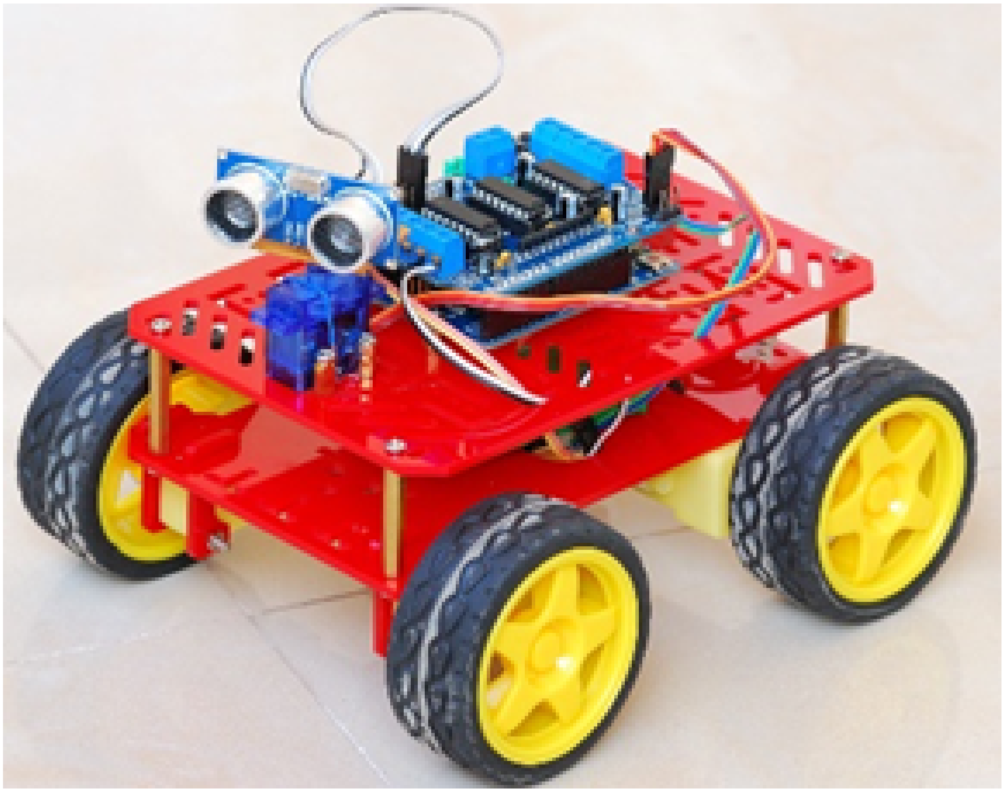


Design Obstacle Avoidance Car

Team no.2
Mario Saad
Nadeen Adel
Peter Essam

1. Introduction:

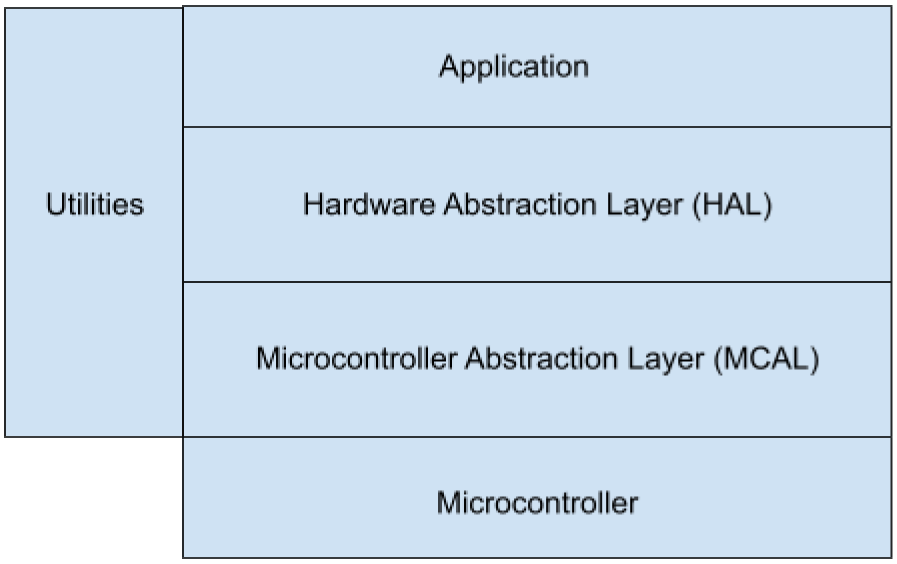
This project is a development of a robotic car system with a wide range of features and functionalities. It is built around the ATmega32 microcontroller and integrates components such as motors, a button, a keypad, an ultrasonic sensor, and an LCD display. The primary goal of this project is to create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

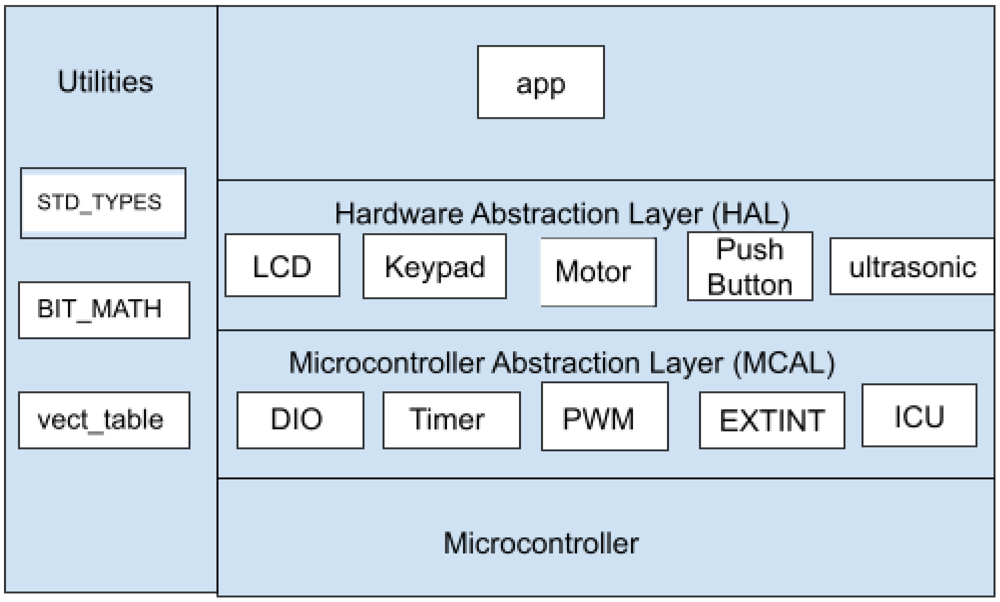
The system's core functionalities include managing the car's speed, direction, and obstacle detection. The car starts from a standstill and initially rotates to the right. The user can initiate its movement using a keypad, with options to start, stop, and change the default rotation direction. The robot 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.

1. High Level Design:

2.1 Layered Architecture:


Utilities
Application
Hardware Abstraction Layer (HAL)
Microcontroller Abstraction Layer (MCAL)
Microcontroller



2.2 Modules Description:

MCAL Layer:

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

HAL Layer:

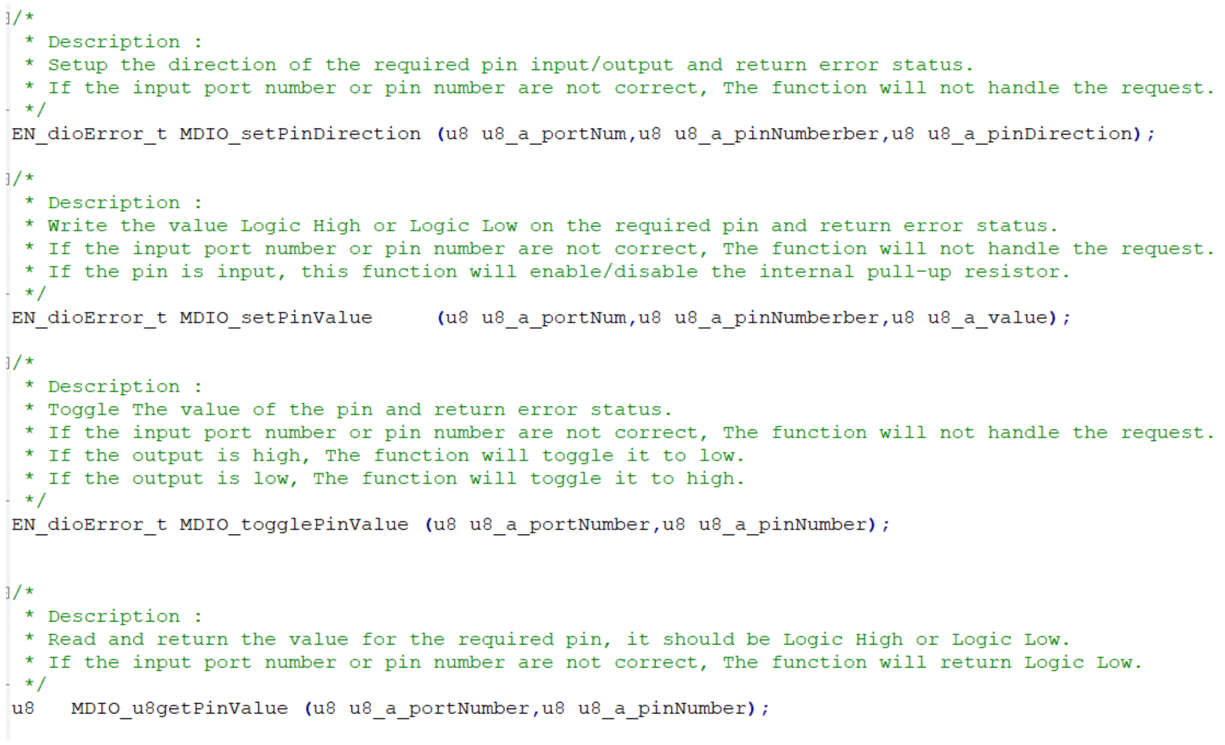
- Button: dealing with the rotation button .
- Keypad : dealing with buttons (Start and Stop buttons)
- LCD: display state of the car.
- Motor: controls movement state of the car
- Ultrasonic: calculate distance using ICU.

Application Layer:

- main logic of the code

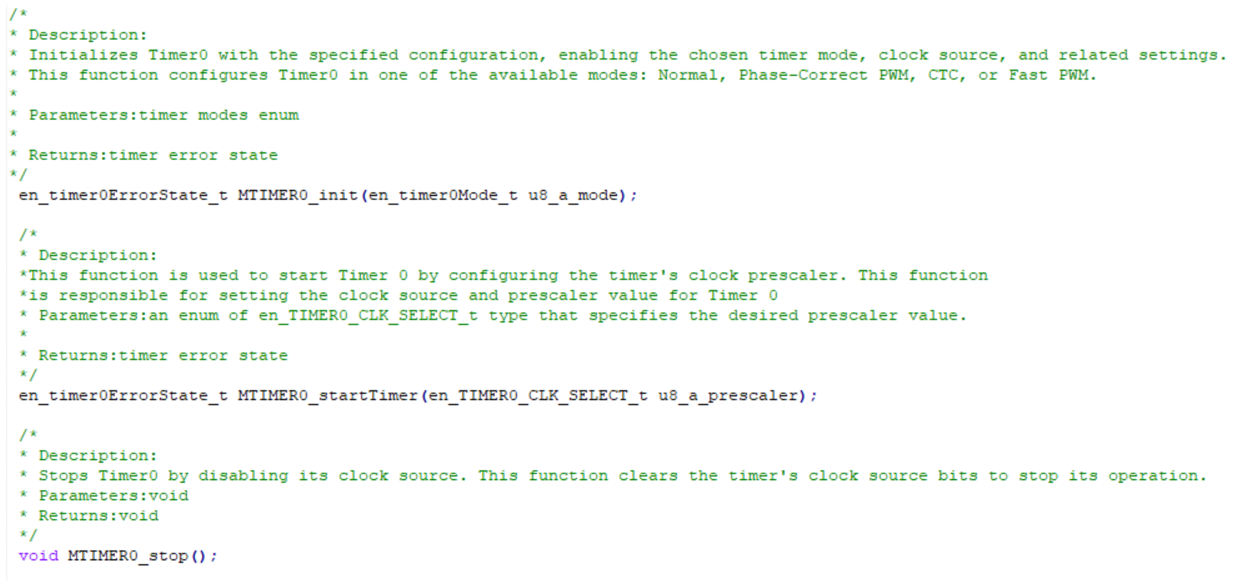
2.3 Drivers' Documentation:

2.3.1 DIO:

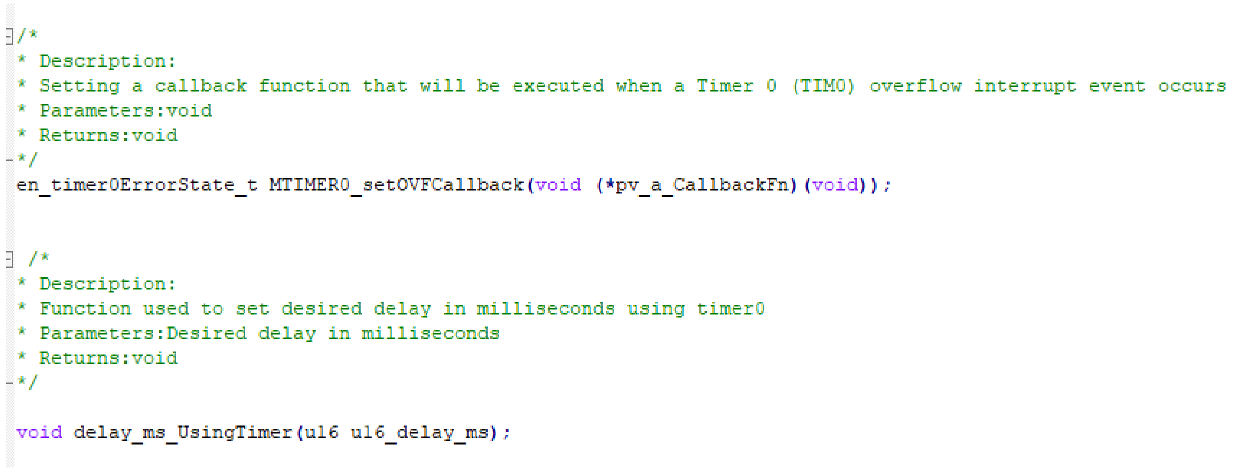


```
/*
 * Description :
 * Setup the direction of the required pin input/output and return error status.
 * If the input port number or pin number are not correct, The function will not handle the request.
 */
EN_dioError_t MDIO_setPinDirection (u8 u8_a_portNum,u8 u8_a_pinNumberber,u8 u8_a_pinDirection);

/*
 * 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:

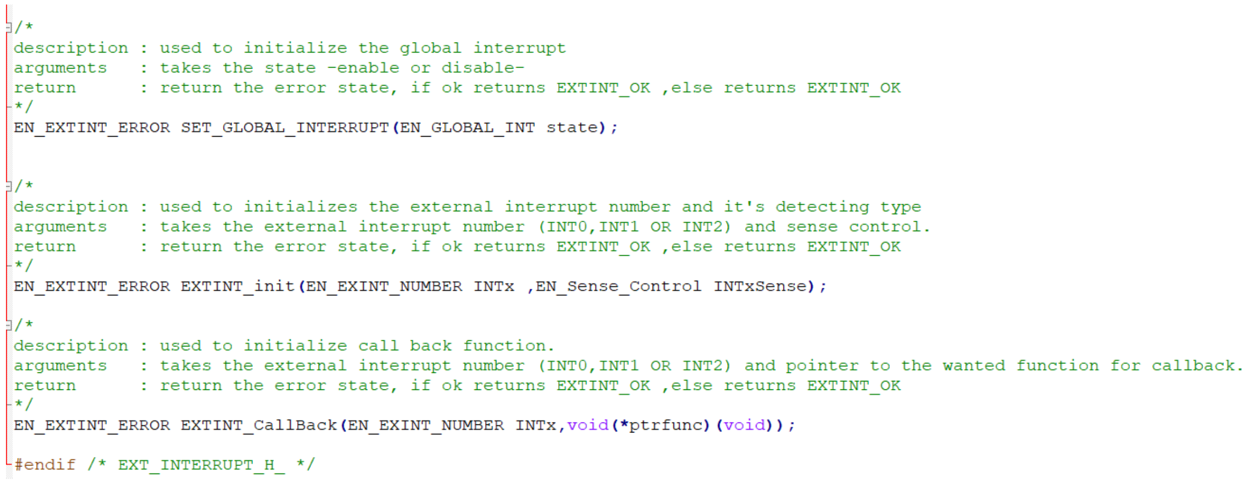


```
/*
* Description:
* Initializes Timer0 with the specified configuration, enabling the chosen timer mode, clock source, and related settings.
* This function configures Timer0 in one of the available modes: Normal, Phase-Correct PWM, CTC, or Fast PWM.
*
* Parameters:timer modes enum
*
* Returns:timer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



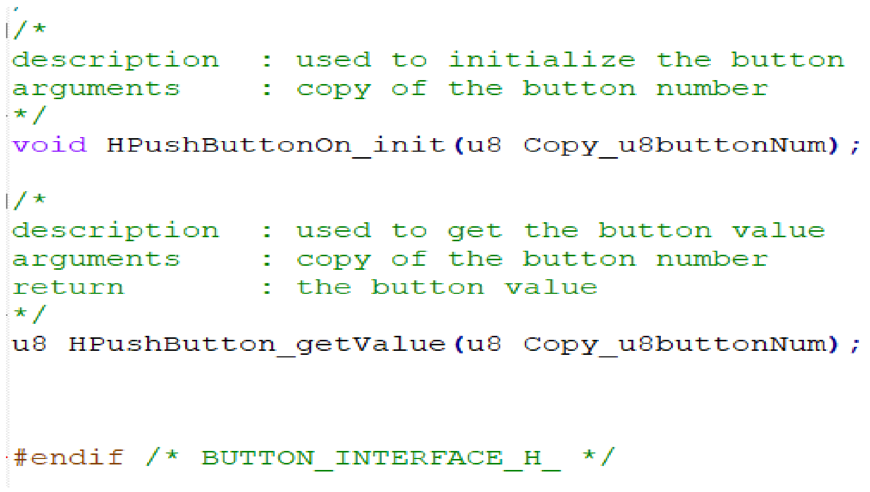
```
/*
 * Description:
 * Setting a callback function that will be executed when a Timer 0 (TIM0) overflow interrupt event occurs
 * Parameters:void
 * Returns:void
 */
en_timer0ErrorState_t MTIMER0_setOVFCallback(void (*pv_a_CallbackFn)(void));


 /*
 * 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:



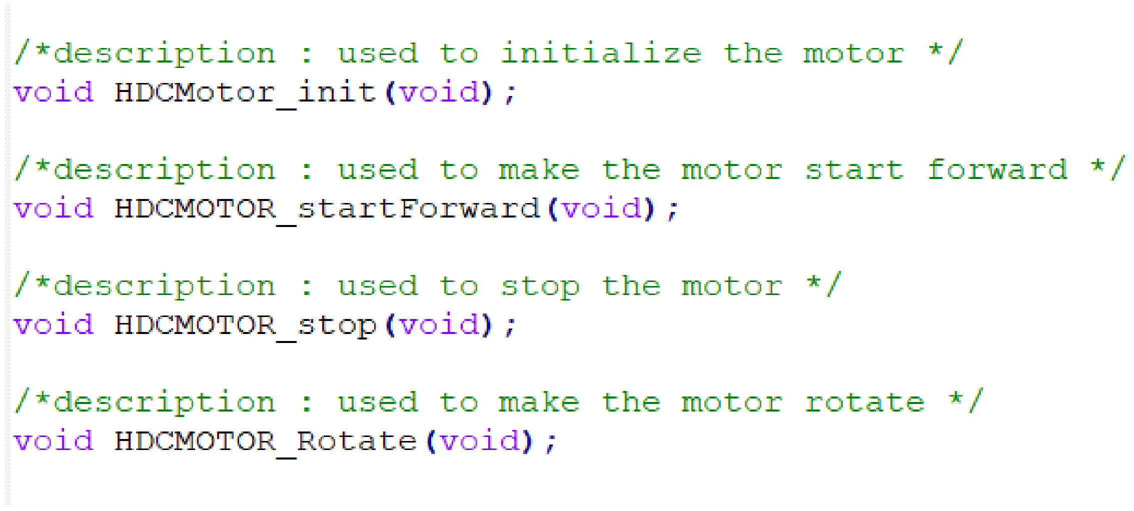
```
/*
description : used to initialize the global interrupt
arguments   : takes the state -enable or disable-
return      : return the error state, if ok returns EXTINT_OK ,else returns EXTINT_OK
*/
EN_EXTINT_ERROR SET_GLOBAL_INTERRUPT(EN_GLOBAL_INT state);


/*
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:



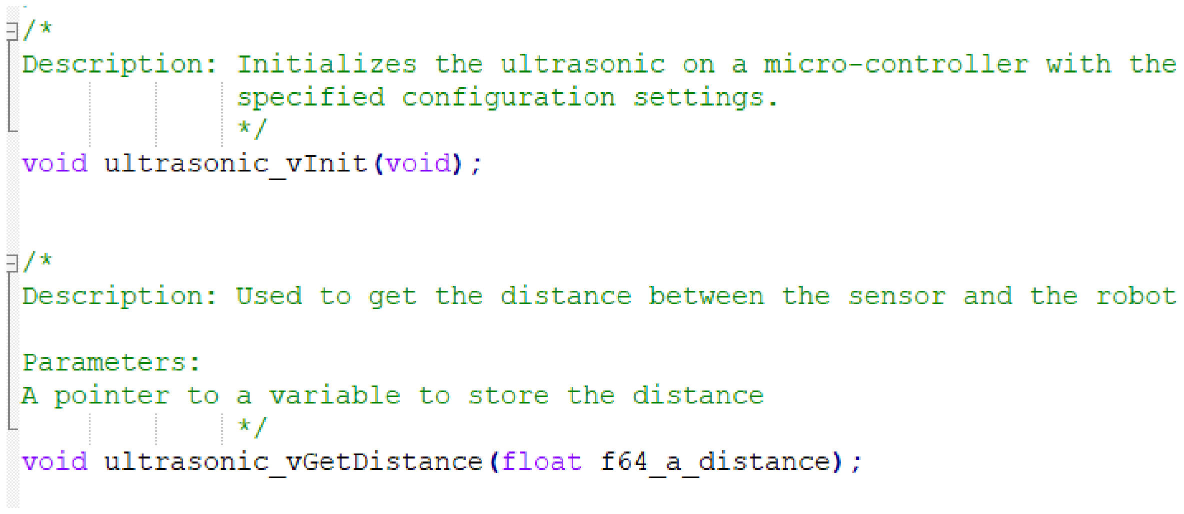
```
/*
description   : used to initialize the button
arguments     : copy of the button number
*/
void HPushButtonOn_init(u8 Copy_u8buttonNum);

/*
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:

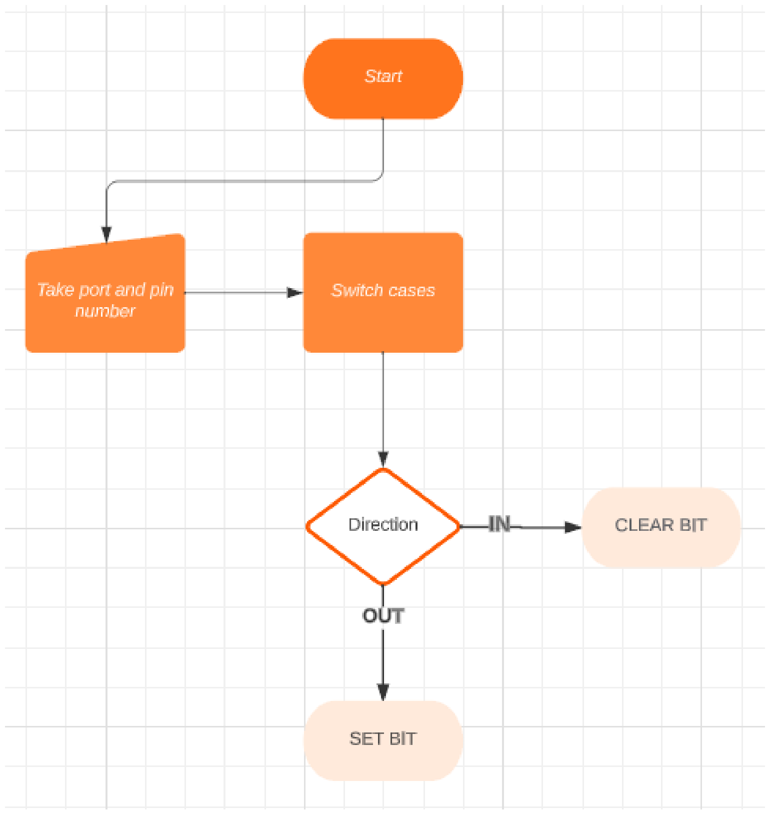


```
/*
Description: Initializes the ultrasonic on a micro-controller with the
                 specified configuration settings.
                 */
void ultrasonic_vInit(void);


/*
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);
```

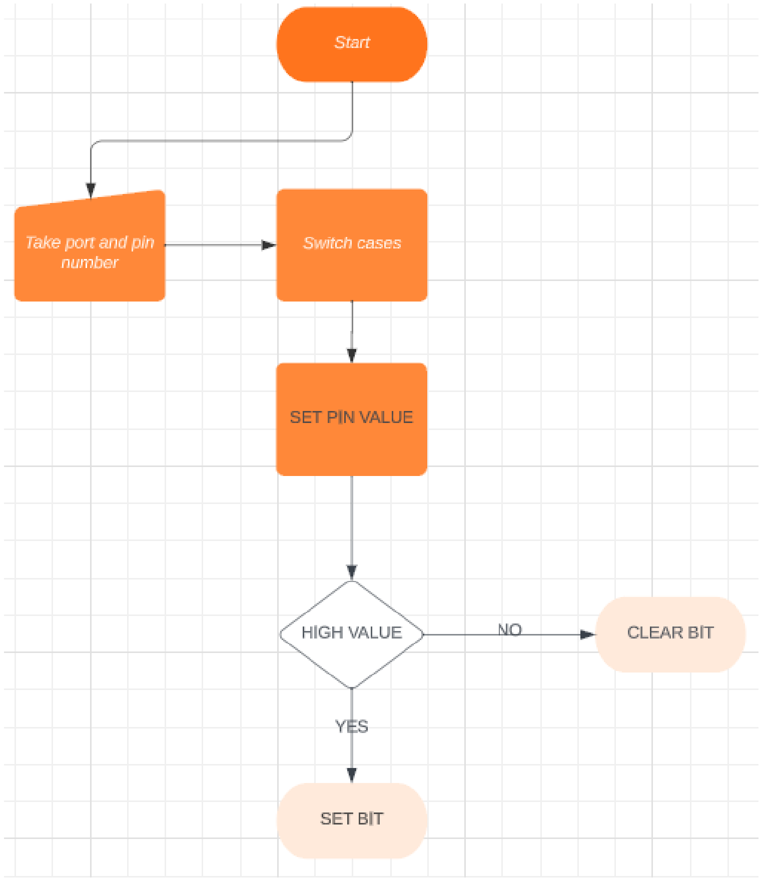
Functions Flowcharts:

DIO:

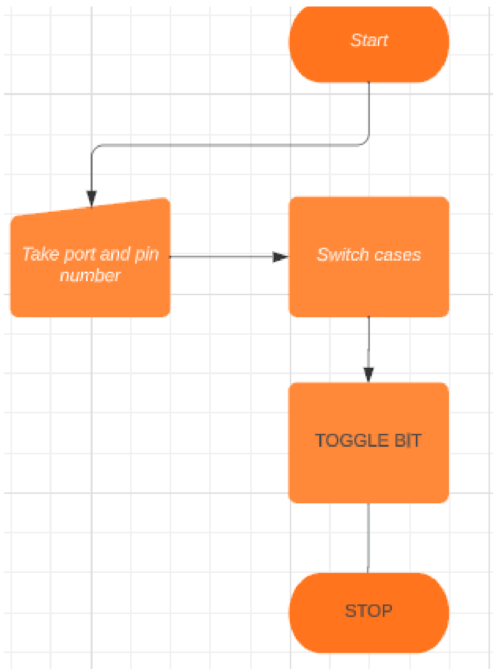
void MDIO_voidSetPinDirection (u8 Copy_u8Port , u8 Copy_u8Pin , u8 Copy_u8Dir)



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

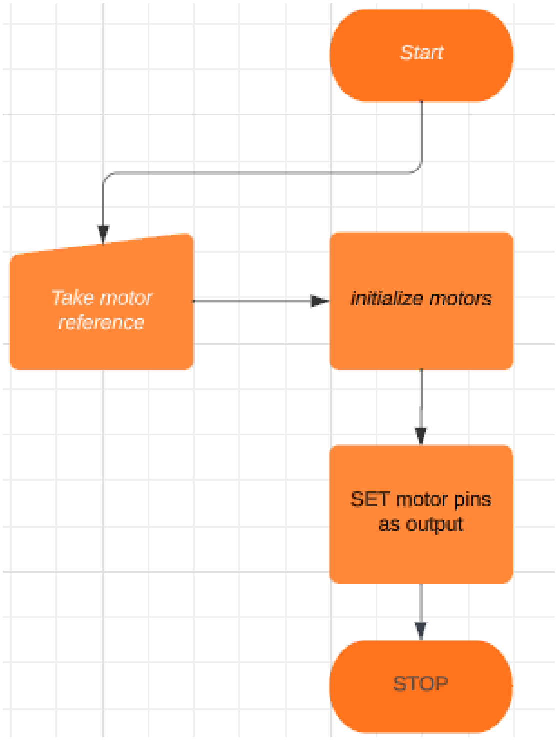


void MDIO_voidTogglePinValue(u8 Copy_u8Port , u8 Copy_u8Pin)

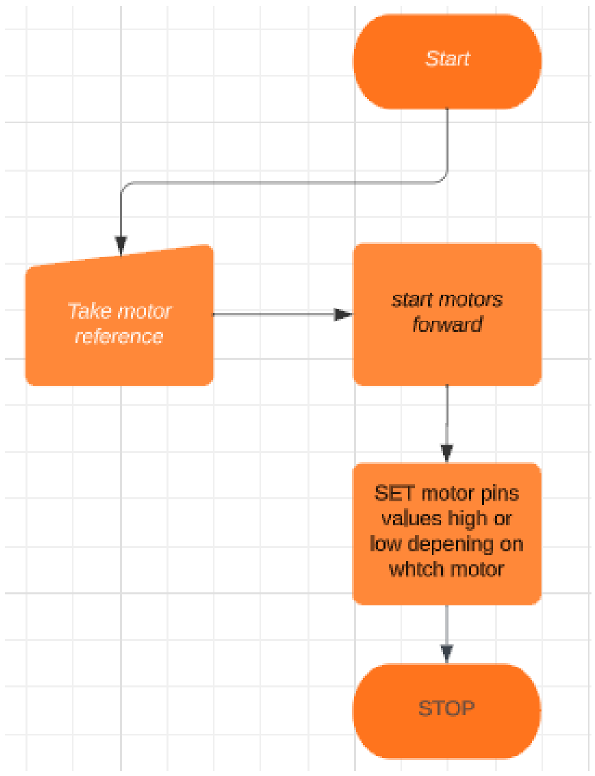


MOTOR:

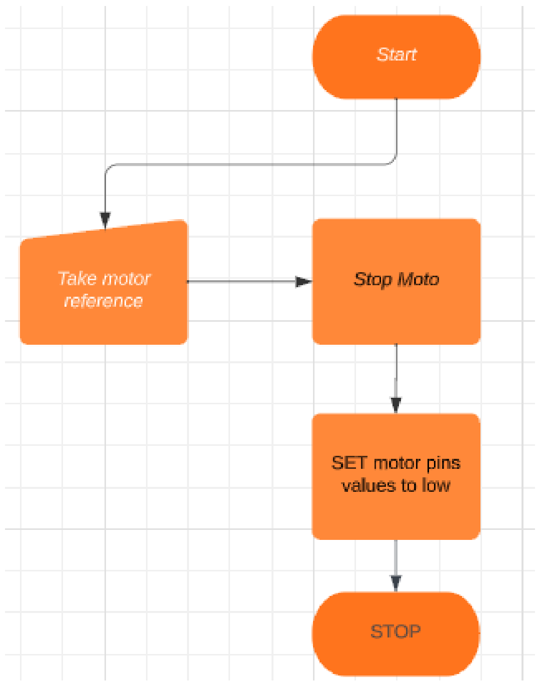
Init function



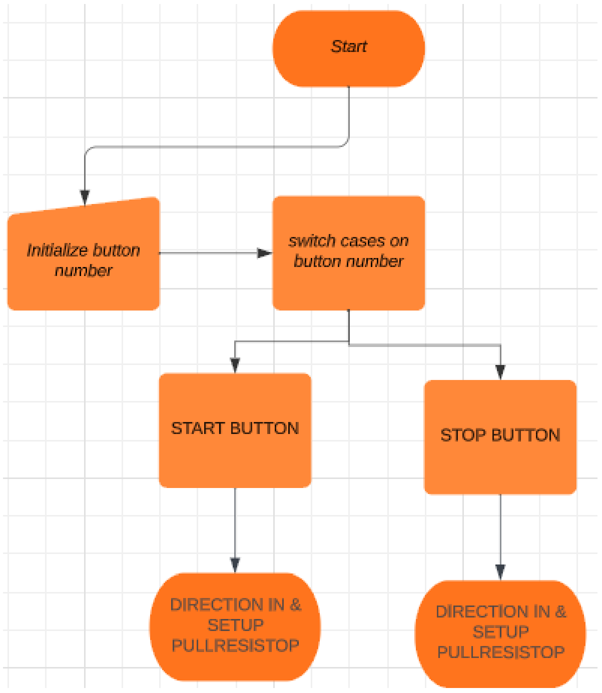
start forward function



Stop function

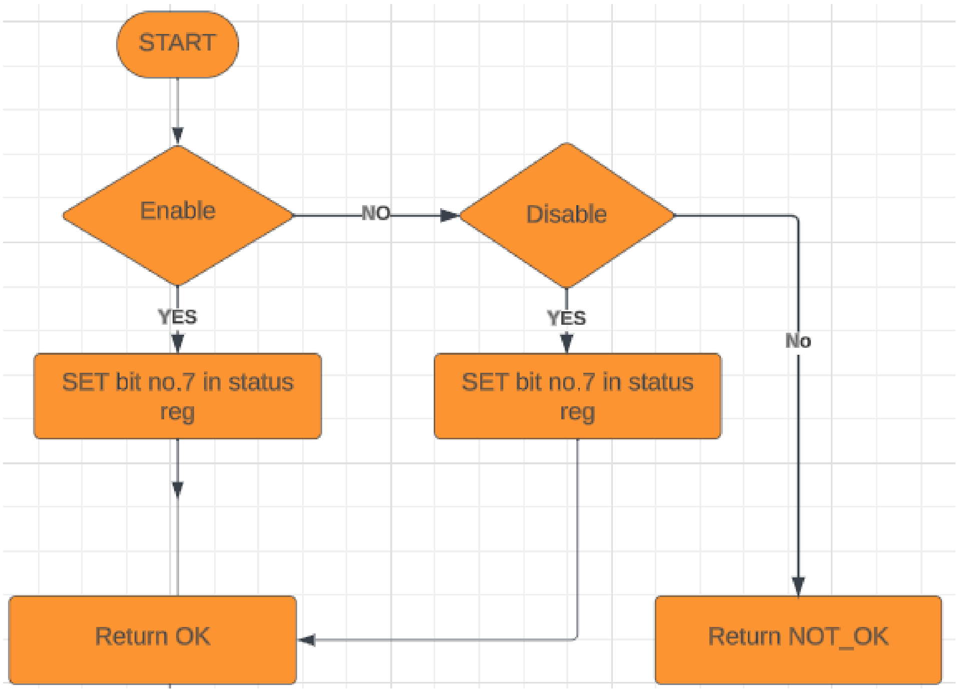


BUTTON:

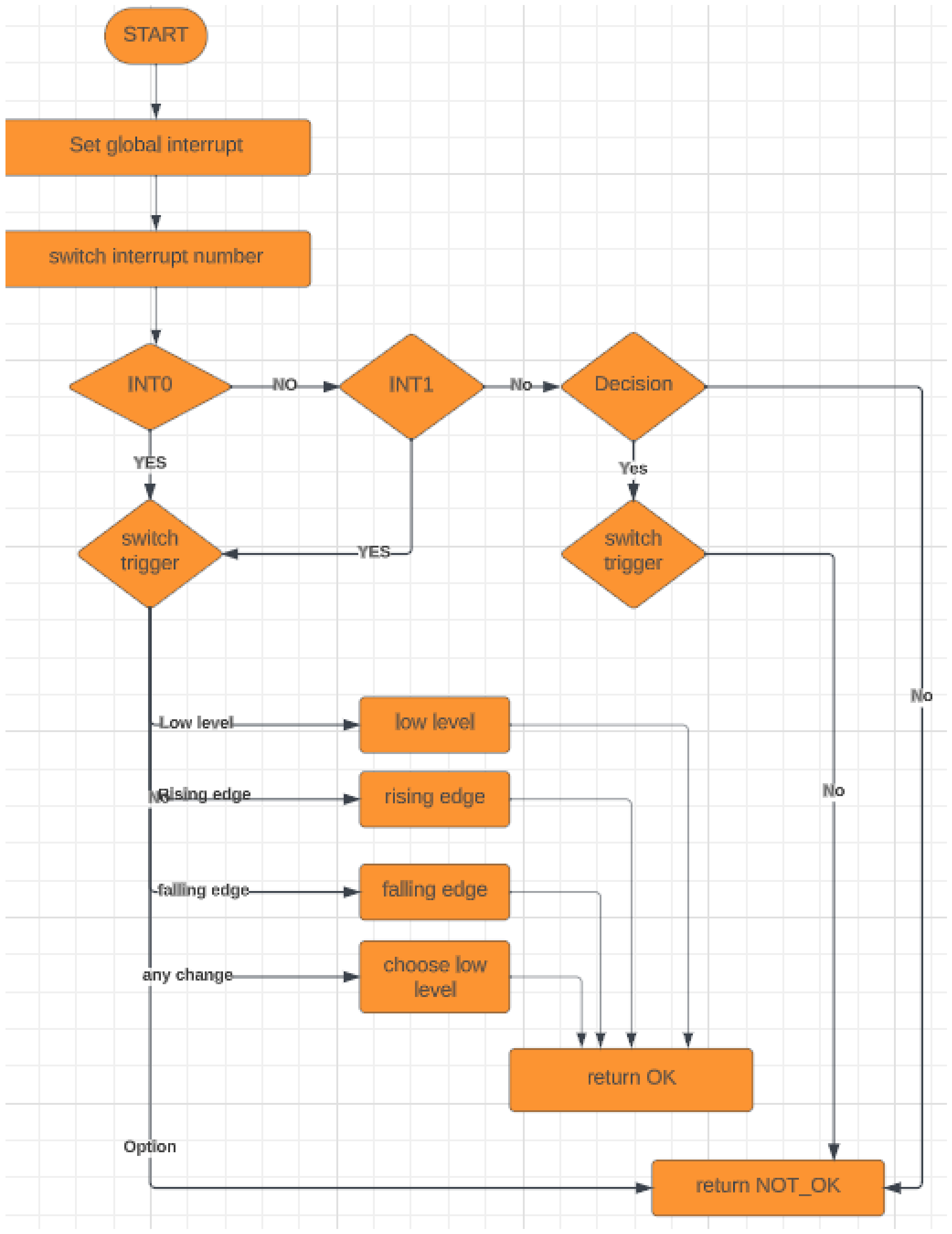

Start
Initialize button number
switch cases on button number
START BUTTON
STOP BUTTON
DIRECTION IN & SETUP PULLRESISTOP
DIRECTION IN & SETUP PULLRESISTOP

External Interrupt:

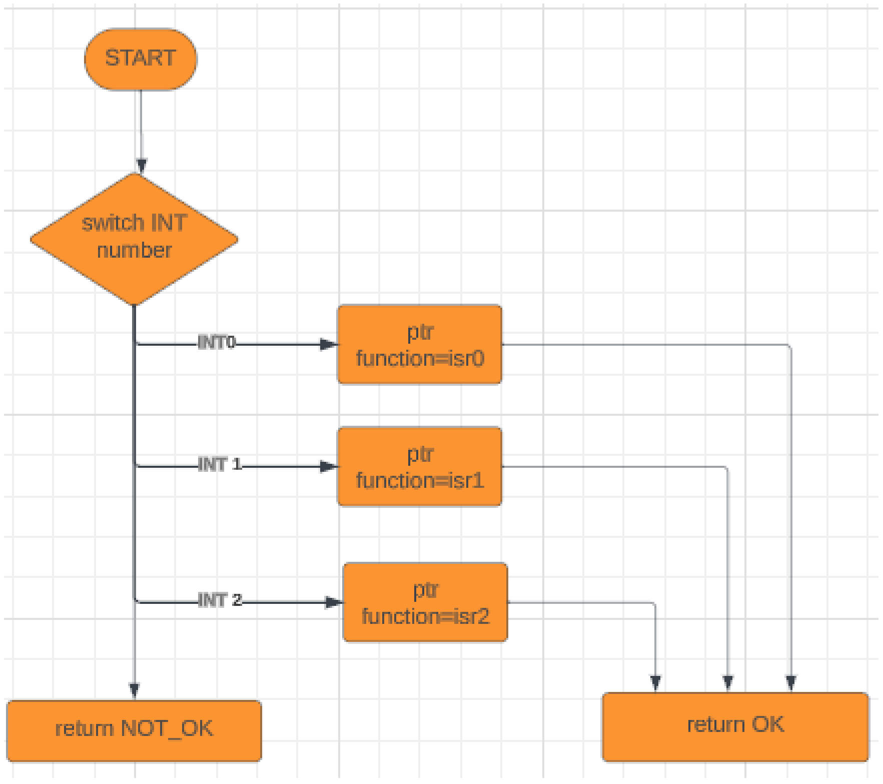
EN_EXTINT_ERROR SET_GLOBAL_INTERRUPT(EN_GLOBAL_INT state)



EN_EXTINT_ERROR EXTINT_init(EN_EXINT_NUMBER INTx ,EN_Sense_Control INTxSense)

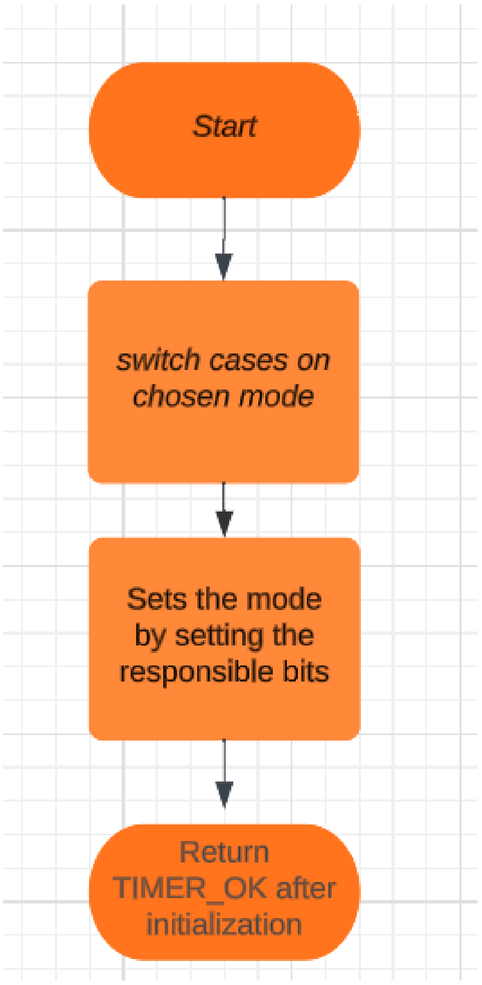


EN_EXTINT_ERROR EXTINT_CallBack(EN_EXINT_NUMBER INTx,void(*ptrfunc)(void))

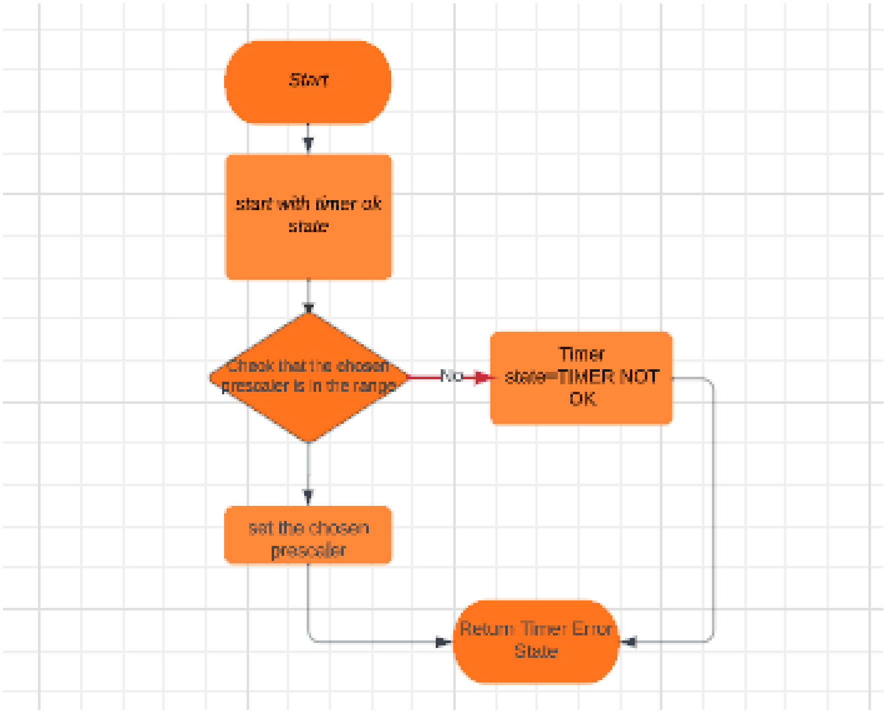


Timer:

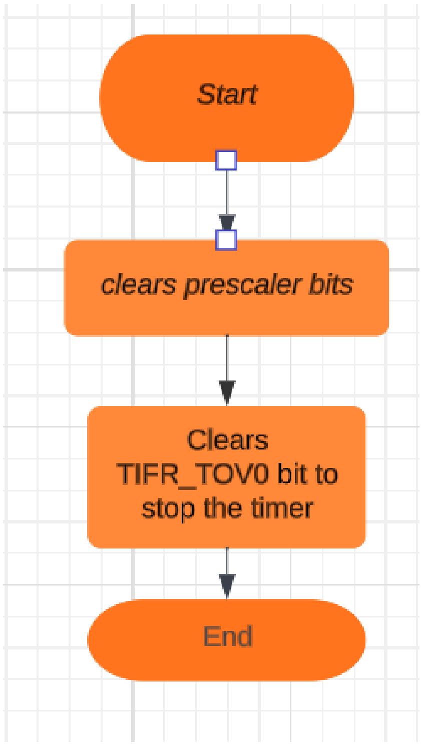
en_timer0ErrorState_t MTIMER0_init(en_timer0Mode_t u8_a_mode)


Start
switch cases on chosen mode
Sets the mode by setting the responsible bits
Return TIMER_OK after initialization

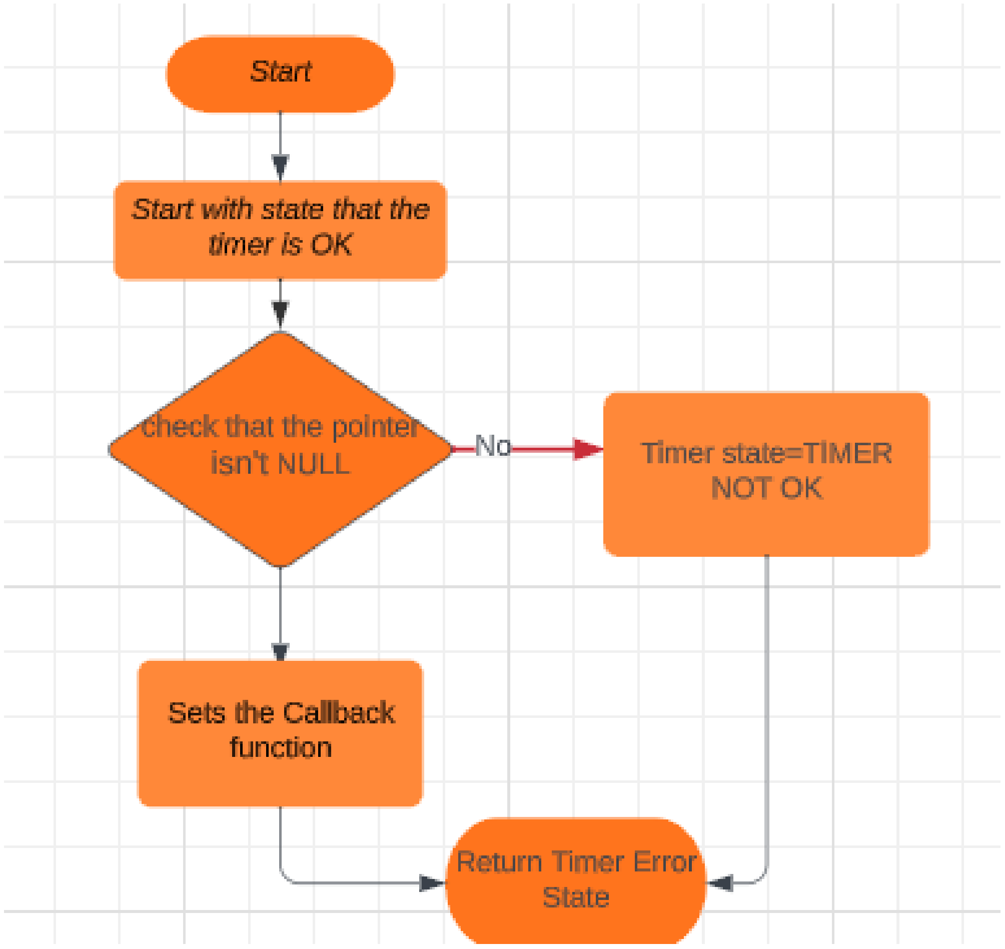
en_timer0ErrorState_t MTIMER0_startTimer(en_TIMER0_CLK_SELECT_t u8_a_prescaler)



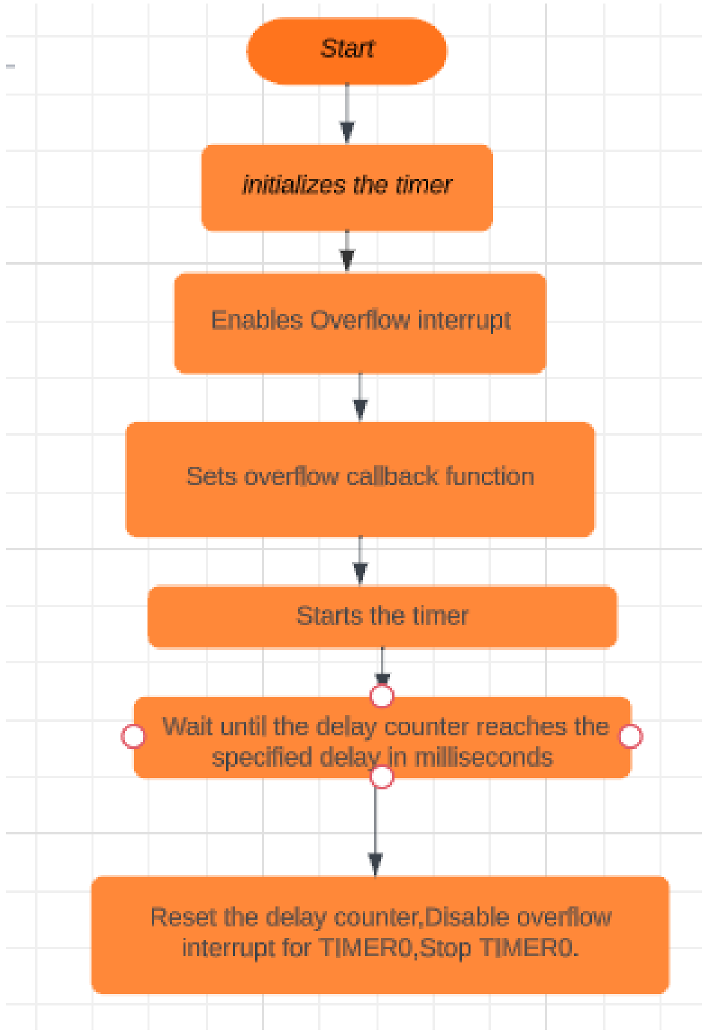
void MTIMER0_stop()



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


Start
Start with state that the timer is OK
check that the pointer isn't NULL
No
Timer state=TIMER NOT OK
Sets the Callback function
Return Timer Error State

void delay_ms_UsingTimer(u16 u16_delay_ms)


Start
initializes the timer
Enables Overflow interrupt
Sets overflow callback function
Starts the timer
Wait until the delay counter reaches the specified delay in milliseconds
Reset the delay counter,Disable overflow interrupt for TIMER0,Stop TIMER0.

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)

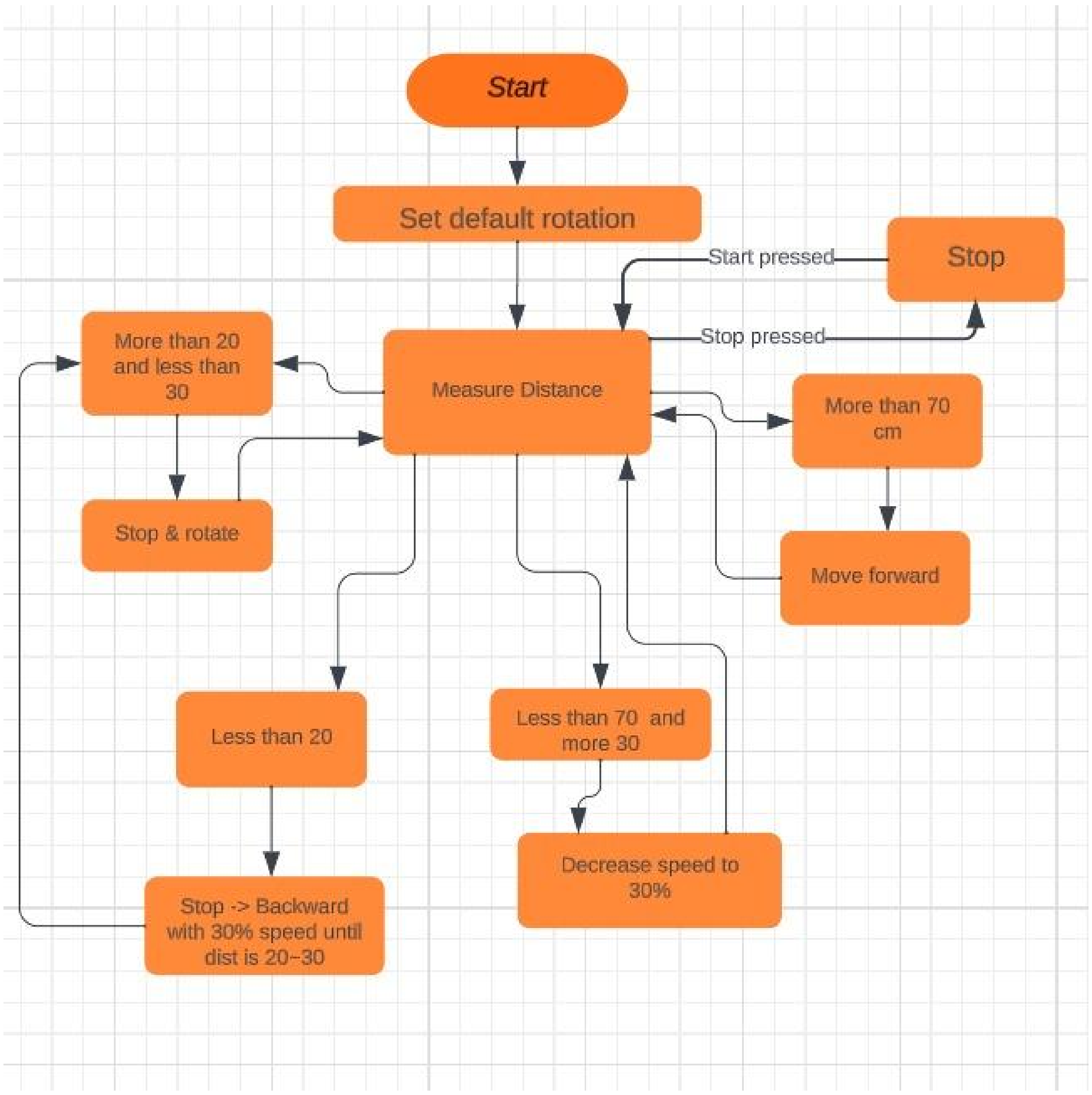
If the robot rotates 360 degrees without finding a distance greater than 20 cm, stop and update LCD data.

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:


Start
Set default rotation
Stop
Start pressed
Stop pressed
Measure Distance
More than 20 and less than 30
Stop & rotate
More than 70 cm
Move forward
Less than 20
Less than 70 and more 30
Stop -> Backward with 30% speed until dist is 20~30
Decrease speed to 30%