DESIGN DOCUMENT

MOVING CAR PROJECT DESCRIPTION

Team7:

-Amr El-Abd

-Kareem Polaky

-Nada Abdelazim

-Moustafa Abdelrahim

Table of Contents: -

Project introduction:

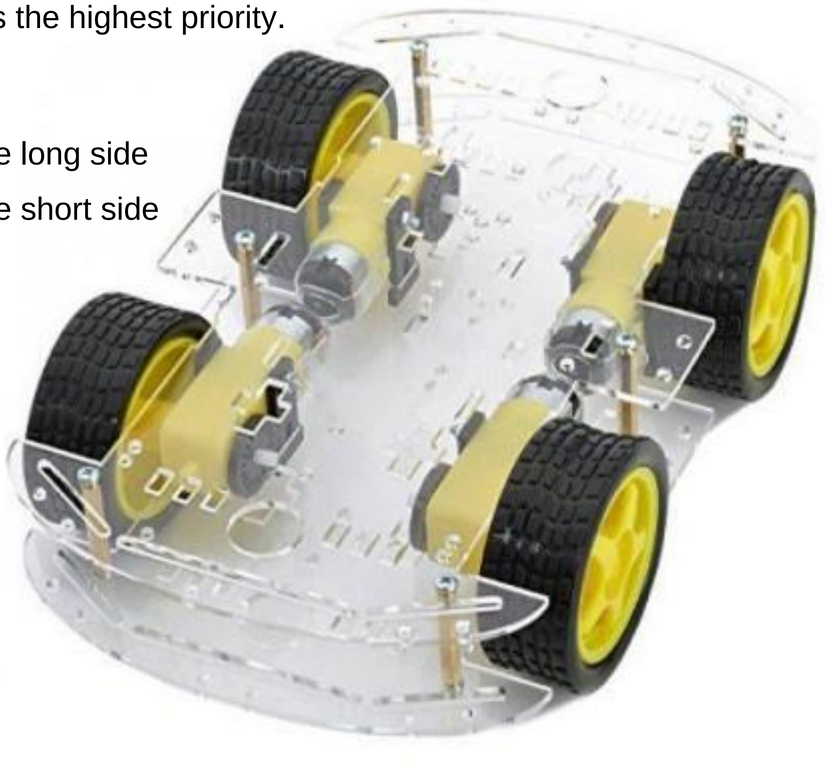
The aim of this project is to design a system for a four-driving wheel robot that moves in a rectangular shape. The system will consist of several modules, including Interrupt, timers, motor driver, LED indicators, and button drivers. The robot will be equipped with four motors, two buttons (one for start and one for stop), and four LEDs to indicate the state of movement.

To ensure smooth operation, a backlog for the team will be created in an Excel sheet, which will contain the task name, assignee, task status, expected time to finish, and actual time to finish.

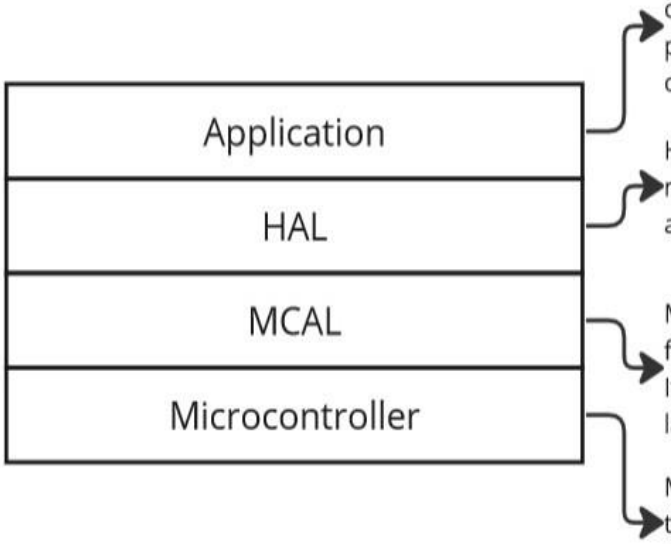
Project sequence :-

1. The car starts initially from 0 speed.
2. When PB1 is pressed, the car will move forward after 1 second.
3. The car will move forward to create the longest side of the rectangle for 3 seconds with 50% of its maximum speed.
4. After finishing the first longest side the car will stop for 0.5 seconds, rotate 90 degrees to the right, and stop for 0.5 second.
5. The car will move to create the short side of the rectangle at 30% of its speed for 2 seconds.
6. After finishing the shortest side, the car will stop for 0.5 seconds, rotate 90 degrees to the right, and stop for 0.5 second.
7. Steps 3 to 6 will be repeated infinitely until you press the stop button (PB2).
8. PB2 acts as a sudden break, and it has the highest priority.
9. **<u>LEDs Operations</u>**

1. LED1: On means moving forward on the long side
2. LED2: On means moving forward on the short side
3. LED3: On means stop
4. LED4: On means Rotating



Layered Architectures:-



Application

HAL

MCAL

Microcontroller

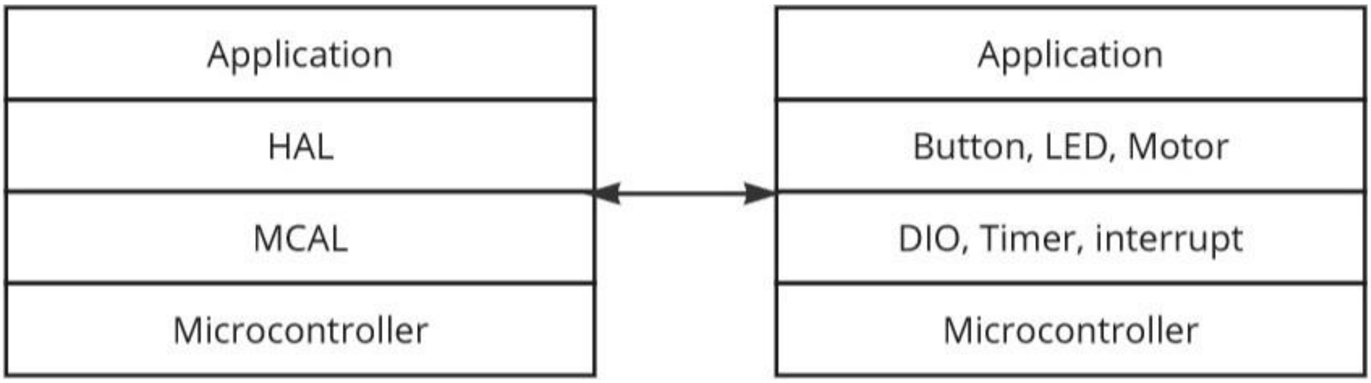
Application Layer: This is the topmost layer of the software stack, which contains the actual application logic. It interacts with the lower layers to perform its tasks. It is responsible for implementing the desired functionality of the system.

HAL: This layer provides an abstraction for external devices connected to the microcontroller. The HAL layer provides interface to access external devices and hides the implementation details from the application layer.

MCAL (Microcontroller Abstraction Layer): This layer provides an abstraction for the microcontroller hardware. It includes low-level drivers for peripherals. It hides the hardware details and provides a uniform interface to the upper layers.

Microcontroller: This layer represents the physical hardware layer consisting of the microcontroller chip. The microcontroller is responsible for executing the code stored in its memory and controlling the behavior of the system.

Modules description:-



Application	Application
HAL	Button, LED, Motor
MCAL	DIO, Timer, interrupt
Microcontroller	Microcontroller

- Button module will use DIO module to register its state
- LED module will use DIO module to detect its state
- Motor module will use timer's PWM function to control its speed
- DIO module provides low-level hardware access to the microcontroller
- Timer module can be used by both the motor and the Application layer to implement timing functions.
- Interrupt module can be used by the app module to control the logic

Drivers' documentation:-

1. DIO Driver:

Description: The DIO (Digital Input Output) driver is responsible for setting up the digital pins of the microcontroller to either input or output mode. This driver will be used to control the buttons and LEDs.

Functions:

```
DIO_ERROR_TYPE DIO_INITPIN(DIO_PIN_TYPE PIN,DIO_PINSTATUS_TYPE STATUS);
DIO_ERROR_TYPE DIO_WRITEPIN(DIO_PIN_TYPE PIN,DIO_VOLTAGE_TYPE VOLTAGE);
DIO_ERROR_TYPE DIO_READPIN(DIO_PIN_TYPE PIN,DIO_VOLTAGE_TYPE* VOLT);
void DIO_TogglePin(DIO_PIN_TYPE pin);
```

2. Timer Driver:

Description: The Timer driver is responsible for setting up and controlling the timers of the microcontroller. This driver will be used to create the timing delays required in the project.

Functions:

//timer 0 prototypes

```
Timer_ErrorStatus TIMER_0_init(Timer_Mode mode);
Timer_ErrorStatus TIMER_0_start(Timer_Prescaler prescaler);
void TIMER_0_stop(void);
Timer_ErrorStatus TIMER_0_setIntialValue(uint8_t value);
Timer_ErrorStatus TIMER_0_OvfNum(double overflow);
void TIMER_0_DELAY_MS(double _delay);
```

//timer 2 prototypes

```
Timer_ErrorStatus TIMER_2_init(Timer_Mode mode);
Timer_ErrorStatus TIMER_2_start(Timer_Prescaler prescaler);
void TIMER_2_stop(void);
Timer_ErrorStatus TIMER_2_setIntialValue(uint8_t value);
Timer_ErrorStatus TIMER_2_OvfNum(double overflow);
void TIMER_2_DELAY_MS(double _delay);
void TIMER_2_INT();

//PWM Function prototype

void TIMER_0_pwm(float intial);
```

3. Interrupt Driver:

Description: The Interrupt driver is responsible for setting up and controlling the interrupts of the microcontroller. This driver will be used to detect button presses.

Functions:

```
EN_int__error_t EXI_Enable (EN_int_t Interrupt);
EN_int__error_t EXI_Disable (EN_int_t Interrupt);
EN_int__error_t EXI_Trigger(EN_int_t Interrupt,EN_trig trigger);
void EXI_SetCallBack(EN_int_t Interrupt,void(*ptrf)(void));
```

4. Button Driver:

Description: The Button driver is responsible for setting up and controlling the buttons of the microcontroller. This driver will be used to detect button presses.

Functions:

```
BUTTON_ERROR_TYPE Button_INIT(DIO_PIN_TYPE PIN);
BUTTON_ERROR_TYPE Button_read(DIO_PIN_TYPE PIN,DIO_VOLTAGE_TYPE*VOLT);
```

5. LED Driver:

Description: The LED driver is responsible for setting up and controlling the LEDs of the microcontroller. This driver will be used to indicate the status of the car.

Functions:

```
LED_ERROR_TYPE LED_INIT(DIO_PIN_TYPE PIN);
LED_ERROR_TYPE LED_ON(DIO_PIN_TYPE PIN);
LED_ERROR_TYPE LED_OFF(DIO_PIN_TYPE PIN);
```

6. Motor Driver:

Description: The Motor driver is responsible for setting up and controlling the motors of the car. This driver will be used to control the speed and direction of the motors.

Functions:

```
void Motors_init(void);
void Motors_Start(void);
void Motors_Rotating(void);
void Motors_Stop(void);
```

Application stages and prototypes:-

Ready state : this is the first state of the project sequence which occurs after pressing on the start push button. In this state all motors and LEDs are off and the car waits for 1 second.

```
void ready_State(void);
```

Long Side stage : in this stage the car will move forward to create the longest side of the rectangle for 3 seconds with 50% of its maximum speed.

```
void longSide_start();
```

Rotate stage : After finishing the first longest side the car will stop for 0.5 seconds, rotate 90 degrees to the right, and stop for 0.5 second.

```
void car_stop_state (void);
void rotate_90degree_calculation (void);
void rotate_90degree_state (void);
```

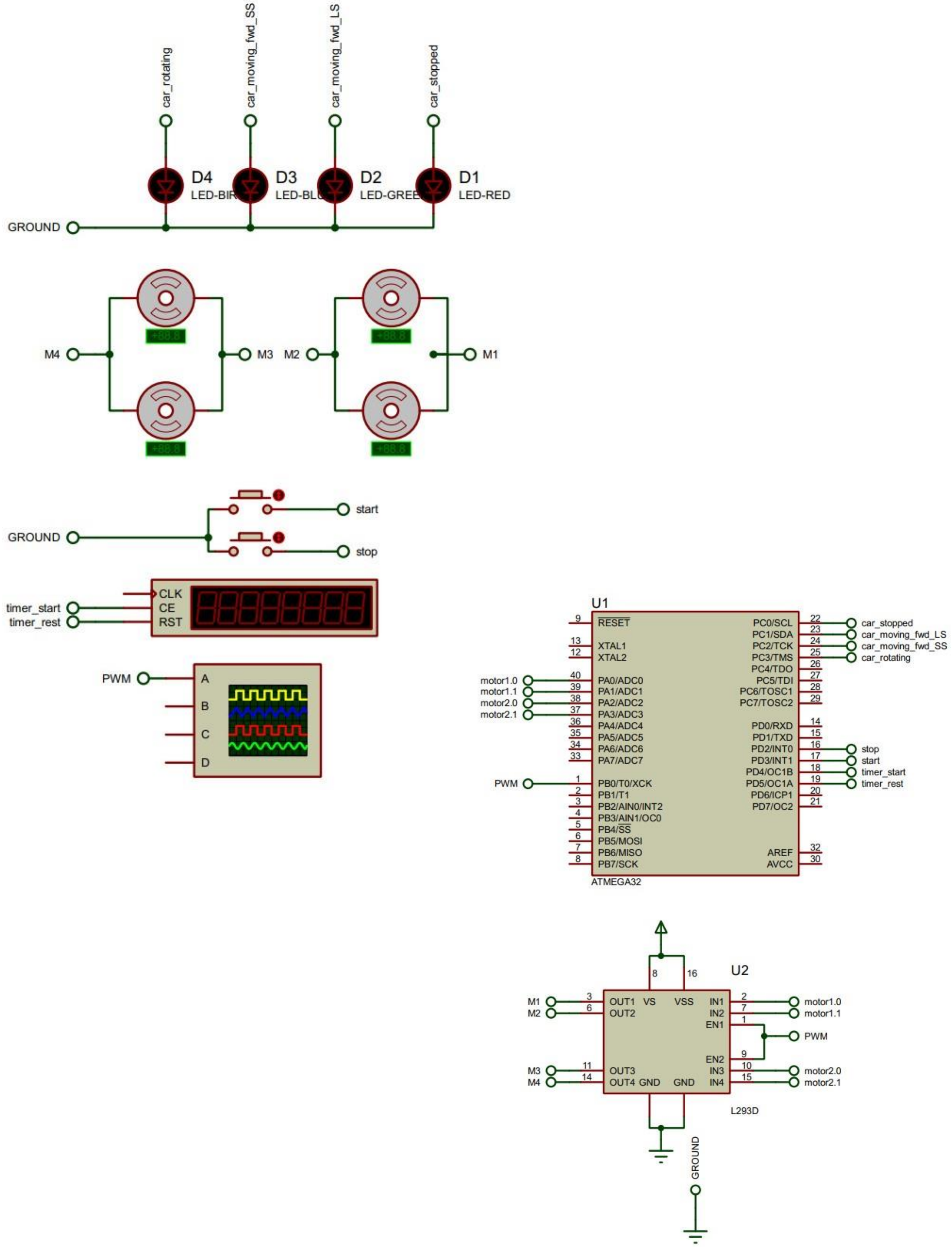
Short Side stage : in this stage the car will move forward to create the short side of the rectangle for 2 seconds with 30% of its maximum speed.

```
void shortSide_start();
```

Main : this is the gathering of the stages functions into one logical indefinite looping cycle to complete the required project sequence.

```
void app_Start(void);
void app_Init(void);
```

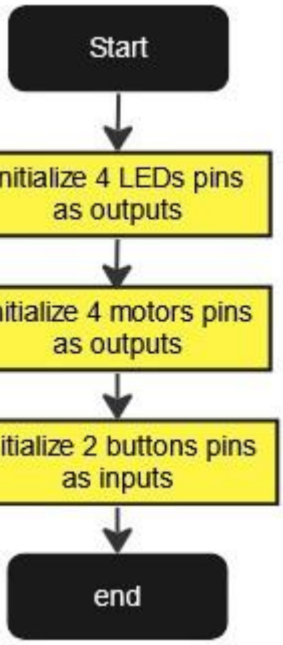
Proteus simulation design :-



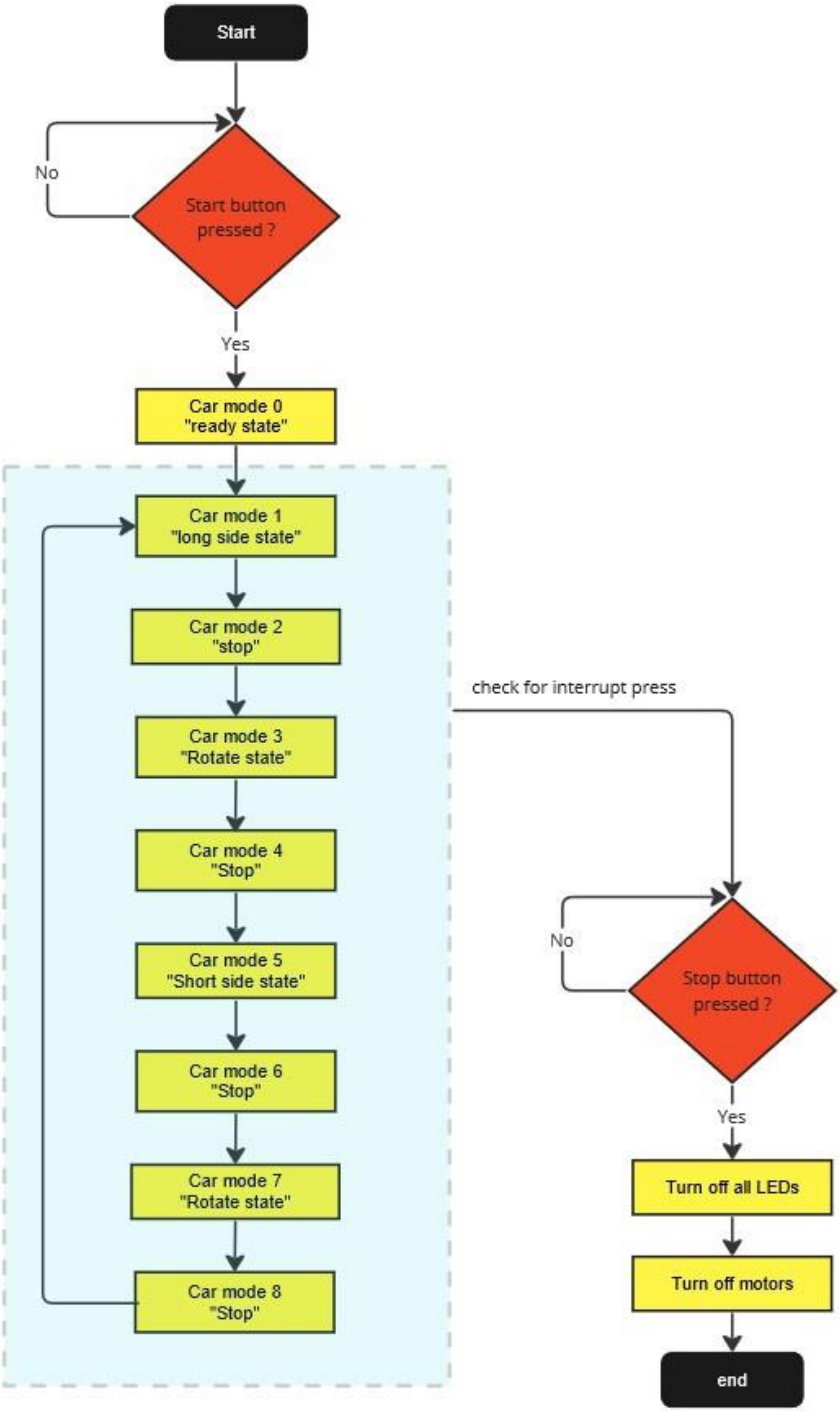
Flowcharts for Functions from Higher layers downwords:

App.h functions:

```
void app_Init(void);
```


Start
initialize 4 LEDs pins as outputs
initialize 4 motors pins as outputs
initialize 2 buttons pins as inputs
end

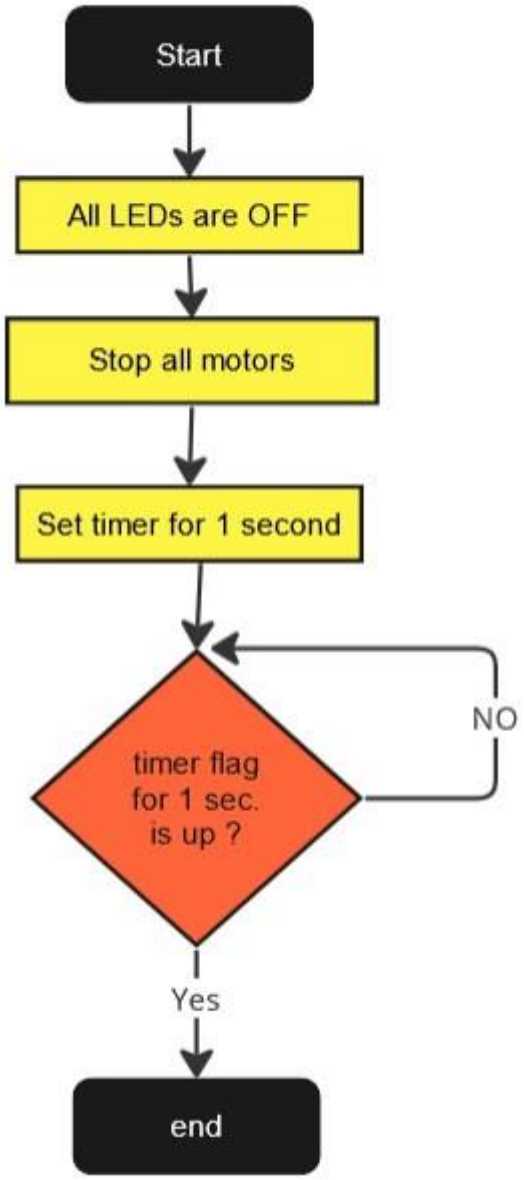
```
void app_Start(void);
```



Stages flowcharts:

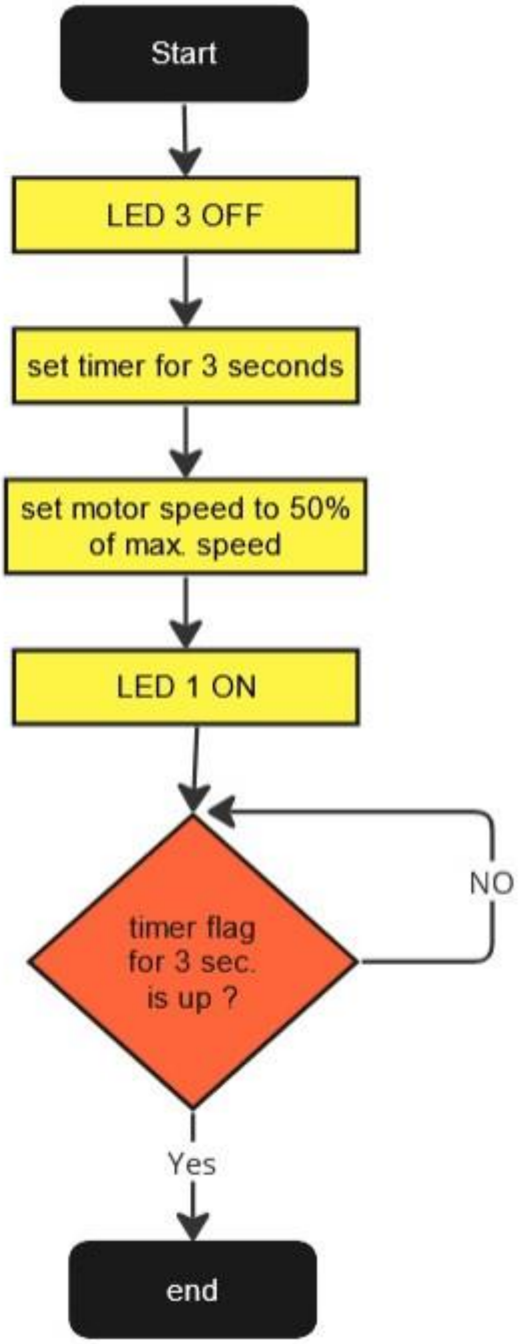
Ready state :

```
void ready_State(void);
```



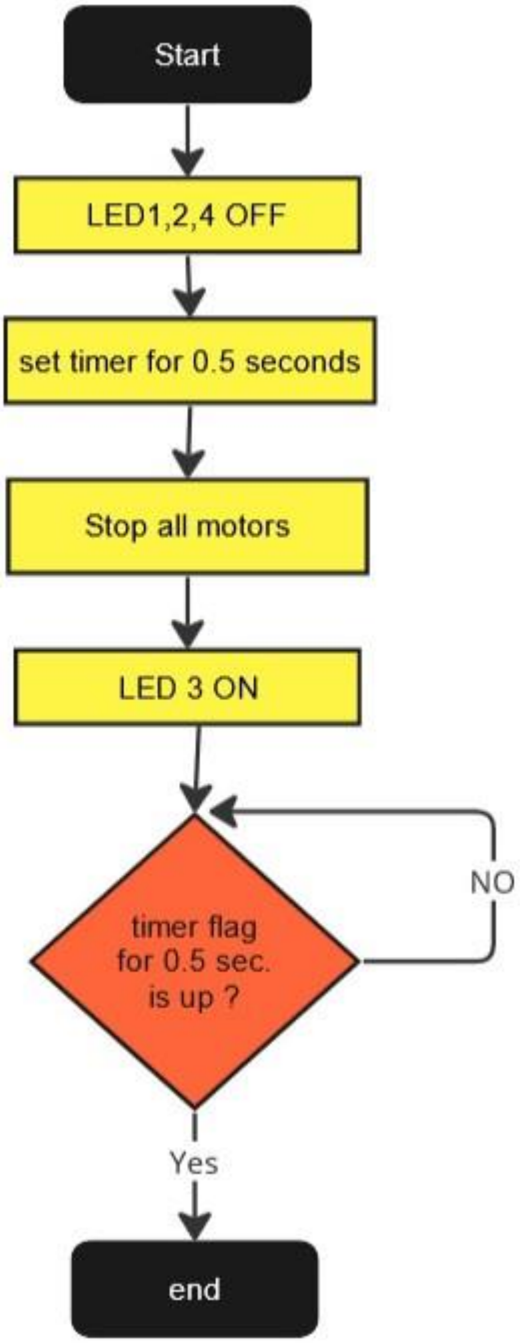
Long Side state:

```
void longSide_start();
```


Start
LED 3 OFF
set timer for 3 seconds
set motor speed to 50% of max. speed
LED 1 ON
timer flag for 3 sec. is up ?
NO
Yes
end

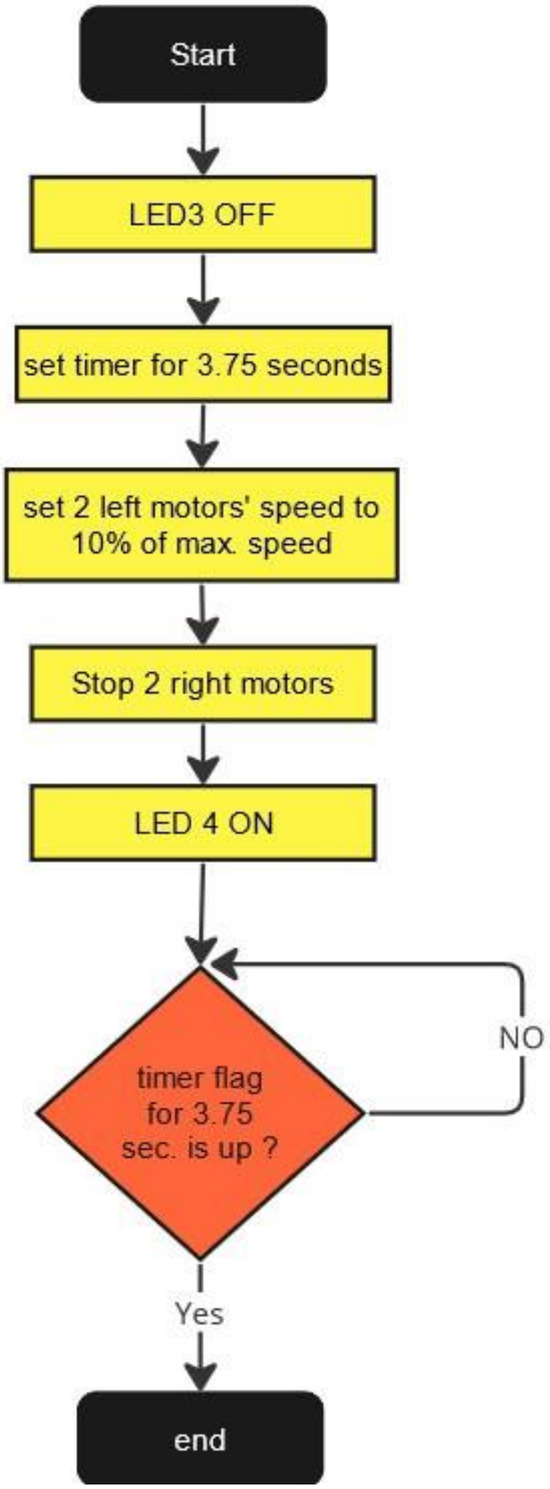
Stop state:

```
void car_stop_state (void);
```



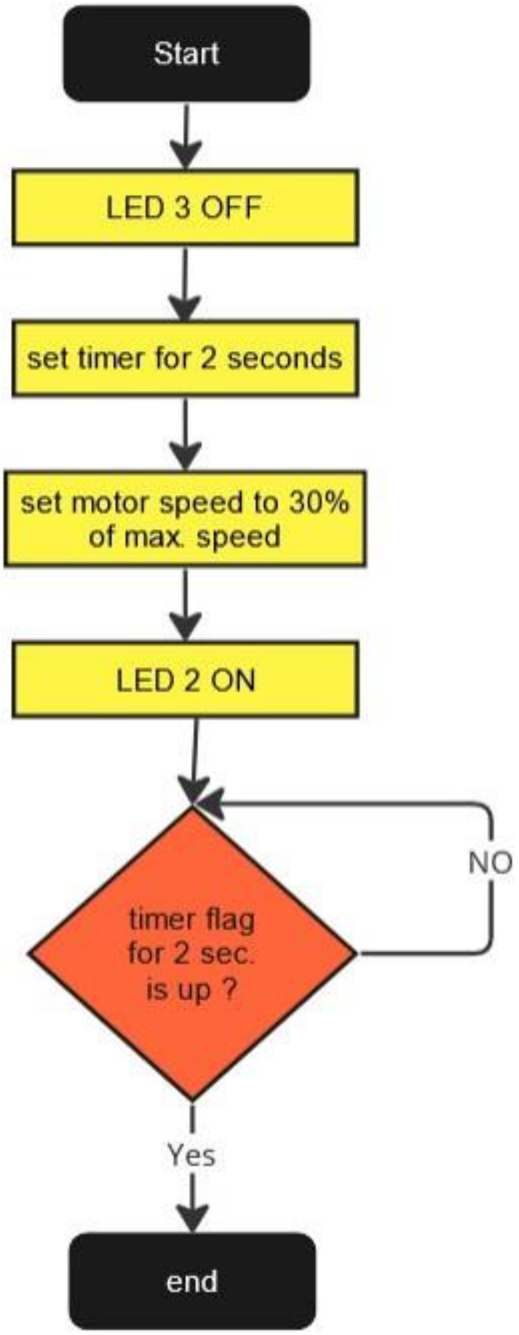
Rotate State:

```
void rotate_90degree_state (void);
```


Start
LED3 OFF
set timer for 3.75 seconds
set 2 left motors' speed to 10% of max. speed
Stop 2 right motors
LED 4 ON
timer flag for 3.75 sec. is up ?
NO
Yes
end

Short side state:

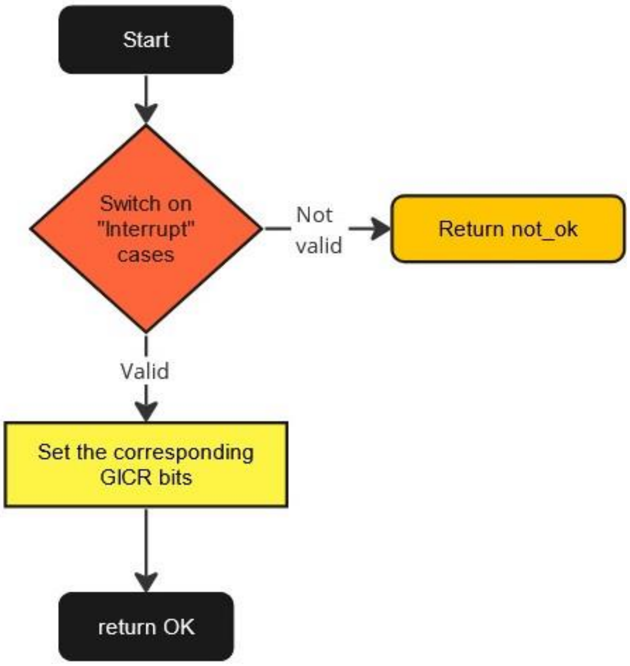
```
void shortSide_start(void);
```


Start
LED 3 OFF
set timer for 2 seconds
set motor speed to 30% of max. speed
LED 2 ON
timer flag for 2 sec. is up ?
NO
Yes
end

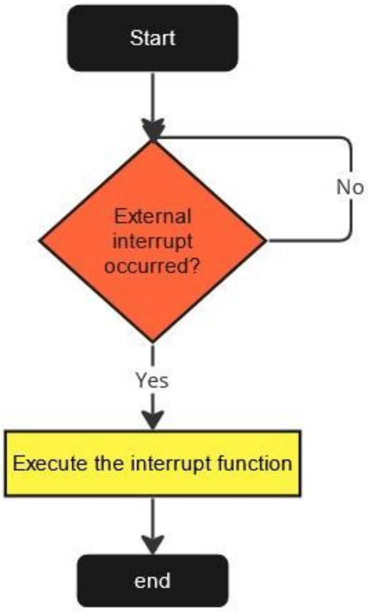
MCAL Layer:-

Interrupt functions’ flowcharts

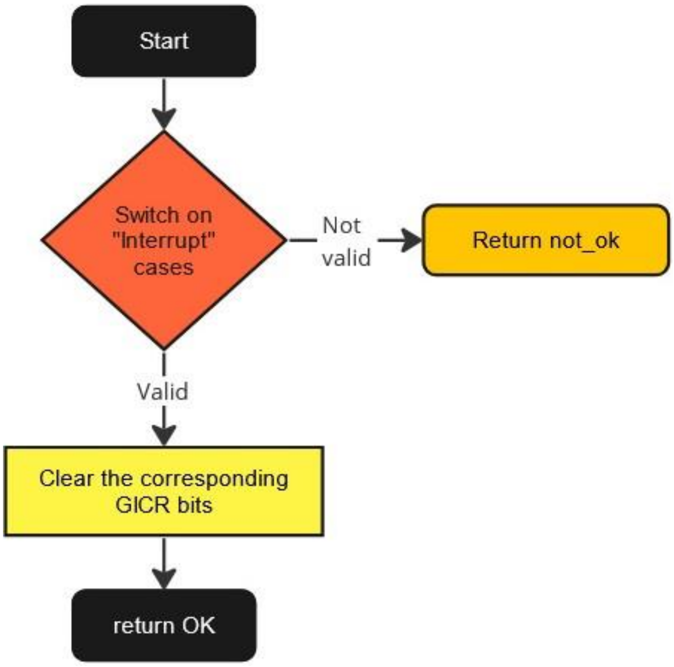
```
EN_int__error_t EXI_Enable (EN_int_t Interrupt)
```



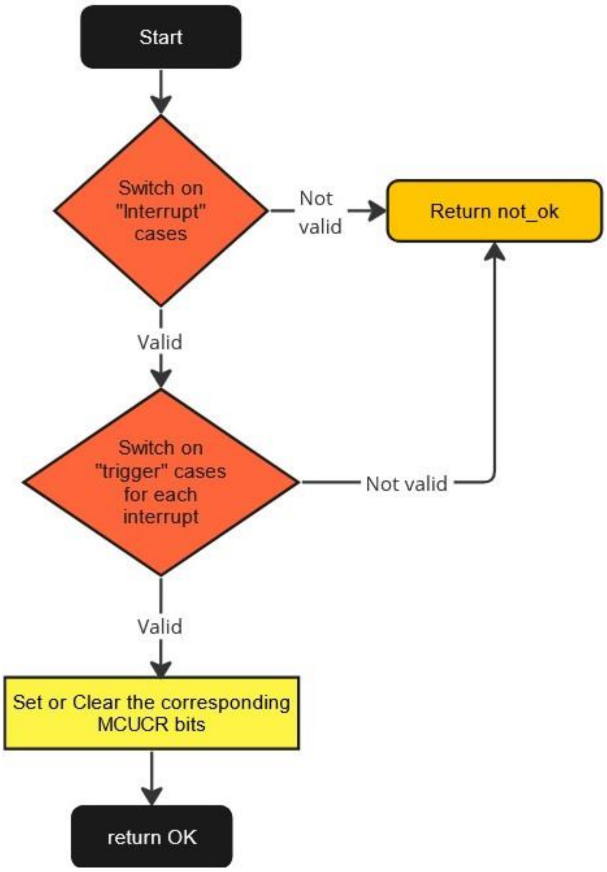
```
ISR (INTx_vect)
```



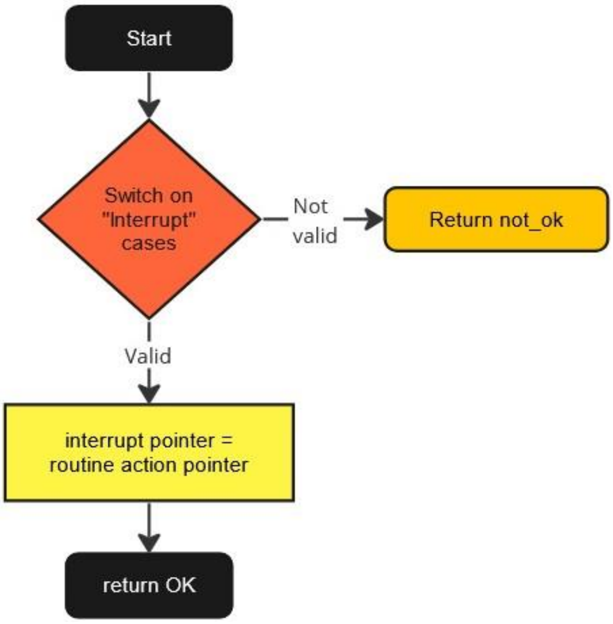
```
EN_int__error_t EXI_Disable (EN_int_t Interrupt)
```


Start
Switch on "Interrupt" cases
Not valid
Return not_ok
Valid
Clear the corresponding GICR bits
return OK

```
EN_int__error_t EXI_Trigger(EN_int_t Interrupt,EN_trig trigger)
```

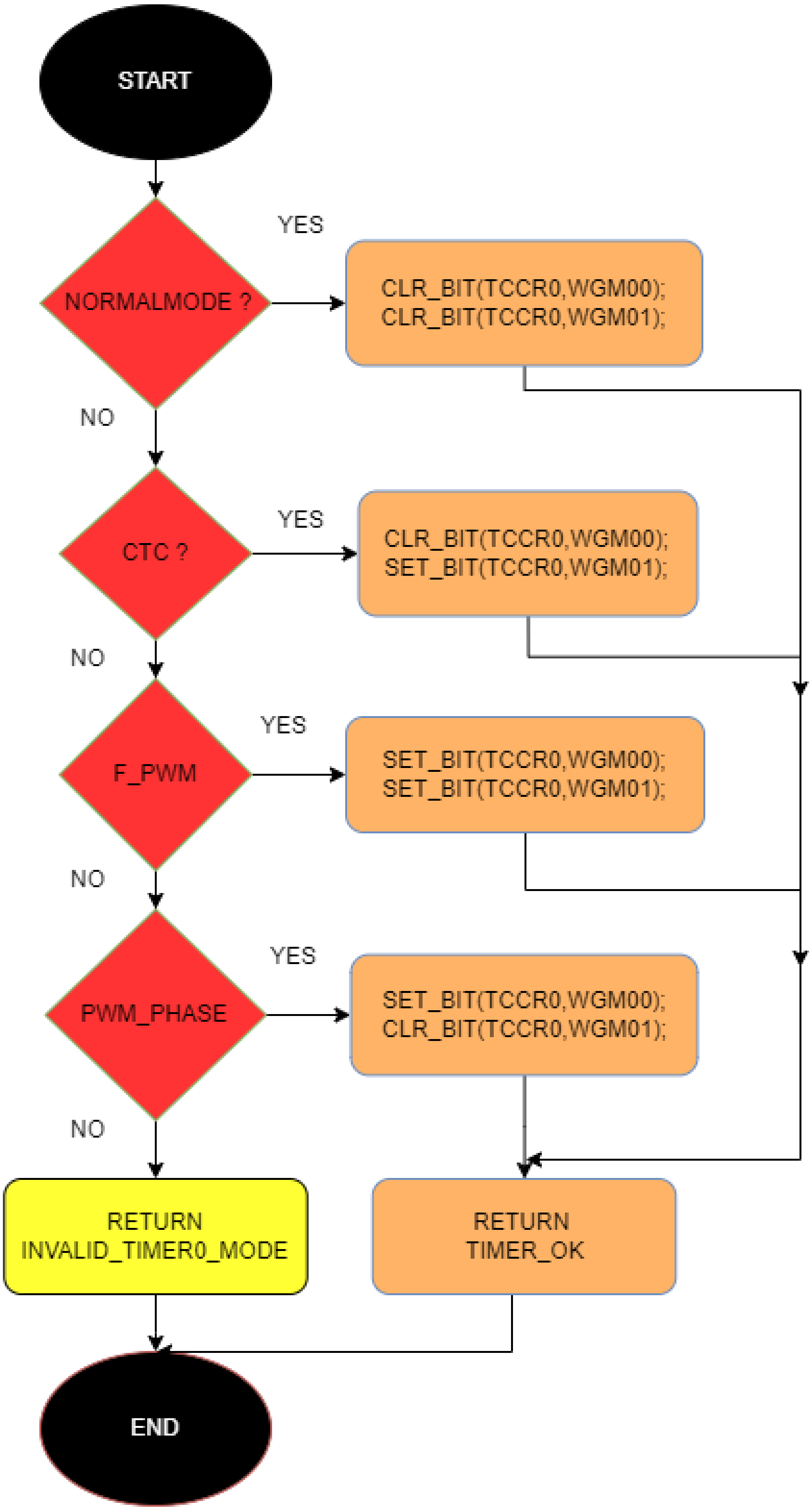

Start
Switch on "Interrupt" cases
Not valid
Return not_ok
Valid
Switch on "trigger" cases for each interrupt
Not valid
Valid
Set or Clear the corresponding MCUCR bits
return OK

```
EN_int__error_t EXI_SetCallBack(EN_int_t Interrupt,void(*ptrf)(void))
```

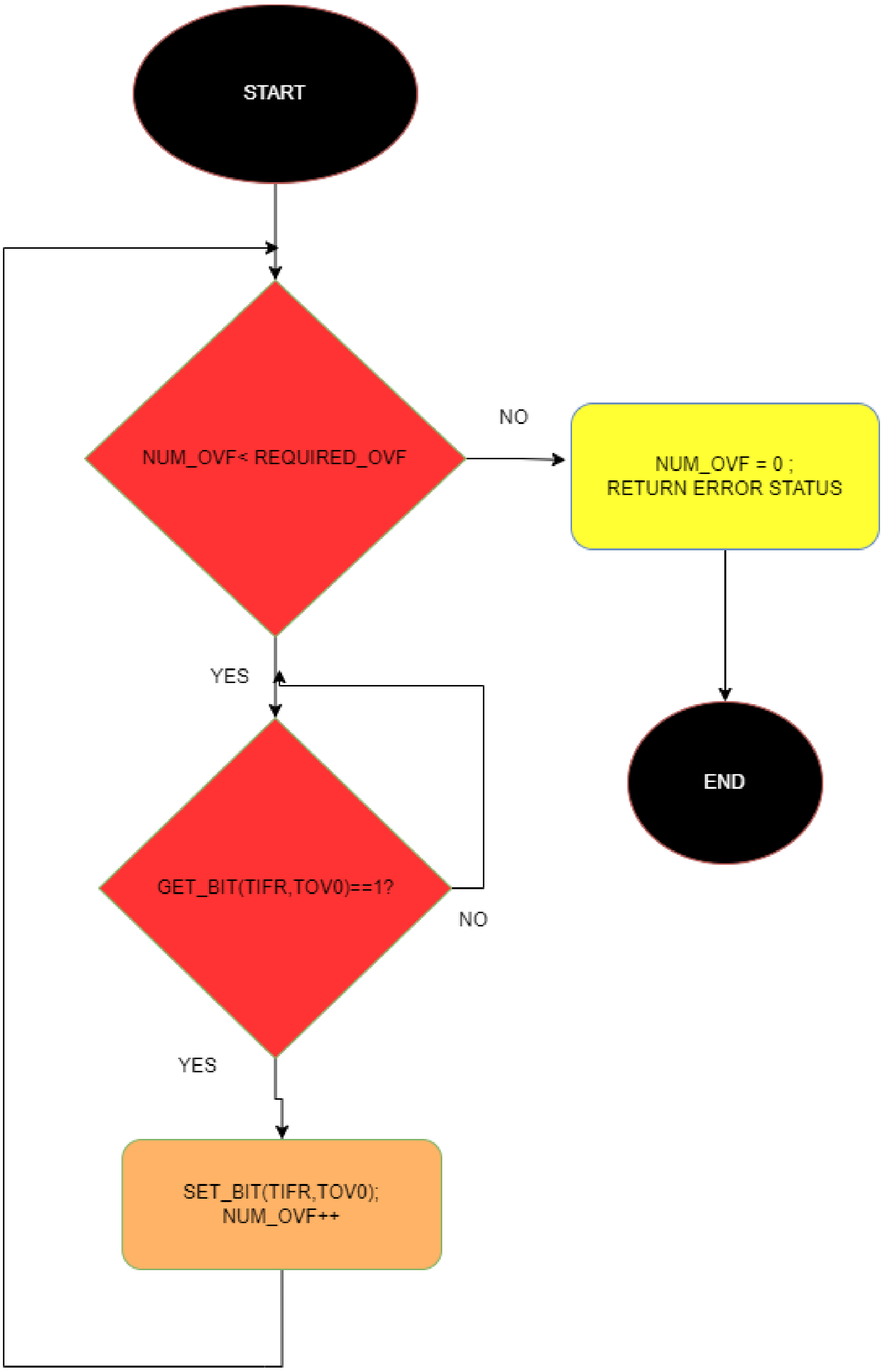

Start
Switch on "Interrupt" cases
Not valid
Return not_ok
Valid
interrupt pointer = routine action pointer
return OK

Timer functions' flowcharts:-

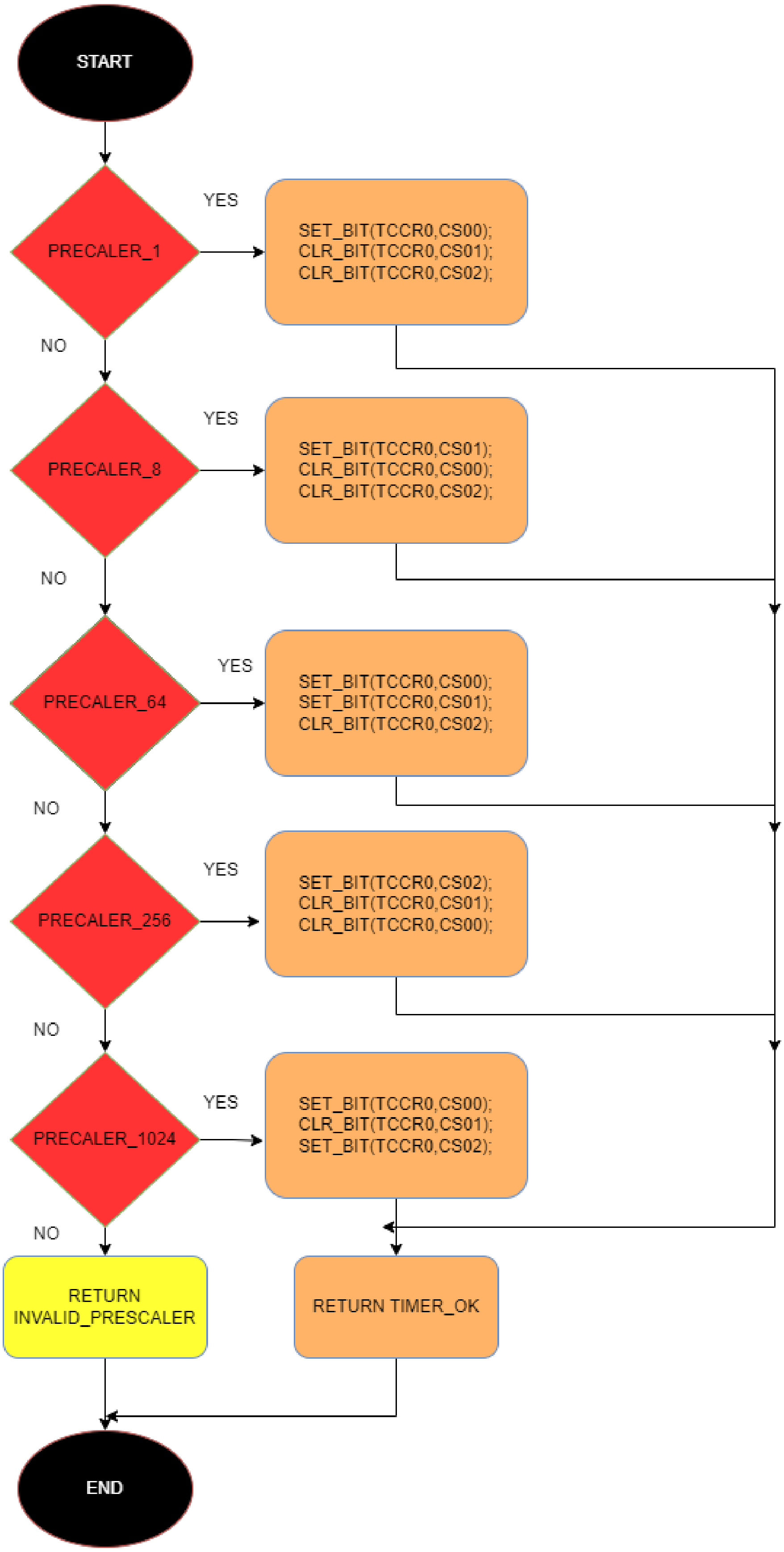
TIMER 0 INIT

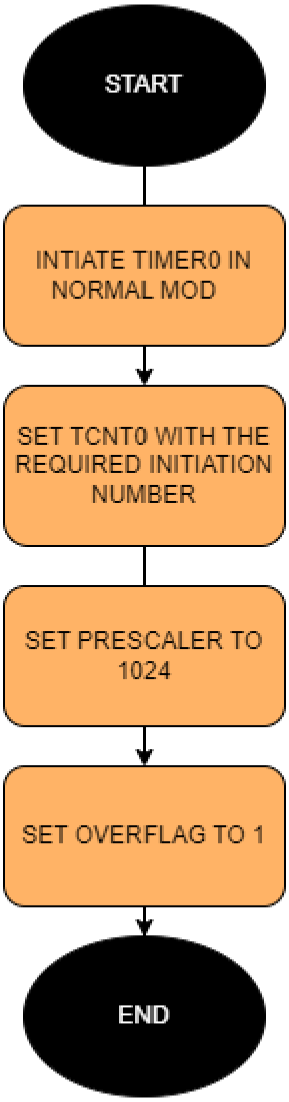


TIMER 0 SET OVERFLAG



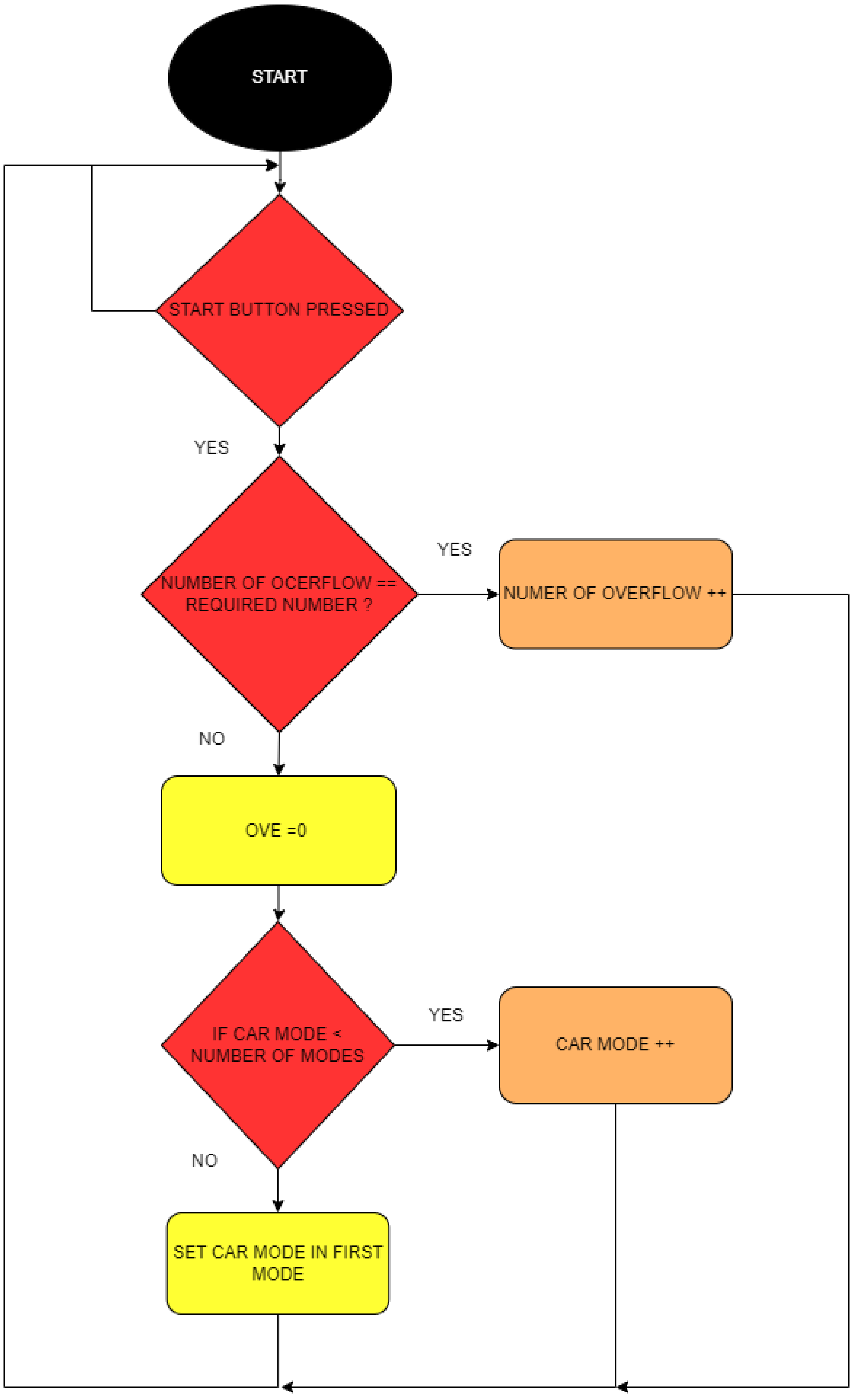
TIMER 0 SET PRESCALER



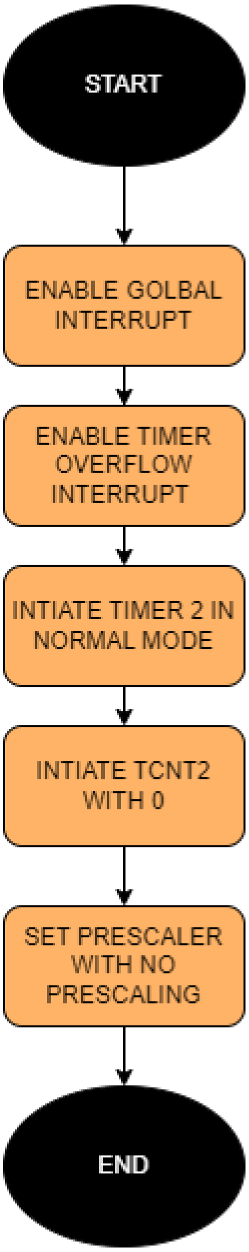

START
INTIATE TIMER0 IN NORMAL MOD
SET TCNT0 WITH THE REQUIRED INITIATION NUMBER
SET PRESCALER TO 1024
SET OVERFLAG TO 1
END

TIMER 0 PWM WITH NORMAL MODE

TIMER_2_ISR


START
START BUTTON PRESSED
YES
NUMBER OF OCERFLOW == REQUIRED NUMBER ?
YES
NUMER OF OVERFLOW ++
NO
OVE =0
IF CAR MODE < NUMBER OF MODES
YES
CAR MODE ++
NO
SET CAR MODE IN FIRST MODE

TIMER 2 WITH INTERRUPT


START
ENABLE GOLBAL INTERRUPT
ENABLE TIMER OVERFLOW INTERRUPT
INTIATE TIMER 2 IN NORMAL MODE
INTIATE TCNT2 WITH 0
SET PRESCALER WITH NO PRESCALING
END