

GXSW

Switching Instruments

User's Guide

Last Updated December 16, 2016



Safety and Handling

Each product shipped by Marvin Test Solutions is carefully inspected and tested prior to shipping. The shipping box provides protection during shipment, and can be used for storage of both the hardware and the software when they are not in use.

The circuit boards are extremely delicate and require care in handling and installation. Do not remove the boards from their protective plastic coverings or from the shipping box until you are ready to install the boards into your computer.

If a board is removed from the computer for any reason, be sure to store it in its original shipping box. Do not store boards on top of workbenches or other areas where they might be susceptible to damage or exposure to strong electromagnetic or electrostatic fields. Store circuit boards in protective anti-electrostatic wrapping and away from electromagnetic fields.

Be sure to make a single copy of the software CD for installation. Store the original CD in a safe place away from electromagnetic or electrostatic fields. Return compact disks (CD) to their protective case or sleeve and store in the original shipping box or other suitable location.

Warranty

Marvin Test Solutions products are warranted against defects in materials and workmanship for a period of 12 months. Marvin Test Solutions shall repair or replace (at its discretion) any defective product during the stated warranty period. The software warranty includes any revisions or new versions released during the warranty period. Revisions and new versions may be covered by a software support agreement. If you need to return a board, please contact Marvin Test Solutions Customer Technical Services Department via <https://www.MarvinTest.com/magic/> - the Marvin Test Solutions on-line support system.

If You Need Help

Visit our web site at <https://www.MarvinTest.com> for more information about Marvin Test Solutions products, services and support options. Our web site contains sections describing support options and application notes, as well as a download area for downloading patches, example, patches and new or revised instrument drivers. To submit a support issue including suggestion, bug report or questions please use the following link: <https://www.MarvinTest.com/magic/>

You can also use Marvin Test Solutions technical support phone line (949) 263-2222. This service is available between 8:30 AM and 5:30 PM Pacific Standard Time.

Our address (check our website for latest address): Marvin Test Solutions, 1770 Kettering, Irvine, CA 92614, USA.

Disclaimer

In no event, shall Marvin Test Solutions or any of its representatives be liable for any consequential damages whatsoever (including unlimited damages for loss of business profits, business interruption, loss of business information, or any other losses) arising out of the use of or inability to use this product, even if Marvin Test Solutions has been advised of the possibility for such damages.

Copyright

Copyright © 2011-2016 by Marvin Test Solutions, Inc. All rights reserved. No part of this document can be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Marvin Test Solutions.

Trademarks

ATEasy®, CalEasy, DIOEasy®, DtifEasy, WaveEasy	Marvin Test Solutions (prior name is Geotest - Marvin Test Systems Inc.)
C++ Builder, Delphi	Embarcadero Technologies Inc.
LabView, LabWindows™/CVI	National Instruments
Microsoft Developer Studio, Microsoft Visual C++, Microsoft Visual Basic, .NET and Windows	Microsoft Corporation

All other trademarks are the property of their respective owners.

Table of Contents

Safety and Handling.....	i
Warranty	i
If You Need Help.....	i
Disclaimer	i
Copyright	i
Trademarks	ii
Chapter 1 - Introduction	5
Manual Scope and Organization	5
Manual Scope.....	5
Manual Organization.....	5
Conventions Used in this Manual	5
Supported Switching instruments	6
List of Acronyms	7
Chapter 2 - Introduction to Switching.....	9
Test Systems	9
The Basic Test System	9
Reasons to Automate Test Systems	11
Designing Automatic Test Systems	11
Basic Switch Topology	12
Multiplex or Scan Topology	12
Switch Matrix Topology	13
Switching Instrument Criteria	14
Understanding Switching Components.....	15
Switching Types.....	15
Switching Elements.....	16
Types of Relays.....	17
Electromechanical Relays	17
Dry Reed Relays	18
Maximum Relay Voltage	18
Maximum Relay Current.....	19
Maximum Relay Power.....	19
Contact Potential	19
Operating Speed.....	20
Contact Resistance versus Life	20
Relay Coil Power	21

Insulation Resistance.....	22
Relay Capacitance.....	22
Signal Frequency.....	23
Switching System Topologies.....	23
Switching Topology.....	23
Scan/Multiplex Topology.....	24
Matrix Topology.....	26
Chapter 3 - Installation and Connections	31
Getting Started.....	31
Packing List.....	31
Unpacking and Inspection.....	31
System Requirements.....	31
Installation of the GXSW Software.....	32
Setup Maintenance Program.....	32
Overview of the GXSW Software.....	33
Installation Folders.....	33
GXSW driver Files Description.....	34
Driver and Virtual Panel.....	34
Programming Interface Files.....	34
Documentation.....	35
GXSW Example Programs.....	35
Configuring Your PXI System using the PXI/PCI Explorer.....	39
Board Installation.....	40
Before you Begin.....	40
Electric Static Discharge (ESD) Precautions.....	40
Installing a Board.....	40
Plug & Play Driver Installation.....	42
Removing a Board.....	42
Chapter 4 - Programming the Board	43
The GXSW driver.....	43
Programming Using C/C++ Tools.....	43
Programming Using Visual Basic.....	43
Programming Using Pascal/Delphi.....	44
Programming GXSW Boards Using ATEasy®.....	44
Using the GXSW Driver Functions.....	44
Initialization, HW Slot Numbers and VISA Resource.....	44
Board Handle.....	46

Reset.....	46
Error Handling	46
Driver Version.....	46
Panel.....	46
Distributing the Driver	46
Sample Programs	47
Sample Program Listing	47
Chapter 5 - Functions Reference.....	59
Introduction.....	59
GXSW Functions.....	59
GxSWGetDriverSummary	60
GxSWGetErrorString.....	61
Index	65

Chapter 1 - Introduction

Manual Scope and Organization

Manual Scope





This manual provides all the information necessary for installation, operation, and maintenance of the **GXSW** PXI Switch Matrix Board. The manual also covers the **GXSW** software package that includes the GXSW driver. This manual assumes the reader has a general knowledge of PC based computers, Windows operating systems, and a general knowledge of modular test equipment.

Manual Organization

The GXSW manual is organized in the following manner:

Chapter	Content
Chapter 1 – Introduction	Introduces the GXSW manual. Lists all the supported switching instruments and shows warning conventions used in the manual.
Chapter 2 – Introduction to Switching	Describes the theoretical and practical considerations for using and configuring switching instruments.
Chapter 3 – Installation and Connections	Provides instructions on how to install the various switching instruments and accompanying software.
Chapter 4 – Programming the Board	Provides a listing of GXSW driver files, general purpose/generic driver functions, and programming methods. Discusses various supported operating systems and development tools.
Chapter 5– Functions Reference	Provides a list of the GXSW functions. Each function is described along with its syntax, parameters, and special programming comments.

Conventions Used in this Manual

Symbol Convention	Meaning
	Static Sensitive Electronic Devices. Handle Carefully.
	Warnings that may pose a personal danger to your health. For example, shock hazard.
	Cautions where computer components may be damaged if not handled carefully.
	Tips that aid you in your work.

Formatting Convention	Meaning
Monospaced Text	Examples of field syntax and programming samples.
Bold type	Words or characters you type as the manual instructs. For example: function or panel names.
<i>Italic type</i>	Specialized terms. Titles of other references and information sources. Placeholders for items you must supply, such as function parameters

Supported Switching instruments

The following switching instruments are discussed in this manual and are fully documented in their separate manuals:

- GX6021 – 20 channels, RF multiplexer board
- GX6062 – 60 channels, RF multiplexer board
- GX6115 – 15 channels, high current relay board
- GX6125 – 25 channels, high density SPDT relay board
- GX6138 – 38 channels, Form-A relay board
- GX6264 – 2 x 32 Differential or 64 single-ended channels, scanner / multiplexer board
- GX6315 – 3 x 15 channels, high current relay board
- GX6325 – 3 x 25 channels, high density SPDT Relay board
- GX6338 – 3 x 38 channels, relay board
- GX6377 – Multifunction switching board
- GX6384 – Configurable high-density switch matrix board
- GX6616 – 6 x 16 channels, high density switch matrix board
- GX7016 – 20 slot 6U PXI chassis supports and the GENASYS switching subsystem modules:
 - GX6864 - 75 Ohm RF Multiplexer Switch Card, (4) 2 x 16 Multiplexers
 - GX6192 - High Frequency Multiplexer / Matrix Switch Card, 192 x 16 x 16
 - GX6256 - LF Multiplexer / Matrix Switch Card, 256 x 16 x 16

Additional boards may be available.

List of Acronyms

Acronym	Meaning
ADC	Analog to Digital Converter
ATE	Automated Test Equipment
BIT	Built-In Test
CO	Change Over
DPDT	Double-Pole, Double Throw
EMF	Electro Magnetic Field
ITA	Interface Test Adapter
NC	Normally Closed
NO	Normally Open
PCB	Printed Circuit Board
SPDT	Single-Pole, Double-Throw
SPST	Single Pole, Single Throw
TRD	Test Requirements Document
UUT	Unit Under Test

Chapter 2 - Introduction to Switching

This chapter discusses switching components and the various switching methods and topologies.

Test Systems

The Basic Test System

To effectively test a device, component, or subassembly involves monitoring its response to a set of prescribed input conditions. Testing generally means applying a stimulus to a unit under test and checking the response, as shown in

Figure 2-1:

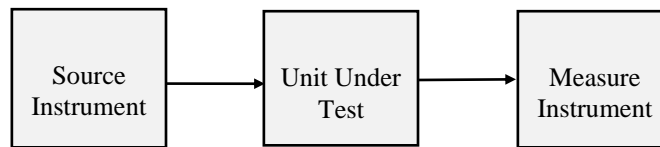


Figure 2-1: A Simple Test System

Almost all stimulus and response signals are electrical, but they could be non-electrical, such as heat, light, or motion. Non-electrical signals are usually converted to, or from, electrical signals to permit computer control or data analysis. Electrical signals are measured, or sourced, by the following common test instruments:

- *Measure* - Oscilloscope, voltmeter, frequency counter, spectrum analyzer, waveform analyzer.
- *Source* - Power supply, current source, pattern generator, pulse generator, or frequency synthesizer.
- *Source/Measure* - Ohmmeter, capacitance meter, network analyzer, signature analyzer.

Measure and source functions can be combined in the same instrument and even in the same leads, for example, an ohmmeter that measures voltage and source current on two leads.

In basic test systems, the source and measure instruments are connected to the unit under test (UUT) by direct interface wiring. A better test system makes these connections via switching. The purpose of employing a switching system is to:

- Test several UUTs during one test session.
- Test multiple points on a UUT.
- Connect many measure or source instruments.

The various configurations are shown in Figure 2-2 through Figure 2-4. The basic configurations may be combined to form more complex systems. In most systems, it is best to measure the output of the source instruments directly. This feature may require additional switching.

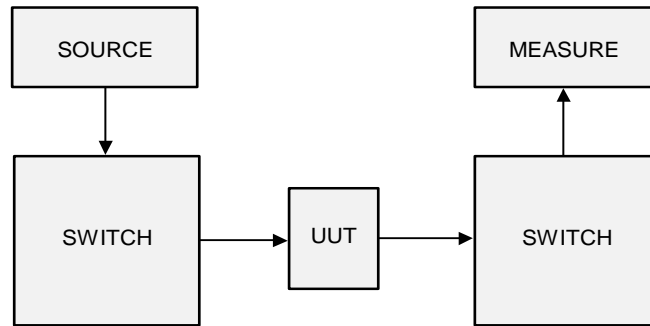


Figure 2-2: Multiple Test Points

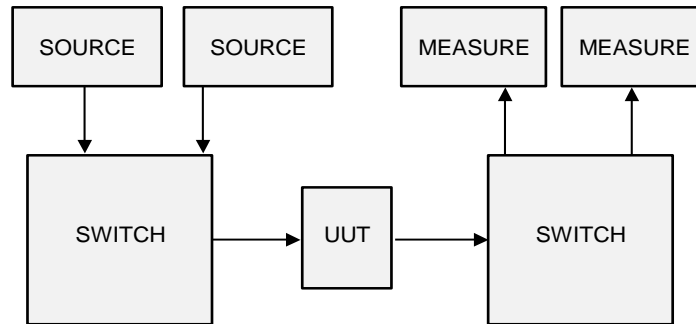


Figure 2-3: Multiple Instruments

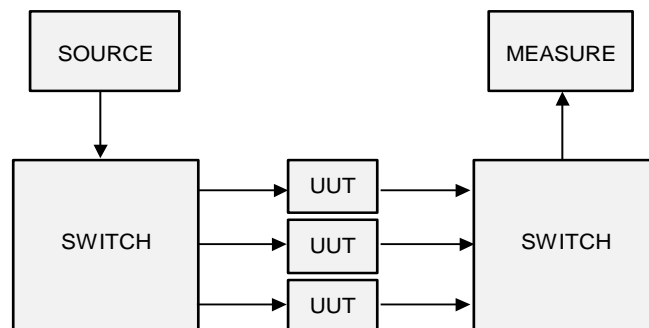


Figure 2-4: Multiple UUTs

Reasons to Automate Test Systems

Automated systems are produced by using the built-in capability of the instruments to run test programs and output the data results as shown in Figure 2-5. Often, the switching function is included within the sourcing or measuring instrument, such as an oscilloscope.

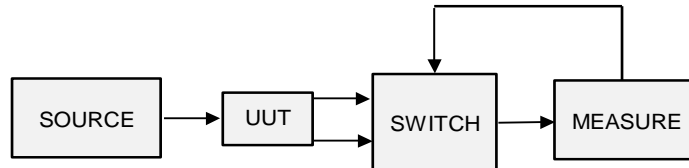


Figure 2-5: Measure Instrument Limited Control

As the automatic test system size increases, or as test requirements become more complex, a computer is added to run a test program and execute tests. Advantages to automatic test systems include:

- Testing permits operator interaction. The operator can base current test parameters on previous test results.
- Known system problems are corrected.
- Because operator intervention is less, there are fewer chances for test errors.
- Test results are more effectively analyzed since they are compiled into a database. Complex calculations are possible.
- Throughput of units under test is higher.

When automating the test system, switching functions must also be automated. In systems where the signals are electrical, a programmable instrument called a scanner, matrix, programmable switch, multiplexer, or switching network is employed.

Designing Automatic Test Systems

Designing switching systems for an automatic test system requires knowledge of the input/output signals to be switched and the test routine to perform the test. Automated test systems are usually designed with the flexibility to handle a variety of signals because test requirements change frequently.

The first step in designing a test system is to produce a Test Requirements Document (TRD), which outlines the system configuration and switching needs. Given that versatility is of utmost importance, designing the switching function may be one of the most difficult parts. To help design a switching system, one of the following switching topologies may be used, alone or in combinations. Figure 2-6 to Figure 2-8 describe the basic topologies.

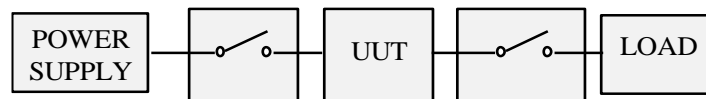


Figure 2-6: Switch Topology

Basic Switch Topology

This basic switch topology connects one input to one output. It is generally used in control applications to:

- Connect a UUT to a load.
- Connect a power supply or control signal to the UUT separate from other source-measure-interface circuitry.
- Initiate an independent element of the system. For example, a device with non-electrical input or output, such as a motor or transducer.

Marvin Test Solutions' GX6315 provides 3x15 independent Single Pole, Double Throw (SPDT) relays that may be used for power switching in the basic switch topology. The GX6338 provides 3x38 independent Single Pole, Single Throw (SPST) relays that may be used for low current switching in the basic switch topology. For more information about the SPDT and SPST configurations, refer to Switching Types further in this section.

Multiplex or Scan Topology

The multiplex or scan topology is used to connect one instrument to multiple UUTs or multiple instruments to a single UUT. The connections can be sequential (such as scan in Figure 2-7) or non-sequential (such as multiplex in Figure 2-8). In addition, multiplexing permits more than one simultaneous connection. For example, routing the UUT output to an AC voltmeter and to a frequency counter.

The 1:X configuration is sometimes used to connect a source to many points or a single point to many measure instruments. The X:1 configuration is used to connect several sources to a single point or many points to a single instrument. To synchronize source and measurement connections, a 1:X/X:1 pair can be switched together.

As shown in Figure 2-7, the scan topology is equivalent to a rotary selector switch. With this type of topology, there are two common versions:

- **Make-Before-Break**, which makes the new connection before breaking the existing connection.
- **Break-Before-Make**, which breaks the existing connection before making the new connection.

Either of these types can be implemented in a test system. Unless otherwise indicated, the non-shorting type is assumed.

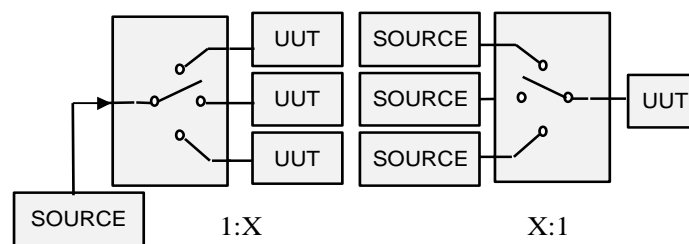


Figure 2-7: Scan Topology

Marvin Test Solutions' GX6264 is Multiplexer/Scanner that can be used for the Scan Topology. All products are of the Break-Before-Make version, although it is possible to switch multiple points simultaneously using different Scan Groups.

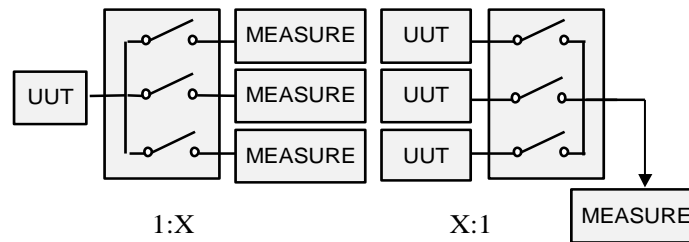


Figure 2-8: Multiplex Topology

Switch Matrix Topology

The Switch Matrix Topology connects multiple inputs to multiple outputs. With this configuration, multiple instruments are often connected to multiple points on a UUT. Though the switch matrix topology is the most complex, it certainly is the most versatile. Figure 2-9 illustrates the switch matrix topology in simple terms.

The Switch Matrix topology provides the most flexible switching method. However, the following issues must be addressed when using a switch matrix:

1. Stubs (unused relays) may generate electrical noise if the frequency is above 10 MHz. If you are using a switch matrix to switch high-frequency signals, use the "cleaner" switch paths with shorter stubs. For example, the bottom-left switch shown in Figure 2-9 is the "cleanest" path since it has the shortest stubs. The top-right switch in the same figure is the "noisiest" path as it has the longest stubs.
2. The flexibility provided by the switch matrix means that any point could be connected to any other point. Extreme care should be taken when developing the control software to prevent shorts between low-impedance sources (such as power supplies). A typical method to prevent such shorts is to use a resistor in series between the power supplies and the switch matrix if these signals are for measurement purposes only.

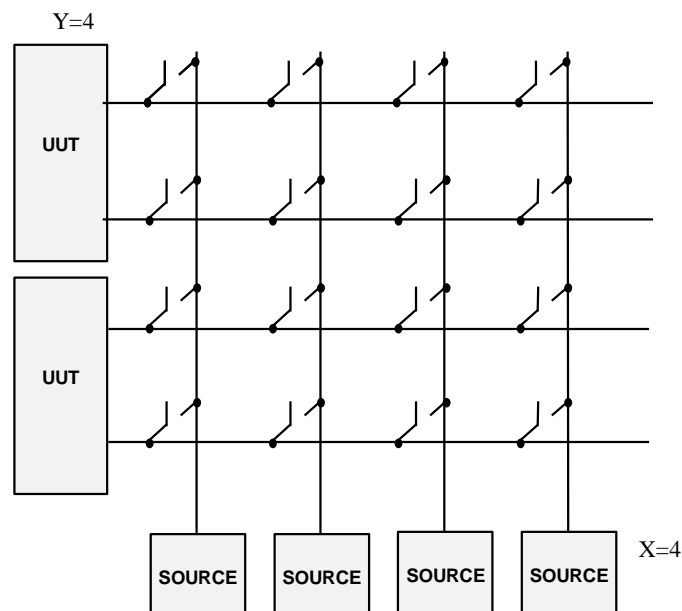


Figure 2-9: Switch Matrix Topology

Marvin Test Solutions' GX6616 is a true switch matrix that can be used when the switch matrix topology is selected. This product has 6 groups of 2x16 and offer multiple configurations from 2x96 to 6x16 (any of the 6 points may be connected to any of the 16 points).

Switching Instrument Criteria

Every time a signal passes through a connecting cable or switch point, errors may be introduced. It is important to carefully select the correct switching elements and proper topology in order to maintain the integrity of the signal.

Newer switching elements come very close to the ideal switch. The ideal switch is:

- An open circuit (zero capacitance, infinite ohms) when open
- A short circuit when closed (zero ohms)
- Completely isolated from the other switches
- A discrete channel completely isolated from the others.

All of these factors are critical in high accuracy and low-level signal applications. In many systems, different types of switches must be included to balance the technical and economic limitations.

The following is a list of important factors when choosing a switching instrument:

- Cost
- Cost of expansion/adding more boards
- Switching topologies available
- Minimum/maximum number of switch points
- Variety of switching elements available
- Electrical specifications
- Ease of change, as the system requirements change.

Understanding Switching Components

Switching Types

When designing the switching portion of a given system, it is important to define the configuration and implementation of the individual switches.

There are three terms used to describe switch configurations:

- Number of poles (e.g. Single/double/triple Pole).
- Single/double Throw
- Contact Form.

The term *pole* refers to the number of common terminals contained within a given switch. Figure 2-10 shows a single-pole switch in the open position. The term *throw* refers to the number of positions in which the switch may be placed that create a signal path or connection. A complete description of Figure 2-10A would then be single-pole, single-throw, normally-open (SPSTNO).

Figure 2-10B shows a single-pole, double-throw (SPDT) switch. One terminal in Figure 2-10B is normally-open (NO) and the other is normally-closed (NC). Only one of the terminals is connected, depending on the state of the switch. A complete description of Figure 2-10B would then be Single Pole, Double Throw (SPDT).

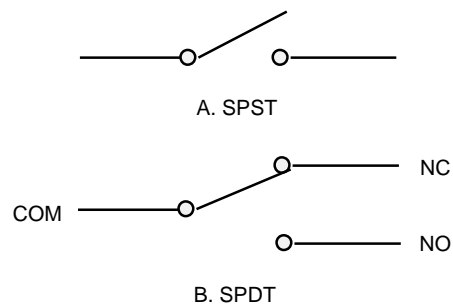


Figure 2-10: Single Pole Schematics

Figure 2-11 shows a double-pole, double-throw (DPDT) switch. Both poles are actuated simultaneously when the relay is energized.

The “form”, or “contact form”, is a term used by relay manufacturers to describe the contact configuration of a relay. “Form A” refers to a single-throw, normally-open switch. “Form B” refers to a single-throw, normally-closed switch, and “Form C” indicates a double-throw switch. An alternate designation for a double throw switch is a Change Over (CO). Using this nomenclature scheme, virtually any contact configuration may be described.

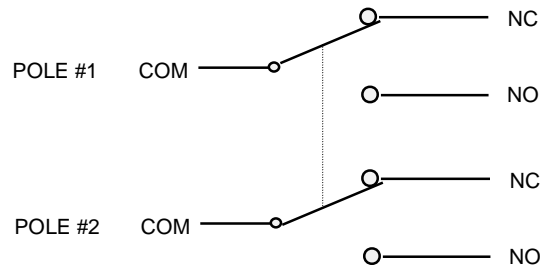


Figure 2-11: DPDT Schematic

Switching Elements

When a switch turns on, it behaves according to the characteristics and properties of the switching element. Therefore, any element chosen for use in a switching system should approach the ideal switch model.

Figure 2-12 illustrates the schematic representations of the four types of electrically controllable switching elements. Because of the advancement of relay designs, the relay family of switching elements offers the best overall performance at a reasonable cost.

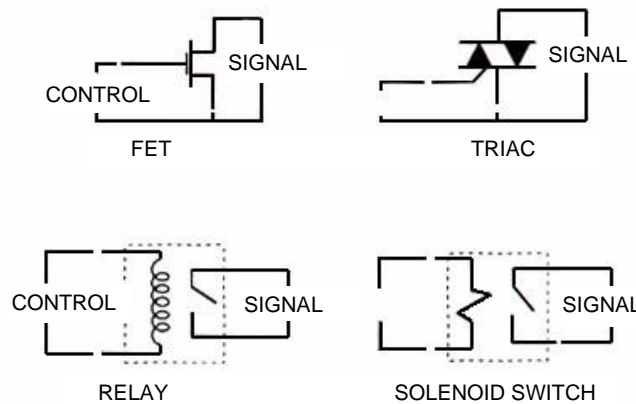


Figure 2-12: Switching Elements

Each of the four types shown above has advantages and disadvantages. For Automatic Testing purposes however, the Relay offers the best performance. A FET switch is the fastest switch of the group, although it has a high ON resistance and the switchable signal level is typically limited. The Triac is a good solution for high-current, high-voltage switching but is rarely being used in ATE. The same holds true for the Solenoid Switch, although some applications require its use. Since the relay is the most common switching type used in ATE system, the rest of this section will focus on types of relays and their advantages.

Types of Relays

Many types of relays exist, but only a few are suitable for ATE applications. Among the more common types are electromechanical (armature) and dry reed. Table 2-1 compares some key parameters of these two relays.

Relay Type	Isolation (EI)	Speed (ms)*	Power (VA)	Life at rated load
Electro-mechanical	10^7 - 10^{14}	2-6ms	1-750	10^5 - 10^7
Dry Reed	10^7 - 10^{14}	0.5-1ms	0.5-60	10^5 - 10^7

* Speed (ms) represents Operate Time - does not include bounce time.

Table 2-1: Relay Comparison, Typical Specifications

Marvin Test Solutions' switching products use both types of relays to offer the best benefits for ATE applications.

Electromechanical Relays

All electromechanical relays have a coil of wire with a rod passing through the middle (to form an electromagnet), an armature, and one or more sets of contacts. When the coil is energized, the electromagnet attracts one end of the armature mechanism, which in turn moves the contacts.

Figure 2-13 shows a typical electromechanical relay with the main operating elements labeled and a scheme for actuating the armature.

A typical coil must generate a strong magnetic field to completely actuate the relay. Several parameters relate to the force necessary to actuate the relay, among them:

- Spring tension of the armature
- Spacing of the contacts

The mass of the armature mechanism.

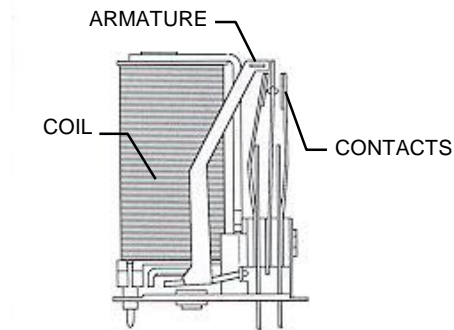


Figure 2-13: Electromechanical Relay

Dry Reed Relays

A dry reed relay also operates by energizing a coil. But with this relay type, the coil is wound around the switch so that the induced magnetic field closes the switch. Figure 2-14 illustrates a simplified representation of a typical reed relay.

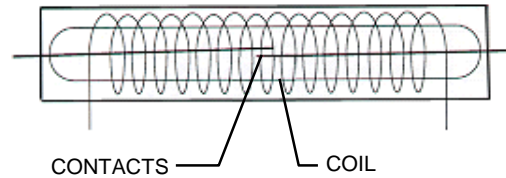


Figure 2-14: Reed Relay

The switch is made from two thin, flat strips of ferromagnetic material (called reeds) with contacts on the overlapping ends. Leads are connected to the outside ends of the reeds and the entire assembly is sealed in a hermetic glass tube. The tube holds the leads in place. See Figure 2-15.

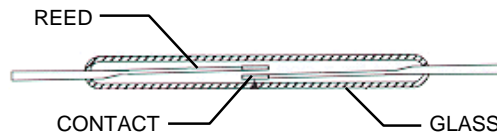


Figure 2-15: Dry Reed Switch

Most relays are manufactured normally-opened. Some however are produced normally-closed. Normally-closed switches are made in one of two ways. The first method is to make the switch so that the contacts are touching each other. The second method uses a small permanent magnet to hold normally-open contacts together. The field from the coil opposes the field of the magnet, allowing the contacts to open.

In an attempt to lower the characteristic capacitance and leakage currents of the reed relays, an electrostatic shield is sometimes added between the switches and the coil. This shield is typically metal foil that is grounded to a low impedance point in the circuit. Common connection points (low impedance) are signal low and guard. In addition, an electromagnetic shield can also be added around the outside of the coil. This special shield keeps the field generated by the energized coil from interfering with the other components.

Maximum Relay Voltage

The maximum voltage that a relay can reliably switch at is determined by:

- The gap (spacing) between the contacts
- The dielectric of the gas inside the relay.

An arc will develop if the gap is too small between the two contacts. Arcing is common when switching high voltage, but if the relay has the proper gap, arcing is reduced.

In addition, the environment affects the degree of arcing. A humid, or highly oxygenated, environment promotes arcing and a dry climate reduces arcing. Open relays allow ambient air into the switching area while sealed relays contain a controlled internal environment from the relay manufacturer. Mechanical type relays are made as open or sealed units. Reed relays, by virtue of their construction, are always sealed.

When the voltage across the relay contacts exceeds the maximum switching voltage specified for the relay, arcing is usually present. An arc due to an AC signal is usually reduced as soon as the voltage level drops to zero.

Maximum Relay Current

Typical relay specifications break down maximum current into maximum carry current and maximum-switched current.

Carry current is defined as the current that can be passed by previously closed contacts. The cross-sectional area of the path from pin to pin through the switch limits carry current.

Switched current is the maximum current that can be interrupted by the relay. Contact plating and material are the primary factors in the switched current specification. If a high current is switched constantly across the contacts, the heat rises and arcing at the contacts degrades them at a rapid rate. In extreme cases, the contacts may weld together.

If the specified life of the relay is to be attained, power rating of the relay must also be complied with the current and voltage values.

Maximum Relay Power

The maximum power that a relay can switch relates to the temperature rise in relation to reasonable contact life. An over-power condition dramatically shortens relay life. Signal handling relays have their power rating specified in either volt-amps (VA) or watts.

Relays that have their power rating given in watts are DC relays. If the relay is a DC type, the relay's power must be used to rate the relay's power handling capability.

AC type relays are designed to handle AC loads and are usually rated in volt-amps. When power is rated in VA, the power factor is not a consideration and volts switched times amperes switched or carried is the calculation used.

Contact Potential

EMF is generated when two dissimilar metals come into contact. Because a relay contact is typically a plated alloy, even the two supposedly similar halves of the contact are not the same. There are differences due to alloy composition and the working of the metal as well as the thermal EMFs generated by the plating to the alloy junction.

In addition, EMF is generated when heat is conducted along the leads of the switch. The difference in the heat flow between the different leads causes a thermal EMF. A secondary source of thermal EMF is from winding inconsistencies and variations in coil wire diameter. In a reed relay, these radiate heats, this causes temperature variations in the switch.

In a test system, these thermal EMFs add a voltage error to the measurement system. Depending on the relay, this error may range from less than a microvolt to tens of millivolts. If the error is significant with respect to the sourced or measured reading, the thermal EMF must be known and compensated.

Table 2-2 gives some thermoelectric potential for common connection materials. Clean copper connections are very important, since the potential across a clean Cu - Cu junction is $< 0.2\mu\text{V}/^\circ\text{C}$, and the potential across a Cu - CuO (copper oxide) junction is $1000\mu\text{V}/^\circ\text{C}$.

Materials	Potential
Cu - Cu	$< 0.2\mu\text{V}/^\circ\text{C}$
Cu - Ag	$0.3\mu\text{V}/^\circ\text{C}$
Cu - Au	$0.3\mu\text{V}/^\circ\text{C}$
Cu - Cd/Sn	$0.3\mu\text{V}/^\circ\text{C}$
Cu - Pb/Sn	$1-3\mu\text{V}/^\circ\text{C}$
Cu - Si	$400\mu\text{V}/^\circ\text{C}$
Cu - Kovar	$40\mu\text{V}/^\circ\text{C}$
Cu - CuO	$1000\mu\text{V}/^\circ\text{C}$

Table 2-2: Typical Thermoelectric Potentials

Operating Speed

The rate at which the contacts may be cycled for reliable operation is known as the operating speed. It is limited by the actuation and release times. Actuation time is when power is applied to the coil until the contacts have settled. Release time is the opposite of actuation time. It is measured from the time power is removed from the coil until the contacts have settled again (including bounce).

An operating frequency, or repetition rate, lower than that allowed by the switching times is often specified for a relay. This is done to limit temperature rise and allow for contact settling. Of these two, temperature rise is more critical. When a relay is used near its power or voltage rating, the duty cycle must be adjusted to allow for cooling time. A lower operating frequency or altered duty cycle accomplishes cooler temperatures.

Contact Resistance versus Life

Contact resistance is the resistance across a closed set of contacts. This parameter is typically measured using a four-wire configuration with nominal voltage applied to the coil. Unless otherwise specified, a contact resistance specification is for new conditions.

For dry switches, the contact resistance is inversely related to the contact area. Electromechanical relays with large contacts have a lower contact resistance than a reed switch with relatively small contacts.

Life, or the number of operations for which a relay is expected to perform, is typically limited by mechanical operations and by the number of operations under a certain load until a specified resistance across closed contacts is reached. Without an end-of-life contact resistance specification, the life under load specification is meaningless. Dry switches typically have a life specified to a contact resistance less than 1 or 2Ω .

A relay may fail in one of several modes, including mechanical (actuation) failure and contact failure. A coil wire break, a glass seal fracture or reed fatigue in reed switches, or an armature mechanism failure in electromechanical relays can cause mechanical failure.

Contact failure can occur when excessive heat or a high current or voltage pulse causes the contacts to weld. Oxidation, excessive charring and pitting, or insulating deposits on the contacts can cause an open circuit.

Because actual contact life is important, much effort has gone into finding and choosing "good" contact materials. Gold, silver, and palladium have traditionally been used for their good electrical conductivity and relatively low susceptibility to oxidation. Increasingly, rhodium and ruthenium are being plated over more traditional contact

materials to increase the contact life. Rhodium's structure is cubic, which is more stable than that of earlier contact plating. For even greater stability, the hexagonal structure of ruthenium is used.

Relay Coil Power

Relay coil power is the amount of power in watts needed to actuate the relay. A reed switch or the armature of an electromechanical relay requires a set amp-turns product to actuate the relay. Many turns of fine wire are used to keep the necessary current and power to a minimum.

The amp-turns requirement and the resistance of the coil also determine the pull-in, hold, and dropout voltages. These are shown for a typical 5V DC relay in Figure 2-16. Often more power is needed to actuate a relay than to maintain a given state. Once the switch is closed, the magnetic path has a lower reluctance. A lower reluctance means a weaker field can hold the switch closed.

As with any wire, the coil's resistance changes with temperature. This affects the pull-in voltage, as shown in Figure 2-17.

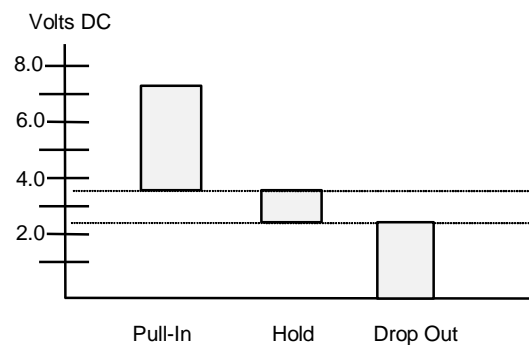


Figure 2-16: Coil Voltage Levels

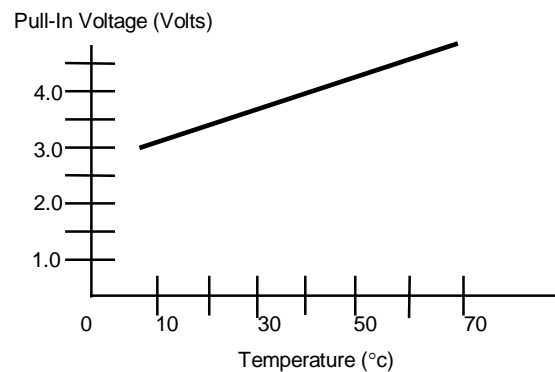


Figure 2-17: Pull-in vs. Temperature

A relay coil also produces inductance because it is a coil of wire. As an inductor, the current in the coil cannot be changed instantaneously, since $V=L(di/dt)$, without generating an excessive voltage spike. A diode is often used across the coil with the cathode at the positive voltage end of the coil. The suppression or “back” diode will conduct the coil current when the relay drive is removed. See Figure 2-18.

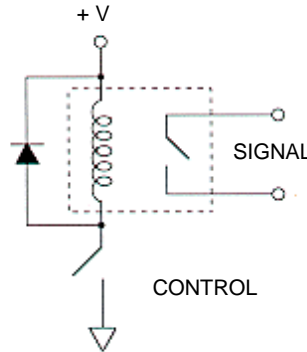


Figure 2-18: Suppression Diode

Insulation Resistance

Insulation resistance (IR) is a measure of the resistance between two isolated pins on the relay. Available insulator materials provide a wide range of insulation resistance (10^5 - $10^{14}\Omega$) in various environments.

The primary factor to limiting IR is bulk leakage and surface leakage. Bulk leakage is through an insulating material, usually after it has absorbed moisture from the environment.

Surface leakage is across the surface of the material, usually due to contamination such as salts or flux and/or other chemicals from fingers.

The properties of some insulators change with applied voltage, so that it is important to evaluate IR at or above the expected operating voltage.

Relay Capacitance

All the components of a relay have inherent capacitances that are determined by the materials and physical configuration of the relay.

The typical capacitance from switch contacts to the coil is 0.2 - 15pF. This capacitance can couple noise or other signals from the contacts to the control lines, and vice versa. Noise can also be coupled to a different switch through its contact-coil capacitance if it is driven by the same coil supply. An electrostatic shield between the switch and the coil can reduce noise by as much as a factor of 110. In addition, there is also capacitance across open contacts. This type of capacitance depends on the contact area and the gap of the switch and is typically less than 2pF.

These inherent capacitances, due to physical characteristics of the relay, are one factor in limiting the frequency of the signal the relay switches.

Signal Frequency

To obtain higher switching rates and good RF performance specialized contacts and relay architectures are used. Operating frequencies greater than 50 MHz are possible with these types of relays.

Intercontact capacitance and contact circuit loss are factors that determine how well a relay is suited to high frequency applications. In addition, the design of the printed circuit board (PCB) also contributes to the selection of RF relays. Beveled etched signal paths and ground-guards are integrated into newer switch matrix PCB designs.

Switching System Topologies

In a typical test system one or all of the three switching topologies exist: switch, scan/multiplex, and/or matrix. The various topologies with some additional specialized information are shown below. The various switching topologies can be combined to meet the requirements of the test system.

Switching Topology

As illustrated in block diagram form, the basic switch topology connects one input to one output. This input can be one or more signal lines, connected through Form A, B, or C switching elements. One pole is required for each signal line, with all poles controlled together. Figure 2-19 shows an example using each switch topology.

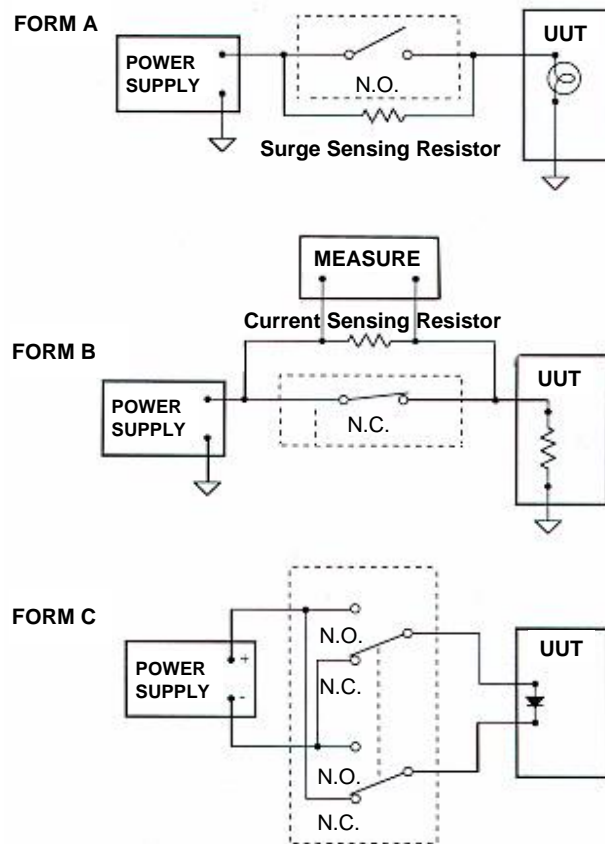


Figure 2-19: Switch Topology (Form A, B, C)

A typical application for this switch topology is power and control applications. Because each switch is independent and isolated, more than one switch can be closed at the same time.

Hardware that implements the switch topology often provides the flexibility to select Form A, B, or C switching and the number of poles controlled together. If the test system designer is willing to do external wiring, then any topology can be built from the basic switch topology.

Marvin Test Solutions' GX6315 () can be used to perform all switching functions shown in figure 2-19. Since these products contain individually controlled Form-C relays, The COM and NO poles of the relay will be used in the first example, the COM and NC poles will be used in the second, and two relays (COM, NO, and NC) will be used in the third.

Scan/Multiplex Topology

Most scan and multiplex topologies connect one input to "n" outputs or "n" inputs to one output. These are the most commonly used topologies in signal switching for small systems. The simplest scanner is a Form C contact as shown in Figure 2-20A, switching a supply to either of two loads. Implementing this with two Form A contacts gives a simple multiplexer. The supply can now be connected to either one or both loads as seen in Figure 2-20B.

The major advantages of multiplexing over scanning are:

Multiplexing permits an all-open state.

Multiplexing permits any order. Scanning is sequential.

Multiplexing allows simultaneous connections. Scanning does not.

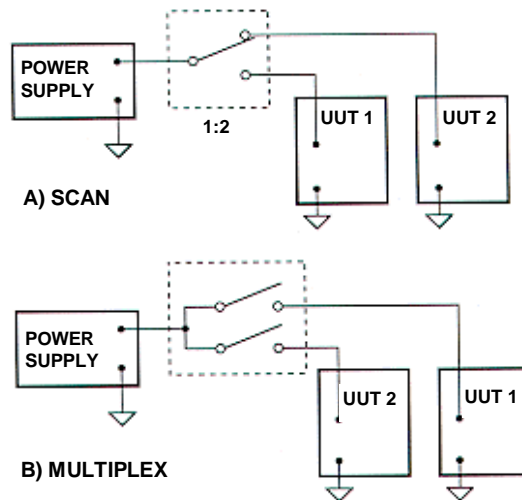


Figure 2-20: Basic Scan/Multiplex

The multiplexing method is the most used because it provides the most flexibility. In addition, sequencing of multiplex connections can be programmed in any order. Scanning is most often used in stand-alone systems without computer control. When designing a multiplex test system, the designer must consider the consequences of simultaneous connections in the system in case of software error or hardware failure (stuck relay). Various strategies can be used to handle this, from reconfiguring the switching or adding protection circuitry to adding software checks in the test program.

Multiplexing and Scanning are used in a test system to connect one instrument to multiple UUTs or multiple instruments to one UUT. Sample configurations are shown in Figures 2-21 and 2-22. Figure 2-21 shows 2-pole multiplexing of high and low connections to the UUT. The same hardware can be used as both the 1:N and N:1 blocks. Figure 2-22 shows a ground-referenced system with 1-pole switching. Two additional multiplexers are used to connect the same instruments to a second UUT.

Marvin Test Solutions' GX6264, GX6315, and GX6338 may be used in the examples shown in Figures 2-20, 2-21, and 2-22. The GX6264 offers programmable configuration of differential (2 wire) and single-ended (one wire).

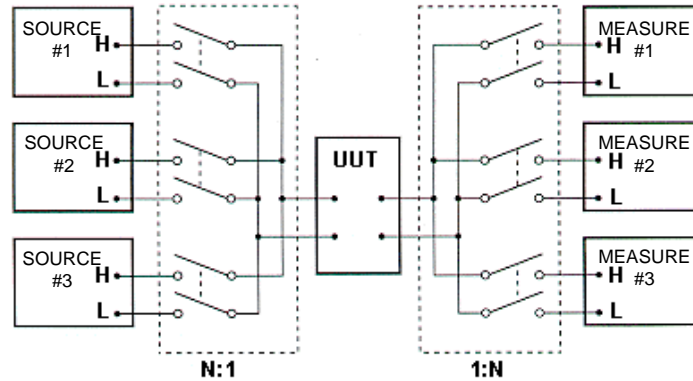


Figure 2-21: Two-Pole Multiplexing

For multiplexed switching, Form A contacts are used almost exclusively (Figure 2-19). One and two-pole configurations are common, with more poles used in special situations (i.e. guarding).

Configurations equivalent to multiple pole configurations can be built from single-pole multiplexers controlled together. However, proper shielding, guarding, and noise cancellation are often more difficult. Control can also be synchronized to implement a 1:N/N:1 switched pair.

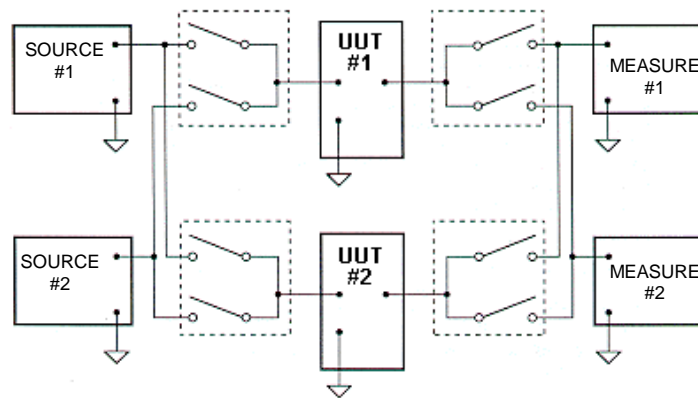


Figure 2-22: One Pole Multiplexing

Matrix Topology

The matrix topology is used to connect any input to any output. Simultaneous connections are possible with one input driving many outputs or, less often, several inputs driving one output. The most common terminology to describe matrix size is by rows and columns. As shown in Figure 2-23, any row can be connected to any column by closure of the relay at the intersection (cross point) of a row and column. This flexibility of the connections is the main advantage of a matrix switching system. The matrix topology connects multiple instruments to multiple points on a UUT, or multiple instruments to multiple UUTs. Direct connections between source and measure instruments are also possible.

Figure 2-23 (exploded view) illustrates a cross point implementing a 3-pole, Form A, switching of high, low, and guard signals. This type of structure permits maximum flexibility when connecting instruments and the UUT to the matrix. When designing the matrix, the number of poles per cross point is related to how the instruments are connected to the matrix. Single-pole matrix switching is the most common type.

Marvin Test Solutions' GX6616 is true switch matrix that may be used in the examples shown in figures 2-23, 2-24 and 2-25. These products offer a single-ended configuration of up to 6 rows by 16 columns or 2 rows by 96 columns. In the differential (2 wire) configuration, these products may be used to switch 3 rows by 16 columns or 2 rows by 48 columns. Using several boards, these products may be used to create unlimited switch matrices.

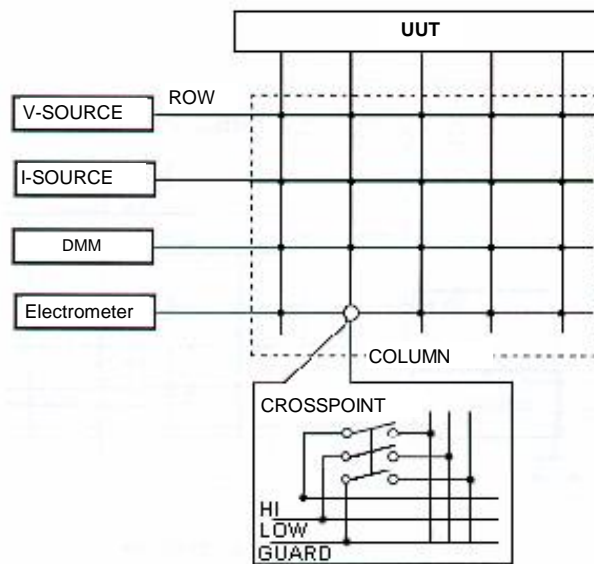


Figure 2-23: Matrix Topology

In switching matrices, two connection methods can be used, as shown in Figure 2-24. The common ground system is used in high frequency systems where low impedance returns and coaxial cabling are required. The switched low configuration is used in low frequency or DC applications where floating measurements are required.

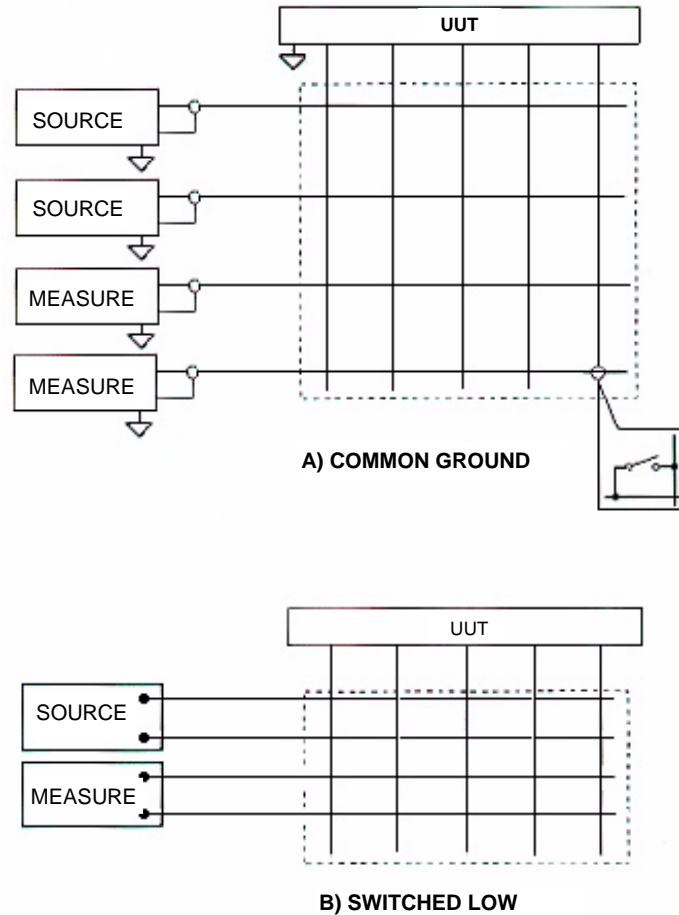


Figure 2-24: Single-Pole Matrix

Two-pole matrix switching is used for differential (balanced) systems or to make 4-wire measurements. An example is shown in Figure 2-25 where a combined source/measure instrument (DMM) makes a 4-wire measurement using two 2-pole rows of the matrix.

The 2-pole configuration is also used for guarded source or measure connections involving high impedance or low-level signals as shown in Figure 2-25B. Since guarding isolates sensitive lines from adjacent circuitry, the construction and wiring of hardware for guarded systems differs from that used in a general-purpose 2-wire matrix.

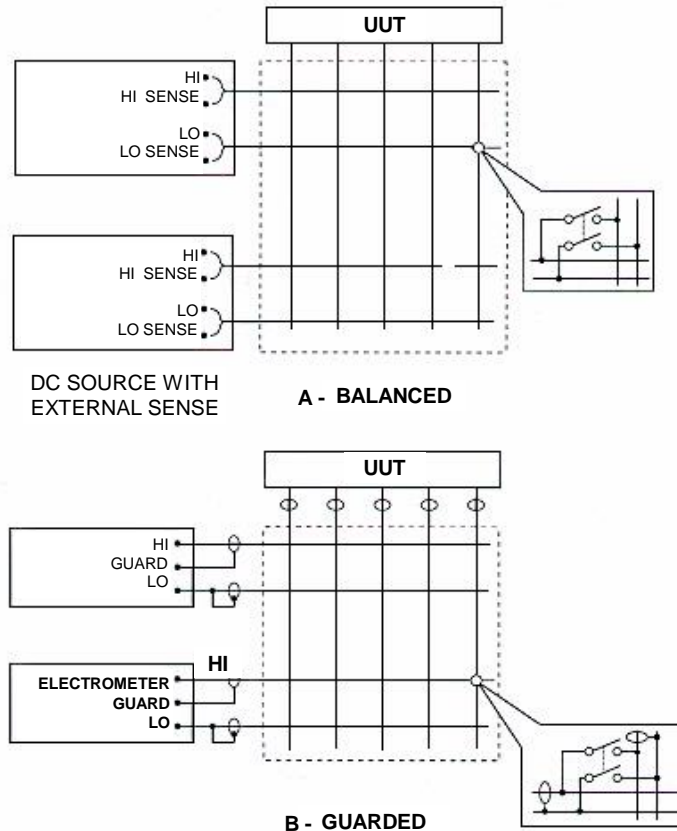


Figure 2-25: Two-Pole Matrix

For sensitive measuring applications, the 3-pole matrix switching method is useful. The 3-pole method isolates the high and low signals from the other signals. This is done by additional shielding which is not at circuit low or earth potential.

The 3-pole configuration is also used for isolating source/measure signal pairs with a driven shield (guard).

Switching matrices may be used as a building block for larger matrices by interconnecting them along the columns or rows. Matrix blocks can be connected to form a long narrow matrix. This is useful when a large number of instruments and UUT connections are required, but only a few signals are used for each test step.

As the following points illustrate, the matrix topology is the most efficient method because it offers:

- Reduced hardware and cost
- Straight forward programming
- Maximum switching flexibility via software control

- Maximum test flexibility to run different devices or different test programs. However, there are several drawbacks to matrix switching:
- Safety considerations. It is difficult to confine hazardous signals to certain paths.
- Increased cross talk between paths. Shunt loading and impedance matching limits bandwidth.
- Additional cost of switches for every possible signal path.
- Lower reliability. More switches and increased possibility of misconnection.

Chapter 3 - Installation and Connections

Getting Started

This section includes general hardware installation procedures for all GXSW switching instruments and installation instructions for the GXSW software. Before proceeding, please refer to the appropriate chapter to become familiar with the board being installed.

To Find Information on..	Refer to..
GXSW driver Installation	This Chapter
Programming the boards	Chapter 4
GXSW Reference Functions	Chapter 5
Specific GXSW boards (e.g. GX6264, GX6315, etc.) and their driver functions.	The User's Guide for the specific board

Packing List

All GXSW boards have the same basic packing list, which includes:

1. GXSW User's Guide
2. The specific GXXXXX board User's Guide (e.g. GX6264 User's Guide)
3. GXSW driver Disk
4. GXSW Switching instrument
5. Mating Connector(s)

Unpacking and Inspection

After removing the board from the shipping carton:



Caution - Static sensitive devices are present. Ground yourself to discharge static.

1. Remove the board from the static bag by handling only the metal portions.
2. Be sure to check the contents of the shipping carton to verify that all of the items found in it match the packing list.
3. Inspect the board for possible damage. If there is any sign of damage, return the board immediately. Please refer to the warranty information at the beginning of the manual.

System Requirements

The GXSW instrument board is designed for use with a 3U or 6U cPCI or PXI compatible chassis. The software is compatible with any computer system running Windows XP SP3, VISTA, 7, 8, and 10 (32/64 bit) operating systems. In addition, Microsoft Windows Explorer version 4.0 or above is required to view the online help.

Each board requires one unoccupied 3U PXI bus slot.

Installation of the GXSW Software

Before installing the board, it is recommended that you install the GXSW software as described in this section. To install the GXSW software, follow the instruction described below:

1. Insert the Marvin Test Solutions CD-ROM and locate the GXSW.EXE setup program. If your computer's Auto Run is configured, when inserting the CD, a browser will show several options. Select the Marvin Test Solutions Files option, and then locate the setup file. If Auto Run is not configured, you can open the Windows explorer and locate the setup files (usually located under \Files\Setup folder). You can also download the file from Marvin Test Solutions' web site (www.MarvinTest.com).
2. Run the GXSW setup and follow the instruction on the Setup screen to install the GXSW driver.

Note: When installing under Windows, you may be required to restart the setup after logging-in as a user with Administrator privileges. This is required in-order to upgrade your system with newer Windows components and to install kernel-mode device drivers (HW.SYS and HWDEVICE.SYS) which are required by the GXSW driver to access resources on your board.

3. The first setup screen to appear is the Welcome screen. Click Next to continue.
4. Enter the folder where GXSW is to be installed. Either click Browse to set up a new folder, or click Next to accept the default entry. The default entry for 32 bit machines is C:\Program Files\Marvin Test Solutions\GXSW, and for 64 bit Windows C:\Program Files (x86)\Marvin Test Solutions\GXSW.
5. Select the type of Setup you wish and click Next. You can choose between Typical, Run-Time and Custom setups types. The Typical setup type installs all files. Run-Time setup type will install only the files required for controlling the board either from its driver or from its virtual panel. The Custom setup type lets you select from the available components.

The program will now start its installation. During the installation, Setup may upgrade some of the Windows shared components and files. The Setup may ask you to reboot after completion if some of the components it replaced were used by another application during the installation – do so before attempting to use the software.

You can now continue with the installation to install the board. After the board installation is complete you can test your installation by starting a panel program that lets you control the board interactively. The panel program can be started by selecting it from the Start, Programs, GXSW menu located in the Windows Taskbar.

Setup Maintenance Program

You can run the Setup again after GXSW has been installed from the original disk or from the Windows Control Panel – Add Remove Programs applet. Setup will be in the Maintenance mode when running for the second time. The Maintenance window show below allows you to modify the current GXSW installation. The following options are available in Maintenance mode:

- **Modify.** When you want to add, or remove GXSW components.
- **Repair.** When you have corrupted files and need to reinstall.
- **Remove.** When you want to completely remove GXSW.

Select one of the options and click **Next** and follow the instruction on the screen until Setup is complete.

Overview of the GXSW Software

Once the software is installed, the following tools and software components are available:

- **GXSW Panel** – Configures and controls the GXSW various features via an interactive user interface.
- **GXSW driver** - A DLL based function library (GXSW.DLL, located in the Windows System folder) used to program and control the board.
- **Programming files and examples** – Interface files and libraries for support of various programming tools. A complete list of files and development tools supported by the driver is included in subsequent sections of this manual.
- **Documentation** – On-Line help and User's Guide for the GXSW board, GXSW driver and panel.

HW driver and PXI/PCI Explorer applet – HW driver allows the GXSW driver to access and program the supported boards. The explorer applet configures the PXI chassis, controllers and devices. This is required for accurate identification of your PXI instruments later on when installed in your system. The applet configuration is saved to PXISYS.ini and PXIE SYS.ini and is used by Marvin Test Solutions instruments HW driver and VISA. The applet can be used to assign chassis numbers, Legacy Slot numbers and instrument alias names. The HW driver is installed and shared with all Marvin Test Solutions products to support accessing the PC resources. Similar to HW driver, VISA provides a standard way for instrument manufacturers and users to write and use instruments drivers. VISA is a standard maintained by the VXI Plug & Play System Alliance and the PXI Systems Alliance organizations (<http://www.ivifoundation.org>, <http://www.pxisa.org/>). The VISA resource manager such as National Instruments **Measurement & Automation** (NI-MAX) displays and configures instruments and their address (similar to Marvin Test Solutions' PXI/PCI Explorer). The GXSW driver can work with either HW or VISA to control an access the supported boards.

Installation Folders

The GXSW driver files are installed in the default folder **C:\Program Files\Marvin Test Solutions\GXSW**, or on 64 bit machines they are installed in the default folder **C:\Program Files (x86)\Marvin Test Solutions\GXSW**. You can change the default GXSW folder to one of your choosing at the time of installation.

During the installation, GXSW Setup creates and copies files to the following folders:

Name	Purpose / Contents
...\Marvin Test Solutions\GXSW	The GXSW directory. Contains panel programs, programming libraries, interface files and examples, on-line help files and other documentation.
...\Marvin Test Solutions\HW	HW device driver. Provide access to your board hardware resources such as memory, IO ports and PCI board configuration. See the README.TXT located in this directory for more information.
...\ATEasy\Drivers	ATEasy drivers directory. GXSW Driver and example are copied to this directory only if ATEasy is installed to your machine.
...\Windows\System32, or ...\Windows\SysWOW64 when running 64 bit Windows	Windows System directory. Contains the GXSW.DLL driver, HW driver shared files and some upgraded system components, such as the HTML help viewer, etc.

GXSW driver Files Description

The Setup program copies the GXSW driver, a panel executable; the GXSW help file, the README.TXT file, and driver samples. The following is a brief description of each installation file:

Driver and Virtual Panel

- GXSW.dll – Windows DLL for applications running under Windows (32 bit).
- GxXXXXPanel.exe – An instrument front panel program for all GXSW supported boards.

Programming Interface Files

The following GXSW interface files are used to support the various development tools:

- GXSW.h – header file for accessing the DLL functions using the C/C++ programming language. The header file is compatible with the following 32-bit development tools:
 - Microsoft Visual C++, Microsoft Visual C++ .NET
 - Borland C++
- GXSW.lib – Import library for GXSW.DLL (used when linking C/C++ application that uses GXSW.dll).
- GXSWBC.lib – Import library for GXSW.DLL (used when linking Borland C/C++ application that uses GXSW.dll).
- GXSW.pas – interface file to support Borland Pascal Borland Delphi.
- GXSW.bas – Supports Microsoft Visual Basic 4.0, 5.0 and 6.0.
- GXSW.vb – Supports Microsoft Visual Basic .NET.
- ATEasy Driver Files. These include a driver for each of the GXSW instruments:
 - GX6021.drv - ATEasy driver for GX6021 board.
 - GX6062.drv - ATEasy driver for GX6062 board.
 - GX6115.drv - ATEasy driver for GX6115 board.
 - GX6125.drv - ATEasy driver for GX6125 board.
 - GX6138.drv - ATEasy driver for GX6138 board.
 - GX6264.drv - ATEasy driver for GX6264 board.
 - GX6315.drv - ATEasy driver for GX6315 board.
 - GX6325.drv - ATEasy driver for GX6325 board.
 - GX6338.drv - ATEasy driver for GX6338 board.
 - GX6377.drv - ATEasy driver for GX6377 board.
 - GX6384.drv - ATEasy driver for GX6384 board.
 - GX6616.drv - ATEasy driver for GX6616 board.

Support for additional GXSW drivers may be included in the future.

- GXSW.llb – LabView library.

Documentation

- GXSW.chm – On-line version of the GX1110 User's Guide. The help file is provided in a Windows Compiled HTML help file (.chm). The file contains information about the all the switching boards, programming reference and panel operation.
- GXSWUG.pdf – On line, printable version of the all the switching boards User's Guide in Adobe Acrobat format. To view or print the file you must have the reader installed. If not, you can download the Adobe Acrobat reader (free) from <http://www.adobe.com>.
- ReadMe.txt – Contains important last minute information not available when the manual was printed. This text file covers topics such as a list of files required for installation, additional technical notes, and corrections to the GXSW manuals. You can view and/or print this file using the Windows NOTEPAD.EXE or other text file editors.

GXSW Example Programs

The sample programs include a C/C++ sample compiled with various development tools, Visual Basic example and an ATEasy example. Other examples may be available for other programming tools The examples are installed to the GXSW Examples subfolder.

Microsoft Visual C++ .NET example files:

- GxSwExampleC.cpp – Source file
- GxSwExampleC.ico – Icon file
- GxSwExampleC.rc – Resource file
- GxSwExampleC.vcproj – VC++ .NET project file
- GxSwExampleC.exe – Example executable

Microsoft Visual C++ 6.0 example files:

- GxSwExampleC.cpp – Source file
- GxSwExampleC.ico – Icon file
- GxSwExampleC.rc – Resource file
- GxSwExampleC.dsp – VC++ project file
- GxSwExampleC.exe – Example executable

Visual Basic .NET example files:

- GX6021.VB - GX6021 form.
- GX6021.RESX - GX6021 form binary data.
- GX6062.VB - GX6062 form.
- GX6062.RESX - GX6062 form binary data.
- GX6115.VB - GX6115 form.
- GX6115.RESX - GX6115 form binary data.
- GX6125.VB - GX6125 form.
- GX6125.RESX - GX6125 form binary data.
- GX6138.VB - GX6138 form.
- GX6138.RESX - GX6138 form binary data.

- GX6264.VB - GX6264 form.
- GX6264.RESX - GX6264 form binary data.
- GX6315.VB - GX6315 form.
- GX6315.RESX - GX6315 form binary data.
- GX6325.VB - GX6325 form.
- GX6325.RESX - GX6325 form binary data.
- GX6338.VB - GX6338 form.
- GX6338.RESX - GX6338 form binary data.
- GX6377.VB - GX6377 form.
- GX6377.RESX - GX6377 form binary data.
- GX6384.VB - GX6384 form.
- GX6384.RESX - GX6384 form binary data.
- GX6616.VB - GX6616 form.
- GX6616.RESX - GX6616 form binary data.
- GXSWEXAMPLE.VB - Example main form.
- GXSWEXAMPLE.RESX - Example main form binary data.
- GXSWEXAMPLEPUBLIC.VB - Example global module.
- GXSWEXAMPLEASSEMBLYINFO.VB - Project assembly file.
- GXSWEXAMPLEVB.VBPROJ - Project file
- GXSWEXAMPLEVB.EXE - Example executable.

Microsoft Visual Basic 6.0 example files:

- GX6021.FRM - GX6021 form.
- GX6021.FRX - GX6021 form binary data.
- GX6062.FRM - GX6062 form.
- GX6062.FRX - GX6062 form binary data.
- GX6115.FRM - GX6115 form.
- GX6115.FRX - GX6115 form binary data.
- GX6125.FRM - GX6125 form.
- GX6125.FRX - GX6125.FRX form binary data.
- GX6138.FRM - GX6138 form.
- GX6138.FRX - GX6138 form binary data.
- GX6264.FRM - GX6264 form.
- GX6264.FRX - GX6264 form binary data.
- GX6315.FRM - GX6315 form.
- GX6315.FRX - GX6315 form binary data.
- GX6325.FRM - GX6325 form.

- GX6325.FRX - GX6325 form binary data.
- GX6338.FRM - GX6338 form.
- GX6338.FRX - GX6338 form binary data.
- GX6377.FRM - GX6377 form.
- GX6377.FRX - GX6377 form binary data.
- GX6384.FRM - GX6384 form.
- GX6384.FRX - GX6384 form binary data.
- GX6616.FRM - GX6616 form.
- GX6616.FRX - GX6616 form binary data.
- GXSWEXAMPLE.FRM - Example main form.
- GXSWEXAMPLE.FRX - Example main form binary data.
- GXSWEXAMPLEPUBLIC.BAS - Example main module.
- GXSWEXAMPLEVB6.VBP - Project file

ATEasy driver and examples files (ATEasy Drivers directory):

- GX6021.prj - example project
- GX6021.sys - example system
- GX6021.prg - example program
- GX6062.prj - example project
- GX6062.sys - example system
- GX6062.prg - example program
- GX6115.prj - example project
- GX6115.sys - example system
- GX6115.prg - example program
- GX6125.prj - example project
- Gx6125.sys - example system
- GX6125.prg - example program
- GX6138.prg - example program
- GX6138.sys - example system
- GX6138.prj - example project
- GX6264.prj - example project
- GX6264.sys - example system
- GX6264.prg - example program
- GX6315.prj - example project
- GX6315.sys - example system
- GX6315.prg - example program
- GX6325.prj - example project

- GX6325.sys - example system
- GX6325.prg - example program
- GX6338.prg - example program
- GX6338.sys - example system
- GX6338.prj - example project
- GX6377.prg - example program
- GX6377.sys - example system
- GX6377.prj - example project
- GX6384.prg - example program
- GX6384.sys - example system
- GX6384.prj - example project
- GX6616.prg - example project
- GX6616.sys - example system
- GX6616.prg - example program

LabView example:

- GxSwExample.vi

Configuring Your PXI System using the PXI/PCI Explorer

To configure your PXI/PCI system using the **PXI/PCI Explorer** applet follow these steps:

1. Start the PXI/PCI Explorer applet. The applet can be start from the Windows Control Panel or from the Windows Start Menu, Marvin Test Solutions, HW, PXI/PCI Explorer.
2. Identify Chassis and Controllers. After the PXI/PCI Explorer is started, it will scan your system for changes and will display the current configuration. The PXI/PCI Explorer automatically detects systems that have Marvin Test Solutions controllers and chassis. In addition, the applet detects PXI-MXI-3/4 extenders in your system (manufactured by National Instruments). If your chassis is not shown in the explorer main window, use the Identify Chassis/Controller commands to identify your system. Chassis and Controller manufacturers should provide INI and driver files for their chassis and controllers which are used by these commands.
3. **Change chassis numbers, PXI devices Legacy Slot numbering and PXI devices Alias names.** These are optional steps and can be performed if you would like your chassis to have different numbers. Legacy slots numbers are used by older Marvin Test Solutions or VISA drivers. Alias names can provide a way to address a PXI device using a logical name (e.g. "DMM1"). For more information regarding slot numbers and alias names, see the **GxXXXXInitialize** and **GxXXXXInitializeVisa** functions.
4. Save your work. PXI Explorer saves the configuration to the following files located in the Windows folder: PXISYS.ini, PXIeSYS.ini and GxPxiSys.ini. Click on the Save button to save your changes. The PXI/Explorer will prompt you to save the changes if changes were made or detected (an asterisk sign ‘*’ in the caption indicated changes).

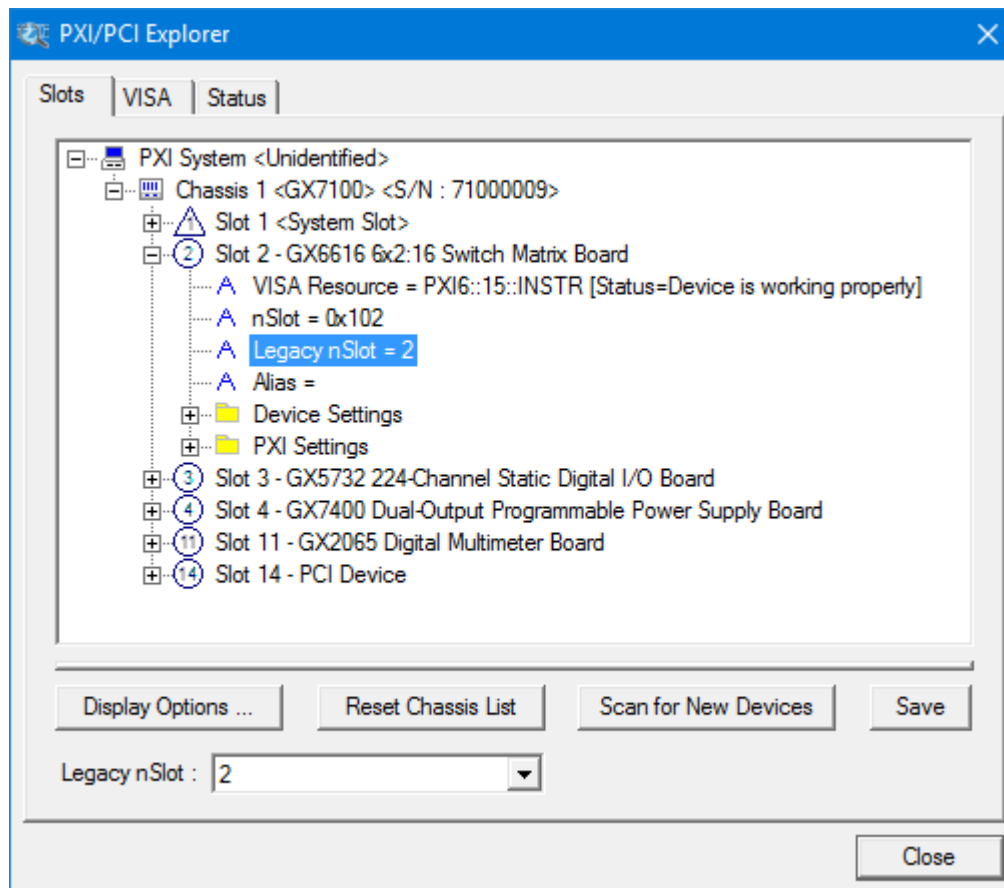


Figure 3-1: PXI/PCI Explorer

Board Installation

Before you Begin

- Install the GXSW driver as described in the prior section.
- Configure your PXI/PC system using **PXI/PCI Explorer** as described in the prior section.
- Verify that all the components listed in the packing list (see previous section in this chapter) are present.

Electric Static Discharge (ESD) Precautions

To reduce the risk of damage to the board, the following precautions should be observed:

- Leave the board in the anti-static bags until installation requires removal. The anti-static bag protects the board from harmful static electricity.
- Save the anti-static bag in case the board is removed from the computer in the future.
- Carefully unpack and install the board. Do not drop or handle the board roughly.
- Handle the board by the edges. Avoid contact with any components on the circuit board.



Caution – Do not insert or remove any board while the computer is on. Turn off the power from the PXI chassis before installation.

Installing a Board

Install the board as follows:

1. Install first the GXSW driver as described in the next section.
2. Turn off the PXI chassis and unplug the power cord.
3. Locate a PXI empty slot on the PXI chassis.
4. Place the module edges into the PXI chassis rails (top and bottom).
5. Carefully slide the PXI board to the rear of the chassis, make sure that the ejector handles are pushed out (as shown in Figure 3-2).

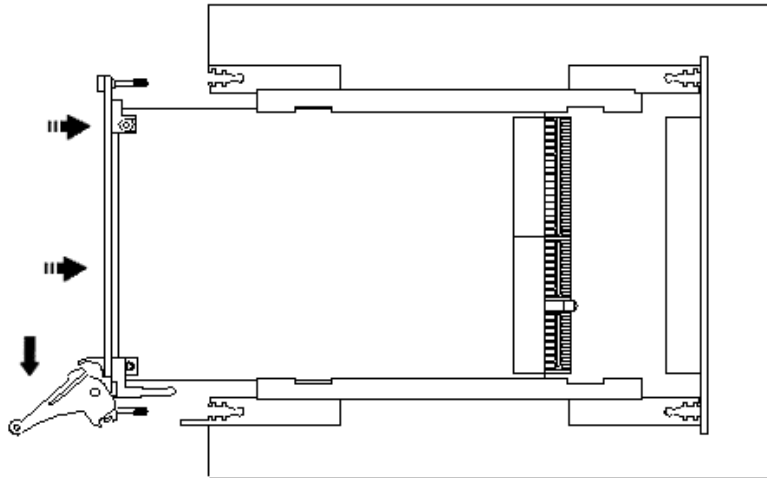


Figure 3-2: Ejector handles position during module insertion

6. After you feel resistance, push in the ejector handles as shown in Figure 3-3 to secure the module into the frame.

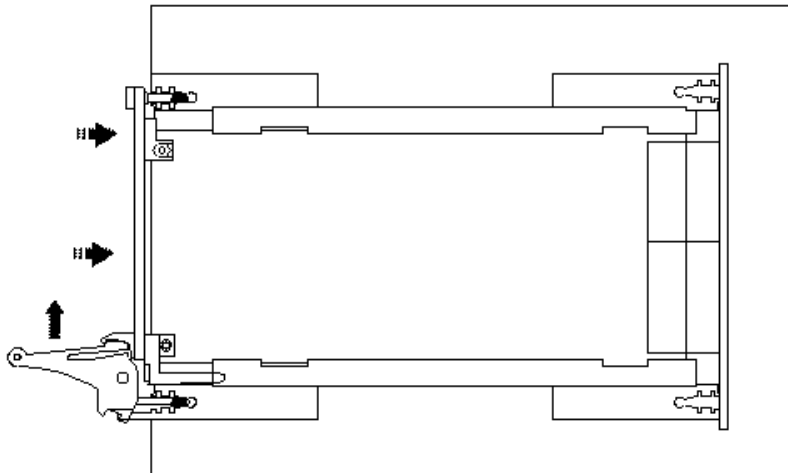


Figure 3-3: Ejector handles position after module insertion

7. Tighten the module's front panel to the chassis to secure the module in.
8. Connect any necessary cables to the board.
9. Plug the power cord in and turn on the PXI chassis.

Plug & Play Driver Installation

Plug & Play operating systems such as Windows notifies the user that a new board was found using the **New Hardware Found** wizard after restarting the system with the new board.

If another Marvin Test Solutions board software package was already installed, Windows will suggest using the driver information file: HW.INF. The file is located in your Program Files\Marvin Test Solutions\HW folder. Click **Next** to confirm and follow the instructions on the screen to complete the driver installation.

If the operating system was unable to find the driver (since the GXSW driver was not installed prior to the board installation), you may install the GXSW driver as described in the prior section, then click on the **Have Disk** button and browse to select the HW.INF file located in **C:\Program File\Marvin Test Solutions\HW**. On 64 bit systems the HW.INF file is located in **C:\Program File (x86)\Marvin Test Solutions\HW**.

If you are unable to locate the driver click **Cancel** to the found New Hardware wizard and exit the New Hardware Found Wizard, install the GXSW driver, reboot your computer and repeat this procedure.

The Windows Device Manager (open from the System applet from the Windows Control Panel) must display the proper board name before continuing to use the board software (no Yellow warning icon shown next to device). If the device is displayed with an error, you can select it and press delete and then press F5 to rescan the system again and to start the New Hardware Found wizard.

Removing a Board

Remove the board as follows:

1. Turn off the PXI chassis and unplug the power cord.
2. Locate a PXI slot on the PXI chassis.
3. Disconnect and remove any cables/connectors connected to the board.
4. Un-tighten the module's front panel screws to the chassis.
5. Push out the ejector handles and slide the PXI board away from the chassis.
6. Optionally – uninstall the GXSW driver.

Chapter 4 - Programming the Board

This chapter contains information about how to program the GXSW boards using the GXSW driver. The GXSW driver contains functions to initialize, reset, and control the GXSW instruments. A brief description of the functions, as well as how and when to use them, is included in this chapter. Chapter 5 and the specific instrument User's Guide contain a complete and detailed description of the available programming functions.

The driver supports many development tools. Using these tools with the driver is described in this chapter. In addition, the GXSW folder contains examples written for these development tools. Refer to Chapter 3 for a list of the available examples.

An example using the DLL driver with Microsoft Visual C++ is included at the end of this chapter. Since the driver functions and parameters are identical for all operating systems and development tools, the example can serve as a framework when using other programming languages, programming tools, and other driver types.

The GXSW driver

The GXSW driver is provided as 32-bit Windows DLL file: GXSW.DLL and a 64-bit Windows DLL GXSW64.DLL. The 32-bit DLL is used with 32-bit applications running under Windows while the 64-bit DLL with 64-bit applications. The DLLs use a device driver to access the board resources. The device driver HW.SYS or HW64.sys are installed by the setup program and is shared by other Marvin Test Solutions products (ATEasy, GXDIO, etc.).

The DLLs can be used with various development tools such as Microsoft Visual C++, Borland C++ Builder, Microsoft Visual Basic, Borland Pascal or Delphi, ATEasy, LabVIEW, and more. The following paragraphs describe how to create an application that uses the driver with various development tools. Refer to the paragraph describing the specific development tool for more information.

Programming Using C/C++ Tools

The following steps are required to use the GXSW driver with C/C++ development tools:

- Include the GXSW.H header file in the C/C++ source file that uses the GXSW function. This header file is used for all driver types. The file contains function prototypes and constant declarations which are used by the compiler for the application.
- Add the required .LIB file to the projects. This can be the imported library GXSW.LIB for Microsoft Visual C++ and GXSWBC.LIB for Borland C++. Windows based applications that explicitly load the DLL by calling the Windows **LoadLibrary** API should not include the .LIB file in the project.
- Add code to call the board function as required by the application.
- Build the project.
- Run, test, and debug the application.

Programming Using Visual Basic

To use the driver with Visual Basic 4.0, 5.0 or 6.0 (for 32-bit applications), the user must include the GXSW.BAS to the project. For Visual Basic .NET use the GXSW.VB.

The file can be loaded using *Add File* from the Visual Basic *File menu*. The GXSW.BAS/.VB contains function declarations for the DLL driver.

Programming Using Pascal/Delphi

To use the driver with Borland Pascal or Delphi, the user must include the GXSW.PAS to the project. The GXSW.PAS file contains a unit with function prototypes for the DLL functions. Include the GXSW unit in the **uses** statement before making calls to the GXSW functions.

Programming GXSW Boards Using ATEasy®

The ATEasy drivers supplied with the GXSW driver (GX6XXX.drv - for example GX6615.drv) uses the GXSW.DLL to program the board. In addition, each driver is supplied with an example that contains a program and a system file pre-configured with the ATEasy driver. Use the driver shortcut property page from the System Drivers sub-module to change the PCI slot number before attempting to run the example.

Using commands declared in the ATEasy driver are easier to use than using the DLL functions directly. The driver commands will also generate exception that allows the ATEasy application to trap errors without checking the status code returned by the DLL function after each function call.

The ATEasy driver contains commands that are similar to the DLL functions in name and parameters, with the following exceptions:

- The *nHandle* parameter is omitted. The driver handles this parameter automatically. ATEasy uses driver logical names instead i.e. MUX1, MUX2 for GX6264.
- The *nStatus* parameter was omitted. Use the Get Status commands instead of checking the status. After calling a DLL function the ATEasy driver will check the returned status and will call the error statement (in case of an error status) to generate exception that can be easily trapped by the application using the **OnError** module event or using the **try-catch** statement.

Some ATEasy drivers contain additional commands to permit easier access to the board features. For example, parameters for a function may be omitted by using a command item instead of typing the parameter value. The commands are self-documented. Their syntax is similar to English. In addition, you may generate the commands from the code editor context menu or by using the ATEasy's code completion feature instead of typing them directly.

Using the GXSW Driver Functions

The GXSW driver contains a set of functions for the GXSW boards. Function names that start with the **GxSW** prefix apply to all GXSW boards (i.e. **GxSWGetDriverSummary**). Other functions are specific to a board and start with the board model (e.g. Gx6264Initialize). The GXSW functions are designed with a consistent set of arguments and functionality. All boards have a function that initializes the GXSW driver for a specific board, resets the board, and displays the virtual panel. All the functions use handles to identify and reference a specific board and all functions return status and share the same functions to handle error codes.

Initialization, HW Slot Numbers and VISA Resource

The GXSW driver supports two device drivers HW and VISA which are used to initialize, identify and control the board. The user can use the **GxXXXXInitialize** (XXXX is the board model) to initialize the board 's driver using HW and **GxXXXXInitializeVisa** to initialize using VISA. The following describes the two different methods used to initialize:

1. Marvin Test Solutions' HW – This is the default device driver that is installed by the GXSW driver. To initialize and control the board using the HW use the **GxXXXXInitialize**(*nSlot*, *pnHandle*, *pnStatus*) function (XXXX is the board model for example, Gx6115Initialize). The function initializes the driver for the board at the specified PXI slot number (*nSlot*) and returns boards handle. The PXI/PCI Explorer applet in the Windows Control Panel displays the PXI slot assignments. You can specify the *nSlot* parameter in the following way:

- A combination of chassis number (chassis # x 256) with the chassis slot number, e.g. 0x102 for chassis 1 and slot 2. The chassis number can be set by the **PXI/PCI Explorer** applet.
- Legacy *nSlot* is used by earlier versions of HW/VISA. The slot number contains no chassis number and can be changed using the **PXI/PCI Explorer** applet: 2 in this example.

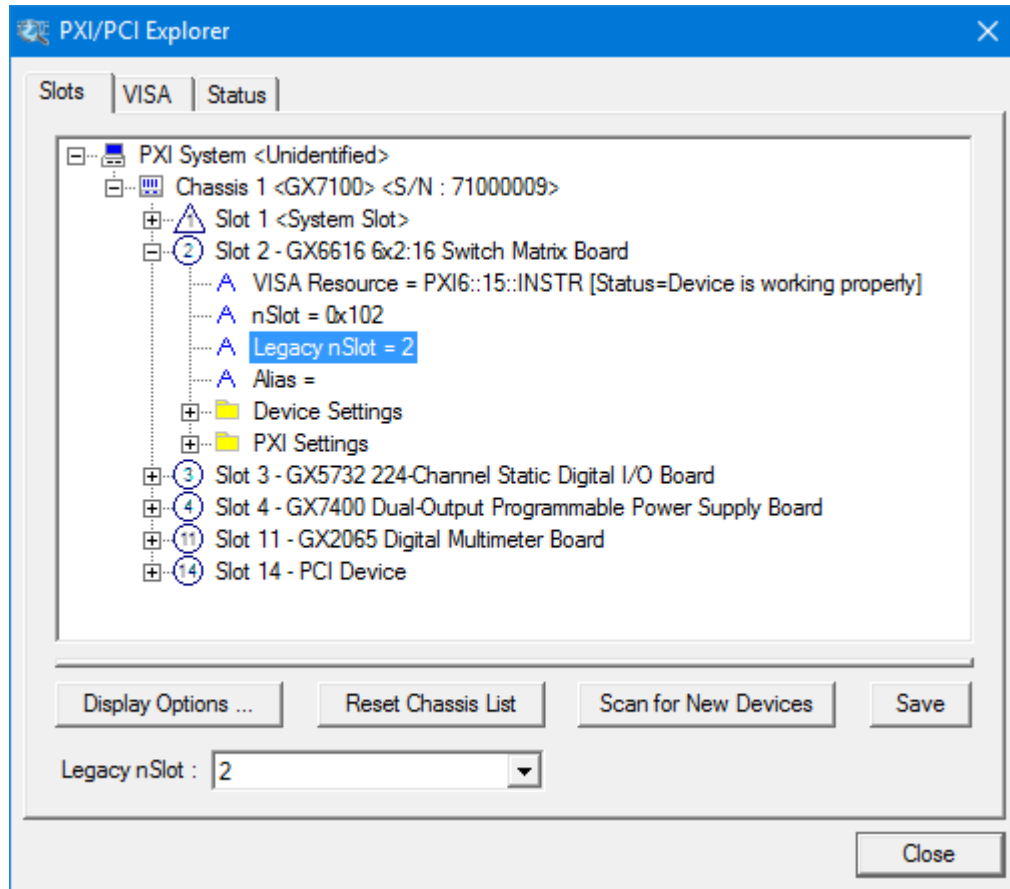


Figure 5-1: PXI/PCI Explorer

2. VISA – This is a third-party library usually supplied by National Instruments (NI-VISA). You must ensure that the VISA installed supports PXI and PCI devices (not all VISA providers supports PXI/PCI). GXSW setup installs a VISA compatible driver for the GXSW board in-order to be recognized by the VISA provider. Use the GXSW function **GxXXXXInitializeVisa** (szVisaResource, pnHandle, pnStatus) to initialize the driver’s board using VISA. The first argument szVisaResource is a string that is displayed by the VISA resource manager such as NI Measurement and Automation (NI_MAX). It is also displayed by Marvin Test Solutions PXI/PCI Explorer as shown in the prior figure. The VISA resource string can be specified in several ways as the following examples demonstrate:

- Using chassis, slot: “PXI0::CHASSIS1::SLOT5”
- Using the PCI Bus/Device combination: “PXI9::13::INSTR” (bus 9, device 9).
- Using the alias: for example “COUNTER1”. Use the PXI/PCI Explorer to set the device alias.

Information about VISA is available at <http://www.pxisa.org>.

Board Handle

The **GxXXXXInitialize** and the **GxXXXXInitializeVisa** functions return a handle that is required by other driver functions in order to program the board. This handle is usually saved in the program as a global variable for later use when calling other functions. The initialize function does not change the state of the board or its settings.

Reset

The Reset function causes the driver to change all settings to their default state. The application software issue a Reset after the initializing the board, but a Reset can be issued at any time. All GXSW boards have the **GxXXXXReset**(*nHandle*, *nStatus*) function. See the Function Reference section for more information regarding specific board functionality.

Error Handling

All GXSW functions pass a fail or success status – *pnStatus* – in the last parameter. A successful function call passes zero in the status parameter upon return. If the status is non-zero, then the function call fails. This parameter can be later used for error handling. When the status is error, the program can call the **GxSWGetErrorString** function to return a string representing the error. The **GxSWGetErrorString** reference contains possible error numbers and their associated error strings.

Driver Version

The **GxSWGetDriverSummary** function can be used to return the current GXSW driver version. It can be used to differentiate between the driver versions. See the Function Reference section for more information.

Panel

Calling the **GxXXXXPanel** will display the instrument's front panel dialog window. The panel can be used to initialize and control the board interactively. The panel function may be used by the application to allow the user to directly interact with the board.

The **GxXXXXPanel** function is also used by the GXXXXXPANEL.EXE panel program that is supplied with this package and provides a stand-alone Windows application that displays the instrument panel.

Distributing the Driver

Once the application is developed, the driver files (GXSW.DLL, GXSW64.DLL and the HW device driver files located in the HW folder) can be shipped with the application. Typically, the GXSW.DLL and GXSW64.DLL should be copied to the Windows System (32/64 bit directories). The HW device driver files should be installed using a special setup program HWSETUP.EXE that is provided with GXSW driver files. Alternatively, you can provide the GXSW.EXE setup program to be installed along with the board. The setup program can be invoked in silent mode (with /s) to install the driver using the command line specified settings with no user interface. Marvin Test Solutions provides permission to re-distribute the driver software providing it is in support of an application using the target hardware.

Sample Programs

The following example demonstrates how to program the board using the C programming language under Windows. The example shows how to close or open a relay.

To run, Enter the following command line:

GXSWEX <Model> <PciSlot> <specific_command>

Where:

<Model>	Enter one of the following: 6264, 6315, 6338, or 6616
<PciSlot>	Pci slot number where the board is installed. For example: 3
<specific_command>	Depends on the model, see comments in the beginning of the sample listing.

Sample Program Listing

```

/*****

FILE      : GxSwExampleC.cpp

PURPOSE   : WIN32/LINUX example program for GX2200 boards
            using the GxSw driver.

CREATED   : Mar 2002

COPYRIGHT : Copyright 2002-2011 Marvin Test Solutions-, Inc.

COMMENTS  :

To compile the example:

1. Microsoft VC++
   Load GxSwExampleC.dsp, .vcproj or .mak, depends on
   the VC++ version from the Project\File/Open... menu
   Select Project/Rebuild all from the menu

2. Borland C++ Builder
   Load GxSwExampleC.bpr from the Project/Open
   Project... menu
   Select Project/Build all from the menu

3. Linux (GCC for CPP and Make must be available)
   make -fGxSwExampleC.mk [CFG=Release[64] | Debug[64]] [rebuild |
   clean]

*****/
#ifdef __GNUC__
#include "windows.h"
#endif
#include "GxSw.h"
#include <stdio.h>

```

```

#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#if defined(__BORLANDC__)
#pragma hdrstop
#include <condefs.h>
USELIB("GxSwBC.lib");
USERC("GxSwExampleC.rc");
#endif

//*****
//      DisplayMsg
//*****
void DispMsg(PCSTR lpszMsg)
{
#ifdef __GNUC__
    MessageBeep(0);
    MessageBox(0, lpszMsg, "GxSw example program", MB_OK);
#else
    printf("\r\nGxSw example program: %s\r\n", lpszMsg);
#endif
    return;
}

//*****
//      __strupr
//*****
char * __strupr(char * sz)
{
    int i;

    for (i=0; sz[i]; i++)
        sz[i]=toupper(sz[i]);
    return sz;
}

//*****
//      DispUsage
//*****
void DispUsage(void)
{
    DispMsg(
        "\r\nThis example shows how to \
        Close or Open GXSW board relays\r\n"

        "Usage:\r\n"
        "GxSwExample <model> <slot|address> <command>"
        "\r\n\r\nWhere:\r\n"
        "<model> can be"
        "\t6021 | 6062 | 6115 | 6125 | 6138 | 6264 | 6315 | 6325 | 6338 \
        \t!6377 | 6384 | 6616\r\n"
        "<slot> - Under Windows: PCI/PXI slot number as shown by the PXI \

```

```

explorer\r\n"
"<address> - Under Linux: Bus/device as displayed with lspci \
utility\r\n"
"<command> depends on the model:\r\n"
"\tSUM=Print board summary (applies to all models)\r\n"
"\t6021: C | O Group# Relay#\r\n"
"\t\tWhere: C=Close, O=Open, Group#=0-3 for group number,\r\n"
"\t\tChannel#=0-5 relay number\r\n\r\n"

"\t6062: C | O Group# Relay#\r\n"
"\t\tWhere: C=Close, O=Open, Group#=0-11 for group number,\r\n"
"\t\tChannel#=0-5 relay number\r\n\r\n"

"\t6115: C | O Type# Relay#\r\n"
"\t\tWhere: C=Close, O=Open, Type#=0-1, 0=relay, 1=driver relay\r\n"
"\t\tRelay#=1-15 for relay and 1-3 for driver relay number\r\n"

"\t6125: C | O Relay#\r\n"
"\t\tWhere: C=Close, O=Open, Relay#=1-25\r\n"

"\t6138: C | O Relay#\r\n"
"\t\tWhere: C=Close, O=Open, Relay#=1-38\r\n"

"\t6264: C | O Group# Relay#\r\n"
"\t\tWhere: C=Close, O=Open, Group#=0-7 for group number,\r\n"
"\t\tChannel#=0-7 relay number(single ended mode)\r\n\r\n"

"\t6315: C | O Type# Relay#\r\n"
"\t\tWhere: C=Close, O=Open, Type#=0-1, 0=relay, 1=driver relay\r\n"
"\t\tRelay#=1-45 for relay and 1-9 for driver relay number\r\n"

"\t6325: C | O Relay#\r\n"
"\t\tWhere: C=Close, O=Open, Relay#=1-75\r\n"

"\t6338: C | O Relay#\r\n"
"\t\tWhere: C=Close, O=Open, Relay#=1-114\r\n"

"\t6377: C | O Type# RelayGroup# Row# Column#\r\n"
"\t\tWhere: C=Close, O=Open\r\n"
"\t\tType#=0-Hi Current, 1-Form A, 2-Form C, 3-Matrix\r\n"
"\t\tRelayGroup#= if Type= 0-2 RelayGroup#=1-5 (Type=0),\r\n"
"\t\t1-4 (Type=1), 1-4 (Type=2)\r\n"
"\t\tIf Type#=3(Matrix) RelayGroup#= 0-1 for group number\r\n"
"\t\tRow#=0-1, column#=0-15\r\n"

"\t6384: C | O Group# Row# Column#\r\n"
"\t\tWhere: C=Close, O=Open, Group#=0-1 for group number,\r\n"
"\t\tRow#=0-5, column#=0-31\r\n"

"\t6616: C | O Group# Row# Column#\r\n"
"\t\tWhere: C=Close, O=Open, Group#=0-5 for group number,\r\n"
"\t\tRow#=0-1, column#=0-15\r\n"

```

```

        "\r\nThe example should be run from the Windows command prompt/Linux \
        terminal.\r\n"
        "\tExample:\r\n"
        "\tGxSwExample 6338 0x105 C 5 (under Windows, slot 5 at chasisis 1)\r\n"
        "\tGxSwExample 6338 0x60C C 5 ((under Linux, bus 6 device 12)\r\n"
        "\tCloses relay number 5 on the GX6338 reside at slot 1\r\n"
    );
    exit(1);
}

//*****
//    CheckStatus
//*****
void CheckStatus(SHORT nStatus)
{
    CHAR    sz[256];

    if (!nStatus) return;
    GxSWGetErrorString(nStatus, sz, sizeof sz, &nStatus);
    DispMsg(sz);
    DispMsg("Aborting the program...");
    exit(nStatus);
}

//*****
//    MAIN
//
// This main function receives four to five parameters:
//
// GXSW model (e.g. 6021=GX6021, 6062=GX6062, 6264=GX6264,
//             6315=GX6315, 6315=GX6315, 6338=GX6138,
//             6325=GX6325, 6338=GX6338, 6616=GX6616)
// GXSW board PXI slot/address number(e.g. 0x105 for chassis 1 slot 5)
// GXSW operation(e.g. O=Open relay, C=Close relay, SUM=Print board summary)
//
//     for GXSW GX6021 group #(0-3) , channel #(0-5)
//     for GXSW GX6062 group #(0-11) , channel #(0-5)
//     for GXSW GX6115 type #(0-1) relay #(1-15)
//     for GXSW GX6125 relay #(1-25)
//     for GXSW GX6138 relay #(1-38)
//     for GXSW GX6264 group #(0-7) , channel #(0-7)
//     for GXSW GX6315 type #(0-1) relay #(1-15)
//     for GXSW GX6325 relay #(1-75)
//     for GXSW GX6338 relay #(1-114)
//     for GXSW GX6377 type #(0-1)
//     for GXSW GX6384 group #(0-1), row # (0-5), column #(0-31)
//     for GXSW GX6616 group #(0-5), row # (0-1), column #(0-15)
//
//*****
int main(int argc, char **argv)
{
    short    nBoardType;        // Board Type
    short    nSlotNum;         // Board PXI slot number

```

```

char * pszOperation;      // Board Operation (C/V/SUM)
short  nGroup;           // Group number
short  nRow;             // Row number
short  nColumn;         // Column number
short  nChannel;        // Channel number
short  nHandle;         // Board handle
short  nStatus;         // Returned status
short  nType;           // Relay type number
short  nRelayGroup;     // Relay or Group number
char   sz[512];         // board summary

// ** Check number of arguments rcvd
if (argc<4 || argc > 8) DispUsage();

// ** Parse command line parameters
nBoardType=(SHORT)strtol(++argv, NULL, 0);
nSlotNum=(SHORT)strtol(++argv, NULL, 0);

// ** Process operation (Current/Voltage)
pszOperation=++argv;
__strupr(pszOperation);
if (strcmp(pszOperation, "C") && strcmp(pszOperation, "V") &&
    strcmp(pszOperation, "SUM"))
    DispUsage();

// ** Get Group, channel, initialize, and perform operation on GXSW card
switch (nBoardType)
{
    case 6021:
        Gx6021Initialize(nSlotNum, &nHandle, &nStatus);
        CheckStatus(nStatus);

        if (!strcmp(pszOperation, "SUM"))
        {
            // print board summary
            Gx6021GetBoardSummary(nHandle, sz, sizeof sz, &nStatus);
            CheckStatus(nStatus);
            printf("Board Summary: %s.\n", sz);
            return 0;
        }

        nGroup=(SHORT)strtol(++argv, NULL, 0);
        nChannel=(SHORT)strtol(++argv, NULL, 0);
        switch (*pszOperation)
        {
            case 'C':
                Gx6021Close(nHandle, nGroup, nChannel, &nStatus);
                CheckStatus(nStatus);
                DispMsg("You have closed a switch");
                break;
            case 'O':
                Gx6021Open(nHandle, nGroup, nChannel, &nStatus);
                CheckStatus(nStatus);
                DispMsg("You have opened a switch");
                break;
        }
    }
}

```

```

    }
    break;

case 6062:

    Gx6062Initialize(nSlotNum, &nHandle, &nStatus);
    CheckStatus(nStatus);
    if (!strcmp(pszOperation, "SUM"))
    { // print board summary
        Gx6062GetBoardSummary(nHandle, sz, sizeof sz, &nStatus);
        CheckStatus(nStatus);
        printf("Board Summary: %s.\n", sz);
        return 0;
    }

    nGroup=(SHORT)strtol(++argv, NULL, 0);
    nChannel=(SHORT)strtol(++argv, NULL, 0);
    switch (*pszOperation)
    { case 'C':
        Gx6062Close(nHandle, nGroup, nChannel, &nStatus);
        CheckStatus(nStatus);
        DispMsg("You have closed a switch");
        break;
      case 'O':
        Gx6062Open(nHandle, nGroup, nChannel, &nStatus);
        CheckStatus(nStatus);
        DispMsg("You have opened a switch");
    }
    break;

case 6115:

    Gx6115Initialize(nSlotNum, &nHandle, &nStatus);
    CheckStatus(nStatus);

    if (!strcmp(pszOperation, "SUM"))
    { // print board summary
        Gx6115GetBoardSummary(nHandle, sz, sizeof sz, &nStatus);
        CheckStatus(nStatus);
        printf("Board Summary: %s.\n", sz);
        return 0;
    }

    nType=(SHORT)strtol(++argv, NULL, 0);
    nChannel=(SHORT)strtol(++argv, NULL, 0);
    switch (*pszOperation)
    { case 'C':
        Gx6115Close(nHandle, nType, nChannel, &nStatus);
        CheckStatus(nStatus);
        DispMsg("You have closed a switch");
        break;
      case 'O':
        Gx6115Open(nHandle, nType, nChannel, &nStatus);
    }

```



```

        CheckStatus(nStatus);
        DispMsg("You have opened a switch");
    }
    break;

case 6125:

    Gx6125Initialize(nSlotNum, &nHandle, &nStatus);
    CheckStatus(nStatus);
    if (!strcmp(pszOperation, "SUM"))
    { // print board summary
        Gx6125GetBoardSummary(nHandle, sz, sizeof sz, &nStatus);
        CheckStatus(nStatus);
        printf("Board Summary: %s.\n", sz);
        return 0;
    }
    nChannel=(SHORT)strtol(++argv, NULL, 0);
    switch (*pszOperation)
    { case 'C':
        Gx6125Close(nHandle, nChannel, &nStatus);
        CheckStatus(nStatus);
        DispMsg("You have closed a switch");
        break;
      case 'O':
        Gx6125Open(nHandle, nChannel, &nStatus);
        CheckStatus(nStatus);
        DispMsg("You have opened a switch");
    }
    break;

case 6138:

    Gx6138Initialize(nSlotNum, &nHandle, &nStatus);
    CheckStatus(nStatus);
    if (!strcmp(pszOperation, "SUM"))
    { // print board summary
        Gx6138GetBoardSummary(nHandle, sz, sizeof sz, &nStatus);
        CheckStatus(nStatus);
        printf("Board Summary: %s.\n", sz);
        return 0;
    }
    nChannel=(SHORT)strtol(++argv, NULL, 0);
    switch (*pszOperation)
    { case 'C':
        Gx6138Close(nHandle, nChannel, &nStatus);
        CheckStatus(nStatus);
        DispMsg("You have closed a switch");
        break;
      case 'O':
        Gx6138Open(nHandle, nChannel, &nStatus);
        CheckStatus(nStatus);
        DispMsg("You have opened a switch");
    }
}

```

```

        break;

case 6264:
    Gx6264Initialize(nSlotNum, &nHandle, &nStatus);
    CheckStatus(nStatus);
    if (!strcmp(pszOperation, "SUM"))
    { // print board summary
        Gx6264GetBoardSummary(nHandle, sz, sizeof sz, &nStatus);
        CheckStatus(nStatus);
        printf("Board Summary: %s.\n", sz);
        return 0;
    }

    if (argc != 6) DispUsage();
    nGroup=(SHORT)strtol(++argv, NULL, 0);
    nChannel=(SHORT)strtol(++argv, NULL, 0);
    switch (*pszOperation)
    { case 'C':
        Gx6264ConnectChannel(nHandle, nGroup, nChannel, &nStatus);
        CheckStatus(nStatus);
        DispMsg("You have closed a switch");
        break;
      case 'O':
        Gx6264ResetGroup(nHandle, nGroup, &nStatus);
        CheckStatus(nStatus);
        DispMsg("You have opened a switch");
    }
    break;

case 6315:

    Gx6315Initialize(nSlotNum, &nHandle, &nStatus);
    CheckStatus(nStatus);

    if (!strcmp(pszOperation, "SUM"))
    { // print board summary
        Gx6315GetBoardSummary(nHandle, sz, sizeof sz, &nStatus);
        CheckStatus(nStatus);
        printf("Board Summary: %s.\n", sz);
        return 0;
    }
    nType=(SHORT)strtol(++argv, NULL, 0);
    nChannel=(SHORT)strtol(++argv, NULL, 0);
    switch (*pszOperation)
    { case 'C':
        Gx6315Close(nHandle, nType, nChannel, &nStatus);
        CheckStatus(nStatus);
        DispMsg("You have closed a switch");
        break;
      case 'O':
        Gx6315Open(nHandle, nType, nChannel, &nStatus);
        CheckStatus(nStatus);
        DispMsg("You have opened a switch");
    }

```

```

    }
    break;

case 6325:

    Gx6325Initialize(nSlotNum, &nHandle, &nStatus);
    CheckStatus(nStatus);
    if (!strcmp(pszOperation, "SUM"))
    { // print board summary
        Gx6325GetBoardSummary(nHandle, sz, sizeof sz, &nStatus);
        CheckStatus(nStatus);
        printf("Board Summary: %s.\n", sz);
        return 0;
    }
    nChannel=(SHORT)strtol(++argv, NULL, 0);
    switch (*pszOperation)
    { case 'C':
        Gx6325Close(nHandle, nChannel, &nStatus);
        CheckStatus(nStatus);
        DispMsg("You have closed a switch");
        break;
      case 'O':
        Gx6325Open(nHandle, nChannel, &nStatus);
        CheckStatus(nStatus);
        DispMsg("You have opened a switch");
    }
    break;

case 6338:

    Gx6338Initialize(nSlotNum, &nHandle, &nStatus);
    CheckStatus(nStatus);
    if (!strcmp(pszOperation, "SUM"))
    { // print board summary
        Gx6338GetBoardSummary(nHandle, sz, sizeof sz, &nStatus);
        CheckStatus(nStatus);
        printf("Board Summary: %s.\n", sz);
        return 0;
    }
    nChannel=(SHORT)strtol(++argv, NULL, 0);
    switch (*pszOperation)
    { case 'C':
        Gx6338Close(nHandle, nChannel, &nStatus);
        CheckStatus(nStatus);
        DispMsg("You have closed a switch");
        break;
      case 'O':
        Gx6338Open(nHandle, nChannel, &nStatus);
        CheckStatus(nStatus);
        DispMsg("You have opened a switch");
    }
    break;

case 6377:

```

```

Gx6377Initialize(nSlotNum, &nHandle, &nStatus);
CheckStatus(nStatus);
if (!strcmp(pszOperation, "SUM"))
{ // print board summary
  Gx6377GetBoardSummary(nHandle, sz, sizeof sz, &nStatus);
  CheckStatus(nStatus);
  printf("Board Summary: %s.\n", sz);
  return 0;
}

if (argc < 6) DispUsage();
nType=(SHORT)strtol(++argv, NULL, 0);
nRelayGroup=(SHORT)strtol(++argv, NULL, 0);
if (nType==0 || nType==1 || nType==2)
{ switch (*pszOperation)
  { case 'C':
    Gx6377RelayClose(nHandle, nType, nRelayGroup, &nStatus);
    CheckStatus(nStatus);
    DispMsg("You have closed a switch");
    break;
    case 'O':
    Gx6377RelayOpen(nHandle, nType, nRelayGroup, &nStatus);
    CheckStatus(nStatus);
    DispMsg("You have opened a switch");
  }
}
else if (nType==3)
{ if (argc != 8) DispUsage();
  nRow=(SHORT)strtol(++argv, NULL, 0);
  nColumn=(SHORT)strtol(++argv, NULL, 0);
  switch (*pszOperation)
  { case 'C':
    Gx6377MatrixClose(nHandle, nRelayGroup, nRow, nColumn, &nStatus);
    CheckStatus(nStatus);
    DispMsg("You have closed a switch");
    break;
    case 'O':
    Gx6377MatrixOpen(nHandle, nRelayGroup, nRow, nColumn, &nStatus);
    CheckStatus(nStatus);
    DispMsg("You have opened a switch");
  }
}
break;

case 6384:

Gx6384Initialize(nSlotNum, &nHandle, &nStatus);
CheckStatus(nStatus);
if (!strcmp(pszOperation, "SUM"))
{ // print board summary
  Gx6384GetBoardSummary(nHandle, sz, sizeof sz, &nStatus);
  CheckStatus(nStatus);

```

```

        printf("Board Summary: %s.\n", sz);
        return 0;
    }
    if (argc != 7) DispUsage();
    nGroup=(SHORT)strtol(++argv, NULL, 0);
    nRow=(SHORT)strtol(++argv, NULL, 0);
    nColumn=(SHORT)strtol(++argv, NULL, 0);
    switch (*pszOperation)
    {
        case 'C':
            Gx6384Close(nHandle, nGroup, nRow, nColumn, &nStatus);
            CheckStatus(nStatus);
            DispMsg("You have closed a switch");
            break;
        case 'O':
            Gx6384Open(nHandle, nGroup, nRow, nColumn, &nStatus);
            CheckStatus(nStatus);
            DispMsg("You have opened a switch");
    }
    break;

case 6616:
    Gx6616Initialize(nSlotNum, &nHandle, &nStatus);
    CheckStatus(nStatus);
    if (!strcmp(pszOperation, "SUM"))
    {
        // print board summary
        Gx6616GetBoardSummary(nHandle, sz, sizeof sz, &nStatus);
        CheckStatus(nStatus);
        printf("Board Summary: %s.\n", sz);
        return 0;
    }
    if (argc != 7) DispUsage();

    nGroup=(SHORT)strtol(++argv, NULL, 0);
    nRow=(SHORT)strtol(++argv, NULL, 0);
    nColumn=(SHORT)strtol(++argv, NULL, 0);
    switch (*pszOperation)
    {
        case 'C':
            Gx6616Close(nHandle, nGroup, nRow, nColumn, &nStatus);
            CheckStatus(nStatus);
            DispMsg("You have closed a switch");
            break;
        case 'O':
            Gx6616Open(nHandle, nGroup, nRow, nColumn, &nStatus);
            CheckStatus(nStatus);
            DispMsg("You have opened a switch");
    }
    break;

default:
    DispUsage();
}
return 0;
}

```

```
//*****  
//      End Of File  
//*****
```

Chapter 5 - Functions Reference

Introduction

The Functions Reference chapter describes the functions that are applicable to all switching instruments supported by the GXSW software, for a specific instrument function reference refer to the appropriate instrument user's guide function reference chapter.

The chapter is organized in alphabetical order. Each function description contains the function name, purpose, syntax, parameters description and type followed by a Comments, an Example (written in C), and a See Also sections.

All function parameters follow the same rules:

- Strings are ASCIIZ (null or zero character terminated).
- The first parameter of most functions is *nHandle* (16-bit integer). This parameter is required for operating the board and it returned by the specific **GxXXXXInitialize** or **GxXXXXInitializeVisa** functions of the board. The *nHandle* is used to identify the board when calling other driver functions.
- All functions return a status with the last parameter named *pnStatus*. The *pnStatus* is zero if the function was successful, or less than a zero on error. The description of the error is available using the **GxSWGetErrorString** function or by using a predefined constant, defined in the driver interface files: GXSW.H, GXSW.BAS, GXSW.PAS or the driver specific ATEasy driver (e.g. GX6338.DRV).
- Parameter name are prefixed as follows:

Prefix	Type	Example
a	Array, prefix this before the simple type.	<i>anArray</i> (Array of Short)
n	Short (signed 16-bit)	<i>nMode</i>
d	Double - 8 bytes floating point	<i>dReading</i>
dw	Double word (unsigned 32-bit)	<i>dwTimeout</i>
hwnd	Window handle (32-bit integer).	<i>hwndPanel</i>
l	Long (signed 32-bit)	<i>lBits</i>
p	Pointer. Usually used to return a value. Prefix this before the simple type.	<i>pnStatus</i>
sz	Null (zero value character) terminated string	<i>szMsg</i>
w	Unsigned short (unsigned 16-bit)	<i>wParam</i>

Table 5-1: Parameter Prefixes

GXSW Functions

The following list is a summary of functions available for all GXSW switching instruments:

Driver Functions	Description
GxSWGetDriverSummary	Returns the driver description string and version number.
GxSWGetErrorString	Returns the error string as specified by the error number.

GxSWGetDriverSummary

Purpose

Returns the driver description string and version number.

Syntax

GxSWGetDriverSummary (*pszSummary*, *nSummaryMaxLen*, *pdwVersion*, *pnStatus*)

Parameters

Name	Type	Comments
<i>pszSummary</i>	LPSTR	Buffer to receive the summary string.
<i>nSummaryMaxLen</i>	SHORT	Buffer size passed by pszSummary.
<i>pdwVersion</i>	LPDWORD	Returned version number. The high 16-bits contains the major version while the lower 16-bits contains the minor version number.
<i>pnStatus</i>	LPSHORT	Returned status: 0 on success, negative number on failure.

Comments

The returned string is similar to the following: "GXSW driver for GX6021, GX6062, GX6264, GX6315, GX6325, GX6338 and GX6616, Version 2.41, Copyright(c) 2005, Marvin Test Solutions, Inc.".

Example

The following example prints the driver version:

```
CHAR    sz[128];
DWORD   dwVersion;
SHORT   nStatus;
...
GxSWGetDriverSummary(sz, sizeof sz, &dwVersion, &nStatus);
printf("GXSW version %d.%d", (INT)(dwVersion>>16), (INT) dwVersion &0xFFFF);
```


GxSWGetErrorString

Purpose

Returns the error string associated with the specified error number.

Syntax

GxSWGetErrorString (*nError*, *pszMsg*, *nErrorMaxLen*, *pnStatus*)

Parameters

Name	Type	Comments
nError	SHORT	Error number as returned by the <i>pnStatus</i> of any GXSW function. See table below for possible error numbers values. The error number should be a negative number, otherwise the function returns the “No error has occurred” string.
pszMsg	LPSTR	Buffer containing the returned error string (null terminated string).
nErrorMaxLen	SHORT	Size of the buffer <i>pszMsg</i> .
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Comments

The function returns the error string associated with the *nError* as returned from other driver functions.

This function returns error value or 0 on success.

The following table displays the possible error values; not all errors apply to this board type:

Resource Errors

- 1 Board does not exist in this slot
- 2 Unable to open the HW device/Service
- 3 Different board exist in the specified PCI slot
- 4 PCI slot not configured properly. You may configure it by using the **PCIExplorer** from the Control Panel
- 5 Unable to register the PCI device
- 6 Unable to allocate system resource or memory for the PCI device
- 7 Too many boards
- 8 Unable to create panel
- 9 Unable to create a Windows timer

Parameter Errors

- 20 Invalid parameter
- 21 Invalid PCI slot number
- 22 Invalid board handle
- 23 Invalid channel number
- 24 Invalid bus number
- 25 Invalid mode
- 26 Invalid group number
- 27 Invalid string length

Board Errors/Warnings

- 50 BIT error: Adapter not connected
- 51 BIT error: Comparator Error
- 52 BIT error: Unable to open/close a relay in Group
- 63 BIT error: Unable to close Group %c1(%d), or short to ground of one or all relays
- 54 BIT error: Error occurred while switching a relay in Group
- 55 The board successfully passed the BIT.\nOnly one BIT adapter was found and tested

Board Errors/Warnings

- 60 Error: connection will create a closed loop
- 61 Error: specified relay number is out of range
- 62 Error: specified number of relays is out of range
- 63 Error: specified relays cycles limit is out of range
- 64 Error: specified relay cycles array size is out of range
- 65 Error: EPROM communication generated timeout
- 66 Error: Trying to close more relays then allowed by this board type at any given time.

Board specific parameter error

- 80 Invalid configuration
- 81 Invalid channel number
- 82 Illegal bus number
- 83 Illegal group number
- 84 Illegal row number
- 85 Illegal column number
- 86 Invalid Relay number
- 87 Invalid Daisy Chain Mode
- 88 Invalid relay type
- 89 Invalid Group Mode
- 90 Invalid Section number

Miscellaneous Errors

- 99 Invalid or unknown error number

Example

The following example initializes the board at slot 3. If the initialization failed the following error string is printed:

```
CHAR    sz[256];
SHORT   nStatus, nHandle;

GX6338Initialize(3, &Handle, &Status);
if (nStatus<0)
{   GxSWGGetErrorString(nStatus, sz, sizeof sz, &nStatus);
    printf(sz);    // print the error string return;
}
```


Index

A

ADC.....7
 ATE7, 16, 17
 ATEasy33, 34, 35, 43, 44

B

BIT.....7
 Board Handle.....46
 Borland34, 43, 44
 Borland-Delphi44
 Break-Before-Make12, 13

C

C/C++34, 35, 43
 C++43
 CO7, 16
 Connectors.....42
 Copyright.....i
 Corrupt files.....32

D

Delphi34, 43, 44
 Disclaimer.....i
 Distributing.....46
 DMM28
 DPDT.....7, 16
 Driver
 Directory.....33
 Files33
 Driver Version46
 Dry-Reed-Relay.....18

E

Electromechanical-Relays17
 EMF.....7, 19
 Error Handling.....46
 Error-Handling46
 Example.....35

F

FET.....16
 Form C.....16, 24

G

GX60216
 GX60626
 GX61156
 GX61256
 GX61386
 GX62646, 13, 25, 31
 GX63156, 12, 24, 25, 31
 GX63256
 GX63386, 24, 25
 GX63776
 GX63846
 GX6616.....6, 14, 26
 GXSW5
 GXSW32
 GXSW
 Help-File-Description35
 GXSW
 Driver-Description43
 GXSW
 Supported-Development-Tools.....43
 GXSW
 Header-file43
 GXSW.BAS.....43
 GXSW.DLL.....34, 43, 44
 GXSW.EXE.....32, 46
 GXSW.H43
 GXSW.LIB43
 GXSW.PAS44
 GXSWBC.LIB.....43
 GxSWGetDriverSummary.....44, 46, 60
 GxSWGetErrorString46, 61

GXSWPANEL.EXE.....	46
GxSwReset.....	46
GxXXXXInitialize.....	39, 44, 46, 59
GxXXXXInitializeVisa.....	39, 44, 45
GxXXXXPanel.....	34, 46
H	
handle.....	11, 19, 24
Handle.....	40, 41, 44, 46
HW.....	32, 33, 42, 43, 46
I	
If You Need Help.....	i
Installation.....	33
Installation Folders.....	33
Installation:.....	40, 42
ITA.....	7
L	
LabView.....	34, 38
Listing.....	47
M	
Make-Before-Break.....	12
Matrix-Topology.....	26
Multiplexing.....	24
Multiplex-Topology.....	12, 24
N	
NC.....	7, 15, 24
nHandle.....	44, 46, 59
NO.....	7, 15, 24
O	
OnError	44
P	
Panel.....	32, 34, 44, 46
Pascal.....	34, 43, 44
PCB.....	7, 23
PCI.....	33
Plug & Play.....	42
<i>pnStatus</i>	46
Program-File-Descriptions.....	34

Programming	
Borland-Delphi.....	44
Error-Handling.....	46
Panel-Program.....	46
Programming the Board.....	43
PXI.....	31, 39, 40, 41, 42, 44, 45
PXI/PCI Explorer.....	39, 40, 44, 45
R	
README.TXT.....	33, 34
Readme-File.....	35
Relay	
Capacitance.....	22
Contact-Potential.....	19
Contact-Resistance.....	20
Dry-Reed.....	18
Electromechanical.....	17
Insulation-Resistance.....	22
Maximum-Current.....	19
Maximum-Power.....	19
Maximum-Voltage.....	18
Operating-Speed.....	20
Relay-Coil-Power.....	21
Signal-Frequency.....	23
Types-Defined.....	17
Types-of.....	15
Relay-Coil-Power.....	21
Reset.....	46
S	
Safety and Handling.....	i
Sample.....	24, 47
Setup.....	32, 33, 34
Setup Maintenance.....	32
Setup-and-Installation.....	31
slot.....	47
Slot.....	40, 42, 44
SPDT.....	7, 12, 15
SPST.....	7

Switch Matrix topology	13	Matrix	26
Switching	23	Multiplex-(Scan).....	12, 24
Switching-Systems		Switching	23
Automating	11	Switch-Matrix	13
Components	15	Trademarks	ii
Design.....	11	TRD	7, 11
How-to-choose-Instruments	14	Two-Pole Matrix.....	28
Relay-Types.....	17	U	
Topologies	12, 23	Using the GXSW driver functions.....	44
Switch-Matrix-Topology	13	UUT	7, 9, 12, 13, 24, 26, 28
Switch-Topology	12	V	
System		Virtual Panel.....	32, 34, 44
Directory.....	33	VISA.....	33, 39, 44, 45
T		Visual Basic.....	36, 43
Test System	9	Visual Basic .NET	35
Topology		Visual C++	34, 43
Basic-Type.....	12	W	
Defined-Topologies	23	Warranty	i

