Computer-Based Instruments

NI-DMM Software User Manual

NATIONAL INSTRUMENTS™

March 2000 Edition
Part Number 370153A-01

Worldwide Technical Support and Product Information

`www.ni.com`

National Instruments Corporate Headquarters

11500 North Mopac Expressway   Austin, Texas 78759-3504   USA   Tel: 512 794 0100

Worldwide Offices

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20, Brazil 011 284 5011,
Canada (Calgary) 403 274 9391, Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521,
China 0755 3904939, Denmark 45 76 26 00, Finland 09 725 725 11, France 01 48 14 24 24,
Germany 089 741 31 30, Greece 30 1 42 96 427, Hong Kong 2645 3186, India 91805275406,
Israel 03 6120092, Italy 02 413091, Japan 03 5472 2970, Korea 02 596 7456, Mexico (D.F.) 5 280 7625,
Mexico (Monterrey) 8 357 7695, Netherlands 0348 433466, New Zealand 09 914 0488, Norway 32 27 73 00,
Poland 0 22 528 94 06, Portugal 351 1 726 9011, Singapore 2265886, Spain 91 640 0085,
Sweden 08 587 895 00, Switzerland 056 200 51 51, Taiwan 02 2528 7227, United Kingdom 01635 523545

For further support information, see the *Technical Support Resources* appendix. To comment on the documentation, send e-mail to `techpubs@ni.com`

Important Information

Conventions

The following conventions are used in this manual:

»	The **»** symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** from the last dialog box.
✎	This icon denotes a note, which alerts you to important information.
bold	Bold text denotes items that you must select or click on in the software, such as menu items and dialog box options. Bold text also denotes parameter names.
italic	Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. This font also denotes text that is a placeholder for a word or value that you must supply.
`monospace`	Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames and extensions, and code excerpts.
`monospace bold`	Bold text in this font denotes the messages and responses that the computer automatically prints to the screen. This font also emphasizes lines of code that are different from the other examples.

Contents

Glossary

Index

Figures

1 Introduction

This chapter discusses how to use this manual, lists what you need to get started, and presents a background of the NI-DMM software.

How to Use This Manual

Use this manual sequentially to learn how to set up a system to use National Instruments DMM hardware through the NI-DMM driver and through the NI-DAQ driver. Make sure you have all the components described in the *What You Need to Get Started* section. Also, check the `ReadmeDMM.htm` file installed with the instrument driver, which includes possible last-minute changes and updates. Refer to the where to start document that came with your hardware for instructions about setting up your system.

After you have set up your system, start with Chapter 2, *Getting Started*, which contains some general overview material to introduce some of the concepts used in the driver. Chapter 3, *Introductory Programming Examples Using Easy I/O and Application Functions*, will familiarize you with some simple examples. Chapter 4, *NI-DMM Driver Application Examples*, and Chapter 5, *NI-DAQ Driver Application Examples*, contain more in-depth information about the different elements that make up the NI-DMM software and about the usage of the DMM device with the NI-DAQ driver. Chapter 6, *Build Your Own Application*, suggests the sequence of steps needed to implement a measurement application from scratch.

In addition to the *NI-DMM Software User Manual*, you have the following documentation to use as reference.

- DMM hardware user manual—This manual gives information about the electrical and mechanical aspects and features of your National Instruments DMM device.
- Online help—This electronic document (`.hlp`) gives detailed information on the operations and attributes of NI-DMM.
- `ReadmeDMM.htm`—Check this file to see if there is any updated information regarding the hardware and software kit.

What You Need to Get Started

The following items are needed to get started.

- ❑ Appropriate National Instruments hardware, such as the NI 4050 or NI 4060 devices
- ❑ NI-DMM instrument driver software

Background

NI-DMM is designed to be an easy-to-use software interface to the line of National Instruments digital multimeters (DMMs). This driver is based on two industry standards for instrument drivers—VXI*plug&play* and Interchangeable Virtual Instruments (IVI).

VXI*plug&play*

The VXI*plug&play* Systems Alliance was formed to solve some of the remaining difficulties in integrating a VXI system. One of the most important components the VXI*plug&play* Systems Alliance addressed was that of the instrument driver. The VXI*plug&play* Systems Alliance created a standard for instrument drivers and required any VXI device have such an instrument driver to be VXI*plug&play* compliant. The NI-DMM instrument driver follows this standard by providing an instrument driver that conforms to the VXI*plug&play* standards.

IVI

In 1997, National Instruments promoted the creation of the IVI Foundation, an open consortium of companies chartered with the purpose defining software standards for interchangeable virtual instruments (IVI). As a result, the IVI Foundation (`www.ivifoundation.org`) has introduced an instrument driver model that is not only VXI*plug&play* compliant but also extends functionality to include many new features, such as state-caching and simulation modes, that users had requested. This architecture uses a support driver, known as the *IVI Engine*, to handle many of the complex features. The NI-DMM instrument driver is IVI compliant.

Note NI-DMM version 1.5 supports IVI-DMM Specification versions 1.0 and 2.0. Portions of the IVI-DMM Specification version 1.0 are considered obsolete, and later versions of NI-DMM will not support them. For more information about obsolete functions and attributes, we encourage you to carefully read the `ReadmeDMM.htm` file.

2 Getting Started

This chapter contains an overview of NI-DMM, defines terms you need to understand, and introduces the various operations defined by the VXI*plug&play* specifications on instrument drivers. The operations specified by these documents are standard across all VXI*plug&play* instrument drivers.

In addition, this chapter presents information on other operations unique to NI-DMM, and gives an overview of the main groups of attributes. For a complete list of the operations and their parameters, go to **Programs»National Instruments DMM»National Instruments DMM Help**.

For the latest versions of manuals, drivers, and examples, visit `www.ni.com/instruments` for free downloads.

Introduction

NI-DMM is designed to be consistent with the VXI*plug&play* Systems Alliance, VISA, and IVI technologies. These technologies have it possible for several object-oriented concepts to exist in the API. NI-DMM has three groups of information associated with it: session, attributes, and operations.

Sessions, or *handles*, are the unique identifiers of the object and are used to communicate with the object. Sessions are similar to file I/O handles.

Attributes give information about the object. For example, attributes of a car include its color, number of doors, and performance. Attributes of a DMM device include the measurement function, the resolution, the range, and the AutoZero setting. In addition, attributes can also control features of the DMM such as the type of trigger, trigger sources, and trigger destinations.

Operations are often called the *verbs of the object* because they describe what the object can do. For example, a car’s operations include accelerating, braking, and turning. Operations of a DMM card include initialization, initiating a reading, fetching a reading, and closing.

Session Communication

The NI-DMM driver is a *scalable* driver, which means it can talk to any of the National Instruments DMMs such as the NI 4050 or NI 4060. Therefore, when you want to use the driver, you must be able to uniquely identify the appropriate hardware. You can identify the hardware through `Initialize`, where you pass in a unique instrument descriptor of the hardware. This *descriptor* gives the driver the physical address of the hardware. The driver then returns a special handle (called an instrument handle) to you that you will use whenever you want to perform an action on this hardware. See Figure 2-1.

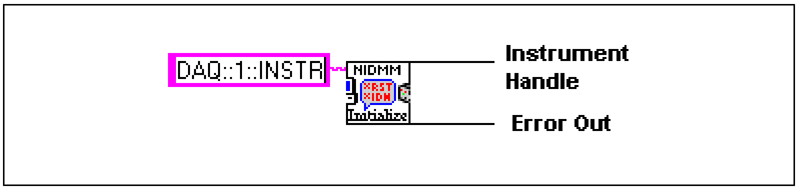


Figure 2-1. niDMM Initialize Block Diagram

```
status = niDMM_init("DAQ::1::INSTR", VI_TRUE, VI_TRUE, &instr);
```

This code opens communication with a DMM device with a device number value of 1. The session is returned in the last parameter, in this case in the variable `instr`.

As an alternative to `Initialize`, consider a similar operation—`Initialize With Options`. This operation initializes the driver into a state you specify using options you pass into the operation. For example, you can operate the NI-DMM driver in simulation mode, in which you can run programs without needing the DMM hardware to be connected to the computer.

To open a session to the simulation driver, you would use `Initialize With Options` as follows:

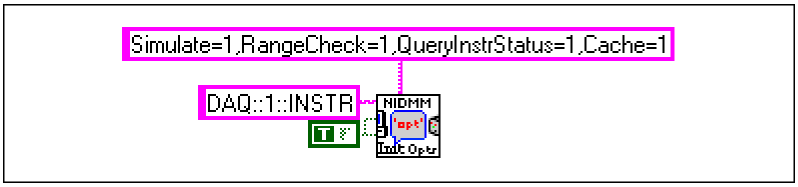


Figure 2-2. Initialize With Options Block Diagram

```
niDMM_InitWithOptions ("DAQ::2::INSTR", VI_ON, VI_ON,
                    "Simulate=0,RangeCheck=1,QueryInstrStatus=1,Cache=1",
                    &instr);
```

Note Refer to the online help for a detailed explaination of the different attributes, functions, and VIs.

To talk to the hardware, use this session as the first parameter of every operation from then on. See Figure 2-3.

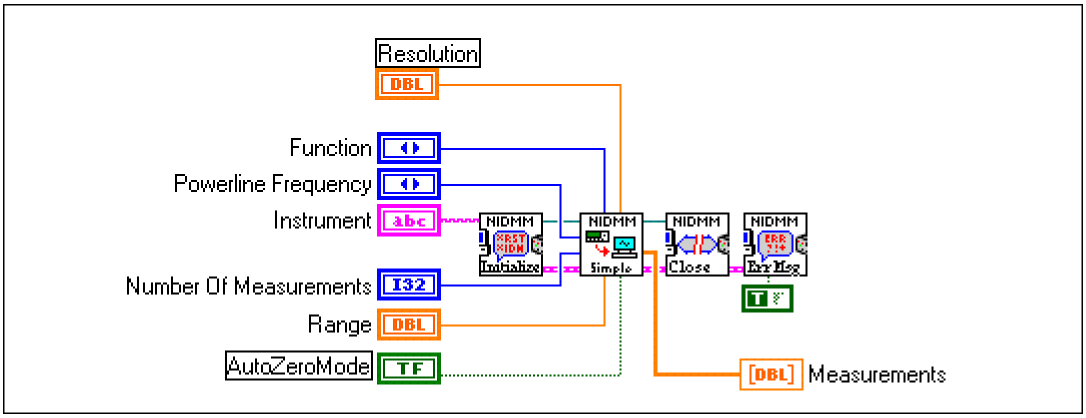


Figure 2-3. Session Block Diagram

Here is an example of opening a session with `niDMM_init` and passing the session further to the next function:

```
niDMM_init(resourceName, idQuery, reset, &instr));
niDMM_SimpleAcquisition(instr, powerlineFreq, function,
                    range,measCompDest,triggerSource, triggerDelay,
                    handInit,numOfMeas, measurements);
```

Note You can have only one session open to a unique piece of hardware at a time.

Due to the advanced features of IVI, such as state-caching, you should not have multiple sessions to—or multiple views of—the same hardware. Therefore, if your application is multi-threaded, each thread shares the same session. For protection between the threads, NI-DMM includes a set of lock operations.

Attributes

You can use attributes to get information about the state of the device or to set the state of the device. For example, by retrieving an attribute from the driver, you can determine whether the powerline frequency is set to 50 Hz or 60 Hz, or you can set the source of a trigger.

Accessing Attributes

Accessing attributes in LabVIEW works slightly differently than for C and Visual Basic. LabVIEW uses a feature known as the *property node*, which is available in the **Application Function** control panel. A LabVIEW programmer can use the property node to get and set multiple attributes at the same time. You can access the attributes in LabVIEW only through the property nodes, which display the list of possible attributes you can access.

C and Visual Basic users use special functions such as `niDMM_GetAttributeViInt32()` and `niDMM_SetAttributeViBoolean()`. In these languages, the attributes themselves are identified by their names, which always start with `NIDMM_ATTR_`. The following examples show how to get and set attributes in LabVIEW and C:

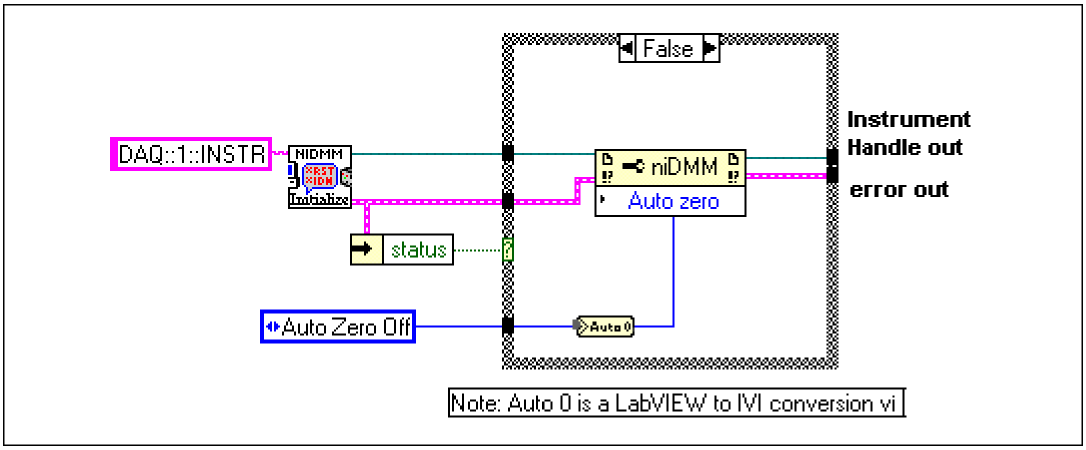


Figure 2-4. Set Auto Zero Block Diagram

```
status = niDMM_SetAttributeViInt32 (instr, "", NIDMM_ATTR_AUTOZERO,
                  &attrVal);
```

or

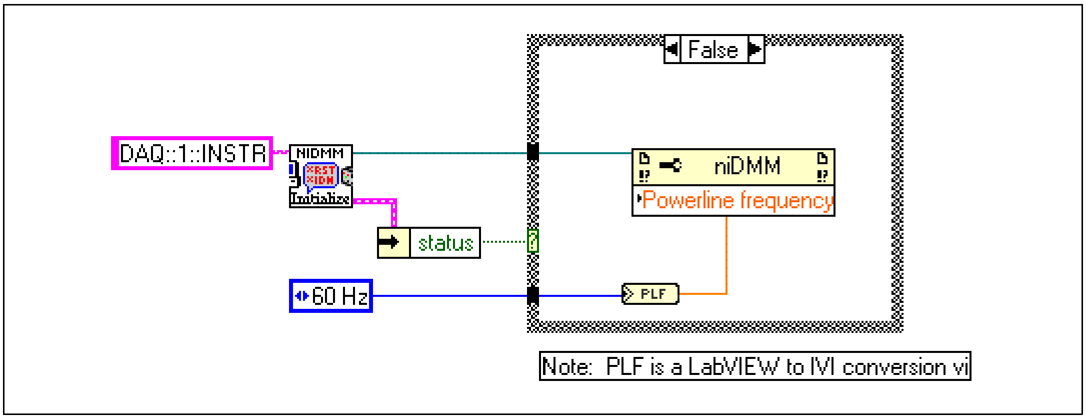


Figure 2-5. Set Powerline Block Diagram

```
niDMM_SetAttributeViInt32(instr, "", NIDMM_ATTR_POWERLINE_FREQ,
                            NIDMM_VAL_50_HERTZ);
```

These two operations also point out two other special features of the attribute operations. First, notice that the C operations take the data type of the attribute as part of the name. Therefore, each data type (Boolean, integer, and so on) has a unique function. Second, notice that in LabVIEW you can use the LabVIEW to IVI conversion VIs for passing attribute values.

Attribute Functionality

The NI-DMM online help fully describes all of the attributes for the NI-DMM driver. However, review this section for a general overview of the main groups, so you can better understand what information and control is available through the attributes.

Read-Only State Attributes

The first group of attributes is the read-only state attributes. A programmer can use these attributes to query the DMM device for basic information. Read-only state attributes make it possible for a program to be scalable across different boards because the program is able to configure itself depending on the actual hardware present (such as number of channels). Examples of read-only state attributes in the NI-DMM driver are as follows:

```
NIDMM_ATTR_NUM_CHANNELS
NIDMM_ATTR_AUTO_RANGE_VALUE
```

Other attributes are the read-write (R/W) type and contain specifications for the board such as fundamental capabilities (for example, function, resolution, and range), or extended functionality (for example, measurement complete, sample count, and sample interval).

```
NIDMM_ATTR_FUNCTION
NIDMM_ATTR_RESOLUTION
NIDMM_ATTR_AUTO_RANGE
NIDMM_ATTR_TRIGGER_SOURCE
NIDMM_ATTR_TRIGGER_DELAY
NIDMM_ATTR_AC_MIN_FREQ
NIDMM_ATTR_AC_MAX_FREQ
NIDMM_ATTR_SAMPLE_TRIGGER
NIDMM_ATTR_SAMPLE_INTERVAL
NIDMM_ATTR_TRIGGER_COUNT
NIDMM_ATTR_MEAS_COMPLETE_DEST
NIDMM_ATTR_POWERLINE_FREQ
NIDMM_ATTR_TRIGGER_SLOPE
```

Operations

An operation is another name for a VI in LabVIEW or a function in C or Visual Basic. These operations give you full access to the NI-DMM driver, including access to attributes (through the `get` and `set` operations) for C and Visual Basic users. Remember, in LabVIEW you use attribute nodes to access attributes. Consult the online help for more information on NI-DMM operations.

Overview of VXI*plug&play* Required Operations

Initialize and Close

As described earlier in this chapter, the central component to communicating with the driver and hardware is the session. Use `Initialize` and `Close` in your program to create and destroy the session, respectively. In addition, `Initialize` can require a verification that the address actually corresponds to the correct hardware, and can perform a complete reset on the board.

Reset

In `Initialize`, the program can order a board reset. However, you can achieve the same functionality through the `Reset` operation. In both cases, the reset causes the board to return to its powered-on state.

Self Test

The `Self Test` operation is designed to verify that the board is operational. The level of this verification depends on what information the driver can obtain about the board. In general, the NI-DMM driver can verify that the board is responding and that the driver can access the registers of the hardware.

Error Query and Error Message

All operations return status information that indicates whether the operation executed successfully. However, you can use the `Error Query` operation to retrieve the status code returned by the last operation called. `Error Message` translates an NI-DMM status code into human-readable text, which can be displayed via a dialog box.

Revision Query

`Revision Query` returns the revision of the instrument driver, in this case the revision of NI-DMM, as well as the firmware revision.

3 Introductory Programming Examples Using Easy I/O and Application Functions

This chapter contains examples that use NI-DMM to explore some of the basic functionality of your DMM hardware. The purpose of these examples is to introduce the programming concepts and to familiarize you with the instrument driver. Subsequent chapters describe these concepts in more detail. The examples in the chapters of this manual use LabVIEW and C source code, which you can use in the National Instruments LabVIEW, LabWindows/CVI, and Microsoft Visual C++ programming environments. The NI-DMM driver also works with Microsoft Visual Basic for Windows NT/98/95 programming environments. You can find equivalent examples for this environment in Appendix A, *Microsoft Visual Basic Examples*. The program shown in Example 3-1 is relatively simple. Later examples in this chapter build on the first example by repeating its basic structure and introducing additional features.

This manual uses a special font and naming convention for VI or function calls. For example, the term `Initialize`, when shown with this font, refers both to the LabVIEW VI, `niDMM Initialize.vi`, and to the C function, `niDMM_Init()`, shown in Figure 3-1.

You can find these VIs and functions as follows:

- LabVIEW—Instrument Drivers palette, inside the NI-DMM submenu
- LabWindows/CVI—load the intrument driver front panels (`nidmm.fp`), which are located in **VXIpnp/win95/NIDMM**

Note All examples are shown first as a LabVIEW VI then in C code.

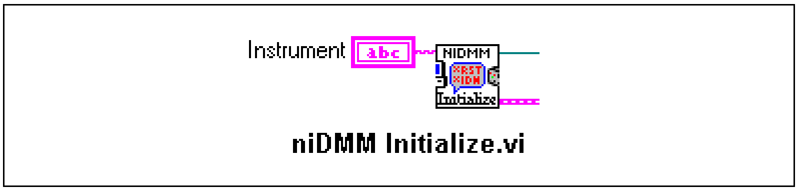


Figure 3-1. niDMM Initialize Block Diagram

```
niDMM_init(instr, idQuery, reset, handle);
```

Basic Startup

The NI-DMM application-programming interface (API) uses standard initialization and close routines at the beginning and end of the program. The initialization routine reserves computer resources for your DMM and creates a unique instrument handler (used by your program) to identify the DMM you are programming. Example 3-1 and Figure 3-2 show how to get a handle to the instrument and retrieve information about the state of the hardware.

Example 3-1. Initialization

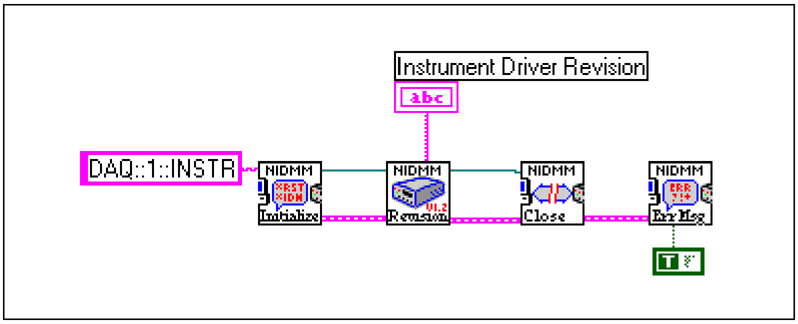


Figure 3-2. Initialize Example Block Diagram

```
int main (void)
{
     ViSession vi; /* Communication Channel */
     ViStatus status; /* For checking errors */
     ViChar firmRev[256]; /* Strings for revision info */
     ViChar driverRev[256];

     /* Begin by opening a communication channel to the instrument */
     checkErr(nidmm_init("DAQ::1::INSTR", VI_TRUE, VI_TRUE, &vi));
     checkErr(niDMM_revision_query (vi, driverRev,firmRev);
     Error:
     if (vi)
     niDMM_close(vi);
     if (error < VI_SUCCESS)
     messageHandler(error);
}

void _VI_FUNC messageHandler(ViStatus error)
     {
     ViChar errorMessage[256] = "";
     /*- Get the error description ---------------*/
     niDMM_error_message(VI_NULL, error, errorMessage);
     /*- Display the error message ----------------*/
     #ifdef _CVI_
          MessagePopup ("Error", errorMessage);
     #else
          MessageBox(NULL, errorMessage, "Error", MB_OK | MB_ICONERROR);
     #endif
     }
Note: checkErr is an IVI.h declared macro, as follows:
ifndef checkErr
#define checkErr(fCall)
     if (error = (fCall), (error = (error < 0) ? error : VI_SUCCESS)) {goto
     Error;}  else
#endif
```

Discussion

Example 3-1 breaks down into the following steps:

1. Open a communication channel to the device by using `Initialize`. Specify the address of the device you want to talk to through a VISA-style resource string. An example of a VISA-style resource string is `DAQ::n::INSTR`, *where* n is the device number you assigned to the instrument.

Note For more detailed information on how to set a device number, refer to your DMM's Where to Start document.

To use an analogy of telephone communication, this step compares to dialing a phone number. The operation places the call and connects you.

The variable returned—**VI**—is a *session*, or communication channel. This session represents the connection to the other person on the phone. In this case, the connection is to the actual DMM hardware.

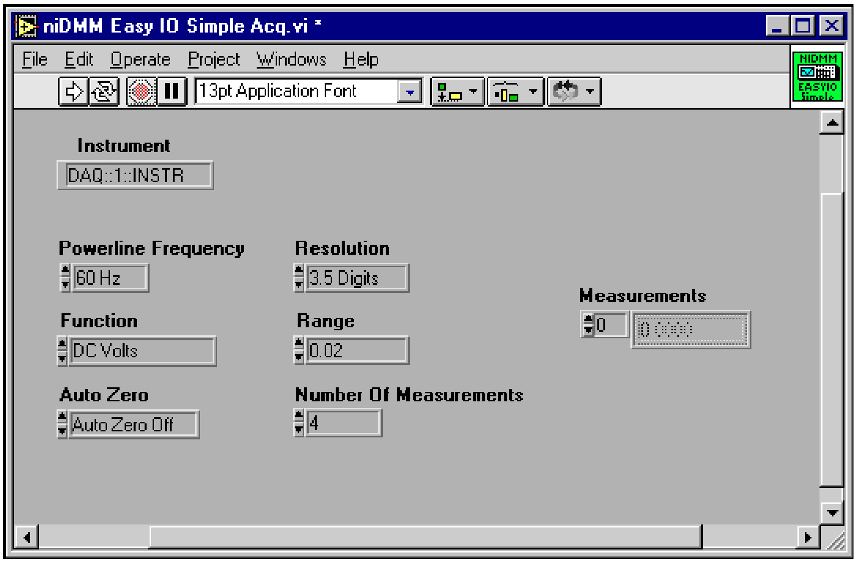
Note This operation is taken directly from the VXI*plug&play* specifications**.**

2. After successfully establishing a communication channel, you are ready to perform some action. In this example, you query the driver to check the revision of the driver.
3. At this point, you are done with this example. To finish the communication, you need to close the session. Closing the session will free up computer resources that were reserved with the `Initialize.vi` or function.

Example 3-2. Measuring DC Volts with the NI-DMM

Example 3-2 shows how to measure DC volts using the NI-DMM instrument driver. For this example, assume you have an NI-DMM device in your computer, your signal is connected to the top terminal pair, and you want to measure a DC volts signal.

If you are using LabVIEW, open the `niDMM Easy IO Simple Acq.vi` example, located in the **niDMM Instrument Driver»niDMM» Examples»Easy IO** subpalette.


niDMM Easy IO Simple Acq.vi *

File Edit Operate Project Windows Help

13pt Application Font

Instrument: DAQ::1::INSTR

Powerline Frequency: 60 Hz

Resolution: 3.5 Digits

Function: DC Volts

Range: 0.02

Auto Zero: Auto Zero Off

Number Of Measurements: 4

Measurements: 0 | 0.0000

Figure 3-3. niDMM Easy IO Simple Acquisition Front Panel

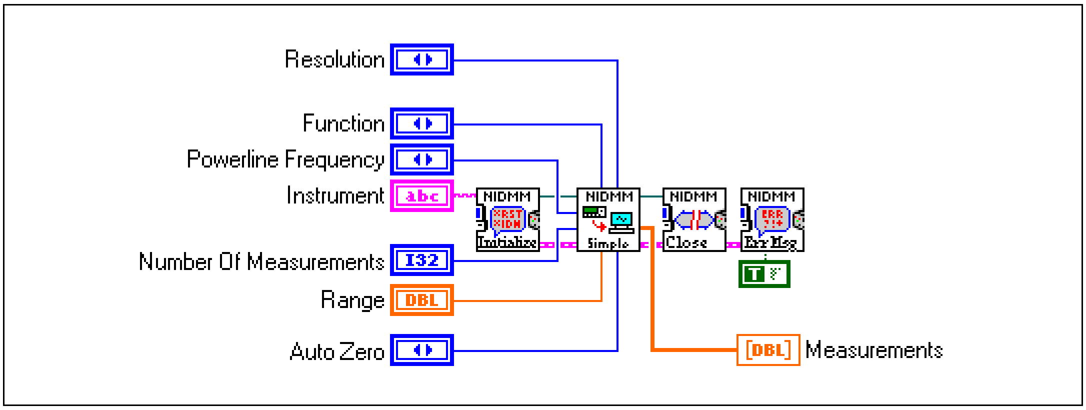


Figure 3-4. niDMM Easy IO Simple Acquisition Block Diagram

If you are using LabWindows/CVI or C programming, open the `simplAcq.prj` project, located in the `VXIPnP/Win95/niDMM/Examples/cvi` directory.

```
void main(void)
{
 ViStatus  error          = VI_SUCCESS;
 ViSession vi             = VI_NULL;
 ViString  resourceName   = "DAQ::1::INSTR";
 ViBoolean idQuery        = VI_TRUE;
 ViBoolean reset          = VI_TRUE;
 ViReal64  powerlineFreq  = NIDMM_VAL_60_HERTZ;
 ViReal64  resolution     = 0.00001;
 ViInt32   function       = NIDMM_VAL_DC_VOLTS;
 ViReal64  range          = 2.00;
 ViInt32   autoZero       = NIDMM_VAL_AUTO_ZERO_OFF;
 ViInt32   numOfMeas      = 4;
 ViReal64  measurements[4];
 checkErr(niDMM_init(resourceName, idQuery, reset, &vi));

 checkErr(niDMM_SimpleAcquisition(vi, powerlineFreq, function,
                   range,resolution, autoZero, numOfMeas, measurements));

 Error:
 if (vi)
  niDMM_close(vi);
 if (error < VI_SUCCESS)
  messageHandler(error);
}
```

Note A complete list of supported features for your DMM is listed in your DMM's user manual in the *Specifications* appendix.

Prior to measuring, you need to configure the hardware by setting your front panel controls as follows:

1. **Instrument**—This input control contains the resource name of the device to initialize. Valid Syntax: **`"DAQ::device number[::INSTR]"`**. Set Instrument to **`DAQ::1::INSTR`**, if your DMM was assigned deviceID 1 in Measurement & Automation Explorer.

Note To find the device number you assigned to your DMM, double-click the **Measurement & Automation Explorer** icon on your PC desktop, then click on **Devices and Interfaces**.

2. **Function**—The NI-DMM can perform many different measurement functions such as DC/AC voltage, DC/AC current, 2-wire/4-wire resistance, and diode measurements. Set **Function** to DC Volts.

3. **Range—**The NI-DMM has several input ranges available for measuring DC voltages. Set **Range** to the range input value that best accommodates your measurement.
4. **Resolution**—The NI-DMM has programmable resolution. Set the resolution to the absolute units you want (for example, 1, 0.1, 0.001, 0.00001, and so on). Note that you can calculate your digits of precision by using the formula:

   $$\text{digits} = \log_{10}(\text{range/absolute units})$$
5. **Powerline Frequency**—The NI-DMM hardware uses a digital filter that heavily rejects power line frequencies at 50 Hz and/or 60 Hz and their harmonics, as well as high-frequency noise. Set **Powerline Frequency** to 60 Hz.
6. **Number of Measurements**—This control represents the number of points the DMM reads. Set **Number of Measurements** to 4.
7. **Auto Zero—**When auto zeroing is on, the DMM internally disconnects the input signal following each measurement and takes a zero reading. The DMM then subtracts the zero reading from the preceding reading. This prevents offset voltages present on the DMM's input circuitry from affecting measurement accuracy. When auto zeroing is off, the DMM does not compensate for zero reading offset. Set **Auto Zero** to Off.

After you configure the hardware, run `niDMM Easy IO Simple Acq.vi`, or `SimplAcq.prj` in C.

8. **Measurements**—Measurement data is shown in the **Measurements** indicator.

Discussion

Example 3-2 breaks down into the following steps:

1. Open a communication channel to the device by using `Initialize`. You specify the address of the device you want to talk to through a VISA-style resource string (`DAQ::n::INSTR`, *where* `n` is the device number).
2. After successfully establishing a communication channel, you are ready to perform some action. In this example, you call the `niDMM Simple Acquisition` function, which executes and returns with the data points specified in **Number of Measurements**.
3. After the **Number of Measurements** data points are acquired, the function returns, and you are done with this example. To finish the communication, you need to close the session.

Measuring AC Volts with the NI-DMM

In the AC voltage ranges, the NI-DMM hardware measures the root mean squared (rms) value of a signal. Example 3-2 can also be used to measure AC volts with the NI-DMM hardware. Assume you have an NI-DMM device in your computer, your signal is connected to the top terminal pair, and you want to measure an AC volts signal. Set your front panel controls as follows:

- **Function**—Set **Function** to AC Volts.
- **Range**—The NI-DMM hardware has several input ranges available for measuring AC voltages. Set **Range** to the range input value that best accommodates your measurement.
- **Frequency**—The default frequency range for AC volts measurement is 20.0–25000.0 Hz. This example does not have a frequency input control on the panel. Look inside its diagram; open `niDMM Simple Acquisition.vi` by double-clicking on the VI icon; and then open `niDMM ConfigureACBandwidth.vi` by double-clicking on the icon in the `niDMM Simple Acquisition.vi` block diagram. You will see that this lower level configuration VI has two controls: one for the **Minimum Frequency** with the default value set to 20 Hz, and one for the **Maximum Frequency** with the default value set to 25000 Hz. These values specify the frequency band in which your AC signal may reside. If the default values are not valid for your signal, change these values appropriately.

Measuring Resistance with the NI-DMM

Example 3-2 can be used to measure 2-wire and 4-wire resistance with the NI-DMM device. The NI-DMM hardware measures 2-wire resistance by passing a current through the device under test and reading the resulting voltage drop through the same connections.

The 4-wire resistance measurement works in a similar manner as the 2-wire resistance measurement. The difference is that in the 4-wire resistance measurement, separate leads are used for the current excitation and for the voltage sense. This method allows greater accuracy for low value resistance measurement.

Assume you have an NI-DMM device in your computer, and your signal is connected appropriately to the DMM terminals in either the 2-wire or 4-wire resistance measurement configuration. Set your front panel controls to the following:

- **Function**—Set **Function** to 2-wire or 4-wire resistance.
- **Range**—The NI-DMM has several input ranges available for 2-wire or 4-wire measurements. Refer to your DMM's User Manual *Specifications* section for a complete listing of ranges.
- **Resolution**—Enter the appropriate resolution that best fits the resistance meaurement you want to take.

Measuring Current with the NI-DMM

Example 3-2 can be used to measure DC/AC current with the NI-DMM.

Assume you have an NI-DMM device in your computer, your signal is connected to the bottom terminal pair, and you want to measure DC/AC current. Set your front panel controls as follows:

- **Function**—Set **Function** to DC current or AC current.
- **Range**—The NI-DMM has several input ranges for current measurements. Set **Range** to the input range value that best accommodates your signal.

Testing Diodes with the NI-DMM

Example 3-2 can be used to measure a diode with the NI-DMM. The NI-DMM can measure the forward voltage of a diode. For this measurement, the diode is biased with a current, and the voltage drop across the diode is measured and returned.

Assume you have an NI-DMM device in your computer, your diode is connected to the top terminal pair, and you want to measure this diode. Set your front panel controls as shown in Example 3-2 with the following exception:

- **Function**—Set **Function** to Diode.

Note The diode measurements are made with a fixed range of 2 V.

4

NI-DMM Driver Application Examples

This chapter describes the classes of functions in NI-DMM and briefly describes each function. Figure 4-1 shows the acquisition model implemented by the NI-DMM instrument driver.

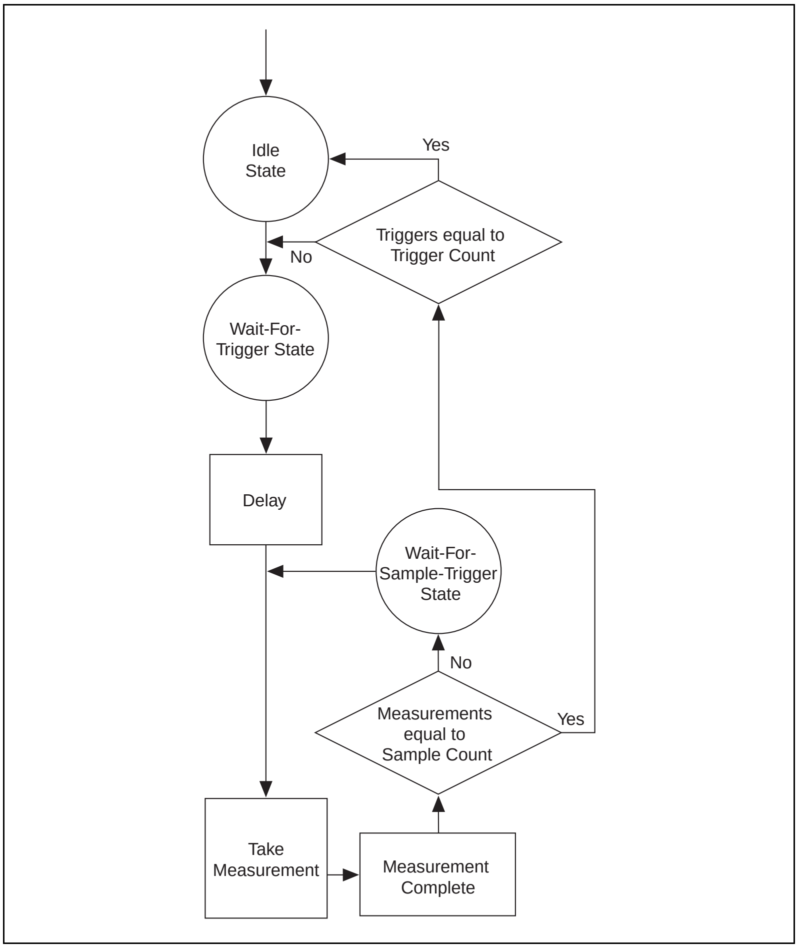

Idle State
Yes
Triggers equal to Trigger Count
No
Wait-For-Trigger State
Delay
Wait-For-Sample-Trigger State
No
Measurements equal to Sample Count
Yes
Take Measurement
Measurement Complete

Figure 4-1. Acquisition Model Flowchart

NI-DMM functions are grouped according to the following classes:

- **Examples**
  - `niDMM Getting Started`
  - `niDMM Easy IO Simple Acq`
  - `niDMM Easy IO Trigger Acq`
  - `niDMM Easy IO Scanning Acq`
  - `niDMM Easy IO Advanced Acq`
  - `niDMM Easy IO Interval Scanning Acq`
- **Open**
  - `Initialize`
  - `Initialize with Options`
- **Application Functions**
  - `niDMM Simple Acquisition`
  - `niDMM Triggered Acquisition`
  - `niDMM Advanced Acquisition`
  - `niDMM Scanning Acquisition`
  - `niDMM Interval Scanning Acquisition`
    - **Measure**
      - `niDMM Measure`
      - `niDMM Measure Multi Point`
      - **Configure**
        - `niDMM Configure Measurement`
        - `niDMM Configure Multi Point`
      - **Read**
        - `niDMM Read`
        - `niDMM Read Multi Point`
        - **Control**
          - `niDMM Initiate`
          - `niDMM Abort`
        - **Fetch**
          - `niDMM Fetch`
          - `niDMM Fetch Multi Point`
      - **Low-Level Measure**
        - `niDMM Send Software Trigger`

        - `niDMM Configure Trigger`
        - `niDMM Configure Powerline Frequency`
        - `niDMM Configure AC Bandwidth`
        - `niDMM Configure Meas Complete Dest`
        - `niDMM Configure Meas Complete Slope`
        - `niDMM Configure Trigger Slope`
        - `niDMM Configure Sample Trigger Slope`
        - `niDMM Configure Sample Delay Mode`
        - `niDMM Configure Auto Zero Mode`
        - `niDMM Get Digits of Precision`
        - `niDMM Is Over Range`
    - **Obsolete**
        - `niDMM Configure`
        - `niDMM Set Powerline Frequency`
        - `niDMM Configure Start Trigger`
        - `niDMM Configure Measurement Complete`
        - `niDMM Set Trigger Slope`
        - `niDMM Set Auto Zero`
        - `niDMM Format Measurements`
- **Utility**
    - `niDMM Reset`
    - `niDMM Self Test`
    - `niDMM Error Message`
    - `niDMM Error Query`
    - `niDMM Revision Query`
    - `niDMM Format Measurements Absolute`
    - `niDMM Read Status`
    - **Error Info**
        - `niDMM Get Error Info`
        - `niDMM Clear Error Info`
        - `niDMM Error Handler`
    - **Capabilities**
        - `niDMM Calculate Accuracy`
        - `niDMM Get Measurement period`
        - `niDMM Geet Auto Range Value`

- **Close**
  - `niDMM Close`

Note Lock and Unlock cannot be used in the LabVIEW environment.

Examples

Examples are simple applications that initialize the instrument, call an application function to configure measurement parameters, initiate the measurement, wait until the DMM has returned to the idle state, return the measured values, close the instrument session, and check for error.

For example, in the `niDMM Easy IO Simple Acq.vi`, `Initialize` generates a session that goes into `niDMM Simple Acquisition.vi` where the hardware is configured, acquisition is initiated, and data is retrieved. `niDMM Close` is called to terminate the session and to bring the instrument to its idle state.

To access this example, open `niDMM Easy IO Simple Acq.vi` located in the `Examples` folder of the NI-DMM driver.

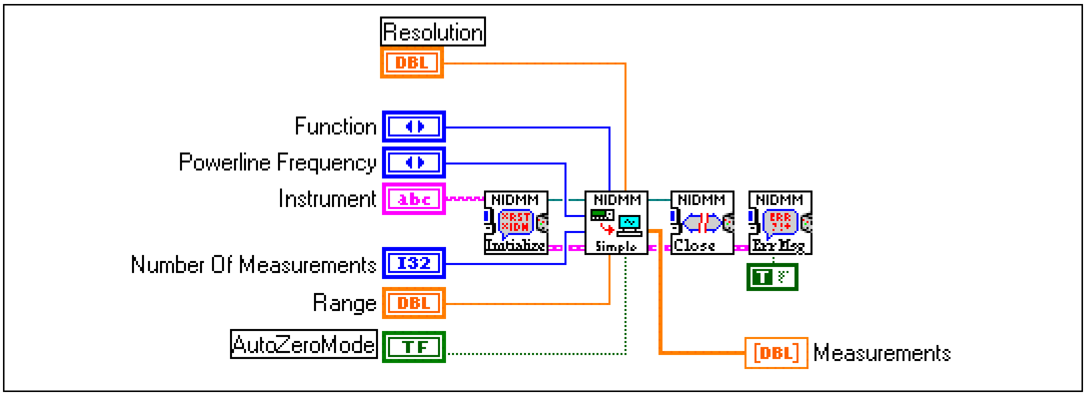


Figure 4-2. Easy I/O Simple Acquisition Block Diagram

Examples give immediate access to the DMM functionality. If your application does not require other configurations or other acquisition techniques than those covered by the examples, then you should use them directly as your application, or if your application requires other configuations, you can use them as a starting point in building your application.

Initialize and Close Functions

Initialize functions create a new IVI instrument driver session and open a session to the specified device using the device you specify for the **Resource Name** parameter. If the **ID Query** parameter is set to TRUE, Initialize functions query the instrument ID and check that it is valid for this instrument driver. If the Reset parameter is set to TRUE, Open functions reset the instrument to a known state.

Initialize functions send initialization commands to set the instrument to the state necessary for instrument driver operation; they also return a VI session handle that you use to identify the instrument in all subsequent instrument driver function calls.

Note Initialize functions create a new session each time you invoke them. Although, theoretically, you can open more than one IVI session for the same resource, as previously mentioned, it is best not to do so. You can use the same session in multiple program threads. You can use `LockSession` and `UnlockSession` VIs to protect sections of code that require exclusive access to the resource.

The `close` function closes the specified session and deallocates the resources that it occupied.

```
/* Open DMM */
status = niDMM_init("DAQ::16::INSTR", VI_TRUE, VI_TRUE, &instr);
/* Close DMM */
status = niDMM_close(instr);
```

Application Functions

Application functions are NI-DMM functions that configure the hardware for a certain type of acquisition (simple, triggered, scanning, and advanced), initiate the acquisition, wait until the DMM has returned to the idle state, and return the measured values.

If you want to build an application using application functions, you need to first initialize the device by calling `Init` or `InitWithOptions` and then close the session with `Close`. If you wish to configure additional parameters in your application, you can do so by calling `SetAttribute` or other configuration VIs in the driver library before calling the application function and after calling `Init`. The `niDMM Easy IO Simple Acq.vi` is an example of how to use an application function like `niDMM Simple Acquisition`.

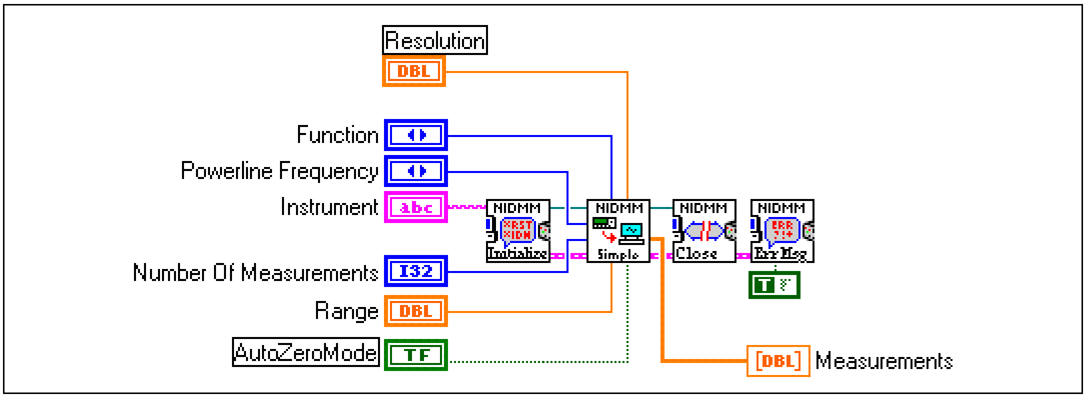


Figure 4-3. Simple Acquisition Block Diagram

```
void main(void)
{
      ViStatus  error                = VI_SUCCESS;
      ViSession vi                   = VI_NULL;
      ViString  resourceName         = "DAQ::1::INSTR";
      ViBoolean idQuery              = VI_TRUE;
      ViBoolean reset                = VI_TRUE;
      ViReal64  powerlineFreq        = NIDMM_VAL_60_HERTZ;
      ViReal64  resolution           = 0.01;
      ViInt32   function             = NIDMM_VAL_DC_VOLTS;
      ViReal64  range                = 2.00;
      ViInt32   autoZero             = NIDMM_VAL_AUTO_ZERO_OFF;
      ViInt32   numOfMeas            = 4;
      ViReal64  measurements[4];

checkErr(niDMM_init(resourceName, idQuery, reset, &vi));

error = niDMM_SimpleAcquisition (vi, powerlineFreq, function, range,
                    resolution, autoZero, numOfMeas, measurements);

Error:
if (vi)
niDMM_close(vi);

if (error < VI_SUCCESS)
messageHandler(error);

}
```

Advanced Programming Using Low-Level Functions

Functions like `niDMM Simple Acquisition` are themselves made of lower level specialized functions, which configure the hardware, read one or several data points, or initiate acquisition and fetch one or several readings into the RAM buffer.

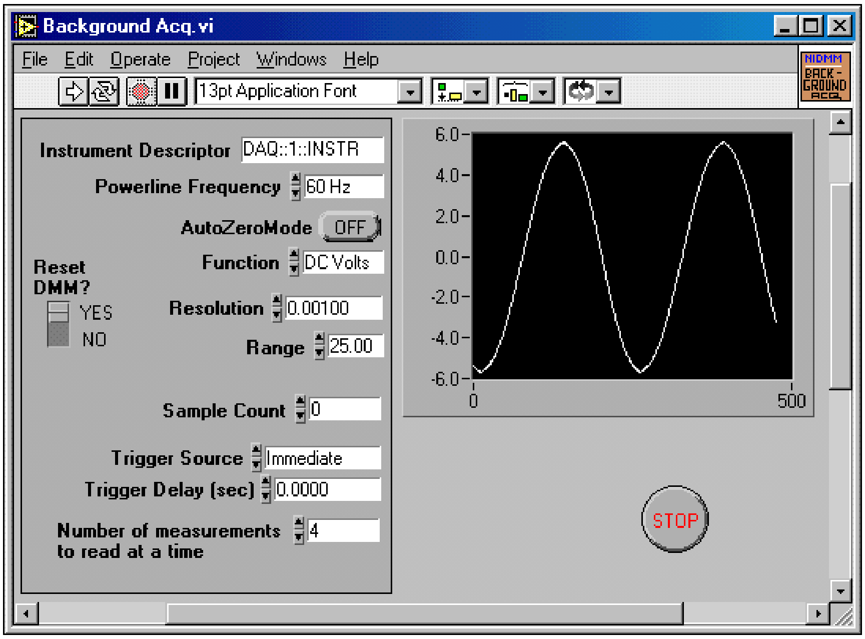
These functions are not visible at the Application Functions level, but certainly—as lower level functions—are part of the driver; you may use them if your application so requires. After configuring the DMM, you can choose to implement the acquisition in the foreground or in the background of your application.

Background Acquisition

A background acquisition will initiate the data acquisition process and return to the next function caller immediately. This process is implemented in NI-DMM by using the sequence of calls, `niDMM Initiate/niDMM Fetch`.

The practical result of this programming method is that after executing `niDMM Initiate`, the control is returned immediately to your program while, in the background, the DMM is acquiring data, converting it into digital format, and sending it to your computer's RAM memory.

In order to configure a continuous acquisition, you need to set Sample Count to 0 in `niDMM Configure Multi Point`. In this case, the acquisition process will continue until `niDMM Abort` is called. If you set Sample Count—in `niDMM ConfigureMultipoint`—to a finite number *N*, your acquisition will stop after *N* points have been acquired. The *sample count* is the total number of measurements you want the DMM to perform. The following example shows how to implement a continuous background acquisition.



Instrument Descriptor: DAQ::1::INSTR
Powerline Frequency: 60 Hz
AutoZeroMode: OFF
Reset DMM?: YES [✓ NO]
Function: DC Volts
Resolution: 0.00100
Range: 25.00
Sample Count: 0
Trigger Source: Immediate
Trigger Delay (sec): 0.0000
Number of measurements to read at a time: 4

Figure 4-4. Background Acquisition Front Panel

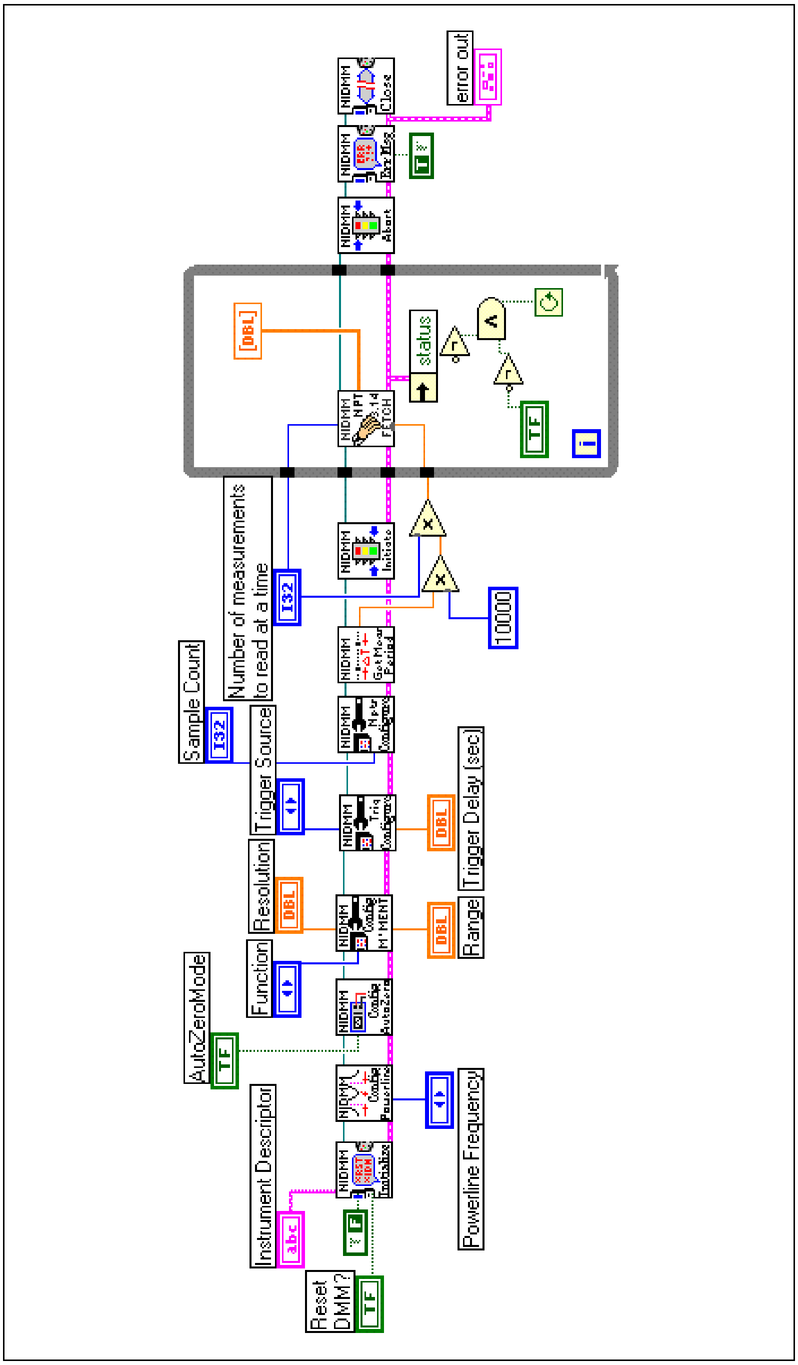


Figure 4-5. Background Acquisition Block Diagram

```
void main(void)
   {
   ViStatus  error           = VI_SUCCESS;
   ViSession vi              = VI_NULL;
   ViString  resourceName    = "DAQ::1::INSTR";
   ViBoolean idQuery         = VI_TRUE;
   ViBoolean reset           = VI_TRUE;
   ViReal64  powerlineFreq   = NIDMM_VAL_60_HERTZ;
   ViInt32   function        = NIDMM_VAL_DC_VOLTS;
   ViReal64  range           = 25.00;
   ViInt32   numOfMeas       = 10;
   ViReal64  measurements[10];
   ViReal64  resolution      = 0.001;
   ViInt32   i;
   ViInt32   numPointsRead;

   checkErr(niDMM_init(resourceName, idQuery, reset, &vi));
  /*- Configure Powerline frequency--------------------------------------*/
   checkErr(niDMM_ConfigurePowerlineFrequency(vi, powerlineFreq));

   /*- Call NIDMM_ConfigureMeasurement() to set function, range and res---*/
   checkErr(niDMM_ConfigureMeasurement(vi, function, range, resolution));

   /*- Configure a multipoint acquisition---------------------------------*/
  checkErr(niDMM_ConfigureMultiPoint(vi, 1, numOfMeas, NIDMM_VAL_IMMEDIATE,
                   0.0));

 /*- Start or initiate the acquisition-----------------------------------*/
   checkErr(niDMM_Initiate(vi));

  /*- Fetch the data-----------------------------------------------------*/
   checkErr(niDMM_FetchMultiPoint(vi, NIDMM_VAL_TIME_LIMIT_AUTO, numOfMeas,
                   measurements, &numPointsRead));
   for (i=0;i<10;i++)
         printf("measurements[%d] = %f \n", i, measurements[i]);
   printf("\n\n");

   Error:
   if (vi)
      niDMM_close(vi);

   if (error < VI_SUCCESS)
      messageHandler(error);
   }
```

This previous example is using a sequence of low-level NI-DMM driver functions to implement the following functionality:

1. Open a session on the device described by the instrument descriptor.
2. Configure the measurement.
3. Configure powerline frequency.
4. Configure trigger.
5. Configure a multipoint acquisition. Set the sample count, which represents the number of points to acquire. If sample count is set to 0, this configures for a continuous acquisition operation, which eventually will be turned off by using the abort function.
6. Get the measurement period to determine how much time in milliseconds a single measurement point takes. You can use this number to specify the timeout value in the `Fetch` function as an alternative to using the auto time limit value to wait between consecutive readings (in `Fetch`).
7. Initiate the acquisition. `niDMM Initiate` starts a background acquisition process and returns to the function caller immediately.
8. Start fetching four points at a time inside a while loop.
9. Go out of the while loop when the **Stop** button is pressed (LabVIEW example), or any key is hit (C code).
10. Abort continuous acquisition (LabVIEW example).
11. Close the session.
12. Check for errors.

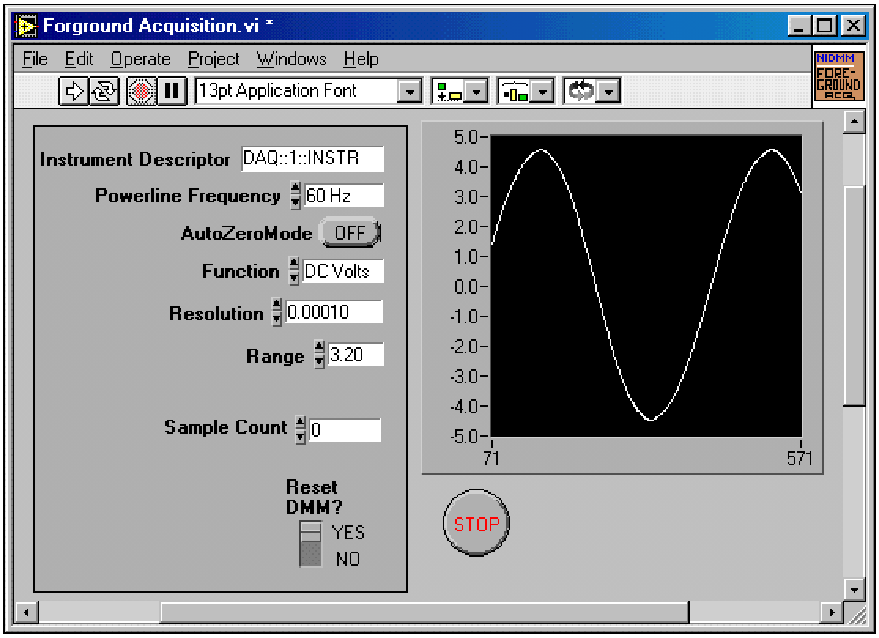
`GetMeasurementPeriod` calculates the period of time it takes to make one measurement in hardware, as a function of powerline frequency, resolution, and range. Each of the `niDMM Fetch` calls, returns data from an ongoing acquisition (without interrupting the acquisition), and transfers the last acquired data point from the acquisition buffer into the read buffer for further processing.

If `Fetch` does not complete within the timeout period given by the maximum time input control, the function returns the `MAX_TIME_EXCEEDED` error code. When this error occurs, you can call `Abort` to cancel the measure operation and `Close` to return the instrument to the idle state. Current implementation of the NI-DMM driver provides a way of monitoring the data acquisition in progress. Basically, you are retrieving data with `niDMM Fetch` in two ways:

- When you are ready to do so
- When data is available by calling `niDMM Read Status`

Foreground Acquisition

The sequence of calls used above will start the acquisition once, then read data, inside a loop, while data is acquired. A foreground acquisition function, like `niDMM_Read`, will not return until the NI-DMM has acquired the data you requested. `niDMM_Read` initiates an acquisition (of 1 or *N* points), stops the acquisition, reads the acquired data (1 or *N* points), and returns to the next function caller.



Instrument Descriptor: DAQ::1::INSTR
Powerline Frequency: 60 Hz
AutoZeroMode: OFF
Function: DC Volts
Resolution: 0.00010
Range: 3.20
Sample Count: 0
Reset DMM? YES NO

Figure 4-6. Foreground Acquisition Front Panel

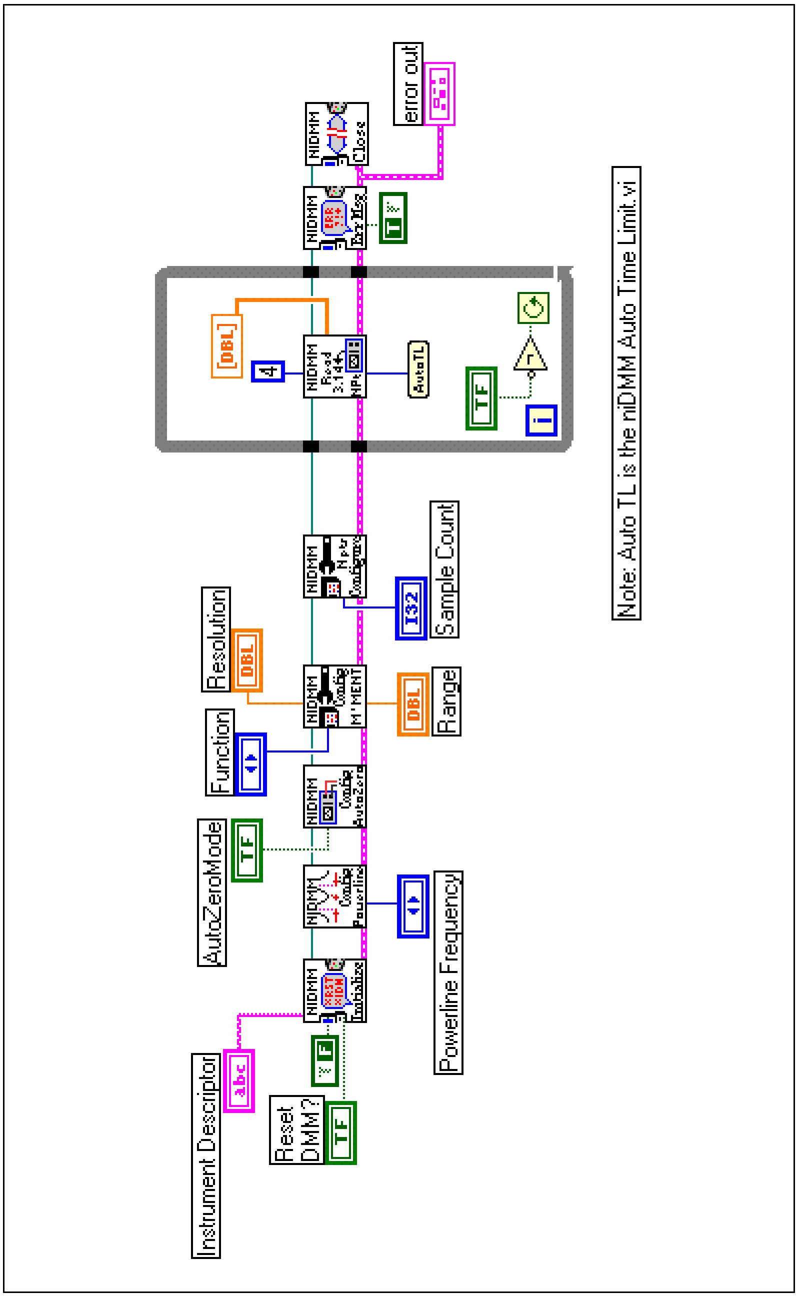


Figure 4-7. Foreground Acquisition Block Diagram

```
void main(void)
   {
   ViStatus  error           = VI_SUCCESS;
   ViSession vi              = VI_NULL;
   ViString  resourceName    = "DAQ::1::INSTR";
   ViBoolean idQuery         = VI_TRUE;
   ViBoolean reset           = VI_TRUE;
   ViReal64  powerlineFreq   = NIDMM_VAL_60_HERTZ;
   ViInt32   function        = NIDMM_VAL_DC_VOLTS;
   ViReal64  range           = 25.00;
   ViInt32   numOfMeas       = 10;
   ViReal64  measurements[10];
   ViReal64  resolution      = 0.001;
   ViInt32   i;
   ViInt32   numPointsRead;
   ViReal64  reading;

   checkErr(niDMM_init(resourceName, idQuery, reset, &vi));

   /*- Configure the powerline frequency----------------------------------*/
   checkErr(niDMM_ConfigurePowerlineFrequency(vi, powerlineFreq));

   /*- Call NIDMM_ConfigureMeasurement() to set function, range and res---*/
   checkErr(niDMM_ConfigureMeasurement(vi, function, range, resolution));

   /*- Configure a multipoint acquisition---------------------------------*/
   checkErr(niDMM_ConfigureMultiPoint(vi, 1, numOfMeas, NIDMM_VAL_IMMEDIATE,
                    0.0));

   /*- Read measurements using ReadMultiPoint()---------------------------*/
   checkErr(niDMM_ReadMultiPoint (vi, NIDMM_VAL_TIME_LIMIT_AUTO, numOfMeas,
                    measurements, &numPointsRead));
   for (i=0;i<10;i++)
                    printf("measurements[%d] = %f \n", i, measurements[i]);
   printf("\n\n");

   Error:
   if (vi)
      niDMM_close(vi);

   if (error < VI_SUCCESS)
      messageHandler(error);
   }
```

In the *Background Acquisition* section, `GetMeasurementPeriod` calculated the period of time it takes to make one measurement. In this example, the `GetMeasurementPeriod` call is removed, and instead the maximum length of time in which to allow the measure operation to complete using the auto time limit read value is passed to `niDMM_ReadMultiPoint` . If the measurement operation does not complete within this time interval, the `niDMM_ReadMultiPoint` function returns the `MAX_TIME_EXCEEDED` error code.

Triggered Acquisition

The DMM hardware can be configured to:

- self trigger its measurements based on an internal clock
- perform its measurements based on an externally provided trigger

Note The triggering features shown in this section do *not* apply to the NI 4050 for PCMCIA.

If the DMM is configured for external trigger, a pulse received on the EXT TRIG IN line, or one of the TTL lines, will trigger a measurement operation. The DMM has two types of triggers: `Trigger` and `Sample Trigger`.

A `Trigger` is an edge-triggered signal that initiates a data acquisition sequence, which is a collection of samples (A/D conversions). A `Sample Trigger` is a signal that causes an A/D conversion to be initiated. Both `Trigger` and `Sample Trigger` can be generated externally or internally. The NI-DMM instrument driver is implementing two types of trigger pulses:

- Immediate (or internal)
- External
  - Applied on the EXT TRIG IN line
  - Applied on a PXI TTL line (0-6)

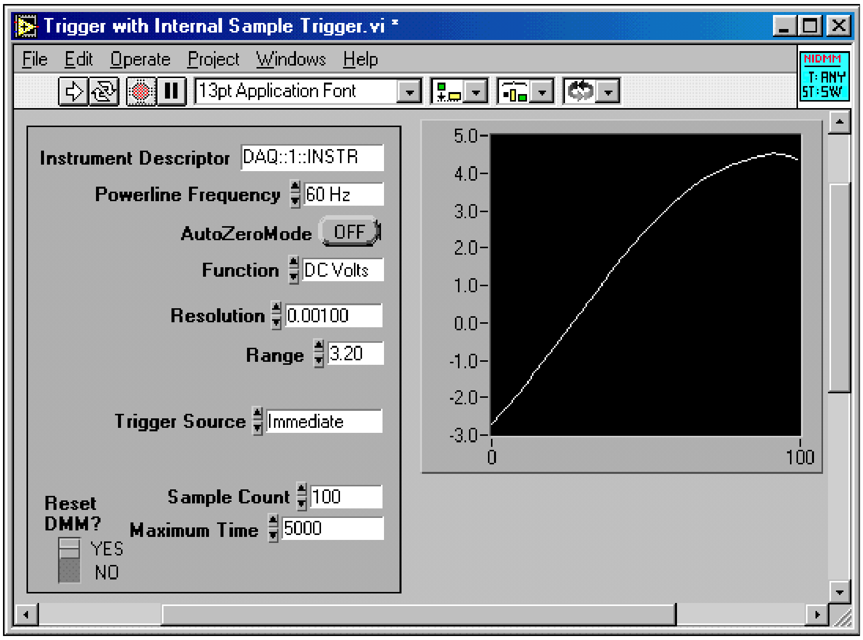
The NI-DMM instrument driver is implementing the following types of `Sample Trigger` sources:

- Immediate (or internal)
- External
  - Applied on a PXI TTL line (0-6)
- Interval
- Software

If `Sample Trigger` is set to Interval, the DMM is configured for interval scanning using the sample interval control (real number in seconds) as scan interval. If `Sample Trigger` is set to software, the DMM is configured to acquire one sample every time a `Send Software Trigger` call is issued.

The following permutations of sources for `Trigger/Sample Trigger` pulses apply:

- Immediate `Trigger`, immediate `Sample Trigger`
- External `Trigger`, immediate `Sample Trigger`


Trigger with Internal Sample Trigger.vi *
File Edit Operate Project Windows Help
13pt Application Font
NIDMM
T: ANY
ST: SW
5.0
4.0
3.0
2.0
1.0
0.0
-1.0
-2.0
-3.0
0
100

Instrument Descriptor: DAQ::1::INSTR
Powerline Frequency: 60 Hz
AutoZeroMode: OFF
Function: DC Volts
Resolution: 0.00100
Range: 3.20
Trigger Source: Immediate
Sample Count: 100
Maximum Time: 5000
Reset DMM? YES NO

Figure 4-8. Trigger with Internal Sample Trigger Front Panel

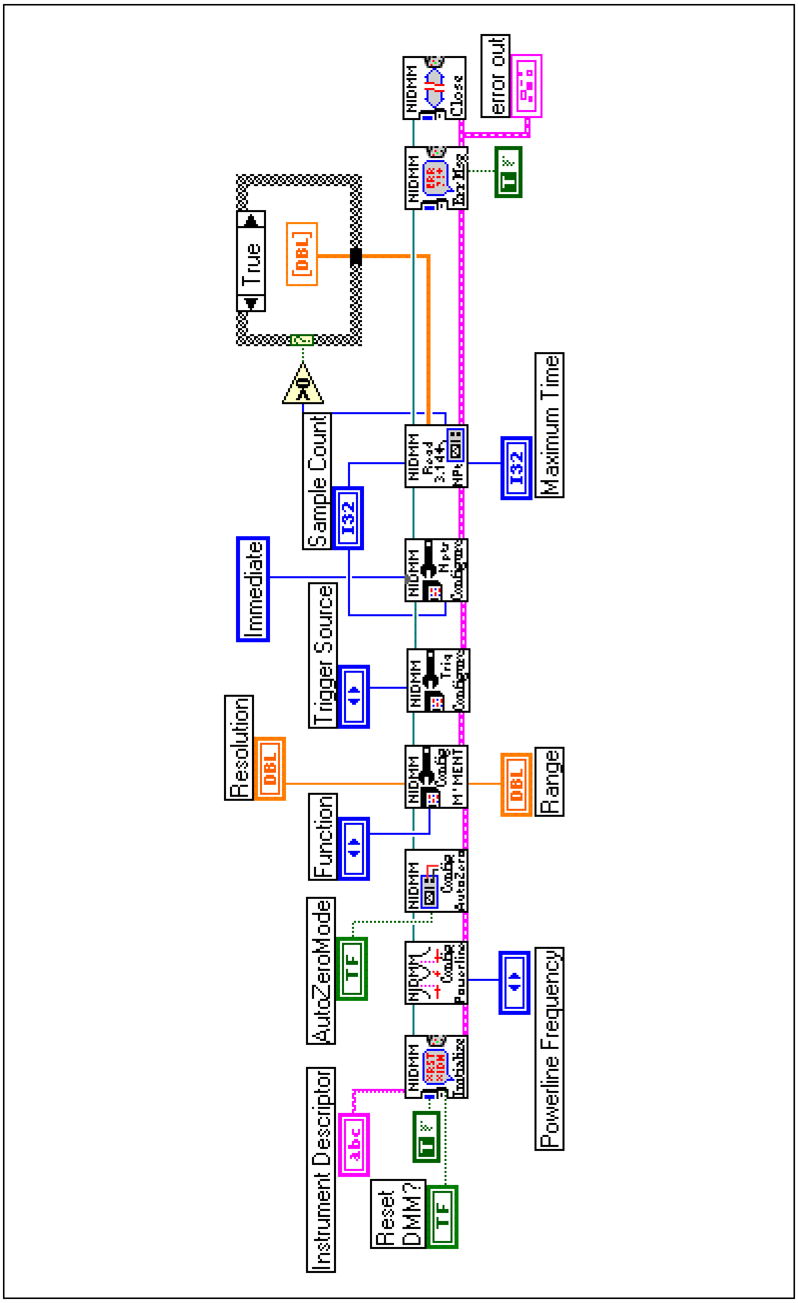


Figure 4-9. Trigger with Internal Sample Trigger Block Diagram

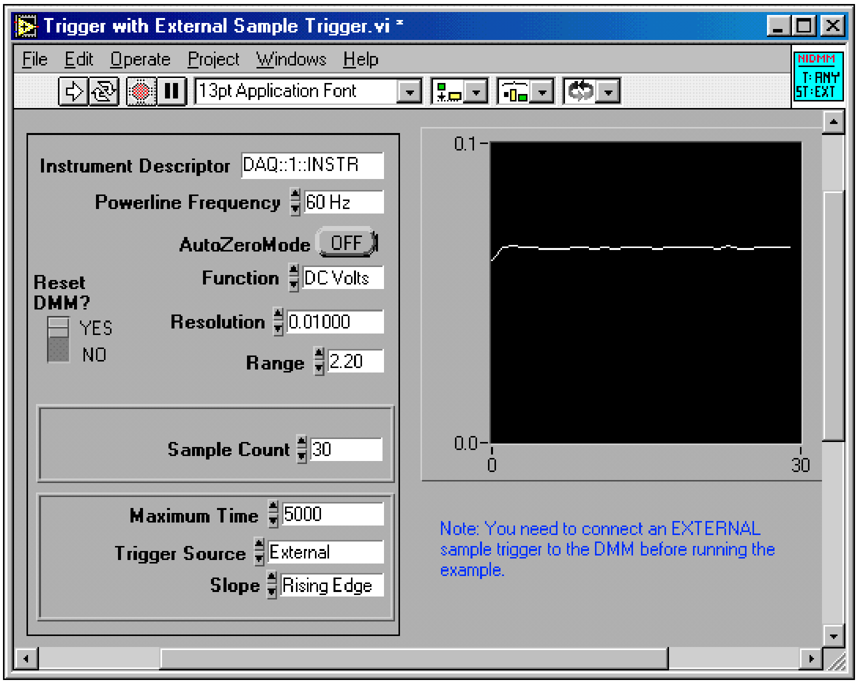
```
void main(void)
   {
   ViStatus  error            = VI_SUCCESS;
   ViSession vi               = VI_NULL;
   ViString  resourceName     = "DAQ::1::INSTR";
   ViBoolean idQuery          = VI_TRUE;
   ViBoolean reset            = VI_TRUE;
   ViReal64  powerlineFreq    = NIDMM_VAL_60_HERTZ;
   ViInt32   function         = NIDMM_VAL_DC_VOLTS;
   ViReal64  range            = 25.00;
   ViInt32   TrigSource       = NIDMM_VAL_IMMEDIATE ;
   ViInt32   TrigSlope        = NIDMM_VAL_POS;
   ViInt32   SampleTrigSource = NIDMM_VAL_IMMEDIATE;
   ViInt32   SampleTrigSlope  = NIDMM_VAL_POS;
   ViInt32   numOfMeas        = 10;
   ViReal64  measurements[10];
   ViReal64  resolution       = 0.001;
   ViInt32   i, timeLimit;
   ViInt32   numPointsRead;

   checkErr(niDMM_init(resourceName, idQuery, reset, &vi));
     /*- Calculate time limit--------------------------------------------*/
   timeLimit = 3000;

   /*- Configure Powerline frequency--------------------------------------*/
   checkErr(niDMM_ConfigurePowerlineFrequency(vi, powerlineFreq));

   /*- Call NIDMM_ConfigureMeasurement() to set function, range and res---*/
   checkErr(niDMM_ConfigureMeasurement(vi, function, range, resolution));

   /*- Set Trigger Source and Trigger Delay-------------------------------*/
   checkErr(niDMM_ConfigureTrigger (vi, TrigSource, 0.0));
   /*- Set Trigger Slope--------------------------------------------------*/
   checkErr(niDMM_ConfigureTriggerSlope (vi, TrigSlope));

   /*- Configure a multipoint acquisition---------------------------------*/
   checkErr(niDMM_ConfigureMultiPoint(vi, 1, numOfMeas, SampleTrigSource,
                    0.0));
   /*- Set Sample Trigger Slope-------------------------------------------*/
   checkErr(niDMM_ConfigureSampleTriggerSlope (vi, SampleTrigSlope));

   /*- Read measurements using NIDMM_ReadMultiPoint()---------------------*/
   checkErr(niDMM_ReadMultiPoint(vi, timeLimit, numOfMeas, measurements,
                    &numPointsRead));
   for (i=0;i<10;i++)
                    printf("measurements[%d] = %f \n", i, measurements[i]);
   printf("\n\n");
Error:
if (vi)
   niDMM_close(vi);
if (error < VI_SUCCESS)
   messageHandler(error);
}
```

Use the `niDMM Configure Trigger` and `niDMM Configure Trigger Slope` functions to implement a trigger. Specify the trigger source delay and slope. If you set **Trigger Source** to External on the panel, then you need to connect a TTL signal to the EXT TRIG IN line. When the NI-DMM receives this pulse, it starts to take **Sample Trigger Count** number of measurements.

If the trigger pulse does not arrive during the next time limit number of milliseconds, the error message **`“Max Time exceeded before the operation completed”`** appears.

Other permutations of sources for Trigger/Sample Trigger pulses are as follows:

- External Trigger, External Sample Trigger
- Internal Trigger, External Sample Trigger


Trigger with External Sample Trigger.vi *

Instrument Descriptor: DAQ::1::INSTR
Powerline Frequency: 60 Hz
AutoZeroMode: OFF
Function: DC Volts
Reset DMM?: YES / NO
Resolution: 0.01000
Range: 2.20
Sample Count: 30
Maximum Time: 5000
Trigger Source: External
Slope: Rising Edge

Note: You need to connect an EXTERNAL sample trigger to the DMM before running the example.

Figure 4-10. Trigger with External Sample Trigger Front Panel

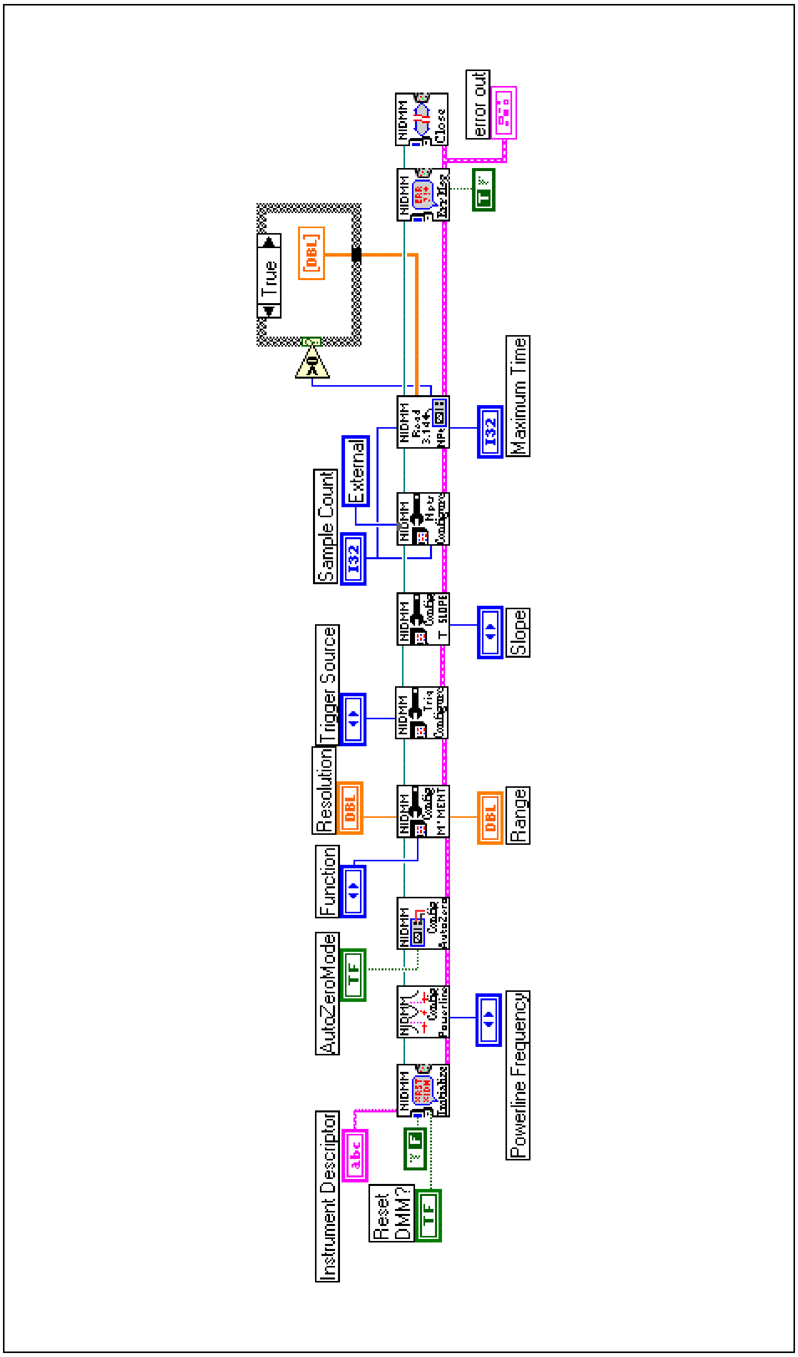


Figure 4-11. Trigger with External Sample Trigger Block Diagram

```
void main(void)
   {
   ViStatus  error             = VI_SUCCESS;
   ViSession vi                = VI_NULL;
   ViString  resourceName      = "DAQ::1::INSTR";
   ViBoolean idQuery           = VI_TRUE;
   ViBoolean reset             = VI_TRUE;
   ViReal64  powerlineFreq     = NIDMM_VAL_60_HERTZ;
   ViInt32   function          = NIDMM_VAL_DC_VOLTS;
   ViReal64  range             = 25.00;
   ViInt32   TrigSource        = NIDMM_VAL_EXTERNAL ;
   ViInt32   TrigSlope         = NIDMM_VAL_POS;
   ViInt32   SampleTrigSource  = NIDMM_VAL_EXTERNAL;
   ViInt32   SampleTrigSlope   = NIDMM_VAL_POS;
   ViInt32   numOfMeas         = 10;
   ViReal64  measurements[10];
   ViReal64  resolution        = 0.001;
   ViInt32   i, timeLimit;
   ViInt32   numPointsRead;

   checkErr(niDMM_init(resourceName, idQuery, reset, &vi));
     /*- Calculate time limit--------------------------------------------*/
   timeLimit = 3000;

   /*- Configure Powerline frequency-------------------------------------*/
   checkErr(niDMM_ConfigurePowerlineFrequency(vi, powerlineFreq));

   /*- Call NIDMM_ConfigureMeasurement() to set function, range and res---*/
   checkErr(niDMM_ConfigureMeasurement(vi, function, range, resolution));

   /*- Set Trigger Source and Trigger Delay-------------------------------*/
   checkErr(niDMM_ConfigureTrigger (vi, TrigSource, 0.0));
   /*- Set Trigger Slope--------------------------------------------------*/
   checkErr(niDMM_ConfigureTriggerSlope (vi, TrigSlope));

   /*- Configure a multipoint acquisition---------------------------------*/
   checkErr(niDMM_ConfigureMultiPoint(vi, 1, numOfMeas, SampleTrigSource,
                    0.0));
   /*- Set Sample Trigger Slope-------------------------------------------*/
   checkErr(niDMM_ConfigureSampleTriggerSlope (vi, SampleTrigSlope));

   /*- Read measurements using NIDMM_ReadMultiPoint()---------------------*/
   checkErr(niDMM_ReadMultiPoint(vi, timeLimit, numOfMeas, measurements,
                    &numPointsRead));
   for (i=0;i<10;i++)
                    printf("measurements[%d] = %f \n", i, measurements[i]);
   printf("\n\n");
Error:
if (vi)
   niDMM_close(vi);
if (error < VI_SUCCESS)
   messageHandler(error);
}
```

Use the `niDMM ConfigureMultipoint` function to implement a Sample Trigger Pulse. This function configures the sample DMM trigger attributes. These attributes include the trigger source and sample interval. If you set **Trigger Source** to External on the panel, then you need to connect a TTL signal to the EXT TRIG IN line.

If you are configured for an external trigger and an external sample trigger, the moment the DMM receives the first pulse the DMM starts to take sample count number of measurements, one every time a sample trigger pulse arrives.

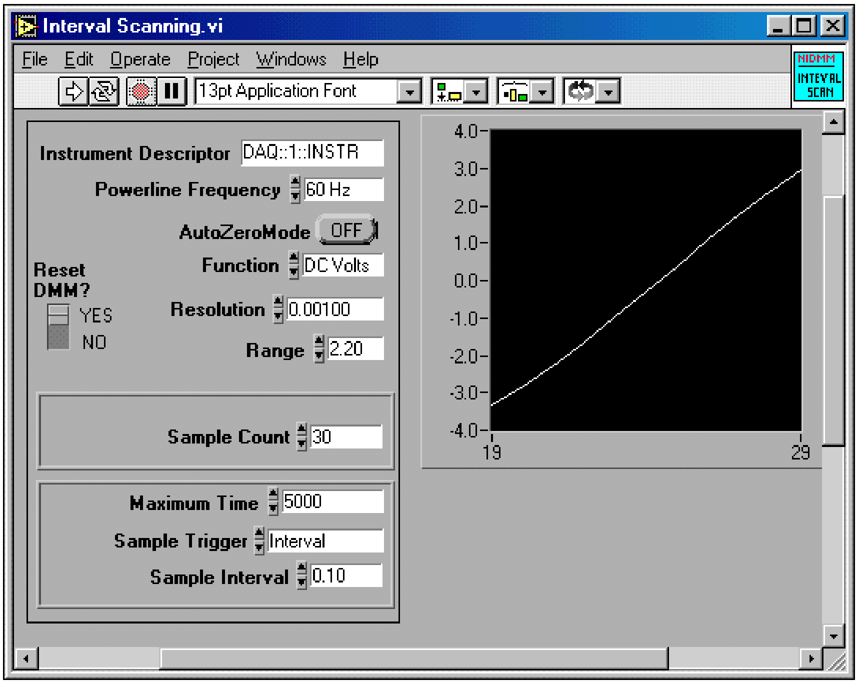
If you are configured for an immediate trigger and an external sample trigger, at `niDMM Read` function execution time, the DMM starts making sample count number of measurements, one every time a sample trigger pulse arrives. Refer to your DMM's User Manual for more information about triggering.

Interval Scanning

Interval scanning is a scanning operation in which you have control over the sample interval. In order to configure the DMM for interval scanning, use `niDMM Configure MultiPoint`, and set:

- Sample Trigger to interval
- Sample Interval to the real number in seconds representing the desired scan interval

Note If you set Sample Trigger to software, the DMM will acquire one data point every time you are calling `niDMM Send Software Trigger`.



Instrument Descriptor DAQ::1::INSTR
Powerline Frequency 60 Hz
AutoZeroMode OFF
Function DC Volts
Reset DMM? YES NO
Resolution 0.00100
Range 2.20
Sample Count 30
Maximum Time 5000
Sample Trigger Interval
Sample Interval 0.10

Figure 4-12. Interval Scanning Front Panel

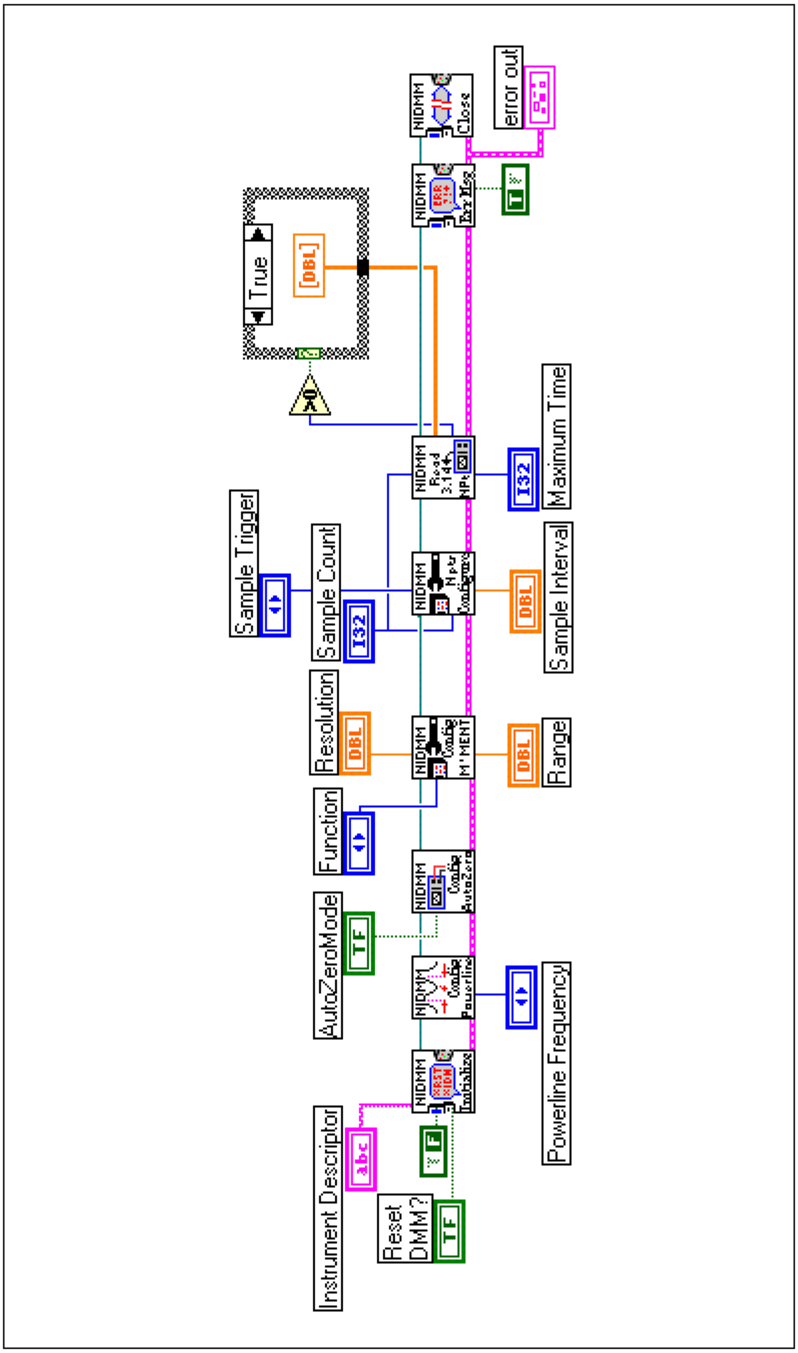


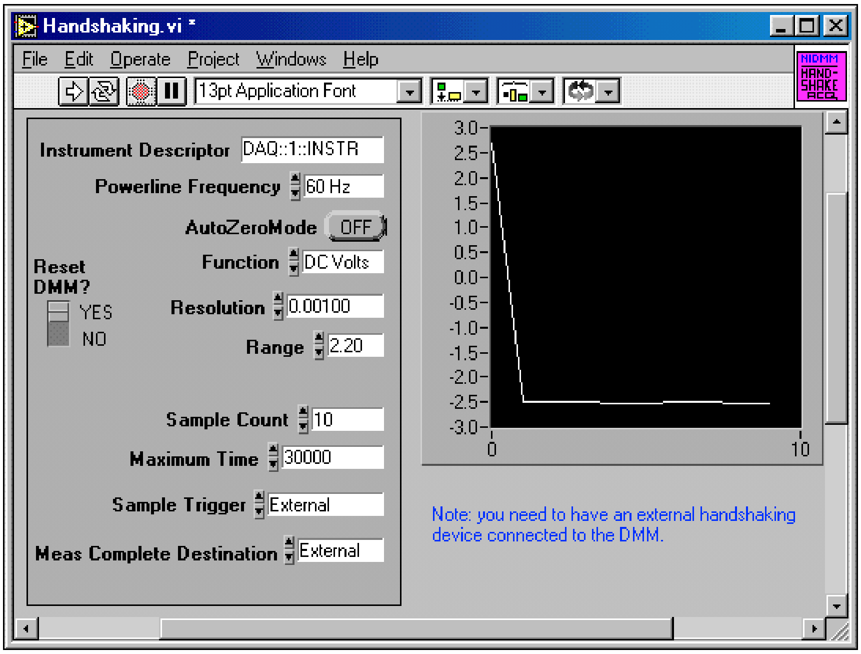
Figure 4-13. Interval Scanning Block Diagram

```
void main(void)
   {
   ViStatus  error              = VI_SUCCESS;
   ViSession vi                 = VI_NULL;
   ViString  resourceName       = "DAQ::1::INSTR";
   ViBoolean idQuery            = VI_TRUE;
   ViBoolean reset              = VI_TRUE;
   ViReal64  powerlineFreq      = NIDMM_VAL_60_HERTZ;
   ViInt32   function           = NIDMM_VAL_DC_VOLTS;
   ViReal64  range              = 25.00;
   ViInt32   TrigSource         = NIDMM_VAL_IMMEDIATE ;
   ViInt32   TrigSlope          = NIDMM_VAL_POS;
   ViInt32   SampleTrigSource   = NIDMM_VAL_INTERVAL;
   ViInt32   numOfMeas          = 10;
   ViReal64  measurements[10];
   ViReal64  resolution         = 0.001;
   ViInt32   i, timeLimit;
   ViInt32   numPointsRead;
   ViReal64  sampleInterval     = 0.1;
   checkErr(niDMM_init(resourceName, idQuery, reset, &vi));
     /*- Calculate time limit--------------------------------------------*/
   timeLimit = 3000;
   /*- Configure Powerline frequency-------------------------------------*/
   checkErr(niDMM_ConfigurePowerlineFrequency(vi, powerlineFreq));
   /*- Call NIDMM_ConfigureMeasurement() to set function, range and res---*/
   checkErr(niDMM_ConfigureMeasurement(vi, function, range, resolution));
   /*- Set Trigger Source and Trigger Delay-------------------------------*/
   checkErr(niDMM_ConfigureTrigger (vi, TrigSource, 0.0));
   /*- Set Trigger Slope--------------------------------------------------*/
   checkErr(niDMM_ConfigureTriggerSlope (vi, TrigSlope));
   /*- Configure a multipoint acquisition, Sample Trigger and sample
                    Interval-*/
   checkErr(niDMM_ConfigureMultiPoint(vi, 1, numOfMeas, SampleTrigSource,
                    sampleInterval));
   /*- Read measurements using NIDMM_ReadMultiPoint()---------------------*/
   checkErr(niDMM_ReadMultiPoint(vi, timeLimit, numOfMeas, measurements,
                    &numPointsRead));
   for (i=0;i<10;i++)
                    printf("measurements[%d] = %f \n", i, measurements[i]);
   printf("\n\n");
Error:
if (vi)
   niDMM_close(vi);
if (error < VI_SUCCESS)
   messageHandler(error);
}
```

Handshaking Mode

In handshaking mode, the DMM generates a digital pulse on the voltmeter complete (VMC) line when it finishes taking a measurement. If a multiplexer is attached to the DMM, this pulse can be used to advance the multiplexer to the next channel. When the analog circuitry of the multiplexer has settled, the multiplexer generates a digital pulse on the EXT TRIG IN line to the DMM. When the DMM receives this pulse, it takes another measurement.

Handshaking mode can be considered as a particular case of the External Trigger Pulse example, where the trigger comes from a multiplexer and the voltmeter complete (VMC) pulse, which is generated by the NI-DMM hardware, goes to the multiplexer. Use the `niDMM Configure Measurement Complete` function to implement the emission of a digital pulse on the VMC line at the end of each measurement. This function configures the DMM Measurement Complete attributes. These attributes include the destination and slope. Set **Measurement Complete Destination** to External on the application panel for activation of the VMC line.



Instrument Descriptor DAQ::1::INSTR
Powerline Frequency 60 Hz
AutoZeroMode OFF
Function DC Volts
Reset DMM? YES NO
Resolution 0.00100
Range 2.20
Sample Count 10
Maximum Time 30000
Sample Trigger External
Meas Complete Destination External

Note: you need to have an external handshaking device connected to the DMM.

Figure 4-14. Handshaking Front Panel

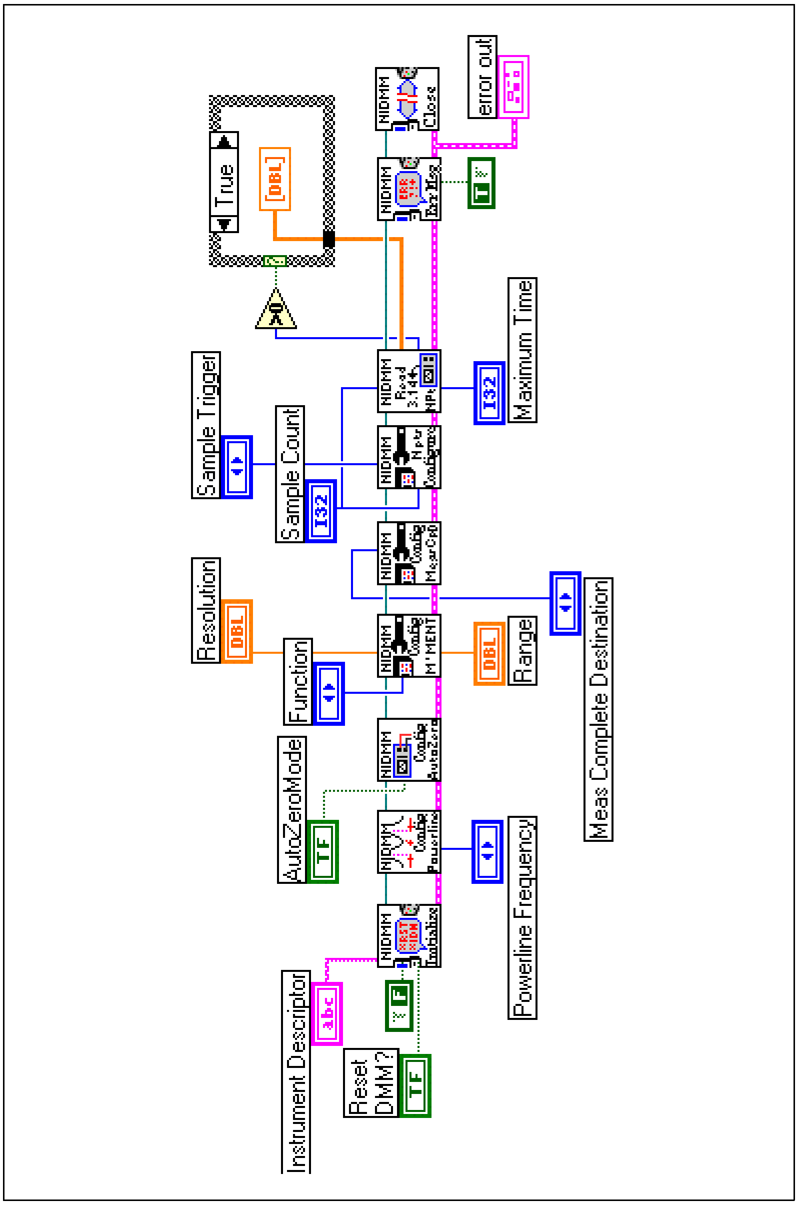


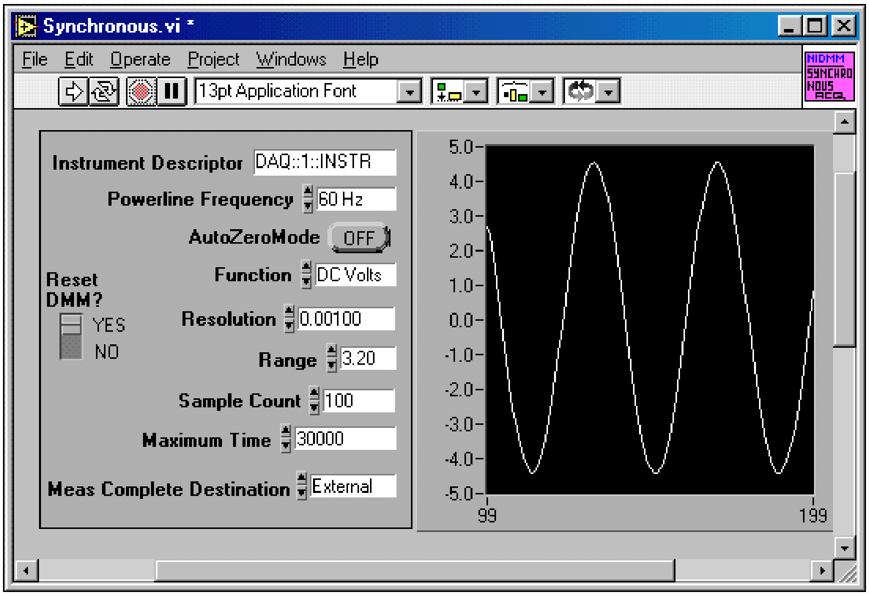
Figure 4-15. Handshaking Block Diagram

```
void main(void)
   {
   ViStatus  error           = VI_SUCCESS;
   ViSession vi              = VI_NULL;
   ViString  resourceName    = "DAQ::1::INSTR";
   ViBoolean idQuery         = VI_TRUE;
   ViBoolean reset           = VI_TRUE;
   ViReal64  powerlineFreq   = NIDMM_VAL_60_HERTZ;
   ViInt32   function        = NIDMM_VAL_DC_VOLTS;
   ViReal64  range           = 25.00;
   ViInt32   numOfMeas       = 10;
   ViReal64  measurements[10];
   ViReal64  resolution      = 0.001;
   ViInt32   timeLimit;
   ViInt32   i, numPointsRead;

   checkErr(niDMM_init(resourceName, idQuery, reset, &vi));
     /*- Calculate time limit--------------------------------------------*/
   timeLimit = 5000;

   /*- Configure Powerline frequency--------------------------------------*/
   checkErr(niDMM_ConfigurePowerlineFrequency(vi, powerlineFreq));

   /*- Call NIDMM_ConfigureMeasurement() to set function, range and res---*/
   checkErr(niDMM_ConfigureMeasurement(vi, function, range, resolution));

   /*- Configure measurement complete destination to EXTERNAL-------------*/
   checkErr(niDMM_ConfigureMeasurementComplete (vi, NIDMM_VAL_EXTERNAL,
                    NIDMM_VAL_NEG));

   /*- Configure a multipoint acquisition---------------------------------*/
   checkErr(niDMM_ConfigureMultiPoint (vi, 1, numOfMeas, NIDMM_VAL_EXTERNAL,
                    0.0));

   /*- Read measurements using NIDMM_ReadMultiPoint()---------------------*/
   checkErr(niDMM_ReadMultiPoint(vi, timeLimit, numOfMeas, measurements,
                    &numPointsRead));

   for (i=0;i<10;i++)
   printf("measurements[%d] = %f \n", i, measurements[i]);
   printf("\n\n");

   Error:
   if (vi)
      niDMM_close(vi);

   if (error < VI_SUCCESS)
      messageHandler(error);
   }
```

Synchronous Mode

In synchronous mode, the DMM emits a digital pulse on the VMC line when it finishes taking a measurement. If a multiplexer is attached to the DMM, this pulse causes the multiplexer to advance to the next channel. After a programmable delay has occurred, the DMM takes another measurement without requiring an external trigger. Synchronous mode can be considered as a particular case of the Internal Trigger Pulse example, where the trigger is issued internally by the hardware after a programmed delay.

Use the `niDMM Configure Trigger` function to implement the delay of the **Internal Trigger Pulse**. Set **Trigger Delay** to the desired value in seconds on the application panel. Use the `niDMM Configure Measurement Complete` function to implement the emission of a digital pulse on the VMC line at the end of each measurement. This function configures the DMM Measurement Complete attributes. These attributes include the destination and slope. Set **Measurement Complete Destination** to External on the application panel for activation of the VMC line, located on the front connector. Use TTL0–TTL7 to send VMC over PXI trigger lines.



Instrument Descriptor: DAQ::1::INSTR
Powerline Frequency: 60 Hz
AutoZeroMode: OFF
Function: DC Volts
Reset DMM?: YES / NO
Resolution: 0.00100
Range: 3.20
Sample Count: 100
Maximum Time: 30000
Meas Complete Destination: External

Figure 4-16. Synchronous Triggering Front Panel

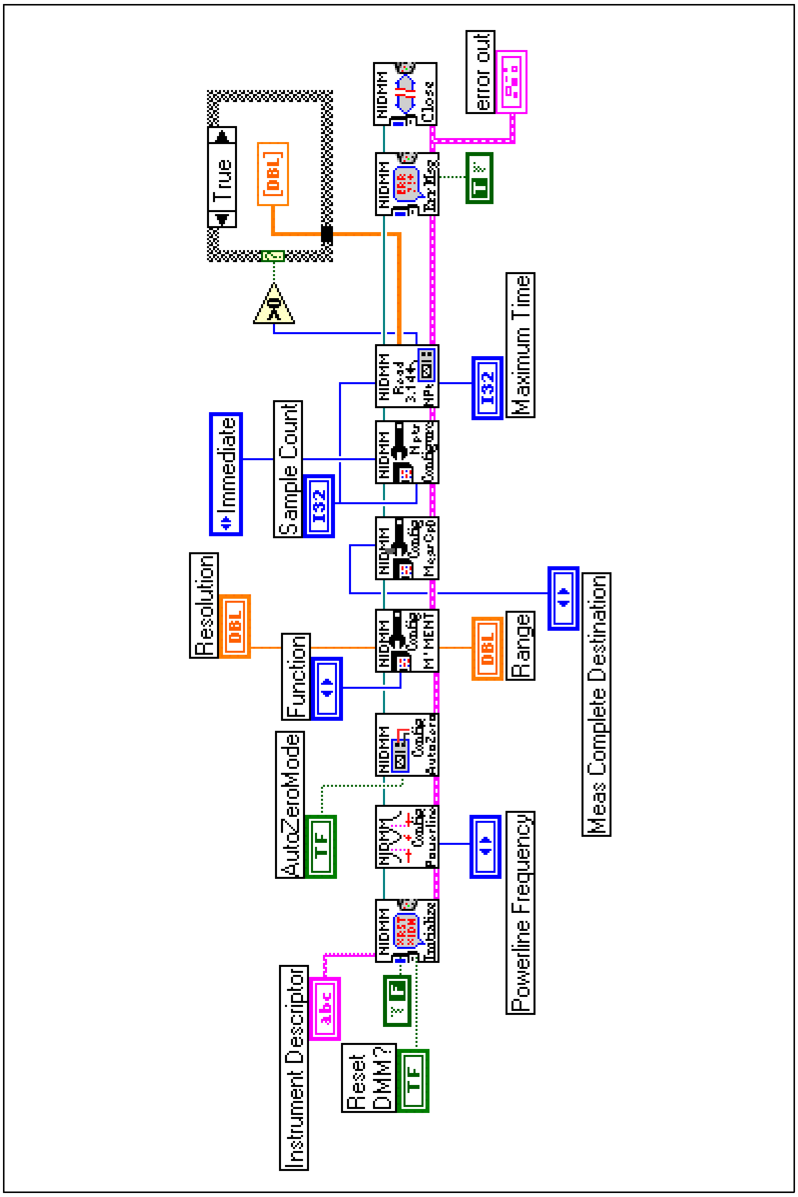


Figure 4-17. Synchronous Triggering Block Diagram

```
void main(void)
   {
   ViStatus  error           = VI_SUCCESS;
   ViSession vi              = VI_NULL;
   ViString  resourceName    = "DAQ::1::INSTR";
   ViBoolean idQuery         = VI_TRUE;
   ViBoolean reset           = VI_TRUE;
   ViReal64  powerlineFreq   = NIDMM_VAL_60_HERTZ;
   ViInt32   function        = NIDMM_VAL_DC_VOLTS;
   ViReal64  range           = 25.00;
   ViInt32   numOfMeas       = 10;
   ViReal64  measurements[10];
   ViReal64  resolution      = 0.001;
   ViInt32   timeLimit;
   ViInt32   i, numPointsRead;

   checkErr(niDMM_init(resourceName, idQuery, reset, &vi));
     /*- Calculate time limit -------------------------------------------*/
   timeLimit = 5000;

   /*- Configure Powerline frequency-------------------------------------*/
   checkErr(niDMM_ConfigurePowerlineFrequency(vi, powerlineFreq));

   /*- Call NIDMM_ConfigureMeasurement() to set function, range and res---*/
   checkErr(niDMM_ConfigureMeasurement(vi, function, range, resolution));

   /*- Configure measurement complete destination to EXTERNAL-------------*/
   checkErr(niDMM_ConfigureMeasCompleteDest (vi, NIDMM_VAL_EXTERNAL));

   /*- Configure a multipoint acquisition---------------------------------*/
   checkErr(niDMM_ConfigureMultiPoint (vi, 1, numOfMeas,
                    NIDMM_VAL_IMMEDIATE, 0.0));

   /*- Read measurements using NIDMM_ReadMultiPoint()---------------------*/
   checkErr(niDMM_ReadMultiPoint(vi, timeLimit, numOfMeas, measurements,
                    &numPointsRead));

   for (i=0;i<10;i++)
   printf("measurements[%d] = %f \n", i, measurements[i]);
   printf("\n\n");

   Error:
   if (vi)
      niDMM_close(vi);

   if (error < VI_SUCCESS)
      messageHandler(error);
   }
```

Scanning an SCXI Multiplexer Module Using the NI-DMM

You can use the NI-DMM products with National Instruments multiplexers as well as third-party multiplexers that use traditional voltmeter signaling.

Traditional voltmeter signaling is a handshake protocol in which the voltmeter starts a measurement when a trigger pulse occurs on its EXT TRIG IN line, and emits a digital pulse on the voltmeter complete VMC line when it finishes taking a measurement.

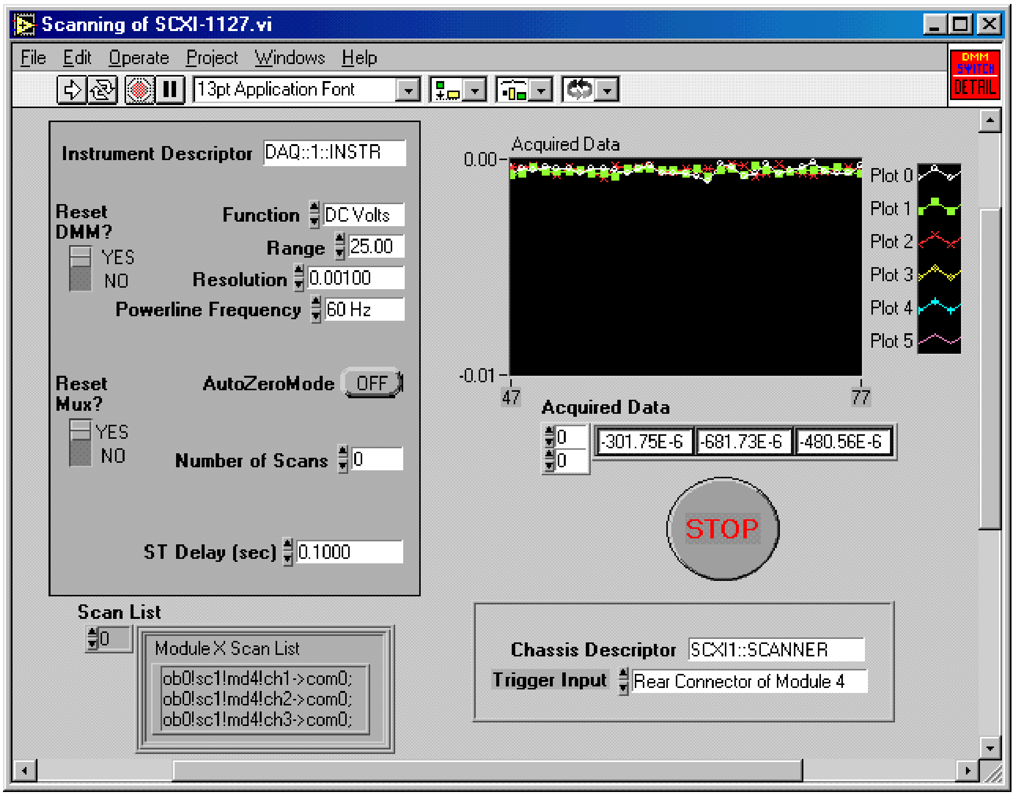
The following example shows, in LabVIEW, how to scan a measurement using the the NI-DMM and NI-SWITCH instrument drivers with devices such as the NI 4060 board with the SCXI-1127 switch module. You can find this example on the NI-SWITCH and the NI-DMM CD under the `Example` directory found in the root directory.

Note NI-SWITCH is the National Instruments switch driver, please refer to the *NI-SWITCH Software User Manual* for information regarding these functions.

Perform the following steps to scan a measurement:

1. Initialize and configure the switch. Configuration of the switch module SCXI-1127 is implemented by using NI-SWITCH functions.
   a. Set **Chassis Descriptor** to SCXI1::4::SCANNER.
   b. Set **Scan List** (ex: ob0!sc1!md4!ch1->com0; ob0!sc1!md4!ch2->com0. This example will scan channels 1 and 2 on the module in slot 4).
   c. Set **Reset MUX** to YES.
   d. Set **Trigger Input** to Rear Connector of Module 4.
   e. Set **Scan Advanced Output** to NONE.
   f. Configure the scan list using `niSwitch Configure Scan List`.
2. Initialize and configure the DMM. This acquisition is of Continuous type, which was previously discussed.
   a. Set the Function, Resolution, AutoZero, Powerline Frequency, and Range attributes of the DMM hardware.
   b. Set the delay and get the measurement period value.
3. Initiate the switch.
4. Initiate the DMM.

5. Fetch and demultiplex the acquired data inside the loop.
6. Abort and close the DMM.
7. Abort and close the switch.



Instrument Descriptor DAQ::1::INSTR
Reset DMM? YES NO
Function DC Volts
Range 25.00
Resolution 0.00100
Powerline Frequency 60 Hz
Reset Mux? YES NO
AutoZeroMode OFF
Number of Scans 0
ST Delay (sec) 0.1000
Scan List 0
ob0!sc1!md4!ch1->com0;
ob0!sc1!md4!ch2->com0;
ob0!sc1!md4!ch3->com0;
Acquired Data 0 0 -301.75E-6 -681.73E-6 -480.56E-6
Chassis Descriptor SCXI1::SCANNER
Trigger Input Rear Connector of Module 4

Figure 4-18. Scanning of SCXI Front Panel

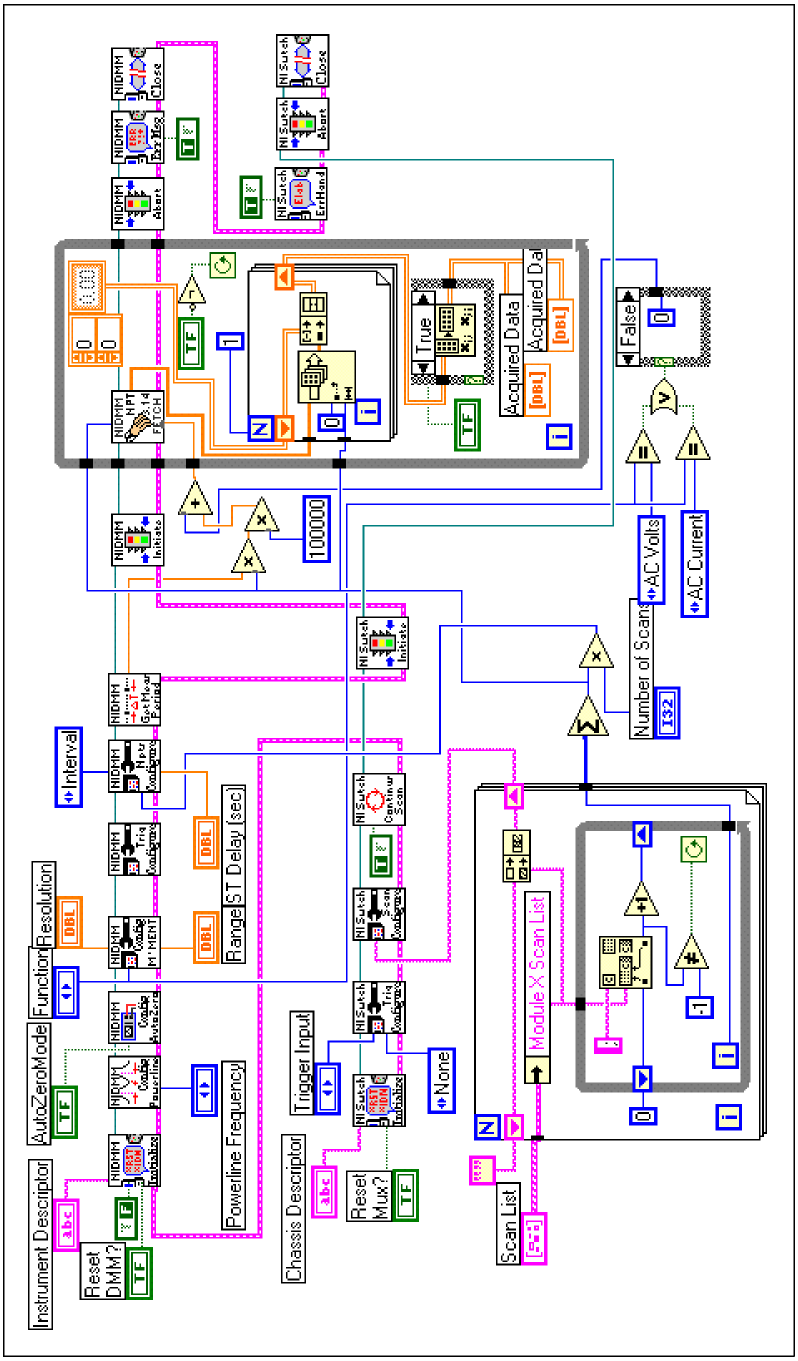


Figure 4-19. Scanning of SCXI Block Diagram

```
void main(void)
   {
   ViStatus  error           = VI_SUCCESS;
   ViSession vi              = VI_NULL;
   ViSession instr           = VI_NULL;
   ViString  resourceName    = "DAQ::1::INSTR";
   ViString  scxichan        = "sc1!md4!ch26->com0;";
   ViBoolean idQuery         = VI_TRUE;
   ViBoolean reset           = VI_TRUE;
   ViReal64  powerlineFreq   = NIDMM_VAL_60_HERTZ;
   ViInt32   function        = NIDMM_VAL_DC_VOLTS;
   ViReal64  range           = 25.00;
   ViInt32   numOfMeas       = 10;
   ViReal64  measurements[10];
   ViReal64  resolution      = 0.001;
   ViInt32   timeLimit;
   ViReal64  delay           = 0.5;
   ViReal64  sampleInterval  = 0.1;
   ViReal64  period;
   ViInt32   numpts,i;

   /*- NI-SWITCH configurations-------------------------------------------*/
   checkErr(niSwitch_init ("SCXI1::SCANNER", VI_TRUE,
                  VI_TRUE, &instr));
   checkErr(niSwitch_ConfigureScanTrigger (instr, 0.0,
                  NISWITCH_VAL_REARCONNECTOR_MODULE4,
                  NISWITCH_VAL_NONE));
   checkErr(niSwitch_ConfigureScanList (instr, scxichan,
                  NISWITCH_VAL_BREAK_BEFORE_MAKE));
   checkErr(niSwitch_SetContinuousScan (instr, 1));

   checkErr(niDMM_init(resourceName, idQuery, reset, &vi));

   /*- Calculate time limit-----------------------------------------------*/
   timeLimit = 30000;

   /*- Configure Powerline frequency--------------------------------------*/
   checkErr(niDMM_ConfigurePowerlineFrequency(vi, powerlineFreq));

   /*- Call NIDMM_ConfigureMeasurement() to set function, range and res---*/
   checkErr(niDMM_ConfigureMeasurement(vi, function, range, resolution));

   /*- Call NIDMM_Configure Trigger() to set trigger delay----------------*/
   checkErr(niDMM_ConfigureTrigger (vi, NIDMM_VAL_IMMEDIATE, delay));

   /*- Configure a multipoint acquisition, set sample interval------------*/
   checkErr(niDMM_ConfigureMultiPoint(vi, 1, numOfMeas, NIDMM_VAL_INTERVAL,
                  sampleInterval));
   /*- Get period measurement---------------------------------------------*/
   checkErr(niDMM_GetMeasurementPeriod (vi, &period));

   /*- Initiate switch and DMM--------------------------------------------*/
   checkErr(niSwitch_InitiateScan (instr));
   checkErr(niDMM_Initiate (vi));

   /*- Fetch 30 measurementswith DMM--------------------------------------*/
   checkErr(niDMM_FetchMultiPoint (vi, timeLimit, 10, measurements,
                  &numpts));
   for (i=0;i<10;i++)
   printf("measurements[%d] = %f \n", i, measurements[i]);
   printf("\n\n");

  /*- Abort and Close DMM and Switch--------------------------------------*/
   checkErr(niDMM_Abort (vi));
   checkErr(niDMM_close (vi));
   checkErr(niSwitch_AbortScan(instr));
   checkErr(niSwitch_close (instr));

Error:
   if (vi)
      niDMM_close(vi);

   if (error < VI_SUCCESS)
      messageHandler(error);
   }
```

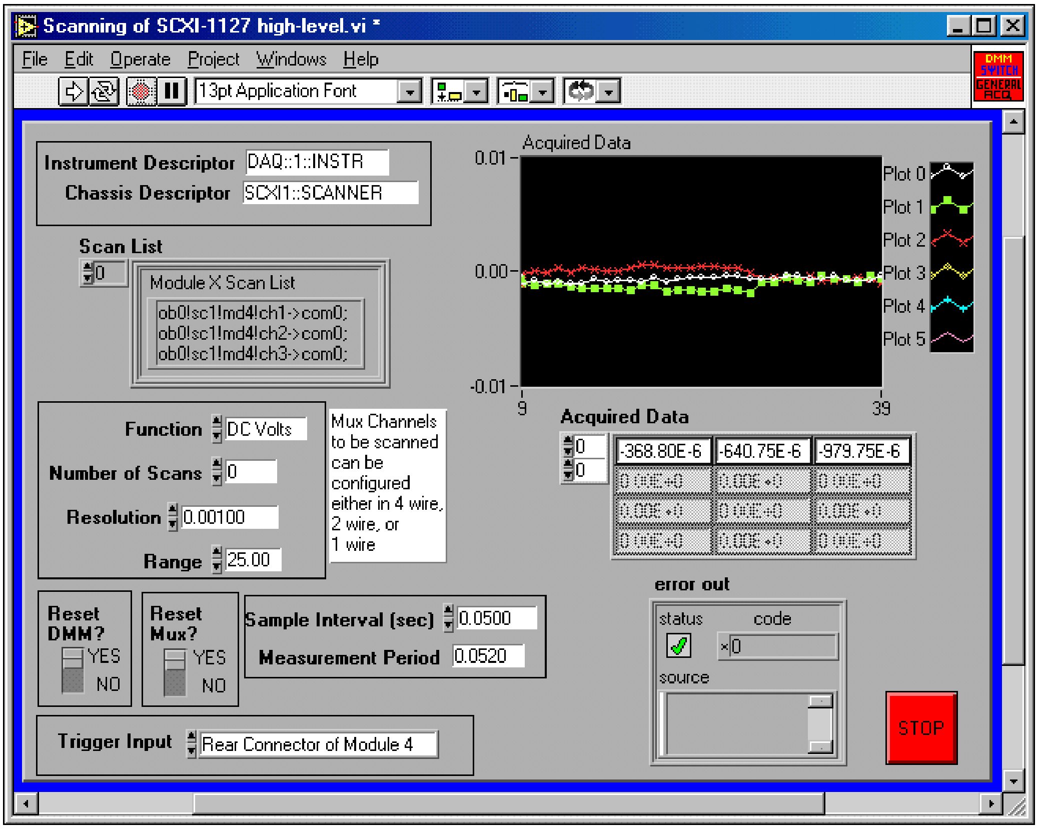
Notice that you did not use `niDMM Configure Measurement Complete` to implement the generation of a digital pulse on the VMC line at the end of each measurement. This is possible because when you configure the SCXI-1127 as connected to NI 4060 in Measurement & Automation Explorer, prior to running the switch application, Measurement & Automation Explorer automatically connects the VMC line of the NI-DMM hardware, via the SCXI backplane, to the SCXI-1127 switch module.

The scanning of the SCXI-1127 example can be simplified by using higher level functions that implement part of the functionality. You can rebuild this example using higher-level functions that configure the switch; configure the NI-DMM; and fetch and demultiplex the readings. The scanning of SCXI-1127 high-level example is a compressed remake, which shows how implementation of higher-level functions make it easier to read an application. The following high-level functions are used in this example:

- Create Channel String
- Initialize and Configure Switch

- Initialize and Configure DMM
- Fetch and Demux Data

Note You can find these example in the NI-SWITCH CD in the `Example` directory, in the `niDMM&Switch Application.llb` file.



Instrument Descriptor DAQ::1::INSTR
Chassis Descriptor SCXI1::SCANNER

Scan List 0
Module X Scan List
```
ob0!sc1!md4!ch1->com0;
ob0!sc1!md4!ch2->com0;
ob0!sc1!md4!ch3->com0;
```

Function DC Volts
Number of Scans 0
Resolution 0.00100
Range 25.00

Mux Channels to be scanned can be configured either in 4 wire, 2 wire, or 1 wire

Acquired Data 0 0

-368.80E-6	-640.75E-6	-979.75E-6
0.00E+0	0.00E+0	0.00E+0
0.00E+0	0.00E+0	0.00E+0
0.00E+0	0.00E+0	0.00E+0

Reset DMM? YES NO
Reset Mux? YES NO

Sample Interval (sec) 0.0500
Measurement Period 0.0520

error out
status ✓ code 0
source

Trigger Input Rear Connector of Module 4

Figure 4-20. Scanning of SCXI High-Level Front Panel

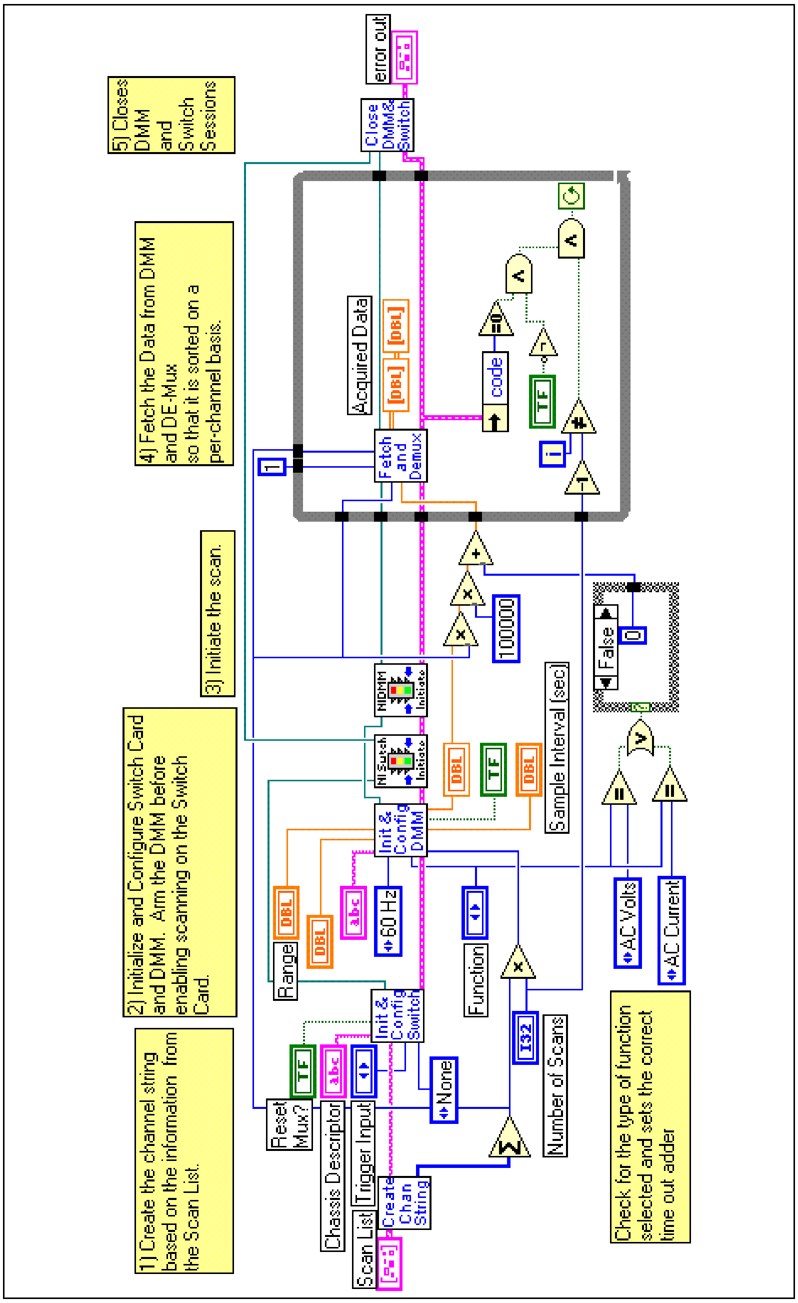
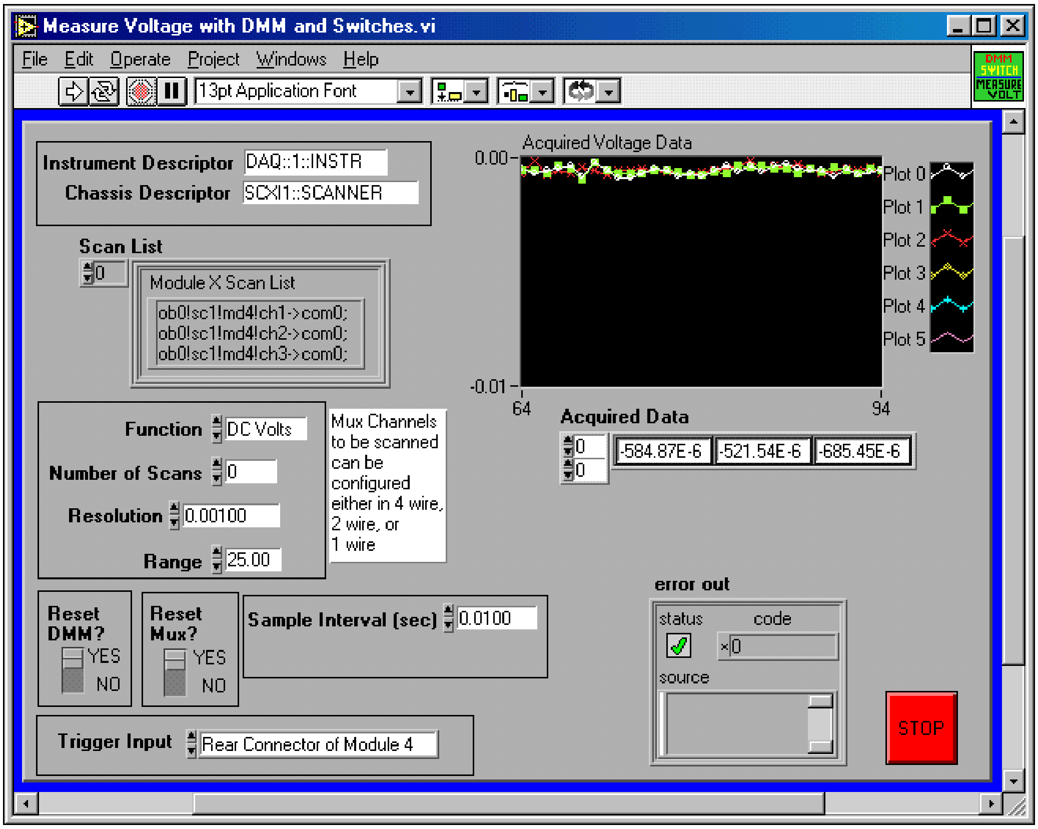

1) Create the channel string based on the information from the Scan List.
2) Initialize and Configure Switch Card and DMM. Arm the DMM before enabling scanning on the Switch Card.
3) Initiate the scan.
4) Fetch the Data from DMM and DE-Mux so that it is sorted on a per-channel basis.
5) Closes DMM and Switch Sessions
Check for the type of function selected and sets the correct time out adder
Scan List
Create Chan String
Chassis Descriptor
Trigger Input
Reset Mux?
None
Init & Config Switch
Range
60 Hz
Init & Config DMM
Function
Number of Scans
I32
NI Switch Initiate
NIDMM Initiate
Sample Interval (sec)
100000
False
0
AC Volts
AC Current
1
Fetch and Demux
Acquired Data
code
Close DMM& Switch
error out

Figure 4-21. Scanning of SCXI High-Level Diagram

We can compress the previous example even further by including all its functionality into three distinct blocks:

- Initialize and initiate the measurement.
- Fetch the data and do the scaling (if necessary).
- Close DMM and switch sessions. Abort scanning (if necessary).

This model is a good tool for measuring voltage and scaling the reading into different measurement units, depending on the sensor type.



Instrument Descriptor: DAQ::1::INSTR
Chassis Descriptor: SCXI1::SCANNER

Scan List: 0
Module X Scan List
ob0!sc1!md4!ch1->com0;
ob0!sc1!md4!ch2->com0;
ob0!sc1!md4!ch3->com0;

Function: DC Volts
Number of Scans: 0
Resolution: 0.00100
Range: 25.00

Mux Channels to be scanned can be configured either in 4 wire, 2 wire, or 1 wire

Acquired Data: 0, 0 | -584.87E-6 | -521.54E-6 | -685.45E-6

Reset DMM? YES / NO
Reset Mux? YES / NO
Sample Interval (sec): 0.0100

error out
status: ✓
code: 0
source:

Trigger Input: Rear Connector of Module 4

Figure 4-22. Measure Voltage with DMM and Switches Front Panel

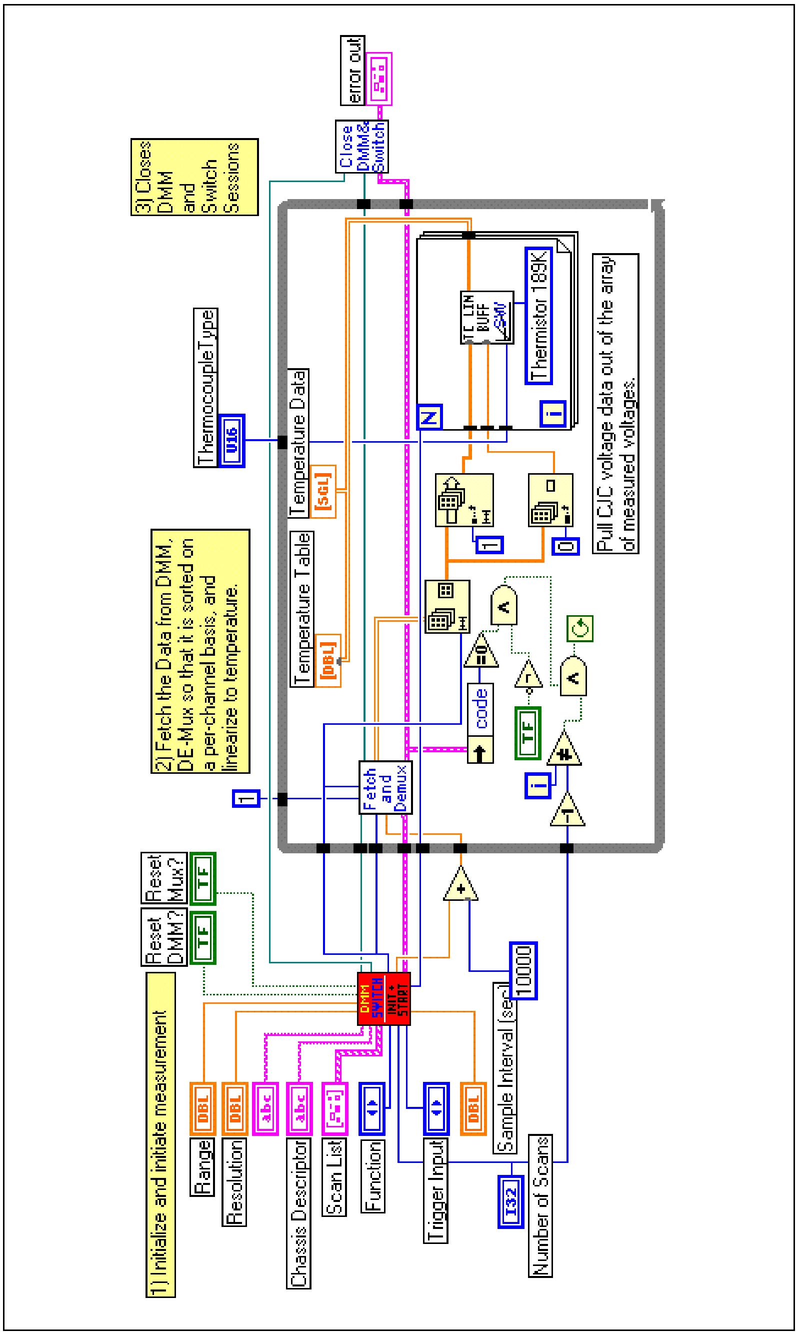


Figure 4-23. Measure Voltage with DMM and Switches Block Diagram

The idea is that once you have the acquisition running, the driver hides its details within high-level functions and shifts the application focus to other tasks like scaling, presentation, storage, transmission of data, and so on.

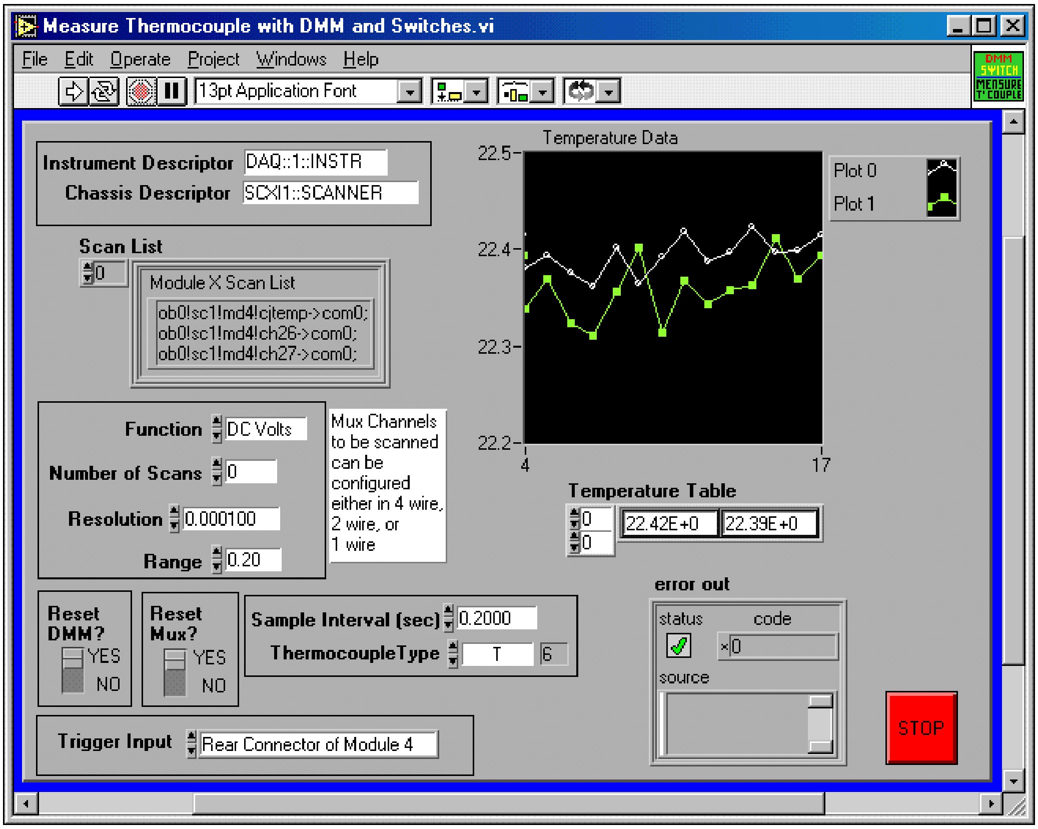
Temperature with Thermocouple Measurement

To implement a thermocouple measurement application using the combination DMM-Switch, complete the following steps:

1. Open the Voltage Measurement example.
2. Inside the fetch and demux loop, implement linearization for the type of thermocouple used. For cold-junction compensation, the SCXI-1331 terminal block has a thermistor with a 189 kΩ reference resistor. We used this method to create the `Measure Thermocouple with DMM and Switches.vi` example.

Note For this implementation to work properly, you need to retrieve the CJS voltage data out of the measured output array. To retrieve the data, complete the following steps:

1. Assign one module only to each element of the scan list.
2. Enter each channel of the module explicitly.
3. Make the CJS channel the first channel in each module channel list. The CJS channel should always be the first channel.


Measure Thermocouple with DMM and Switches.vi

File Edit Operate Project Windows Help

13pt Application Font

Instrument Descriptor DAQ::1::INSTR
Chassis Descriptor SCXI1::SCANNER

Scan List 0

Module X Scan List
ob0!sc1!md4!cjtemp->com0;
ob0!sc1!md4!ch26->com0;
ob0!sc1!md4!ch27->com0;

Function DC Volts
Number of Scans 0
Resolution 0.000100
Range 0.20

Mux Channels to be scanned can be configured either in 4 wire, 2 wire, or 1 wire

Reset DMM? YES NO
Reset Mux? YES NO

Sample Interval (sec) 0.2000
ThermocoupleType T 6

Trigger Input Rear Connector of Module 4

Temperature Data

Temperature Table
0 0 22.42E+0 22.39E+0

error out
status ✓ code 0
source

Figure 4-24. Measure Thermocouple with DMM and Switches Front Panel

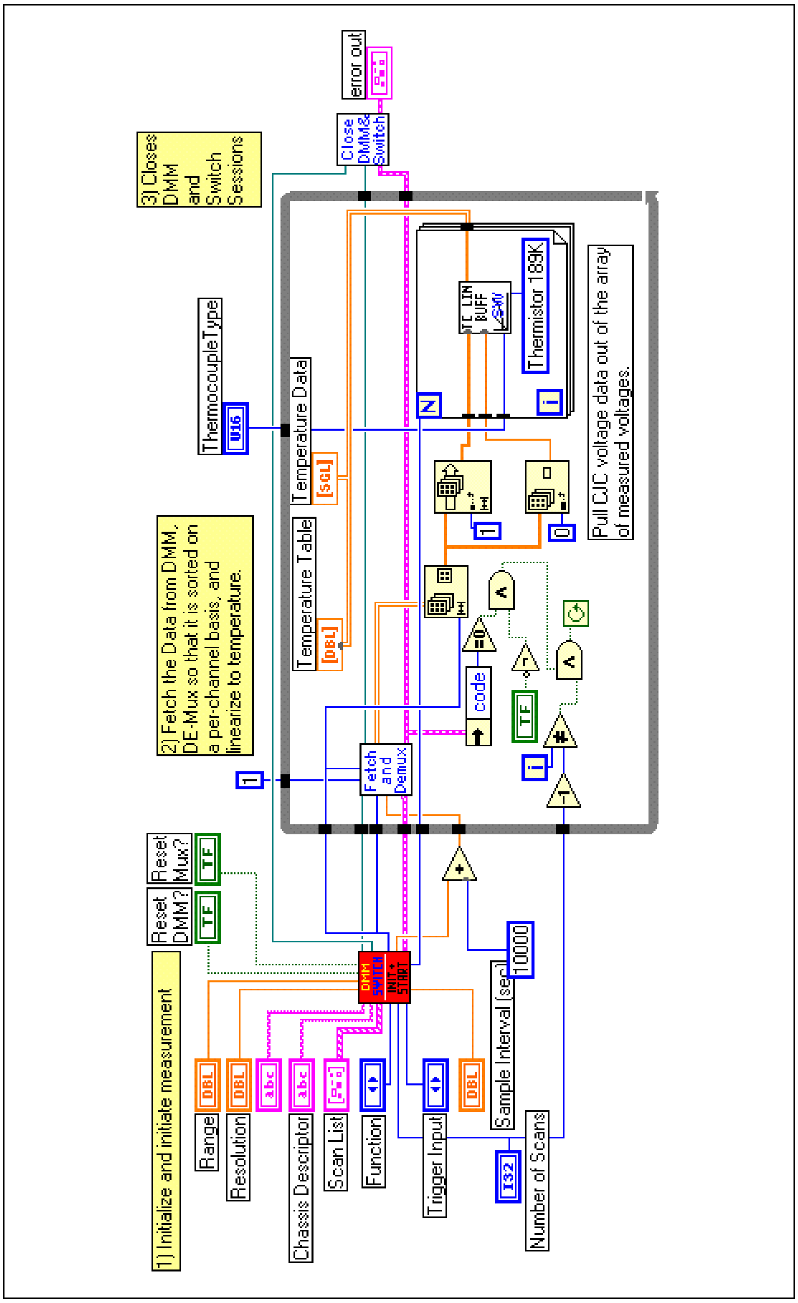
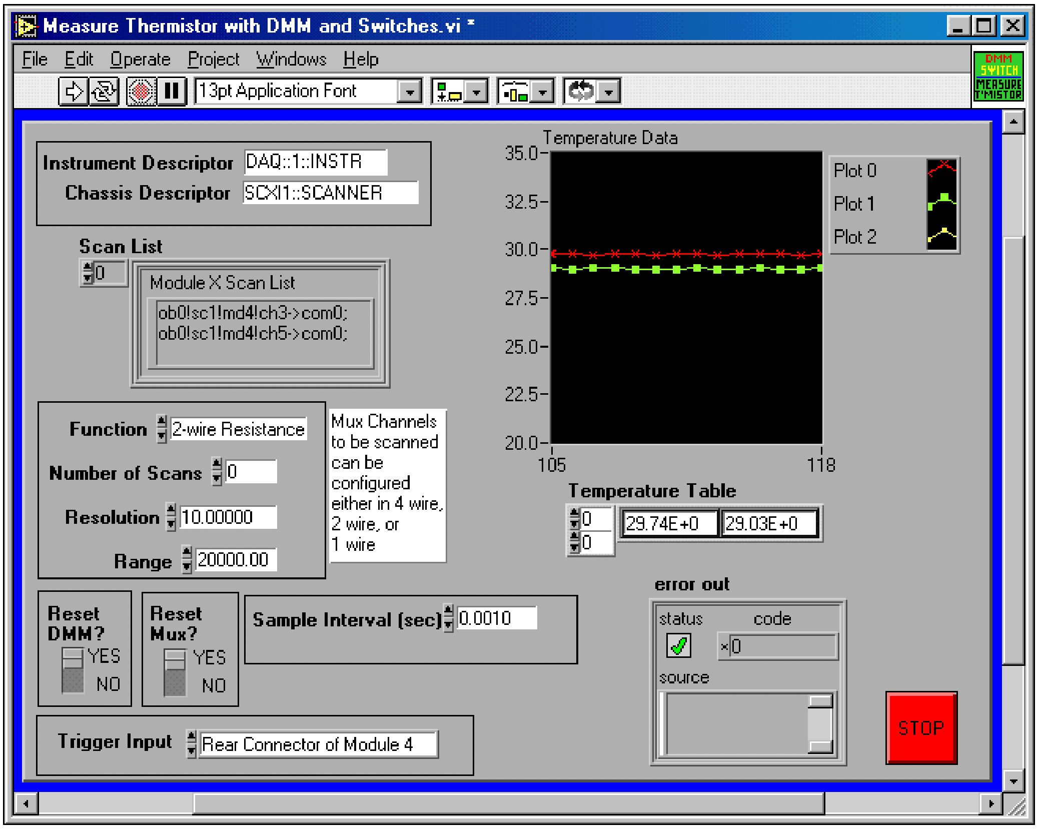


Figure 4-25. Measure Thermocouple with DMM and Switches Block Diagram

Temperature with Thermistor Measurement

To implement a temperature with thermistor measurement with the combination DMM-Switch, complete the following steps:

1. Open the Voltage Measurement example.
2. Go inside the fetch and demux loop, and implement linearization for the type of thermistor used.
3. Set **Excitation Current** to 1.00 and the **Type of Excitation** to Current Reference as inputs to the `Convert Thermistor Reading` function.
4. Set **Function** to two-wire resistance, and the **Range** to 20000 Ω, on the application panel.


Measure Thermistor with DMM and Switches.vi *

Instrument Descriptor: DAQ::1::INSTR
Chassis Descriptor: SCXI1::SCANNER

Scan List: 0
Module X Scan List:
ob0!sc1!md4!ch3->com0;
ob0!sc1!md4!ch5->com0;

Function: 2-wire Resistance
Number of Scans: 0
Resolution: 10.00000
Range: 20000.00

Mux Channels to be scanned can be configured either in 4 wire, 2 wire, or 1 wire

Reset DMM? YES / NO
Reset Mux? YES / NO
Sample Interval (sec): 0.0010
Trigger Input: Rear Connector of Module 4

Temperature Table: 29.74E+0 29.03E+0

error out: status ✓ code 0 source

Figure 4-26. Measure Thermistor with DMM and Switches Front Panel

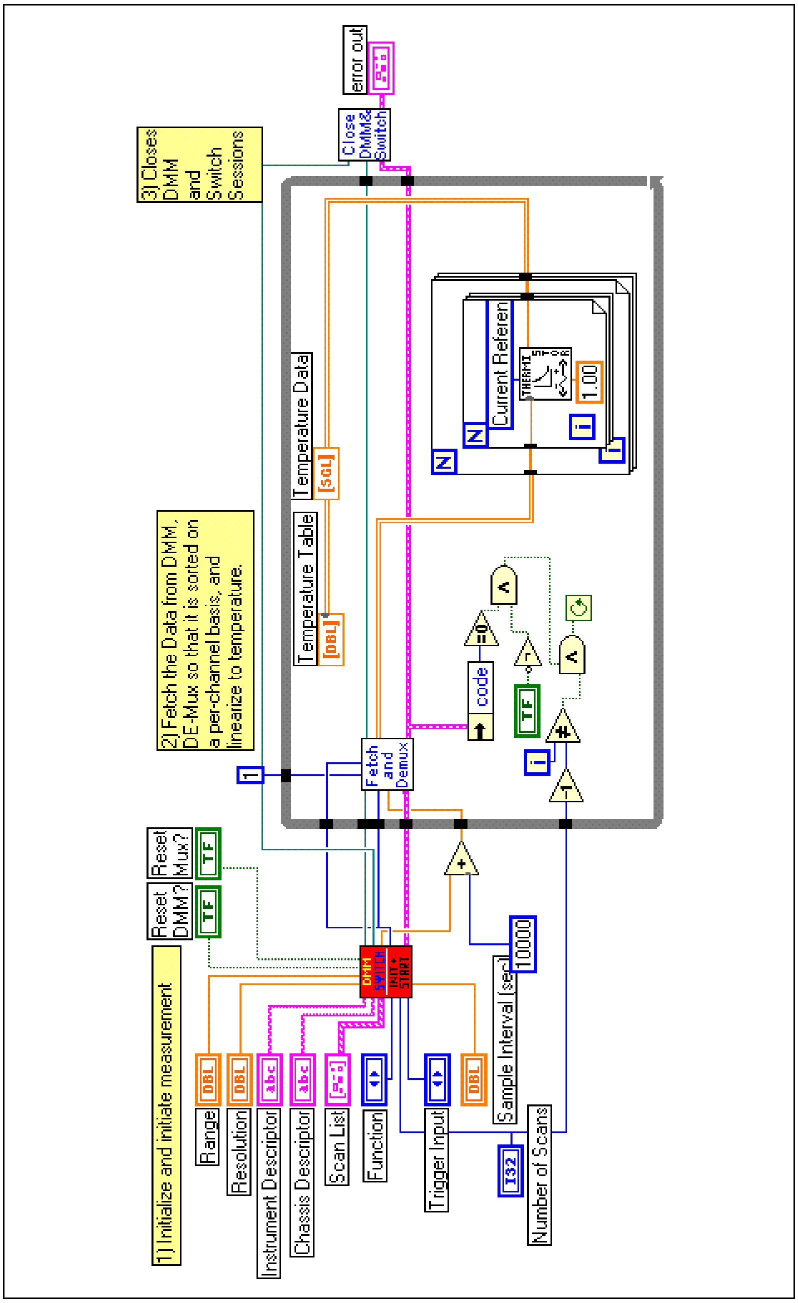
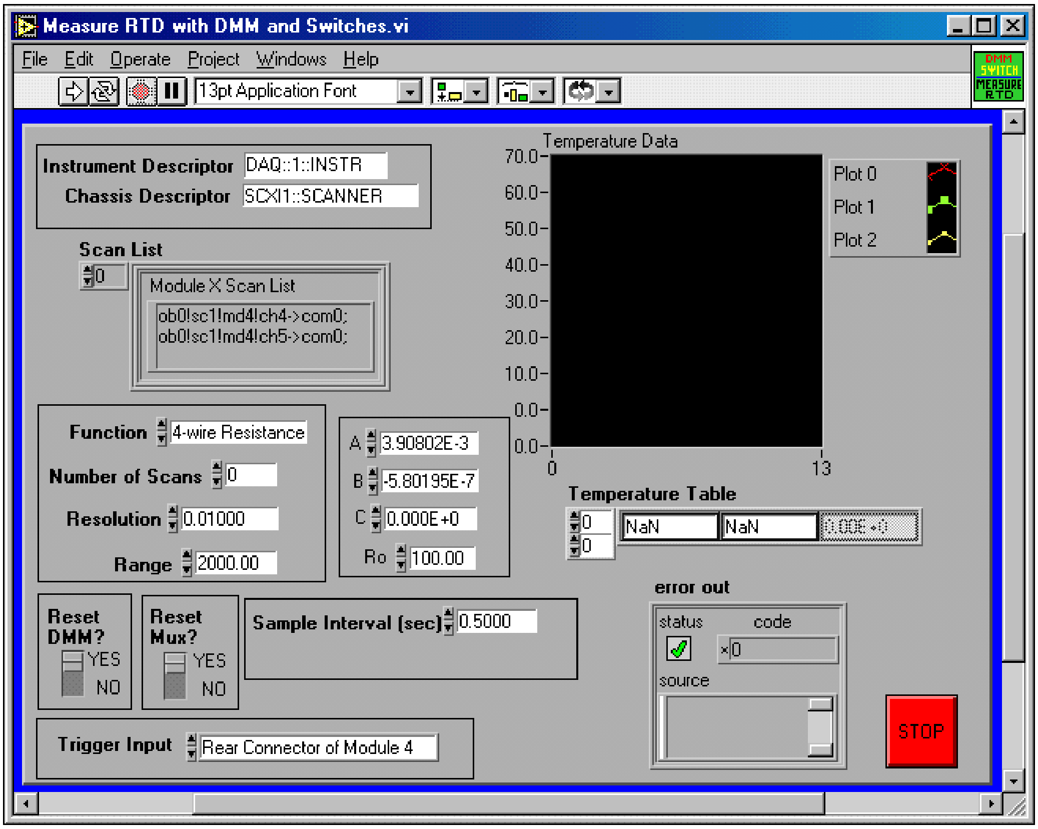


Figure 4-27. Measure Thermistor with DMM and Switches Block Diagram

Temperature with RTD Measurement

To implement a temperature measurement with RTDs using the DMM-Switch combination, complete the following steps:

1. Open the Voltage Measurement example.
2. Go inside the fetch and demux loop, and implement the linearization for the type of RTD used.
3. Specify the RTD coefficients A, B, C, and R0 in the `Convert RTD Reading` function.
4. On the application panel, set the **DMM Function** to four-wire resistance and the **Range** to 2000 Ω, or to the appropiate range that covers the RTD resistance range.



Instrument Descriptor DAQ::1::INSTR
Chassis Descriptor SCXI1::SCANNER
Scan List 0
Module X Scan List
ob0!sc1!md4!ch4->com0;
ob0!sc1!md4!ch5->com0;
Function 4-wire Resistance
Number of Scans 0
Resolution 0.01000
Range 2000.00
A 3.90802E-3
B -5.80195E-7
C 0.000E+0
Ro 100.00
Sample Interval (sec) 0.5000
Trigger Input Rear Connector of Module 4
Temperature Table 0 0 NaN NaN 0.00E+0
error out
status ✓ code 0
source

Figure 4-28. Measure RTD with DMM and Switches Front Panel

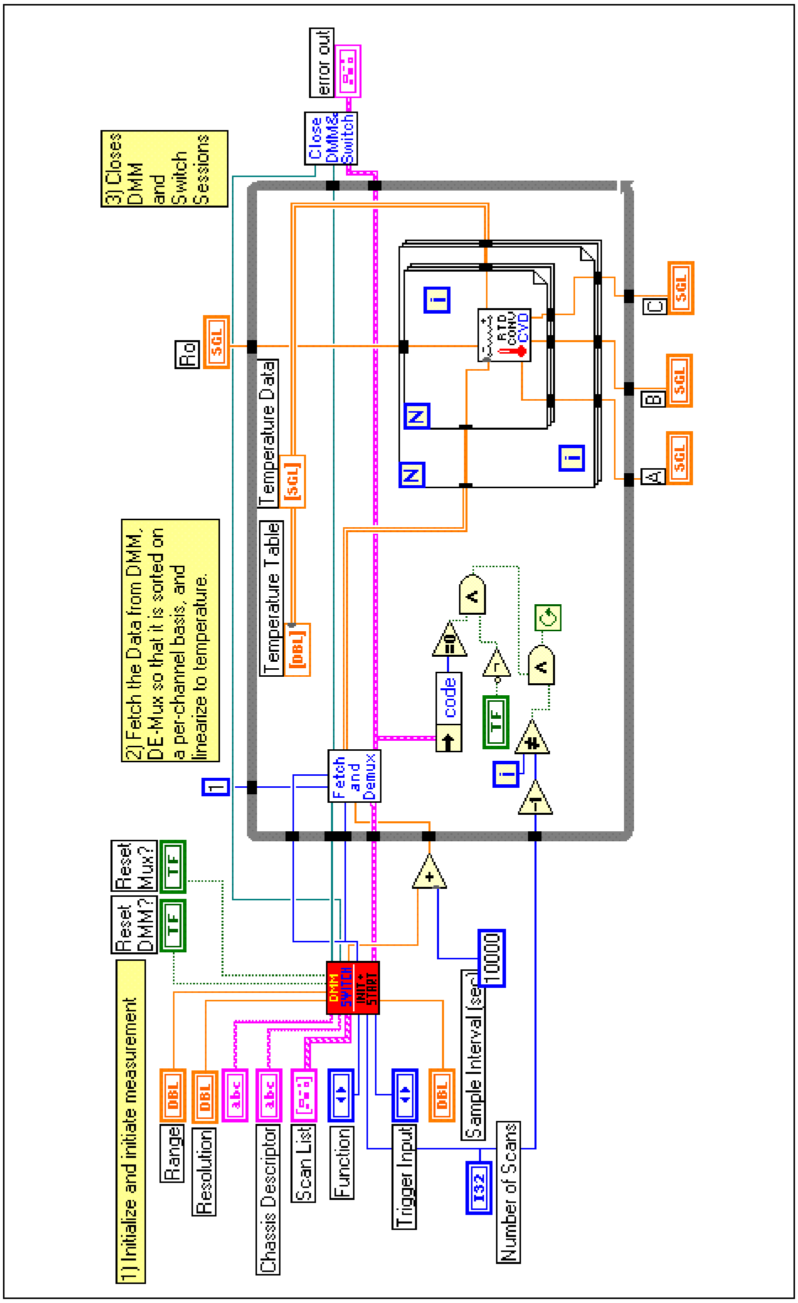
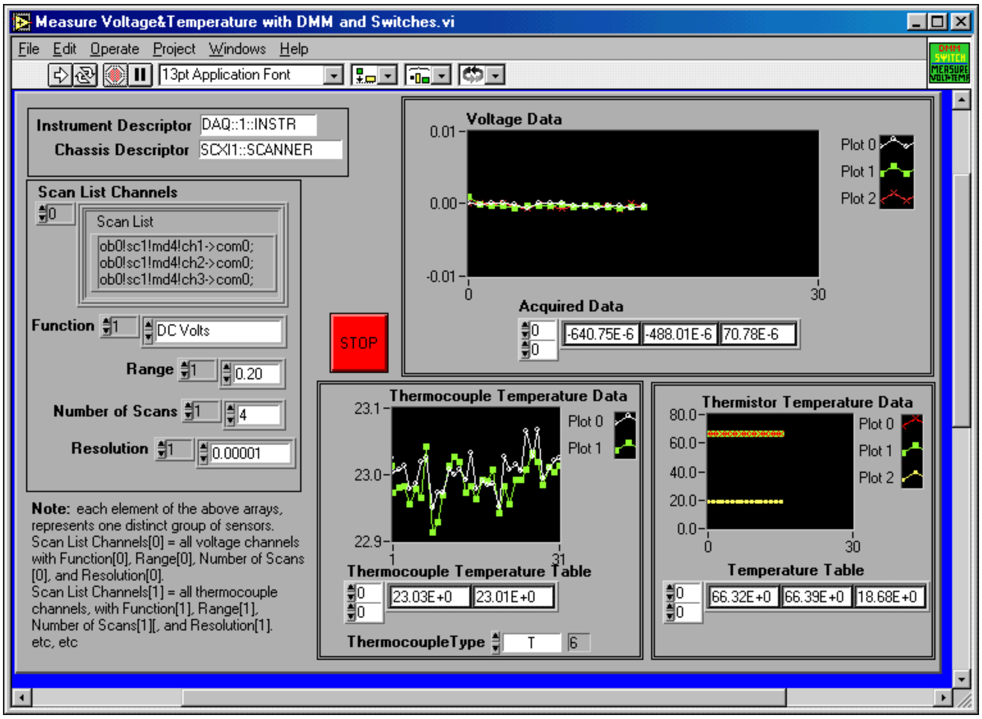


Figure 4-29. Measure RTD with DMM and Switches Block Diagram

Several Sensor-Type Measurements in the Same Application

Sometimes you may need to use combinations of different sensors. In the following example, the VIs are reading three voltage sensors, followed by three thermocouples, followed by three RTDs for temperature—everything inside the same application. Since the focus should be on scaling and presentation, not on setting up the scanning operation, the VI runs as a higher-level function, `Configure and Read Measurement`, which reads groups of channels of the same function, range, number of samples, and resolution type. This high-level function takes care of the acquisition part completely; it implements a foreground-type acquisition by scanning each group of sensors one at a time.

Once data from each distinct group of channels is acquired, it is easy to implement the scaling and presentation of data.



Instrument Descriptor DAQ::1::INSTR
Chassis Descriptor SCXI1::SCANNER

Scan List Channels 0
Scan List
ob0!sc1!md4!ch1->com0;
ob0!sc1!md4!ch2->com0;
ob0!sc1!md4!ch3->com0;

Function 1 DC Volts
Range 1 0.20
Number of Scans 1 4
Resolution 1 0.00001

Note: each element of the above arrays, represents one distinct group of sensors.
Scan List Channels[0] = all voltage channels with Function[0], Range[0], Number of Scans [0], and Resolution[0].
Scan List Channels[1] = all thermocouple channels, with Function[1], Range[1], Number of Scans[1][, and Resolution[1].
etc, etc

Acquired Data 0 0 -640.75E-6 -488.01E-6 70.78E-6

Thermocouple Temperature Table 0 0 23.03E+0 23.01E+0

ThermocoupleType T 6

Temperature Table 0 0 66.32E+0 66.39E+0 18.68E+0

Figure 4-30. Measure Voltage and Temperature with DMM and Switches Front Panel

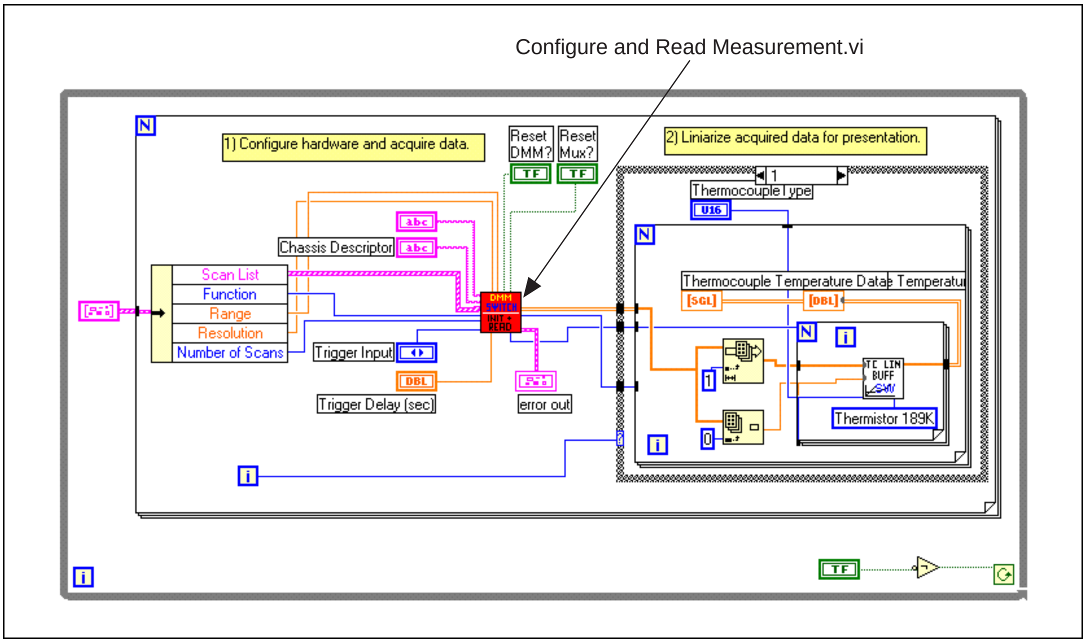
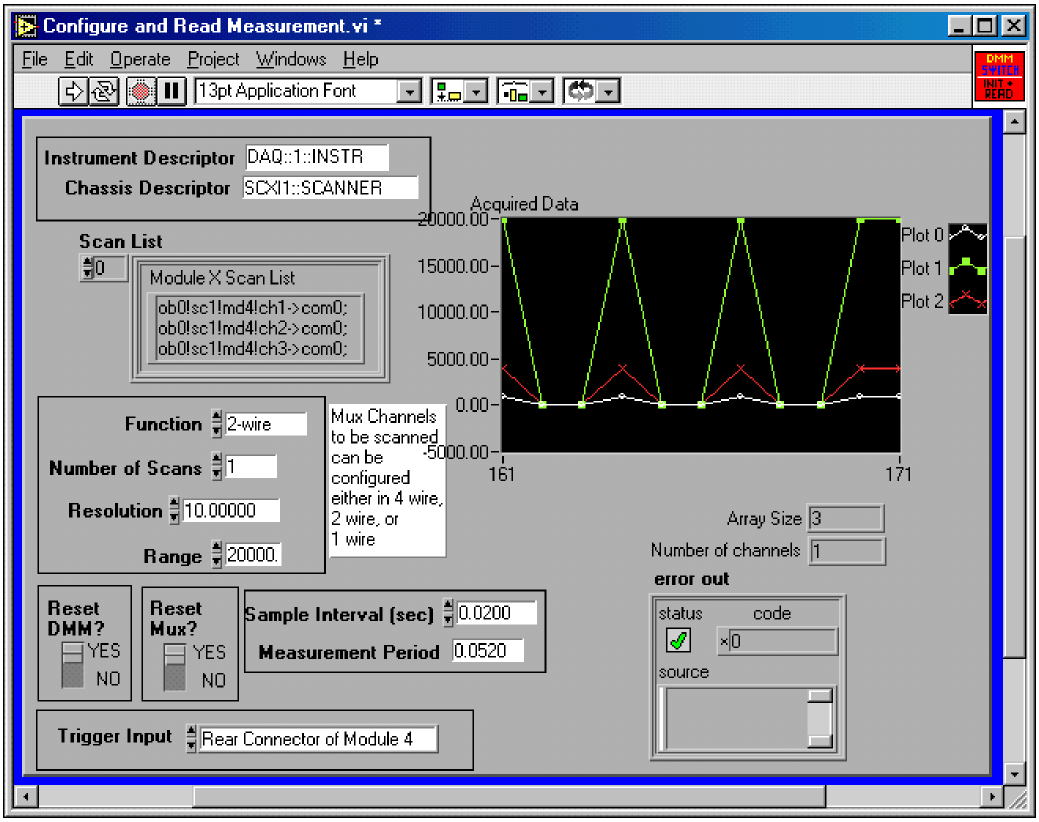


Figure 4-31. Measure Voltage and Temperature with DMM and Switches Block Diagram

The `Configure and Read Measurement.vi` is shown in Figure 4-32 and Figure 4-33.



Instrument Descriptor DAQ::1::INSTR
Chassis Descriptor SCXI1::SCANNER

Scan List 0

Module X Scan List
ob0!sc1!md4!ch1->com0;
ob0!sc1!md4!ch2->com0;
ob0!sc1!md4!ch3->com0;

Function 2-wire
Number of Scans 1
Resolution 10.00000
Range 20000.

Mux Channels to be scanned can be configured either in 4 wire, 2 wire, or 1 wire

Array Size 3
Number of channels 1

Reset DMM? YES NO
Reset Mux? YES NO

Sample Interval (sec) 0.0200
Measurement Period 0.0520

Trigger Input Rear Connector of Module 4

error out
status ✓ code 0
source

Figure 4-32. Configure and Read Measurement Front Panel

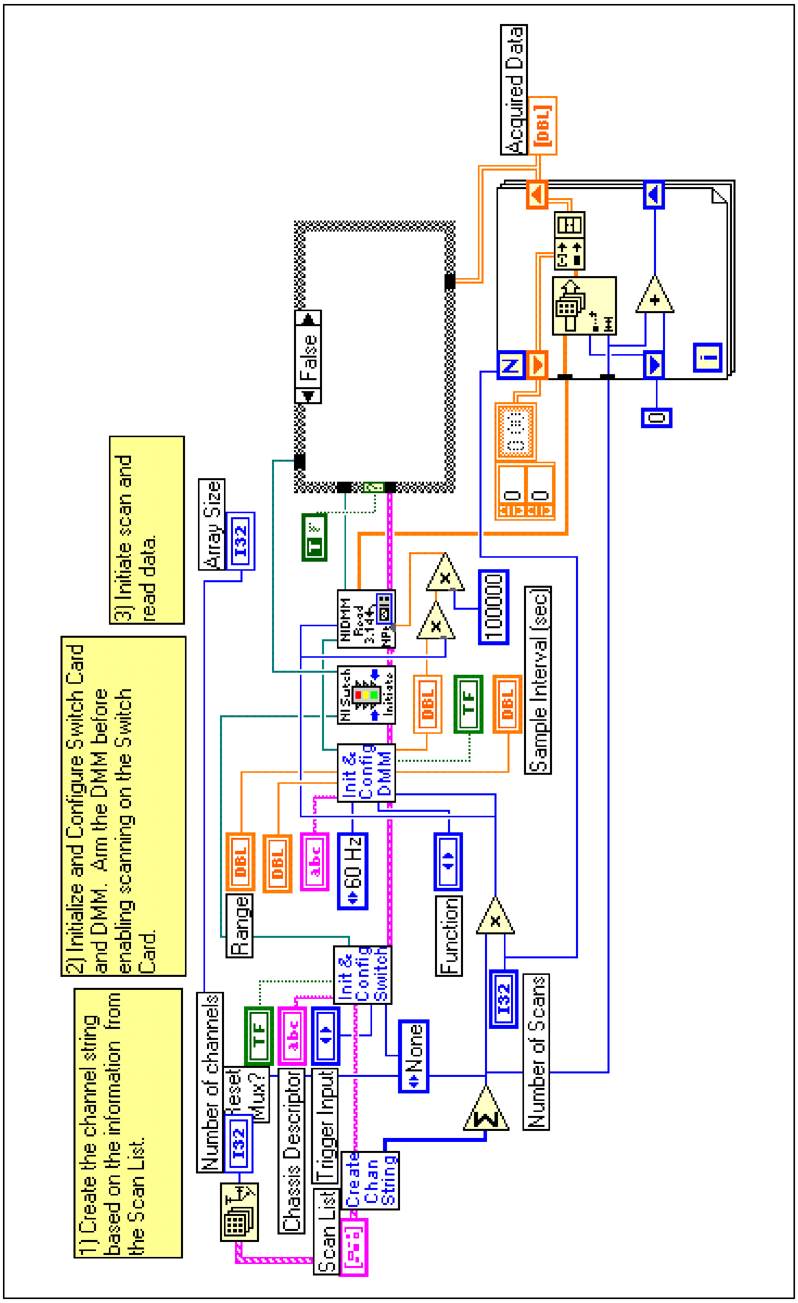


Figure 4-33. Configure and Read Measurement Block Diagram

5

NI-DAQ Driver Application Examples

NI-DAQ is a set of functions that control all of the National Instruments plug-in DAQ devices for analog I/O, digital I/O, timing I/O, SCXI signal conditioning, and Real-Time System Integration (RTSI) multiboard synchronization. DAQ devices are general-purpose data acquisition devices, which National Instruments makes and offers under the software umbrella of a unique driver software—NI-DAQ.

Computer-based instruments like the NI 4060 are instruments and so are not classified as general-purpose DAQ devices, but they match functionality in common with general-purpose DAQ devices. The general-purpose functionality NI-DAQ offers is also supported by the DMM product line. This architecture offers you two choices when using the DMM hardware: to use the hardware as an instrument with NI-DMM, or to use the hardware as a general-purpose DAQ device with NI-DAQ.

NI-DAQ Application Example

The most common technique for a nonbuffered, continuous analog input data acquisition with NI-DAQ is the sequence `AI_Configure` followed by `AI_Single_Scan` inside a While Loop. Call `AI_Configure` at the beginning of your application to inform NI-DAQ about the input mode (single-ended or differential), input range, and input polarity selected for the device. `AI_Single_Scan` returns one scan of data from the previously configured channel. This function starts an acquisition, retrieves a scan of data, and then terminates the acquisition. You can repeat these tasks by calling the function again and again inside a loop, as shown in Figure 5-2.

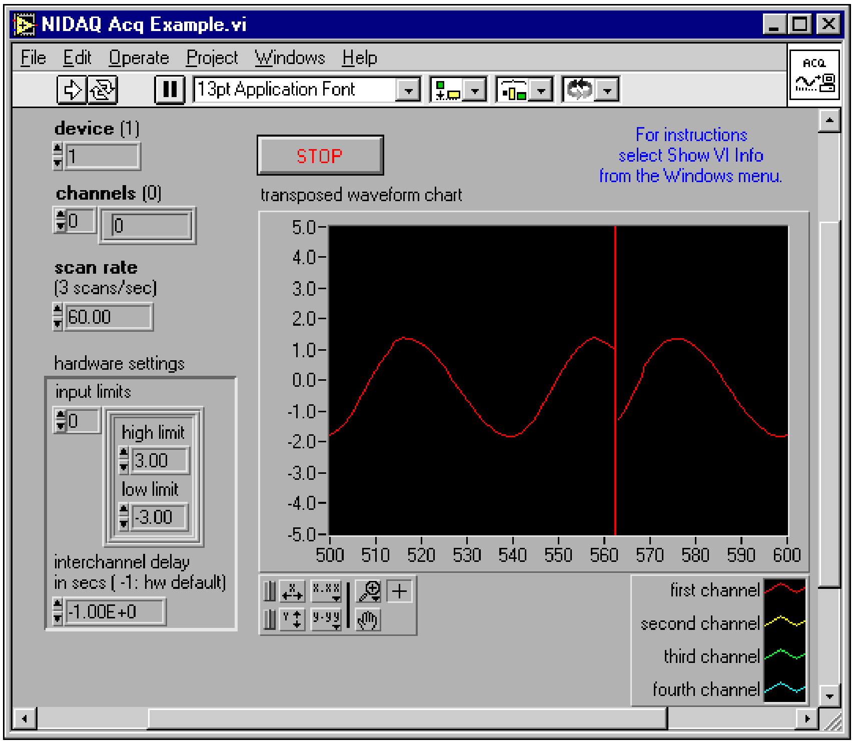


Figure 5-1. NI-DAQ Acquisition Example Front Panel

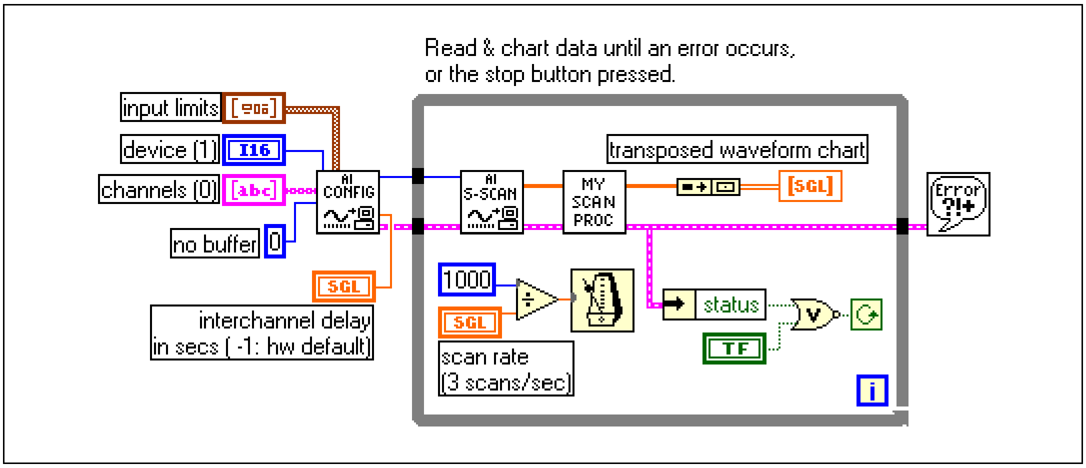

Read & chart data until an error occurs, or the stop button pressed.
input limits
device (1)
I16
channels (0)
abc
no buffer
0
SGL
interchannel delay in secs (-1: hw default)
AI CONFIG
AI S-SCAN
MY SCAN PROC
transposed waveform chart
SGL
1000
SGL
scan rate (3 scans/sec)
status
TF
i
Error ?!+

Figure 5-2. NI-DAQ Acquisition Example Block Diagram

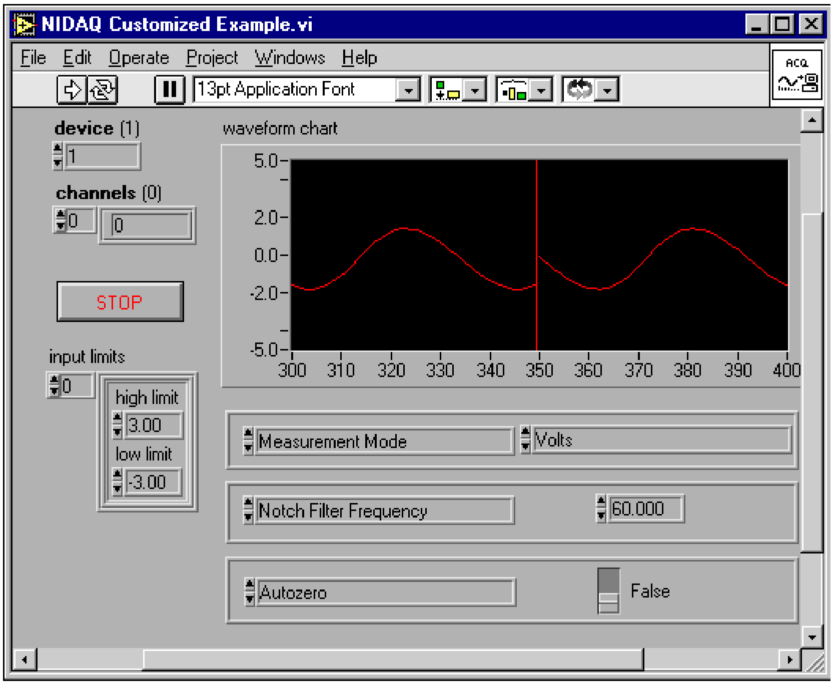
This general data acquisition example works with the DMM hardware.

With the examples in this chapter, you can configure the input limits, which in instrument terms are equivalent with configuring the range, and implement access to other configurations like the auto zero mode, powerline frequency, resolution, and function settings.

To be able to implement more instrument-type configurations with NI-DAQ, you need to divide the `AI_Configure` function into its components, `AI_Group_Config`, `AI_Parameter`, and `AI_Hardware_Config`. `AI_Group_Config` associates the device with a task ID. `AI_Paramater` gets, sets, or translates specific analog input parameters. `AI_Hardware_Config` configures the range and polarity.

Figure 5-3 shows a modified example that configures the DMM for DC volt measurement with 50 Hz or 60 Hz powerline frequency rejection. To use this example, complete the following steps:

1. Call `AI_Group_Config` to set up the device and channel.
2. Call as many `AI_Parameter` function calls as the number of configuration parameters you want to configure.
3. Call `AI_Hardware_Config` and `AI_Single_Scan` in a loop to acquire the data one scan at a time.



device (1)
1
channels (0)
0
0
input limits
0
high limit
3.00
low limit
-3.00
Measurement Mode
Volts
Notch Filter Frequency
60.000
Autozero
False

Figure 5-3. NI-DAQ Customized Example Front Panel

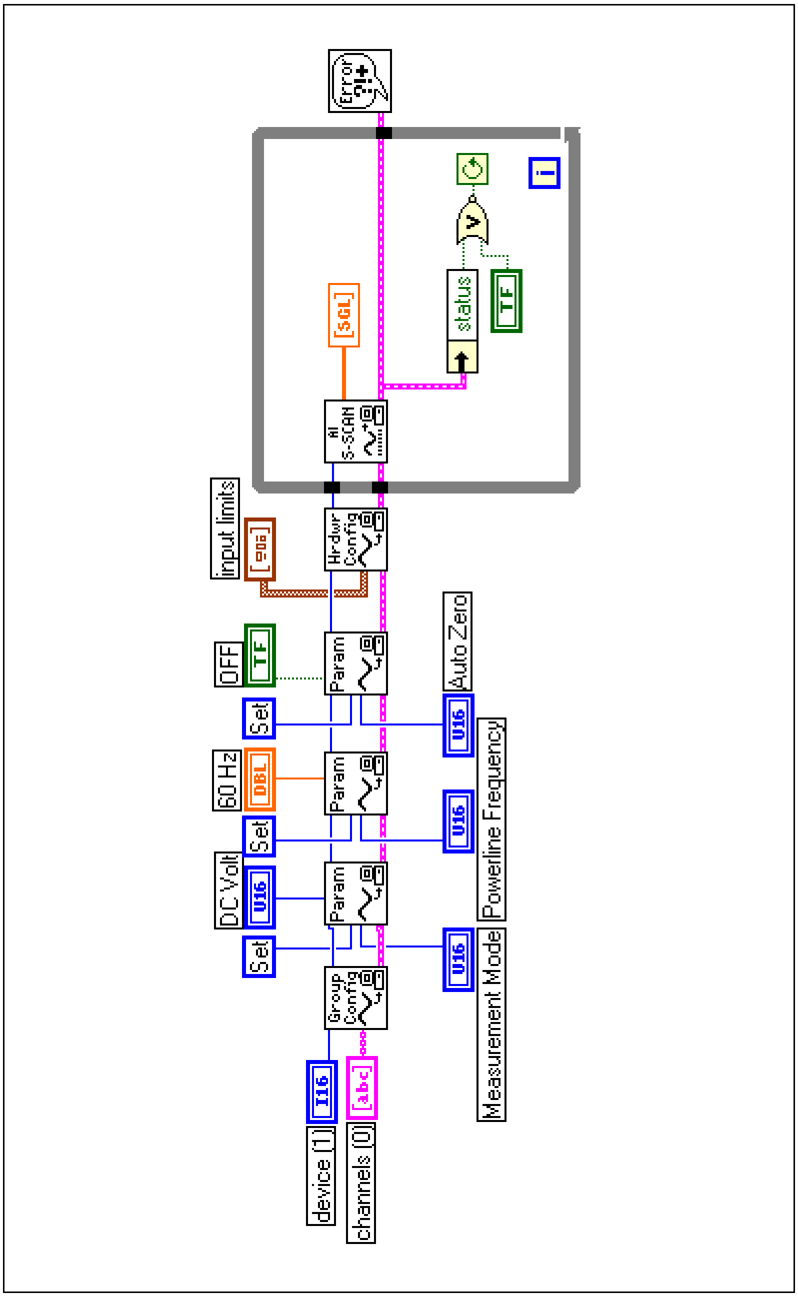


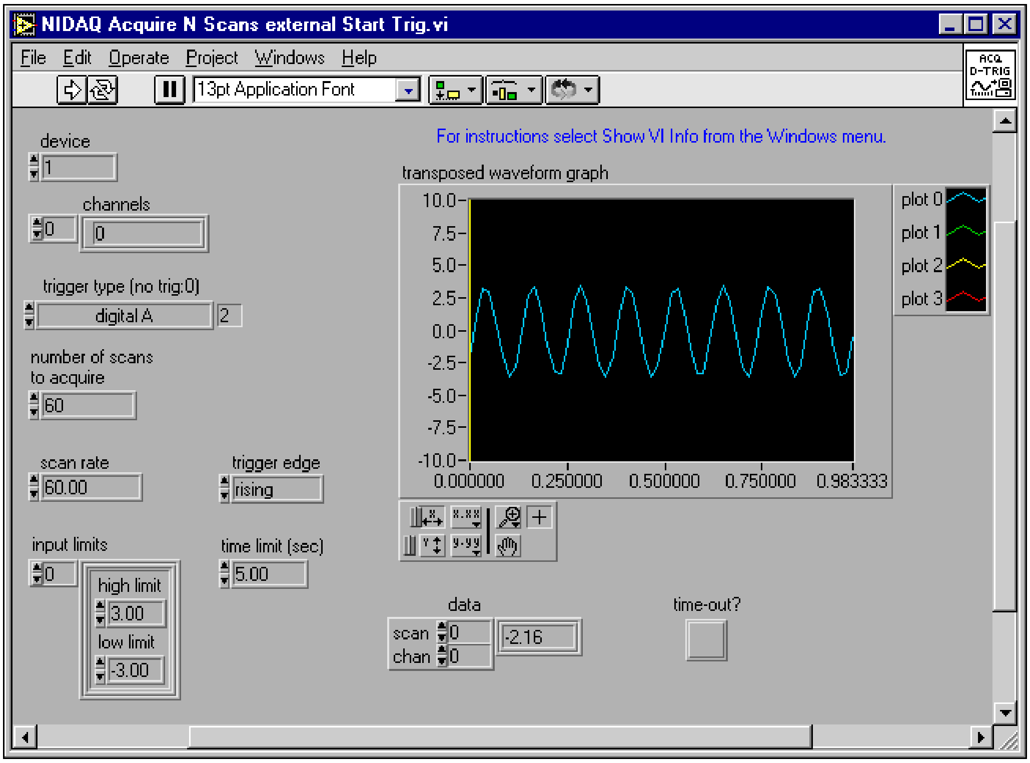
Figure 5-4. NI-DAQ Customized Example Block Diagram

NI-DAQ installs a similar LabVIEW example for NI-DMM hardware under **LabVIEW»Examples»Daq»AnalogIn»AnalogIn.llb»NI40*XX* Digital Multimeter.vi**.

Note If you used Data Neighborhood to create and configure a virtual DMM channel, you do not need to call `AI_Parameters`.

Using Triggering with NI-DAQ

If you want to implement a trigger, specify the type of trigger as digital A in the `AI_Start.vi` front panel. Make the read/search position parameter equal to relative to trigger point in the `AI_Read.vi` block diagram. See Figure 5-6.



For instructions select Show VI Info from the Windows menu.

device: 1
channels: 0 | 0
trigger type (no trig:0): digital A | 2
number of scans to acquire: 60
scan rate: 60.00
trigger edge: rising
input limits: 0
high limit: 3.00
low limit: -3.00
time limit (sec): 5.00
data: scan 0, chan 0, -2.16
time-out?

Figure 5-5. NI-DAQ Acquire *N* Scans External Start Trigger Front Panel

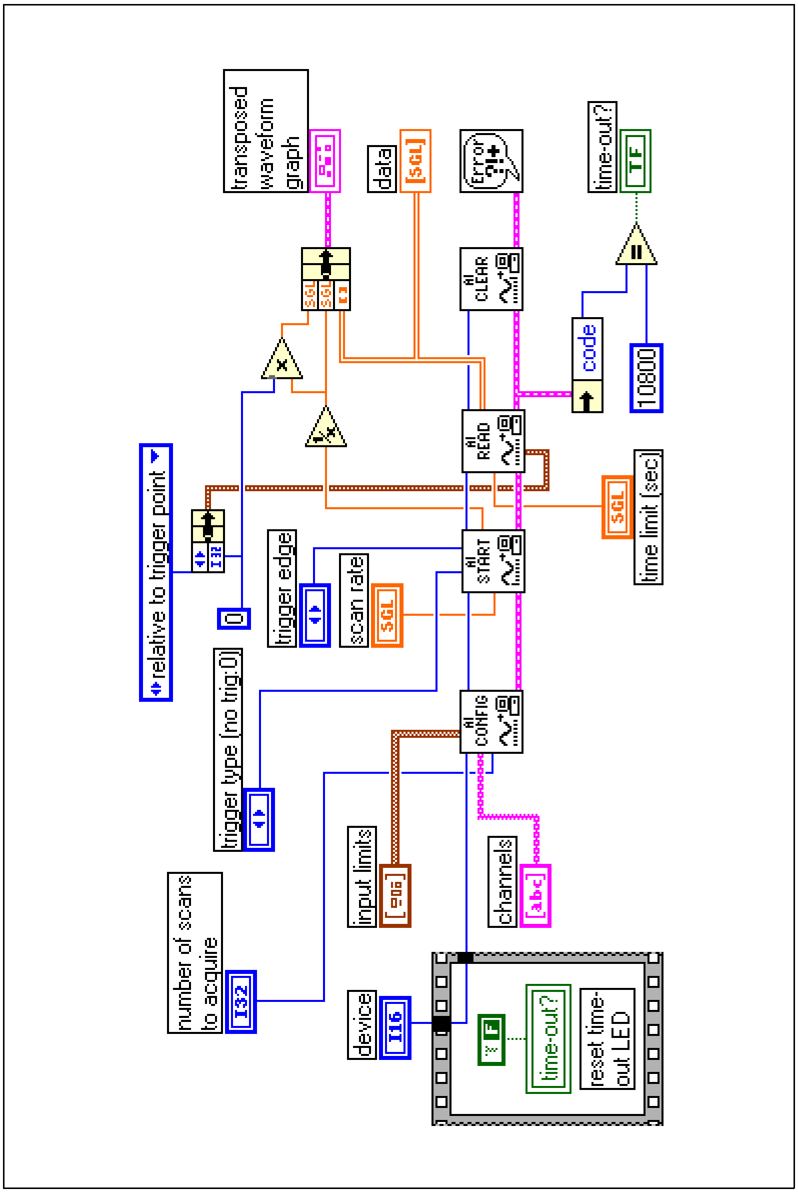
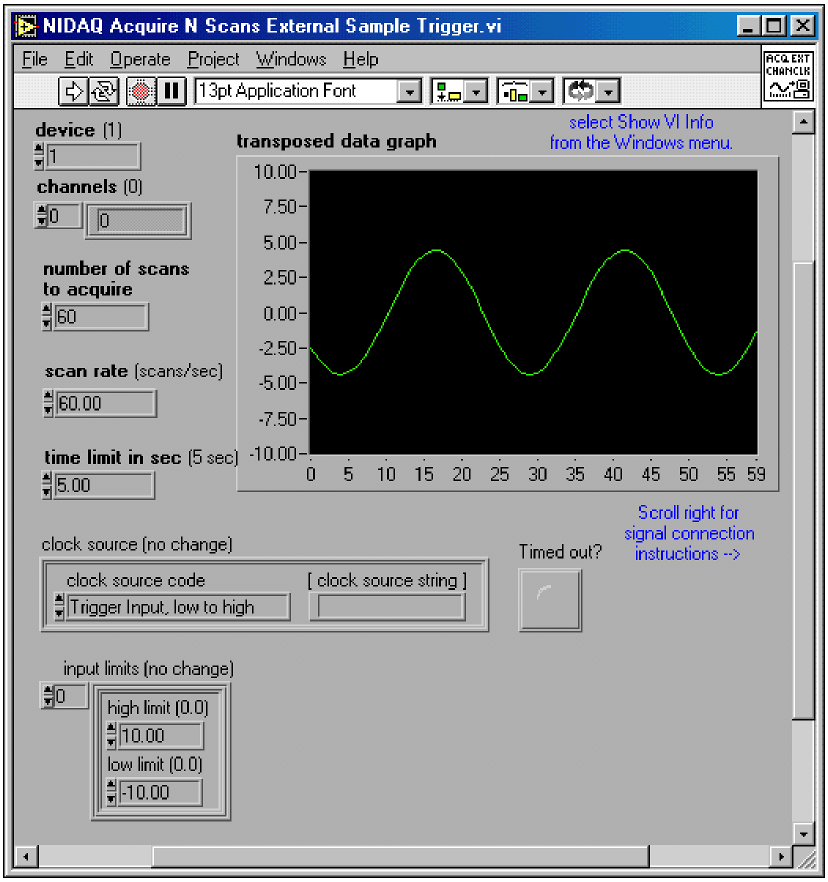


Figure 5-6. NI-DAQ Acquire *N* Scans External Start Trigger Block Diagram

If you want to implement an external sample trigger, you need to specify which clock (scan clock 1) and clock source code (Trigger Input, low to high) in the `AI_Clock_Config.vi` front panel. As shown in Figure 5-7.


NIDAQ Acquire N Scans External Sample Trigger.vi
File Edit Operate Project Windows Help
13pt Application Font
ACQ EXT CHANCLK
select Show VI Info from the Windows menu.
transposed data graph
10.00
7.50
5.00
2.50
0.00
-2.50
-5.00
-7.50
-10.00
0 5 10 15 20 25 30 35 40 45 50 55 59
Scroll right for signal connection instructions -->
Timed out?

device (1): 1
channels (0): 0, 0
number of scans to acquire: 60
scan rate (scans/sec): 60.00
time limit in sec (5 sec): 5.00
clock source (no change)
clock source code: Trigger Input, low to high
[clock source string]:
input limits (no change): 0
high limit (0.0): 10.00
low limit (0.0): -10.00

Figure 5-7. NI-DAQ Acquire *N* Scans External Sample Trigger Front Panel

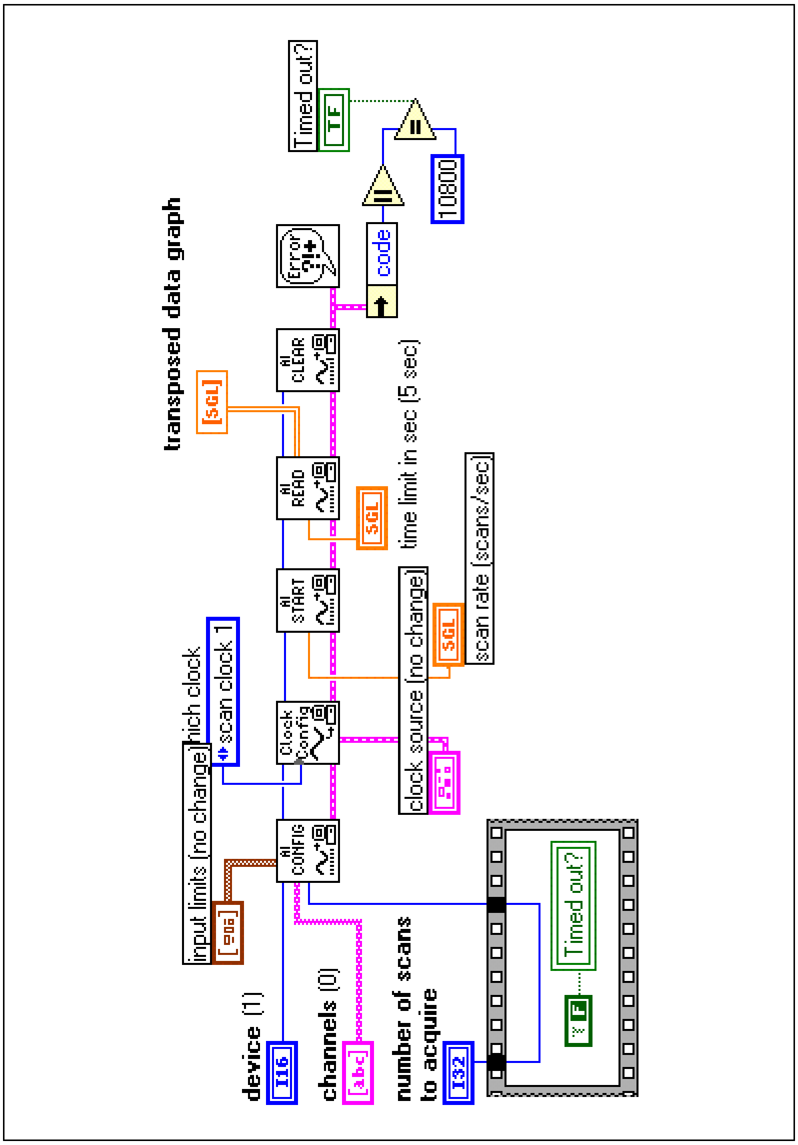


Figure 5-8. NI-DAQ Acquire *N* Scans External Sample Trigger Block Diagram

Using the DMM with SCXI Multiplexers in NI-DAQ

You can scan an SCXI multiplexer such as the SCXI-1127 using the following sequence of calls in a loop:

- `AI_Configure`
- `AI_Start`
- `AI_Read`

For example, if you have one SCXI chassis (your SCXI-1127 module is located in slot 4 of the chassis) and if you are scanning channels 1, 2, and 3 of the module, then you would set the channel input to ob0!sc1!md4!1:3 on the panel. NI-DAQ ensures the scan clock is routed from the DMM to the SCXI-1127 module.

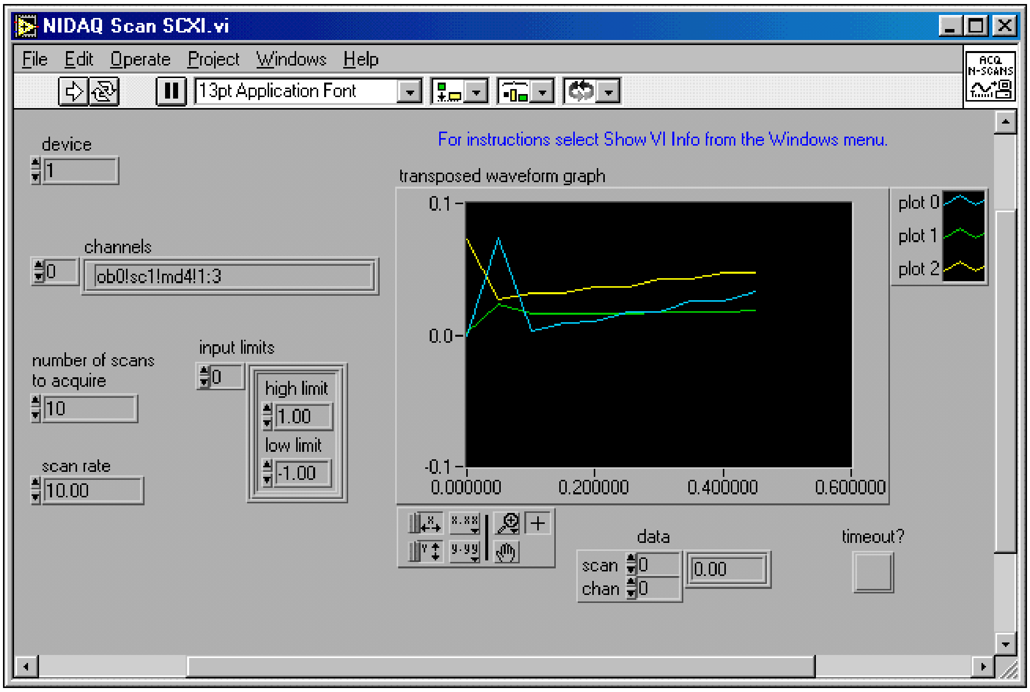


Figure 5-9. NI-DAQ Scan SCXI Front Panel

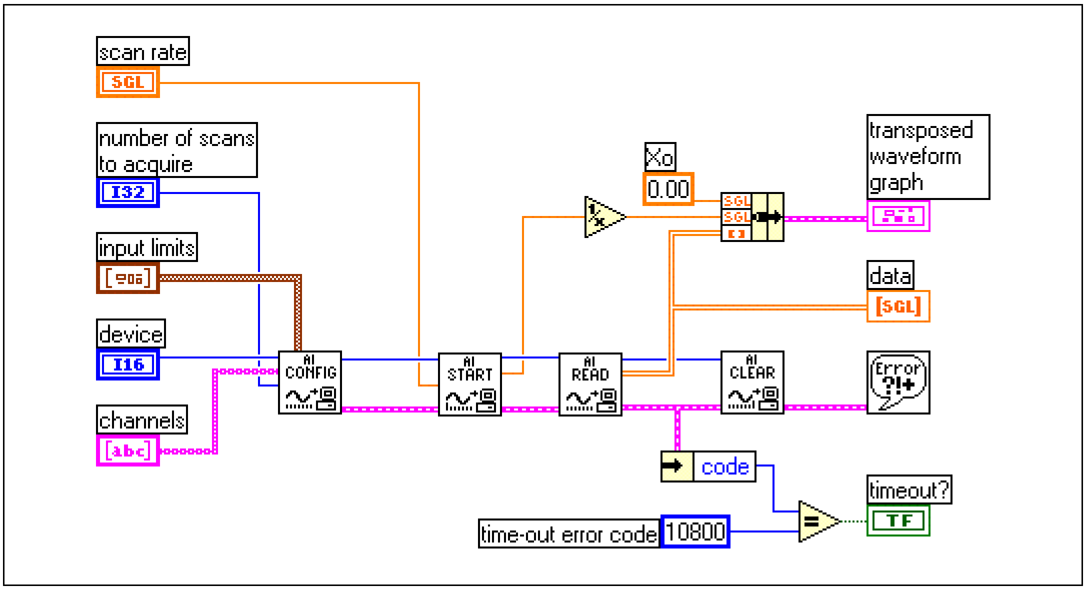


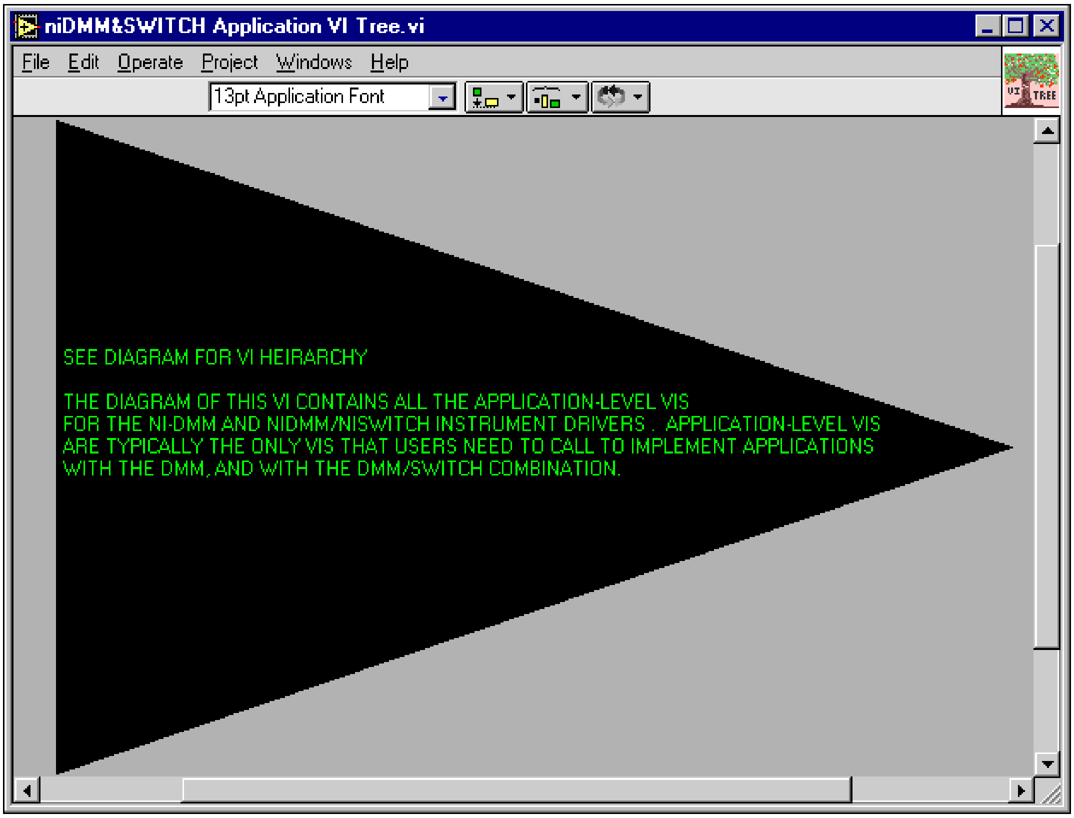
Figure 5-10. NI-DAQ Scan SCXI Block Diagram

6

Build Your Own Application

The most important principle to remember when building your new application is to *never start from scratch*! You can save time and effort and eliminate mistakes by using or modifying the example programs provided with NI-DMM.

If you are using LabVIEW, open the `niDMM&Switch Application VI Tree`. Shown in Figure 6-1. You can find the application tree on the NI-SWITCH CD under the `Example` directory.


niDMM&SWITCH Application VI Tree.vi
File Edit Operate Project Windows Help
13pt Application Font
VI TREE

SEE DIAGRAM FOR VI HEIRARCHY

THE DIAGRAM OF THIS VI CONTAINS ALL THE APPLICATION-LEVEL VIS FOR THE NI-DMM AND NIDMM/NISWITCH INSTRUMENT DRIVERS . APPLICATION-LEVEL VIS ARE TYPICALLY THE ONLY VIS THAT USERS NEED TO CALL TO IMPLEMENT APPLICATIONS WITH THE DMM, AND WITH THE DMM/SWITCH COMBINATION.

Figure 6-1. Application Tree Front Panel

Choose the example that best applies to your application. See Figure 6-2. You can use the example as it is, or modify it according to what you want to implement, by using any of the other examples that show how the modification should be implemented.

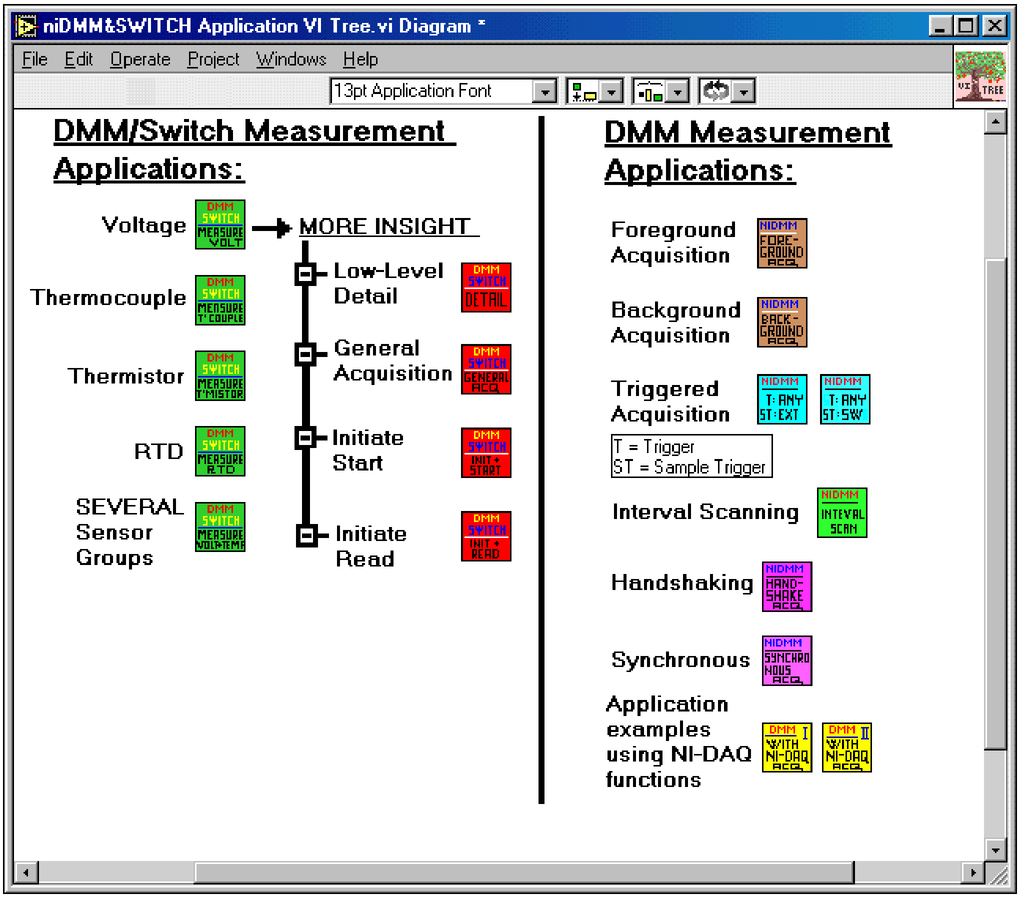


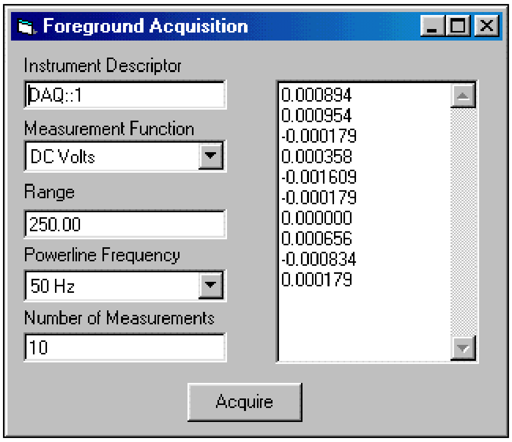
Figure 6-2. Application Tree Block Diagram

The application tree contains examples for measuring voltages and taking measurements with thermocouples, RTDs, and thermistors. For other sensor types, use the Measure Voltage example and apply your own scaling to voltage data.

Microsoft Visual Basic Examples

This appendix shows the Visual Basic syntax of the examples in this manual.

Foreground Acquisition



Instrument Descriptor: DAQ::1
Measurement Function: DC Volts
Range: 250.00
Powerline Frequency: 50 Hz
Number of Measurements: 10

```
0.000894
0.000954
-0.000179
0.000358
-0.001609
-0.000179
0.000000
0.000656
-0.000834
0.000179
```

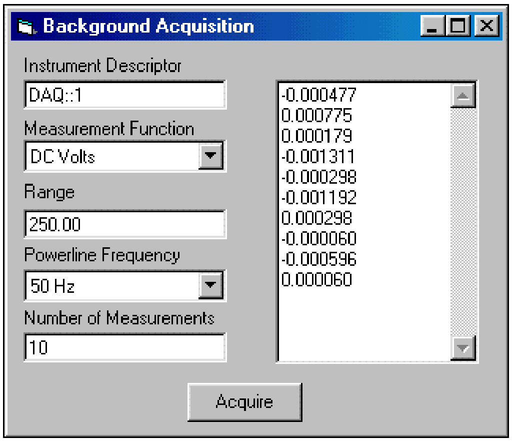
Figure A-1. Foreground Acquisition Front Panel

```
Private Sub Acquire_Click()

    Dim vi As ViSession
    Dim resolution As ViReal64
    Dim triggerDelay As ViReal64
    Dim timeLimit As ViInt32
    Dim numPointsRead As ViInt32
    Dim i As ViInt32
    Dim numelements As ViInt32
    Dim measurementArray() As ViReal64
    ReDim measurementArray(1 To Val(numOfMeasurements.Text))
    measurements.Text = ""

    'Disable Acquire button until acquistion complete
```

```
Acquire.Enabled = False
'Open a session to the instrument
If CheckError(niDMM_init(instrumentDescriptor.Text, VI_TRUE, VI_TRUE, _
                vi)) Then
    GoTo Error
End If
'Set these values to desired defaults.
resolution = 1#
triggerDelay = 0#
timeLimit = 10000
resolution = 1#
'Set the powerline frequency
If CheckError(niDMM_ConfigurePowerlineFrequency(vi, _
                plfVal(powerlineFreq.ListIndex))) Then
 GoTo Error
End If
'Configure the acquistion
If CheckError(niDMM_ConfigureMeasurement(vi, CtlVal(measFunction), _
                Val(range.Text), resolution)) Then
 GoTo Error
End If
'Set trigger source and delay
If CheckError(niDMM_ConfigureTrigger(vi, NIDMM_VAL_IMMEDIATE, _
                triggerDelay)) Then
GoTo Error
End If
'Set the number of measurements
If CheckError(niDMM_ConfigureMultiPoint(vi, 1,
                Val(numOfMeasurements.Text), _
                NIDMM_VAL_IMMEDIATE, 0#)) Then
GoTo Error
End If
'Read the measurements
If CheckError(niDMM_ReadMultiPoint(vi, timeLimit, _
                CInt(numOfMeasurements.Text), _
                measurementArray(1), numelements)) Then
GoTo Error
End If
    'Format and display measurements
    For i = 1 To CInt(numOfMeasurements.Text)
        measurements.Text = measurements.Text & _
```

```
                    Format(measurementArray(i), "##0.000000") & _
                        vbCrLf
        Next i
    'Abort the acquisition
    If CheckError(niDMM_Abort(vi)) Then
    GoTo Error
    End If
    'Always close the instrument, even if an error occured in
    'niDMM_SimpleAcquisition
    niDMM_close (vi)
Error:
    'Enable acquire button
    Acquire.Enabled = True
End Sub
```

Background Acquisition



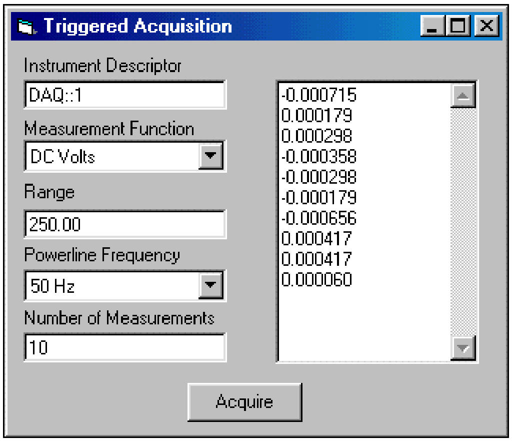
Instrument Descriptor: DAQ::1
Measurement Function: DC Volts
Range: 250.00
Powerline Frequency: 50 Hz
Number of Measurements: 10

```
-0.000477
0.000775
0.000179
-0.001311
-0.000298
-0.001192
0.000298
-0.000060
-0.000596
0.000060
```

Figure A-2. Background Acquisition Front Panel

```
Private Sub Acquire_Click()
    Dim vi As ViSession
    Dim resolution As ViReal64
    Dim triggerDelay As ViReal64
    Dim timeLimit As ViInt32
    Dim numPointsRead As ViInt32
    Dim i As ViInt32
    Dim numelements As ViInt32
    Dim measurementArray() As ViReal64
    ReDim measurementArray(1 To Val(numOfMeasurements.Text))
    measurements.Text = ""
    'Disable Acquire button until acquistion complete
    Acquire.Enabled = False
    'Open a session to the instrument
    If CheckError(niDMM_init(instrumentDescriptor.Text, VI_TRUE, VI_TRUE, _
                    vi)) Then
        GoTo Error
    End If
    'Set these values to desired defaults.
    resolution = 1#
    triggerDelay = 0#
    timeLimit = 10000
    resolution = 1#
    'Set the powerline frequency
    If CheckError(niDMM_ConfigurePowerlineFrequency(vi, _
                    plfVal(powerlineFreq.ListIndex))) Then
     GoTo Error
    End If
'Configure the acquistion
    If CheckError(niDMM_ConfigureMeasurement(vi, CtlVal(measFunction), _
                    Val(range.Text), resolution)) Then
     GoTo Error
    End If
    'Set trigger source and delay
    If CheckError(niDMM_ConfigureTrigger(vi, NIDMM_VAL_IMMEDIATE, _
                    triggerDelay)) Then
    GoTo Error
    End If
    'Set the number of measurements
    If CheckError(niDMM_ConfigureMultiPoint(vi, 1,
                    Val(numOfMeasurements.Text), _
                    NIDMM_VAL_IMMEDIATE, 0#)) Then
    GoTo Error
    End If
    'Initiate the measurements
    If CheckError(niDMM_Initiate(vi)) Then
    GoTo Error
    End If
    'Read the measurements
    If CheckError(niDMM_FetchMultiPoint(vi, timeLimit, _
                CInt(numOfMeasurements.Text), _
                measurementArray(1), numelements)) Then
    GoTo Error
    End If

        'Format and display measurements
        For i = 1 To CInt(numOfMeasurements.Text)
            measurements.Text = measurements.Text & _
                    Format(measurementArray(i), "##0.000000") & _
                    vbCrLf

        Next i

    'Abort the acquisition
    If CheckError(niDMM_Abort(vi)) Then
    GoTo Error
    End If

    'Always close the instrument, even if an error occured in
    'niDMM_SimpleAcquisition
    niDMM_close (vi)

Error:
    'Enable acquire button
    Acquire.Enabled = True

End Sub
```

Triggered Acquisition



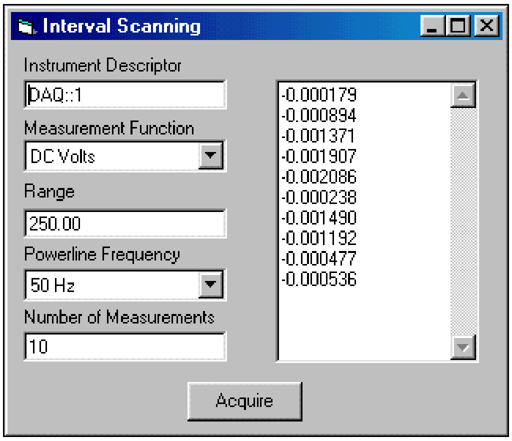
```
-0.000715
0.000179
0.000298
-0.000358
-0.000298
-0.000179
-0.000656
0.000417
0.000417
0.000060
```

Figure A-3. Triggered Acquisition Front Panel

```
Private Sub Acquire_Click()

    Dim vi As ViSession
    Dim resolution As ViReal64
    Dim triggerDelay As ViReal64
    Dim TrigSource As ViInt32
    Dim TrigSlope As ViInt32
    Dim SampleTrigSource As ViInt32
    Dim SampleTrigSlope As ViInt32
    Dim timeLimit As ViInt32
    Dim numPointsRead As ViInt32
    Dim i As ViInt32
    Dim numelements As ViInt32
    Dim measurementArray() As ViReal64
    ReDim measurementArray(1 To Val(numOfMeasurements.Text))
    measurements.Text = ""

    TrigSource = NIDMM_VAL_IMMEDIATE
    TrigSlope = NIDMM_VAL_POS
    SampleTrigSource = NIDMM_VAL_IMMEDIATE
    SampleTrigSlope = NIDMM_VAL_POS

    'Disable Acquire button until acquistion complete
    Acquire.Enabled = False

    'Open a session to the instrument
    If CheckError(niDMM_init(instrumentDescriptor.Text, VI_TRUE, VI_TRUE, _
                    vi)) Then
    GoTo Error
End If
'Set these values to desired defaults.
resolution = 1#
triggerDelay = 0#
timeLimit = 10000
'Set the powerline frequency
If CheckError(niDMM_ConfigurePowerlineFrequency(vi, _
                plfVal(powerlineFreq.ListIndex))) Then
 GoTo Error
End If
'Configure the acquistion
If CheckError(niDMM_ConfigureMeasurement(vi, CtlVal(measFunction), _
                Val(range.Text), resolution)) Then
 GoTo Error
End If
'Set trigger source and delay
If CheckError(niDMM_ConfigureTrigger(vi, TrigSource, _
                triggerDelay)) Then
GoTo Error
End If
'Set trigger slope
If CheckError(niDMM_ConfigureTriggerSlope(vi, TrigSlope)) Then
GoTo Error
End If
'Set the number of measurements
If CheckError(niDMM_ConfigureMultiPoint(vi, 1,
                Val(numOfMeasurements.Text), _
                SampleTrigSource, 0#)) Then
GoTo Error
End If
'Set sample trigger slope
'If CheckError(niDMM_ConfigureSampleTriggerSlope(vi, SampleTrigSlope))
                Then
'GoTo Error
'End If
'Read the measurements
If CheckError(niDMM_ReadMultiPoint(vi, timeLimit, _
                CInt(numOfMeasurements.Text), _
                measurementArray(1), numelements)) Then
GoTo Error
```

```
    End If
        'Format and display measurements
        For i = 1 To CInt(numOfMeasurements.Text)
            measurements.Text = measurements.Text & _
                    Format(measurementArray(i), "##0.000000") & _
                    vbCrLf
        Next i
    'Abort the acquisition
    If CheckError(niDMM_Abort(vi)) Then
    GoTo Error
    End If
    'Always close the instrument, even if an error occured in
    'niDMM_SimpleAcquisition
    niDMM_close (vi)
Error:
    'Enable acquire button
```

Interval Scanning



```
-0.000179
-0.000894
-0.001371
-0.001907
-0.002086
-0.000238
-0.001490
-0.001192
-0.000477
-0.000536
```

Figure A-4. Interval Scanning Front Panel

```
Private Sub Acquire_Click()
    Dim vi As ViSession
    Dim resolution As ViReal64
    Dim SampleTrigSource As ViInt32
    Dim SampleInterval As ViReal64
    Dim timeLimit As ViInt32
    Dim numPointsRead As ViInt32
    Dim i As ViInt32
    Dim numelements As ViInt32
    Dim measurementArray() As ViReal64
    ReDim measurementArray(1 To Val(numOfMeasurements.Text))
    measurements.Text = ""
    SampleTrigSource = NIDMM_VAL_INTERVAL
    SampleInterval = 0.1
    'Disable Acquire button until acquistion complete
    Acquire.Enabled = False
    'Open a session to the instrument
    If CheckError(niDMM_init(instrumentDescriptor.Text, VI_TRUE, VI_TRUE, _
                    vi)) Then
        GoTo Error
    End If
    'Set these values to desired defaults.
    resolution = 1#
    timeLimit = 10000
    'Set the powerline frequency
    If CheckError(niDMM_ConfigurePowerlineFrequency(vi, _
                    plfVal(powerlineFreq.ListIndex))) Then
     GoTo Error
    End If
    'Configure the acquistion
    If CheckError(niDMM_ConfigureMeasurement(vi, CtlVal(measFunction), _
                    Val(range.Text), resolution)) Then
     GoTo Error
    End If
    'Set the number of measurements
    If CheckError(niDMM_ConfigureMultiPoint(vi, 1,
                    Val(numOfMeasurements.Text), _
                    SampleTrigSource, SampleInterval)) Then
    GoTo Error
    End If
    'Read the measurements
    If CheckError(niDMM_ReadMultiPoint(vi, timeLimit, _
                    CInt(numOfMeasurements.Text), _
                    measurementArray(1), numelements)) Then
    GoTo Error
    End If
'Format and display measurements
```

```
        For i = 1 To CInt(numOfMeasurements.Text)
            measurements.Text = measurements.Text & _
                    Format(measurementArray(i), "##0.000000") & _
                    vbCrLf
        Next i
    'Abort the acquisition
    If CheckError(niDMM_Abort(vi)) Then
    GoTo Error
    End If
    'Always close the instrument, even if an error occured in
    'niDMM_SimpleAcquisition
    niDMM_close (vi)
Error:
    'Enable acquire button
    Acquire.Enabled = True
End Sub
```

Handshaking Acquisition

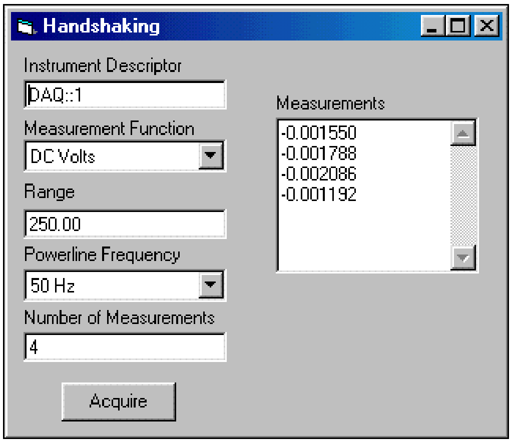


Figure A-5. Handshaking Front Panel

```
Private Sub Acquire_Click()
Private Sub Acquire_Click()
    Dim vi As ViSession
    Dim measurementArray() As ViReal64
    Dim i As ViInt32
    ReDim measurementArray(1 To Val(numOfMeasurements.Text))
```

```
measurements.Text = ""
Dim resolution As ViReal64
Dim timeLimit As ViInt32
Dim numPointsRead As ViInt32
'Set these values to desired defaults.
resolution = 1#
timeLimit = 10000
'Disable Acquire button until acquistion complete
Acquire.Enabled = False
'Open a session to the instrument
If CheckError(niDMM_init(instrumentDescriptor.Text, VI_TRUE, VI_TRUE,
                vi)) Then
    GoTo Error
End If
'Set the powerline frequency
If CheckError(niDMM_ConfigurePowerlineFrequency(vi,
                CtlVal(powerlineFreq))) Then
 GoTo Error
End If
'Configure the acquistion
 If CheckError(niDMM_ConfigureMeasurement(vi, CtlVal(measFunction), _
                Val(range.Text), resolution)) Then
 GoTo Error
End If
'Set the trigger
If CheckError(niDMM_ConfigureTrigger(vi, NIDMM_VAL_IMMEDIATE,  0#)) Then
GoTo Error
End If
'Configure measurement complte destination to external
If CheckError(niDMM_ConfigureMeasCompleteDest(vi,  NIDMM_VAL_EXTERNAL))
                Then
GoTo Error
End If
'Set the number of measurements
If CheckError(niDMM_ConfigureMultiPoint(vi, 1, numOfMeasurements, _
                NIDMM_VAL_EXTERNAL, 0#)) Then
GoTo Error
End If
'Acquire the measurements
If CheckError(niDMM_ReadMultiPoint(vi, timeLimit, numOfMeasurements, _
                measurementArray(1), numPointsRead)) Then
    GoTo Error
    End If
        'Format and display measurements
        For i = 1 To CInt(numOfMeasurements.Text)
            measurements.Text = measurements.Text & _
                              Format(measurementArray(i), "##0.000000") & _
                                vbCrLf
        Next i
    'Always close the instrument, even if an error occured in
    'niDMM_SimpleAcquisition
    niDMM_close (vi)
Error:
    'Enable acquire button
    Acquire.Enabled = True
End Sub
```

B Technical Support Resources

This appendix describes the comprehensive resources available to you in the Technical Support section of the National Instruments Web site and provides technical support telephone numbers for you to use if you have trouble connecting to our Web site or if you do not have internet access.

NI Web Support

To provide you with immediate answers and solutions 24 hours a day, 365 days a year, National Instruments maintains extensive online technical support resources. They are available to you at no cost, are updated daily, and can be found in the Technical Support section of our Web site at `www.ni.com/support`

Online Problem-Solving and Diagnostic Resources

- **KnowledgeBase**—A searchable database containing thousands of frequently asked questions (FAQs) and their corresponding answers or solutions, including special sections devoted to our newest products. The database is updated daily in response to new customer experiences and feedback.
- **Troubleshooting Wizards**—Step-by-step guides lead you through common problems and answer questions about our entire product line. Wizards include screen shots that illustrate the steps being described and provide detailed information ranging from simple getting started instructions to advanced topics.
- **Product Manuals**—A comprehensive, searchable library of the latest editions of National Instruments hardware and software product manuals.
- **Hardware Reference Database**—A searchable database containing brief hardware descriptions, mechanical drawings, and helpful images of jumper settings and connector pinouts.
- **Application Notes**—A library with more than 100 short papers addressing specific topics such as creating and calling DLLs, developing your own instrument driver software, and porting applications between platforms and operating systems.

Software-Related Resources

- **Instrument Driver Network**—A library with hundreds of instrument drivers for control of standalone instruments via GPIB, VXI, or serial interfaces. You also can submit a request for a particular instrument driver if it does not already appear in the library.
- **Example Programs Database**—A database with numerous, non-shipping example programs for National Instruments programming environments. You can use them to complement the example programs that are already included with National Instruments products.
- **Software Library**—A library with updates and patches to application software, links to the latest versions of driver software for National Instruments hardware products, and utility routines.

Worldwide Support

National Instruments has offices located around the globe. Many branch offices maintain a Web site to provide information on local services. You can access these Web sites from `www.ni.com/worldwide`

If you have trouble connecting to our Web site, please contact your local National Instruments office or the source from which you purchased your National Instruments product(s) to obtain support.

For telephone support in the United States, dial 512 795 8248. For telephone support outside the United States, contact your local branch office:

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20,
Brazil 011 284 5011, Canada (Calgary) 403 274 9391,
Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521,
China 0755 3904939, Denmark 45 76 26 00, Finland 09 725 725 11,
France 01 48 14 24 24, Germany 089 741 31 30, Greece 30 1 42 96 427,
Hong Kong 2645 3186, India 91805275406, Israel 03 6120092,
Italy 02 413091, Japan 03 5472 2970, Korea 02 596 7456,
Mexico (D.F.) 5 280 7625, Mexico (Monterrey) 8 357 7695,
Netherlands 0348 433466, New Zealand 09 914 0488,
Norway 32 27 73 00, Poland 0 22 528 94 06, Portugal 351 1 726 9011,
Singapore 2265886, Spain 91 640 0085, Sweden 08 587 895 00,
Switzerland 056 200 51 51, Taiwan 02 2528 7227,
United Kingdom 01635 523545

Glossary

Prefix	Meaning	Value
p-	pico-	10^{-12}
n-	nano-	10^{-9}
μ-	micro-	10^{-6}
m-	milli-	10^{-3}
k-	kilo-	10^{3}
M-	mega-	10^{6}
G-	giga-	10^{9}

Numbers/Symbols

°	degrees
>	greater than
≥	greater than or equal to
<	less than
–	negative of, or minus
Ω	ohms
%	percent
±	plus or minus
+	positive of, or plus

A

AC	alternating current

C

C	Celsius
CJC	cold junction compensation
CMOS	complementary metal-oxide semiconductor
CMRR	common-mode rejection ratio

D

DC	direct current
DMM	digital multimeter
driver	software that controls a specific hardware device such as a DAQ device or GPIB interface device

E

EEPROM	electrically erased programmable read-only memory
EXTCLK	External Clock signal

F

fc	cutoff frequency
Fext	external frequency
firmware	software stored on chips that can store data without electricity, such as EEPROM chips

G

Gs	gain

H

handle	a unique variable you use to refer to a window or other interface element in C programming
hex	hexadecimal
Hz	hertz

I

in	inch
IN+	positive input channel signal
IN–	negative input channel signal
I/O	input/output
INTR*	Interrupt signal

M

max	maximum
MB	megabytes
min	minutes/minimum
MIO	multifunction I/O
MISO	Master-In-Slave-Out signal
MOSI	Master-Out-Slave-In signal

N

NP	no pull-up
NRSE	nonreferenced single-ended

O

OUTCLK	Output Clock signal
OUTPUT	Output signal
OUTPUT REF	Output Reference signal

P

P	pull-up
ppm	parts per million

R

rms	root mean square
RSVD	Reserved signal/bit
RTSI	Real-Time System Integration

S

s	seconds
S/s	samples per second—used to express the rate at which a DAQ board samples an analog signal
SCANCLK	Scan Clock Signal
SCXI	Signal Conditioning eXtensions for Instrumentation (bus)
SERCLK	Serial Clock signal
SERDATIN	Serial Data In signal
SERDATOUT	Serial Data Out signal
SLOT0SEL	Slot 0 Select signal
SPICLK	Serial Peripheral Interface Clock signal

T

THD	total harmonic distortion
thermistor	A semiconductor sensor that exhibits a repeatable change in electrical resistance as a function of temperature. Most thermistors exhibit a negative temperature coefficient.

V

V	volts
VI	Virtual Instrument—a program in the graphical programming language G; so called because it models the appearance and function of a physical instrument
VISA	Virtual Instrument Software Architecture—a single interface library for controlling GPIB, VXI, RS-232, and other types of instruments
Vrms	volts, root mean square
VMC	voltmeter complete
VXI	VME eXtensions for Instrumentation (bus)
VXI*plug&play* Systems Alliance	A group of VXI developers dedicated to making VXI devices as easy to use as possible, primarily by simplifying software development.

Index