DESIGN DOCUMENT

BASIC COMMUNICATION MANAGER (BCM)

Prepared by:

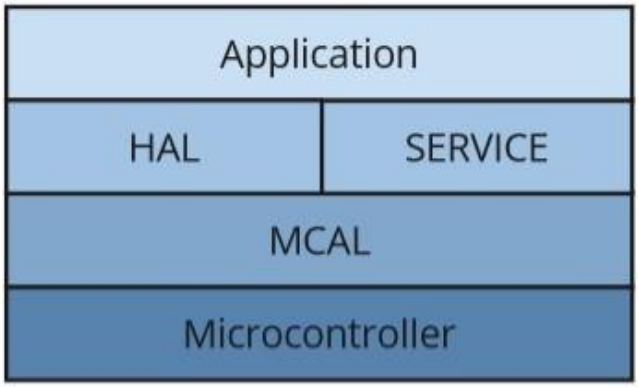
Amr Gamal El-Abd

Table of Contents: -

Project introduction:

The project aims to design a basic communication management system to handle a series of events of sending and receiving on different communication protocols , we applied this exact task to handle the functionalities of UART communication protocol with the other protocols coming on future upgrades.

Layered Architectures:-



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.

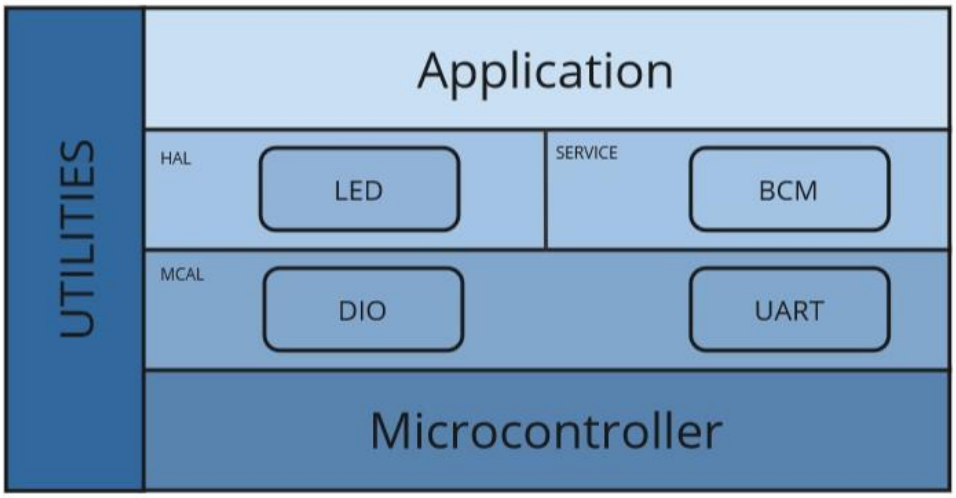
SERVICE Layer: This layer plays a key role in abstracting and encapsulating the low-level hardware details provided by the MCAL. It provides a set of services, functions, and interfaces that enable the application layer to interact with the underlying hardware

HAL Layer: 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 Layer (Microcontroller Abstraction Layer): This layer provides an abstraction for the microcontroller hardware. It includes low-level drivers for peripherals. It hides the hardware details and provides a uniform interface to the upper layers.

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.

Modules description:



BCM Module:

Description: The BCM will serve as a crucial component for establishing seamless and reliable communication between devices. By using the capabilities of UART, SPI, and I2C protocols. the BCM will enable efficient transmission and reception of data, allowing devices to exchange information in a serial asynchronous manner.

LED Module:

Description: The LED driver is responsible for setting up and controlling the LEDs of the microcontroller. This driver will be used to indicate the status of the BCM.

UART Module

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.

DIO Module:

Description: The DIO (Digital Input Output) driver is responsible for setting up the digital pins of the microcontroller to either input or output mode.

Drivers' documentation:-

Service layer:

BCM Driver:

Description : The BCM will serve as a crucial component for establishing seamless and reliable communication between devices. By using the capabilities of UART, SPI, and I2C protocols. the BCM will enable efficient transmission and reception of data, allowing devices to exchange information in a serial asynchronous manner.

Functions:

```
enu_system_status_t bcm_init (enu_protocol_t enu_protocol);
enu_system_status_t bcm_deinit (enu_protocol_t enu_protocol);
void bcm_send(u8 a_u8_data);
void bcm_send_n (u8 *u8_a_ptr_data);
void bcm_receive (u8 * u8_ptr);

void bcm_send_dispatcher (void);
void bcm_receive_dispatcher (void);

void bcm_sendComplete_callback(void(*local_fptr)(void));
void bcm_receiveComplete_callback(void(*local_fptr)(void));
```

HAL drivers:

1. LED Driver:

Description: The LED driver is responsible for setting up and controlling the LEDs of the microcontroller. This driver will be used to indicate the status of the BCM.

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);
```

MCAL drivers:

1. DIO Driver:

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);
```

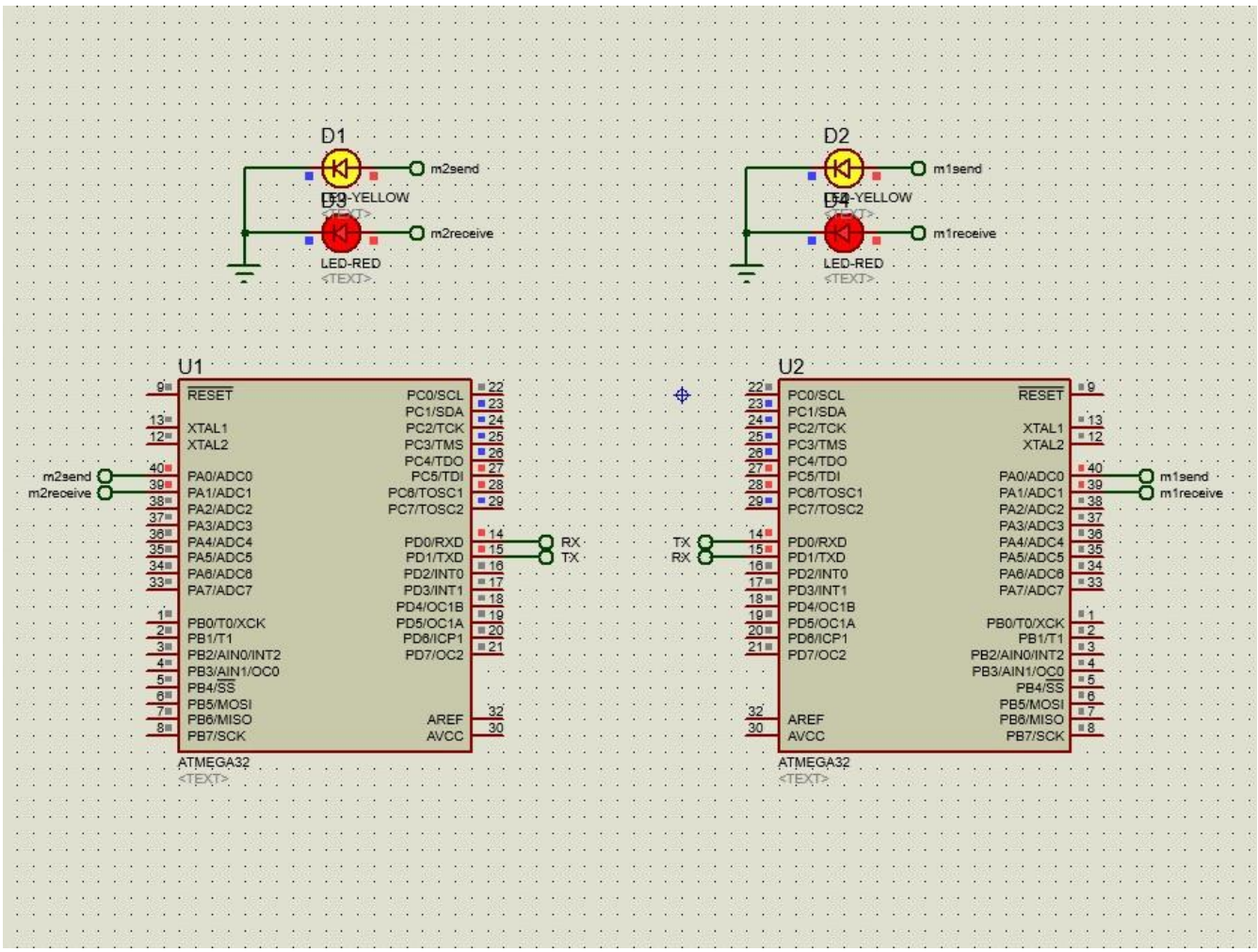
2. UART communication protocol:-

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.

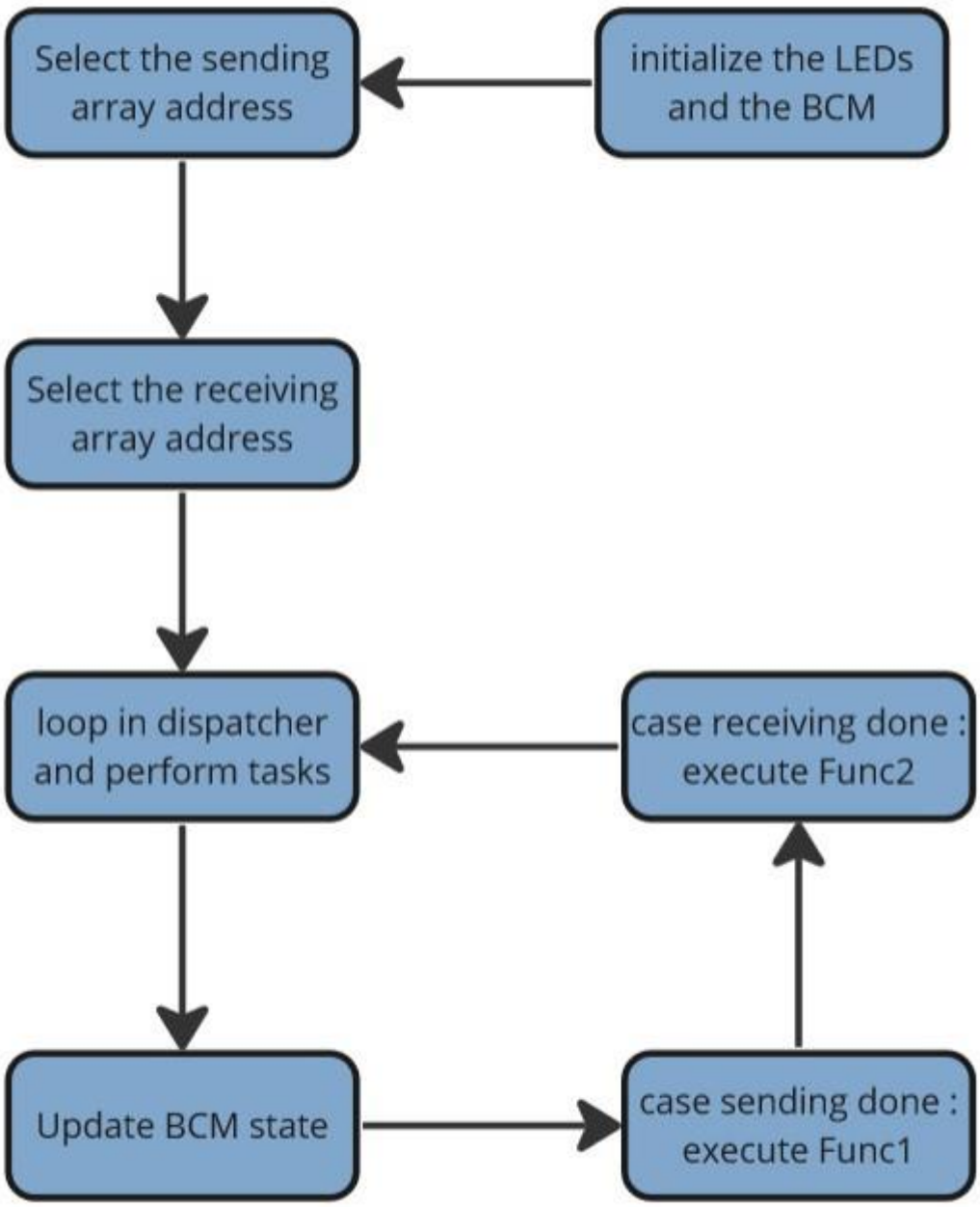
Functions:

```
enu_return_error_t uart_init(str_uart_config_t* ptr_str_uart_config);
void uart_transmit(u8 data);
u8 uart_receive(void);
void uart_Transmit_string (u8 *str);
u8 uart_receive_string (u8 *string);

void uart_RX_SetCallBack (void(*local_fptr)(void));
void uart_TX_SetCallBack (void(*local_fptr)(void));

void uart_RX_intEnable (void);
void uart_RX_intDisable(void);

void uart_TX_intEnable (void);
void uart_TX_intDisable(void);
```

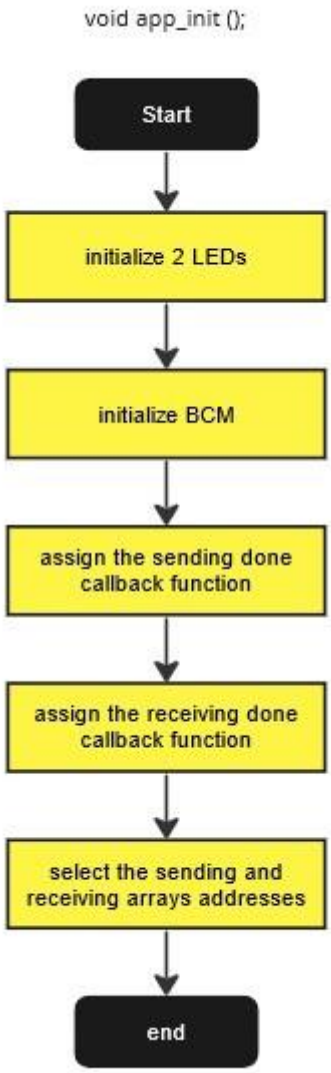
Proteus simulation design :-

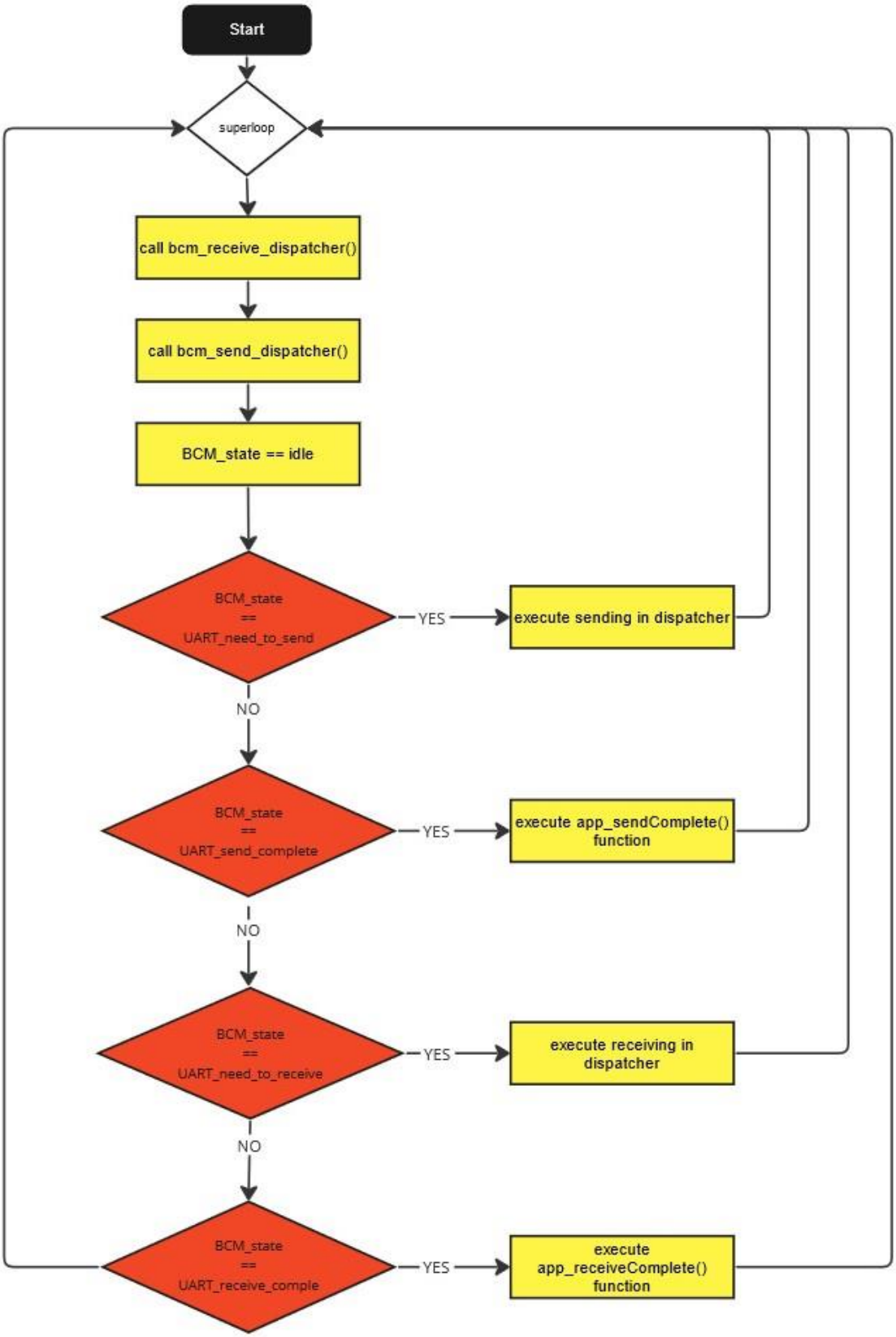


State machine for APP:



App flowcharts:




Start
superloop
call bcm_receive_dispatcher()
call bcm_send_dispatcher()
BCM_state == idle
BCM_state == UART_need_to_send
YES
execute sending in dispatcher
NO
BCM_state == UART_send_complete
YES
execute app_sendComplete() function
NO
BCM_state == UART_need_to_receive
YES
execute receiving in dispatcher
NO
BCM_state == UART_receive_comple
YES
execute app_receiveComplete() function

Linking config.

```
const DIO_PinStatus_type  PinsStatusArray[TOTAL_PINS]={
    OUTPUT,      /* Port A Pin 0 LED1*/
    OUTPUT,      /* Port A Pin 1 LED2*/
    OUTPUT,      /* Port A Pin 2 */
    OUTPUT,      /* Port A Pin 3 */
    OUTPUT,      /* Port A Pin 4 */
    OUTPUT,      /* Port A Pin 5 */
    OUTPUT,      /* Port A Pin 6 */
    OUTPUT,      /* Port A Pin 7 */
    OUTPUT,      /* Port B Pin 0 */
    OUTPUT,      /* Port B Pin 1 */
    OUTPUT,      /* Port B Pin 2 */
    OUTPUT,      /* Port B Pin 3 */
    OUTPUT,      /* Port B Pin 4 */
    OUTPUT,      /* Port B Pin 5 */
    OUTPUT,      /* Port B Pin 6 */
    OUTPUT,      /* Port B Pin 7 */
    OUTPUT,      /* Port C Pin 0 */
    OUTPUT,      /* Port C Pin 1 */
    OUTPUT,      /* Port C Pin 2 */
    OUTPUT,      /* Port C Pin 3 */
    OUTPUT,      /* Port C Pin 4 */
    OUTPUT,      /* Port C Pin 5 */
    OUTPUT,      /* Port C Pin 6 */
    OUTPUT,      /* Port C Pin 7 */
    OUTPUT,      /* Port D Pin 0 */
    OUTPUT,      /* Port D Pin 1 */
    OUTPUT,      /* Port D Pin 2 */
    OUTPUT,      /* Port D Pin 3 */
    OUTPUT,      /* Port D Pin 4 */
    OUTPUT,      /* Port D Pin 5 */
    OUTPUT,      /* Port D Pin 6 */
    OUTPUT       /* Port D Pin 7 */
};
```

```
str_uart_config_t g_str_uart_config =
{
normal_speed,
transmit_receive_enable,
no_parity,
asynchronous,
one_stop_bit,
_8_data_bits
};
```