NED University of Engineering & TechnologyDepartment of Electrical Engineering

LAB MANUAL

For the course

FEEDBACK CONTRO SYSTEMS (EE-375) For B.E.(EE)

Instructor name: _____

Student name: _____

Roll no: _____ **Batch:** _____

Semester: _____ **Year:** _____

LAB MANUAL

For the course

FEEDBACK CONTROL SYSTEMS(EE-375) For B.E.(EE),

Content Revision Team:

Muhammad Arshad and Muhammad Hassan ul Haq

Last Revision Date:

Approved By

The Board of Studies of Department of Electrical Engineering

_____ _____

_____ _____

_____ _____

Summary of Rubric Sheets for Data Entry in OBE MIS System [to be entered as they are on OBE portal with no scaling]

	SS	Extent of achievement				
		0	1	2	3	4
Lab 2	1					
	2					
	3					
	4					
	5					
	6					
	7					
	8					
Total out of 32						

	SS	Extent of achievement				
		0	1	2	3	4
Lab 4	1					
	2					
	3					
	4					
	5					
	6					
	7					
	8					
Total out of 32						

	SS	Extent of achievement				
		0	1	2	3	4
Lab 6	1					
	2					
	3					
	4					
	5					
	6					
	7					
	8					
Total out of 32						

	SS	Extent of achievement				
		0	1	2	3	4
Lab 7	1					
	2					
	3					
	4					
	5					
	6					
	7					
	8					
Total out of 32						

	SS	Extent of achievement				
		0	1	2	3	4
Lab 8	1					
	2					
	3					
	4					
	5					
	6					
	7					
	8					
Total out of 32						

	SS	Extent of achievement				
		0	1	2	3	4
Lab 9	1					
	2					
	3					
	4					
	5					
	6					
	7					
	8					
Total out of 32						

Add six total scores from above to get total Rubric score Out of 192	R=
Scaled to 25 marks (R/192)*25	S=
Lab Attendance percentage from portal:	A=
Attendance scaled to 5 marks: (A/100)*5	P=
For entry on NED exam portal Total lab sessionals out of 30: S + P	T=

	SS	Extent of achievement				
		0	1	2	3	4
PBL/OEL	1					
	2					
	3					
	4					
	5					
	6					
	7					
	8					
Total out of 32						

	SS	Extent of achievement				
		0	1	2	3	4
Final	1					
	2					
	3					
	4					
	5					
	6					
	7					
	8					
Total out of 32						

Add two total scores from above to get total final Rubric Score Out of 64	Z=
For entry on NED exam portal Total Final Lab Marks scaled to 20 marks (Z/64)*20	F=

CONTENTS

Psychomotor Level 3				
S.No.	**Date**	**Title of Experiment**	**Total Marks**	**Signature**
1		Introduction to MATLAB: Introduction to polynomials, script writing and programming aspect of MATLAB from control systems point of view, mathematical models of physical systems.		
2		*****Mathematical modeling of Mechanical Translation and Electrical Systems		
3		Developing linear model for DC motor, performance analysis of first order and second order systems and development of time response specifications function		
4		*****Study the three term PID controller and its effects on the feedback loop response, investigating the characteristics of the each of proportional (P), the integral (I), and the derivative (D) controls and obtaining a desired response by using them		
5		Introduction to Programmable Logic Controllers (PLCs), their use and applications in industry, method for configuring and programming PLCs using ladder language		
6		*****Digital I/O interfacing and manipulation in PLCs, their application in designing On-Off type feedback control systems using ladder language		
7		*****Digital I/O manipulation in PLCs with timers, counters, and PWM (Pulse Width Modulation) generators for designing On-Off type feedback control systems using ladder language		
8		*****Analog I/O interfacing and manipulation in PLCs, their application in sensing transducer outputs and transmitting signals to actuators		
9		*****Introduction to HMI (Human Machine Interface), its programming via PLC and communication set-up between PLC and HMI for measurement viasualisation		
10		DC motor speed measurement and control via PLC utilising analogue I/O, digital I/O, PWM generator HMI and other PLC peripherals		
11		Open Ended Lab - To simulate and design hardware of a feedback control system using Buck converter as plant		

Lab titles marked with an asterisk (*) are assessed through OBE Lab Rubrics for Feedback Control Systems

LAB SESSION 01 (Part-1)

Objective:

Introduction to MATLAB briefly including tutorial of polynomials, script writing and programming aspect of MATLAB from control systems view point.

THEORY:

MATLAB Stands for **MATrix LABoratory**.

MATLAB is a computer program that combines computation and visualization power that makes it particularly useful tool for engineers. It is an executive program, and a script can be made with a list of MATLAB commands like other programming language. The windows in MATLAB are:

- Command window: Commands can be entered, data and results are displayed
- Workspace: list all the variables you are using
- command history window: it displays a log of the command used.
- Graphic (Figure) Window: Displays plots and graphs, created in response to graphics commands.
- M-file editor/debugger window: Create and edit scripts of commands called M-files.

Variable declaration:

The variables are declared as:

Must start with a letter

May contain only letters, digits, and the underscore "_"

Matlab is case sensitive, i.e. one & ONE are different variables. For assigning statement:

Variable = number;

Special variables:

ans : default variable name for the result

pi: $\pi = 3.1415926$

NaN or nan: not-a-number

Commands involving variables:

who: lists the names of defined variables

whos: lists the names and sizes of defined variables

clear: clears all variables, reset the default values of special variables.

clear name: clears the variable name

clc: clears the command window

clf: clears the current figure and the graph window

Matrix Array

A Matrix array is two-dimensional, having both multiple rows and multiple columns, similar to vector arrays:

- It begins with [and ends with]
- spaces or commas are used to separate elements in a row.
- semicolon or enter is used to separate rows.

Example:

```
>> f = [ 1 2 3; 4 5 6]

f =

1     2     3

4     5     6
```

Transpose	B = A′
Identity Matrix	eye(n) ➔ returns an n x n identity matrix eye(m,n) ➔ returns an m x n matrix with ones on the main diagonal and zeros elsewhere.
Addition and subtraction	C = A + B C = A – B
Scalar Multiplication	B = αA, where α is a scalar.
Matrix Multiplication	C = A*B
Matrix Inverse	B = inv(A), A must be a square matrix in this case. rank (A) ➔ returns the rank of the matrix A.
Matrix Powers	B = A.^2 ➔ squares each element in the matrix C = A * A ➔ computes A*A, and A must be a square matrix.
Determinant	det (A), and A must be a square matrix.

A, B, C are matrices, and m, n, α are scalars.

A system of 3 linear equations with 3 unknowns (x1, x2, x3):

3x1+2x2-x3=10

-x1+3x2+2x3=5

x1-x2-x3= -1

Let

$$A=\begin{bmatrix}3 & 2 & 1\\ -1 & 3 & 2\\ 1 & -1 & -1\end{bmatrix} \qquad x=\begin{bmatrix}x_1\\ x_2\\ x_3\end{bmatrix} \qquad b=\begin{bmatrix}10\\ 5\\ -1\end{bmatrix}$$

- <u>Solution by Matrix Inverse:</u>

  $Ax = b$

  $A^{-1}Ax = A^{-1}b$

  $x = A^{-1}b$

- MATLAB:

```
>> A = [ 3 2 -1; -1 3 2; 1 -1 -1];
>> b = [ 10; 5; -1];
>> x = inv(A)*b
x =
   -2.0000
    5.0000
   -6.0000
```

<u>Answer</u>:
$x_1 = -2,\ x_2 = 5,\ x_3 = -6$

- <u>Solution by Matrix Division:</u>

  The solution to the equation

  $Ax = b$

  can be computed using left division

- MATLAB:

```
>> A = [ 3 2 -1; -1 3 2; 1 -1 -1];
>> b = [ 10; 5; -1];
>> x = A\b
x =
   -2.0000
    5.0000
   -6.0000
```

<u>Answer</u>:
$x_1 = -2,\ x_2 = 5,\ x_3 = -6$

Some useful commands

command	description
axis ([xmin xmax ymin ymax])	Define minimum and maximum values of the axes
axis square	Produce a square plot
axis equal	equal scaling factors for both axes
axis normal	turn off axis square, equal
axis (auto)	return the axis to defaults

Plotting Curves:

plot (x,y) – generates a linear plot of the values of x (horizontal axis) and y (vertical axis).

semilogx (x,y) – generate a plot of the values of x and y using a logarithmic scale for x and a linear scale for y

semilogy (x,y) – generate a plot of the values of x and y using a linear scale for x and a logarithmic scale for y.

loglog(x,y) – generate a plot of the values of x and y using logarithmic scales for both x and y

Example: (polynomial function)

Plot the polynomial using linear/linear scale, log/linear scale, linear/log scale, & log/log2 scale:

$y = 2x^2 + 7x + 9$

```
% Generate the polynomial:
x = linspace (0, 10, 100);
y = 2*x.^2 + 7*x + 9;

% plotting the polynomial:
figure (1);
subplot (2,2,1), plot (x,y);
title ('Polynomial, linear/linear scale');
ylabel ('y'), grid;
subplot (2,2,2), semilogx (x,y);
title ('Polynomial, log/linear scale');
ylabel ('y'), grid;
subplot (2,2,3), semilogy (x,y);
title ('Polynomial, linear/log scale');
xlabel('x'), ylabel ('y'), grid;
subplot (2,2,4), loglog (x,y);
title ('Polynomial, log/log scale');
xlabel('x'), ylabel ('y'), grid;
```

Adding new curves to the existing graph:

Use the hold command to add lines/points to an existing plot.

hold on – retain existing axes, add new curves to current axes. Axes are rescaled when necessary.

hold off – release the current figure window for new plots

Grids and Labels:

Command	Description
grid on	Adds dashed grids lines at the tick marks
grid off	removes grid lines (default)
grid	toggles grid status (off to on, or on to off)
title ('text')	labels top of plot with text in quotes
xlabel ('text')	labels horizontal (x) axis with text is quotes
ylabel ('text')	labels vertical (y) axis with text is quotes
text (x,y,'text')	Adds text in quotes to location (x,y) on the current axes, where (x,y) is in units from the current plot.

Additional commands for Plotting

Color of the point or curve

Symbol	Color
y	yellow
m	magenta
c	cyan
r	red
g	green
b	blue
w	white
k	black

Marker of the data point

Symbol	Marker
.	•
o	∘
x	×
+	+
*	*
s	□
d	◊
v	▽
^	△
h	hexagram

Plot line styles

Symbol	Line Style
–	solid line
:	dotted line
–.	dash-dot line
– –	dashed line

Polynomial evaluation:

Function	Description
Conv	Multiply polynomials
Deconv	Divide polynomials
Poly	Polynomial with specified roots
Polyder	Polynomial derivative
Polyfit	Polynomial curve fitting
Polyval	Polynomial evaluation
Polyvalm	Matrix polynomial evaluation
Residue	Partial-fraction expansion (residues)
Roots	Find polynomial roots

Polynomial Roots

The roots function calculates the roots of a polynomial:

```
>>p = [1 0 -2 -5];
```

r =2.0946

-1.0473 + 1.1359i -1.0473 - 1.1359i

Polynomial Evaluation

The polyval function evaluates a polynomial at a specified value. To evaluate p at s = 5, use

```
>>polyval(p,5)

ans = 110
```

To evaluate the polynomial p at X:

```
>>X = [2 4 5; -1 0 3; 7 1 5];

>>Y = polyvalm(p,X)

Y =

377 179 439

111 81 136

490 253 639
```

Convolution and Deconvolution

Polynomial multiplication and division correspond to the operations convolution and deconvolution. The functions conv and deconv implement these operations. Consider the

```
>>a = [1 2 3]; b = [4 5 6];

 >>c = conv(a,b)

c =4 13 28 27 18
```

Use deconvolution to divide back out of the product:

```
>>[q,r] = deconv(c,a)

q =4 5 6

r =0 0 0 0 0
```

Polynomial Derivatives

The polyder function computes the derivative of any polynomial. To obtain the derivative of the polynomial

```
>>p= [1 0 -2 -5]

 >>q = polyder(p)

q =3  0 -2
```

polyder also computes the derivative of the product or quotient of two polynomials. For example, create two polynomials a and b:

```
>>a = [1 3 5];

>>b = [2 4 6];
```

Calculate the derivative of the product a*b by calling polyder with a single output argument:

```
>>c = polyder(a,b)

c =

8 30 56 38
```

Calculate the derivative of the quotient a/b by calling polyder with two output arguments:

```
>>[q,d] = polyder(a,b)

q =

-2 -8 -2

d =

4 16 40 48 36
```

q/d is the result of the operation.

Partial Fraction Expansion

'residue' finds the partial fraction expansion of the ratio of two polynomials. This is particularly useful for applications that represent systems in transfer function form. If there are no multiple roots, where r is a column vector of residues, p is a column vector of pole locations, and k is a row vector of direct terms.

Consider the transfer function >>b = [-4 8];

```
>>a = [1 6 8];

>>[r,p,k] = residue(b,a)

r = -12 8

p = -4 -2

k = []
```

Given three input arguments (r, p, and k), residue converts back to polynomial form:

```
>>[b2,a2] = residue(r,p,k)

b2 =-4 8

a2 =1 6 8
```

Scripts and Functions

MATLAB is a powerful programming language as well as an interactive computational environment. Files that contain code in the MATLAB language are called M-files. You create M-files using a text editor, then use them as you would any other MATLAB function or command. There are two kinds of M-files:

Scripts, which do not accept input arguments or return output arguments. They operate on data in the workspace. MATLAB provides a full programming language that enables you to write a series of MATLAB statements into a file and then execute them with a single command. You write your program in an ordinary text file, giving the file a name of 'filename.m'. The term you use for 'filename' becomes the new command that MATLAB associates with the program. The file extension of .m makes this a MATLAB M-file.

Functions, which can accept input arguments and return output arguments. Internal variables are local to the function.

If you're a new MATLAB programmer, just create the M-files that you want to try out in the current directory. As you develop more of your own M-files, you will want to organize them into other directories and personal toolboxes that you can add to your MATLAB search path. If you duplicate function names, MATLAB executes the one that occurs first in the search path.

Scripts:

When you invoke a script, MATLAB simply executes the commands found in the file. Scripts can operate on existing data in the workspace, or they can create new data on which to operate. Although scripts do not return output arguments, any variables that they create remain in the workspace, to be used in subsequent computations. In addition, scripts can produce graphical output using functions like plot. For example, create a file called 'myprogram.m' that contains these MATLAB commands:

% Create random numbers and plot these numbers clc clear

r = rand(1,50);

plot(r)

Typing the statement 'myprogram' at command prompt causes MATLAB to execute the commands, creating fifty random numbers and plots the result in a new window. After execution of the file is complete, the variable 'r' remains in the workspace.

Functions:

Functions are M-files that can accept input arguments and return output arguments. The names of the M-file and of the function should be the same. Functions operate on variables within their own workspace, separate from the workspace you access at the MATLAB command prompt.

M-File Element	Description
Function definition line (functions only)	Defines the function name, and the number and order of input and output arguments.
H1 line	A one line summary description of the program, displayed when you request help on an entire directory, or when you use 'lookfor'.
Help text	A more detailed description of the program, displayed together with the H1 line when you request help on a specific function
Function or script body	Program code that performs the actual computations and assigns values to any output arguments.
Comments	Text in the body of the program that explains the internal workings of the program.

The first line of a function M-file starts with the keyword 'function'. It gives the function name and order of arguments. In this case, there is one input arguments and one output argument. The next several lines, up to the first blank or executable line, are comment lines that provide the help text. These lines are printed when you type 'help fact'. The first line of the help text is the H1 line, which MATLAB displays when you use the 'lookfor' command or request help on a directory. The rest of the file is the executable MATLAB code defining the function.

The variable n & f introduced in the body of the function as well as the variables on the first line are all local to the function; they are separate from any variables in the MATLAB workspace. This example illustrates one aspect of MATLAB functions that is not ordinarily found in other programming languages—a variable number of arguments. Many M-files work this way. If no output argument is supplied, the result is stored in ans. If the second input argument is not supplied, the function computes a default value.

Flow Control:

Conditional Control – if, else, switch

This section covers those MATLAB functions that provide conditional program control. if, else, and elseif. The if statement evaluates a logical expression and executes a group of statements when the expression is true. The optional elseif and else keywords provide for the execution of alternate groups of statements. An end keyword, which matches the if, terminates the last group of statements.

The groups of statements are delineated by the four keywords—no braces or brackets are involved as given below:

if <condition> <statements>;

elseif <condition> <statements>;

else

<statements>;

end

It is important to understand how relational operators and if statements work with matrices. When you want to check for equality between two variables, you might use if A == B.

This is valid MATLAB code, and does what you expect when A and B are scalars. But when A and B are matrices, A == B does not test if they are equal, it tests where they are equal; the result is another matrix of 0's and 1's showing element-by-element equality. (In fact, if A and B are not the same size, then A == B is an error.)

if isequal(A, B),

isequal returns a scalar logical value of 1 (representing true) or 0 (false), instead of a matrix, as the expression to be evaluated by the if function.

Using the A and B matrices from above, you get

>>isequal(A, B) ans =0.

Mn Here is another example to emphasize this point. If A and B are scalars, the following program will never reach the "unexpected situation". But for most pairs of matrices, including

if A > B 'greater' elseif A < B 'less' elseif A == B 'equal' else

error('Unexpected situation') end our magic squares with interchanged columns, none of the matrix conditions A > B, A < B, or A == B is true for all elements and so the else clause is executed:

Several functions are helpful for reducing the results of matrix comparisons to scalar conditions for use with if, including ‘isequal’, ‘isempty’, ‘all’, ‘any’.

Switch and Case:

The switch statement executes groups of statements based on the value of a variable or expression. The keywords case and otherwise delineate the groups. Only the first matching case is executed. The syntax is as follows:

switch <condition or expression>

case <condition>

<statements>;

…

case <condition>

<statements>;

…

otherwise

<statements>;

end

There must always be an end to match the switch. An example is shown below.

n=5

switch rem(n,2) % to find remainder of any number ‘n’

```
case 0
disp('Even Number') % if remainder is zero
case 1
disp('Odd Number') % if remainder is one
end
```

Lab exercise:

Exercise: 1

Consider the two polynomials $p(s)=s^2+2s+1$ and $q(s)=s+1$.

Use MATLAB to compute

a. p(s)*q(s)

b. Roots of p(s) and q(s)

c. p(-1) and q(6)

Exercise 2:

Use MATLAB command to find the partial fraction of the following

$$\frac{B(s)}{A(s)} = \frac{2s^3+5s^2+3s+6}{s^3+6s^2+11s+6}$$

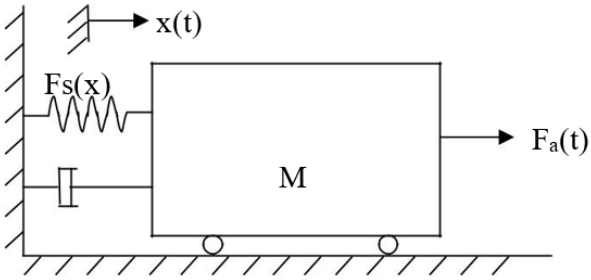
LAB SESSION 01 (Part-2)

Objective: Mathematical models of physical systems in the design and analysis of control systems.

Theory

Mass-Spring System Model

Consider the following Mass-Spring system shown in the figure. Where Fs(x) is the spring force, Fa(t) applied force:



$a=dv(t)/dt = d^2x(t)/dt^2$ is the acceleration dx(t) is the displacement

According to the laws of physics:

Ma + Ff(v) + Fs(x) = Fa(t)

The differential equation for the above Mass-Spring system can then be written as follows

$$M(d^2x(t)/dt^2) + B(dx(t)/dt) + Kx(t) = Fa(t) \text{ ----------(1)}$$

Where,

- B is called the friction coefficient and
- K is called the spring constant.

The linear differential equation of second order (2) describes the relationship between the displacement and the applied force. The differential equation can then be used to study the time behavior of x(t) under various changes of the applied force. In reality, the spring force and/or the friction force can have a more complicated expression or could be represented by a graph or data table. For instance, a nonlinear spring can be designed (see figure 2.2) such that r > 1.

Solving the differential equation using MATLAB:

The objectives behind modeling the mass-damper system can be many and may include

- Understanding the dynamics of such system
- Studying the effect of each parameter on the system such as mass M, the friction coefficient B, and the elastic characteristic Fs(x).
- Designing a new component such as damper or spring.

- Reproducing a problem in order to suggest a solution.

MATLAB can help solve linear or nonlinear ordinary differential equations (ODE). To show how you can solve ODE using MATLAB we will proceed in two steps. We first see how can we solve first order ODE and second order ODE.

PROCEDURE:

Speed Cruise Control example:

When Fs(x)=0, which means that K=0, Equation (1) becomes

$M(d^2x(t)/dt^2) + B(dx(t)/dt) = Fa(t)$

Or,

$M(dv(t)/dt) + Bv(t) = Fa(t)$

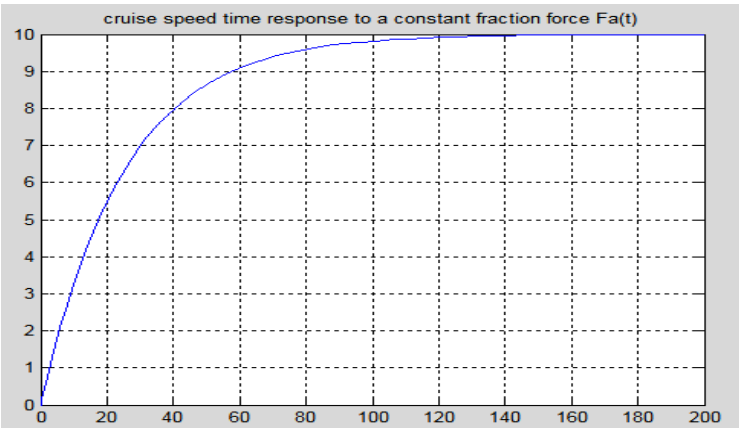
Using MATLAB solver ode45 we can write do the following:

1) Create a MATLAB-function cruise_speed.m

```
function dvdt=cruise_speed(t, v)

%flow rate M=750; %(Kg)

B=30; %( Nsec/m) Fa=300; %N

% dv/dt=Fa/M-B/M v dvdt=Fa/M-B/M*v;
```

2) Create a new MATLAB m-file and write

```
v0= 0; %(initial speed)

[t,v]=ode45('cruise_speed', [0 200],v0);

plot(t,v);

grid on;

title('cruise speed time response to a constant traction force Fa(t) ')
```



In the above program the behavior of a car speed is shown in which the car starts with rest position, after that it attains its maximum speed so that it reaches its maximum limit then after that its speed becomes constant throughout the time.

Mass-Spring System Example:

$M(d^2(t)/dt^2) + B(dx(t)/dt) + Kx(t) = Fa(t)$

Variables	New variable	Differential equation
x(t)	X_1	$\frac{dX_1}{dt} = X_2$
dx(t)/dt	X_2	$\frac{dX_2}{dt} = -\frac{B}{M}X_2 - \frac{K}{M}X_1{}^r(t) + \frac{F_a(t)}{M}$

In vector form

$$X = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} \qquad \frac{dX}{dt} = \begin{bmatrix} \frac{dX_1}{dt} \\ \frac{dX_2}{dt} \end{bmatrix}$$

The system equations can be written as:

$$\frac{dX}{dt} = -\frac{B}{M}X_2 - \frac{K}{M}X_1{}^r(t) + \frac{F_a(t)}{M}$$

1) create a MATLAB-function mass_spring.m function dXdt=mass_spring(t,X)

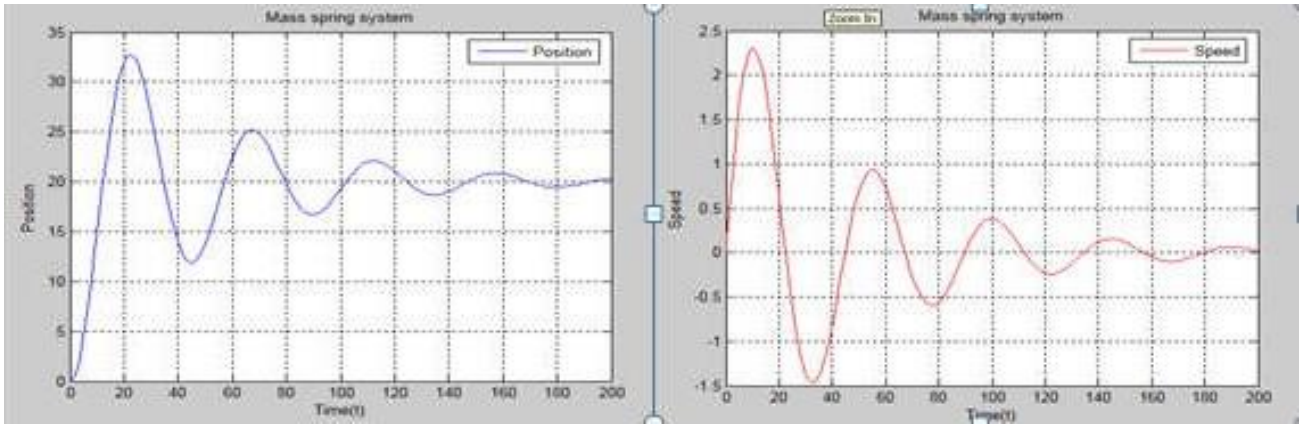
```
M=705;% (Kg)
B=30;   % (Nsec/m)
Fa=300; % (N)
K=15;   % (N/m)
r=1;      % dX/dt
dXdt(1,1)=X(2);
dXdt(2,1)=-B/M*X(2)-K/M*X(1)^r+Fa/M;
```

2) Program of mass spring system with r=1

```
clear all
close all
clc
X0=[0;0];% (Initial speed and position)
[t,X]=ode45('mass_spring',[0 200],X0);
figure;
plot(t,X(:,1));
xlabel('Time(t)'); ylabel('Position'); title('Mass spring system'); legend('Position ');
grid;
figure;
plot(t,X(:,2),'r');
xlabel('Time(t)');
label('Speed');
```

title('Mass spring system'); legend('Speed ');

grid;



OBSERVATIONS:

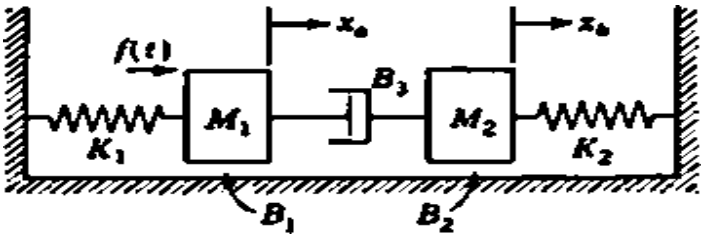
Parameter	Behavior of system
Mass	
Friction Coefficient	
Stiffness	
Applied Force	

CONCLUSION:

LAB SESSION 02

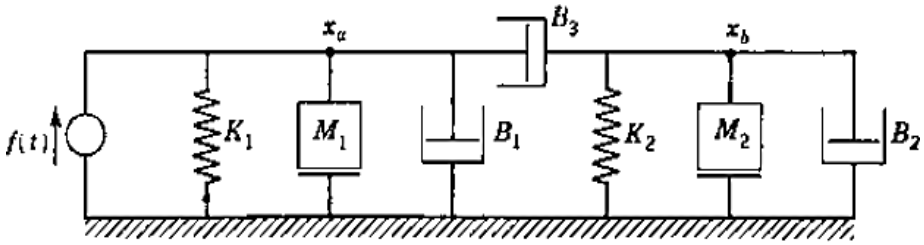
Objective: Mathematical modeling of Multiple-Element Mechanical Translation and Electrical Systems

Theory:



Consider the given multiple-element spring-mass-damper system, where;

- f(t) is applied force to the mass M1.
- B1 and B2 are the viscous friction coefficients indicating the sliding friction between the masses M1 and M2 and the surface.



According to the rules for node equations:

For node a:

(M1D2 + B1D + B3D + K1)xa – (B3D)xb = f

For node b:

-(B3D)xa + (M2D2 + B2D + B3D + K2)xb = 0

X1=Xb for spring K2 X2=X´1=Vb

X3=Xa for spring K1 X4=X´3=Va

The system equations are:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -\frac{K_2}{M_2} & -\frac{B_2+B_3}{M_2} & 0 & \frac{B_3}{M_2} \\ 0 & 0 & 0 & 1 \\ 0 & \frac{B_3}{M_1} & -\frac{K_1}{M_1} & -\frac{B_1+B_3}{M_1} \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{1}{M_1} \end{bmatrix}$$

PROCEDURE:

1) Create a MATLAB-function multiple_element_sys.m

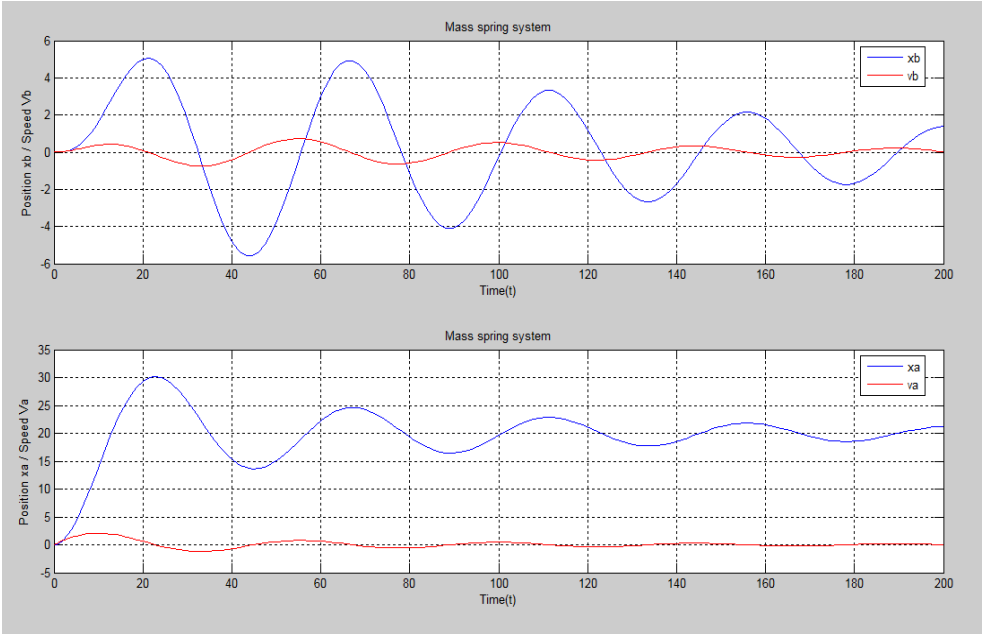
function dXdt=multiple_element_sys (t,X)

```
Fa=300;          %(N)
M1=750;          %(Kg)
M2=750;          %(Kg)
B1=20; %(Nsec/m)
B2=20; %(Nsec/m)
B3=30; %(Nsec/m)
K1=15; %(N/m)
K2=15; %(N/m)
dXdt(1,1)=X(2);
dXdt(2,1)=-K2/M2*X(1)-((B1+B2)/M2)*X(2)+B3*X(4)/M2;
dXdt(3,1)=X(4);
dXdt(4,1)=B3/M1*X(2)-K1/M1*X(3)-((B1+B3)/M1)*X(4)+Fa/M1;
```

2) Write another m file to call the function:

```
clear all;
close all;
clc;
X0=[0;0;0;0]; % (Initial xb, Vb, xa, Va)
[t,X]=ode45('multiple_element_sys',[0 200],X0);
figure;
subplot(2,1,1);
plot(t,X(:,1));
plot(t,X(:,2),'r');
xlabel('Time(t)');
ylabel('Position xb / Speed Vb');
title('Mass spring system'); legend('xb', 'Vb');
grid;
subplot(2,1,2);
plot(t,X(:,3));
hold;
plot(t,X(:,4),'r');
xlabel('Time(t)');
ylabel('Position xa / Speed Va');
title('Mass spring system');
legend('xa', 'Va');
```

grid;



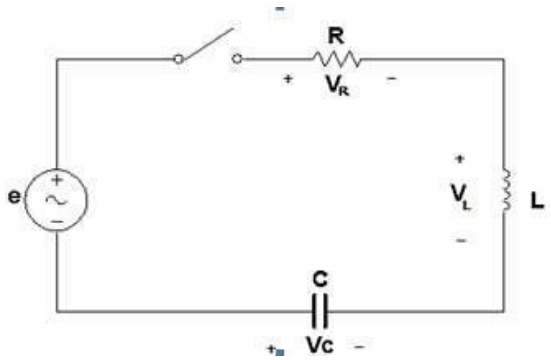
OBSERVATIONS:

Parameter		**Behavior of system**
Mass	**M1**	
	M2	
Friction Coefficient	**B1**	
	B2	
	B3	
Stifness	**K1**	
	K2	
Applied Force	**F_a**	

CONCLUSION:

Mathematical modeling of Electrical System

Theory:



Consider the 2nd order circuit shown in the above diagram.

- e is applied Potential.
- i is the mesh current.

The differential equations for the given figure.

According to Mesh Analysis:

$e(t)=V_L+V_C+V_R$

e(t)=LD i + (1/CD) + R i

The state equations for the given figure.

This circuit contains two energy-storage elements, Inductor and capacitor.

Let state variables are

X1= Vc the voltage across the capacitor, and

X2= i the current in the inductor.

STATE EQUATION:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{C} \\ -\frac{1}{L} & -\frac{R}{L} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{L} \end{bmatrix} [u]$$

PROCEDURE:

1) create a MATLAB-function RLC.m

```
function dXdt=RLC(t,X)

e=60;    % (V)

R=10;    % (Ohm)

L=1;     % (H)

C=10;    % (F)

%dX/dt dXdt(1,1)=(1/C)*X(2);

dXdt(2,1)=(-1/L)*X(1)-(R/L)*X(2)+(1/L)*e;
```

2) Write an other M. file to call the function:

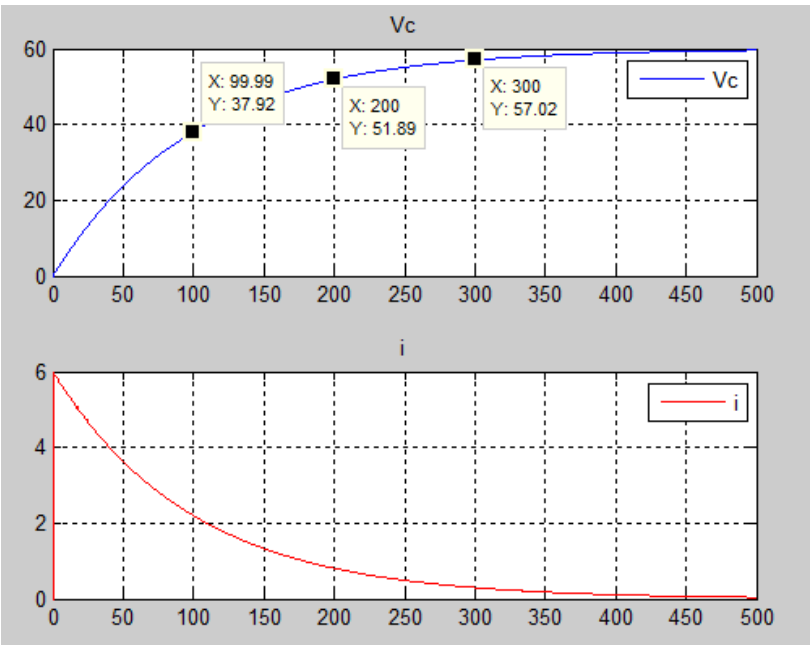
```
clear all
close all
clc
X0=[0 0];
[t,X] =ode45('RLC',[ 0 500],X0);
subplot(2,1,1);
plot(t,X(:,1));
legend('Vc');
grid on;
title('Vc');
subplot(2,1,2);
plot(t,X(:,2),'r');
legend('i');
grid on;
title('i');
```

Graph:

- Time constant = RC = 10*10= 100 sec

For first time constant:

- Vc=63.2% * e = 0.632*60 =37.92 V



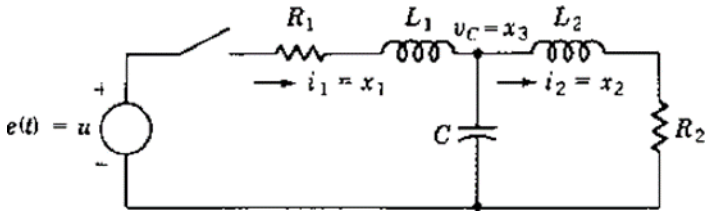
OBSERVATIONS:

Parameter	**Behavior of system**
Voltage source(e)	
Resistance(R)	
Inductance(L)	
Capacitance(C)	

CONCLUSION:

EXERCISE:

Write the function and program of the following circuit diagram. Also explain the plots of the respective state variables.



NED University of Engineering & Technology
Department of _____ Engineering

Psychomotor Domain Assessment Rubric for Laboratory (Level P3)					
Skill(s) to be assessed	Extent of Achievement				
	0	1	2	3	4
Software Menu Identification and Usage: Ability to initialise, configure and *operate* software environment *under supervision*, using menus, shortcuts, instructions etc.	Unable to understand and use software menu	Little ability and understanding of software menu operation, makes many mistake	Moderate ability and understanding of software menu operation, makes lesser mistakes	Reasonable understanding of software menu operation, makes no major mistakes	Demonstrates command over software menu usage with frequent use of advance menu options
Procedural Programming of given model: *Practice* procedural programming techniques, in order to code specific model	Little to no understanding of procedural programming techniques	Slight ability to use procedural programming techniques for coding given algorithm	Mostly correct recognition and application of procedural programming techniques but makes crucial errors for the given model	Correctly recognises and uses procedural programming techniques with no errors but unable to run model successfully	Correctly recognises and uses procedural programming techniques with no errors and runs model successfully
Relating Theoretical Concepts, Equations and Transforms to Code: *Recognise* relation between model concepts and written code and *manipulate* the code in accordance of requirements	Completely unable to relate between model concepts and written code, unable to do manipulations	Able to recognise some relation between model concepts and written code, unable to do manipulations	Able to recognise relation between model concepts and written code, unable to do manipulations	Able to recognise relation between model concepts and written code, able to do some manipulations	Able to recognise relation between model concepts and written code, able to completely manipulate code in line with theoretical concepts
Detecting and Removing Errors: *Detect* Errors/Exceptions and in simulation and *manipulate* code to rectify the simulation	Unable to check and detect error messages and indications in software	Able to find error messages and indications in software but no understanding of detecting those errors and their types	Able to find error messages and indications in software as well as understanding of detecting some of those errors and their types	Able to find error messages in software as well as understanding of detecting all of those errors and their types	Able to find error messages in software along with the understanding to detect and rectify them

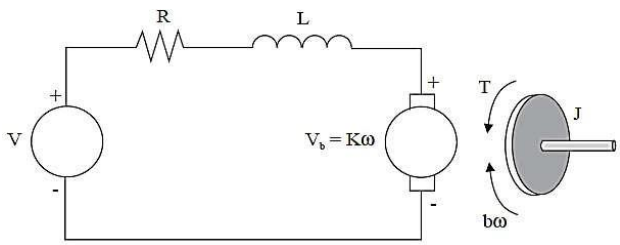
Graphical Visualisation and Comparison of model **Parameters:** *Manipulate* given simulation under supervision, in order to produce graphs/plots for measuring and comparing model parameters	Unable to understand and utilise visualisation or plotting features	Ability to understand and utilise visualisation and plotting features with frequent errors	Ability to understand and utilise visualisation and plotting features successfully but unable to compare and analyse them	Ability to understand and utilise visualisation and plotting features successfully, partially able to compare and analyse them	Ability to understand and utilise visualisation and plotting features successfully, also able to compare and analyse them
Following step-by-step procedure to complete lab work: *Observe, imitate and operate* software to complete the provided sequence of steps	Inability to recognise and perform given lab procedures	Able to recognise given lab procedures and perform them but could not follow the prescribed order of steps	Able to recognise given lab procedures and perform them by following prescribed order of steps, with frequent mistakes	Able to recognise given lab procedures and perform them by following prescribed order of steps, with occasional mistakes	Able to recognise given lab procedures and perform them by following prescribed order of steps, with no mistakes
Recording Simulation Observations: *Observe and copy* prescribed or required simulation results in accordance with lab manual instructions	Inability to recognise prescribed or required simulation measurements	Able to recognise prescribed or required simulation measurements but does not record according to given instructions	—	Able to recognise prescribed or required simulation measurements but records them incompletely	Able to recognise prescribed or required simulation measurements and records them completely, in tabular form
Discussion and Conclusion: *Demonstrate* discussion capacity on the recorded observations and draw conclusions from it, relating them to theoretical principles/concepts	Complete inability to discuss recorded observations and draw conclusions	Slight ability to discuss recorded observations and draw conclusions	Moderate ability to discuss recorded observations and draw conclusions	Reasonable ability to discuss recorded observations and draw conclusions	Full ability to discuss recorded observations and draw conclusions

LAB SESSION 03

Objective: To Develop a linear model for a DC motor, and Performance analysis of First order and Second order systems and development of Time response specification's function.

THEORY:

Consider a DC motor, whose electric circuit of the armature and the free body diagram of the rotor are shown in Figure.



Consider the following values for the physical parameters:

moment of inertia of the rotor	$J = 0.01$ kg· m^2
damping (friction) of the mechanical system	$b = 0.1$ Nms
(back-)electromotive force constant	$K = 0.01$ Nm/A
electric resistance	$R = 1\ \Omega$
electric inductance	$L = 0.5$ H

The input is the armature voltage V (ea) in Volts (driven by a voltage source).

Measured variables are the angular velocity of the shaft w in radians per second, and the shaft angle Q in radians.

We can write the following equations based on the Newton's law combined with the Kirchhoff's law:

$$L(di/dt) + Ri = V - K(d\text{Ø}/dt)$$

$$J(d^2\text{Ø}/dt^2) + b(d\text{Ø}/dt) = Ki$$

Or

$$LmDim + Rmim + em = ea$$

$$JDwn + Bwn = T$$

Transfer Function:

The transfer function from the input voltage, V(s), to the output angle, Q, directly follows:

$$\frac{\theta(s)}{V(s)} = \frac{K}{s[(R+Ls)(Js+b)+K^2]}$$

And the transfer function from the input voltage, V(s), to the output velocity of the shaft

$$\frac{\omega(s)}{V(s)} = \frac{K}{(R+Ls)(Js+b)+K^2}$$

w in radians per second.

PROCEDURE:

SIMULINK Model

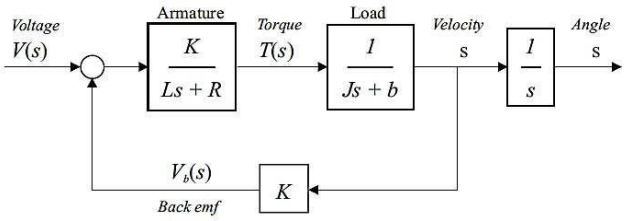
J = 0.01 %kg m2

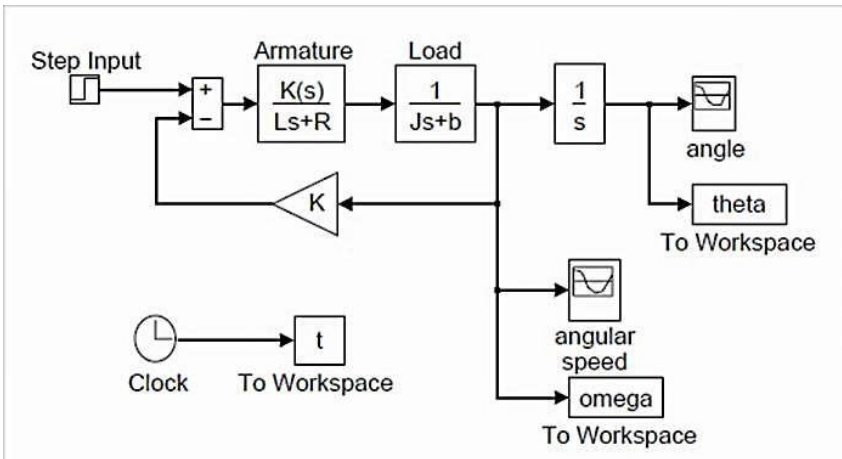
b = 0.1 %Nms

K = 0.01 %Nm/A

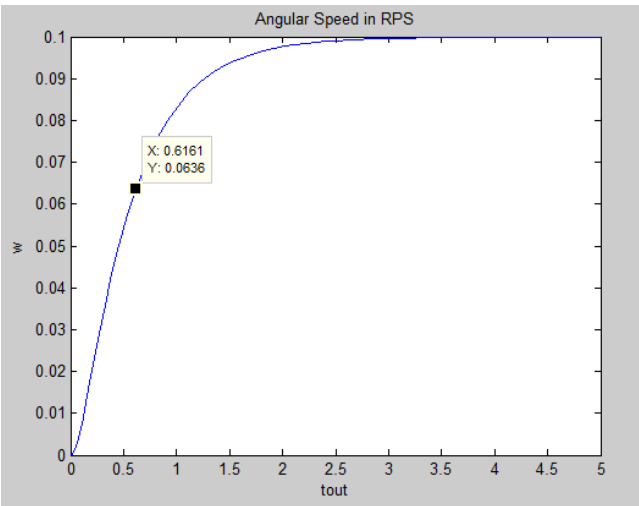
R = 1 %ohm

L = 0.5 %H





GRAPH:



OBSERVATIONS:

Parameters	**Effects on system**
Moment of inertia	
Resistance	
Inductance	
Friction Coefficient	

CONCLUSION:

EXERCISE:

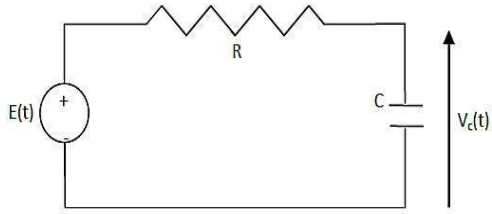
1. Change the electrical parameters such as ‘R’ or ‘L’ to reduce the Time Constant of motor.
2. Also change the mechanical parameters such as ‘J’ or ‘B’ to reduce the Time Constant to zero.
3. What are the parameters which are responsible to change the speed of the rotor? Explain with graph.

Performance of First order and Second order systems and development of Time response specifications function

THEORY:

First order system:

An electrical RC-circuit is the simplest example of a first order system. It comprises of a resistor and capacitor connected in series to a voltage supply as shown below on Figure 1



Where ;

- Vc(t) is the voltage across the capacitor,
- R is the resistance and
- C is the capacitance.

Obtain the transfer function of the above electrical circuit. (Take Vc as output and Vc(0)=Vo)

For the RC-circuit as shown in Figure, the equation governing its behavior is given by :

$$\frac{dv_c(t)}{dt} + \frac{1}{RC}v_c(t) = \frac{1}{RC}E \text{ where } v_c(0) = v_0$$

The constant is the time constant of the system and is defined as the time required by the system output i.e. Vc(t) to rise to 63% of its final value (which is E). Hence the above equation can be expressed in terms of the time constant as:

$$\tau\frac{dv_c(t)}{dt} + v_c(t) = E$$

Transfer Function

Obtaining the transfer function of the above differential equation, we get

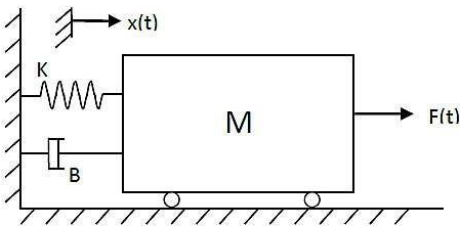
$$\frac{V_c(s)}{E(s)} = \frac{1}{\tau s+1}$$

The above system is known as the first order system.

The performance measures of a first order system are its time constant and its steady state.

Second Order System:

Consider the following Mass-Spring system shown



Where ;

- K is the spring constant,
- B is the friction coefficient,
- x(t) is the displacement and
- F(t) is the applied force:

The differential equation for the above Mass-Spring system can be derived as follows:

$$M\frac{d^2x(t)}{dt^2} + B\frac{\partial x(t)}{\partial t} + Kx(t) = F(t)$$

Transfer Function

Applying the Laplace transformation, we get

$$(Ms^2 + Bs + K) * X(s) = F(s)$$

Provided that, all the initial conditions are zero. Then the transfer function representation of the system is given by

$$TF = \frac{Output}{Input} = \frac{F(s)}{X(s)} = \frac{1}{(Ms^2 + Bs + K)}$$

The above system is known as a second order system.

The generalized notation for a second order system described above can be written as

$$Y(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} R(s)$$

With the step input applied to the system, we obtain

$$Y(s) = \frac{\omega_n^2}{s(s^2 + 2\zeta\omega_n s + \omega_n^2)}$$

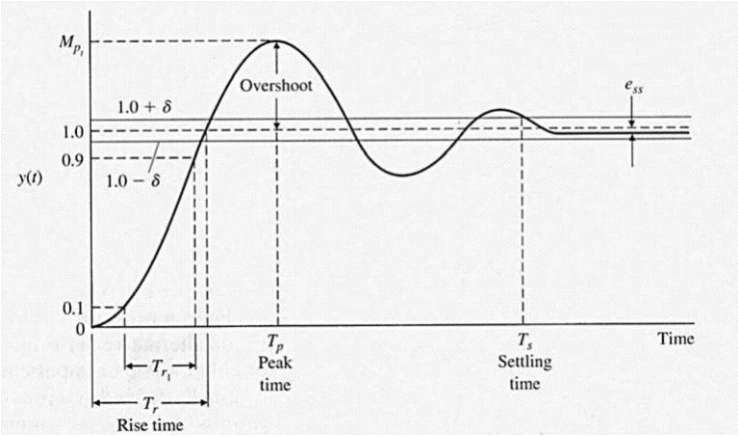
For which the transient output, as obtained from the Laplace transform table

$$y(t) = 1 - \frac{1}{\sqrt{1-\zeta^2}} e^{-\zeta\omega_n t} \sin(\omega_n \sqrt{1-\zeta^2}\, t + \cos^{-1}(\zeta))$$

o where $0 < \zeta < 1$.

o The transient response of the system changes for different values of damping ratio, ζ.

o Standard performance measures for a second order feedback system are defined in terms of step response of a system.



The performance measures could be described as follows

- **Rise Time 'Tr':**

measures the time from 10% to 90% of the response to the step input.

- **Peak Time 'Tp':**

The time for a system to respond to a step input and rise to peak response.

Overshoot

The amount by which the system output response proceeds beyond the desired response.

It is calculated as

$$\text{P.O.}=\frac{M_{p_t}-fv}{fv}\times 100\%$$

where M_{Pt} is the peak value of the time response, and fv is the final value of the response.

Settling Time 'Ts':

The time required for the system's output to settle within a certain percentage of the input amplitude (which is usually taken as 2%). Then, settling time, Ts, is calculated as

$$T_s=\frac{4}{\zeta\omega_n}$$

Delay Time 'Td':

It is the time required for the response to reach 50% of the final value the very first time.

OBSERVATIONS:

1. Effect of damping ratio 'ζ' on performance measures of the second order system. Find the step response of the system for values of ωn = 1 and ζ = 0.1, 0.4, 0.7, 1.0 and 2.0.

Plot all the results in the same figure window and fill the following table.

ζ	Rise time	Peak Time	% Overshoot	Settling time	Steady state value
0.1					
0.4					
0.7					
1.0					
2.0					

CONCLUSION:

Exercise

1. Given the values of R and C, obtain the unit step response of the first order system.

ii. R=2KΩ and C=0.01F

iii. R=2.5KΩ and C=0.003F

Verify in each case that the calculated time constant (τ=RC) and the one measured from the figure as 63% of the final value are same. Obtain the steady state value of the system.

2. Understand the below codes for the time specification of second order.

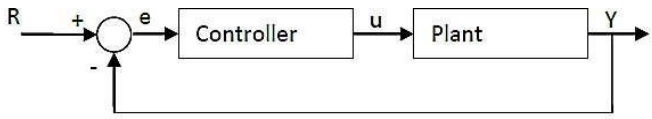
```
function steptimespec % find time specification for step
response of a second order system and calculates the rise
time,
%delay time, maximum overshoot, peak time and settling time
of a system
%whose damping ratio and natural frequency are known.
clc;
zeta=input('Enter the value of damping ratio ');
wn=input('Enter the value of Natural frequency ');
n=wn*wn;
d=[1 2*zeta*wn wn*wn];
disp('The transfer function is: ')
printsys(n,d);
t=0:0.02:6.0;
[y,x,t]=step(n,d,t);
plot(t,y);
grid on;
title('step response');
%to find rise time i.e. time taken for output to rise from
10% to 90%
k=1;
while y(k)<=0.1;
    k=k+1;
end

tenpercent=t(k);
while y(k)<=0.9;
    k=k+1;
end
nintypercent=t(k);
rtime=nintypercent-tenpercent;
fprintf('The rise time is: %f sec \n',rtime);
format short
% to find delay time i.e. time taken to rise to 50% of step
k=1;
while y(k)<=0.5;
    k=k+1;
end
dtime=t(k);
fprintf('The delay time is: %f sec\n', dtime);


% to find maximum
overshootfor k=1:1:300;
    if y(k+1)<=y(k);
        % to find value of k till response keeps
        risingbreak;
    end
;end;
```

```
Oshoot=y(k)-1;
fprintf('The overshoot is: %f sec\n', Oshoot);
% to find the peak
timetp=t(k);
fprintf('the peak time is :%f sec\n',tp)
% to find the settling time
%maximum tolerance for comnsidering output to be in
steadystate taken as
%2%
tol=0.02;
for k=300:-1:2;
    if(abs(y(k)-y(300))>tol)
        break;
    end;
end;
stime=t(k)
;
fprintf('the settling time is :%f sec\n',stime)
```

LAB SESSION 04

Objective: Study the three-term (PID) controller and its effects on the feedback loop response. Also investigate the characteristics of the each of proportional (P), the integral (I), and the derivative (D) controls and obtaining a desired response by using them.

THEORY: Consider the following unity feedback system:



Plant: A system to be controlled.

Controller: Provides excitation for the plant; Designed to control the overall system behavior.

The three-term controller: The transfer function of the PID controller looks like the following:

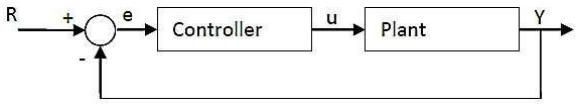
$$K_P + \frac{K_I}{s} + K_D s = \frac{K_D s^2 + K_P s + K_I}{s}$$

KP = Proportional gain

KI = Integral gain

KD = Derivative gain

First, let's take a look at how the PID controller works in a closed-loop system using the schematic shown.



The variable (e) represents the tracking error, the difference between the desired input value (R) and the actual output (Y).

This error signal (e) will be sent to the PID controller, and the controller computes both the derivative and the integral of this error signal.

The signal (u) just past the controller is now equal to the proportional gain (KP) times the magnitude of the error plus the integral gain (KI) times the integral of the error plus the derivative gain (KD) times the derivative of the error.

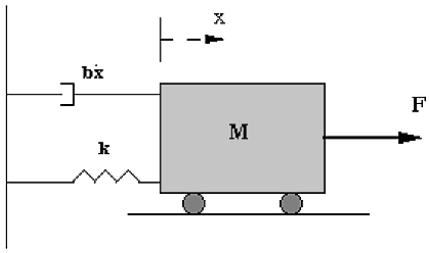
$$u = K_P e(t) + K_I \int e(t)dt + K_D \frac{de(t)}{dt}$$

This signal (u) will be sent to the plant, and the new output (Y) will be obtained.

This new output (Y) will be sent back to the sensor again to find the new error signal (e). The controller takes this new error signal and computes its derivatives and its internal again. The process goes on and on.

PROCEDURE:

For a simple mass, spring, and damper problem.



The transfer function between the displacement X(s) and the input F(s) then becomes:

$$\frac{X(s)}{F(s)} = \frac{1}{Ms^2 + bs + k}$$

Let

- M = 1kg
- b = 10 N.s/m
- k = 20 N/m
- F(s) = 1
- Plug these values into the above transfer function

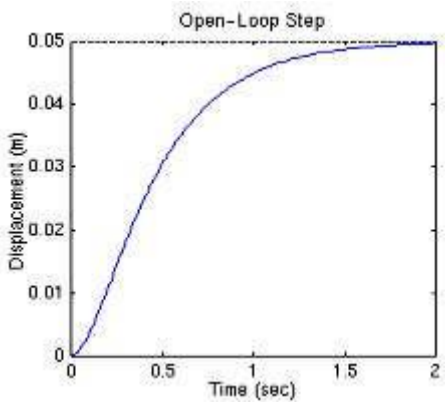
$$\frac{X(s)}{F(s)} = \frac{1}{s^2 + 10s + 20}$$

The goal of this problem is to show you how each of Kp, Ki and Kd contributes to obtain

- Fast rise time
- Minimum overshoot
- No steady-state error
- Open-loop step response:
- Let's first view the open-loop step response.

MATLAB command window should give you the plot shown below.

```
num=1;
den=[1 10 20];
plant=tf(num,den);
step(plant)
```



o 0.05 is the final value of the output to a unit step input.

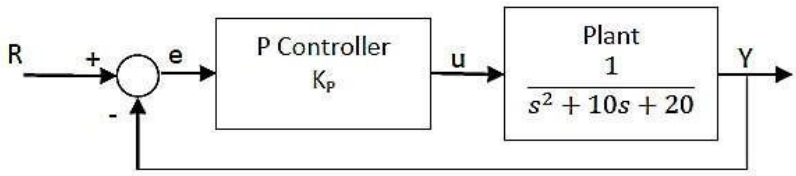
o This corresponds to the steady-state error of 0.95, quite large indeed.

o Furthermore, the rise time is about one second, and the settling time is about 1.5 seconds.

o Let's design a controller that will reduce the rise time, reduce the settling time, and eliminates the steady-state error.

Proportional control:

- The closed-loop transfer function of the above system with a proportional controller is:



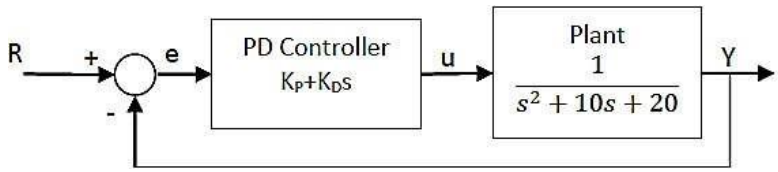
$$\frac{X(s)}{F(s)} = \frac{K_P}{s^2 + 10s + (20 + K_P)}$$

o Let the proportional gain (KP) equal 300: MATLAB PROGRAM:

```
Kp=300;

contr=Kp;

sys_cl=feedback(contr*plant,1); %by default –ve feedback t=0:0.01:2;

step(sys_cl,t)
```

Proportional-Derivative control:

- The closed-loop transfer function of the given system with a PD controller is:



$$\frac{X(s)}{F(s)} = \frac{K_D s + K_P}{s^2 + (10 + K_D)s + (20 + K_P)}$$

O Let KP equal 300 as before and let KD equal 10.

Proportional-Derivative control:

o Kp=300;

o Kd=10;

o contr=tf([Kd Kp],1);

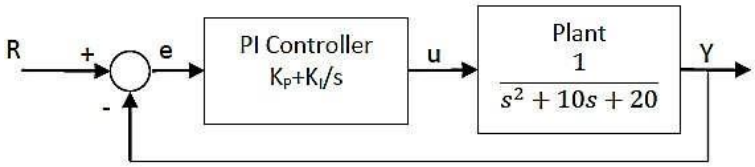
o sys_cl=feedback(contr*plant,1);

o t=0:0.01:2;

o step(sys_cl,t)

Proportional-Integral control:

- The closed-loop transfer function of the given system with a PI controller is:



$$\frac{X(s)}{F(s)} = \frac{K_P s + K_I}{s^3 + 10s^2 + (20 + K_P)s + K_I}$$

o Let KP equal 30 and let KI equal 70.

Proportional-Integral control:

o Kp=30;

o Ki=70;

o contr=tf([Kp Ki],[1 0]);

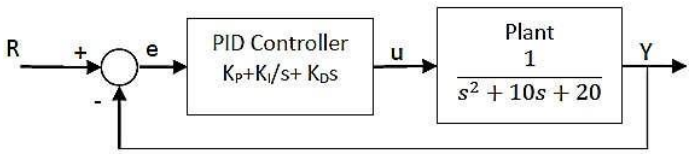
o sys_cl=feedback(contr*plant,1);

o t=0:0.01:2;

o step(sys_cl,t)

Proportional-Integral-Derivative control:

- Now, let's take a look at a PID controller. The closed-loop transfer function of the given system with a PID controller is:



$$\frac{X(s)}{F(s)} = \frac{K_D s^2 + K_P s + K_I}{s^3 + (10 + K_D)s^2 + (20 + K_P)s + K_I}$$

After several trial and error runs, the gains Kp=350, Ki=300, and Kd=50 provided the desired response.

Proportional-Integral-Derivative control:

- Kp=350;
- Ki=300;
- Kd=50;
- contr=tf([Kd Kp Ki],[1 0]);
- sys_cl=feedback(contr*plant,1);
- t=0:0.01:2;
- step(sys_cl,t)

OBSERVATIONS:

CL Response	**Rise time**	**Overshoot time**	**Settling time**	**S-S error**
K_P				
K_I				
K_D				

RESULTS:

Plots of all the simulated systems with their rise time, settling time and final value

NED University of Engineering & Technology
Department of _____ Engineering

Psychomotor Domain Assessment Rubric for Laboratory (Level P3)					
Skill(s) to be assessed	Extent of Achievement				
	0	1	2	3	4
Software Menu Identification and Usage: Ability to initialise, configure and *operate* software environment *under supervision*, using menus, shortcuts, instructions etc.	Unable to understand and use software menu	Little ability and understanding of software menu operation, makes many mistake	Moderate ability and understanding of software menu operation, makes lesser mistakes	Reasonable understanding of software menu operation, makes no major mistakes	Demonstrates command over software menu usage with frequent use of advance menu options
Procedural Programming of given model: *Practice* procedural programming techniques, in order to code specific model	Little to no understanding of procedural programming techniques	Slight ability to use procedural programming techniques for coding given algorithm	Mostly correct recognition and application of procedural programming techniques but makes crucial errors for the given model	Correctly recognises and uses procedural programming techniques with no errors but unable to run model successfully	Correctly recognises and uses procedural programming techniques with no errors and runs model successfully
Relating Theoretical Concepts, Equations and Transforms to Code: *Recognise* relation between model concepts and written code and *manipulate* the code in accordance of requirements	Completely unable to relate between model concepts and written code, unable to do manipulations	Able to recognise some relation between model concepts and written code, unable to do manipulations	Able to recognise relation between model concepts and written code, unable to do manipulations	Able to recognise relation between model concepts and written code, able to do some manipulations	Able to recognise relation between model concepts and written code, able to completely manipulate code in line with theoretical concepts
Detecting and Removing Errors: *Detect* Errors/Exceptions and in simulation and *manipulate* code to rectify the simulation	Unable to check and detect error messages and indications in software	Able to find error messages and indications in software but no understanding of detecting those errors and their types	Able to find error messages and indications in software as well as understanding of detecting some of those errors and their types	Able to find error messages in software as well as understanding of detecting all of those errors and their types	Able to find error messages in software along with the understanding to detect and rectify them

Graphical Visualisation and Comparison of model **Parameters:** *Manipulate* given simulation under supervision, in order to produce graphs/plots for measuring and comparing model parameters	Unable to understand and utilise visualisation or plotting features	Ability to understand and utilise visualisation and plotting features with frequent errors	Ability to understand and utilise visualisation and plotting features successfully but unable to compare and analyse them	Ability to understand and utilise visualisation and plotting features successfully, partially able to compare and analyse them	Ability to understand and utilise visualisation and plotting features successfully, also able to compare and analyse them
Following step-by-step procedure to complete lab work: *Observe, imitate and operate* software to complete the provided sequence of steps	Inability to recognise and perform given lab procedures	Able to recognise given lab procedures and perform them but could not follow the prescribed order of steps	Able to recognise given lab procedures and perform them by following prescribed order of steps, with frequent mistakes	Able to recognise given lab procedures and perform them by following prescribed order of steps, with occasional mistakes	Able to recognise given lab procedures and perform them by following prescribed order of steps, with no mistakes
Recording Simulation Observations: *Observe and copy* prescribed or required simulation results in accordance with lab manual instructions	Inability to recognise prescribed or required simulation measurements	Able to recognise prescribed or required simulation measurements but does not record according to given instructions	—	Able to recognise prescribed or required simulation measurements but records them incompletely	Able to recognise prescribed or required simulation measurements and records them completely, in tabular form
Discussion and Conclusion: *Demonstrate* discussion capacity on the recorded observations and draw conclusions from it, relating them to theoretical principles/concepts	Complete inability to discuss recorded observations and draw conclusions	Slight ability to discuss recorded observations and draw conclusions	Moderate ability to discuss recorded observations and draw conclusions	Reasonable ability to discuss recorded observations and draw conclusions	Full ability to discuss recorded observations and draw conclusions

LAB SESSION 05

Objective:

Introduction to Programmable Logic Controllers (PLCs), their use and applications in industry, method for configuring and programming PLCs using ladder language

Background of Industrial Automation:

Industrial automation refers to the use of advanced technologies and machines to automate manufacturing processes in various industries such as automotive, aerospace, food and beverage, pharmaceuticals, and more. The goal of industrial automation is to increase efficiency, productivity, and quality while reducing production costs and human error.

One of the main components of industrial automation is the use of robotic systems to perform repetitive and hazardous tasks. These robots can be programmed to perform a wide range of tasks, from assembly and welding to packaging and material handling. By using robots, companies can improve production speed and accuracy while reducing the risk of accidents and injuries to workers.

Another key aspect of industrial automation is the use of advanced sensors and monitoring systems to gather data about production processes in real-time. This data can be used to optimize production processes, detect and prevent equipment failures, and ensure consistent product quality. Additionally, automated systems can be equipped with artificial intelligence and machine learning algorithms that can analyze data and make adjustments to improve performance.

Fig.1 Depiction of the elements involved with Industrial Automation

Overall, industrial automation has revolutionized the manufacturing industry by making production processes faster, more efficient, and safer. As technology continues to advance, we can expect to see even more innovative solutions that further improve the capabilities of automated systems.

Methods for Deploying Industrial Control Systems:

In industries, control systems are deployed in a variety of ways. The major methods of such deployment is discussed hereunder:

1) Discrete control systems where the system output is usually a simple on /off signal. They are sometimes called on/off or bang-bang control systems. Examples of such systems include water tank filling system, conveyor belt object detection etc.

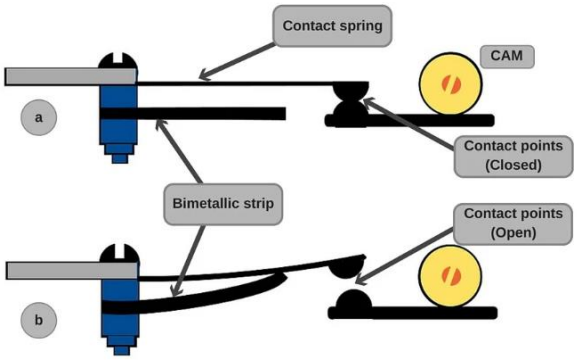


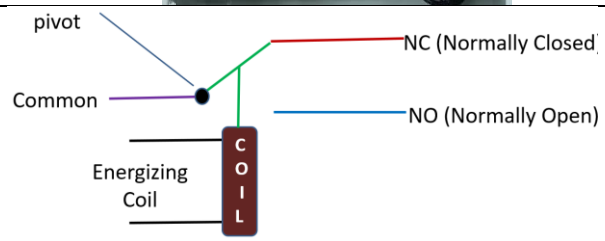
Fig. 2 On-Off control system inside an electric iron. (a) shows position of bi-metallic strip under normal temperature and (b) shows it under high temperature

2) Continuous feedback control system where the system is constantly monitoring the set value and the measured value and adjusting the output to minimise the error. Examples of such systems are motor seed control system, boiler temperature control system etc.

	Fig.3 A continuous feedback temperature controller

3) Open loop and Closed Loop Systems: Some processes require the measured output to be compared with an input and fed back into the system. These are called closed loop systems. In other simpler processes input is changed irrespective of the output e.g. traffic signals and such systems are called open loop systems.

4) State Machine / Sequential Control and Logic: Techniques used to design digital control systems using modules like logic gates, flip-flops, timers, counters etc. A great example of such a deployment is the star-delta motor starter circuit used to run high power induction motors. This method of deployment is sometimes called ‘Relay-Logic’

	Fig. 4a Star-Delta Motor Starter designed with Relay Logic
	Fig. 4b An electromechanical relay lies at the core of Relay Logic circuits

5) Computer Control: Includes PLCs, PLD, CPLDs etc. This is the state of the art in industrial control systems and almost all newer systems are using this method of deployment.

Digital Control and Automation Hardware used in Industry:

Control and automation in industry is applied through a variety of hardware platforms. Most of the control system implementation is done through digital control methods that are derived from those techniques that you have already explored in the previous labs. Digital control is preferred in most cases due to its 'reliability' and 'reprogrammable' nature. A diverse selection of hardware is utilised for implementing digital control under different application scenarios. The major technologies are enumerated below:

1. GAL : Generic Logic Array
2. PAL : Programmable Array Logic
3. PLD : Programmable Logic Device
4. CPLD : Complex Programmable Logic Device
5. FPGA : Field Programmable Gate Array
6. PLC: Programmable Logic Controllers

Introduction to Programmable Logic Controllers (PLCs):

Programmable logic controllers (PLCs) are digital electronic devices that are designed to control a wide variety of industrial processes and machines. They are essentially specialized computers that are programmed to automate specific tasks, such as monitoring and controlling temperature, pressure, and other parameters in manufacturing processes. PLCs have become increasingly common in industry due to their versatility, reliability, and ability to improve productivity and reduce costs.

The primary function of a PLC is to monitor inputs from sensors or other devices, and based on pre-programmed logic, activate or deactivate outputs that control the operation of various machines and equipment. The programming language used to program PLCs is typically ladder logic, a graphical language that is easy to understand and use for people familiar with electrical circuit diagrams. The programming is done through specialized software and can be modified as needed to adapt to changing production requirements or new technology.

PLCs are used in a wide range of industries, including automotive, food and beverage, pharmaceuticals, and manufacturing. In automotive manufacturing, for example, PLCs are used to control robots that assemble parts, paint cars, and perform other tasks. In the food and beverage industry, PLCs are used to monitor and control temperature, humidity, and other conditions in the production process to ensure consistent quality and safety of the products.

The use of PLCs in industry has several advantages over traditional mechanical or electromechanical controls. One of the main benefits is the ability to program the controller to perform complex tasks with high precision and accuracy. This eliminates the need for manual adjustments or corrections, which can be time-consuming and error-prone. Additionally, PLCs are more reliable than mechanical or electromechanical controls because they have no moving parts and are not subject to wear and tear. They can also be easily integrated with other industrial control systems, such as SCADA (Supervisory Control and Data Acquisition) systems, to provide a complete solution for monitoring and controlling industrial processes.

Another advantage of PLCs is their flexibility. Because they are programmable, they can be easily modified to accommodate changes in production requirements or to incorporate new technologies. This means that companies can adapt quickly to changing market conditions or customer needs, which is essential in today's fast-paced business environment.

PLC Brands and Manufacturers:

There are several manufacturers of PLCs, some of the well-known manufacturers are:

Siemens, Allen Bradley (Rockwell Automation), Schneider Electric, Mitsubishi Electric, ABB, Omron, Delta Electronics, Beckhoff Automation, Bosch Rexroth, General Electric (GE).

Fig.5: Allen Bradley PLC

Fig.6: General Electric PLC

Fig.7: Mitsubishi PLC

Fig.8: Delta PLC

These are just a few of the many manufacturers of PLCs, and the choice of manufacturer often depends on specific industrial needs and applications.

Introduction to our PLC experimental setup – The SIEMENS S7-1200

The Siemens S7-1200 is a popular programmable logic controller (PLC) used in various industries. It is a compact and versatile controller that is designed for small to medium-sized automation projects, making it ideal for applications in machine building, plant engineering, and building automation.

Fig.9: The SIEMENS PLC Portfolio. Note that the smallest (simplest) offering is SIEMENS Logo and the largest (complex) is S7-400

Some of the features and specifications of the S7-1200 PLC are:

Modular design - The S7-1200 is a modular system that can be expanded by adding up to three communication modules and eight I/O modules, allowing users to customize the controller to their specific needs.

Programming languages - The S7-1200 supports several programming languages, including ladder logic, function block diagram, and structured text.

Communication capabilities - The S7-1200 has built-in Ethernet and RS485 communication ports, allowing it to communicate with other devices on the network.

Built-in digital and analog inputs and outputs - The S7-1200 has built-in digital and analog inputs and outputs, making it suitable for a wide range of applications.

High-speed counters and pulse outputs - The S7-1200 has high-speed counters and pulse outputs, making it suitable for applications that require precise timing and control.

Data logging - The S7-1200 can log data to a microSD card, allowing users to monitor and analyze system performance over time.

Fig.10: Our SIEMENS S7-1200 PLC rack. The central module is the processor along with IO and communication devices

Overall, the Siemens S7-1200 PLC is a versatile and reliable controller that is suitable for a wide range of automation applications. Its modular design and support for multiple programming languages make it easy to customize and adapt to specific needs, while its communication capabilities and built-in web server make it easy to monitor and control the system remotely.

PLC Programming Languages

There are several programming languages used to program PLCs, each with its own strengths and weaknesses. Here are some of the most commonly used programming languages for PLCs:

Ladder Logic - Ladder logic is the most widely used programming language for PLCs. It is a graphical language that uses ladder-like diagrams to represent the control logic of a system. It is easy to learn and use for people familiar with electrical circuit diagrams.

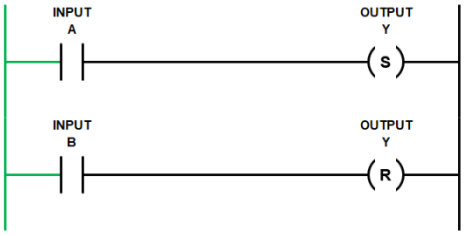


Fig.11(a): Example of Ladder Logic Program

Function Block Diagram (FBD) - FBD is another graphical programming language that uses blocks to represent different functions and logic elements. It is well-suited for complex control systems.

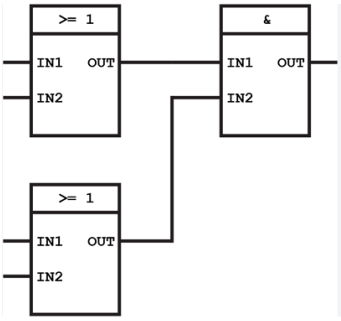


Fig.11(b): Example of FBD Program

Structured Text (ST) - ST is a high-level programming language that resembles Pascal or C. It is used for more complex programming tasks, such as mathematical calculations and data processing.

```
IF Start THEN
    Start:=FALSE; //Remove Start Latch
    WHILE Initialised = FALSE DO
        System_Prime:=TRUE; //Set Output On For System Primer
    END_WHILE
    System_Prime:=FALSE; //Turn Off Output For System Primer Once Initialised
END_IF
```

Fig.11(c): Example of ST Program

Instruction List (IL) - IL is a low-level programming language that is used for more advanced programming tasks, such as configuring hardware interrupts and system functions.

```
LD %I1.1
R   %C8
LD  %I1.2
AND %M0
CU  %C8
LD  %C8.D
ST  %Q2.0
```

Fig.11(d): Example of IL Program

Sequential Function Chart (SFC) - SFC is a graphical programming language that is used to model complex processes with multiple steps and stages.

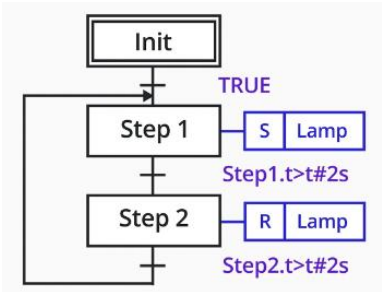


Fig.11(e): Example of SFC Program

Each programming language has its own strengths and weaknesses, and the choice of language depends on the complexity of the task and the programmer's experience and familiarity with the language. Some PLCs support multiple programming languages, allowing programmers to choose the language that best suits their needs. In this lab, we shall use a combination of **Ladder Logic** and **Function Block Diagram** to attain our objectives.

SIEMENS S71200 Trainer Hardware

The training case comprises a SIMATIC S7-1200 automation system. The automation system is mounted in a carrying case for transportation purposes. It consists of:

- S7-1200 Power Supply
- CPU1214 with Signal Board
- Analog output SB1234
- Analog input / output module SM 1234
- Digital input / output module SM 1223
- Ethernet Switch CSM 1277
- Basic Panel KTP600 (HMI LCD with six hard buttons)

Fig. 12 S71200 PLC Trainer (PLC Processor and modules on the top rack, input switches, knobs LEDs and HMI LCD at the base)

In the following section we get to know the individual hardware components of this trainer system.

Getting to know the SIEMENS S71200 Hardware Trainer

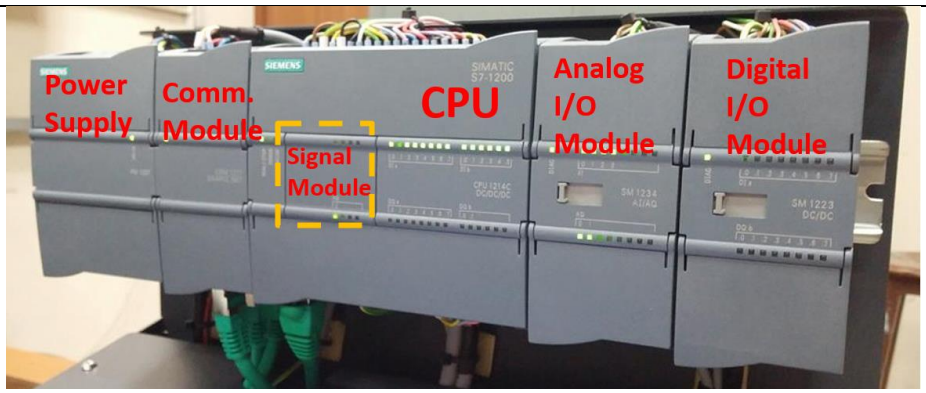
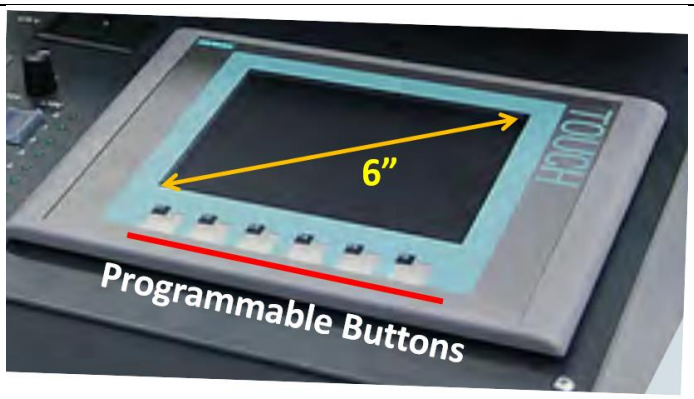
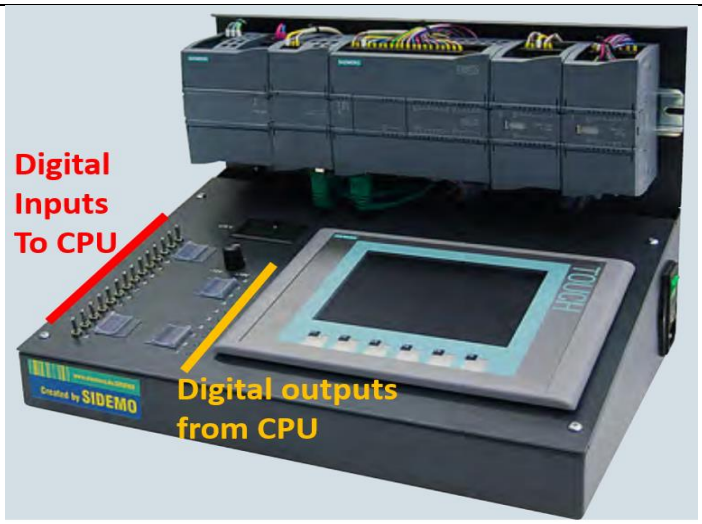
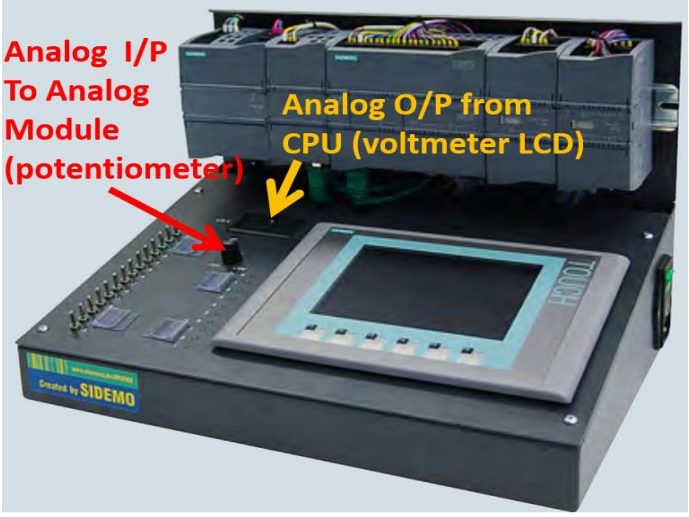
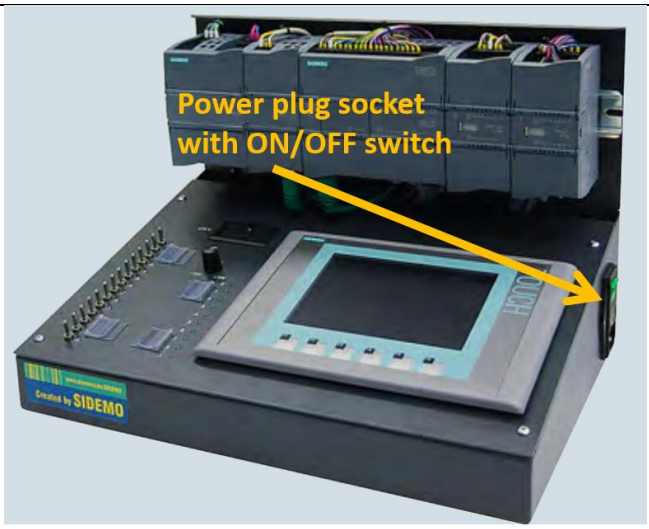
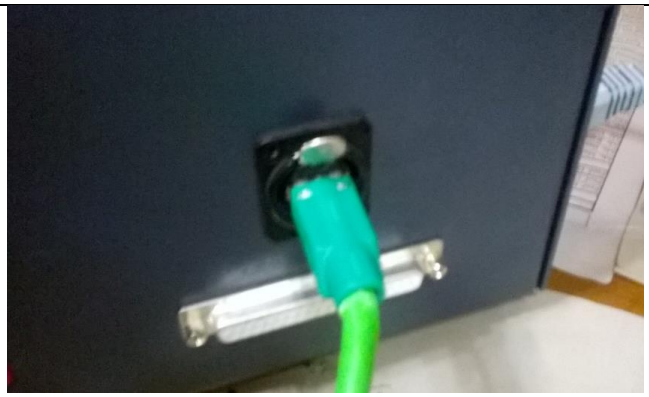
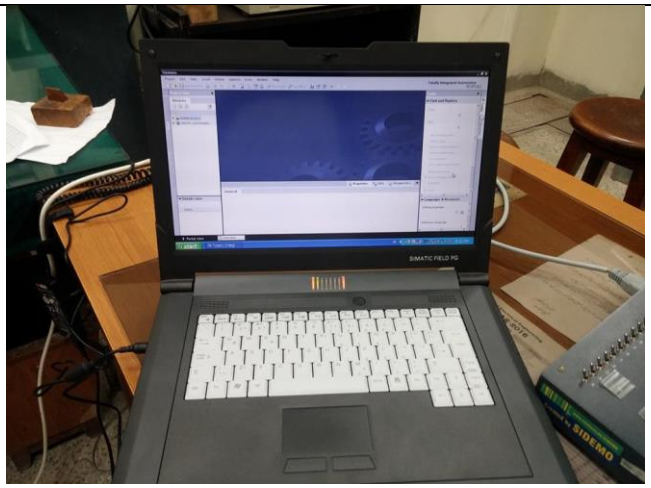


Fig.13(a):The PLC rack consists of , from left to right, a **power supply(PM1207)**, an **ethernet switch** (comm. Module **CSM1277**) with 4 connections, a **CPU (1214C)** with a replaceable **signal module**, an **analogue i/o module(SM1234)** and a **digital i/o module(SM1223)**

	Fig.13(b): KTP600 Human Machine Interface that is composed of a **6” touch screen LCD** along with **six programmable hardware buttons**. The HMI is connected to the CPU via ethernet switch.
	Fig.13(c): 14 digital input switches connected to digital inputs on the CPU. They can work both as an on/off toggle switch (LEFT) and a momentary switch (RIGHT). Also, there are 10 LEDs connected to the digital outputs on the CPU.
	Fig.13(d): An analogue input connected to a potentiometer with -12V to +12V range. The input goes to input number 0.1 of the analogue module. Also, an LCD voltmeter that is connected to analogue output on the signal module that is present on the CPU.

	Fig.13(e): Power plug socket along with ON/OFF switch
	Fig.13(f): Ethernet connector behind the trainer connects the ethernet switch to SIMATIC PG Programmer laptop.
	Fig.13(g): The SIMATIC FIELD PG Programming Laptop. It is installed with TIA Portal Version 10.
	Fig.13(h): Ethernet connector behind the laptop that connects to the trainer. Note two ethernet cards being present of which anyone can be used.

Once the FIELD PG Laptop is connected to the PLC Trainer Via Ethernet cable, a star network topology is created as shown in Fig. 14.

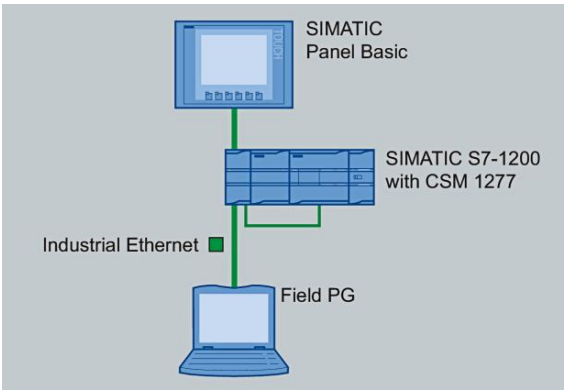


Fig.14: Star topology connecting the FIELD PG laptop to the ethernet switch (CSM 1277) that extends connection to CPU and SIMATIC HMI (LCD) panel

Programming Environment – The SIEMENS TIA Portal

The Totally Integrated Automation Portal (TIA Portal) provides you with unrestricted access to our complete range of digitalized automation services, from digital planning and integrated engineering to transparent operation.

TIA Portal shortens time to market by integrating all important components of your automation project in a single framework: safety, security, control, HMI, drives, switchgear, decentralized peripherals and now also motion control and power distribution. A shared database and a smart library concept allow you to use super-ordinate functions.

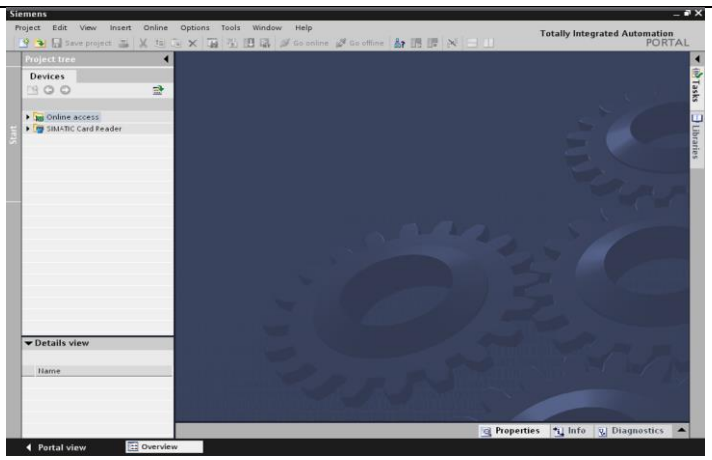
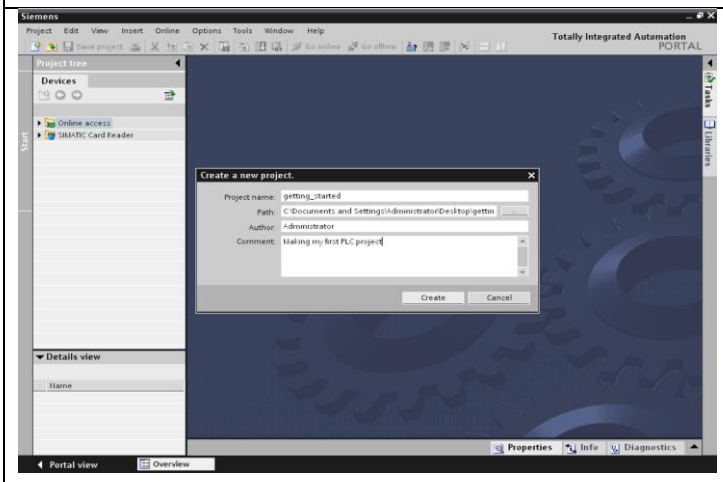
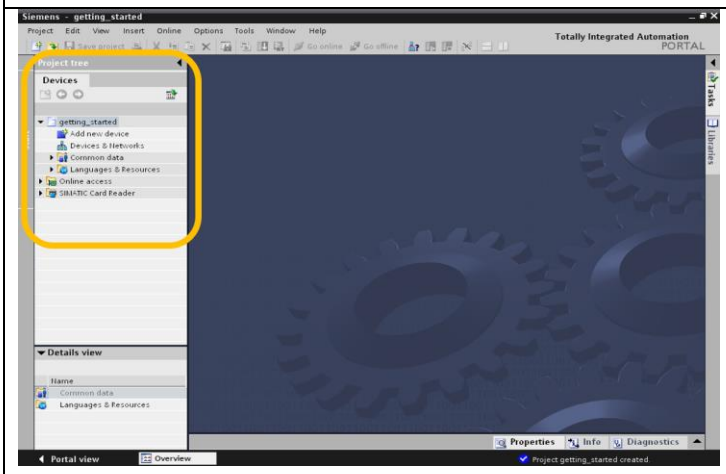
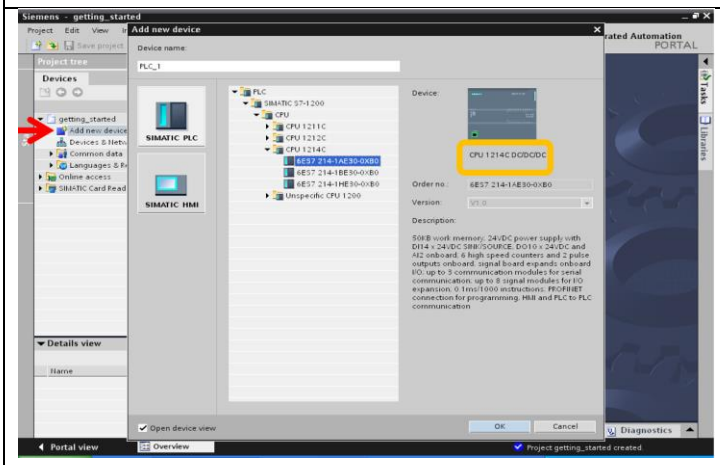
Siemens TIA Portal supports several programming languages for programmable logic controllers (PLCs), including:

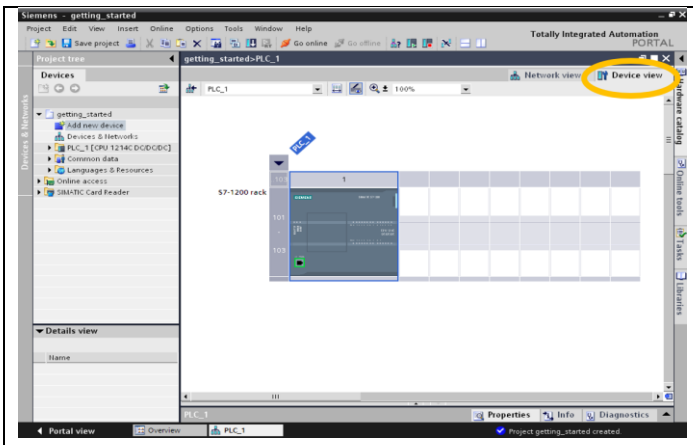
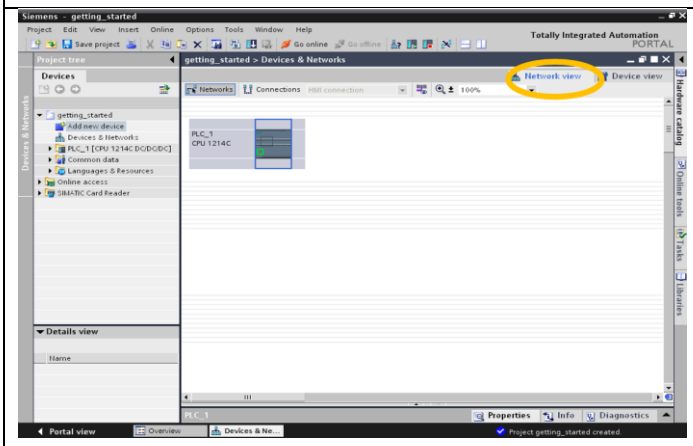
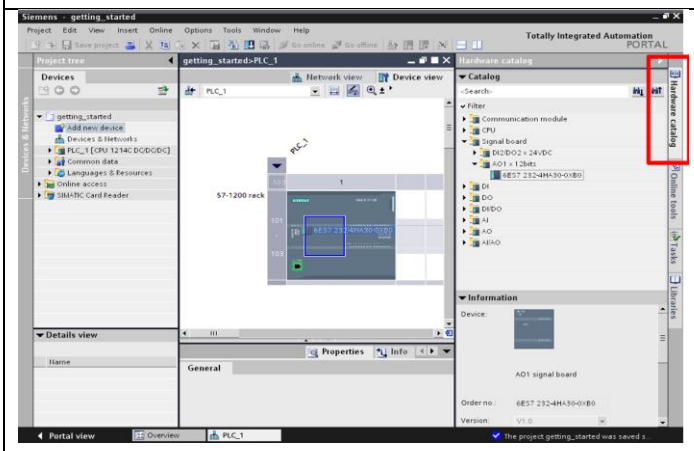
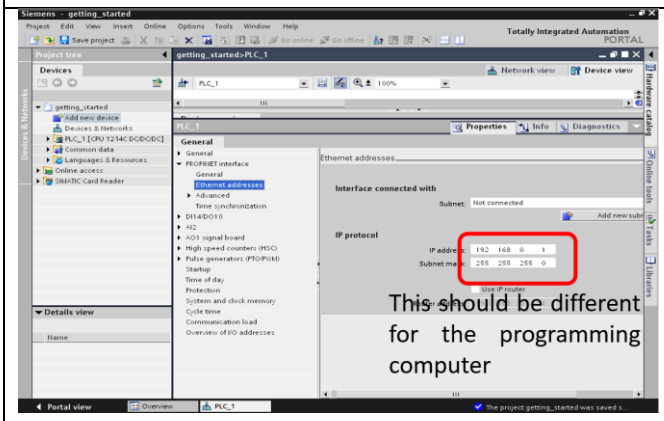
- Ladder Diagram (LD): LD is a graphical programming language that uses ladder-like diagrams to represent logical operations.
- Function Block Diagram (FBD): FBD is a graphical programming language that uses blocks to represent logical operations and functions.
- Structured Text (ST): ST is a text-based programming language that uses structured programming concepts such as loops and conditional statements.
- Sequential Function Chart (SFC): SFC is a graphical programming language that uses a flowchart-like structure to represent the sequence of operations.
- Graphical Function Chart (GFC): GFC is a graphical programming language that combines the elements of FBD and SFC.
- Statement List (STL): STL is a low-level programming language that uses a sequence of instructions to represent logical operations.

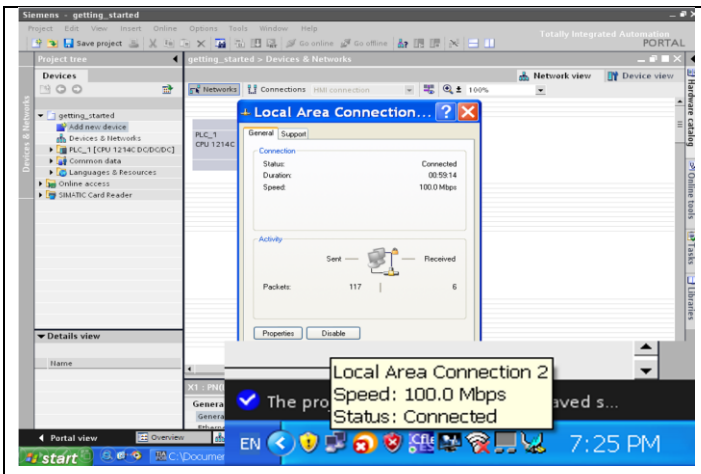
Siemens TIA Portal also supports other programming languages for specific applications, such as C/C++ for embedded systems programming and SIMATIC S7-GRAPH for graph-based programming. However, in this lab we shall stick to **Ladder Logic** and **Function Block Diagram** only.

Exercise 1: Creating the First TIA Portal Project

The following table shall take us through the steps for creating our first PLC program on TIA Portal and deploy it on the PLC.

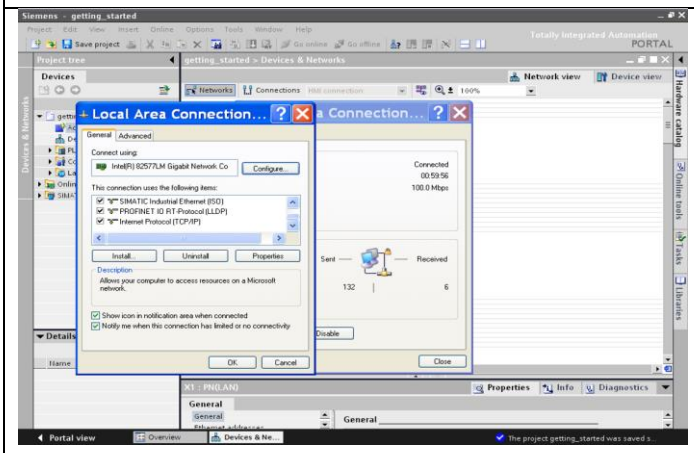
	Step 1: Fig.15(a): Open TIA PORTAL by clicking on the desktop icon
	Step 2: Fig.15(b): Create a new project, assign it a name and set its destination (keep it on the desktop)
	Step 3: Fig.15(c): From the Devices menu, choose Add New Device
	Step 4: Fig.15(d): In the Devices menu that appear, select CPU1214C DC/DC/DC

	Step 5: Fig.15(e): To check your selected device, click on Device View tab present on the top right. Your device appears here on a virtual rack
	Step 6: Fig.15(f): Now check whether your device appears in the network by clicking on the Network View tab on the top right. Your device appearing here means it is going to be connected to the LAN that contains your PLC
	Step 7: Fig.15(g): Now go back to the Device View where you can find Hardware Catalog tab on the right. In this menu, select Signal Board. Pick the one with the name AO1x12bits. This is the signal board physically installed on the CPU. This completes the hardware selection procedure for this lab
	Step 8: Fig.15(h): Double click on the CPU now. This shall open configuration information for the CPU in the bottom pane. Here, you must note down the IP address of the CPU. This shall be helpful later. The IP address is present in the PROFINET Interface menu



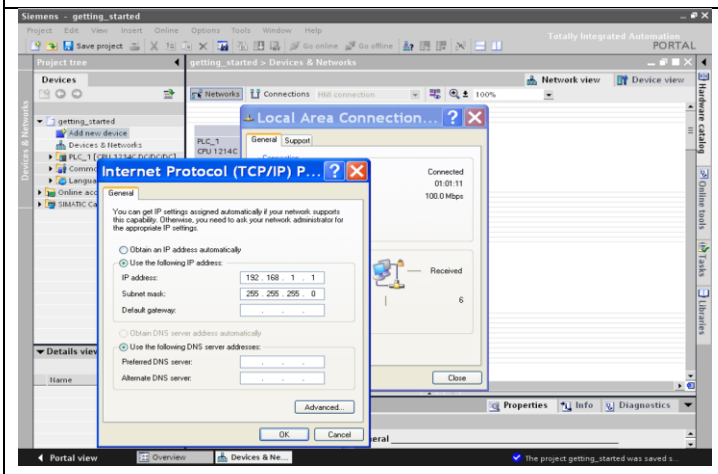
Step 9:

Fig.15(i): Now, set the IP Address of your computer by clicking on the Network Adaptor settings in Windows. This shall lead to the next step



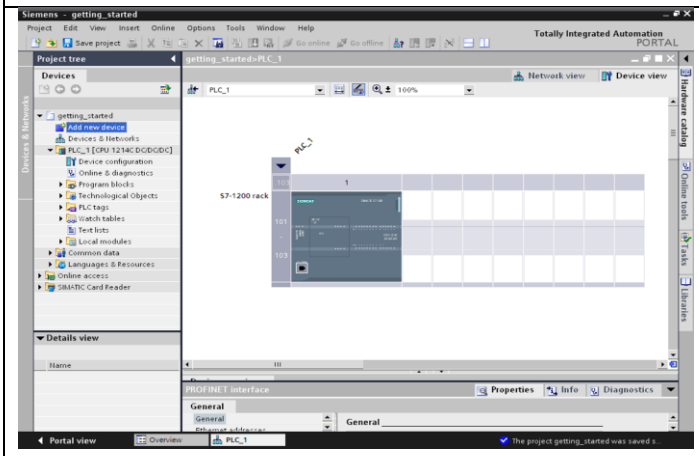
Step 10:

Fig.15(j): Click on Properties and then select Internet Protocol (TCP/IP)



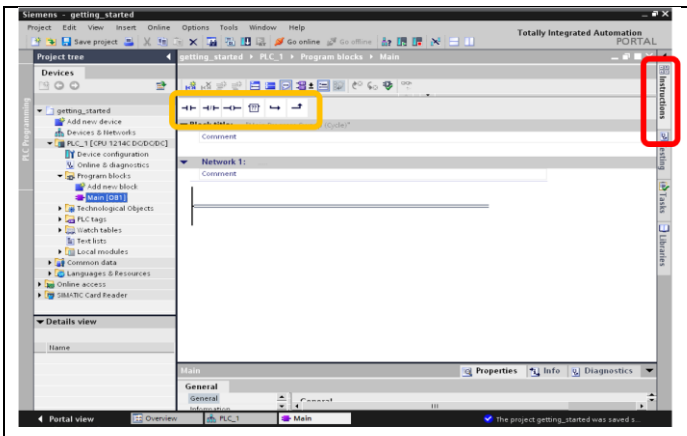
Step 11:

Fig.15(k): Here, you shall see your computer's IP address. It must be different from that of the PLC CPU that you set earlier. Note that the subnet shall be the same as that of PLC.



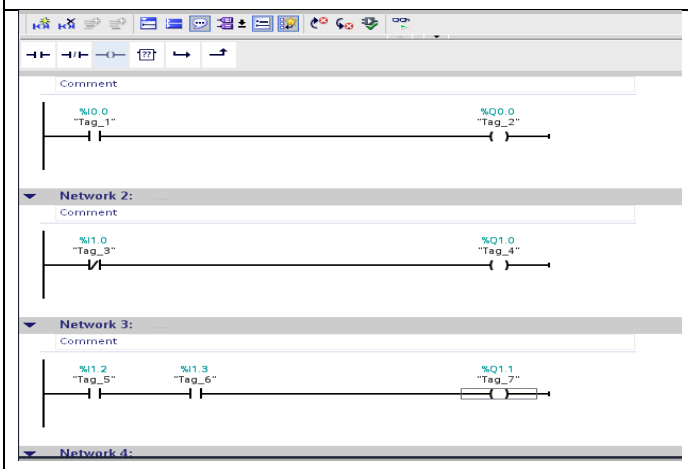
Step 12:

Fig.15(l): Now, in TIA Portal, go to PLC_1 in the project tree and in the Program Blocks sub-menu select Main OB1. You need to double click on it. This shall open a Ladder Logic programming interface



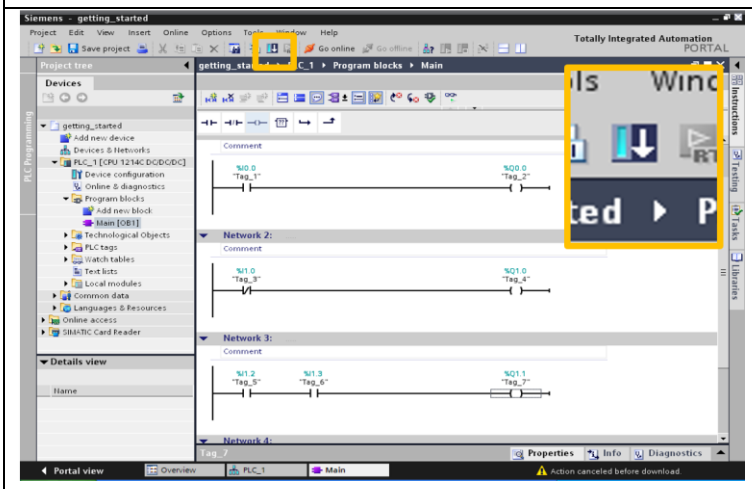
Step 13:

Fig.15(m): Inside Main_OB1, you can write your PLC program just analogous to the main() function in C/C++. Using the basic Ladder instructions present in the highlighted location, create three basic Ladders.



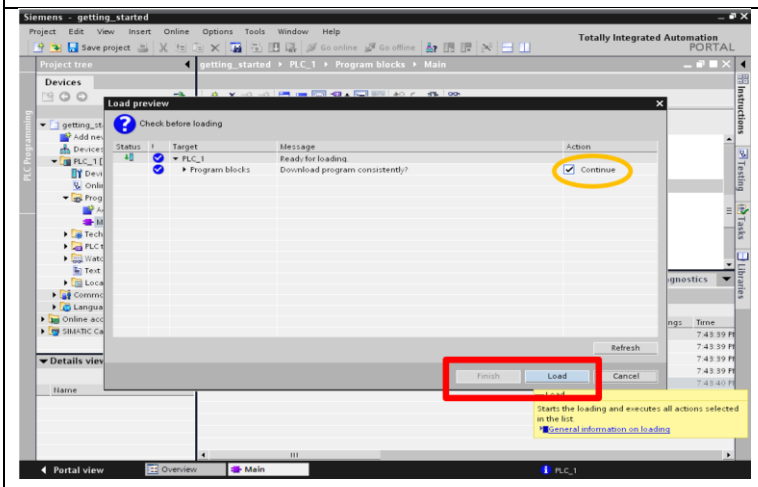
Step 14:

Fig.15(n): Note that you need to drag and drop the instructions on the ladder rung. Each rung can have multiple i/o's. For now, just understand that inputs are connected to the left side and outputs to the right side. Add the tags %I0.0, %Q0.0, %I1.0, %Q1.0, %I1.2, %I1.3 and %Q1.1 on the three rungs shown.



Step 15:

Fig.15(o): Once the program is created, click on the Download button



Step 16:

Fig.15(p): The Download requires you to check the Continue option and click Load

With this your first PLC Program is complete. What function does it perform? You can test that by turning ON/OFF the switches on the PLC trainer. Turn all the switches ON and observe the LEDs. What do you observe? Write down your observation.

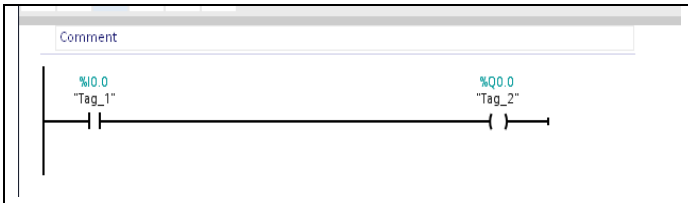
_____

_____

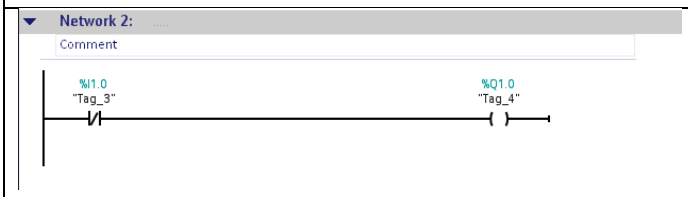
_____

Now, complete the following tasks:

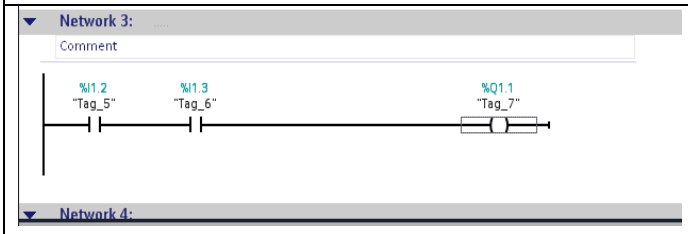
Task 1: Complete the following truth-tables. They are each related to the first, second and third rung of your PLC program, respectively. Turn the toggle switches ON/OFF on the trainer in order to complete the truth table. The mentioned LEDs are to be observed for output



Toggle Switch **0.0**	LED **0.0** (output – this is the top LED)
ON	
OFF	

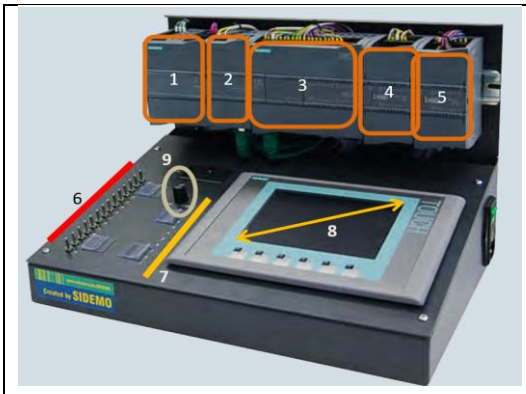


Toggle Switch **1.0** (This is the lower .0 switch)	LED **1.0** (output – this is the lower .0 LED)
ON	
OFF	



Toggle Switch **1.2** (this is the lower .2 switch)	Toggle Switch **1.3** (this is the lower .3 switch)	LED **1.1** (This is the lower .1 LED, bottom one)
ON	ON	
ON	OFF	
OFF	ON	
OFF	OFF	

Task 2: With reference to the following picture, identify and write down the name of each of the highlighted-numbered component/device on the PLC trainer.



1. _____
2. _____
3. _____
4. _____
5. _____
6. _____
7. _____
8. _____
9. _____

LAB SESSION 06

Objective:

Digital I/O interfacing and manipulation in PLCs, their application in designing On-Off type feedback control systems using ladder language.

What is a Digital I/O on a PLC?

Digital inputs have two states: OFF and ON. If voltage is present, the circuit is ON. If it's not present, the circuit is OFF. The voltage values for ON and OFF state depend on the type of digital logic used. In SIEMENS PLCs it is 24V for HIGH/ON and 0V for LOW/OFF

Fig.1: A sensor connected to PLC Digital Input

The simplest control you can use over an electrical device is digital output. In this case, you would either turn something OFF, or ON. Digital outputs are often used to control other electrical devices, through transistors or relays. In S7-1200, digital output HIGH/ON is 24V and output LOW/OFF is 0V.

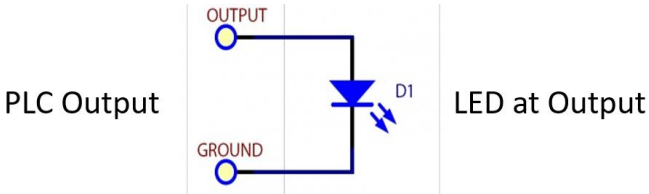


Fig.2: An actuator (LED) connected to PLC Digital Output

Digital I/O Nomenclature for SIEMENS S7-1200

Digital inputs are marked as DI, whereas, digital outputs are abbreviated as DQ in SIEMENS PLC portfolio. Every digital input/output is called upon by its address in a PLC program. The address is given by the following format:

%IX.X for input. To spell it out, a percentage sign is followed by capital i 'I'. Without any space, two single-digit numbers follow, separated by a dot '.'. The first number is called Port Number and the second is called Pin Number.

%QX.X for output. To spell it out, a percentage sign is followed by capital q 'Q'. Without any space, two single-digit numbers follow, separated by a dot '.'. The first number is called Port Number and the second is called Pin Number.

For example, in our trainer, the CPU has two digital input ports. Port 0 and port 1. Port 0 has eight pins with addresses %I0.0 to %I0.7. Port 1 on the other hand has six pins with addresses %I1.0 to %I1.5. You can see them on the processor as well as through the switches that connect to them on the trainer. See Fig. 6.

Similar to inputs, the CPU has two digital output ports; Port 0 and Port 1. Port 0 is eight bits wide containing addresses %Q0.0 to %Q0.7. Port 1 has just two pins with addresses %Q1.0 and %Q1.1. See Fig. 7.

NOTE: I/O addresses are absolute and hardware defined in the Process Image which is a memory to store I/O states in real-time. You can also associate Tags with each I/O address, which act as variable names and can help in keeping track of logic in long programs. Default tags are assigned to each I/O address but it is advisable to rename them to something that is logically meaningful, for example proximity_sensor, level_switch, alarm etc.

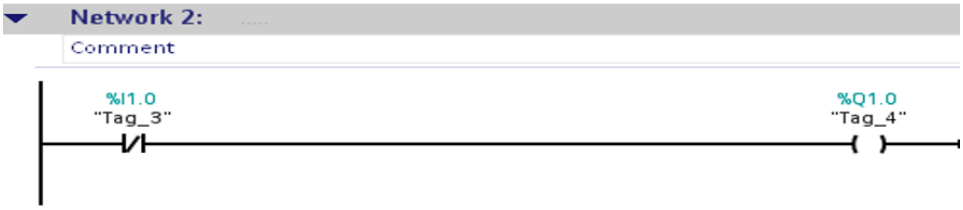


Fig.3 I/O Addresses are in green whereas Tag Names are in black. Remember if you have made mistake in writing I/O address, it turns red.

Digital I/O available on the CPU

The CPU has information about the type of Digital I/O it contains within its name. The CPU name 1214C DC/DC/DC not only describes the processor specifications but also gives three key features of the CPU:

Fig.4 CPU Model Number used in the PLC Trainer

DC/DC/DC=> Power Supply/ Digital Input Type/ Digital Output Type

The first DC signifies the type of power supply for the processor. The second DC tells us about the kind of digital input and the third one tells us about the kind of Digital output. In this case, the CPU is powered by a DC source, it accepts a DC type of digital input and generates a DC type of digital output. Here DC means Direct Current. Just to clarify, other options are also available in PLCs and are briefly explained below:

Other Power Supply options: AC (Alternating Current)

Other Digital Input Type options: none (only DC input is available)

Other Digital Output Type options: AC / RLY (RLY stands for Relay output. It means each output is a Normally Open relay contact which closes when the output goes high. See Fig.3. AC Digital Output is a 24V AC signal turned on by PLC)

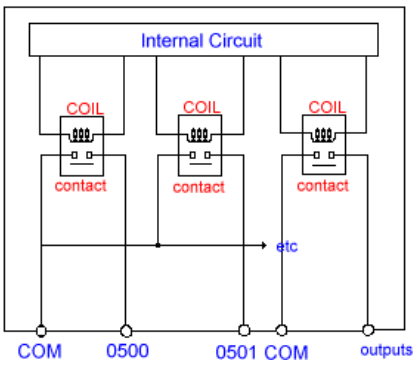


Fig.5 Depiction of a Digital Output that is operated through a Relay built inside the PLC. This allows us to connect high current loads/actuator to the PLC output, for example induction motor

The Digital Inputs available on the CPU is connected to the switches present on the trainer base. All these are simple toggle switches that connect a 24V DC source to the CPU pin when turned ON. The only exception is the second

switch (from the top) I0.1, which is a Normally Closed switch. When these switches are toggled, an indicator LED on the CPU lights up to show activity on the respective input pin. See Fig. 6

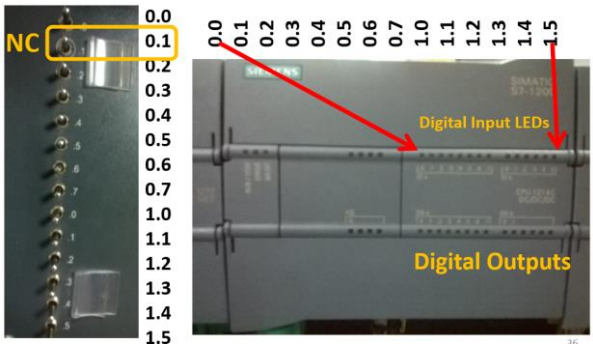


Fig. 6: Digital Inputs from the trainer connect to the respective input pin on the CPU

Digital Outputs from the CPU are also connected to the trainer. The LEDs at the trainer base are sequentially connected to the respective digital output pin as marked on the CPU. Digital outputs also have status LEDs to show their activity.

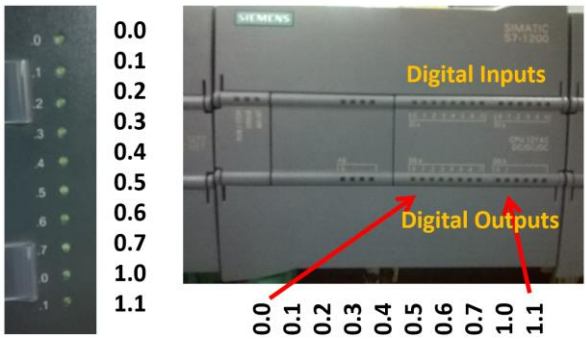


Fig. 7: Digital Outputs from the CPU connect to the respective LED on the trainer base

Digital I/O available through expansion module SM1223

The expansion digital I/O module SM1223 is present at the extreme right of the PLC rack. It also adds 8 digital inputs and 8 digital outputs. However, its is not connected to any point on the trainer. We can use its to connect to other I/O sensors and actuators.

Fig. 8: Digital I/O Expansion Module SM1223 (on the right). Its has no existing connections to the trainer

Types of PLC Digital Sensors and Methods to connect them

Digital Sensor is any physical sensor that has two stable output states. For a PLC, digital sensor has a built-in interface circuit that converts sensor voltages to 0-24V range and limits current drawn by the sensor to 5mA. Fig. 9 depicts a Digital Sensor model.

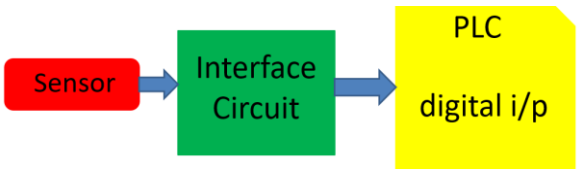


Fig. 9 Digital Input Sensor Model

Digital Sensors that can be connected to a PLC can be categorized into 3 groups:

1) Digital Level Sensors: Sensors with two o/p levels that hold for a measurable/observable amount of time (e.g. fluid level sensor, elevator door switch)

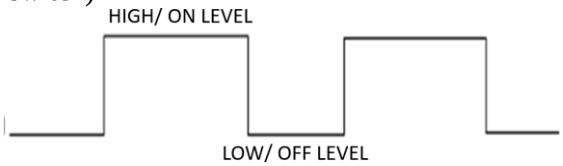


Fig. 10 Depiction of Digital Level Sensor Signal

2) Digital Edge Trigger Sensors: Sensors with two o/p levels that change at a rate faster than the measuring system OR generate momentary pulses that are few and far between (e.g. object counting proximity sensor on conveyor belt, level sensors used in elevators for counting floors)

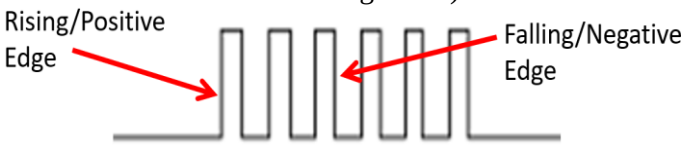


Fig. 11 Depiction of Digital Edge Trigger Sensor Signal

3) Digital Pulse Out Sensors: Sensors that generate a burst of pulses at high speed. Number of pulses represent a physical quantity. (e.g. shaft encoder for dc motors for measuring speed). This kind of input cannot be interfaced to all digital inputs and dedicated high-speed digital inputs are present for this.

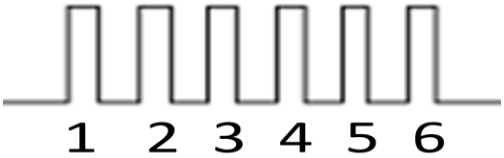


Fig. 12 Depiction of Digital Pulse Out Sensor Signal where pulses are counted instead of measuring their time

Types of PLC Digital Actuators and Methods to connect them

Digital outputs of a PLC are connected to relays or other ON/OFF actuators that have only two discrete levels of operation. These ON/OFF actuators can be categorized into two different kinds of outputs:

1) ON-OFF Outputs: Devices that simply turn ON or OFF. Interface to the device is usually a relay or high power MOSFET (e.g. induction motor, lighting etc.)

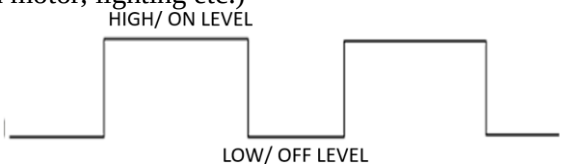


Fig. 13 Depiction of Digital Level Actuator Output

2) PWM (Pulse Width Modulated) Output: A digital output of the form of a square wave with a certain frequency. The on-time and off-time may be controlled. Ratio of on-time over total cycle time is called PWM duty. This is used to control speed of DC motors and change angular position of servo motors. It is not available on all digital outputs. And only dedicated outputs can perform this feature.

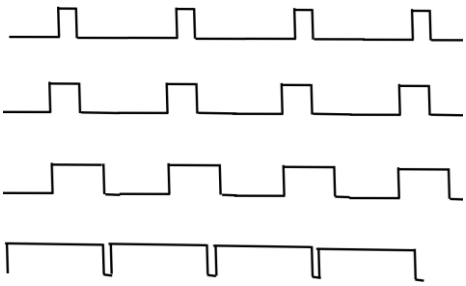


Fig. 14 Depiction of PWM Digital Output Signal

Wiring Digital Inputs and Outputs to PLC

The final thing we need to understand in order to interface digital inputs and outputs are the input/output wiring connections. For our CPU 1214C DC/DC/DC, we can connect a 24V DC source between the L+ (+ve) and M(-ve) terminals in the processor, these are exposed when the flaps of the CPU are turned over. For the individual sensors, which can be taken as switches for all practical purposes, one terminal of the sensor needs to be connected to a 24V DC voltage that has its negative terminal connected to 1M terminal. The other terminal then needs to connect to one of the digital inputs (e.g. 0.0 or 0.1). This connection scheme works for normally open (NO) type sensors.

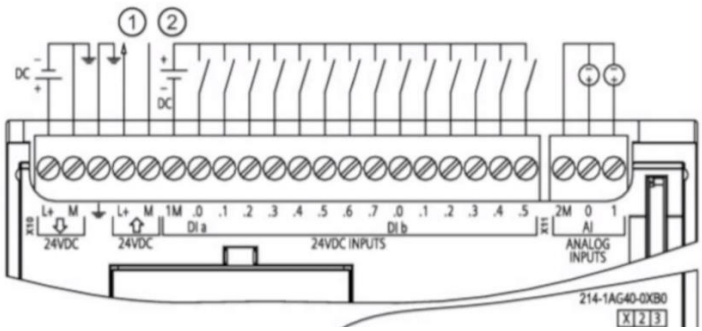


Fig. 15 Note DC supply on L+ and M terminals. Another DC supply is used to power the Digital Sensors and grounded through 1M terminal

As for the digital outputs, the connection required connecting 24V DC source between 3L+ and 3M as shown in Fig. 14. All digital output actuators are then connected between the respective digital output and 3M (common ground).

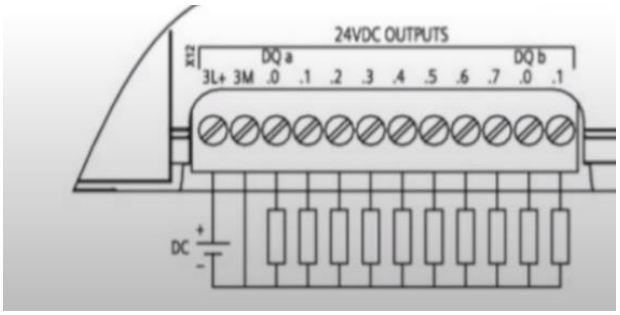
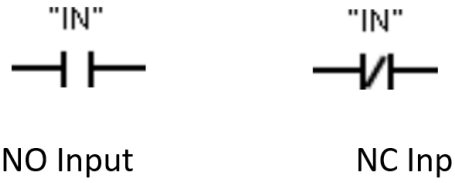


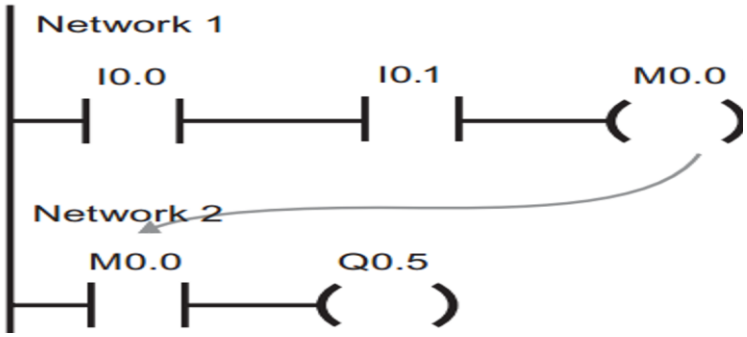
Fig. 16 Note DC supply on 3L+ and 3M terminals. Digital Outputs are connected between the digital output pin and common ground (3M)

NOTE: We are using simple LEDs and ON/OFF relays as digital output actuators for the major portion of these labs.

Types of Digital I/O Instructions and their Usage

Digital I/O is used in conjunction with specific software instructions in Ladder or FBD language. There is even more specification with regards to the type of instruction to be used with certain kind of digital input sensor or digital output actuator. In the following table, different kinds of digital I/O instructions are presented along with the type of sensor/actuator they are designed to be used.

Digital Level Sensors Sensors with two o/p levels that hold for a measurable/observable amount of time (e.g. fluid level sensor, elevator door switch)	Digital Level Sensors These sensors are used with simple input contacts  NO Input NC Input "IN" is a boolean address of the input e.g. **%I0.5**
Digital Edge Trigger Sensors Sensors with two o/p levels that change at a rate faster than the measuring system (e.g. object counting proximity sensor on conveyor belt, level sensors used in elevators for counting floors)	Digital Edge Trigger Sensors These sensors are used with edge trigger input contacts +ve edge input -ve edge input "IN" is boolean digital i/p, "M_BIT" is boolean memory address e.g. **%M0.0**
Digital Pulse Out Sensors Sensors that generate a burst of pulses at high speed. Number of pulses represent a physical quantity. (e.g. shaft encoder for dc motors for measuring speed). Not available on all digital inputs.	These can't be used with regular digital inputs. They connect with specialized digital inputs that have built-in functionality to register and count pulses. This shall be used in later lab sessions.
ON-OFF Outputs: Devices that simply turn ON or OFF. Interface to the device is usually a relay or high power MOSFET (e.g. induction motor, lighting etc.)	ON-OFF Outputs These actuators can be connected with simple output coils in the PLC programming NO output NC output "OUT" is boolean address of the output e.g. **%Q0.3**

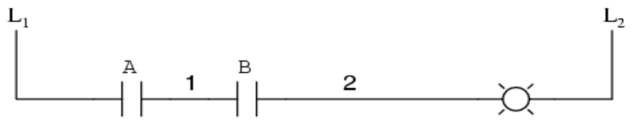
ON-OFF Outputs: They can also be interfaced using SET/RESET instructions	There are other ways to connect the digital outputs. Example: SET_BIT and RESET_BIT Output RESET Output SET "OUT" is boolean address of the output e.g. **%Q0.3** NOTE: In digital logic SET means to turn output HIGH and RESET means to turn output LOW
PWM (Pulse Width Modulated) Output: A digital output of the form of a square wave with a certain frequency. The on-time and off-time may be controlled. Ratio of on-time over total cycle time is called PWM duty. This is used to control speed of DC motors and change angular position of servo motors. It is not available on all digital outputs. And only dedicated outputs can perform this feature.	They are used with regular Output instructions but need to be configured on specified Output connections in advance of writing a Ladder program.
ON-OFF Memory Output: This special instruction is not exactly an output but simple a 1 bit memory location used to store an ON-OFF output variable.	We can use bit memories for the pulse-operated switch. A simple example will serve here to show you how to work with them 

Exercise 1: Implementing basic Logic Gates in Ladder Language

Ladder programming is a graphical programming language used for creating logic circuits. It is commonly used in industrial automation and control systems. Here is a brief tutorial on ladder programming:

1) Understand ladder logic elements:
   The basic elements of ladder programming are contacts, coils, timers, and counters. Contacts represent input signals, coils represent output signals, timers are used to create time delays, and counters are used to count the number of times an input signal occurs.
2) Create a ladder diagram:
   Ladder programming is based on creating a ladder diagram. The diagram consists of two vertical rails representing the power supply, and horizontal rungs representing the logic circuit. You can add contacts, coils, timers, and counters to the rungs by dragging and dropping them from a library.
3) Connect elements:
   Contacts and coils are connected using vertical lines called rails. Contacts are connected to the left rail and coils are connected to the right rail. When a contact is closed, it allows current to flow to the coil, which then activates the output signal.

Now create the following logic gates in Ladder language and verify that the given truth-table verifies your observations.



A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

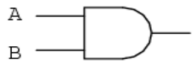
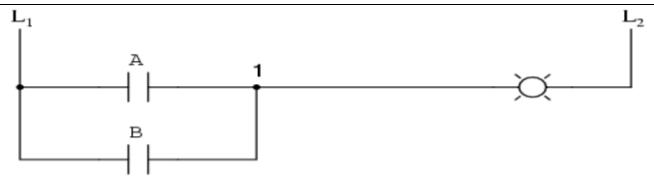
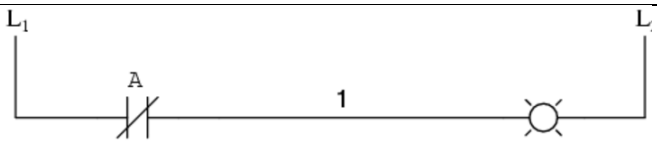


Fig.17 AND Logic Gate in Ladder Language. Take %I0.0 and %I0.2 as inputs and %Q0.0 as output



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

Fig.18 OR Logic Gate in Ladder Language. Take %I0.3 and %I0.4 as inputs and %Q0.1 as output



A	Output
0	1
1	0

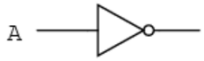


Fig.19 NOT Logic Gate in Ladder Language. Take %I0.5 as input and %Q0.2 as output

In order to make these logic gates, the following steps need to be taken:

1) Create a new project in TIA Portal and name it.
2) Go to "Project View" and select "Add New Device" from the Project Management pane
3) Add your PLC CPU (1214C DC/DC/DC) and add the correct signal board (AI/AQ)
4) Now a CPU_1 has been added to the Project Management pane, go into it and then go into Program Blocks
5) Inside Program Blocks, go to Main_OB1

In order to complete this exercise, detailed steps are shown in Lab 05 with pictorial representation. Please follow it and complete the following tables.

Table for recording output of AND Gate

%I0.0	%I0.2	%Q0.0

Table for recording output of NOT Gate

%I0.5	%Q0.2

Table for recording output of OR Gate

%I0.3	%I0.4	%Q0.1

Exercise 2: XOR Gate Application as a two-way switch

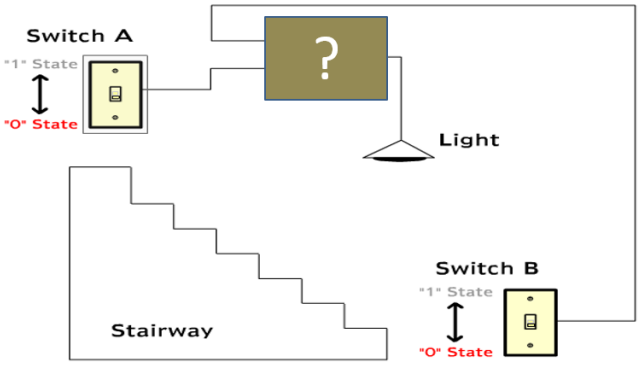
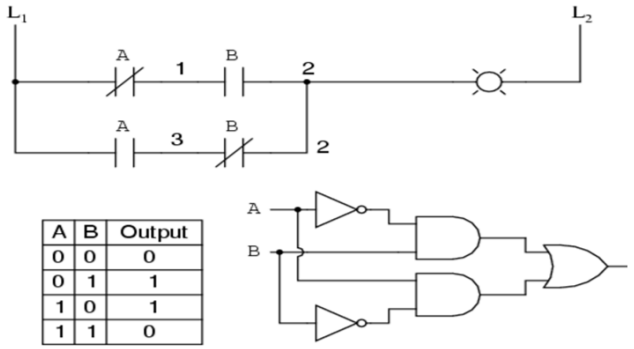


Fig. 20 Wiring problem for a staircase light with two switches. The light needs to turn ON or OFF from both the switches

Fig.20 demonstrates a common wiring challenge that allows a person going up stairs to switch the light ON and when he has reached the top stair, he can turn it OFF. The switch works in the same way for the downwards journey also. Conventionally, this problem is solved through a two-way switch, however, it can be easily solved with a XOR Gate also. Complete the following table by making a Ladder language XOR gate.



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

Fig.21 XOR gate implementation in Ladder language. Use %I0.6 and %I0.7 as inputs and %Q0.3 as output.

%I0.6	%I0.7	%Q0.3

Exercise 3: Operating a carwash "Enter" indicator using two proximity sensors

Modern carwash is an automated systems with many sensors and actuators. One basic sensor system in a carwash is an "Enter/Stop" indicator. It works using two proximity sensors interfaced to digital inputs as shown in Fig. 22.

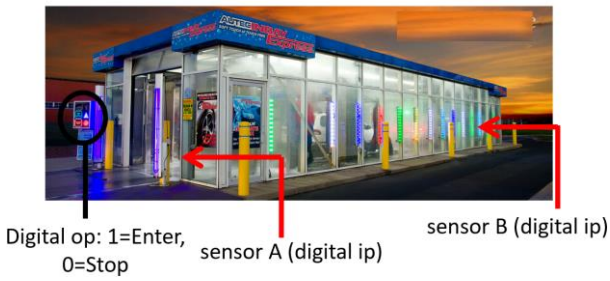


Fig. 22 "Enter/Stop" indicator outside a carwash. Note the two proximity sensors as the entrance and exit

How can we create a logic such that an entering car triggers and holds the indicator in ON state and when the car exits, the indicator turns OFF? Look at the following description and complete the truth-table.

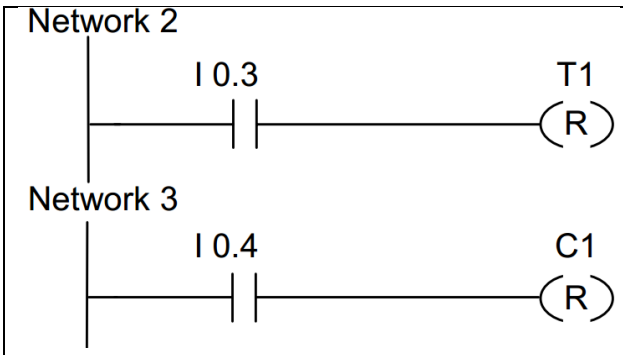


Fig.23 Solving the carwash indicator problem with SET/RESET output instructions. Complete this Ladder on your PLC and record observations in the accompanying table

	%I0.3 proximity sensor at entrance	%I0.4 proximity sensor at exit	%Q0.6 indicator
No car is inside			
Car enters, still inside			
Car exits			
No car is inside			

Exercise 4: Fluid Tank Filling

A fluid tank needs to be filled with some process fluid. It has two float switches sensing level at the bottom and top of the tank as shown in Fig. 24. The PLC needs to turn the motor ON to keep the fluid level maintained to the top level. However, another input is also provided to the on-site operator in the form of a manual over-ride switch which stops the motor at any point. A truth table for all three inputs and their impact on output is shown.

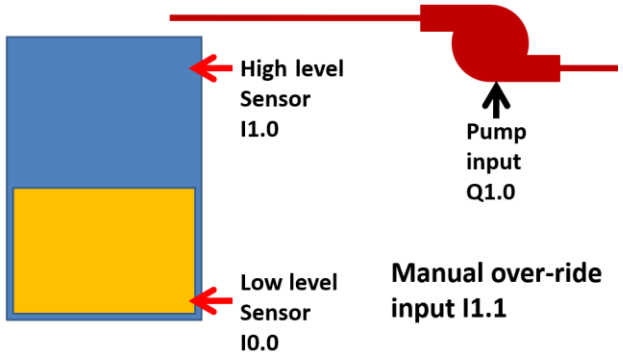


Fig.24 Fluid Tank Filing problem with three inputs and one output

Over-ride	Low level	High level	Output
I1.1	I0.0	I1.0	Q0.0
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

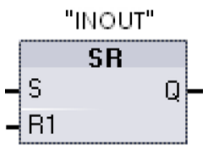
Write down the Boolean Logic expression for output Q0.0 in terms of the three inputs I1.1, I0.0 and I 1.0

Q0.0 = _____

Now draw your Ladder logic solution to this problem in the space below:

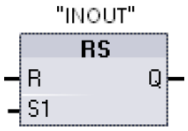
Task 1: Using Flip-Flop to solve the Fluid Tank Filling problem

Using the same inputs and outputs, now solve the Fluid Tank Filling problem with SR <u>or</u> RS flipflop. Make sure to attach your Ladder program and also shoe its output on a truth table. Comment on any undesired or unexpected outcome. SR and RS instructions are part of the SIEMENS PLC instruction set. A quick recap of how SR and RS flip-flops work is presented here.



S	R1	
0	0	Previous state
0	1	0
1	0	1
1	1	0

"INOUT" is a bit memory



S1	R	"INOUT" bit
0	0	Previous state
0	1	0
1	0	1
1	1	1

"INOUT" is a bit memory

Task 2: Creating a two-way switch system for corridor-lighting

Similar to staircase lighting, corridor lighting is also done conventionally through two-way switches. However, in industrial settings, PLC can be used to solve it with more control. Consider the following case where a long corridor with one entrance and one exit has three lights. For a person who could be walking into the corridor from either side, we need to give him control of the light just in from of him. The logic should work in such a way that when he turns on the first switch, the first light in front of him turns ON. When he approaches the second switch and flicks it, the next light turns ON while the previous one turns OFF. When he reaches the third switch and flicks it, the third and final light turns ON and the second light is turned. Finally, he leaves while flicking the last switch which turns the third light OFF. Make a Ladder program and truth table for this.

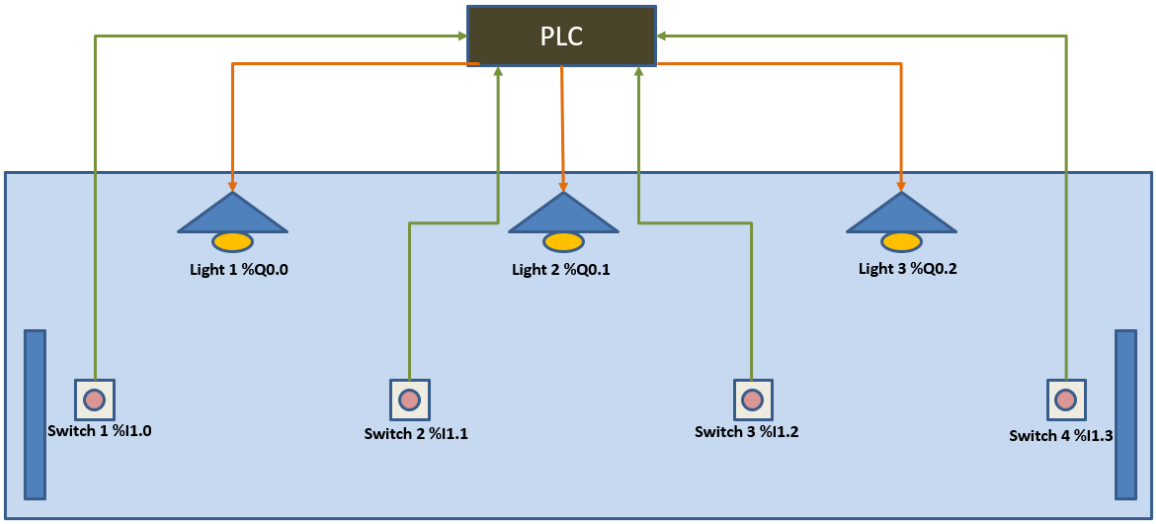


Fig.25 Lighting for corridor using PLC. Inputs and outputs are defined with switches and lights

Attach Ladder program along with truth-table with your answer.

NED University of Engineering & Technology
Department of Electrical Engineering

Course Code: **EE-375** Course Title: **Feedback Control Systems**
Laboratory Session No.: _____ Date: _____

Psychomotor Domain Assessment Rubric for Laboratory (Level P3)					
Skill(s) to be assessed	Extent of Achievement				
	0	1	2	3	4
Software Initialisation and Configuration: Set up and *recognise* software initialisation and configuration steps	Completely unable to recognise initialisation and configuration	Able to recognise initialisation but could not configure	Able to recognise initialisation but configuration is erroneous	Able to recognise initialisation and configuration with minimal errors	Able to recognise initialisation and configuration with complete success
Equipment Identification and Handling: *Sensory* skill to identify equipment and its components along with adherence to safe handling	Completely unable to identify equipment and components and no regard to safe handling	—	Ability to identify equipment but makes mistakes in recognising components, demonstrates decent equipment handling capacity	—	Ability to identify equipment and recognises all components, practices careful and safe handling
Establish and Verify Hardware-Software Connection: *Recognise* interface between computer and hardware kit and *establish* connectivity with software	Unable to perform hardware and software connection verification	—	Able to verify hardware connection but unable to establish software connection verification	—	Able to verify hardware connection and successfully establishes software connection verification
Following step-by-step procedure to complete lab work: *Observe, imitate and operate* hardware in conjunction with software to complete the provided sequence of steps	Inability to recognise and perform given lab procedures	Able to recognise given lab procedures and perform them but could not follow the prescribed order of steps	Able to recognise given lab procedures and perform them by following prescribed order of steps, with frequent mistakes	Able to recognise given lab procedures and perform them by following prescribed order of steps, with occasional mistakes	Able to recognise given lab procedures and perform them by following prescribed order of steps, with no mistakes

Psychomotor Domain Assessment Rubric for Laboratory (Level P3)					
Skill(s) to be assessed	Extent of Achievement				
	0	1	2	3	4
Programming the Controller for given Control System Problem: *Imitate* and *practice* given Ladder instructions for implementing specific control strategy and store required variables	Incorrect selection and use of programming constructs and instructions	Correct selection of programming constructs and instructions but their use is incorrect	Correct selection and use of programming constructs and instructions with many syntax/logical errors	Correct selection and use of programming constructs and instructions with little to no syntax/logical errors	Correct selection and use of programming constructs and instructions with no syntax/logical errors
Software Menu Identification and Usage: Ability to *operate* software environment *under supervision*, using menus, shortcuts, instructions etc.	Unable to understand and use software menu	Little ability and understanding of software menu operation, makes many mistake	Moderate ability and understanding of software menu operation, makes lesser mistakes	Reasonable understanding of software menu operation, makes no major mistakes	Demonstrates command over software menu usage with occasional use of advance menu options
Detecting and Removing Errors/Exceptions in Hardware and Software: *Detect* Errors/Exceptions and *manipulate,* under supervision, to rectify the Ladder program	Unable to check and detect error messages in software and hardware	Able to find error messages in software but no sense of hardware error identification	Able to find error messages in software and recognise them on hardware. Still unable to understand the error type and possible causes	Able to find error messages in software and recognise them on hardware. Moderately able in understanding error type and possible causes	Able to find error messages in software and recognise them on hardware. Reasonably able in understanding error type and possible causes
Visualisation, Comparison and analysis of results: *Copy* or enter results in analysis software to visualise and compare them with inputs. Use analysis tools to compute standard indices from result	Unable to understand and utilise visualisation, plotting and analysis software	Ability to understand and utilise visualisation and plotting instructions with errors. Unable to compute standard indices	Ability to understand and utilise visualisation and plotting instructions with occasional errors. Able to partially compute standard indices	Ability to understand and utilise visualisation and plotting instructions with no errors. Able to partially compute standard indices	Ability to understand and utilise visualisation and plotting instructions without errors. Able to compute standard indices completely

LAB SESSION 07

Objective:

Digital I/O manipulation in PLCs with timers, counters, and PWM (Pulse Width Modulation) generators for designing On-Off type feedback control systems using Ladder language.

Operating Modes of PLC CPU

The CPU has three modes of operation: STOP mode, STARTUP mode, and RUN mode. Status LEDs on the front of the CPU indicate the current mode of operation.

● In STOP mode, the CPU is not executing the program, and you can download a project. The RUN/STOP LED is solid yellow.

● In STARTUP mode, the CPU executes any startup logic (if present). The CPU does not process interrupt events during the startup mode. The RUN/STOP LED alternates flashing between green and yellow.

● In RUN mode, the scan cycle executes repeatedly. Interrupt events can occur and the CPU can process them at any point within the program cycle phase. You can download some parts of a project in RUN mode. The RUN/STOP LED is solid green.

The CPU supports the warm restart method for entering the RUN mode. Warm restart does not include a memory reset, but you can command a memory reset from TIA Portal. A memory reset clears all work memory, clears retentive and non-retentive memory areas, copies load memory to work memory, and sets outputs to the configured "Reaction to CPU STOP". A memory reset does not clear the diagnostics buffer or the permanently saved IP address. A warm restart initializes all non-retentive system and user data. We can change the CPU mode by connecting to the PLC and going online as depicted and described in Fig. 3

Brief overview of the Scan Cycle of SIEMENS S71200 CPU

Just like microcontrollers, PLCs also process instructions in two steps: 1) initialization of variables and inputs/outputs and 2) continuous loop that keeps running the user program in an infinite fashion – this infinite loop is called Scan Cycle or, simply, Cycle in SIEMENS PLCs. Each scan cycle includes writing the outputs, reading the inputs, executing the user program instructions, and performing system maintenance or background processing. The cycle is referred to as a scan cycle or scan. Under default conditions, all digital and analog I/O points are updated synchronously with the scan cycle using an internal memory area called the Process Image (See Fig.1). The process image contains a snapshot of the physical inputs and outputs on the CPU, signal board, and signal modules.

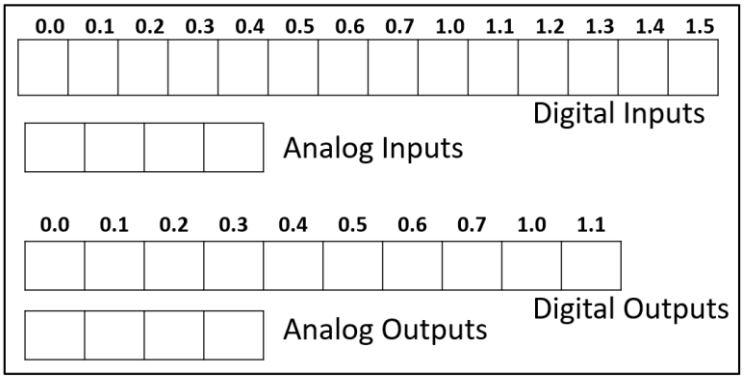


Fig.1 Process Image of CPU1214C. It stores the most recent I/O status in Memory and is used by the user program

The CPU reads the physical inputs just prior to the execution of the user program and stores the input values in the process image input area. This ensures that these values remain consistent throughout the execution of the user instructions.

The CPU executes the logic of the user instructions and updates the output values in the process image output area instead of writing to the actual physical outputs.

After executing the user program, the CPU writes the resulting outputs from the process image output area to the physical outputs. d outputs on the CPU, signal board, and signal modules This process provides consistent logic through the execution of the user instructions for a given cycle and prevents the flickering of physical output points that might change state multiple times in the process image output area.

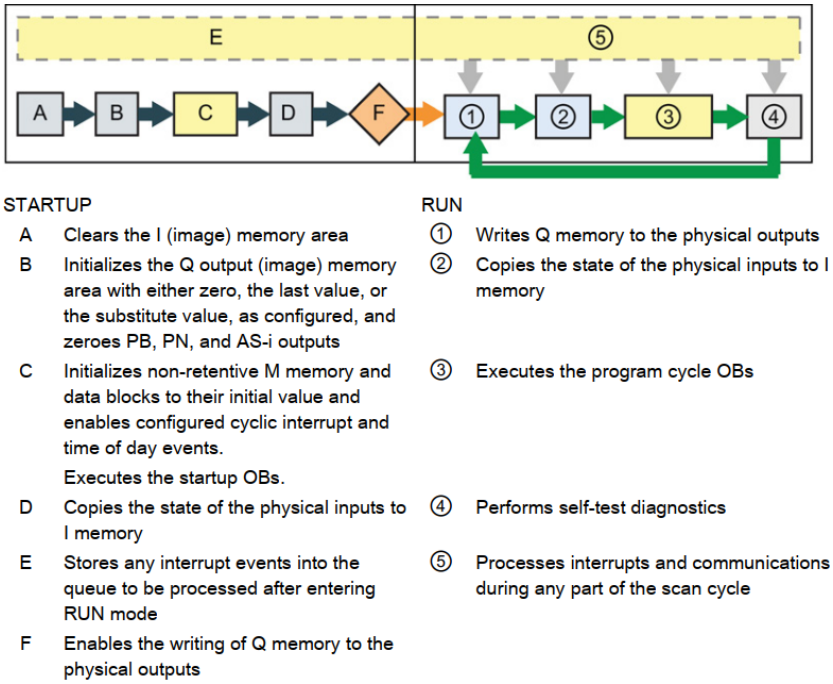


Fig.2 Scan Cycle or simply Scan of a SIEMENS S71200 CPU. STARTUP processes take place when the CPU is in the STOP mode. Once it starts and enter START mode, processes 1 through 5 are performed in a continuous loop.

While connected to the PLC, we can go to Online and Diagnostics and see the Scan Cycle duration, memory consumption and other useful characteristics of the current settings.

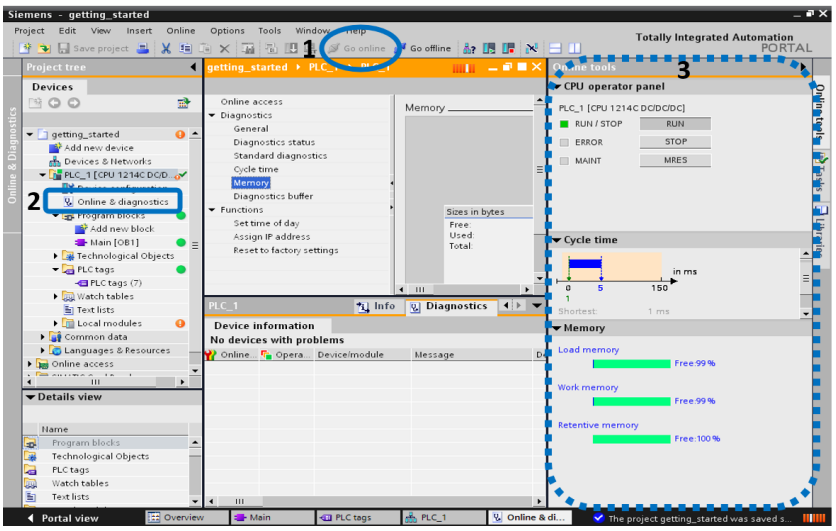


Fig. 3 While running a program, we can click on “Go Online” (highlight **1**), select “Online and Diagnostics” inside CPU_1 (highlight **2**) in order to see details related to Scan Cycle (highlight **3**). Note that CPU Modes can also be controlled from this pane

For each scan cycle, the CPU writes the outputs, reads the inputs, executes the user program, updates communication modules, and responds to user interrupt events and communication requests. Communication requests are handled periodically throughout the scan.

These actions (except for user interrupt events) are serviced regularly and in sequential order. User interrupt events that are enabled are serviced according to priority in the order in which they occur. For interrupt events, the CPU reads the inputs, executes the OB, and then writes the outputs, using the associated process image partition (PIP), if applicable. The system guarantees that the scan cycle will be completed in a time period called the maximum cycle time; otherwise, a time error event is generated. Please note that Interrupt and its handling are out of the scope of this lab session.

SIEMENS S71200 Memory Area, Addressing and Data Types

The CPU provides the following memory areas to store the user program, data, and configuration:

● Load memory is non-volatile storage for the user program, data and configuration. When a project is downloaded to the CPU, it is first stored in the Load memory area. This area is located either in a memory card (if present) or in the CPU. This non-volatile memory area is maintained through a power loss. You can increase the amount of load memory available for data logs by installing a memory card.

● Work memory is volatile storage for some elements of the user project while executing the user program. The CPU copies some elements of the project from load memory into work memory. This volatile area is lost when power is removed, and is restored by the CPU when power is restored.

● Retentive memory is non-volatile storage for a limited quantity of work memory values. The retentive memory area is used to store the values of selected user memory locations during power loss. When a power down or power loss occurs, the CPU restores these retentive values upon power up.

The Work Memory acts as the location for storing variables. In CPU1214C, this memory is 4kB (4096 Bytes). The interesting aspect of this memory is that it is **bit-addressable**. Fig. 4 shows the layout of work memory and gives examples for assigning bit, word and double word in this location.

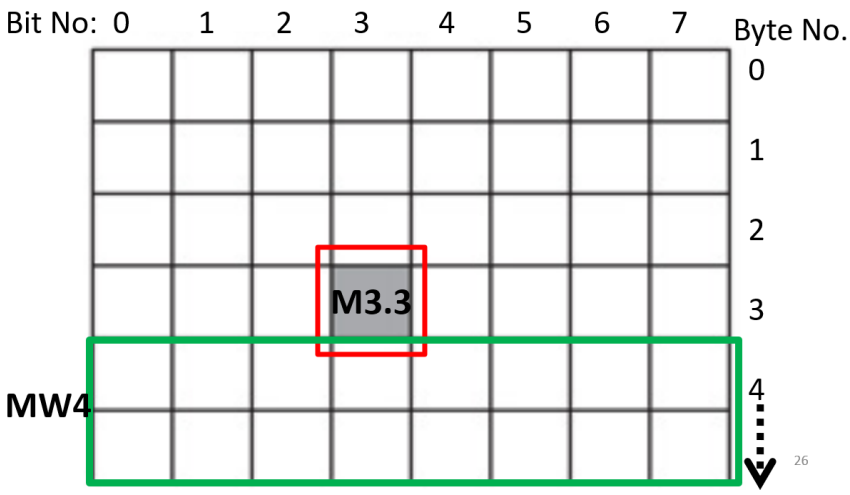


Fig.4 Work Memory in S71200 CPU1214C. It consists of 4096 Bytes with bit-addressable access. In this figure, M3.3 is a bit declared in the 3rd bit of the 3rd Byte. Whereas, MW4 is a Word (2 Bytes) starting at Byte 4 and also consists Byte 5 of the Work Memory

Data types are used to specify both the size of a data element as well as how the data are to be interpreted. Each instruction parameter supports at least one data type, and some parameters support multiple data types. Hold the cursor over the parameter field of an instruction to see which data types are supported for a given parameter.

Data Type	Bit Size	Number Range	Address Examples
Bool	1	TRUE, FALSE	%I0.1 (Input bit) %Q0.3 (Output bit) %M3.2 (Memory bit)

Byte	8	0 to 255	%IB1 (Input byte) %MB3 (Memory byte)
Word	16	0 to 65535	%MW4 (Memory Word) Consists of Byte 4, 5, 6 and 7
Double Word	32	0 to 4294967295	%MD7 (Memory Double Word) Consists of Bytes 7, 8, 9, 10, 11, 12, 13 and 14
Char	8	0 to 255	'A', 't', '@'

Table 1: Basic Data Types and their ranges and example syntax

In the following tables, more advanced data types are given along with their ranges and example values.

SInt (short integer)	8 bits (1 byte)	-128 to 127	123, -123
USInt (unsigned short integer)	8 bits (1 byte)	0 to 255	123
Int (integer)	16 bits (2 bytes)	-32,768 to 32,767	123, -123
UInt (unsigned integer)	16 bits (2 bytes)	0 to 65,535	123
DInt (double integer)	32 bits (4 bytes)	-2,147,483,648 to 2,147,483,647	123, -123

Table 2: Advance data-types

UDInt (unsigned double integer)	32 bits (4 bytes)	0 to 4,294,967,295	123
Real (real or floating point)	32 bits (4 bytes)	+/-1.18 x 10^{-38} to +/-3.40 x 10^{38}	123.456, -3.4, -1.2E+12, 3.4E-3
LReal (long real)	64 bits (8 bytes)	+/-2.23 x 10^{-308} to +/-1.79 x 10^{308}	12345.123456789 -1.2E+40
Time (time)	32 bits (4 bytes)	T#-24d_20h_31m_23s_648ms to T#24d_20h_31m_23s_647ms Stored as: -2,147,483,648 ms to +2,147,483,647 ms	T#5m_30s 5#-2d T#1d_2h_15m_30x_45ms
String (character string)	Variable	0 to 254 byte-size characters	'ABC'
DTL[1] (date and time long)	12 bytes	Minimum: DTL#1970-01-01-00:00:00.0 Maximum: DTL#2554-12-31-23:59:59.999 999 999	DTL#2008-12-16-20:30:20.250

Table 3: More advance data-types

CPU Process Control and Online Monitoring of I/O

Once a PLC program is downloaded to the CPU we can connect to it in real-time and perform diagnostics and debugging. This is done by connecting to the PLC via "Go Online" option. Please make sure that you have the same project filed opened as that of the program downloaded to the PLC. In this way, we can monitor all the inputs and outputs in real-time and thus can debug the program in an effective manner. Note that in Online mode, the input and output states are being transmitted by the CPU and thus are accurate. If any physical input or output is not responding in accordance with the activity reported by "Go Online" option then the physical connections to inputs/outputs must

be inspected. Moreover, remaining online also allows you to inspect the PLC Tags – a collection of all PLC variables with values shown in real-time.

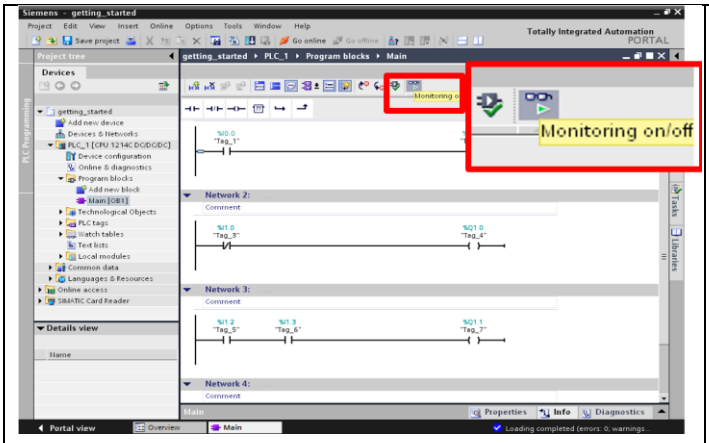


Fig.5 Online Monitoring can be turned on by connecting to the PLC with the same project open in TIA Portal as that downloaded to the PLC and clicking on Monitoring

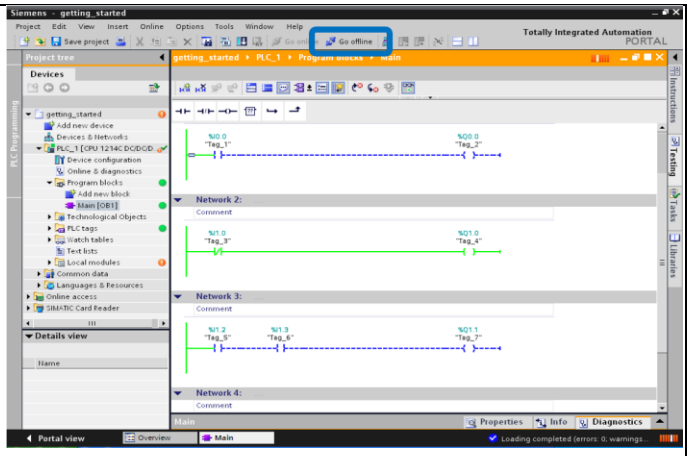


Fig.6 Once Online, the connected circuits/outputs turn green and the disconnected ones are shown with a dotted blue line. To turn off monitoring, click on "Go Offline"

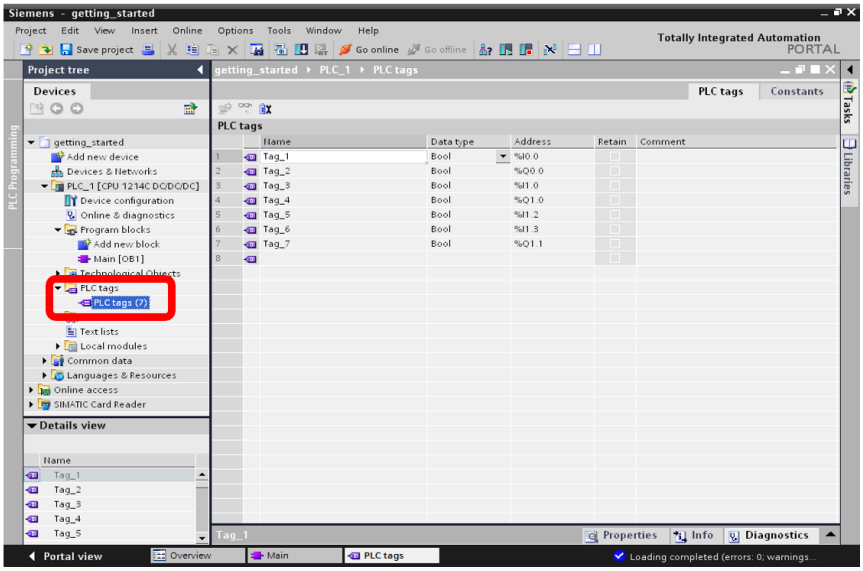


Fig.7: PLC Tags, present in the PLC_1 Device in the Project Tree can be used to monitor all the variable live.

Exercise 1: Complete Task 01 of Lab 06 and observe its output using Online Monitoring and Watch-table.

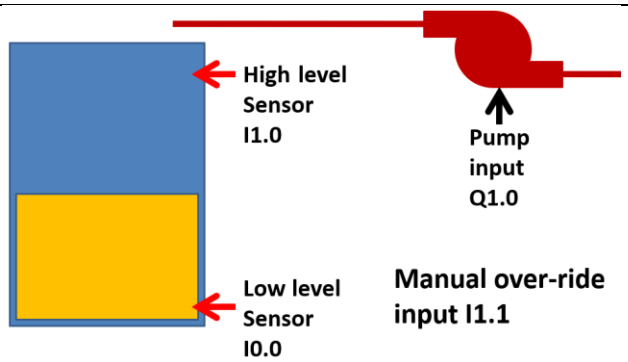


Fig.8 Fluid Tank Filing problem – use online monitoring and watch table to complete the accompanying table

Over-ride	Low level	High level	Output
I1.1	I0.0	I1.0	Q0.0
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

PLC Timers and their usage

PLC Timer is a block used to create delays for turning on inputs, turning off inputs, or simply run an appliance for a prescribed amount of time. In SIEMENS PLCs, three types of timer instructions are present:

1) On delay timer
2) Off delay timer
3) Pulse on timer

Before explaining these timer types, we discuss two timer connections that are present on each type of timer. These connections are PT (Preset Time) and ET (Elapsed Time). Both connections require a variable and value to be assigned before the timer could be used. These two connections are described in detail here.

a) PT (Preset Time): Pre- programmed time of operation. It is given as a constant of data type 'Time' in hours, minutes and seconds. It's value gives the timer a reference/threshold value of time.
b) ET(Elapsed Time): Elapsed Time is a variable that is initially zero but starts to increase in a linear fashion once the timer input is activated. Once the ET value equals PT, the timer stops its working and the output is changed.

Syntax for writing PT is elaborated through these examples:

PT value of 5000 milli-seconds: **T#5000ms**

PT value of 1 minute: **T#1m**

PT value of 2 seconds: **T#2s**

PT value of 1 hour: **T#1h**

Now, we shall quickly understand the operation of the three types of timers.

On Delay Timer

An On Delay Timer (ODT) in a PLC works by introducing a time delay between the occurrence of an input condition and the activation of an output device or action. When the input condition is met, the ODT starts counting down from its preset time, which is typically specified in milliseconds, seconds, or minutes. During the delay period, the ODT's output remains off or inactive, regardless of the input condition. Once the preset time has elapsed, the ODT's output is activated, allowing the connected output device or action to be triggered.

It is important to note that the ODT's output will stay active as long as the input condition is present. Once the input condition is no longer met, the ODT's output will deactivate, and the timer will reset to its initial state, ready to start a new delay cycle when the input condition occurs again.

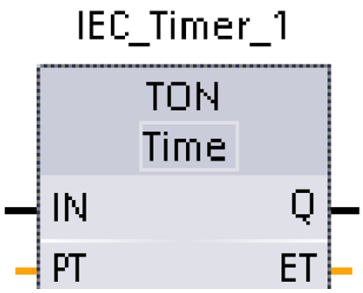


Fig. 9 On Delay Timer instruction in TIA Portal

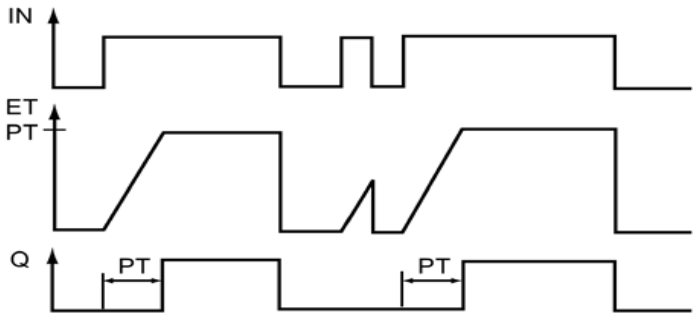


Fig.10 Timing diagram relating IN, ET and Q. Note that PT is a fixed value

Off Delay Timer

An Off Delay Timer (ODT) in a PLC functions by introducing a time delay between the deactivation of an input condition and the deactivation of an output device or action. When the input condition is deactivated, the ODT starts counting down from its preset time, which is typically specified in milliseconds, seconds, or minutes. During the delay period, the ODT's output remains active, even if the input condition is no longer present. This ensures that the output device or action continues to operate for the specified duration. Once the preset time has elapsed, the ODT's output deactivates, ceasing the operation of the connected output device or action.

It is important to note that if the input condition is reactivated during the delay period, the ODT's countdown will reset, effectively extending the operation of the output device until the input condition remains inactive for the entire preset time.

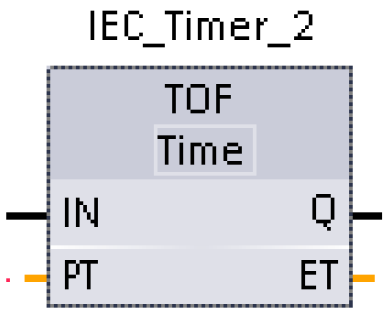


Fig. 11 Off Delay Timer instruction in TIA Portal

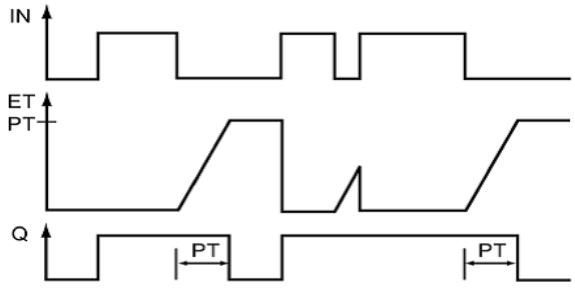


Fig.12 Timing diagram relating IN, ET and Q. Note that PT is a fixed value

Pulse Timer

A Pulse Timer in a PLC operates by generating a fixed-duration pulse or signal upon the occurrence of an input condition. When the input condition is met, the Pulse Timer immediately starts its countdown from the preset time, which is typically specified in milliseconds or seconds. During the countdown, the Pulse Timer activates its output, producing a pulse signal with a fixed duration. Once the preset time elapses, the Pulse Timer deactivates its output, ending the pulse signal.

It's important to note that the Pulse Timer does not reset automatically. To generate subsequent pulses, the input condition needs to be deactivated and reactivated, triggering the Pulse Timer to start a new countdown and generate another pulse upon completion.

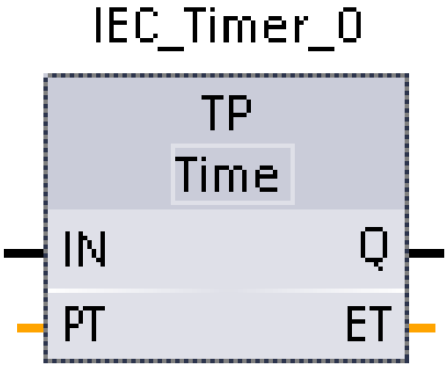


Fig. 13 Off Delay Timer instruction in TIA Portal

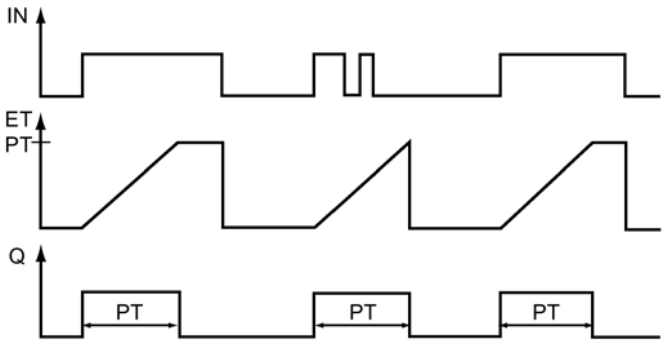


Fig.14 Timing diagram relating IN, ET and Q. Note that PT is a fixed value

PLC Counters and their usage

Counters are sequential logic elements which can increase/decrease a variable value w.r.t. the change in input. A PLC counter is a component that tracks the occurrence of a specific input condition or event. It operates by configuring its parameters, including the counting mode (up or down), initial value, and storage address for the current count value. The counter incrementally or decrementally changes its value based on the configured counting mode when the input

condition is met. The current count value is stored for further processing or display, and the counter's output can be used to trigger actions or events at specific count values. Additionally, counters often have a reset functionality to return the count value to its initial state. By utilizing counters, PLCs can accurately monitor and control events, cycles, or processes in industrial applications, enabling precise automation.

In SIEMENS TIA Portal, we have three types of counters:

1) Up Counter
2) Down Counter
3) Up-Down Counter

Up Counter

An up counter in a PLC operates by incrementing its count value each time a specified input condition or event occurs. Upon meeting the input condition, the up counter increases its count value by one, progressing towards the desired count target. The up counter's count value is typically stored in a designated memory address for further processing or display purposes. The counting operation of an up counter is continuous, and it keeps increasing the count value as long as the input condition is met.

Up counters are commonly used to track the number of cycles, events, or objects in industrial processes, enabling accurate monitoring and control of operations.

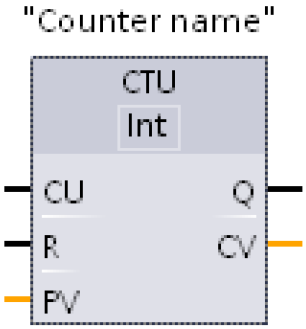


Fig. 15 Up Counter instruction in TIA Portal

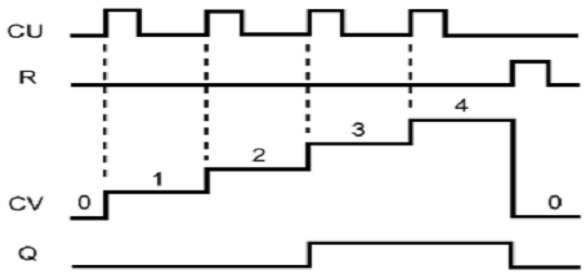


Fig.16 Timing diagram relating CU(Count Up),R(Reset) CV(Count Value) and Q(Output). Note that PV (Preset Value) is 3 here.

Down Counter

A down counter in a PLC operates by decrementing its count value each time a specified input condition or event occurs. When the input condition is met, the down counter reduces its count value by one, moving towards the desired count target. The count value of the down counter is typically stored in a designated memory address for further processing or display. The counting operation of a down counter continues until the count value reaches zero or a specified stopping point.

Down counters are commonly used in applications such as countdown timers, where they track the remaining time or number of cycles until a specific event or action occurs, enabling precise timing and control in industrial processes.

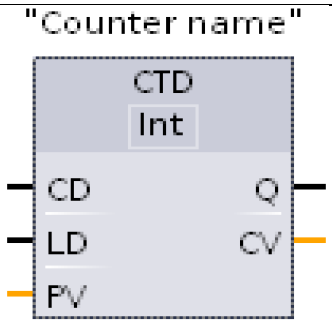


Fig. 17 Down Counter instruction in TIA Portal

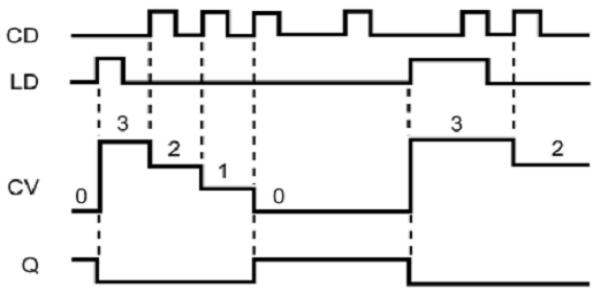


	Fig.18 Timing diagram relating CD(Count Down),L(Load) CV(Count Value) and Q(Output). Note that CV (Count Value) is 3 here.

Up-Down Counter

An up-down counter in a PLC combines the functionality of both an up counter and a down counter, allowing it to increment or decrement its count value based on specified input conditions or events. The up-down counter can be configured to increase its count value when one input condition is met and decrease the count value when another input condition is met. The count value of the up-down counter is typically stored in a designated memory address and can be used for further processing or display purposes. The up-down counter's counting operation is bidirectional, allowing it to track both positive and negative changes in the count value.

Up-down counters are commonly used in applications where the count value needs to be adjusted in both directions, such as keeping track of the net flow of items in a manufacturing process or controlling position movements in automated systems.

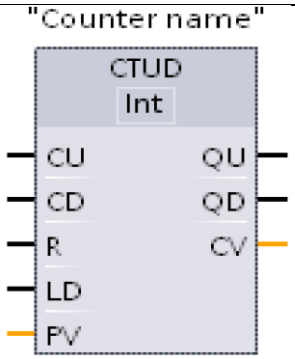


Fig. 19 Up-Down Counter instruction in TIA Portal

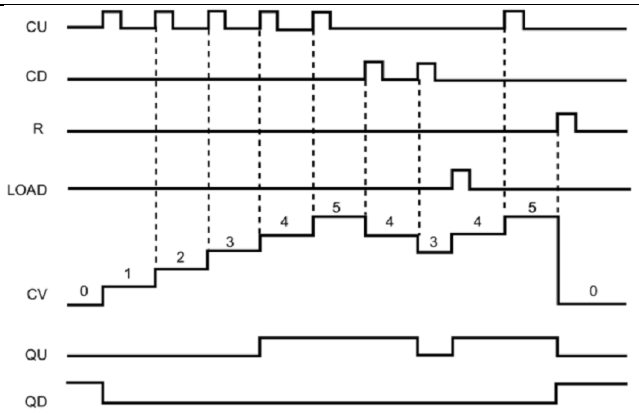


Fig.20 Timing diagram relating CU(Cunt Up), CD(Count Down), R(Reset), L(Load), CV(Count Value), QU(Up Output) and QD (Dow Output).

Exercise 2: In TIA Portal open a new project and connect all three timers and all three counters in separate networks.

Connect all timers with separate input switches and also attach separate outputs to them. Don't forget to add Preset Time to each of them and then observe their operation.

Connect all counters with input switches on their CU, CD, L, and R inputs. Timer outputs Q, QU and QD should be connected with outputs. Don't forget to assign PV(Preset Value) and attach a Memory Word (%MW) variable to their respective CV outputs.

NOTE: All MW variables must be separate and must not overlap in the Work Memory

Exercise 3: Conveyor-Belt Tablet Filling example part-1: Detecting Bottles and Counting Tablets, separately

Consider the following conveyor system with a bottle proximity sensor and a motor that runs the conveyor. Design a Ladder program such that motor runs until a bottle comes in front of bottle detector. As soon as the bottle is detected, conveyor stops and after a delay of 2 seconds it starts running again.

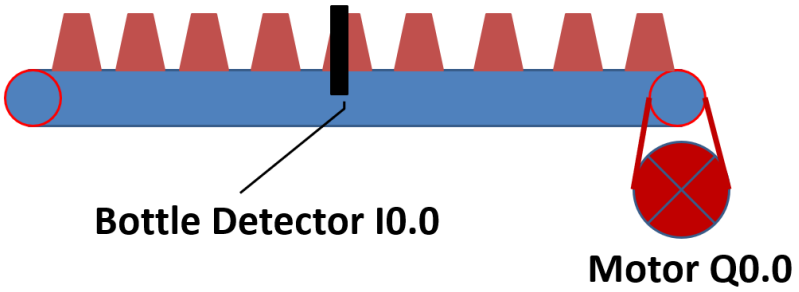


Fig.21 Set-up and I/O connections for Conveyor-Belt Tablet Filling example part-1

Daw your Ladder network here:

Now, separately consider a tablet dispenser filled with tablets. The dispenser is placed just over the bottle that has been stopped by the bottle detector. Create a Ladder network that opens the tablet dispenser valve, tablet sensor counts 10 tablets before closing the valve again.

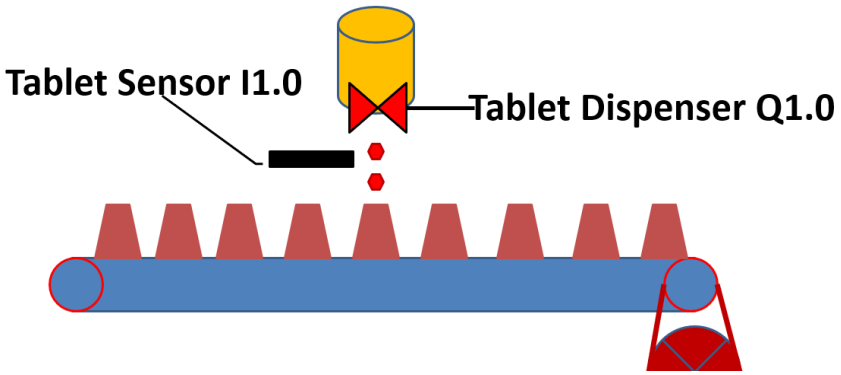


Fig.22 Tablet counting for Conveyor-Belt Tablet Filling example part-1

Draw your Ladder network here, for the tablet counting part.

Task 1: Conveyor-Belt Tablet Filling example part-2: Detecting Bottles and Counting Tablets

In this task we need to combine both tablet filling and bottle detection operations. The sequence goes like this: The conveyor runs until the bottle detector detects a bottle in front of it. At this point conveyor stops. After a delay of 1 second, the tablet dispenser valve opens to fill the bottle. The valve is closed only after counting 10 tablets. At this point a delay of 1 s takes place and the conveyor starts running again. The cycle repeats afterwards.

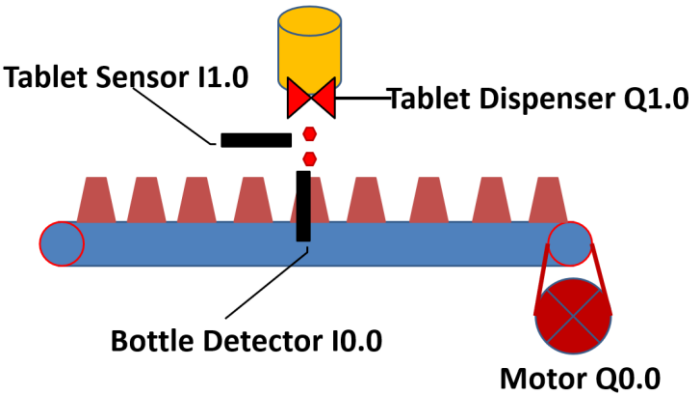


Fig.23 Complete layout for Conveyor-Belt Tablet Filling example part-2

Task 2: To start asynchronous wound motors, resistors are connected in the rotor circuit to avoid a high inrush current. After pushing the start button S1 (connected to %I0.3), the main relay (K1) is closed. Then relays K2 (connected to %Q0.0), K3 (%Q0.1) and K4 (%Q0.2) are closed, each after a time delay of 5 seconds. Write the program to start the motor M3 in this sequence. Note that M3 would not e connected directly to the PLC.

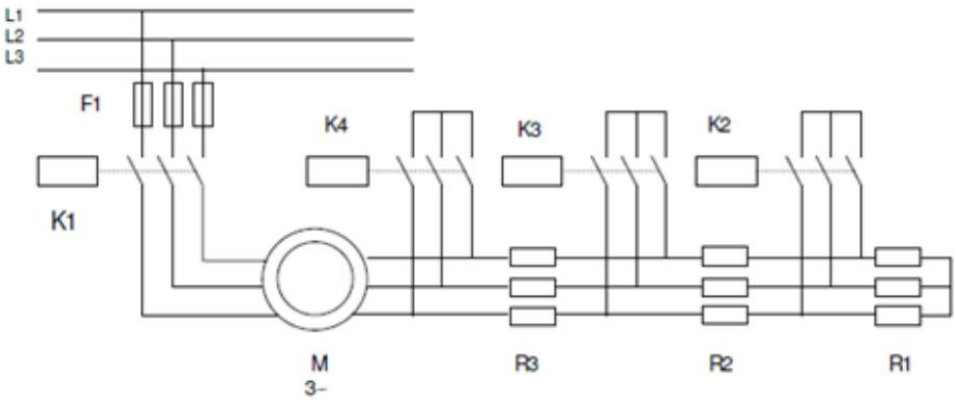


Fig.24 Asynchronous wound motor startup using time delayed relays

NED University of Engineering & Technology
Department of Electrical Engineering

Course Code: **EE-375** Course Title: **Feedback Control Systems**
Laboratory Session No.: _____ Date: _____

Psychomotor Domain Assessment Rubric for Laboratory (Level P3)					
Skill(s) to be assessed	Extent of Achievement				
	0	1	2	3	4
Software Initialisation and Configuration: Set up and *recognise* software initialisation and configuration steps	Completely unable to recognise initialisation and configuration	Able to recognise initialisation but could not configure	Able to recognise initialisation but configuration is erroneous	Able to recognise initialisation and configuration with minimal errors	Able to recognise initialisation and configuration with complete success
Equipment Identification and Handling: *Sensory* skill to identify equipment and its components along with adherence to safe handling	Completely unable to identify equipment and components and no regard to safe handling	—	Ability to identify equipment but makes mistakes in recognising components, demonstrates decent equipment handling capacity	—	Ability to identify equipment and recognises all components, practices careful and safe handling
Establish and Verify Hardware-Software Connection: *Recognise* interface between computer and hardware kit and *establish* connectivity with software	Unable to perform hardware and software connection verification	—	Able to verify hardware connection but unable to establish software connection verification	—	Able to verify hardware connection and successfully establishes software connection verification
Following step-by-step procedure to complete lab work: *Observe, imitate and operate* hardware in conjunction with software to complete the provided sequence of steps	Inability to recognise and perform given lab procedures	Able to recognise given lab procedures and perform them but could not follow the prescribed order of steps	Able to recognise given lab procedures and perform them by following prescribed order of steps, with frequent mistakes	Able to recognise given lab procedures and perform them by following prescribed order of steps, with occasional mistakes	Able to recognise given lab procedures and perform them by following prescribed order of steps, with no mistakes

Psychomotor Domain Assessment Rubric for Laboratory (Level P3)					
Skill(s) to be assessed	Extent of Achievement				
	0	1	2	3	4
Programming the Controller for given Control System Problem: *Imitate* and *practice* given Ladder instructions for implementing specific control strategy and store required variables	Incorrect selection and use of programming constructs and instructions	Correct selection of programming constructs and instructions but their use is incorrect	Correct selection and use of programming constructs and instructions with many syntax/logical errors	Correct selection and use of programming constructs and instructions with little to no syntax/logical errors	Correct selection and use of programming constructs and instructions with no syntax/logical errors
Software Menu Identification and Usage: Ability to *operate* software environment *under supervision*, using menus, shortcuts, instructions etc.	Unable to understand and use software menu	Little ability and understanding of software menu operation, makes many mistake	Moderate ability and understanding of software menu operation, makes lesser mistakes	Reasonable understanding of software menu operation, makes no major mistakes	Demonstrates command over software menu usage with occasional use of advance menu options
Detecting and Removing Errors/Exceptions in Hardware and Software: *Detect* Errors/Exceptions and *manipulate,* under supervision, to rectify the Ladder program	Unable to check and detect error messages in software and hardware	Able to find error messages in software but no sense of hardware error identification	Able to find error messages in software and recognise them on hardware. Still unable to understand the error type and possible causes	Able to find error messages in software and recognise them on hardware. Moderately able in understanding error type and possible causes	Able to find error messages in software and recognise them on hardware. Reasonably able in understanding error type and possible causes
Visualisation, Comparison and analysis of results: *Copy* or enter results in analysis software to visualise and compare them with inputs. Use analysis tools to compute standard indices from result	Unable to understand and utilise visualisation, plotting and analysis software	Ability to understand and utilise visualisation and plotting instructions with errors. Unable to compute standard indices	Ability to understand and utilise visualisation and plotting instructions with occasional errors. Able to partially compute standard indices	Ability to understand and utilise visualisation and plotting instructions with no errors. Able to partially compute standard indices	Ability to understand and utilise visualisation and plotting instructions without errors. Able to compute standard indices completely

LAB SESSION 08

Objective:

Analog I/O interfacing and manipulation in PLCs, their application in sensing transducer outputs and transmitting signals to actuators.

Analog I/O present on the S71200 PLC Trainer

The Siemens S7-1200 PLC Trainer is a compact programmable logic controller (PLC) commonly used for industrial automation and control applications. The S7-1200 series offers various models, and the availability of analog input/output (I/O) modules may vary depending on the specific model and configuration.

Typically, the S7-1200 series PLC trainers have analog I/O capabilities through dedicated analog input and output modules. These modules can be added to the PLC to expand its functionality and support analog signals.

Here are some common analog I/O modules that can be used with the S7-1200 series:

- SM 1231 AI: This module provides analog input channels for measuring voltage and current signals. It supports a range of voltage and current inputs, such as 0-10V and 4-20mA.
- SM 1232 AO: This module offers analog output channels for generating analog signals. It can provide voltage or current outputs, depending on the configuration.
- SM 1234 AI/AO: This module combines both analog input and output channels. It allows you to measure analog signals and generate analog signals simultaneously.

These modules can be connected to the S7-1200 PLC using the onboard expansion port. In our trainer, we have the SM 1234 AI/AO module that comprises of 4 analog inputs (13-bit ADC, each) and 2 analog outputs (14-bit DAC, each).

General information	
Product type designation	SM 1234, AI 4x13 bit/AQ 2x14 bit
Supply voltage	
Rated value (DC)	24 V
Input current	
Current consumption, typ.	60 mA
from backplane bus 5 V DC, typ.	80 mA
Power loss	
Power loss, typ.	2 W
Analog inputs	
Number of analog inputs	4; Current or voltage differential inputs
permissible input voltage for voltage input (destruction limit), max.	35 V
permissible input current for current input (destruction limit), max.	40 mA
Cycle time (all channels) max.	625 µs
Input ranges	
• Voltage	Yes; ±10V, ±5V, ±2.5V
• Current	Yes; 4 to 20 mA, 0 to 20 mA
• Thermocouple	No
• Resistance thermometer	No
• Resistance	No
Input ranges (rated values), voltages	
• -10 V to +10 V	Yes
— Input resistance (-10 V to +10 V)	≥9 MOhm
• -2.5 V to +2.5 V	Yes
— Input resistance (-2.5 V to +2.5 V)	≥9 MOhm
• -5 V to +5 V	Yes
— Input resistance (-5 V to +5 V)	≥9 MOhm
Input ranges (rated values), currents	
• 0 to 20 mA	Yes
— Input resistance (0 to 20 mA)	280 Ω
• 4 mA to 20 mA	Yes
Analog outputs	
Number of analog outputs	2; Current or voltage
Output ranges, voltage	
• -10 V to +10 V	Yes
Output ranges, current	
• 0 to 20 mA	Yes

Fig.1 Specifications of SM 1234 AI/AO. Note that inputs and outputs can be configured for current or voltage

In this lab we shall use the potentiometer available on the trainer, which is connected to analog channel 0 (AI0) of SM 1234 to read analog voltage. Later, we shall use the digital voltmeter available on the trainer, which is connected to analog output channel 0 (AQ0) of SM 1234 to observe voltages generated by the analog output.

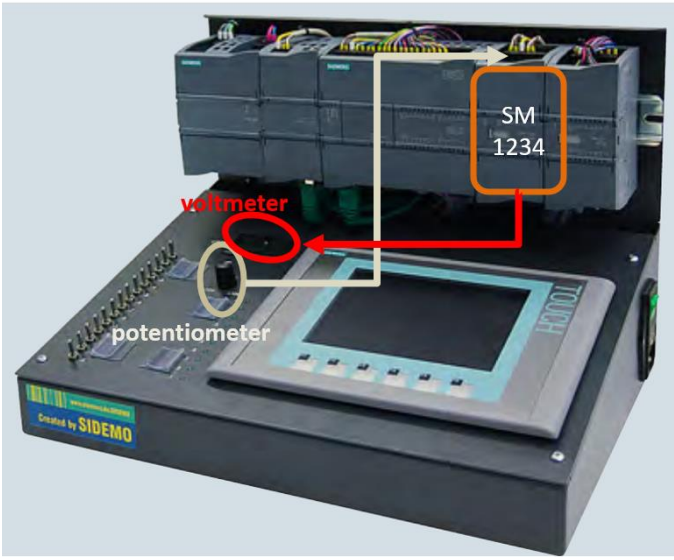


Fig.2 Analog input from potentiometer connected to AI0 of SM1234, whereas, the output AQ0 goes to voltmeter

Types and Ranges of Analogue Inputs available on PLCs

PLCs (Programmable Logic Controllers) support various types and ranges of analog inputs, depending on the specific model and manufacturer. Here are some common types of analog inputs found in PLCs:

1. Voltage Inputs (analog voltage): PLCs often include analog input channels that can measure voltage signals. The voltage ranges supported can vary, but common ranges include 0-10V, -10V to +10V, and 0-5V.
2. Current Inputs (analog current): Some PLCs provide analog input channels capable of measuring current signals. The current ranges supported may include 4-20mA, 0-20mA, or other ranges depending on the PLC model.
3. Resistance Inputs: PLCs can have analog inputs designed to measure resistance. These inputs are often used for applications such as temperature measurement using resistance temperature detectors (RTDs) or thermistors. The input range for resistance inputs can vary depending on the specific PLC and the type of resistance sensor being used.
4. Frequency Inputs: Certain PLCs have analog inputs capable of measuring frequency signals. These inputs can be used for applications such as reading pulses from flow meters or other devices that provide frequency output. The supported frequency range is typically specified by the manufacturer.
5. Thermocouple Inputs: Some PLCs offer analog inputs specifically designed for thermocouples. Thermocouples are temperature sensors that generate small voltage signals proportional to the temperature being measured. PLCs with thermocouple inputs typically support common thermocouple types such as J, K, T, etc.

It's important to note that the supported ranges and types of analog inputs can vary significantly depending on the specific PLC model and manufacturer. For example, in SIEMENS S71200 CPU 1214C DC/DC/DC, analogue inputs can be configured as either voltage or current. On voltage setting, its range can be programmed for 10 volt to -10 volt operation, 0 to 10 volt operation or 0 to 5 volt operation. On the other hand, if current setting is used, the analog input can be configured to 0 to 20 mA or 4 to 20 mA setting.

In this lab we shall not discuss frequency inputs but the CPU 1214C does have high speed counter inputs which are used for frequency measurement.

Types and Ranges of Analogue Outputs available on PLCs

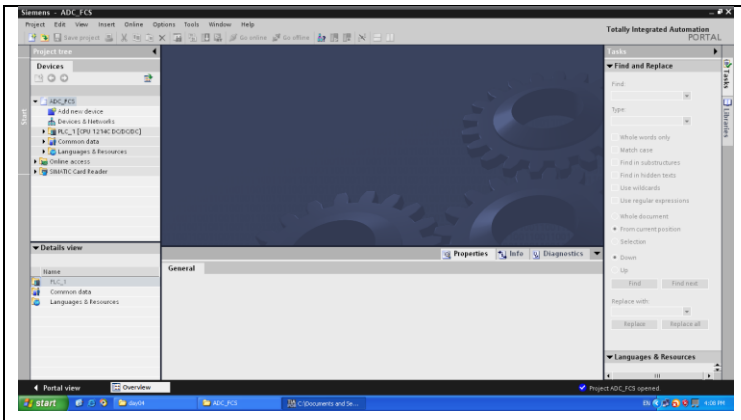
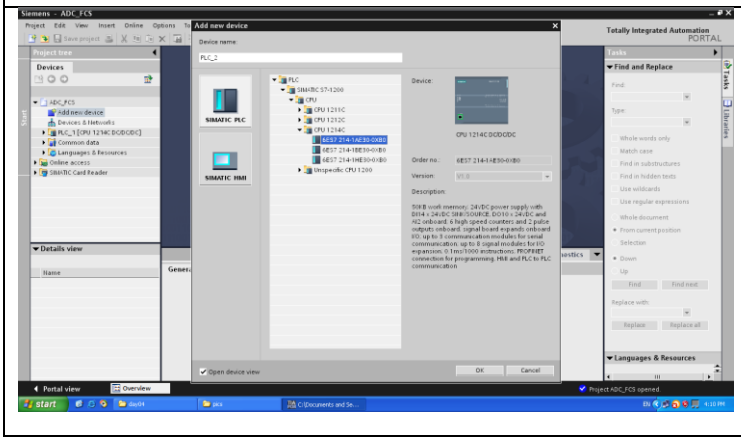
PLCs (Programmable Logic Controllers) support various types and ranges of analog outputs, depending on the specific model and manufacturer. Here are some common types of analog outputs found in PLCs:

1. Voltage Outputs (analog voltage): PLCs often include analog output channels that can generate voltage signals. The voltage ranges supported can vary, but common ranges include 0-10V, -10V to +10V, and 0-5V.
2. Current Outputs (analog current): Some PLCs provide analog output channels capable of generating current signals. The current ranges supported may include 4-20mA, 0-20mA, or other ranges depending on the PLC model.
3. Pulse Width Modulation (PWM) Outputs: Though they are not categorized strictly as analogue, certain PLCs offer analog outputs that utilize pulse width modulation. PWM outputs generate square wave signals with varying duty cycles, allowing control over the average voltage or current. PWM outputs are commonly used for controlling motors, valves, or other devices.
4. Resistance Outputs: PLCs can have analog outputs designed to control resistance. These outputs are often used for applications such as controlling variable resistors, heaters, or other devices that require resistance control.
5. Frequency Outputs: Some PLCs provide analog outputs capable of generating frequency signals. These outputs can be used for applications such as controlling variable frequency drives (VFDs) or generating pulse signals for specific devices.

In SIEMENS S71200 CPU 1214C DC/DC/DC, analogue outputs can be configured as either voltage or current. On voltage setting, its range can be programmed for 10 to -10 volt operation, 0 to 10 volt operation or 0 to 5 volt operation. On the other hand, if current setting is used, the analog input can be configured to 0 to 20 mA or 4 to 20 mA setting. PWM output is also present in CPU1214C which won't be covered in this lab, however, it must be clarified that it is not strictly a digital output.

Exercise 1: Interfacing Potentiometer on the PLC Trainer with Analogue Input present on the AI/AO module

As already described in Fig. 2, the potentiometer on the trainer is connected to SM 1234 module. Let's configure it in TIA Portal and read the analog voltage value through the PLC.

	Fig.3 Create a new project
	Fig.4 Add New Device from the Project Tree and select SIEMENS PLC and then the CPU1214C

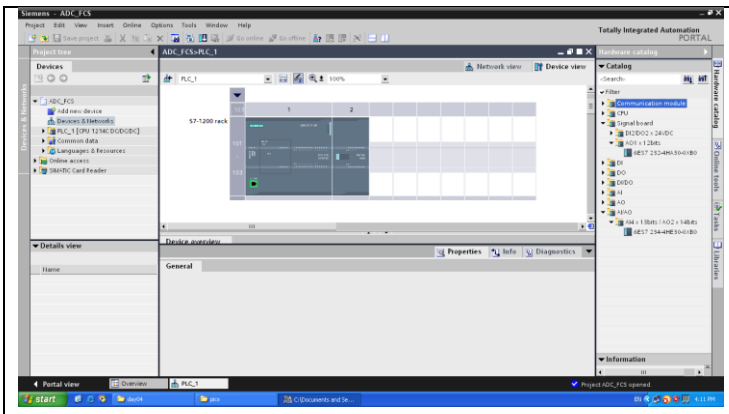


Fig.5 From the Catalog pane on the right, select signal board (AQ0) and analogue module (SM1234) and drag them to the PLC rack

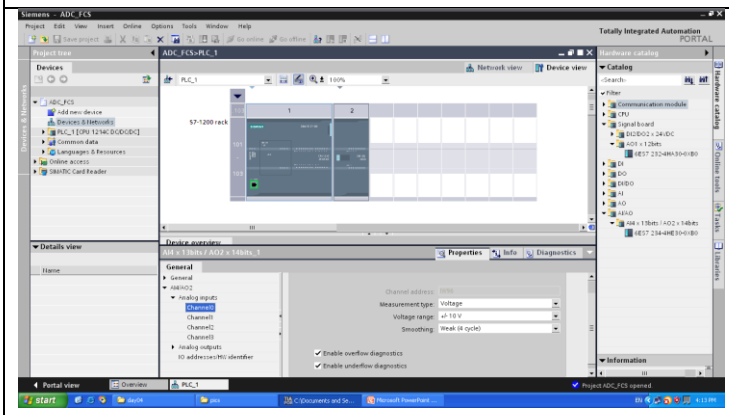


Fig.6 Double click on the CPU. In the Device Overview section (bottom) go to Analog Inputs and select Channel 0. This is where the potentiometer is connected. Don't change any settings but do explore all the features. Note that the address for Channel 0 is %IW96

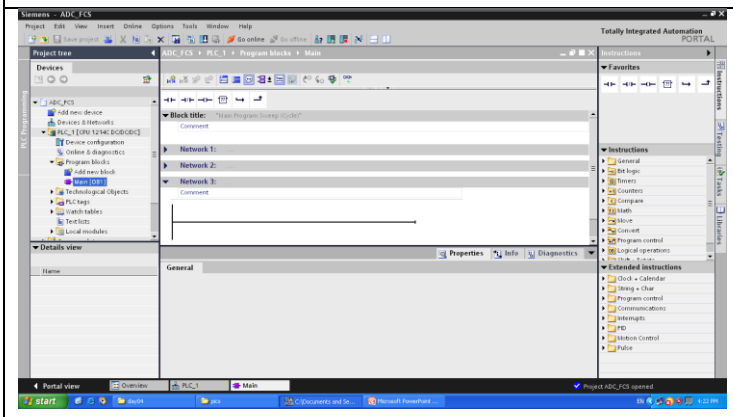


Fig.7 Now, go to Main_OB1 inside CPU_1 (from Project Tree). Here start your Ladder network

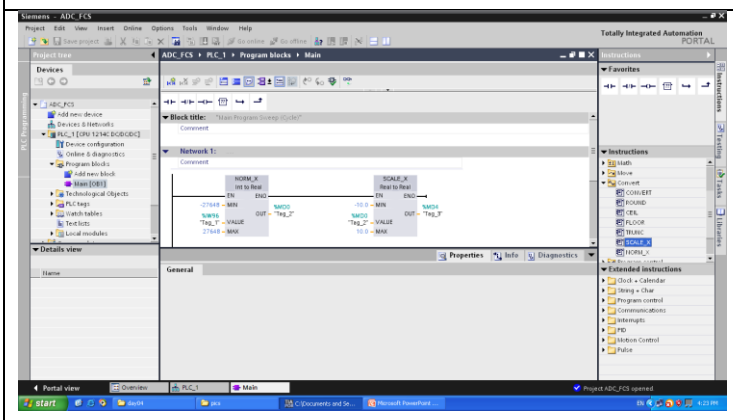


Fig. 8 For reading ADC, we shall use the NORMx and SCALEx instructions in this order.

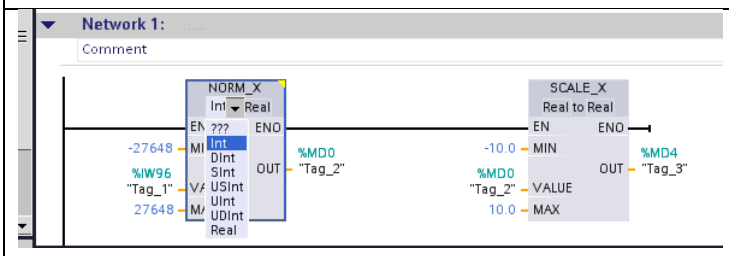


Fig.9 NORMx needs to be configured in the way shown here. Note that you need to set the input as Int first then provide all the necessary arguments. Notice IW96, the place where analogue channel 0 placed ADC data.

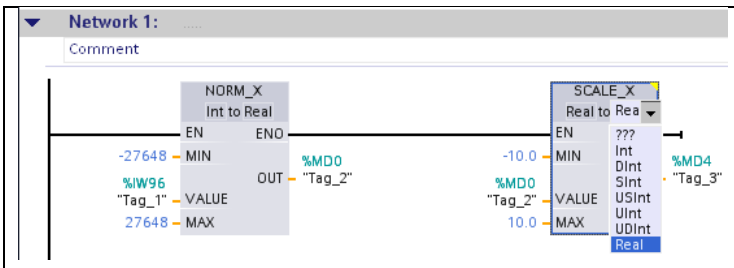


Fig. 10 SCALEx needs to be configured in the way shown here. Note that you need to set the input as Real first then provide all the necessary arguments. Notice %MD4, the place where we are placing the floating point value of voltage.

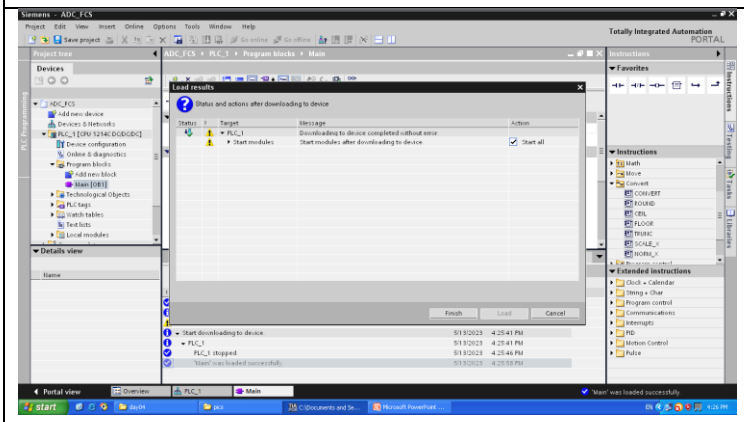


Fig. 11 Now download the program to the PLC

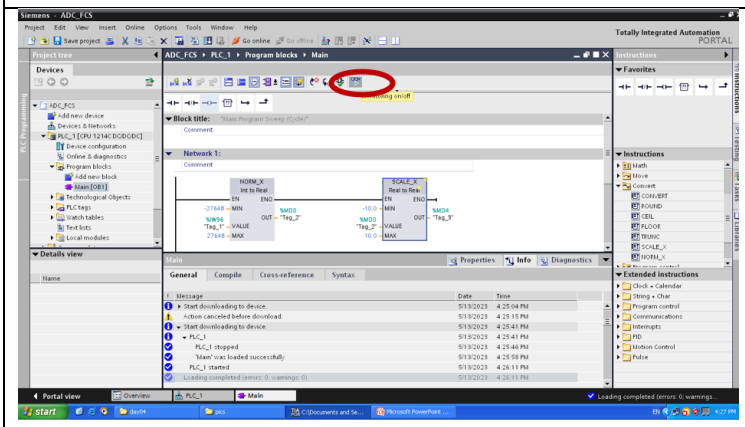


Fig.12 Click on the Spectacle like icon to monitor the values online.

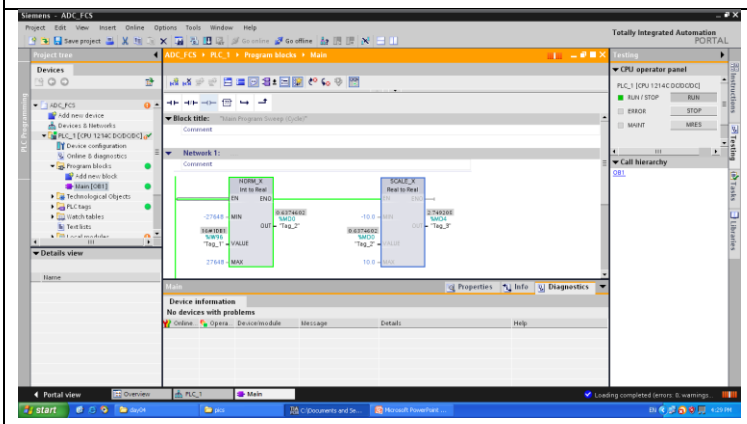


Fig. 13 Vary the potentiometer and observe the value of %MD4

Observations:_____

_____

_____

_____

Exercise 2: Interfacing the LCD Voltmeter on PLC trainer with the Analogue Output present on AI/AO module

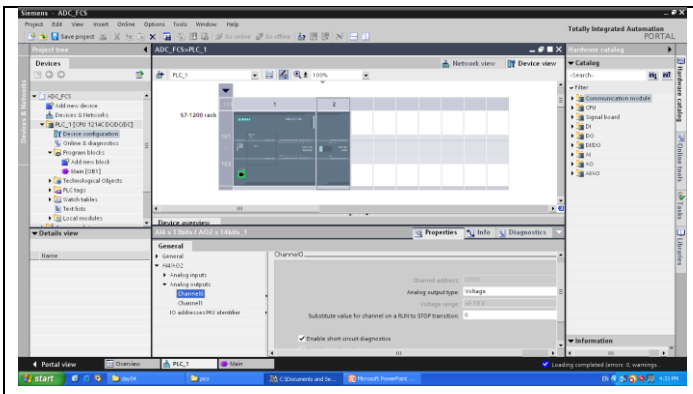


Fig.14 Keeping the same project intact, we shall now use the DAC on AQ0 (%Q0.0 of SM1234) and generate an analog voltage that is read by the LCD voltmeter on the trainer. To see the analog output channel 0, double click on SM1234 in the Devices menu. The details are available at the bottom in the General info section, under Analog Channel>Channel 0. The address of this channel is %QW96

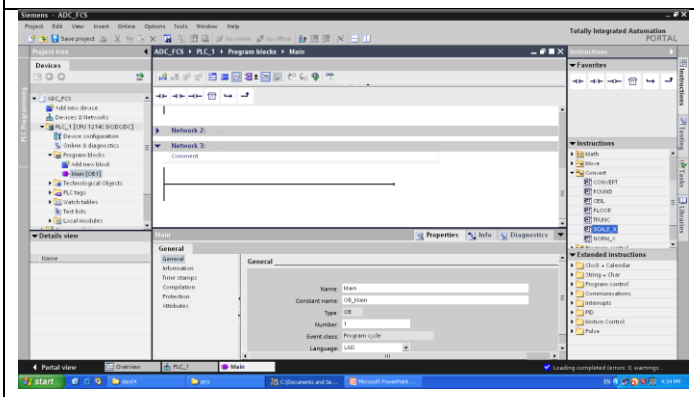


Fig.15 Now, add a new network in the Ladder program, inside Main_OB1. We shall use the NORMx and SCALEx instructions, in this order, again for this exercise.

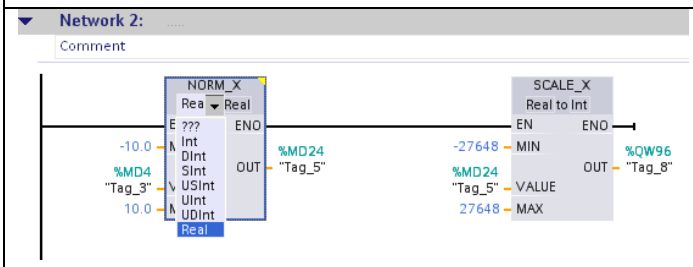


Fig. 16 NORMx needs to be configured in the way shown here. Note that you need to set the input as Real first then provide all the necessary arguments.

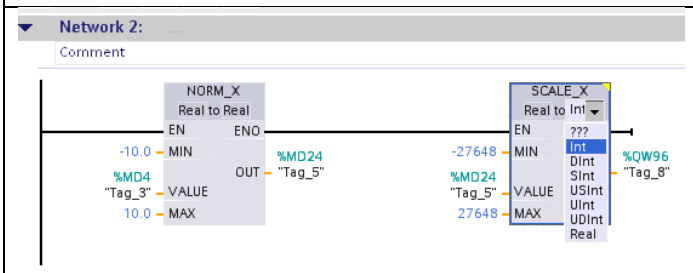


Fig.17 SCALEx needs to be configured in the way shown here. Note that you need to set the output as Int first then provide all the necessary arguments. Notice %QW96, the place where we are placing the integer value of voltage.

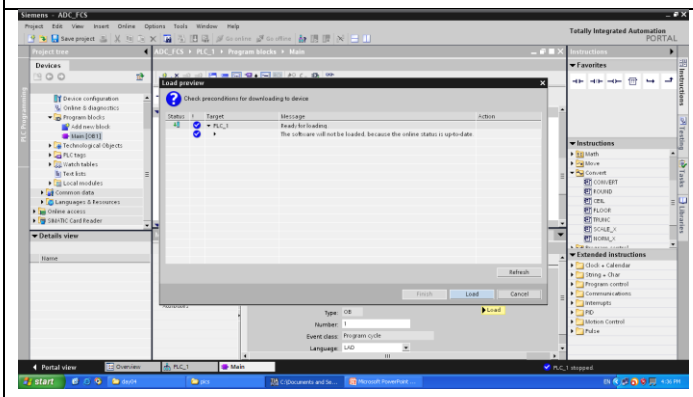


Fig.18 Download the program to the PLC

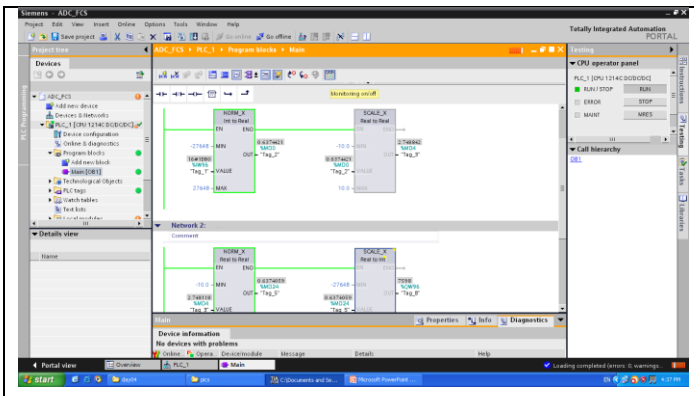


Fig.19 Turn on Real-time Monitoring and see the variables (%IW96 – input from ADC- and %QW96 – output to DAC). Also observe he trainer for the changes.

Observations_____

_____

_____

Task: LED bar-graph for representing analogue input value as a percentage

Keeping the above PLC program intact, add more networks/rungs to achieve the following objective. Create an 8 LED bar-graph using the digital outputs DQ0 to DQ7 (%Q0.0 to %Q0.7) such that when the analogue input from potentiometer varies, the LEDs on the trainer start lighting up linearly from output 0.0 to 0.7. The exact voltage ranges and the LEDs that need to turn on are presented in the table below.

Analogue Voltage Value (as read on LCD voltmeter)	DQ0 to DQ7 Status							
Less than or equal to 0 V	ON	OFF	OFF	OFF	OFF	OFF	OFF	OFF
	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7
0 V to 1.5 V	ON	ON	OFF	OFF	OFF	OFF	OFF	OFF
	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7
1.5 V to 3 V	ON	ON	ON	OFF	OFF	OFF	OFF	OFF
	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7
3 V to 4.5 V	ON	ON	ON	ON	OFF	OFF	OFF	OFF
	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7
4.5 V to 6 V	ON	ON	ON	ON	ON	OFF	OFF	OFF
	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7
6 V to 7.5 V	ON	ON	ON	ON	ON	ON	OFF	OFF
	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7
7.5 V to 9 V	ON	ON	ON	ON	ON	ON	ON	OFF
	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7
9 V to 10 V or greater	ON	ON	ON	ON	ON	ON	ON	ON
	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7

Attach your Ladder program along with a picture of the output on trainer showing voltage on LCD and LEDs.

NED University of Engineering & Technology
Department of Electrical Engineering

Course Code: **EE-375** Course Title: **Feedback Control Systems**
Laboratory Session No.: _____ Date: _____

Psychomotor Domain Assessment Rubric for Laboratory (Level P3)					
Skill(s) to be assessed	Extent of Achievement				
	0	1	2	3	4
Software Initialisation and Configuration: Set up and *recognise* software initialisation and configuration steps	Completely unable to recognise initialisation and configuration	Able to recognise initialisation but could not configure	Able to recognise initialisation but configuration is erroneous	Able to recognise initialisation and configuration with minimal errors	Able to recognise initialisation and configuration with complete success
Equipment Identification and Handling: *Sensory* skill to identify equipment and its components along with adherence to safe handling	Completely unable to identify equipment and components and no regard to safe handling	—	Ability to identify equipment but makes mistakes in recognising components, demonstrates decent equipment handling capacity	—	Ability to identify equipment and recognises all components, practices careful and safe handling
Establish and Verify Hardware-Software Connection: *Recognise* interface between computer and hardware kit and *establish* connectivity with software	Unable to perform hardware and software connection verification	—	Able to verify hardware connection but unable to establish software connection verification	—	Able to verify hardware connection and successfully establishes software connection verification
Following step-by-step procedure to complete lab work: *Observe, imitate and operate* hardware in conjunction with software to complete the provided sequence of steps	Inability to recognise and perform given lab procedures	Able to recognise given lab procedures and perform them but could not follow the prescribed order of steps	Able to recognise given lab procedures and perform them by following prescribed order of steps, with frequent mistakes	Able to recognise given lab procedures and perform them by following prescribed order of steps, with occasional mistakes	Able to recognise given lab procedures and perform them by following prescribed order of steps, with no mistakes

Psychomotor Domain Assessment Rubric for Laboratory (Level P3)					
Skill(s) to be assessed	Extent of Achievement				
	0	1	2	3	4
Programming the Controller for given Control System Problem: *Imitate* and *practice* given Ladder instructions for implementing specific control strategy and store required variables	Incorrect selection and use of programming constructs and instructions	Correct selection of programming constructs and instructions but their use is incorrect	Correct selection and use of programming constructs and instructions with many syntax/logical errors	Correct selection and use of programming constructs and instructions with little to no syntax/logical errors	Correct selection and use of programming constructs and instructions with no syntax/logical errors
Software Menu Identification and Usage: Ability to *operate* software environment *under supervision*, using menus, shortcuts, instructions etc.	Unable to understand and use software menu	Little ability and understanding of software menu operation, makes many mistake	Moderate ability and understanding of software menu operation, makes lesser mistakes	Reasonable understanding of software menu operation, makes no major mistakes	Demonstrates command over software menu usage with occasional use of advance menu options
Detecting and Removing Errors/Exceptions in Hardware and Software: *Detect* Errors/Exceptions and *manipulate,* under supervision, to rectify the Ladder program	Unable to check and detect error messages in software and hardware	Able to find error messages in software but no sense of hardware error identification	Able to find error messages in software and recognise them on hardware. Still unable to understand the error type and possible causes	Able to find error messages in software and recognise them on hardware. Moderately able in understanding error type and possible causes	Able to find error messages in software and recognise them on hardware. Reasonably able in understanding error type and possible causes
Visualisation, Comparison and analysis of results: *Copy* or enter results in analysis software to visualise and compare them with inputs. Use analysis tools to compute standard indices from result	Unable to understand and utilise visualisation, plotting and analysis software	Ability to understand and utilise visualisation and plotting instructions with errors. Unable to compute standard indices	Ability to understand and utilise visualisation and plotting instructions with occasional errors. Able to partially compute standard indices	Ability to understand and utilise visualisation and plotting instructions with no errors. Able to partially compute standard indices	Ability to understand and utilise visualisation and plotting instructions without errors. Able to compute standard indices completely

LAB SESSION 09

Objective:

Introduction to HMI (Human Machine Interface), its programming via PLC and communication set-up between PLC and HMI for measurement viasualisation

Concept and use of Human Machine Interface (HMI) in Industrial Automation

HMI, or Human-Machine Interface, is a critical component of industrial automation systems. It serves as a graphical user interface that allows operators to interact with and monitor industrial processes and machinery. HMIs provide visualization and monitoring capabilities, allowing operators to view real-time data, track trends, and monitor alarms. They also enable control and operation of industrial processes, allowing operators to adjust setpoints, start/stop processes, and activate alarms. HMIs incorporate features for alarm management, data logging, and analysis, allowing operators to respond promptly to critical situations and perform troubleshooting and optimization tasks.

Fig.1 HMI installed in an industrial setup. It allows for real-time monitoring and configuration of industrial process

HMIs are configured and programmed using software tools, enabling customization and integration with various devices within the automation system. They support connectivity with PLCs, DCS, motor drives, sensors, and other devices through industrial protocols. Some HMIs also offer remote access and monitoring capabilities, allowing authorized personnel to access and control the interface from remote locations using computers or mobile devices. HMIs simplify complex systems, enhance operator efficiency, and improve overall productivity and safety in industrial automation.

SIEMENS HMI Introduction – the KTP600

The Siemens KTP600 HMI (Human-Machine Interface) is a compact operator panel designed for use in industrial automation applications. It serves as a user interface for interacting with a Siemens SIMATIC S7-1200 or S7-1500 PLC or other compatible Siemens devices. Here is a brief description of the Siemens KTP600 HMI:

1. Display: The KTP600 features a TFT (Thin-Film Transistor) color display with a size of 5.7 inches. It provides clear and vibrant visualization of process data, alarms, and control elements.
2. Touchscreen: The HMI utilizes a resistive touchscreen that allows operators to interact with the displayed information. The touchscreen supports single-touch inputs and offers reliable and precise touch response.
3. Function Keys: The KTP600 includes six tactile function keys positioned below the display. These keys can be programmed to perform specific functions or to navigate through the HMI screens, enabling quick and easy access to commonly used features.
4. Communication Interfaces: The HMI is equipped with various communication interfaces to establish connections with the PLC or other devices. It supports industrial protocols such as Ethernet, MPI/Profibus DP, and USB for seamless integration into the automation system.

5. HMI Software: The KTP600 uses the Siemens WinCC Basic software for configuration and runtime functionality. This software allows users to create intuitive and visually appealing HMI screens, define alarms, and set up data logging capabilities.
6. IP Rating: The KTP600 is designed to withstand industrial environments and has an IP65 rating, which means it is dust-tight and protected against water jets, making it suitable for installation in harsh conditions.
7. Mounting Options: The KTP600 can be panel-mounted or mounted on a support arm, providing flexibility for installation in different control cabinet designs.

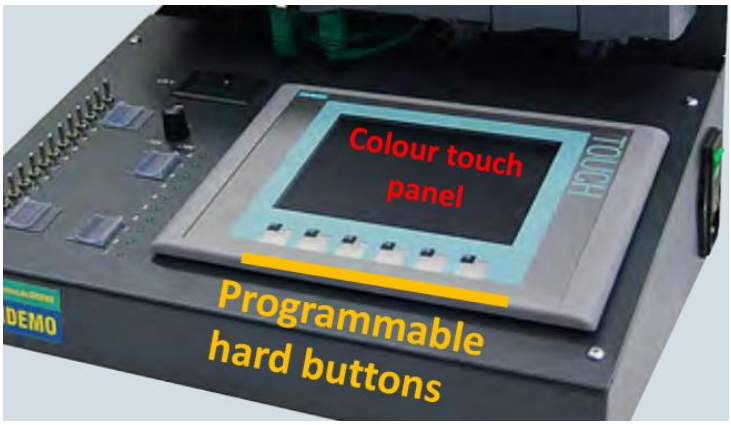


Fig.2 SIEMENS KTP600 HMI

The Siemens KTP600 HMI offers a user-friendly interface that enables operators to monitor and control industrial processes efficiently. It provides essential features and connectivity options for seamless integration into the Siemens automation ecosystem.

HMI Connectivity and Interfacing

HMI KTP600 has ethernet connectivity and can be interfaced with the PLC either directly with the CPU or through an internet switch (communication module CSM 1277). The actual connection is in our trainer system is depicted in Fig. 3, where a star network is formed.

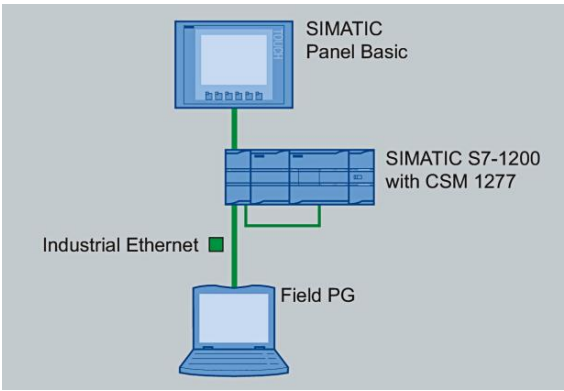
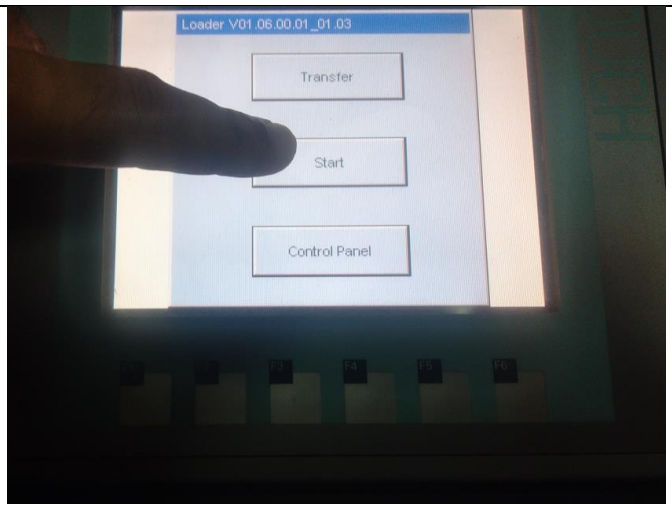
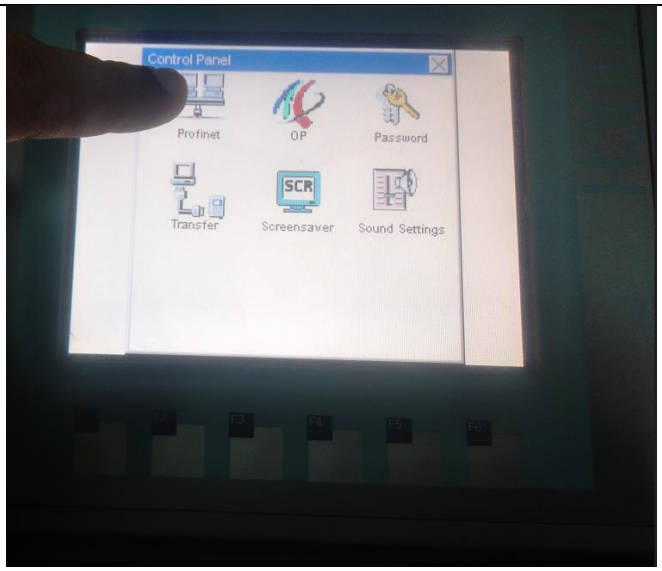
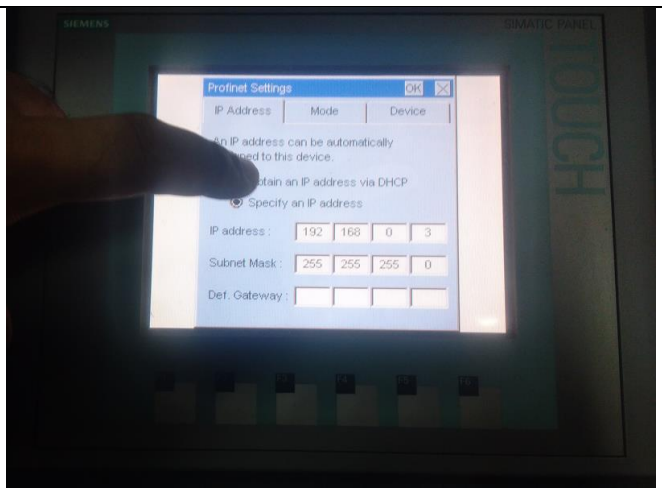
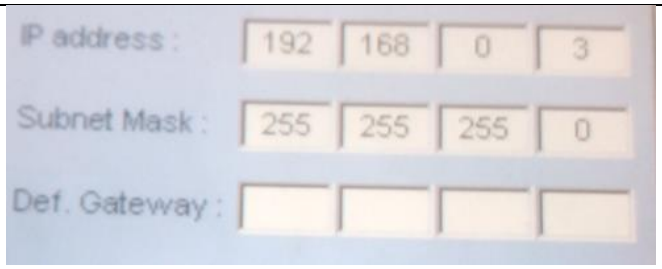


Fig.3: Star topology connecting the FIELD PG laptop to the ethernet switch (CSM 1277) that extends connection to CPU and SIMATIC HMI (LCD) panel

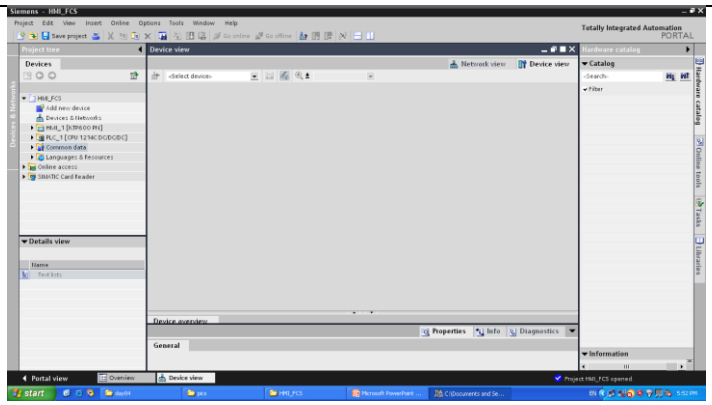
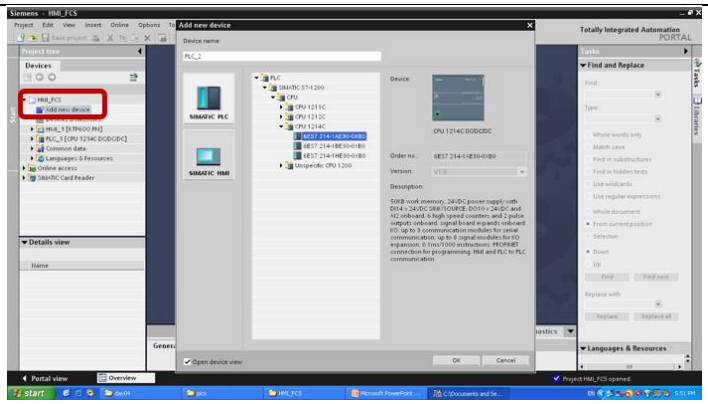
This connection requires both the PLC and the HMI to be under the same subnet address. This address can either be set automatically using the DHCP protocol or manually through the HMI configuration interface. In the following steps, we describe how to set the IP address and subnet manually on the HMI.

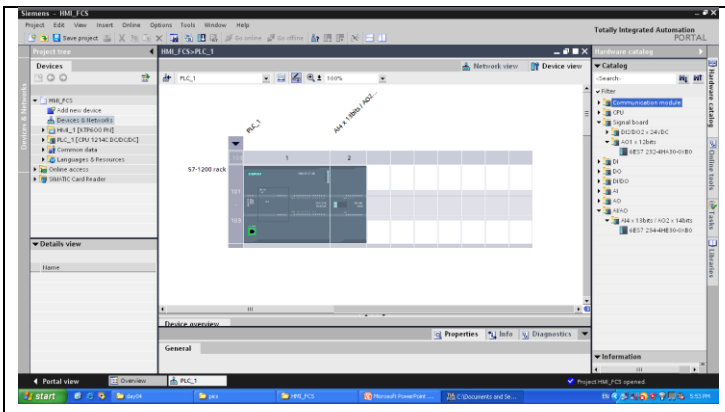
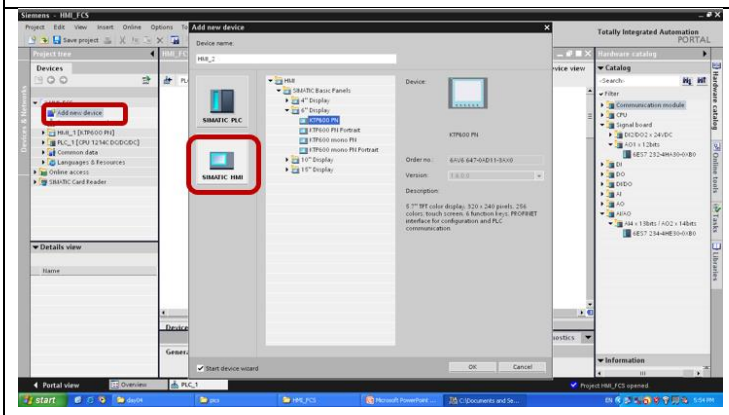
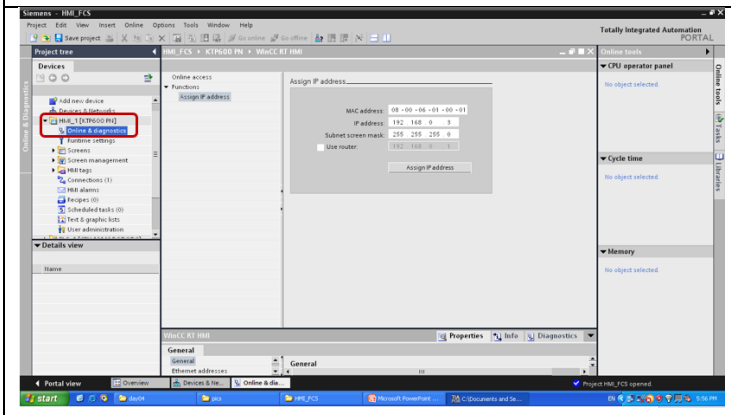
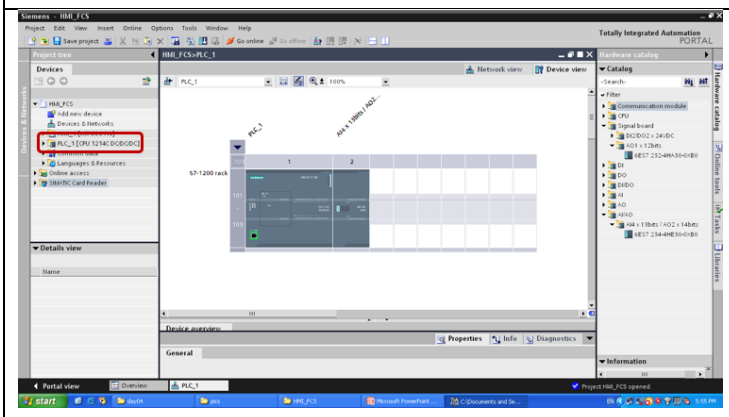
	Fig.4 Turn the PLC trainer OFF by switching the power switch and turn it back on again. The shown screen shall greet you when it starts.
	Fig. 5: A menu shall appear with three options: Transfer, Start and Control Panel. This is going to disappear after 5 seconds so quickly press the Control Panel option.
	Fig. 6 In the Control Panel, select Profinet. This is the name for industrial ethernet connection that is used across PLC models and manufacturers.

	Fig.7 Now a screen with IP address and subnet mask shall appear. You can change these by touching the respective field and an on-screen keypad shall allow you o make the changes.
	Fig. 8 Finally set the IP as 192.168.0.3 and subnet mask as 255.255.255.0. Later, when we create the PLC project in TIA portal we will make sure that the PLC's subnet mask is the same.

Exercise: Taking analogue input from potentiometer and displaying it on the HMI panel

In this task, we shall measure analogue voltage from the potentiometer present on the trainer, and then display it on the HMI via a readout and a bar-graph. The steps needed to be taken to complete the project are presented here.

	Fig.9 Create a new project and enter into Project View
	Fig.10 Add New Device from the Project Tree and select SIEMENS PLC > S71200 > CPU1214C (DC/DC/DC)

	Fig.11 From the Catalog menu on the right, select the appropriate Signal Board (AQ0) and Signal Module (SM1234) and drag them to the PLC rail
	Fig.12 Now, again go to Add New Device, this time selecting SIMATIC HMI and subsequently select KTP600 PN
	Fig.13 Now set the IP address and subnet mask of the HMI by selecting HMI_1 from the Project Tree and then select Online and Diagnostics. Here, set the IP as 192.168.0.3 and subnet mask as 255.255.255.0
	Fig.14 The PLCs IP address needs to be checked now. Select CPU_1

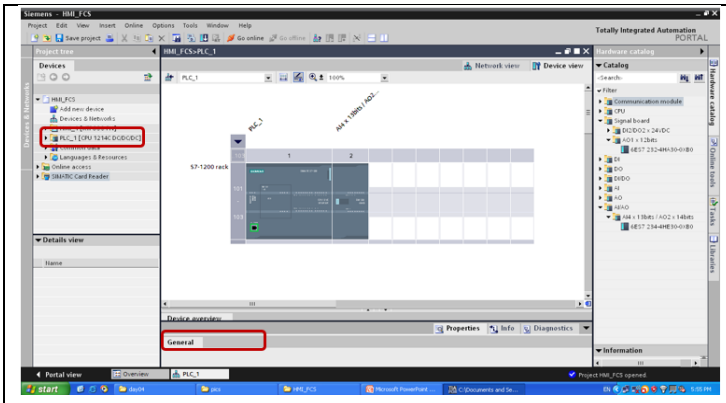


Fig.15 Inside CPU_1, select Network and Diagnostics. Here the subnet mask f the CPU needs to be same as that of the HMI

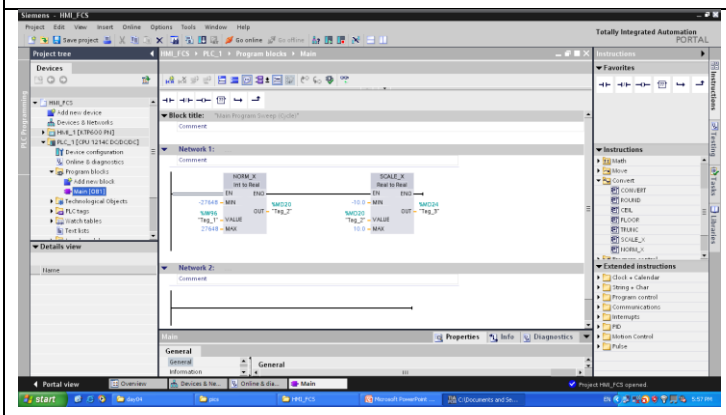


Fig.16 Now create a Ladder program that measures voltage from on-board potentiometer. Here, you can follow Exercise 1 of Lab 8

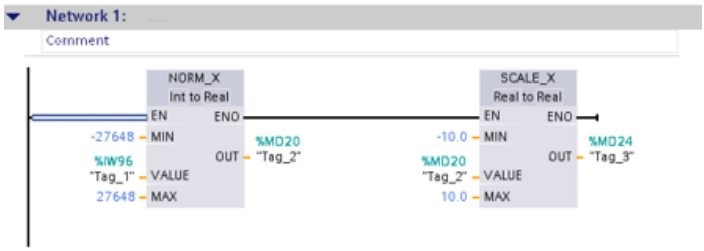


Fig.17 For reference, the Ladder program for this lab is available here

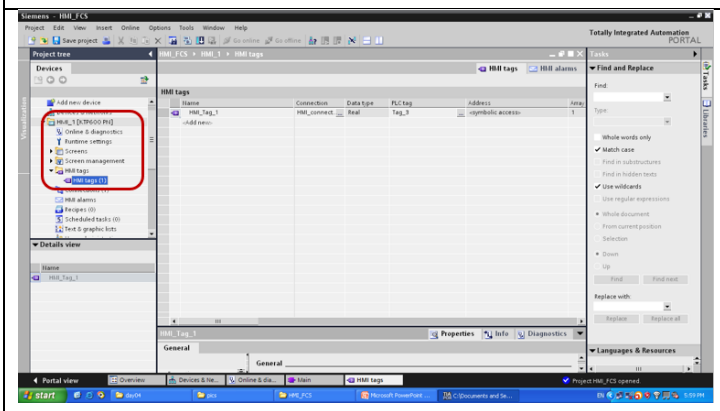


Fig.18 As the output of analogue input is stored in %MD24, we need to transfer it to PLC Tags. PLC Tags is present under HMI_1>PLC_Tags.

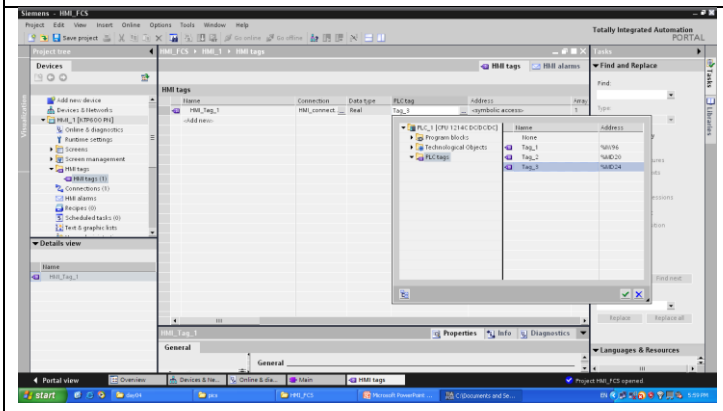


Fig.19 In PLC Tags, click on Add New. A menu will appear beneath the PLC Tag column. From this column, select the tag for %MD24

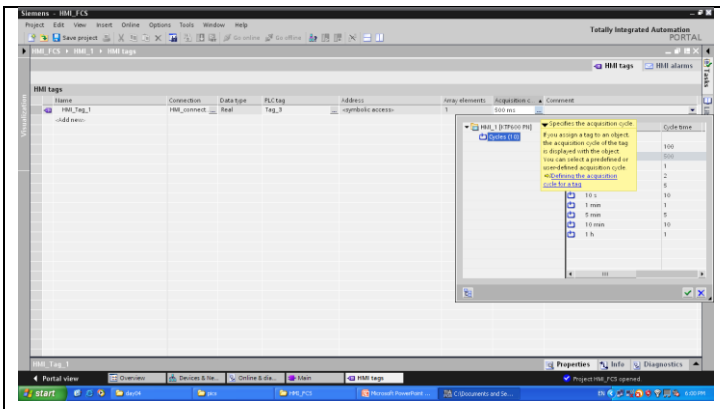


Fig.20 A few columns over to the right, click on the Acquisitions column and select 500ms as the variable acquisition time. Note that this is not the sampling frequency. It is the update time of variable on the HMI screen.

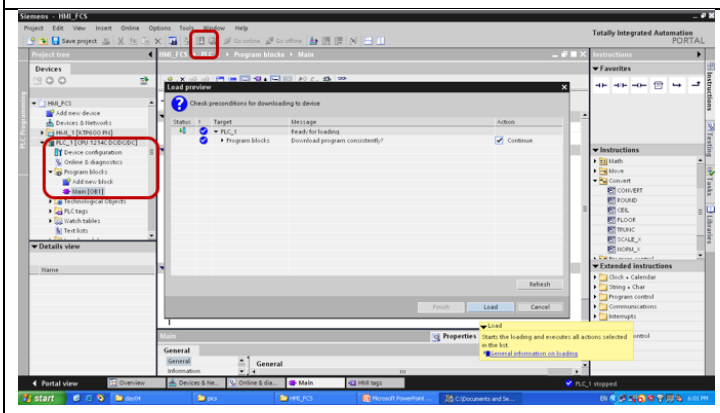


Fig.21 The PLC program is ready and now can be downloaded to the PLC. For this first go to CPU_1 in Project Tree and then select Program Blocks>Main_OB1 and then click on the Download button from the top menu.

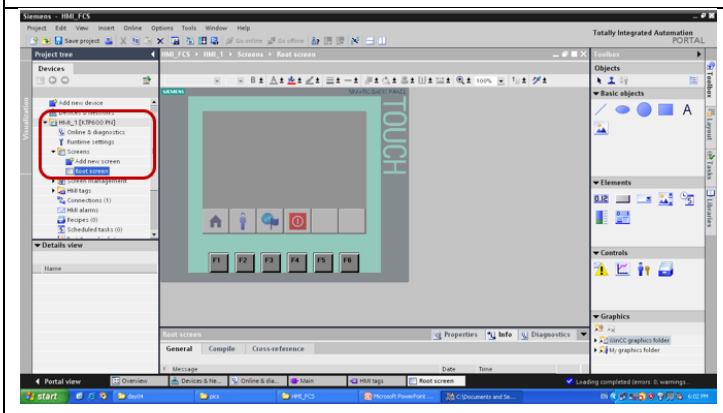


Fig.22 This brings us to write program for the HMI, which is written on the Root Screen. Select HMI_1 in Project Tree and then choose HMI Screens>Root Screen. A graphic window shall appear in the programming pane.

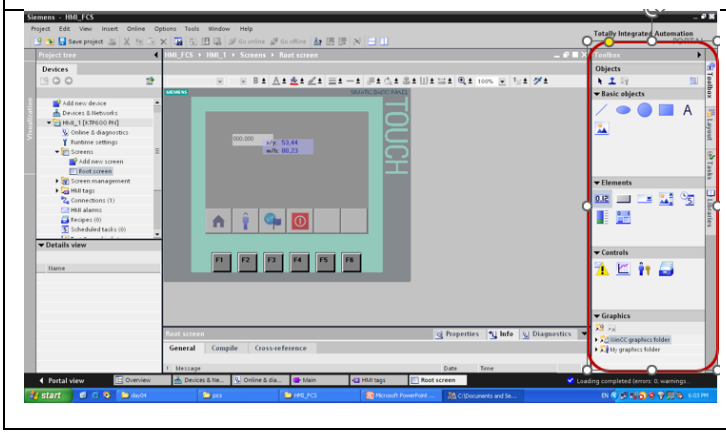
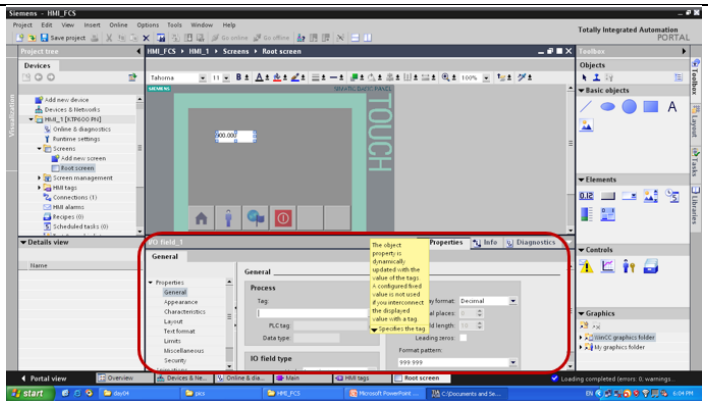
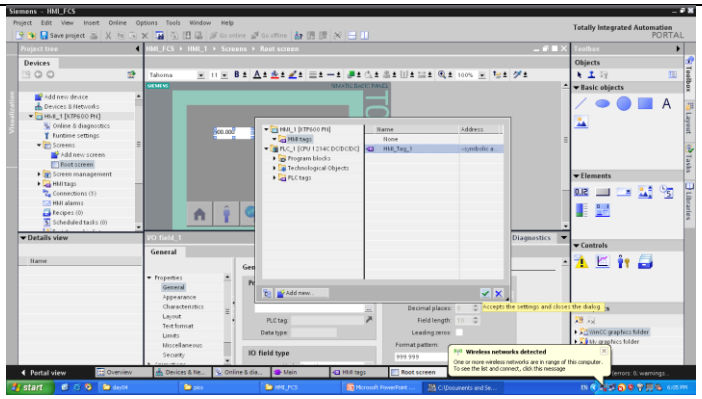
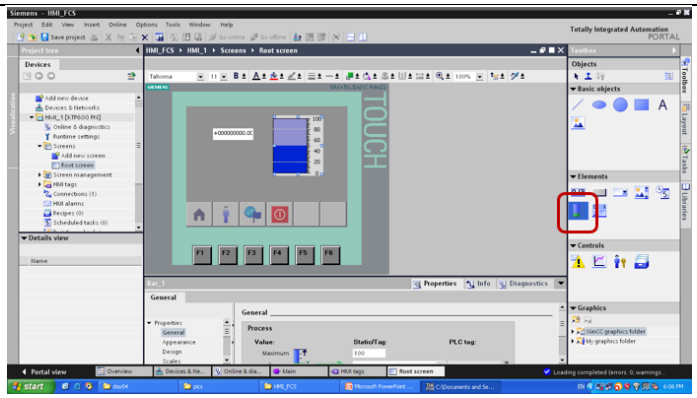
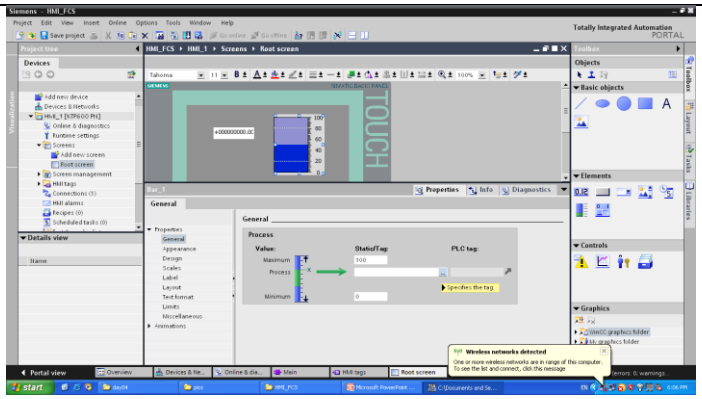
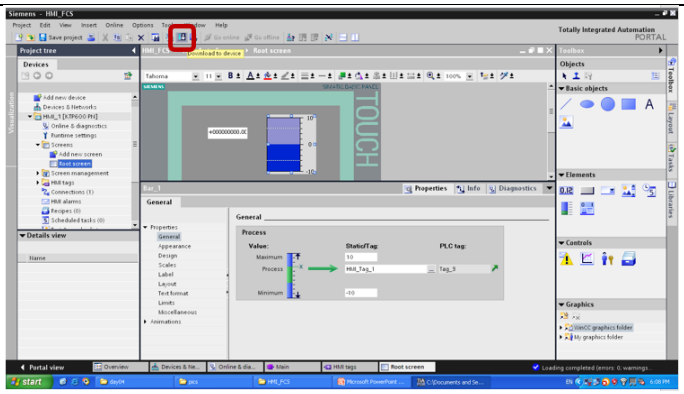
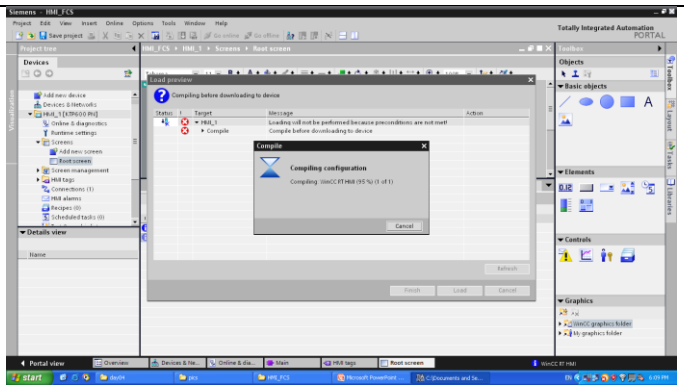
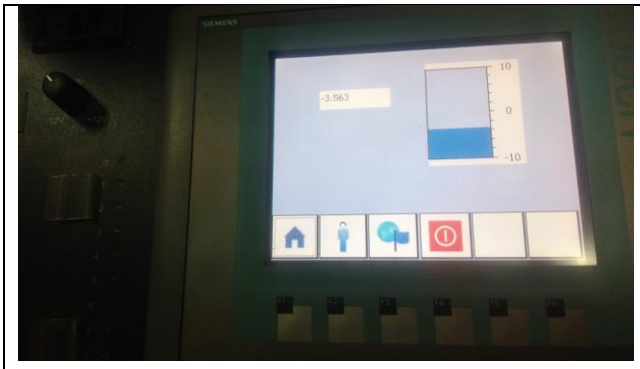
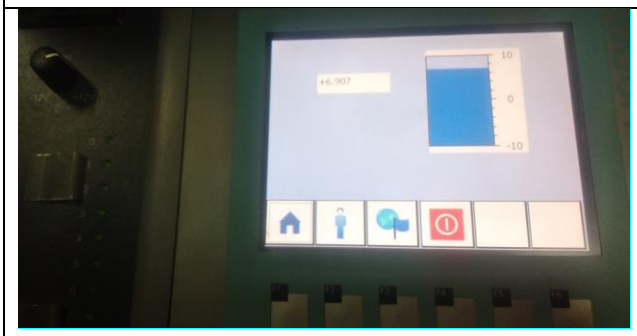


Fig.23 Basic graphical programming can be done by dragging any one of the Objects from the right instructions pane and dropping them on the screen.

	Fig.24 We shall use I/O Field to show our voltage on screen. Once I/O Field is placed on the HMI Screen, its tag can be associated in the I/O Field General menu below.
	Fig.25 This figure shows the tag being selected for the I/O Field
	Fig.26 In the same way as the I/O Field, select bar-graph from the Toolbox and drag it to the HMI Screen.
	Fig.27 Select the same tag for the bar-graph and set its range from -10 to 10

	Fig.28 Now, download the HMI program to the HMI by clicking on the Download button on the top menu.
	Fig. 29 HMI Program being downloaded to the PLC

Observation

	
	Rotate the potentiometer and see the change it brings to the HMI display. Report your observation:_____ _____ _____ Now, report what happens when the potentiometer exceeds the +/-10 V range: _____ _____ _____

Task: Program the PLC to accept two digital inputs from %I0.2 and %I0.4 and one analogue input from %IW96 (%I0.0 input of analogue module SM1234) and display them on the HMI via the following method.

Input Name	Display Object	When OFF	When ON
Digital Input %I0.2	Circle	Circle filled with green colour	Circle filled with red colour
Digital Input %I0.4	Circle	Circle filled with green colour	Circle filled with red colour
Analogue Input %I0.0	Circle	Time-series graph	

Submit screen capture of your Ladder program, PLC Root Screen design and picture of the actual output as observed on the HMI Screen.

NED University of Engineering & Technology
Department of Electrical Engineering

Course Code: **EE-375** Course Title: **Feedback Control Systems**
Laboratory Session No.: _____ Date: _____

Psychomotor Domain Assessment Rubric for Laboratory (Level P3)					
Skill(s) to be assessed	Extent of Achievement				
	0	1	2	3	4
Software Initialisation and Configuration: Set up and *recognise* software initialisation and configuration steps	Completely unable to recognise initialisation and configuration	Able to recognise initialisation but could not configure	Able to recognise initialisation but configuration is erroneous	Able to recognise initialisation and configuration with minimal errors	Able to recognise initialisation and configuration with complete success
Equipment Identification and Handling: *Sensory* skill to identify equipment and its components along with adherence to safe handling	Completely unable to identify equipment and components and no regard to safe handling	—	Ability to identify equipment but makes mistakes in recognising components, demonstrates decent equipment handling capacity	—	Ability to identify equipment and recognises all components, practices careful and safe handling
Establish and Verify Hardware-Software Connection: *Recognise* interface between computer and hardware kit and *establish* connectivity with software	Unable to perform hardware and software connection verification	—	Able to verify hardware connection but unable to establish software connection verification	—	Able to verify hardware connection and successfully establishes software connection verification
Following step-by-step procedure to complete lab work: *Observe, imitate and operate* hardware in conjunction with software to complete the provided sequence of steps	Inability to recognise and perform given lab procedures	Able to recognise given lab procedures and perform them but could not follow the prescribed order of steps	Able to recognise given lab procedures and perform them by following prescribed order of steps, with frequent mistakes	Able to recognise given lab procedures and perform them by following prescribed order of steps, with occasional mistakes	Able to recognise given lab procedures and perform them by following prescribed order of steps, with no mistakes

Psychomotor Domain Assessment Rubric for Laboratory (Level P3)					
Skill(s) to be assessed	Extent of Achievement				
	0	1	2	3	4
Programming the Controller for given Control System Problem: *Imitate* and *practice* given Ladder instructions for implementing specific control strategy and store required variables	Incorrect selection and use of programming constructs and instructions	Correct selection of programming constructs and instructions but their use is incorrect	Correct selection and use of programming constructs and instructions with many syntax/logical errors	Correct selection and use of programming constructs and instructions with little to no syntax/logical errors	Correct selection and use of programming constructs and instructions with no syntax/logical errors
Software Menu Identification and Usage: Ability to *operate* software environment *under supervision*, using menus, shortcuts, instructions etc.	Unable to understand and use software menu	Little ability and understanding of software menu operation, makes many mistake	Moderate ability and understanding of software menu operation, makes lesser mistakes	Reasonable understanding of software menu operation, makes no major mistakes	Demonstrates command over software menu usage with occasional use of advance menu options
Detecting and Removing Errors/Exceptions in Hardware and Software: *Detect* Errors/Exceptions and *manipulate,* under supervision, to rectify the Ladder program	Unable to check and detect error messages in software and hardware	Able to find error messages in software but no sense of hardware error identification	Able to find error messages in software and recognise them on hardware. Still unable to understand the error type and possible causes	Able to find error messages in software and recognise them on hardware. Moderately able in understanding error type and possible causes	Able to find error messages in software and recognise them on hardware. Reasonably able in understanding error type and possible causes
Visualisation, Comparison and analysis of results: *Copy* or enter results in analysis software to visualise and compare them with inputs. Use analysis tools to compute standard indices from result	Unable to understand and utilise visualisation, plotting and analysis software	Ability to understand and utilise visualisation and plotting instructions with errors. Unable to compute standard indices	Ability to understand and utilise visualisation and plotting instructions with occasional errors. Able to partially compute standard indices	Ability to understand and utilise visualisation and plotting instructions with no errors. Able to partially compute standard indices	Ability to understand and utilise visualisation and plotting instructions without errors. Able to compute standard indices completely

LAB SESSION 10

Objective:

DC motor speed measurement and control via PLC utilising analogue I/O, digital I/O, PWM generator HMI and other PLC peripherals

Connecting DC Motor to PLC Digital Output

DC Motors are not connected directly with the digital outputs of a PLC in real-world settings. Power converter modules are usually connected to the PLC output, which in turn drives DC Motor. However, for simple application where the DC Motor does not require high drive current and inductive voltages are also constrained, we can connect DC Motor with PLC digital output directly.

Two things must be verified before connecting DC Motor to a PLC digital output:

a) Current supplying limit of the PLC digital output
b) If speed of the Motor needs to be controlled, the digital output must have PWM(Pulse Width Modulation) capability.

In our case, S71200 CPU 1214C has a current drive capacity of 0.5A on all its digital outputs. Moreover, two of the CPU digital outputs - %Q0.0 and %Q0.1 – are also configurable as PWM Pulse Outputs.

Connecting Speed Transducer to PLC Digital Input

DC Motor in industries are coupled with shaft encoders to measure its speed and direction. This requires reading square wave pulses by the PLC, counting them and then using formula to find the speed in RPM (Revolutions Per Minute). In PLCs digital inputs generally don't have high speed pulse counting capability. Some inputs can be configured as High Speed Counters (HSCs). In S71200 CPU1214C, eight digital inputs - %I0.0 to %I0.7 – can be configured as HSC.

Requirements for this lab

Students need to bring geared DC motor with 24V input voltage and drive current less than 0.5A. Along with it, a single phase Pulse sensor is also needed with its encoder disc. Note that this type pf pulse sensor – single phase – can only measure speed and can't discern the direction of the motor.

The detailed list of components needed for this lab are:

1) Henkwell 12-24 V DC geared motor (max. 142 RPM)
2) Optical slot speed sensor based on LM393
3) DC motor speed encoder disc (20 slots)
4) 1N4002 diode
5) Connection wires

Connections

The DC motor can be connected to the CPU DQ0 (%Q0.0) which can be accessed by lifting the bottom flap. The two motor wires must be connected between %Q0.0 and M terminals. Diode 1N4002 must be connected such that its anode goes to %Q0.0 terminal and its cathode to M terminal.

The pulse sensor output wire (Pulse Out) should be connected to CPU digital input DI0 (%I0.0). The other wire from the sensor is connected to the M terminal. Note that DI0 and M can be accessed by lowering down the top flap of the CPU.

Exercise 1: Ladder program for measuring DC Motor speed via Pulse Transducer

After creating a new project, Add New Hardware and select CPU1214C. Now add the signal board to the CPU.

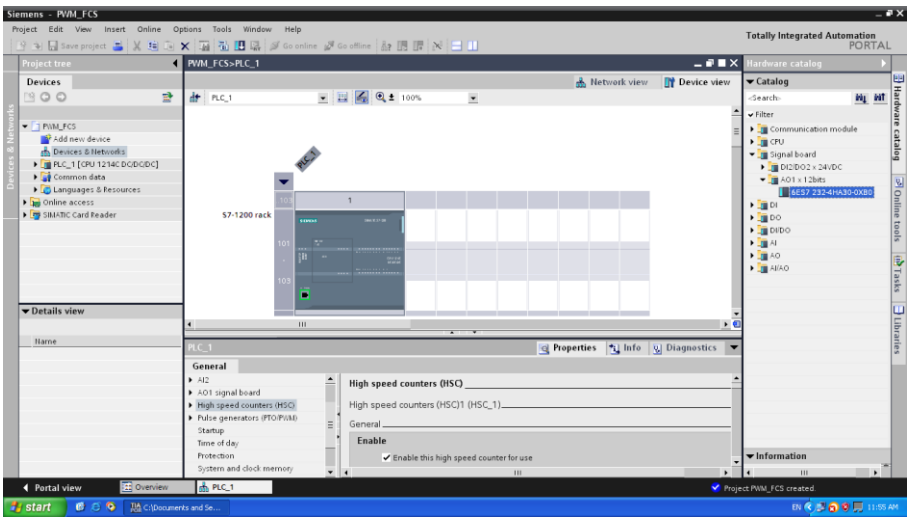


Fig.1 Select CPU1214C from Add New Device menu and add the Signal Board AO0

Here, we must configure the HSC on digital input DI0. Double click on the CPU and in the Device Overview section choose High Speed Counter (HSC)> High Speed Counter (HSC) 1. Enable HSC 1 by checking the Enable box. Note the address of the counter variable for HSC 1 (It is %ID1000).

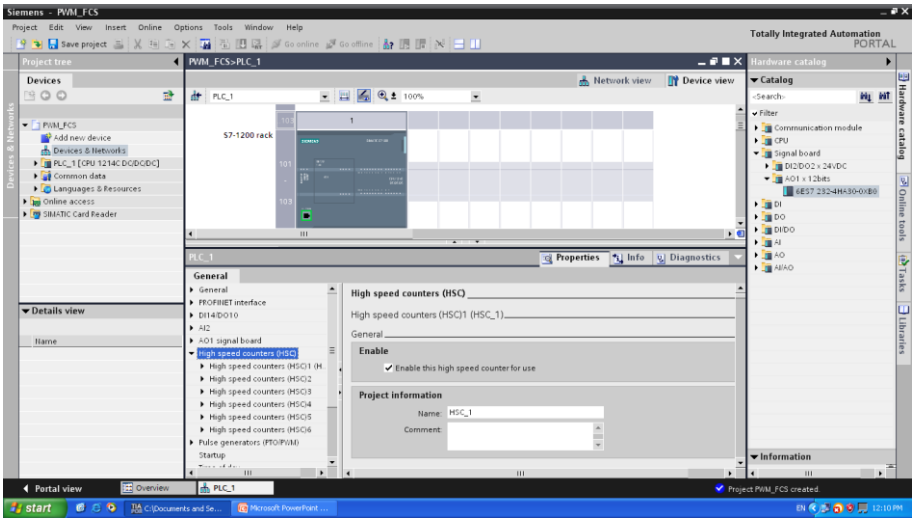


Fig.2 High Speed Counter 1, enabling and configuring it

Another configuration you need to do is set the HSC 1 function as Frequency.

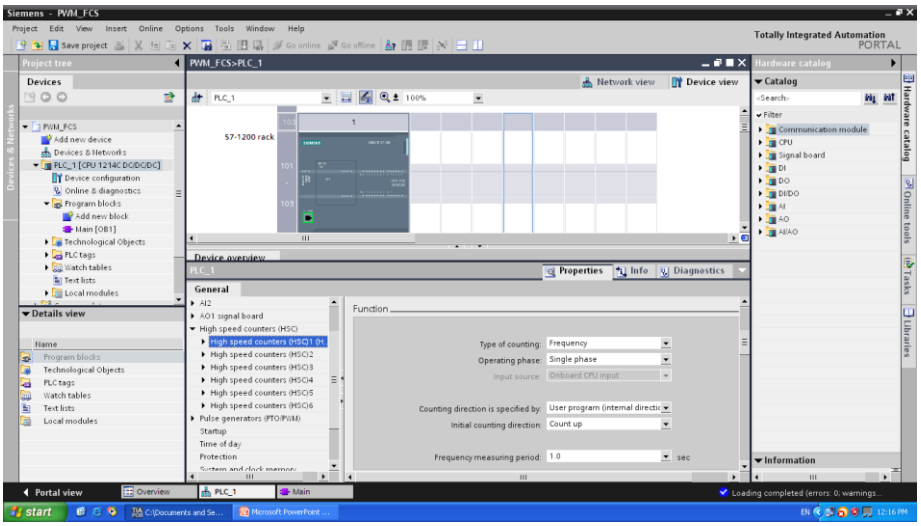


Fig.3 Setting HSC 1 to measure Frequency

Now create the following Ladder network in Main_OB1

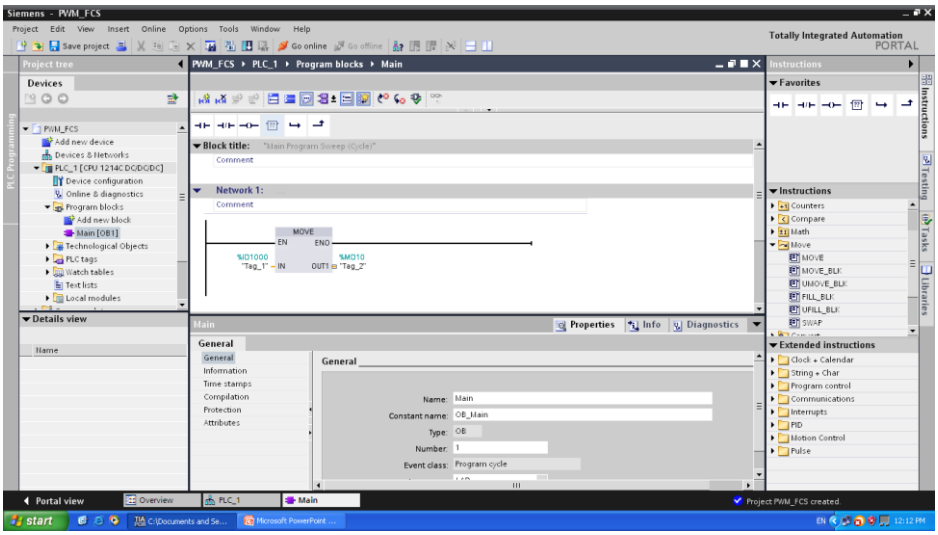


Fig.4 Using MOVE instruction to read HSC1 ID1000 and storing it in MD10

Exercise 2: Ladder program for changing DC Motor speed via analogue input

In the same project as above, first interface the analogue input and read its value using the following network. This will be the second network of this project.

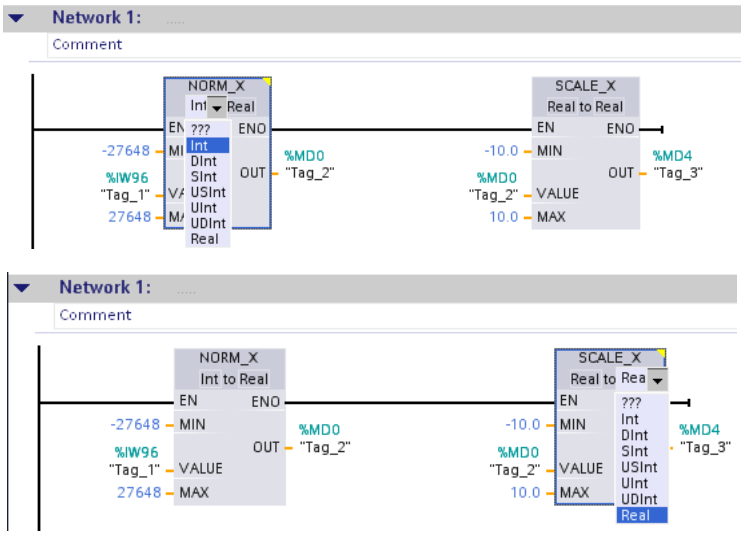


Fig.5 Using NORM and SCALE to measure and store values from analogue channel 0

Now, configure the PWM output on digital output DQ0 by double clicking on the PLC in the Devices and Networks menu.

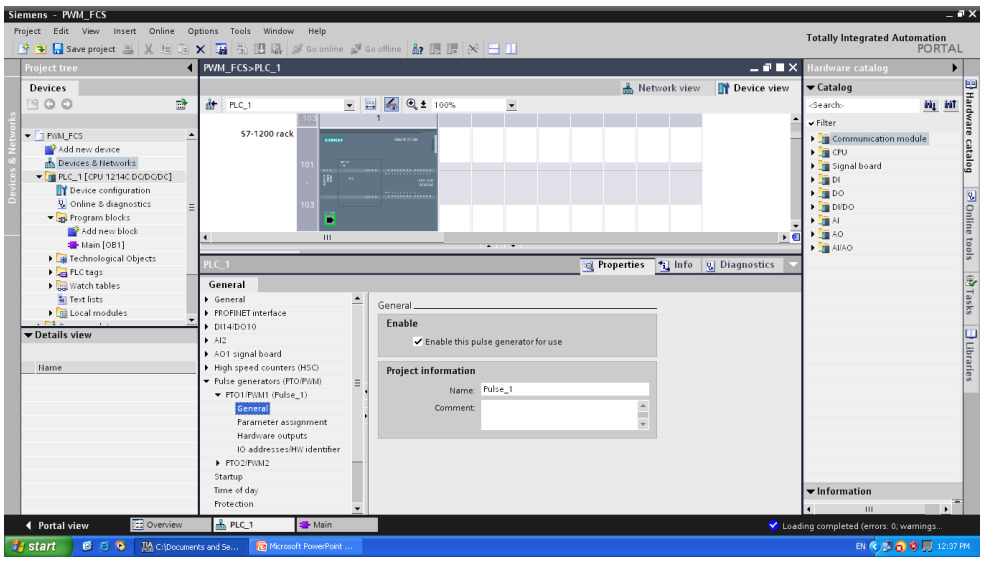


Fig.6 Enabling PTO/PWM 1 from Pulse Generator in Device Overview

Here, you can find, Pulse Generators (PTO/PWM), where two modules are shown. Enable PTO/PWM 1 and note down its address.

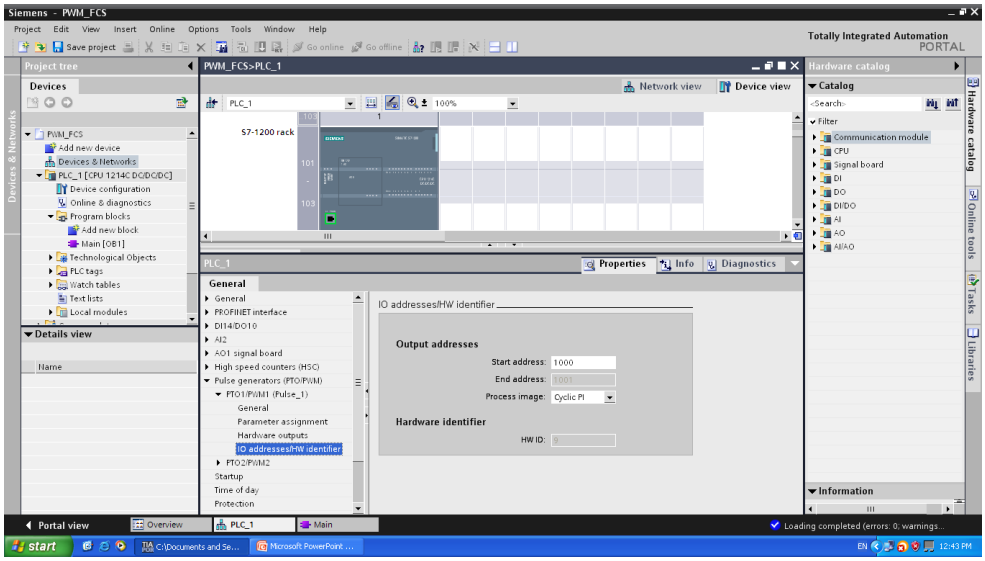


Fig.7 Hardware address of PTO/PWM 1. This is the address where the PWM duty cycle needs to be written

Finally create the following Ladders in Main_OB1. With this you can download the program in the PLC and observe the inputs and outputs via Online Monitoring.

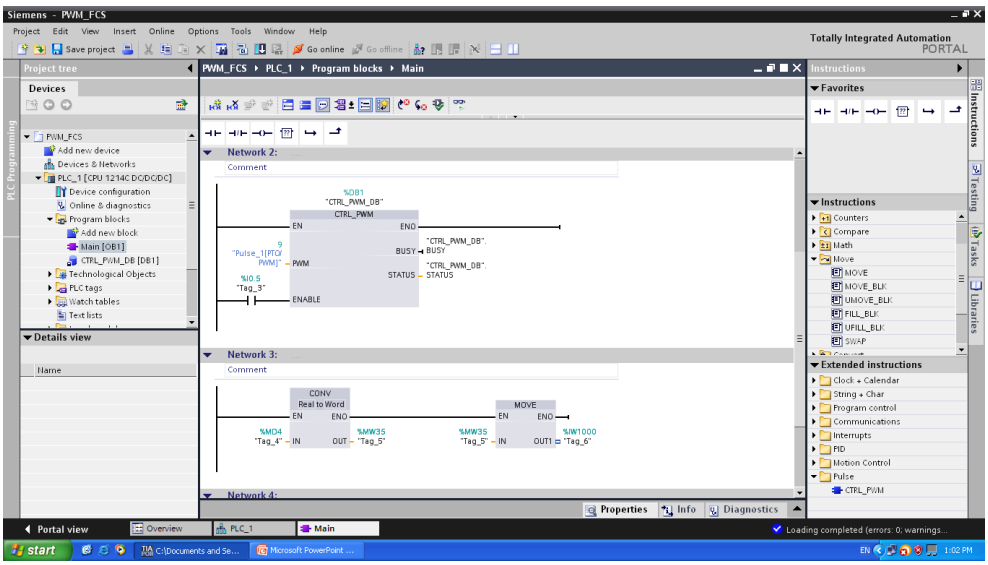


Fig.8 Using CTR_PWM instruction from Extended Instructions and then CONV and MOVE to use PWM output

Task: Draw the connection diagram of DC Motor and Pulse Sensor with the PLC CPU. Also, note down your observations how DC motor speed varies with the rotation of potentiometer.

Cover Page for Each PBL/OEL

Course Code:	EE-375
Course Name:	Feedback Control Systems
Semester:	Spring / Fall
Year:	20__
Section:	
Batch:	
Lab Instructor name:	
Submission deadline:	

PBL or OEL Statement: To simulate and design hardware of a feedback control system using Buck converter as plant.

Deliverables:

Write the report containing all calculations by hand and simulation on Matlab/Simulink. Include all code and waveforms. Also submit hardware.

Methodology:

Task:

a) The mathematical model of Buck converter (Figure 1) in frequency domain is given by the following transfer function:

T(s)= (V_0 (s))/(d(s)) = V_s/(LCs^2+(L/R)s+1)

where;

Vo(s)= output voltage in s-domain

d(s)= duty cycle in s-domain

L= inductance

C= capacitance

R= load resistance

Simulate the converter in Simulink (using powergui library) with given circuit parameters to find out the step response.

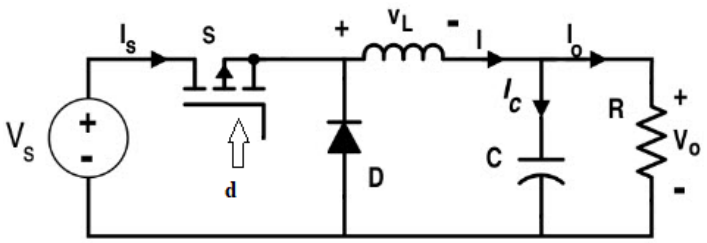


Figure 1: Buck Converter

R= 1 Ω

L=0.5 mH

C=940 µF

d=0.5 to 0.8 (50% to 80%duty cycle), Switching frequency = 3.9 kHz

V_s=5V

b) Identify the transfer function, (V_0 (s))/(d(s)) experimentally by recording the step response of the output voltage V_o of the following buck converter either by using controller (For, e.g. PLC, Arduino, PIC etc.) with following system parameters.

R= 1 Ω 10 Watt
L=0.5 mH
C=940 µF
d=0.5 to 0.8 (50% to 80%duty cycle), Switching frequency = 3.9 kHz
V_s=5 -12 V (battery or dc adaptor with 1A current capacity, use of mobile chargers is discouraged)
N-channel MOSFET IRLZ44N (please note the L here as there are other versions of this switch)
10kΩ Resistors (0.25Watt) [2 Resistors]
500Ω Resistor (0.25Watt) [1 Resistor]
Microcontroller (Arduino Uno/Nano, PIC, 8051 etc)
Potentiometer 100kΩ
Vero-board

Guidelines: The report should be maximum 5 pages long which should include figures, calculations, simulation results and waveforms Attach these two pages on top of the report. Attach the screenshot of Simulink model along with the plot of Vo by giving step PWM input of fixed duty cycle between 50% and 80%. Using the plot find ζ and peak time and use these parameters to estimate transfer function. Attach the screenshot of the plot of Vo. Using plot find ζ and peak time and use these parameters to estimate transfer function. Also make a neat and labelled circuit diagram of your setup with each detail.

c) Using potentiometer acquire voltage set value in the controller and then close the control loop by automatically adjusting the PWM by sensing voltage output Vo and comparing it with set value.

d) [Optional] Using PID control technique, adjust the PWM output by sensing Vo and set value. You are allowed to use any open-source libraries for this task with proper reference.

Rubrics: Standard software and hardware rubrics as defined for EE-375

NED University of Engineering & Technology
Department of _____ Engineering

Psychomotor Domain Assessment Rubric for Laboratory (Level P3)					
Skill(s) to be assessed	Extent of Achievement				
	0	1	2	3	4
Software Menu Identification and Usage: Ability to initialise, configure and *operate* software environment *under supervision*, using menus, shortcuts, instructions etc.	Unable to understand and use software menu	Little ability and understanding of software menu operation, makes many mistake	Moderate ability and understanding of software menu operation, makes lesser mistakes	Reasonable understanding of software menu operation, makes no major mistakes	Demonstrates command over software menu usage with frequent use of advance menu options
Procedural Programming of given model: *Practice* procedural programming techniques, in order to code specific model	Little to no understanding of procedural programming techniques	Slight ability to use procedural programming techniques for coding given algorithm	Mostly correct recognition and application of procedural programming techniques but makes crucial errors for the given model	Correctly recognises and uses procedural programming techniques with no errors but unable to run model successfully	Correctly recognises and uses procedural programming techniques with no errors and runs model successfully
Relating Theoretical Concepts, Equations and Transforms to Code: *Recognise* relation between model concepts and written code and *manipulate* the code in accordance of requirements	Completely unable to relate between model concepts and written code, unable to do manipulations	Able to recognise some relation between model concepts and written code, unable to do manipulations	Able to recognise relation between model concepts and written code, unable to do manipulations	Able to recognise relation between model concepts and written code, able to do some manipulations	Able to recognise relation between model concepts and written code, able to completely manipulate code in line with theoretical concepts
Detecting and Removing Errors: *Detect* Errors/Exceptions and in simulation and *manipulate* code to rectify the simulation	Unable to check and detect error messages and indications in software	Able to find error messages and indications in software but no understanding of detecting those errors and their types	Able to find error messages and indications in software as well as understanding of detecting some of those errors and their types	Able to find error messages in software as well as understanding of detecting all of those errors and their types	Able to find error messages in software along with the understanding to detect and rectify them

Graphical Visualisation and Comparison of model **Parameters:** *Manipulate* given simulation under supervision, in order to produce graphs/plots for measuring and comparing model parameters	Unable to understand and utilise visualisation or plotting features	Ability to understand and utilise visualisation and plotting features with frequent errors	Ability to understand and utilise visualisation and plotting features successfully but unable to compare and analyse them	Ability to understand and utilise visualisation and plotting features successfully, partially able to compare and analyse them	Ability to understand and utilise visualisation and plotting features successfully, also able to compare and analyse them
Following step-by-step procedure to complete lab work: *Observe, imitate and operate* software to complete the provided sequence of steps	Inability to recognise and perform given lab procedures	Able to recognise given lab procedures and perform them but could not follow the prescribed order of steps	Able to recognise given lab procedures and perform them by following prescribed order of steps, with frequent mistakes	Able to recognise given lab procedures and perform them by following prescribed order of steps, with occasional mistakes	Able to recognise given lab procedures and perform them by following prescribed order of steps, with no mistakes
Recording Simulation Observations: *Observe and copy* prescribed or required simulation results in accordance with lab manual instructions	Inability to recognise prescribed or required simulation measurements	Able to recognise prescribed or required simulation measurements but does not record according to given instructions	—	Able to recognise prescribed or required simulation measurements but records them incompletely	Able to recognise prescribed or required simulation measurements and records them completely, in tabular form
Discussion and Conclusion: *Demonstrate* discussion capacity on the recorded observations and draw conclusions from it, relating them to theoretical principles/concepts	Complete inability to discuss recorded observations and draw conclusions	Slight ability to discuss recorded observations and draw conclusions	Moderate ability to discuss recorded observations and draw conclusions	Reasonable ability to discuss recorded observations and draw conclusions	Full ability to discuss recorded observations and draw conclusions

NED University of Engineering & Technology
Department of Electrical Engineering

Course Code: **EE-375** Course Title: **Feedback Control Systems**
Laboratory Session No.: _____ Date: _____

Psychomotor Domain Assessment Rubric for Laboratory (Level P3)					
Skill(s) to be assessed	Extent of Achievement				
	0	1	2	3	4
Software Initialisation and Configuration: Set up and *recognise* software initialisation and configuration steps	Completely unable to recognise initialisation and configuration	Able to recognise initialisation but could not configure	Able to recognise initialisation but configuration is erroneous	Able to recognise initialisation and configuration with minimal errors	Able to recognise initialisation and configuration with complete success
Equipment Identification and Handling: *Sensory* skill to identify equipment and its components along with adherence to safe handling	Completely unable to identify equipment and components and no regard to safe handling	—	Ability to identify equipment but makes mistakes in recognising components, demonstrates decent equipment handling capacity	—	Ability to identify equipment and recognises all components, practices careful and safe handling
Establish and Verify Hardware-Software Connection: *Recognise* interface between computer and hardware kit and *establish* connectivity with software	Unable to perform hardware and software connection verification	—	Able to verify hardware connection but unable to establish software connection verification	—	Able to verify hardware connection and successfully establishes software connection verification
Following step-by-step procedure to complete lab work: *Observe, imitate and operate* hardware in conjunction with software to complete the provided sequence of steps	Inability to recognise and perform given lab procedures	Able to recognise given lab procedures and perform them but could not follow the prescribed order of steps	Able to recognise given lab procedures and perform them by following prescribed order of steps, with frequent mistakes	Able to recognise given lab procedures and perform them by following prescribed order of steps, with occasional mistakes	Able to recognise given lab procedures and perform them by following prescribed order of steps, with no mistakes

Psychomotor Domain Assessment Rubric for Laboratory (Level P3)					
Skill(s) to be assessed	Extent of Achievement				
	0	1	2	3	4
Programming the Controller for given Control System Problem: *Imitate* and *practice* given Ladder instructions for implementing specific control strategy and store required variables	Incorrect selection and use of programming constructs and instructions	Correct selection of programming constructs and instructions but their use is incorrect	Correct selection and use of programming constructs and instructions with many syntax/logical errors	Correct selection and use of programming constructs and instructions with little to no syntax/logical errors	Correct selection and use of programming constructs and instructions with no syntax/logical errors
Software Menu Identification and Usage: Ability to *operate* software environment *under supervision*, using menus, shortcuts, instructions etc.	Unable to understand and use software menu	Little ability and understanding of software menu operation, makes many mistake	Moderate ability and understanding of software menu operation, makes lesser mistakes	Reasonable understanding of software menu operation, makes no major mistakes	Demonstrates command over software menu usage with occasional use of advance menu options
Detecting and Removing Errors/Exceptions in Hardware and Software: *Detect* Errors/Exceptions and *manipulate,* under supervision, to rectify the Ladder program	Unable to check and detect error messages in software and hardware	Able to find error messages in software but no sense of hardware error identification	Able to find error messages in software and recognise them on hardware. Still unable to understand the error type and possible causes	Able to find error messages in software and recognise them on hardware. Moderately able in understanding error type and possible causes	Able to find error messages in software and recognise them on hardware. Reasonably able in understanding error type and possible causes
Visualisation, Comparison and analysis of results: *Copy* or enter results in analysis software to visualise and compare them with inputs. Use analysis tools to compute standard indices from result	Unable to understand and utilise visualisation, plotting and analysis software	Ability to understand and utilise visualisation and plotting instructions with errors. Unable to compute standard indices	Ability to understand and utilise visualisation and plotting instructions with occasional errors. Able to partially compute standard indices	Ability to understand and utilise visualisation and plotting instructions with no errors. Able to partially compute standard indices	Ability to understand and utilise visualisation and plotting instructions without errors. Able to compute standard indices completely

NED University of Engineering & Technology
Department of _____ Engineering

Psychomotor Domain Assessment Rubric for Laboratory (Level P3)					
Skill(s) to be assessed	Extent of Achievement				
	0	1	2	3	4
Software Menu Identification and Usage: Ability to initialise, configure and *operate* software environment *under supervision*, using menus, shortcuts, instructions etc.	Unable to understand and use software menu	Little ability and understanding of software menu operation, makes many mistake	Moderate ability and understanding of software menu operation, makes lesser mistakes	Reasonable understanding of software menu operation, makes no major mistakes	Demonstrates command over software menu usage with frequent use of advance menu options
Procedural Programming of given model: *Practice* procedural programming techniques, in order to code specific model	Little to no understanding of procedural programming techniques	Slight ability to use procedural programming techniques for coding given algorithm	Mostly correct recognition and application of procedural programming techniques but makes crucial errors for the given model	Correctly recognises and uses procedural programming techniques with no errors but unable to run model successfully	Correctly recognises and uses procedural programming techniques with no errors and runs model successfully
Relating Theoretical Concepts, Equations and Transforms to Code: *Recognise* relation between model concepts and written code and *manipulate* the code in accordance of requirements	Completely unable to relate between model concepts and written code, unable to do manipulations	Able to recognise some relation between model concepts and written code, unable to do manipulations	Able to recognise relation between model concepts and written code, unable to do manipulations	Able to recognise relation between model concepts and written code, able to do some manipulations	Able to recognise relation between model concepts and written code, able to completely manipulate code in line with theoretical concepts
Detecting and Removing Errors: *Detect* Errors/Exceptions and in simulation and *manipulate* code to rectify the simulation	Unable to check and detect error messages and indications in software	Able to find error messages and indications in software but no understanding of detecting those errors and their types	Able to find error messages and indications in software as well as understanding of detecting some of those errors and their types	Able to find error messages in software as well as understanding of detecting all of those errors and their types	Able to find error messages in software along with the understanding to detect and rectify them

Graphical Visualisation and Comparison of model **Parameters:** *Manipulate* given simulation under supervision, in order to produce graphs/plots for measuring and comparing model parameters	Unable to understand and utilise visualisation or plotting features	Ability to understand and utilise visualisation and plotting features with frequent errors	Ability to understand and utilise visualisation and plotting features successfully but unable to compare and analyse them	Ability to understand and utilise visualisation and plotting features successfully, partially able to compare and analyse them	Ability to understand and utilise visualisation and plotting features successfully, also able to compare and analyse them
Following step-by-step procedure to complete lab work: *Observe, imitate and operate* software to complete the provided sequence of steps	Inability to recognise and perform given lab procedures	Able to recognise given lab procedures and perform them but could not follow the prescribed order of steps	Able to recognise given lab procedures and perform them by following prescribed order of steps, with frequent mistakes	Able to recognise given lab procedures and perform them by following prescribed order of steps, with occasional mistakes	Able to recognise given lab procedures and perform them by following prescribed order of steps, with no mistakes
Recording Simulation Observations: *Observe and copy* prescribed or required simulation results in accordance with lab manual instructions	Inability to recognise prescribed or required simulation measurements	Able to recognise prescribed or required simulation measurements but does not record according to given instructions	—	Able to recognise prescribed or required simulation measurements but records them incompletely	Able to recognise prescribed or required simulation measurements and records them completely, in tabular form
Discussion and Conclusion: *Demonstrate* discussion capacity on the recorded observations and draw conclusions from it, relating them to theoretical principles/concepts	Complete inability to discuss recorded observations and draw conclusions	Slight ability to discuss recorded observations and draw conclusions	Moderate ability to discuss recorded observations and draw conclusions	Reasonable ability to discuss recorded observations and draw conclusions	Full ability to discuss recorded observations and draw conclusions