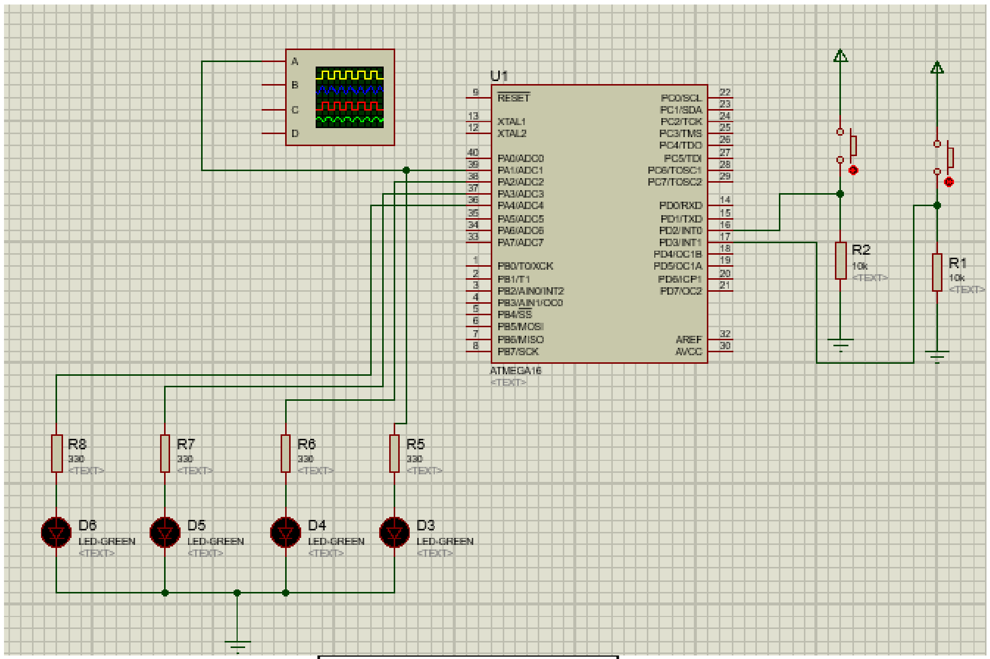
Design document

project title: LED sequence V3.0

Name: Hazem Ashraf

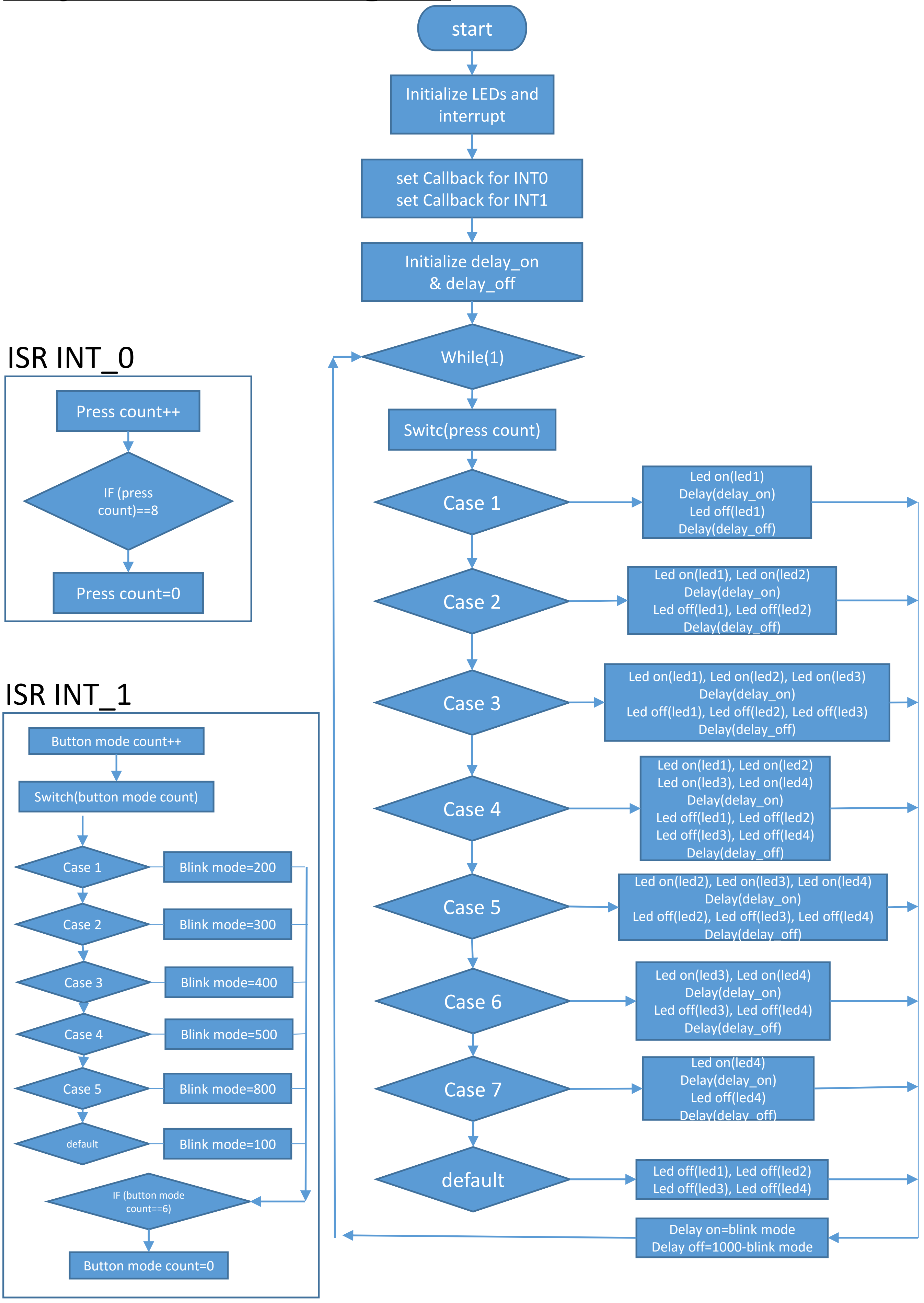
Project Description

- ***Hardware Requirements***
  - Four LEDs (**LED0**, **LED1**, **LED2**, **LED3**)
  - **Two** buttons (**BUTTON0** and **BUTTON1**)
- ***Software Requirements***
  - Initially, all LEDs are OFF
  - Once **BUTTON0** is pressed, **LED0** will blink with **BLINK_1** mode
  - Each press further will make another LED blinks **BLINK_1** mode
  - At the **fifth press**, **LED0** will changed to be **OFF**
  - Each **press further** will make only one LED is **OFF**
  - This will be repeated forever
  - The sequence is described below
    - Initially (OFF, OFF, OFF, OFF)
    - Press 1 (BLINK_1, OFF, OFF, OFF)
    - Press 2 (BLINK_1, BLINK_1, OFF, OFF)
    - Press 3 (BLINK_1, BLINK_1, BLINK_1, OFF)
    - Press 4 (BLINK_1, BLINK_1, BLINK_1, BLINK_1)
    - Press 5 (OFF, BLINK_1, BLINK_1, BLINK_1)
    - Press 6 (OFF, OFF, BLINK_1, BLINK_1)
    - Press 7 (OFF, OFF, OFF, BLINK_1)
    - Press 8 (OFF, OFF, OFF, OFF)
    - Press 9 (BLINK_1, OFF, OFF, OFF)
  - When BUTTON1 has pressed the blinking on and off durations will be changed
    - No press → **BLINK_1** mode (**ON**: 100ms, **OFF**: 900ms)
    - First press → **BLINK_2** mode (**ON**: 200ms, **OFF**: 800ms)
    - Second press → **BLINK_3** mode (**ON**: 300ms, **OFF**: 700ms)
    - Third press → **BLINK_4** mode (**ON**: 500ms, **OFF**: 500ms)
    - Fourth press → **BLINK_5** mode (**ON**: 800ms, **OFF**: 200ms)
    - Fifth press → **BLINK_1** mode
  - **USE EXTERNAL INTERRUPTS**

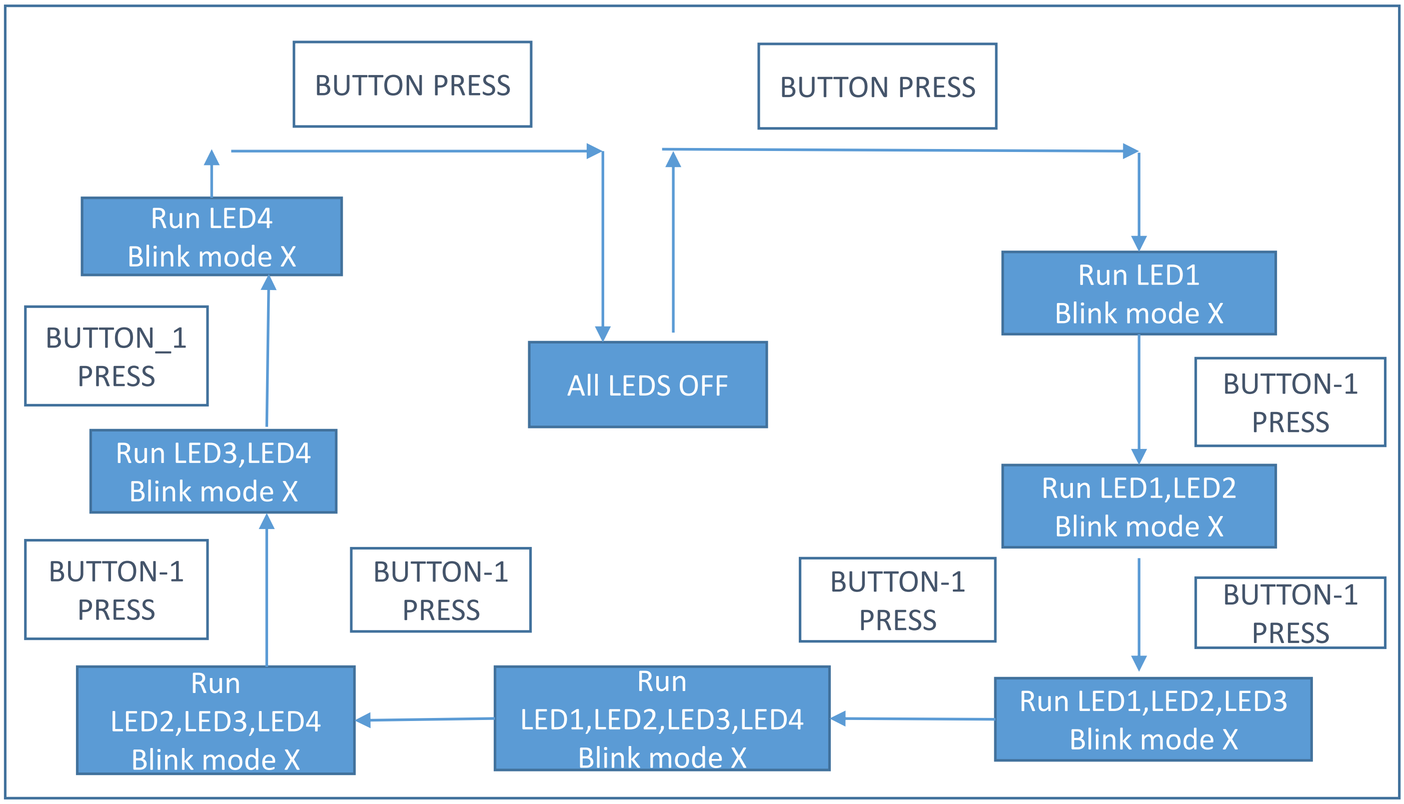


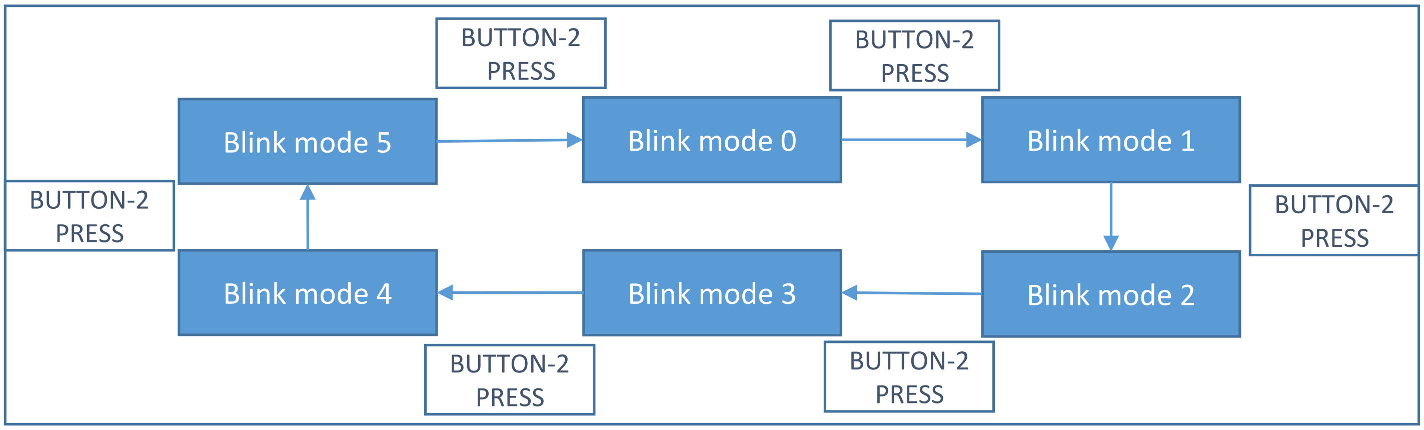
Circuit wiring

Project flowchart diagram

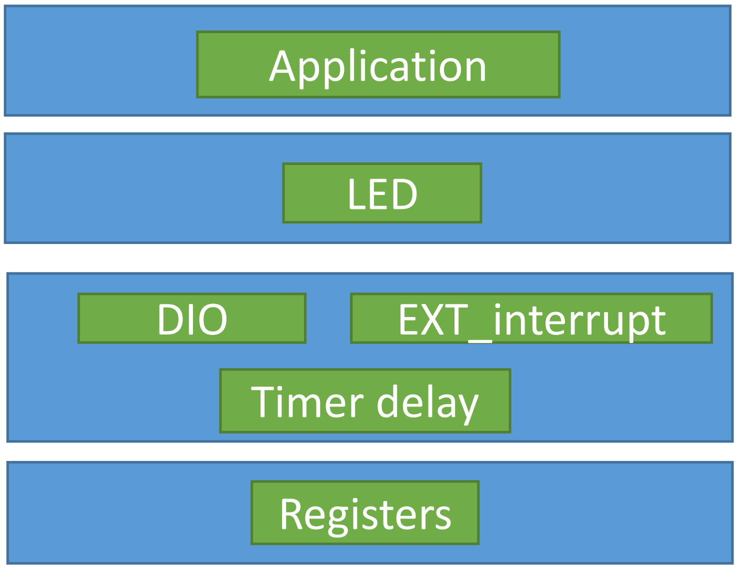

start
Initialize LEDs and interrupt
set Callback for INT0
set Callback for INT1
Initialize delay_on & delay_off
While(1)
Switc(press count)
Case 1
Led on(led1) Delay(delay_on) Led off(led1) Delay(delay_off)
Case 2
Led on(led1), Led on(led2) Delay(delay_on) Led off(led1), Led off(led2) Delay(delay_off)
Case 3
Led on(led1), Led on(led2), Led on(led3) Delay(delay_on) Led off(led1), Led off(led2), Led off(led3) Delay(delay_off)
Case 4
Led on(led1), Led on(led2) Led on(led3), Led on(led4) Delay(delay_on) Led off(led1), Led off(led2) Led off(led3), Led off(led4) Delay(delay_off)
Case 5
Led on(led2), Led on(led3), Led on(led4) Delay(delay_on) Led off(led2), Led off(led3), Led off(led4) Delay(delay_off)
Case 6
Led on(led3), Led on(led4) Delay(delay_on) Led off(led3), Led off(led4) Delay(delay_off)
Case 7
Led on(led4) Delay(delay_on) Led off(led4) Delay(delay_off)
default
Led off(led1), Led off(led2) Led off(led3), Led off(led4)
Delay on=blink mode
Delay off=1000-blink mode
ISR INT_0
Press count++
IF (press count)==8
Press count=0
ISR INT_1
Button mode count++
Switch(button mode count)
Case 1
Blink mode=200
Case 2
Blink mode=300
Case 3
Blink mode=400
Case 4
Blink mode=500
Case 5
Blink mode=800
default
Blink mode=100
IF (button mode count==6)
Button mode count=0

Project state machine diagram





Layered architecture



Project Modules APIs

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;

typedef struct{
uint8 pinx;
uint8 ddrx;
uint8 portx;
}ST_register_name;
typedef ST_register_name* REG_NAME;
```

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

EN_STATE **pinMode(uint8 pin_no,EN_PIN_DIRECTION pin_direction);**

Description:

- PinMode:used to set pin direction input/output
- function parameters
- pin_no:pin number to set
- pin_direction:direction of the pin
- Return success pin number is in the range, FAILED if pin number out of the range

EN_STATE **digitalWrite(uint8 pin_no,EN_PIN_VALUE pin_val);**

Description:

- digitalWrite:used to write high/low to specific pin
- function parameters
- pin_no:pin number to write
- pin_val:output value high / low
- Return success pin number is in the range, FAILED if pin number out of the range

Project Modules APIs

- ## GPIO module APIs

EN_STATE **digitalRead(uint8 pin_no,uint8 *pin_val);**

Description:
- digitalRead:used to read specific pin value
- function parameters
- pin_no:pin number to read
- pin_val:address to variable of the return reading
- Return success pin number is in the range, FAILED if pin number out of the range

EN_STATE portMode(REG_NAME port,EN_PORT_DIRECTION port_direction);

Description:
- portMode: used to specific port direction
- function parameters
- port: port name (PORTA-PORTB-PORTC-PORTD)
- port_direction: direction of the port
- Return success port name is in the range, FAILED if port name out of the range

EN_STATE digitalWrite_Port(REG_NAME port,EN_PORT_VALUE port_val);

Description
- digitalWrite_PORT:used to write high/low to specific port
- function parameters
- port: port name (PORTA-PORTB-PORTC-PORTD)
- port_val: output value HIGH / LOW
- Return success port name is in the range, FAILED if port name out of the range

EN_STATE digitalRead_Port(REG_NAME port,uint8 *port_val);

Description
- digitalRead_PORT:used to read specific port value
- function parameters
- port: port name (PORTA-PORTB-PORTC-PORTD)
- port_val: address to variable of the return reading
- Return success port name is in the range, FAILED if port name out of the range

EN_STATE **Enable_PULLUP (uint8 pin_no);**

Description
- active internal pull up resistor for specific pin
- pin_no:pin number to set
- Return success pin number is in the range, FAILED if pin number out of the range

Project Modules APIs

- 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

Project Modules APIs

- EXT-interrupt module APIs

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

Project Modules APIs

- 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:

ST_timer0_config delay_on={100};

The remaining members don't care about initialization

```
/*============ MACRO DEFINITION ============*/
#define TCCR0 (*((volatile uint8*)0x53))
#define TCNT0 (*((volatile uint8*)0x52))
#define OCR0  (*((volatile uint8*)0x5C))
#define TIFR  (*((volatile uint8*)0x58))
#define TIMSK (*((volatile uint8*)0x59))
//TCCR0 timer counter control register
#define CS00  0
#define CS01  1
#define CS02  2
#define WGM01 3
#define COM00 4
#define COM01 5
#define WGM00 6
#define FOC0  7
//TIMSK interrupt mask register
#define TOIE0  0
#define OCIE0  1
#define TOIE1  2
#define OCIE1B 3
#define OCIE1A 4
#define TICIE1 5
#define TOIE2  6
#define OCIE2  7
//TIFR interrupt flag register
#define TOV0  0
#define OCF0  1
#define TOV1  2
#define OCF1B 3
#define OCF1A 4
#define ICF1  5
```

Project Modules APIs

- Timer0_delay module APIs (TIMER0_Utilities.h)

```
/*============MACRO DEFINITION ============*/
#define max_count 256
#define min_count  1
#define init_value(T_max,T_delay,tick)  (((float)T_max-T_delay)/tick)

//pre_scaler values for TIMER0
#define N0    0
#define N1    1
#define N8    8
#define N64   64
#define N256  256
#define N1024 1024

//T_max in (ms) delay for each pre_scaler
#define Tmax_N1    0.26F
#define Tmax_N8    2.05F
#define Tmax_N64   16.38F
#define Tmax_N256  65.54F
#define Tmax_N1024 262.14F

//T_min in (ms) delay for each pre_scaler
#define Tmin_N1    0.001F
#define Tmin_N8    0.008F
#define Tmin_N64   0.064F
#define Tmin_N256  0.256F
#define Tmin_N1024 1.024F
```

Project Modules APIs

- Timer0_delay module APIs (TIMER_0.h)

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

void Timer0_Config(ST_timer0_config* T);

Description:

- used to calculate timer settings
- calculate pre_scaler value
- calculate number of overflows
- calculate timer initial value

void Timer0_Delay(ST_timer0_config* T);

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

Project Modules APIs

❑ LED module APIs

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

#define LED_logic 1 //1:positive logic , 2:negative logic
Description : Macro used to configure LED logic connection

/*============= FUNCTION PROTOTYPE =============*/
EN_STATE **LED_init(uint8 led);**
Description:
- LED_init: used to initialize LED direction and initial value for the pin
- function parameters
- Led: pin number to be set
- Return success pin number is in the range, FAILED if pin number out of the range

EN_STATE **LED_digitalwrite(uint8 led,EN_PIN_VALUE value);**
Description
- LED_digitalwrite**:** used to write high/low to specific led
- function parameters
- Led: pin number to be set
- Value: led high/low
- Return success pin number is in the range, FAILED if pin number out of the range

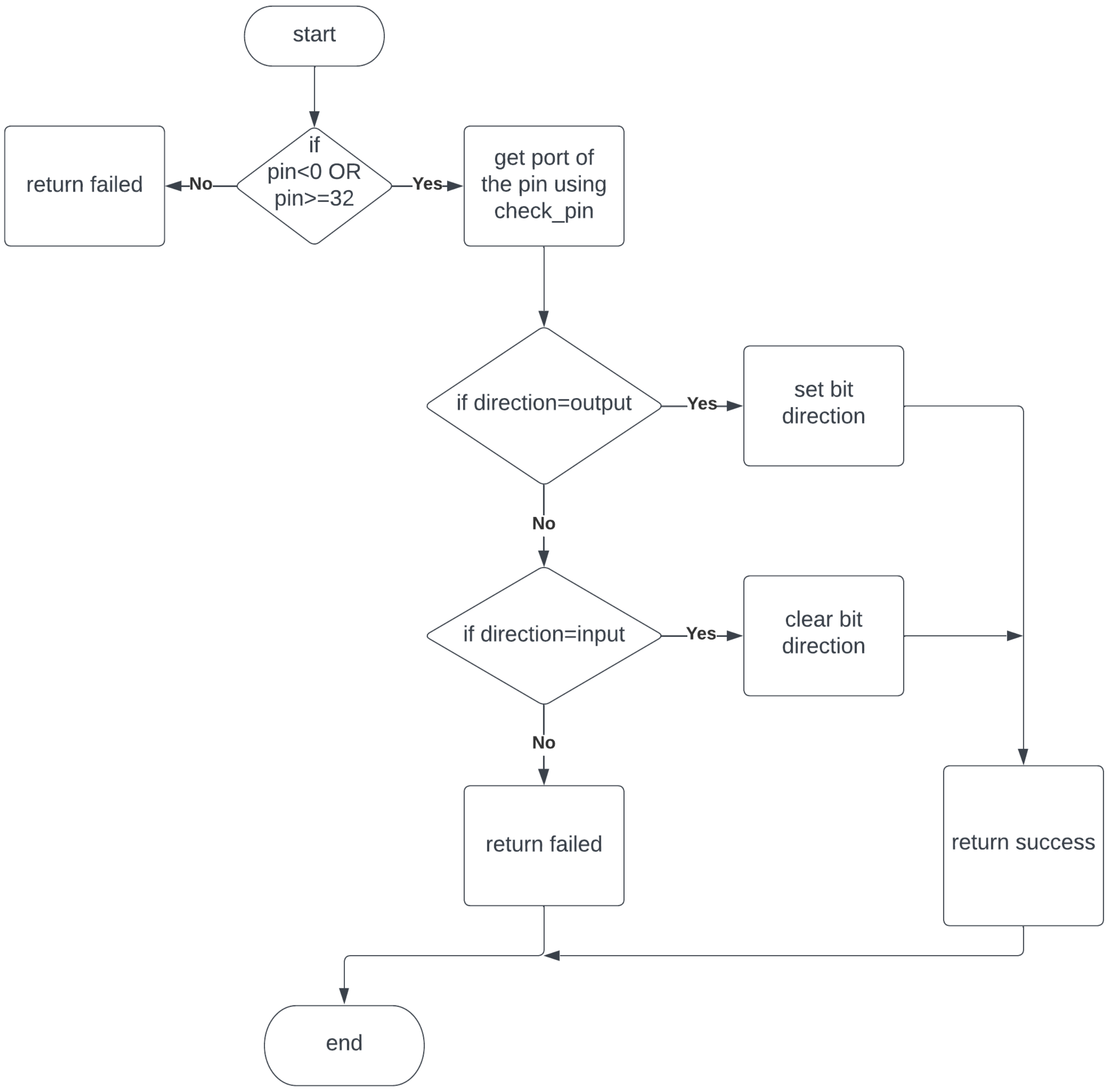
❑ BUTTON module APIs

/*============= 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

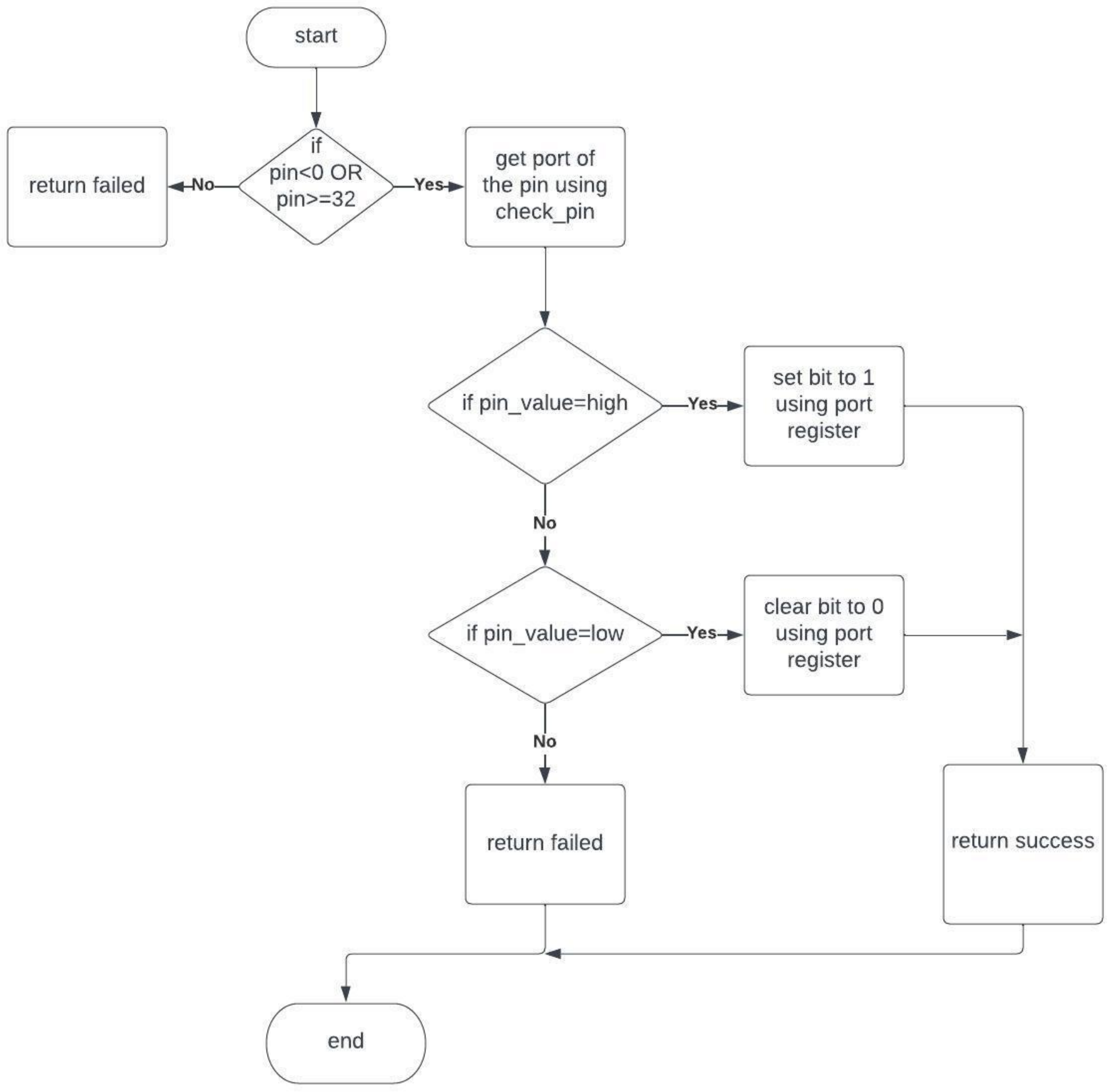
APIs flowcharts

EN_STATE **pinMode(uint8 pin_no,EN_PIN_DIRECTION pin_direction);**



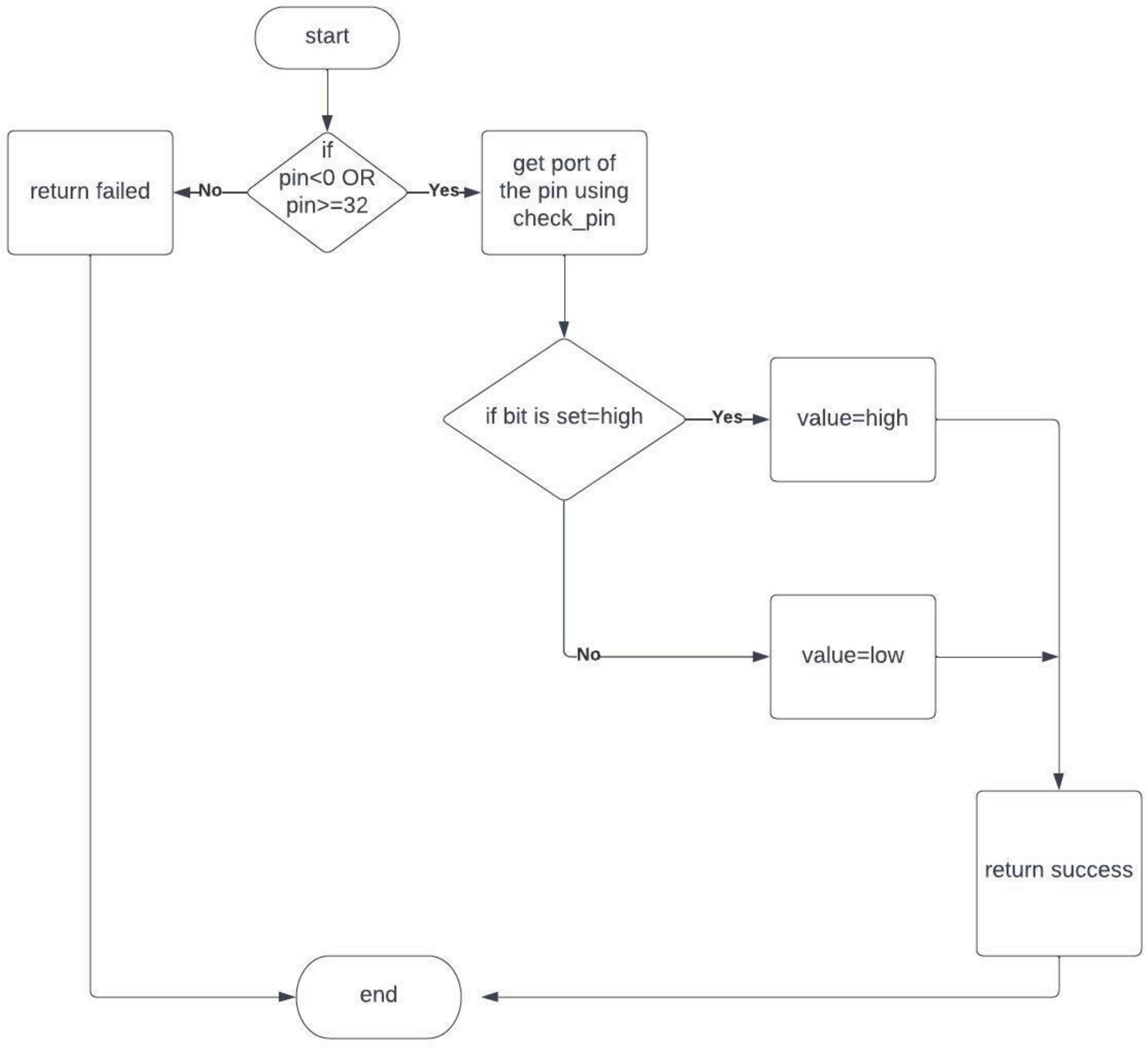
APIs flowcharts

- EN_STATE **digitalWrite(uint8 pin_no,EN_PIN_VALUE pin_val);**



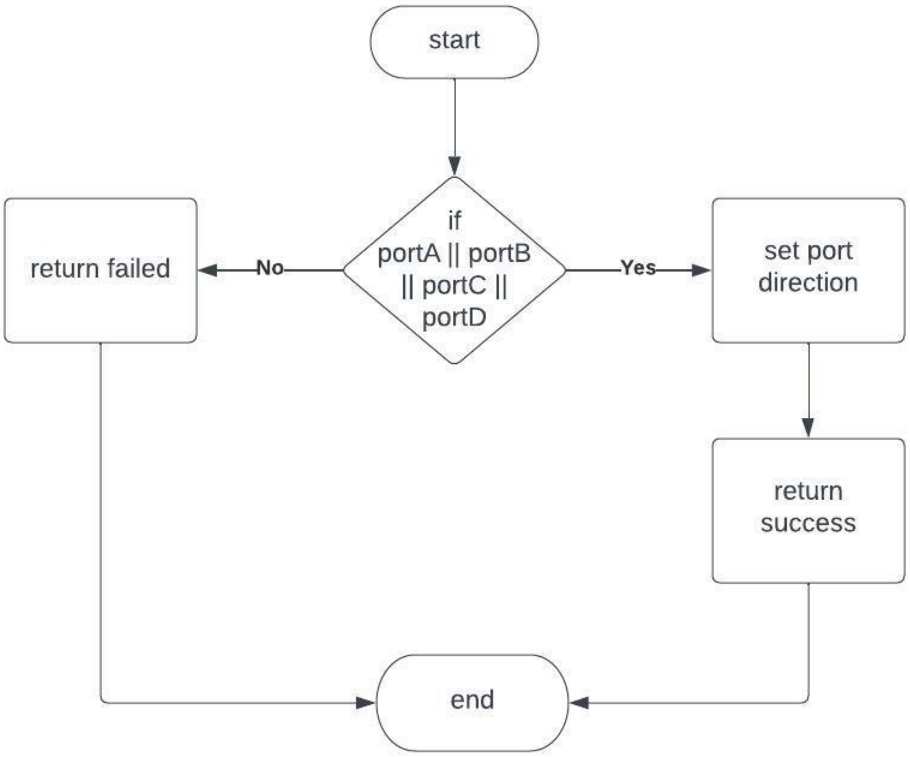
APIs flowcharts

- EN_STATE **digitalRead(uint8 pin_no,uint8 *pin_val);**

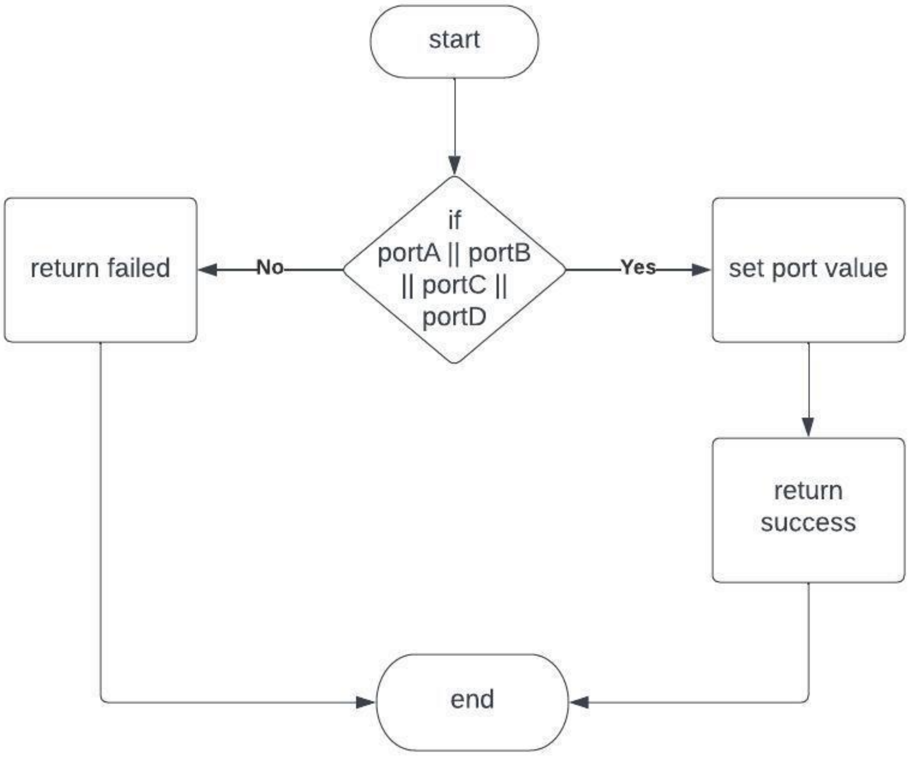


APIs flowcharts

EN_STATE portMode(REG_NAME port,EN_PORT_DIRECTION port_direction);

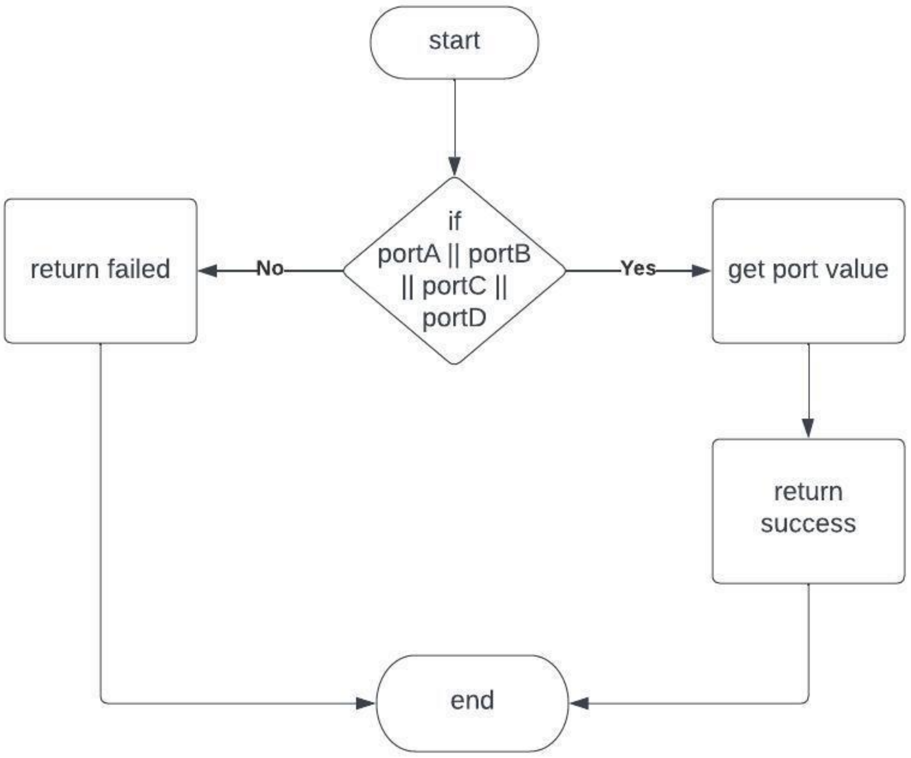


EN_STATE digitalWrite_Port(REG_NAME port,EN_PORT_VALUE port_val);

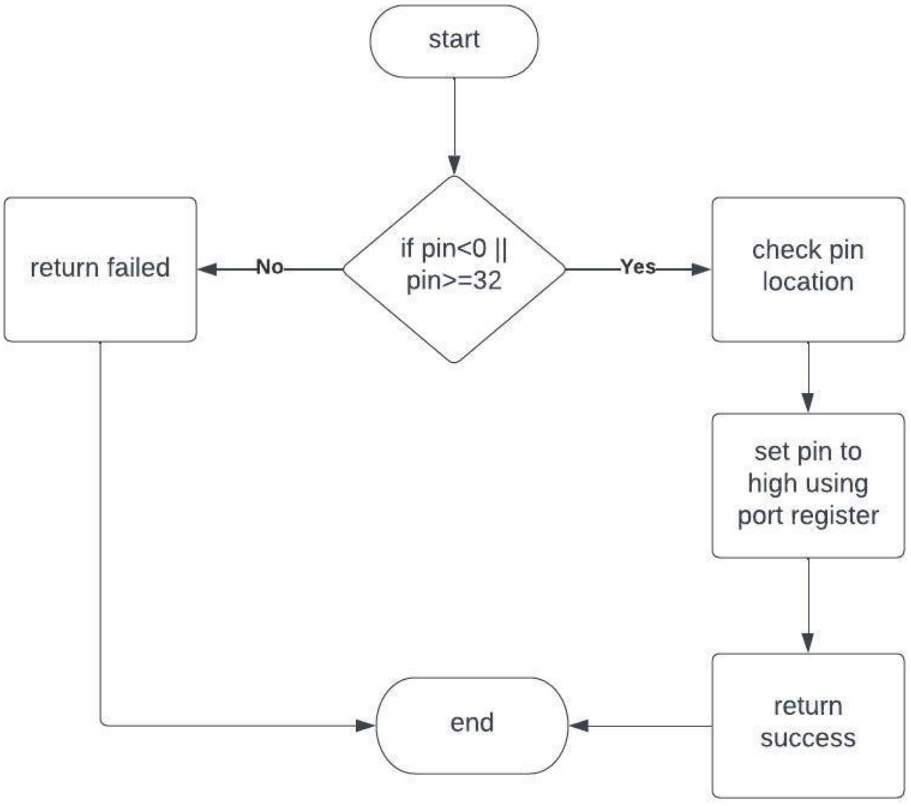


APIs flowcharts

EN_STATE digitalRead_Port(REG_NAME port,uint8 *port_val);

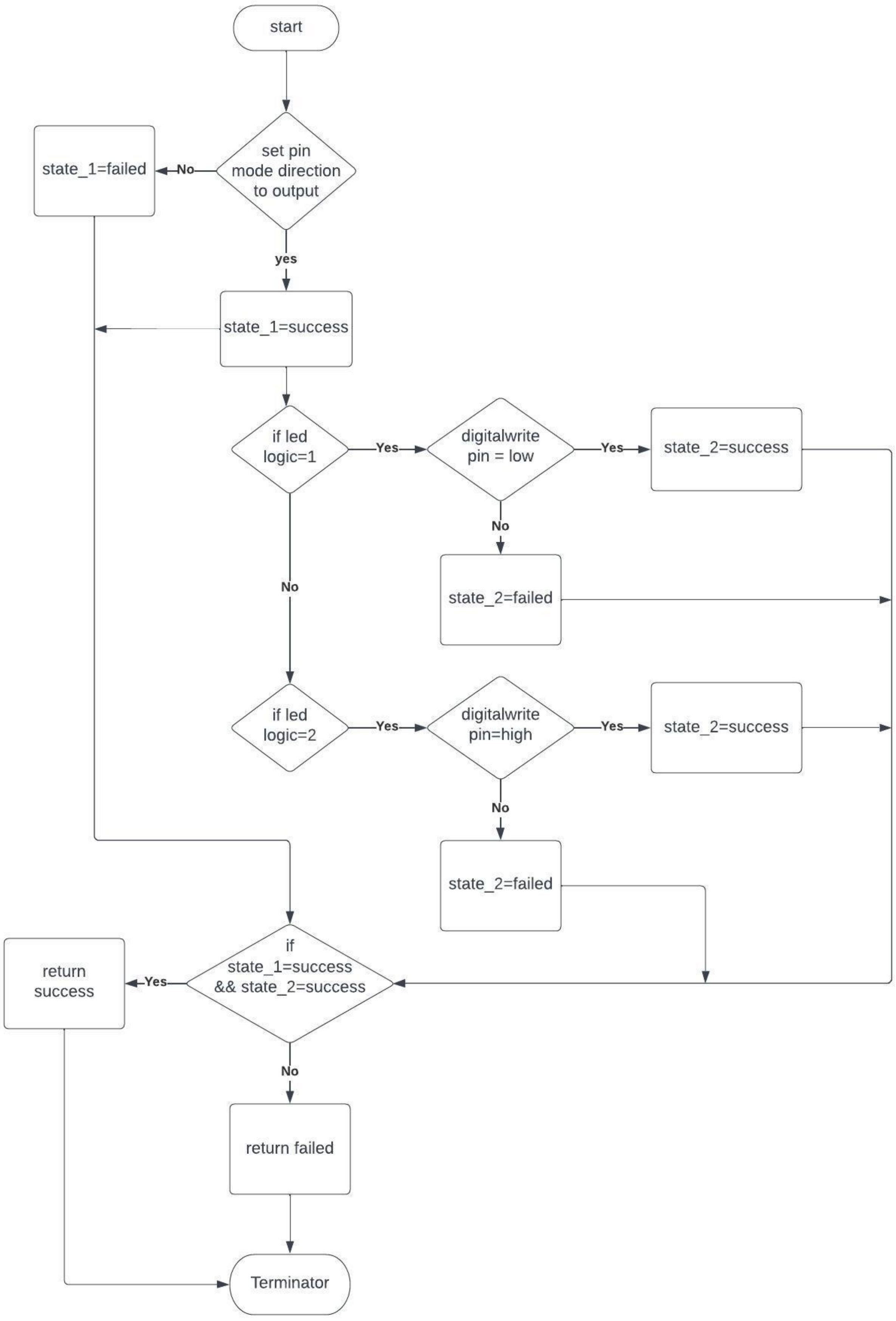


EN_STATE Enable_PULLUP (uint8 pin_no);



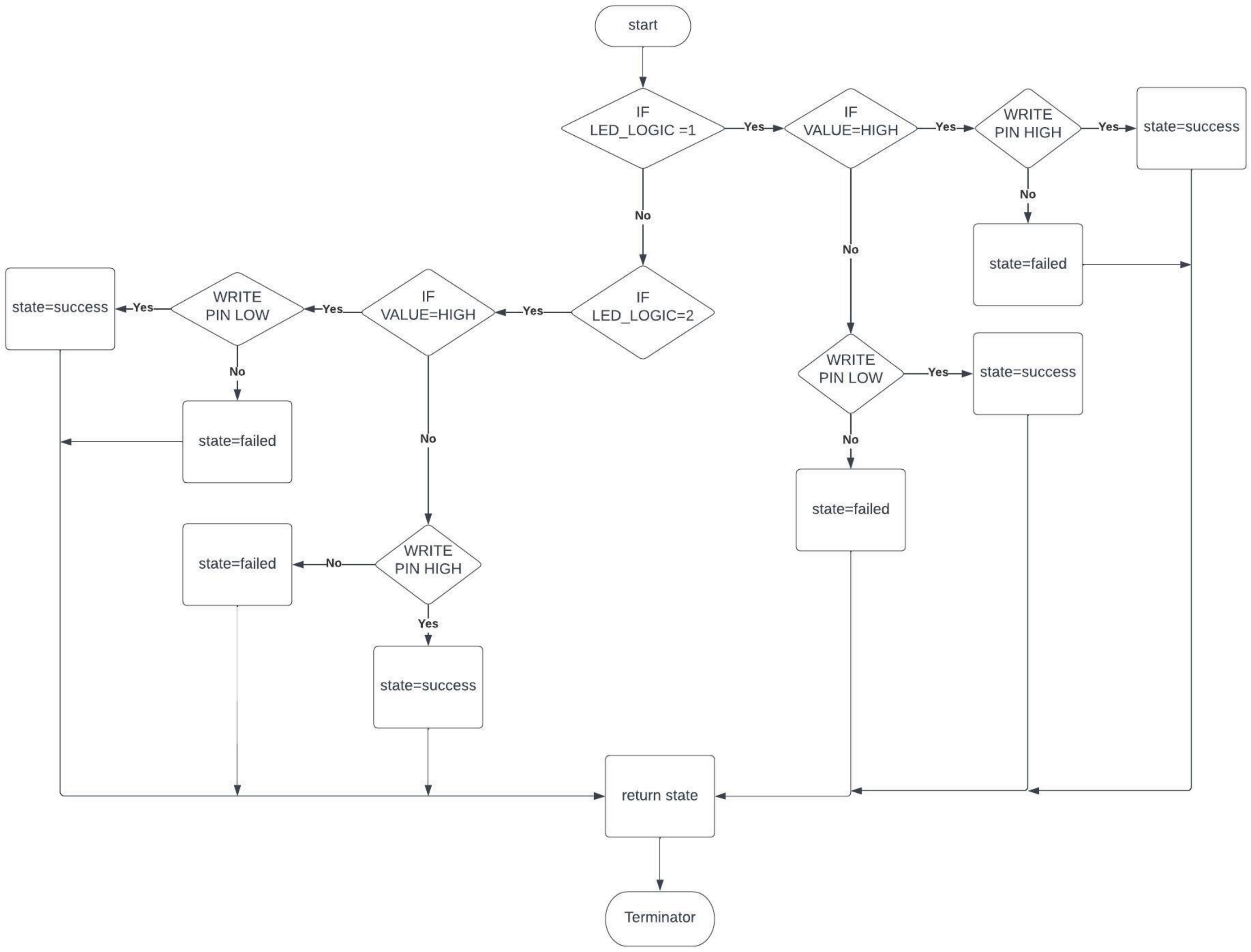
APIs flowcharts

EN_STATE **LED_init(uint8 led);**



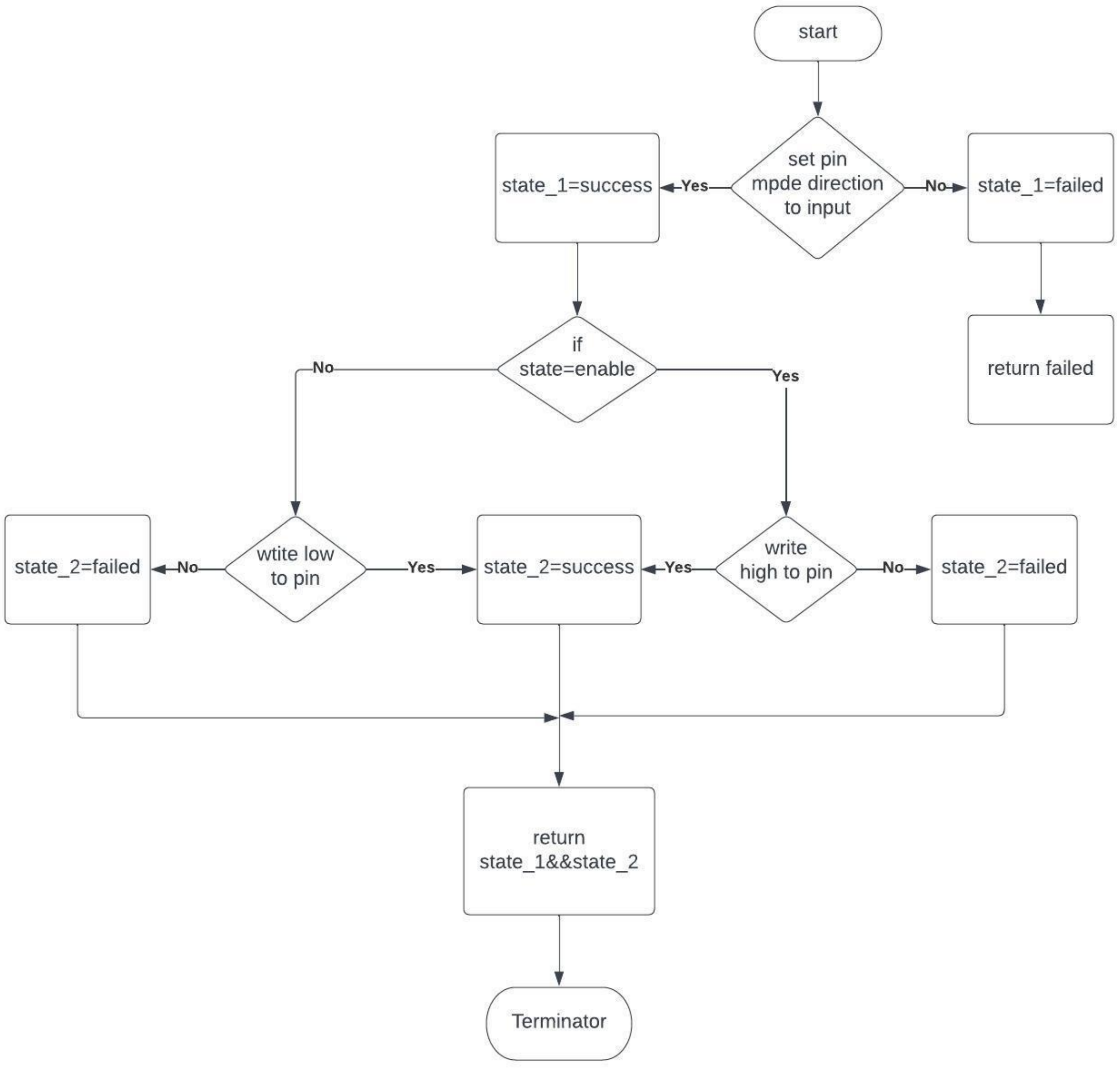
APIs flowcharts

EN_STATE **LED_digitalwrite(uint8 led,EN_PIN_VALUE value);**

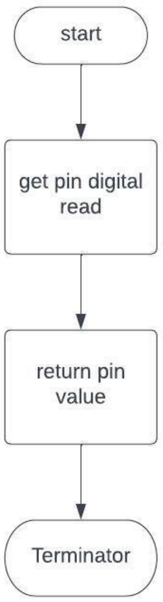


APIs flowcharts

EN_STATE Button_init(uint8 pin,EN_internal_pullup state);

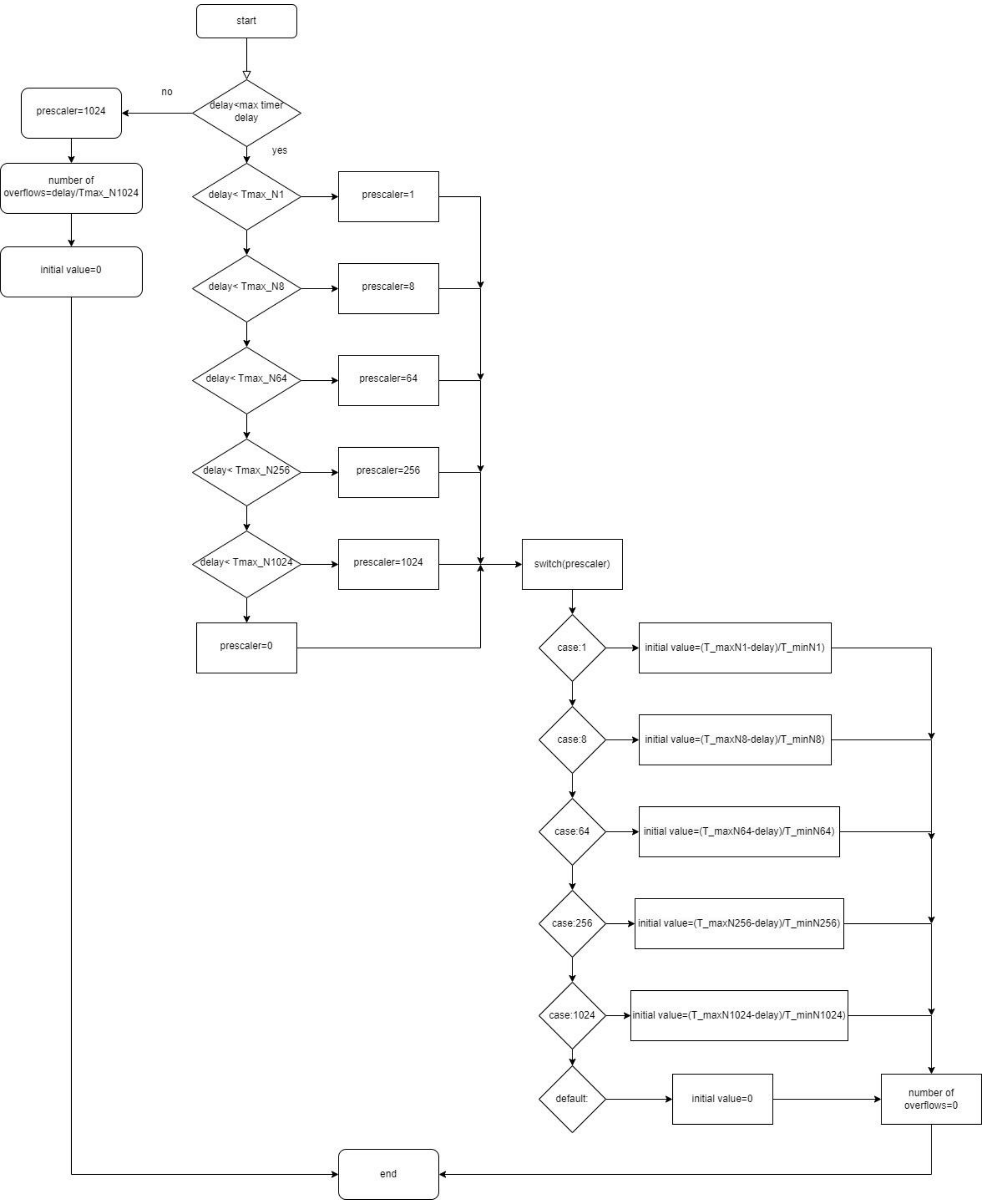


uint8 Button_Read(uint8 pin)



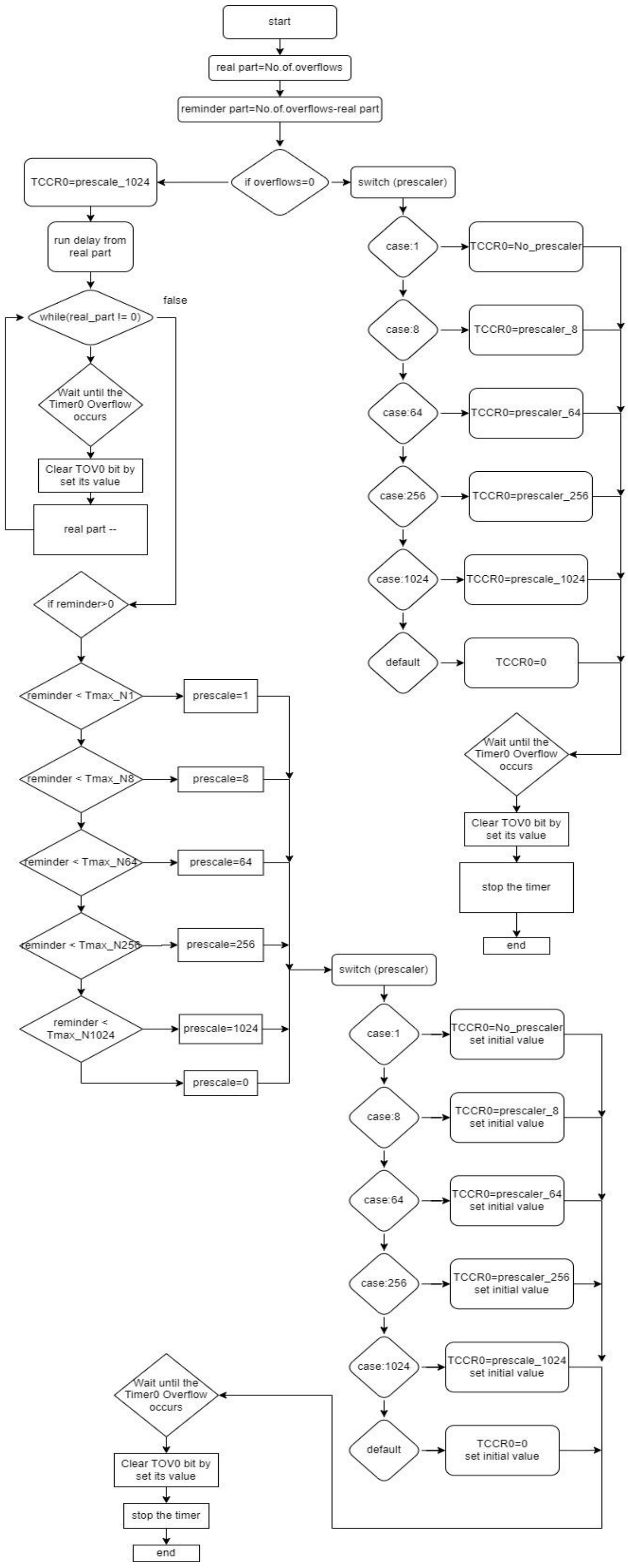
APIs flowchart

void Timer0_Config(ST_timer0_config* T);



APIs flowchart

void Timer0_Delay(ST_timer0_config* T);


start
real part=No.of.overflows
reminder part=No.of.overflows-real part
if overflows=0
TCCR0=prescale_1024
run delay from real part
while(real_part != 0)
false
Wait until the Timer0 Overflow occurs
Clear TOV0 bit by set its value
real part --
if reminder>0
reminder < Tmax_N1
prescale=1
reminder < Tmax_N8
prescale=8
reminder < Tmax_N64
prescale=64
reminder < Tmax_N256
prescale=256
reminder < Tmax_N1024
prescale=1024
prescale=0
switch (prescaler)
case:1
TCCR0=No_prescaler
case:8
TCCR0=prescaler_8
case:64
TCCR0=prescaler_64
case:256
TCCR0=prescaler_256
case:1024
TCCR0=prescale_1024
default
TCCR0=0
Wait until the Timer0 Overflow occurs
Clear TOV0 bit by set its value
stop the timer
end
switch (prescaler)
case:1
TCCR0=No_prescaler set initial value
case:8
TCCR0=prescaler_8 set initial value
case:64
TCCR0=prescaler_64 set initial value
case:256
TCCR0=prescaler_256 set initial value
case:1024
TCCR0=prescale_1024 set initial value
default
TCCR0=0 set initial value
Wait until the Timer0 Overflow occurs
Clear TOV0 bit by set its value
stop the timer
end