Design document

project title: Obstacle Avoidance Robot V1.0 Design

Represented by: Hazem Ashraf

Table of contents

Project introduction

- **Description**

  Suppose you have a four-diving wheel robot, you are required to design the system so that the robot avoid any object in front.

- **System Requirement Specifications**

Car Components:

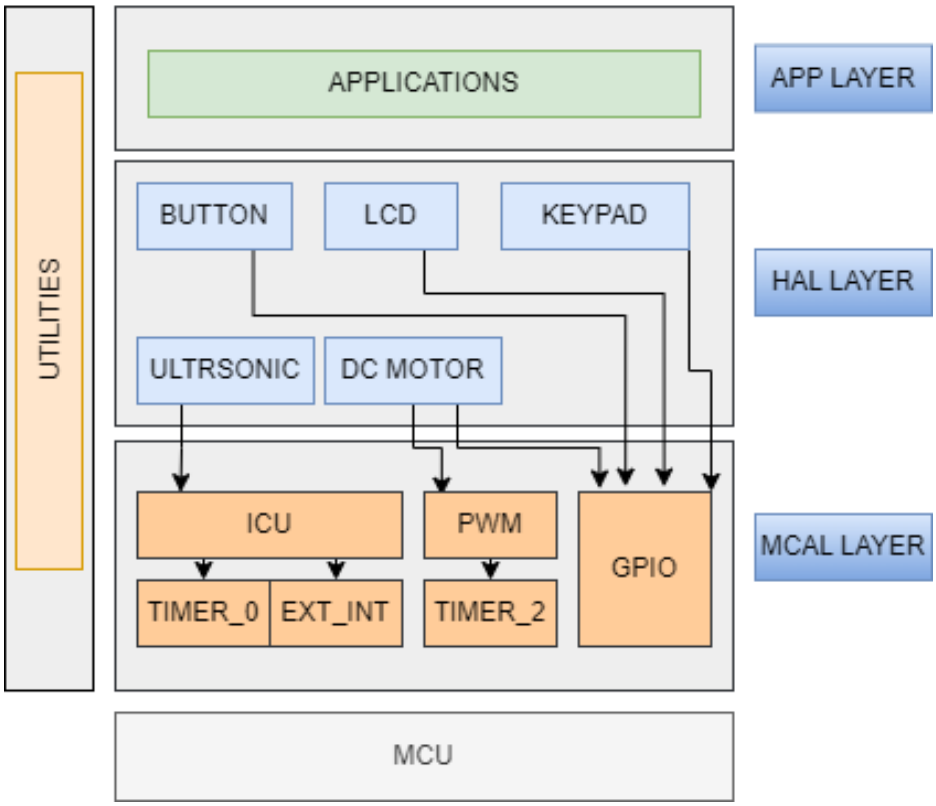
1. **ATmega32** microcontroller
2. **Four** motors (**M1**, **M2**, **M3**, **M4**)
3. **One** button to change default direction of rotation (**PBUTTON0**)
4. Keypad button 1 to start
5. Keypad button 2 to stop
6. **One Ultrasonic** sensor connected as follows
    1. **Vcc to 5V in the Board**
    2. **GND to the ground In the Board**
    3. **Trig to PB3 (Port B, Pin 3)**
    4. **Echo to PB2 (Port B, Pin 2)**
7. **LCD**

System Requirements:

1. The car **starts initially** from **0 speed**
2. The default rotation direction is to the **right**
3. Press (Keypad Btn 1), (Keypad Btn 2) to start or stop the robot respectively
4. **After Pressing Start**:
    1. The LCD will display a centered message in line 1 **"Set Def. Rot."**
    2. The LCD will display the selected option in line 2 **"Right"**
    3. The robot will **wait** for **5 seconds** to choose between **Right and Left**
        1. When **PBUTTON0** is pressed **once**, the default rotation will be **Left** and the **LCD line 2 will be updated**
        2. When **PBUTTON0** is pressed **again**, the default rotation will be **Right** and the **LCD line 2 will be updated**
        3. *For each press the default rotation will changed and the LCD line 2 is updated*
        4. **After the 5 seconds the default value of rotation is set**
    4. The robot will move **after 2 seconds** from setting the default direction of rotation.

5. **For No obstacles or object is far than 70 centimeters:**
    1. The robot will move forward with 30% speed for 5 seconds
    2. After 5 seconds it will move with 50% speed as long as there was no object or objects are located at more than 70 centimeters distance
    3. The LCD will display the speed and moving direction in line 1: **"Speed:00% Dir: F/B/R/S"**, **F**: forward, **B**: Backwards, **R**: Rotating, and **S**: Stopped
    4. The LCD will display Object distance in line 2 **"Dist.: 000 Cm"**
6. **For Obstacles located between 30 and 70 centimeters**
    1. The robot will decrease its speed to 30%
    2. LCD data is updated
7. **For Obstacles located between 20 and 30 centimeters**
    1. The robot will stop and rotates 90 degrees to right/left according to the chosen configuration
    2. The LCD data is updated
8. **For Obstacles located less than 20 centimeters**
    1. The robot will **stop**, move **backwards** with **30% speed** until distance is **greater than 20 and less than 30**
    2. The LCD data is updated
    3. **Then preform point 8**
9. **Obstacles surrounding the robot (Bonus)**
    1. If the robot **rotated for 360 degrees without finding any distance greater than 20 it will stop**
    2. LCD data will be updated.
    3. The robot will frequently (each 3 seconds) check if any of the obstacles was removed or not and move in the direction of the furthest object

- Layered architecture



- Modules Description

MCAL modules

GPIO

Using GPIO for initialize button ,keypad ,lcd and dc motor

PWM

Use PWM to generate PWM signal for motor speed control

ICU

Use ICU with ULTRASONIC sensor for calculating distance by calculate time period of the return signal

Timer_0

Use Timer with ICU to calculate time period of the signal

EXT_INT

Use EXT_INT with ICU to configure the edge detection of the signal

TIMER_2

Use Timer_2 to implement PWM using software implementation

- Modules Description

HAL modules

ULTRASONIC

Use ultrasonic sensor for get distance of around obstacles

BUTTON

Use push button for determine rotation direction

LCD

Use LCD to display states of the car speed , direction and distance

KEYPAD

Use Keypad button 1 and 2 for start and stop car

DC motor

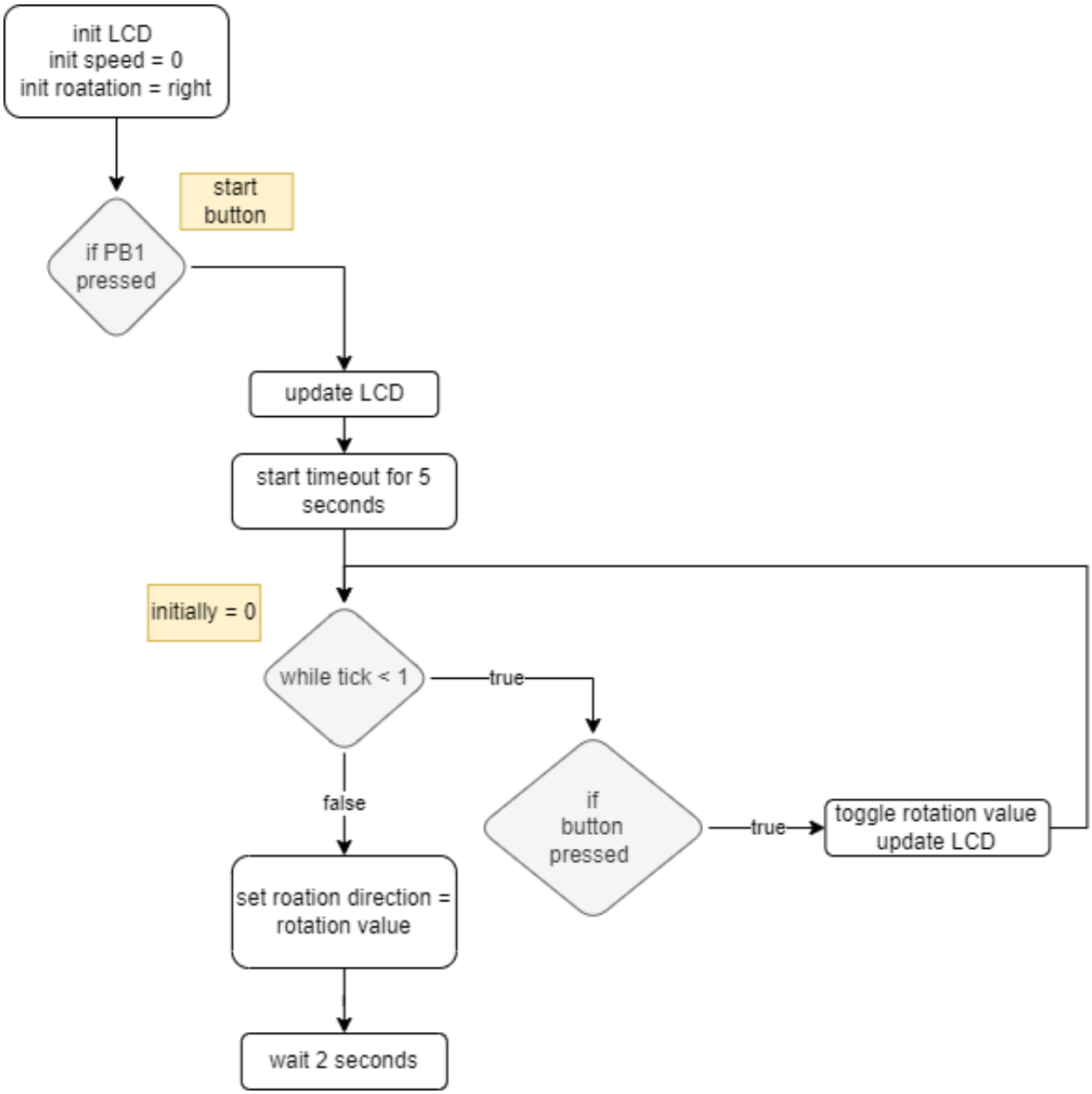
Use DC motor for control car movement direction and speed

APP Layer

Contain software algorithm for avoiding obstacles

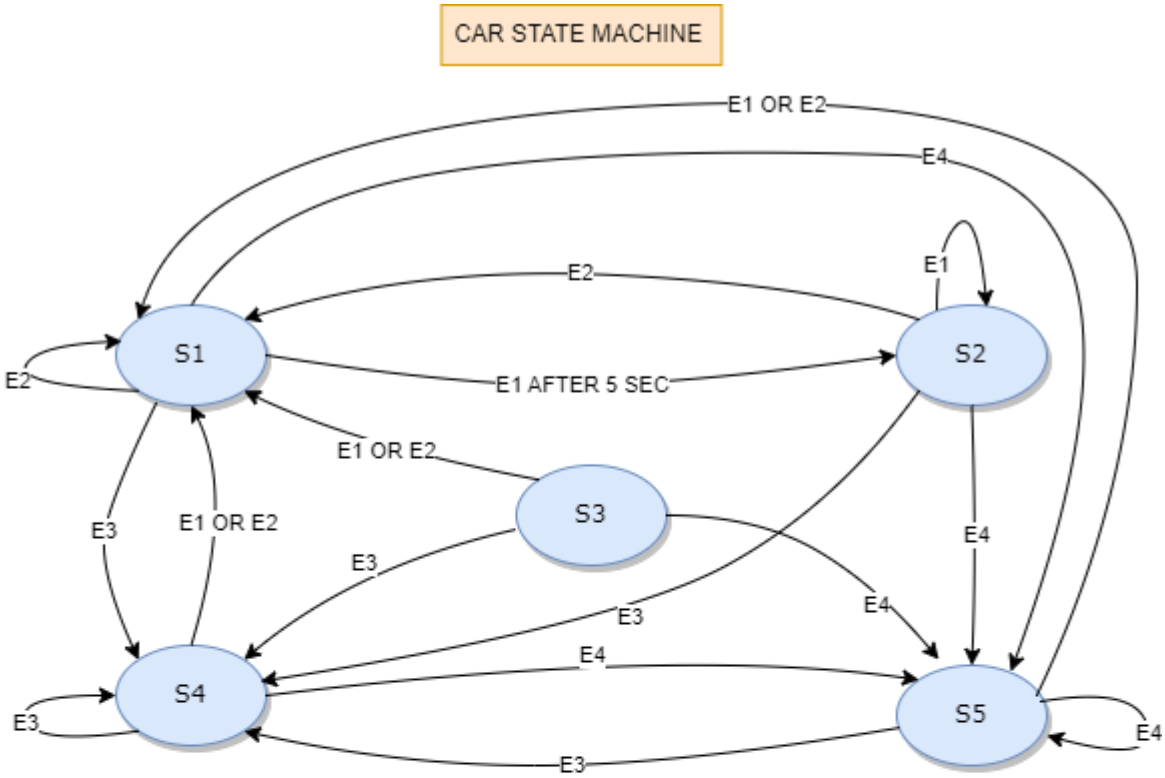
- Initialization workflow

We start application by initialize the external hardware and find the rotation direction to be left / right



- State machine diagram

By assuming the system is consist of states and events we can implement the moving car cases according to this stateMachine flow



STATE	
S1	CAR MOVE WITH 30%
S2	CAR MOVE WITH 50%
S3	STOP
S4	STOP & ROTATE 90*
S5	MOVE BACKWORD 30%

EVENTS	
E1	OBSTACLE AT DISTANCE > 70
E2	OBSTACLE AT 70 > D > 30
E3	OBSTACLE AT 30 > D > 20
E4	OBSTACLE AT 20 > D

Application design

- Application variables
  - Define enum for car state
    {S1,S2,S3,S4,S5,MAX_STATE}
  - Define enum for events
    {E1,E2,E3,E4}
  - Define enum for car directions
    { FW , BW , ROTATE , STOP }
  - Define enum for rotation direction
    { Right , Left }
  - System variables
    Uint8 g_currentstate
    Uint8 speed
    Enum direction
    Enum rotation
    Enum event
- Application APIs

En_event Readevent (void)
Use to read the distance and return the corresponding event

Void stateMachine(En_event)
Use to run state machine workflow and determine new state

Void Car_init (void)
Used to initialize rotation direction and initial speed

Void motor_control (motor ID1, motor ID2, motor ID3, motor ID4, direction , speed)
Used for control car movement

Void LCD_display (speed,direction,distance)
Used to display and update car informations on LCD

- **GPIO module APIs**

/*============= TYPE DEFINITION =============*/

```
typedef enum{
PIN_INPUT,PIN_OUTPUT
}EN_PIN_DIRECTION;

typedef enum
{
PORT_INPUT,PORT_OUTPUT=0xFF
}EN_PORT_DIRECTION;

typedef enum{
Low,High
}EN_PIN_VALUE;

typedef enum{
LOW,HIGH=0xFF
}EN_PORT_VALUE;

typedef enum{
FAILED,SUCCESS
}EN_STATE;
```

/*============= FUNCTION PROTOTYPE=============*/

EN_STATE GPIO_setPinDirection(uint8 port_num, uint8 pin_num, EN_PIN_DIRECTION direction);

Description:

Used to set specific pin direction as input or output pin

- **Port_num: determine port in which pin is connected, you have to use port_ID which defined as MACROS**
- **Pin_num: determine pin number , you have to use PIN which defined as MACROS**
- **Direction: used to determine direction of the pin, you have to use Enum EN_PIN_DIRECTION (PIN_INPUT,PIN_OUTPUT)**
- **Return : function check for the range of port_ID and PIN number Return SUCCESS if true and FAILED if out of range**

```
#define PORTA_ID     0
#define PORTB_ID     1
#define PORTC_ID     2
#define PORTD_ID     3
#define MAX_PORT_ID 4

#define PIN0     0
#define PIN1     1
#define PIN2     2
#define PIN3     3
#define PIN4     4
#define PIN5     5
#define PIN6     6
#define PIN7     7
#define MAX_PIN 8
```

EN_STATE GPIO_checkstate(uint8 port_num,uint8 pin_num);

Description:

Used to check for the range of port_ID and pin range

- **Port_num: determine port in which pin is connected, you have to use port_ID which defined as MACROS**
- **Pin_num: determine pin number , you have to use PIN which defined as MACROS**
- **Return : function check for the range of port_ID Return SUCCESS if true and FAILED if out of range**

GPIO module APIs

EN_STATE GPIO_writePin(uint8 port_num, uint8 pin_num, EN_PIN_VALUE value);

Description:

used to write high or low to specific pin

- **Port_num: determine port in which pin is connected, you have to use port_ID which defined as MACROS**
- **Pin_num: determine pin number , you have to use PIN which defined as MACROS**
- **Value: used to determine direction of the pin, you have to use Enum EN_PIN_VALUE (Low,High)**
- **Return : function check for the range of port_ID and PIN number Return SUCCESS if true and FAILED if out of range**

EN_STATE GPIO_readPin(uint8 port_num, uint8 pin_num,uint8* value);

Description:

used to read specific pin value

- **Port_num: determine port in which pin is connected, you have to use port_ID which defined as MACROS**
- **Pin_num: determine pin number , you have to use PIN which defined as MACROS**
- **Value: the address to variable of the return reading (High, Low)**
- **Return : function check for the range of port_ID and PIN number Return SUCCESS if true and FAILED if out of range**

EN_STATE GPIO_togglePin(uint8 port_num, uint8 pin_num);

Description:

used to toggle the output state of the pin

- **Port_num: determine port in which pin is connected, you have to use port_ID which defined as MACROS**
- **Pin_num: determine pin number , you have to use PIN which defined as MACROS**
- **Return : function check for the range of port_ID and PIN number Return SUCCESS if true and FAILED if out of range**

EN_STATE GPIO_setPortDirection(uint8 port_num, EN_PORT_DIRECTION direction);

Description:

used to determine port direction

- **Port_num: determine port ,you have to use port_ID which defined as MACROS**
- **Direction: used to determine direction of the pin, you have to use Enum EN_PORT_DIRECTION (PORT_INPUT,PORT_OUTPUT)**
- **Return : function check for the range of port_ID Return SUCCESS if true and FAILED if out of range**

- **GPIO module APIs**

EN_STATE GPIO_writePort(uint8 port_num, uint8 value);

Description:

used to write high/low to specific port

- **Port_num: determine port ,you have to use port_ID which defined as MACROS**
- **Value: used to determine direction of the port, you have to use Enum EN_PORT_VALUE [LOW,HIGH]**
- **Return : function check for the range of port_ID Return SUCCESS if true and FAILED if out of range**

EN_STATE GPIO_readPort(uint8 port_num,uint8* value);

Description:

used to read the value of specific port

- **Port_num: determine port ,you have to use port_ID which defined as MACROS**
- **Value: the address to variable of the return reading (High, Low)**
- **Return : function check for the range of port_ID Return SUCCESS if true and FAILED if out of range**

Timer0_delay module APIs (TIMER_0.h)

/*============= TYPE DEFINITION =============*/
typedef struct{
  float delay;
  uint16 prescaler;
  uint8 init_value;
  float NO_OF_OV;
}ST_timer0_config;
Description:
the structure is used to implement delay object, to define delay variable:

Timer0_delay module APIs (TIMER_0.h)

/*============= FUNCTION PROTOTYPE =============*/
void Timer0_Delay(float delay);
Description:
- used to apply delay using polling technique
- it convert number of overflows to integer number to implement the required delay correctly
- example: if number of overflows=3.8
- mean perform 3 overflows and calculate the remaining time to complete the delay

void Timer0_event(uint16 delay,void(*g_ptr)(void));
Description:
- used to apply time out delay and run function if a period of time has passed
- Delay: delay time
- g_ptr: pointer to function which is called when time has passed

- **EXT-interrupt module APIs**

/*============= TYPE DEFINITION =============*/

typedef enum{
EN_INT0,EN_INT1,EN_INT2
}EN_INT_source;

typedef enum{
LOW_LEVEL,ANY_CHANGE,FALLING,RISING
}EN_INT_TRIGGER;

typedef enum{
INT_FAILED,INT_SUCCESS
}EN_INT_error;

typedef struct{
EN_INT_source source;
EN_INT_TRIGGER trigger;
}ST_INT_Config;

#define INT0_pin 2 //PD2
#define INT1_pin 3 //PD3
#define INT2_pin 3 //PB2

/*============= FUNCTION PROTOTYPE =============*/

EN_INT_error **INT_init(ST_INT_Config* Int_config)**

Description

- INT_init: used to initialize the interrupt by:
- disable global interrupt
- enable external interrupt source and set pin to input
- set external interrupt trigger signal type
- enable global interrupt
- Function parameters
- Int_config: pointer to structure of ST_INT_Config
- Return : FAILED if passing parameters is not correct, SUCCESS if the passing parameters is correct

void INT0_setCallBack(void(*a_ptr)(void));

Description:

INT0_setCallBack:used to set call back function for external INT_0

void INT1_setCallBack(void(*a_ptr)(void));

Description:

INT1_setCallBack:used to set call back function for external INT_1

void INT2_setCallBack(void(*a_ptr)(void));

Description:

INT2_setCallBack:used to set call back function for external INT_2

void INT_Deinit(ST_INT_Config* Int_config);

Description:

- INT_init: used to initialize the interrupt by:
- Disable specific external interrupt source

- **ICU module APIs**

/*============= TYPE DEFINITION =============*/

```
typedef enum
{
NO_CLOCK,F_CPU_CLOCK,F_CPU_8,F_CPU_64,F_CPU_256,F_CPU_1024
}Icu_Clock;

typedef enum
{
FALLING,RISING
}Icu_EdgeType;

typedef struct
{
Icu_Clock clock;
Icu_EdgeType edge;
}Icu_ConfigType;
```

/*============= FUNCTION PROTOTYPE =============*/

void Icu_init(Icu_ConfigType * Config_Ptr);

Description : Function to initialize the ICU driver

1. Set the required clock.
2. Set the required edge detection.
3. Enable the External Interrupt.
4. Initialize Timer1 Registers

void Icu_setCallBack(void(*a_ptr)(void));

Description: Function to set the Call Back function address.

void Icu_setEdgeDetectionType(Icu_EdgeType edgeType);

Description: Function to set the required edge detection.

uint16 Icu_getInputCaptureValue(void);

Description: Function to get the Timer1 Value when the external interrupt catch edge The value stored at timer register saved to a variable

void Icu_clearTimerValue(void);

Description: Function to clear the Timer1 Value to start count from ZERO

void Icu_DeInit(void);

Description: Function to disable the Timer1 and External Interrupt to stop the ICU Driver

- **Keypad driver APIs**

This is a keypad static configuration, before using driver you have to configure the macros according to keypad specifications and hardware connections

```
/*====Keypad configurations for number of rows and columns==== */
#define KEYPAD_NUM_COLS                 3
#define KEYPAD_NUM_ROWS                 3

/*============= Keypad Port Configurations =============*/
#define KEYPAD_PORT_ID                      PORTB_ID
#define KEYPAD_FIRST_ROW_PIN_ID             PIN3
#define KEYPAD_FIRST_COLUMN_PIN_ID          PIN0

/*=========== Keypad button logic configurations ========== */
#define KEYPAD_BUTTON_PRESSED           LOGIC_LOW

/*============= FUNCTION PROTOTYPE=============*/
uint8 KEYPAD_getPressedKey(void);
```

Description:

Used to get the pressed key from the keypad

```
static uint8 KEYPAD_3x3_adjustKeyNumber(uint8 button_number);
```

Description:

Before code compilation, modify the function to determine keypad map for each key location

```
static uint8 KEYPAD_3x3_adjustKeyNumber(uint8 button_number)
{
    uint8 keypad_button = 0;
    switch(button_number)
    {
        case 1: keypad_button = 1;
                break;
        case 2: keypad_button = 2;
                break;
        case 3: keypad_button = 3;
                break;
        case 4: keypad_button = 4;
                break;
        case 5: keypad_button = 5;
                break;
        case 6: keypad_button = 6;
                break;
        case 7: keypad_button = 7;
                break;
        case 8: keypad_button = 8;
                break;
        case 9: keypad_button = 9;
                break;
        default: keypad_button = button_number;
                break;
    }
    return keypad_button;
```

LCD driver APIs

/*============= MACRO DEFINITION =============*/

/* LCD HW Ports and Pins Ids */

```
#define LCD_RS_PORT_ID                  PORTD_ID
#define LCD_RS_PIN_ID                   PIN4_ID

#define LCD_RW_PORT_ID                  PORTD_ID
#define LCD_RW_PIN_ID                   PIN5_ID

#define LCD_E_PORT_ID                   PORTD_ID
#define LCD_E_PIN_ID                    PIN6_ID

#define LCD_DATA_PORT_ID                PORTC_ID
```

/* LCD Commands */

```
#define LCD_CLEAR_COMMAND                       0x01
#define LCD_GO_TO_HOME                          0x02
#define LCD_TWO_LINES_EIGHT_BITS_MODE           0x38
#define LCD_TWO_LINES_FOUR_BITS_MODE            0x28
#define LCD_CURSOR_OFF                          0x0C
#define LCD_CURSOR_ON                           0x0E
#define LCD_SET_CURSOR_LOCATION                 0x80
```

/*============= FUNCTION PROTOTYPE=============*/

void LCD_init(void);

Description :Initialize the LCD:

- Setup the LCD pins directions by use the GPIO driver.
- Setup the LCD Data Mode 4-bits or 8-bits.

void LCD_sendCommand(uint8 command);

Description :Send the required command to the screen

void LCD_displayCharacter(uint8 data);

Description :Display the required character on the screen

void LCD_displayString(const char *Str);

Description :Display the required string on the screen

void LCD_moveCursor(uint8 row,uint8 col);

Description :Move the cursor to a specified row and column index on the screen

void LCD_displayStringRowColumn(uint8 row,uint8 col,const char *Str);

Description :Display the required string in a specified row and column index on the screen

void LCD_intgerToString(int data);

Description :Display the required decimal value on the screen

- **Ultrasonic driver APIs**

void Ultrasonic_init(void);
Description : Function to initialize the ultrasonic driver
1-initialize ICU driver
2-set callback function
3-setup trigger pin direction as output

void Ultrasonic_Trigger(void)
Description : function to send trigger signal

void Ultrasonic_edgeProcessing(void)
Description : function to count number of edges and to calculate time of generated pulse

uint16 Ultrasonic_readDistance(void);
Description : function to read distance of from the sensor

- **BUTTON module APIs**

/*============= TYPE DEFINITION =============*/
typedef enum{
disable,enable
}EN_internal_pullup;
/*============= FUNCTION PROTOTYPE =============*/
EN_STATE Button_init(uint8 pin,EN_internal_pullup state);

Description

- Button_init: used to initialize BUTTON direction and set internal pullup resistor
- function parameters
- pin: pin number to be set
- State: to disable/enable internal pullup resistor
- Return success pin number is in the range, FAILED if pin number out of the range

uint8 Button_Read(uint8 pin);

Description

- Button_Read: used to read button state high/low
- function parameters
- pin: pin number to read
- Return button state high / low

- **DC motor module APIs**

/*============= TYPE DEFINITION =============*/
typedef struct{
uint8 port_ID;
uint8 PIN_ID;
}ST_PIN;

typedef struct{
ST_PIN pin1;
ST_PIN pin2;
}ST_DCmotor_configtype;

typedef enum{
CW,ACW,stop
}EN_DCMotor_state;

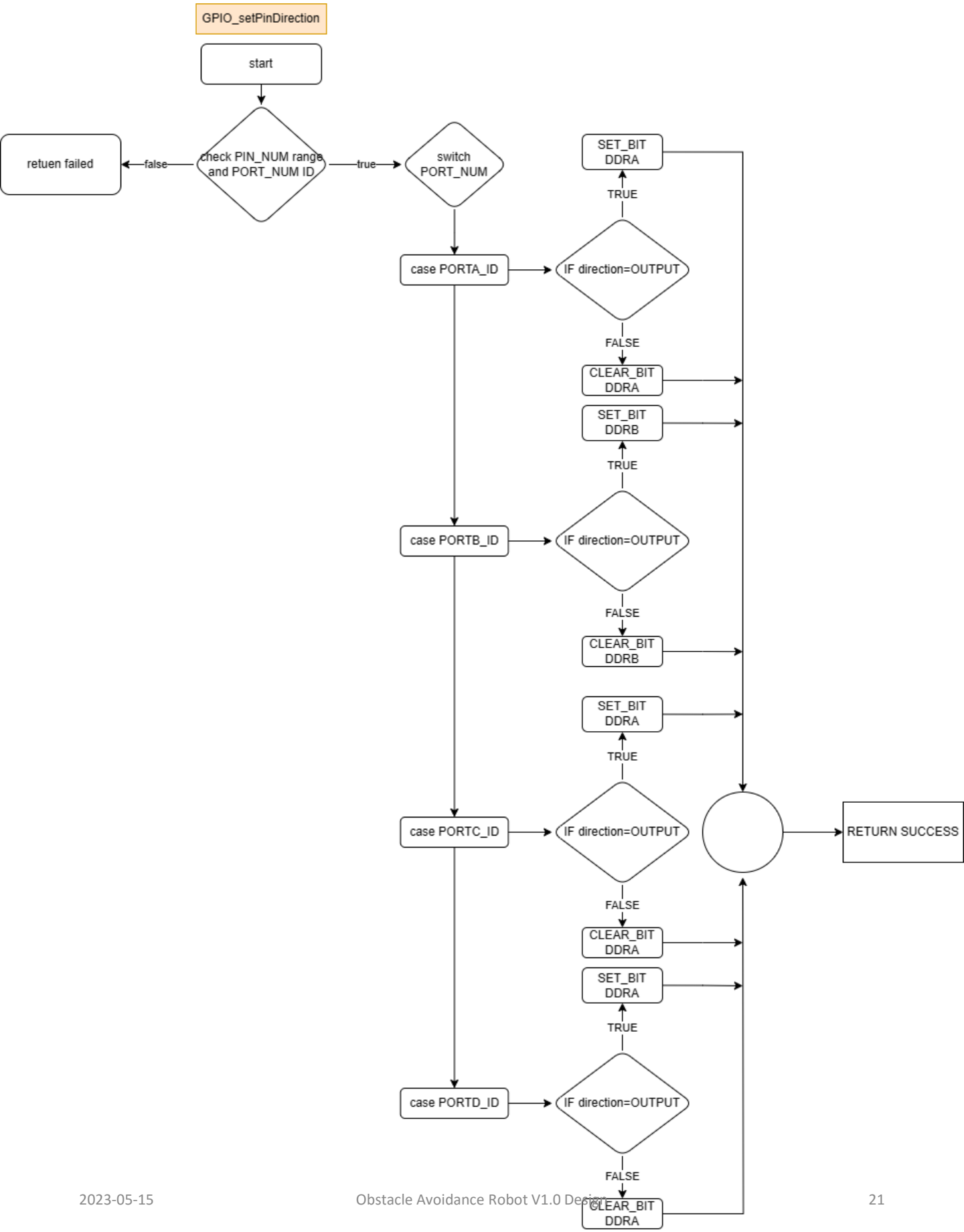
/*============= FUNCTION PROTOTYPE =============*/
EN_STATE **DCmotor_init(ST_DCmotor_configtype *motor);**
Description:
Used to initialize motor direction and set to low as initial value
Enable PWM module
Return success pin number is in the range, FAILED if pin number out of the range

void DCmotor_start(ST_DCmotor_configtype *motor,EN_DCMotor_state State,uint8 speed);
Description:
Used to control motor direction and speed using PWM

void DCmotor_stop(ST_DCmotor_configtype *motor);
Description:
Used to stop the motor by disable PWM module and stop motor rotating

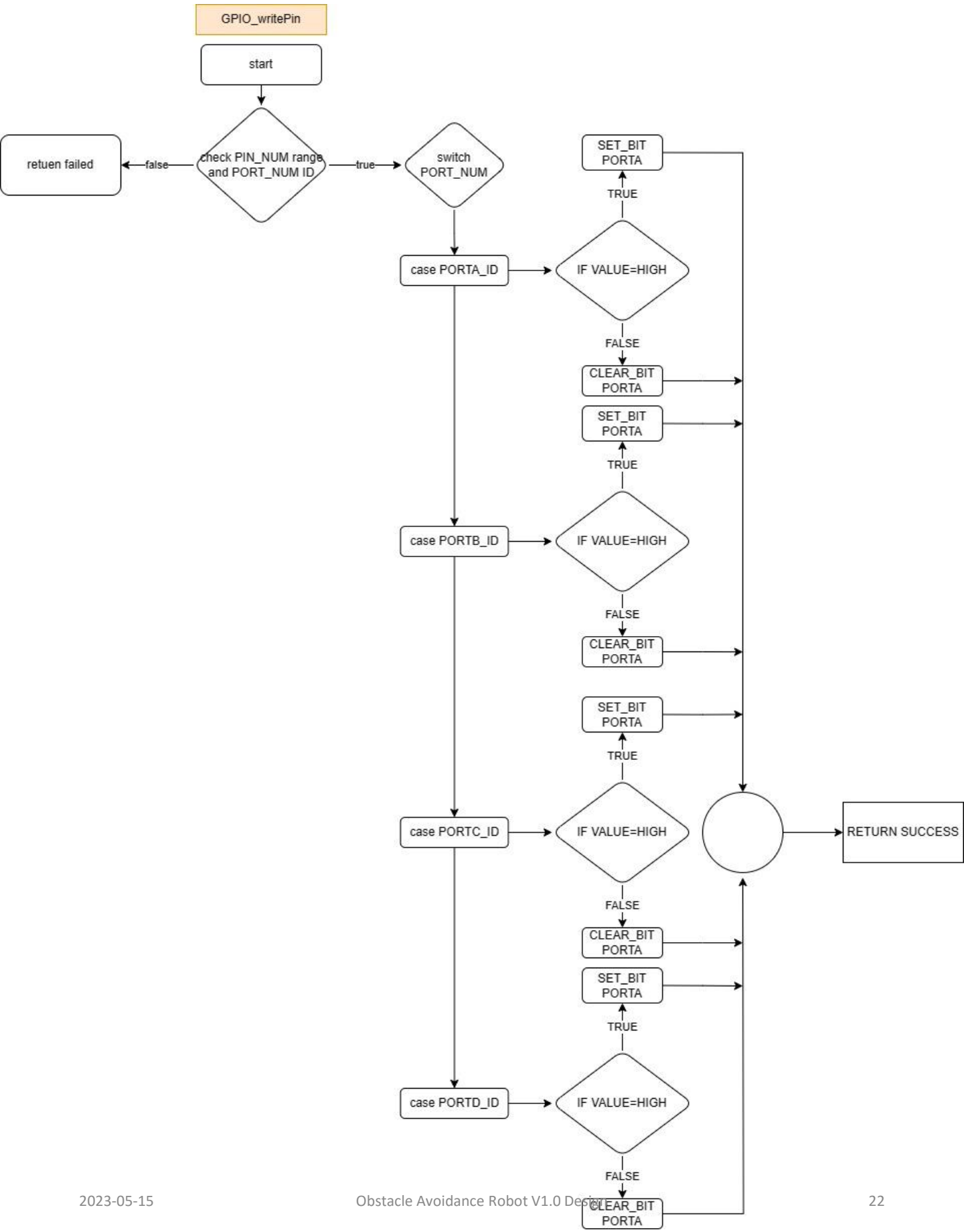
GPIO APIs flowchart

EN_STATE GPIO_setPinDirection(uint8 port_num, uint8 pin_num, EN_PIN_DIRECTION direction);



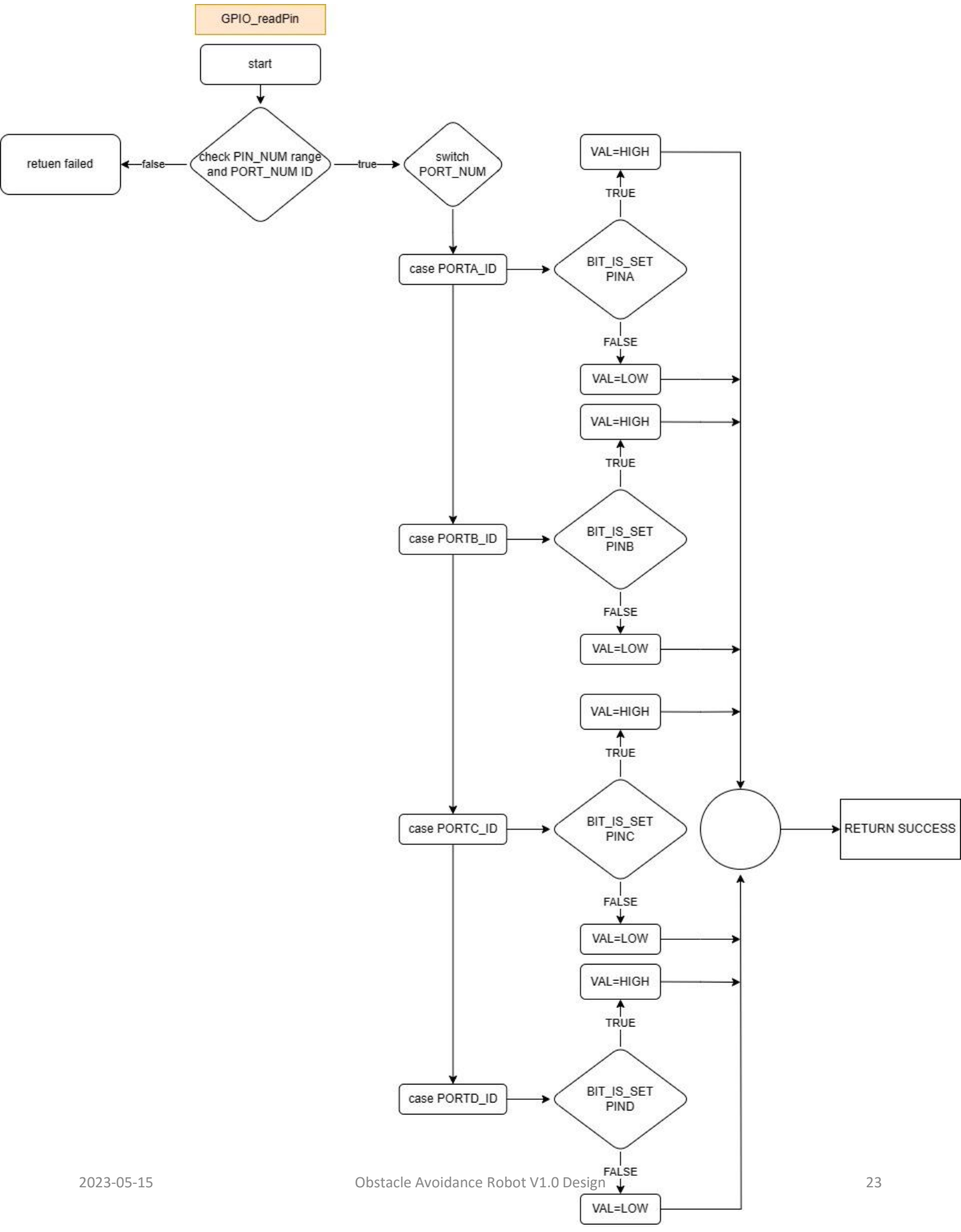
GPIO APIs flowchart

EN_STATE GPIO_writePin(uint8 port_num, uint8 pin_num, EN_PIN_VALUE value);

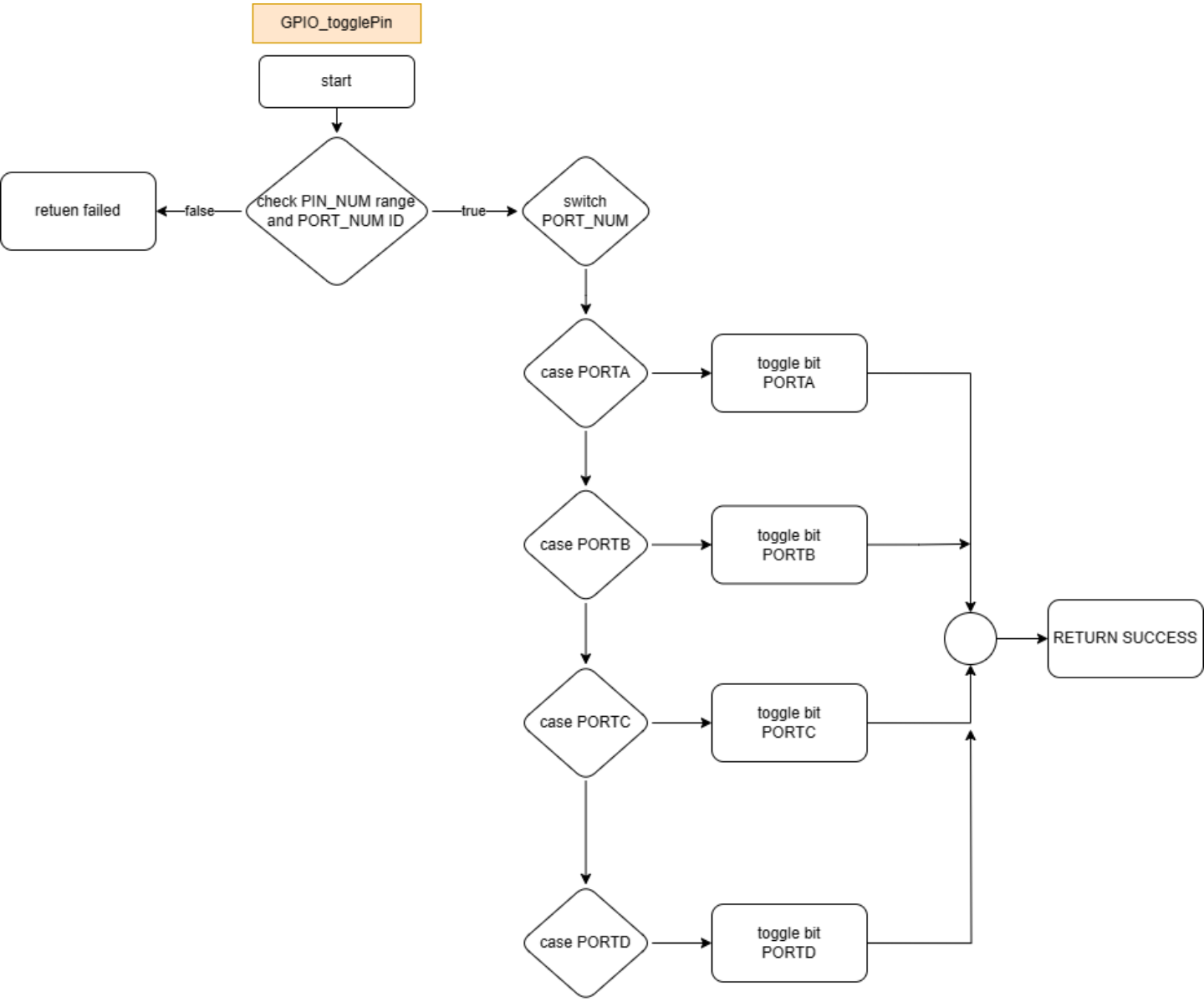


GPIO APIs flowchart

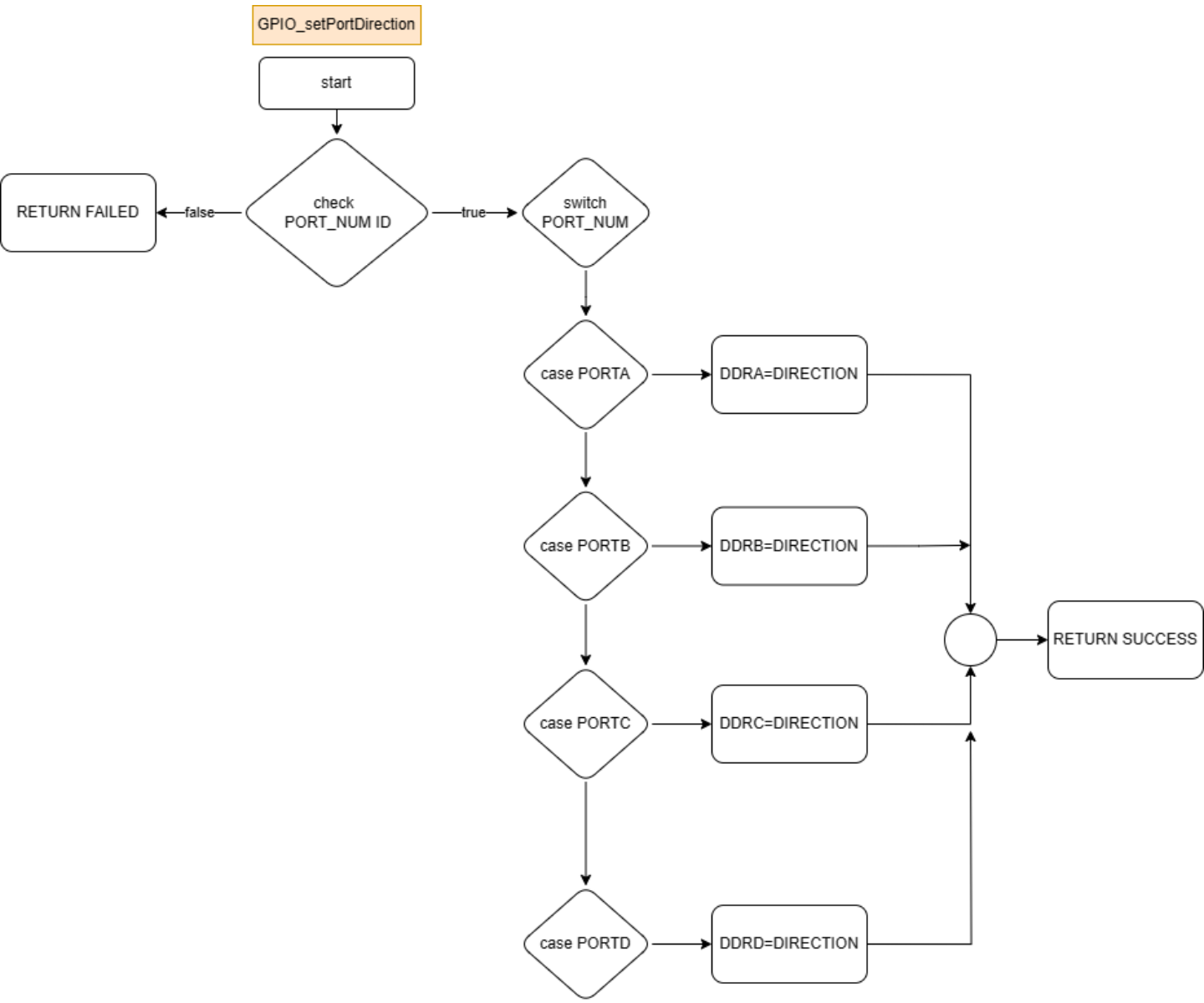
EN_STATE GPIO_readPin(uint8 port_num, uint8 pin_num,uint8* value);



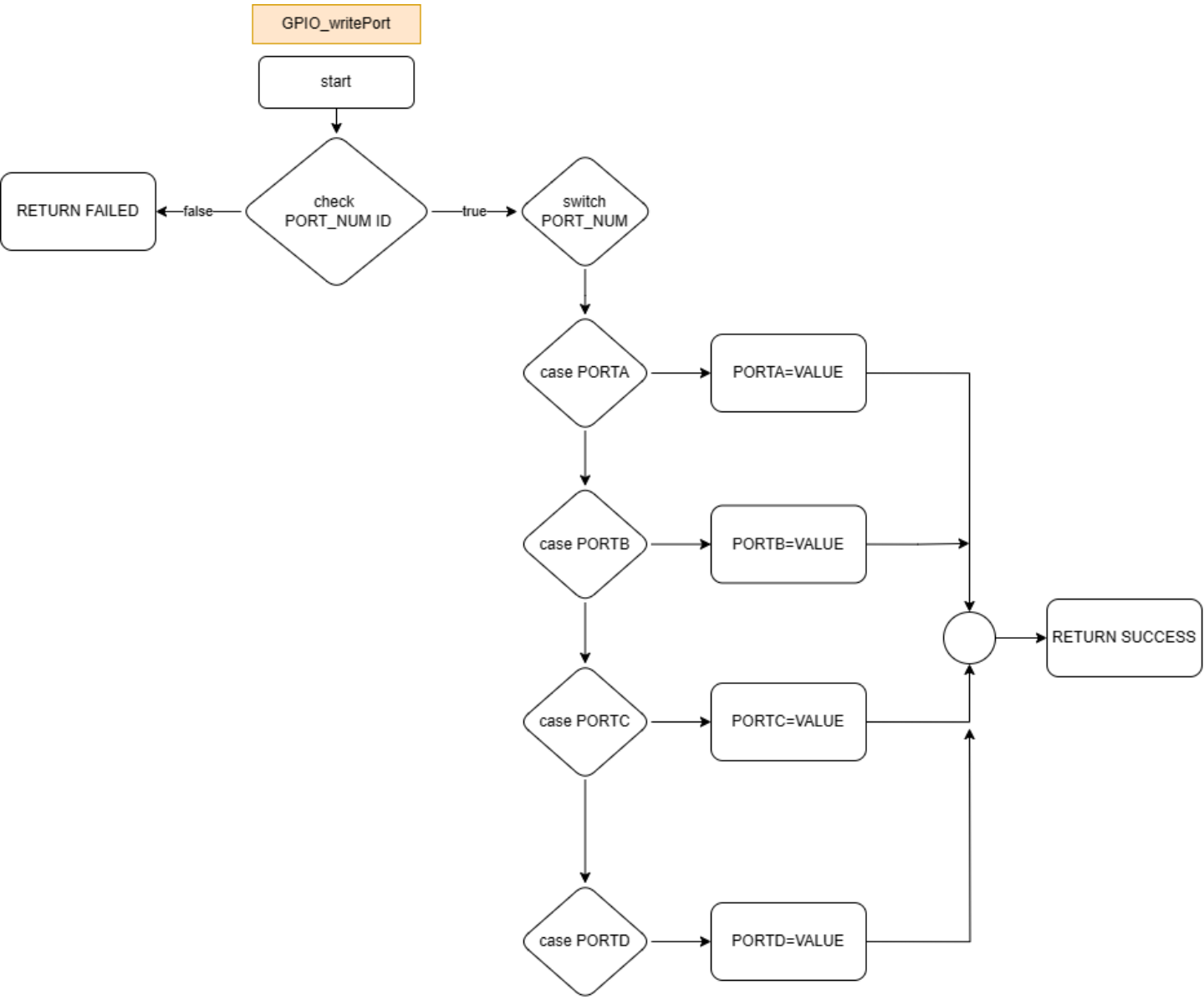
EN_STATE GPIO_togglePin(uint8 port_num, uint8 pin_num);



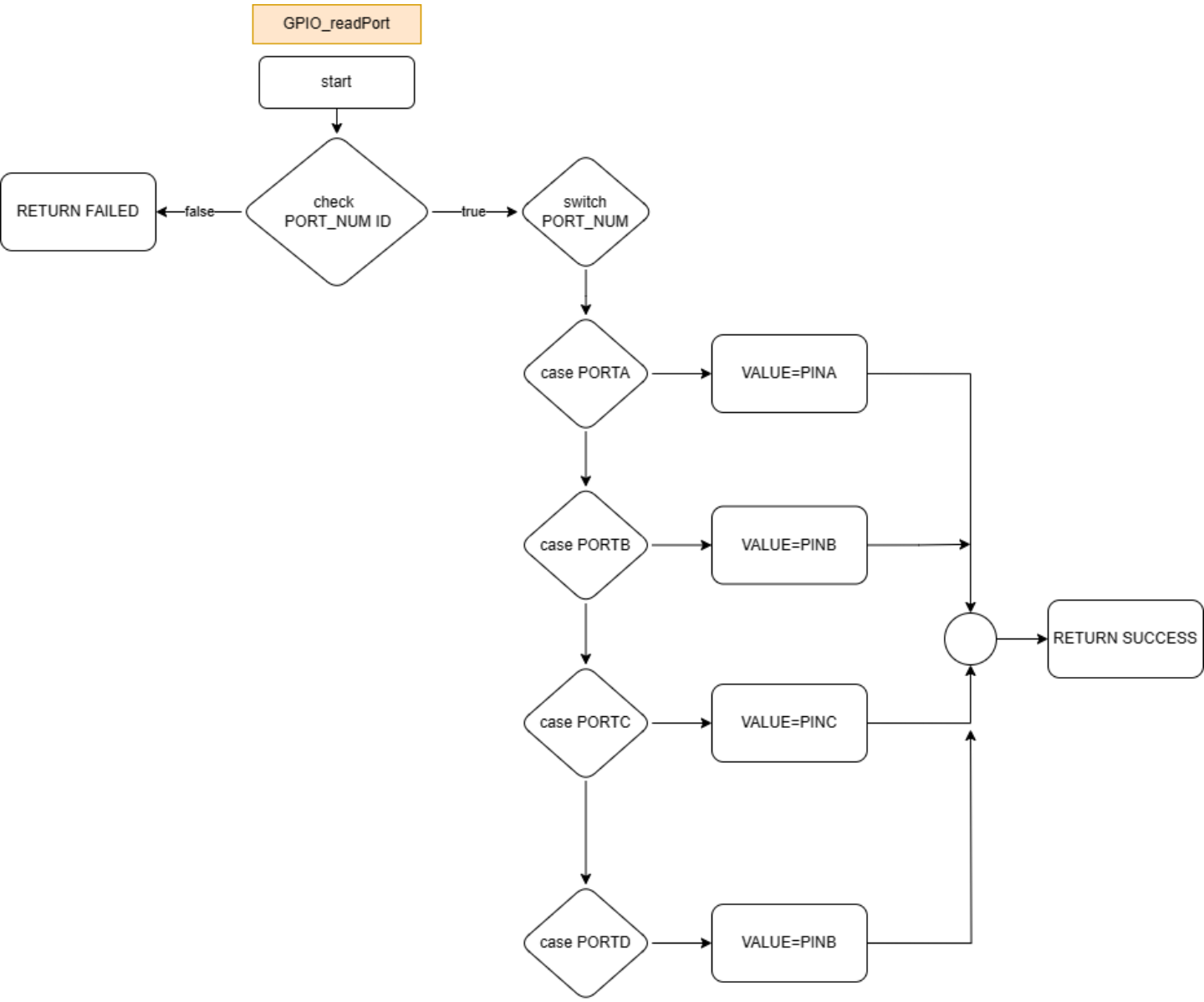
EN_STATE GPIO_setPortDirection(uint8 port_num, EN_PORT_DIRECTION direction);



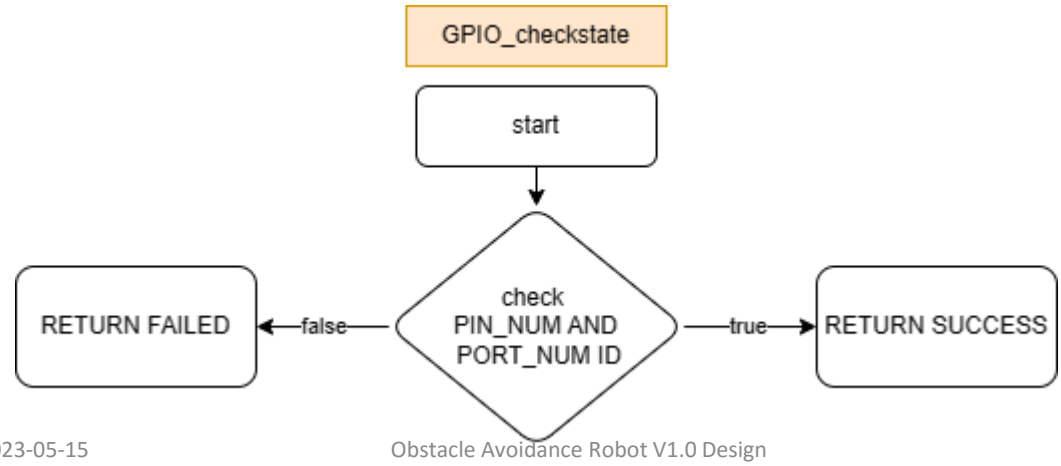
EN_STATE GPIO_writePort(uint8 port_num, uint8 value);



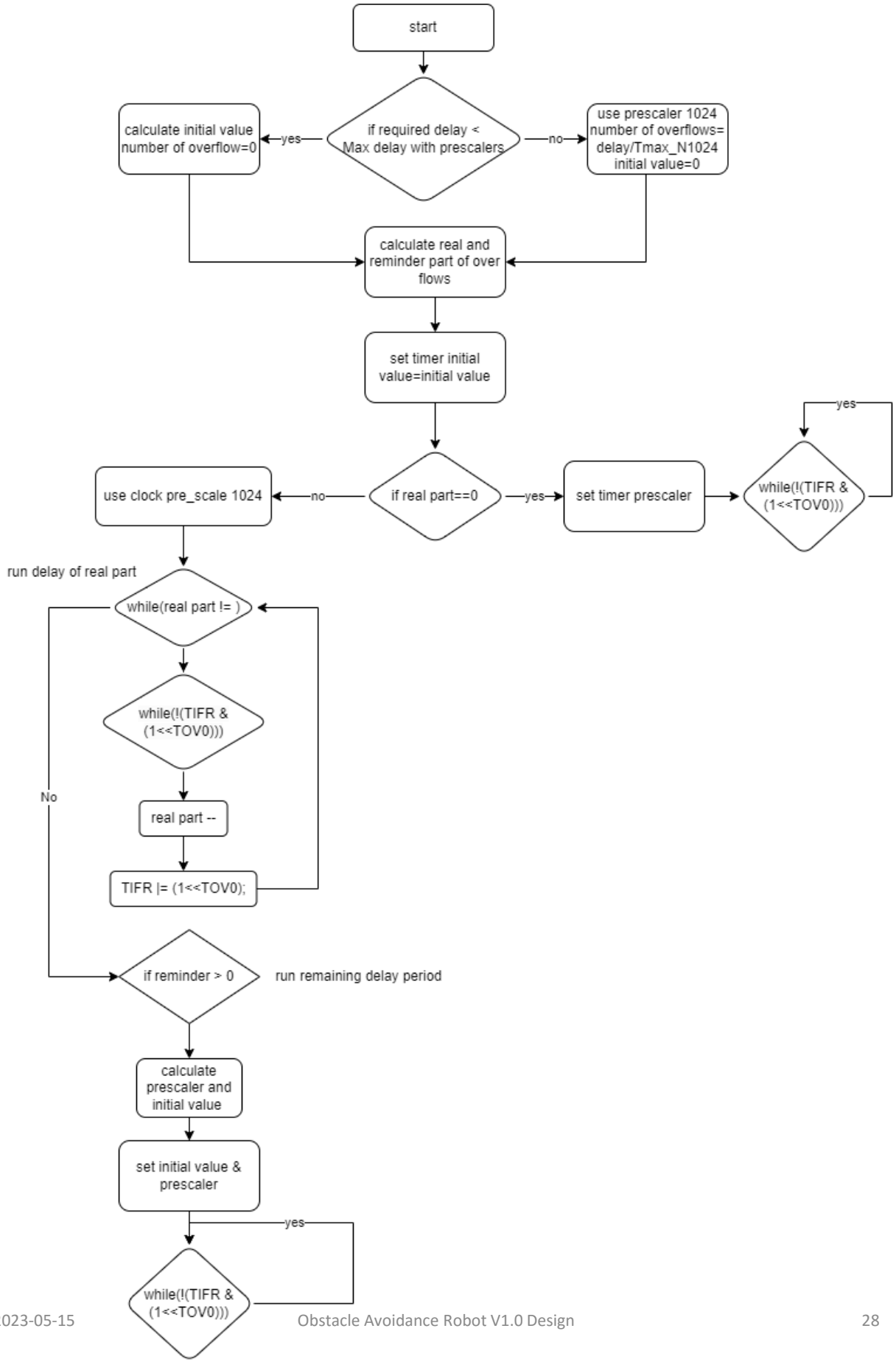
EN_STATE GPIO_readPort(uint8 port_num,uint8* value);



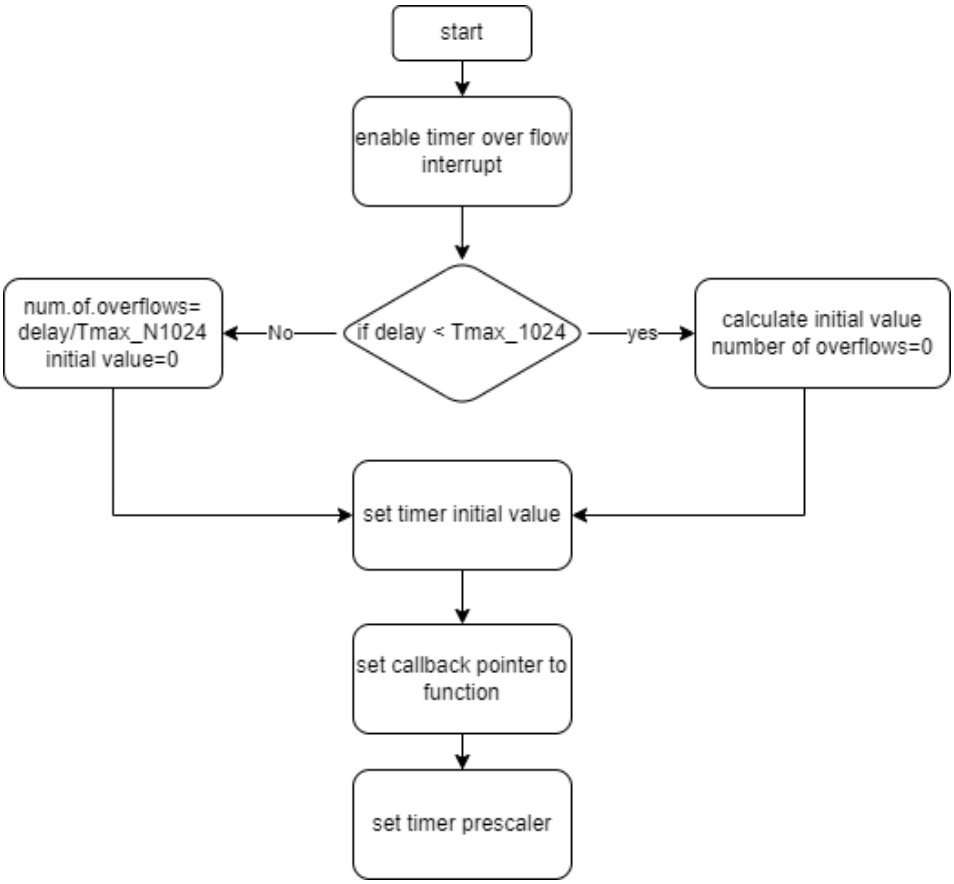
EN_STATE GPIO_checkstate(uint8 port_num,uint8 pin_num);



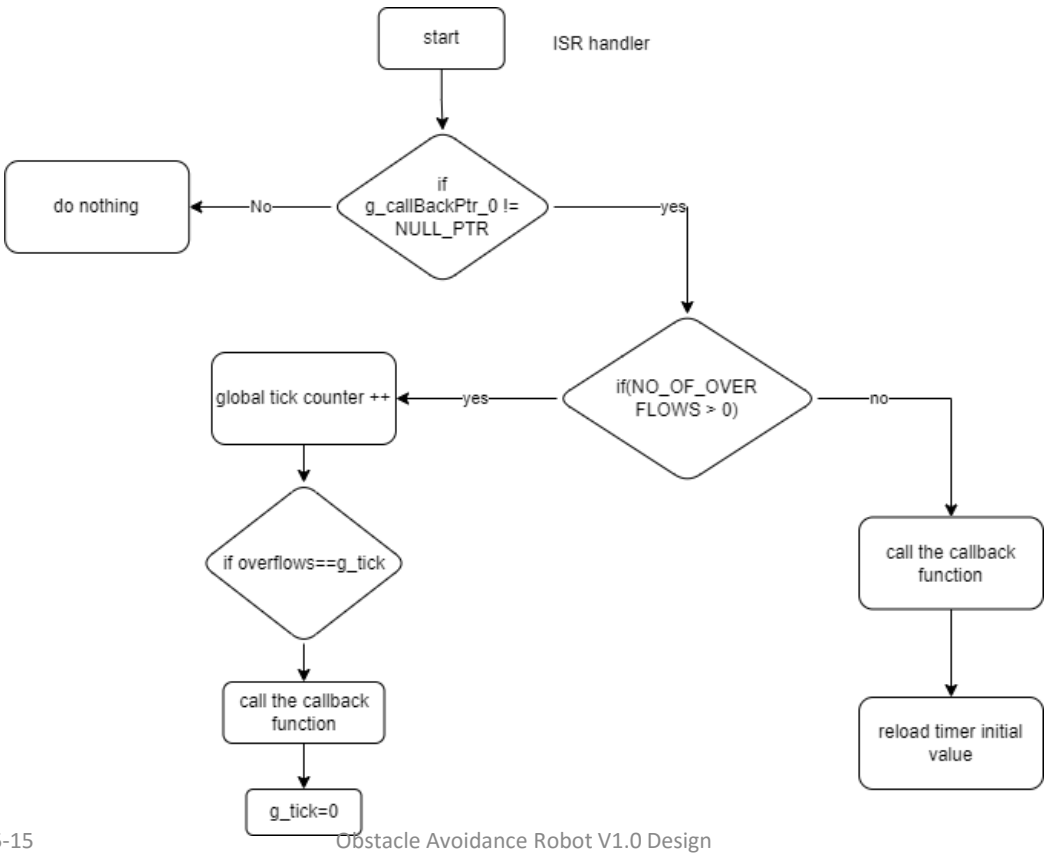
void Timer0_Delay(float delay);



void Timer0_event(uint16 delay,void(*g_ptr)(void));

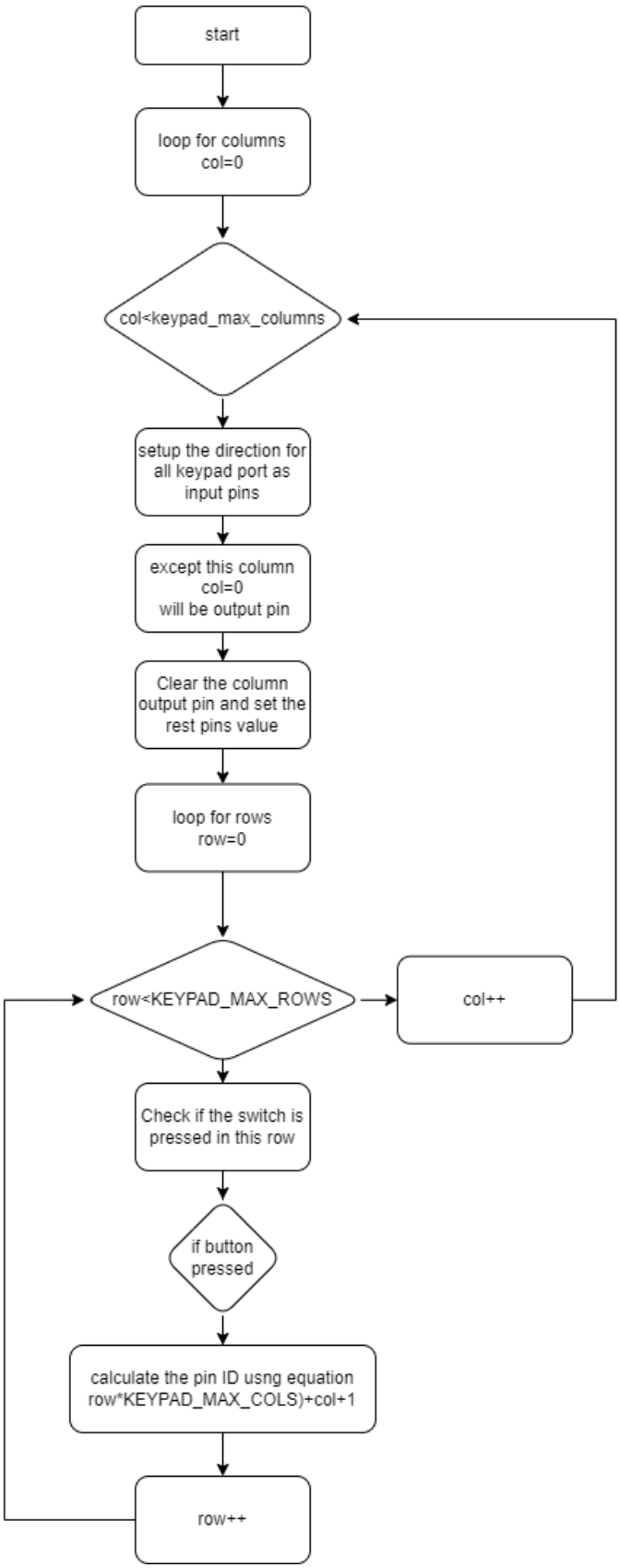


ISR (TIMER0_OVF_vect)



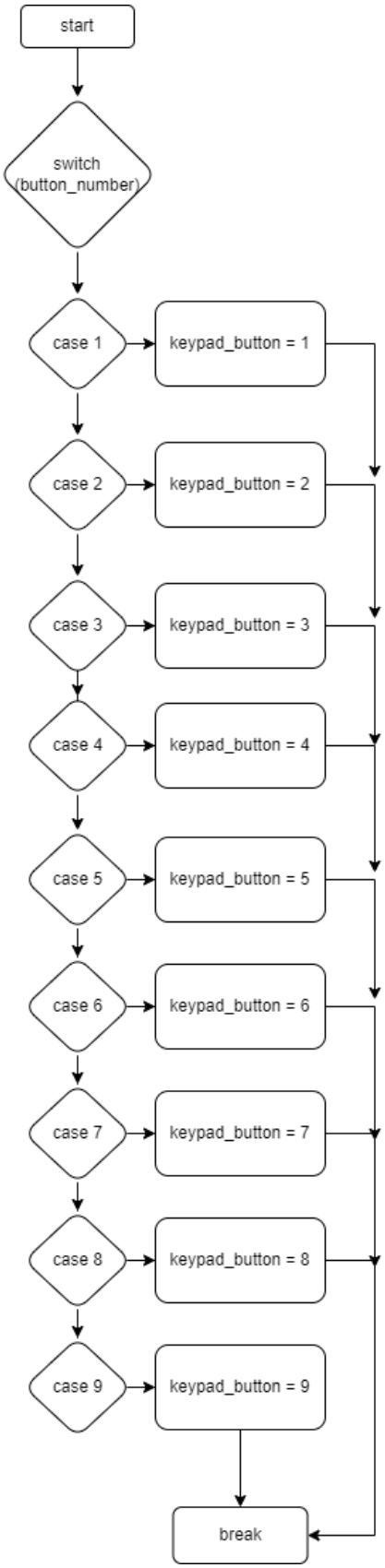
KEYPAD APIs flowchart

uint8 KEYPAD_getPressedKey(void);



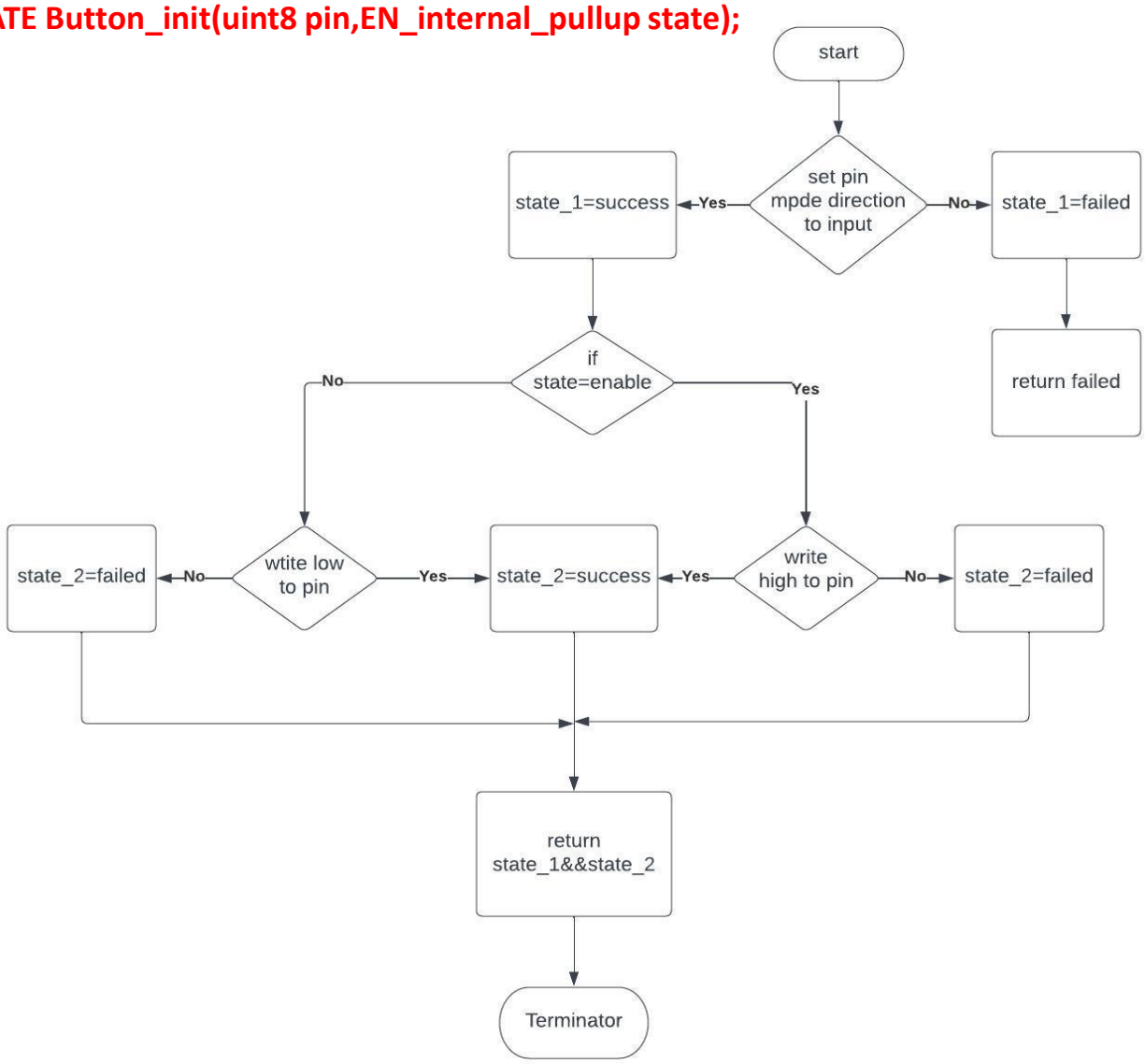
KEYPAD APIs flowchart

static uint8 KEYPAD_3x3_adjustKeyNumber(uint8 button_number);

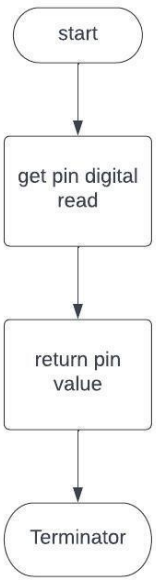


BUTTON APIs flowchart

EN_STATE Button_init(uint8 pin,EN_internal_pullup state);



uint8 **Button_Read(uint8 pin)**



GPIO pre compiling configuration

```
/*============= MACRO DEFINITION  =============*/
#define PORTA                (*((volatile uint8*)0x3B))
#define DDRA                 (*((volatile uint8*)0x3A))
#define PINA                 (*((volatile uint8*)0x39))
#define PORTB                (*((volatile uint8*)0x38))
#define DDRB                 (*((volatile uint8*)0x37))
#define PINB                 (*((volatile uint8*)0x36))
#define PORTC                (*((volatile uint8*)0x35))
#define DDRC                 (*((volatile uint8*)0x34))
#define PINC                 (*((volatile uint8*)0x33))
#define PORTD                (*((volatile uint8*)0x32))
#define DDRD                 (*((volatile uint8*)0x31))
#define PIND                 (*((volatile uint8*)0x30))

#define PORTA_ID             0
#define PORTB_ID             1
#define PORTC_ID             2
#define PORTD_ID             3
#define MAX_PORT_ID          4

#define PIN0   0
#define PIN1   1
#define PIN2   2
#define PIN3   3
#define PIN4   4
#define PIN5   5
#define PIN6   6
#define PIN7   7
#define MAX_PIN  8

/*=======PORT ID configuration =======*/
#define PORT_ID  0
#if (PORT_ID == 0)
#define PORT_NAME  PORTA_ID
#elif (PORT_ID == 1)
#define PORT_NAME  PORTB_ID
#elif (PORT_ID == 2)
#define PORT_NAME  PORTC_ID
#elif (PORT_ID == 3)
#define PORT_NAME  PORTD_ID
#else
#error "Number of PORT_ID should be equal to 0 or 1 or 2 or 3"
#endif
```

Description: PORT_NAME configuration ,its value should be 0(port A) or 1(port B) or 2(port C) or 3(port D)

GPIO pre compiling configuration

```
/*=======PIN ID configuration =======*/
#define PIN_ID  0
#if (PIN_ID == 0)
#define PIN_NUM  PIN0
#elif (PIN_ID == 1)
#define PIN_NUM PIN1
#elif (PIN_ID == 2)
#define PIN_NUM PIN2
#elif (PIN_ID == 3)
#define PIN_NUM PIN3
#elif (PIN_ID == 4)
#define PIN_NUM PIN4
#elif (PIN_ID == 5)
#define PIN_NUM PIN5
#elif (PIN_ID == 6)
#define PIN_NUM PIN6
#elif (PIN_ID == 7)
#define PIN_NUM PIN7
#else
#error "Number of PIN_ID should be equal to from 0 to 7"
#endif
```

Description: PIN_NUMBER configuration ,its value should be 0(port A) or 1(port B) or 2(port C) or 3(port D)

```
/*============== FUNCTION PROTOTYPE ==============*/
void set_direction(void);
void write_pin(void);
Uint8 read_pin(void);
void set_port_direction(void);
void write_port (void);
void read_port (void);
```

GPIO Linking configuration

```
/*============= MACRO DEFINITION  =============*/
#define PORTA ((volatile char*)0x39)
#define PORTB ((volatile char*)0x36)
#define PORTC ((volatile char*)0x33)
#define PORTD ((volatile char*)0x30)

#define OUTPUT 1
#define INPUT 0

#define HIGH 1
#define LOW 0

#define PIN0 0
#define PIN1 1
#define PIN2 2
#define PIN3 3
#define PIN4 4
#define PIN5 5
#define PIN6 6
#define PIN7 7
/*============= TYPE DEFINITION  =============*/
typedef struct portname{
                uint8 pinx;
                uint8 ddrx;
                uint8 portx;
}Sportname;

typedef struct pinconf{
                Sportname *PPortx;
                uint8 pin_no;
                uint8 dir;
                uint8 val;
}Spinconf;

typedef struct portconf{
                Sportname *PPortx;
                uint8 dir;
                uint8 val;
}Sportconf;
/*============= FUNCTION PROTOTYPE =============*/
void set_direction(Spinconf *port);
void write_pin(Spinconf *port);
Uint8 read_pin(Spinconf *port);
void set_port_direction(Sportconf *port);
void write_port (Sportconf *port);
void read_port (Sportconf *port);
```

External Interrupt pre compiling configuration

```
/*============= MACRO DEFINITION  =============*/
#define INT0_ID             0
#define INT1_ID             1
#define INT2_ID             2
#define LOW_LEVEL           0
#define ANY_CHANGE          1
#define FALLING             2
#define RISING              3

/*=======INT_source configuration =======*/
#define INT_source  0
#if (INT_source == 0)
#define INT_ID INT0_ID
#elif (INT_source == 1)
#define INT_ID INT1_ID
#elif (INT_source == 2)
#define INT_ID INT2_ID
#else
#error "Number of INT_source should be equal to 0 or 1 or 2"
#endif
```

Description: INT_source configuration ,its value should be 0(INT0_ID) or 1(INT1_ID) or 2(INT2_ID)

```
/*=======INT_TRIGGER configuration =======*/
#define INT_TRIGGER  0
#if (INT_TRIGGER== 0)
#define INT_MODE LOW_LEVEL
#elif (INT_TRIGGER== 1)
#define INT_MODE ANY_CHANGE
#elif (INT_TRIGGER== 2)
#define INT_MODE FALLING
#elif (INT_TRIGGER== 3)
#define INT_MODE RISING
#else
#error "Number of INT_TRIGGER should be equal to 0 or 1 or 2 or 3"
#endif
```

Description: INT_TRIGGER configuration ,its value should be 0(LOW_LEVEL) or 1(ANY_CHANGE) or 2(FALLING) or 3(RISING)

```
/*============= FUNCTION PROTOTYPE =============*/
void INT_init(void);
void INT0_setCallBack(void(*a_ptr)(void));
void INT1_setCallBack(void(*a_ptr)(void));
void INT2_setCallBack(void(*a_ptr)(void));
void INT_Deinit(void);
```

External Interrupt Linking configuration

```
/*============= MACRO DEFINITION  =============*/
#define INT0_pin            2           //PD2
#define INT1_pin            3           //PD3
#define INT2_pin            3           //PB2

/*============= TYPE DEFINITION  =============*/
typedef enum{
        EN_INT0,EN_INT1,EN_INT2
}EN_INT_source;

typedef enum{
        LOW_LEVEL,ANY_CHANGE,FALLING,RISING
}EN_INT_TRIGGER;

typedef struct{
        EN_INT_source source;
        EN_INT_TRIGGER trigger;
}ST_INT_Config;


/*============= FUNCTION PROTOTYPE =============*/
void INT_init(ST_INT_Config* Int_config);
void INT0_setCallBack(void(*a_ptr)(void));
void INT1_setCallBack(void(*a_ptr)(void));
void INT2_setCallBack(void(*a_ptr)(void));
void INT_Deinit(ST_INT_Config* Int_config);
```