Design document

project title: Simple ATM Machine Design

TEAM 3:

Sara Mohamed

Yousef Abbas

Khaled Mustafa

Hazem Ashraf

Table of contents:

Project introduction

- **Description**

1. ATM MCU
    1. This MCU will handle transaction main flows
    2. After Reset
        1. Welcome message is displayed for 1s "Welcome to ATM"
        2. "Insert a Card" message is displayed in the first line
        3. No action can be taken further and all other input devices are blocked until a trigger signal came from the CARD ECU
    3. After a trigger signal is received from the CARD ECU
        1. "Enter Your PIN" message is displayed in the first line
        2. Waiting for the input from the keypad and type it in '****' format in the second line
        3. PIN is only four numeric characters
        4. Pressing the Enter/Zero button for 2s will initiate a communication between the ATM ECU and the CARD ECU to validate if the PIN is correct or not
        5. If the PIN is not correct, repeat for further 2 trials, and then if it is still wrong, sound the alarm and lock every input in the ATM, this blocking can be revealed by hard reset.
        6. If the PIN is correct, then "Enter Amount" message is displayed in the first line and wait for the amount to be entered from the keypad and appeared in the second line
        7. Amount is a float string with max 4 integer digits and 2 decimal digits "0000.00"
        8. You can enter '0' when pressing Enter/Zero button for less than 2s
        9. After entering the amount to withdraw, several checks on the database are done to finalize the transaction
            1. Check if there is an account attached to this card
            2. Check if the card is blocked or not
            3. Check if the amount required exceeds the maximum daily limit or not
            4. Check for available amount
        10. If one of the checks failed, a declined message will appear accordingly
            1. "This is a fraud card" – if the card PAN is not found – Alarm will be initiated
            2. "This card is stolen" – if the card is blocked– Alarm will be initiated
            3. "Maximum limit is exceeded" – if the required amount exceeds the maximum allowed limit
            4. "Insufficient fund" – if the balance is lower than the required amount
        11. If all checks are passed then "Approved Transaction" message is displayed for 1s and the remaining balance is displayed for 1s "Remaining Balance: 0000.00"
        12. After the checks are done and messages are displayed, display "Ejecting Card" message for 1s
        13. Repeat from the after reset again
    4. Data base
        1. The data base will be hard coded array of structures for accounts that contains (PAN, Account State (blocked/running), and balance)
        2. The maximum allowed limit will be hardcoded "5000.00"

Project introduction

CARD MCU

1. The CARD MCU has two modes of operations
    1. Programming Mode
        1. The CARD MCU will enter this mode after reset
        2. For the first time only the MCU will send the following messages to the terminal
            1. "Please Enter Card PAN:" and wait for the PAN
            2. "Please Enter New PIN:" and wait for the PIN
            3. "Please Confirm New PIN:" and wait for the PIN
            4. If PIN is matched, then change to user mode
            5. If PIN is not matched, not numeric, and exceeds 4 characters, then "Wrong PIN" message is displayed and repeat from step 2
        3. For any further after resets
            1. "Please press 1 for entering user mode and 2 for programming mode: " message is sent to the terminal and wait for a valid response, only accepts 1 or 2
        4. PAN is 16 to 19 length numeric string
        5. PIN is 4 numeric digits
        6. All data taken will be stored in the EEPROM
    2. User Mode
        1. The CARD MCU will enter this mode
            1. After completing the programming mode
            2. Or after choosing 2 in any further after resets
        2. In this mode, the CARD ECU will send a trigger signal to the ATM ECU that will make the ATM initiate its flow

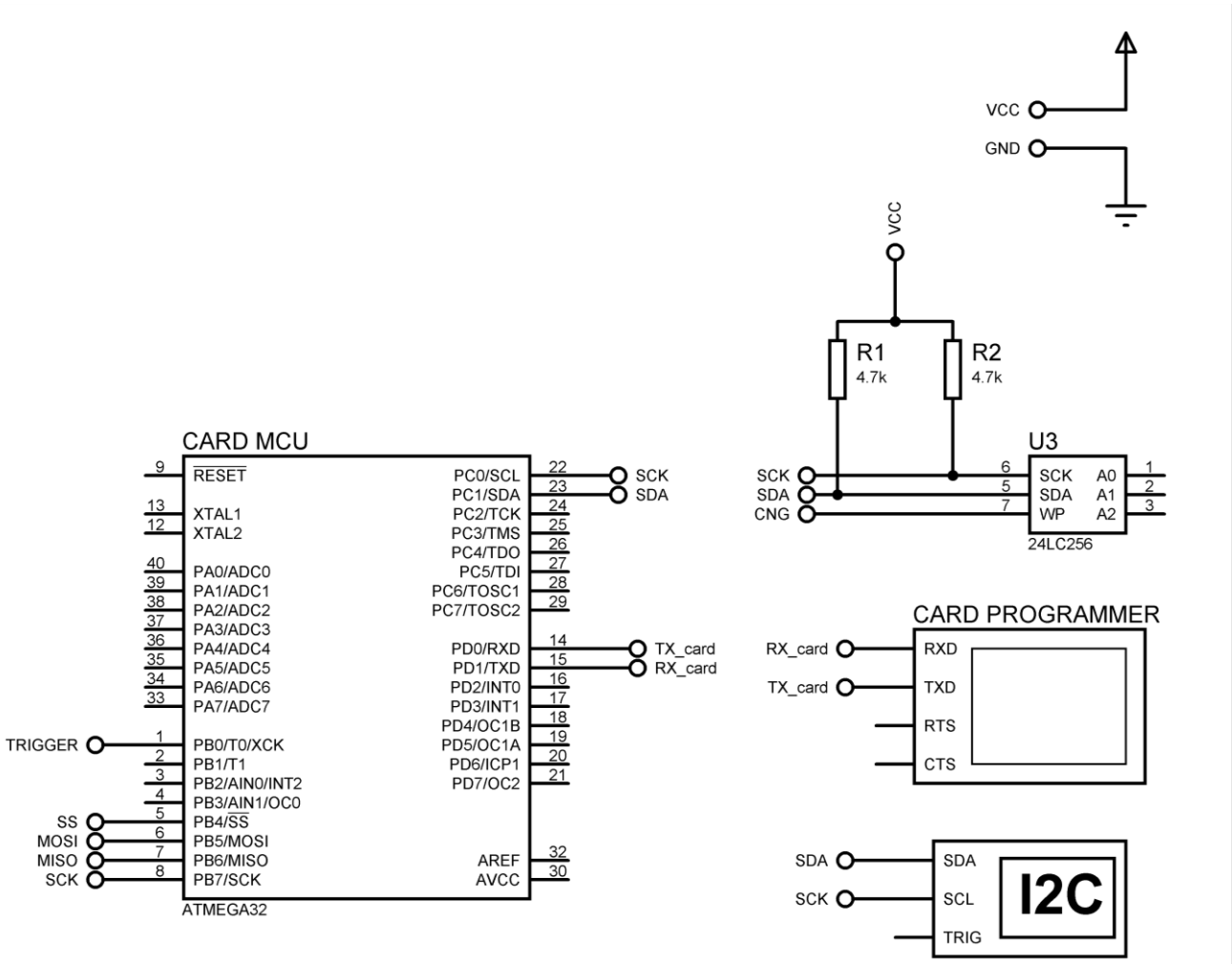
Project introduction

- **System Requirements**

Hardware Requirements:

1. ATM ECU
    1. ATM MCU
    2. 16 x 2 LCD
    3. 3 x 3 Keypad
    4. Buzzer
    5. Enter/Set Button
2. CARD ECU
    1. CARD MCU
    2. EEPROM
    3. Serial Terminal (Use Putty on your PC)

- CARD MCU Wiring

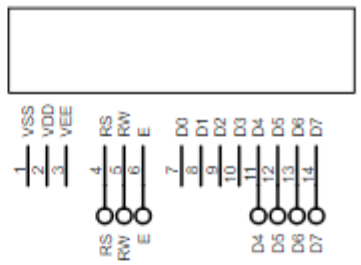


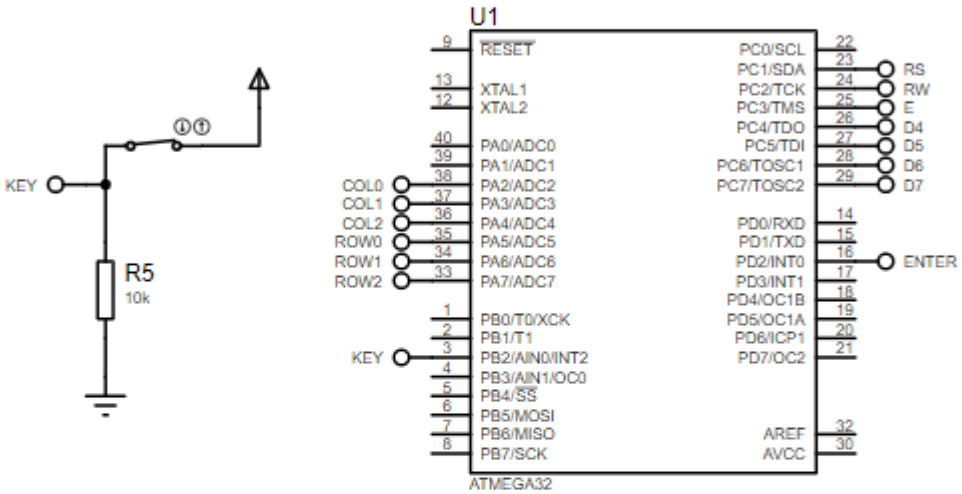
CARD MCU
Wiring diagram

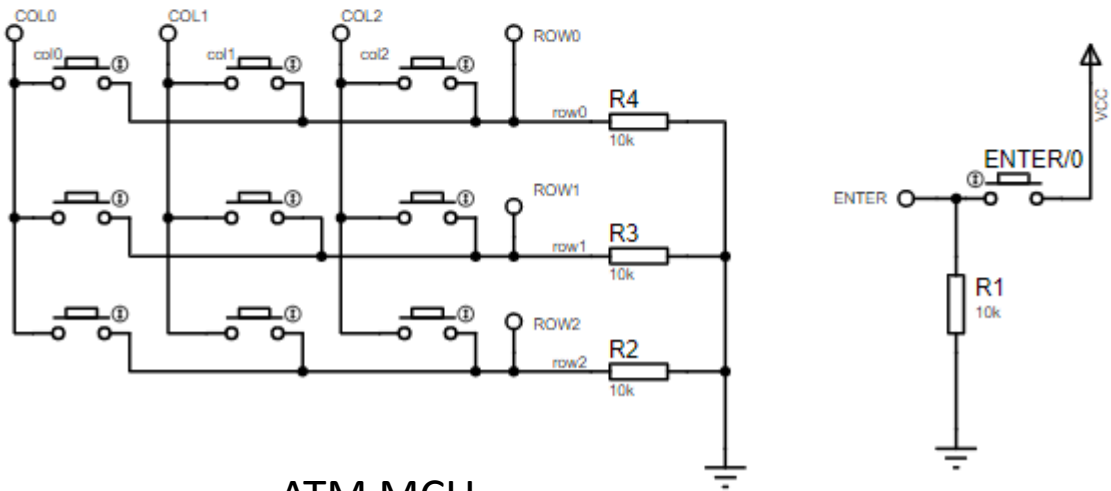
- ATM MCU Wiring

Hardware Requirements:

1. ATM ECU
    1. ATM MCU
    2. 16 x 2 LCD
    3. 3 x 3 Keypad
    4. Buzzer
    5. Enter/Set Button
2. CARD ECU
    1. CARD MCU
    2. EEPROM
    3. Serial Terminal (Use Putty on your PC)





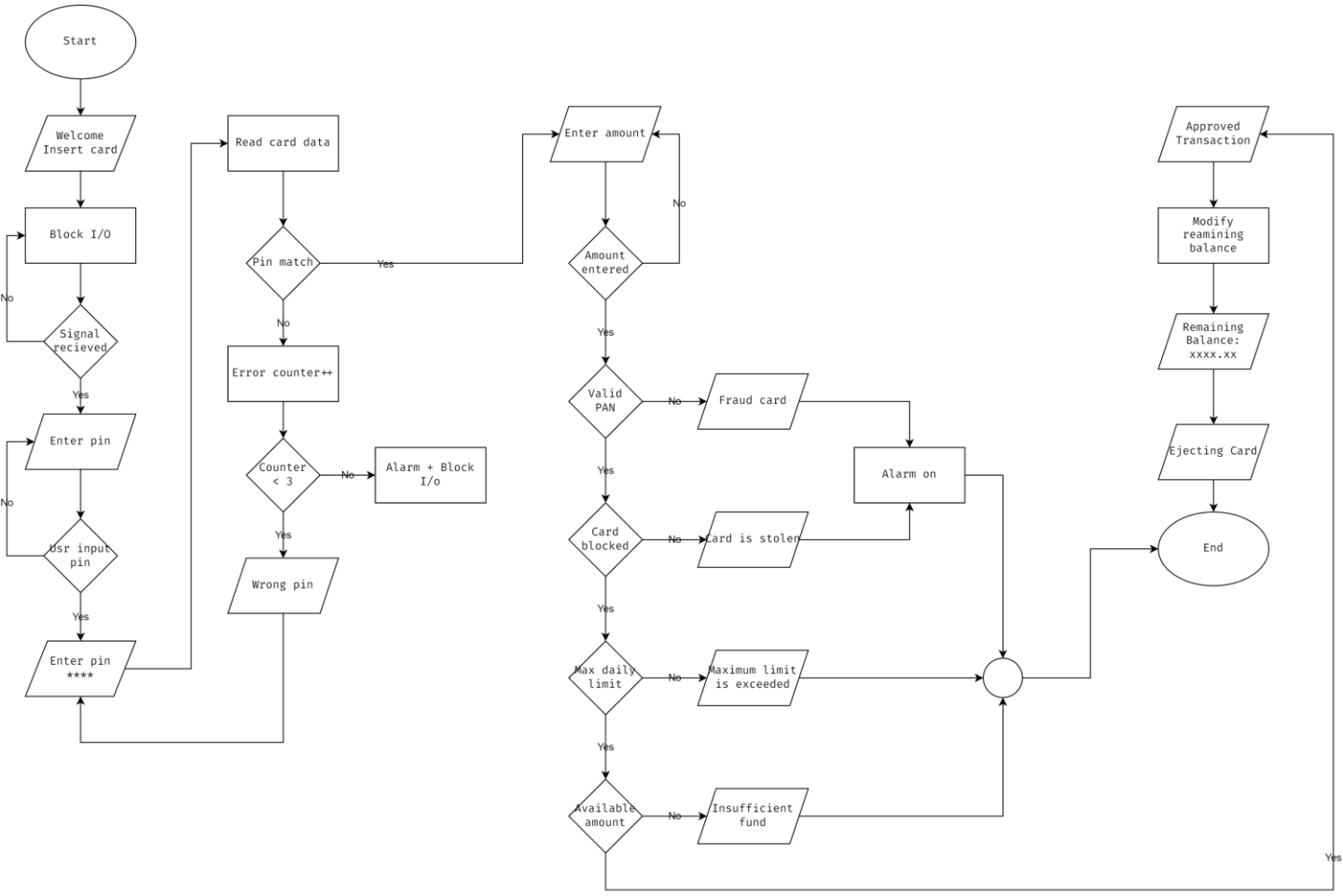


ATM MCU
Wiring diagram

ATM ECU High Level Design

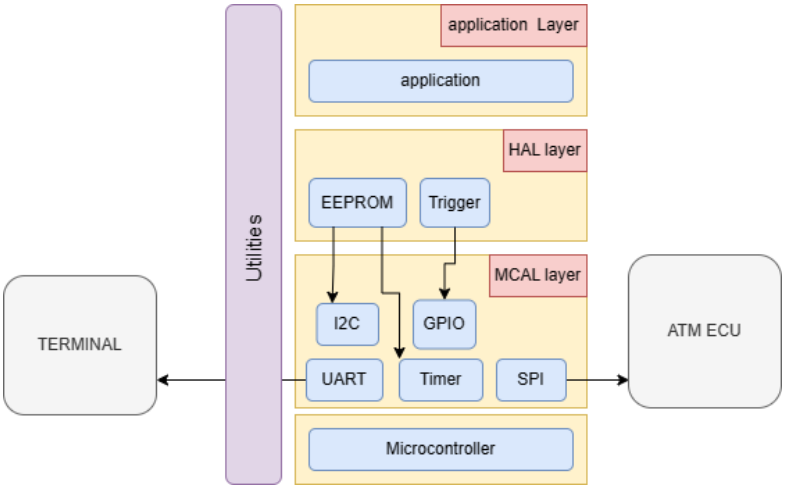
- Layered architecture
- Modules Description
- Driver’s documentation

ATM ECU flowchart Design



CARD ECU High Level Design

- Layered architecture



- Modules Description

MCAL modules

GPIO

Using GPIO for initialize trigger function and apply trigger signal (rising – falling) edge to a specific pin

I2C

Use I2C for communication with EEPROM, module should have APIs to initialize the protocol, send and receive data

UART

Use UART for communication with Terminal, module should have APIs to initialize the protocol, send and receive data

SPI

Use SPI for communication with ATM ECU, module should have APIs to initialize the protocol, send and receive data

TIMER

Use TIMER for different delays with EEPROM read/write

HAL modules

EEPROM

Used for store data such as PIN and PAN numbers

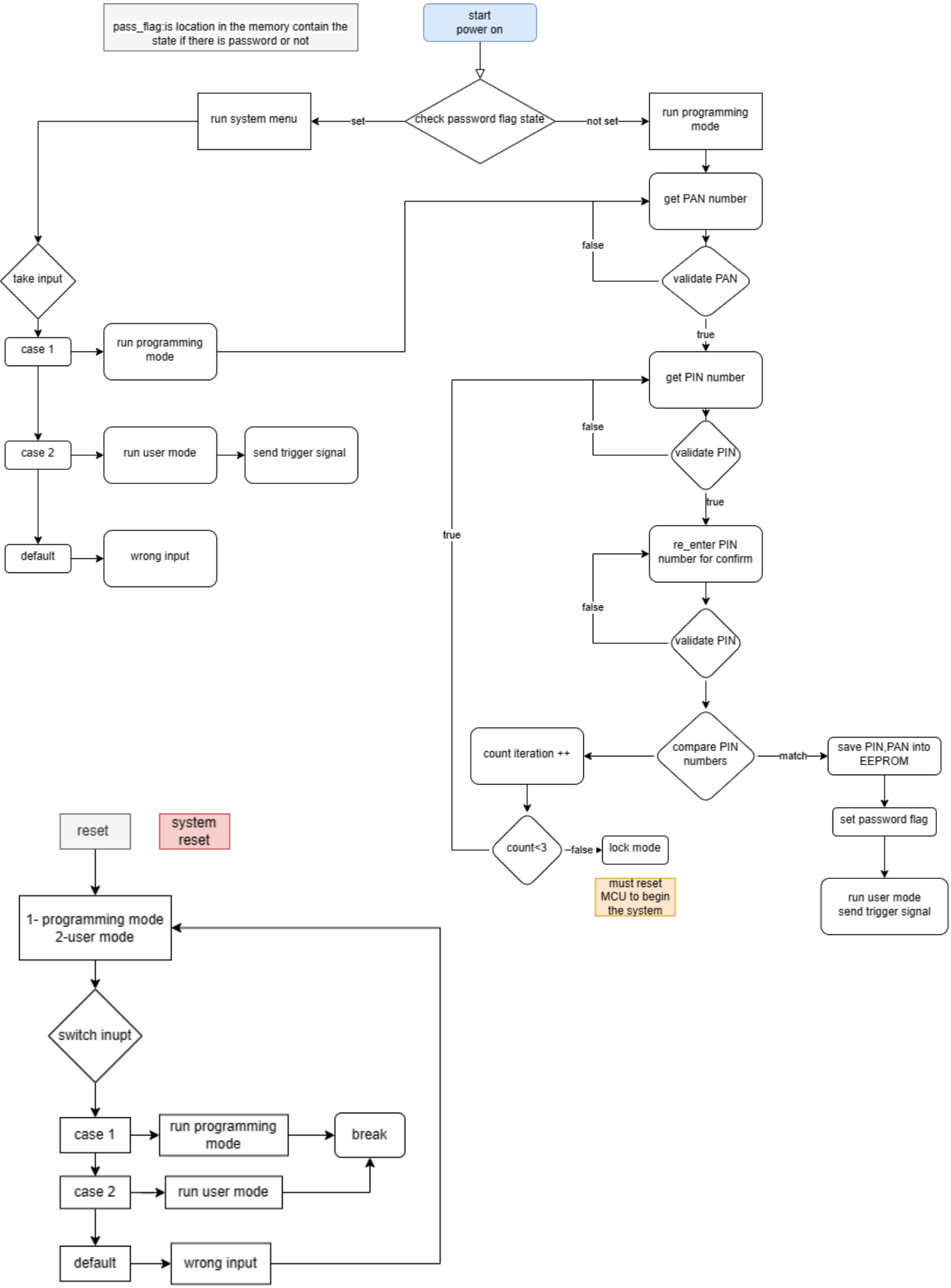
TERMINIAL

Used as interface screen with CARD ECU. Used to display and receive data from PC

Trigger

Use trigger function to send trigger edge to the ATM ECU

CARD ECU flowchart Design



CARD ECU APPLICATION APIs

void APP_check_PAN_number(uint8 arr[],uint8* a_size,uint8* str,uint8 min,uint8 max)

Description

Used to get PAN number from terminal using UART send/receive

Inputs:

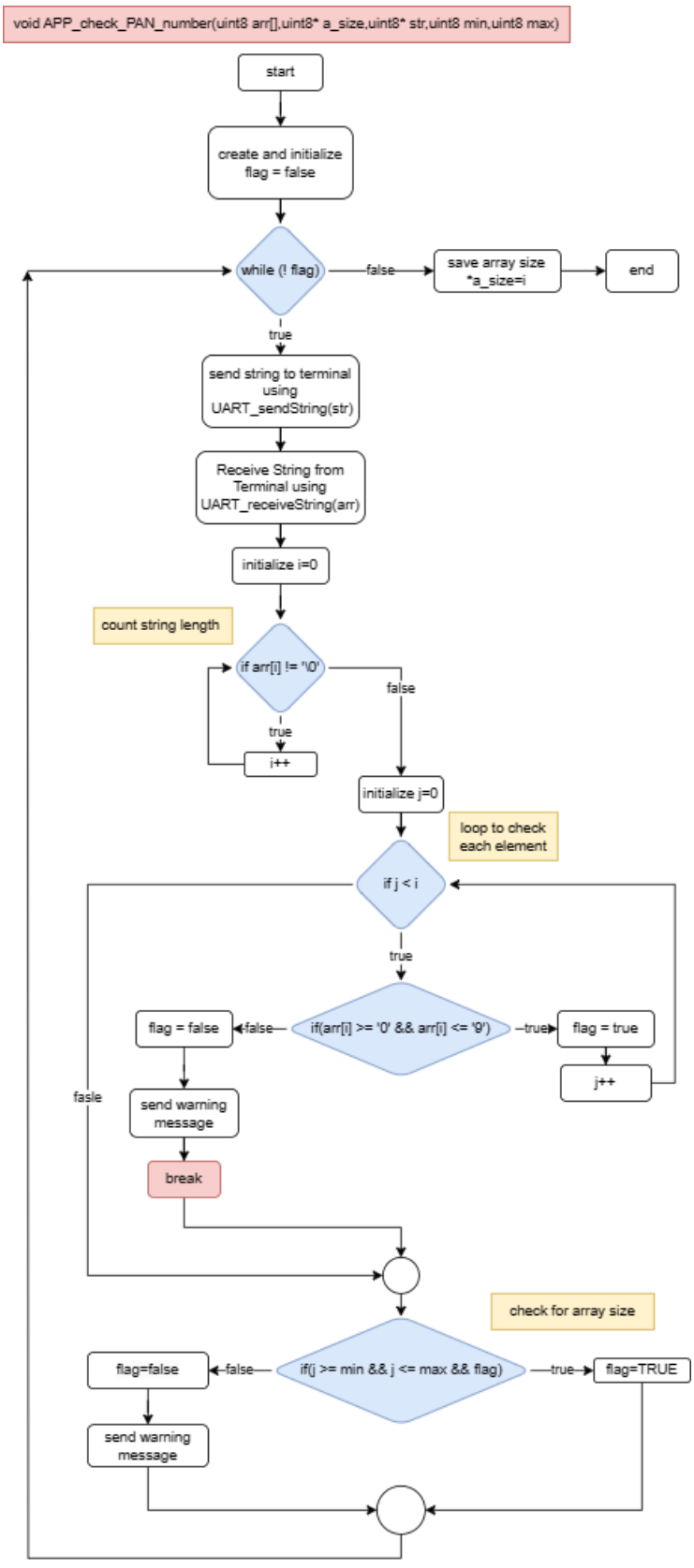
arr: array to save PAN number

a_size: size of the array

str: string to display into terminal using UART_sendString

min: minimum size of the PAN number

max maximum size of the PAN number



CARD ECU APPLICATION APIs

void APP_check_PIN_number(uint8 arr[],uint8* a_size,uint8* str,uint8 max)

Description

Used to get PAN number from terminal using UART send/receive

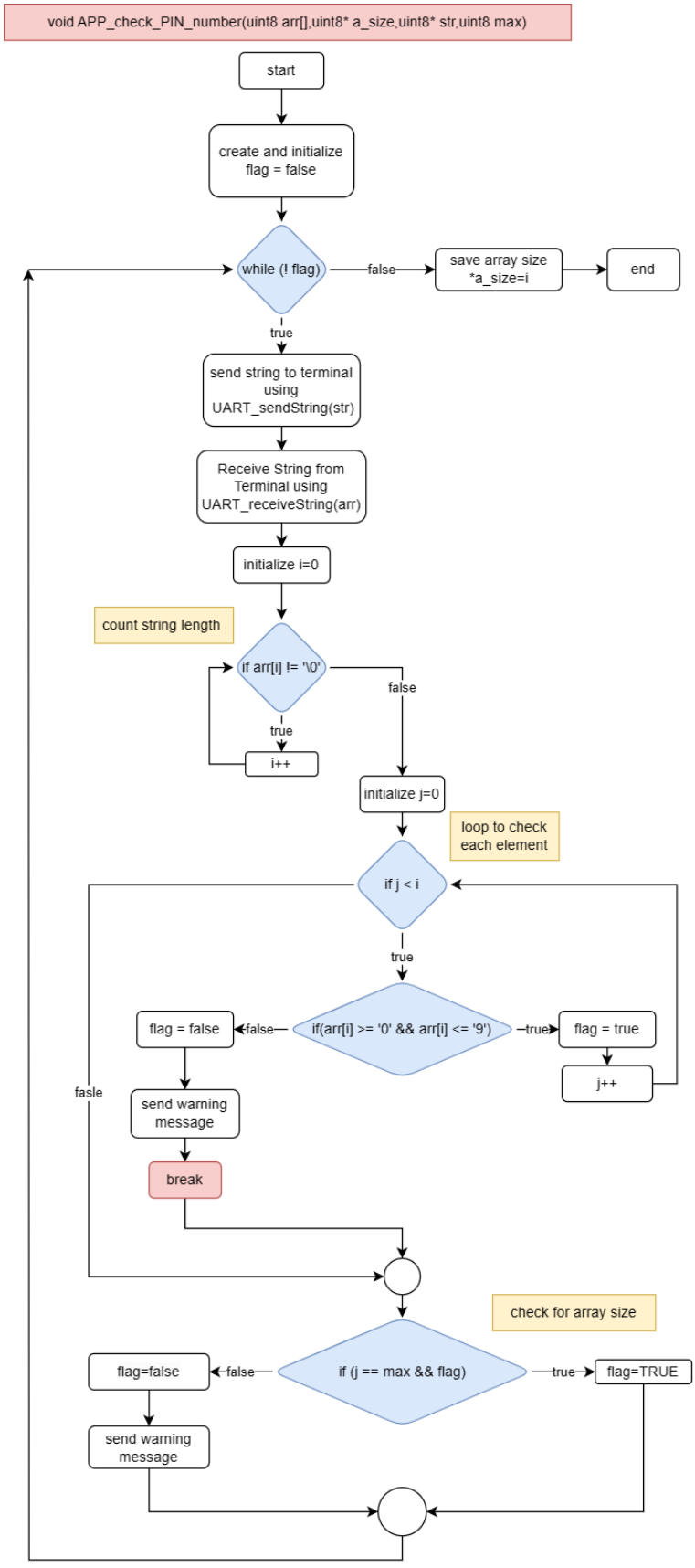
Inputs:

arr: array to save PIN number

a_size: size of the array

str: string to display into terminal using UART_sendString

max maximum size of the PIN number



CARD ECU APPLICATION APIs

uint8 APP_compare_PIN (uint8* pin_1,uint8* pin_2,uint8 pin_size)

Description

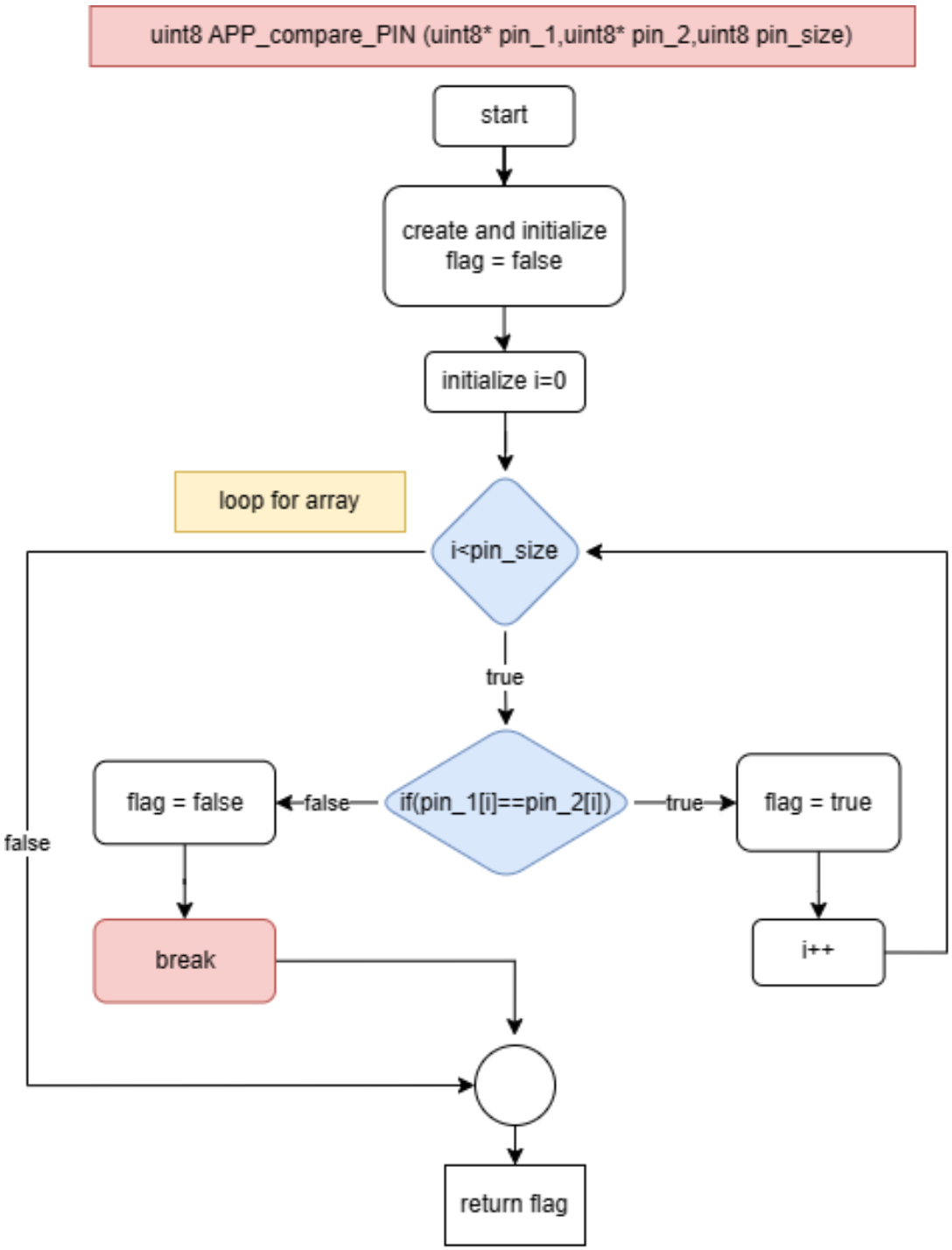
used to compare PIN numbers

Inputs:

pin_1:first pin number **pin_2**:second pin number **pin_size**: size of the PIN number

return:

TRUE: compare match **FALSE**: compare not match



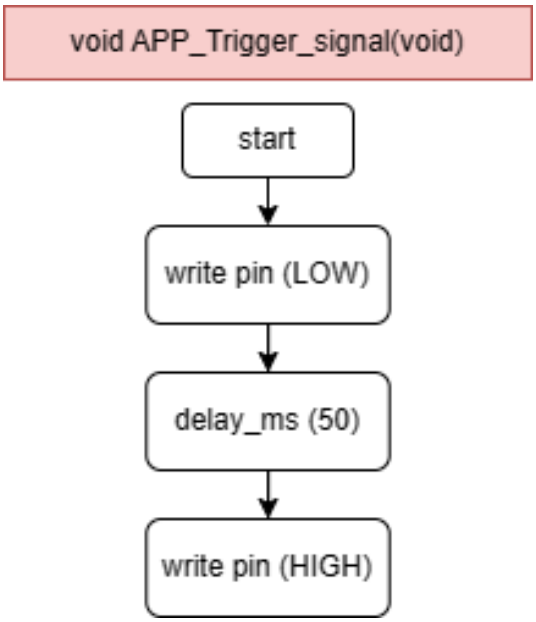
CARD ECU APPLICATION APIs

void APP_Trigger_signal(void)

Description

trigger signal low to high (rising edge)

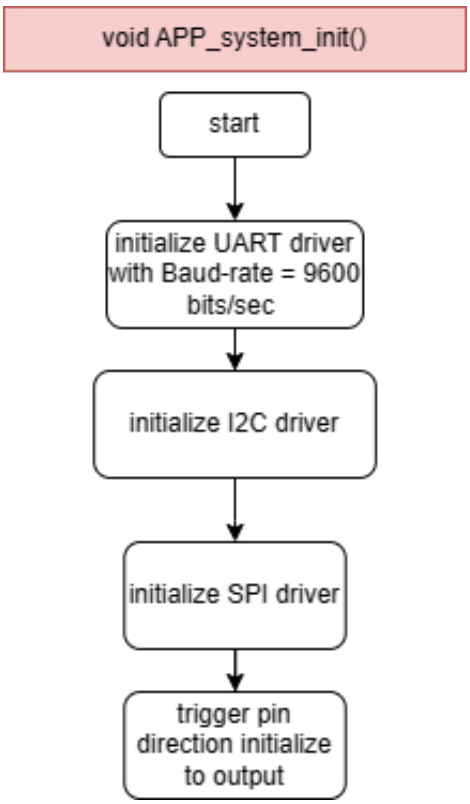
Inputs: void **return: void**



void APP_system_init(void)

Description

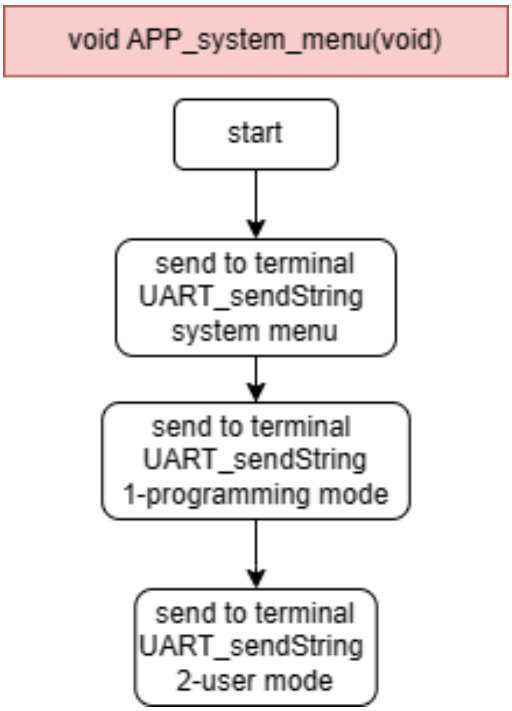
Used to initialize system peripherals and communications



void APP_system_menu(void)

Description

Used to display user main options

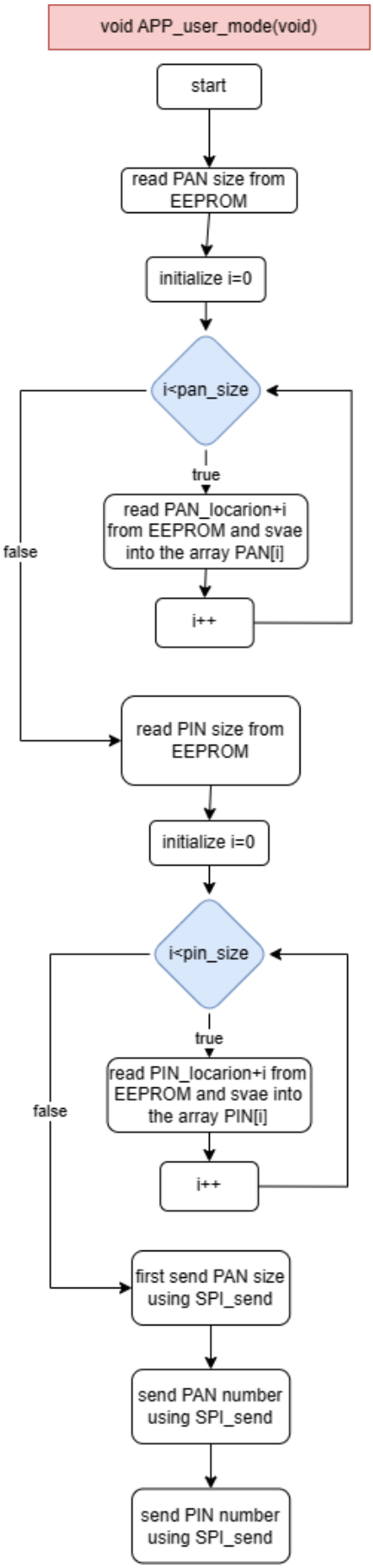


CARD ECU APPLICATION APIs

void APP_user_mode(void)

Description

Used to run system in USER mode, read saved data from EEPROM and send PIN and PAN to the ATM MCU



Drivers Documentation 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;
```

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

Drivers Documentation Project Modules APIs

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**

Drivers Documentation Project Modules APIs

- **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**

Drivers Documentation 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:

/*============= 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
#define TOV2  6
#define OCF2  7
```

Drivers Documentation 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
```

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

Drivers Documentation Project Modules APIs

- **Timer0_delay module APIs** **(TIMER0_Utilities.h)**

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

Drivers Documentation Project Modules APIs

- **TWI module APIs**

/*============= MACRO DEFINITIONS=============*/

/* I2C Status Bits in the TWSR Register */

```
/* start has been sent */
#define TWI_START         0x08
/* repeated start */
#define TWI_REP_START     0x10
/* Master transmit ( slave address + Write request ) to slave + ACK received from slave. */
#define TWI_MT_SLA_W_ACK  0x18
/* Master transmit ( slave address + Read request ) to slave + ACK received from slave. */
#define TWI_MT_SLA_R_ACK  0x40
/* Master transmit data and ACK has been received from Slave. */
#define TWI_MT_DATA_ACK   0x28
/* Master received data and send ACK to slave. */
#define TWI_MR_DATA_ACK   0x50
/* Master received data but doesn't send ACK to slave. */
#define TWI_MR_DATA_NACK  0x58
```

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

void TWI_init(void);

Description:

Used for initialize Baud rate and set slave address in case work in slave mode and enable I2C for communication

void TWI_start(void);

Description:

Used to send start bit and wait until start bit is sent successfully

void TWI_stop(void);

Description

Used to send stop bit

void TWI_writeByte(uint8 data);

Description

Used to send data by putting data in TWI data register and wait for TWINT flag set in TWCR register so data is sent successfully

Drivers Documentation Project Modules APIs

- **TWI module APIs**

uint8 TWI_readByteWithACK(void);

Description:

Used to read data and send ACK after reading or receiving data and wait for TWINT flag set in TWCR register so data is read successfully

uint8 TWI_readByteWithNACK(void);

Description:

Used to read data and send NACK after reading or receiving data and wait for TWINT flag set in TWCR register so data is read successfully

uint8 TWI_getStatus(void);

Description:

Used to get status after any read or write operation by masking to eliminate first 3 bits and get the last 5 bits (status bits)

Drivers Documentation Project Modules APIs

- **EEPROM module APIs**

```
/*============= MACRO DEFINITIONS=============*/
#define ERROR       0
#define E_SUCCESS 1

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

uint8 EEPROM_writeByte(uint32 u16addr,uint8 u8data);

Description:

Used to write byte into the EEPROM by sending the frame

- Send the Start Bit
- Send the device address, we need to get A8 A9 A10 address bits from the memory location address and R/W=0 (write)
- Send the required memory location address
- Send the required memory location address
- write byte to EEPROM
- Send the Stop Bit

uint8 EEPROM_readByte(uint32 u16addr,uint8 *u8data);

Description:

Used to Read byte from the EEPROM by sending the frame

- Send the Start Bit
- Send the device address, we need to get A8 A9 A10 address bits from the memory location address and R/W=0 (write)
- Send the required memory location address
- Send the required memory location address
- Send the Repeated Start Bit
- Send the device address, we need to get A8 A9 A10 address bits from the memory location address and R/W=1 (Read)
- Read Byte from Memory without send ACK
- Send the Stop Bit

Drivers Documentation Project Modules APIs

- **UART module APIs**

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

void UART_init(uint32 baud_rate);

Description :

Functional responsible for Initialize the UART device by:

- Setup the Frame format like number of data bits, parity bit type and number of stop bits.
- Enable the UART.
- Setup the UART baud rate.

void UART_sendByte(const uint8 data);

Description :

Functional responsible for send byte to another UART device.

uint8 UART_recieveByte(void);

Description :

Functional responsible for receive byte from another UART device.

void UART_sendString(const uint8 *Str);

Description :

Send the required string through UART to the other UART device.

void UART_receiveString(uint8 *Str);

Description :

Receive the required string until the '#' symbol through UART from the other UART device. Receive until ‘#’

void UART_integerTostring(const uint8 data);

Description :

Display the required decimal value on the screen

Drivers Documentation Project Modules APIs

- **SPI module APIs**

/*============= MACRO DEFINITIONS=============*/

#define SPI_DEFAULT_DATA_VALUE 0xFF
Description :
Default SPI data value used in case we need to receive a byte from the other device, without need to send a data to it

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

void SPI_initMaster(void);
Description :
Initialize the SPI device as Master.

void SPI_initSlave(void);
Description :
Initialize the SPI device as Slave.

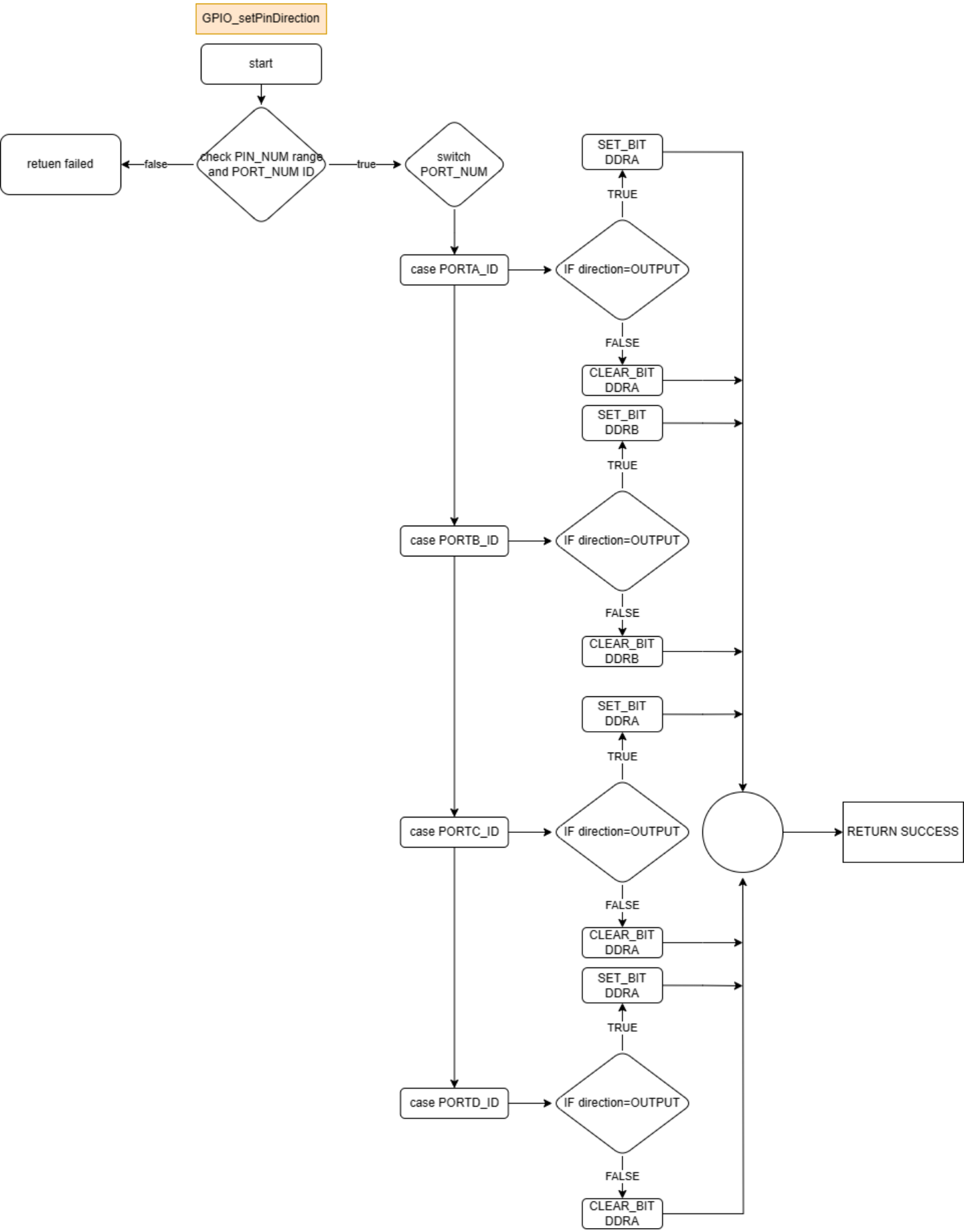
uint8 SPI_sendReceiveByte(uint8 data);
Description :
Send the required data through SPI to the other SPI device.
In the same time data will be received from the other device.

void SPI_sendString(const uint8 *str);
Description :
Send the required string through SPI to the other SPI device.

void SPI_receiveString(uint8 *str);
Description :
Receive the required string until the '#' symbol through SPI from the other SPI device.

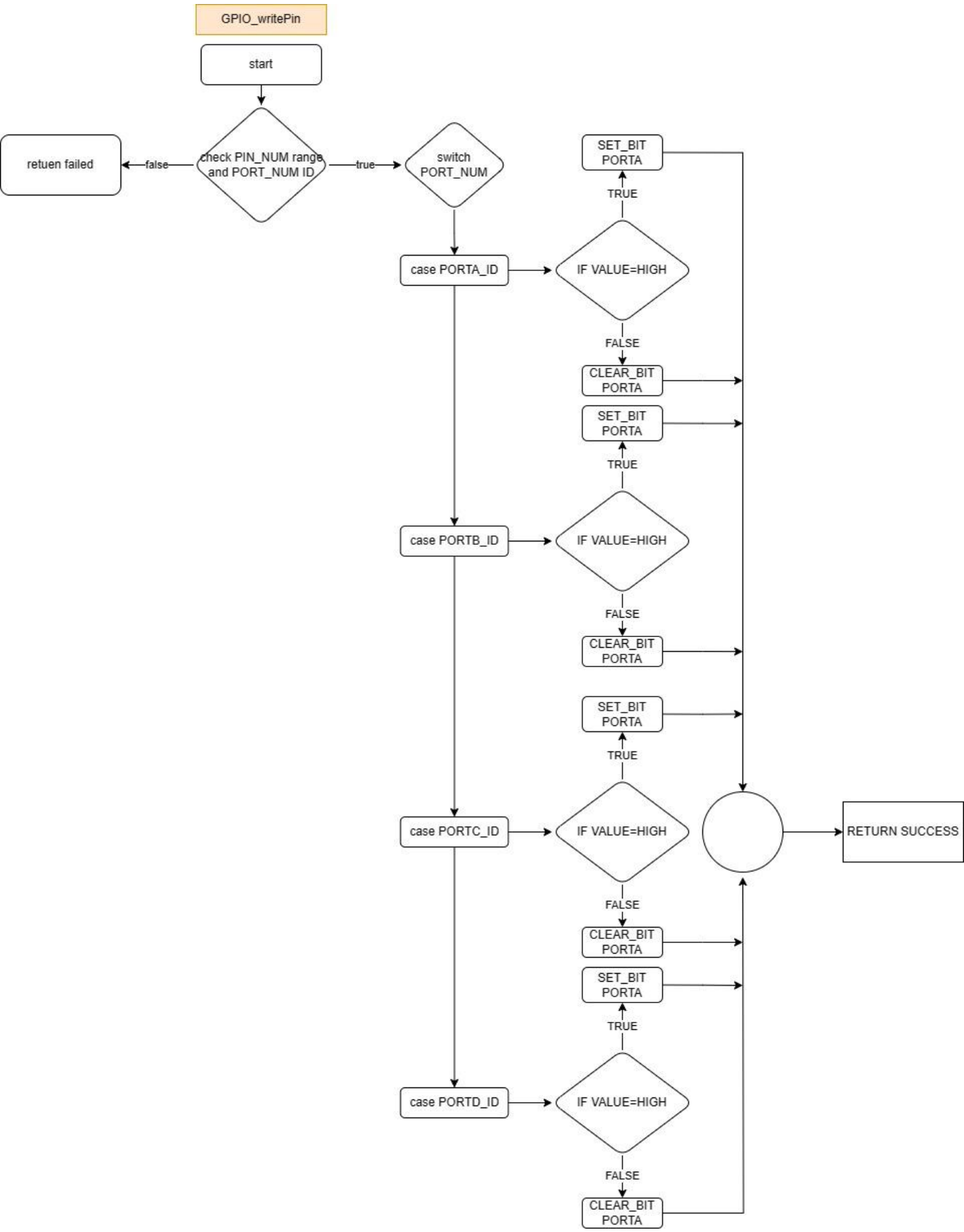
GPIO APIs flowchart

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



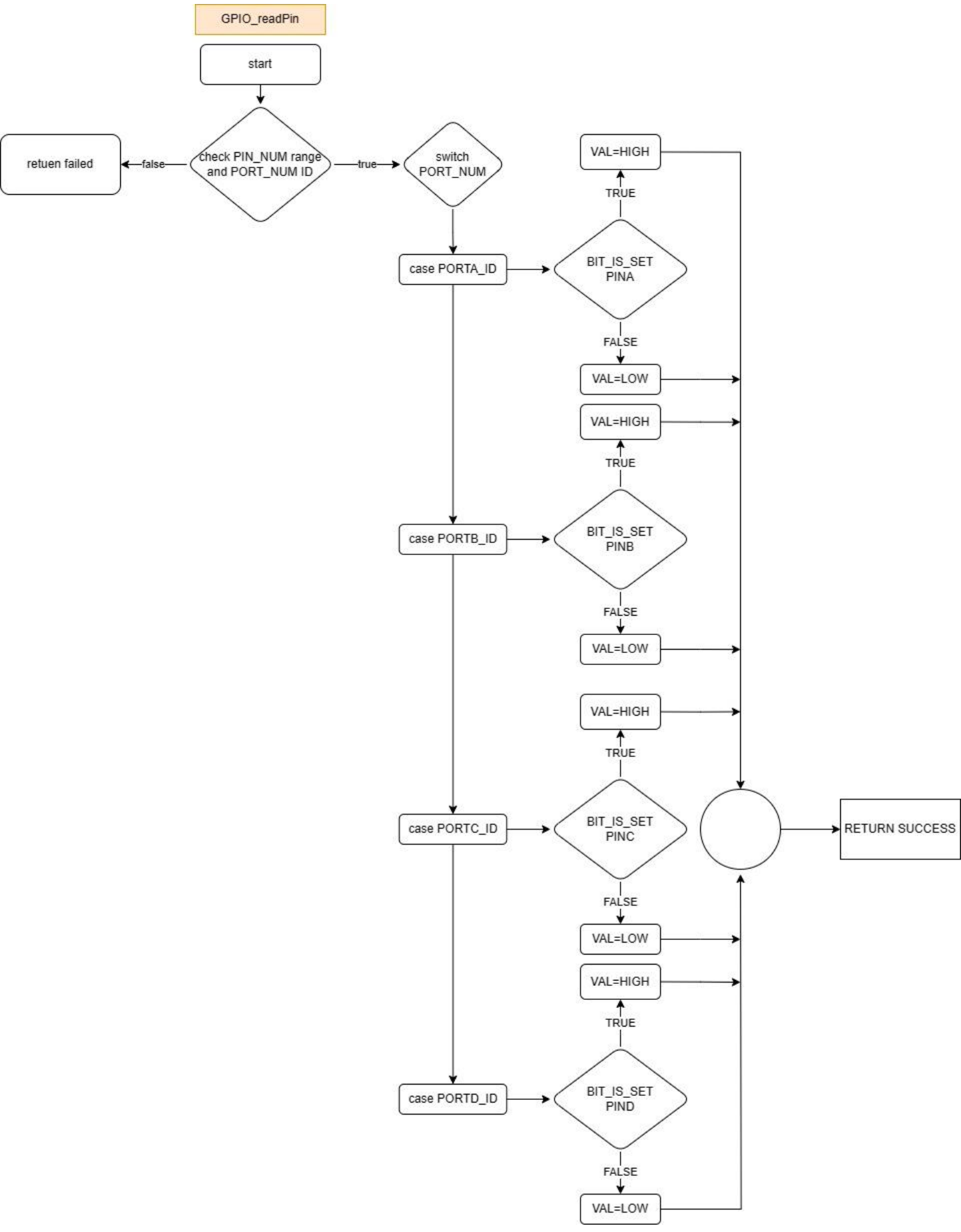
GPIO APIs flowchart

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



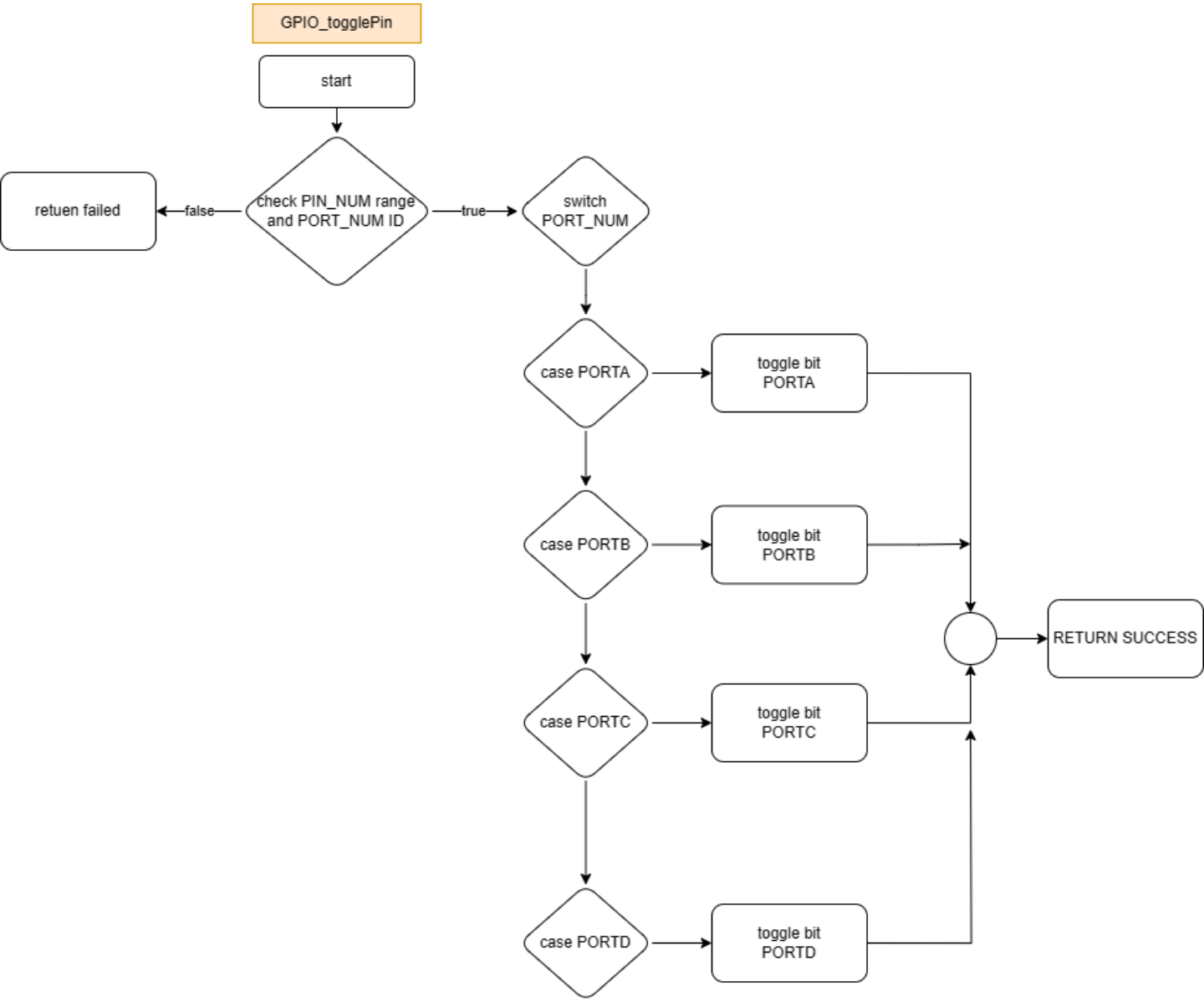
GPIO APIs flowchart

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



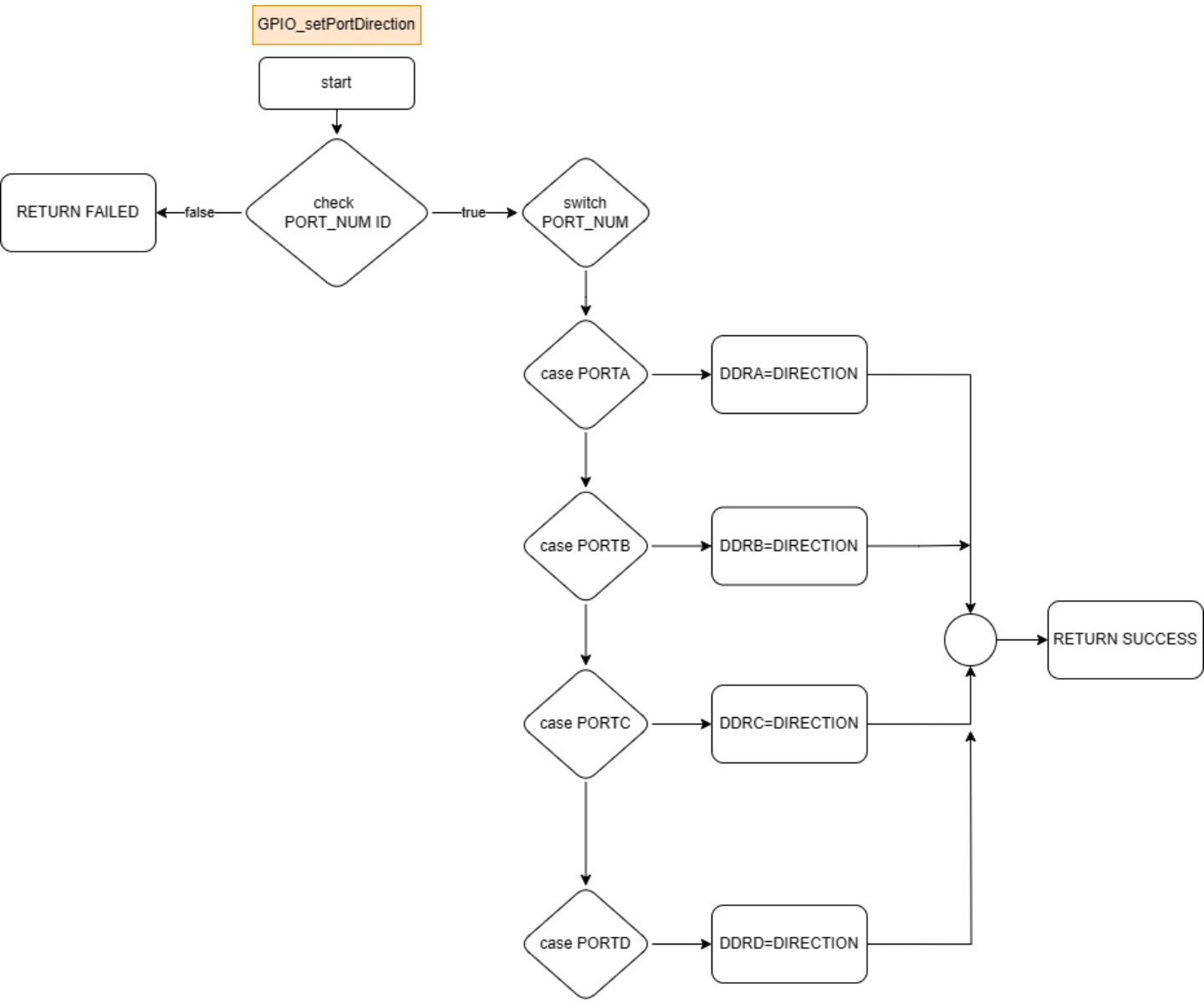
GPIO APIs flowchart

EN_STATE GPIO_togglePin(uint8 port_num, uint8 pin_num);



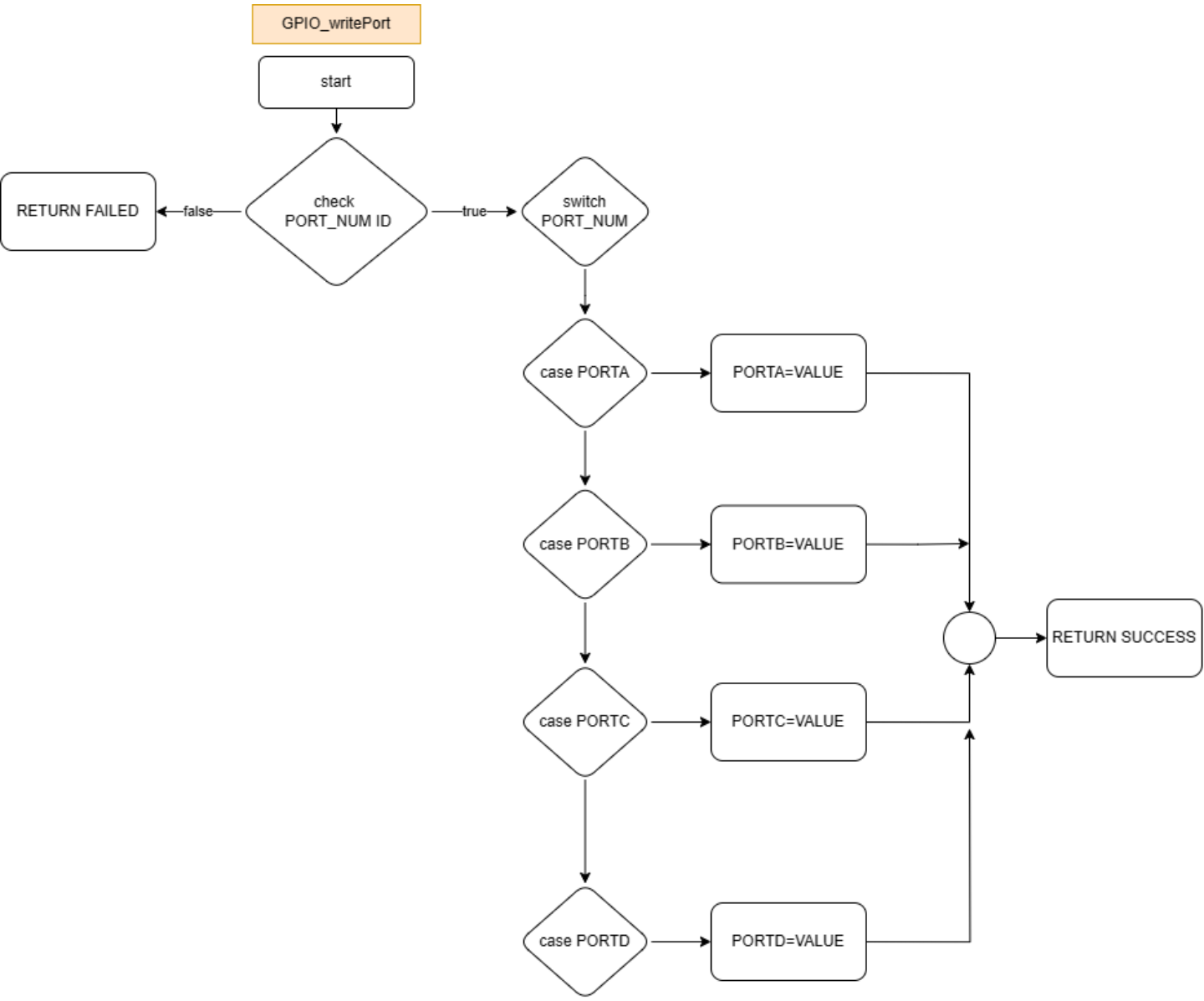
GPIO APIs flowchart

EN_STATE GPIO_setPortDirection(uint8 port_num, EN_PORT_DIRECTION direction);



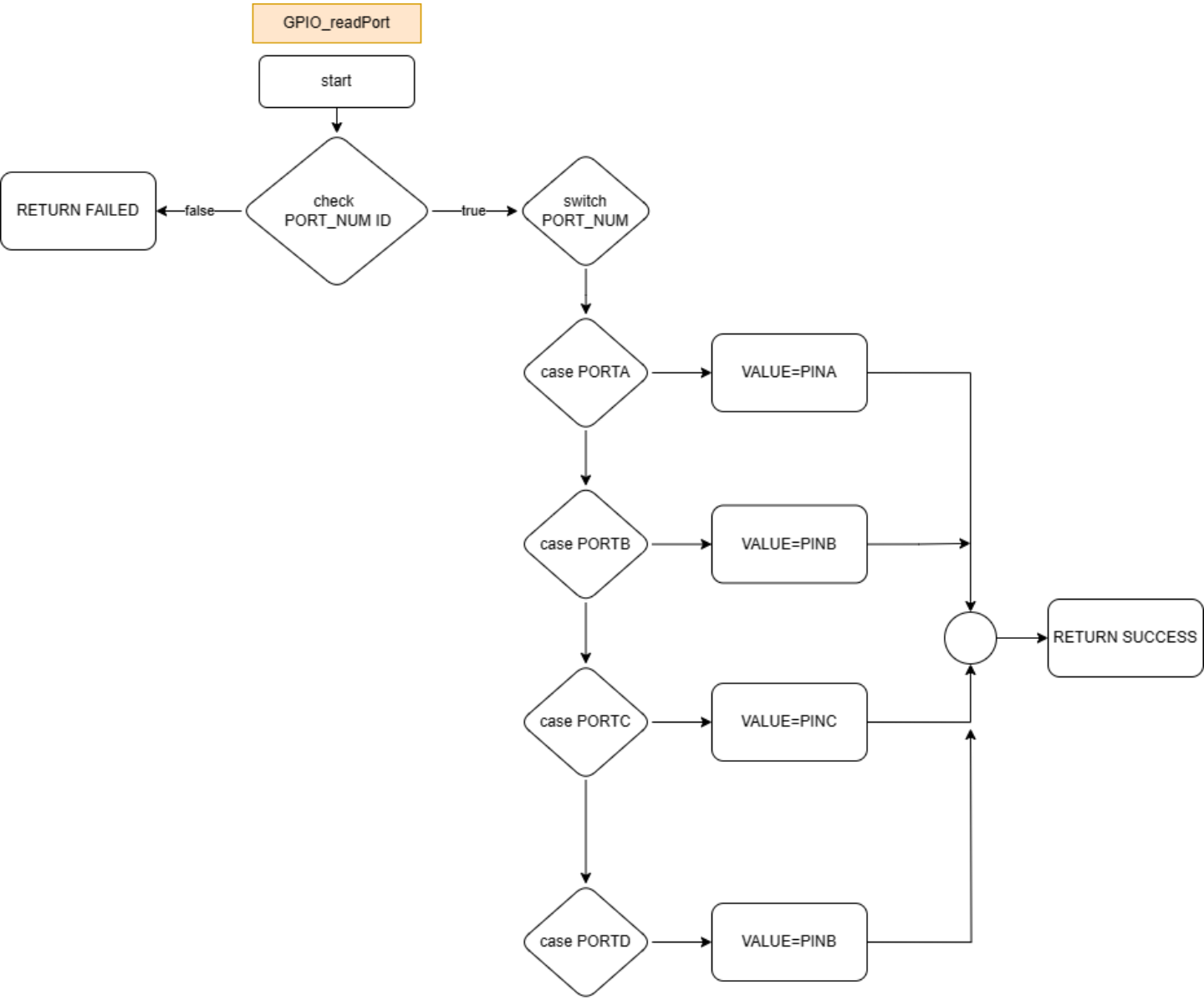
GPIO APIs flowchart

EN_STATE GPIO_writePort(uint8 port_num, uint8 value);

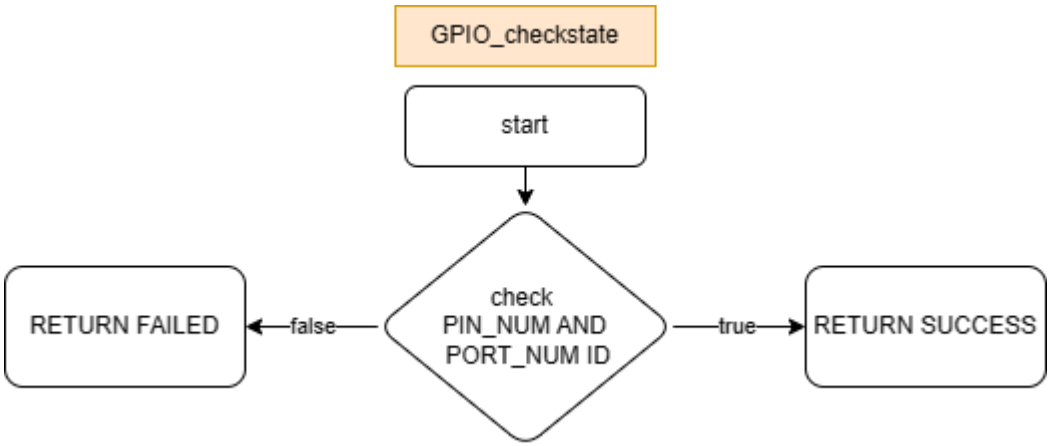


GPIO APIs flowchart

EN_STATE GPIO_readPort(uint8 port_num,uint8* value);

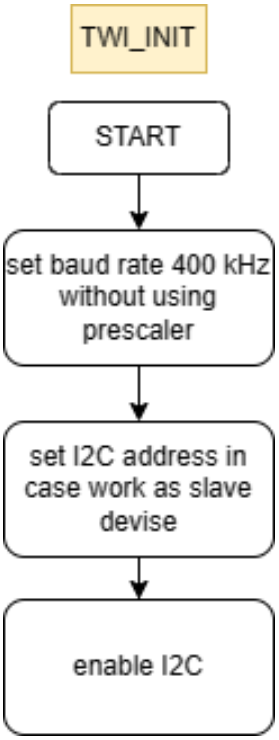


EN_STATE GPIO_checkstate(uint8 port_num,uint8 pin_num);

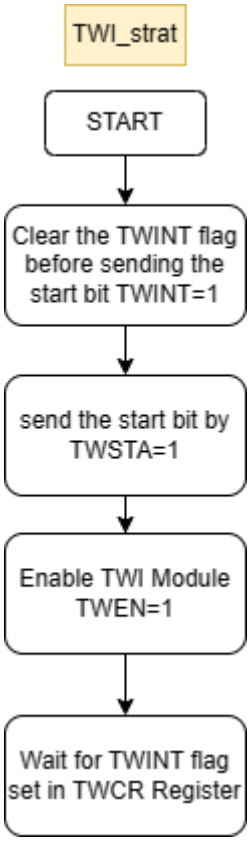


I2C APIs flowchart

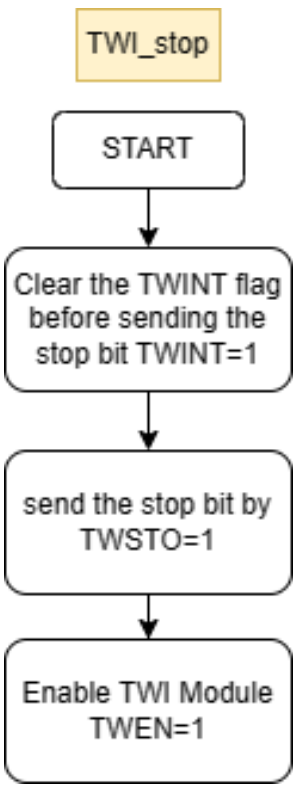
void TWI_init(void);



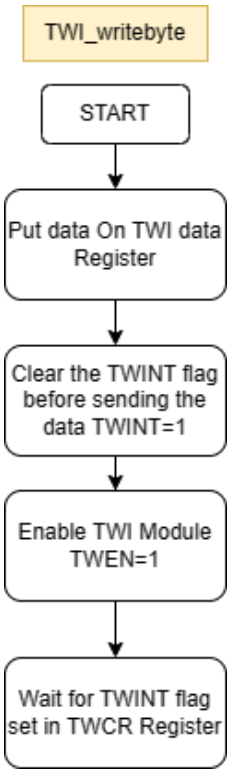
void TWI_start(void);



void TWI_stop(void);

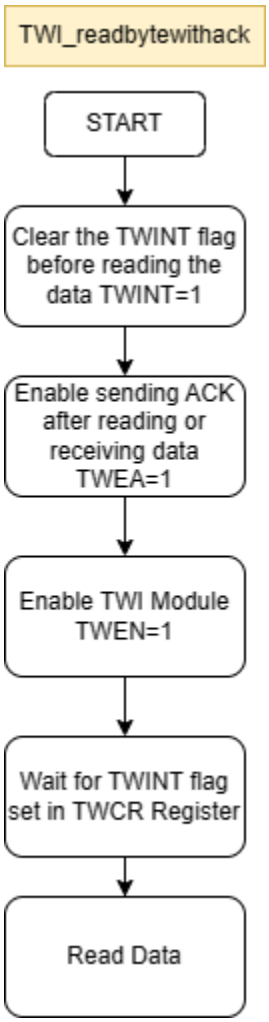


void TWI_writeByte(uint8 data)

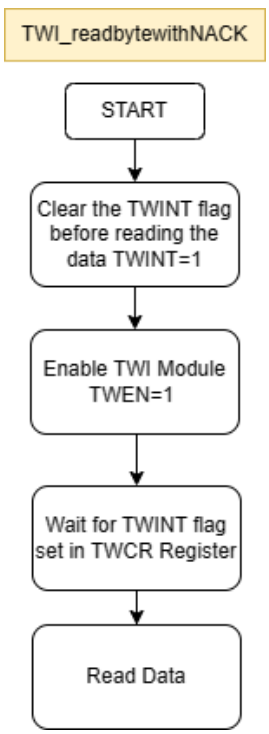


I2C APIs flowchart

uint8 TWI_readByteWithACK(void)

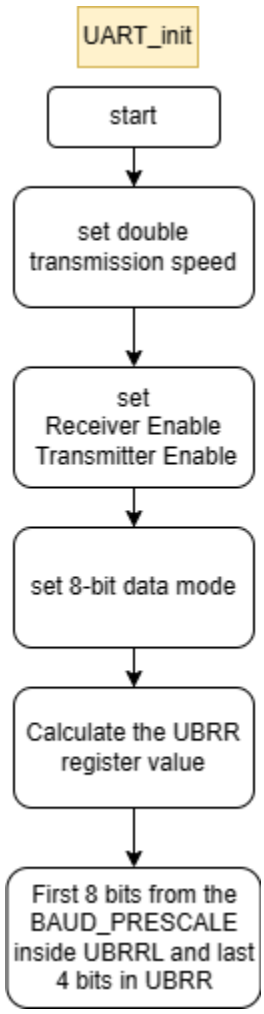


uint8 TWI_readByteWithNACK(void)

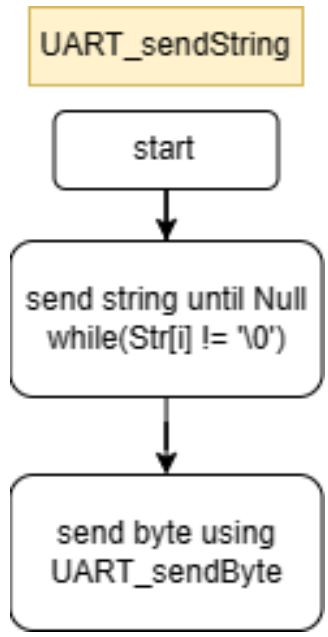


UART APIs flowchart

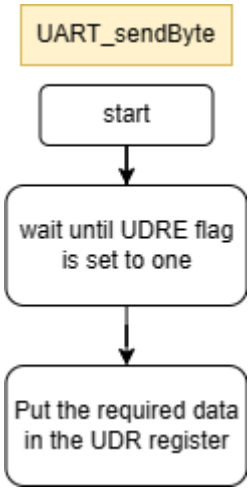
void UART_init(uint32 baud_rate)



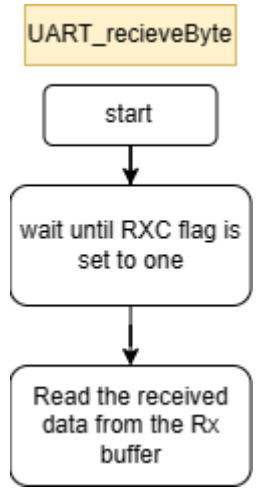
void UART_sendString(const uint8 *Str)



void UART_sendByte(const uint8 data)

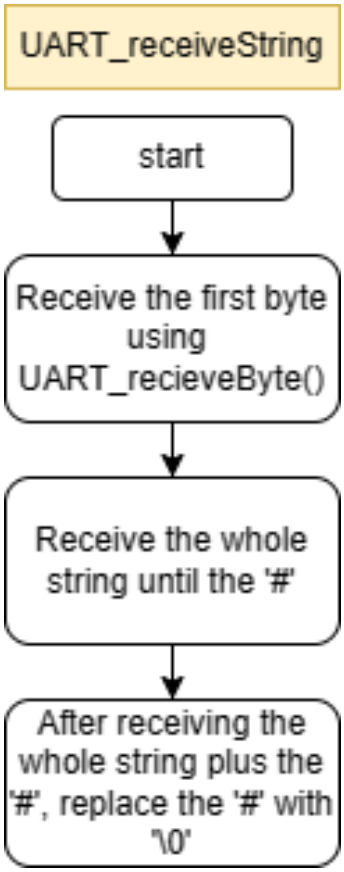


uint8 UART_recieveByte(void)

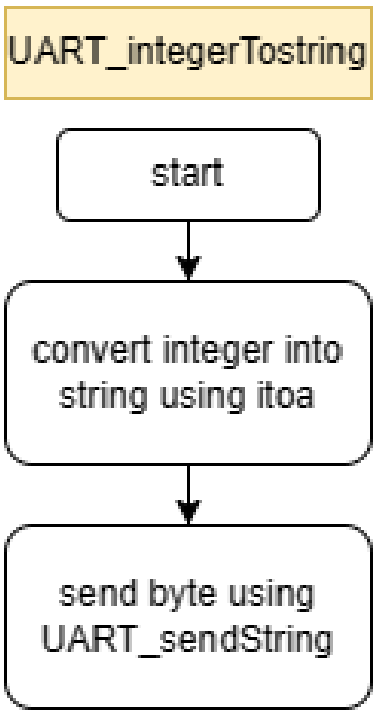


UART APIs flowchart

void UART_receiveString(uint8 *Str)

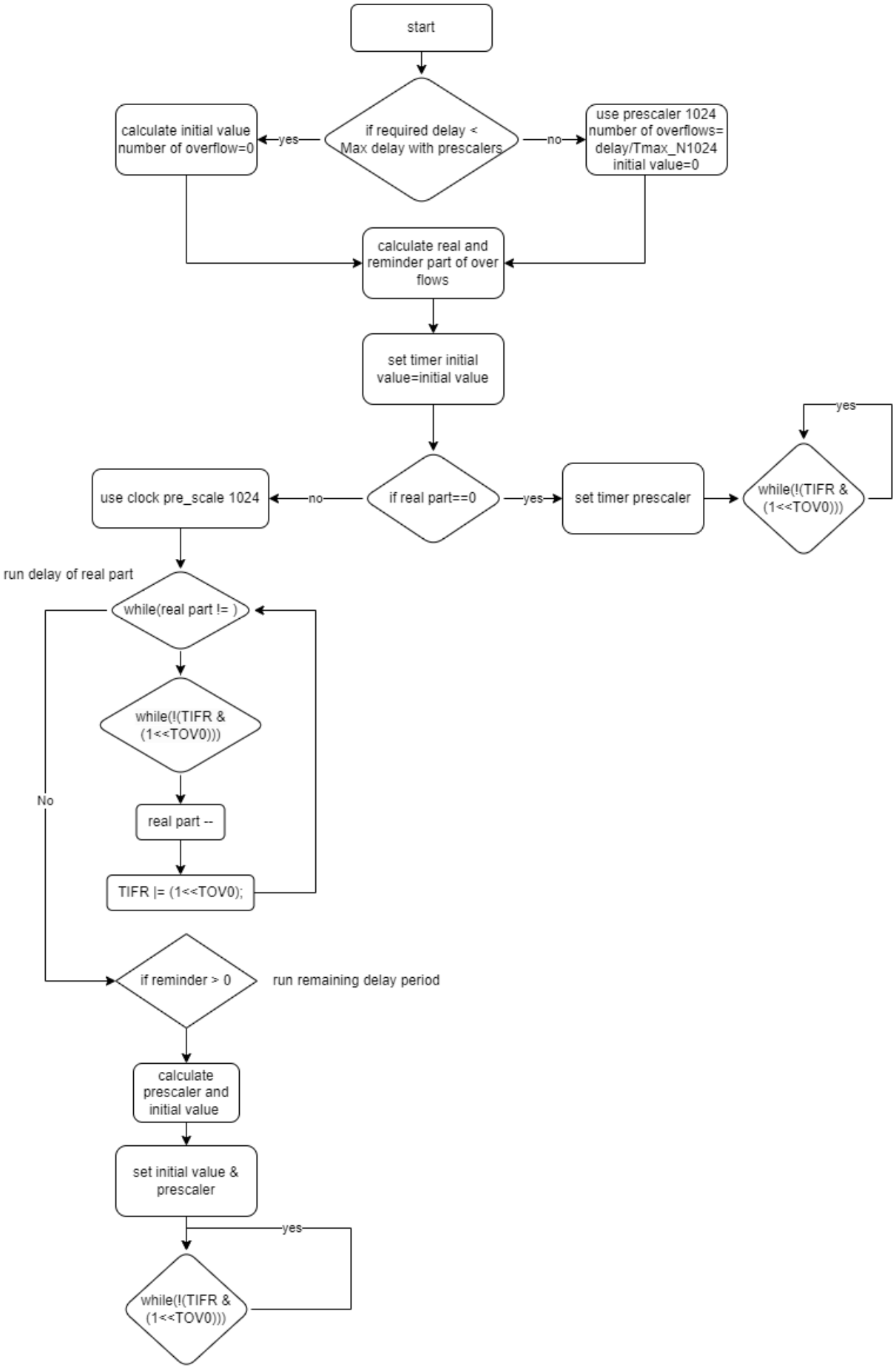


void UART_integerTostring(uint8 data)



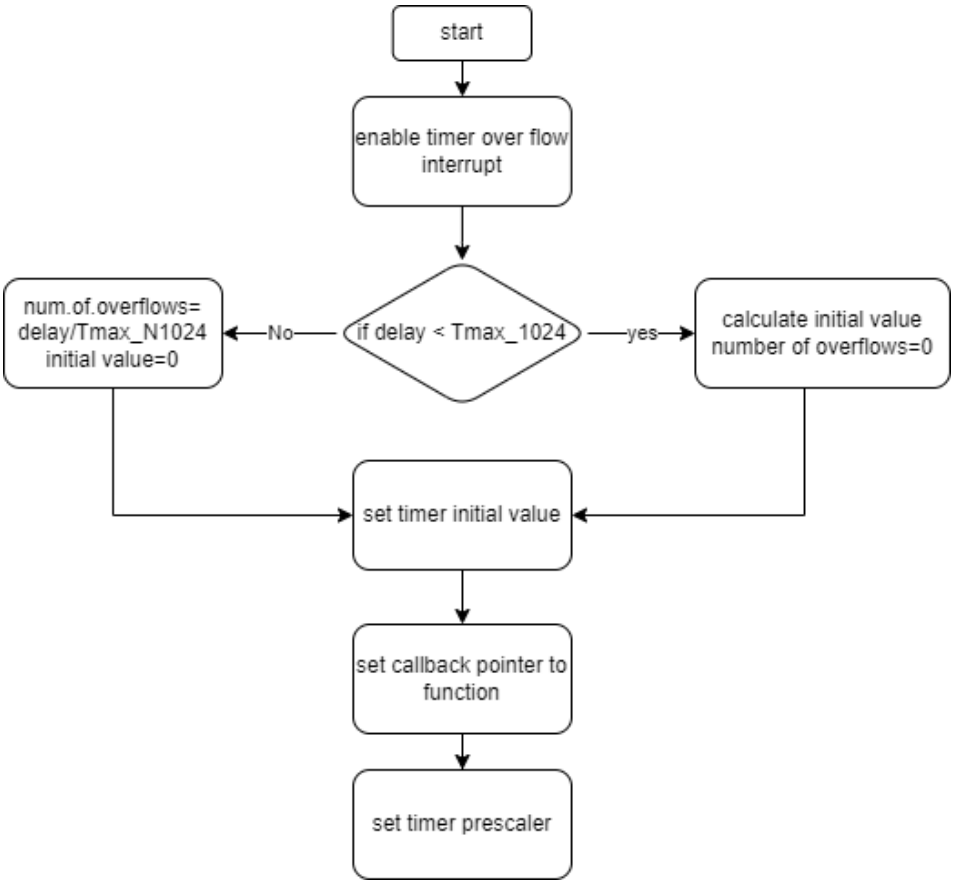
Timer0 APIs flowchart

void Timer0_Delay(float delay);

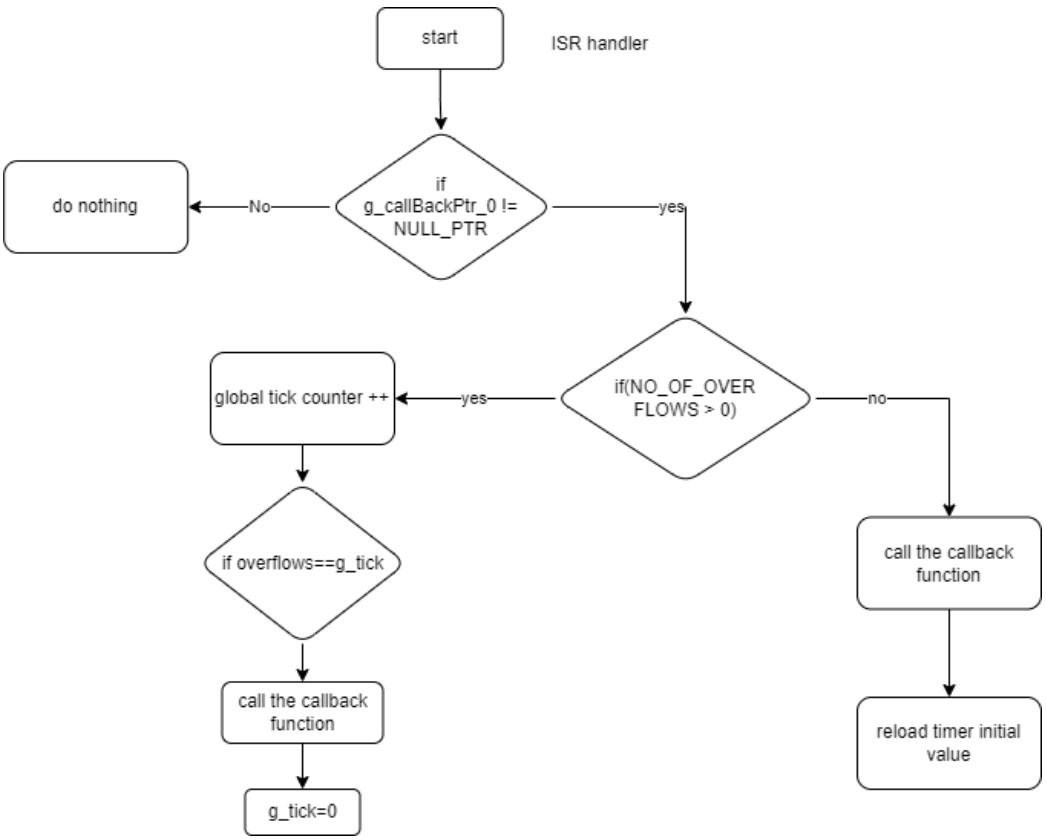


Timer0 APIs flowchart

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

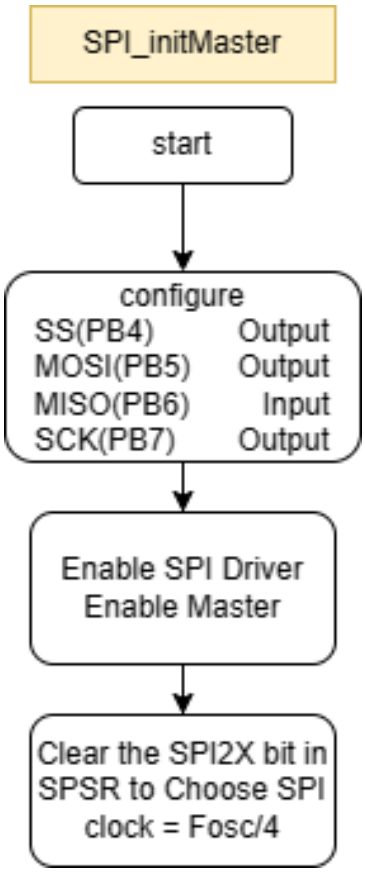


ISR (TIMER0_OVF_vect)

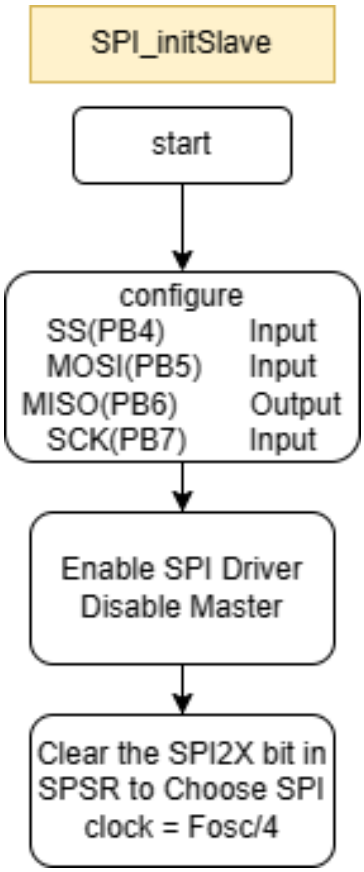


SPI APIs flowchart

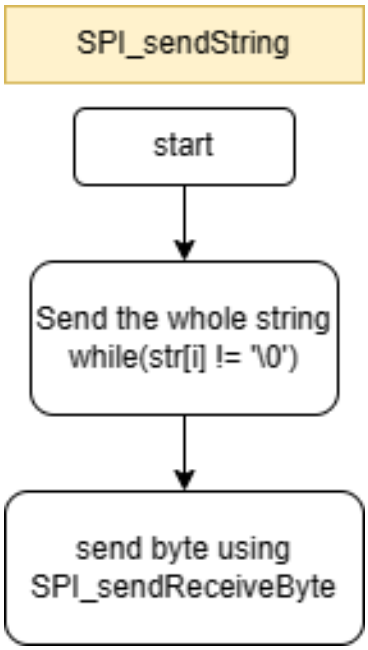
void SPI_initMaster(void)



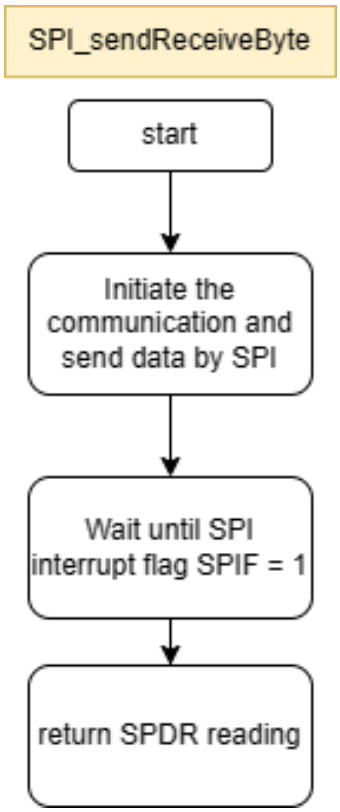
void SPI_initSlave(void)



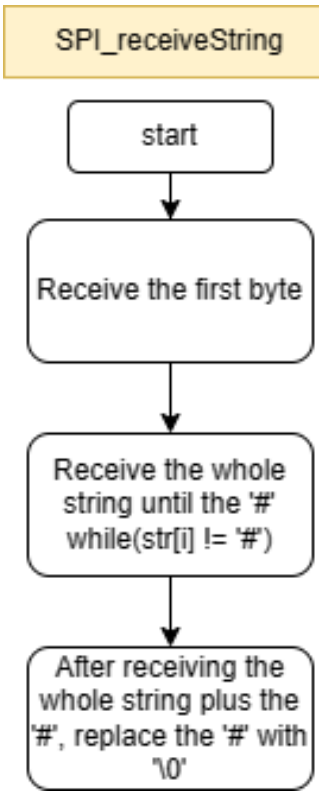
Void SPI_sendString (const uint8 *str)



uint8 SPI_sendReceiveByte (uint8 data)

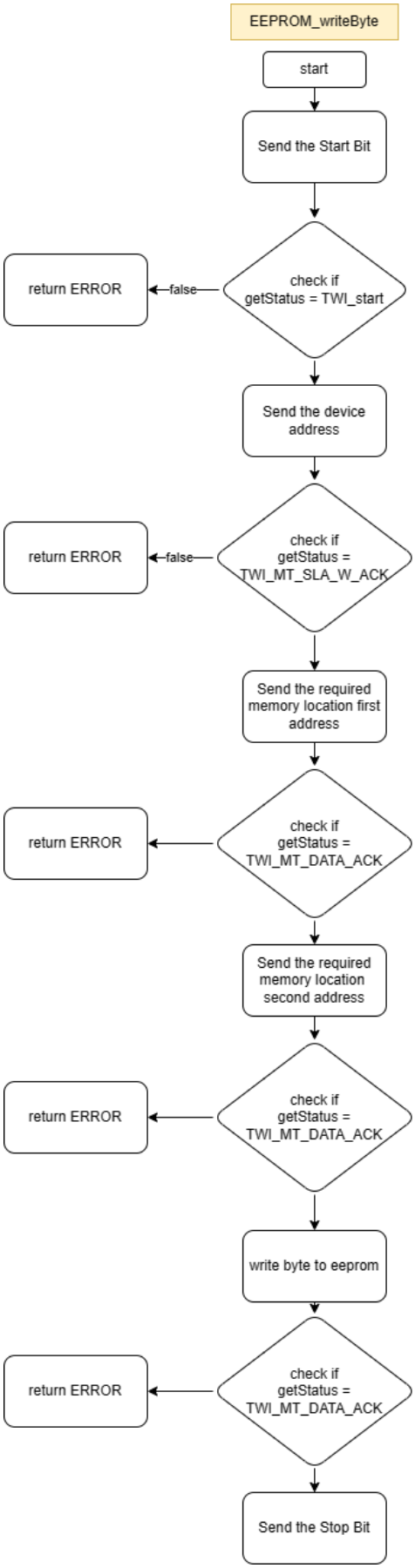


void SPI_receiveString(uint8 *str)



EEPROM APIs flowchart

uint8 EEPROM_writeByte(uint32 u16addr, uint8 u8data)



EEPROM APIs flowchart

uint8 EEPROM_readByte(uint32 u16addr, uint8 *u8data)