Autonomous car project

submitted by

Abdullah Ashraf
Ayman Afifi
Mahmoud Ashour
Saif El-din Sultan Osman
Mariam Khaled

submitted on

11/12/2023

Table of Contents

I) Introduction:

Project description:

We will develop a car that moves autonomously using the ultrasonic sensor to measure the forward distance and with the aid of servo motor we will move that ultra sonic sensor in every possible direction to measure the best path for the car to choose and move to it, we also added another mode to our car ,it is controlling the car using Bluetooth module by sending character to the car from your phone which will force the car to move as you wish.

The components used in developing the autonomous car project are:

1) Atmega32 Microcontroller
2) car chassis
3) 4 wheels with 4 DC motors
4) Servo motor “SG90”
5) Ultrasonic sensor “HC-SR04”
6) LEDS
7) Buzzer
8) Bluetooth module “HC-05”
9) batteries
10) bread boards and jumper wires

II) Project design:

Our Design:

We used the layered architecture design to develop our project which means we have 4 layers APP, HAL, MCAL, UTILITIES.

how does our project work?

we will answer that question by discussing the APP layer and how it works.
Steps of our APP working:

1) So, at first there is a button that will let the user choose between the 2 modes available Ultrasonic or Bluetooth.

2) If he chooses Ultrasonic now, we call the Ultrasonic functions to initialize the ultrasonic and make it ready to measure the forward distance.

3) then we measure the forward distance and here we have two scenarios. First 1 is that forward distance is more than 30cm which means that there is no object near our car within 30cm range which means that the path is clear for our car to move forward. Second scenario: is that the forward distance is less than 30 now the car moves backward and stops.

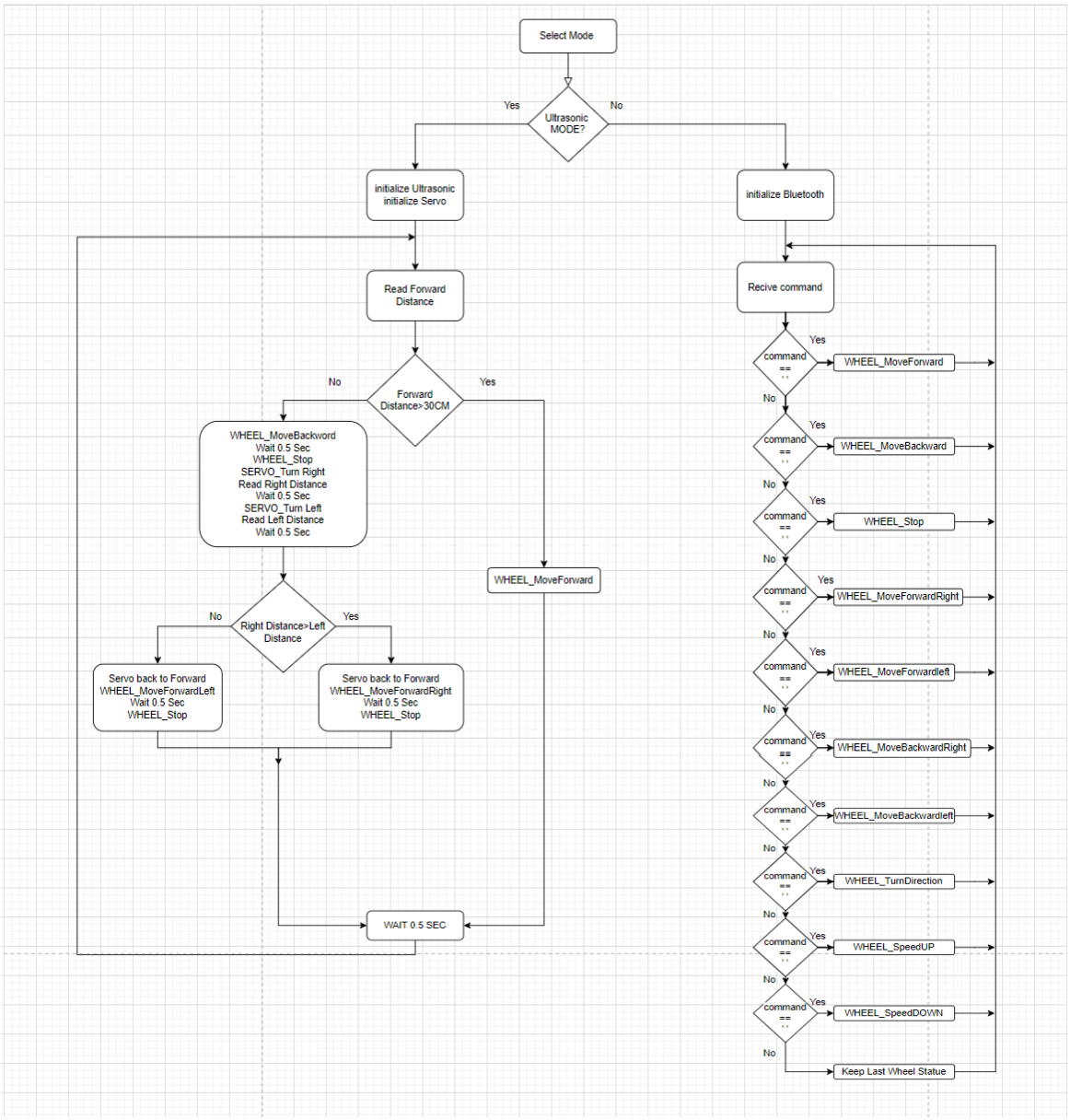
4) Now the we use the servo to move the Ultrasonic left and right and read every distance in the left and right

5) we compare both left and right distance and choose the clearer path which has bigger distance read. And make the car move there using the motors functions.

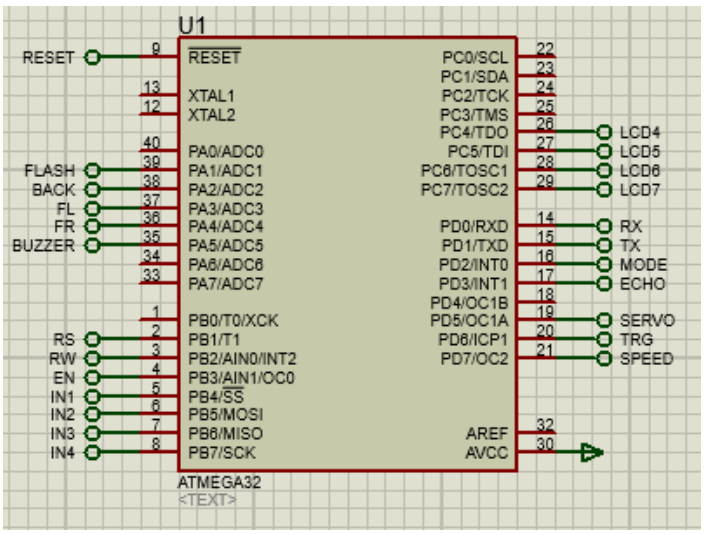
6) lets jump back to step 2 and let the user choose the Bluetooth mode now we will make the needed initialization to get the Bluetooth ready and start sending characters which are configured to make certain movement from the car.

7) and that character will be put in a switch case which will make the movement according to the character send using the needed motor functions.

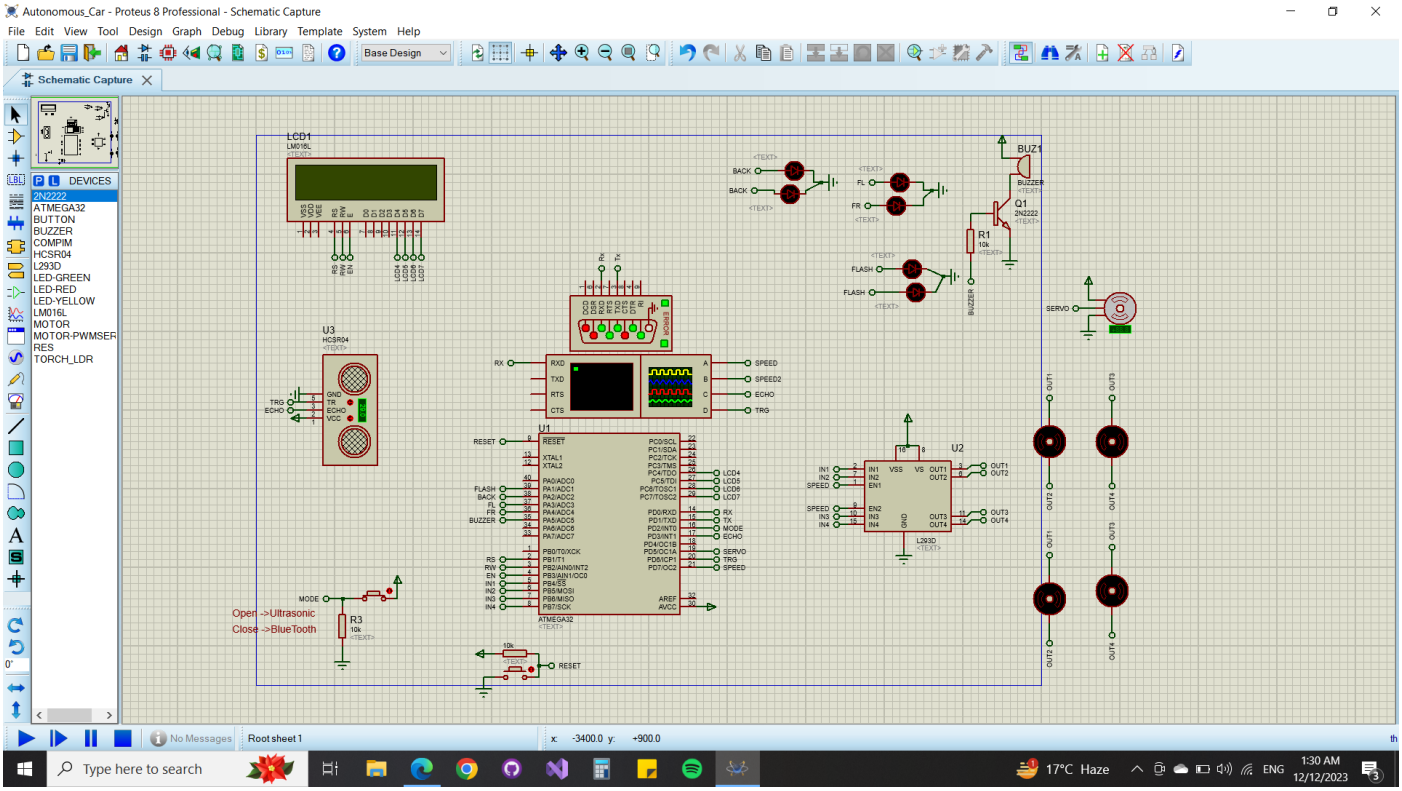
And here is the Application flow chart diagram:



III) Pins configuration:



IV) Simulation design using proteus:



V) Wheel driver:

We need to make our car capable of moving in every possible direction but since we have only 1 H bridge IC which has only 2 H bridge circuits and 4 wheels with 4 DC motors. We decided to connect the motors in the left direction together on the first H-bridge and the other 2 motors on the right will be connected to the other H bridge, so that we can control each 2 motors together, and that will make us able to move our car in all the directions possible using the 4 inputs of the 2 H-bridges.

Wheels connection:

***A) Atmega32 and H-bridge IC connection*:**

1) power connections: Each bridge high pin will be connected into the 9-12V voltage, while the 2 other ground pins of each bridge will be connected to our ground bus.

2) control pins: for each bridge we have 1 enable pin which will be connected to the PWM signal that is coming from Timer 2, and 2 inputs pins that will be controlled using the DIO peripheral.

B) H-bridge IC and 4 DC motors connection:

high Output from Hbridge1 will be connected to both high wires of motors on the left.
Low Output from Hbridge1 will be connected to both low wires of motors on the left.
and same for other 2 motors in the right with H bridge2.

Interface functions:

1) void WHEEL_Init();
2) void WHEEL_MoveForward ();
3) void WHEEL_MoveBackward ();
4) void WHEEL_Stop ();
5) void WHEEL_MoveForwardRight ();
6) void WHEEL_MoveForwardleft ();
7) void WHEEL_MoveBackwardRight ();
8) void WHEEL_MoveBackwardleft ();
9) void WHEEL_SendDutyCycleAndStart(u8 speed);

Implementation ideas:

1) void WHEEL_Init();
>>This function helps in setting the direction of the DIO pins that are chosen to be input 1 for left and right bridges and input 2 for left and right bridges and the PWM signals that are sent to enable pins of the 2 h-bridges we are having and this is done using the DIO functions.

2) void WHEEL_MoveForward ()
>>Enables the motors, it sets a configurable pin to high “1”, which typically turns on the motors.
>>Setting the left motors to rotate in a forward direction by choosing the input1 right pin into 0 and input2 right pin into “1”.
>>Setting the left motors to rotate in a forward direction by choosing the input1 left pin into 0 and input left into “1”

3) void WHEEL_MoveBackward ()
>>Enables the motors
>>Makes the left motor rotate in the backward direction, resulting in backward movement done by the left motors, and that’s done by choosing the inputs of the left bridge opposite to the forward ones.
>>it also configures the right motor for backward movement. Similar to the left motor, by setting **MOTOR_RIGHTINPUT1** to high and **MOTOR_RIGHTINPUT2** to low this makes the right motors rotate in the backward direction, achieving backward movement to the whole car.

4) void WHEEL_MoveForwardRight ()
>>Enables the motors
>>Setting **MOTOR_LEFTINPUT1** to low and **MOTOR_LEFTINPUT2** to high to make the left motor rotate in one direction, achieving forward movement.”
>>it also configures the right motor to stop. Both **MOTOR_RIGHTINPUT1** and **MOTOR_RIGHTINPUT2** are set to low, which typically stops the right motor.”
Summary, the **WHEEL_MoveForwardRight** function is configuring the pins associated with motor control to make the left wheels move forward while the right wheels stop. This configuration makes the car turn to the right.

---

5) void WHEEL_MoveForwardleft()
>>To enable the motors
>>Both **MOTOR_LEFTINPUT1** and **MOTOR_LEFTINPUT2** are set to low, which typically stops the left motors."
>>it also configures the right motor for forward movement,by setting **MOTOR_RIGHTINPUT1** to low and **MOTOR_RIGHTINPUT2** to high to make the right motors rotate in one direction, resulting in forward movement."
Summary, the **WHEEL_MoveForwardLeft** function is configuring the pins associated with motor control to make the right wheels move forward while the left wheels stop. This configuration makes the car turn to the left.

6) void WHEEL_MoveBackwardleft ()
>>to enable the motors
>>Setting both **MOTOR_LEFTINPUT1** and **MOTOR_LEFTINPUT2** to low to make the left motor Stop.
>>it also configures the right motor for backward movement , by setting **MOTOR_RIGHTINPUT1** to high and **MOTOR_RIGHTINPUT2** to low to make the right motor rotate in one direction, resulting in backward movement.
>>The **WHEEL_MoveBackwardLeft** function is configuring the pins associated with motor control to make the left wheels stop while the right wheels move backward. This configuration makes the car turn to the left while moving backward.

7) void WHEEL_MoveBackwardRight ()
>>To enable the motors
>>this configures the left motor for backward movement , by setting **MOTOR_LEFTINPUT1** into high and **MOTOR_LEFTINPUT2** to low to make the left motor rotate in the opposite direction, resulting backward right movement.
>>Setting **MOTOR_RIGHTINPUT1** to Low and **MOTOR_RIGHTINPUT2** to Low make the right motors stop.
>>so that the car will move in backward right direction.

8) void WHEEL_Stop();
>>in this function we use the DIO functions to put low "0" values at our 4 inputs to force the motors to not rotate so that the car will stop.

9) void WHEEL_SendDutyCycleAndStart(u8 speed);
>> This function initialize Timer 2 and then sent the duty percentage of the PWM as needed to change the speed of the motors using TMR2_SetDutyCycle_FastPWM(Speed); then call the TMR2_Start();

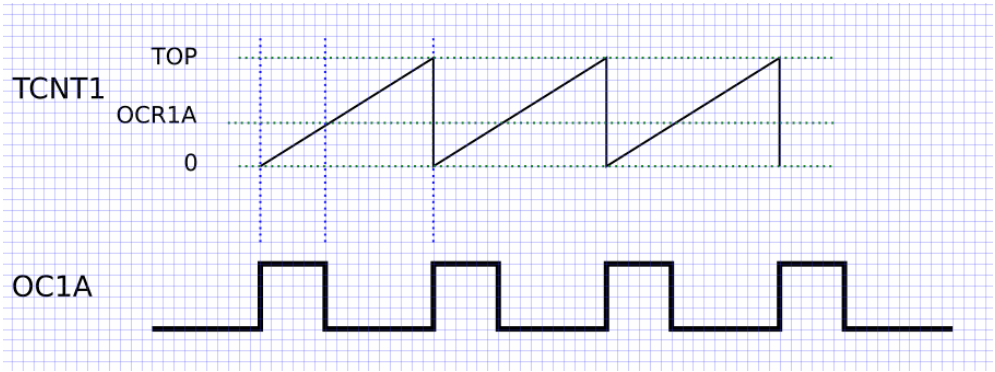
VI) Servo motor "SG90":

"SG90" can give any **angle ranges** from 0 up to 180.

1) The main idea of interfacing with the Servo motor is sending to it a PWM signal as a control signal. At which the angle of the servo will change according to the duty percentage and frequency of the PWM sent to it.

1.2) But how will we produce that PWM signal?

Since we are using ATmega32 as our microcontroller therefore we can produce PWM signals using Timer/counters, at their fast PWM mode. But we need to be able to control its frequency so Timer0 is excluded because it has a fixed top on the other hand Timer1 has a variable top in mode 14 and two compare match registers. So we are going to use Timer1 and mode 14 to send the PWM with the needed frequency and duty.



In mode 14, The top is variable and will be addressed as ICR1A.

---

2)Interface functions And Their implementation:

2.1) void SERVO_init(void);
This function will do the needed initializations.
2.2) void SERVO_Left(void);
This function is responsible for making the most anti-clockwise angle. (Angle = 0)
2.3) void SERVO_Right(void);
This function is responsible for making the most clockwise angle. (Angle = 180)
2.4) void SERVO_Forward(void);
This function is responsible for making the forward angle. (Angle = 90)
2.5) void SERVO_Anyangle(u8 AnyAngleindegrees);
This function will be used to get any angle in general.

Ideas of Implementation:

for 2.1 function: we need to
1) Call the TMR1_init()
2) Set the Servo pin direction which will receive the PWM
3) TMR1_start()
so that the Servo is ready to work at any angle.

for 2.2 , 2.3, 2.4, 2.5:
So after doing the math we deduced that:
knowing that the prescaler is equal to 64
Frequency = 1 / ((ICR1A+1) * 4microseconds)
Duty = (OCR1A / ICR1A) * 100

and from research we got the relation between the needed angle ,frequency and duty:
Frequency = 50hz
duty = (angle/18) + 2

So now we can call the Timer1 function that will update the frequency and duty by changing the ICR1A and OCR1A as needed:
TMR1_ModeFastPWM(f32 duty , f32 Frequency);

For example:
In Servo_left(); we will call the TMR1_ModeFastPWM(2 , 50); . and same for function 2.3, 2.4, 2.5 but with the change of the parameters of Timer1 function.

VII) Ultrasonic sensor "HC-SR04":

•Ultrasonic is used to measure the distance throughout put a high signal on trigger pin for 10 microseconds then it sends 8 pulses in the air and calculate the time, that the signal takes to get back to the ultrasonic receiver.

• We created some global variables that will help with calculating the distance like:

1. u8 sensor_working = 0; // as a flag for knowing if the sensor is working
2. u8 rising_edge = 0; // as a flag to know if echo is high
3. u32 timer_counter = 0; // to count number of overflows done by timer0
4. f64 distance = 0; // to save the calculated distance in it.

We also created these functions to help us with calculating the distance:
1. void ULTRASOIC_init(void)
2. void ULTRASOIC_GetDistance(f64* DistanceValue)
3. Timer0 interrupt __vector_11(void) “OVF”
4. external interrupt __vector_2(void)

Function 1: void ULTRASOIC_init(void)
>>This function sets the direction of the Trigger as output and the Echo pin as input using the DIO.
>>It enables the external enable and setting it to any logical change using that function EXTI_Enable(EXTI_INT1,EXTI_ANY_LOGICAL_CHANGE);
>>it also initializes timer0 and start it using timer0 functions.

Function 2: external interrupt __vector_2(void)
>> this is the function that is responsible to react to the echo signal sudden change and start calculating the distance and that’s done by making the EXTI_INT1 that’s is passed to the interrupt be equal to the echo pin so that whenever a sudden change happens in the echo it will jump into that interrupt.
>>Inside this interrupt we check if the sensor is working by our flag “sensor_working” which means that a trigger had been sent.
>>if sensor is working we check if it is the rising edge or the falling edge of the echo by the flag “rising edge” , if it is the rising edge we will set the timer0 counter into 0 by using “TCNT0 = 0x00”.
>>but if it was the falling edge we then start calculating the distance and store it in our global variable “Distance” using the values stored in TCNT0 knowing that we have calculated the total numbers of overflows.
>>then we set rising edge and timer_counter to zero again.

```
distance=(343*(timer_counter*256+TCNT0)/320000)+1;
```

Function 3: Timer0 interrupt __vector_11(void)
>>This function is responsible for incrementing and handling the number of overflows and assigning it to our global variable “timer_counter”.

Function 4: void ULTRASOIC_GetDistance(f64* DistanceValue)
>>This functions checks at first if the sensor is not working and if condition is true.
>>it will start sending a pulse of 15microseconds on the trigger pin using the DIO.
>>which means that now the sensor is working so we need to set sensor_working to “1”
>>And now the signals had been sent and echo is being handled through the external interrupt and we got the distance calculated and stored back to our global variable “distance”
>>So at the end of the function we will assign the *DistanceValue into distance to store the calculated distance.

VIII) Bluetooth "HC-05":

we will use the UART communication protocol to transmit and receive data between the user and into our controller.

ATmega32 – "HC-05" connections:

Pin 1 into power voltage bus
pin 4 grounded
pin 2 ,3 Rx and Tx from the MC Rx and Tx pins into the Bluetooth Rx Tx pins.

Interface Functions:

1) void BLUETOOTH_init(void);
2) void BLUETOOTH_TransimitChar (u8 TransmitData);
3) void BLUETOOTH_TransimitString (u8* TransmitData);
4) void BLUETOOTH_ReceiveChar(u8* ReceiveData);

functions Implementations:

1) void BLUETOOTH_init(void)
>> This function is responsible for initializing the UART

2) void BLUETOOTH_TransmitChar(u8 TransmitData)
>> This function is responsible for transmitting char using the UART_TransmitChar(u8 TransmitData)

3) void BLUETOOTH_ReceieveChar(u8* ReceiveData)
>> This function is responsible for Receiving data using the UART_ReceiveChar(u8 TransmitData)

4) void BLUETOOTH_TrabsmiteString(u8* Transmitidata)
>> This function is responsible for Transmitting strings using the UART_TransmiteString(u8 TransmitiData)

IX) Buzzer:

We will use the buzzer to buzz when the car is moving backwards or stopping. We will choose 1 pin of the MC pins to connect to the buzzer and control it using the DIO peripheral.

interface functions:

1) void BUZZER_init (u8 BuzzerPort, u8 BuzzerPin);
2) void BUZZER_TurnOn (u8 BuzzerPort, u8 BuzzerPin);
3) void BUZZER_TunrOff (u8 BuzzerPort, u8 BuzzerPin);
4) void BUZZER_Toggle (u8 BuzzerPort, u8 BuzzerPin);

Implementation:
1) void BUZZER_init (u8 BuzzerPort, u8 BuzzerPin);
>>This function will set the direction of the pin as output using the Dio functions.

2) void BUZZER_TurnOn (u8 BuzzerPort, u8 BuzzerPin);
>>This function will set the value of the pin as high “1” using the Dio functions.

3) void BUZZER_TunrOff (u8 BuzzerPort, u8 BuzzerPin);
>>This function will set the value of the pin as low “0” using the Dio functions.

4) void BUZZER_Toggle (u8 BuzzerPort, u8 BuzzerPin);
>>This function will toogle the value of that pin using the Dio functions.

X) LEDS:

We have 6 LEDS that we will use as a response according to the car movements.

ATmega32 - LEDS connections:
1) 2 LEDS connected together into a configurable pin “flash”
2) 2 LEDS connected together into a configurable pin “Back”
3) 1 LED connected alone into a configurable pin “FR”
4) 1 LED connected alone into a configurable pin “FL”

Functionality:
When the car is going back the “Back” LEDS will turn on. And If car is moving forward right the “FR” pin will turn on. And if the car is moving forward left “FL” will turn on , and that will be done using the DIO peripheral.

Interface Functions:
1) void LED_init(u8 LedPort,u8 LedPin);
2) void LED_TurnON(u8 LedPort,u8 LedPin);

Implementation:
1) void LED_init(u8 LedPort,u8 LedPin);
>>This function has conditional statement that checks whether the provided “**LedPort”** is less than or equal to 3 and “**LedPin**” is less than or equal to 7. This check ensures that the provided port and pin values are within valid ranges for the the ATmega32 pins.
>>If the condition is true:
a function **DIO_SetPinDirection** to set the direction of the LED pin as an output.

2) void LED_TurnON(u8 LedPort,u8 LedPin);
>>The same for the “conditional if” part, the **LED_TurnON** function is designed to turn on an LED by setting the LED pin to a high “1”using the DIO functions.
