

**Air Conditioning Project**

**Team Members:**

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

Contents

[INTRODUCTION 1](#_Toc132230076)

[High Level Design 2](#_Toc132230077)

[**01)** **Layered Architecture** 2](#_Toc132230078)

[**02)** **Modules Description** 3](#_Toc132230079)

[**03)** **Drivers’ Documentation** 4](#_Toc132230080)

[Low Level Design 14](#_Toc132230081)

[Figure 1: Project Layered Architecture 2](file:///D:\02_workspace\MovingCarProject\Moving_Car.docx#_Toc132213547)

# INTRODUCTION

# High Level Design

## **Layered Architecture**

Figure 1: Project Layered Architecture

## **Modules Description**

**MCAL Layer:**

* **DIO:** For controlling GPIO pins
* **External Interrupt:** this module takes place in MCAL layer. it controlsthree external hardware interrupts on pins PD2, PD3, and PB2 which are referred to as INT0, INT1, and INT2 respectively. External interrupts can be level-triggered or edge-triggered.  
  We program this triggering. INT0 and INT1 can be level-triggered and edge-triggered whereas INT2 can be only edge-triggered.
* **Timer:** Provides an interface with timer 0 low-level capabilities.
* **ADC:**

**HAL Layer:**

* **Button:**
* **Keypad:**
* **LCD:** Use for display data
* **HTimer:** Provides high-level functions using the lower level timer 0 module capabilities.
* **HExtInt:** Provides high-level functions using the lower level External interrupt module capabilities.

**Service Layer:**

**Application Layer:**

## **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)

* **External Interrupt:**

// 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));

* **Timer:**

/\*\*

\* \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

\*/

**HAL Layer:**

**Button:**

/\*\*

\* \brief : This Function Use To Initialize Button Pin And Port

\*

\* \param ST\_PUSH\_BTN\_t \*btn

\*

\* \return Std\_ReturnType

\*/

Std\_ReturnType PUSH\_BTN\_intialize(const ST\_PUSH\_BTN\_t \*btn)

/\*\*

\* \brief : This Function Will Use To Read Button Statues If It Is Pressed Or Releassed

\*

\* \param ST\_PUSH\_BTN\_t \*btn

\* \param EN\_PUSH\_BTN\_state\_t \*btn\_state

\*

\* \return Std\_ReturnType

\*/

Std\_ReturnType PUSH\_BTN\_read\_state(const ST\_PUSH\_BTN\_t \*btn , EN\_PUSH\_BTN\_state\_t \*btn\_state)

* **Keypad:**
* **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();

* **HExtInt:**

/\*

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));

* **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)

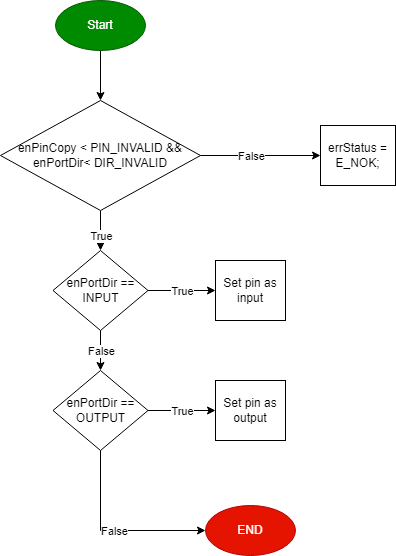
**Application Layer:**

# Low Level Design

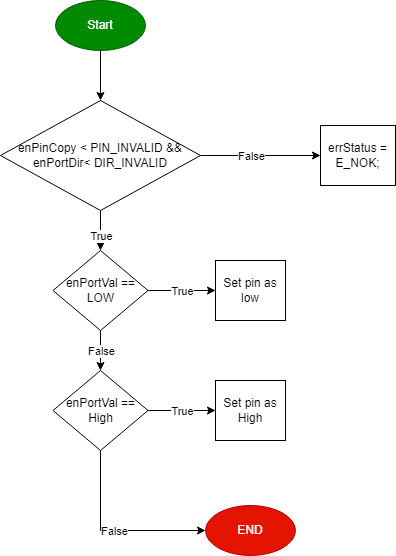
**MCAL Layer:**

* **DIO:**

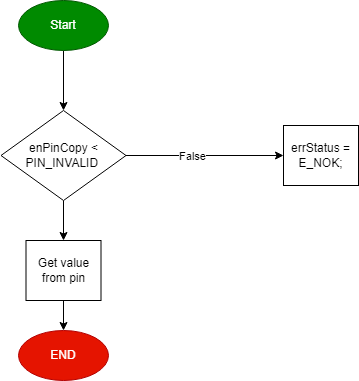
Sint8\_t **DIO\_s8SETPinDir** (enu\_pin enPinCopy, enu\_dir enPortDir)

****

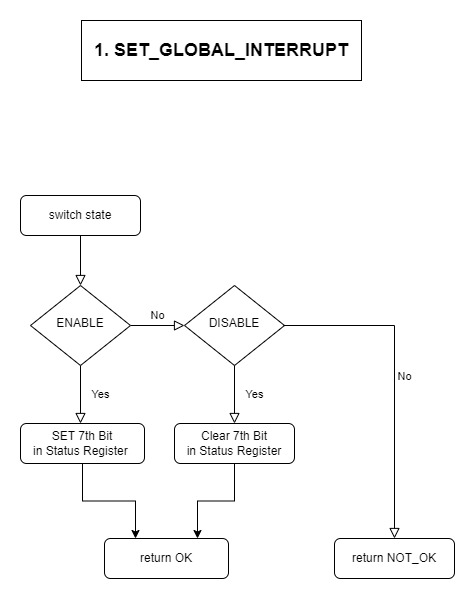
Sint8\_t **DIO\_s8SETPinVal** (enu\_pin enPinCopy, enu\_val enPortVal)

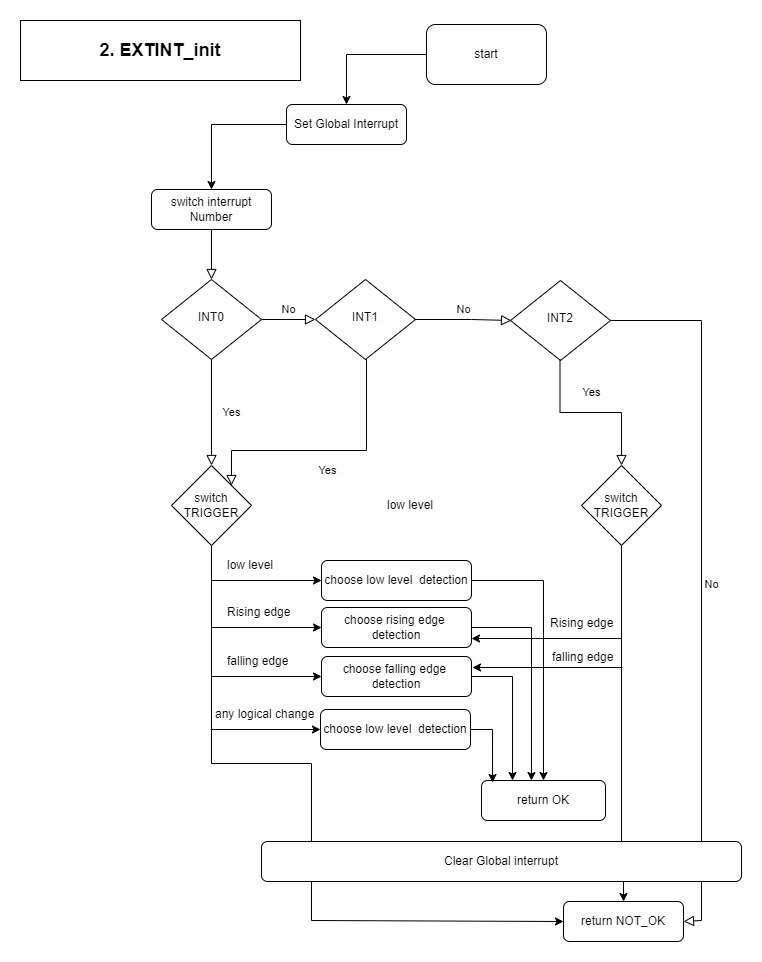
****

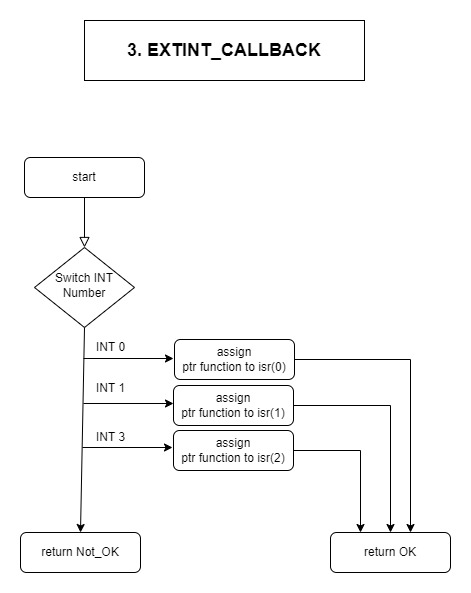
Sint8\_t **DIO\_s8GETPinVal** (enu\_pin enPinCopy, Uint8\_t\* pu8Val)

****

* **External Interrupt:**

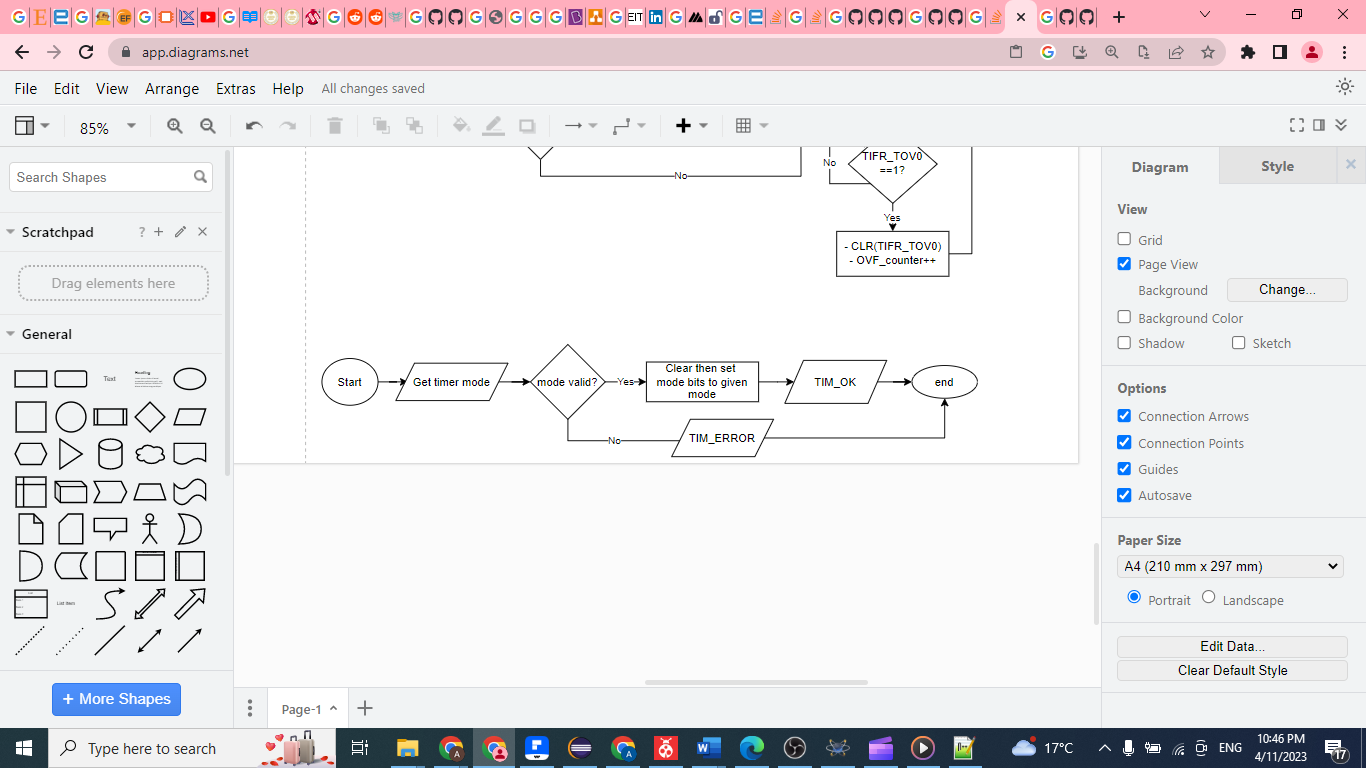






* **Timer:**

**TIM0\_Init**



**TIM0\_Start**

****

**TIM0\_Stop**

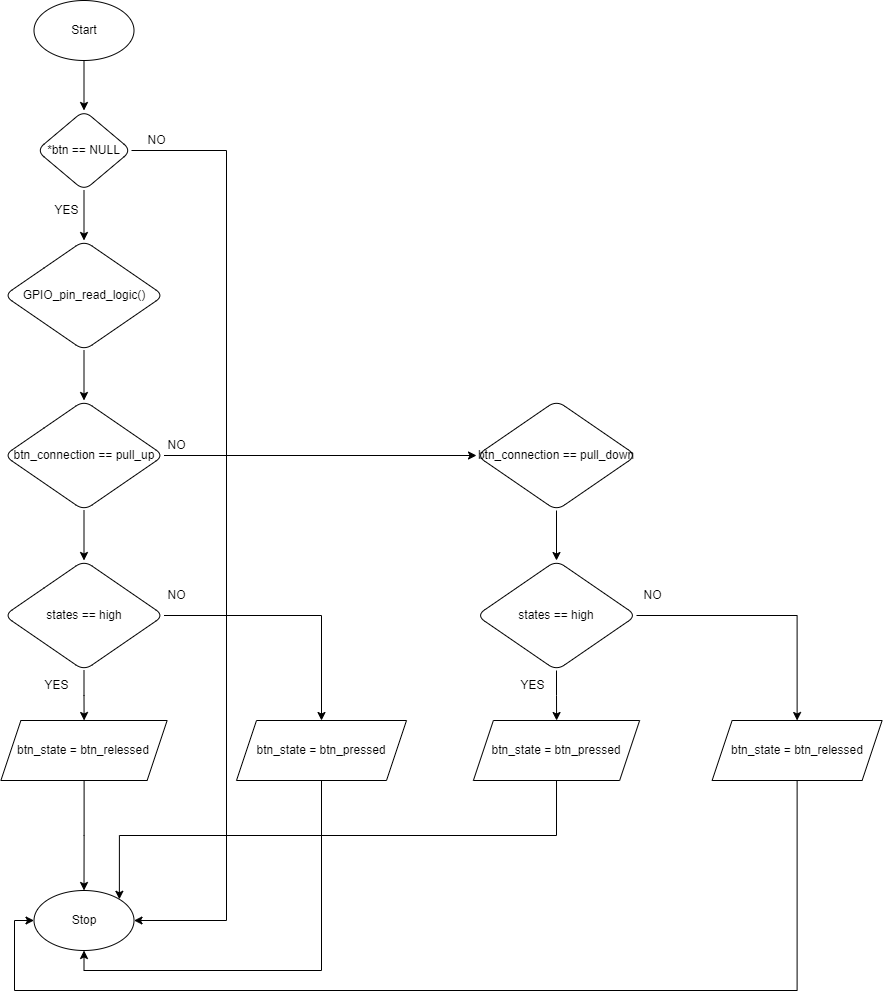
****

**HAL Layer:**

* **Button:**

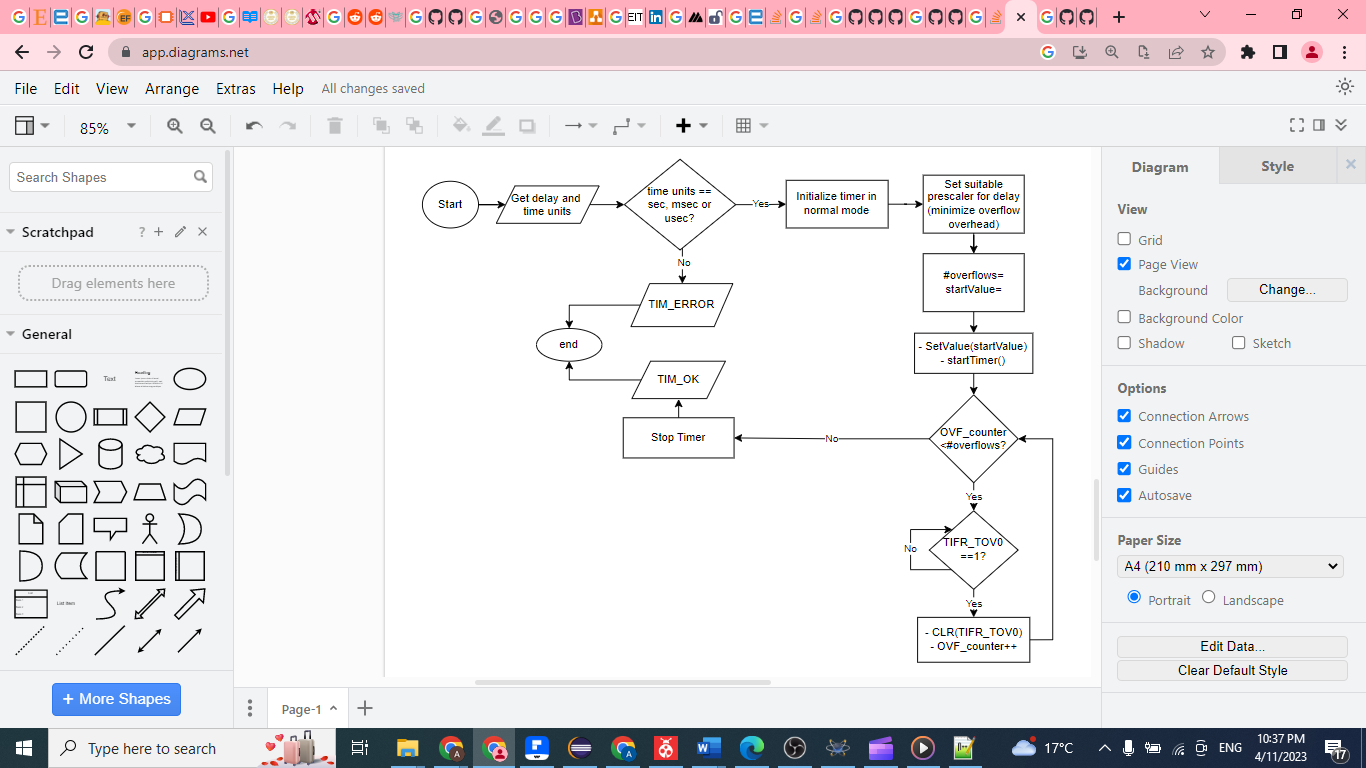
**Diagram

Description automatically generated**

****

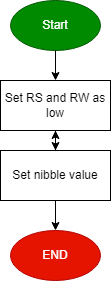
* **Timer0**

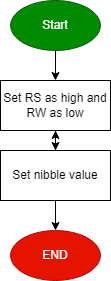
**HTIM\_SyncDelay**



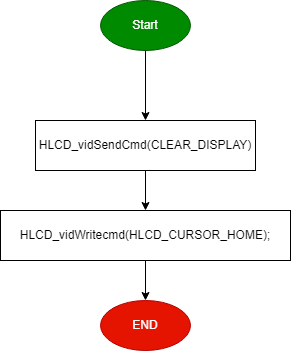
* **LCD:**

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

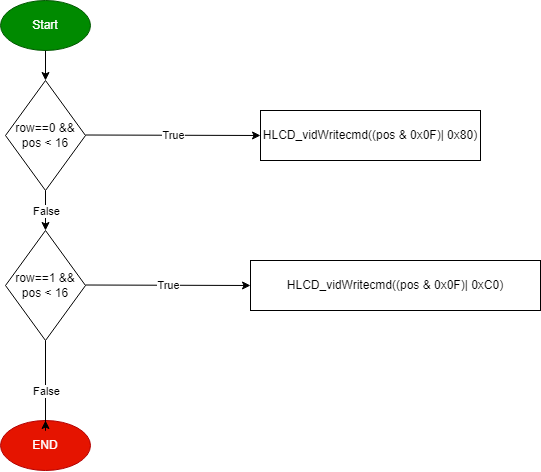
**void** **HLCD\_vidWritecmd**(Uint8\_t u8commandCopy)

****

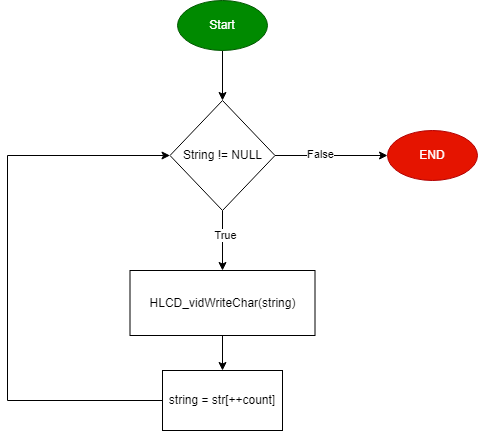
**void** **HLCD\_vidWriteChar**(Uint8\_t u8CharCopy)

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

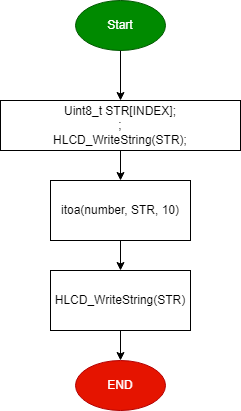
**void** **HLCD\_gotoXY**(Uint8\_t row, Uint8\_t pos)

****

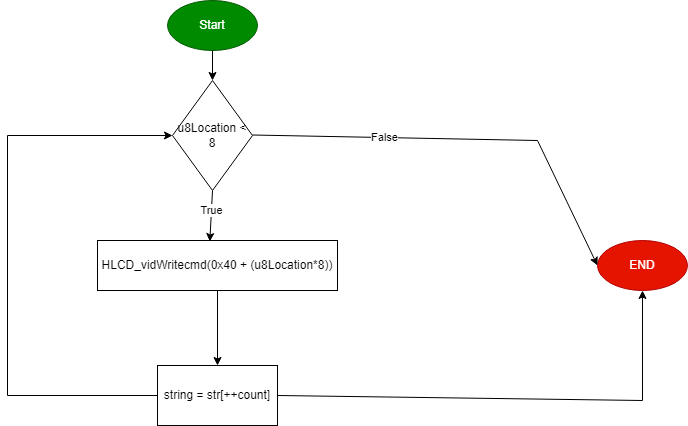
**void** **HLCD\_WriteString**(Uint8\_t\* str)

****

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

****

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

****

**Application Layer:**

* APP\_Init