Ninevah University
College of Electronics Engineering
Communications Department
Microprocessors and DSP Lab
Third Class
Experiment No. (1)

Introduction to Discrete-time Signals

Objectives

- To study basic discrete-time signals and perform various operations.
- To learn how to represent and implement discrete-time signals using MATLAB.

Introduction

A discrete signal will be denoted by $x(n)$, in which the variable n is an integer value that represents discrete instances in time. Therefore $x(n)$ is also called a discrete-time signal, which is a number sequence and can be written as follows:

$x(n) = \{x(n)\} = \{\ldots, x(-1), \underset{\uparrow}{x(0)}, x(1), \ldots\}$,

where the up-arrow indicates the sample at $n = 0$.

Infinite-duration sequences cannot be represented in MATLAB due to the memory limitation. Contrarily, finite-duration sequences can be represented in Matlab by a row vector of values. However, such a vector does not have any information about sample position (n). A correct representation of $x(n)$ requires two vectors, one for x and one for n. For example:

$x(n) = \{2, 1, -1, \underset{\uparrow}{3}, 1, 4, 3, 7\}$ can be represented in MATLAB by:

```
n = [ -3, -2, -1, 0, 1, 2, 3, 4]
x = [2, 1, -1, 3, 1, 4, 3, 7]
```

Basic Sequences

The basic sequences used in digital signal processing for analysis purposes include Unit sample sequence $\delta(n)$ (also called Impulse sequence), Unit step sequence $u(n)$, complex-valued exponential sequence, sinusoidal sequence and random signals.

In MATLAB, two functions are available to generat random sequences: the funcion **rand**(1, N) which generates a length N random sequences whose elements are uniformly distributed between 0 and 1, and the function **randn**(1, N) which generates a length N Gaussian random sequence with a mean of 0 and variance of 1.

Basic Operations

(1) ***Signal Addition:*** This is a sample-by-sample addition given by:

$$x_3(n)=x_1(n)+x_2(n)$$

In signal addition:

- The sample value of x_1 at $n=-1$ will be added to the sample value of x_2 at $n=-1$.
- The sample value of x_1 at $n=0$ will be added to the sample value of x_2 at $n=0$.
- The sample value of x_1 at $n=1$ will be added to the sample value of x_2 at $n=1$,

and so on (for all values of n).

(2) ***Signal Multiplication:*** This is a sample-by-sample multiplication (or "dot" multiplication) given by:

$$x_3(n)=x_1(n) \bullet x_2(n)$$

In signal Multiplication:

- The sample value of x_1 at $n=-1$ will be multiplied by the sample value of x_2 at $n=-1$.
- The sample value of x_1 at $n=0$ will be multiplied to the sample value of x_2 at $n=0$.
- The sample value of x_1 at $n=1$ will be multiplied to the sample value of x_2 at $n=1$,

and so on (for all values of n).

(3) ***Scaling:*** in this operation, each sample in the signal is multiplied by a scalar α:

$$x_2(n)= \alpha \, . \, x_1(n)$$

(4) ***Shifting:*** in this operation, each sample in the signal is shifted by an amount (k) to obtain a shifted sequence:

$$x_2(n)= x_1(n-k)$$

(5) ***Folding:*** in this operation, each sample in the signal is flipped around $n=0$ to obtain a folded sequence:

$$x_2(n)= x_1(-n)$$

(6) ***Sample summation:*** this operation adds all the sample values of $x(n)$ between n_{min} and n_{max} , and the result is a single value:

$$s = \sum_{n=n_{min}}^{n_{max}} x(n) = x(n_{min}) + x(n_{min}+1) + \cdots + x(n_{max}), \qquad (n_{min} \le n \le n_{max})$$

(7) ***Sample product:*** this operation multiplies all the sample values of $x(n)$ between n_{min} and n_{max} , and the result is a single value:

$$p = \prod_{n=n_{min}}^{n_{max}} x(n) = x(n_{min}) * x(n_{min}+1) * \ldots * x(n_{max}), \qquad (n_{min} \le n \le n_{max})$$

Matlab Functions

In Matlab, it is possible to create and use a function by following these steps:

1- Open a new m-file.
2- Define the inputs, outputs and name of the function as follows:
function [output1, output2, …] = func_name(input1, input2, …)
3- Press *enter* and start writing the operations performed by this function.
4- Save the m-file with the **SAME** name given to the function in step 2 (i.e. func_name.m).
5- In the main program, call the function as follows:
[result1, result2, …] = func_name (input1, input2,…);

NOTE 1: Both the function and main program **MUST** be saved in the same directory or folder; otherwise, calling the function will result in an ERROR.
NOTE 2: Writing two or more functions in the same m-file is **NOT** allowed.

Experiment

1. Create the following function that generates an impulse sequence $\delta(n)$, where:

$$\delta(n)=\begin{cases}1, & n=0\\ 0, & n\neq 0\end{cases}=\{\ldots,\ 0,\ 0,\ \underset{\uparrow}{1},\ 0,\ 0,\ \ldots\}$$

```
function [x, n] = impseq(n0, n1, n2)  %Generate x(n)=delta(n-n0)
                                      % n1 <= n <= n2
  n = n1:n2;
  x = [(n-n0) == 0];
```

Now use this function to create and plot an impulse $\delta(n)$ with n_1=-5 and n_2=5.

2. Create the following function that generates a unit step sequence, where:

$$u(n)=\begin{cases}1, & n\geq 0\\ 0, & n<0\end{cases}=\{\ldots,\ 0,\ 0,\ \underset{\uparrow}{1},\ 1,\ 1,\ \ldots\}$$

```
function [x, n] = stepseq(n0, n1, n2) % Generate x(n)=u(n-n0)
                                      % n1 <= n <= n2
  n = n1:n2;
  x = [(n-n0) >= 0];
```

Now use this function to create and plot a unit step $u(n)$ with n_1 = -5 and n_2 = 5.

3. Create the following function that adds two signals:

```
function[y,n]=sigadd(x1, n1, x2, n2) %implements y(n)= x1(n)+x2(n)
%  y = resulting sequence
%  n = Index vector of y
% x1 = first sequence over n1
% x2 = second sequence over n2 (n2 can be different from n1)

n = min(min(n1), min(n2)):max(max(n1), max(n2)); %duration of y(n)
y1 = zeros(1, length(n));
y2 = y1;                                  % initialization
y1((n>=min(n1))&(n<=max(n1)))=x1;         % x1 with a new duration
y2((n>=min(n2))&(n<=max(n2)))=x2;         % x2 with a new duration
y = y1+y2;
```

Now create a unit step *u(n)* with n_1=-5 and n_2=5 and *u(n-2)* with n_1=-5 and n_2=10, then plot the result of adding these two step signals.

4. Modify the function in step (3) so that it performs multiplication instead of addition. Use "sigmult" as a name, then use this function to plot the result of *u(n)*u(n-2)*.

5. Create the following function that shifts an input signal:

```
function [y,n] = sigshift(x, m, n0)   % implements y(n) = x(n-n0)
%   m is an index vector of x
%   n0 is the shifting value
n = m+n0;
y = x;
```

Now create an impulse signal $\delta(n)$ and apply a shifting by 3 to this impulse.

6. Create the following function that folds an input signal:

```
function [y,n] = sigfold(x, n)        % implements y(n) = x(-n)
y = fliplr(x);
n = -fliplr(n);
```

7. Write a MATLAB program to generate and plot each of the following sequences:

(a) $x(n) = 2\delta(n+2) - \delta(n-4)$, $-5 \le n \le 5$.

(b) $x(n) = n[u(n)-u(n-10)]+(20-n)[u(n-10)-u(n-20)]$ $-20 \le n \le 20$

8. Let x(n) = {1, 2, 3, 4, 5, 6, 7, 6, 5, 4, 3, 2, 1},
↑

Write a MATLAB program to generate and plot the following sequences:

(a) x1(n) = 2x(n-5) – 3x(n-4)
(b) x2(n) = x(3-n) + x(n)x(n-2)

9. Let $x(n)$ = {1, 2, 3, 4, 3, 2, 1},
↑

Design an experiment to show that signal folding and time shifting are not commutative, e.g., $TS_k(SF(x))$ is not equal to $SF(TS_k(x))$, where TS_k is time shifting by k samples (Assume $k = 3$), and SF is signal folding.

10.
A- What is the Matlab function that can be directly used for sample summation?
B- What is the Matlab function that can be directly used for sample product?
C- Use the functions in (A) and (B) to determine the sample summation and multiplication of the following sequence:
x(n) = { 1, -3, 7, 5, 2, 3, 9, -8, 4, 1, 3 }.
↑

11. To represent a signal in MATLAB, why do we need 2 sequences instead of only one sequence?

Ninevah University
College of Electronics Engineering
Communications Department
Microprocessors and DSP Lab
Third Class
Experiment sheet No. (2)

Introduction to 8086 Microprocessor Emulator

Objectives

To introduce the basic operations and functions provided by the Emulator program.

Introduction

8086 Microprocessor Emulator, also known as EMU8086, is an emulator of the 8086 Microprocessor. It is developed with a built-in 8086 assembler. This tool is primarily designed to copy or emulate hardware. These include the memory of a program, CPU, RAM, input and output devices and even the display screen.

Emulator Window

The Emulator window displays all the 8086MP registers, flags and memory (addresses and contents). This window is shown in *figure 1*. It provides a user interface that is simple and easy to manage. To open this window from the toolbar, click (emulate) and select (show emulator). This window will also be opened whenever you select to emulate a code after compilation.

- You can modify the value of any register directly from the **Emulator window** by double clicking the register, where an **Extended Value Viewer** window will be opened, with value of that register converted to all possible forms. Also you can modify the values of registers on runtime by typing over the existing values.

- Double clicking on any memory location opens **Extended Value Viewer** with WORD value loaded from memory at the selected location. LOW BYTE is loaded from the selected position and HIGH BYTE from the next memory address. You can modify the value of any memory word directly in the (Extended Value Viewer) window. Also, dealing with only one byte of the memory is also allowed.

- **Flags** button allows you to view and modify flags on runtime.

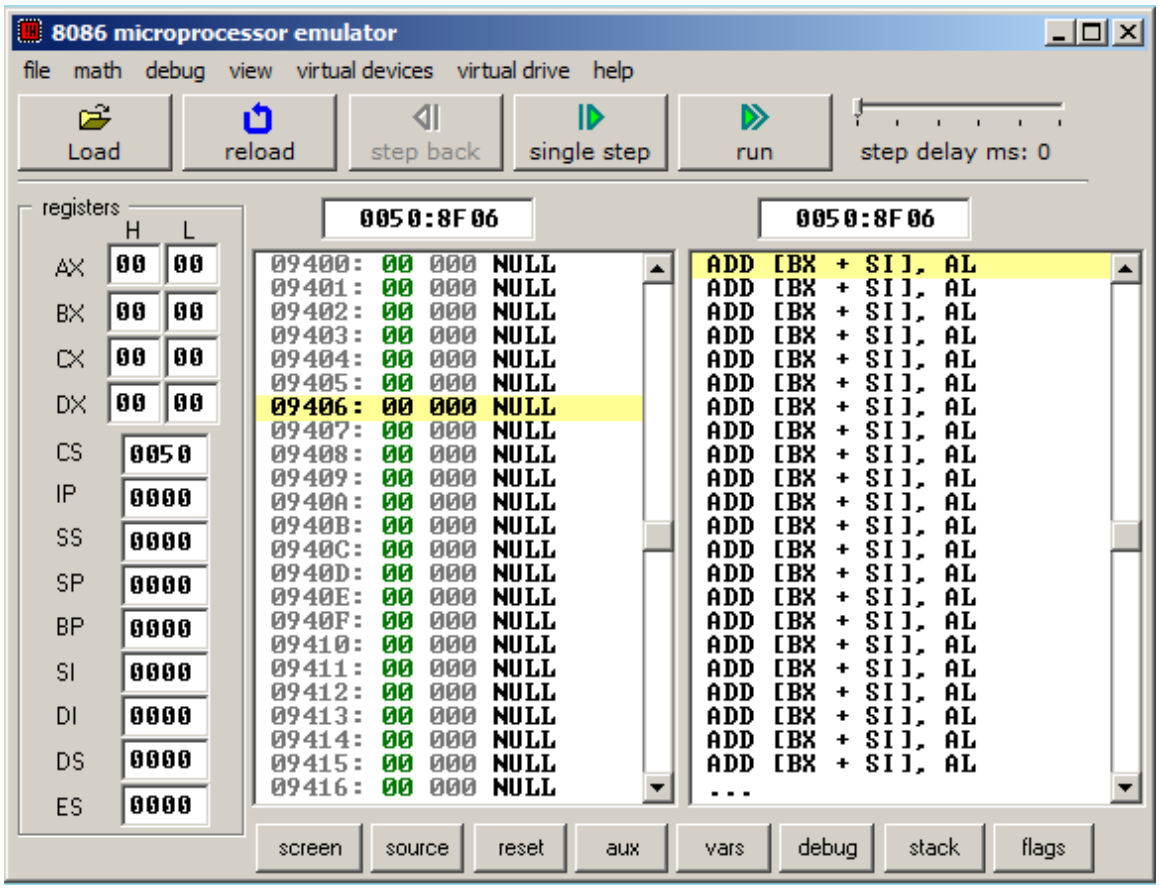


Figure 1: Emulator window

Compiling Assembly Language Code

1. When opening EMU8086 program, select COM file to write your Assembly language code, where the window shown in *figure 2* will be opened. A COM file is a simple tiny executable file, in which assembly language programs can be written.

2. Type your code inside the text area, and click **compile** button.
   a. If the code contains any errors, a new window will be opened, showing the first error, with a description of the error reason and location. An example error is shown in *figure 3*.
   b. If the code contains no error, you will be asked to save the compiled file. The compiler converts the program source to a set of bytes, which is called ***Machine Code***. Processor understands the machine code and executes it.

3. After successful compilation, click **emulate** button to load the compiled file in emulator. Two windows will be opened; the first window shows the original source code, while the second window is the **Emulator window**.

4. From the Emulator window, click **Run** button to execute the complete code, or **Single Step** to execute the code step by step.

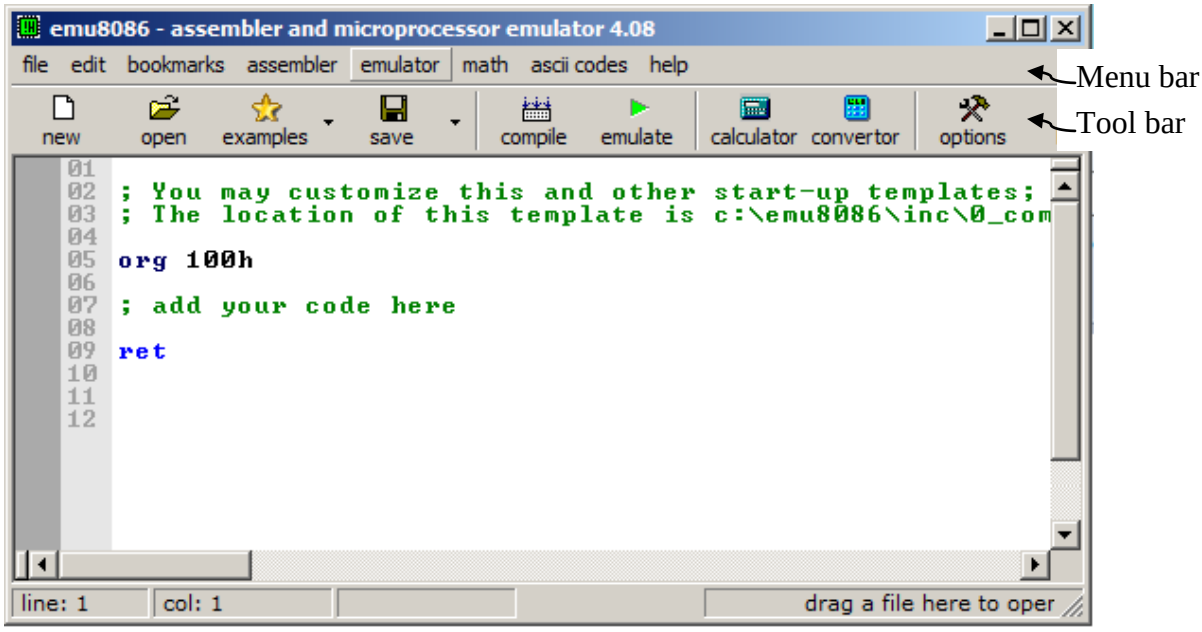


Figure 2: An opened COM file

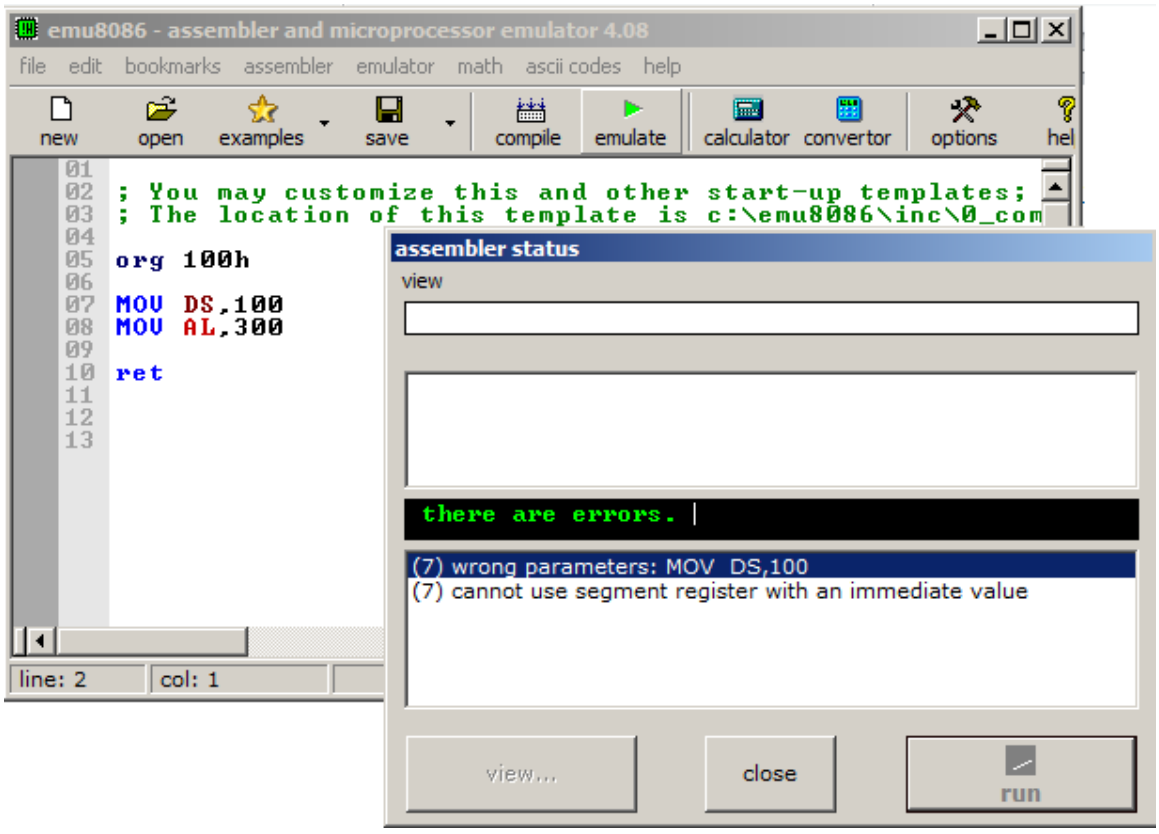


Figure 3: Compiling an example code containing errors

Other Features

Students can also benefit from other features provided by the Emulator program. Those include:

- A table from the Menu bar that contains the **ASCII codes** and their equivalent decimal and hexadecimal values.

- A **calculator** from the Tool bar that evaluates multiple mathematical and logical expressions in decimal, hexadecimal, octal and binary numbering systems.

- A **converter** from the Tool bar that converts numbers from a numbering system to another.

- A complete guide to the Assembly language instructions set with examples, provided in the Tool bar from (Help) icon. This guide also contains tutorials on numbering systems, memory access, variables, interrupts and other topics.

- Code examples on several applications, available from (Examples) in the tool bar.

- **Virtual I/O devices** at specified port numbers, available in (Virtual devices) menu from the menu bar in the Emulator window. Examples on using specific devices like the stepper motor, traffic lights and robot controller are included in the tutorials.

Experiment

1- Display the registers of the 8086 Mp from the Emulator window.

   a. What is the value of the code segment (CS) register?

   b. What is the value of the instruction pointer (IP) register?

   c. To what physical memory address is the instruction pointer referring?

2- Change the value of the data segment register (DS) to (A00H) and the value of the extra segment register (ES) to (1A00H).

a. What is the range of physical addresses for each of those segments?

b. Are the data segment and extra segment separated, overlapped, or adjacent?

3- Change the value of BL to (FAH). Find the equivalent value of BL in:

i. binary:

ii. decimal signed:

iii. decimal unsigned:

(Verify your answers from the *Extended Value Viewer*).

4- Display the 8086 Mp registers:

a. What are the values of the **segment registers**?

b. Make **segmentation** by giving different values to the segment registers, then give the physical address of the first location of each segment:

5- Display the 8086 Mp memory:

A- Go to address (11A H) and find the contents of this memory location as:

a. Byte:

b. Word:

B- Give two logical addresses that allow you to reach this physical address:

C- By double clicking this memory location, change its contents to (1234H).

a. How many bits does each memory location contain?

b. Specify the **address** where each byte of (1234H) is stored:

c. What is the equivalent value of (1234H) in:

i. binary:

ii. decimal signed:

iii. decimal unsigned:

(Verify your answers from the *Extended Value Viewer*).

6- If (ES=0028H), (DS=F100H), (SS=0800H), what is the byte content of the memory at offset (0007H) in the **extra segment**? What is its ASCII equivalent?

7- Enter the ASCII codes of the characters string (*Hello*) into memory, starting from address (7A00:0170H).

Examine Your Knowledge:

1- What is the physical address of the last memory location in the 8086 Mp (1MByte) memory? What logical address do you need to use to reach this location?

2- If: (DS=0200H), (ES=1300H), (SS=2000H). What is the range of addresses for each segment? Are they separated, overlapped, or adjacent?

3- If: (DS=0372H), (ES=752dH), (SS=3456H), (SP=06d3H), (DI=9af3H), what is the physical address of the location that (SP) is pointing to?

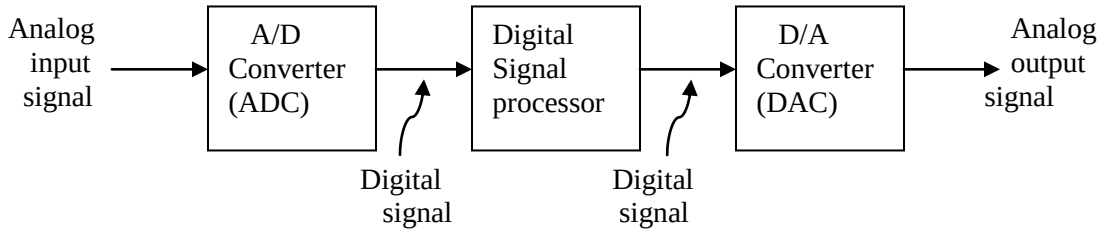
Ninevah University
College of Electronics Engineering
Communications Department
Microprocessors and DSP Lab
Third Class
Experiment No. (3)

Sampling and Reconstruction of Analog Signals

Objectives

-To study the sampling principle and its effect on the frequency-domain quantities.
-To study several reconstruction approaches.

Introduction



In many DSP applications, real world analog signals are converted into discrete signals using sampling and quantization operations (collectively called analog-to-digital conversion or ADC). These discrete signals are processed by digital signal processors, and the processed signals are converted into analog signals using a reconstruction operation (called digital-to-analog conversion or DAC).

To understand how the DSP system works, we need to know the relation between an analog signal and its discrete time sampled version. In time domain, relation between an analog signal and a sampled discrete time signal is given by:

$$x(n) = x_a(nT_s), \qquad \ldots (1)$$

where T_s is the sampling interval.

However, in the frequency domain, the relation between spectra of an analog signal and its discretized version is more complicated. Here, Fourier analysis can be used to explain this relation and then address the reconstruction operation as follows:
The continuous time Fourier transform is given by:

$$X_a(F) = \int_{-\infty}^{\infty} x_a(t) e^{-j2\pi Ft} dt \ . \qquad \ldots (2)$$

The inverse continuous time Fourier transform is given by:

$$x_a(t) = \int_{-\infty}^{\infty} X_a(F) e^{j2\pi Ft} dF \ . \qquad \ldots (3)$$

The discrete time Fourier transform is given by:

$$X(f) = \sum_{n=-\infty}^{\infty} x(n) e^{-j2\pi fn} \ . \qquad \ldots (4)$$

The inverse discrete time Fourier transform is given by:

$$x(t) = \frac{1}{F_s}\int_0^{F_s} X(f)e^{j2\pi fnT_s}df \ . \qquad \ldots (5)$$

The relation between $X_a(F)$ and $X(f)$ is given by:

$$X(f) = F_s \sum_{k=-\infty}^{\infty} X_a((f-k)F_s), \qquad \ldots (6)$$

where F_s is the sampling frequency = $1/T_s$. In other words, $X(f)$ consists of infinite numbers of copies of scaled $X_a(F)$ separated by frequency interval $f = 1$. From the relation between discrete time and analog frequencies:

$$f = \frac{F}{F_s}$$

we get:

$$X(\frac{F}{F_s}) = F_s \sum_{k=-\infty}^{\infty} X_a(F - kF_s), \qquad \ldots (7)$$

in which $X(\frac{F}{F_s})$ consists of infinite numbers of copies of scaled $X_a(F)$ separated by interval $F = F_s$.

Sampling Principle

In order to avoid aliasing, a band-limited signal $x_a(t)$ with a maximum frequency (f_m) can be reconstructed from its sample values $x(n) = x_a(nT_s)$ if the sampling frequency $F_s = 1/T_s$ is greater than twice the maximum frequency of $x_a(t)$:

$$F_s > 2f_m \ , \qquad \ldots (8)$$

otherwise aliasing would result in $x(n)$. The sampling rate of $(2f_m)$ for an analog band-limited signal is called the (Nyquist Rate).

Reconstruction

From the sampling theorem and the above examples it is clear that if we sample band-limited $x_a(t)$ above its Nyquist rate, then we can reconstruct $x_a(t)$ from its samples $x(n)$. Using an interpolation formula:

$$x_a(t) = \sum_{n=-\infty}^{\infty} x(n)\text{sinc}(F_s(t - nT_s)), \qquad \ldots (9)$$

where $\text{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$ is an interpolating function derived from an ideal low pass reconstruction filter. However, since an ideal low pass reconstruction filter cannot be implemented, we usually estimated the ideal low pass filter by the following methods:

- **Zero-order-hold (ZOH) interpolation:** In this interpolation, a given sample value is held for the sample interval until the next sample is received:

  $$x_a(t) = x(n), \qquad nT_s \le t < (n+1)T_s \qquad \ldots (10)$$

which can be obtained by filtering the impulse train through an interpolating filter of the form:

$$h_0(t) = \begin{cases} 1 & 0 \le t < T_s \\ 0 & \text{otherwise} \end{cases} \qquad \ldots (11)$$

which is a rectangular pulse. The resulting signal is a piecewise-constant (staircase) waveform which requires an appropriately designed analog post-filter for accurate waveform reconstruction.

x(n) → [ZOH] → $x_a(t)$ → [Post-Filter] → $x_a(t)$

- **First-order-hold (FOH) interpolation:** In this case the adjacent samples are joined by straight lines. This can be obtained by filtering the impulse train through:

$$h_1(t) = \begin{cases} 1 + \dfrac{t}{T_s} & 0 \le t < T_s \\ 1 - \dfrac{t}{T_s} & T_s \le t < 2T_s \\ 0 & \text{otherwise} \end{cases} \qquad \ldots (12)$$

Experiment

To study the effect of sampling on the spectrum of discrete signals, we will sample at two different sampling frequencies and then reconstruct the signals as follows:

Let $x_a(t) = e^{-10|t|}$. The continuous time Fourier transform of this signal is given by:

$$X_a(F) = \left(\frac{2*10}{10^2 + (2\pi F)^2} \right).$$

1. Sampling $x_a(t)$ at F_s = 50 samples/sec:

```
clc
clear all
close all
tmin = -1;
tmax = 1;
% Analog signal
t = tmin:0.001:tmax;
xa = exp(-10*abs(t));
% Sampling rate (sample/second):
Fs = 50;
% Sample period
Ts = 1/Fs
% Discrete time signal
n = tmin/Ts:tmax/Ts;
x = exp(-10*abs(n*Ts));
%Display signals in time domain
figure;
subplot(211)
```

```
plot(t,xa)
title('Analog and discrete time signals')
xlabel('time (sec)')
ylabel('Analog signal x(t)')
subplot(212)
stem(n,x)
xlabel('n')
ylabel('Discrete time signal x(n)')

% Computing Fourier transform
% Analog frequency (Hertz)
F = -100:0.1:100;
W = 2*pi*F;
%Discrete time frequency
f = F/Fs;
w = 2*pi*f;
%Analog spectrum for continuous time Fourier transform
XaF = 2.*(10./(10^2+W.^2));
%Discrete time Fourier transform (implementation of eq.4):
XF = x*exp(-j*n'*w);
%Display spectra in frequency domain
figure;
subplot(311)
plot(F,abs(XaF))
title('Spectra of signals')
xlabel('Frequency [Hz]')
ylabel('Original Xa(F)')
subplot(312)
plot(F,abs(XF))
xlabel('Frequency [Hz]')
ylabel('X(F)')
subplot(313)
plot(f,abs(XF))
xlabel('F/Fs')
ylabel('X(F/Fs)')
```

Explain whether experimental results are consistent with theoretical results.

2. Reconstruction of $x_a(t)$:

```
figure;
clf
subplot(211)
hold on
stem(n*Ts,x,'r')
for i = 1:length(n)
    xsinc(i,:) = x(i)*sinc(Fs*(t -(i+min(n)-1)*Ts));
    plot(t,xsinc(i,:))
end
title('Signal Reconstruction')
xlabel('time [sec]')
```

```
ylabel('x(n)*Sinc(Fs*(t-nTs))')
hold off
xar = sum(xsinc);
subplot(212)
plot(t,xar,'b-',t,xa,'r:')
legend('Reconstructed signal','Original signal')
ylabel('Reconstructed signal xa(t)')
xlabel('time [sec]')
% reconstruction error:
maxerror = max(abs(xa - xar));
```

Explain the property of *sinc* function that enables the reconstruction of analog signals without overlapping.

3. Repeat steps 1 and 2 using *Fs* = 10 samples/sec. Explain why using *Fs* = 50 samples/sec is better than using *Fs* = 10 samples/sec. Locate the area where the spectra are most likely to overlap with each other, resulting in aliasing.

4. Consider an analog signal $x_a(t) = sin(20\pi t)$, $0 \le t \le 1$. It is sampled at T_s = 0.01, 0.03, 0.05, 0.07 and 0.1 sec intervals to obtain *x(n)*.

   a. For each T_s plot *x(n)*.
   b. Reconstruct the analog signal $y_a(t)$ from the samples *x(n)* using the sinc interpolation (use Δt = 0.001) and determine the frequency in $y_a(t)$ from your plot.
   c. Comment on your results.

Discussion

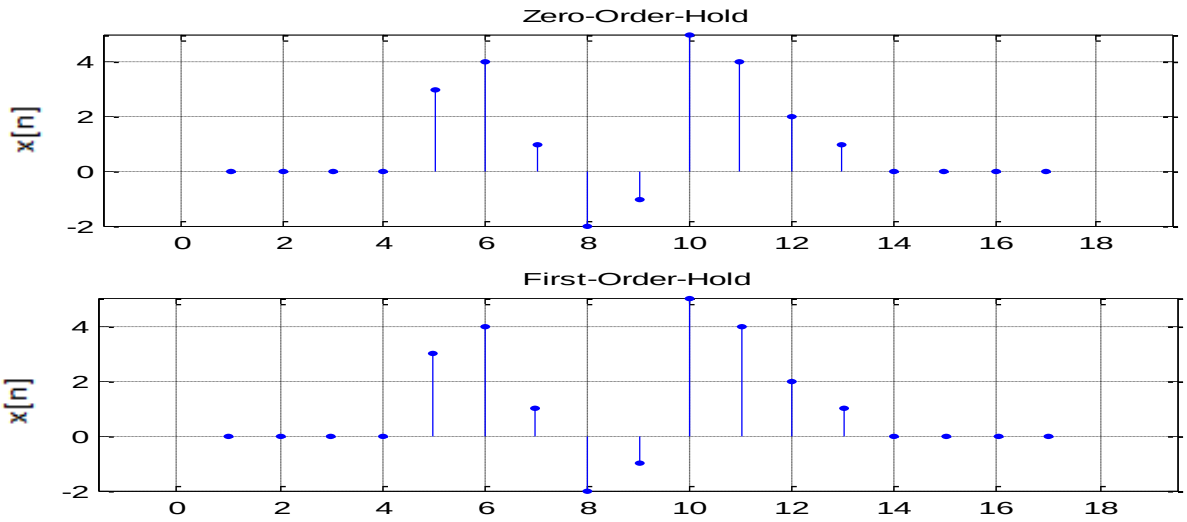
1. What is the Matlab function that can be used to plot staircase (ZOH) interpolation of analog signals?

2. What is the Matlab function that would be used to plot linear (FOH) interpolation of analog signals?

3. From this experiment, describe why the minimum sampling rate must be at least twice the bandwidth of an analog signal.

4. From the figure in Page 1, explain briefly the function of each part of the DSP system.

5. For a discrete time signal in the figure below, draw the results of using zero-order-hold and first-order-hold interpolation.


Zero-Order-Hold
x[n]
4
2
0
-2
0
2
4
6
8
10
12
14
16
18
First-Order-Hold
x[n]
4
2
0
-2
0
2
4
6
8
10
12
14
16
18

Ninevah University
College of Electronics Engineering
Communications Department
Microprocessors and DSP Lab
Third Class
Experiment sheet No. (4)

Introduction to Assembly Language

Objectives

- To use Assembly language instructions to apply changes and perform simple tasks on the 8086Mp registers and memory.
- To learn the meaning and purpose of machine language.

Assembly Language

Assembly language is a low-level language which is developed to enable the programmer to perform various mathematical, logical and/or hardware-manipulation operations, using the hardware provided inside the microprocessor. In order to use assembly language, the programmer needs to have sufficient knowledge on the internal architecture and instruction set of the microprocessor.

Machine Language

The microprocessor can only understand the binary language (0's and 1's). Therefore, the assembler converts assembly language to a binary language called (Machine Language). Machine code is stored in a part of the memory called (code segment) and it can be executed by the microprocessor to perform the required task(s).

8086Mp instructions vary in the number of bytes used to encode them. Some instructions can be encoded with just 1 byte; others can be done in 2 bytes. The maximum number of bytes of an instruction is 6. In order to know the machine code of specific instructions, write these instructions in a COM file and click on (emulate) from the tool bar. The emulator window will display memory starting from the location where the first instruction is stored. The machine code of any instruction is stored starting ***from the higher byte to the lower***. Figure 1 shows the emulator window displaying the machine code of an example instruction.

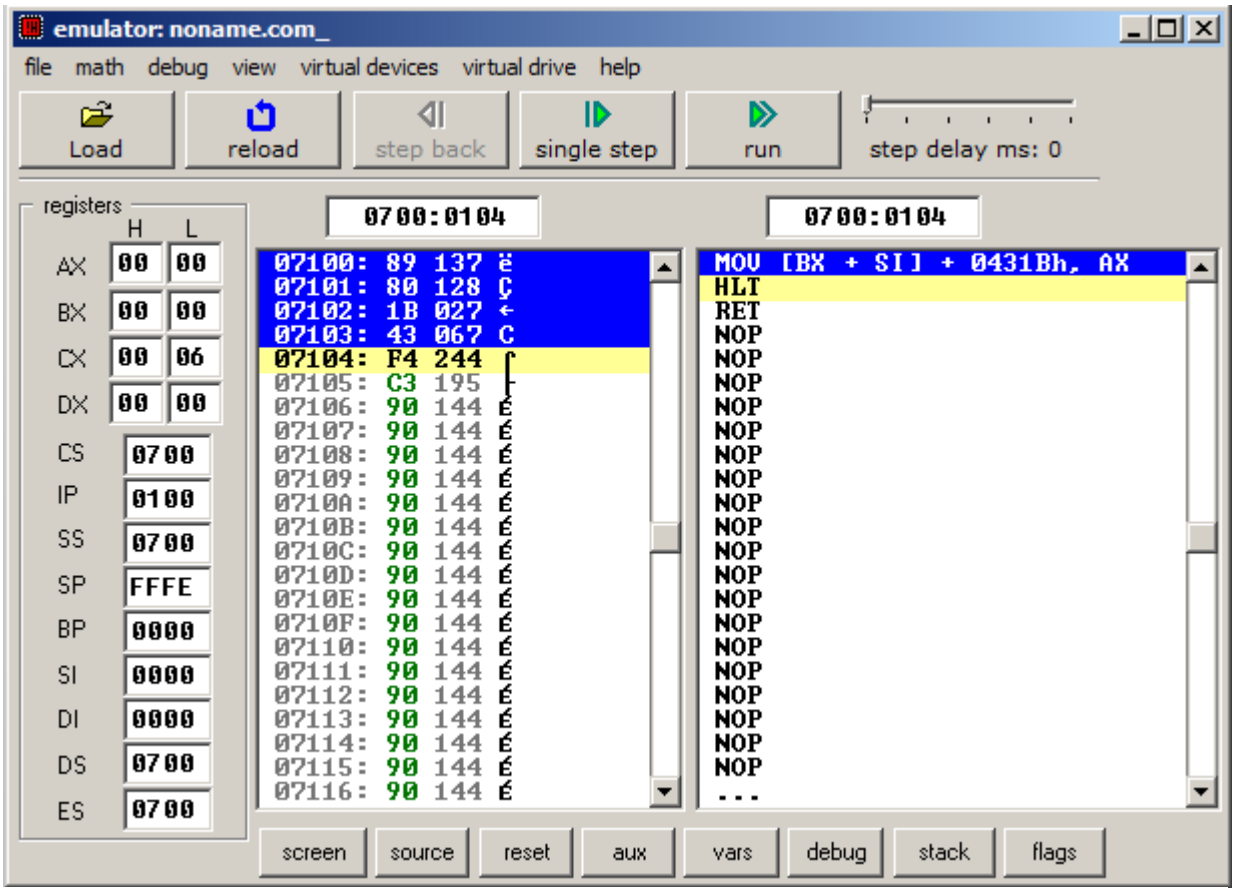


Figure 1: The emulator window displaying machine code: (89801B43 H) of the instruction: (MOV [BX+SI+431BH], AX)

Procedures

1- Open a new COM file and assemble the instruction: MOV CL, [BX+DI+2], then set the value of BX to 35H, DI to 212H, CX to 1234H and DS to 1000H.

   a. What is the physical address of the **data** used in this instruction?

   b. What is the logical address of the memory where this instruction is stored?

   c. What is the length of this instruction?

   d. What is the machine code of this instruction?

   e. Put the value 55H in the memory address specified by the instruction and execute it. What is the value of CL after execution and why?

f. Check the value of IP before and after executing this instruction. What is the difference between the two values and why?

2- Open a new COM file and assemble the instruction: `ADD AL,4BH` , then change AL to 6EH.

a. What is the length of this instruction?

b. What is the machine code of this instruction?

c. Calculate the result of this instruction:

i. Manually:

ii. by executing the instruction and checking the value of AL:

Are the two results equal?

d. Check the status of the Overflow flag (OF). What does this status indicate?

3- Convert the following machine codes into Assembly language instructions:

89daH

00c8H

51H

04d4H

fec9H

fec0H

80eb05H

4- Open a new COM file and assemble the following code:

```
      MOV   CL, 19H
      MOV   BH, 3
RPT:  ADD   CL, 5
      DEC   BH
      JNZ   RPT
      HLT
```

Run the program one instruction at a time (single step) and fill one row in the following table with each step, until the end of the code.

	IP	Executed Instruction	BH	CL	Machine Code
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					

5- Describe the work of the following program:

```
        MOV   CX , 0AH
        MOV   SI , 300H
        MOV   BX , 520H
RPT:    MOV   AL , [ SI ]
        MOV   [ BX ] , AL
        INC   SI
        INC   BX
        DEC   CX
        JNZ   RPT
        HLT
```

Ninevah University
College of Electronics Engineering
Communications Department
Microprocessors and DSP Lab
Third Class
Experiment sheet No. (5)

8086 Microprocessor Instruction Set

Objectives

- To learn the interpretation of various 8086Mp instructions.
- To gain facility in using 8086Mp registers and memory.

Procedure

1- The value of the flag bits are usually affected by the result of the last executed instruction. What are the initial values of the flag register bits (before executing any instruction)?

2- Open a new COM file and assemble the instruction: **MOV [DI], CX** ; then set the value of DI to C4DH, CX to A254H, and set the value of the parity flag to (Odd). Run the instruction and answer the following:

a. What is the addressing mode of the instruction?

b. After execution, what is the new value of:

1. CX
2. DI
3. PF

3- Open a new COM file and assemble the instruction: **XCHG AX, CX** ; then set AX to ABCDH and CX to 1234H. Run the instruction and give the new value of:

a. AX

b. CX

4- Open a new COM file and assemble the instruction: **Neg BX** ; then set BX to 45H and set the sign flag to positive. Run the instruction and give the new value of:

a. BX

b. SF

5- A way to allow the programmer to change the default segment into another segment is called (Segment Override Prefix). It is an additional byte that is added to the front of an instruction to select an alternate segment register. It may be added to almost any instruction in any memory addressing mode.

Examples

```
MOV  AX, DS:[BP]      ; changing from stack segment to data segment.
MOV  AX, ES:[BP]      ; changing from stack segment to extra segment.
MOV  SS:[DI], AX      ; changing from data segment to stack segment.
```

NOW: Open a new COM file and assemble the instruction: **MOV AL, ES:[04B6H]** ; then Make *segmentation*, enter the string: "Good Morning" into memory at ES:4B6H and set AX to 0H. Run the instruction and answer the following:

a. What is the new value of AX?

b. What is the meaning of (ES:) in this instruction?

6- Write ONE instruction that adds the byte stored at memory location SS:333H with the value in BH and saves the result in BH.

7- Open a new COM file and assemble the instructions:

```
PUSH BX
PUSH CX
```

Then set BX to 4567H, CX to FEDCH and SP to A59FH. Run the instructions and answer the following:

a. What is the new value of SP ?

b. Display the memory locations where you pushed the values of BX and CX in the stack.

8- Open a new COM file and assemble the instruction: **ADC AX, BX** ; then set AX to 5H, BX to 8H, and set the carry flag to 1. Run the instruction and give the new value of:

a. AX

b. CF

Discussion

1- Write an assembly language program to fill a block of data consists of 12 words with 22H. This block starts at ES:210H.

2- What are the equivalent instructions of (LOOP N1)?

3-Write an Assembly program to set the Flags register bits so that:

- there is no carry
- Parity is even
- no zero
- Sign is positive
- and there is no overflow

4- Write an Assembly language program to exchange between two blocks, each of them consists of 14 Bytes, the first block starts at SS:27EH and the second starts at ES:64BH.

Ninevah University
College of Electronics Engineering
Communications Department
Microprocessors and DSP Lab
Third Class
Experiment No. (6)

Convolution and Correlation of Discrete-Time signals

Objectives

- To study the convolution and correlation of discrete-time signals.
- To learn how to implement the two operations using MATLAB.

Introduction

Convolution of sequences

As the mathematical operation, a linear convolution sum is defined as:

$$y(n) = x(n) * h(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k)$$

In general, the convolution operation is used to describe the response of an LTI system. In DSP it is an important operation and has many other uses.

If arbitrary sequences are of infinite duration, then MATLAB cannot be used directly to compute the convolution. MATLAB does provide a built-in function called (**conv)** that computes the convolution between two finite-duration sequences. The **conv** function assumes that the two sequences begin at n = 0 and it is invoked by:

```
>> y = conv(x, h);
```

However, the **conv** function neither provides nor accepts any timing information of the convolved sequences. To solve this inconvenience, a simple extension of the **conv** function, called **conv_m**, which performs the convolution of arbitrary support sequences can be designed as:

```
function [y, ny] = conv_m(x, nx, h, nh)
    % Modified convolution routine for signal processing
    % ---------------------------------------------------
    % [y, ny] = convolution result
    % [x, nx] = first signal
    % [h, nh] = second signal
    %
    nyb = nx(1) + nh(1);
    nye = nx(length(x)) + nh(length(h));
    ny = nyb:nye;
    y = conv(x, h);
    % End of the function
```

Correlation of sequences

Correlation is an operation used in many applications in digital signal processing. It is a measure of the degree to which two sequences are similar. Given two real-valued

sequences *x(n)* and *y(n)* of finite energy, the crosscorrelation of *x(n)* and *y(n)* is a sequence $r_{xy}(l)$ defined as:

$$r_{xy}(l) = \sum_{n=-\infty}^{\infty} x(n)y(n-l)$$

The index l is called the shift or lag parameter. The special case when $y(n) = x(n)$ is called *autocorrelation* and is defined by:

$$r_{xx}(l) = \sum_{n=-\infty}^{\infty} x(n)x(n-l)$$

When comparing the convolution operation with the crosscorrelation of two sequences, a close similarity can be observed. The crosscorrelation $r_{xy}(l)$ can be put in the form:

$$r_{xy}(l) = x(l) * y(-l)\,,$$

while the autocorrelation $r_{xx}(l)$ in the form:

$$r_{xx}(l) = x(l) * x(-l)\,.$$

Therefore, correlation can be computed using the **`conv`** function if sequences are of finite duration.

The signal-processing toolbox in MATLAB provides a function called **`xcorr`** for sequence correlation computations. However, the function does not provide the timing (or lag) information, which then must be obtained by some other means.

Experiment

(Before starting the lab, you're supposed to have all functions from (DSP-Exp.1) stored in your working folder).

A. Convolution of Sequences

Given the following two sequences:

$x_1(n) = [\ 0, 0, \underset{\uparrow}{1}, 1, 1, 1, 1]$ and $x_2(n) = [0, 0, 0, \underset{\uparrow}{1}, 1, 1]$

1. Plot $x_1(k)$ and $x_2(k)$ using the (**`stem`**) function.
2. Write a program to determine: $y(0) = \sum_{k=-10}^{10} x_1(k)x_2(-k)$
3. Write a program to determine: $y(1) = \sum_{k=-10}^{10} x_1(k)x_2(1-k)$

4. Write a program to determine: $y(-1) = \sum_{k=-10}^{10} x_1(k)x_2(-1-k)$

5. Develop a program to determine:

$$y(n) = x_1(n) * x_2(n) = \sum_{k=-\infty}^{\infty} x_1(k)x_2(n-k) \qquad -15 \le n \le 15$$

6. Plot the sequence *y(n)* obtained in (5). What are the beginning point and end point of *y(n)* that are non-zero values?
7. Write the following MATLAB commands and plot the result (*y* versus *ny*):

```
x1 = [0, 0, 1, 1, 1, 1, 1]; nx1 = [-2:4];
x2 = [0, 0, 0, 1, 1, 1];    nx2 = [-3:2];
[y, ny] = conv_m(x1, nx1, x2, nx2);
```

8. Compare the plot in (7) with the plot in (6). Are these plots identical? Explain.

B. Correlation of Sequences

1. Create a MATLAB function called **`xcorr_m`** to perform the correlation operation based on the convolution operation.
2. Let: `x(n)=[3, 11, 7, 0, -1, 4, 2]` be a prototype sequence,
   ↑
   `y(n)=x(n-2)+w(n)` be its noise-corrupted and shifted version. Write and run the following MATLAB code:

```
% noise sequence 1:
x =[3, 11, 7, 0, -1, 4, 2]; nx = [-3:3];     % given signal x(n)
[y, ny] = sigshift(x, nx, 2);            % obtain x(n-2)
w = randn(1, length(y)); nw= ny;             % generate w(n)
[y, ny] = sigadd(y, ny, w, nw);      % obtain y(n) = x(n-2) + w(n)
[rxy, nrxy] = xcorr_m(x, nx, y, ny);         % crosscorrelation
subplot(2, 1, 1); stem(nrxy, rxy);
axis([-5, 10, -50, 250]); xlabel('lag variable l');
ylabel('rxy'); title('Crosscorrelation: noise sequence  1');
% noise sequence 2:
x =[3, 11, 7, 0, -1, 4, 2]; nx = [-3:3];     % given signal x(n)
[y, ny] = sigshift(x, nx, 2);            % obtain x(n-2)
w = randn(1, length(y)); nw = ny;            % generate w(n)
[y, ny] = sigadd(y, ny, w, nw);      % obtain y(n) = x(n-2) + w(n)
[rxy, nrxy] = xcorr_m(x, nx, y, ny);         % crosscorrelation
subplot(2, 1, 2); stem(nrxy, rxy);
axis([-5, 10, -50, 250]); xlabel('lag variable l');
ylabel('rxy'); title('Crosscorrelation: noise sequence 2');
```

3. From the result of the crosscorrelation, what is the value of *l* at which the crosscorrelation got peak? What would it imply?

Ninevah University
College of Electronics Engineering
Communications Department
Microprocessors and DSP Lab
Third Class
Experiment No. (7)

Synthesis of Simple Audio Signals

Objectives

- To introduce the digital synthesis of simple audio signals using Matlab.
- To introduce and implement some audio effects.

Introduction

A. Dual Tone Multi-Frequency (DTMF)

Dual Tone Multi-Frequency is a format for transmitting data by telephone and radio. The tones you hear when you press the keys on your phone are the DTMF tones. It was first designed by engineers at Bell Labs for sending data across long distances over a variety of systems.

DTMF signal consists of the sum of two pure sinusoids at valid frequencies. Table 1 shows the format of DTMF. Each digit is represented by two tones, determined by the intersection of the row and column where the digit sits. Two tones are used instead of only one to provide protection against false digits appearing during transmission due to noise.

Table 1: Key positions and corresponding DTMF tones

1	**2**	**3**	**A**	**697Hz**
4	**5**	**6**	**B**	**770Hz**
7	**8**	**9**	**C**	**852Hz**
*****	**0**	**#**	**D**	**941Hz**
1209Hz	**1336Hz**	**1477Hz**	**1633Hz**	

An example of DTMF tones in use is when your call is answered by a recorded message which instructs you to "press a key" to be transferred to a particular department or extension. It is also used to allow users to enter, for example, their credit card number to obtain a balance, when calling your credit card company.

B. Musical Notes

Musical notes are arranged in groups of twelve, called octaves. The notes that will be used in this experiment are in the octave containing frequencies ranging from 440Hz to 880Hz. The twelve notes in each octave are logarithmically spaced in frequency, with each note frequency being $2^{1/12}$ times the frequency of the next lowest note. Thus, a 1-octave pitch shift upwards corresponds to a doubling of the

frequencies of the notes in the original octave. Table 2 shows the ordering of notes in the octave to be used to synthesize the music in this experiment as well as the fundamental frequencies for those notes.

Table 2: Notes in the 440-880 Hz octave

Note	Frequency (Hz)
A	440
$A^{\#}, B^{b}$	$440*2^{1/12}$
B	$440*2^{2/12}$
C	$440*2^{3/12}$
$C^{\#}, D^{b}$	$440*2^{4/12}$
D	$440*2^{5/12}$
$D^{\#}, E^{b}$	$440*2^{6/12}$
E	$440*2^{7/12}$
F	$440*2^{8/12}$
$F^{\#}, G^{b}$	$440*2^{9/12}$
G	$440*2^{10/12}$
$G^{\#}, A^{b}$	$440*2^{11/12}$

C. Sound Effects

C.1 Echo

The echo is one of the simplest effects used in musical signal processing. The echo filter adds the current input to a delayed input with a gain that is less than 1 in order to produce the output. This is accomplished by using the delay routine shown in figure 1.

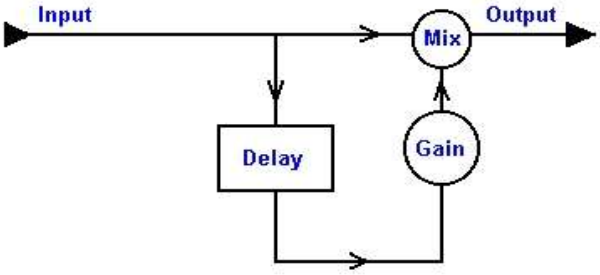


Figure 1: Illustration of echo system

C.2 Reverb

By putting a few echo filters in series, the user can create a reverberating filter, which causes the output to sound like a person yelling into the mountains. These filters are also known as comb filters. This is one of the main effects that are used on electric guitars to make notes last a lot longer and fade out. A schematic for a basic reverb filter is given in figure 2.

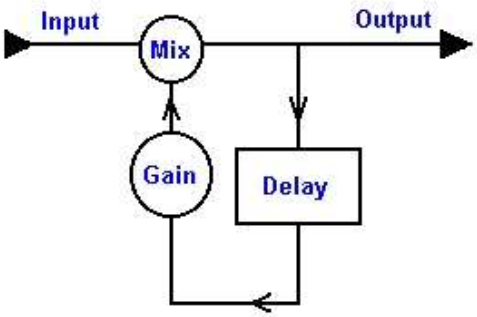


Figure 2: Illustration of reverb system

Working with Audio Signals

- Matlab has the ability to play an audio signal *x* at sampling rate `Fs` using the instruction:
  `sound(x,Fs);`

- To save a sound signal *x* created by Matlab in a *wav* file called (MYSONG), the following instruction is used:
  `wavwrite(x,'MYSONG');`

- To read an audio signal from an existing *wav* file called (MYSONG), the following instruction is used:
  `[x,Fs]=wavread('MYSONG');`

Procedures

A. *Musical Notes*

1- Enter the following code to an M-file, then run this M-file and report the result.

```
Fs = 8000;              % sampling rate
Ts = 1/Fs;
ToneDuration = 0.5;     % duration of each tone
n =0:(ToneDuration/Ts)-1; % calculate the number of samples
pit = 2*pi*n*Ts;        % calculate the pitch

f = [523.35, 587.33, 659.26, 698.46,...
783.99, 880.1, 987.77, 1046.50];
tone = [];
for i = 1:length(f);
      tone = [tone sin(f(i)*pit)];
end
sound([tone fliplr(tone)], Fs);
```

2- Write a program that creates a signal representing the notes given in figure (3) and then play back the created signal.

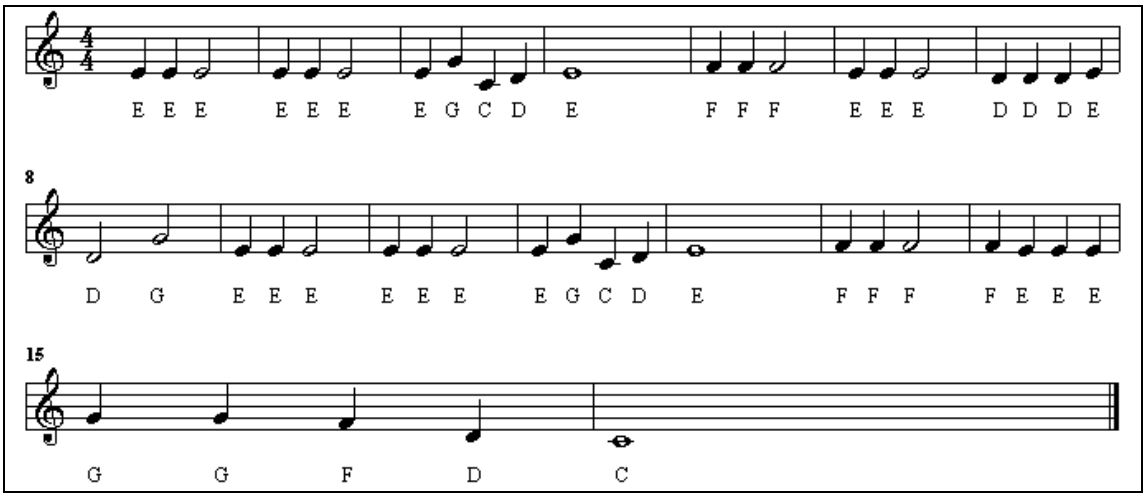


Figure 3: Musical Tones

3- Using the built-in function '*wavwrite*', write the song created in step 2 as a *wav* file.

B. *Sound Effects*

1- Using the built-in function '*wavread*', load the sampled data and sampling rate from the *wav* file available on the desktop of your computer.

2- What happens when playing back the signal in step 1 using:

   a) half the sampling frequency?

   b) twice the sampling frequency?

3- Determine the recording time of the signal in step 1.

4- What is the number of samples needed to create 500ms time delay if Fs=8000Hz?

5- Write a program that performs the echo system shown in Figure 1 and apply it to the signal in step 1 with Gain = 0.5 and delay =500ms.

6- Write a program that performs the reverb system shown in Figure 2 and apply it to the signal in step 1with Gain = 0.5 and delay = 500ms.

7- Add a silence of 3 seconds at the end of the original signal used in step 6 so that reverbs can be heard even after the original signal is completely played.

8- Fix the delay to 500ms, and then try the program in step 7 with different gains such as 0.3, 0.8 and 1, etc. What is the effect of the gain value on the resulting signal?

9- Fix the gain to 0.5, and then try the program in step 7 with different delays such as 100ms, 1000ms and 2000ms, etc. What is the effect of the delay value on the resulting signal?

Discussion

1. With using a sampling frequency of 8000Hz, write a program that generates all DTMF tones having 0.4 second length each.

2. Draw the block diagram and equation of an echo system that introduces 2 echo sounds to an input sound signal at the same time, delayed by 100ms and 200ms.

Ninevah University
College of Electronics Engineering
Communications Department
Microprocessors and DSP Lab
Third Class
Experiment No. (8)

Subroutines and Interrupt 21h Tasks

Objectives

- To examine various tasks performed by Interrupt 21h using 8086Mp Emulator.
- To learn how to write and call subroutines in 8086Mp Emulator.

Introduction

The MSDOS operating system contains many built-in subroutines for accessing the input and output devices which are connected to the computer (keyboard, screen, etc.). The interrupt instruction (INT) is often used to access some of these routines.

The instruction (INT 21h) will cause the computer to access the subroutines that perform various tasks depending on the value of (AH) register at the moment that the (INT 21h) instruction is performed. Some of those tasks are given in table 1.

Table 1: Various Interrupt 21h Tasks

AH	**The task performed by Interrupt 21H**
1	Read a character from keyboard and load it into (AL) register
2	Print the character that has its ASCII code stored in (DL) register on screen
9	Print the string stored starting at (DS:DX) on screen and stop when '$' is reached
10	Read a string from keyboard and store it in memory starting at (DS:DX+2), where the two bytes at (DS:DX) contain the total available storage and the actual number of entered bytes, respectively. (Enter) indicates the end of the string.

Subroutines

To write a subroutine and call it in the Emulator program, the following template can be used:

```
ORG 100H
   …….
   …….
   …….
   CALL   FUNC1
   CALL   FUNC2
   …….
   …….
   …….
   RET

   FUNC1   PROC
   …….
   …….
   …….
   RET
   FUNC1   ENDP

   FUNC2   PROC
   …….
   …….
   …….
   RET
   FUNC2   ENDP

NUM    DW   8888H
ARRAY1   DB   ?

END
```

Main Program

A subroutine called (FUNC1)

A subroutine called (FUNC2)

Procedure

Write a code in Assembly language to do the following:

1. Enter a character.
2. Make sure that the entered character is between 0 and 9. After that print the following message: (the user requested service number:) where the entered number must be printed at the end of the message.
3. If the character is out of that range, print the following error message: (ERROR, the number must be between 0 and 9, try again:), then the user is allowed to enter a character again.
4. Modify the code above so that if the entered character is between 0 and 9, print stars with the same number (for example: if the entered number is 4, then four stars are printed).
5. Modify the code so that each star is printed by calling a subroutine called (STARPRINT).
6. Repeat step (4) with the stars printed in a column and not in a row.

Ninevah University
College of Electronics Engineering
Communications Department
Microprocessors and DSP Lab
Third Class
Experiment sheet No. (9)

Applications on Interrupt 21H

Procedure

1. Write a code in Assembly language to read a number in Hexadecimal consisting of 2 digits. Put this number in BL register. Use only capital letters.

   (**NOTE:** read the first digit, store it, then read and store the second digit)

2. Rewrite the code in step 1 to read a number in Hexadecimal consisting of 4 digits (16-bit number). Put this number in BX.

   (**NOTE:** use loops instead of repeating your code four times)

3. Rewrite the code in step 2 with reading the 4 Hexadecimal digits of the number first, storing them in a buffer, then put the value of the entered number in BX.
4. Rewrite the code in step 3 with giving the user the ability to enter both capital and small letters.

Ninevah University
College of Electronics Engineering
Communications Department
Microprocessors and DSP Lab
Third Class
Experiment No. (10)

Discrete Fourier Transform (DFT)

Objectives

- To study and implement Discrete Fourier Transform (DFT) using MATLAB.
- To learn the analysis and synthesis of discrete-time signals using DFT and IDFT, respectively.

Introduction

The discrete-time Fourier transform (DTFT) provides frequency-domain representation of sequences. DTFT is a function of the continuous variable (ω), which is defined for infinite-length sequences. This is impractical because it makes DTFT a numerically *incomputable* transform.

Therefore, attention is turned to a numerically computable transform, obtained by sampling DTFT in the frequency domain. When applied to finite-duration sequences, it gives us a new transform called Discrete Fourier Transform (DFT), which is the ultimate numerically computable Fourier transform.

The Discrete Fourier Transform of an N-point sequence is given by:

$$X(k)=\sum_{n=0}^{N-1}x(n)W_N^{nk}, \qquad 0\le k\le N-1 \qquad (1)$$

where $W_N=e^{-j\frac{2\pi}{N}}$. Note that $X(k)$ (which is the result of DFT) is also an N-point sequence, where: $0\le k\le N-1$.

The inverse discrete Fourier transform (IDFT) of an N-point signal $X(k)$ is given by:

$$x(n)=\frac{1}{N}\sum_{k=0}^{N-1}X(k)W_N^{-nk}, \qquad 0\le n\le N-1 \qquad (2)$$

In MATLAB, an efficient implementation of DFT and IDFT would be to use a matrix-vector multiplication for the relations in equations (1) and (2). If $x(n)$ and $X(k)$ are arranged as row vectors **x** and **X**, respectively, then from equations (1) and (2) we have:

$$\mathbf{X}=\mathbf{x}\mathbf{W}_N \qquad (3)$$

$$\mathbf{x}=\frac{\mathbf{1}}{N}\mathbf{X}\mathbf{W}_N^{*} \ , \qquad (4)$$

where $\mathbf{W}_N$ and $\mathbf{W}_N^{*}$ are the DFT matrix and its conjugate, respectively. The following MATLAB functions are used to implement the above equations:

```
function [Xk] = dft(xn, N)

n = [0:1:N-1];       % row vector for n
k = [0:1:N-1];        % row vector for k
WN = exp(-j*2*pi/N); % Wn factor
nk = n'*k;            % creates a NxN matrix of nk values
WNnk = WN .^ nk;      % DFT matrix
Xk = xn * WNnk;       % row vector for DFT coefficients
% End of function

function [xn] = idft(Xk, N)

n = [0:1:N-1];       % row vector for n
k = [0:1:N-1];        % row vector for k
WN = exp(-j*2*pi/N);     % Wn factor
nk = n'*k;           % creates a NxN matrix of nk values
WNnk = WN .^ (-nk); % IDFT matrix
xn = (Xk*WNnk)/N;        % row vector of IDFT values
% End of function
```

Experiment

1. For the sequence $x(n)=[1,1,1,1]$, use the following MATLAB commands to compute DFT:

```
x = [1, 1, 1, 1]; N = 4;
X = dft(x, N);
magX = abs(X);
phaX = angle(X)*180/pi;
```

2. Plot the magnitude and phase of the DFT signal in step (1).
3. Apply zero-padding operation, which is a technique that allows us to sample at dense (or finer) frequencies, to get an 8-point sequence. Then compute its DFT. Use the following MATLAB commands:

```
x = [1, 1, 1, 1, zeros(1,4)]; N = 8;
X = dft(x, N);
magX = abs(X);
phaX = angle(X)*180/pi;
```

4. Plot the magnitude and phase of the 8-point DFT signal in step (3).
5. Treat $x(n)$ in step (1) as a 16-point sequence by padding 12 zeros instead of 4 zeros. Plot the magnitude and phase of the resulting 16-point DFT signal.
6. For the sequence: $x(n)=\cos(0.48\pi n)+\cos(0.52\pi n)$, use the following MATLAB commands that determine the 10-point DFT of $x(n)$ to obtain an estimate of its DTFT:

```
n = [0:1:99]; x = cos(0.48*pi*n)+cos(0.52*pi*n);
n1 = [0:1:9]; y1 = x(1:1:10);
subplot(2, 1, 1); stem(n1, y1);
```

```
title('signal x(n), 0 <= n <= 9');
xlabel('n');
Y1 = dft(y1, 10); magY1 = abs(Y1(1:1:6));
k1 = 0:1:5; w1 = 2*pi*k1/10;
subplot(2, 1,  2); plot(w1/pi, magY1);
title('DTFT Magnitude');
xlabel('Frequency in pi units');
```

7. Use the following commands that pad 90 zeros to $x(n)$ in step (6) to obtain a dense spectrum:

```
n2 = [0:1:99]; y2 = [x(1:1:10) zeros(1,90)];
subplot(2, 1, 1); stem(n2, y2);
title('signal x(n), 0 <= n <= 9 + 90 zeros');
xlabel('n');
Y2 = dft(y2, 100); magY2 = abs(Y2(1:1:51));
k2 = 0:1:50; w2 = 2*pi*k2/100;
subplot(2, 1, 2); plot(w2/pi, magY2);
title('DTFT Magnitude');
xlabel('Frequency in pi units');
```

8. Use the following commands to determine DTFT for the first 100 samples of $x(n)$:

```
subplot(2, 1, 1); stem(n, x);
title('signal x(n), 0 <= n <= 99');
xlabel('n');
X = dft(x, 100); magX = abs(X(1:1:51));
k =  0:1:50; w = 2*pi*k/100;
subplot(2, 1, 2); plot(w/pi, magX);
title('DTFT Magnitude');
xlabel('Frequency in pi units');
```

9. Use the following commands to synthesize the sequence $x(n)$ using the Inverse Discrete Fourier Transform (IDFT):

```
x1 = idft(X, 100);
subplot(2, 1, 2); stem(n, x1);
title('Synthesized x(n)');
xlabel('n');
```

Questions

1. How can you improve the spectrum density?
2. How can you improve the spectrum resolution?
3. What are the differences between DFT and IDFT?
4. What are the differences between DFT and DTFT?

Ninevah University
College of Electronics Engineering
Communications Department
Microprocessors and DSP Lab
Third Class
Experiment No. (11)

Fast Fourier Transform (FFT)

Objectives

1. To study and investigate the Fast Fourier Transform algorithm.
2. To learn the analysis and reconstruction of discrete-time signals using FFT and IFFT, respectively.

Introduction

The DFT introduced previously is the only transform that is discrete in both time and frequency domains, and it is defined for finite-duration sequences. Although it is a computable transform, its straightforward implementation is very inefficient, especially when the sequence length N is large. In 1965 Cooley and Tukey showed a procedure to substantially reduce the amount of computations involved in DFT. This led to the explosion of applications of the DFT, including digital signal processing area, and also led to the development of other efficient algorithms. These algorithms are collectively known as Fast Fourier Transform (FFT) algorithms.

In an efficiently designed algorithm, the number of computations should be constant per data sample, and therefore the total number of computations should be linear with respect to N. The number of DFT computations for an N-point sequence depends quadratically on N, which will be denoted by the notation:

$$C_N = o(N^2). \qquad (1)$$

FFT algorithms can reduce the quadratic dependence on N of the DFT. Theoretically, the number of computations used in the FFT algorithms could be as small as, depending on the radix used in the algorithm,

$$C_N = o(N \log_2 N) \qquad (2)$$

MATLAB provides a function called **fft** to compute the DFT of a vector ***x***. It is invoked by **x = fft(x, N),** which computes the N-point DFT. If the length of ***x*** is less than N, then ***x*** is padded with zeros. If the argument N is omitted, then the length of the DFT is the length of ***x***. If ***x*** is a matrix, then **fft(x,N)** computes the N-point of each column of ***x***.

Notice that the FFT algorithm is not a different mathematical transform, it is simply an efficient means to compute the DFT. **fft** function is written in machine language and not using MATLAB commands (i.e., it is not available as a *.m* file). Therefore it executes very fast. The inverse DFT is computed using the **ifft** function, which has the same characteristics as **fft**.

The execution time for FFT depends on the length of the transform. It is fastest for powers of two. It is almost as fast for lengths that have only small prime factors. It is typically several times slower for lengths that are prime numbers or which have large prime factors. It is important to notice that the spectrum of FFT is always displayed between 0 and *fs* as shown in figure 1. In addition, the signal after *0.5fs* is only a repetition of the first part and can be neglected.

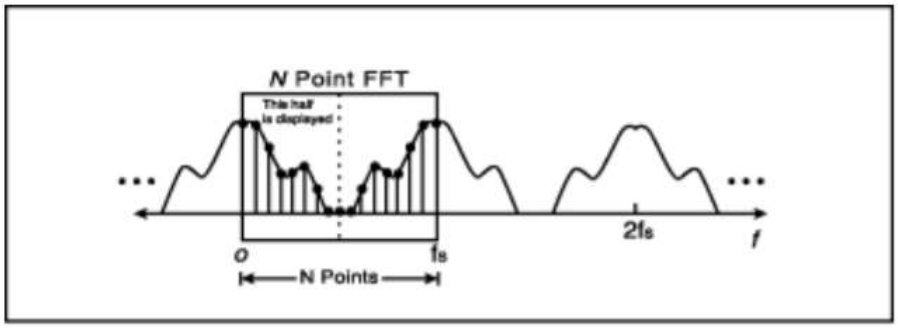


Figure (1): N-point FFT spectrum amplitude.

Experiment

1. Compute the FFT of *x[n]*, which is a cosine wave with a frequency of 10Hz and sampled using a sampling frequency of 100Hz. *N* (the number of points in the FFT) must be at least as large as the number of samples in *x[n]*. Use the following code to plot the spectrum amplitude:

```
clc
clear all
close all
fs=100;
Ts=1/fs;
n = 0:29;
x1 = cos(20*pi*n*Ts);
N = 30;
X1 = abs(fft(x1,N));
F = [0:N-1]/N *fs;
figure
stem(F,X1,'*');title('Amplitude of spectrum');
```

2. To demonstrate the effect of N on the spectrum, repeat the previous code with using three different values of N: 64, 128 and 256, then plot the resulting spectrum amplitudes in one figure using **`subplot`**.

   Theoretically, sinusoid should transform to an impulse in the frequency domain, why do we have *sincs* in the frequency domain? When FFT is computed with an *N* larger than the number of samples in *x[n]*, it fills in the samples after *x[n]* with zeros (as happened in step 2, where Matlab computed FFT with filling the spaces after *n* =30 with zeros). This is like taking a sine wave and multiplying it with a rectangular box of length 30. A multiplication of a box by a

sine wave in the time domain equals to the convolution of a *sinc* and impulse in the frequency domain. The previous Matlab experiment in step (2) supports this conclusion.

3. Reduce the execution time of FFT to the minimum by using N that equals to the next power of 2 from the length of *x[n]*, using **`nextpow2`** as follows:

```
N_FFT = 2^nextpow2(L); % L is the number of samples in x[n]
```

4. Compare the execution time of FFT with that of DFT for the signal in step (1) using N=2048 and 2039. Justify your results. Use the functions **`clock`**, **`etime`** and **`tic toc`**.

5. When the region between *0* and *fs* is examined, it can be seen that there is even symmetry around the center point *0.5fs*, where the data between *0.5fs* and *fs* is a mirror image of the data between *0* and *0.5fs*. Remove the redundant information in step (3) by displaying only half the spectrum amplitude.

6. A common use of Fourier transforms is to find the frequency components of a signal buried in a noisy time domain signal. For the signal in step (3), use a length of 1000 for *x[n]* and add random noise to *x[n]* using the function **`randn`** with an amplitude of 2, then plot both the noisy time domain signal and its spectrum amplitude in one figure using **`subplot`**.
   - Can you distinguish the frequency of the original signal in time domain?
   - Can you distinguish the frequency of the original signal in frequency domain?

7. Add a noise signal that has a single frequency of 40Hz (sine wave) to *x[n]*, then remove this frequency from the spectrum and synthesize the original signal using **`ifft`**, as follows:

```
clc
clear all
close all
fs=100;
Ts=1/fs;
N=1000;
n = [0:N-1];
x1 = cos(20*pi*n*Ts)+sin(80*pi*n*Ts);
subplot(211);
stem(n*Ts,x1);
title('Signal + noise');
xlabel('nTs')
ylabel('x(nTs)')
X1 = abs(fft(x1,N));
F =[0:N-1]/N *fs;
subplot(212);
stem(F,X1);
title('Amplitude of spectrum');
xlabel('F (Hz)');
```

```
ylabel('|X(F)|');
%%%%%%%%%%%%%% IFFT %%%%%%%%%%%%%
x3=[X1(1:N/4)  zeros(1,N/2)  X1(N*3/4+1:end)];
figure ;
subplot(211);
stem(F, x3);
xlabel('F (Hz)')
ylabel('|X(F)|')
xr=ifft(x3, length(x3));
subplot(212);
stem(n*Ts, xr)
xlabel('nTs')
ylabel('x(nTs)')
title('Original signal')
```

Ninevah University
College of Electronics Engineering
Communications Department
Microprocessors and DSP Lab
Third Class
Experiment No. (12)

FIR Filter Design Using Window Techniques

Objectives

1. To determine the specifications needed for FIR filter design.
2. To learn how to design FIR filters using window techniques.
3. To learn how to implement the designed FIR filters in MATLAB.

Introduction

In order to process signals, we have to design and implement systems called filters. The filter design issue is influenced by factors such as the type of the filter and the form of its implementations. Two types of digital filters can be identified, FIR and IIR filters. FIR filters are characterized by finite-duration impulse response. IIR filters are characterized by infinite-duration impulse response. In this experiment, the design and Matlab implementation of FIR filters are studied.

An FIR filter is a recursive filter that has a transfer function of the form:

$$H(z) = b_0 + b_1 z^{-1} + \cdots + b_{M-1} z^{1-M} = \sum_{n=0}^{M-1} b_n z^{-n} \quad , \qquad (1)$$

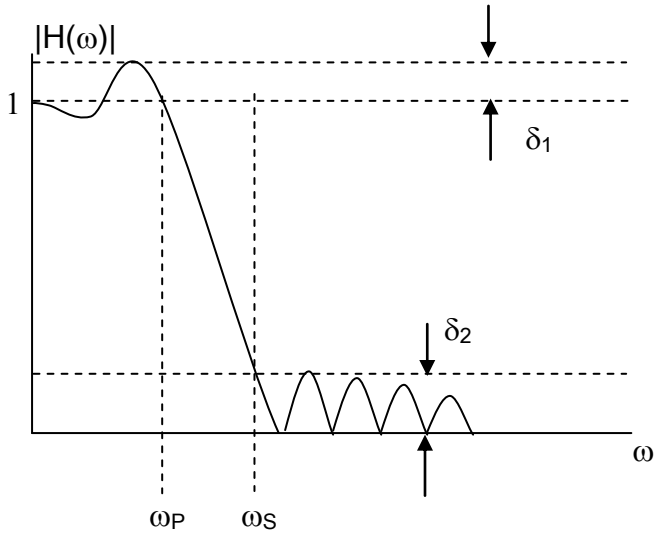
Hence the impulse response $h(n)$ is:

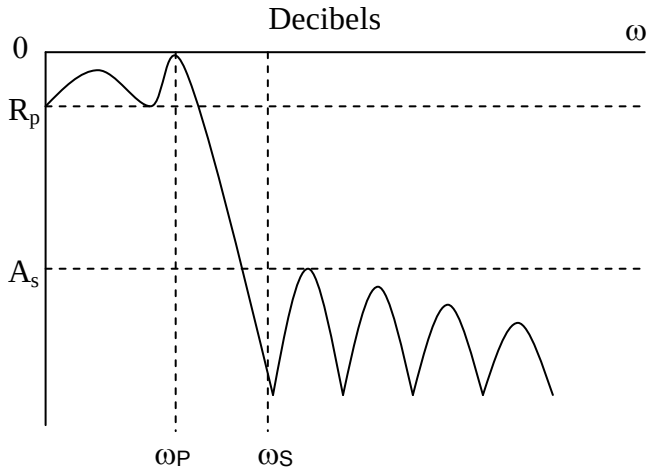
$$h(n) = \begin{cases} b_n, & 0 \le n \le M-1 \\ 0, & else \end{cases} \qquad (2)$$

and the difference equation representation is:

$$y(n) = b_0 x(n) + b_1 x(n-1) + \ldots + b_{M-1} x(n-M-1) \qquad (3)$$

Essentially, an FIR filter is designed based on specifications. Absolute and relative specifications of a lowpass filter are shown as:





where δ_1 is passband ripple, R_p is passband ripple in dB, δ_2 is stopband ripple, A_s is stopband ripple in dB, ω_p is passband edge frequency, and ω_s is stopband edge frequency. The parameters given in the above two specifications are obviously related by:

$$R_p = -20\log_{10}\frac{1-\delta_1}{1+\delta_1} > 0 \ (\approx 0)$$

$$A_s = -20\log_{10}\frac{\delta_2}{1+\delta_1} > 0 \ (>>1) \qquad (4)$$

Note: R_p and A_s are in decibel (dB). By the meaning of attenuation, we always have positive values for R_p and A_s.

The basic idea behind the window design is to select an appropriate *window* function and an appropriate *ideal* filter that provides a linear-phase and causal FIR filter.

We will denote an ideal lowpass filter by $H_d(e^{j\omega})$, which has a unity magnitude gain and linear-phase characteristics over its passband, and zero response over its stopband. An ideal LPF of bandwidth $\omega_c < \pi$ is given by:

$$H_d(e^{j\omega}) = \begin{cases} 1 \cdot e^{-j\alpha\omega}, & |\omega| \le \omega_c \\ 0, & \omega_c < |\omega| \le \pi \end{cases}$$

where ω_c is also called the cutoff frequency, and α is called the sample delay. The impulse response of this filter is of infinite duration and is given by:

$$\begin{aligned} h_d(n) &= \mathcal{F}^{-1}\left[H_d(e^{j\omega})\right] = \frac{1}{2\pi}\int_{-\pi}^{\pi} H_d(e^{j\omega})e^{j\omega n}d\omega \\ &= \frac{1}{2\pi}\int_{-\omega_c}^{\omega_c} 1 \cdot e^{-j\alpha\omega}e^{j\omega n}d\omega \\ &= \frac{\sin\left[\omega_c(n-\alpha)\right]}{\pi(n-\alpha)} \end{aligned}$$

Let $w(n)$ and $h_d(n)$ be the window function and the impulse response of the ideal filter, respectively. The impulse response of the designed filter can be formed by:

$$h(n) = h_d(n)w(n) \qquad (5)$$

There are several window functions available. Each of those window functions has different characteristics in both time-domain and frequency-domain. We now briefly describe various well-known window functions:

1. **Rectangular Window:** This is the simplest window function but it provides the worst performance from the viewpoint of stopband attenuation. It is defined by:

$$w(n)=\begin{cases}1, & 0\le n\le M-1\\ 0, & otherwise\end{cases} \tag{6}$$

2. **Bartlett Window:** Since the Gibbs phenomenon results from the fact that the Rectangular Window has a sudden transition from 0 to 1 (or 1 to 0), Bartlett suggested a more gradual transition in the form of a Triangular Window, which is given by:

$$w(n)=\begin{cases}\dfrac{2n}{M-1}, & 0\le n\le \dfrac{M-1}{2}\\ 2-\dfrac{2n}{M-1}, & \dfrac{M-1}{2}\le n\le M-1\\ 0, & otherwise\end{cases} \tag{7}$$

3. **Hanning Window:** This is a raised cosine window given by:

$$w(n)=\begin{cases}0.5*\left[1-\cos\left(\dfrac{2\pi n}{M-1}\right)\right], & 0\le n\le M-1\\ 0, & otherwise\end{cases} \tag{8}$$

4. **Hamming Window:** This window is similar to the Hanning Window except that it has a small amount of discontinuity and is given by:

$$w(n)=\begin{cases}0.54-0.46*\cos\left(\dfrac{2\pi n}{M-1}\right), & 0\le n\le M-1\\ 0, & otherwise\end{cases} \tag{9}$$

5. **Blackman Window:** This is also similar to the previous two but contains a second harmonic term and is given by:

$$w(n)=\begin{cases}0.42-0.5\cos\left(\dfrac{2\pi n}{M-1}\right)+0.08\cos\left(\dfrac{4\pi n}{M-1}\right), & 0\le n\le M-1\\ 0, & otherwise\end{cases} \tag{10}$$

Given the transition width and minimum stopband attenuation of the desired filter, we can select a suitable window type and widow size based on the information in Table 1.

6. Kaiser window:

$$w(n) = \begin{cases} I_0\left(\beta\left(1-\left((n-\alpha)/\alpha\right)^2\right)^{1/2}\right)/I_0(\beta) & 0 \le n \le M-1 \\ 0 & otherwise \end{cases} \quad (11)$$

where $\alpha = (M\text{-}1)/2$ and I_0 = the zeroth order Modified Bessel Function.

If ω_p, ω_s, R_p, and A_s are given, then the following equations are needed for the design:

$$\text{Transition width} = \Delta f = \frac{\omega_s - \omega_p}{2\pi}$$

$$\text{Filter order } M \cong \frac{A_s - 7.95}{14.36\Delta f} + 1 \quad (12)$$

$$Parameter\ \beta = \begin{cases} 0.1102(A_s - 8.7), & A_s \ge 50 \\ 0.5842(A_s - 21)^{0.4} + 0.07886(A_s - 21), & 21 > A_s > 50 \\ 0 & A_s < 21 \end{cases}$$

Window Name	Transition Width Δω		Min. Stopband Attenuation
	Approximate	Exact Values	
Rectangular	$\frac{4\pi}{M}$	$\frac{1.8\pi}{M}$	21 dB
Bartlett	$\frac{8\pi}{M}$	$\frac{6.1\pi}{M}$	25 dB
Hanning	$\frac{8\pi}{M}$	$\frac{6.2\pi}{M}$	44 dB
Hamming	$\frac{8\pi}{M}$	$\frac{6.6\pi}{M}$	53 dB
Blackman	$\frac{12\pi}{M}$	$\frac{11\pi}{M}$	74 dB

Table 1. Summary of commonly used window function characteristics

MATLAB provides several routines to implement window functions. These routines are:

- **`w = boxcar(M)`** returns the M-point Rectangular Window function in array w.
- **`w = triang(M)`** returns the M-point Bartlett (triangular) Window function in array w.
- **`w = hanning(M)`** returns the M-point Hanning Window function in array w.
- **`w = hammimg(M)`** returns the M-point Hamming Window function in array w.
- **`w = blackman(M)`** returns the M-point Blackman Window function in array w.
- **`w = kaiser(M,betta)`** returns the M-point Kaiser Window function in array w.

To display the frequency-domain plots of digital filters, the Matlab function **freqz** is used to return the magnitude response **H** and the frequency samples **w** over which, frequency response is plotted. **freqz** finds **H** and **w** from the transfer function of the filter. The following example uses **freqz** to find the 1000 points frequency response of a digital filter:

```
[H,w] = freqz(b,a,1000,'whole');%freqz returns the frequency response
                                of a filter from its transfer function.
% b = numerator polynomial of H(z) (for FIR: b=h).
% a = denominator polynomial of H(z) (for FIR: a=[1]).
% w = frequency samples between 0 to 2π.
% H = frequency response vector.
```

Experiment

1. To design FIR filters based on the window technique, an ideal lowpass impulse response $h_d(n)$ is required. Therefore it is convenient to have a simple routine that creates $h_d(n)$ as shown below:

```
function hd = ideal_lp(wc, M)
% Ideal LowPass filter computation
% ---------------------------------------
% [hd] = ideal_lp(wc, M)
%   hd = ideal impulse response between 0 to M-1
%   wc = cutoff frequency in radians
%    M = length of the ideal filter

alpha=(M-1)/2;
n=[0:1:(M-1)]
M = n-alpha + eps;  % add smallest number to avoid division by zero
hd = sin(wc*M) ./ (pi*M);
```

2. With the given specifications: $\omega_p = 0.3\pi,\ \omega_s = 0.4\pi,\ R_p = 0.25\,\text{dB}, and\ A_s = 50\,\text{dB}$, use the following commands as design steps of the FIR lowpass filter using Hamming Window:

```
wp = 0.3*pi; ws = 0.4*pi;
tr_width = ws-wp;
M = ceil(6.6*pi/tr_width)+1;
n = [0:1:M-1];
wc = (ws+wp)/2;
hd = ideal_lp(wc,M);
w_ham = (hamming(M));
h = hd .* w_ham';
[H,w] = freqz(h,[1], 1000, 'whole');
H=(H(1:1:501))'; %taking half of the frequency response samples.
w=(w(1:1:501))';%taking half the frequencies (between 0 and π radians).
mag=abs(H); %mag =absolute magnitude computed over 0 to π radians.
db = 20*log10((mag+eps)/max(mag));  %db = Relative magnitude in dB.
pha = angle(H); %Phase response in radians over 0 to pi radians.
figure;
```

```
subplot(2,2,1); stem(n,hd); title('Ideal Impulse Response')
xlabel('n'); ylabel('hd(n)')
subplot(2,2,2); stem(n, w_ham); title('Hamming Window')
xlabel('n'); ylabel('w(n)')
subplot(2,2,3); stem(n,h); title('Actual Impulse Response');
xlabel('n'); ylabel('h(n)');
subplot(2,2,4); plot(w/pi, db); title('Magnitude Response in dB');
grid on
xlabel('frequency in pi units'); ylabel('Decibels')
```

3. Find the values of M, R_p, and A_s of the filter designed in step (2).
4. Using a MATLAB function called **`filter`**, apply the filter designed in step (2) to signal x, where x is a sine wave with w=0.3π, distorted with a random noise of amplitude 0.1 using **`rand`** function. Then plot the input signal and the filtered output signal in one figure. Use the following code:

```
t=0:0.001:10;
x=sin(0.3*pi*t)+0.1*rand(size(t));
y=filter(h,1,x);
figure;
plot(t,x,'r')
hold on;
plot(t,y)
legend('noisy signal','filtered signal');
```

5. Design an FIR highpass filter using Kaiser Window with the following specifications:

```
wp = 0.3*pi; ws = 0.4*pi; As = 50;
    tr_width = (ws-wp)/(2*pi);
```

6. Kaiser Window:

6.1 Write your own equation to compute the filter order M for Kaiser Window:

```
M =
n = [0:1:M-1];
wc = (ws+wp)/2;
hd = ideal_lp(pi, M)-ideal_lp(wc,M);
```

6.2 Write your own equation to compute parameter β for Kaiser Window:

```
betta =
```

6.3 Write your own equation to compute Kaiser Window coefficients:

```
w_kaiser =
h = hd .* w_kaiser;
[H,w] = freqz(h,[1], 1000);
mag=abs(H);
db = 20*log10((mag+eps)/max(mag));
figure;
subplot(2,2,1); stem(n,hd); title('Ideal Impulse Response')
```

```
xlabel('n'); ylabel('hd(n)')
subplot(2,2,2); stem(n, w_kaiser); title('Kaiser Window')
xlabel('n'); ylabel('w(n)')
subplot(2,2,3); stem(n,h); title('Actual Impulse Response');
xlabel('n'); ylabel('h(n)');
subplot(2,2,4); plot(w/pi, db); title('Magnitude Response in dB');
grid
xlabel('frequency in pi units'); ylabel('Decibels')
```

Discussion

1. Given the specifications for a digital bandpass filter as below:

lower stopband edge :	$\omega_{1s} = 0.2\pi,$	$A_s = 60$ dB
lower passband edge :	$\omega_{1p} = 0.35\pi,$	$R_p = 1$ dB
upper passband edge :	$\omega_{2p} = 0.65\pi,$	$R_p = 1$ dB
upper stopband edge :	$\omega_{2s} = 0.8\pi,$	$A_s = 60$ dB

Design the filter using Blackman Window.

2. What is a built-in MATLAB function that can be used to design FIR filters? Explain how to use that function.

Ninevah University
College of Electronics Engineering
Communications Department
Microprocessors and DSP Lab
Third Class
Experiment No. (13)

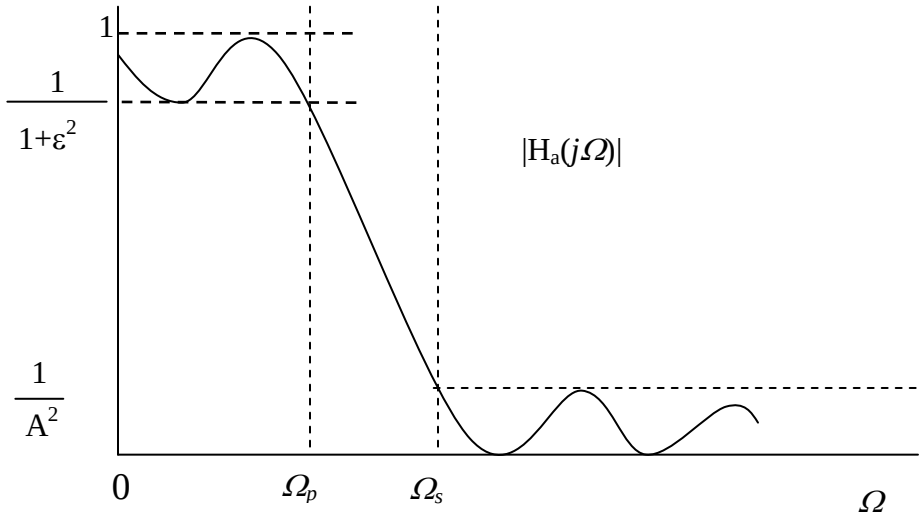
IIR Digital Filter Design

Objectives

1. To determine the specifications needed for IIR filter design.
2. To learn how to design and implement IIR filters using MATLAB.

Introduction

IIR filters have infinite-duration impulse responses; hence they can be matched to analog filters, all of which generally have infinitely long impulse responses. Therefore, the basic technique of IIR filter design transforms well-known analog filters into digital filters. Typically, specifications of IIR filters are shown in the relative linear scale as below.



The parameters ε and A are related to parameters R_p and A_s, respectively, of the dB scale. These relations are given by:

$$R_p = -10\log_{10}\frac{1}{1+\varepsilon^2} \Rightarrow \varepsilon = \sqrt{10^{R_p/10}-1}$$

$$A_s = -10\log_{10}\frac{1}{A^2} \Rightarrow A = 10^{A_s/20}$$

The ripples, δ_1 and δ_2, of the absolute scale are related to ε and A by:

$$\frac{1-\delta_1}{1+\delta_1} = \sqrt{\frac{1}{1+\varepsilon^2}} \Rightarrow \varepsilon = \frac{2\sqrt{\delta_1}}{1-\delta_1}$$

and:

$$\frac{\delta_2}{1+\delta_1} = \frac{1}{A} \Rightarrow A = \frac{1+\delta_1}{\delta_2}$$

In MATLAB, there are 4 functions to design digital lowpass filters. These same functions can also be used to design highpass, bandpass and bandstop filters. For purpose of illustration, the function butter will be used. It can be used with the following variations in its input arguments.

- **`[b,a]= butter(N,wn)`** designs an N[th]-order lowpass filter with the cutoff frequency **`wn`** in units of π.
- **`[b,a]= butter(N,wn,'high')`** designs an N[th]-order highpass filter with the cutoff frequency **`wn`** in units of π.
- **`[b,a]= butter(N, wn)`** designs an order 2N bandpass filter if **`wn`** is a two-element vector, **`wn= [w1,w2]`**, with 3-dB passband (**`w1 < w < w`**2) in units of π.
- **`[b,a]= butter(N, wn, 'stop')`** designs an order 2N bandstop filter if **`wn=[w1,w2]`** with 3-dB stopband (**`w1 < w < w2`**) in units of π.

To design any frequency-selective Butterworth filter, we need to know the order **`N`** and the 3-dB cutoff frequency vector **`wn`**. With its Signal Processing toolbox, MATLAB provides a function called **`buttord`** to compute these parameters. Givens the specifications: ω_p, ω_s, R_p and A_s, this function determines the necessary parameters. Its syntax is:

```
[N, wn] = buttord(wp, ws, Rp, As)
```

The parameters **`wp`** and **`ws`** have some restrictions, depending on the type of filter:

- for lowpass filters: **`wp < ws`**,
- for highpass filters: **`wp > ws`**,
- for bandpass filters: **`wp`** and **`ws`** are two-element vectors: **`wp = [wp1, wp2]`** and **`ws = [ws1, ws2]`**, such that: **`ws1 < wp1 < wp2 < ws2`**
- for bandstop filters: **`wp1 < ws1 < ws2 < wp2`**.

Now using the **`buttord`** function in conjunction with the **`butter`** function, we can design any Butterworth IIR filter. Similar discussions apply for **`cheby1`**, **`cheby2`**, and **`ellip`** functions corresponding to Chebyshev-I, Chebyshev-II and Elliptic IIR filters, respectively.

Experiment

1. For the following specifications:

$$\omega_p = 0.2\pi,\ \omega_s = 0.3\pi,\ R_p = 1\ dB,\ \text{and}\ A_s = 15\ dB$$

Enter the following commands as design steps of the digital Butterworth lowpass filter:

```
wp = 0.2*pi;
ws = 0.3*pi;
Rp = 1;
As = 15;
[N, wn] = buttord(wp/pi, ws/pi, Rp, As);
[b, a] = butter(N, wn);
```

2. Find the order and the cutoff frequency of the filter designed in step (1), then use the function **`freqz`** to plot the magnitude response, phase response and magnitude in dB of the filter.

3. Using the MATLAB function **`filter`**, apply the filter designed in step (1) to signal **`x`**, where **`x`** is a sine wave with w=0.3π, distorted with a random noise of amplitude 0.1 using **`rand`** function. Plot the input signal and the filtered output signal in one figure. Use the following code:

```
t=0:0.001:10;
x=sin(0.3*pi*t)+0.1*rand(size(t));
y=filter(b,a,x);
figure;
plot(t,x,'r')
hold on;
plot(t,y)
legend('noisy signal','filtered signal');
```

4. For the following specifications:

$$\omega_p = 0.6\pi,\ \omega_s = 0.4586\pi,\ R_p = 1\ dB,\ \text{and}\ A_s = 15\ dB$$

Enter the following commands as design steps of the digital Chebyshev-I highpass filter:

```
wp = 0.6*pi;
ws = 0.4586*pi;
Rp = 1;
As = 15;
[N, wn] = cheb1ord(wp/pi, ws/pi, Rp, As);
[b, a] = cheby1(N, Rp, wn, 'high');
```

5. Find the order and the cutoff frequency of the filter designed in step (4), then use the function **`freqz`** to plot the magnitude response, phase response and magnitude in dB of the filter.

6. For the following specifications:

$$\omega_{p1} = 0.25\pi, \omega_{p2} = 0.8\pi, \omega_{s1} = 0.4\pi,\ \omega_{s2} = 0.7\pi,\ R_p = 1\ dB, \text{ and } A_s = 40\ dB$$

Enter the following commands as design steps of the digital Chebyshev-II bandstop filter:

```
ws = [0.4*pi 0.7*pi];
wp = [0.25*pi 0.8*pi];
Rp = 1;
As = 40;
[N, wn] = cheb2ord(wp/pi, ws/pi, Rp, As);
[b, a] = cheby2(N, As, ws/pi, 'stop');
```

7. Find the order and the cutoff frequency of the filter designed in step (6), then use the function **`freqz`** to plot the magnitude response, phase response and magnitude in dB of the filter.

Discussion

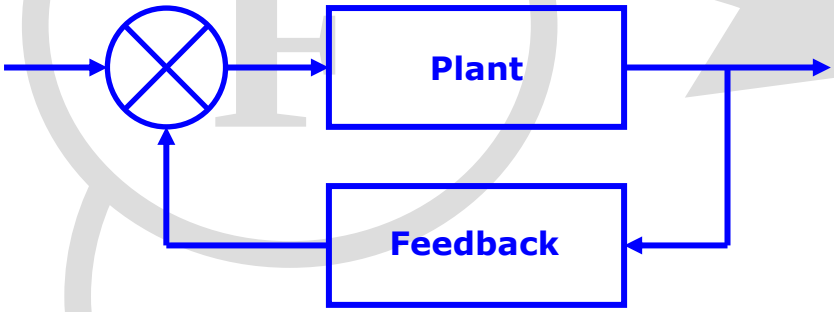
1. Design a digital Elliptic lowpass filter with the following specifications:

$$\omega_p = 0.3\pi,\ \omega_s = 0.4\pi,\ R_p = 1\ dB, \text{ and } A_s = 40\ dB$$

2. Explain the advantages and disadvantages of IIR Filters compared to FIR filters.

Feedback

Control-instrumentation



University of Mosul
College of Electronics Engineering.
Communication Department
3rd Class

Sh ( ) Speed Control Loops

Objective

This assignment is about controlling the speed of a motor using both analogue and digital speed sensors.

The experiments in this assignment are:

1. A speed control system using an analogue sensor.

2. A speed control system using a digital sensor.

Speed Servo Mechanisms

A speed control system using feedback is used when it is necessary to have a shaft rotating at a certain speed independent of load. In some applications the speed may be fixed but in others it may need to change in response to a varying set value.

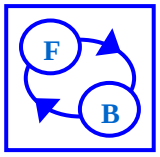
As in positional control this experiment uses a simple proportional feedback system with the error signal controlling the motor. It follows that there must always be an error in the system or the motor would stop.

Multiplying the actual error by a gain and using this to drive the motor can help reduce the required minimum error. However, there is a limit to how much gain can be applied before instability occurs.

As well as instability there are other problems which can occur such as poor performance due to tachometer ripple.

More complex systems solve some of these problems by using other methods like integral control which does allow the motor to keep running with zero error; this is explained later.

Tachometers



Various types of sensor can be found in control systems, the most commonly used are sensors for *position*, *velocity*, and *acceleration*.

A frequent requirement in a mechanical control system is to measure the speed of a motor. Various methods are available, such as to measure the shaft position and then differentiate this signal. This method tends to be erroneous when the position measurement contains high frequency noise (the amplitude of the noise changes rapidly).

The differentiation will increase the amplitude of this noise since the differentiator output is equal to the rate of change of signal amplitude, therefore resulting in corruption of the data.

A better method is to measure the velocity of the motor directly using a device called tachometer, which is a dc voltage generator. The generator is a permanent magnet motor which produces a back e.m.f proportional to the shaft speed. The polarity of the tachometer output voltage changes with the shaft rotation direction.

An ideal tachometer (more properly called a tachogenerator) produces a dc output, but in practice there is a superimposed ac ripple. Specialised designs can minimise this unwanted effect.

In a control system, tachometer ripple may be amplified sufficiently so as to cause performance deterioration, and a filter to remove it may be required. However, this filter can cause further problems by introducing delays in the response.

Digital to Analogue Converters

A digital signal is a sequence of pulses, where each pulse is represented by a digital word which has a finite number of bits (binary digits).

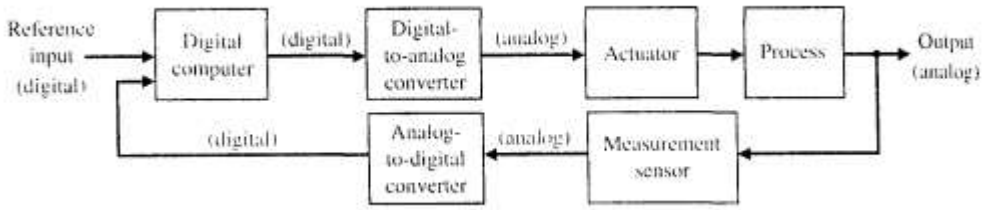
Digital signals derived from a computer may be required in a continuous (analogue) form, for various applications. Typical examples are motor control or audio signals in a digital communication system.

A D/A converter (or simply DAC), is a circuit that provides an analogue voltage (or current) that is the weighted sum of the bits in the digital word.

In an A/D conversion, the analogue waveform is sampled and the resulting discrete-time analogue samples are quantised (see quantisation) by truncating or rounding, and finally these quantised values are encoded into a digital word.

Therefore, unlike the A/D conversion, the D/A conversion process is unique; that is, there is a one-to-one correspondence between a given digital word and an analogue value.

However, there are gaps between levels, arising from the existing quantised values. Thus, the converted signal has abrupt discontinuities and presents a



staircase type of pattern. Hence, low pass filtering is often required to smooth the restored analogue signal by removing the high-frequency content.

An eight bit device gives 256 levels and a 10 bit one 1024 levels. The output voltage is often passed through a buffer to obtain the range required for a particular application.

The D/A converter in the Digital Unit is an eight bit converter and the output is buffered to give a range of +10 to -10 volts

Digital Shaft Position Sensing

The advanced shaft position or speed measurements can be obtained by employing digital techniques.

In this case, a *rotary encoder* is used as a position or speed sensor. There is Two encoders The absolute encoder and the incremental encoder that will be dealt with in this assignment.

Digital Shaft Encoders are relatively more complex and more expensive than a potentiometer used for shaft position sensing .However, the advantages of the digital encoders over a potentiometer are numerous: less prone to noise and error, easy determination of the direction of rotation and of the absolute position, including the number of complete disc revolutions.

Also, if digital computer control is required, there is no need for an A/D converter with its associated calibration problems.

Moreover, it is easy to measure speed by counting encoder pulses whereas in an analogue system with a potentiometer, a separate tachometer would be needed.

Incremental Encoders

The Optical Incremental Encoder is less complex than the absolute encoder. It consists of a rotating disc having a track of transparent windows. On one side of the disc, there is a light source, and just opposite to it, on the other side of the disc there is a light sensor.

A photograph of the sensor that is used with the hardware is shown below:

The encoder outputs a voltage pulse every time a transparent window passes the light source. With this sensor, electronic circuitry must be available to count the pulses, in order to determine the angle of rotation.

This encoder is called an incremental encoder, because the generation of a pulse indicates an incremental change in position, and not the actual (absolute) position.

A reference window is required to determine absolute position. The reference window is used for initialisation, representing zero position. In order to be able to determine the direction of rotation, a second light sensor is placed at a different window of the encoder track and is offset from the first sensor. The second sensor points at the edge of a window, when the first sensor points at the centre of a window.

Thus, for movement at a constant velocity, sensor 1 square wave output leading sensor 2 output by 90^{o} indicates one direction of rotation; sensor 1 output lagging sensor 2 output by 90^{o} indicates the other direction of rotation. You can see the two encoders in the mechanical unit . Most encoders are of the incremental type, because these can have greater resolution due to the simpler track arrangement and they are much cheaper.

Motor Static Friction (Stiction)

An ideal motor will turn very slowly for a small applied voltage. In practical motors there is a constant friction due to the brushes which prevent the motor rotating with small voltages. In specially designed motors this can be much reduced.

The effect of this friction is that if the voltage applied to the motor is slowly increased from zero, the motor remains stationary and then suddenly rotates at a significant speed when the torque overcomes the brush friction.

This effect is called stiction and impairs precise system control. It can be reduced by using PWM because the motor vibrates, or as in the Digital Unit, a drive amplifier with a non-linear response at low drive voltages.

Practical 1: Speed Control with Analogue Sensor

In this practical the computer is used, together with the **tachometer**, to make a simple closed loop speed control system.

Step 1:

1-Connect the close loop circuit show in the computer screen, Set ***SW1*** up to +10 and Adjust the magnitude of the. set value change with the ***disturbance control*** on the digital unit and the ***frequency control*** on the mechanical unit. To run the motor at *1000 r/min* (*31.25 r/min* at output) with no brake. Observe the behavior of the error and measured value for different values of gain using the drop-down box to change between the two outputs.

For different value of brake measure and fill the following table:

Brake	**Noload**	**Full load**	**Gain**
Speed (rpm)			1
Speed droop (rpm)			3

2- Connect the open loop circuit show in the computer screen, Set ***SW1*** up to +10 and Adjust the magnitude of the. set value change with the ***disturbance control*** on the digital unit and the ***frequency control*** on the mechanical unit. To run the motor at *1000 r/min* (*31.25 r/min* at output) with no brake. Observe the behavior of the error and measured value for different values of gain using the drop-down box to change between the two outputs.

For different value of brake measure and fill the following table:

Brake	**No load**	**Full load**	**Gain**
Speed (rpm)			1
Speed droop (rpm)			3

3-return to the close loop circuit and change the set value to triangle wave generator with 0.1Hz (is used to provide an input to the system that the

output shaft attempts to track). The input (called the set value) is connected to the A/D. converter and on to the computer. Let the motor running at 500RPM then observe the response and the error.

4- Change the set value to square wave with 0.1Hz then let the motor running at 500RPM then observe the response and the error.

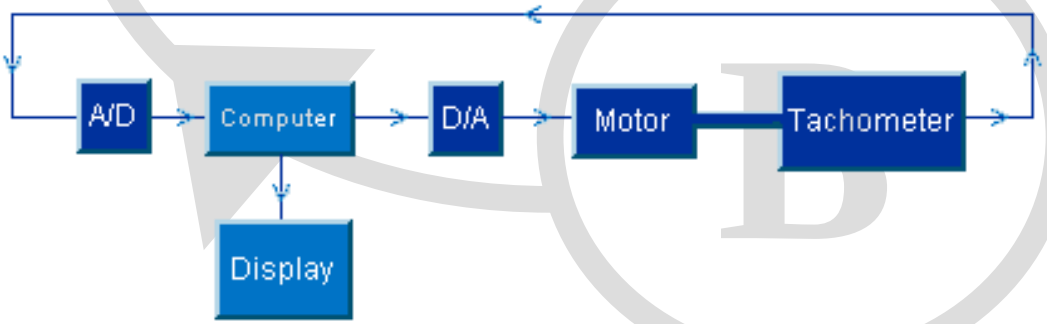
Note 1:

The actual speed of the shaft (the measured value) is sensed by the tachometer and this is also sent to the computer via the same A/D converter using a controlled switch to multiplex the A/D input.

Note 2:

The difference between the two inputs (called *error*) is calculated and the result used to drive the motor via the D/A converter.

This diagram shows how the system blocks are configured for this practical.



Practical 2: Speed Control with Digital Sensor

This practical shows that a positional servo control system can be implemented using a **digital shaft encoder** instead of the tachometer and A/D converter

Note 3:

The concept is the same as in the previous practical, the only difference is that the measured value is derived by counting pulses from the incremental shaft encoder.

Note 4:

The response time of this system is slow because of the fixed time taken to count the pulses.

Step 2:

1- Connect the close loop circuit show in the computer screen, Set ***SW1*** up to +10 and Adjust the magnitude of the. set value change with the ***disturbance control*** on the digital unit and the ***frequency control*** on the mechanical unit. To run the motor at *1000 r/min* (*31.25 r/min* at output) with no brake. Observe the behavior of the error and measured value

for different values of gain using the drop-down box to change between the two outputs.

Brake	**No load**	**Full load**	**Gain**
Speed (rpm)			1
Speed droop (rpm)			3

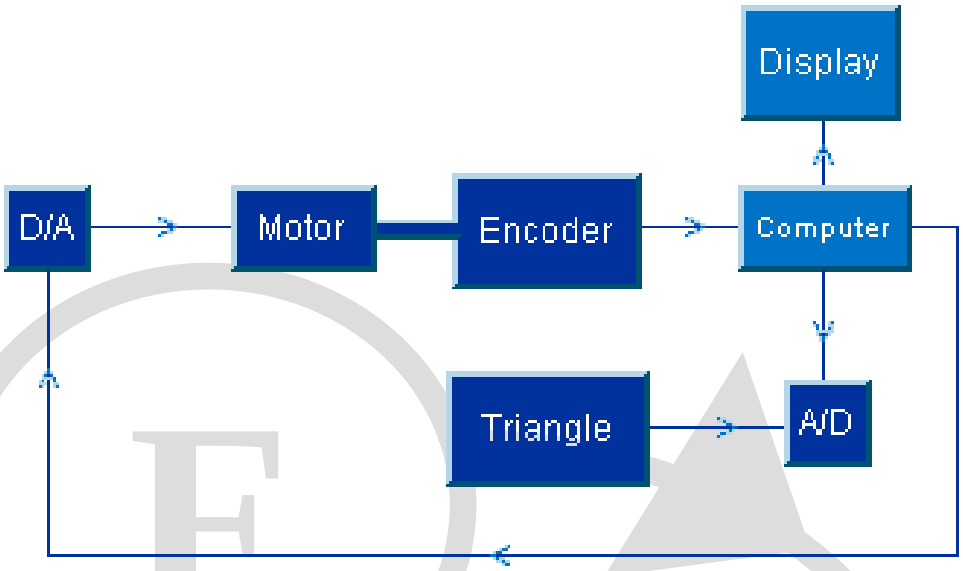
2- Change the set value to square wave with 0.1Hz and let the motor running at 500RPM then observe the response and the error.

3- Change the set value to triangle wave with 0.1Hz and let the motor running at 500RPM then observe the response and the error.

Note 5:

In order for the actual error to be small the error signal that drives the motor must be magnified so that small errors still correct the output, this magnification factor is called gain.

This diagram shows how the system blocks are connected for this practical.



Note 6:

The gain value has a large effect on the behaviour of the system and too much or too little both cause problems. Note that with low gain the error is large, but with high gain the system becomes unstable.

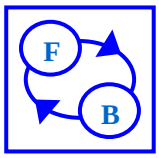
Questions in class work:

1- Why does increasing the load have little effect on the speed when using the proportional control system?

2-What effect does increasing the rate of change of the input signal have on the error signal?

3- What other factors involving the motor will affect the accuracy of control?

4- Compare the advantages and disadvantages of the two methods of speed measurement (tachogenerator and shaft encoder).


F
B

Microprocessors Laboratory

College of Electronics Engineering **Experiment sheet No.( )**
Communication Engineering Dept.

Electrical First Order System Analysis and Block diagram reduction

Object:

1. To derive the transfer function of the linear first order electric system, to plot simulated step response by using MATLAB program and Simulink, to study and investigate the experimental step response characteristics, and to compare it with the simulated step response .
2. To use the MATLAB program for the reduction of block diagram that represents complicated control system and then to transform this system to the standard or canonical form.

Theory:

Part A: Electrical First Order System Analysis

One of the most important tasks in the analysis of control systems is the mathematical modeling of the systems.
A mathematical modeling of a dynamic electric system is defined as a set of equations that represents the dynamics of the system accurately or, at least, fairly well.

The dynamics of electrical systems may be described in terms of differential equations. Such equations may be obtained by using Kirchhoff's laws. For any input the output response is obtained by solving these differential equations. However, this method is difficult. For these reasons the linear differential equation method is replaced by the transfer function concept.

The transfer function is defined as the ratio of the Laplace Transform of the output variable from the system to the Laplace Transform of the input signal causing that output, in this case all initial conditions must be zero.
The simplest electric first order control system can be represented by the passive filter (R–C) or (L–R) circuits shown in fig. (3-1) and (3-2).

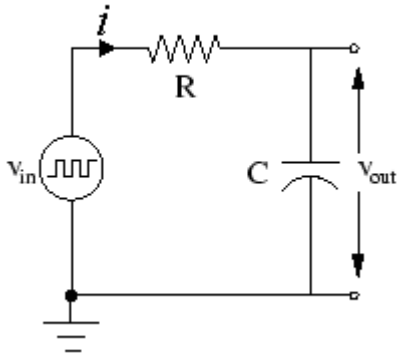


Fig. (3-1) R-C circuit

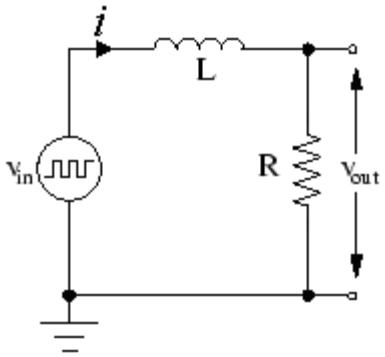


Fig. (3-2) L-R circuit

By applying Kirchhoff's laws to the (L-R) circuit yields

$$V_i(t) = L\frac{di}{dt} + V_o(t) \quad \text{..... (3-1)}$$

$$V_o(t) = Ri \quad \text{..... (3-2)}$$

Applying the Laplace Transform on both sides

$$V_i(s) = LsI + V_o(s) \quad \text{..... (3-3)}$$

$$V_o(s) = RI \quad \text{..... (3-4)}$$

$$I = \frac{V_o(s)}{R} \quad \text{..... (3-5)}$$

$$V_i(s) = \frac{LsV_o(s)}{R} + V_o(s) \quad \text{..... (3-6)}$$

$$\frac{V_o(s)}{V_i(s)} = \frac{1}{\frac{L}{R}s+1} = \frac{1}{\tau s+1} \quad \text{..... (3-7)}$$

Where $\frac{V_o(s)}{V_i(s)}$ is the transfer function of (L-R)

And τ is the electric time constant

The transient response to a unit step for a first order control can be represented as shown in fig. (3-3).

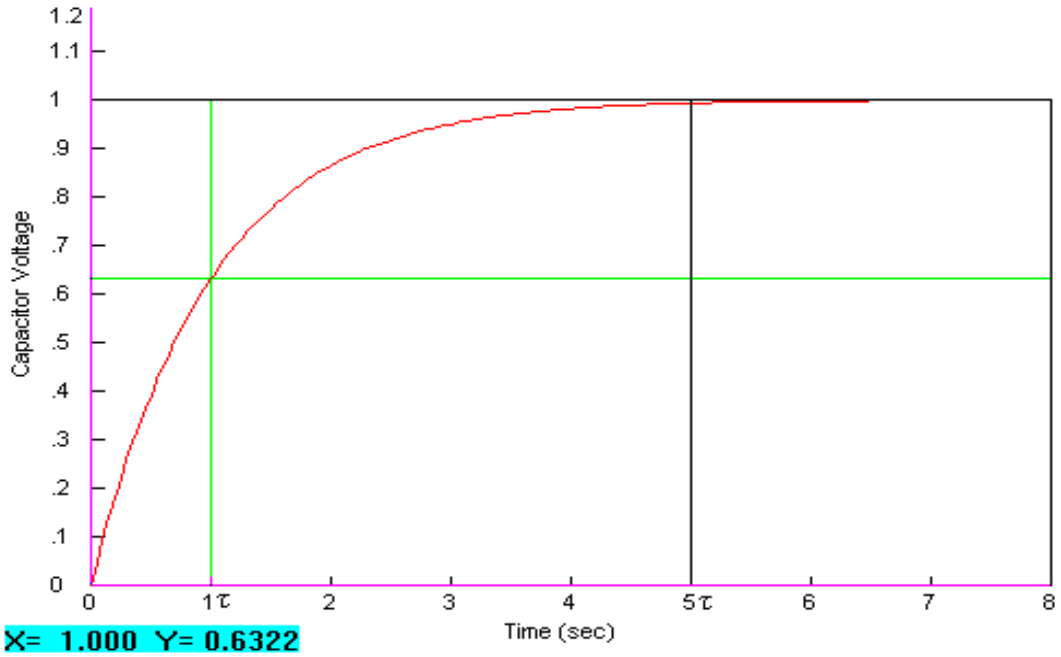


Fig. (3-3) transient response

The Simulink representation for (L-R) circuit transfer function is shown in fig(3-4).

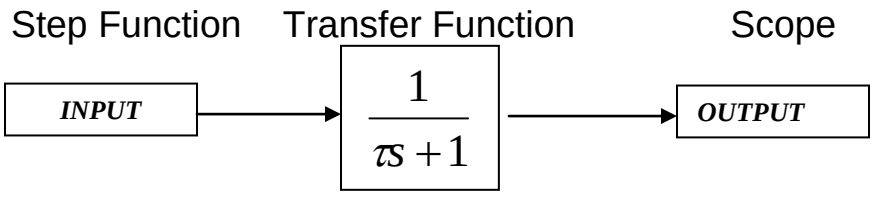


Fig. (3-4) simulink representation (block diagram)

The MATLAB program below is used to plot the step response of first order system.

num=1, den=[τ 1]
sys=tf(num,den)
printsys=(num,den)
step(sys)

Where:
num = numerator of the transfer function
den = denominator of the transfer function
tf =transfer function
sys = system
printsys(NUM,DEN,'s') or printsys(NUM,DEN,'z') prints the transfer function as a ratio of two polynomials in the transform variable 's' or 'z'.

Part B: Block diagram reduction

Block diagram is a pictorial representation of a control system showing inter-relation between the transfer function of various components. The block diagram is obtained after obtaining the differential equation and transfer function of all components of a control system. Fig. 3.5 shows an element of the block diagram. The arrowhead pointing towards the block indicates the input $R(s)$ and the one pointing away from the block indicates the output $C(s)$. The term $G(s)$ is the transfer function, i.e.

$$\frac{C(s)}{R(s)} = G(s)$$

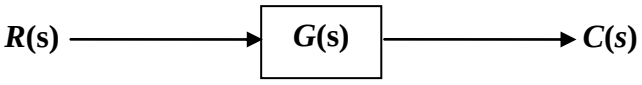


Fig. 3.5

After obtaining the block diagram for each and every components, all blocks are combined to obtain a complete representation. It is then reduced to a simple form with the help of simple block diagram algebra.

The following block diagram reduction algebra is often used:

1. Blocks in Cascade (Series):

Any finite number of blocks in series may be algebraically combined by multiplication.

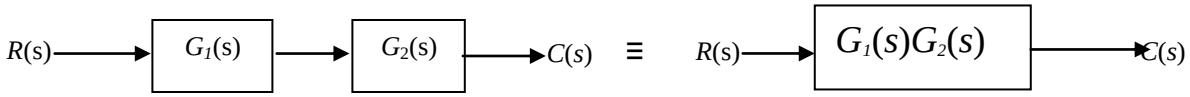


Fig. 3.6 blocks in series

2. Blocks in Parallel (eliminating forward loop):

Any finite number of blocks in parallel may be algebraically combined by Addition or Subtraction

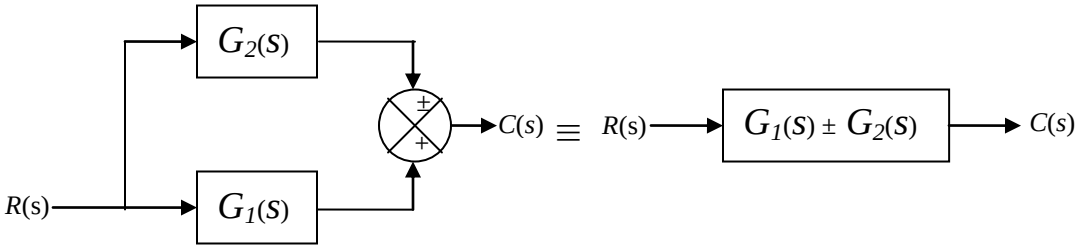


Fig. 3.7 blocks in parallel

3. Eliminating a feedback loop

The standard form of block diagram for a control system with feedback is shown below, where

$C(s) = G(s)\,E(s)$ and $E(s) = R(s) \pm C(s)\,H(s)$

$$\frac{C(s)}{R(s)} = \frac{G(s)}{1 \pm G(s)H(s)}$$

where $\frac{C(s)}{R(s)}$ is the closed–loop transfer function which is also called the ***control ratio***.

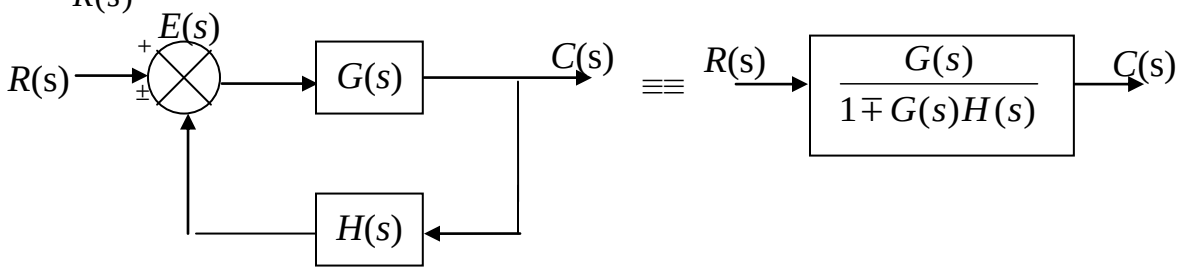


Fig 3.8 feed back

Apparatus:

1. Bread Board
2. Resistors, Capacitors, and Inductors
3. Function generator and oscilloscope
4. Computer and MATLAB program

Procedure:

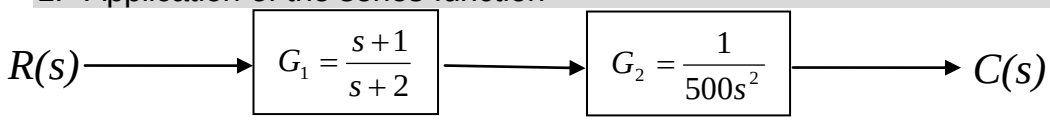
Part A: Electrical First Order System Analysis

1. Connect the circuit shown in fig. (3-1) and apply a square wave input signal of 2 V peak to peak and a frequency of 125 Hz, using a value of R equal to 270Ω and C equal to 1μF.
2. Plot the output response $V_o(t)$ and find the time constant, rise time, settling time, and steady state final value
3. Plot the output response *Vout(t)* on personal computer using MATLAB program and Simulink.

Part B: Block diagram reduction

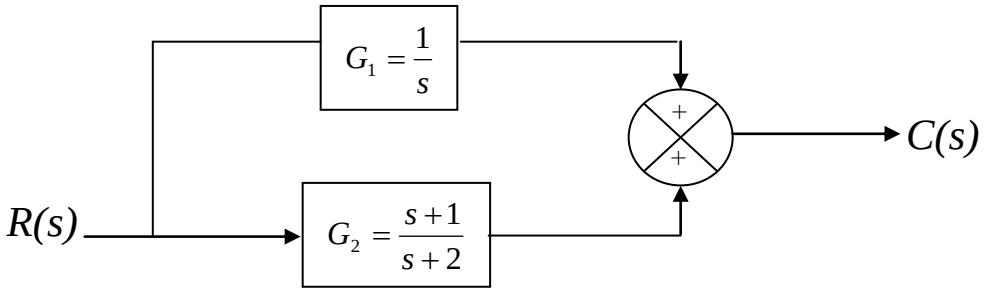
For the block diagrams which are shown below Find the transfer function for the following applications:

1. Application of the series function



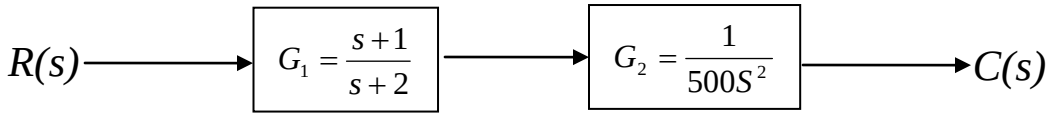
The series function that is used in Matlab program as:
>>[num,den]=series(numg1,deng1,numg2,deng2];

2. Application of the parallel function



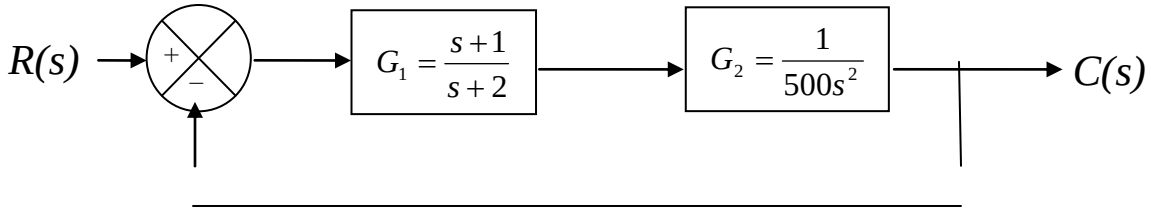
The parallel function that is used in Matlab program as:
>>[num,den]=parallel(numg1,deng1,numg2,deng2);

3. Application of the Conv Function



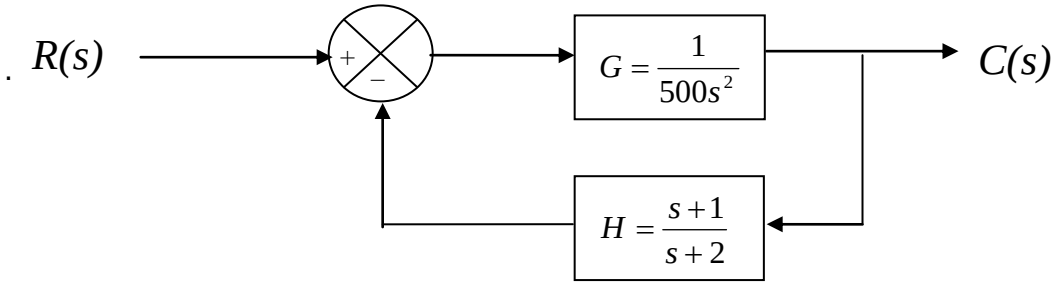
The convolution function that is used in Matlab program as:
num=conv(numg1,numg2);den=conv(deng1,deng2);

4. Application of the Cloop Function



The close loop function that is used in Matlab program as:
[num,den]=cloop(num1,den1,sign)
note +1 , for positive feedback and -1 for negative feedback

5. Application of the Feedback Function



The feed back function that is used in Matlab program as:

>>[num,den]=feedback(numg,deng,numh,denh,sign);

6. Application of the Minreal Function

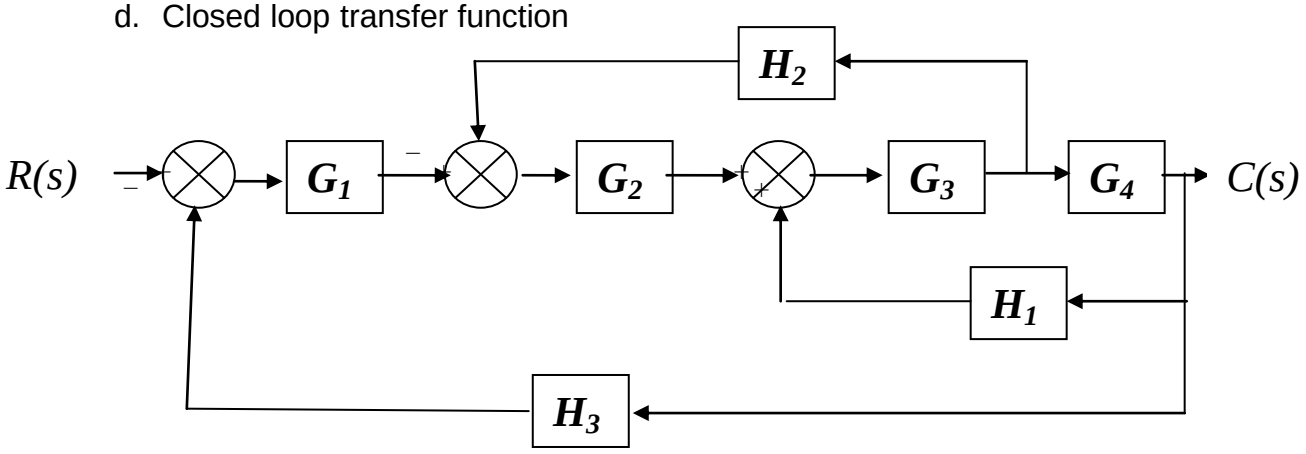
```
          s (s-1)
   ------------------
   (s-1) (s^2 + s + 1)
```

Minreal function is Minimal realization or pole-zero cancellation and it's used in matlab program as:
>>[num,den]=minreal(numg,deng)

Discussion:

1. Derive the transfer function of the electric circuit shown in fig. 3.1
2. Compare the experimental response with simulated response (time constant, rise time, settling time, and steady state final value).
3. Reduce the following block diagram shown below to a standard form using MATLAB program and find:

a. Forward transfer function
b. Feedback transfer function
c. Open loop transfer function
d. Closed loop transfer function



$$G_1 = \frac{1}{s+10} \qquad G_2 = \frac{1}{s+1} \qquad G_3 = \frac{s^2+1}{s^2+4s+4} \qquad G_4 = \frac{s+1}{s+6}$$

$$H_1 = \frac{s+1}{s+2} \qquad H_2 = 2 \qquad H_3 = 1$$

College of Electronics Engineering

Communication Engineering Dept.

Microprocessors & Signals Processing Lab

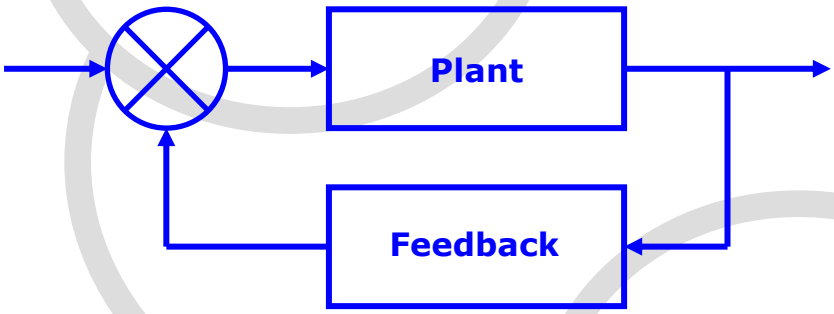
3rd Class

Experiment sheet No. ( )

Title: Closed-loop Speed Control

Feedback

Control-instrumentation



Closed-loop Speed Control

Speed Control System

Objective

- Velocity feedback can be used to enable a speed to be closely regulated. The polarity of the feedback is important.

- The effectiveness of the Control depends mainly on the gain employed.

Closed-Loop Control System

The difference or error signal may be thought of as producing effects which move forward, from the point of comparison to the resulting action. The comparison itself depends on a signal which is fed back from the output of the process to be compared with the reference or input signal. The forward flow and feedback of signals form a loop around which information flows, fig 1.1

Such a system is therefore called a closed-loop system.

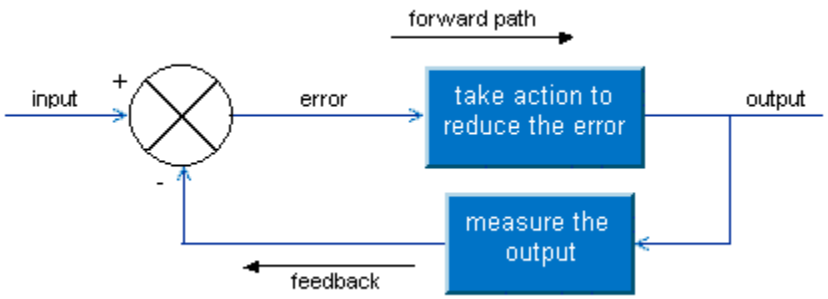


Fig 1.1 - The Closed Control Loop

It is usual for control engineers to describe their systems in a block diagram form. The block diagram Fig (1.2) describes the type of system we shall be using in the assignments. Here there is a comparison by the error channel of the input and output, the error is then amplified to drive a motor and gearing in the forward path so that the speed or position of the output shaft can be modified.

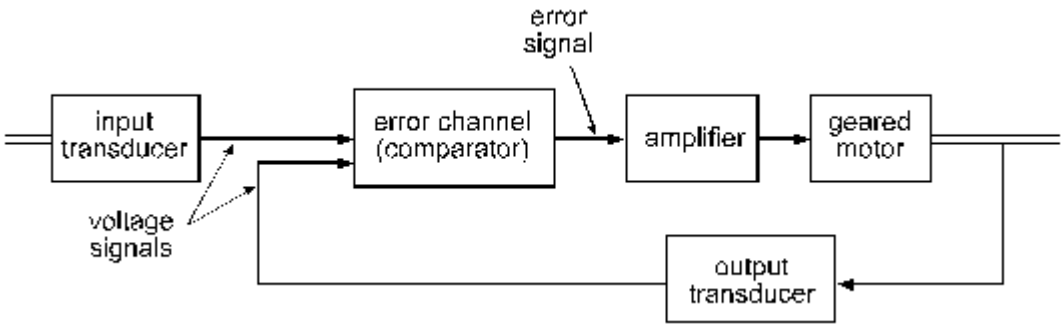


Fig 1.2 - Block Diagram of an Analogue Closed-Loop System.

Speed Control – Introduction

An important aspect of closed-loop control is speed control, which has many industrial applications, varying from heavy industrial, such as paper mills or steel rolling mills, to tape or video transport mechanisms

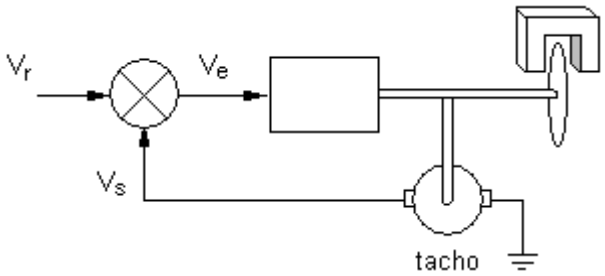


Fig 1.3 - Essential features of a *Closed Loop Speed Control*

The essential principle of closed-loop speed control is that the feedback signal is now an output velocity signal V_s, normally from a tachogenerator, which is compared with a reference voltage V_r to give an error:

$$V_e = V_r - V_s$$

In operation the reference is set to a required value, which drives the motor to generate V_s, which reduces the error until the system reaches a steady speed.

If the motor is loaded, e.g. with the magnetic brake on the 33-100, the speed falls; this tends to increase the error, increasing the motor drive and thus reducing the fall of speed for a given load. Note that this implies negative feedback around the loop.

The speed fall with load, sometimes termed **droop** is a very important characteristic in speed control systems.

The rotation direction can be reversed by reversing the reference voltage, though many industrial speed control systems are required to operate in one direction only.

A speed control system which can be made with the 33-002 corresponding to fig 1.3 is shown below in fig 1.4. Connect the board as shown in the diagram below for this practical.

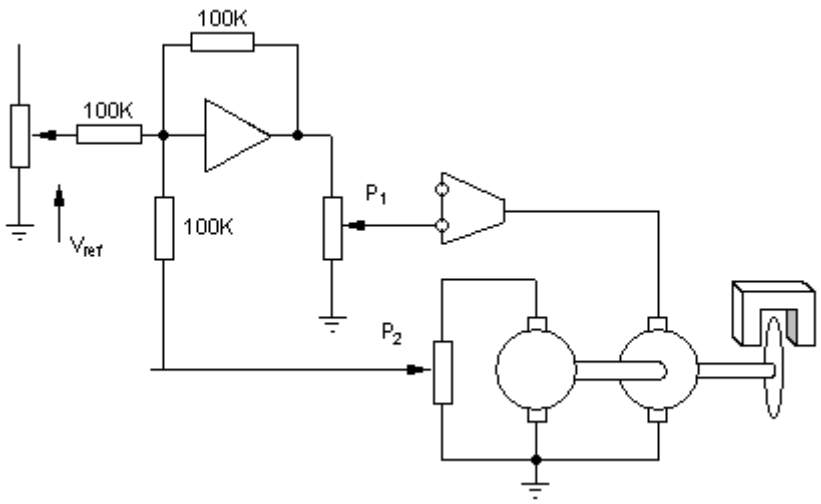


Fig 1.4 - Speed Control System

Motor, Tacho generator and Brake Characteristics

Motor and Tacho generator

The **motor** and **tachogenerator** were used to display the speed response characteristics.

The motor is a **permanent magnet** type and can be represented in idealized form as in fig 1.5 (a), where R_a is the armature resistance and T_1, T_2 are the actual motor terminals.

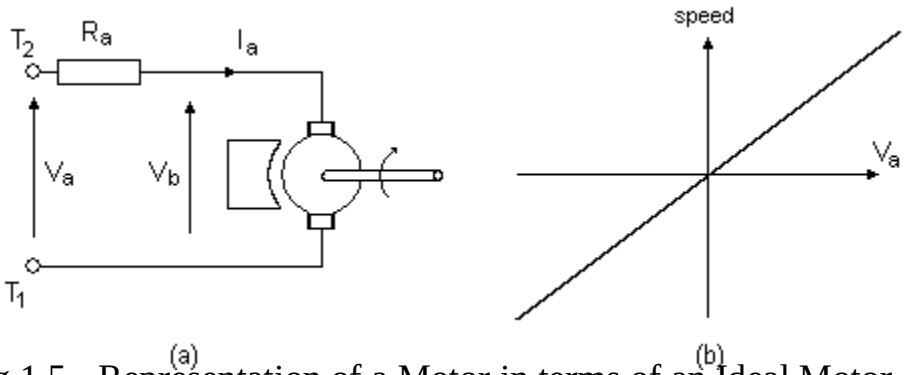


Fig 1.5 - Representation of a Motor in terms of an Ideal Motor.

If the motor is stationary and a voltage V_a is applied, a current I_a flows which causes the motor to rotate. As the motor rotates a back emf V_b is generated. As the motor speeds up the back emf increases and I_a falls.

In the 33-100 the armature voltage V_a is provided by a **power amplifier**. A power amplifier is necessary, because although the voltages in the error channel may be of the same order as V_a, the motor current may be up to *1A*, while the error channel operates with currents of less than 1mA and could not drive the motor directly. The amplifier has two input sockets, enabling the motor rotation direction to be reversed for a given input.

The **tachogenerator** is a small permanent magnet machine and hence when rotated produces an emf proportional to speed which can be used as a measure of the rotation speed.

Brake Characteristics

The **magnetic brake** consists of a permanent magnet which can be swung over an aluminum disc. When the disc is rotated eddy currents circulate in the area of the disc within the magnet gap, and these react with the magnet field to produce a torque which opposes rotation. This gives an adjustable torque speed relation of the form of fig 1.6, and provides a very convenient load for the motor.

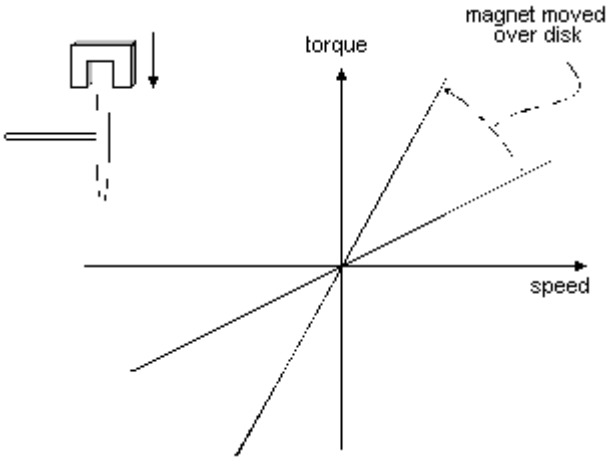


Fig 1.6- Characteristic of Magnetic Brake

The overall characteristics of a motor may be considered from two aspects, both of which can be related to the idealized representation of fig 1.7 (a). These aspects are:

- **Steady-state**, which are concerned with Constant or very slowly changing operating conditions, and
- **Transient**, corresponding with sudden changes.

Both are important in control system applications

Tacho generator

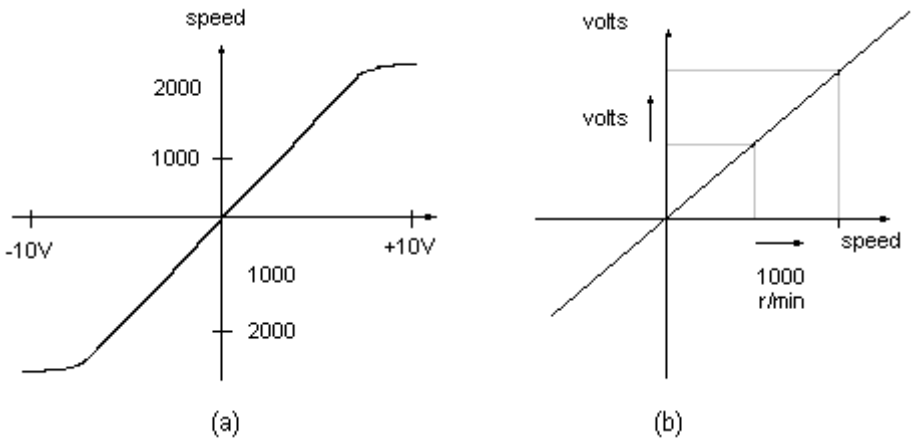


Fig 1.7 - Characteristics of Motor (a) and Tacho generator (b)

The tacho generator provides a voltage proportional to speed, which is required for various aspects of control system operation.
Plot of tacho generator characteristics should be a straight line with the general form of fig 1.7 (b).

An important parameter in the use of tachogenerators is the tacho generator factor in volts per *1000 r/min*.
The factor should be approximately *2.5V* per *1000 r/min*.

Motor speed

To plot the speed against amplifier input, make the scale of the vertical axis in units of *1000 r/min*. The plot should have the general shape of fig **1.7** (a). Initially the motor speed increases substantially linearly with the voltage to the amplifier because the motor back emf V_b, see fig **1.7** (a), approximately equals the amplifier output, but finally the amplifier limits before the full *±10V* input is reached.

Note: Since the reduction to the output shaft is 32:1, the motor speed is calculated by multiplying the r/min reading by 32. e.g. a reading of 31.25 = a motor speed of *1000 r/min*.

Steady-State Characteristics – Brake Load

Considering the idealized motor shown in fig 1.8(a), when the motor is unloaded the back emf V_b substantially equals the applied voltage V_a, the armature current being very small.

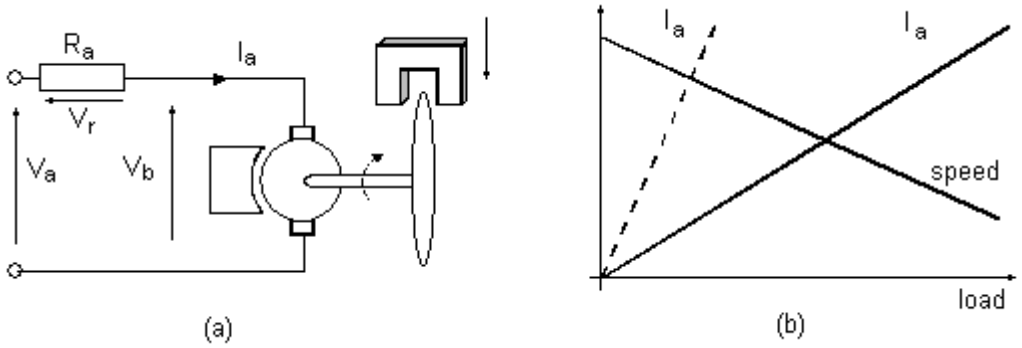


Fig 1.8a and Fig1.8b - Motor Characteristics Related to Load

When the motor is loaded the speed falls, the back emf falls, and the armature current increases and the voltage drop in the armature resistance V_r (= I_aR_a) added to V_b matches V_a, that is:

$$V_a = V_r + V_b$$
$$= I_aR_a + V_b$$

Hence, if the motor is loaded so that the speed falls, the armature current increases, the general characteristic being as the solid lines in fig 1.8(b). If the armature resistance is low, which is the situation for a normal motor, the current increases greatly, as shown dotted, for a small change in speed. The proper operating range of the motor would be up to load corresponding with a few percent drops in speed, perhaps to the point when the dotted current line crosses the speed.

Practical Aspects

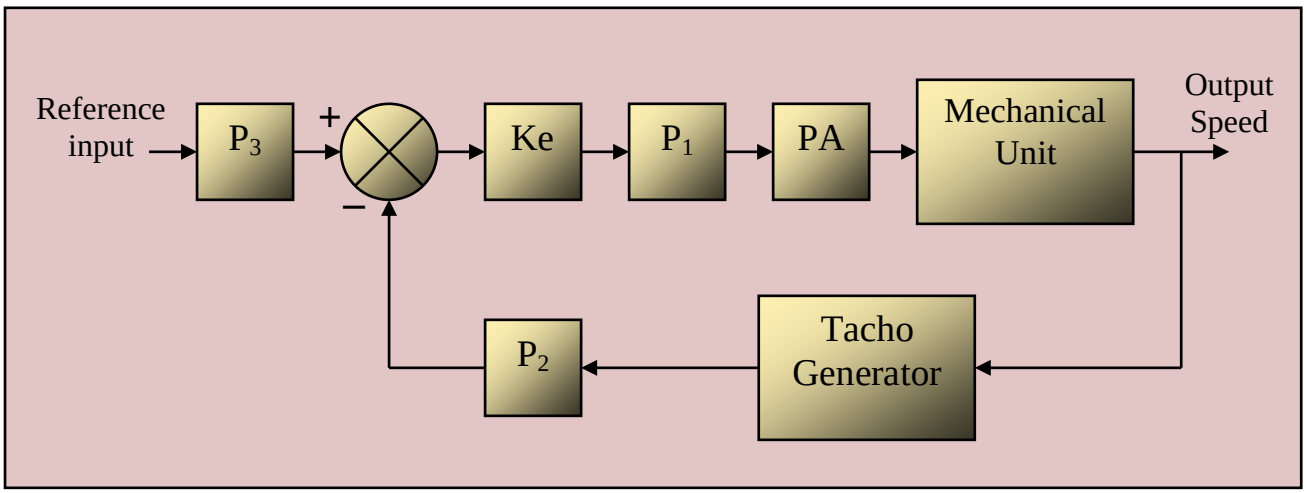
A problem that may arise in speed control systems, where the full feedback signal is obtained from a tachogenerators that the generator output may have an appreciable ripple component due to **commutation**. This component may be amplified in the system and cause saturation. The **ripple** can be reduced by **filtering** in the system or better by specialized design of the generator.

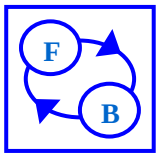
If the armature resistance is low, as for a normal motor the initial armature current may be very large (dangerously so). Thus some starting equipment (a starter) is used to limit the current while the motor is being run up to speed. This applies especially with large motors.

Note 1: The **armature current**, which increases with loading, can be measured by correcting the DVM to the Armature Current socket on the Mechanical Unit.
Note 2: The brake include 6 levels.

Practical step

1-By using **Analogue Unit: 33-110 .**Connect the circuit as shown in the control block diagram as the following:





1-Connect the open loop part from the black diagramed, By Set ***P2*** (tacho) to *zero* and set the amplifier feedback resistor to *100KΩ*, this gives G = 1=Ke. Set ***P1*** to *100*. Set ***SW1*** up to +10 and adjust ***P3*** to run the motor at *1000 r/min* (*31.25 r/min* at output) with no brake.

For different value of brake measure and fill the following table:

Brake	**No brake**	**One level**	**Tow level**	**Three level**	**Four level**	**Fife level**	**Six level Full load**
Speed (rpm)							
armature current (A)							
Error voltage (V)							
Speed droop (rpm)							

how much Reference input so you have 1000 r/min.

2-Repeat the same steps but when Ke=3.3.

3-Repeat the same steps but when Ke=10.

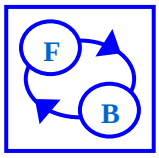
4- Connect the close loop part from the black Diagram , By Set ***P2*** (tacho) to *100* and set the amplifier feedback resistor to *100KΩ*, this gives G = 1 = Ke. Set ***P1*** to *100*. Set ***SW1*** up to +10 and adjust ***P3*** to run the motor at *1000 r/min* (*31.25 r/min* at output) with no brake.

For different value of brake measure and fill the following table:

Brake	**No brake**	**One level**	**Tow level**	**Three level**	**Four level**	**Fife level**	**Six level Full load**
Speed (rpm)							
armature current (A)							
Error voltage (V)							
Speed droop (rpm)							

5-Repeat the same steps but when Ke=3.3.

6-Repeat the same steps but when Ke=10.



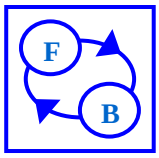
7- The tacho generator output contains a ripple Component so, With $G = 10$, set the speed to 1000 *r/min* on no load and examine the ripple Error amplifier using the oscilloscope. then connect the $0.1\mu F$ capacitor across the Output resistor, then plot the ripple.

8- With $G = 10$, apply a square wave at *0.1Hz* and Adjust P_3 to give a steady speed of ± 1000 *r/min* With no load and remove the $0.1\mu F$ capacitor Across the Output resistor,

a- Plot the input and the step Response of the system.
Then connect the $0.1\mu F$ capacitor across the Output resistor.
b- Plot the input and the step Response of the system.

Conclusion

- This assignment shows the general principle of **speed control**.
- Increasing the gain would give the system less speed fall at full load.
- Theoretically an infinite gain would give zero speed fall, but this is impractical. However, good results can be achieved with a different control technique in the error channel.
- The motor, with no load, runs at a speed almost proportional to the applied voltage.
- The armature current increases with increasing load torque, causing a volt drop in the armature resistance. This effectively reduces the applied voltage, causing a drop in speed.
- The magnetic brake provides a torque proportional to speed and dependent also on the overlap between the magnet and the disc.
- If the applied voltage is suddenly changed, the motor does not respond instantly. Its time constant is defined as the time it would take to reach its final speed if the initial acceleration were maintained.



Question

1- Plot speed, (I_a), (V_e) against brake in one figure. From each tables you filled up in open loop steps (Make 3 plot figure from 3 tables when Ke = 1 , 3.3 and 10).

2- Plot speed, (I_a), (V_e) against brake in one figure. From each tables you filled up in close loop steps (Make 3 plot figure from 3 tables when Ke = 1 , 3.3 and 10).

3- From the step response you plot in step 8-b Find the following :-

A - close loop transfer function of the system.

B - Maximum overshot, Maximum undershot.

C - t_{max} , t_r and t_s.

D - Fine the operating point for open loop system

E - using MATLAB simulink to connect the block diagram of the close loop speed control system and plot step response ,and compare with practical.

4- Explain influence of increasing Gain (Ke) on your Experiment To the response, error signal, speed of Motor and Stability of the system.

5- Practically how you can make the motor running in the opposite direction and the system is still stable (negative feedback) .

6- How you can make check that the system is negatively feedback practically.