OBSTACLE AVOIDANCE CAR DESIGN

Presented to: Sprints
Prepared by: Ahmed Hesham

Obstacle Avoidance Car Design

1. Project Introduction

Obstacle avoidance car is an embedded systems project which is using Atmega32 microcontroller to control a bunch of sensors and dc motors to make the car avoid crashing when it finds obstacle in front of it and it will control it's direction to detect the best route to move through it.

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

2. High Level Design

2.1. System Architecture

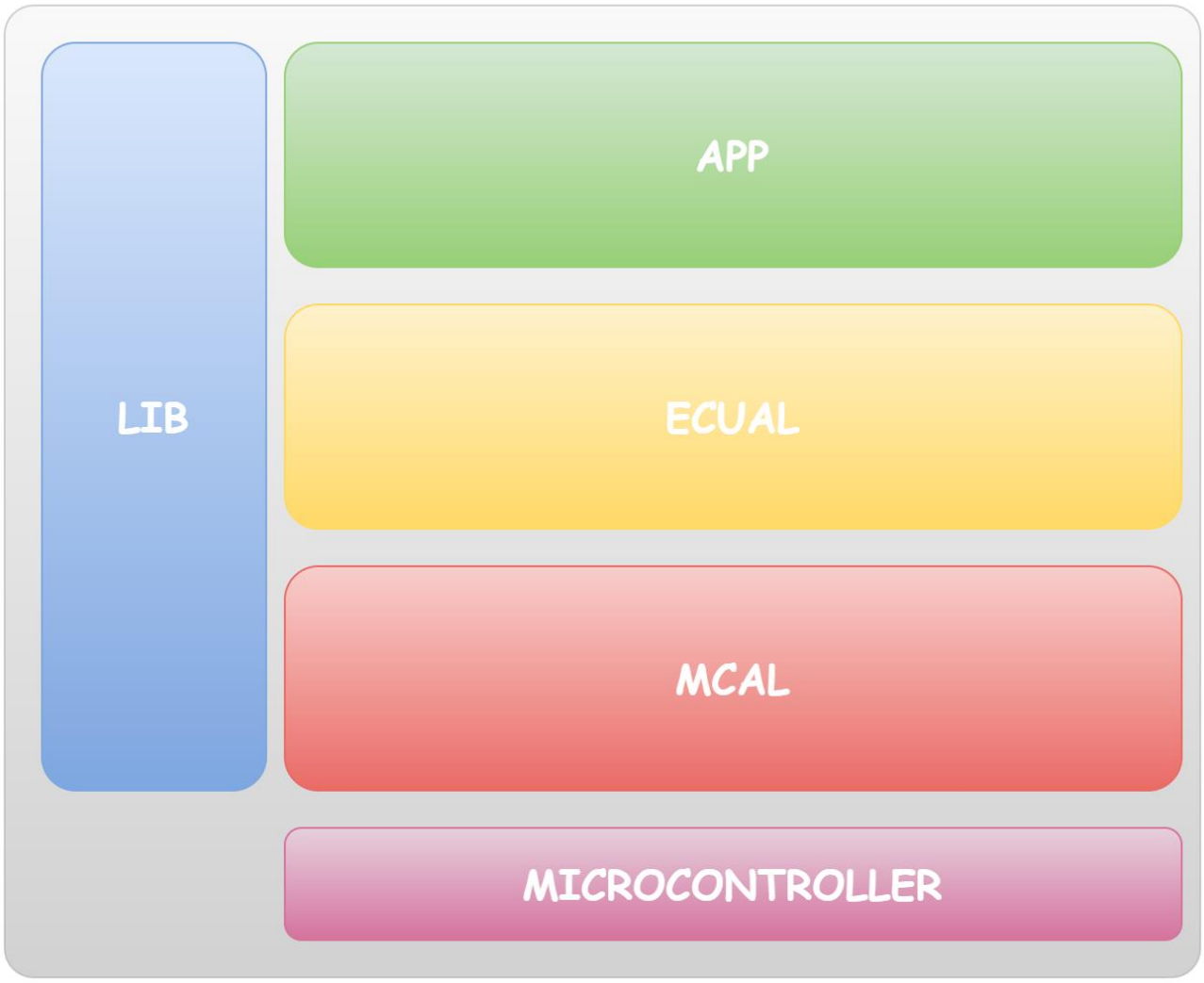
2.1.1. Definition

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

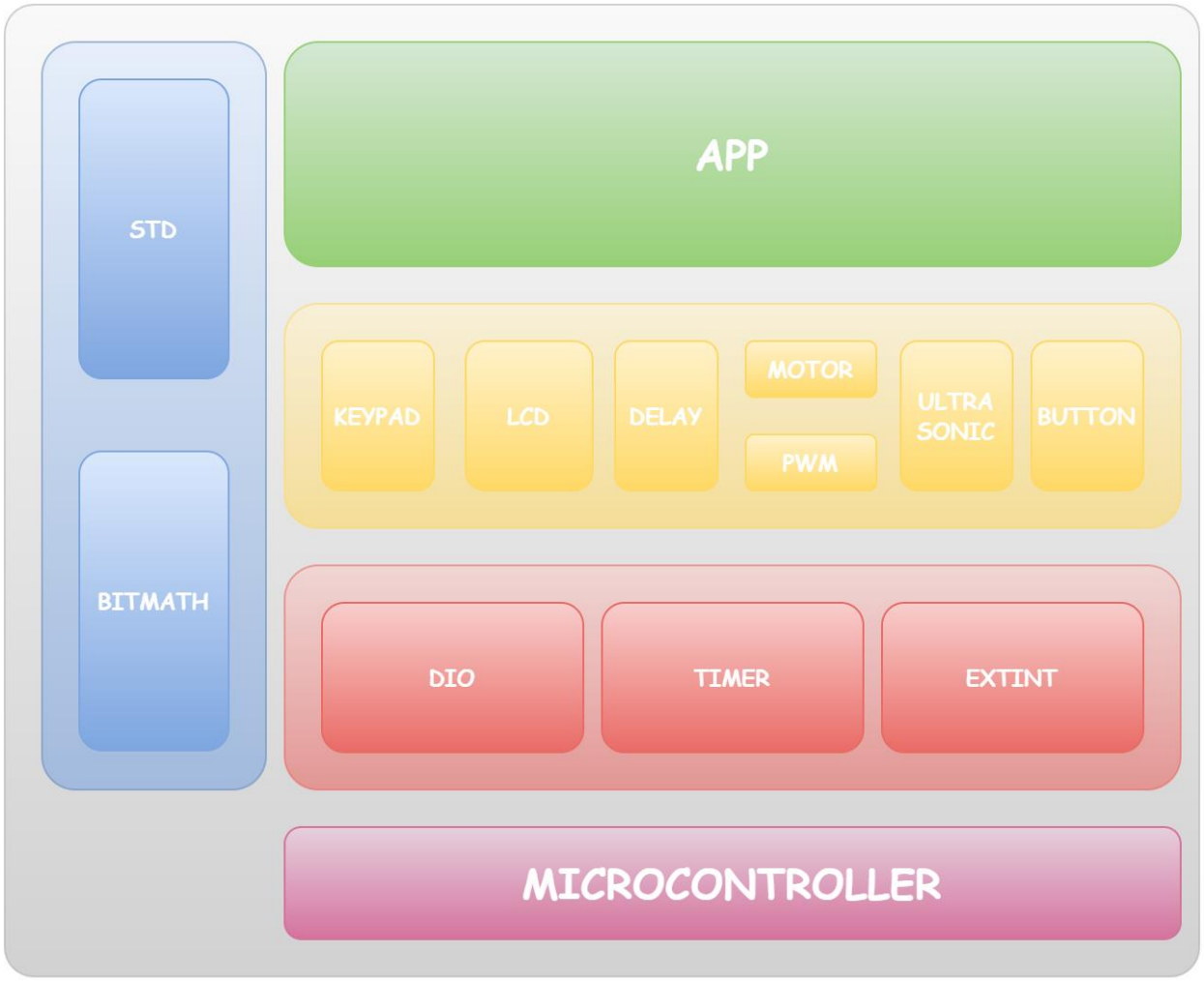
Microcontroller Abstraction Layer (MCAL) is a software module that directly accesses on-chip MCU peripheral modules and external devices that are mapped to memory, and makes the upper software layer independent of the MCU.

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


LIB
APP
ECUAL
MCAL
MICROCONTROLLER

2.1.3. Design


STD
BITMATH
APP
KEYPAD
LCD
DELAY
MOTOR
PWM
ULTRA SONIC
BUTTON
DIO
TIMER
EXTINT
MICROCONTROLLER

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. EXTINT 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. Button 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. Keypad 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. Delay Module

Delay is a software-implemented module that uses a normal timer to generate both non-blocking delays and blocking delays. it also sends to the timer call back function to be called when the timer interrupt occurs.

2.2.10. PWM Module

PWM is a software-implemented module that uses a normal timer to generate pulse width module signals through chosen pins. It has the accessibility to edit the duty cycle of the output signal too.

2.2.11. 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.

2.3. 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

```
|
| Syntax          : DIO_init(u8 u8_a_portNumber, u8 u8_a_pinNumber, u8 u8_a_direction)
| Description     : Initializes DIO pin
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : u8                                  u8_a_portNumber
|                   u8                                  u8_a_pinNumber
|                   u8                                  u8_a_direction
| Parameters (out): None
| Return value:   : en_DIO_error_t                DIO_OK  = 0
|                                                 WRONG_PORT_NUMBER = 1
|                                                 WRONG_PIN_NUMBER = 2
|                                                 WRONG_DIRECTION = 4
|

en_DIO_error_t DIO_init(u8 u8_a_portNumber, u8 u8_a_pinNumber, u8 u8_a_direction);

|
| Syntax          : DIO_write(u8 u8_a_portNumber, u8 u8_a_pinNumber, u8 u8_a_value)
| Description     : Assigns DIO pins LEVEL
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : u8                                  u8_a_portNumber
|                   u8                                  u8_a_pinNumber
|                   u8                                  u8_a_value
| Parameters (out): None
| Return value:   : en_DIO_error_t                DIO_OK  = 0
|                                                 WRONG_PORT_NUMBER = 1
|                                                 WRONG_PIN_NUMBER = 2
|                                                 WRONG_VALUE =3
|
en_DIO_error_t DIO_write(u8 u8_a_portNumber, u8 u8_a_pinNumber, u8 u8_a_value);




|
```

```
| Syntax          : DIO_toggle(u8 u8_a_portNumber, u8 u8_a_pinNumber)
| Description     : Toggles DIO pins LEVEL
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : u8                                  u8_a_portNumber
|                   u8                                  u8_a_pinNumber
| Parameters (out): None
| Return value:   : en_DIO_error_t                DIO_OK  = 0
|                                                 WRONG_PORT_NUMBER = 1
|                                                 WRONG_PIN_NUMBER = 2
|
en_DIO_error_t DIO_toggle(u8 u8_a_portNumber, u8 u8_a_pinNumber);

|
| Syntax          : DIO_read(u8 u8_a_portNumber, u8 u8_a_pinNumber, u8u8_a_value)
| Description     : Gets DIO pins LEVEL
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : u8                                            u8_a_portNumber
|                   u8                                            u8_a_pinNumber
| Parameters (out): u8                                            u8_a_value
| Return value:   : en_DIO_error_t                DIO_OK  = 0
|                                                 WRONG_PORT_NUMBER = 1
|                                                 WRONG_PIN_NUMBER = 2
|
en_DIO_error_t DIO_read(u8 u8_a_portNumber, u8 u8_a_pinNumber, u8 *u8_a_value);

|
| Syntax          : DIO_pinPullUp(u8 u8_a_portNumber, u8 u8_a_pinNumber, u8 u8_a_pullUpState);
| Description     : Assigns DIO pin's pullup state
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : u8                                            u8_a_portNumber
|                    u8                                           u8_a_pinNumber
|                    u8                                           u8_a_pullUpState
| Parameters (out): None
| Return value:   : en_DIO_error_t                DIO_OK  = 0
|                                                 WRONG_PORT_NUMBER = 1
|                                                 WRONG_PIN_NUMBER = 2
|                                                 DIO_NOK = 5
|
en_DIO_error_t DIO_pinPullUp(u8 u8_a_portNumber, u8 u8_a_pinNumber, u8 pullUpState);
```

2.3.2.2. EXTINT Driver

```
|
| Syntax          : EXTINT_init (void)
| Description     : Initializes External Interrupts
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : None
| Parameters (out): None
| Return value:   : en_EXTINT_error_t             EXTINT_OK = 0
|                                                 EXTINT_NOK = 1
|
en_EXTINT_error_t EXTINT_init (void);

|
| Syntax          : EXTINT_setCallBackInt (u8 u8_a_intNumber, void (*funPtr) (void))
| Description     : Sets callback function for a specific external interrupt
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : u8                                u8_a_intNumber
|                   void                              (*funPtr) (void)
| Parameters (out): None
| Return value:   : en_EXTINT_error_t             EXTINT_OK = 0
|                                                 EXTINT_NOK = 1
|
en_EXTINT_error_t EXTINT_setCallBackInt (u8 u8_a_intNumber, void (*funPtr) (void));
```

2.3.2.3. TIMER Driver

```
|
| Syntax          : en_TIMER_error_t TIMER_init( void )
| Description     : Initialize Timer according to preprocessed
|                    configured definitions
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : None
| Parameters (out): None
| Return value    : en_TIMER_error_t              TIMER_OK = 0
|                                                 TIMER_WRONG_TIMER_USED = 1
|                                                 TIMER_WRONG_DESIRED_TIME = 2
|                                                 TIMER_NOK = 3
|
```

```
en_TIMER_error_t TIMER_init( void );
```

```
|
|  Syntax          : en_TIMER_error_t TIMER_setTime
|                    (en_TIMER_number_t en_a_timerUsed, f32 f32_a_desiredTime)
|  Description     : set the time at which the timer interrupts
|  Sync\Async      : Synchronous
|  Reentrancy      : Reentrant
|  Parameters (in) : en_TIMER_number_t            en_a_timerUsed
|                    f32                          f32_a_desiredTime
|  Parameters (out): None
|  Return value:   : en_TIMER_error_t             TIMER_OK = 0
|                                                 TIMER_WRONG_TIMER_USED = 1
|                                                 TIMER_WRONG_DESIRED_TIME = 2
|                                                 TIMER_NOK = 3
|
```

```
en_TIMER_error_t TIMER_setTime(en_TIMER_number_t en_a_timerUsed, f32
f32_a_desiredTime);
```

```
|
|  Syntax          : en_TIMER_error_t TIMER_pwmGenerator
|             (en_TIMER_number_t en_a_timerUsed , u16 u16_a_onTime, u16 u16_a_offTime)
|  Description     : initialize the timer to generates pwm signal using
|                    normal mode
|  Sync\Async      : Synchronous
|  Reentrancy      : Reentrant
|  Parameters (in) : en_TIMER_number_t            en_a_timerUsed
|                      u16                        u16_a_onTime
|                      u16                        u16_a_offTime
|  Parameters (out): None
|  Return value:   : en_TIMER_error_t             TIMER_OK = 0
|                                                 TIMER_WRONG_TIMER_USED = 1
|                                                 TIMER_WRONG_DESIRED_TIME = 2
|                                                 TIMER_NOK = 3
|
```

```
en_TIMER_error_t TIMER_pwmGenerator(en_TIMER_number_t en_a_timerUsed , u16
u16_a_onTime, u16 u16_a_offTime);
```

```
|
| Syntax          : en_TIMER_error_t TIMER_resume(en_TIMER_number_t en_a_timerUsed)
| Description     : makes the timer to start/resume counting
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : en_TIMER_number_t             en_a_timerUsed
| Parameters (out): None
| Return value:   : en_TIMER_error_t              TIMER_OK = 0
|                                                 TIMER_WRONG_TIMER_USED = 1
|                                                 TIMER_WRONG_DESIRED_TIME = 2
|                                                 TIMER_NOK = 3
|
en_TIMER_error_t TIMER_resume(en_TIMER_number_t en_a_timerUsed);

|
| Syntax          : en_TIMER_error_t TIMER_reset(en_TIMER_number_t en_a_timerUsed)
| Description     : makes the timer to reset counting from the beginning
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : en_TIMER_number_t             en_a_timerUsed
| Parameters (out): None
| Return value:   : en_TIMER_error_t              TIMER_OK = 0
|                                                 TIMER_WRONG_TIMER_USED = 1
|                                                 TIMER_WRONG_DESIRED_TIME = 2
|                                                 TIMER_NOK = 3
|
en_TIMER_error_t TIMER_reset(en_TIMER_number_t en_a_timerUsed);

|
| Syntax          : en_TIMER_error_t TIMER_getElapsedTime
|                     (en_TIMER_number_t en_a_timerUsed, u32* u32_a_elapsedTime)
| Description     : returns the elapsed time since the timer started
|                    from the beginning in microseconds
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : en_TIMER_number_t             en_a_timerUsed
| Parameters (out): u32                           u32_a_elapsedTime
| Return value:   : en_TIMER_error_t              TIMER_OK = 0
|                                                 TIMER_WRONG_TIMER_USED = 1
|                                                 TIMER_WRONG_DESIRED_TIME = 2
|                                                 TIMER_NOK = 3
|
en_TIMER_error_t TIMER_getElapsedTime(en_TIMER_number_t en_a_timerUsed, u32*
u32_a_elapsedTime);
```

```
|
| Syntax          : en_TIMER_error_t TIMER_pause(en_TIMER_number_t en_a_timerUsed)
| Description     : makes the timer to pause counting
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : en_TIMER_number_t             en_a_timerUsed
| Parameters (out): None
| Return value:   : en_TIMER_error_t       TIMER_OK = 0
|                                          TIMER_WRONG_TIMER_USED = 1
|                                          TIMER_WRONG_DESIRED_TIME = 2
|                                          TIMER_NOK = 3
|
en_TIMER_error_t TIMER_pause(en_TIMER_number_t en_a_timerUsed);

|
| Syntax          : en_TIMER_error_t TIMER_disableInterrupt(en_TIMER_number_t en_a_timerUsed)
| Description     : Disables timer's interrupts
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : en_TIMER_number_t             en_a_timerUsed
| Parameters (out): None
| Return value:   : en_TIMER_error_t       TIMER_OK = 0
|                                          TIMER_WRONG_TIMER_USED = 1
|                                          TIMER_WRONG_DESIRED_TIME = 2
|                                          TIMER_NOK = 3
|
en_TIMER_error_t TIMER_disableInterrupt(en_TIMER_number_t en_a_timerUsed);

|
| Syntax          : en_TIMER_error_t TIMER_enableInterrupt(en_TIMER_number_t en_a_timerUsed)
| Description     : Enables timer's interrupts
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : en_TIMER_number_t             en_a_timerUsed
| Parameters (out): None
| Return value:   : en_TIMER_error_t       TIMER_OK = 0
|                                          TIMER_WRONG_TIMER_USED = 1
|                                          TIMER_WRONG_DESIRED_TIME = 2
|                                          TIMER_NOK = 3
|
en_TIMER_error_t TIMER_enableInterrupt(en_TIMER_number_t en_a_timerUsed);
```

```
|
| Syntax          : en_TIMER_error_t TIMER_setCallBack
|                     (en_TIMER_number_t en_a_timerUsed, void (*funPtr)(void))
| Description     : sets the call back function for a specific timer
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : en_TIMER_number_t             en_a_timerUsed
|                    void                         (*funPtr)(void)
| Parameters (out): None
| Return value:   : en_TIMER_error_t       TIMER_OK = 0
|                                          TIMER_WRONG_TIMER_USED = 1
|                                          TIMER_WRONG_DESIRED_TIME = 2
|                                          TIMER_NOK = 3
|
en_TIMER_error_t TIMER_setCallBack(en_TIMER_number_t en_a_timerUsed, void
(*funPtr)(void));

|
| Syntax          : en_TIMER_error_t TIMER_setDelayTime
|                                  (en_TIMER_number_t en_a_timerUsed, f32 f32_a_timeInMS)
| Description     : Set delay time for blocking delay using a specific timer
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : en_TIMER_number_t             en_a_timerUsed
|                      f32                        f32_a_timeInMS
| Parameters (out): None
| Return value:   : en_TIMER_error_t       TIMER_OK = 0
|                                          TIMER_WRONG_TIMER_USED = 1
|                                          TIMER_WRONG_DESIRED_TIME = 2
|                                          TIMER_NOK = 3
|
en_TIMER_error_t TIMER_setDelayTime(en_TIMER_number_t en_a_timerUsed, f32
f32_a_timeInMS);

|
| Syntax          : en_TIMER_error_t TIMER_setPwmOnCallBack
|                                  (en_TIMER_number_t en_a_timerUsed, void (*funPtr)(void))
| Description     : Set callback function for the task done while signal is high
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : en_TIMER_number_t             en_a_timerUsed
|                    void                         (*funPtr)(void)
| Parameters (out): None
| Return value:   : en_TIMER_error_t       TIMER_OK = 0
|                                          TIMER_WRONG_TIMER_USED = 1
|                                          TIMER_WRONG_DESIRED_TIME = 2
|                                          TIMER_NOK = 3
|
en_TIMER_error_t TIMER_setPwmOnCallBack(en_TIMER_number_t en_a_timerUsed, void
(*funPtr)(void));
```

```
|
| Syntax          : en_TIMER_error_t TIMER_setPwmOffCallBack
|                                    (en_TIMER_number_t en_a_timerUsed, void (*funPtr)(void))
| Description     : Set callback function for the task done while signal is low
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : en_TIMER_number_t              en_a_timerUsed
|                     void                         (*funPtr)(void)
| Parameters (out): None
| Return value:   : en_TIMER_error_t        TIMER_OK = 0
|                                           TIMER_WRONG_TIMER_USED = 1
|                                           TIMER_WRONG_DESIRED_TIME = 2
|                                           TIMER_NOK = 3
|
en_TIMER_error_t TIMER_setPwmOffCallBack(en_TIMER_number_t en_a_timerUsed, void
(*funPtr)(void));
```

2.3.3. HAL APIs

2.3.3.1. LCD APIs

```
|
| Syntax          : LCD_init (void)
| Description     : Initializes LCD
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : None
| Parameters (out): None
| Return value:   : en_LCD_error_t          LCD_OK = 0
|                                           LCD_NOK = 1
|

en_LCD_error_t LCD_init                             (void);

|
| Syntax          : LCD_writeChar (u8 u8_a_char)
| Description     : writes character on LCD
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : u8                              u8_a_char
| Parameters (out): None
| Return value:   : en_LCD_error_t          LCD_OK = 0
|                                           LCD_NOK = 1
|
en_LCD_error_t LCD_writeChar        (u8 u8_a_char);

|
| Syntax          : LCD_writeCmd(u8 u8_a_cmd)
| Description     : Do command on LCD
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : u8                              u8_a_cmd
| Parameters (out): None
| Return value:   : en_LCD_error_t          LCD_OK = 0
|                                           LCD_NOK = 1
|
en_LCD_error_t LCD_writeCmd                 (u8 u8_a_cmd);

|
| Syntax          : LCD_writeNum (s32 s32_a_num)
| Description     : Writes an signed number on LCD
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : u8                              s32_a_num
| Parameters (out): None
| Return value:   : en_LCD_error_t          LCD_OK = 0
|                                                                   LCD_NOK = 1
|
en_LCD_error_t LCD_writeNum                 (s32 s32_a_num);
```

```
|
| Syntax          : LCD_clr(void)
| Description     : Clears LCD
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : None
| Parameters (out): None
| Return value:   : en_LCD_error_t          LCD_OK = 0
|                                           LCD_NOK = 1
|
en_LCD_error_t LCD_clr                              (void);

|
| Syntax          : LCD_writeString (u8* u8_a_string)
| Description     : Writes a string on LCD
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : u8*                     u8_a_string
| Parameters (out): None
| Return value:   : en_LCD_error_t          LCD_OK = 0
|                                           LCD_NOK = 1
|
en_LCD_error_t LCD_writeString              (u8* u8_a_string);

|
| Syntax          : LCD_goTo(u8 u8_a_row, u8 u8_a_Col)
| Description     : Moves cursor on lcd to the desired location
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : u8                      u8_a_row,
|                   u8                      u8_a_Col
| Parameters (out): None
| Return value:   : en_LCD_error_t          LCD_OK = 0
|                                           LCD_NOK = 1
|
en_LCD_error_t LCD_goTo                             (u8 u8_a_row, u8 u8_a_Col);
```

2.3.3.2. BUTTON APIs

```
|
| Syntax          : BUTTON_init (void)
| Description     : Initializes buttons
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : None
| Parameters (out): None
| Return value:   : en_BUTTON_error_t             BUTTON_OK = 0
|                                                 WRONG_BUTTON_PORT = 1
|                                                 WRONG_BUTTON_PIN = 2
|                                                 WRONG_BUTTON_NUMBER = 3
|
en_BUTTON_error_t BUTTON_init(void);

|
| Syntax          : BUTTON_read(u8 u8_a_buttonNumber, u8* u8_a_buttonState)
| Description     : Gets a specific button's state
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : u8                            u8_a_buttonNumber
| Parameters (out): u8*                           u8_a_buttonState
| Return value:   : en_BUTTON_error_t             BUTTON_OK = 0
|                                                 WRONG_BUTTON_PORT = 1
|                                                 WRONG_BUTTON_PIN = 2
|                                                 WRONG_BUTTON_NUMBER = 3
|
en_BUTTON_error_t BUTTON_read(u8 u8_a_buttonNumber, u8 *u8_a_buttonState);
```

2.3.3.3. DCM APIs

```
|
| Syntax          : en_DCM_error_t DCM_init (void)
| Description     : Initializes DCM module
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : None
| Parameters (out): None
| Return value:   : en_DCM_error_t                DCM_OK = 0
|                                                         DCM_NOK = 1
|
en_DCM_error_t DCM_init                           (void);
```

```
| Syntax          : en_DCM_error_t DCM_setDirection
|       (en_DCM_number_t en_a_dcmNumber, en_DCM_direction_t en_a_direction)
| Description     : Sets Directions for the a specific DCM
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : en_DCM_number_t               en_a_dcmNumber
|                   en_DCM_direction_t            en_a_direction
| Parameters (out): None
| Return value:   : en_DCM_error_t                DCM_OK = 0
|                                                 DCM_NOK = 1
|
en_DCM_error_t DCM_setDirection (en_DCM_number_t en_a_dcmNumber, en_DCM_direction_t
en_a_direction);

|
| Syntax          : en_DCM_error_t DCM_speed (u8 u8_a_speed)
| Description     : Sets speed for DCMs
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : u8                            u8_a_speed
| Parameters (out): None
| Return value:   : en_DCM_error_t                DCM_OK = 0
|                                                 DCM_NOK = 1
|
en_DCM_error_t DCM_speed                    (u8 u8_a_speed);

|
| Syntax          : en_DCM_error_t DCM_start (void)
| Description     : Starts DCMs to rotate
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : None
| Parameters (out): None
| Return value:   : en_DCM_error_t                DCM_OK = 0
|                                                 DCM_NOK = 1
|
en_DCM_error_t DCM_start                    (void);

|
| Syntax          : en_DCM_error_t DCM_stop (void)
| Description     : Stops DCMs from rotating
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : None
| Parameters (out): None
| Return value:   : en_DCM_error_t                DCM_OK = 0
|                                                 DCM_NOK = 1
|
en_DCM_error_t DCM_stop                         (void);
```

2.3.3.4. KEYPAD APIs

```
|
| Syntax          : KEYPAD_init (void)
| Description     : Initializes keypad
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : None
| Parameters (out): None
| Return value:   : en_KEYPAD_error_t             KEYPAD_OK = 0
|                                                 KEYPAD_NOK = 1
|
en_KEYPAD_error_t KEYPAD_init (void);

|
| Syntax          : KEYPAD_read (u8* u8_a_keyPressed)
| Description     : gets the value of key pressed
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : None
| Parameters (out): u8*                           u8_a_keyPressed
| Return value:   : en_KEYPAD_error_t             KEYPAD_OK = 0
|                                                 KEYPAD_NOK = 1
|
en_KEYPAD_error_t KEYPAD_read (u8 *u8_a_keyPressed);
```

2.3.3.5. ICU APIs

```
|
| Syntax          : ICU_init (void)
| Description     : Initializes ICU
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : None
| Parameters (out): None
| Return value:   : en_ICU_error_t          ICU_OK = 0
|                                           ICU_NOK = 1
|
en_ICU_error_t ICU_init (void);

|
| Syntax          : ICU_getElapsedTime (u32 *u32_a_elapsedTime)
| Description     : gets the value of the elapsed time from the start to end of signal
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : None
| Parameters (out): u32*                          u32_a_elapsedTime
| Return value:   : en_ICU_error_t          ICU_OK = 0
|                                                 ICU_NOK = 1
|
en_ICU_error_t ICU_getElapsedTime (u32 *u32_a_elapsedTime);
```

```
|
| Syntax          : ICU_task (void)
| Description     : ICU task sent to interrupt callback function
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : None
| Parameters (out): None
| Return value:   : None
|
void ICU_task (void);
```

2.3.3.6. Ultrasonic APIs

```
|
| Syntax          : US_init (void)
| Description     : Initializes UltrasSonic
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : None
| Parameters (out): None
| Return value:   : en_US_error_t          US_OK = 0
|                                          US_NOK = 1
|
en_US_error_t US_init (void);

|
| Syntax          : US_getDistance (u16 *u16_a_distance)
| Description     : gets the distance between the object and the obstacle
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : None
| Parameters (out): u16*                          u16_a_distance
| Return value:   : en_US_error_t                 US_OK = 0
|                                                  US_NOK = 1
|
en_US_error_t US_getDistance (u16 *u16_a_distance);
```

2.3.3.7. Delay APIs

```
|
| Syntax          : en_DELAY_error_t DELAY_init (void)
| Description     : Initializes delay module
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : None
| Parameters (out): None
| Return value:   : en_DELAY_error_t          DELAY_OK = 0
|                                             DELAY_NOK = 1
|
en_DELAY_error_t DELAY_init   (void);

|
| Syntax          : en_DELAY_error_t DELAY_setTime (f32 f32_a_timeInMS)
| Description     : Sets blocking delay time
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : f32                       f32_a_timeInMS
| Parameters (out): None
| Return value:   : en_DELAY_error_t          DELAY_OK = 0
|                                             DELAY_NOK = 1
|
en_DELAY_error_t DELAY_setTime (f32 f32_a_timeInMS);

|
| Syntax          : en_DELAY_error_t DELAY_setTime (f32 f32_a_timeInMS)
| Description     : Sets non-blocking delay time
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : f32                       f32_a_timeInMS
| Parameters (out): None
| Return value:   : en_DELAY_error_t          DELAY_OK = 0
|                                             DELAY_NOK = 1
|
en_DELAY_error_t DELAY_setTimeNonBlocking (f32 f32_a_timeInMs);

|
| Syntax          : en_DELAY_error_t DELAY_setCallBack (void (*funPtr)(void))
| Description     : Sets call back for the function which gonna be performed
|                   when the nonblocking delay times out
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : void                      (*funPtr)(void)
| Parameters (out): None
| Return value:   : en_DELAY_error_t          DELAY_OK = 0
|                                             DELAY_NOK = 1
|
en_DELAY_error_t DELAY_setCallBack (void (*funPtr)(void));
```

2.3.3.8. PWM APIs

```
|
| Syntax          : en_PWM_error_t PWM_init (void)
| Description     : Initializes PWM module
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : None
| Parameters (out): None
| Return value:   : en_PWM_error_t                  PWM_OK = 0
|                                                   PWM_NOK = 1
|
en_PWM_error_t PWM_init (void);

|
| Syntax          : en_PWM_error_t PWM_setDutyCycle (u8 u8_a_speed)
| Description     : Sets PWM duty cycle
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : u8                              u8_a_speed
| Parameters (out): None
| Return value:   : en_PWM_error_t                  PWM_OK = 0
|                                                   PWM_NOK = 1
|
en_PWM_error_t PWM_setDutyCycle (u8 u8_a_speed);

|
| Syntax          : en_PWM_error_t PWM_start (void)
| Description     : Starts PWM
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : None
| Parameters (out): None
| Return value:   : en_PWM_error_t                  PWM_OK = 0
|                                                   PWM_NOK = 1
|
en_PWM_error_t PWM_start (void);

|
| Syntax          : en_PWM_error_t PWM_stop (void)
| Description     : Stops PWM
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : None
| Parameters (out): None
| Return value:   : en_PWM_error_t                  PWM_OK = 0
|                                                   PWM_NOK = 1
|
en_PWM_error_t PWM_stop (void);
```

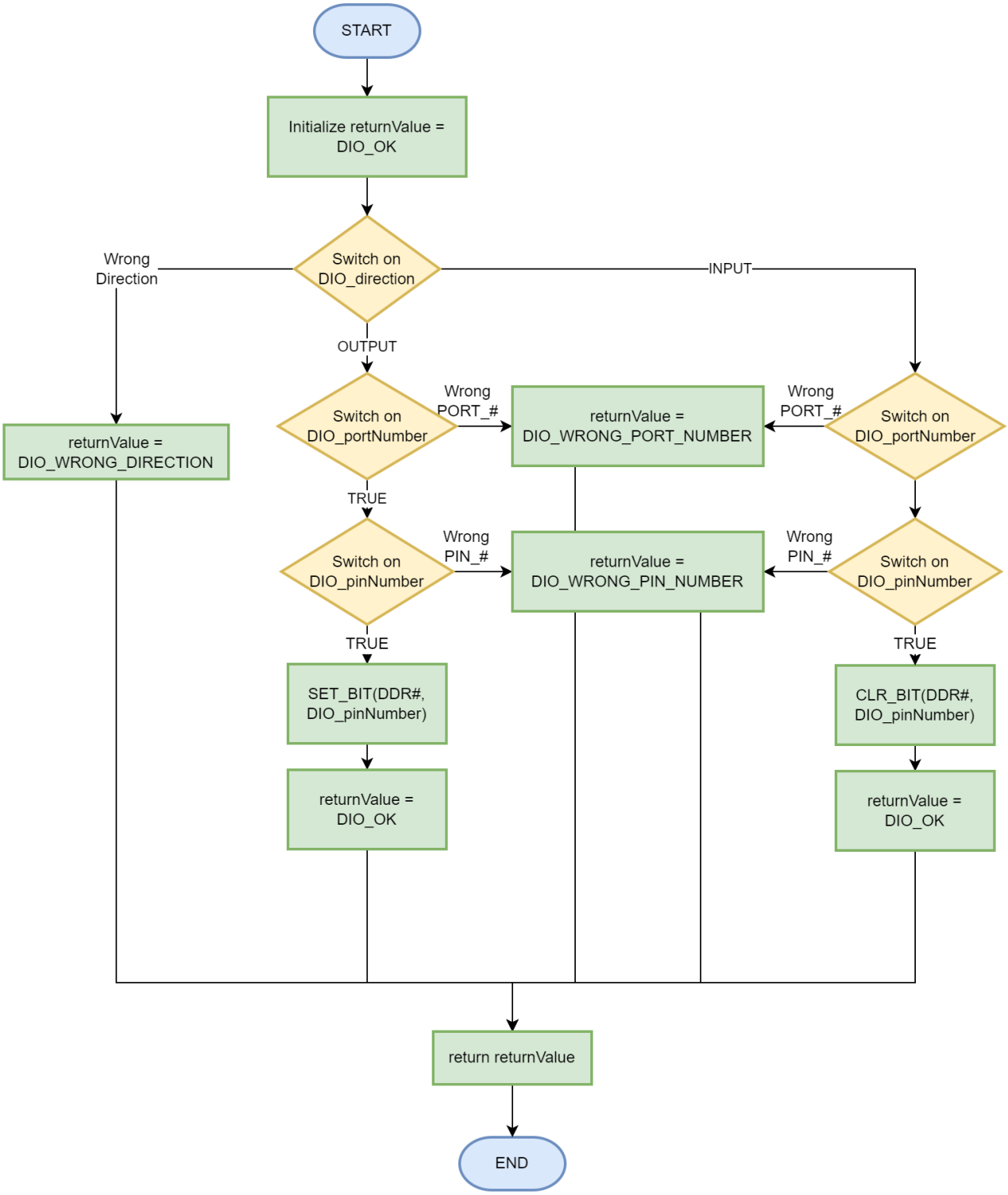
```
|
| Syntax          : en_PWM_error_t PWM_onTask (void)
| Description     : task called by the interrupt when the signal is in HIGH phase
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : None
| Parameters (out): None
| Return value:   : en_PWM_error_t                  PWM_OK = 0
|                                                   PWM_NOK = 1
|
void PWM_onTask (void);

|
| Syntax          : en_PWM_error_t PWM_onTask (void)
| Description     : task called by the interrupt when the signal is in LOW phase
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : None
| Parameters (out): None
| Return value:   : en_PWM_error_t                  PWM_OK = 0
|                                                   PWM_NOK = 1
|
void PWM_offTask (void);
```

3. Low Level Design

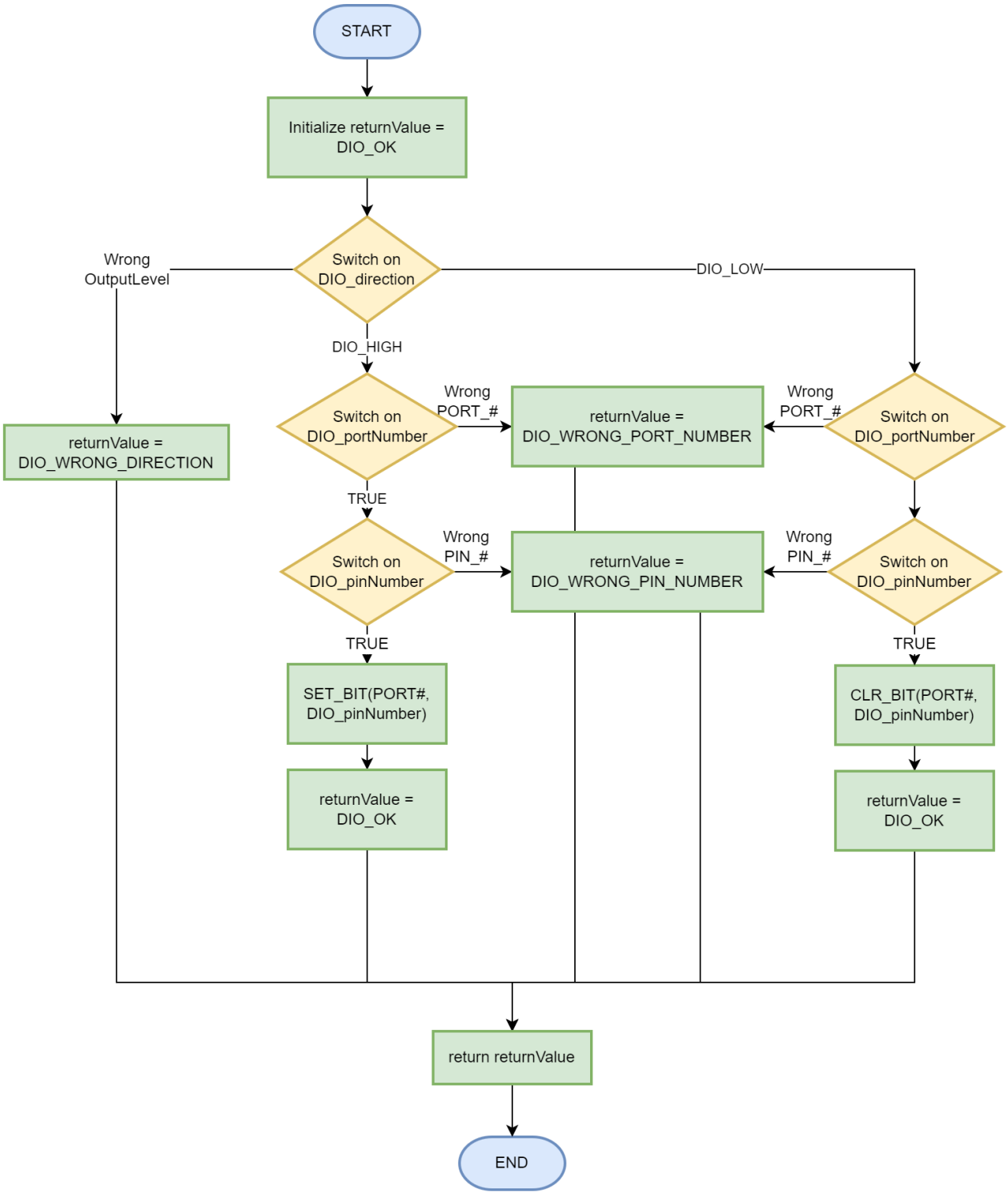
3.1. MCAL Layer

3.1.1. DIO Module

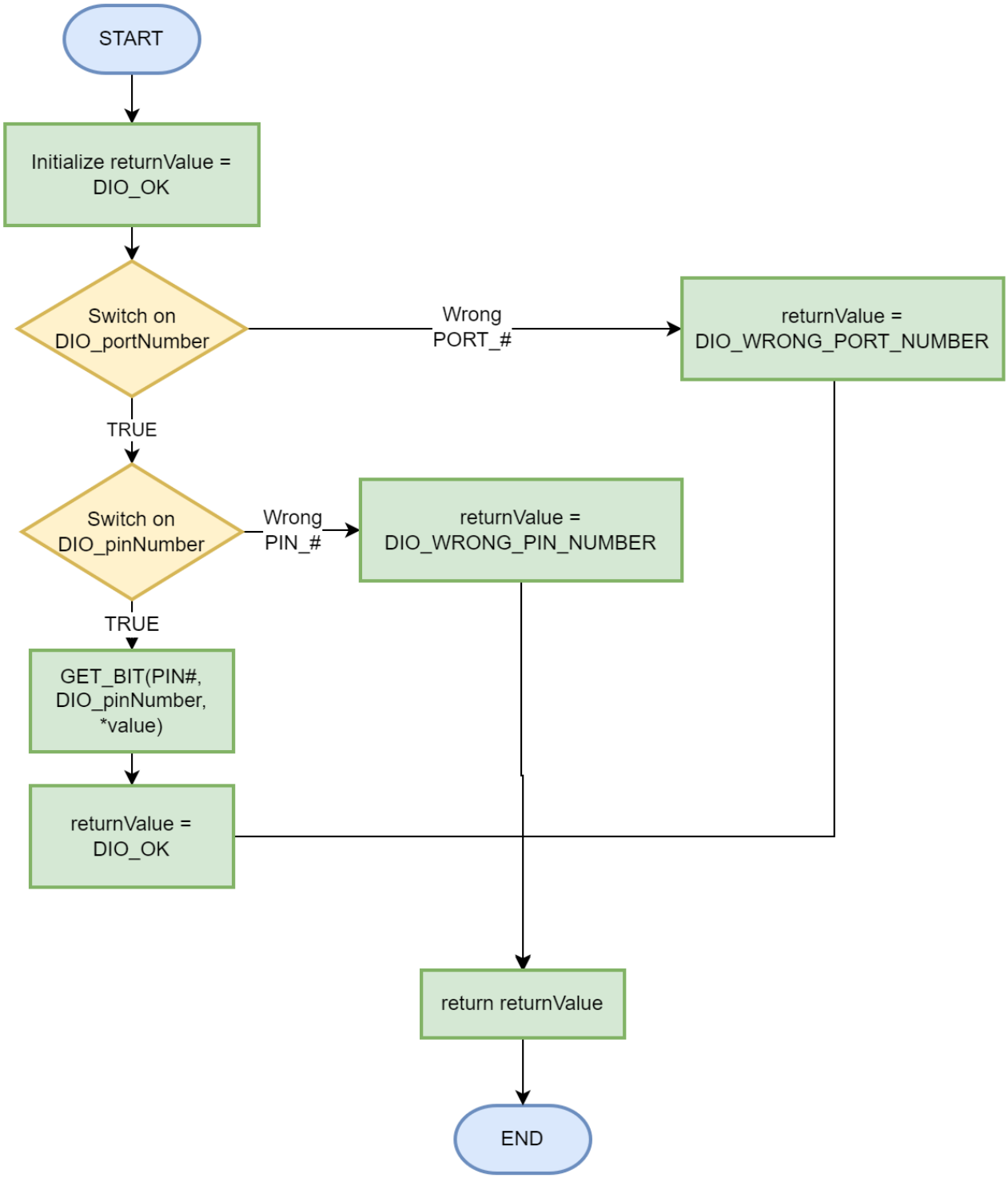
3.1.1.1. DIO_init



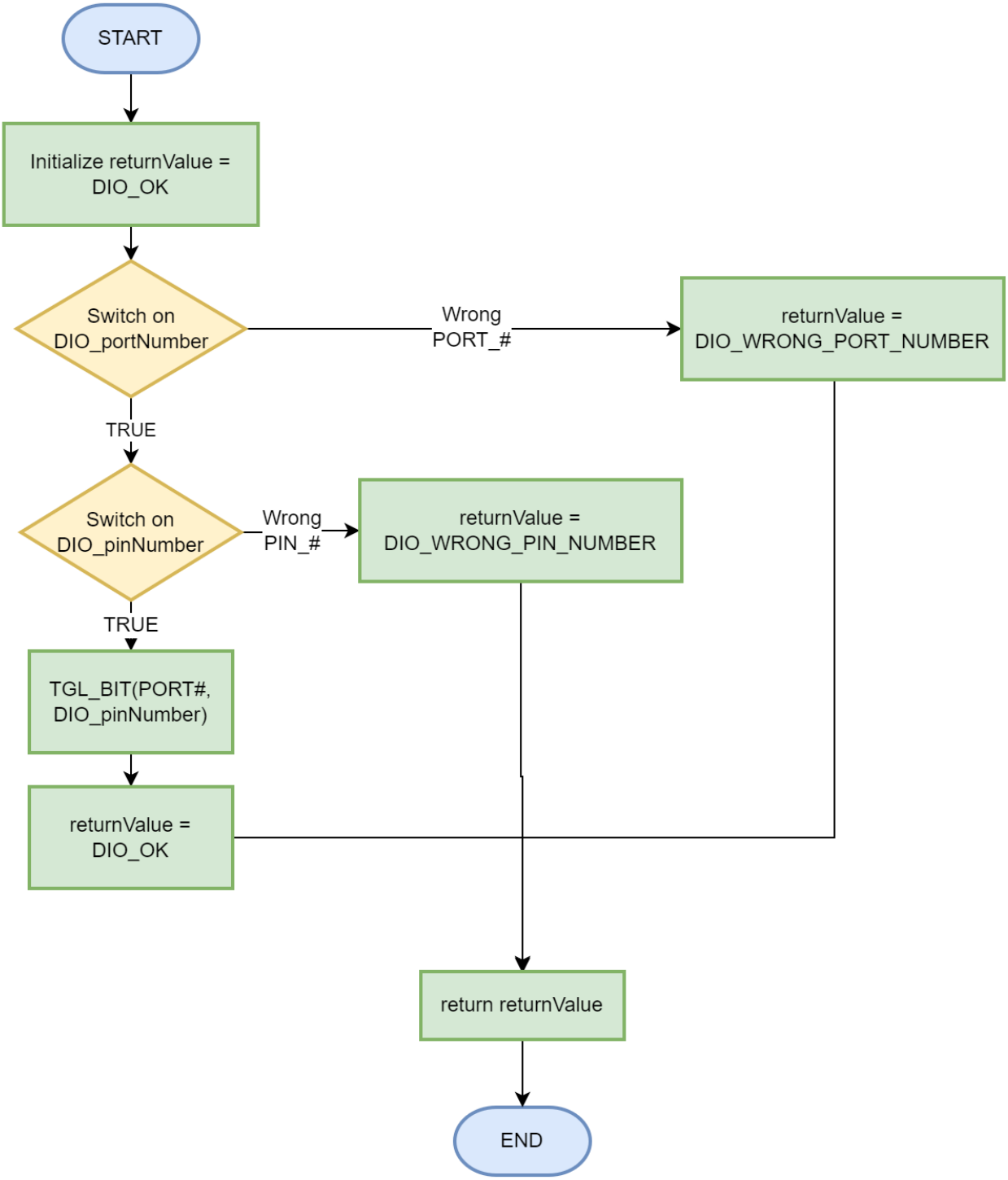
3.1.1.2. DIO_write



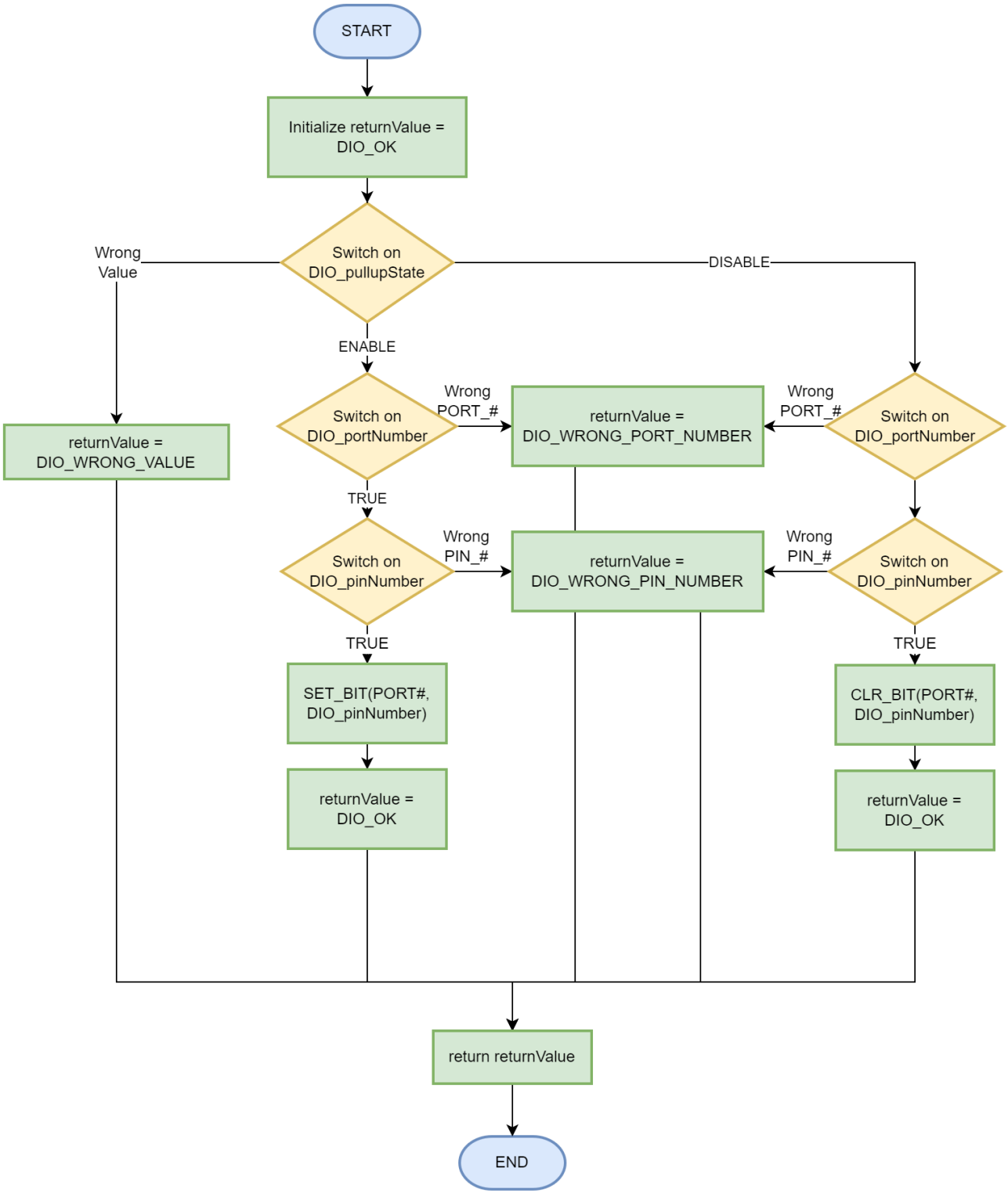
3.1.1.3. DIO_read



3.1.1.4. DIO_toggle

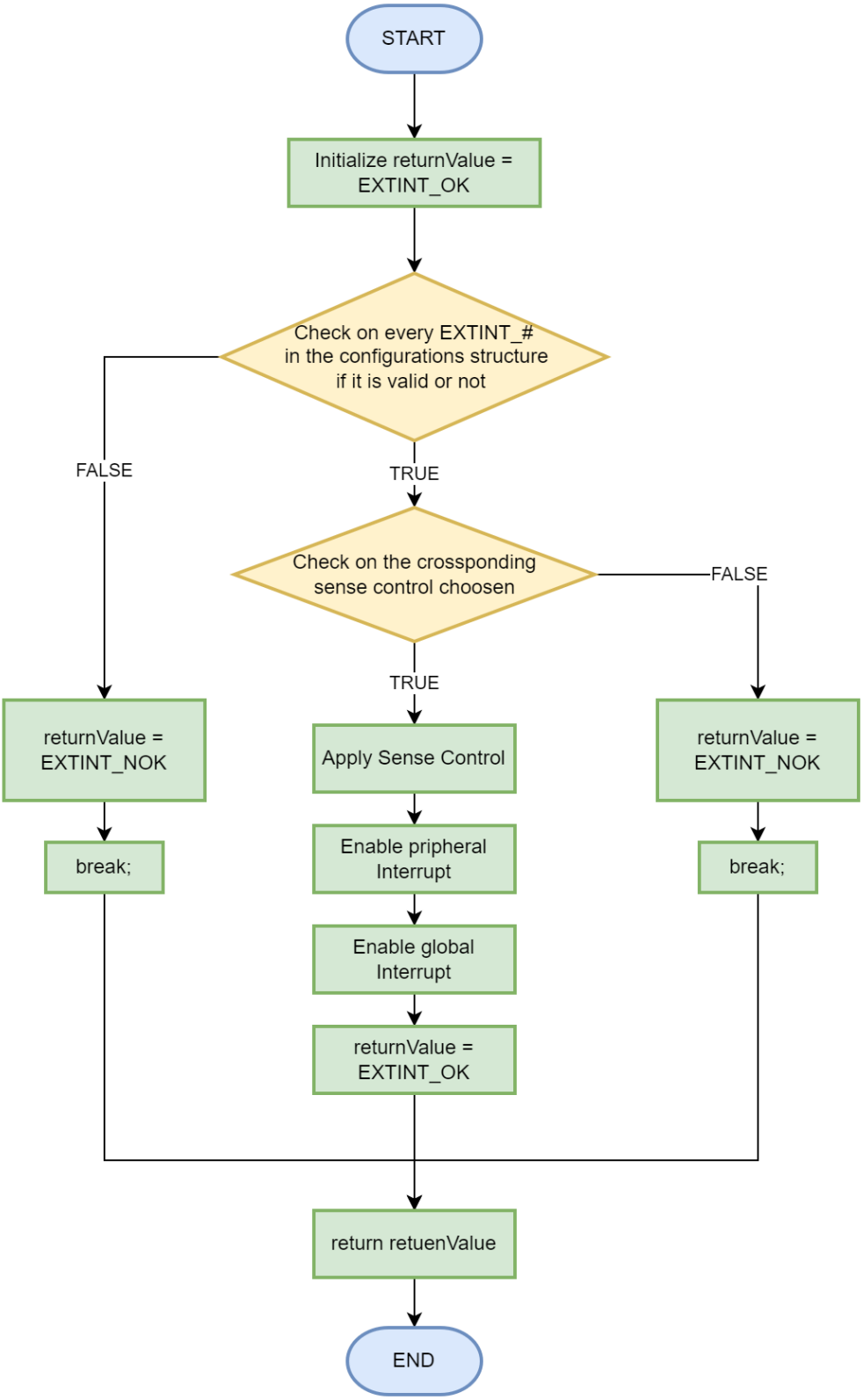


3.1.1.5. DIO_pinPullUp

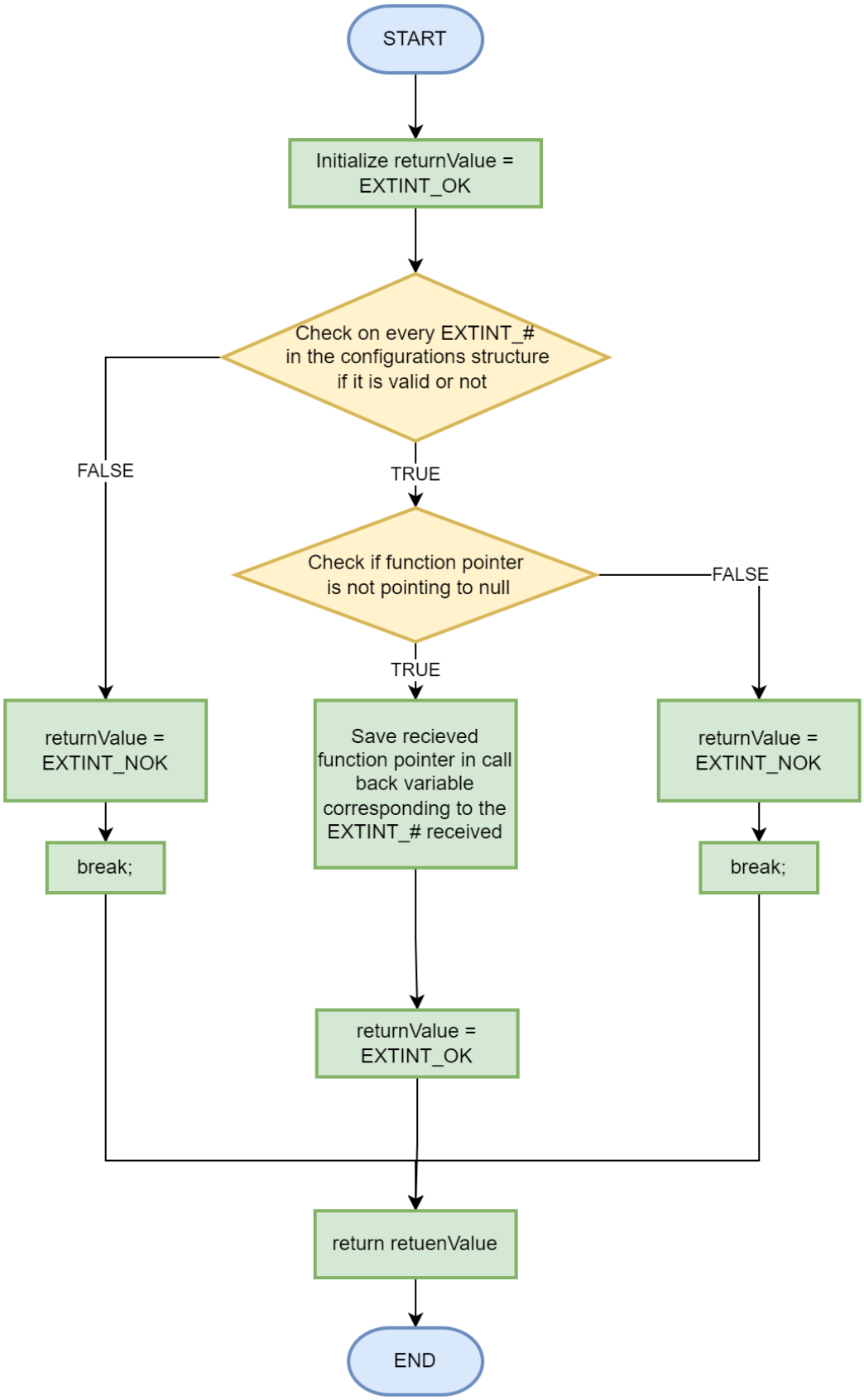


3.1.2. EXTINT Module

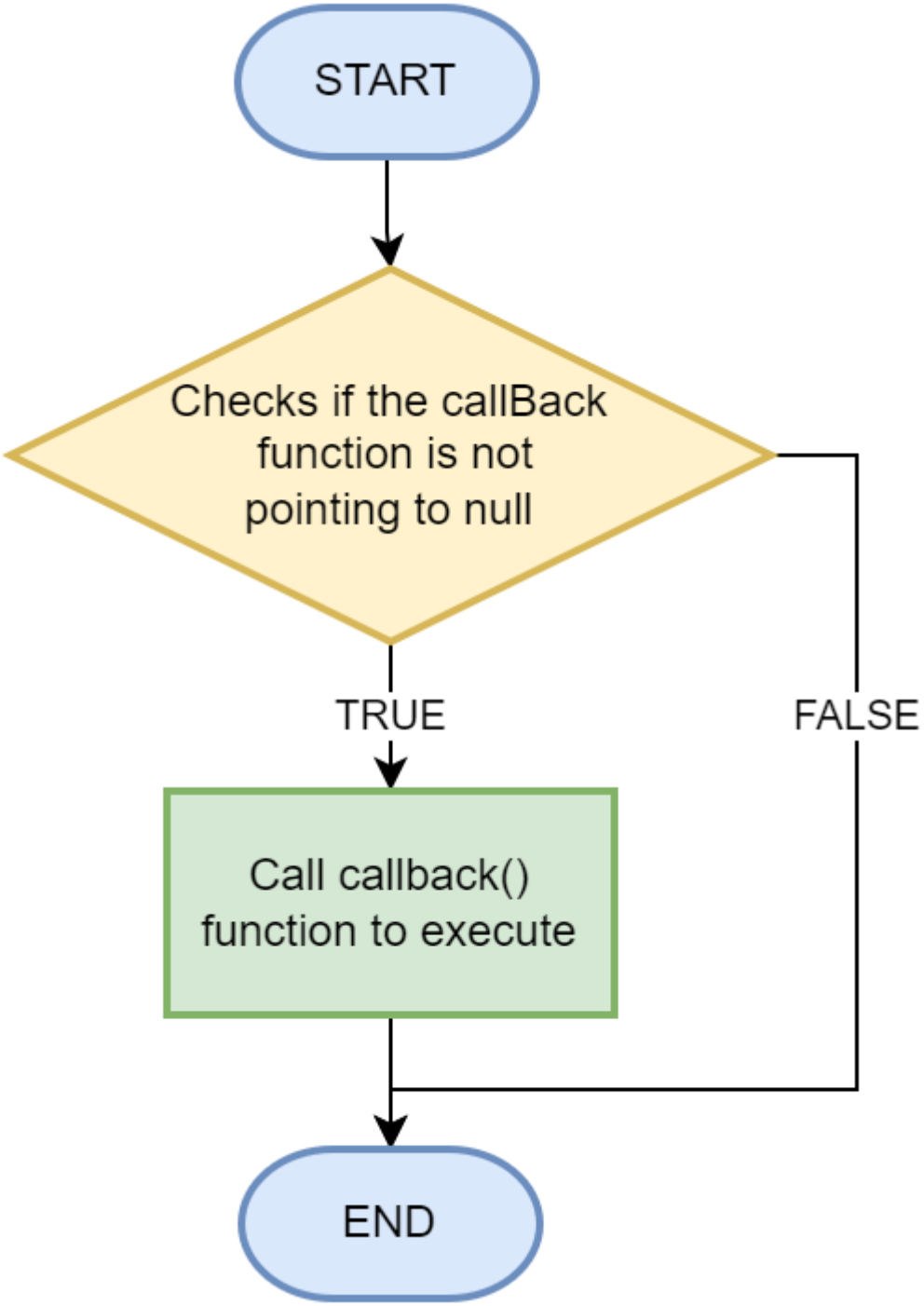
3.1.2.1. EXTINT_init



3.1.2.2. EXTINT_setCallBack

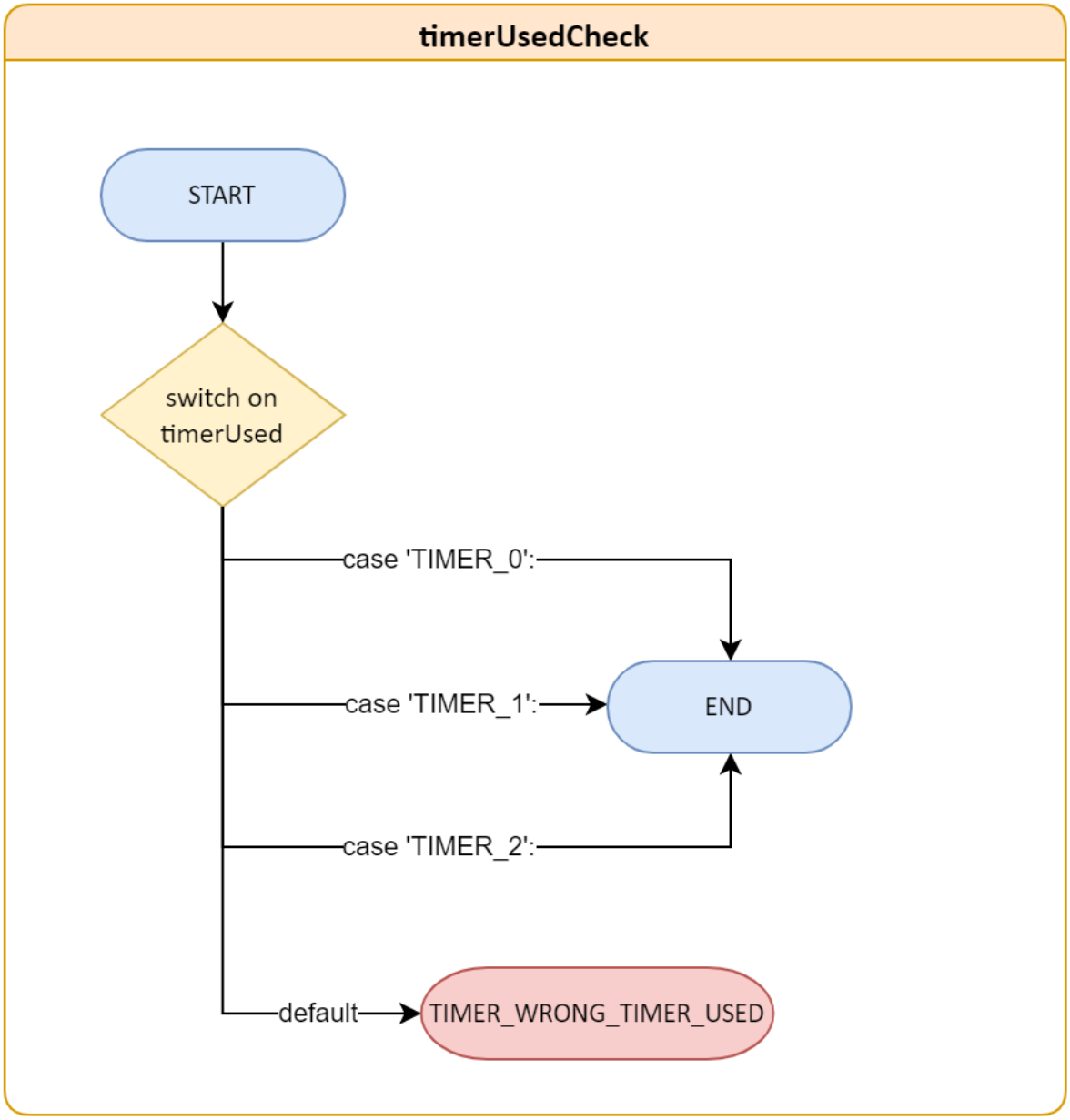


3.1.2.3. ISR(EXT_INT_#)

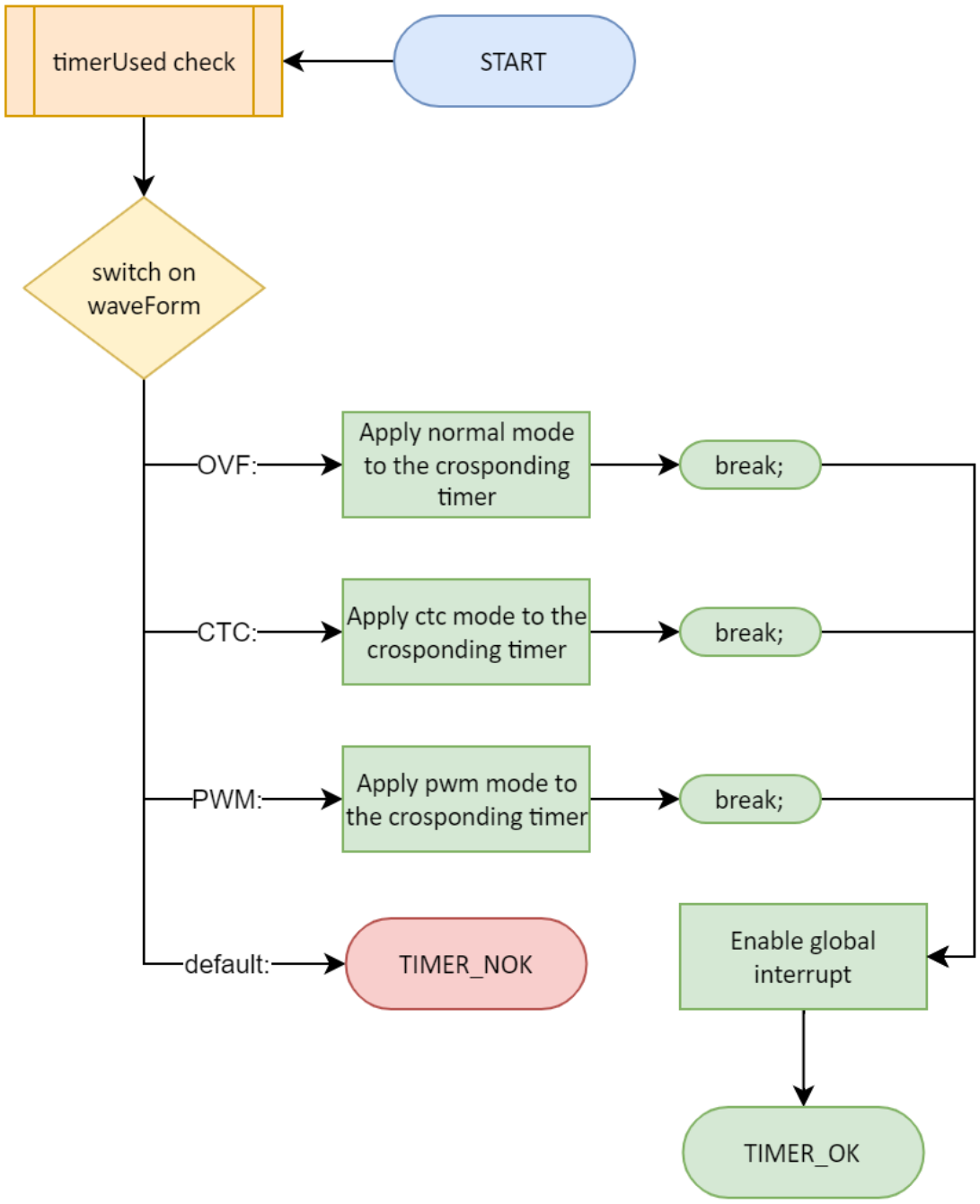


3.1.3. Timer Module

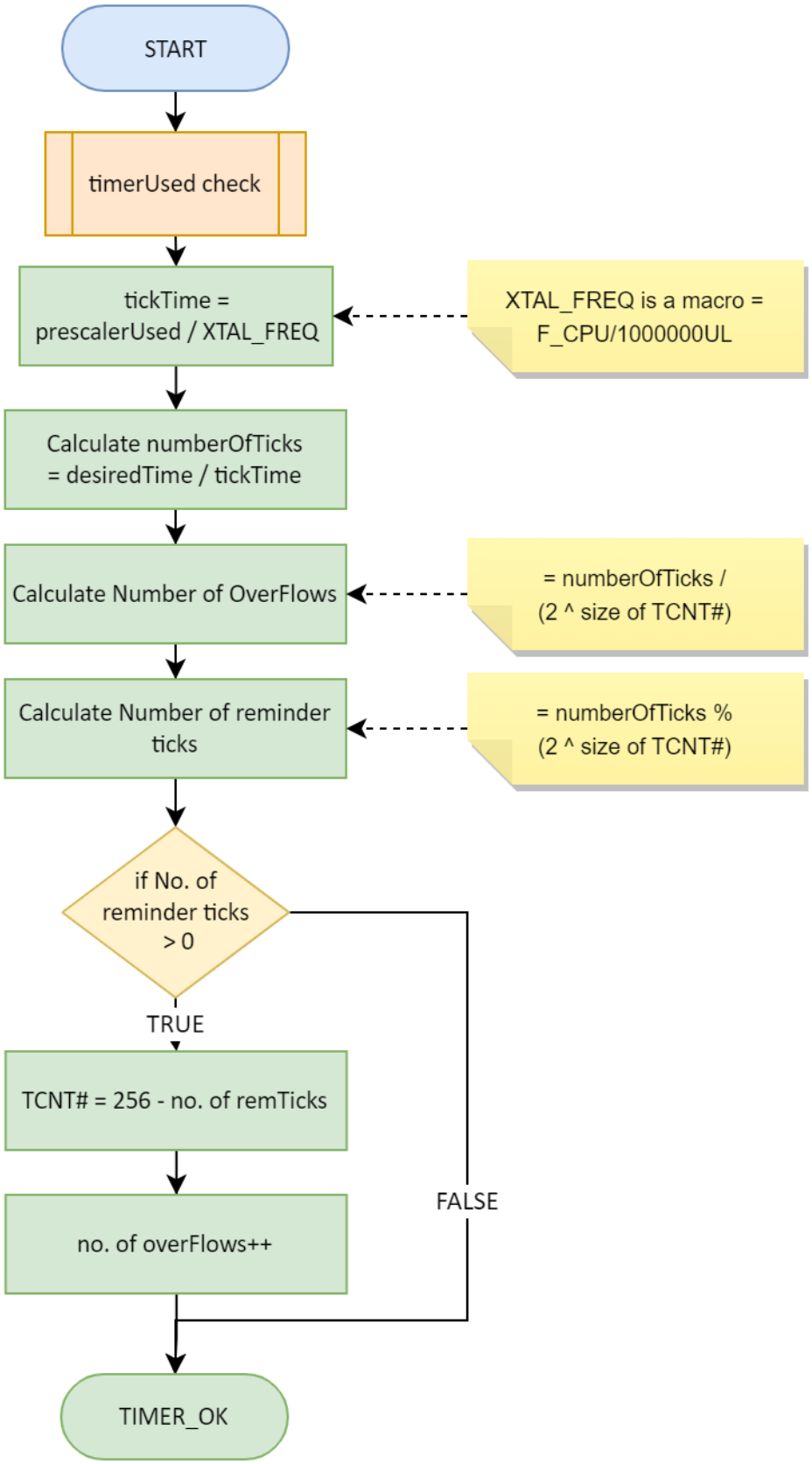
3.1.3.1. sub process



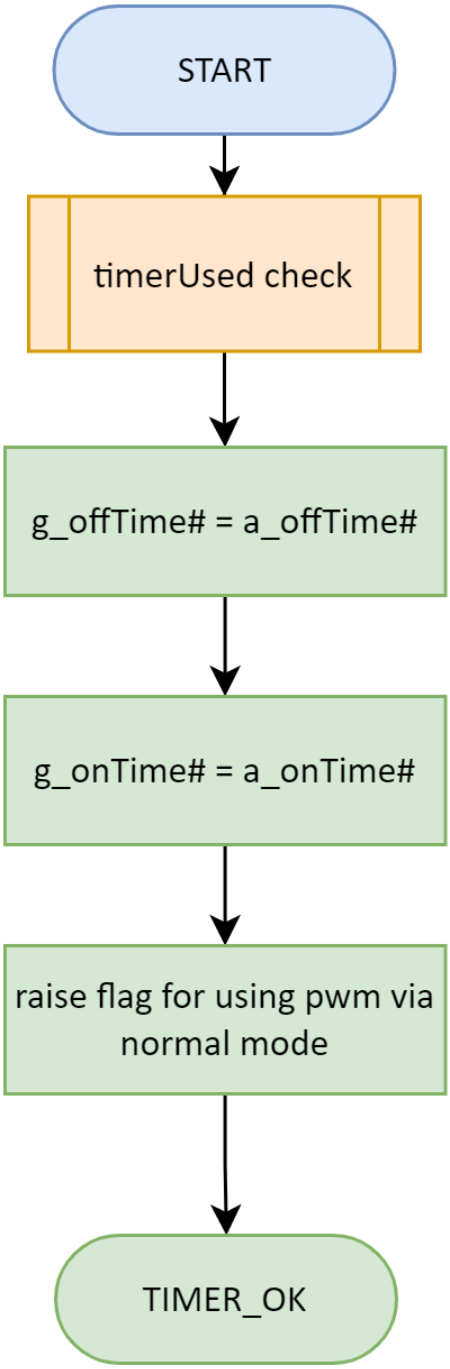
3.1.3.2. TIMER_init



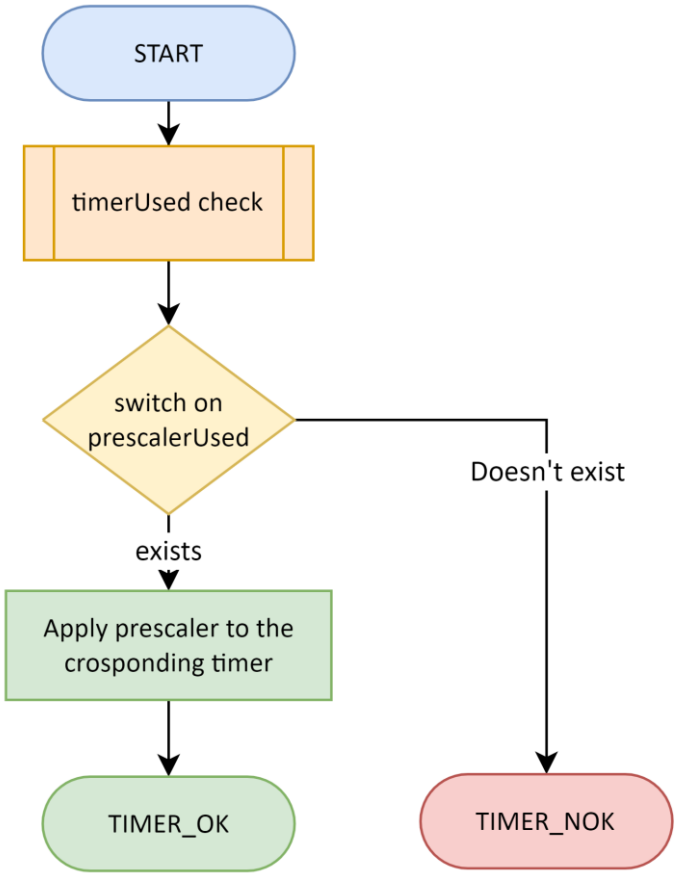
3.1.3.3. TIMER_setTime



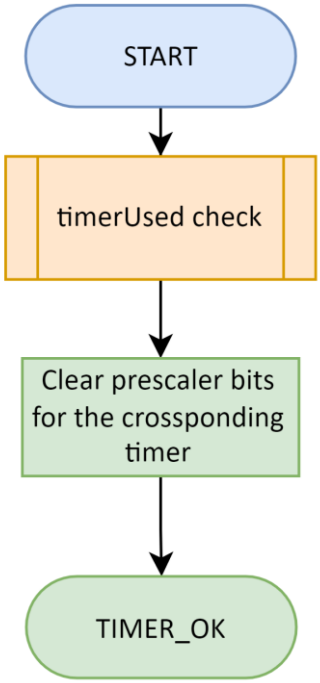
3.1.3.4. TIMER_pwmGenerator



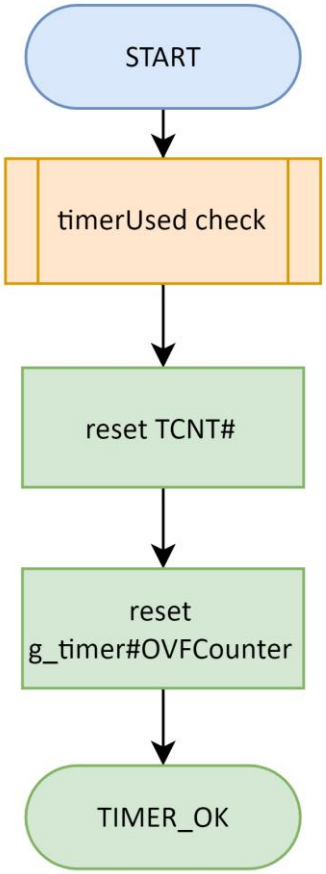
3.1.3.5. TIMER_resume



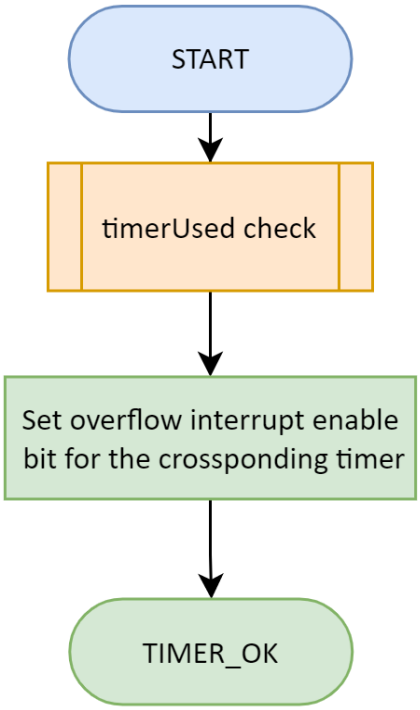
3.1.3.6. TIMER_pause



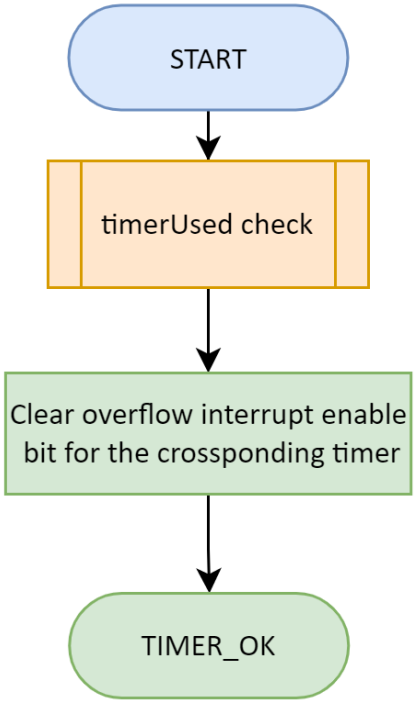
3.1.3.7. TIMER_reset



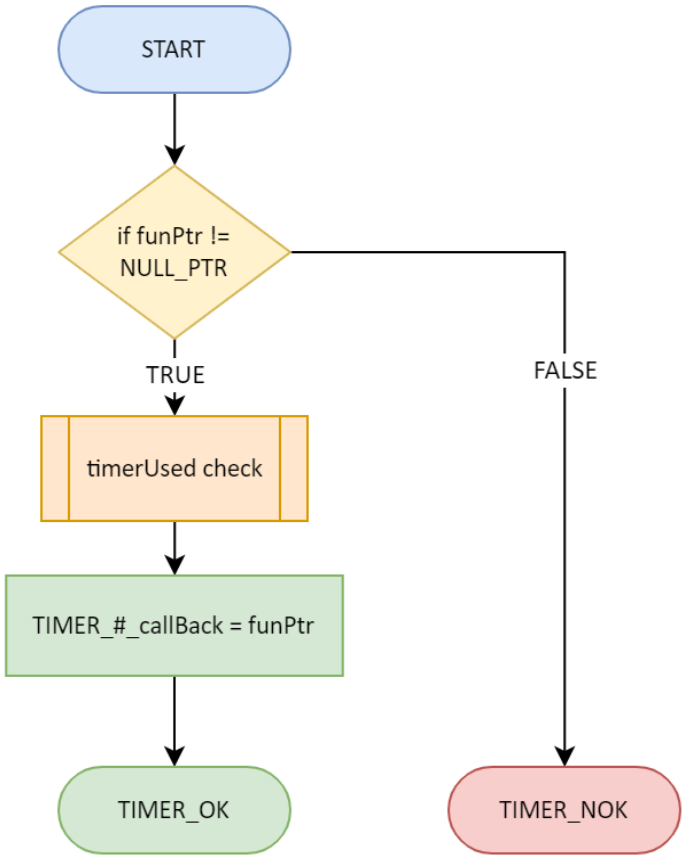
3.1.3.8. TIMER_enableInterrupt



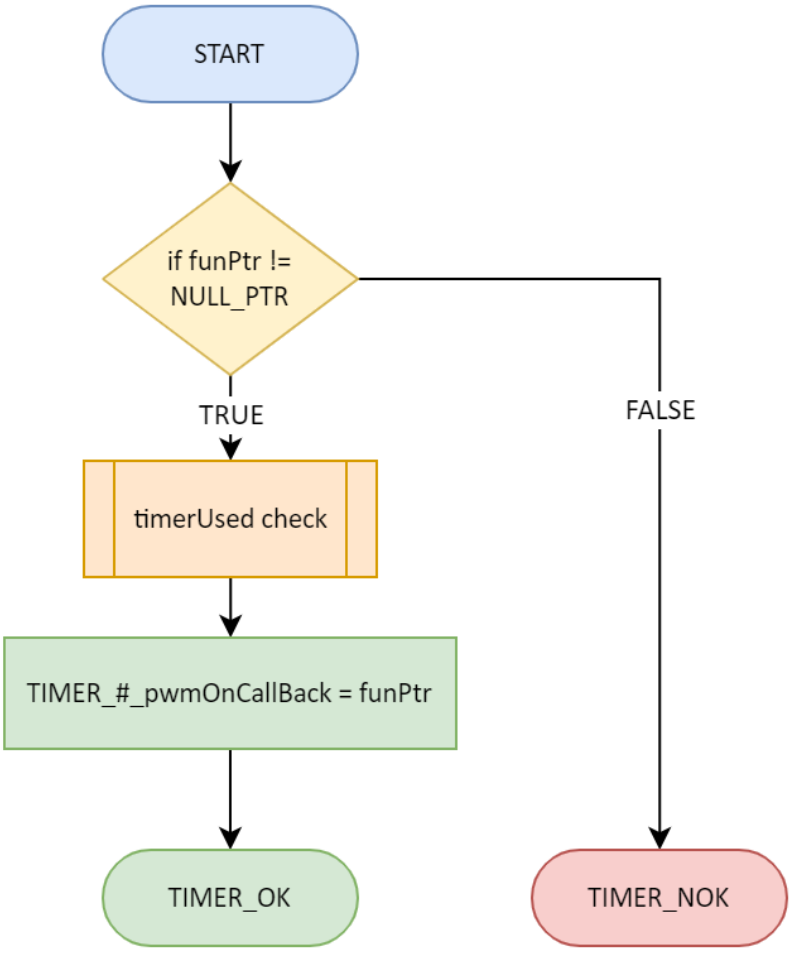
3.1.3.9. TIMER_enableInterrupt



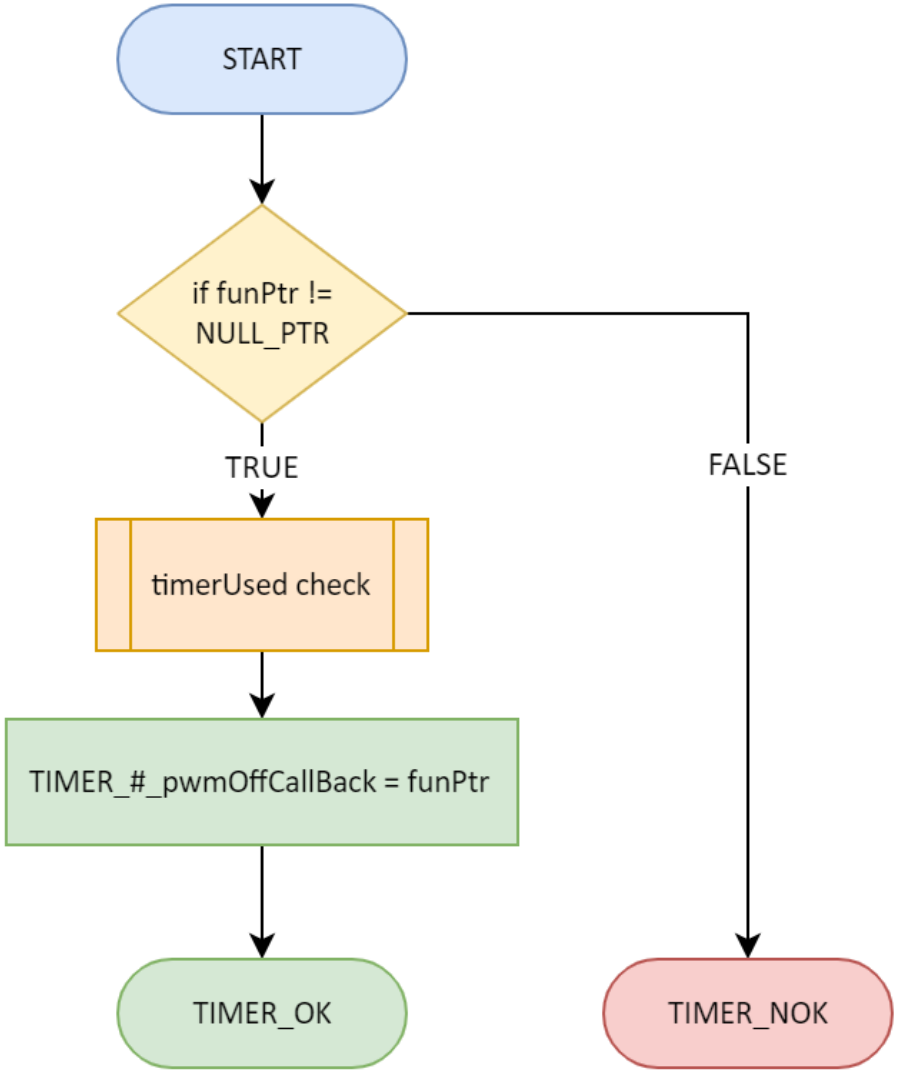
3.1.3.10. TIMER_setCallBack



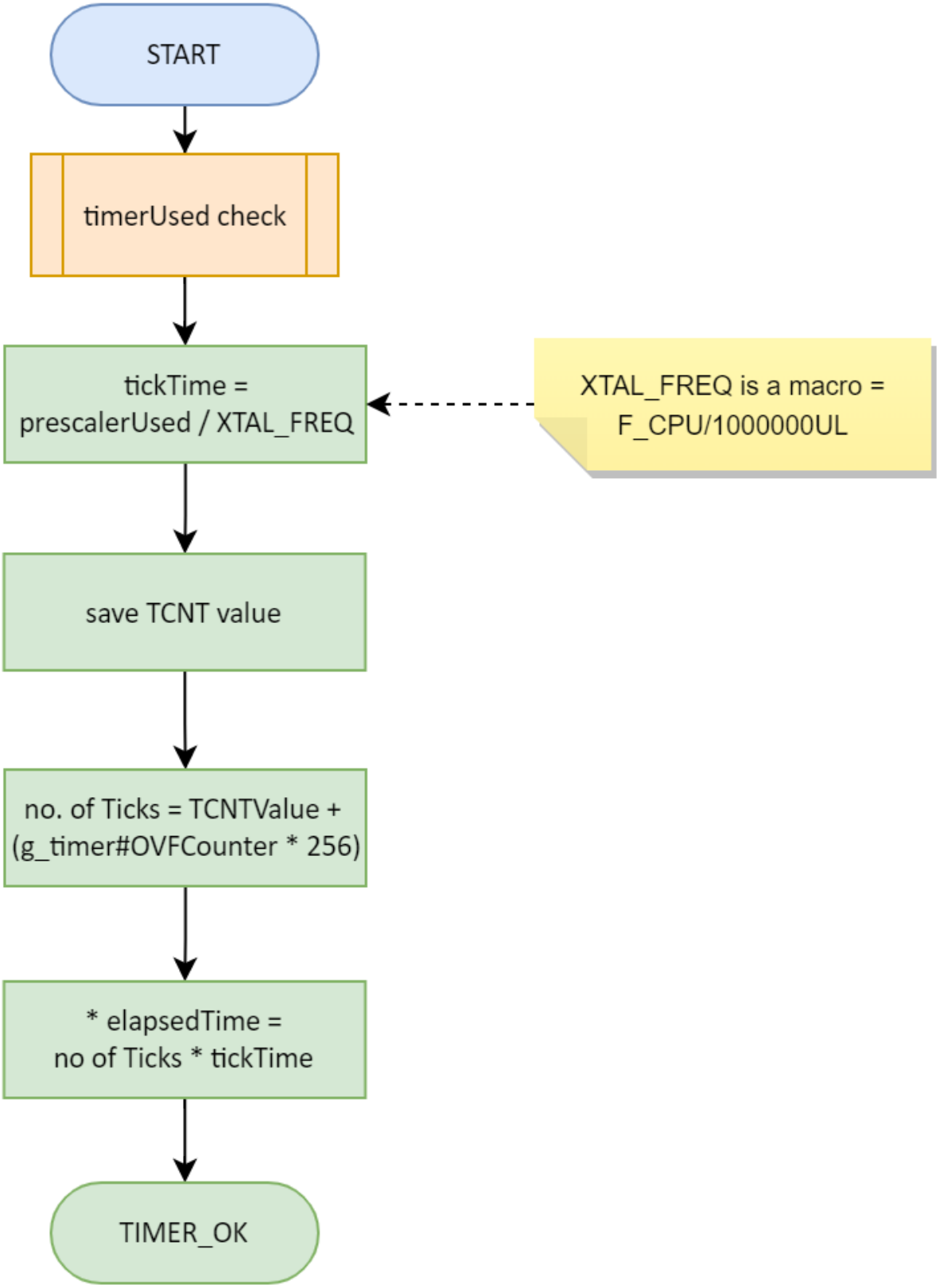
3.1.3.11. TIMER_setPwmOnCallBack



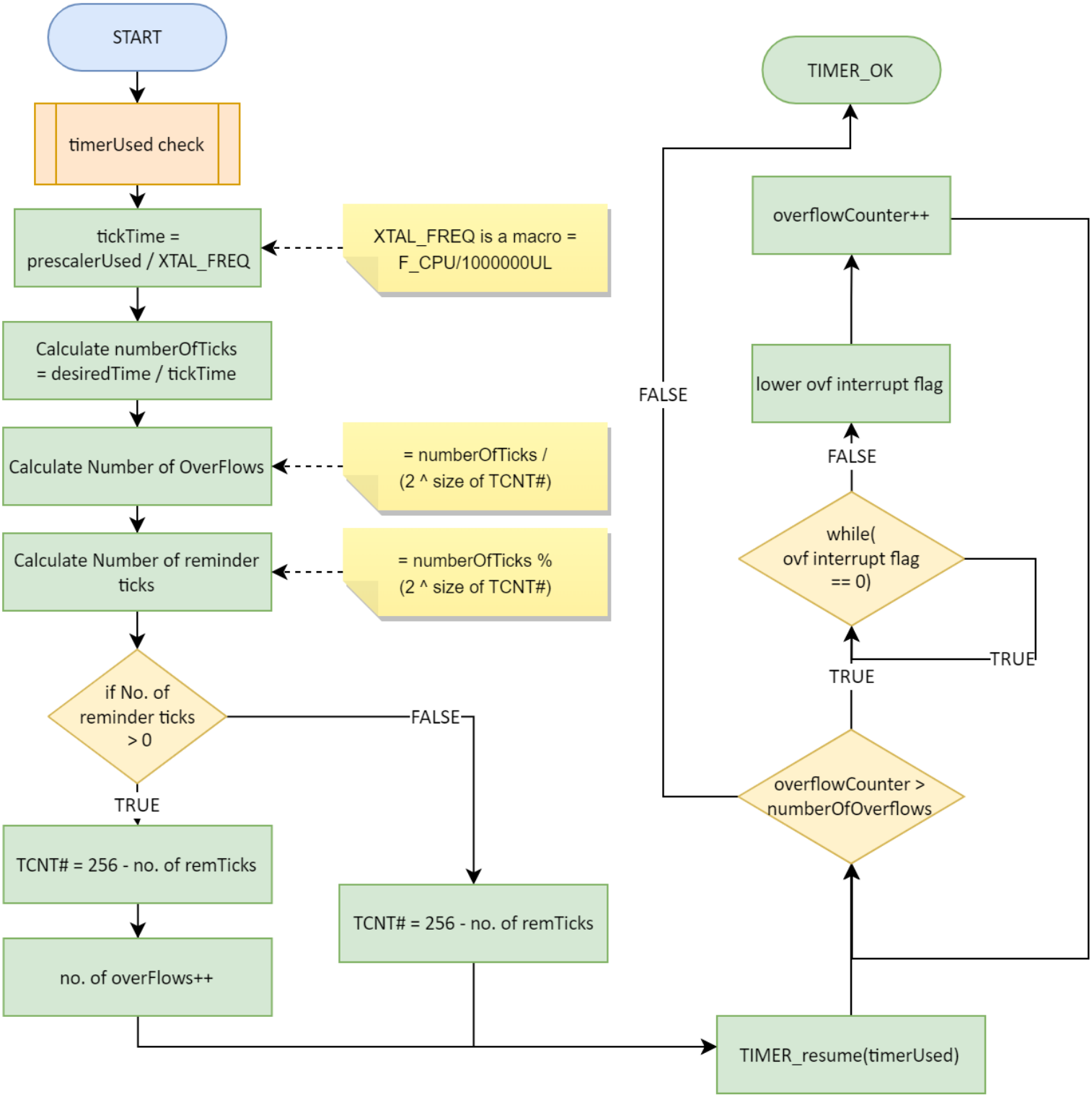
3.1.3.12. TIMER_setPwmOffCallBack



3.1.3.13. TIMER_getElapsedTime



3.1.3.14. TIMER_setDelayTime

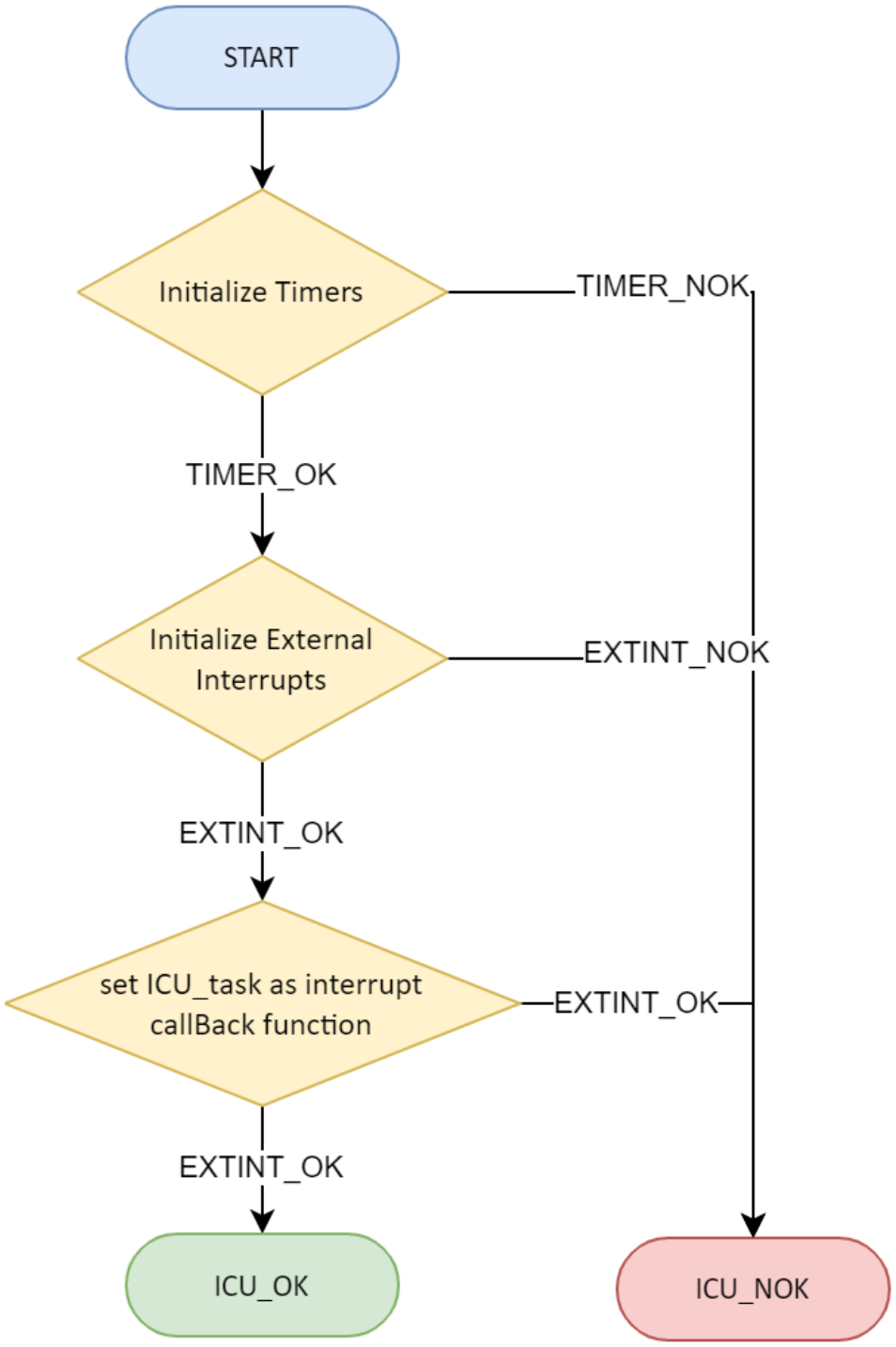


3.2. HAL Layer

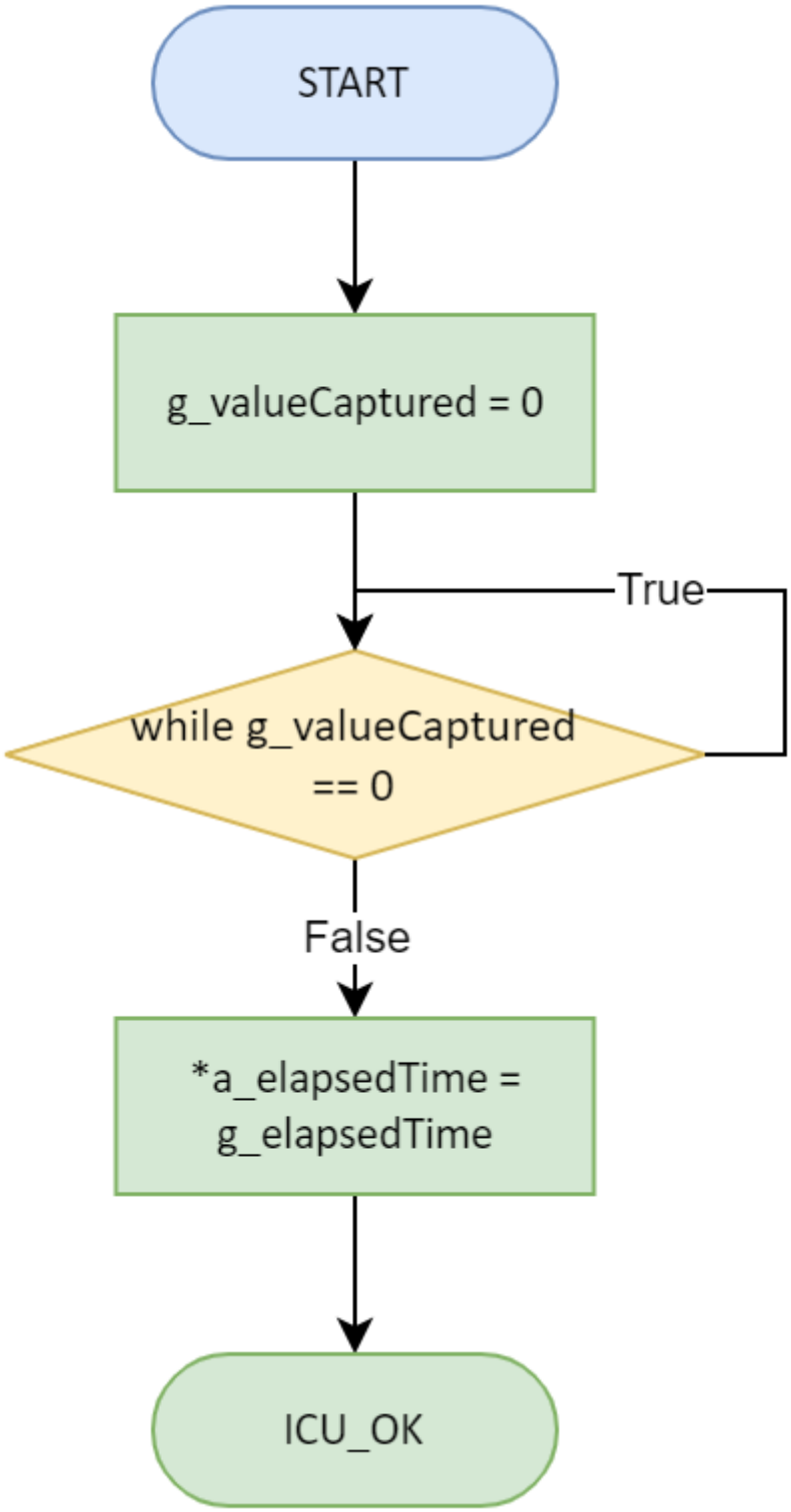
3.2.1. ICU Module (Input Capture Unit)

Any variable has g_ before it's naming is a volatile global variable

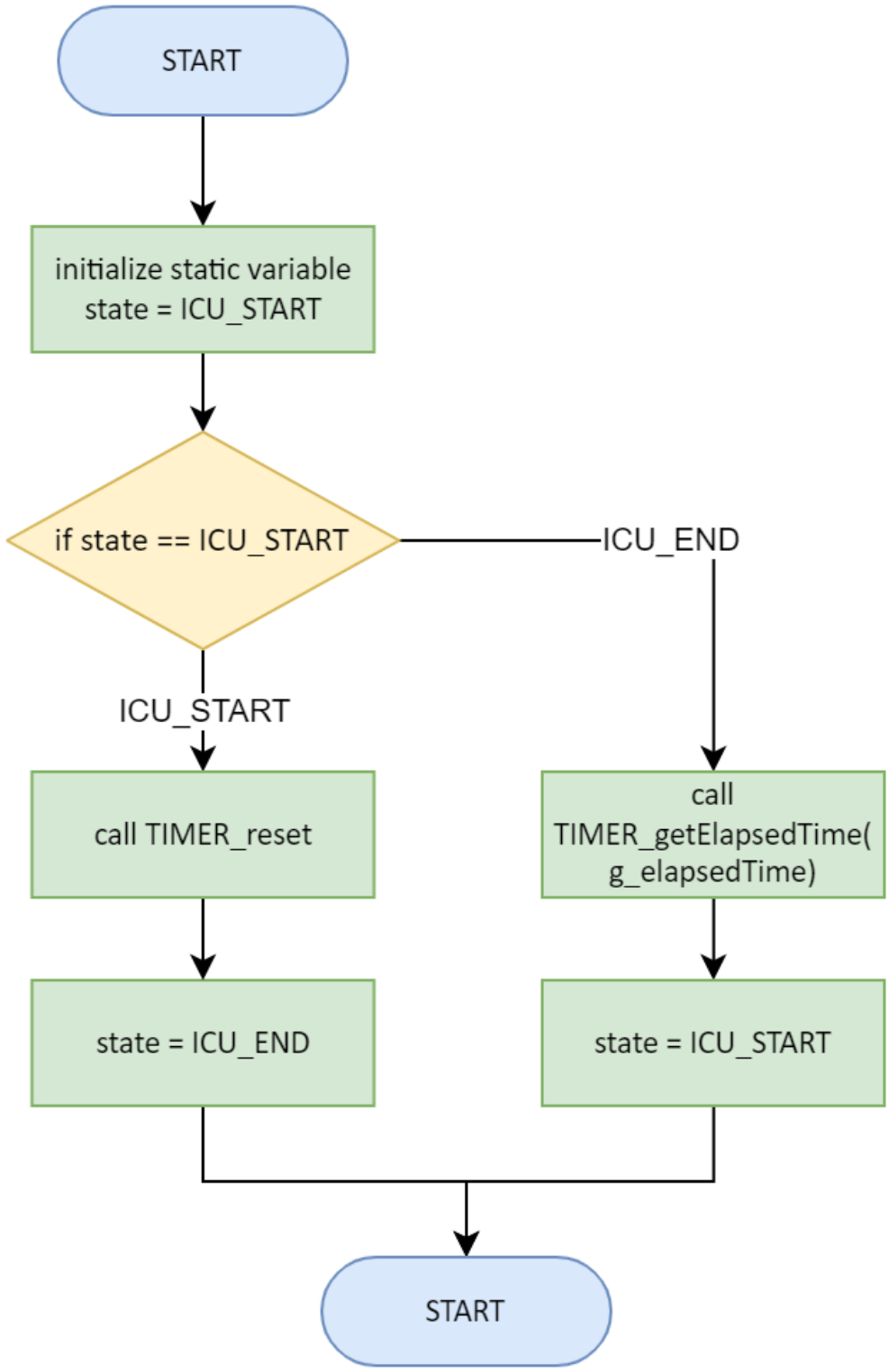
3.2.1.1. ICU_init



3.2.1.2. ICU_getElapsedTime



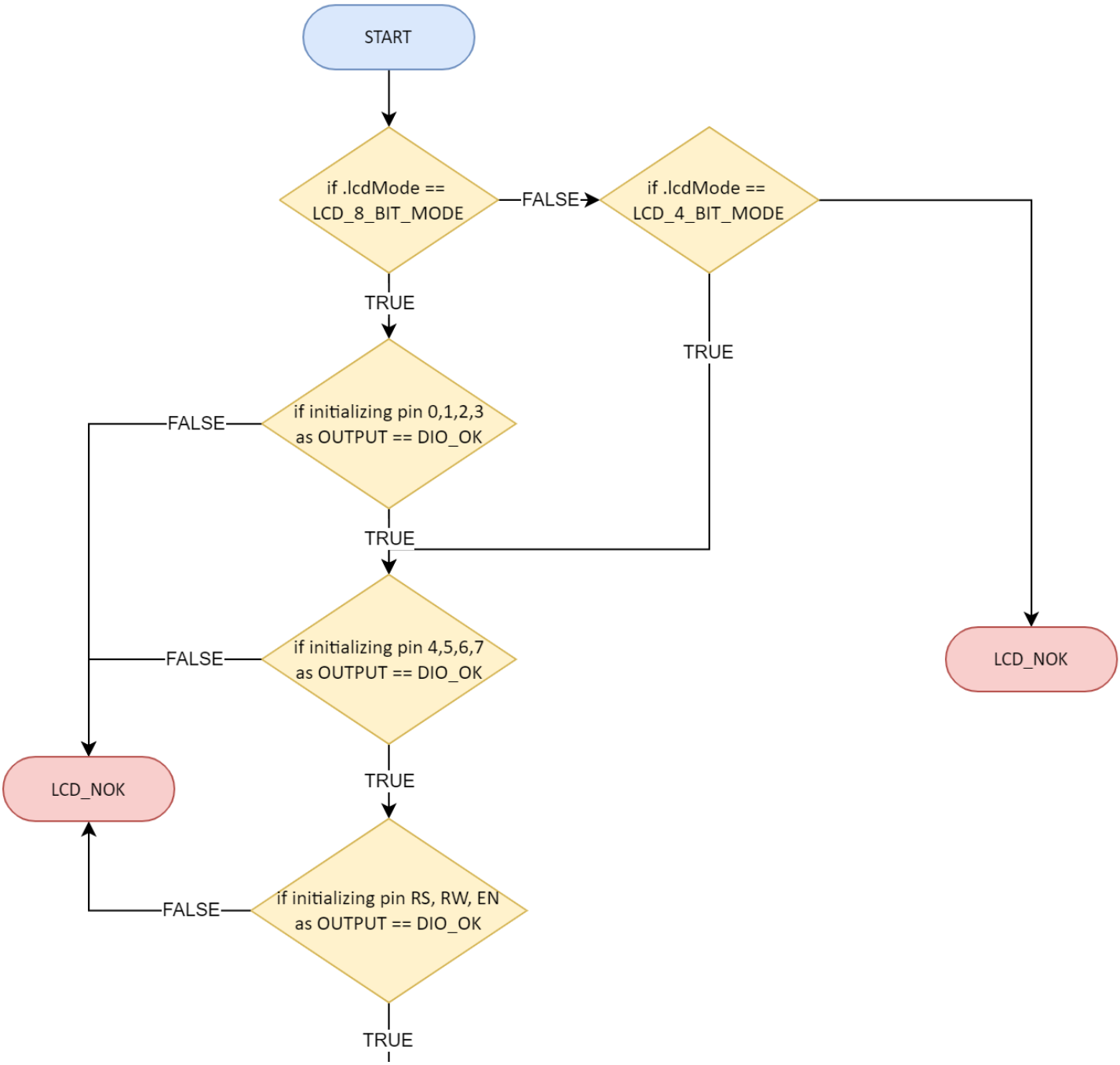
3.2.1.3. ICU_task



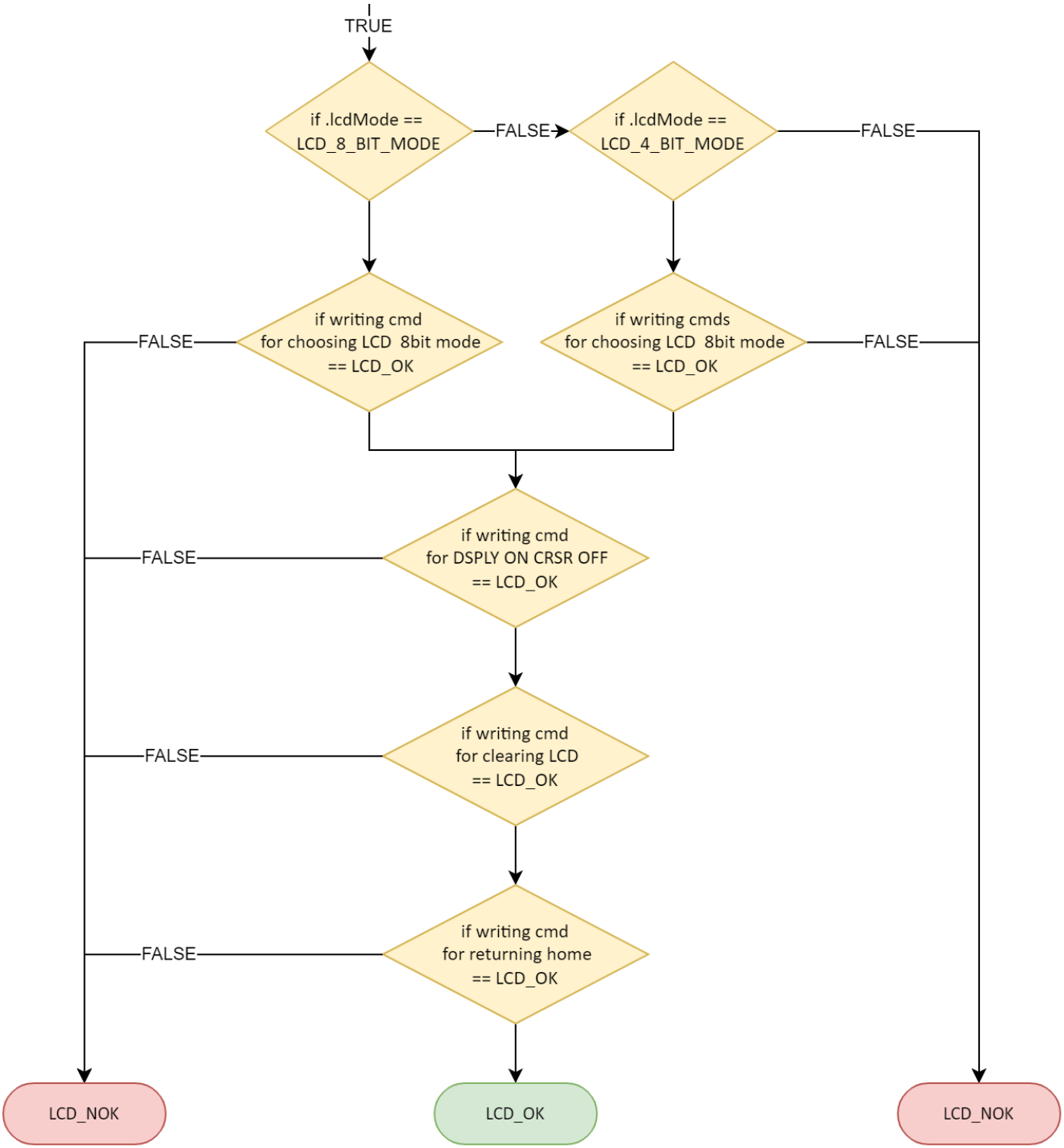
3.2.2. LCD Module

3.2.2.1. LCD_init

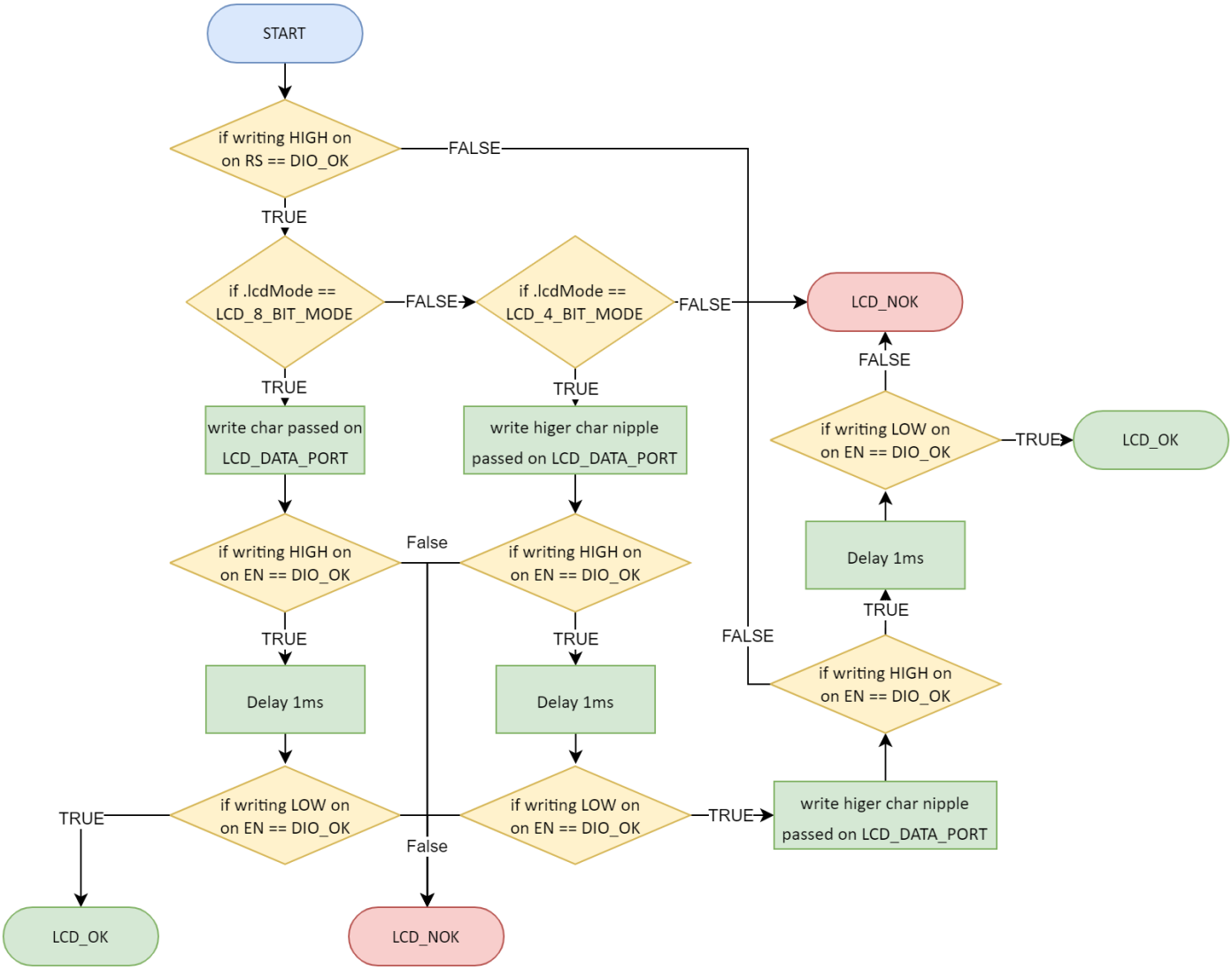
Part 1



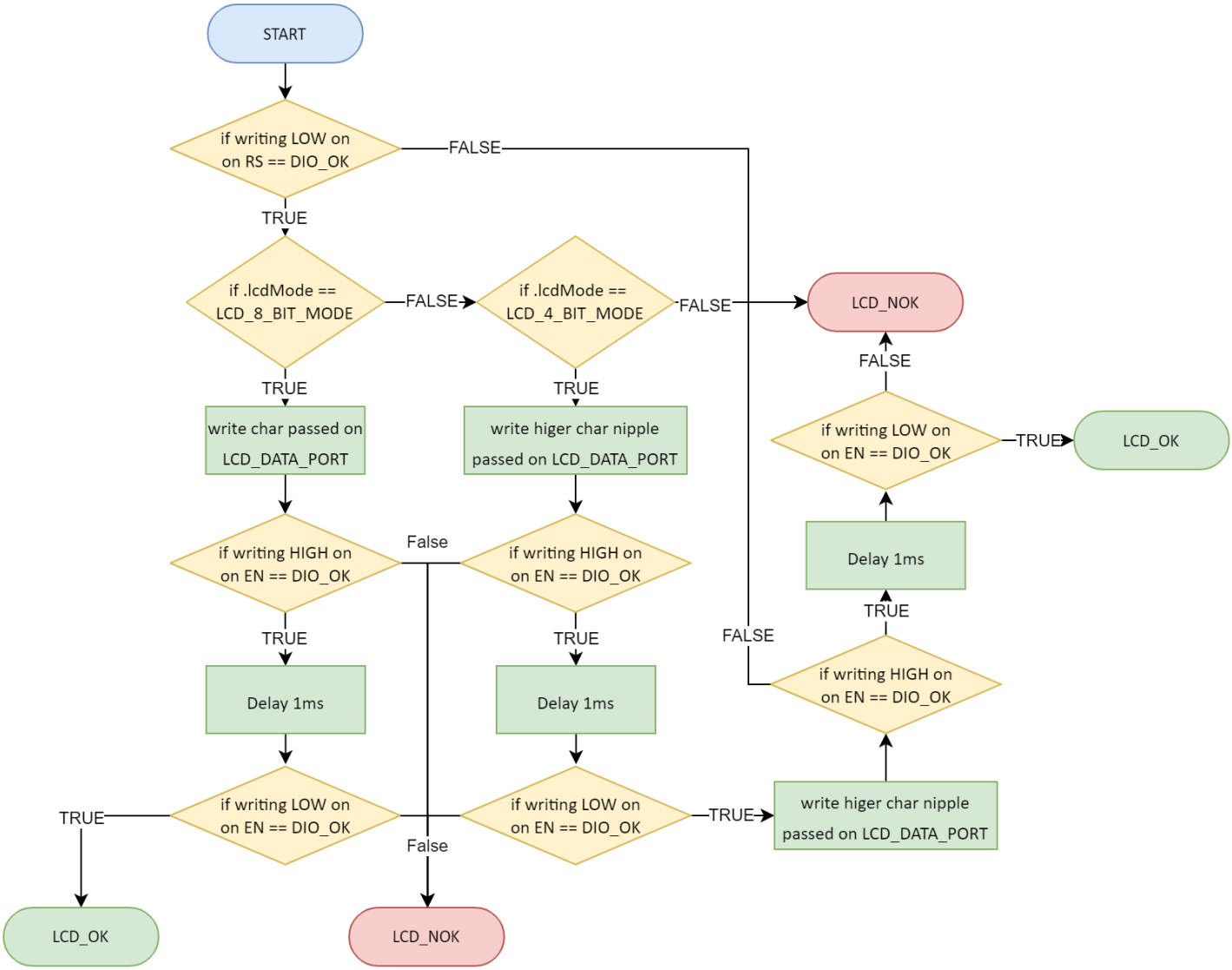
Part 2


TRUE
if .lcdMode == LCD_8_BIT_MODE
FALSE
if .lcdMode == LCD_4_BIT_MODE
FALSE
if writing cmd for choosing LCD 8bit mode == LCD_OK
FALSE
if writing cmds for choosing LCD 8bit mode == LCD_OK
FALSE
if writing cmd for DSPLY ON CRSR OFF == LCD_OK
FALSE
if writing cmd for clearing LCD == LCD_OK
FALSE
if writing cmd for returning home == LCD_OK
FALSE
LCD_NOK
LCD_OK
LCD_NOK

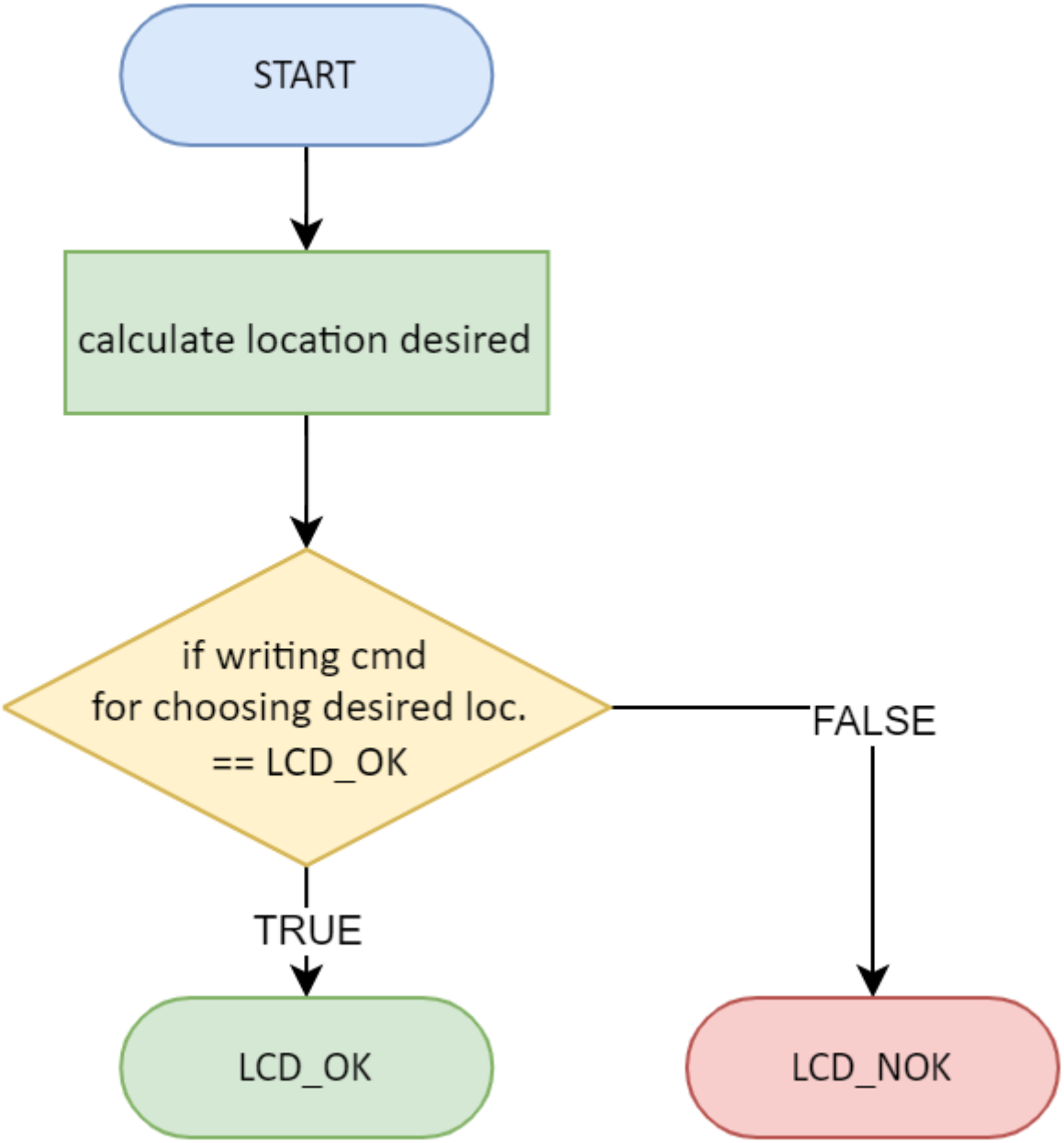
3.2.2.2. LCD_writeChar



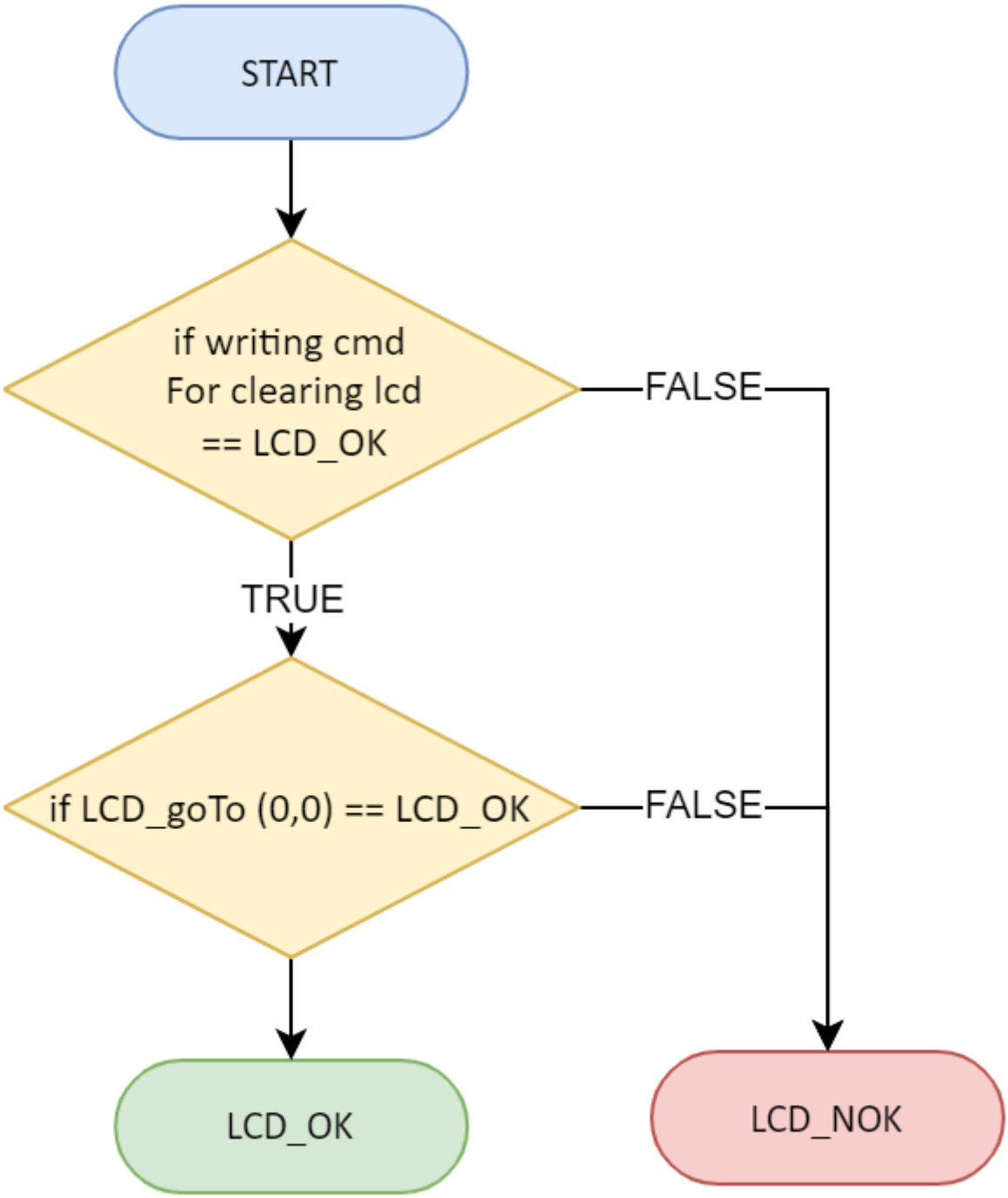
3.2.2.3. LCD_writeCmd



3.2.2.4. LCD_clr

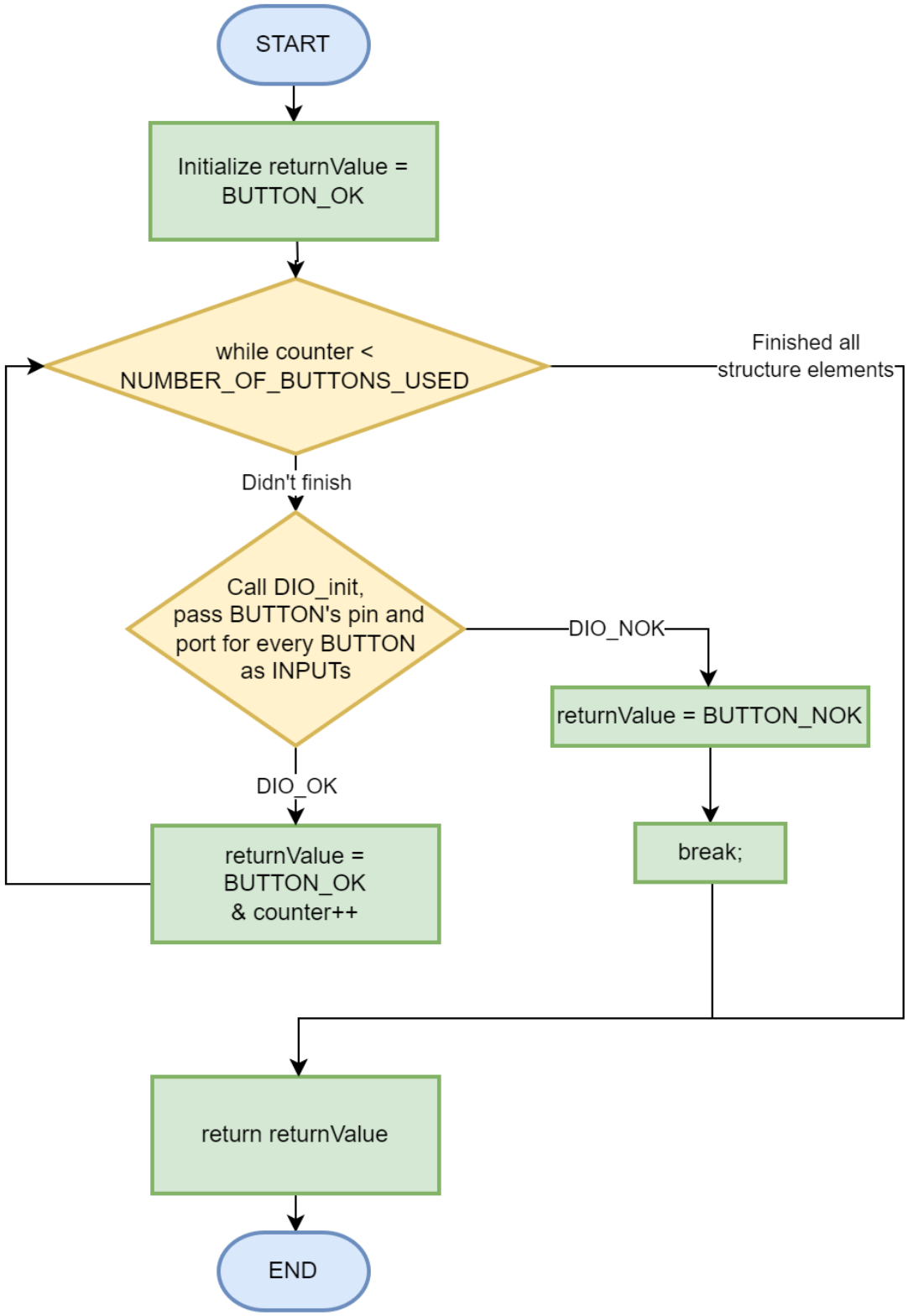

START
calculate location desired
if writing cmd
for choosing desired loc.
== LCD_OK
FALSE
TRUE
LCD_OK
LCD_NOK

3.2.2.5. LCD_goTo

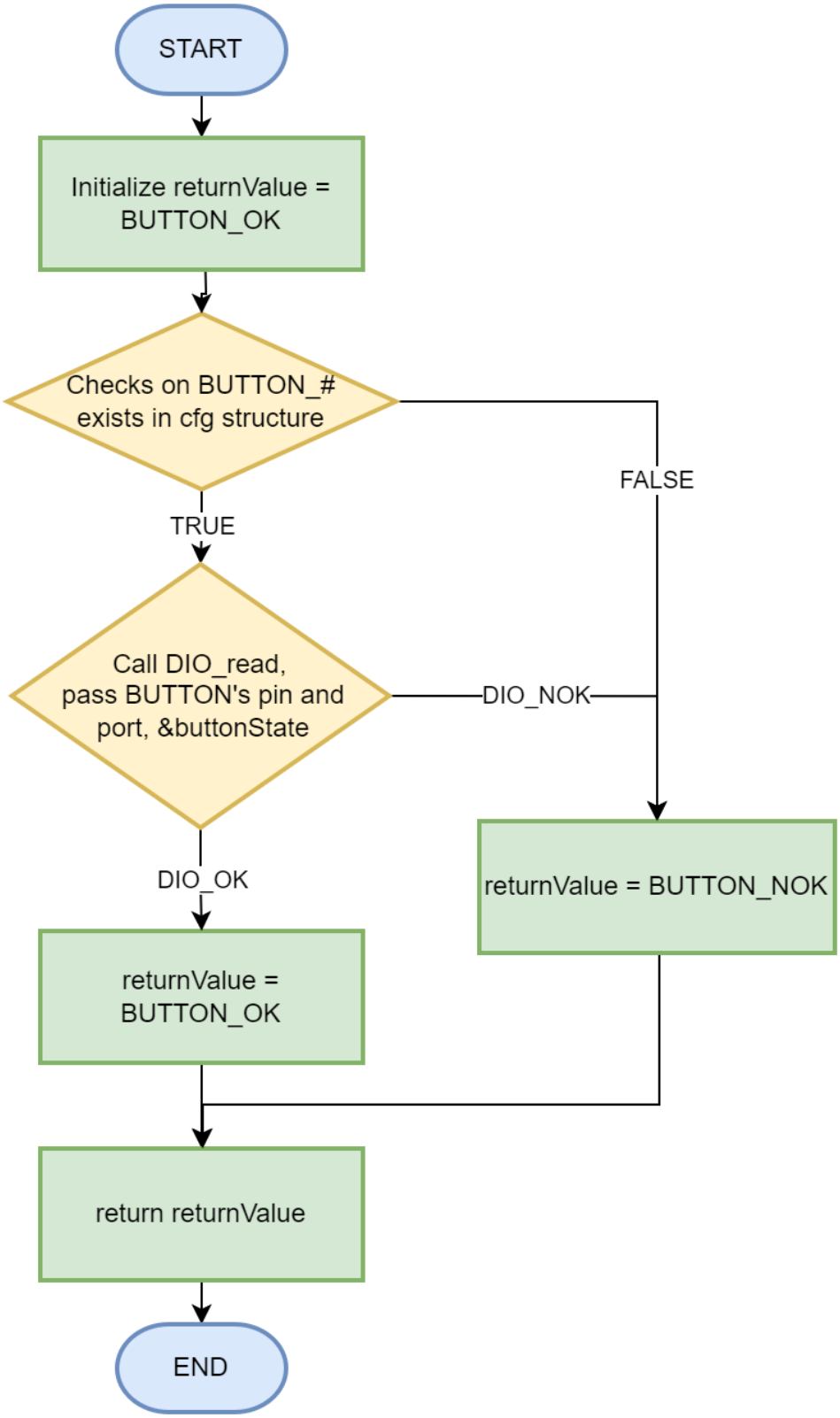


3.2.3. BTN Module

3.2.3.1. BUTTON_init

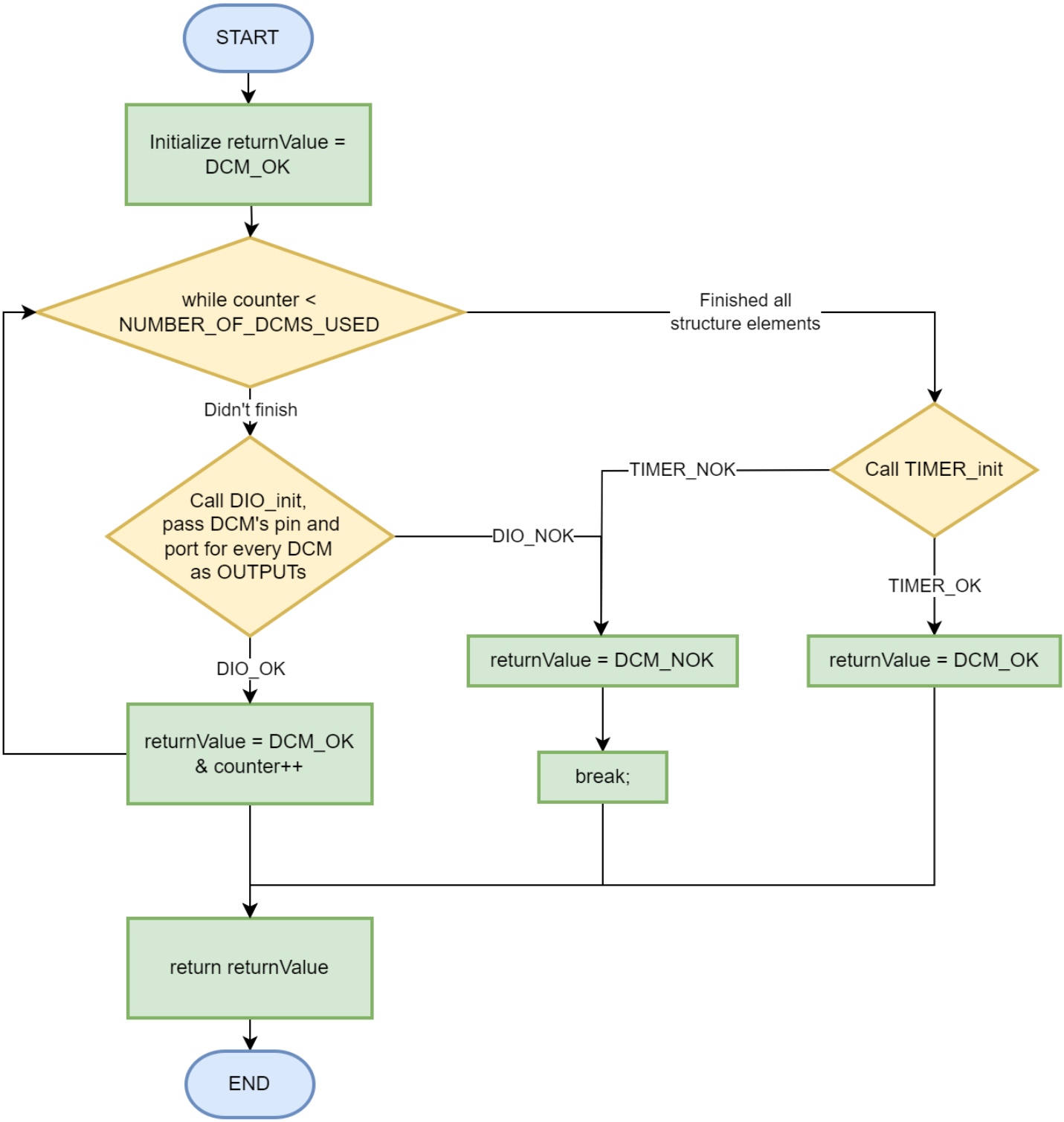


3.2.3.2. BUTTON_read

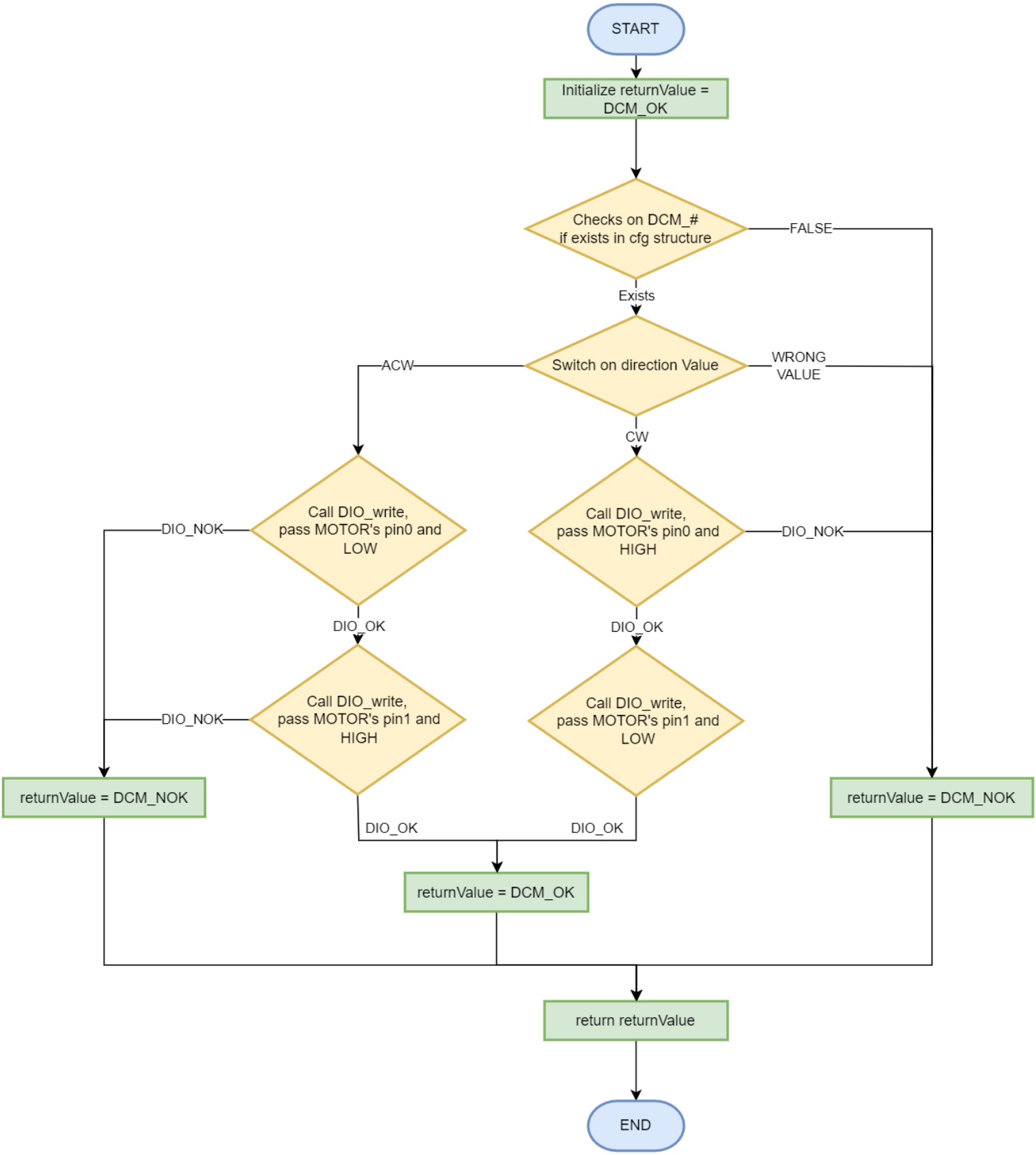


3.2.4. DCM Module

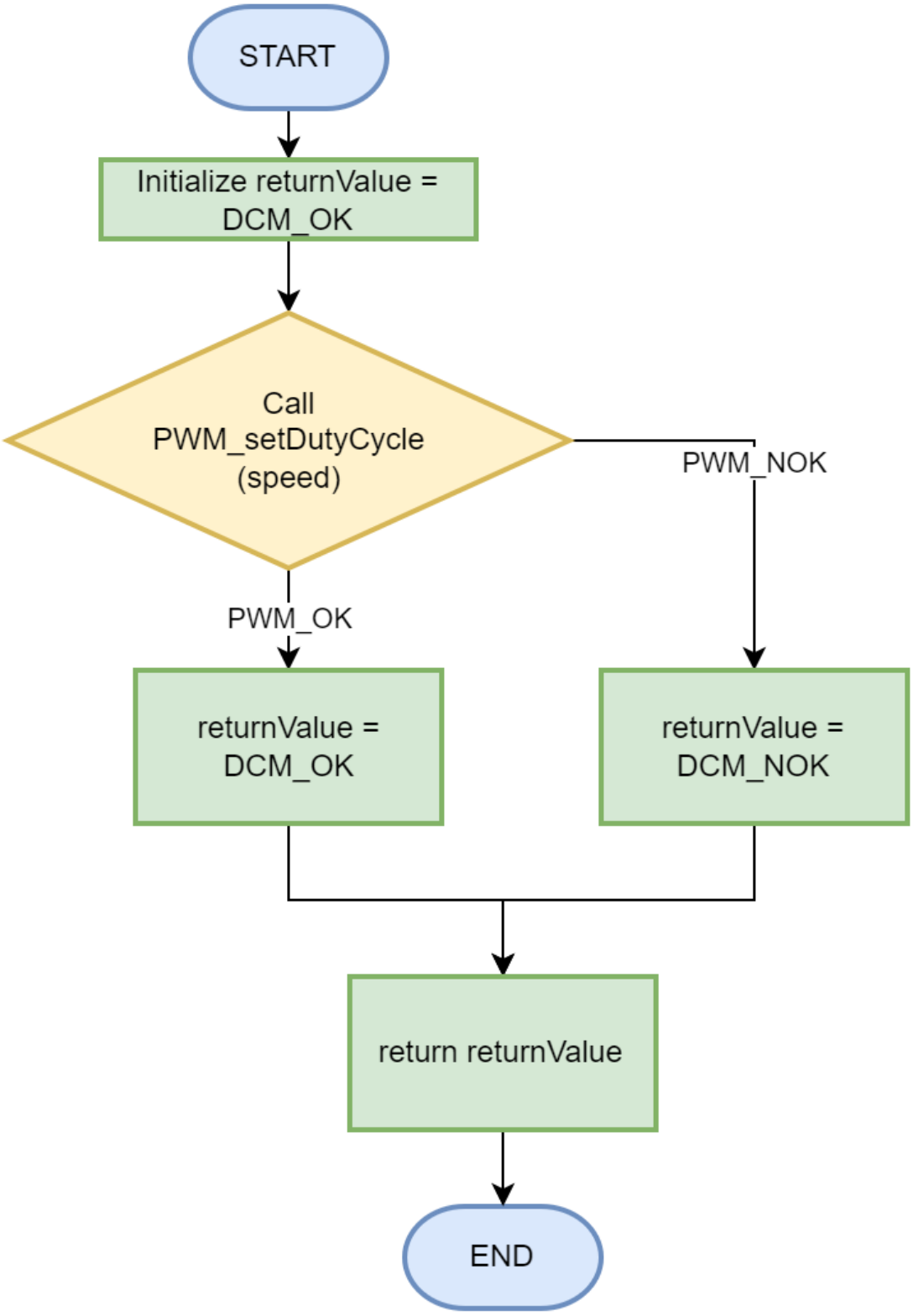
3.2.4.1. DCM_init



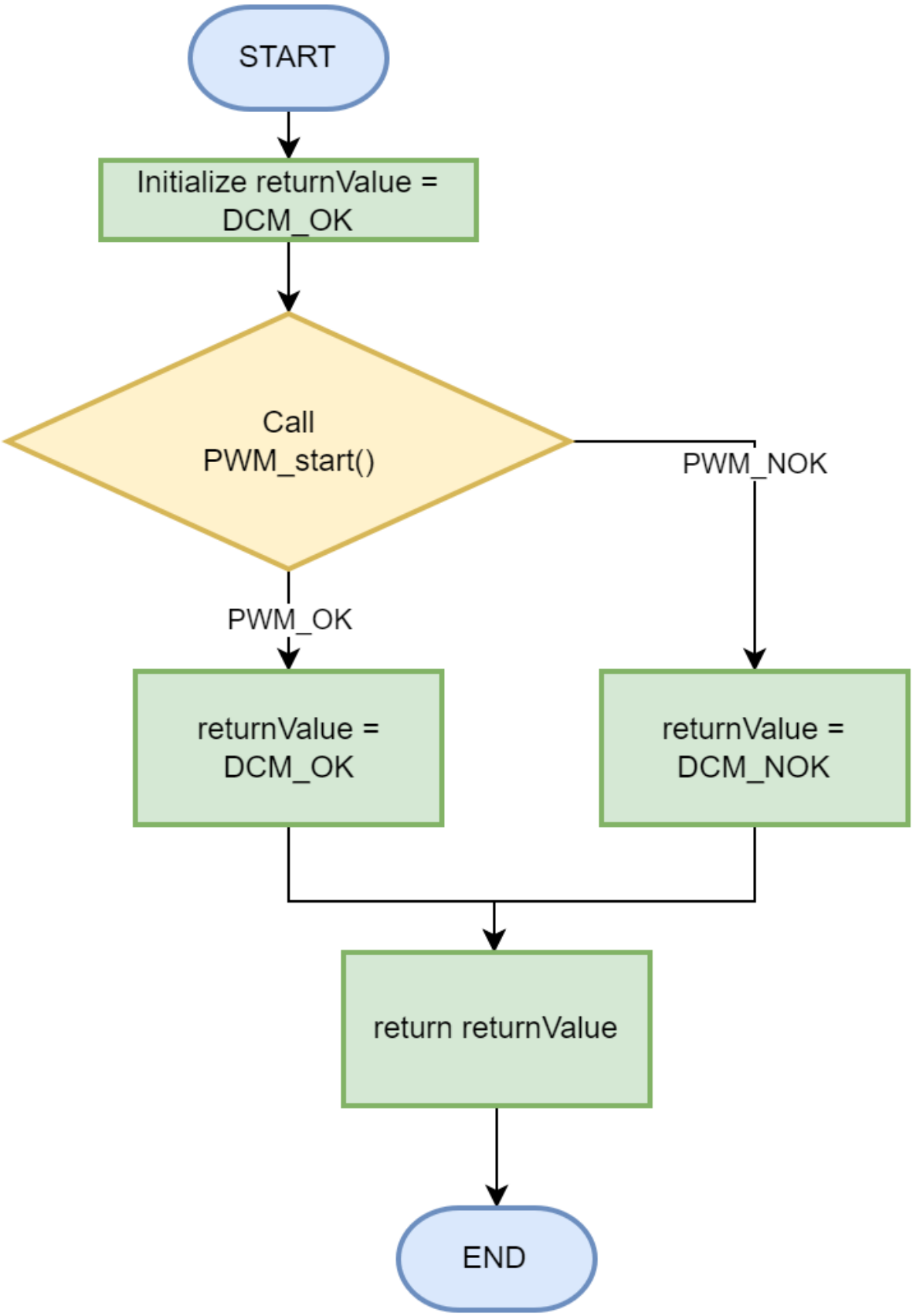
3.2.4.2. DCM_setDirection



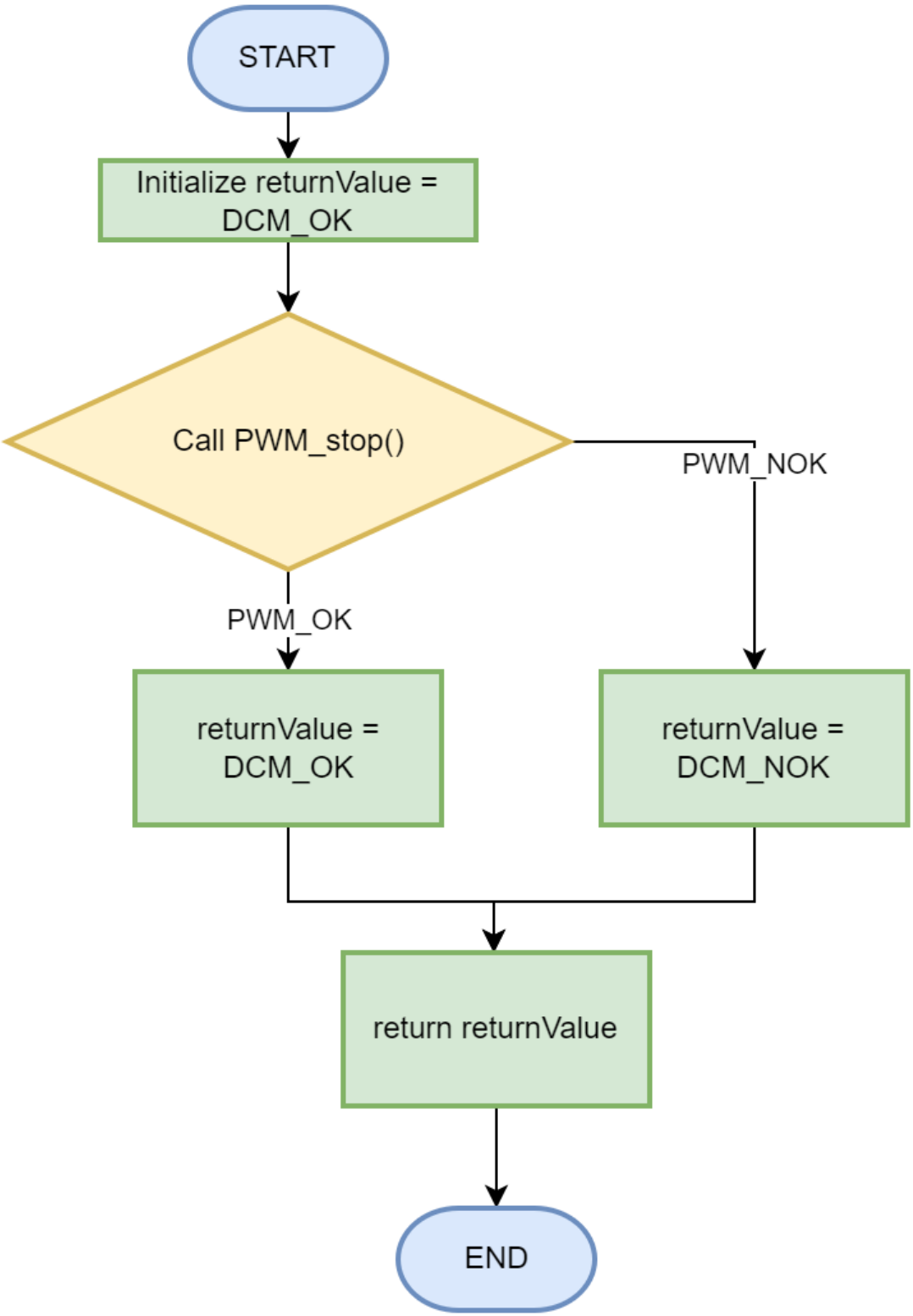
3.2.4.3. DCM_speed



3.2.4.4. DCM_start

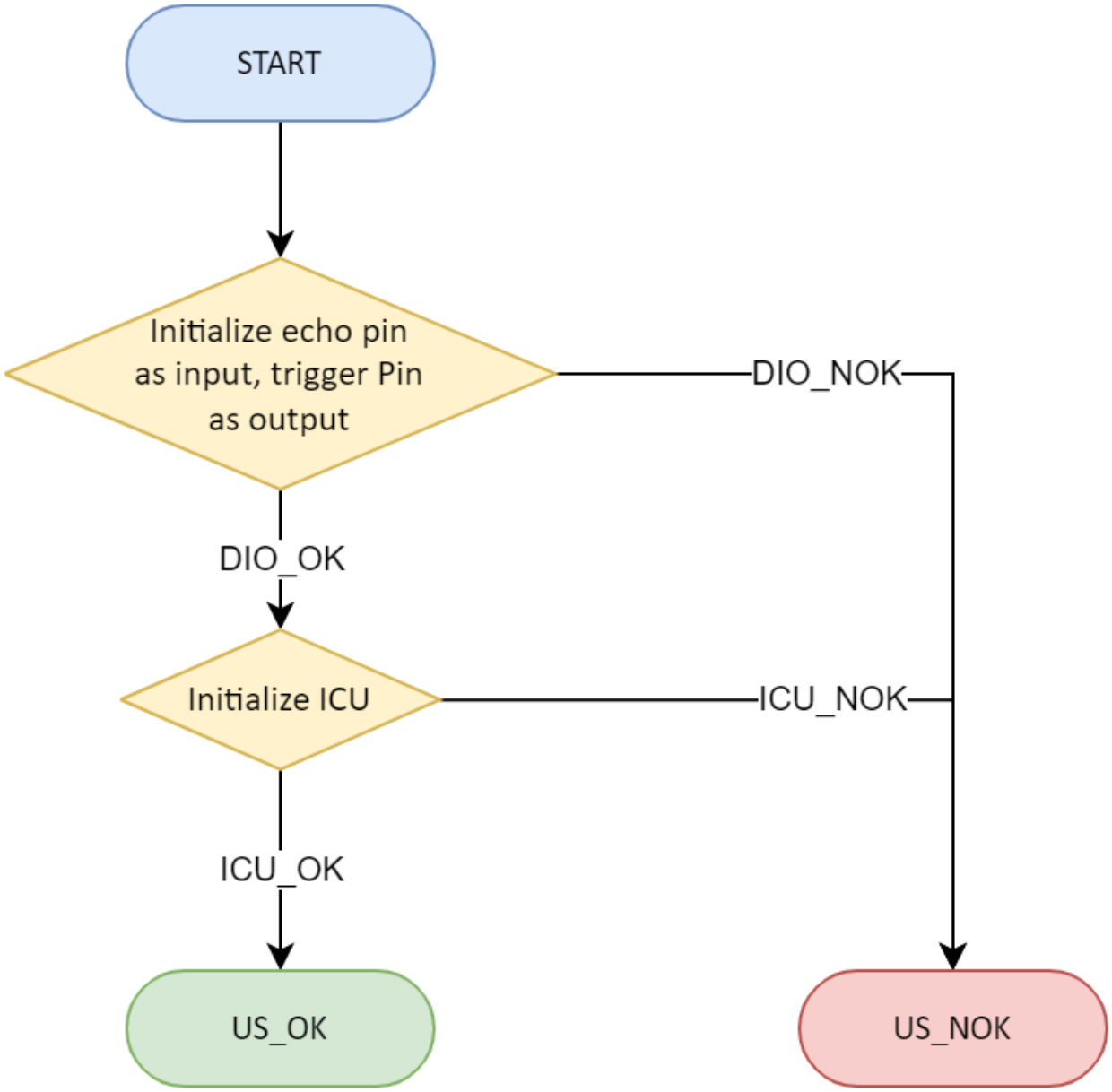


3.2.4.5. DCM_stop

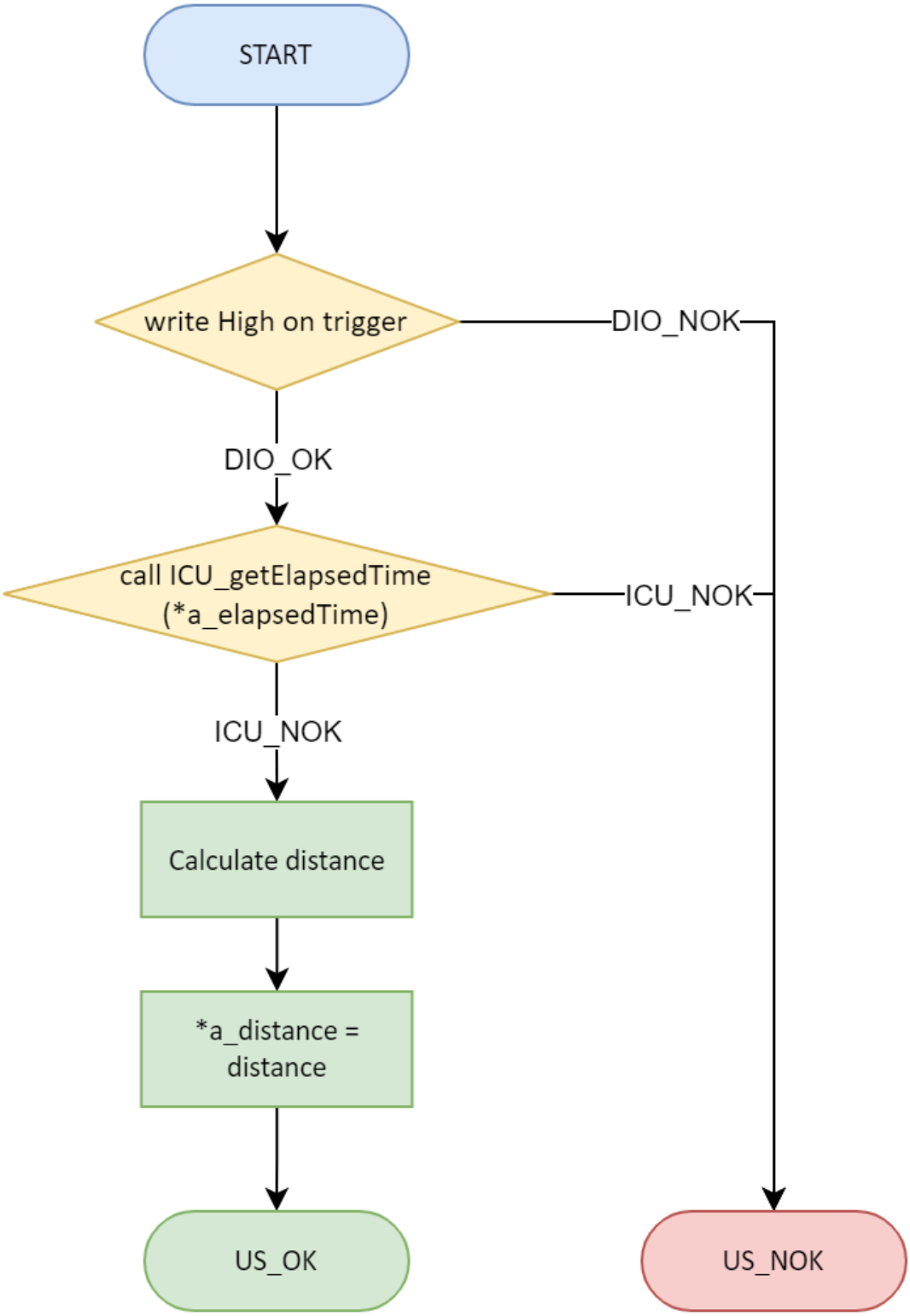


3.2.5. Ultrasonic Module

3.2.5.1. US_init

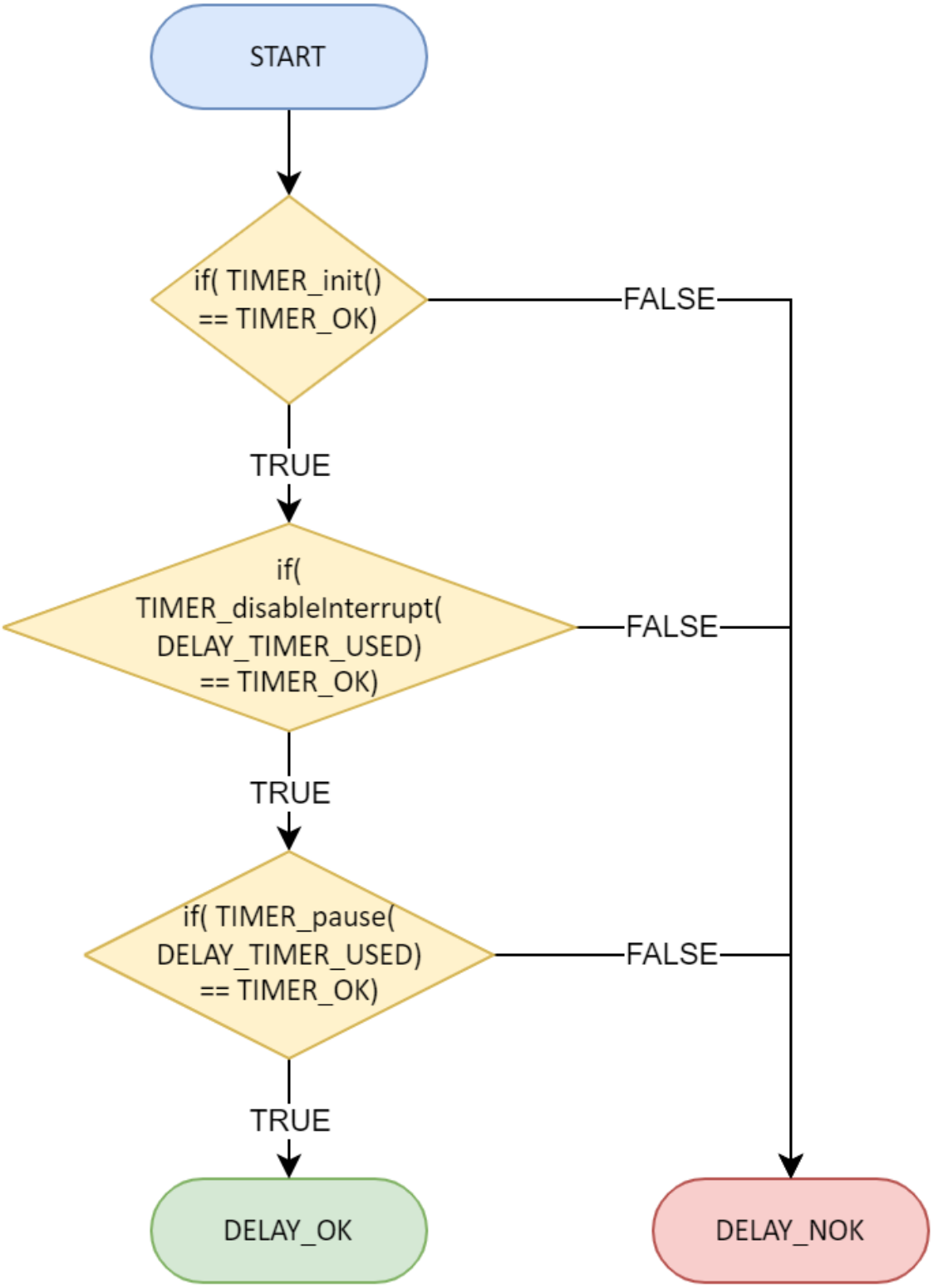


3.2.5.2. US_getDistance

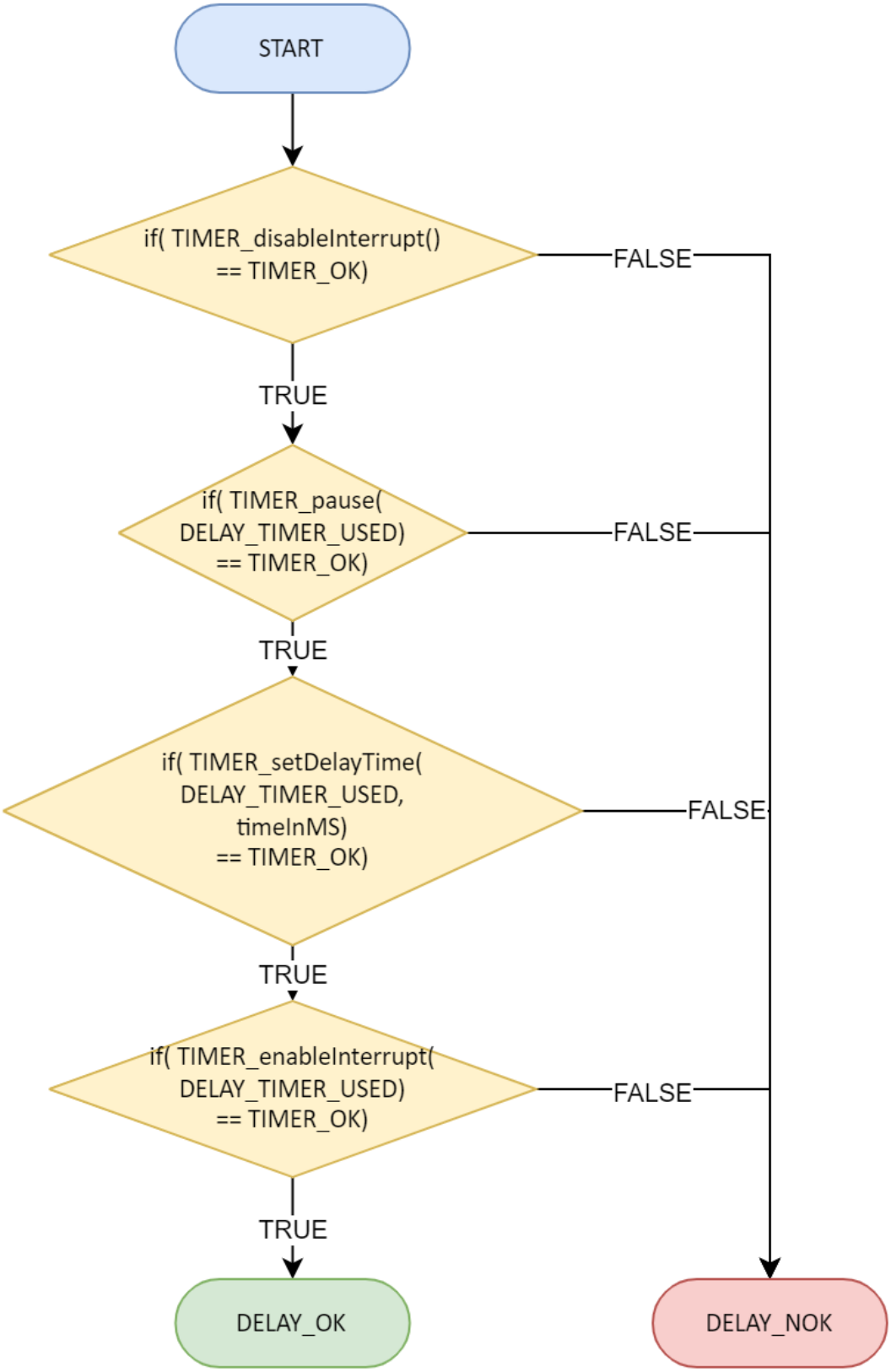


3.2.6. Delay Module

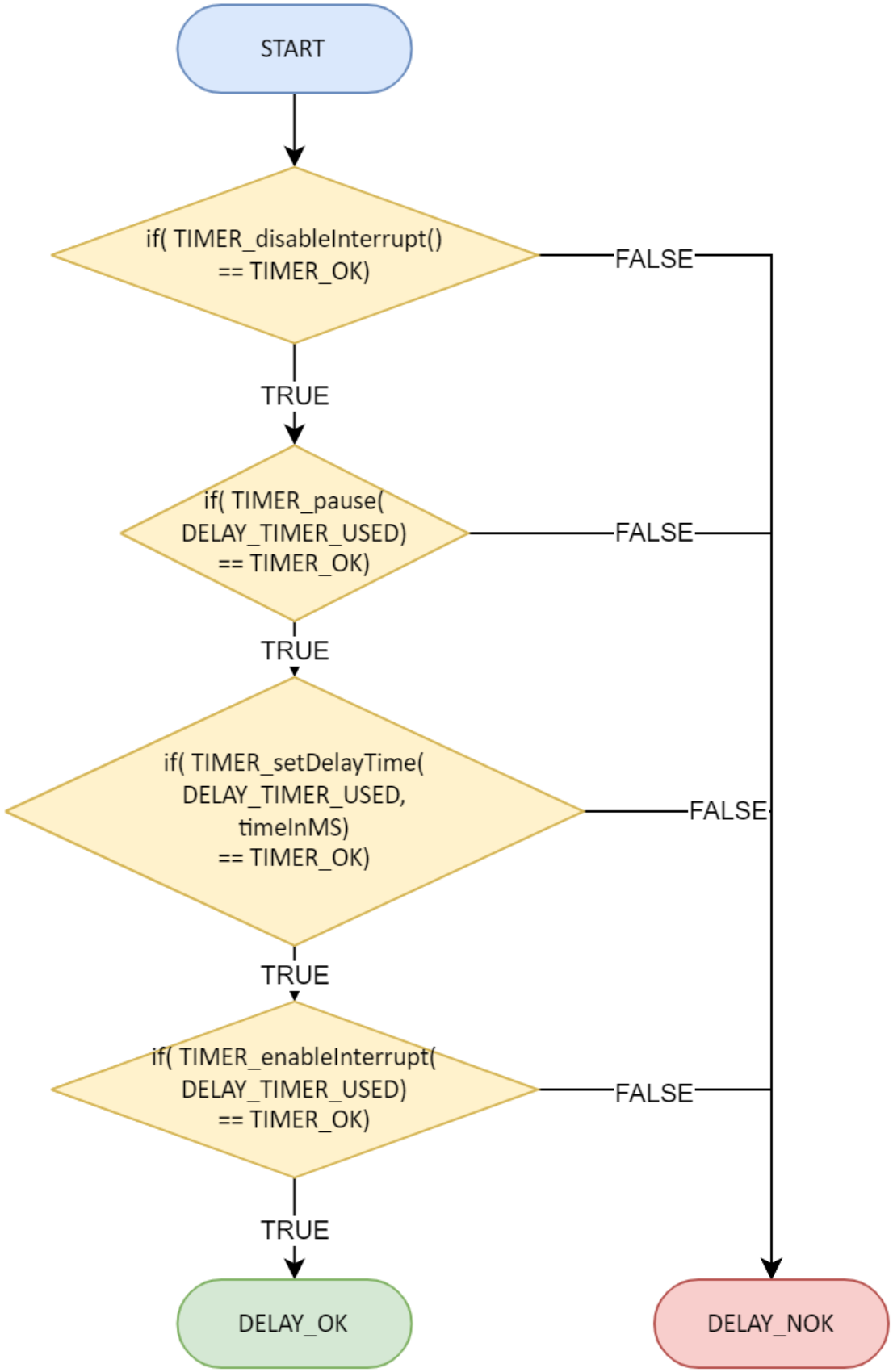
3.2.6.1. DELAY_init



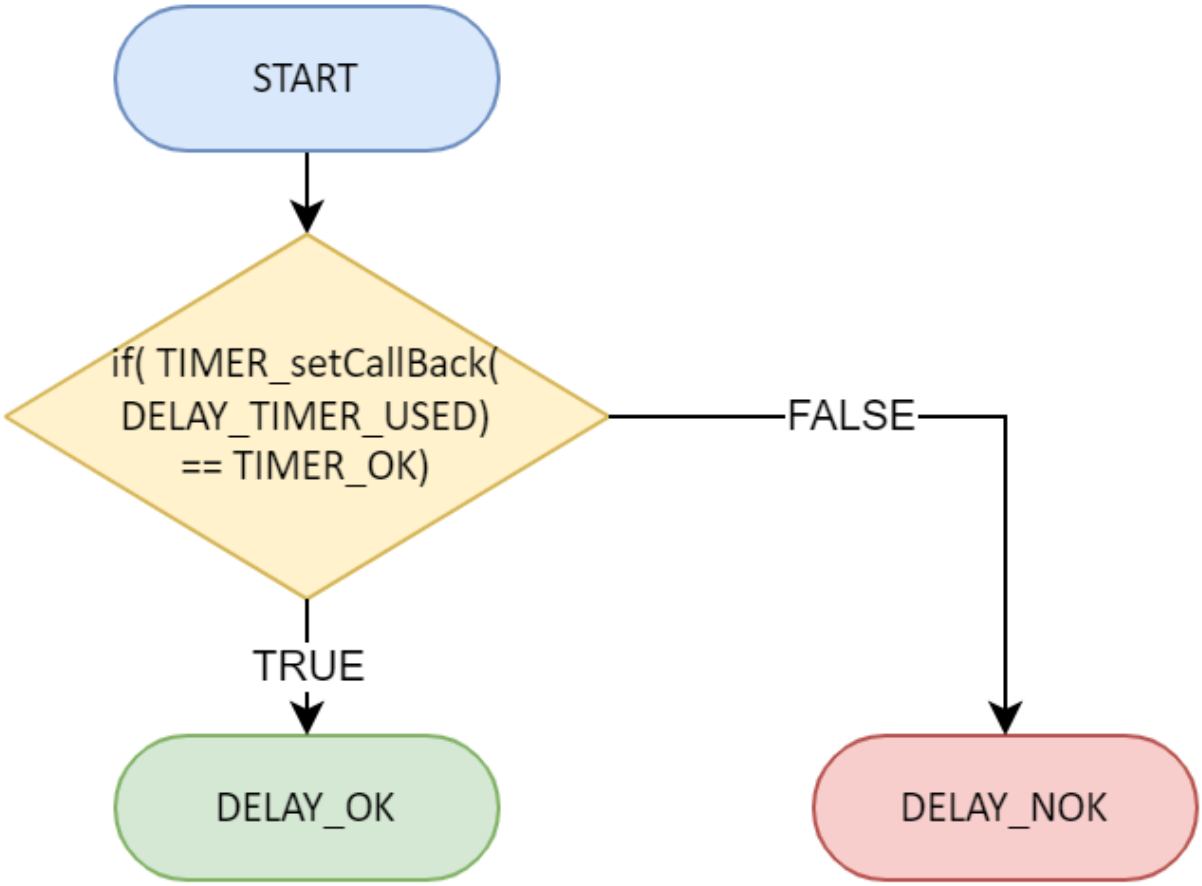
3.2.6.2. DELAY_setTime



3.2.6.3. DELAY_setTimeBlocking

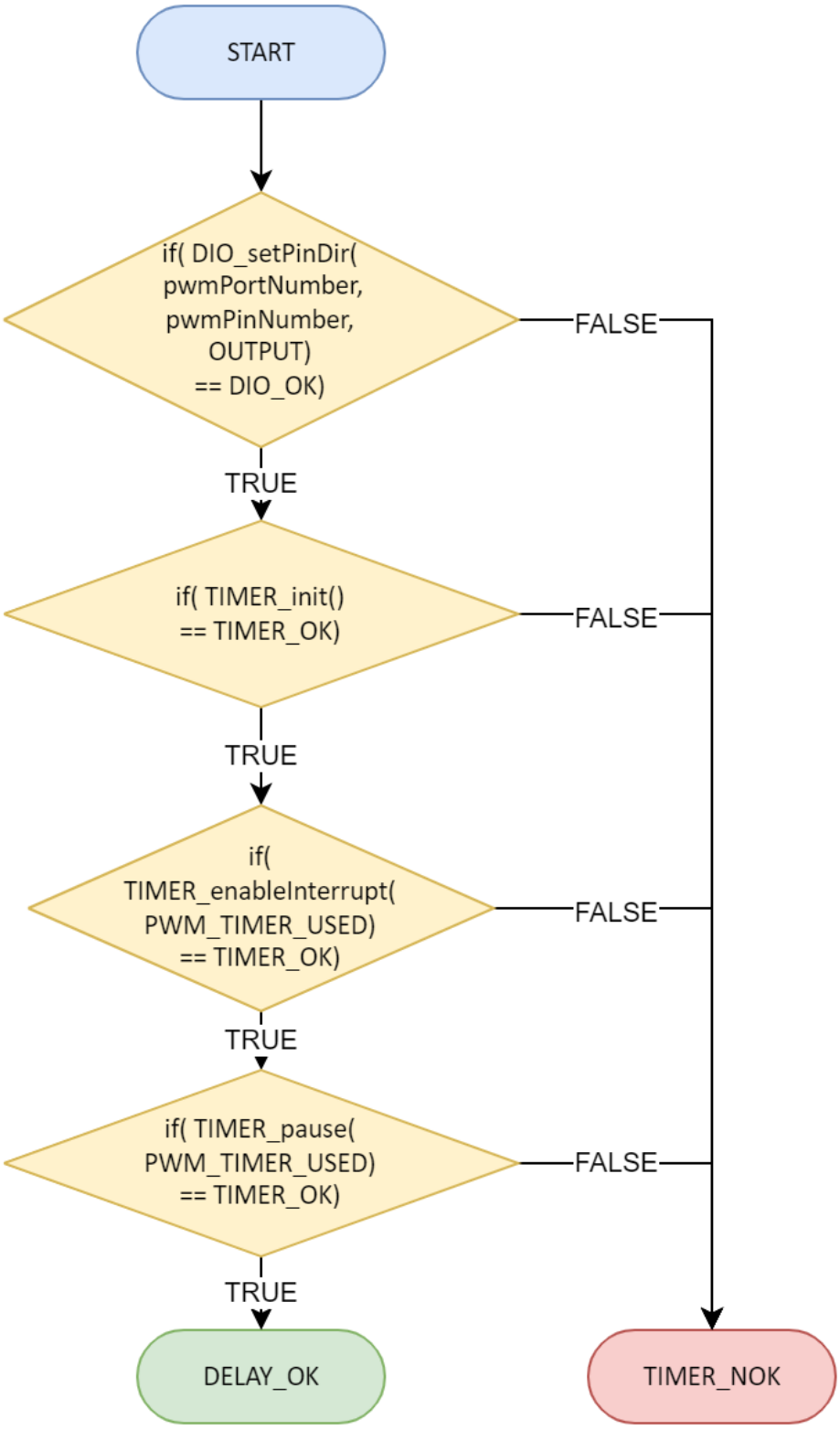


3.2.6.4. DELAY_setTimeBlocking

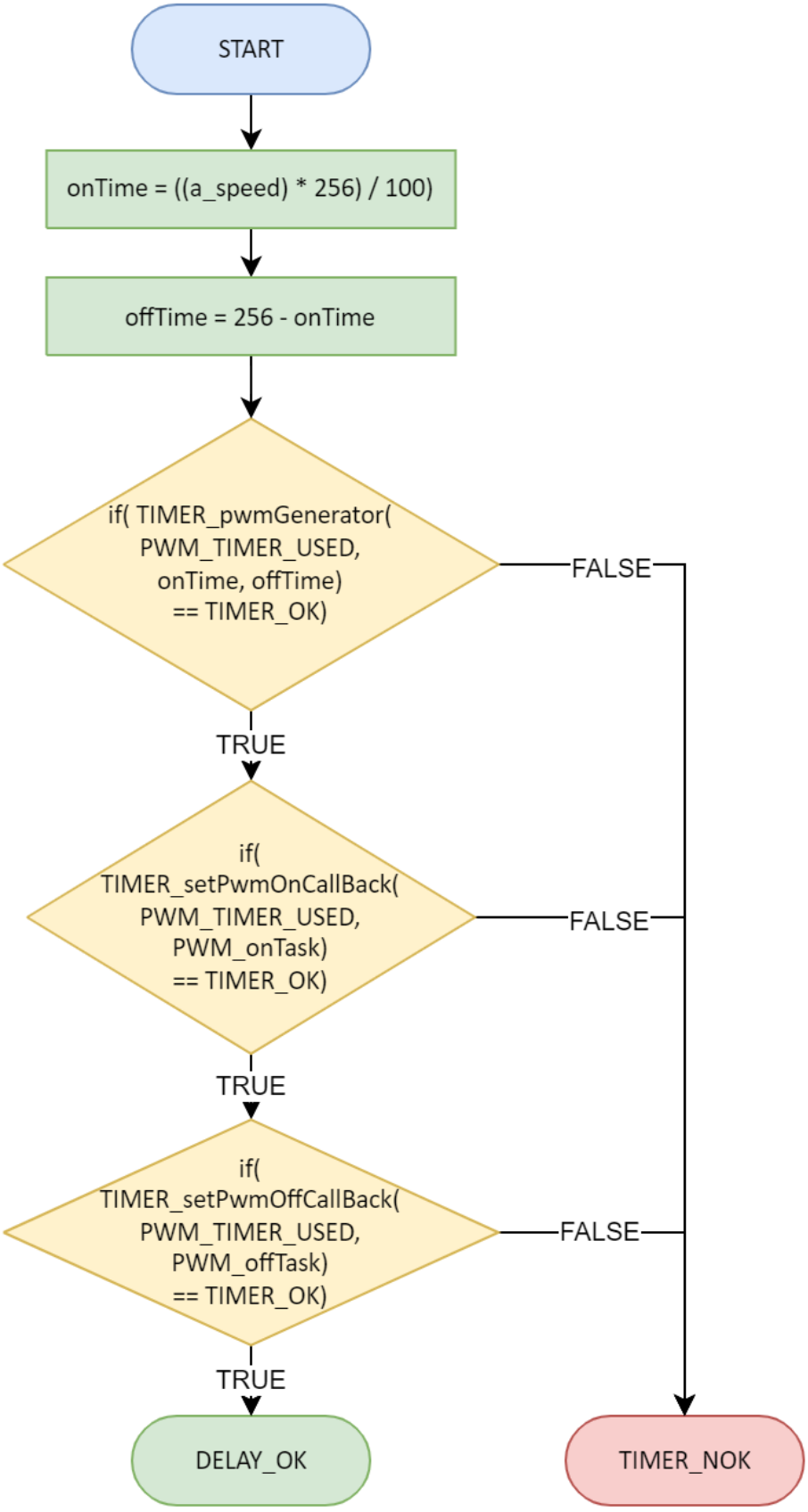


3.2.7. PWM Module

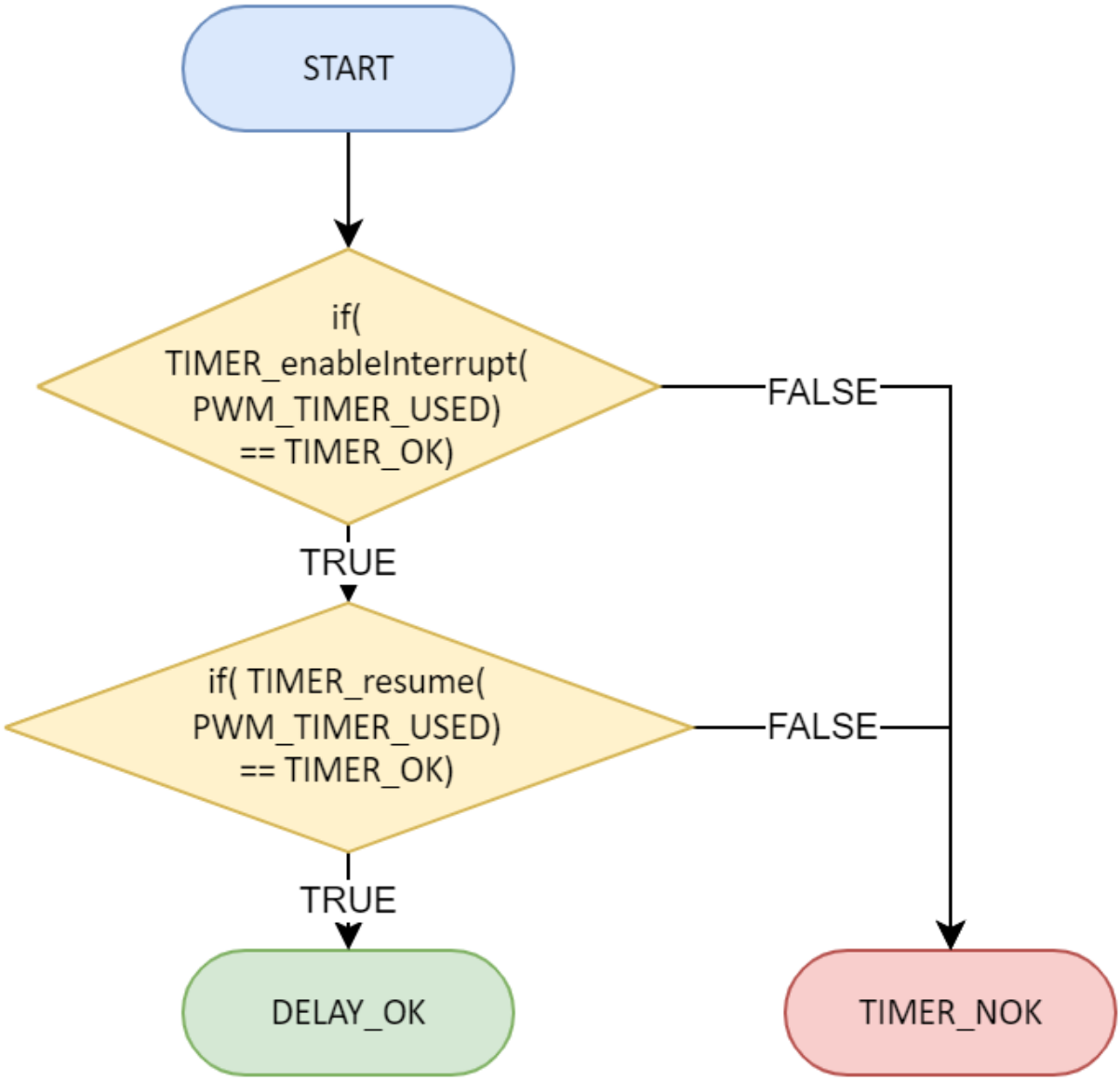
3.2.7.1. PWM_init



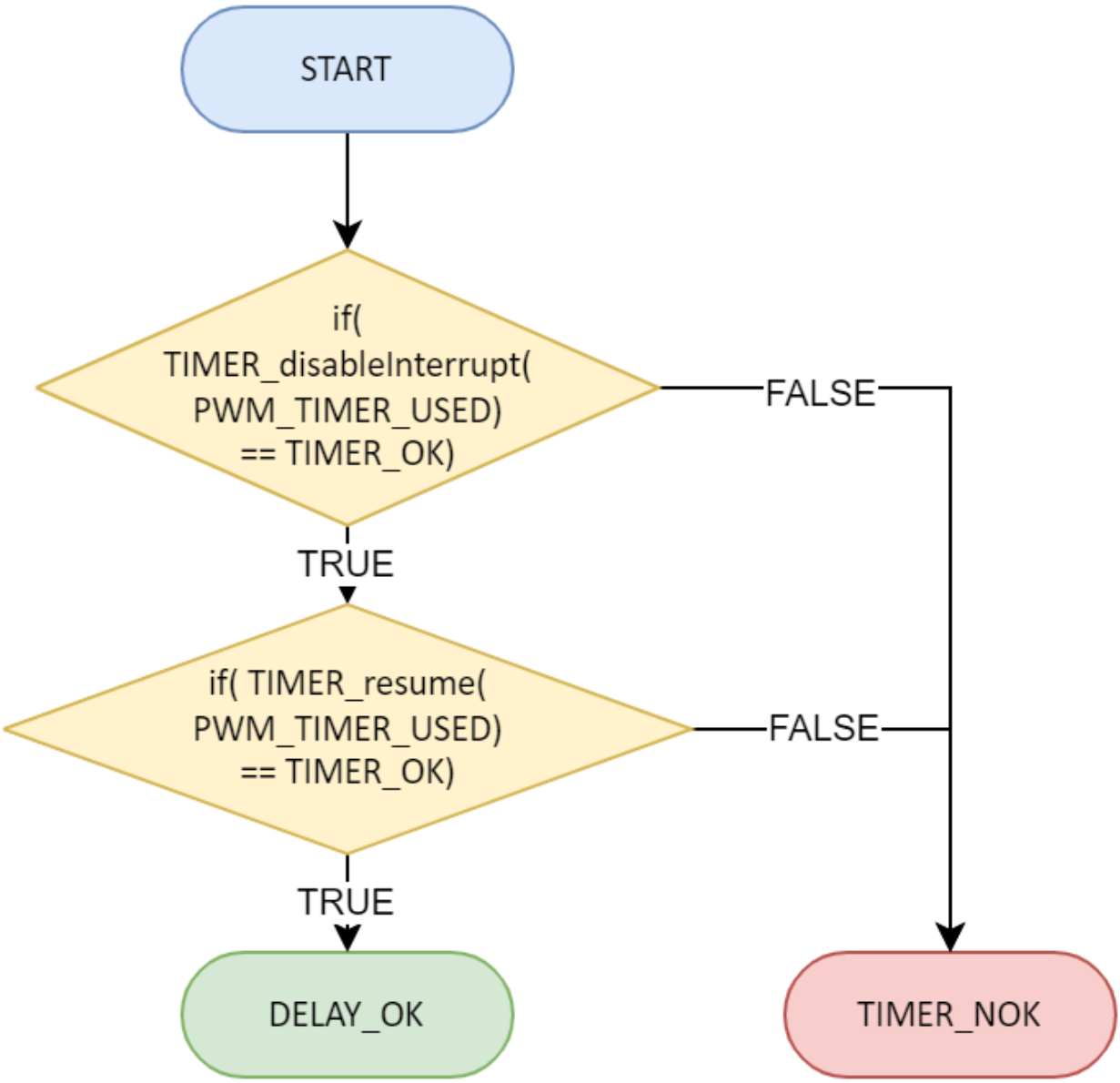
3.2.7.2. PWM_setDutyCycle



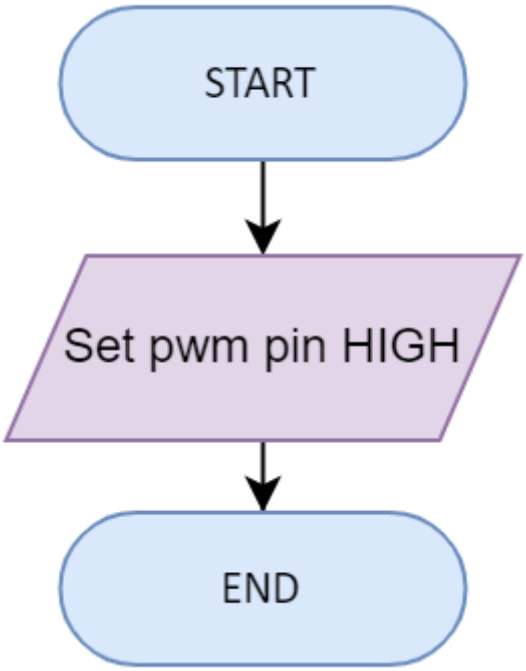
3.2.7.3. PWM_start



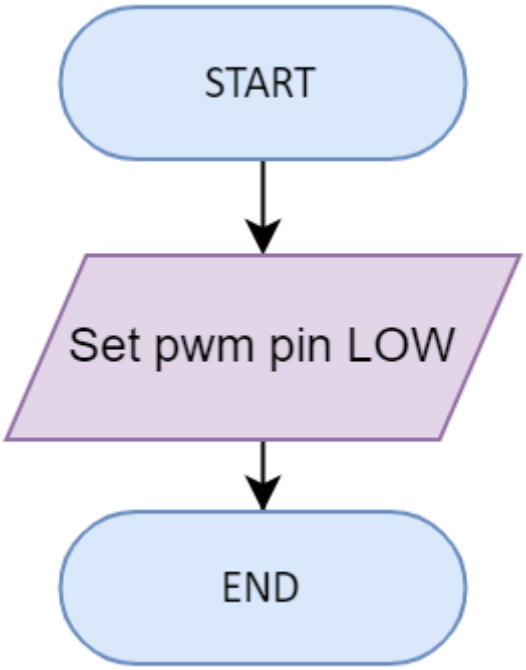
3.2.7.4. PWM_stop



3.2.7.5. PWM_onTask

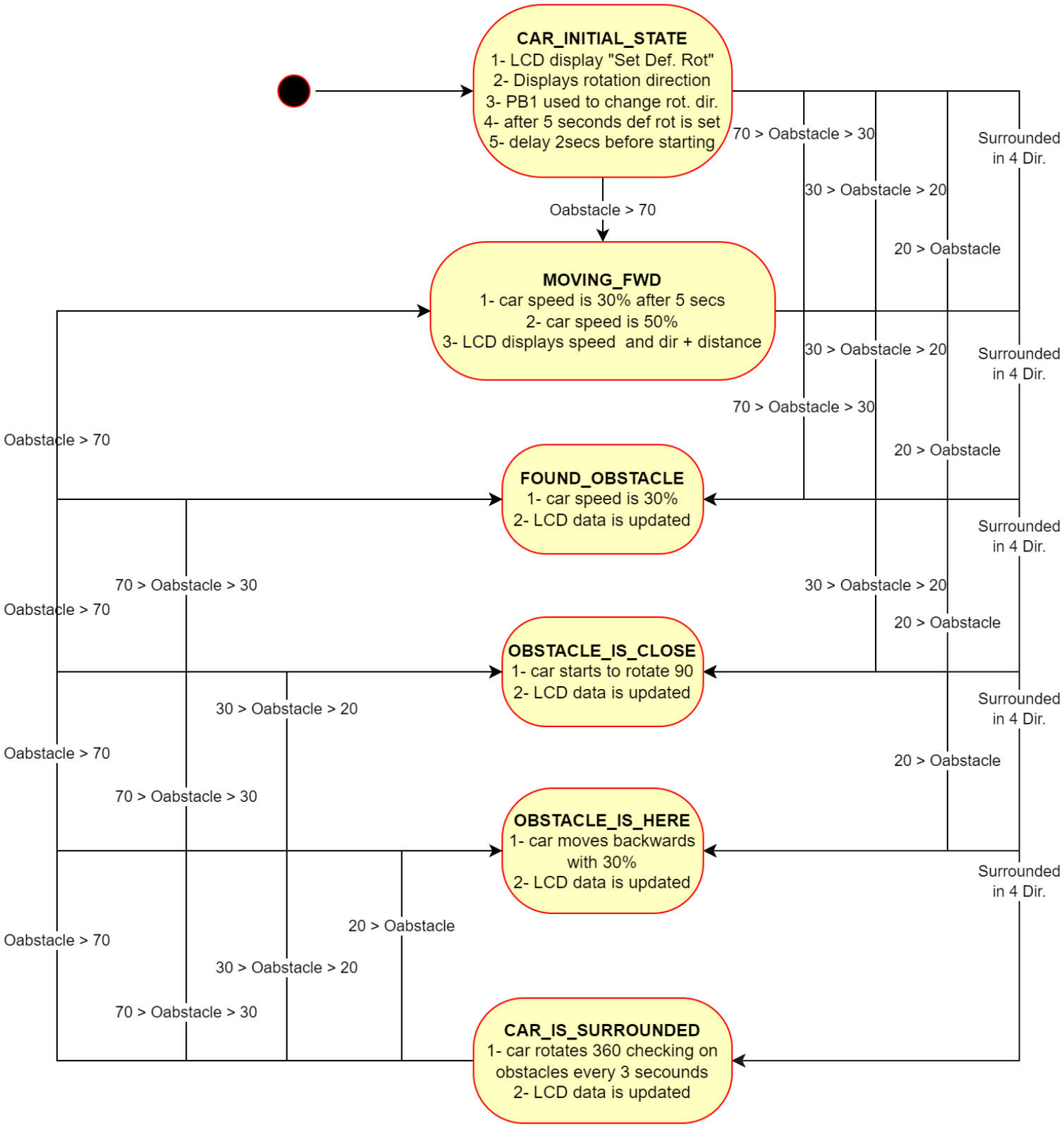


3.2.7.6. PWM_offTask



3.3. APP Layer

3.3.1. App State Diagram



4. Pre-compiling and linking configurations

4.1. EXTINT Driver

4.1.1. Linking Configurations

```
const st_EXTINT_config_t EXTINT_config [NUMBER_OF_TIMERS_USED] =
{
/*     EXTINT_number,      senseControlUsed*/
       {INT_0,             EXTINT_FALLING_EDGE},
       {INT_1,             EXTINT_FALLING_EDGE},
       {INT_2,             EXTINT_RISING_EDGE}
};
```

4.1.2. Pre_compiling Configurations

```
/************************_NUMBER_OF_EXTINTS_USED_/************************/
/*
 *    number of external interrupts used
 */

#define NUMBER_OF_EXTINTS_USED                3
```

4.2. Timer Driver

4.2.1. Linking configurations

```
const st_TIMER_config_t st_TIMER_config [NUMBER_OF_TIMERS_USED] =
{
/*     TIMER_number,waveformUsed,prescalerUsed*/
       {TIMER_0,           TIMER_OV,           TIMER_PRESCLNG_64},
       {TIMER_1,           TIMER_OV,           TIMER_PRESCLNG_64},
       {TIMER_2,           TIMER_OV,           TIMER_PRESCLNG_64}
};
```

4.2.2. Pre-compiled Configurations

```
/*********************_SYSTEM_OSCILLARTOR_CLOCK_FREQUENCY_*********************/
/*
 *     Enter microcontrollers frequency in Hz writing UL besides it
 */
#define F_CPU                                          8000000UL

/************************_NUMBER_OF_TIMERS_USED_************************/
/*
 *     number of timers used
 */
#define NUMBER_OF_TIMERS_USED                                3
```

4.3. ICU Driver

4.3.1. Linking Configuration

```
/* Predefined Pin options */
st_ICU_capturePins_t st_ICU_predefinedPins = {
        .PORT_D_PIN_2 = {
                .capturePin = PIN_2,
                .capturePort = PORT_D,
                .interruptNo = INT0
        },
        .PORT_D_PIN_3 = {
                .capturePin = PIN_3,
                .capturePort = PORT_D,
                .interruptNo = INT1
        },
        .PORT_B_PIN_2 = {
                .capturePin = PIN_2,
                .capturePort = PORT_B,
                .interruptNo = INT2
        }
};

/* Configuration */
static st_ICU_config_t st_gs_icuConfig = {
    // Capture Pin
        .icuCapturePin = PORT_B_PIN_2,

    // Callback Function on time received
        .timeReceivedCallbackFun = NULL // callback function
};
/* Functions */
st_ICU_config_t ICU_getConfig()
{
    switch (st_gs_icuConfig.icuCapturePin) {
        case PORT_D_PIN_2:
```

```
            st_gs_icuConfig.icuCapturePinData = st_ICU_predefinedPins.PORT_D_PIN_2;
            break;
        case PORT_D_PIN_3:
            st_gs_icuConfig.icuCapturePinData = st_ICU_predefinedPins.PORT_D_PIN_3;
            break;
        case PORT_B_PIN_2:
            st_gs_icuConfig.icuCapturePinData = st_ICU_predefinedPins.PORT_B_PIN_2;
            break;
    }
       return st_gs_icuConfig;
}

void ICU_setConfig(st_ICU_config_t st_a_icuConfig)
{
    st_gs_icuConfig = st_a_icuConfig;
}
```

4.4. PWM Driver

4.4.1. Linking Configurations

```
 const st_PWM_config_t st_PWM_config [NUMBER_OF_PWM_PINS] =
{
/*      PWM_#  PORT0_#,      PIN0_#*/
       {PWM_0, DIO_PORTD, DIO_PIN_4},
       {PWM_1, DIO_PORTD, DIO_PIN_5}
};
```

4.4.2. Pre-compiled Configurations

```
/*********************_SYSTEM_OSCILLARTOR_CLOCK_FREQUENCY_*******************/
/*
 *     Enter microcontrollers frequency in Hz writing UL besides it
 */
#define F_CPU                                              8000000UL

/*****************************_TIMER_USED_*********************************/
/*
 *     Timer used to generate delay
 */
#define PWM_TIMER_USED                                      TIMER_0

/************************_NUMBER_OF_PWM_PINS_************************/
/*
 *     Number of pwm pins used to generate PWM signal on them
 */
#define NUMBER_OF_PWM_PINS                                 2
#define PWM_0                                              0
#define PWM_1                                              1
```

4.5. Delay Driver

4.5.1. Pre-compiled Configurations

```
/************************_SYSTEM_OSCILLARTOR_CLOCK_FREQUENCY_***************/
/*
 *     Enter microcontrollers frequency in Hz writing UL besides it
 */
#define F_CPU                                                8000000UL

/************************_TIMER_USED_************************/
/*
 *     Timer used to generate delay
 */
#define DELAY_TIMER_USED                                    TIMER_2
```

4.6. DCM Driver

4.6.1. Linking Configurations

```
const st_DCM_config_t st_DCM_config [NUMBER_OF_DCMS_USED] =
{
/*      DCM_#, PORT0_#,        PIN0_#*, PORT1_#,    PIN1_#*/
       {DCM_0, DIO_PORTD, DIO_PIN_2, DIO_PORTD, DIO_PIN_3},
       {DCM_1, DIO_PORTD, DIO_PIN_6, DIO_PORTD, DIO_PIN_7}
};
```

4.6.2. Pre-compiled Configurations

```
/************************_NUMBER_OF_DCMS_USED_************************/
/*
 *     number of dcms used
 */

#define NUMBER_OF_DCMS_USED                2
```

4.7. US (ultrasonic) Driver

4.7.1. Linking Configuration

```
static st_US_config_t st_gs_usConfig = {
       DIO_PORTB, DIO_P3, DIO_P2
};
```

4.8. LCD Driver

4.8.1. pre-compiling configuration

```
/*****************************_SELECT_LCD_MODE*********************************/
/*
 * OPTION 1-> [ _4_BIT_MODE ]
 * OPTION 1-> [ _8_BIT_MODE ]
 */
#define LCD_MODE                _4_BIT_MODE

/*************************_SELECT_DATA_REGISTER*******************************/
#define LCD_DATA_PORT           DIO_PORT_A

/*******************************_SELECT_MC_PINS*********************************/
#define LCD_RS_PIN              DIO_PA1
#define LCD_RW_PIN              DIO_PA2
#define LCD_EN_PIN              DIO_PA3
#define LCD_DATA_4_PIN          DIO_PA4
#define LCD_DATA_5_PIN          DIO_PA5
#define LCD_DATA_6_PIN          DIO_PA6
#define LCD_DATA_7_PIN          DIO_PA7
```

4.9. KEYPAD Driver

4.9.1. pre-compiling configuration

```
/*******************************************************************************
 *  GLOBAL CONSTANT MACROS
 *******************************************************************************/
#define KEYPAD_NUMBER_OF_ROWS          3
#define KEYPAD_NUMBER_OF_COLUMNS 3

#define KEYPAD_R0_PORT                          DIO_PORT_C
#define KEYPAD_R1_PORT                          DIO_PORT_C
#define KEYPAD_R2_PORT                          DIO_PORT_C

#define KEYPAD_C0_PORT                          DIO_PORT_C
#define KEYPAD_C1_PORT                          DIO_PORT_C
#define KEYPAD_C2_PORT                          DIO_PORT_C

#define KEYPAD_R0_PIN                           DIO_PC2
#define KEYPAD_R1_PIN                           DIO_PC3
#define KEYPAD_R2_PIN                           DIO_PC4

#define KEYPAD_C0_PIN                           DIO_PC5
#define KEYPAD_C1_PIN                           DIO_PC6
#define KEYPAD_C2_PIN                           DIO_PC7
```

4.9.2. Linking configurations

```
const u8 u8_arr_g_keypadArr [KEYPAD_NUMBER_OF_ROWS][KEYPAD_NUMBER_OF_COLUMNS]  ={
                                                    {'1','2','3'},
                                                    {'4','5','6'},
                                                    {'7','8','9'} };
```

4.10. BUTTON Driver

4.10.1. pre-compiling configuration

```
/************************_NUMBER_OF_BUTTONS_USED_/************************/
/*
 *     number of buttons used
 */

#define NUMBER_OF_BUTTONS_USED   1
```

4.10.2. Linking configurations

```
const st_BUTTON_config_t st_BUTTON_config [NUMBER_OF_MOTORS_USED] =
{
/*      BUTTON_#,   PORT_#,    PIN_#*/
       {BUTTON_0, DIO_PORT_C, DIO_PC4}
};
```