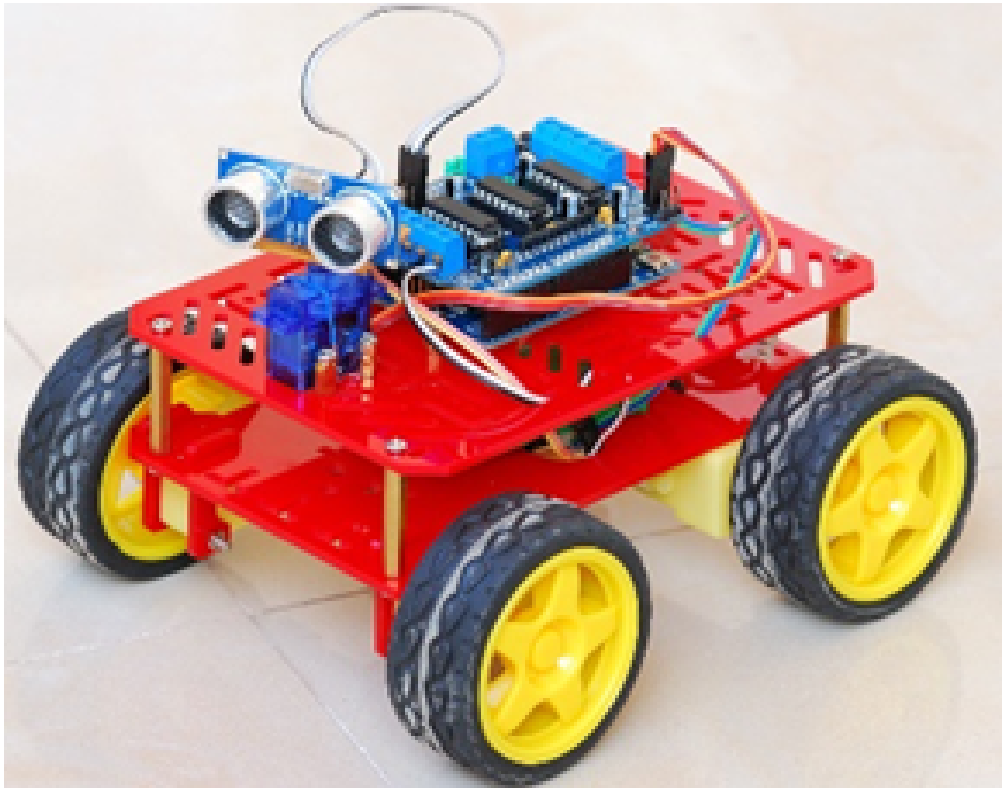


Design Obstacle Avoidance Car

Team no.2

Mario Saad

Nadeen Adel

Peter Essam

1. Introduction: 1
2. High Level Design: 2
2.1 Layered Architecture: 2
2.2 Modules Description: 3
2.3 Drivers' Documentation: 4
2.3.1 DIO: 4
2.3.2 Timer0: 5
2.3.3 External Interrupt: 6
2.3.4 BUTTON: 6
2.3.5 MOTOR: 7
2.3.6 ULTRASONIC: 7
2.3.7 ICU: 8
2.3.8 LCD: 9
2.3.9 KEYPAD: 9
2.3.10 PWM: 9
Configurations: 10
Functions Flowcharts: 10
MOTOR: 12
BUTTON: 14
External Interrupt: 14
Timer: 17
App State Machine: 22
App Flowchart: 24

1. Introduction:

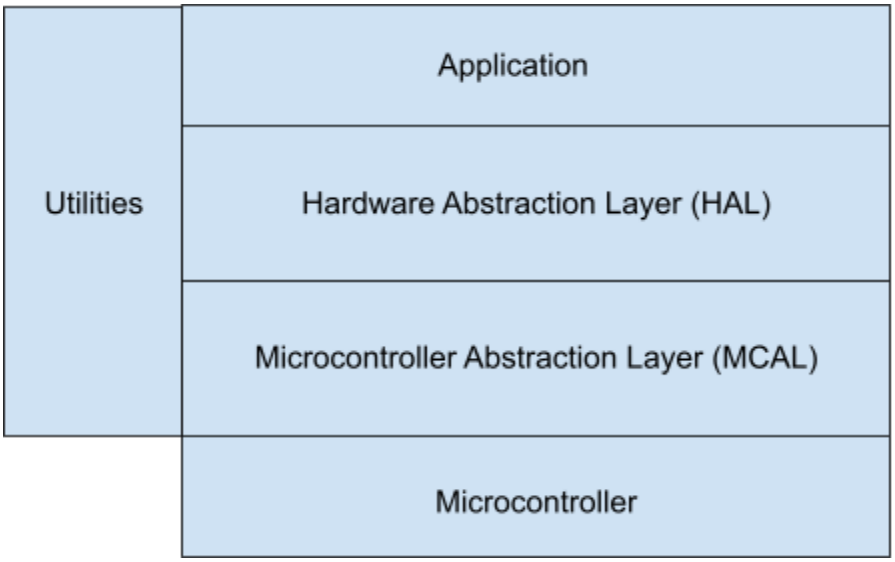
This project is a development of a robotic car system with a wide range of features and functionalities. It is built around the ATmega32 microcontroller and integrates components such as motors, a button, a keypad, an ultrasonic sensor, and an LCD display. The primary goal of this project is to create an autonomous car system that is capable of responding to its environment, navigating around obstacles, and adapting to user-defined settings.

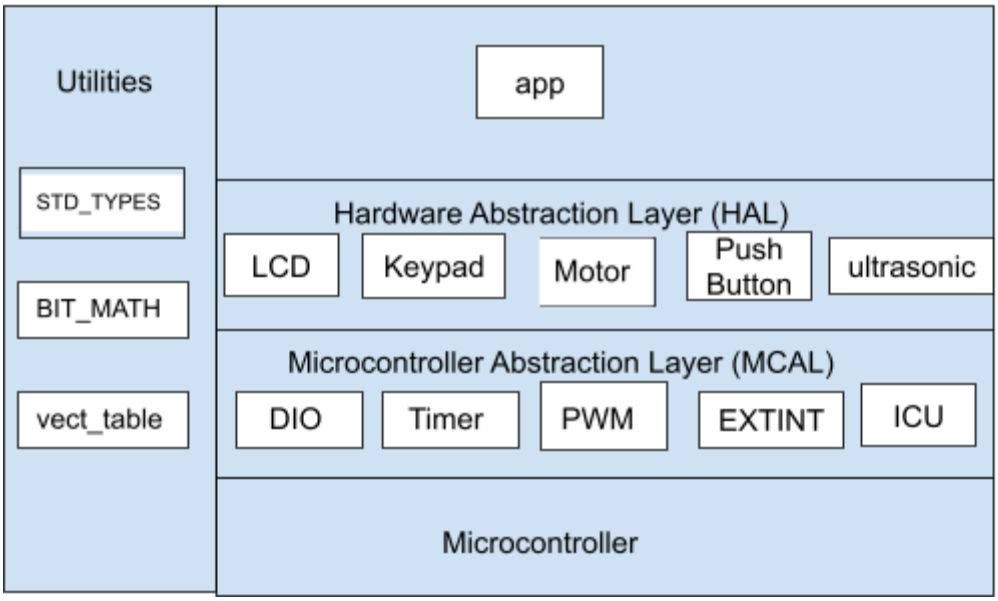
The system's core functionalities include managing the car's speed, direction, and obstacle detection. The car starts from a standstill and initially rotates to the right. The user can initiate its movement using a keypad, with options to start, stop, and change the default rotation direction. The robot intelligently detects objects in its path using an ultrasonic sensor, adjusts its speed, and takes appropriate actions to avoid collisions. Furthermore, if the car rotates 360 degrees without finding any obstacles, it will come to a halt.

This project is a demonstration of an autonomous robotic system with adaptive features, making it a valuable tool for applications such as indoor navigation, obstacle avoidance, and environmental monitoring.

2. High Level Design:

2.1 Layered Architecture:





2.2 Modules Description:

MCAL Layer:

- DIO: controls GPIO pins.
- Timer: controls timing in the code .
- External Interrupt: Handle external interrupt events.
- PWM: controls the speed of motors.
- ICU: calculate time.

HAL Layer:

- Button: dealing with the rotation button .
- Keypad : dealing with buttons (Start and Stop buttons)
- LCD: display state of the car.
- Motor: controls movement state of the car
- Ultrasonic: calculate distance using ICU.

Application Layer:

- main logic of the code

2.3 Drivers' Documentation:

2.3.1 DIO:

```
/*
 * Description :
 * Setup the direction of the required pin input/output and return error status.
 * If the input port number or pin number are not correct, The function will not handle the request.
 */
EN_dioError_t MDIO_setPinDirection (u8 u8_a_portNum,u8 u8_a_pinNumberber,u8 u8_a_pinDirection);

/*
 * Description :
 * Write the value Logic High or Logic Low on the required pin and return error status.
 * If the input port number or pin number are not correct, The function will not handle the request.
 * If the pin is input, this function will enable/disable the internal pull-up resistor.
 */
EN_dioError_t MDIO_setPinValue     (u8 u8_a_portNum,u8 u8_a_pinNumberber,u8 u8_a_value);

/*
 * Description :
 * Toggle The value of the pin and return error status.
 * If the input port number or pin number are not correct, The function will not handle the request.
 * If the output is high, The function will toggle it to low.
 * If the output is low, The function will toggle it to high.
 */
EN_dioError_t MDIO_togglePinValue (u8 u8_a_portNumber,u8 u8_a_pinNumber);


/*
 * Description :
 * Read and return the value for the required pin, it should be Logic High or Logic Low.
 * If the input port number or pin number are not correct, The function will return Logic Low.
 */
u8   MDIO_u8getPinValue (u8 u8_a_portNumber,u8 u8_a_pinNumber);
```

2.3.2 Timer0:

```
/*
* Description:
* Initializes Timer0 with the specified configuration, enabling the chosen timer mode, clock source, and related settings.
* This function configures Timer0 in one of the available modes: Normal, Phase-Correct PWM, CTC, or Fast PWM.
*
* Parameters:timer modes enum
*
* Returns:timer error state
*/
en_timer0ErrorState_t MTIMER0_init(en_timer0Mode_t u8_a_mode);

/*
* Description:
*This function is used to start Timer 0 by configuring the timer's clock prescaler. This function
*is responsible for setting the clock source and prescaler value for Timer 0
* Parameters:an enum of en_TIMER0_CLK_SELECT_t type that specifies the desired prescaler value.
*
* Returns:timer error state
*/
en_timer0ErrorState_t MTIMER0_startTimer(en_TIMER0_CLK_SELECT_t u8_a_prescaler);

/*
* Description:
* Stops Timer0 by disabling its clock source. This function clears the timer's clock source bits to stop its operation.
* Parameters:void
* Returns:void
*/
void MTIMER0_stop();
```

```c
/*
 * Description:
 * Setting a callback function that will be executed when a Timer 0 (TIM0) overflow interrupt event occurs
 * Parameters:void
 * Returns:void
 */
en_timer0ErrorState_t MTIMER0_setOVFCallback(void (*pv_a_CallbackFn)(void));


 /*
 * Description:
 * Function used to set desired delay in milliseconds using timer0
 * Parameters:Desired delay in milliseconds
 * Returns:void
 */

void delay_ms_UsingTimer(u16 u16_delay_ms);
```

2.3.3 External Interrupt:

```c
/*
description : used to initialize the global interrupt
arguments   : takes the state -enable or disable-
return      : return the error state, if ok returns EXTINT_OK ,else returns EXTINT_OK
*/
EN_EXTINT_ERROR SET_GLOBAL_INTERRUPT(EN_GLOBAL_INT state);


/*
description : used to initializes the external interrupt number and it's detecting type
arguments   : takes the external interrupt number (INT0,INT1 OR INT2) and sense control.
return      : return the error state, if ok returns EXTINT_OK ,else returns EXTINT_OK
*/
EN_EXTINT_ERROR EXTINT_init(EN_EXINT_NUMBER INTx ,EN_Sense_Control INTxSense);

/*
description : used to initialize call back function.
arguments   : takes the external interrupt number (INT0,INT1 OR INT2) and pointer to the wanted function for callback.
return      : return the error state, if ok returns EXTINT_OK ,else returns EXTINT_OK
*/
EN_EXTINT_ERROR EXTINT_CallBack(EN_EXINT_NUMBER INTx,void(*ptrfunc)(void));

#endif /* EXT_INTERRUPT_H_ */
```

2.3.4 BUTTON:

```c
/*
description   : used to initialize the button
arguments     : copy of the button number
*/
void HPushButtonOn_init(u8 Copy_u8buttonNum);

/*
description   : used to get the button value
arguments     : copy of the button number
return        : the button value
*/
u8 HPushButton_getValue(u8 Copy_u8buttonNum);


#endif /* BUTTON_INTERFACE_H_ */
```

2.3.5 MOTOR:

```
/*description : used to initialize the motor */
void HDCMotor_init(void);

/*description : used to make the motor start forward */
void HDCMOTOR_startForward(void);

/*description : used to stop the motor */
void HDCMOTOR_stop(void);

/*description : used to make the motor rotate */
void HDCMOTOR_Rotate(void);
```

2.3.6 ULTRASONIC:

```
/*
Description: Initializes the ultrasonic on a micro-controller with the
             specified configuration settings.
             */
void ultrasonic_vInit(void);


/*
Description: Used to get the distance between the sensor and the robot

Parameters:
A pointer to a variable to store the distance
             */
void ultrasonic_vGetDistance(float f64_a_distance);
```

2.3.7 ICU:

```c
/*
 * Description: This function configures the Timer1 in normal mode and initializes the External Interrupts to capture rising edges.
 * It sets up the required settings for Timer1, External Interrupts, and initializes the edge flag as RISING.
 * Then, it starts Timer1 to capture rising edges based on the specified prescaler.
 */
void ICU_RisingEdgeCapture(void);

/**
 * Description: This function retrieves the latest captured value from Timer1's Input Capture Register (ICR).
 * It stores this value in the provided pointer 'u32_l_ICR_value' for the caller to use and analyze.
 *
 * @param u32_l_ICR_value: A pointer to a 32-bit unsigned integer where the captured value will be stored.
 */
void ICU_getValue(u32 *var);

/**
 * Description: This function enables external interrupt INT2 and sets its sense control based on the provided parameters.
 * It first enables global interrupts, as this is a prerequisite for using external interrupts in AVR.
 * Then, it enables external interrupt INT2 by setting the corresponding bit in the General Interrupt Control Register (GICR).
 * Finally, it configures the sense control for INT2 based on the provided 'u8_a_senseControl' parameter, which can be either
 * falling edge or rising edge.
 *
 * @param u8_a_interruptId: The ID of the external interrupt, which is always INT2 in this function.
 * @param u8_a_senseControl: The desired sense control for INT2, which can be EXI_U8_SENSE_FALLING_EDGE or EXI_U8_SENSE_RISING_EDGE.
 */
void EXI_enablePIE( u8 u8_a_interruptId, u8 u8_a_senseControl );


/**
 * Description: This function initializes Timer1 in normal mode with or without interrupt enable.
 * It configures Timer1 to operate in normal mode (non-PWM) by clearing WGM10, WGM11, WGM12, and WGM13 bits in the Control Registers A and B.
 * Additionally, it sets the FOC1A and FOC1B bits as required for normal mode operation.
 * If interrupt enable is enabled, it also sets the global interrupt enable bit and enables the timer1 overflow interrupt (TOIE1).
 *
 * @param en_a_interrputEnable: Specify whether to enable (ENABLED) or disable (DISABLED) the timer1 overflow interrupt.
 * @return TIMER_ERROR if an invalid interrupt state is provided, otherwise TIMER_OK.
 */
EN_TIMER_ERROR_T TIMER_tmr1NormalModeInit(EN_TIMER_INTERRPUT_T en_a_interrputEnable);

/**
 * Description: This function starts Timer1 with the specified prescaler value.
 * It configures Timer1 to use the provided prescaler value for time base generation by setting the appropriate bits (CS10, CS11, and CS12)
 * in the Control Register B (TCCR1B).
 * The function accepts prescaler values of 1, 8, 64, 256, or 1024 and configures the timer accordingly.
 * If an invalid prescaler value is provided, the function returns TIMER_ERROR to indicate an error.
 *
 * @param u16_a_prescaler: The desired prescaler value for Timer1 (1, 8, 64, 256, or 1024).
 * @return TIMER_ERROR if an invalid prescaler value is provided, otherwise TIMER_OK.
 */
EN_TIMER_ERROR_T TIMER_tmr1Start(u8 u16_a_prescaler);

/**
 * Description: This function stops Timer1 by clearing the prescaler bits (CS10, CS11, and CS12) in the Control Register B (TCCR1B).
 * It effectively halts Timer1's operation and stops counting.
 */
void TIMER_tmr1Stop(void);
```

2.3.8 LCD:

```c
/**
 * Description: This function initializes the LCD module. It configures the direction of control and data pins,
 * performs a series of initialization commands, and sets up the LCD for normal operation.
 * The function also uses the "delay_ms_UsingTimer" function to add delays for stable operation.
 *
 * @return EN_lcdStates_t: Returns an enumeration indicating the initialization status (LCD_OK or LCDNOK).
 */
EN_lcdStates_t HLCD_Init(void);

/**
 * Description: This function writes a command to the LCD module. It configures the control pins for command mode and sends the high and low nibbles of the command to the LCD.
 * The function also handles the enable pin to trigger the LCD to read the command and adds a delay using the "delay_ms_UsingTimer" function.
 *
 * @param u8_a_command: The command to be sent to the LCD.
 * @return EN_lcdStates_t: Returns an enumeration indicating the status of the command write operation (LCD_OK or LCDNOK).
 */
EN_lcdStates_t HLCD_writeCMD(u8 u8_a_command);

/**
 * Description: This function writes a character to the LCD module. It configures the control pins for data mode (RS pin set to high) and sends the high and low nibbles of the character to the LCD.
 * The function also handles the enable pin to trigger the LCD to read the character and adds appropriate delays using the "delay_ms_UsingTimer" function.
 *
 * @param u8_a_Char: The character to be sent to the LCD.
 * @return EN_lcdStates_t: Returns an enumeration indicating the status of the character write operation (LCD_OK or LCDNOK).
 */
EN_lcdStates_t HLCD_writeChar(u8 u8_a_Char);

/**
 * Description: This function clears the display of the LCD by sending the command "LCD_CLEAR_DISPLAY" to the LCD module.
 * After sending the command, it waits for specific delays using the "delay_ms_UsingTimer" function to ensure proper operation.
 * The function returns an enumeration indicating the status of the clear display operation (LCD_OK or LCDNOK).
 *
 * @return EN_lcdStates_t: Returns an enumeration indicating the status of the clear display operation (LCD_OK or LCDNOK).
 */
EN_lcdStates_t HLCD_clearDisplay(void);

/**
 * Description: This function shifts the entire display to the left by sending the "LCD_DISPLAY_SHIFT_LEFT" command to the LCD module.
 * After sending the command, it waits for a specific delay using the "delay_ms_UsingTimer" function to ensure proper operation.
 * The function returns an enumeration indicating the status of the shift operation (LCD_OK or LCDNOK).
 *
 * @return EN_lcdStates_t: Returns an enumeration indicating the status of the display shift operation (LCD_OK or LCDNOK).
 */
EN_lcdStates_t HLCD_shiftLeft(void);


 * Description: This function is responsible for setting the cursor position on the LCD module.
 * It takes two parameters, u8_a_row and u8_a_col, to specify the desired row and column on the LCD display.
 * The function validates the input values to ensure they are within the range, and then sends the appropriate command to set the cursor position.
 * The function returns an enumeration indicating the status of the operation (LCD_OK or LCDNOK).
 *
 * @param u8_a_row: The desired row on the LCD display (0 or 1).
 * @param u8_a_col: The desired column on the LCD display (0 to 15).
 *
 * @return EN_lcdStates_t: Returns an enumeration indicating the status of the cursor position setting operation (LCD_OK or LCDNOK).
 */
EN_lcdStates_t HLCD_goToPosition(u8 u8_a_row, u8 u8_a_col);

/**
 * Description: This function is responsible for writing a string of characters to the LCD display.
 * It takes a pointer to a null-terminated string as input (pu8_a_str) and iterates through the characters in the string.
 * For each character, the function calls the HLCD_writeChar function to display it on the LCD.
 * The function continues this process until it reaches the null character ('\0') at the end of the string.
 *
 * @param pu8_a_str: A pointer to the null-terminated string to be displayed on the LCD.
 *
 * @return EN_lcdStates_t: Always returns LCD_OK, as there is no error checking in this function.
 */
EN_lcdStates_t HLCD_writeString(u8* pu8_a_str);

/**
 * Description: This function is responsible for displaying a decimal number (u32_a_number) on the LCD.
 * It converts the decimal number into a string of characters and then calls HLCD_writeString to display the string.
 *
 * @param u32_a_number: The decimal number to be displayed on the LCD.
 *
 * @return EN_lcdStates_t: Returns LCD_OK if the number is displayed successfully, although there is no error checking in this function.
 */
EN_lcdStates_t HLCD_WriteNumber(u32 u32_a_number);



/**
 * Description: This function is responsible for creating a custom character on the LCD.
 * It takes an array of 8 bytes (pu8_a_custom) that represent the custom character and a location (u8_a_Location)
 * in the CGRAM (Character Generator RAM) where the custom character will be stored.
 *
 * @param pu8_a_custom: An array of 8 bytes representing the custom character.
 * @param u8_a_Location: The location (0-7) in the CGRAM where the custom character will be stored.
 *
 * @return EN_lcdStates_t: Returns LCD_OK if the custom character is created successfully. It checks if the location is within valid range.
 */
EN_lcdStates_t HLCD_CreateSpecialChar(u8* pu8_a_custom, u8 u8_a_Location);
```

2.3.9 KEYPAD:

```
/*
 * Function: HKPD_init
 * Description: Initializes the keypad by configuring pins and setting their initial states.
 * Parameters: None.
 * Return Values:
 * - KEYPAD_OK: Initialization successful.
 * - KEYPAD_NOK: Initialization encountered an error.
 */
EN_keypadInitStatus_t HKPD_init(void);



/*
 * Function: HKPD_u8GetPressedKey
 * Description: Scans the keypad for a pressed key and returns the pressed key value, taking care of debouncing.
 * Parameters:
 * - pu8_a_key (output): Pointer to a variable that will store the pressed key value.
 * Return Values:
 * - KEYPAD_OK: A key was pressed and successfully debounced. The pressed key value is stored in pu8_a_key.
 * - KEYPAD_NOK: No key was pressed or an error occurred during key press detection. The value of pu8_a_key remains unchanged.
 */
EN_keypadInitStatus_t HKPD_u8GetPressedKey(u8* pu8_a_key);
```

2.3.10 PWM:

```
Description : this function initializes timer0 with normal mode Also enable peripheral and Global interrupt.
*/
void TIMER0_init(void);

/*
Function    : TIMER0_start
Description : this function set three clock bits with chosen prescaller in config file (timer starts when we call this function)
*/
void TIMER0_start(void);

/*
Function    : TIMER0_stop
Description : this function clear three clock bits (timer stops when we call this function)
*/
void TIMER0_stop(void);

/*
Function    : TIMER0_initPWM
Description : this function initializes all pwm pins as outputs and set high on them, also calls TIMER0_init ....
*/
void TIMER0_initPWM(void);

/*
Function    : TIMER0_setPwm
Description : this function calculates onTime and offTime , also calls TIMER0_start ....
Args        : DutyCycle (0--->100)
*/
void TIMER0_setPwm(u8 u8_a_dutyCycle);
```

Configurations:

DIO:

```
/*****************************************************************************/
/*                       Pin modes                                           */
/*****************************************************************************/
#define DIOMODE_INPUT     0
#define DIOMODE_OUTPUT    1

/*****************************************************************************/
/*                       Pin Direction Setting                               */
/*****************************************************************************/
#define DIOOUTPUT_LOW      0
#define DIOOUTPUT_HIGH     1

/*****************************************************************************/
/*                       Pin Pull Up Value                                   */
/*****************************************************************************/
#define DIOINPUT_FLOATING  0
#define DIOINPUT_PULLUP    1

/*****************************************************************************/
/*                       Pin Pull Up Configuration                           */
/*****************************************************************************/
#define DIOPULLUP_DISABLED 0
#define DIOPULLUP_ENABLED  1
```

```c
/*-----------------------------------------------------------------------------*/
typedef struct{
    EN_dio_port_t   dio_port;
    EN_dio_pin_t    dio_pin;
    EN_dio_mode_t   dio_mode;
    EN_dio_value_t  dio_initial_value;
    EN_dio_pullup_t dio_pullup_resistor;
}ST_DIO_ConfigType;

ST_DIO_ConfigType DIO_ConfigArray[];


/*****************************************************************************/
/*                       ENUMS DIO PRECOMPILED                               */
/*****************************************************************************/
typedef enum{
    PA=0,
    PB,
    PC,
    PD
}EN_DIO_Port_type;


typedef enum{
    OUTPUT,
    INFREE,
    INPULL
}EN_DIO_PinStatus_type;

typedef enum{
    LOW=0,
    HIGH,
}EN_DIO_PinVoltage_type;
```

```c
typedef enum{
    DIO_PORTA,
    DIO_PORTB,
    DIO_PORTC,
    DIO_PORTD
}EN_dio_port_t;

/*****************************************************************************/
/*                          DIO PINS                                         */
/*****************************************************************************/
typedef enum{
    DIO_PIN0,
    DIO_PIN1,
    DIO_PIN2,
    DIO_PIN3,
    DIO_PIN4,
    DIO_PIN5,
    DIO_PIN6,
    DIO_PIN7
}EN_dio_pin_t;

/*****************************************************************************/
/*                        DIO PIN MODE DIRECTION                             */
/*****************************************************************************/
typedef enum{
    DIO_MODE_INPUT,
    DIO_MODE_OUTPUT
}EN_dio_mode_t;

/*****************************************************************************/
/*                          DIO PIN VALUE                                    */
/*****************************************************************************/
typedef enum{
    DIO_HIGH,
    DIO_LOW
}EN_dio_value_t;

/*****************************************************************************/
/*                          DIO PIN PULL UP CONFIG                           */
/*****************************************************************************/
typedef enum{
    DIO_PULLUP_DISABLED,
    DIO_PULLUP_ENABLED
}EN dio pullup t;
```

External Interrupt:

```
/*******************************************************************************/
/*                      PIN OF EXT INTERRUPT                                   */
/*******************************************************************************/
#define EXT_INTERRUPT_PINS          1

/*******************************************************************************/
/*                     MCUCR register Bits                                     */
/*******************************************************************************/
/*
Enum: EN_MCUCR_REG_BITS
Description: An enumeration that defines the bit fields for the `MCUCR` register on a micro-controller.

Members:
-  MCUCR_REG_ISC00_BITS : Represents the bit field for the `ISC00` bit of the `MCUCR` register.
-  MCUCR_REG_ISC01_BITS : Represents the bit field for the `ISC01` bit of the `MCUCR` register.
-  MCUCR_REG_ISC10_BITS : Represents the bit field for the `ISC10` bit of the `MCUCR` register.
-  MCUCR_REG_ISC11_BITS : Represents the bit field for the `ISC11` bit of the `MCUCR` register.


Overall, the EN_MCUCR_REG_BITS enumeration provides a way to represent and manage the individual bit
fields within the `MCUCR` register on a micro-controller in a standardized and easy-to-understand manner.
By using this enumeration, the software can read and modify the individual bits within this register as
needed for interrupt configuration and other purposes.
*/

typedef enum
{
    MCUCR_REG_ISC00_BITS = 0,
    MCUCR_REG_ISC01_BITS,
    MCUCR_REG_ISC10_BITS,
    MCUCR_REG_ISC11_BITS

}EN_MCUCR_REG_BITS;


typedef enum
{
    MCUCSR_REG_ISC2_BITS = 6,

}EN_MCUCSR_REG_BITS;

/*******************************************************************************/
/*                     GICR register Bits                                      */
/*******************************************************************************/
/*
Enum: EN_GICR_REG_BITS
Description: An enumeration that defines the bit fields for the `GICR` register on a micro-controller.

Members:
-  GICR_REG_INT2_BITS : Represents the bit field for the `INT2` bit of the `GICR` register.
-  GICR_REG_INT0_BITS : Represents the bit field for the `INT0` bit of the `GICR` register.
-  GICR_REG_INT1_BITS : Represents the bit field for the `INT1` bit of the `GICR` register.


Overall, the EN_GICR_REG_BITS enumeration provides a way to represent and manage the individual bit fields
within the `GICR` register on a micro-controller in a standardized and easy-to-understand manner.
By using this enumeration, the software can read and modify the individual bits within this register
as needed for interrupt configuration and other purposes.
*/

typedef enum
{
    GICR_REG_INT2_BITS = 5,
    GICR_REG_INT0_BITS,
    GICR_REG_INT1_BITS

}EN_GICR_REG_BITS;
```

```c
typedef enum
{
    GIFR_REG_INTF2_BITS = 5,
    GIFR_REG_INTF0_BITS,
    GIFR_REG_INTF1_BITS

}EN_GIFR_REG_BITS;

/*******************************************************************************/
/*                      EXT_INTERRUPT_Sense_Control                            */
/*******************************************************************************/
/*
Members:
- LOW_LEVEL_SENSE_CONTROL    : Represents the sense control mode where the interrupt is triggered when
                               the input signal is at a low level.
- ANY_LOGICAL_SENSE_CONTROL  : Represents the sense control mode where the interrupt is triggered when
                               there is any change in the logical en_g_state of the input signal.
- FALLING_EDGE_SENSE_CONTROL : Represents the sense control mode where the interrupt is triggered when
                               the input signal changes from a high level to a low level.
- RISING_EDGE_SENSE_CONTROL  : Represents the sense control mode where the interrupt is triggered when
                               the input signal changes from a low level to a high level.

Overall, the EN_EXT_INTERRUPT_Sense_Control enumeration provides a way to represent and manage the
different sense control modes for external interrupts on a micro-controller in a standardized and
easy-to-understand manner. By using this enumeration, the software can configure and handle external
interrupts based on the desired sense control mode for the specific input signal being used.
*/
typedef enum
{
    LOW_LEVEL_SENSE_CONTROL = 0,
    ANY_LOGICAL_SENSE_CONTROL,
    FALLING_EDGE_SENSE_CONTROL,
    RISING_EDGE_SENSE_CONTROL

}EN_EXT_INTERRUPT_Sense_Control;


typedef enum
{
    EXT0_INTERRUPTS = 0,
    EXT1_INTERRUPTS,
    EXT2_INTERRUPTS
}EN_EXT_INTERRUPTS;


/*******************************************************************************/
/*                      EXT_INTERRUPTS STRUCT CONFIG                           */
/*******************************************************************************/
/*
Struct: ST_EXT_INTERRUPTS_CFG
Description: A structure that contains the configuration settings for an external
             interrupt on a micro-controller.
Members:
-  INTERRUPT_EXTERNAL_HANDLER : A function pointer to the interrupt service routine (ISR)
                                for the external interrupt(call-back function).
-  EXTERNAL_INTERRUPRT_Number : An instance of the EN_EXT_INTERRUPTS enum that specifies the
                                external interrupt number to be configured.
-  EXTERNAL_INTERRUPRT_Sense_Control : An instance of the EN_EXT_INTERRUPT_Sense_Control enum
                                       that specifies the sense control mode for the external interrupt.

Overall, the ST_EXT_INTERRUPTS_CFG structure provides a way to represent and manage the configuration
settings for an external interrupt on a micro-controller in a standardized and easy-to-understand manner.
By using this structure, the software can configure and handle external interrupts based on the desired
interrupt number and sense control mode, and execute the appropriate ISR when the interrupt is triggered.
*/
typedef struct
{
    void(*INTERRUPT_EXTERNAL_HANDLER)(void);
    EN_EXT_INTERRUPTS EXTERNAL_INTERRUPRT_Number;
    EN_EXT_INTERRUPT_Sense_Control EXTERNAL_INTERRUPRT_Sense_Control;

}ST_EXT_INTERRUPTS_CFG;


const ST_EXT_INTERRUPTS_CFG A_interruptConfig[EXT_INTERRUPT_PINS];
```

ICU:

```
/*******************************************************************************/
/*                          EXT INT NUMBERS                                    */
/*******************************************************************************/
typedef enum {
    EXI_U8_INT0,
    EXI_U8_INT1,
    EXI_U8_INT2
}EN_ICU_usedExti_t;

/*******************************************************************************/
/*                          EXT INT SENSE CONTROL                              */
/*******************************************************************************/
typedef enum {
    EXI_U8_SENSE_LOW_LEVEL,
    EXI_U8_SENSE_LOGICAL_CHANGE,
    EXI_U8_SENSE_FALLING_EDGE,
    EXI_U8_SENSE_RISING_EDGE
}EN_ICU_senseControl_t;

/*******************************************************************************/
/*                          EXT INT SENSE MODE                                 */
/*******************************************************************************/
/* Interrupts Sense Control */
typedef enum {
    ISR_DISABLED,
    ISR_ENABLED
}EN_ICU_timer1ISR_t;


/*******************************************************************************/
/*                          EXT INT && ICU STRUCT                              */
/*******************************************************************************/
typedef struct
{
    EN_ICU_usedExti_t ICU_exti;
    EN_ICU_senseControl_t ICU_firstSenseControl;
    EN_ICU_senseControl_t ICU_secondSenseControl;
    EN_ICU_timer1ISR_t timer1_ISR;
}ST_ICU_g_Config_t;
```

```
/*******************************************************************************/
/*                          GLOBAL ARRAY OF STRUCT                             */
/*******************************************************************************/
extern ST_ICU_g_Config_t ST_g_softwareICU[1];
/*******************************************************************************/
/*                          ICU FLAG                                           */
/*******************************************************************************/
typedef enum {
    RISING,
    FALLING
    }EN_icuEdgeFlag;
```

PWM:

```
/*-------------------------------------------------------------------------------------------*/
#define TIMER_STOPPED                                0
#define TIMER_NO_PRESCALER                           1
#define TIMER_8_PRESCALER                            2
#define TIMER_64_PRESCALER                           3
#define TIMER_256_PRESCALER                          4
#define TIMER_1024_PRESCALER                         5
#define TIMER_EXTERNAL_CLOCK_SOURCE_FALLING_EDGE     6
#define TIMER_EXTERNAL_CLOCK_SOURCE_RISING_EDGE      7


/*******************************************************************************************/
/*                          MACROS TIMER PRESCALER                                         */
/*******************************************************************************************/
/*Timer Prescaler Options:
 * 0- TIMER_STOPPED
 * 1- TIMER_NO_PRESCALER
 * 2- TIMER_8_PRESCALER
 * 3- TIMER_64_PRESCALER
 * 4- TIMER_256_PRESCALER
 * 5- TIMER_1024_PRESCALER
 * 6- TIMER_EXTERNAL_CLOCK_SOURCE_FALLING_EDGE
 * 7- TIMER_EXTERNAL_CLOCK_SOURCE_RISING_EDGE
 */
#define TIMER_SET_PRESCALER    TIMER_1024_PRESCALER
#define  REG_SIZE                                256

/*******************************************************************************************/
/*                          MACROS PWM NUMBERS                                             */
/*******************************************************************************************/
/*configuration of pwm pins*/
#define PWM_PINS_NUMBER     2


typedef struct ST_PWM_PINS_CONFIGS{

    EN_DIO_Pin_type en_pwm_pin ;

}ST_PWM_PINS_CONFIGS;
```

```c
typedef enum {
    INTERRUPT,
    POLLING
}EN_TIMER_mode_T;

#define FACTOR                              4
#define OC0_PIN_DIR                         DDRB
#define OC0_PIN                              3
#define GLOBAL_INTERRUPT_ENABLE_BIT          7

/* timer 0 macros */
#define MAX_TIMER_DELAY                     (MAX_DELAY * 65535)
#define MAX_DELAY                           (0.262144f) // in sec
#define MAX_COUNTS                           256
#define TICK_TIME                           (0.001024f) // in sec


#define MAX_TIMER_DELAY_8_PRESCALLER        (MAX_DELAY_8_PRESCALLER * 65535)
#define MAX_DELAY_8_PRESCALLER              (0.000256f) // in sec
#define TICK_TIME_8_PRESCALLER              (0.000001f) // in sec
#define SECOND_OPERATOR                     (1000.0f)
#define MICRO_SECOND_OPERATOR               (1000000.0f)
#define NO_PRESCALER                        1

/*error definitions*/
typedef enum {
    TIMER_OK, TIMER_ERROR
} EN_TIMER_ERROR_T;

typedef enum {
    ENABLED, DISABLED
} EN_TIMER_INTERRPUT_T;
```

TIMER:

```
typedef enum
{
    TMR_OVERFLOW_MODE,
    TMR_CTC_MODE,
    TMR_PWM_MODE,
    TMR_COUNTER_MODE,
    TMR_MAX_TIMERMODES
}EN_TimerMode_t;

typedef enum
{
    TMR_INTERNAL,
    TMR_EXTERNAL
}EN_TimerClockSource_t;
typedef enum {
    TMR_ENABLED,
    TMR_DISABLED
}EN_TimerEnable_t;

typedef enum {
    TMR_ISR_ENABLED,
    TMR_ISR_DISABLED
}EN_TimerISREnable_t;

typedef enum {
    TMR_MODULE_CLK,
    TMR_RISING_EDGE,
    TMR_FALLING_EDGE,
}EN_TimerClockMode_t;

typedef enum {
    TMR_NORMAL_PORT_OPERATION_OC_PIN_DISCONNECTED,
    TMR_TOGGLE_OC_PIN_ON_COMPARE_MATCH,
    TMR_CLEAR_OC_PIN_ON_COMPARE_MATCH,
    TMR_SET_OC_PIN_ON_COMPARE_MATCH
}EN_TimerCompMatchOutputMode_t;
```

```
typedef enum
{
    TIMER0,                     /*0*/
    TIMER1,                     /*1*/
    TIMER2,                     /*2*/
    MAX_TIMERS                  /*on microcontroller*/
}EN_TimerChannel_t;

typedef enum
{
    TMR_NO_CLOCK_SOURCE_TIMER_COUNTER_ATOPPED,
    TMR_NO_PRESCALING,
    TMR_CLK_8_FROM_PRESCALER,
    TMR_CLK_64_FROM_PRESCALER,
    TMR_CLK_256_FROM_PRESCALER,
    TMR_CLK_1024_FROM_PRESCALER,
    TMR_EXT_CLOCK_SOURCE_ON_T1_PIN_CLK_ON_FALLING_EDGE,
    TMR_EXT_CLOCK_SOURCE_ON_T1_PIN_CLK_ON_RISING_EDGE
}EN_TimerPrescaler_t;

/*****************************************************************************/
/*                      TMR Configurations  Structure                        */
/*****************************************************************************/
typedef struct
{
    EN_TimerChannel_t TimerChannel;
    EN_TimerEnable_t TimerEnable;                    /*Timer Enable en_g_state*/
    EN_TimerMode_t TimerMode;                       /* Mode, and OVF, CTC, PWM Timer mode Settings*/
    EN_TimerCompMatchOutputMode_t TimerCtcMode;
    EN_TimerClockSource_t ClockSource;             /* clock source internal/external according to this it operates as timer or counter*/
    EN_TimerClockMode_t ClockMode;                /* Clock Mode for counter rising, falling,......*/
    EN_TimerPrescaler_t ClockPrescaler;          /*Clock Prescaler*/
    EN_TimerISREnable_t ISREnable;              /*ISR Enable en_g_state*/
}ST_TimerConfig_t;


const ST_TimerConfig_t* Tmr_ConfigGet(void);
```

ULTRASONIC:

```
/*******************************************************************************/
/*                          ULTRASONIC STRUCT CONFIG                           */
/*******************************************************************************/
typedef struct
{
    ST_DIO_ConfigType triggerpin;
    ST_DIO_ConfigType echopin;
}ST_ultrasonicPins;


static ST_ultrasonicPins ultra =
{
    .triggerpin =
    {
        .dio_port = DIO_PORTB,
        .dio_pin = DIO_PIN3,
        .dio_mode = DIO_MODE_OUTPUT,
        .dio_initial_value = DIO_LOW,
        .dio_pullup_resistor = DIO_PULLUP_DISABLED
    }
};
```

PUSH BUTTON:

```
/*******************************************************************************/
/*                      PUSH_BTN_state_t                                       */
/*******************************************************************************/
/*
Enum: EN_PUSH_BTN_state_t
Description : An enumeration that defines two possible states for a push button: pressed or released.
Members:
-  PUSH_BTN_STATE_PRESSED : Represents the en_g_state of a push button when it is pressed down or activated.
-  PUSH_BTN_STATE_RELEASED : Represents the en_g_state of a push button when it is not pressed or deactivated.

Overall, the EN_PUSH_BTN_state_t enumeration provides a way to represent the two possible states of a push
button in a standardized and easy-to-understand manner. By using this enumeration, the software can check the
en_g_state of a push button and take appropriate action based on whether it is pressed or released.
*/
typedef enum
{
    PUSH_BTN_STATE_PRESSED = 0,
    PUSH_BTN_STATE_RELEASED
}EN_PUSH_BTN_state_t;

/*******************************************************************************/
/*                      PUSH_BTN_active_t                                      */
/*******************************************************************************/
/*
Enum: EN_PUSH_BTN_active_t
Description: An enumeration that defines two possible active states for a push button: pull-up or pull-down.

Members:
-  PUSH_BTN_PULL_UP : Represents the active en_g_state of a push button when it is connected to a pull-up resistor.
                      In this en_g_state, the button is normally open and the pull-up resistor pulls the voltage of
                      the pin to a high en_g_state.
-  PUSH_BTN_PULL_DOWN : Represents the active en_g_state of a push button when it is connected to a pull-down resistor.
                        In this en_g_state, the button is normally closed and the pull-down resistor pulls the voltage
                        of the pin to a low en_g_state.

Overall, the EN_PUSH_BTN_active_t enumeration provides a way to represent the two possible active states of a
push button in a standardized and easy-to-understand manner. By using this enumeration, the software can
determine the active en_g_state of a push button and configure the pin accordingly.
*/
typedef enum
{
    PUSH_BTN_PULL_UP = 0,
    PUSH_BTN_PULL_DOWN
}EN_PUSH_BTN_active_t;
```

```
/*******************************************************************************/
/*                      PUSH_BTN_STRUCT CONFIG                                 */
/*******************************************************************************/
/*
Struct                  : ST_PUSH_BTN_t
Description             : A structure that contains the configuration and current en_g_state information for a
                          push button.
Members:
-  PUSH_BTN_pin         : An instance of the ST_pin_config_t struct that contains the configuration settings
                          for the pin used by the push button.
-  PUSH_BTN_state       : An instance of the EN_PUSH_BTN_state_t enum that represents the current en_g_state of
                          the push button (pressed or released).
-  PUSH_BTN_connection  : An instance of the EN_PUSH_BTN_active_t enum that represents the active en_g_state of
                          the push button (pull-up or pull-down).

Overall, the ST_PUSH_BTN_t structure provides a standardized way to represent and manage the configuration
and en_g_state information for a push button on a micro-controller. By using this structure, the software can easily
read the current en_g_state of the push button and take appropriate action based on its configuration and
connection type. The use of enums for the en_g_state and connection fields allows for consistent and
easy-to-understand representation of these values.
*/
typedef struct
{
    ST_DIO_ConfigType PUSH_BTN_pin;
    EN_PUSH_BTN_state_t PUSH_BTN_state;
    EN_PUSH_BTN_active_t PUSH_BTN_connection;
}ST_PUSH_BTN_t;
```

MOTOR:

```
/*******************************************************************************/
/* DCM Macros */
#define MOTORS_NUMBER                   2
#define ZERO_SPEED                      0
#define MAX_DUTY_CYCLE                  100
#define PERIOD_TIME                     10
#define ROTATION_DUTY_CYCLE             50

/*type definition*/
typedef enum {
    DCM_OK,
    DCM_ERROR
}EN_DCM_ERROR_T;

typedef enum {
    MOTOR_RIGHT,
    MOTOR_LEFT
}EN_DCM_MOTORSIDE;

typedef enum {
    FALSE,
    TRUE
}EN_DCM_FLAG;

extern ST_DCM_g_Config_t ST_g_carMotors[2];
```

```
/*******************************************************************************/
/*                          DC MOTOR CONFIG STRUCT                             */
/*******************************************************************************/
typedef struct ST_DCM_g_Config_t {
    u8 DCM_g_motEnPinNumber0;
    u8 DCM_g_motEnPinNumber1;
    u8 DCM_g_motEnPortNumber;
}ST_DCM_g_Config_t;
```

LCD:

```
#define LCD_DATA_PORT       PORTA
#define LCD_CONTROL_PORT    PORTA
#define REG_PORTA           *((volatile u8*)0x3B)
#define LCD_RS_PIN          PIN1
#define LCD_RW_PIN          PIN2
#define LCD_EN_PIN          PIN3
#define LCD_D4              PIN4
#define LCD_D5              PIN5
#define LCD_D6              PIN6
#define LCD_D7              PIN7
```

Keypad:

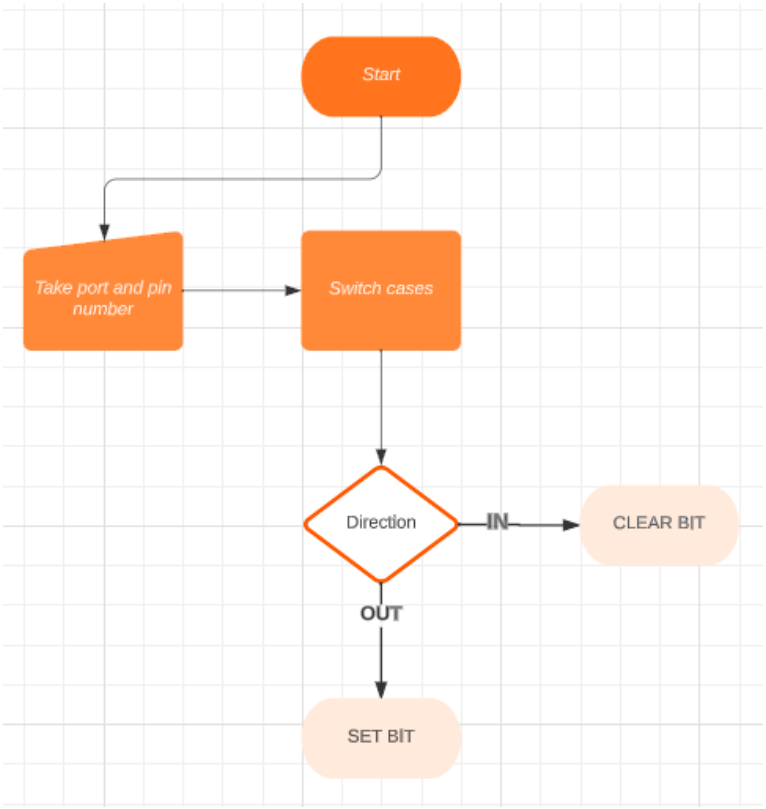
```
#define Column_num  3   //Number of columns
#define DEBOUNCING_DELAY 500
#define Row_num     3   //Number of rows


#define POOLING         1
#define NO_POOLING      2

#define PRESSED         0
#define RELEASED        1
#define DEBOUNCING      2
```

```
typedef enum{
    KEYPAD_OK = 0,
    KEYPAD_NOK
}EN_keypadInitStatus_t;
```

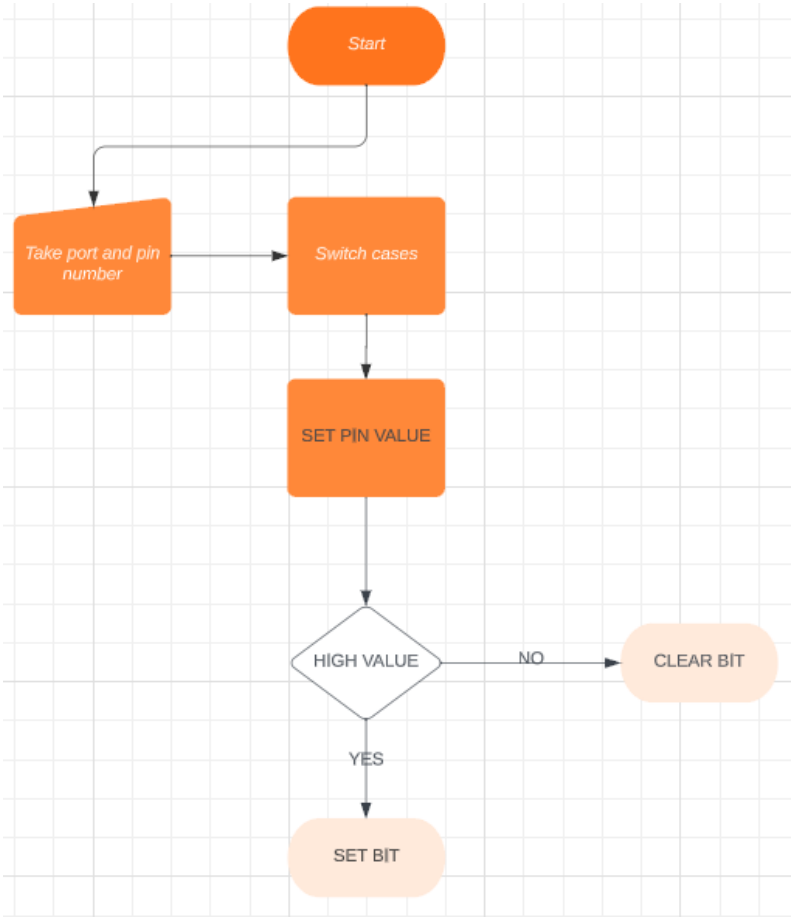
Functions Flowcharts:

DIO:

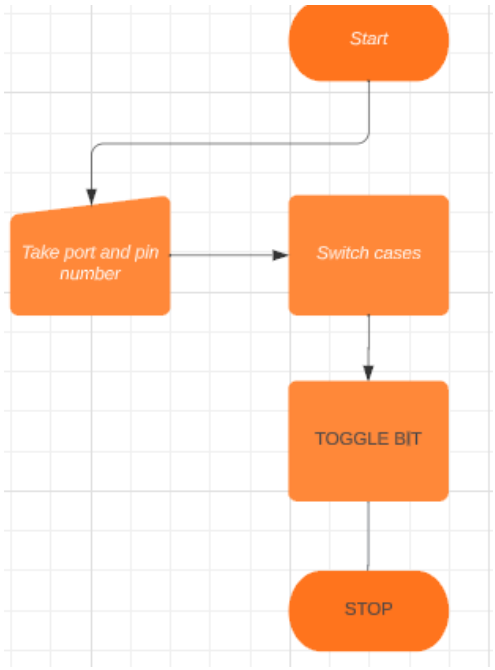
void MDIO_voidSetPinDirection (u8 Copy_u8Port , u8 Copy_u8Pin , u8 Copy_u8Dir)



void MDIO_voidSetPinValue(u8 Copy_u8Port,u8 Copy_u8Pin,u8 Copy_u8Value)

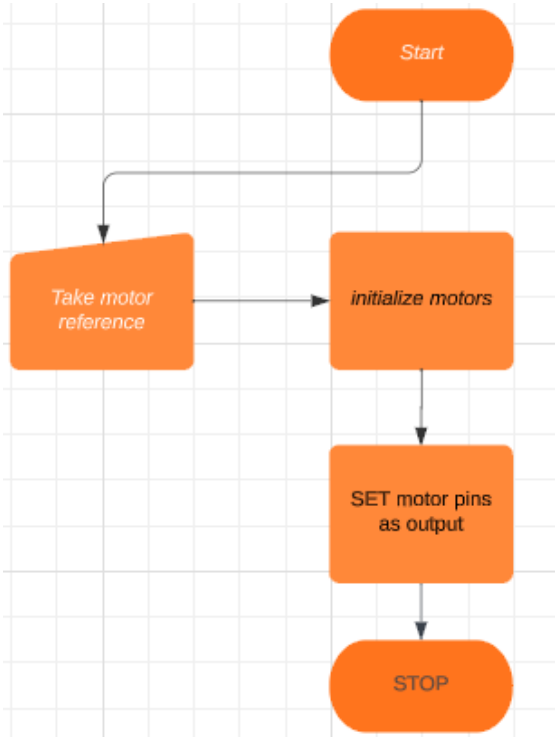


void MDIO_voidTogglePinValue(u8 Copy_u8Port , u8 Copy_u8Pin)

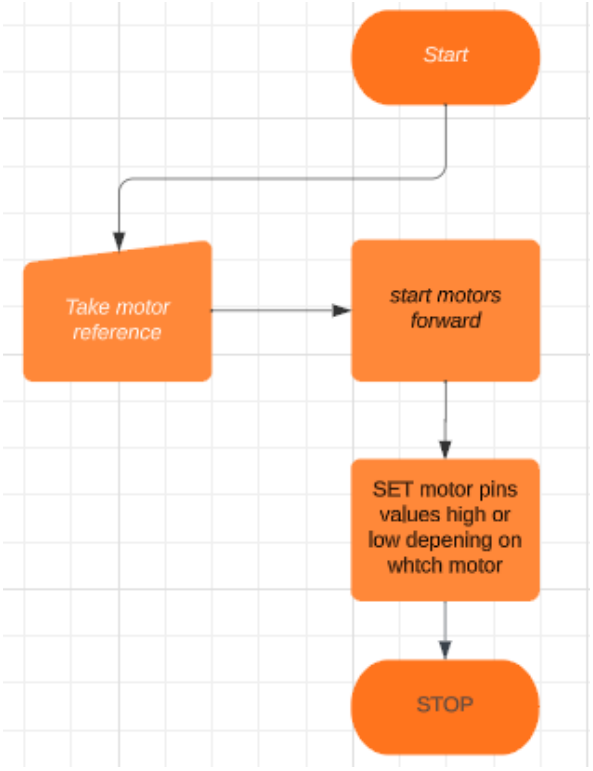


MOTOR:

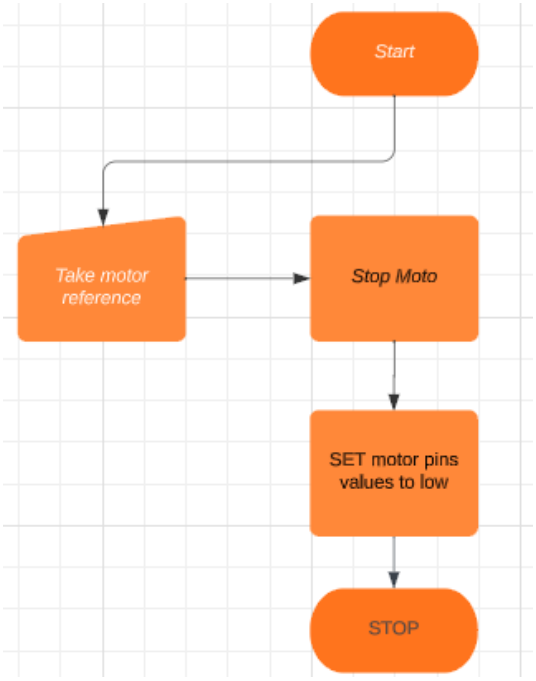
Init function



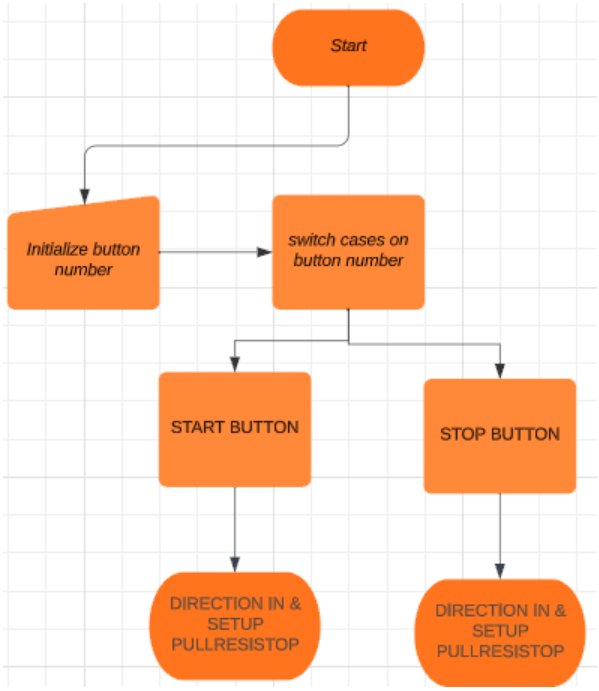
start forward function



Stop function

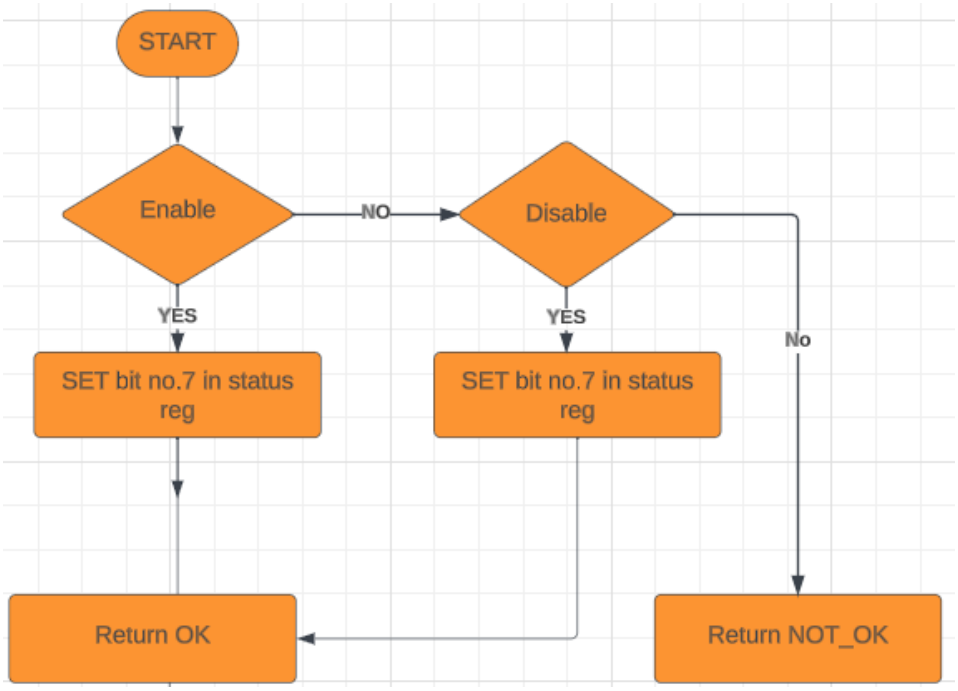


BUTTON:

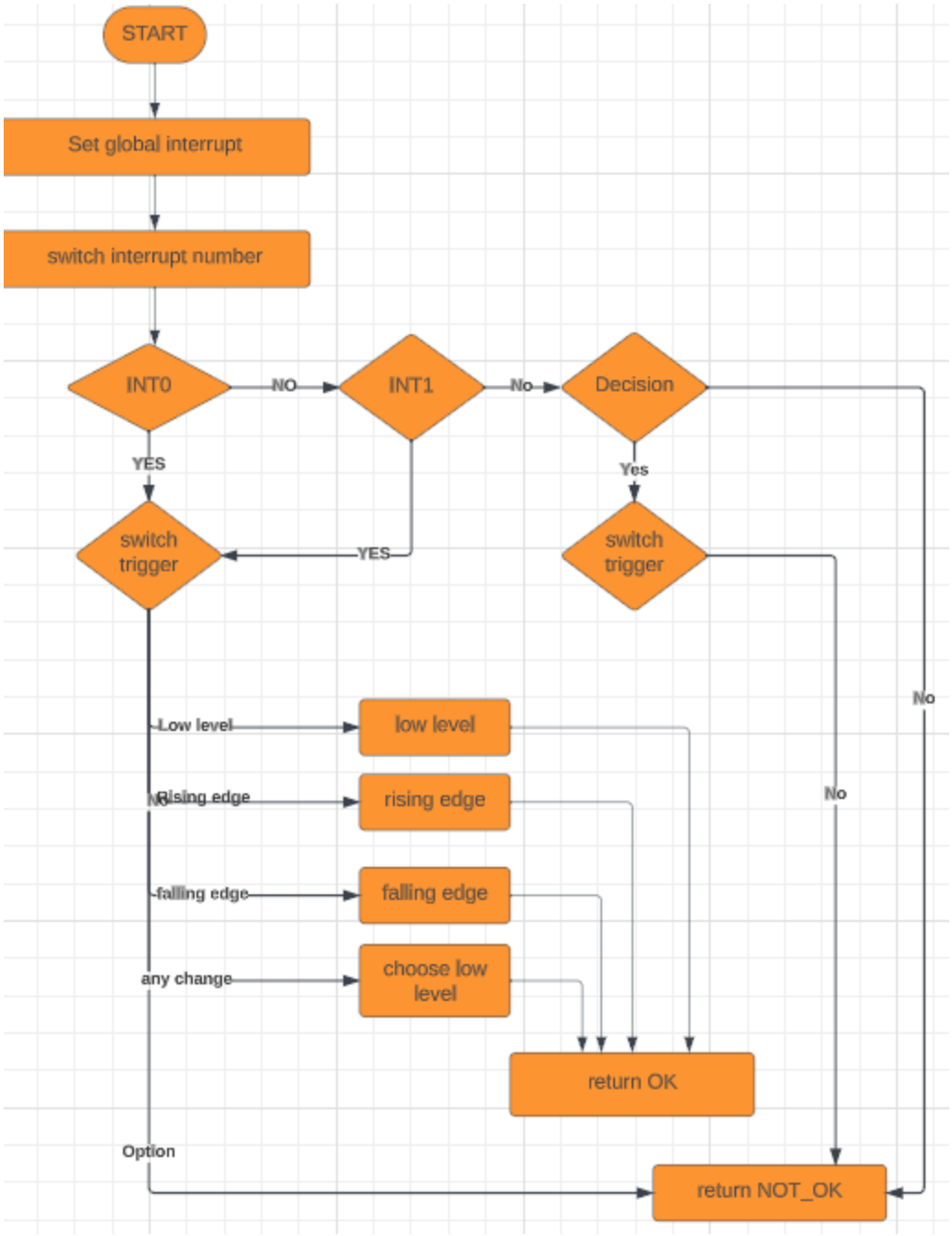


External Interrupt:

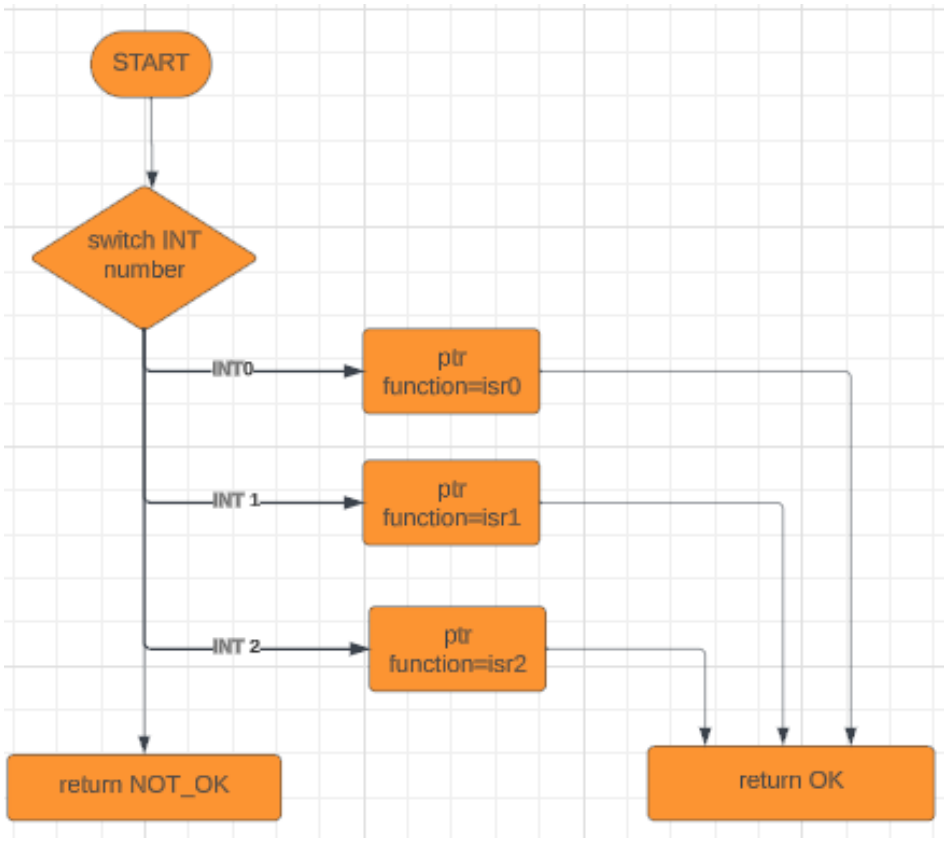
EN_EXTINT_ERROR SET_GLOBAL_INTERRUPT(EN_GLOBAL_INT state)


START
Enable
NO
Disable
YES
YES
No
SET bit no.7 in status reg
SET bit no.7 in status reg
Return OK
Return NOT_OK

EN_EXTINT_ERROR EXTINT_init(EN_EXINT_NUMBER INTx ,EN_Sense_Control INTxSense)

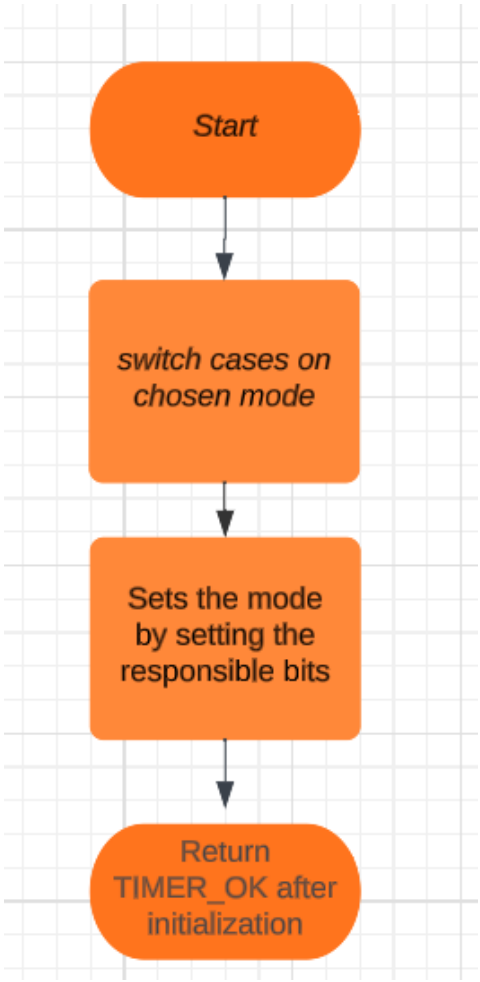


EN_EXTINT_ERROR EXTINT_CallBack(EN_EXINT_NUMBER INTx,void(*ptrfunc)(void))

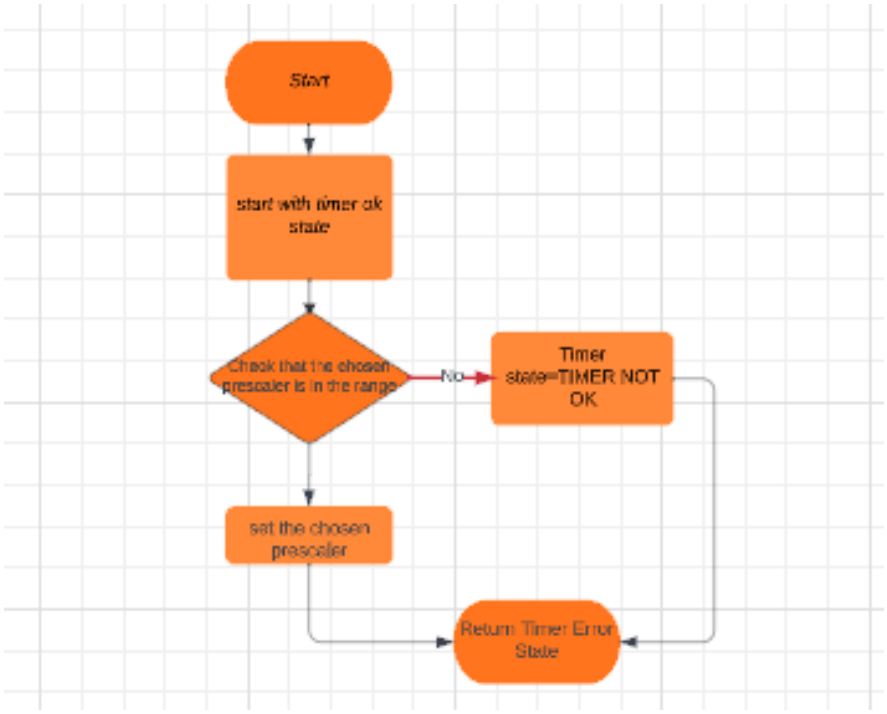


Timer:

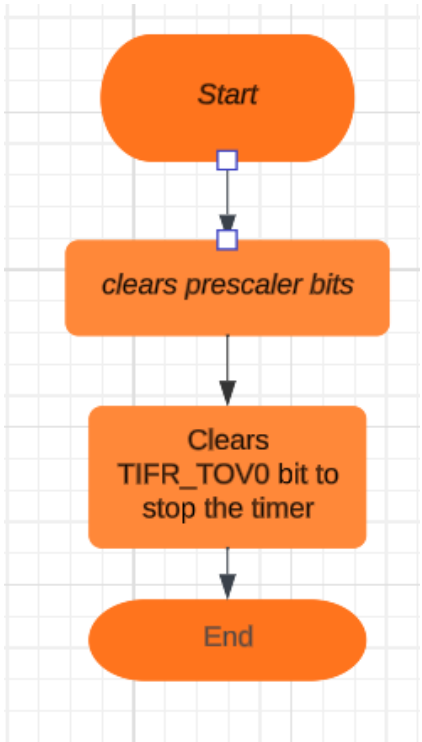
en_timer0ErrorState_t MTIMER0_init(en_timer0Mode_t u8_a_mode)



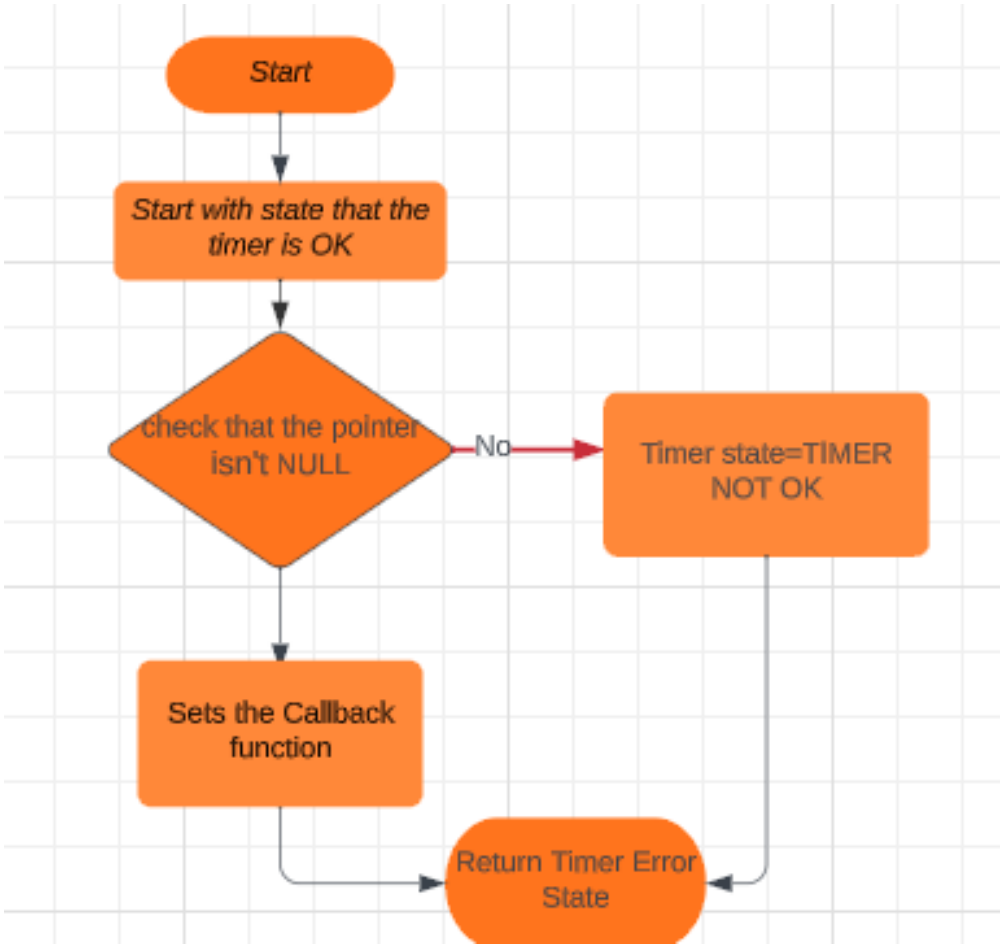
en_timer0ErrorState_t MTIMER0_startTimer(en_TIMER0_CLK_SELECT_t u8_a_prescaler)



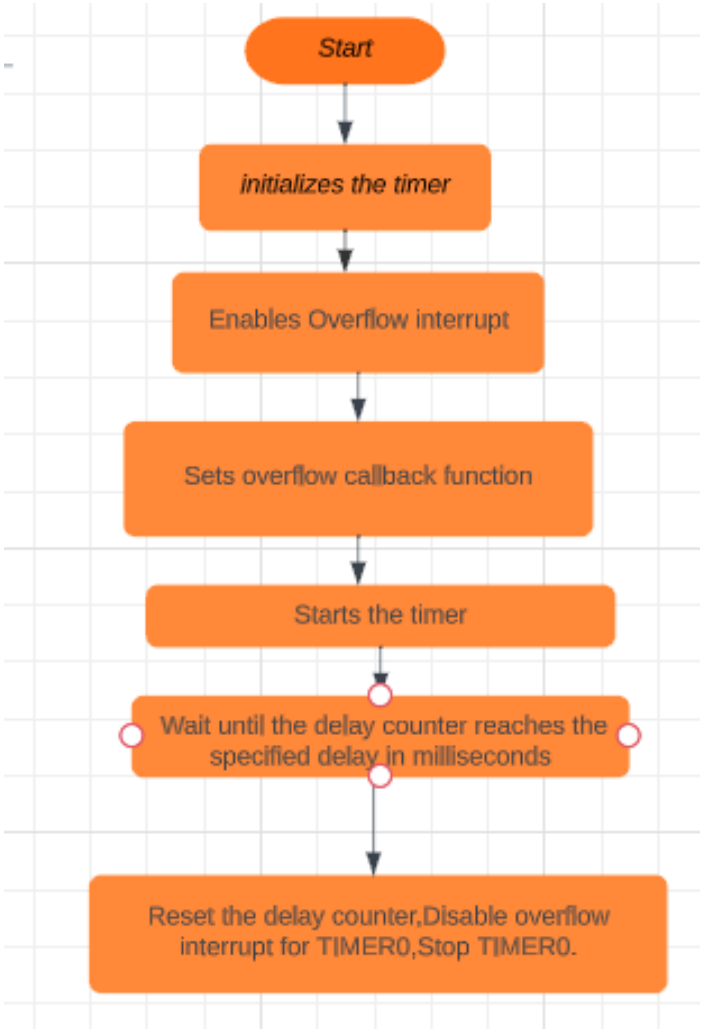
void MTIMER0_stop()



en_timer0ErrorState_t MTIMER0_setOVFCallback(void (*pv_a_CallbackFn)(void))

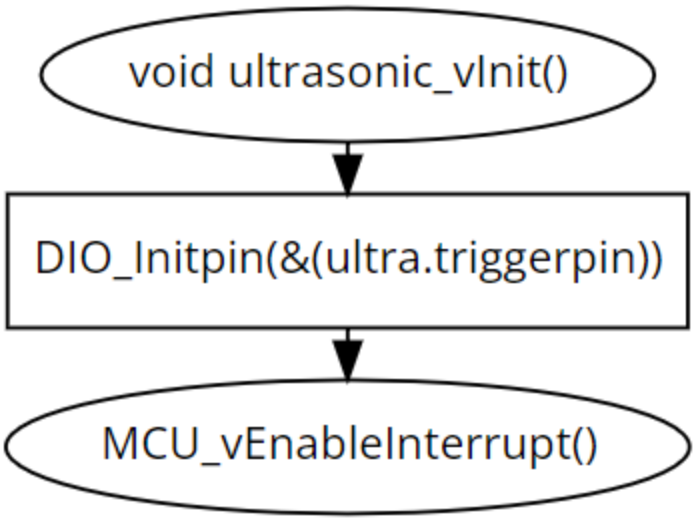


void delay_ms_UsingTimer(u16 u16_delay_ms)

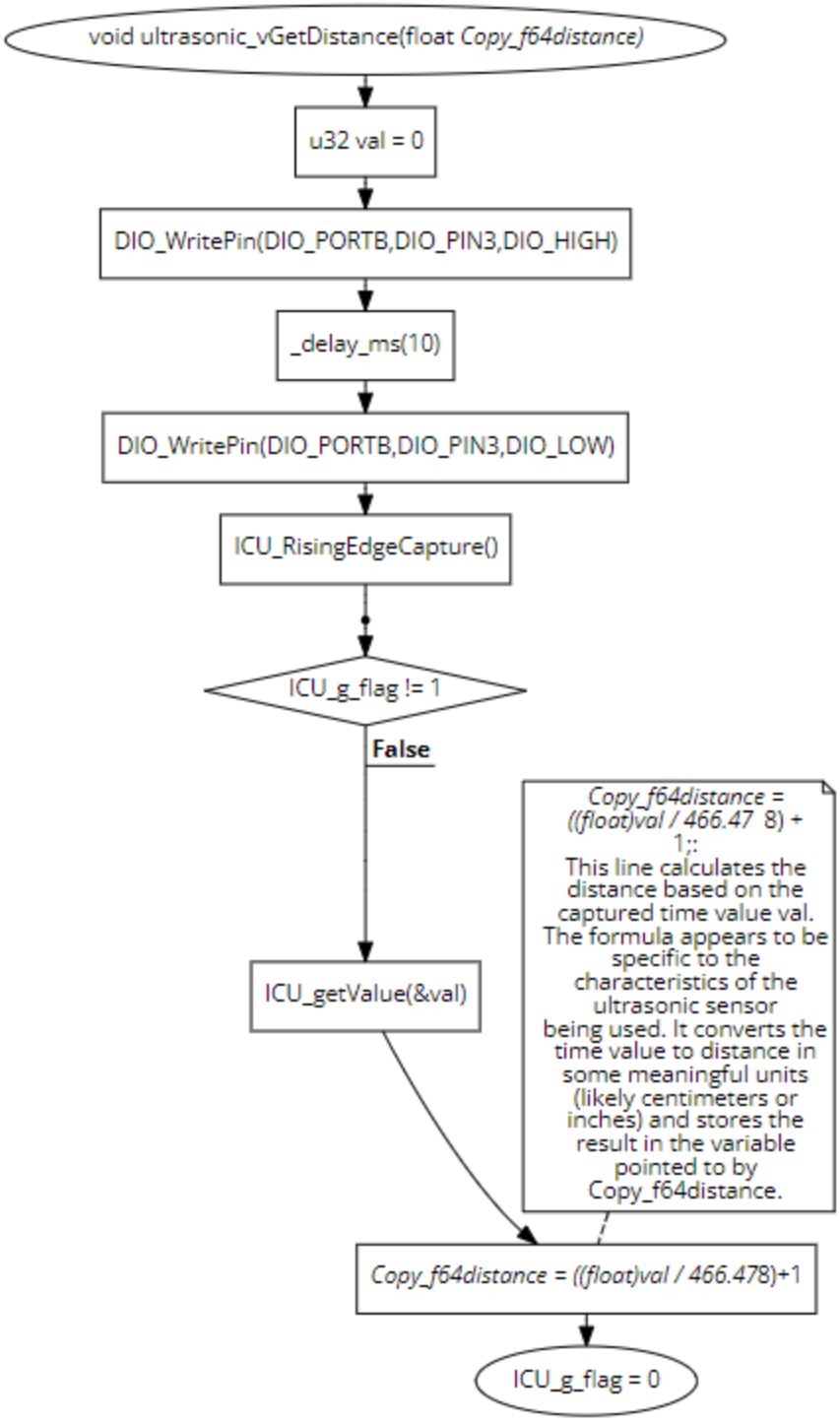


ULTRASONIC:

void ultrasonic_vInit()

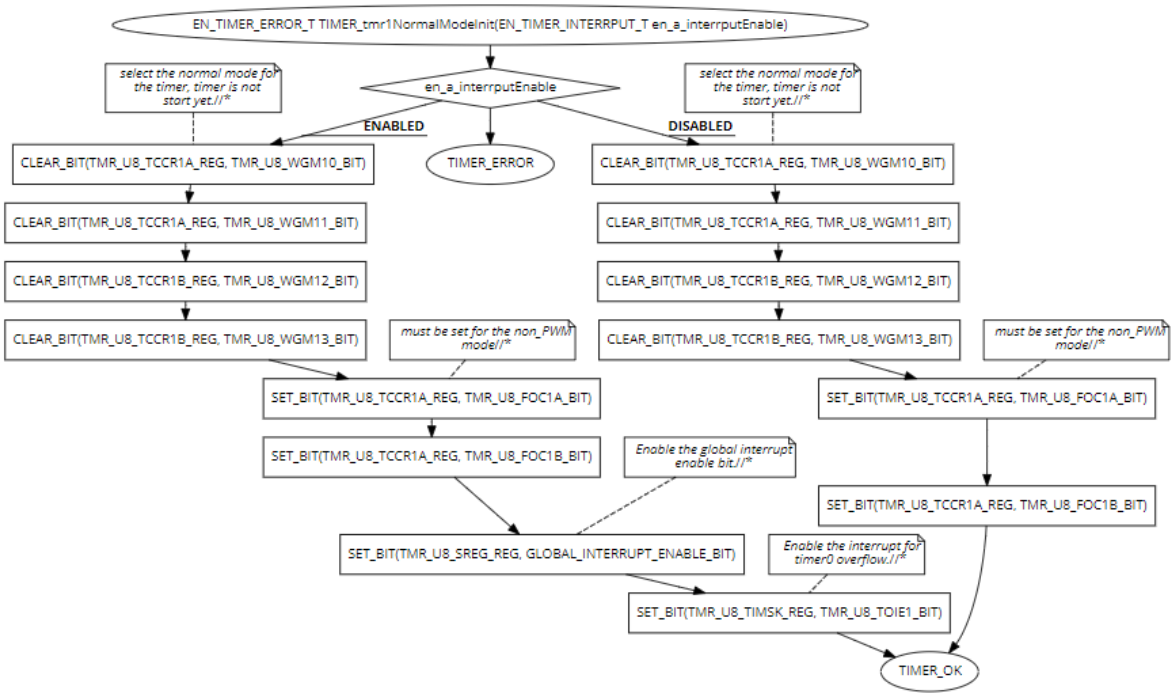


void ultrasonic_vGetDistance(float *Copy_f64distance)

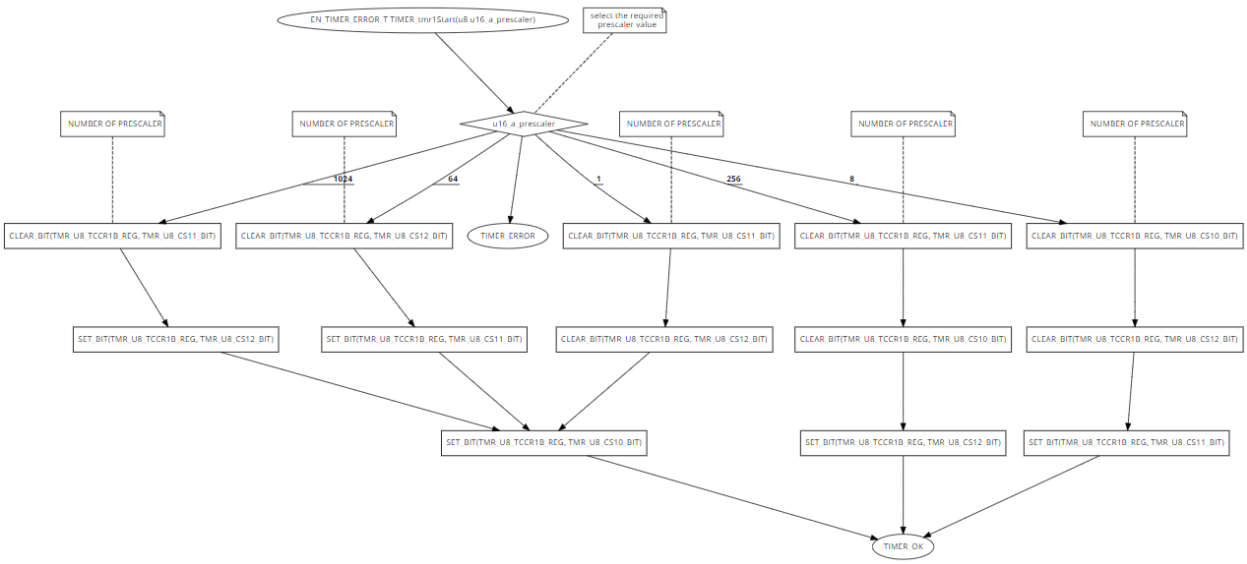


ICU:

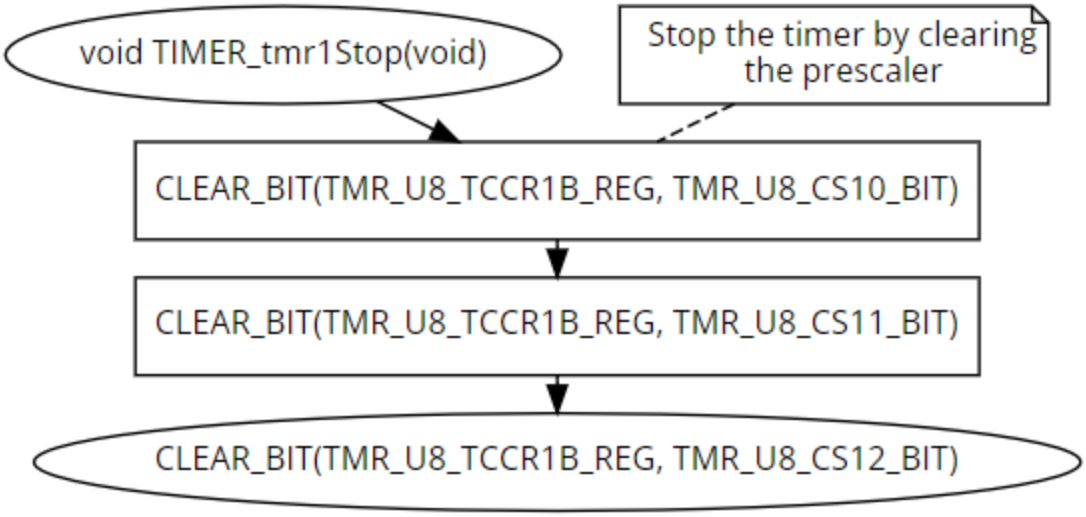
EN_TIMER_ERROR_T TIMER_tmr1NormalModeInit(EN_TIMER_INTERRPUT_T en_a_interrputEnable)



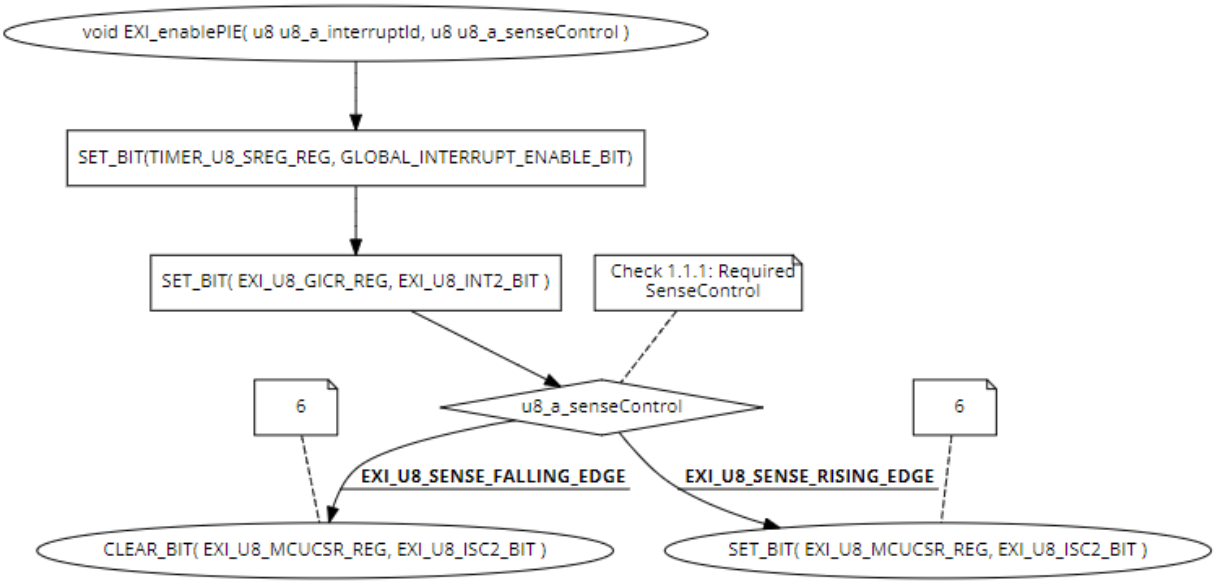
EN_TIMER_ERROR_T TIMER_tmr1Start(u8 u16_a_prescaler)



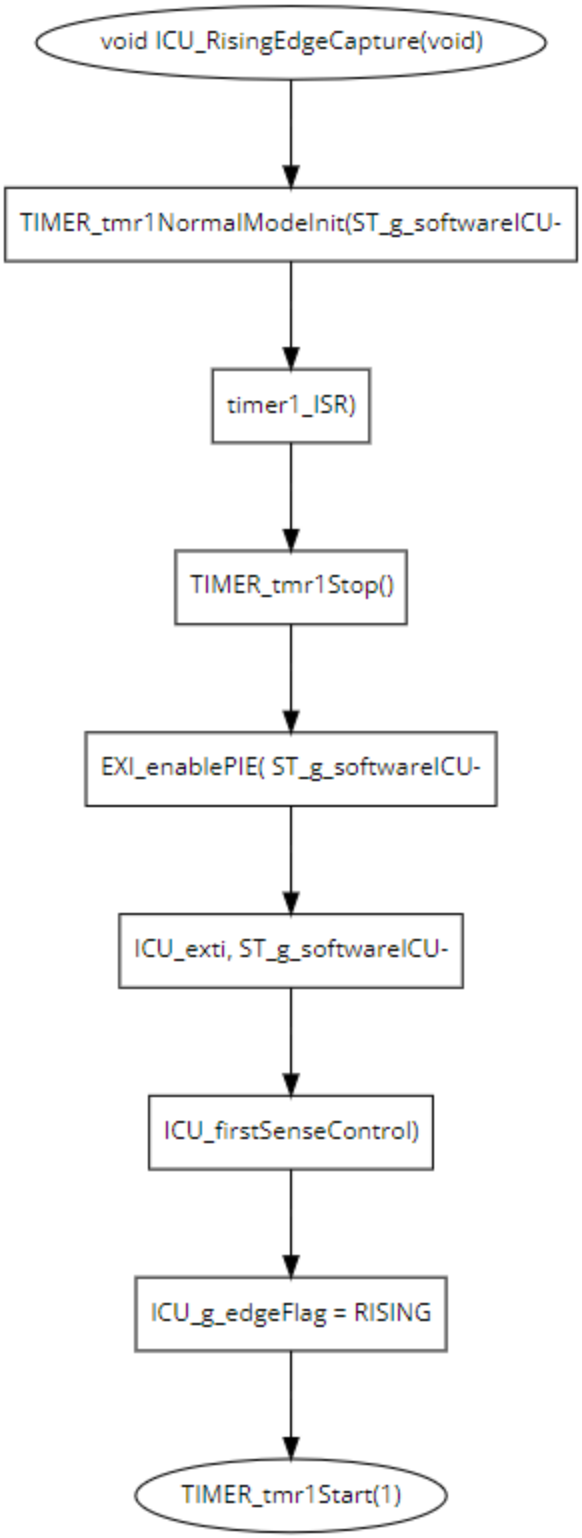
void TIMER_tmr1Stop(void)



void EXI_enablePIE(u8 u8_a_interruptId, u8 u8_a_senseControl)

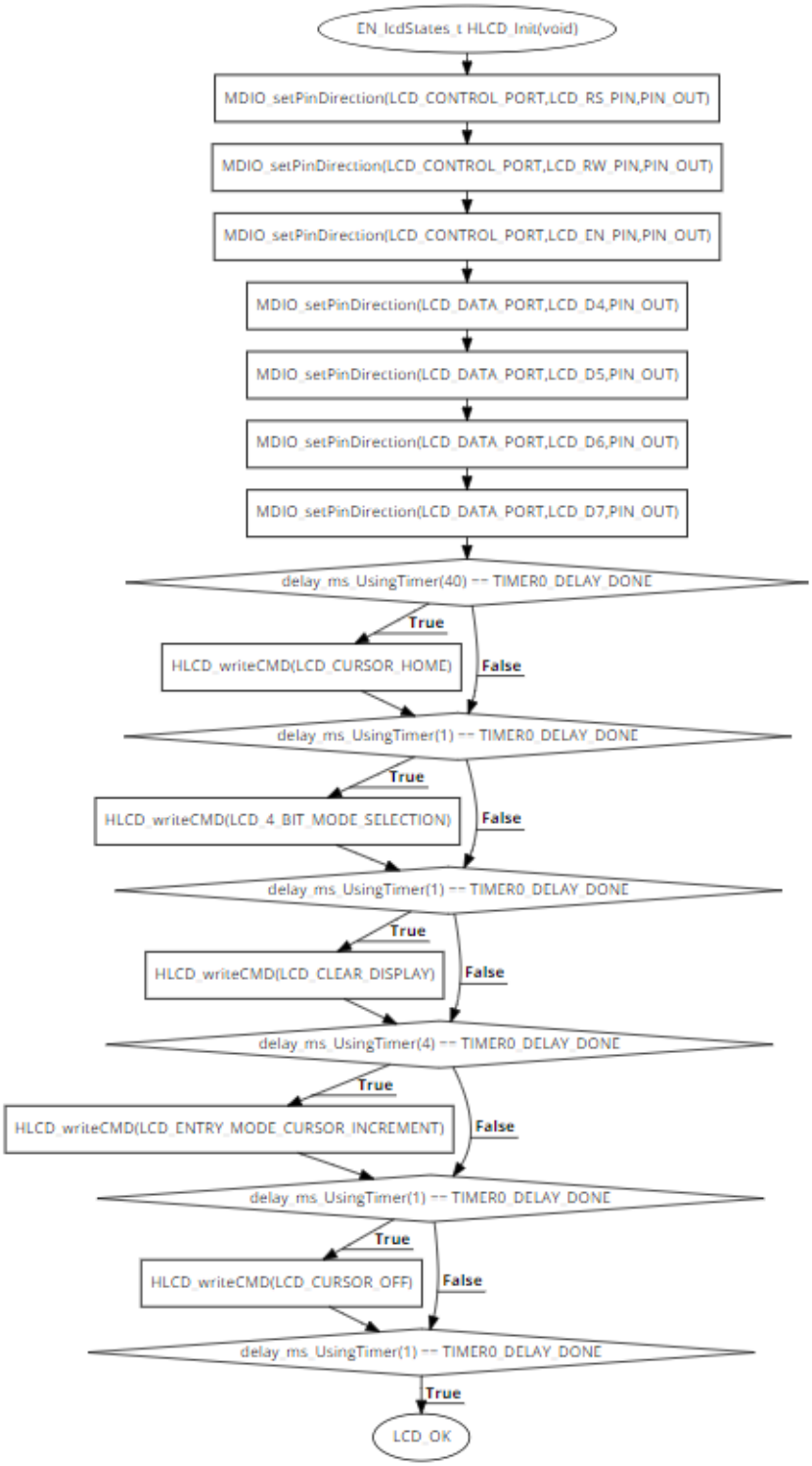


void ICU_RisingEdgeCapture(void)

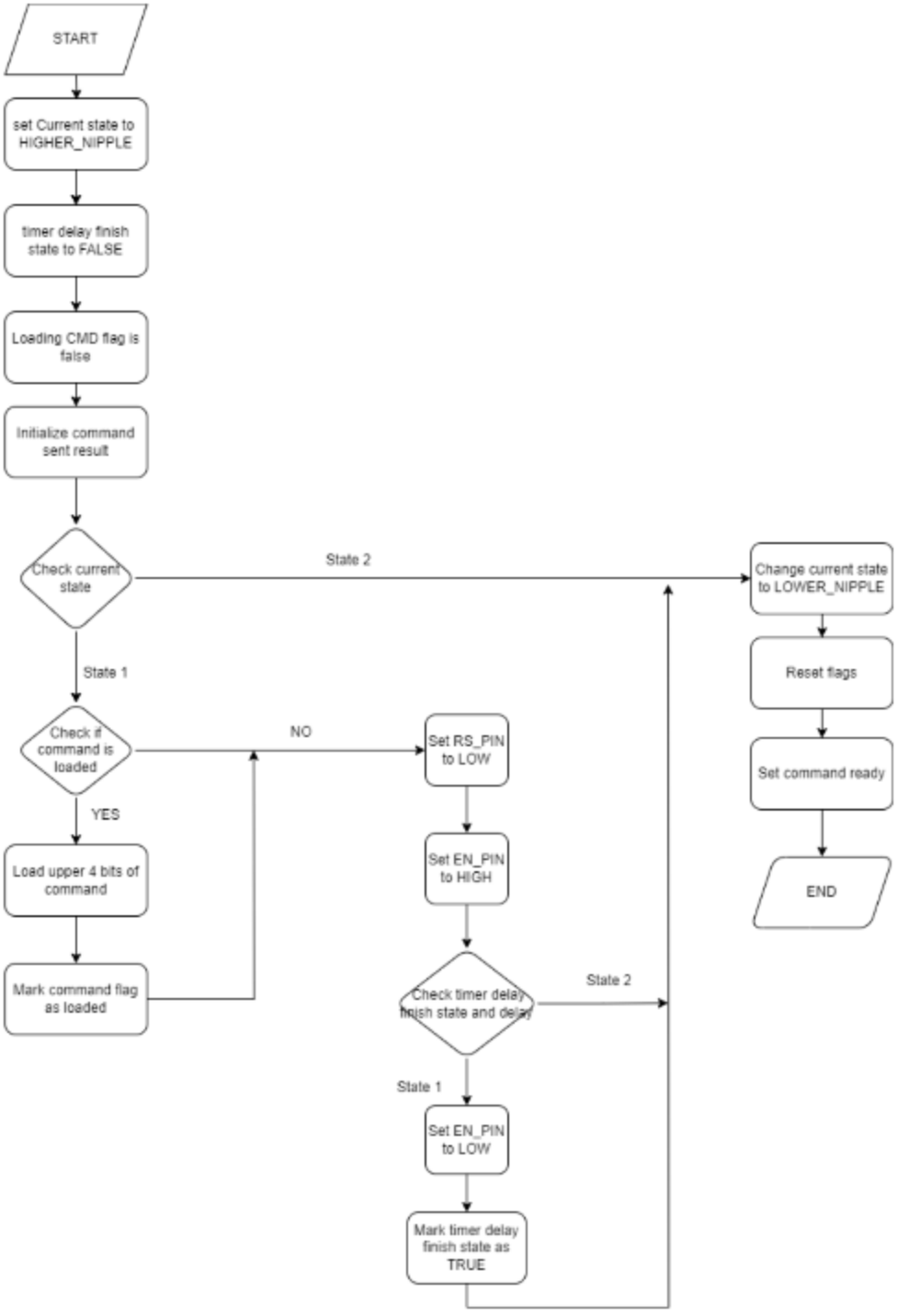


LCD:

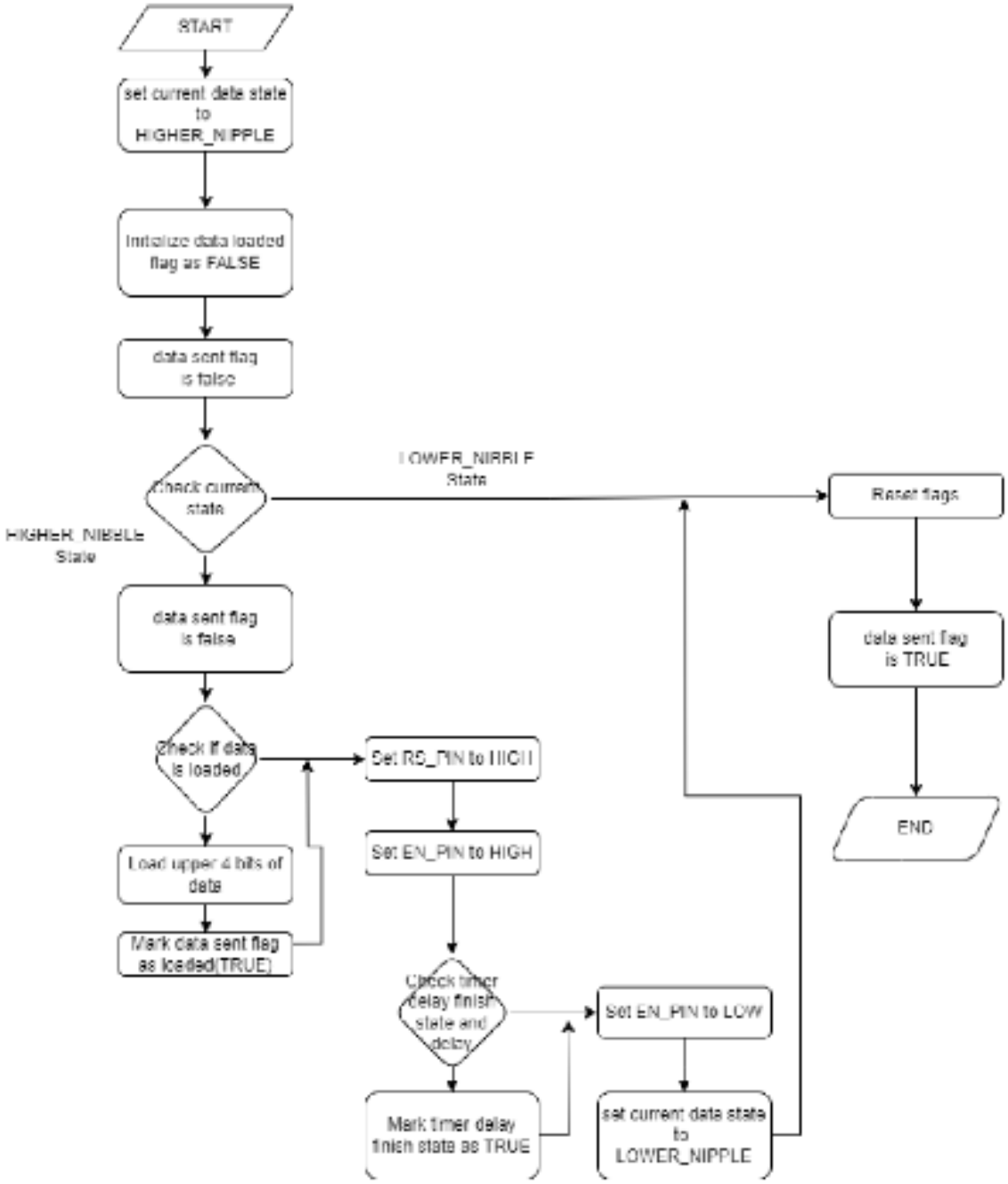
EN_lcdStates_t HLCD_Init(void)



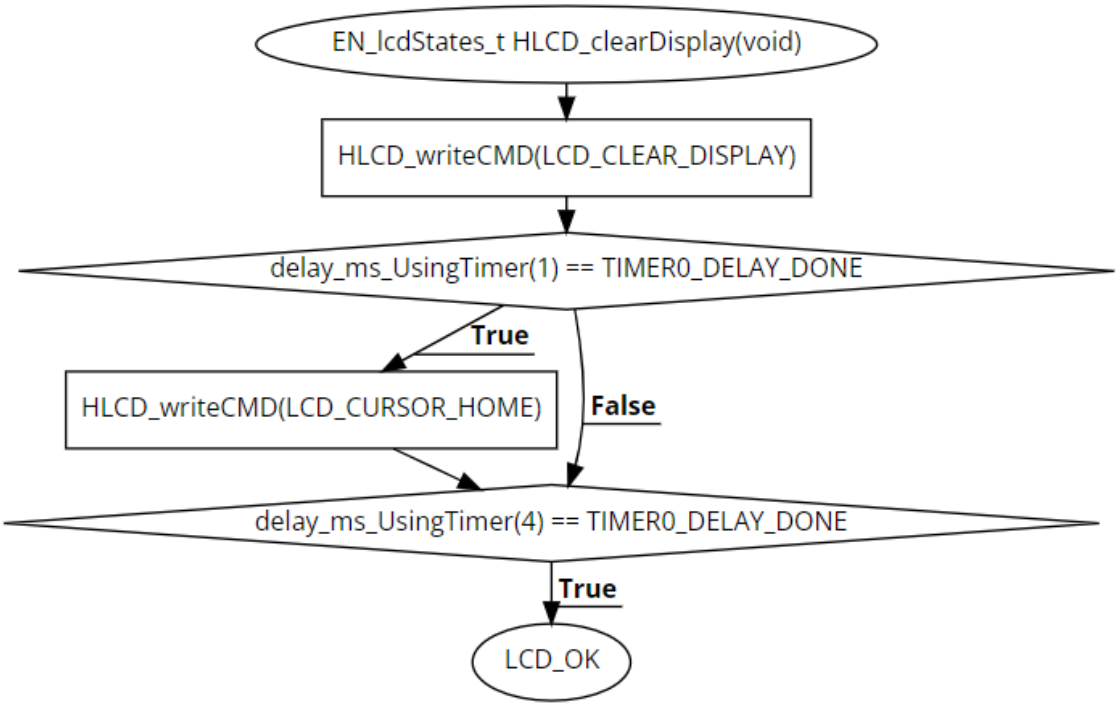
EN_lcdStates_t HLCD_writeCMD(u8 u8_a_command)



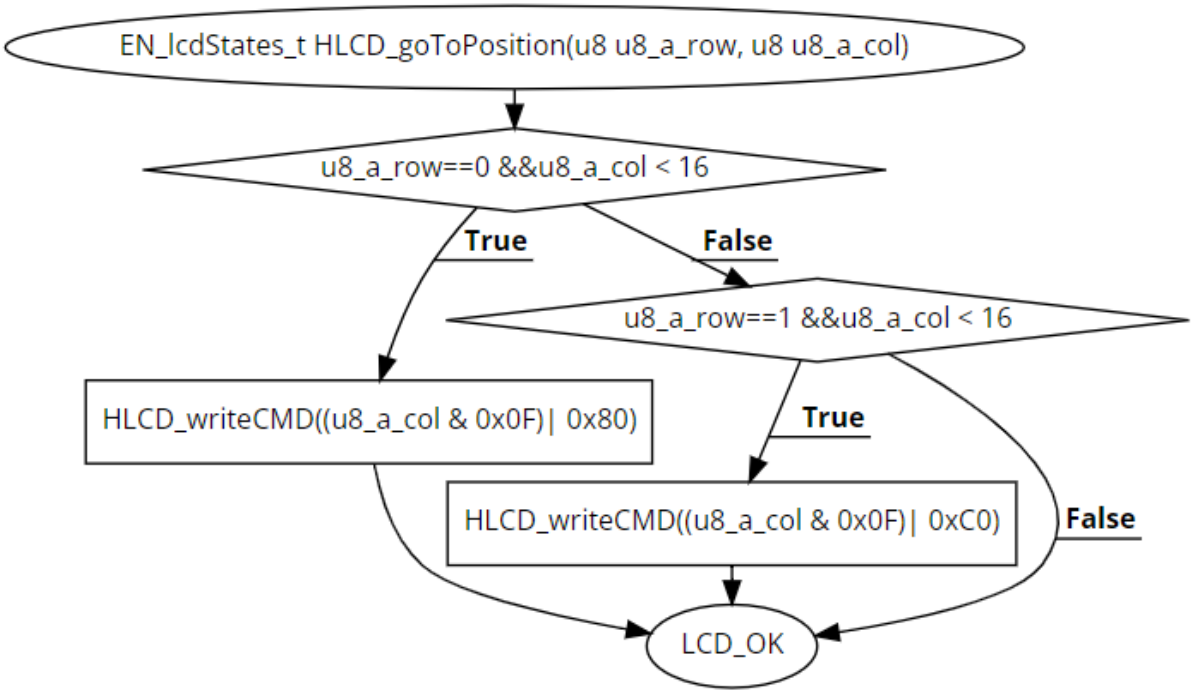
EN_lcdStates_t HLCD_writeChar(u8 u8_a_Char)



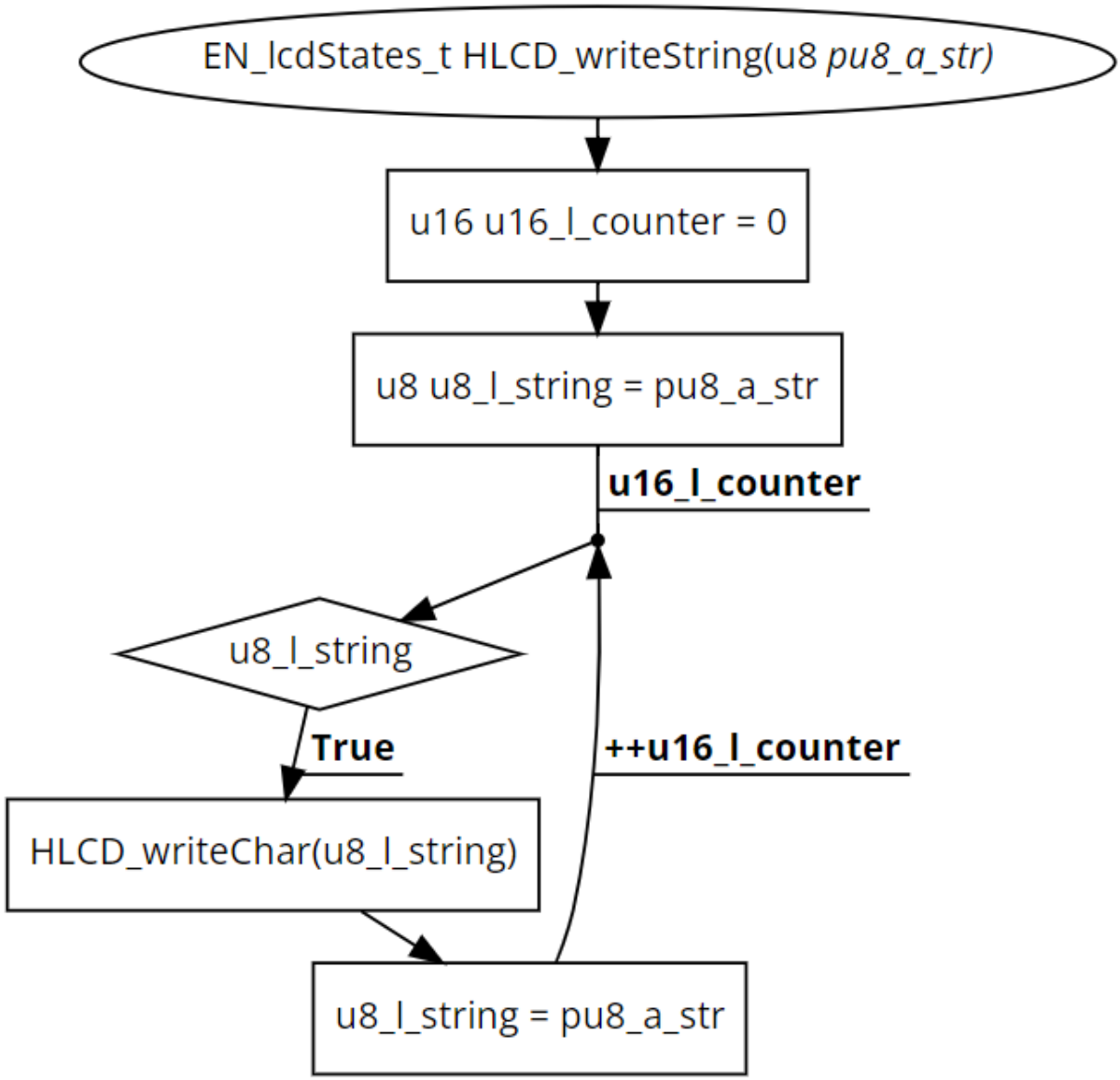
EN_lcdStates_t HLCD_clearDisplay(void)



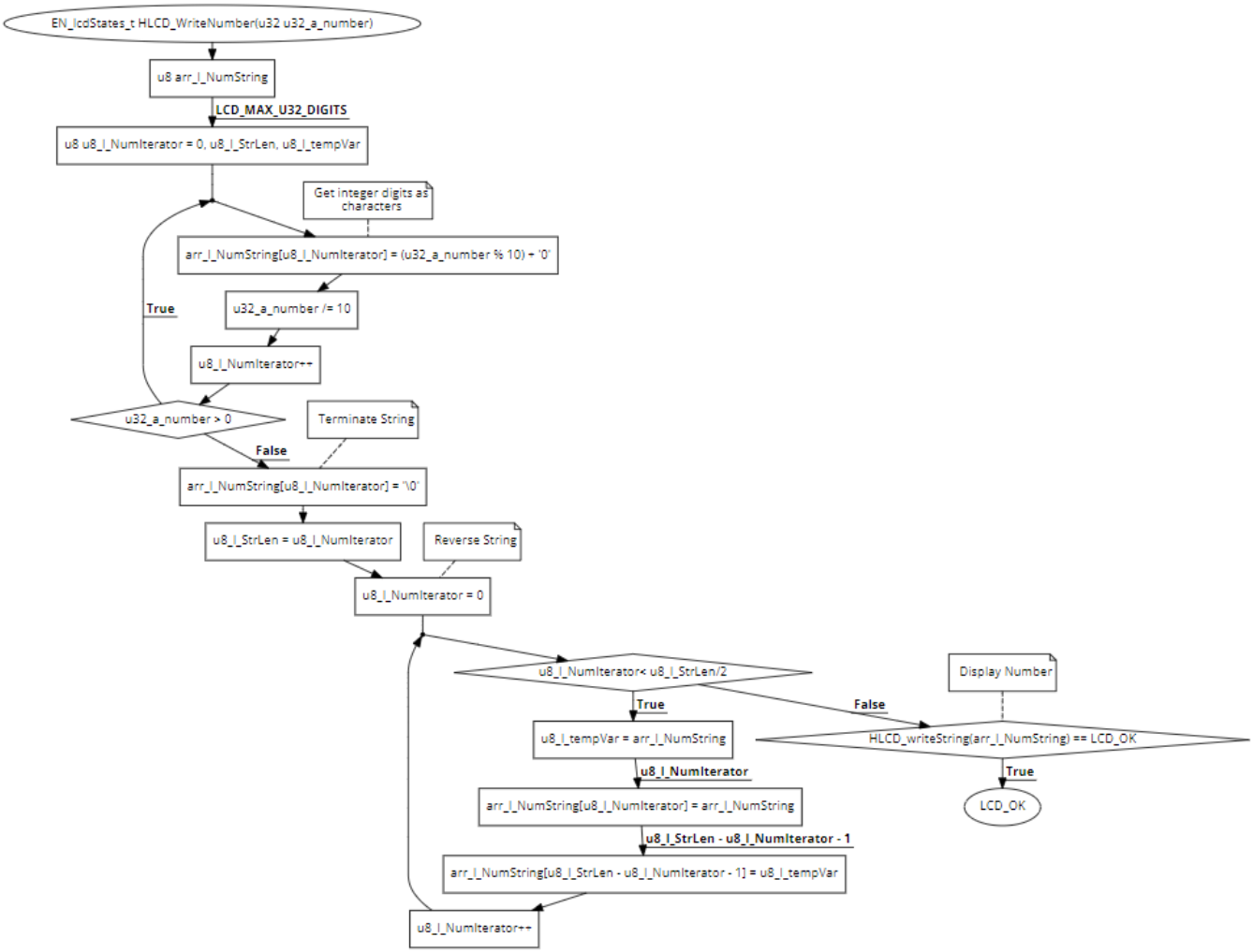
EN_lcdStates_t HLCD_goToPosition(u8 u8_a_row, u8 u8_a_col)



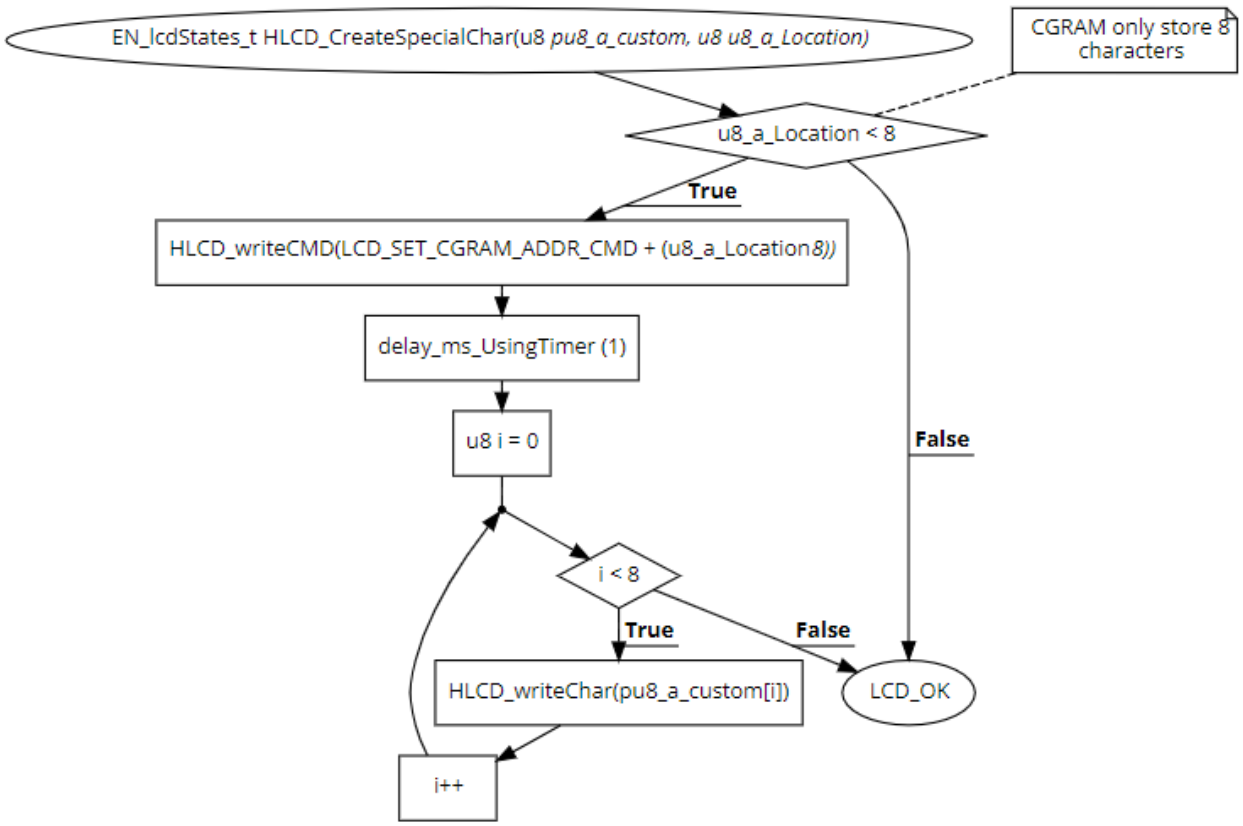
EN_lcdStates_t HLCD_writeString(u8* pu8_a_str)



EN_lcdStates_t HLCD_WriteNumber(u32 u32_a_number)



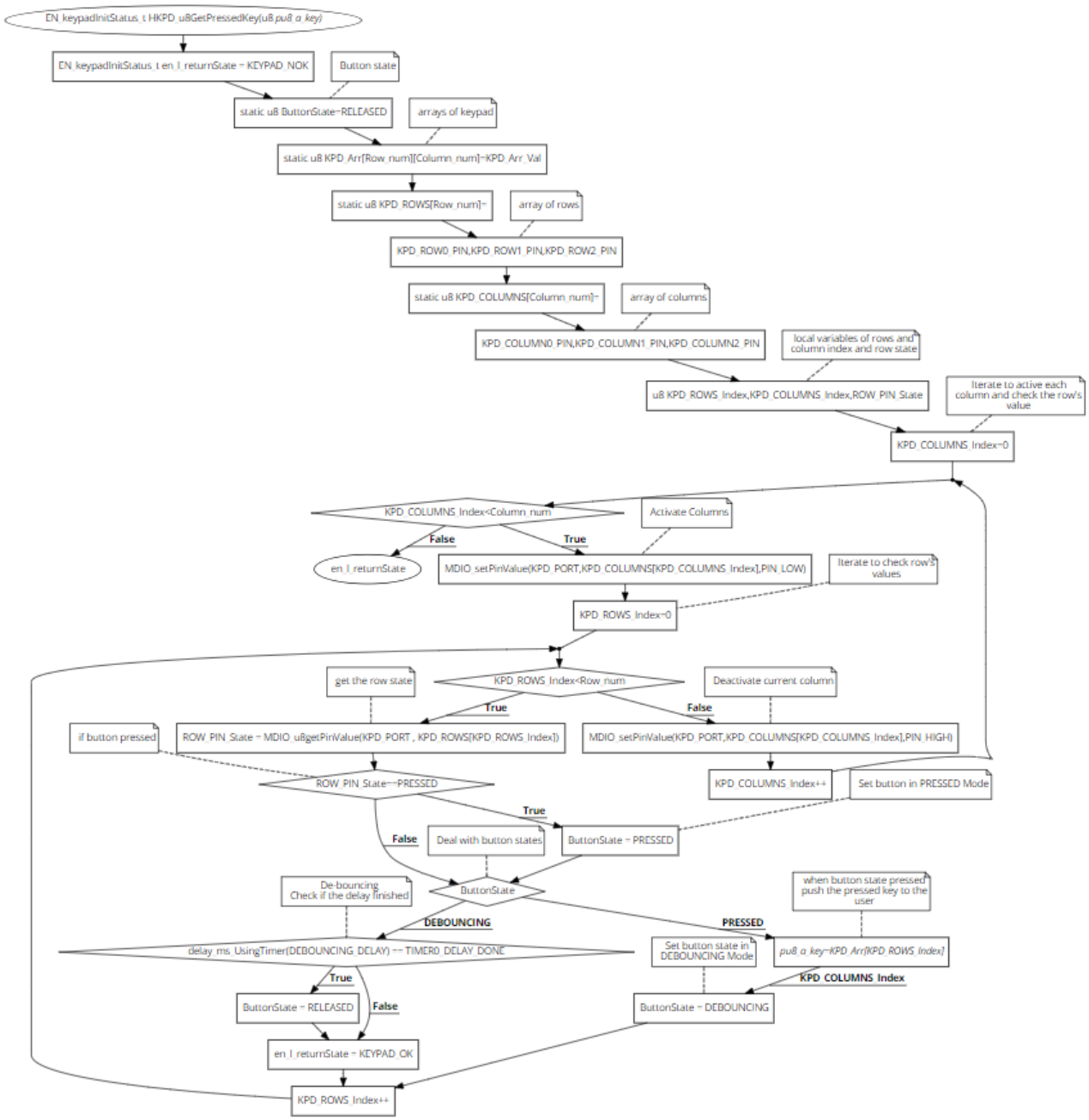
EN_lcdStates_t HLCD_CreateSpecialChar(u8* pu8_a_custom, u8 u8_a_Location)



KEYPAD:

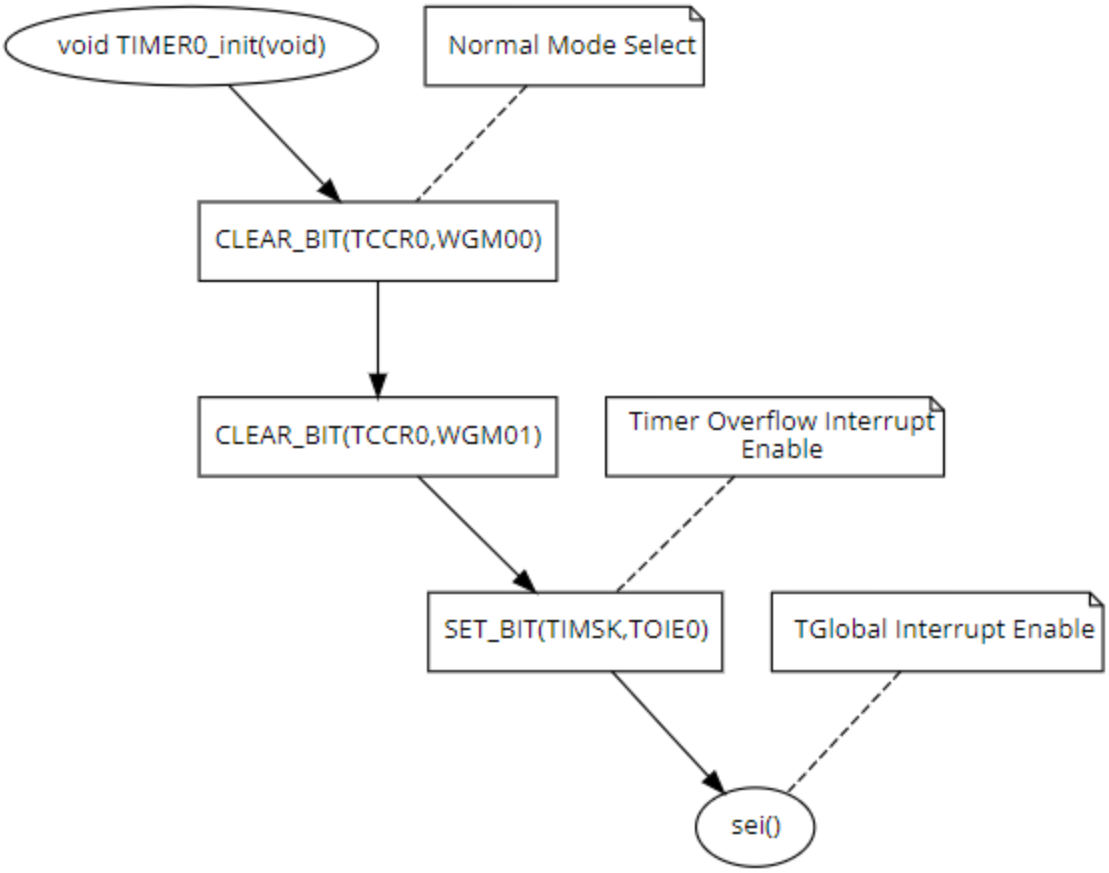
EN_keypadInitStatus_t HKPD_init(void)

EN_keypadInitStatus_t HKPD_u8GetPressedKey(u8* pu8_a_key)

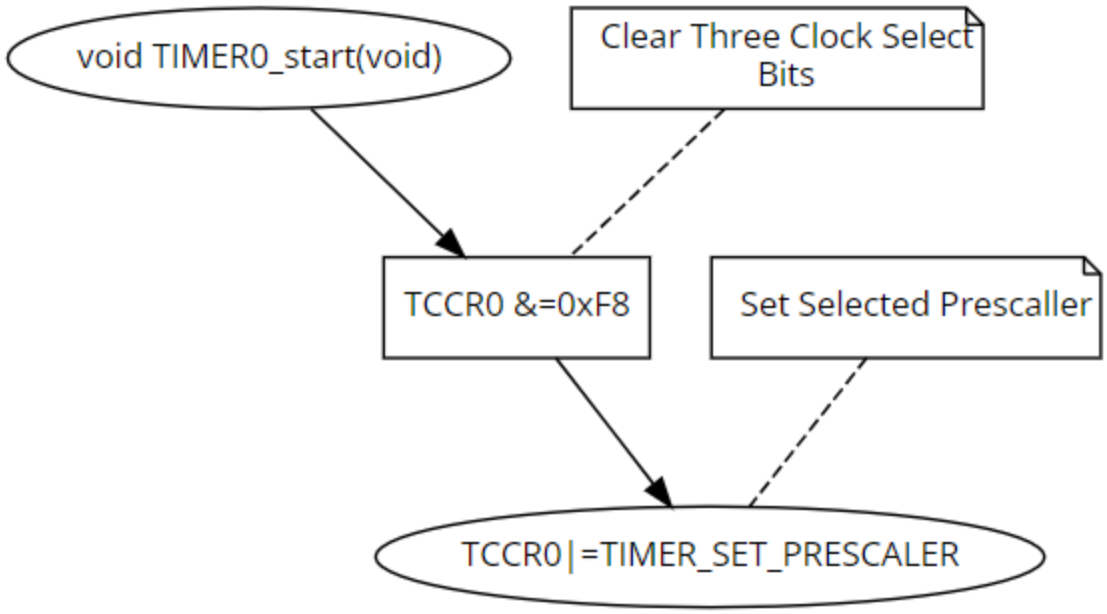


PWM:

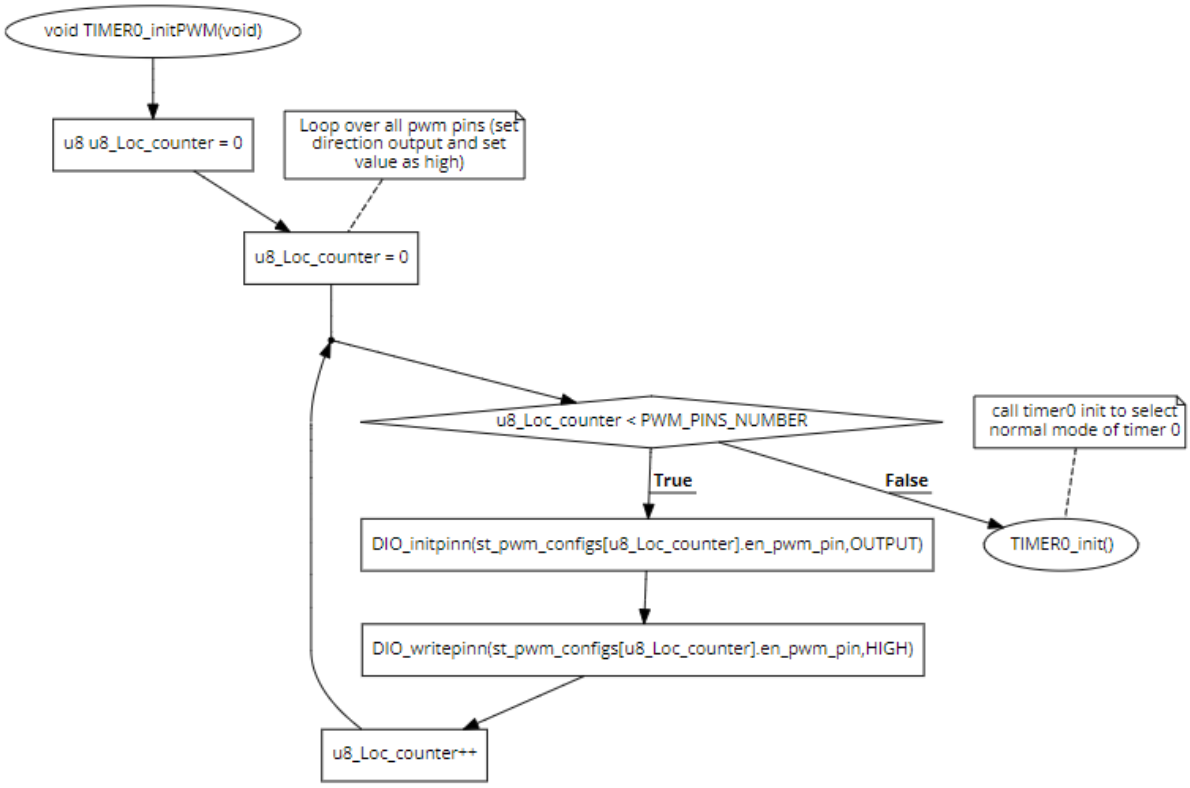
void TIMER0_init(void)



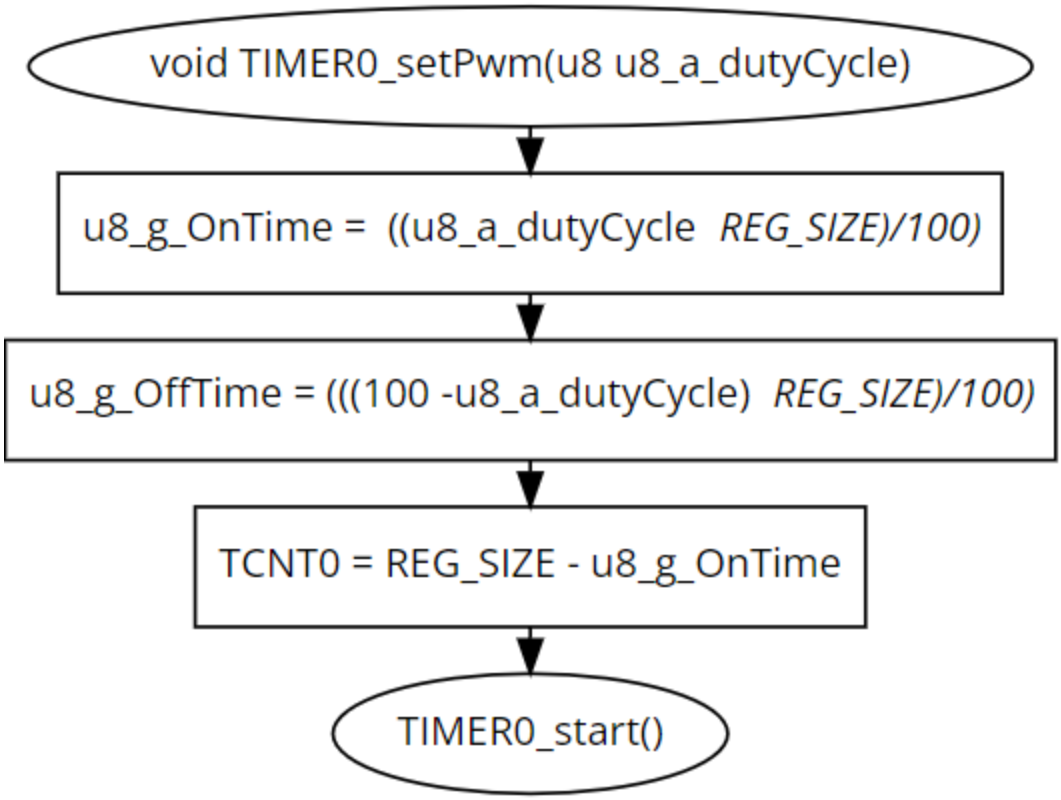
void TIMER0_start(void)



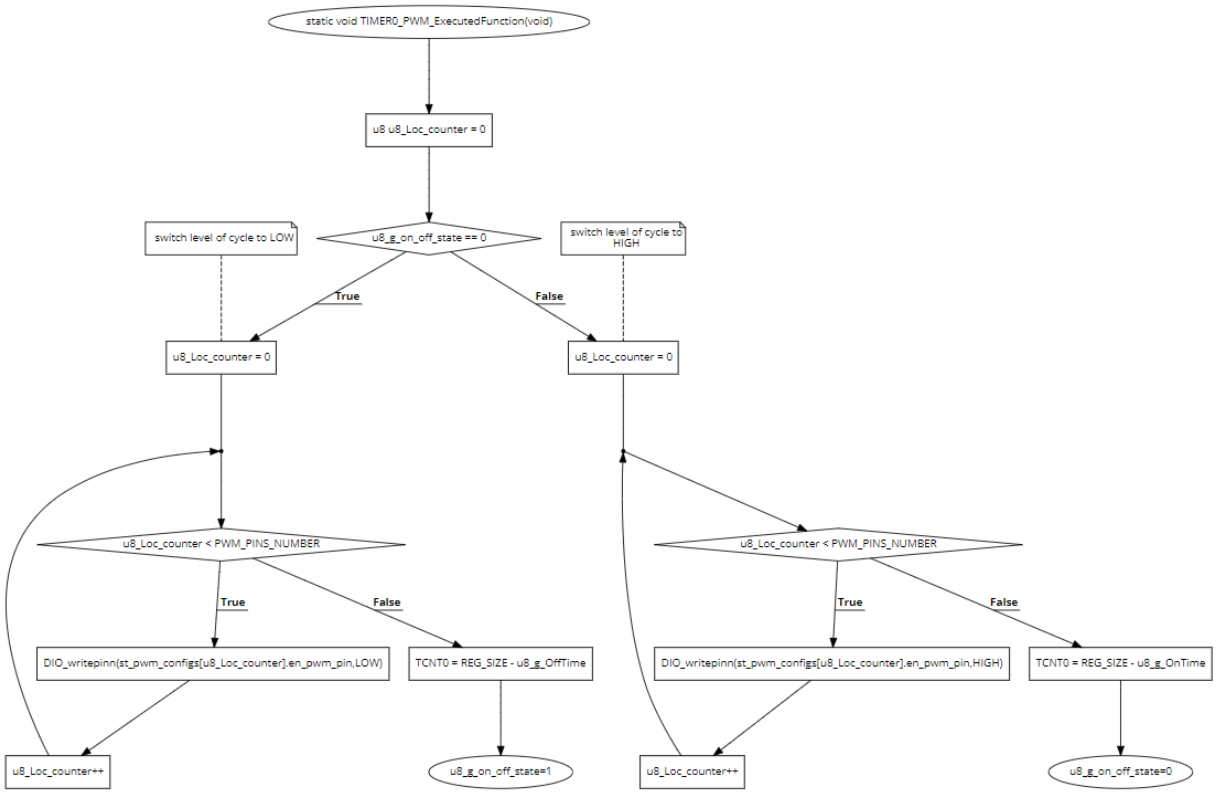
void TIMER0_initPWM(void)



void TIMER0_setPwm(u8 u8_a_dutyCycle)



static void TIMER0_PWM_ExecutedFunction(void)



App State Machine:

Idle State (Initial State)

Robot starts with 0 speed.

Default rotation direction is right.

Start Requested State

Transition to this state when Keypad button 1 is pressed.

Display "Set Def. Rot." on line 1 of the LCD.

Display "Right" on line 2.

Wait for 5 seconds.

Change Rotation Direction State

If Keypad button 0 is pressed, toggle the default rotation direction and update the LCD line 2.

Transition back to the Start Requested State after changing the rotation direction.

Set Default Rotation State

After 5 seconds in the Start Requested State, set the default rotation direction.

Wait for 2 seconds.

Moving Forward State

If there are no obstacles (distance > 70 cm):

Move forward at 30% speed for 5 seconds.

If no obstacles, switch to "Moving Forward 2" state.

Moving Forward 2 State

Continue moving forward at 50% speed as long as no obstacles are detected.

Update LCD line 1 with speed and direction information.

Obstacle Detected State

Depending on obstacle distance (20-70 cm), stop and take appropriate action (rotate or move backward).

Update LCD line 2 with distance information.

Transition back to "Moving Forward" or "Moving Forward 2" state if the obstacle is cleared.

360-Degree Rotation State (Bonus)

If the robot rotates 360 degrees without finding a distance greater than 20 cm, stop and update LCD data.

Check for Obstacle Removal State (Bonus)

Every 3 seconds, check if any obstacles were removed.

Move in the direction of the furthest object if necessary.

App Flowchart: