OBSTACLE AVOIDANCE CAR DESIGN

PROPOSAL

This proposal outlines the design of an obstacle avoidance car system using the ATmega32 microcontroller. The project aims to develop a versatile car that can navigate its environment while intelligently avoiding obstacles. The document provides an overview of the system, its components, and the planned approach for implementation.

Prepared By
Hossam Elwahsh

Sprints

Obstacle Avoidance Car Design

1. Project Introduction

This project aims to design a collision avoidance system for a four-wheel drive robot. By implementing intelligent sensing and control mechanisms, the system enables the robot to detect and avoid obstacles in its path, ensuring safe navigation.

1.1. Project Components

- ATmega32 microcontroller
- **Four** motors (**M1**, **M2**, **M3**, **M4**)
- One button to change default rotation direction (**PBUTTON0**)
- Keypad 3x3 (KPD1: start, KPD2: stop)
- One ultrasonic sensor
    - Vcc to **5v**
    - GND to GND
    - Trig to **PB3 (INT1)**
    - Echo to **PB2 (INT0)**
- LCD **2x16**

1.2. System Requirements

System 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:**
    - 4.1. The LCD will display a centered message in **line 1 "Set Def. Rot."**
    - 4.2. The LCD will display the selected option in **line 2 "Right"**
    - 4.3. The robot will wait for **5 seconds** to choose between **Right and Left**
        - 4.3.1. When **PBUTTON0** is pressed **once**, the default rotation will be **Left** and the **LCD line 2 will be updated**
        - 4.3.2. When **PBUTTON0** is pressed again, the default rotation will be **Right** and the **LCD line 2 will be updated**
        - 4.3.3. For each press the default rotation will changed and the **LCD line 2** is updated
        - **4.3.4. After the 5 seconds the default value of rotation is set (timeout)**
    - 4.4. 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**:
    - 5.1. The robot will move **forward** with **30%** speed for **5 seconds**
    - 5.2. **After 5 seconds** it will move with **50%** speed **as long as** there was **no object** or objects are located at **more than 70 centimeters** distance
    - 5.3. 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
    - 5.4. The LCD will display Object distance in line 2 **"Dist.: 000 Cm"**

6. For Obstacles located **between 30 and 70 centimeters**
    - 6.1. The robot will **decrease** its **speed to 30%**
    - 6.2. LCD data is updated

7. For Obstacles located **between 20 and 30 centimeters**
    - 7.1. The robot will **stop** and **rotates 90 degrees** to **right/left** according to the chosen configuration
    - 7.2. The LCD data is updated

8. For Obstacles located **less than 20 centimeters**
    - 8.1. The robot will **stop**, **move backwards** with **30% speed until distance is greater than 20 and less than 30**
    - 8.2. The LCD data is updated
    - 8.3. Then perform **point 8**

9. <u>Obstacles surrounding the robot (Bonus)</u>
   9.1. If the robot rotated for **360 degrees** without finding any distance greater **than 20** it will **stop**
   9.2. LCD data will be updated.
   9.3. 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

1.3. Assumptions

- 4WD Robot Specifications - [4WD Complete Mini Plastic Robot Chassis Kit]
- For the Robot to rotate in place around its pivot point we calculated that:
  - Left motors going forward, Right motors going backward
  - Rotation frequency: 100 Hz
  - Rotation duration: 620 ms
  - Rotation duty cycle: 50%

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

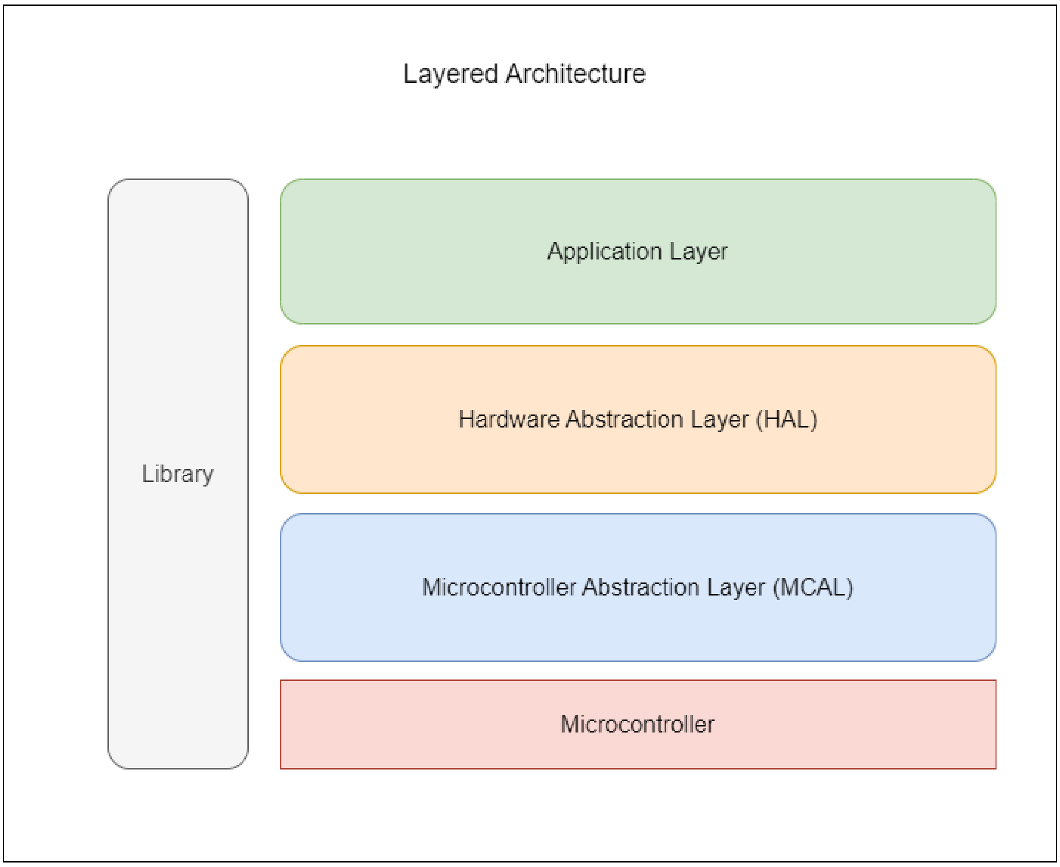


Figure 1. Layered Architecture Design

2.2. Modules Description

2.2.1. DIO (Digital Input/Output) Module

The *DIO* 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.

2.2.2. 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.

2.2.3. TIMER Module

The *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.

2.2.4. ICU Module

The ICU Module is a software component designed to interface with an Ultrasonic sensor in our project. It enables the accurate detection of the distance between the car and surrounding objects in all directions. By leveraging input capture techniques, the ICU Module captures the sensor's readings and provides real-time distance measurements. This crucial information aids in implementing a comprehensive collision avoidance system, ensuring safe navigation for the vehicle in various scenarios.

2.2.5. LCD Module

LCD stands for "Liquid Crystal Display," which is a type of flat-panel display used in electronic devices to display text and graphics. We're using the 4-bit mode to reduce the number of I/O pins needed to interface with the LCD. The module includes a controller, a display, and a backlight. By interfacing with the LCD module and writing the necessary code, we're able to provide the user with real-time information about the current speed and direction of the car as well as providing an interface to set the default rotation direction.

2.2.6. BTN Module

The *BTN* (Button) module is responsible for reading the state of the system's buttons. It provides a set of APIs to enable/disable button interrupts, set the button trigger edge (rising/falling/both), and define an ISR that will be executed when a button press is detected.

2.2.7. DCM Module

The *DCM* (DC Motor) module is responsible for controlling the speed and direction of the system's DC motors. It provides a set of APIs to set the speed and direction of each motor, and to stop all motors. It also uses the *TIMER* module to generate *PWM* (Pulse Width Modulation) signals that control the motor speed.

2.2.8. KPD Module

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 Keypad 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.

2.2.9. Ultrasonic Module

The Ultrasonic module is a key component in our project for distance sensing. It utilizes ultrasonic waves to measure distances between the module and surrounding objects. By emitting ultrasonic pulses and calculating the time taken for the echoes to return, the module provides accurate distance measurements. Its compact size, low power consumption, and wide detection range make it an ideal choice for collision avoidance applications. With its reliable performance and easy integration, the Ultrasonic module enhances the safety and precision of our project's distance sensing capabilities.

2.2.10. Robot Controller Module

The Robot Controller module serves as the central control unit for our obstacle avoidance car project. It seamlessly integrates and manages the peripherals including the ICU for input capture, LCD for display, Ultrasonic sensor for distance sensing, and DC Motors for motion control. This module coordinates the sensor readings, interprets the data, and employs intelligent algorithms to make informed decisions. It provides precise control signals to the DC Motors for maneuvering the car and displays relevant information on the LCD. With its comprehensive functionality and efficient coordination of peripherals, the Robot Controller module ensures optimal performance and reliable obstacle avoidance capabilities for our car.

2.2.11. Design

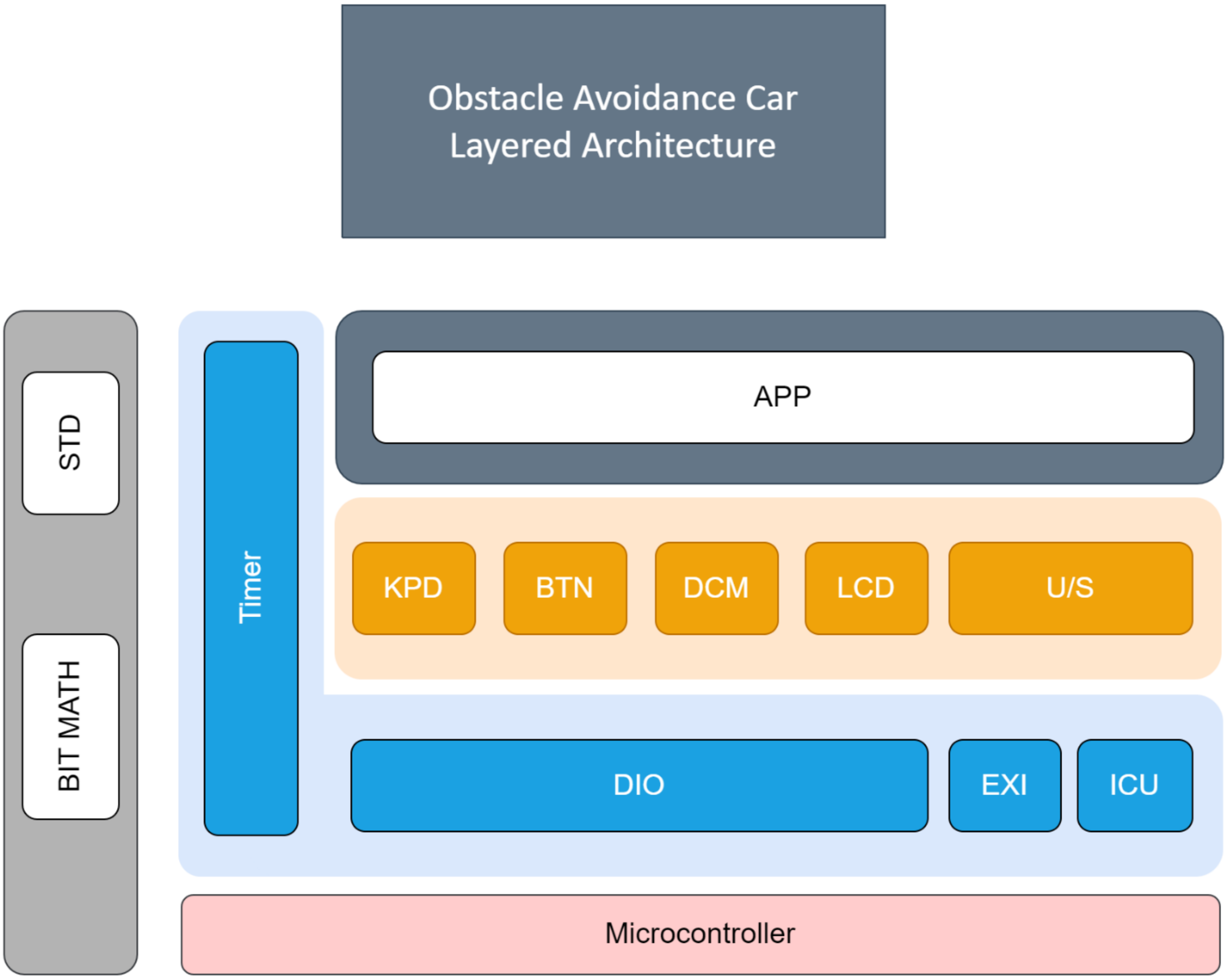


Figure 3. System Modules Design

2.3. Drivers' Documentation (APIs)

2.3.1 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.

2.3.2. MCAL APIs

2.3.2.1. DIO Driver

```
| Enumeration of possible DIO ports
typedef enum EN_DIO_PORT_T
{
      PORT_A, /*!< Port A */
      PORT_B, /*!< Port B */
      PORT_C, /*!< Port C */
      PORT_D  /*!< Port D */
}EN_DIO_PORT_T;

| Enumeration for DIO direction.
|
| This enumeration defines the available directions for a
| Digital Input/Output (DIO) pin.
|
| Note
|     This enumeration is used as input to the DIO driver functions
|     for setting the pin direction.
|
typedef enum EN_DIO_DIRECTION_T
{
      DIO_IN = 0,       /**< Input direction */
      DIO_OUT = 1       /**< Output direction */
} EN_DIO_DIRECTION_T;

| Enumeration of DIO error codes
typedef enum EN_DIO_ERROR_T
{
      DIO_OK,      /**< Operation completed successfully */
      DIO_ERROR    /**< An error occurred during the operation */
} EN_DIO_ERROR_T;
```

```
| Initializes a pin of the DIO interface with a given direction
|
| Parameters
|        [in] u8_a_pinNumber  The pin number of the DIO interface to initialize
|        [in] en_a_portNumber The port number of the DIO interface to initialize
|                                (PORT_A, PORT_B, PORT_C or |PORT_D)
|        [in] en_a_direction  The direction to set for the pin
|                                (DIO_IN or DIO_OUT)
| Returns
|        An EN_DIO_ERROR_T value indicating the success or failure of the
|        operation (DIO_OK if the operation succeeded, DIO_ERROR otherwise)
|
EN_DIO_ERROR_T DIO_init(u8 u8_a_pinNumber, EN_DIO_PORT_T en_a_portNumber,
EN_DIO_DIRECTION_T en_a_direction);


| Reads the value of a pin on a port of the DIO interface
|
| Parameters
|        [in] u8_a_pinNumber  The pin number to read from the port
|        [in] en_a_portNumber The port number to read from
|                                (PORT_A, PORT_B, PORT_C or |PORT_D)
|        [out] u8_a_value     Pointer to an unsigned 8-bit integer where
|                                the value of the pin will be stored
| Returns
|        An EN_DIO_ERROR_T value indicating the success or failure of the
|        operation (DIO_OK if the operation succeeded, DIO_ERROR otherwise)
|
EN_DIO_ERROR_T DIO_read(u8 u8_a_pinNumber, EN_DIO_PORT_T en_a_portNumber, u8 *
u8_a_value);


| Writes a digital value to a specific pin in a specific port.
|
| Parameters
|        [in] u8_a_pinNumber  The pin number to write to
|        [in] en_a_portNumber The port number to write to
|                                (PORT_A, PORT_B, PORT_C or |PORT_D)
|        [in] u8_a_value     The digital value to write
|                                (either DIO_U8_PIN_HIGH or DIO_U8_PIN_LOW)
| Returns
|        EN_DIO_ERROR_T Returns DIO_OK if the write is successful,
|        DIO_ERROR otherwise.
|
EN_DIO_ERROR_T DIO_write(u8 u8_a_pinNumber, EN_DIO_PORT_T en_a_portNumber, u8
u8_a_value);
```

```
| Initializes a port of the DIO interface with a given direction and mask
|
| Parameters
|     [in] en_a_portNumber The port number of the DIO interface to initialize
|                               (PORT_A, PORT_B, PORT_C or PORT_D)
|     [in] en_a_dir        The direction to set for the port (INPUT or OUTPUT)
|     [in] u8_a_mask       The mask to use when setting the DDR of the port
|                               (DIO_NO_MASK, DIO_MASK_BITS_n..)
| Returns
|     An EN_DIO_ERROR_T value indicating the success or failure of the
|     operation (DIO_OK if the operation succeeded, DIO_ERROR otherwise)
|
EN_DIO_ERROR_T DIO_portInit(EN_DIO_PORT_T en_a_portNumber, EN_DIO_DIRECTION_T
en_a_dir, u8 u8_a_mask);

| Writes a byte to a port of the DIO interface
|
| Parameters
|     [in] en_a_portNumber  The port number of the DIO interface to write to
|                               (PORT_A, PORT_B, PORT_C or PORT_D)
|     [in] u8_a_portValue   The byte value to write to the port
|                               (DIO_U8_PORT_LOW, DIO_U8_PORT_HIGH)
|     [in] u8_a_mask        The mask to use when setting the PORT of the port
|                               (DIO_NO_MASK, DIO_MASK_BITS_n..)
| Returns
|     An EN_DIO_ERROR_T value indicating the success or failure of the operation
|     (DIO_OK if the operation succeeded, DIO_ERROR otherwise)
|
EN_DIO_ERROR_T DIO_portWrite(EN_DIO_PORT_T en_a_portNumber, u8 u8_a_portValue,
u8 u8_a_mask);

| Toggles the state of the pins of a port of the DIO interface
|
| Parameters
|     [in] en_a_portNumber  The port number of the DIO interface to toggle
|                               (PORT_A, PORT_B, PORT_C or PORT_D)
|     [in] u8_a_mask        The mask to use when toggling the PORT of the port
|                               (DIO_NO_MASK, DIO_MASK_BITS_n..)
| Returns
|     An EN_DIO_ERROR_T value indicating the success or failure of the operation
|     (DIO_OK if the operation succeeded, DIO_ERROR otherwise)
|
EN_DIO_ERROR_T DIO_portToggle(EN_DIO_PORT_T en_a_portNumber, u8 u8_a_mask);
```

2.3.2.2. EXI Driver

```
| The function enables a specific external interrupt with a specified sense control.
|
| Parameters
|      [in] u8_a_interruptId specifies the ex. interrupt ID
|                                  (EXI_U8_INT0, EXI_U8_INT1, or EXI_U8_INT2)
|      [in] u8_a_senseControl specifies sense control for the EXI.
|                                  (EXI_U8_SENSE_LOW_LEVEL,...)
| Return
|      If the function executes successfully, it will return STD_OK (0)
|      If there is an error, it will return STD_NOK (1).
|
u8 EXI_enablePIE(u8 u8_a_interruptId, u8 u8_a_senseControl);

| The function disables a specified external interrupt.
|
| Parameters
|      [in] u8_a_interruptId interrupt ID to disable. It should be a
|            value between 0 and 2, where 0 represents INT0, 1 represents INT1, and 2
|            represents INT2.
| Return
|      STD_OK if the function executed successfully, and STD_NOK if there was an error
|
u8 EXI_disablePIE(u8 u8_a_interruptId);

| function sets a callback function for a specific interrupt and returns an error
state.
|
| Parameters
|      [in] u8_a_interruptId An unsigned 8-bit integer representing
|            the ID of the interrupt. It should be in the range of 0 to 2, inclusive.
|      [in] pf_a_interruptAction A pointer to a function that will be
|            executed when the specified interrupt occurs.
| Return
|      a u8 value which represents the error state. It can be either
|      STD_OK (0) or STD_NOK (1).
|
u8 EXI_intSetCallBack(u8 u8_a_interruptId, void (*pf_a_interruptAction)(void))
```

2.3.2.3. TIMER Driver

```
| Initializes timer0 at normal mode
|
| This function initializes/selects the timer_0 normal mode for the
|  timer, and enables the ISR for this timer.
| Parameters
|            [in] en_a_interrputEnable value to set the interrupt
|                                      bit for timer_0 in the TIMSK reg.
|             [in] **u8_a_shutdownFlag double pointer, acts as a main switch for
|                                      timer0 operations.
| Return
|      An EN_TIMER_ERROR_T value indicating the success or failure of
|            the operation (TIMER_OK if the operation succeeded, TIMER_ERROR
|            otherwise)
|
EN_TIMER_ERROR_T TIMER_timer0NormalModeInit(EN_TIMER_INTERRPUT_T
en_a_interrputEnable, u8 ** u8_a_shutdownFlag);



| Creates a delay using timer_0 in overflow mode
|
| This function Creates the desired delay on timer_0 normal mode.
| Parameters
|            [in] u16_a_interval value to set the desired delay.
| Return
|      An EN_TIMER_ERROR_T value indicating the success or failure of
|            the operation (TIMER_OK if the operation succeeded, TIMER_ERROR
|            otherwise)
|
EN_TIMER_ERROR_T TIMER_timer0Delay(u16 u16_a_interval);



| Start the timer by setting the desired prescaler.
|
| This function sets the prescaler for timer_0.
| Parameters
|            [in] u16_a_prescaler value to set the desired prescaler.
| Return
|      An EN_TIMER_ERROR_T value indicating the success or failure of
|            the operation
|             (TIMER_OK if the operation succeeded, TIMER_ERROR otherwise)
|
EN_TIMER_ERROR_T TIMER_timer0Start(u16 u16_a_prescaler);
```

```
| Stop the timer by setting the prescaler to be 000--> timer is stopped.
|
| This function clears the prescaler for timer_0.
|
| Return
|      void
|
void TIMER_timer0Stop(void);

| Initializes timer2 at normal mode
|
| This function initializes/selects the timer_2 normal mode for the
|  timer, and enables the ISR for this timer.
| Parameters
|             [in] en_a_interrputEnable value to set
|             the interrupt bit for timer_2 in the TIMSK reg.
|              [in] **u8_a_shutdownFlag double pointer, acts as a main switch for
|                                       timer0 operations.
| Return
|      An EN_TIMER_ERROR_T value indicating the success or failure of
|             the operation (TIMER_OK if the operation succeeded, TIMER_ERROR
/             otherwise)
|
EN_TIMER_ERROR_T TIMER_timer2NormalModeInit(EN_TIMER_INTERRPUT_T
en_a_interrputEnable, u8 **u8_a_shutdownFlag);

| Stop the timer by setting the prescaler to be 000--> timer is stopped.
|
| This function clears the prescaler for timer_2.
| Parameters
|             [in] void.
| Return
|      void
|
void TIMER_timer2Stop(void);

| Start the timer by setting the desired prescaler.
|
| This function sets the prescaler for timer_2.
| Parameters
|             [in] u16_a_prescaler value to set the desired prescaler.
| Return
|      An EN_TIMER_ERROR_T value indicating the success or failure of
|             the operation (TIMER_OK if the operation succeeded, TIMER_ERROR
|             otherwise)
|
EN_TIMER_ERROR_T TIMER_timer2Start(u16 u16_a_prescaler);
```

```
| Creates a delay using timer_2 in overflow mode
|
| This function Creates the desired delay on timer_2 normal mode.
| Parameters
|            [in] u16_a_interval value to set the desired delay.
| Return
|      An EN_TIMER_ERROR_T value indicating the success or failure of
|            the operation
|             (TIMER_OK if the operation succeeded, TIMER_ERROR otherwise)
|
EN_TIMER_ERROR_T TIMER_timer2Delay(u16 u16_a_interval);



| Set callback function for timer overflow interrupt
|
| Parameters
|            void_a_pfOvfInterruptAction Pointer to the function to be
|                                        called on timer overflow interrupt
| Return
|      EN_TIMER_ERROR_T Returns TIMER_OK if callback function is set
|                       successfully, else returns TIMER_ERROR
|
EN_TIMER_ERROR_T TIMER_ovfSetCallback(void
(*void_a_pfOvfInterruptAction)(void));



| Interrupt Service Routine for Timer Overflow.
|      This function is executed when Timer2 Overflows.
|      It increments u16_g_overflow2Ticks counter and checks whether
|      u16_g_overflow2Numbers is greater than u16_g_overflow2Ticks.
|      If true, it resets u16_g_overflow2Ticks and stops Timer2.
|      It then checks whether void_g_pfOvfInterruptAction is not null.
|      If true, it calls the function pointed to by
|     void_g_pfOvfInterruptAction.
|
| Return
|     void
|
ISR(TIMER_ovfVect);
```

2.3.2.4. ICU Driver

```
| Initializes the ICU driver
|
| This function initializes a software ICU driver, init echo pin as input,
| uses timer to calculate elapsed time, when echo pin is triggered it sends the
| elapsed time duration to the callback function
|
| Parameters
|           [in]u8_a_echoPin I/O pin number to receive echoed signal
|           [in]cf_a_timeReceived callback function to send elapsed duration
|
| Return
|     None
|
void ICU_init(u8 u8_a_echoPin, CallbackFunc cf_a_timeReceived);

| Resets and starts the ICU algorithm to capture the elapsed time duration now
| starting until echo signal is received back
|
| Return
|     None
|
void ICU_getCaptureValue(void);
```

2.3.3. HAL APIs

2.3.3.1. LCD APIs

```
| Initializes the LCD module.
|
| This function initializes the LCD module by configuring the data port,
| configuring the LCD to 4-bit mode,setting the display to on with cursor
| and blink, setting the cursor to increment to the right, and clearing
|  the display.
| It also pre-stores a bell shape at CGRAM location 0.
|
| Return
|      void
|
void LCD_init(void);



| Sends a command to the LCD controller
|
| Sends the upper nibble of the command to the LCD's data pins, selects
|  the command register by setting RS to low,
| generates an enable pulse, delays for a short period, then sends the
|  lower nibble of the command and generates
| another enable pulse. Finally, it delays for a longer period to ensure
|  the command has been executed by the LCD
| controller.
|
| Parameters
|            [in] u8_a_cmd The command to be sent
|
void LCD_sendCommand(u8 u8_a_cmd);



| Sends a single character to the LCD display
|
| This function sends a single character to the LCD display by selecting
|  the data register and sending the
| higher nibble and lower nibble of the character through the data port.
| The function uses a pulse on the enable pin to signal the LCD to read
|  the data on the data port.
| The function also includes delays to ensure proper timing for the LCD
|  to read the data.
|
| Parameters
|            [in] u8_a_data single char ASCII data to show
|
void LCD_sendChar(u8 u8_a_data);
```

```
| Displays a null-terminated string on the LCD screen.
|
| This function iterates through a null-terminated string and displays it
| on the LCD screen. If the character '\n' is encountered, the cursor is
| moved to the beginning of the next line.
|
| Parameters
|           [in]u8Ptr_a_str A pointer to the null-terminated string to be
|                          displayed.
| Return
|     void
|
void LCD_sendString(u8 * u8Ptr_a_str);



| Set the cursor position on the LCD.
|
| Parameters
|           [in]u8_a_line the line number to set the cursor to, either
|                          LCD_LINE0 or LCD_LINE1
|           [in]u8_a_col the column number to set the cursor to, from
|                          LCD_COL0 to LCD_COL15
| Return
|     STD_OK if the operation was successful, STD_NOK otherwise.
|
u8 LCD_setCursor(u8 u8_a_line, u8 u8_a_col);



| Stores a custom character bitmap pattern in the CGRAM of the LCD module
|
| Parameters
|           [in] u8_a_pattern Pointer to an array of 8 bytes representing
|                          the bitmap pattern of the custom character
|           [in] u8_a_location The CGRAM location where the custom
|                          character should be stored (from LCD_CUSTOMCHAR_LOC0 to 7)
| Return
|     STD_OK if successful, otherwise STD_NOK
|
u8 LCD_storeCustomCharacter(u8 * u8_a_pattern, u8 u8_a_location);
```

```
| Show/Hide cursor
|
| Parameters
|           [in]u8_a_show 0: hide, otherwise: show
|
void LCD_changeCursor(u8 u8_a_show);


| Shift clears the LCD display
void LCD_shiftClear(void);

| Clears the LCD display
void LCD_clear(void);
```

2.3.3.2. BTN APIs

```
| Initialize a GPIO pin as an input pin
|
| This function initializes a specified GPIO pin as an input pin using
|  the DIO_init() function.
|
| Parameters
|           [in]u8_a_pinNumber The pin number to be initialized (0-7).
|           [in]en_a_portNumber The port number to which the pin belongs
|                               (PORT_A, ..).
| Return
|     STD_OK if the pin initialization was successful, STD_NOK
|           otherwise.
|
u8 BTN_init(u8 u8_a_pinNumber, EN_DIO_PORT_T en_a_portNumber);



| This function reads the current state of a specified button by calling
|  the DIO_read() function.
|
| Parameters
|            [in]u8_a_btnId The ID of the button to read (BTN_U8_1 to BTN_U8_8).
|           [out]u8ptr_a_returnedBtnState A pointer to an 8-bit unsigned
|                                          integer where the button state will be stored.
| Return
|     STD_OK if the button state was read successfully, STD_NOK otherwise.
|
u8 BTN_getBtnState(u8 u8_a_btnId, u8 *u8ptr_a_returnedBtnState);
```

2.3.3.3. DCM APIs

```
| Initialize the DC Motors by initializing their pins.
|
| Parameters
|           u8_a_shutdownFlag Pointer to the Shutdown flag variable that
|                         acts as a main kill switch.
| Return
|     EN_DCM_ERROR_T Returns DCM_OK if initialization is successful, or
|                    DCM_ERROR if initialization failed.
|
EN_DCM_ERROR_T DCM_motorInit(u8 ** u8_a_shutdownFlag);



| Rotates the DC motor.
|
| This function rotates the DC motor 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.
|
| Return
|     EN_DCM_ERROR_T DCM_OK if the operation is successful, DCM_ERROR
|                    otherwise.
|
EN_DCM_ERROR_T DCM_rotateDCM(void);



| Changes the direction of the motor rotation for the specified motor.
|
| Parameters
|           en_a_motorNum The motor number whose direction needs to be changed.
| Return
|     EN_DCM_ERROR_T DCM_OK if the operation is successful, DCM_ERROR otherwise.
|
EN_DCM_ERROR_T DCM_changeDCMDirection(EN_DCM_MOTORSIDE en_a_motorNum);
```

```
| Sets the duty cycle of the PWM for the motor.
|
| This function sets the duty cycle of the PWM for the motor. The duty
|  cycle value
| provided should be between 0 and 100, where 0 indicates a duty cycle of
|  0% and 100
| indicates a duty cycle of 100%.
|
| Parameters
|            u8_a_dutyCycleValue The duty cycle value for the motor.
| Return
|      EN_DCM_ERROR_T The error status of the function.
|      - DCM_OK: The function executed successfully.
|      - DCM_ERROR: The duty cycle value provided was out of range.
|
EN_DCM_ERROR_T DCM_setDutyCycleOfPWM(u8 u8_a_dutyCycleValue);

| Stops the DC motors by setting the PWM output pins to low and resetting
|  the stop flag.
|
void DCM_vdStopDCM(void);

| Updates the stop flag.
|
| This function is called by the timer overflow callback function to
|  update the stop flag.
| It sets the `en_g_stopFlag` variable to TRUE, which is used by other
|  functions to stop the
| motor movement.
|
void DCM_updateStopFlag(void);
```

2.3.3.4. KPD APIs

```
| Initializes the KPD module.
|
| This function initializes the pins of a keypad by setting some as output and others
| as input.
|
| Return
|     void
|
void KPD_initKPD(void);


| Enables or Re-enables the KPD module.
|
| This function enables or re-enables a keypad by setting one pin as output and the
| other three as input.
|
| Return
|     void
|
void KPD_enableKPD(void);


| Disables the KPD module.
|
| This function disables the keypad by setting its output pins to input.
|
| Return
|     void
|
void KPD_disableKPD(void);


| This function reads input from a keypad and returns the pressed key value after
| debouncing.
|
| Parameters
|  -         [in] pu8_a_returnedKeyValue Pointer to a u8 variable that will hold the
|                 value of the pressed key.
| Return
|     STD_OK if successful, otherwise STD_NOK
|
u8 KPD_getPressedKey(u8 *pu8_a_returnedKeyValue);
```

2.3.3.5. Ultrasonic APIs

```
| Initializes the ultrasonic driver
|
| Parameters
|           [in]u8_a_triggerPin I/O pin number to send trigger signal
|           [in]u8_a_echoPin I/O pin number to receive echoed signal
|
| Return
|     None
|
void US_init(u8 u8_a_triggerPin, u8 u8_a_echoPin);

| Initiates a get distance request
|
| This function sends a signal out to the trigger pin, waits for echo signal
| to come back and finally calculates the distance the signal traveled using
| the elapsed time duration used by the signal to arrive back on the echo pin
|
| Return
|     float distance (in mm)
|
float US_getDistance(void);

| Event Handler: called when echo time is received from ICU (input capture unit)
|
| This function is called by the ICU as an event callback when the trigger signal
| is received back on the echo pin, the function receives the elapsed time taken.
|
| Parameters
|      [in]u8_a_timeElapsed time elapsed (duration) by trigger signal to echo back
|
| Return
|     None
|
void US_evtEchoTimeReceived(u8 u8_a_timeElapsed);
```

2.3.5. APP APIs

```
| Initializes the application by initializing MCAL and HAL.
| This function initializes the General Interrupt Enable (GIE), sets up
|  callback functions
|
| Return
|     None
|
void APP_initialization(void);



| This function starts the car program and keeps it running indefinitely.
| The function uses a while loop to continuously check for the required
|  app mode.
| The app mode is checked using a switch statement, which contains
|  various cases that correspond to the different modes of operation for the car
|  program.
| Each case contains a series of steps to be executed to perform the desired
|  action for that mode.
|
| Return
|     void
|
void APP_startProgram(void);


| Used to switch between app states to initialize standard UI elements
| before main app flow (loop)
|
| Parameters
|           [in] u8_a_state state to set (APP_STATE_LAUNCH, APP_STATE_...)
|
| Return
|     void
|
void APP_switchState(u8 u8_a_state);
```

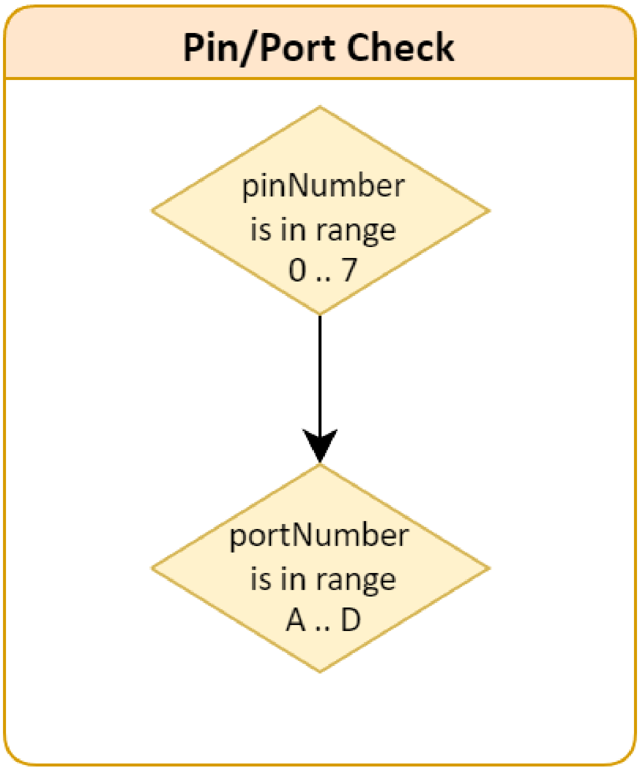
3. Low Level Design

3.1. MCAL Layer

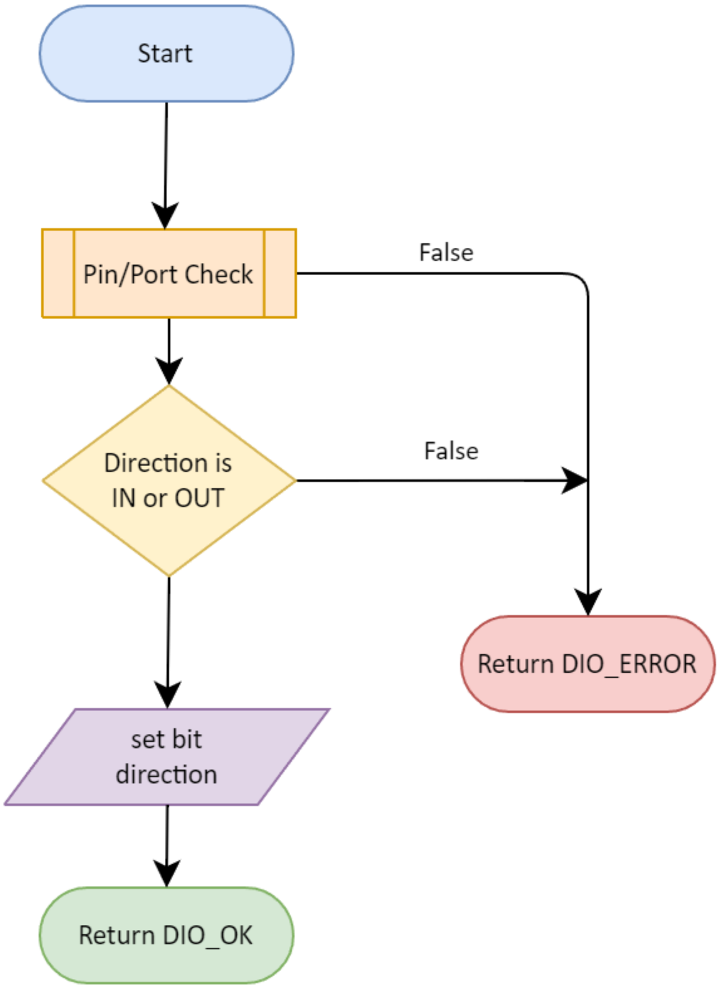
3.1.1. DIO Module

3.1.1.a. sub process

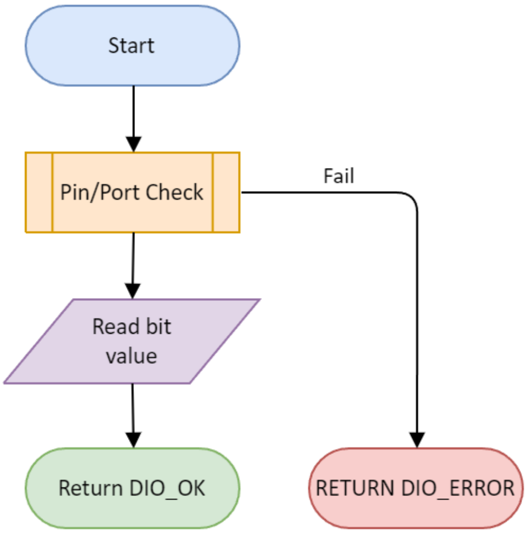
The following Pin/Port check subprocess is used in some of the DIO APIs flowcharts



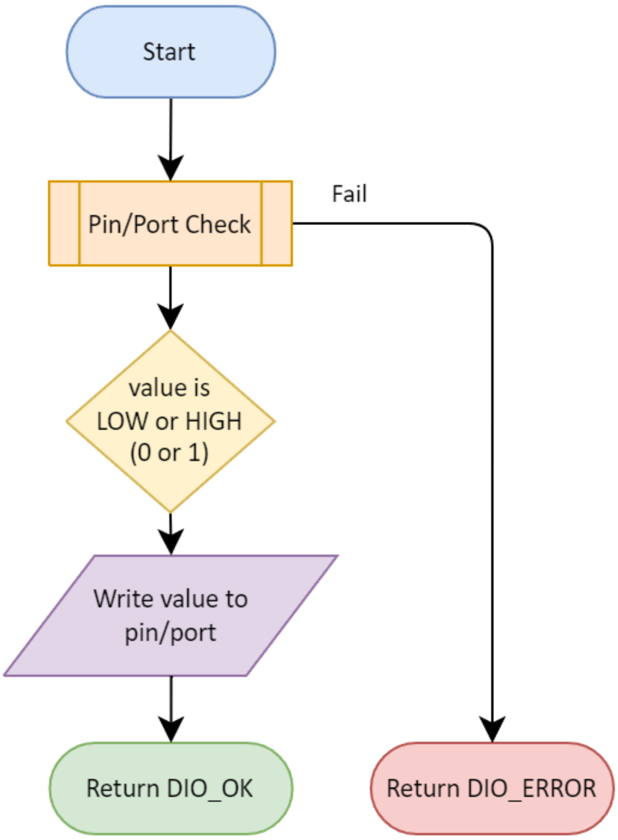
3.1.1.1. DIO_init



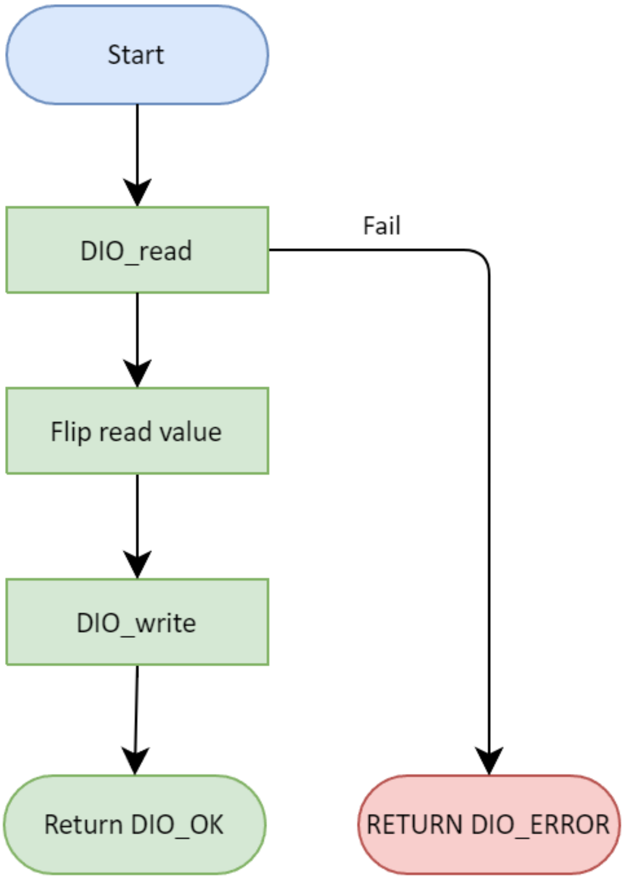
3.1.1.2. DIO_read



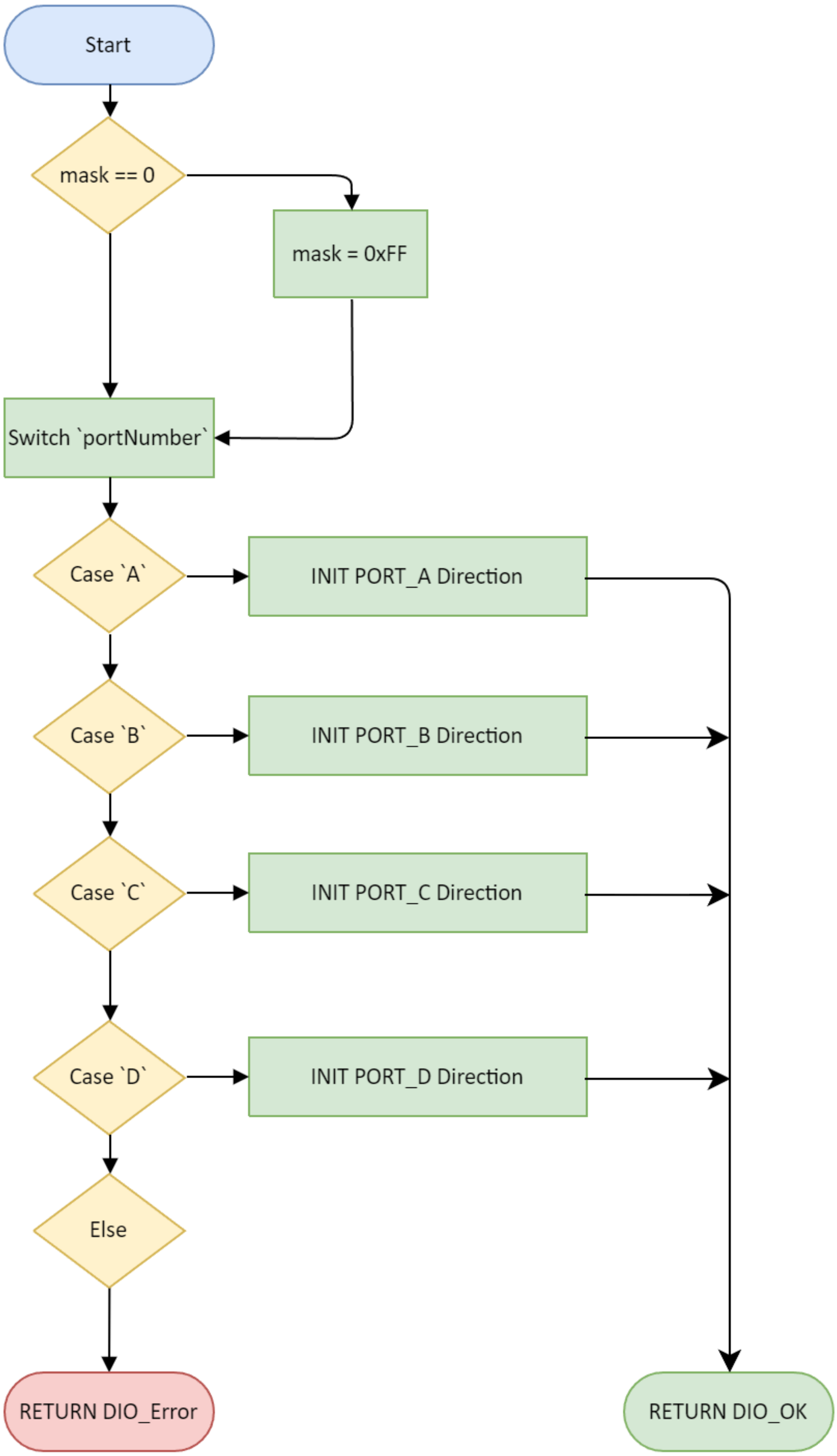
3.1.1.3. DIO_write



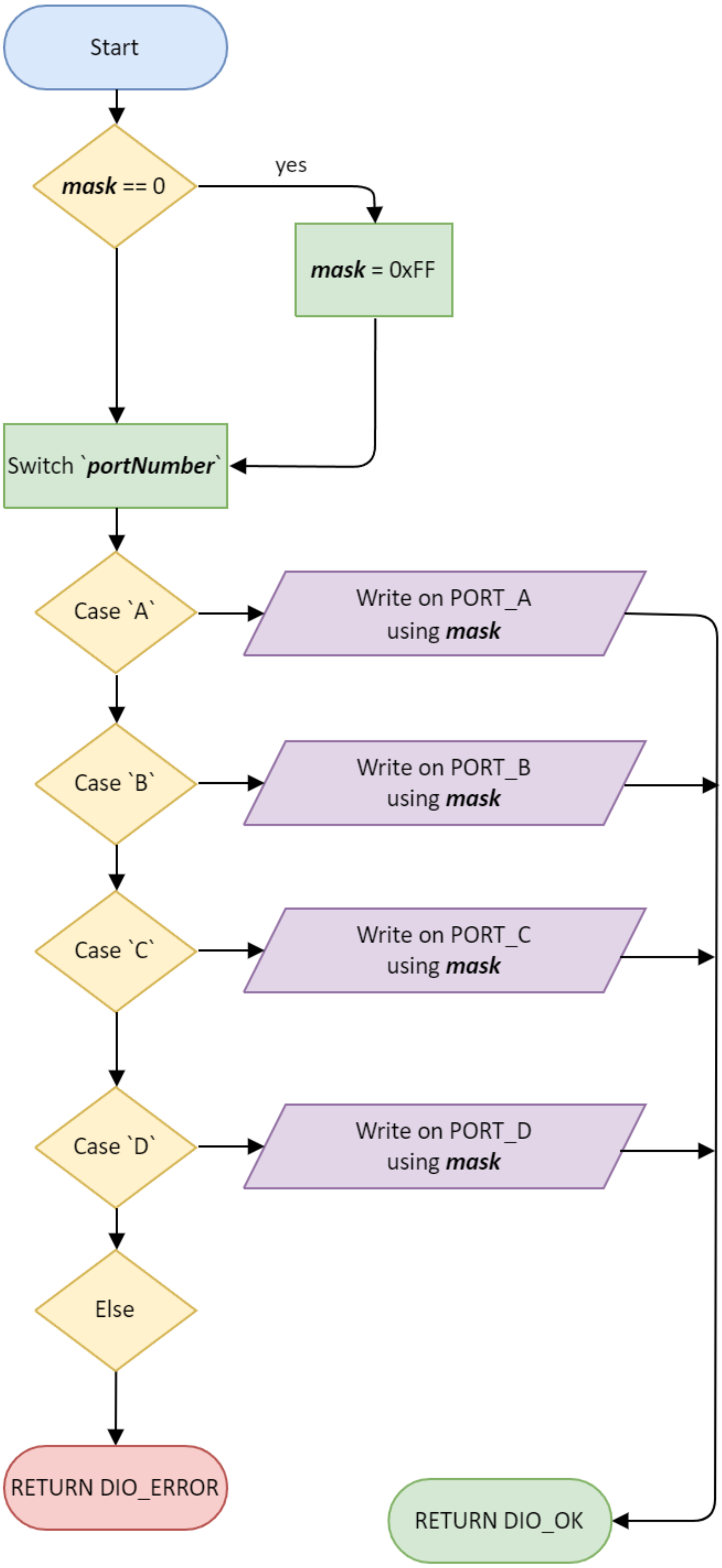
3.1.1.4. DIO_toggle



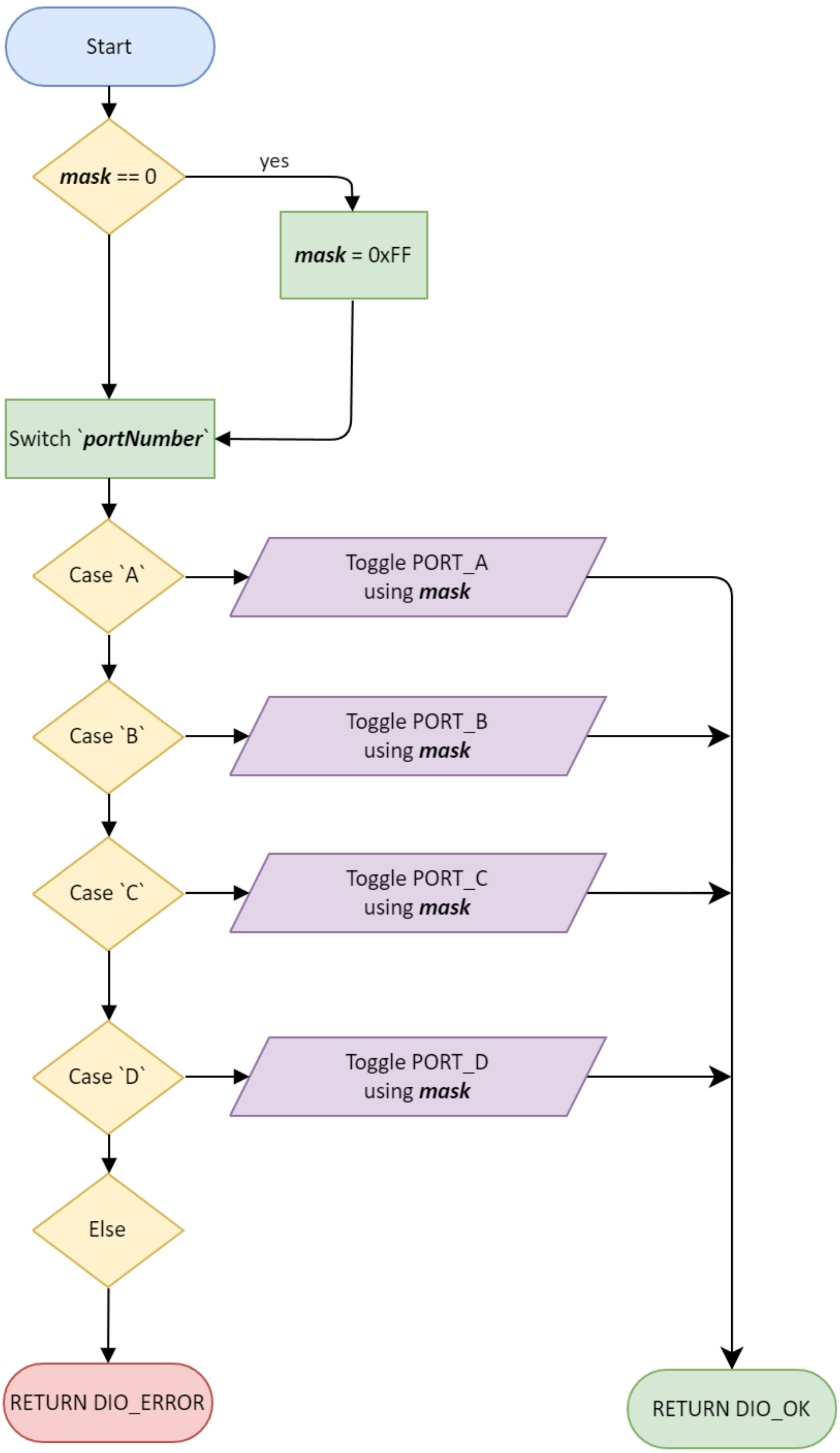
3.1.1.5. DIO_portInit



3.1.1.6. DIO_portWrite

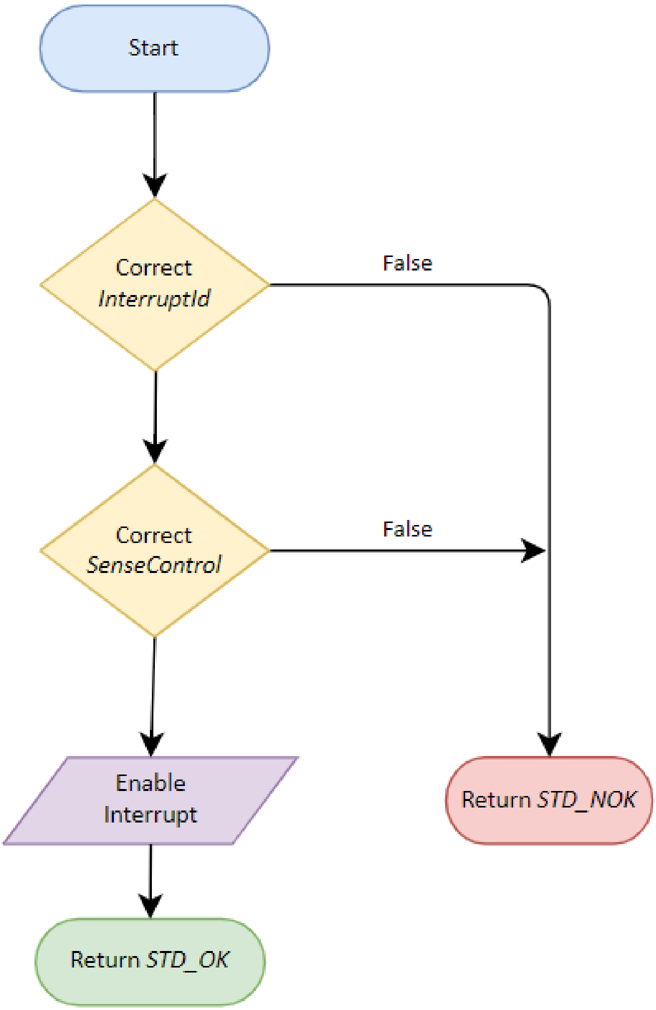


3.1.1.7. DIO_portToggle

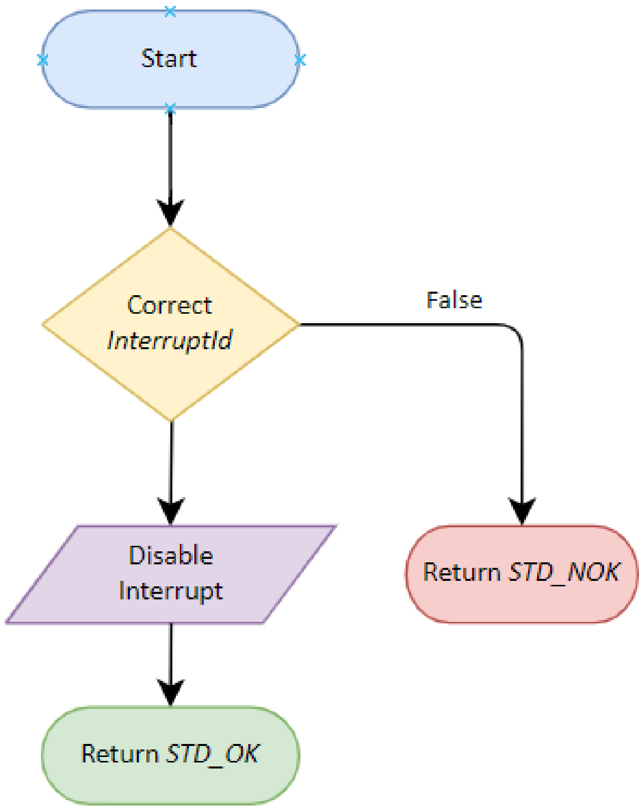


3.1.2. EXI Module

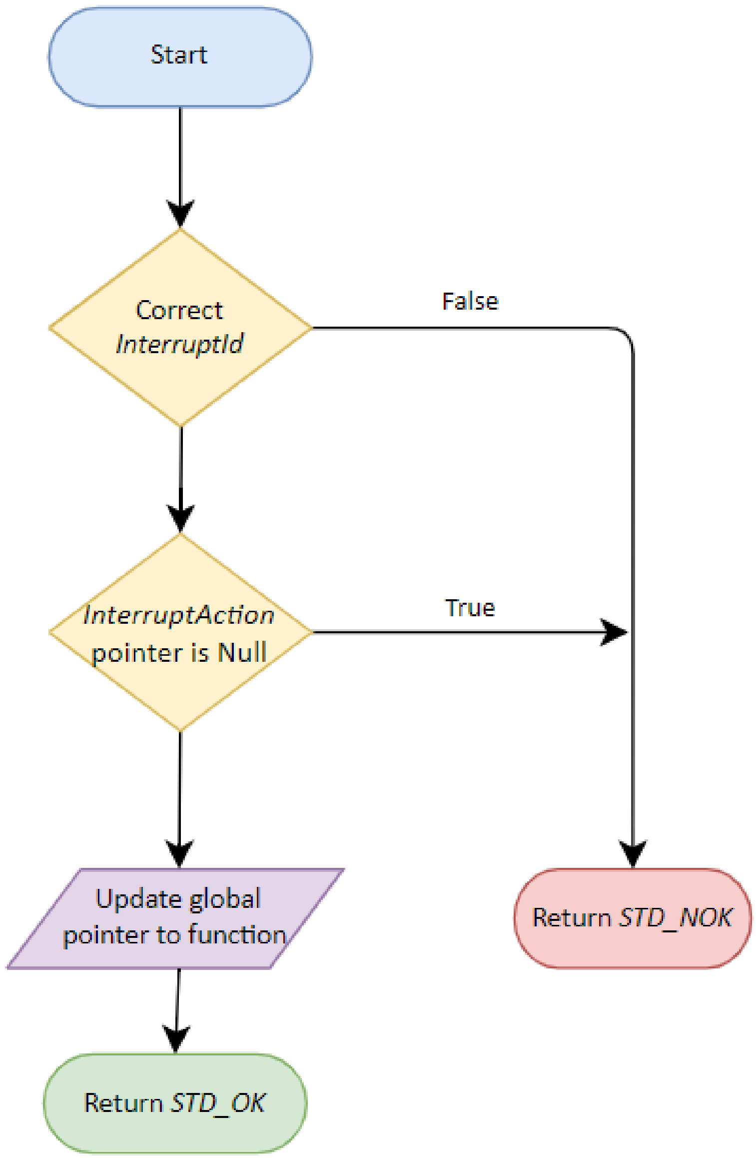
3.1.2.1. EXI_enablePIE



3.1.2.2. EXI_disablePIE

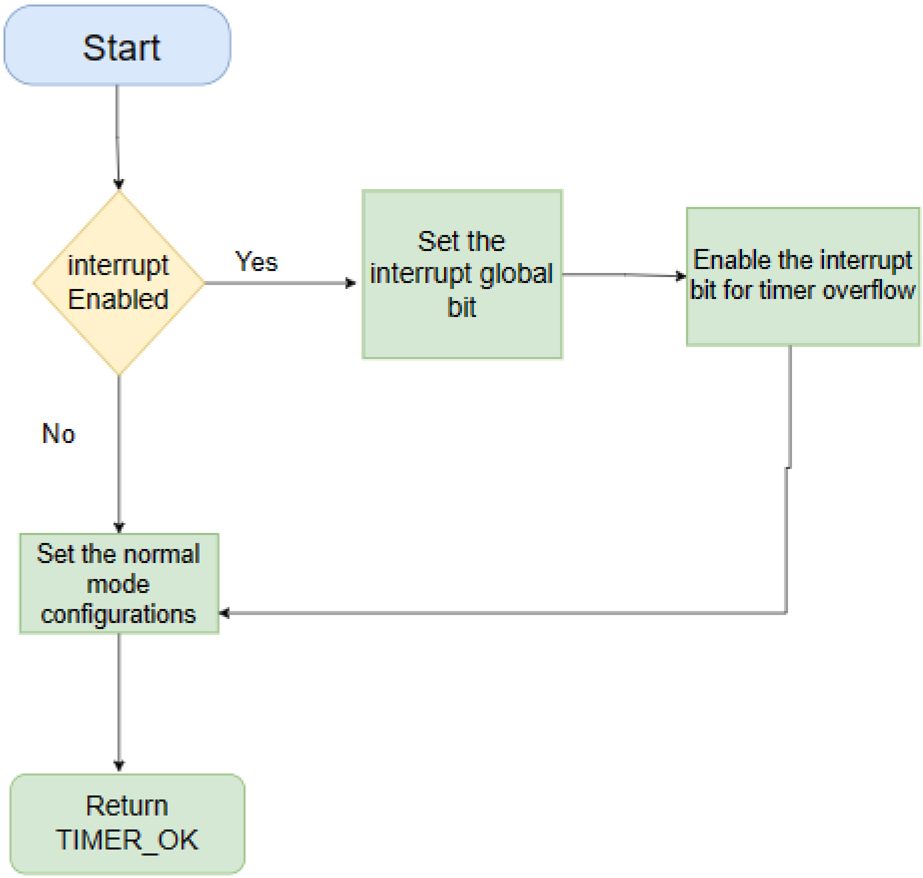


3.1.2.3. EXI_intSetCallback

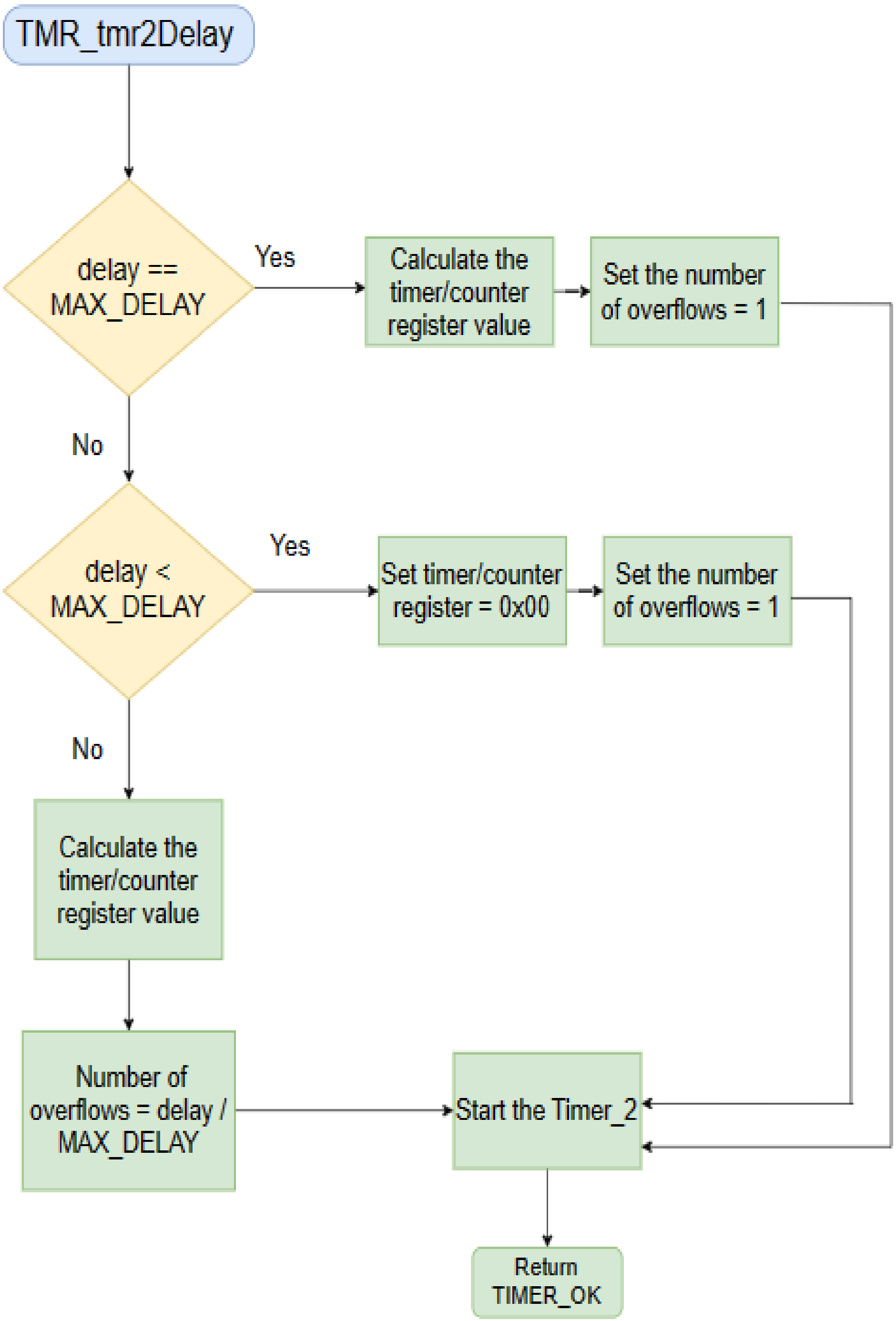


3.1.3. Timer Module

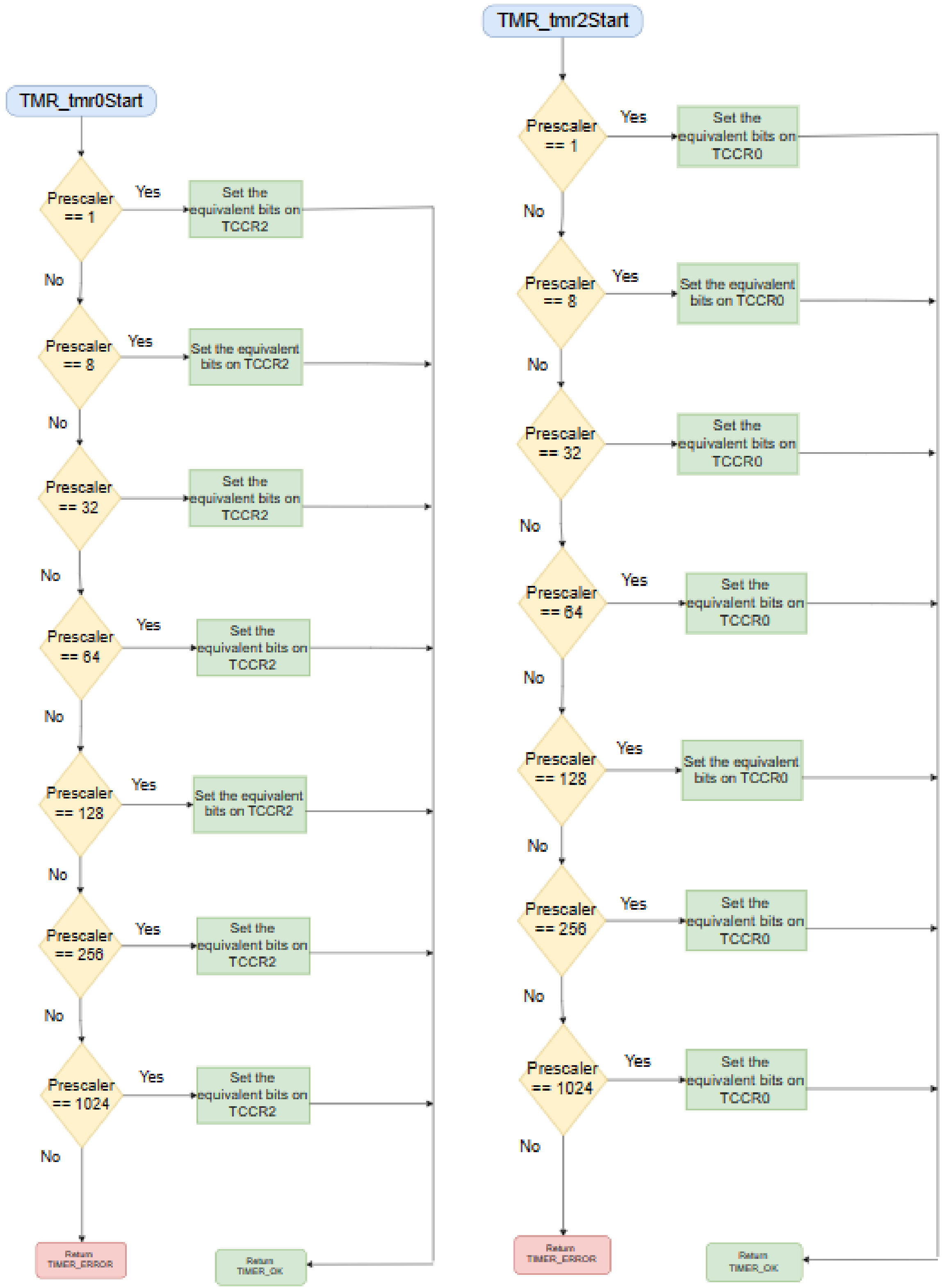
3.1.3.1. TMR_tmr0NormalModeInit / TMR_tmr2NormalModeInit



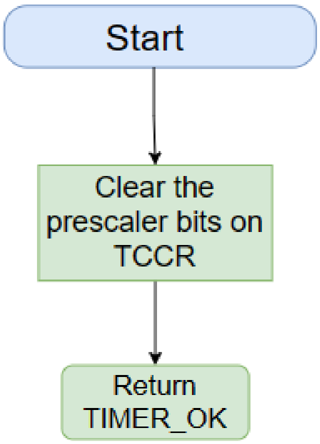
3.1.3.2. TMR_tmr0Delay / TMR_tmr2Delay



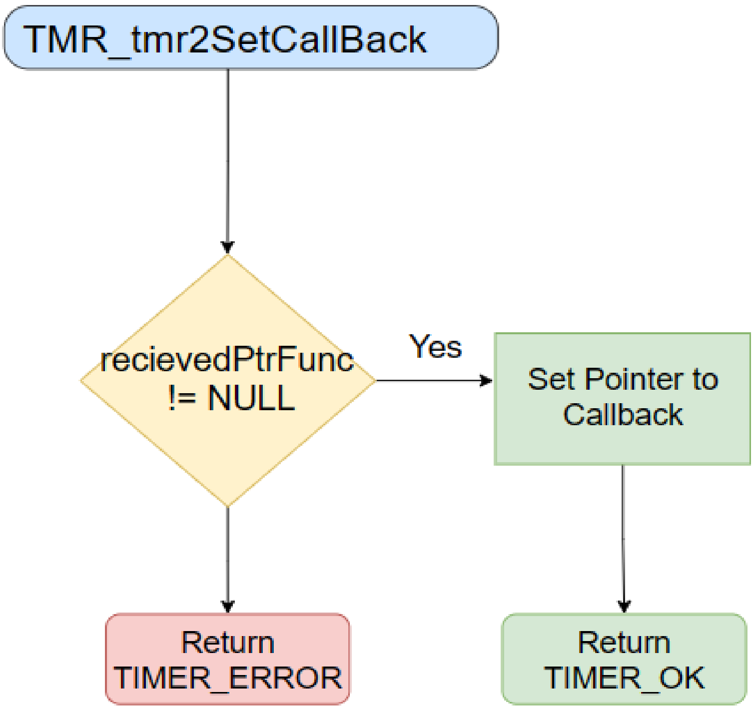
3.1.3.3. TMR_tmr0Start / TMR_tmr2Start



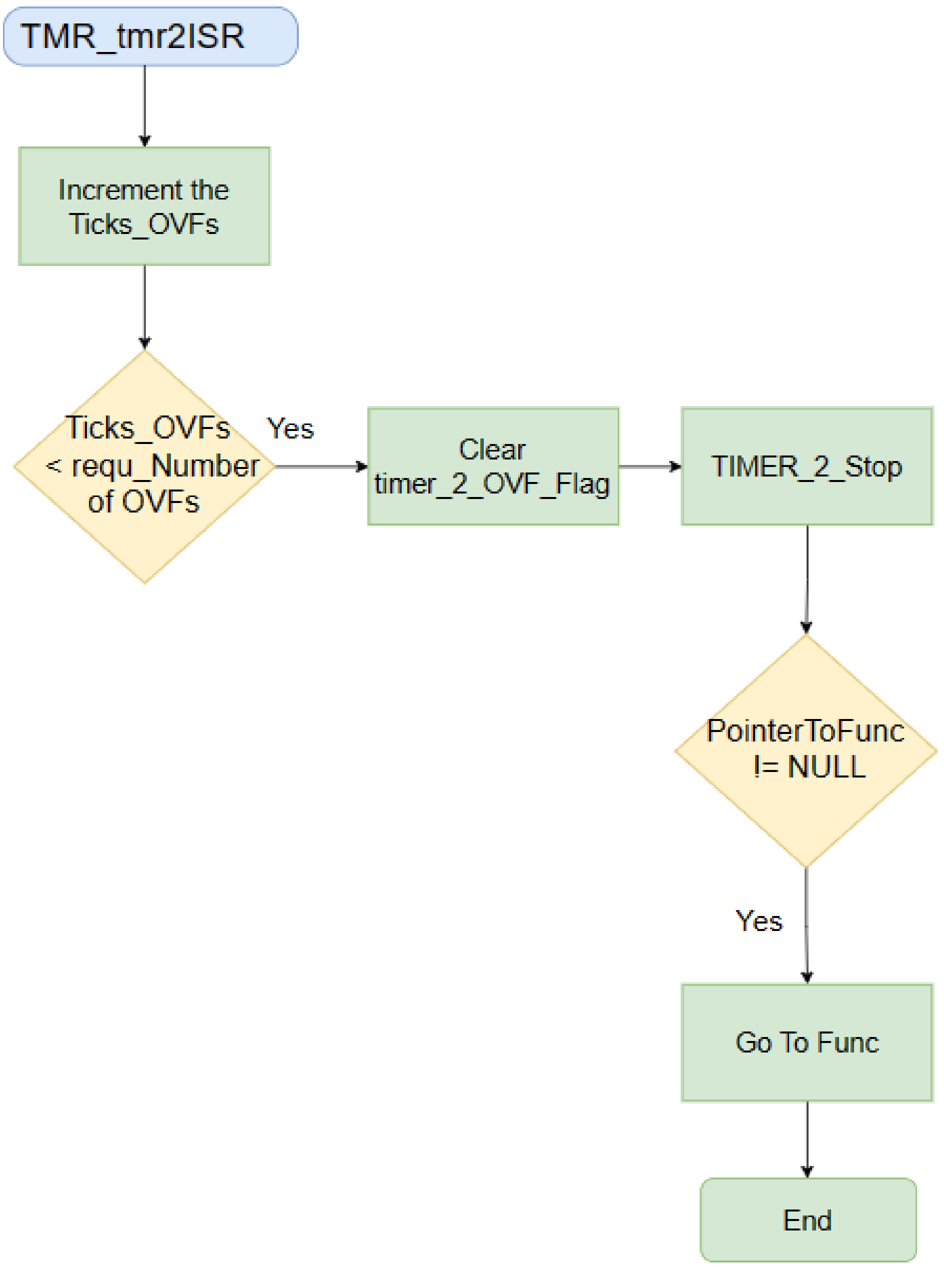
3.1.3.4. TMR_tmr0Stop / TMR_tmr2Stop



3.1.3.5. TMR_ovfSetCallback

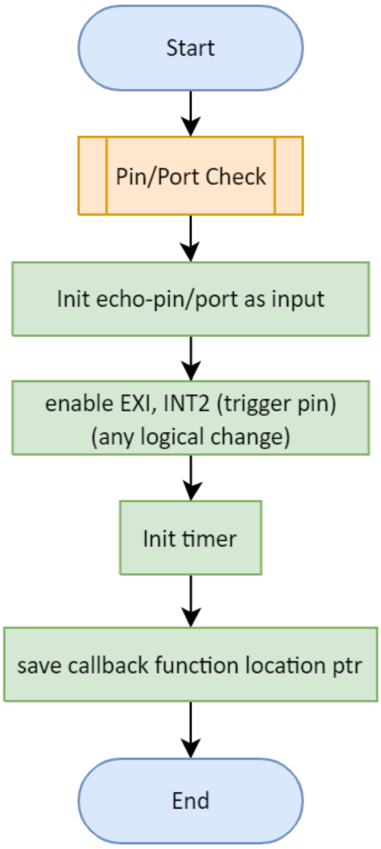


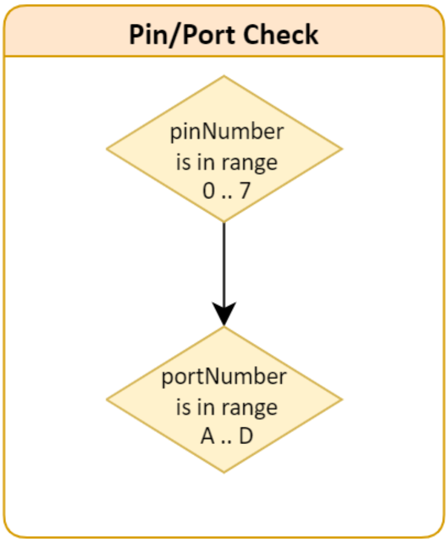
3.1.3.6. TMR_ovfVect



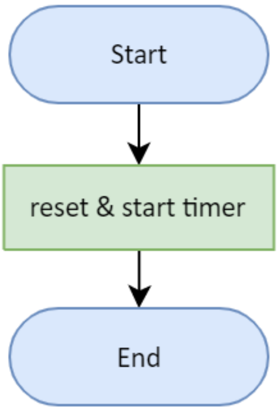
3.1.4. ICU Module (Input Capture Unit)

3.1.4.1. ICU_init

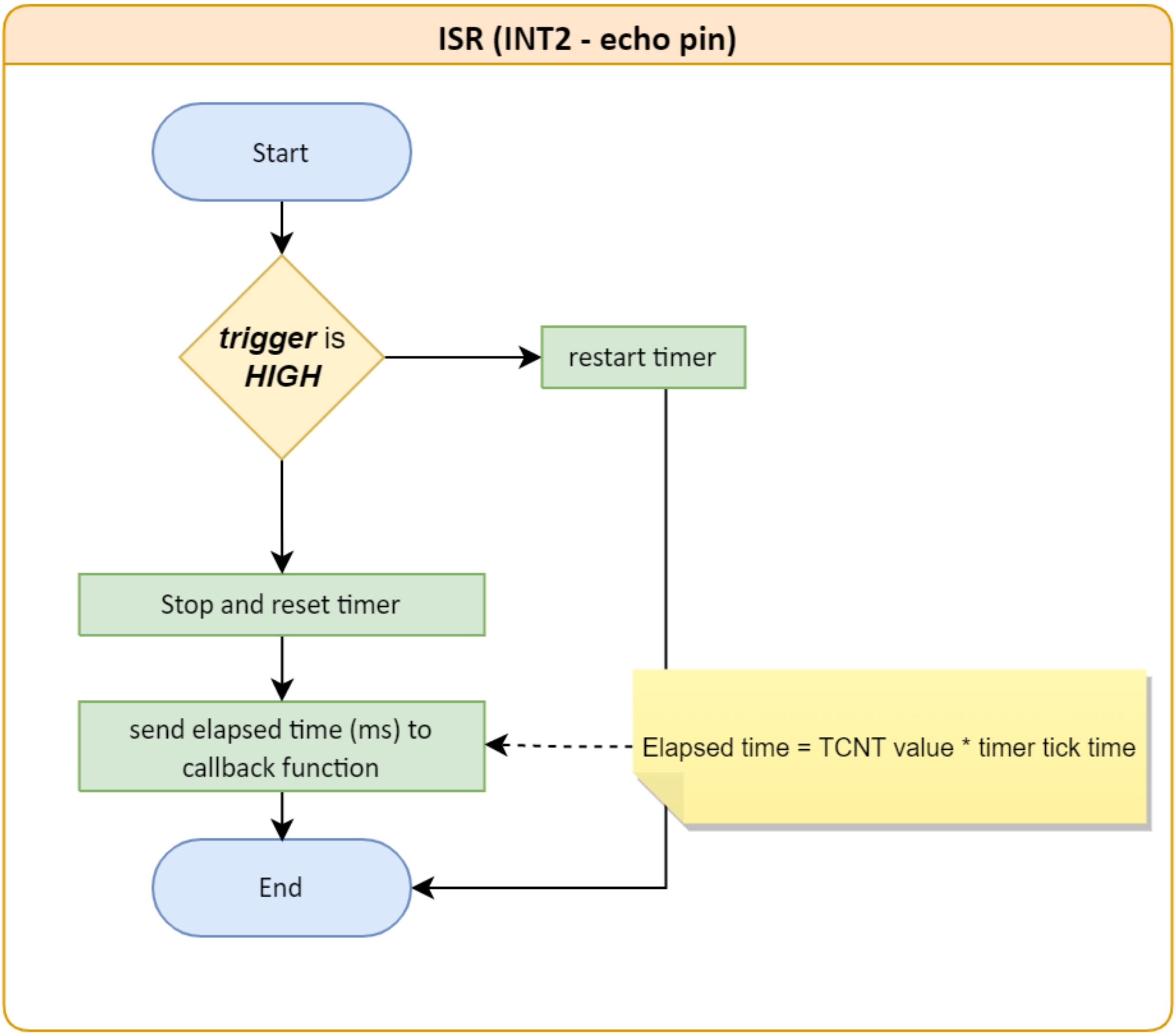




3.1.4.2. ICU_getCaptureValue



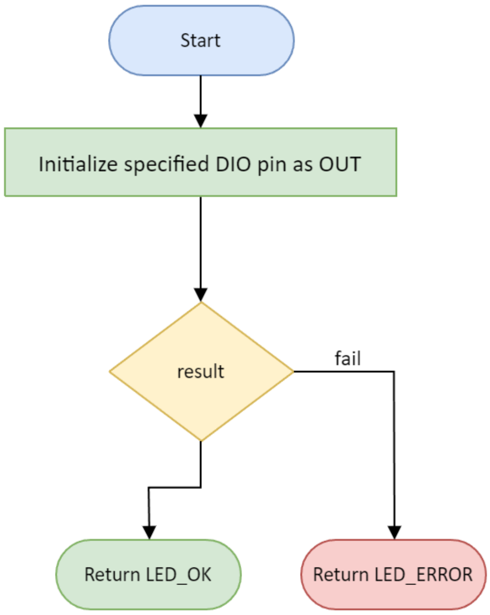
3.1.4.3. ISR (INT2 - Echo pin)



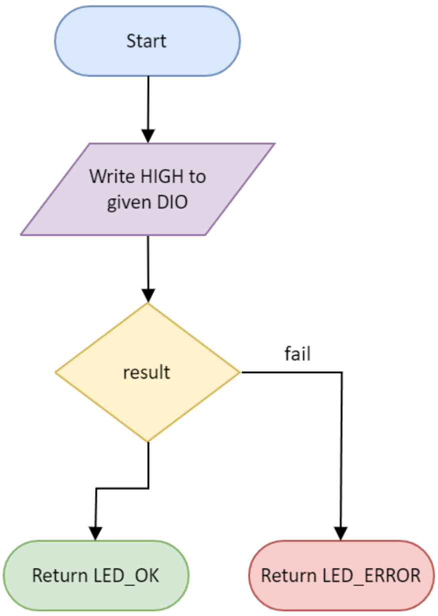
3.2. HAL Layer

3.2.1. LED Module

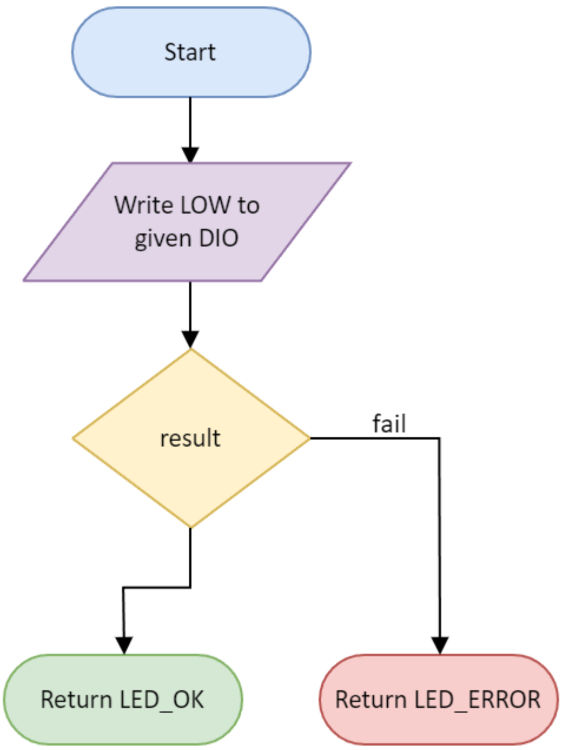
3.2.1.1. LED_init



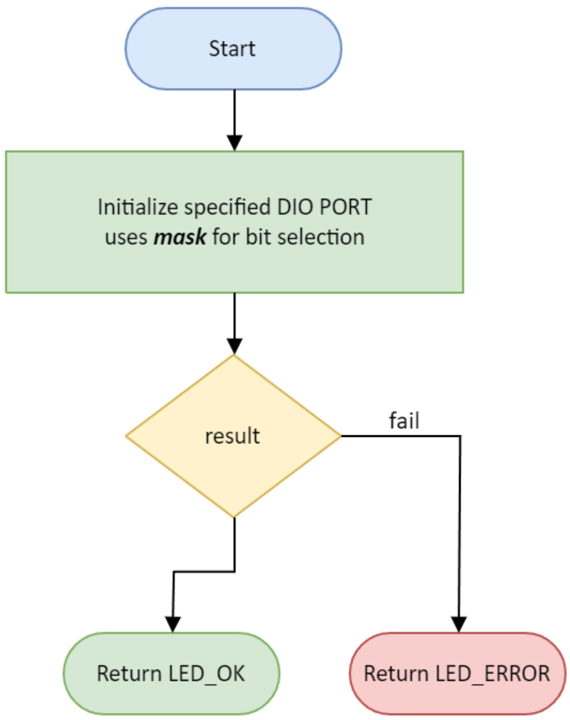
3.2.1.2. LED_on



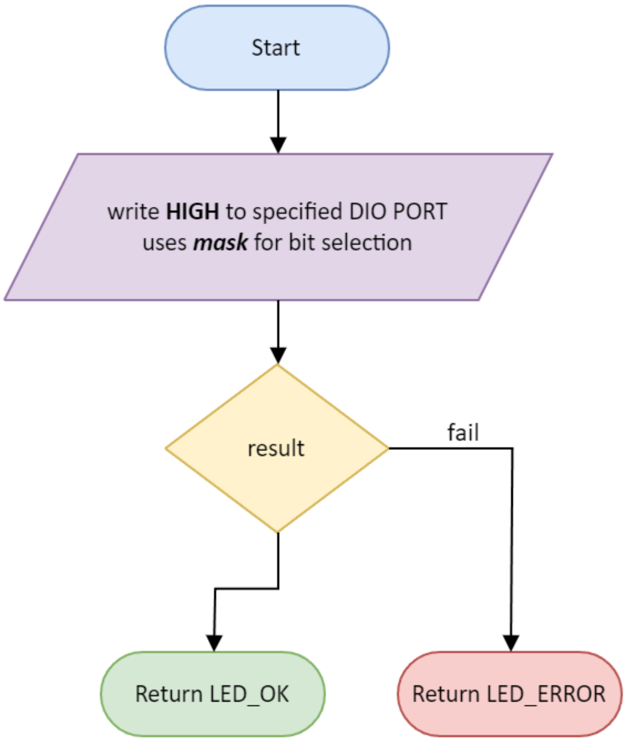
3.2.1.3. LED_off



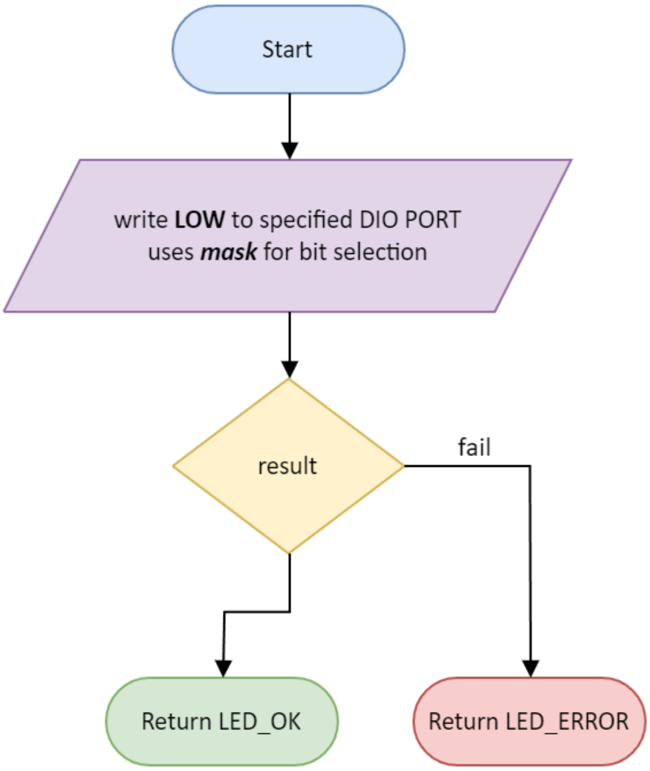
3.2.1.4. LED_arrayInit



3.2.1.5. LED_arrayOn

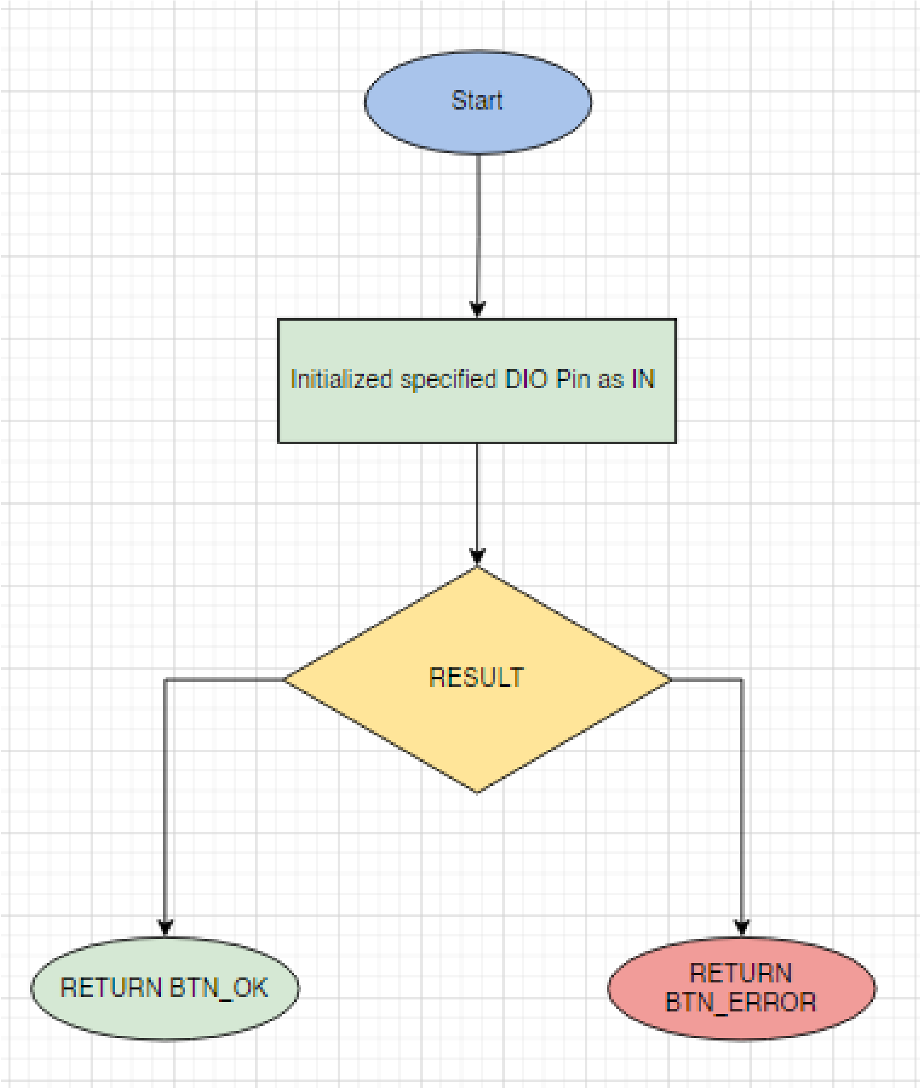


3.2.1.6. LED_arrayOff

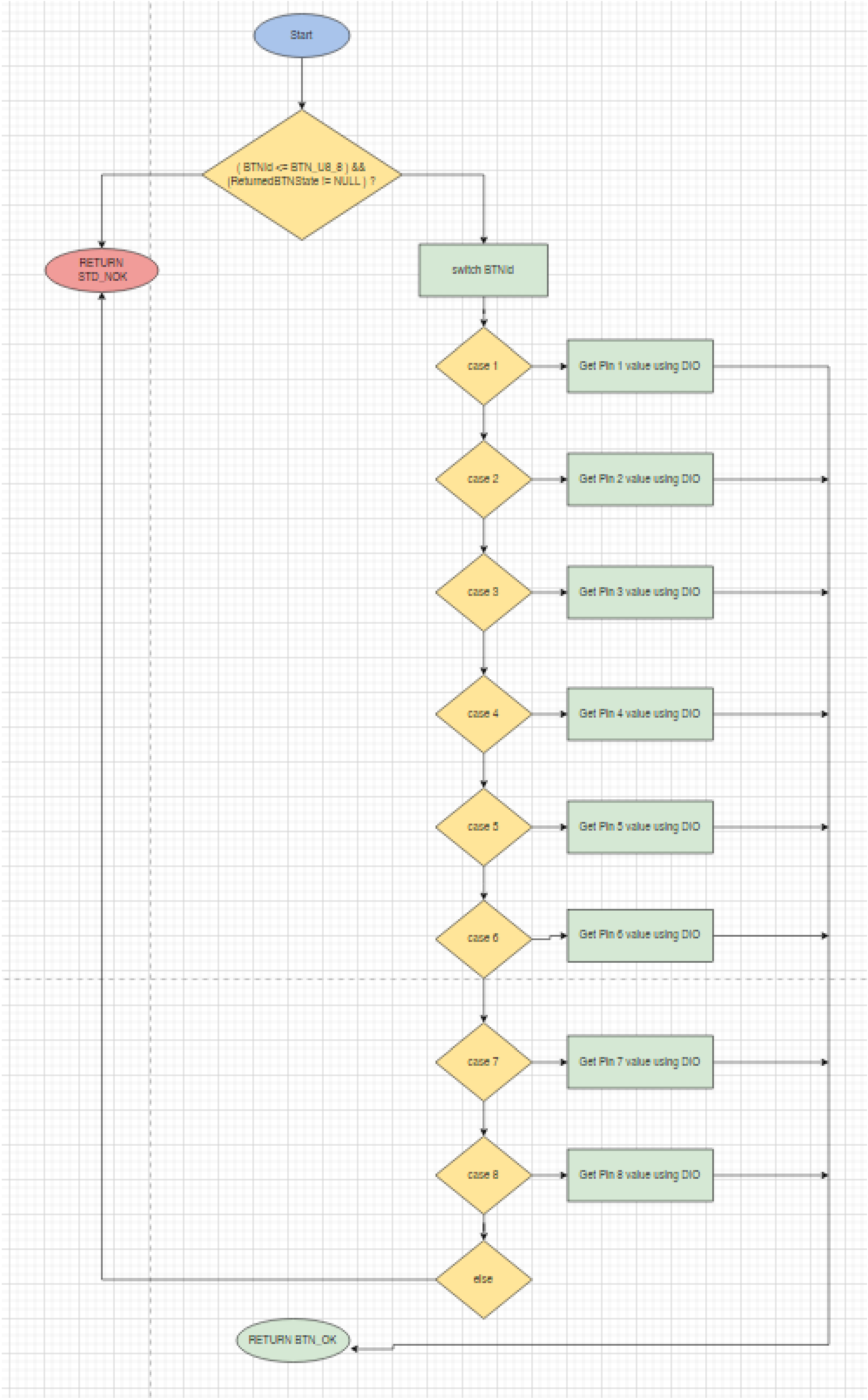


3.2.2. BTN Module

3.2.2.1. BTN_init

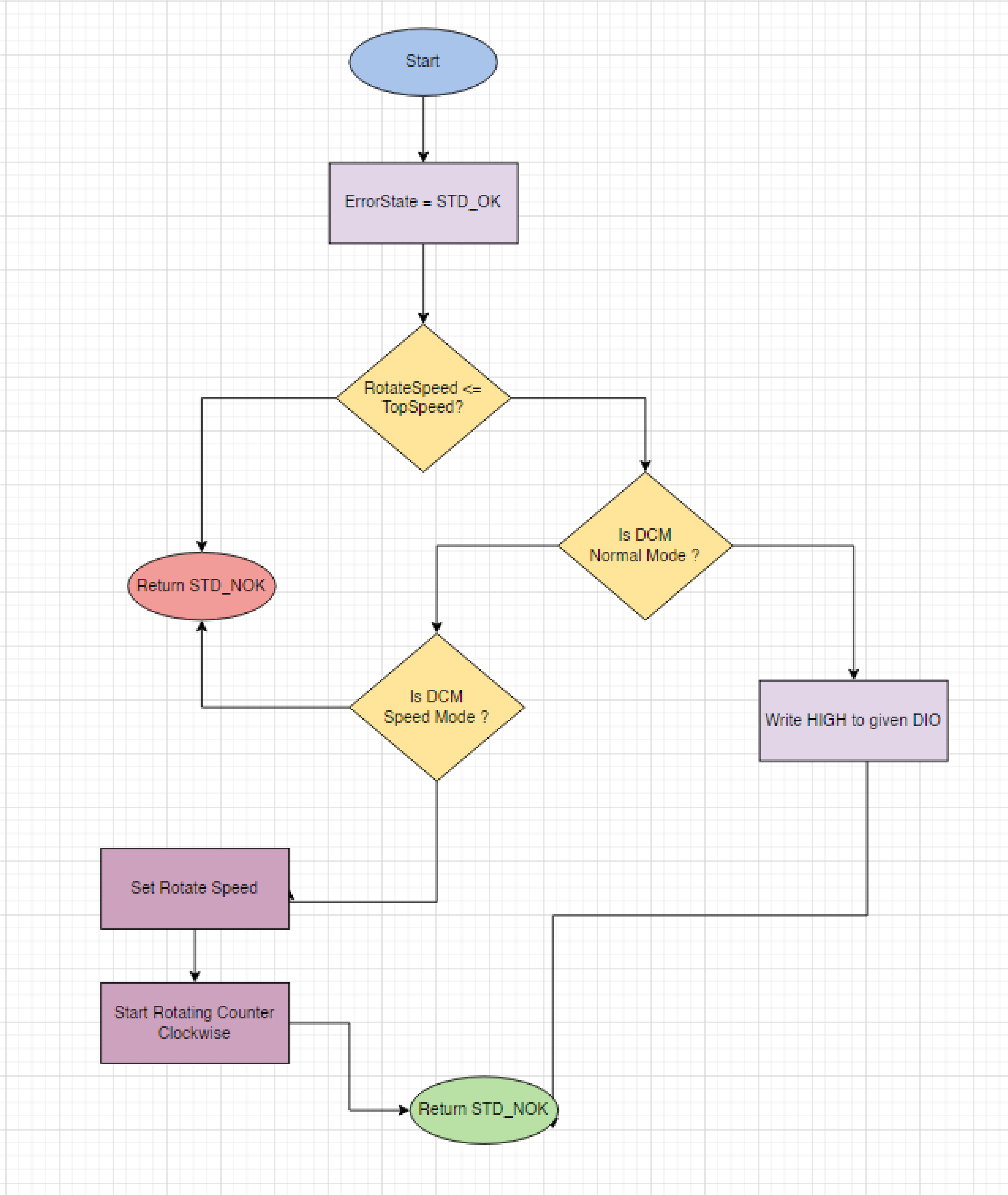


3.2.3.2. BTN_getBTNState

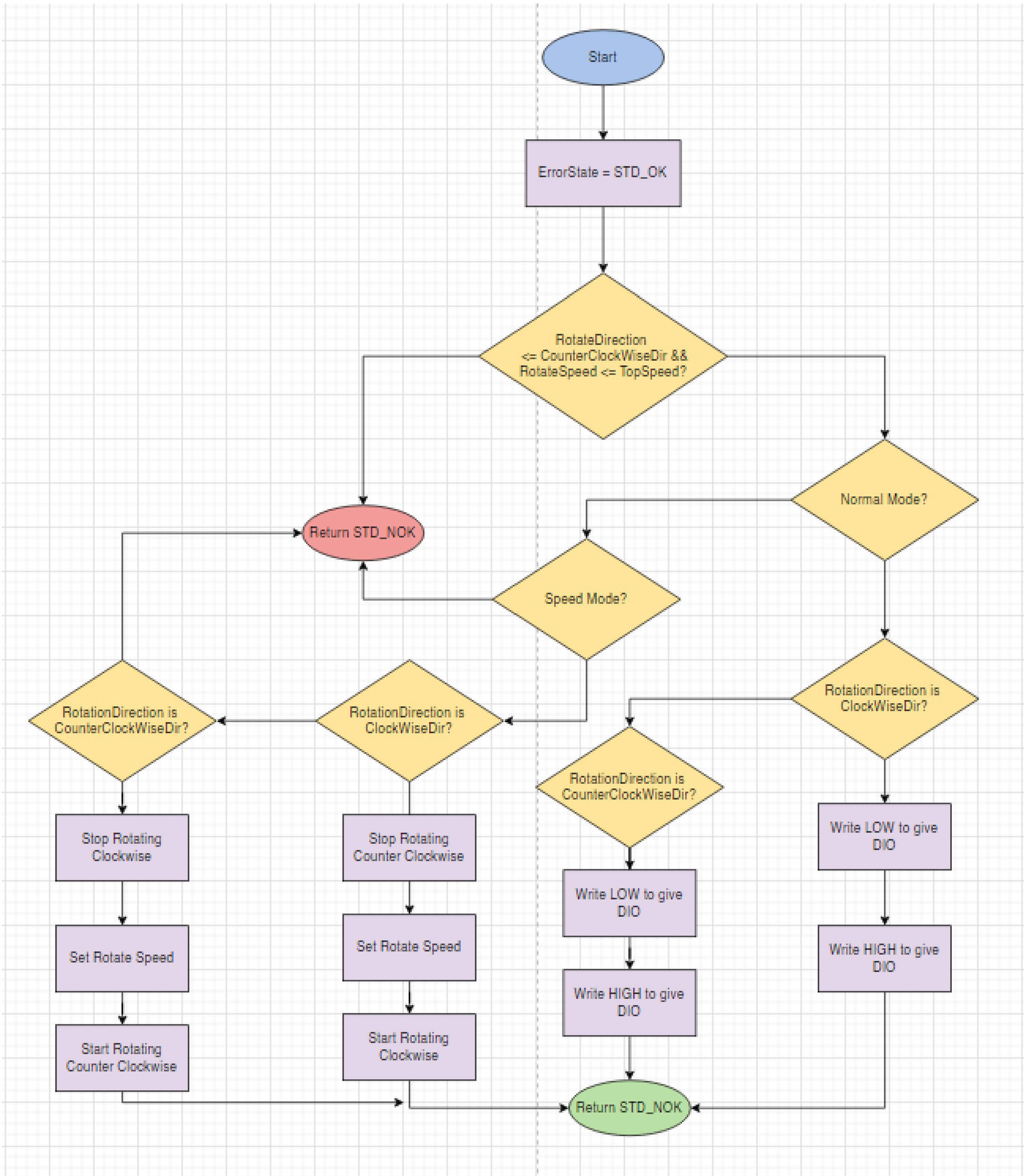


3.2.3. DCM Module

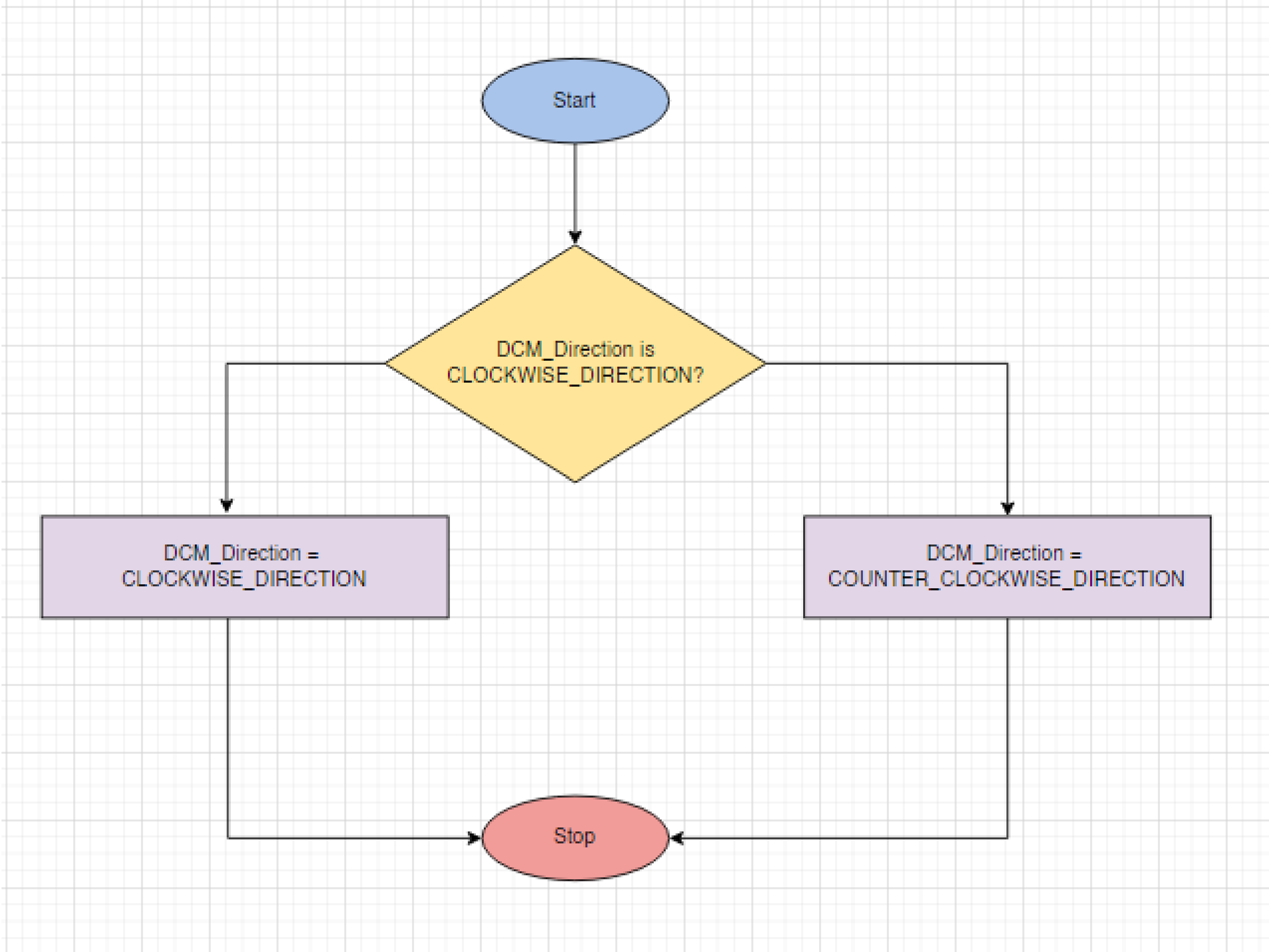
3.2.3.1. DCM_rotateDCMInOneDirection



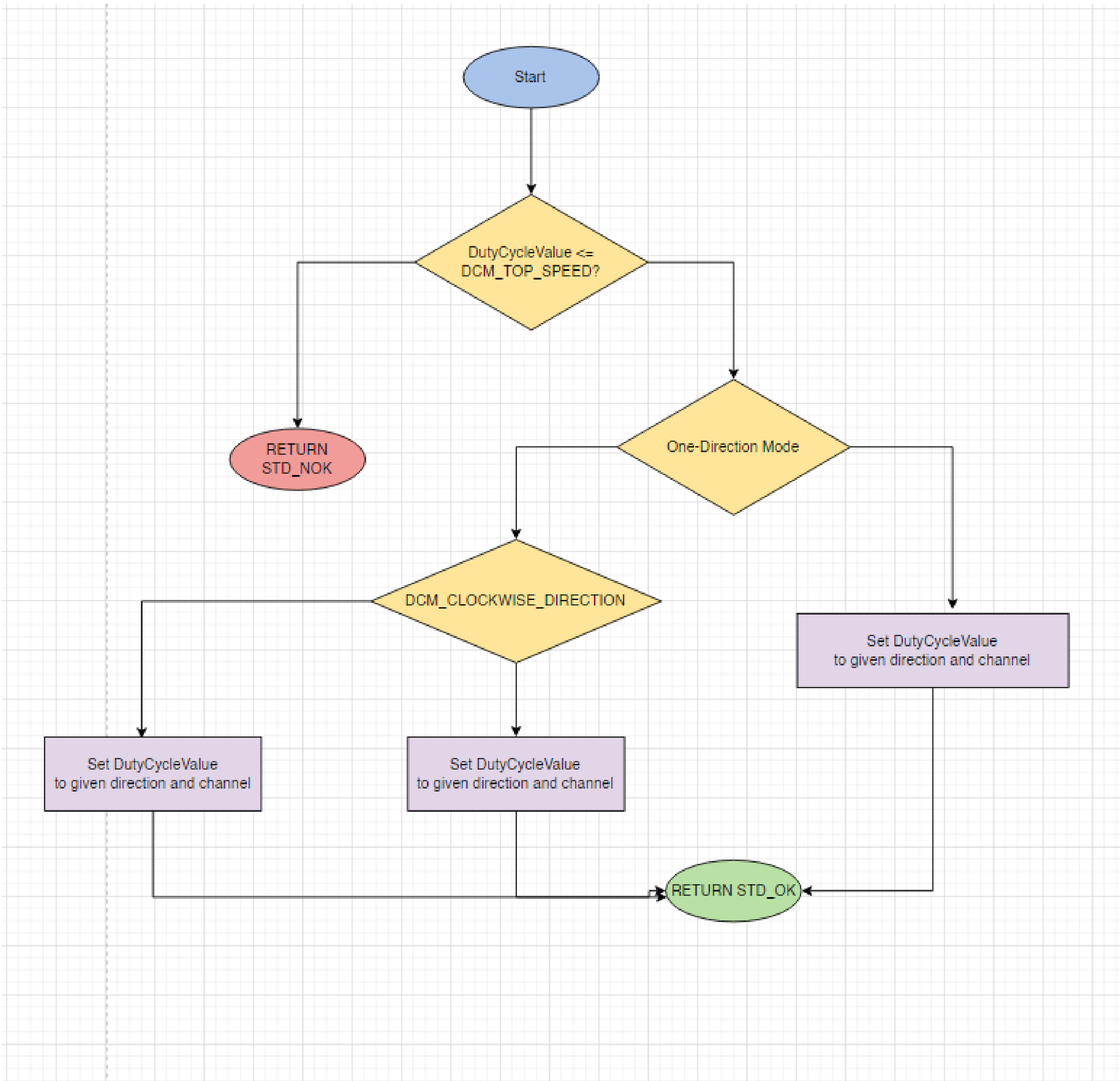
3.2.3.2. DCM_rotateDCMInTwoDirections



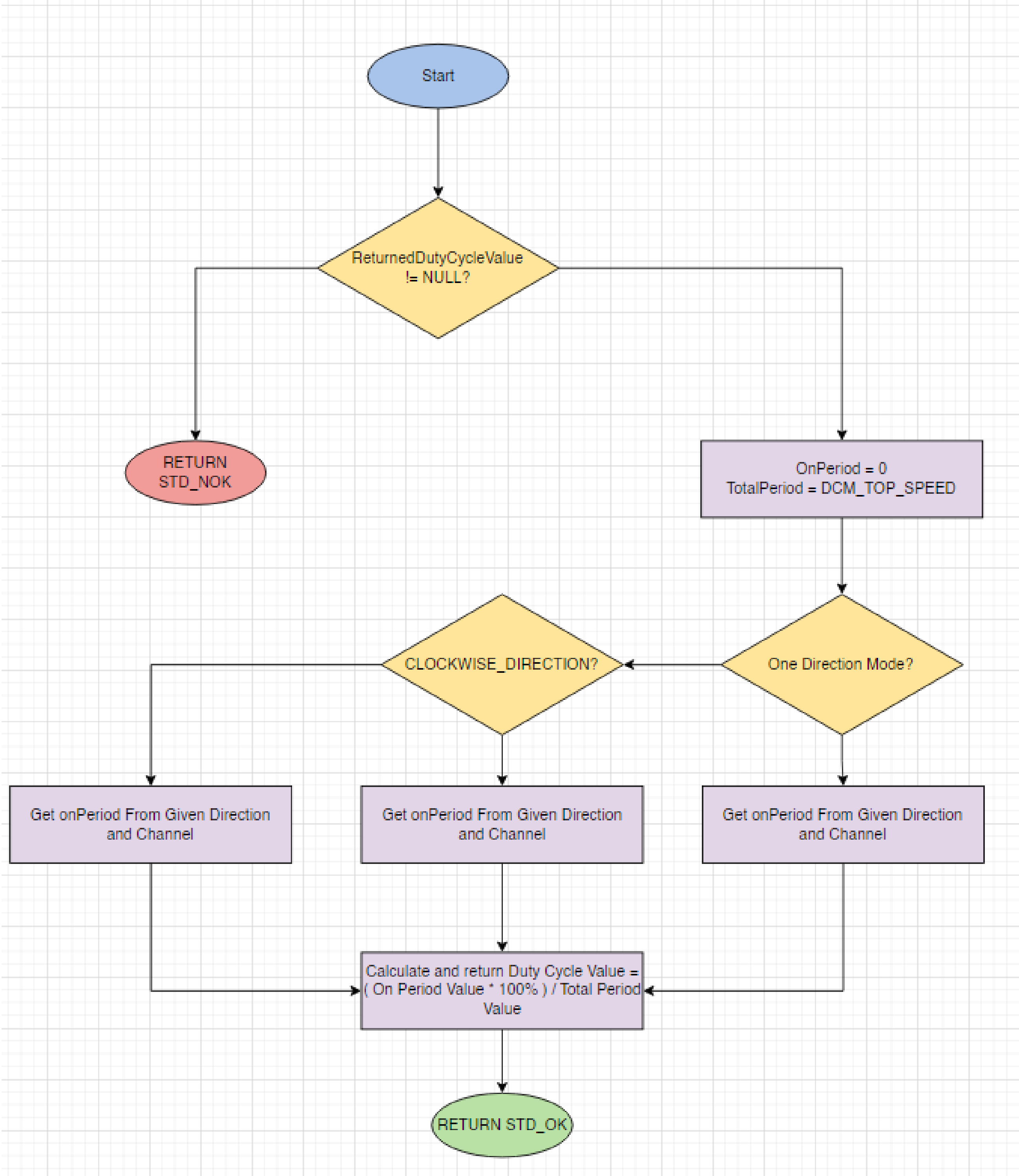
3.2.3.3. DCM_changeDCMDirection



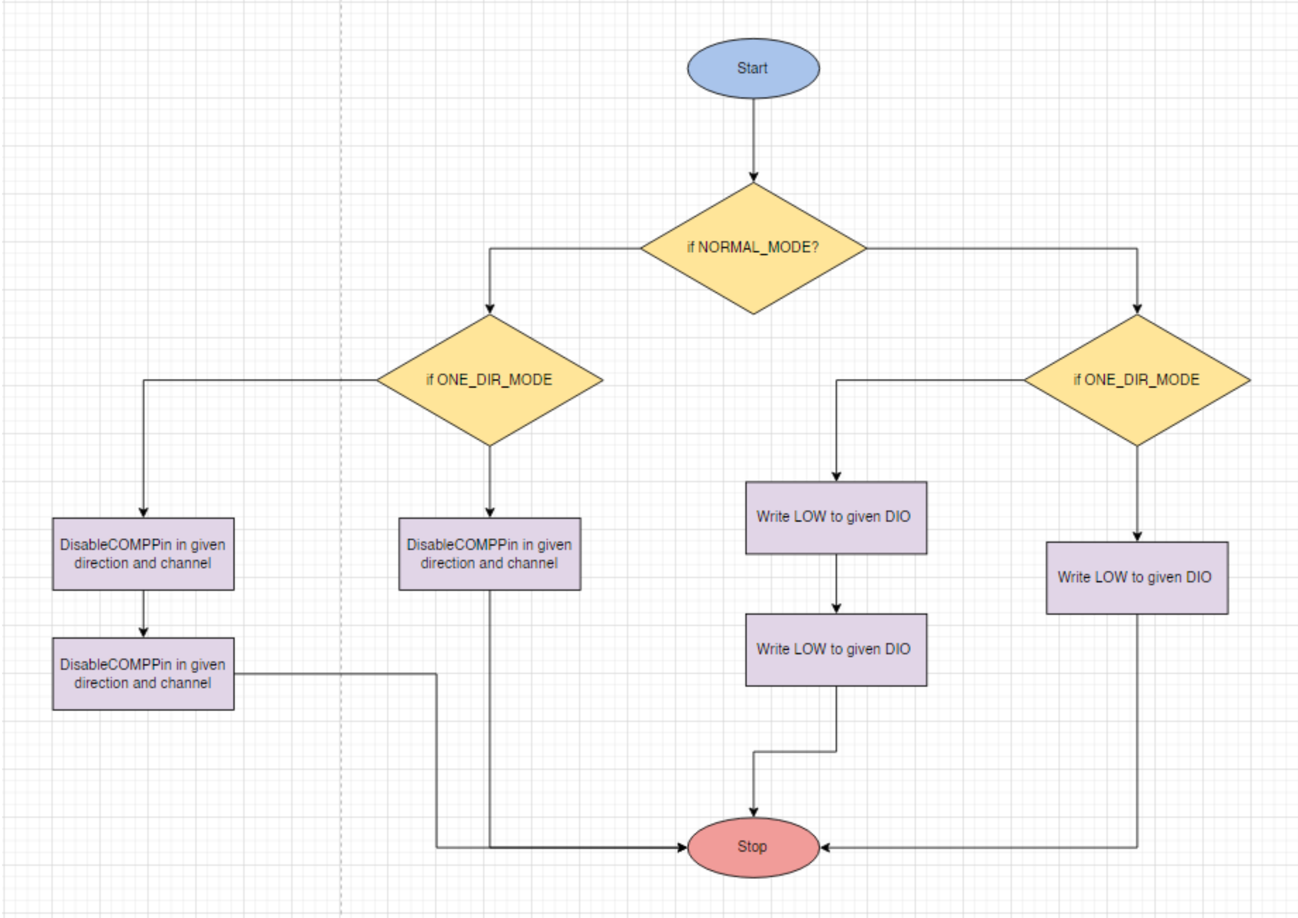
3.2.3.4. DCM_setDutyCycleOfPWM



3.2.3.5. DCM_getDutyCycleOfPWM

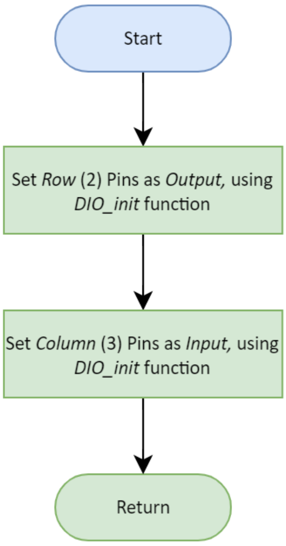


3.2.3.6. DCM_stopDCM

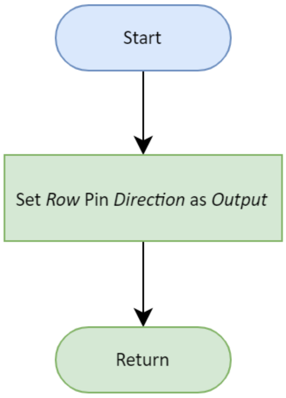


3.2.4. KPD Module

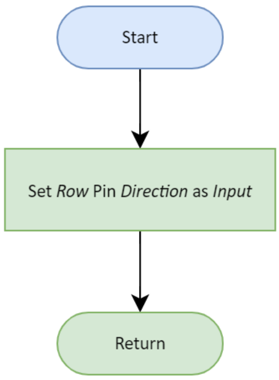
3.2.4.1. KPD_initKPD



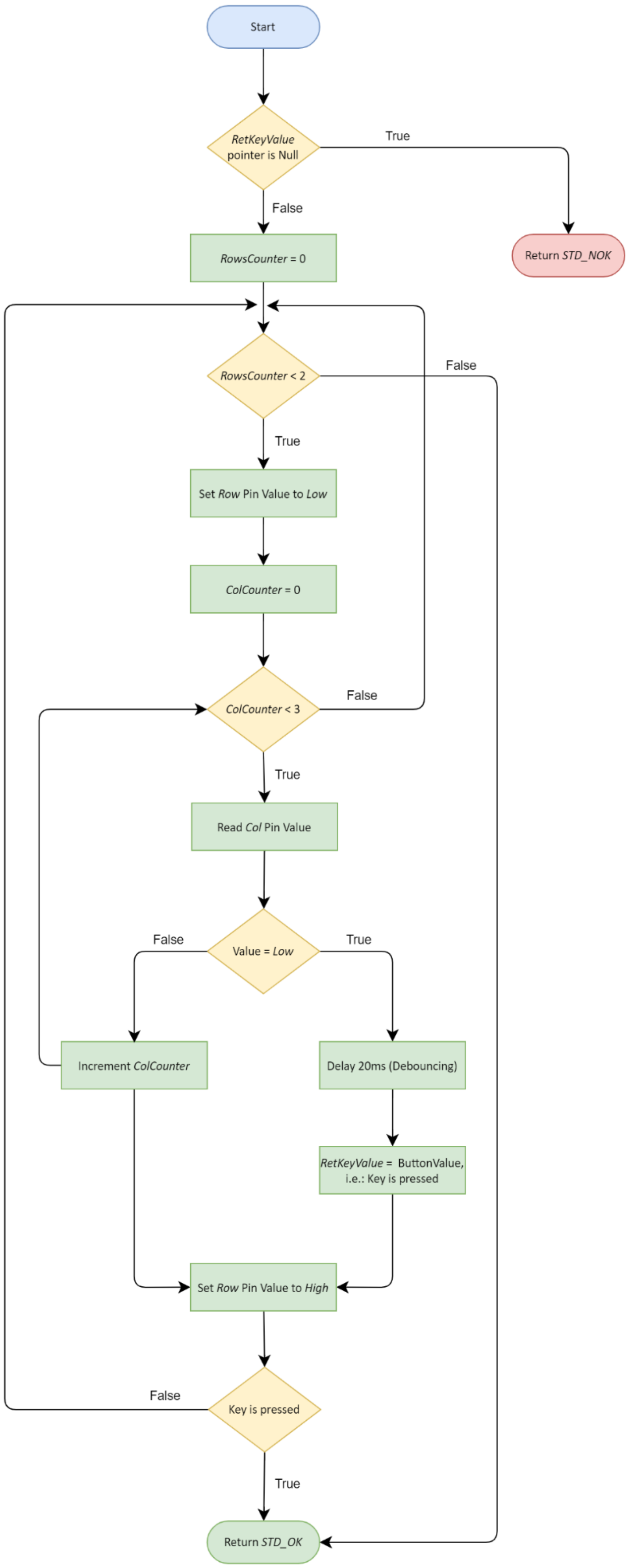
3.2.4.2. KPD_enableKPD



3.2.4.3. KPD_disableKPD

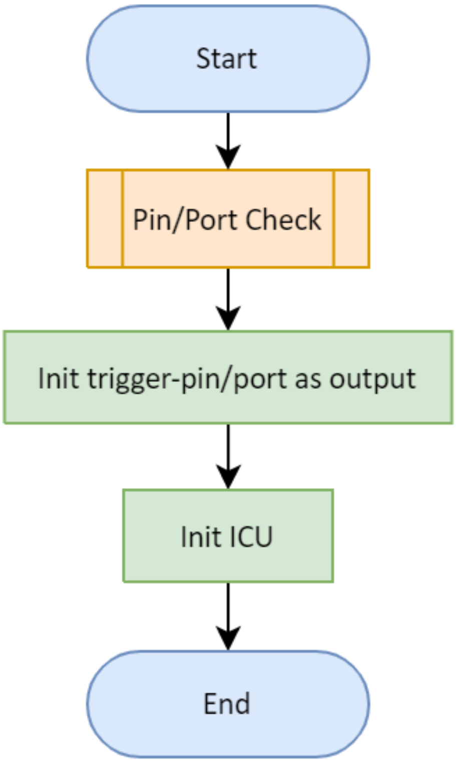


3.2.4.4. KPD_getPressedKey

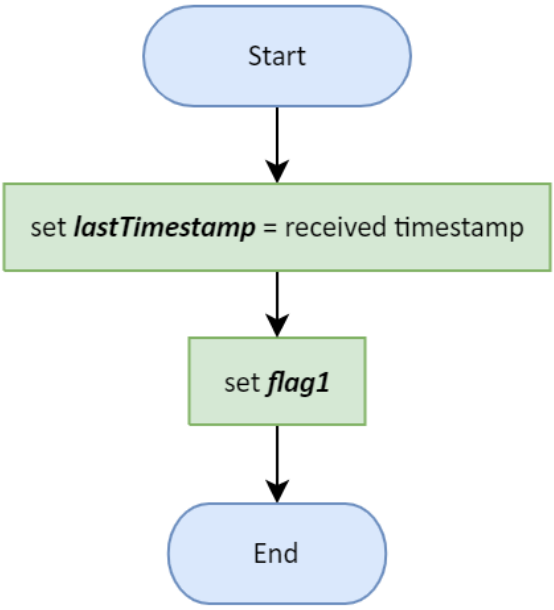


3.2.5. Ultrasonic Module

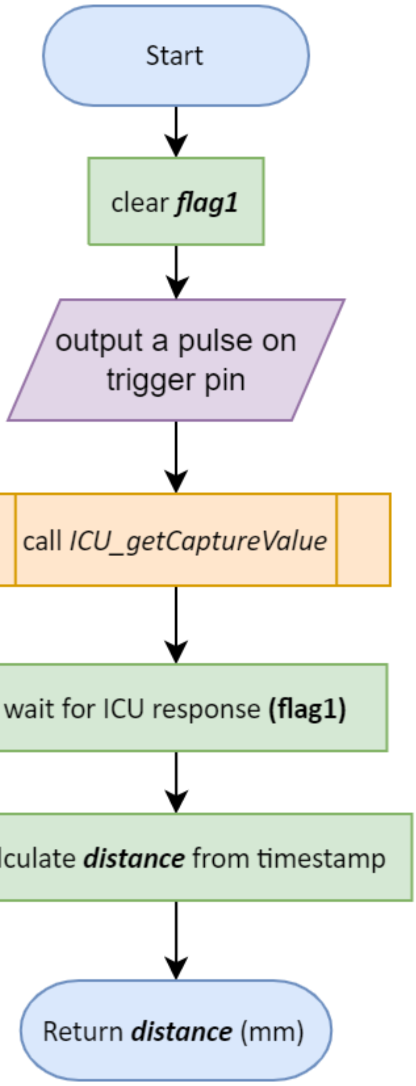
3.2.5.1. US_init



3.2.5.2. US_evtDistance

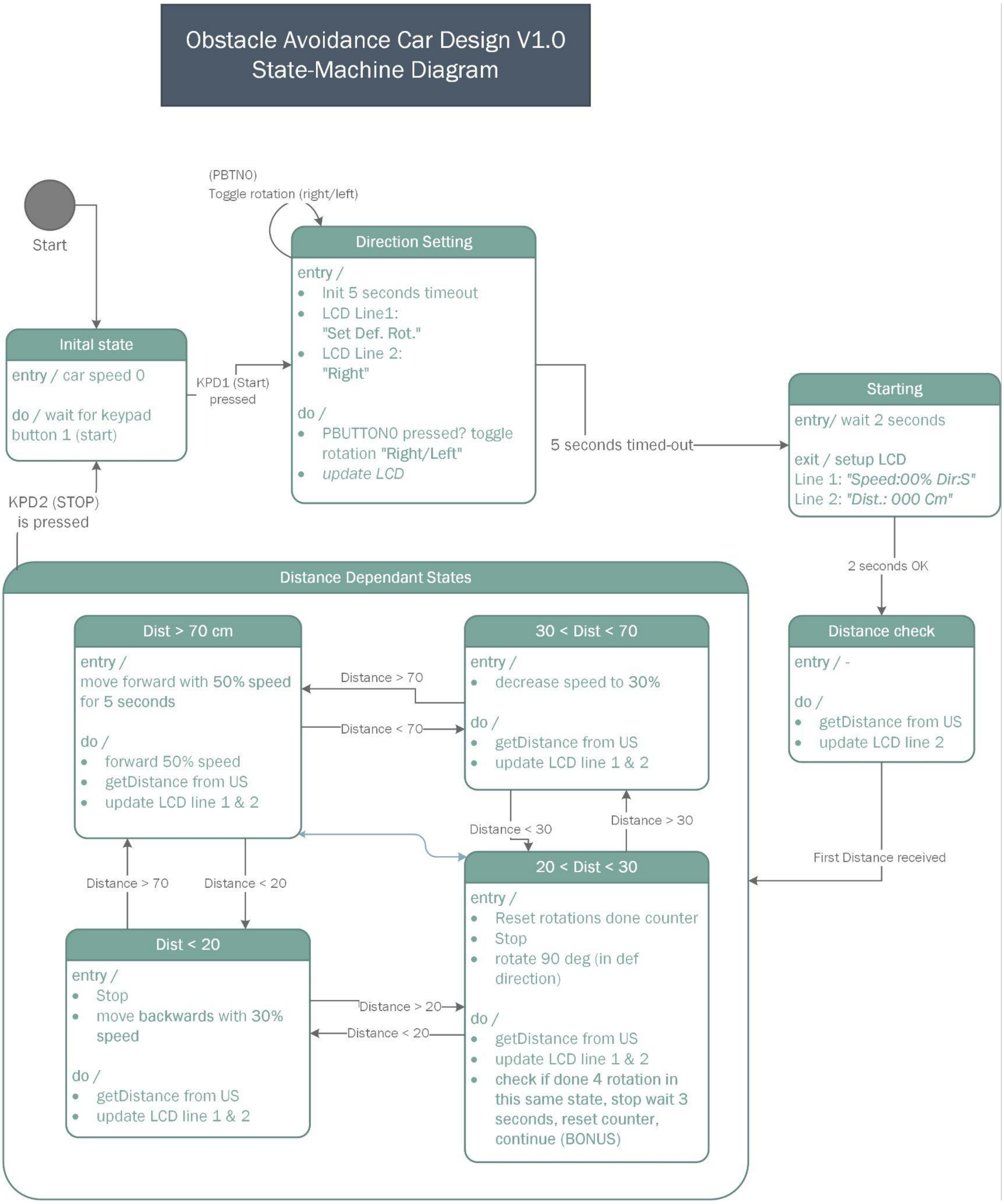


3.2.5.3. US_getDistance



3.3. APP Layer

3.3.1. App State Diagram *(for HQ click me)*



4. Pre-compiling and linking configurations

4.1. DIO

```
#ifndef DIO_INTERFACE_H_
#define DIO_INTERFACE_H_

/* Libraries */
#include "../../LIB/bit_math.h"
#include "../../LIB/std.h"

/* DIO Macros/Enums **/

/* DIO Pins */
#define DIO_U8_PIN_0 0
#define DIO_U8_PIN_1 1
#define DIO_U8_PIN_2 2
#define DIO_U8_PIN_3 3
#define DIO_U8_PIN_4 4
#define DIO_U8_PIN_5 5
#define DIO_U8_PIN_6 6
#define DIO_U8_PIN_7 7


/* DIO Pin value as low = 0 or high = 1 */
#define DIO_U8_PIN_LOW      0
#define DIO_U8_PIN_HIGH     1

#define DIO_U8_PORT_LOW      0x00
#define DIO_U8_PORT_HIGH     0xFF

/**
* @brief Defines masks for selecting certain bits in a byte.
*
* This pragma region contains pre-defined mask values for selecting specific bits
in a byte, where DIO_NO_MASK selects
* all bits in the byte, and DIO_MASK_BITS_0, DIO_MASK_BITS_0_1, and so on, select a
specific number of bits in the
* byte, starting from the 0th bit. The mask values can be used with bitwise AND
operations to extract the selected
* bits from a byte.
*
* @note The values defined in this region assume that a byte is 8 bits wide.
*/
//region MASK OPTIONS
#define DIO_NO_MASK                        0xFF
#define DIO_MASK_BITS_0                    0x1
#define DIO_MASK_BITS_0_1                  0x3
#define DIO_MASK_BITS_0_1_2                 0x7
#define DIO_MASK_BITS_0_1_2_3               0xF
#define DIO_MASK_BITS_0_1_2_3_4              0x1F
#define DIO_MASK_BITS_0_1_2_3_4_5            0x3F
```

```
#define DIO_MASK_BITS_0_1_2_3_4_5_6         0x7F
#define DIO_MASK_BITS_0_1_2_3_4_5_7         0xBF
#define DIO_MASK_BITS_0_1_2_3_4_6          0x5F
………
#define DIO_MASK_BITS_7                    0x80
//endregion

/**
* @brief Enumeration of possible DIO ports
*/
typedef enum EN_DIO_PORT_T
{
   PORT_A, /*!< Port A */
   PORT_B, /*!< Port B */
   PORT_C, /*!< Port C */
   PORT_D  /*!< Port D */
}EN_DIO_PORT_T;

/**
* @brief Enumeration for DIO direction.
*
* This enumeration defines the available directions for a Digital Input/Output
(DIO) pin.
*
* @note This enumeration is used as input to the DIO driver functions for setting
the pin direction.
*/
typedef enum EN_DIO_DIRECTION_T
{
   DIO_IN = 0,      /**< Input direction */
   DIO_OUT = 1      /**< Output direction */
} EN_DIO_DIRECTION_T;

/**
* @brief Enumeration for DIO direction.
*
* This enumeration defines the available directions for a Digital Input/Output
(DIO) pin.
*
* @note This enumeration is used as input to the DIO driver functions for setting
the pin direction.
*/
typedef enum EN_DIO_PORT_DIRECTION_T
{
   DIO_PORT_IN = 0,      /**< Input direction */
   DIO_PORT_OUT = 0xFF    /**< Output direction */
} EN_DIO_PORT_DIRECTION_T;

/**
* @brief Enumeration of DIO error codes
*
* This enumeration defines the possible error codes that can be returned by
```

```
* functions in the DIO driver.
*
*/
typedef enum EN_DIO_ERROR_T
{
   DIO_OK,      /**< Operation completed successfully */
   DIO_ERROR    /**< An error occurred during the operation */
} EN_DIO_ERROR_T;
#endif /* DIO_INTERFACE_H_ */
```

4.2. EXI

```
#ifndef EXI_INTERFACE_H_
#define EXI_INTERFACE_H_

/* EXI Includes */

/* LIB */
#include "../../LIB/std.h"
#include "../../LIB/bit_math.h"


/* The 3 External Interrupts counted from 0 to 2 */
#define EXI_U8_INT0                0
#define EXI_U8_INT1                 1
#define EXI_U8_INT2                 2

/* Interrupts Sense Control */
#define EXI_U8_SENSE_LOW_LEVEL         0
#define EXI_U8_SENSE_LOGICAL_CHANGE     1
#define EXI_U8_SENSE_FALLING_EDGE       2
#define EXI_U8_SENSE_RISING_EDGE        3

/* EXI Functions' Prototypes */

u8 EXI_enablePIE      ( u8 u8_a_interruptId, u8 u8_a_senseControl );
u8 EXI_disablePIE     ( u8 u8_a_interruptId );
u8 EXI_intSetCallBack( u8 u8_a_interruptId, void ( *pf_a_interruptAction ) ( void )
);
```

4.3. Timer

```
#ifndef TIMER_CONFIG_H_
#define TIMER_CONFIG_H_

/*
*********************************************************************************
***********/
/* TIMER Configurations' Definitions */

#define ISR(INT_VECT)    void INT_VECT(void) __attribute__((signal,used));\
```

```
void INT_VECT(void)

#define TIMER2_OVF_vect            __vector_5
#define TIMER0_OVF_vect           __vector_4

/*
* 8-bit Timer/Counter0 Configurations' Definitions
*/

/* TIMER0 Waveform Generation Modes */
#define TIMER_U8_TIMER_0_NORMAL_MODE                       0
#define TIMER_U8_TIMER_0_PWM_PHASE_CORRECT_MODE               1
#define TIMER_U8_TIMER_0_CTC_MODE                            2
#define TIMER_U8_TIMER_0_FAST_PWM_MODE                         3

/* TIMER0 Compare Match Output Modes */
#define TIMER_U8_TIMER_0_DISCONNECT_OC0_PIN                   0
#define TIMER_U8_TIMER_0_TOG_OC0_PIN                       1
#define TIMER_U8_TIMER_0_CLR_OC0_PIN                       2
#define TIMER_U8_TIMER_0_SET_OC0_PIN                       3

/* TIMER0 Interrupt Sources */
#define TIMER_U8_TIMER_0_NO_INTERRUPT                        0
#define TIMER_U8_TIMER_0_COMP_INTERRUPT                        1
#define TIMER_U8_TIMER_0_OVF_INTERRUPT                       2

/* TIMER0 Clock Sources */
#define TIMER_U8_TIMER_0_NO_CLOCK_SOURCE                   0
#define TIMER_U8_TIMER_0_NO_PRESCALER                        1
#define TIMER_U8_TIMER_0_8_PRESCALER                       2
#define TIMER_U8_TIMER_0_64_PRESCALER                        3
#define TIMER_U8_TIMER_0_256_PRESCALER                         4
#define TIMER_U8_TIMER_0_1024_PRESCALER                          5
#define TIMER_U8_TIMER_0_EXTERNAL_CLOCK_SOURCE_FALL_EDGE       6
#define TIMER_U8_TIMER_0_EXTERNAL_CLOCK_SOURCE_RISE_EDGE       7

/* End of Timer/Counter0 Configurations' Definitions */

/*
*********************************************************************************
***********/

/*
* 16-bit Timer/Counter1 Configurations' Definitions
*/

/* TIMER1 Waveform Generation Modes */
#define TIMER_U8_TIMER_1_NORMAL_MODE                       0
#define TIMER_U8_TIMER_1_PWM_PHASE_CORRECT_8_BIT_MODE        1
#define TIMER_U8_TIMER_1_PWM_PHASE_CORRECT_9_BIT_MODE        2
#define TIMER_U8_TIMER_1_PWM_PHASE_CORRECT_10_BIT_MODE         3
#define TIMER_U8_TIMER_1_CTC_OCR1A_TOP                         4
```

```
#define TIMER_U8_TIMER_1_FAST_PWM_8_BIT_MODE                       5
#define TIMER_U8_TIMER_1_FAST_PWM_9_BIT_MODE                       6
#define TIMER_U8_TIMER_1_FAST_PWM_10_BIT_MODE                       7
#define TIMER_U8_TIMER_1_PWM_PHASE_AND_FREQ_CORRECT_ICR1_TOP       8
#define TIMER_U8_TIMER_1_PWM_PHASE_AND_FREQ_CORRECT_OCR1A_TOP       9
#define TIMER_U8_TIMER_1_PWM_PHASE_CORRECT_ICR1_TOP                   10
#define TIMER_U8_TIMER_1_PWM_PHASE_CORRECT_OCR1A_TOP               11
#define TIMER_U8_TIMER_1_CTC_ICR1_TOP                               12
#define TIMER_U8_TIMER_1_FAST_PWM_ICR1_TOP                           13
#define TIMER_U8_TIMER_1_FAST_PWM_OCR1A_TOP                           14

/* TIMER1 Compare Match Output Modes - Channel A */
#define TIMER_U8_TIMER_1_DISCONNECT_OC1A_PIN                       0
#define TIMER_U8_TIMER_1_TOG_OC1A_PIN                               1
#define TIMER_U8_TIMER_1_CLR_OC1A_PIN                               2
#define TIMER_U8_TIMER_1_SET_OC1A_PIN                               3

/* TIMER1 Compare Match Output Modes - Channel B */
#define TIMER_U8_TIMER_1_DISCONNECT_OC1B_PIN                       0
#define TIMER_U8_TIMER_1_TOG_OC1B_PIN                               1
#define TIMER_U8_TIMER_1_CLR_OC1B_PIN                               2
#define TIMER_U8_TIMER_1_SET_OC1B_PIN                               3

/* TIMER1 Interrupt Sources */
#define TIMER_U8_TIMER_1_NO_INTERRUPT                               0
#define TIMER_U8_TIMER_1_CAPT_INTERRUPT                               1
#define TIMER_U8_TIMER_1_COMP_A_INTERRUPT                           2
#define TIMER_U8_TIMER_1_COMP_B_INTERRUPT                           3
#define TIMER_U8_TIMER_1_OVF_INTERRUPT                                4

/* TIMER1 Clock Sources */
#define TIMER_U8_TIMER_1_NO_CLOCK_SOURCE                           0
#define TIMER_U8_TIMER_1_NO_PRESCALER                               1
#define TIMER_U8_TIMER_1_8_PRESCALER                               2
#define TIMER_U8_TIMER_1_64_PRESCALER                               3
#define TIMER_U8_TIMER_1_256_PRESCALER                                4
#define TIMER_U8_TIMER_1_1024_PRESCALER                                5
#define TIMER_U8_TIMER_1_EXTERNAL_CLOCK_SOURCE_FALL_EDGE           6
#define TIMER_U8_TIMER_1_EXTERNAL_CLOCK_SOURCE_RISE_EDGE           7

/* End of Timer/Counter1 Configurations' Definitions */

/*
*********************************************************************************
************/

/*
* 8-bit Timer/Counter2 Configurations' Definitions
*/

/* TIMER2 Waveform Generation Modes */
#define TIMER_U8_TIMER_2_NORMAL_MODE                               0
```

```
#define TIMER_U8_TIMER_2_PWM_PHASE_CORRECT_MODE                     1
#define TIMER_U8_TIMER_2_CTC_MODE                                   2
#define TIMER_U8_TIMER_2_FAST_PWM_MODE                              3

/* TIMER2 Compare Match Output Modes */
#define TIMER_U8_TIMER_2_DISCONNECT_OC2_PIN                         0
#define TIMER_U8_TIMER_2_TOG_OC2_PIN                                1
#define TIMER_U8_TIMER_2_CLR_OC2_PIN                                2
#define TIMER_U8_TIMER_2_SET_OC2_PIN                                3

/* TIMER2 Interrupt Sources */
#define TIMER_U8_TIMER_2_NO_INTERRUPT                               0
#define TIMER_U8_TIMER_2_COMP_INTERRUPT                             1
#define TIMER_U8_TIMER_2_OVF_INTERRUPT                              2

/* TIMER2 Clock Sources */
#define TIMER_U8_TIMER_2_NO_CLOCK_SOURCE                            0
#define TIMER_U8_TIMER_2_NO_PRESCALER                               1
#define TIMER_U8_TIMER_2_8_PRESCALER                                2
#define TIMER_U8_TIMER_2_32_PRESCALER                               3
#define TIMER_U8_TIMER_2_64_PRESCALER                               4
#define TIMER_U8_TIMER_2_128_PRESCALER                              5
#define TIMER_U8_TIMER_2_256_PRESCALER                              6
#define TIMER_U8_TIMER_2_1024_PRESCALER                             7

/* End of Timer/Counter2 Configurations' Definitions */

/*
*********************************************************************************
***********/
/* TIMER Configurations */

/*
* 8-bit Timer/Counter0 Configurations
*/

/* TIMER0 Waveform Generation Mode Select */
/* Options: TIMER_U8_TIMER_0_NORMAL_MODE
*           TIMER_U8_TIMER_0_PWM_PHASE_CORRECT_MODE
*           TIMER_U8_TIMER_0_CTC_MODE
*           TIMER_U8_TIMER_0_FAST_PWM_MODE
*/
#define TIMER_U8_TIMER_0_MODE_SELECT            TIMER_U8_TIMER_0_NORMAL_MODE

/* TIMER0 Compare Match Output Mode Select */
/* Options: TIMER_U8_TIMER_0_DISCONNECT_OC0_PIN                     // Any Mode
*           TIMER_U8_TIMER_0_TOG_OC0_PIN                            // Non-PWM Modes
only
*           TIMER_U8_TIMER_0_CLR_OC0_PIN                            // Any Mode (
PWM -> Non-Inverting Mode )
*           TIMER_U8_TIMER_0_SET_OC0_PIN                            // Any Mode (
PWM -> Inverting Mode )
```

```
*/
#define TIMER_U8_TIMER_0_COMP_OUTPUT_MODE
TIMER_U8_TIMER_0_DISCONNECT_OC0_PIN

/* TIMER0 Interrupt Select */
/* Options: TIMER_U8_TIMER_0_NO_INTERRUPT
*         TIMER_U8_TIMER_0_COMP_INTERRUPT
*         TIMER_U8_TIMER_0_OVF_INTERRUPT
*/
#define TIMER_U8_TIMER_0_INTERRUPT_SELECT        TIMER_U8_TIMER_0_NO_INTERRUPT

/* TIMER0 Clock Select */
/* Options: TIMER_U8_TIMER_0_NO_CLOCK_SOURCE                   // No clock source (
Timer/Counter0 stopped )
*         TIMER_U8_TIMER_0_NO_PRESCALER                    // CLK IO/1    ( No
prescaling )
*         TIMER_U8_TIMER_0_8_PRESCALER                     // CLK IO/8    ( From
prescaler )
*         TIMER_U8_TIMER_0_64_PRESCALER                    // CLK IO/64   ( From
prescaler )
*         TIMER_U8_TIMER_0_256_PRESCALER                   // CLK IO/256  ( From
prescaler )
*         TIMER_U8_TIMER_0_1024_PRESCALER                      // CLK IO/1024( From
prescaler )
*         TIMER_U8_TIMER_0_EXTERNAL_CLOCK_SOURCE_FALL_EDGE      // External clock
source on T0 pin. Clock on falling edge.
*         TIMER_U8_TIMER_0_EXTERNAL_CLOCK_SOURCE_RISE_EDGE      // External clock
source on T0 pin. Clock on rising edge.
*/
#define TIMER_U8_TIMER_0_CLOCK_SELECT            TIMER_U8_TIMER_0_NO_CLOCK_SOURCE

/* TIMER0 Other Configurations */
#define TIMER_U8_TIMER_0_PRELOAD_VALUE           0
#define TIMER_U8_TIMER_0_COMPARE_VALUE           0
#define TIMER_U16_TIMER_0_NUM_OF_OVERFLOWS       1

/* End of Timer/Counter0 Configurations */

/*
*********************************************************************************
***********/

/*
* 16-bit Timer/Counter1 Configurations
*/

/* TIMER1 Waveform Generation Mode Select */
/* Options: TIMER_U8_TIMER_1_NORMAL_MODE                     // Overflow Value (
TOP ) = 0xFFFF
*         TIMER_U8_TIMER_1_PWM_PHASE_CORRECT_8_BIT_MODE        // Overflow Value (
TOP ) = 0x00FF
```

```
*         TIMER_U8_TIMER_1_PWM_PHASE_CORRECT_9_BIT_MODE        // Overflow Value (
TOP ) = 0x01FF
*         TIMER_U8_TIMER_1_PWM_PHASE_CORRECT_10_BIT_MODE       // Overflow Value (
TOP ) = 0x03FF
*         TIMER_U8_TIMER_1_CTC_OCR1A_TOP                     // Overflow Value ( TOP )
= OCR1A
*         TIMER_U8_TIMER_1_FAST_PWM_8_BIT_MODE                 // Overflow Value ( TOP
) = 0x00FF
*         TIMER_U8_TIMER_1_FAST_PWM_9_BIT_MODE                 // Overflow Value ( TOP
) = 0x01FF
*         TIMER_U8_TIMER_1_FAST_PWM_10_BIT_MODE                // Overflow Value ( TOP
) = 0x03FF
*         TIMER_U8_TIMER_1_PWM_PHASE_AND_FREQ_CORRECT_ICR1_TOP   // Overflow Value (
TOP ) = ICR1
*         TIMER_U8_TIMER_1_PWM_PHASE_AND_FREQ_CORRECT_OCR1A_TOP  // Overflow Value (
TOP ) = OCR1A
*         TIMER_U8_TIMER_1_PWM_PHASE_CORRECT_ICR1_TOP            // Overflow Value
( TOP ) = ICR1
*         TIMER_U8_TIMER_1_PWM_PHASE_CORRECT_OCR1A_TOP         // Overflow Value (
TOP ) = OCR1A
*         TIMER_U8_TIMER_1_CTC_ICR1_TOP                      // Overflow Value ( TOP )
= ICR1
*         TIMER_U8_TIMER_1_FAST_PWM_ICR1_TOP                  // Overflow Value ( TOP
) = ICR1
*         TIMER_U8_TIMER_1_FAST_PWM_OCR1A_TOP                    // Overflow Value (
TOP ) = OCR1A
*/
#define TIMER_U8_TIMER_1_MODE_SELECT            TIMER_U8_TIMER_1_NORMAL_MODE

/* TIMER1 Compare Match Output Mode Select - Channel A */
/* Options: TIMER_U8_TIMER_1_DISCONNECT_OC1A_PIN                    // Any Mode
*           TIMER_U8_TIMER_1_TOG_OC1A_PIN                          // Selected
Modes only
*           TIMER_U8_TIMER_1_CLR_OC1A_PIN                          // Any Mode (
PWM -> Non-Inverting Mode )
*           TIMER_U8_TIMER_1_SET_OC1A_PIN                          // Any Mode (
PWM -> Inverting Mode )
*/
#define TIMER_U8_TIMER_1_COMP_OUTPUT_MODE_A
TIMER_U8_TIMER_1_DISCONNECT_OC1A_PIN

/* TIMER1 Compare Match Output Mode Select - Channel B */
/* Options: TIMER_U8_TIMER_1_DISCONNECT_OC1B_PIN                // Any Mode
*         TIMER_U8_TIMER_1_TOG_OC1B_PIN                       // Non-PWM Modes
only
*           TIMER_U8_TIMER_1_CLR_OC1B_PIN                          // Any Mode (
PWM -> Non-Inverting Mode )
*           TIMER_U8_TIMER_1_SET_OC1B_PIN                          // Any Mode (
PWM -> Inverting Mode )
*/
#define TIMER_U8_TIMER_1_COMP_OUTPUT_MODE_B
TIMER_U8_TIMER_1_DISCONNECT_OC1A_PIN
```

```
/* TIMER1 Interrupt Select */
/* Options: TIMER_U8_TIMER_1_NO_INTERRUPT
*         TIMER_U8_TIMER_1_CAPT_INTERRUPT
*         TIMER_U8_TIMER_1_COMP_A_INTERRUPT
*         TIMER_U8_TIMER_1_COMP_B_INTERRUPT
*         TIMER_U8_TIMER_1_OVF_INTERRUPT
*/
#define TIMER_U8_TIMER_1_INTERRUPT_SELECT        TIMER_U8_TIMER_1_NO_INTERRUPT

/* TIMER1 Clock Select */
/* Options: TIMER_U8_TIMER_1_NO_CLOCK_SOURCE                   // No clock source (
Timer/Counter1 stopped )
*         TIMER_U8_TIMER_1_NO_PRESCALER                    // CLK IO/1    ( No
prescaling )
*         TIMER_U8_TIMER_1_8_PRESCALER                     // CLK IO/8    ( From
prescaler )
*         TIMER_U8_TIMER_1_64_PRESCALER                    // CLK IO/64   ( From
prescaler )
*         TIMER_U8_TIMER_1_256_PRESCALER                   // CLK IO/256  ( From
prescaler )
*         TIMER_U8_TIMER_1_1024_PRESCALER                      // CLK IO/1024( From
prescaler )
*         TIMER_U8_TIMER_1_EXTERNAL_CLOCK_SOURCE_FALL_EDGE      // External clock
source on T1 pin. Clock on falling edge.
*         TIMER_U8_TIMER_1_EXTERNAL_CLOCK_SOURCE_RISE_EDGE      // External clock
source on T1 pin. Clock on rising edge.
*/
#define TIMER_U8_TIMER_1_CLOCK_SELECT            TIMER_U8_TIMER_1_NO_CLOCK_SOURCE

/* TIMER1 Other Configurations */
#define TIMER_U16_TIMER_1_PRELOAD_VALUE            0
#define TIMER_U16_TIMER_1_COMPARE_VALUE_A        0
#define TIMER_U16_TIMER_1_COMPARE_VALUE_B        0
#define TIMER_U16_TIMER_1_INPUT_CAPTURE_VALUE    0
#define TIMER_U16_TIMER_1_NUM_OF_OVERFLOWS         1

/* End of Timer/Counter1 Configurations */

/*
*********************************************************************************
***********/

/*
* 8-bit Timer/Counter2 Configurations
*/

/* TIMER2 Waveform Generation Mode Select */
/* Options: TIMER_U8_TIMER_2_NORMAL_MODE
*           TIMER_U8_TIMER_2_PWM_PHASE_CORRECT_MODE
*           TIMER_U8_TIMER_2_CTC_MODE
*           TIMER_U8_TIMER_2_FAST_PWM_MODE
```

```
*/
#define TIMER_U8_TIMER_2_MODE_SELECT            TIMER_U8_TIMER_2_NORMAL_MODE

/* TIMER2 Compare Match Output Mode Select */
/* Options: TIMER_U8_TIMER_2_DISCONNECT_OC2_PIN                // Any Mode
*           TIMER_U8_TIMER_2_TOG_OC2_PIN                      // Non-PWM Modes only
*           TIMER_U8_TIMER_2_CLR_OC2_PIN                      // Any Mode ( PWM ->
Non-Inverting Mode )
*           TIMER_U8_TIMER_2_SET_OC2_PIN                      // Any Mode ( PWM ->
Inverting Mode )
*/
#define TIMER_U8_TIMER_2_COMP_OUTPUT_MODE
TIMER_U8_TIMER_2_DISCONNECT_OC2_PIN

/* TIMER2 Interrupt Select */
/* Options: TIMER_U8_TIMER_2_NO_INTERRUPT
*         TIMER_U8_TIMER_2_COMP_INTERRUPT
*         TIMER_U8_TIMER_2_OVF_INTERRUPT
*/
#define TIMER_U8_TIMER_2_INTERRUPT_SELECT        TIMER_U8_TIMER_2_NO_INTERRUPT

/* TIMER2 Clock Select */
/* Options: TIMER_U8_TIMER_2_NO_CLOCK_SOURCE                  // No clock source (
Timer/Counter2 stopped )
*         TIMER_U8_TIMER_2_NO_PRESCALER                     // CLK T2S/1    ( No
prescaling )
*         TIMER_U8_TIMER_2_8_PRESCALER                      // CLK T2S/8    ( From
prescaler )
*         TIMER_U8_TIMER_2_32_PRESCALER                     // CLK T2S/32   ( From
prescaler )
*         TIMER_U8_TIMER_2_64_PRESCALER                     // CLK T2S/64   ( From
prescaler )
*         TIMER_U8_TIMER_2_128_PRESCALER                    // CLK T2S/128 ( From
prescaler )
*         TIMER_U8_TIMER_2_256_PRESCALER                    // CLK T2S/256 ( From
prescaler )
*         TIMER_U8_TIMER_2_1024_PRESCALER                       // CLK T2S/1024( From
prescaler )
*/
#define TIMER_U8_TIMER_2_CLOCK_SELECT            TIMER_U8_TIMER_2_NO_CLOCK_SOURCE

/* TIMER2 Other Configurations */
#define TIMER_U8_TIMER_2_PRELOAD_VALUE            0
#define TIMER_U8_TIMER_2_COMPARE_VALUE            0
#define TIMER_U16_TIMER_2_NUM_OF_OVERFLOWS        1

/* End of Timer/Counter2 Configurations */

/*
*********************************************************************************
************/
```

```
/*
* TIMER Other Configurations
*/

/* Timers Flags */
#define TIMER_U8_FLAG_DOWN                    0
#define TIMER_U8_FLAG_UP                        1

/* End of Configurations */

/*
**********************************************************************************
***********/
```

4.4. ICU

4.4.1. icu_interface.h

```
#ifndef OBSTACLEAVCAR_ICU_INTERFACE_H
#define OBSTACLEAVCAR_ICU_INTERFACE_H

#include "../../LIB/std.h"

/**
* Initializes the ICU driver
*
* This function initializes a software ICU driver, init echo pin as input,
* uses timer to calculate elapsed time, when echo pin is triggered it sends the
* elapsed time duration to the callback function
*
* @param [in]u8_a_echoPin I/O pin number to receive echoed signal
* @param [in]cf_a_timeReceived callback function to send elapsed duration
*/
void ICU_init(u8 u8_a_echoPin, void (* cf_a_timeReceived));

/**
* Resets and starts the ICU algorithm to capture the elapsed time duration now
* starting until echo signal is received back
*/
void ICU_getCaptureValue(void);

#endif //OBSTACLEAVCAR_ICU_INTERFACE_H
```

4.4.2. icu_cfg.h

```
#ifndef OBSTACLEAVCAR_ICU_CFG_H
#define OBSTACLEAVCAR_ICU_CFG_H

#include "../../LIB/std.h"

typedef struct{
   u8 echoPort;
   u8 echoPin;
   void (* timeReceivedCallbackFun);
```

```
}st_icuConfig_t;

#endif //OBSTACLEAVCAR_ICU_CFG_H
```

4.4.3. icu_cfg.c

```
#include "icu_cfg.h"

st_icuConfig_t config = {
        1, // Port B
        2, // Pin 2
        NULL // callback function
};

st_icuConfig_t ICU_getConfig()
{
   return config;
}

void ICU_setConfig(st_icuConfig_t stPtr_a_customConfig)
{
   config = stPtr_a_customConfig;
}
```

4.5. LCD

4.5.1. lcd_interface.h

```
#ifndef LCD_INTERFACE_H_
#define LCD_INTERFACE_H_

/* Includes */
#include "../../LIB/std.h"
#include "lcd_config.h"
#include "../../MCAL/timer/timer_interface.h"

/* Macros */

// LCD Lines
#define LCD_LINE0    0
#define LCD_LINE1    1

// LCD Columns
#define LCD_COL0     0
#define LCD_COL1     1
#define LCD_COL2     2
#define LCD_COL3     3
#define LCD_COL4     4
#define LCD_COL5     5
#define LCD_COL6     6
#define LCD_COL7     7
#define LCD_COL8     8
#define LCD_COL9     9
#define LCD_COL10    10
#define LCD_COL11    11
#define LCD_COL12    12
#define LCD_COL13    13
#define LCD_COL14    14
#define LCD_COL15    15

// LCD Custom Chars Locations
#define LCD_CUSTOMCHAR_LOC0 0
#define LCD_CUSTOMCHAR_LOC1 1
#define LCD_CUSTOMCHAR_LOC2 2
#define LCD_CUSTOMCHAR_LOC3 3
#define LCD_CUSTOMCHAR_LOC4 4
#define LCD_CUSTOMCHAR_LOC5 5
#define LCD_CUSTOMCHAR_LOC6 6
#define LCD_CUSTOMCHAR_LOC7 7

/* Custom Chars */
#define LCD_CUSTOM_SOLID_BLOCK 0xFF

/* Prototypes */

/**
* @brief Initializes the LCD module.
```

```
* This function initializes the LCD module by configuring the data port,
* configuring the LCD to 4-bit mode,setting the display to on with cursor
* and blink, setting the cursor to increment to the right, and clearing the
display.
* It also pre-stores a bell shape at CGRAM location 0.
*
* @return void
*/
void LCD_init(void);


/**
* @brief Sends a command to the LCD controller
*
* Sends the upper nibble of the command to the LCD's data pins, selects the command
register by setting RS to low,
* generates an enable pulse, delays for a short period, then sends the lower nibble
of the command and generates
* another enable pulse. Finally, it delays for a longer period to ensure the
command has been executed by the LCD
* controller.
*
* @param [in]u8_a_cmd The command to be sent
*/
void LCD_sendCommand(u8 u8_a_cmd);

/**
* @brief Sends a single character to the LCD display
*
* This function sends a single character to the LCD display by selecting the data
register and sending the
* higher nibble and lower nibble of the character through the data port.
* The function uses a pulse on the enable pin to signal the LCD to read the data on
the data port.
* The function also includes delays to ensure proper timing for the LCD to read the
data.
*
* @param [in]u8_a_data single char ASCII data to show
*/
void LCD_sendChar(u8 u8_a_data);

/**
* @brief Displays a null-terminated string on the LCD screen.
*
* This function iterates through a null-terminated string and displays it
* on the LCD screen. If the character '\\n' is encountered, the cursor is
* moved to the beginning of the next line.
*
* @param [in]u8Ptr_a_str A pointer to the null-terminated string to be displayed.
*
* @return void
*/
```

```
void LCD_sendString(u8 * u8Ptr_a_str);

/**
* @brief Set the cursor position on the LCD.
*
* @param [in]u8_a_line the line number to set the cursor to, either LCD_LINE0 or
LCD_LINE1
* @param [in]u8_a_col the column number to set the cursor to, from LCD_COL0 to
LCD_COL15
*
* @return STD_OK if the operation was successful, STD_NOK otherwise.
*/
u8 LCD_setCursor(u8 u8_a_line, u8 u8_a_col);

/**
* @brief Stores a custom character bitmap pattern in the CGRAM of the LCD module
*
* @param [in]u8_a_pattern Pointer to an array of 8 bytes representing the bitmap
pattern of the custom character
* @param [in]u8_a_location The CGRAM location where the custom character should be
stored (from LCD_CUSTOMCHAR_LOC0 to 7)
*
* @return STD_OK if successful, otherwise STD_NOK
*/
u8 LCD_storeCustomCharacter(u8 * u8_a_pattern, u8 u8_a_location);

/**
* Show/Hide cursor
* @param u8_a_show hide: 0, show: otherwise
*/
void LCD_changeCursor(u8 u8_a_show);

/**
* Clears the LCD display
*/
void LCD_clear(void);

/**
* Fancy right shift clear for the LCD display
*/
void LCD_shiftClear(void);
#endif /* LCD_INTERFACE_H_ */
```

4.5.2. lcd_config.h

```
#ifndef LCD_CONFIG_H_
#define LCD_CONFIG_H_

#include "../../MCAL/dio/dio_interface.h"


/** LCD Data */
/* LCD_DATA_PORT Options
```

```
* PORT_A: 0
* PORT_B: 1
* PORT_C: 2
* PORT_D: 3
* */
#define LCD_DATA_PORT 0

// DATA PINS
#define LCD_DATA_PIN_D4      DIO_U8_PIN_4
#define LCD_DATA_PIN_D5      DIO_U8_PIN_5
#define LCD_DATA_PIN_D6      DIO_U8_PIN_6
#define LCD_DATA_PIN_D7      DIO_U8_PIN_7
#define LCD_DATA_PINS_MASK     DIO_MASK_BITS_4_5_6_7

/** LCD Control */
/* LCD_CTRL_PORT Options
* PORT_A: 0
* PORT_B: 1
* PORT_C: 2
* PORT_D: 3
* */
#define LCD_CTRL_PORT 0
#define LCD_CTRL_PIN_RS      DIO_U8_PIN_1
#define LCD_CTRL_PIN_RW      DIO_U8_PIN_2
#define LCD_CTRL_PIN_EN      DIO_U8_PIN_3


#define HIGHER_NIBBLE_SHIFT(cmd)     cmd
#define LOWER_NIBBLE_SHIFT(cmd)      cmd << 4


#endif /* LCD_CONFIG_H_ */
```

4.6. BTN

```
#ifndef BTN_CONFIG_H_
#define BTN_CONFIG_H_

/* BTN(s) Configurations */

/* The Port connected to BTN(s) */
/* Options: 0 // PORT A
         1 // PORT B
         2 // PORT C
         3 // PORT D
*/
#define BTN_U8_PORT        0 // enum -> EN_DIO_PORT_T.PORT_A in
MCAL/dio/dio_interface.h

/* The Pins connected to BTN(s) */
/* Options: 0 // DIO_U8_PIN0
         1 // DIO_U8_PIN1
```

```
         2 // DIO_U8_PIN2
         3 // DIO_U8_PIN3
         4 // DIO_U8_PIN4
         5 // DIO_U8_PIN5
         6 // DIO_U8_PIN6
         7 // DIO_U8_PIN7
*/
#define BTN_U8_1_PIN       0//DIO_U8_PIN0
#define BTN_U8_2_PIN       1//DIO_U8_PIN1
#define BTN_U8_3_PIN       2//DIO_U8_PIN2
#define BTN_U8_4_PIN       3//DIO_U8_PIN3
#define BTN_U8_5_PIN       4//DIO_U8_PIN4
#define BTN_U8_6_PIN       5//DIO_U8_PIN5
#define BTN_U8_7_PIN       6//DIO_U8_PIN6
#define BTN_U8_8_PIN       7//DIO_U8_PIN7

#endif /* BTN_CONFIG_H_ */
```

4.7. DCM

```
#ifndef DCM_CONFIG_H_
#define DCM_CONFIG_H_

/******************** Motor_0 Configurations ***********************/
#define     MOT0_EN_PIN_NUMBER_0           4
#define     MOT0_EN_PIN_NUMBER_1           5
#define     MOT0_PWM_PIN_NUMBER               0
#define     MOT0_EN_PORT_NUMBER               PORT_C
#define     MOT0_PWM_PORT_NUMBER           PORT_C

/******************** Motor_1 Configurations ***********************/
#define     MOT1_EN_PIN_NUMBER_0           6
#define     MOT1_EN_PIN_NUMBER_1           7
#define     MOT1_PWM_PIN_NUMBER               1
#define     MOT1_EN_PORT_NUMBER               PORT_C
#define     MOT1_PWM_PORT_NUMBER           PORT_C

#endif /* DCM_CONFIG_H_ */
```

4.8. KPD

```
#ifndef KPD_INTERFACE_H_
#define KPD_INTERFACE_H_

/* KPD Includes */

/* LIB */
#include "../../LIB/std.h"
#include "../../LIB/bit_math.h"

/* MCAL */
#include "../../MCAL/dio/dio_interface.h"
```

```
#include "../../MCAL/timer/timer_interface.h"

/* KPD Macros */

/* KPD Flag Values */
#define KPD_U8_KEY_NOT_FOUND        0
#define KPD_U8_KEY_FOUND           1
/* KPD Initial Value of Key */
#define KPD_U8_KEY_NOT_PRESSED     0xff

/* KPD Functions' Prototypes */

void KPD_initKPD    ( void );
void KPD_enableKPD  ( void );
void KPD_disableKPD ( void );

u8 KPD_getPressedKey( u8 *pu8_a_returnedKeyValue );

#endif /* KPD_INTERFACE_H_ */
```

4.9. US

4.9.1. us_interface.c

```c
#ifndef OBSTACLEAVCAR_US_INTERFACE_H
#define OBSTACLEAVCAR_US_INTERFACE_H

#include "../../LIB/std.h"

/**
* Initializes the ultrasonic driver
*
* @param u8_a_triggerPin I/O pin number to send trigger signal
* @param u8_a_echoPin I/O pin number to receive echoed signal
*
*/
void US_init(u8 u8_a_triggerPin, u8 u8_a_echoPin);

/**
*
* Initiates a get distance request
*
* This function sends a signal out to the trigger pin, waits for echo signal
* to come back and finally calculates the distance the signal traveled using
* the elapsed time duration used by the signal to arrive back on the echo pin
*
* @return float distance in mm
*/
float US_getDistance(void);

/**
* Event Handler: called when echo time is received from ICU (input capture unit)
*
* This function is called by the ICU as an event callback when the trigger signal
* is received back on the echo pin, the function receives the elapsed time taken.
*
* @param [in]u8_a_timeElapsed time elapsed (duration) by trigger signal to echo
back
*/
void US_evtEchoTimeReceived(u8 u8_a_timeElapsed);

#endif //OBSTACLEAVCAR_US_INTERFACE_H
```

4.9.2. us_cfg.h

```
#ifndef OBSTACLEAVCAR_US_CFG_H
#define OBSTACLEAVCAR_US_CFG_H

#include "../../LIB/std.h"

// 343 m/s = 0.0343 cm/uS = 1/29.1 cm/uS
#define SPEED_OF_SOUND_IN_AIR (1/29.1) // 1/29.1 cm/uS

#define CALC_DISTANCE_CM(travelTimeMs) (((travelTime/2)/1000) * (SPEED_OF_SOUND_IN_AIR))

typedef struct{
   u8 US_Port;
   u8 triggerPin;
   u8 echoPin;
}st_usConfig_t;

#endif //OBSTACLEAVCAR_US_CFG_H
```

4.9.3. us_cfg.c

```
#include "us_cfg.h"

st_usConfig_t config = {
       1, // Port B
       2, // Pin 2
       3, // Pin 3
};

st_usConfig_t US_getConfig()
{
   return config;
}

void US_setConfig(st_usConfig_t stPtr_a_customConfig)
{
   config = stPtr_a_customConfig;
}
```

5. 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. *4WD Complete Mini Plastic Robot Chassis Kit - With 4x 90 Degree Motors - RAM*
8. *Moving-Car-Project: Embedded C application controlling a four-driving wheel robot, and moving it in a rectangular shape. (github.com)*