Hashemite University
Mechatronics Engineering Department
Mechatronics Systems Laboratory Manual

Prepared By: Eng.Shatha AlQadomi

Eng.Sarah AlBarguothi

Experiment 1
Introduction to LabVIEW

Objectives:

- Introduce the student to LabVIEW software.
- Design simple LabVIEW programs to demonstrate the importance of LabVIEW software in engineering application.

Apparatus:

LabVIEW software downloaded on the laboratory PCs.

Theoretical Background:

Part 1: Getting Started with LabVIEW Virtual Instruments
LabVIEW programs are called virtual instruments, or VIs, because their appearance and operation imitate physical instruments, such as oscilloscopes and multimeters. LabVIEW contains a comprehensive set of tools for acquiring, analyzing, displaying, and storing data, as well as tools to help you troubleshoot code you write.

In LabVIEW, you build a user interface, or front panel, with controls and indicators. Controls are knobs, push buttons, dials, and other input mechanisms. Indicators are graphs, LEDs, and other output displays. After you build the front panel, you add code using VIs and structures to control the front panel objects. The block diagram contains this code.

You can use LabVIEW to communicate with hardware such as data acquisition, vision, and motion control devices, as well as GPIB, PXI, VXI, RS232, and RS485 instruments.

Part 2: Launching the LabVIEW Environment
When you launch LabVIEW, the **Getting Started** window appears as shown by Figure 1.1.

Use the **Getting Started** window to create new projects and VIs. You can create items from scratch or from templates and samples. You can also open existing LabVIEW files and access LabVIEW community resources and help.
The **Getting Started** window disappears when you open an existing file or create a new file, and reappears when you close all open front panels and block diagrams. You can display the window at any time by selecting **View»Getting Started Window**.

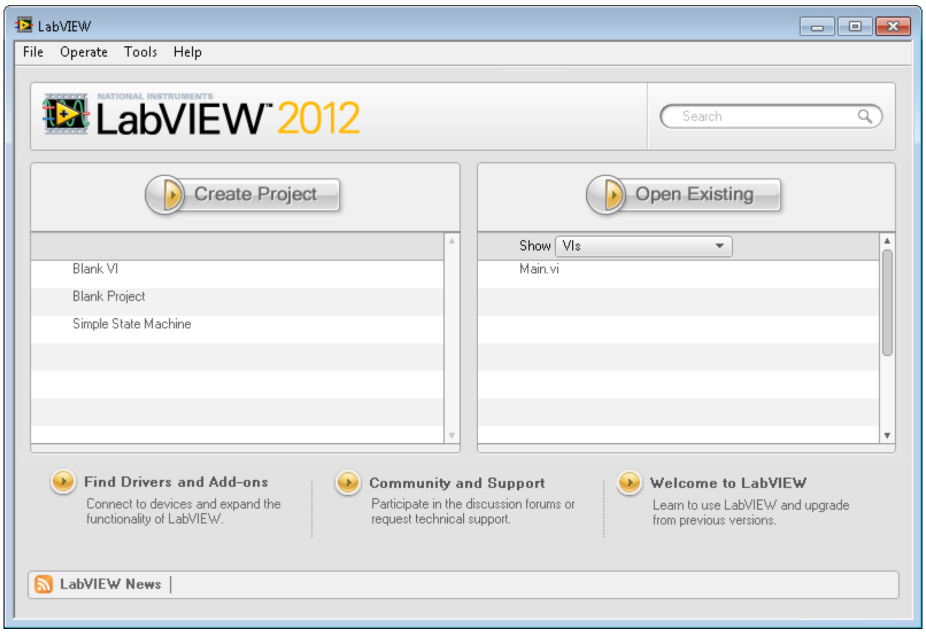

LabVIEW
File Operate Tools Help
NATIONAL INSTRUMENTS LabVIEW 2012
Search
Create Project
Open Existing
Blank VI
Blank Project
Simple State Machine
Show VIs
Main.vi
Find Drivers and Add-ons
Connect to devices and expand the functionality of LabVIEW.
Community and Support
Participate in the discussion forums or request technical support.
Welcome to LabVIEW
Learn to use LabVIEW and upgrade from previous versions.
LabVIEW News

Figure 1.1: LabVIEW Getting Started Window

<u>Projects</u>

VIs are LabVIEW programs, and you can use multiple VIs together to make a LabVIEW application. To group these application-related VIs together, use a LabVIEW project. When you save a LabVIEW project from the **Project Explorer** window, LabVIEW creates a project file (.lvproj), that includes references to all the LabVIEW files and non-LabVIEW files in the project, configuration information, build information, and deployment information.

The **Project Explorer** window includes the following items by default:

• **Project root**—Contains all other items in the **Project Explorer** window. The label on the project root includes the filename for the project.

• **My Computer**—Represents the local computer as a target in the project.

• **Dependencies**—Includes VIs and items that VIs under a target require.

• **Build Specifications**—Includes build configurations for source distributions and other types of builds available in LabVIEW toolkits and modules. If you have the LabVIEW Professional Development System or Application Builder installed, you can use **Build Specifications** to configure stand-alone applications, shared libraries, installers, and zip files.

Part 3: VI Components

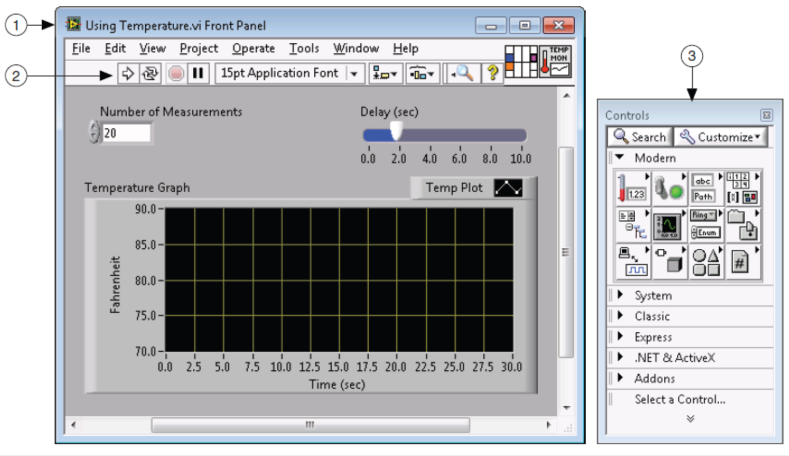
A VI contains the following three components:

1- Front panel—Serves as the user interface.

2- Block diagram—Contains the graphical source code that defines the functionality of the VI.
3- Icon and connector pane—Identifies the interface to the VI so that you can use the VI in another VI. A VI within another VI is called a subVI. A subVI corresponds to a subroutine in text-based programming languages.

1. Front Panel

The front panel, shown as follows, is the user interface of the VI. You build the front panel using controls and indicators, which are the interactive input and output terminals of the VI, respectively. Controls are knobs, push buttons, dials, and other input mechanisms. Indicators are graphs, LEDs, and other output displays. Controls simulate instrument input mechanisms and supply data to the block diagram of the VI. Indicators simulate instrument output mechanisms and display data the block diagram acquires or generates.



1	Front Panel Window	2	Toolbar	3	Controls Palette

Figure 1.2: Front Panel Window

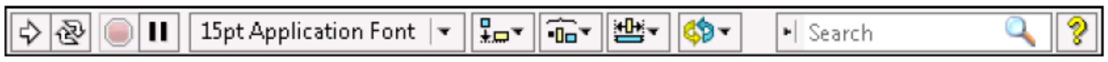
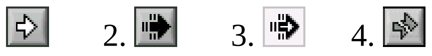


Figure 1.3: Front Panel Toolbar 2

Click the ***<u>Run button</u>*** to run a VI. LabVIEW compiles the VI, if necessary. You can run a VI if the **Run** button appears as a solid white arrow, in **1**. The solid white arrow also indicates you can use the VI as a subVI if you create a connector pane for the VI. While the VI runs, the **Run** button appears as shown in **2** if the VI is a top-level VI, meaning it has no callers and therefore is not a subVI. If the VI that is running is a subVI, the **Run** button appears as shown in **3**. The **Run** button appears broken when the VI you are creating or editing contains errors. If the **Run** button still appears broken as in **4** after you finish wiring the block diagram, the VI is broken and cannot run. Click this button to display the **Error list** window, which lists all errors and warnings.

1. 2. 3. 4.


Click the **Run Continuously** button to run the VI until you abort or pause execution. You also can click the button again to disable continuous running.

While the VI runs, the **Abort Execution** button appears. Click this button to stop the VI immediately if there is no other way to stop the VI. If more than one running top-level VI uses the VI, the button is dimmed.

Click the **Pause** button to pause a running VI. When you click the **Pause** button, LabVIEW highlights on the block diagram the location where you paused execution, and the **Pause** button appears red. Click the **Pause** button again to continue running the VI.

The following buttons are used to control the size and position of objects on the front panel.

2. Block Diagram

After you build the front panel, you add code using graphical representations of functions to control the front panel objects. The block diagram contains this graphical source code, also known as G code or block diagram code. Front panel objects appear as terminals on the block diagram.

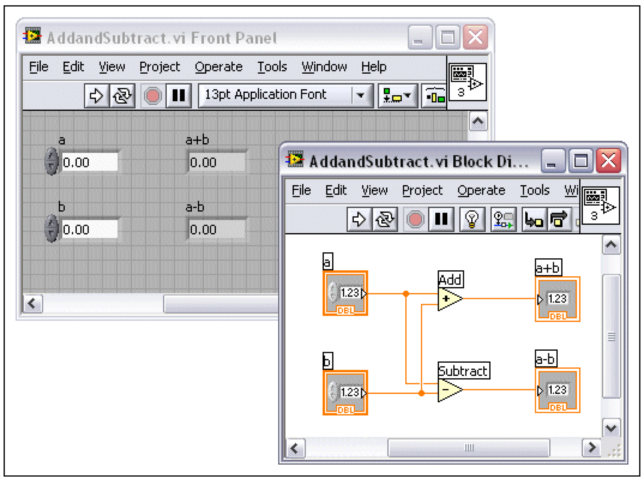
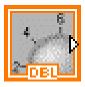

AddandSubtract.vi Front Panel
File Edit View Project Operate Tools Window Help
13pt Application Font
a
a+b
b
a-b
0.00
AddandSubtract.vi Block Di...
Add
Subtract
1.23
123

Figure 1.4: Block Diagram Window

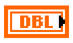
VI contains several primary block diagram objects—terminals, functions, and wires.

a. Terminals

The terminals represent the data type of the control or indicator. You can configure front panel controls or indicators to appear as icon or data type terminals on the block diagram. By default, front panel objects appear as icon terminals. For example, a knob icon terminal, shown as follows, represents a knob on the front panel.



The DBL at the bottom of the terminal represents a data type of double-precision, floating-point numeric. A DBL terminal, shown as follows, represents a double-precision, floating-point numeric control.



Terminals are entry and exit ports that exchange information between the front panel and block diagram. Data you enter into the front panel controls (a and b in the previous front panel) enter the block diagram through the control terminals. The data then enter the Add and Subtract functions. When the Add and Subtract functions complete their calculations, they produce new data values. The data values flow to the indicator terminals, where they update the front panel indicators (a+b and a-b in the previous front panel).

b. Nodes

Nodes are objects on the block diagram that have inputs and/or outputs and perform operations when a VI runs. They are analogous to statements, operators, functions, and subroutines in text-based programming languages. The Add and Subtract functions in the previous block diagram are examples of nodes.

c. Structures

Structures, a type of node, are graphical representations of the loops and case statements of text-based programming languages. Use structures on the block diagram to repeat blocks of code and to execute code conditionally or in a specific order.

d. Wires

You transfer data among block diagram objects through wires. In the previous block diagram, wires connect the control and indicator terminals to the Add and Subtract functions. Each wire has a single data source, but you can wire it to many VIs and functions that read the data. Wires are different colors, styles, and thicknesses, depending on their data types. A broken wire appears as a dashed black line with a red X in the middle. Broken wires occur for a variety of reasons, such as when you try to wire two objects with incompatible data types.

Table 1.1: Data Types in LabVIEW

Wire Type	Scalar	1D Array	2D Array	Color
Numeric	━ ━	━ ━	═ ═	Orange (floating-point), Blue (integer)
Boolean	┈	┈	┈	Green
String	〰	〰	〰	Pink

Block Diagram Toolbar

Figure 1.5 shows the elements of the block diagram toolbar.

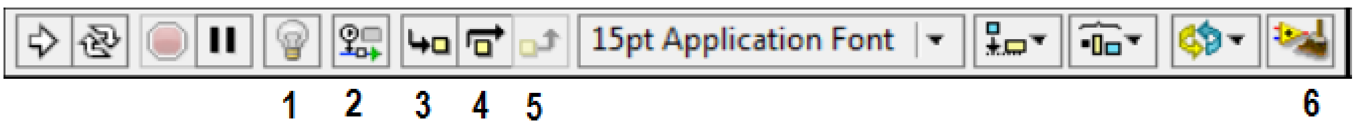


Figure 1.5: Block Diagram Toolbar

1) Click the **Highlight Execution** button to display an animation of the block diagram execution when you run the VI. Notice the flow of data through the block diagram. Click the button again to disable execution highlighting.

2) Click the **Retain Wire Values** button to save the wire values at each point in the flow of execution so that when you place a probe on the wire you can immediately retain the most recent value of the data that passed through the wire. You must successfully run the VI at least once before you can retain the wire values.

3) Click the **Step Into** button to open a node and pause. When you click the **Step Into** button again, it executes the first action and pauses at the next action of the subVI or structure. You also can press the <Ctrl> and down arrow keys. Single-stepping through a VI steps through the VI node by node. Each node blinks to denote when it is ready to execute.

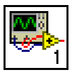
4) Click the **Step Over** button to execute a node and pause at the next node. You also can press the <Ctrl> and right arrow keys. By stepping over the node, you execute the node without single-stepping through the node.

5) Click the **Step Out** button to finish executing the current node and pause. When the VI finishes executing, the **Step Out** button is dimmed. You also can press the <Ctrl> and up arrow keys. By stepping out of a node, you complete single-stepping through the node and navigate to the next node.

6) Click the **Clean Up Diagram** button to automatically reroute all existing wires and rearrange objects on the block diagram to generate a cleaner layout. To configure the clean up options, select **Tools»Options** to display the Options dialog box and select **Block Diagram** from the **Category** list. You can configure the settings in the Block Diagram Cleanup section.

3. Icon and Connector Pane

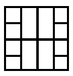
After you build a VI front panel and block diagram, build the icon and the connector pane so you can use the VI as a subVI. The icon and connector pane correspond to the function prototype in text-based programming languages. Every VI displays an icon, such as the one shown as follows, in the upper right corner of the front panel and block diagram windows.



A VI icon is a graphical representation of a VI. It can contain text, images, or a combination of both. If you use a VI as a subVI, the icon identifies the subVI on the block diagram of the VI. If you add the VI to a palette, the VI icon also appears on the Functions palette. You can double-click the icon in the front panel window or block diagram window to customize or edit it.

Note Customizing VI icons is recommended, but optional. Using the default LabVIEW icon does not affect functionality.

You also need to build a connector pane, shown as follows, to use the VI as a subVI.



The connector pane is a set of terminals that correspond to the controls and indicators of that VI, similar to the parameter list of a function call in text-based programming languages. The connector pane defines the inputs and outputs you can wire to the VI so you can use it as a subVI. A connector pane receives data at its input terminals and passes the data to the block diagram code through the front panel controls and receives the results at its output terminals from the front panel indicators.

Part 4: Data types

- Booleans have only two possible values: True or False and are indicated by green data wires.
- Numerics have many sizes and representations:
  - Integers can be signed or unsigned whole numbers and are indicated by blue data wires.
  - Doubles and Singles are signed numbers with a decimal component and are indicated by orange data wires.

- A numeric's size is indicated in bits and determines the range of possible values.

- Strings are sequences of characters and are indicated by pink data wires.
- Arrays are a groups of one data type and are indicated by thicker data wires.
  - A 1 dimensional array can be thought of as a column, a 2 dimensional array as a table, and so on.
- Clusters are a groups of various data types and indicated by a thick brown data wire.
- An Error Cluster is a special type of cluster used to indicate warnings and errors.
  - Error clusters are composed of a boolean status, a numeric error code, and a string source.

Note: you may need conversions between different data types: on the block diagram, right click programming>>numeric>>conversion, right click programming>>Boolean.

Part 5: Setting up the Connector Pane

Define connections by assigning a front panel control or indicator to each of the connector pane terminals. The connector pane is located to the left of the VI icon in the upper right corner of the front panel window. When you open LabVIEW, you see a default connector pattern. Select a different terminal pattern for a VI by right-clicking the connector pane and selecting **Patterns** from the shortcut menu.

To assign a terminal to a front panel control or indicator, click a terminal of the connector pane, then click the front panel control or indicator you want to assign to that terminal. Click an open space on the front panel window. The terminal changes to the data type color of the control to indicate that you connected the terminal.

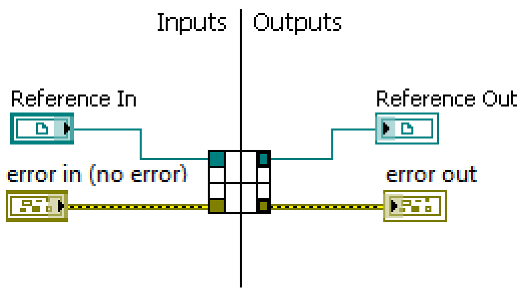


Figure 1.6: Connector Pane

Part 6: Creating Sub VIs

1- <u>Creating SubVIs from existing VI.</u>

Prepare the connector pane of the VI you intend to use as a subVI. Then from function palette , choose "Select a VI".

2- Creating SubVIs from Sections of a VI

Convert a section of a VI into a subVI by using the Positioning tool to select the section of the block diagram you want to reuse and selecting **Edit»Create SubVI**. An icon for the new subVI replaces the selected section of the block diagram. LabVIEW creates controls and indicators for the new subVI, automatically configures the connector pane based on the number of control and indicator terminals you selected, and wires the subVI to the existing wires.

Part 7: Using SubVIs

To place a subVI on the block diagram, click the **Select a VI** button on the **Functions** palette. Navigate to the VI you want to use as a subVI and double-click to place it on the block diagram.

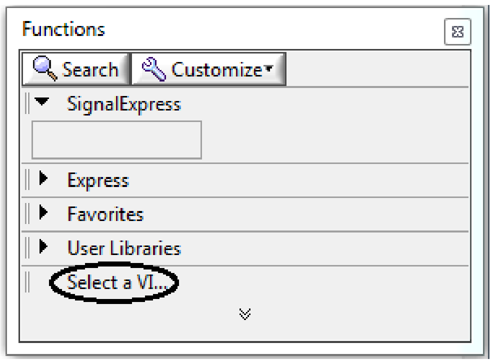


Figure 1.7: Adding a SubVI to The Block Diagram

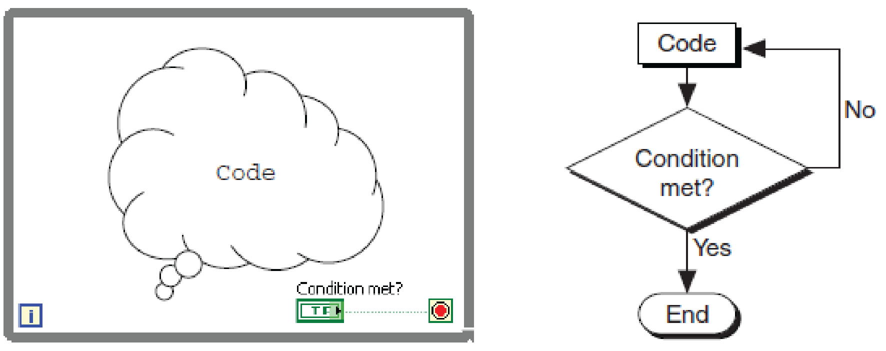
You also can place an open VI on the block diagram of another open VI. Use the Positioning tool to click the icon in the upper right corner of the front panel or block diagram of the VI you want to use as a subVI and drag the icon to the block diagram of the other VI.

Part 8: Structures

1. While Loop

A While Loop executes a subdiagram until a condition occurs. The While Loop is located on the **Structures** palette. Select the While Loop from the palette then use the cursor to drag a selection rectangle around the section of the block diagram you want to repeat. When you release the mouse button, a While Loop boundary encloses the section you selected. Add block diagram objects to the While Loop by dragging and dropping them inside the While Loop.

Tip The While Loop always executes at least once. Changing the value of the controls outside the loop, does not affect the loop because the value is only read once.



```
Repeat (code);
Until Condition met;
End;
```

Figure 1.8: How Do While Loop Work?

The iteration terminal is an output terminal that contains the number of completed iterations. The iteration terminal provides the current loop iteration count, which is zero for the first iteration.

Tunnels feed data into and out of structures like While Loops. The block is the color of the data type wired to the tunnel. Data pass out of a loop after the loop terminates.

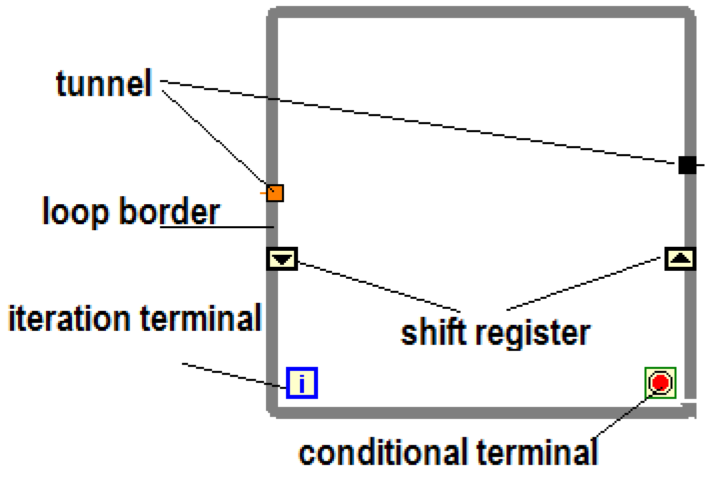


Figure 1.9: While Loop Elements

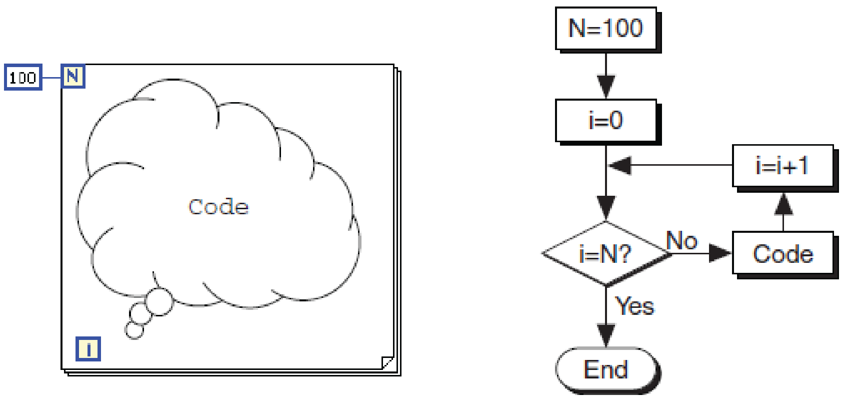
Use shift registers when you want to pass values from previous iterations through the loop to the next iteration. A shift register appears as a pair of terminals directly opposite each other on the vertical sides of the loop border. Initializing a shift register resets the value the shift register passes to the first iteration of the loop when the VI runs. If you do not initialize the shift register, the loop uses the value written to the shift register when the loop last executed or, if the loop has never executed, the default value for the data type.

Stacked shift registers let you access data from previous loop iterations. Stacked shift register remember values from multiple previous iterations and carry those values to the next iterations. To create a stacked shift register, right-click the left terminal and select **Add Element** from the shortcut menu.

If you add another element to the left terminal in the previous block diagram, values from the last two iterations carry over to the next iteration, with the most recent iteration value stored in the top shift register. The bottom terminal stores the data passed to it from the previous iteration.

2. For Loop

A For Loop executes a sub diagram a set number of times. The count terminal is an input terminal whose value indicates how many times to repeat the sub diagram. The iteration count for the For Loop always starts at zero. An extra conditional terminal can be added by right clicking on the loop border. A For Loop with a conditional terminal executes until the condition occurs or until all iterations are complete, whichever happens first. The for loop may not execute at all.



```
N=100;
i=0;
Until i=N:
    Repeat (code; i=i+1);
End;
```

Figure 1.10: How Do For Loop Work?

The output tunnel of the for loop can be configured by right-click the tunnel, and select **Tunnel Mode»**

1. **Concatenating**: To automatically concatenate an array leaving a loop, wire the array directly to the loop output tunnel. Selecting **Concatenating** appends all inputs in order, forming an output array of the same dimension as the array input wired.
2. **Last Value**: this mode displays the last value from the last loop iteration.
3. **Indexing** builds an array of higher dimension.

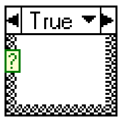
Note: You can change when LabVIEW writes values to the loop output tunnel based on a condition.

Table 1.2: For Loop Vs. While Loop

For Loop	While Loop
Executes a set number of times unless a conditional terminal is added	Stops executing only if the value at the conditional terminal meets the condition
Can execute zero times	Must execute at least once
Tunnels automatically output an array of data	Tunnels automatically output the last value

3. Case structure

A Case structure has two or more subdiagrams, or cases. Only one subdiagram is visible at a time, and the structure executes only one case at a time. An input value determines which subdiagram executes. The Case structure is similar to switch statements or if...then...else statements in text-based programming languages.



The case selector label ◄True▼► at the top of the Case structure contains the name of the selector value that corresponds to the case in the center and decrement and increment arrows on each side.

Click the decrement and increment arrows to scroll through the available cases. You also can click the down arrow next to the case name and select a case from the pull-down menu.

Wire an input value, or selector, to the selector terminal ? to determine which case executes.

You must wire an integer, Boolean value, string, or enumerated type value to the selector terminal. You can position the selector terminal anywhere on the left border of the Case structure. If the data type of the selector terminal is Boolean, the structure has a True case and a False case. If the selector terminal is an integer, string, or enumerated type value, the structure can have any number of cases.

If you do not specify a default case for the Case structure to handle out-of-range values, you must explicitly list every possible input value. For example, if the selector is an integer and you specify cases for 1, 2, and 3, you must specify a default case to execute if the input value is 4 or any other unspecified integer value.

Part 9: Timing VIs

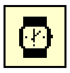
When a loop finishes executing an iteration, it immediately begins executing the next iteration, unless it reaches a stop condition. Most often, you need to control the iteration frequency or timing.

1. Wait Functions

Place a wait function inside a loop to allow a VI to sleep for a set amount of time. This allows your processor to address other tasks during the wait time. Wait functions use the millisecond clock of the operating system.

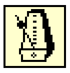
a. The **Wait (ms)** function waits until the millisecond counter counts to an amount equal to the input you specify. This function guarantees that the loop execution rate is at least the amount of the input you specify.

When LabVIEW calls a VI for example, if **millisecond timer value** is 112 ms and **milliseconds to wait** is 10 ms, the VI finishes when **millisecond timer value** equals 122 ms.



b. The **Wait Until Next ms Multiple** function waits until the value of the millisecond timer becomes a multiple of the specified **millisecond multiple**. Use this function to synchronize activities. You can call this function in a loop to control the loop execution rate.

When LabVIEW calls a VI for example, if **millisecond multiple** is 10 ms and **millisecond timer value** is 112 ms, the VI waits 8 more milliseconds until the millisecond timer value is 120 ms, a multiple of 10.



2. Elapsed Time

Indicates the amount of time that has elapsed since the specified start time.

Part 10: Plotting data

1. Waveform Graphs and Charts

Display data typically acquired at a constant rate.

a. <u>Waveform Chart</u>

The waveform chart is a special type of numeric indicator that displays one or more plots of data typically acquired at a constant rate. Waveform charts can display single or multiple plots. Configure how the chart updates to display new data. Right-click the chart and select **Advanced»Update Mode** from the shortcut menu to set the chart update mode.

The waveform chart maintains a history of data, or buffer, from previous updates.

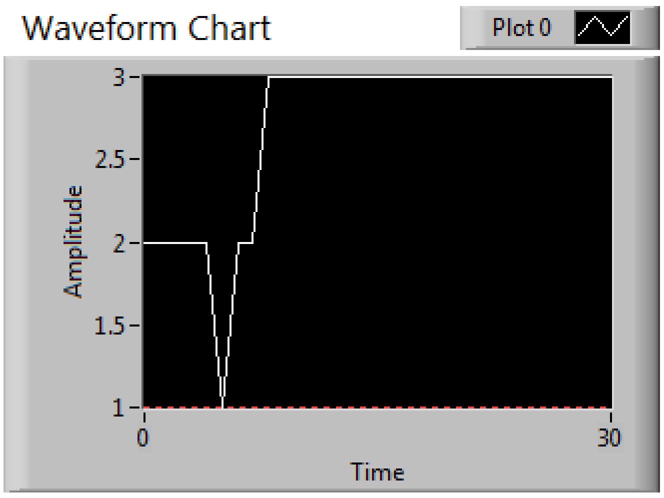


Figure 1.11: Waveform Chart

b. Waveform Graphs

The waveform graph displays one or more plots of evenly sampled measurements. The waveform graph plots only single-valued functions, as in y = f(x), with points evenly distributed along the x-axis, such as acquired time-varying waveforms. The following front panel shows an example of a waveform graph.

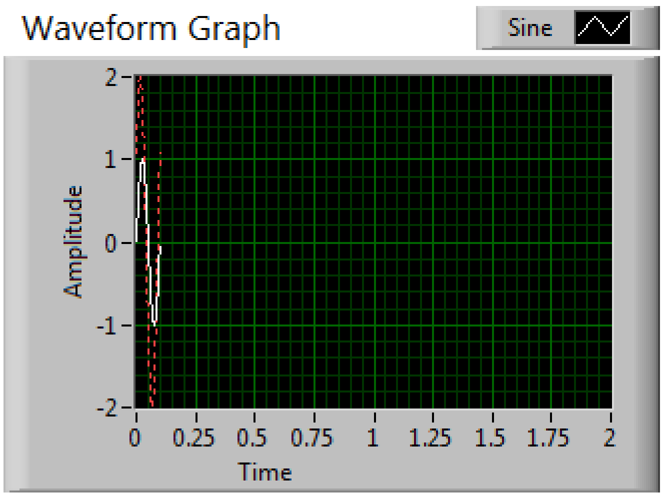


Figure 1.12: Waveform Graph

Note: If you want to pass multiple points per plot in a single update, wire an array of clusters of numeric values to the chart/graph. Each numeric represents a single y value point for each of the plots. You can use "Bundle" or "Merge" depending on the signal data type.

2. XY Graphs

Display data acquired at a non-constant rate and data for multivalued functions. You input the x and y data to the plot. The XY graph is a general-purpose, Cartesian graphing object that plots multivalued functions, such as circular shapes or waveforms with a varying time base. The XY graph displays any set of points, evenly sampled or not. You also can display Nyquist planes, Nichols planes, S planes, and Z planes on the XY graph.

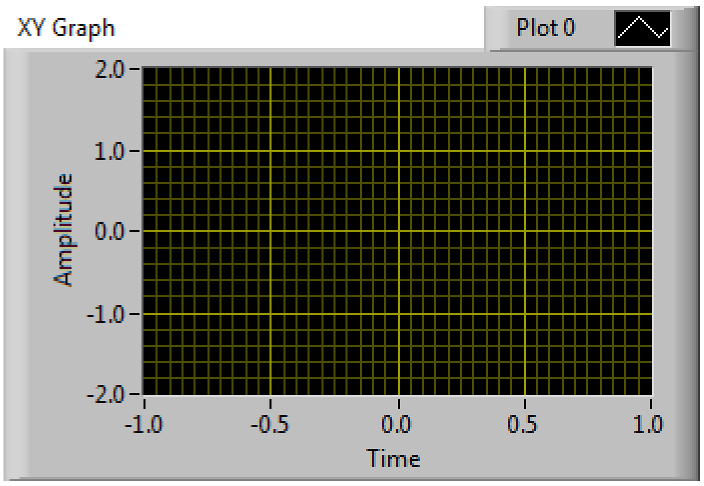


Figure 1.13: XY Graph

Part 12: Configuring DAQ Using LabVIEW

1. Inserting a DAQ assistant

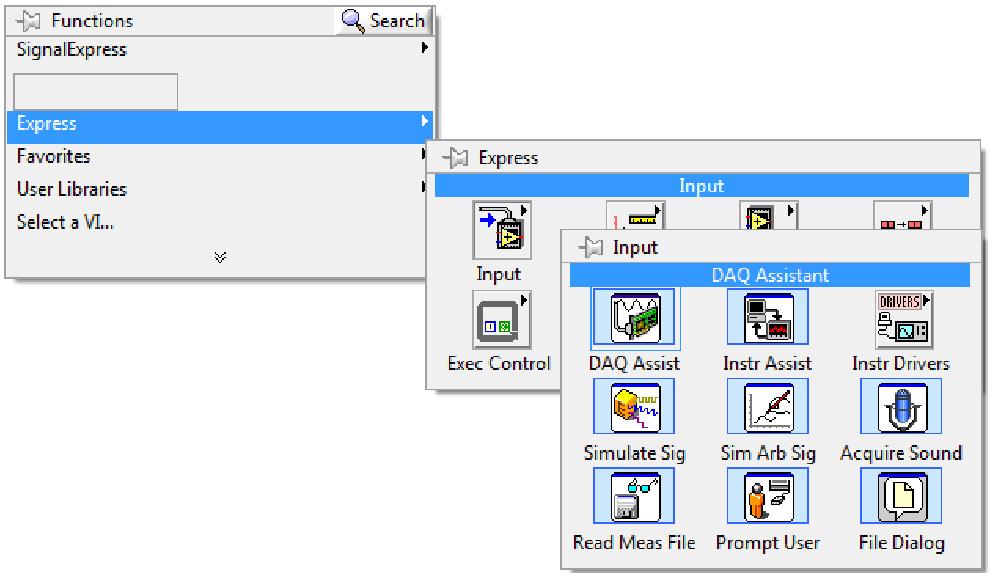
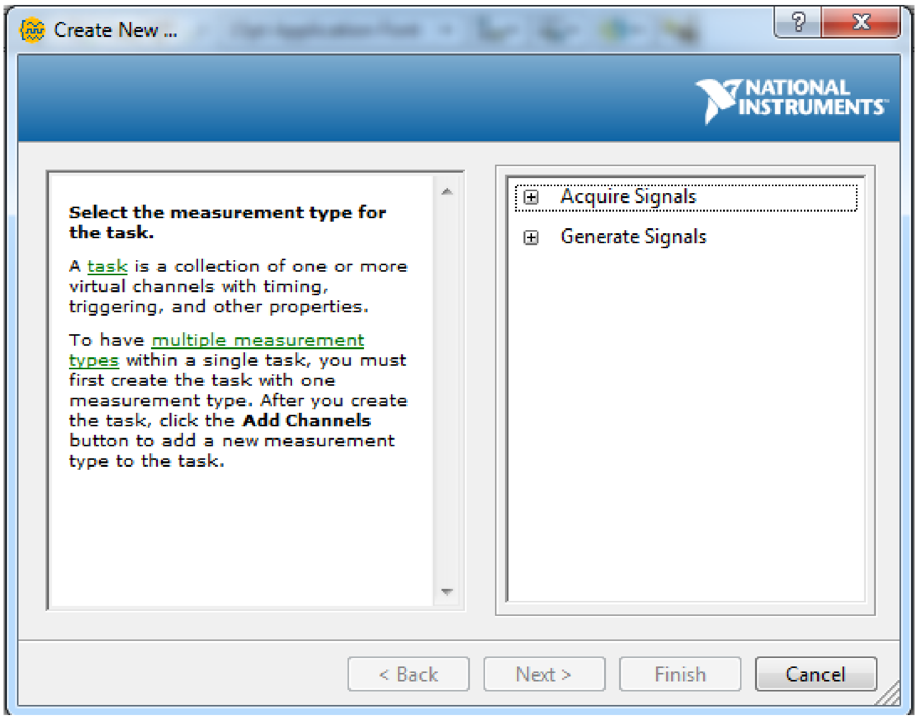


Figure 1.14: Inserting a DAQ Assistant

The following dialog box will open:



Select the measurement type for the task.

A task is a collection of one or more virtual channels with timing, triggering, and other properties.

To have multiple measurement types within a single task, you must first create the task with one measurement type. After you create the task, click the **Add Channels** button to add a new measurement type to the task.

Figure 1.15: DAQ Assistant Dialog Box

- Select **Acquire Signals** to read inputs such as sensors and push buttons.
- Select **Generate Signals** to write output signals such as operating leds or drive circuits.

2. Acquiring signal

Selecting "Acquire signal" will open the following options:

- Analog inputs:

You can select the proper analog input based on the measurement you are making provided that your **Data Acquisition Card** supports that type of measurement.

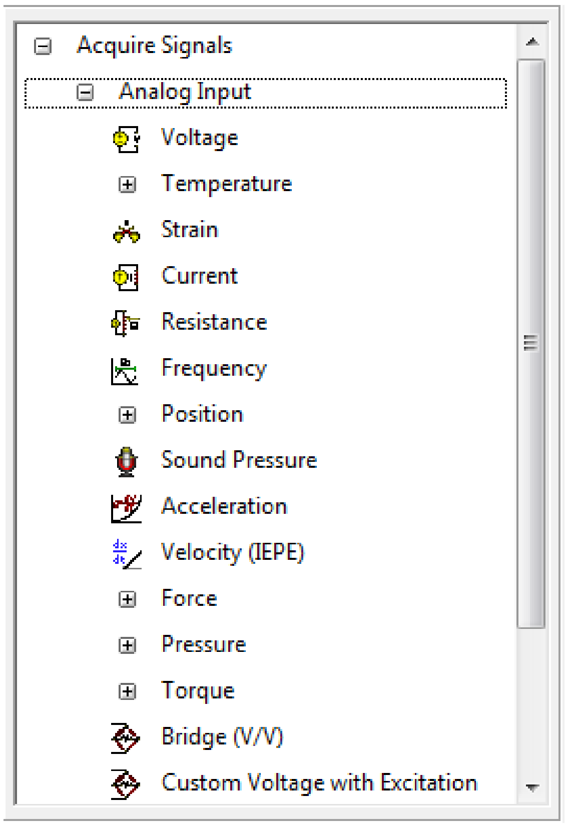


Figure 1.16: Analog Signals

- Counter input:

You can select the proper counter input based on the measurement you are making provided that your **Data Acquisition Card** supports that type of measurement. Counter input maybe used with encoders.

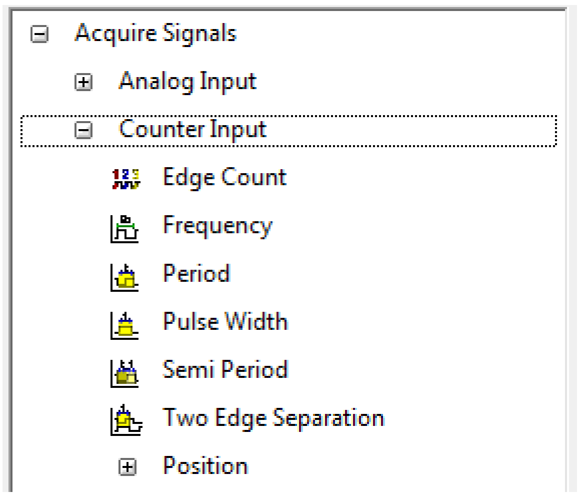


Figure 1.17: Counter signals

- Digital inputs

You can select the proper digital input based on the measurement you are making provided that your **Data Acquisition Card** supports that type of measurement. Digital inputs have only two types: either configure the DAQ as a port where all 8 pins are dealt with as one single input or you can deal lines where each pin is used as a digital input.

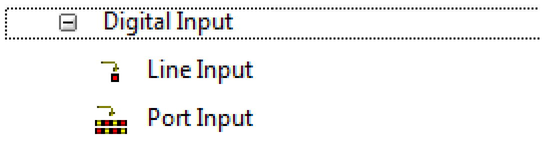


Figure 1.18: Digital Signals

The following inputs are obtained when configuring an *analog voltage input*:

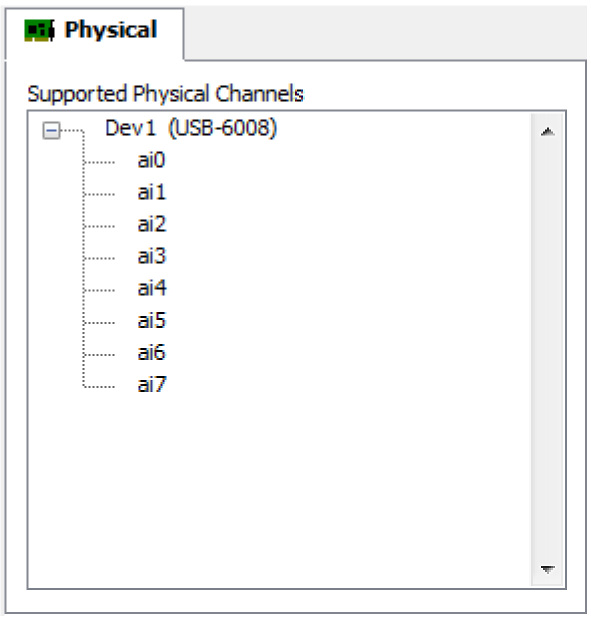


Figure 1.19: Analog Input Voltage Channals

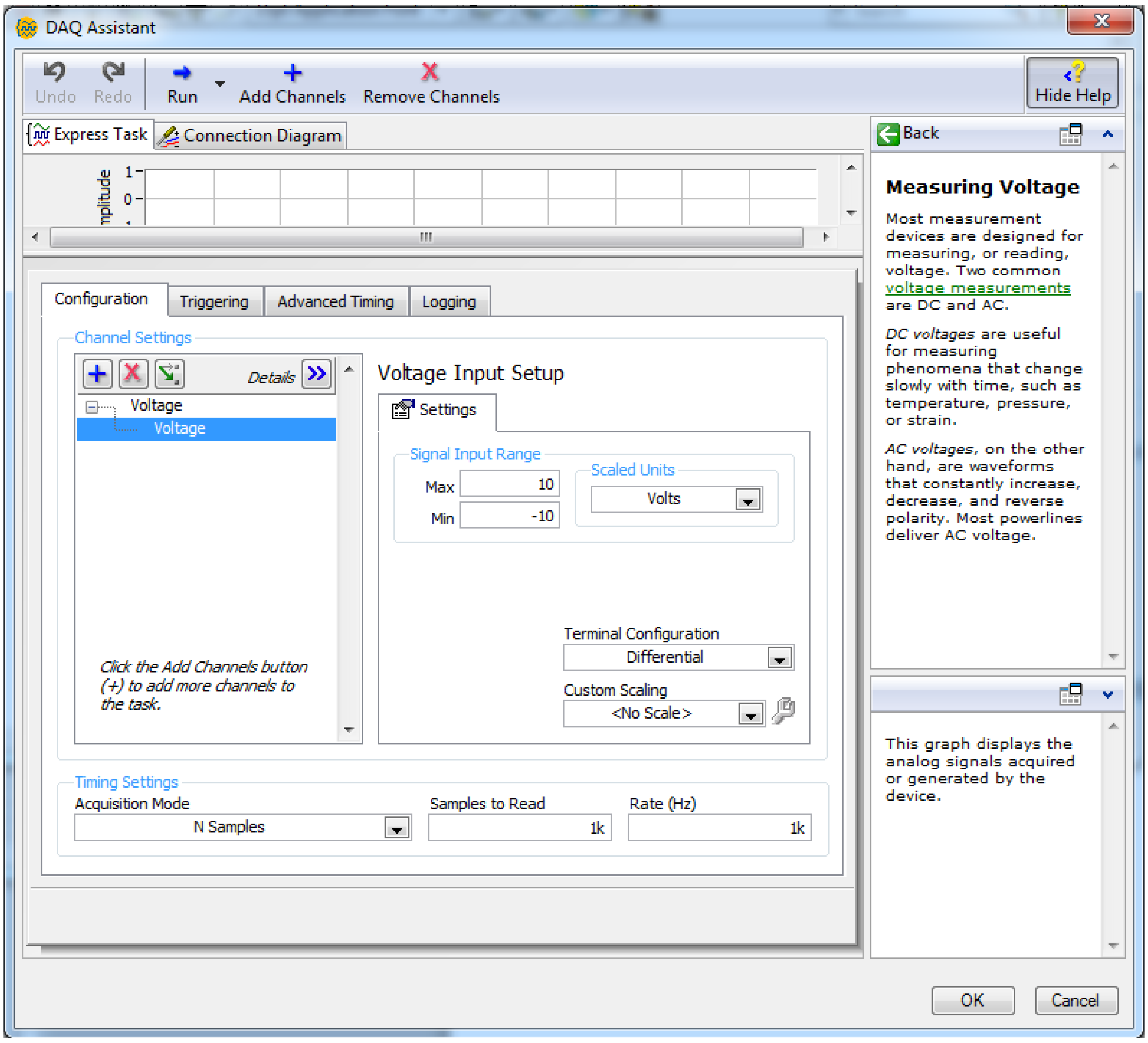
Select the channel your hardware connected to from ai0 to ai7 and click **Finish**.

1. The following dialog box will open where you have to fill the following:

   - **Max:** maximum value of the input voltage you are reading as long as it doesn’t exceed the maximum input voltage that can be read by the DAQ.

   - **Min:** minimum value of the input voltage you are reading as long as it doesn’t exceed the minimum input voltage that can be read by the DAQ.

   - **Terminal configuration**: select "Differential" for differential inputs and "RSE" for single ended inputs.



Channel Settings

Voltage
Voltage

Click the Add Channels button (+) to add more channels to the task.

Voltage Input Setup

Settings

Signal Input Range

Max 10

Min -10

Scaled Units: Volts

Terminal Configuration: Differential

Custom Scaling: <No Scale>

Timing Settings

Acquisition Mode: N Samples

Samples to Read: 1k

Rate (Hz): 1k

Measuring Voltage

Most measurement devices are designed for measuring, or reading, voltage. Two common voltage measurements are DC and AC.

DC voltages are useful for measuring phenomena that change slowly with time, such as temperature, pressure, or strain.

AC voltages, on the other hand, are waveforms that constantly increase, decrease, and reverse polarity. Most powerlines deliver AC voltage.

This graph displays the analog signals acquired or generated by the device.

Figure 1.20: Configuring Analog Input Voltage

- **Acquisition mode:** it can be "one sample" where the DAQ reads only one sample and the update rate depends on the timing of the program. You can select "N samples" where a predefined number of samples "N" is read, then the program is updated with all of the readings at once. To read another "N" samples, you have to put your DAQ inside a loop. "Continuous samples" is equivalent to "N sample" inside a loop.

- **Samples to read (N):** what is the total number of samples you want your program to get updated with after each reading is finished.

- **Rate (Hz):** how frequent you want your samples to get read. Its unit is S/s (sample per second). i.e. during 1sec how many samples you want read.

How to calculate the time needed to get an update in the program (total time needed to read all the samples)?

This time is calculated as follows:

Time=N/Hz

If you want to read 1000 sample and your rate is 1KHz then you will get an update each 1sec with 1000 sample read.

If you want to read 100 sample and your rate is 1KHz then you will get an update each 0.1 sec with 100 sample read.

Notes:

- **The Express Task at the top of the dialog box is used to test the acquired or the generated signal before finishing the configuration.**
- **Dealing with digital inputs is not much different but will less options.**

3. Generating signal

Generating signals is not much different except that it has less options since mostly the actuators has much less operating signals that the variety of input signals that can be generated by measurements.

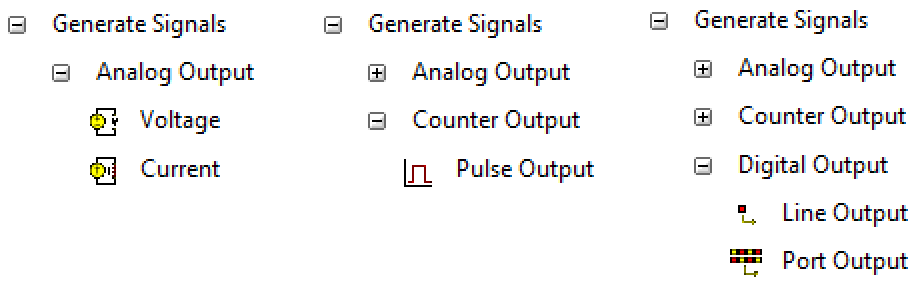


Figure 1.21: Generating Output Signals

It is recommended to have 6 DAQ at max. one for all the analog inputs, one for all the analog outputs, one for all the digital inputs, one for all the digital outputs, one for all the counter inputs and one for all the counter outputs.

4. Dynamic Data Type

When acquiring signals from or generating signals to the actual world, these signals are called dynamic data type. Ordinary indicators and controls cant deal with such data type. You will need the following convertors:

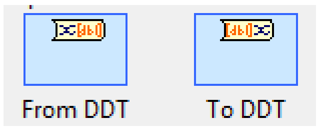
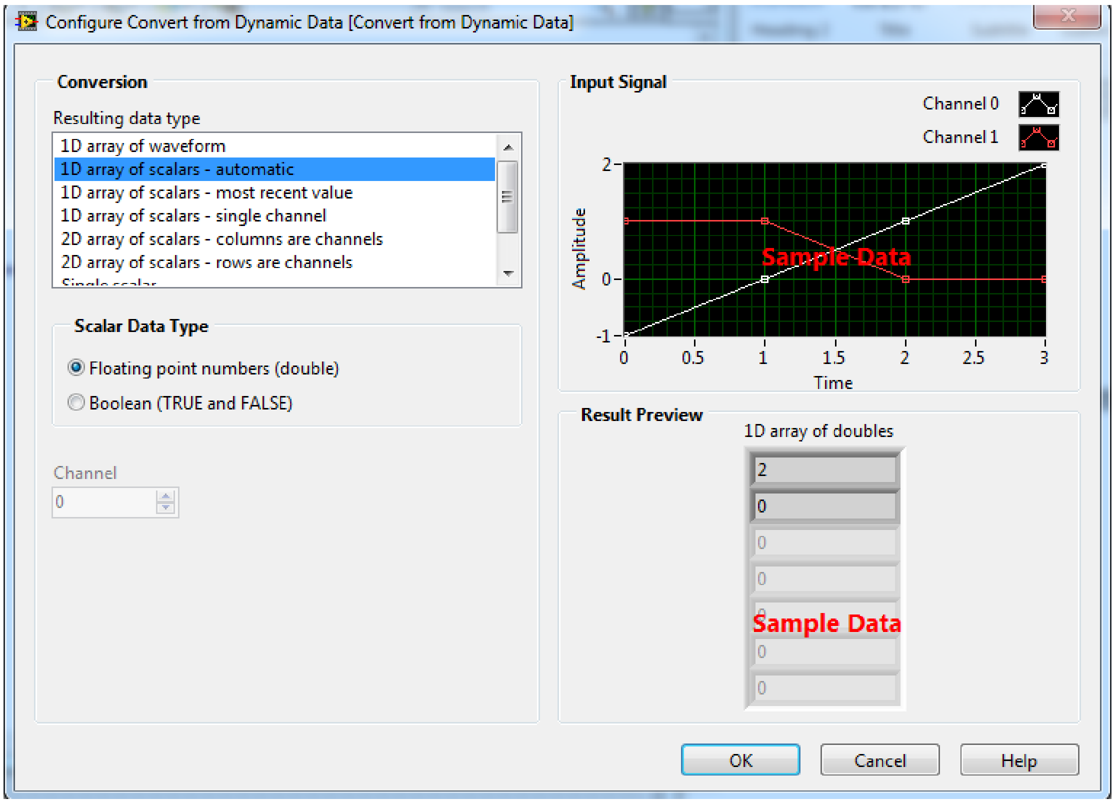


Figure 1.22: Dynamic Data Type Converters

To obtain them, right click the block diagram the select **Express** from the drop down menu >> **signal manipulation**. The following dialog box will open when you insert DDT:


Configure Convert from Dynamic Data [Convert from Dynamic Data]

Conversion

Resulting data type

- 1D array of waveform
- 1D array of scalars - automatic
- 1D array of scalars - most recent value
- 1D array of scalars - single channel
- 2D array of scalars - columns are channels
- 2D array of scalars - rows are channels
- Single scalar

Scalar Data Type

- (●) Floating point numbers (double)
- ( ) Boolean (TRUE and FALSE)

Channel: 0

Figure 1.23: Configuring "From DDT"

Floating is selected when dealing with analog data while Boolean is selected for digital data. The **Resulting data type** is selected to meet your demand mostly for the lab application it is a **Single scalar**.

Procedure:

1. Design the LabVIEW program to meet the requirements of the exercise provided by your lab supervisor.
2. Run and test your program.
3. Save it with the proper name on the proper directory.

Discussion and Analysis:

1. Based on the front panel and block diagram below, if all default values are zeroes, answer the related questions:

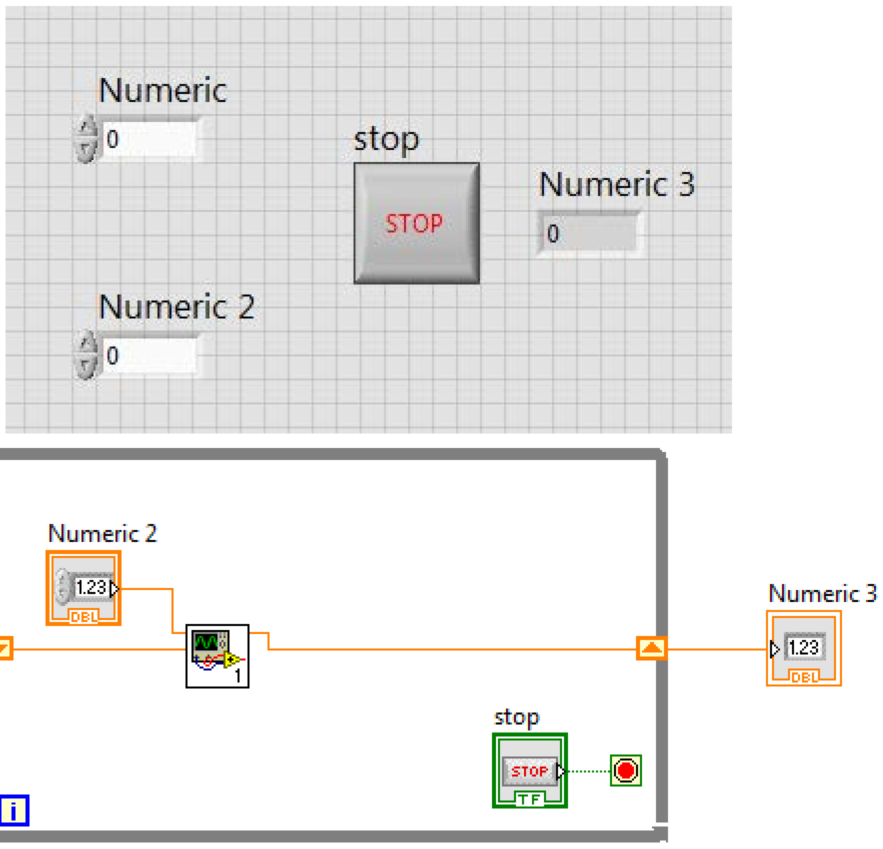


Figure 1.24: Front Panel and Block Diagram for Question 1

1. How many controls are there in the Figure?
2. How many indicators are there in the Figure?
3. What will happen if "Stop" push button is pressed after three iterations?
4. Locate the subVI in the Figure?

References:

1- http://www.ni.com
2- LabVIEW^{TM} Core 1 Course Manual

Experiment 2
Data Acquisition

Objectives:

- Familiarize the student with the concept of Data Acquisition Systems.
- Familiarize the student with connecting analog input/outputs, digital inputs/outputs and counter inputs/outputs to the DAC.
- Learning the basic signal conditioning, interfacing and drive circuits required with DAQ.

Apparatus:

The systems used during this experiment are Data Acquisition Cards (DAQ) including USB 6008 DAQ.

NATIONAL INSTRUMENTS
NI USB-6009
8 Inputs, 14-bit, Multifunction I/O
Analog
Digital

Figure 2.1: USB 6008 DAQ

Theoretical Background:

What is DAQ?

Data acquisition (DAQ) is the process of measuring an electrical or physical phenomenon such as voltage, current, temperature, pressure, or sound with a computer. A DAQ system consists of sensors, DAQ measurement hardware, and a computer with programmable software. Compared to traditional measurement systems, PC-based DAQ systems exploit the processing power, productivity, display, and connectivity capabilities of industry-standard computers providing a more powerful, flexible, and cost-effective measurement solution.

Data acquisition applications are usually controlled by software programs developed using various general purpose programming languages such as Assembly, BASIC, Fortran, LabVIEW, Java, …etc. Stand-alone data acquisition systems are often called data loggers.

A complete data acquisition system consists of:

- Sensors and Actuators (Plant).
- Signal Conditioning circuit (which in modern systems is integrated within the DAQ hardware).
- DAQ Measurement Hardware (Interface Devices).
- DAQ Software and Device Driver.

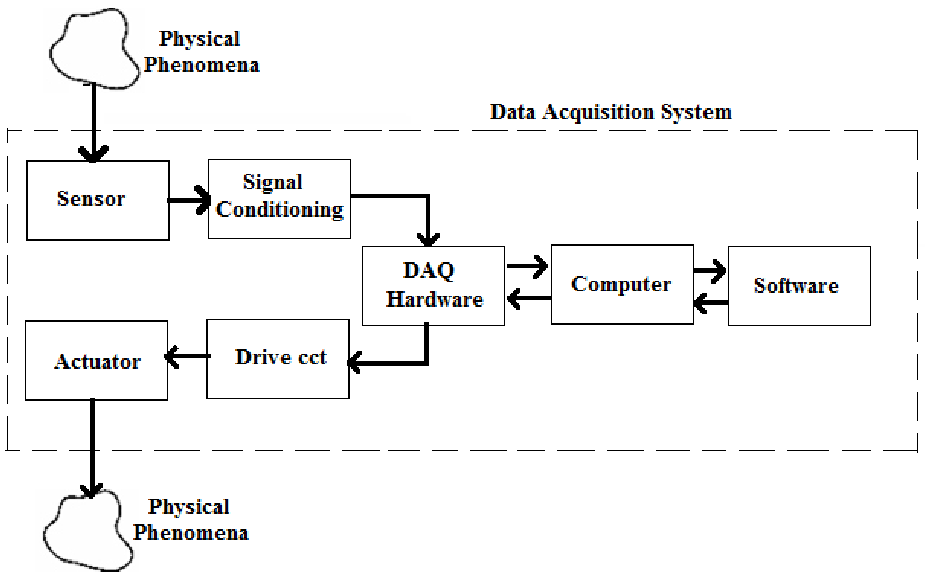


Figure 2.2: Data Acquisition System

<u>Different types of DAQ cards available in the mechatronics system design laboratory include:</u>

PCI DAQ, USB DAQ, Compact DAC and Compact RIO.

Part 1: Connecting Analog Current Inputs

NI DAQ devices measure current by means of an internal, precision shunt resistor. Most devices can measure ± 20 mA, which maps to the output range of most current transducers and 4 mA to 20 mA current loops. Before making signal connections, check the specifications of your device. When making signal connections, you must be careful with the voltage level of the signal going into the device.

You can make two types of connections:

- <u>Differential Measurements (Current Loop Measurements)</u>

Devices capable of different current measurements have two terminals per measurement channel: AI+ and AI-. Connect the positive side of the current source to the AI+ terminal, and connect the negative side of the current source to the AI– terminal.

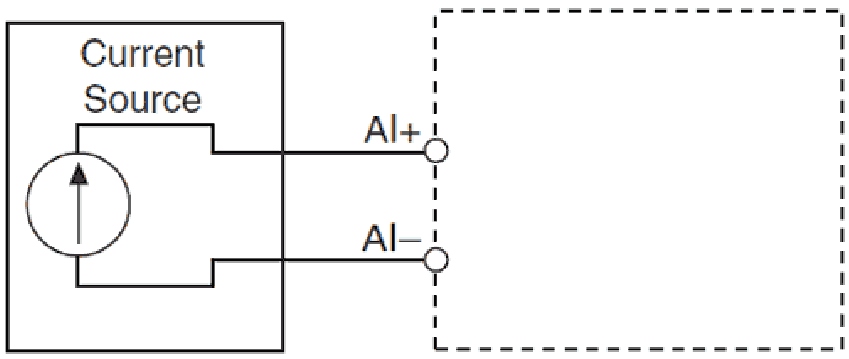


Figure 2.3: Connecting a Differential Current Measurement

This measurement is made using a DAQ containing a **shunt resistor**. The main components of a current loop include a DC power supply, transducer, a data acquisition device, and wires connecting them together in a series, as shown below.

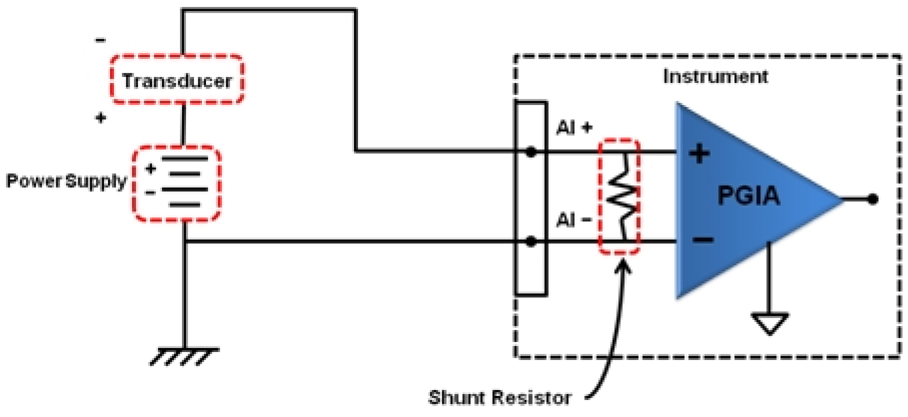


Figure 2.4: Current Loop System

- <u>Single-Ended Measurements (3-wire current transducer)</u>

Devices capable of single-ended measurements have one terminal per measurement channel (AI), a supply terminal and a common ground terminal (COM). Connect the positive lead of the current signal to the AI terminal. Connect the negative lead to the COM terminal.

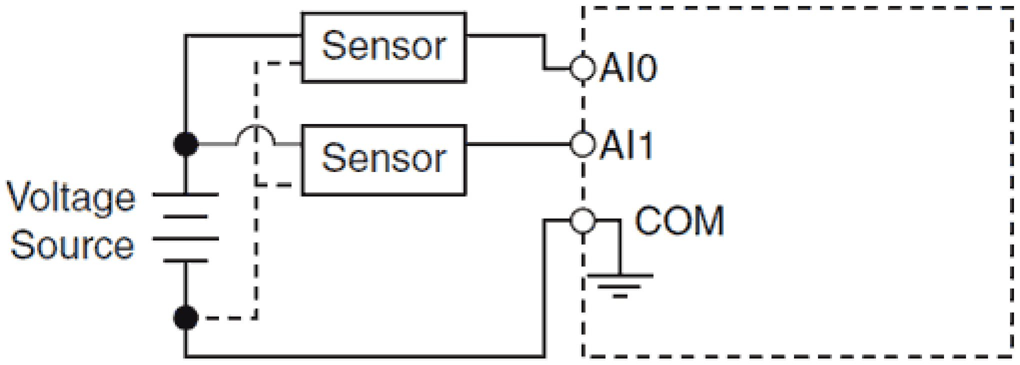


Figure 2.5: Connecting a Single-Ended Current Measurement

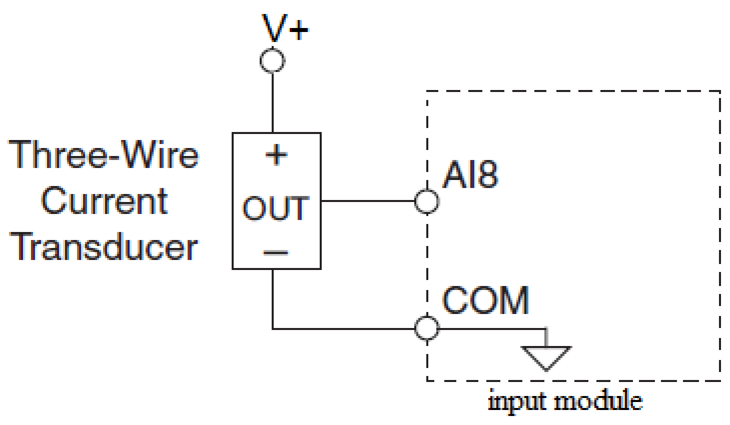


Figure 2.6: 3-Wire Current Transducer

Part 2: Connecting Analog Voltage Inputs

- Floating Vs. Referenced Signal Sources

  - **Floating Signal Sources**

A floating signal source is not connected to the building ground system but has an isolated ground reference point. Some examples of floating signal sources are outputs of **transformers, thermocouples, battery-powered devices, optical isolators, and isolation amplifiers.** An instrument or device that has an isolated output is a floating signal source. The ground reference of a floating signal must be connected to the ground of the device to establish a local or onboard reference for the signal.

  - **Ground Referenced Signal Sources**

A ground referenced signal source is connected to the building system ground, so it is already connected to a common ground point with respect to the device, assuming that the **measurement device is plugged into the same power system as the source**. Non-isolated outputs of instruments and devices that plug into the building power system fall into this category.

- Single Ended Vs. Differential

  - **Single ended Sources**

The voltage is measured between a point and the ground. This type of configuration allows for the DAQ system to support more channels but is susceptible to noise from the common ground. **Examples for this type include outputs of NPN sensors (such as proximity), outputs of amplifier circuits and outputs of voltage divider circuits.**

  - **Differential Sources**

The voltage is measured between two individual points within a common mode range. The measurement is independent of the ground which makes it more immune to noise. The reason for this is that the noise source is present on both signal leads and when in differential mode only the difference between the two leads is measured. **Examples for this type include outputs of bridge circuits, thermocouple and strain gauges.**

The figure below shows the different types of signal source types as well as the optimal connection diagrams based on the individual measurement method. Please note that depending on the type of signal, a particular voltage measurement method may provide better results than others.

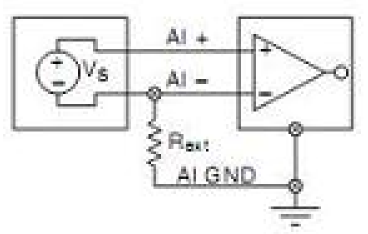
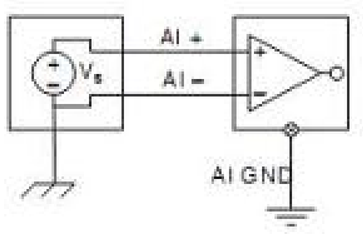
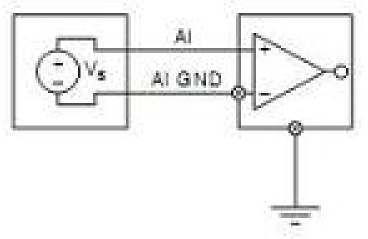
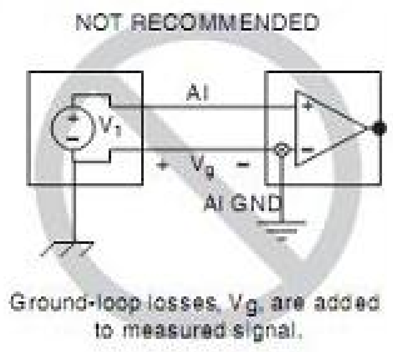
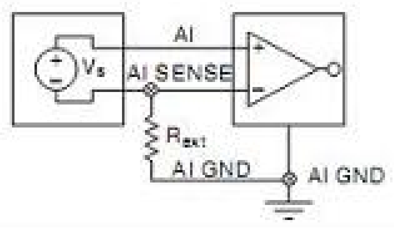
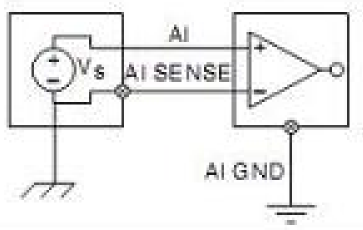
	Signal Source Type	
	Floating Signal Sources (Not Connect To Building Ground)	**Ground-Referenced Signal Sources**
Input	Examples • Ungrounded thermocouples • Signal conditioning with isolated outputs • Battery devices	Examples • Plug-in instruments with non-isolated outputs
Differential (DIFF)		
Single-Ended—Ground Referenced (RSE)		 NOT RECOMMENDED V1 AI + Vg – AI GND Ground-loop losses, Vg, are added to measured signal.
Single-Ended—Non-Referenced (NRSE)		

Figure 2.7: Optimal Connection Diagrams for Different Types of Sources

Part 3: Connecting Digital Inputs/Outputs

The configurations shown below represent what the DAQ internal digital lines look like and how to deal with them. The push pull configuration (to the left) has both NPN and PNP transistors which means that it can be either a sinking or sourcing [input/output]. Either pull up or pull down resistors can be connected to it. The open collector configuration (to the right) can be used only as a sourcing [input/output]. This mode can be used only with pull up resistors.

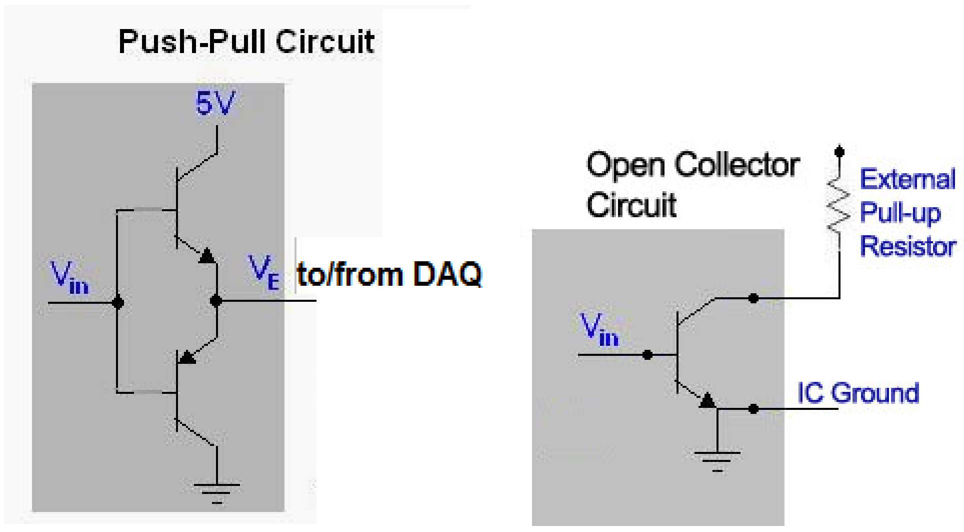
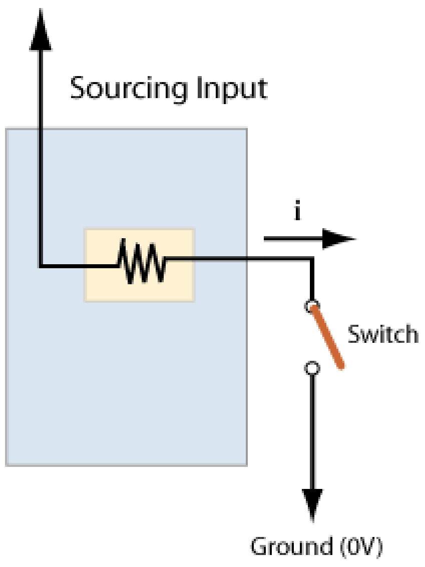
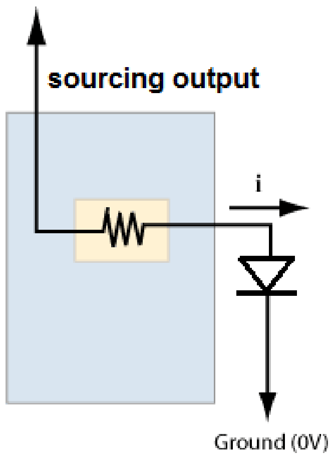
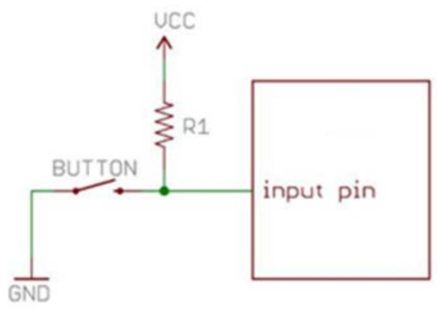
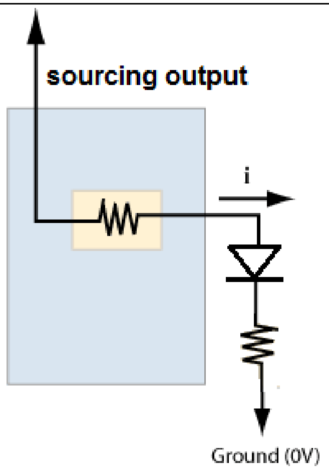
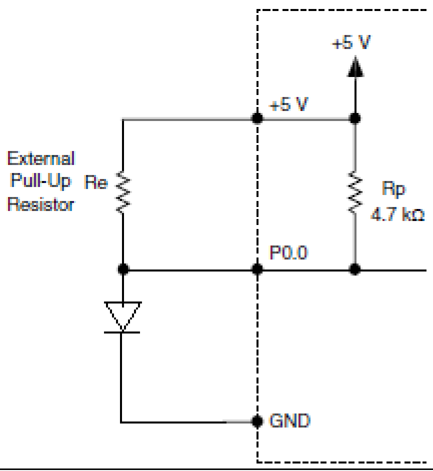


Figure 2.8: Push Pull Vs. Open Collector Configurations

The table below illustrates the different connection of digital inputs and output to DAQ based on the system current requirements.

Table 2.1: Open Collector Different connections as Input and Output

Open collector as input	**Open collector as output**
Sourcing input with internal pull up  Sourcing input with external pull up, this configuration will increase the current of the input.	Digital output without a resistor (load doesn't consume high currents)  Digital output without a resistor to decrease load current

	 Digital output with external pull up resistor to increase the load current  +5 V +5 V External Pull-Up Resistor Re Rp 4.7 kΩ P0.0 GND

Examples for digital inputs include **digital proximity sensors, push buttons and limit switches**. **LEDs, Buzzers and digital control signal** are considered as digital outputs.

Part 4: Connecting Counter Inputs/Outputs

Counter inputs/outs are basically dealt with as any digital input.

Part 5: Connecting Analog Output Voltage Signals

Connecting analog outs is mostly a simple process. You connect the output pin to the output device and the other terminal of the device to the analog GND.

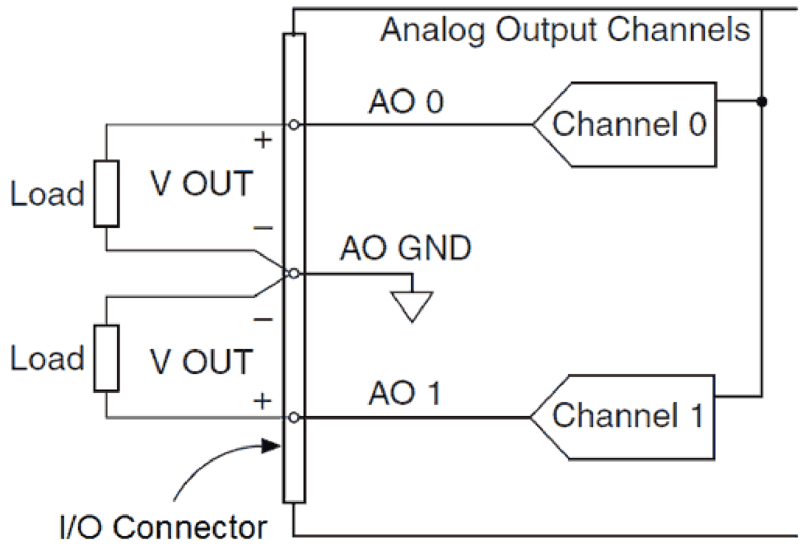


Figure 2.9: Connecting analog outputs

Sometimes the level of the voltage that operates the output device doesn't match the range of the DAQ signal. In these cases some interfacing and drive circuits maybe required such as H-bridges, inverters, optical isolators.

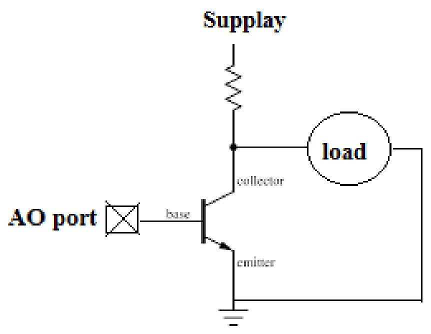
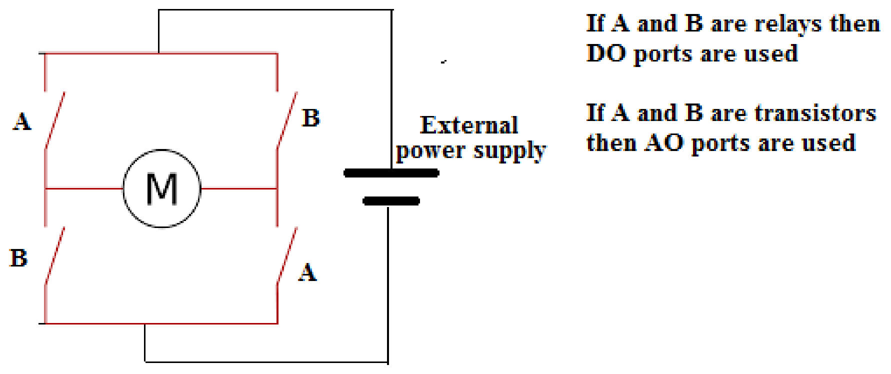
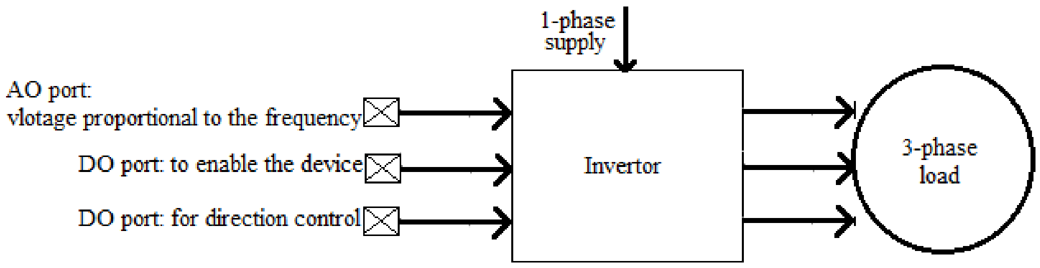


Figure 2.10: Connecting to analog outputs using transistors


A
B
M
B
A
External power supply
If A and B are relays then DO ports are used
If A and B are transistors then AO ports are used

2.11: Connecting to dc motor using H-Bridge


1-phase supply
AO port:
vlotage proportional to the frequency
DO port: to enable the device
DO port: for direction control
Invertor
3-phase load

2.12: Connecting to induction motor using Inverter

Part 6: Signal Conditioning

Many applications involve environmental or structural measurements, such as temperature and vibration, from sensors. These sensors, in turn, require signal conditioning before a data acquisition device can effectively and accurately measure the signal. Signal conditioning is one of the most important components of a data acquisition system because without optimizing real-world signals for the digitizer in use, you cannot rely on the accuracy of the measurement.

The following list offers common signal conditioning types.

<u>Amplification</u>

Amplifiers increase voltage level to better match the DAQ analog-to-digital converter (ADC) range‹ thus increasing the measurement resolution and sensitivity. Typical sensors that require amplification are thermocouples and strain gages.

<u>Attenuation</u>

Attenuation, the opposite of amplification, is necessary when voltages to be digitized are beyond the ADC range. This form of signal conditioning decreases the input signal amplitude so that the conditioned signal is within the ADC range. Attenuation is typically necessary when measuring voltages that are more than 10 V.

<u>Filtering</u>

Filters reject unwanted noise within a certain frequency range. Often, lowpass filters are used to block out noise in electrical measurements, such as 50/60 Hz power.

<u>Isolation</u>

Voltage signals well outside the range of the digitizer can damage the measurement system and harm the operator. For that reason, isolation is usually required in conjunction with attenuation to protect the system and the user from dangerous voltages or voltage spikes.

Isolation may also be needed when the sensor is on a different ground plane from the measurement sensor, such as a thermocouple mounted on an engine.

Excitation

Excitation is required for many types of transducers. For example, strain gages, accelerometers، thermistors, and RTDs require external voltage or current excitation. RTD and thermistor measurements are made with a current source that converts the variation in resistance to a measureable voltage. Accelerometers often have an integrated amplifier, which requires current excitation provided by the measurement device. Strain gages, which are very-low-resistance devices, are typically used in a Wheatstone bridge configuration with a voltage excitation source.

Linearization

Linearization is necessary when sensors produce voltage signals that are not linearly related to the physical measurement. Linearization, the process of interpreting the signal from the sensor, can be implemented either with signal conditioning or through software. A thermocouple is the classic example of a sensor that requires linearization.

Procedure:

1. Inspect the USB 6008 DAQ provided by your lab supervisor.
2. Design and implement the circuit using the components available with the DAQ, to meet the requirements the task handed to you by your supervisor.
3. Using LabVIEW, design the proper program needed to operate your circuit.
4. Run your program and test your circuit.

Discussion and Analysis:

1. Draw a schematic of the circuit you connected in the lab using Figure 2.13.

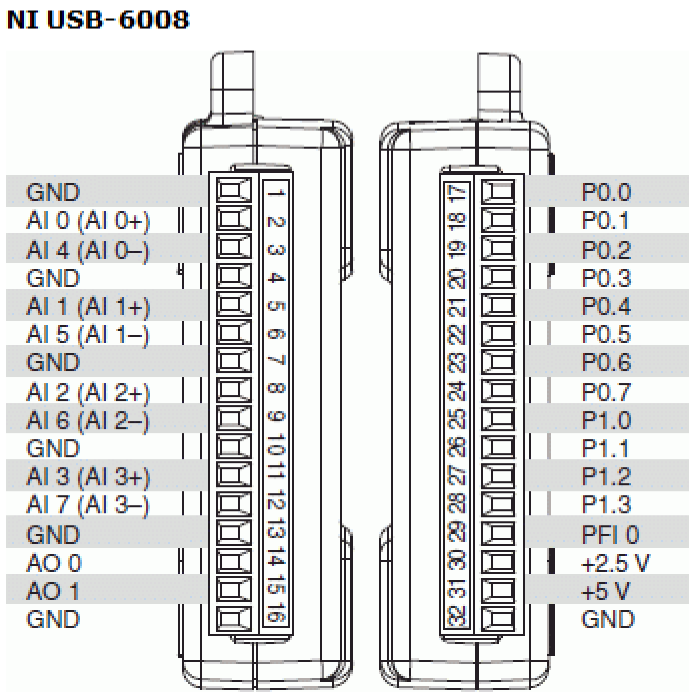


Figure 2.13: USB 6008 Pinout.

2. Show all the calculation you made in order to select the values of the external resistors added to the circuit.

References:

1- http://sine.ni.com/np/app/main/p/ap/daq/lang/en/pg/1/sn/n17:daq,n21:11/fmid/2998/
2- http://www.ni.com/tutorial/7113/en/

Experiment 3
USB 6008 DAQ

Objectives:

- Familiarize the student with USB 6008 DAQ.
- Study the basic features of the DAQ.

Apparatus:

The system used during this experiment is the USB 6008 DAQ.

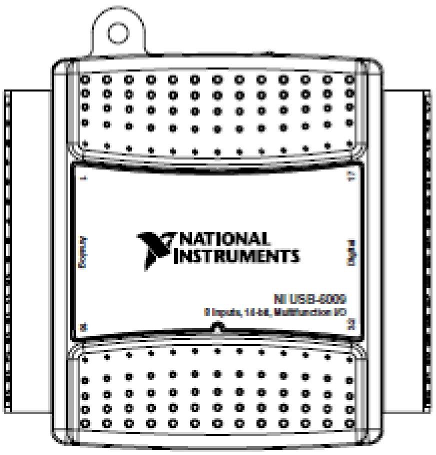


Figure 3.1: USB 6008 DAQ

Theoretical Background:

USB 6008 is the data acquisition card available in the mechatronics lab to which the students can connect their designed systems. During this experiment, the basic points related to using and connecting to USB 6008 DAQ will be illustrated by means of the data sheet of the DAQ.

The National Instruments USB-6008 is a low-cost, multifunction data acquisition device (DAQ). It has 8 analog inputs, 2 analog outputs, and 12 digital input/outputs. The digital channels are divided into two ports. When one or more channels on each port is set to either input or output, the port is locked into that particular mode.

Introduction

The NI USB-6008/6009 provides connection to eight analog input (AI) channels, two analog output (AO) channels, 12 digital input/output (DIO) channels, and a 32-bit counter when using a full-speed USB interface.

Table 3.1: Differences between the USB-6008 and USB-6009

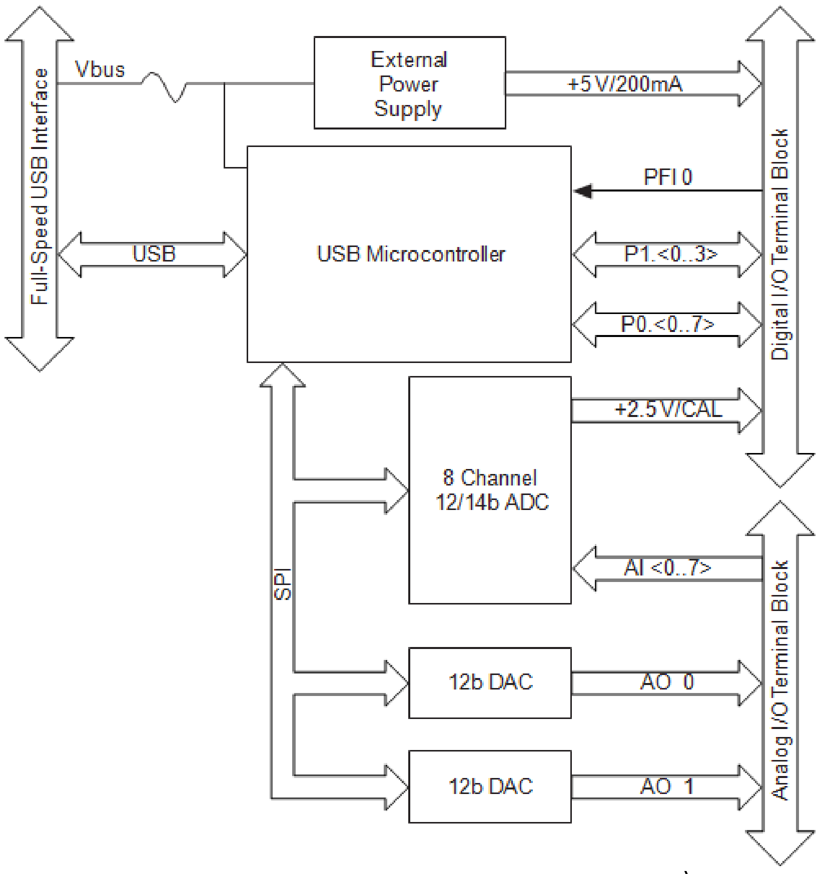
Feature	USB-6008	USB-6009
AI Resolution	12 bits differential,	14 bits differential,
Maximum AI Sample	10 kS/s	48 kS/s
DIO Configuration	Open-drain	Open-drain or push-pull

Open drain is the same as open collector

Hardware

The following block diagram shows key functional components of the USB-6008/6009.

From DAQ: the DAQ can supply any external circuit with 5 V provided that the total current consumed by all circuits doesn't exceed 200mA.



Configurable as input or output using the program

Very low current output

Analog inputs; 4 differential or 8 single ended

2 analog outputs

Figure 3.2: Device block diagram

Signal Description

Table 3.2 describes the signals available on the I/O connectors.

Table 3.2 Signal Descriptions

Signal Name	Reference	Direction	Description
GND	—	—	**Ground**—The reference point for the single-ended AI measurements, bias current return point for differential mode measurements, AO voltages, digital signals at the I/O connector, +5 VDC supply, and the +2.5 VDC reference.
AI <0..7>	Varies	Input	**Analog Input Channels 0 to 7**—For single-ended measurements, each signal is an analog input voltage channel. For differential measurements, AI 0 and AI 4 are the positive and negative inputs of differential analog input channel 0. The following signal pairs also form differential input channels: <AI 1, AI 5>, <AI 2, AI 6>, and <AI 3, AI 7>.
AO 0	GND	Output	**Analog Channel 0 Output**—Supplies the voltage output of AO channel 0.
AO 1	GND	Output	**Analog Channel 1 Output**—Supplies the voltage output of AO channel 1.
P1.<0..3> P0.<0..7>	GND	Input or Output	**Digital I/O Signals**—You can individually configure each signal as an input or output.
+2.5 V	GND	Output	**+2.5 V External Reference**—Provides a reference for wrap-back testing.
+5 V	GND	Output	**+5 V Power Source**—Provides +5 V power up to 200 mA.
PFI 0	GND	Input	**PFI 0**—This pin is configurable as either a digital trigger or an event counter input.

Analog Input

You can connect analog input signals to the USB-6008/6009 through the I/O connector. Refer to Table 3.3 for more information about connecting analog input signals.

Analog Input Circuitry

Figure 3.3 illustrates the analog input circuitry of the USB-6008/6009.

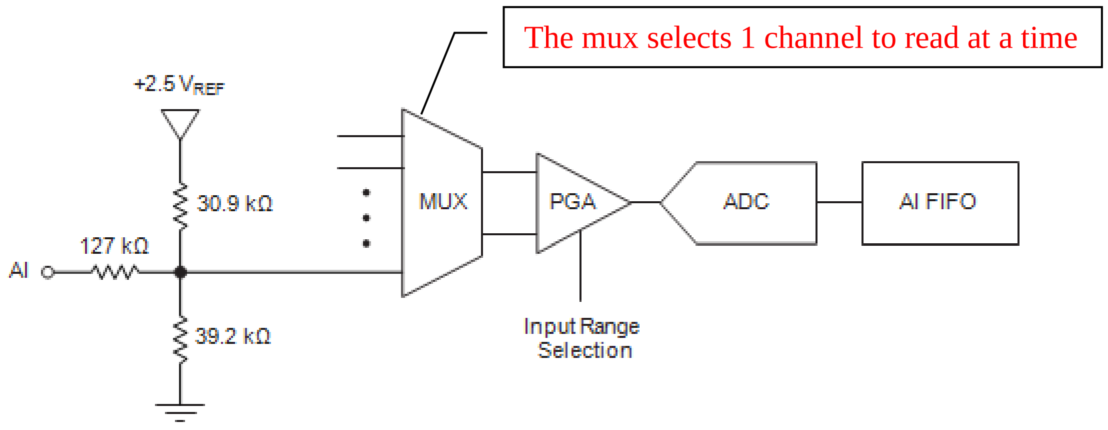


Figure 3.3: Analog Input Circuitry

- MUX

The USB 6008/6009 has one analog-to-digital converter (ADC). The multiplexer (MUX) routes one AI channel at a time to the PGA.

- PGA

The programmable-gain amplifier provides input gains of 1, 2, 4, 5, 8, 10, 16, or 20 when configured for differential measurements and gain of 1 when configured for single-ended measurements. The PGA gain is automatically calculated based on the voltage range selected in the measurement application.

- A/D Converter

The analog-to-digital converter (ADC) digitizes the AI signal by converting the analog voltage into a digital code.

- AI FIFO

The USB-6008/6009 can perform both single and multiple A/D conversions of a fixed or infinite number of samples. A first-in-first-out (FIFO) buffer holds data during AI acquisitions to ensure that no data is lost.

<u>Analog Input Modes</u>

You can configure the AI channels on the USB-6008/6009 to take single-ended or differential measurements.

- Connecting Differential Voltage Signals

For differential signals, connect the positive lead of the signal to the AI+ terminal, and the negative lead to the AI– terminal.

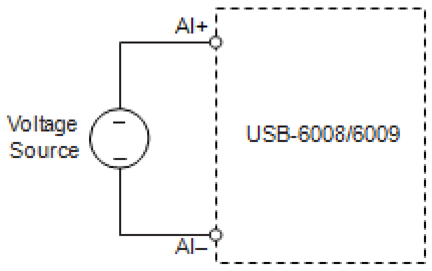


Figure 3.4: Connecting a Differential Voltage Signal

Peak to peak voltage

The differential input mode can measure ±20 V signals in the ±20 V range. However, **the maximum voltage on any one pin is ±10 V with respect to GND.** For example, if AI 1 is +10 V and AI 5 is –10 V, then the measurement returned from the device is +20 V.

- Connecting Reference Single-Ended Voltage Signals

To connect reference single-ended voltage signals (RSE) to the USB-6008/6009, connect the positive voltage signal to the desired AI terminal, and the ground signal to a GND terminal.

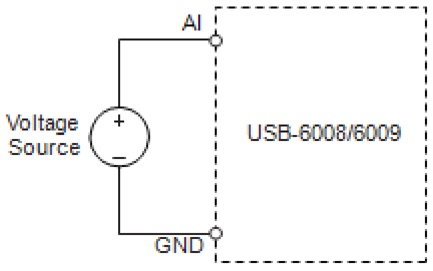


Figure 3.5: Connecting a single ended Voltage Signal

Analog Output

The USB-6008/6009 has two independent AO channels that can generate outputs from 0–5 V. All updates of AO lines are software-timed.

Analog Output Circuitry

Figure 3.6 illustrates the analog output circuitry for the USB-6008/6009.

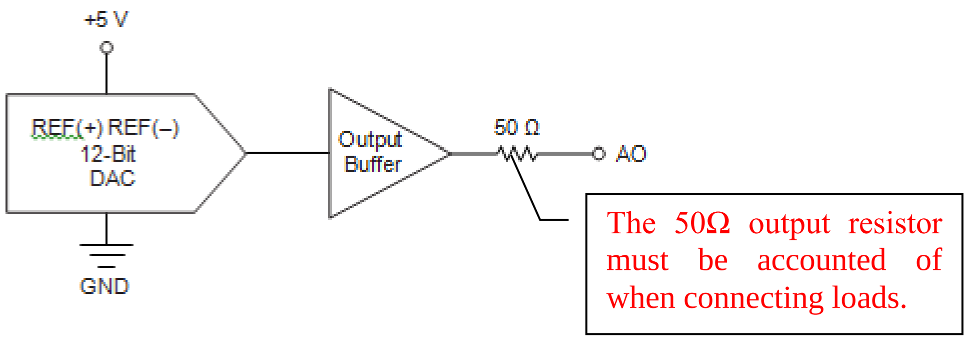

+5 V
REF(+) REF(–)
12-Bit
DAC
GND
Output
Buffer
50 Ω
AO
The 50Ω output resistor must be accounted of when connecting loads.

Figure 3.6: Analog Output Circuitry Connecting a Load

DACs

Digital-to-analog converts (DACs) convert digital codes to analog voltages.

- Connecting Analog Output Loads

To connect loads to the USB-6008/6009, connect the positive lead of the load to the AO terminal, and connect the ground of the load to a GND terminal.

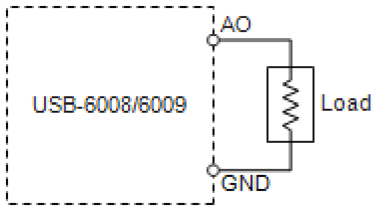


Figure 3.7: Connecting a Load

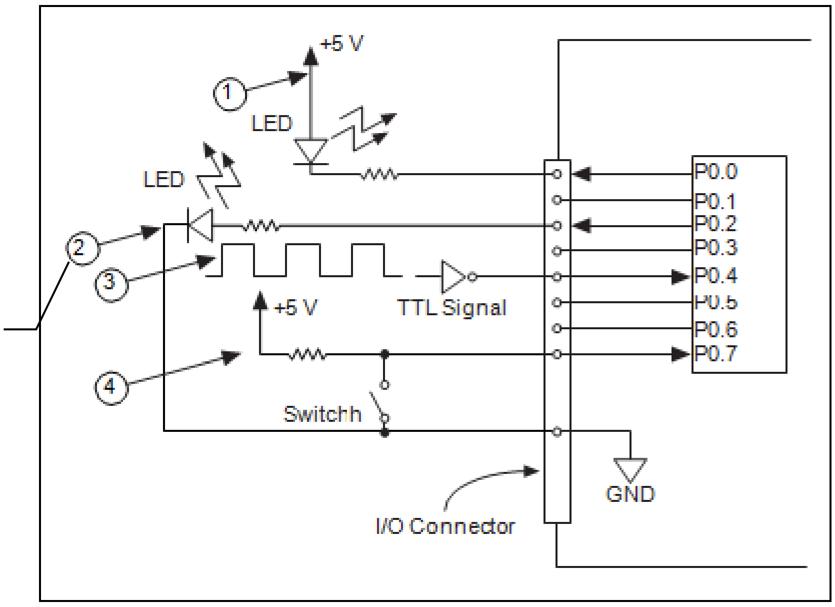
Digital I/O

The USB-6008/6009 has 12 digital lines, P0.<0..7> and P1.<0..3>, which comprise the DIO port. GND is the ground-reference signal for the DIO port. You can individually program all lines as inputs or outputs.

Digital I/O Circuitry

Figure 3.8 shows P0.<0..7> connected to example signals configured as digital inputs and digital outputs. You can configure P1.<0..3> similarly.

This configuration is not valid in DAQ 6008

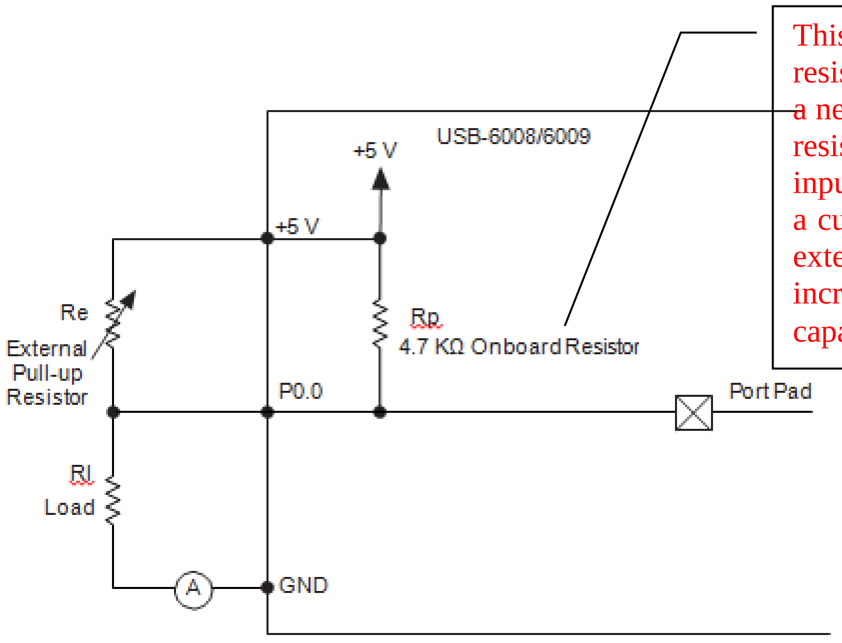


1 P0.0 configured as an open-drain digital output driving a LED
2 P0.2 configured as a push-pull digital output driving a LED
3 P0.4 configured as a digital input receiving a TTL signal from a gated invertor
4 P0.7 configured as a digital input receiving a 0 V or 5 V signal from a switch

Figure 3.7: Example of Connecting a Load

Source/Sink Information

The default configuration of the USB-6008/6009 DIO ports is **open-drain, allowing 5 V operations, with an onboard 4.7 kΩ pull-up resistor**. An external, user-provided, pull-up resistor can be added to increase the source current drive up to **a 8.5 mA limit per line** as shown in Figure 3.8.



This resistor is an internal pull resistor inside the DAQ, so it is NOT a necessity to add an external pull up resistor when connection a digital input. In this case you are limited to a current of 0.6mA.You may add an external resistor in case you want to increase the output current to its full capacity of 8.5mA.

Figure 3.8: Example of Connecting External User-Provided Resistor

Complete the following steps to determine the value of the user-provided pull-up resistor:

1. Place an ammeter in series with the load.
2. Place a variable resistor between the digital output line and the +5 V.
3. Adjust the variable resistor until the ammeter current reads as the intended current. The intended current must be less than 8.5 mA.
4. Remove the ammeter and variable resistor from your circuit.
5. Measure the resistance of the variable resistor. The measured resistance is the ideal value of the pull-up resistor.
6. Select a static resistor value for your pull-up resistor that is greater than or equal to the ideal resistance.
7. Re-connect the load circuit and the pull-up resistor.

Power-On States

At system startup and reset, the hardware sets all DIO lines to high-impedance inputs. The DAQ device does not drive the signal high or low. Each line has a weak pull-up resistor connected to it.

At floating conditions(when the program is not running and the input or output state is decided), during such cases the pull resistor will case 5V to appear at the digital pin regardless it is configured as input or output.

Event Counter

You can configure PFI 0 as a source for a gated invertor counter input edge count task. In this mode, **falling-edge events** are counted using a 32-bit counter.

Only works at falling edges and is used as input

Specifications

The following specifications are typical at 25 °C, unless otherwise noted.

Analog Input

Number of bits reflected the size of the register in the microprocessor used in the ADC process
Resolution=Vin(max)/(2^(number of bits) -1)

Resolution is defined as the minimum voltage change required to be detected by DAQ

Converter type...................................... Successive approximation

Analog inputs .... 8 single-ended/4 differential,

software selectable

Input resolution

USB-6008 .... 12 bits differential,

11 bits single-ended

USB-6009 .... 14 bits differential,

13 bits single-ended

Up to how many samples can the DAQ acquire during a second

Max sampling rate

USB-6008 .... 48 kS/s

USB-6009 .... 10 kS/s

S/s: Sample per second

Input range

Single-ended .... ±10 V

Differential.................................... ±20 V, ±10 V, ±5 V, ±4 V,

Peak to peak voltage

±2.5 V, ±2 V, ±1.25 V, ±1 V

Working voltage.................................... ±10 V

Input impedance .... 144 kΩ

Overvoltage protection.......................... ±35

Input impedance: high input impedance which means that the DAQ will not draw current from analog input devices reducing the loading effect (ideally this value should be infinity

Analog Output

Converter type ....Successive approximation

Analog outputs 2

Output resolution ....12 bits

Resolution= Vout(max)/((2^12)-1)

Maximum update rate ....150 Hz, software-timed

Power-on state ....0 V

Output range ....0 to +5 V

Output impedance..................................50 Ω

Output current drive...............................5 mA

Each analog output has an on-board 50Ω resistor and can drive loads with a maximum current of 5mA.

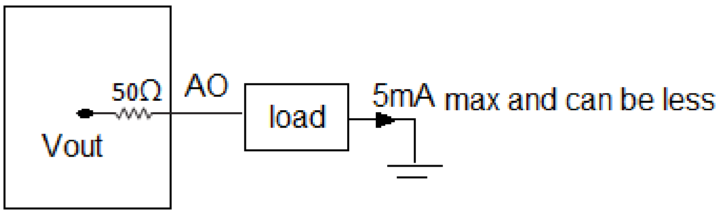


Figure 3.9: Analog output calculations

Digital I/O

P0.<0..7>....................................... 8 lines

PI.<0..3> .... 4 lines

Direction control .... Each channel individually

programmable as input or output

Output driver type

USB-6008 .... Open-drain Open collector

USB-6009 .... Each channel individually

programmable as push-pull or open-drain

Compatibility .... TTL, LVTTL, CMOS

Absolute maximum voltage range .... –0.5 to 5.8 V with respect to GND

Pull-up resistor .... 4.7 kΩ to 5 V

Power-on state...................................... Input (high impedance)

Table 3.3 Signal Descriptions

Level	Min	Max	Units
Input low voltage	–0.3	0.8	V
Input high voltage	2.0	5.8	V
Input leakage current	—	50	µA
Output low voltage (I = 8.5 mA)	—	0.8	V
Output high voltage			
Push-pull, I = –8.5mA	2.0	3.5	V
Open-drain, I = –0.6mA, nominal	2.0	5.0	V
Open-drain, I = –8.5mA, with external pull-up resistor	2.0	—	V

When used as input

Not supported by DAQ 6008

When used as output

Only the 4.7KΩ onboard

Counter

Number of counters

Resolution ....32 bits

Counter measurements ....Edge counting (falling-edge)

Pull-up resistor.......................................4.7 kΩ to 5 V

Maximum input frequency ....5 MHz

Bus Interface

USB specification ....USB 2.0 full-speed

USB bus speed.......................................12 Mb/s

Power Requirements

USB 4.10 to 5.25 VDC............................80 mA typical, 500 mA max

USB suspend ....300 µA typical, 500 µA max

External Voltage

+5 V output (200 mA maximum) .... +5 V typical, +4.85 V minimum

+2.5 V output (1 mA maximum) .... +2.5 V typical

Very low current; the supply is used for self test only

Procedure:

1. Inspect the USB 6008 DAQ provided by your lab supervisor.
2. Design and implement the circuit using the components available with the DAQ, to meet the requirements of your design problem.
3. Using LabVIEW, design the proper program needed to operate your circuit.
4. Run your program and test your circuit.

Discussion and Analysis:

1. Draw a schematic of the circuit you connected in the lab using Figure 3.10.

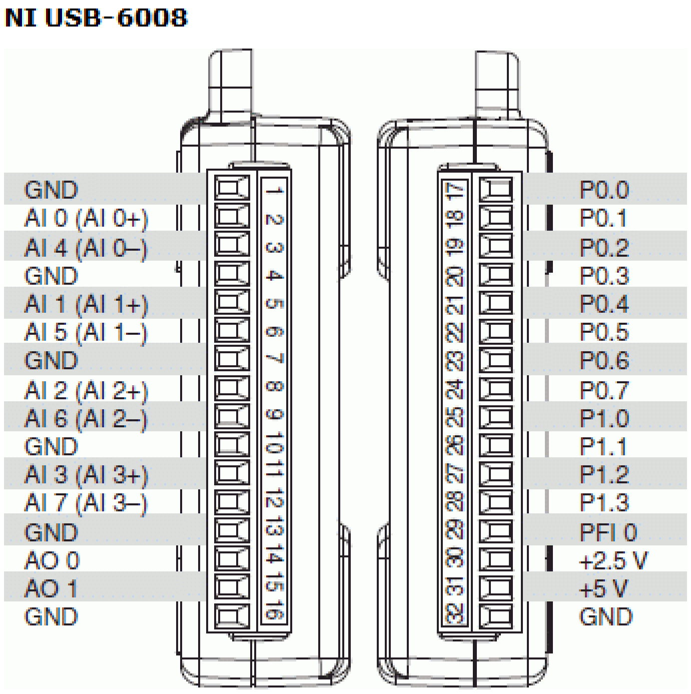


Figure 3.10: USB 6008 Pinout.

2. Show all the calculation you made in order to select the values of the external resistors added to the circuit.

Experiment 4
Process Trainer System

Objectives:

- Familiarize the student with the process trainer system (PT001).
- Lean how to model simple level and flow control systems.

Apparatus:

The system used during this experiment is the PT001 System.

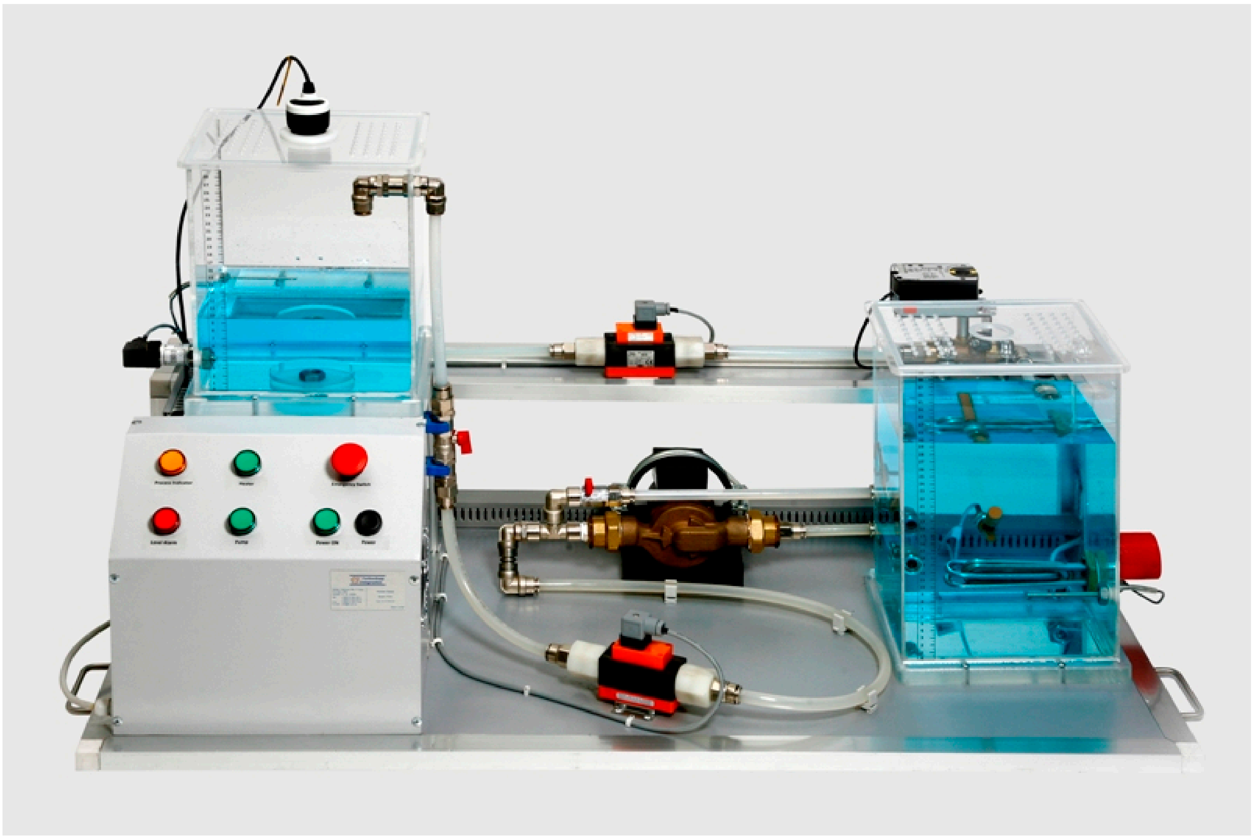


Figure 4.1: PT001 Trainer

Theoretical Background:

The PT001 Process Trainer is a bench-mounted, Process Trainer for Level, Flow, Temperature, and Pressure control. It comprises all the required sensors to perform the many control operations.

1. Hardware:

a. Description:

The setup consists of the following components:

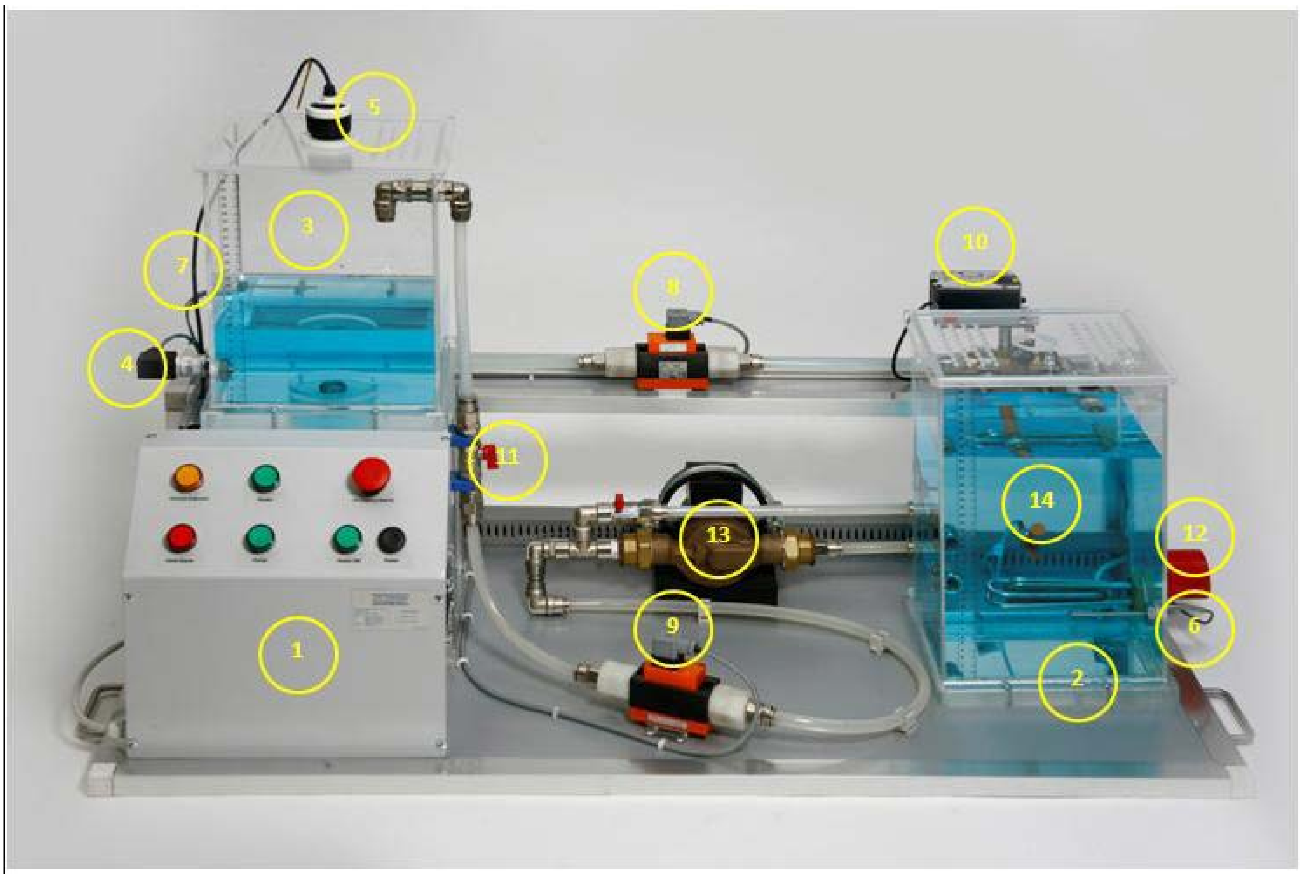


Figure 4.2: Indicated Components for the Process Trainer System

Table 4.1: The Components of the Process Trainer System

1 . Electric Control Box	8. Flow Meter (Upper)
2. Tank 1	9. Flow Meter (Lower)
3. Tank 2	10. Electric Flow Control Valve
4. Pressure Sensor	11. Manual Flow Control Valve
5. Ultrasonic Level Meter	12. Heater
6. Thermocouple Tank 1	13. Pump
7. Thermocouple Tank2	14. Level Switch

The main buttons of the Electric Control Box are as follows:

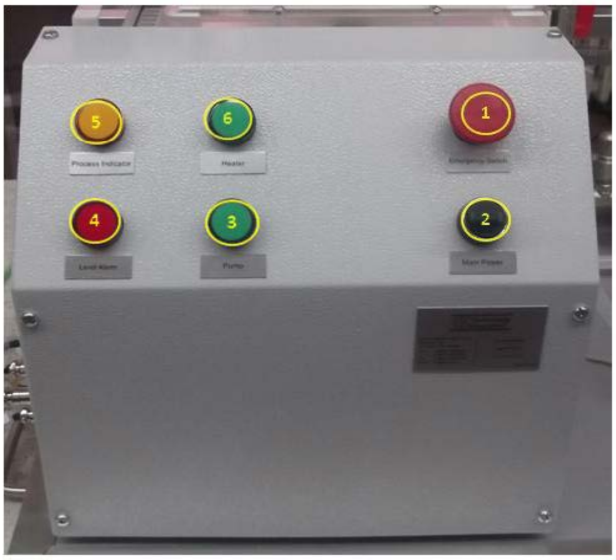


Figure 4.3: Indicated Components for the Process Trainer System

Table 4.2: The Components of the Process Trainer System

1. Emergency Switch	4. Level Alarm
2. Main Power	5. Process Indicator
3. Pump Indicator	6. Heater Indicator

b. Specifications:

Table 4.3 summarizes the specification related to the sensors and actuators of the system. These specifications are important when you want to configure the DAQ assistant in the LabVIEW programs need to run the process trainer.

Table 4.3: Sensors and Actuators specifications

1. Pressure Sensor	**2. J-Type Thermocouple**
✓ Measuring Range: 0 to 0.1 bar ✓ Supply Voltage: 24 VDC ✓ Output: 4 - 20mA ✓ Temperature Range: 0 - 80 ºC ✓ Media: Water ✓ Housing: Stainless Steel ✓ Accuracy: ±0.5% of the full range	✓ Type: J-Type ✓ With Protective Tube ✓ Immersion Length: 10 cm

3. Ultrasonic Level Meter ✓ Measuring Range: 5 to 30 cm ✓ Supply Voltage: 24 VDC ✓ Output: 4 - 20mA ✓ Temperature Range: -40 - 71 ºC ✓ Accuracy: ±0.15% of span in air ✓ Dead Band: 5 cm	**4. Flow Control Valve** ✓ Supply Voltage: 24 VAC ✓ Control Signal: 0 – 10 VDC ✓ Feedback Signal: 0 - 10 VDC ✓ Angle of Rotation: 0º - 95º ✓ Running Time: 108 sec
5. Level Switch ✓ Temperature: -10º - 50º ✓ Pressure: 100 psig	**6. Heater** ✓ Power: 1200 W ✓ Supply Voltage: 220 VAC, Single Phase
7. Pump ✓ Power: 0.17 HP ✓ Supply Voltage: 220 VAC, Single Phase ✓ Maximum Fluid Temperature: 100 ºC	**8. Variable Speed Drive** ✓ Motor Output Rating: 0.5 HP ✓ Supply Voltage/ Phases: 220/ 1Ø ✓ Output Voltage/ Phases: 0-240/ 1Ø ✓ Supply Frequency: 48 to 62 Hz ✓ Max. Ambient Temperature 50 ºC ✓ Storage Temperature -40 - 60 ºC
9 . Inline Flow sensors ✓ Measuring Range: 1.6-32 L/min ✓ Supply Voltage: 24 VDC ✓ Output: 4-20 mA ✓ Temperature Range: 0 - 60 ºC	**10. Plexiglas Tanks** ✓ Maximum Medium Temperature: 70 ºC ✓ Medium: Water

c. Data Acquisition:

The acquisition system is done with sensors and actuators are connected to a Compact DAQ. Compact DAQ USB chassis provide the plug-and-play simplicity of USB to sensor and electrical measurements on the benchtop, in the field, and on the production line. By combining more than 50 sensor-specific NI C Series I/O modules with patented NI Signal Streaming technology, the NI Compact DAQ platform delivers high-speed data and ease of use in a flexible, mixed-measurement system. The DAQ system in the lab is composed from a chassis and 4 modules as illustrated below.

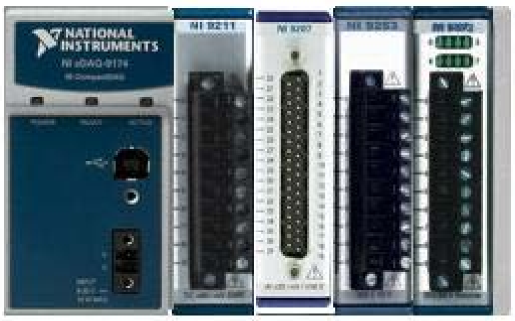


Figure 4.4: cDAQ & Its Modules

1. <u>National Instruments, NI CompactDAQ 4-Slot USB Chassis (NI cDAQ-9174)</u>

- Choose from more than 50 hot-swappable I/O modules with integrated signal conditioning
- Four general-purpose 32-bit counter/timers built into chassis (access through digital module)
- Run up to seven hardware-timed analog I/O, digital I/O, or counter/timer operations simultaneously
- Stream continuous waveform measurements with patented NI Signal Streaming technology

2. <u>National Instruments 16-Channel AI (8 Voltage and 8 Current) (NI 9207)</u>

- 8 current inputs (±21.5 mA) and 8 voltage (±10 V)
- High-resolution mode with 50/60 Hz rejection
- 500 S/s sample rate (high-speed mode)
- Vsup pins for external power routing (2 A/30 V max)

3. <u>National Instruments 4-Channel, 100 kS/s, 16-bit, ±10 V, Analog Output Module (NI 9263)</u>

- 4 simultaneously updated analog outputs, 100 kS/s
- 16-bit resolution
- Hot-swappable operation

4. <u>National Instruments 4-Channel, 14 S/s, 24-Bit, ±80 mV Thermocouple Input Module (NI 9211)</u>

- 4 thermocouple or ±80 mV analog inputs
- 24-bit resolution; 50/60 Hz noise rejection

- Hot-swappable operation
- NIST-traceable calibration

5. <u>National Instruments 8-Channel 24 V Logic, 100 µs, Sourcing Digital Output Module (NI 9472)</u>

- 8-channel, 100 µs digital output
- 6 to 30 V range, sourcing digital output
- D-Sub or screw-terminal connector options
- Hot-swappable operation
- Extreme industrial certifications/ratings

The voltage ranges and DAQ module numbers are necessary when configuring the DAQ assistant.

d. Hardware Connection:

Table 4.4 illustrates the connection of the sensors and actuators to their proper modules and channels.

2. <u>Software</u>

The system is driven by means of LabVIEW programs designed from the manufacturer or else designed by the students to meet certain requirements.

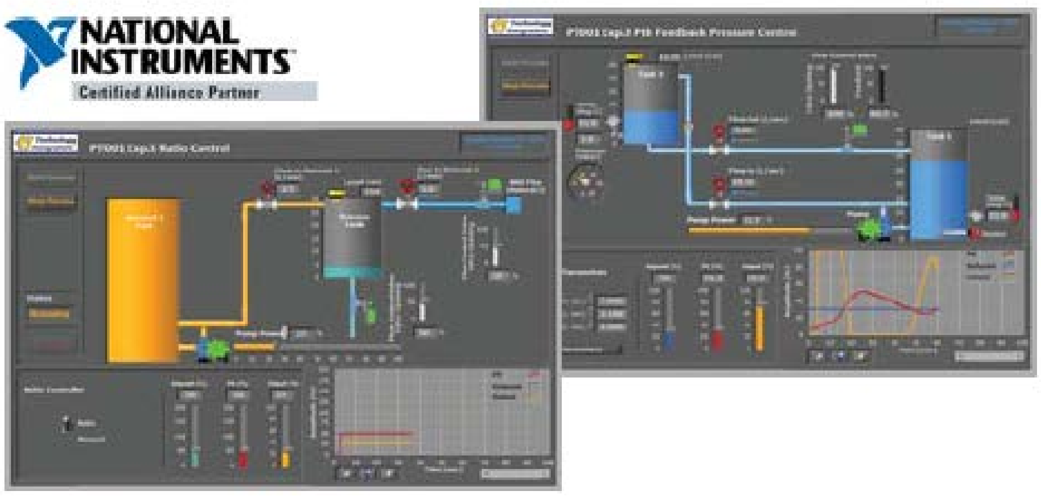

NATIONAL INSTRUMENTS
Certified Alliance Partner
PT001 Exp.3 PID Feedback Pressure Control
PT001 Exp.1 Ratio Control

Figure 4.5: Manufacturer Designed Software

Table 4.4: Connection Table

Slot No.	Module Name	Signal Name	Channel Number	Pin Number	Wire Number	Wire Color	
						Main	Strip
1	9211	Thermocouple Upper Tank	CH 0	0	...	Black	...
				1	...	White	...
		Thermocouple Lower Tank	CH 1	2	...	Black	...
				3	...	White	...
2	9207	Pressure Sensor	CH 8	11	3	Green	...
		Level Meter	CH 9	12	1	Brown	...
		Flow Meter Upper	CH 10	13	5	Gray	...
		Flow Meter Lower	CH 11	14	4	Yellow	...
		Common	...	28	2	White	...
		Flow Control Valve Feed Back	CH 0	1	1	Brown	
				20	2	White	
3	9263	Variable Frequency Drive +	CH 0	0	1	Brown	...
		Variable Frequency Drive -		1	4	Yellow	...
		Flow Control Valve Open +	CH 1	2	2	Green	...
		Flow Control Valve Open -		3	3	White	...
4	9472	Pump Enable	CH 0	0	4	Yellow	...
		Heater Enable	CH 1	1	3	Green	...
		Process Indicator Enable	CH 2	2	5	Gray	...
		24 VDC	...	8	1	Brown	...
		Common	...	9	2	White	...
5	24VDC	Supply Module 9207	...	V_{sup}	35	Brown	...
			...	COM	29	White	...

3. Fluid System Modeling

In fluid systems there are three basic building blocks: Resistance, capacitance and inertance.

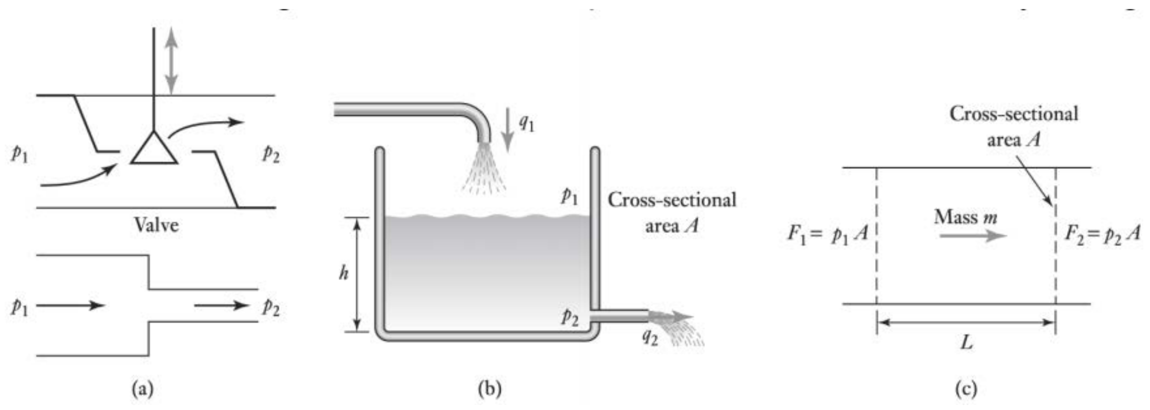


Figure 4.6: Hydraulic examples: (a) resistance, (b) capacitance, (c) inertance

1. Fluid Resistance

Hydraulic resistance is the resistance to flow which occurs as a result of a liquid flowing through valves or changes in a pipe diameter. The relationship between the change in flow rate of liquid *q* through the resistance element and the resulting pressure difference *(p1 – p2)* is given as

$$P_1 - P_2 = R * q \tag{4.1}$$

Where *R* is a constant called the **hydraulic resistance**. The higher the resistance, the higher the pressure difference for a given rate of flow. Equation 4.1 like that for the electrical resistance and Ohm's law, assumes a linear relationship. Such hydraulic linear resistances occur with orderly flow through capillary tubes and porous plugs but nonlinear resistances occur with flow through sharp-edge orifices or if the flow is turbulent.

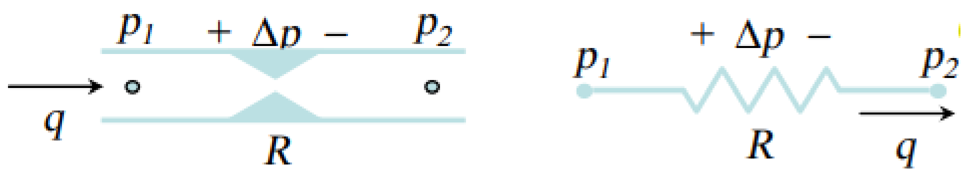


Figure 4.7: Hydraulic Resistance

2. Hydraulic Capacitance

Hydraulic capacitance is the term used to describe energy storage with a liquid where it is stored in the form of potential energy as shown in Figure 4.8. A height of liquid in the container shown in Figure 4.8 (called pressure head) is one form of such storage.

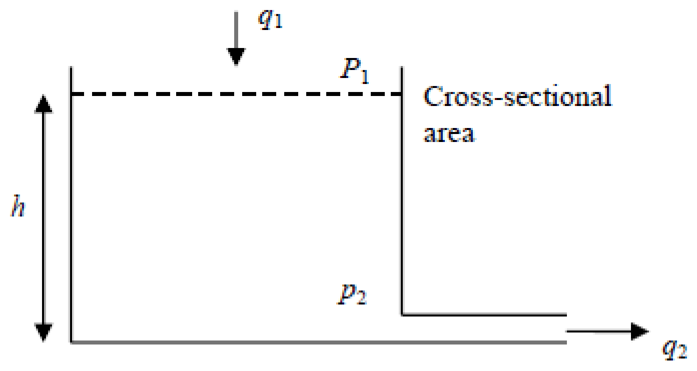


Figure 4.8: Hydraulic Capacitance

For such a capacitance, the rate of change of volume *V* in the tank *(dV/dt)* is equal to the difference between the volumetric rate at which liquid enters the container *q1* and the rate at which it leaves the container *q2*

$$q_1 - q_2 = \frac{dV}{dt} \tag{4.2}$$

But *V* = *Ah*, where *A* is the cross-sectional area of the container and *h* is the height of liquid in it. Hence

$$q_1 - q_2 = \frac{d(Ah)}{dt} = A\frac{dh}{dt} \tag{4.3}$$

The pressure difference between the input and output is p, where $p = h\rho g$ with ρ being the liquid density and *g* the acceleration due to gravity. Therefore, if the liquid is assumed to be incompressible (density does not change with pressure), then Equation (4.3) can be written as

$$q_1 - q_2 = \frac{d(p/\rho g)}{dt} = \frac{A}{\rho g}\frac{dp}{dt} \tag{4.4}$$

The hydraulic capacitance *C* is defined as

$$C = \frac{A}{\rho g} \tag{4.5}$$

Therefore

$$q_1 - q_2 = C\frac{dp}{dt} \tag{4.6}$$

3. Hydraulic Inertance

Hydraulic inertance is the equivalent of inductance in electrical systems or a spring in mechanical systems. To accelerate a fluid and to increase its velocity a force is required. Consider a block of liquid of mass m as shown in Figure 4. The net force acting on the liquid is

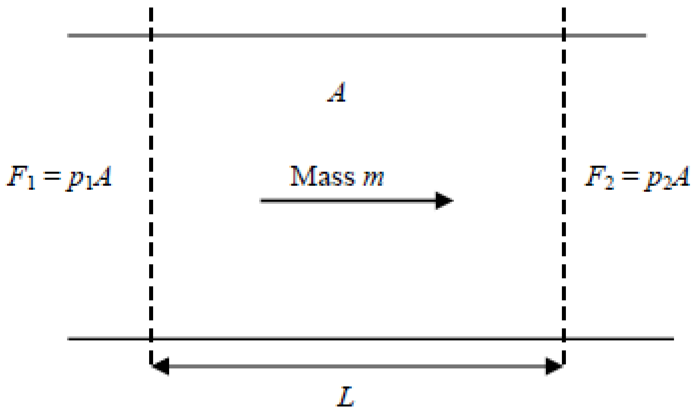


Figure 4.9: Hydraulic Inertance

$$F_1 - F_2 = p_1A - p_2A = (p_1 - p_2)A \tag{4.7}$$

Where *(p1 – p2)* is the pressure difference and A the cross-sectional area. This net force causes the mass to accelerate with an acceleration a, and so

$$(p_1 - p_2)A = ma \tag{4.8}$$

However, *a* is the rate of change of velocity *dv/dt*, therefore

$$(p_1 - p_2)A = m\frac{dv}{dt} \tag{4.9}$$

The mass of liquid concerned has a volume of *AL*, where *L* is the length of the block or the distance between the points in the liquid where the pressures *p1* and *p2* are measured. If the liquid has a density ρ then $m = AL\rho$, and

$$(p_1 - p_2)A = AL\rho\frac{dv}{dt} \tag{4.10}$$

But the volume rate of flow $q = Av$, hence

$$(p_1 - p_2)A = L\rho\frac{dq}{dt} \tag{4.11}$$

$$(p_1 - p_2) = I\frac{dq}{dt} \tag{4.12}$$

Where the hydraulic inertance *I* is defined as

$$I = \frac{L\rho}{A} \tag{4.13}$$

The following system represents the system laboratory:

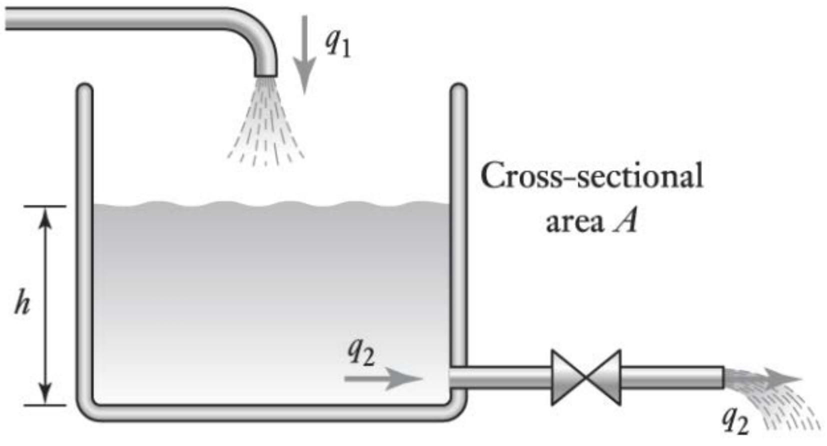


Figure 4.10: Process Trainer System

Procedure:

1. Inspect Tables 4.3 and 4.4.
2. Design a LabVIEW program to meet the task provided by your lab supervisor. You may need the following formulas:
- For the ultrasonic level meter : H= 1543.75 *I - 6.175
- For any analog signal, select "one sample on demand".
- The flow control valve at 0V is fully opened.
3. Fill the following Table to help you configure your DAQ assistant.

Table 4.5: Hardware Configuration

	Input/output	Digital/Analog	Signal type (voltage/current)	Range of Signal	Module name	Channel number
Actuator1						
Sensor1						
Actuator2						
Sensor2						

Discussion and Analysis:

1. How many sensors are there in the system you designed? List them?
2. How many actuator are there in the system you designed? List them?
3. Draw a flow chart of the program you designed.
4. How many DAQ assistant you used in your program? What are their types?
5. How many samples per second do the analog modules support?

References:

1. Mechatronics: electronic control systems in mechanical and electrical engineering, 4th edition, W.Bolton.

Experiment 5
PID Controller: Concept and Tuning

Objectives:

- Familiarize the student with the concept of PID controller.
- Study the effect of each action of the PID controller.
- Learn how tune the PID controller gains.

Apparatus:

The system used during this experiment is the PT001 System.

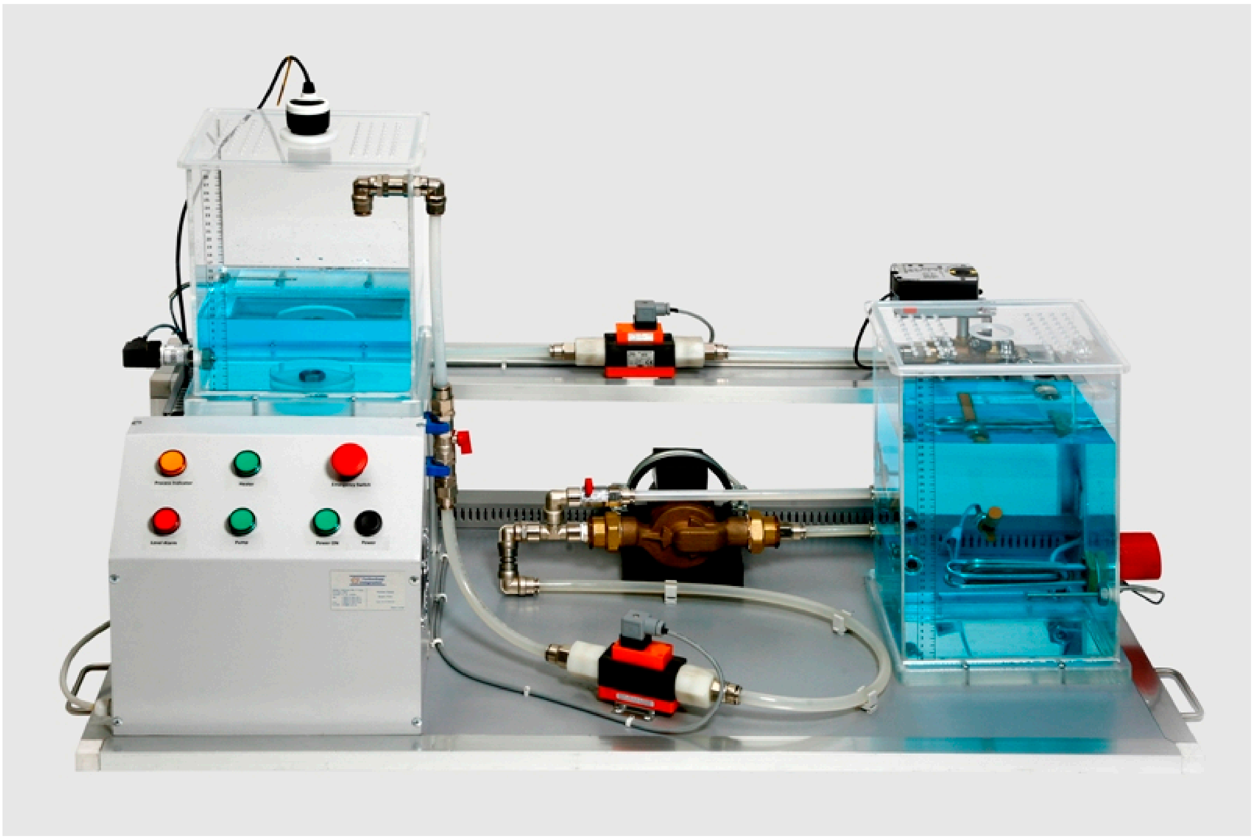


Figure 5.1: PT001 Trainer

Theoretical Background:

Introduction:

Proportional-Integral-Derivative (PID) control is the most common control algorithm used in the industry and has been universally accepted in industrial control. The popularity of PID controllers can be attributed partly to their robust performance in a wide range of operating conditions and partly to their functional simplicity, which allows engineers to operate them easily. As the name suggests, PID algorithm consists of three basic coefficients; proportional, integral and derivative, which are varied to get optimal response.

$$PID = K_p * e(t) + K_i * \int e(t)dt + K_d * \frac{de(t)}{dt} \tag{5.1}$$

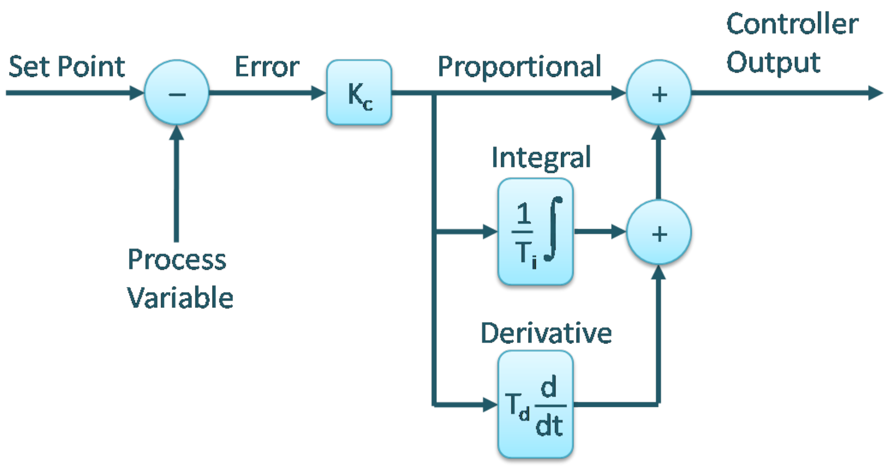


Figure 5.2: PID Controller Algorithm

- Proportional Response

The proportional component depends only on the difference between the setpoint and the process variable. This difference is referred to as the Error term. The proportional gain (Kc) determines the ratio of output response to the error signal. For instance, if the error term has a magnitude of 10, a proportional gain of 5 would produce a proportional response of 50. In general, increasing the proportional gain will increase the **speed** of the control system response. However, if the proportional gain is too large, the process variable will begin to **oscillate**. If Kc is increased further, the oscillations will become larger and the system will become unstable and may even oscillate out of control.

$$P_{out} = K_p * e(t) \tag{5.2}$$

- Integral Response

The integral component summates the error term over time. The result is that even a small error term will cause the integral component to increase slowly. The integral response will continually increase over time unless the error is zero, so the effect is to drive the Steady-State **error to zero**. Steady-State error is the final difference between the process variable and setpoint. A phenomenon called integral windup results when integral action saturates a controller without the controller driving the error signal toward zero.

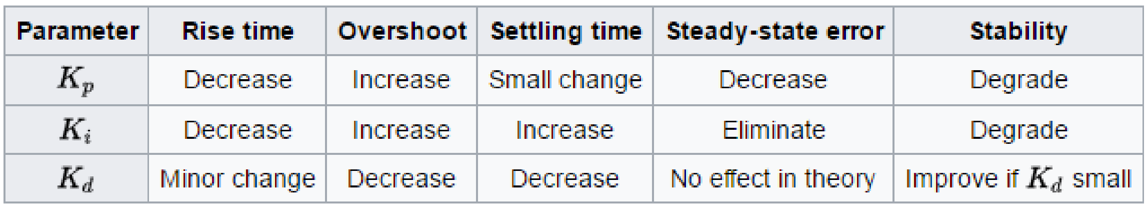
$$I_{out} = K_i * \int e(t)dt \tag{5.3}$$

- Derivative Response

The derivative component causes the output to decrease if the process variable is increasing rapidly. The derivative response is proportional to the rate of change of the process variable. Increasing the derivative time (Td) parameter will cause the control system to react more strongly to changes in the error term and will increase the speed of the overall control system response. Most practical control systems use **very small derivative time** (Td), because the Derivative Response is highly sensitive to noise in the process variable signal. If the sensor feedback signal is noisy or if the control loop rate is too slow, the derivative response can make the control **system unstable**.

$$D_{out} = T_d * \frac{de(t)}{dt} \qquad (5.4)$$

Table 5.1: PID controller's Elemet Actions



Parameter	Rise time	Overshoot	Settling time	Steady-state error	Stability
K_p	Decrease	Increase	Small change	Decrease	Degrade
K_i	Decrease	Increase	Increase	Eliminate	Degrade
K_d	Minor change	Decrease	Decrease	No effect in theory	Improve if K_d small

PID Tuning Using Ziegler and Nichols

Ziegler and Nichols described simple mathematical procedures, the first and second methods respectively, for tuning PID controllers. These procedures are now accepted as standard in control systems practice. Both techniques make a priori assumptions on the system model, but do not require that these models be specifically known. Ziegler-Nichols formula for specifying the controllers are based on plant step responses.

- The First Method

The first method is applied to plants with step responses of the form displayed in Figure 5.3 (response exhibit s-shaped curve for unit step input). This type of response is typical of a first order system with transportation delay, such as that induced by fluid flow from a tank along a pipe line. It is also typical of a plant made up of a series of first order systems.

The response is characterized by two parameters, L the delay time and T the time constant. These are found by drawing a tangent to the step response at its point of inflection and noting its intersections with the time axis and the steady state value. The plant model is therefore G(s) $= \frac{ke^{-sl}}{Ts+1}$, Ziegler and Nichols derived the following control parameters based on this model which is describes in Table 5.2:

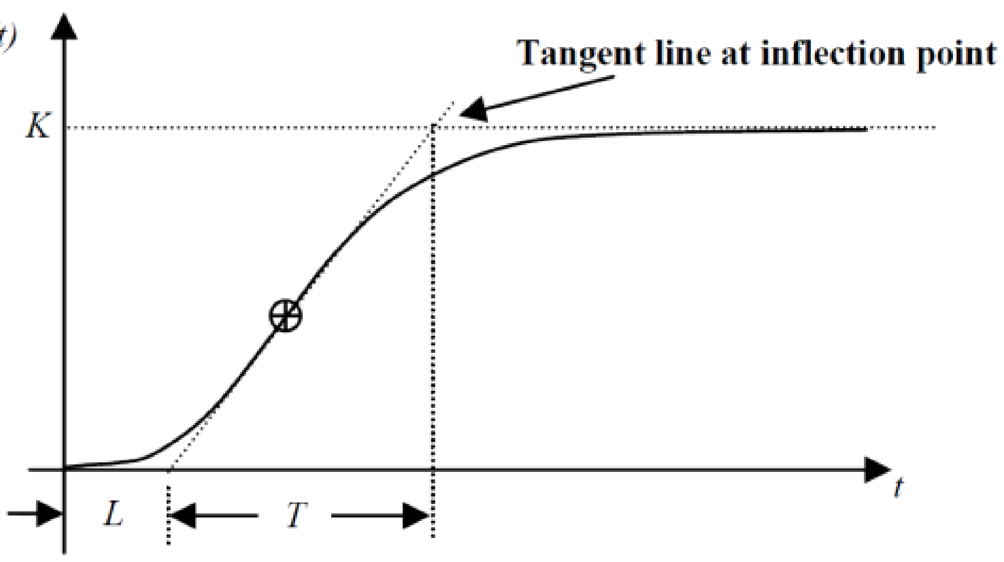


Figure 5.3: First Order S-Shape Response

Table 5.2: PID Controller Tuning Using Ziegler-Nichols First Method

Type of Controller	K_p	T_i	T_d
P	T/L	∞	0
PI	0.9 T/L	L / 0.3	0
PID	1.2 T/L	2L	0.5L

Notes:

- Using this method, the tuning is done for closed loop system.
- Complete PID is considered unnecessary because it is a first order system that sustains no oscillations; thus no need for the derivative action.

- <u>The Second Method</u>

The second method targets plants that can be rendered unstable under proportional control. The technique is designed to result in a **closed loop** system with 25% overshoot. This is rarely achieved as Ziegler and Nichols determined the adjustments based on a specific plant model. The steps for tuning a PID controller via the 2nd method are as follows:

Using only proportional feedback control:

1. Reduce the integrator and derivative gains to 0.

2. Increase Kp from 0 to some critical value Kp=Kcr at which sustained oscillations occur. If it does not occur then another method has to be applied.

3. Note the value Kcr and the corresponding period of sustained oscillation, Pcr.

The controller gains are now specified as follows in Table 5.3.

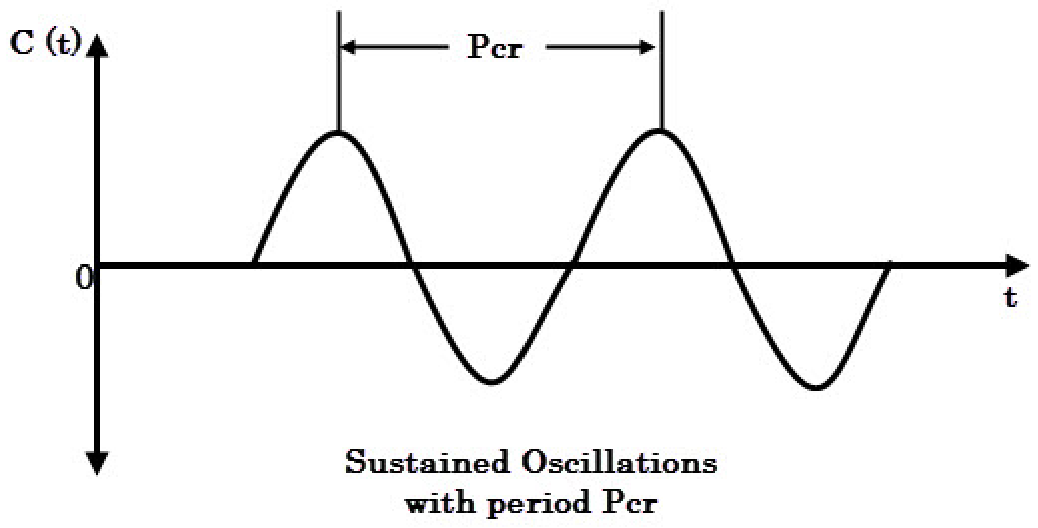


Figure 5.4: Second Order Response

Table 5.3: PID Controller Tuning Using Ziegler-Nichols Second Method

Type of Controller	K_p	T_i	T_d
P	$0.5\ K_{cr}$	∞	0
PI	$0.45\ K_{cr}$	$1/1.2\ \ P_{cr}$	0
PID	$0.6\ K_{cr}$	$0.5\ P_{cr}$	$0.125\ P_{cr}$

Note: using this method, the tuning is done for closed loop system.

Process Trainer System

For the Process Trainer System in the labratory, the PID Feedback Level Experiment, the PID level controller takes the feedback signal from the level meter and compares it with the desired value of level (Setpoint) and accordingly gives a control signal to the pump to deliver the required water flow & maintain the level at the desired value.

The same principle is applied to the pressure and flow controllers, except that the feedback signal is taken from the pressure sensor and the flow meter respectively. This type of controllers rejects the disturbances after they affect the PV (Water Level)

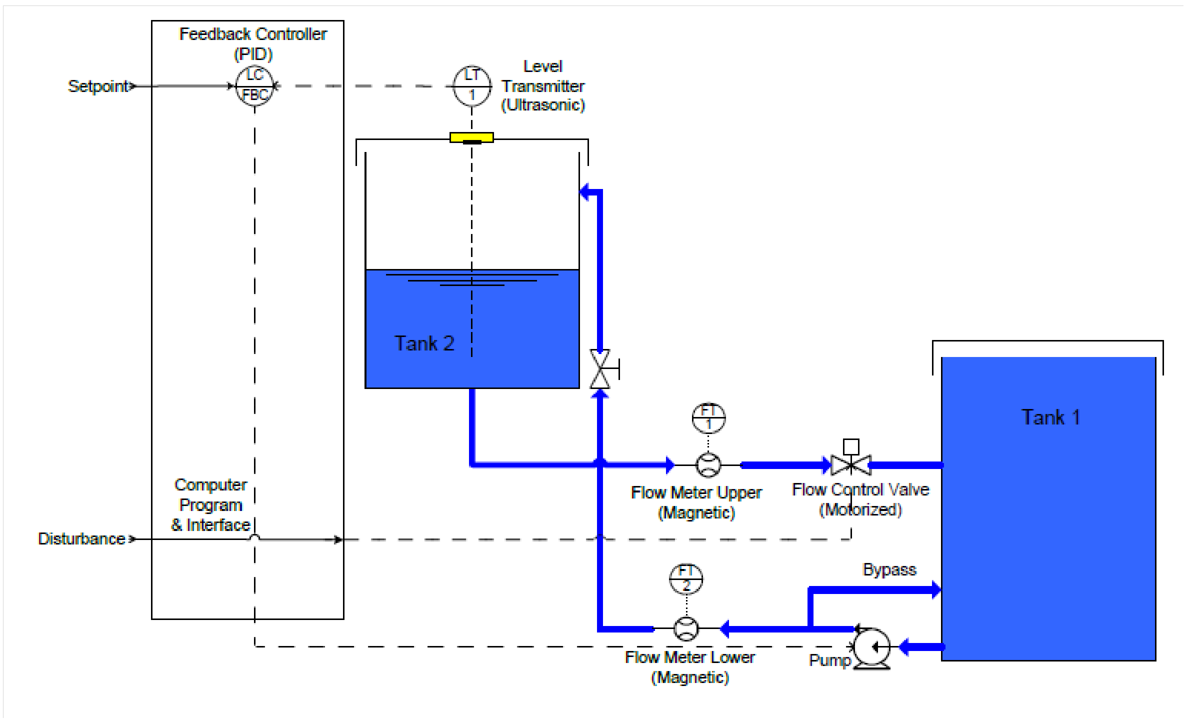


Figure 5.5: PID Level Control Implementation in PT001

Procedure:

1. Click on Experiment 3: **“PID Feedback Control”**. Another menu will appear; choose Experiment 3.1: **“PID Feedback Level Control”**.

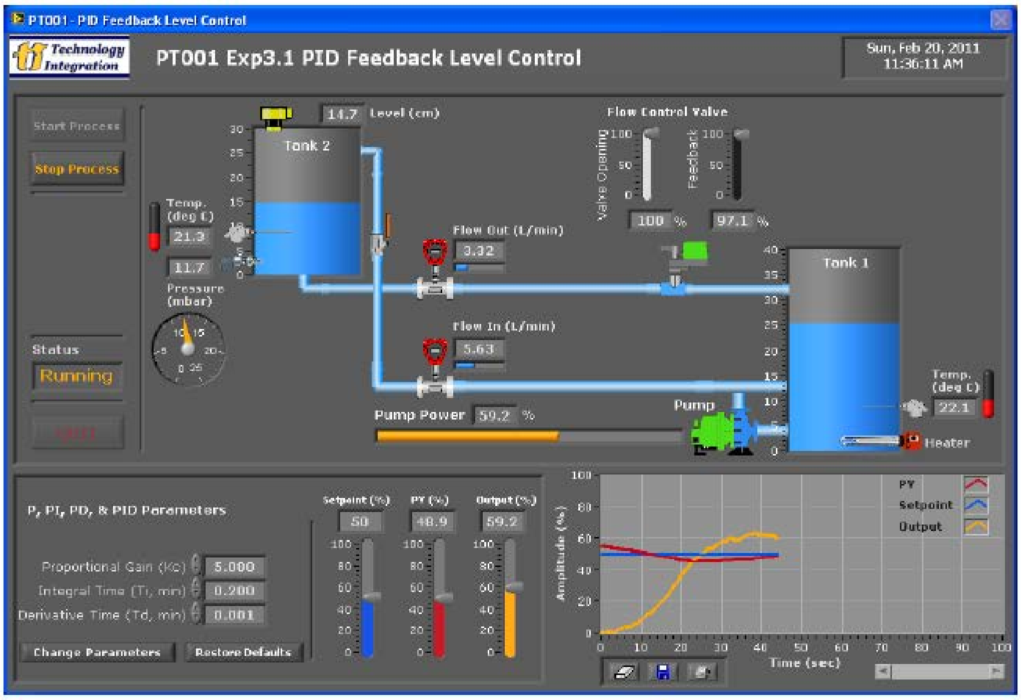
2. Drag the Setpoint slider to 30% of its value. Press the [Start Process] button to start the control process. Observe the behavior of the (PV) and the controller’s output on the chart.

3. After the controller and (PV) settle down and stabilize, apply a disturbance to the system by changing the Flow Control Valve’s opening. Observe the disturbance rejection and the reaction of the controller.

4. Set the Flow Control Valve to “Fully Open”, and change the setpoint to 60%. When the level settles at 60% of its full scale, observe the behavior of the pump.

5. Change the setpoint to other values between (20% & 75%). Observe the controller’s behavior at these new setpoint values.

6. Stop the process and press the [Quit] button to go back to the experiments menu.



PT001 - PID Feedback Level Control

PT001 Exp3.1 PID Feedback Level Control

Sun, Feb 20, 2011 11:36:11 AM

Level (cm) 14.7

Temp. (deg C) 21.3

Pressure (mbar) 11.7

Flow Out (L/min) 3.32

Flow In (L/min) 5.63

Valve Opening 100 %

Feedback 97.1 %

Pump Power 59.2 %

Temp. (deg C) 22.1

Status Running

P, PI, PD, & PID Parameters

Proportional Gain (Kc) 5.000

Integral Time (Ti, min) 0.200

Derivative Time (Td, min) 0.001

Setpoint (%) 50

PV (%) 48.9

Output (%) 59.2

Figure 5.6: PID Feedback Level Control Experiment

Discussion and Analysis:

1. For the system you operated, check the PID controller’s parameters. What type of a controller is it?

P Controller	PD Controller
PI Controller	PID Controller

2. Sketch the response you obtained during step 2 of the experiment.
3. What is the effect of applying the disturbance?
4. When you changed the opening to 60% in step 4, does the pump operate at a higher or lower rate in comparison to its operation at the 30% setpoint? Why?

References:

1. http://www.ni.com/white-paper/3782/en/
2. The Design of PID Controllers using Ziegler Nichols Tuning, Brian R Copeland , 2008.

Experiment 6
SCARA and Gryphon Robots Forward and Inverse Kinematics

Objectives:

- To introduce the students to the forward and inverse kinematics of robots.
- To familiarize the student with a four degree of freedom SCARA robot.
- To familiarize the student with a five degree of freedom Gryphon robot.

Apparatus:

The system used during this experiment is the PT001 System.

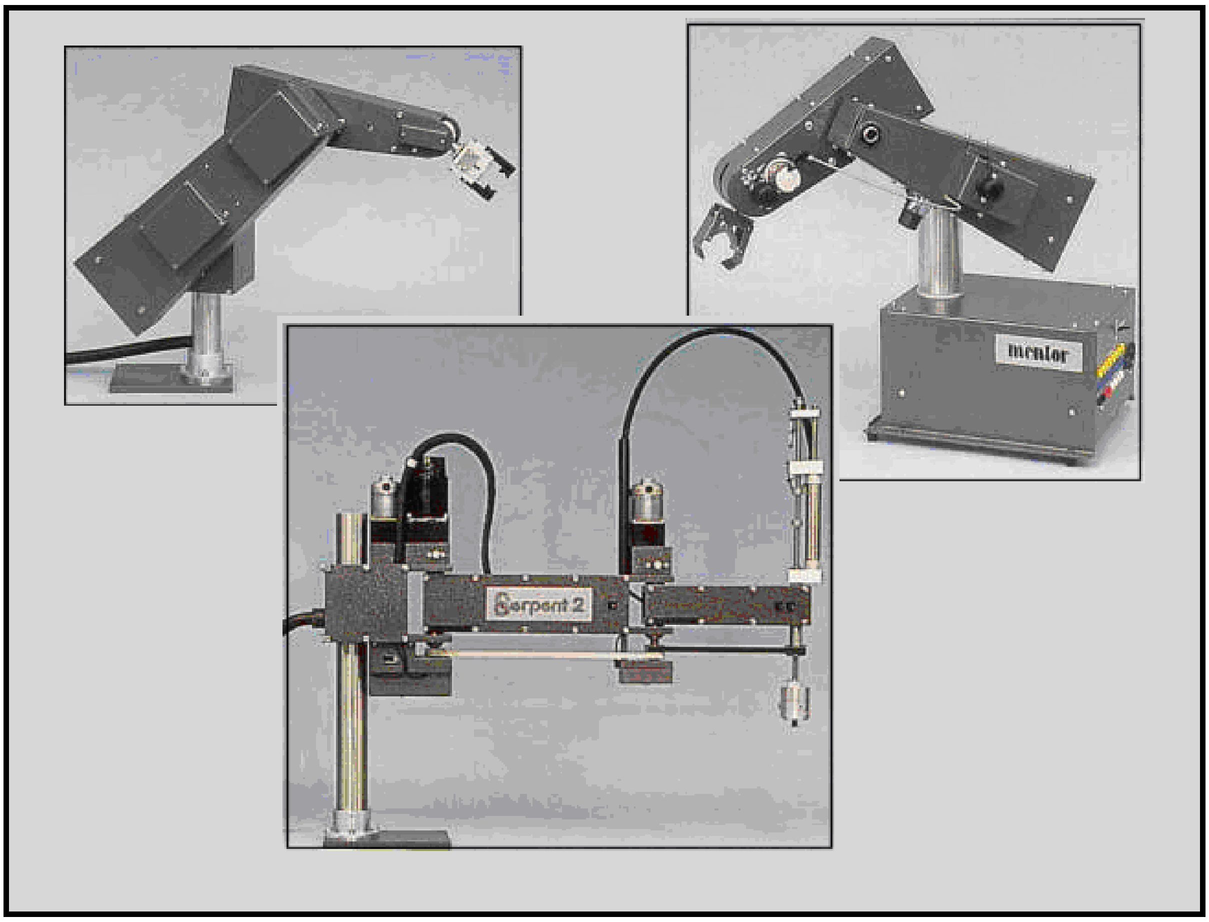


Figure 6.1: Robots

Theoretical Background:

SCARA robots are one of the most popular in industry and the Serpent ECs are typical of this class of machine. The movement of a SCARA is simple but entirely adequate for a vast number of assembly and pick-and-place applications.

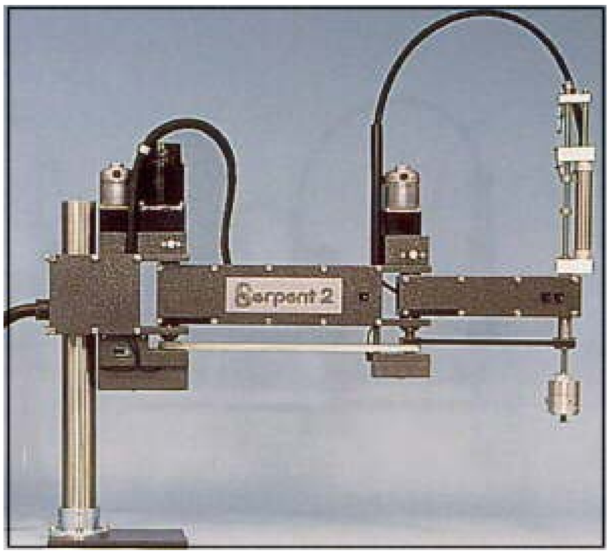


Figure 6.2: SCARA Robot

SCARA is an acronym for Selectively Compliant Articulated Robot Arm which means there is a small amount of springiness in the plane of operation. Providing there is a small lead-in, on the component, the compliance allows placement of a part even where there is some misalignment.

The two joints of the Serpent EC arms and the wrist are driven by DC servo motors with encoder feedback to achieve accurate closed-loop control. As is usual for SCARAs, the wrist motor is situated back at the column and connected by belts to the end of the arm. This arrangement maintains a constant wrist angle relative to the bench when the arm moves.

The vertical movement is by pneumatic cylinder, operating at adjustable speed between moveable end stops.

The Serpents may be programmed from the computer either by setting the data for each axis or by steering the arm by hand using the lead-by-the-nose buttons. Alternatively the Serpent EC will follow the hand-held control pendant or simulator.

In inverse kinematics problems you will be given the position and orientation of the TCP, and then you should calculate the joints variables of the manipulator in order to solve the problem.

It is very important to know how to derive the inverse kinematics equations of different Robots, because in many Robots there is no way to program them using Cartesian coordinates (X, Y, Z), so you will need to derive the Inverse Kinematics Equations to match your (X, Y, Z) coordinates with Robot’s joints variables (θ, D).

Gryphon Robot is a human arm based configuration as widely used in industry.

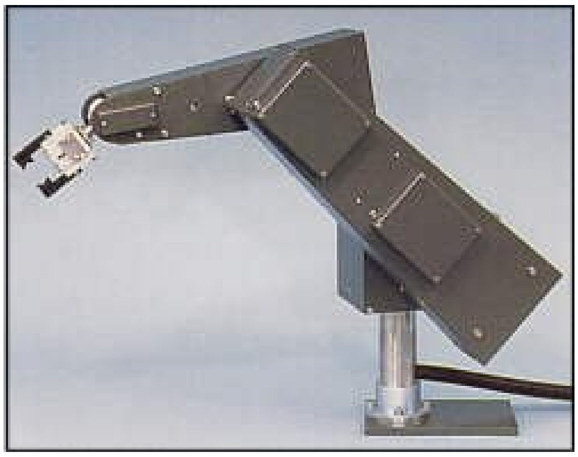


Figure 6.3: Gryphon Robot

Precise, smooth, fast travel characterizes this robot. The articulated arm is under the control of four micro-processors and will accurately place components in a CNC machine or work-cell.

Each axis is powered by a stepper motor with encoder feedback to provide closed-loop control.

In the controller there is one micro-processor to monitor the positions of the axes, two more to control the motors and another one to supervise the first three and to communicate with the host computer.

Either a two fingered or a vacuum gripper may be fitted and these may be readily interchanged.

The Joints are driven using stepper motors and their feedback is measured using incremental optical encoders. The wrist movements are obtained through a differential gear mechanism.

In inverse kinematics problems you will be given the position and orientation of the TCP, and then you should calculate the joints variables of the manipulator in order to solve the problem.

It is very important to know how to derive the inverse kinematics equations of different Robots, because in many Robots there is no way to program them using Cartesian coordinates (X, Y, Z), so you will need to derive the Inverse Kinematics Equations to match your (X, Y, Z) coordinates with Robot's joints variables (θ, D).

Procedure:

1. Inspect the SCARA Robot. Measure its dimensions.
2. Inspect the Gryphon Robot. Measure its dimensions.
3. Actuate the robots and analyze their workspaces.

Discussion and Analysis:

1. Write the forward kinematics equations describing the SCARA robot.
2. Derive the inverse kinematic equations describing the SCARA robot.
3. Sketch a simple work space for the SCARA robot.
4. Write the forward kinematics equations describing the Gryphon robot.
5. Derive the inverse kinematic equations describing the Gryphon robot.
6. Sketch a simple work space for the Gryphon robot.