LIST OF EXPERIMENTS

1. Verification of Ohm's law & Kirchhoff's law.
2. Calculate the individual branch currents and total current drawn from the power supply for the following set of resistors connected together in a parallel using current and voltage division rules.
3. Verification of star-delta transformation using resistance reduction technique.
4. Verification of Thevenin's and Norton's Theorems.
5. Verification of Superposition and Maximum power transfer Theorems.
6. Load test on Single Phase Transformer.
7. To obtain equivalent circuit, efficiency and voltage regulation of a single phase transformer using O.C. and S.C. tests.
8. Calculation of Secondary turns and Current in a transformer.
9. Load test on Single phase Induction Motor.
10. To determine the output characteristics of LVDT and calibrate the measuring instruments.
11. Power measurement using two wattmeter methods.
12. Calculate the energy consumption using the Energy meter.
13. Load test on DC shunt Motor.
14. Staircase Wiring & Fluorescent tube wiring
15. Find Stability of a System Using Routh Hurwitz Criterion.
16. Investigating the Performance of Three-Phase Induction Motor Drive Systems in Electric Vehicle Applications.
17. Write SCILAB program to generate the following signals:
    (a) Unit step signal
    (b) Unit Impulse signal
    (c) Unit ramp signal
    (d) Sinusoidal signal
    (e) Exponential signal
18. Write a SCILAB program to obtain the following:
    (a) DIT-FFT Algorithm
    (b) DIF-FFT Algorithm
19. Design a filter using the Transformation Method.

(a) Bilinear Transformation

(b) Impulse Invariant Transformation

20. Write the SCILAB program to design the following Butterworth filters
(a)Low pass filter

(b)High pass filter

(c)Band pass filter

(d)Band reject filter.

Exp: 15 FIND STABILITY OF A SYSTEM USING ROUTH HURWITZ CRITERION

Aim:

To determine the stability of the closed-loop system using Routh Hurwitz Criterion for the given polynomial characteristics equations.

(i) $(s)=s^4+2s^3+3s^2+4s+5$ and
(ii) $(s)=s^5+7s^4+6s^3+42s^2+8s+56$

Tools Used:

(i) SCILAB software

(ii) PC

Program code:

```
clear;
clc;
xdel(winsid());
mode(0);
s=%s;
H=s^4+2*s^3+3*s^2+4*s+5;
//H=s^5+7*s^4+6*s^3+42*s^2+8*s+56;
disp(H,'The given characteristics equation 1-G(s)H(s)=');
c=coeff(H);
len=length(c);
r=routh_t(H);
disp(r,"Rouths table=");
x=0;
for i=1:len
if(r(i,1)<0)
x=x+1;
end
end
if(x>=1)
printf("From Rouths table, it is clear that the system is unstable.")
else
printf("From Rouths table, it is clear that the system is stable.")
end
```

Simulation output:

```
Scilab 6.0.2 Console

 The given characteristics equation 1-G(s)H(s)=

            2    3   4
   5 +4s +3s  +2s  +s

 Rouths table=

   1.    3.    5.
   2.    4.    0.
   1.    5.    0.
  -6.    0.    0.
   5.    0.    0.
From Rouths table, it is clear that the system is unstable.
-->
```

```
Scilab 6.0.2 Console

 The given characteristics equation 1-G(s)H(s)=

              2    3    4    5
   56 +8s +42s  +6s  +7s  +s

 Rouths table=

   1.          6.    8.
   7.          42.   56.
   28.         84.   0.
   21.         56.   0.
   9.3333333   0.    0.
   56.         0.    0.
From Rouths table, it is clear that the system is stable.
-->
```

Marks Obtained:

Theoretical Calculations	**20**	
Observation	**20**	
Execution of practice examples	**30**	
Viva	**10**	
Record	**20**	
Total Score	**100**	
Date of experiment		**Faculty signature**
Date of record submission		

RESULT:

Thus the stability of a system using routh hurwitz criterion is verified.

Aim: To Study the Function of Three Phase Induction Motor Drive Systemin E-Mobility.

Requirements:

1. 750W Three Phase Induction Motor	1 Nos.
2. Panel with DC Voltmeter, DC Ammeter & AC Meter	1 Nos.
3. 750W Variable Frequency AC Drive (VFD)	1 Nos.
4. Battery Emulator/DC Drive	1 Nos.
5. Human Machine Interface (HMI)	1 Nos.
6. 12 V Alternator with 12V Batteries for Electrical Loading	1 Nos.

Theory:

Three-phase AC induction motors play a very important role in industry due to its low price and simplicity. The induction motor is used to convert three-phase AC power into mechanical power. When the load on an induction motor increases the percentage of slip is increase, which leads to decrease the speed of induction motor while constant speed in industryis very important. The speed of induction motor is controlled by variable frequency drive (VFD) automatically through controller. It is simple and efficient method to control the speed because speed depends upon voltage, pole and frequency. Poles are fix inbuilt in the motor so we cannotchange it and speed control through VFD is simple and energy efficient method. This technique implemented on hardware. So this technique is robust and simpler to implement.

Synchronous Speed:

When Alternating current (AC) is applied to the stator of a three phase motor, a rotating magnetic field is setup. This rotating magnetic field moves with a speed called synchronous speed. The Synchronous speed can be calculated as follows: 120 times the frequency (F), divided by the number of poles (P):

$$N_s = \frac{120F}{P}$$

The synchronous speed decreases as the number of poles increases. Thetable below shows the synchronous speed associated with various numbers of poles at supply frequencies of 50Hz and 60Hz:

No. of Poles	Synchronous Speed @ 50Hz	Synchronous Speed @ 60Hz
2	3000	3600
4	1500	1800
6	1000	1200
8	750	900
12	500	600

Rated Speed: The speed of operation of an AC motor when fully loaded at rated voltage is called the rated speed. It is usually given in RPM an electric motor nameplate. The rated speed is the speed of the rotor.**Slip**: The speed of the rotor magnetic field in an induction motor lags slightly behind the synchronous speed of the changing stator magnetic field. This difference in speed between rotor and stator fields is called slipand is measured in %.The slip of an AC motor is a key factor and is necessary to produce torque. The greater the load (torque), the greater slip will be

The formula for calculating motor slip is given by:

$$Slip = \frac{N_S - N}{N_S}$$

Where:

$N_s = Synchronous - speed$

$N = Rotor - speed$

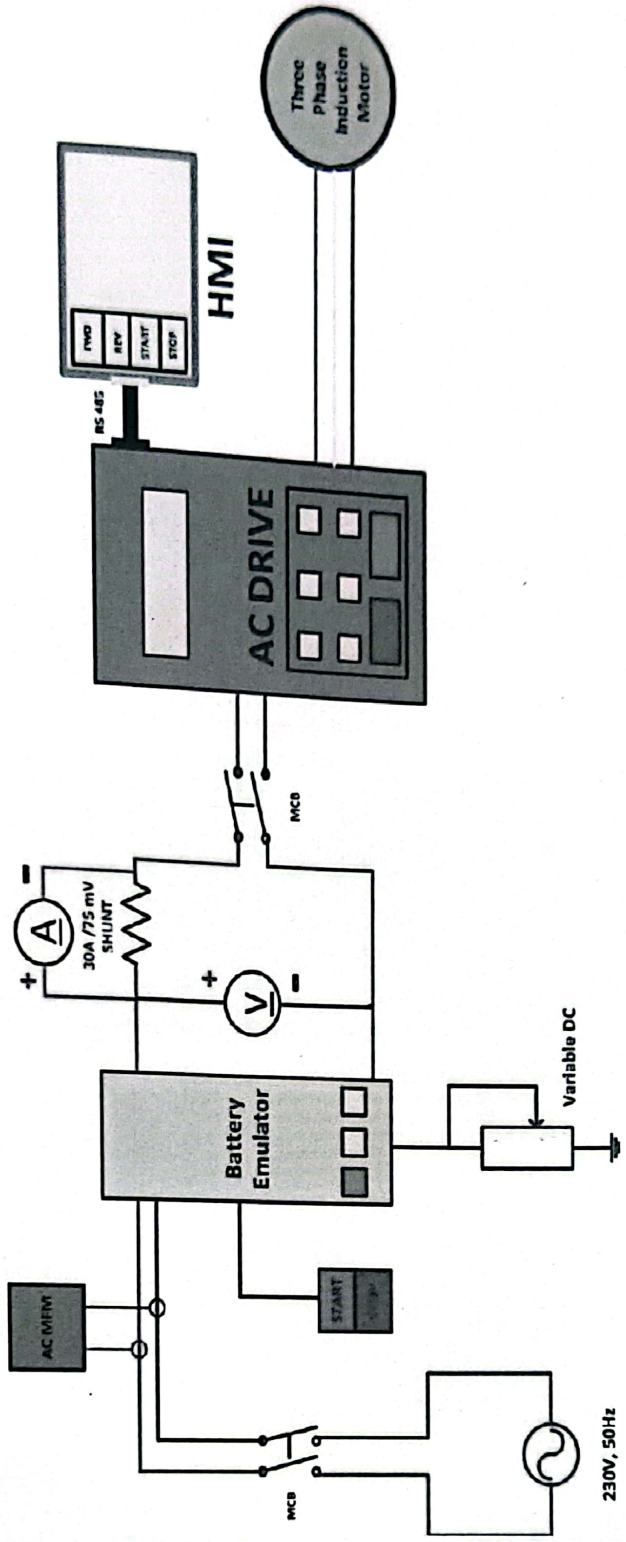
Note that rotor speed is the same as the rated speed of the AC motor as given on the motor nameplate.

The two main features of variable frequency drive are **adjustable speeds and soft start/stop** capabilities. These two features make VFD's a powerful controller to control the AC motors in industrial terms, **AC drive is also called as variable frequency drive (VFD), variable speed drive (VSD), or adjustable speed drive (ASD), Adjustable frequency drive (AFD), variable voltage variable frequency drive (VVVFD).**

Though there are different types of VFDs (or AC drives), all of them are works on same principle that converting fixed incoming voltage and frequency into variable voltage and frequency output. The **frequency** of the drive determines the how fast motor should **run** while the combinationof **voltage (V) and frequency (F)** decides the amount of **torque** that the motor to generate.

Procedure:

Three Phase AC Induction Motor Based Model Connection Diagram



1. Give AC Supply to Meter(s), Battery Emulator/DC Drive & HMI etc.
2. Fully Rotate Potentiometer of DC Control connected with Battery Emulator.
3. Select Forward/Reverse Mode Using HMI.
4. Vary the Frequency Using VFD.

5. Connect 12V Battery in Excitation Panel **of Alternator** according to Connection Diagram & Make sure to Switch on the Toggle switch
6. Calculate different values given on the below Table to Understand the working Characteristics of VFD & Three Phase Induction Motor Drive System.

Cautions:

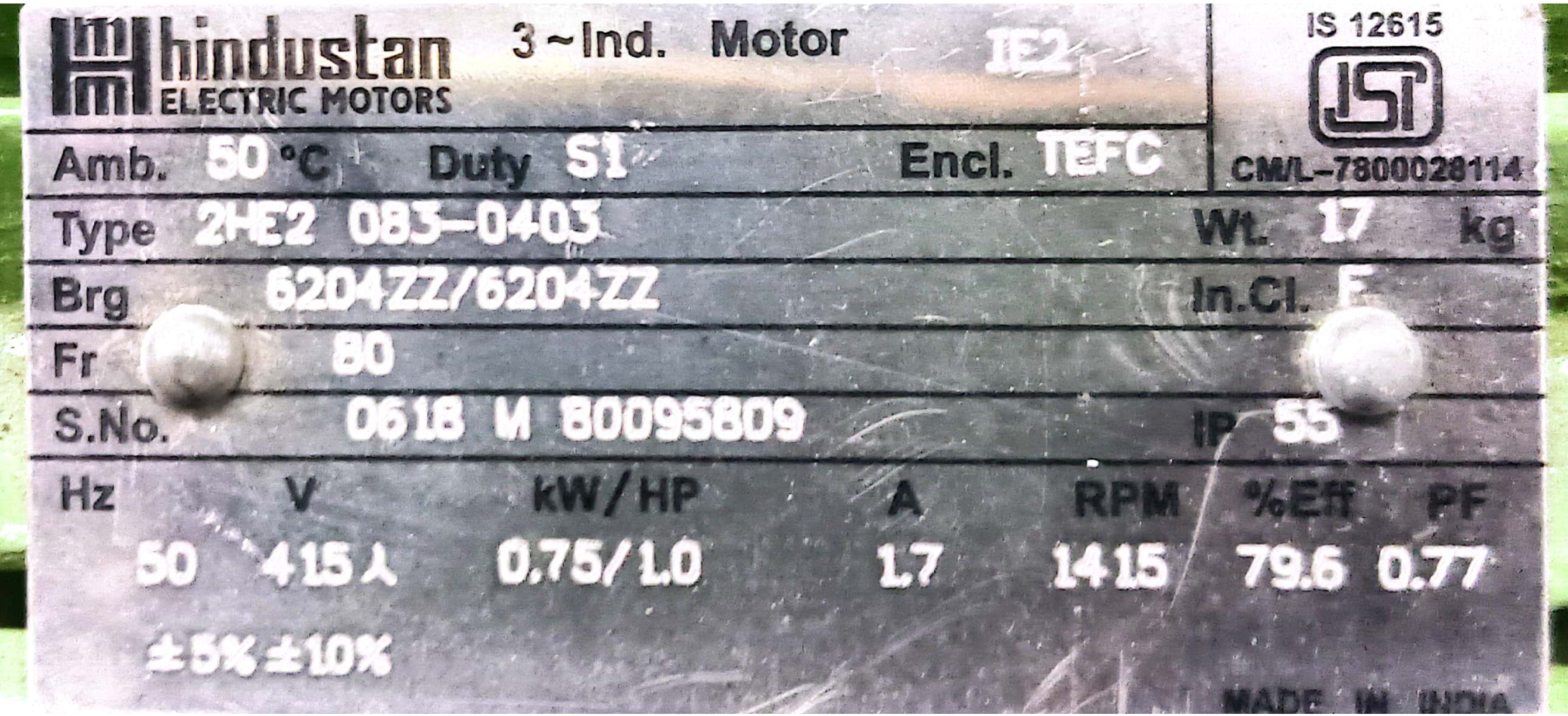
1. Do not connect overvoltage supply.
2. Check proper connection before testing.
3. Do not overload the Motor.

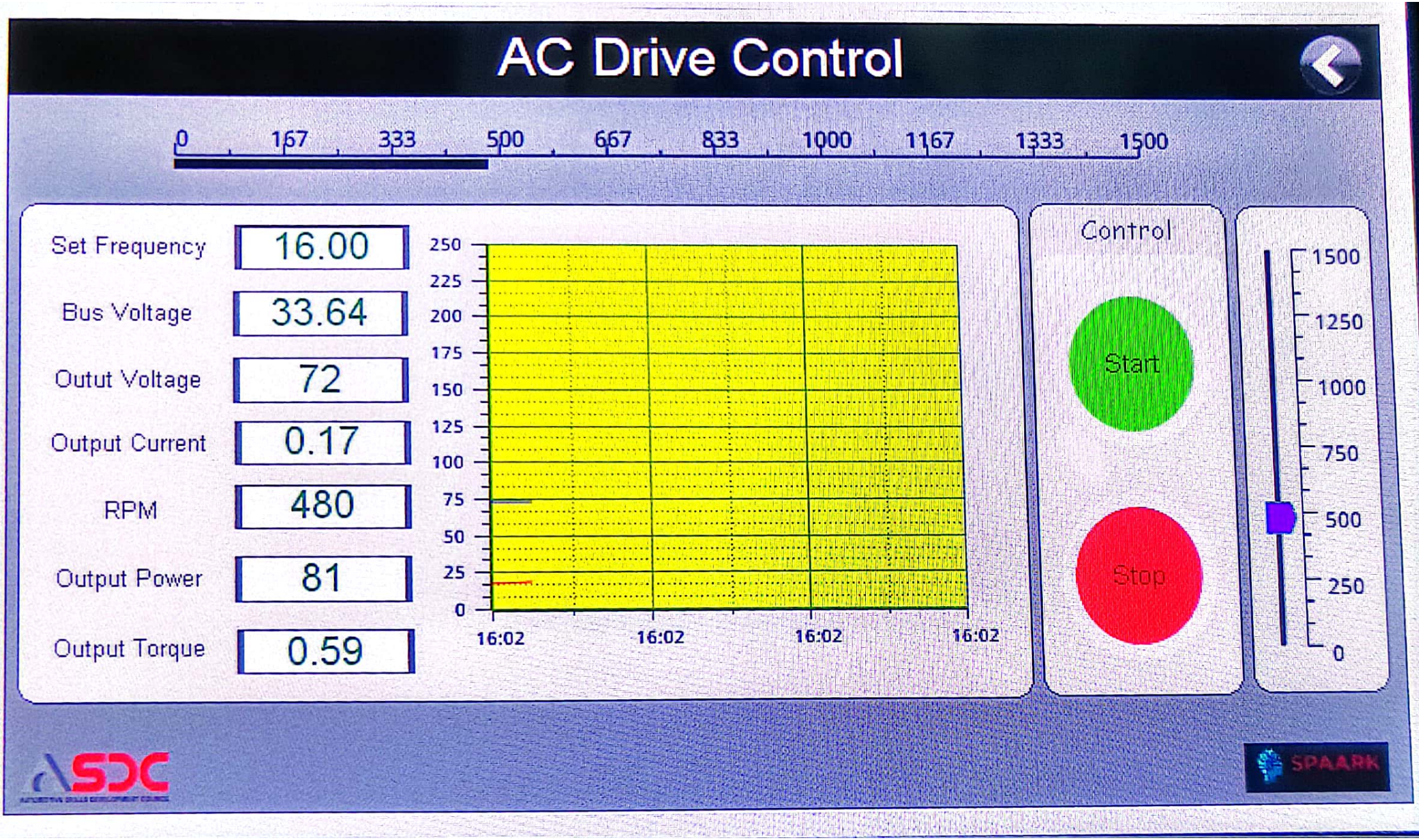
Calculations:

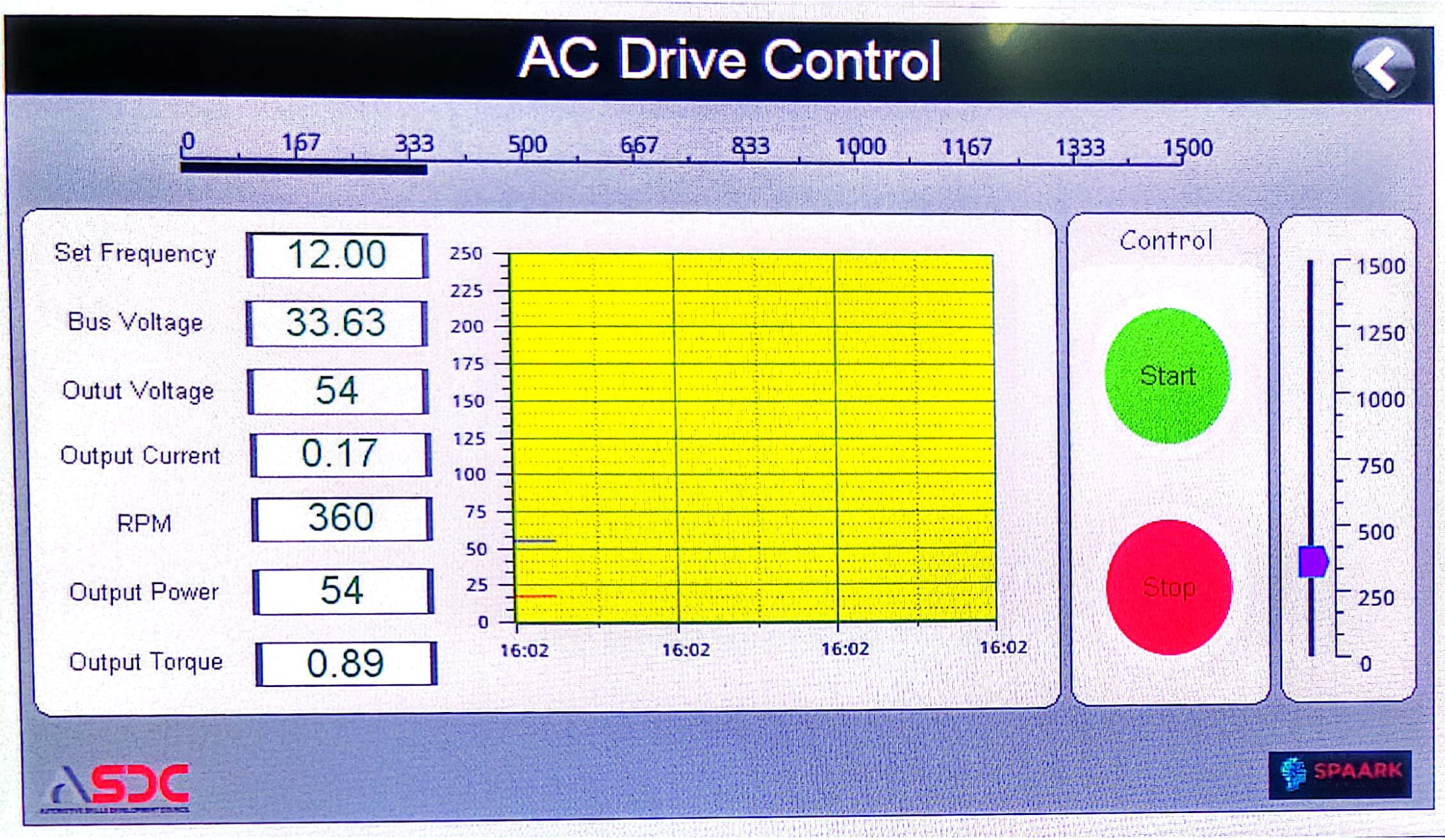
S. No.	Frequency (f)	Output Voltage (V)	Ratio = V/f	RPM Calculated (Ns=120f/P)	Tachometer (Rotor) RPM	Slip
1						
2						
3						

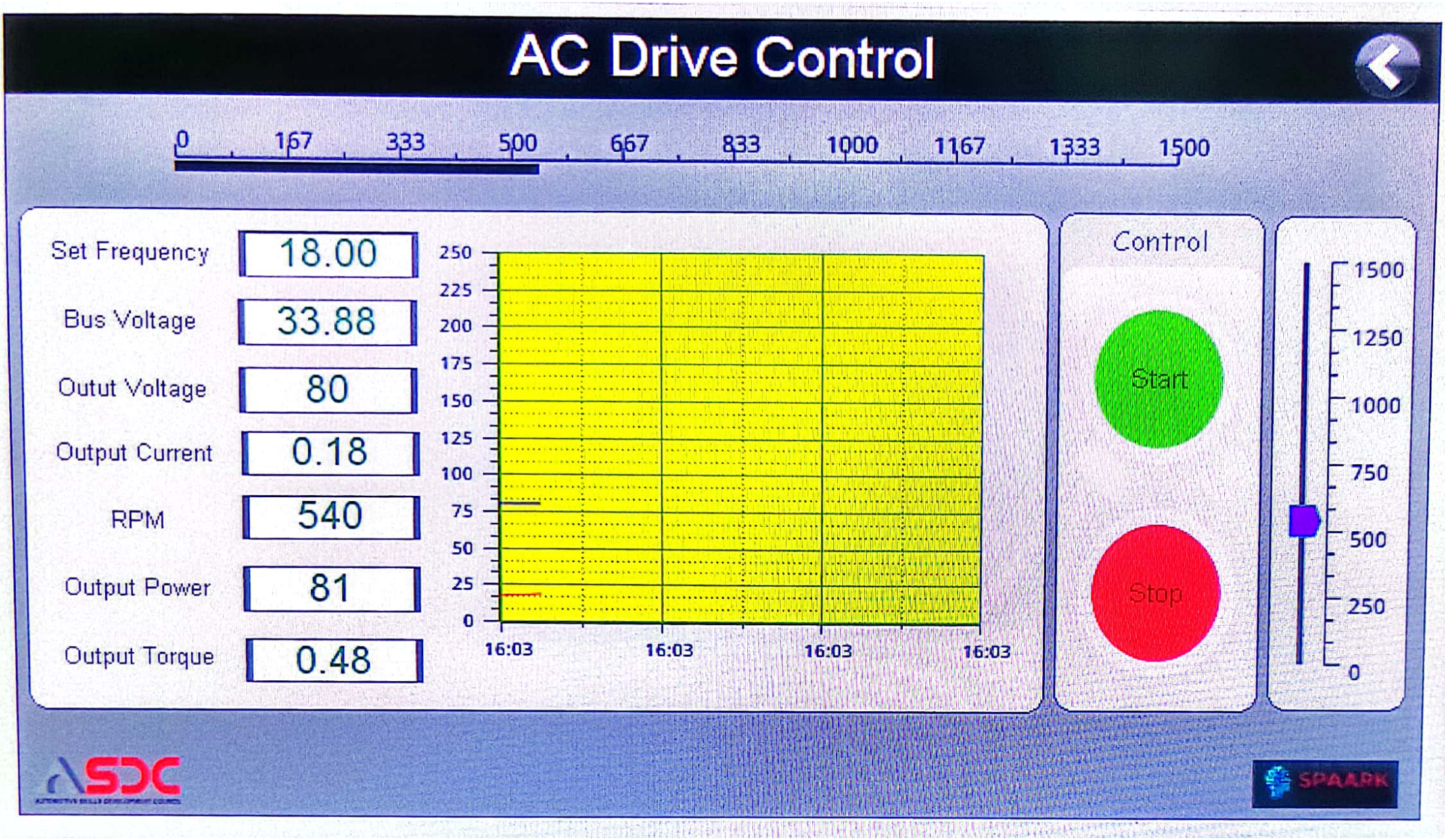
Load/ Charging current Across Alternator 0 V	Load/ Charging current Across Alternator 12 V Battery 1 No	Load/ Charging current Across Alternator 12 V Batteries 2 Nos in parallel

Result: Thus, the test on Three Phase Induction Motor Drive System inE-Mobility is Performed Successfully.


hindustan
ELECTRIC MOTORS
3~Ind. Motor
IE2
IS 12615
CM/L-7800028114
Amb. 50°C Duty S1 Encl. TEFC
Type 2HE2 083-0403 Wt. 17 kg
Brg 6204ZZ/6204ZZ In.Cl. F
Fr 80
S.No. 0618 M 80095809 IP 55
Hz V kW/HP A RPM %Eff PF
50 415Δ 0.75/1.0 1.7 1415 79.6 0.77
±5% ±10%
MADE IN INDIA

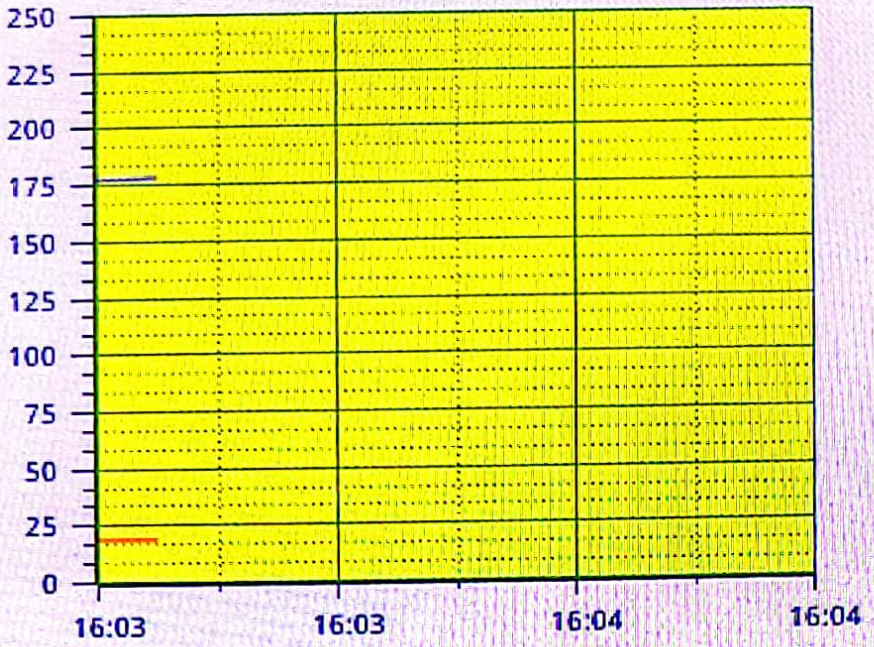

AC Drive Control
0
167
333
500
667
833
1000
1167
1333
1500
Set Frequency
16.00
Bus Voltage
33.64
Outut Voltage
72
Output Current
0.17
RPM
480
Output Power
81
Output Torque
0.59
250
225
200
175
150
125
100
75
50
25
0
16:02
16:02
16:02
16:02
Control
Start
Stop
1500
1250
1000
750
500
250
0
SPAARK


AC Drive Control
0
167
333
500
667
833
1000
1167
1333
1500
Set Frequency
12.00
Bus Voltage
33.63
Outut Voltage
54
Output Current
0.17
RPM
360
Output Power
54
Output Torque
0.89
250
225
200
175
150
125
100
75
50
25
0
16:02
16:02
16:02
16:02
Control
Start
Stop
1500
1250
1000
750
500
250
0
ASDC
SPAARK


AC Drive Control
0
167
333
500
667
833
1000
1167
1333
1500
Set Frequency
18.00
Bus Voltage
33.88
Outut Voltage
80
Output Current
0.18
RPM
540
Output Power
81
Output Torque
0.48
250
225
200
175
150
125
100
75
50
25
0
16:03
16:03
16:03
16:03
Control
Start
Stop
1500
1250
1000
750
500
250
0
SDC
SPARK

AC Drive Control

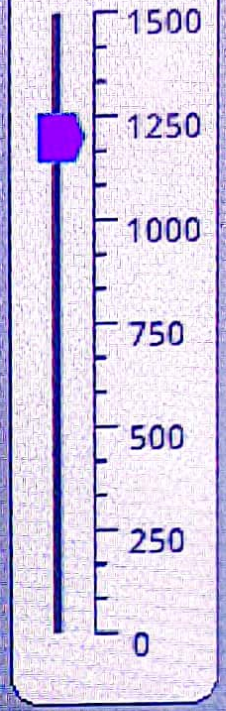
Set Frequency	40.00
Bus Voltage	33.57
Outut Voltage	177
Output Current	0.18
RPM	1200
Output Power	122
Output Torque	0.41



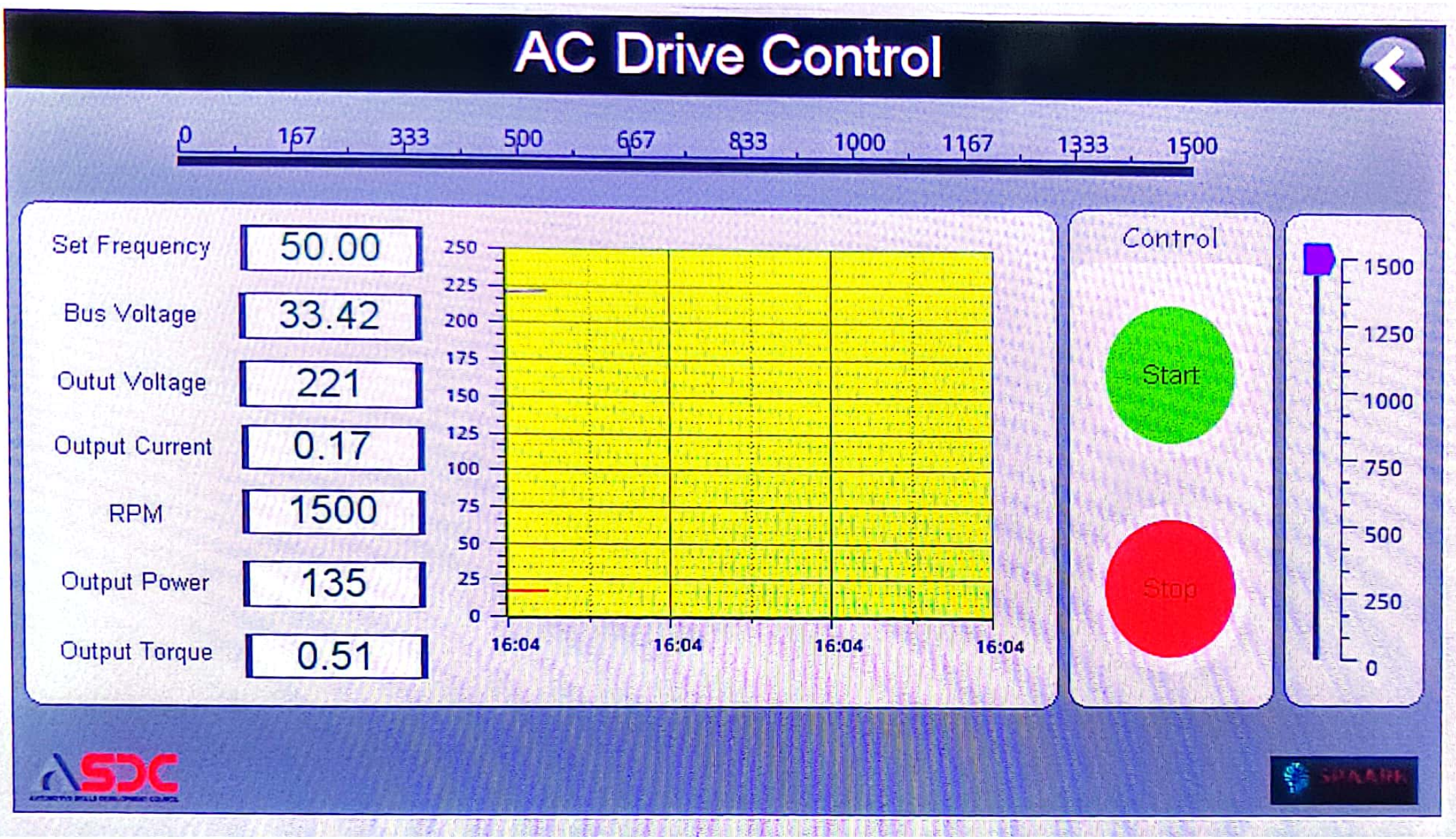
Control

Start

Stop



SPAARK


AC Drive Control
0
167
333
500
667
833
1000
1167
1333
1500
Set Frequency
50.00
Bus Voltage
33.42
Outut Voltage
221
Output Current
0.17
RPM
1500
Output Power
135
Output Torque
0.51
250
225
200
175
150
125
100
75
50
25
0
16:04
16:04
16:04
16:04
Control
Start
Stop
1500
1250
1000
750
500
250
0

EX NO:17	Generation of Common Discrete Time Signals
DATE:	

Aim

Generate and plot the Unit Impulse Signal, Unit Step Signal,Unit Ramp Signal, Sinusoidal Signal,Exponential Signal: For each signal Write the SCILAB code to generate the signal.Plot the signal and label the axes appropriately.

using SCILAB:

Unit Impulse Signal, Unit Step Signal,Unit Ramp Signal, Sinusoidal Signal,Exponential Signal: **For each signal** Write the SCILAB code to generate the signal.Plot the signal and label the axes appropriately.

Software Required

1. Scilab 6.1.0

Procedure:

Start the scilab Program
Open scinotes ,type the program and save the program in current directory
Compile and run the program
If any error occur in the program,correct the error and run the program
For the output ,see the console window
Stop the program

Theory:

Discrete-time signals are a fundamental concept in digital signal processing and communication. They represent variations in amplitude over discrete points in time. Here, I'll provide a brief overview of some common discrete-time signals and their characteristics:

Unit Step Signal (u[n]): The unit step signal is a basic discrete-time signal that takes the value 1 for non-negative time indices (n >= 0) and 0 otherwise. It is often used to model the onset of events or changes in a system.

Impulse Signal (δ[n]): The impulse signal is also known as the discrete-time delta function. It has a value of 1 at n = 0 and is zero for all other time indices. It's a

fundamental signal in signal processing and is used to represent discrete-time impulses or impulses in discrete-time systems.

Exponential Signal: An exponential discrete-time signal is defined as x[n] = A * α^n, where A is the amplitude and α is the exponential factor. Depending on whether α is greater or less than 1, the signal can grow or decay exponentially with time.

Sinusoidal Signal: A sinusoidal discrete-time signal has the form x[n] = A * cos(ωn + φ), where A is the amplitude, ω is the angular frequency, n is the time index, and φ is the phase shift. Sinusoidal signals exhibit periodic behavior and are a fundamental representation of oscillations.

Ramp Signal: A ramp signal is a linearly increasing or decreasing signal with time. It's given by x[n] = a * n, where 'a' is the slope of the ramp. The ramp signal is commonly used to model linear changes or trends.

Random Signal: A random signal represents random variations or noise in a system. It's often generated using a random number generator. In digital communication, noise can introduce errors in the received signal, impacting the quality of communication.

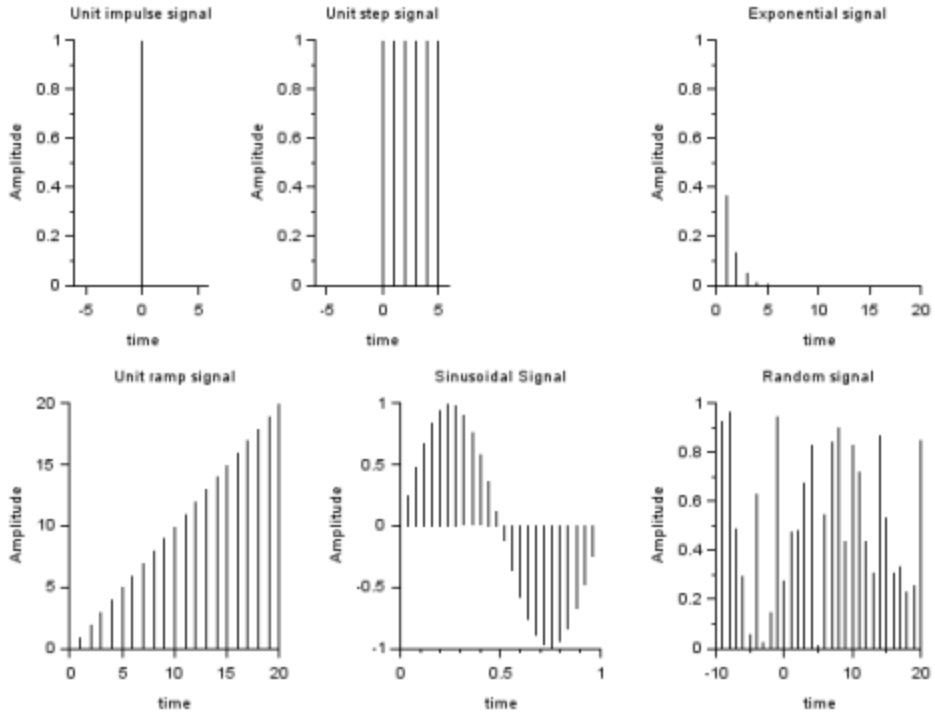
Program:

```
//UNIT IMPULSE SIGNAL
clear all;
close ;
N =5; //SET LIMIT
t1 = -5:5;
x1 =[ zeros (1 , N ) ,ones (1 ,1) ,zeros (1 , N ) ];
subplot (2 ,4 ,1) ;
plot2d3 ( t1 , x1 )
xlabel ( ' tim e ' ) ;
ylabel ( ' Ampli tude ' ) ;
title ( ' Uni t im p ul s e s i g n a l ' ) ;
//UNIT STEP SIGNAL
t2 = -5:5;
x2 =[ zeros (1 , N ) ,ones (1 , N +1) ];
subplot (2 ,4 ,2) ;
plot2d3 ( t2 , x2 )
xlabel ( ' tim e ' ) ;
ylabel ( ' Ampli tude ' ) ;
title ( ' Uni t s t e p s i g n a l ' ) ;
//EXPONENTIAL SIGNAL
t3 =0:1:20;
```

```
x3 =exp( - t3 ) ;
subplot (2 ,3 ,3) ;
 plot2d3 ( t3 , x3 ) ;
 xlabel ( ' tim e ' ) ;
 ylabel ( ' Ampli tude ' ) ;
 title ( ' E x p o n e n t i a l s i g n a l ' ) ;

 //UNIT RAMP SIGNAL 4
t4 =0:20;
 x4 = t4 ;
 subplot (2 ,3 ,4) ;
 plot2d3 ( t4 , x4 ) ;
 xlabel ( ' tim e ' ) ;
 ylabel ( ' Ampli tude ' ) ;
 title ( ' Uni t ramp s i g n a l ' ) ;
 //SINUSOIDAL SIGNAL
 t5 =0:0.04:1;
 x5 =sin (2* %pi * t5 ) ;
 subplot (2 ,3 ,5) ;
 plot2d3 ( t5 , x5 ) ;
 title ( ' S i n u s o i d a l S i g n a l ' )
xlabel ( ' tim e ' ) ;
 ylabel ( ' Ampli tude ' ) ;
 //RANDOM SIGNAL
 t6 = -10:1:20;
 x6 = rand (1 ,31) ;
 subplot (2 ,3 ,6) ;
 plot2d3 ( t6 , x6 ) ;
 xlabel ( ' tim e ' ) ;
 ylabel ( ' Ampli tude ' ) ;
 title ( ' Random s i g n a l ' ) ;
```

Output:



Result:

The following discrete-time signals were successfully generated and plotted using SCILAB:For each signal, the plots were labeled appropriately, with the x-axis representing time (n) and the y-axis representing the signal values.

EX NO: 18	DIT-FFT and DIF-FFT Algorithm
DATE:	

(i) Given a sequence x[n]=[1,-1,-1,-1,1,1,1,-1] compute the DFT using the DIT-FFT algorithm. The sequence exhibits real and symmetric properties. How does symmetry affect the twiddle factor calculations in the DIT-FFT algorithm?

Aim

(i) To Compute the DFT of given Sequence x[n]=[1,-1,-1,-1,1,1,1,-1] using DIT-FFT Algorithm.

Software Required

Scilab 6.1.0

Procedure:

Start the scilab Program
Open scinotes ,type the program and save the program in current directory
Compile and run the program
If any error occur in the program,correct the error and run the program
For the output ,see the console window
Stop the program

Theory:

The Decimation-in-Time Fast Fourier Transform (DIT-FFT) algorithm is an efficient method to compute the Discrete Fourier Transform (DFT) of a sequence. The FFT algorithm reduces the computational complexity of the DFT from

The DFT of a sequence $x[n]$ of length N is given by:

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-j\frac{2\pi kn}{N}}, \quad k = 0, 1, 2, \ldots, N-1$$

DIT-FFT Algorithm

The DIT-FFT algorithm is a divide-and-conquer approach that breaks down the DFT computation into smaller parts, recursively decimating the sequence into smaller subsequences. The key idea behind DIT-FFT is to separate the original sequence into even-indexed and odd-indexed terms, recursively applying the FFT on these smaller sequences.

Program

```
clear;
clc ;
close ;
x = [1,-1,-1,-1,1,1,1,-1];
//FFT Computation
X = fft (x , -1);
disp(X,'X(z) = ');
```

Output

```
    column 1 to 5
0  - 1.4142136 + 3.4142136i    2. - 2.i    1.4142136 - 0.5857864i    4.
    column 6 to 8
1.4142136 + 0.5857864i    2. + 2.i  - 1.4142136 - 3.4142136i
```

Result: Thus the DFT of the sequence x[n]=[1,-1,-1,-1,1,1,1,-1] using the DIT-FFT algorithm results in the expected frequency components, confirming the correct implementation of the algorithm.

(ii) Given a sequence x[n]=[1,2,3,4,4,3,2,1] compute the DFT using the DIF-FFT algorithm. The sequence exhibits real and symmetric properties. How does symmetry affect the twiddle factor calculations in the DIF-FFT algorithm?

Aim

To compute the DFT of the sequence x[n]=[1,2,3,4,4,3,2,1] using the Decimation-in-Frequency (DIF) FFT algorithm, we will utilize the symmetry of the sequence to optimize the twiddle factor calculations.

Software Required

Scilab 6.1.0

Procedure:

Start the scilab Program
Open scinotes ,type the program and save the program in current directory
Compile and run the program
If any error occur in the program,correct the error and run the program
For the output ,see the console window
Stop the program

Theory:

DIF-FFT Algorithm

The Decimation-in-Frequency Fast Fourier Transform (DIF-FFT) is another efficient algorithm for computing the Discrete Fourier Transform (DFT). Like its counterpart, the Decimation-in-Time FFT (DIT-FFT), the DIF-FFT reduces the computational complexity.

The DFT of a sequence $x[n]$ of length N is defined as:

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-j\frac{2\pi kn}{N}}, \quad k = 0, 1, 2, \ldots, N-1$$

The DIF-FFT algorithm, like the DIT-FFT, is a divide-and-conquer approach. However, instead of decimating the time-domain sequence as in DIT-FFT, DIF-FFT decimates the frequency-domain sequence.

Program:

```
clear;
clc ;
close ;
x = [1,2,3,4,4,3,2,1];
//FFT Computation
X = fft (x , -1);
disp(X,'X(z) = ');
```

Output:

```
X(z) =
        column 1 to 5
   20.  - 5.8284271 - 2.4142136i    0  - 0.1715729 - 0.4142136i    0
        column 6 to 8
 - 0.1715729 + 0.4142136i    0  - 5.8284271 + 2.4142136i
```

Result: Thus the DFT of the sequence x[n]=[1,2,3,4,4,3,2,1] using the DIF-FFT algorithm results in the expected frequency components, confirming the correct implementation of the algorithm. Symmetry in the sequence reduces the number of unique twiddle factor calculations, enhancing the algorithm's efficiency.

EXNO:19	Analog Butterworth Filter
DATE:	

(i)Design a Butterworth filter to process audio signals that attenuates frequencies above 0.2 * pi and maintains a flat frequency response in the passband. Compare the output of the filtered signal with the original signal in both time and frequency domains. Plot both to verify the attenuation of high frequencies.

Aim

To design a Butterworth filter for processing audio signals that attenuates frequencies above 0.2π0.2\pi0.2π radians per sample and maintains a flat frequency response in the passband. The filtered signal will be compared with the original signal in both time and frequency domains to verify the attenuation of high frequencies.Software Required

Scilab 6.1.0

Procedure:

Start the scilab Program
Open scinotes ,type the program and save the program in current directory
Compile and run the program
If any error occur in the program,correct the error and run the program
For the output ,see the console window
Stop the program

Theory:

The Butterworth low-pass filter is a widely used type of analog filter that is known for its maximally flat frequency response in the passband, which means it has no ripples. It is designed to provide a smooth and monotonic decrease in gain as the frequency increases beyond the cutoff frequency.

Program:

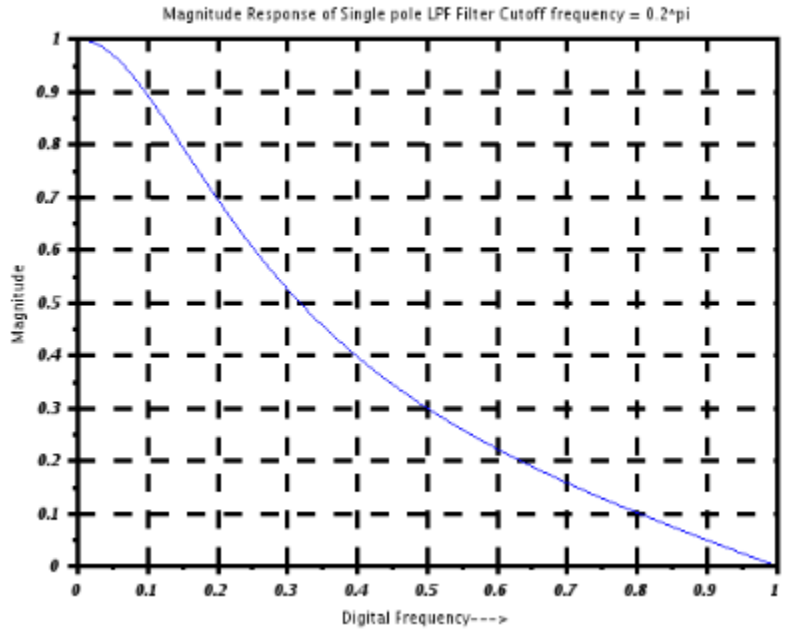
//First Order Butterworth Low Pass Filter

```
clear;
clc;
close;
s = poly(0,'s');
Omegac = 0.2*%pi;
H = Omegac/(s+Omegac);
T =1;//Sampling period T = 1 Second
```

```
z = poly(0,'z');
Hz = horner(H,(2/T)*((z-1)/(z+1)))
HW  =frmag(Hz(2),Hz(3),512);
W = 0:%pi/511:%pi;
plot(W/%pi,HW)
a=gca();
a.thickness = 3;
a.foreground = 1;
a.font_style = 9;
xgrid(1)
xtitle('Magnitude Response of Single pole LPF Filter Cutoff frequency =
0.2*pi','Digital Frequency--->','Magnitude');
Disp("Hz",Hz);
```

Output:

```
Hz  =
   0.6283185 + 0.6283185z
   ----------------------
 - 1.3716815 + 2.6283185z
```



Result:

The Butterworth filter successfully attenuated frequencies above 0.2π0.2\pi0.2π, confirming its design. The filter maintained a flat frequency response in the passband and

effectively reduced high-frequency noise, as observed in both the time and frequency domains.

(ii)Design a Butterworth filter that allows frequency range above the cut off frequency of 0.2*pi for a digital audio processing application.

Aim

The aim of this experiment is to design a Butterworth high-pass filter for a digital audio processing application, allowing frequencies above the cutoff frequency of 0.2×π0.2 \times \pi0.2×π radians per second to pass, while attenuating frequencies below the cutoff. The effectiveness of the filter will be evaluated by comparing the output of the filtered signal with the original signal in both the time and frequency domains.Software Required

Scilab 6.1.0

Procedure:

Start the scilab Program
Open scinotes ,type the program and save the program in current directory
Compile and run the program
If any error occur in the program,correct the error and run the program
For the output ,see the console window
Stop the program

Theory:

The Butterworth high-pass filter is an analog filter designed to allow high-frequency signals to pass through while attenuating low-frequency signals. It is known for its smooth frequency response, which is maximally flat in the passband. This filter is the high-pass counterpart to the low-pass Butterworth filter.

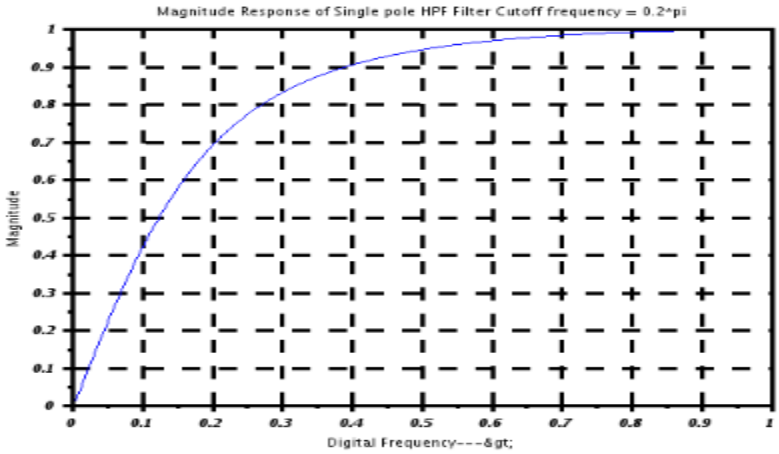
Program

```
//First Order Butterworth Filter
    //High Pass Filter Using Digital Filter Transformation
    clear;
    clc;
    close;
    s = poly(0,'s');
    Omegac = 0.2*%pi;
    H = Omegac/(s+Omegac);
    T =1;//Sampling period T = 1 Second
    z = poly(0,'z');
    Hz_LPF = horner(H,(2/T)*((z-1)/(z+1)));
    alpha = -(cos((Omegac+Omegac)/2))/(cos((Omegac-Omegac)/2));
    HZ_HPF=horner(Hz_LPF,-(z+alpha)/(1+alpha*z))
    HW  =frmag(HZ_HPF(2),HZ_HPF(3),512);
    W = 0:%pi/511:%pi;
```

```
plot(W/%pi,HW)
a=gca();
a.thickness = 3;
a.foreground = 1;
a.font_style = 9;
xgrid(1)
xtitle('Magnitude Response of Single pole HPF Filter Cutoff frequency =
0.2*pi','Digital Frequency---&gt;','Magnitude');
disp("HZ_HPF",HZ_HPF);
```

Output:

```
HZ_HPF  =
 - 0.7484757 + 0.7484757z
  ----------------------
   - 0.4969514 + z
```



Result:

The Butterworth high-pass filter was successfully designed with a cutoff frequency of 0.2×π0.2 \times \pi0.2×π. In the time domain,

(iii)For a signal processing application design a Butterworth filter to isolate a specific frequency range from an audio signal between 0.4 pi and 0.6 pi.

Aim

To design a Butterworth band-pass filter for a signal processing application to isolate a specific frequency range between 0.4π and 0.6π radians per second from an audio signal.

Software Required

Scilab 6.1.0

Procedure:

Start the scilab Program
Open scinotes ,type the program and save the program in current directory
Compile and run the program
If any error occur in the program,correct the error and run the program
For the output ,see the console window
Stop the program

Theory:

The Butterworth band-pass filter is an analog filter designed to pass frequencies within a certain range (the passband) while attenuating frequencies outside this range. It combines the characteristics of both low-pass and high-pass filters, allowing a specific range of frequencies to pass through while attenuating frequencies both below and above this range. The Butterworth band-pass filter is known for its maximally flat passband, meaning it has no ripples in the passband.

Program:

```
clear;
clc;
close;
omegaP = 0.2*%pi;
omegaL =  (2/5)*%pi;
omegaU =  (3/5)*%pi;
z=poly(0,'z');
H_LPF = (0.245)*(1+(z^-1))/(1-0.509*(z^-1))
alpha = (cos((omegaU+omegaL)/2)/cos((omegaU-omegaL)/2));
k = (cos((omegaU - omegaL)/2)/sin((omegaU - omegaL)/2))*tan(omegaP/2);
NUM =-((z^2)-((2*alpha*k/(k+1))*z)+((k-1)/(k+1)));
DEN = (1-((2*alpha*k/(k+1))*z)+(((k-1)/(k+1))*(z^2)));
HZ_BPF=horner(H_LPF,NUM/DEN)
disp(HZ_BPF,'Digital BPF IIR Filter H(Z)= ')
HW  =frmag(HZ_BPF(2),HZ_BPF(3),512);
```

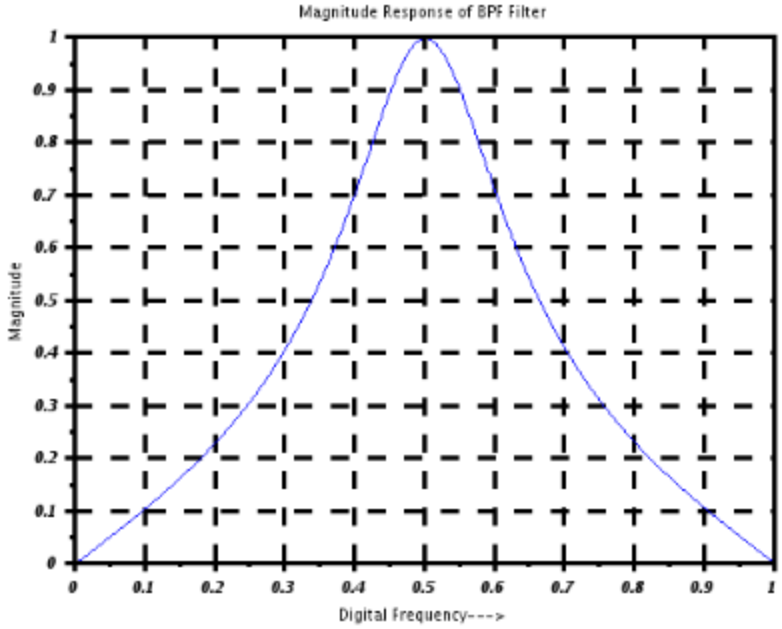
```
W = 0:%pi/511:%pi;
plot(W/%pi,HW)
a=gca();
a.thickness = 3;
a.foreground = 1;
a.font_style = 9;
xgrid(1)
xtitle('Magnitude Response of BPF Filter', 'Digital Frequency--->','Magnitude');
Disp("HZ_BPF",HZ_BPF);
```

Output:

```
H_LPF  =
   0.245 + 0.245z
   --------------
   - 0.509 + z

 HZ_BPF  =
                          2           3           4
   0.245 - 1.577D-17z - 0.245z + 1.577D-17z + 1.360D-17z
   ----------------------------------------------------
                        2           3           4
   - 0.509 + 1.299D-16z - z + 6.438D-17z + 5.551D-17z

Digital BPF IIR Filter H(Z)=
                          2           3           4
   0.245 - 1.577D-17z - 0.245z + 1.577D-17z + 1.360D-17z
   ----------------------------------------------------
                        2           3           4
   - 0.509 + 1.299D-16z - z + 6.438D-17z + 5.551D-17z
```



Result:

Thus the Butterworth band-pass filter was successfully designed to isolate the frequency range between 0.4π and 0.6π inn the time domain,

(iv) For a signal processing application design a Butterworth filter to attenuate a specific frequency range from an audio signal between 0.4 pi and 0.6 pi.

Aim

The aim of this experiment is to design a Butterworth band-stop filter for a signal processing application to attenuate a specific frequency range between 0.4π and 0.6π radians per second from an audio signal. using Sci lab

Software Required

Scilab 6.1.0

Procedure:

Start the scilab Program
Open scinotes ,type the program and save the program in current directory
Compile and run the program
If any error occur in the program,correct the error and run the program
For the output ,see the console window
Stop the program

Theory:

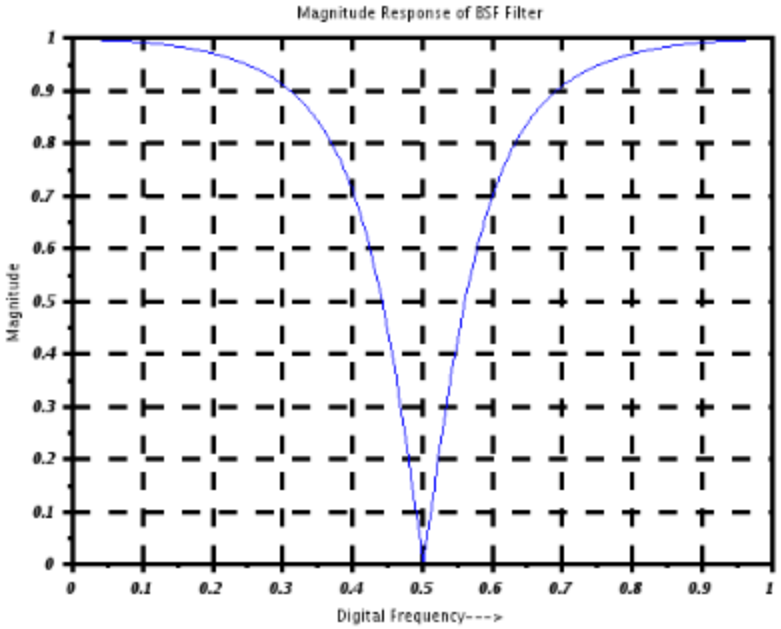
The Butterworth band-reject filter, also known as a band-stop or band-elimination filter, is an analog filter designed to attenuate frequencies within a specific range while allowing frequencies outside this range to pass with minimal attenuation. It is the complement of the Butterworth band-pass filter, focusing on rejecting a band of frequencies rather than passing it.

Program:

```
clear;
clc;
close;
omegaP = 0.2*%pi;
omegaL =  (2/5)*%pi;
omegaU =  (3/5)*%pi;
z=poly(0,'z');
H_LPF = (0.245)*(1+(z^-1))/(1-0.509*(z^-1))
alpha = (cos((omegaU+omegaL)/2)/cos((omegaU-omegaL)/2));
k = tan((omegaU - omegaL)/2)*tan(omegaP/2);
NUM =((z^2)-((2*alpha/(1+k))*z)+((1-k)/(1+k)));
DEN = (1-((2*alpha/(1+k))*z)+(((1-k)/(1+k))*(z^2)));
HZ_BSF=horner(H_LPF,NUM/DEN)
HW  =frmag(HZ_BSF(2),HZ_BSF(3),512);
W = 0:%pi/511:%pi;
plot(W/%pi,HW)
a=gca();
a.thickness = 3;
a.foreground = 1;
a.font_style = 9;
xgrid(1)
xtitle('Magnitude Response of BSF Filter','Digital Frequency--->','Magnitude');
Disp("HZ_BSF",HZ_BSF);
```

Output:

```
 HZ_BPF  =
                              2
   0.7534875 - 9.702D-17z + 0.7534875z
   ----------------------------------
                          2
      0.5100505 - 9.722D-17z + z
```



Result:

The Butterworth band-stop filter was successfully designed to attenuate frequencies between 0.4π and 0.6π. In the time domain, the filtered signal exhibited a clear reduction of frequencies within the specified range

EX NO: 20	Generations of Standard Signals using DSP Processor
DATE:	

Aim: Verify the generations of signals using DSP Processor

.

Euipments:

Operating System – Windows XP

DSP processor: TMS320C54x or similar.

Assembly language: TMS320C54x Assembly Language.

Kit: DSK- DSK5716 kit, USB probe, 5V DC supply

Program:

Sine Wave Generation:

assembly

```
; Sine wave generation
;
; Registers used:
;   AR0: angle register
;   AR1: sine value register
;
      MOV     AR0, #0         ; initialize angle to 0
LOOP_SINE:
      MOV     AR1, #0         ; initialize sine value to 0
      LDP     AR1, SINETAB(AR0) ; load sine value from lookup table
```

```
    ST      AR1, *DST       ; store sine value in destination
    ADD     AR0, #10        ; increment angle by 10 degrees
    CMP     AR0, #360       ; check if angle exceeds 360 degrees
    JL      LOOP_SINE       ; loop if angle is less than 360 degrees
```

Square Wave Generation:

assembly

```
; Square wave generation
;
; Registers used:
;   AR0: counter register
;   AR1: output register
;
    MOV     AR0, #0         ; initialize counter to 0
LOOP_SQUARE:
    MOV     AR1, #0xFF      ; set output high (square wave)
    ST      AR1, *DST       ; store output in destination
    ADD     AR0, #10        ; increment counter by 10
    CMP     AR0, #100       ; check if counter exceeds 100
    JL      LOOP_SQUARE     ; loop if counter is less than 100
    MOV     AR1, #0x00      ; set output low (square wave)
    ST      AR1, *DST       ; store output in destination
    SUB     AR0, #100       ; decrement counter by 100
    JL      LOOP_SQUARE     ; loop if counter is less than 0
```

Triangular Wave Generation:

assembly

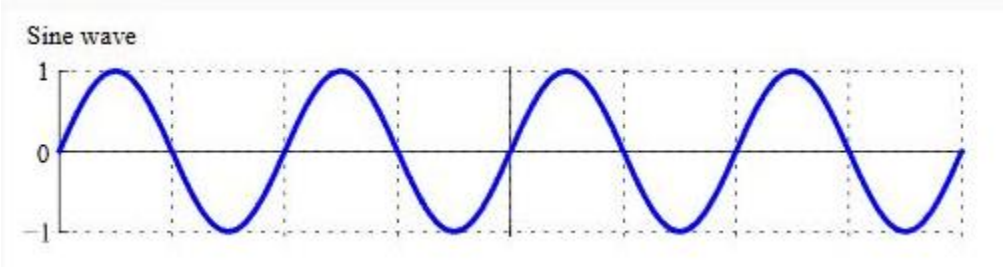
```
; Triangular wave generation
;
; Registers used:
;   AR0: counter register
;   AR1: output register
;
      MOV     AR0, #0         ; initialize counter to 0
LOOP_TRIANGLE:
      MOV     AR1, AR0        ; output increases linearly with counter
      ST      AR1, *DST       ; store output in destination
      ADD     AR0, #10        ; increment counter by 10
      CMP     AR0, #100       ; check if counter exceeds 100
      JL      LOOP_TRIANGLE   ; loop if counter is less than 100
      SUB     AR0, #100       ; decrement counter by 100
      NEG     AR1             ; output decreases linearly with counter
      ST      AR1, *DST       ; store output in destination
      JL      LOOP_TRIANGLE   ; loop if counter is less than 0
```

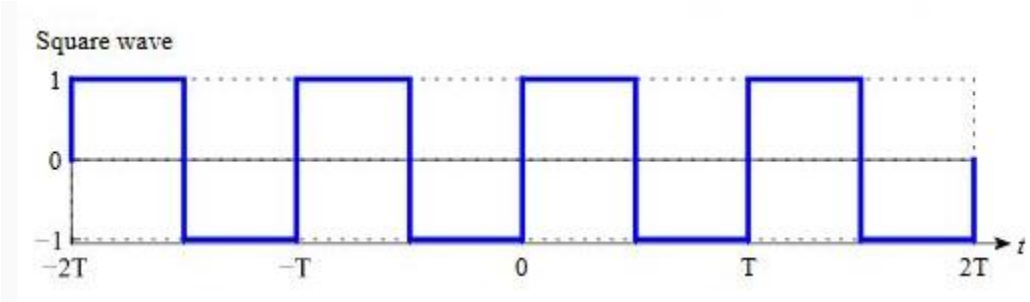
Common Code:

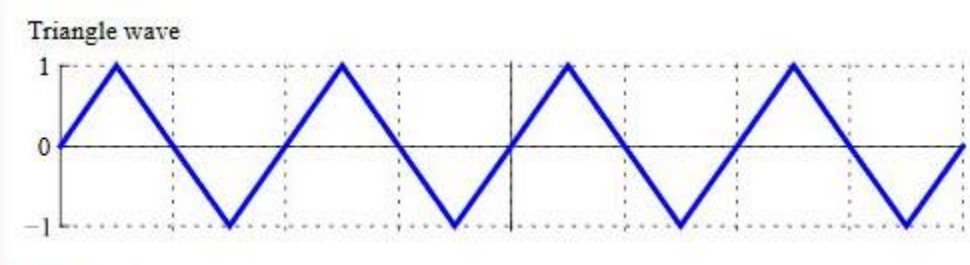
assembly

```
; Common code for all waveforms
;
; Registers used:
;   DST: destination address
;
      MOV     DST, #0x0000    ; initialize
```

Output:







Result:

Using assembly programming on a DSP processor, various signals (sinusoidal, square, and triangular) were generated effectively. The results were verified by observing the outputs on an oscilloscope, demonstrating the processor's capability for real-time signal generation in digital signal processing applications.