SIMPLE ATM MACHINE PROJECT

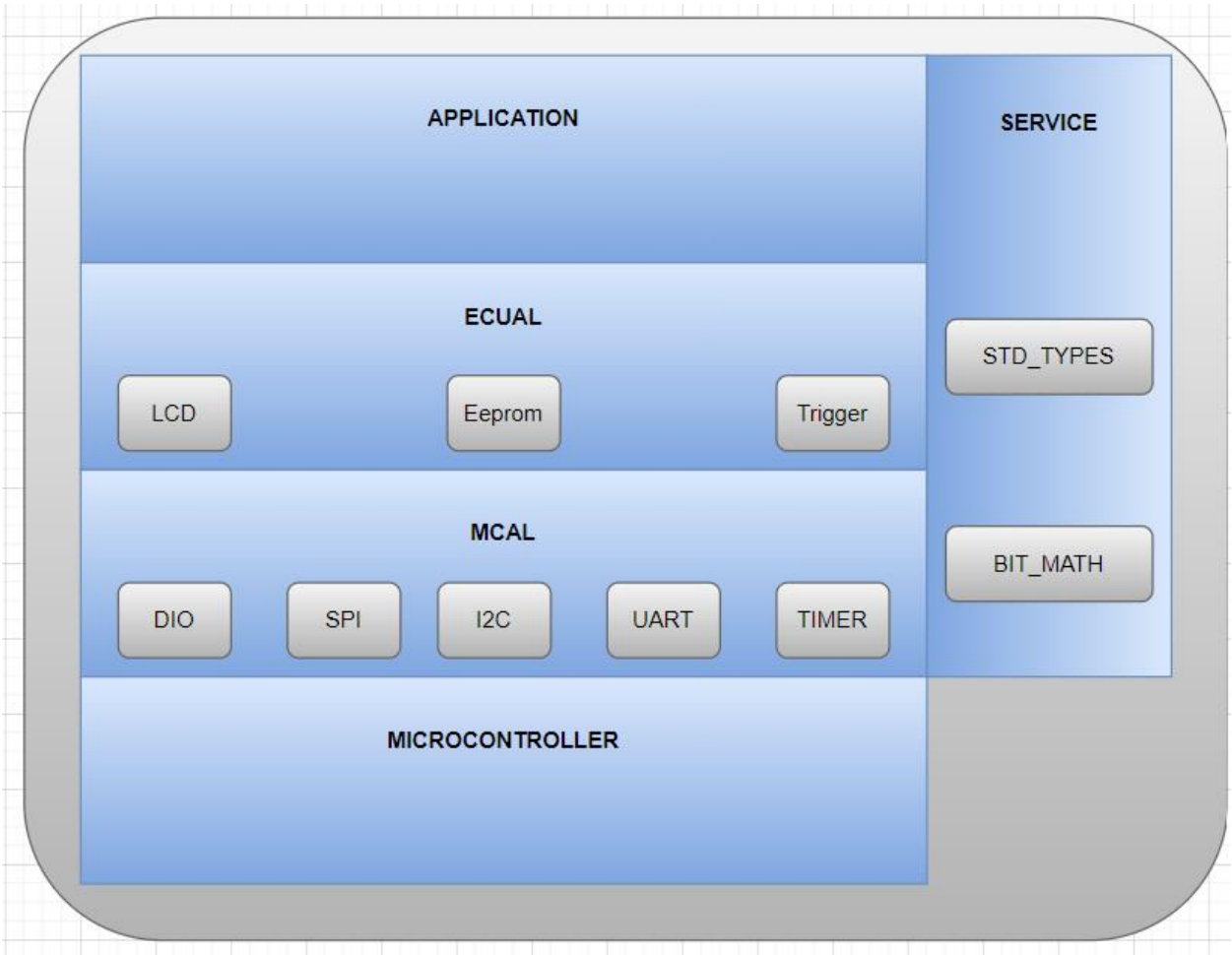
BY: Momen Hassan, Ahmed Atef, and Ahmed Mohamed Hesham

Contents

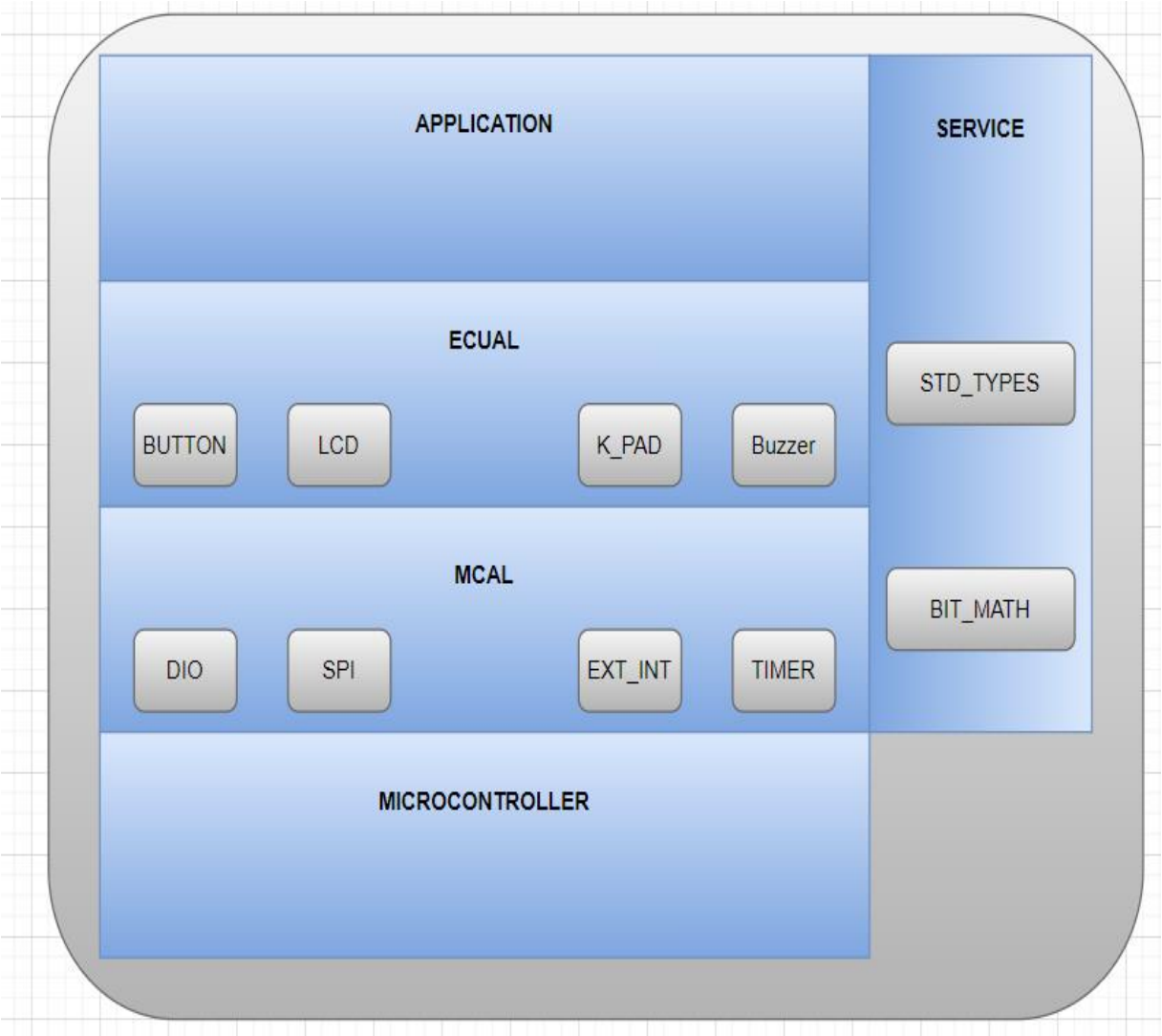
Simple Atm Machine Design

Layered Architecture

1.Card



2. ATM


APPLICATION
SERVICE
ECUAL
STD_TYPES
BUTTON
LCD
K_PAD
Buzzer
MCAL
BIT_MATH
DIO
SPI
EXT_INT
TIMER
MICROCONTROLLER

INTRODUCTION

The ATM machine project aims to provide a secure and convenient way for users to perform financial transactions such as withdrawing cash and checking their account balance. The project consists of two main components: the ATM MCU and the CARD MCU.

The ATM MCU is responsible for managing the transaction flows and user interactions. When a user inserts their card into the ATM machine, the MCU will display a welcome message and prompt the user to enter their PIN. The ATM MCU will validate the PIN by communicating with the CARD MCU and the database. If the PIN is correct, the user will be prompted to enter the amount they wish to withdraw. The ATM MCU will then perform several checks on the database to ensure that the transaction can be completed safely and securely. If all checks pass, the ATM MCU will dispense the cash and display the remaining balance on the user's account.

The CARD MCU, on the other hand, is responsible for managing the card's information and communicating with the ATM MCU. When the card is inserted into the ATM machine, the CARD MCU will prompt the user to enter their PAN and PIN in programming mode. Once the user has completed this process, the CARD MCU will switch to user mode and send a trigger signal to the ATM MCU to initiate the transaction flow.

Overall, the ATM machine project provides a secure and reliable way for users to access their funds and perform financial transactions. The project's software requirements ensure that user data is protected, transactions are verified and validated, and system errors are minimized.

Card It consists of four layers:

1-App: This module is responsible for running the logic of the card. It will handle the user interaction and send trigger signals to the ATM ECU when necessary. It will also communicate with the EEPROM to store and retrieve data such as the card PAN and PIN.

2-ECUAL: This module provides an abstraction layer between the MCU hardware and the higher-level modules. It includes drivers for peripherals such as the LCD, EEPROM, and Trigger. These drivers provide an interface that allows the app module to interact with the hardware in a simplified way.

3-MCAL: This module provides an abstraction layer for the MCU hardware itself. It includes drivers for low-level peripherals such as DIO (Digital Input/Output), SPI (Serial Peripheral Interface), I2C (Inter-Integrated Circuit), UART (Universal Asynchronous Receiver-Transmitter), and TIMER. These drivers provide an interface that allows the ECUAL module to interact with the MCU hardware in a simplified way.

4-Service: This module includes standard data types and bit manipulation functions that can be used by the other modules to simplify their implementation.

And ATM It consists of four layers:

1-APP: The application layer is responsible for managing the main flow of the ATM, such as handling user inputs, displaying messages on the LCD screen, and managing the state of the ATM. It uses services from the service layer to perform specific operations.

2-MCUAL: The ECU abstraction layer is responsible for abstracting the hardware interface of the ATM from the application layer. It provides a set of functions that can be called from the application layer to interact with the hardware, such as turning on and off the buzzer or displaying a message on the LCD screen.

3-MCAL: The MCAL layer is responsible for abstracting the hardware interface of the microcontroller from the ECU abstraction layer. It provides a set of low-level functions that can be called from the ECU abstraction layer to interact with the microcontroller hardware, such as setting the state of a GPIO pin or sending data over SPI.

4-Service Layer: This module includes standard data types and bit manipulation functions that can be used by the other modules to simplify their implementation.

Module, Peripherals, & Supporting Drivers Description

DIO (Digital Input/Output): This module deals with the digital input and output operations, such as reading and writing to digital pins of a microcontroller or a microprocessor. It may include functions for setting pin direction, reading and writing digital values, and handling interrupts related to digital pins.

Timer: This module deals with timer operations, such as configuring and handling timers in the microcontroller or microprocessor. It may include functions for setting timer intervals, handling timer interrupts, and measuring time. And This module deals with generating PWM signals using normal mode, which are used for controlling the intensity of an output signal, such as controlling the speed of motors or the brightness of LEDs. It may include functions for configuring and controlling PWM signals.

ADC: Through this module we can initialize ADC peripheral which is responsible for converting analog input signal to digital signal, and we check on the completion of the conversation by polling on interrupt flag.

LCD: LCD stands for Liquid Crystal Display. LCD is a type of flat panel display which uses liquid crystals in its primary form of operation. It uses the light-modulating properties of liquid crystals combined with polarizers to display images. It has two pieces of polarized glass (also called substrate) that contain a liquid crystal material between them. A backlight or a reflector creates light that passes through the first substrate and the liquid crystal. The liquid crystals can twist or untwist depending on the electric voltage applied across them. This changes the angle of light passing through the second substrate and the polarizing film. The angel of light determines the color and brightness of the pixels that form the images on the LCD.

Keypad: We are using 3*3 keypad which means we have 3 rows and 3 columns, rows are connected to output high, and columns are connected to be inputs and enable internal pullups. For reading we pass low output simultaneously to the row pins and check if there is any change in the columns.

Buzzer: is an output module that buzzes when you write on it HIGH output.

I2C (Inter-Integrated Circuit): is a serial communication protocol that allows multiple devices to communicate with each other using only two wires, a clock line (SCL) and a data line (SDA). I2C is commonly used to communicate between microcontrollers, sensors, and other peripherals.

SPI (Serial Peripheral Interface): is a synchronous serial communication protocol that allows for high-speed data transfer between a master device and one or more slave devices. SPI typically uses four wires: a clock line (SCK), a data in line (MOSI), a data out line (MISO), and a slave select line (SS). SPI is commonly used in applications that require fast, full-duplex communication, such as memory devices, display drivers, and sensors.

UART (Universal Asynchronous Receiver/Transmitter): is a serial communication protocol that allows for asynchronous data transfer between devices. UART typically uses two wires: a transmit line (TX) and a receive line (RX). Unlike I2C and SPI, UART does not use a clock signal to synchronize communication. Instead, data is transmitted one bit at a time with a start and stop bit to signal the beginning and end of each byte. UART is commonly used for simple communication between a microcontroller and a computer or other device.

EEPROM (Electrically Erasable Programmable Read-Only Memory) : is a non-volatile memory that can store data even when the power is turned off. It is commonly used in microcontroller-based systems to store configuration data, calibration values, and other important information.

BIT_MATH: This module provides functions for performing bitwise operations, such as AND, OR, XOR, and shifting, which are commonly used for manipulating individual bits in registers or memory locations.

Standard Types: This module includes standard data types, such as integer types, floating-point types, and Boolean types, which are used for representing data in a standardized way across the system.

DRIVERS' DOCUMENTATION

1. DIO

DIO_init(uint8_t portNumber, uint8_t pinNumber, uint8_t direction);

Function Name	**DIO_init**
Description	Initializes DIO pins' direction, output current, and internal attach
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	uint8_t portNumber, uint8_t pinNumber, uint8_t direction
Parameters (out)	None
Return Value	WRONG_PORT_NUMBER, WRONG_PIN_NUMBER, WRONG_DIRECTION, E_OK

DIO_write(uint8_t portNumber, uint8_t pinNumber, uint8_t value);

Function Name	**DIO_write**
Description	Write on DIO pins' a specific output High or Low
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	uint8_t portNumber, uint8_t pinNumber, uint8_t value
Parameters (out)	None
Return Value	WRONG_PORT_NUMBER, WRONG_PIN_NUMBER, WRONG_VALUE, E_OK

DIO_toggle(uint8_t portNumber, uint8_t pinNumber);

Function Name	**DIO_toggle**
Description	Toggle the output of a specific pin
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	uint8_t portNumber, uint8_t pinNumber
Parameters (out)	None
Return Value	WRONG_PORT_NUMBER, WRONG_PIN_NUMBER, E_OK

DIO_read(uint8_t portNumber, uint8_t pinNumber, uint8_t *value);

Function Name	**DIO_read**
Description	Read input from a pin and send it back in a pointer to uint8_t
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	uint8_t portNumber, uint8_t pinNumber
Parameters (out)	uint8_t *value
Return Value	WRONG_PORT_NUMBER, WRONG_PIN_NUMBER, E_OK

2. TIMERS

en_timerError_t TIMER_init(u8 u8_a_timerUsed);

Function Name	**TIMER_init**
Description	Initializes a specific timer to work as a CTC or overflow timer
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	uint8_t timerUsed
Parameters (out)	None
Return Value	EN_timerError_t

en_timerError_t TIMER_setTime(u8 u8_a_timerUsed, u32 u32_a_desiredTime);

Function Name	**TIMER_setTime**
Description	Used to set time at which the timer interrupt will fires and execute a desired function
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	uint8_t timerUsed, uint32_t desiredTime
Parameters (out)	None
Return Value	EN_timerError_t

en_timerError_t TIMER_start(u8 u8_a_timerUsed);

Function Name	**TIMER_start**
Description	Start specific timer to count
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	uint8_t timerUsed
Parameters (out)	None
Return Value	EN_timerError_t

en_timerError_t TIMER_stop(u8 u8_a_timerUsed);

Function Name	**TIMER_stop**
Description	Stop specific timer from counting
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	uint8_t timerUsed
Parameters (out)	None
Return Value	EN_timerError_t

en_timerError_t TIMER_pwmGenerator(u8 u8_a_timerUsed, u32 u32_a_desiredDutyCycle);

Function Name	**TIMER_pwmGenerator**
Description	Generates PWM signal using normal mode for a specific timer
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	u8_a_timerUsed, u8_a_desiredDutyCycle
Parameters (out)	None
Return Value	en_timerError_t

void TIMER_setCallBack(u8 u8_a_timerUsed, void (*funPtr)(void));

Function Name	**TIMER_setCallBack**
Description	Initializes Sends pointer to function to be called when the timer’s interrupt fires
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	uint8_t portNumber, uint8_t pinNumber, uint8_t direction
Parameters (out)	None
Return Value	None

en_timerError_t TIMER_stopInterrupt(u8 u8_a_timerUsed);

Function Name	**TIMER_stopInterrupt**
Description	Disable a specific timer’s peripheral interrupt
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	u8_a_timerUsed
Parameters (out)	None
Return Value	en_timerError_t

en_timerError_t TIMER_delay(u8 u8_a_timerUsed, u32 u32_a_timeInMS);

Function Name	**TIMER_enableInterrupt**
Description	Generates a delay using a specific timer
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	u8_a_timerUsed, u32_a_timeInMS
Parameters (out)	None
Return Value	en_timerError_t

en_timerError_t TIMER_enableInterrupt(u8 u8_a_timerUsed);

Function Name	**TIMER_enableInterrupt**
Description	Enables a specific timer’s peripheral interrupt
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	u8_a_timerUsed
Parameters (out)	None
Return Value	en_timerError_t

3. SPI

void SPI_initMaster(void);

Function Name	**SPI_initMaster**
Description	Initializes the SPI module as a master.
Sync\Async	Synchronous
Reentrancy	None-Reentrant
Parameters (in)	None
Parameters (out)	None
Return Value	None

void SPI_initSlave(void);

Function Name	**SPI_initSlave**
Description	Initializes the SPI module as a slave.
Sync\Async	Synchronous
Reentrancy	None-Reentrant
Parameters (in)	None
Parameters (out)	None
Return Value	None

u8 SPI_transmitByte(u8 data);

Function Name	**SPI_trasmitByte**
Description	Enables the SPI communication by bringing the SS line low.
Sync\Async	Synchronous
Reentrancy	None-Reentrant
Parameters (in)	data
Parameters (out)	None
Return Value	U8

4. LCD

void LCD_Init(void);

Function Name	**LCD_Init**
Description	Initialize LCD according to preprocessed configured definitions
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters (out)	None
Return Value	None

void LCD_PinsInit ();

Function Name	**LCD_PinInit**
Description	Initialize LCD pins directions according to preprocessed configured definitions
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters (out)	None
Return Value	None

void LCD_WriteChar(u8 u8_a_ch);

Function Name	**LCD_WriteChar**
Description	Prints Character on LCD
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	U8_a_ch
Parameters (out)	None
Return Value	None

void LCD_WriteString(u8 *u8_a_str);

Function Name	**LCD_WriteString**
Description	Prints string on LCD
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	*u8_a_str
Parameters (out)	None
Return Value	None

void LCD_WriteNumber(i32 i32_a_num);

Function Name	**LCD_WriteNumber**
Description	Prints a specific number on LCD
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	i32_a_num
Parameters (out)	None
Return Value	None

void LCD_SetCursor(u8 u8_a_line,u8 u8_a_cell);

Function Name	**LCD_SetCursor**
Description	Changes Cursor’s Location
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	u8_a_line, u8_a_cell
Parameters (out)	None
Return Value	None

void LCD_Clear(void);

Function Name	**LCD_Clear**
Description	Clears LCD's screen and set cursor at line 0 cell 0
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters (out)	None
Return Value	None

void LCD_ClearLoc(u8 u8_a_line ,u8 u8_a_cell,u8 u8_a_num);

Function Name	**LCD_ClearLoc**
Description	Clear specific cells from a specific location
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	u8_a_line, u8_a_cell_, u8_a_num
Parameters (out)	None
Return Value	None

void LCD_CustomChar(u8 u8_a_loc,u8 *u8_a_pattern);

Function Name	**LCD_CustomChar**
Description	Creates a customized character
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	u8_a_loc, *u8_a_pattern
Parameters (out)	None
Return Value	None

5. KEYPAD

void KEYPAD_init (void);

Function Name	**KEYPAD_init**
Description	Initialize KEYPAD according to preprocessed configured definitions
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters (out)	None
Return Value	None

u8 KEYPAD_read (void);

Function Name	**KEYPAD_read**
Description	returns 0 if there is no key pressed or equivalent value for the key if there is a key pressed
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters (out)	None
Return Value	U8

6. BUZZEER

en_buzzerError_t BUZZER_init (u8 u8_a_buzzerNumber);

Function Name	**BUZZER_init**
Description	Initialize Buzzer according to preprocessed configured definitions
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	u8_a_buzzerNumber
Parameters (out)	None
Return Value	BUZZER_OK, WRONG_BUZZER.

en_buzzerError_t BUZZER_on (u8 u8_a_buzzerNumber);

Function Name	**BUZZER_on**
Description	Switches Buzzer On
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	u8_a_buzzerNumber
Parameters (out)	None
Return Value	BUZZER_OK, WRONG_BUZZER.

en_buzzerError_t BUZZER_off (u8 u8_a_buzzerNumber);

Function Name	**BUZZER_off**
Description	Turns Buzzer off
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	u8_a_buzzerNumber
Parameters (out)	None
Return Value	BUZZER_OK, WRONG_BUZZER.

7. UART

void UART_Init(void);

Function Name	**UART_Init**
Description	Initializes the UART module by setting the baud rate and frame format, enabling the receiver and transmitter, and configuring the stop bit.
Sync\Async	Synchronous
Reentrancy	None-Reentrant
Parameters (in)	None
Parameters (out)	None
Return Value	None

void UART_SendChar(u8 data);

Function Name	**Uart_SendChar**
Description	Sends a single character through the UART interface.
Sync\Async	Synchronous
Reentrancy	None-Reentrant
Parameters (in)	Data
Parameters (out)	None
Return Value	BUZZER_OK, WRONG_BUZZER.

u8 UART_GetChar(void);

Function Name	**UART_GetChar**
Description	Receives a single character through the UART interface.
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters (out)	None
Return Value	U8

8. I2C

en_I2CError_t I2C_start(void);

Function Name	**I2C_start**
Description	Initializes the I2C module with the specified configuration settings.
Sync\Async	Synchronous
Reentrancy	None-Reentrant
Parameters (in)	None
Parameters (out)	None
Return Value	en_I2CError_t

en_I2CError_t I2C_repeated_start(void);

Function Name	**I2C_repeated_start**
Description	Generates a repeated start condition on the I2C bus.
Sync\Async	Synchronous
Reentrancy	None-Reentrant
Parameters (in)	Data
Parameters (out)	None
Return Value	en_I2CError_t

en_I2CError_t I2C_address_select(u8 adress,u8 rw);

Function Name	**I2C_address_select**
Description	Selects an I2C slave device address and sets the read/write bit.
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	U8
Parameters (out)	None
Return Value	en_I2CError_t

en_I2CError_t I2C_data_rw(u8 *data,u8 rw,u8 ack);

Function Name	**I2C_data_rw**
Description	Reads or writes data on the I2C bus, with or without an acknowledgment.
Sync\Async	Synchronous
Reentrancy	None-Reentrant
Parameters (in)	Data ,ack,rw
Parameters (out)	None
Return Value	en_I2CError_t

void I2C_init(void);

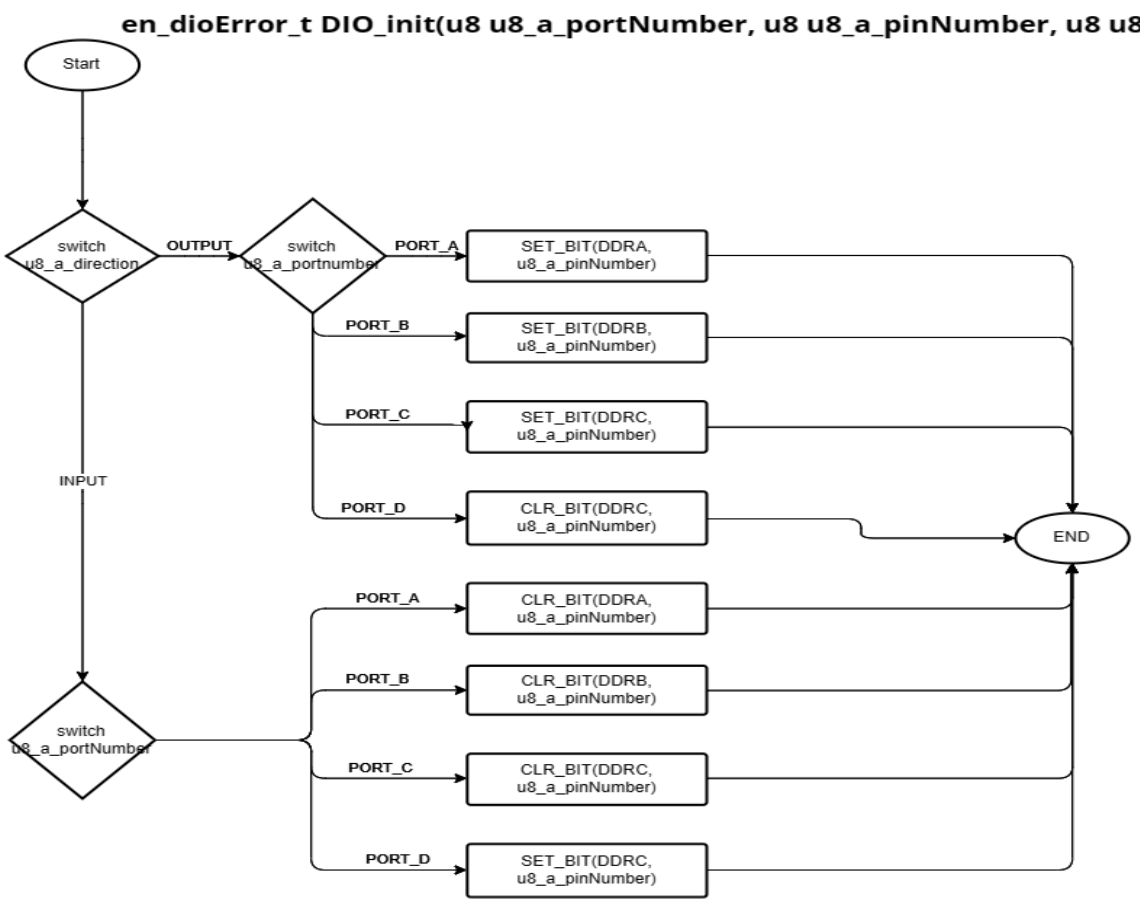
Function Name	**I2C_init**
Description	Initializes the I2C module.
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters (out)	None
Return Value	None

void I2C_stop(void);

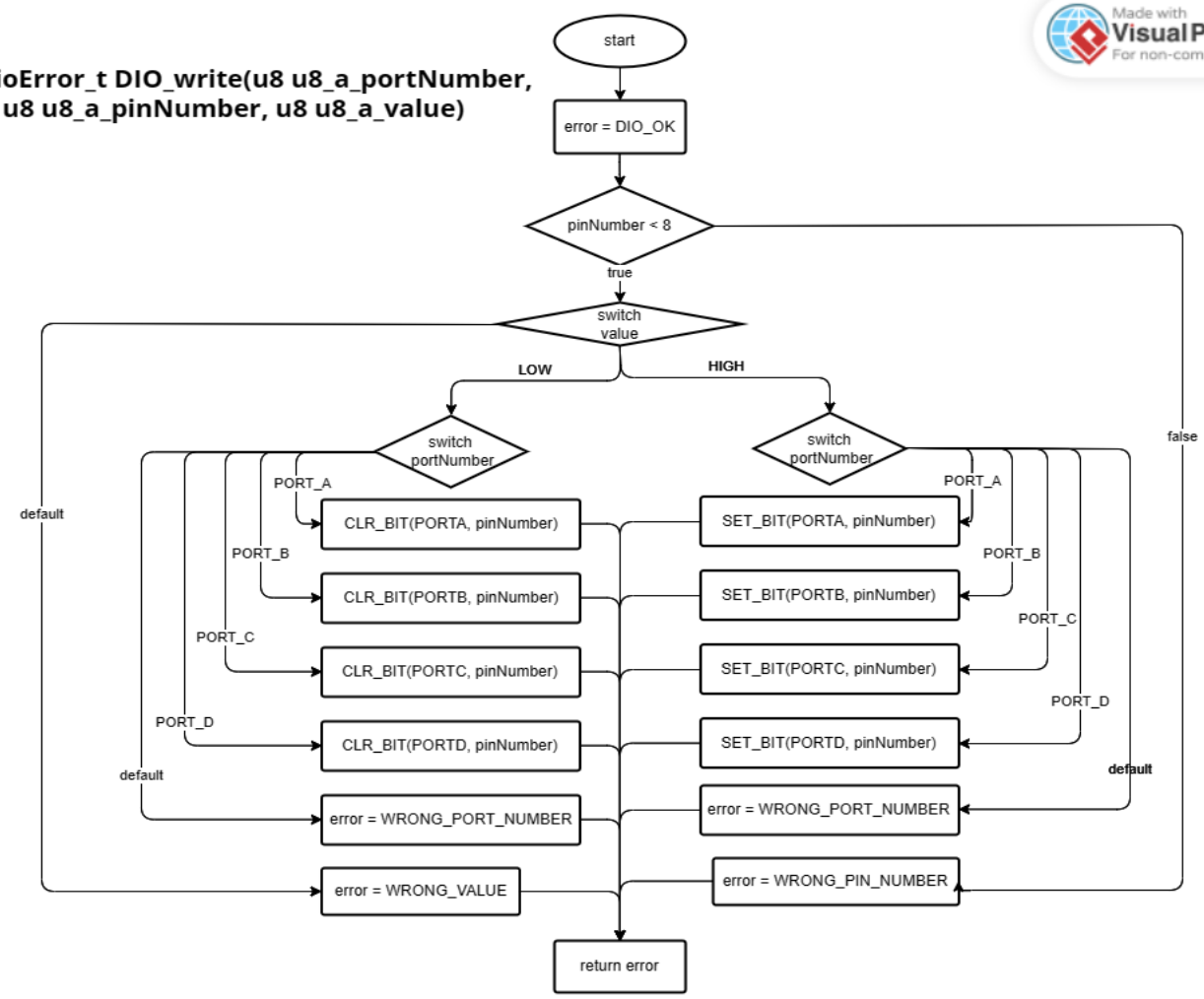
Function Name	**I2C_stop**
Description	Generates a stop condition on the I2C bus.
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters (out)	None
Return Value	None

FUNCTIONS' FLOWCHARTS

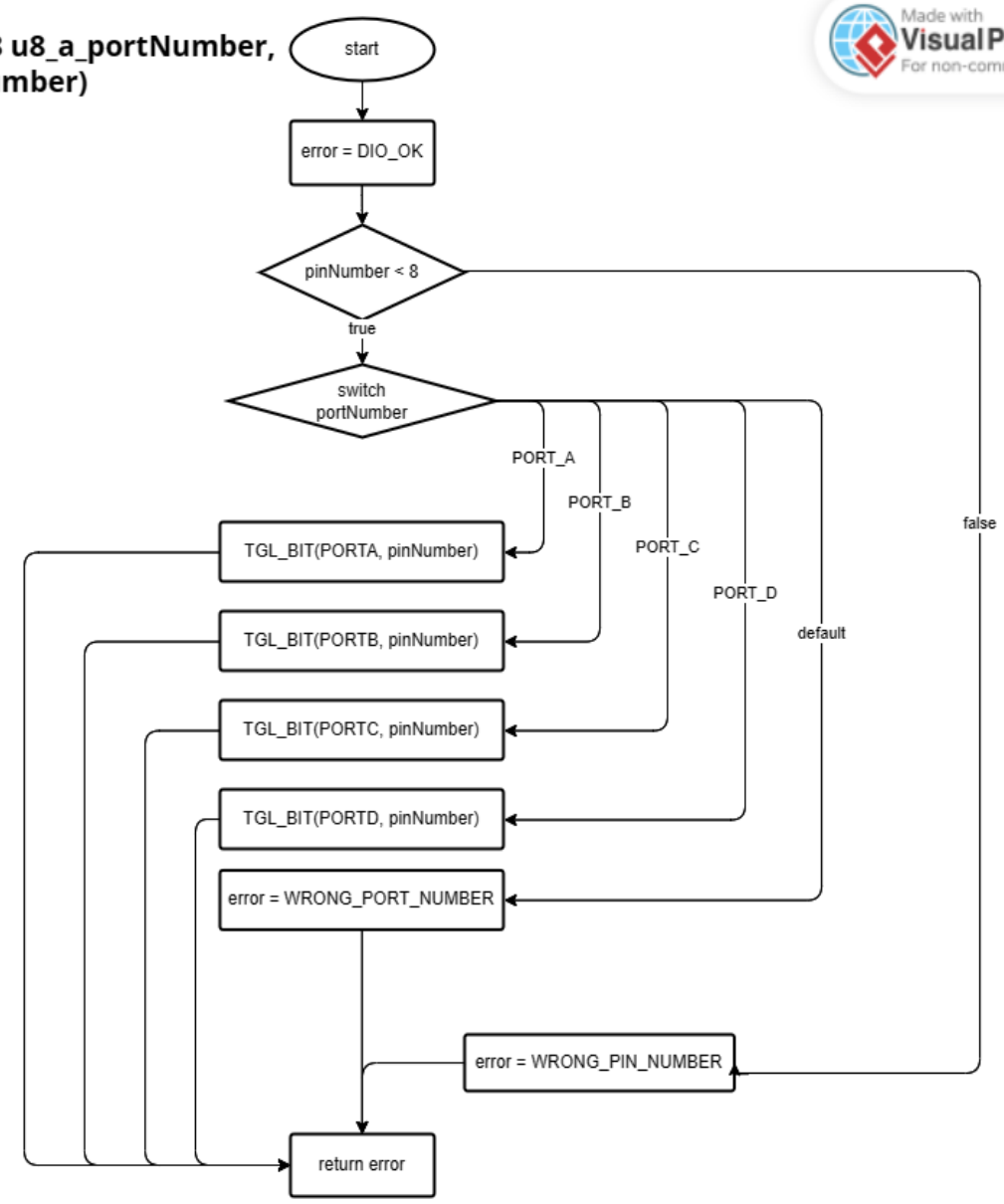
1. DIO



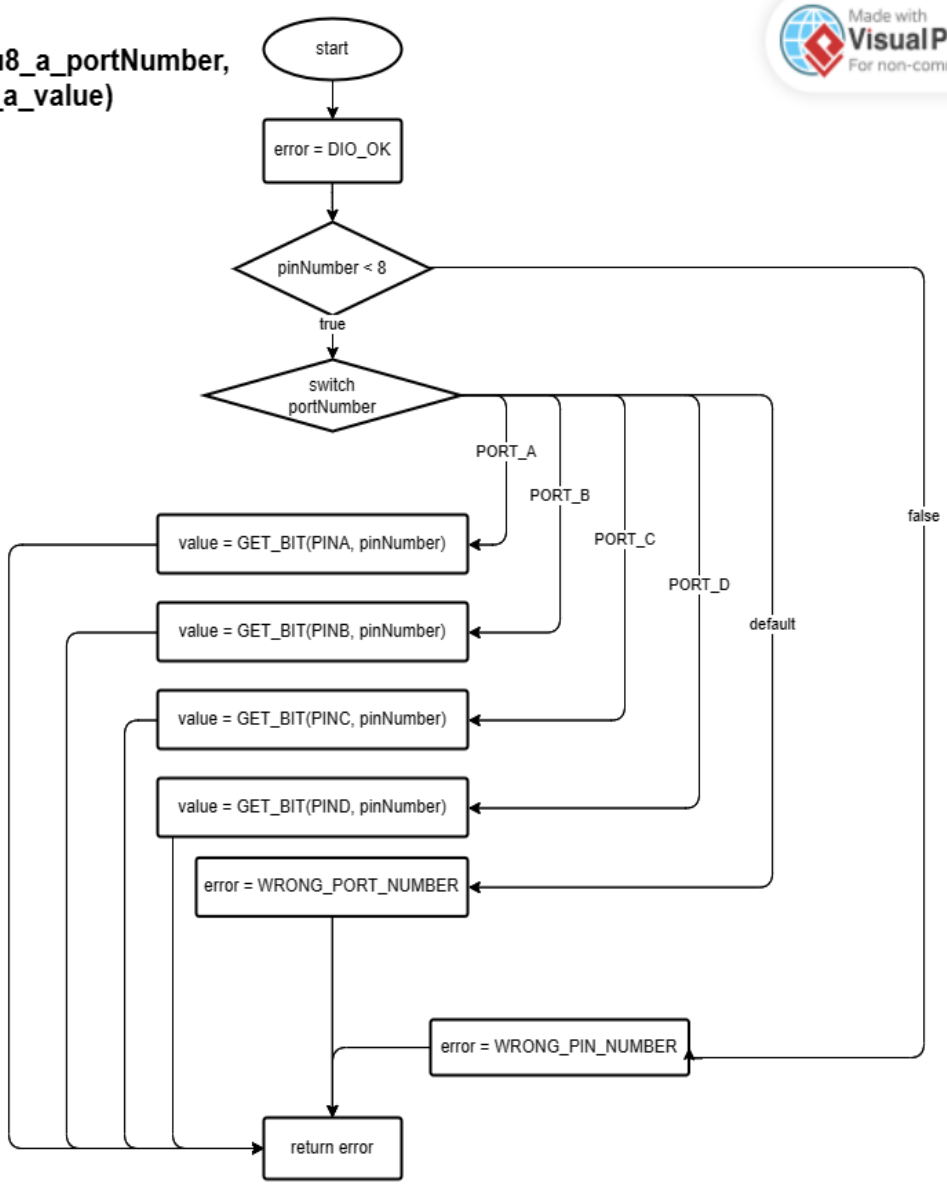
en_dioError_t DIO_write(u8 u8_a_portNumber, u8 u8_a_pinNumber, u8 u8_a_value)



en_dioError_t DIO_toggle(u8 u8_a_portNumber, u8 u8_a_pinNumber)

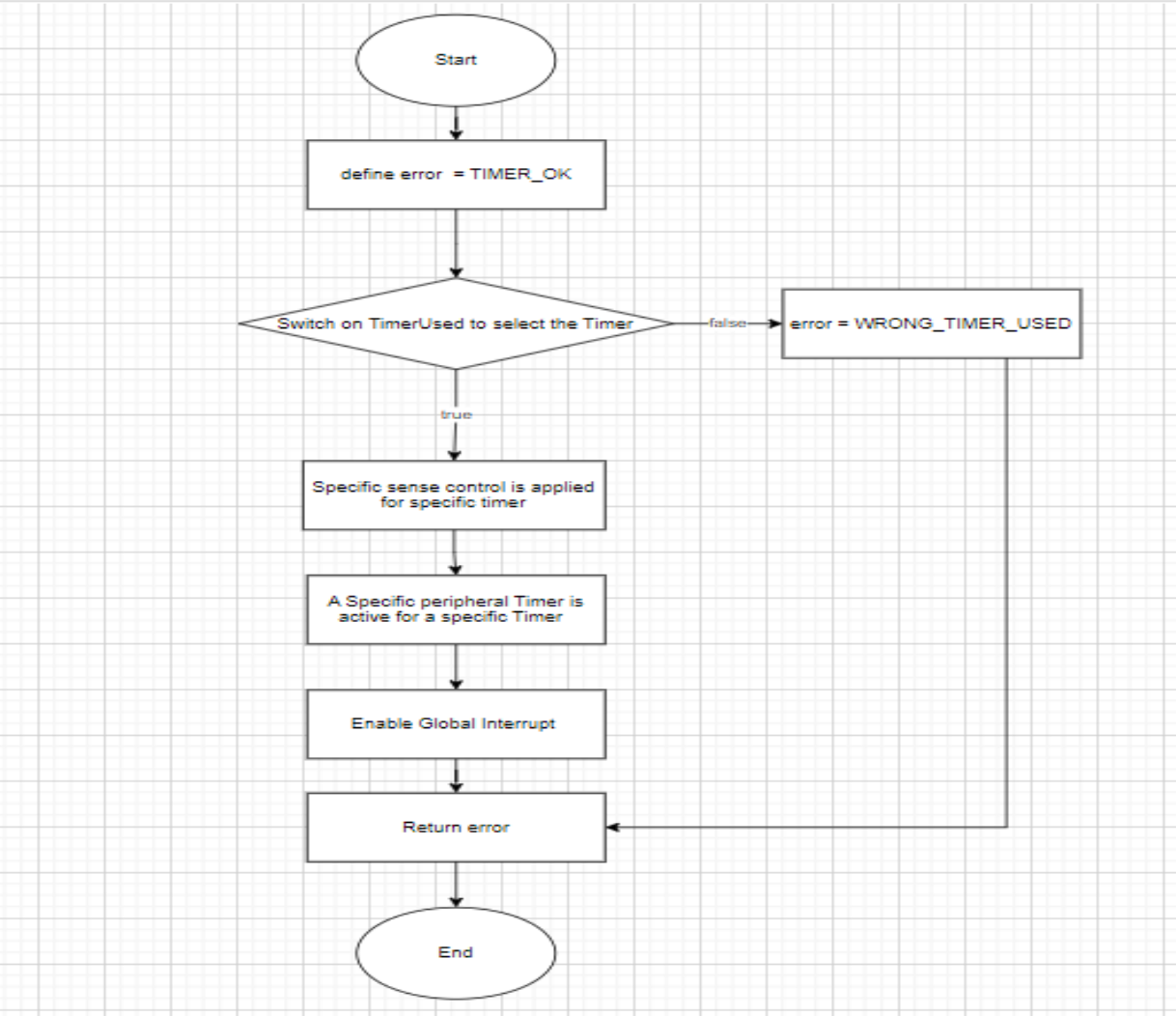


en_dioError_t DIO_read(u8 u8_a_portNumber, u8 u8_a_pinNumber, u8 *u8_a_value)

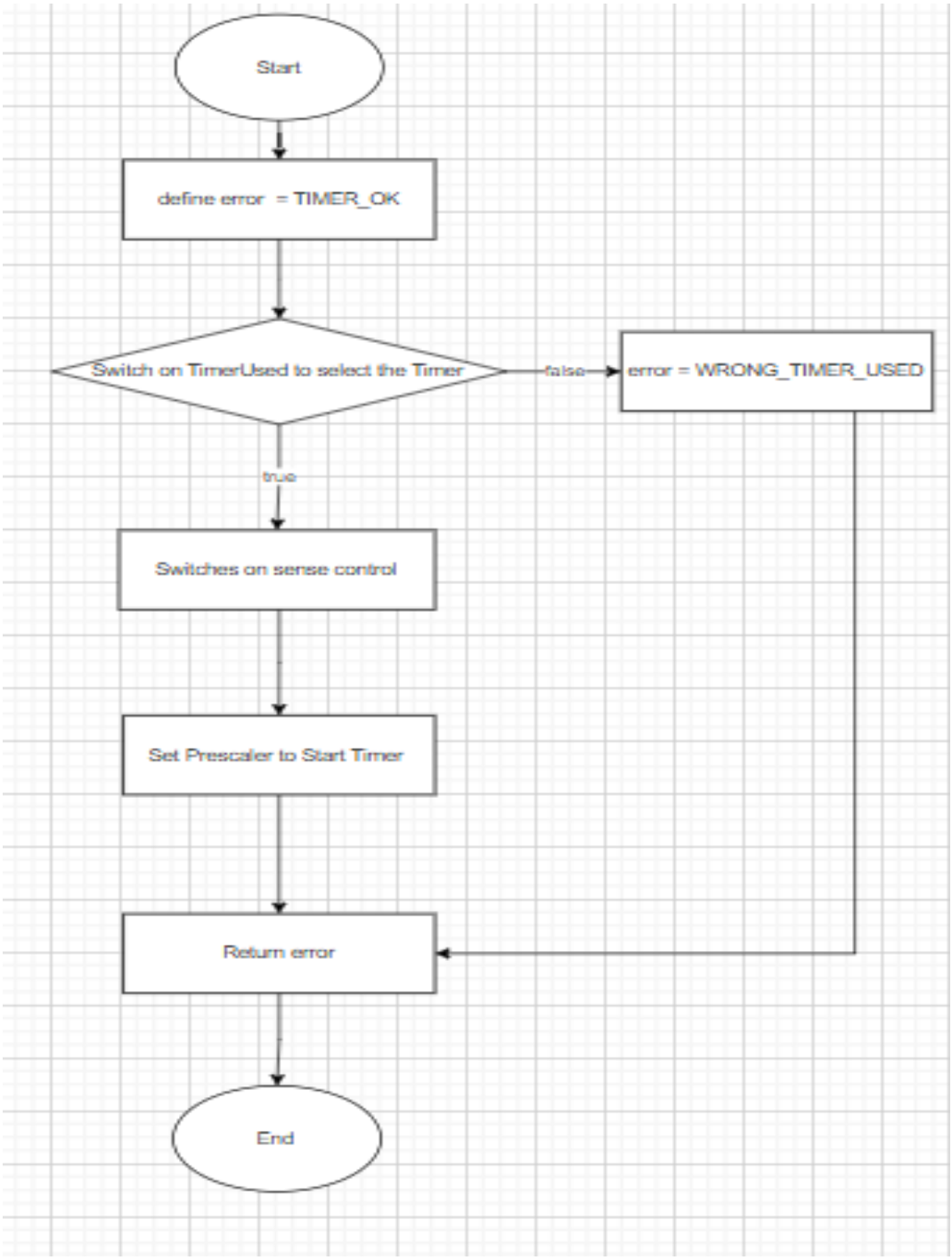


2. TIMERS

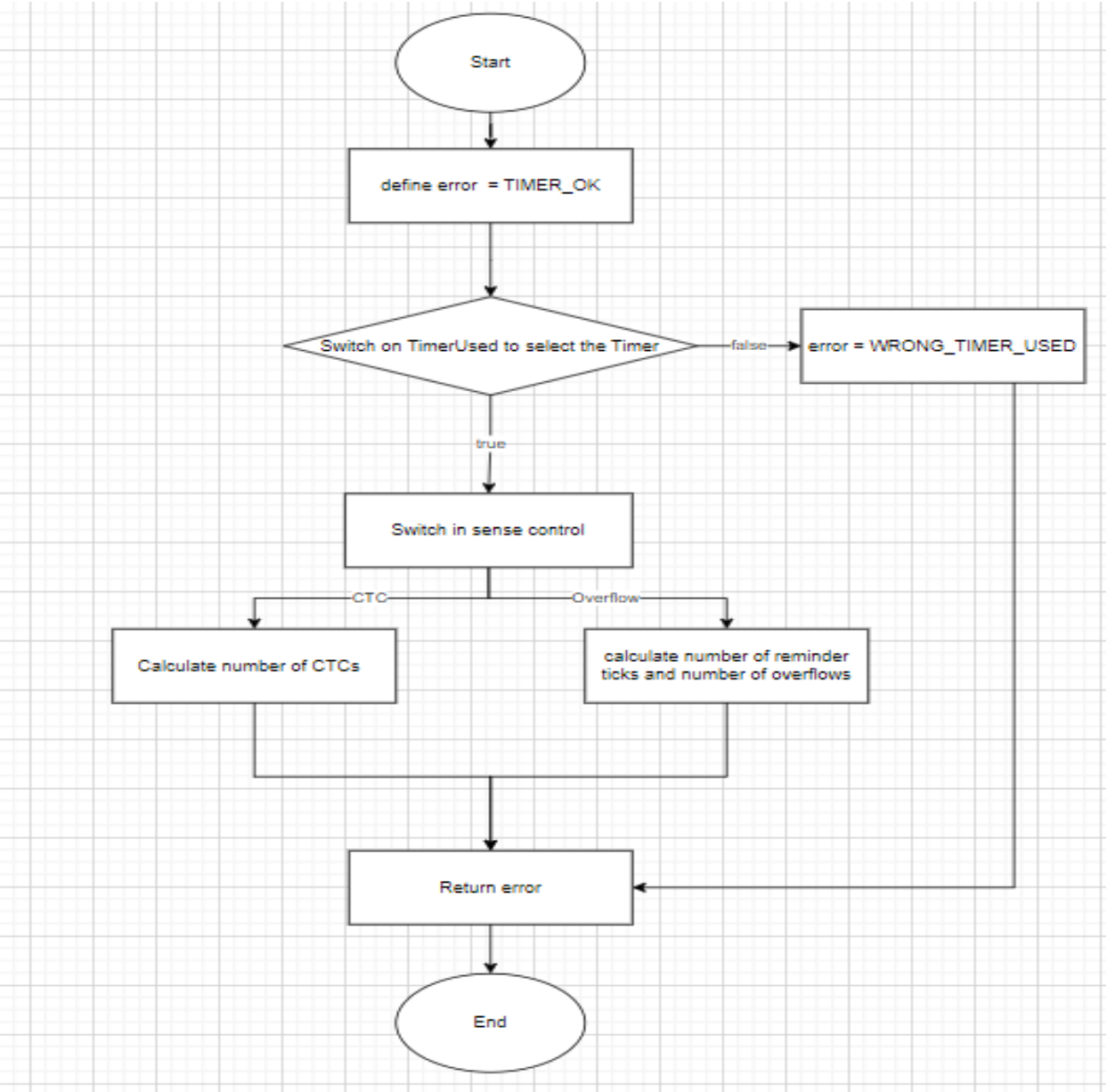
en_timerError_t TIMER_init(u8 u8_a_timerUsed);



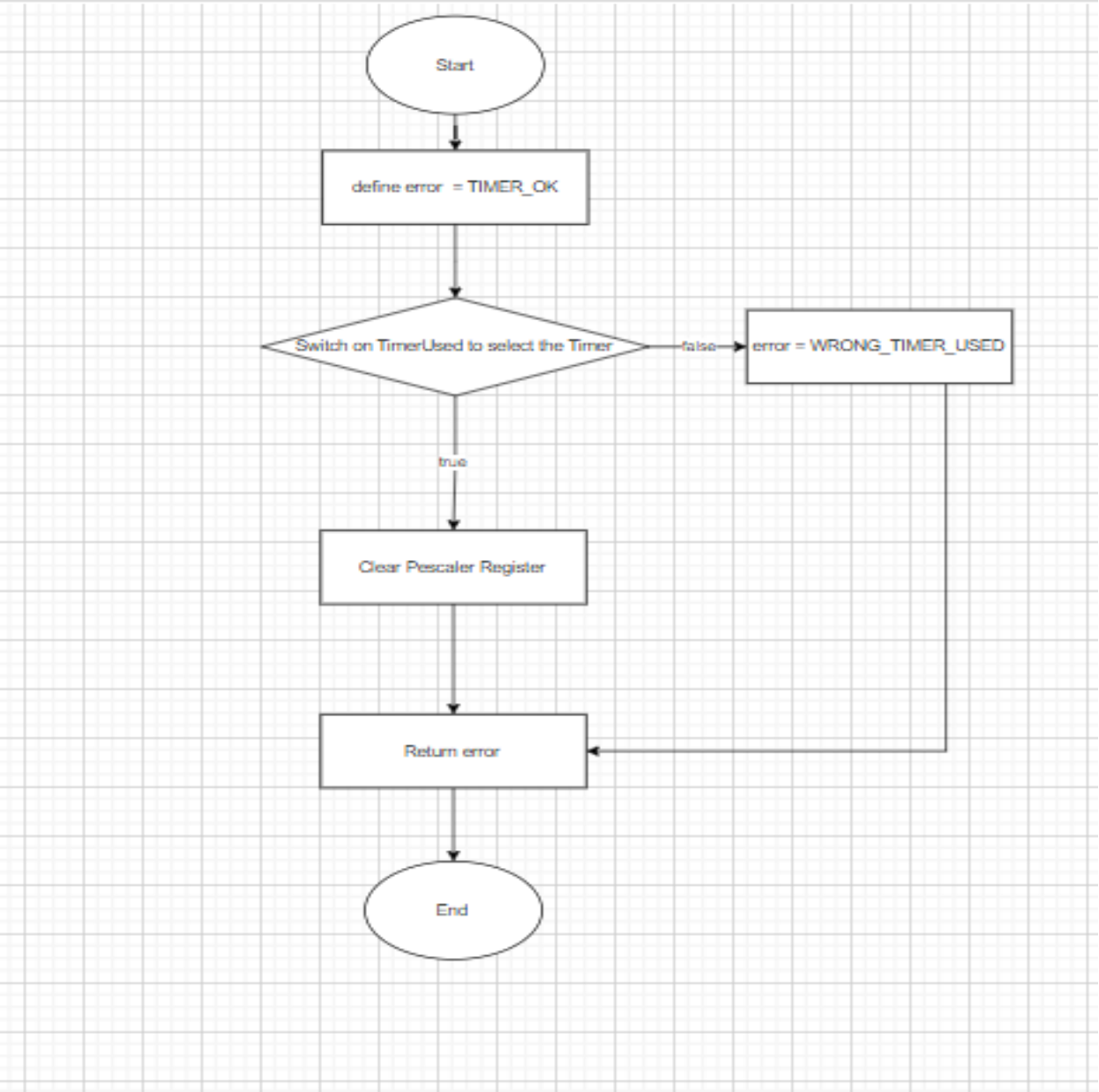
en_timerError_t TIMER_start(u8 u8_a_timerUsed);



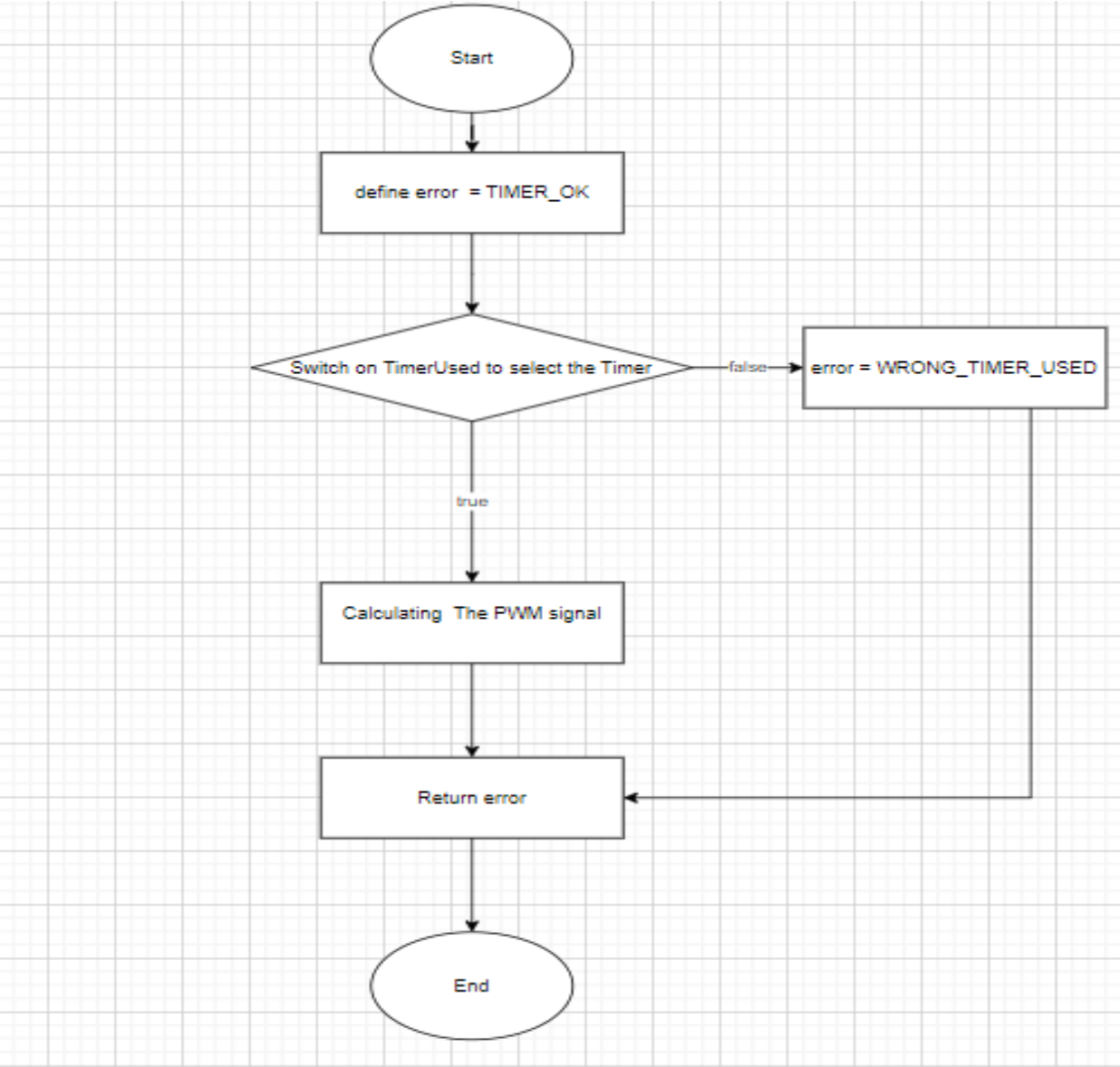
en_timerError_t TIMER_setTime(u8 u8_a_timerUsed, u32 u32_a_desiredTime);



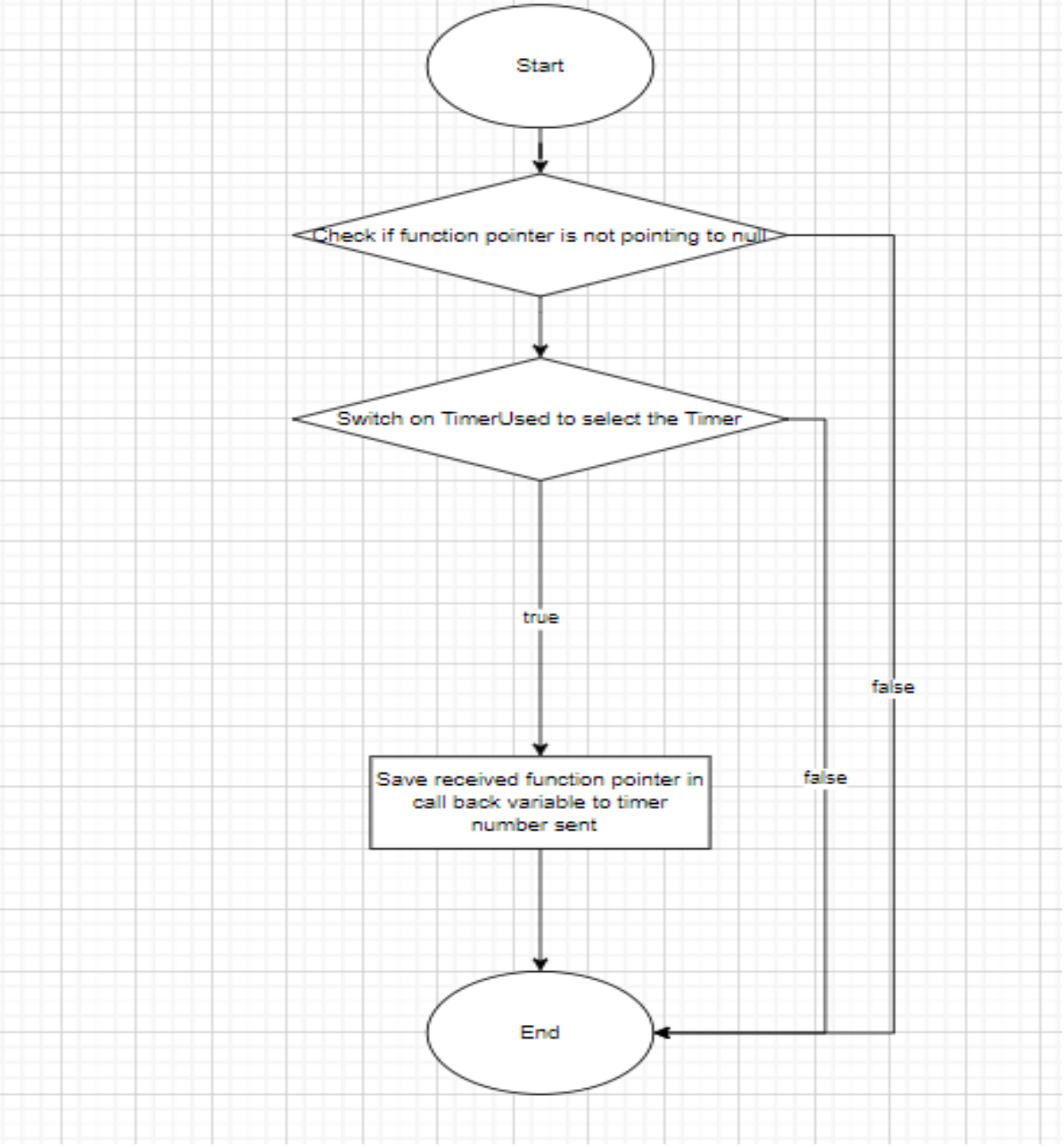
en_timerError_t TIMER_stop(u8 u8_a_timerUsed);



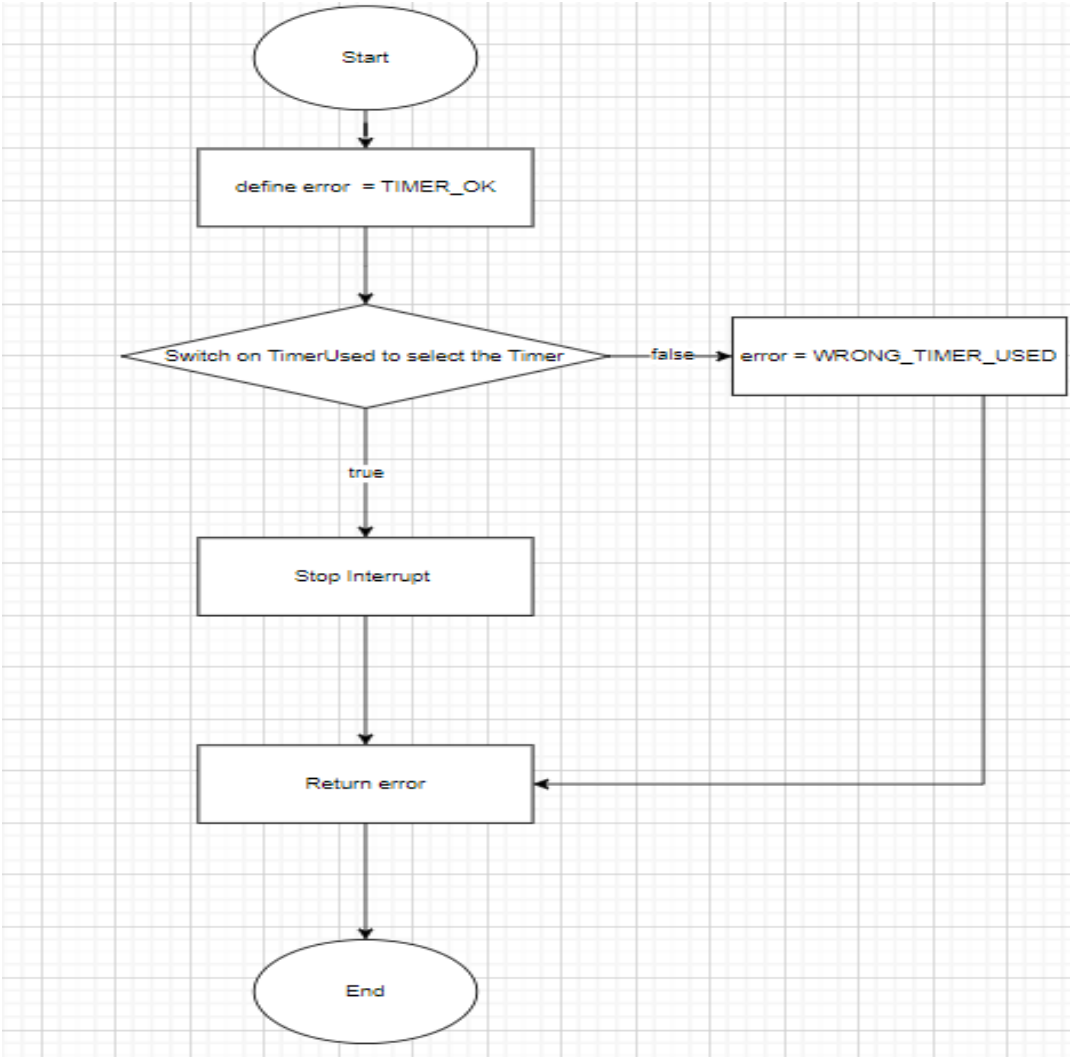
en_timerError_t TIMER_pwmGenerator(u8 u8_a_timerUsed, u32 u32_a_desiredDutyCycle);



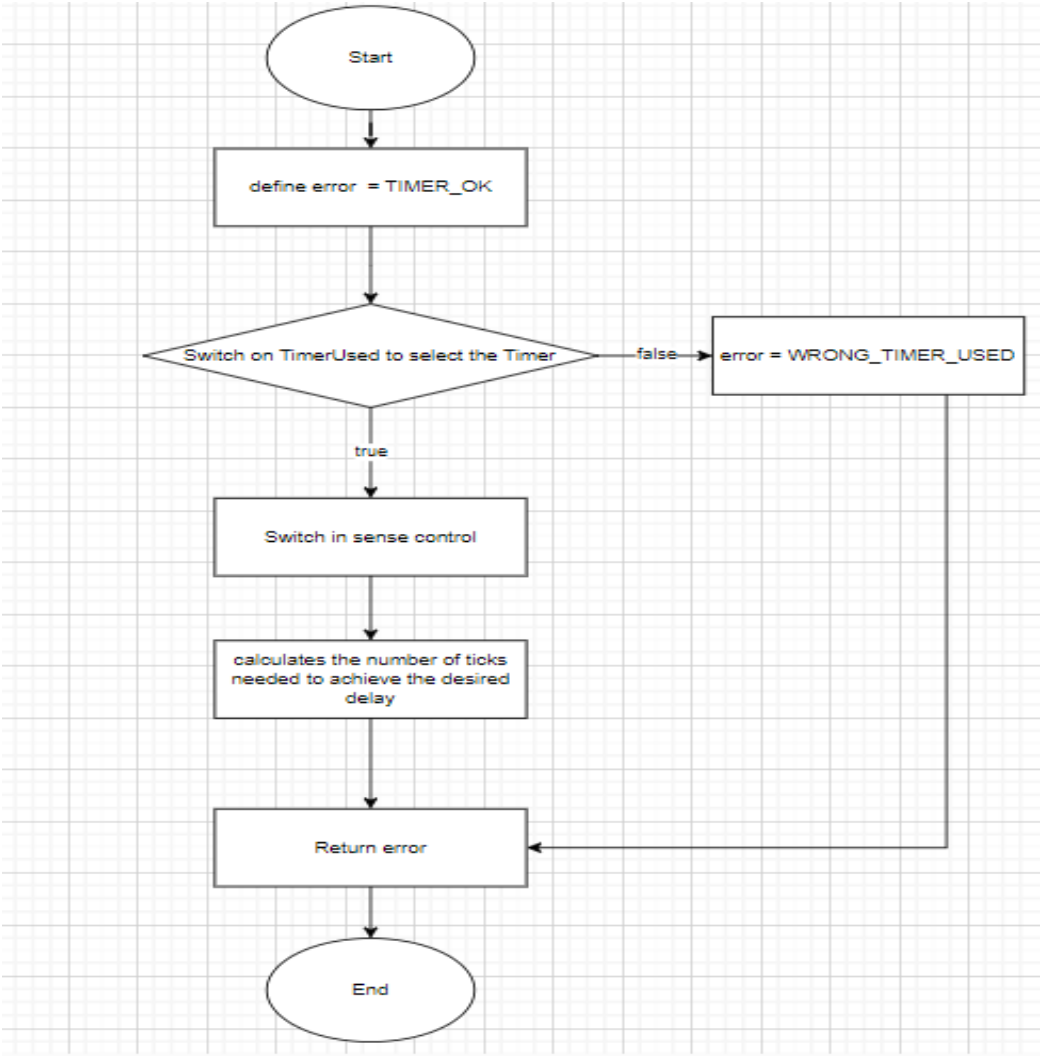
Void TIMER_setCallBack(u8 u8_a_timerUsed, void (*funPtr)(void));



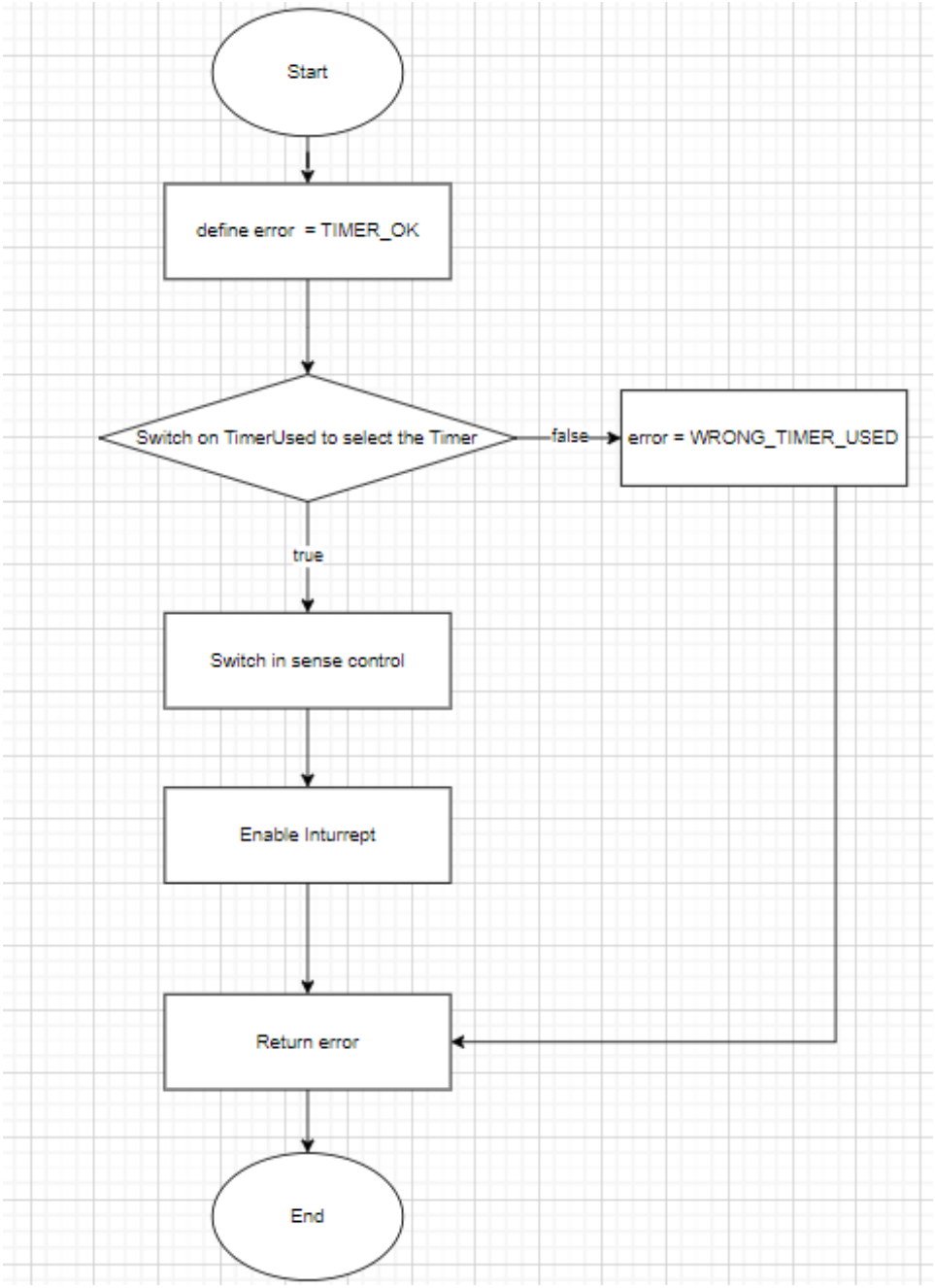
en_timerError_t TIMER_stopInterrupt(u8 u8_a_timerUsed);



en_timerError_t TIMER_delay(u8 u8_a_timerUsed, u32 u32_a_timeInMS);

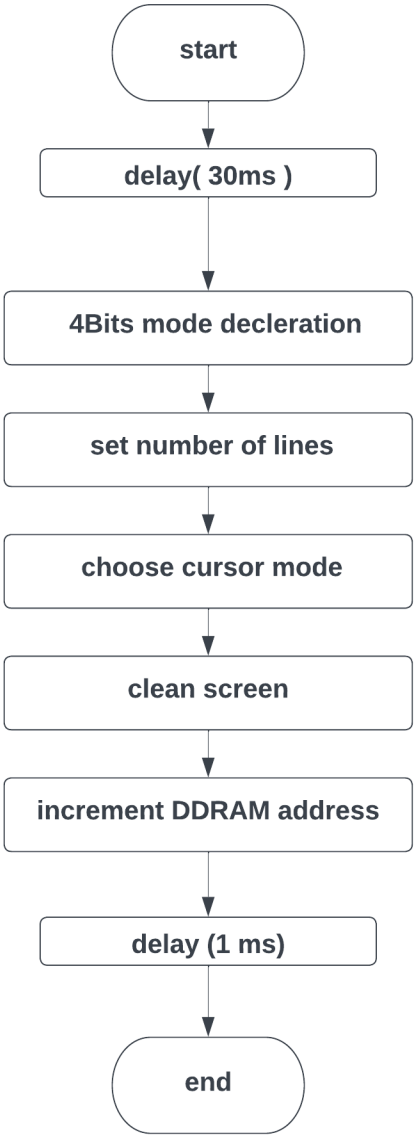


en_timerError_t TIMER_enableInterrupt(u8 u8_a_timerUsed);

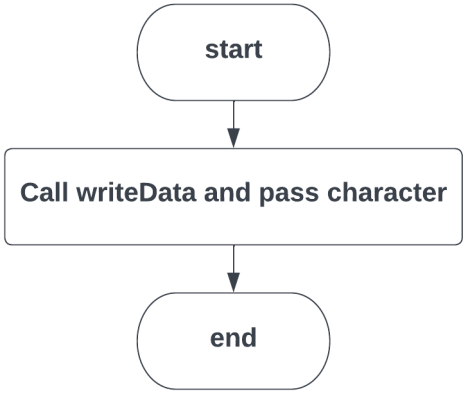


3. LCD

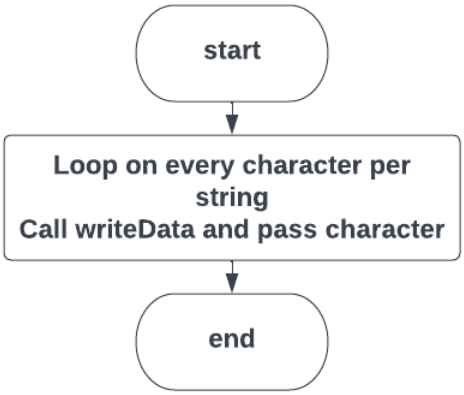
LCD_Init()



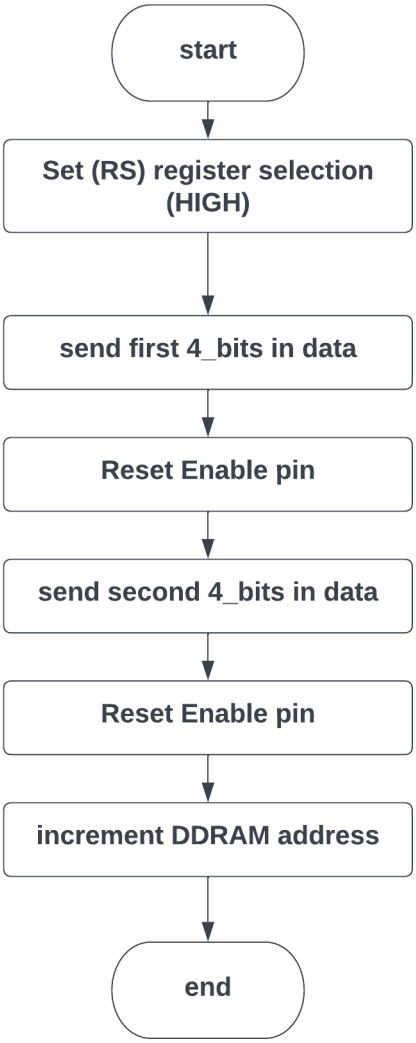
LCD_WriteChar(u8 ch)



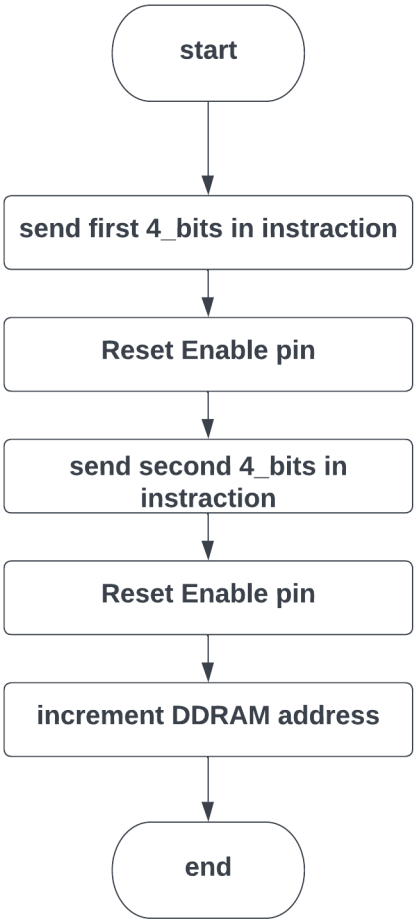
LCD_WriteString(u8*str)



WriteData(u8 data)

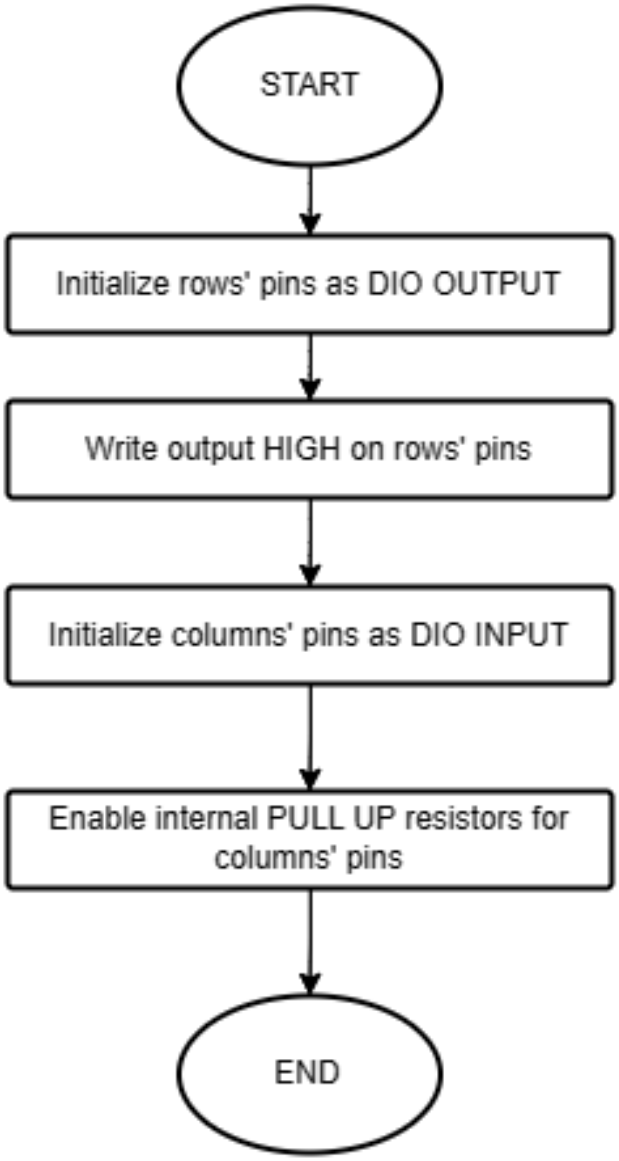


WriteIns(u8 ins)

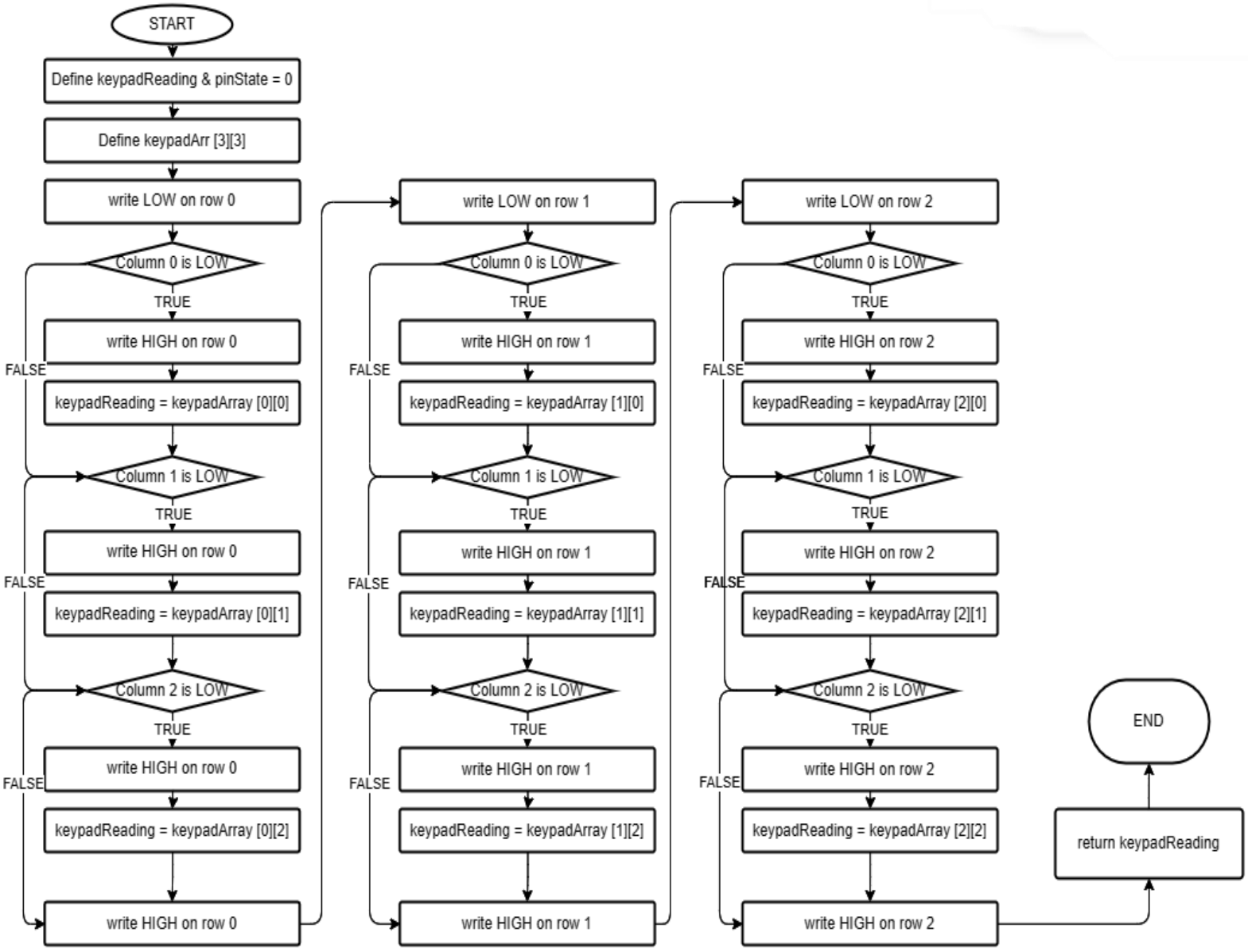


4. KEYPAD

void KEYPAD_init(void);

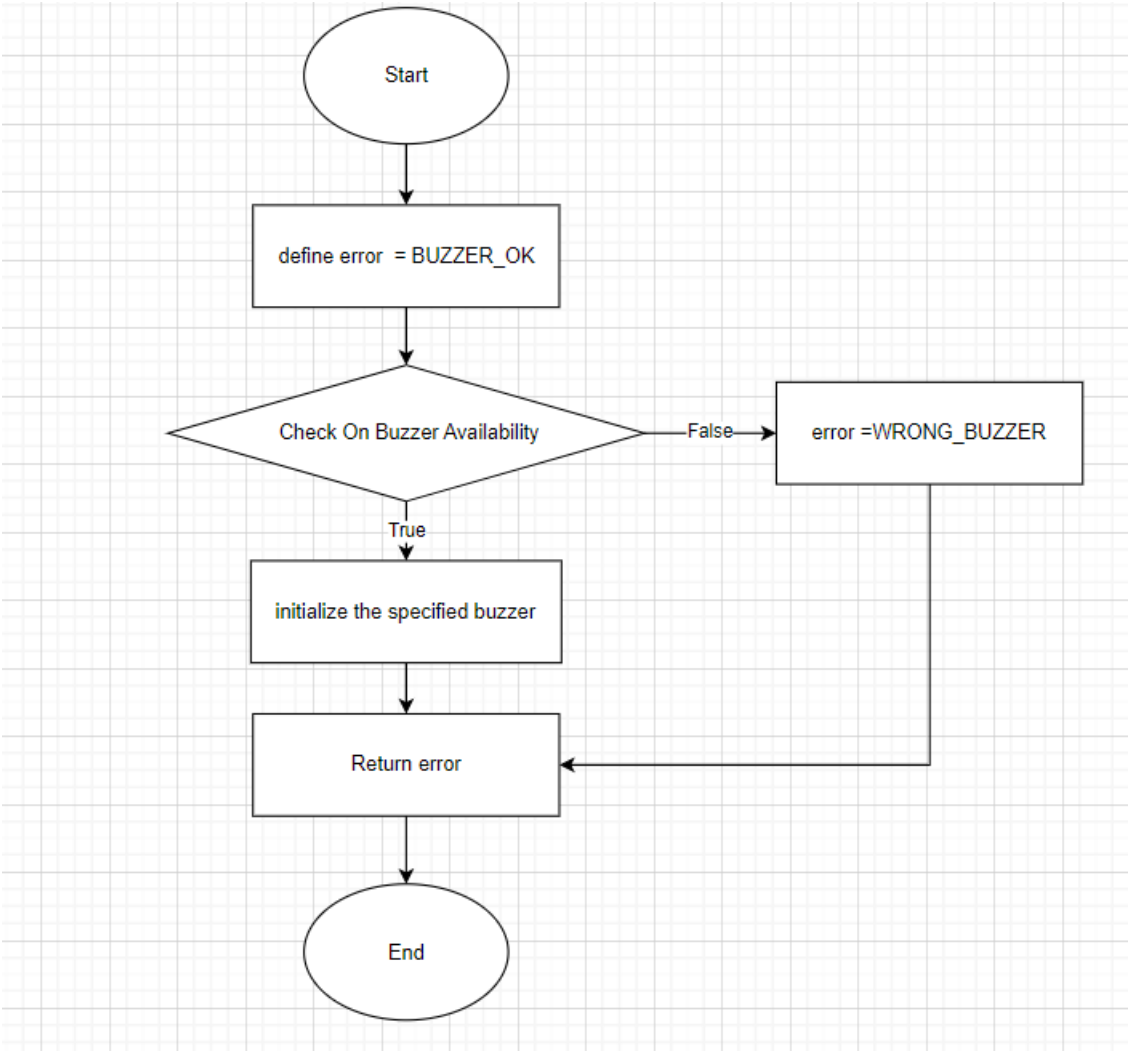


u8 KEYPAD_read(void)

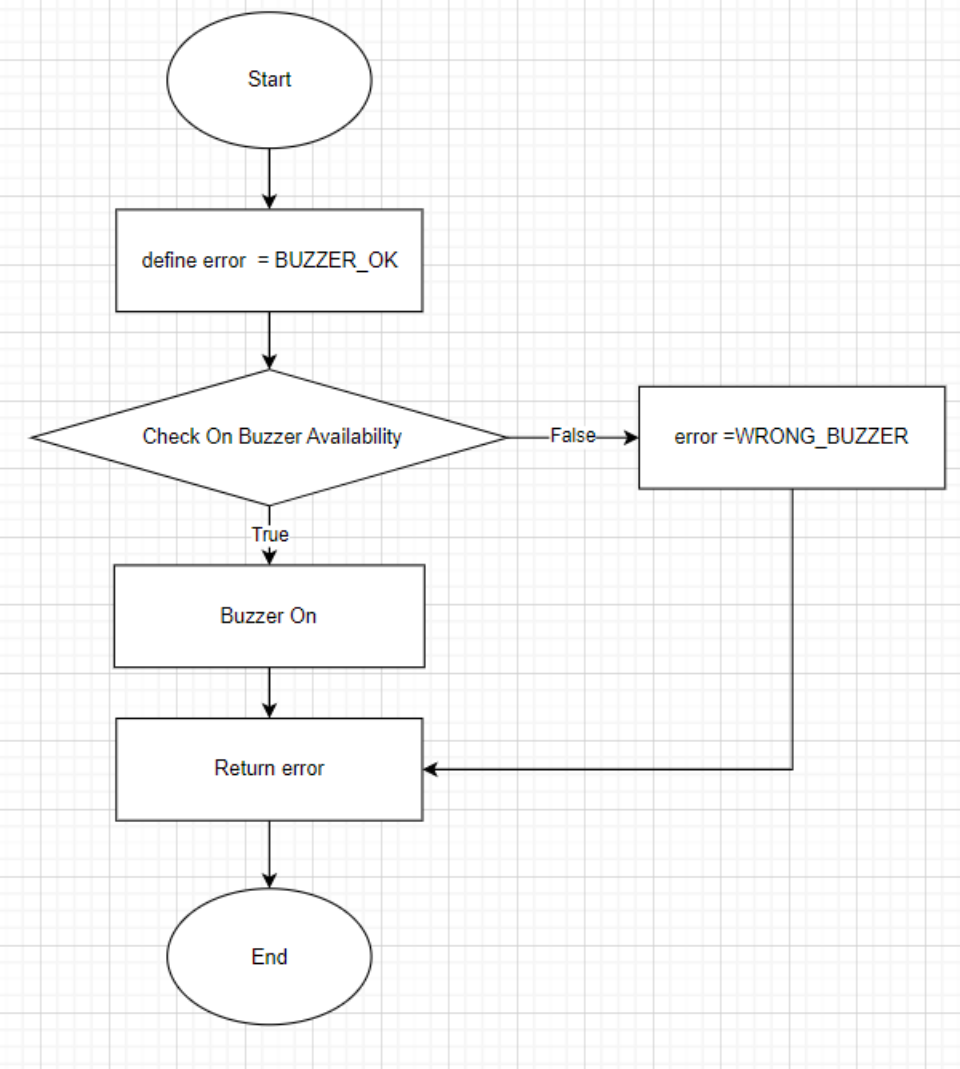


5. BUZZER

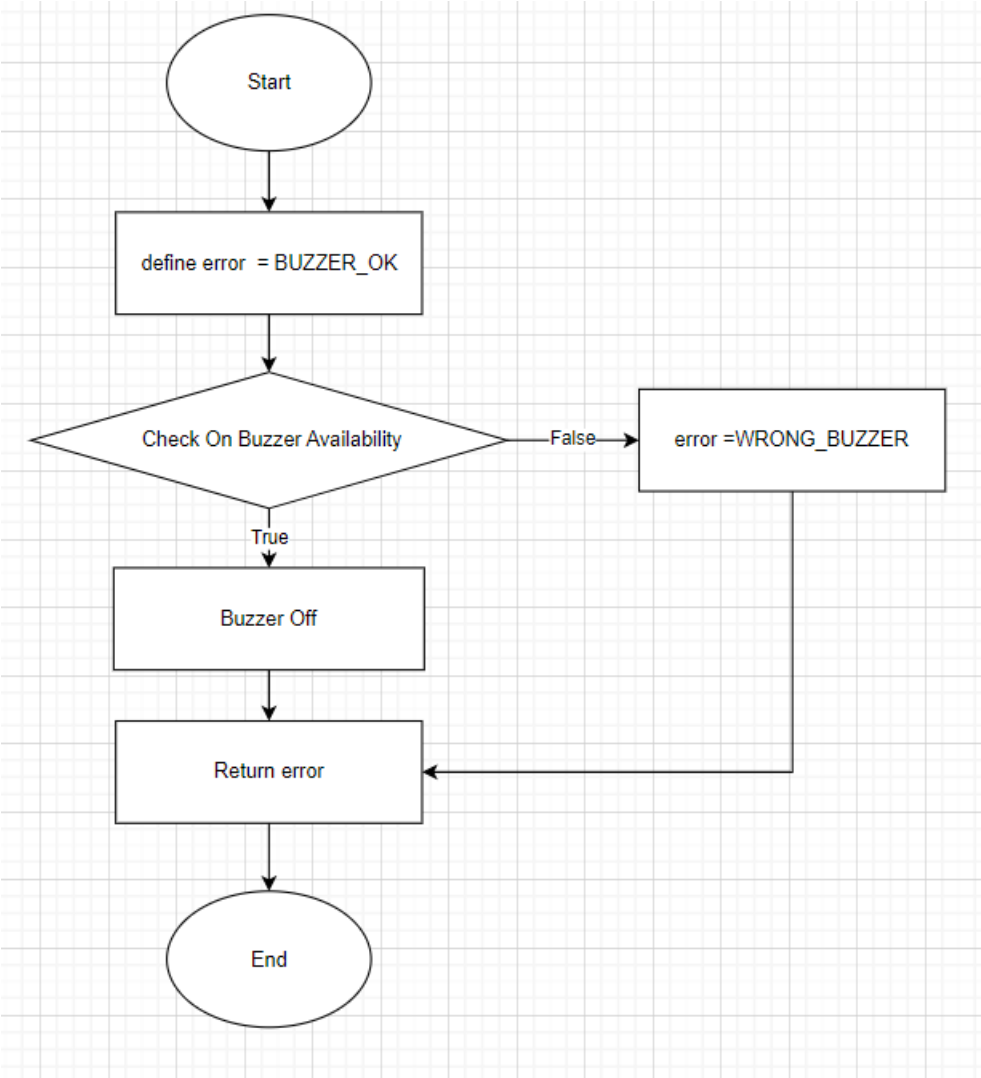
BUZZER_init()



BUZZER_on

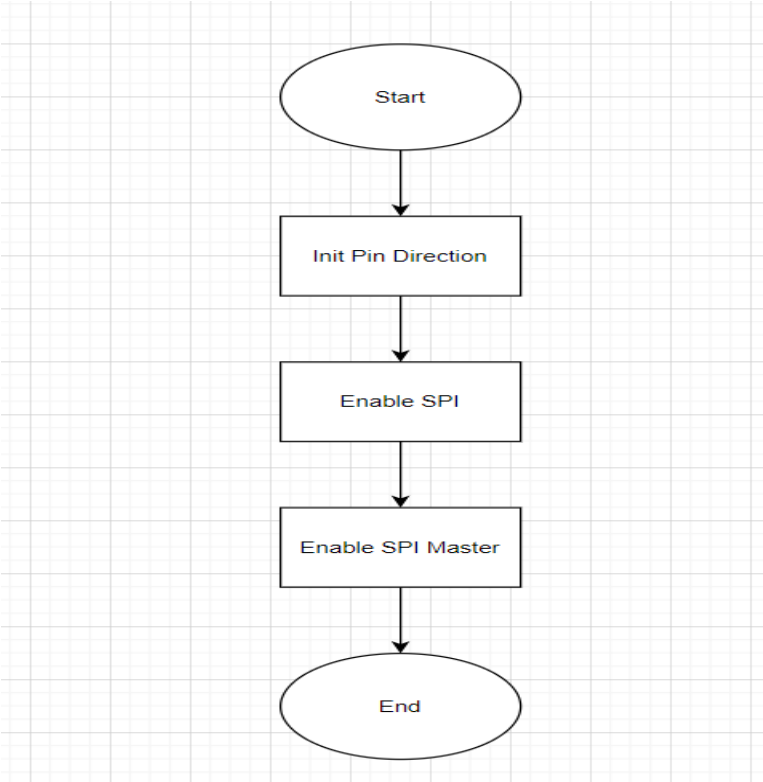

Start
define error = BUZZER_OK
Check On Buzzer Availability
False
error =WRONG_BUZZER
True
Buzzer On
Return error
End

BUZZER_OFF

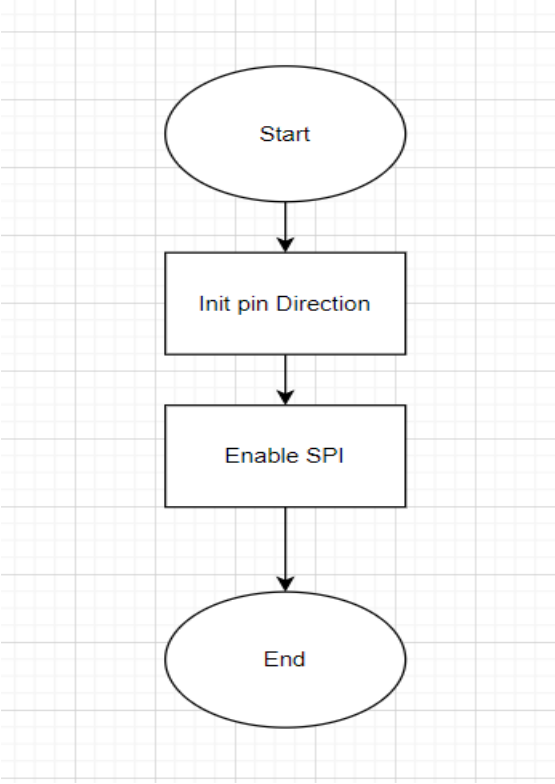

Start
define error = BUZZER_OK
Check On Buzzer Availability
False
error =WRONG_BUZZER
True
Buzzer Off
Return error
End

6. SPI

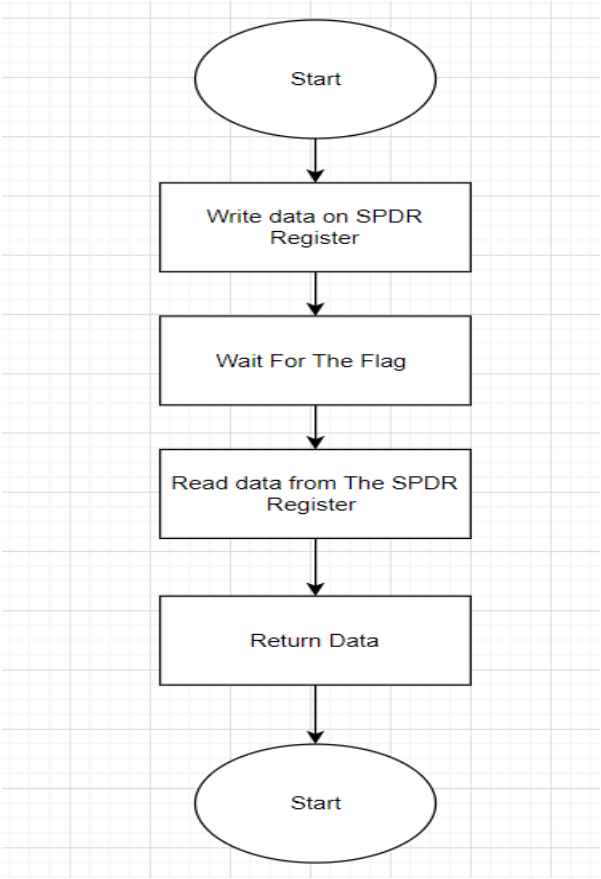
SPI_initMaster



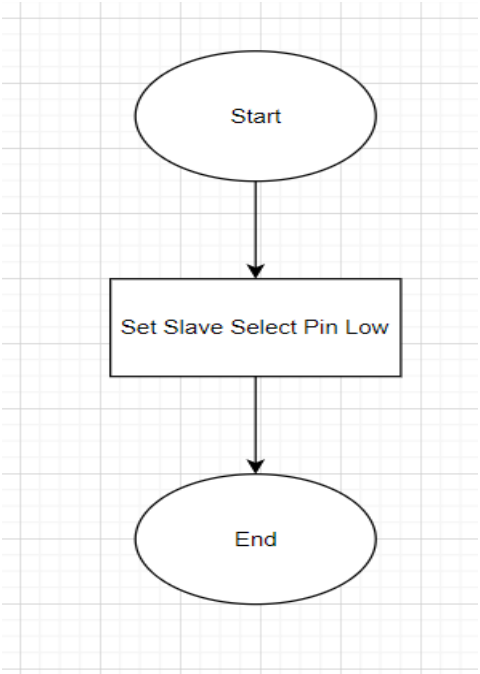
SPI_initSlave



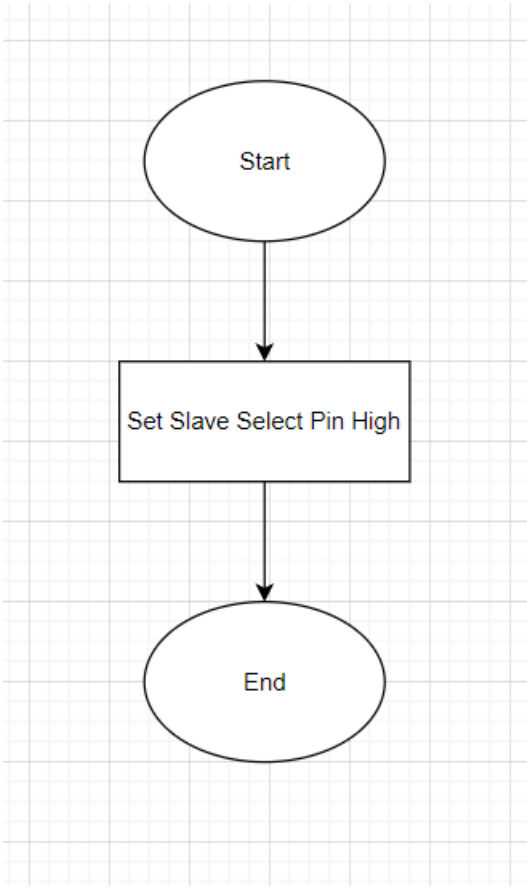
SPI_transmitByte



SPI_startTransmission

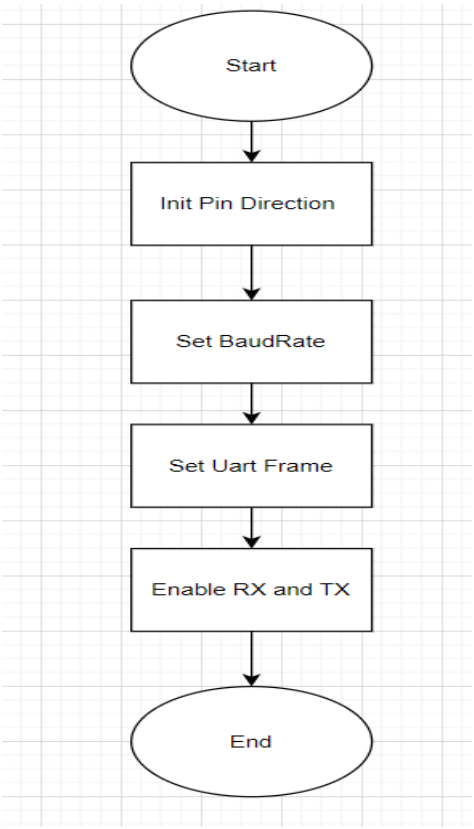


SPI_stopTransmission

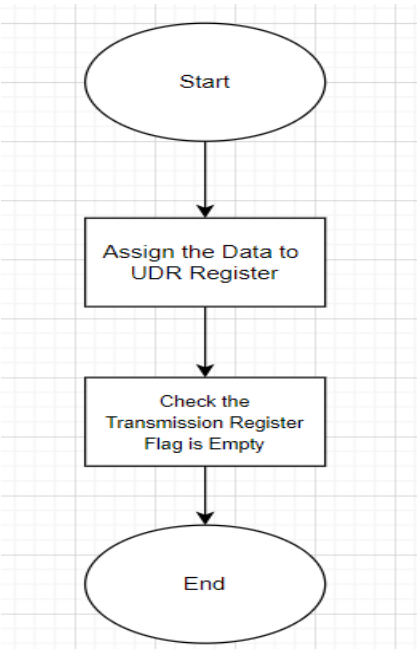


7. UART

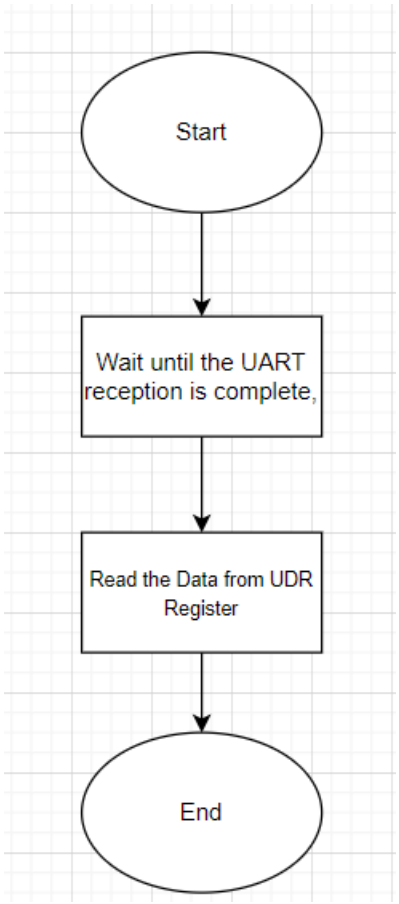
UART_Init



UART_SendChar

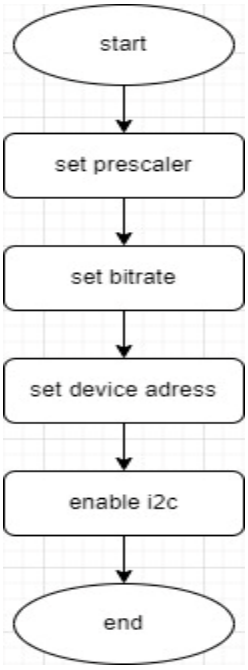


UART_GetChar

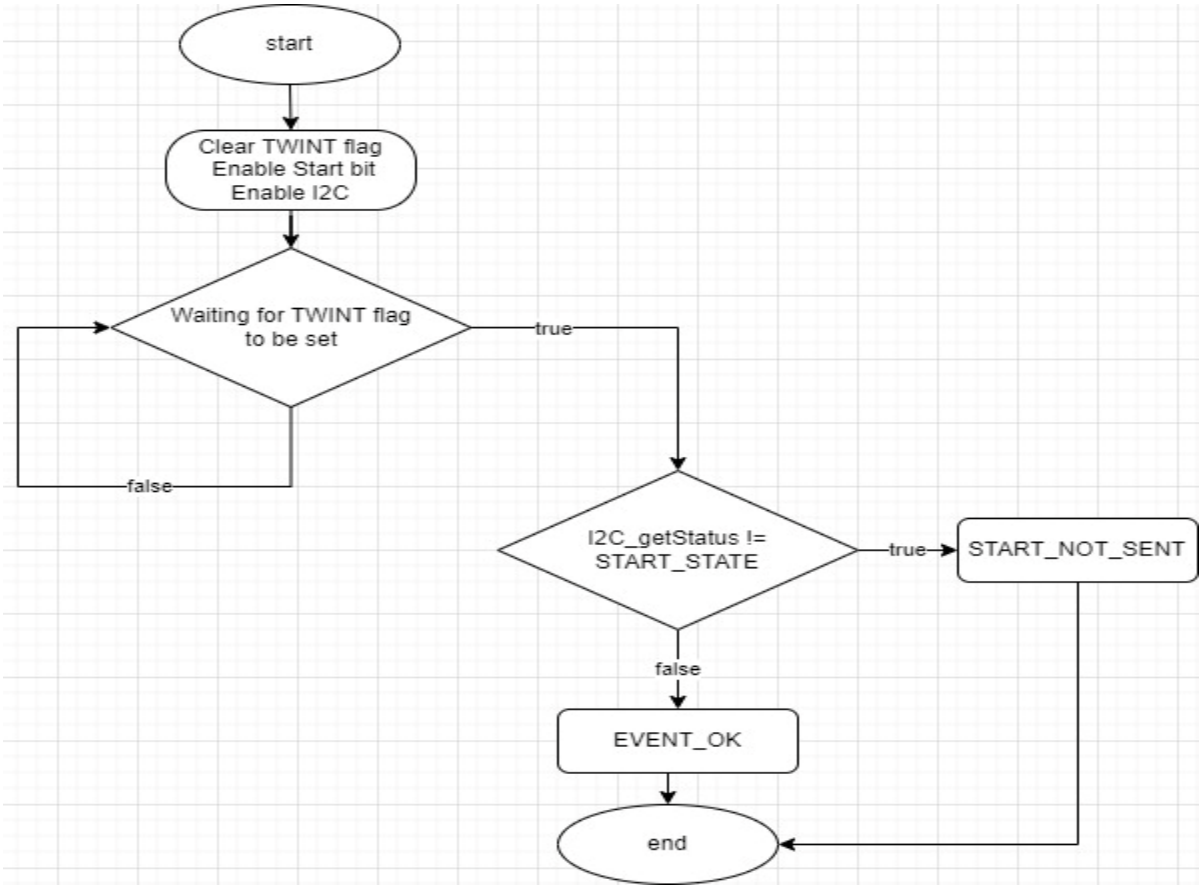


8. I2C

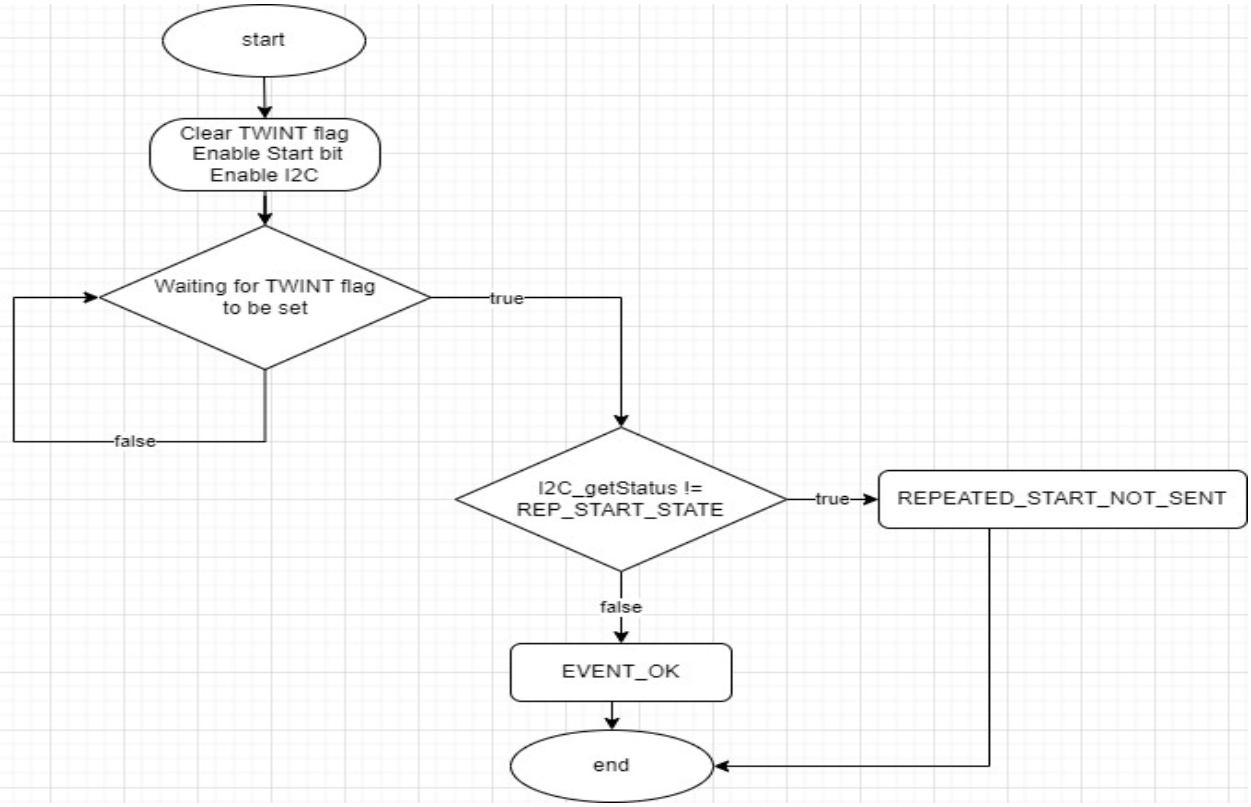
I2C_init



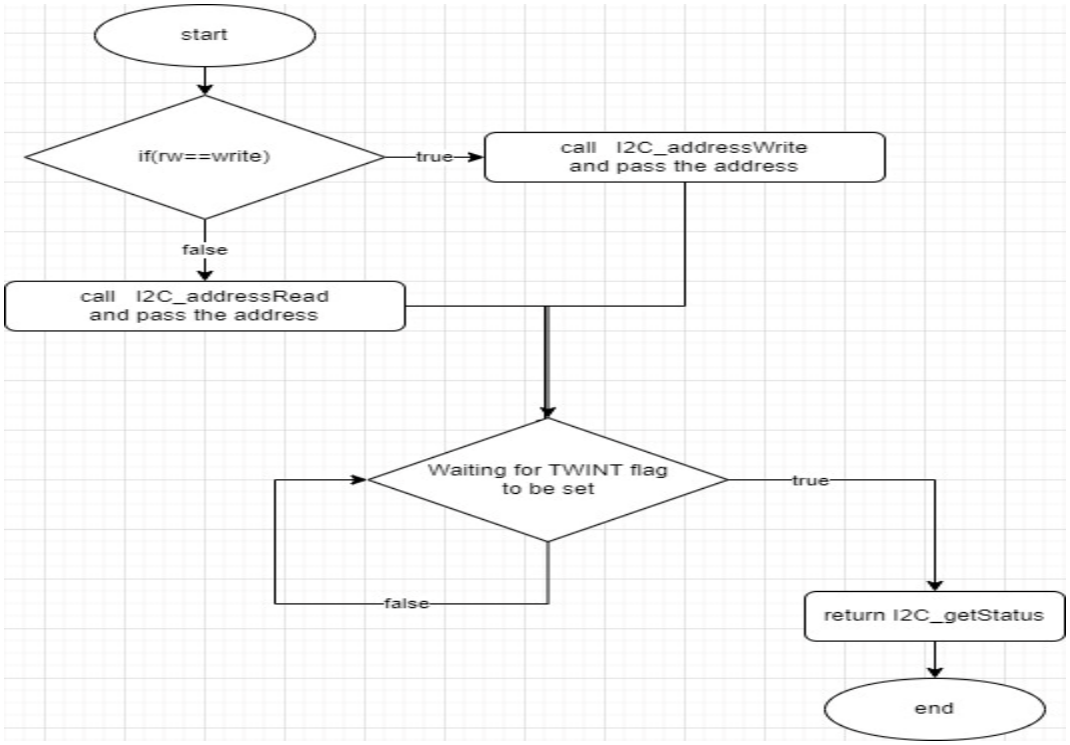
I2C_start



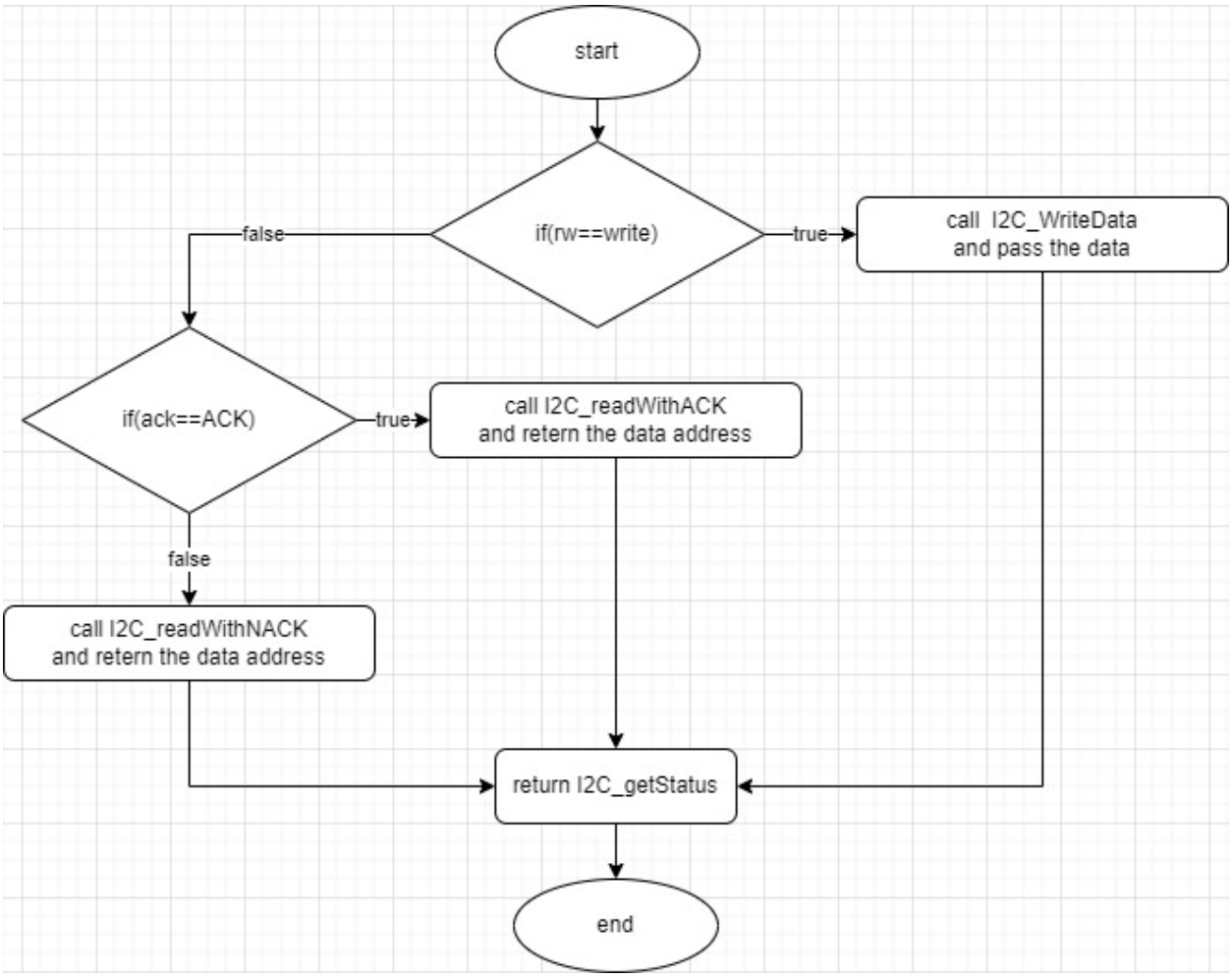
I2C_repeated_start



I2C_address_select



I2C_data_rw


start
if(rw==write)
false
true
call I2C_WriteData
and pass the data
if(ack==ACK)
true
call I2C_readWithACK
and retern the data address
false
call I2C_readWithNACK
and retern the data address
return I2C_getStatus
end