|  |
| --- |
|  |
| ATM Machine Project  BY: Momen Hassan, Ahmed Atef, and Ahmed Mohamed Hesham |
| |  |  |  | | --- | --- | --- | | 5/1/23 |  | Automotive Boot Camp | |

Contents

[LAYERED ARCHITECTURE 2](#_Toc132826437)

[PROJECT STATE MACHINE DIAGRAM 3](#_Toc132826438)

[INTRODUCTION 4](#_Toc132826439)

[MODULE, PERIPHERALS, & SUPPORTING DRIVERS DESCRIPTION 5](#_Toc132826452)

[DRIVERS’ DOCUMENTATION 6](#_Toc132826453)

[1. DIO 6](#_Toc132826454)

[2. TIMERS 8](#_Toc132826455)

[3. ADC 11](#_Toc132826456)

[4. LCD 12](#_Toc132826457)

[5. KEYPAD 15](#_Toc132826458)

[6. LM35 16](#_Toc132826459)

[7. BUZZEER 17](#_Toc132826460)

[FUNCTIONS’ FLOWCHARTS 18](#_Toc132826461)

[1. DIO 18](#_Toc132826462)

[2. TIMERS 22](#_Toc132826463)

[3. ADC 31](#_Toc132826464)

[4. LCD 33](#_Toc132826465)

[5. KEYPAD 36](#_Toc132826466)

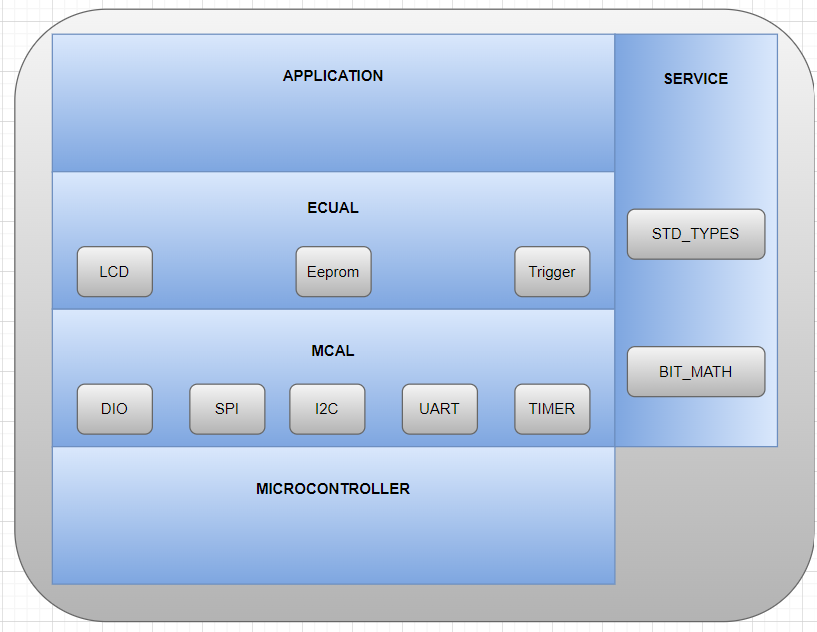
[6. LM35 38](#_Toc132826467)

[7. BUZZER 39](#_Toc132826468)

AIR CONDITIONER DESIGN

# LAYERED ARCHITECTURE

# 1-Card



# 2-ATM

# 

# 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

## 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 |

## 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 |

## 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 |

## 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 |

## 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 |

## 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);

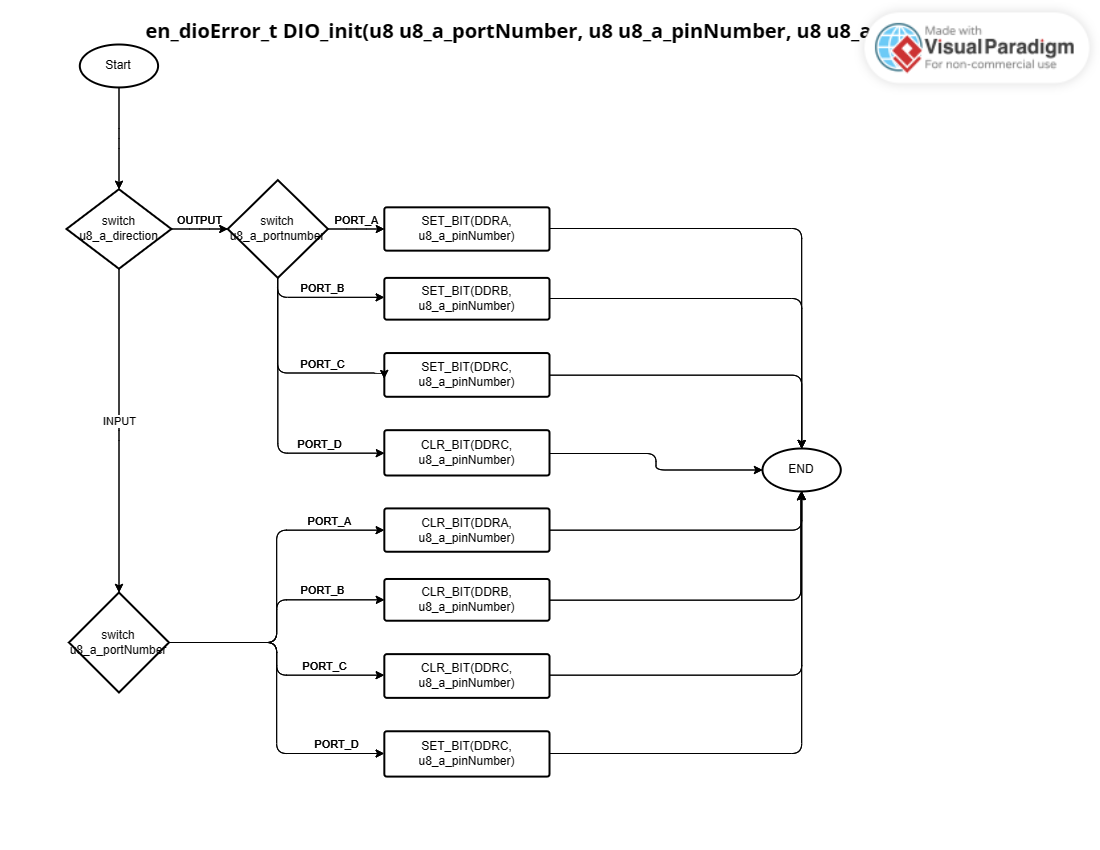
|  |  |
| --- | --- |
| Function Name | I2C\_init |
| Description | Initializes the I2C module. |
| Sync\Async | Synchronous |
| Reentrancy | Reentrant |
| Parameters (in) | None |
| Parameters (out) | None |
| Return Value | None |

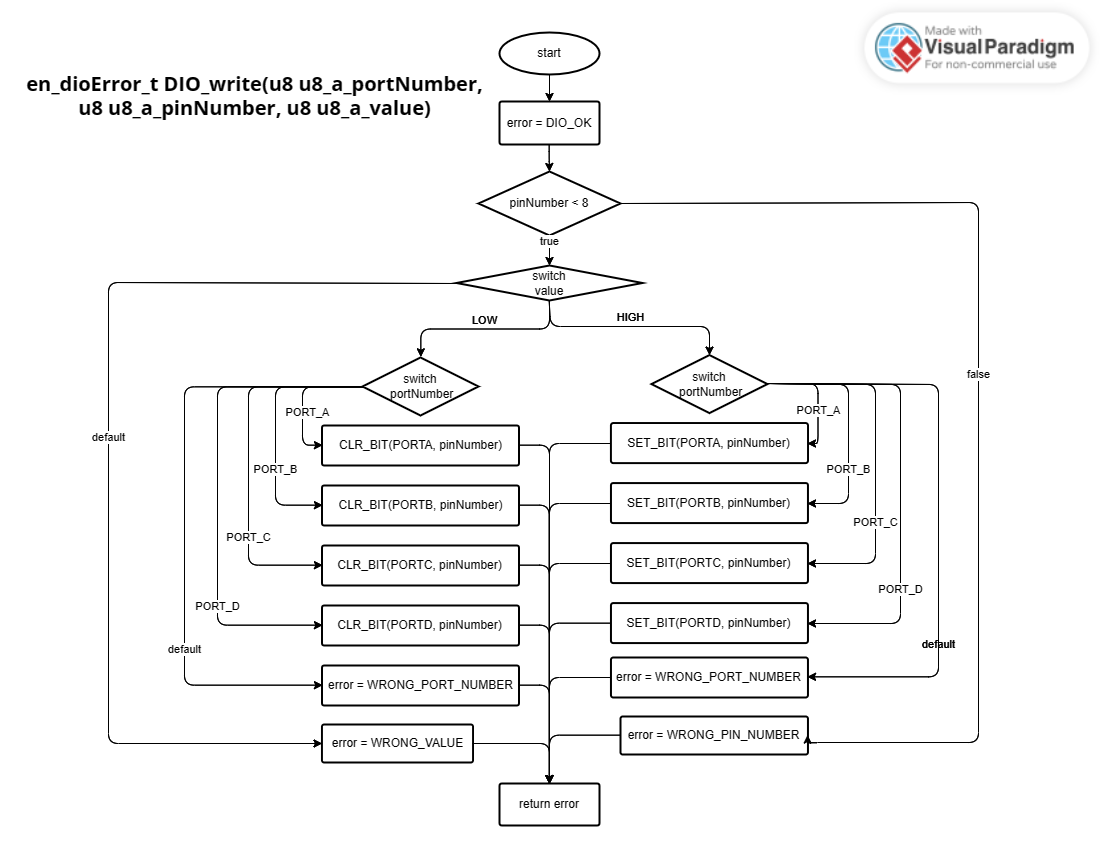
void I2C\_stop(void);

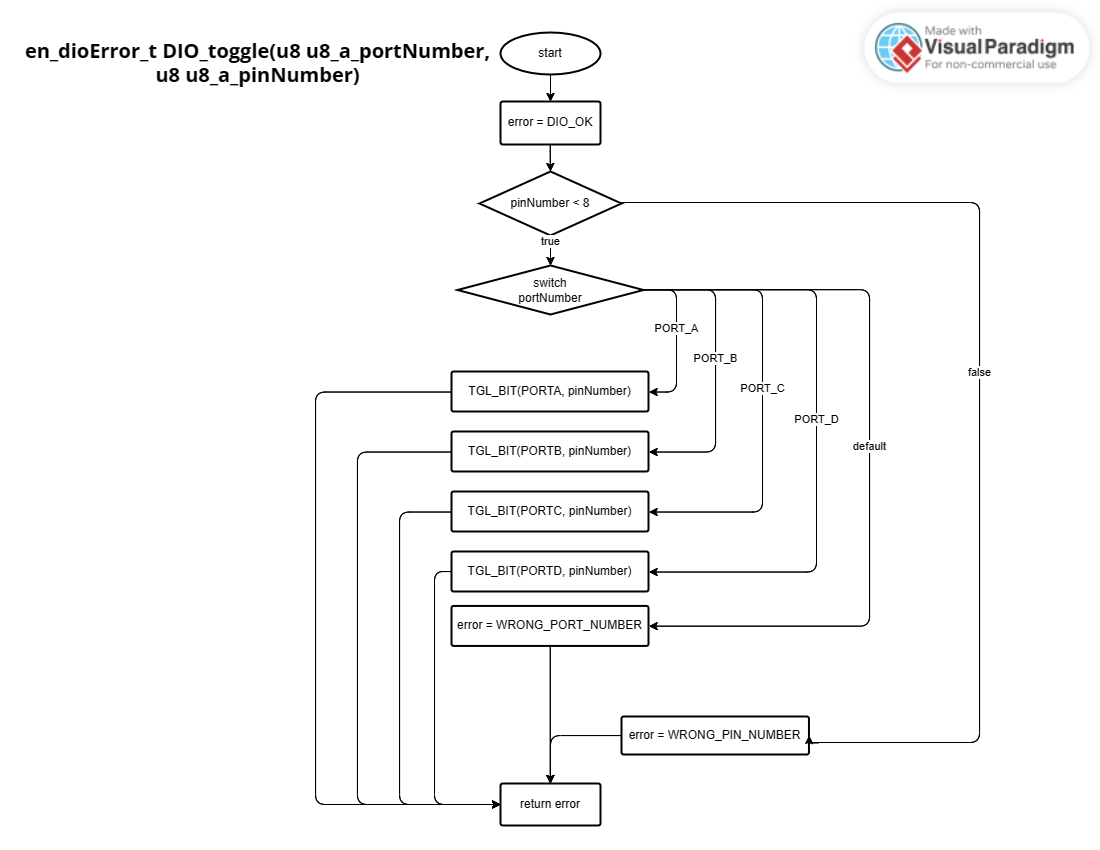
|  |  |
| --- | --- |
| Function Name | I2C\_stop |
| Description | Generates a stop condition on the I2C bus. |
| Sync\Async | Synchronous |
| Reentrancy | Reentrant |
| Parameters (in) | None |
| Parameters (out) | None |
| Return Value | None |

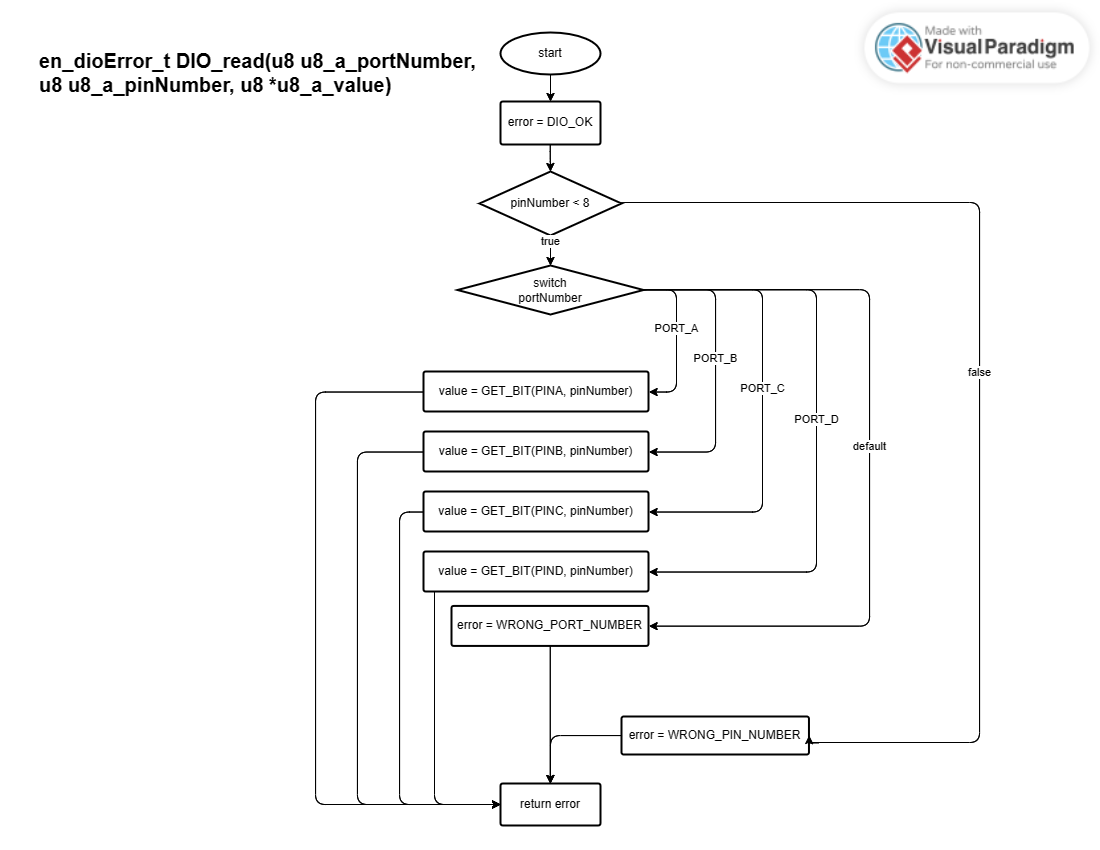
# FUNCTIONS’ FLOWCHARTS

## DIO



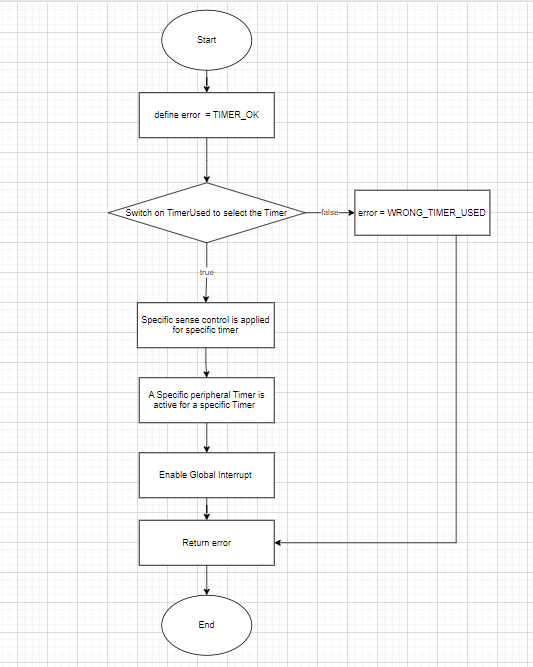




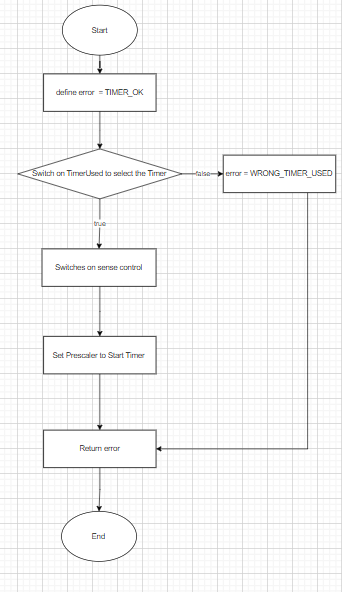


## TIMERS

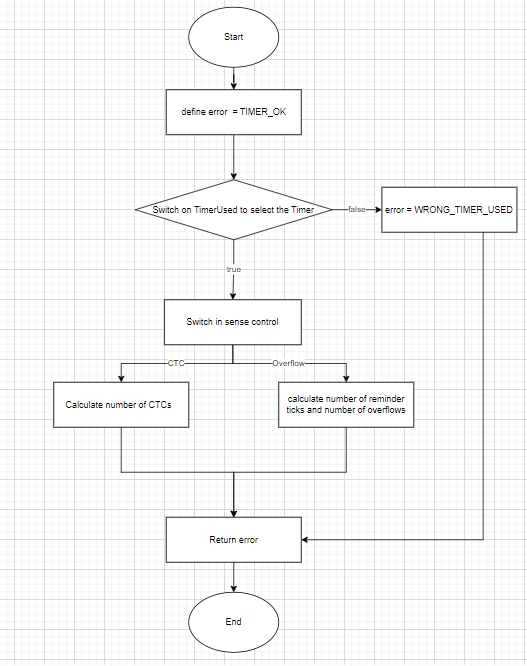
en\_timerError\_t TIMER\_init(u8 u8\_a\_timerUsed);



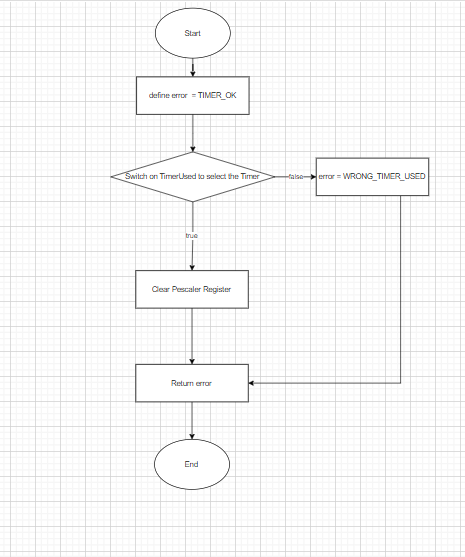
en\_timerError\_t TIMER\_start(u8 u8\_a\_timerUsed);



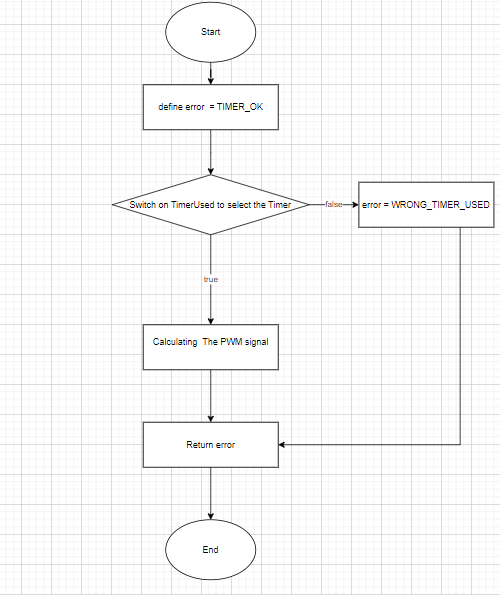
en\_timerError\_t TIMER\_setTime(u8 u8\_a\_timerUsed, u32 u32\_a\_desiredTime);



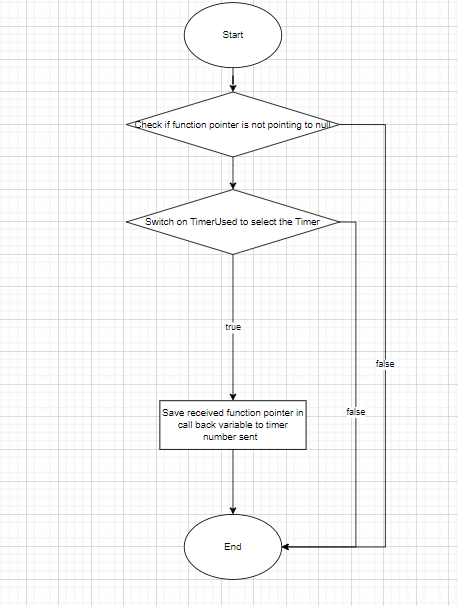
en\_timerError\_t TIMER\_stop(u8 u8\_a\_timerUsed);



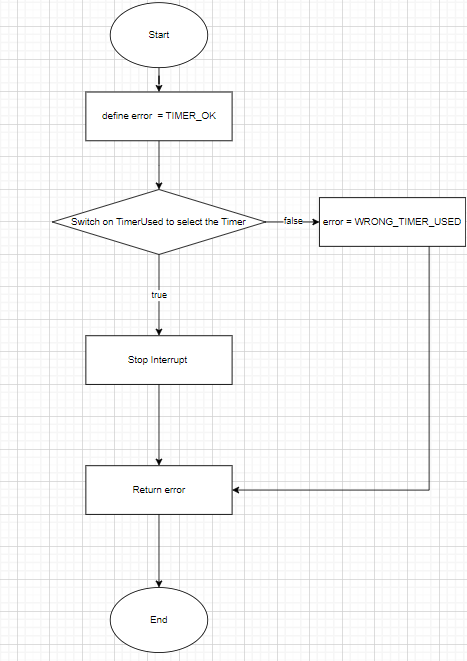
en\_timerError\_t TIMER\_pwmGenerator(u8 u8\_a\_timerUsed, u32 u32\_a\_desiredDutyCycle);



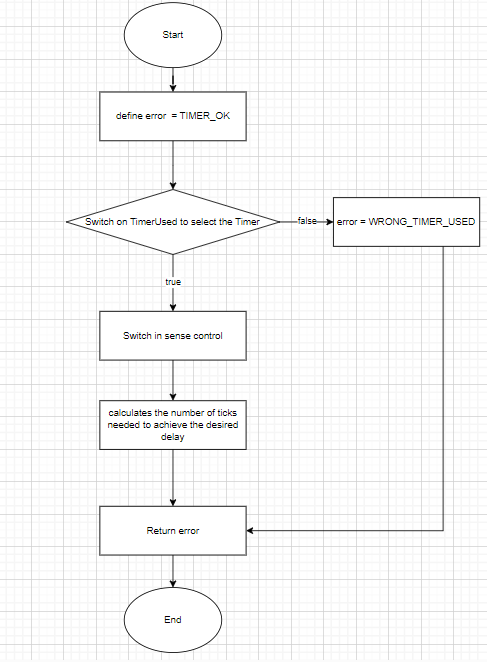
Void TIMER\_setCallBack(u8 u8\_a\_timerUsed, void (\*funPtr)(void));



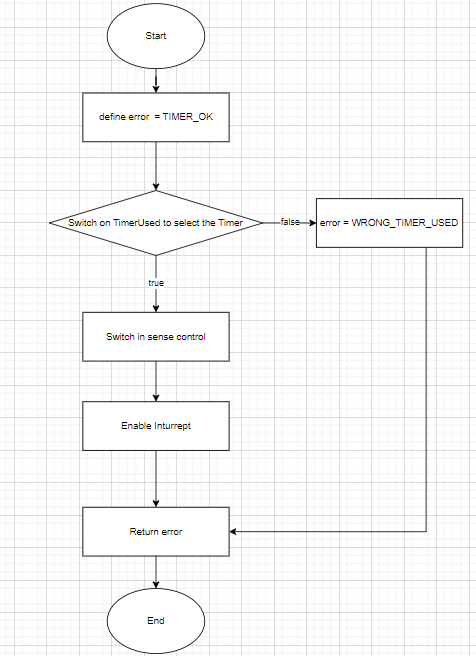
en\_timerError\_t TIMER\_stopInterrupt(u8 u8\_a\_timerUsed);



en\_timerError\_t TIMER\_delay(u8 u8\_a\_timerUsed, u32 u32\_a\_timeInMS);

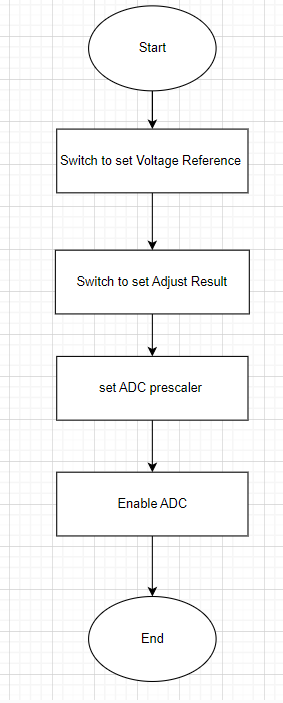


en\_timerError\_t TIMER\_enableInterrupt(u8 u8\_a\_timerUsed);

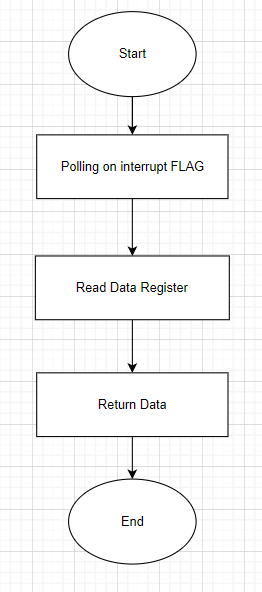


## ADC

ADC\_init()

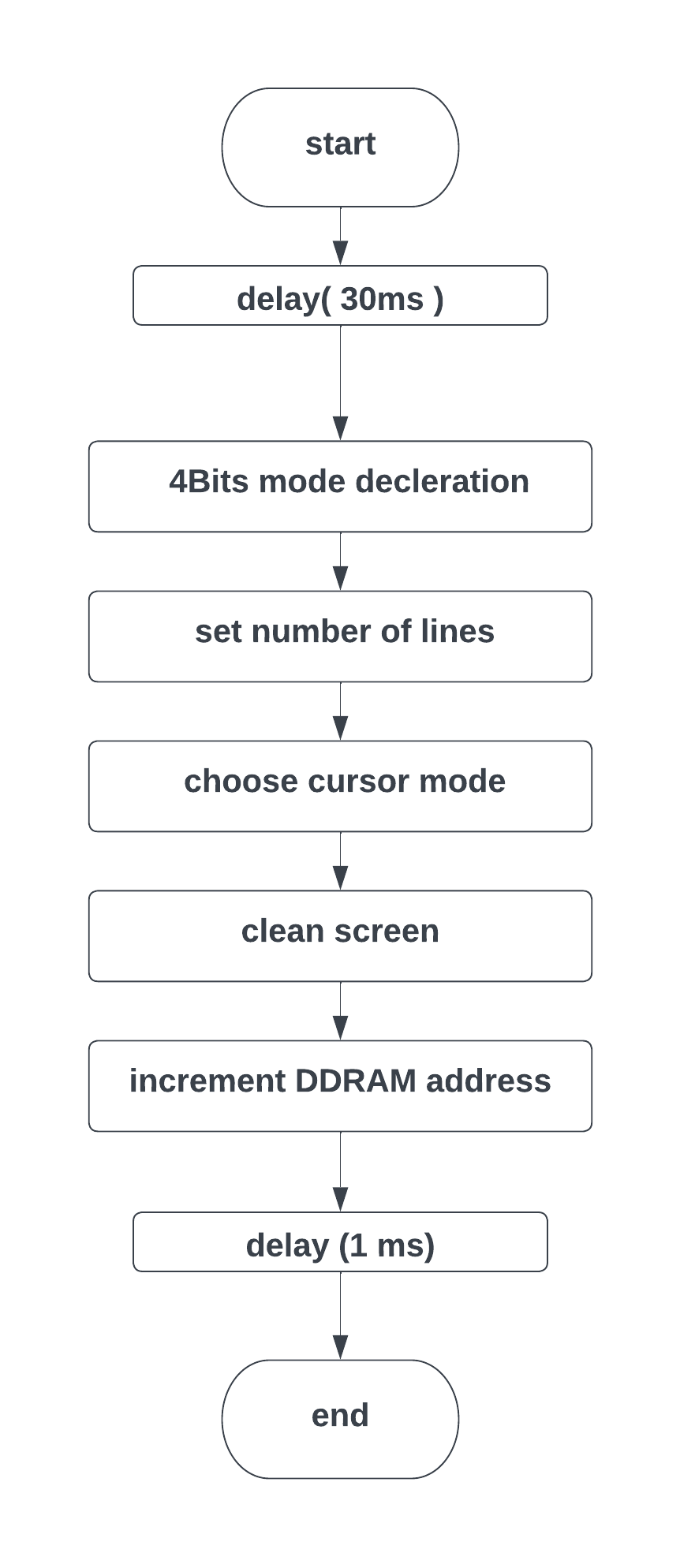


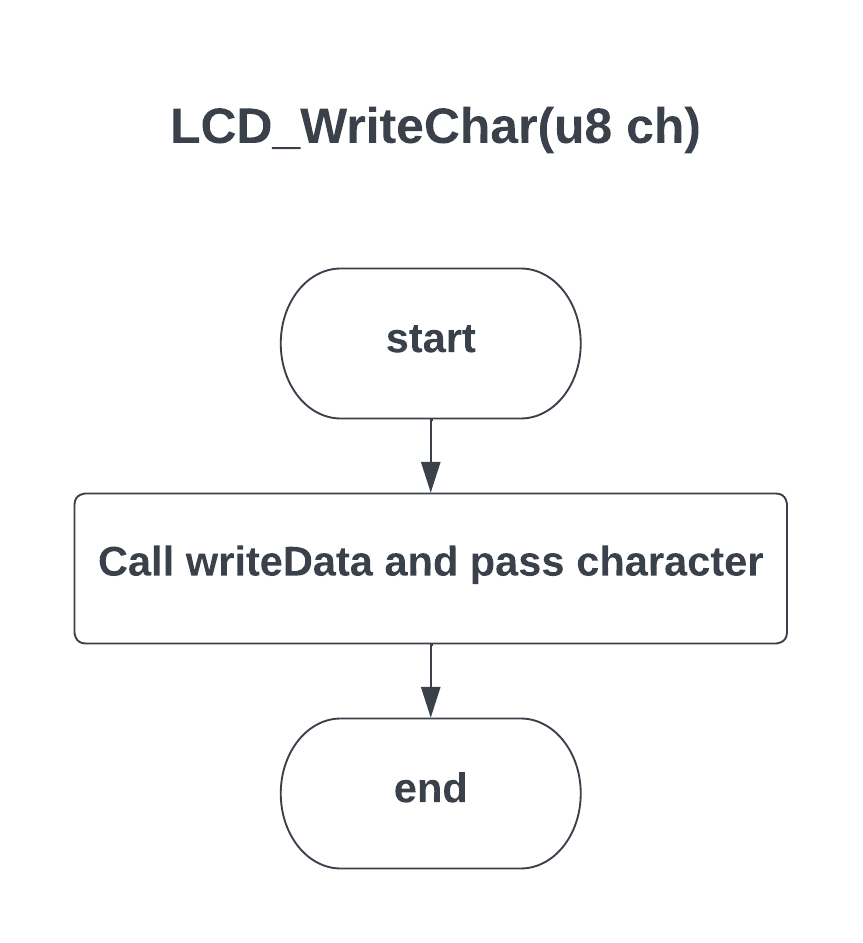
u16 ADC\_read()

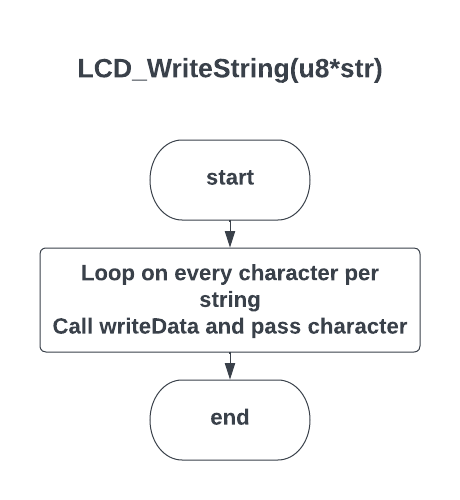


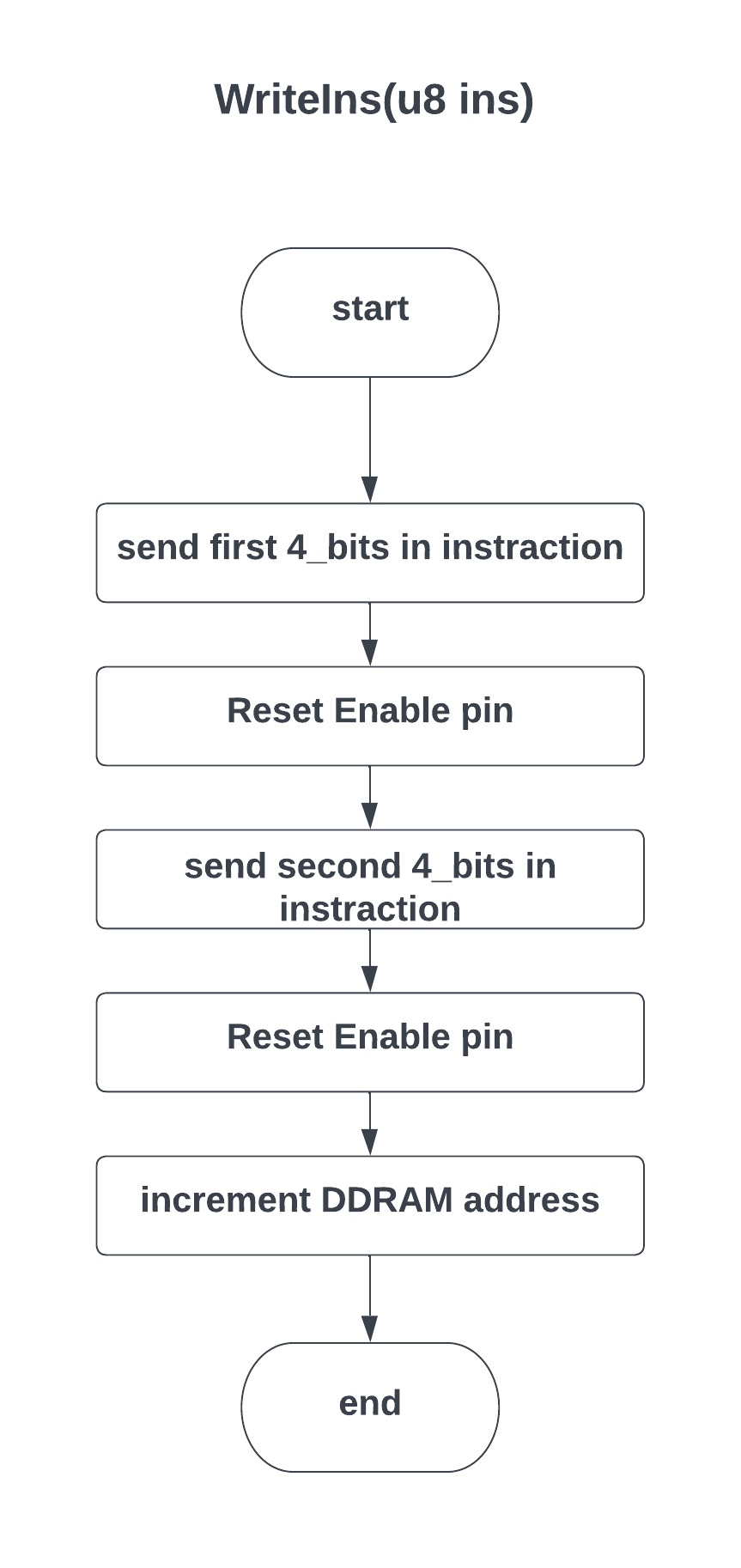
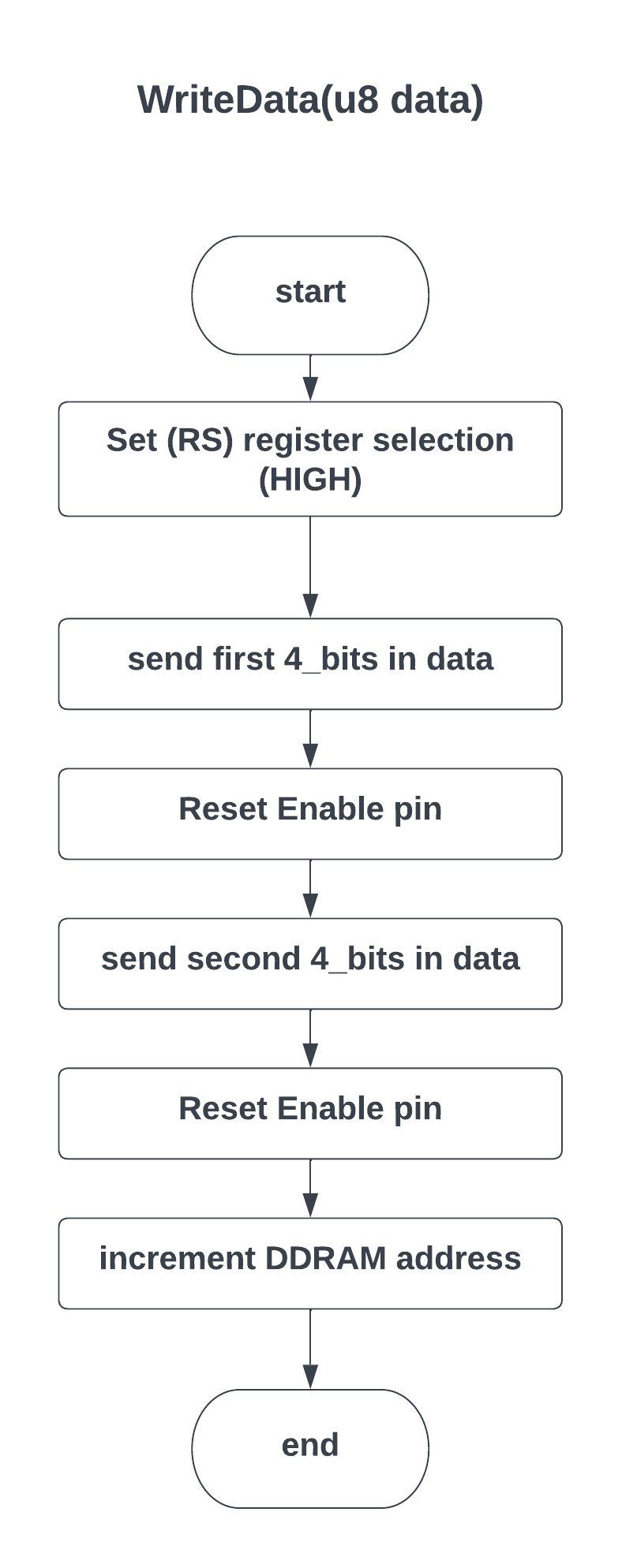
## LCD

LCD\_Init()

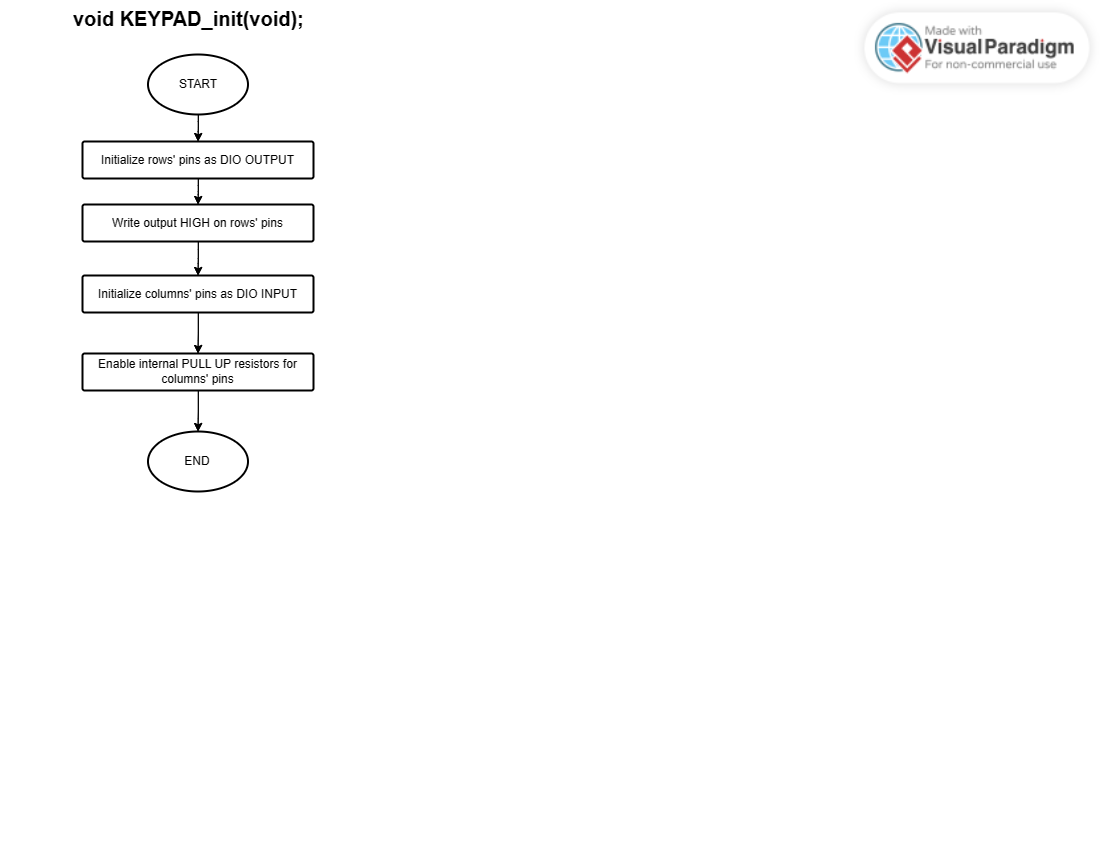


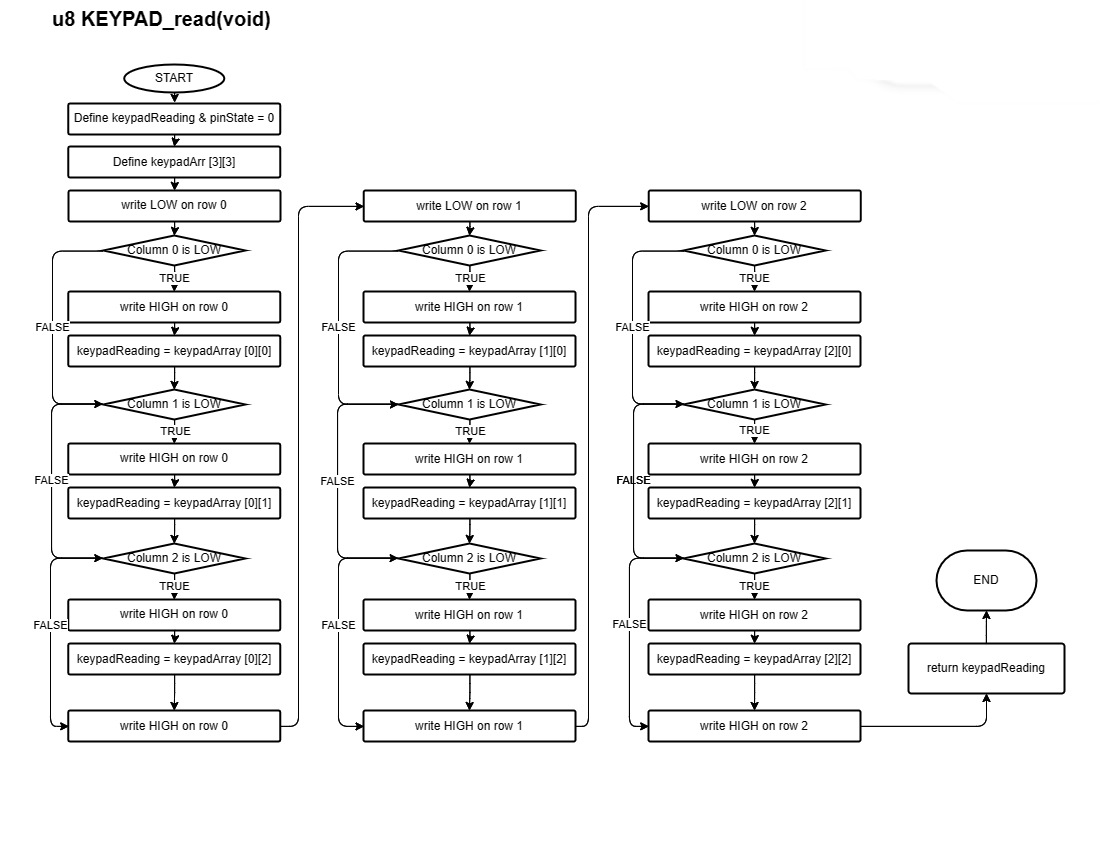




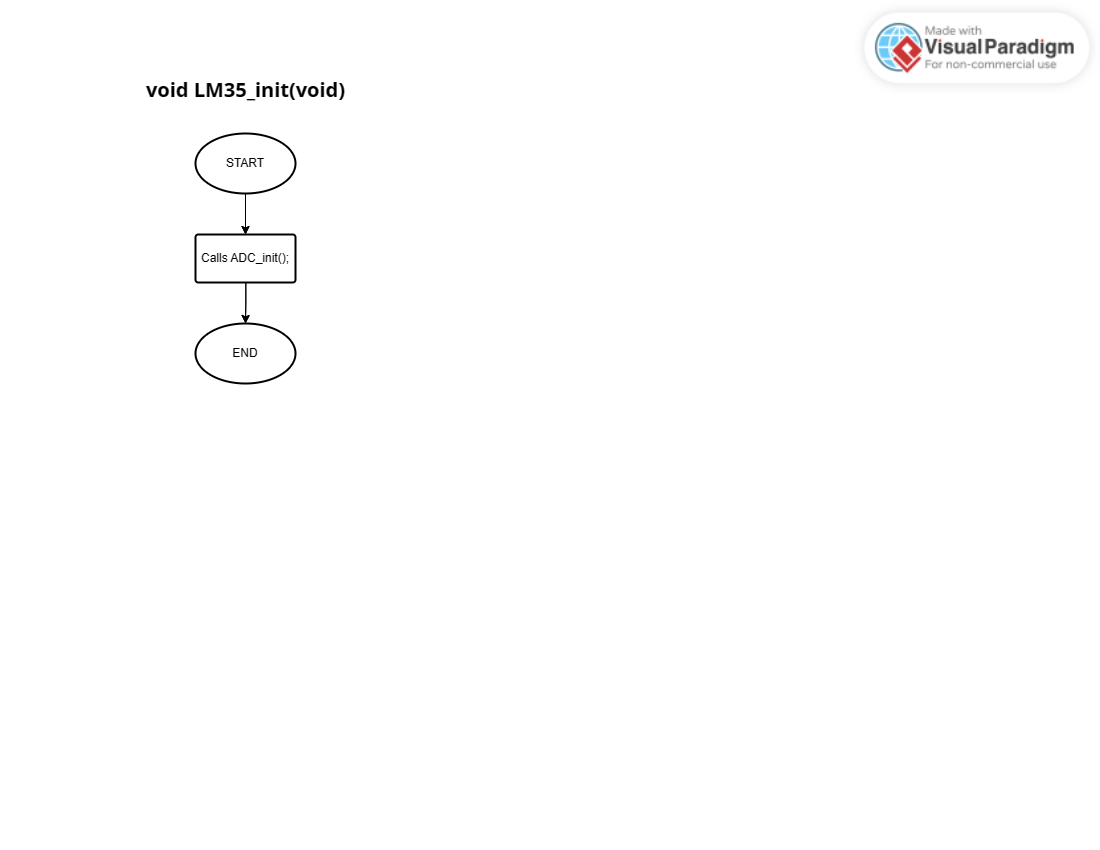


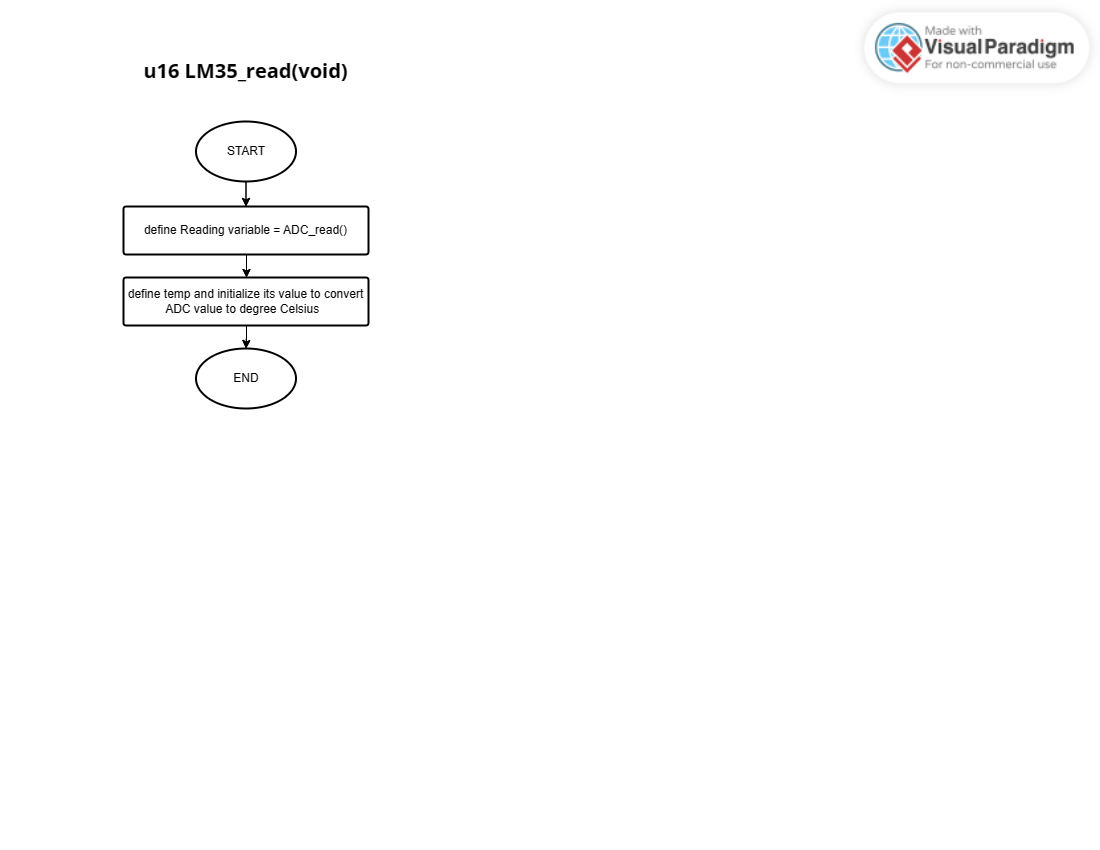
## KEYPAD





## LM35

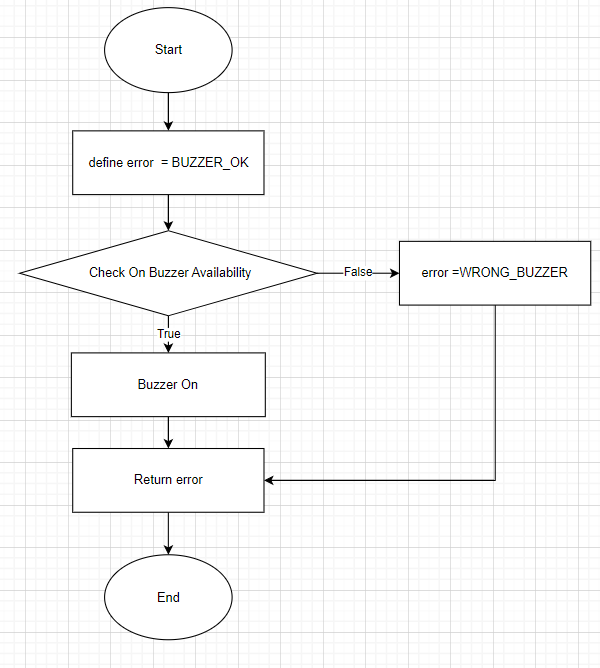


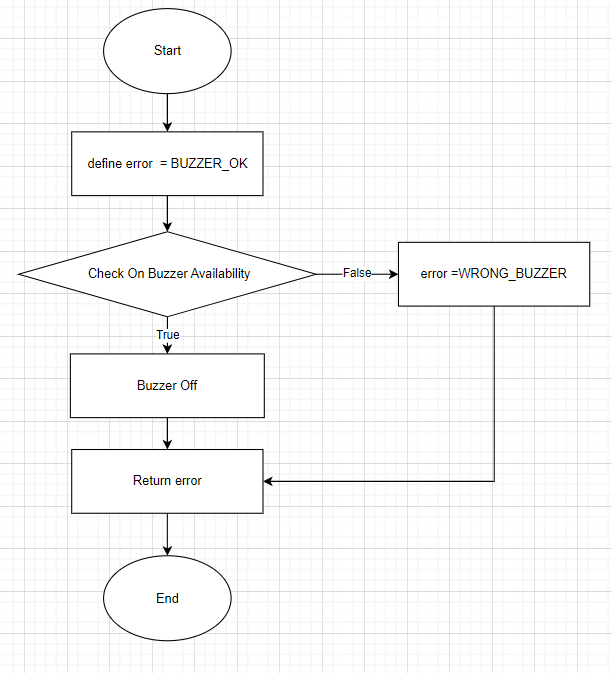


## BUZZER

BUZZER\_init()

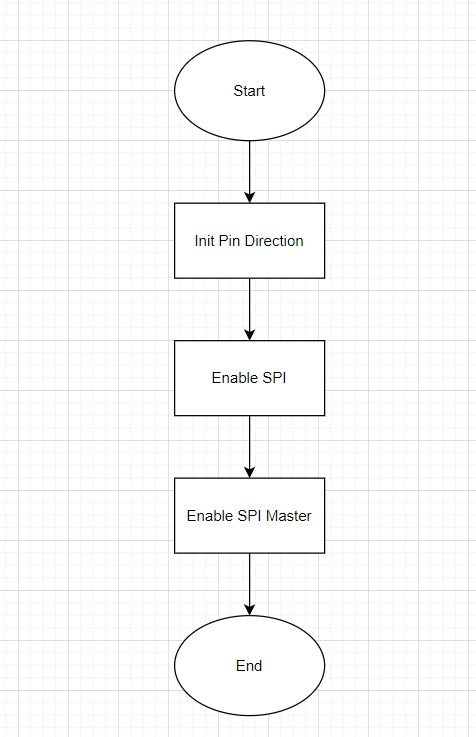
BUZZER\_on



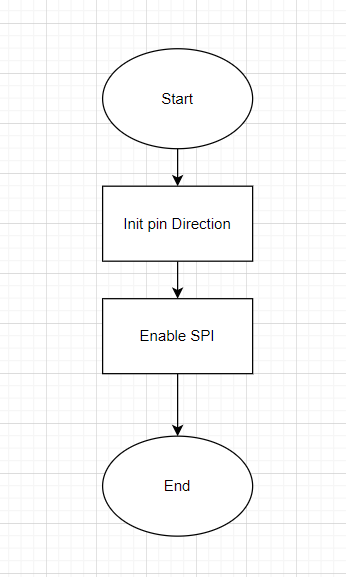
BUZZER\_OFF

## SPI

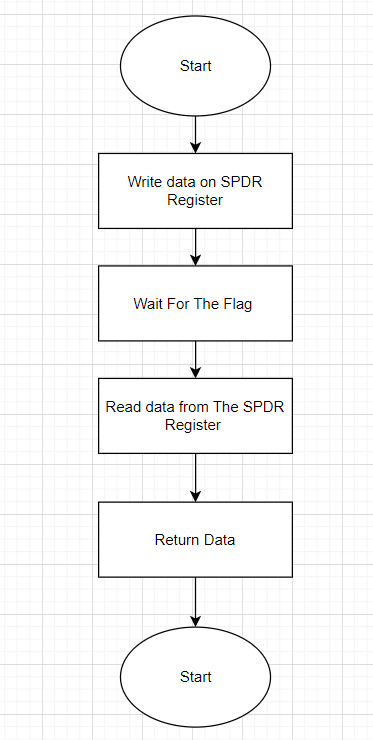
SPI\_initMaster



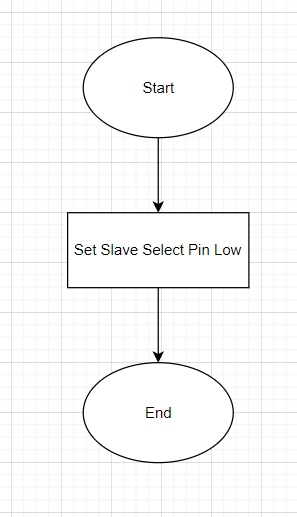
SPI\_initSlave



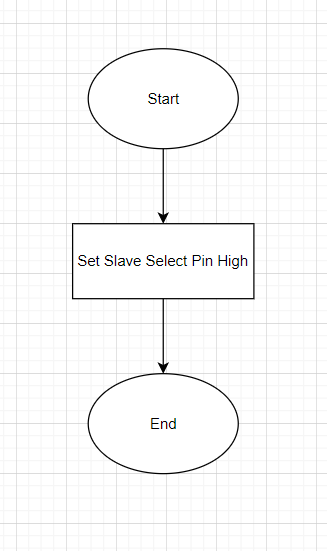
SPI\_transmitByte



SPI\_startTransmission

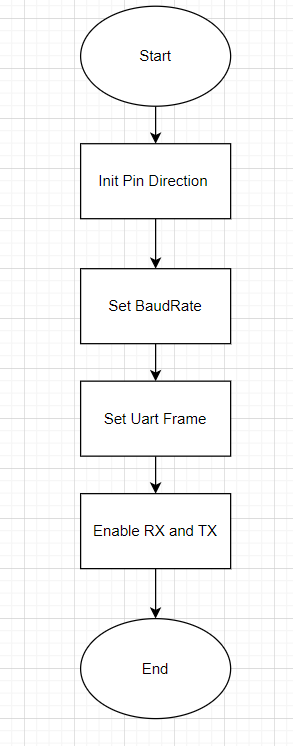


SPI\_stopTransmission

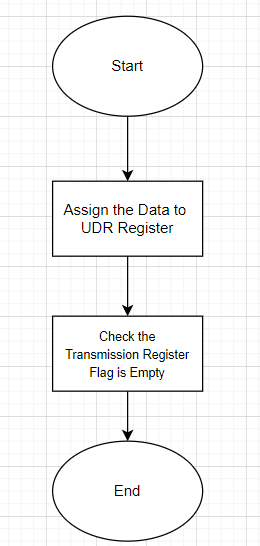


## 9.SPI

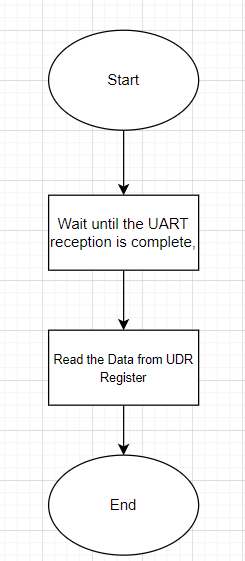
UART\_Init



UART\_SendChar

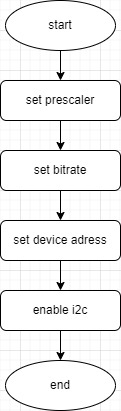


UART\_GetChar

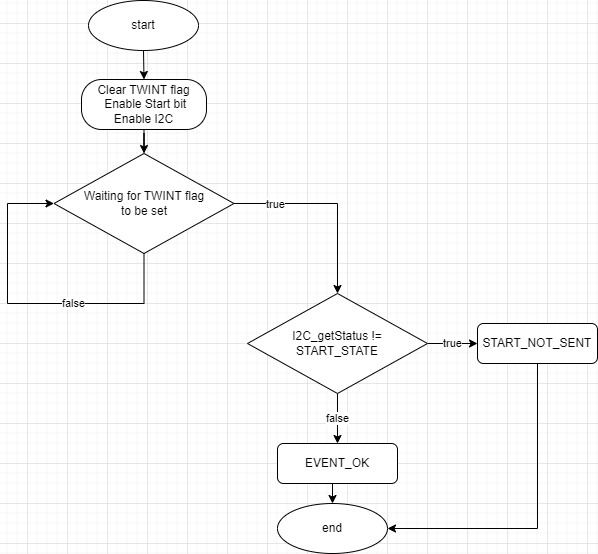


## 9.I2C

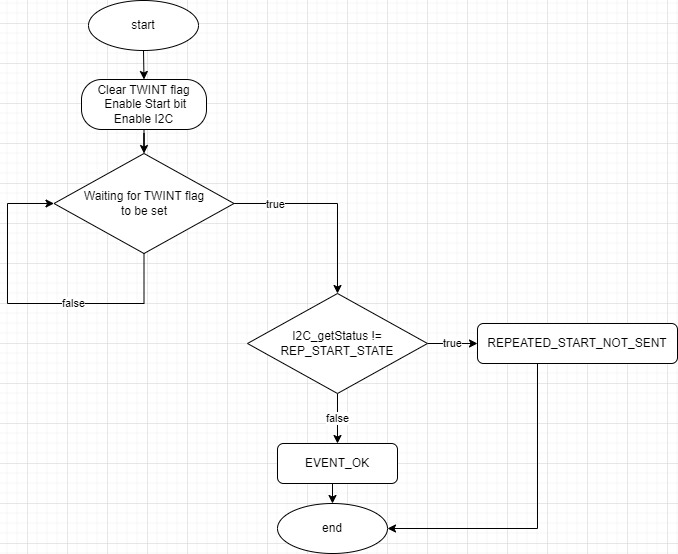
I2C\_init



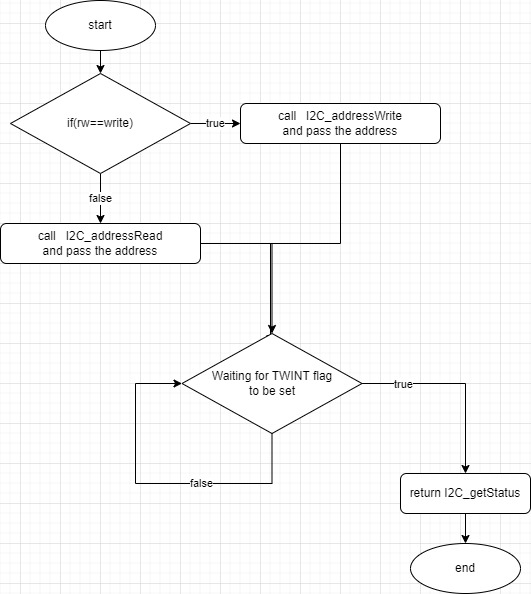
I2C\_start



I2C\_repeated\_start



I2C\_address\_select



I2C\_data\_rw

