EC383 VLSI MINI PROJECT

8 BITS DIGITAL TO ANALOG CONVERTER

FINAL REPORT

UNDER THE SUPERVISION OF	**: MRS KALPANA BHAT**	
MENTEES	**: ANIRRVHINYAN K**	**191EC205**
	: SHRIDEVI KUMAR	**191EC150**
	: KAVEEN SRIKANTHAN	**191EC215**

TABLE OF CONTENTS

8 BIT DIGITAL TO ANALOG CONVERTER

OBJECTIVE:

Implementation of the 8-bit DAC using different techniques on various software and building an application unit for it as well. Further we also implement a ADC and understand its applications.

SOFTWARES USED:

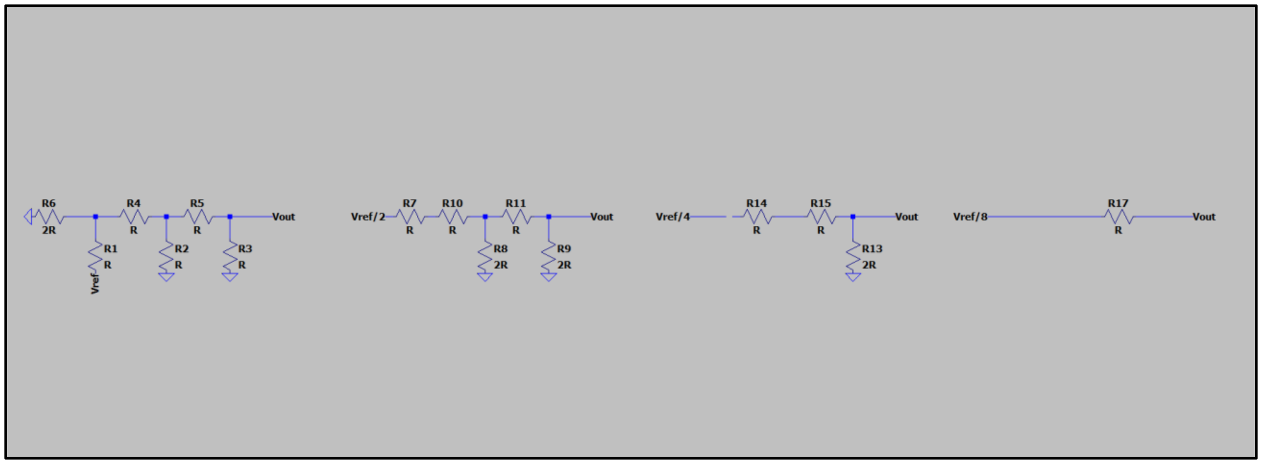
- LTSpice
- Magic
- Ngspice
- Proteus
- Tinkercad
- Arduino.cc

PHASE 1 : R-2R DAC

We aim to build the Digital to Analog Convertor by first designing it and implementing it on LTSpice software and later simulating it on Proteus software.

METHODOLOGY

Let's take 3 – bit DAC and Inputs as D_2 D_1 D_0



For example

$D_2 D_1 D_0 = 001$

At $D_0 = Vref = 10$ V

$Vth = [\frac{2R}{2R+2R}]\ Vref = \frac{\mathbf{Vref}}{\mathbf{2}}$

$Rth = R$

Again for resullt circuit $Vth = \frac{\frac{\mathbf{Vref}}{\mathbf{2}}}{\mathbf{2}} = \frac{\mathbf{Vref}}{\mathbf{4}}$

Rth remains R

In the final circuit $Vth = \frac{\frac{\mathbf{Vref}}{\mathbf{4}}}{\mathbf{2}} = \frac{\mathbf{Vref}}{\mathbf{8}}$ and Rth is R

$Vout = \frac{\mathbf{Vref}}{\mathbf{8}}$

This technique is followed for all values which gives $\mathbf{Vout = Vref\ [\frac{D2}{2} + \frac{D1}{4} + \frac{D0}{8}]}$

Similarly for 8 bit R-2R DAC:

The Vout calculation depending on Input:

Let's consider all Inputs ie. $V_0, V_1, V_2 \ldots V_7 = 0$

Then Node A = Node B = ….. = Node G = Vout = 0V as all are connected to ground

If all inputs are 10 V, we get Vout as 10V because all nodes are in 10V

All values from 0 – 255 will have values output between 0 – 10V

The equation of Vout for different binary inputs is:

$$\mathbf{Vout = Vref\ [\frac{D7}{2} + \frac{D6}{4} + \frac{D5}{8} + \frac{D4}{16} + \frac{D3}{32} + \frac{D2}{64} + \frac{D1}{128} + \frac{D0}{256}]}$$

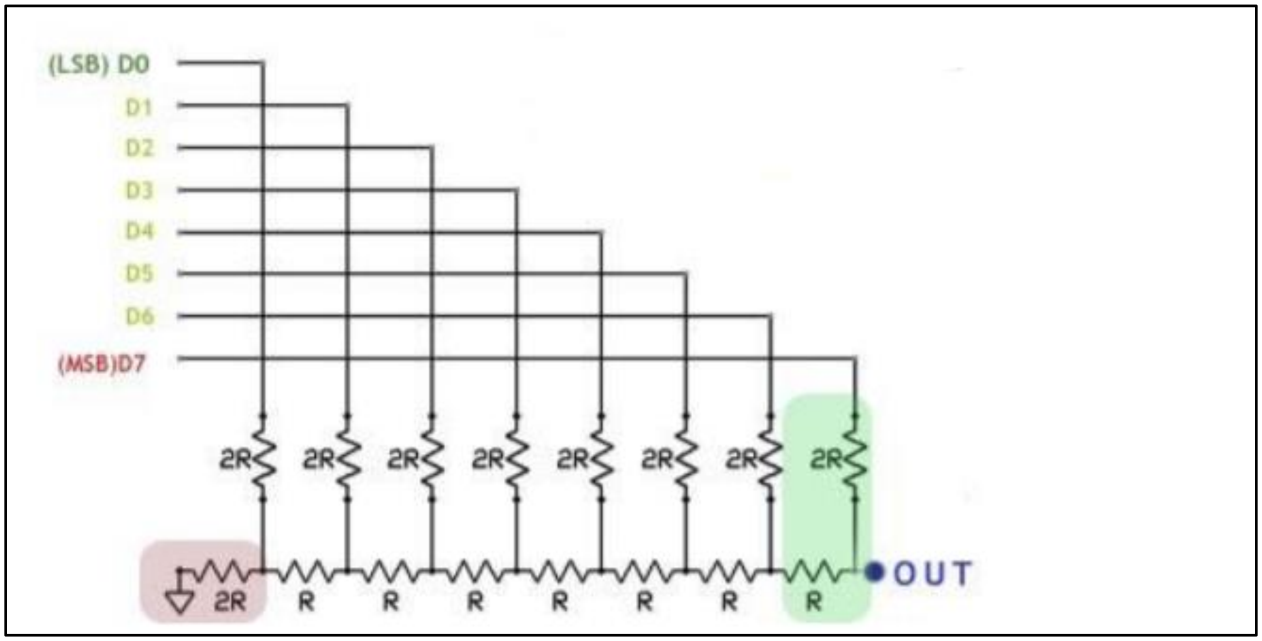
Here $V_0, V_1, V_2 \ldots V_7$ are 0 if 0V and 1 if 10V.

Example:

If $V_0 = V_1$ = High are rest are Low

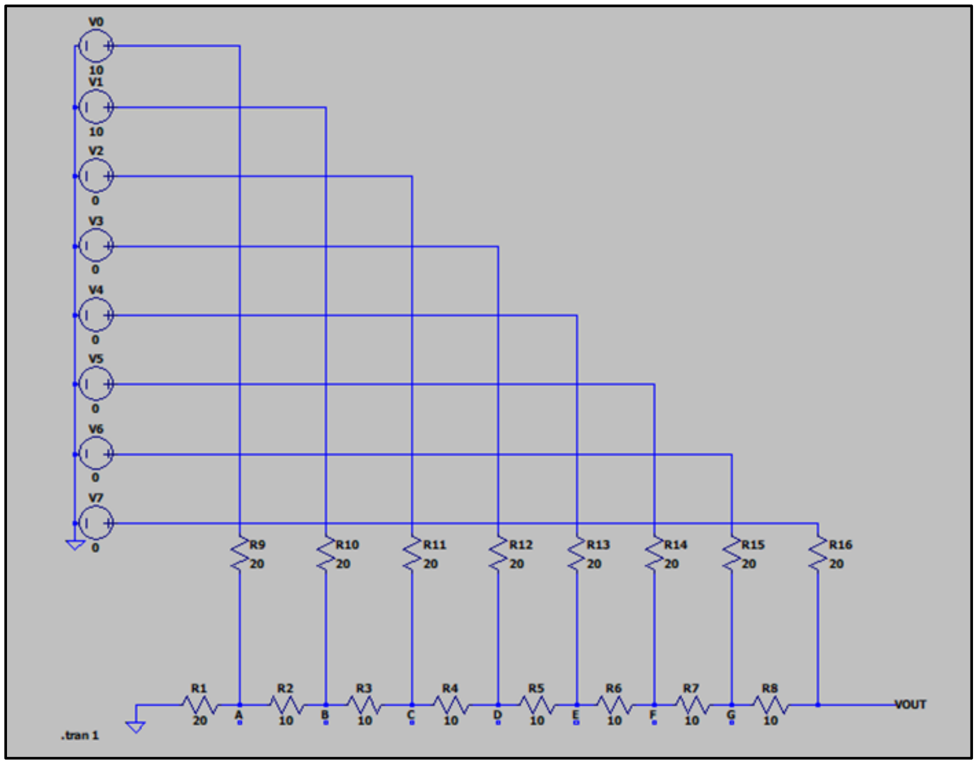
$\mathbf{Vout = 10\ [\frac{1}{128} + \frac{1}{256}] = 0.1172\ V}$

Circuit Diagram:

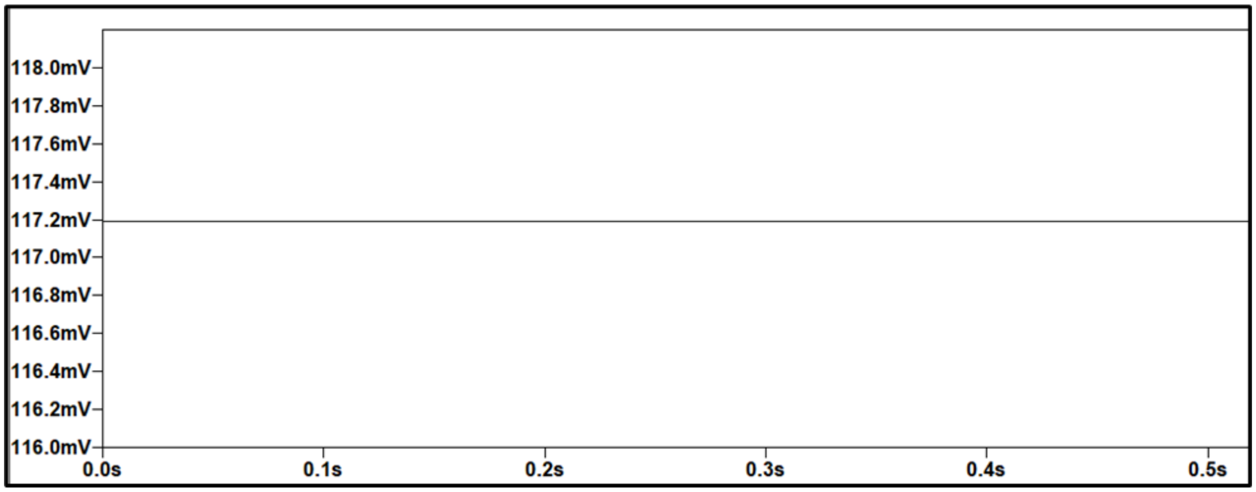


<u>Simulation in LTSpice:</u>

<u>Circuit:</u>



Output:



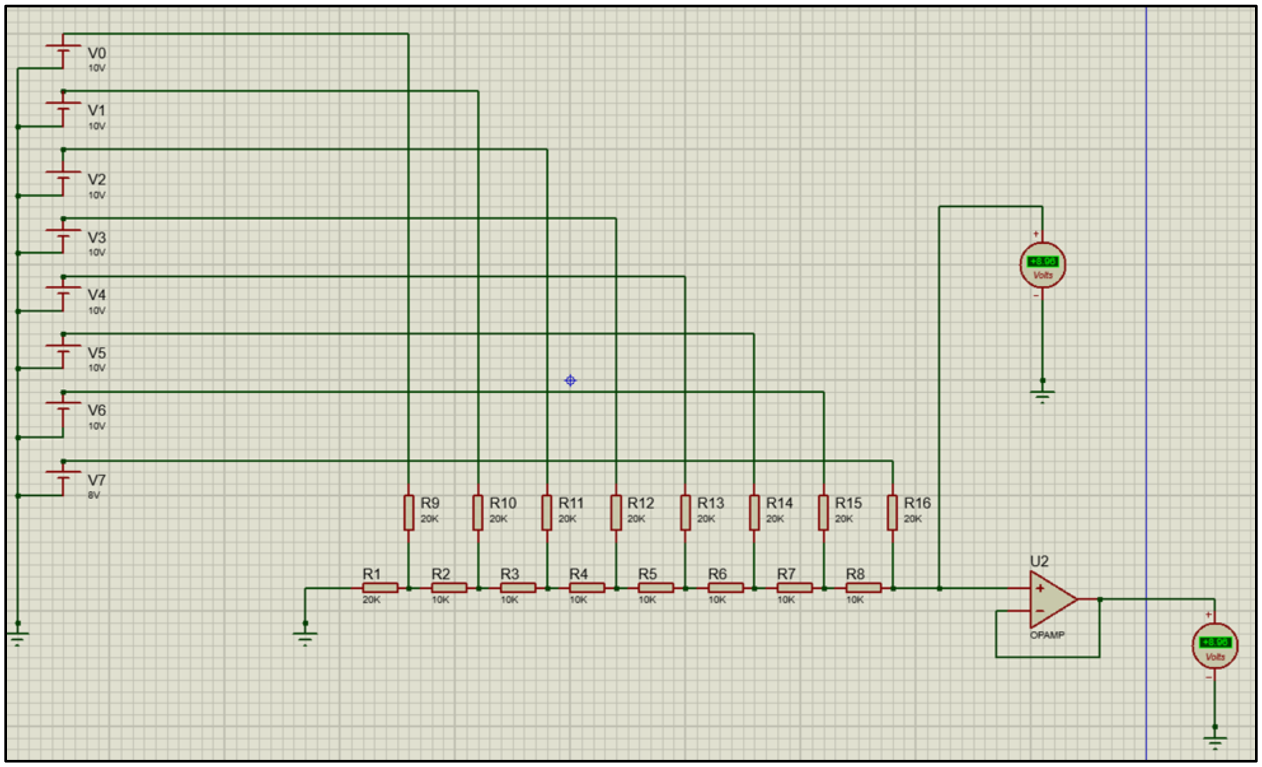
Vout = 117.2 mV

- Now that this is a resistor network, if we apply any load on the output of the first stage,
  this load will be considered as an additional resistor in the network, and thus will disturb
  the network which will no longer provide the correct & desired output voltage.
- Therefore, to overcome this problem, we need a voltage buffer, here is where the next
  stage comes.

Simulation in Proteus:

Circuit:

Output is also shown in DC Voltmeter



With voltage buffer:

This stage will isolate the point V1 from the final output V2, while always keeping the voltage V2 at the exact same value as V1. This is what we call a voltage buffer. For the voltage buffer, we use an opamp with the output connected to the inverting input (this special configuration of the Op Amp is also called Voltage Follower). The most important things to note are:

1. No current (almost 0A) will flow from the point V1 into the OpAmp, so we won't be disturbing the resistor network configuration.

2. V2 will always equal V1 (theoretically, as shown in the rest of this document).

3. The current going out from the point V2 to any other stage is sourced from the power supply of the OpAmp.

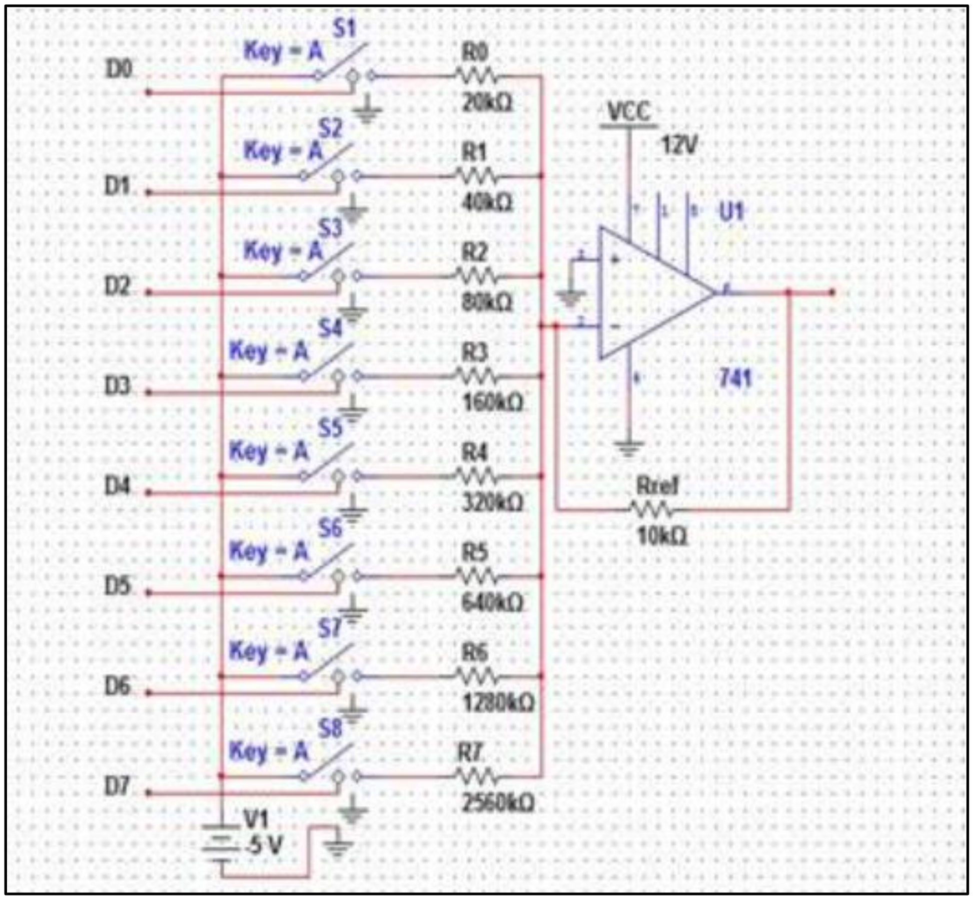
PHASE 2 : BINARY WEIGHTED RESISTOR DAC

METHODOLOGY:

The binary-weighted-resistor DAC employs the characteristics of the inverting summer Op Amp circuit. In this type of DAC, the output voltage is the inverted sum of all the input voltages.

If the input resistor values are set to multiples of two: 1R, 2R and 4R and so on, the output voltage would be equal to the sum of V1, V2/2 and V3/4 and so on . V1 corresponds to the most significant bit (MSB) while Vn-1 corresponds to the least significant bit (LSB), where n is the number of bits

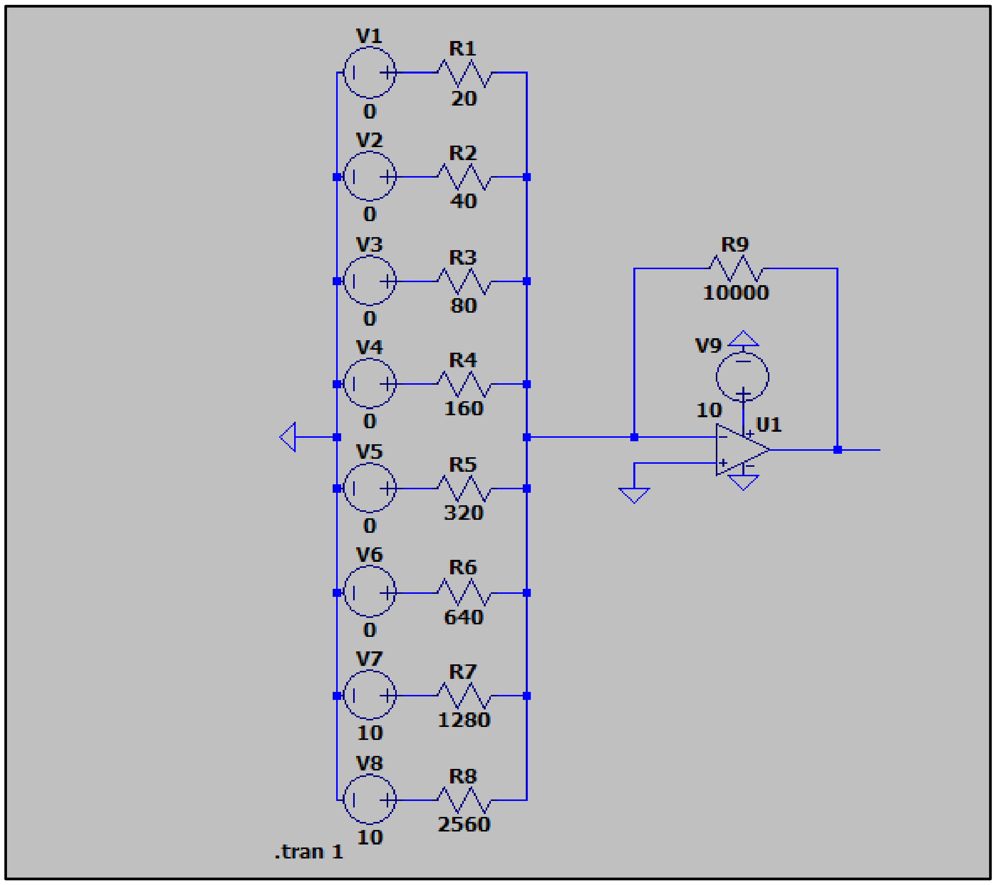
Circuital Diagram:



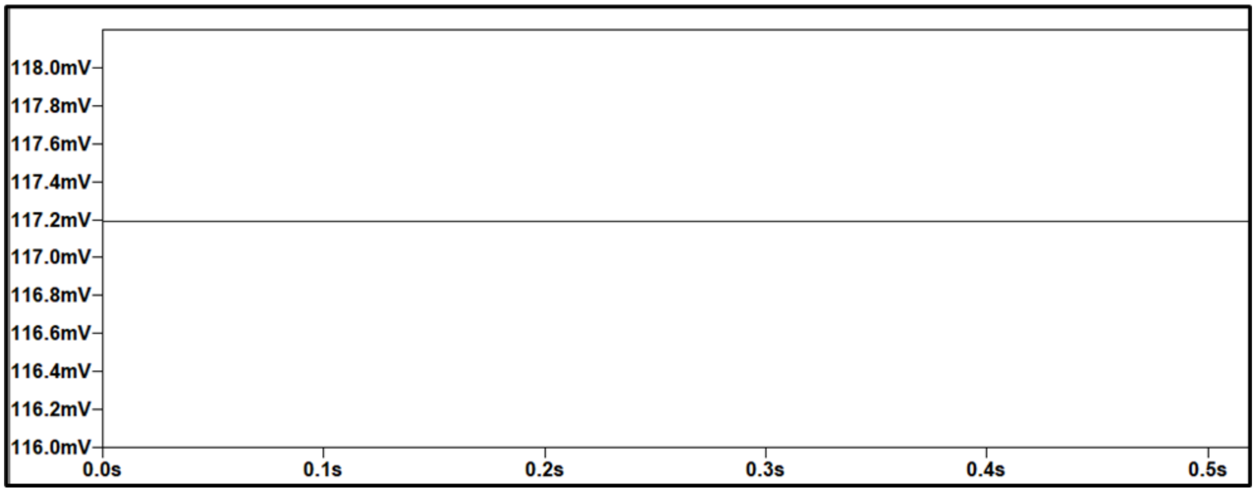
The values of resistance can be different but relatives between every resistor should not be different from one circuit to another.

Simulation in LTSpice:

Circuit:



Output:



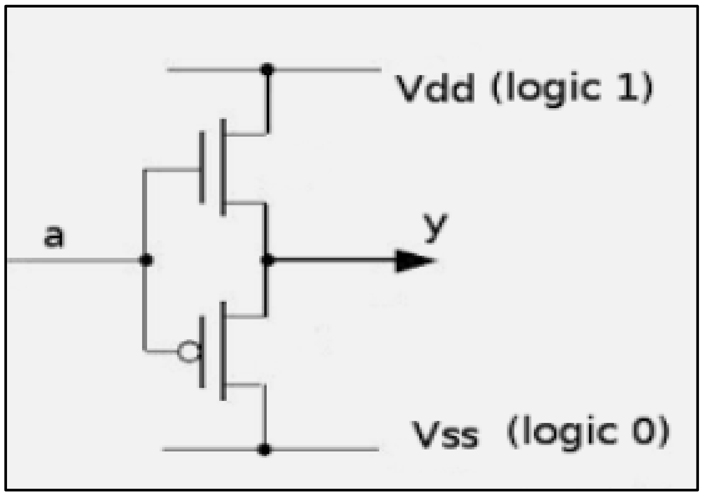
Vout = 117.2 mV

This value matches with the Ladder circuit's output value.

BUFFER CIRCUIT

As we have seen the requirement of a buffer circuit, we should also see which circuit can be used as a buffer. In phase 1, we have used a simple op amp with unity feedback, but when VLSI design is considered, it's difficult to design an op amp.

So in our case, we use CMOS Buffer as shown below:

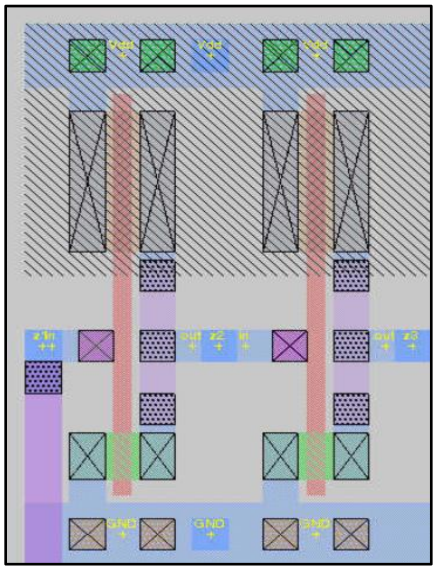


One of the advantages of using CMOS Buffer is extremely helpful in signal restoration in communicating over long wires. Also as far as Magic Layout Simulation is concerned, it's better to make a buffer using CMOS technology.

From standard cells, we can add two inverters in series to make a buffer or else we can do like shown above connecting Nmos to Vdd and Pmos to Vss.

Magic Layout of a Buffer:

When 2 inverters are in series:



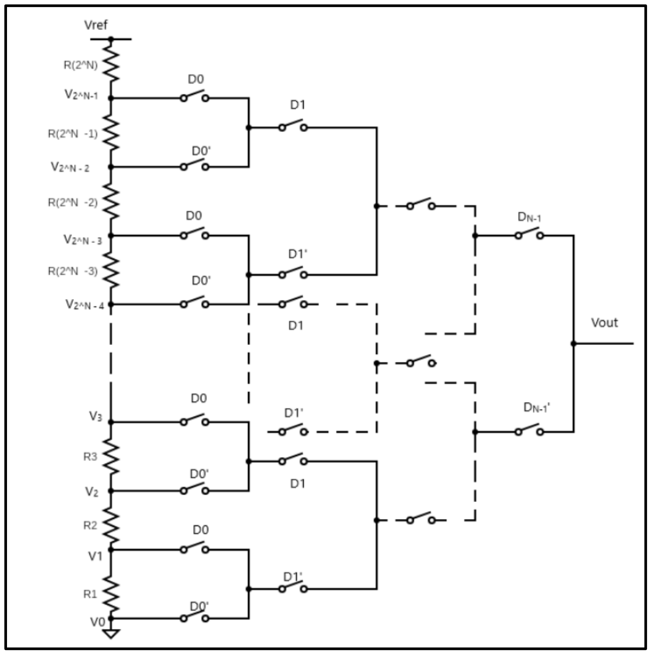
PHASE 3 :- 4 - DAC IN MAGIC TOOL:

Methodology:

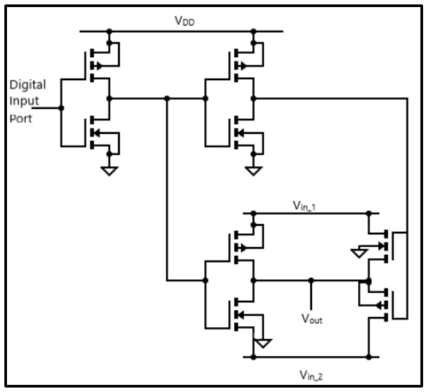
The basic idea is to divide the voltage into 4 different voltage values in the range of 0 to VRef for an 4-Bit DAC. The design used here to achieve this is the simple resistor string DAC which consists of resistors in series.

These resistors are then connected to various switches in such a fashion that it routes the exact voltage to the output. The problem of the largeness of the circuit is reduced by building hierarchical subcircuits of 4-Bit potentiometric DAC – Switch, 2-bit, 3-bit, 4-bit.

DAC Circuit:



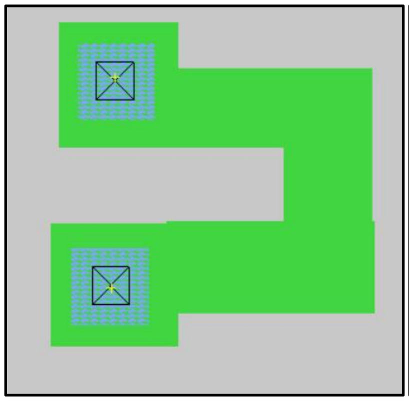
Switch:

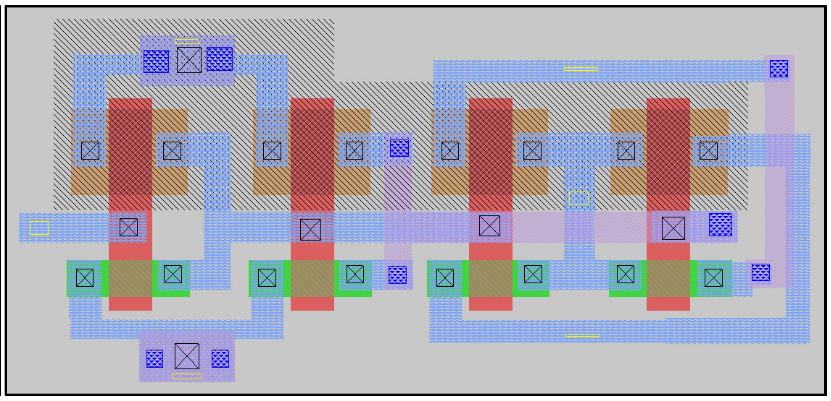


Each resistor is connected with two switches

The basic building blocks in our circuit is resistor and switch, we have designed both using magic tool as shown below:

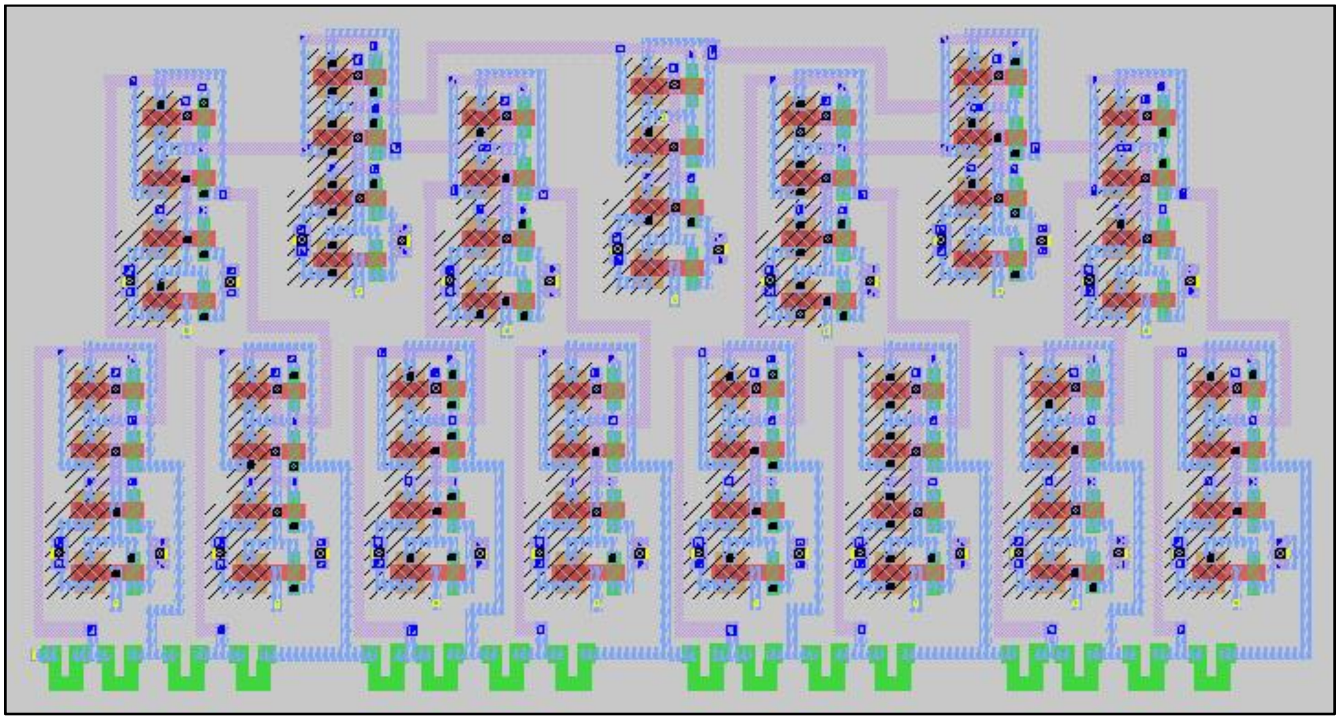
Resistor: **Switch:**



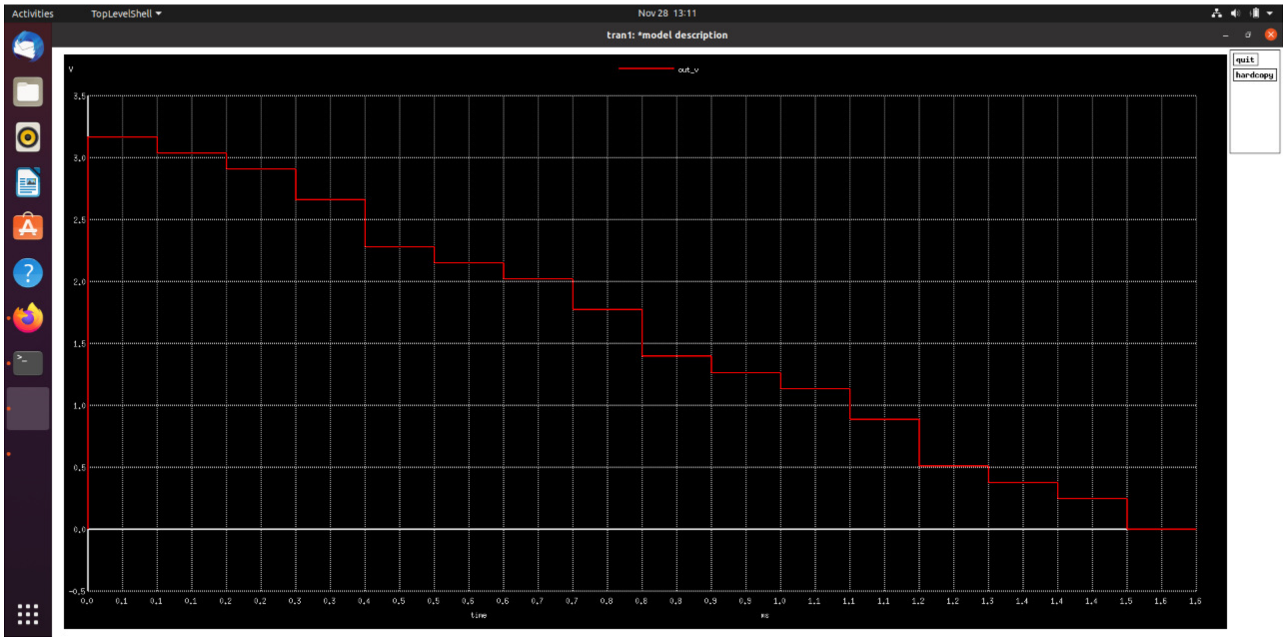


These building blocks are then arranged in a similar way as shown in the circuit diagram. The realization of the above circuit shows the Binary weighted DAC, hence the value of each resistor offered differs based on the place of existence.

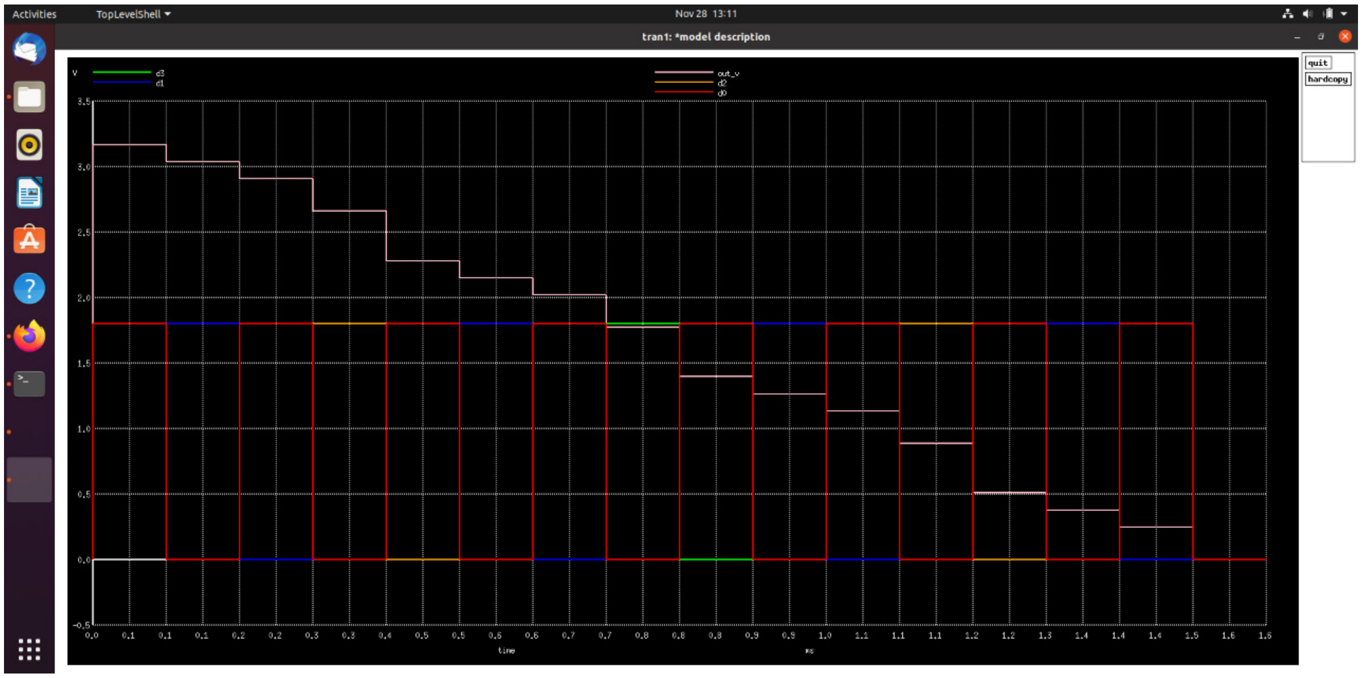
In this pattern, the 4 Bit DAC is shown below:



Pre-Layout Ng spice simulation:



Final Output:



Vref = 3.3 V (We have used optimum voltage as we are doing in CMOS level technology)

Red line	- V0
Blue line	- V1
Orange line	- V2
Green line	- V3

APPLICATION OF DIGITAL TO ANALOG CONVERTOR:

As we have already seen, DAC can be used as a **Potentiometer** as its basic principles depend on Voltage drops between a given Load. Leaving that aside, there are many more practical applications of DAC.

Function Generator using the 8 bit DAC-

For our project we are building a Function generator using the DAC to produce

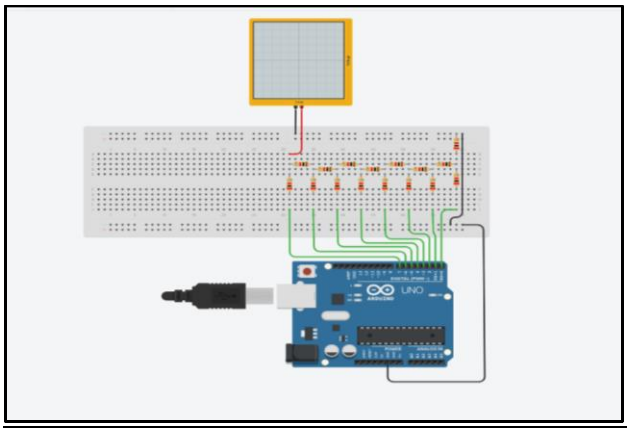
1. Rectangle Function
2. Sawtooth Function
3. Triangle function

DACs are commonly used to get signals of a particular voltage that other devices are unable to do .For example an Arduino would supply only a high and a low voltage ,but by using a DAC we can get voltages of any value to the point. That is what we intend to do with this application circuit. Further using the values we get waveforms of the required function.

We built this application unit using Tinker cad software where we used the following components-

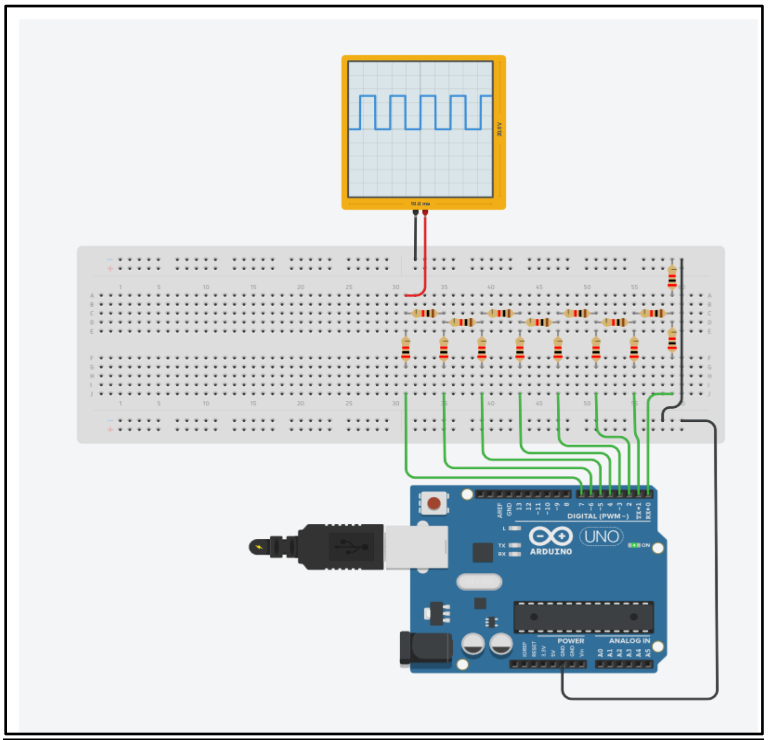
1. Oscilloscope
2. Arduino
3. Resistors
4. Wires
5. Breadboard

Circuit -

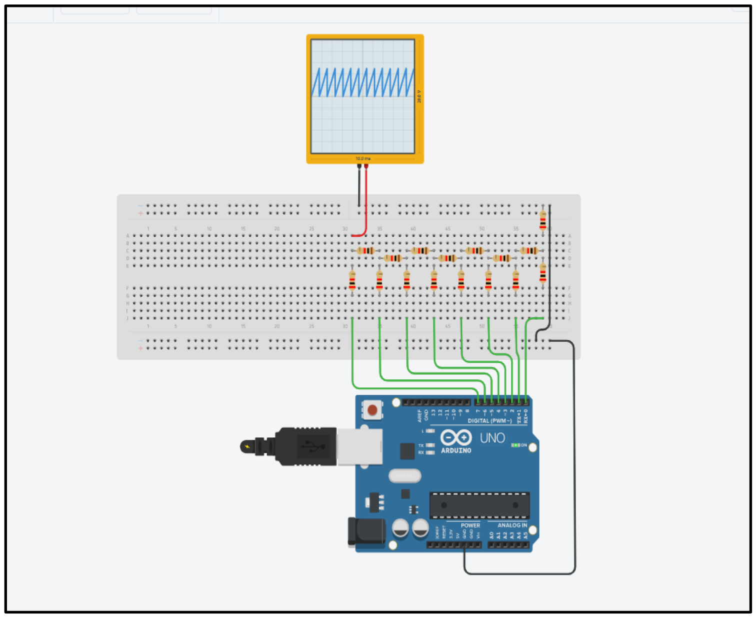


Output-

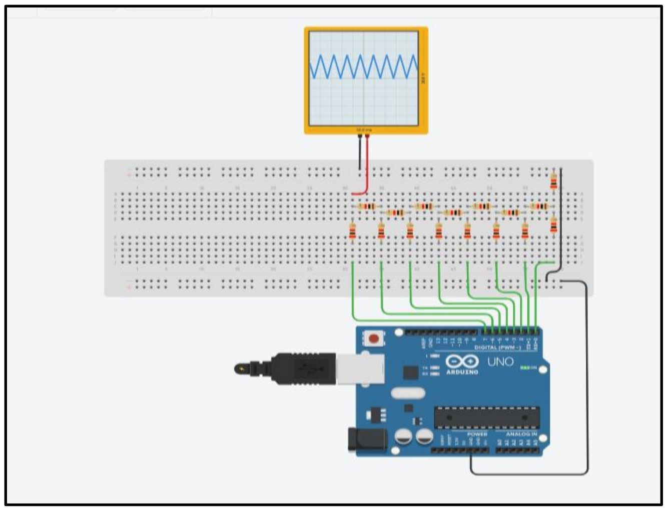
1.Rectangle Function-



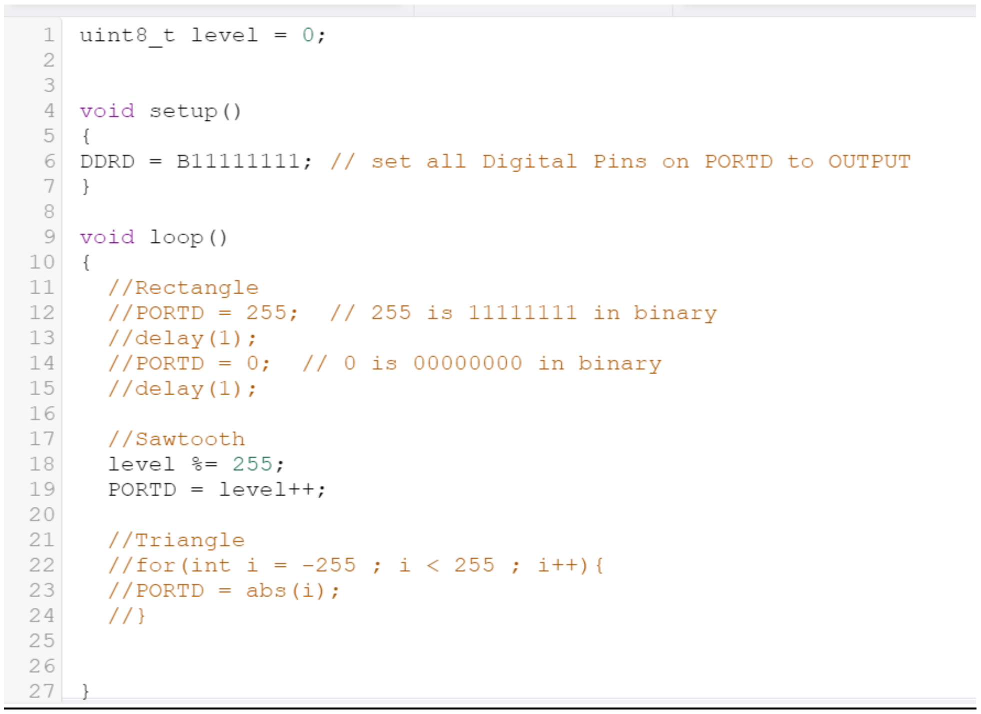
2.Sawtooth Function-



3.Triangle Function-



Code-



```
uint8_t level = 0;


void setup()
{
DDRD = B11111111; // set all Digital Pins on PORTD to OUTPUT
}

void loop()
{
  //Rectangle
  //PORTD = 255;  // 255 is 11111111 in binary
  //delay(1);
  //PORTD = 0;  // 0 is 00000000 in binary
  //delay(1);

  //Sawtooth
  level %= 255;
  PORTD = level++;

  //Triangle
  //for(int i = -255 ; i < 255 ; i++){
  //PORTD = abs(i);
  //}


}
```

Result -

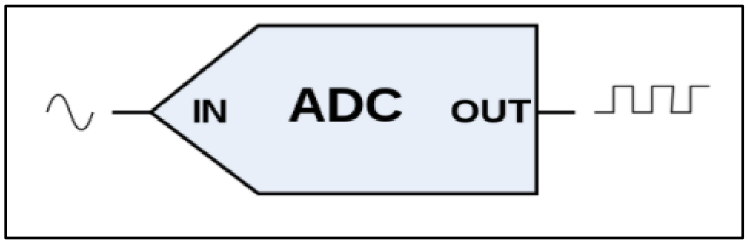
We were able to successfully simulate the three functions and get their graphs on the oscilloscope.

PHASE 4 :- ANALOG TO DIGITAL CONVERTOR:

Basically, an analog to digital converter takes a snapshot of an analog voltage at one instant in time and produces a digital output code which represents this analog voltage. The number of binary digits, or bits used to represent this analog voltage value depends on the resolution of an A/D converter. In place of A/D converter, we can use other modes like Arduino etc.

This is just reverse mechanism of DAC

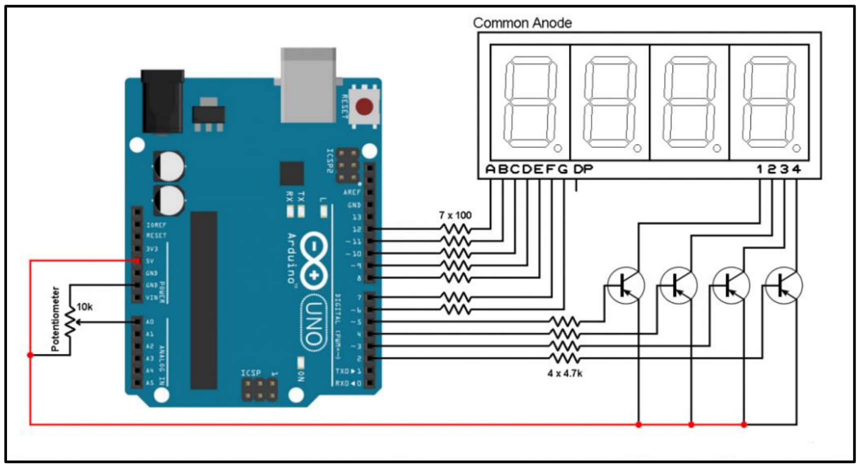
Block Diagram of ADC:



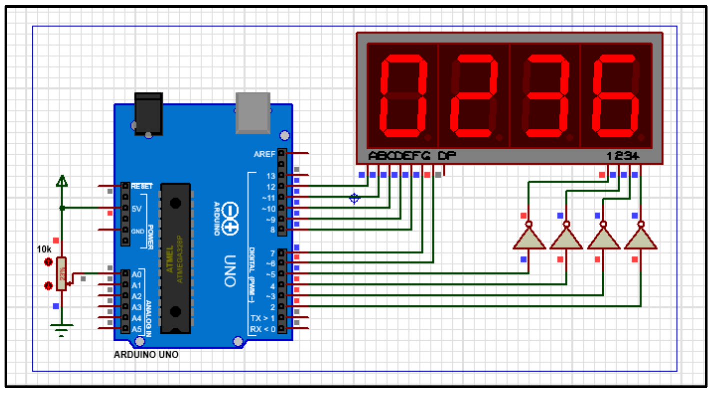
Applications of ADC are:

- Used together with the transducer.
- Used in computers to convert the analog signal to digital signal.
- Used in cell phones.
- Used in microcontrollers.
- Used in digital signal processing.
- Used in digital storage oscilloscopes.
- Used in scientific instruments.
- Used in music reproduction technology etc.

Let us see design of ADC using Arduino:



Simulation in Proteus:



The rheostat connected to A0 pin decides the voltage drop across A0 which is considered as input voltage.

And then necessary connections are made in the digital pins side which converts Analog voltage in the form of a number displayed in a 7-segment display.

Code:

```
// potentiometer output pin definition
#define POT    A0
// segment pin definitions
#define SegA   12
#define SegB   11
#define SegC   10
#define SegD    9
#define SegE    8
#define SegF    7
#define SegG    6
// common pins of the four digits definitions
#define Dig1    5
#define Dig2    4
#define Dig3    3
#define Dig4    2
// variable declarations
byte current_digit;
int  adc_value;
void setup()
{
  pinMode(SegA, OUTPUT);
  pinMode(SegB, OUTPUT);
  pinMode(SegC, OUTPUT);
  pinMode(SegD, OUTPUT);
  pinMode(SegE, OUTPUT);
  pinMode(SegF, OUTPUT);
  pinMode(SegG, OUTPUT);
  pinMode(Dig1, OUTPUT);
  pinMode(Dig2, OUTPUT);
  pinMode(Dig3, OUTPUT);
  pinMode(Dig4, OUTPUT);
  disp_off();  // turn off the display
  // Timer1 module overflow interrupt configuration
  TCCR1A = 0;
  TCCR1B = 1;  // enable Timer1 with prescaler = 1 ( 16 ticks each 1 µs)
  TCNT1  = 0;  // set Timer1 preload value to 0 (reset)
  TIMSK1 = 1;  // enable Timer1 overflow interrupt
}
```

```
ISR(TIMER1_OVF_vect)   // Timer1 interrupt service routine (ISR)
{
  disp_off();  // turn off the displays
  switch (current_digit)
  {
    case 1:
      disp(adc_value / 1000);   // prepare to display digit 1 (most left)
      digitalWrite(Dig1, LOW);  // turn on digit 1
      break;

    case 2:
      disp( (adc_value / 100) % 10);   // prepare to display digit 2
      digitalWrite(Dig2, LOW);     // turn on digit 2
      break;

    case 3:
      disp( (adc_value / 10) % 10);   // prepare to display digit 3
      digitalWrite(Dig3, LOW);    // turn on digit 3
      break;

    case 4:
      disp(adc_value % 10);   // prepare to display digit 4 (most right)
      digitalWrite(Dig4, LOW);  // turn on digit 4
  }

  current_digit = (current_digit % 4) + 1;
}

// main loop
void loop()
{
  adc_value = analogRead(POT);
  delay(100);
}
```

```
void disp(byte number)
{
  switch (number)
  {
    case 0:  // print 0
      digitalWrite(SegA, LOW);
      digitalWrite(SegB, LOW);
      digitalWrite(SegC, LOW);
      digitalWrite(SegD, LOW);
      digitalWrite(SegE, LOW);
      digitalWrite(SegF, LOW);
      digitalWrite(SegG, HIGH);
      break;

    case 1:  // print 1
      digitalWrite(SegA, HIGH);
      digitalWrite(SegB, LOW);
      digitalWrite(SegC, LOW);
      digitalWrite(SegD, HIGH);
      digitalWrite(SegE, HIGH);
      digitalWrite(SegF, HIGH);
      digitalWrite(SegG, HIGH);
      break;

    case 2:  // print 2
      digitalWrite(SegA, LOW);
      digitalWrite(SegB, LOW);
      digitalWrite(SegC, HIGH);
      digitalWrite(SegD, LOW);
      digitalWrite(SegE, LOW);
      digitalWrite(SegF, HIGH);
      digitalWrite(SegG, LOW);
      break;
```

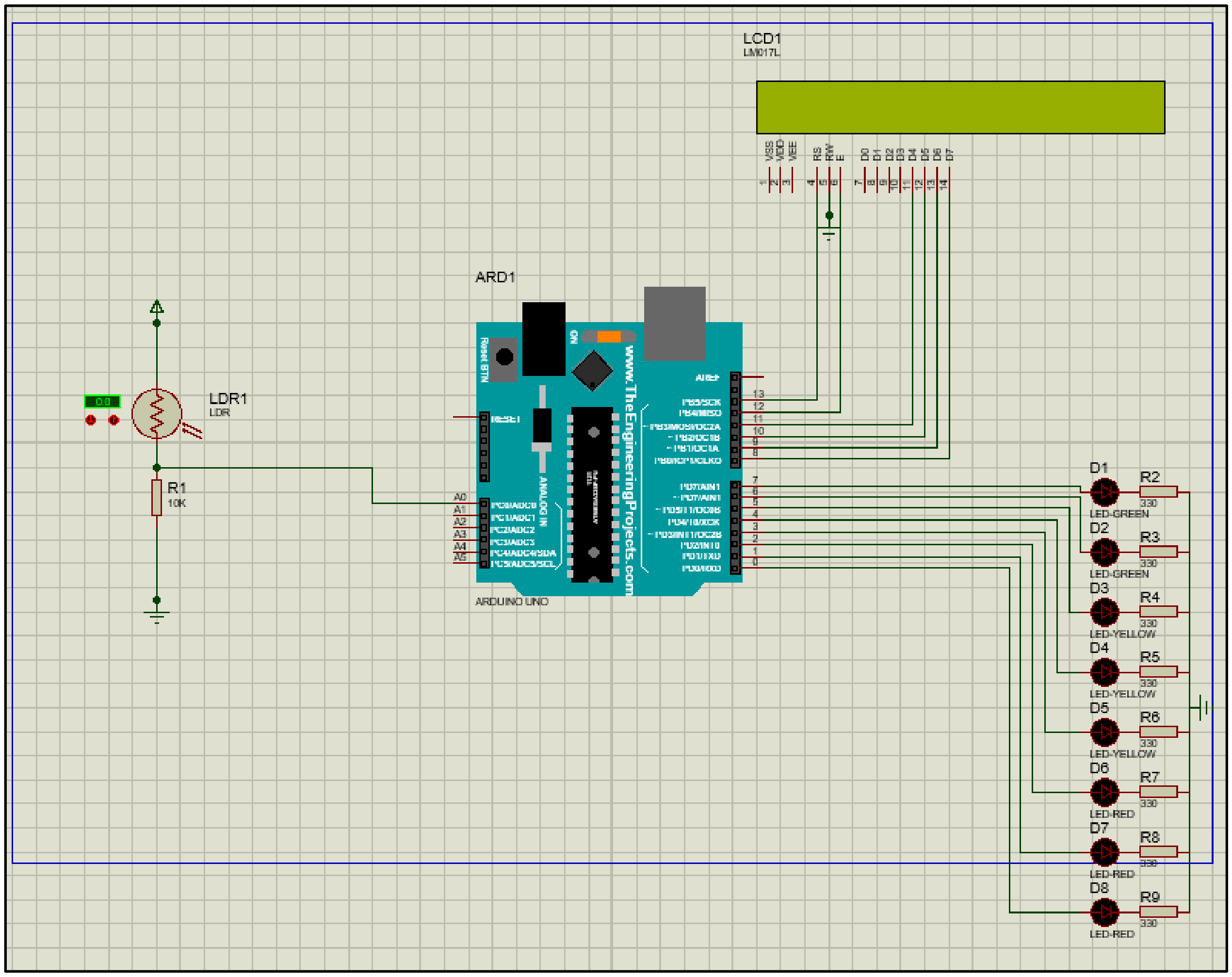
```
case 4:  // print 4
  digitalWrite(SegA, HIGH);
  digitalWrite(SegB, LOW);
  digitalWrite(SegC, LOW);
  digitalWrite(SegD, HIGH);
  digitalWrite(SegE, HIGH);
  digitalWrite(SegF, LOW);
  digitalWrite(SegG, LOW);
  break;

case 5:  // print 5
  digitalWrite(SegA, LOW);
  digitalWrite(SegB, HIGH);
  digitalWrite(SegC, LOW);
  digitalWrite(SegD, LOW);
  digitalWrite(SegE, HIGH);
  digitalWrite(SegF, LOW);
  digitalWrite(SegG, LOW);
  break;

case 6:  // print 6
  digitalWrite(SegA, LOW);
  digitalWrite(SegB, HIGH);
  digitalWrite(SegC, LOW);
  digitalWrite(SegD, LOW);
  digitalWrite(SegE, LOW);
  digitalWrite(SegF, LOW);
  digitalWrite(SegG, LOW);
  break;

case 7:  // print 7
  digitalWrite(SegA, LOW);
  digitalWrite(SegB, LOW);
  digitalWrite(SegC, LOW);
  digitalWrite(SegD, HIGH);
  digitalWrite(SegE, HIGH);
  digitalWrite(SegF, HIGH);
  digitalWrite(SegG, HIGH);
  break;
```

```
      case 8:  // print 8
        digitalWrite(SegA, LOW);
        digitalWrite(SegB, LOW);
        digitalWrite(SegC, LOW);
        digitalWrite(SegD, LOW);
        digitalWrite(SegE, LOW);
        digitalWrite(SegF, LOW);
        digitalWrite(SegG, LOW);
        break;

      case 9:  // print 9
        digitalWrite(SegA, LOW);
        digitalWrite(SegB, LOW);
        digitalWrite(SegC, LOW);
        digitalWrite(SegD, LOW);
        digitalWrite(SegE, HIGH);
        digitalWrite(SegF, LOW);
        digitalWrite(SegG, LOW);
    }
  }

void disp_off()
{
   digitalWrite(Dig1, HIGH);
   digitalWrite(Dig2, HIGH);
   digitalWrite(Dig3, HIGH);
   digitalWrite(Dig4, HIGH);
}

// end of code.
```

Application of ADC:

Sometimes, we can't measure intensity of light just by looking at it, so we can use ADC in that place to determine intensity and convert that to binary format. The process is same as above method where instead of rheostat, we use LDR (Light Dependent resistor)

Simulation in Proteus:

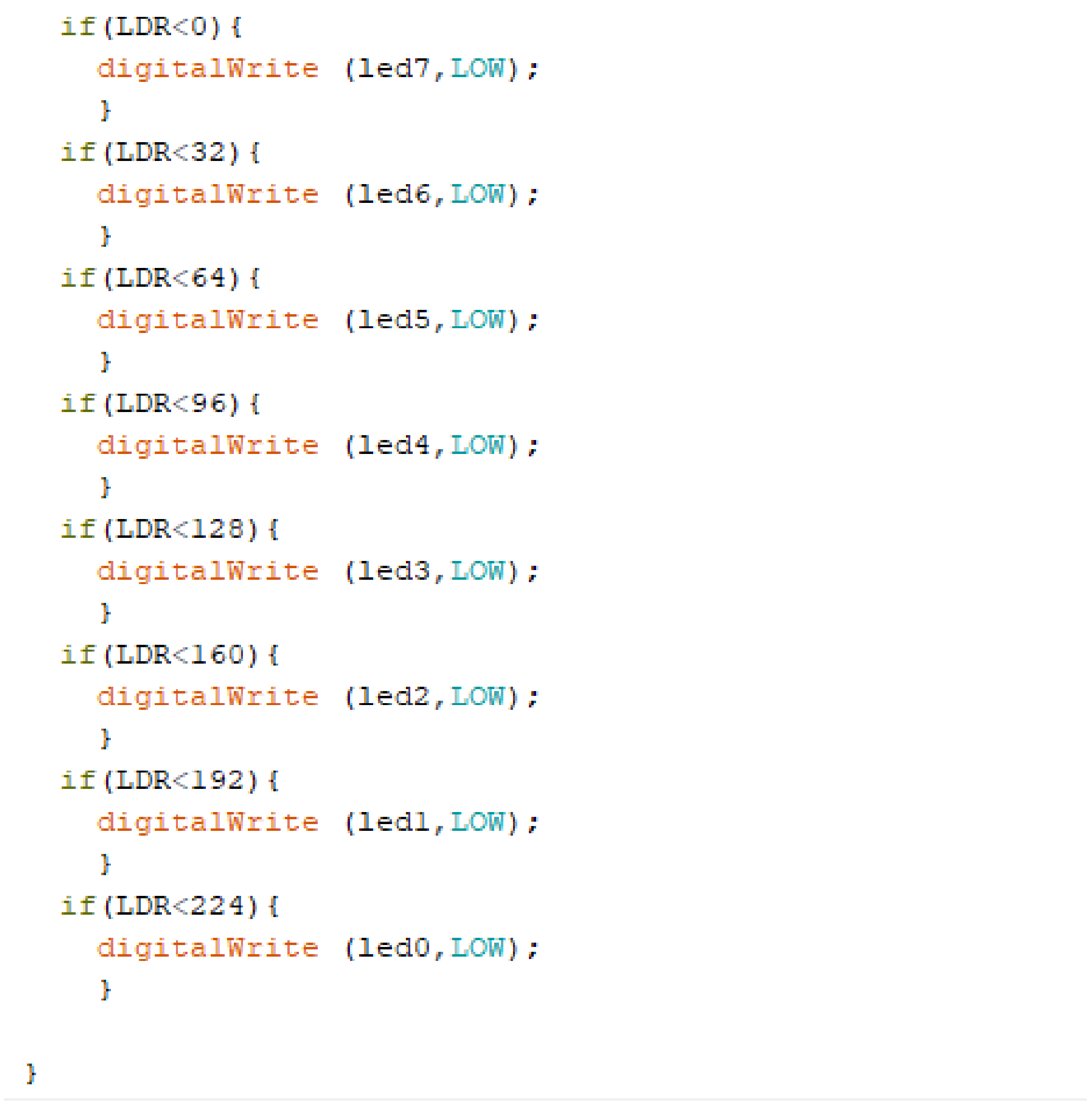


Here, the LDR glows based on value of resistor, as Intensity and Resistance are correlated, the voltage across Analog Pin depends on Intensity of light, and then the intensity is shown on LED lights D1 -D8 as D8 D7 D6 D5 D4 D3 D2 D1, representing 8-bit binary number.

Code:

```
#include <LiquidCrystal.h>
int led7 = 7;
int led6 = 6;
int led5 = 5;
int led4 = 4;
int led3 = 3;
int led2 = 2;
int led1 = 1;
int led0 = 0;
LiquidCrystal lcd(13,12,11,10,9,8);
void setup() {
  // put your setup code here, to run once:
  lcd.begin(16,2);

  pinMode(led0, OUTPUT);
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  pinMode(led3, OUTPUT);
  pinMode(led4, OUTPUT);
  pinMode(led5, OUTPUT);
  pinMode(led6, OUTPUT);
  pinMode(led7, OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  int LDR = analogRead(A0);
  int temp = map(LDR,0,1023,0,255);
  lcd.setCursor(0,0);
  lcd.print("ADC data:");
  lcd.setCursor(0,1);
  lcd.print(temp);
  delay(10);
  lcd.clear();
```

```
if(LDR==0){
  digitalWrite(led1, LOW);
  digitalWrite(led2, LOW);
  digitalWrite(led3, LOW);
  digitalWrite(led4, LOW);
  digitalWrite(led5, LOW);
  digitalWrite(led6, LOW);
  digitalWrite(led7, LOW);
  }
if(LDR>0){
  digitalWrite (led7,HIGH);
  }
if(LDR>32){
  digitalWrite (led6,HIGH);
  }
if(LDR>64){
  digitalWrite (led5,HIGH);
  }
if(LDR>96){
  digitalWrite (led4,HIGH);
  }
if(LDR>128){
  digitalWrite (led3,HIGH);
  }
if(LDR>160){
  digitalWrite (led2,HIGH);
  }
if(LDR>192){
  digitalWrite (led1,HIGH);
  }
if(LDR>224){
  digitalWrite (led0,HIGH);
  }
```



```
  if(LDR<0){
    digitalWrite (led7,LOW);
    }
  if(LDR<32){
    digitalWrite (led6,LOW);
    }
  if(LDR<64){
    digitalWrite (led5,LOW);
    }
  if(LDR<96){
    digitalWrite (led4,LOW);
    }
  if(LDR<128){
    digitalWrite (led3,LOW);
    }
  if(LDR<160){
    digitalWrite (led2,LOW);
    }
  if(LDR<192){
    digitalWrite (led1,LOW);
    }
  if(LDR<224){
    digitalWrite (led0,LOW);
    }

}
```

CONCLUSION:

In conclusion, we were able to design a digital to analog converter using two different circuits. A DAC using R-2R Ladder was designed and implemented having only two different values of resistors, thus having an advantage based on the number of components unlike the binary weighted resistors DAC wherein different resistor values were used hence having difficulty in attaining the needed output required. In addition to that, both circuits were monotonic.

Nevertheless, both circuits were properly designed, fabricated, tested and compared in this research and in conclusion, the binary weighted circuit is considered as a much more efficient DAC. We had also seen a buffer implementation of the same.

We were able to successfully implement the 4 bit DAC as well on magic(CMOS level realization) .We also built a signal generator and showed the three forms of output.

ADC has reverse mechanism of DAC and it has its own importance, In this project we have used Arduino as a medium to be used in place of ADC but we can also design it using simple elements which is beyond the scope of our project, Finally we have learnt how a Convertor works and its importance in industrial applications.

All the circuits built in this project were properly designed and produced the values as expected from the start and hence the project has been implemented successfully

References-

1. Study and simulation of ADC convertor 8 bits[Diouba sacko,Alpha Amadou Keita,Drissa Traore] (IJNTR)-2018
   https://media.neliti.com/media/publications/263159-study-and-simulation-of-an-analogical-to-a7cf5f52.pdf

2. Digital to Analog Convertor [Raghu Tumati]-2006
   https://ece.umaine.edu/wp-content/uploads/sites/203/2012/05/ECE547_RaghuTumati.pdf

3. DAC and ADC-1998
   https://www.semanticscholar.org/paper/Analog-To-Digital-(Adc)-And-Digital-To-Analog-(Dac)-Rabiee/f01f5d15d5c4ca0bf671910c02a05231100ba750

4. Arduino
   https://atmega32-avr.com/

5. Buffer Circuits
   https://www.youtube.com/watch?v=52ZR7jrOuW0
