2005

Integration Of Real-Time Experiments With Internet Access

Pavan Talakala
University of Central Florida

INTEGRATION OF REAL – TIME EXPERIMENTS WITH INTERNET ACCESS

by

PAVAN TALAKALA
B.E. Osmania University, India 2000

A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science
in the Department of Computer Engineering
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Spring Term
2005

ABSTRACT

The revolution of Internet – enabled instrumentation is emerging as a revolution in Measurement and Automation. New standards are being developed for transmitting data and connecting instruments to the Internet. The main purpose of this thesis is to design and develop a system to Integrate various Real – time experiments and be able to monitor and control them over Internet using LabVIEW. LabVIEW is a graphical programming package capable of data acquisition, data analysis, data presentation and real time remote control.

In this thesis a Real – time system is developed which integrates several real time experiments and remote control access over Internet is provided using LabVIEW. The latest remote panel technology is used to provide the remote control access. There are four Real – time instrument experiments developed in this thesis project. They are Automatic Mixing Controller, The Digital Storage Oscilloscope from Gould Instruments, Temperature Controller and an Electronic Recording Rain/Precipitation Gauge. These four instruments are connected and communicated from the main computer by external Data Acquisition Board (DAQ) for the Automatic Mixing Controller, By General Purpose Interface Board (GPIB) for the Oscilloscope, by Data Acquisition board (DAQ) for the Temperature controller and by Serial port for the Electronic Recording Rain/Precipitation Gauge. A system is developed to integrate all the applications listed above into one application and are monitored and controlled remotely over Internet using LabVIEW.

ACKNOWLEDGMENTS

First and foremost, I would like to thank Dr.Janusz Zalewski, my advisor who has given me constant guidance dedicated support through out my thesis work. Without his encouragement and advice this thesis would never have been completed

I would like to thank Dr.Samuel Richie and Dr.Fernando Gonzalez for agreeing to be on thesis committee and giving me valuable suggestions while working on my thesis.

I would like to thank my friends for their understanding and support over the years and for proof reading this document.

Finally, I want to express my deepest regards to my parents and family members for their support, motivation and inspiration throughout my education and without whose blessings and best wishes I would not have been able to reach my goals.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

1.1 Computer Based Measurement And Automation

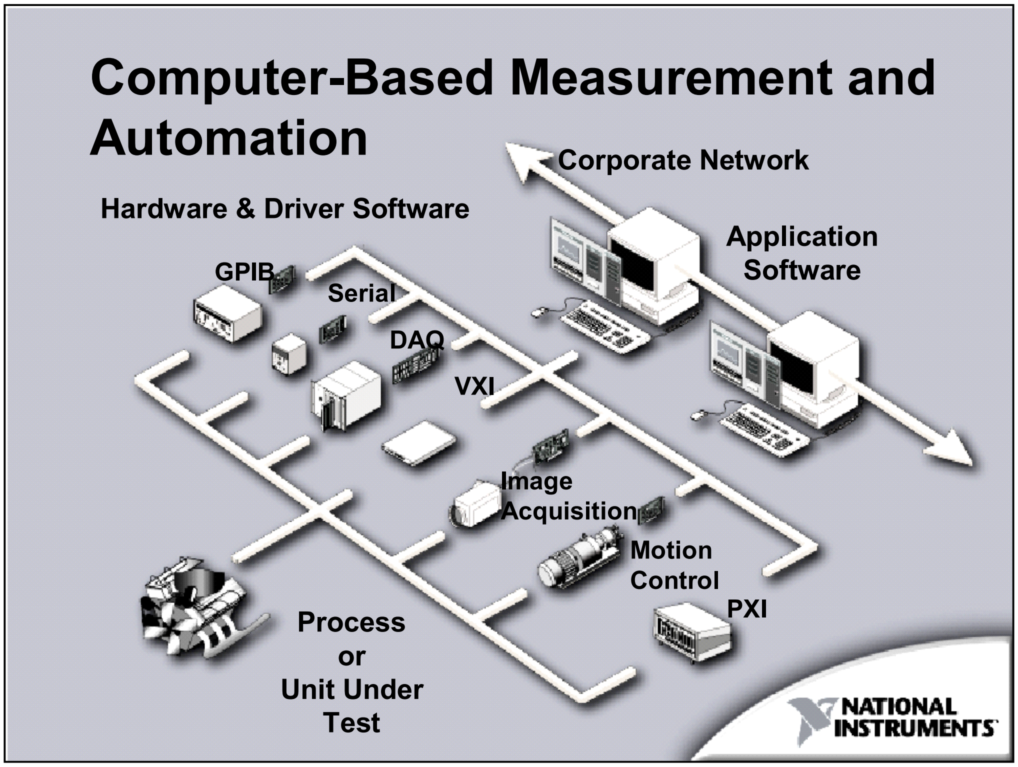
For more than 25 years, National Instruments has revolutionized the way scientists and engineers' work by leveraging the personal computer and its related technologies. Around the world, National Instruments is changing the way measurements are made by providing the software, hardware, and technologies necessary to transform personal computers into powerful computer-based and networked measurement and automation systems. [1] Whether testing cars at Honda in Japan, validating cardiac defibrillators at Tecktronics in Australia, or increasing test efficiency of phone lines at British Telecom in the UK, companies using NI products can achieve their goals - faster, better.

Laboratory Virtual Instrument Engineering Workbench (LabVIEW) is a Programming environment in which one can create programs with graphics. National Instruments (NI) Corporations developed LabVIEW in 1986. [1] Thousands of successful engineers, scientists, and technicians use LabVIEW to create solutions for their demanding application needs. LabVIEW is a revolutionary graphical programming development environment based on the Graphical programming language for data acquisition and control, data analysis, and data presentation. LabVIEW gives the flexibility of a powerful programming language without the associated difficulty and complexity because its graphical programming methodology is inherently intuitive to scientists and engineers. LabVIEW is used to program instrumentation systems in a faster way without sacrificing performance.

National Instruments provides with several types of computer-based measurement and automation hardware and software tools. A developer can make all the required measurements from PCs using different combinations of I/O that are appropriate for an application:

- Rack-n-stack GPIB instruments (NI makes the GPIB host interface)
- Serial instruments (NI makes RS-232 & RS-485 host interfaces)
- Data acquisition boards w/ & w/o signal conditioning
- VXI instruments (NI makes controllers, chassis, DAQ boards)
- Image acquisition
- Motion control
- PXI / Compact PCI

The PC's making these measurements are typically networked together, and are running software that makes taking measurements fast and easy. [2]



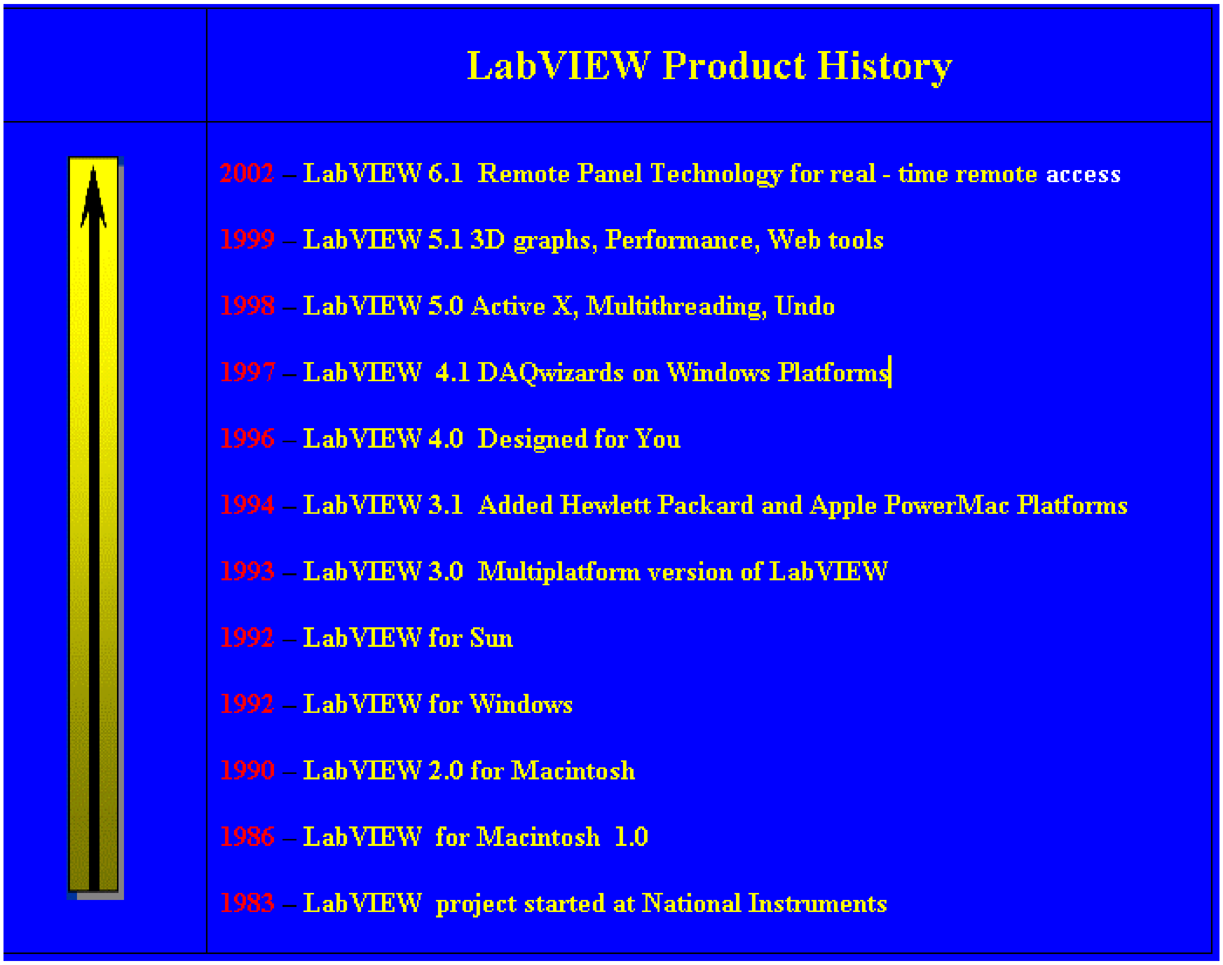
Computer-Based Measurement and Automation

Figure 1.1: Computer – Based Measurement and Automation

LabVIEW is one member of National Instruments family of software tools and languages [1]. LabVIEW is a graphical programming language that is fast and easy to write. Virtual Bench is a set of soft front panel instruments that resemble bench top instruments—they use any NI DAQ [3] or computer-based instrument. Measure is used for inserting data directly into cells of an Excel spreadsheet.

LabVIEW is a programming language just like other programming languages, such as C, Basic, or Pascal, but LabVIEW is higher-level. In text-based programming languages, a developer will be concerned about the code and must pay close attention to the syntax (commas, periods, semicolons, square brackets, curly brackets, round brackets). LabVIEW is much higher-level language, it uses icons to represent subroutines, and the developer has to wire these icons together in order to define the flow of data through the program. It is sort of like flow-charting the code as it is written - and the net effect is that the program can be written in a lot less time than it would take in a text-based programming language [4].

LabVIEW has been in existence since October of 1986. It has gone through many revisions and feature enhancements over the years, and today it is the most popular test and measurement programming language. It has all the depth and breadth that any programming language has. [2]



LabVIEW Product History

2002 – LabVIEW 6.1 Remote Panel Technology for real - time remote access

1999 – LabVIEW 5.1 3D graphs, Performance, Web tools

1998 – LabVIEW 5.0 Active X, Multithreading, Undo

1997 – LabVIEW 4.1 DAQwizards on Windows Platforms

1996 – LabVIEW 4.0 Designed for You

1994 – LabVIEW 3.1 Added Hewlett Packard and Apple PowerMac Platforms

1993 – LabVIEW 3.0 Multiplatform version of LabVIEW

1992 – LabVIEW for Sun

1992 – LabVIEW for Windows

1990 – LabVIEW 2.0 for Macintosh

1986 – LabVIEW for Macintosh 1.0

1983 – LabVIEW project started at National Instruments

Figure 1.2: LabVIEW Product History

LabVIEW programs are called “Virtual Instruments” or “VIs” for short. They are modeled after traditional bench top instruments, which have a front panel for the instrument’s inputs and outputs, and underneath the instrument’s front panel is the hardware of the instrument.

LabVIEW Virtual Instruments are similar—they have a Front Panel for the routine inputs and outputs, and underneath the Front Panel is the graphical source code area, which we call the “Block Diagram.”

LabVIEW is technically what is known as a dataflow programming language. Data flows from a source of data to one or more sinks of data—and it propagates through the program in this fashion. LabVIEW lends itself well to modular programming techniques—as a developer can write his own subVIs, and use them as subroutines to a higher-level program. LabVIEW is a multi-platform programming language, which means that any piece of LabVIEW source code written on one platform, can be brought to any other of the supported platforms, and recompiled. LabVIEW is thus portable across the various platforms.

Today, the Web is an essential part of the way a business operates. The Web can be used to gain visibility, share information, and sell products, and also to improve the ways to design, manufacture, and test products; to decrease design time; and to ensure quality. Just as advancements in PC technologies transformed the way measurements are automated, networks are revolutionizing the fundamental architecture of PC-based measurement solutions. Some have proposed that network technologies are ushering in a post-PC era. Networks are not antiquating the PC, but revolutionizing the PC platform. The basic components found in a PC platform, such as I/O, processors, memory and storage, and displays are still the building blocks, but they no longer need to be packaged as a self-contained unit. With networking technologies, the developer can distribute these components to the location most appropriate for the required application. The platform is fundamentally the same, but the deployment capabilities of the platform are enhanced radically. Using network technologies in the measurement solutions, one can perform I/O on the production floor, deploy additional processing power for in-depth analysis in the control center, log and store post analysis information to a corporate database, and display key information to clients around the world via a Web browser. The essential tool that is required to tie all of these pieces together is the software. LabVIEW provides a platform for designing the test system to take advantage of all of the latest technologies while providing an environment still focused on r developing the application quickly. National Instruments LabVIEW focuses on solving measurement and automation needs by incorporating commercial technologies and balancing powerful ease-of-use functionality.

LabVIEW provides the most productive development environment for empowering scientists and engineers to develop their own solutions. Any LabVIEW application, without any additional development time, can be turned into a remote application accessed via a Web browser. High-level functions are available to generate Web reports through Microsoft Excel or Word with out the necessity to design and develop the low-level interfaces. Live data sharing can be enabled for individual user interface items simply by right-clicking the item and placing a checkmark in a checkbox. These technologies not only save valuable development time, but also help to take advantage of the economies of scale that the Web provides. Many technologies are available for sharing information and data on the Web. For convenience and clarity Labview applications are classified into four main types [2]

- Publishing Data
- Sharing Data

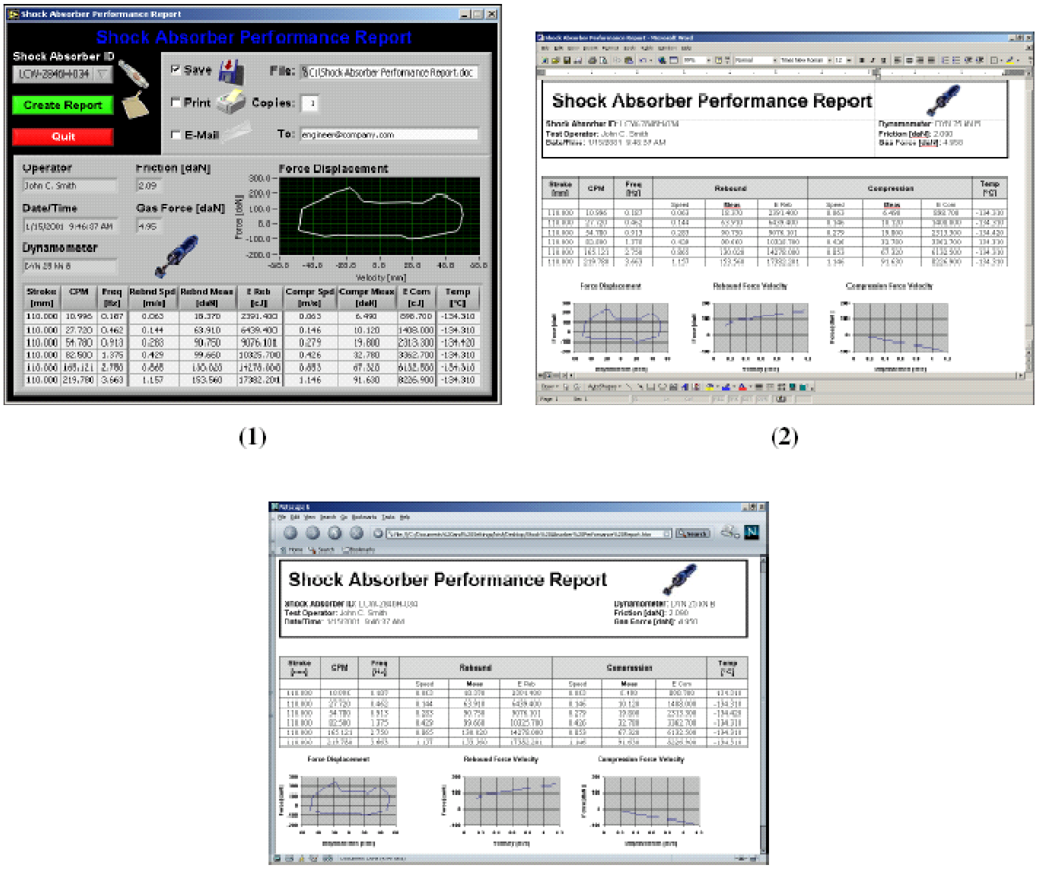
- Remote Control
- Distributed Execution

1.2 Publishing Data

Generates a static Web report of test results that can be shared with others. This method is the electronic version of the traditional printed report but has the advantage of easy access through a standard Web browser.

The use of the Web for distributing information from invoices of new cars, to e-mail alerts of airline specials, to continuously updated research and stock quotes has become everyday business. In the same way the Web provides this daily information, it is the ideal medium for sharing measurement results with others. One of the simplest ways to share information across a network or the Internet is by publishing a report or summary. These reports help disseminate vital information quickly to various groups in the company in a form that can easily be accessed using Web browsers. With the built-in Web Server in LabVIEW, it is possible to publish the front panel of the application without adding any development time to project. After the Web Server is enabled, LabVIEW generates front panel images that can be accessed from any Web browser.

Report generation functions to publish reports in HTML format, ready for publishing to the Web can be used to extend the reporting capability beyond publishing an image of the front panel, such as creating a report that includes tables, lists, operator information, dates, and times along with the graphs and analysis on the front panel. These functions professionally document the results of an application quickly and easily by adding images, front panels, bulleted lists, and tables, thereby making it easy to further integrate the applications on the Web. Often, a developer might want to generate a report using a standard application such as Microsoft Word or Excel. The Report Generation Toolkit for Microsoft Office can be added to LabVIEW to provide high-level functions that make creating these reports fast and easy. It is not necessary to understand the hierarchical ActiveX interface to interact with Microsoft Word or Excel, because these high-level functions incorporate the most common tools required to generate professional reports. Beyond the ability to create the reports for internal use, the toolkit also has functions to generate HTML from Microsoft Word or Excel required to share the results of the report easily throughout the world.



Shock Absorber Performance Report

☑ Save File: C:\Shock Absorber Performance Report.doc

□ Print Copies: 1

□ E-Mail To: engineer@company.com

Operator: John C. Smith

Friction [daN]: 2.09

Date/Time: 1/15/2001 9:46:37 AM

Gas Force [daN]: 4.95

Dynamometer: DYN 25 kN 8

Stroke [mm]	CPM	Freq [Hz]	Rebnd Spd [m/s]	Rebnd Meas [daN]	E Reb [cJ]	Compr Spd [m/s]	Compr Meas [daN]	E Com [cJ]	Temp [°C]
110.000	10.995	0.187	0.063	18.370	2391.400	0.063	6.490	898.700	-134.310
110.000	27.720	0.462	0.144	63.910	6439.400	0.146	10.120	1408.000	-134.310
110.000	54.780	0.913	0.283	90.750	9076.101	0.279	19.800	2313.300	-134.420
110.000	82.500	1.375	0.429	99.660	10325.700	0.426	32.780	3362.700	-134.310
110.000	165.121	2.750	0.568	130.020	14278.000	0.853	67.320	6132.500	-134.310
110.000	219.780	3.663	1.157	153.560	17382.201	1.146	91.630	8226.900	-134.310

(1)

Shock Absorber Performance Report

(2)

Shock Absorber Performance Report

(3)

Figure 1.3: Publishing Data using LabVIEW

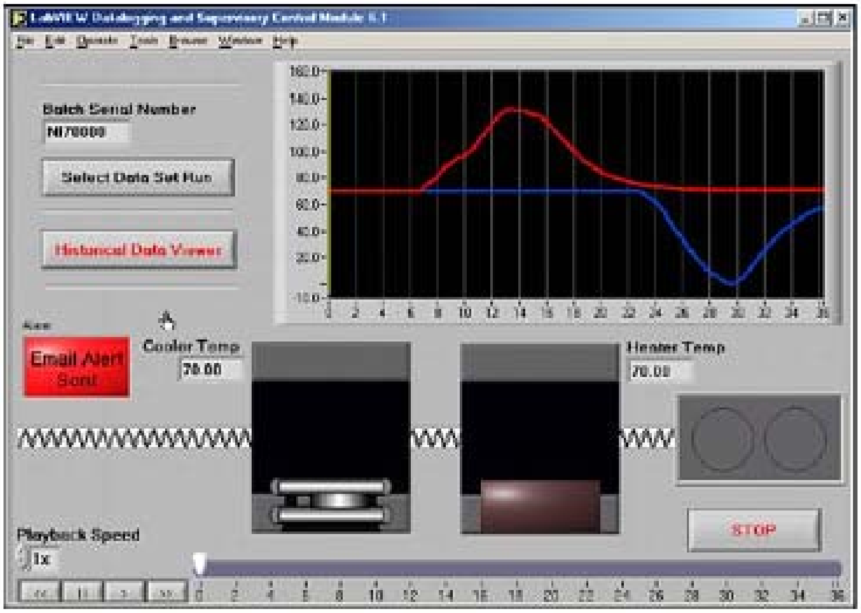
In the above figure use of LabVIEW to generate a report in Microsoft Word (1), and then create HTML (2) to display the data on the Web (3) are displayed. In addition to Word and Excel, some corporations use databases to store the data and use a Web interface for viewing reports. For quality control purposes, manufacturers might prefer to use a database that the rest of the company can easily access through a Web interface. The Database Connectivity Toolkit provides a common ADO interface for easy transfer of data and information to and from a database. Many databases have a front end that is accessible and searchable through a Web interface. Anyone who has access can then search and view the data in a Web browser. LabVIEW Internet Toolkit can also be used to do FTP, to transfer data files on the web, when a large amount of information is gathered and is preferred to place these files on an FTP site where it can be shared easily. FTP sites are a good way to share larger files that might not be appropriate to send in e-mail and around which it might be difficult to create HTML. As the Web has become the preferred medium for disseminating test data, because with it one can easily create reports and send updates. With the built-in tools in LabVIEW, it is easy to take advantage of these technologies.

1.3 **Sharing Data**

Expands the concept of publishing data to include transferring the actual data among computers where a user can perform different analyses on that data, depending on the needs. Some applications might require streaming the actual data for additional processing, storage, or monitoring. For example, out-of-bound parameters can be updated while the test is still progressing.

A report offers a static view of compiled information. While this may deliver an easy-to-read summary of a completed test, it does not provide an easy way to access the data that created the report. In many applications, real-time access to acquired data is needed to control or monitor a process or perform a test across a network. The results of one measurement or automation process can be passed directly to the next process. In this type of application, it is sharing the actual information that is important; as opposed to the data reporting applications where sharing static views, such as a report or a Web page, were sufficient. In this kind of application, manufacturing might share quality data with the R&D organization, which could then analyze the data directly instead of just looking at a report that manufacturing generated. For data sharing, XML is quickly becoming a standard way to transfer, in a text-readable fashion, data that can easily be displayed on the Web. [1] Because of the universal XML standard, it is possible to generate a Web report that has a defined set of data. Because the data is readily accessible, applications can download any XML document, parse the data, and perform custom analyses. LabVIEW provides built-in functions that can be used to create or read XML documents.

Another method of sharing data directly with other parts of the organization is through DataSocket. Using DataSocket, a National Instruments technology built on top of TCP/IP, can quickly stream data between computers and applications. DataSocket implementation requires no extra development time because it enables a graph or other user interface item to stream the data it is displaying over the network. Because DataSocket also is implemented as an ActiveX control, a Java Bean, and a component of Measurement Studio for C/C++ and Visual Basic development, the developer can incorporate this technology in many other applications. To subscribe to the DataSocket Server item containing the data, these applications use a URL address to begin receiving data and any updates that are sent. Using DataSocket, Web page can be generated to show updates of quality information from a manufacturing floor, changing properties of materials during an ongoing test, or even updates of the weather.



Batch Serial Number
NI70000

Cooler Temp
70.00

Heater Temp
70.00

Figure 1.4: Data sharing using LabVIEW

This application in the above figure sends an e-mail alert when specified limits are exceeded. With the Internet Toolkit, LabVIEW can be used to send e-mail alerts. Electronic notifications can then be created for operators that use pagers or mobile phones to receive alerts from the production area when certain process values exceed established limits. The operator gets updated as the test runs, and his time is freed up to spend on more productive tasks.

1.4 **Remote Control**

Expands the concept of sharing data to include enabling another computer to connect to the experiment and control that experiment remotely. For many applications, the test might be in a harsh environment or run overnight where it is inconvenient for someone to be at the terminal continuously.

With remote control, anyone can control the execution of a system from another computer in a remote location. One instance where this would be useful is with a system in a harsh environment where a person will have limited access and might want a way to perform remote monitoring, control, even run diagnostics, while the system itself is dedicated to running acquisition and control. Sometimes tests are designed to run for long periods of time, but at certain intervals, parameters must be changed or other tests begun. Because the operator of the test does not want to have to drive back to work on the weekend or at night just to make these small modifications to the tests, the operator logs onto the network from home, connects to the test, and makes those changes just as if he or she were on site.

With LabVIEW, it is possible to achieve remote execution for every application developed with just a few clicks. Without any additional programming, any LabVIEW application for remote control through a common Web browser can be configured. The user simply points the Web browser to the Web page associated with the application. The remote user can access fully the user interface that appears in the Web browser. The acquisition still occurs on the host computer, but the remote user has complete control of the application. Other users also can point their Web browser to the same URL to view the test. To avoid confusion, only one client can control the application at a time, but that control can pass easily among the various clients at run-time. At any time during this process, the host computer can take control of the application away from any of the remote clients who are in control.

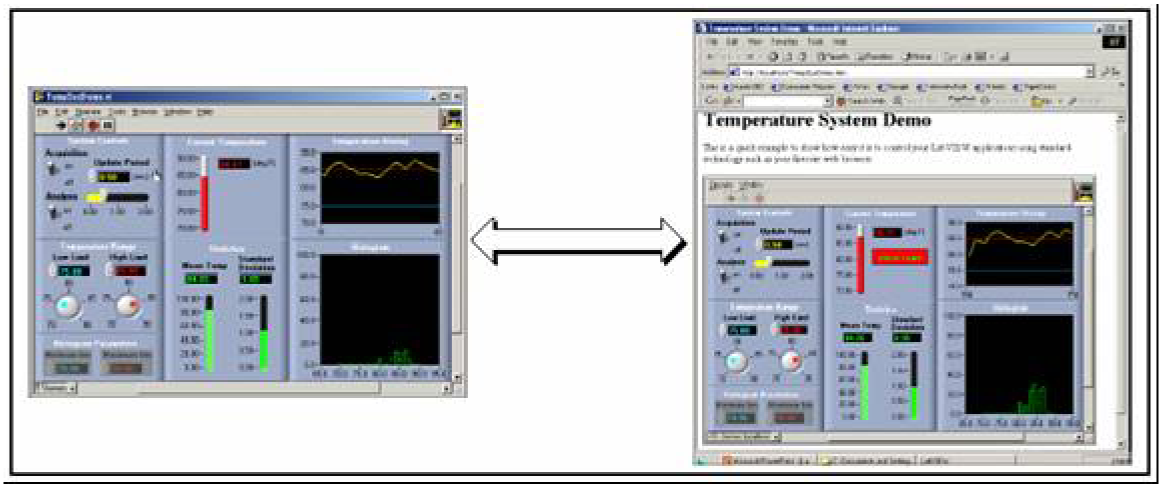


Figure 1.5 : Running Both in LabVIEW and Embedded in a Web Browser

Built-in LabVIEW tools can be used to take advantage of powerful networking technologies for Web-based applications through common Web browser interfaces, as well as use the latest in Windows technologies with the LabVIEW applications. Windows XP introduces Remote Desktop and Remote Assistance, which one can use to help debug deployed systems. After the system is deployed in the field, often it is cost prohibitive for the support staff to visit every site. With Remote Desktop, a support operator can log into the Windows XP machine and act as if he or she were sitting at the desk where that machine is located. With Remote Assistance, the operator can remain in control of the desktop but the support operator can view the desktop on his/her remote machine. At any time, the operator can give up control of the desktop to the support operator and still see which troubleshooting techniques are in use. These new technologies in Windows XP make troubleshooting deployed systems much easier than ever before. As the industry-standard software development tool, LabVIEW takes advantage of these new features and provides additional features that complement them. At times, the Web browser can be used to initiate the measurement or automation application, but not actually control the experiment. In this case, an operator can log in, set certain parameters, and run the application. This can be accomplished using the Remote Panel technology. This program normally builds HTML dynamically by accessing other data sources such as a database. As part of the HTTP request, the browser can send to the server the parameters to use in the application.

1.5 Distributed Execution

Combines several of the concepts, by developing a system architecture that shares the acquisition and analysis of the test among several computers. Systems of the future will consist of measurement nodes that can transfer data between computers, so different parts of the test can run at different places, and the data still can be correlated and used to control other hardware items.

The advantage of networking can be used to distribute execution, dedicating certain machines for the acquisition and control while offloading analysis and presentation on other systems. Each system is optimized to perform specific functions. Because the user can share data among the distributed components and each component accomplishes a unique task, this network functions as a complete system. With network access to various measurement nodes, one can develop software that uses each computer to complete a portion of the application. A test can have several acquisition nodes; each sending data back to a main computer or cluster of computers where the analysis is performed and reports are generated and sent to the Web. For certain test and control applications, an embedded and reliable solution might be needed. For these applications, LabVIEW Real-Time Module can be downloaded to a PXI instrumentation system or a Field Point distributed I/O module. LabVIEW runs on a real-time operating system on these systems, but still can be accessed from a host computer through Ethernet. From this computer, either remote front panels in LabVIEW can be used to control the application, or can use DataSocket to transfer the data directly from the real-time target to a host computer, which could then act as a server to further disseminate that data to other computers. [1]

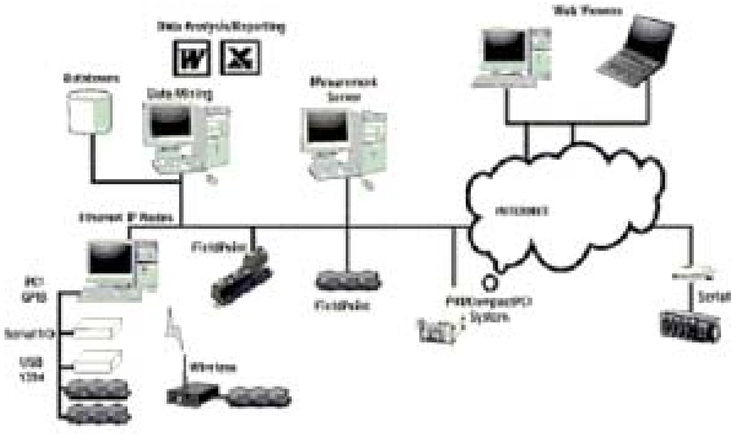


Figure 1.6 : Distributed Execution using LabVIEW

Using LabVIEW and the Web, different systems work together in one application. An example application is a structural test system measuring the vibration and harmonics of a bridge design. One node can be set up with a camera to monitor the testing of the bridge, another node to measure parameters such as wind direction and speed, temperature, and humidity. Finally, a node can be set up to measure the load, strain, and displacement on certain areas of the bridge. All the data will be sent back to a main computer that correlates the data, analyzes the data, and displays the results of the test on a Web page.

Each of these nodes would need to be running autonomously, acquiring data and sending it onto other computers to correlate the data and create reports. Using the LabVIEW Real-Time Module and PXI, each of the measurement nodes could become an embedded, reliable, and durable solution. Remote front panels in LabVIEW are extensible to the LabVIEW Real-Time Module, so the user could easily control any of the measurement nodes to modify parameters of the test. The original creation and testing of the code is also completed using a Windows operating system and then downloaded to the measurement node. So, the user can make major modifications to the test and download them to the embedded target without visiting the site. Next, one of the live data-sharing techniques could be used to transfer the data to another cluster of computers that would correlate and analyze the data. Finally, an Internet server could be set up to share the Web reports and analysis with others around the country.

The Web is changing the way the measurements are taken and the results are distributed. Many different options exist for publishing reports, sharing data, and remotely controlling applications. LabVIEW incorporates the latest Web technologies, to take advantage of the power of the Web without having to become experts in any Web technologies. With LabVIEW, the user can incorporate the Web into many different aspects of the application, from just sharing the data with colleagues to creating a unique, powerful distributed application combining different measurement nodes and multiple computers together into one measurement and control system. With LabVIEW, applications can be integrated easily into the existing corporate networking infrastructure so that the corporation can better share data and increase the productivity of those performing the measurements.

2. LITERATURE REVIEW

2.1 Overview of LabVIEW

LabVIEW is a revolutionary graphical development environment with built-in functionality for data acquisition, instrument control, measurement analysis, and data presentation. LabVIEW gives the flexibility of a powerful programming language without the complexity of traditional development environments. LabVIEW delivers extensive acquisition, analysis, and presentation capabilities in a single environment, to seamlessly develop a complete solution on the platform of the users choice. Unlike general purpose programming languages, National Instruments (NI) LabVIEW provides functionality specifically tailored to the needs of measurement, control, and automation applications, accelerating the development process. From built-in analysis capabilities to connectivity with a wide variety of I/O, LabVIEW delivers what engineers and scientists need to quickly build test and measurement, data acquisition, embedded control, scientific research, and process monitoring systems. The LabVIEW graphical development environment is equipped with powerful tools to create applications without writing any lines of text-based code. With LabVIEW, a user can drag and drop pre-built objects to quickly and simply create user interfaces for an application. [1] Then, the developer can specify system functionality by assembling block diagrams -- a natural design notation for scientists and engineers.

LabVIEW delivers seamless connectivity with measurement hardware, to quickly configure and use virtually any measurement device, including everything from stand-alone instruments to plug-in data acquisition devices, motion controllers, image acquisition systems, and programmable logic controllers (PLC). LabVIEW works with more than 1,000 instrument libraries from hundreds of vendors [2].

2.2 Open Connectivity with Other Applications

With LabVIEW, anyone can connect to other applications and share data through ActiveX, the Web, DLLs, shared libraries, SQL, TCP/IP, XML, OPC, wireless communication and other methods. LabVIEW's open connectivity makes it possible to create open, flexible applications that can communicate with other applications across an organization. [2]

In many applications, execution speed is critical. With a built-in compiler that generates optimized code, LabVIEW applications deliver execution speeds comparable to compiled C programs. With LabVIEW, it is possible to develop systems that meet even the most demanding performance requirements across a variety of platforms including Windows, Macintosh, UNIX, or real-time systems.

LabVIEW is a graphical programming language that uses icons instead of lines of text to create applications. In contrast to text-based programming languages, where instructions determine program execution, LabVIEW uses dataflow programming, where the flow of data determines execution. In LabVIEW, a user interface is built by using a set of tools and objects.

The user interface is known as the front panel. Then the code is added using graphical representations of functions to control the front panel objects. The block diagram contains this code. In some ways, the block diagram resembles a flowchart. Several add-on software toolsets can be purchased for developing specialized applications. All the toolsets integrate seamlessly in LabVIEW.

LabVIEW is integrated fully for communication with hardware such as GPIB, VXI, PXI, RS-232, RS-485, and data acquisition control, vision, and motion control devices. LabVIEW also has built-in features for connecting an application to the Internet using the LabVIEW web server and software standards such as TCP/IP networking and ActiveX [2].

LabVIEW can be used to create 32-bit compiled applications that give fast execution speeds needed for custom data acquisition, test, measurement, and control solutions. Stand-alone executables and shared libraries, like DLLs, can also be created because LabVIEW is a true 32-bit compiler [2].

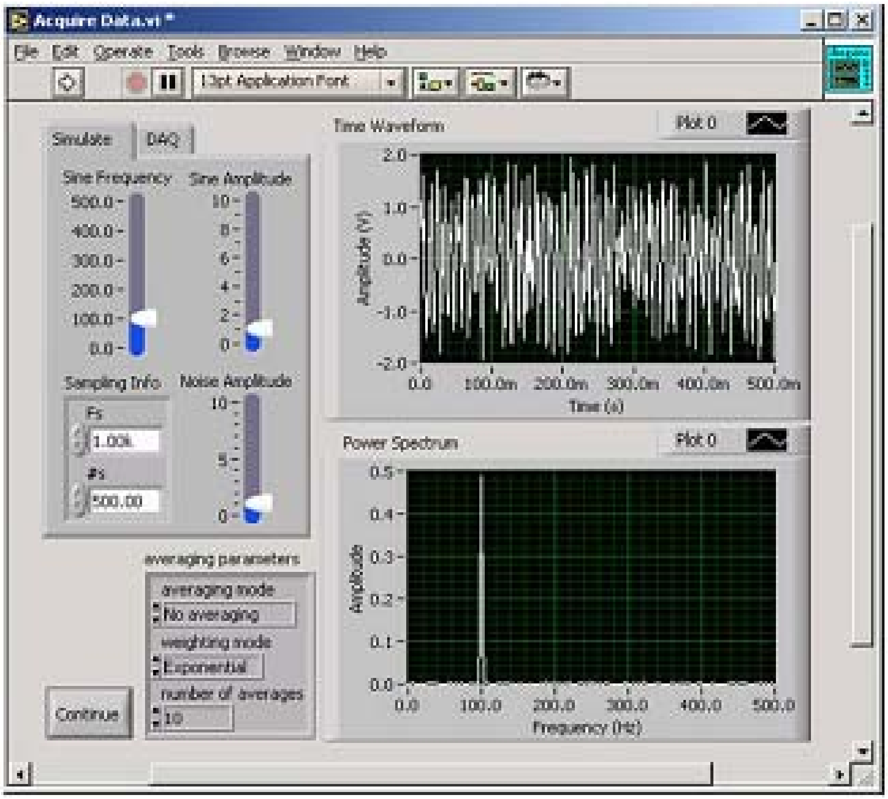
LabVIEW contains comprehensive libraries for data collection, analysis, presentation, and storage. LabVIEW also includes traditional program development tools. LabVIEW has different options to set breakpoints, animate program execution, and single-step through the program to make debugging and development easier. LabVIEW also provides numerous mechanisms for connecting to external code or software through DLLs, shared libraries, ActiveX, and more. In addition, numerous add-on tools are available for a variety of Application needs.

LabVIEW empowers a user to build his own solutions for scientific and engineering systems. LabVIEW gives the flexibility and performance of a powerful programming language without the associated difficulty and complexity. LabVIEW gives thousands of successful users a faster way to program instrumentation, data acquisition, and control systems. By using LabVIEW to prototype, design, test, and implement instrument systems, the development Engineers can reduce system development time and increase productivity by a factor of 4 to 10. LabVIEW also gives the benefits of a large installed user base, years of product feedback, and powerful add-on tools. [2].

LabVIEW programs are called virtual instruments, or VIs, because their appearance and operation imitate physical instruments, such as oscilloscopes and multi-meters. Every VI uses functions that manipulate input from the user interface or other sources and displays that information or moves it to other files or other computers.

A VI contains the following three components:

- Front panel - Serves as the user interface.
- Block diagram - Contains graphical source code that defines the functionality of a VI.
- Icon and connector pane - Identifies the VI so that it can use the VI in another VI. A VI within another VI is called a subVI. A subVI corresponds to a subroutine in text-based programming languages.



Fs 1.00k

#s 500.00

averaging mode No averaging

weighting mode Exponential

number of averages 10

Figure 2.1: Front panel of a LabVIEW virtual instrument.

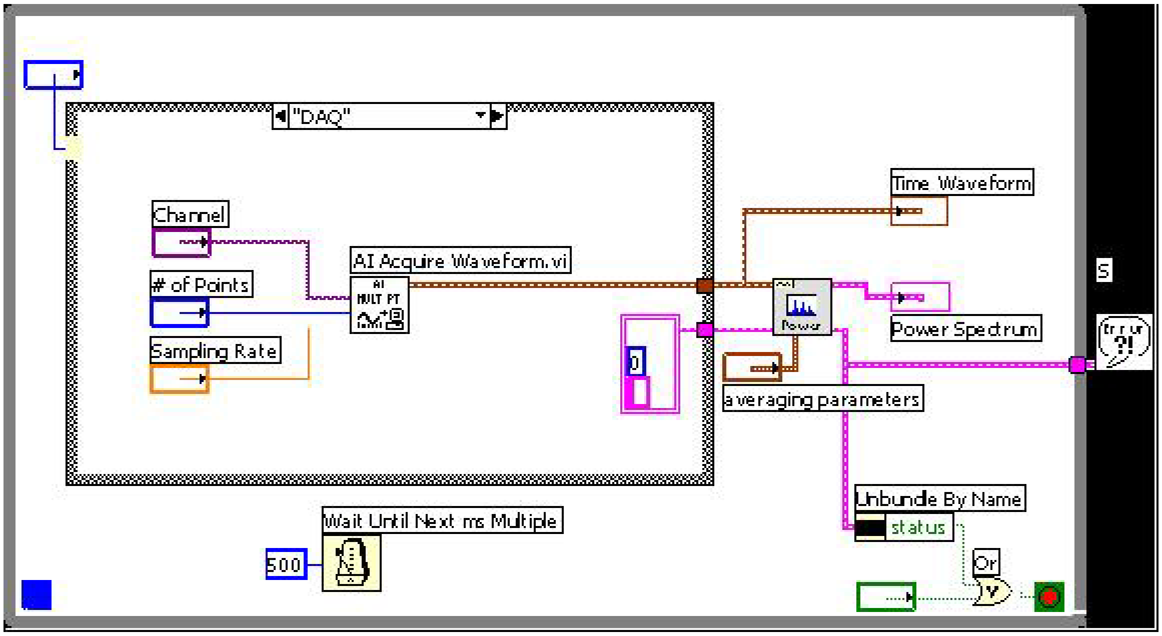


Figure 2.2 : The block diagram of the above virtual instrument.

LabVIEW programs are called virtual instruments (VI) because their appearance and operation imitate actual instruments. However, behind the scenes they are analogous to main programs, functions, and subroutines from popular programming languages like C or BASIC. Hereafter, a LabVIEW program is referred to as a ‘VI’. VI has three main parts:

The front panel is the interactive user interface of a VI, so named because it simulates the front panel of a physical instrument. The front panel can contain knobs, push buttons, graphs, and many other controls (which are user inputs) and indicators (which are program outputs). Input data using a mouse, keyboard, and then view the results produced by the program on the screen.

The block diagram is the VI's source code; constructed in LabVIEW's graphical programming language, G. The block diagram is the actual executable program. The components of a block diagram are lower-level VI’s, built-in functions, constants, and program execution control structures. The programmers draw wires to connect the appropriate objects together to indicate the flow of data between them. Front panel objects have corresponding terminals on the block diagram so data can pass from the user to the program and back to the user. [2]

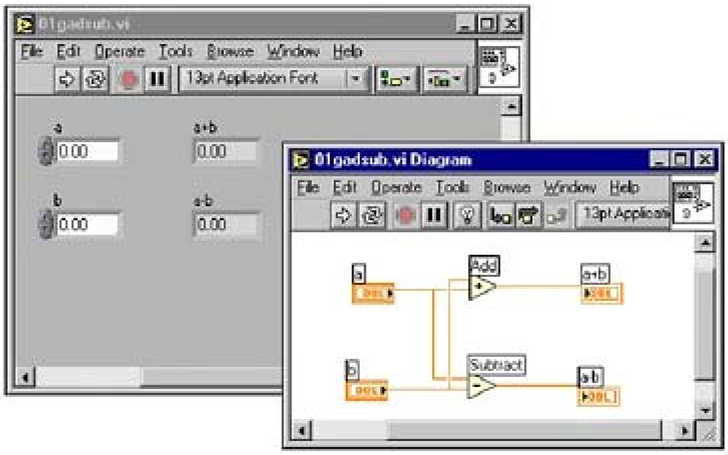


Figure 2.3 : Front Panel and Block Diagram

In order to use a VI as a subroutine in the block diagram of another VI, it must have an icon and a connector. A VI that is used within another VI is called a subVI and is analogous to a subroutine. The icon is a VI's pictorial representation and is used as an object in the block diagram of another VI. A VI's connector is the mechanism used to wire data into the VI from other block diagrams when the VI is used as a subVI. Such like parameters of a subroutine, the connector defines the inputs and outputs of the VI Virtual instruments are hierarchical and modular. The icon can be used as top-level programs or subprograms. With this architecture, LabVIEW promotes the concept of modular programming. First, an application is divided into a series of simple subtasks. Next, build a VI to accomplish each subtask and then combine those VIs on a top-level block diagram to complete the larger task.

Modular programming is a plus because each subVI can be executed by itself, which facilitates debugging. Furthermore, many low-level subVIs often perform tasks common to several applications and can be used independently by each individual application. A few common LabVIEW terms with their conventional programming equivalents are listed below. [1]

LabVIEW	Conventional Language
VI	Program
Function	Function
SubVI	Subroutine, subprogram
Front panel	User interface
Block diagram	Program code
G	C, Pascal, BASIC, etc.

Figure 2.4: LabVIEW Terms and Their Conventional Equivalents

2.3 Other Features of LabVIEW

2.3.1 Portability

As mentioned above, LabVIEW programs are portable across platforms, so it is possible to write a program on a Macintosh and then load and run it on a Windows machine without changing a thing in most applications.

2.3.2 Productivity

One of the initiatives of LabVIEW is to ease the work of professionals and engineers who have no or few programming practices. Therefore, the LabVIEW programming language is very easy to use. Once the user has figured out the data flow of a program, he can easily build a VI by using the rich built-in controls, indicators, functions, device drivers, I/O, and networking protocols. The program can be run without compiling, linking, or building which are required to be done when using other programming environments. LabVIEW have features and instant help to ease the debugging process. The advantages of the graphical features and the easiness of use to build rapid prototypes should be utilized in requirement phases. Instrument drivers, ready-to-run software modules for controlling programmable instruments, are a key to enhancing productivity. Instrument drivers have been a standard part of virtual instrumentation software, with modules for hundreds of instruments from many different instrument manufacturers.

2.3.3 Flexibility

Virtual instrumentation uses cost-effective, multifunction instrumentation hardware and user-friendly, sophisticated software to leverage the power and economies of scale of the computer. The fierce competition and huge investment in the computer industry ensure that virtual instrumentation is both affordable and powerful. Virtual instruments are built and maintained at a fraction of the cost and in a fraction of the time of traditional instruments. They are also easily reconfigurable, so development efforts are highly leveraged for future systems. Virtual instrumentation software easily integrates the four most popular types of instruments, GPIB, VXI, serial and plug-in data acquisition (DAQ) boards. A consistent development methodology is applied to each instrument type so a system optimized for both performance and cost is achieved by combining many vendors' instruments and standard drivers.

2.3.4 Reusability

The program of LabVIEW is called virtual instrument (VI). Each VI can be used (called) by other VIs as a subVI. Actually many built-in functions of LabVIEW are subVIs. A subVI can be called at different level of hierarchy, in different places of the VI at the same time. Once a VI is built into a subVI, it can be reused repeatedly in the later programming.

2.3.5 Maintainability

Basically, LabVIEW uses data flow and subVIs to build VIs. If all the subVI’s are well tested, then it is easy to add functionality, easy to trace along the data flow to locate faults.

2.3.6 Interoperability

If the operating system used is Windows 95, 98, or NT, ActiveX in the LabVIEW allows people to use any combination of applications they want to solve their programming problems. One can run a LabVIEW VI from Excel, use a Visual Basic program to load results from a VI into a database. LabVIEW supports multithreading. Users of Windows 95, 98, and NT, Sun Solaris 2, and Concurrent PowerMAX will be able to write applications that take full advantage of LabVIEW's new multithreading capability. If there is a single processor in a computer, LabVIEW will preemptively multitask areas of the application that can execute independently or in separate threads. This is similar to what these operating systems do when they share processor time between separate applications.

If there are multiple processors in a system, the tasks will be able to run on separate processors. The tasks are preempted according to the priority assigned to the sub-virtual instruments (VIs) of each task. Although the number of threads available is limited, it is definitely an advantage to programmers who want to speed up their code and assure the tasks in an application are given the attention they deserve, without monopolizing the processor. To take advantage of multithreading in LabVIEW, a programmer can specify that subVIs run in separate execution systems.

2.3.7 OOP

The nature of LabVIEW supports object-oriented programming. The subVIs can be treated as modules. Some subVIs has single input and single output. Some received multiple inputs and output different data, but these inputs are related and all are necessary for the subVI to process and generate the appropriate outputs. The subVIs are connected by data flow, this ensures that the subVIs have the data coupling. Because of the graphical feature of LabVIEW, it is suitable for rapid prototyping. This property eases the work at the specification phase of the development cycle.

2.4 Other G languages

There are several graphical languages; one of them is HP VEE. Regarding performance, it is appreciably slower than LabVIEW except when performing built-in tasks. When comparing library LabVIEW icons vs. built-in HP VEE functions, the speed of HP VEE and LabVIEW comparable. Sometimes HP VEE was faster, sometimes LabVIEW. Even when the wiring is added, LabVIEW still is clearly faster. In almost all cases, either will have the performance to do the task. Most likely, if HP VEE is not fast enough, LabVIEW will not be either. Most tasks requiring high performance are butting against the real-time OS question, not a HP VEE vs. LabVIEW issue. The more serious concern is whether either package can break the real-time barrier. That's where almost all the performance questions really lie.

The area of difference between HP VEE and LabVIEW is the architecture of the graphical programming. HP VEE customizes its icons with a label rather than changing the form of the icon as LabVIEW does. As a result, the LabVIEW programmer approaching HP VEE wonders where the differences are. The HP VEE programmer approaching LabVIEW wonders how they can ever remember what all the symbols mean.

The approach to flow structures in the two languages also is completely different. HP VEE is based on the flow chart: Everything needed to understand a layer or a level of abstraction is visible. Structures are constructed much the same way as it is done in a flow chart. For example, in HP VEE, a case structure would have an if/then/else object with a bunch of ELSEIF tests. From each of these tests, a line would lead to the next step. LabVIEW embodies the structures as windows in the syntax. In LabVIEW, the case number would be constructed as an integer. The construction would be up to the programmer. Then, each of the cases would reside in a frame, all sharing the same screen window [4].

2.5 User Interface Event Programming

Since the beginning, LabVIEW programmers have written user interfaces (UIs) in essentially the same way, using a technique commonly known as polling. A user interface VI would read all controls on its front panel in a loop, checking each one to see if it had changed since the last iteration. Over the years, clever users have developed advanced tricks and techniques to make UI programming easier, such as grouping together Boolean controls into invisible clusters so they can be tested in fewer operations, or using queued state machines to manage the overall state of the UI. However, the underlying mechanism has remained the same since LabVIEW's inception. Unfortunately, this mechanism has a few inherent problems. Polling is inefficient because every control must be tested repeatedly, regardless of whether it has changed or not. Poll too fast and the user can dominate CPU time. Poll too slowly and the user might miss something, or be unable to determine the order in which multiple user actions occurred [5].

The Event Structure, which is the first new structure node since LabVIEW 1.0, makes it possible to do event-driven programming in LabVIEW. Event-driven programming is a popular paradigm for managing user interfaces in other language environments such as LabWindows/CVI and Visual Basic. With event-driven programming, an application can sleep until something "interesting" happens on the front panel, rather than having to repeatedly poll for such activity. LabVIEW takes advantage of features in the operating system to be notified when user interface activity occurs, so the OS can give the CPU to other programs running on a computer while the application is idle. The Event Structure can be configured to listen only for the events that are important for an application.

An Event Structure is kind of like a cross between a Wait on Occurrence function and a Case Structure. Like a Case Structure, the Event Structure contains multiple sub diagrams, each of which is configured to handle one or more events, which are user actions such as Key Down or Mouse Move. When the user drops an Event Structure on the block diagram, the same way it would be for any other G object, and it executes according to the normal data-flow rules. When LabVIEW executes the Event Structure, it puts the VI to sleep until one of the events it is configured to listen for occurs, just as a Wait on Occurrence sleeps until an occurrence is fired. When an event of interest happens, the Event Structure automatically wakes up and executes the appropriate sub diagram to handle that event. Each sub diagram has an Event Data node affixed to its inside left border that allows access to specific information about the event that occurred. The node looks and works like an unbundle by Name function, so a user can resize it and select only the data fields needed. [5]

2.5.1 Simple Example: Event Structure

Figures below show a simple VI that uses an Event Structure to monitor "OK" and "Cancel" buttons on the front panel, and brings up a confirmation dialog when the user attempts to close the panel.

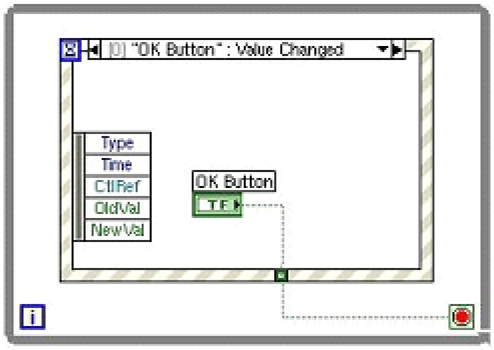


Figure 2.5 : A simple Even Loop

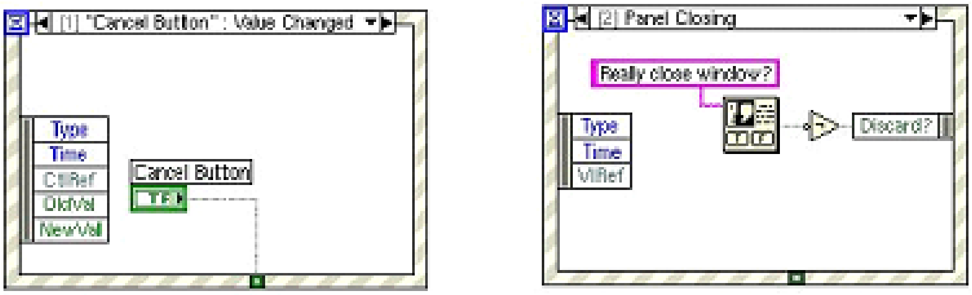


Figure 2.6 : A While Loop with Event Structure

2.6 Nonvisible Event Structure Cases

In the above While Loop an Event Structure is used because when the Event Structure executes, it will only wait for and handle exactly one event. It is very important to place an Event Structure in a loop to repeatedly process events. Failure to do so can cause the VI's front panel to hang if an event occurs a second time while the VI is running, but the Event Structure has completed once and is not called again. This is because when an event occurs that any Event Structure on a VI's block diagram is configured to handle, LabVIEW will lock the front panel until some Event Structure has handled the event. This is done to ensure that events are never lost and are always processed in the order they are received. It is possible to avoid the hanging scenario by clearing the "Lock Panel" option, but this might produce other undesired results. The key is to place the required Event Structure in a loop.

The user actions that generate events include mouse button presses and releases, key presses and releases, control value changes, menu selection, and a VI front panel window closing, among others. The user selects which events are required to be monitored by using a configuration dialog that lets the users choose one or more events to associate with each sub diagram. These events are divided into two classes:

- Notify Events
- Filter Events

A Notify Event is a simple after-the-fact notification that some-thing happened on the front panel. For example, the value changed events for the OK and Cancel buttons in the above figures are Notify Events. Filter Events, on the other hand, allow a user to programmatically affect the outcome of the action that triggered the event. The user can can either discard the event entirely or modify the data for the event before LabVIEW finishes processing it. For example, the Panel Closing event handled in the second figure above is a Filter Event; the user can recognize this by the presence of the Event Filter Node on the inside right border of the sub diagram. If the user presses Cancel on the "Really close window?" dialog, the Two Button Dialog will return FALSE, and the "Discard?" terminal will be TRUE, so LabVIEW will discard the event without closing the panel window. All Filter Events will have a "Discard?" terminal; some will additionally have other fields, which can be modified as the event "passes through" the handling diagram. For example, because Key Down is a Filter Event, the user could make a string control only accept upper case characters by translating keys from lower case to upper case as they are typed. If a value is not wired to an Event Filter node terminal on the right, the value passes through unmodified.

There are a few more things to mention about the simple example before we move on. It should be noticed that even though both the old and new values of the controls are available as event data in the OK and Cancel button event cases, the values of these controls are still read from their front panel terminals. Because the OK and Cancel buttons are Latching Booleans, it is still necessary to read their front panel terminals in order to get them to reset.

If the Value Changed event and never read the terminal are merely handled, the buttons will stay pressed and never "pop up." In general, it is always a good idea to place the front panel terminal of a control inside the event case for its Value Change event if it is not needed elsewhere. Special attention is required to be paid for this rule if a "Stop" button is used to end an event-handling loop; if the value is read outside the event case, LabVIEW will read the terminal before the Event Structure executes and goes to sleep, so the loop might execute one more time than it is expected. There is a little white box displayed on the tunnels that exit from the Event Structure. This is a new kind of tunnel in LabVIEW 6.1; it automatically uses the default value for its data type when unwired. This behavior can be turned on and off by right-clicking the tunnel and toggling the "Use Default If Unwired" option. (Case Structure tunnels also have this feature, but it is off by default) The blue hourglass terminal in the upper left corner of the Event Structures is the Timeout terminal; if a value in milliseconds is wired, the Event Structure will wake up and execute a special Timeout case if no other events occur within the specified duration. If the terminal is unwired (or wired the value -1), the Event Structure will sleep forever if no "interesting" events ever occur. So it might be better use a Timeout case if periodic processing in the background is required.

The above explanation covers what can be learnt from the simple example. Though, real-world applications will be more complicated than this simple example, more advanced users might find it intuitive to use the Event Structure in conjunction with a state machine.

2.6.1 State Machine

LabVIEW 6i demonstrates the use of a queued state machine to handle the user interface of a VI. Traditionally, in the Idle or No Event state, the VI polls for front panel activity, comparing the current value of controls to old values stored in shift registers. The results of these comparisons will tell what buttons have been pressed or menu items selected, and thus what the next state should be. This approach has two unfortunate limitations.

If multiple transitions occurred, there is no way to determine their order. It is possible that a control's value was changed to a new value, and then back to its original value before the "Idle" state executes, and actions that would have triggered a state transition go undetected.

These transitions might be important. As mentioned earlier, the Event Structure has the potential to eliminate both shortcomings. Modifying an existing UI state machine to use an Event Structure is easy. This can be achieved by replacing the code in the "Idle" state that is checking controls for value changes with an Event Structure configured to handle Value Changed events for those same controls. And each event handling case should then write the action to the queue.

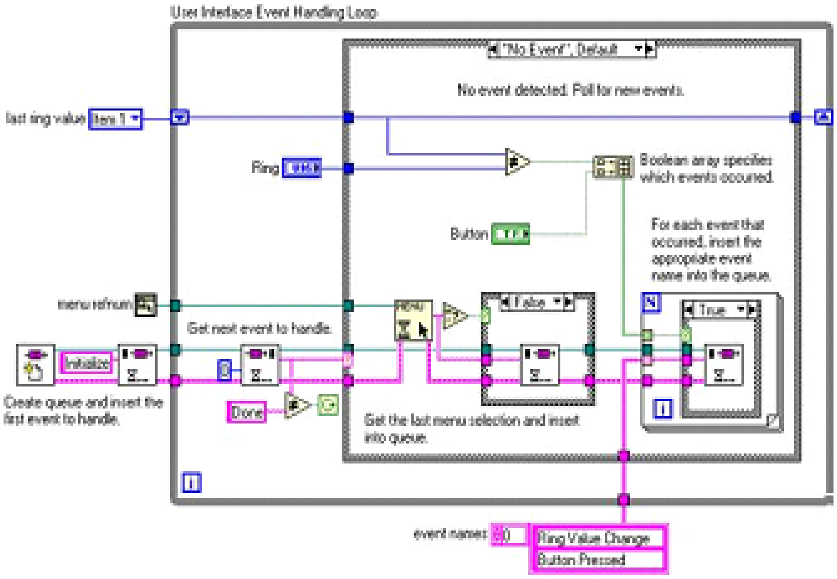

User Interface Event Handling Loop
"No Event", Default
No event detected. Poll for new events.
last ring value
Item 1
Ring
Boolean array specifies which events occurred.
Button
For each event that occurred, insert the appropriate event name into the queue.
menu refnum
Get next event to handle.
False
True
N
Initialize
Create queue and insert the first event to handle.
0
Done
Get the last menu selection and insert into queue.
i
event names
0
Ring Value Change
Button Pressed

Figure 2.7 : A Traditional User Interface State Machine

The above Figure illustrates the block diagram of a simplified version of the example VI. The "No Event" case checks for Menu selections, a button being pressed, and the value of a ring changing. Using the Event Structure, things like selection of User and Application menu items, and Value Changes of controls can be detected, so that the user can replace all the code inside that case with a single Event Structure. The resulting diagram code is shown in the following Figures.

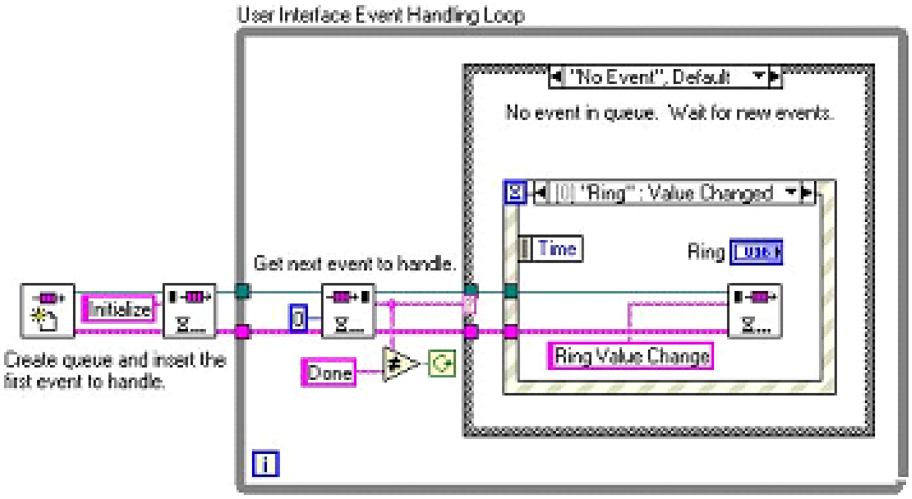

User Interface Event Handling Loop
"No Event", Default
No event in queue. Wait for new events.
[0] "Ring": Value Changed
Time
Ring
U16
Get next event to handle.
Initialize
0
Done
Ring Value Change
Create queue and insert the first event to handle.

Figure 2.8 : A UI State Machine Using the Event Structure

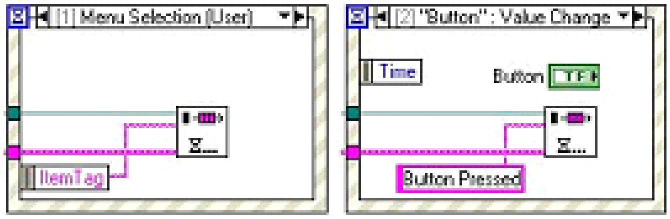


Figure 2.9 : Nonvisible Event Structure Cases

In addition to simplifying the diagram dramatically, the use of the Event Structure has two more benefits. Primarily, all events are queued, even if the Event Structure is not presently waiting on them. Due to this the user does not miss any state changes. Also, event handling is synchronous. The events are guaranteed to be processed in the order they are received unless two Event Structures are executing in parallel. A couple of potential problems can arise when using the Event Structure with a more complicated state machine. First, the Event Structure by default sleeps indefinitely until an event fires. In the above example, the state machine will remain in the

"No Event" case until a user interface event occurs. In many cases, this is desirable, as it minimizes CPU usage.

Another caveat when using Event Structures with state machines is that it is very important to make sure there is always an Event Structure available to handle any user interface event that is generated. As mentioned earlier, if an event occurs and there is no Event Structure waiting to handle it, the user interface will be locked until the diagram code executes in such a way that an Event Structure configured to handle that event runs. In the best case, this will cause some latency in the feedback of the user interface. In the worst case, it will cause deadlock in the VI. The Abort button, which is immune to event locks, should be used in such situations. [5]

3. METHODOLOGY

3.1 Problem Description

In this thesis a Real – time system is developed which integrates several real time experiments and remote control access over Internet is provided using Labview. The objective of this work is to develop and test a real-time experiment system with LabVIEW and provide remote control access over the Internet. The system will allow the users to access and control the experiments and collect data automatically. The ways to do this depend on the physical instruments or devices on which the users do the experiments and the interfaces between them and the controller (the computer).

There are four computerized instruments available in this project: The Gould Classic 6100 Series Digital Storage Oscilloscopes from Gould Instruments [6], Inc., a PT 236 Process Trainer (temperature controller) from Feedback Instruments Ltd., [7,9] and an Electronic Recording Rain/Precipitation Gauge (rain logger) from RainWise Inc., [8,9] and Automatic Mixing Controller [1]. There are also four methods for an application to communicate with the instruments from a computer: by General Purpose Interface Bus (GPIB), [1] by Data Acquisition (DAQ) board, by serial port and by external Data acquisition Board. [3] The four instruments listed above will be monitored and controlled by using these four methods, respectively, as shown in Figure below.

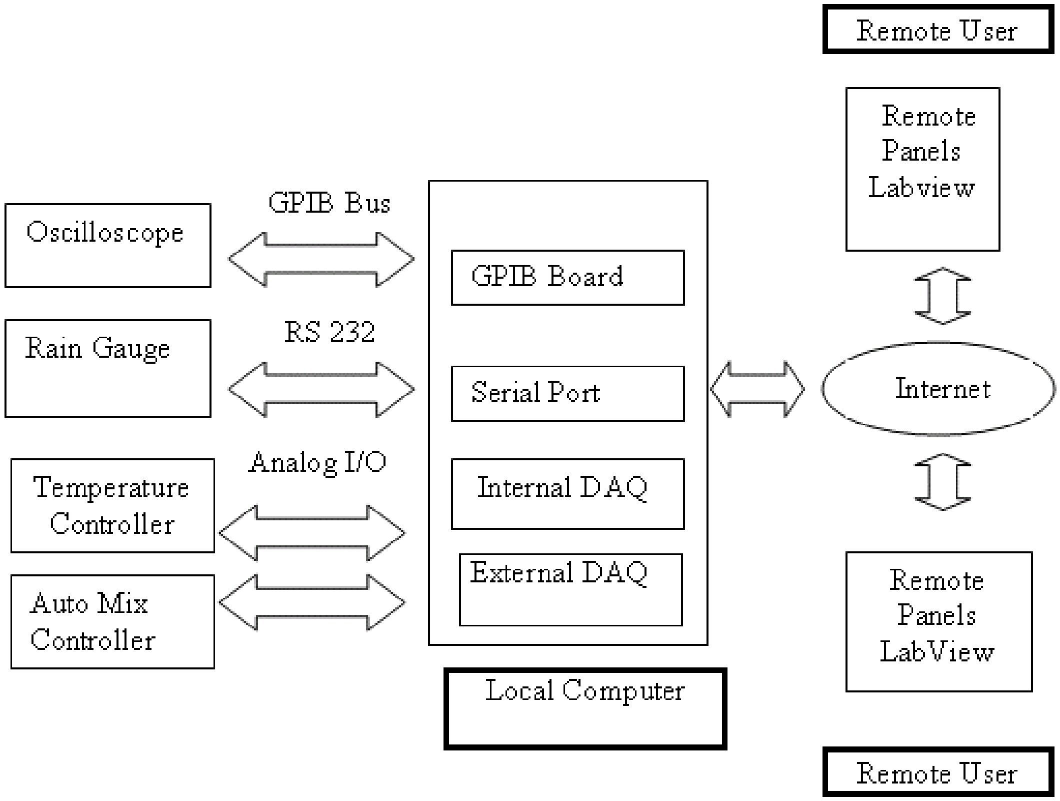


Figure 3.1 : The Architecture of the Integrated Real – Time System

All the four experiments can be accessed from anywhere via Internet to monitor and control them in real time. There are different methods of remote access, such as Common Gateway Interface (CGI) and ActiveX, and Remote Panels Technology using LabVIEW 6.1, to implement the Internet connections. The connectivity via different methods is made possible due to the use of the web server (G web server) provided in the add-on Internet Toolkit (ITK) for LabVIEW.

3.2 The Automated Controlled Mixing Process Experiment

Automatic control system facilitates process automation where dynamics of the process are known. In today's industrial market, efficiency and reliability of Automatic control systems have revolutionized the industries such as automakers to petrochemical. Controlling a mixing process is important part of production application for industries such as pharmaceutical, steel makers and food and beverage. Mixing processes usually are controlled by variety of software packages and hardware components. The main objective of this Thesis project is not to reinvent the wheel but to demonstrate capabilities of National Instrument's LabVIEW software package and a DAQ card by National Instrument to control an automated mixing process [1,3]

3.3 Experiment with the Digital Storage Oscilloscope

Gould has developed a series of Classic Digital Storage Oscilloscopes suitable for a wide range of applications like data analysis and data acquisition. Incorporating state of the art integrated electronic components and using the latest manufacturing techniques, high performance and ease of use ensures excellent value. [6]

This oscilloscope has four channels; each has a record length of 1000 samples and 8-bit vertical resolution at 100 MS/s and 12 – bit High Resolution mode to 25 MS/s. All the channels acquire waveforms simultaneously. The maximum sampling rate is 1 Giga-samples per second and has 200 MHz analog bandwidth and fastest base setting of 2.5 ns/div. Offering a high performance 12-bit resolution mode as standard, the Classic 6100 will be of particular interest to users involved in analytical scientific research where the dynamic range of the Classic 6100 guarantees that the complete signal is acquired, including the fine detail. The high-resolution capability is achieved using a system that ensures low noise, linearity and high accuracy performance to give both high quality data and results when analyzed or measured.

For meaningful analysis and to preserve data integrity, it is of paramount importance to avoid aliasing on high frequencies. The Classic 6100 has an Anti-Alias filter for each time base, which removes high frequency noise beyond the maximum sample rate. With FFT averaging, distortion 80 dB down from the fundamental can be observed.

This oscilloscope has a rear GPIB connector. When connected to the GPIB board of the PC running LabVIEW applications, it can be programmed and controlled. In practice, a number of selected functions of the oscilloscope will be implemented by using LabVIEW. A LabVIEW application with a front panel depicting the oscilloscope will be developed. The user will control the oscilloscope via this software panel. The application should retrieve waveform data from the oscilloscope and present the data in a waveform graph. The front panel operations will be transformed to command sets and sent to the oscilloscope via the GPIB bus.

3.4 Experiment with the Temperature Controller

The temperature controller is in fact an entire process trainer used to teach control theory via experiments. Its objective is to do close-loop proportional control. The trainer accepts a SET VALUE voltage (0 to 10 V) that represents the desired temperature. Then the output temperature (voltage) is measured. The deviation from the SET VALUE is read by the computer running LabVIEW. According to the control algorithm, a certain control signal is sent back to the trainer to take the corrective action. The LabVIEW application can set the desired temperature; acquire the deviation data from the trainer and write the control signal to the trainer so that it can take the corresponding corrective actions. The communication between the LabVIEW application and the trainer will be implemented via the analog I/O channels in the DAQ board, through its analog input and output. The LabVIEW application will show the SET VALUE and the deviation and present them graphically as waveforms in the front panel.

3.5 Experiment with RainWise Recording Rain/Precipitation Gauge

The RainWise Recording Rain/Precipitation Gauge consists of industrial tipping bucket rain gauge connected to a battery operated 32K data logger, housed in a NEMA 4X weatherproof enclosure. This assembly is mounted on a PVC pipe for simple hole in the ground installation. The logger records the date once every 24 hours, and records the time and rainfall at the logging interval selected. The user can select this interval from once a minute to once an hour. The logger records the accumulation in increments of 0.01 inch/0.25mm during each time interval and adds this number to the existing amount. The rate of rain/precipitation is also calculated for each recording interval.

The information stored can be retrieved with a computer via the serial line (RS 232). A built-in microprocessor with an on-board RAM allows it to transfer data to a PC through the serial port or directly drive a dot matrix printer. To control the rain gauge, a PC running LabVIEW will be connected with it via the RS232 bus. The application will acquire data of rainfall and will convert the data into a special format so that they can be viewed in LabVIEW panel and be presented graphically. [8]

3.6 Remote Access to the Experiments via Internet

The revolution of Internet-enabled instrumentation is emerging as a revolution in Measurement and Automation. New standards are being developed for transmitting data and connecting instruments to the Internet, such as Remote Panels from LabVIEW 6.1, Data Socket from National Instruments, and others such as Java and Jini from Sun, continue to mature. Making instrumentation information available through a network is likely to bring a dramatic increase in productivity, collaboration, and technological capability.

The needs for remote or web-based instrumentation may include:

- Remote or geographically scattered data acquisition, which may require collecting data from many remote sites around the world.
- Sharing the experiment sources among different users, which may require a global collaboration of researchers.
- Distributed computing, when it is needed to share computing and hardware resources to accomplish a task that might be burdensome, inconvenient, or impossible to perform on one machine.
- Harsh experiment conditions, which may not allow personnel presence at the experiment sites.

In this project, the users will get access to the four experiments, monitor and control them in real time via Internet, using Remote Panels of LabVIEW 6.1. Before the emergence of remote Panels techniques in LabVIEW 6.1 several techniques like CGI, Active X were used to provide Internet access to the applications

3.7 Internet Connectivity Using Remote Panels in LabVIEW 6.1

With LabVIEW 6.1, the user can operate the front panel on a machine that is separate from where the VI resides and executes. Furthermore, as the front panel is embedded into a web page the user can operate it within that page. All that is required on the client machine for executing in a web page is a browser and the LabVIEW 6.1 run-time engine and browser plug-in.

It is very easy to configure and use remote panels. [2] There are two steps:

- Enable the LabVIEW Web Server on the server machine.
- Connect and execute remote panels on the client machine.

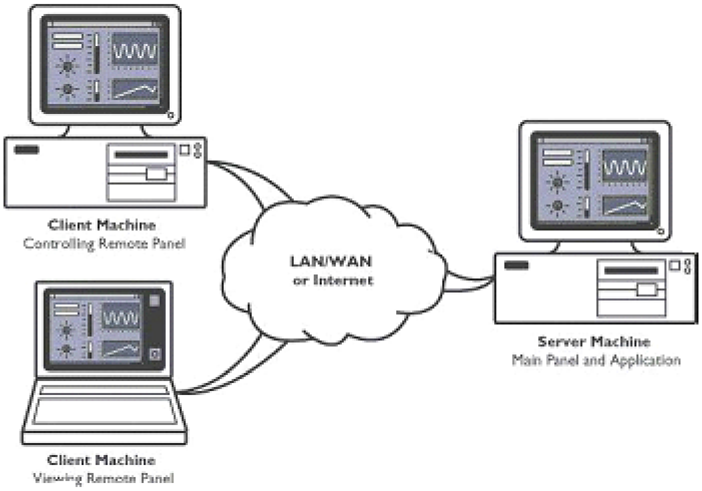


Figure 3.2 : Distributed Application

4. DESIGNING THE INTEGRATED APPLICATION

4.1 Experiment with the Automated Controlled Mixer

This project demonstrates a simple model of Automated Controlled Mixing Process with relatively simple parameters. This project concerns the control of a mixing process. The process will consist of two separate tanks of water, varying in temperature that will be combined in a third tank to meet temperature and volume selection. Here the liquid will be mixed and dispensed. The temperature and volume parameters are to be selected by the user. The general idea of the project is to develop a simplified model of an industrial process control application and design the parameters around which the model will operate. For example, there are many industries such as pharmaceutical, food and beverage, and petrochemical, in which process control is fully automated. A central processing unit in conjunction with multiple data I/O cards takes into account all of the variables and parameters involved in the process. Through software programming, specific amounts of one substance can be added to another at a pre-determined rate. These rates are calculated by opening and/or closing certain valves for a determined amount of time, based on the size of the medium and the pressure of the substance present in the medium. The process allows the mixture to be passed down the line for further modification, and eventually packaging.

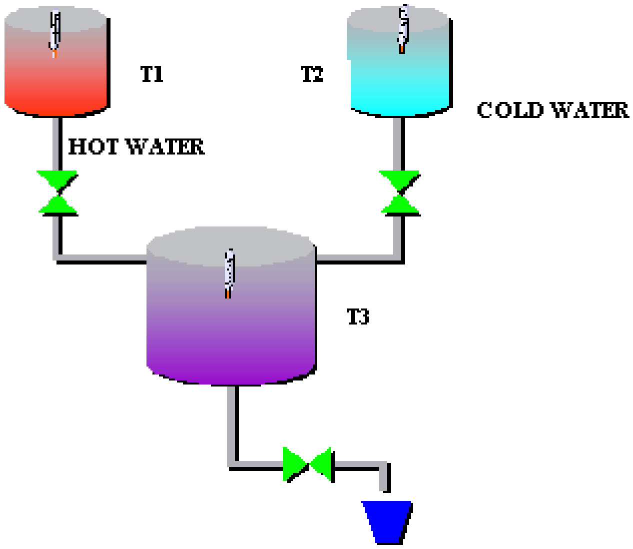


Figure 4.1 : Mixing Tank with Dispense reservoir

Design Specifications:

- Two 5-gallon plastic tanks.
- Three Direct Acting Normally Closed Solenoid valves.
- A conical tank to ensure all liquid is dispensed.
- T-type thermo coupling
- Gravity fed system.
- Vinyl tubing
- 120 VAC -60 Hz -4 Watts -60 RPM Mixing motor.
- National Instruments PCI-6024E Data Acquisition board.
- National Instruments SCB-68 connector block with cold junction compensation.

4.2 Design Description:

The Automated-mixing controller has two 5-gallon tanks to hold cold and hot water. Only up to 4 gallons of required to be filled to prevent water leakage from Thermocouple insertion opening on top of stop valve side of the tanks. The mixing reservoir is conical to ensure all liquid is dispensed. For the automated process three major questions are on hands.

- How long will each valve need to be open to arrive at selected temperature and volume?
- Determination of t1 and t2 for hot and cold values respectively. How do we implement this in software?
- What are the fluid dynamics formulas that will do this?

  Mass Flow Properties

  Fluid formulas

What is known?

$M = \rho V$ $V = dx/dt$

Mass (kg) = density (rho (kg/m^3))*Volume (m^3)

Cp = specific heat

T = temperature

t = time (sec)

$M1\ Cp\ T1 + M2\ Cp\ T2 = M3\ Cp\ T3$ (fluid formula)

Assumptions for the Automated – Mixing controller application:

- Same size tubing and solenoid valves with same Flow rates.
- Same substance in each tank, therefore same properties such as specific heat and density.

- Short tubing length results in negligible thermal expansion.
- Volume of hot and cold-water tanks much larger than amount dispensed therefore Flow rate is unaffected.

$$M3 = M1 + M2$$

So $$\rho\ V1\ T1 + \rho\ V2\ T2 = \rho\ (V1+V2)\ T3$$

$$V1\ T1 + V2\ T2 = (V1+V2)\ T3$$

V1 and V2 depend on time the valves are open and Flow rate.

= Flow rate (oz/sec)

$$V1 = \nu\ 1\ t1$$

$$V1 + V2 = V3$$

$$\nu 1\ t1 + \nu 2\ t2 = V3 \qquad (1)$$

$$(\nu 1\ t1)\ T1 + (\nu 2\ t2)\ T2 = V3\ T3 \qquad (2)$$

$\nu 1 = \nu 2 = \nu 3 = \nu$ same flow rate.

Two equations and two unknowns solved for t1 and t2.

2) Open valves time

$$t1 = (V3\ (T3\text{-}T2)/\ \nu(T1\text{-}T2)) \qquad t2 = (V3\ (T3\text{-}T1)/\ \nu(T2\text{-}T1))$$

So third tank open valve time is $t3 = (t1 + t2)$

4.3 Implementation

Flow control is accomplished using 120 VAC 60 Hz Normally Closed solenoid values. There are two types of valves available normally closed or normally open. Our project uses normally closed type. Widely used for dispensing, collating, gas shut off, vacuum holding, and tank draining applications. The valve flow rate is proportional to the orifice size and pressure in medium. For Direct Acting Normally Closed Solenoid valve of pipe size 3/8", orifice size 1/8", Cv Flow factor is .35 gallon per minute operating from 0 to 150 PSI. Temperature Reading

For temperature reading T type Thermocouples are used because it has very linear increase in mV in the range of our project's temperature therefore better reading of water temperature. Low mV induced by medium generated by Thermo couple is amplified and sampled by the DAQ board via Analog Input Channel (0). The connection schematics are developed referring to National Instruments SCB-68 Cold Junction Compensated Connector Block Diagram.

4.3.1 TTL Digital Line

Four digital signal lines are needed to enable the three valves open and close and mixing motor on and off. Opti couplers are used to switch power on and off to the inductive load when TTL Line (Logic line) goes low (fall) and high (True) respectively. Digital channel (0) Line0 and Line1 for Cold and Hot valves were used respectively, and line2 for mixing motor and Line 3 for dispensing valve. Connection schematics were developed referring to National Instruments SCB-68 Cold Junction Compensated Connector Block Diagram. [1]

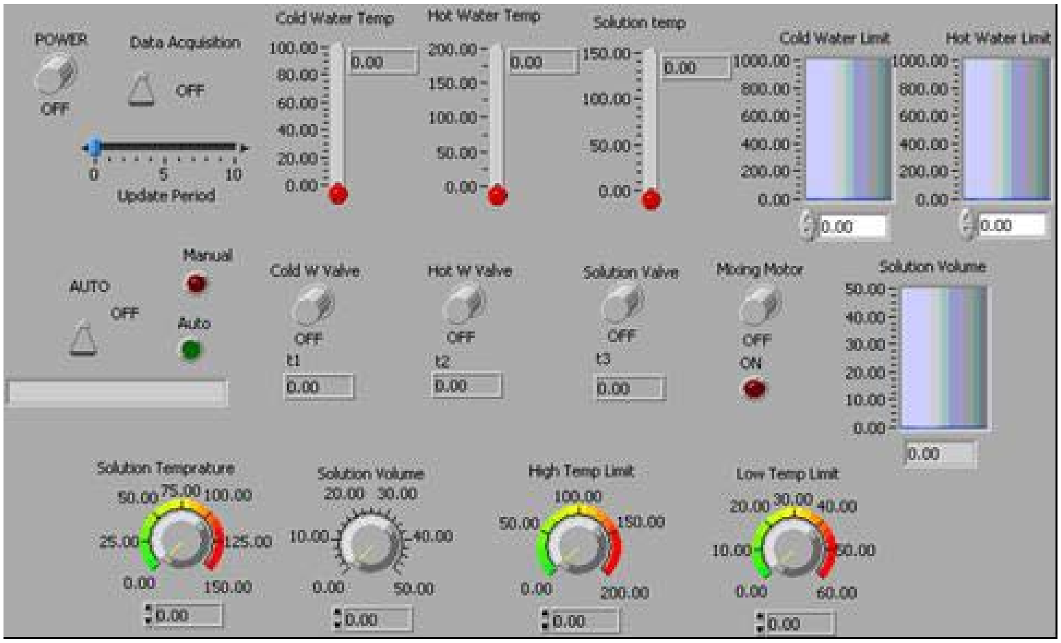


Figure 4.2: Front Panel of the Automatic Mixing Controller VI

The Process acquires data through Thermo couples and selected values by user and manipulates the data and through series of structured While Loop, Case and Sequence sends digital signals sequentially to valves and motor to complete the mixing process. First Loop determines the data Acquisition is on (True) to acquire reading from Thermo couples or disabled (False).

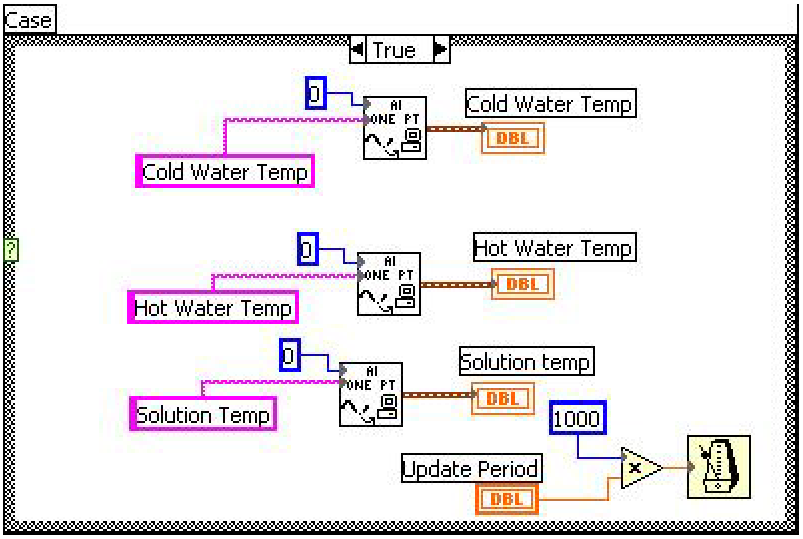


Figure 4.3: Block Diagram of the Automatic Mixing Controller Experiment

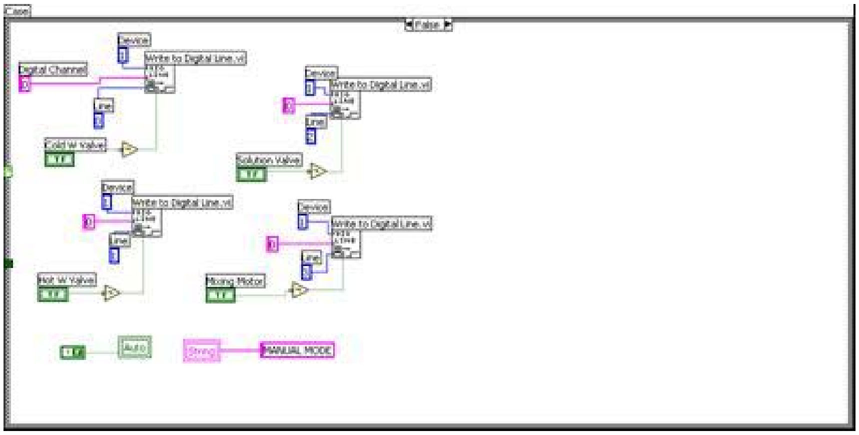


Figure 4.4 : Manual Mode Case:

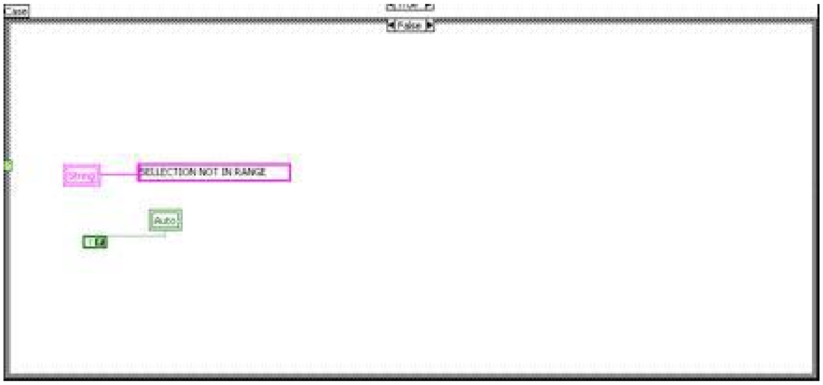


Figure 4.5 : AUTO False Case:

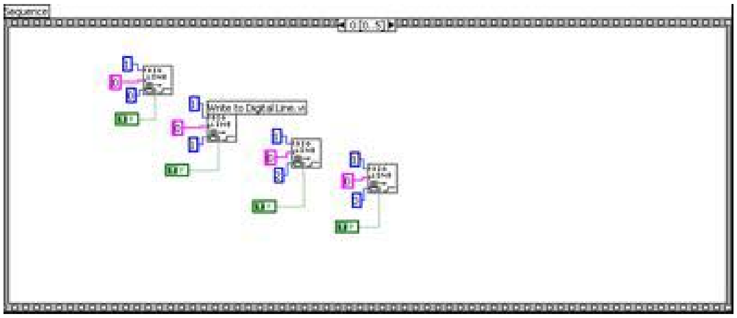
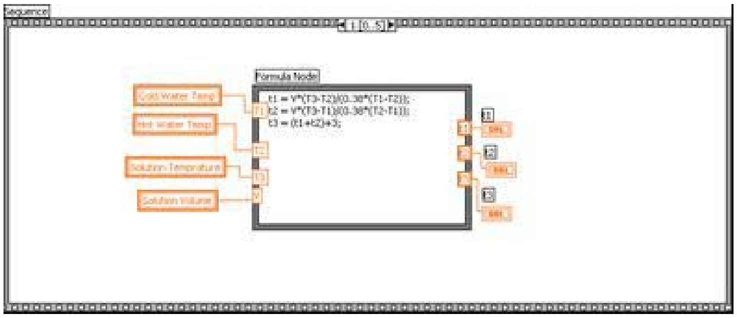


Figure 4.6 : AUTO write to digital line sequence



```
t1 = V*(T3-T2)/(0.38*(T1-T2));
t2 = V*(T3-T1)/(0.38*(T2-T1));
t3 = (t1+t2)+3;
```

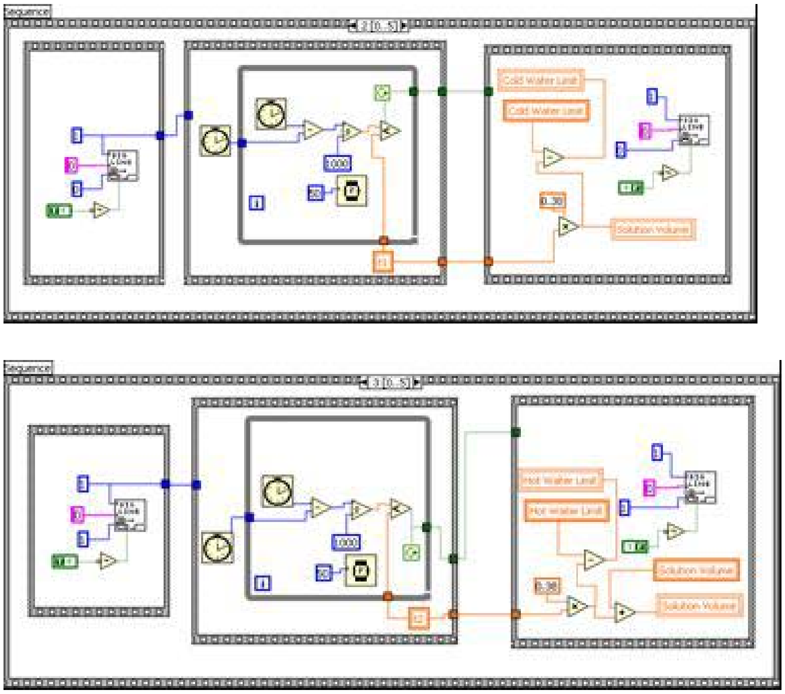


Figure 4.8 : Block Diagram of the Formula node sequence

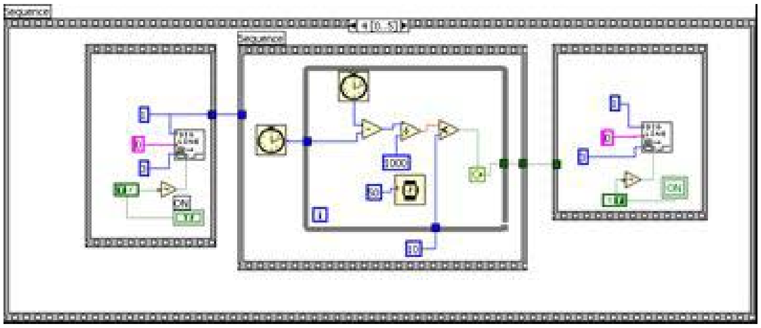


Figure 4.9: Cold Water Valve, Hot Water Valve, Motor and Solution Valve sequences

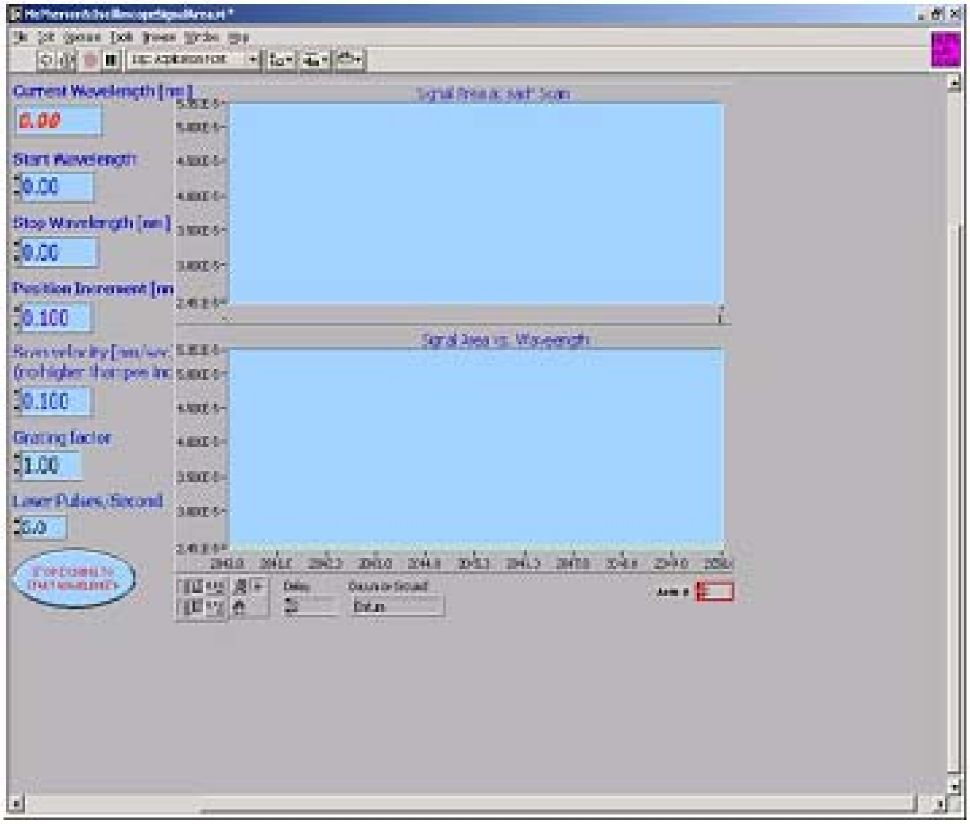
4.4 Experiment with the Gould Digital Storage Oscilloscope

This experiment will allow the user access and control of the oscilloscope remotely over the Internet. The physical entities in this experiment include a PC as the controller to the oscilloscope, the oscilloscope itself, and the remote host as the web terminal. The local PC has the LabVIEW application and the web server for LabVIEW residing, and running in it. The LabVIEW application should have three parts, the Panel VI which is the simulation of the functions of the oscilloscope, the Remote Panels VI which serves as the bridge for the remote web terminal user to control the oscilloscope, and the web page (an HTML file) which brings the panel image to the remote user, and lets the user input the control message to the oscilloscope.

As mentioned above, to do the experiment with the Gould Classic 6100 Oscilloscope remotely, three programs are needed: the HTML coded web page, Remote VI, and Panel VI. The web page serves as the remote user interface. Users can input controls and get replies via the web page. The Remote Panel VI is the bridge between the web page and the Panel VI. It receives controls from the web page and sets the corresponding properties of the Panel VI. Then, the Panel VI sends control commands to the oscilloscope. Once the Panel VI gets the response from the oscilloscope, it will send the updated image to the web page to complete the control loop. In the following sections, the implementation of each program is explained in detail. The techniques and tools used are also introduced.

4.5 The Panel VI

The Panel VI is the program that controls the oscilloscope. Only parts of the functions of the oscilloscope are implemented in this VI, since it is not necessary to realize all the functionality of the instrument. Moreover, it is impossible to do this because not all of the functions are programmable. Figure below shows the front panel of the Gould Classic 6100 Oscilloscope.



Current Wavelength [nm] 0.00

Start Wavelength 0.00

Stop Wavelength [nm] 0.00

Position Increment [nm 0.100

Scan velocity [nm/sec (not higher than pos inc 0.100

Grating factor 1.00

Laser Pulses/Second 5.0

Figure 4.10 : Front Panel of the Oscilloscope.vi

The above figure shows the front panel of the LabVIEW VI that will control the oscilloscope via a NI GPIB board (AT-GPIB/TNT).

The Panel VI can acquire data from the oscilloscope and present the data in the graph window. The channel selection slider sets the source of the data. The Acquire and Display menus control the data acquisition and display properties. The basic GPIB communication includes two LabVIEW built-in VI’s, "GPIB Write VI" and "GPIB Read VI." The former is used to write commands to the instrument and the latter is to read the response from the instrument. The following figure shows the diagram (code) of a typical GPIB communication VI.

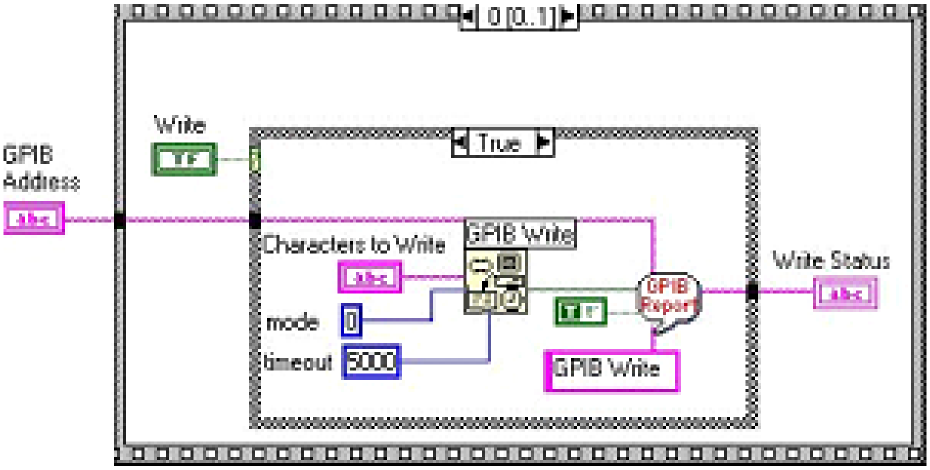


Figure 4.11 : Diagram of a GPIB Write VI.

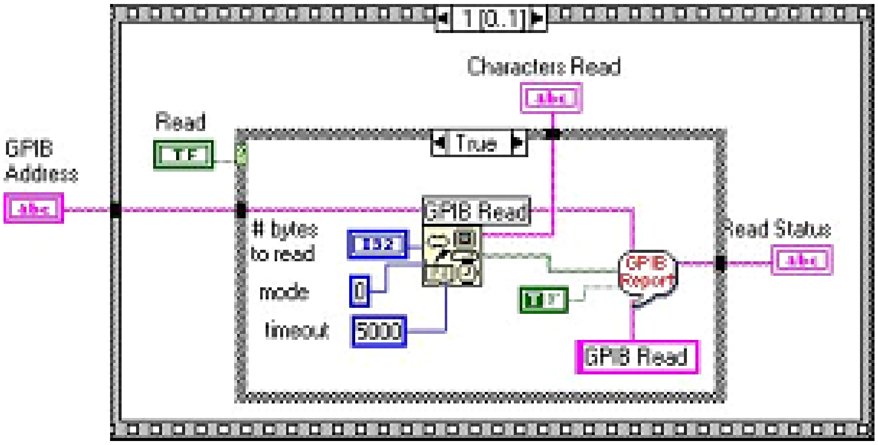
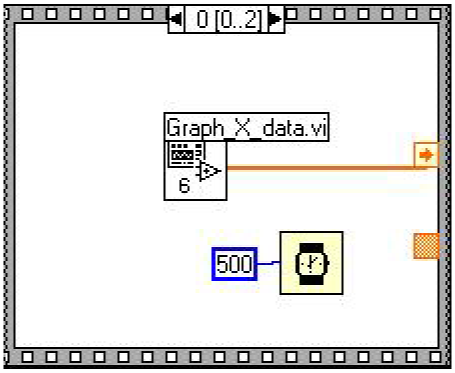


Figure 4.12 : Diagram of a GPIB Read VI.

Any operation applied to the oscilloscope uses one of the write and read operations, or both. The most important part of the Panel VI is the data acquisition and presentation. The visual part is the graph window on the front panel.



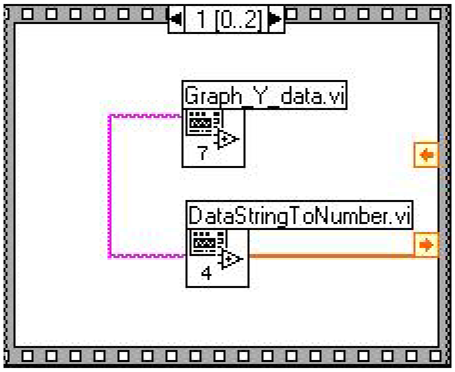


Figure 4.13 : Waveform acquisition and presentation diagram.

This program contains a sequential structure (only sequence 0 and 1 are shown). The data will be presented in a XY graph (a LabVIEW built-in graph object). First, the data for the x-axis (relative time from sampling interval) is obtained, then the y-axis data that are the data points of the waveform are also obtained. After that, the waveform data is transferred from the ASCII format into the numerical format. Finally, the data is sent to the XY graph for presentation. Several subVIs, Graph_X_Data, Graph_Y_Data, and DataStringToNumber are used to complete the above operations (A sub VI is a piece of reusable code. Any VI can be used as a subVI in the VI diagram). Each subVI may have a combination of write and read operations as discussed above.

The oscilloscope channel selection is implemented by using a slider that is also a built-in object of LabVIEW. When a channel is selected, it becomes the data source of the graph window. The default channel is channel 1. If channel 2 is selected, the corresponding command set will be written to the oscilloscope. At the same time, the LED of channel 2 will light up.

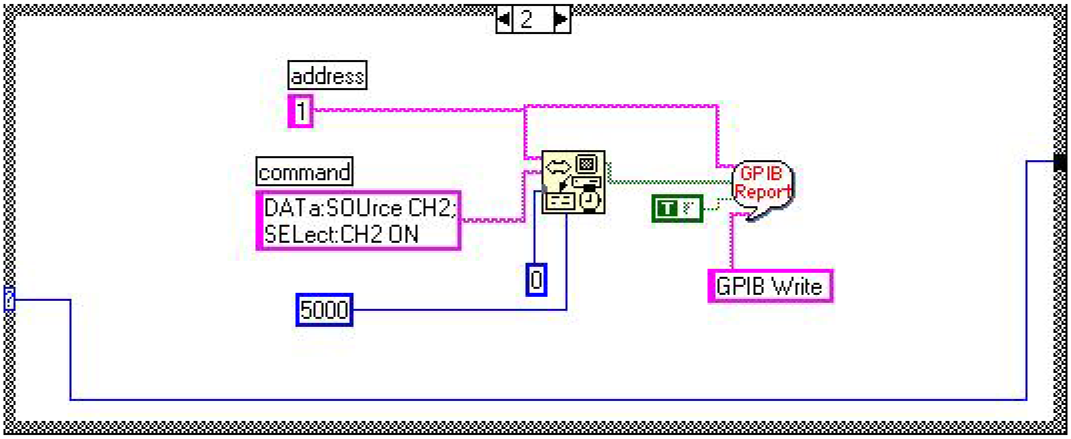


Figure 4.14: Part of the Channel VI block diagram.

There are four knobs on the front panel of the Panel VI. They are used to adjust the horizontal and vertical positions and scale of the waveform. They have similar VI diagrams in which the actions on the knobs are collected and the related command sets are sent to the instrument. Whenever the value of the knob is changed, the Boolean control becomes true and the command set containing the updated knob value will be written to the oscilloscope. The subVI "W" is just the combination of subVIs.

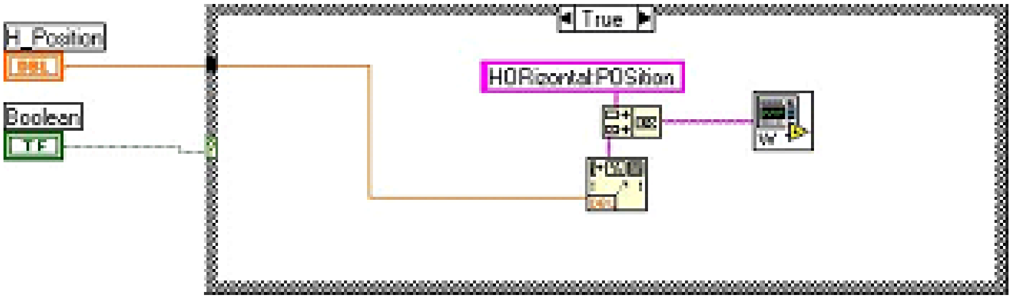


Figure 4.15: A simplified diagram of knob operation on the Panel VI.

4.6 Experiment with the Temperature Controller

The temperature controller that does close-loop proportional control is used in this experiment. After setting the temperature (a voltage between 0 and 10 V), the deviation of the controlled variable from the present value is measured and sent to the LabVIEW VI via NI-DAQ board. According to the control algorithm, a certain control signal is sent back to the controller to take the corrective action.

The instrument used in this experiment is PT 326 Process Trainer from Feedback Instrument Limited [7,9]. In this equipment, air drawn from the atmosphere by a centrifugal blower is driven past a heater grid and through a length of tubing to the atmosphere again. The process consists of heating the air flowing in the tube to the desired temperature level, and the purpose of the control equipment is to measure the air temperature, compare it with a value set by the operator and generate a control signal which determines the amount of electrical power supplied to a correcting element, in this case a heater mounted adjacent to the blower. The elements, which form the system and the connections for this experiment, are shown in the Figure below.

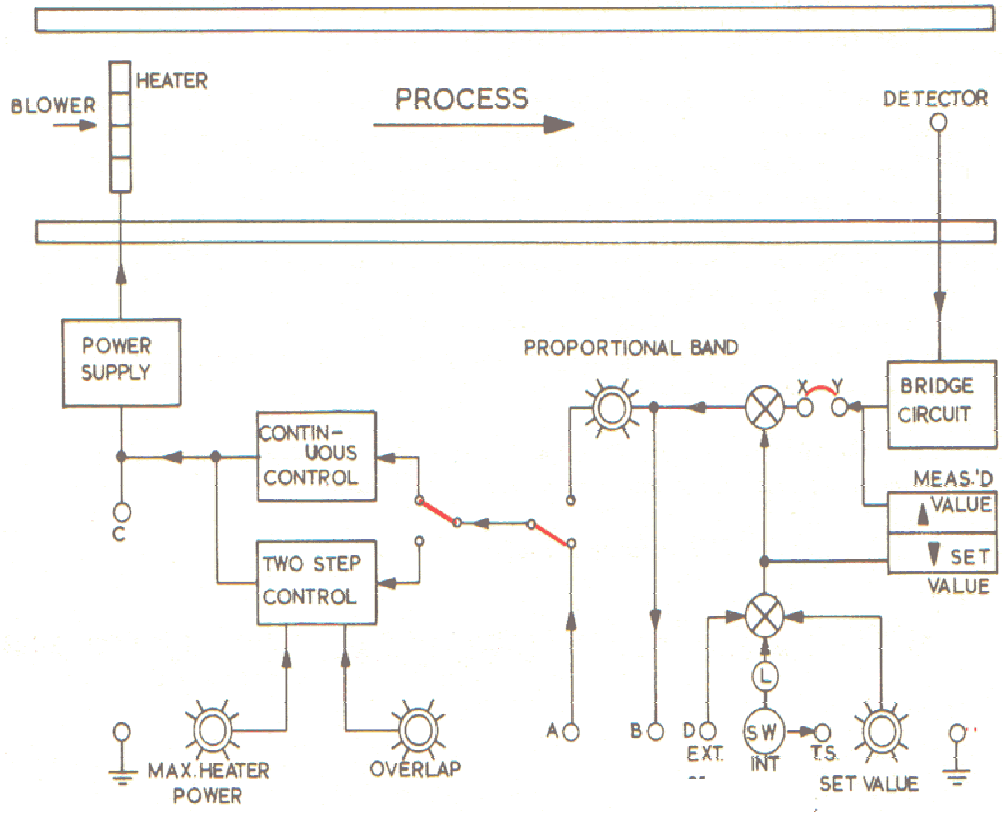


Figure 4.16 : The temperature control process and connections for this experiment

In the Figure above, the desired temperature is set by sending an analog signal, called Set Value, to socket D. The Set Value is between 0 and -10V, representing temperature between 30 and 60°C. The deviation between the set value and the measured value can be read from socket B. The control signal is sent back to the process trainer through socket A for the process trainer to do the correction. The control signal is generated, according to proportional control algorithm, by multiplying the deviation by a factor called Gain Value.

The software components of this experiment are similar to the last one: a LabVIEW VI which controls the temperature controller and communicates with Server and an HTML file. To control the temperature controller, LabVIEW's DAQ card (AT-MIO-16) is used. There are analog input and output channels in the interface of the DAQ card, which are used to write to and read from the temperature controller. This feature allows to program this instrument very easily.

The front panel of the temperature controller is as shown below

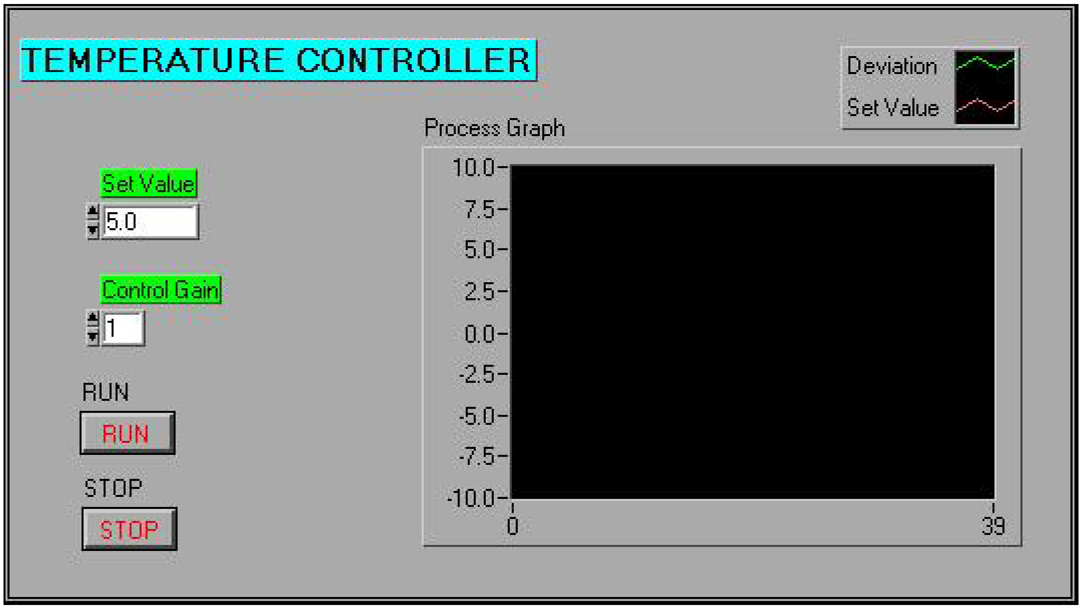


Figure 4.17: The front panel to control the temperature controller.

In the Figure above, a user can input "Set Value" and "Control Gain" from input boxes. When the "RUN" button is clicked, the VI will run the control process continuously until the "STOP" button is pushed. The "Set Value" and deviation will be presented in the graph. Part of the block diagram of the Temperature Controller VI is shown in the following figure

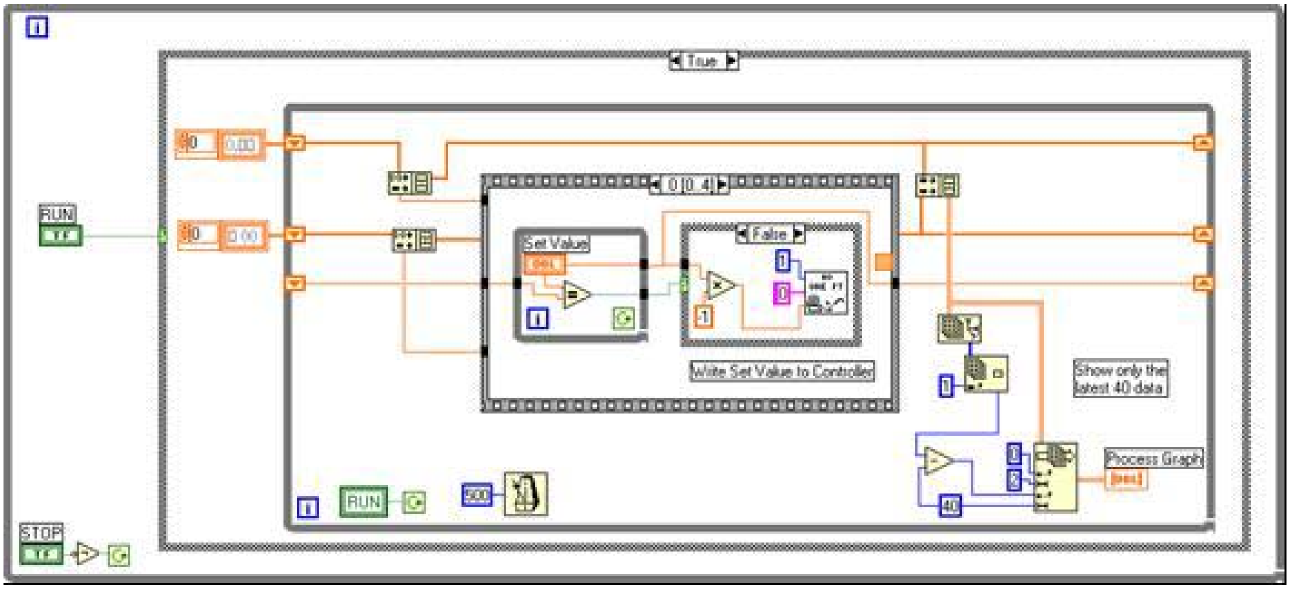


Figure 4.18 : Block Diagram of the Tempcontroller.vi

The sequential structure in the middle of the diagram is the core of the VI. In the structure, the set value is written to the controller first. After the deviation is measured, it is multiplied by the control gain to form the control signal that will be sent back to the controller. A while loop will let the VI detect the changes of set value and control gain and adjust the output signals accordingly. Because the correction action involves motor action and airflow, a delay is added to allow enough time of reaction between writing control signal and reading deviation.

4.7 **Experiment with the Rain Gauge**

In this experiment, the instrument to be controlled is a rain gauge called weather log that measures the amount of rainfall automatically. It is connected via the serial port to a PC running LabVIEW application to acquire real-time data. The user can access the system remotely and view the data in real time from a web browser over the Internet. The local PC has the LabVIEW application and the web server for LabVIEW residing and running on it. The mechanisms to implement the remote control is by using Remote Panels for embedding the front panel into a web page and operate it within that page control. To let the web browser communicate with LabVIEW, an HTML file that embeds the front panel into a web page and operate it within that page at the same time, the Data Socket Sever should also be enabled. The LabVIEW VI controls the rain gauge via an RS232 bus. The function of the Data Socket is to transfer the control message from the remote user to the LabVIEW VI and send data back to the user. The web page is actually an HTML file that contains an image similar to the front panel of LabVIEW VI. The remote user can view the data and input the control messages to the rain gauge from the web page.

RainWise Recording Rain/Precipitation Logger (rain logger) is a device that stores the time and rainfall amount for each recording interval in which rainfall is measured. The stored data can be retrieved from the rain logger by connecting the computer's RS232 port to the rain logger's serial port with the 9-pin cable provided.

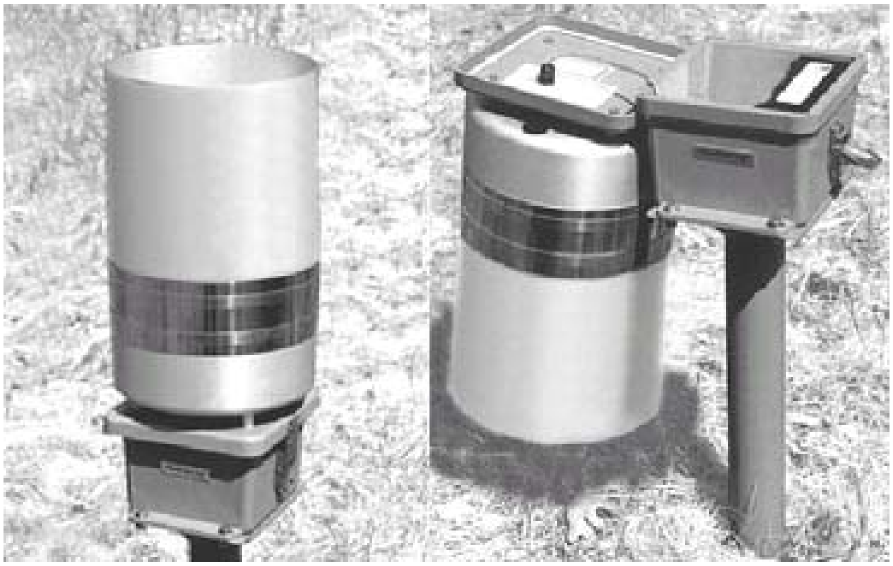


Figure 4.19 : The Rain/Precipitation Recording Gauge

In this experiment, a VI that retrieves data from rain logger is built. This VI will communicate with the rain gauge via the serial port of the host computer. A web page is provided to assess the experiment as an HTML file with a Remote Panel embedded in it. As the remote user interface, the web page, when used with LabVIEW's Remote Panels protocol together, allows users view the raw data, process the data and present the data graphically

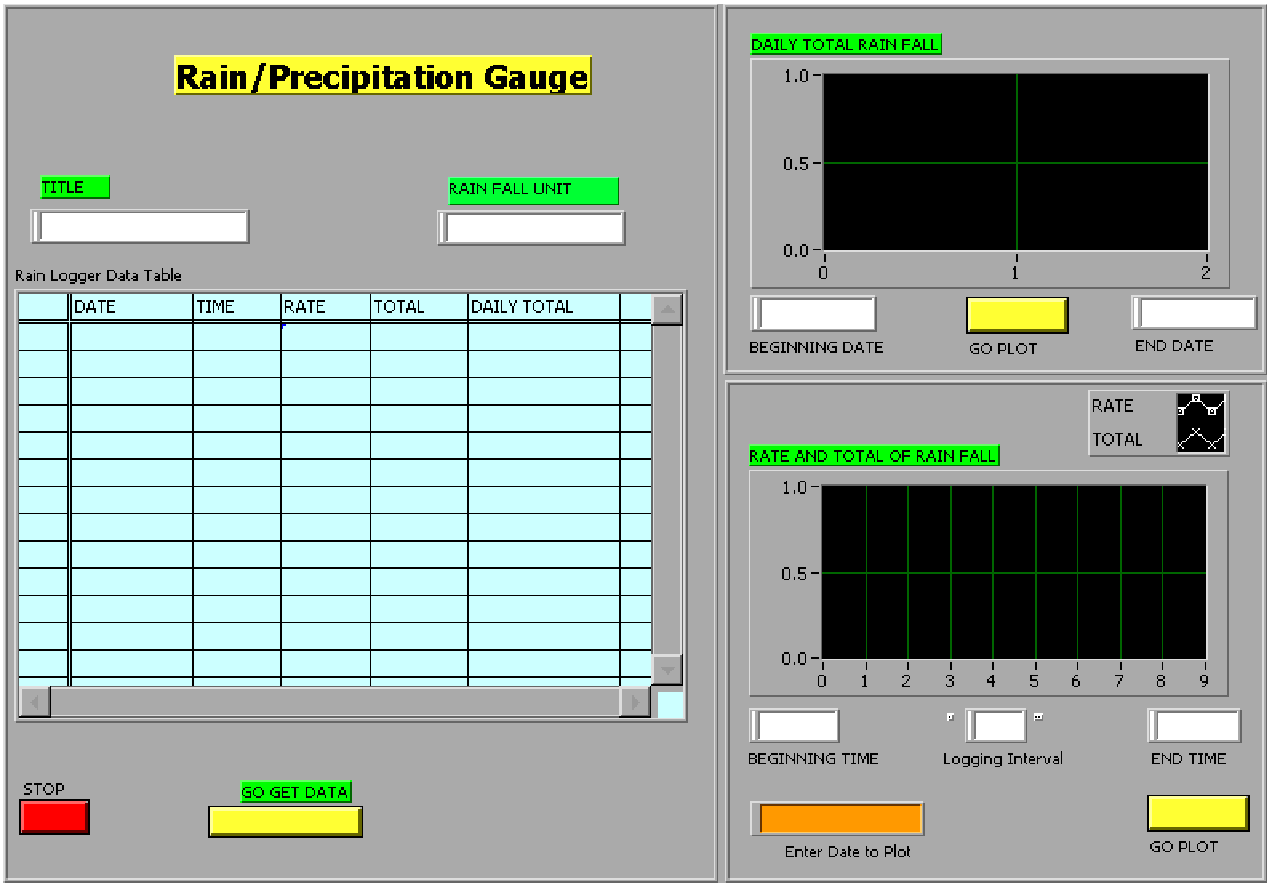


Figure 4.20 : Front Panel for the Rain –Precipitation Gauge.vi

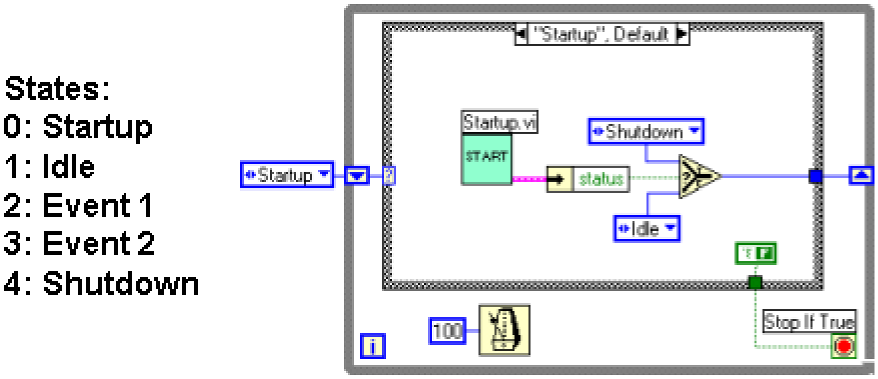
The RainWise Recording Rain/Precipitation Logger consists of a industrial tipping bucket rain gauge connected to a battery operated 32K data logger, housed in a NEMA 4X weatherproof enclosure. This assembly is mounted on a sturdy PVC pipe for simple hole-in-the ground installation. The logger records the date once every 24 hours, and records the time and rainfall at the logging interval selected. The user can select this interval from once a minute to once an hour. The logger records the accumulation in increments of .01 inch/0.25mm during each time interval and adds this number to the existing amount. The rate of rain/precipitation is also calculated for each recording interval. The data logger is supplied with non-volatile data

RAM. Logged data will be preserved for up to 10 years by a built-in Lithium battery on this RAM with a storage capacity of 8,000 events. The rechargeable lead-acid battery will power the system for 110 days. The information stored can be retrieved with a laptop PC or the optional RainWise Data Sponge. The RainWise Data Sponge is a handheld battery operated recorder. A built-in microprocessor with an on-board RAM allows it to transfer data to a PC through the serial port or directly drive a dot matrix printer.

4.8 Integration of the Real – Time Experiments

The four Real - time Experiments are integrated using the State Machine VI architecture of LabVIEW. The State Machine VI architecture is a method for controlling the execution of VIs in a nonlinear fashion. This programming technique is very useful in VIs that are easily split into several simpler tasks such as VIs that act as user interface. Using the state Machine VI architecture the block diagrams can be made more compact by using a single case structure to handle all the events.

A state machine in LabVIEW can be created using a while Loop, a Case structure and a shift register. Each state of the state machine is a case in the Case structure. The VIs and the other code that the state should execute are placed within the appropriate case. A shift register stores the state to be executed upon the next iteration of the loop.



State Machine VI architecture

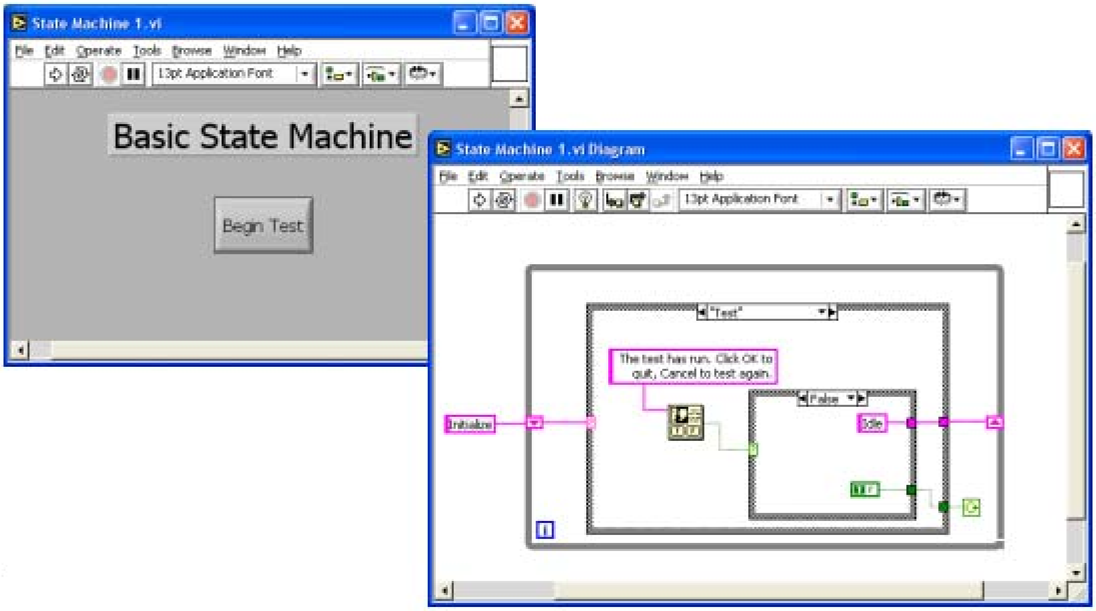

State Machine 1.vi
File Edit Operate Tools Browse Window Help
13pt Application Font
Basic State Machine
Begin Test
State Machine 1.vi Diagram
File Edit Operate Tools Browse Window Help
13pt Application Font
"Test"
The test has run. Click OK to quit, Cancel to test again.
False
Initialize
Idle

Figure 4.21: Block Diagram of the State Machine VI architecture

In this architecture the list of possible events, or states are designed and the then they are mapped to each case. For the VI in the previous block diagram the possible states are startup, Idle, event 1,event 2 and shutdown. The states are stored in an enumerated constant. Each state has its own case where the appropriate nodes are placed.

In this thesis project the four events are

(i). Automated Mixing Controller

(ii). Digital storage Oscilloscope Experiment

(iii). Temperature Controller Experiment and

(iv). Rain/Precipitation recording Gauge

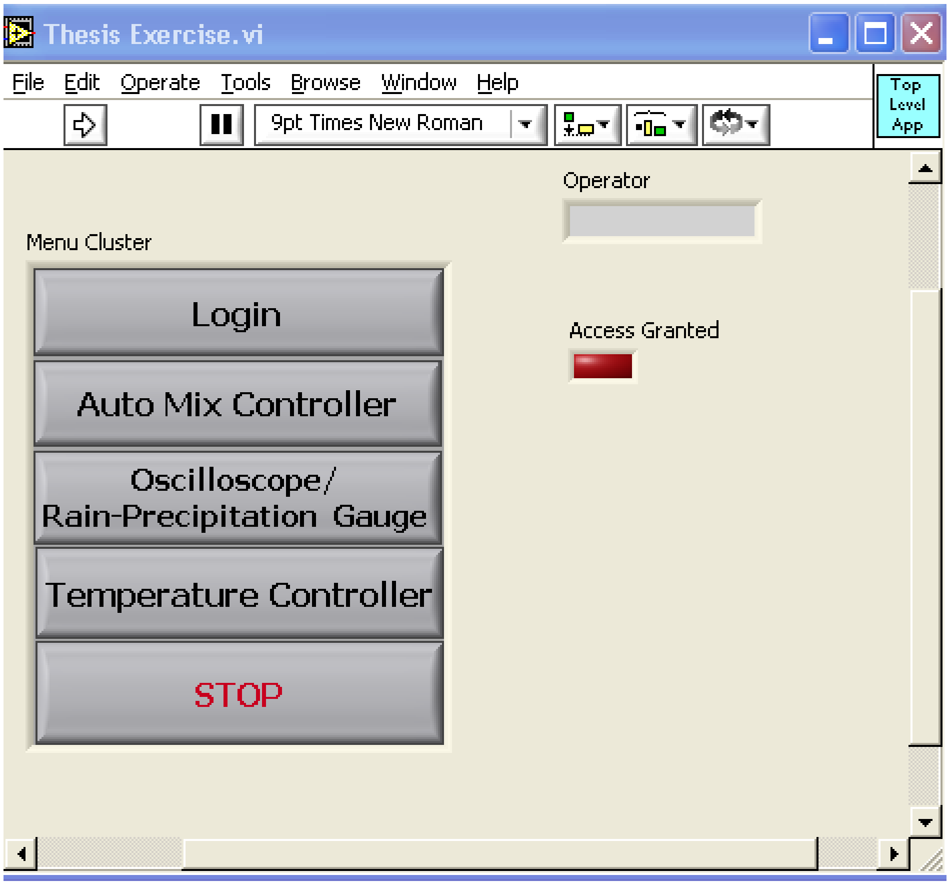


Figure 4.22: Front Panel of the Integrated Real - time Application

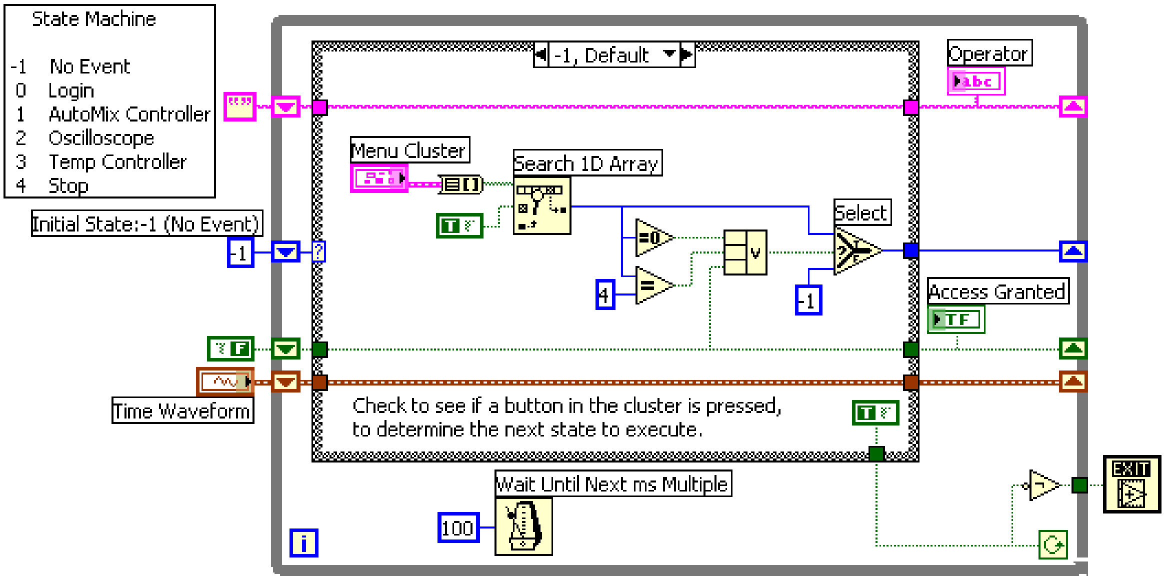
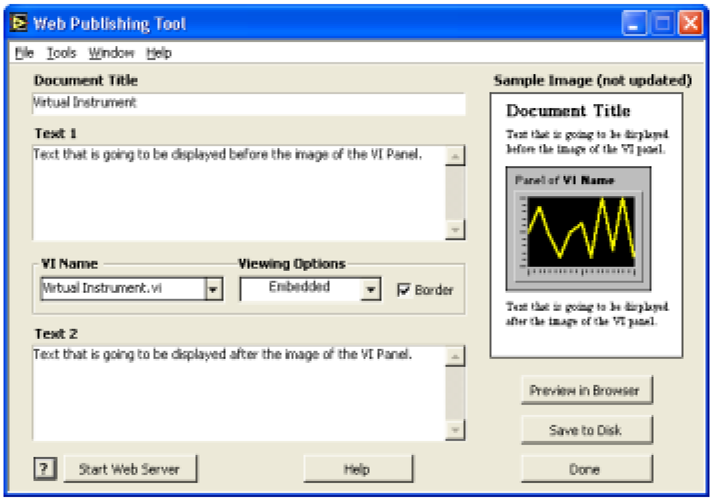


Figure 4.23: Block Diagram of the State Machine used for the Real – time Application

The advantage of the State Machine VI architecture is that the block diagram can become much smaller making it easier to read and debug. A disadvantage of the state machine VI architecture is that if two events occur at the same time, it handles only the first one and the second one is lost. This can lead to errors that are difficult to debug because they can occur only occasionally. More complex versions of the State Machine VI architecture contain extra code that builds a queue of events, or states so that the user does not miss any event.

4.9 Remote Panels

The Integrated Real-time application is provided with Internet access using the Remote Panel technology available in LabVIEW 6.1 version. Using this technology it is possible to view and control a VI front panel remotely either from within LabVIEW or from within a Web browser, by connecting to the LabVIEW built-in Web Server. When a front panel is opened remotely from a client, the Web Server sends the front panel to the client, but the block diagram and all the sub VIs remain on the server computer. The User can interact with the front panel in the same way as if the VIs were running on the client except the block diagram executes on the server. Remote clients can see live front Panel updates using this technology. Also multiple clients can view the same panel simultaneously. But only one client can control the front panel at a time. Using the Web Publishing tool the LabVIEW VI is embedded into an HTML file. This file can be opened and customized in any HTML editor.



Document Title: Virtual Instrument

Text 1: Text that is going to be displayed before the image of the VI Panel.

VI Name: Virtual Instrument.vi

Viewing Options: Embedded ☑ Border

Text 2: Text that is going to be displayed after the image of the VI Panel.

Document Title

Text that is going to be displayed before the image of the VI panel.

Text that is going to be displayed after the image of the VI panel.

Figure 4.24: LabVIEW Web Publishing Tool

5. TESTING

As explained above all the four Experiments are developed using LabVIEW 6.1 using different software technologies. Finally all the above four experiments are integrated using the State Machine VI architecture, User Interface Event programming and Remote Panel technology. All the four experiments are clustered into one application and are run on event based method. The following is the complete Integrated Application of all the real – time experiments mentioned above.

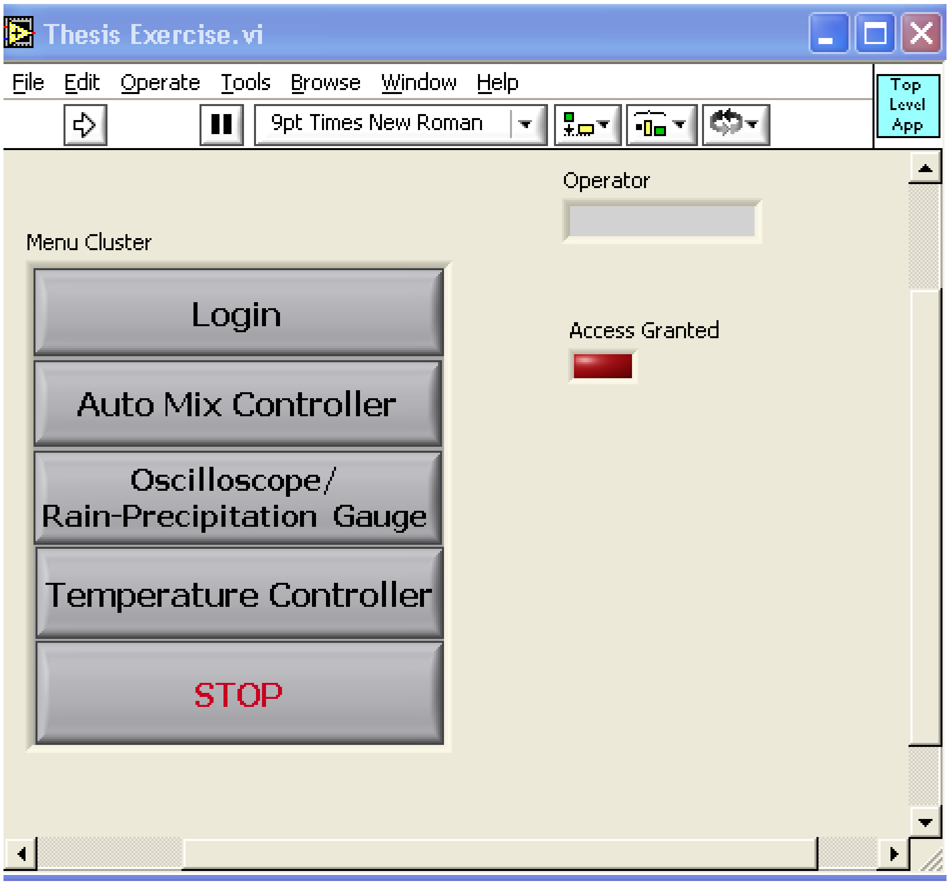
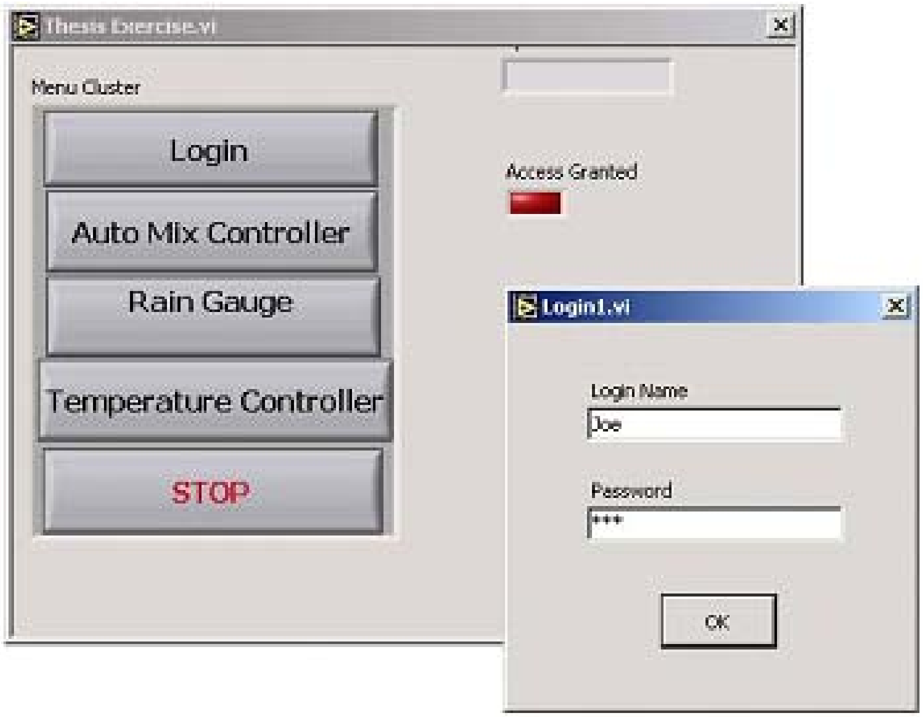


Figure 5.1 : Front Panel of the Integrated Application

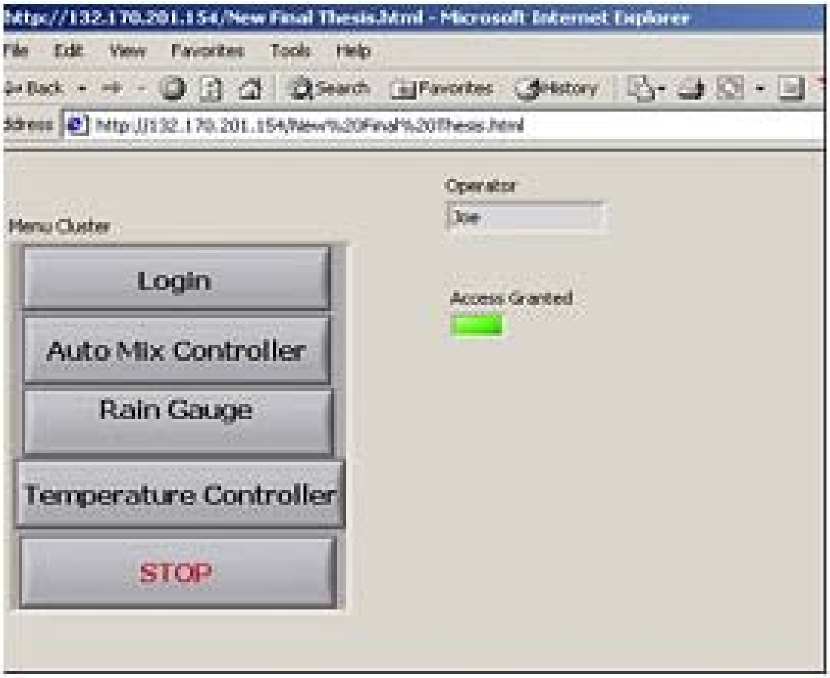


Login Name
Joe

Password
***

Figure 5.2 : Screen shot of the Application when executed – Prompts for User Name

When the Integrated application is executed the inbuilt Login VI, Prompts for the Login Name and Password. This facility is developed to ensure security of the application and safe guard the access to the applications.

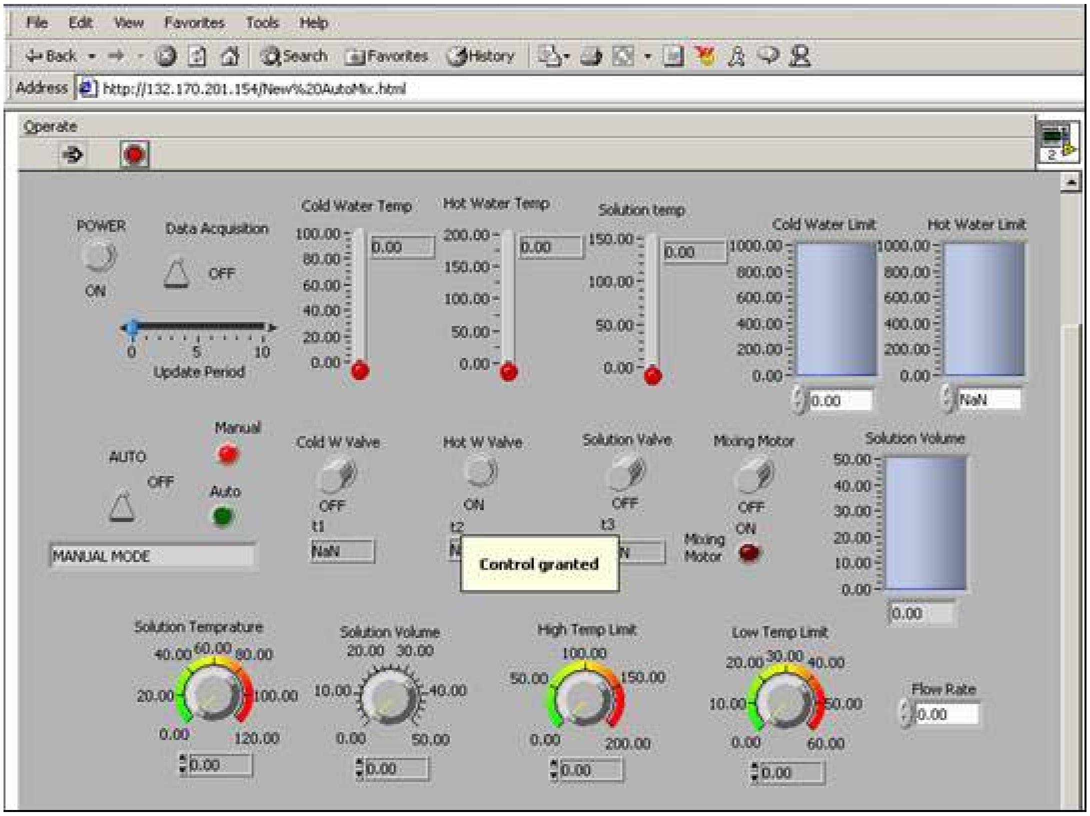

http://132.170.201.154/New Final Thesis.html - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Menu Cluster
Login
Auto Mix Controller
Rain Gauge
Temperature Controller
STOP
Access Granted

Address http://132.170.201.154/New%20Final%20Thesis.html

Operator: Joe

Figure 5.3 : Remote Panel of the Integrated Application

The above figure shows that the Access is granted and the remote panel of the Application being accessed from a remote location. Now the User can choose one of the applications and execute them. The following is a screen shot of the remote panel of the application when the Automatic Mixing Controller Experiment is executed.



Address http://132.170.201.154/New%20AutoMix.html

Cold Water Temp: 0.00
Hot Water Temp: 0.00
Solution temp: 0.00
Cold Water Limit: 0.00
Hot Water Limit: NaN
MANUAL MODE
t1: NaN
Solution Volume: 0.00
Solution Temprature: 0.00
Solution Volume: 0.00
High Temp Limit: 0.00
Low Temp Limit: 0.00
Flow Rate: 0.00

Control granted

Figure 5.4 : Screen shot of the program when accessed remotely and has requested for Control

Screen shot of the remote panel of the integrated application executing the Automatic Mixing Controller Experiment.

6. CONCLUSIONS

In the implementation of these experiments, LabVIEW's ability for data acquisition and possibility in networking are explored. LabVIEW can communicate with instruments via the GPIB board, DAQ card, or the serial port. All of them are used in the experiments successfully. Which one to use in which instrument depends on the types of the signals, data size and transmission speed and instrument communication interface. The main objective of this thesis, which is to integrate the real - time applications and provide remote access to the application is successfully achieved. The state machine VI architecture is used to integrate the real – time applications. The State Machine VI architecture is a method for controlling the execution of VIs in a nonlinear fashion.

There are various ways through which LabVIEW VIs can be controlled over the Internet: CGI, ActiveX, Java applets and Remote Panels. Each one has its own advantages and disadvantages. In this Thesis project the Remote Panel technology is used to access the application over Internet. In addition, the LabVIEW VI can be controlled remotely and locally at the same time. The front panel will be updated whenever there is controls change, no matter if it is from the remote user or local user. The remote user interface (the HTML file) is easier to write with Remote Panels Technique using Web publishing tool. The data is transferred via the Data Socket between the LabVIEW VI and the Remote Panel control. Remote panels are useful in cases when users will need to make measurements remotely or when many users need to have access to the same equipment. The successful execution of the Integrated Real – time experiment emphasizes the use of LabVIEW in developing applications that can be used for distance learning, Remote Control Labs and Monitoring.

APPENDIX A: Front Panels of the Integrated Real – Time Experiments

The Integrated Real-Time Application

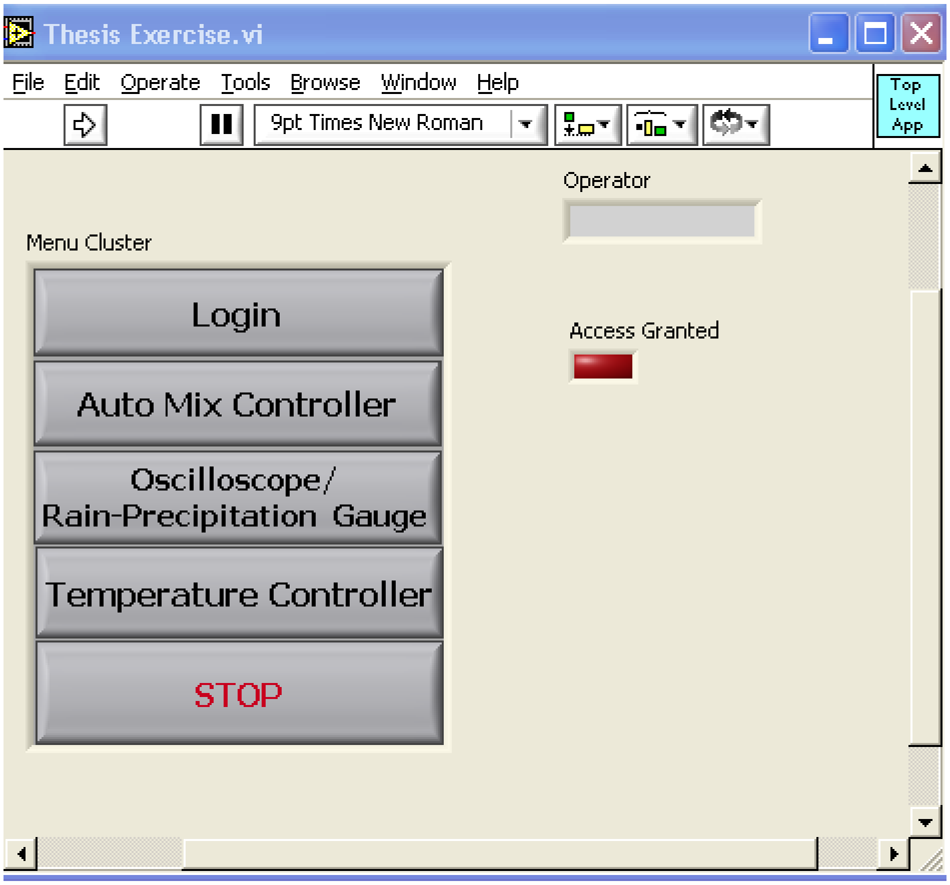


Figure A.1: The front panel for the Labview VI of the Integrated Real-Time Application

Automatic Mixing Controller Application

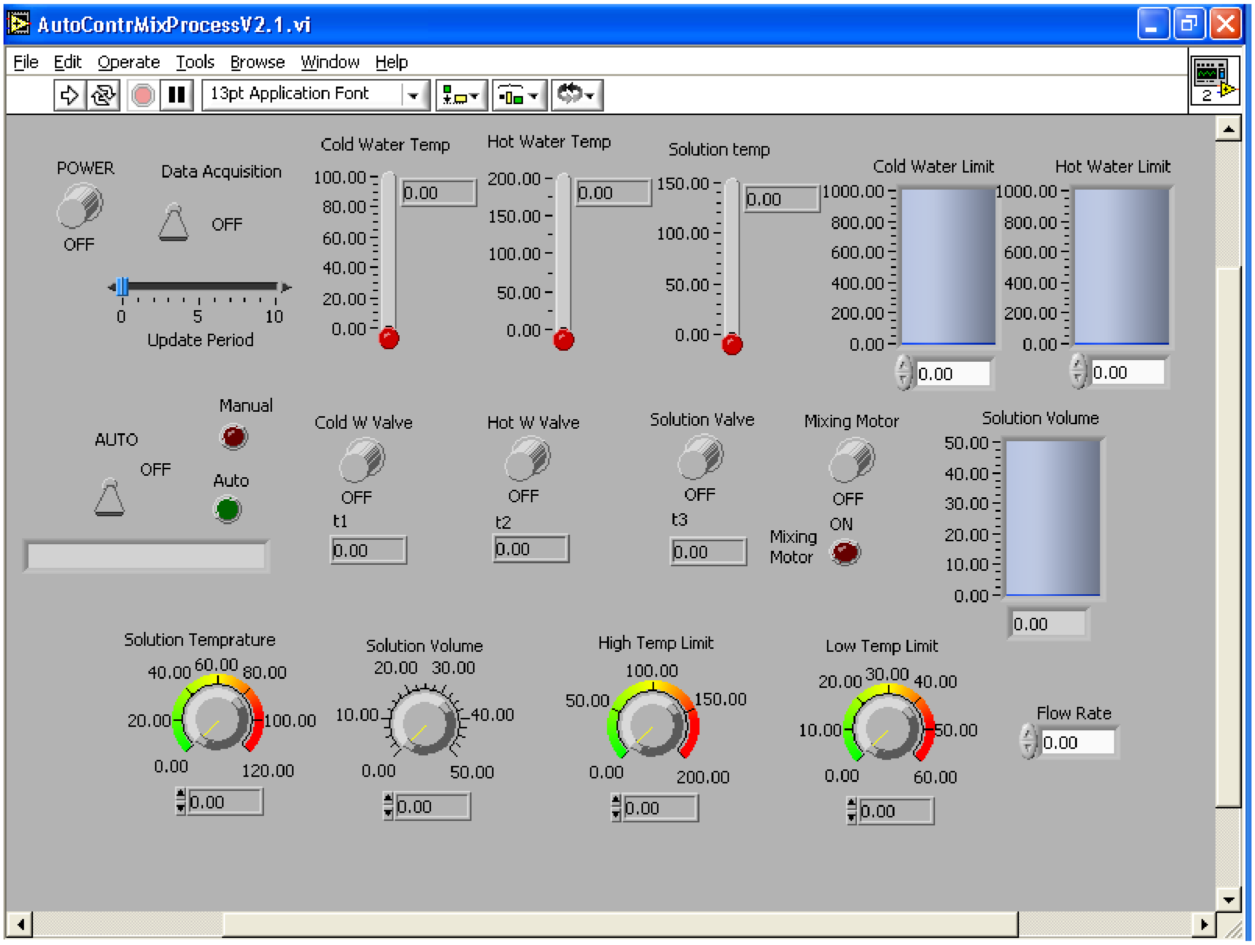
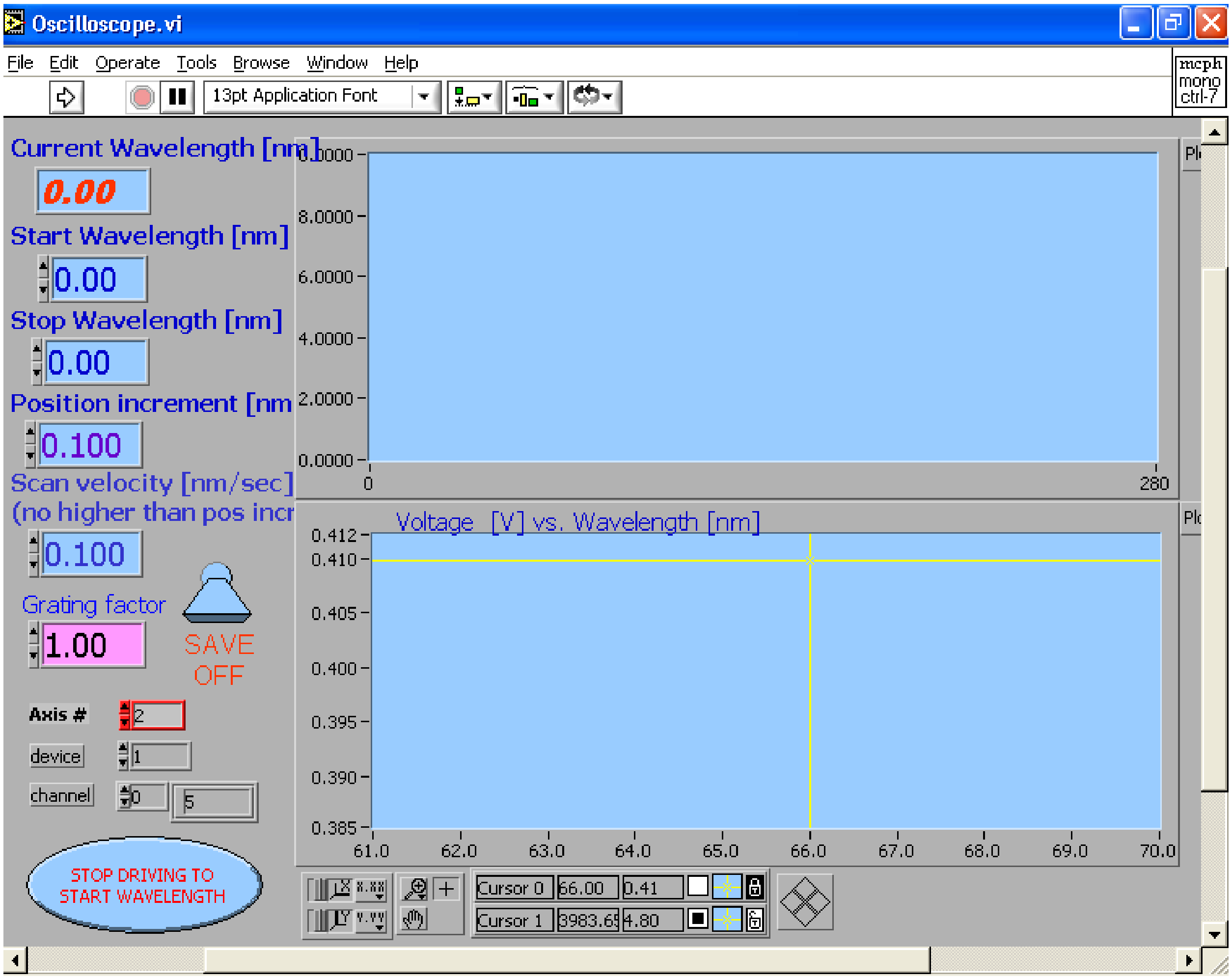


Figure A.2: The front panel for the Labview VI of the Automatic Mixing Controller Application

Digital storage Oscilloscope Application



Current Wavelength [nm] 0.00

Start Wavelength [nm] 0.00

Stop Wavelength [nm] 0.00

Position increment [nm 0.100

Scan velocity [nm/sec] (no higher than pos incr 0.100

Grating factor 1.00

Axis # 2

device 1

channel 0 5

Cursor 0 66.00 0.41

Cursor 1 3983.69 4.80

Figure A.3: The front panel for the Labview VI of the Digital storage Oscilloscope Application

Temperature Controller Application

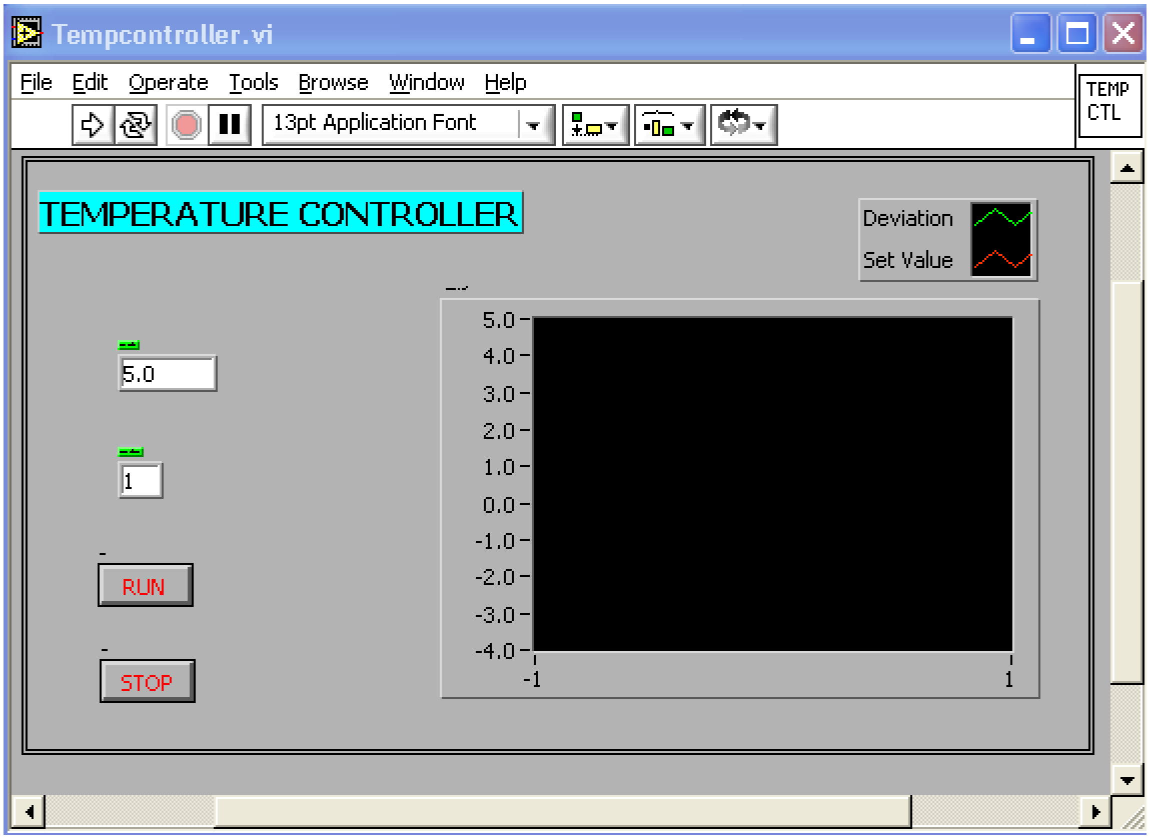


Figure A.4: The front panel for the Labview VI of the Temperature Controller

Rain/Precipitation Gauge Application

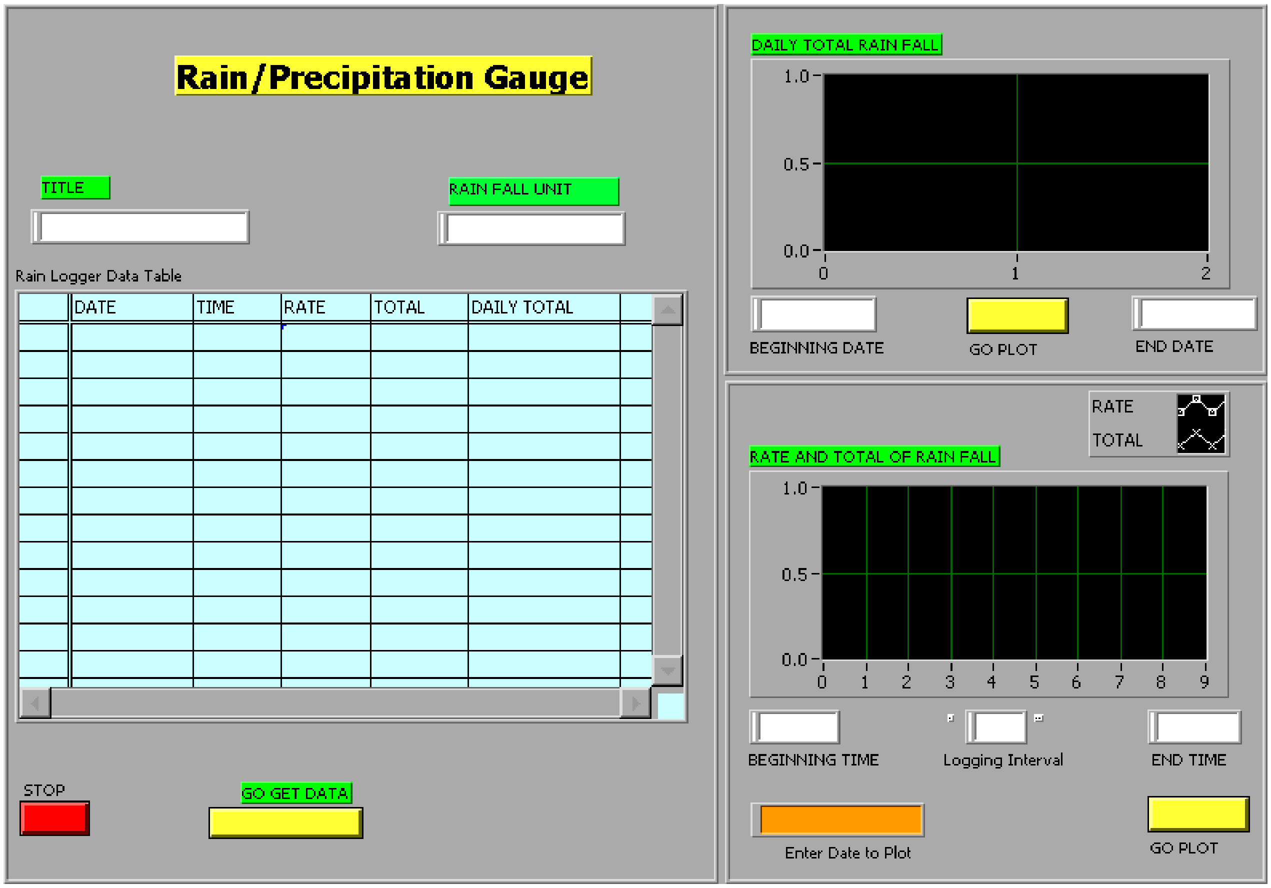


Figure A.5: The front panel for the Labview VI of the Rain / Precipitation Gauge Application

APPENDIX B: Source Code for Remote Panels

* Remote Panel for Integrated Real Time Application - Thesis Exercise

The following is the html code for the Remote Panel

```
<OBJECT ID="LabVIEWControl"
CLASSID="CLSID:A40B0AD4-B50E-4E58-8A1D-8544233807AA"
WIDTH=650 HEIGHT=500>
<PARAM name="LVFPPVINAME" value="Thesis Exercise.vi">
<PARAM name="server" value="http://132.170.201.154">
<EMBED SRC="http://132.170.201.154/.LV_FrontPanelProotocol.rpvi"
LVFPPVINAME="Thesis Exercise.vi"
TYPE="application/x-labviewrpvi" WIDTH=650
HEIGHT=500>
</EMBED>
</OBJECT>
```

* Remote Panel for Automix Controller

The following is the html code for the Remote Panel

```
<OBJECT ID="LabVIEWControl"
CLASSID="CLSID:A40B0AD4-B50E-4E58-8A1D-8544233807AA"
WIDTH=650 HEIGHT=500>
<PARAM name="LVFPPVINAME" value="AutoContrMixProcessV2.1.vi">
<PARAM name="server" value="http://132.170.201.154">
<EMBED SRC="http://132.170.201.154/.LV_FrontPanelProotocol.rpvi"
LVFPPVINAME="AutoContrMixProcessV2.1.vi"
```

```
TYPE="application/x-labviewrpvi" WIDTH=650
HEIGHT=500>
</EMBED>
</OBJECT>
```

* Remote Panel for Digital Oscilloscope

The following is the html code for the Remote Panel

```
<OBJECT ID="LabVIEWControl"
CLASSID="CLSID:A40B0AD4-B50E-4E58-8A1D-8544233807AA"
WIDTH=650 HEIGHT=500>
<PARAM name="LVFPPVINAME" value="Oscilloscope.vi">
<PARAM name="server" value="http://132.170.201.154">
<EMBED SRC="http://132.170.201.154/.LV_FrontPanelProotocol.rpvi"
LVFPPVINAME="Oscilloscope.vi"
TYPE="application/x-labviewrpvi" WIDTH=650
HEIGHT=500>
</EMBED>
</OBJECT>
```

* Remote Panel for Temperature Controller

The following is the html code for the Remote Panel

```
<OBJECT ID="LabVIEWControl"
CLASSID="CLSID:A40B0AD4-B50E-4E58-8A1D-8544233807AA"
```

```
WIDTH=650 HEIGHT=500>
<PARAM name="LVFPPVINAME" value="Tempcontroller.vi">
<PARAM name="server" value="http://132.170.201.154">
<EMBED SRC="http://132.170.201.154/.LV_FrontPanelProotocol.rpvi"
LVFPPVINAME="Tempcontroller.vi"
TYPE="application/x-labviewrpvi" WIDTH=650
HEIGHT=500>
</EMBED>
</OBJECT>
```

* Remote Panel for Rain/Precipitation Gauge

The following is the html code for the Remote Panel

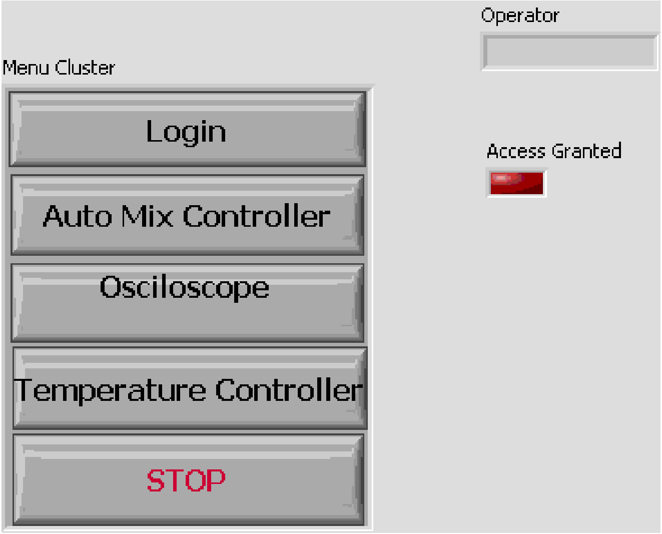
```
<OBJECT ID="LabVIEWControl"
CLASSID="CLSID:A40B0AD4-B50E-4E58-8A1D-8544233807AA"
WIDTH=650 HEIGHT=500>
<PARAM name="LVFPPVINAME" value="Rain-Precipitation Gauge.vi">
<PARAM name="server" value="http://132.170.201.154">
<EMBED SRC="http://132.170.201.154/.LV_FrontPanelProotocol.rpvi"
LVFPPVINAME="Rain-Precipitation Gauge.vi"
TYPE="application/x-labviewrpvi" WIDTH=650
HEIGHT=500>
</EMBED>
</OBJECT>
```

APPENDIX C: LabVIEW Source Code for Integrated Application

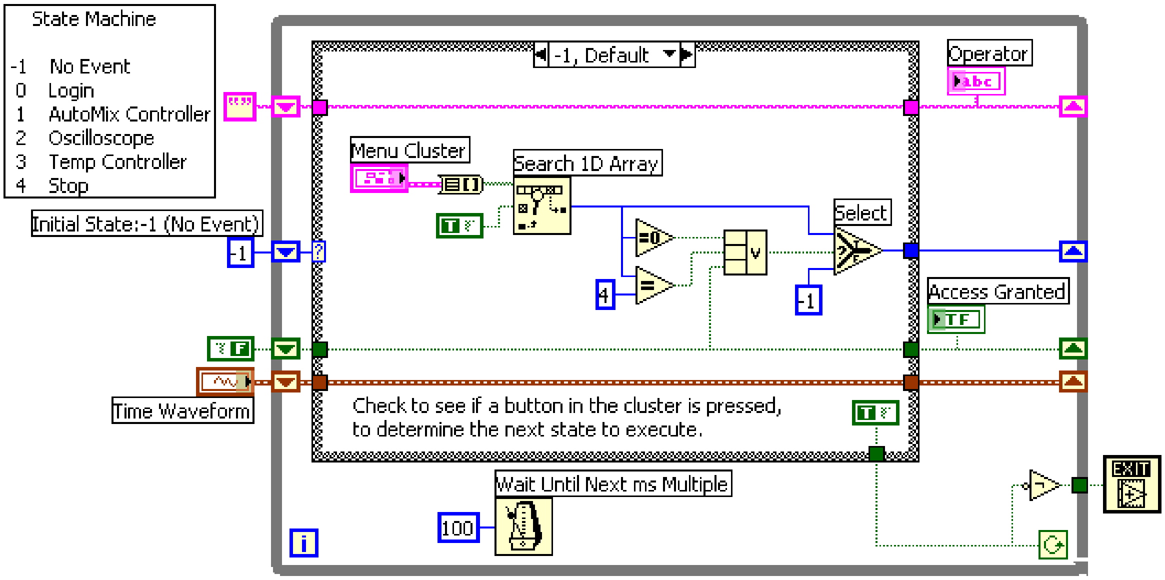
Labview Code for Thesis Exercise.vi

Connector Pane

Front Panel



Block Diagram


State Machine

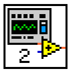
-1 No Event
0 Login
1 AutoMix Controller
2 Oscilloscope
3 Temp Controller
4 Stop

Initial State:-1 (No Event)

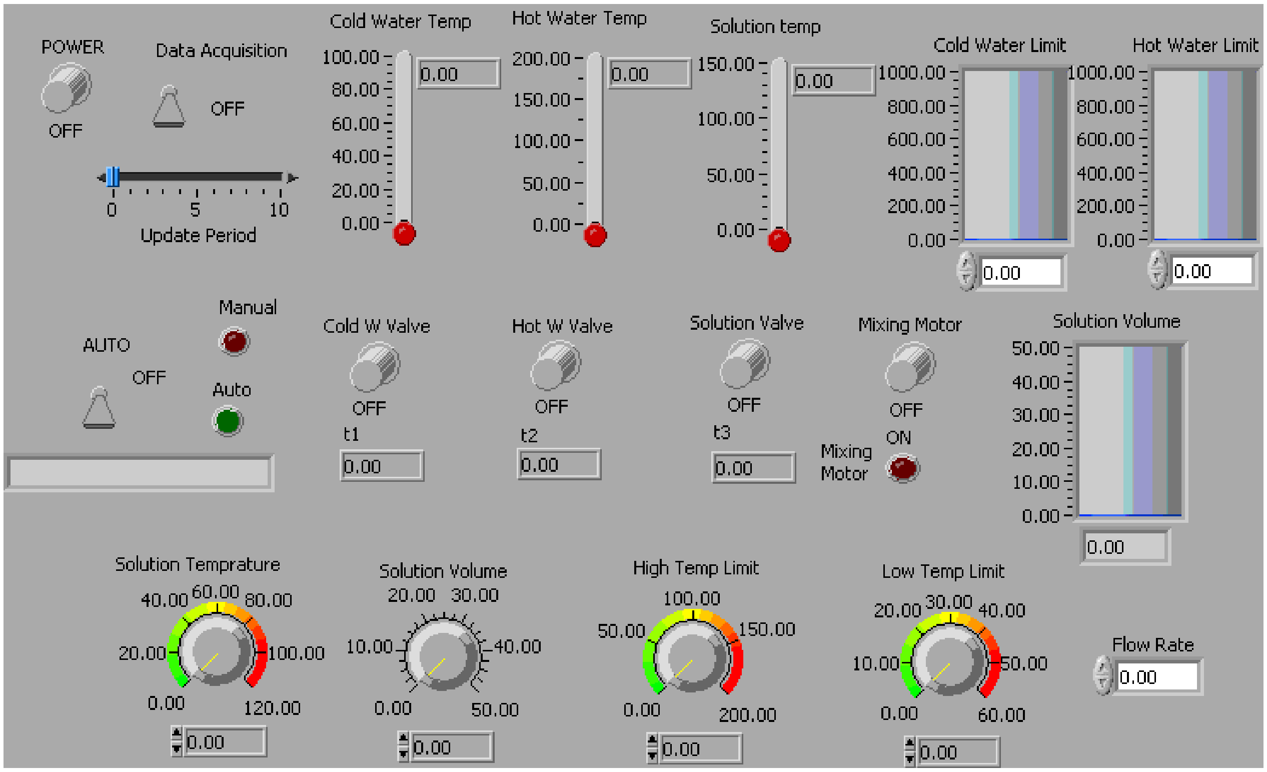
Check to see if a button in the cluster is pressed, to determine the next state to execute.

AutoContrMixProcessV2.1.vi

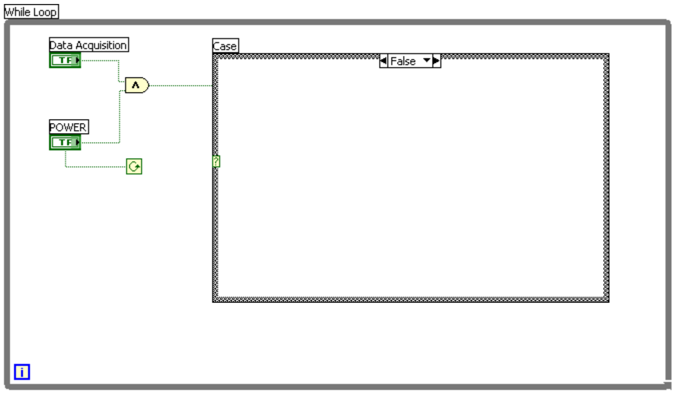
Connector Pane

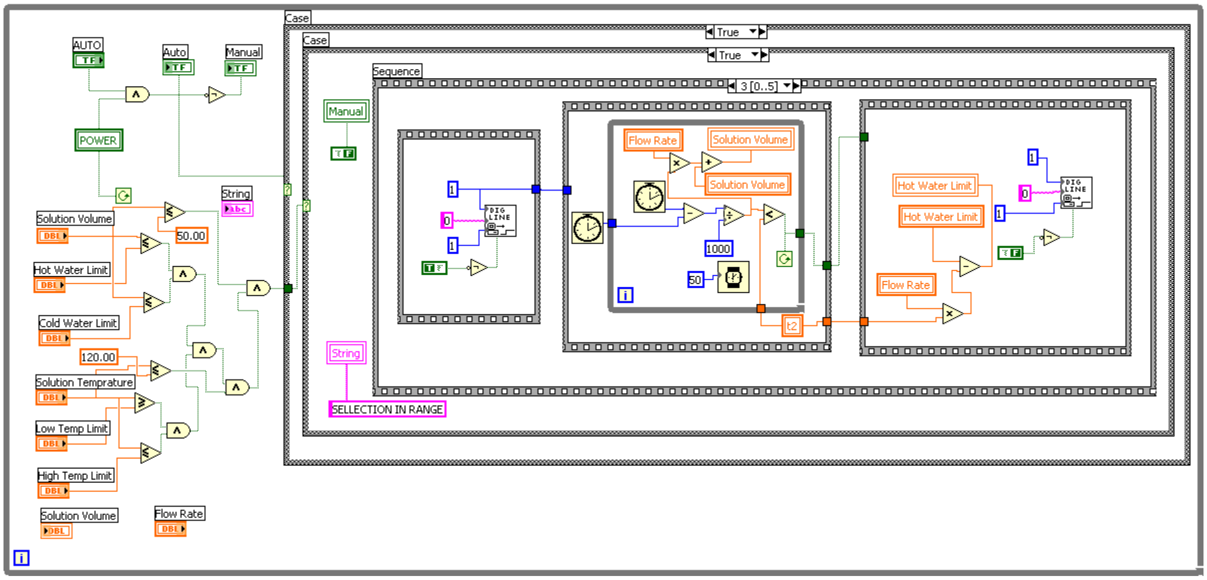


Front Panel



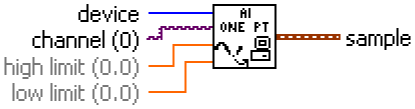
Block Diagram





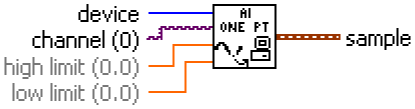
AI Sample Channel.vi

Connector Pane

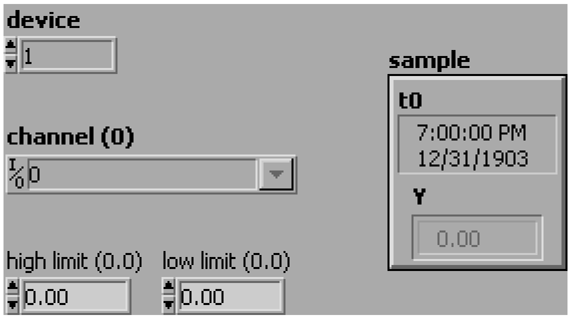


AI Sample Channel (single-point waveform).vi

Connector Pane



Front Panel



device
1

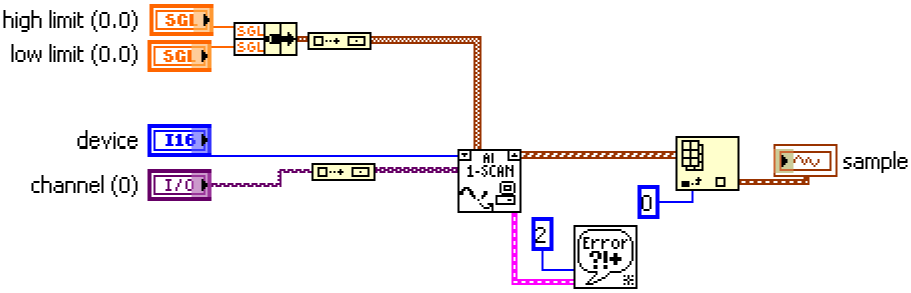
channel (0)
0

high limit (0.0)
0.00

low limit (0.0)
0.00

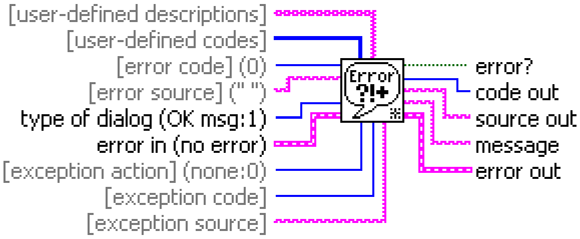
sample
t0
7:00:00 PM
12/31/1903
Y
0.00

Block Diagram

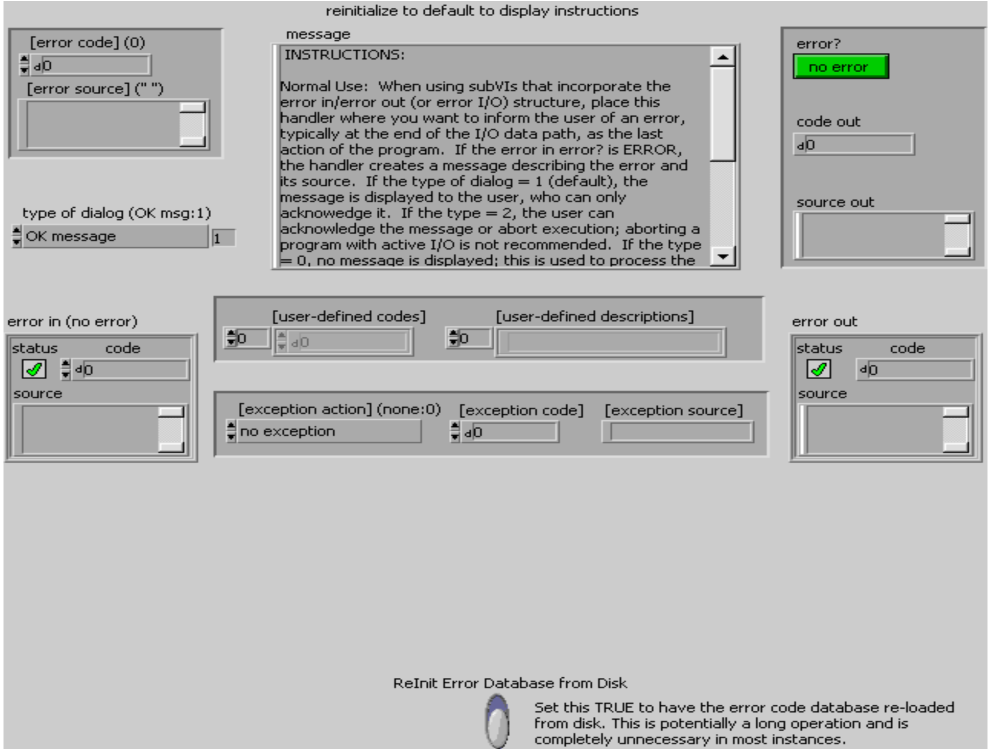


General Error Handler.vi

Connector Pane



Front Panel



reinitialize to default to display instructions

[error code] (0)

0

[error source] (" ")

type of dialog (OK msg:1)

OK message 1

message

INSTRUCTIONS:

Normal Use: When using subVIs that incorporate the error in/error out (or error I/O) structure, place this handler where you want to inform the user of an error, typically at the end of the I/O data path, as the last action of the program. If the error in error? is ERROR, the handler creates a message describing the error and its source. If the type of dialog = 1 (default), the message is displayed to the user, who can only acknowledge it. If the type = 2, the user can acknowledge the message or abort execution; aborting a program with active I/O is not recommended. If the type = 0, no message is displayed; this is used to process the

error?

no error

code out

0

source out

error in (no error)

status ✓ code 0

source

[user-defined codes] 0 0

[user-defined descriptions] 0

error out

status ✓ code 0

source

[exception action] (none:0)

no exception

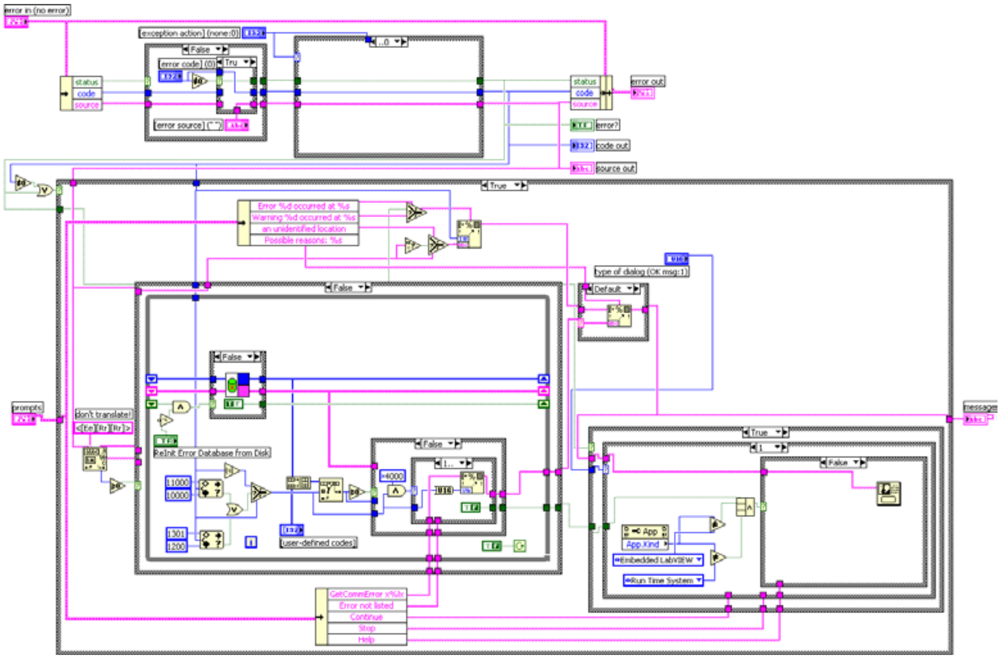
[exception code] 0

[exception source]

ReInit Error Database from Disk

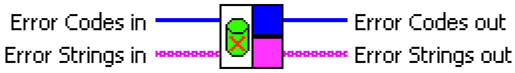
Set this TRUE to have the error code database re-loaded from disk. This is potentially a long operation and is completely unnecessary in most instances.

Block Diagram

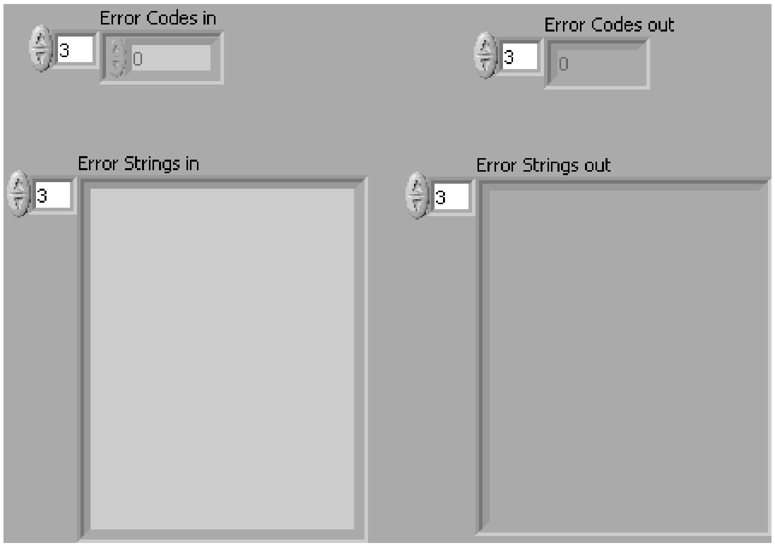


Error Code Database.vi

Connector Pane

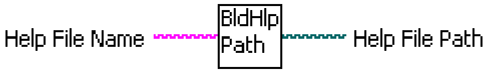


Front Panel

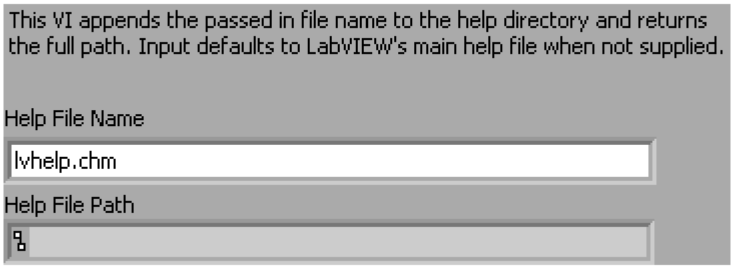


BuildHelpPath.vi

Connector Pane



Front Panel



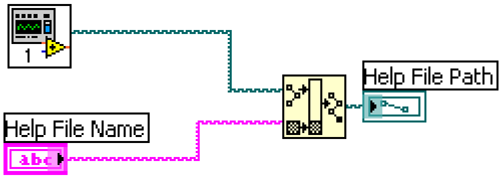
This VI appends the passed in file name to the help directory and returns the full path. Input defaults to LabVIEW's main help file when not supplied.

Help File Name

lvhelp.chm

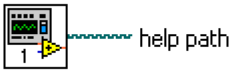
Help File Path

Block Diagram



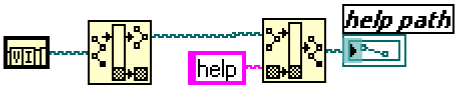
GetHelpDir.vi

Connector Pane



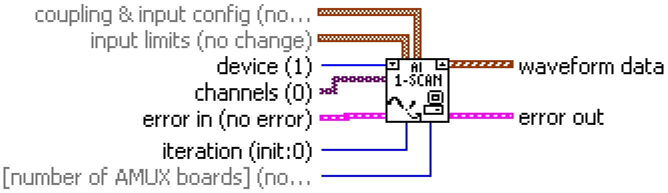
Front Panel

Block Diagram



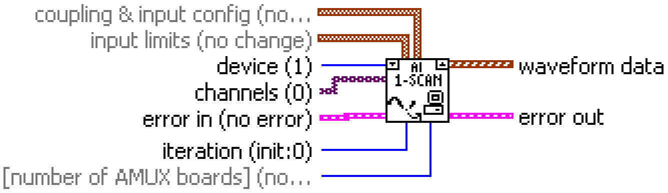
AI Read One Scan.vi

Connector Pane

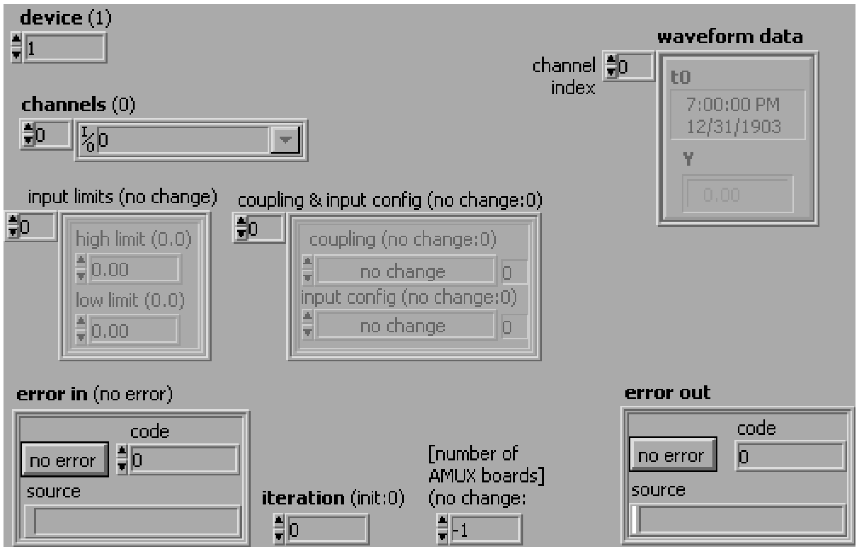


AI Read One Scan (single-point waveform).vi

Connector Pane

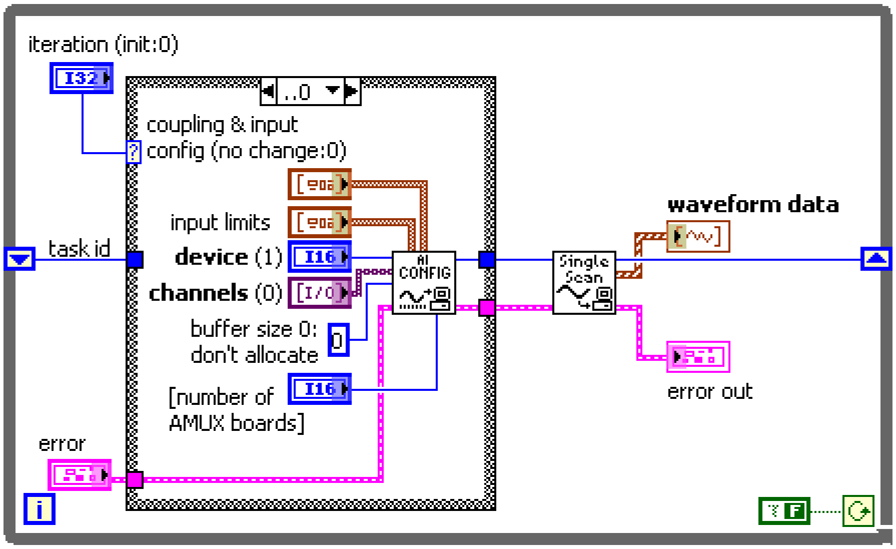


Front Panel



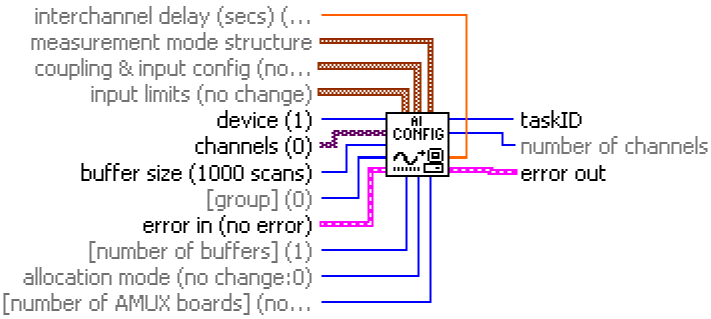
Block Diagram

This While Loop executes only once, but uses its shift register to remember **task id** from a previous call to this VI.

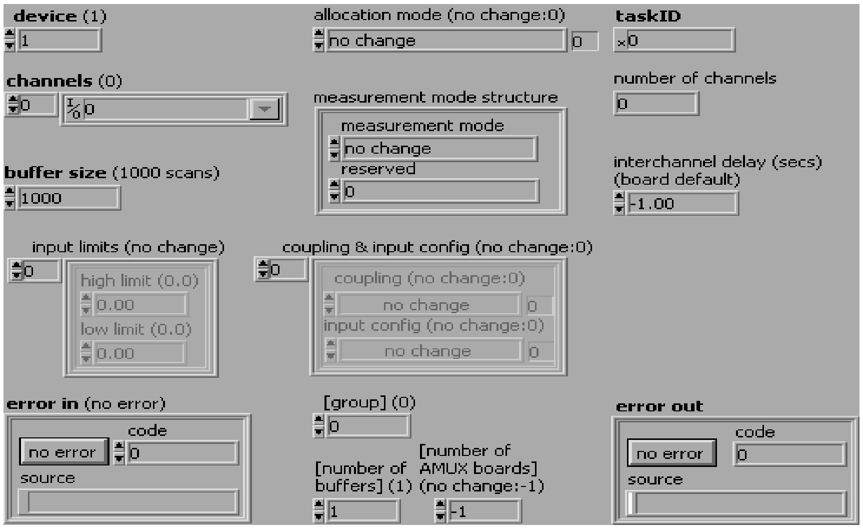


AI Config.vi

Connector Pane



Front Panel



device (1)
1

channels (0)
0 | 0

buffer size (1000 scans)
1000

allocation mode (no change:0)
no change | 0

measurement mode structure
measurement mode
no change
reserved
0

taskID
0

number of channels
0

interchannel delay (secs) (board default)
-1.00

input limits (no change)
0
high limit (0.0)
0.00
low limit (0.0)
0.00

coupling & input config (no change:0)
0
coupling (no change:0)
no change | 0
input config (no change:0)
no change | 0

error in (no error)
code
no error | 0
source

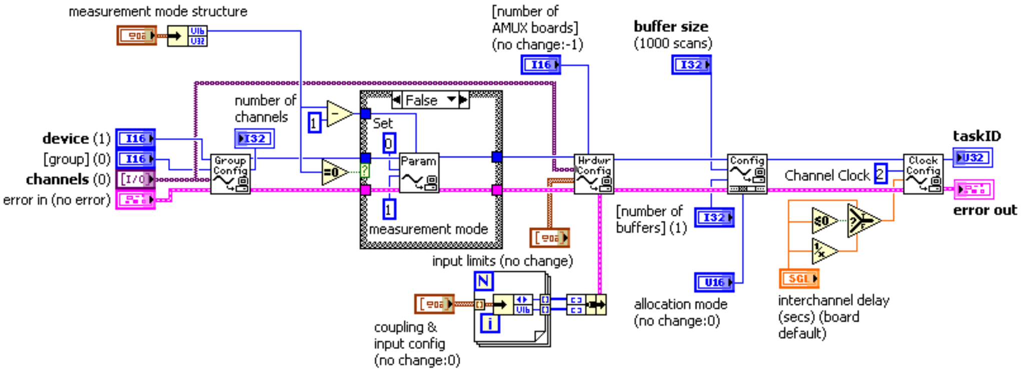
[group] (0)
0

[number of buffers] (1)
1

[number of AMUX boards] (no change:-1)
-1

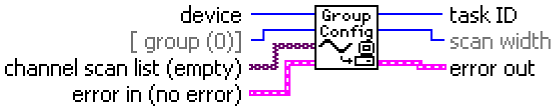
error out
code
no error | 0
source

Block Diagram

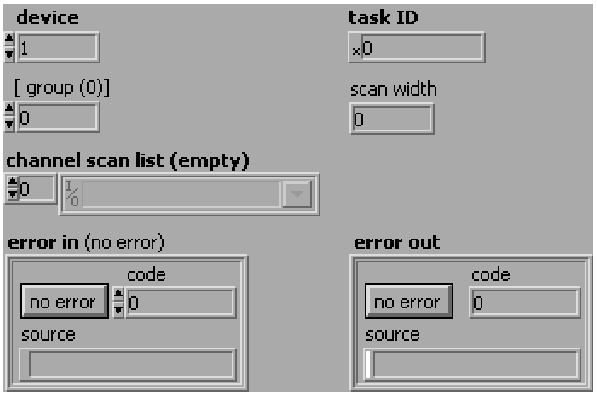


AI Group Config.vi

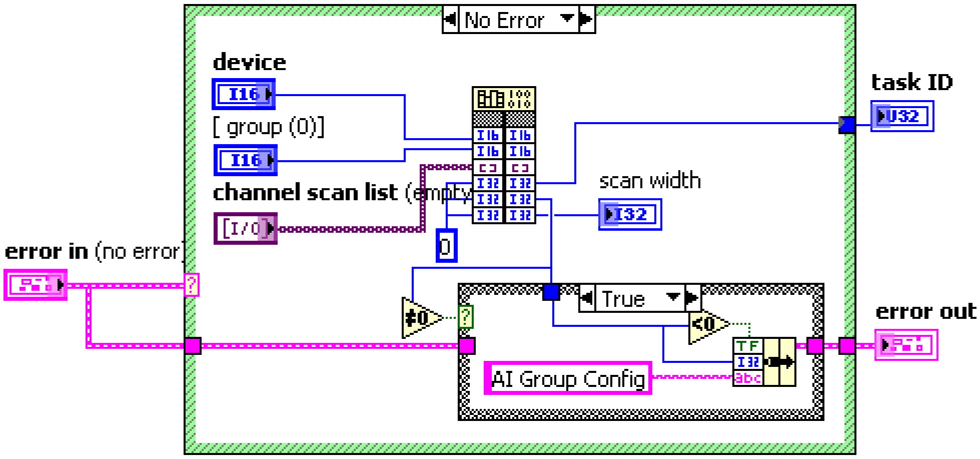
Connector Pane



Front Panel

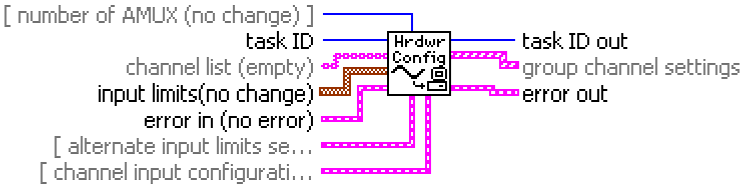

device
1
[group (0)]
0
channel scan list (empty)
0
error in (no error)
code
no error
0
source
task ID
x0
scan width
0
error out
code
no error
0
source

Block Diagram

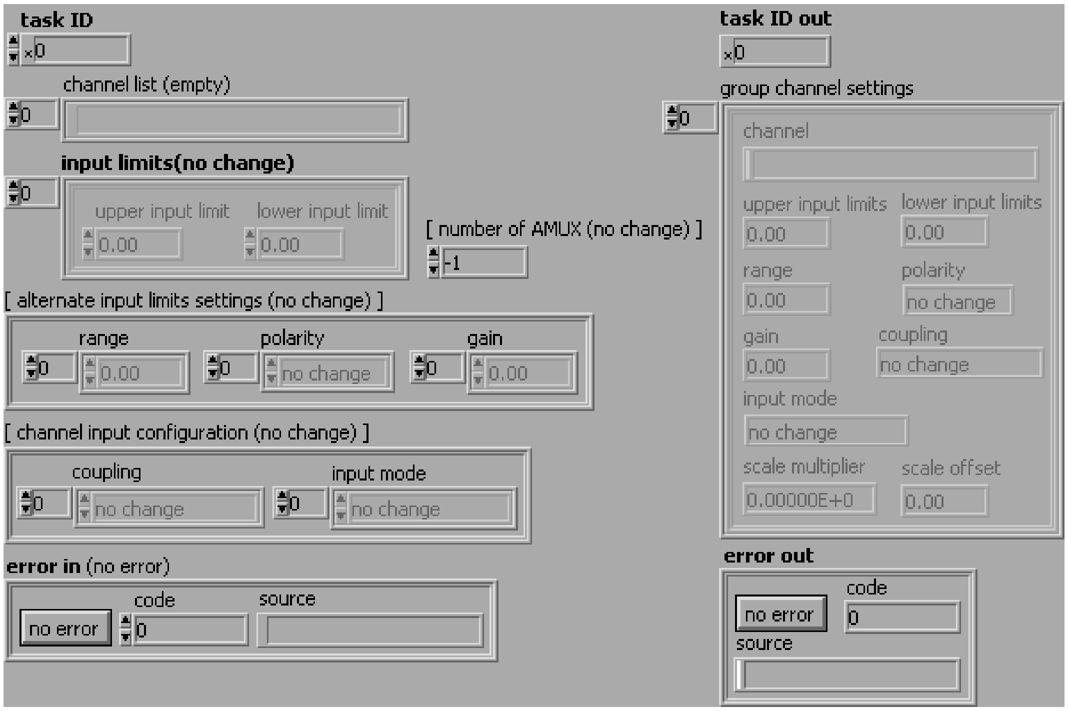


AI Hardware Config.vi

Connector Pane



Front Panel



task ID
x0

channel list (empty)
0

input limits(no change)
0
upper input limit 0.00
lower input limit 0.00

[number of AMUX (no change)]
-1

[alternate input limits settings (no change)]
range 0 0.00
polarity 0 no change
gain 0 0.00

[channel input configuration (no change)]
coupling 0 no change
input mode 0 no change

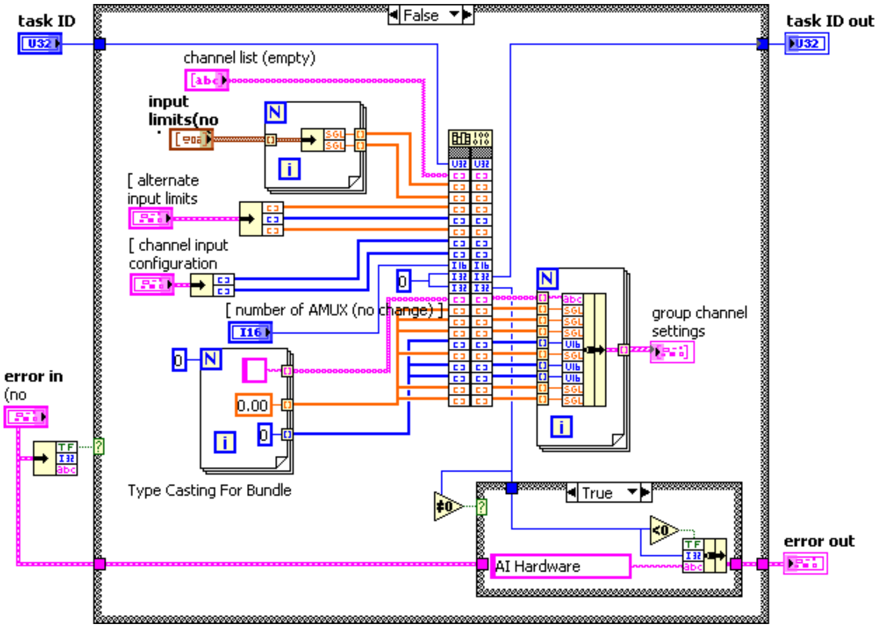
error in (no error)
no error
code 0
source

task ID out
x0

group channel settings
0
channel
upper input limits 0.00
lower input limits 0.00
range 0.00
polarity no change
gain 0.00
coupling no change
input mode no change
scale multiplier 0.00000E+0
scale offset 0.00

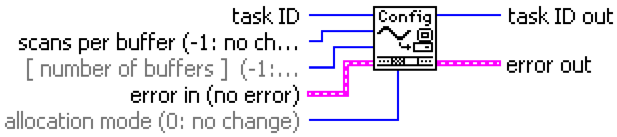
error out
no error
code 0
source

Block Diagram

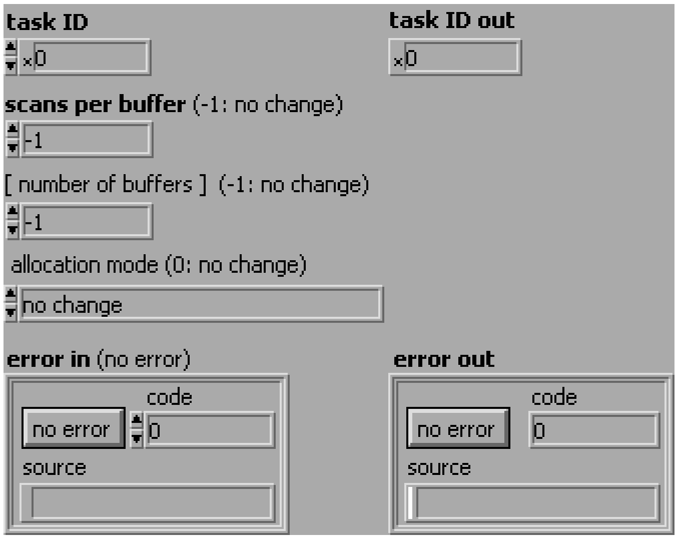


AI Buffer Config.vi

Connector Pane


task ID
scans per buffer (-1: no ch…
[number of buffers] (-1:…
error in (no error)
allocation mode (0: no change)
Config
task ID out
error out

Front Panel



task ID 0

task ID out 0

scans per buffer (-1: no change) -1

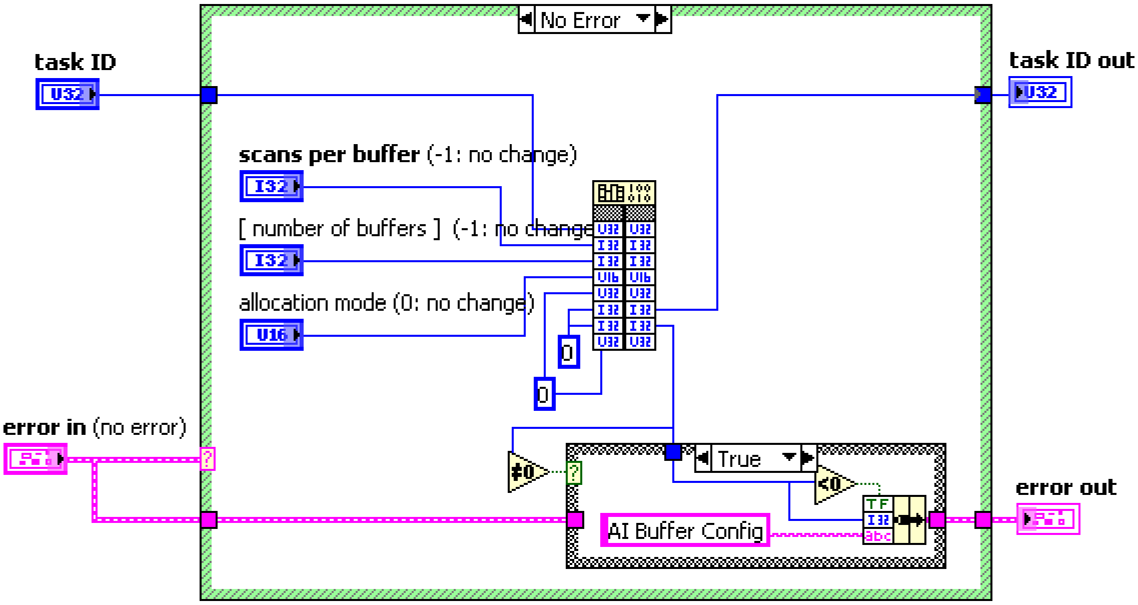
[number of buffers] (-1: no change) -1

allocation mode (0: no change) no change

error in (no error)
no error
code 0
source

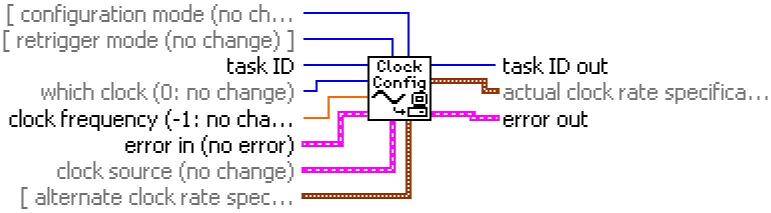
error out
no error
code 0
source

Block Diagram

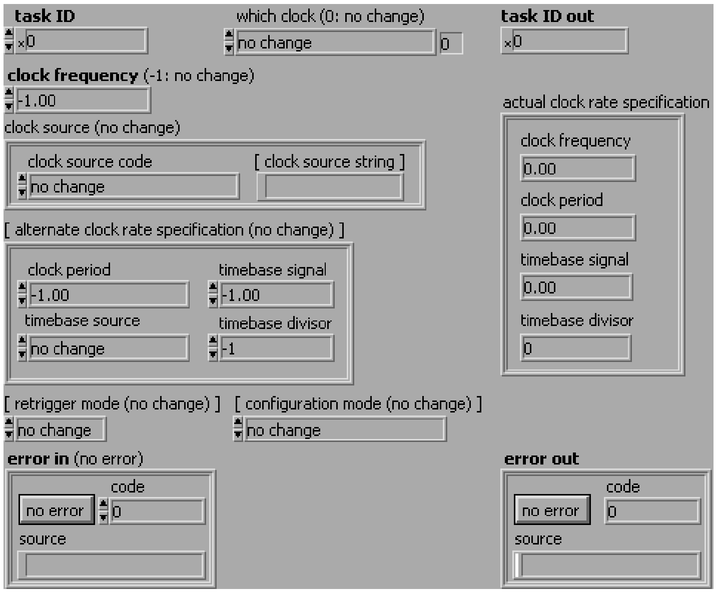


AI Clock Config.vi

Connector Pane


[configuration mode (no ch…
[retrigger mode (no change)]
task ID
which clock (0: no change)
clock frequency (-1: no cha…
error in (no error)
clock source (no change)
[alternate clock rate spec…
Clock Config
task ID out
actual clock rate specifica…
error out

Front Panel



task ID 0

which clock (0: no change) no change 0

task ID out 0

clock frequency (-1: no change) -1.00

clock source (no change)
- clock source code: no change
- [clock source string]:

actual clock rate specification
- clock frequency: 0.00
- clock period: 0.00
- timebase signal: 0.00
- timebase divisor: 0

[alternate clock rate specification (no change)]
- clock period: -1.00
- timebase signal: -1.00
- timebase source: no change
- timebase divisor: -1

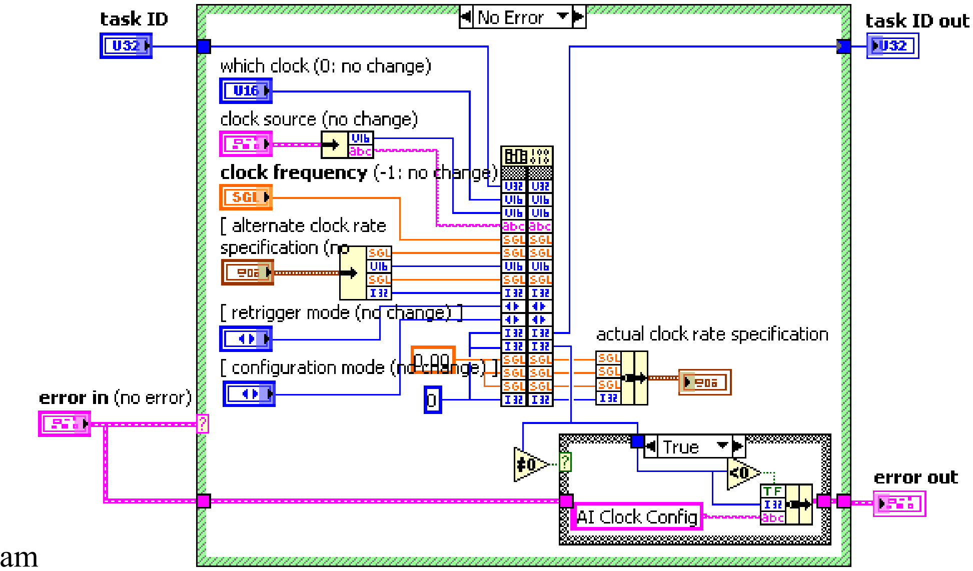
[retrigger mode (no change)] no change

[configuration mode (no change)] no change

error in (no error)
- no error
- code: 0
- source:

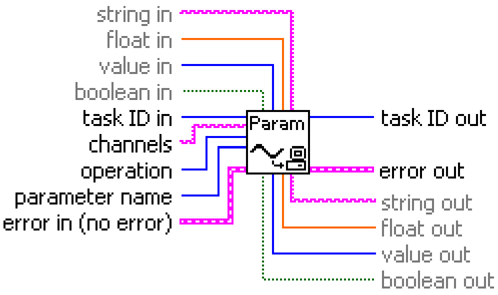
error out
- no error
- code: 0
- source:

Block Diagram

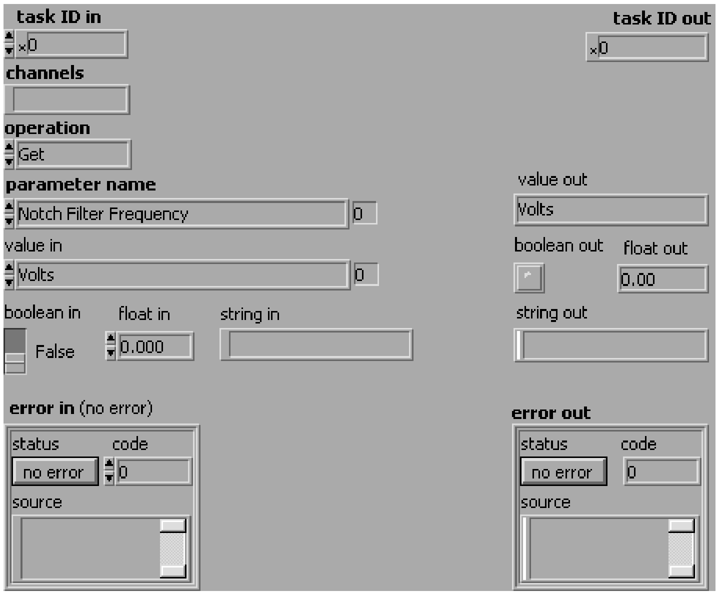

task ID
U32
No Error
task ID out
U32
which clock (0: no change)
U16
clock source (no change)
clock frequency (-1: no change)
SGL
[alternate clock rate specification (no
[retrigger mode (no change)]
[configuration mode (no change)]
0.00
0
actual clock rate specification
error in (no error)
error out
True
≠0
<0
AI Clock Config

AI Parameter.vi

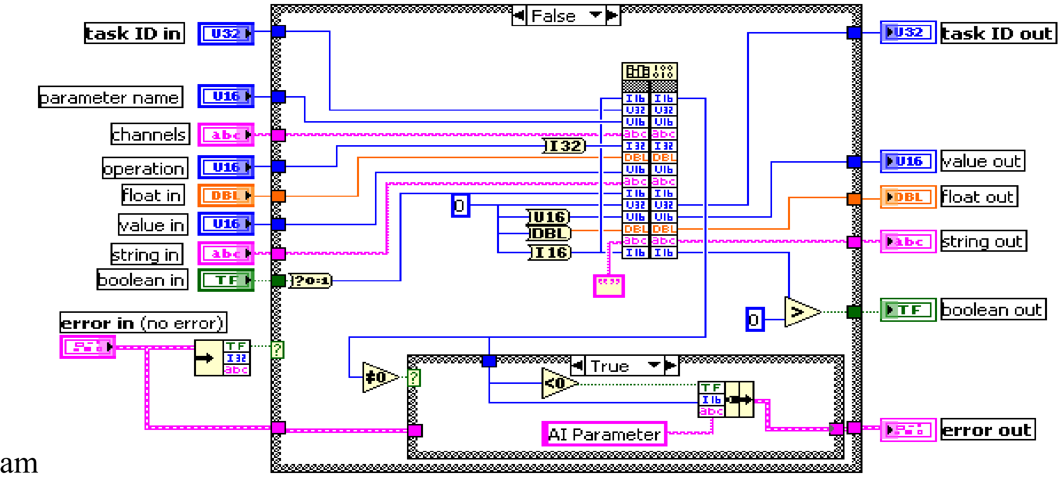
Connector Pane



Front Panel

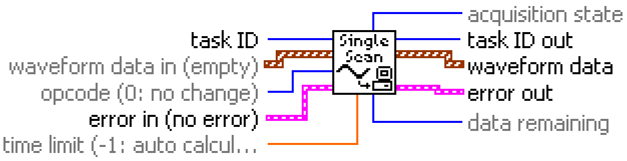

task ID in
0
task ID out
0
channels
operation
Get
parameter name
Notch Filter Frequency 0
value out
Volts
value in
Volts 0
boolean out
float out
0.00
boolean in
False
float in
0.000
string in
string out
error in (no error)
status code
no error 0
source
error out
status code
no error 0
source

Block Diagram



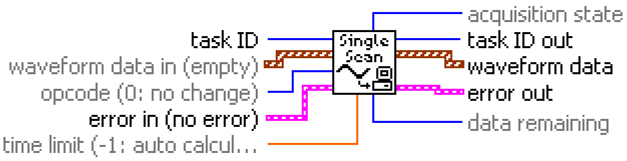
AI SingleScan.vi

Connector Pane

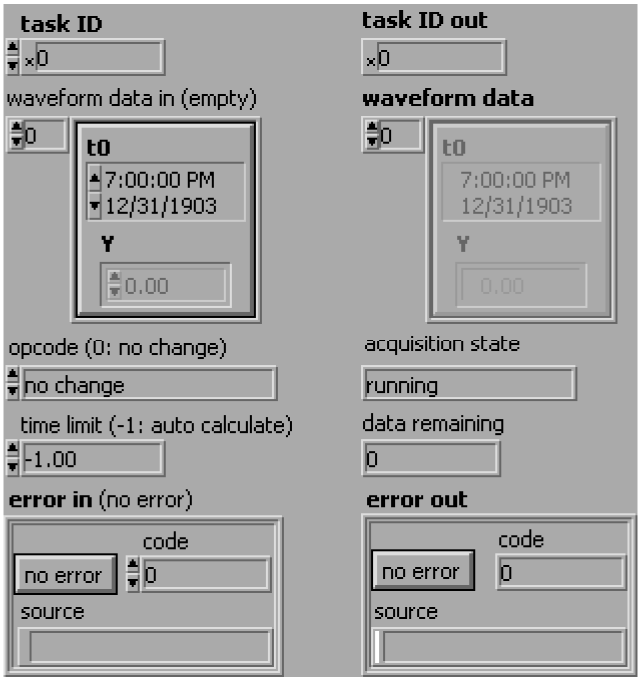


AI SingleScan (single-point waveform).vi

Connector Pane



Front Panel



task ID
0

waveform data in (empty)
0
t0
7:00:00 PM
12/31/1903
Y
0.00

opcode (0: no change)
no change

time limit (-1: auto calculate)
-1.00

error in (no error)
no error
code
0
source

task ID out
0

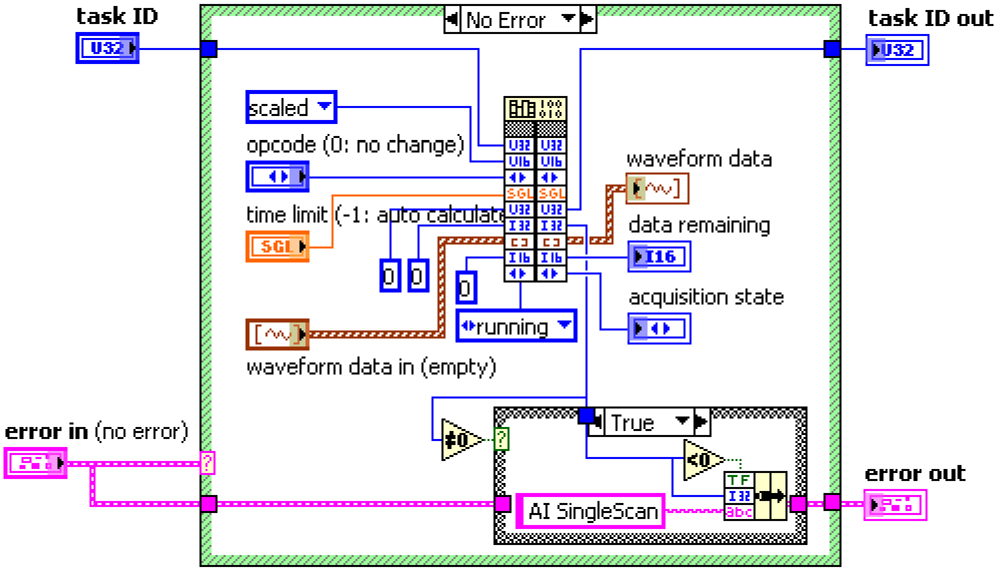
waveform data
0
t0
7:00:00 PM
12/31/1903
Y
0.00

acquisition state
running

data remaining
0

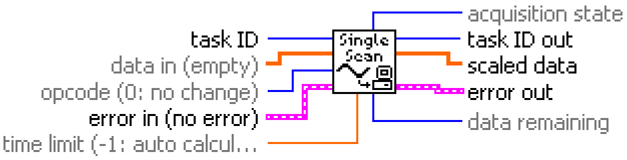
error out
no error
code
0
source

Block Diagram

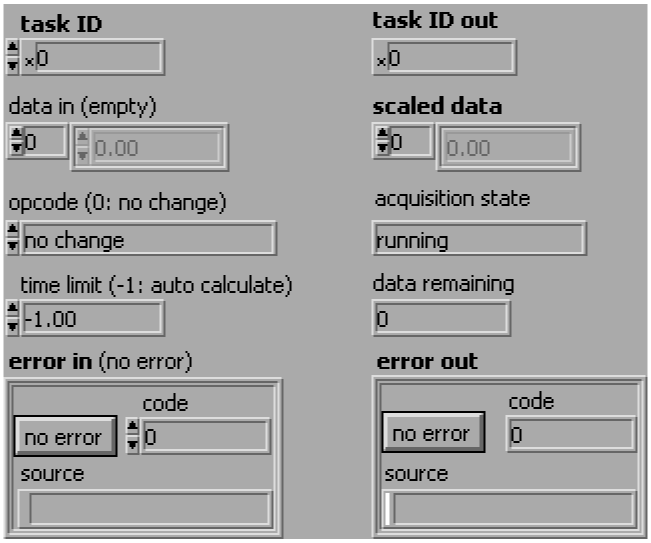


AI SingleScan (scaled array).vi

Connector Pane



Front Panel



task ID
x0

data in (empty)
0 0.00

opcode (0: no change)
no change

time limit (-1: auto calculate)
-1.00

error in (no error)
code
no error 0
source

task ID out
x0

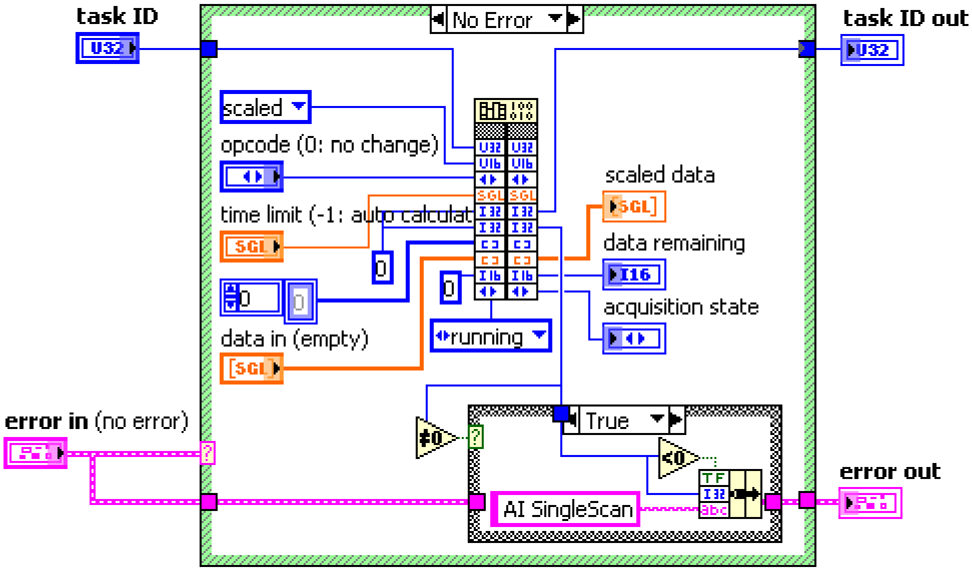
scaled data
0 0.00

acquisition state
running

data remaining
0

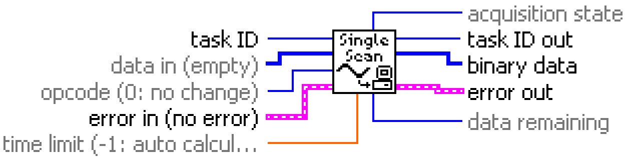
error out
code
no error 0
source

Block Diagram

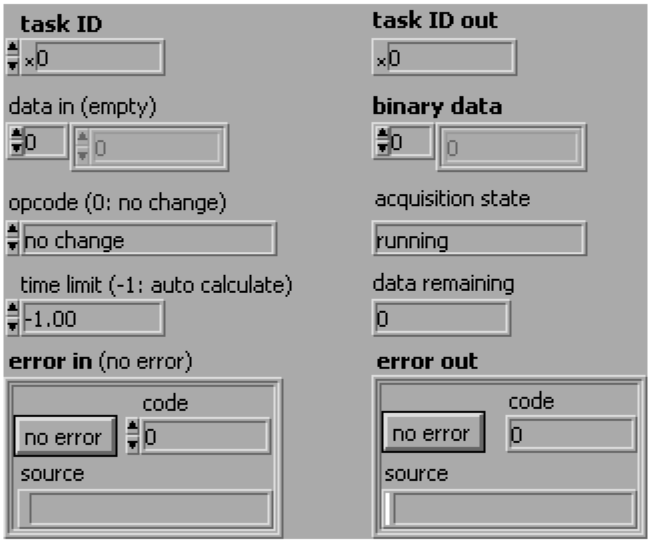


AI SingleScan (binary array).vi

Connector Pane



Front Panel



task ID

x0

data in (empty)

0 | 0

opcode (0: no change)

no change

time limit (-1: auto calculate)

-1.00

error in (no error)

code

no error | 0

source

task ID out

x0

binary data

0 | 0

acquisition state

running

data remaining

0

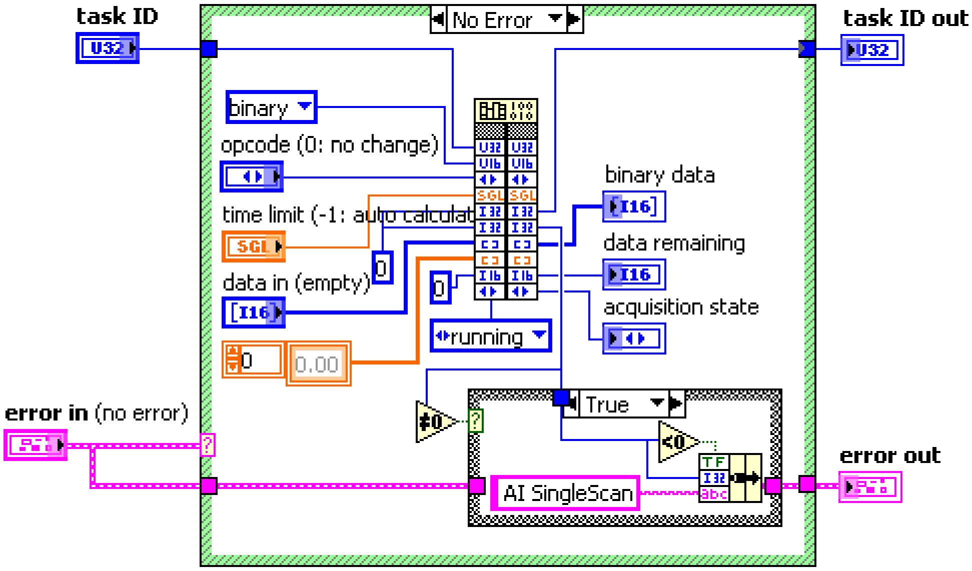
error out

code

no error | 0

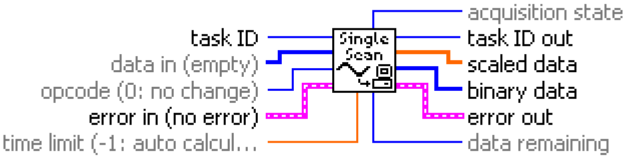
source

Block Diagram

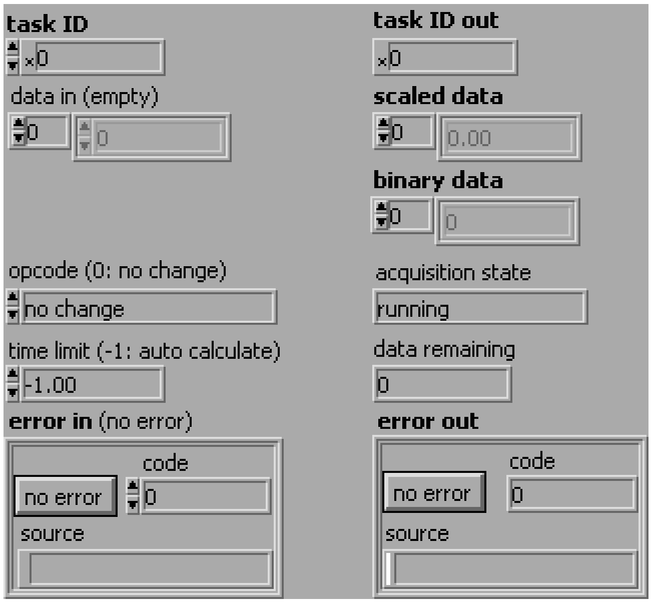


AI SingleScan (scaled & binary arrays).vi

Connector Pane



Front Panel



task ID
0

data in (empty)
0 | 0

opcode (0: no change)
no change

time limit (-1: auto calculate)
-1.00

error in (no error)
no error | code 0
source

task ID out
0

scaled data
0 | 0.00

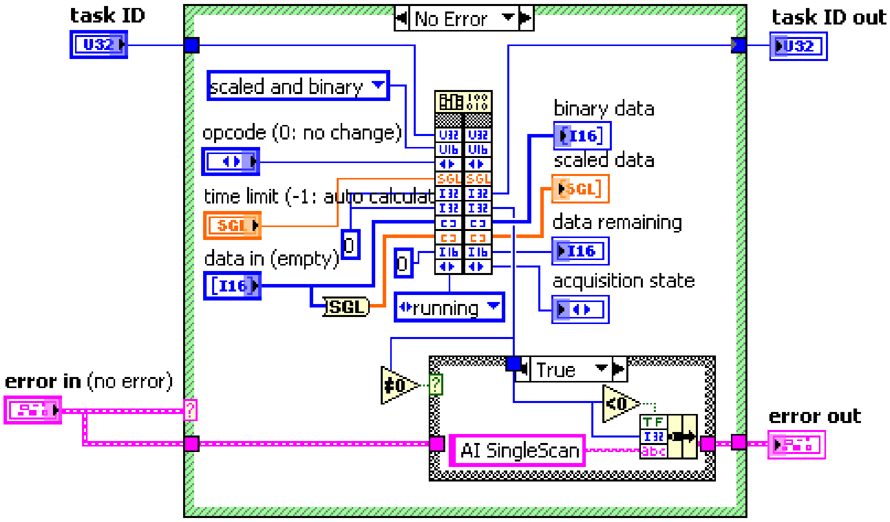
binary data
0 | 0

acquisition state
running

data remaining
0

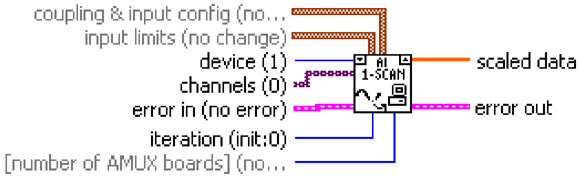
error out
no error | code 0
source

Block Diagram

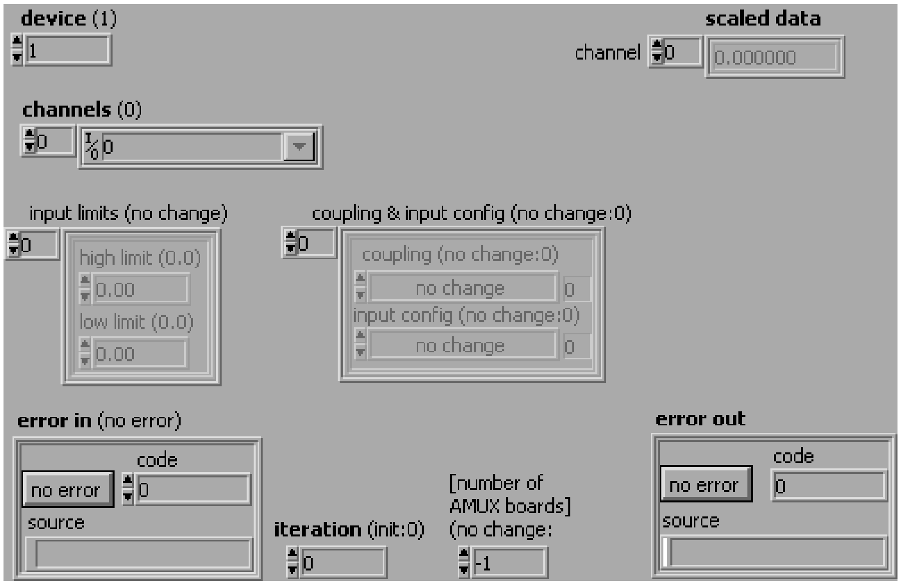


AI Read One Scan (scaled array).vi

Connector Pane



Front Panel


device (1)
1

scaled data
channel 0
0.000000

channels (0)
0
0

input limits (no change)
0
high limit (0.0)
0.00
low limit (0.0)
0.00

coupling & input config (no change:0)
0
coupling (no change:0)
no change 0
input config (no change:0)
no change 0

error in (no error)
code
no error 0
source

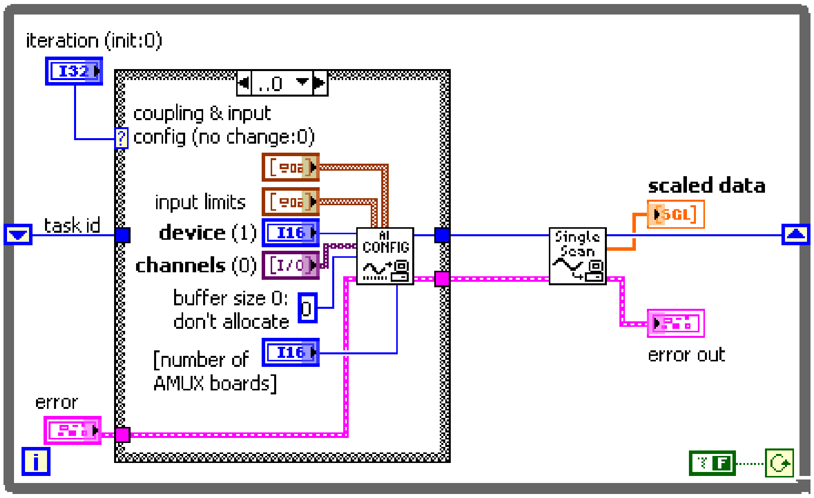
iteration (init:0)
0

[number of AMUX boards] (no change:
-1

error out
code
no error 0
source

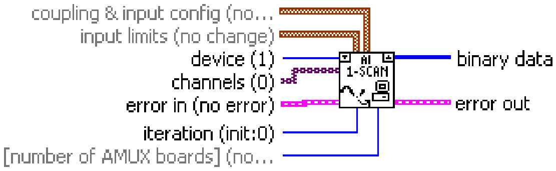
Block Diagram

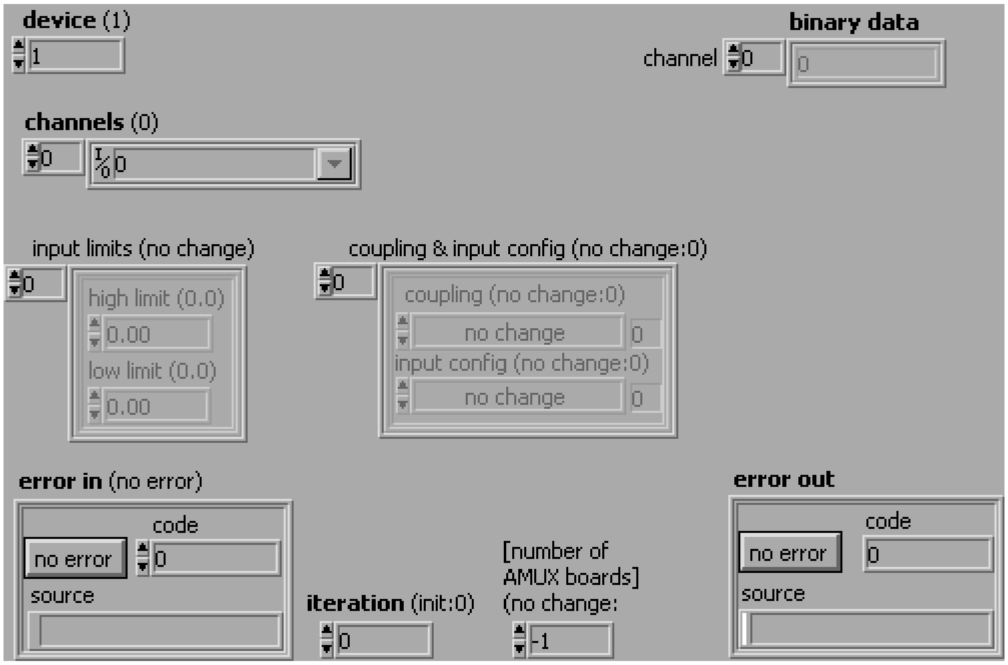
This While Loop executes only once, but uses its shift register to remember **task id** from a previous call to this VI.



AI Read One Scan (binary array).vi

Connector Pane





device (1)
1

binary data
channel 0
0

channels (0)
0
0

input limits (no change)
0
high limit (0.0)
0.00
low limit (0.0)
0.00

coupling & input config (no change:0)
0
coupling (no change:0)
no change 0
input config (no change:0)
no change 0

error in (no error)
no error
code
0
source

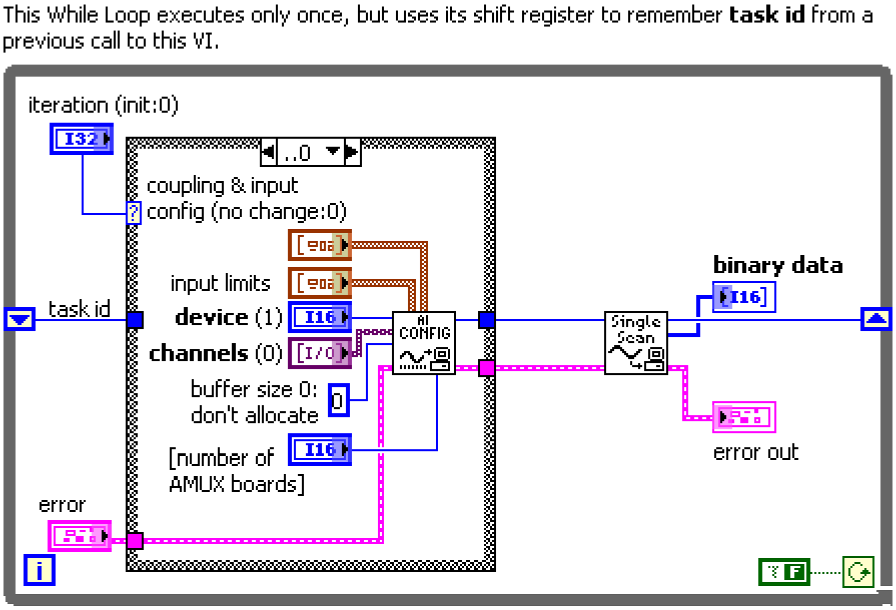
iteration (init:0)
0

[number of AMUX boards] (no change:
-1

error out
no error
code
0
source

Front Panel

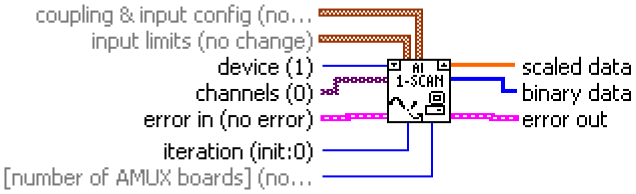
Block Diagram

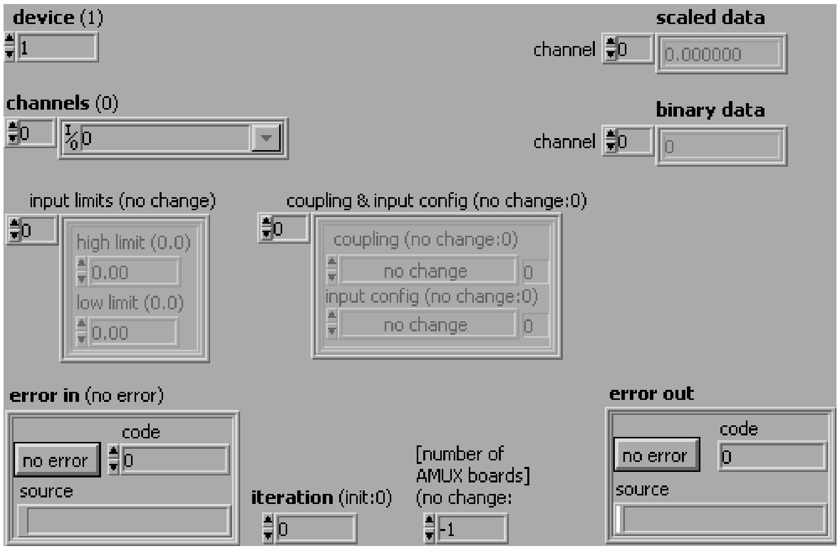


This While Loop executes only once, but uses its shift register to remember **task id** from a previous call to this VI.

AI Read One Scan (scaled & binary arrays).vi

Connector Pane





device (1)
1

channels (0)
0 | 0

input limits (no change)
0
high limit (0.0)
0.00
low limit (0.0)
0.00

coupling & input config (no change:0)
0
coupling (no change:0)
no change 0
input config (no change:0)
no change 0

error in (no error)
no error | code 0
source

iteration (init:0)
0

[number of AMUX boards] (no change:
-1

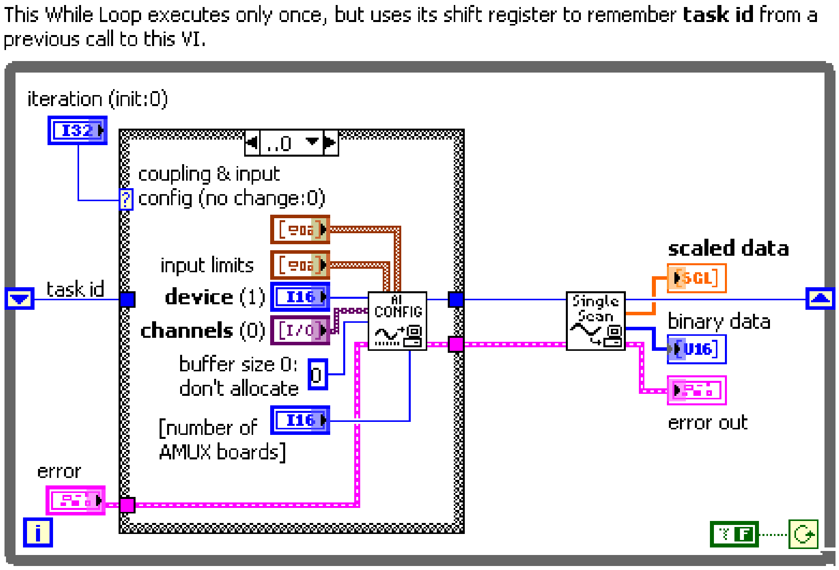
scaled data
channel 0 | 0.000000

binary data
channel 0 | 0

error out
no error | code 0
source

Front Panel

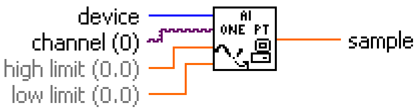
Block Diagram



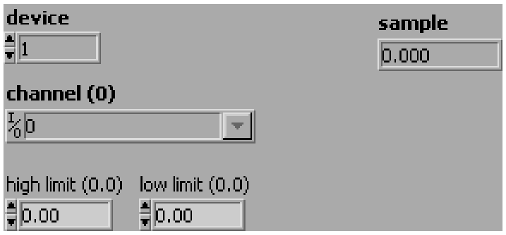
This While Loop executes only once, but uses its shift register to remember **task id** from a previous call to this VI.

AI Sample Channel (scaled value).vi

Connector Pane



Front Panel



device
1

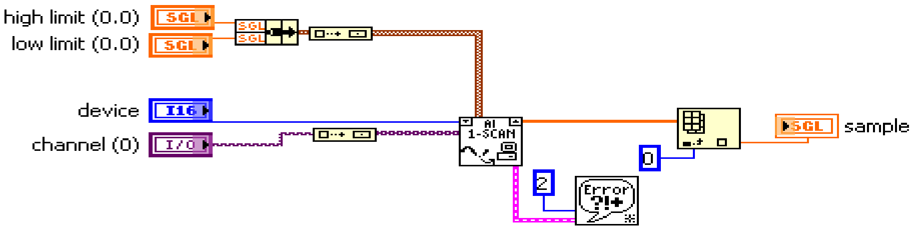
channel (0)
0

high limit (0.0)
0.00

low limit (0.0)
0.00

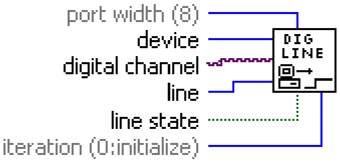
sample
0.000

Block Diagram

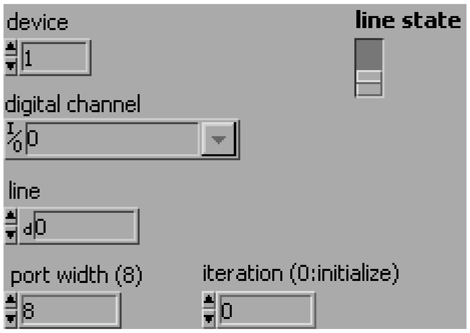


Write to Digital Line.vi

Connector Pane



Front Panel


device: 1

digital channel: 0

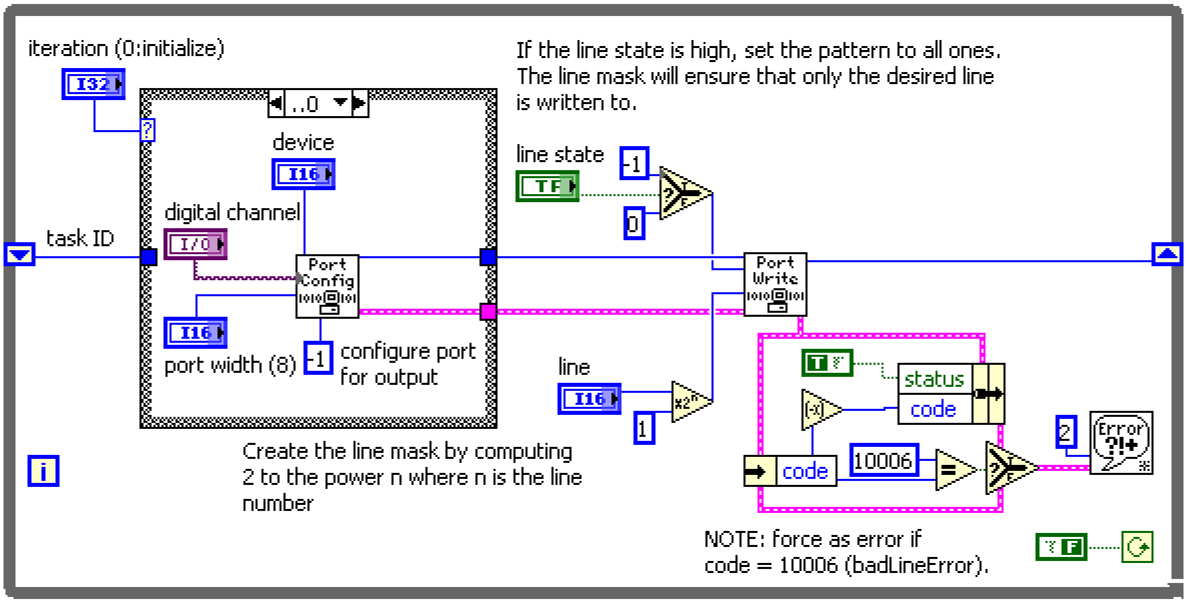
line: 0

port width (8): 8

iteration (0:initialize): 0

line state

Block Diagram

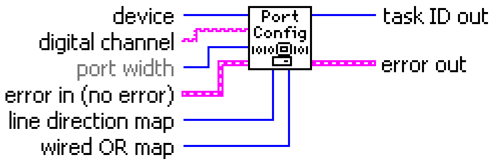

If the line state is high, set the pattern to all ones. The line mask will ensure that only the desired line is written to.

Create the line mask by computing 2 to the power n where n is the line number

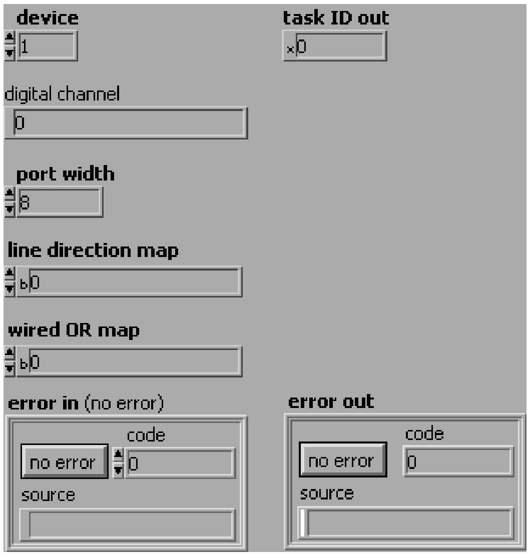
NOTE: force as error if code = 10006 (badLineError).

DIO Port Config.vi

Connector Pane



Front Panel



device: 1

task ID out: 0

digital channel: 0

port width: 8

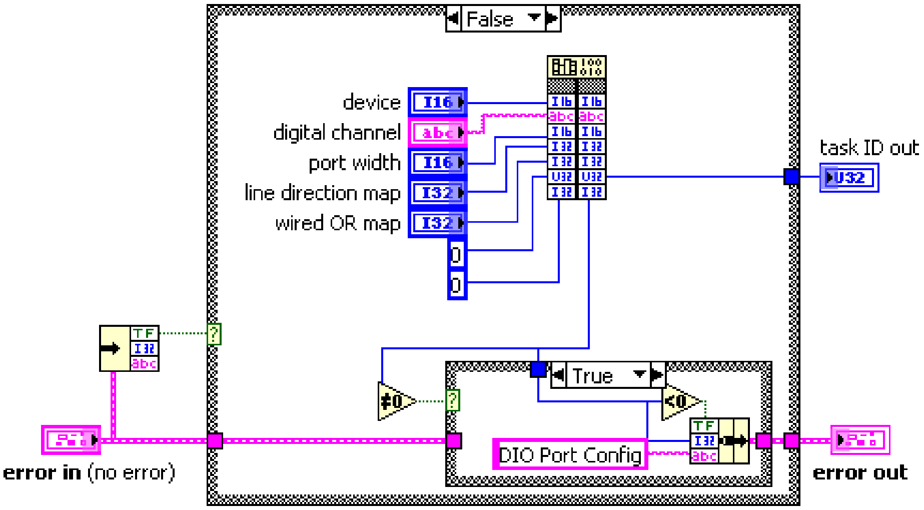
line direction map: 0

wired OR map: 0

error in (no error): no error, code 0, source

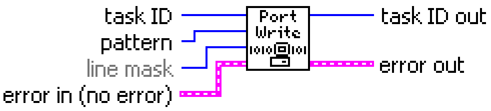
error out: no error, code 0, source

Block Diagram

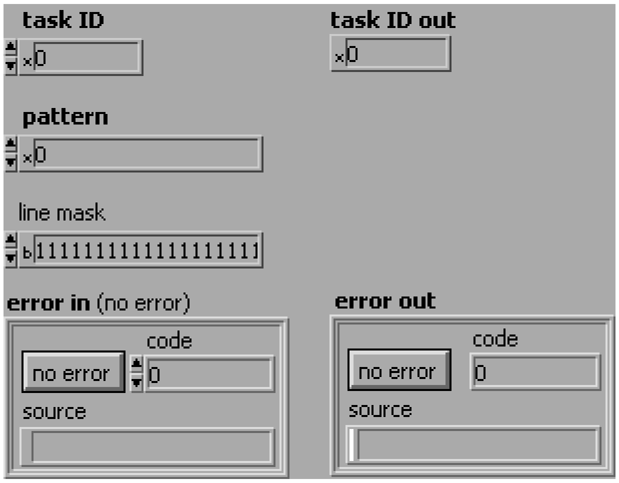


DIO Port Write.vi

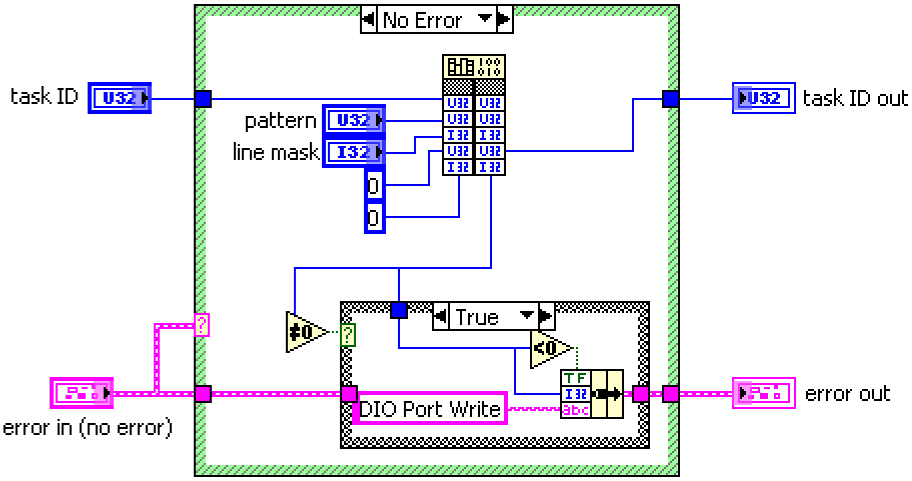
Connector Pane



Front Panel

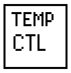


Block Diagram

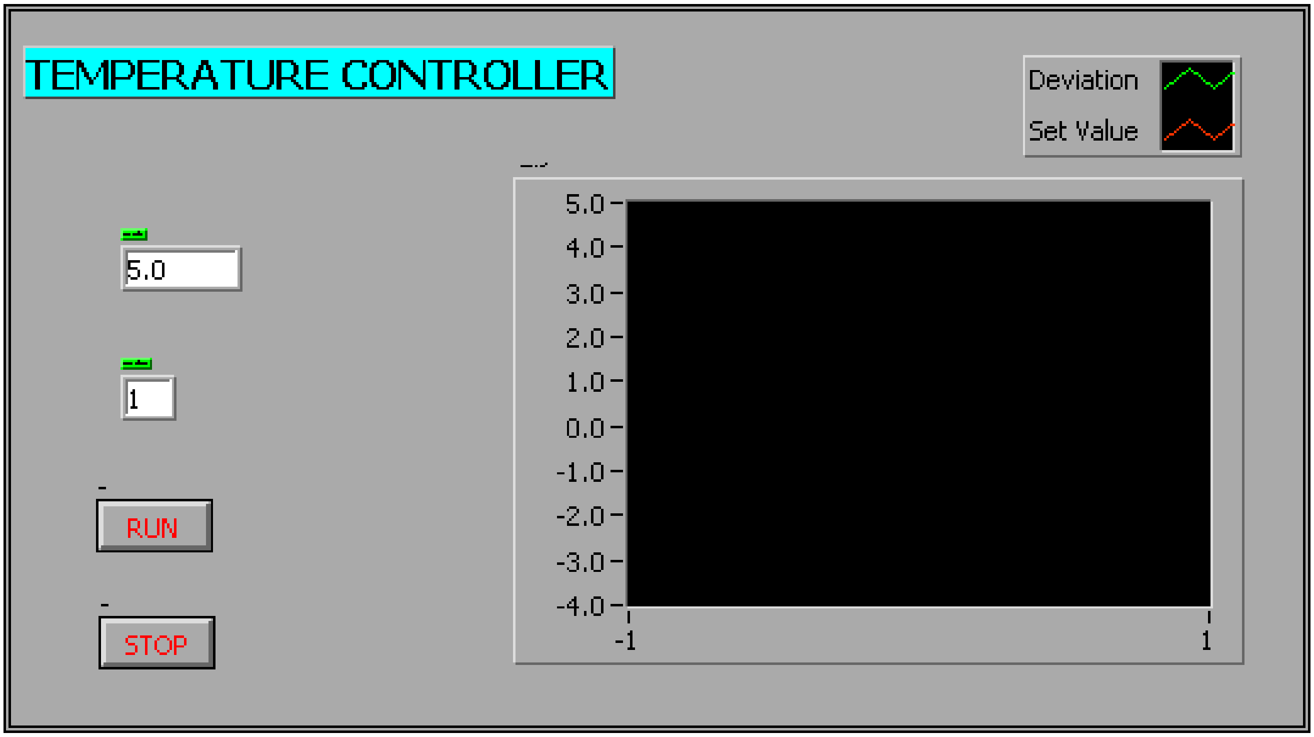


TempController.vi

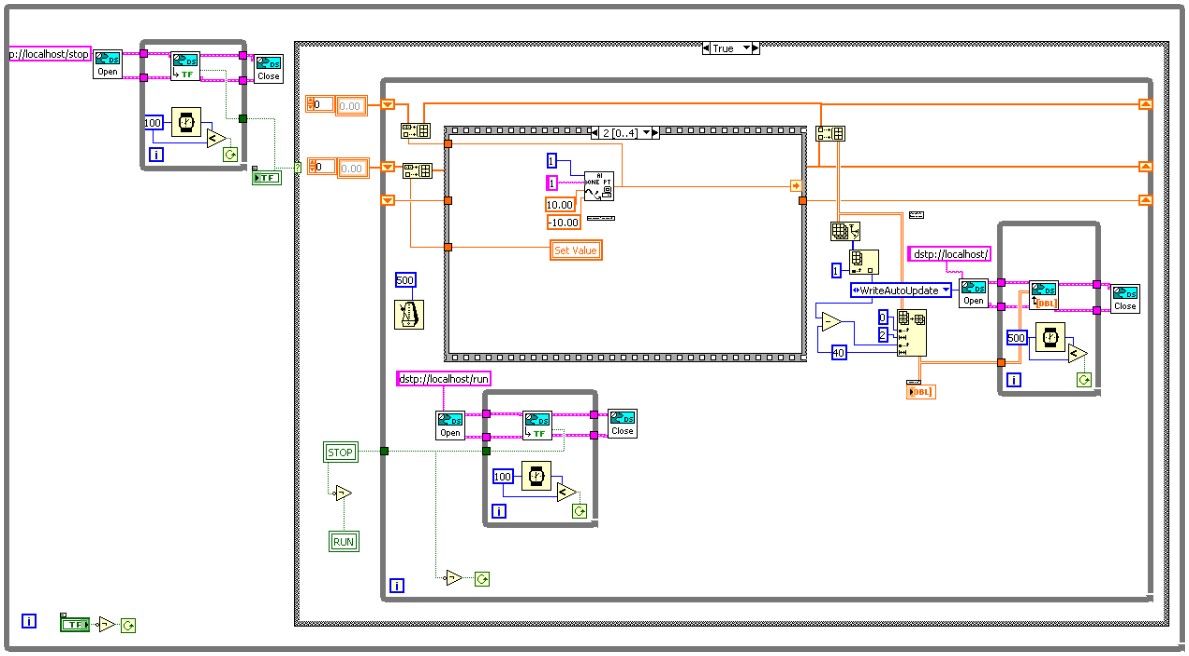
Connector Pane



Front Panel

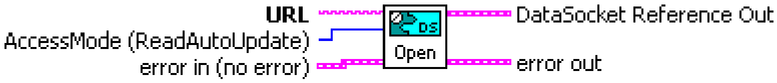


Block Diagram

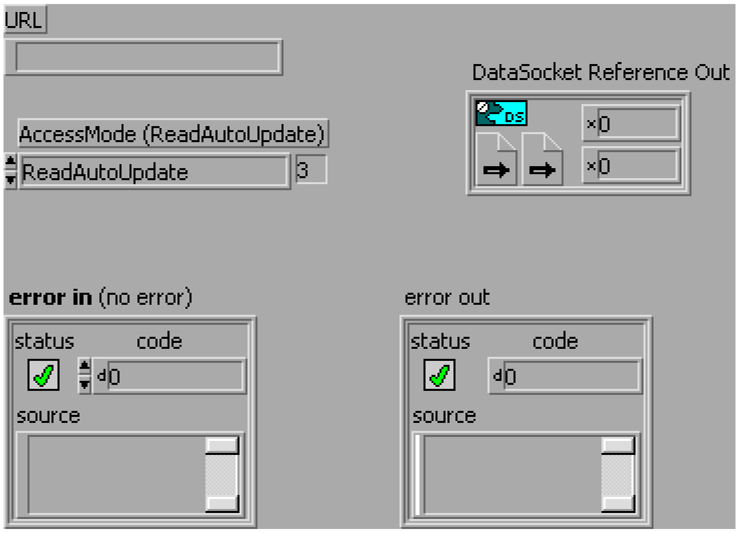


DataSocket Open Connection.vi

Connector Pane



Front Panel



URL

AccessMode (ReadAutoUpdate)

ReadAutoUpdate 3

DataSocket Reference Out

x0

x0

error in (no error)

status code

✓ d0

source

error out

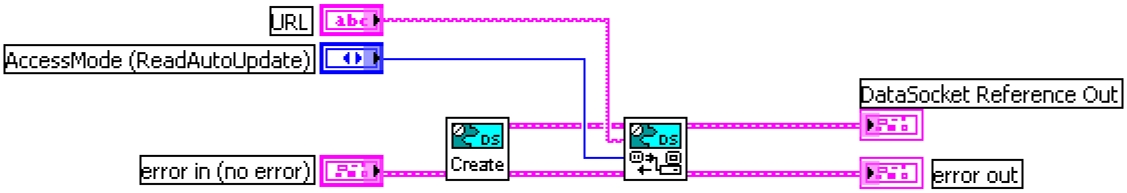
status code

✓ d0

source

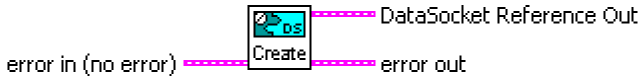
112

Block Diagram

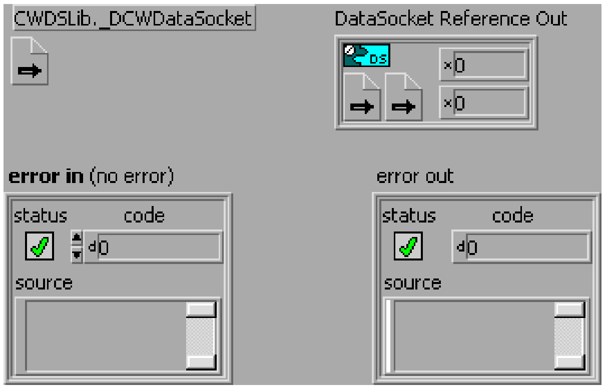


DataSocket Create.vi

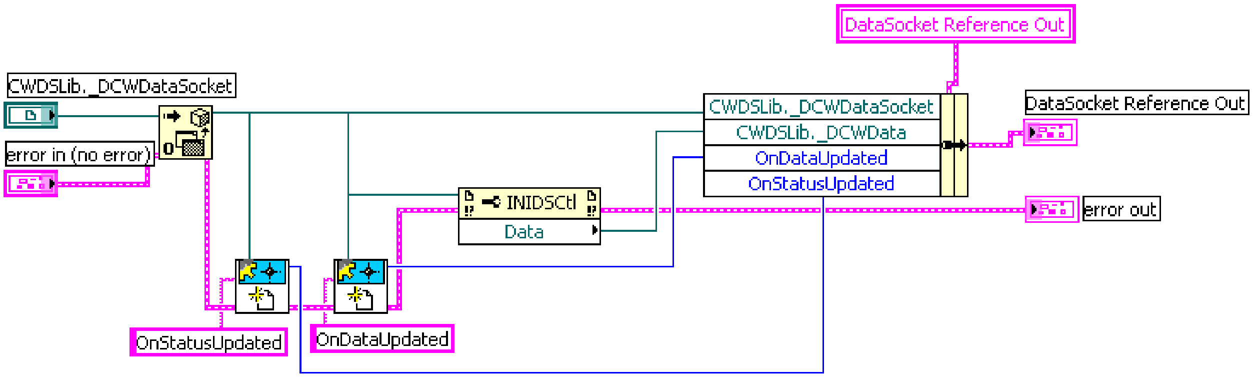
Connector Pane



Front Panel

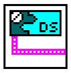


Block Diagram

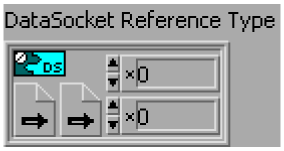


DataSocket Reference Type.ctl

Connector Pane



Front Panel



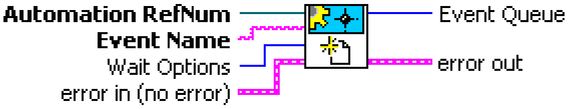
DataSocket Reference Type

0

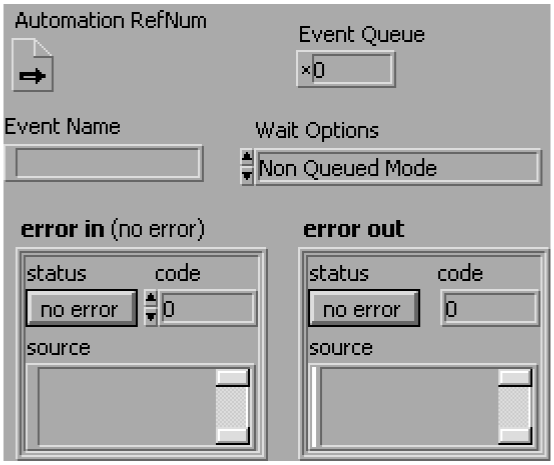
0

Create ActiveX Event Queue.vi

Connector Pane



Front Panel



Automation RefNum

Event Queue
0

Event Name

Wait Options
Non Queued Mode

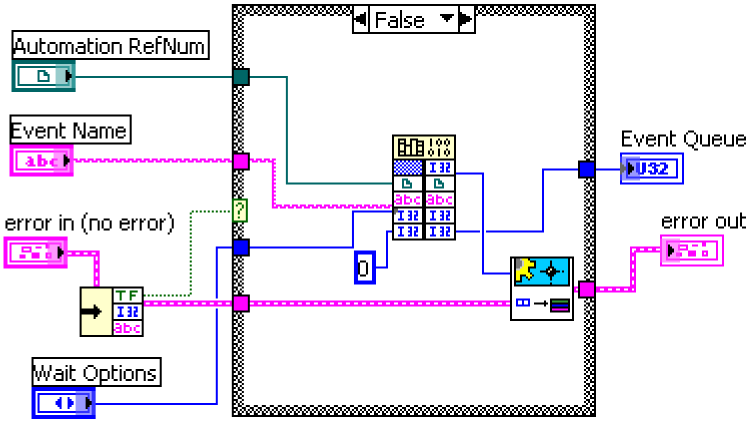
error in (no error)

status: no error
code: 0
source

error out

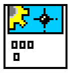
status: no error
code: 0
source

Block Diagram



Wait types.ctl

Connector Pane



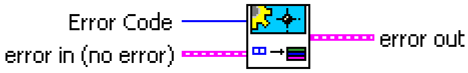
Front Panel

Wait Options

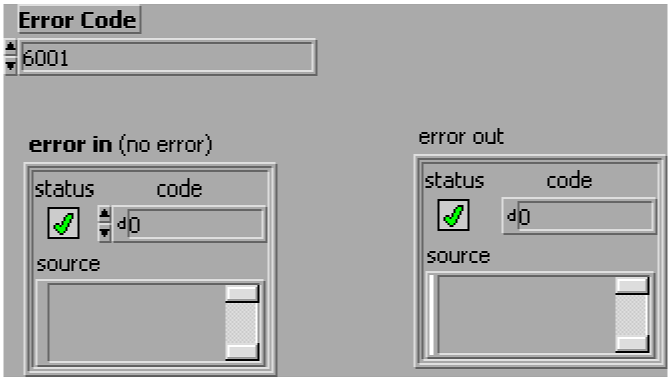
Queued Mode

Create Error Clust.vi

Connector Pane



Front Panel



Error Code

6001

error in (no error)

status ✓ code 0

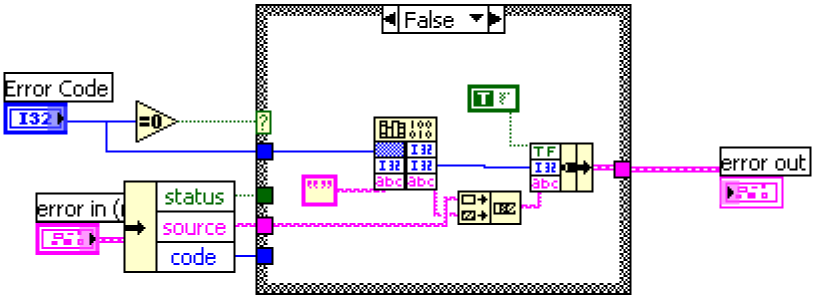
source

error out

status ✓ code 0

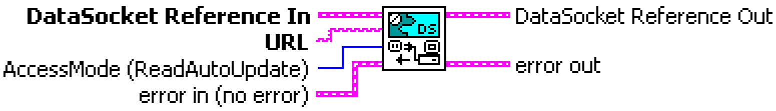
source

Block Diagram

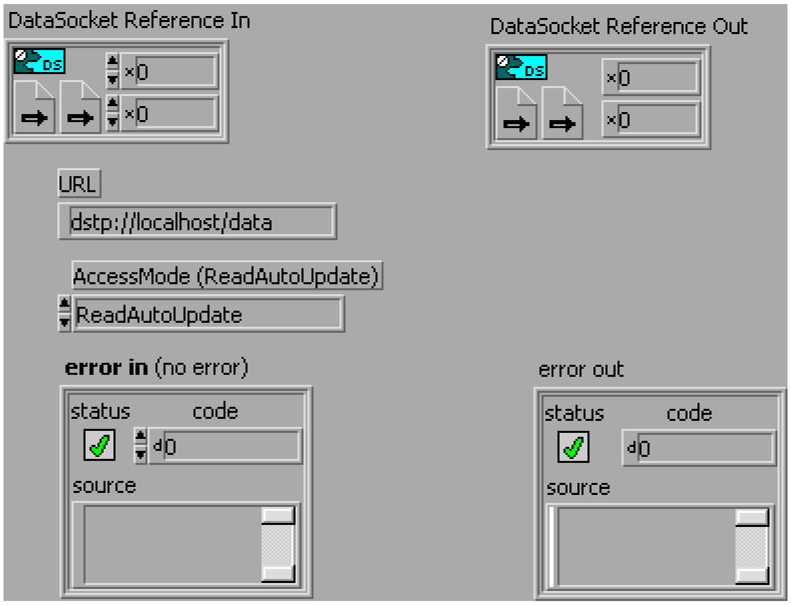


DataSocket Connect.vi

Connector Pane



Front Panel



DataSocket Reference In

0

0

DataSocket Reference Out

0

0

URL

dstp://localhost/data

AccessMode (ReadAutoUpdate)

ReadAutoUpdate

error in (no error)

status ✓ code 0

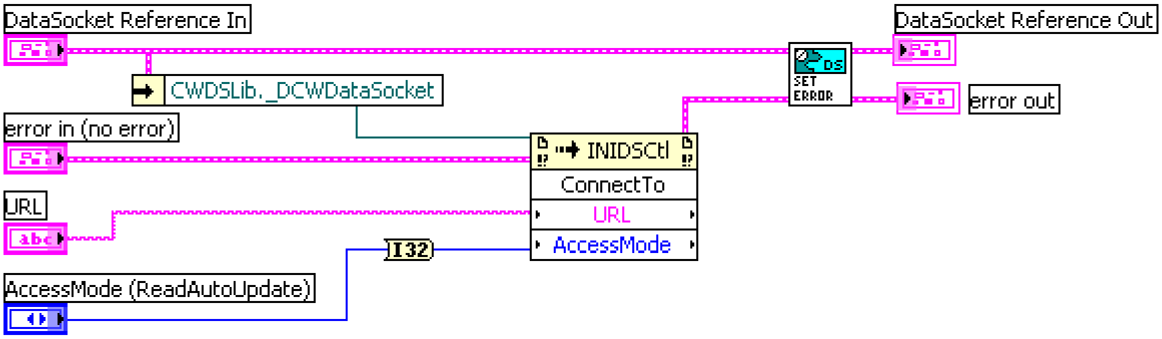
source

error out

status ✓ code 0

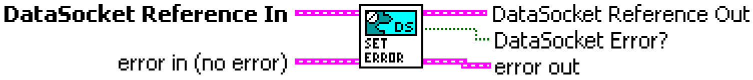
source

Block Diagram

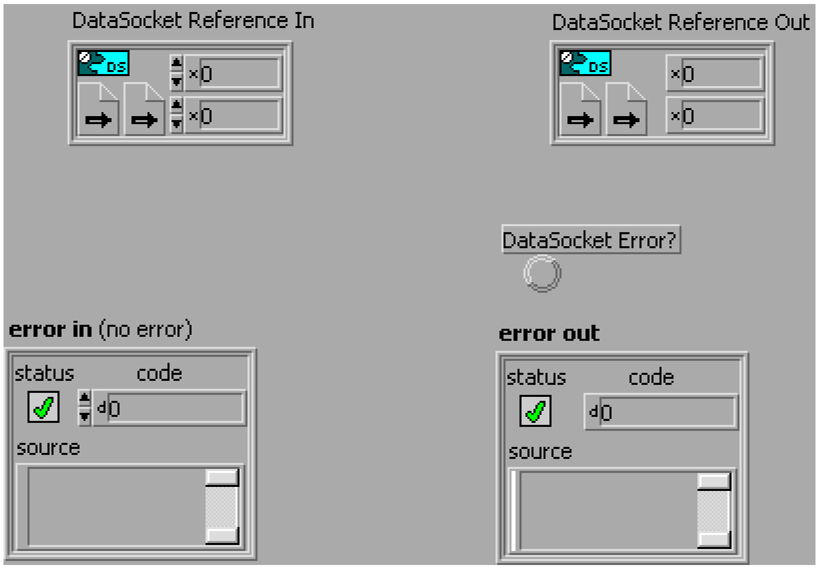


DataSocket Set Error From Status.vi

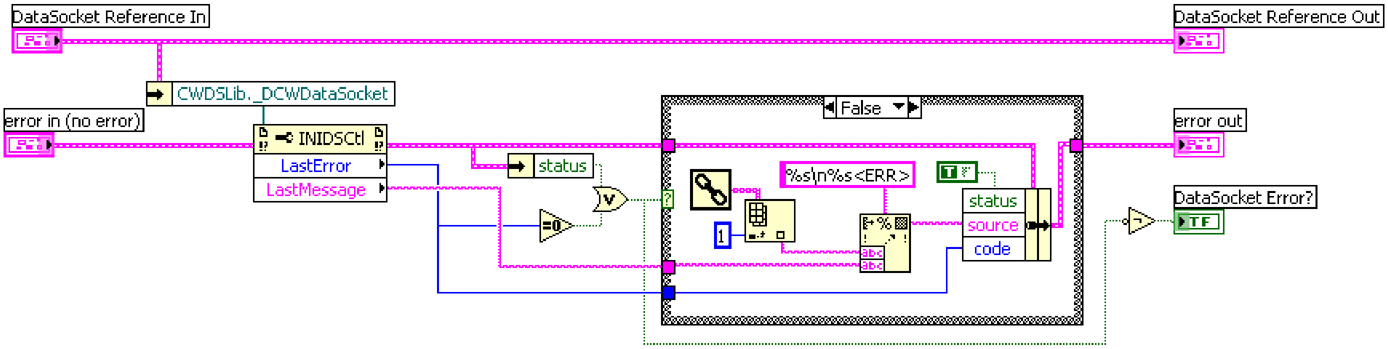
Connector Pane



Front Panel

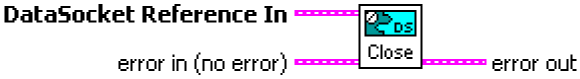


Block Diagram

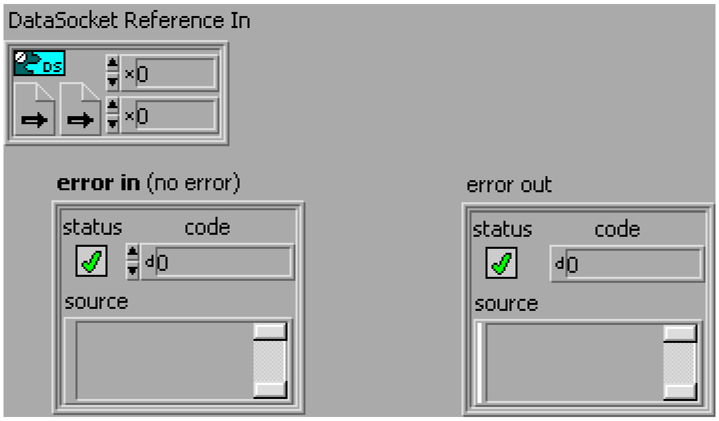


DataSocket Close Connection.vi

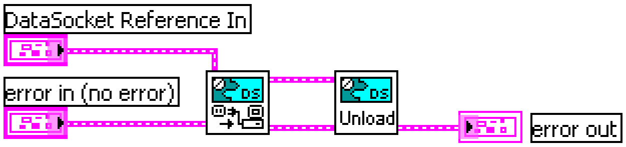
Connector Pane



Front Panel

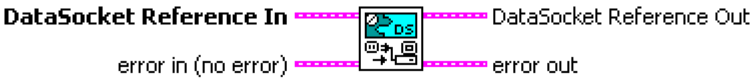


Block Diagram

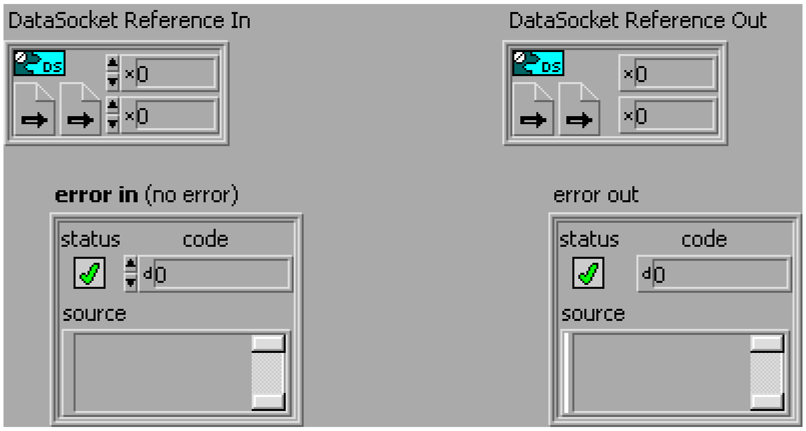


DataSocket Disconnect.vi

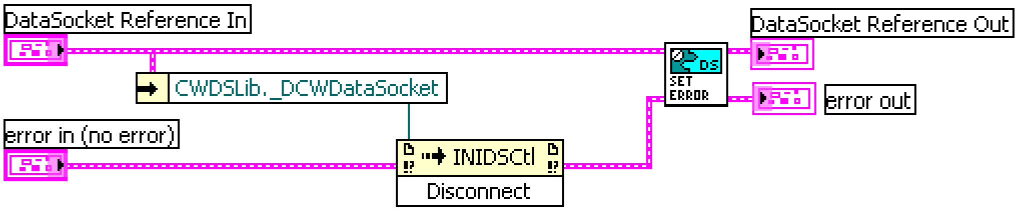
Connector Pane



Front Panel

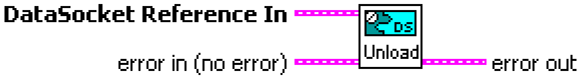


Block Diagram

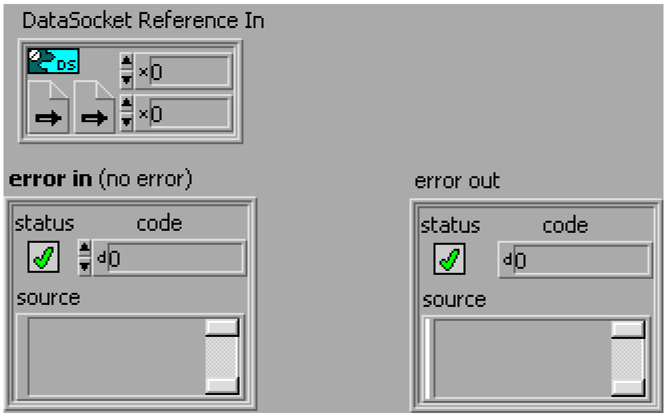


DataSocket Unload.vi

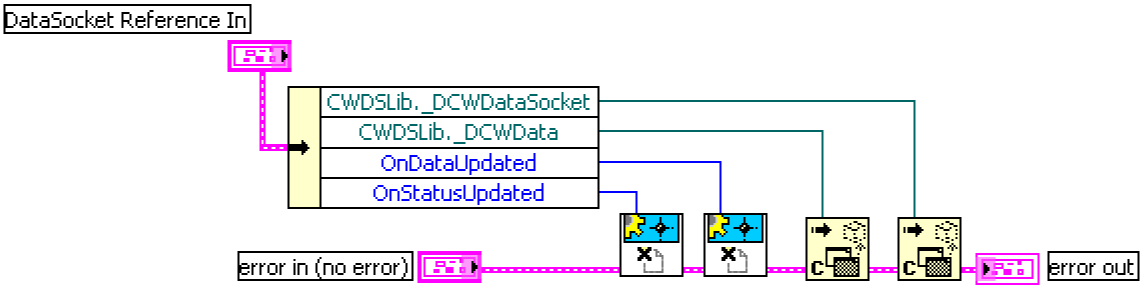
Connector Pane



Front Panel

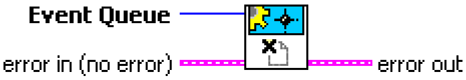


Block Diagram

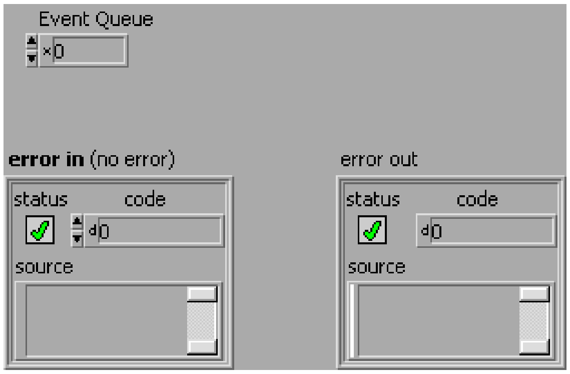


Destroy ActiveX Event Queue.vi

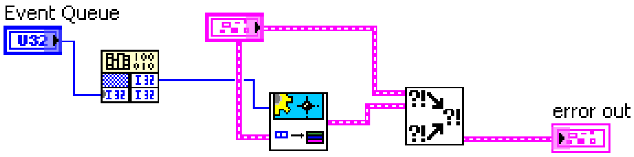
Connector Pane



Front Panel

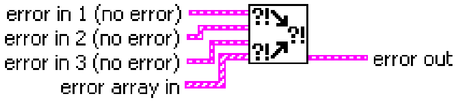


Block Diagram

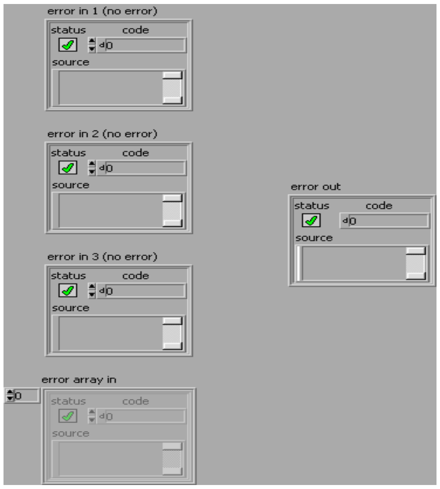


Merge Errors.vi

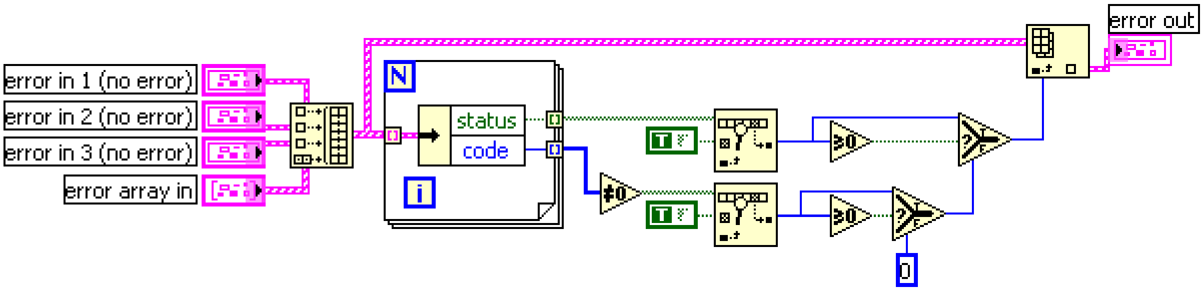
Connector Pane



Front Panel

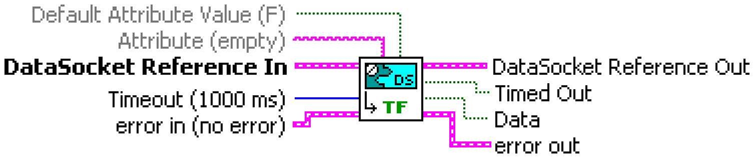


Block Diagram

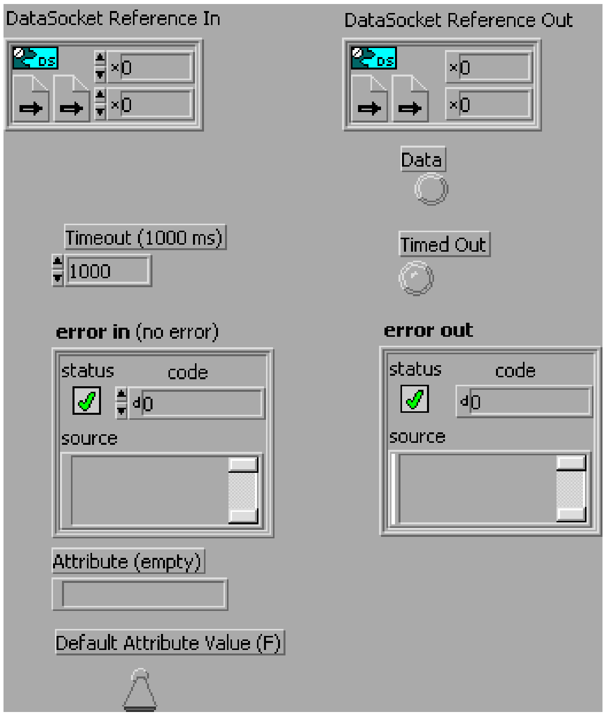


DataSocket Read Boolean.vi

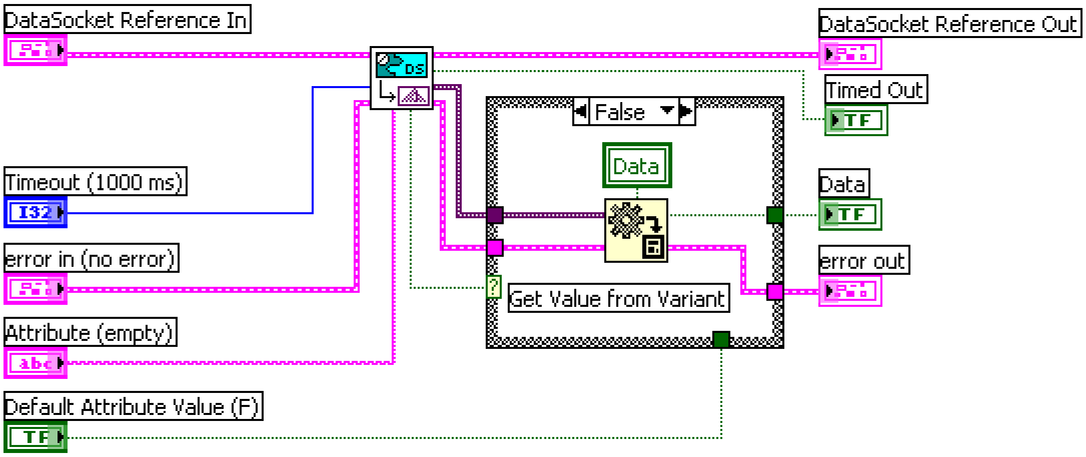
Connector Pane



Front Panel

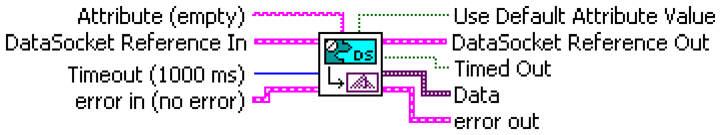


Block Diagram

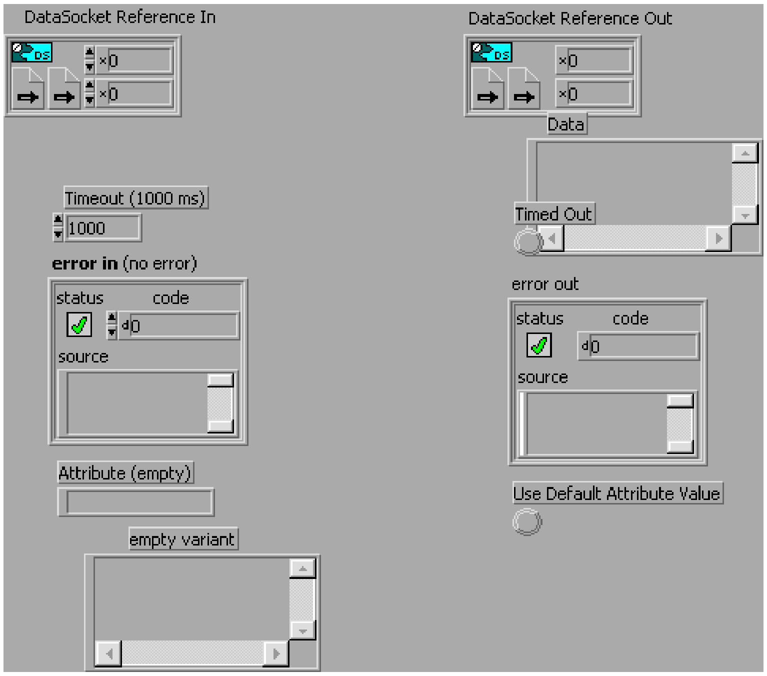


DataSocket Read Variant.vi

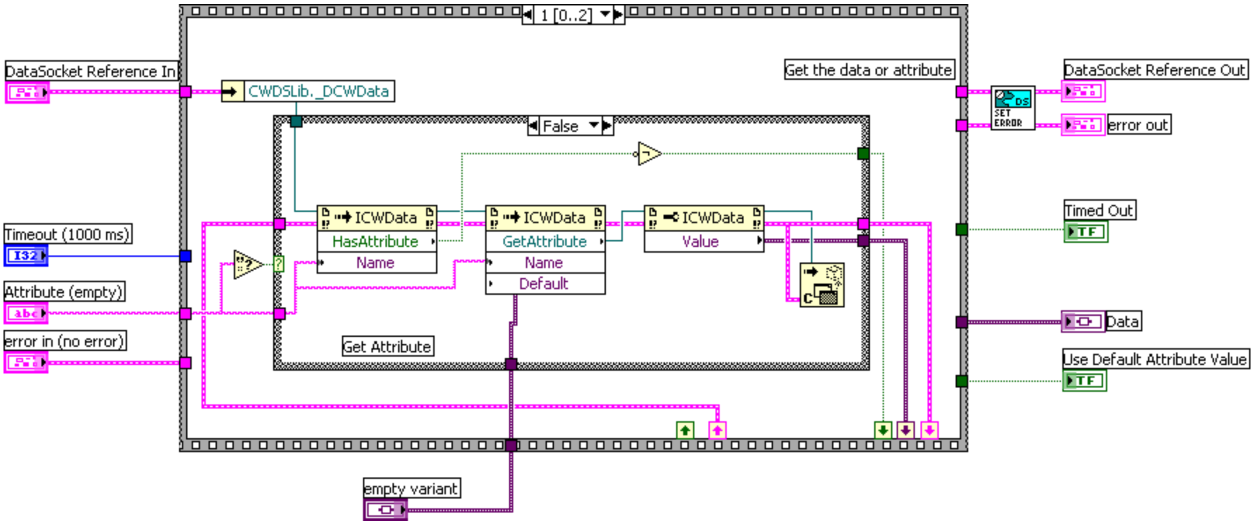
Connector Pane



Front Panel

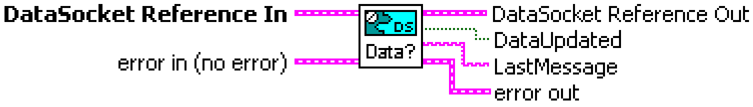


Block Diagram

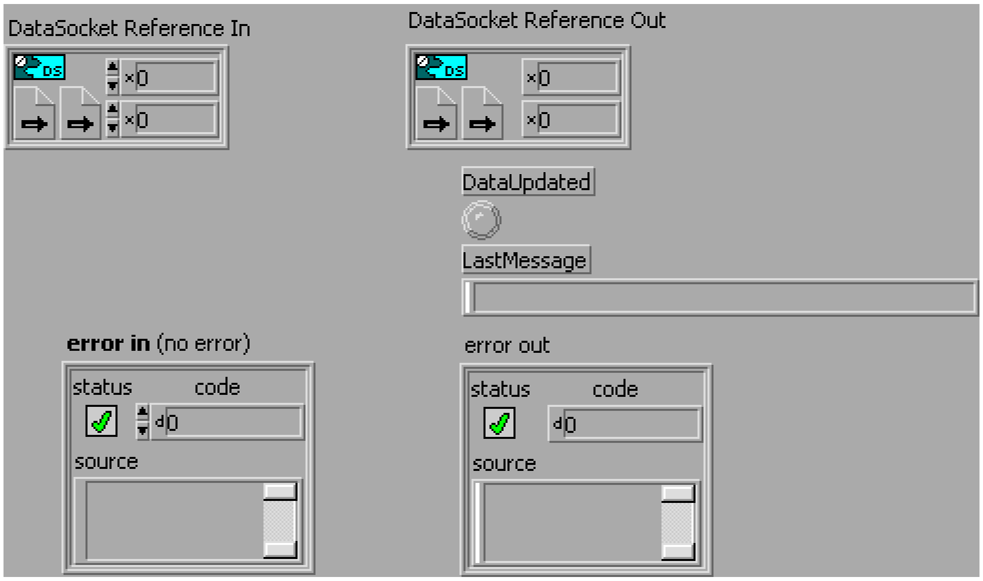


DataSocket Data Updated.vi

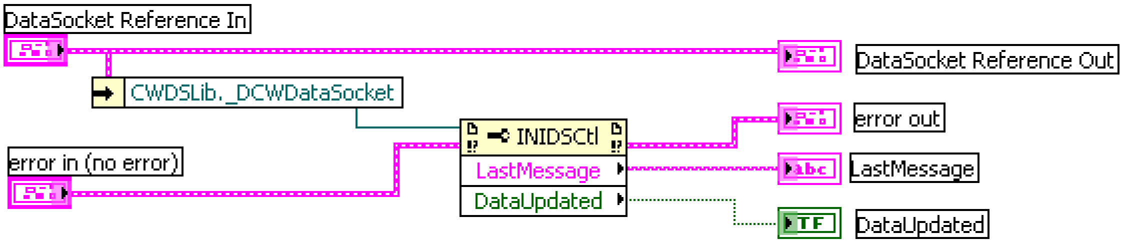
Connector Pane



Front Panel

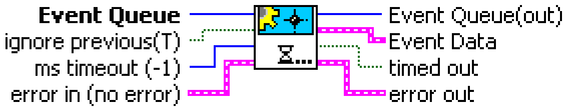


Block Diagram

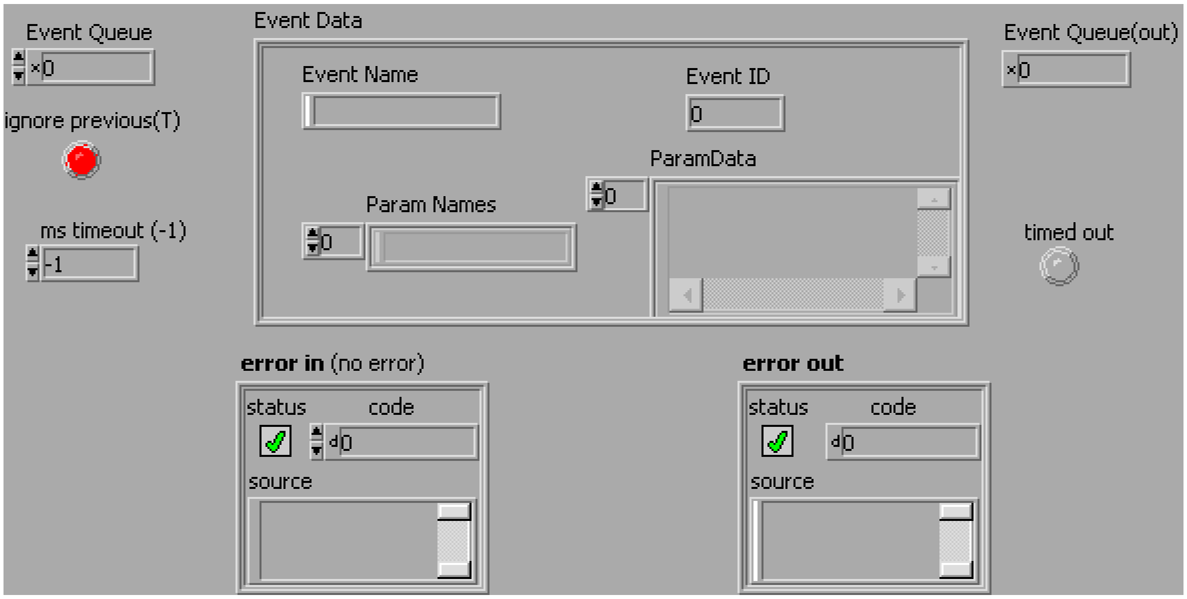


Wait On ActiveX Event.vi

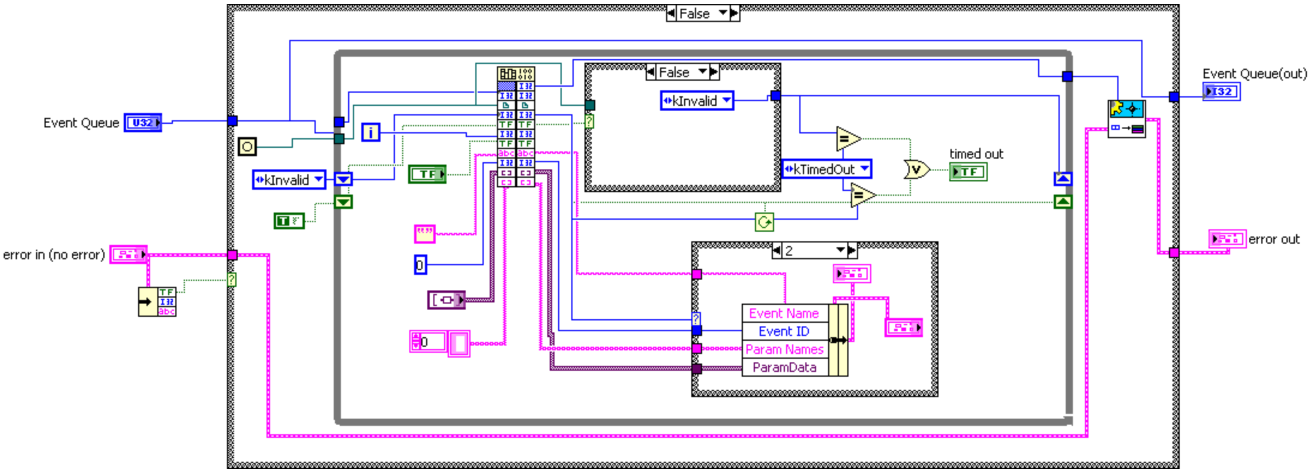
Connector Pane



Front Panel

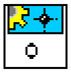


Block Diagram

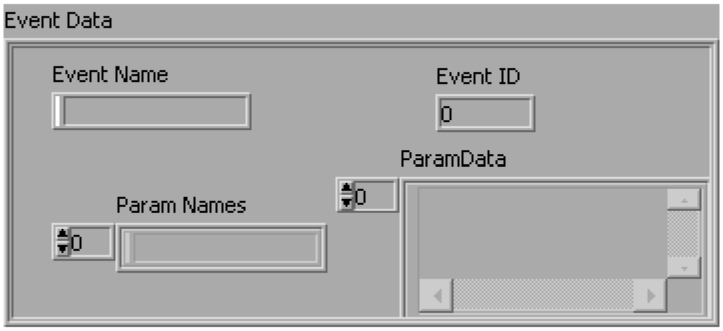


EventData.ctl

Connector Pane



Front Panel


Event Data

Event Name

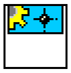
Event ID 0

ParamData 0

Param Names 0

OccurFireType.clt

Connector Pane

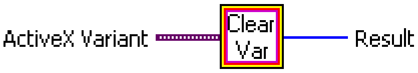


Front Panel

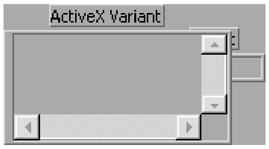
OccurTypes kInvalid

DataSocket Clear Variant.vi

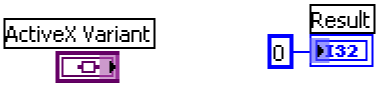
Connector Pane



Front Panel

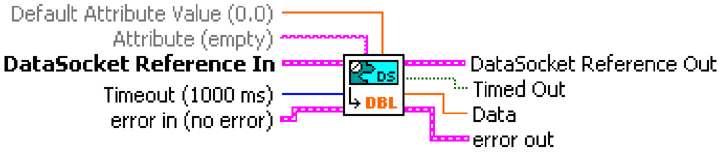


Block Diagram

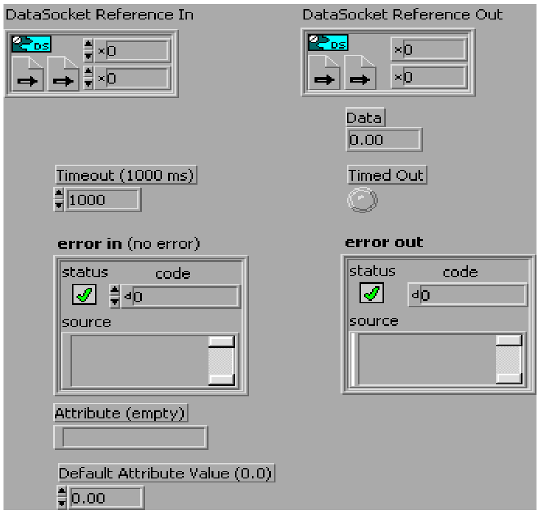


DataSocket Read Double.vi

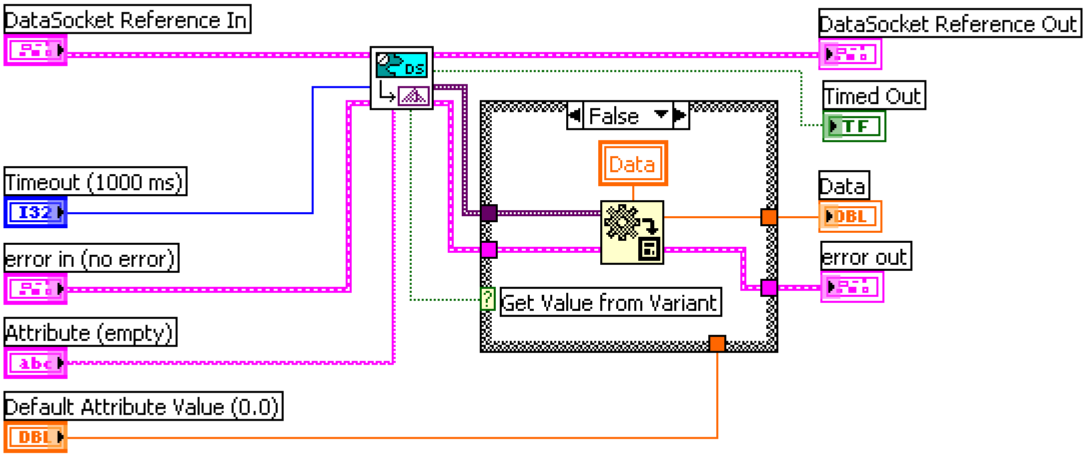
Connector Pane



Front Panel

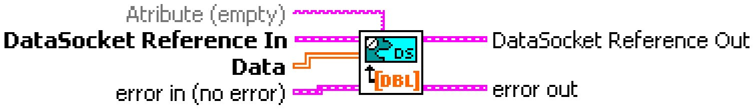


Block Diagram

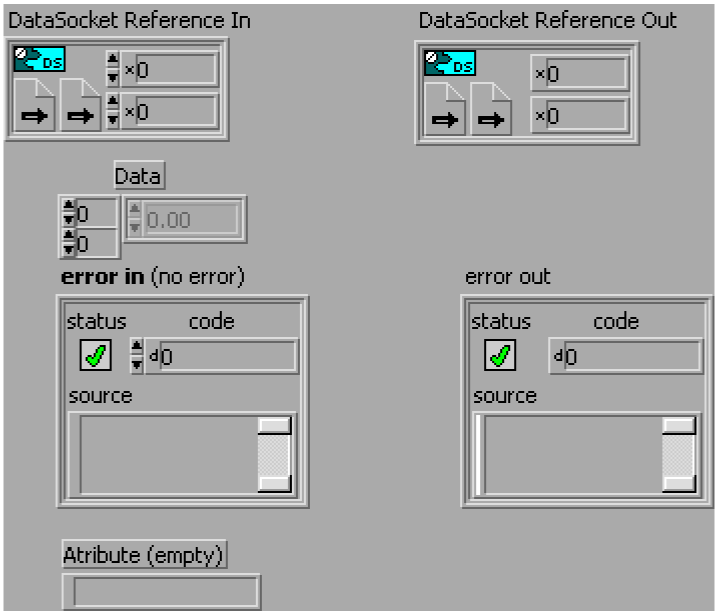


DataSocket Write Double Matrix.vi

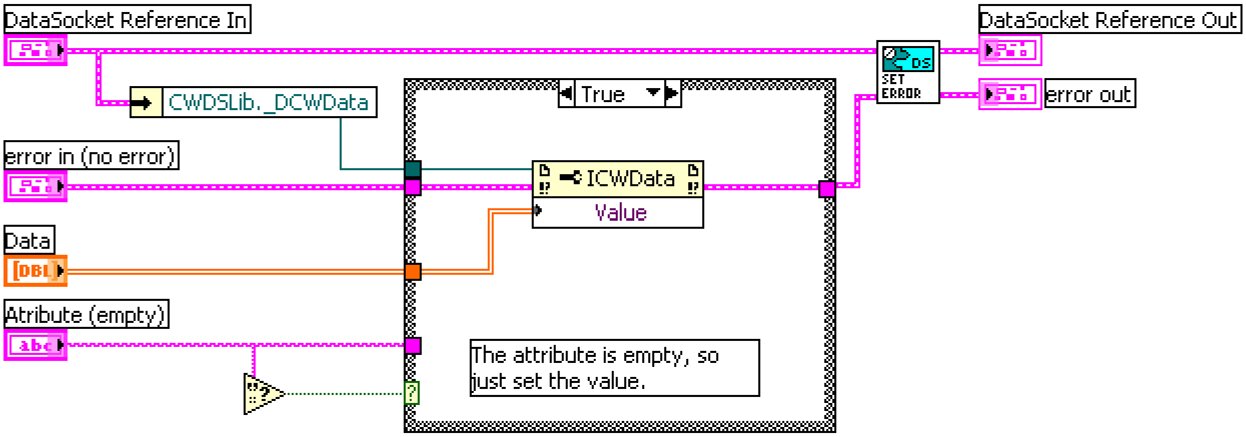
Connector Pane



Front Panel



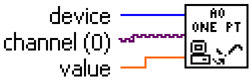
Block Diagram



The attribute is empty, so just set the value.

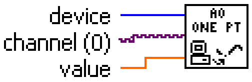
AO Update Channel.vi

Connector Pane

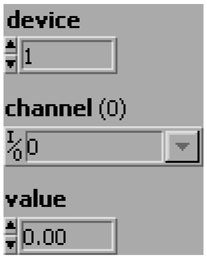


AO Update Channel (scaled value).vi

Connector Pane



Front Panel



device

1

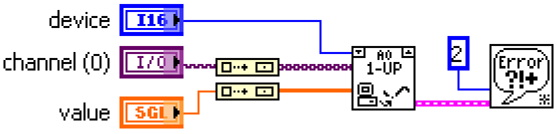
channel (0)

0

value

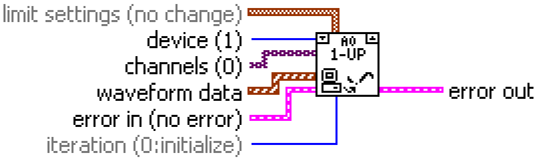
0.00

Block Diagram



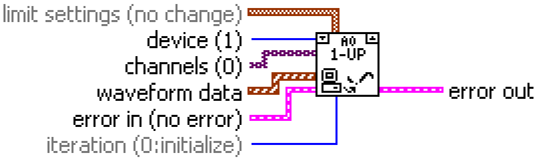
AO Write One Update.vi

Connector Pane

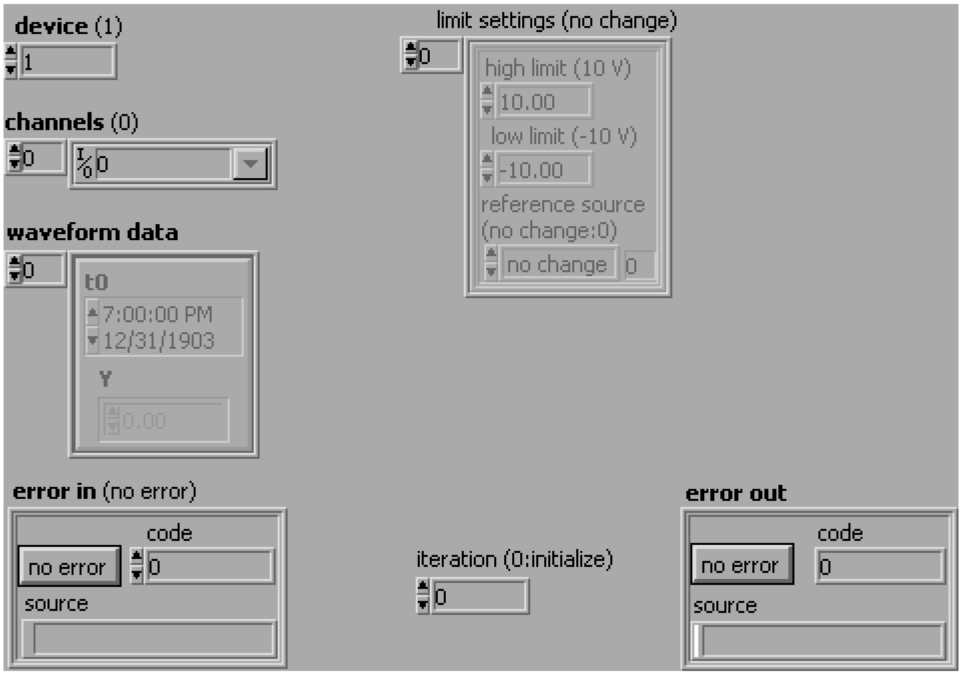


AO Write One Update (single-point waveform).vi

Connector Pane



Front Panel



device (1)
1

channels (0)
0 0

waveform data
0
t0
7:00:00 PM
12/31/1903
Y
0.00

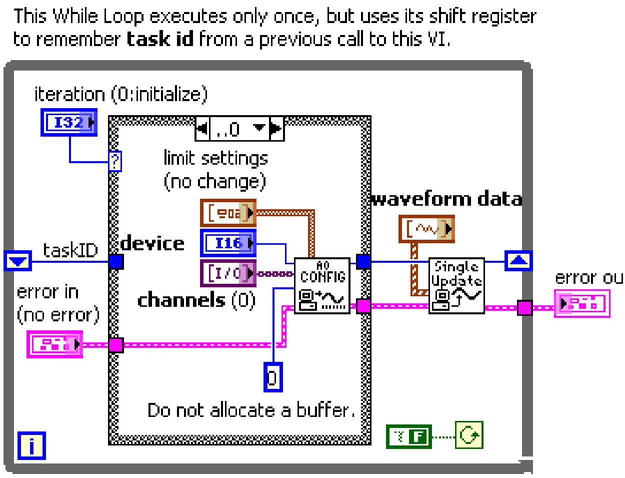
limit settings (no change)
0
high limit (10 V)
10.00
low limit (-10 V)
-10.00
reference source
(no change:0)
no change 0

error in (no error)
no error
code
0
source

iteration (0:initialize)
0

error out
no error
code
0
source

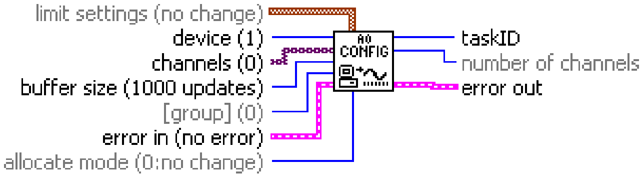
Block Diagram



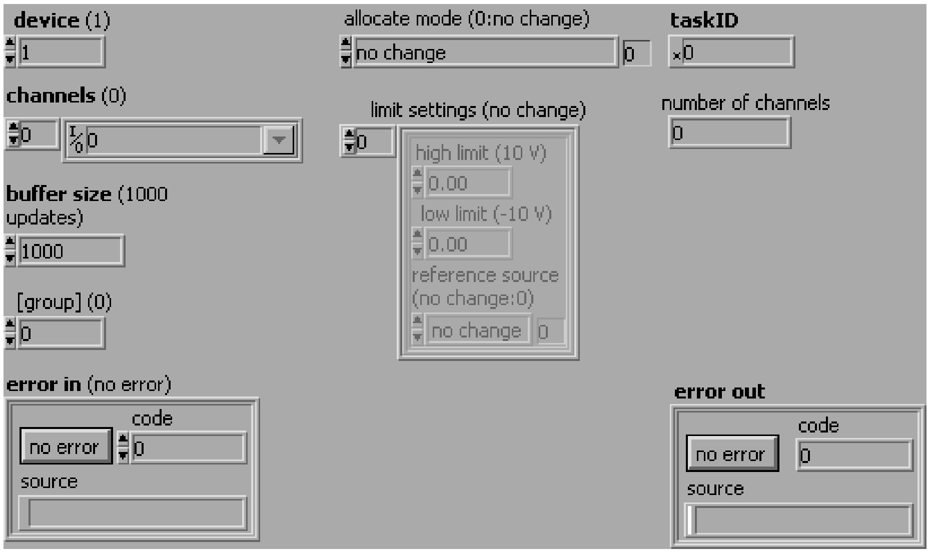
This While Loop executes only once, but uses its shift register to remember **task id** from a previous call to this VI.

AO Config.vi

Connector Pane



Front Panel



device (1)
1

channels (0)
0 0

buffer size (1000 updates)
1000

[group] (0)
0

error in (no error)
code
no error 0
source

allocate mode (0:no change)
no change 0

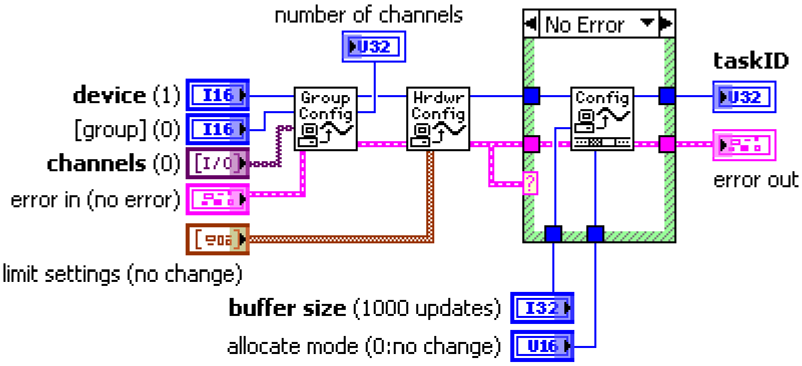
limit settings (no change)
0
high limit (10 V)
0.00
low limit (-10 V)
0.00
reference source (no change:0)
no change 0

taskID
0

number of channels
0

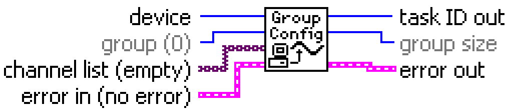
error out
code
no error 0
source

Block Diagram

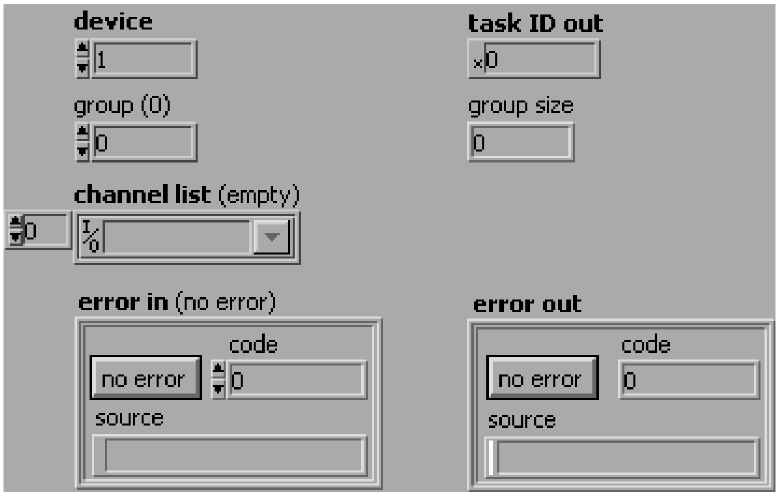


AO Group Config.vi

Connector Pane



Front Panel


device
1

task ID out
0

group (0)
0

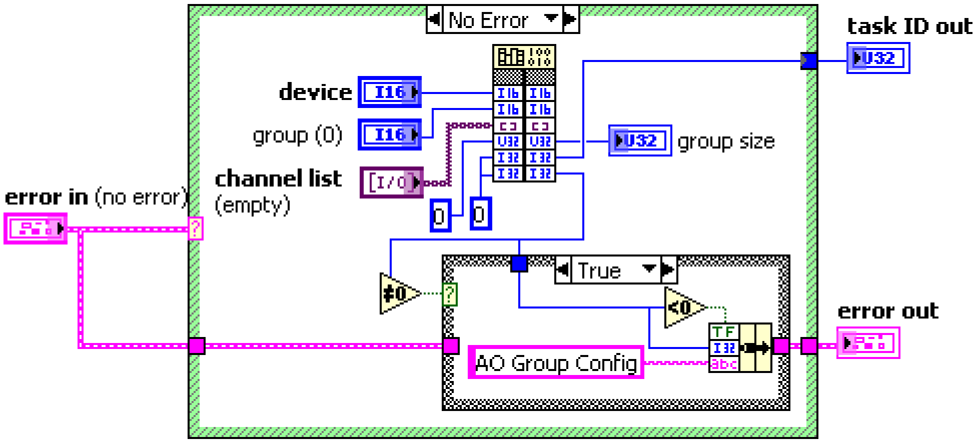
group size
0

channel list (empty)
0

error in (no error)
no error
code
0
source

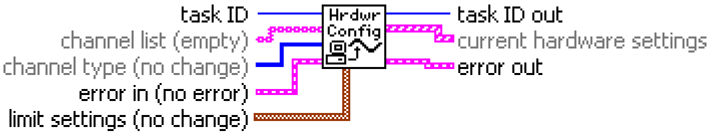
error out
no error
code
0
source

Block Diagram

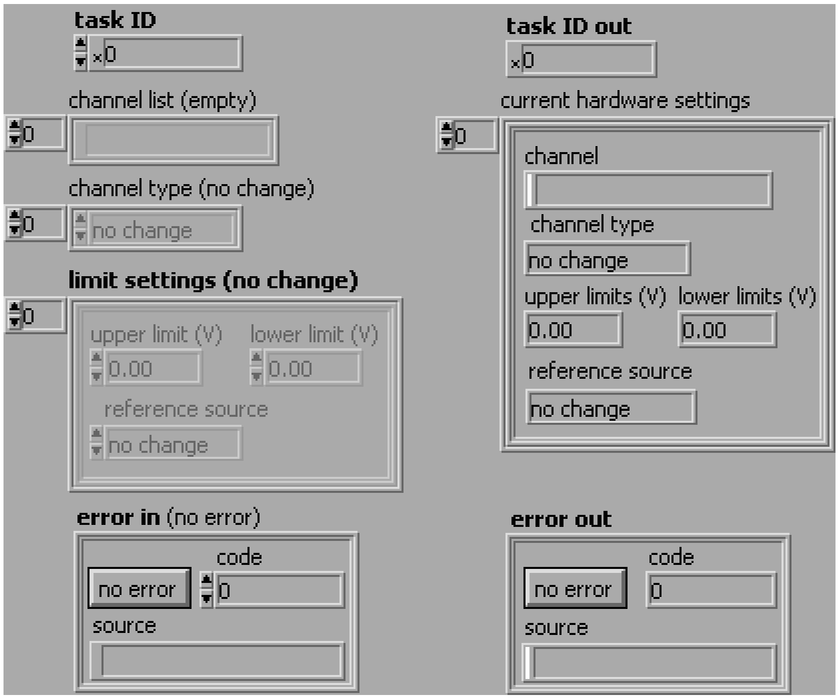


AO Hardware Config.vi

Connector Pane



Front Panel


task ID
0

channel list (empty)
0

channel type (no change)
0
no change

limit settings (no change)
0
upper limit (V) 0.00
lower limit (V) 0.00
reference source
no change

task ID out
0

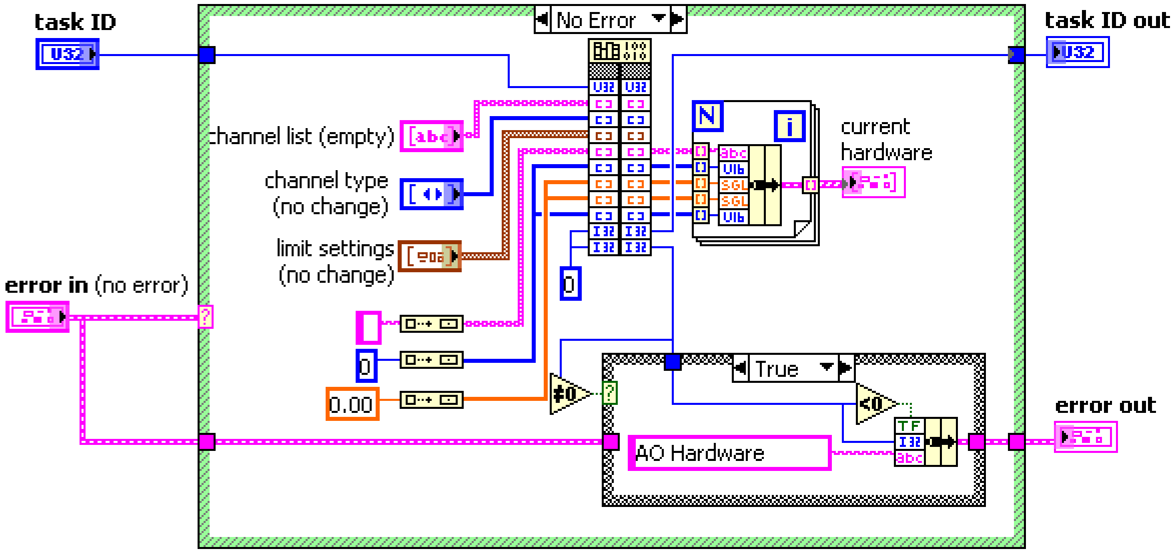
current hardware settings
0
channel

channel type
no change
upper limits (V) 0.00
lower limits (V) 0.00
reference source
no change

error in (no error)
no error
code
0
source

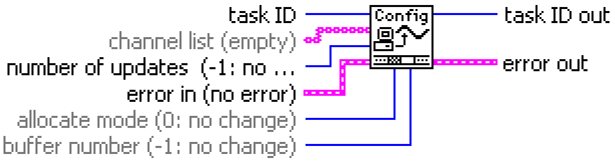
error out
no error
code
0
source

Block Diagram

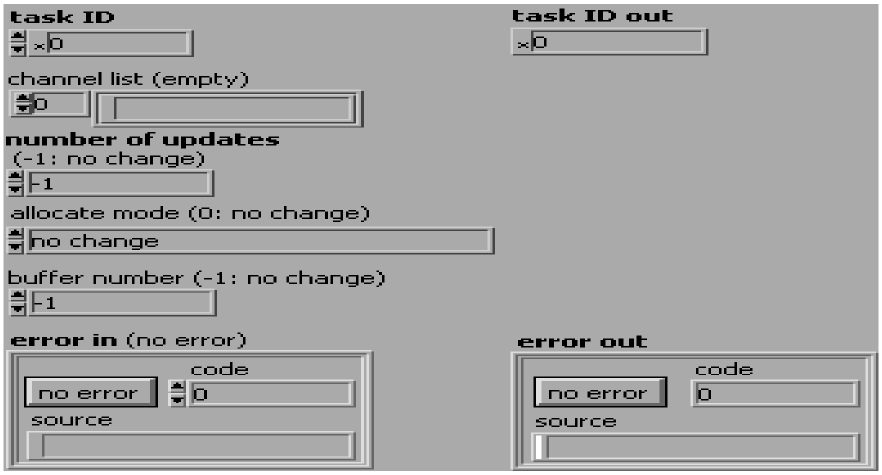


AO Buffer Config.vi

Connector Pane


task ID
channel list (empty)
number of updates (-1: no ...
error in (no error)
allocate mode (0: no change)
buffer number (-1: no change)
Config
task ID out
error out

Front Panel


task ID

0

task ID out

0

channel list (empty)

0

number of updates
(-1: no change)

-1

allocate mode (0: no change)

no change

buffer number (-1: no change)

-1

error in (no error)

no error

code

0

source

error out

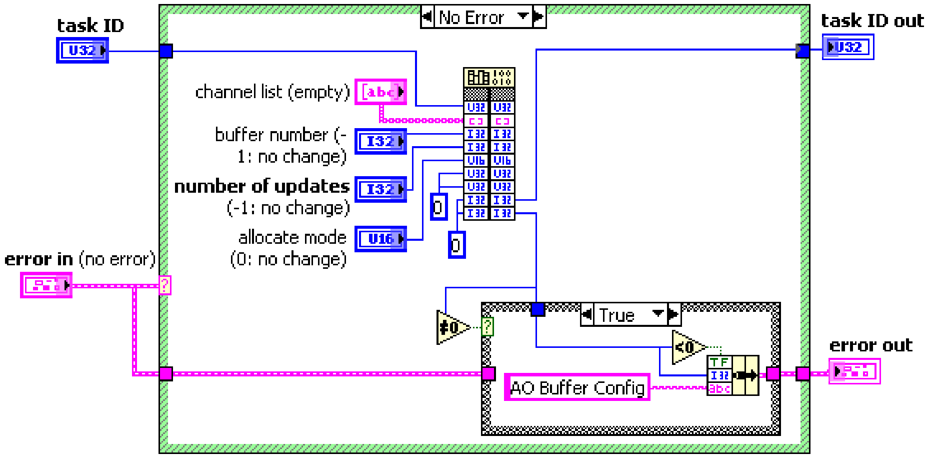
no error

code

0

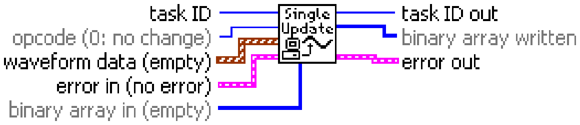
source

Block Diagram



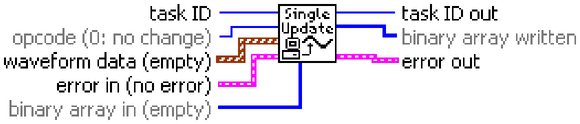
AO Single Update.vi

Connector Pane

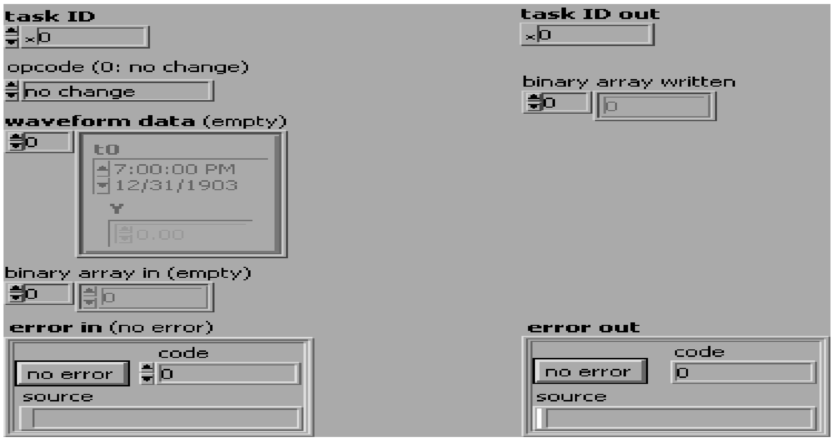


AO Single Update (single-point waveform).vi

Connector Pane



Front Panel



task ID
0

opcode (0: no change)
no change

waveform data (empty)
0
t0
7:00:00 PM
12/31/1903
Y
0.00

binary array in (empty)
0
0

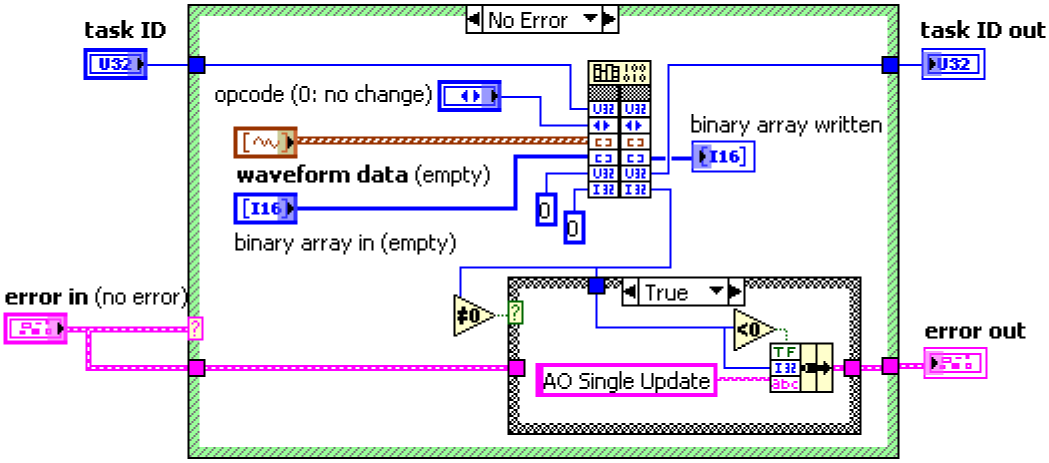
error in (no error)
no error
code
0
source

task ID out
0

binary array written
0
0

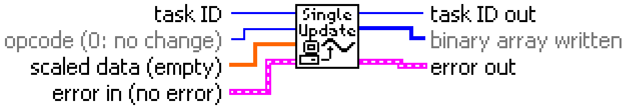
error out
no error
code
0
source

Block Diagram

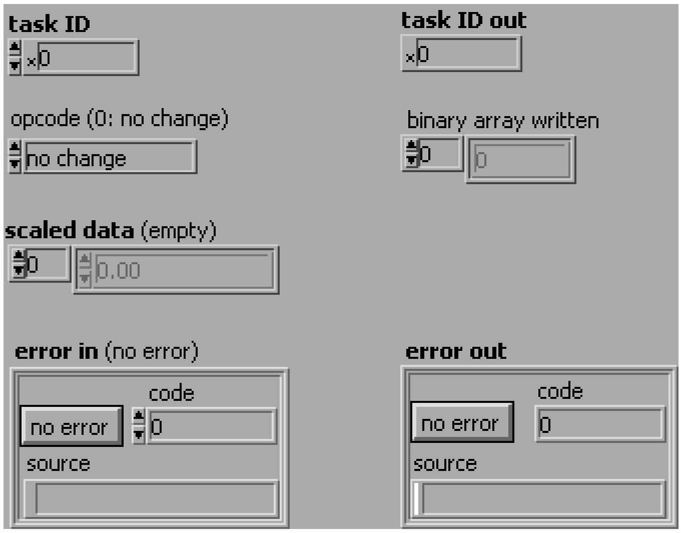


AO Single Update (scaled array).vi

Connector Pane



Front Panel



task ID
0

task ID out
0

opcode (0: no change)
no change

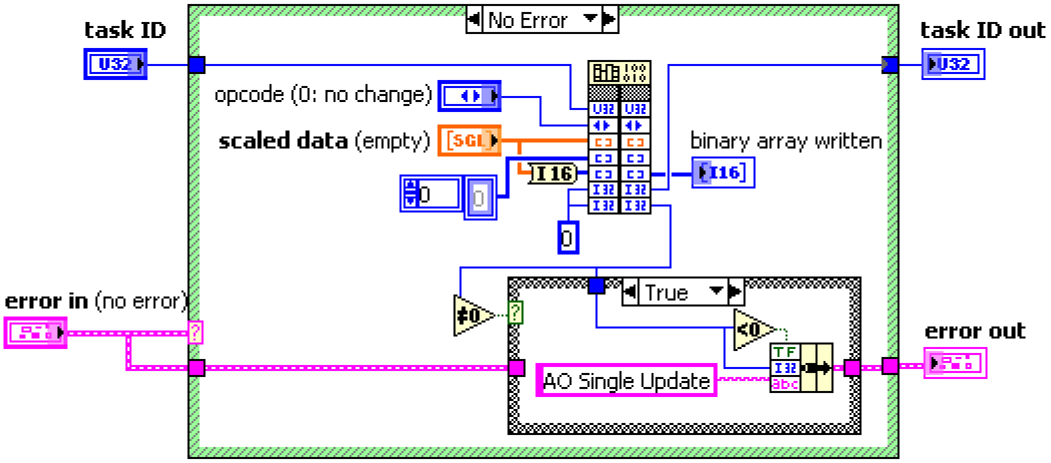
binary array written
0 0

scaled data (empty)
0 0.00

error in (no error)
no error
code
0
source

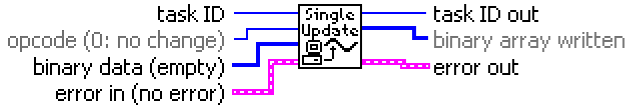
error out
no error
code
0
source

Block Diagram

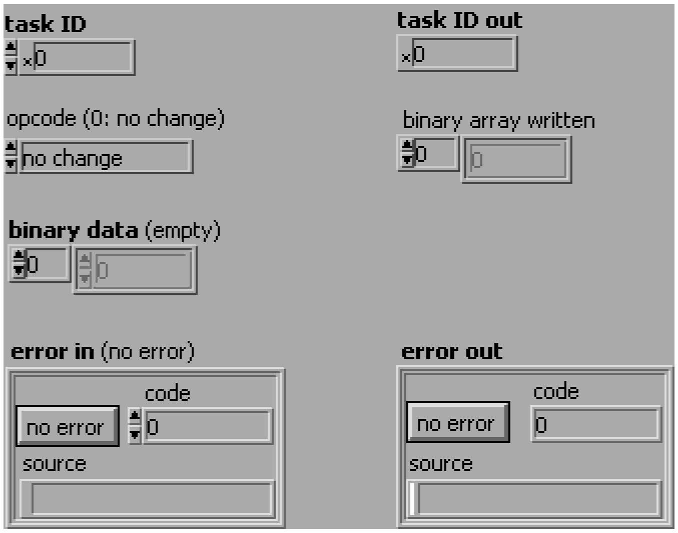


AO Single Update (binary array).vi

Connector Pane



Front Panel



task ID
0

task ID out
0

opcode (0: no change)
no change

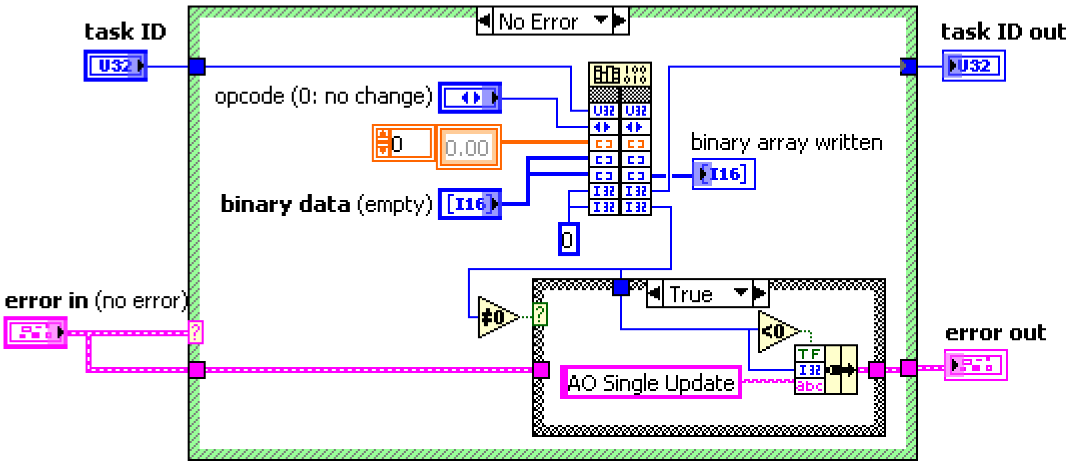
binary array written
0 0

binary data (empty)
0 0

error in (no error)
no error
code 0
source

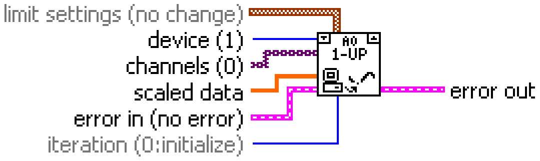
error out
no error
code 0
source

Block Diagram

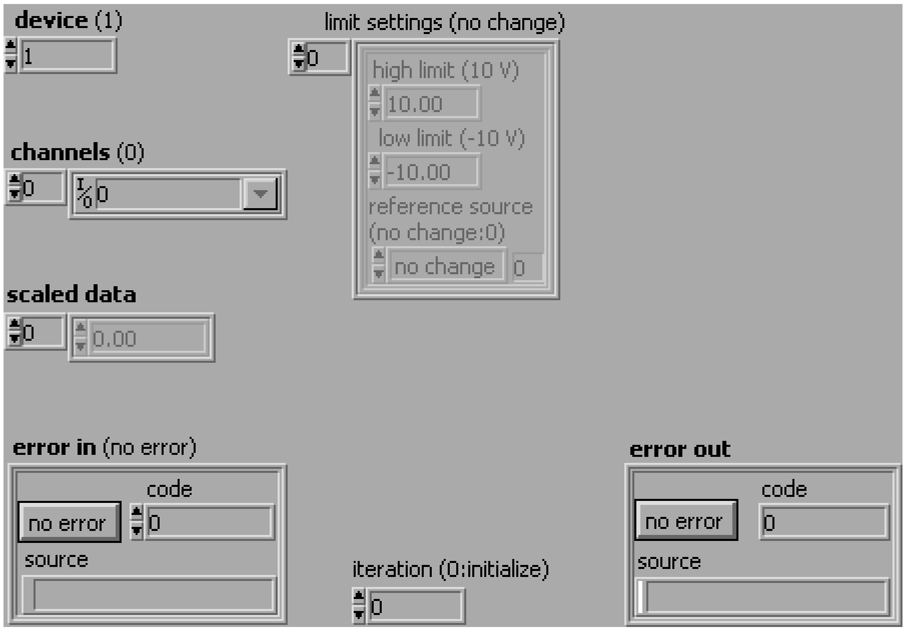


AO Write One Update (scaled array).vi

Connector Pane



Front Panel


device (1): 1

limit settings (no change): 0

high limit (10 V): 10.00

low limit (-10 V): -10.00

reference source (no change:0): no change 0

channels (0): 0 | 0

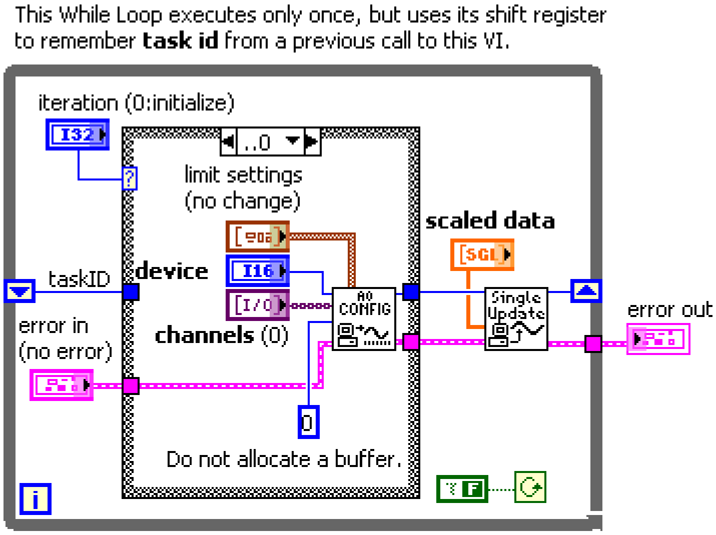
scaled data: 0 | 0.00

error in (no error): no error; code 0; source

iteration (0:initialize): 0

error out: no error; code 0; source

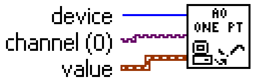
Block Diagram



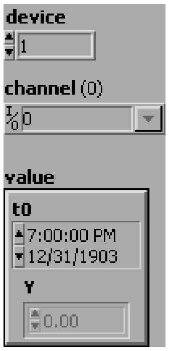
This While Loop executes only once, but uses its shift register to remember **task id** from a previous call to this VI.

AO Update Channel (single-point waveform).vi

Connector Pane



Front Panel



device

1

channel (0)

0

value

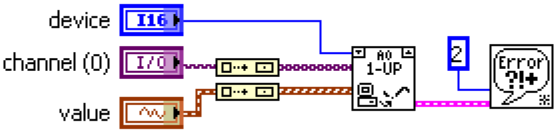
t0

7:00:00 PM
12/31/1903

Y

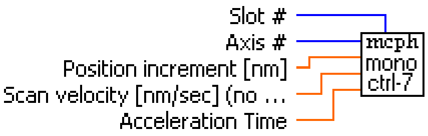
0.00

Block Diagram

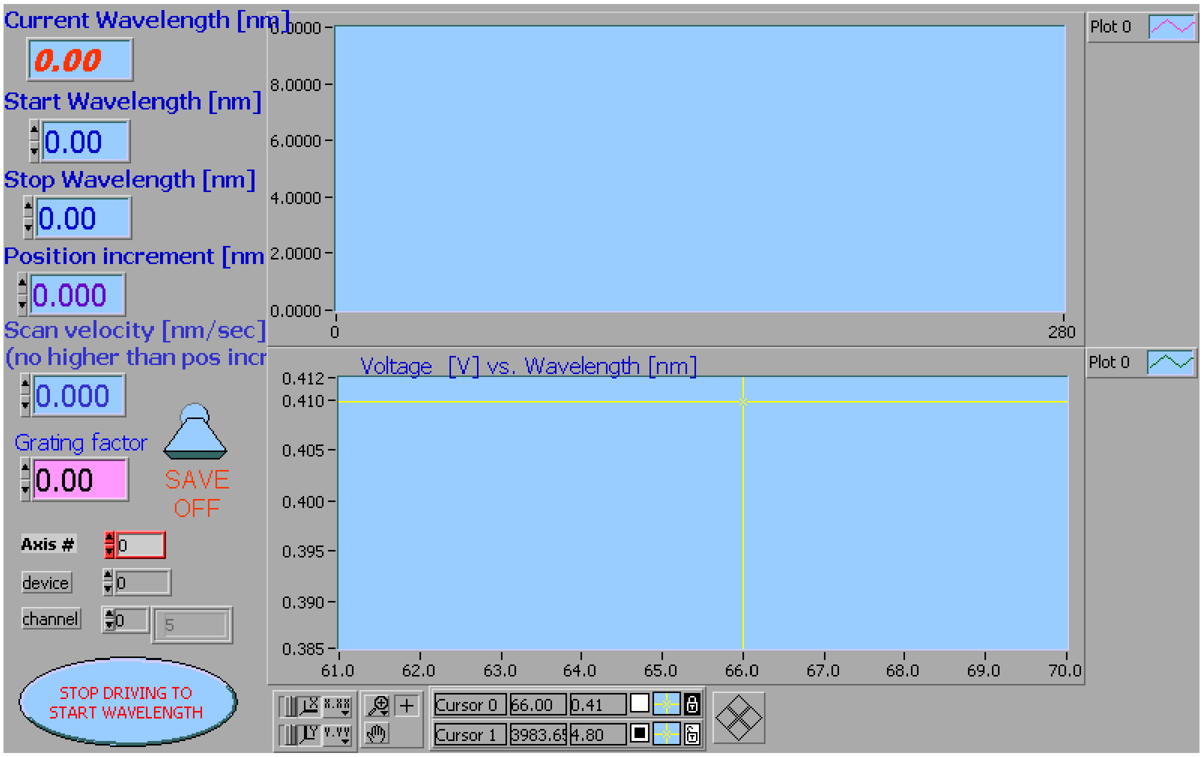


Oscilloscope.vi

Connector Pane



Front Panel


Current Wavelength [nm]: 0.00

Start Wavelength [nm]: 0.00

Stop Wavelength [nm]: 0.00

Position increment [nm: 0.000

Scan velocity [nm/sec] (no higher than pos incr: 0.000

Grating factor: 0.00

Axis #: 0

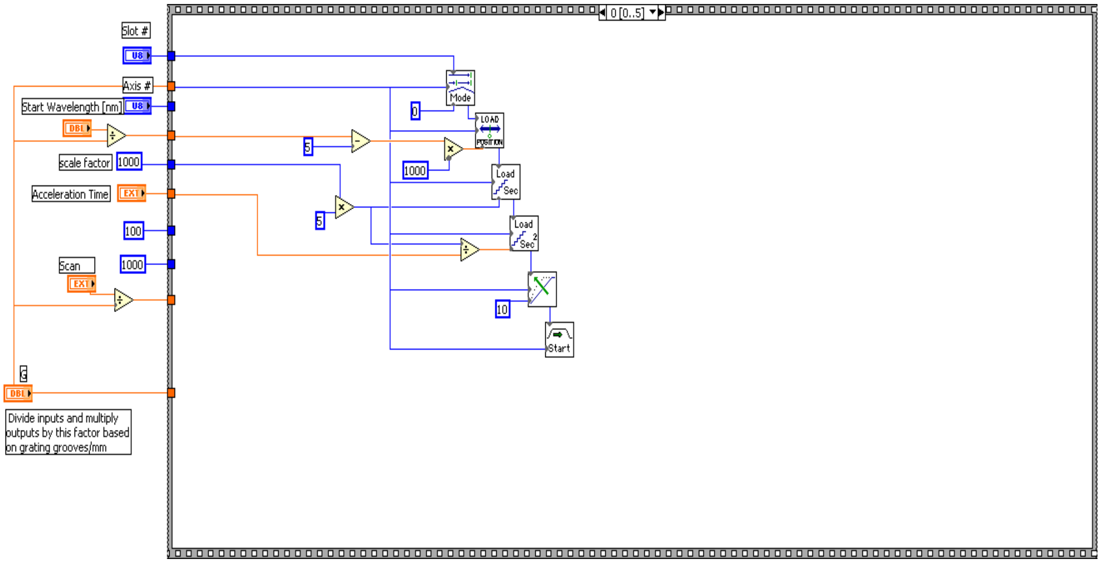
device: 0

channel: 0

Cursor 0: 66.00, 0.41

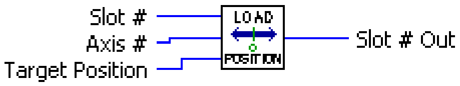
Cursor 1: 3983.6, 4.80

Block Diagram

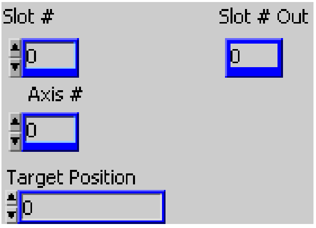


Load Target Position

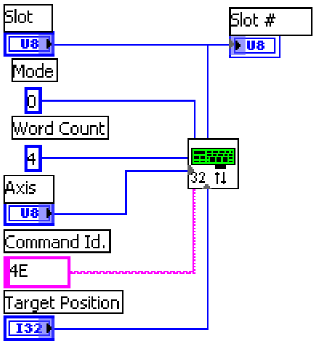
Connector Pane



Front Panel

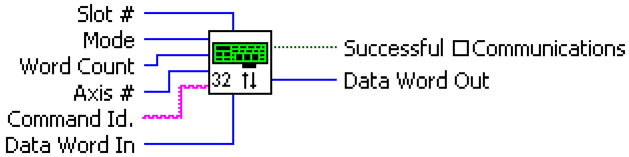


Block Diagram

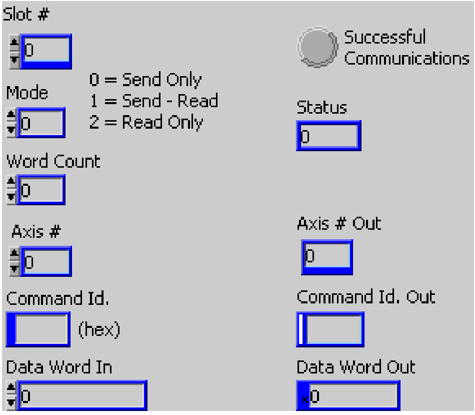


Communicate

Connector Pane



Front Panel


Slot #

0

Mode

0

0 = Send Only
1 = Send - Read
2 = Read Only

Word Count

0

Axis #

0

Command Id.

(hex)

Data Word In

0

Successful Communications

Status

0

Axis # Out

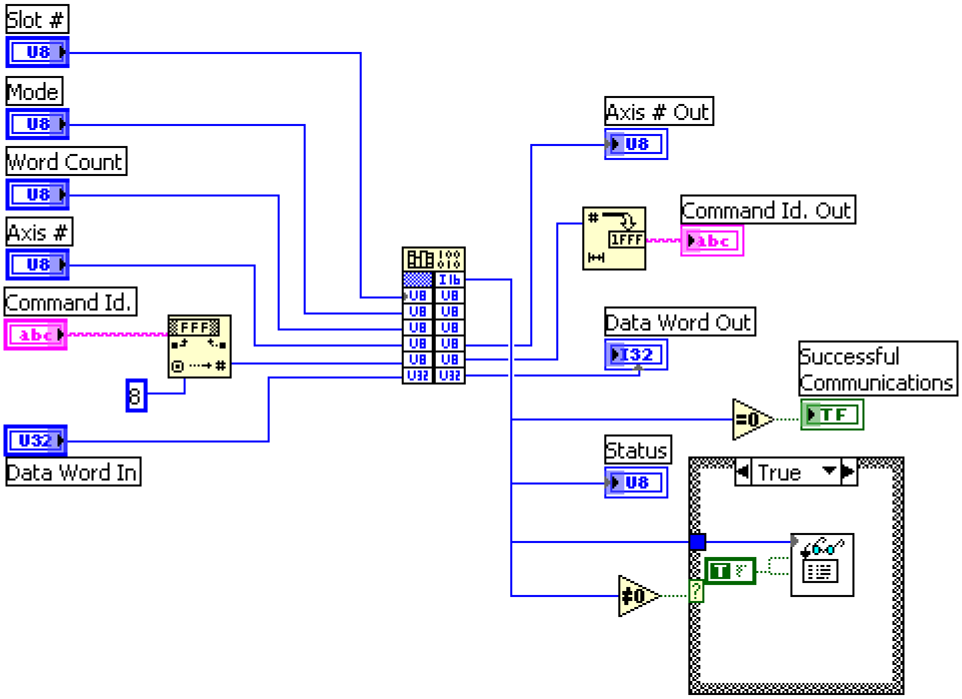
0

Command Id. Out

Data Word Out

0

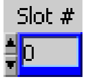
Block Diagram



ValueMotion Slot #

Connector Pane

Front Panel

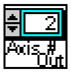


Slot #

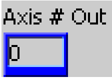
0

ValueMotion Axis # Out

Connector Pane



Front Panel

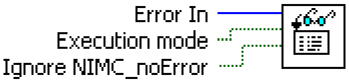


Axis # Out

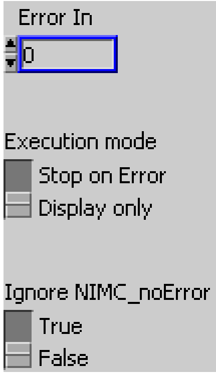
0

Simple Error Handler

Connector Pane



Front Panel



Error In

0

Execution mode

Stop on Error

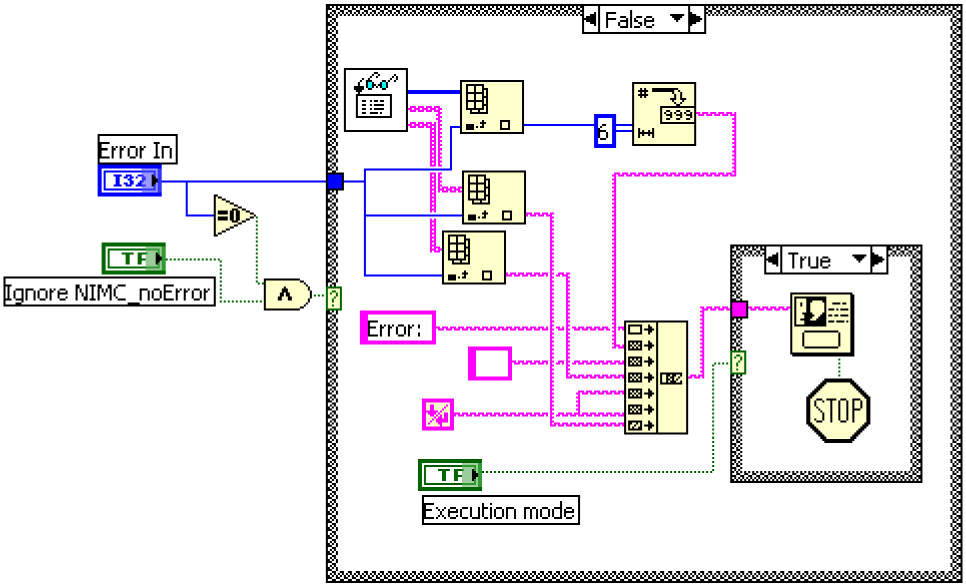
Display only

Ignore NIMC_noError

True

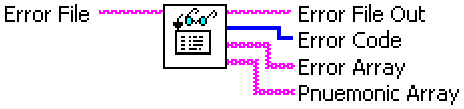
False

Block Diagram

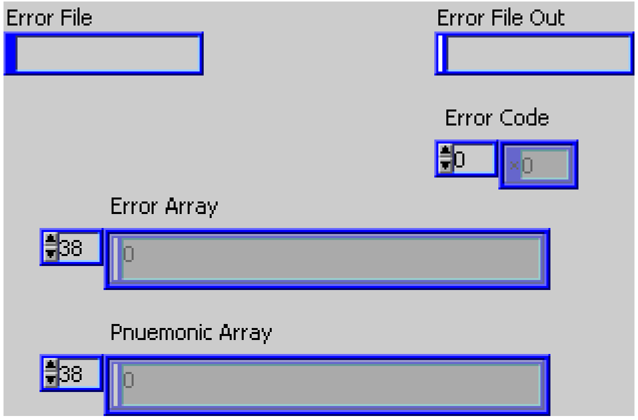


Error File Parser

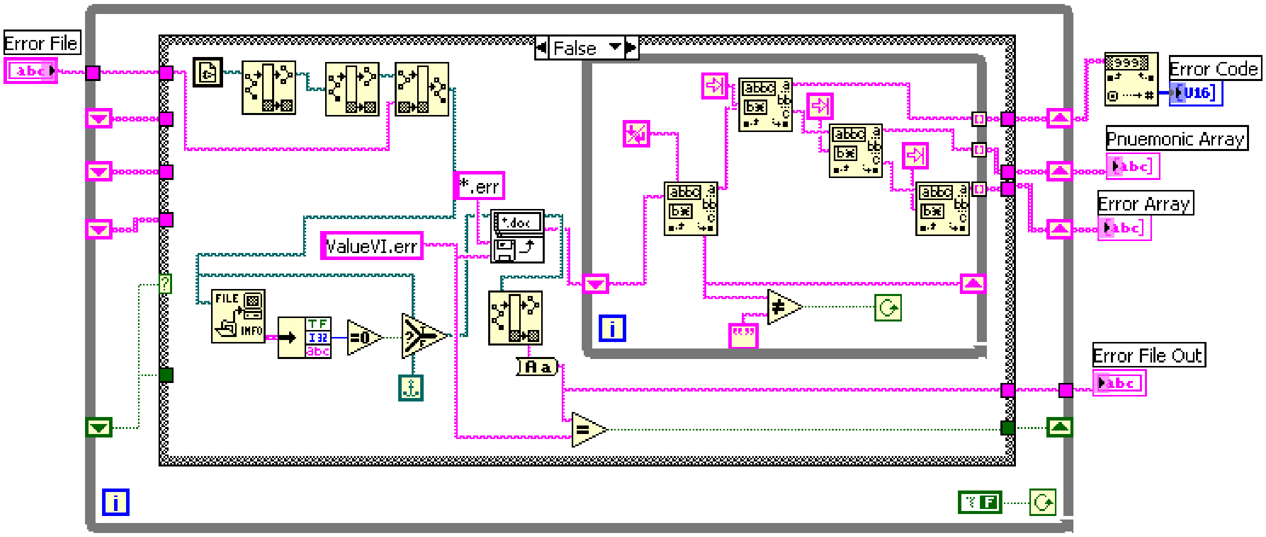
Connector Pane



Front Panel

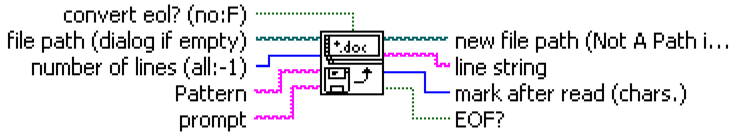


Block Diagram

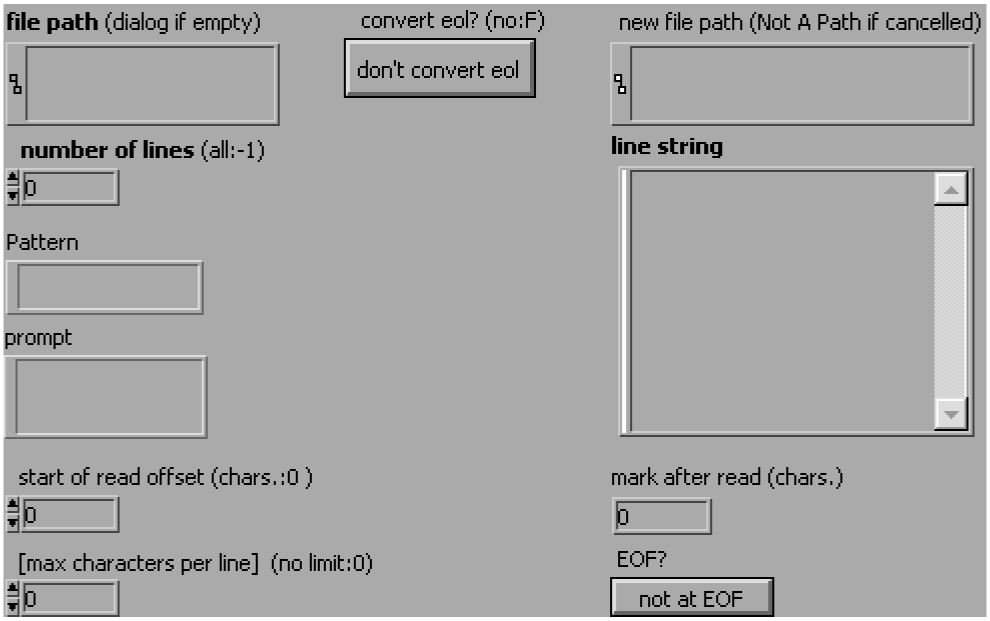


Read Lines From Files

Connector Pane



Front Panel



file path (dialog if empty)

convert eol? (no:F)

don't convert eol

new file path (Not A Path if cancelled)

number of lines (all:-1)

0

line string

Pattern

prompt

start of read offset (chars.:0)

0

mark after read (chars.)

0

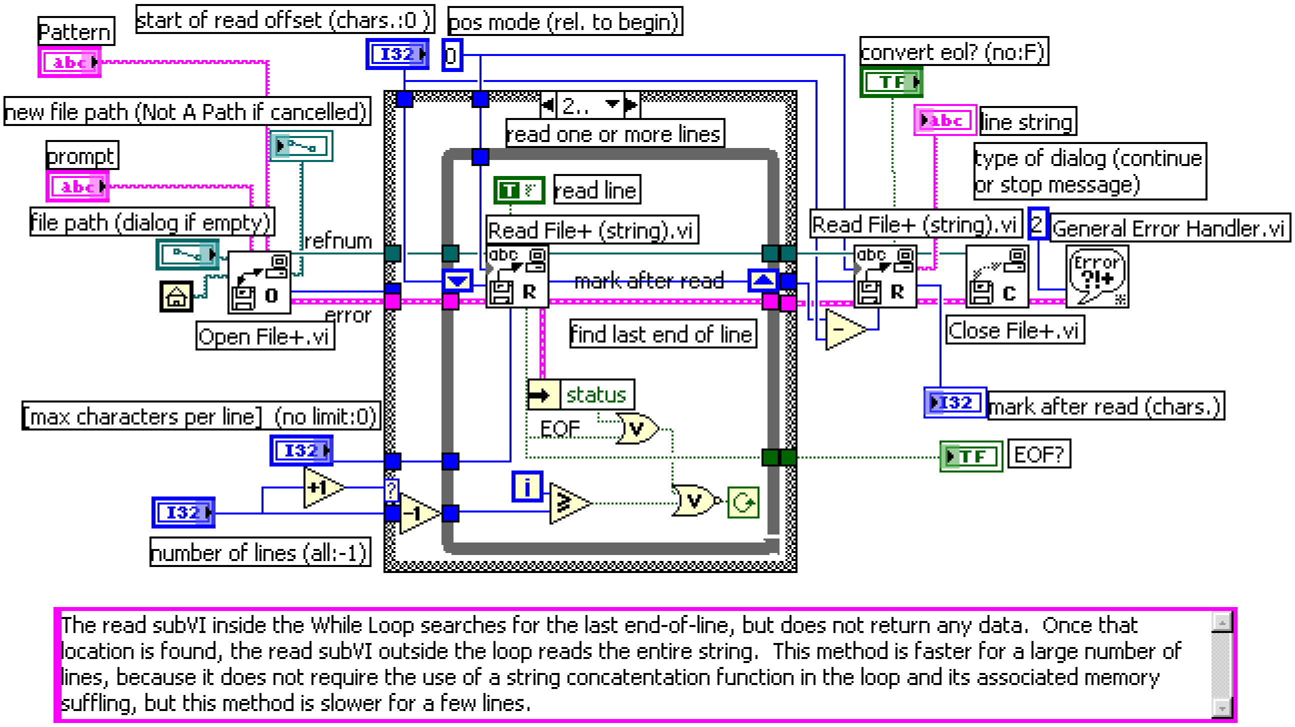
[max characters per line] (no limit:0)

0

EOF?

not at EOF

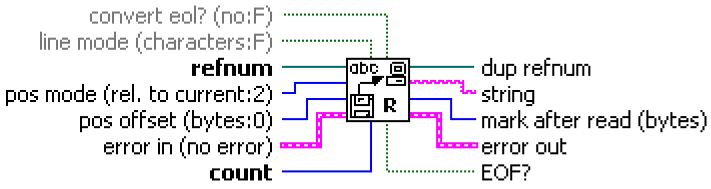
Block Diagram



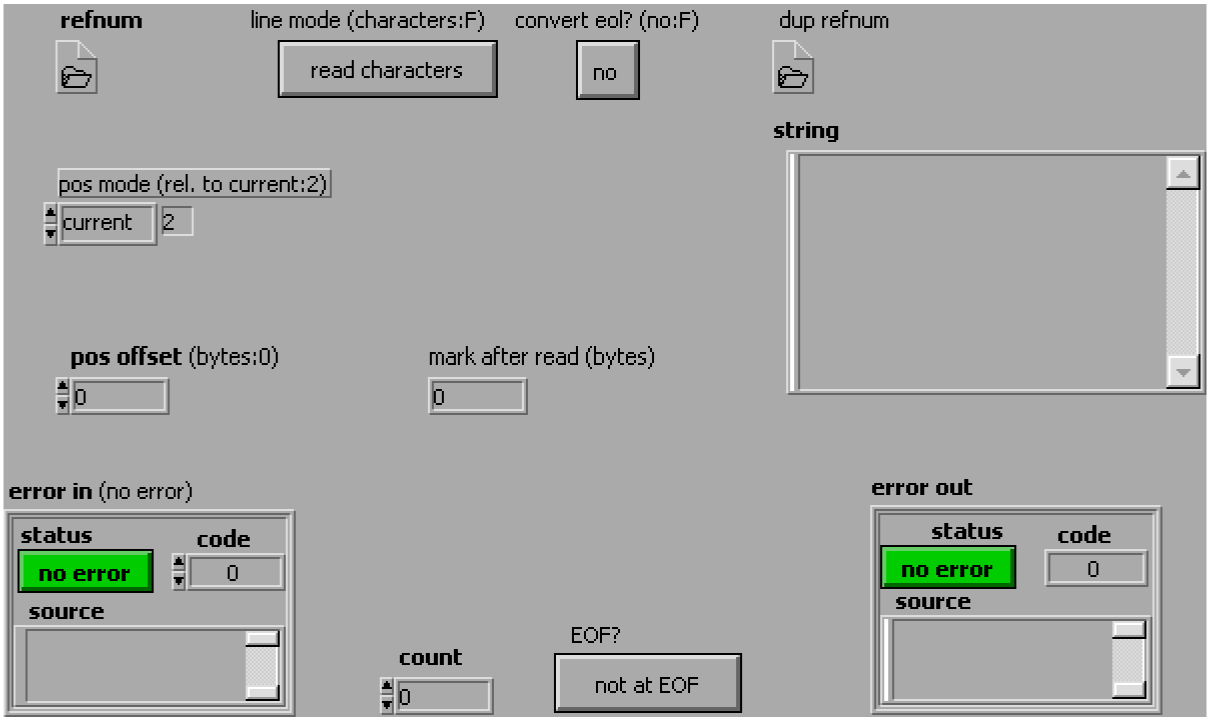
The read subVI inside the While Loop searches for the last end-of-line, but does not return any data. Once that location is found, the read subVI outside the loop reads the entire string. This method is faster for a large number of lines, because it does not require the use of a string concatentation function in the loop and its associated memory suffling, but this method is slower for a few lines.

Read File+ (string).vi

Connector Pane



Front Panel



refnum

line mode (characters:F): read characters

convert eol? (no:F): no

dup refnum

string

pos mode (rel. to current:2): current 2

pos offset (bytes:0): 0

mark after read (bytes): 0

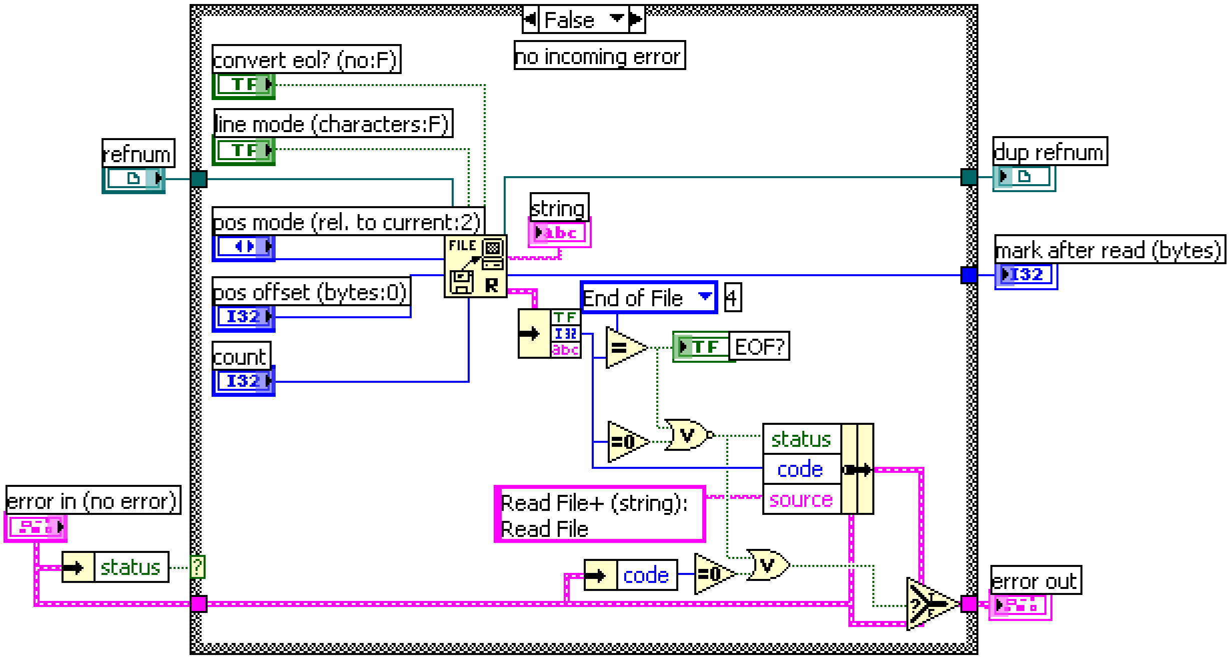
error in (no error)
status: no error
code: 0
source

count: 0

EOF?: not at EOF

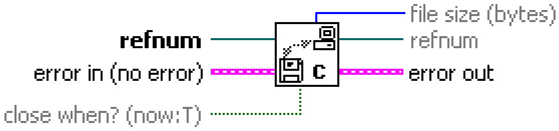
error out
status: no error
code: 0
source

Block Diagram

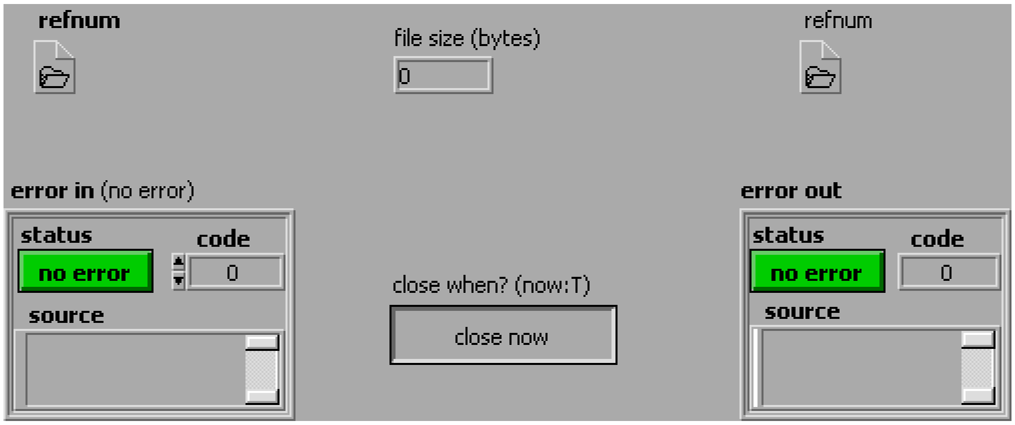


Close File+.vi

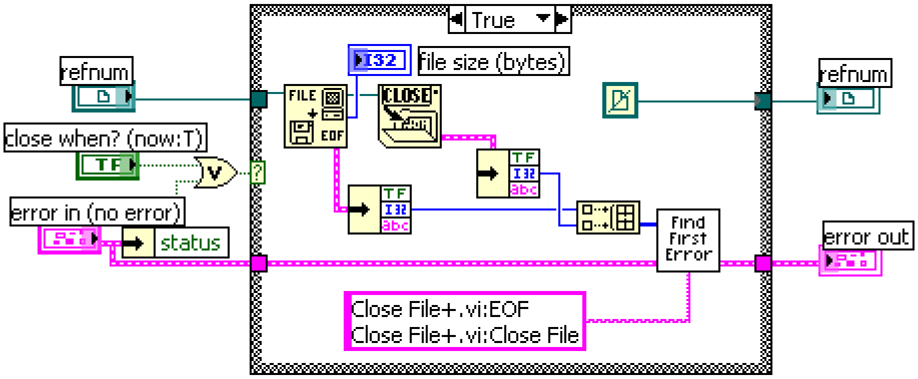
Connector Pane



Front Panel

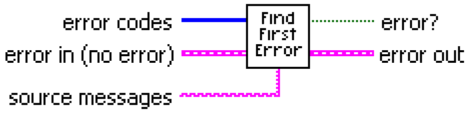


Block Diagram

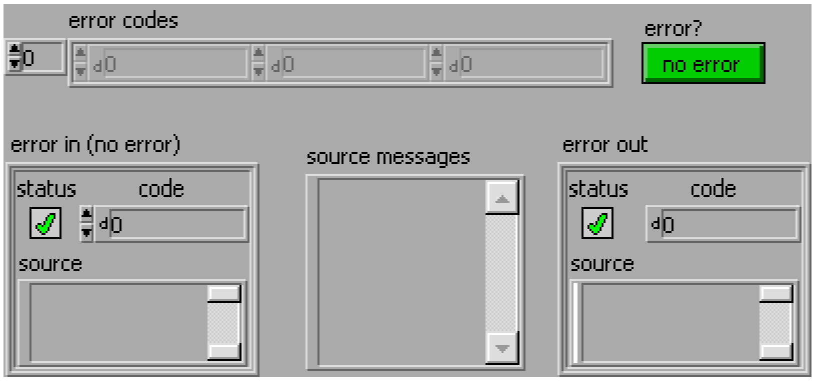


Find First Error.vi

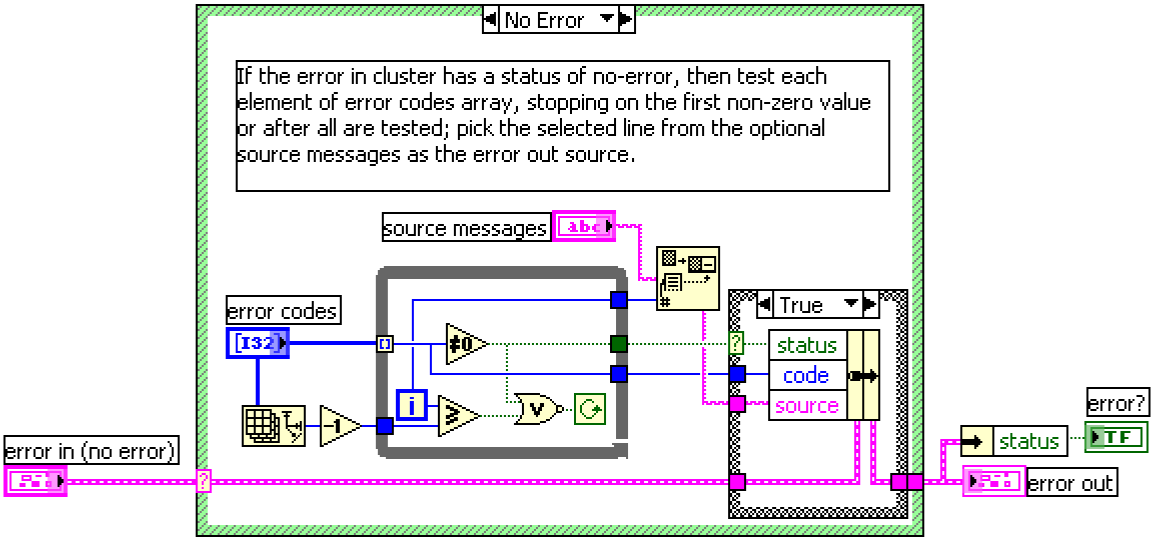
Connector Pane



Front Panel



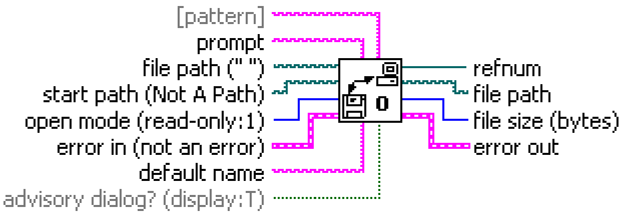
Block Diagram



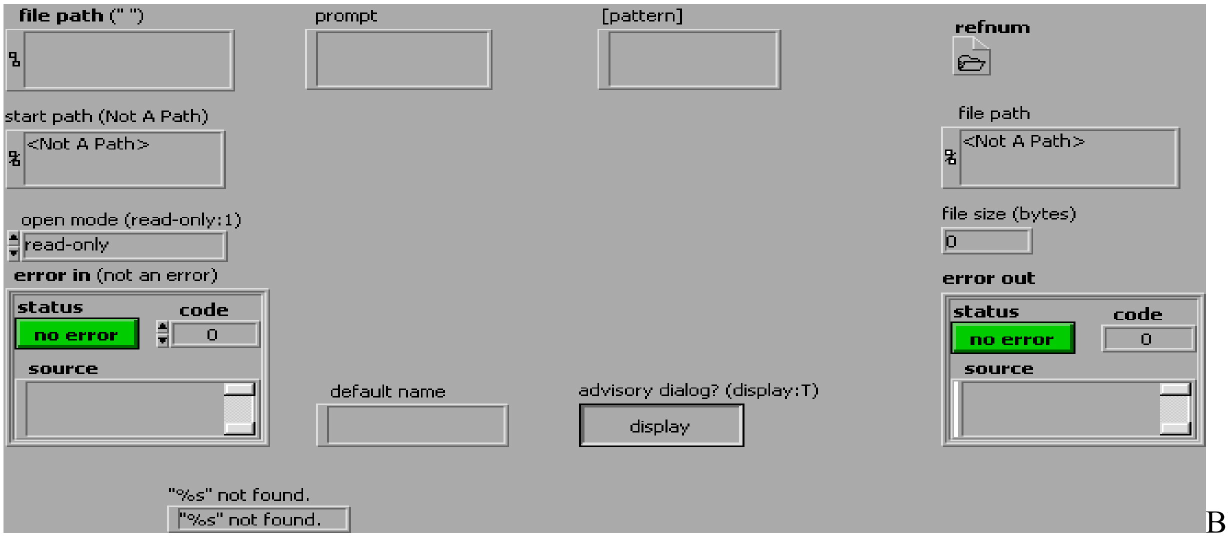
If the error in cluster has a status of no-error, then test each element of error codes array, stopping on the first non-zero value or after all are tested; pick the selected line from the optional source messages as the error out source.

Open File+.vi

Connector Pane



Front Panel



file path (" ")

prompt

[pattern]

refnum

start path (Not A Path)
<Not A Path>

file path
<Not A Path>

open mode (read-only:1)
read-only

file size (bytes)
0

error in (not an error)
status: no error
code: 0
source

error out
status: no error
code: 0
source

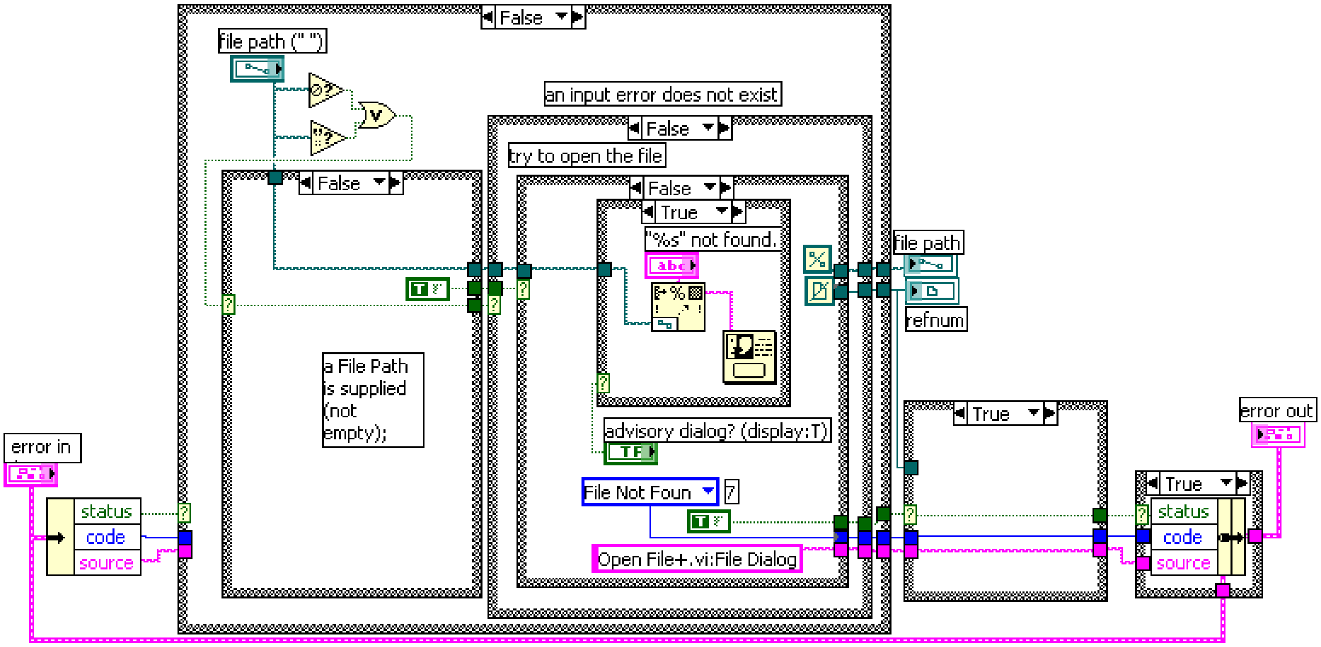
default name

advisory dialog? (display:T)
display

"%s" not found.
"%s" not found.

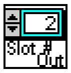
Bl

Block Diagram

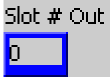


ValueMotion Slot # Out

Connector Pane

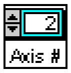


Front Panel

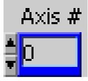


ValueMotion Axis #

Connector Pane

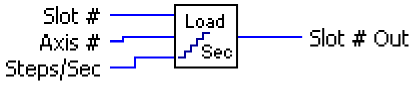


Front Panel

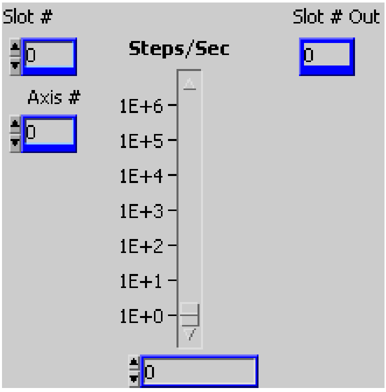


Load Steps/Sec

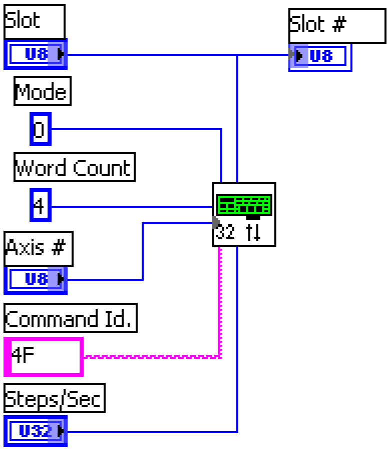
Connector Pane



Front Panel

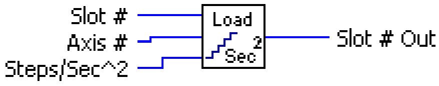


Block Diagram

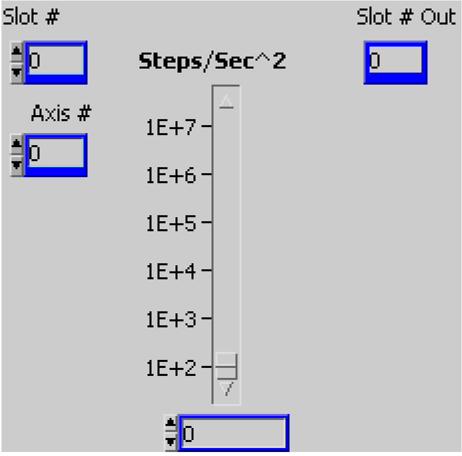


Load Steps/Sec^2

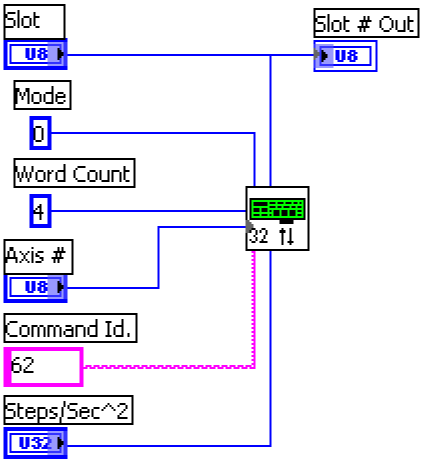
Connector Pane



Front Panel

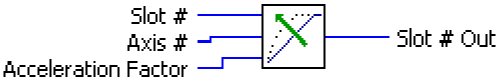


Block Diagram

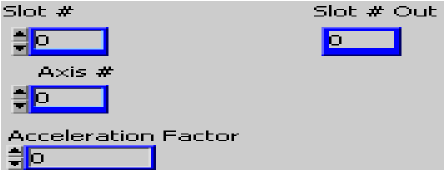


Load Acceleration Factor

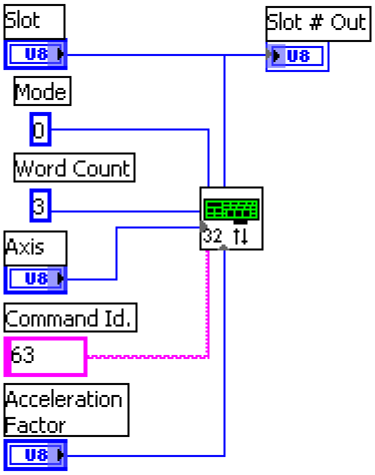
Connector Pane



Front Panel

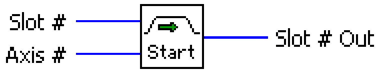


Block Diagram

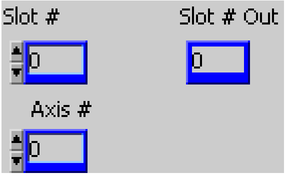


Start Motion

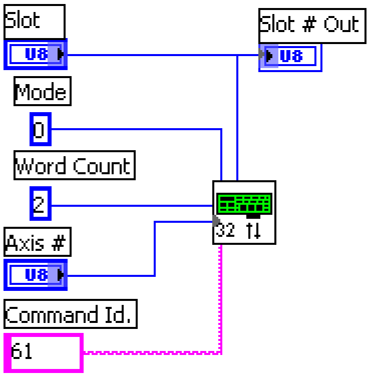
Connector Pane



Front Panel

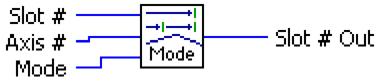


Block Diagram

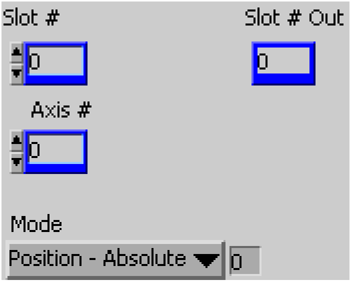


Set Position Mode

Connector Pane



Front Panel



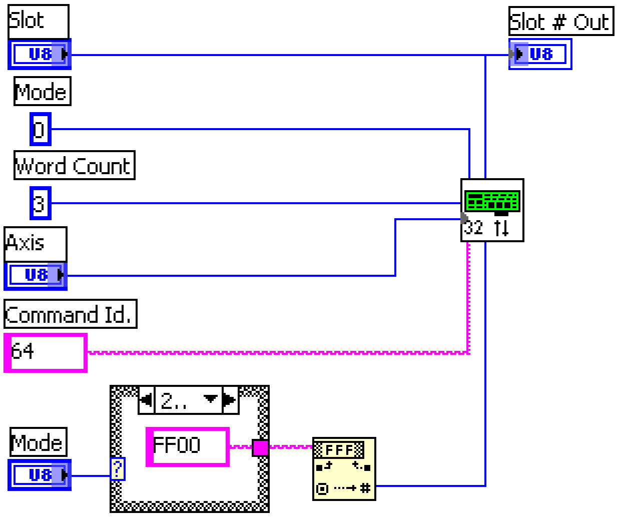
Slot # 0

Slot # Out 0

Axis # 0

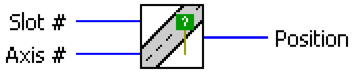
Mode Position - Absolute 0

Block Diagram

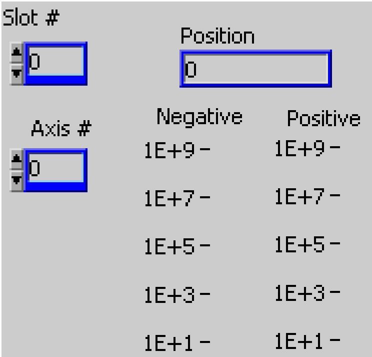


Read Position

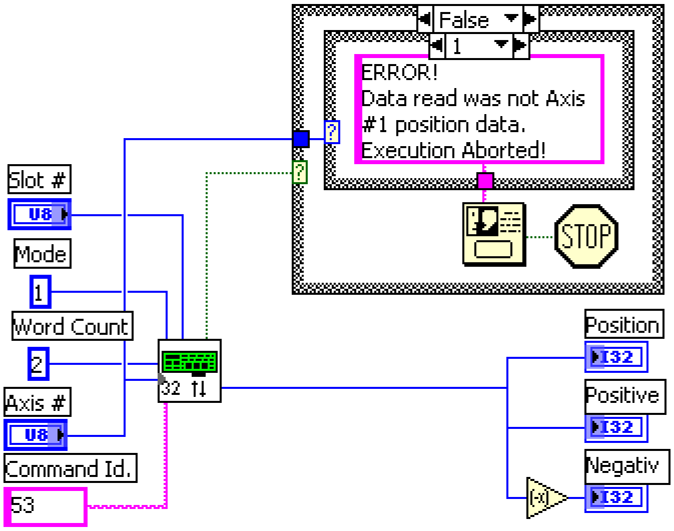
Connector Pane



Front Panel

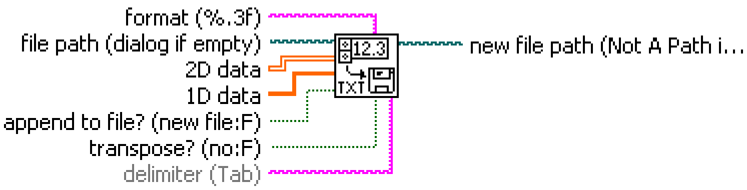


Block Diagram

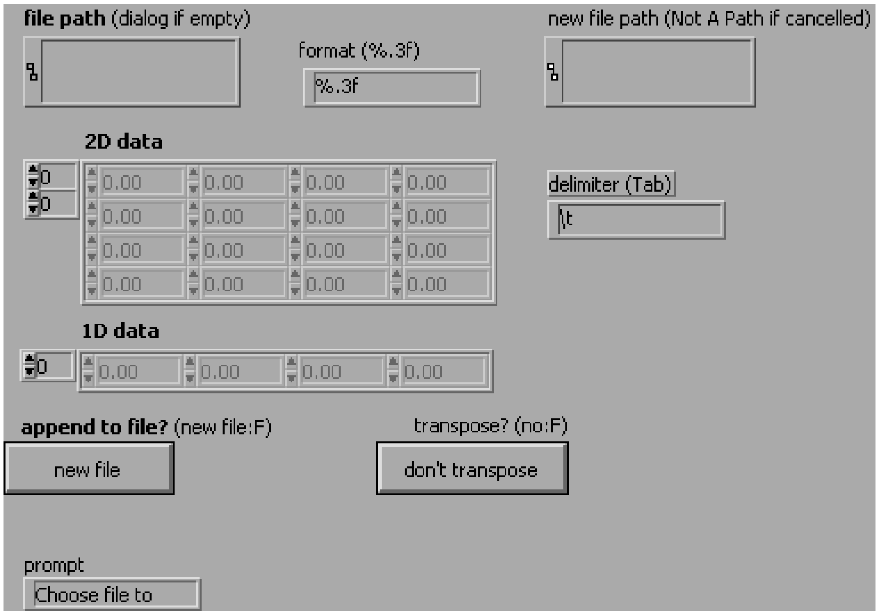

ERROR!
Data read was not Axis
#1 position data.
Execution Aborted!

Write To Spreadsheet File.vi

Connector Pane


format (%.3f)
file path (dialog if empty)
2D data
1D data
append to file? (new file:F)
transpose? (no:F)
delimiter (Tab)
new file path (Not A Path i...
12.3
TXT

Front Panel



file path (dialog if empty)

format (%.3f): %.3f

new file path (Not A Path if cancelled)

2D data

0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00

delimiter (Tab): \t

1D data

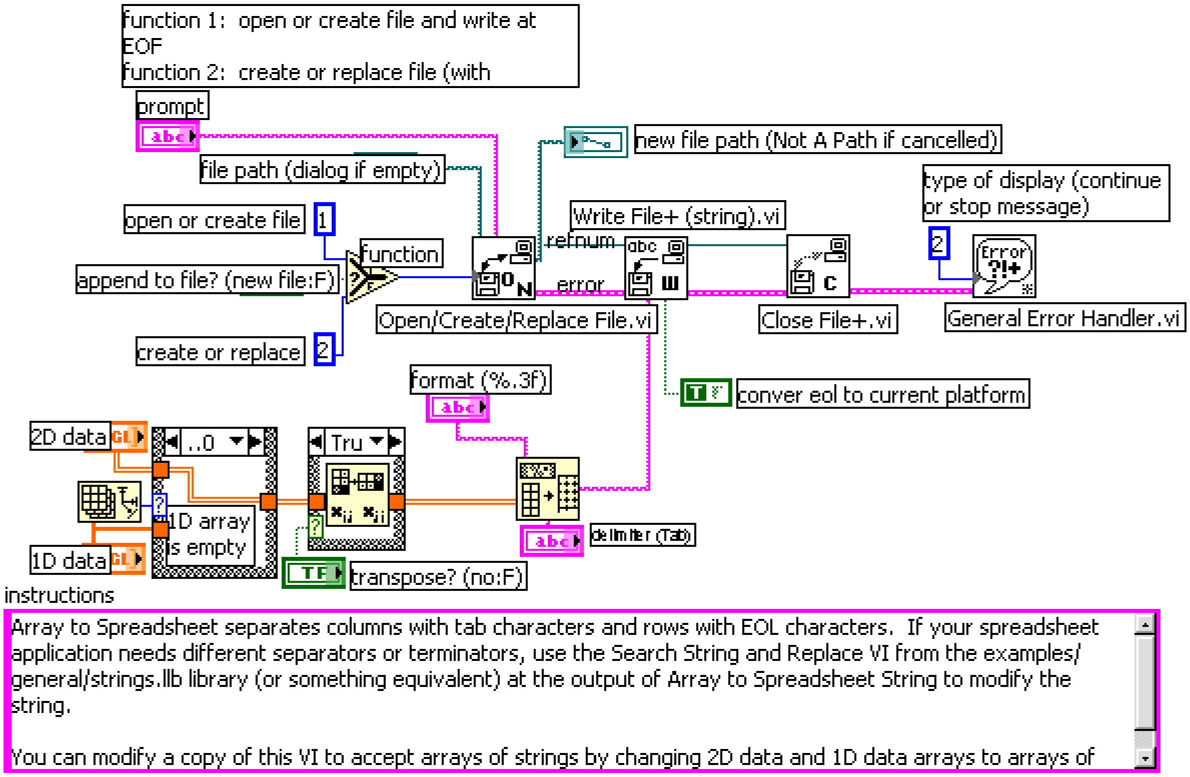
0.00	0.00	0.00	0.00

append to file? (new file:F): new file

transpose? (no:F): don't transpose

prompt: Choose file to

Block Diagram



function 1: open or create file and write at EOF
function 2: create or replace file (with

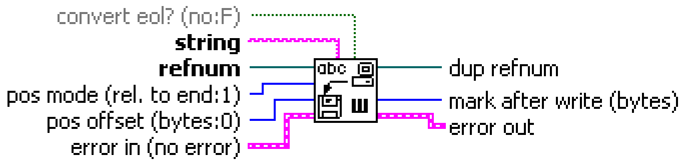
instructions

Array to Spreadsheet separates columns with tab characters and rows with EOL characters. If your spreadsheet application needs different separators or terminators, use the Search String and Replace VI from the examples/general/strings.llb library (or something equivalent) at the output of Array to Spreadsheet String to modify the string.

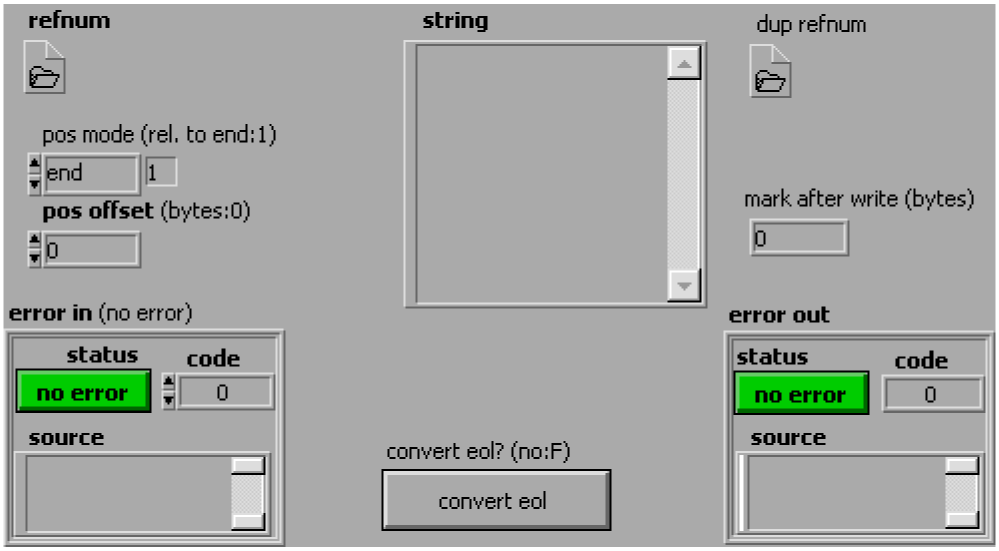
You can modify a copy of this VI to accept arrays of strings by changing 2D data and 1D data arrays to arrays of

Write File+ (string).vi

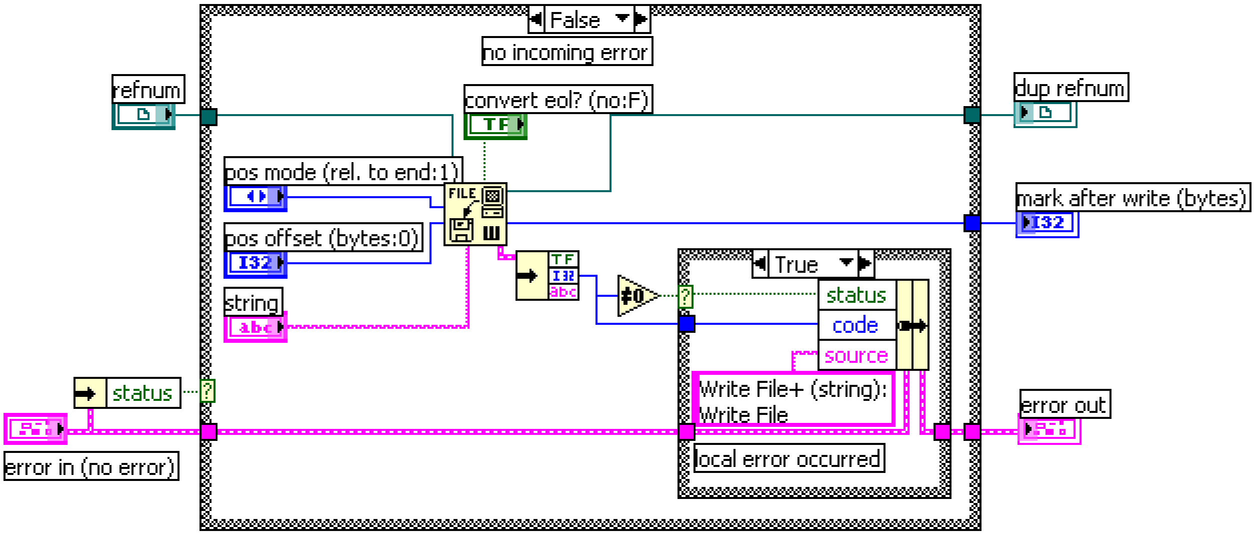
Connector Pane



Front Panel

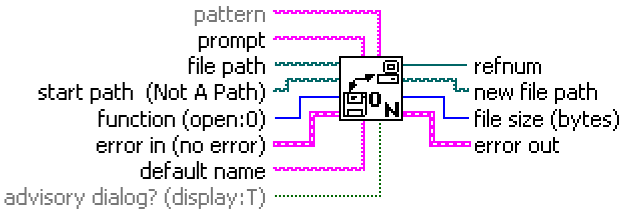


Block Diagram

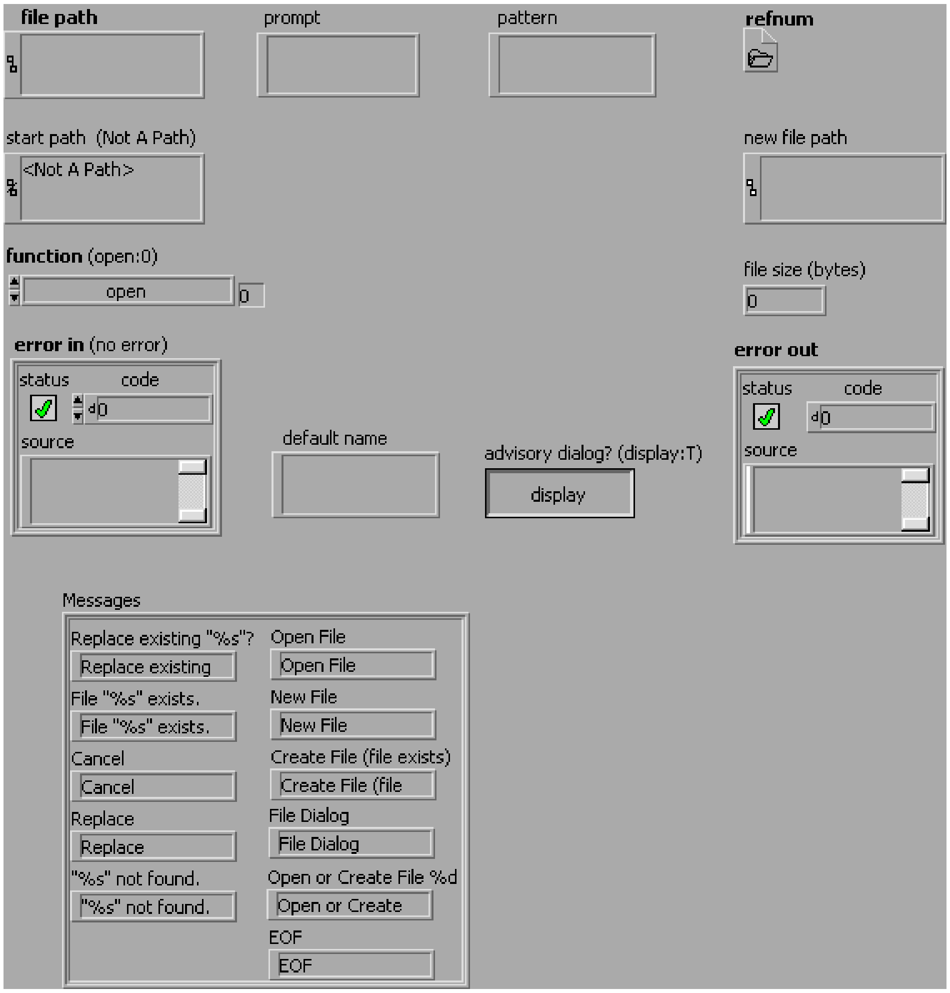


Open/Create/Replace File.vi

Connector Pane



Front Panel



file path

prompt

pattern

refnum

start path (Not A Path)

<Not A Path>

new file path

function (open:0)

open 0

file size (bytes)

0

error in (no error)

status ✓ code 0

source

default name

advisory dialog? (display:T)

display

error out

status ✓ code 0

source

Messages

Replace existing "%s"?

Replace existing

File "%s" exists.

File "%s" exists.

Cancel

Cancel

Replace

Replace

"%s" not found.

"%s" not found.

Open File

Open File

New File

New File

Create File (file exists)

Create File (file

File Dialog

File Dialog

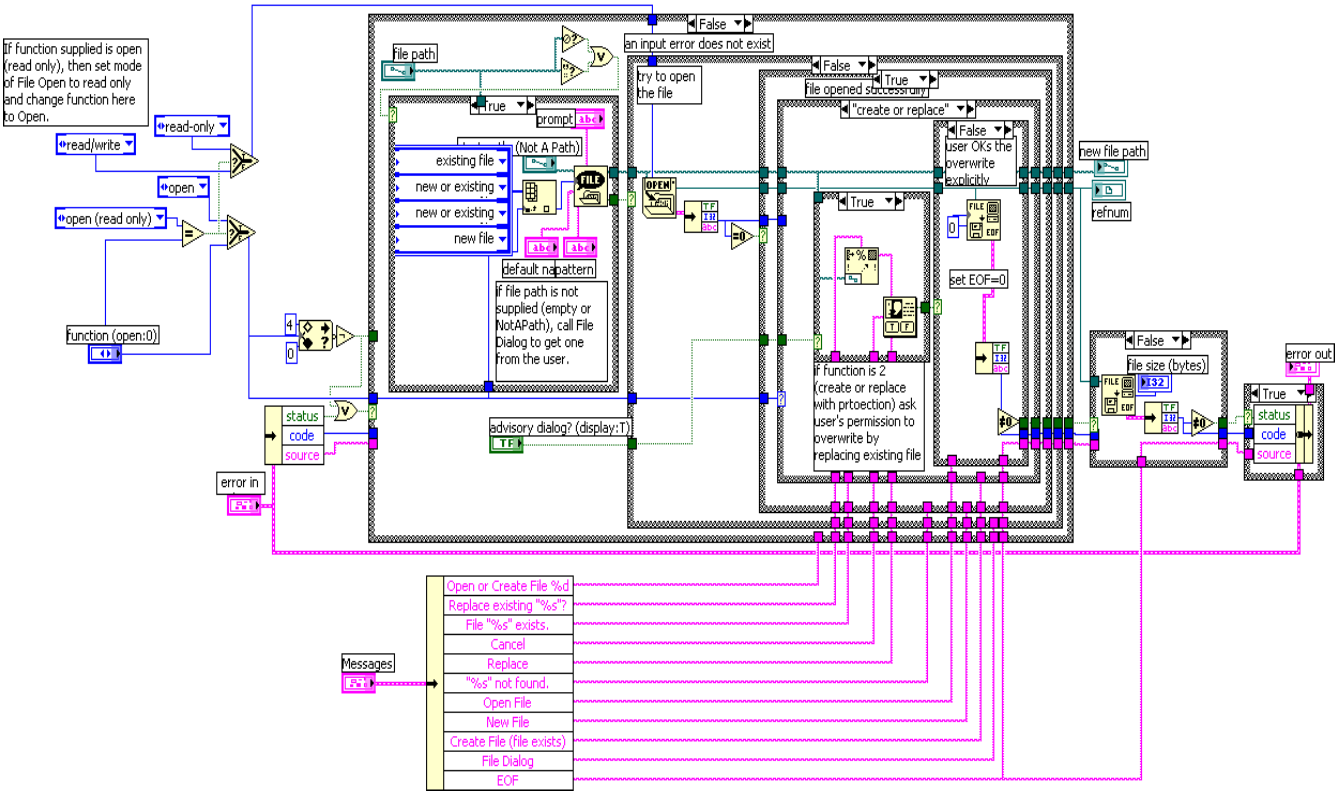
Open or Create File %d

Open or Create

EOF

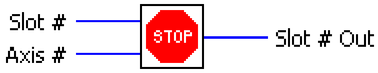
EOF

Block Diagram

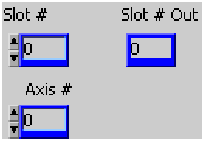


Stop Motion

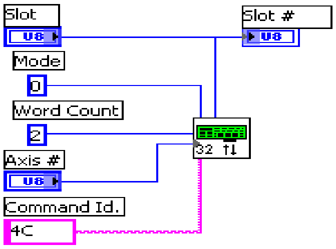
Connector Pane



Front Panel

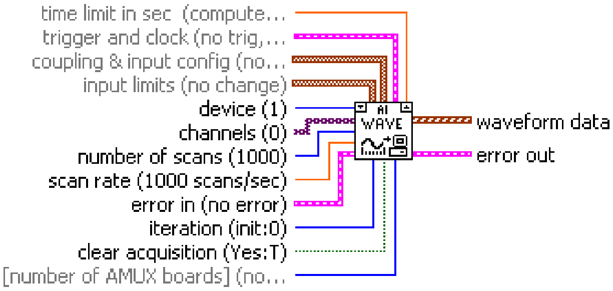


Block Diagram



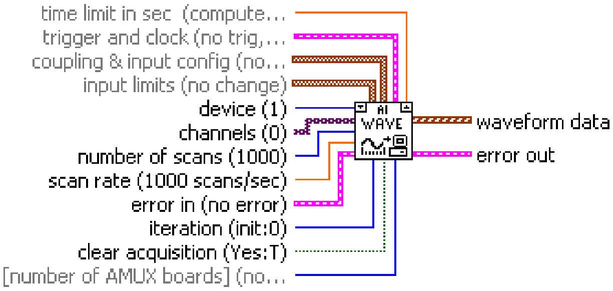
AI Waveform Scan.vi

Connector Pane

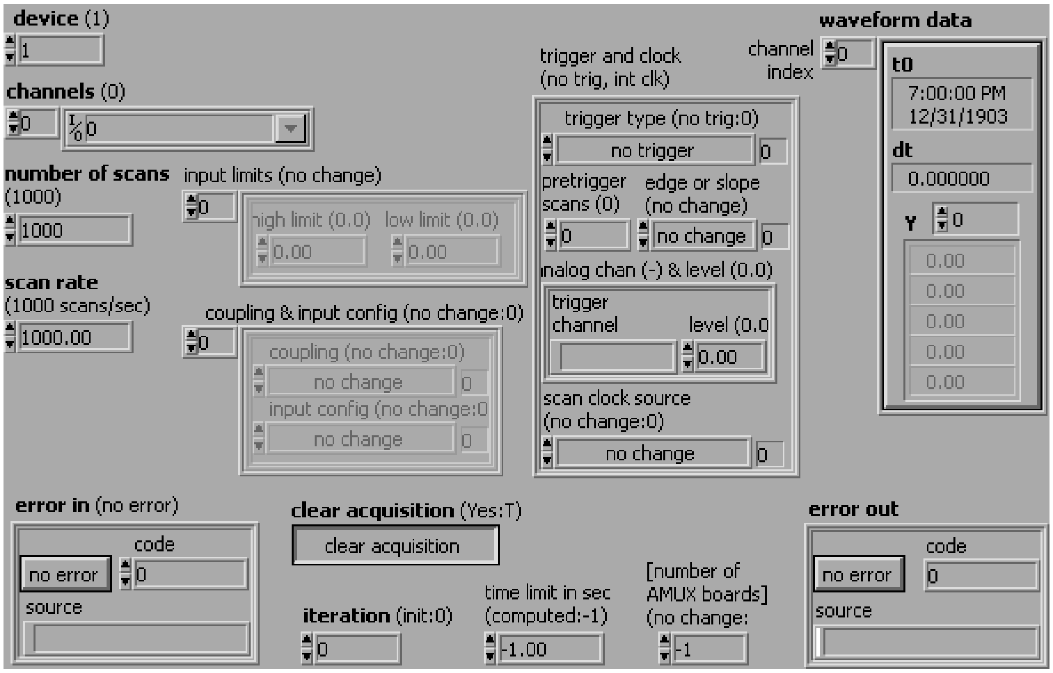

time limit in sec (compute...
trigger and clock (no trig,...
coupling & input config (no...
input limits (no change)
device (1)
channels (0)
number of scans (1000)
scan rate (1000 scans/sec)
error in (no error)
iteration (init:0)
clear acquisition (Yes:T)
[number of AMUX boards] (no...
AI WAVE
waveform data
error out

AI Waveform Scan (waveform).vi

Connector Pane


time limit in sec (compute...
trigger and clock (no trig,...
coupling & input config (no...
input limits (no change)
device (1)
channels (0)
number of scans (1000)
scan rate (1000 scans/sec)
error in (no error)
iteration (init:0)
clear acquisition (Yes:T)
[number of AMUX boards] (no...
AI WAVE
waveform data
error out

Front Panel



device (1)
1

channels (0)
0 | 0

number of scans (1000)
1000

input limits (no change)
0
high limit (0.0) 0.00 | low limit (0.0) 0.00

scan rate (1000 scans/sec)
1000.00

coupling & input config (no change:0)
0
coupling (no change:0)
no change 0
input config (no change:0
no change 0

trigger and clock (no trig, int clk)
trigger type (no trig:0)
no trigger 0
pretrigger scans (0) 0
edge or slope (no change) no change 0
analog chan (-) & level (0.0)
trigger channel | level (0.0 0.00
scan clock source (no change:0)
no change 0

waveform data
channel index 0
t0
7:00:00 PM 12/31/1903
dt
0.000000
Y 0
0.00
0.00
0.00
0.00
0.00

error in (no error)
no error | code 0
source

clear acquisition (Yes:T)
clear acquisition

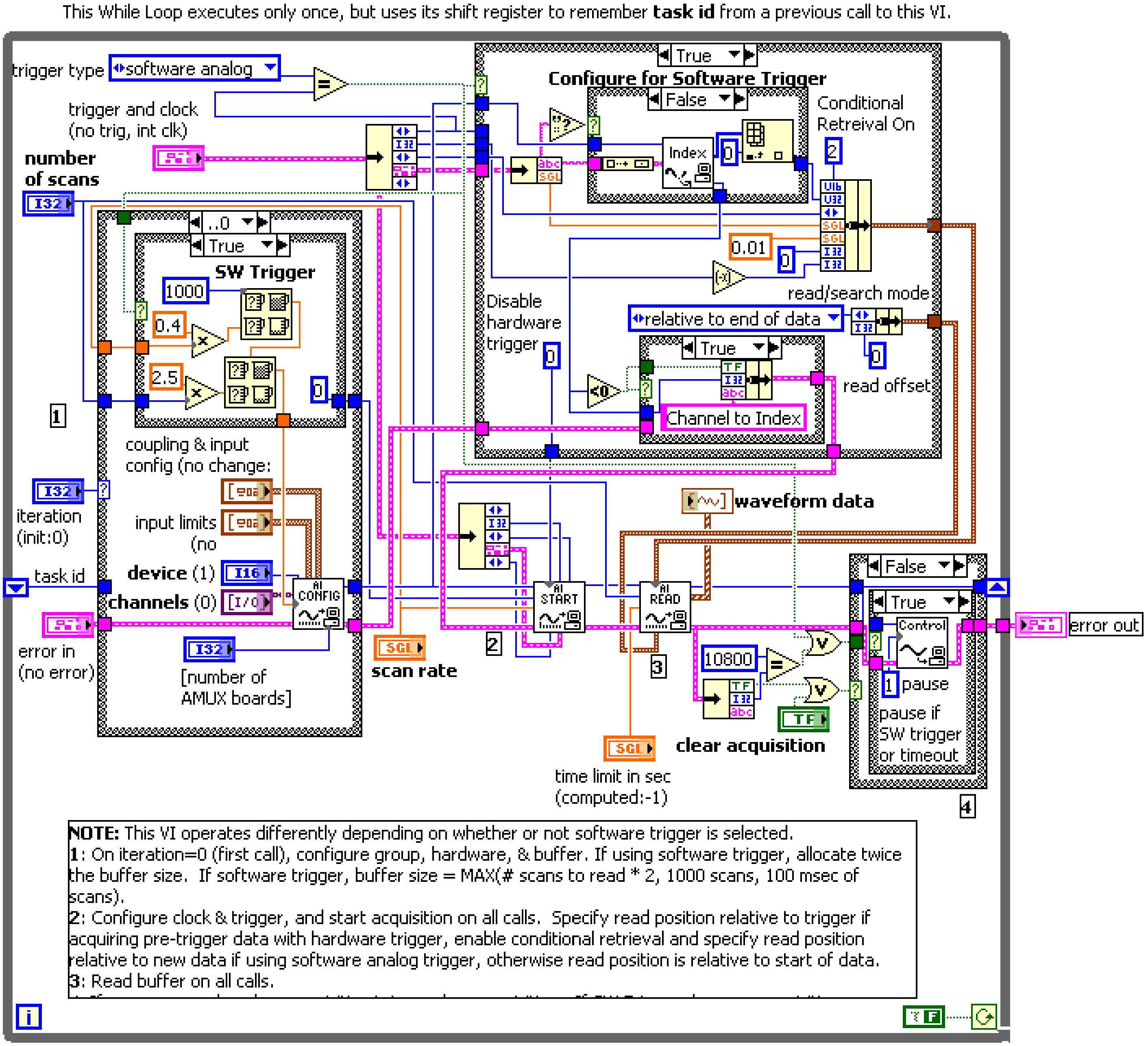
iteration (init:0)
0

time limit in sec (computed:-1)
-1.00

[number of AMUX boards] (no change:
-1

error out
no error | code 0
source

Block Diagram

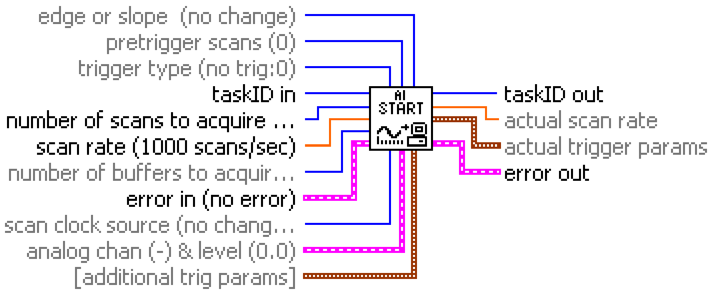


This While Loop executes only once, but uses its shift register to remember **task id** from a previous call to this VI.

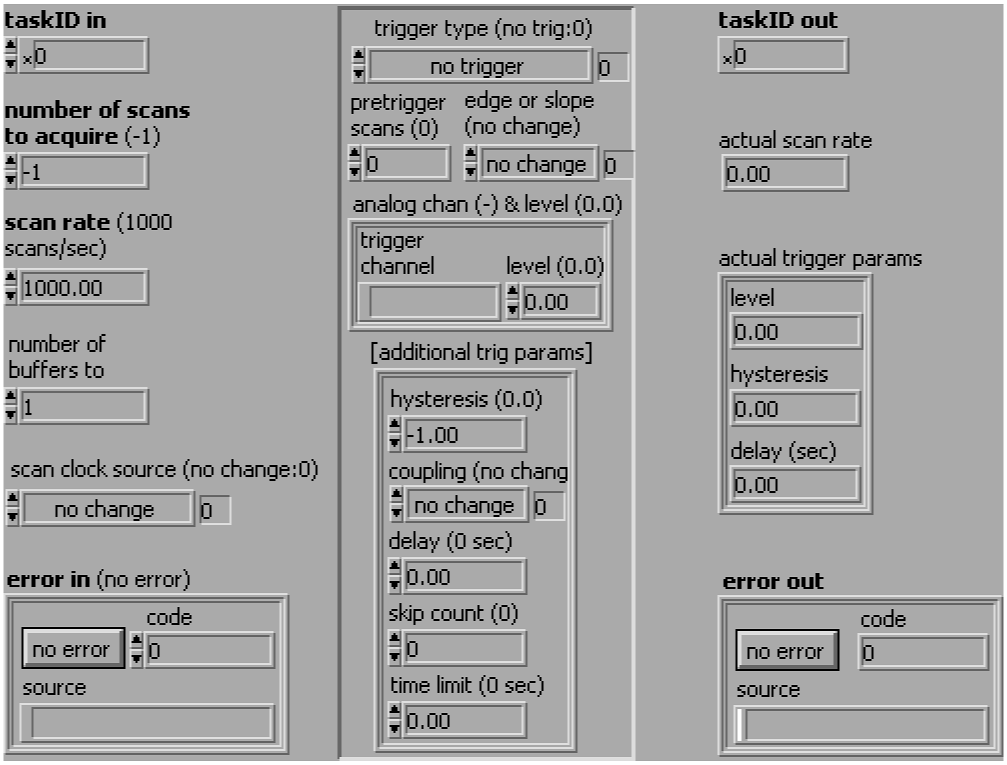
NOTE: This VI operates differently depending on whether or not software trigger is selected.
1: On iteration=0 (first call), configure group, hardware, & buffer. If using software trigger, allocate twice the buffer size. If software trigger, buffer size = MAX(# scans to read * 2, 1000 scans, 100 msec of scans).
2: Configure clock & trigger, and start acquisition on all calls. Specify read position relative to trigger if acquiring pre-trigger data with hardware trigger, enable conditional retrieval and specify read position relative to new data if using software analog trigger, otherwise read position is relative to start of data.
3: Read buffer on all calls.

AI Start.vi

Connector Pane


edge or slope (no change)
pretrigger scans (0)
trigger type (no trig:0)
taskID in
number of scans to acquire ...
scan rate (1000 scans/sec)
number of buffers to acquir...
error in (no error)
scan clock source (no chang...
analog chan (-) & level (0.0)
[additional trig params]
AI START
taskID out
actual scan rate
actual trigger params
error out

Front Panel



taskID in
0

number of scans to acquire (-1)
-1

scan rate (1000 scans/sec)
1000.00

number of buffers to
1

scan clock source (no change:0)
no change 0

error in (no error)
no error
code 0
source

trigger type (no trig:0)
no trigger 0

pretrigger scans (0)
0

edge or slope (no change)
no change 0

analog chan (-) & level (0.0)
trigger channel
level (0.0) 0.00

[additional trig params]
hysteresis (0.0) -1.00
coupling (no chang
no change 0
delay (0 sec) 0.00
skip count (0) 0
time limit (0 sec) 0.00

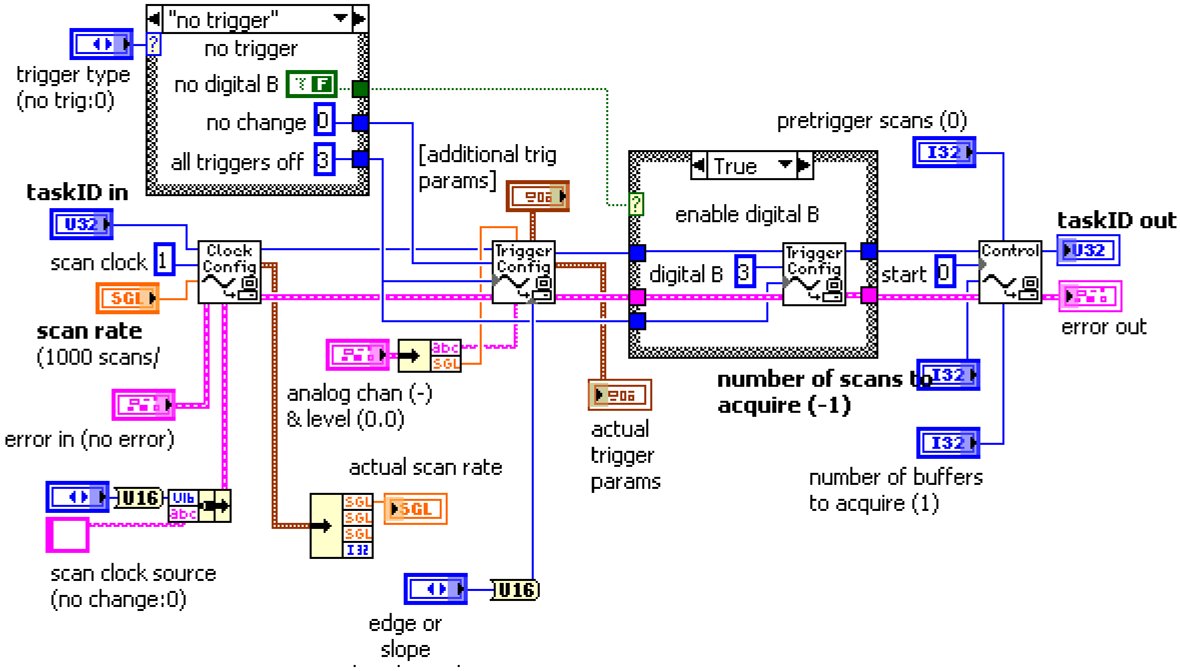
taskID out
0

actual scan rate
0.00

actual trigger params
level 0.00
hysteresis 0.00
delay (sec) 0.00

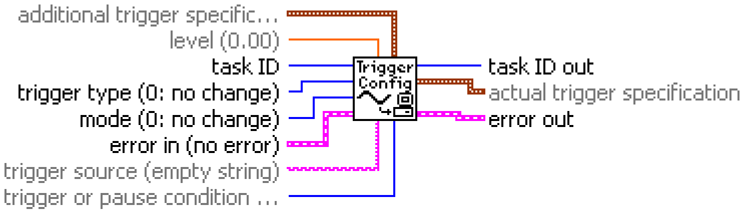
error out
no error
code 0
source

Block Diagram

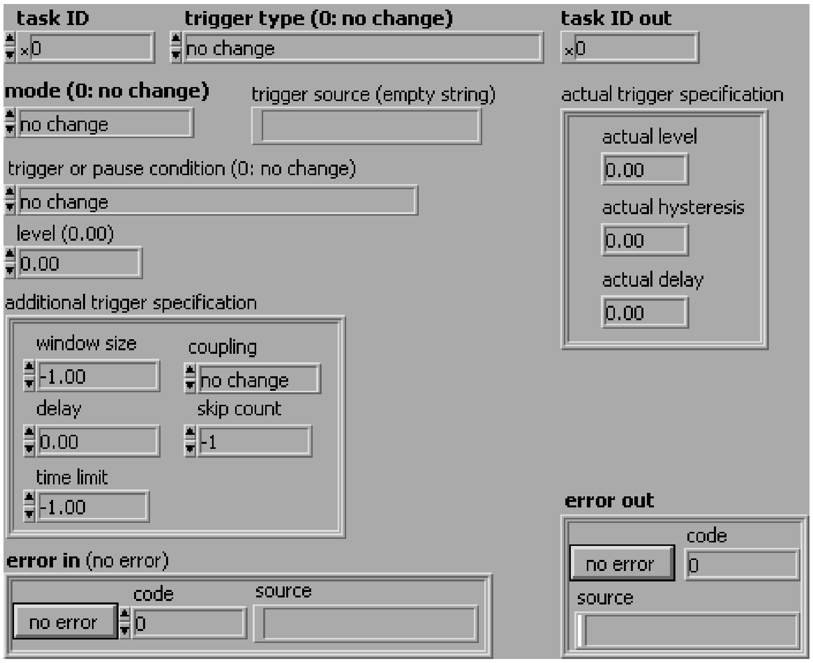


AI Trigger Config.vi

Connector Pane


additional trigger specific...
level (0.00)
task ID
trigger type (0: no change)
mode (0: no change)
error in (no error)
trigger source (empty string)
trigger or pause condition ...
Trigger Config
task ID out
actual trigger specification
error out

Front Panel



task ID: 0

trigger type (0: no change): no change

task ID out: 0

mode (0: no change): no change

trigger source (empty string):

actual trigger specification

actual level: 0.00

actual hysteresis: 0.00

actual delay: 0.00

trigger or pause condition (0: no change): no change

level (0.00): 0.00

additional trigger specification

window size: -1.00

coupling: no change

delay: 0.00

skip count: -1

time limit: -1.00

error out

no error

code: 0

source:

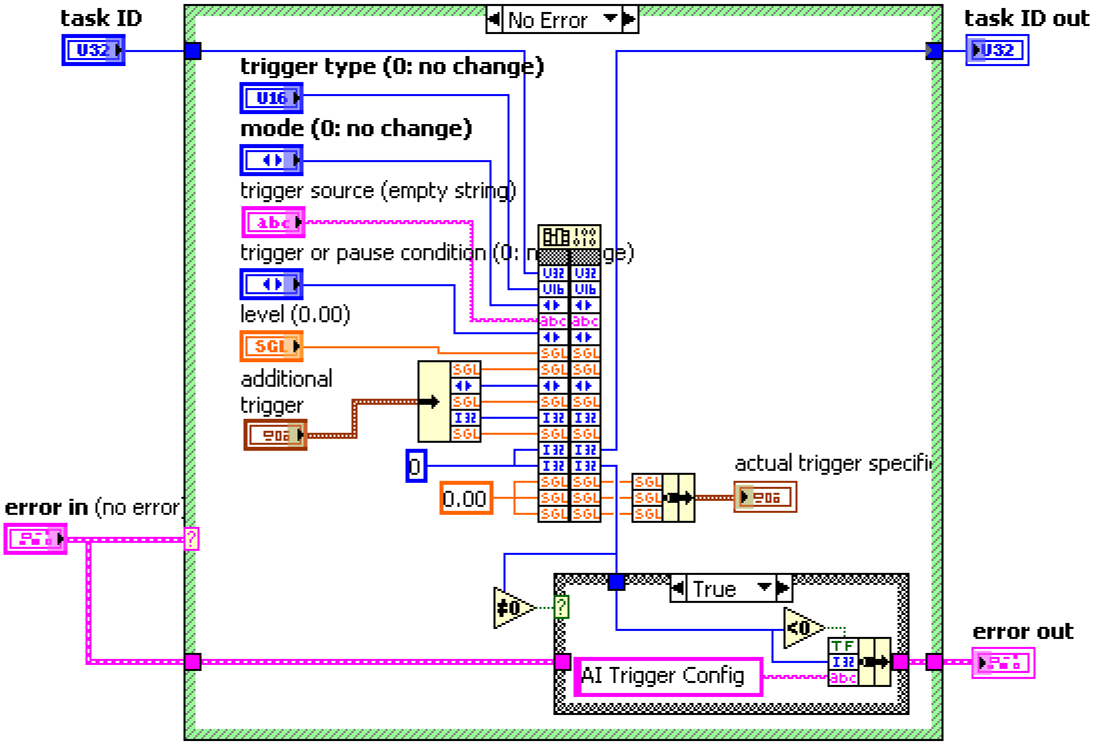
error in (no error)

no error

code: 0

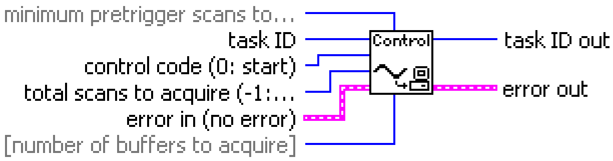
source:

Block Diagram

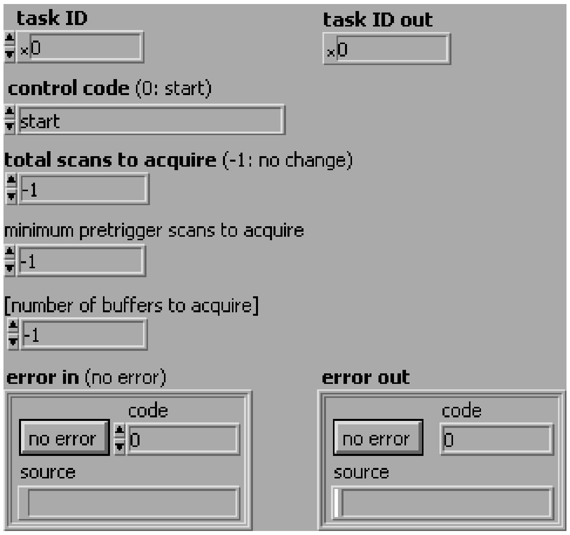


AI Control.vi

Connector Pane


minimum pretrigger scans to...
task ID
control code (0: start)
total scans to acquire (-1:...
error in (no error)
[number of buffers to acquire]
Control
task ID out
error out

Front Panel



task ID ×0

task ID out ×0

control code (0: start) start

total scans to acquire (-1: no change) -1

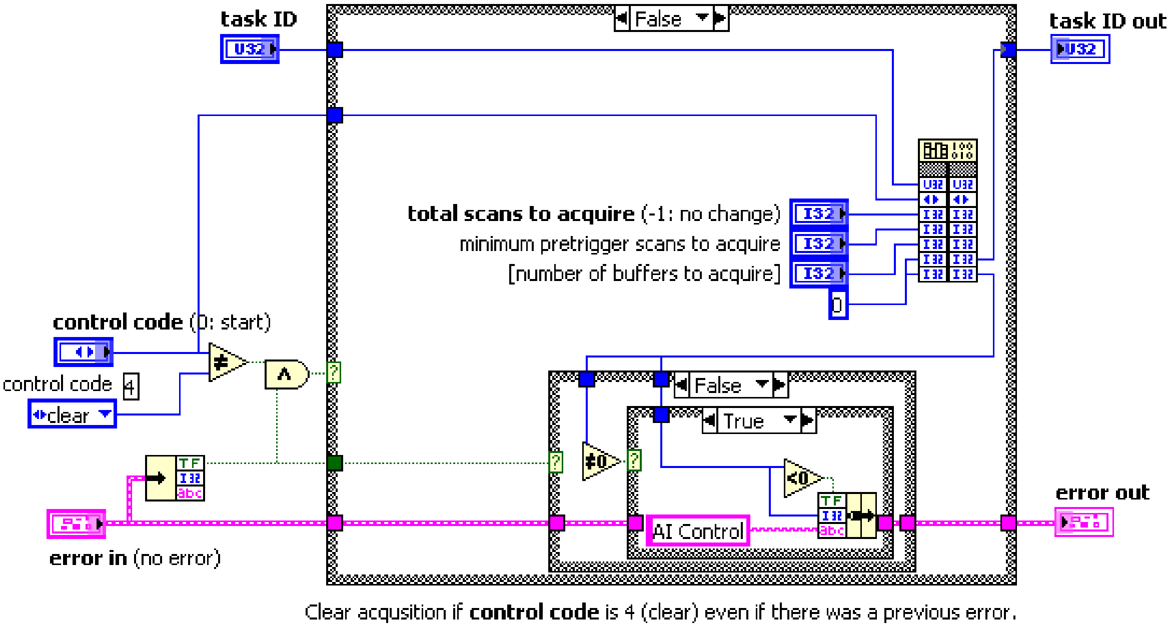
minimum pretrigger scans to acquire -1

[number of buffers to acquire] -1

error in (no error) code no error 0 source

error out code no error 0 source

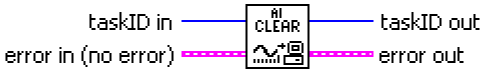
Block Diagram



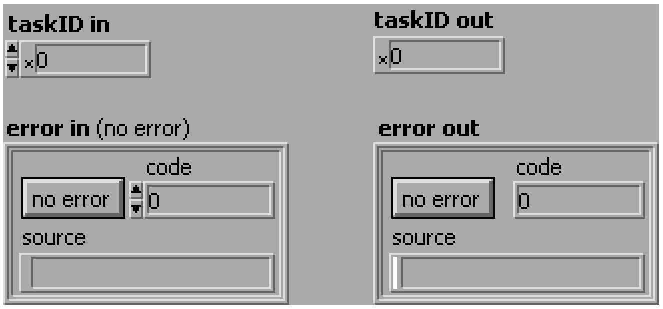
Clear acqusition if **control code** is 4 (clear) even if there was a previous error.

AI Clear.vi

Connector Pane



Front Panel

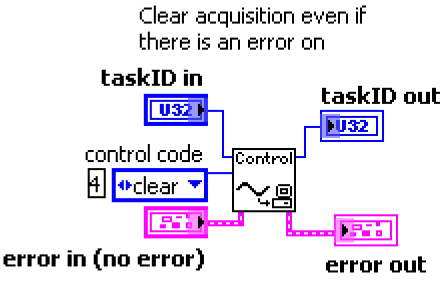

taskID in
x0

taskID out
x0

error in (no error)
code
no error
0
source

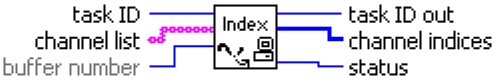
error out
code
no error
0
source

Block Diagram

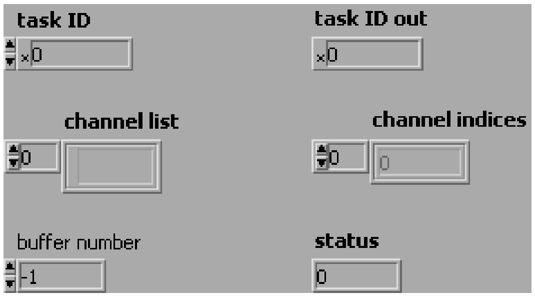


Channel To Index.vi

Connector Pane



Front Panel


task ID
x0

task ID out
x0

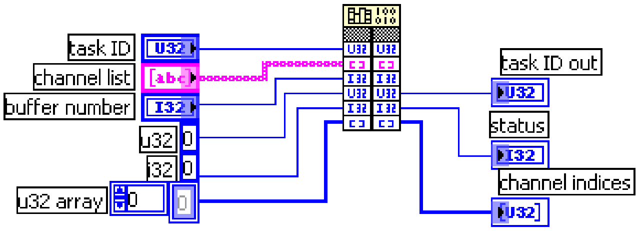
channel list
0

channel indices
0
0

buffer number
-1

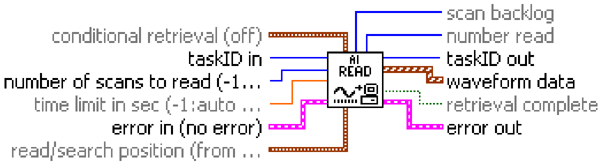
status
0

Block Diagram



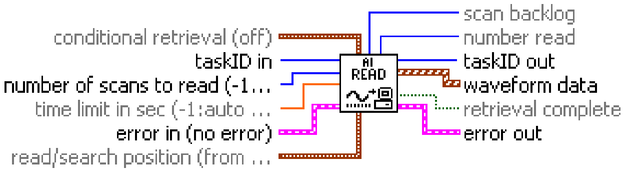
AI Read.vi

Connector Pane

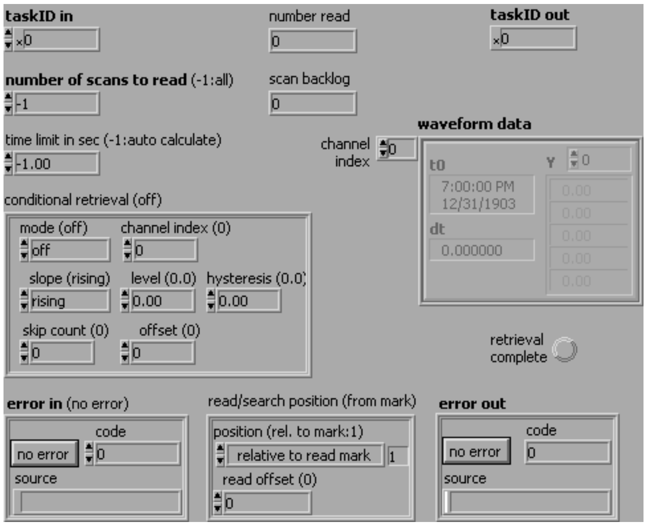

conditional retrieval (off)
taskID in
number of scans to read (-1…
time limit in sec (-1:auto …
error in (no error)
read/search position (from …
AI READ
scan backlog
number read
taskID out
waveform data
retrieval complete
error out

AI Read (waveform).vi

Connector Pane


conditional retrieval (off)
taskID in
number of scans to read (-1…
time limit in sec (-1:auto …
error in (no error)
read/search position (from …
AI READ
scan backlog
number read
taskID out
waveform data
retrieval complete
error out

Front Panel



taskID in: x0

number read: 0

taskID out: x0

number of scans to read (-1:all): -1

scan backlog: 0

time limit in sec (-1:auto calculate): -1.00

waveform data

channel index: 0

t0: 7:00:00 PM 12/31/1903

dt: 0.000000

Y: 0

0.00
0.00
0.00
0.00
0.00

conditional retrieval (off)

mode (off): off

channel index (0): 0

slope (rising): rising

level (0.0): 0.00

hysteresis (0.0): 0.00

skip count (0): 0

offset (0): 0

retrieval complete ○

error in (no error)

no error

code: 0

source:

read/search position (from mark)

position (rel. to mark:1): relative to read mark 1

read offset (0): 0

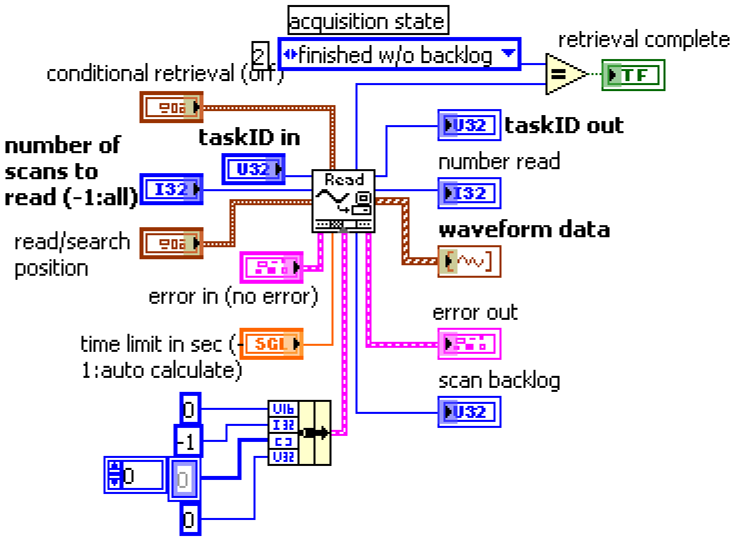
error out

no error

code: 0

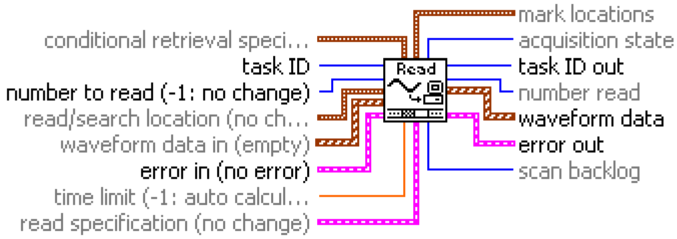
source:

Block Diagram



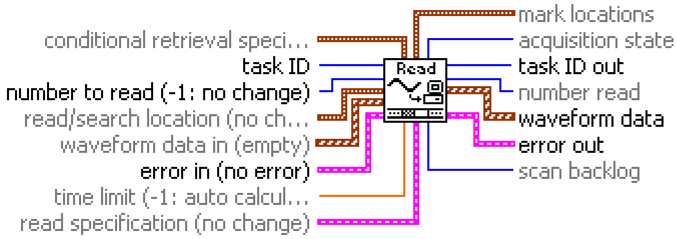
AI Buffer Read.vi

Connector Pane

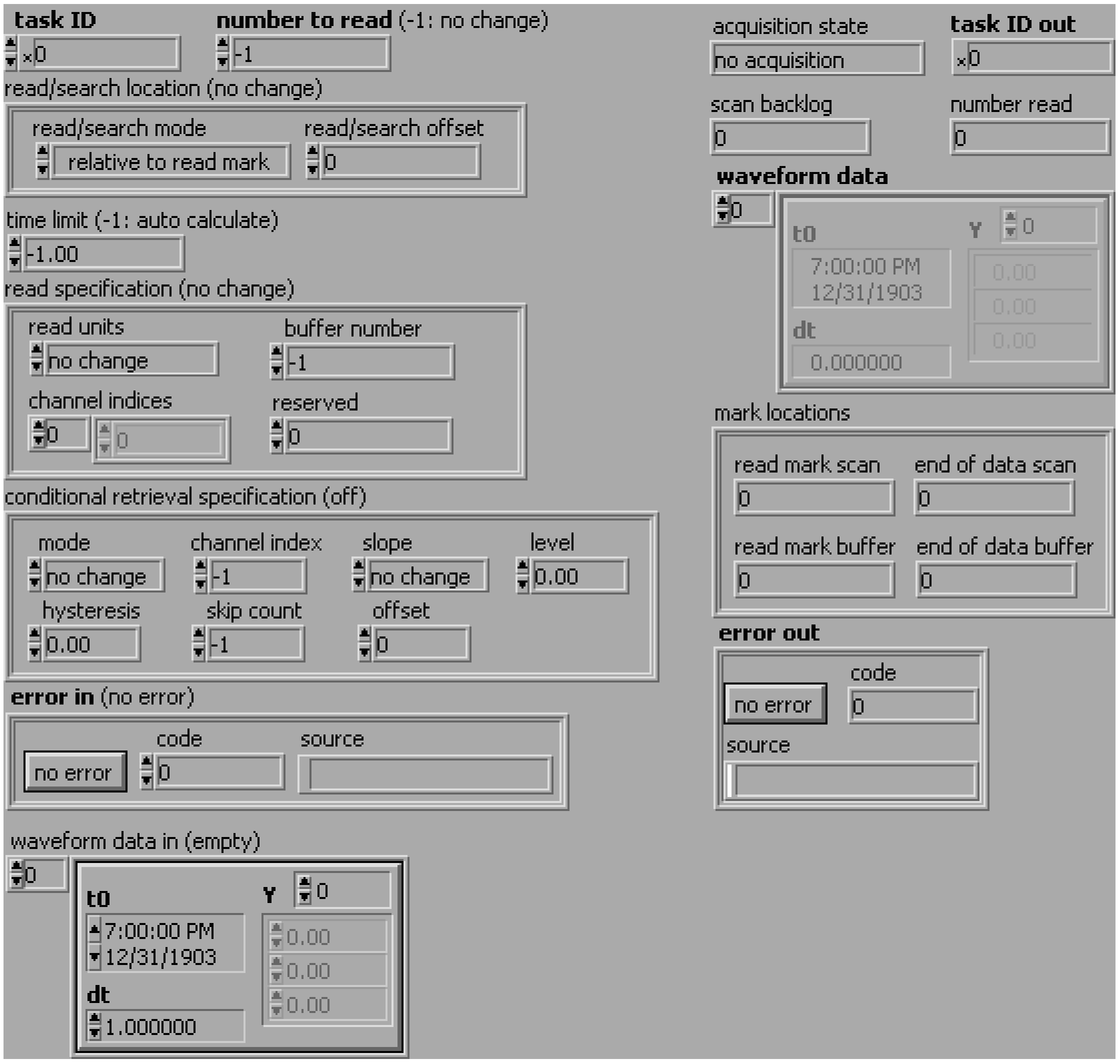

conditional retrieval speci...
task ID
number to read (-1: no change)
read/search location (no ch...
waveform data in (empty)
error in (no error)
time limit (-1: auto calcul...
read specification (no change)
Read
mark locations
acquisition state
task ID out
number read
waveform data
error out
scan backlog

AI Buffer Read (waveform).vi

Connector Pane


conditional retrieval speci...
task ID
number to read (-1: no change)
read/search location (no ch...
waveform data in (empty)
error in (no error)
time limit (-1: auto calcul...
read specification (no change)
Read
mark locations
acquisition state
task ID out
number read
waveform data
error out
scan backlog

Front Panel



task ID 0

number to read (-1: no change) -1

read/search location (no change)
- read/search mode: relative to read mark
- read/search offset: 0

time limit (-1: auto calculate) -1.00

read specification (no change)
- read units: no change
- buffer number: -1
- channel indices: 0 | 0
- reserved: 0

conditional retrieval specification (off)
- mode: no change
- channel index: -1
- slope: no change
- level: 0.00
- hysteresis: 0.00
- skip count: -1
- offset: 0

error in (no error)
- no error
- code: 0
- source:

waveform data in (empty)
- 0
- **t0**: 7:00:00 PM 12/31/1903
- **dt**: 1.000000
- **Y**: 0
- 0.00
- 0.00
- 0.00

acquisition state: no acquisition

task ID out 0

scan backlog: 0

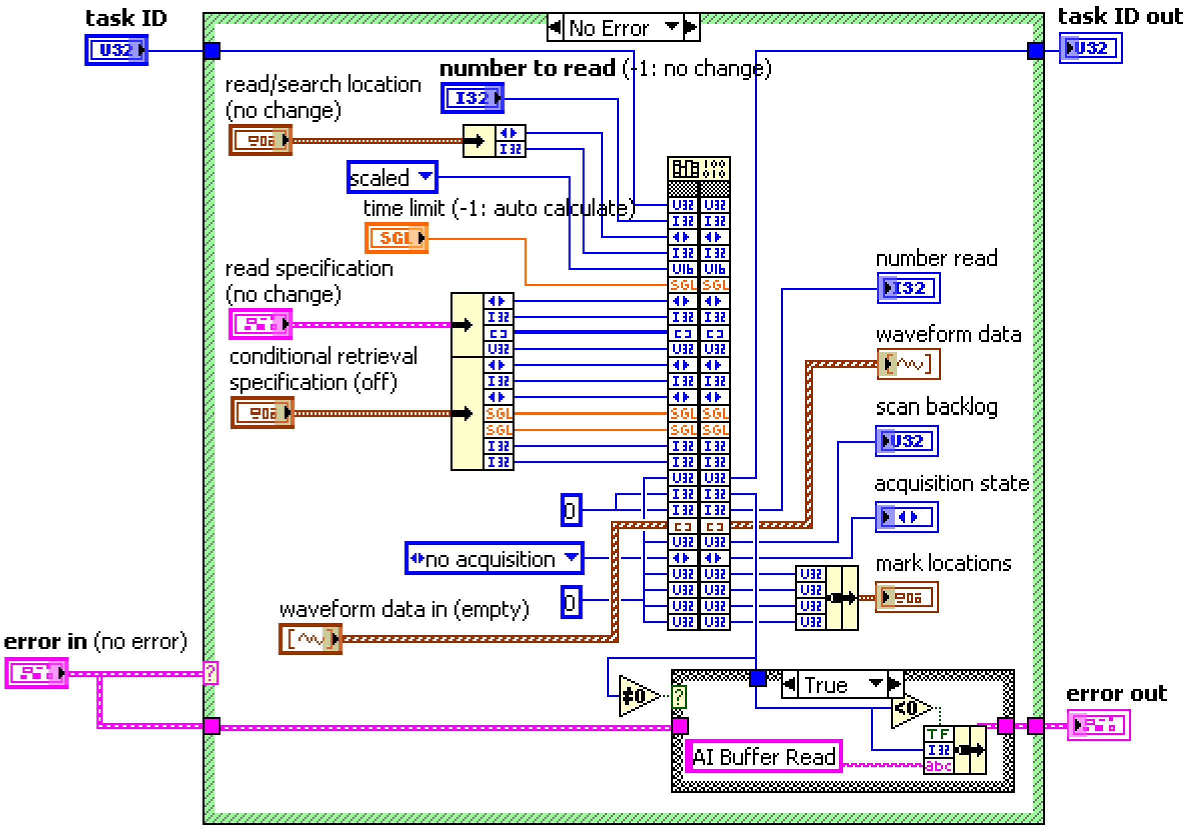
number read: 0

waveform data
- 0
- **t0**: 7:00:00 PM 12/31/1903
- **dt**: 0.000000
- **Y**: 0
- 0.00
- 0.00
- 0.00

mark locations
- read mark scan: 0
- end of data scan: 0
- read mark buffer: 0
- end of data buffer: 0

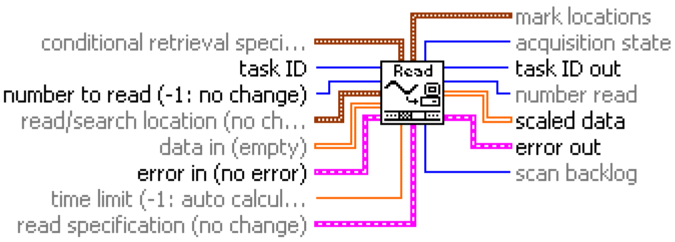
error out
- no error
- code: 0
- source:

Block Diagram

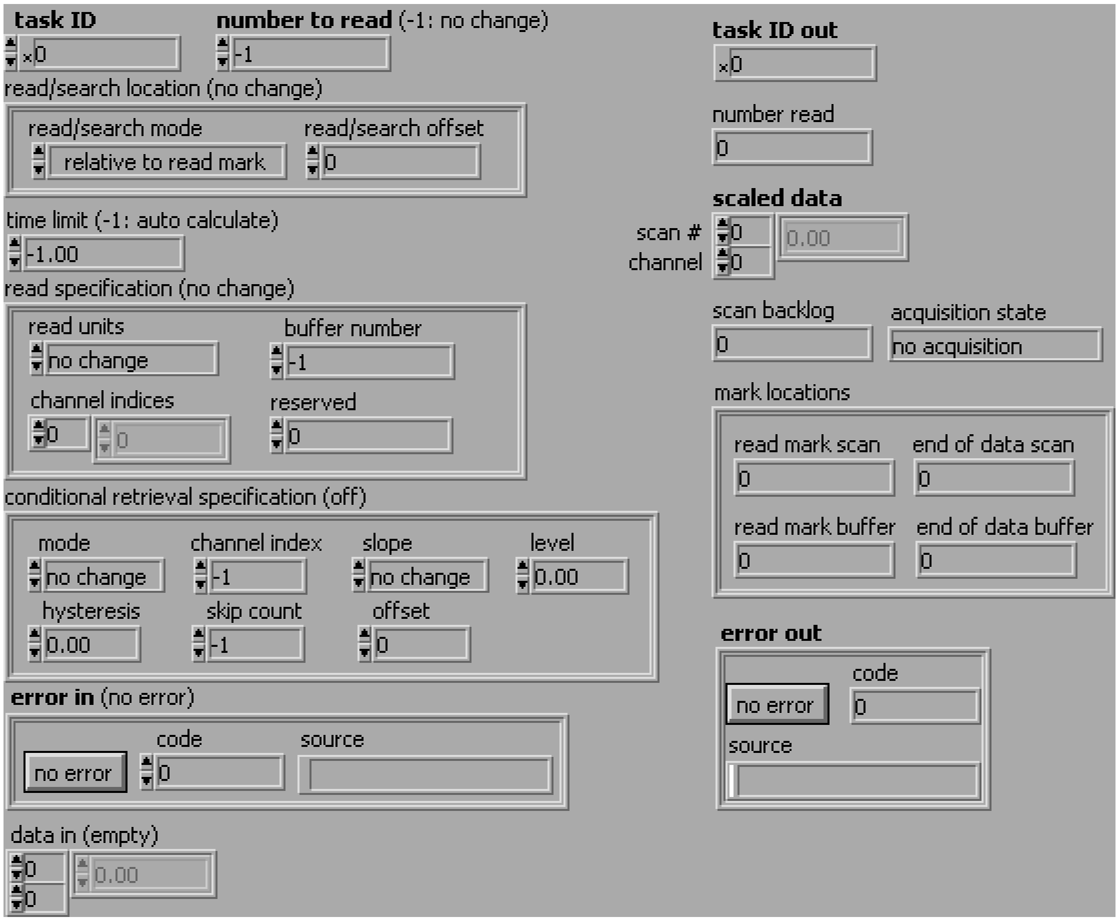


AI Buffer Read (scaled array).vi

Connector Pane


conditional retrieval speci...
task ID
number to read (-1: no change)
read/search location (no ch...
data in (empty)
error in (no error)
time limit (-1: auto calcul...
read specification (no change)
Read
mark locations
acquisition state
task ID out
number read
scaled data
error out
scan backlog

Front Panel



task ID x0

number to read (-1: no change) -1

read/search location (no change)

read/search mode: relative to read mark

read/search offset: 0

time limit (-1: auto calculate) -1.00

read specification (no change)

read units: no change

buffer number: -1

channel indices: 0 0

reserved: 0

conditional retrieval specification (off)

mode: no change

channel index: -1

slope: no change

level: 0.00

hysteresis: 0.00

skip count: -1

offset: 0

error in (no error)

no error

code: 0

source:

data in (empty) 0 0 0.00

task ID out x0

number read: 0

scaled data

scan # 0

channel 0

0.00

scan backlog: 0

acquisition state: no acquisition

mark locations

read mark scan: 0

end of data scan: 0

read mark buffer: 0

end of data buffer: 0

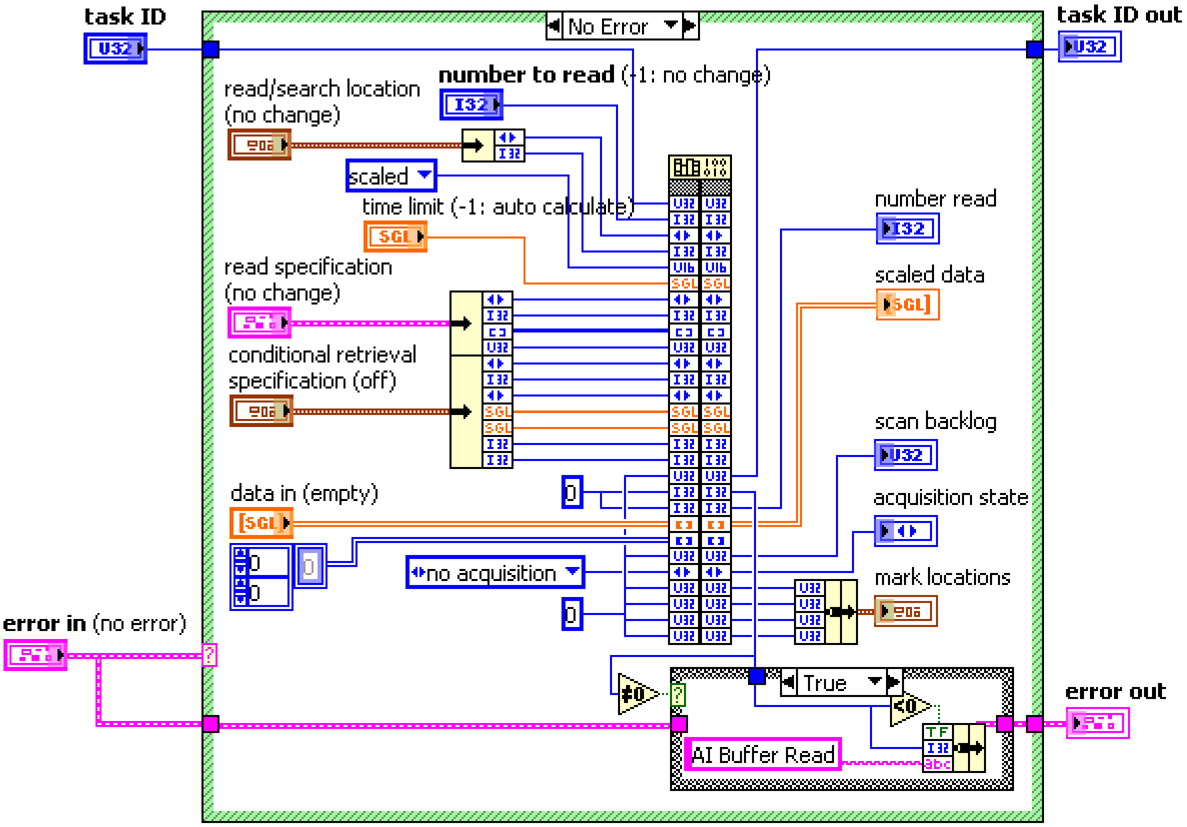
error out

no error

code: 0

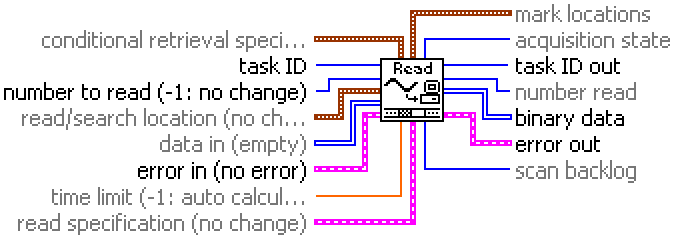
source:

Block Diagram

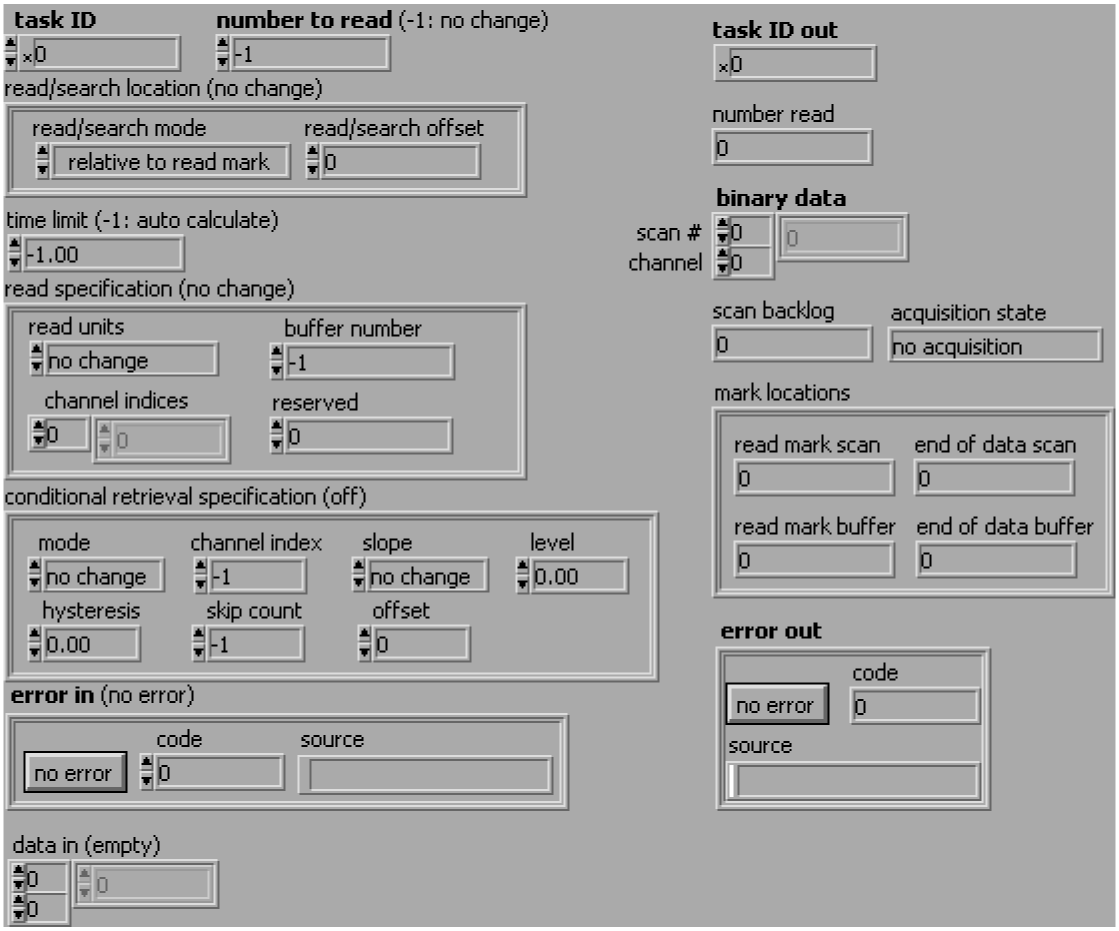


AI Buffer Read (binary array).vi

Connector Pane


conditional retrieval speci…
task ID
number to read (-1: no change)
read/search location (no ch…
data in (empty)
error in (no error)
time limit (-1: auto calcul…
read specification (no change)
Read
mark locations
acquisition state
task ID out
number read
binary data
error out
scan backlog

Front Panel



task ID x0

number to read (-1: no change) -1

read/search location (no change)

read/search mode: relative to read mark

read/search offset: 0

time limit (-1: auto calculate) -1.00

read specification (no change)

read units: no change

buffer number: -1

channel indices: 0 0

reserved: 0

conditional retrieval specification (off)

mode: no change

channel index: -1

slope: no change

level: 0.00

hysteresis: 0.00

skip count: -1

offset: 0

error in (no error)

no error

code: 0

source:

data in (empty) 0 0 0

task ID out x0

number read: 0

binary data

scan # 0

channel 0

0

scan backlog: 0

acquisition state: no acquisition

mark locations

read mark scan: 0

end of data scan: 0

read mark buffer: 0

end of data buffer: 0

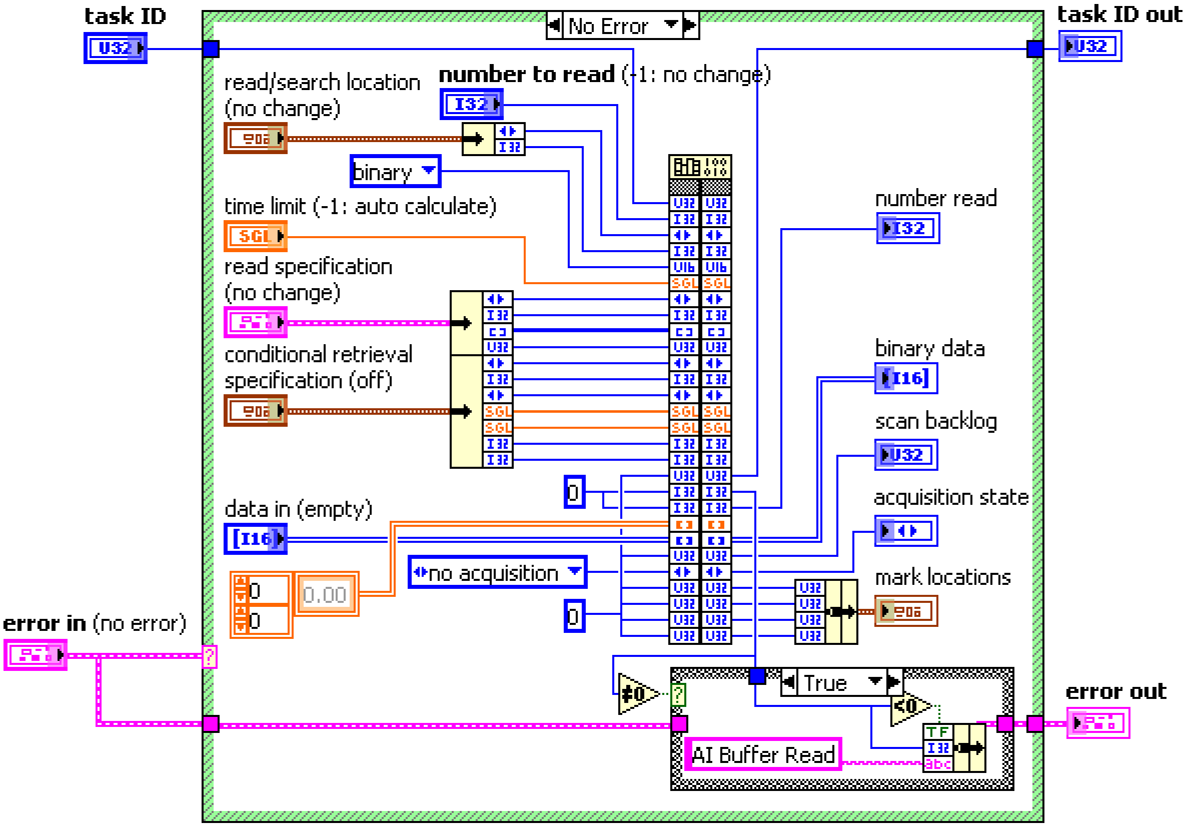
error out

no error

code: 0

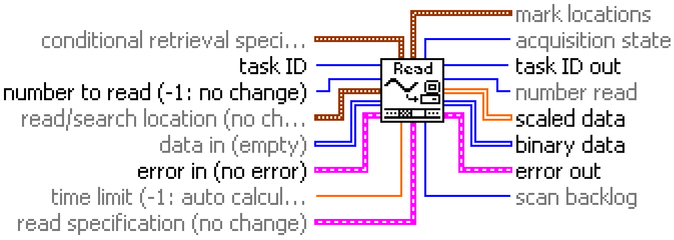
source:

Block Diagram

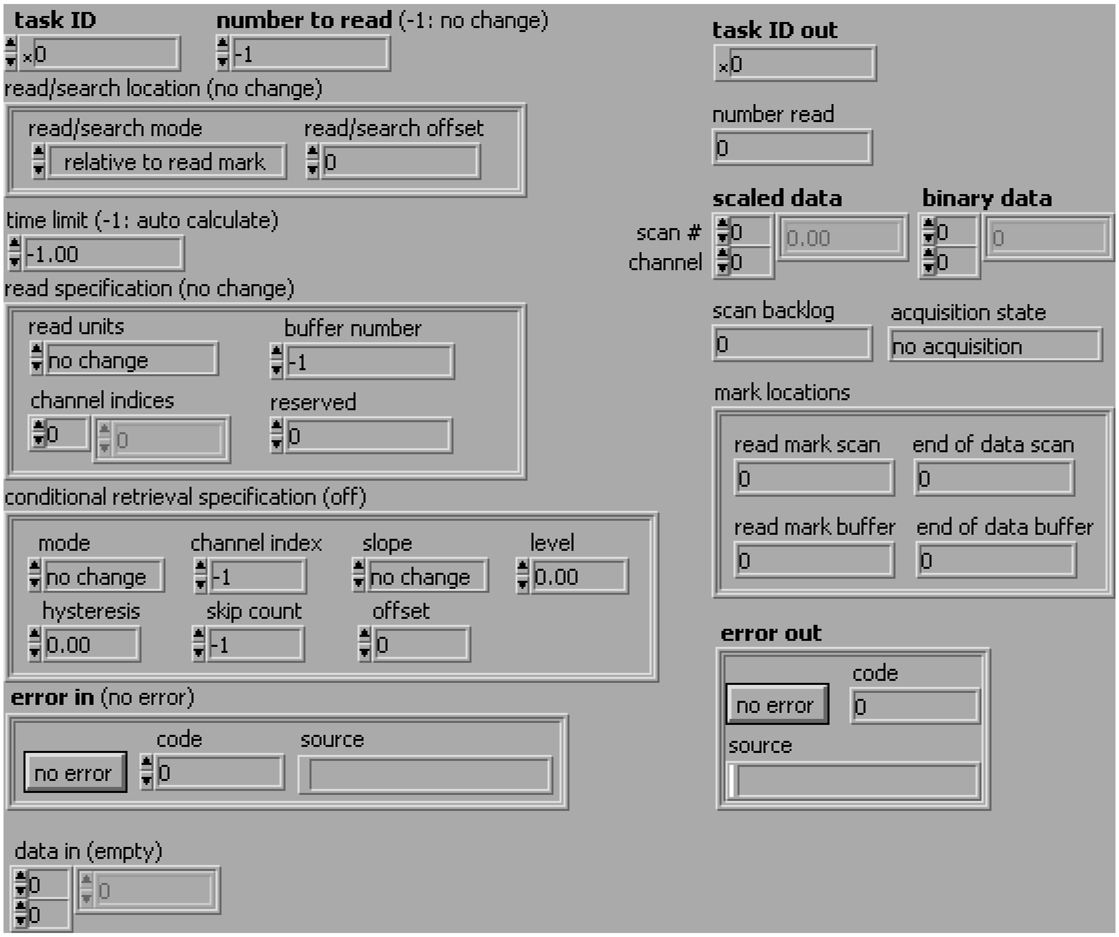


AI Buffer Read (scaled & binary arrays).vi

Connector Pane


conditional retrieval speci...
task ID
number to read (-1: no change)
read/search location (no ch...
data in (empty)
error in (no error)
time limit (-1: auto calcul...
read specification (no change)
Read
mark locations
acquisition state
task ID out
number read
scaled data
binary data
error out
scan backlog

Front Panel



task ID ×0

number to read (-1: no change) -1

read/search location (no change)

read/search mode: relative to read mark

read/search offset: 0

time limit (-1: auto calculate) -1.00

read specification (no change)

read units: no change

buffer number: -1

channel indices: 0 0

reserved: 0

conditional retrieval specification (off)

mode: no change

channel index: -1

slope: no change

level: 0.00

hysteresis: 0.00

skip count: -1

offset: 0

error in (no error)

no error

code: 0

source:

data in (empty) 0 0 0

task ID out ×0

number read: 0

scaled data scan # 0 channel 0 0.00

binary data 0 0 0

scan backlog: 0

acquisition state: no acquisition

mark locations

read mark scan: 0

end of data scan: 0

read mark buffer: 0

end of data buffer: 0

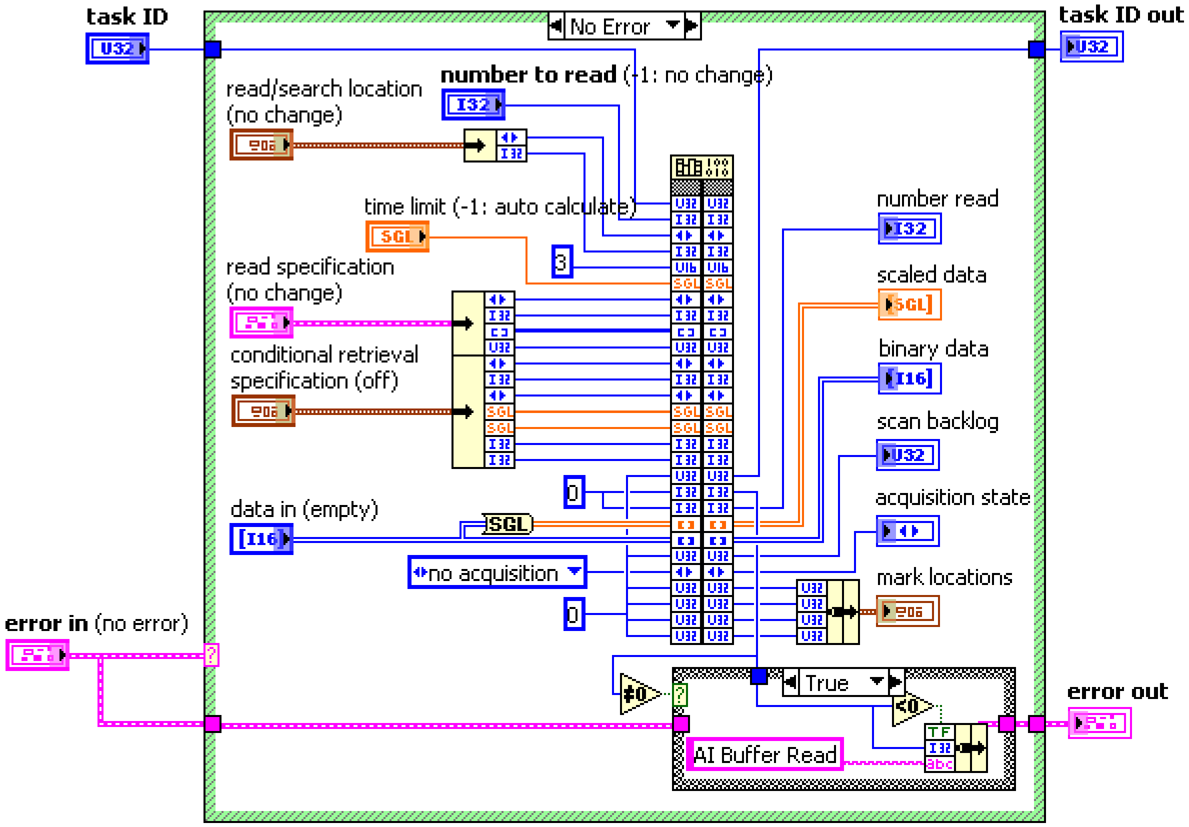
error out

no error

code: 0

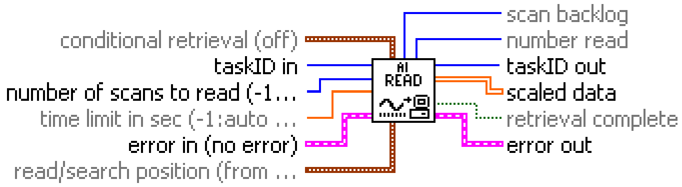
source:

Block Diagram

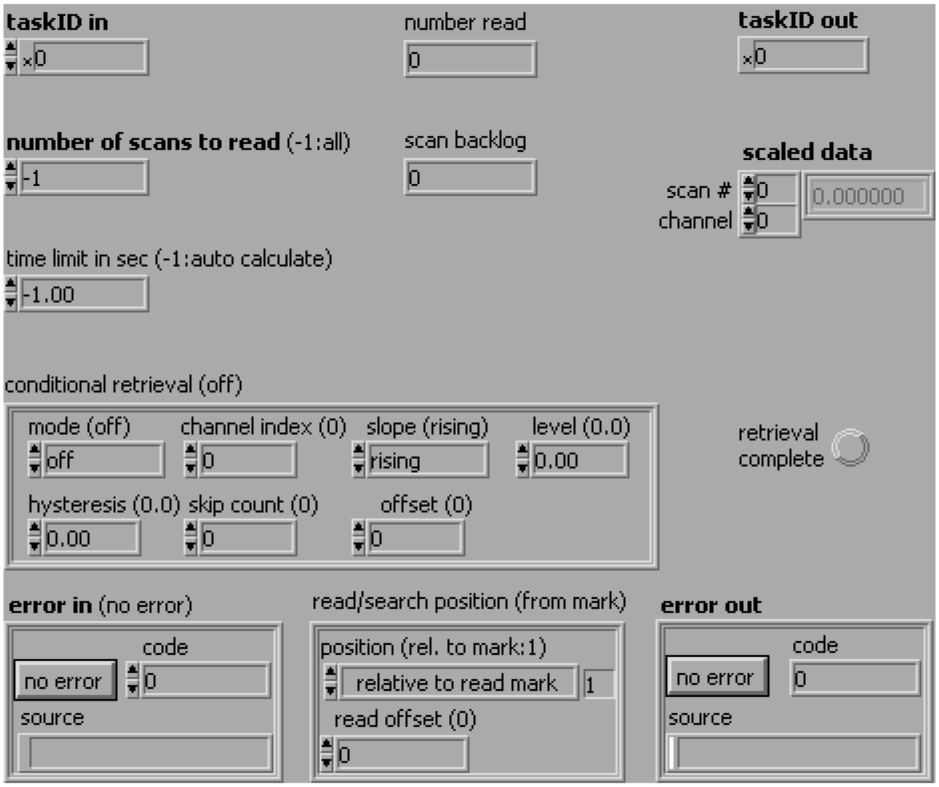


AI Read (scaled array).vi

Connector Pane


conditional retrieval (off)
taskID in
number of scans to read (-1...
time limit in sec (-1:auto ...
error in (no error)
read/search position (from ...
AI READ
scan backlog
number read
taskID out
scaled data
retrieval complete
error out

Front Panel



taskID in: 0

number read: 0

taskID out: 0

number of scans to read (-1:all): -1

scan backlog: 0

scaled data
scan # 0
channel 0
0.000000

time limit in sec (-1:auto calculate): -1.00

conditional retrieval (off)
mode (off): off
channel index (0): 0
slope (rising): rising
level (0.0): 0.00
hysteresis (0.0): 0.00
skip count (0): 0
offset (0): 0

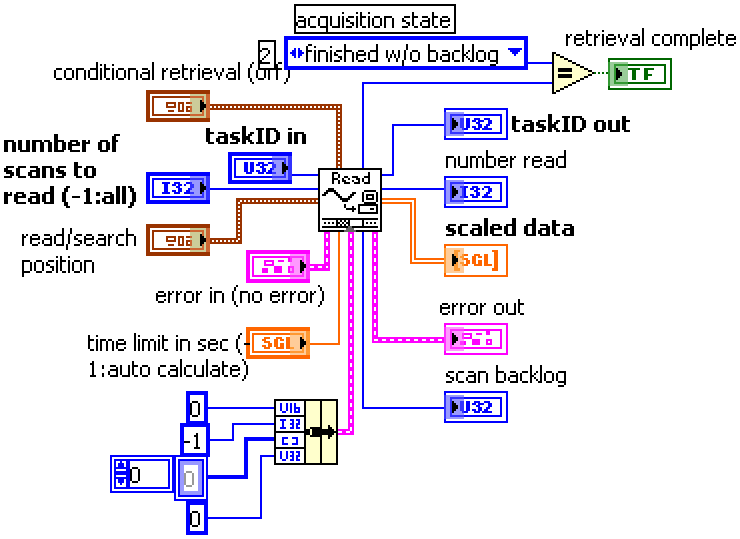
retrieval complete ○

error in (no error)
code: no error 0
source:

read/search position (from mark)
position (rel. to mark:1): relative to mark 1
read offset (0): 0

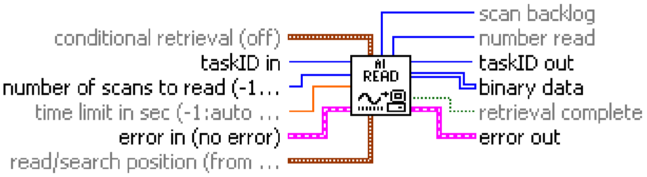
error out
code: no error 0
source:

Block Diagram



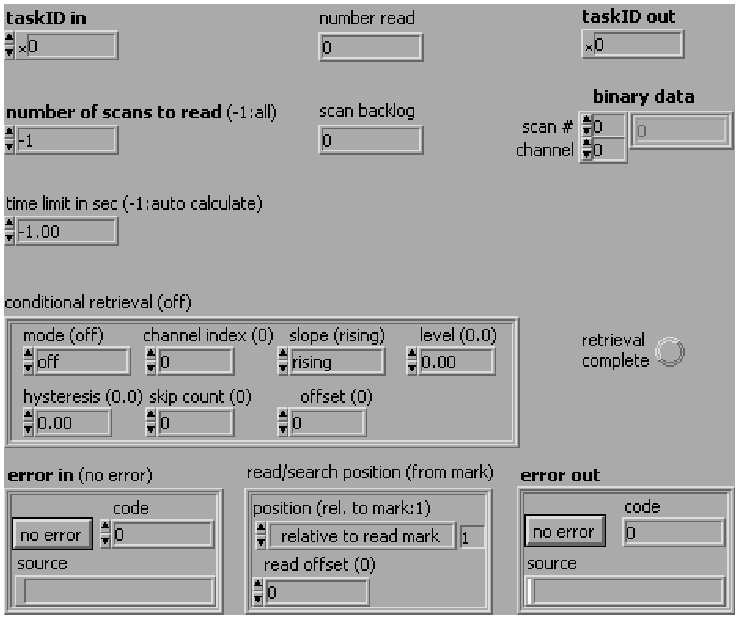
AI Read (binary array).vi

Connector Pane



conditional retrieval (off)
taskID in
number of scans to read (-1…
time limit in sec (-1:auto …
error in (no error)
read/search position (from …
AI READ
scan backlog
number read
taskID out
binary data
retrieval complete
error out

Front Panel



taskID in
x0

number read
0

taskID out
x0

number of scans to read (-1:all)
-1

scan backlog
0

binary data
scan # 0
channel 0
0

time limit in sec (-1:auto calculate)
-1.00

conditional retrieval (off)
mode (off) off
channel index (0) 0
slope (rising) rising
level (0.0) 0.00
hysteresis (0.0) 0.00
skip count (0) 0
offset (0) 0

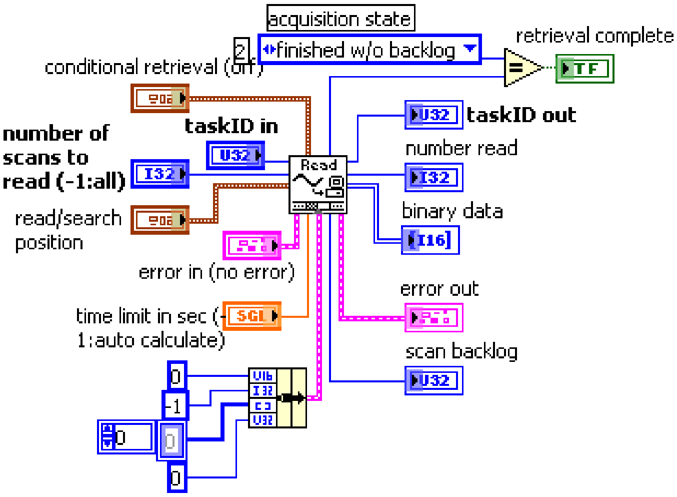
retrieval complete ○

error in (no error)
no error
code 0
source

read/search position (from mark)
position (rel. to mark:1) relative to read mark 1
read offset (0) 0

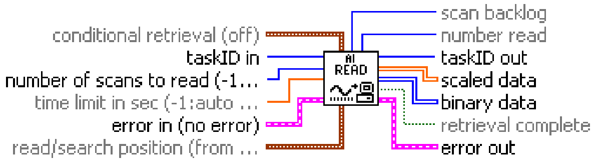
error out
no error
code 0
source

Block Diagram

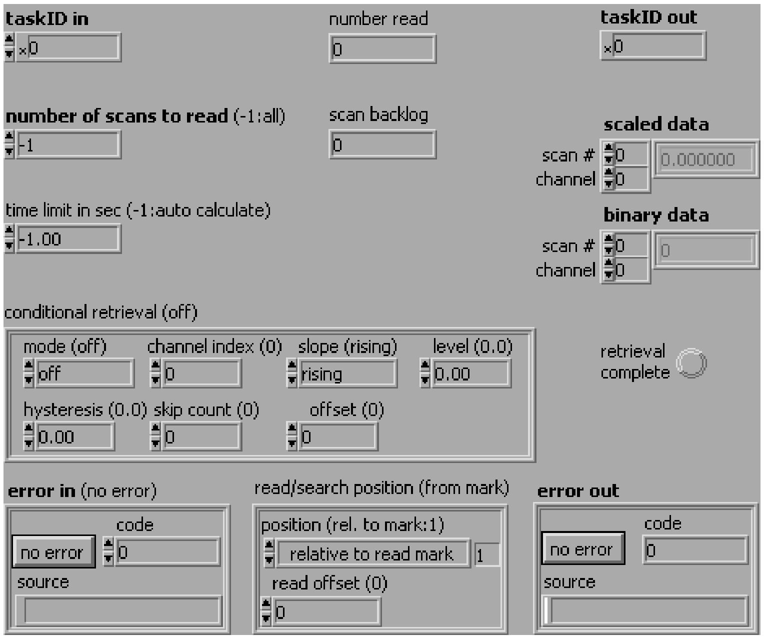


AI Read (scaled & binary arrays).vi

Connector Pane



Front Panel


taskID in: 0

number read: 0

taskID out: 0

number of scans to read (-1:all): -1

scan backlog: 0

scaled data: scan # 0, channel 0, 0.000000

binary data: scan # 0, channel 0, 0

time limit in sec (-1:auto calculate): -1.00

conditional retrieval (off)

mode (off): off
channel index (0): 0
slope (rising): rising
level (0.0): 0.00
hysteresis (0.0): 0.00
skip count (0): 0
offset (0): 0

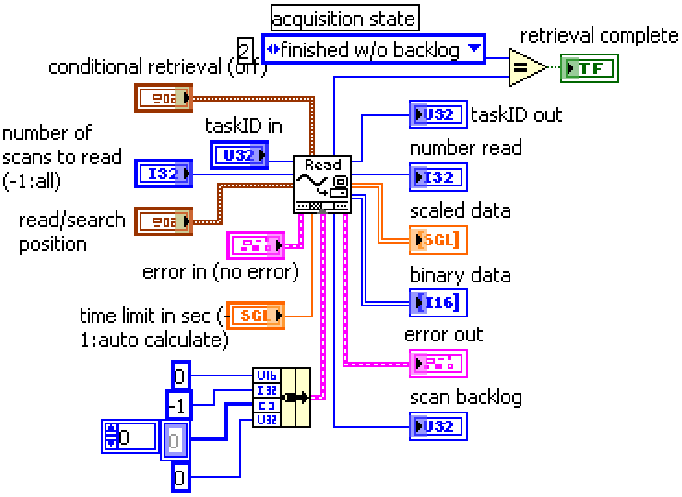
retrieval complete

error in (no error): no error, code 0, source

read/search position (from mark): position (rel. to mark:1): relative to read mark 1, read offset (0): 0

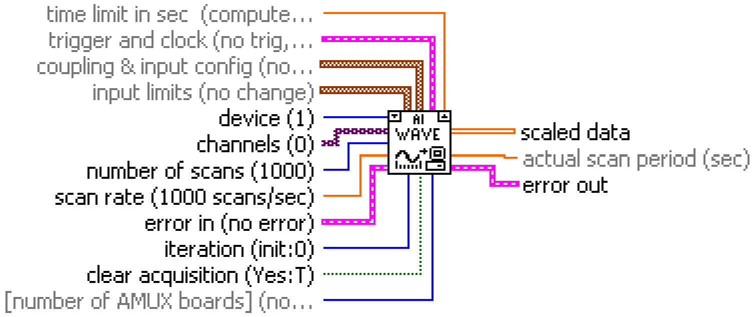
error out: no error, code 0, source

Block Diagram

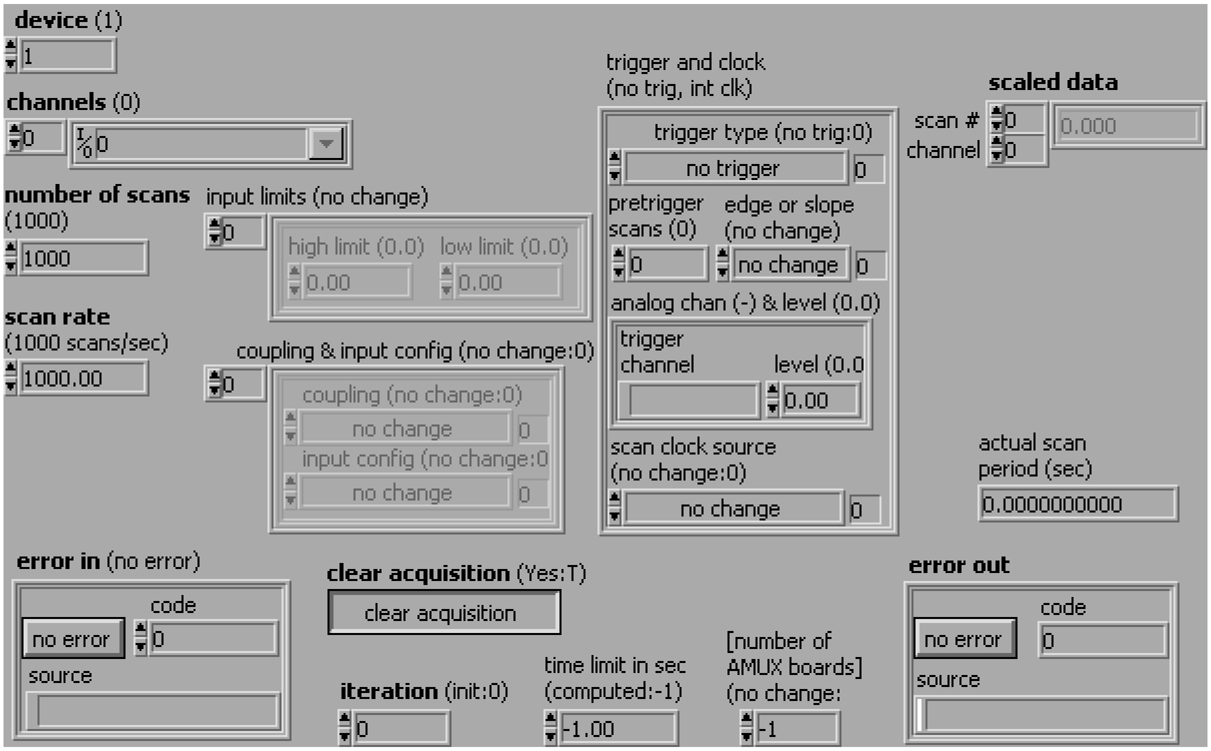


AI Waveform Scan (scaled array).vi

Connector Pane


time limit in sec (compute...
trigger and clock (no trig,...
coupling & input config (no...
input limits (no change)
device (1)
channels (0)
number of scans (1000)
scan rate (1000 scans/sec)
error in (no error)
iteration (init:0)
clear acquisition (Yes:T)
[number of AMUX boards] (no...
AI WAVE
scaled data
actual scan period (sec)
error out

Front Panel


device (1)
1

channels (0)
0 | 0

number of scans (1000)
1000

scan rate (1000 scans/sec)
1000.00

input limits (no change)
0
high limit (0.0) 0.00
low limit (0.0) 0.00

coupling & input config (no change:0)
0
coupling (no change:0) no change 0
input config (no change:0 no change 0

trigger and clock (no trig, int clk)
trigger type (no trig:0) no trigger 0
pretrigger scans (0) 0
edge or slope (no change) no change 0
analog chan (-) & level (0.0)
trigger channel
level (0.0 0.00
scan clock source (no change:0) no change 0

scaled data
scan # 0
channel 0
0.000

actual scan period (sec)
0.0000000000

error in (no error)
no error
code 0
source

clear acquisition (Yes:T)
clear acquisition

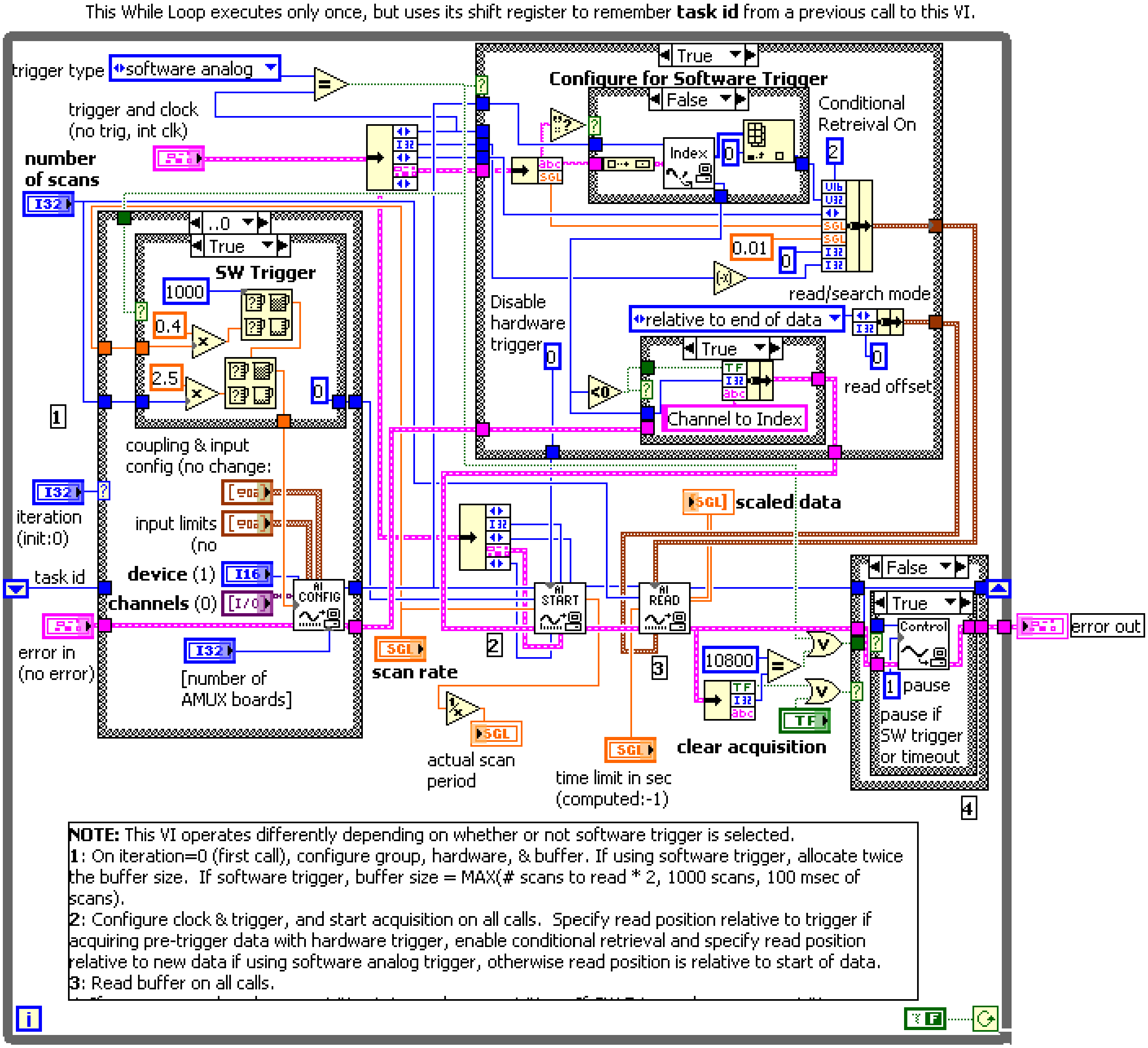
iteration (init:0)
0

time limit in sec (computed:-1)
-1.00

[number of AMUX boards] (no change:
-1

error out
no error
code 0
source

Block Diagram

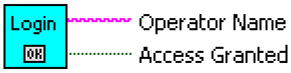


This While Loop executes only once, but uses its shift register to remember **task id** from a previous call to this VI.

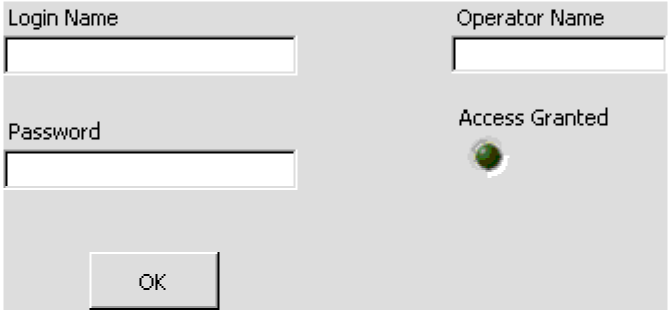
NOTE: This VI operates differently depending on whether or not software trigger is selected.
1: On iteration=0 (first call), configure group, hardware, & buffer. If using software trigger, allocate twice the buffer size. If software trigger, buffer size = MAX(# scans to read * 2, 1000 scans, 100 msec of scans).
2: Configure clock & trigger, and start acquisition on all calls. Specify read position relative to trigger if acquiring pre-trigger data with hardware trigger, enable conditional retrieval and specify read position relative to new data if using software analog trigger, otherwise read position is relative to start of data.
3: Read buffer on all calls.

Login.vi

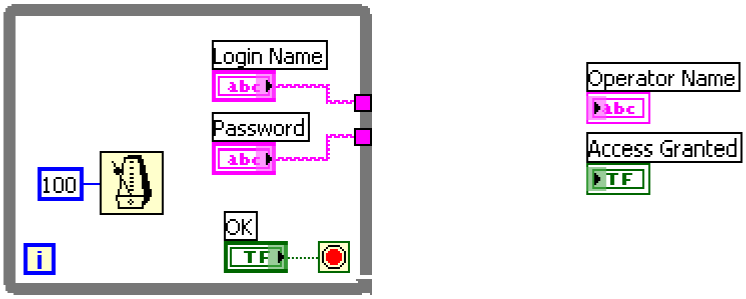
Connector Pane



Front Panel



Block Diagram



REFERENCES

1. http://www.ni.com
2. http://www.ni.com/labview/
3. http://www.ni.com/dataacquisition
4. Wells L. K., Travis J., LabVIEW for everyone, Graphical programming made more easy. Prentice Hall, 1997.
5. Craig Smith and Jason King, A Powerful New Tool for UI Programming User Interface Event Programming LabVIEW Technical Resource, Volume 9 No. 3
6. http://www.gouldis.com
7. Feedback Instruments Limited, Process trainer, PT326.
8. Kostic M., Data acquisition and control for an innovative thermal conductivity apparatus using LabVIEW/sup r/ virtual instrument. Laboratory robotics and automation, 10(2) 107-111
9. Jia, Huiping.; Real – time experiments with LabVIEW over Internet, 2001
10. Travis J., Internet Application of LabVIEW. Prentice Hall, 1999
11. Filippetti F., A labVIEW based virtual instrument for on line induction motor parameters identification. IEEE power engineering review, 18(6) 47-48
12. Beyon, Jeffrey Y., LabVIEW programming Data acquisition and analysis, 2001.
13. Bitter, Rick.; LabVIEW advanced programming techniques, 2001
14. Brenner E., Application of LabVIEW as measuring system in field power electronics. Proceedings of the 7th European conference on power electronics and applications. 4:984-989