# Microprocessor Systems and Interfacing

## MINI PROJECT 3


| | |
|---|---|
| Group Members Name & Reg #: | **Abdul Ahad** **(FA18-BCE-003)** **Muhammad Haris Irfan** **(FA18-BCE-090)** |
| Class | Microprocessor Systems and Interfacing CPE342 (**BCE-6B**) |
| Instructor's Name | Dr. Omer Ahmed |

–

## This Document contains the following content:

- Design Logic.
- Simulation Explanation.
- Additional learning that helped.
- OCR1A Calculations.
- Problems faced.

## Part 1: DESIGN LOGIC

The code that forms the basis of our implementation works by making use of a Servo motor and three LDRs (in the final version more might get added to make detection go more smoothly).

One LDR is placed right above the dinosaur's head and is used for detecting the background (black or white),the other two LDR's are placed on the screen such that they detect the obstacles in the dinosaur's way and on detecting the make it avoid them (by jumping) by turning on the servo motor- when an obstacle is detected, the servo motor moves from its initial position and presses the button, after a very small delay it releases the button, hence coming back to its initial position.

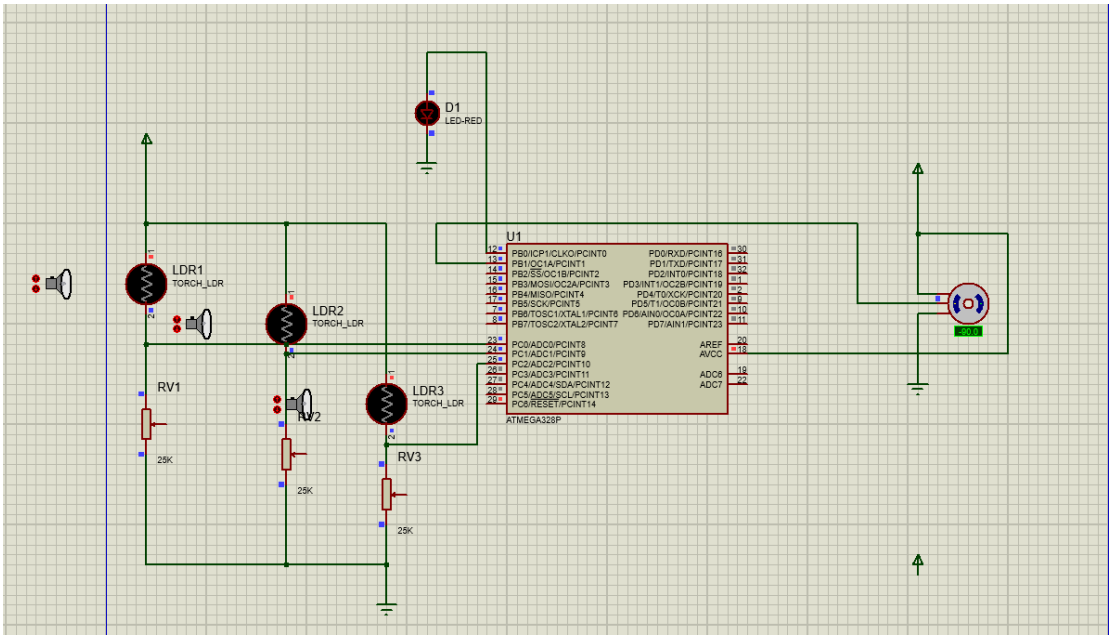
**NOTE:** we are making use of only 1 servo motor as in the initial testing phase we noticed that only jumping is sufficient to overcome the obstacles up until 1800 points.

## Part 2: SIMULATION EXPLANATION

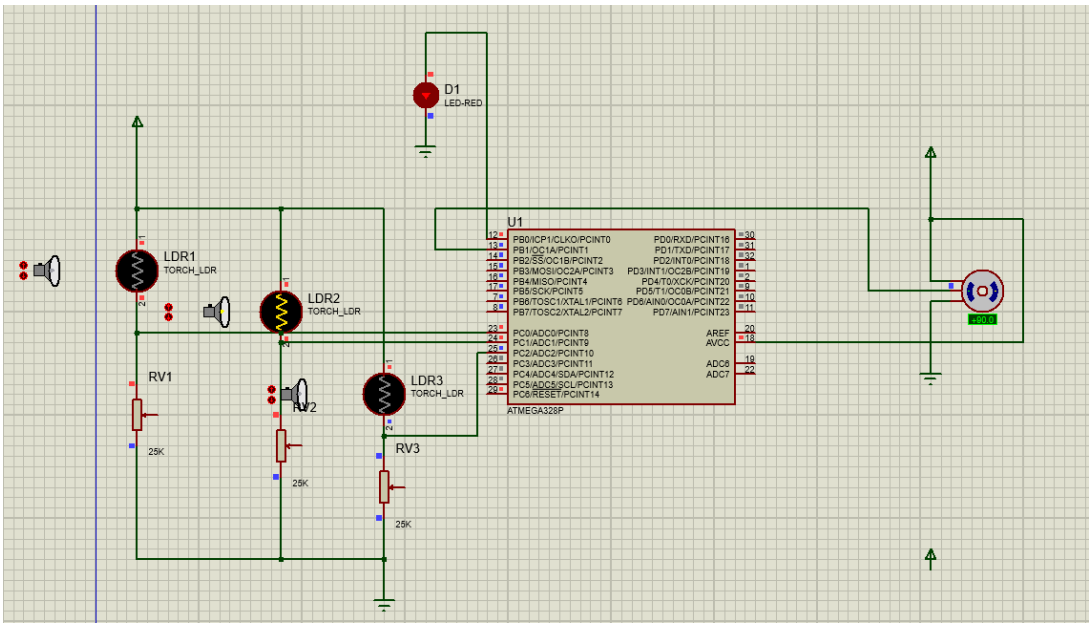
The basic working is explained below using schematic pictures, but in project 3 we added additional LDR's.

Two additional LDR's were used for speed detection.

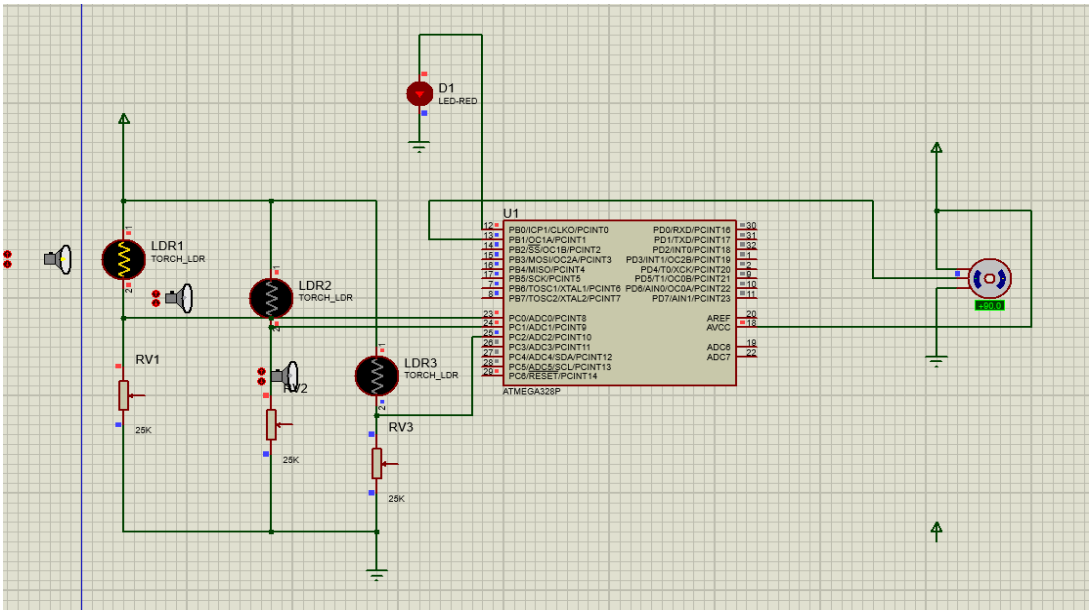
Here LDR3 is used for detecting the background color while LDR1 and LDR2 are used for detecting the obstacles. D1 LED is used for testing purposes.



Here LDR3 is dark which means the background is black and LDR 1 and LDR 2 are black show that there is no obstacle detection so far.




Here LDR3 again shows the background is dark while LDR2 is turned on which shows an obstacle was detected hence the testing LED turned on and the motor rotated from -90 to +90 degrees (in simulation)



Similarly, LDR3 again shows the background is dark while LDR1 is turned on which shows an obstacle was again detected hence the testing LED turned itself on and the motor rotated from -90 to +90 degrees.

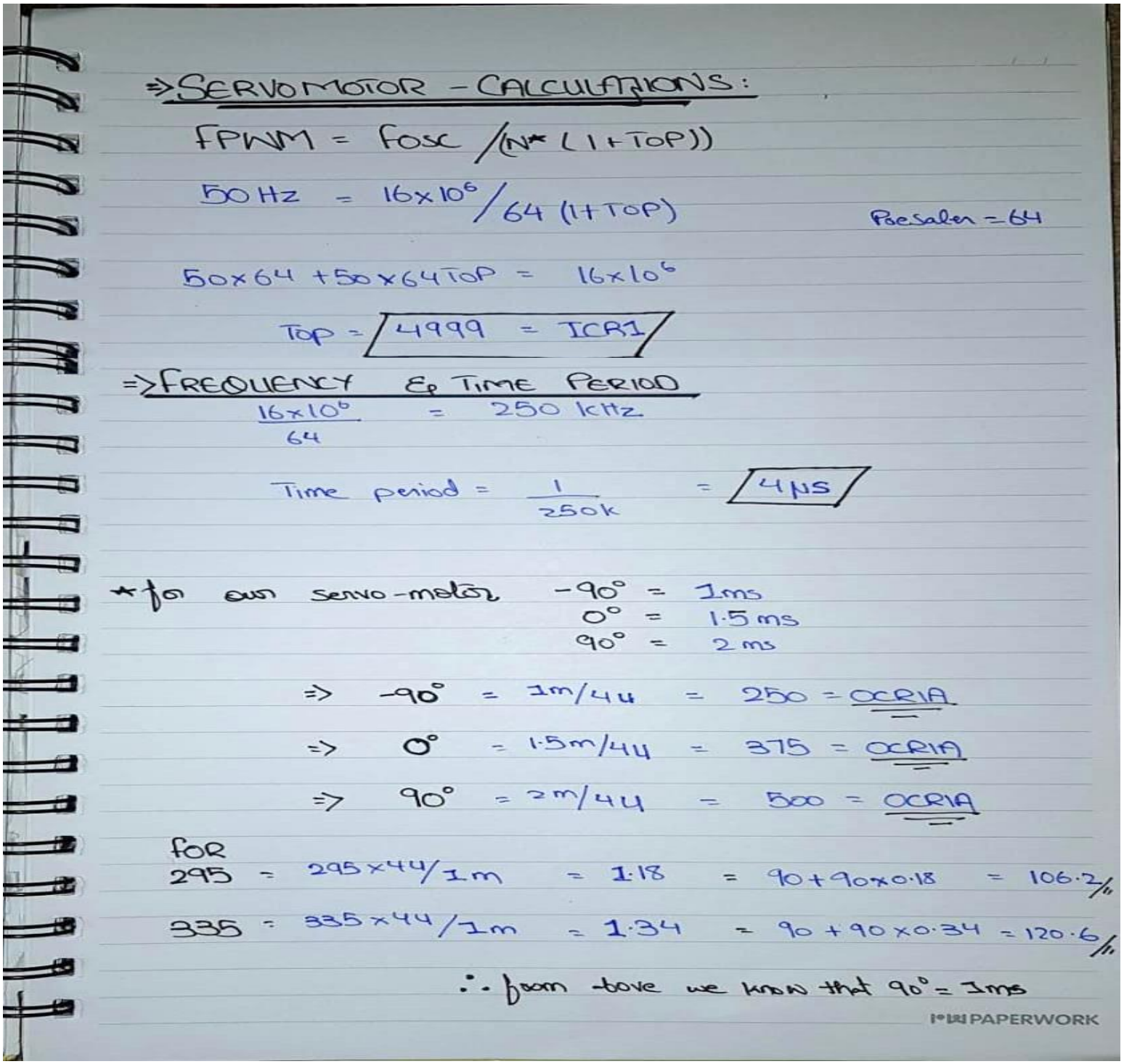
**NOTE:** when the background is light the simulation works on reverse logic all the LDRs remain ON at all times and as soon as LDR 1 or LDR 2 turns off (detects an obstacle) the testing LED turns on and the motor rotates.

## Part 3: Additional Learning that helped

During testing we were faced with issues while reading the correct value from the LDR, but then we learnt that we could burn the Microchip code on Arduino IDE and use the serial monitor of Arduino ide to our advantage.

This one thing helped us a lot as we were able to see what exact values our LDR was reading due to different intensity of light.

## Part 4: ICR1/OCR1 calculations



⇒ SERVOMOTOR – CALCULATIONS:

$f_{PWM} = f_{osc} / (N * (1 + TOP))$

$50\ Hz = 16 \times 10^6 / 64\ (1 + TOP)$ Prescaler = 64

$50 \times 64 + 50 \times 64\ TOP = 16 \times 10^6$

$TOP = 4999 = ICR1$

⇒ FREQUENCY & TIME PERIOD

$\frac{16 \times 10^6}{64} = 250\ kHz$

Time period $= \frac{1}{250k} = 4\ \mu s$

* for our servo-motor $-90° = 1\ ms$, $0° = 1.5\ ms$, $90° = 2\ ms$

⇒ $-90° = 1m/4 = 250 =$ OCR1A

⇒ $0° = 1.5m/4 = 375 =$ OCR1A

⇒ $90° = 2m/4 = 500 =$ OCR1A

for

$295 = 295 \times 4/1m = 1.18 = 90 + 90 \times 0.18 = 106.2°$

$335 = 335 \times 4/1m = 1.34 = 90 + 90 \times 0.34 = 120.6°$

∴ from above we know that $90° = 1ms$

PAPERWORK

**The above calculated values were used in our code.**

## Part 5: Problems Faced

We faced problems in several faces, they are given in detail below:

1- **SENSOR CALIBRATION**
2- **SENSOR OPTIMISATION**
3- **SPEED SENSING**
4- **MODE TRANSITION**

## 1- SENSOR CALIBRATION

An issue identified in the first phase of the project was that each time the project module was run the values received from LDR's changed even if the conditions were kept the same slight fluctuations were noticed which were significant enough to cause errors in obstacle detection.

To solve this problem, we devised a solution which automatically takes 100 values for each of the sensors every time the code is run it then averages them out and sets the obstacle detection values accordingly.

To implement this each time the module is taken into a different environment the sensors are placed for a brief time on dark and light versions of the screen, so they configure themselves.

## Piece of code for calibration

## Prompt to show White screen

```
for(int i=0 ; i<100 ; i++)
    {
        temp1 = temp1 + adc_read(0);
        temp2 = temp2 + adc_read(1);
  Serial.println("Go to white");
        //// calibration for bright areas
       _delay_ms(10);

    }
    temp1=temp1/100;
    temp2=temp2/100;
    temp1=temp1;
    temp2=temp2-10;
```

## Prompt to show black screen

```
  for(int k=0;k<100;k++)
   {
   Serial.println("Go to black");
    _delay_ms(10);
    }
   darkval1 = adc_read(0);
   darkval2 = adc_read(1);

    for(int i=0 ; i<100 ; i++)
    {
        darktemp1 = darktemp1 + adc_read(0);
        darktemp2 = darktemp2 + adc_read(1);
        //// calibration for bright areas
       _delay_ms(10);

    }
    darktemp1=darktemp1/100;
    darktemp2=darktemp2/100;
    darktemp1=darktemp1;
    darktemp2=darktemp2;
```

## 2- SENSOR OPTIMISATION

To optimize the values of the sensors so major continuous fluctuations are not received shielding of the sensors was implemented which further streamlined the values and made obstacle detection go smoother.

## 3- SPEED SENSING

To implement speed sensing three different approaches were considered, all of these approaches used the same number of timers but what varied was the amount of LDR's and the positioning of LDR's

- The first approach made use of two LDRs and only timer2 the purpose of this was to get an idea on how to use timers and in what settings the timers were to be used in.
- In the next approach Three LDRs were placed on the screen and this approach made use of two timers (Timer0 and Timer2 both working in normal mode). The Basic idea was that when on the furthest LDR from the dino an obstacle is detected Timer2 is started and till that obstacle is not detected on the center LDR the overflow count of the timer is being recorded. Then when the obstacle is detected on the closest LDR from the Dino Timer0 is started and when its overflow count becomes equal to the overflow count recorded from Timer2 a jump is performed. (The finalized version of his algorithm would have required 6 LDRs and thus it was scrapped)
- Due to the limited number of Analog pins (A0-A5: 6 pins) no more than 5 LDRs could be assigned for obstacle detection as 1 LDR was required for Mode

detection. Hence the previous algorithm with its need for 6 LDRs for obstacle was not viable. To go around this an approach which used 4 LDRS was considered: when the first pair of LDRs detected an obstacle Timer2 was started when the second pair of LDRs detected the same obstacle Timer2 was stopped its Overflow count stored and Timer0 was started and when Timer0's overflow count became equal to that of Timer2 a jump was performed.

Other issues that were addressed in this version of code was handling more than one object in a given time by the use of an array and by successfully handling the garbage value problem in filling of the arrays (as the speed of processing of statements is way faster than the time an obstacle passes under the LDR hence garbage values were being stored in the array in previous versions).

- Below is an example of our test values.

```
23:53:13.546 -> 19:   0
23:53:13.786 -> ---------FIRST SENSOR
23:53:13.818 -> 735--------SECOND SENSOR
23:53:14.055 -> 739 timer_overflowcount
23:53:14.090 -> 250
23:53:14.090 ->  timer_overflowcount
23:53:14.124 -> 262
23:53:14.467 -> ---------THIRD SENSOR
23:53:14.502 -> 769
23:53:14.502 -> array values
23:53:14.502 -> 0:   63160
23:53:14.536 -> 1:   64164
23:53:14.536 -> 2:   0
```

## 4- MODE TRANSITION

One of the major hurdles to tackle was for the sensors to smoothly transition from sensing dark obstacles to the light obstacles this was especially hard as during the brief transition period the constant flickering from dark to light and vice versa triggered most of the 'if' conditions making the dinosaur jump at irregular intervals.

To counter this, 2 approaches were considered.

- First approach.

To shift the way our sensors detect

In all the instances of our code before, certain values of LDR's were considered and when a value beneath or above them was received (dark or light mode) a jump was performed.

*For example, say LDR 3 is used to detect the mode, whereas LDRs 2 and 1 detect obstacles.*

*When LDR3 detects a value less than 200 then the mode is dark and LDR's 1 and 2 jump when values are greater than 200 (light obstacles) and the opposite for Light mode LDR3 detects a value greater than 500 and jumps occur when LDRs 1 & 2 detect values less than 300.*

the above example functions on the raw values received from the LDRs and require 2 sets of instructions (one for each mode) and using this way transition period was giving errors.

- To counter this a new approach was considered which significantly decreased the number of statements as

  1- different statements to recognize the mode were not required

  2- jumps were to be triggered not using the raw values from LDRs but by computing the difference between these raw values.

To implement this, 3 steps were to be taken before starting the game.

1- Use of variable resistors so that all the LDRs give similar values for similar intensities.
2- Placing the LDRs such that one LDR is at a place where there are no obstacles and others for obstacles (LDR3 above all obstacles and LDRs 1 and 2 for obstacles)
3- Calculating and storing the difference values between LDR3 and LDR1 and LDR2 respectively for all modes (when LDR3 is light, and the others are dark and vice versa)

The Scenario will work as follows:

- Say the mode is Dark then LDR3 will detect a value of 200, and when there are no obstacles.

  LDRs 1 & 2 will also detect a value of 200 (LDR3 – LDR2= 0)
- Now say the mode is again dark LDR3 = 200 but LDR2 detects an obstacle LDR2 = 400 (LDR3-LDR2 = -200)

- Now reversing the modes LDR3 is light hence LDR3= 400 and no obstacle is detected LDR2 = 400 as well (LDR3-LDR2 = 0)
- LDR3 is again light hence LDR3= 400 and obstacle is detected LDR2 = 200 (LDR3-LDR2 = 200)

Thus, based on this, a simple statement worked for all modes even with the transition and this was a major breakthrough.

**NOTE:** *The project CODEs are also shared in the file.*