

Bangladesh University of Engineering & Technology

Course : EEE-304 (Digital Electronics Laboratory)

PROJECT REPORT

***PROJECT Title: Snake Game Implementation In Verilog And
Hardware simulation In Proteus***

Name: Shahedul Hasan

Student's ID: 1606179

Name: Yousuf Ahmed

Student's ID: 1606181

Name: Rayhan Al MAhmud Rifat

Student's ID: 1606185

Department: EEE

Section: C2

Level: III Term: II

Date of submission: 22-12-2020

Drive

Link: <https://drive.google.com/drive/folders/1OU3Lh0PzvAmpLHEELtbSdQhDN6kpyUSZ?usp=sharing>

Introduction

Our project involves a virtual snake and the object of the game is to be able to eat as much food as possible, if the snake collides with boundary of the LED matrix, the Game Over will be displayed .

For the following project, we decided upon implementing a game we used Verilog coding and Hardware simulation in proteus using Arduino board.

we had decided to use a 8 by 8 LED Matrix display for displaying the output.

In order to implement the snake game, we divided the project into four phases as mentioned earlier. These are:

- Creating the Virtual Snake
- Generating Random food
- Adding the constraints to the game
- Figuring out the working of the LED Matrix and the Shift register

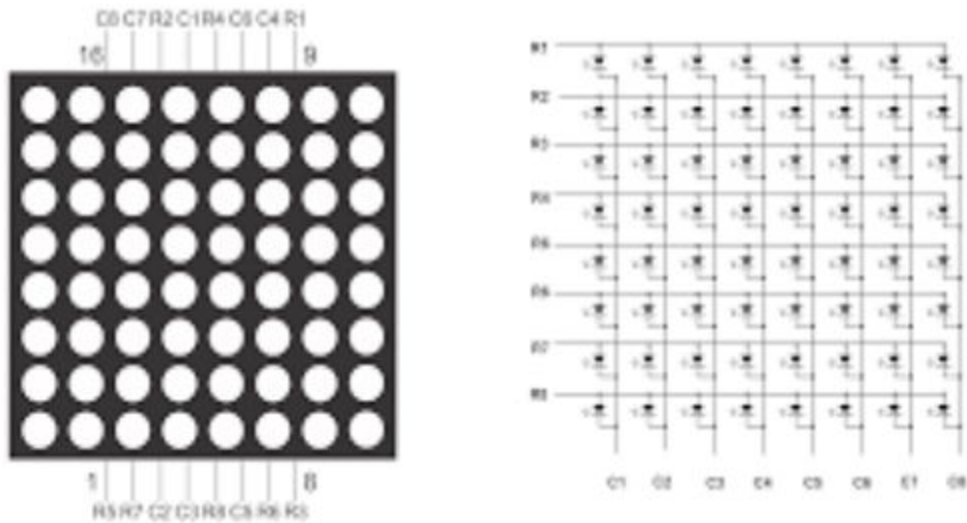
Through deep understanding and proper implementation of all four steps, we were successful in being able to create the classical Snake Game using Aduino In proteus simulation.

Procedure

In order to go about creating the game, our initial step was about displaying of the snake using the **LED matrix**

Display Using the LED Matrix

The LED Matrix we had decided to be used for the project is an 8X8 matrix. The pin configuration of this matrix is as the following.#



The LED Matrix has 16 pins in total, 8 anodes and 8 cathodes shorted accordingly. Individually controlling each and every LED on the Matrix is not possible, due to the way it is manufactured, however, each row or column can be set individually. Hence in order to display on the LED matrix, each row or column must be displayed faster than the human eye can perceive the change, several times a second so that the human eye is fooled into believing that all of them are glowing at the same time, which on the contrary, are not.

In order to do this, we assigned a 2D matrix register of the size 8X8 with row numbers decreasing from up to down and column numbers decreasing from left to right, so that each individual cell of the register represents each LED on the matrix. That way whenever we want an LED to glow we assign it a 1, and if we don't want it to glow we assign it a 0.

For example, we assign

reg [7:0]

```
display [7:0];                                //now if we want the LED on the //fourth  
column(from right) and third row(from bottom)  
//to glow wes say,
```

```
a[2][3] = 1;                                 //( since assignment starts from 0 )
```

And now in order to display this matrix, we display each row or column multiple times in a second so that it seems that all the rows are lit at the same time. In order to this, we assign another 1D register which represents each of the cathodes of the LED Matrix. We then run a for loop for each of the columns, where for the respected column, the respected cathode is turned to 0 so that it's in forward bias, while the others are turned to 1 so that they don't glow and put into reverse bias. The anodes are given the configuration of the respected column in the 2D matrix register.

```
Reg [7:0] column;
```

```
Integer i;
```

```
Intial begin column = 8'b11111111; end
```

```
for ( i = 0; i < 8 ; i = i + 1) begin
```

```
    column[i] = 0;
```

```
    output = display[i]
```

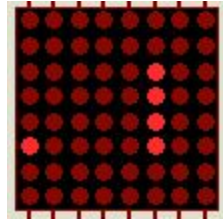
```
End
```

This way with the help of the edge of the clock, we can display each row for about a millisecond, which is enough for the human eye to be tricked.

Creating the Virtual Snake

The Snake we had decided to make would have the following features,

- The Snake would be represented by a combination of LEDs on the LED Matrix, all in a single line, either horizontally or vertically, depending on the game setting at that moment. It can be seen in the following picture.



- The Snake is to be controlled using four input buttons left, right, up, down, which help in the direction change function.

- It is also to be noted that as the snake cannot cross the boundary of the LED matrix.

For coding we use Verilog coding in Quartus, Also we wrote an arduino Code, create a .hex file so that it can be dumped onto the Arduino Uno.

As the snake we will be creating will have a fixed length of 2 units, //we will assign a 2D matrix register, the number of rows being 2, which correspond to the X coordinate and the Y coordinate and the number of columns corresponding to the length of the snake.//

reg [1:0] snake [n-1:0], where n is the length of the snake(3).

Each of these registers will hold the coordinates of all the parts of the snake at any point in time. These are then copied respectively into the LED Matrix display logic and printed.

The next task is the movement of the snake. In order to do this, we define four states of snake movement each for up, down, left and right. We then each state separately as the following

Up(2'b00) – each of the Y coordinates is incremented by 1 (as the Y coordinates decrease from top to bottom and we take care that after it reaches the maximum value of 7, it becomes 0).

Down(2'b01) – each of the Y coordinates is decremented by 1 and we take care that after the value reaches the minimum value of 0, it becomes 7.

Left(2'b10) – each of the X coordinates is incremented by 1 (as the X coordinate decreases from left to right) and we take care that after it reaches the maximum value of 7, it becomes 0.

Right(2'b11) – each of the X coordinates is decremented by 1 and we take care that after the value reaches the minimum value of 0, it becomes 7.

We now we look at the third and the final condition, that prevents the user from giving an input which will make the snake overlap with itself. This happens when the user inputs down when the snake is moving up and the vice-versa and when the snake is left and the user inputs right and the vice-versa. For this, we had an additional when changing states.

For example –

If (input = 2'b00(Up) && state != 2'b01(Down))

we have now finally completed creating the virtual snake and the necessary features in it with it. We now move on to creating a program to generate random food.

Generating Random Food

Due to the inability to create a truly random function on hardware, we hence use our own random function in order to place the food at a random place after every iteration. In order to do to this, we create 64 variables, each corresponding to each position of the LED Matrix. We then randomly map these to a time variable.

For example,

T34 = matrix[2][4];

T35 = matrix[6][0]; and so on...

Based on this, we combine it with the time logic to provide the food

Adding Constraints to the Game

We now move to adding the final three constraints of the game. They are:

- Creating the Score logic - making sure the score increases after the user is successfully able to make the snake eat the food.
- Adding the timer – to reset the game after the time for playing the game is over ie, creating a time limit for the game.
- The Crash Logic – making the game end after the user mistakenly makes the snake overlap with itself during the course of the game, causing the game to end.

To create the **Scoring logic** of the game, we basically need to see if the head of the snake coincides with the food coordinates on the LED Matrix as this will signify that the snake has eaten the food. We hence use a simple if condition for this

At first we need to define the head coordinates for the snake, which is simply given by

head_x = snake[0][0];

head_y = snake[0][1];

we then implement the if condition by the following code.

If (food_x == head_x && food_y == head_y)

score = score + 1

reg [3:0] score_tens; and

reg [3:0] score_units;

Now we add the **Timer Logic** which is also a simple step, by simply setting the timer to initialize to a value, let's say 60 seconds and then counting down till it reaches 0. After 0 the score then resets to 0, for the user to play the next game. Similar to how we divide score into 2 registers of 4bits each in order to display them, we divide the timer into two parts.

Reg [3:0] timer_tens;

Reg [3:0] timer_units;

And for the score to reset to 0, we define a simple logic for it

If (timer == 0)

score = 0;

However, we have another thing to code, which we do because of the fact that we chose the timer and score values to be represented by two registers of 4bits, each representing the tens and units digit of the variable respectively, we now must additionally write the increment /decrement logic of the following.

For this we had coded it, to make sure the bits change after the units digits reach 9 ie,

For the increment logic (for the example of score logic)

after score_units == 9

score_units = 0; score_tens = score_tens + 1;

similarly for the decrement logic (for the example of time logic)

after timer_units == 0

timer_units = 9; timer_tens = timer_tens - 1;

Now we add the **Time Logic** to the game and this we incorporate with the food logic. In order to incorporate the food logic into the separate, we create a separate food counter or the actual timer of the game, which tracks the time from when the game has initialized. We make sure that this timer resets after every 64 ticks, the number of positions presents on the LED logic. We then map the food counter/actual counter to the random function of the food and code it such that the time at which the previous food is eaten determines where the next food will be placed.

For example, if the food is initially at matrix[2][3], and the snake eats this food at actual_time = T45, the next food then appears at the place which is mapped to variable T45.

This position is naturally updated to the display matrix and printed every cycle.

We Finally create the **Additional Crash Logic**. This naturally works for snakes which have a length more than 4, where the snake when it collides with itself, must end the game as it is an invalid move. We simply do this by checking the coordinates of all the snake parts and check if all of them are unique. If any two of them happen to be the same, it will indicate that the snake has overlapped with itself and hence has crashed, thus ending the game. We do this through a simple search and compare function.

for (i = 0; i < length of snake; i = i + 1)

check uniqueness of snake[i];

if at all the snake has crashed the score and the timer are then reset back to their original values.

Hardware Implementation:

We implemented the game In proteus Using Arduino UNO. We Completed the circuit using Arduino UNO, 8x8 LED Dot Matrix Display, Shift Register 74HC595, 16x2 LCD, POT 1K, Push Buttons, Connecting wires, Bread Board, Power Supply.

Arduino Board: We used Arduino Uno Which has ATmega328P IC. It has **14** digital input/output pins (of which **6** can be used as PWM outputs), **6** analog inputs, a 16 MHz ceramic resonator (CSTCE16M0V53-R0), a USB connection, a power jack, an ICSP header and a reset button.

LED Matrix:

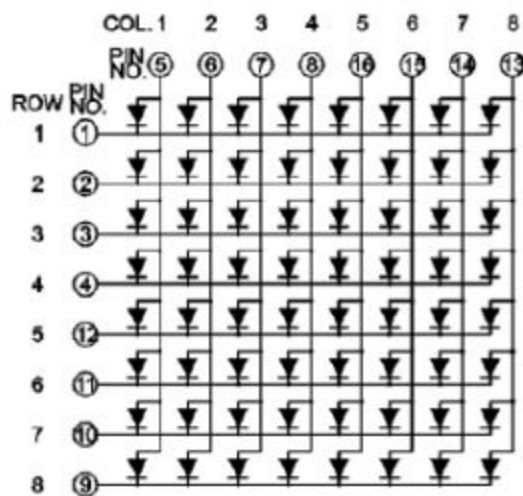


Fig: Pin Configuration of 8X8 Matrix in Proteus

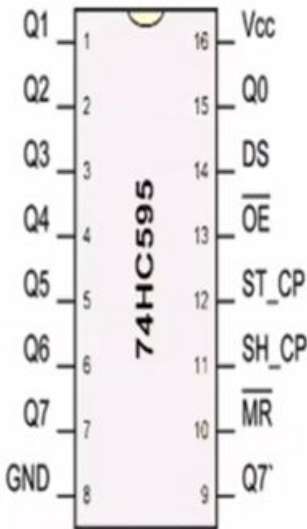
We want to describe LED matrix's operation by an example. Suppose, we want to light up bulb in row 3 and column 4. This is done by latching Column 4 pin to +5V and 0V to remaining column pins. Now, the row 3 is enabled by latching it to 0V and remaining rows are latched to +5V.

Shift Register 74HC595: It is a Serial In Parallel Out (SIPO) Shift register with Master reset and enable pin. It was used to control the LED matrix.

How shift register controls LED matrix:

74HC595 Shift Register:

Pin	Symbol	Description
1	Q1	Parallel data output (bit-1)
2	Q2	Parallel data output (bit-2)
3	Q3	Parallel data output (bit-3)
4	Q4	Parallel data output (bit-4)
5	Q5	Parallel data output (bit-5)
6	Q6	Parallel data output (bit-6)
7	Q7	Parallel data output (bit-7)
8	GND	Ground (0 V)
9	Q7'	Serial Data Output
10	\overline{MR}	Master Reset (Active Low)
11	SH_CP	Shift Register Clock Input
12	ST_CP	Storage Register Clock Input
13	\overline{OE}	Output Enable (Active Low)
14	DS	Serial Data Input
15	Q0	Parallel data output (bit-8)
16	Vcc	Positive Supply Voltage



DS - pin is used to feed data into the shift register a bit at a time.

SH_CP- is the clock for the shift register. The 595 is clock-driven on the rising edge. This means that in order to shift bits into the shift register, the clock must be HIGH. And bits are transferred in on the rising edge of the clock.

ST_CP- is a very important pin. When driven HIGH, the contents of Shift Register are copied into the Storage/Latch Register; which ultimately shows up at the output. So the latch pin can be seen as like the final step in the process to seeing our results at the output, which in this case are LEDs.

MR- pin allows us to reset the entire Shift Register, making all its bits 0, at once. This is a negative logic pin, so to perform this reset; we need to set the SRCLR pin LOW. When no reset is required, this pin should be HIGH.

OE- is negative logic too: When the voltage on it is HIGH, the output pins are disabled/set to high impedance state and don't allow current to flow. When OE gets low voltage, the output pins work normally.

GND-should be connected to the ground of Arduino.

VCC-is the power supply for 74HC595 shift register which we connect the 5V pin on the Arduino.

Q0-Q7 - are the output pins and should be connected to some type of output like LEDs, 7 Segments etc.

Q7'- We can have serial Data output from this pin.if we connect this Q7' to the DS pin of another 74HC595, and give both ICs the same clock signal, they will behave like a single IC with 16 outputs.

The 74HC595 has two registers (which can be thought of as “memory containers”), each with just 8 bits of data. The first one is called the Shift Register. The Shift Register lies deep within the IC circuits, quietly accepting input.

Whenever we apply a clock pulse to a 595, two things happen:

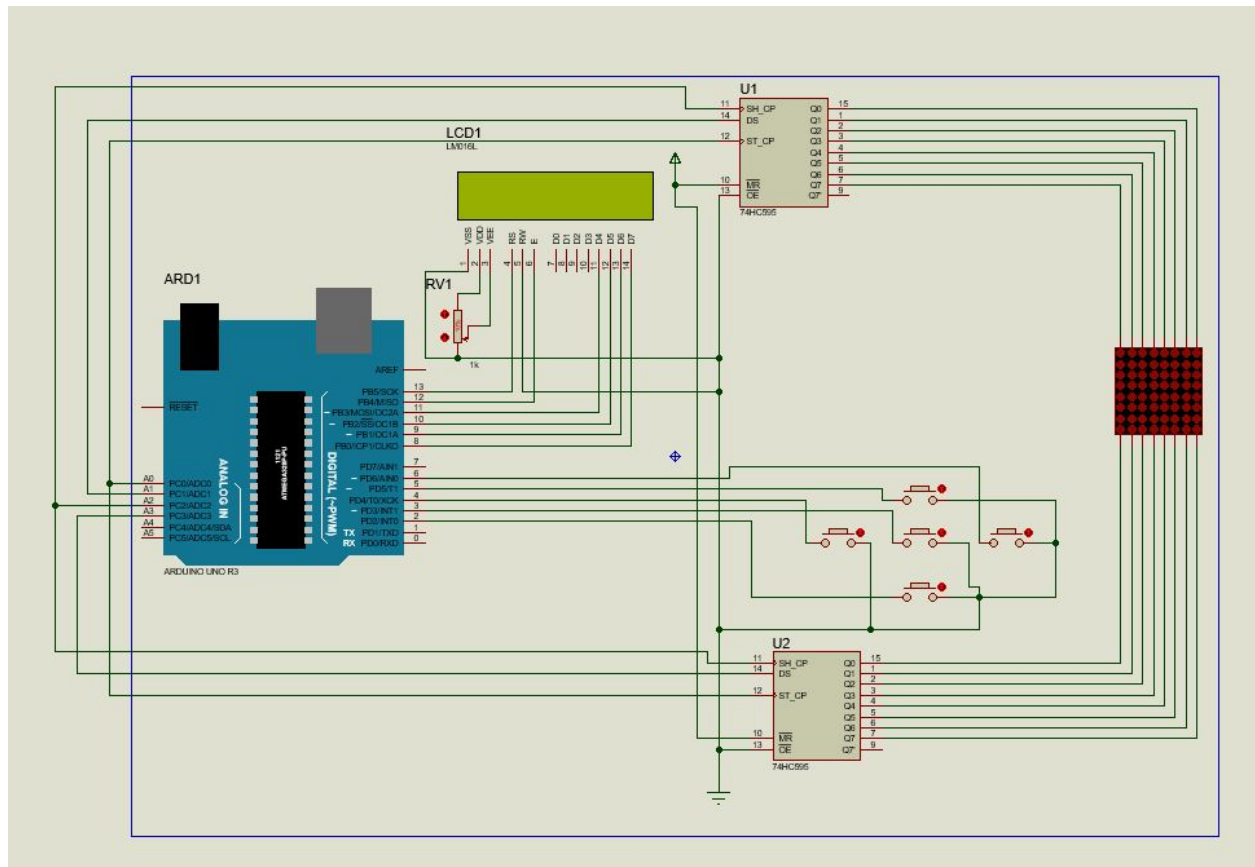
- The bits in the Shift Register move one step to the left. For example, Bit 7 accepts the value that was previously in bit 6, bit 6 gets the value of bit 5 etc.

- Bit 0 in the Shift Register accepts the current value on DATA pin. At the rising edge of the pulse, if the data pin is high, then a 1 gets pushed into the shift register. Otherwise, it is a 0.

On enabling the Latch pin, the contents of Shift Register are copied into the second register, called the Storage/Latch Register. Each bit of the Storage Register is connected to one of the output pins Q0-Q7 of the IC, so in general, when the value in the Storage Register changes, so do the outputs.

PUSH BUTTON: We Used push buttons, Each for directions and starting the game.

Schematic



Arduino sketch and proteus file has been included in the Drive link.

Arduino CODE:

```
#include<LiquidCrystal.h>

LiquidCrystal lcd(13,12,11,10,9,8);

//LiquidCrystal lcd(13=RS,E=12,D4=11,D5=10,D6=9,D7=8);


#define ds_col 15 // Serial Data_in for Col shift register

#define sh_col 16 // shift register Clock pulse

#define st_col 14 // Storage register Clock Pulse

#define ds_row 17 // Serial Data_in for row shift register


#define start 3

#define up 2

#define down 5

#define left 4

#define right 6

// here we defined input and output Pins as well as Pins connected to the shift register.//
```

//in the next part, we defined a simplified function which can read digital input and checks for match with the if else conditions to perform up, down, left, right operation.//

```
void read_button()
{
    if(!digitalRead(left))
    {
        move_r=0;
        move_c!=1 ? move_c=-1 : move_c=1;
        while(!digitalRead(left));
    }
    if(!digitalRead(right))
    {
        move_r=0;
        move_c!=1 ? move_c=1 : move_c=-1;
        while(!digitalRead(right));
    }
    if(!digitalRead(up))
    {
        move_c=0;
        move_r!=1 ? move_r=-1 : move_r=1;
        while(!digitalRead(up));
    }
}
```

```

}
if(!digitalRead(down))
{
    move_c=0;
    move_r!=1 ? move_r=1 : move_r=-1;
    while(!digitalRead(down));
}
}

```

//This function shows the snake in the LED matrix.and outputs serial data in and when to provide positive edge clock pulse in registers.

```

void show_snake(int temp)
{
    for(int n=0;n<temp;n++)
    {
        int r,c;
        for(int k=0;k<21;k++)
        {
            int temp1=Col[k];
            c=colum_data(temp1);
            int temp2=Row[k];
            r=0xff-row_data(temp2);
            for(int i=0;i<8;i++)

```



```

{
    int ds=(c & 0x01);
    digitalWrite(ds_col, ds);
    ds=(r & 0x01);
    digitalWrite(ds_row, ds);
    digitalWrite(sh_col, HIGH);
    c>>=1;  //bitshift right operator >>
    r>>=1;
    digitalWrite(sh_col, LOW);
}

digitalWrite(st_col, HIGH);
digitalWrite(st_col, LOW);
read_button();
delayMicroseconds(500);
}
}
}

```

```

void setup()
{
    lcd.begin(16,2);

```

```
pinMode(ds_col, OUTPUT);
pinMode(sh_col, OUTPUT);
pinMode(st_col, OUTPUT);
pinMode(ds_row, OUTPUT);
pinMode(start, INPUT);
pinMode(up, INPUT);
pinMode(down, INPUT);
pinMode(left, INPUT);
pinMode(right, INPUT);
digitalWrite(up, HIGH);
digitalWrite(down, HIGH);
digitalWrite(left, HIGH);
digitalWrite(right, HIGH);
digitalWrite(start, HIGH);
lcd.setCursor(0,0);
lcd.print(" Snake game ");
lcd.setCursor(0,1);
lcd.print(" EEE 304 ");
delay(1000);
lcd.setCursor(0,0);
lcd.print(" Press Start ");
```

```
        lcd.setCursor(0,1);  
        lcd.print("  To Play  ");  
        delay(2000);  
    }
```

```
void loop()  
{  
    int j,k,Speed=45,score=0;  
    j=k=move_c=0;  
    move_r=1;  
    lcd.clear();  
    lcd.setCursor(0,0);  
    lcd.print("Score: ");  
    lcd.print(score);  
    while(1)  
    {  
        for(int i=3;i<21;i++)  
        {  
            Row[i]=100;  
            Col[i]=100;  
        }  
    }
```

```

Row[0]=rand()%8+1;
Col[0]=rand()%8+1;
Row[1]=1; // snake head position Row number
Col[1]=1; // snake head position col number
Row[2]=2; // snake tail position row number
Col[2]=1; // snake tail position col number
j=2,k=1;
while(k==1)
{
    move_c=0;
    move_r=1;
    show_snake(1);
    lcd.setCursor(7,0);
    lcd.print(score);
    if(!digitalRead(start))
    {
        k=2;
        Speed=40;
        score=0;
    }
}

```

```

while(k==2)

{

    show_snake(Speed);

    //This block checks for constraints upon the snake.If the snake reaches the
    //boundary of the LED matrix, the game will be terminated.

    if(Row[1]>8 || Col[1]>8 || Row[1]<0 || Col[1]<0)

        {

            Row[1]=1;

            Col[1]=1;

            k=1;

            lcd.setCursor(0,1);

            lcd.print("Game Over");

            delay(5000);

            score=0;

            lcd.clear();

            lcd.setCursor(0,0);

            lcd.print("Score: ");

            lcd.print(score);

        }

    //this block below checks if snake location reaches food location

    if(Row[0]==Row[1]+move_r && Col[0]==Col[1]+move_c)

```

```

    {
        j++;
        Speed-=2;
        score=score+5;
        lcd.setCursor(7,0);
        lcd.print(score);
        Row[0]=rand()%8+1;
        Col[0]=rand()%8+1;
    }

    for(int i=j;i>1;i--)
    {
        Col[i]=Col[i-1];
        Row[i]=Row[i-1];
    }

    Col[1]=Col[2]+move_c;
    Row[1]=Row[2]+move_r;
}
}
}

```

Contribution(Shahedul Hasan):

In this project, I had to implement movement logic of the snake. I also wrote the Main functions of the Arduino Code and simplified the code and constraints when necessary to get an output in limited time. Overall, I had to manage the work flow of the project.