UNIT IV
DATA ACQUISITION SYSTEM


INTRODUCTION

- Computer-based measurement and automation is popular because our PC provides the platform we need to make our measurement and automation systems dependable and efficient.
- Its extensive processing capabilities empower us to create flexible solutions based on industry standards.
- With this flexibility, we can adjust our application specifications more easily than with traditional tools.
- National Instruments software, including LabVIEW and Measurement Studio, delivers PC-based data analysis, connectivity, and presentation power to new levels in measurement and automation applications.
- National Instruments hardware and software connect the computer to your application.
- By providing an extensive hardware selection, including data acquisition and signal conditioning devices, instrument control interfaces (such as GPIB, Serial, VXI and PXI), image acquisition, motion control and industrial communications interfaces, National Instruments offers the widest range of solutions for practically any measurement as shown in Figure 4.1.
- This chapter describes instrument control of stand-alone instruments using a GPIB or serial interface.
- Use LabVIEW to control and acquire data from instruments with the Instrument I/O Assistant, the VISA API, and instrument drivers.
- If you choose industry-standard control technologies, you are not limited to the type of instrument you can control.
- You can mix and match instruments from various categories as shown in Figure 4.2.
- The most common categories of instrument interfaces are GPIB, serial, modular instruments and PXI modular instruments.
- Additional types of instruments include image acquisition, motion control, USB, Ethernet, parallel port, NI-CAN and other devices.
Figure 4.1 Computer-based measurement and automation.

Figure 4.2 PC control of instruments

- When you use PCs to control instruments, you need to understand properties of the instrument, such as the communication protocols to use.
- You must consider the following issues with PC control of instrumentation:
  - Type of connector (pinouts) on the instrument
  - Cables needed—null-modem, number of pins, male/female
Electrical properties involved—signal levels, grounding, cable length restrictions
Communication protocols used—ASCII commands, binary commands, data format
Software drivers available

**GPIB:**
- The ANSI/IEEE Standard 488.1-1987, also known as General Purpose Interface Bus (GPIB), describes a standard interface for communication between instruments and controllers from various vendors.
- GPIB, instruments offer test and manufacturing engineers the widest selection of vendors and instruments for general-purpose to specialized vertical market test applications as shown in Figure 4.3. GPIB instruments are often used as standalone benchtop instruments where measurements are taken by hand.
- You can automate these measurements by using a PC to control the GPIB instruments.

![GPIB Interface](image)

**Figure 4.3** GPIB communication.
- IEEE 488.1 contains information about electrical, mechanical and functional specifications.
- The ANSI/IEEE Standard 488.2-1992 extends IEEE 488.1 by defining a bus communication protocol, a common set of data codes and formats, and a generic set of common device commands.
- GPIB is a digital, 8-bit parallel communication interface with data transfer rates of 1 Mbyte/s and higher, using a three-wire handshake.
- The bus supports one system controller, usually a computer, and up to 14 additional instruments.
- The GPIB protocol categorizes devices as controllers, talkers, or listeners to determine which device has active control of the bus.
- Each device has a unique GPIB primary address between 0 and 30.
The controller defines the communication links, responds to devices that request service, sends GPIB commands, and passes/receives control of the bus.

Controllers instruct talkers to talk and to place data on the GPIB. You can address only one device at a time to talk.

The controller addresses the listener to listen and to read data from the GPIB. You can address several devices to listen.

**Data Transfer Termination**

Termination informs listeners that all data has been transferred. You can terminate a GPIB data transfer in the following three ways:

- The GPIB includes an end-or-Identify (EOI) hardware line that can be asserted with the last data byte. This is the preferred method.
- Place a specific end-of-string (EOS) character at the end of the data string itself. Some instruments use this method instead of or in addition to the EOI line assertion.
- The listener counts the bytes transferred by handshaking and stops reading when the listener reaches a byte count limit. This method is often used as a default termination method because the transfer stops on the logical OR of EOI, EOS (if used) in conjunction with the byte count. Thus, you typically set the byte count to equal or exceed the expected number of bytes to be read.

**Data Transfer Rate**

To achieve the high data transfer rate that the GPIB was designed for, you must limit the number of devices on the bus and the physical distance between devices.

You can obtain faster data rates with HS488 devices and controllers. HS488 is an extension to GPIB that most NI controllers support.

**HARDWARE SPECIFICATIONS**

The GPIB is a digital, 24-conductor parallel bus. As shown in Figure 4.4, it consists of eight data lines (DIO 1-8), five bus management lines (EOI, IFC, SRQ, ATN, REN), three handshake lines (DAV, NRFD, NDAC), and eight ground lines.

The GPIB uses an eight-bit parallel, byte-serial, asynchronous data transfer scheme.

This means that whole bytes are sequentially handshake across the bus at a speed that the slowest participant in the transfer determines.

Because the unit of data on the GPIB is a byte (eight bits), the messages transferred are frequently encoded as ASCII character strings.
Additional electrical specifications allow data to be transferred across the GPIB at the maximum rate of 1 MB/sec because the GPIB is a transmission line system.

These specifications are:
- A maximum separation of 4 m between any two devices and an average separation of 2 m over the entire bus.
- A maximum cable length of 20 m.
- A maximum of 15 devices connected to each bus with at least two-thirds of the devices powered on.

If you exceed any of these limits, you can use additional hardware to extend the bus cable lengths or expand the number of devices allowed.

SOFTWARE ARCHITECTURE

The software architecture for the instrument control using LabVIEW is similar to the architecture for DAQ.

Instrument interfaces such as GPIB include a set of drivers.

Use MAX to configure the interface. VISA, Virtual Instrument Software Architecture, is a common API to communicate with the interface drivers and is the preferred method used when programming for instrument control in LabVIEW, because VISA abstracts the type of interface used.

Many LabVIEW VIs used for instrument control use the VISA API.

For example, the Instrument I/O Assistant is a LabVIEW Express VI that can use VISA to communicate with message-based instruments and convert the response from raw data to an ASCII representation.

Use the Instrument I/O Assistant when an instrument driver is not available.
In LabVIEW, an instrument driver is a set of VIs specially written to communicate with an instrument.

Underneath and between the different hardware components are the software components that are the heart of your instrumentation system.

Figure 4.5 shows the software architecture for Windows 2000/XP from the perspective of the computer.

The instrument is connected to the computer through a built-in connector such as RS-232 or an installed interface board such as GPIB or VXI/MXI.

Driver-level software for instrument is in the form of a Dynamic Link Library (DLL).

In the example shown in the figure, LabVIEW is the high-level application software layer.

LabVIEW includes built-in functions for GPIB, serial, VXI, and computer-based instruments that load and use the installed driver software.

As with all Windows device drivers, any DLLs you install for interface boards will also interact with the Windows Registry so that resources such as base addresses, interrupt levels, and DMA channels can be assigned.

Configuration and diagnostic software tools are also available with National Instruments hardware.

MAX (Windows; GPIB)

Use MAX to configure and test the GPIB interface.

MAX interacts with the various diagnostic and configuration tools installed with the driver and also with the Windows Registry and Device Manager.

The driver-level software is in the form of a DLL and contains all the functions that directly communicate with the GPIB interface.

The Instrument I/O VIs and functions directly call the driver software.

Open MAX by double-clicking the icon on the desktop or by selecting Tools>Measurement & Automation Explorer in LabVIEW to open the window as in Figure 4.6.
Configure the objects listed in MAX by right-clicking each item and selecting an option from the shortcut menu.

The Measurement and Automation Explorer (MAX) allows you to configure the GPIB card and search for instruments, set the VISA alias name, run NI-Spy and configure IVI instrument drivers.

If a GPIB+ card is installed, you can run the GPIB analyzer software from MAX.

INSTRUMENT I/O ASSISTANT

The Instrument I/O Assistant is a LabVIEW Express VI which you can use to communicate with message-based instruments and convert the response from raw data to an ASCII representation.

You can communicate with an instrument that uses a serial, Ethernet, or GPIB interface.

The Instrument I/O Assistant organizes instrument communication into ordered steps.

To use Instrument I/O Assistant, you place steps into a sequence.

As you add steps to the sequence, they appear in the Step Sequence window.

Use the view associated with a step to configure instrument I/O.

The Instrument I/O Assistant shown in Figure 4.7 is located on the Functions » Input and Functions » All Functions » Instrument I/O palettes is a LabVIEW Express VI that allows you to easily test communication with your instrument and develop a sequence of query, parse and write steps.
These steps can be saved as an Express VI for instant use or can be converted to a LabVIEW subVI. Use the Instrument I/O Assistant when an instrument driver is not available.

**Figure 4.7** Instrument I/O Assistant.

- To launch the Instrument I/O Assistant, place the Instrument I/O Assistant Express VI on the block diagram in LabVIEW.
- The Instrument I/O Assistant Express VI is available in the Instrument I/O category of the **Functions** palette.
- The **Instrument I/O Assistant** configuration dialog box appears.
- If it does not appear, double-click the Instrument I/O Assistant icon.
- Complete the following steps to configure the Instrument I/O Assistant.
  - **Step 1:** Select an instrument. Instruments that have been configured in MAX appear in the **Select an instrument** pull-down menu.
  - **Step 2:** Choose a **Code generation type.** VISA code generation allows for more flexibility and modularity than GPIB code generation.
  - **Step 3:** Select from the following communication steps using the **Add Step** button:
    - **Query and Parse**—Sends a query to the instrument, such as “*IDN?” and parses the returned string. This step combines the Write command and Read and Parse command.
    - **Write**—Sends a command to the instrument.
    - **Read and Parse**—Reads and parses data from the instrument.
  - **Step 4:** After adding the desired number of steps, click the **Run** button to test the sequence of communication that you have configured for the Express VI.
  - **Step 5:** Click the **OK** button to exit the **Instrument I/O Assistant** configuration dialog box.
- LabVIEW adds input and output terminals to the Instrument I/O Assistant Express VI on the block diagram that correspond to the data you receive from the instrument.
- To view the code generated by the Instrument I/O Assistant, right-click the Instrument I/O Assistant icon and select **Open Front Panel** from the shortcut menu.
- This converts the Express VI to a subVI. Switch to the block diagram to see the code generated.
- After you convert an Express VI to a subVI, you cannot reconvert the Express VI.
Once you have placed the I/O Assistant on the block diagram, the wizard opens.
- The wizard starts in the Select Instrument step, where you can choose a GPIB or serial instrument.
- You also can check the interface properties from this window.
- After selecting the instrument, you can add sequences to Query and Parse, Write, or Read and Parse.
- In addition, after you have set up a communication sequence with one instrument, you can set up additional instruments in the same Express VI.
- In Figure 4.8, a GPIB instrument was set up, then a query (*IDN?) was sent to the instrument.
- The response was automatically parsed, resulting in a string output.

![Figure 4.8 Serial Configuration of the Instrument I/O Assistant.](image)

**VISA:**
- Virtual Instrument Software Architecture (VISA) is the lower layer of functions in the LabVIEW instrument driver VIs that communicates with the driver software.
- VISA by itself does not provide instrumentation programming capability.
- VISA is a high-level API that calls low-level drivers.
- As shown in Figure 4.9 VISA can control VXI, GPIB, serial, or computer-based instruments and makes the appropriate driver calls depending on the type of instrument used.
- When debugging VISA problems, remember that an apparent VISA problem could be an installation problem with one of the drivers that VISA calls.
In LabVIEW, VISA is a single library of functions you use to communicate with GPIB, serial, VXI and computer-based instruments.

You do not need to use separate I/O palettes to program an instrument.

For example, some instruments give you a choice for the type of interface.

If the LabVIEW instrument driver was written with functions on the Functions»All Functions»Instrument I/O»GPIB palette, those instrument driver VIs would not work for the instrument with the serial port interface.

VISA solves this problem by providing a single set of functions that work for any type of interface.

Therefore, many LabVIEW instrument drivers use VISA as the I/O language.

For many years, industry has moved toward purchasing instrumentation from a variety of vendors.

This allows engineers to select the best possible equipment for their applications without being locked into a specific vendor.

This trend required the definition of hardware standards to ensure the compatibility between different modules.

This was one of the factors leading to the development of the VXI specification.

But even with these improved hardware standards, a system was time consuming and expensive to put together.

Successful integration of a multivendor system requires all hardware and software products work together, eliminating system-level compatibility issues for end-users.

National Instruments initially addressed the software problems with instrument drivers, which helped reduce both integration time and software development costs.

In 1993, National Instruments joined with GenRad, Racal Instruments, Tektronix and Wavetek to form the VXI plug & play Systems Alliance.

The goals of the alliance are to ensure multivendor interoperability for VXI systems and to reduce the development time for an operational system.

**VISA Programming Terminology**

VISA or Virtual Instrument Software Architecture is a protocol built upon 488.2 driver and functions to meet the industry needs for having a way to easily interface with multiple I/Os and have all manufacturers of instruments and instrument drivers follow a protocol.

VISA created by the VXIplug&play Alliance which is composed of the top 35 instrument manufacturers such as HP.

National Instruments is a leading member of the alliance.
The resource name contains information on the type of I/O interface and the device address.

You can use an alias you assign in MAX instead of the instrument descriptor. (Mac OS) Edit the visaconf.ini file to assign a VISA alias.

(UNIX) Use the visaconf utility. If you choose not to use the Instrument I/O Assistant to automatically generate code for you, you can still write a VI to communicate with the instrument.

Before being introduced to VISA programming, you should become familiar with some of the VISA terminology.

The most important objects in the VISA language are known as resources.

The functions you can use with an object are known as operations.

In addition to the operations that you can use an object, the object has variables, known as attributes, associated with it that contains information related to the object.

Three terms need definitions and the following terminology is similar to that used for instrument driver VIs.

- **Resource**—Any instrument in the system, including serial and parallel ports.
- **Session**—You must open a VISA session to a resource to communicate with it, similar to a communication channel. When you open a session to a resource, LabVIEW returns a VISA session number which is a unique refnum to that instrument. You must use the session number in all subsequent VISA functions.
- **Instrument Descriptor**—Exact name of a resource. The descriptor specifies the interface type (GPIB, VXI, ASRL), the address of the device (logical address or primary address) and the VISA session type (INSTR or Event). The instrument descriptor is similar to a telephone number, the resource is similar to the person with whom you want to speak and the session is similar to the telephone line. Each call uses its own line, and crossing these lines results in an error. Table 4.1 shows the proper syntax for the instrument descriptor.

**TABLE 4.1 Syntax for various instrument interfaces**

<table>
<thead>
<tr>
<th>Interface</th>
<th>Resource Name Grammar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial</td>
<td>ASRL[board][::INSTR]</td>
</tr>
<tr>
<td>GPIB</td>
<td>GPIB[board][::primary address][::INSTR]</td>
</tr>
<tr>
<td>VXI</td>
<td>VXI[board][::VXI logical address][::INSTR]</td>
</tr>
<tr>
<td>GPIB-VXI</td>
<td>GPIB-VXI[board][::GPIB-VXI primary address][::VXI logical address][::INSTR]</td>
</tr>
</tbody>
</table>

The most commonly used VISA communication functions are the VISA Write and VISA Read functions.

Most instruments require you to send information in the form of a command or query before you can read information back from the instrument.

Therefore, the VISA Write function is usually followed by a VISA Read function.
The VISA Write and VISA Read functions work with any type of instrument communication and are the same whether you are doing GPIB or serial communication.

However, because serial communication requires you to configure extra parameters, you must start the serial port communication with the VISA Configure Serial Port VI.

**VISA and Serial**

- The VISA Configure Serial Port VI initializes the port identified by *VISA resource name* to the specified settings.
- *Timeout* sets the timeout value for the serial communication.
- *Baud rate, data bits, parity and flow control* specify those specific serial port parameters.
- The *error in* and *error out* clusters maintain the error conditions for this VI.
- The VISA Configure Serial Port VI opens communication with COM2 and sets it to 9,600 baud, eight data bits, odd parity, one stop bit and XON/XOFF software handshaking.
- Then, the VISA Write function sends the command.
- The VISA Read function reads back up to 200 bytes into the read buffer, and the Simple Error Handler VI checks the error condition.
- The VIs and functions located on the *Functions → All Functions → Instrument I/O → Serial* palette are also used for parallel port communication.
- You specify the VISA resource name as being one of the LPT ports.

**INSTRUMENT DRIVERS:**

- LabVIEW provides more than 1200 LabVIEW instrument drivers from more than 50 vendors.
- You can use these instrument drivers to build complete systems quickly.
- Instrument drivers drastically reduce software development costs because developers do not need to spend time programming their instruments.
- You can reuse the drivers in a variety of systems and configurations. LabVIEW instrument drivers simplify instrument programming to high-level commands, so you do not need to learn the low-level instrument-specific syntax needed to control your instruments.
- The programming is broken down into general functions for the DMM.
- These instrument drivers are called LabVIEW instrument drivers because the source code is graphical programming made from standard LabVIEW functions and VIs.
- LabVIEW instrument drivers are in the form of VI libraries and are organized into categories of instrument functions.
- Consider an example where you wrote a LabVIEW VI that communicates with a specific oscilloscope in your lab.
- Unfortunately, the oscilloscope no longer works, and you must replace it.
- However, this particular oscilloscope is no longer made.
- You found a different brand of oscilloscope that you want to purchase, but your VI no longer works with the new oscilloscope.
- You must rewrite your VI.
When you use an instrument driver, the driver contains the code specific to the instrument.

Therefore, if you change instruments, you must replace only the instrument driver VIs with the instrument driver VIs for the new instrument, which greatly reduces your redevelopment time.

Instrument drivers help make test applications easier to maintain because the drivers contain all the I/O for an instrument in one library, separate from other code.

When you upgrade hardware, upgrading the application is easier because the instrument driver contains all the code specific to that instrument.

A LabVIEW Plug and Play instrument driver is a set of VIs that control a programmable instrument. Each VI corresponds to an instrument operation, such as configuring, triggering and reading measurements from the instrument.

Instrument drivers help users get started using instruments from a PC and saves them development time and cost because users do not need to learn the programming protocol for each instrument.

With open-source, well documented instrument drivers, end users can customize their operation for better performance. A modular design makes the driver easier to customize.

You can locate most LabVIEW Plug and Play instrument driver in the Instrument Driver Finder.

You can access the Instrument Driver Finder within LabVIEW by selecting Tools » Instrumentation » Find Instrument Drivers or Help » Find Instrument Drivers.

You can find the LabVIEW instrument drivers on your LabVIEW installation CD or download them from the National Instruments web page at www.ni.com/idnet.

The Instrument Driver Network at ni.com is the industry’s largest source of instrument drivers, featuring drivers for over 2,200 instruments from 150 vendors.

Here, you can learn about, download, develop, and submit drivers for controlling instruments from LabVIEW, LabWindows/CVI, and Measurement Studio.

The LabVIEW Instrument Wizard can automatically install instrument drivers.

If you install the LabVIEW instrument drivers from the CD yourself or download them from the web page, you first decompress the instrument driver files to get a directory of instrument driver files.

Place this directory into the LabVIEW\instr.lib directory on your computer.

The next time you launch LabVIEW, you can access the instrument driver VIs from the Input » Instrument Drivers subpalette of the Functions palette.

Many programmable instruments have a large number of functions and modes.

With this complexity, it is necessary to provide a consistent design model that aids both instrument driver developers as well as end users who develop instrument control applications.

The LabVIEW Plug and Play instrument driver model contains both external structure and internal structure guidelines.

The external structure shows how the instrument driver interfaces with the user and to other software components in the system.
The internal structure shows the internal organization of the instrument driver software module.

For the external structure of the instrument driver, the user interacts with the instrument driver using an API or an interactive interface.

Usually, the interactive interface is used for testing or for end-users.

The API is accessed through LabVIEW. The instrument driver communicates with the instrument using VISA. Internally, the VIs in an instrument driver are organized into six categories.

These categories are summarized in Table 4.2.

---

**TABLE 4.2 VIs in an instrument driver**

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize</td>
<td>The initialize VI establishes communication with the instrument and is the first instrument driver VI called.</td>
</tr>
<tr>
<td>Configure</td>
<td>This collection of VIs are software routines that configure the instrument to perform specific operations. After calling these VIs, the instrument is ready to take measurements or stimulate a system.</td>
</tr>
<tr>
<td>Action/Status</td>
<td>This collection of VIs command the instrument to carry out an action (i.e. arming a trigger) or obtain the current status of the instrument or pending operations.</td>
</tr>
<tr>
<td>Data</td>
<td>The data VIs transfer data to or from the instrument.</td>
</tr>
<tr>
<td>Utility</td>
<td>This collection of VIs perform a variety of auxiliary operations, such as reset and self-test.</td>
</tr>
<tr>
<td>Close</td>
<td>The close VI terminates the software connection to the instrument. This is the last instrument driver VI called.</td>
</tr>
</tbody>
</table>

---

All instrument drivers in the library have the same VI Tree structure.

Therefore, once you learn to use one instrument driver, all others have the same basic hierarchy.

In fact, this hierarchy, sequence of VIs and error checking are used in many other areas of I/O in LabVIEW such as file I/O, data acquisition (DAQ) and TCP/IP communications.
The structure of an instrument driver is shown in Figure 4.10.
- The high-level functions are built from the lower-level functions.
- For the most control over the instrument, you would use the lower-level functions.
- However, the high-level functions like the *Getting Started VI* you used in the previous lesson are easy to use and have soft front panels that resemble the instrument.

**Instrument Driver VIs**
- As shown in Figure 4.11 an instrument driver VI initializes the DMM with its VISA Alias, uses an Application Example VI to configure and read data from the meter, and closes the instrument, and then the error status is checked.
- You will see this same sequence of events in every application that uses an instrument driver.
- Notice how the Instrument Descriptor, VISA Sessions, and Error I/O terminals are wired.
- Remember that you can right click on the instrument driver VI terminals and choose Create Constant, Create Control, or Create Indicator as needed.
- Instrument drivers drastically reduce software development costs because developers do not need to spend time programming their instruments.
- You can reuse the drivers in a variety of systems and configurations.
- The instrument driver VIs are as follows:
  - **Initialize**—Initializes the communication channel to the instrument. The initialize VI can optionally perform an identification query and reset operation. In addition, it can perform any necessary actions to place the instrument in its default power-on state or other specified state.
  - **Configure**—A collection of VIs that configure the instrument for the operations you want to perform. An example is a function to set up the trigger rate.
- **Action/Status**—Contains both action and status VIs. Action VIs cause the instrument to initiate or terminate test and measurement operations. Status VIs obtain the current status of the instrument or the status of pending operations. An example of an action function is *Acquire Single Shot*. An example of a status function is *Query Transfer Pending*.

- **Data**—VIs that transfer data to or from the instrument such as reading a measured waveform from the instrument or downloading a waveform to the instrument.

- **Utility**—VIs that perform a wide variety of useful functions such as reset, self-test, error query and revision query.

- **Close**—Terminates the communication channel to the instrument and deallocates the resources set aside for that instrument.

![Diagram](image.png)

**Figure 4.11** VIs to communicate with an instrument

- LabVIEW instrument drivers simplify instrument programming to high-level commands, so you do not need to learn the low-level instrument-specific syntax needed to control your instruments.

- The programming is broken down into general functions for the DMM.

**SERIAL PORT COMMUNICATION:**

- Serial communication is a popular means of transmitting data between a computer and a peripheral device such as a programmable instrument or even another computer.

- Serial communication uses a transmitter to send data, one bit at a time, over a single communication line to a receiver.

- You can use this method when data transfer rates are low or you must transfer data over long distances.

- Serial communication is popular because most computers have one or more serial ports, so no extra hardware is needed other than a cable to connect your instrument to the computer (or two computers together) as shown in Figure 4.12.
You must specify four parameters for serial communication:
- Baud rate of the transmission
- Number of data bits that encode a character
- Sense of the optional parity bit
- Number of stop bits.

A character frame packages each transmitted character as a single start bit followed by the data bits.

The baud rate is a measure of how fast data moves between instruments that use serial communication.

The baud rate informs how many bits are transferred per second on the serial cable.

Data bits indicate how many bits represent a data value.

Data bits are transmitted upside down and backwards, which means that inverted logic is used and the order of transmission is from the least significant bit (LSB) to the most significant bit (MSB).

Parity provides optional error checking bit that is added to the data.

Stop bits are a certain number of bits added to the end of each data transfer.

Flow control is optional hardware or software handshaking parameters for communicating with a device.

To interpret the data bits in a character frame as shown in Figure 4.13, you must read from right to left and read 1 for a negative voltage and 0 for a positive voltage.

An optional parity bit follows the data bits in the character frame.

The parity bit, if present, also follows inverted logic.

This bit is included as a means of error checking.

You specify ahead of time for the parity of the transmission to be even or odd.

If you choose for the parity to be odd, the parity bit is set in such a way so the number of 1s add up to make an odd number among the data bits and the parity bit.

The last part of a character frame consists of 1, 1.5, or 2 stop bits that are always represented by a negative voltage.

If no further characters are transmitted, the line stays in the negative (MARK) condition.

The transmission of the next character frame, if any, begins with a start bit of positive (SPACE) voltage.
RS-232 uses only two voltage states called MARK and SPACE.

In such a two-state coding scheme, the baud rate is identical to the maximum number of bits of information, including control bits that are transmitted per second.

MARK is a negative voltage, and SPACE is positive.

The following is the truth table for RS-232:

- Signal > +3 V = 0
- Signal < −3 V = 1

![Character frame diagram](image)

**Figure 4.13 A typical serial communication**

- The output signal level usually swings between +12 V and −12 V.
- The dead area between +3 V and −3 V is designed to absorb line noise.
- A start bit signals the beginning of each character frame.
- It is a transition from negative (MARK) to positive (SPACE) voltage. Its duration in seconds is the reciprocal of the baud rate.
- If the instrument is transmitting at 9,600 baud, the duration of the start bit and each subsequent bit is about 0.104 ms.
- The entire character frame of eleven bits would be transmitted in about 1.146 ms.
- Interpreting the data bits for the transmission yields 1101101 (binary) or 6D (hex).
- An ASCII conversion table shows that this is the letter m.
- This transmission uses odd parity.
- There are five ones among the data bits, already an odd number, so the parity bit is set to 0.
- Select COMX as the instrument address. Use the I/O Assistant just like with GPIB communication as shown in Figure 4.14.
Data Transfer Rate
- You can calculate the maximum transmission rate in characters per second for a given communication setting by dividing the baud rate by the bits per character frame.
- In the previous example, there are a total of eleven bits per character frame.
- If the transmission rate is set at 9,600 baud, you get $\frac{9,600}{11} = 872$ characters per second.
- Notice that this is the maximum character transmission rate.
- The hardware on one end or the other of the serial link might not be able to reach these rates, for various reasons.

Serial Port Standards
- There are several recommended standards (RS) for serial communication.
- Each varies in hardware and software specifications.
- One must be familiar with their instrument and what connector is used before you can begin controlling that device with their computer.
- Figure 4.15 shows serial hardware connection.
There are three main Serial I/O types that are most common recommended standards of serial port communication.

- **RS-232** (ANSI/EIA-232 Standard) is used for many purposes such as connecting a mouse, printer, or modem. It is also used with industrial instrumentation. Because of improvements in line drivers and cables, applications often increase the performance of RS-232 beyond the distance and speed in the standards list. RS-232 is limited to point-to-point connections between PC serial ports and devices. RS-232 is the connector found on most PCs. This is a single-ended communication method where only one device can be connected per port. Two connector types are 9 or 25-pin. Two configurations are DCE or DTE.

- **RS-422** (AIA RS-422A Standard) uses a differential electrical signal as opposed to the unbalanced (single-ended) signals referenced to ground with RS-232. Differential transmission, which uses two lines each to transmit and receive signals, results in greater noise immunity and longer transmission distances as compared to RS-232. RS-422 is the Connector found on most Macs. This is a differential communication method. Connector has 8 pins.

- **RS-485** (EIA-485 Standard) is a variation of RS-422 that allows you to connect up to 32 devices to a single port and define the necessary electrical characteristics to ensure adequate signal voltages under maximum load. With this enhanced multidrop capability, you can create networks of devices connected to a single RS-485 serial port. The noise immunity and multidrop capability make RS-485 an attractive choice in industrial applications that require many distributed devices networked to a PC or other controller for data collection and other operations.

**DATA ACQUISITION:**

- The fundamental task of a DAQ (Data Acquisition) system is to measure or generate real-world physical signals.
- Data acquisition involves gathering signals from measurement sources and digitizing the signal for storage, analysis and presentation on a personal computer (PC).
- Data acquisition systems come in many different PC technology forms for great flexibility.
- Scientists and engineers can choose from PCI, PXI, Compact PCI, PCMCIA, USB, Firewire, parallel, or serial ports for data acquisition in test, measurement, and automation applications.
The five components to be considered when building a basic DAQ system as shown in Figure 4.16 are transducers, signals, signal conditioning, DAQ hardware, and driver and application software.

**Figure 4.16** Data acquisition system.

**REVIEW OF TRANSDUCERS AND SIGNAL CONDITIONING:**
- Data acquisition begins with the physical phenomenon to be measured.
- This physical phenomenon could be the temperature of a room, the intensity of a light source, the pressure inside a chamber, the force applied to an object, or many other things.
- An effective DAQ system can measure all of these different phenomena.
- A transducer is a device that converts a physical phenomenon into a measurable electrical signal, such as voltage or current.
- The ability of a DAQ system to measure different phenomena depends on the transducers to convert the physical phenomena into signals measurable by the DAQ hardware.
- Transducers are synonymous with sensors in DAQ systems.
- There are specific transducers for many different applications, such as measuring temperature, pressure or fluid flow.
- Table 4.3 shows a short list of some common transducers and the phenomena they can measure.
- Different transducers have different requirements for converting phenomena into a measurable signal.
- Some transducers may require excitation in the form of voltage or current.
- Other transducers may require additional components and even resistive networks to produce a signal.
TABLE 4.3 Phenomena and existing transducers

<table>
<thead>
<tr>
<th>Phenomena</th>
<th>Transducer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temperature</td>
<td>Thermocouples</td>
</tr>
<tr>
<td></td>
<td>Resistive Temperature Devices (RTDs)</td>
</tr>
<tr>
<td></td>
<td>Thermistors</td>
</tr>
<tr>
<td>Light</td>
<td>Vacuum Tube</td>
</tr>
<tr>
<td></td>
<td>Photo Sensors</td>
</tr>
<tr>
<td>Sound</td>
<td>Microphone</td>
</tr>
<tr>
<td>Force and Pressure</td>
<td>Strain Gauges</td>
</tr>
<tr>
<td></td>
<td>Piezoelectric Transducers</td>
</tr>
<tr>
<td>Position and Displacement</td>
<td>Potentiometers</td>
</tr>
<tr>
<td></td>
<td>Linear Voltage Differential Transformer</td>
</tr>
<tr>
<td></td>
<td>Optical Encoder</td>
</tr>
<tr>
<td>Fluid</td>
<td>Head Meters</td>
</tr>
<tr>
<td></td>
<td>Rotational Flowmeters</td>
</tr>
<tr>
<td>pH</td>
<td>pH Electrodes</td>
</tr>
</tbody>
</table>

**Signals**
- The appropriate transducer converts the physical phenomena into measurable signals.
- However, different signals need to be measured in different ways.
- For this reason, it is important to understand the different types of signals and their corresponding attributes.
- Signals can be categorized into two groups: analog and digital.

**Analog Signals**
- An analog signal can be at any value with respect to time.
- A few examples of analog signals include voltage, temperature, pressure, sound and load.
- The three primary characteristics of an analog signal include level, shape and frequency as shown in Figure 4.17.
- *Level* gives vital information about the measured analog signal since analog signals can take on any value.
- The intensity of a light source, the temperature in a room, and the pressure inside a chamber are all examples that demonstrate the importance of the level of a signal.
- When measuring the level of a signal, the signal generally does not change quickly with respect to time.
- The accuracy of the measurement, however, is very important.
A DAQ system that yields maximum accuracy should be chosen to aid in analog level measurements.

Some signals are named after their specific shape—sine, square, sawtooth and triangle.

The shape of an analog signal can be as important as the level, because measuring the shape of an analog signal allows further analysis of the signal, including peak values, DC values and slope.

Signals where the shape is of interest generally change rapidly with respect to time, but system accuracy is still important.

The analysis of heartbeats, video signals, sounds, vibrations and circuit responses are some applications involving shape measurements.

All analog signals can be categorized by their frequency.

Unlike the level or shape of the signal, frequency cannot be directly measured.

The signal must be analyzed using software to determine the frequency information.

This analysis is usually done using an algorithm known as the Fourier transform.

When frequency is the most important piece of information, it is important to consider to include both accuracy and acquisition speed.

Although the acquisition speed for acquiring the frequency of a signal is less than the speed required for obtaining the shape of a signal, the signal must still be acquired fast enough that the pertinent information is not lost while the analog signal is being acquired.

The condition that stipulates this speed is known as the Nyquist sampling theorem.
Speech analysis, telecommunication and earthquake analysis are some examples of common applications where the frequency of the signal must be known.

**Digital Signals**
- A digital signal cannot take on any value with respect to time. Instead, a digital signal has two possible levels: *high* and *low*.
- Digital signals generally conform to certain specifications that define the characteristics of the signal.
- Digital signals are commonly referred to as Transistor-to-Transistor Logic (TTL).
- TTL specifications indicate a digital signal to be low when the level falls within 0 to 0.8 volts, and the signal is high between 2 to 5 volts.
- The useful information that can be measured from a digital signal includes the state and the rate as shown in Figure 4.18.

![Figure 4.18 Primary characteristics of a digital signal.](image)

- The state of a digital signal is essentially the level of the signal—on or off, high or low.
- Monitoring the state of a switch—open or closed—is a common application showing the importance of knowing the state of a digital signal.
- The rate of a digital signal defines how the digital signal changes state with respect to time.
- An example of measuring the rate of a digital signal includes determining how fast a motor shaft spins.
Unlike frequency, the rate of a digital signal measures how often a portion of a signal occurs.

A software algorithm is not required to determine the rate of a signal.

**SIGNAL CONDITIONING**

- Signal conditioning is the process of measuring and manipulating signals to improve accuracy, isolation, filtering, and so on.
- Many stand-alone instruments and DAQ devices have built-in signal conditioning.
- Signal conditioning also can be applied externally, by designing a circuit to condition the signal or by using devices specifically made for signal conditioning.
- National Instruments has SCXI devices and other devices that are designed for this purpose.
- Signal conditioning accessories can be used in a variety of important applications.
- Signal conditioning accessories amplify low-level signals and then isolate and filter them for more accurate measurements.
- In addition, some transducers use voltage or current excitation to generate a voltage output.
- Common types of signal conditioning are amplification, isolation, multiplexing, filtering, transducer excitation and linearization.
- Figure 4.19 shows some common types of transducers and signals and the signal conditioning each requires.

**Figure 4.19** Common transducers and signal conditioning types.
Sometimes transducers generate signals too difficult or too dangerous to measure directly with a DAQ device.

For instance, when dealing with high voltages, noisy environments, extreme high and low signals, or simultaneous signal measurement, signal conditioning is essential for an effective DAQ system.

Signal conditioning maximizes the accuracy of a system, allows sensors to operate properly and guarantees safety.

It is important to select the right hardware for signal conditioning.

Signal conditioning is offered in both modular and integrated forms as shown in Figure 4.20.

Signal conditioning accessories can be used in a variety of applications including amplification, attenuation, isolation, bridge completion, simultaneous sampling, sensor excitation, multiplexing, etc.

Other important criteria to consider with signal conditioning include packaging (modular versus integrated), performance, I/O count, advanced features, and cost.

**Amplification**

- Amplification is the most common type of signal conditioning.
- Amplifying electrical signals improves accuracy in the resulting digitized signal and reduces the effects of noise.
- Signals should be amplified as close to the signal source as possible.
- By amplifying a signal near the device, any noise that attached to the signal is also amplified.
- Amplifying near the signal source results in the largest signal-to-noise ratio (SNR).
- For the highest possible accuracy, amplify the signal so the maximum voltage range equals the maximum input range of the analog-to-digital converter (ADC).
Low-level thermocouple signals, for example, should be amplified to increase the resolution and reduce noise.

If you amplify the signal at the DAQ device while digitizing and measuring the signal, noise might have entered the lead wire, which decreases SNR.

However, if you amplify the signal close to the signal source with an SCXI module, noise has a less destructive effect on the signal, and the digitized representation is a better reflection of the original low-level signal.

SCXI has several signal conditioning modules that amplify input signals.

The gain is applied to the low-level signals within the SCXI chassis that are located near the transducers, so the module sends only high-level signals to the PC, minimizing the effects of noise on the readings.

**Isolation**

Another common signal conditioning application is isolating the transducer signals from the computer for safety purposes.

The system being monitored may contain high-voltage transients that could damage the computer without signal conditioning.

An additional reason for isolation is ensuring that the readings from the plug-in DAQ device are unaffected by differences in ground potentials or common-mode voltages.

When the DAQ device input and the signal being acquired are each referenced to ‘ground’, problems occur if there is a potential difference in the two grounds.

This difference can lead to what is known as a ground loop, which may cause inaccurate representation of the acquired signal; or if the difference is too large, it may damage the measurement system.

Using isolated signal conditioning modules eliminates ground loops and ensures that the signals are accurately acquired.

You also can use isolation to ensure that differences in ground potentials do not affect measurements from the DAQ device.

When you do not reference the DAQ device and the signal to the same ground potential, a ground loop can occur.

Ground loops can cause an inaccurate representation of the measured signal.

If the potential difference between the signal ground and the DAQ device ground is large, damage can occur to the measuring system.

Isolating the signal eliminates the ground loop and ensures that the signals are accurately measured.

**Multiplexing**

A common technique for measuring several signals with a single measuring device is multiplexing.

Signal conditioning hardware for analog signals often provides multiplexing for use with slowly changing signals like temperature.

The ADC samples one channel, switches to the next channel, samples it, switches to the next channel, and so on.

Because the same ADC samples many channels instead of one, the effective sampling rate of each individual channel is inversely proportional to the number of channels sampled.
For example, a PCI-MIO-16E-1 sampling at 1 MS/s on 10 channels will effectively sample each individual channel at:
\[
\frac{1 \text{ MS/s}}{10 \text{ channels}} = 100 \text{ kS/s per channel}
\]

SCXI modules for analog signals employ multiplexing so that as many as 3,072 signals can be measured with one DAQ device.

With the AMUX-64T analog multiplexer, you can measure up to 256 signals with a single device.

This feature is in addition to any built-in multiplexing on the DAQ device.

**Filtering**

- The purpose of a filter is to remove unwanted signals from the signal that you are trying to measure.
- A noise filter is used on DC-class signals, such as temperature, to attenuate higher frequency signals that can reduce your measurement accuracy.
- For example, many SCXI modules use 4 Hz and 10 kHz lowpass filters to eliminate noise before the signals are digitized by the DAQ device.
- AC-class signals, such as vibration, often require a different type of filter known as an antialiasing filter.
- Like the noise filter, the antialiasing filter is also a lowpass filter; however, it requires a very steep cutoff rate, so it almost completely removes all signal frequencies that are higher than the input bandwidth of the device.
- If the signals were not removed, they would erroneously appear as signals within the input bandwidth of the device.
- Devices designed specifically for AC-class signal measurement – the NI 455x, NI 445x, and NI 447x dynamic signal acquisition (DSA) devices, the NI 61xx simultaneous-sampling multifunction I/O devices, and the SCXI-1141 module have built-in antialiasing filters.

**Transduces Excitation**

- Signal conditioning systems can generate excitation, which some transducers require for operation.
- Strain gauges and RTDs require external voltage and currents, respectively, to excite their circuitry into measuring physical phenomena. This type of excitation is similar to a radio that needs power to receive and decode audio signals.
- Signal conditioning modules for these transducers usually provide these signals.
- RTD measurements are usually made with a current source that converts the variation in resistance to a measurable voltage.
- Strain gauges, which are very low-resistance devices, typically are used in a Wheatstone bridge configuration with a voltage excitation source.
- The SCXI-1121 and SCXI-1122 have onboard excitation sources, configurable as current or voltage that you can use for strain gauges, thermistors, or RTDs.

**Linearization**

- Another common signal conditioning function is linearization.
Many transducers, such as thermocouples, have a nonlinear response to changes in the physical phenomena you measure.
LabVIEW can linearize the voltage levels from transducers so you can scale the voltages to the measured phenomena.
LabVIEW provides scaling functions to convert voltages from strain gages, RTDs, thermocouples, and thermistors.
You should understand the nature of your signal, the configuration that is being used to measure the signal, and the effects of the environment surrounding the system.
Based on this information, you can determine whether signal conditioning will be a necessary part of your DAQ system.

DAQ HARDWARE CONFIGURATION

Before using a data acquisition board, you must confirm that the software can communicate with the board by configuring the devices.
The Windows Configuration Manager keeps track of all the hardware installed in the computer, including National Instruments DAQ devices.
If you have a Plug & Play (PnP) device, such as an E Series MIO device, the Windows Configuration Manager automatically detects and configures the device.
If you have a non-PnP device, or legacy device, you must configure the device manually using the Add New Hardware option in the Control Panel.
You can verify the Windows Configuration by accessing the Device Manager.

Measurement & Automation Explorer

LabVIEW installs Measurement & Automation Explorer (MAX) which establishes all device and channel configuration parameters.
After installing a DAQ device in the computer, you must run this configuration utility.
MAX reads the information the Device Manager records in the Windows Registry and assigns a logical device number to each DAQ device.
Use the device number to refer to the device in LabVIEW.
Access MAX either by double-clicking the icon on the desktop or selecting Tools»Measurement & Automation Explorer in LabVIEW.
The device parameters that you can set using the configuration utility depend on the device.
MAX saves the logical device number and the configuration parameters in the Windows Registry.
The plug and play capability of Windows automatically detects and configures switchless DAQ devices, such as the PCI-6024E.
When you install a device in the computer, the device is automatically detected.
Measurement & Automation Explorer, or MAX, is a software interface that gives you access to all National Instruments DAQ, GPIB, IMAQ, IVI, Motion, VISA, and VXI devices connected to your system.
The shortcut to MAX is placed on the desktop during installation of NI-DAQ.
MAX is used primarily to configure and test National Instruments hardware, but it offers other functionality, such as checking to see if you have the latest version of NI-DAQ installed.

The functionality of MAX is divided into four categories—Data Neighborhood, Devices and Interfaces, Scales and Software.

Scales
- You can configure custom scales for your measurements.
- This is very useful when working with sensors.
- It allows you to bring a scaled value into your application without having to work directly with the raw values.
- For example, in the exercises you use a temperature sensor that represents temperature with a voltage.
- The conversion equation for the temperature is: Voltage ¥ 100 = Celsius.
- After a scale is set, you can use it in your application program, providing the temperature value, rather than the voltage.

Simulating a DAQ Device
- You can create NI-DAQmx simulated devices in NI-DAQmx 7.4 or later.
- Using NI-DAQmx simulated devices, you can try NI products in your application without the hardware.
- When you later acquire the hardware, you can import the NI-DAQmx simulated device configuration to the physical device using the MAX Portable Configuration Wizard.
- With NI-DAQmx simulated devices, you also can export a physical device configuration onto a system that does not have the physical device installed.
- Then, using the NI-DAQmx simulated device, you can work on your applications on a portable system and upon returning to the original system, you can easily import your application work.

To create an NI-DAQmx simulated device, complete the following steps:
  - **Step 1:** Right-click **Devices and Interfaces** and select **Create»New**.
  - **Step 2:** A dialog box prompts you to select a device to add. Select **NI-DAQmx Simulated Device** and click **Finish**.
  - **Step 3:** In the **Choose Device** dialog box, select the family of devices for the device you want to simulate.
  - **Step 4:** Select the device and click **OK**. In the configuration tree in MAX, the icons for NI-DAQmx simulated devices are yellow. The icons for physical devices are green.
  - **Step 5:** If you select a PXI device, you are prompted to select a chassis number and PXI slot number.
  - **Step 6:** If you select an SCXI chassis, the SCXI configuration panels open.

To remove an NI-DAQmx simulated device, complete the following steps:
  - **Step 1:** Expand **Devices and Interfaces»NI-DAQmx Devices**.
  - **Step 2:** Right-click the NI-DAQmx simulated device you want to delete.
  - **Step 3:** Click **Delete**.
DAQ HARDWARE
- The DAQ hardware acts as the interface between the computer and the outside world.
- It primarily functions as a device that digitizes incoming analog signals so that the computer can interpret them.
- Other data acquisition functionality includes Analog Input/Output, Digital Input/Output, Counter/Timers and Multifunction which is a combination of analog, digital, and counter operations on a single device.
- National Instruments offers several hardware platforms for data acquisition.
- The most readily available platform is the desktop computer.
- National Instruments offers PCI DAQ devices that plug into any desktop computer.
- In addition, NI makes DAQ devices for PXI/CompactPCI, a more rugged modular computer platform specifically for measurement and automation applications.
- For distributed measurements, National Instruments Compact FieldPoint platform delivers modular I/O, embedded operation and Ethernet communication.
- For portable or handheld measurements, National Instruments DAQ devices for USB and PCMCIA work with laptops or Pocket PC PDAs as shown in Figure 4.21.
- A typical desktop DAQ system has three basic types of hardware—a terminal block, a cable and a DAQ device as shown in Figure 4.22.
- After you have converted a physical phenomenon into a measurable signal with or without signal conditioning, you need to acquire that signal.
- To acquire a signal, you need a terminal block, a cable, a DAQ device and a computer.
- This hardware combination can transform a standard computer into a measurement and automation system.

Terminal Block and Cable
- A terminal block provides a place to connect signals.
- It consists of screw or spring terminals for connecting signals and a connector for attaching a cable to connect the terminal block to a DAQ device.
- Terminal blocks have 100, 68, or 50 terminals.
- The type of terminal block you should choose depends on two factors—the device and the number of signals you are measuring.
- A terminal block with 68 terminals offers more ground terminals to connect a signal to than a terminal block with 50 terminals.
- Having more ground terminals prevents the need to overlap wires to reach a ground terminal, which can cause interference between the signals.

DAQ Signal Accessory
- Figure 4.22 shows the DAQ Signal Accessory.
- The DAQ Signal Accessory is a customized terminal block designed for learning purposes.
- It has 3 Connectors, Quadrature Encoder, Relay, Digital Trigger, 4 LEDs (reverse logic), Counter I/O, Function Generator, Function Generator Frequency Control, Temperature Sensor, Temperature Sensor Noise Control, Analog Input and Analog Output.

- The three different cable connectors accommodate many different DAQ devices and spring terminals to connect signals.

- You can access three analog input channels, one of which is connected to the temperature sensor and two analog output channels.

**Figure 4.21** DAQ hardware options.

**Figure 4.22** Typical DAQ system.
The DAQ Signal Accessory includes a function generator with a switch to select the frequency range of the signal, and a frequency knob.

- The function generator can produce a sine wave or a square wave.
- A connection to ground is located between the sine wave and square wave terminal.
- A digital trigger button produces a TTL pulse for triggering analog input or output.
- When you press the trigger button, the signal goes from +5 V to 0 V when pressed and returns to +5 V when you release the button.
- Four LEDs connect to the first four digital lines on the DAQ device.
- The LEDs use reverse logic, so when the digital line is high, the LED is off and vice versa.
- The DAQ Signal Accessory has a quadrature encoder that produces two pulse trains when you turn the encoder knob.
- Terminals are provided for the input and output signals of two counters on the DAQ device.
- The DAQ Signal Accessory also has a relay, a thermocouple input and a microphone jack.
- The exercises on DAQ will be based on the terminal block.

**DAQ Device**
Before a computer-based measurement system can measure a physical signal such as temperature, a sensor or transducer, must convert the physical or real world signal into an electrical one such as voltage or current.
You must use signal conditioning accessories to condition the signals before the plug-in DAQ device converts them to digital information.
The software controls the DAQ system by acquiring the raw data, analyzing and presenting the results.
Consider the following options for a DAQ system:
- The plug-in DAQ device shown in Figure 4.24 resides in the computer. You can plug the device into the PCI slot of a desktop computer or the PCMCIA slot of a laptop computer for a portable DAQ measurement system.
- The DAQ device is external and connects to the computer through an existing port, such as the serial port or Ethernet port, which means you can quickly and easily place measurement nodes near sensors.

![Figure 4.24 DAQ device.](image)

- The computer receives raw data through the DAQ device.
- The application you write presents and manipulates the raw data in a form you can understand.
- The software also controls the DAQ system by commanding the DAQ device when and from which channels to acquire data.
- Typically, DAQ software includes drivers and application software.
- Drivers are unique to the device or type of device and include the set of commands the device accepts.
- Application software, such as LabVIEW, sends the drivers commands, such as acquire and return a thermocouple reading.
- The application software also displays and analyzes the acquired data.
NI measurement devices include NI-DAQ driver software, a collection of VIs you use to configure, acquire data from and send data to the measurement devices.  
Most DAQ devices have four standard elements: analog input, analog output, digital I/O, and counters.
You can transfer the signal you measure with the DAQ device to the computer through a variety of different bus structures.
For example, you can use a DAQ device that plugs into the PCI bus of a computer, a DAQ device connected to the PCMCIA socket of a laptop, or a DAQ device connected to the USB port of a computer.
You also can use PXI/CompactPCI to create a portable, versatile, and rugged measurement system.
If you do not have a DAQ device, you can simulate one in Measurement and Automation Explorer to complete your software testing.
I/O Connector connects your signal (via terminal block and cable) to the DAQ device.
Computer I/O Interface Circuitry connects the DAQ device to the computer.
It can be a variety of bus structures PCI, PXI/Compact PCI, ISA/AT, PCMCIA, USB, IEEE 1394 (Firewire).
Real-Time System Integration (RTSI) Bus is used to synchronize multiple DAQ devices and allows sharing of timing and trigger signals between devices.
Analog Input Circuitry has a multiplexer (mux).
This switch has multiple input channels but only lets one at a time through to the instrumentation amplifier.
The instrumentation amplifier either amplifies or attenuates your signal.
Analog-to-Digital Converter (ADC) converts an analog signal to a digital number and is used for analog input.
The applications are circuit testing, power supply testing, dynamometer testing, weather station, geophysical studies and filter analysis.
Digital-to-Analog Converter (DAC) converts a digital number to an analog signal and is used for analog output.
The applications are control systems, function generator, tone generator and servo motor control.
Digital I/O Circuitry can input or output digital signals and is not suitable for measuring rate.
Applications include switch sensing, relay control and controlling LEDs.
Counter Circuitry can input or output digital signals.
It is suitable for measuring rate and has built in timing signals.
Applications includes stepper motor control, measuring frequency of a rotating shaft and oscillator testing.
Depending on your application, there are several different classes of PC-based data acquisition devices that you can use:

- Analog Input/Output
- Digital Input/Output
- Counter/Timers
- Multifunction—a combination of analog, digital and counter operations

**ANALOG INPUTS**

- Analog input is the process of measuring an analog signal and transferring the measurement to a computer for analysis, display or storage.
- An analog signal is a signal that varies continuously.
- Analog input is most commonly used to measure voltage or current.
- You can use many types of devices to perform analog input, such as multifunction DAQ (MIO) devices, high-speed digitizers, digital multimeters (DMMs) and Dynamic Signal Acquisition (DSA) devices.

**Analog-to-Digital Conversion**

- Acquiring an analog signal with a computer requires a process known as *analog-to-digital conversion* which takes an electrical signal and translates it into digital data so that a computer can process it.
- *Analog-to-digital converters* (ADCs) are circuit components that convert a voltage level into a series of ones and zeroes.
- ADCs sample the analog signal on each rising or falling edge of a sample clock.
- In each cycle, the ADC takes a snapshot of the analog signal, so that the signal can be measured and converted into a digital value.
- A *sample clock* controls the rate at which samples of the input signal are taken.
- Because the incoming, or unknown signal is a real world signal with infinite precision, the ADC approximates the signal with fixed precision.
- After the ADC obtains this approximation, the approximation can be converted to a series of digital values.
- Some conversion methods do not require this step, because the conversion generates a digital value directly as the ADC reaches the approximation.

**Task Timing**

- When performing analog input, the task can be timed to Acquire 1 Sample, Acquire n Samples or Acquire Continuously.
- Acquiring a *single sample* is an on-demand operation. In other words, the driver acquires one value from an input channel and immediately returns the value.
- This operation does not require any buffering or hardware timing.
- For example, if you periodically monitor the fluid level in a tank, you would acquire single data points.
You can connect the transducer that produces a voltage representing the fluid level to a single channel on the measurement device and initiate a single channel, single-point acquisition when you want to know the fluid level.

One way to acquire multiple samples for one or more channels is to acquire single samples in a repetitive manner.

However, acquiring a single data sample on one or more channels over and over is inefficient and time consuming.

Moreover, you do not have accurate control over the time between each sample or channel.

Instead you can use hardware timing, which uses a buffer in computer memory, to acquire data more efficiently.

Programmatically, you need to include the timing function and specify the sample rate and the sample mode (finite).

As with other functions, you can acquire multiple samples for a single channel or multiple channels.

With NI-DAQmx, you also can gather data from multiple channels.

For instance, you might want to monitor both the fluid level in the tank and the temperature.

In such a case, you need two transducers connected to two channels on the device.

If you want to view, process or log a subset of the samples as they are acquired, you need to continually acquire samples.

For these types of applications, set the sample mode to continuous.

Task Triggering

When a device controlled by NI-DAQmx does something, it performs an action.

Two very common actions are producing a sample and starting a waveform acquisition.

Every NI-DAQmx action needs a stimulus or cause.

When the stimulus occurs, the action is performed.

Causes for actions are called triggers.

The start trigger starts the acquisition.

The reference trigger establishes the reference point in a set of input samples.

Data acquired up to the reference point is pretrigger data.

Data acquired after the reference point is posttrigger data.

ANALOG OUTPUTS

Analog output is the process of generating electrical signals from your computer.

It is generated by performing digital-to-analog (D/A) conversions.

The available analog output types for a task are voltage and current.
To perform a voltage or current task, a compatible device must be installed that can generate that form of signal.

**Task Timing**

- When performing analog output, the task can be timed to Generate 1 Sample, Generate \( n \) Samples or Generate Continuously.
- To Generate 1 Sample use single updates if the signal level is more important than the generation rate.
- For example, generate one sample at a time if you need to generate a constant, or DC signal.
- You can use software timing to control when the device generates a signal.
- This operation does not require any buffering or hardware timing.
- For example, if you need to generate a known voltage to stimulate a device, a single update would be an appropriate task.
- One way to generate \( n \) samples or multiple samples for one or more channels is to generate single samples in a repetitive manner.
- However, generating a single data sample on one or more channels over and over is inefficient and time consuming.
- Moreover, you do not have accurate control over the time between each sample or channel.
- Instead, you can use hardware timing which uses a buffer in computer memory to generate samples more efficiently.
- You can use software timing or hardware timing to control when a signal is generated.
- With software timing, the rate at which the samples are generated is determined by the software and operating system instead of by the measurement device.
- With hardware timing, a TTL signal, such as a clock on the device, controls the rate of generation.
- A hardware clock can run much faster than a software loop.
- A hardware clock is also more accurate than a software loop.
- Programatically, you need to include the timing function, specifying the sample rate and the sample mode (finite).
- As with other functions, you can generate multiple samples for a single channel or multiple channels.
- Use Generate \( n \) Samples if you want to generate a finite time-varying signal, such as an AC sine wave.
- Continuous generation is similar to Generate \( n \) Samples, except that an event must occur to stop the generation.
- If you want to continuously generate signals, such as generating a non-finite AC sine wave, set the timing mode to continuous.
Task Triggering
- When a device controlled by NI-DAQmx does something, it performs an action.
- Two very common actions are producing a sample and starting a generation.
- Every NI-DAQmx action needs a stimulus or cause.
- When the stimulus occurs, the action is performed.
- Causes for actions are called triggers.
- A start trigger starts the generation.

Digital-to-Analog Conversion
- Digital-to-analog conversion is the opposite of analog-to-digital conversion. In digital-to-analog conversion, the data starts in the computer.
- The data might have been acquired earlier using analog input or may have been generated by software on the computer.
- A digital-to-analog converter (DAC) accepts this data and uses it to vary the voltage on an output pin over time.
- The DAC generates an analog signal that the DAC can send to other devices or circuits.
- A DAC has an update clock that tells the DAC when to generate a new value.
- The function of the update clock is similar to the function of the sample clock for an analog-to-digital converter (ADC).
- At each cycle the clock, the DAC converts a digital value to an analog voltage and creates an output as a voltage on a pin.
- When used with a high speed clock, the DAC can create a signal that appears to vary constantly and smoothly.

DIGITAL I/O (DIO)
- DIO interfaces are often used on PC DAQ systems to control processes, generate patterns for testing and communicate with peripheral equipment.
- In each case, the important parameters include the number of digital lines available, the rate at which you can accept and source digital data on these lines and the drive capability of the lines.
- If the digital lines are used for controlling events such as turning on and off heaters, motors or lights, a high data rate is usually not required because the equipment cannot respond very quickly.
- The number of digital lines, of course, should match the number of processes to be controlled.
- In each of these examples, the amount of current required to turn the devices on and off must be less than the available drive current from the device.
- DIO can also be used in industrial applications to verify that a switch is open or closed and to check the voltage levels as high or low.
- It can also be used for high-speed handshaking or simple communication methods.
- With the proper digital signal conditioning accessories, you can use the low-current TTL signals to/from the DAQ hardware to monitor/control high voltage and current signals from industrial hardware or to drive external relays.
- For example, the voltage and current needed to open and close a large valve may be on the order of 100 VAC at 2 A.
- Because the output of a DIO device is 0 to 5 VDC at several milliamperes, a signal conditioning module such as SCXI is needed to switch the power signal to control the valve.
- A common DIO application is to transfer data between a computer and equipment such as data loggers, data processors and printers.
- Because this equipment usually transfers data in one byte (8-bit) increments, the digital lines on a plug-in DIO device are arranged in groups of eight.
- In addition, some devices with digital capabilities will have handshaking circuitry for communication-synchronization purposes.
- The number of channels, data rate, and handshaking capabilities are all important specifications that should be understood and matched to the application needs.

**DAQ ASSISTANT**

- The DAQ Assistant is a graphical interface for interactively creating, editing, and running NIDAQmx virtual channels and tasks.
- A NI-DAQmx virtual channel consists of a physical channel on a DAQ device and the configuration information for this physical channel such as input range and custom scaling.
- A NI-DAQmx task is a collection of virtual channels, timing and triggering information, and other properties regarding the acquisition or generation.
- DAQ Assistant provides an interactive guide to configuring, testing and acquiring measurement data.
- With a single click, you can even generate code based on your configuration, making it easier and faster to develop complex operations.
- DAQ Assistant is completely menu-driven and you will encounter fewer errors.
- It drastically decreases the time to your first measurement.
- When you place this Express VI on the block diagram, the DAQ Assistant launches to create a new task.
- After you create a task, you can double-click the DAQ Assistant Express VI to edit that task.
- For continuous measurement or generation, place a While Loop around the DAQ Assistant Express VI.
Whenever a DAQmx Express VI is to be used in a While Loop, it is good practice to wire the stop condition into the stop input of the DAQmx Express VI.

Using the DAQ Assistant Express VI creates a task accessible only to the Express VI.

To make the task globally accessible from any application, you must convert the Express VI to an NI-DAQmx task saved in MAX.

In LabVIEW 8.0 and later, you can generate NI-DAQmx API code from DAQ Assistant Express VI.

Right-click the DAQ Assistant Express VI and select Generate NI-DAQmx Code from the shortcut menu to generate both configuration and example code for the task.

For continuous single-point input or output, the DAQ Assistant Express VI might not provide optimal performance.

In NI-DAQmx, virtual channels are integral to every measurement.

**Launch the DAQ Assistant**

- You can launch the DAQ Assistant in several ways.
- Complete the following steps to launch the DAQ Assistant from a LabVIEW block diagram.
- **Step 1:** Open LabVIEW and create a New VI. Switch to the block diagram (Ctrl+E).
- **Step 2:** DAQ Assistant Express VI is located in the Input subpalette of the Functions palette as in Figure 4.25. Place the DAQ Assistant on the block diagram by dragging and dropping it from the Functions palette. The Assistant should automatically launch when you drop the VI on the diagram.
Step 3: It is also available at Express>>Output>>DAQ Assistant. In the Advanced Functions palette, the DAQ Assistant Express VI is located in the NI Measurements >> DAQmx sub-palette as in Figure 4.26. The Create New window opens up for task configuration when the DAQ Assistant is placed on the block diagram. Measurement type can be Analog Input, Analog Output, Counter Input, Counter Output and Digital I/O. DAQ Assistant and configuration

Once you have located the DAQ Assistant Express VI in the appropriate location, select it from the palette and drop it on the block diagram of your VI.

By default, the properties page should pop up, allowing you to configure your task.

The first step is to select your type of measurement.

Create the Task

We will configure a simple voltage analog input measurement as in Figure 4.27.

The analog input task is specific to the measurement.

For other measurement and signal generation types, you would follow similar steps.

Step 1: On the first screen, select Analog Input for your Measurement Type.

Step 2: Next, select Voltage.

Step 3: The next screen lets you select the physical channel (or channels) for which you are creating this task. All supported data acquisition hardware devices should appear in the tree control and you can expand them to view a list of the
physical channels that you can select for your task. To select more than one channel, hold down the Ctrl button while clicking on the channel names.

- **Step 4:** Click Finish to move on to the configuration stage.

![Figure 4.27 Analog input measurement.](image)

**Configure the Task**
- After you create a task, you can configure channel-specific settings such as scaling, input limits and terminal configuration.
- You also can configure task-specific settings such as timing and triggering.
- In this task, you do not need to use scaling or triggering.
- You can configure settings like range (determines gain), terminal configuration, custom scaling and launch test panel.
- To configure the voltage measurement task, complete the following steps.
  - **Step 1:** Specify the input limits. You can use the default values of 5 for Max and –5 for Min if you do not know the theoretical limits for the signal you are measuring.
  - **Step 2:** Select the terminal configuration you used for the signal.
  - **Step 3:** On the Task Timing tab, select Acquire N Samples. Enter 100 for Samples To Read and enter 1000.00 for Rate (Hz) as shown in Figure 4.28.
Test the Task

- You can use test panels in the DAQ Assistant to test the task and make sure you connected the sensors properly.
- There is a test panel for each type of measurement. Complete the following steps to test the task.
- **Step 1:** Launch the test panel for your task by clicking the Test button at the top of the screen.
- **Step 2:** The test runs once automatically. Click the Start button to run the test again. Notice that the graph displays the acquired signal.
- **Step 3:** Click the Close button when you are done. If necessary, modify the settings for the task and retest the task.
- **Step 4:** After the test panel closes, click the OK button. The DAQ Assistant saves the voltage task, containing all the configuration information you entered, to MAX. You have created your voltage task.

Generate LabVIEW Code

- When you configure a task using the DAQ Assistant Express VI, the task is local to the application, and you cannot use it in other applications.
You can convert a DAQ Assistant Express VI to a DAQmx Task Name control if you want to save the task to MAX and use it in other applications or to generate code.

Complete the following steps to convert the DAQ Assistant Express VI to a DAQmx Task Name control.

**Step 1:** Right-click the DAQ Assistant Express VI and select *Convert to Task Name Constant* from the shortcut menu as shown in Figure 4.29.

**Step 2:** The DAQ Assistant launches, and you can modify the task, if necessary.

**Step 3:** Click the OK button.

**Step 4:** A DAQmx Task Name constant replaces the DAQ Assistant Express VI.

You now have three options for generating code in LabVIEW from a task or channel:

- **Example**—It generates all the code necessary to run the task or channel, such as VIs needed to read or write samples, VIs to start and stop the task, loops and graphs. Choose this option if you want to run the task or channel you created to verify that it works or to use your configuration in a simple application. In LabVIEW, this option adds to the VI you are working in or creates a new VI. The generated code is a simple NI-DAQmx example that you can then modify for your application.

- **Configuration**—It generates the code that replicates the configuration of the tasks and channels. LabVIEW replaces the DAQmx Task Name control with a subVI that contains VIs and property nodes used for channel creation and configuration, timing configuration and triggering.
configuration used in the task or channel. Choose this option if you want to deploy your application to another system. Refer to Deployment in the NI-DAQmx Help located at Start»Programs»National Instruments for more information. When you generate Configuration code, the link between the application and the DAQ Assistant is lost. Any changes you make to the configuration code is not reflected in the DAQ Assistant. You can regenerate Configuration code from the DAQ Assistant, but the regenerated code does not incorporate previous changes that you made to the code.

- **Configuration and Example**—It generates both Configuration code and Example code for the task or channel in one step.

Complete the following steps to generate code to run the voltage task.

- **Step 1:** Right-click the control on the front panel and select *Generate Code* » *Example* from the shortcut menu as in Figure 4.30.

![Figure 4.30 Generate code.](image)

- **Step 2:** View the block diagram. Notice that the DAQ Assistant generated all the code necessary to run the task, as shown in Figure 4.31. All of the timing information you set is contained in the DAQmx Task Name constant.

- **Step 3:** Save the VI as MyVoltageTask.vi.

- **Step 4:** View the front panel. Click the *Run* button to run the application.

- **Note:** Any changes you make to the generated code apply only to the VI and are not saved into the task configuration stored in MAX.
REFERENCES: