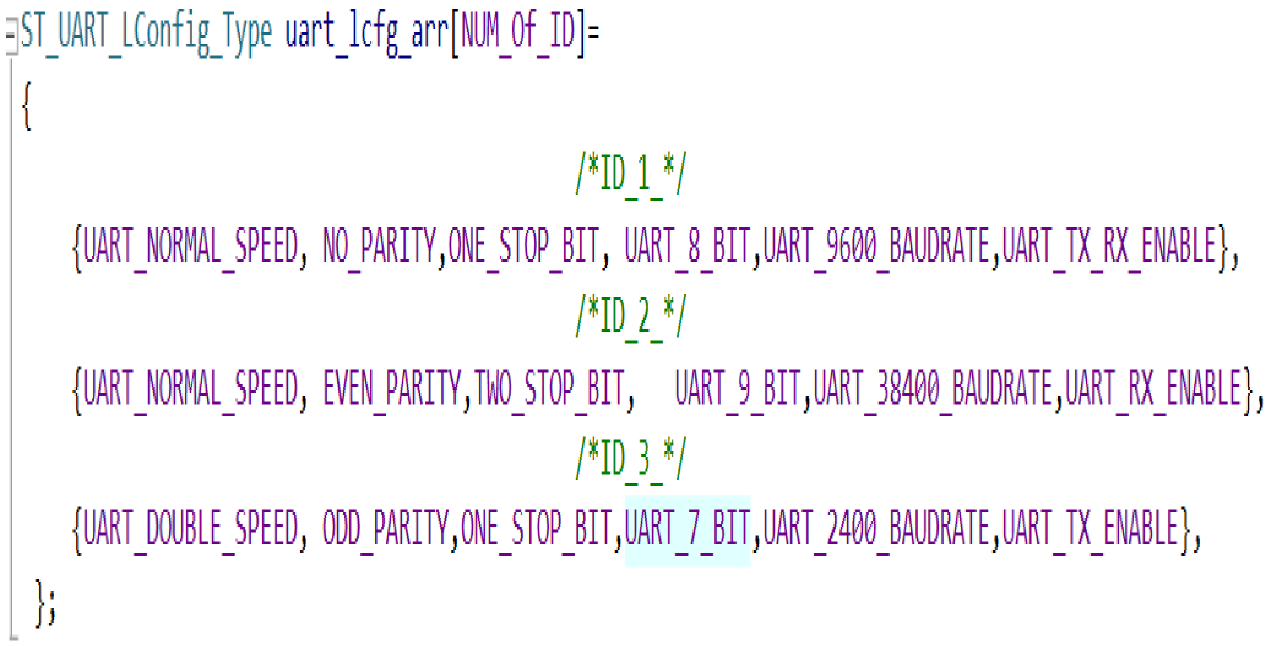
3.2. Linking Configurations

3.2.1. Dio:



```
const DIO_PINSTATUS_TYPE PinsStatusArray[TOTAL_PINS]=
{
    OUTPUT,    /* PINA0 - ADC0 */
    OUTPUT,    /* PINA1 - ADC1 */
    OUTPUT,    /* PINA2 - ADC2 */
    OUTPUT,    /* PINA3 - ADC3 */
    OUTPUT,    /* PINA4 - ADC4 */
    OUTPUT,    /* PINA5 - ADC5 */
    OUTPUT,    /* PINA6 - ADC6 */
    OUTPUT,    /* PINA7 - ADC7 */
    OUTPUT,    /* PINB0 -      */
    OUTPUT,    /* PINB1 -      */
    INPLUP,    /* PINB2   INT2 */
    OUTPUT,    /* PINB3 - OC0  */
    OUTPUT,    /* PINB4 - SS   */
    INPLUP,    /* PINB5 - MOSI */
    INPLUP,    /* PINB6 - MISO */
    OUTPUT,    /* PINB7 - SCK  */
    OUTPUT,    /* PINC0 -      */
    OUTPUT,    /* PINC1 -      */
    OUTPUT,    /* PINC2 -      */
    OUTPUT,    /* PINC3 -      */
    OUTPUT,    /* PINC4 -      */
    OUTPUT,    /* PINC5 -      */
    INFREE,    /* PINC6 -      */
    OUTPUT,    /* PINC7 -      */
    OUTPUT,    /* PIND0 - RX   */
    OUTPUT,    /* PIND1 - TX   */
    OUTPUT,    /* PIND2 - INT0 */
    OUTPUT,    /* PIND3 - INT1 */
    OUTPUT,    /* PIND4 - OC1B */
    OUTPUT,    /* PIND5 - OC1A */
    INFREE,    /* PIND6 - ICP1 */
    OUTPUT     /* PIND7 - OC2  */
};
```

3.2.2. UART:



```
ST_UART_LConfig_Type uart_lcfg_arr[NUM_Of_ID]=
{
                                          /*ID_1_*/
   {UART_NORMAL_SPEED, NO_PARITY,ONE_STOP_BIT, UART_8_BIT,UART_9600_BAUDRATE,UART_TX_RX_ENABLE},
                                          /*ID_2_*/
   {UART_NORMAL_SPEED, EVEN_PARITY,TWO_STOP_BIT,   UART_9_BIT,UART_38400_BAUDRATE,UART_RX_ENABLE},
                                          /*ID_3_*/
   {UART_DOUBLE_SPEED, ODD_PARITY,ONE_STOP_BIT,UART_7_BIT,UART_2400_BAUDRATE,UART_TX_ENABLE},
 };
```