Basic communication manager

Sarah Mohamed Salah

1. Project Introduction

Project aims to use all protocols in basic communication manager

1.1. Project Components

- TWO ATmega32 microcontroller
- TWO leds

1.2. System Requirements

2. High Level Design

2.1. System Architecture

2.2. Modules Description

2.2.1. DIO (Digital Input/Output) Module

The *DIO* 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.

2.2.2. LED Module

The *LED* module is responsible for turning on or off based on signal of microcontroller

2.3. Drivers' Documentation (APIs)

2.3.1 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.

2.3.2. MCAL APIs

2.3.2.1. DIO Driver

```
/*
 * Function to set the direction of a given port
 *
 * This function takes an 8-bit value and sets the direction of each
 * pin in the given port according to the corresponding bit value
 *
 * Parameters
 *          [in] en_a_port    The port to set the direction of
 *          [in] u8_a_portDir The desired port direction
 *
 *
 * Return
 *       en_DIO_error_t value that indicates operation success/failure
 *               (DIO_OK in case of success or DIO_ERROR in case of failure)
 */
en_DIO_error_t DIO_setPortDir(en_DIO_port_t en_a_port,  u8 u8_a_portDir);
/*
 * Function to set the value of a given port
 *
 * This function takes an 8-bit value and sets the value of each
 * pin in the given port according to the corresponding bit value
 *
 * Parameters
 *          [in] en_a_port    The port to set the value of
 *          [in] u8_a_portVal The desired port value
 *
 * Return
 *       en_DIO_error_t value that indicates operation success/failure
 *               (DIO_OK in case of success or DIO_ERROR in case of failure)
 */
en_DIO_error_t DIO_setPortVal(en_DIO_port_t en_a_port,  u8 u8_a_portVal);

/*
 * Function to set the direction of a given pin
 *
 * This function takes an en_DIO_pinDir_t value and sets the direction
 * of the given pin accordingly
 */
```

```
| Parameters
|          [in] en_a_port   The port of the desired pin
|          [in] en_a_pin    The desired pin to set direction of
|          [in] en_a_pinDir The desired pin direction (INPUT/OUTPUT)
|
|
| Return
|       en_DIO_error_t value that indicates operation success/failure
|               (DIO_OK in case of success or DIO_ERROR in case of failure)
|
en_DIO_error_t DIO_setPinDir (en_DIO_port_t en_a_port, en_DIO_pin_t en_a_pin,
en_DIO_pinDir_t en_a_pinDir);

|
|  Function to set the value of a given pin
|
| This function takes an en_DIO_level_t value and sets the value
| of the given pin accordingly
|
| Parameters
|          [in] en_a_port   The port of the desired pin
|          [in] en_a_pin    The desired pin to set value of
|          [in] en_a_pinDir The desired pin value (HIGH/LOW)
|
|
| Return
|       en_DIO_error_t value that indicates operation success/failure
|               (DIO_OK in case of success or DIO_ERROR in case of failure)
|
en_DIO_error_t DIO_setPinVal (en_DIO_port_t en_a_port, en_DIO_pin_t en_a_pin,
en_DIO_level_t en_a_pinVal);
```

```
|
|  Function to toggle the value of a given pin
|
| If the pin value is high, this function sets it to low
| and if it is low it sets it to high
```

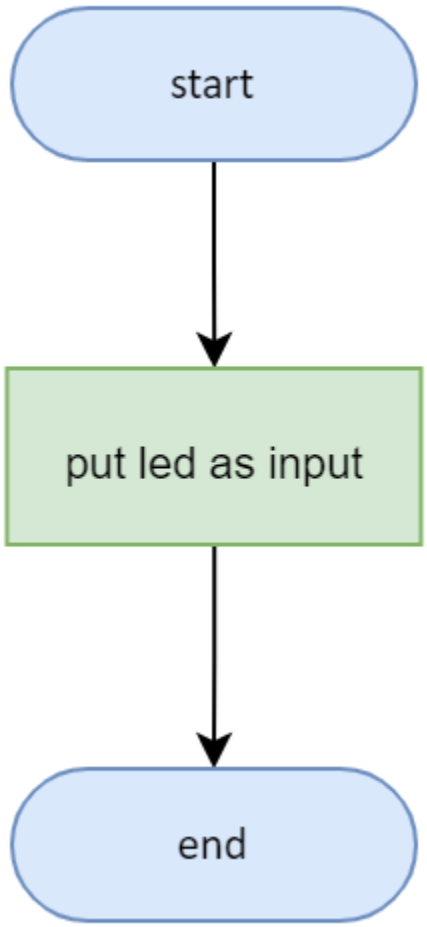
```
|
| Parameters
|          [in] en_a_port    The port of the desired pin
|          [in] en_a_pin     The desired pin to toggle value of
|
|
| Return
|       en_DIO_error_t value that indicates operation success/failure
|               (DIO_OK in case of success or DIO_ERROR in case of failure)
|
en_DIO_error_t DIO_togPinVal (en_DIO_port_t en_a_port, en_DIO_pin_t
en_a_pin);
```

```
|
|  Function to get the value of a given pin
|
| This function reads the value of the given pin and
| returns the value in the given address
|
| Parameters
|          [in] en_a_port    The port of the desired pin
|          [in] en_a_pin     The desired pin to read value of
|          [out] pu8_a_Val   address to return the pin value into
|
|
| Return
|       en_DIO_error_t value that indicates operation success/failure
|               (DIO_OK in case of success or DIO_ERROR in case of failure)
|
en_DIO_error_t DIO_getPinVal (en_DIO_port_t en_a_port, en_DIO_pin_t en_a_pin,
u8* pu8_a_Val);
```

3.2. HAL Layer

3.2.1. LED Module

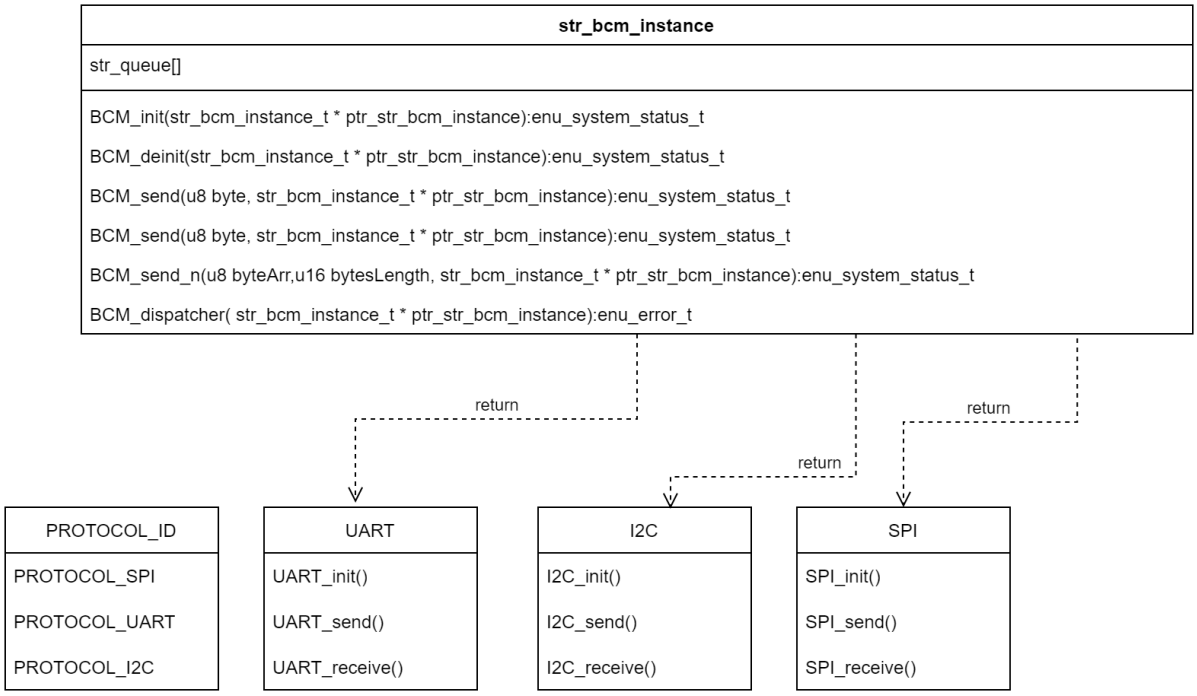
3.2.1.1. LED_init



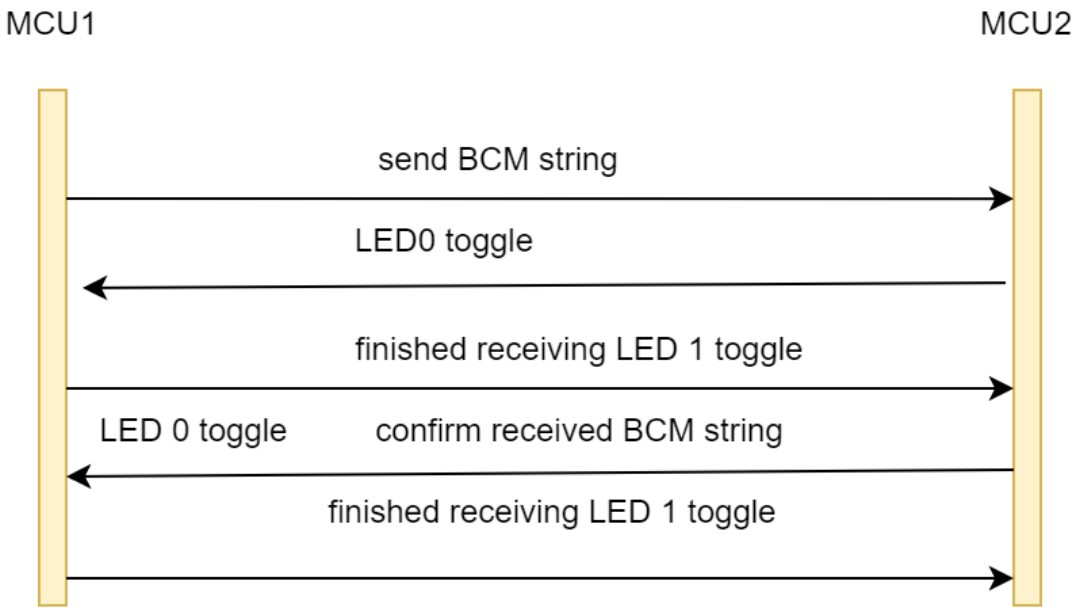
3.2.1.1. LED_on

2.4. UML

2.4.1 UML DIAGRAM



2.5. Sequence diagram

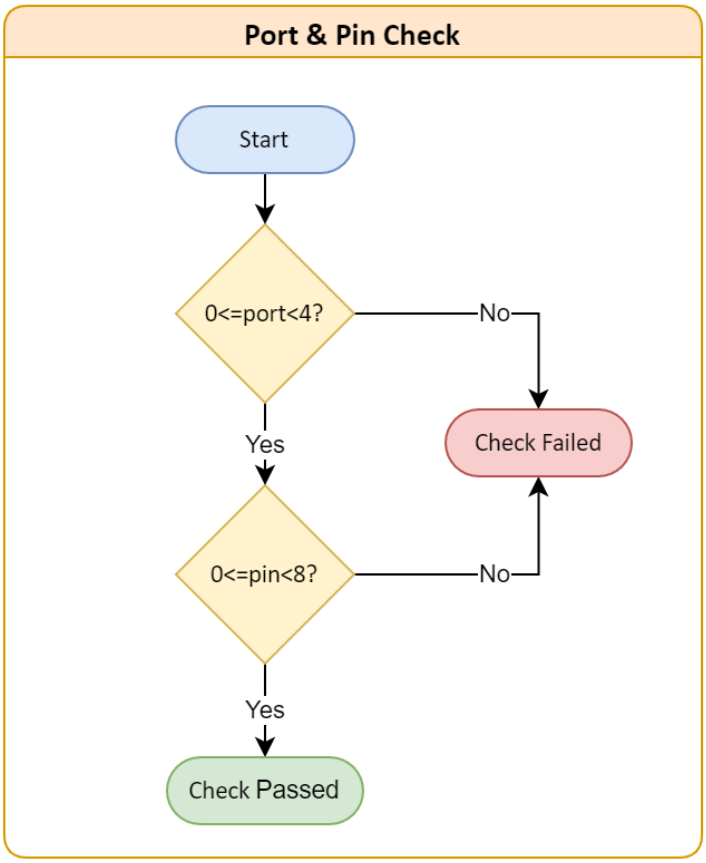


3. Low Level Design

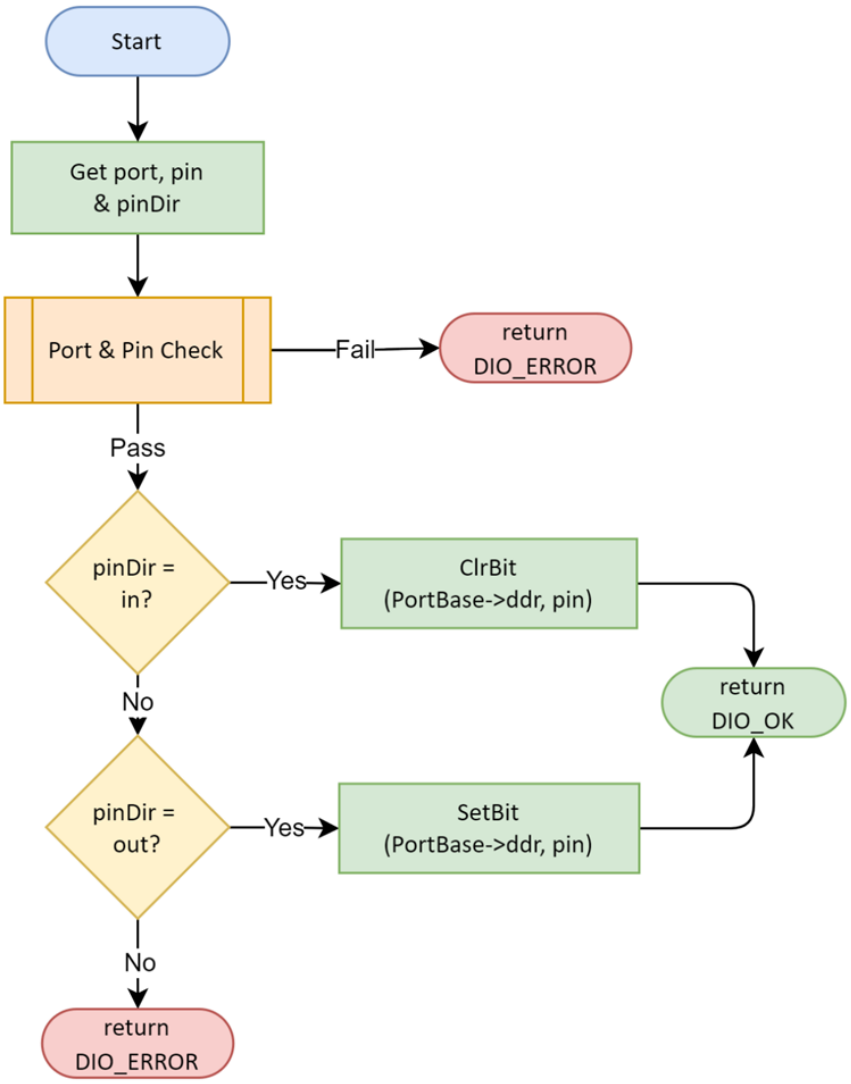
3.1. MCAL Layer

3.1.1. DIO Module

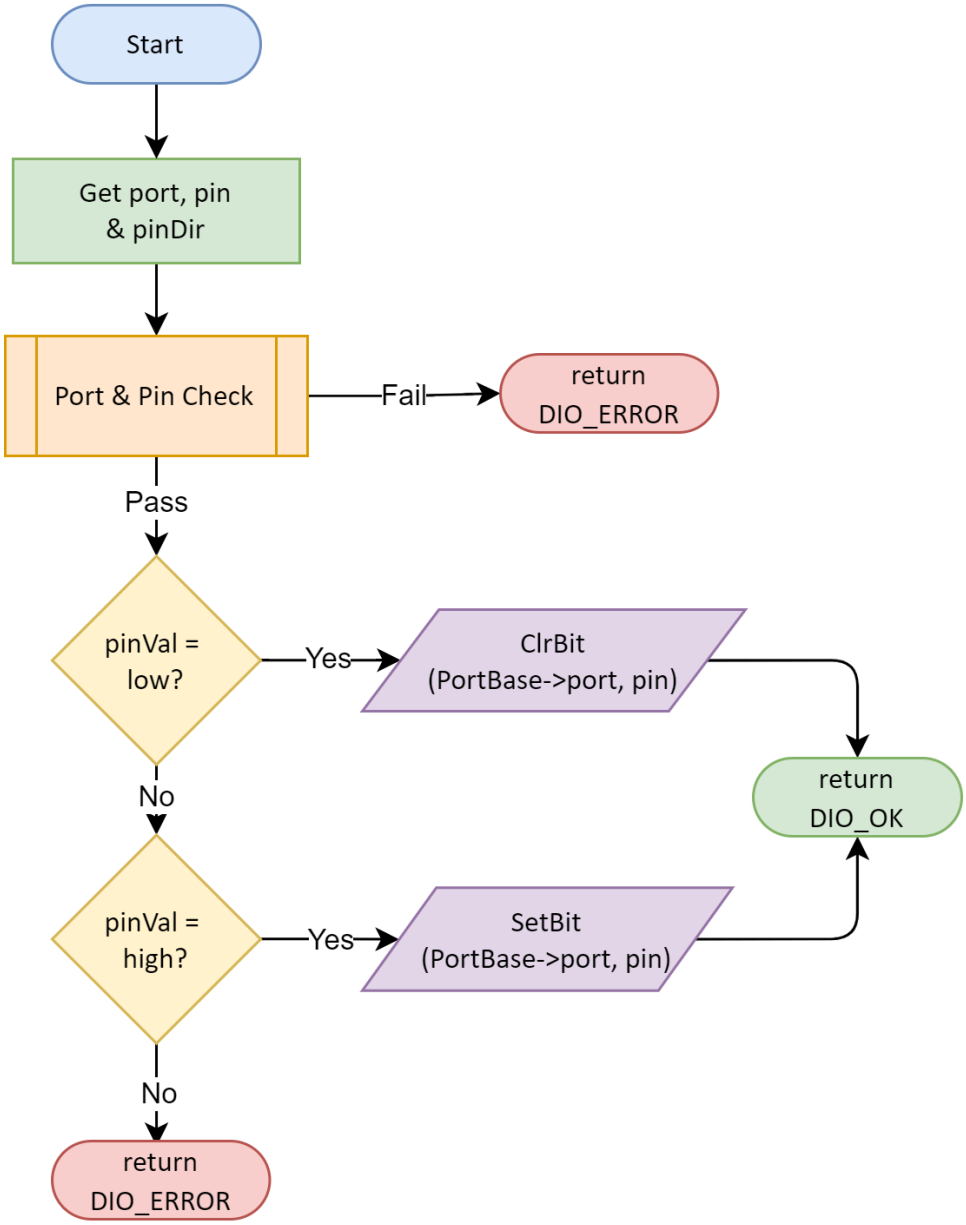
3.1.1.a. sub process



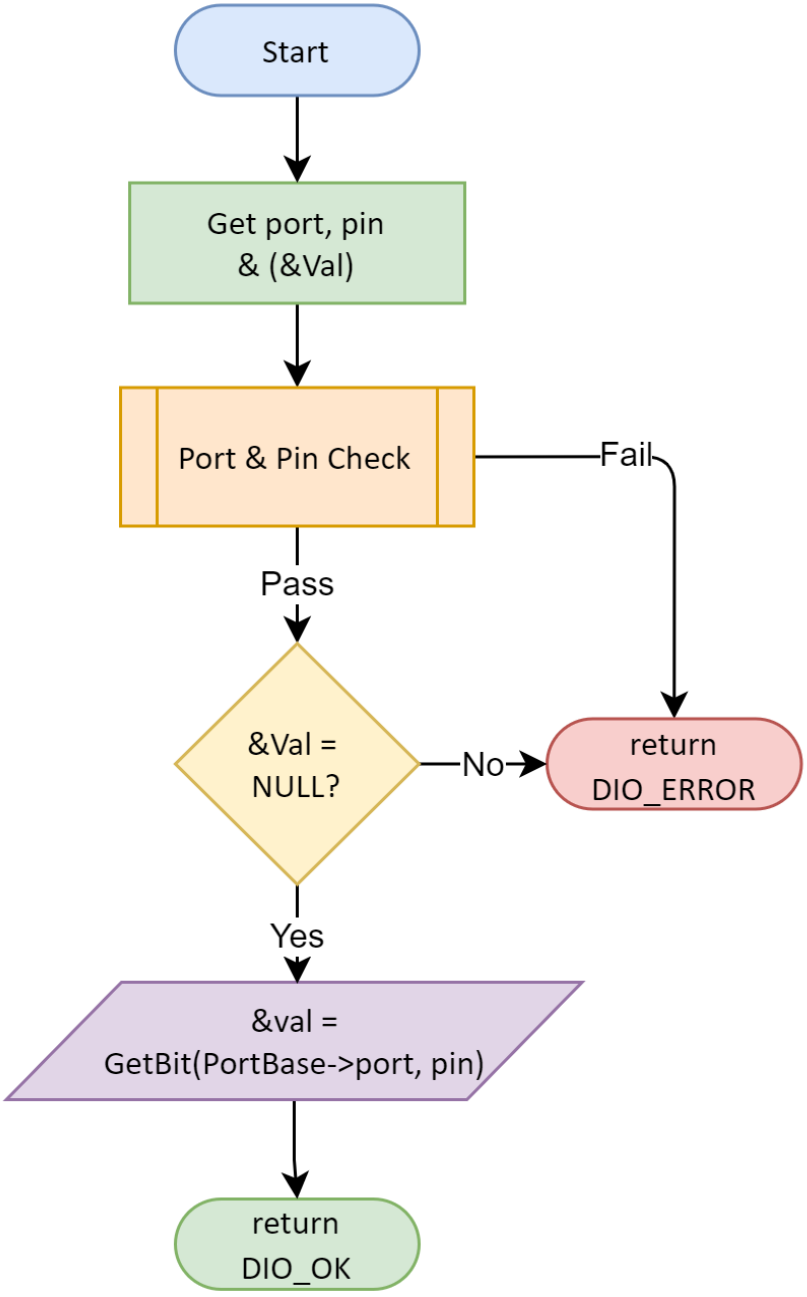
3.1.1.1. DIO_setPinDir



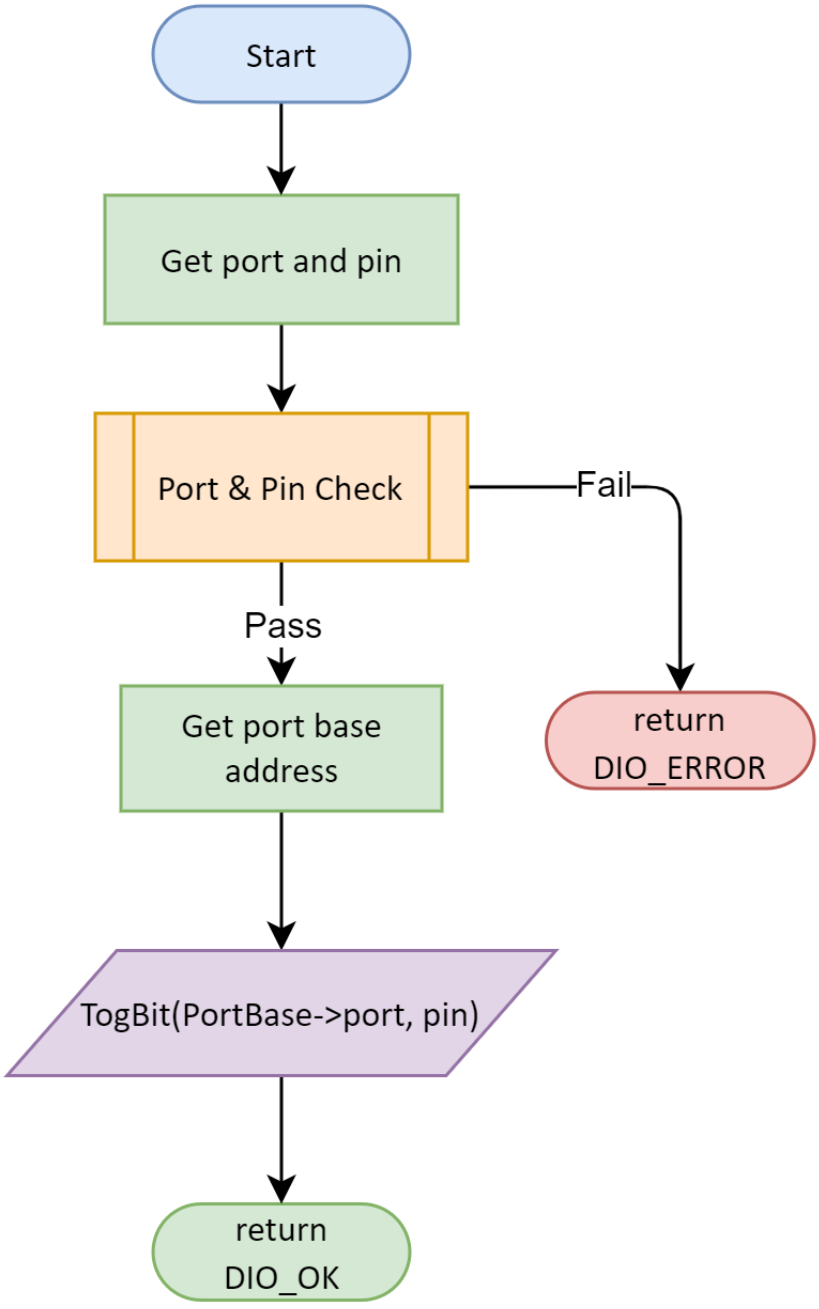
3.1.1.2. DIO_setPinVal



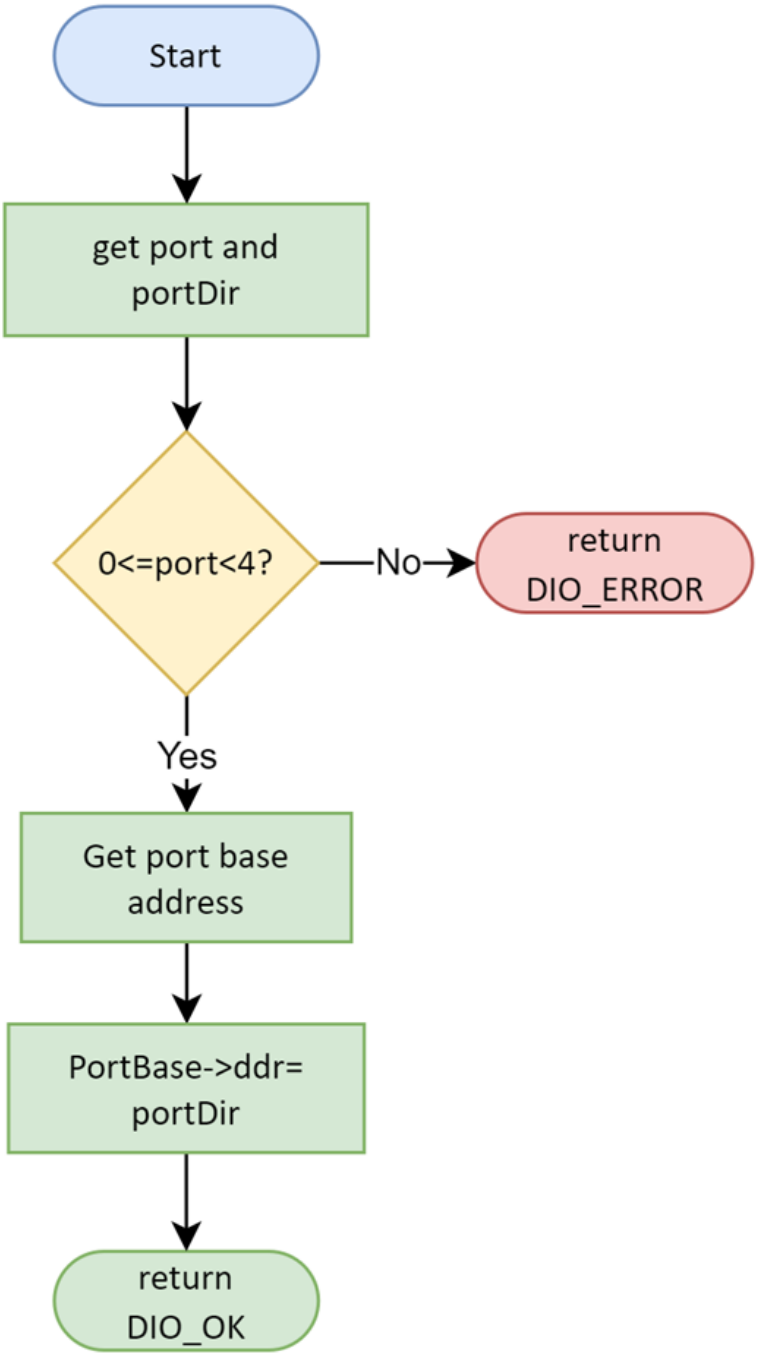
3.1.1.3. DIO_getPinVal



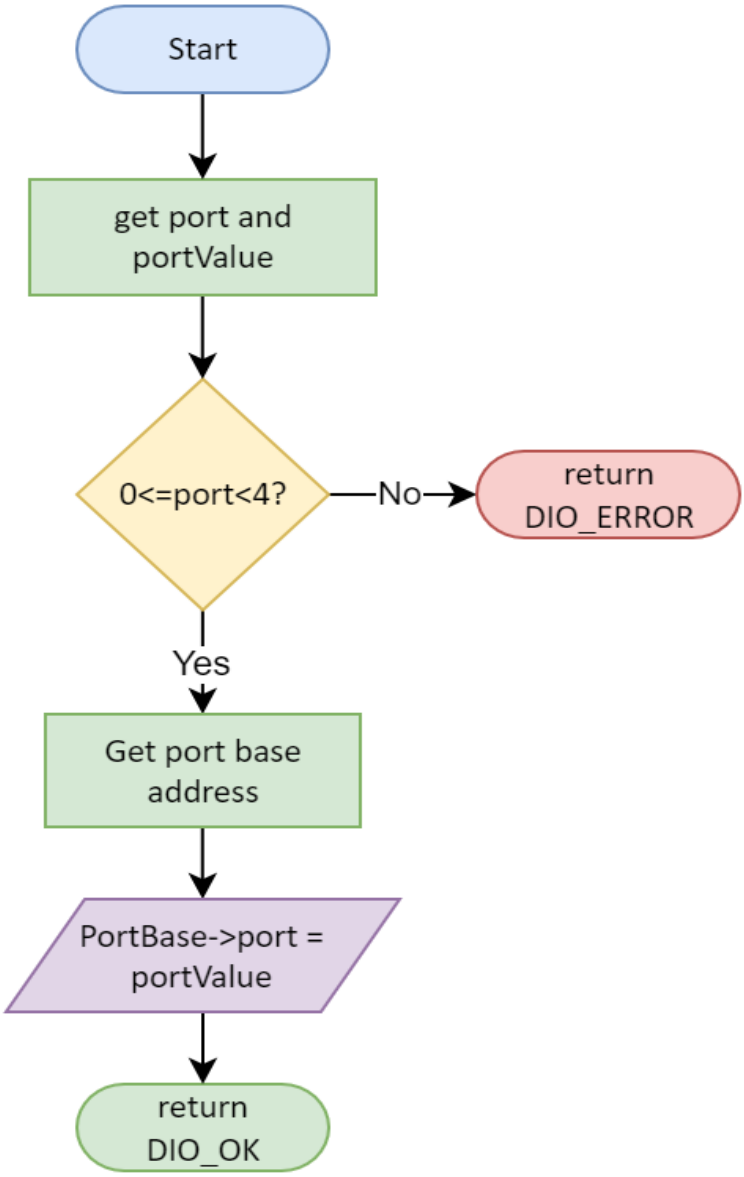
3.1.1.4. DIO_togPinVal



3.1.1.5. DIO_setPortDir

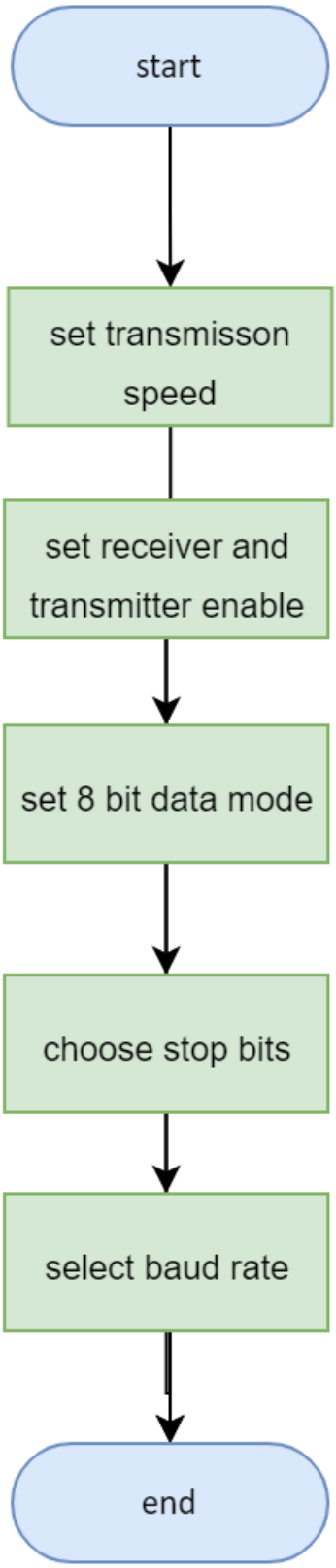


3.1.1.6. DIO_setPortVal

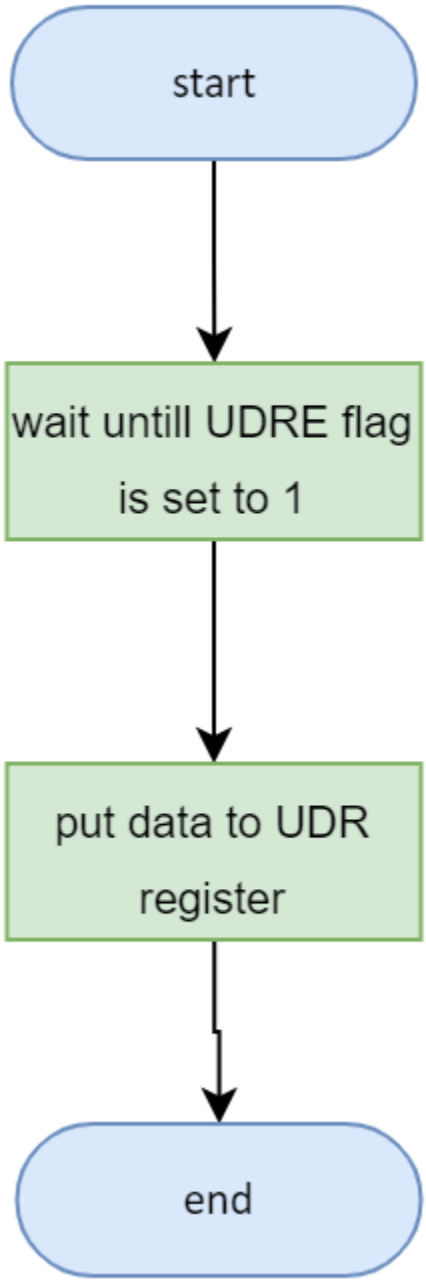

Start
get port and portValue
0<=port<4?
No
return DIO_ERROR
Yes
Get port base address
PortBase->port = portValue
return DIO_OK

3.1.2. UART Module

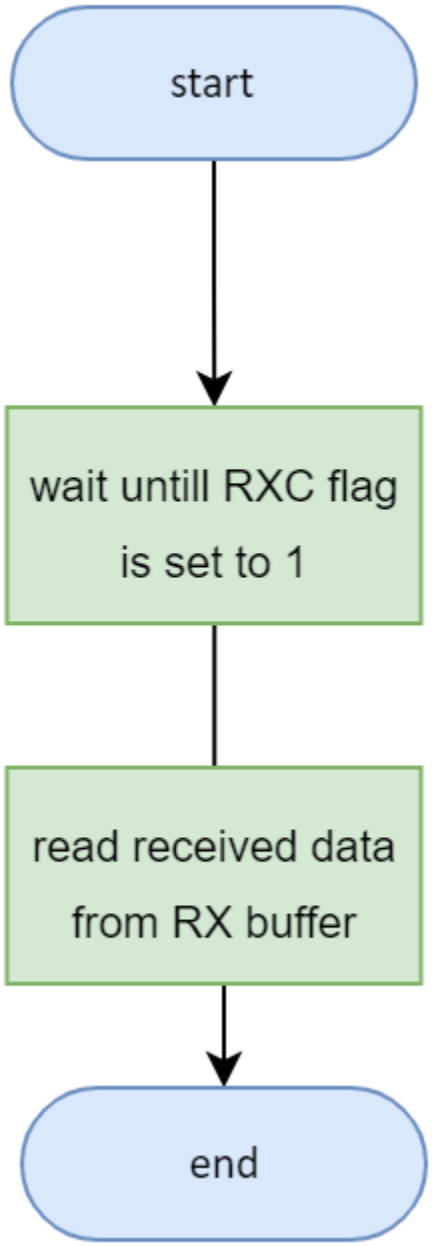
3.1.2.1. UART_init


start
set transmisson speed
set receiver and transmitter enable
set 8 bit data mode
choose stop bits
select baud rate
end

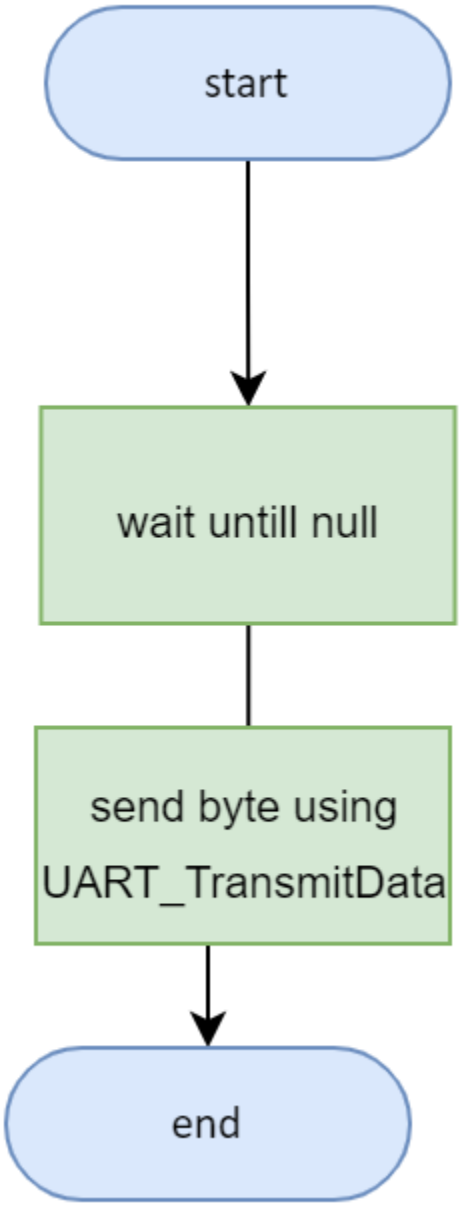
3.1.2.1. UART_TransmitData



3.1.2.1. UART_ReceiveData



3.1.2.1. UART_TransmitString



4. Pre-compiling and linking configurations

4.1. UART Driver

4.1.1 Pre-compiled Configurations

```
/*************** OPERATION MODES ***************/
#define UART_Baud_Rate              0x0033 // 51 decimal // F = 8GHZ,
NORMAL MODE (U2X=0), 9600 Baud Rate
#define UART_SPEED                        NORMAL_SPEED // Use normal if
synchronous
#define UART_OPERATION_MODE         UART_ASYNCHRONOUS
#define UART_PARITY_MODE            PARITY_DISABLE
#define UART_STOP_BITS              STOP_BITS_1
#define UART_CHARACTER_SIZE         CHARACTER_SIZE_8
#define USART_CLOCK_EDGE            UART_ASYNCHRONOUS // Synchronous mode
only
#define UART_INTERRUPT              UART_IE_DISABLE
```

```
/*************** CONDITIONS FOR OPERATION MODES ***************/
#define UART_ASYNCHRONOUS           0
#define UART_SYNCHRONOUS            1

#define PARITY_DISABLE              0
#define PARITY_ENABLE_EVEN          1
#define PARITY_ENABLE_ODD           2

#define STOP_BITS_1                       0
#define STOP_BITS_2                       1

#define CHARACTER_SIZE_5            0
#define CHARACTER_SIZE_6            1
#define CHARACTER_SIZE_7            2
#define CHARACTER_SIZE_8            3
#define CHARACTER_SIZE_9            4

#define CLOCK_RISING_EDGE           1
#define CLOCK_FALLING_EDGE          2

#define UART_IE_DISABLE             0
#define UART_IE_ENABLE              1

#define NORMAL_SPEED                0
#define DOUBLE_SPEED                1

#define TRANSMIT_DISABLE            0
```

```
#define TRANSMIT_ENABLE             1

#define RECEIVE_DISABLE             0
#define RECEIVE_ENABLE              1
```