SPRINTS

Project Design

OBSTACLE AVOIDANCE CAR

Version 1.0

Prepared by:

BitsTribe

Hazem Ashraf

Mohamed Abdelsalam

Abdelrhman Walaa

May 2023

Table of Content

1. Project Introduction....................4
1.1. System Requirements....................4
1.1.1. Hardware Requirements....................4
1.1.2. Software Requirements....................4
2. High Level Design....................6
2.1. System Architecture....................6
2.1.1. Definition....................6
2.1.2. Layered Architecture....................6
2.1.3. Project Circuit Schematic....................7
2.2. Block Diagram....................8
2.2.1. Definition....................8
2.2.2. Design....................8
2.3. System Modules....................9
2.3.1. Definition....................9
2.3.2. Design....................9
2.4. Modules Description....................10
2.4.1. DIO Module....................10
2.4.2. EXI Module....................10
2.4.3. TMR Module....................10
2.4.4. PWM Module....................10
2.4.5. BTN Module....................10
2.4.6. LCD Module....................11
2.4.7. KPD Module....................11
2.4.8. DCM Module....................11
2.4.9. ICU Module....................11
2.4.10. US Module....................12
2.5. Drivers' Documentation (APIs)....................13
2.5.1 Definition....................13
2.5.2. MCAL APIs....................13
2.5.2.1. DIO Driver APIs....................13
2.5.2.2. EXI Driver APIs....................15
2.5.2.3. TMR Driver APIs....................16
2.5.2.4. PWM Driver APIs....................19
2.5.3. HAL APIs....................20
2.5.3.1. BTN Driver APIs....................20
2.5.3.2. LCD Driver APIs....................20
2.5.3.3. KPD Driver APIs....................22
2.5.3.4. DCM Driver APIs....................22
2.5.3.5. ICU Driver APIs....................23
2.5.3.6. US Driver APIs....................24
2.5.4. APP APIs....................24

3. Low Level Design..25
3.1. MCAL Layer..25
3.1.1. DIO Module..25
3.1.2. EXI Module..28
3.1.3. TMR Module..30
3.2. HAL Layer..34
3.2.1. BTN Module..34
3.2.2. LCD Module..35
3.2.3. KPD Module..40
3.2.4. DCM Module..42
3.2.5. ICU Module..46
3.2.6. US Module..49
3.3. APP Layer..51
3.3.1. State Machine Diagram..51
3.3.2. Flowchart Diagram..52
4. Development Issues..53
4.1. Team Issues..53
4.2. Design Issues..53
4.3. Drivers Issues..53
4.4. Application Issues..53
5. References..54

Obstacle Avoidance Car Design

1. Project Introduction

This project involves developing software for a car robot that avoids any object in front.

1.1. System Requirements

1.1.1. Hardware Requirements

1. **ATmega32** microcontroller
2. Fourmotors (**M1**, **M2**, **M3**, **M4**)
3. **One** button to change default direction of rotation (**PBUTTON0**)
4. Keypad button 1 to start
5. Keypad button 2 to stop
6. **One Ultrasonic** sensor connected as follows:
    a. **Vcc to 5V in the Board**
    b. **GND to the ground In the Board**
    c. **Trig to PB3 (Port B, Pin 3)**
    d. **Echo to PB2 (Port B, Pin 2)**
7. **LCD**

1.1.2. Software Requirements

1. The car **starts initially** from **0 speed.**
2. The default rotation direction is to the **right.**
3. Press (Keypad Btn 1), (Keypad Btn 2) to start or stop the robot respectively.
4. **After Pressing Start:**
    a. The LCD will display a centered message in line 1 “**Set Def. Rot.**”.
    b. The LCD will display the selected option in line 2 “**Right**”.
    c. The robot will **wait** for **5 seconds** to choose between **Right** and **Left**
        i. When **PBUTTON0** is pressed **once**, the default rotation will be **Left** and the **LCD line 2 will be updated**.
        ii. When PBUTTON0 is pressed again, the default rotation will be **Right** and the **LCD line 2 will be updated**.
        iii. *For each press the default rotation will change and the LCD line 2 is updated*.
        iv. **After 5 seconds the default value of rotation is set**.
    d. The robot will move **after 2 seconds** from setting the default direction of rotation.

5. **For No obstacles or object is far than 70 centimeters:**
    a. The robot will move forward with 30% speed for 5 seconds.
    b. After 5 seconds it will move with 50% speed as long as there is no object or objects are located at more than 70 centimeters distance.
    c. The LCD will display the speed and moving direction in line 1: “**Speed:00% Dir: F/B/R/S**”, **F**: forward, **B**: Backwards, **R**: Rotating, and **S**: Stopped.
    d. The LCD will display Object distance in line 2 “**Dist.: 000 Cm**”.

6. **For Obstacles located between 30 and 70 centimeters:**
    a. The robot will decrease its speed to 30%.
    b. LCD data is updated.

7. **For Obstacles located between 20 and 30 centimeters:**
    a. The robot will stop and rotate 90 degrees to right/left according to the chosen configuration.
    b. The LCD data is updated.

8. **For Obstacles located less than 20 centimeters:**
    a. The robot will **stop**, move **backwards** with **30% speed** until distance is **greater than 20 and less than 30.**
    b. The LCD data is updated.
    c. Then perform point 7.

9. **Obstacles surrounding the robot (Bonus)**
    a. If the robot **rotated for 360 degrees without finding any distance greater than 20 it would stop.**
    b. LCD data will be updated.
    c. The robot will frequently (each 3 seconds) check if any of the obstacles was removed or not and move in the direction of the furthest object.

2. High Level Design

2.1. System Architecture

2.1.1. Definition

Layered Architecture (*Figure 1*) describes an architectural pattern composed of several separate horizontal layers that function together as a single unit of software.

Microcontroller Abstraction Layer (*MCAL*) is a software module that directly accesses on-chip MCU peripheral modules and external devices that are mapped to memory, and makes the upper software layer independent of the MCU.

Hardware Abstraction Layer (*HAL*) is a layer of programming that allows a computer OS to interact with a hardware device at a general or abstract level rather than at a detailed hardware level.

2.1.2. Layered Architecture

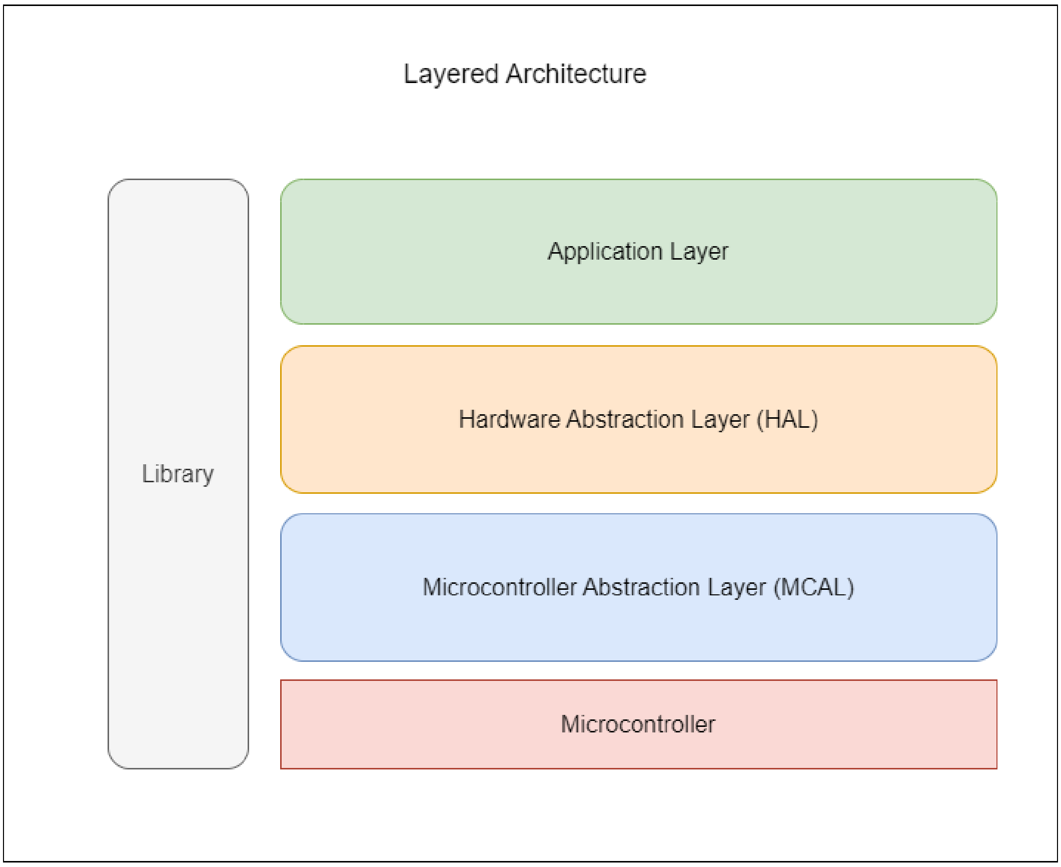

Layered Architecture
Library
Application Layer
Hardware Abstraction Layer (HAL)
Microcontroller Abstraction Layer (MCAL)
Microcontroller

Figure 1. Layered Architecture Design

2.1.3. Project Circuit Schematic

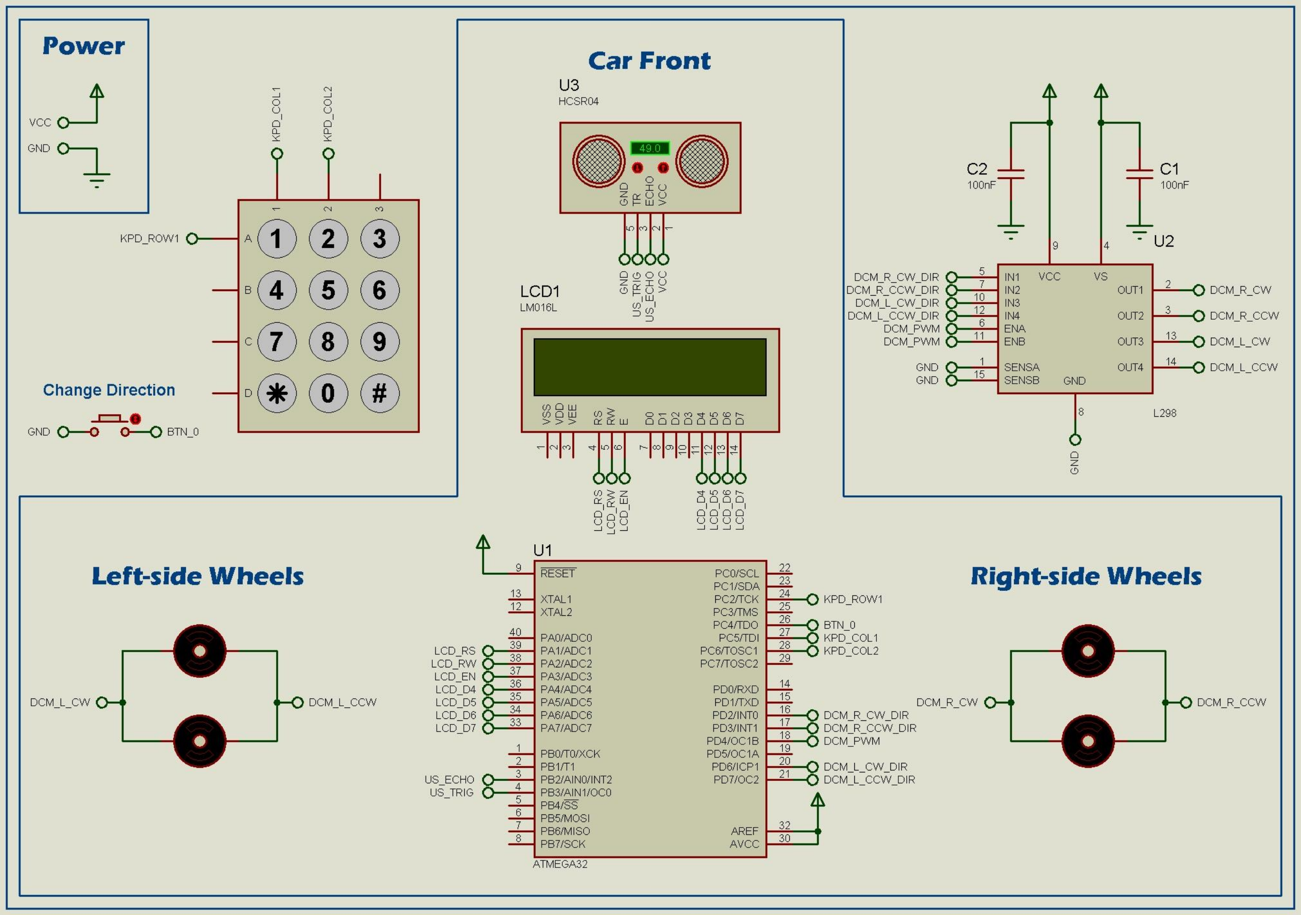


Figure 2. Project Circuit Schematic

2.2. Block Diagram

2.2.1. Definition

A *Block Diagram* (*Figure 3*) is a specialized flowchart used to visualize systems and how they interact.

Block Diagrams give you a high-level overview of a system so you can account for major system components, visualize inputs and outputs, and understand working relationships within the system.

2.2.2. Design

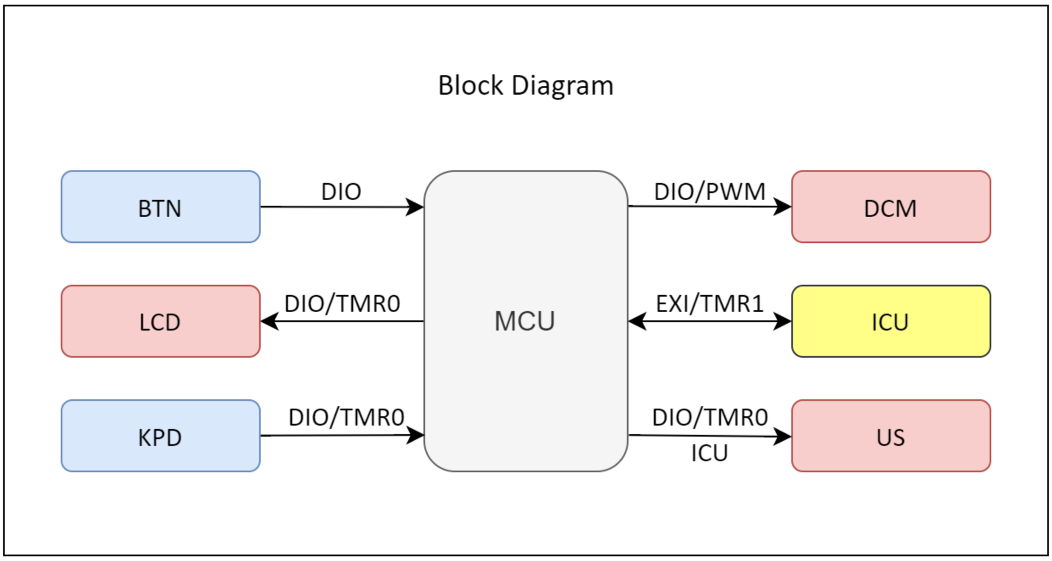


Figure 3. Block Diagram Design

System Input: Blue | *System Output*: Red | *System Input/Output*: Yellow

2.3. System Modules

2.3.1. Definition

A *Module* is a distinct assembly of components that can be easily added, removed or replaced in a larger system. Generally, a *Module* is not functional on its own.

In computer hardware, a *Module* is a component that is designed for easy replacement.

2.3.2. Design

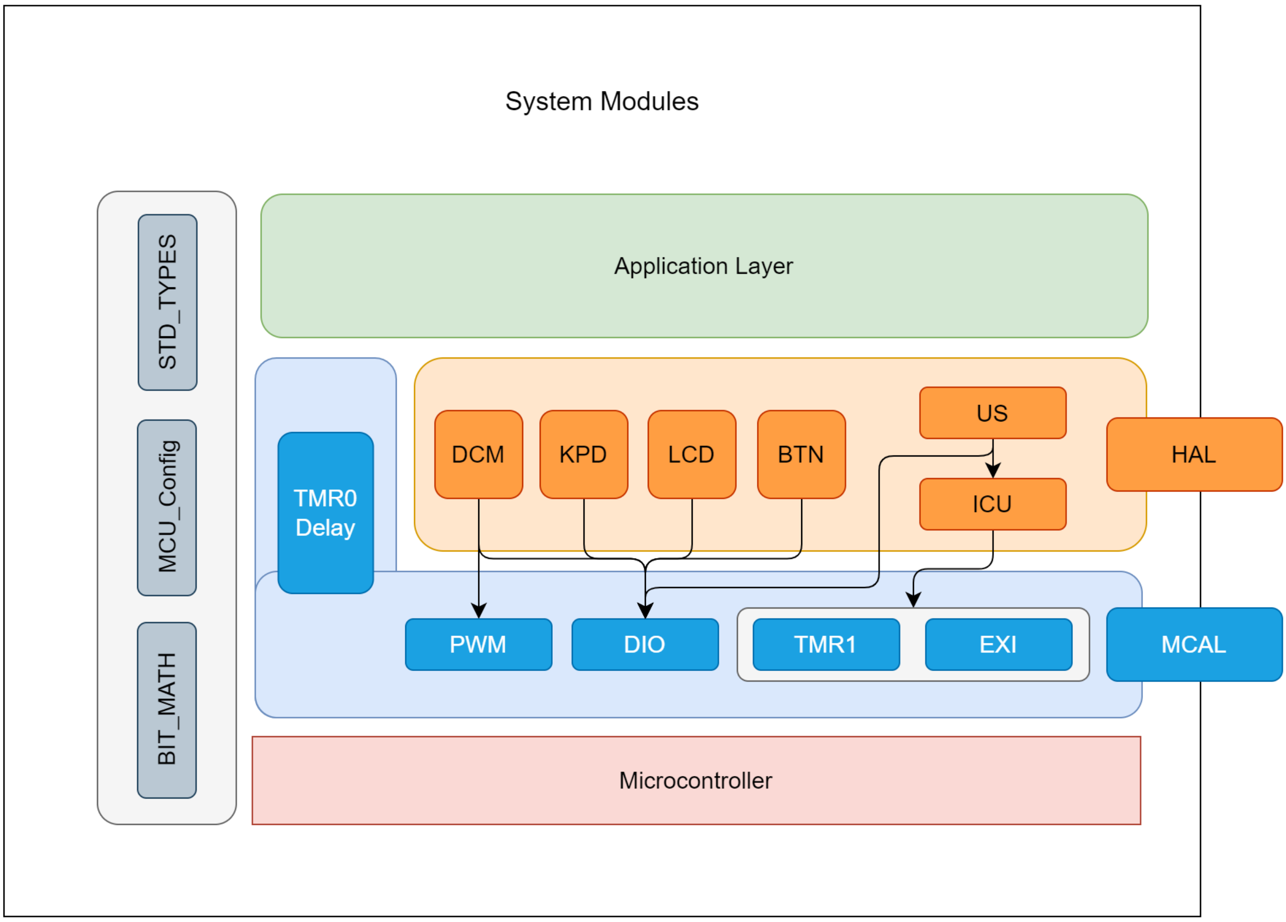


Figure 4. System Modules Design

2.4. Modules Description

2.4.1. DIO Module

The *DIO* (Digital Input/Output) module is responsible for reading input signals from the system's sensors (such as buttons) and driving output signals to the system's actuators (such as *LEDs*). It provides a set of APIs to configure the direction and mode of each pin (input/output, pull-up/down resistor), read the state of an input pin, and set the state of an output pin.

2.4.2. EXI Module

The *EXI* (External Interrupt) module is responsible for detecting external events that require immediate attention from the microcontroller, such as a button press. It provides a set of APIs to enable/disable external interrupts for specific pins, set the interrupt trigger edge (rising/falling/both), and define an interrupt service routine (*ISR*) that will be executed when the interrupt is triggered.

2.4.3. TMR Module

The *TMR* (Timer) module is responsible for generating timing events that are used by other modules in the system. It provides a set of APIs to configure the timer clock source and prescaler, set the timer mode (count up/down), set the timer period, enable/disable timer interrupts, and define an ISR that will be executed when the timer event occurs.

2.4.4. PWM Module

PWM (Pulse Width Modulation) is a technique by which the width of a pulse is varied while keeping the frequency constant. For example, controlling DC motor speed, the more the Pulse width the more the speed. Also, there are applications like controlling light intensity by PWM. A period of a pulse consists of an ON cycle (5V) and an OFF cycle (0V). The fraction for which the signal is ON over a period is known as the duty cycle.

2.4.5. BTN Module

In most of the embedded electronic projects you may want to use a *BTN* (Push Button) switch to give user inputs to the microcontroller. Push Button is basically a small controlling device that is pressed to operate any electrical device.

2.4.6. LCD Module

As we all know *LCD* (Liquid Crystal Display) is an electronic display which is commonly used nowadays in applications such as calculators, laptops, tablets, mobile phones etc. a 16×2 character *LCD* module is a very basic module which is commonly used by electronic hobbyists and is used in many electronic devices and projects. It can display 2 lines of 16 characters and each character is displayed using a 5×7 or 5×10 pixel matrix.

2.4.7. KPD Module

KPD (Keypad) is an analog switching device which is generally available in matrix structure. It is used in many embedded system applications for allowing the user to perform a necessary task. A matrix *KPD* consists of an arrangement of switches connected in matrix format in rows and columns. The rows and columns are connected with a microcontroller such that the rows of switches are connected to one pin and the columns of switches are connected to another pin of a microcontroller.

2.4.8. DCM Module

DCM (DC Motor) converts electrical energy in the form of direct current into mechanical energy. The *DCM* can be rotated at a certain speed by applying a fixed voltage to it. If the voltage varies, the speed of the motor varies. Thus, the *DCM* speed can be controlled by applying varying DC voltage; whereas the direction of rotation of the motor can be changed by reversing the direction of current through it.

2.4.9. ICU Module

Input capture is a method of dealing with input signals in an embedded system. Embedded systems using input capture will record a timestamp in memory when an input signal is received.In the microcontroller world, the *ICU* (Input Capture Unit) may be a stand alone peripheral or a mode of a timer or it may not exist. If the *ICU* peripheral does not exist in the microcontroller, we still can implement the functionality using *EXI* (External Interrupt) Peripheral and Normal *TMR* (Timer).

2.4.10. US Module

US (Ultrasonic) Module HC-SR04 works on the principle of SONAR and RADAR systems. HC-SR-04 module has a US transmitter, receiver, and control circuit on a single board. The module has only 4 pins, Vcc, Gnd, Trig, and Echo. When a pulse of 10µsec or more is given to the Trig pin, 8 pulses of 40 kHz are generated. After this, the Echo pin is made high by the control circuit in the module. The echo pin remains high till it gets an echo signal of the transmitted pulses back. The time for which the echo pin remains high, i.e. the width of the Echo pin gives the time taken for generated ultrasonic sound to travel towards the object and return. Using this time and the speed of sound in air, we can find the distance of the object using a simple formula for distance using speed and time.

2.5. Drivers’ Documentation (APIs)

2.5.1 Definition

An *API* is an *Application Programming Interface* that defines a set of *routines*, *protocols* and *tools* for creating an application. An *API* defines the high level interface of the behavior and capabilities of the component and its inputs and outputs.

An *API* should be created so that it is generic and implementation independent. This allows for the API to be used in multiple applications with changes only to the implementation of the API and not the general interface or behavior.

2.5.2. MCAL APIs

2.5.2.1. DIO Driver APIs

```
| Name: DIO_init
| Input: en PortNumber, en PinNumber, and en PinDirection
| Output: void
| Description: Function to initialize Pin direction.
|
void DIO_init (EN_DIO_PortNumber en_a_portNumber, EN_DIO_PINNumber
en_a_pinNumber, EN_DIO_PINDirection en_a_pinDirection)

| Name: DIO_write
| Input: en PortNumber, en PinNumber, and en PinValue
| Output: void
| Description: Function to set Pin value.
|
void DIO_write (EN_DIO_PortNumber en_a_portNumber, EN_DIO_PINNumber
en_a_pinNumber, EN_DIO_PINValue en_a_pinValue)

| Name: DIO_read
| Input: en PortNumber, en PinNumber, and Pointer to u8 ReturnedData
| Output: void
| Description: Function to get Pin value.
|
void DIO_read (EN_DIO_PortNumber en_a_portNumber, EN_DIO_PINNumber
en_a_pinNumber, u8 *pu8_a_returnedData)
```

```
| Name: DIO_toggle
| Input: en portNumber and en PinNumber
| Output: void
| Description: Function to toggle Pin value.
|
void DIO_toggle (EN_DIO_PortNumber en_a_portNumber, EN_DIO_PINNumber
en_a_pinNumber)

| Name: DIO_setPortDirection
| Input: en PortNumber and en PortDirection
| Output: void
| Description: Function to set Port direction.
|
void DIO_setPortDirection (EN_DIO_PortNumber en_a_portNumber, u8
u8_a_portDirection)

| Name: DIO_setPortValue
| Input: en PortNumber and u8 PortValue
| Output: void
| Description: Function to set Port value.
|
void DIO_setPortValue (EN_DIO_PortNumber en_a_portNumber, u8 u8_a_portValue)

| Name: DIO_getPortValue
| Input: en PortNumber and Pointer to u8 ReturnedPortValue
| Output: void
| Description: Function to get Port value.
|
void DIO_getPortValue (EN_DIO_PortNumber en_a_portNumber, u8
*pu8_a_returnedPortValue)

| Name: DIO_setHigherNibble
| Input: en portNumber and u8 Data
| Output: void
| Description: Function to set Higher Nibble of Port.
|
void DIO_setHigherNibble (EN_DIO_PortNumber en_a_portNumber, u8 u8_a_data)

| Name: DIO_setLowerNibble
| Input: en PortNumber and u8 Data
| Output: void
| Description: Function to set LOWER Nibble of Port.
|
void DIO_setLowerNibble (EN_DIO_PortNumber en_a_portNumber, u8 u8_a_data)
```

2.5.2.2. EXI Driver APIs

```
Precompile and Linking Configurations Snippet:

There is no need for Precompile Configurations nor Linking Configurations as
the defined APIs below could change the EXI peripheral Configurations during
the Runtime.

| Name: EXI_enablePIE
| Input: u8 InterruptId and u8 SenseControl
| Output: u8 Error or No Error
| Description: Function to enable and configure Peripheral Interrupt Enable (PIE),
|              by setting relevant bit for each interrupt in GICR register, then
|              configuring Sense Control in MCUCR (case interrupt 0 or 1) or MCUCSR
|              (case interrupt 2) registers.
|
u8 EXI_enablePIE (u8 u8_a_interruptId, u8 u8_a_senseControl)

| Name: EXI_disablePIE
| Input: u8 InterruptId
| Output: u8 Error or No Error
| Description: Function to disable Peripheral Interrupt Enable (PIE), by clearing
|              relevant bits for each interrupt in the GICR register.
|
u8 EXI_disablePIE (u8 u8_a_interruptId)

| Name: EXI_intSetCallBack
| Input: u8 InterruptId and Pointer to function INTInterruptAction that takes void and
|        returns void
| Output: u8 Error or No Error
| Description: Function to receive an address of a function (in APP Layer) to be
|              called back in ISR function of the passed Interrupt (InterruptId), the
|              address is passed through a pointer to function (INTInterruptAction),
|              and then pass this address to the ISR function.
|
u8 EXI_intSetCallBack (u8 u8_a_interruptId,
void(*pf_a_intInterruptAction)(void))
```

2.5.2.3. TMR Driver APIs

```
TMR0_APIs

| To configure maximum timeout re_define MAX_DELAY_COMP_MODE
| For example: Assume OCR register (0--255)
|              max_timeout = MAX_DELAY_COMP_MODE * OCR
|
#define MAX_DELAY_COMP_MODE 40  //evaluate to 40*250ms=10 seconds

| Name: TMR0_DelayMS
| Input: f32 f32_a_delay
| Output: void
| Description: Function to apply block delay in (ms) using overflow mode .
|
void TMR0_delayMS (f32 f32_a_delay)

| Name: TMR0_Stop
| Input: void
| Output: void
| Description: Function to stop timer operation.
|
void TMR0_stop (void)

| Name: TMR0_CallEvent
| Input: f32 delay and Pointer to function take void and return void
| Output: void
| Description: Function to call event after elapse timeout delay using overflow mode
|              minimum delay is (1 ms)
|
void TMR0_callEvent (f32 f32_a_delay, void(*g_ptr)(void))

| Name: TMR0_TimeOutMs
| Input: f32 delay
| Output: void
| Description: use to apply timeout function using compare match mode, to check that
|              timeout use <g_timeout_flag> which by default is zero and change to 1
|              after timeout
|
| [maximum timeout] is 10.2 second
| to configure maximum timeout redefine MAX_DELAY_COMP_MODE
| for example:
| assume OCR register (0--255) max_timeout = MAX_DELAY_COMP_MODE * OCR
|
void TMR0_timeOutMs (f32 f32_a_delay)
```

TMR1_APIs

```
typedef enum{
      OVF=4,COMP_B=8,COMP_A=16,INPUT_CAPT=32,MAX_SOURCE
}EN_TMR_INT;

typedef enum{
      Disable,Enable
}EN_INT_STATE;

typedef enum{
NO_CLK,CLK_1,CLK_8,CLK_64,CLK_256,CLK_1024,EXT_CLK_FALLING,EXT_CLK_RISING,
MAX_CLK
}EN_TMR_CLK;

typedef enum{
      TMR_FALLING,TMR_RISING
}EN_TME_CAPT_EDGE;

typedef enum{
      Noise_Disable,Noise_Enable
}EN_TMR_CAPT_FILTER;

typedef enum{
      Normal_mode,CMP_toggle,CMP_clear,CMP_set
}EN_TME_CMP_MODE;

typedef enum{
      Normal,PWM,CTC=4,FAST_PWM,MAX_MODE=15
}EN_TMR_MODE;

typedef struct{
      EN_TMR_MODE TMR_mode;
      EN_TMR_CLK CLK_source;
      EN_INT_STATE INT_state;
      EN_TMR_INT INT_source;
      EN_TME_CAPT_EDGE Edge_type;
      EN_TME_CMP_MODE CMP_mode;
      EN_TMR_CAPT_FILTER NO_Noise;
}ST_TME1_ConfigType;
```

```
| Name: TMR1_init
| Input: Pointer to st TMRConfig
| Output: void
| Description: Function to initialize timer with the passed configurations.
|
void TMR1_init (ST_TME1_ConfigType *pst_a_tmrConfig)

| Name: TMR1_readTime
| Input: void
| Output: void
| Description: Function to read value from timer counter.
|
u16 TMR1_readTime (void)

| Name: TMR1_clear
| Input: void
| Output: void
| Description: Function to clear timer counter.
|
void TMR1_clear (void)

| Name: TMR1_stop
| Input: void
| Output: void
| Description: Function to stop timer operation.
|
void TMR1_stop (void)
```

2.5.2.4. PWM Driver APIs

```
| Name: PWM_initialization
| Input: u8 PortId, u8 PinId, and f32 PWMFrequency
| Output: u8 Error or No Error
| Description: Function to Initialize PWM peripheral.
|
u8 PWM_initialization (u8 u8_a_portId, u8 u8_a_pinId, f32 f32_a_pwmFrequency)

| Name: PWM_generatePWM
| Input: u8 DutyCycle
| Output: u8 Error or No Error
| Description: Function to Generate PWM.
|
u8 PWM_generatePWM (u8 u8_a_dutyCycle)

| Name: PWM_calculatePrescaler
| Input: f32 Delay and Pointer to u16 ReturnedPrescaler
| Output: u8 Error or No Error
| Description: Function to calculate Prescaler value.
|
static u8 PWM_calculatePrescaler (f32 f32_a_delay, u16
*pu16_a_returnedPrescaler)

| Name: PWM_calculateInitialValue
| Input: u16 Prescaler, f32 Delay, and Pointer to u16 ReturnedPrescaler
| Output: u8 Error or No Error
| Description: Function to calculate Initial value.
|
static u8 PWM_calculateInitialValue (u16 u16_a_prescaler, f32 f32_a_delay, u8
*pu16_a_returnedInitialValue)

| Name: PWM_setPrescaler
| Input: u16 Prescaler
| Output: u8 Error or No Error
| Description: Function to Set Prescaler value.
|
static u8 PWM_setPrescaler (u16 u16_a_prescaler)
```

2.5.3. HAL APIs

2.5.3.1. BTN Driver APIs

```
| Name: BTN_init
| Input: en PortNumber and en PinNumber
| Output: void
| Description: Function to initialize BTN pin as INPUT.
|
void BTN_init (EN_DIO_PortNumber en_a_portNumber, EN_DIO_PinNumber
en_a_pinNumber)

| Name: BTN_read
| Input: en PortNumber, en PinNumber, and Pointer to u8 ReturnedBTNState
| Output: void
| Description: Function to get BTN state.
|
void BTN_read (EN_DIO_PortNumber en_a_portNumber, EN_DIO_PinNumber
en_a_pinNumber, u8 *pu8_a_returnedBTNState)
```

2.5.3.2. LCD Driver APIs

```
| Name: LCD_init
| Input: void
| Output: void
| Description: Function to initialize (both 4 Bit and 8 Bit Modes) LCD.
|
void LCD_init (void)

| Name: LCD_sendcommand
| Input: u8 Cmnd
| Output: void
| Description: Function to send a Command to LCD through Data pins.
|
void LCD_sendCommand (u8 u8_a_cmnd)

| Name: LCD_sendCharacter
| Input: u8 Char
| Output: void
| Description: Function to send a Character to LCD through Data pins.
|
void LCD_sendCharacter (u8 u8_a_char)
```

```
| Name: LCD_clear
| Input: void
| Output: void
| Description: Function to clear LCD display screen in DDRAM.
|
void LCD_clear (void)

| Name: LCD_setCursor
| Input: u8 Row and u8 Column
| Output: void
| Description: Function to set the Address Counter (AC) of LCD to a certain location
|              in DDRAM.
|
void LCD_setCursor (u8 u8_a_row, u8 u8_a_column)

| Name: LCD_sendString
| Input: Pointer to u8 String
| Output: void
| Description: Function to send an array of characters to LCD through Data pins
|              (From CGROM to DDRAM).
|
void LCD_sendString (u8 *pu8_a_string)

| Name: LCD_floatToString
| Input: f32 FloatValue
| Output: void
| Description: Function to send a float (one decimal) number (positive or negative)
|              to LCD through Data pins (From CGROM to DDRAM).
|
void LCD_floatToString (f32 f32_a_floatValue)

| Name: LCD_createCustomCharacter
| Input: Pointer to u8 Pattern and u8 Location
| Output: void
| Description: Function to send character (stored in array -Special Char- byte
|              by byte) and store it in CGRAM, then display it on DDRAM
|              (From CGRAM to DDRAM).
|
void LCD_createCustomCharacter (u8 *pu8_a_pattern, u8 u8_a_location)
```

2.5.3.3. KPD Driver APIs

```
| Name: KPD_vdEnableKPD
| Input: void
| Output: void
| Description: Function to enable Keypad.
|
void KPD_enableKPD (void)

| Name: KPD_vdDisableKPD
| Input: void
| Output: void
| Description: Function to disable Keypad.
|
void KPD_disableKPD (void)

| Name: KPD_u8GetPressedKey
| Input: Pointer to u8 ReturnedKeyValue
| Output: u8 Error or No Error
| Description: Function to check for the pressed key.
|
u8 KPD_getPressedKey (u8 *pu8_a_returnedKeyValue)
```

2.5.3.4. DCM Driver APIs

```
| Name: DCM_initialization
| Input: Pointer to st DCMConfig
| Output: u8 Error or No Error
| Description: Function to Initialize DCM peripheral.
|
void DCM_initialization (DCM_ST_CONFIG *st_a_DCMConfig);

| Name: DCM_controlDCM
| Input: Pointer to st DCMConfig and u8 ControlMode
| Output: u8 Error or No Error
| Description: Function Control DCM with one of DCM Modes.
|
u8 DCM_controlDCM (DCM_ST_CONFIG *st_a_DCMConfig, u8 u8_a_controlMode)

| Name: DCM_controlDCMSpeed
| Input: u8 SpeedPercentage
| Output: u8 Error or No Error
| Description: Function Control DCM Speed.
|
u8 DCM_controlDCMSpeed (u8 u8_a_speedPercentage)
```

2.5.3.5. ICU Driver APIs

```
| Name: ICU_init
| Input: void
| Output: void
| Description: Function to initialize ICU by configuring EXI to detect the rising
|              edge.
|
void ICU_init (void)

| Name: ICU_setCallBack
| Input: Pointer to Function taking void and returning void
| Output: void
| Description: Function to set the required edge detection.
|
void ICU_setCallBack (void(*a_ptr)(void));

| Name: ICU_setEdgeDetectionType
| Input: const en EdgeType
| Output: void
| Description: Function to set the required edge detection.
|
void ICU_setEdgeDetectionType (const EN_ICU_EdgeType edgeType)

| Name: ICU_getInputCaptureValue
| Input: void
| Output: void
| Description: Function to get the Timer1 Value when the external interrupt is
|              capture edge.
|
u16 ICU_getInputCaptureValue (void)

| Name: ICU_clearTimerValue
| Input: void
| Output: void
| Description: Function to clear the Timer1 Value to start count from 0.
|
void ICU_clearTimerValue (void)

| Name: ICU_deInit
| Input: void
| Output: void
| Description: Function to disable the Timer1 and External interrupt to stop
|              the ICU Driver.
|
void ICU_deInit (void)
```

2.5.3.6. US Driver APIs

```
| Name: US_init
| Input: u8 TriggerPort, u8 TriggerPin, and en EchoPin
| Output: void
| Description: Function to Initialize the US peripheral.
|
void US_init (u8 u8_a_triggerPort, u8 u8_a_triggerPin, EN_ICU_Source
en_a_echoPin )

| Name: US_readDistance
| Input: void
| Output:void
| Description: Function to read the echo signal from the US and convert it to
|              distance.
|
u16 US_readDistance (void)
```

2.5.4. APP APIs

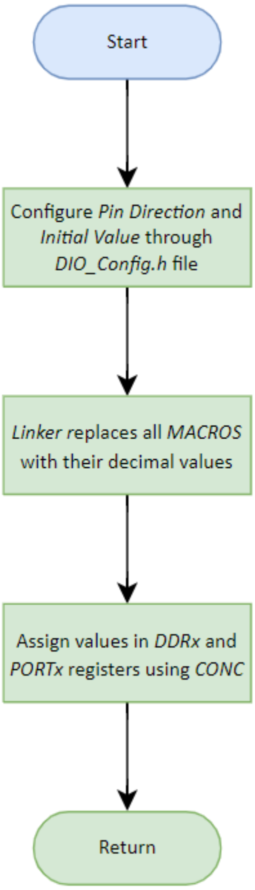
```
| Name: APP_initialization
| Input: void
| Output: void
| Description: Function to Initialize the Application.
|
void APP_initialization (void)

| Name: APP_startProgram
| Input: void
| Output: void
| Description: Function to Start the basic flow of the Application.
|
void APP_startProgram (void)

| Name: APP_stopCar
| Input: void
| Output: void
| Description: Function to Stop the car.
|
void APP_stopCar (void)
```

3. Low Level Design

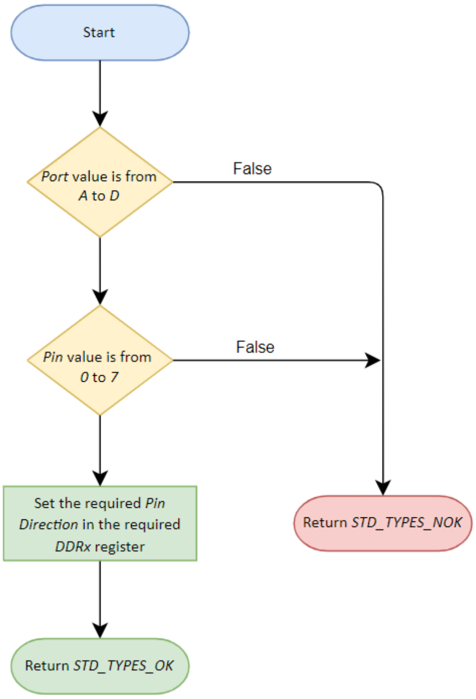
3.1. MCAL Layer

3.1.1. DIO Module

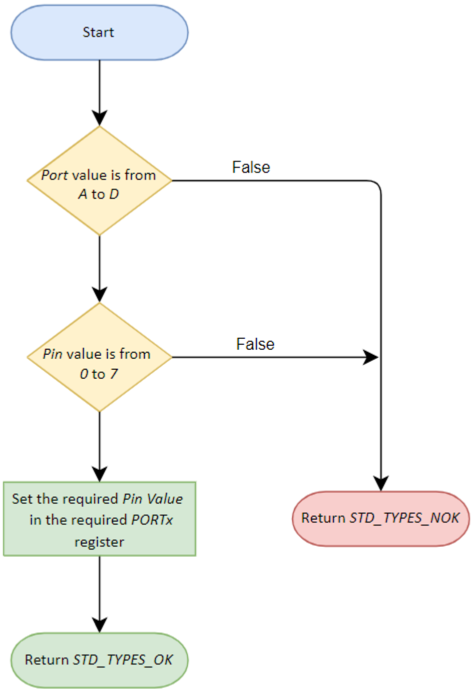
A. DIO_init


Start
Configure *Pin Direction* and *Initial Value* through *DIO_Config.h* file
Linker replaces all *MACROS* with their decimal values
Assign values in *DDRx* and *PORTx* registers using *CONC*
Return

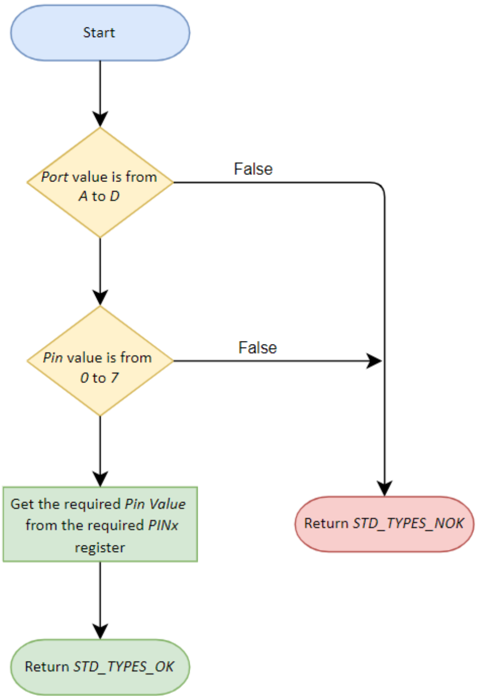
B. DIO_setPinDirection


Start
Port value is from A to D
False
Pin value is from 0 to 7
False
Set the required Pin Direction in the required DDRx register
Return STD_TYPES_NOK
Return STD_TYPES_OK

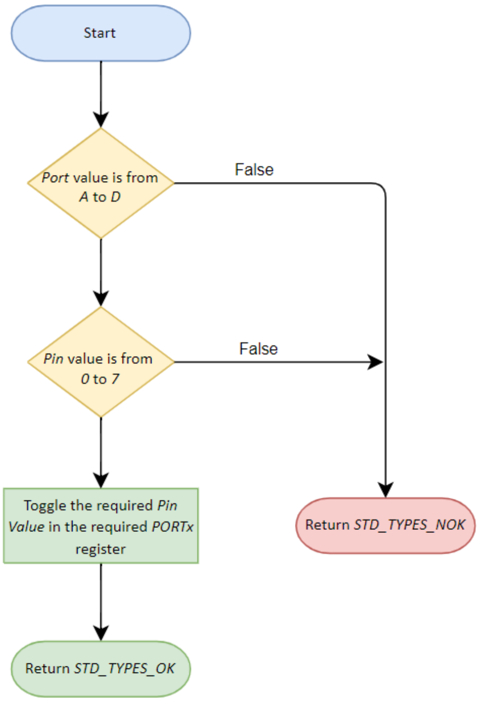
C. DIO_setPinValue


Start
Port value is from A to D
False
Pin value is from 0 to 7
False
Set the required Pin Value in the required PORTx register
Return STD_TYPES_NOK
Return STD_TYPES_OK

D. *DIO_getPinValue*

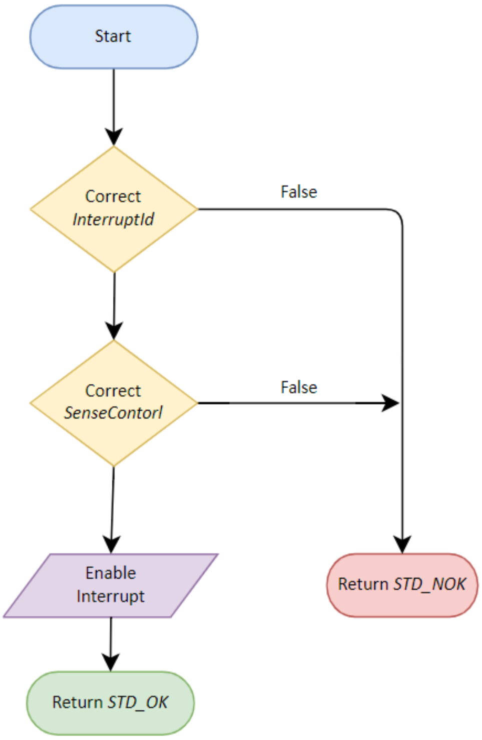

Start
Port value is from *A* to *D*
False
Pin value is from *0* to *7*
False
Get the required *Pin Value* from the required *PINx* register
Return *STD_TYPES_NOK*
Return *STD_TYPES_OK*

E. *DIO_togglePinValue*

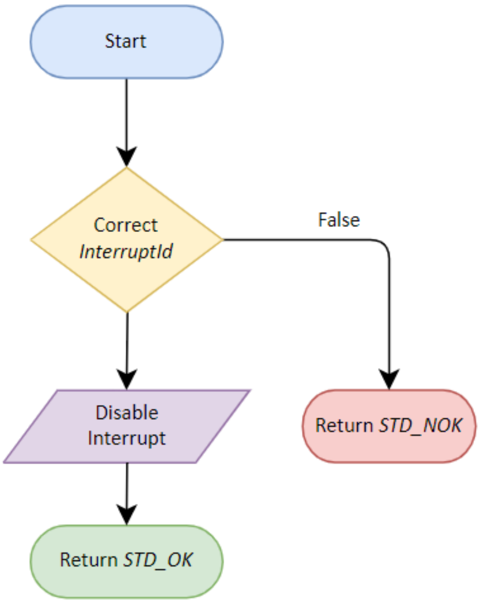

Start
Port value is from *A* to *D*
False
Pin value is from *0* to *7*
False
Toggle the required *Pin Value* in the required *PORTx* register
Return *STD_TYPES_NOK*
Return *STD_TYPES_OK*

3.1.2. EXI Module

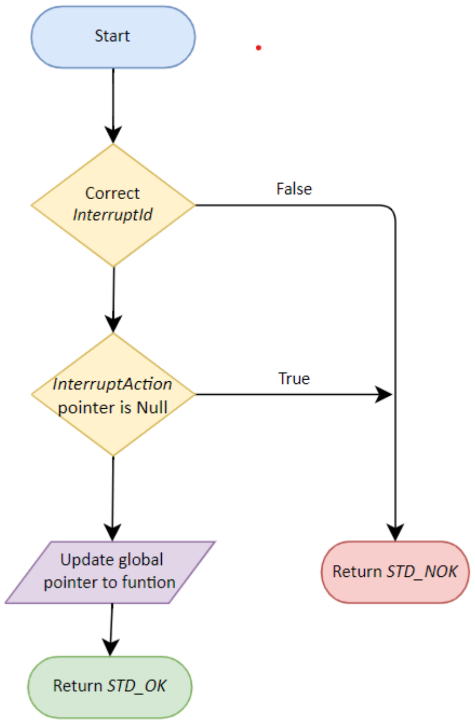
A. EXI_enablePIE



B. EXI_disablePIE

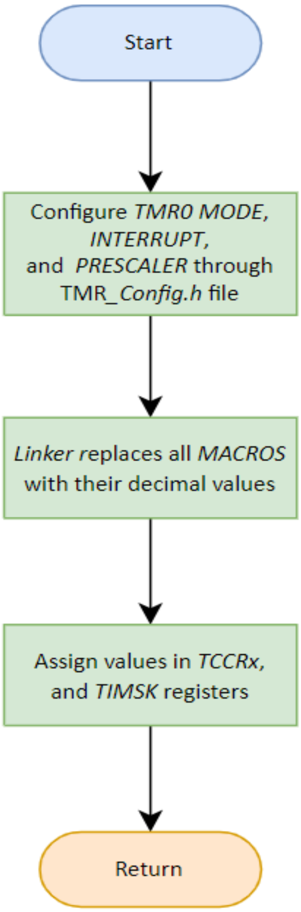


C. *EXI_intSetCallBack*

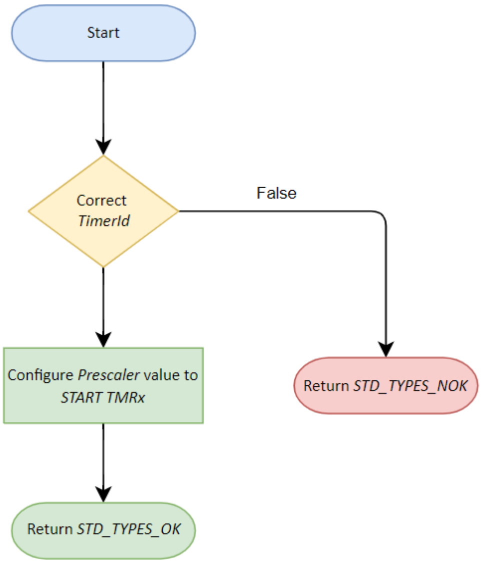


3.1.3. TMR Module

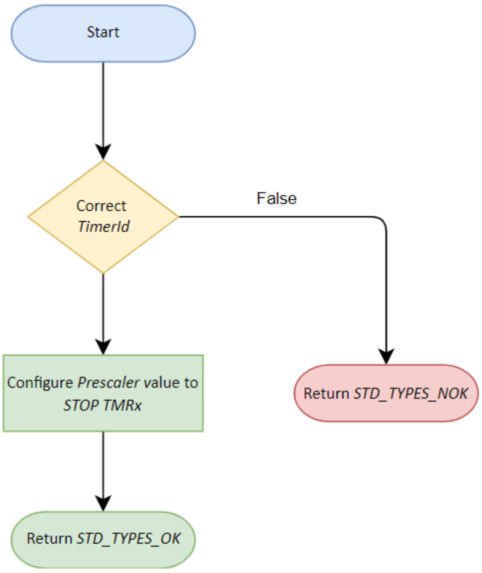
A. *TMR0_init*
B. *TMR1_init*


Start
Configure *TMR0 MODE, INTERRUPT,* and *PRESCALER* through TMR_*Config.h* file
Linker replaces all *MACROS* with their decimal values
Assign values in *TCCRx,* and *TIMSK* registers
Return

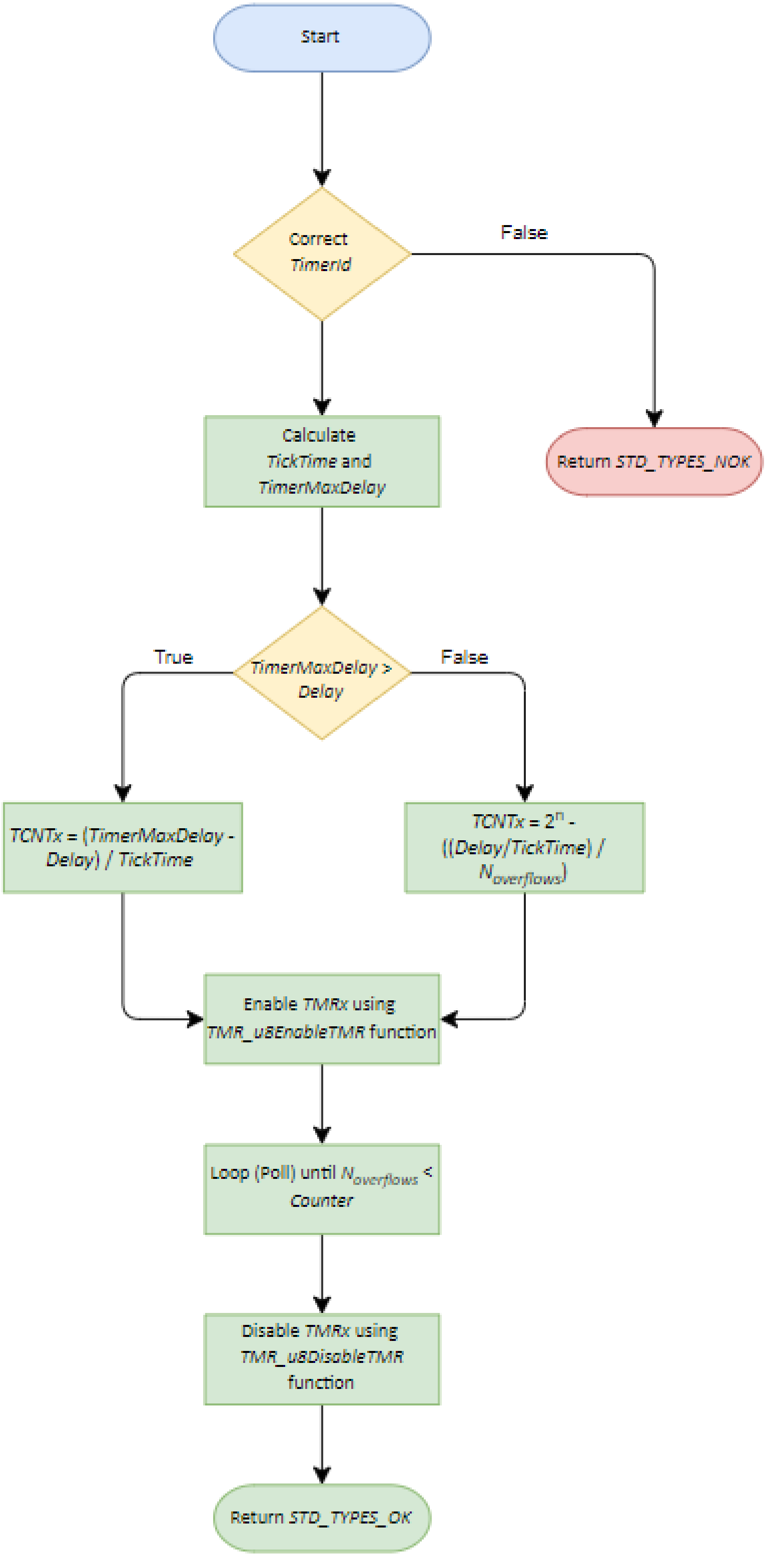
C. *TMR0_enableTMR*



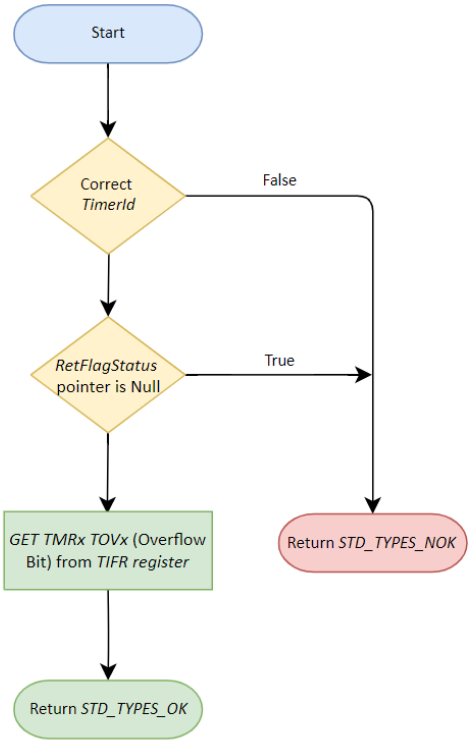
D. *TMR0_disableTMR*



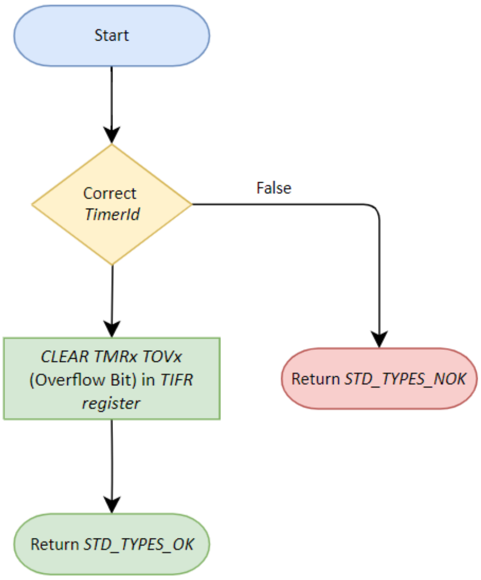
E. *TMR0_delayMS*


Start
Correct TimerId
False
Return STD_TYPES_NOK
Calculate TickTime and TimerMaxDelay
TimerMaxDelay > Delay
True
False
TCNTx = (TimerMaxDelay - Delay) / TickTime
TCNTx = 2^n - ((Delay/TickTime) / $N_{overflows}$)
Enable TMRx using TMR_u8EnableTMR function
Loop (Poll) until $N_{overflows}$ < Counter
Disable TMRx using TMR_u8DisableTMR function
Return STD_TYPES_OK

F. *TMR0_getOVFFlagStatus*



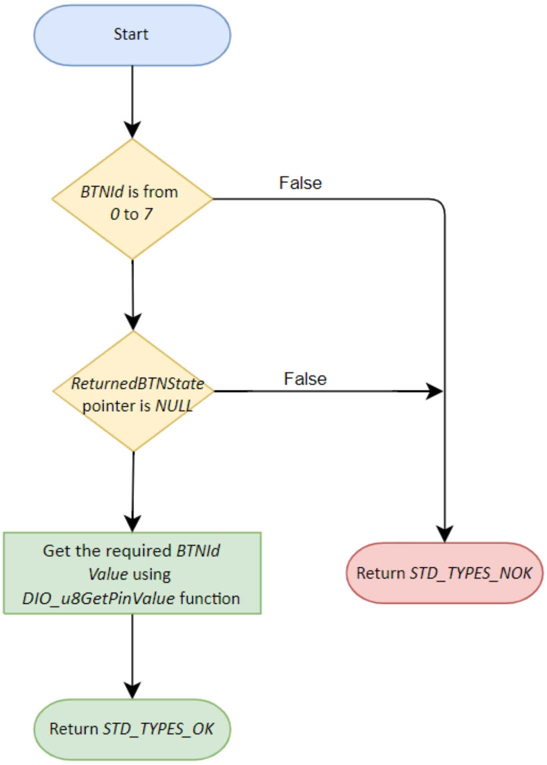
G. *TMR0_clearOVFFlag*



3.2. HAL Layer

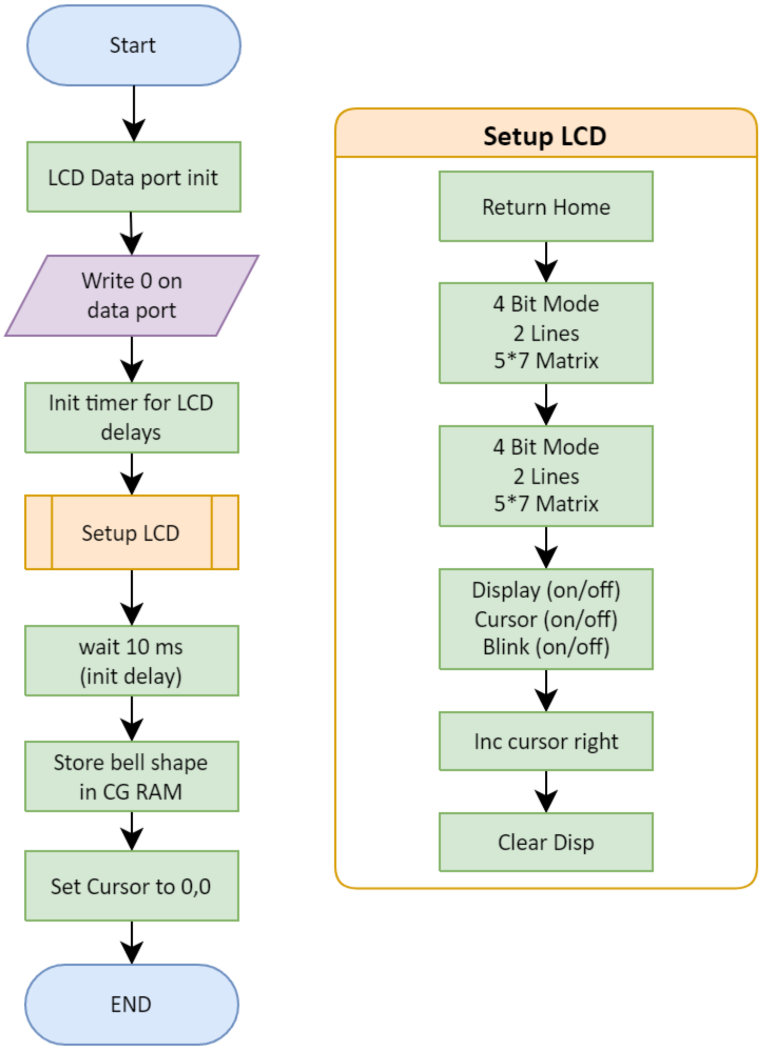
3.2.1. BTN Module

A. BTN_getBTNState

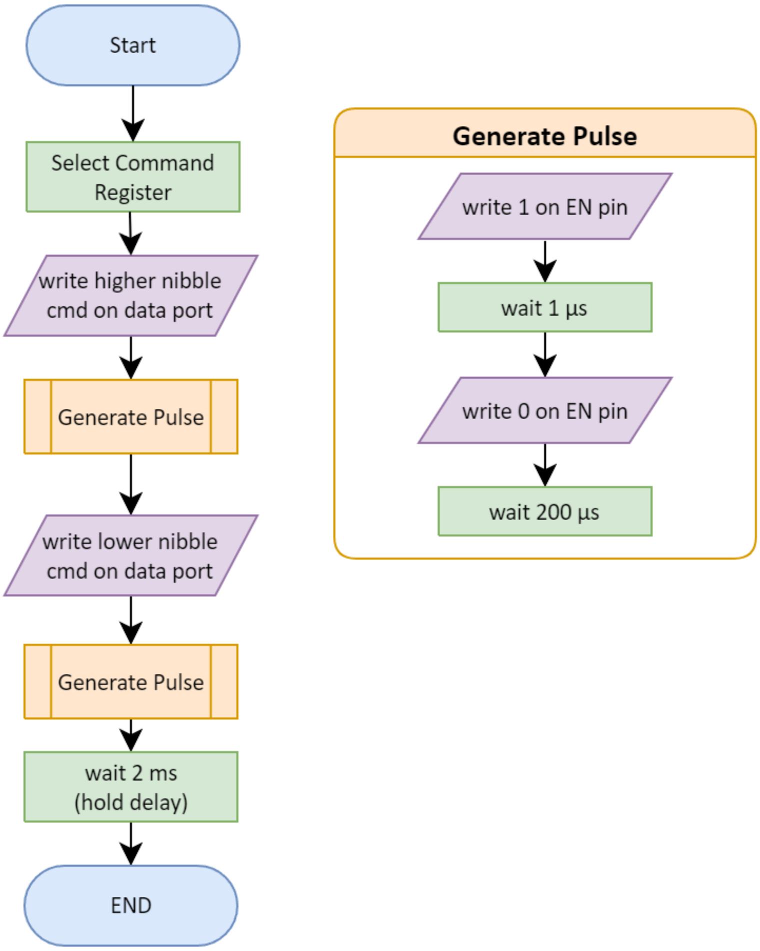

Start
BTNId is from 0 to 7
False
ReturnedBTNState pointer is NULL
False
Get the required BTNId Value using DIO_u8GetPinValue function
Return STD_TYPES_NOK
Return STD_TYPES_OK

3.2.2. LCD Module

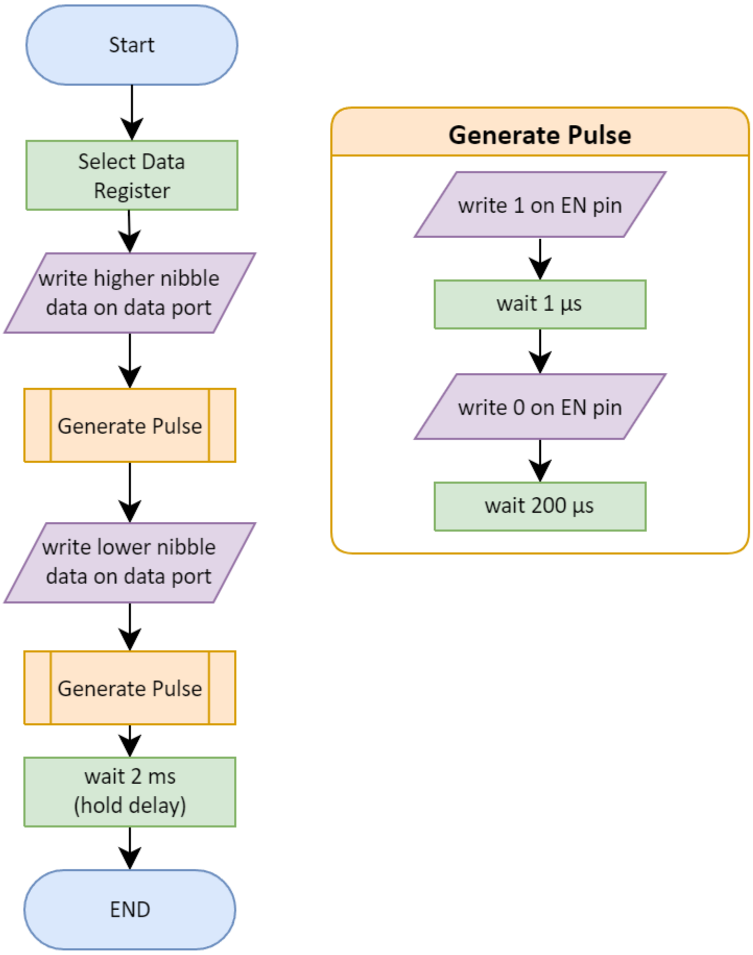
A. LCD_init



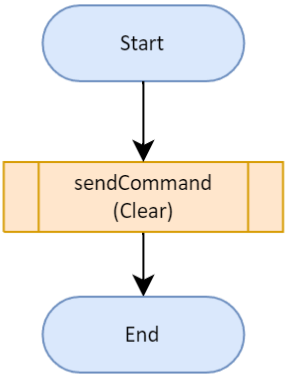
B. LCD_sendCommand



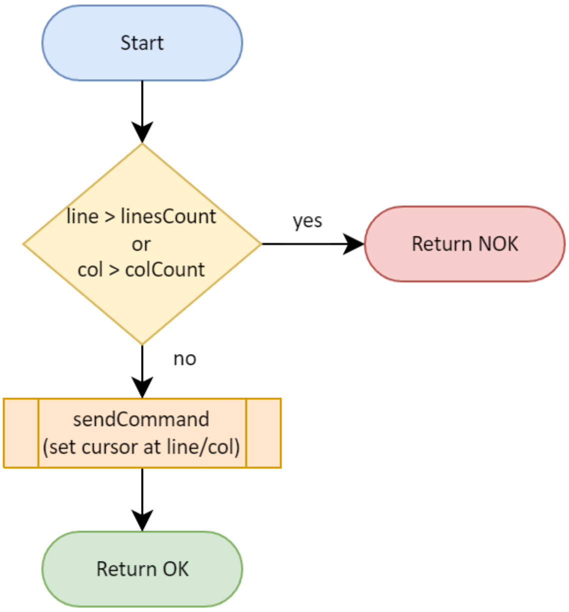
C. *LCD_sendCharacter*



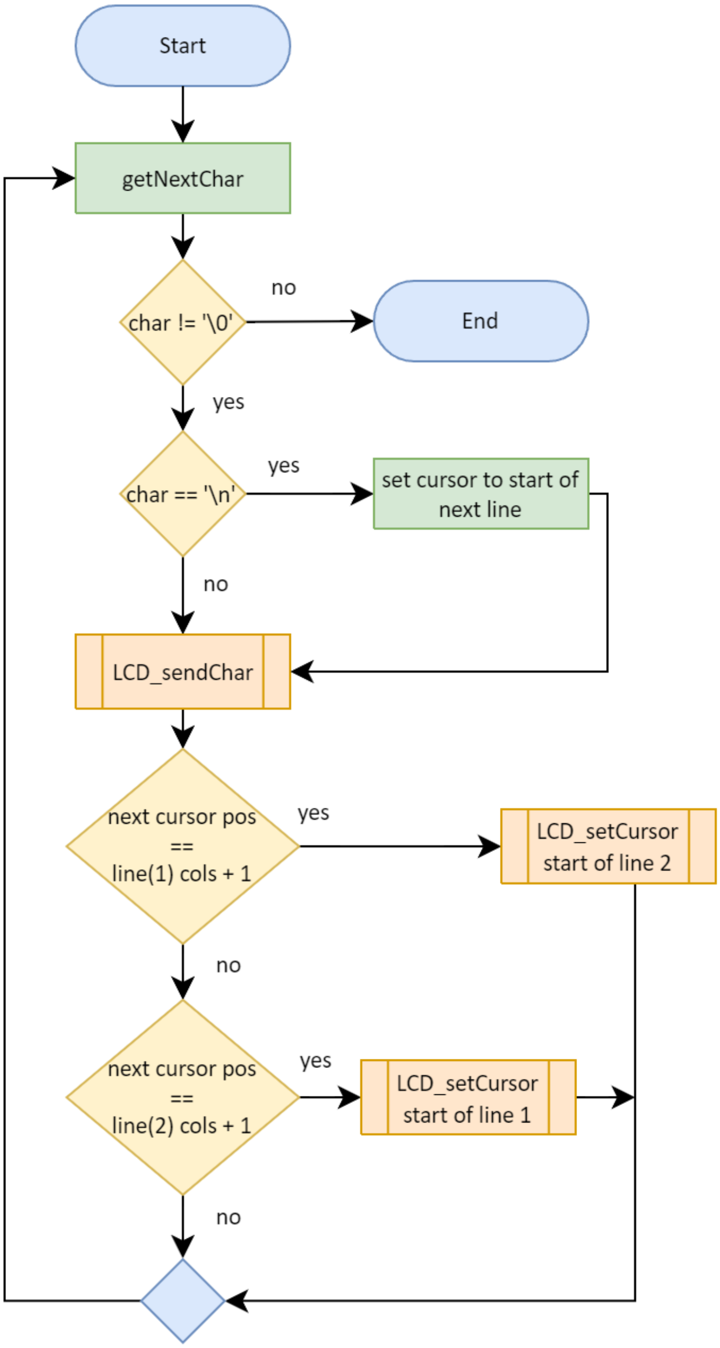
D. LCD_clear



E. LCD_goToLocation

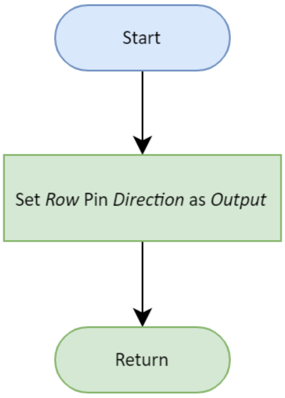


F. *LCD_writeString*

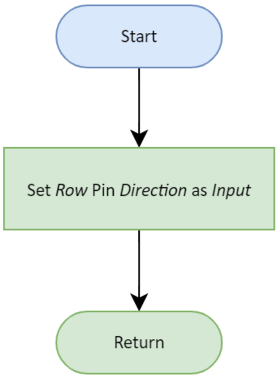

Start
getNextChar
char != '\0'
no
End
yes
char == '\n'
yes
set cursor to start of next line
no
LCD_sendChar
next cursor pos == line(1) cols + 1
yes
LCD_setCursor start of line 2
no
next cursor pos == line(2) cols + 1
yes
LCD_setCursor start of line 1
no

3.2.3. KPD Module

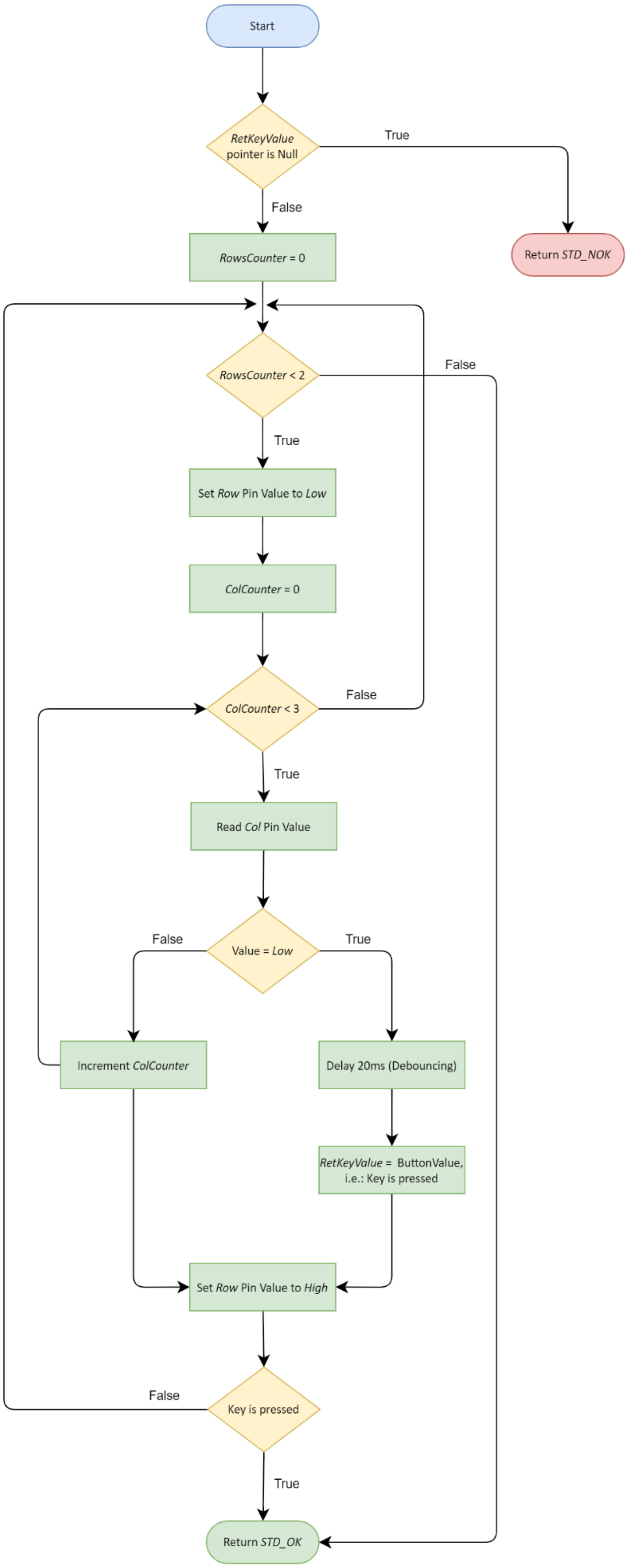
A. *KPD_enableKPD*



B. *KPD_disableKPD*

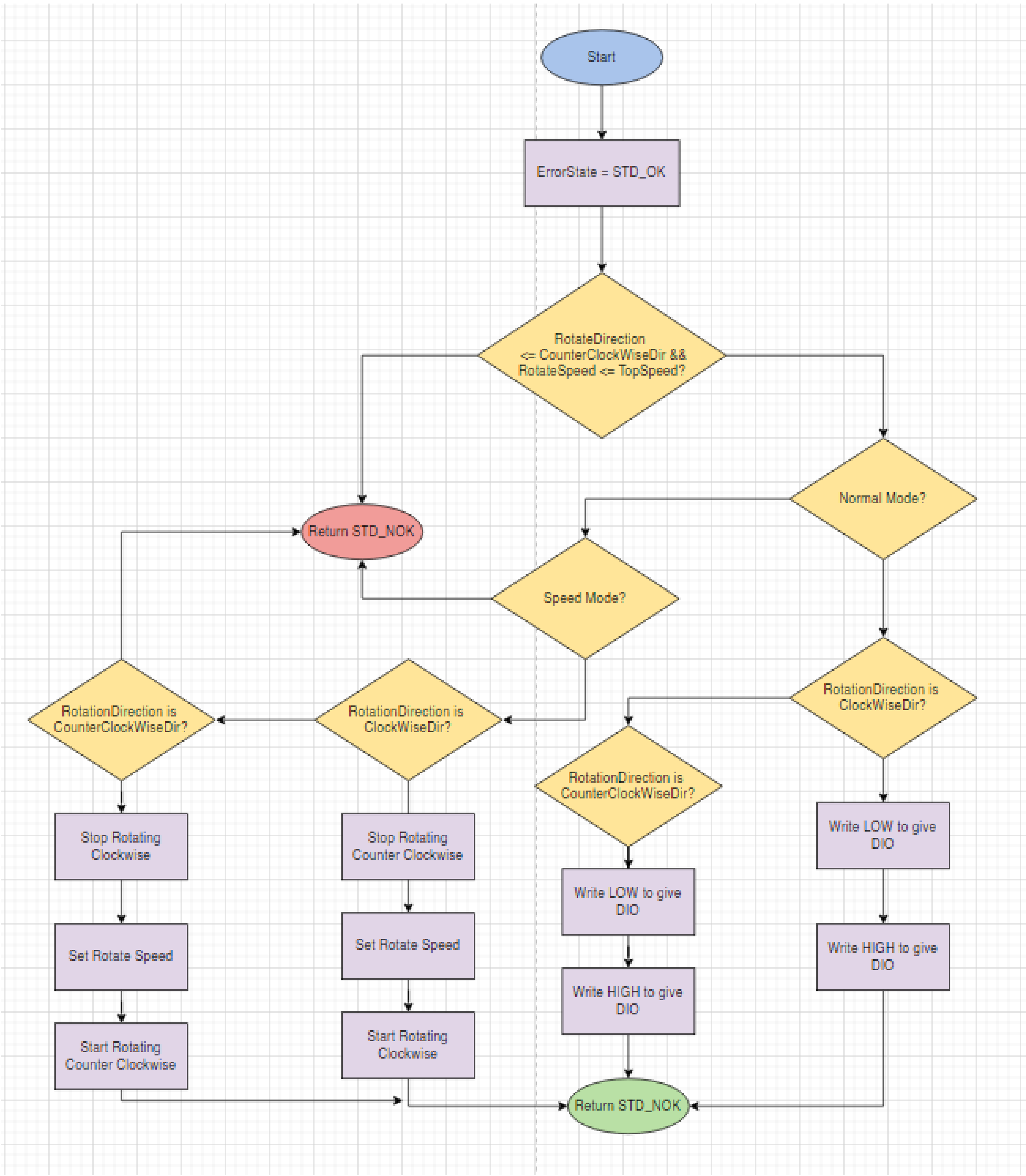


C. *KPD_getPressedKey*

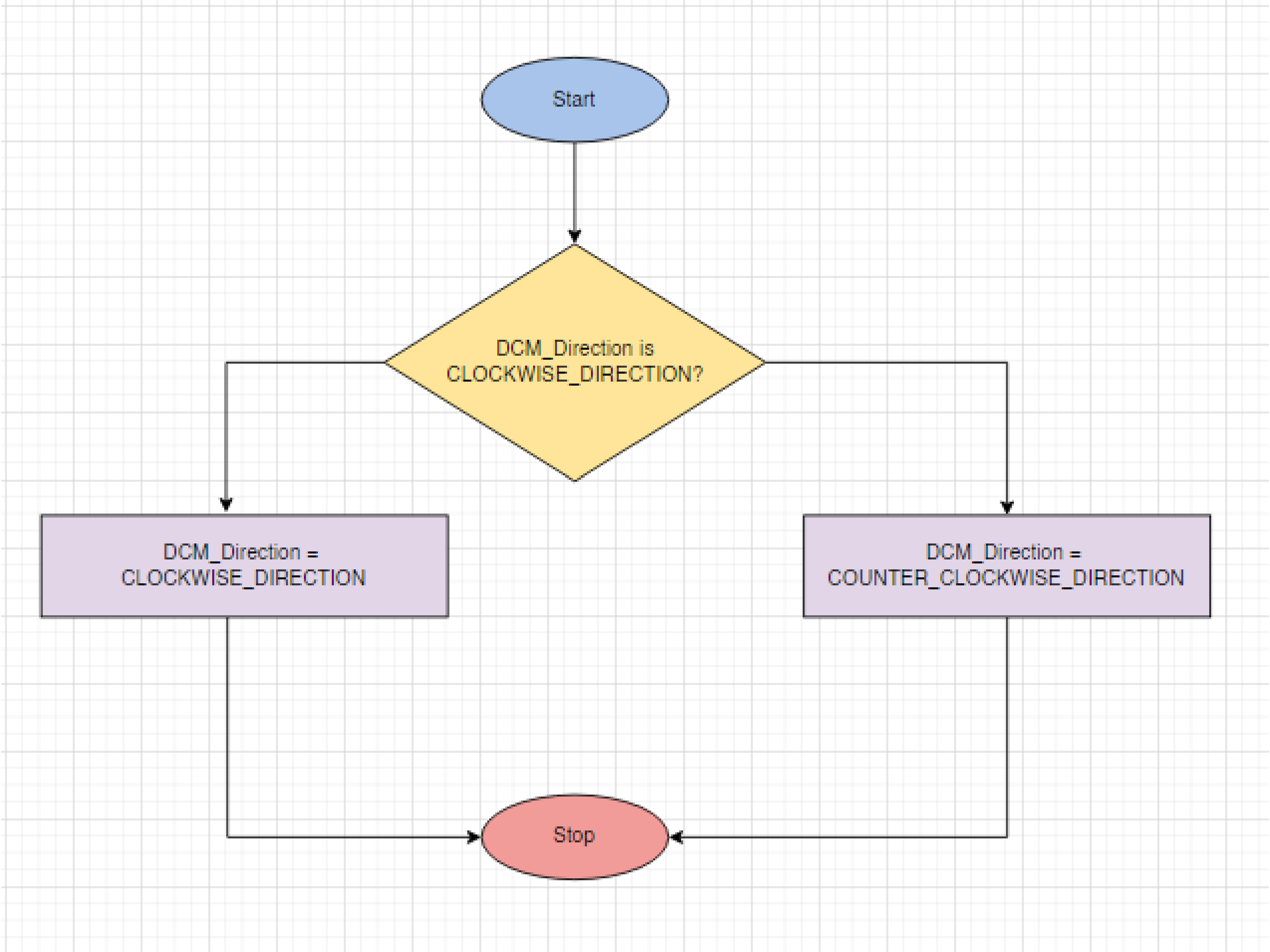


3.2.4. DCM Module

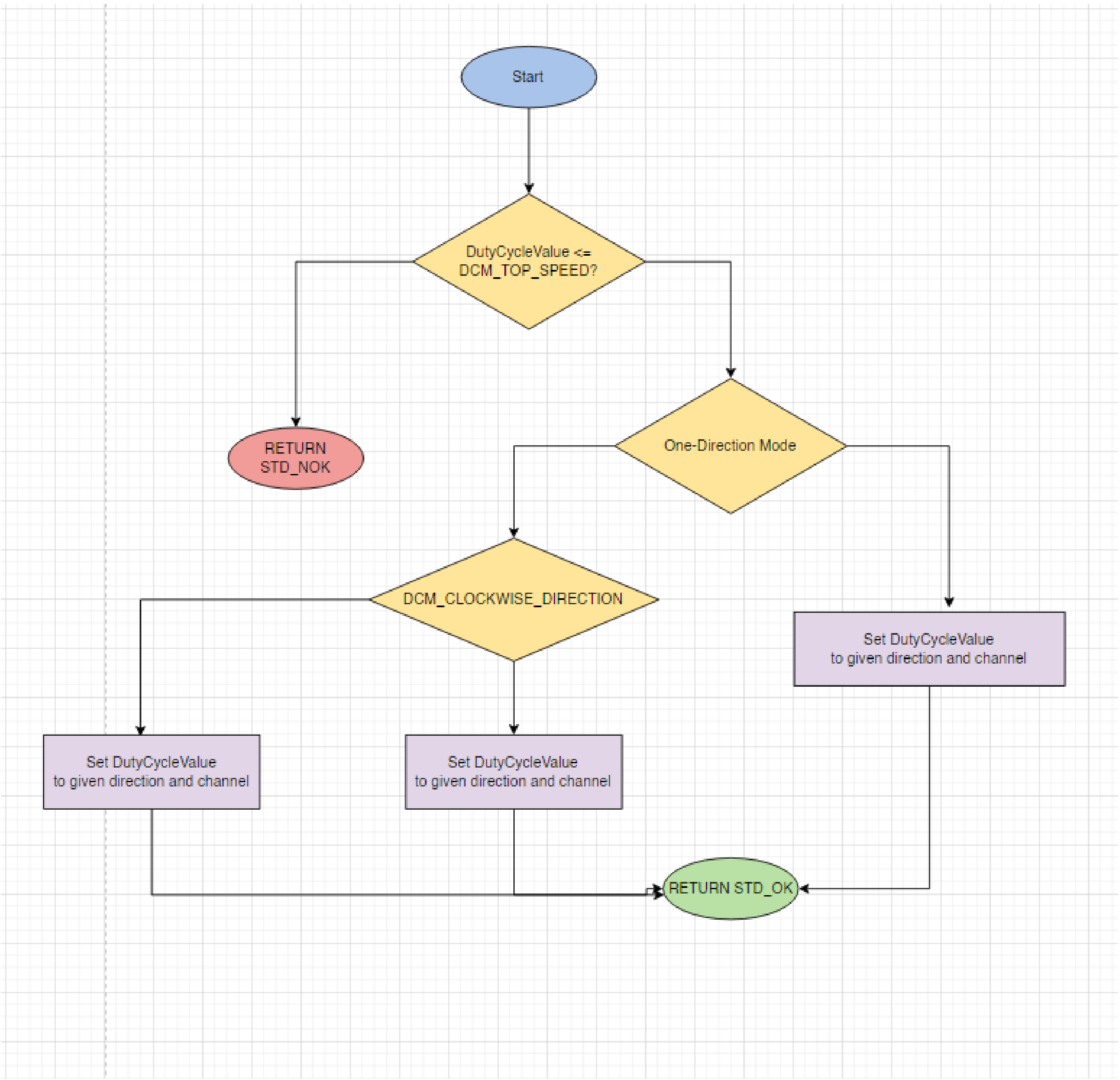
A. DCM_rotateDCM


Start
ErrorState = STD_OK
RotateDirection <= CounterClockWiseDir && RotateSpeed <= TopSpeed?
Normal Mode?
Return STD_NOK
Speed Mode?
RotationDirection is CounterClockWiseDir?
RotationDirection is ClockWiseDir?
RotationDirection is ClockWiseDir?
RotationDirection is CounterClockWiseDir?
Stop Rotating Clockwise
Stop Rotating Counter Clockwise
Write LOW to give DIO
Write LOW to give DIO
Set Rotate Speed
Set Rotate Speed
Write HIGH to give DIO
Write HIGH to give DIO
Start Rotating Counter Clockwise
Start Rotating Clockwise
Return STD_NOK

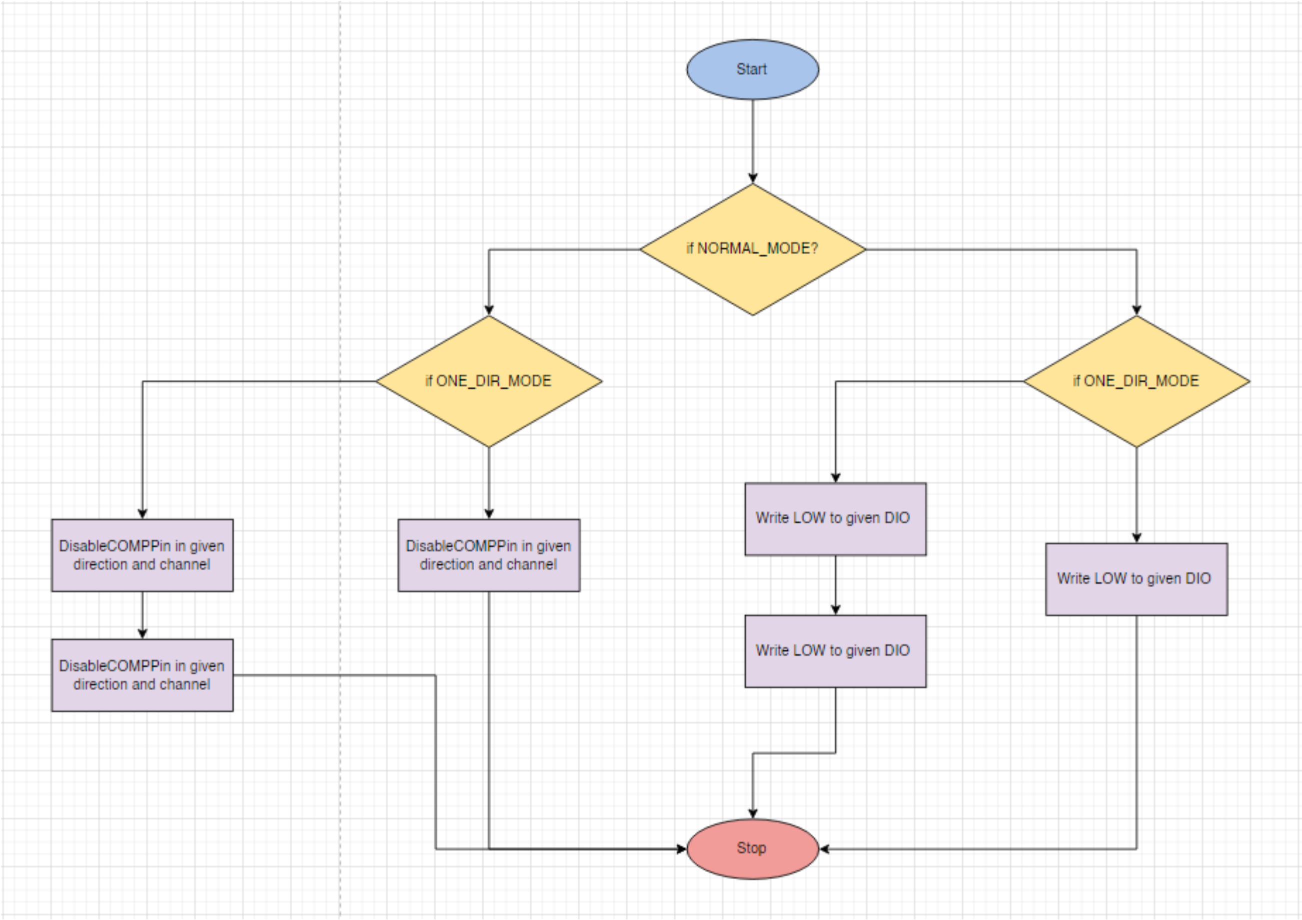
B. *DCM_changeDCMDirection*


Start
DCM_Direction is CLOCKWISE_DIRECTION?
DCM_Direction = CLOCKWISE_DIRECTION
DCM_Direction = COUNTER_CLOCKWISE_DIRECTION
Stop

C. DCM_setDutyCycleOfPWM

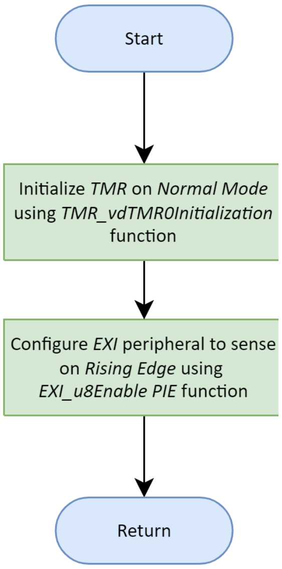

Start
DutyCycleValue <= DCM_TOP_SPEED?
RETURN STD_NOK
One-Direction Mode
DCM_CLOCKWISE_DIRECTION
Set DutyCycleValue to given direction and channel
Set DutyCycleValue to given direction and channel
Set DutyCycleValue to given direction and channel
RETURN STD_OK

D. DCM_stopDCM

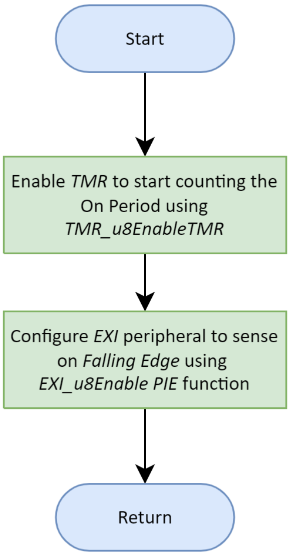

Start
if NORMAL_MODE?
if ONE_DIR_MODE
if ONE_DIR_MODE
DisableCOMPPin in given direction and channel
DisableCOMPPin in given direction and channel
DisableCOMPPin in given direction and channel
Write LOW to given DIO
Write LOW to given DIO
Write LOW to given DIO
Stop

3.2.5. ICU Module

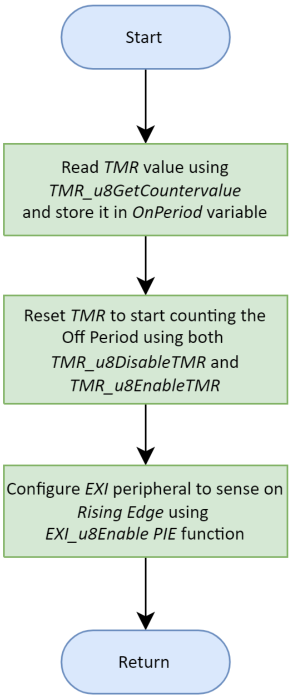
A. ICU_init


Start
Initialize *TMR* on *Normal Mode* using *TMR_vdTMR0Initialization* function
Configure *EXI* peripheral to sense on *Rising Edge* using *EXI_u8Enable PIE* function
Return

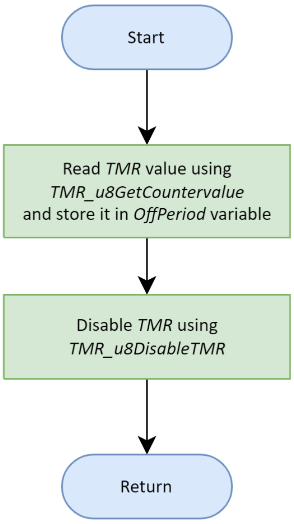
B. ICU_configureOnPeriod


Start
Enable *TMR* to start counting the On Period using *TMR_u8EnableTMR*
Configure *EXI* peripheral to sense on *Falling Edge* using *EXI_u8Enable PIE* function
Return

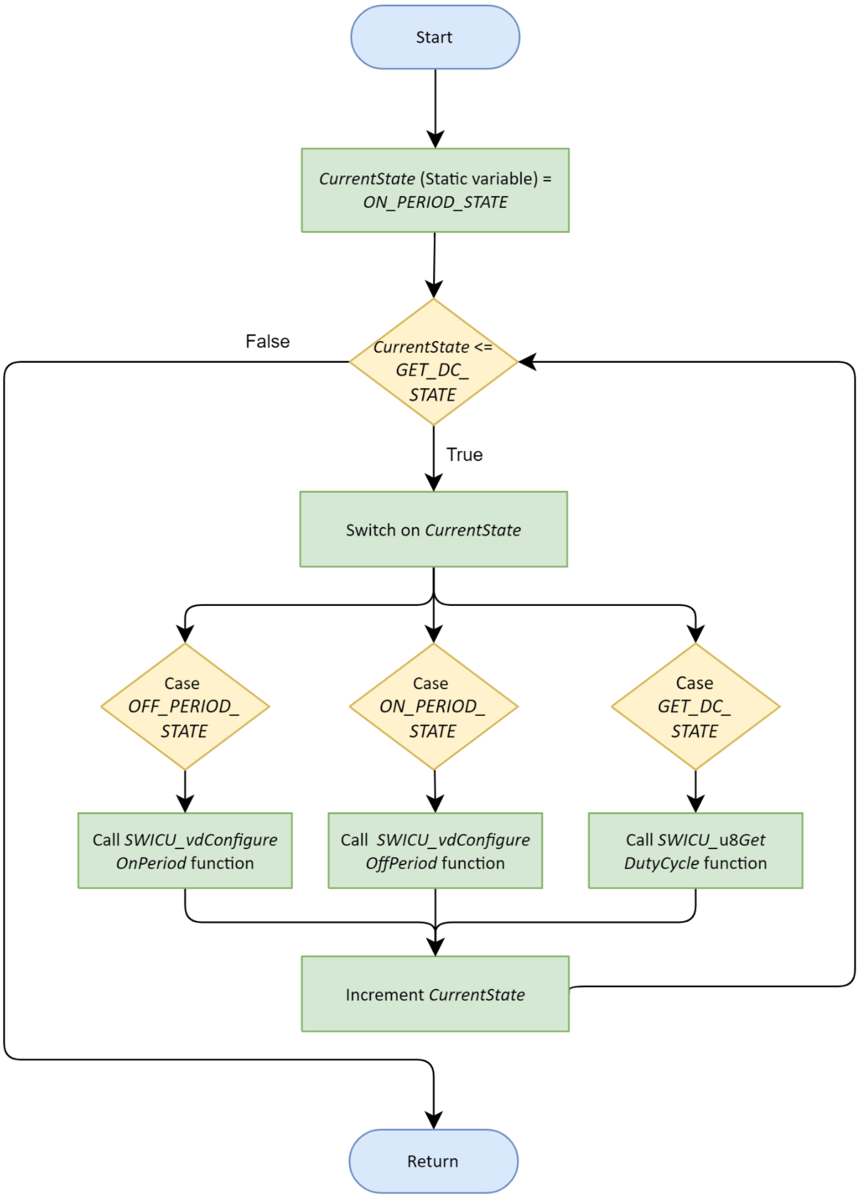
C. *ICU_configureOffPeriod*


Start
Read *TMR* value using *TMR_u8GetCountervalue* and store it in *OnPeriod* variable
Reset *TMR* to start counting the Off Period using both *TMR_u8DisableTMR* and *TMR_u8EnableTMR*
Configure *EXI* peripheral to sense on *Rising Edge* using *EXI_u8Enable PIE* function
Return

D. *ICU_getDutyCycle*

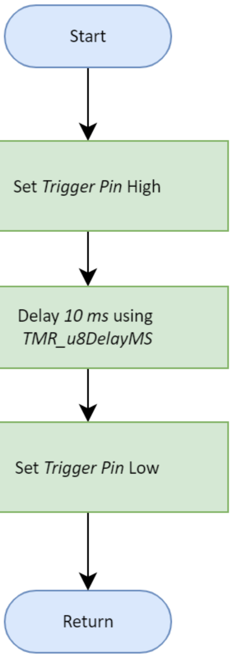

Start
Read *TMR* value using *TMR_u8GetCountervalue* and store it in *OffPeriod* variable
Disable *TMR* using *TMR_u8DisableTMR*
Return

E. ICU_switchState

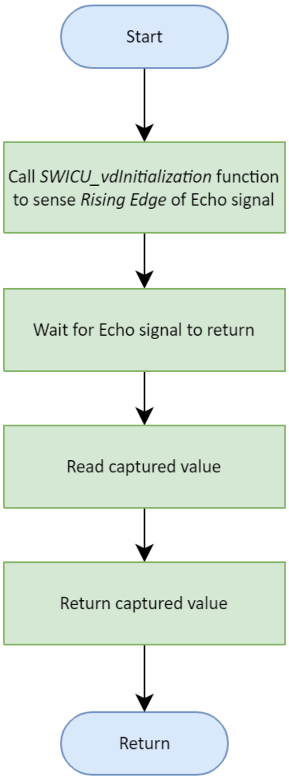

Start
CurrentState (Static variable) = *ON_PERIOD_STATE*
CurrentState <= *GET_DC_STATE*
False
True
Switch on *CurrentState*
Case *OFF_PERIOD_STATE*
Case *ON_PERIOD_STATE*
Case *GET_DC_STATE*
Call *SWICU_vdConfigure OnPeriod* function
Call *SWICU_vdConfigure OffPeriod* function
Call *SWICU_u8Get DutyCycle* function
Increment *CurrentState*
Return

3.2.6. US Module

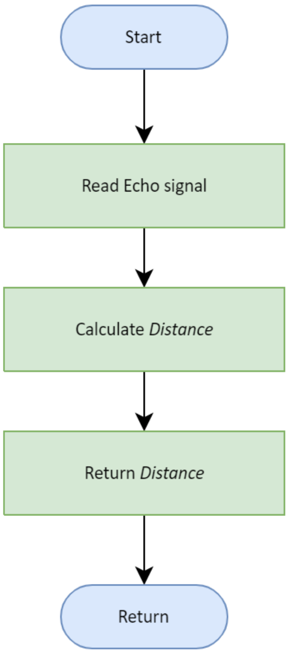
A. *US_sendTriggerPulse*



B. *USI_readEchoSignal*

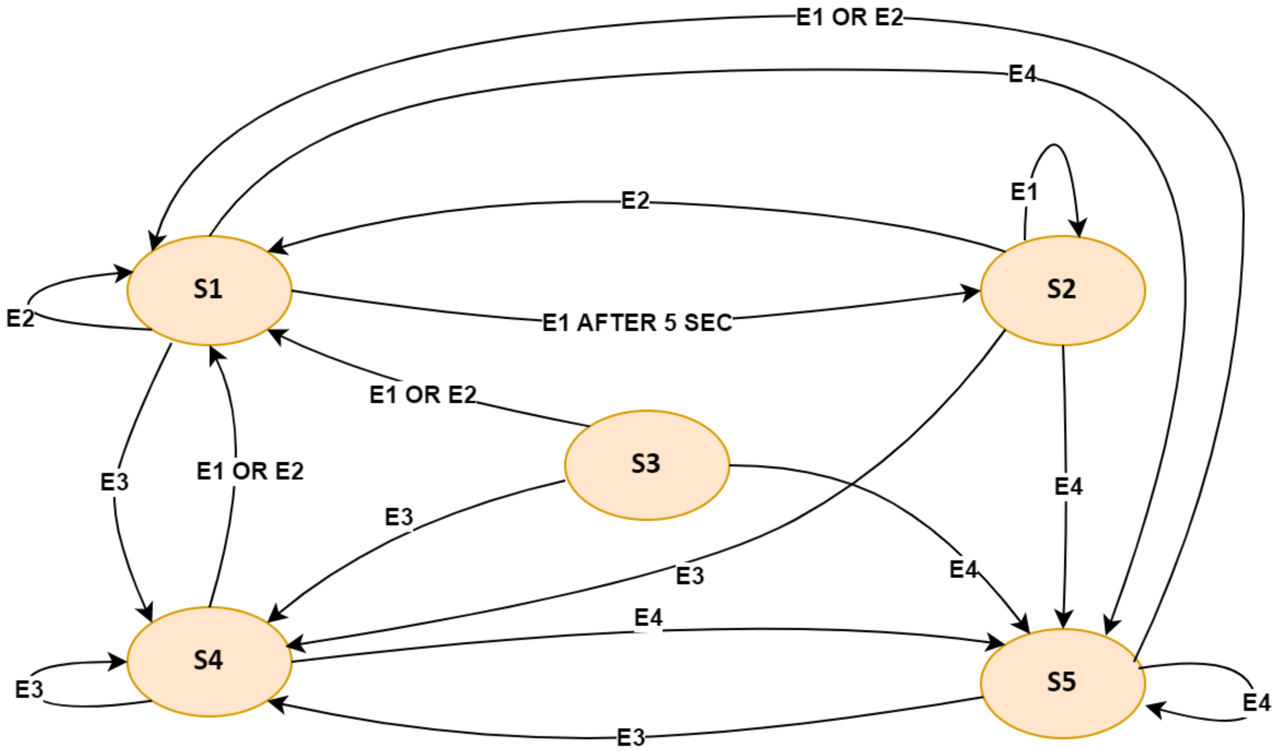

Start
Call *SWICU_vdInitialization* function to sense *Rising Edge* of Echo signal
Wait for Echo signal to return
Read captured value
Return captured value
Return

C. *US_calculateDistance*



3.3. APP Layer

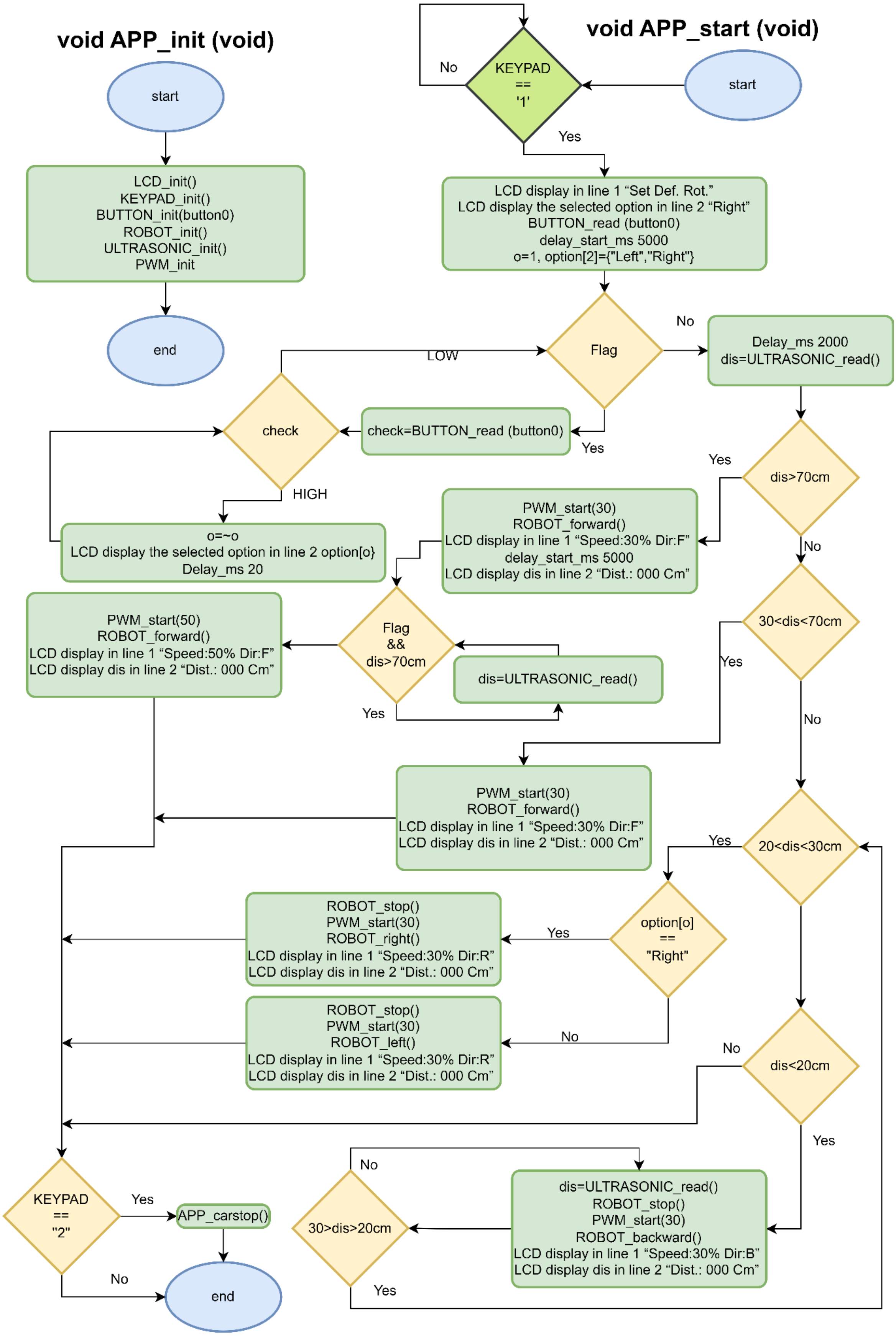
3.3.1. State Machine Diagram



STATES	
S1	CAR MOVE WITH 30%
S2	CAR MOVE WITH 50%
S3	STOP
S4	STOP & ROTATE 90*
S5	MOVE BACKWORD 30%

EVENTS	
E1	OBSTACLE AT DISTANCE > 70
E2	OBSTACLE AT 70 > D > 30
E3	OBSTACLE AT 30 > D > 20
E4	OBSTACLE AT 20 > D

3.3.2. Flowchart Diagram


void APP_init (void)
start
LCD_init()
KEYPAD_init()
BUTTON_init(button0)
ROBOT_init()
ULTRASONIC_init()
PWM_init
end
void APP_start (void)
start
KEYPAD
==
'1'
No
Yes
LCD display in line 1 “Set Def. Rot.”
LCD display the selected option in line 2 “Right”
BUTTON_read (button0)
delay_start_ms 5000
o=1, option[2]={"Left","Right"}
Flag
LOW
No
Delay_ms 2000
dis=ULTRASONIC_read()
check
check=BUTTON_read (button0)
Yes
HIGH
o=~o
LCD display the selected option in line 2 option[o]
Delay_ms 20
Yes
dis>70cm
PWM_start(30)
ROBOT_forward()
LCD display in line 1 “Speed:30% Dir:F”
delay_start_ms 5000
LCD display dis in line 2 “Dist.: 000 Cm”
No
PWM_start(50)
ROBOT_forward()
LCD display in line 1 “Speed:50% Dir:F”
LCD display dis in line 2 “Dist.: 000 Cm”
Flag
&&
dis>70cm
dis=ULTRASONIC_read()
Yes
30<dis<70cm
Yes
No
PWM_start(30)
ROBOT_forward()
LCD display in line 1 “Speed:30% Dir:F”
LCD display dis in line 2 “Dist.: 000 Cm”
Yes
20<dis<30cm
option[o]
==
"Right"
Yes
ROBOT_stop()
PWM_start(30)
ROBOT_right()
LCD display in line 1 “Speed:30% Dir:R”
LCD display dis in line 2 “Dist.: 000 Cm”
No
ROBOT_stop()
PWM_start(30)
ROBOT_left()
LCD display in line 1 “Speed:30% Dir:R”
LCD display dis in line 2 “Dist.: 000 Cm”
No
dis<20cm
Yes
No
dis=ULTRASONIC_read()
ROBOT_stop()
PWM_start(30)
ROBOT_backward()
LCD display in line 1 “Speed:30% Dir:B”
LCD display dis in line 2 “Dist.: 000 Cm”
30>dis>20cm
Yes
KEYPAD
==
"2"
Yes
APP_carstop()
No
end

4. Development Issues

4.1. Team Issues

The first main issue is as a team of ***four*** members who have to engage in completing tasks, we were facing a sort of hardship implementing the drivers and application with only ***three*** members, which led to delay in the project submission .
Secondly, due to the ***team-shift*** process done earlier, we faced ***integration issues*** with drivers previously implemented before the team-shift. Thus, integrating or sometimes redeveloping some APIs led to consuming more time than expected as well.

4.2. Design Issues

We faced multiple issues during implementing the application module using a ***primary design***, leading to consuming more time in order to ***redesign*** our application.

4.3. Drivers Issues

TMR Driver:

Incorrect clock frequency: The *PWM* frequency is determined by the clock frequency and the prescaler value. There was an issue in changing the prescaler in the runtime.

DCM Driver:

Initialization related issues:The DCM driver was not initializing the passed configurations during the runtime, although we checked these configurations multiple times. At the end, we figured out that it was because of the ***multiple*** initialization of the ***PWM*** peripheral, thus it should be the same PWM for both motors when using the same TMR peripheral.

4.4. Application Issues

We really faced an awkward ***bug*** during testing the main ***application***. When we went through initializing ***HAL*** drivers, the ***DCM*** driver wouldn’t work at all if we initialized it before initializing the ***US*** driver, taking into consideration that the two drivers are independent and use different ***TMR*** drivers.

5. References

1. Draw IO
2. Layered Architecture | Baeldung on Computer Science
3. Block Diagram Maker | Free Block Diagram Online | Lucidchart
4. Microcontroller Abstraction Layer (MCAL) | Renesas
5. Hardware Abstraction Layer - an overview | ScienceDirect Topics
6. What is a module in software, hardware and programming?
7. Embedded Basics – API's vs HAL's
8. Using Push Button Switch with Atmega32 and Atmel Studio
9. Interfacing LCD with Atmega32 Microcontroller using Atmel Studio
10. Embedded System Keypad Programming - javatpoint
11. PWM in AVR ATmega16/ATmega32
12. DC motor interfacing with AVR ATmega16/ATmega32
13. Input capture - Wikipedia
14. Ultrasonic Module HC-SR04 interfacing with AVR ATmega16/ATmega32