

**Simple ATM Machine**

**Team Members:**

|  |
| --- |
| Alaa Ibrahim |
| Bassel Yasser |
| Sharpel Malek |
| Sherif Khadr |

Contents

[INTRODUCTION 1](#_Toc134422289)

[High Level Design 1](#_Toc134422290)

[**01)** **Layered Architecture** 1](#_Toc134422291)

[**02)** **Modules Description** 2](#_Toc134422292)

[**03)** **Drivers’ Documentation** 3](#_Toc134422293)

[**MCAL Layer** 3](#_Toc134422294)

[ **DIO** 3](#_Toc134422295)

[ **Timer 0** 4](#_Toc134422296)

[ **EXTINT:** 5](#_Toc134422297)

[ **I2C:** 6](#_Toc134422298)

[ **UART:** 8](#_Toc134422299)

[ **SPI:** 9](#_Toc134422300)

[**HAL Layer:** 10](#_Toc134422301)

[ **Keypad** 10](#_Toc134422302)

[ **HTimer:** 11](#_Toc134422303)

[ **HLCD** 12](#_Toc134422304)

[ **Buzzer** 13](#_Toc134422305)

[ **Button** 13](#_Toc134422306)

[ **HEXT\_INT** 14](#_Toc134422307)

[ **HSPI** 14](#_Toc134422308)

[ **EEPROM:** 15](#_Toc134422309)

[**Card\_Database Layer (CARD MCU)** 17](#_Toc134422310)

[**ATM\_MODULE Layer (ATM MCU)** 18](#_Toc134422311)

[ **ATM:** 18](#_Toc134422312)

[ **Database\_check:** 19](#_Toc134422313)

[**Application Layer:** 20](#_Toc134422314)

[ **Card MCU** 20](#_Toc134422315)

[ **ATM MCU** 20](#_Toc134422316)

[Low Level Design 21](#_Toc134422317)

[**MCAL Layer:** 21](#_Toc134422318)

[ **DIO** 21](#_Toc134422319)

[ **Timer:** 24](#_Toc134422320)

[ **EXTINT:** 26](#_Toc134422321)

[ **UART:** 29](#_Toc134422322)

[ **SPI:** 33](#_Toc134422323)

[ **I2C:** 34](#_Toc134422324)

[**HAL Layer** 38](#_Toc134422325)

[ **HTimer0** 38](#_Toc134422326)

[ **HSPI** 39](#_Toc134422327)

[ **LCD** 40](#_Toc134422328)

[ **Keypad** 47](#_Toc134422329)

[ **Buzzer** 50](#_Toc134422330)

[ **HEXTINT:** 51](#_Toc134422331)

[ **Button:** 52](#_Toc134422332)

[ **EEPROM** 53](#_Toc134422333)

[**Card\_Database Layer (CARD MCU)** 58](#_Toc134422334)

[ **Database\_check:** 62](#_Toc134422335)

[ **ATM Module** 66](#_Toc134422336)

[**Application Layer:** 71](#_Toc134422337)

[ **Card App** 71](#_Toc134422338)

[ **ATM App** 71](#_Toc134422339)

[Figure 1 ATM MCU Layered Architecture 1](file:///C:\Users\Arabtech\Desktop\Doc\Sprints_SimpleATM\Documents\SimpleATMmacine.docx#_Toc134424616)

[Figure 2 Card MCU Layered Architecture 1](file:///C:\Users\Arabtech\Desktop\Doc\Sprints_SimpleATM\Documents\SimpleATMmacine.docx#_Toc134424617)

[Figure 3 DIO\_s8SETPinDir Flow Chart 21](#_Toc134424618)

[Figure 4 DIO\_s8SETPinVal Flow chart 22](#_Toc134424619)

[Figure 5 DIO\_s8GETPinVal Flow Chart 23](#_Toc134424620)

[Figure 6 TIM0\_Init Flow Chart 24](#_Toc134424621)

[Figure 7 TIM0\_Start Flow Chart 24](#_Toc134424622)

[Figure 8 TIM0\_Stop Flow Chart 24](#_Toc134424623)

[Figure 9 TIM0 remaining Flow Charts 25](#_Toc134424624)

[Figure 10 EXTINT: Set\_Global\_Interrupt 26](#_Toc134424625)

[Figure 11 EXTINT\_Init flow chart 27](#_Toc134424626)

[Figure 12 EXTINT\_CALLBACK flow chart 28](#_Toc134424627)

[Figure 13 USART\_init flow chart 29](#_Toc134424628)

[Figure 14 USART\_receiveData flow chart 30](#_Toc134424629)

[Figure 15 USART\_sendData flow chart 30](#_Toc134424630)

[Figure 16 USART\_sendString flow chart 31](#_Toc134424631)

[Figure 17 USART\_receiveString flow chart 32](#_Toc134424632)

[Figure 18 Flow charts of SPI APIs 33](#_Toc134424633)

[Figure 19 i2c\_init\_master flow chart 34](#_Toc134424634)

[Figure 20 i2c\_start flow chart 34](#_Toc134424635)

[Figure 21 i2c\_repeated\_start flow chart 35](#_Toc134424636)

[Figure 22 i2c\_send\_slave\_address\_with\_write\_req flow chart 35](#_Toc134424637)

[Figure 23 i2c\_send\_slave\_address\_with\_read\_req flow chart 36](#_Toc134424638)

[Figure 24 i2c\_write\_byte flow chart 36](#_Toc134424639)

[Figure 25 i2c\_read\_byte flow chart 37](#_Toc134424640)

[Figure 26 i2c\_stop flow chart 37](#_Toc134424641)

[Figure 27 HTIM0\_SyncDelay Flow Chart 38](#_Toc134424642)

[Figure 28 HTIM0\_AsyncDelay and EndDelay 38](#_Toc134424643)

[Figure 29 Flow charts of HSPI APIs 39](#_Toc134424644)

[Figure 30 HLCD\_vidInit Flow Chart 40](file:///C:\Users\Arabtech\Desktop\Doc\Sprints_SimpleATM\Documents\SimpleATMmacine.docx#_Toc134424645)

[Figure 31 HLCD\_vidWritecmd Flow Chart 41](file:///C:\Users\Arabtech\Desktop\Doc\Sprints_SimpleATM\Documents\SimpleATMmacine.docx#_Toc134424646)

[Figure 32 HLCD\_vidWriteChar Flow Chart 41](file:///C:\Users\Arabtech\Desktop\Doc\Sprints_SimpleATM\Documents\SimpleATMmacine.docx#_Toc134424647)

[Figure 33 HLCD\_ClrDisplay Flow Chart 42](file:///C:\Users\Arabtech\Desktop\Doc\Sprints_SimpleATM\Documents\SimpleATMmacine.docx#_Toc134424648)

[Figure 34 HLCD\_gotoXY Flow Chart 43](#_Toc134424649)

[Figure 35 HLCD\_WriteString Flow Chart 44](#_Toc134424650)

[Figure 36 HLCD\_WriteInt Flow Chart 45](#_Toc134424651)

[Figure 37 HLCD\_vidCreatCustomChar Flow Chart 46](#_Toc134424652)

[Figure 38 HLCD\_DisplayFloat flow chart 46](file:///C:\Users\Arabtech\Desktop\Doc\Sprints_SimpleATM\Documents\SimpleATMmacine.docx#_Toc134424653)

[Figure 39 KEYPAD\_Init Flow Chart 47](#_Toc134424654)

[Figure 40 KEYPAD\_CheckRx Flow Chart 48](#_Toc134424655)

[Figure 41 GetButton Flow Chart 49](#_Toc134424656)

[Figure 42 Buzzer Init & SetState Flow Charts 50](file:///C:\Users\Arabtech\Desktop\Doc\Sprints_SimpleATM\Documents\SimpleATMmacine.docx#_Toc134424657)

[Figure 43 H\_EXTINT\_create flow chart 51](#_Toc134424658)

[Figure 44 HButton\_Init flow chart 52](#_Toc134424659)

[Figure 45 HButton\_ExtIntInit flow chart 52](file:///C:\Users\Arabtech\Desktop\Doc\Sprints_SimpleATM\Documents\SimpleATMmacine.docx#_Toc134424660)

[Figure 46 HButton\_getPinVal flow chart 52](#_Toc134424661)

[Figure 47 eeprom\_init flow chart 53](#_Toc134424662)

[Figure 48 eeprom\_write\_byte flow chart 54](#_Toc134424663)

[Figure 49 eeprom\_read\_byte flow chart 55](#_Toc134424664)

[Figure 50 eeprom\_write\_string flow chart 56](#_Toc134424665)

[Figure 51 eeprom\_read\_string flow chart 57](#_Toc134424666)

[Figure 52 APP\_terminalPinGet flow chart 58](#_Toc134424667)

[Figure 53 APP\_terminalPanGet flow chart 59](#_Toc134424668)

[Figure 54 CARD\_PinMatch flow chart 59](file:///C:\Users\Arabtech\Desktop\Doc\Sprints_SimpleATM\Documents\SimpleATMmacine.docx#_Toc134424669)

[Figure 55 SaveCardData flow chart 60](#_Toc134424670)

[Figure 56 ReadCardData flow chart 61](#_Toc134424671)

[Figure 57 isValidPanAccount flow chart 62](#_Toc134424672)

[Figure 58 isRunningAccount flow chart 63](#_Toc134424673)

[Figure 59 isValidAccountAmount flow chart 63](file:///C:\Users\Arabtech\Desktop\Doc\Sprints_SimpleATM\Documents\SimpleATMmacine.docx#_Toc134424674)

[Figure 60 isBelowMaxDailyAmount flow chart 64](file:///C:\Users\Arabtech\Desktop\Doc\Sprints_SimpleATM\Documents\SimpleATMmacine.docx#_Toc134424675)

[Figure 61 DATABASE\_checking flow chart 65](file:///C:\Users\Arabtech\Desktop\Doc\Sprints_SimpleATM\Documents\SimpleATMmacine.docx#_Toc134424676)

[Figure 62 Get\_pin flow chart 66](#_Toc134424677)

[Figure 63 get\_amount\_left flow chart 67](#_Toc134424678)

[Figure 64 PIN\_checkPinMatching flow chart 68](#_Toc134424679)

[Figure 65 deinitAtm flow chart 69](#_Toc134424680)

[Figure 66 ATM\_GetCardData & ATM\_ApprovedCard flow charts 70](file:///C:\Users\Arabtech\Desktop\Doc\Sprints_SimpleATM\Documents\SimpleATMmacine.docx#_Toc134424681)

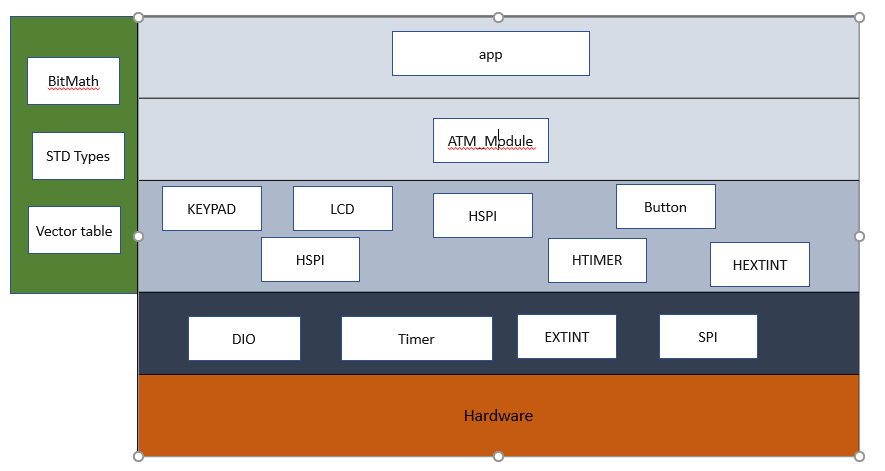
[Figure 67 Card App state machine 71](#_Toc134424682)

[Figure 68 ATM App state machine 71](file:///C:\Users\Arabtech\Desktop\Doc\Sprints_SimpleATM\Documents\SimpleATMmacine.docx#_Toc134424683)

# INTRODUCTION

# High Level Design

## **Layered Architecture**



**ATM MACHINE:**

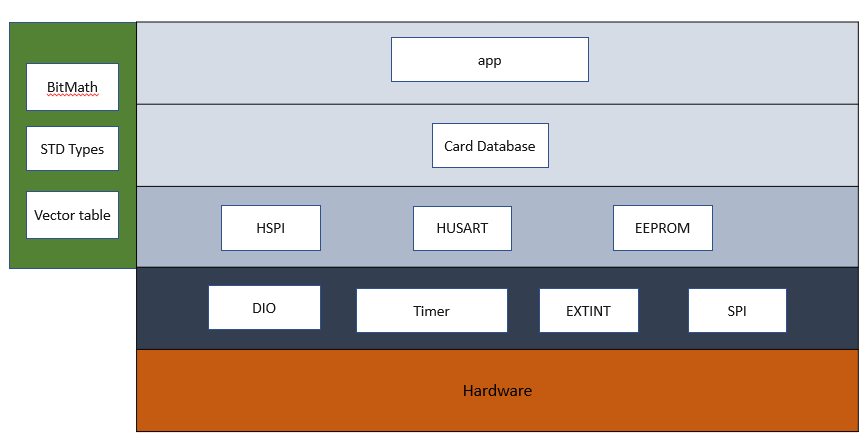
**CARD MCU**

Figure ATM MCU Layered Architecture

Figure Card MCU Layered Architecture

## **Modules Description**

**MCAL Layer:**

* **DIO:** For controlling GPIO pins
* **Timer:** Provides an interface with timer 0 low-level capabilities.
* **EXTINT:** it controls three external hardware interrupts on pins PD2, PD3, and PB2 which are referred to as INT0, INT1, and INT2 respectively
* **USART:** Enables MCU to communicate using serial protocols.
* **SPI:** Provides interface with SPI’s low-level functionalities.
* **I2C:** Handles the i2c address oriented, multi-master, multi-slave communication.

**HAL Layer:**

* **Keypad:** Deal witha set of buttons arranged in a block. The 3 x 3 matrix keypad usually is used as input in a project
* **LCD:** Use for display data
* **Buzzer:** Simple module to control a buzzer.
* **HTimer:** Provides high-level functions using the lower-level timer 0 module capabilities.
* **H\_EXT\_INT:** Provides middle ware layer between app and external interrupt in MCAL.
* **HUSART:** Provides middle ware layer between app and UART in MCAL.
* **HSPI:** Provides high-level functionalities for using SPI communication.
* **EEPROM:** Enables MCU to interface with external EEPROM using i2c.
* **Button:** Interface with button to get its state (Enter/Zero).

**Service Layer:**

* **STD\_Types:** Contains all the standard types used by all the layers.
* **BIT\_Math:** Provides bit-wise operations.
* **Vect\_table:** Contains all interrupt vectors and provides macros for dealing with general interrupt.

**Application Layer:**

Contains the main logic of the project.

## **Drivers’ Documentation**

### **MCAL Layer**

#### **DIO**

/\*

\* AUTHOR : Bassel Yasser

\* Function : DIO\_s8SETPinDir

\* Description : Set Pin Direction

\* Arguments :

\* - enPinCopy {DIO\_PINA\_0...., DIO\_PIND\_7}

\* - enPortDir {INPUT , OUTPUT}

\* Return : Sint8\_t

\*/

Sint8\_t **DIO\_s8SETPinDir** (enu\_pin enPinCopy, enu\_dir enPortDir)

/\*

\* AUTHOR : Bassel Yasser

\* Function : DIO\_s8SETPinVal

\* Description : Set Pin Value

\* Arguments :

\* - enPinCopy {DIO\_PINA\_0...., DIO\_PIND\_7}

\* - enPortDir {HIGH , LOW}

\* Return : Sint8\_t

\*/

Sint8\_t **DIO\_s8SETPinVal** (enu\_pin enPinCopy, enu\_val enPortVal)

/\*

\* AUTHOR : Bassel Yasser

\* Function : DIO\_s8GETPinVal

\* Description : Set Pin Value

\* Arguments :

\* - enPinCopy {DIO\_PINA\_0...., DIO\_PIND\_7}

\* - pu8Val address of variable that u want to save value on it

\* Return : Sint8\_t

\*/

Sint8\_t **DIO\_s8GETPinVal** (enu\_pin enPinCopy, Uint8\_t\* pu8Val)

#### **Timer 0**

/\*\*

\* \brief Initialize the timer with given mode

\* \param u8\_a\_Mode

\* \return en\_TIMErrorState\_t

\*/

en\_TIMErrorState\_t TIM0\_voidInit(en\_TIMMode\_t u8\_a\_Mode);

/\*\*

\* \brief Start the timer clock after prescaling it with given value

\* \param u8\_a\_prescaler

\* \return en\_TIMErrorState\_t

\*/

en\_TIMErrorState\_t TIM0\_Start(en\_TIM\_CLK\_SELECT\_t u8\_a\_prescaler);

/\*\*

\* \brief Function to stop timer 0

\* \return void

\*/

void TIM0\_Stop();

/\*\*

\* \brief Set the timer to start from a certain value

\* \param u8\_a\_FlagValue The value to start the timer from

\* \return void

\*/

void TIM0\_SetValue(Uchar8\_t u8\_a\_startValue);

/\*\*

\* \brief Function to get the value of the overflow flag of timer 0

\* \param u8\_a\_FlagValue reference to a variable to store flag value \*

\* \return en\_TIMErrorState\_t

\*/

en\_TIMErrorState\_t TIM0\_GetOVF(Uchar8\_t\* u8\_a\_FlagValue);

/\*\*

\* \brief Function to clear timer 0 overflow flag

\* \return void

\*/

void TIM0\_ClearOVF(void);

/\*\*

\* \brief Function to get the timer state (running/stopped)

\* \param u8\_a\_State reference to a variable to store timer state

\* \return en\_TIMErrorState\_t

\*/

en\_TIMErrorState\_t TIM0\_GetState(en\_TIMState\_t\* u8\_a\_State);

/\*\*

\* \brief Function to set a function to call when the timer0

\* Overflow Interrupt is triggered

\* \param pv\_a\_CallbackFn reference to the function to call

\* \return en\_TIMErrorState\_t

\*/

#### **EXTINT:**

// EXT\_INT TYPEDEFS

typedef enum EN\_EXTINT\_ERROR {

EXTINT\_OK=0,

EXTINT\_NOT\_OK

}EN\_EXTINT\_ERROR;

typedef enum EN\_Sence\_Control {

LOW\_LEVEL=0,

FALLING\_EDGE,

RISING\_EDGE,

ANY\_LOGICAL\_CHANGE

}EN\_Sence\_Control;

typedef enum EN\_EXINT\_NUMBER{

EXTINT0=0,

EXTINT1,

EXTINT2,

}EN\_EXINT\_NUMBER;

typedef enum EN\_GLOBAL\_INT{

DISABLE=0,

ENABLE

}EN\_GLOBAL\_INT;

// EXT\_INT prototypes

/\*

Description : This function initializes the GLOBAL\_INTERRUPT

ARGS : takes the state ( ENABLE OR DISABLE )

return : return EXTINT\_OK if the PIN initializes correctly, EXTINT\_NOT\_OK otherwise

\*/

EN\_EXTINT\_ERROR SET\_GLOBAL\_INTERRUPT(EN\_GLOBAL\_INT state);

/\*

Description : This function initializes the external interrupt number and it's detecting type

ARGS : takes the EXINT\_NUMBER( INT0,INT1 OR INT2) and sense control.

return : return EXTINT\_OK if the EXINT\_NUMBER initializes correctly, EXTINT\_NOT\_OK otherwise

\*/

EN\_EXTINT\_ERROR EXTINT\_init(EN\_EXINT\_NUMBER INTx ,EN\_Sence\_Control INTxSense);

/\*

Description : This function takes the external interrupt number and initialize call back function.

ARGS : takes the EXINT\_NUMBER( INT0,INT1 OR INT2) and pointer to the function we want to execute.

return : return EXTINT\_OK if the EXINT\_NUMBER initializes correctly, EXTINT\_NOT\_OK otherwise

\*/

EN\_EXTINT\_ERROR EXTINT\_CallBack(EN\_EXINT\_NUMBER INTx,void(\*ptrfunc)(void));

#### **I2C:**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 01- i2c\_init\_master() \*/

/\* ----------------------------------------------------------------------------------

\* @func : I2C master Initialization

\* @in : void

\* @out : void

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** **i2c\_init\_master**(**void**);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 02- i2c\_init\_slaver() \*/

/\* ----------------------------------------------------------------------------------

\* @func : I2C slave Initialization

\* @in : void

\* @out : void

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** **i2c\_init\_slave** (**void**);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 03- i2c\_start() \*/

/\* ----------------------------------------------------------------------------------

\* @func : Send start condition

\* @in : void

\* @out : void

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** **i2c\_start**(**void**);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 04- i2c\_repeated\_start() \*/

/\* ----------------------------------------------------------------------------------

\* @func : Send repeated start condition

\* @in : void

\* @out : void

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** **i2c\_repeated\_start**(**void**);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 05- i2c\_send\_slave\_address\_with\_write\_req() \*/

/\* ----------------------------------------------------------------------------------

\* @func : send write request from master to slave

\* @in[1] : slave\_address

\* - Slave address that you want to write on it

\* @out : void

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** **i2c\_send\_slave\_address\_with\_write\_req**(Uint8\_t slave\_address);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 06- i2c\_send\_slave\_address\_with\_read\_req() \*/

/\* ----------------------------------------------------------------------------------

\* @func : send read request from master to slave

\* @in[1] : slave\_address

\* - Slave address that you want to read from it

\* @out : void

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** **i2c\_send\_slave\_address\_with\_read\_req**(Uint8\_t slave\_address);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 07- i2c\_write\_byte() \*/

/\* ----------------------------------------------------------------------------------

\* @func : Write data to slave

\* @in[1] : byte

\* - write data that u need to send

\* @out : void

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** **i2c\_write\_byte**(Uint8\_t byte);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 08- i2c\_read\_byte() \*/

/\* ----------------------------------------------------------------------------------

\* @func : read data from slave

\* @in : void

\* @out : returned data

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Uint8\_t **i2c\_read\_byte**(**void**);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* 09- i2c\_stop() \*/

/\* ----------------------------------------------------------------------------------

\* @func : Send Stop condition

\* @in : void

\* @out : void

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** **i2c\_stop(void**);

#### **UART:**

typedef enum EN\_USART\_ERROR{

USART\_OK=0,

USART\_NOT\_OK

}EN\_USART\_ERROR;

/\*

Name : USART\_init

Description : This function initializes USART Module with selected options in USART.cfg file.

Args : Void (all options defined as macros).

return : Std\_ReturnType (E\_OK) if Module initializes Correctly, (E\_NOT\_OK) otherwise.

\*/

EN\_USART\_ERROR USART\_init(void);

/\*

Name : USART\_sendData

Description : This function Send Data To Receiver (we can change size of data from USART.cfg file) we select 8 bits data size.

Args : take one argument (data to be sent) must be same size as the size we select in Initialization function.

return : Std\_ReturnType (E\_OK) if module sent data Correctly, (E\_NOT\_OK) otherwise.

\*/

EN\_USART\_ERROR USART\_sendData(Uchar8\_t data);

/\*

Name : USART\_receiveData

Description : This function Receive Data from sender (we can change size of data from USART.cfg file) we select 8 bits data size.

Args : void

return : data received. must be same size as the size we select in Initialization function.

\*/

Uchar8\_t USART\_receiveData(void);

/\*

Name : USART\_sendSTRING

Description : This function send array of data To Receiver. (we can change size of data from USART.cfg file) we select 8 bits data size.

Args : pointer to the array of data to be sent

return : void

\*/

void USART\_sendSTRING(Uchar8\_t \* str);

/\*

Name : USART\_receiveSTRING

Description : This function receive data from sender and store it in array.

Args : pointer to the array to store data received in it and size of data.

return : void

\*/

void USART\_receiveSTRING(Uchar8\_t \* str,Uchar8\_t size);

#### **SPI:**

/\*\*

\* (Author: Alaa Hisham)

\* \brief Initialize the MCU as the Master

\* in SPI communication

\* \return void

\*/

void SPI\_MasterInit(void);

/\*\*

\* (Author: Alaa Hisham)

\* \brief Initialize the MCU as a slave

\* in SPI communication

\*

\* \return void

\*/

void SPI\_SlaveInit(void);

/\*\*

\* (Author: Alaa Hisham)

\* \brief Sets the value of the SPI data register

\* \param u8\_a\_data: Desired value

\* \return void

\*/

void SPI\_SetValue(Uchar8\_t u8\_a\_data);

/\*\*

\* (Author: Alaa Hisham)

\* \brief Exchange a letter with selected slave

\* \param u8\_a\_character: Character to send

\* \param pu8\_a\_receivedChar: Pointer to character to store received value

\* \return en\_SPI\_ErrorState\_t

\*/

en\_SPI\_ErrorState\_t SPI\_TranscieveChar(Uchar8\_t u8\_a\_character, Uchar8\_t\* pu8\_a\_receivedChar);

/\*\*

\* (Author: Alaa Hisham)

\* \brief Exchange a letter with selected slave

\* \param u8\_a\_character: Character to send

\* \param pu8\_a\_receivedChar: Pointer to character to store received value

\* \return en\_SPI\_ErrorState\_t

\*/

en\_SPI\_ErrorState\_t SPI\_SlaveTranscieve(Uchar8\_t u8\_a\_character, Uchar8\_t\* pu8\_a\_receivedChar);

/\*\*

\* (Author: Alaa Hisham)

\* \brief Set a notification function for the SPI Interrupt

\* \param pv\_a\_CallbackFn: reference to the function to callback

\* when the SPI Interrupt is triggered

\* \return en\_SPI\_ErrorState\_t

\*/

en\_SPI\_ErrorState\_t SPI\_SetCallback(void (\*pv\_a\_CallbackFn)(void));

### **HAL Layer:**

#### **Keypad**

// Macros

#define R1 DIO\_PINC\_2

#define R2 DIO\_PINC\_3

#define R3 DIO\_PINC\_4

#define C1 DIO\_PINC\_5

#define C2 DIO\_PINC\_6

#define C3 DIO\_PINC\_7

// user defined datatypes

typedef enum EN\_KEYPAD\_BTNS

{

KEY\_1=0,

KEY\_2,

KEY\_3,

KEY\_4T,

KEY\_5,

KEY\_6,

KEY\_7,

KEY\_8,

KEY\_9,

KEY\_NOTHING

}EN\_KEYPAD\_BTNS;

// functions prototypes

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Name : KEYPAD\_init()

Description : This Function Initializes keypad pins (Rows are outputs & Columns are inputs).

ARGS : void

return : void

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void KEYPAD\_init(void);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

Name : KEYPAD\_GetButton

Description : This Function loops over other three functions (Checks (R1,R2,R3)).

ARGS : void

return : the pressed key or Nothing pressed

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

EN\_KEYPAD\_BTNS KEYPAD\_GetButton(void);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

Name : KEYPAD\_checkR1 , KEYPAD\_checkR2, KEYPAD\_checkR3

Description : functions are checking the entire row if it pressed or not.

ARGS : void

return : the pressed key or Nothing pressed

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

EN\_KEYPAD\_BTNS KEYPAD\_checkR1(void);

EN\_KEYPAD\_BTNS KEYPAD\_checkR2(void);

EN\_KEYPAD\_BTNS KEYPAD\_checkR3(void);

#### **HTimer:**

/\*\*

\* \brief Generate Synchronous delay (busy waiting)\*

\* \param Copy\_delayTime Desired delay

\* \param Copy\_timeUnit Time units (Seconds, mSeconds, uSeconds)

\*

\* \return en\_HTIMErrorState\_t

\*/

en\_HTIMErrorState\_t TIM0\_SyncDelay(Uint32\_t u32\_a\_delay, en\_timeUnits\_t u8\_a\_timeUnit);

/\*\*

\* \brief Generates delay asynchronously

\* \param u32\_a\_delay desired delay

\* \param u8\_a\_timeUnit delay time units

\* \param Copy\_pvCallbackFn function to call when delay is complete

\*

\* \return en\_TIMErrorState\_t

\*/

en\_HTIMErrorState\_t TIM0\_AsyncDelay(Uint32\_t u32\_a\_delay, en\_timeUnits\_t u8\_a\_timeUnit, void (\*Copy\_pvCallbackFn)(void));

/\*\*

\* \brief Function to end a delay asynchronously

\* To Stop Async Delay: No Restrictions

\* To Stop Sync Delay: should only be called in an ISR/Callback function

\*

\* \return void

\*/

void TIM0\_AsyncEndDelay();

#### **HLCD**

/\*

\* function : HLCD\_vidInit

\* description : func to set LCD initialization

\* input param : void

\* return : void

\* \*/

**void** **HLCD\_vidInit**(**void**)

/\*

\* function : HLCD\_vidWritecmd

\* description : func to configure some commands on lcd

\* input param :

\* u8commandCopy --> take lcd cmd instructions from instruction table <https://components101.com/sites/default/files/component\_datasheet/16x2%20LCD%20Datasheet.pdf>

\* return : void

\* \*/

**void** **HLCD\_vidWritecmd**(Uint8\_t u8commandCopy)

/\*

\* function : HLCD\_vidWriteChar

\* description : func to write char on lcd

\* input param : u8CharCopy -> take ascii code of char or char address on CGROM

\* return : void

\* \*/

**void** **HLCD\_vidWriteChar**(Uint8\_t u8CharCopy)

/\*

\* function : HLCD\_ClrDisplay

\* description : func to clear anything on lcd

\* input param : void

\* return : void

\* \*/

**void** **HLCD\_ClrDisplay**(**void**)

/\*

\* function : HLCD\_gotoXY

\* description : func to determine position which char print at this position on lcd ### NOTE : (2rows x 16coloms)

\* input param :

\* row -> take row number 0 or 1

\* pos -> take colom number from 0 ~ 16

\* return : void

\* \*/

**void** **HLCD\_gotoXY**(Uint8\_t row, Uint8\_t pos)

/\*

\* function : HLCD\_WriteString

\* description : func to write string on lcd

\* input param : str --> which take string as argument

\* return : void

\* \*/

**void** **HLCD\_WriteString**(Uint8\_t\* str)

/\*

\* function : HLCD\_WriteInt

\* description : func to write integer number on lcd

\* input param : number --> which take number as argument

\* return : void

\* \*/

**void** **HLCD\_WriteInt**(Uint32\_t number)

/\*

\* function : HLCD\_vidCreatCustomChar

\* description : func to store new patterm on CGRAM

\* input param :

\* pu8custom -> take pointer to array which having LCD Custom Character Generated data ### take only 8 characters

\* u8Location -> determine location on CGRAM [0 ~ 8]

\* return : void

\* \*/

**void** **HLCD\_vidCreatCustomChar**(Uint8\_t\* pu8custom, Uint8\_t u8Location)

/\*\*

\* (Author: Alaa Hisham)

\* \brief Display floating point number on LCD

\*

\* \param f32\_a\_number: number to display

\* \return void

\*/

void HLCD\_DisplayFloat(float32\_t f32\_a\_number);

#### **Buzzer**

/\*\*

\* \brief Initialize buzzer pin as output

\* \param pst\_a\_buzzer reference to buzzer

\* \return void

\*/

void BUZ\_Init(st\_Buzzer\_t\* pst\_a\_buzzer);

/\*\*

\* \brief Turn the buzzer on/off

\* \param pst\_a\_buzzer reference to buzzer

\* \param u16\_a\_state BUZ\_ON (or) BUZ\_OFF

\* \return en\_BuzzerErrorState\_t

\*/

en\_BuzzerErrorState\_t BUZ\_SetState(st\_Buzzer\_t\* pst\_a\_buzzer, en\_BuzzerState\_t en\_a\_state);

#### **Button**

/\*

\* AUTHOR : Bassel Yasser Mahmoud

\* FUNCTION : HButton\_Init

\* DESCRIPTION : Initialize specified pin as input and pull up

\* RETURN : enu\_buttonError\_t {BUTTON\_NOK, BUTTON\_OK}

\*/

enu\_buttonError\_t **HButton\_Init**(enu\_pin en\_pinx);

/\*

\* AUTHOR : Bassel Yasser Mahmoud

\* FUNCTION : HButton\_ExtIntInit

\* DESCRIPTION : Initialize specified as pull up for external interrupt

\* RETURN : enu\_buttonError\_t {BUTTON\_NOK, BUTTON\_OK}

\*/

enu\_buttonError\_t **HButton\_ExtIntInit**(enu\_pin en\_pinx);

/\*

\* AUTHOR : Bassel Yasser Mahmoud

\* FUNCTION : HButton\_getPinVal

\* DESCRIPTION : Get pin status if it is high or low

\* RETURN : enu\_buttonError\_t {BUTTON\_NOK, BUTTON\_OK}

\*/

enu\_buttonError\_t **HButton\_getPinVal**(enu\_pin en\_pinx, Uint8\_t\* pu8\_refVal );

\*

Description : This function initializes the external interrupt number and it's detecting type and initialize call back function.

ARGS : takes the EXINT\_NUMBER( INT0,INT1 OR INT2) and sense control and and pointer to the function we want to execute when interrupt occurs.

return : return EXTINT\_OK if the EXINT\_NUMBER initializes correctly, EXTINT\_NOT\_OK otherwise

\*/

EN\_EXTINT\_ERROR H\_EXTINT\_create(EN\_EXINT\_NUMBER INTx ,EN\_Sence\_Control INTxSense,void(\*ptrfunc)(void));

#### **HEXT\_INT**

\*

Description : This function initializes the external interrupt number and it's detecting type and initialize call back function.

ARGS : takes the EXINT\_NUMBER( INT0,INT1 OR INT2) and sense control and and pointer to the function we want to execute when interrupt occurs.

return : return EXTINT\_OK if the EXINT\_NUMBER initializes correctly, EXTINT\_NOT\_OK otherwise

\*/

EN\_EXTINT\_ERROR H\_EXTINT\_create(EN\_EXINT\_NUMBER INTx ,EN\_Sence\_Control INTxSense,void(\*ptrfunc)(void));

#### **HSPI**

/\*\*

\* (Author: Alaa Hisham)

\* \brief Initialize the MCU as the Master in SPI communication

\* \return void

\*/

void HSPI\_MasterInit(void);

/\*\*

\* (Author: Alaa Hisham)

\* \brief Initialize the MCU as a slave in SPI communication

\* \return void

\*/

void HSPI\_SlaveInit(void);

/\*\*

\* (Author: Alaa Hisham)

\* \brief Function to send a single character

\* \param u8\_a\_character character to send

\* \return void

\*/

void HSPI\_SendChar(Uchar8\_t u8\_a\_character);

/\*\*

\* \brief Receive data in the given buffer

\*

\* \param pu8\_a\_data: reference to buffer to store received data

\* \param u8\_a\_DataSize: Size (length) of data to receive

\*

\* \return void

\*/

void HSPI\_ReceiveData(Uchar8\_t \*pu8\_a\_data, Uchar8\_t u8\_a\_DataSize);

/\*\*

\* \brief Send given data byte by byte to selected slave

\* and receive data in exchange into given array

\* \param pu8\_a\_TxDataArr: Reference to array of data to be transmitted

\* \param pu8\_a\_RxDataArr: Reference to array to store received data

\* \param u8\_a\_DataLen: Length of data to exchange (in bytes)

\* (Must be less than or equal SPI\_BUFFER\_SIZE)

\*

\* \return en\_HSPI\_ErrorState\_t

\*/

en\_HSPI\_ErrorState\_t HSPI\_ExchangeData(Uchar8\_t\* pu8\_a\_TxDataArr, Uchar8\_t\* pu8\_a\_RxDataArr, Uchar8\_t u8\_a\_DataLen);

/\*\*

\* (Author: Alaa Hisham)

\* \brief Request to send data to the master

\* \param u8\_a\_data: data to send

\* \return void

\*/

void HSPI\_SlaveRequest(Uchar8\_t\* pu8\_a\_dataPtr, Uchar8\_t u8\_a\_DataSize);

#### **EEPROM:**

/\*\* AUTHOR : Sherif Ashraf Khadr

\* \brief : This Function Just Call To Initialize I2C as Master

\*

\* \param : Void

\*

\* \return void

\*/

void eeprom\_init(void)

/\*\* AUTHOR : Sherif Ashraf Khadr

\* \brief : This Function Call To Make The Sequence Of I2C Frame To Write Byte On A Device

\*

\* \param : Uint16\_t address : This Is Device Address

\* : Uchar8\_t data : This Is Data That Will Be Write

\*

\* \return void

\*/

void eeprom\_write\_byte(Uint16\_t address, Uchar8\_t data)

/\*\* AUTHOR : Sherif Ashraf Khadr

\* \brief : This Function Call To Make The Sequence Of I2C Frame To Read Byte On A Device

\*

\* \param : Uint16\_t address : This Is Device Address

\*

\* \return Uchar8\_t : Function Will Return Uchar8\_t Contain The Data

\*/

Uchar8\_t eeprom\_read\_byte(Uint16\_t address)

/\*\* AUTHOR : Sherif Ashraf Khadr

\* \brief : This Function Call When You Need To Write A String In The EEPROM

\*

\* \param : Uint16\_t address : This Is Device Address

\* : Uchar8\_t \*str : This Pointer Will Store The Address Of The Array Of The Chars

\* \return Void

\*/

void eeprom\_write\_string(Uint16\_t Copy\_u8Address, const Uchar8\_t\* str)

/\*\* AUTHOR : Sherif Ashraf Khadr

\* \brief : This Function Call When You Need To Read A String From The EEPROM

\*

\* \param : Uint16\_t address : This Is Device Address

\* : Uchar8\_t \*str : This Pointer Will Store The Address Of The Array Of The Chars That Will Return String In It

\* \return Void

\*/

void eeprom\_read\_string(Uint16\_t Copy\_u8Address, Uchar8\_t\* str)

### **Card\_Database Layer (CARD MCU)**

/\*

\* AUTHOR : Bassel Yasser Mahmoud

\* FUNCTION : APP\_terminalPinGet

\* DESCRIPTION : Get pin from User within terminal and doing some validation

\* RETURN : en\_terminalPinGetStatus\_t {PINGET\_NOK or PINGET\_OK}

\*/

en\_terminalPinGetStatus\_t **APP\_terminalPinGet**(Uchar8\_t\* arr);

/\*

\* AUTHOR : Sharpel

\* FUNCTION : APP\_terminalPanGet

\* DESCRIPTION : Get pan from User within terminal and doing some validation

\* RETURN : en\_terminalPanGetStatus\_t {PANGET\_NOK or PANGET\_OK}

\*/

en\_terminalPanGetStatus\_t **APP\_terminalPanGet**(Uchar8\_t\* arr);

/\*

\* FUNCTION : SaveCardData

\* DESCRIPTION : Saving PAN and PIN in EEPROM

\* RETURN : EN\_TerminalDataState {DATA\_SAVED, DATA\_NSAVED, DATA\_READ, DATA\_NREAD}

\*/

EN\_TerminalDataState **SaveCardData**(Uchar8\_t \*CardPan,Uchar8\_t \*CardPin);

/\*

\* FUNCTION : ReadCardData

\* DESCRIPTION : Reading PAN and PIN from EEPROM

\* RETURN : EN\_TerminalDataState {DATA\_SAVED, DATA\_NSAVED, DATA\_READ, DATA\_NREAD}

\*/

EN\_TerminalDataState **ReadCardData**(Uchar8\_t \*CardPan,Uchar8\_t \*CardPin);

/\*

\* FUNCTION : CARD\_MatchPINs

\* DESCRIPTION : Validate if PIN no and Confirmed PIN no is Matched or not

\* RETURN : en\_CardPinMatchError\_t { PIN\_Match\_NOK, PIN\_Match\_OK}

\*/

en\_CardPinMatchError\_t **CARD\_MatchPINs**();

### **ATM\_MODULE Layer (ATM MCU)**

#### **ATM:**

/\*\*

\* \brief Displays welcome routine

\*

\* \return void

\*/

void Welcome(void);

/\*\*

\* (Author: Sherif Ashraf)

\* \brief Check if the pin user enters is the same as the cardholder's pin

\* \param pinFromAtm : reference to pin entered by user

\* \param pinFromServer: reference to pin received from card

\* \return EN\_PinState

\*/

EN\_PinState PIN\_checkPinMatching(Uchar8\_t \*pinFromAtm,Uchar8\_t \*pinFromServer);

/\*

\* AUTHOR : Sharpel

\* FUNCTION : Get\_pin

\* DESCRIPTION : get pin from user (on the atm )

\* ARGS : pointer to array (the size of array must be 5 or more) to store entered pin by user

\* RETURN : PIN\_OK if user enters 4 numbers , PIN\_NOT\_OK otherwise

\*/

EN\_PinState Get\_pin(Uchar8\_t \*enteredpin);

/\*\*

\* (Author: Sherif Ashraf)

\* \brief Locks the system and sound the buzzer

\* \param pst\_a\_buzzer: reference to the buzzer

\* \return en\_BuzzerErrorState\_t

\*/

en\_BuzzerErrorState\_t deinitAtm(st\_Buzzer\_t\* pst\_a\_buzzer);

/\*\*

\* (Author: Alaa Hisham)

\* \brief Get the card pan and pin

\* \param pu8\_a\_pan: reference to buffer to receive pan from card

\* \param pu8\_a\_pin: reference to buffer to receive pin from card

\* \return EN\_PinState

\*/

EN\_PinState ATM\_GetCardData(Uchar8\_t \*pu8\_a\_pan, Uchar8\_t \*pu8\_a\_pin);

/\*

\* AUTHOR : Sharpel

\* FUNCTION : get\_amount\_left

\* DESCRIPTION : get amount from user ( on the atm )

\* ARGS : pointer to array (the size of array must be 8 or more and equal "0000.00" initial value) to store entered pin by user

\* RETURN : void

\*/

void get\_amount\_left (Uchar8\_t \* amount);

/\*

\* AUTHOR : Bassel Yasser

\* FUNCTION : EXTINT\_FUNC

\* DESCRIPTION : when timer 2 ISR is fire it changes the state of (Enter or Zero)

\* ARGS : void

\* RETURN : void

\*/

void EXTINT\_FUNC(void);

/\*\*

\* (Author: Alaa Hisham)

\* \brief Carries out the routine for approved card

\* \param f32\_a\_NewBalance: the balance to display after transaction

\* \return void

\*/

void ATM\_ApprovedCard(float32\_t f32\_a\_NewBalance);

#### **Database\_check:**

/\*

\* FUNCTION : isValidPanAccount

\* DESCRIPTION : Check If PAN No. Valid or not

\* RETURN : EN\_dataError\_t {APPROVED ,FRAUD\_CARD,CARD\_STOLEN,EXCEED\_MAX\_DAILY\_AMOUNT,INSUFFICIENT\_FUND, DATA\_ERROR}

\*/

EN\_dataError\_t **isValidPanAccount**(Uchar8\_t \* pan);

/\*

\* FUNCTION : isRunningAccount

\* DESCRIPTION : Checking if card stolen or not

\* RETURN : EN\_dataError\_t {APPROVED ,FRAUD\_CARD,CARD\_STOLEN,EXCEED\_MAX\_DAILY\_AMOUNT,INSUFFICIENT\_FUND, DATA\_ERROR}

\*/

EN\_dataError\_t **isRunningAccount**(Uchar8\_t \* pan);

/\*

\* FUNCTION : isValidAccountAmount

\* DESCRIPTION : Checking if there is INSUFFICIENT\_FUND or not

\* RETURN : EN\_dataError\_t {APPROVED ,FRAUD\_CARD,CARD\_STOLEN,EXCEED\_MAX\_DAILY\_AMOUNT,INSUFFICIENT\_FUND, DATA\_ERROR}

\*/

EN\_dataError\_t **isValidAccountAmount**(Uchar8\_t \* pan,Uchar8\_t \* amount,float32\_t \*newAmount);

/\*

\* FUNCTION : isBelowMaxDailyAmount

\* DESCRIPTION : Checking if transfered money is below limited daily amount or not

\* RETURN : EN\_dataError\_t {APPROVED ,FRAUD\_CARD,CARD\_STOLEN,EXCEED\_MAX\_DAILY\_AMOUNT,INSUFFICIENT\_FUND, DATA\_ERROR}

\*/

EN\_dataError\_t **isBelowMaxDailyAmount**(Uchar8\_t \* amount);

/\*

\* FUNCTION : DATABASE\_checking

\* DESCRIPTION : Card Database checking

\* RETURN : EN\_dataError\_t {APPROVED ,FRAUD\_CARD,CARD\_STOLEN,EXCEED\_MAX\_DAILY\_AMOUNT,INSUFFICIENT\_FUND, DATA\_ERROR}

\*/

EN\_dataError\_t **DATABASE\_checking** (Uchar8\_t \* pan,Uchar8\_t \* amount,float32\_t \*newAmount);

### **Application Layer:**

#### **Card MCU**

/\*\*

\* \brief Initializations of all used peripherals

\* \return void

\*/

void APP\_Init(void);

/\*\*

\* \brief The main logic of the Card

\* \return void

\*/

void APP\_Start(void);

#### **ATM MCU**

/\*\*

\* \brief Initializations of all used peripherals

\* \return void

\*/

void APP\_Init(void);

/\*\*

\* \brief The main logic of the ATM

\* \return void

\*/

void APP\_Start(void);

# Low Level Design

### **MCAL Layer:**

#### **DIO**

Sint8\_t **DIO\_s8SETPinDir** (enu\_pin enPinCopy, enu\_dir enPortDir)

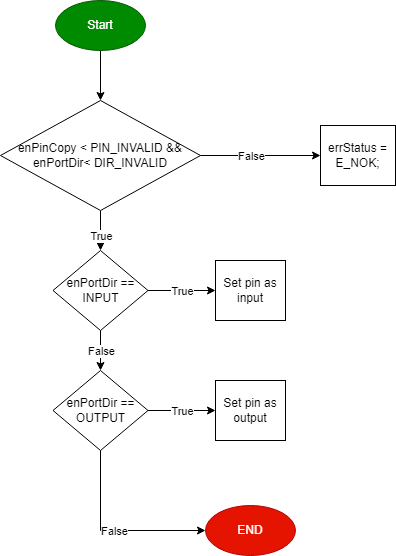
****

Figure DIO\_s8SETPinDir Flow Chart

Sint8\_t **DIO\_s8SETPinVal** (enu\_pin enPinCopy, enu\_val enPortVal)

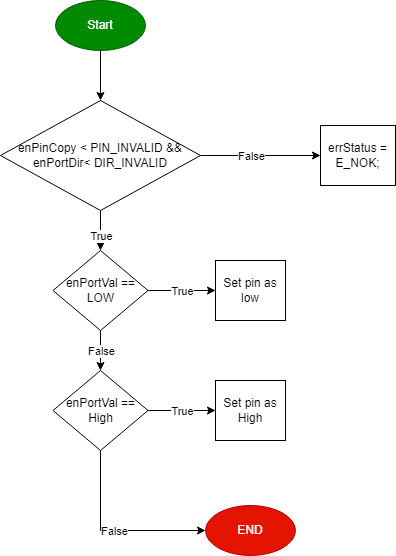
****

Figure DIO\_s8SETPinVal Flow chart

Sint8\_t **DIO\_s8GETPinVal** (enu\_pin enPinCopy, Uint8\_t\* pu8Val)

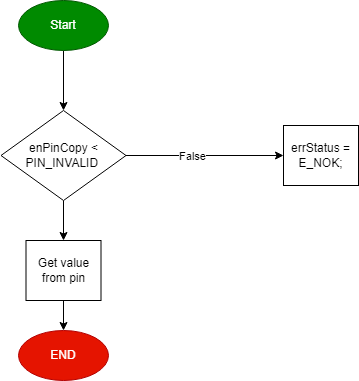
****

Figure DIO\_s8GETPinVal Flow Chart

#### **Timer:**

**TIM0\_Init**

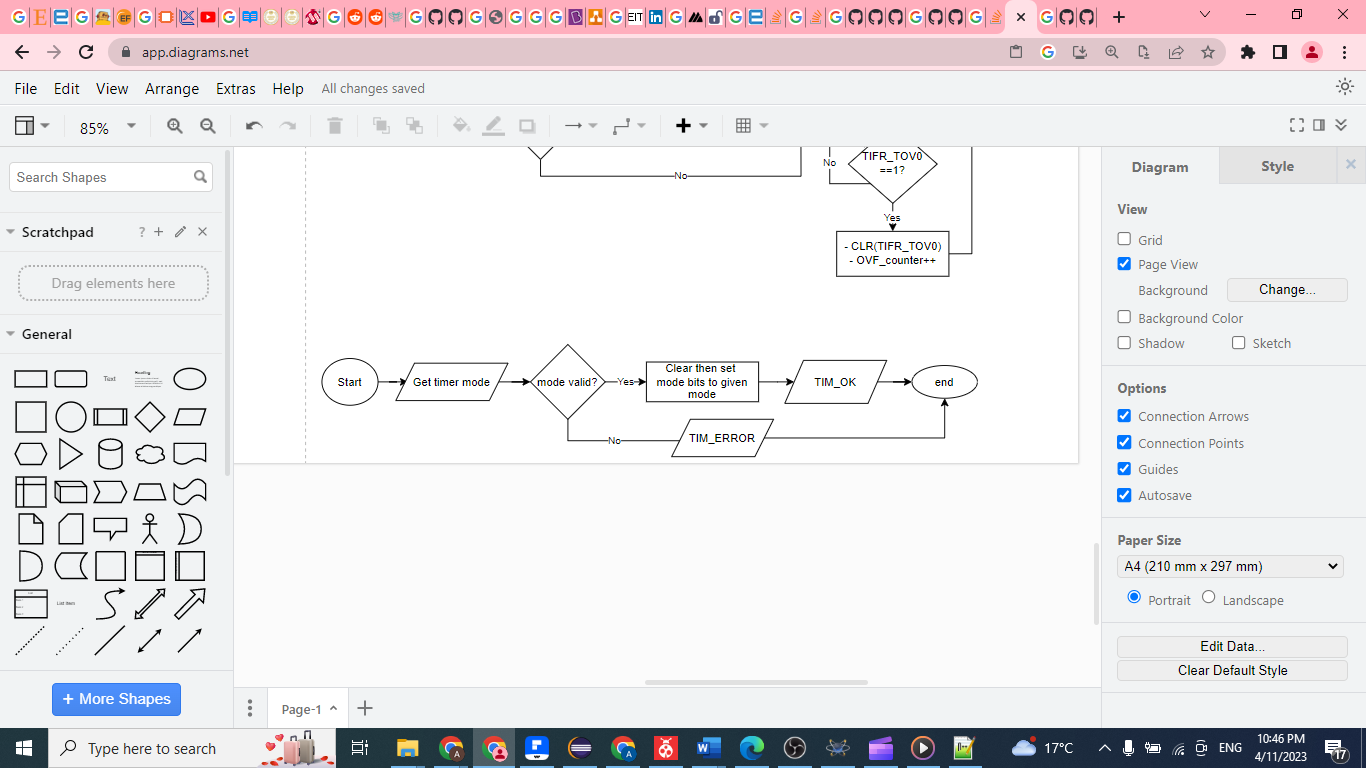


Figure TIM0\_Init Flow Chart

**TIM0\_Start**

****

Figure TIM0\_Start Flow Chart

**TIM0\_Stop**

****

Figure TIM0\_Stop Flow Chart

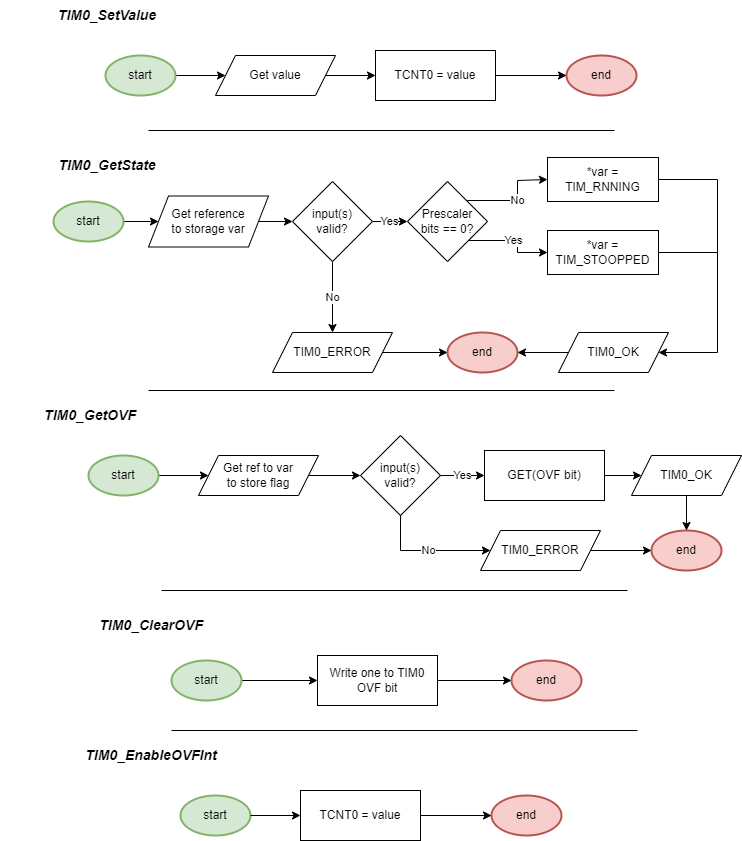


Figure TIM0 remaining Flow Charts

#### **EXTINT:**

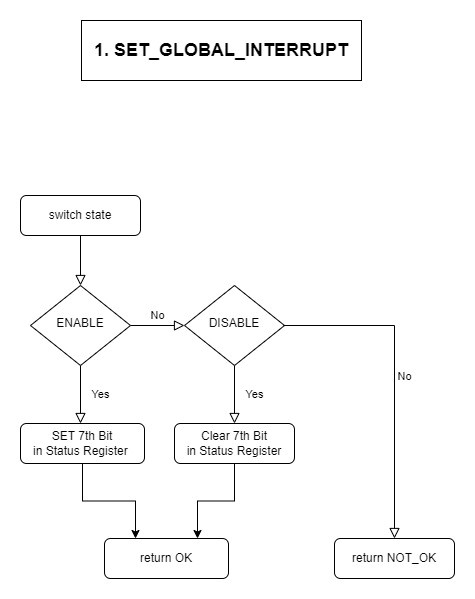
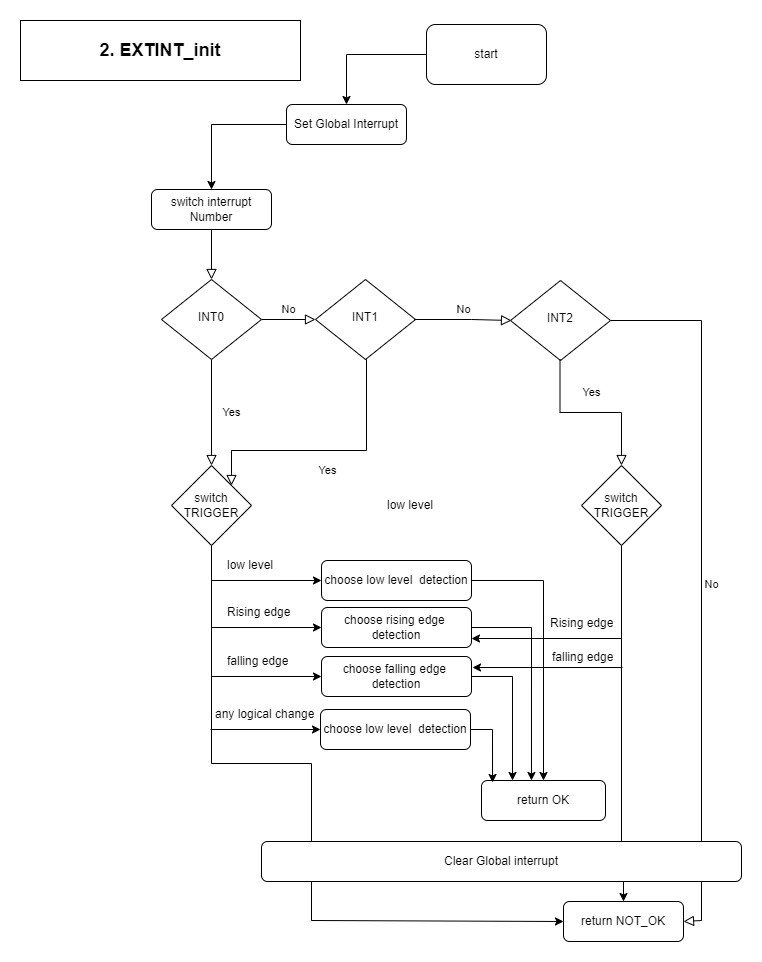


Figure EXTINT: Set\_Global\_Interrupt

**** Figure 11 EXTINT\_Init flow chart

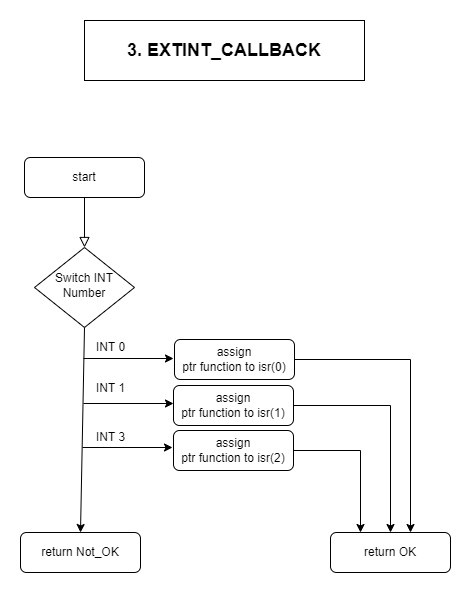
****

Figure EXTINT\_CALLBACK flow chart

#### **UART:**

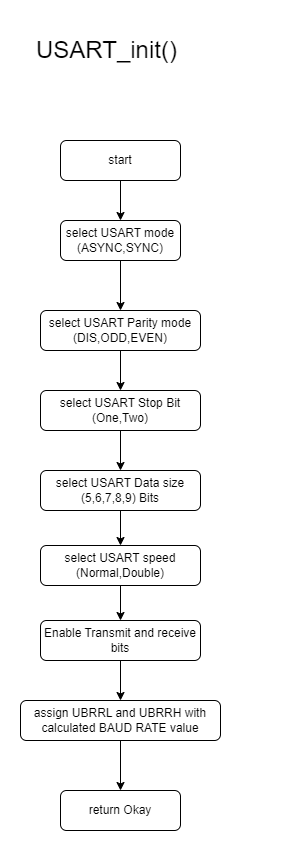


Figure USART\_init flow chart

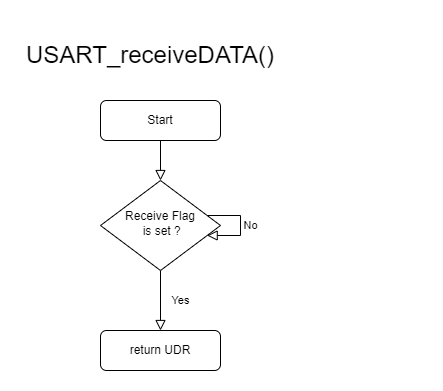


Figure USART\_receiveData flow chart

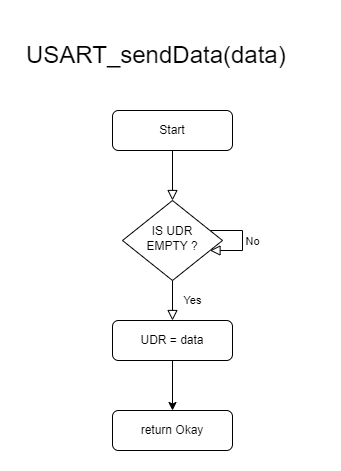


Figure USART\_sendData flow chart

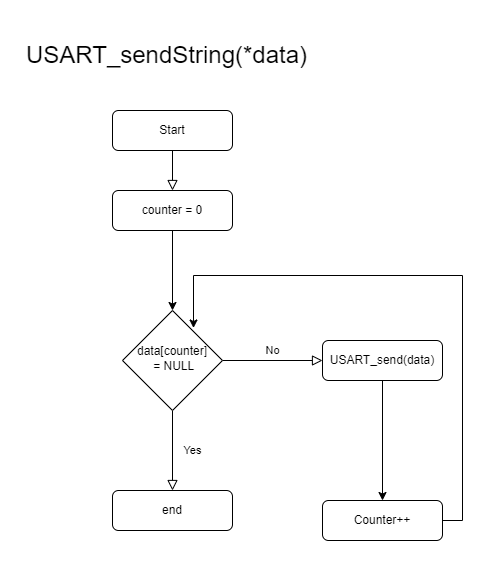


Figure USART\_sendString flow chart

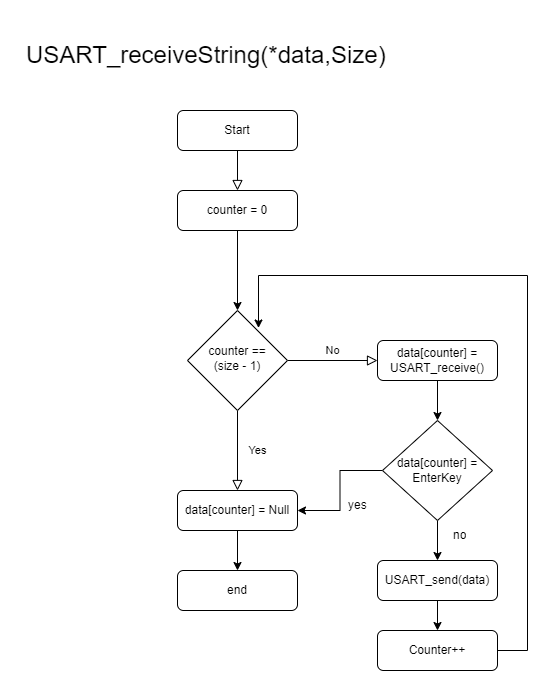


Figure USART\_receiveString flow chart

#### **SPI:**

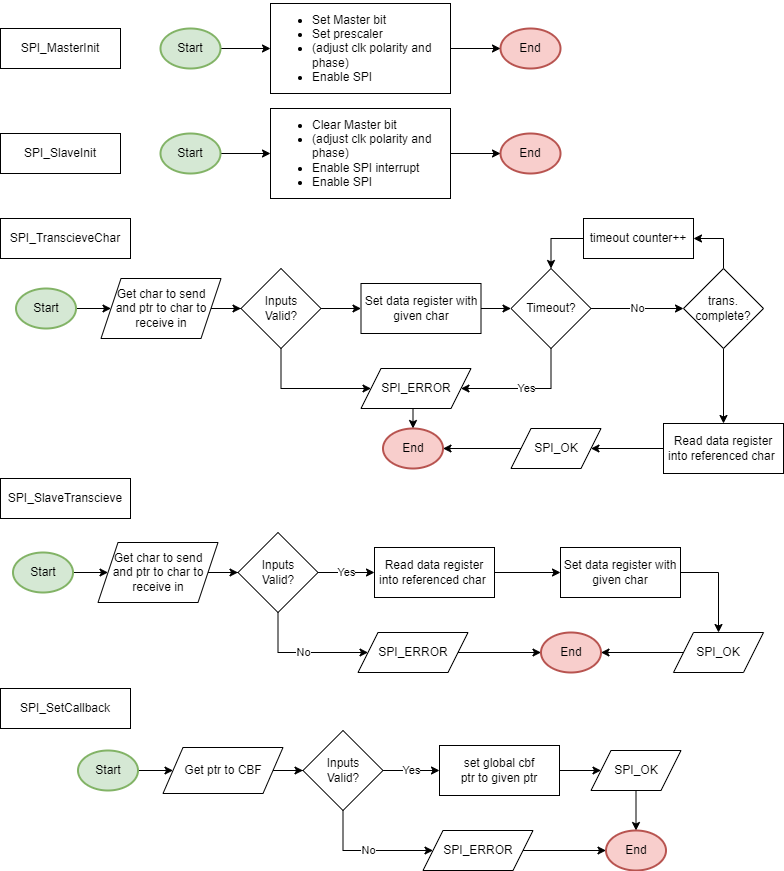


Figure Flow charts of SPI APIs

#### **I2C:**

**void** **i2c\_init\_master**(**void**);

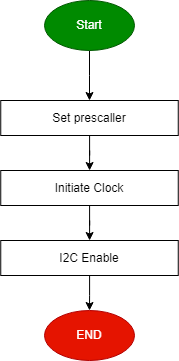


Figure i2c\_init\_master flow chart

**void** **i2c\_start**(**void**);

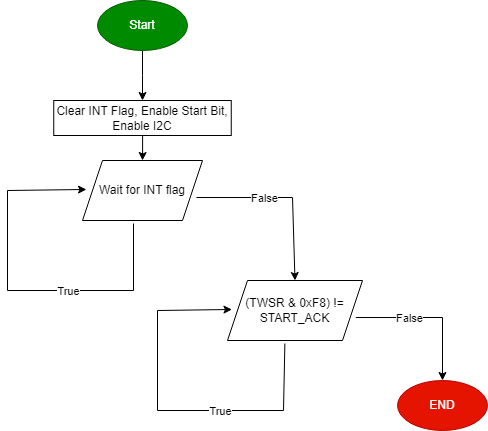


Figure i2c\_start flow chart

**void** **i2c\_repeated\_start**(**void**);

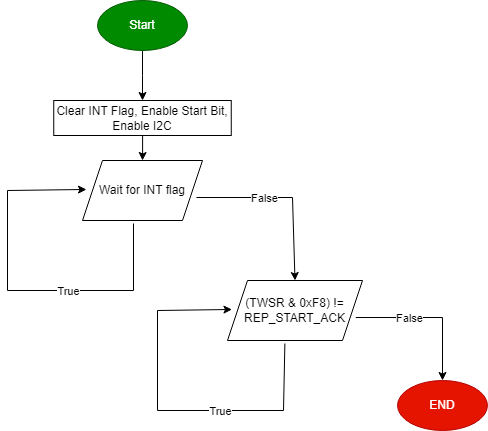


Figure i2c\_repeated\_start flow chart

**void** **i2c\_send\_slave\_address\_with\_write\_req**(Uint8\_t slave\_address);

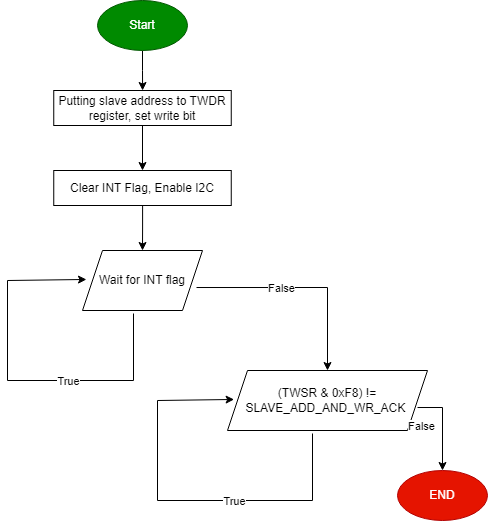


Figure i2c\_send\_slave\_address\_with\_write\_req flow chart

**void** **i2c\_send\_slave\_address\_with\_read\_req**(Uint8\_t slave\_address);

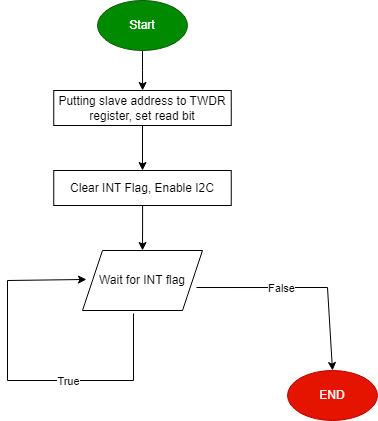


Figure i2c\_send\_slave\_address\_with\_read\_req flow chart

**void** **i2c\_write\_byte**(Uint8\_t byte);

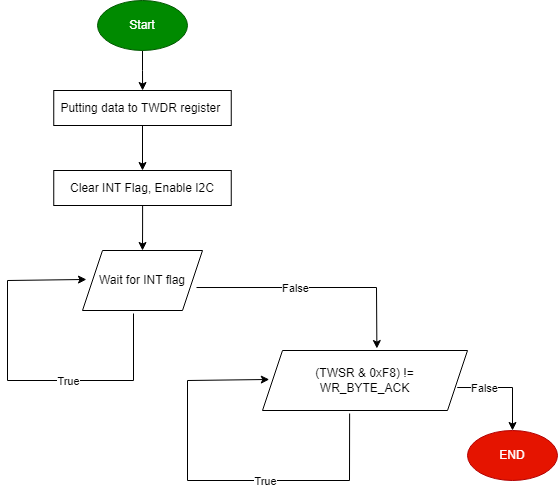


Figure i2c\_write\_byte flow chart

Uint8\_t **i2c\_read\_byte**(**void**);

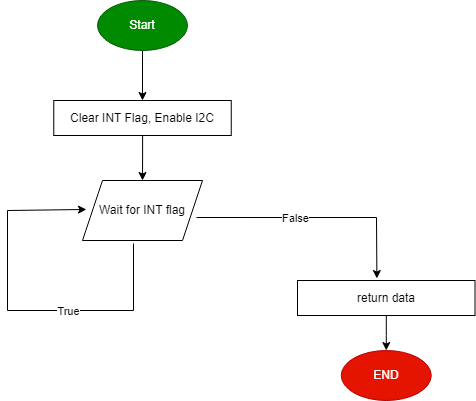


Figure i2c\_read\_byte flow chart

**void** **i2c\_stop**(**void**);

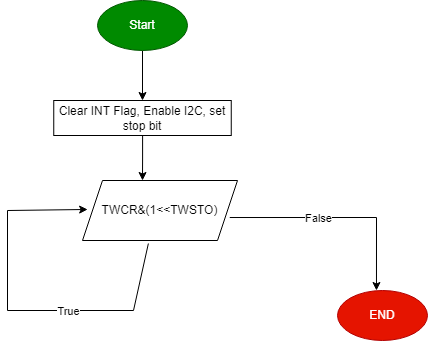
****

Figure i2c\_stop flow chart

### **HAL Layer**

#### **HTimer0**

**HTIM0\_SyncDelay**

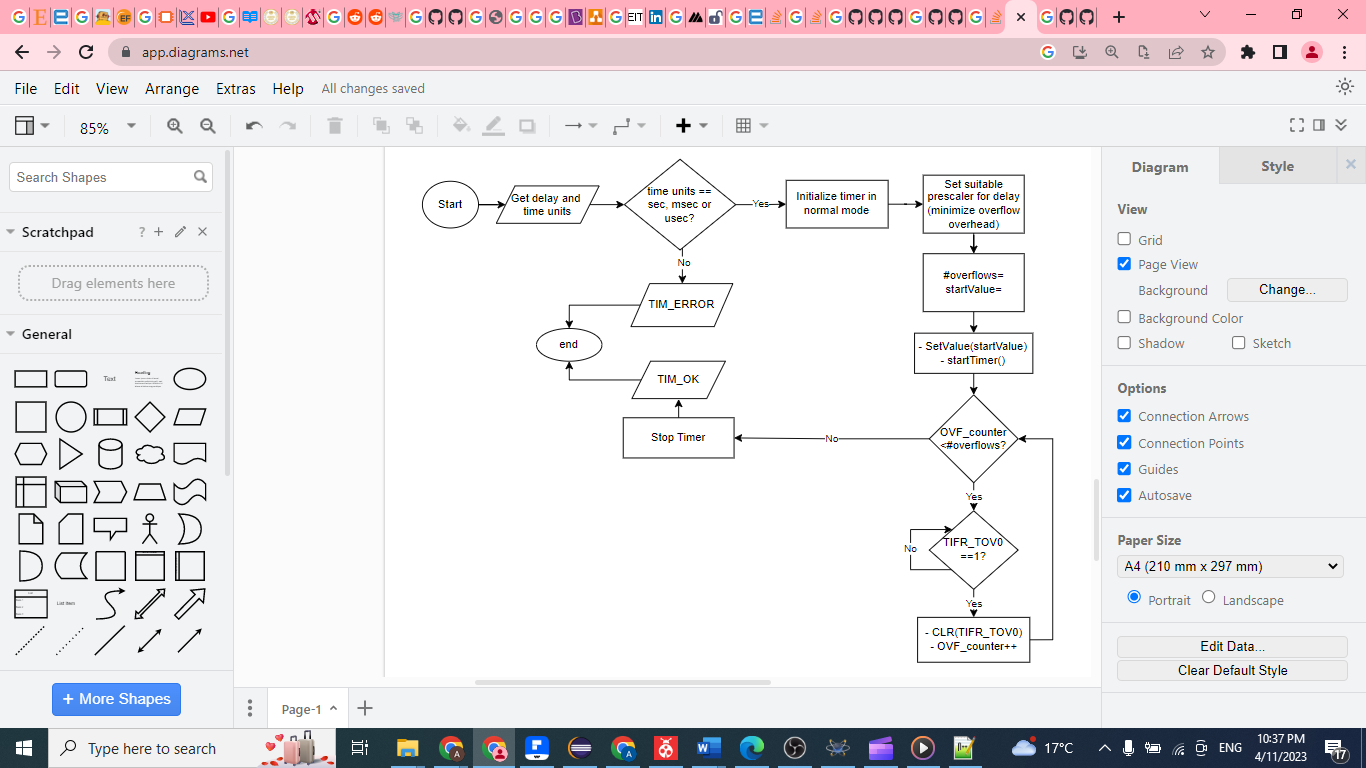


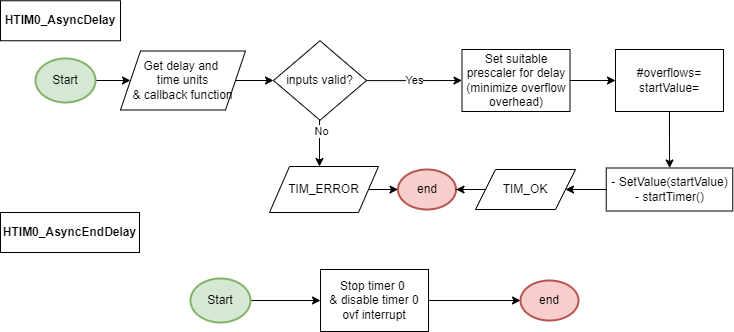
Figure HTIM0\_SyncDelay Flow Chart

Figure HTIM0\_AsyncDelay and EndDelay

#### **HSPI**

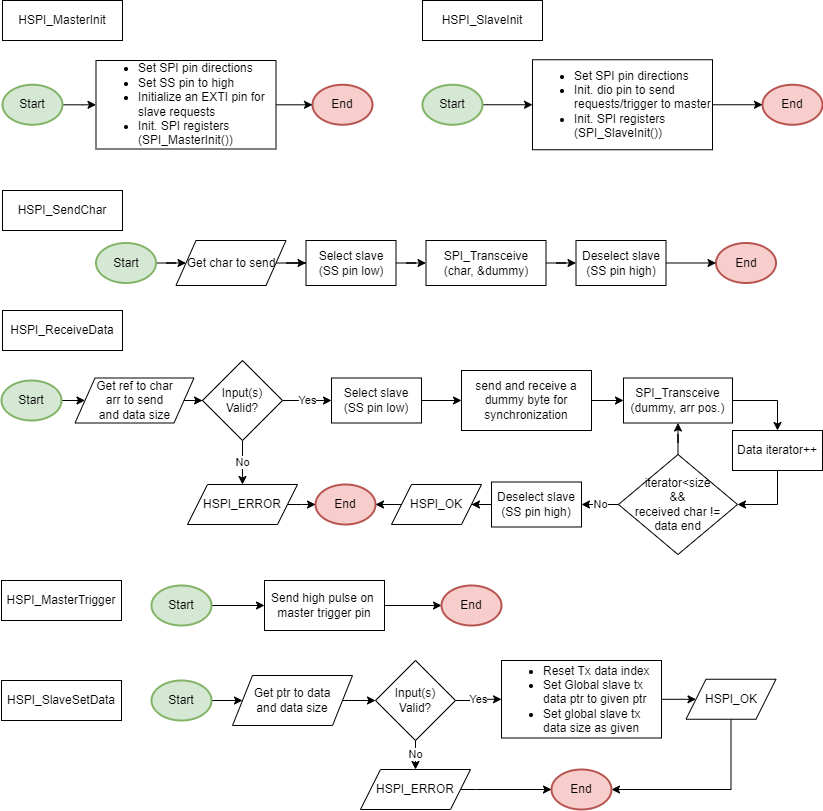


Figure Flow charts of HSPI APIs

#### **LCD**

**void** **HLCD\_vidInit**(**void**)

Figure HLCD\_vidInit Flow Chart

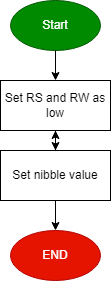
**void** **HLCD\_vidWritecmd**(Uint8\_t u8commandCopy)

Figure HLCD\_vidWritecmd Flow Chart

**void** **HLCD\_vidWriteChar**(Uint8\_t u8CharCopy)

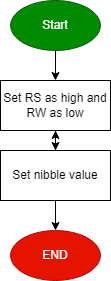
****

Figure HLCD\_vidWriteChar Flow Chart

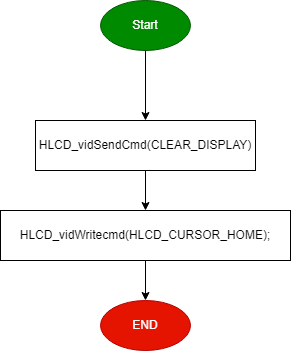
**void** **HLCD\_ClrDisplay**(**void**)

Figure HLCD\_ClrDisplay Flow Chart

**void** **HLCD\_gotoXY**(Uint8\_t row, Uint8\_t pos)

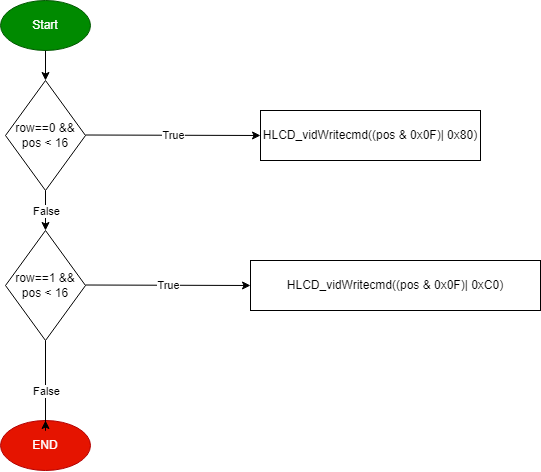
****

Figure HLCD\_gotoXY Flow Chart

**void** **HLCD\_WriteString**(Uint8\_t\* str)

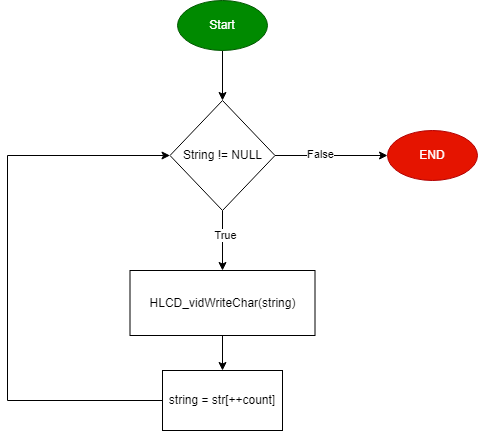
****

Figure HLCD\_WriteString Flow Chart

**void** **HLCD\_WriteInt**(Uint32\_t number)

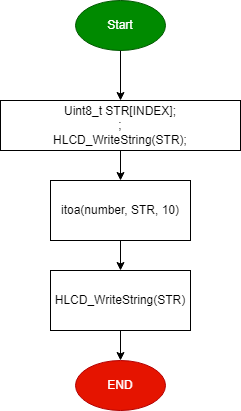
****

Figure HLCD\_WriteInt Flow Chart

**void** **HLCD\_vidCreatCustomChar**(Uint8\_t\* pu8custom, Uint8\_t u8Location)

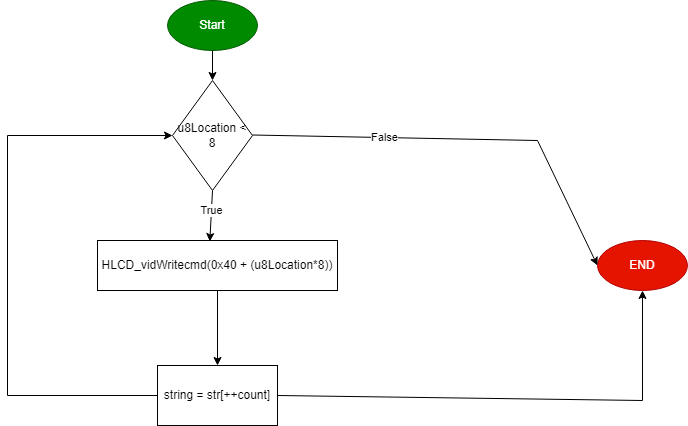
****

Figure HLCD\_vidCreatCustomChar Flow Chart

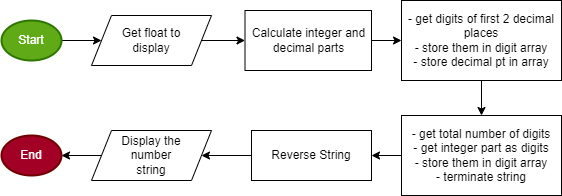
void HLCD\_DisplayFloat(float32\_t f32\_a\_number);

Figure HLCD\_DisplayFloat flow chart

#### **Keypad**

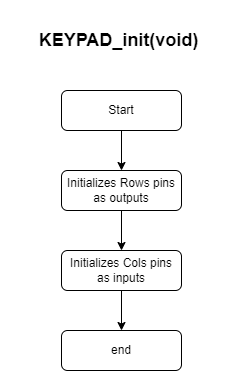


Figure KEYPAD\_Init Flow Chart

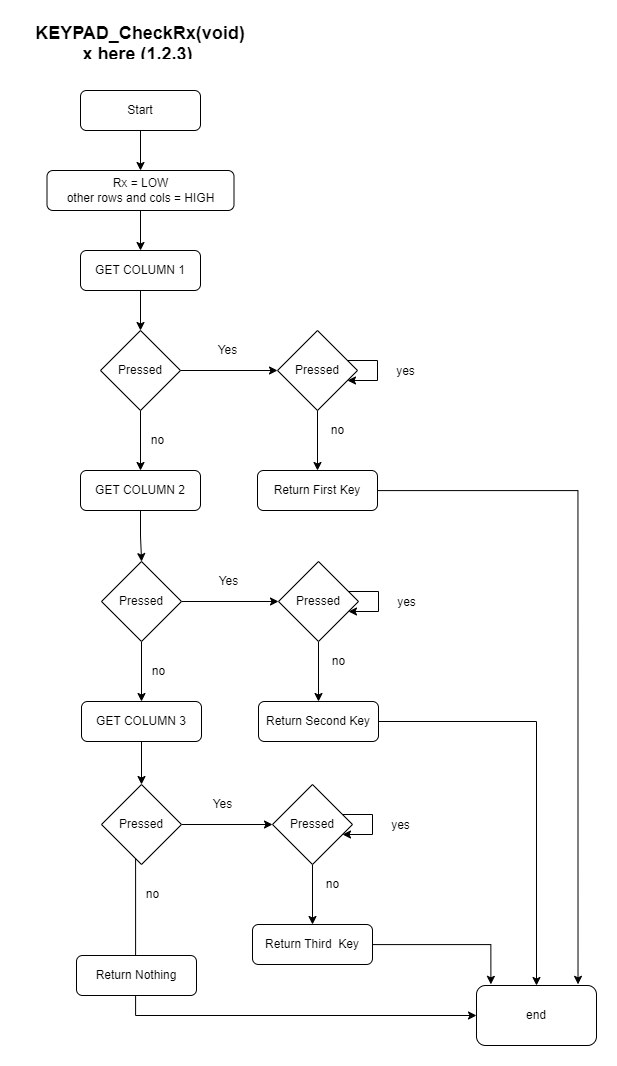


Figure KEYPAD\_CheckRx Flow Chart

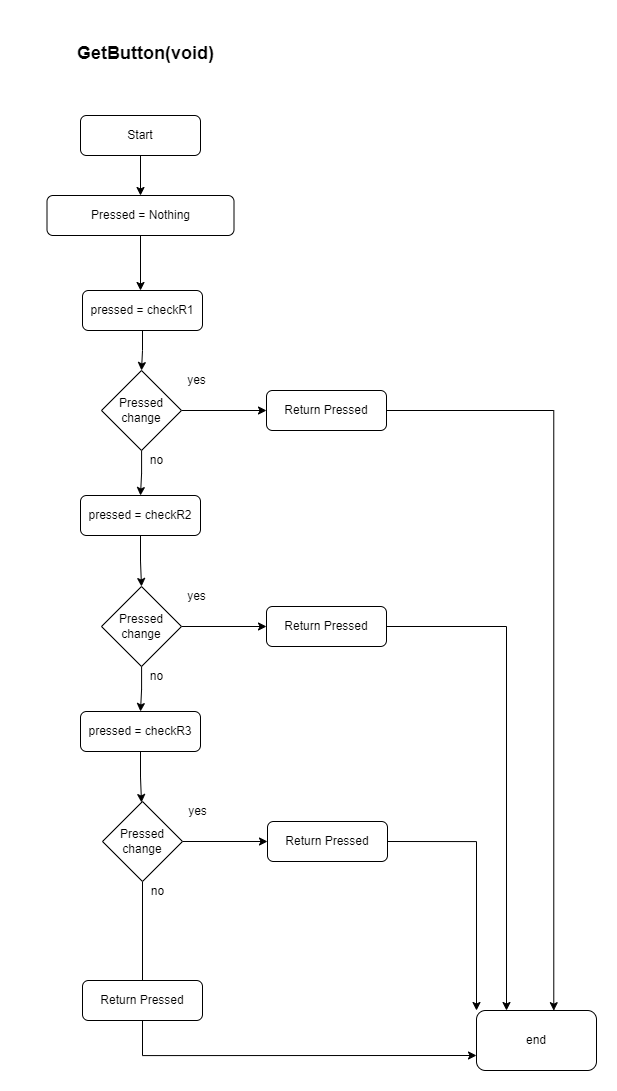
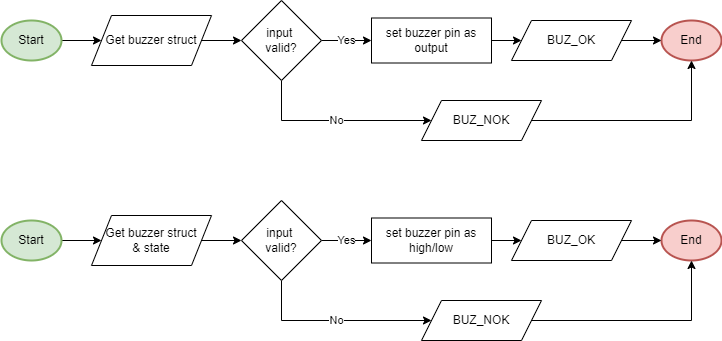


Figure GetButton Flow Chart

#### **Buzzer**

****

**BUZ\_SetState**

Figure Buzzer Init & SetState Flow Charts

**BUZ\_Init**

#### **HEXTINT:**

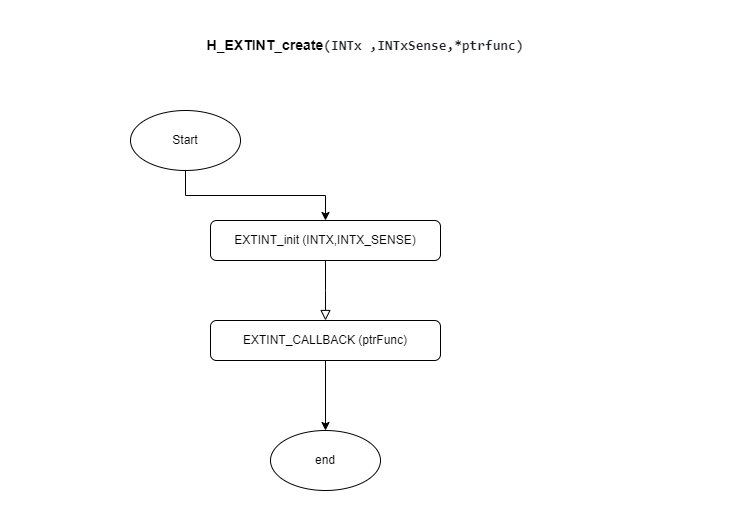
****

Figure H\_EXTINT\_create flow chart

#### **Button:**

enu\_buttonError\_t **HButton\_Init**(enu\_pin); enu\_buttonError\_t **HButton\_ExtIntInit**(enu\_pin);

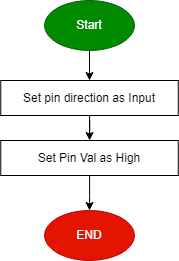
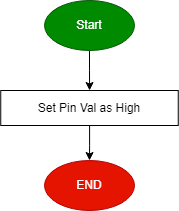
 Figure 44 HButton\_Init flow chart

Figure 5 HButton\_ExtIntInit flow chart

enu\_buttonError\_t **HButton\_getPinVal**(enu\_pin en\_pinx, Uint8\_t\* pu8\_refVal );

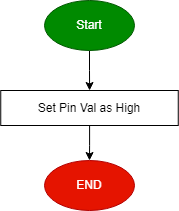
****

Figure HButton\_getPinVal flow chart

#### **EEPROM**

void eeprom\_init(void)

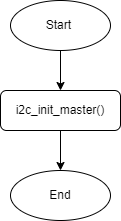


Figure eeprom\_init flow chart

void eeprom\_write\_byte(Uint16\_t address, Uchar8\_t data)

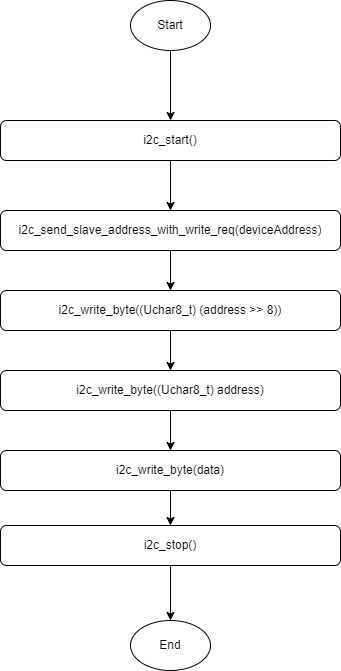


Figure eeprom\_write\_byte flow chart

Uchar8\_t eeprom\_read\_byte(Uint16\_t address)

A picture containing diagram

Description automatically generated

Figure eeprom\_read\_byte flow chart

void eeprom\_write\_string(Uint16\_t Copy\_u8Address, const Uchar8\_t\* str)

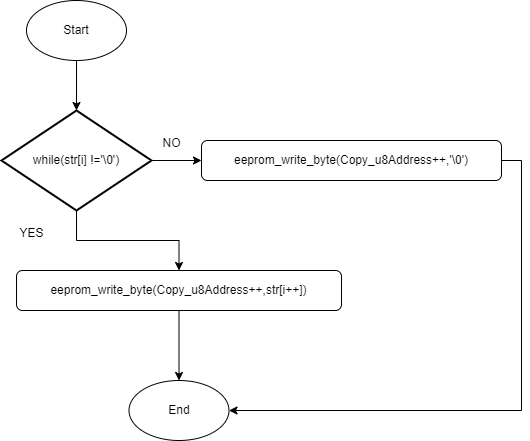


Figure eeprom\_write\_string flow chart

void eeprom\_read\_string(Uint16\_t Copy\_u8Address, Uchar8\_t\* str)

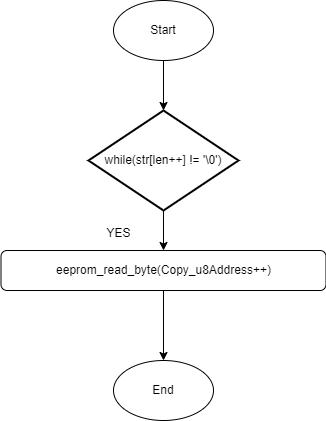


Figure eeprom\_read\_string flow chart

### **Card\_Database Layer (CARD MCU)**

en\_terminalPinGetStatus\_t **APP\_terminalPinGet**(Uchar8\_t\* arr);

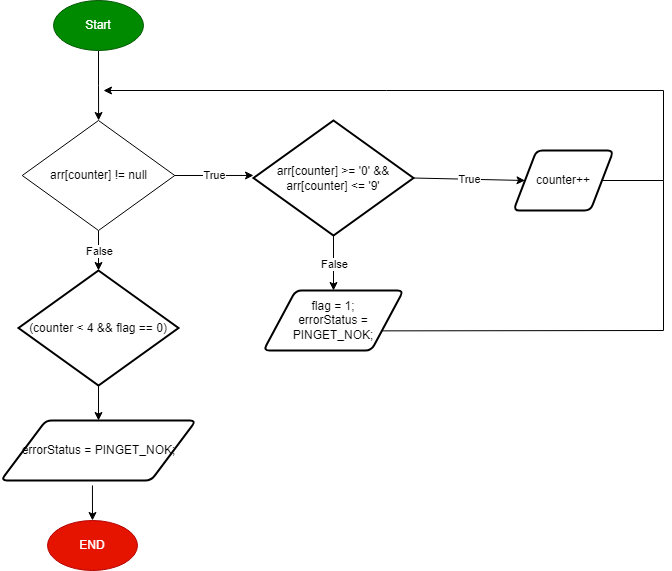


Figure APP\_terminalPinGet flow chart

en\_terminalPanGetStatus\_t **APP\_terminalPanGet**(Uchar8\_t\* arr);

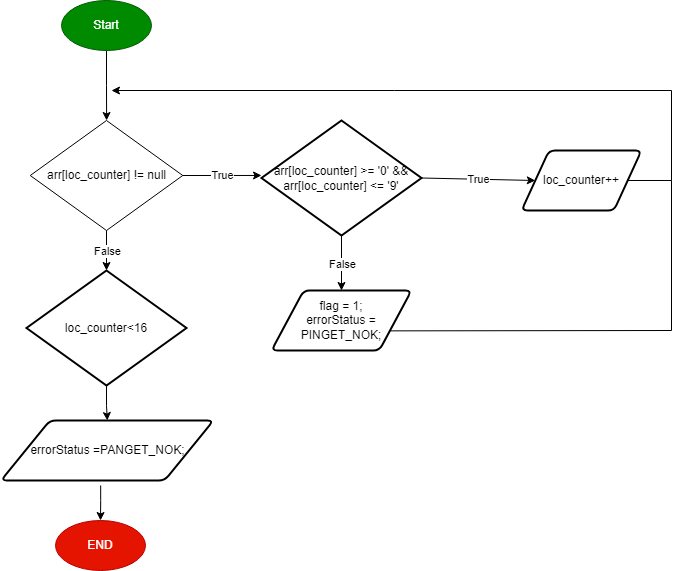
****

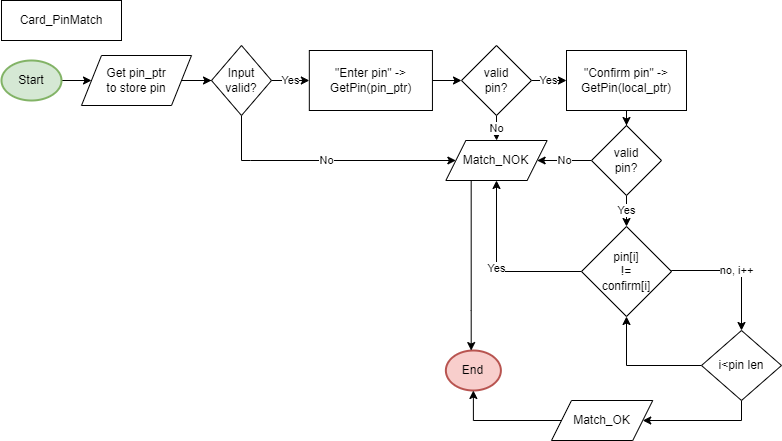
Figure APP\_terminalPanGet flow chart

Figure CARD\_PinMatch flow chart

SaveCardData

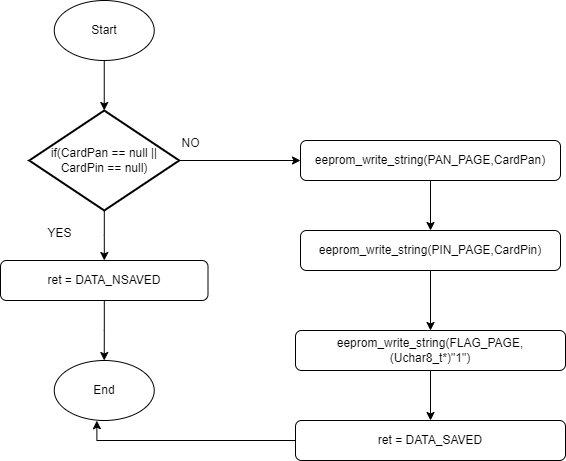


Figure SaveCardData flow chart

ReadCardData

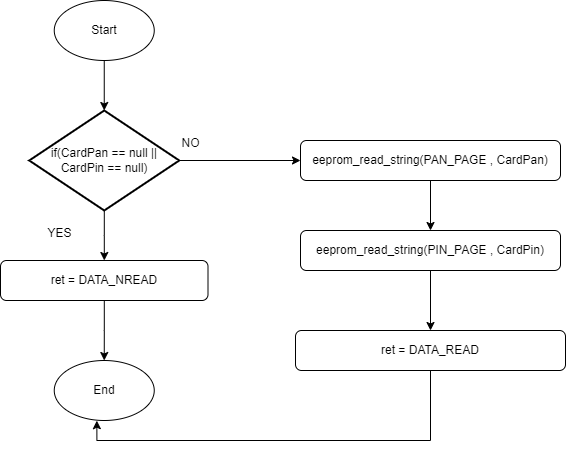


Figure ReadCardData flow chart

#### **Database\_check:**

EN\_dataError\_t **isValidPanAccount**(Uchar8\_t \* pan);

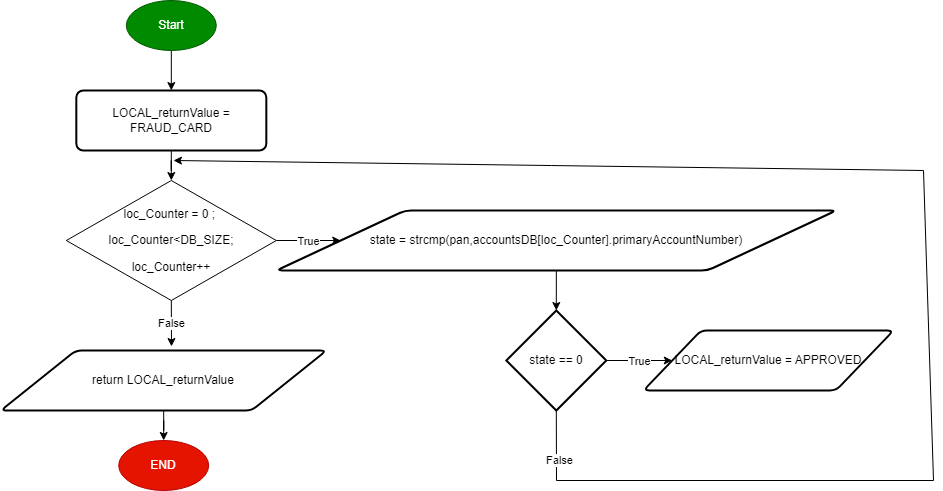


Figure isValidPanAccount flow chart

EN\_dataError\_t **isRunningAccount**(Uchar8\_t \* pan);

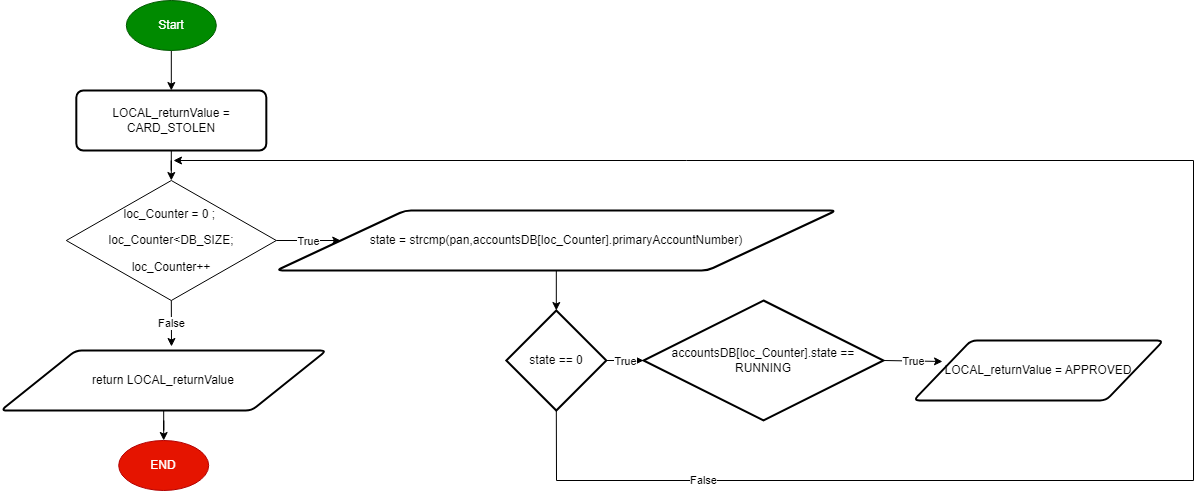


Figure isRunningAccount flow chart

EN\_dataError\_t **isValidAccountAmount**(Uchar8\_t \*pan,Uchar8\_t \*amount,float32\_t \*newAmount);

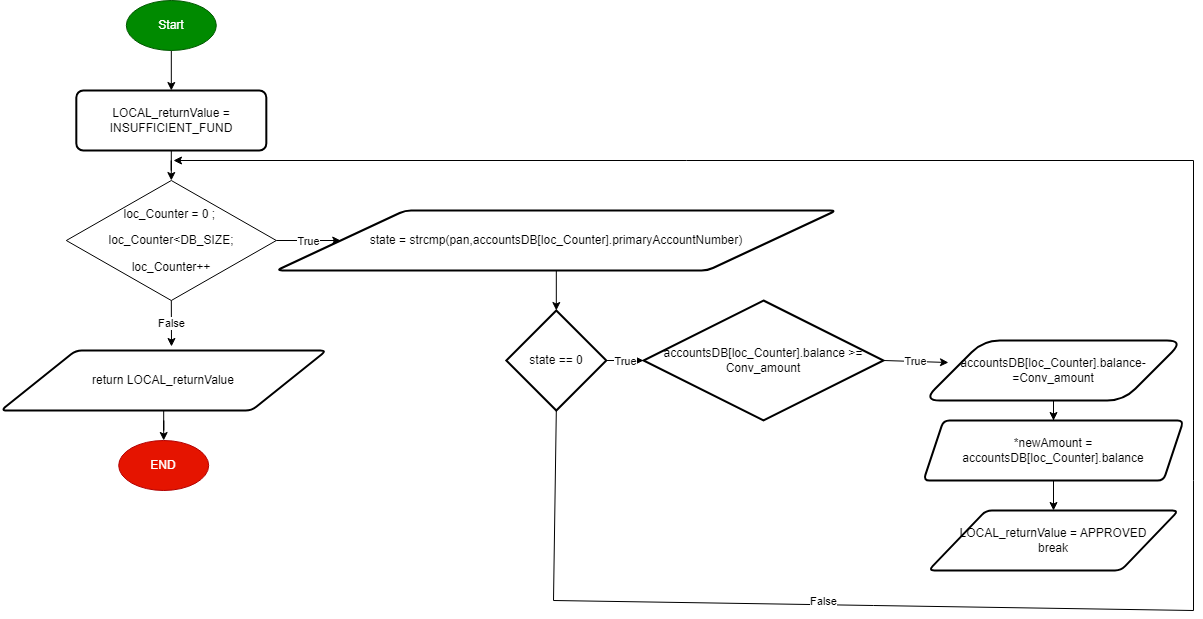


Figure isValidAccountAmount flow chart

EN\_dataError\_t **isBelowMaxDailyAmount**(Uchar8\_t \* amount);

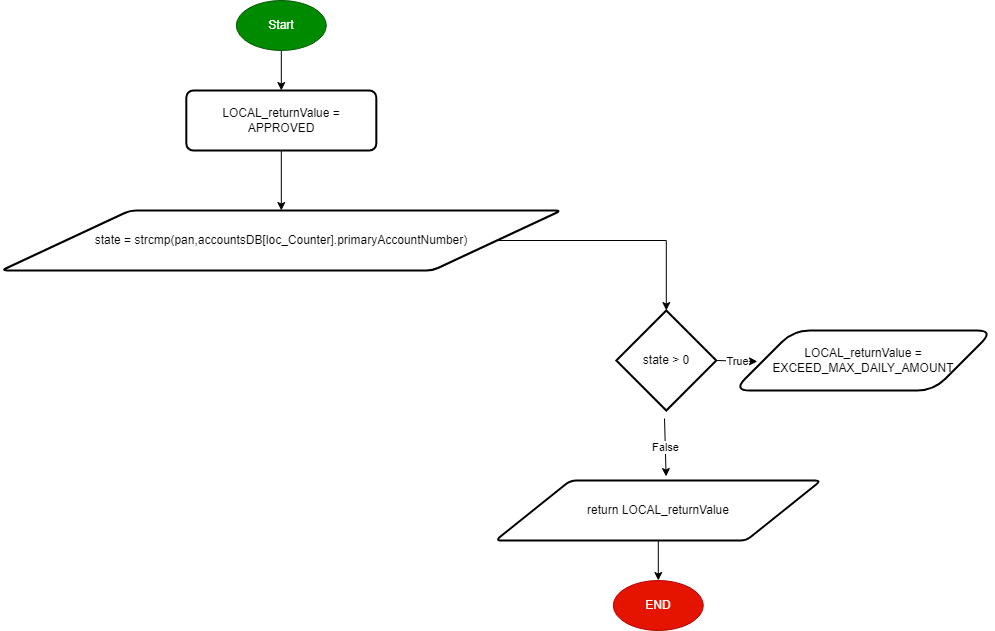


Figure isBelowMaxDailyAmount flow chart

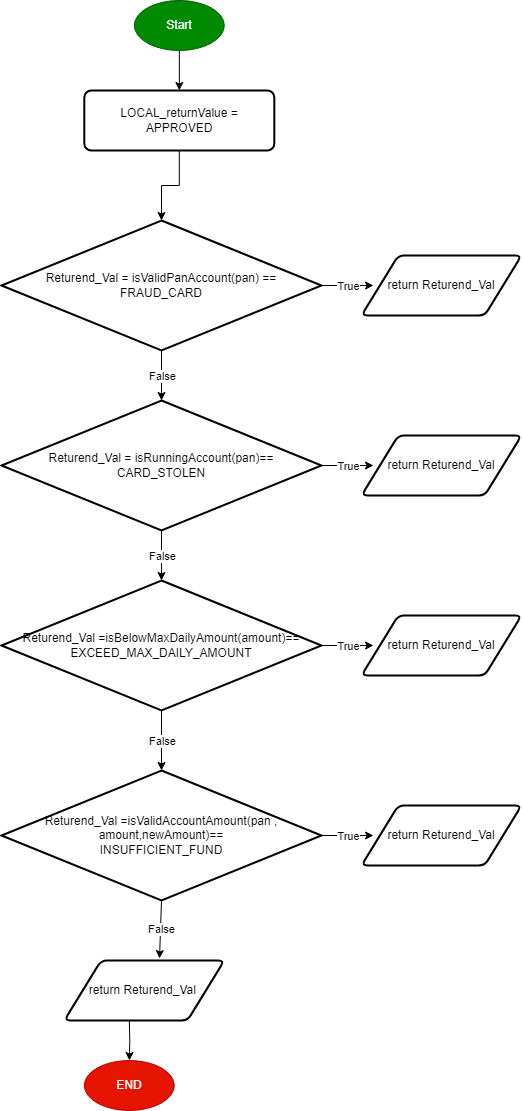
EN\_dataError\_t **DATABASE\_checking** (Uchar8\_t \* pan,Uchar8\_t \* amount,float32\_t \*newAmount);

Figure DATABASE\_checking flow chart

#### **ATM Module**

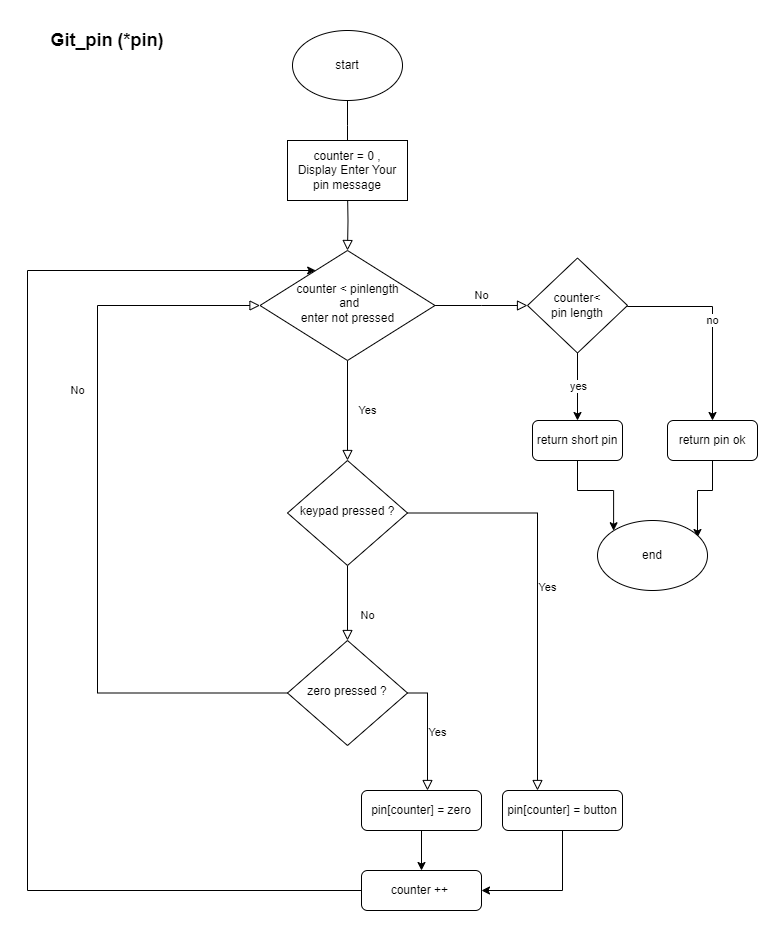


Figure Get\_pin flow chart

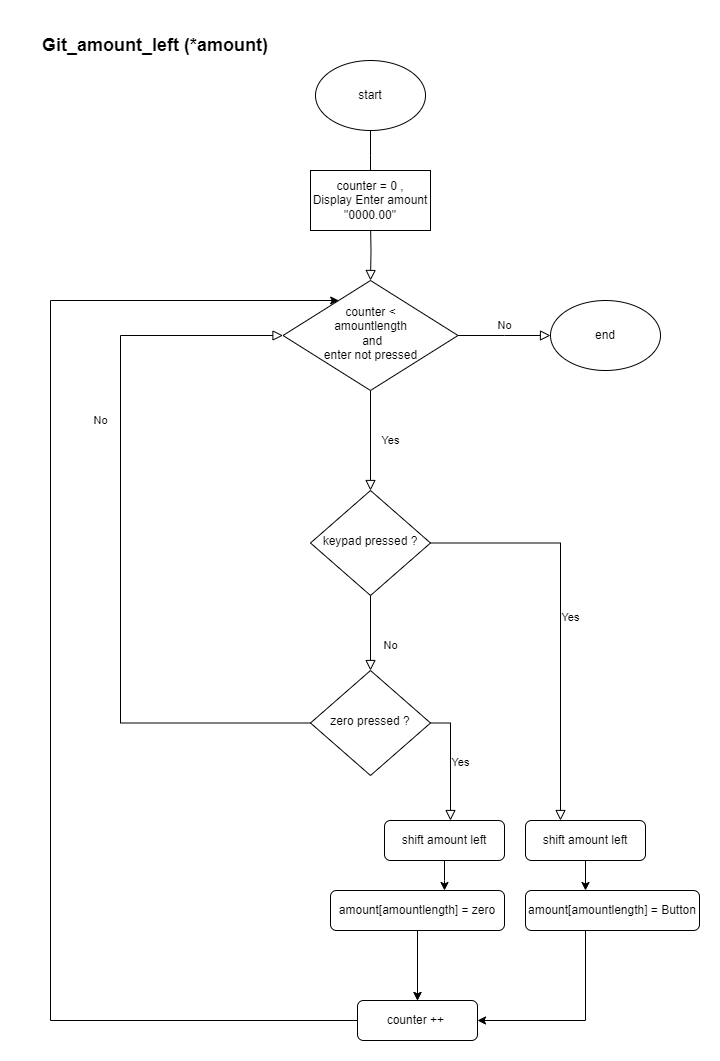


Figure 63 get\_amount\_left flow chart

PIN\_checkPinMatching

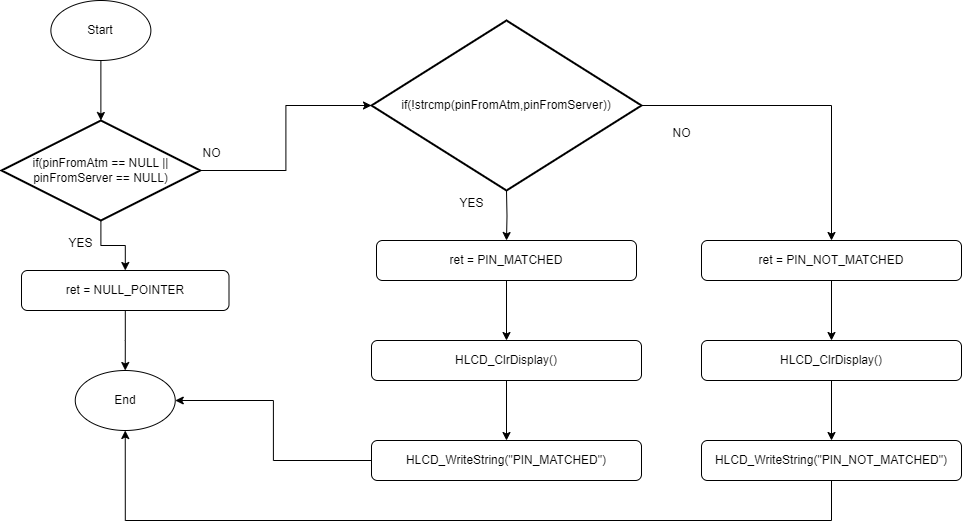


Figure PIN\_checkPinMatching flow chart

deinitAtm

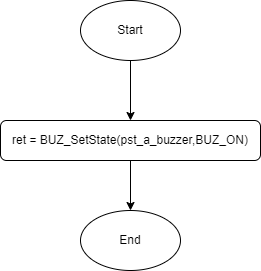


Figure deinitAtm flow chart

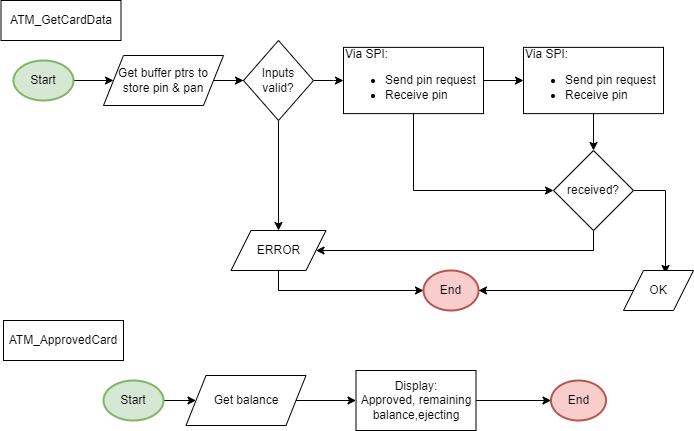


Figure ATM\_GetCardData & ATM\_ApprovedCard flow charts

### **Application Layer:**

### **Card App**

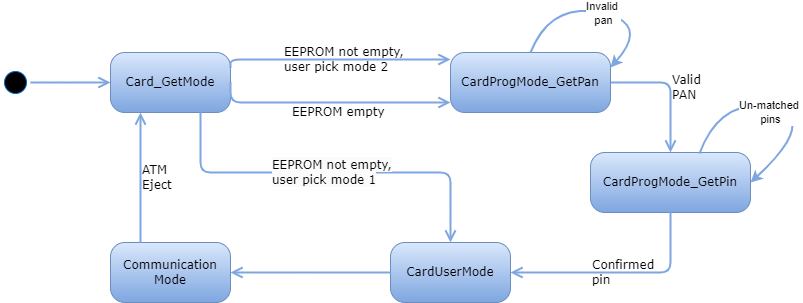


Figure Card App state machine

### **ATM App**

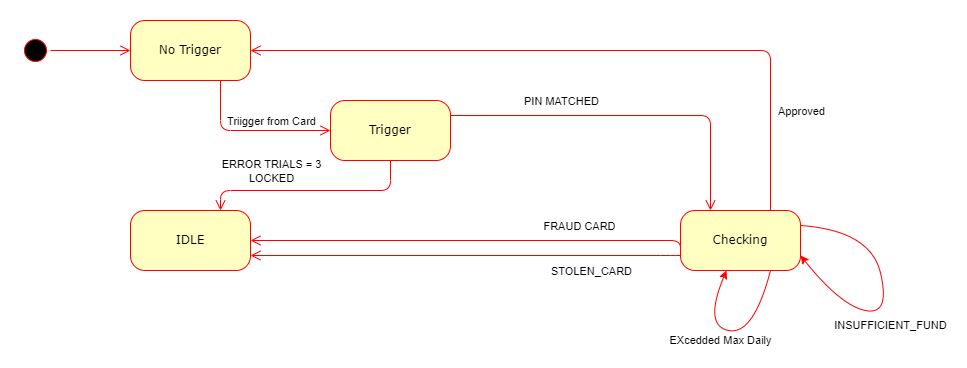


Figure ATM App state machine

