

**Moving Car 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

In our project we have a four-driving wheel robot and moves in a rectangular shape, on this document we will illustrate the module design and how they integrate with each other, we’ll also discuss the used APIs in more detail and providing the flowchart for each function in each module, and making layered architecture.

In this project we used PWM for controlling motor speed, TIMER on normal mode for controlling the duration of motor, DIO for GPIO Pins and External Interrupt Module for Start / Stop Motor

# High Level Design

## **Layered Architecture**

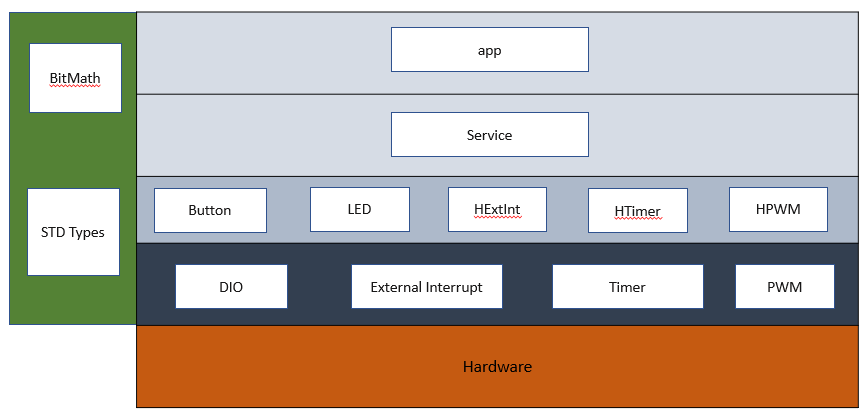


Figure : Project Layered Architecture

## **Modules Description**

**MCAL Layer:**

* **DIO:**
* **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.
* **PWM:** a modulation technique that generates variable-width pulses to represent the amplitude of an analog input signal, and directly communicates to hardware.
* **PWM Normal**: it generates the functionality of PWM using the normal mode of the timer.

**HAL Layer:**

* **Button:**
* **LED**
* **HPWM:** Is in Middle layer which Application can communicate to MCAL layer**.**
* **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.
* **HPWM\_Normal: :** Provides high-level functions using the lower level PWM\_NORMAL\_ MODE module capabilities.

**Service Layer:**

**Application Layer:**

## **Drivers’ Documentation**

**MCAL Layer:**

* **DIO:**

/\*\*

\* \brief : This Function Use To Initialize Port Direction Input Or Output

\*

\* \param ST\_pin\_config\_t \_pin\_config

\*

\* \return Std\_ReturnType

\*/

Std\_ReturnType GPIO\_pin\_direction\_intialize(const ST\_pin\_config\_t \*\_pin\_config)

/\*\*

\* \brief : This Function USe To Get Direction Of The Pin

\*

\* \param ST\_pin\_config\_t \_pin\_config

\* \param EN\_direction\_t direction\_status

\*

\* \return Std\_ReturnType

\*/

Std\_ReturnType GPIO\_pin\_get\_direction\_status(const ST\_pin\_config\_t \*\_pin\_config , EN\_direction\_t \*direction\_status)

/\*\*

\* \brief : This Function Use To Write Logic On The Pin

\*

\* \param ST\_pin\_config\_t \_pin\_config

\* \param EN\_logic\_t logic

\*

\* \return Std\_ReturnType

\*/

Std\_ReturnType GPIO\_pin\_write\_logic(const ST\_pin\_config\_t \*\_pin\_config , EN\_logic\_t logic)

/\*\*

\* \brief : This Function Use To Read Logic On The Pin

\*

\* \param ST\_pin\_config\_t \_pin\_config

\* \param EN\_logic\_t logic

\*

\* \return Std\_ReturnType

\*/

Std\_ReturnType GPIO\_pin\_read\_logic(const ST\_pin\_config\_t \*\_pin\_config , EN\_logic\_t \*logic\_status)

* **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

\*/

en\_TIMErrorState\_t TIM0\_SetOVFCallback(void (\*pv\_a\_CallbackFn)(void));

* **PWM:** 
  + **enu\_timer1Status\_t Timer1\_enuInit (enu\_timer1Mode\_t)**

**Author :** Bassel Yasser Mahmoud

**Function Name:** Timer1\_enuInit

**Function Description**: Initialize Timer1 to Fast PWM Mode

**Arguments:** copy\_enTmerMode {TIMER1\_OVF\_MODE, TIMER1\_FAST\_PWM\_MODE,}

**Return:** enu\_timer1Status\_t {TIMER1\_OK or TIMER1\_NOK}

* + **enu\_timer1Status\_t Timer1\_enuSetPrescallar(enu\_timer1Prescalar\_t)**

**Author**  : Bassel Yasser Mahmoud

**Function Nam**e : Timer1\_enuSetPrescallar

**Function Description** : Set Prescaller

**Arguments:** Timer1\_enuSetPrescallar {TIMER1\_PRE\_1, TIMER1\_PRE\_64, TIMER1\_PRE\_256}

**Return** : enu\_timer1Status\_t {TIMER1\_OK or TIMER1\_NOK}

* + **enu\_timer1Status\_t Timer1\_enuFastPWMInit(enu\_pwm1Mode\_t)**

**Author**  : Bassel Yasser Mahmoud

**Function Name** : Timer1\_enuFastPWMInit

**Function Description :** Set PWM Mode

**Arguments: c**opy\_enPWMMode {TIMER1\_PWM\_NORMAL, TIMER1\_PWM\_CLR\_ON\_CMP, TIMER1\_PWM\_SET\_ON\_CMP}

**Return**  : enu\_timer1Status\_t {TIMER1\_OK or TIMER1\_NOK}

* + **enu\_timer1Status\_t Timer1\_enuPWMGenerate (Uchar8\_t)**

**Author**  : Bassel Yasser Mahmoud

**Function Name** : Timer1\_enuPWMGenerate

**Function Description** : Generate PWM

**Arguments** : copy\_u8DutyCycle {1 ~ 100}

**Return**  : enu\_timer1Status\_t {TIMER1\_OK or TIMER1\_NOK}

* + **enu\_timer2Status\_t Timer2\_enuInit (enu\_timer2Mode\_t)**

**Author :** Bassel Yasser Mahmoud

**Function Name:** Timer1\_enuInit

**Function Description**: Initialize Timer2 to Fast PWM Mode

**Arguments:** copy\_enTmerMode {TIMER2\_OVF\_MODE, TIMER2\_FAST\_PWM\_MODE,}

**Return:** enu\_timer1Status\_t {TIMER2\_OK or TIMER2\_NOK}

* + **enu\_timer2Status\_t Timer2\_enuSetPrescallar(enu\_timer2Prescalar\_t)**

**Author**  : Bassel Yasser Mahmoud

**Function Nam**e : Timer2\_enuSetPrescallar

**Function Description** : Set Prescaller

**Arguments:** Timer2\_enuSetPrescallar {TIMER2\_PRE\_1, TIMER2\_PRE\_64, TIMER2\_PRE\_256}

**Return** : enu\_timer2Status\_t {TIMER2\_OK or TIMER2\_NOK}

* + **enu\_timer2Status\_t Timer2\_enuFastPWMInit(enu\_pwm2Mode\_t)**

**Author**  : Bassel Yasser Mahmoud

**Function Name** : Timer2\_enuFastPWMInit

**Function Description :** Set PWM Mode

**Arguments: c**opy\_enPWMMode {TIMER2\_PWM\_NORMAL, TIMER2\_PWM\_CLR\_ON\_CMP, TIMER2\_PWM\_SET\_ON\_CMP}

**Return**  : enu\_timer2Status\_t {TIMER2\_OK or TIMER2\_NOK}

* + **enu\_timer2Status\_t Timer2\_enuPWMGenerate (Uchar8\_t)**

**Author**  : Bassel Yasser Mahmoud

**Function Name** : Timer2\_enuPWMGenerate

**Function Description** : Generate PWM

**Arguments** : copy\_u8DutyCycle {1 ~ 100}

**Return**  : enu\_timer2Status\_t {TIMER2\_OK or TIMER2\_NOK}

* **PWM NORMAL:**

/\*Description : This function selects the normal mode and enables the GLOBAL\_INTERRUPT and overflow interrupt for timer2

ARGS : void

return : void\*/

void timer2\_init(void);

/\*Description : This function selects the prescaler (clk/1024). the timer start counting once we call this function.

ARGS : void

return : void\*/

void timer2\_start(void);

/\*Description : This function selects the no clock source option. the timer stop counting once we call this function.

ARGS : void

return : void\*/

void timer2\_stop(void);

/\*Description : This function calculate the on time based on duty cycle we need .

ARGS : takes the duty cycle

return : void\*/

void timer2\_set\_pwm\_normal(Uchar8\_t dutycycle);

**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)

* **LED:**

/\*\*

\* \brief : This Function Is Used To Initialize Pin And Port Led

\*

\* \param ST\_LED\_t \*led

\*

\* \return Std\_ReturnType

\*/

Std\_ReturnType LED\_initialize(const ST\_led\_t \*led)

/\*\*

\* \brief : This Function Used To Turn Led On

\*

\* \param ST\_led\_t \*led

\*

\* \return Std\_ReturnType

\*/

Std\_ReturnType LED\_turn\_on(const ST\_led\_t \*led)

/\*\*

\* \brief : This Function Used To Turn Led Off

\*

\* \param ST\_led\_t \*led

\*

\* \return Std\_ReturnType

\*/

Std\_ReturnType LED\_turn\_off(const ST\_led\_t \*led)

/\*\*

\* \brief : This Function Used To Toggle Led

\*

\* \param ST\_led\_t \*led

\*

\* \return Std\_ReturnType

\*/

Std\_ReturnType LED\_toggle(const ST\_led\_t \*led)

* **Motor**

/\*\*

\* \brief initialize motor pins

\* \param pst\_a\_Motor reference to desired motor

\* \return en\_MotorError\_t

\*/

en\_MotorError\_t DCM\_Init(st\_Motor\_t \*pst\_a\_Motor);

/\*\*

\* \brief Function to start the given motor

\* \param pst\_a\_Motor reference to desired motor

\* \return en\_MotorError\_t

\*/

en\_MotorError\_t DCM\_Start(st\_Motor\_t \*pst\_a\_Motor);

/\*\*

\* \brief Function to stop the given motor

\* \param pst\_a\_Motor reference to desired motor

\* \return en\_MotorError\_t

\*/

en\_MotorError\_t DCM\_Stop(st\_Motor\_t \*pst\_a\_Motor);

* **HPWM:** 
  + **enu\_pwmStatus\_t pwm\_enInit(void)**

**Author** : Bassel Yasser Mahmoud

**Function Name** : pwm\_enInit

**Function Description :** Initialize PWM to be fast PWM, set prescaller, Set PWM Mode

**Arguments** : void

**Return**  : enu\_pwmStatus\_t {PWM\_OK or PWM\_NOK}

* + **enu\_pwmStatus\_t pwm\_enGenerate(Uchar8\_t)**

**Author** : Bassel Yasser Mahmoud

**Function Name** : pwm\_enGenerate

**Function Description :** Generate PWM signal

**Arguments** : Uchar8\_t

**Return**  : enu\_pwmStatus\_t {PWM\_OK or PWM\_NOK}

* **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));

* **HPWM\_NORMAL:**

/\*

Description : This function selects the normal mode and enables the GLOBAL\_INTERRUPT and overflow interrupt for timer2

ARGS : void

return : void

\*/

void H\_PWM\_NORMAL\_init(void);

/\*

Description : This function selects the no clock source option. the timer stop counting once we call this function.

ARGS : void

return : void

\*/

void H\_PWM\_NORMAL\_stop(void);

/\*

Description : This function calculate the on time based on duty cycle we need . then start the timer

ARGS : takes the duty cycle

return : void

\*/

void H\_PWM\_NORMAL\_setDutyCycle(Uchar8\_t dutycycle);

**Application Layer:**

/\*\*

\* \brief Function to move car forward

\* \return void

\*/

static void APP\_moveForward(void);

/\*\*

\* \brief Function to stop the car

\* \return void

\*/

static void APP\_Stop(void);

/\*\*

\* \brief Function to rotate car right

\* \return void

\*/

static void APP\_rotate(void);

/\*\*

\* \brief Function to call in EXTI1 ISR when button 0 is pressed

\* \return void

\*/

void btn0\_callback(void);

/\*\*

\* \brief Function to call in EXTI0 ISR when button 1 is pressed

\* \return void

\*/

void btn1\_callback(void);

/\*\*

\* \brief Initialize all used modules

\* \return void

\*/

void initMain(void);

/\*\*

\* \brief Application main logic

\* \return void

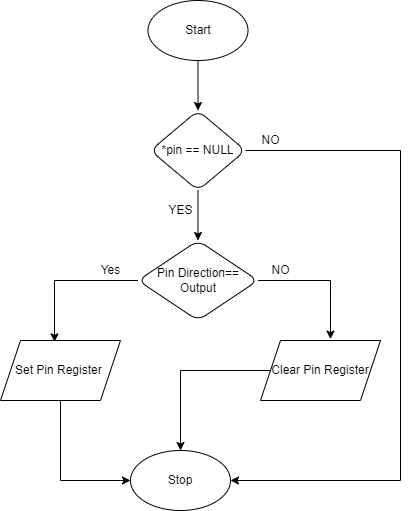
\*/

void appMain(void);

# Low Level Design

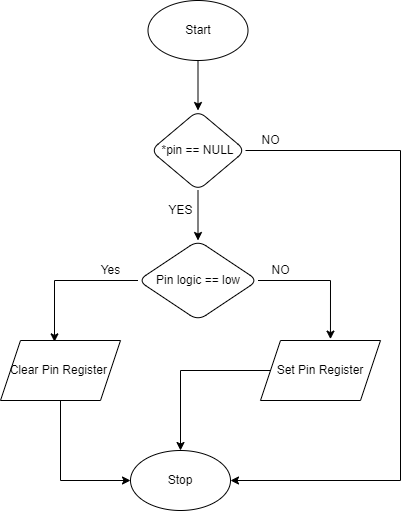
**MCAL Layer:**

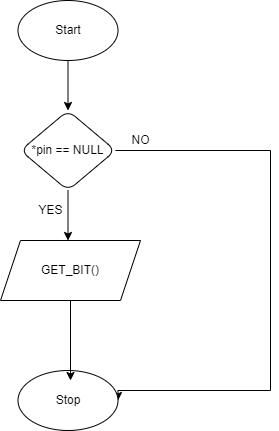
* **DIO:**

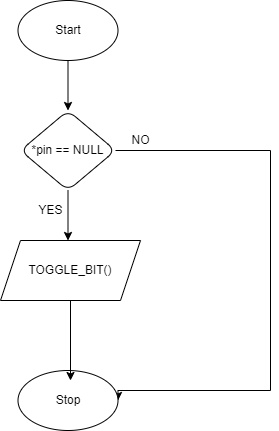
****

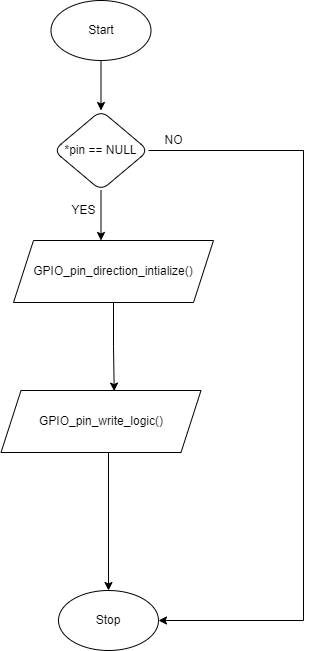
**Diagram

Description automatically generated**

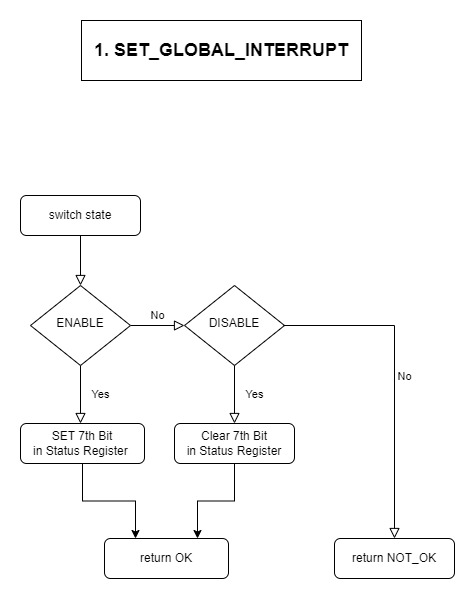
****

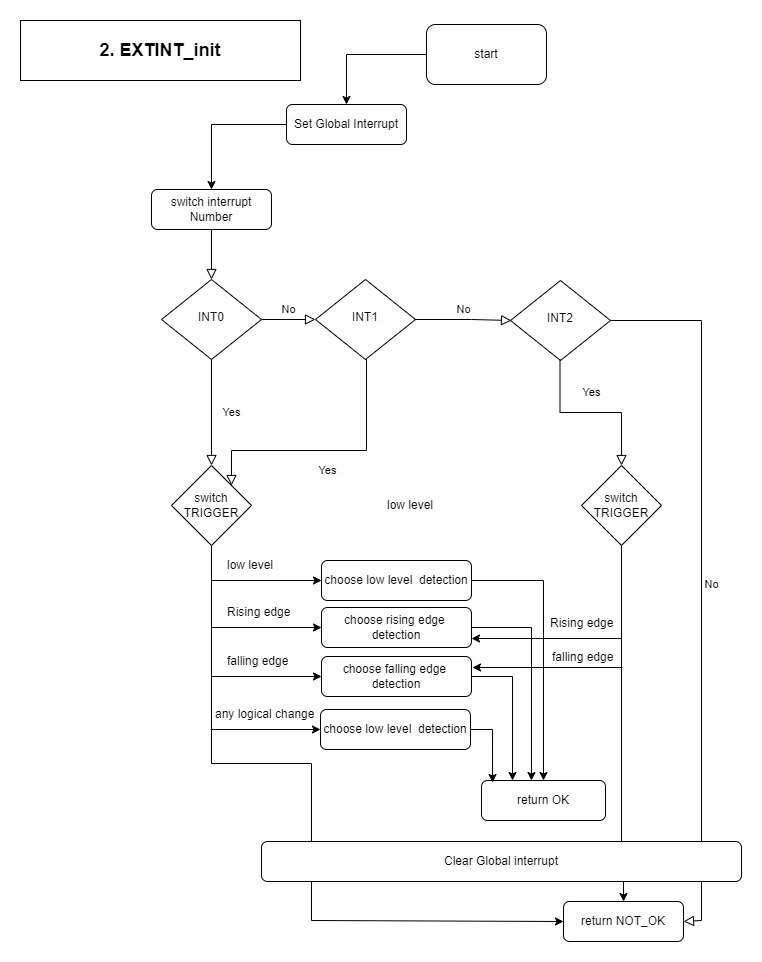
****

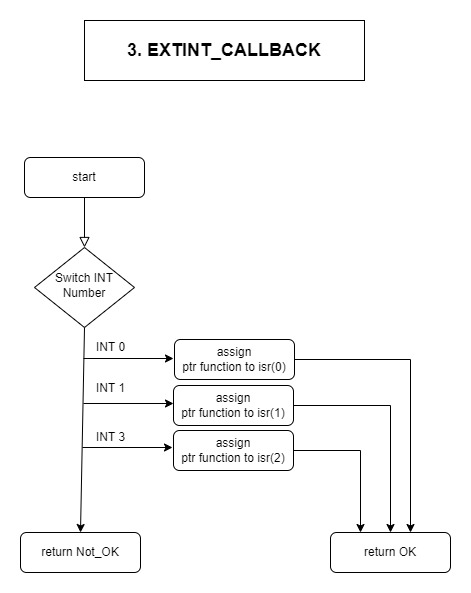
****

****

* **External Interrupt:**

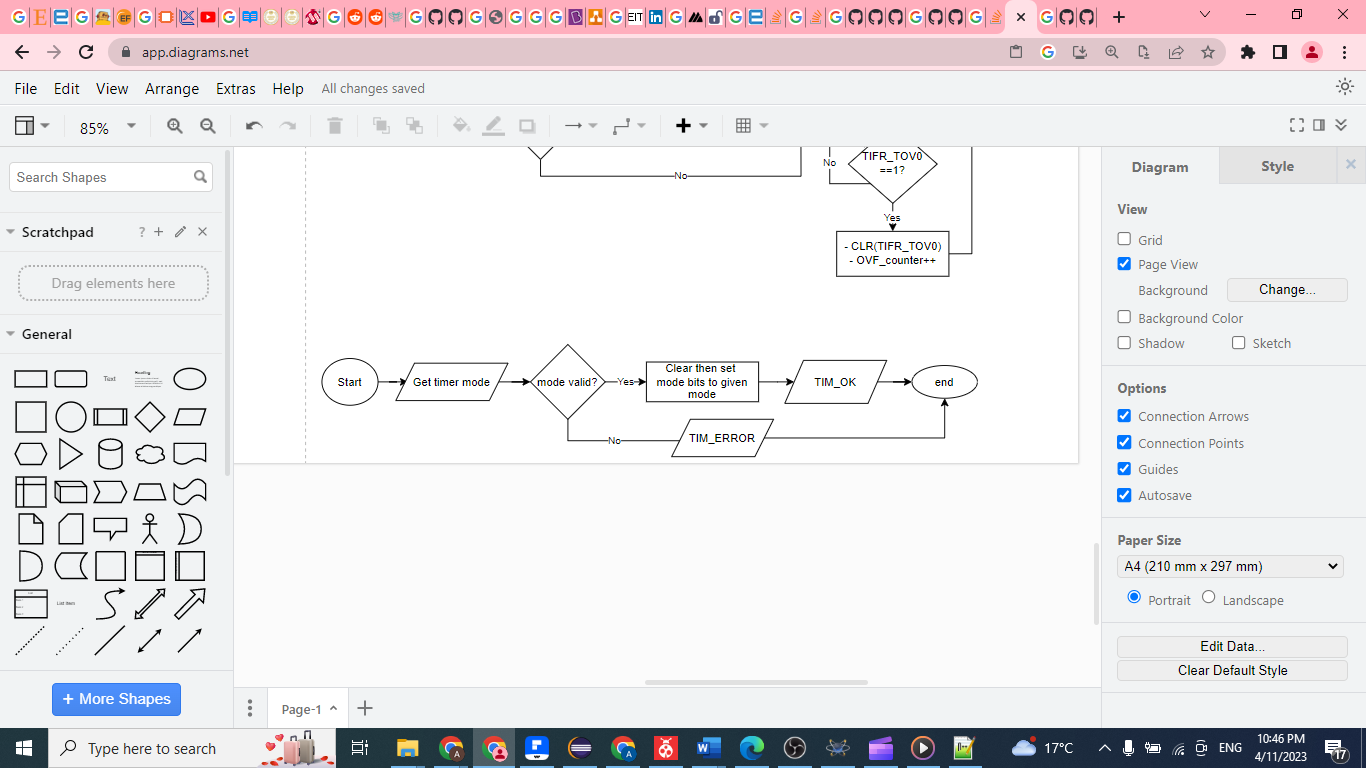






* **Timer:**

**TIM0\_Init**

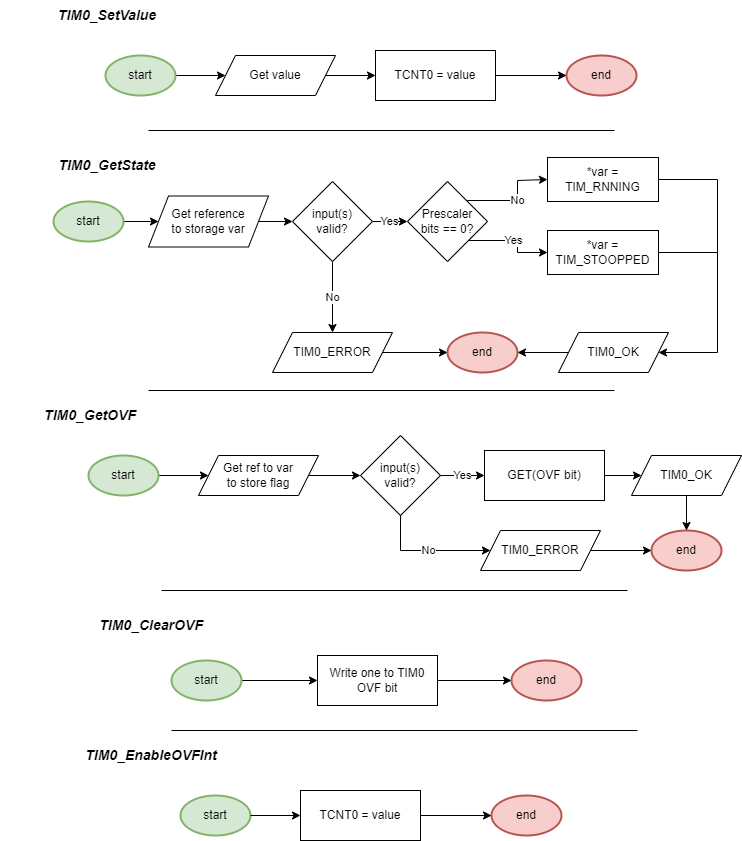


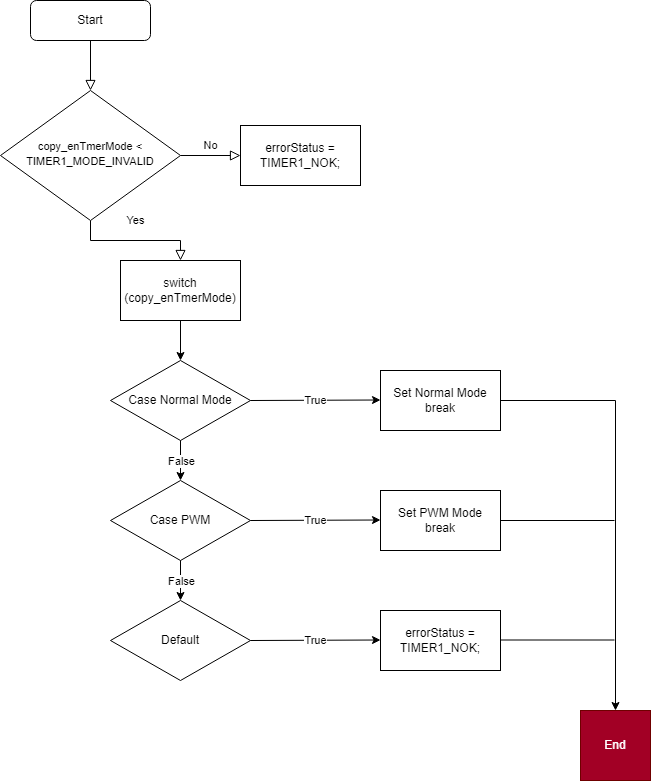
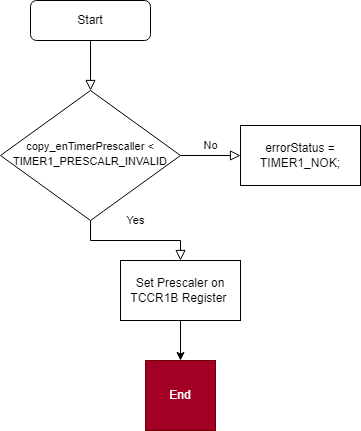
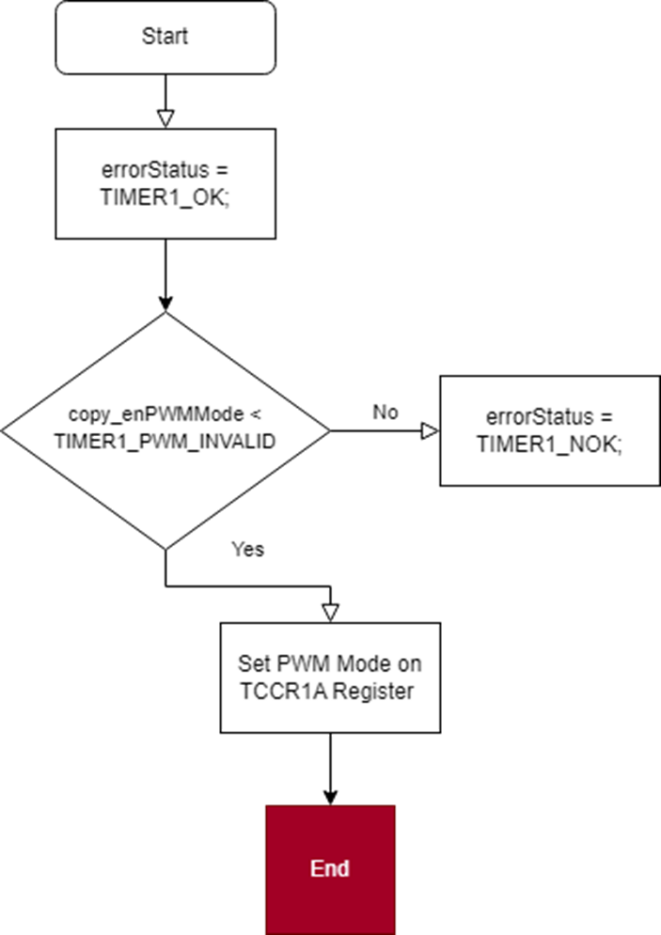
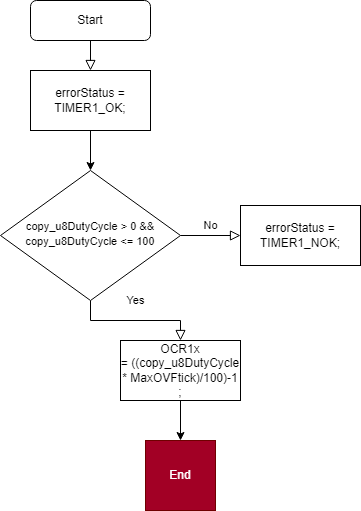
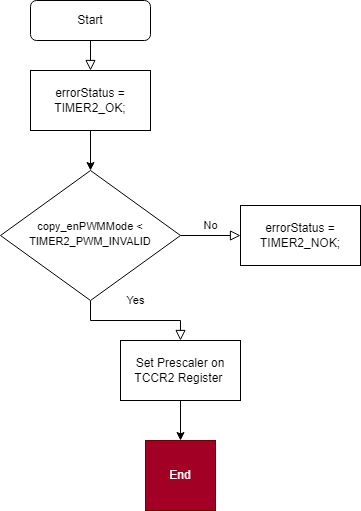
**TIM0\_Start**

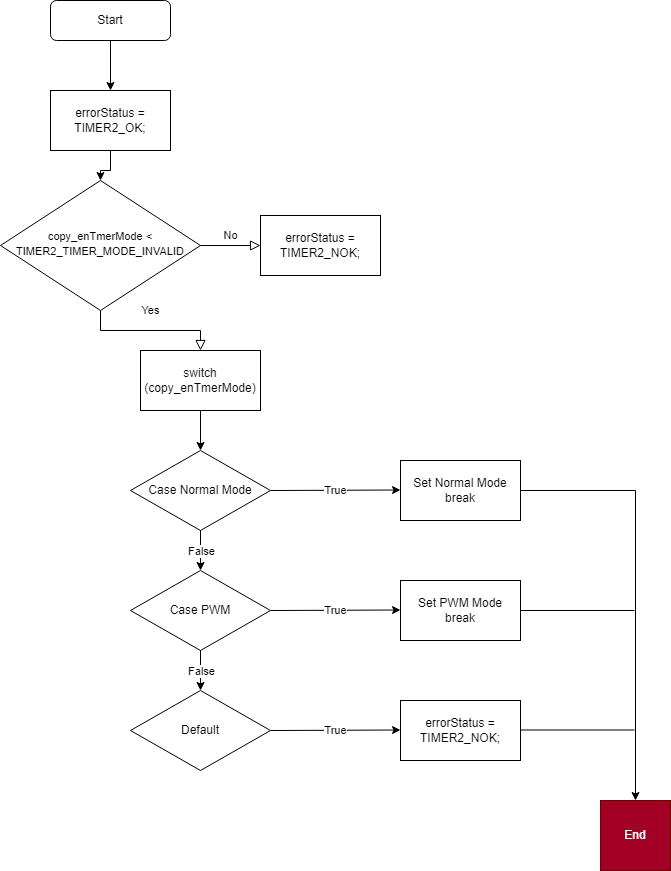
****

**TIM0\_Stop**

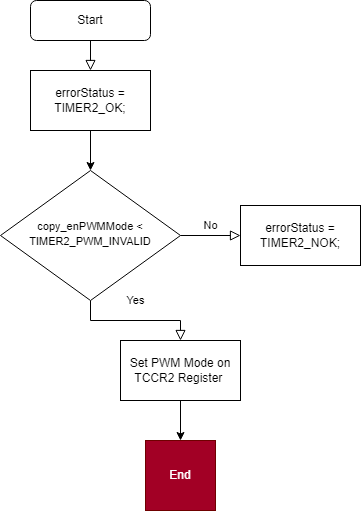
****

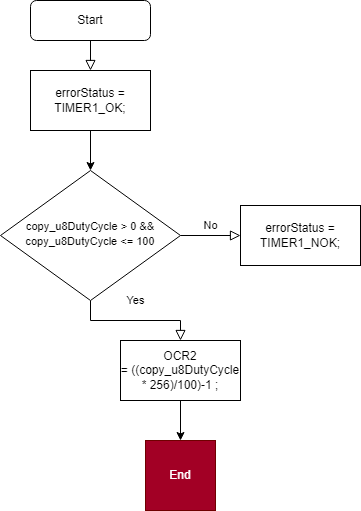


* **PWM:** 
  + **enu\_timer1Status\_t Timer1\_enuInit (enu\_timer1Mode\_t)**
  + **enu\_timer1Status\_t Timer1\_enuSetPrescallar(enu\_timer1Prescalar\_t)**
  + **enu\_timer1Status\_t Timer1\_enuFastPWMInit(enu\_pwm1Mode\_t)**
  + **enu\_timer1Status\_t Timer1\_enuPWMGenerate (Uchar8\_t)**
  + **enu\_timer2Status\_t Timer2\_enuSetPrescallar(enu\_timer2Prescalar\_t)**
  + **enu\_timer2Status\_t Timer2\_enuInit (enu\_timer2Mode\_t)**

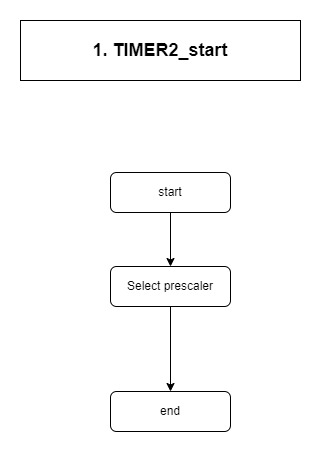
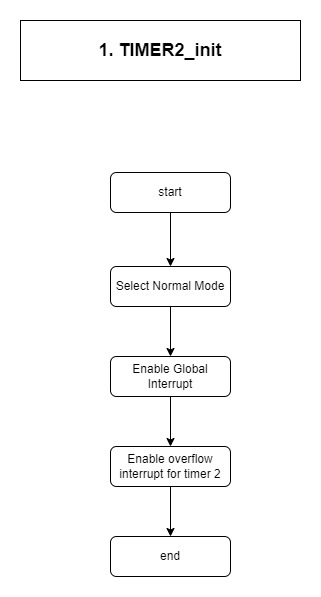
****

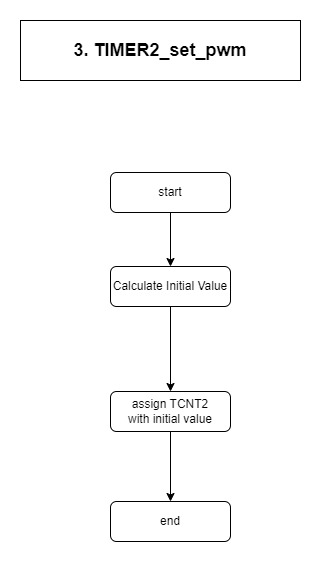
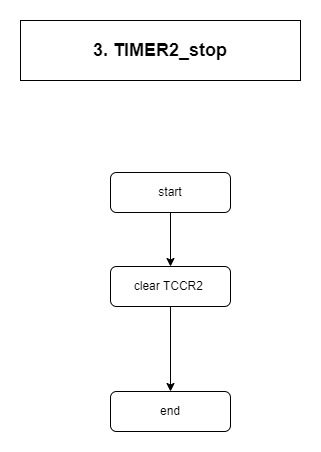
* + **enu\_timer2Status\_t Timer2\_enuFastPWMInit(enu\_pwm2Mode\_t)**

****

****

* + **enu\_timer2Status\_t Timer2\_enuPWMGenerate (Uchar8\_t)**
* **PWM NORMAL:**



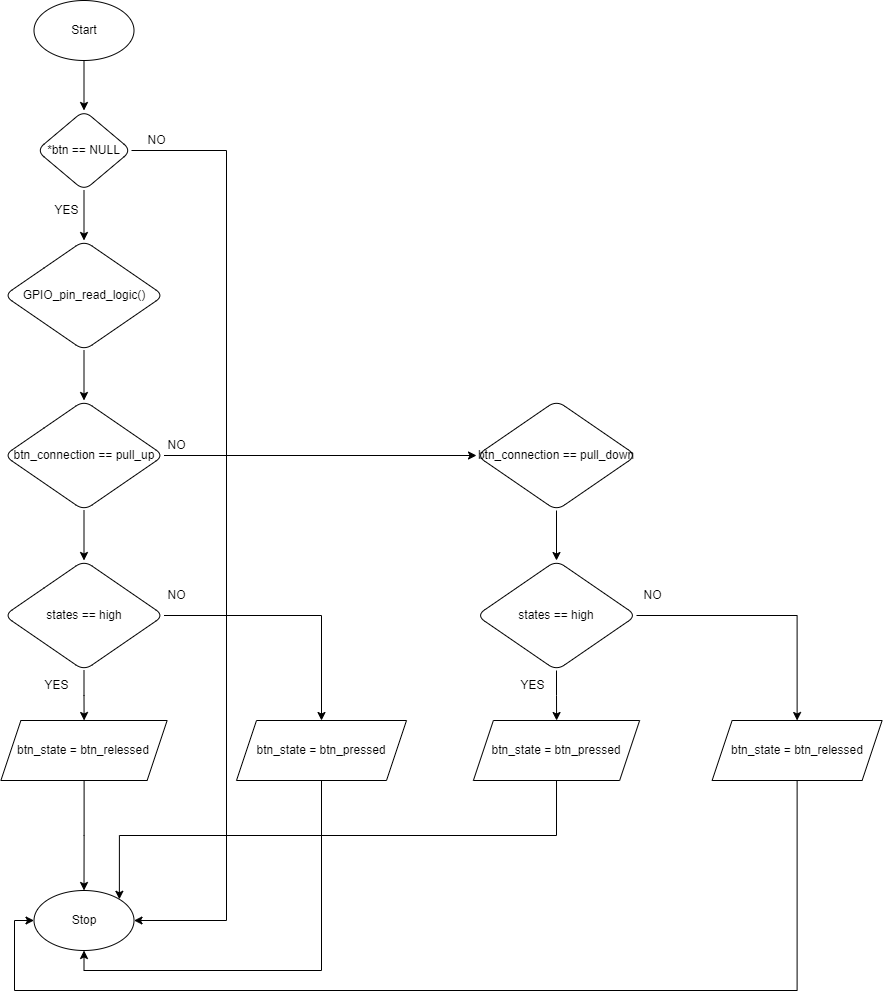


**HAL Layer:**

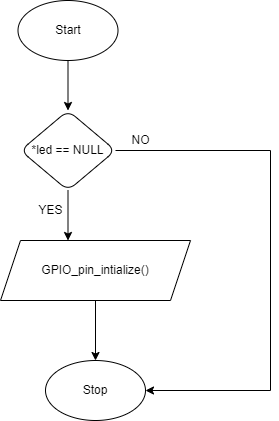
* **Button:**

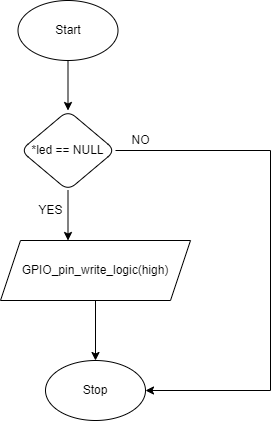
**Diagram

Description automatically generated**

****

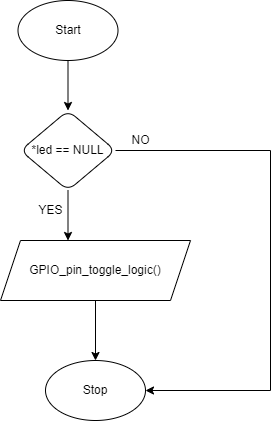
* **LED:**

****

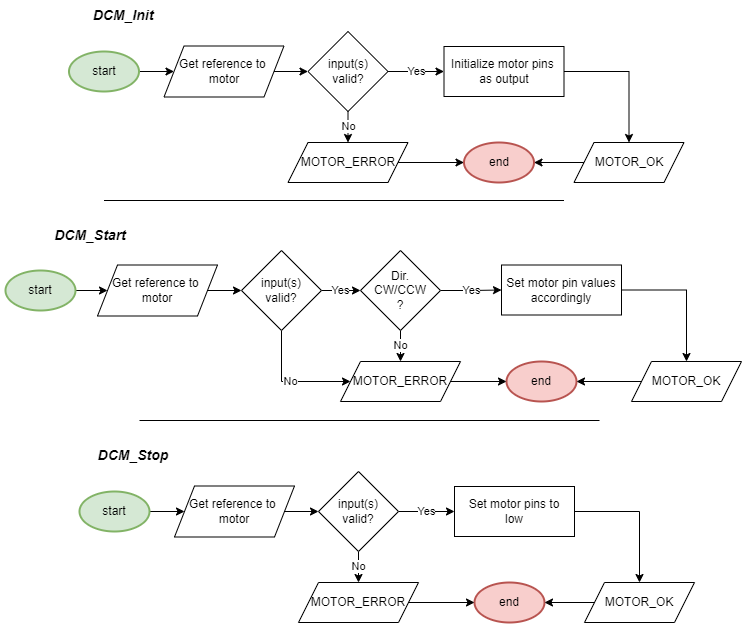
****

**Diagram

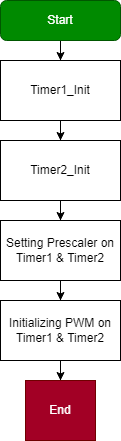
Description automatically generated**

****

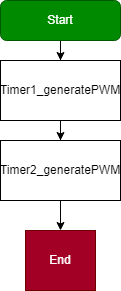
* **Motor**



* **HPWM:** 
  + **enu\_pwmStatus\_t pwm\_enInit(void)**

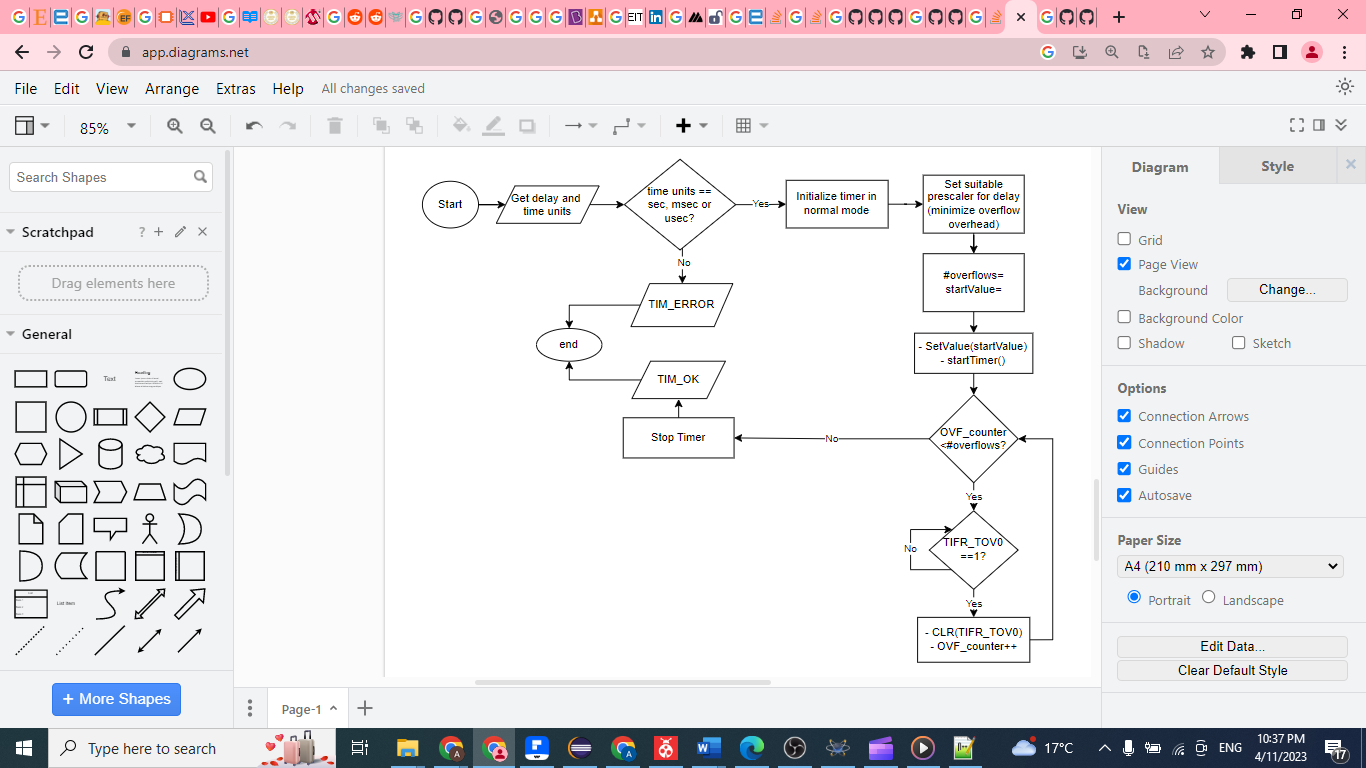


* + **enu\_pwmStatus\_t pwm\_enGenerate(Uchar8\_t)**

****

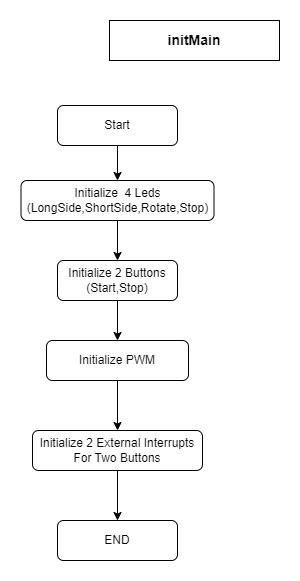
* **Timer0**

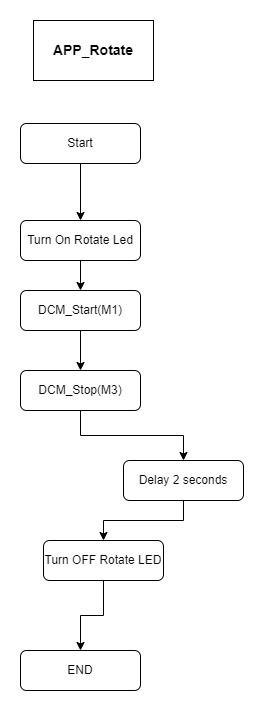
**HTIM\_SyncDelay**



**Application Layer:**

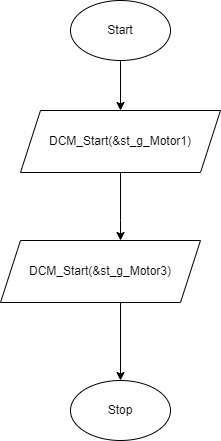
* APP\_Init

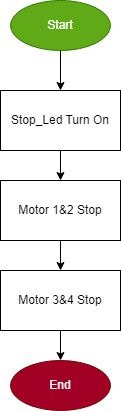




**APP\_Stop**

**APP\_moveForward**





* Main App State Machine

