SPRINTS

Obstacle Avoidance Robot Design

Version 1.0

May 2023

Presented by
Abdelrhman Walaa

Presented to
Sprints

Table Of Content

Obstacle Avoidance Robot V1.0 Design

1. Project Introduction

This project involves developing software for a robot to avoid 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

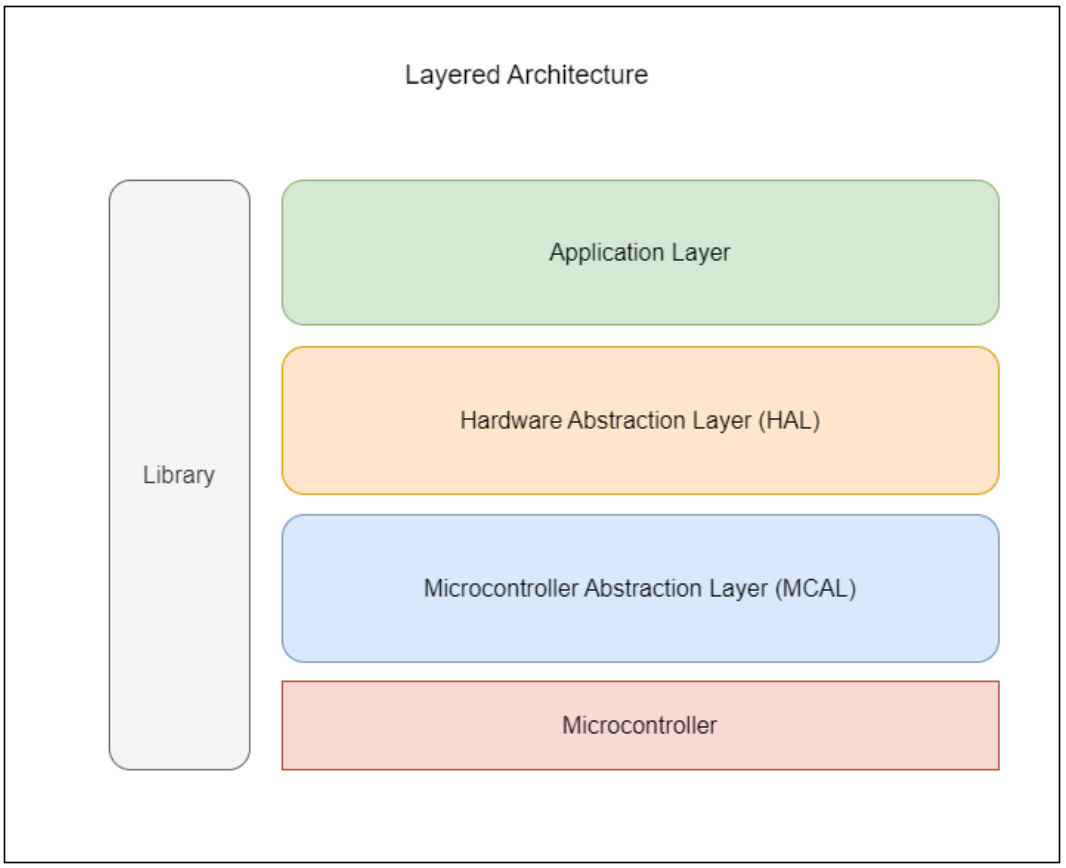


Figure 1. Layered Architecture Design

2.2. System Modules

2.2.1. 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.

2.2.2. Design

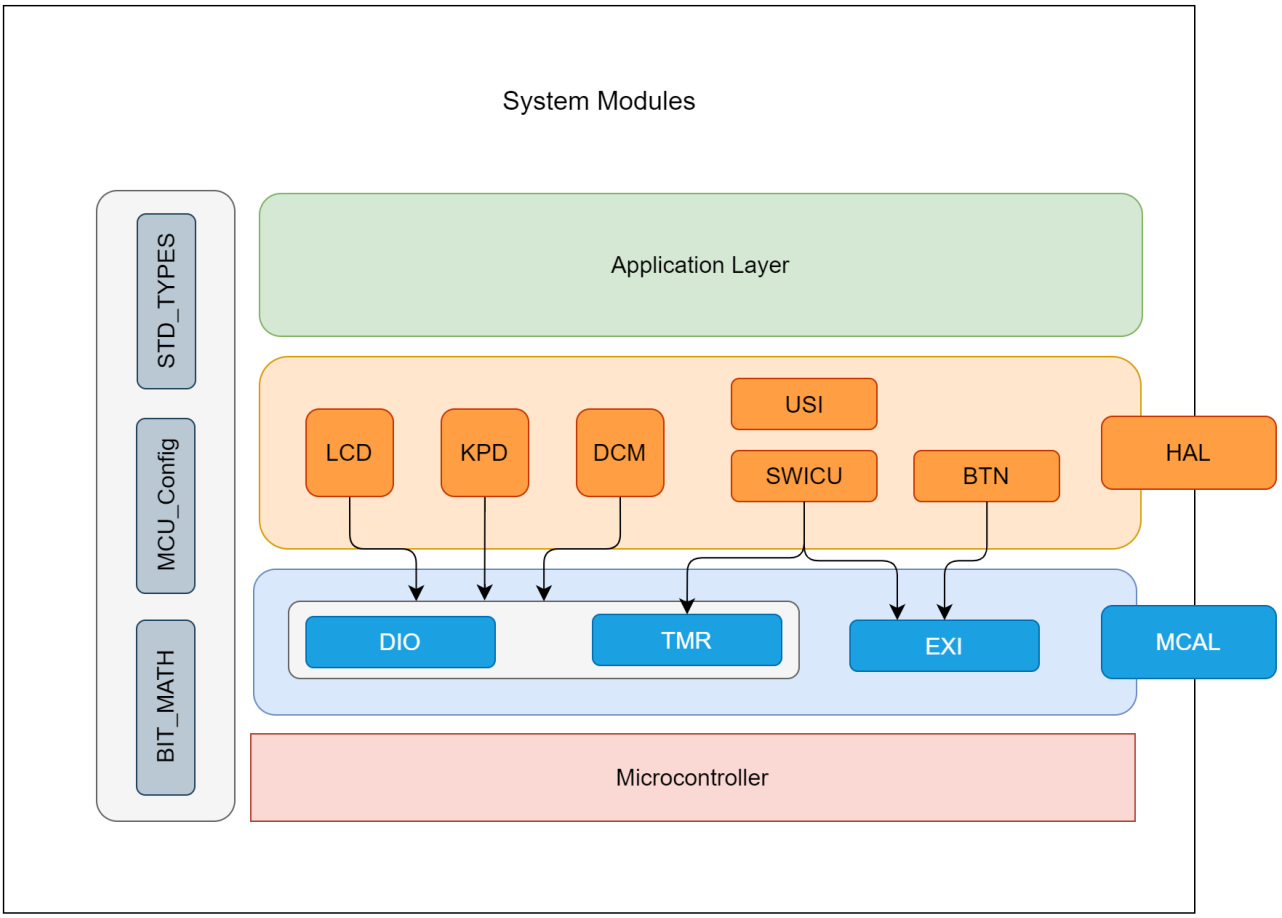


Figure 2. System Modules Design

2.3. Modules Description

2.3.1. 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.

2.3.2. 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.

2.3.3. 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.

2.3.4. 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.

2.3.5. 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.

2.3.6. 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.

2.3.7. 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.

2.3.8. SWICU 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). Thus its a *SWICU* (Software Input Capture Unit)

2.3.9. USI Module

USI (Ultrasonic Interface) Module HC-SR04 works on the principle of SONAR and RADAR systems. HC-SR-04 module has *USI* 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.

2.4. Drivers' Documentation (APIs)

2.4.1 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.

2.4.2. MCAL APIs

2.4.2.1. DIO Driver APIs

```
Precompile Configurations Snippet:

/* Initial Directions of Port A Pins*/
/* Options: DIO_U8_INITIAL_INPUT
 *          DIO_U8_INITIAL_OUTPUT
 */

/* PORTA */
#define DIO_U8_PA0_INITIAL_DIRECTION    DIO_U8_INITIAL_INPUT
#define DIO_U8_PA1_INITIAL_DIRECTION    DIO_U8_INITIAL_INPUT
                .
                .
#define DIO_U8_PA7_INITIAL_DIRECTION    DIO_U8_INITIAL_INPUT

/* Initial Values of Port A Pins */
/* Options:  DIO_U8_INPUT_FLOATING
 *           DIO_U8_INPUT_PULLUP_RESISTOR
 *           DIO_U8_OUTPUT_LOW
 *           DIO_U8_OUTPUT_HIGH
 */

/* PORTA */
#define DIO_U8_PA0_INITIAL_VALUE       DIO_U8_INPUT_FLOATING
#define DIO_U8_PA1_INITIAL_VALUE       DIO_U8_INPUT_FLOATING
                .
                .
#define DIO_U8_PA7_INITIAL_VALUE       DIO_U8_INPUT_FLOATING
```

Linking Configurations Snippet:

There are no Linking Configurations as the defined APIs below could change the DIO peripheral Configurations during the Runtime.

```
| Name: DIO_vdInitialization
| Input: void
| Output: void
| Description: Function to initialize DIO peripheral.
|
vd DIO_vdInitialization (void)

| Name: DIO_u8SetPinDirection
| Input: u8 PortId, u8 PinId, and u8 PinDirection
| Output: u8 Error or No Error
| Description: Function to set Pin direction.
|
u8 DIO_u8SetPinDirection (u8 Cpy_u8PortId, u8 Cpy_u8PinId, u8
Cpy_u8PinDirection)

| Name: DIO_u8SetPinValue
| Input: u8 PortId, u8 PinId, and u8 PinValue
| Output: u8 Error or No Error
| Description: Function to set Pin value.
|
u8 DIO_u8SetPinValue (u8 Cpy_u8PortId, u8 Cpy_u8PinId, u8 Cpy_u8PinValue)

| Name: DIO_u8GetPinValue
| Input: u8 PortId, u8 PinId, and Pointer to u8 ReturnedPinValue
| Output: u8 Error or No Error
| Description: Function to get Pin value.
|
u8 DIO_u8GetPinValue (u8 Cpy_u8PortId, u8 Cpy_u8PinId, u8
*Cpy_pu8ReturnedPinValue)

| Name: DIO_u8TogglePinValue
| Input: u8 PortId and u8 PinId
| Output: u8 Error or No Error
| Description: Function to toggle Pin value.
|
u8 DIO_u8TogglePinValue (u8 Cpy_u8PortId, u8 Cpy_u8PinId)
```

2.4.2.2. 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_u8EnablePIE
| 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_u8EnablePIE (u8 Cpy_u8InterruptId, u8 Cpy_u8SenseControl)
```

```
| Name: EXI_u8DisablePIE
| 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_u8DisablePIE (u8 Cpy_u8InterruptId)
```

```
| Name: EXI_u8SetCallBack
| 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_u8INTSetCallBack (u8 Cpy_u8InterruptId,
void(*Cpy_pfINTInterruptAction)(void))
```

2.4.2.3. TMR Driver APIs

Precompile Configurations Snippet:

```
/* TMR0 Waveform Generation Mode Select */
/* Options: TMR_U8_TMR_0_NORMAL_MODE
 *          TMR_U8_TMR_0_PWM_PHASE_CORRECT_MODE
 *          TMR_U8_TMR_0_CTC_MODE
 *          TMR_U8_TMR_0_FAST_PWM_MODE
 */
#define TMR_U8_TMR_0_MODE_SELECT              TMR_U8_TMR_0_NORMAL_MODE

/* TMR0 Compare Match Output Mode Select */
/* Options: TMR_U8_TMR_0_DISCONNECT_OC0_PIN  // Any Mode
 *          TMR_U8_TMR_0_TOG_OC0_PIN         // Non-PWM Modes only
 *          TMR_U8_TMR_0_CLR_OC0_PIN         // Any Mode (PWM -> Non-Inverting Mode)
 *          TMR_U8_TMR_0_SET_OC0_PIN         // Any Mode (PWM -> Inverting Mode)
 */
#define TMR_U8_TMR_0_COMP_OUTPUT_MODE         TMR_U8_TMR_0_DISCONNECT_OC0_PIN

/* TMR0 Interrupt Select */
/* Options: TMR_U8_TMR_0_NO_INTERRUPT
 *          TMR_U8_TMR_0_COMP_INTERRUPT
 *          TMR_U8_TMR_0_OVF_INTERRUPT
 */
#define TMR_U8_TMR_0_INTERRUPT_SELECT         TMR_U8_TMR_0_NO_INTERRUPT

/* TMR0 Clock Select */
/* Options: TMR_U8_TMR_0_NO_CLOCK_SOURCE // No clock source (Timer/Counter0 stopped)
 *          TMR_U8_TMR_0_NO_PRESCALER    // CLK IO/1 (No prescaling)
 *          TMR_U8_TMR_0_8_PRESCALER     // CLK IO/8 (From prescaler)
 *          TMR_U8_TMR_0_64_PRESCALER    // CLK IO/64 (From prescaler)
 *          TMR_U8_TMR_0_256_PRESCALER   // CLK IO/256 (From prescaler)
 *          TMR_U8_TMR_0_1024_PRESCALER  // CLK IO/1024 (From prescaler)
 *          TMR_U8_TMR_0_EXTERNAL_CLOCK_SOURCE_FALL_EDGE
 *          //External clock source on T0 pin. Clock on falling edge.
 *          TMR_U8_TMR_0_EXTERNAL_CLOCK_SOURCE_RISE_EDGE
 *          // External clock source on T0 pin. Clock on rising edge.
 */
#define TMR_U8_TMR_0_CLOCK_SELECT             TMR_U8_TMR_0_NO_CLOCK_SOURCE

/* TMR0 Other Configurations */
#define TMR_U8_TMR_0_PRELOAD_VALUE            0
#define TMR_U8_TMR_0_COMPARE_VALUE            0
#define TMR_U16_TMR_0_NUM_OF_OVERFLOWS        1
```

Linking Configurations Snippet:

There are no Linking Configurations as the defined APIs below could change the DIO peripheral Configurations during the Runtime.

```
| Name: TMR_vdTMR0Initialization
| Input: void
| Output: void
| Description: Function to Initialize TMR0 peripheral.
|
vd TMR_vdTMR0Initialization (void)

| Name: TMR_vdTMR2Initialization
| Input: void
| Output: void
| Description: Function to Initialize TMR2 peripheral.
|
vd TMR_vdTMR2Initialization (void)

| Name: TMR_u8EnableTMR
| Input: u8 TimerId
| Output: u8 Error or No Error
| Description: Function to Enable TMR peripheral.
|
u8 TMR_u8EnableTMR (u8 Cpy_u8TimerId)

| Name: TMR_u8DisableTMR
| Input: u8 TimerId
| Output: u8 Error or No Error
| Description: Function to Disable TMR peripheral.
|
u8 TMR_u8DisableTMR (u8 Cpy_u8TimerId)

| Name: TMR_u8DelayMS
| Input: u8 TimerId, u8 InterruptionMode, and u32 Delay
| Output: u8 Error or No Error
| Description: Function to use TMR peripheral as Delay in MS.
|
u8 TMR_u8DelayMS (u8 Cpy_u8TimerId, u8 Cpy_u8InterruptionMode, u32
Cpy_u32Delay)

| Name: TMR_u8DelayUS
| Input: u8 TimerId, u8 InterruptionMode, and u32 Delay
| Output: u8 Error or No Error
| Description: Function to use TMR peripheral as Delay in US.
|
u8 TMR_u8DelayUS (u8 Cpy_u8TimerId, u8 Cpy_u8InterruptionMode, u32
Cpy_u32Delay)
```

```
| Name: TMR_u8OVFSetCallBack
| Input: u8 TimerId and Pointer to function OVFInterruptAction taking void and
|        returning 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 Timer (TimerId), the address
|              is passed through a pointer to function (OVFInterruptAction), and then
|              pass this address to the ISR function.
|
u8 TMR_u8OVFSetCallBack (u8 Cpy_u8TimerId,
void(*Cpy_pfOVFInterruptAction)(void))

| Name: TMR_u8GetOVFFlagStatus
| Input: u8 TimerId and Pointer to u8 ReturnedFlagStatus
| Output: u8 Error or No Error
| Description: Function to Get status of the OVF Flag in TMR peripheral.
|
u8 TMR_u8GetOVFFlagStatus (u8 Cpy_u8TimerId, u8 *Cpy_pu8FlagStatus)

| Name: TMR_u8ClearOVFFlag
| Input: u8 TimerId
| Output: u8 Error or No Error
| Description: Function to Clear the OVF Flag in TMR peripheral.
|
u8 TMR_u8ClearOVFFlag (u8 Cpy_u8TimerId)
```

2.4.3. HAL APIs

2.4.3.1. BTN Driver APIs

```
| Name: BTN_u8GetBTNState
| Input: u8 BTNId and Pointer to u8 ReturnedBTNState
| Output: u8 Error or No Error
| Description: Function to get BTN state.
|
u8 BTN_u8GetBTNState (u8 Cpy_u8BTNId, u8 *Cpy_pu8ReturnedBTNState)
```

2.4.3.2. LCD Driver APIs

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

| Name: LCD_vdSendCmnd
| Input: u8 Cmnd
| Output: void
| Description: Function to send a Command to LCD through Data pins.
|
vd LCD_vdSendCmnd (u8 Cpy_u8Cmnd)

| Name: LCD_vdSendChar
| Input: u8 Char
| Output: void
| Description: Function to send a Character to LCD through Data pins.
|
vd LCD_vdSendChar (u8 Cpy_u8Char)

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

```
| Name: LCD_u8GoToLocation
| Input: u8 LineNumber and u8 DisplayLocation
| Output: u8 Error or No Error
| Description: Function to set the Address Counter (AC) of LCD to a certain location
|              in DDRAM.
|
u8 LCD_u8GoToLocation (u8 Cpy_u8LineNumber, u8 Cpy_u8DisplayLocation)

| Name: LCD_u8WriteString
| Input: Pointer to u8 String
| Output: vou8 Error or No Error
| Description: Function to send an array of characters to LCD through Data pins
|              (From CGROM to DDRAM).
u8 LCD_u8WriteString (u8 *Cpy_pu8String)

| Name: LCD_vdWriteNumber
| Input: s64 Number
| Output: void
| Description: Function to send a number (positive or negative) to LCD through Data
|              pins (From CGROM to DDRAM).
|
vd LCD_vdWriteNumber (s64 Cpy_s64Number)
```

2.4.3.3. KPD Driver APIs

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

| Name: KPD_vdDisableKPD
| Input: void
| Output: void
| Description: Function to disable Keypad.
|
vd KPD_vdDisableKPD (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_u8GetPressedKey (u8 *Cpy_pu8ReturnedKeyValue)
```

2.4.3.4. DCM Driver APIs

```
| Name: DCM_vdInitialization
| Input: Pointer to u8 ShutdownFlag variable that acts as a main kill switch.
| Output: void
| Description: Function to initialize the DCM by initializing their pins.
|
vd DCM_vdInitialization (u8 **Cpy_ppu8ShutdownFlag)

| Name: DCM_u8RotateDCM
| Input: void
| Output: u8 Error or No Error
| Description: Function to rotate the DCM by changing its direction to right,
|              setting the duty cycle of the PWM signal to a predefined value,
|              and then changing the direction of the motor again to the right.
|
u8 DCM_u8RotateDCM (void)

| Name: DCM_u8ChangeDCMDirection
| Input: u8 DCMId
| Output: u8 Error or No Error
| Description: Function to change the direction of the motor rotation for the
|              specified motor.
|
u8 DCM_u8ChangeDCMDirection (u8 Cpy_u8DCMId)

| Name: DCM_u8SetDutyCycleOfPWM
| Input: u8 DutyCycleValue
| Output: u8 Error or No Error
| Description: Function to Set Duty Cycle Value of DCM PWM.
|
u8 DCM_u8SetDutyCycleOfPWM (u8 Cpy_u8DutyCycleValue)

| Name: DCM_vdStopMotor
| Input: void
| Output: void
| Description: Function to Stop DCM.
|
vd DCM_vdStopDCM (void)

| Name: DCM_vdStopMotor
| Input: void
| Output: void
| Description: Function to be called by the TMR Overflow Callback function to update
|              the stop flag.
|
vd DCM_vdUpdateStopFlag (void)
```

2.4.3.5. SWICU Driver APIs

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

| Name: SWICU_vdConfigureOnPeriod
| Input: void
| Output: void
| Description: Function to configure On Period of a signal by enabling TMR to start
|              counting and reconfiguring EXI to detect the falling edge.
|
vd SWICU_vdConfigureOnPeriod (void)

| Name: SWICU_vdConfigureOffPeriod
| Input: void
| Output: void
| Description: Function to configure Off Period of a signal by saving the On Period,
|              then resetting TMR to restart counting, and reconfiguring EXI to detect
|              the rising edge.
|
vd SWICU_vdConfigureOffPeriod (void)

| Name: SWICU_u8GetDutyCycle
| Input: void
| Output: void
| Description: Function to Get the Duty Cycle value using the value of both On and
|              Off Periods of a signal.
|
u8 SWICU_u8GetDutyCycle (u8 *Cpy_u8DutyCycleValue)

| Name: SWICU_vdSwitchState
| Input: void
| Output: void
| Description: Function to be called by the EXI Callback function to switch states.
|
vd SWICU_vdSwitchState (void)
```

2.4.3.6. USI Driver APIs

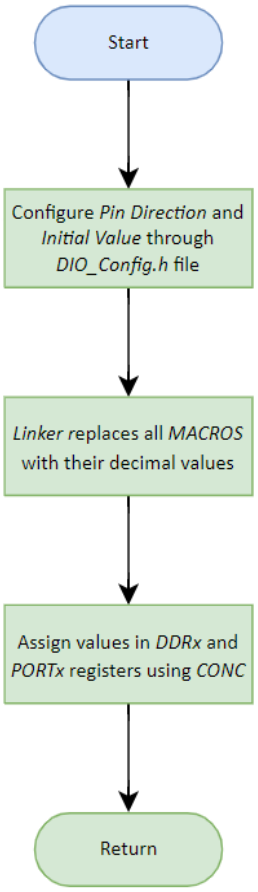
```
| Name: USI_vdSendTriggerPulse
| Input: void
| Output: void
| Description: Function to send a trigger pulse to the USI.
|
vd USI_vdSendTriggerPulse (void)

| Name: USI_u8ReadEchoSignal
| Input: Pointer to u8 ReturnedEchoSignalValue
| Output: u8 Error or No Error
| Description: Function to read the echo signal from the USI.
|
u8 USI_u8ReadEchoSignal (u8 *Cpy_pu8ReturnedEchoSignalValue)

| Name: USI_u8CalculateDistance
| Input: Pointer to u8 ReturnedDistanceValue
| Output: u8 Error or No Error
| Description: Function to calculate the distance to an object from the echo signal.
|
u8 USI_u8CalculateDistance (u16 *Cpy_pu16ReturnedDistanceValue)
```

3. Low Level Design

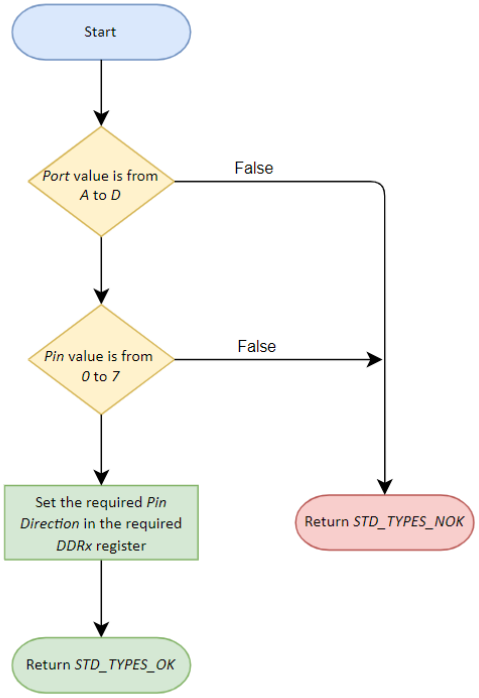
3.1. MCAL Layer

3.1.1. DIO Module

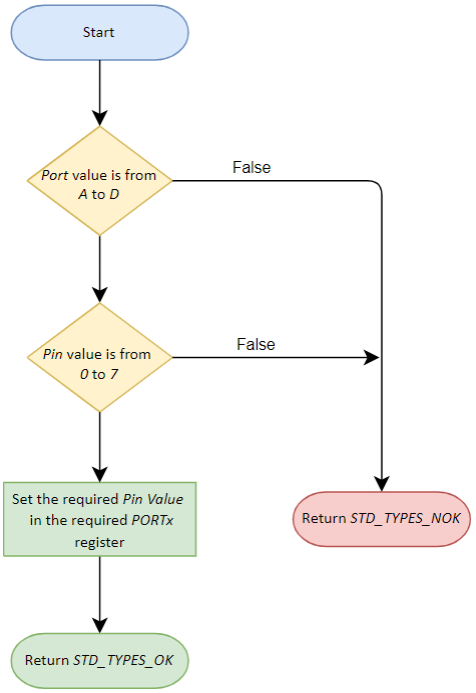
A. DIO_vdInitialization



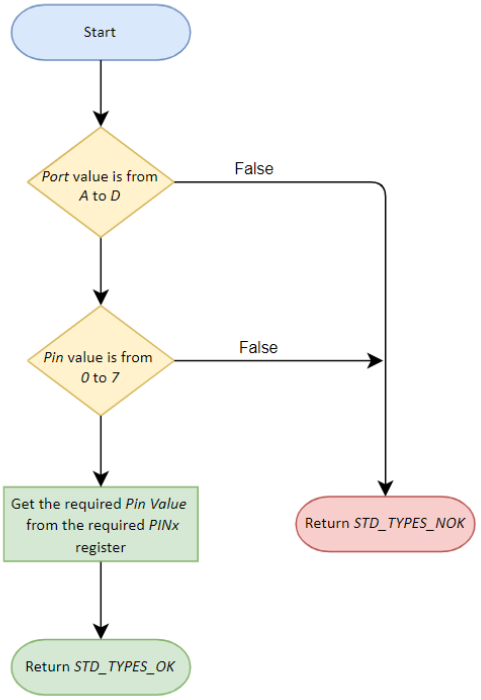
B. DIO_u8SetPinDirection



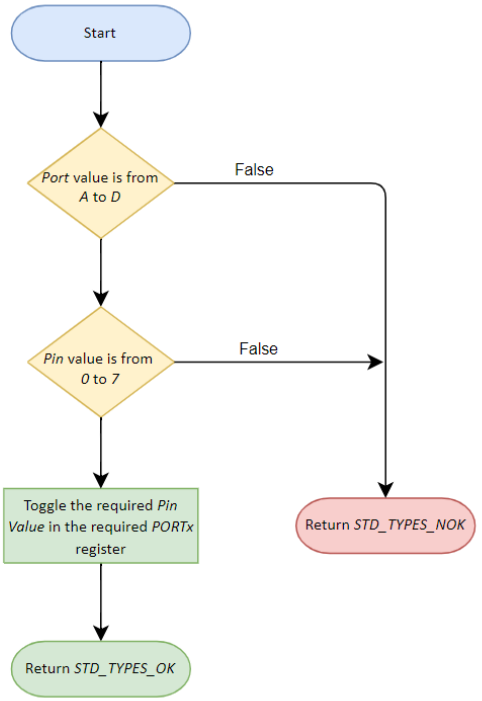
C. DIO_u8SetPinValue



D. DIO_u8GetPinValue

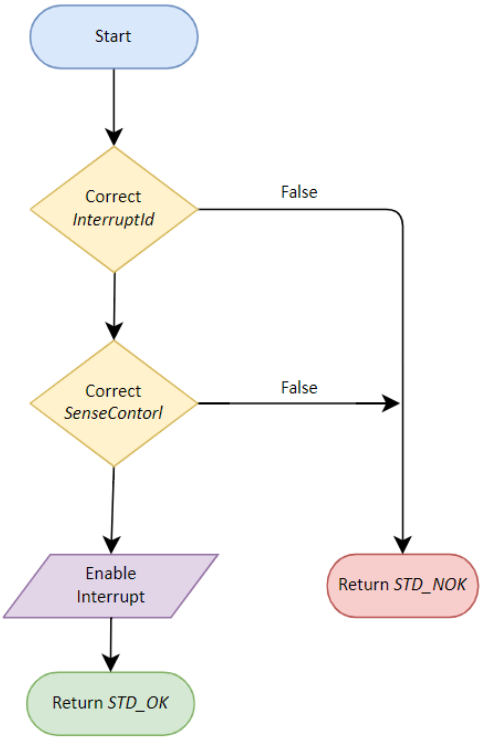


E. DIO_u8TogglePinValue

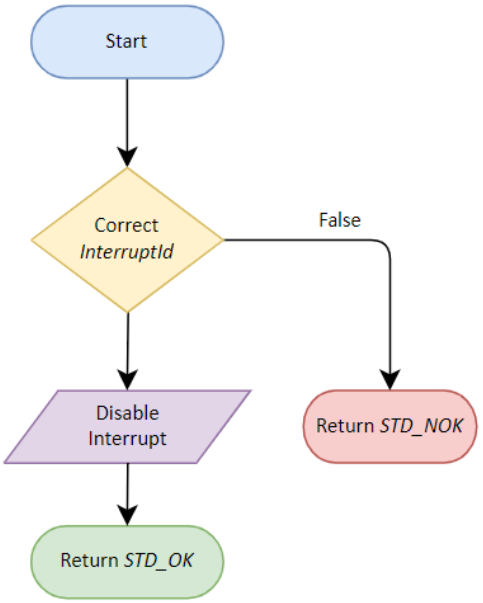


3.1.2. EXI Module

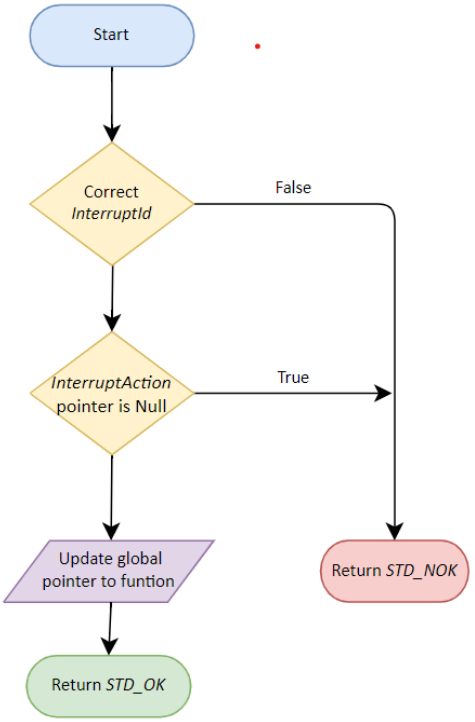
A. EXI_u8EnablePIE



B. EXI_u8DisablePIE

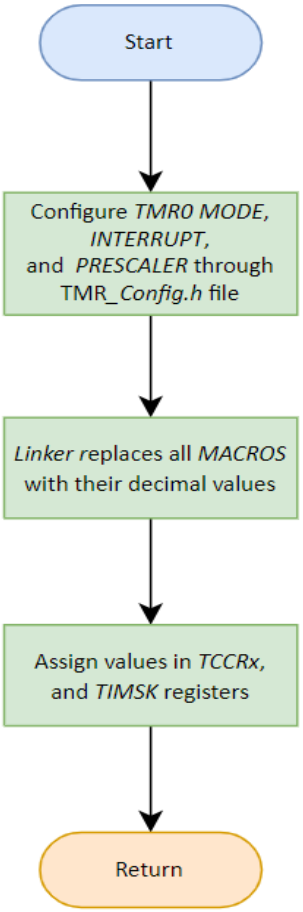


C. *EXI_u8INTSetCallBack*

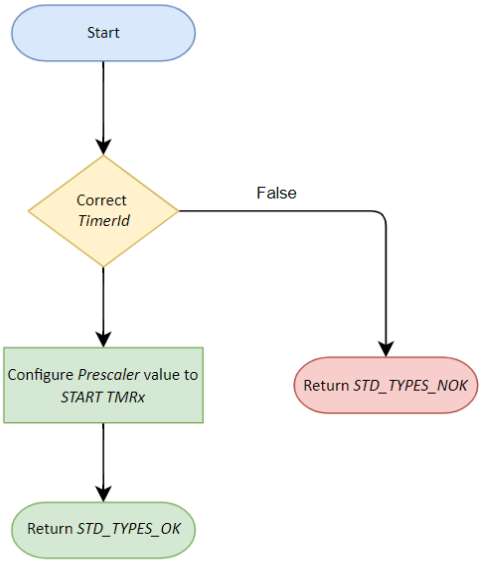


3.1.3. TMR Module

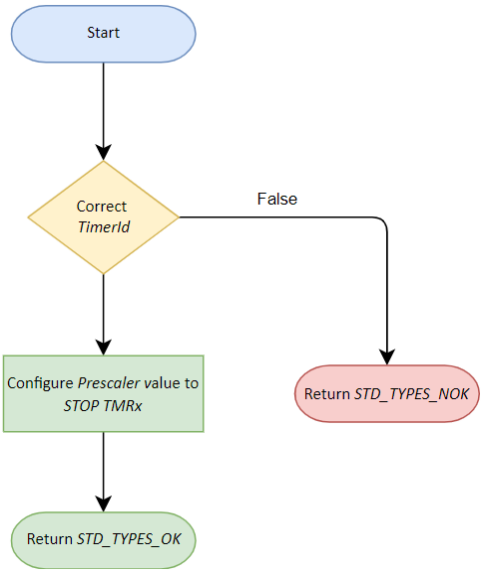
A. *TMR_vdTMR0Initialization*
B. *TMR_vdTMR2Initialization*



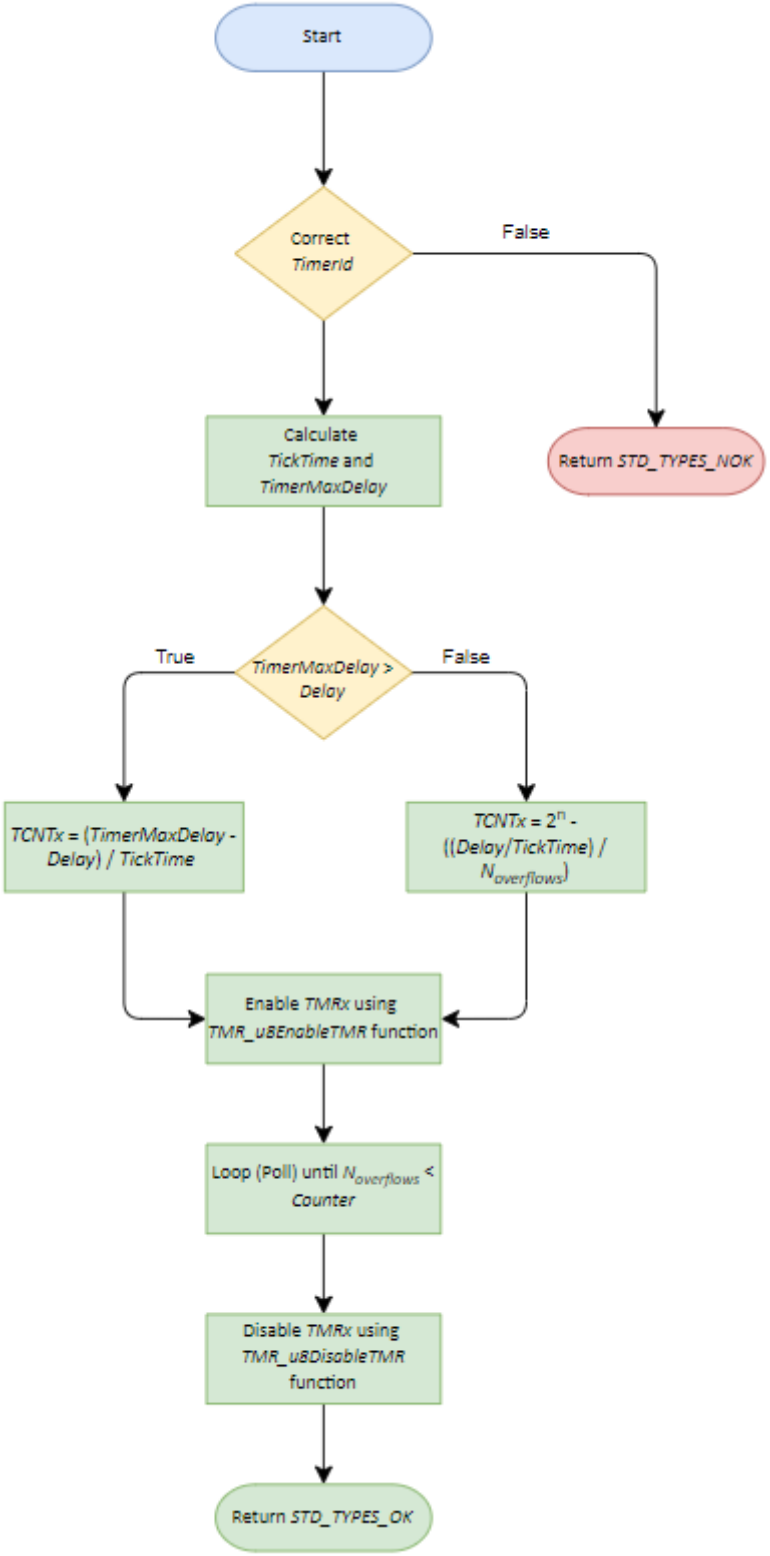
C. *TMR_u8EnableTMR*



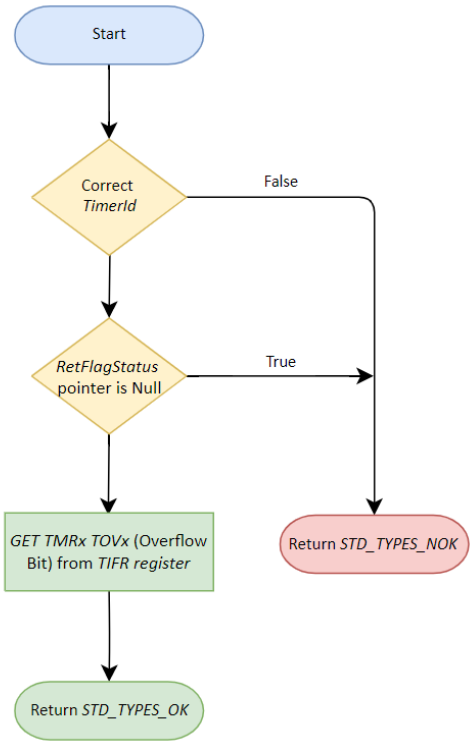
D. *TMR_u8DisableTMR*



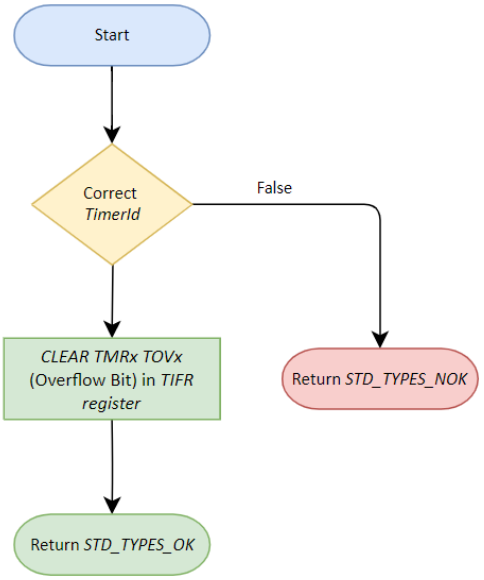
E. TMR_u8DelayMS



F. TMR_u8GetOVFFlagStatus



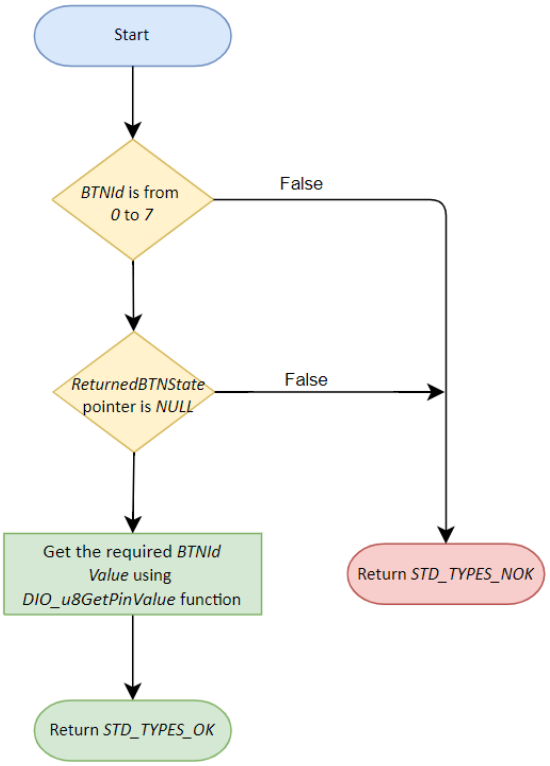
G. TMR_u8ClearOVFFlag



3.2. HAL Layer

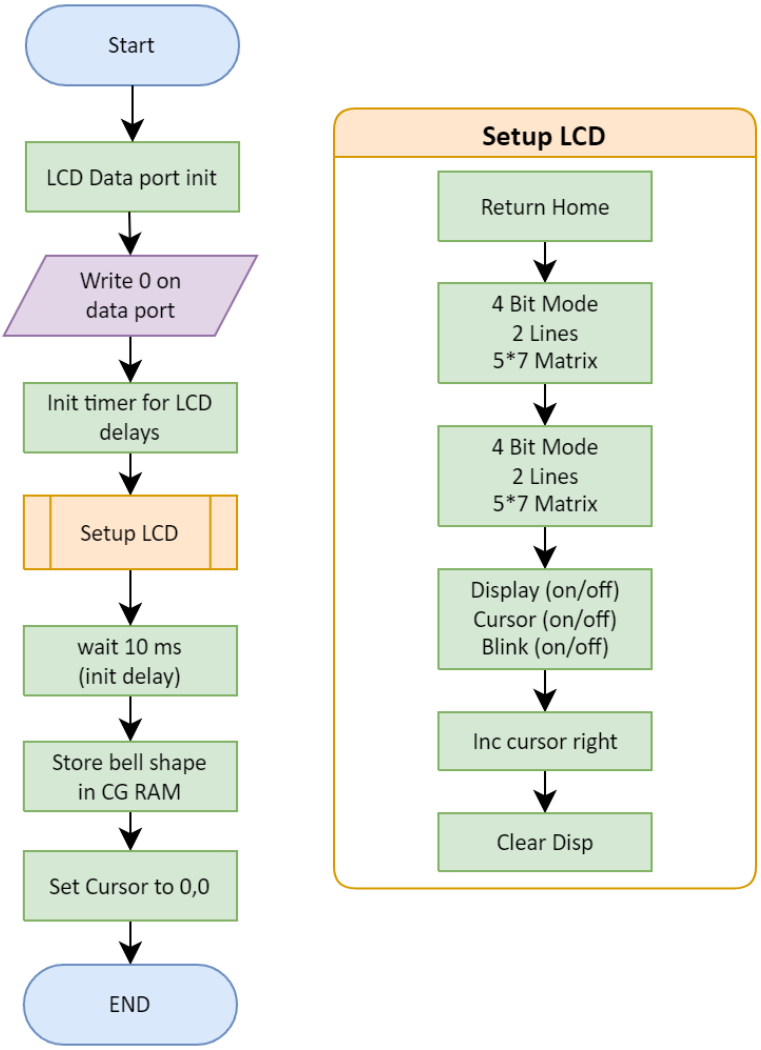
3.2.1. BTN Module

A. *BTN_u8GetBTNState*

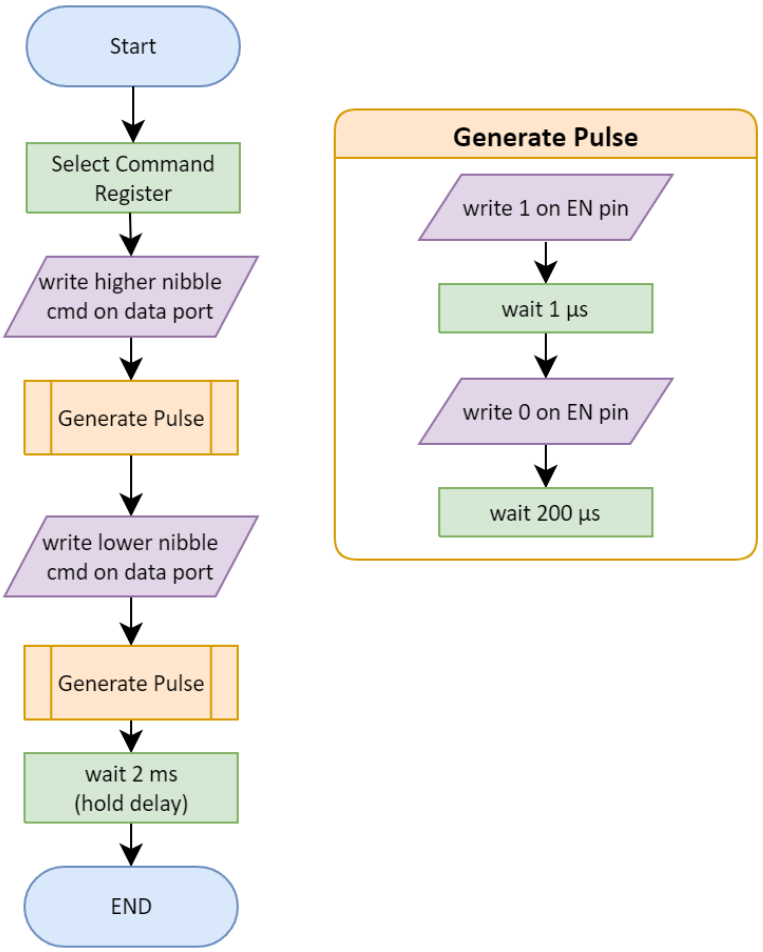


3.2.2. LCD Module

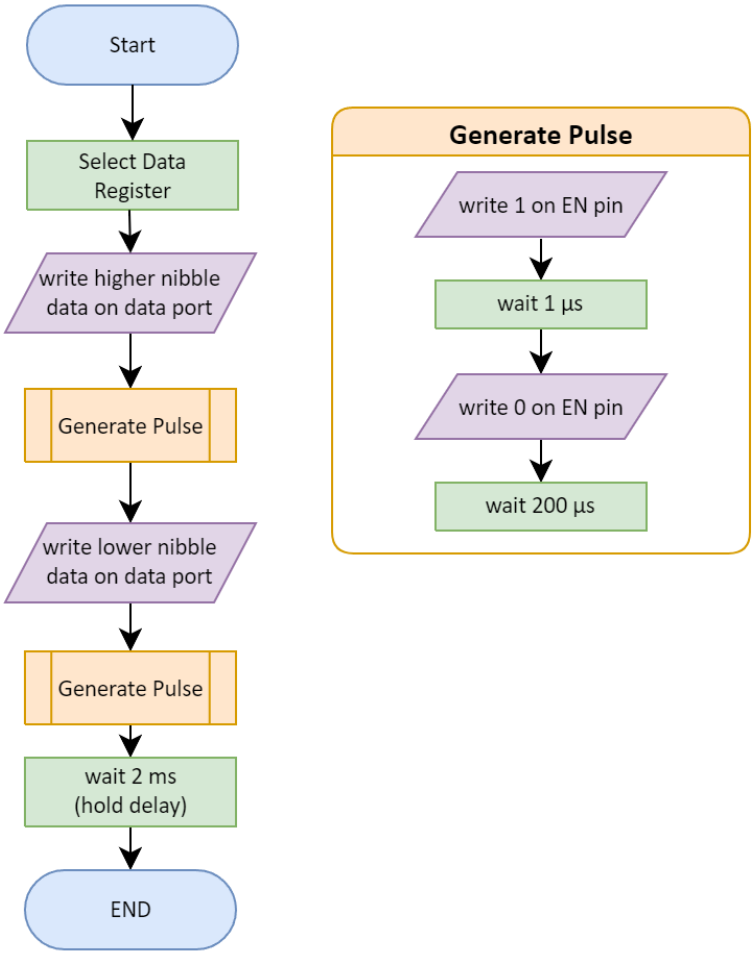
A. LCD_vdInitialization



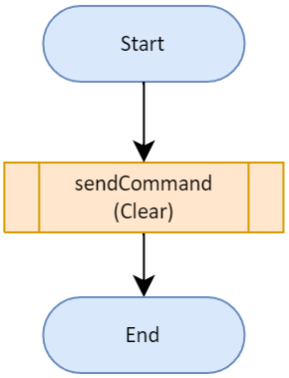
B. LCD_vdSendCmnd



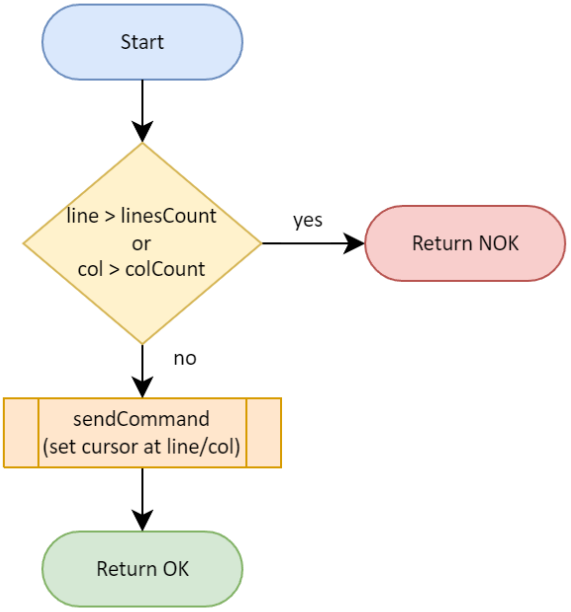
C. LCD_vdSendChar



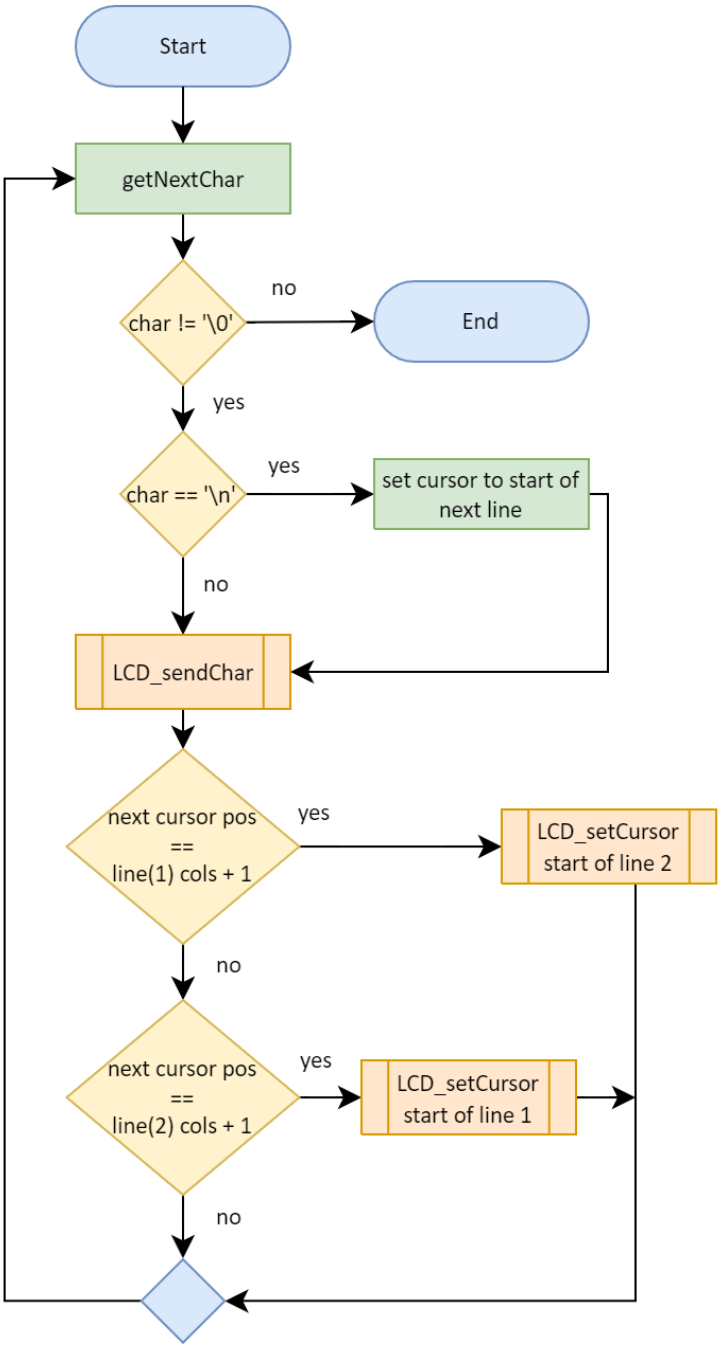
D. LCD_vdClearDisplay



E. LCD_u8GoToLocation

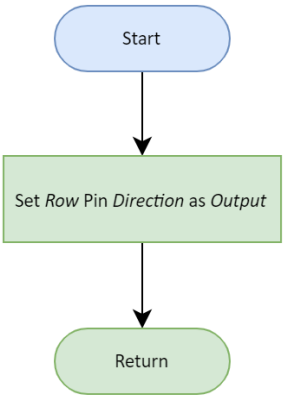


F. LCD_u8WriteString

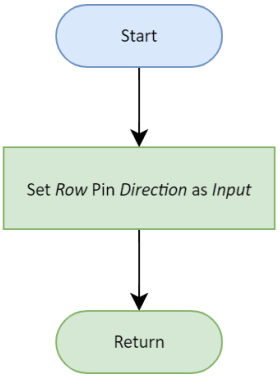


3.2.3. KPD Module

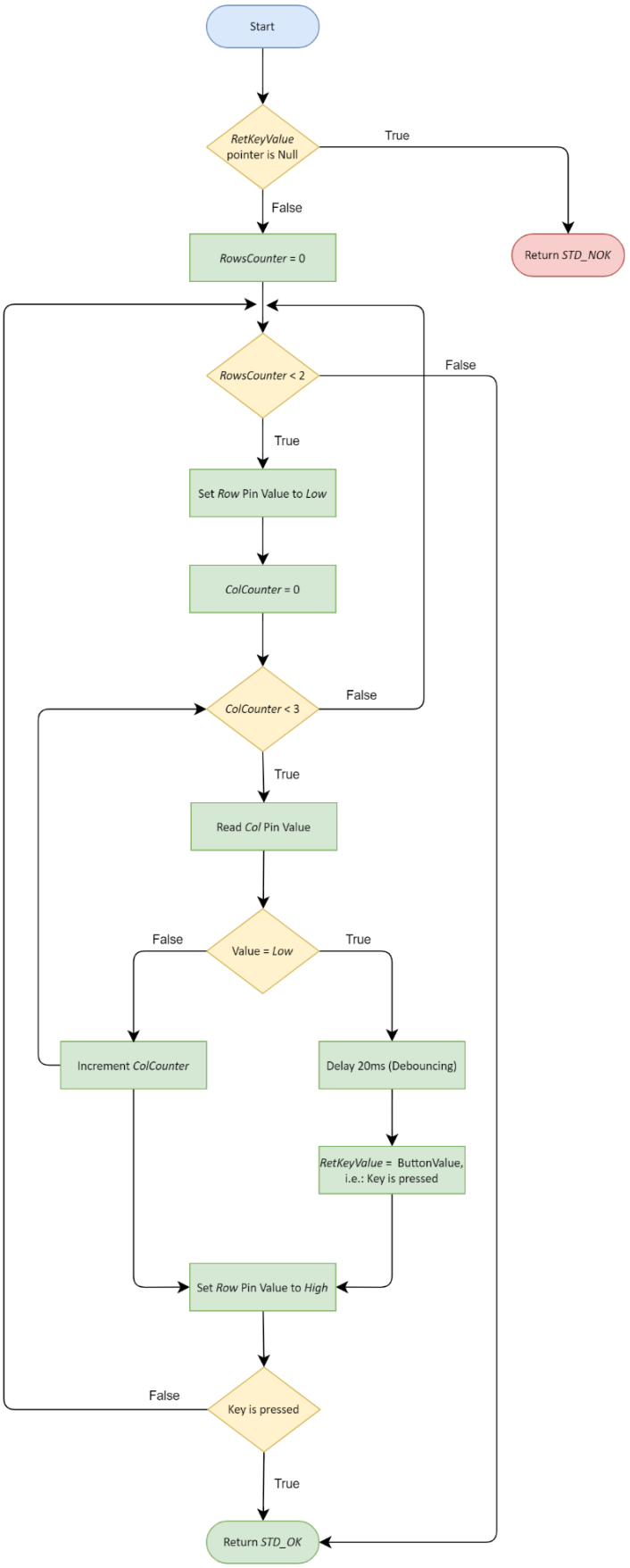
A. KPD_vdEnableKPD



B. KPD_vdDisableKPD

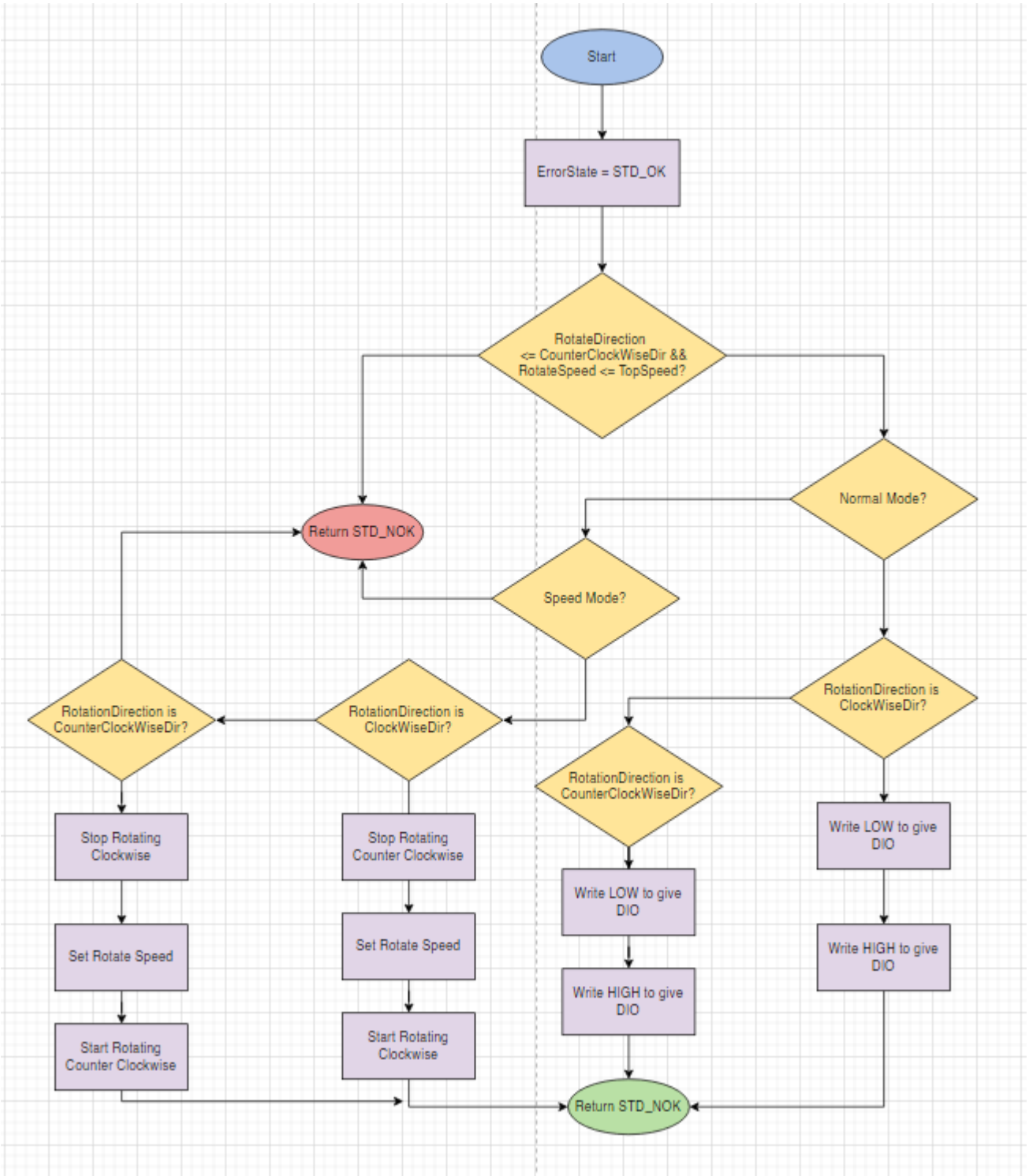


C. KPD_u8GetPressedKey

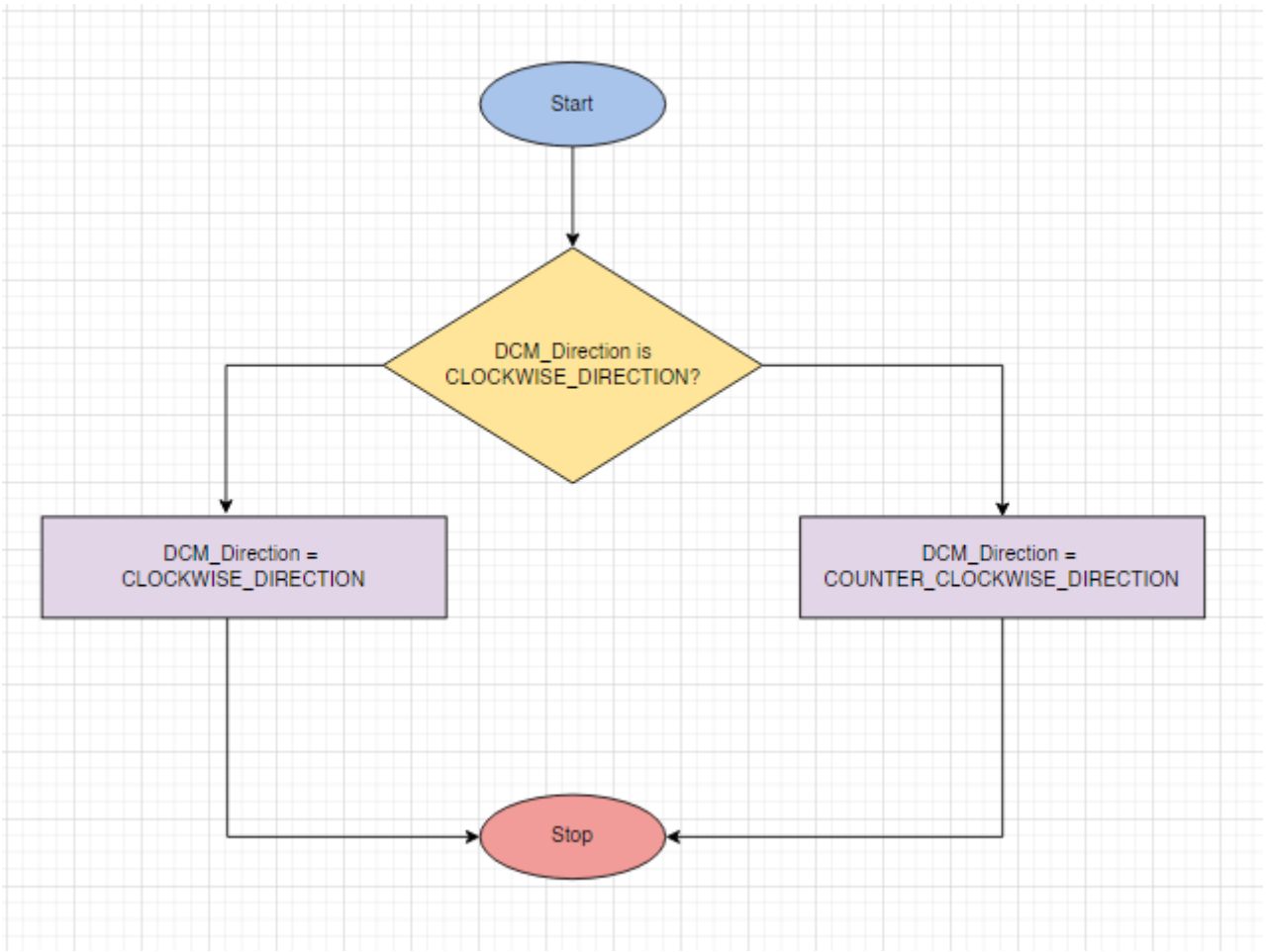


3.2.4. DCM Module

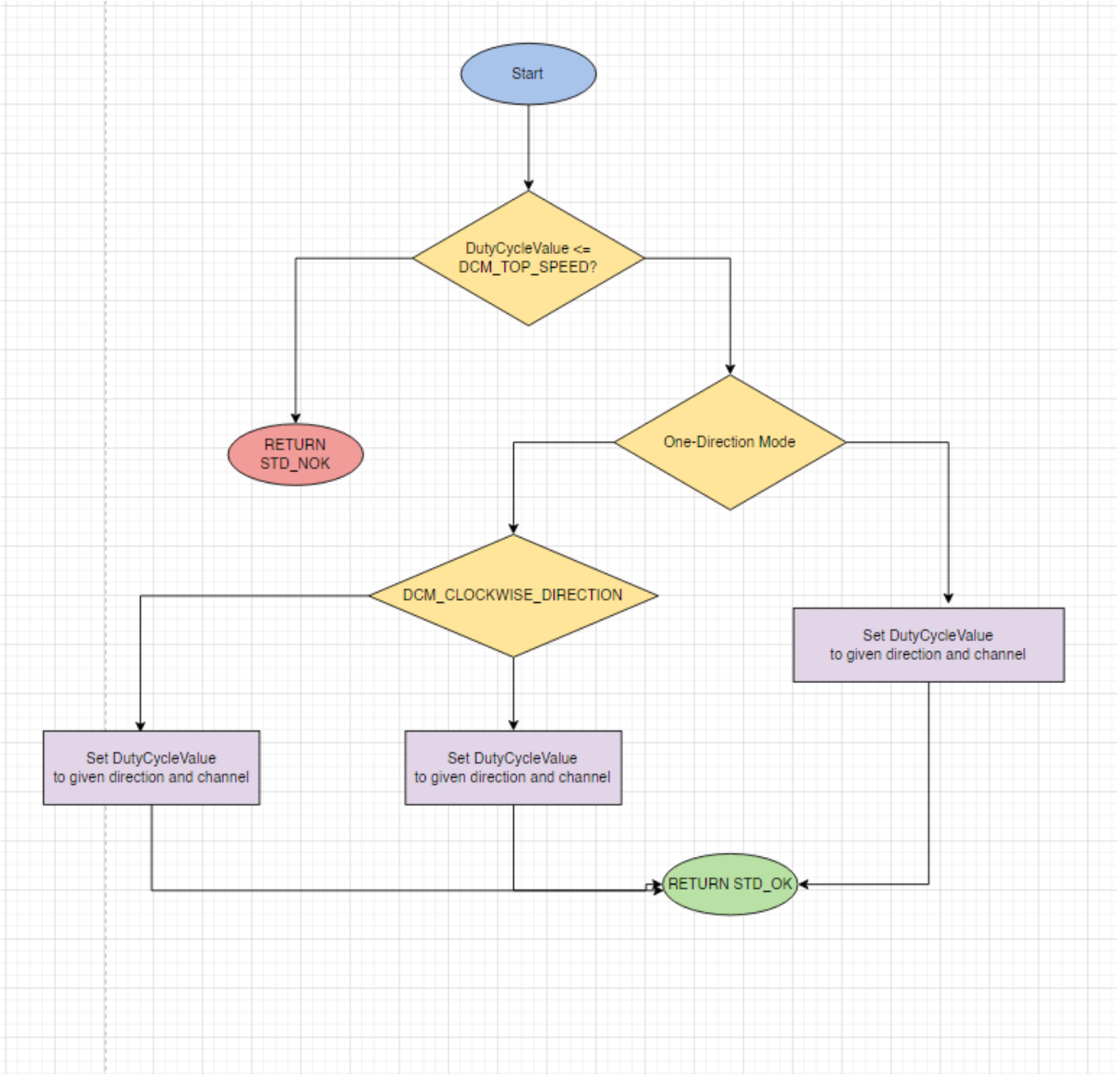
A. DCM_u8RotateDCM



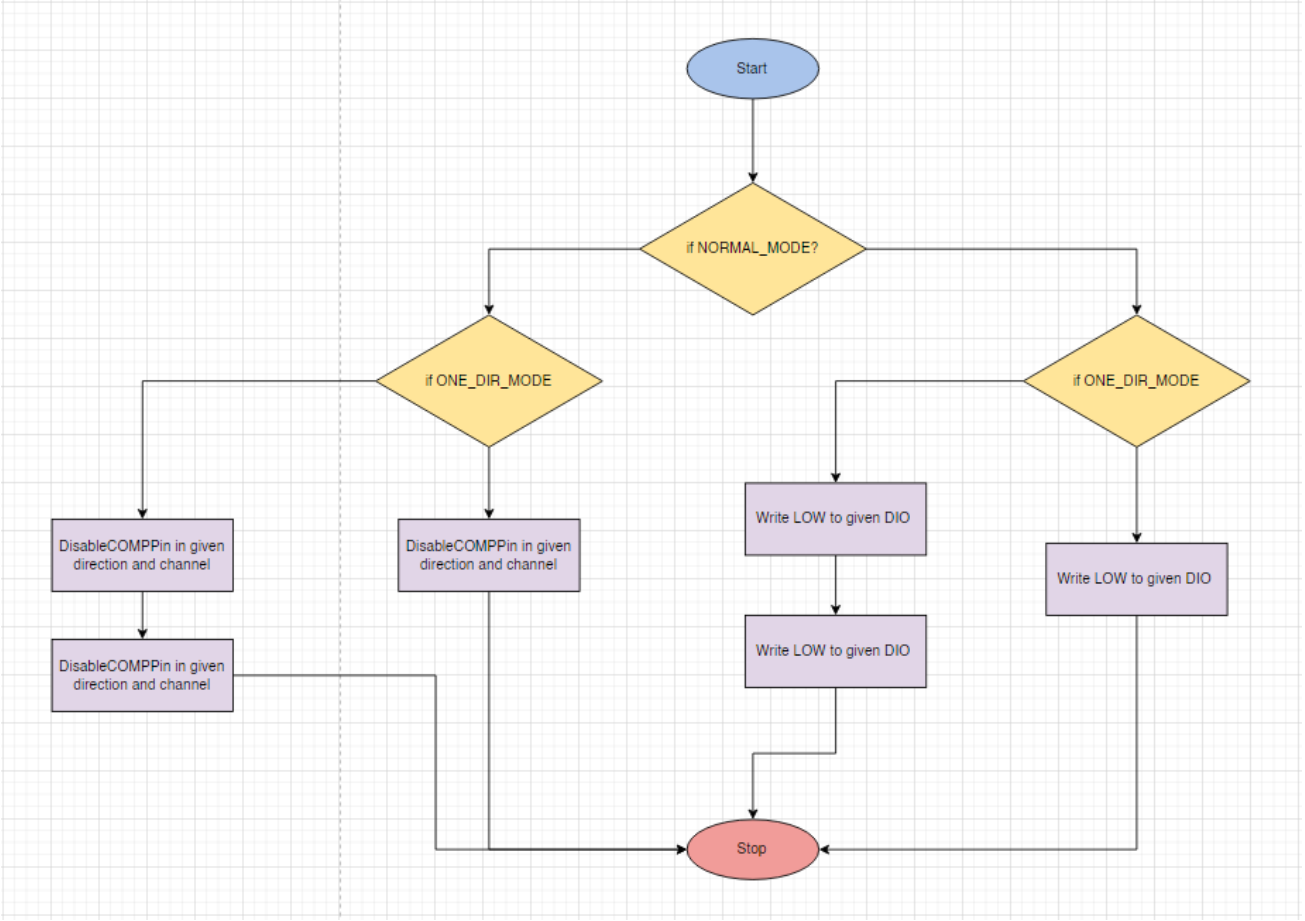
B. DCM_u8ChangeDCMDirection



C. DCM_u8SetDutyCycleOfPWM

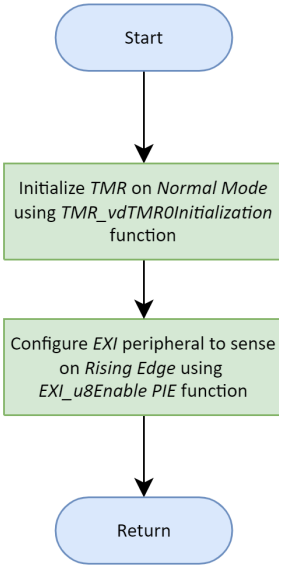


D. DCM_vdStopDCM

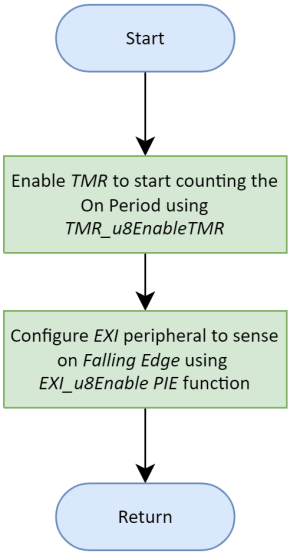


3.2.5. SWICU Module

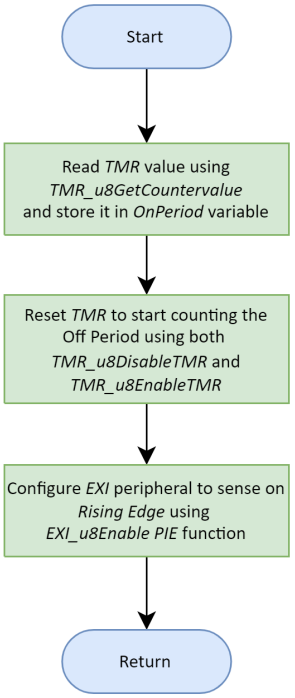
A. *SWICU_vdInitialization*



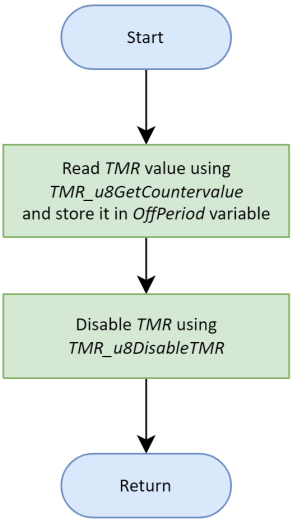
B. *SWICU_vdConfigureOnPeriod*



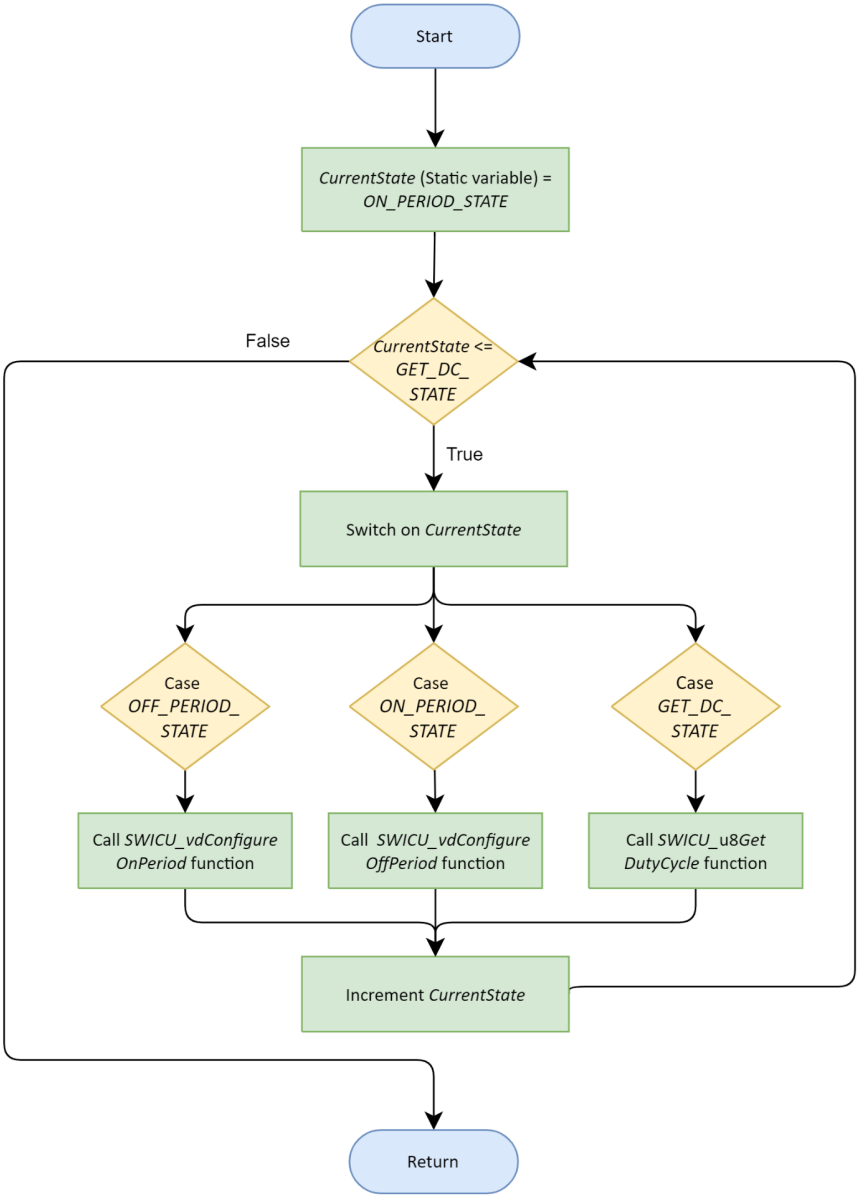
C. SWICU_vdConfigureOffPeriod



D. SWICU_u8GetDutyCycle

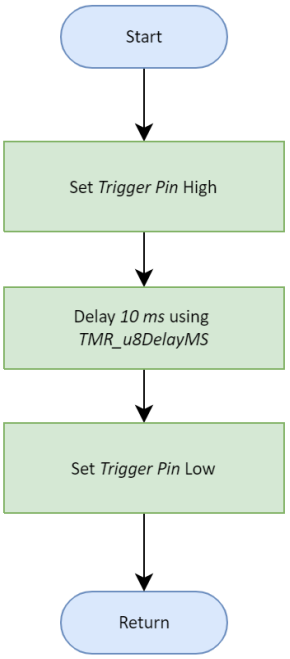


E. SWICU_vdSwitchState

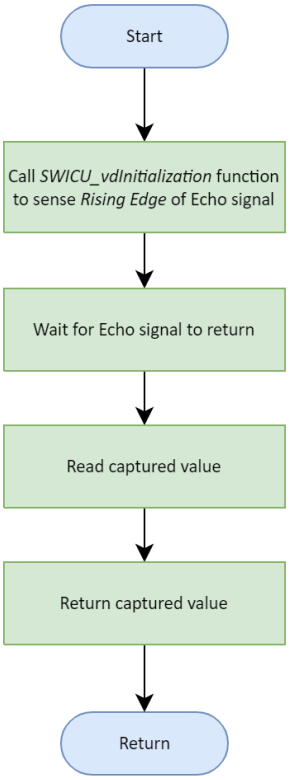


3.2.6. USI Module

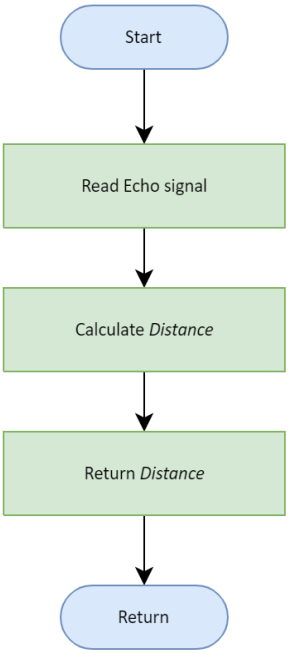
A. USI_vdSendTriggerPulse



B. USI_u8ReadEchoSignal



C. *USI_u8CalculateDistance*



4. References

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