

Operator's Manual

Model SMU4032 35 Channel USB Instrumentation Relay Multiplexer
Model SMU4030 35 Channel USB Relay Multiplexer



Signametrics Corporation

May, 2006

CAUTION

In no event shall Signametrics or its Representatives be liable for any consequential damages whatsoever (including, without limitation, damages for loss of business profits, business interruption, loss of business information, or other loss) arising out of the use of or inability to use Signametrics' products, even if Signametrics has been advised of the possibility of such damages. Because some states do not allow the exclusion or limitation of liability for consequential damages, the above limitations may not apply to you.

© 2003 Signametrics Corp. Printed in the USA. All rights reserved. Contents of this publication must not be reproduced in any form without the permission of Signametrics Corporation.

TABLE OF CONTENTS

1.0 INTRODUCTION	5
1.1 SAFETY CONSIDERATIONS	5
1.2 MINIMUM REQUIREMENTS	5
1.3 DESCRIPTION	6
1.4 CONFIGURATION AND TREE RELAYS	7
1.5 SELF TEST AND CONTACT CLEANING	7
2.0 SPECIFICATIONS AND FEATURE TABLE	8
2.1 TRIGGER INPUTS AND OUTPUTS	8
2.1.1 External Hardware Trigger Inut.....	8
2.1.2 External Hardware Trigger Output.....	9
2.2 OTHER SPECIFICATIONS	9
3.0 GETTING STARTED	9
3.1 SETTING THE SCANNER	9
3.2 INSTALLING THE SOFTWARE.....	9
3.3 INSTALLING THE SCANNER MODULE.....	10
3.4 SCANNER CONNECTOR PIN-OUT.....	11
3.5 STARTING THE CONTROL PANEL	13
3.6 USING THE CONTROL PANEL	13
4.0 SCANNER TUTORIAL	15
4.1 SCANNER CONFIGURATIONS.....	15
4.1.1 Two Wire Multiplexing	16
4.1.2 Four Wire Multiplexing	16
4.1.3 Six Wire Multiplexing	17
4.1.4 Two Groups Configuration.....	17
4.1.5 Four Groups Configuration.....	18
4.1.6 Thermocouple ColdJunction test Configuration.....	18
4.1.7 Disabled Configuration	18
4.2 SCANNER OPERATIONS.....	18
4.2.1 Trigger Output.....	18
4.2.2 Trigger Input.....	19
4.2.3 Auto Scan Operation.....	19
4.2.4 Triggered Auto Scan Operation.....	19
4.2.5 Triggered Scan Operation	20
4.2.6 Single Step Scan Operation	20
4.2.7 Scan List Operations.....	21
4.2.8 Locating Shorted Channel	21
4.2.9 Self Cleaning the Relay Contacts.....	21
4.2.10 Integrity Test.....	22
4.2.11 Self Tests	22
4.2.12 Setting Actuation Time Parameter.....	22
4.2.13 Setting Step Time Parameter	22
4.2.14 Thermocouple Temperature Measurements.....	23
4.3 POLLED TYPE OPERATIONS	23
4.4 INTERFACING TO THE SMU2060 USB DMM	23
4.4.1 Triggering the SMU2060 DMMs.....	24
4.4.2 Multiplexing with the SMU2040 DMMs	24
4.4.3 Interface Commands and Timing.....	24
4.5 SINGLE ENDED APPLICATIONS	25
4.5.1 Point to Point Operation.....	25
4.5.2 Single Ended Selector	26
5.0 SMU4032 SCANNER FAMILY WINDOWS INTERFACE	27

5.1 DISTRIBUTION FILES	27
5.2 USING THE SMU4032 DRIVER WITH C++ OR SIMILAR SOFTWARE	28
5.2.1 Multiple Card Operations under Windows.....	28
5.3 VISUAL BASIC FRONT PANEL APPLICATION	29
5.3.1 Visual Basic Simple Application.....	29
5.4 WINDOWS DLL DEFAULT MODES AND PARAMETERS	31
5.5 USING THE SMU4032 DLL WITH LAB WINDOWS/CVI®.....	31
5.6 WINDOWS COMMAND LANGUAGE	31
SCANAbort.....	31
SCANAutoScan	31
SCANCleanRelays.....	32
SCANClose.....	33
SCANDelay	33
SCANErrString	34
SCANGetActuationTime	35
SCANGetBusInfo.....	35
SCANGetConfig	36
SCANGetGrdVer.....	36
SCANGetHwVer.....	37
SCANGetID.....	37
SCANGetManDate.....	38
SCANGetScanList	38
SCANGetShortedChannel	39
SCANGetStepTime	39
SCANGetTriggerIn	40
SCANGetType	40
SCANGetVer	41
SCANInit	41
SCANIsInitialized.....	42
SCANOpenAllChannels	42
SCANOpen.....	43
SCANReady.....	44
SCANSelectChannel.....	44
SCANSelectChannelCmd	45
SCANSetActuationTime	46
SCANSetChannelRelay	46
SCANSetConfig.....	47
SCANSetConfigRelay.....	48
SCANSetScanList.....	48
SCANSetStepTime	49
SCANSetTriggerOut.....	49
SCANSetupStep.....	50
SCANStep.....	51
SCANStepCmd	51
SCANTerminate	52
SCANTestChanIntegrity.....	52
SCANTestChannelRelay	53
SCANTestConfigRelay	54
SCANTrigAutoScan.....	55
SCANTriggerInState	55
SCANTriggerOutState.....	56
SCANTrigScan.....	56
6.0 ACCESSORIES.....	58
7.0 WARRANTY AND SERVICE.....	58

1.0 Introduction

Congratulations! You have purchased a USB switching instrument with analog and systems performance that rivals the best, all-in-one box, instruments. The SMU4032/30 relay scanner/multiplexer is easy to setup and use, have sophisticated analog and digital circuitry to provide very repeatable switching. Please take a few moments and review this manual before installing and using this instrument.

This manual describes the SMU4032 and SMU4030. This product delivers unmatched switching performance in a USB instrument. With a rich repertoire of functions, the SMU4032 out performs all other USB plug-in Scanners, including most brand-name bench top units.

1.1 Safety Considerations

Safety Considerations

The SMU4032 Scanner is capable of switching 220 VDC, while the SMU4030 is capable of 120V across the High and Low terminals. The SMU4032 is also capable of handling common mode signals that "float" the Scanner above EARTH ground by up to 220 VDC or 220 VAC. When making common mode switching, the majority of the circuits inside the Scanner are at the common mode voltage. **These voltages can be lethal and may KILL! It is important to note that using the SMX40 or SMX40L limit the voltages to 40V due to the structure of these products. Using appropriate and safe connection means to the SMU4030 and SMU4032 is required when voltages above 40V are involved.**

The Scanner comes installed inside a plastic case. **The case must not be removed for performance as well as safety reasons.** Removal of the case and/or improper assembly can result in lethal voltages exposure from either the SMU4030/32 or the computer it is connected to.

Warning

To avoid shock hazard, install the Scanner only into a computer that has its power connector connected to a power receptacle with an earth safety ground.

When connecting to signals above 50V DC or 40V AC, only use Safety Test Leads and harnesses.

Do not use the SM40 and SM40L if voltages connected are above 50V DC or 40V AD.

1.2 Minimum Requirements

The SMU4032 relays Scanner is compatible with any USB 1.1 and higher version. A mouse or other pointing device must be installed when controlling the Scanner from the Windows Control Panel. The install package comes with a Windows' DLL, for operation with Windows 9X/ME/2000/XP/Millennium.

1.3 Description

The SMU4030/32 are 35 channel models that can be arranged in up to four groups of 8:1 differential channels. The SMU4032 is a high reliability instrumentation quality scanner specifically designed for switching in applications requiring precision low noise and low leakage. The most outstanding feature of this model is a very low Thermal EMF, resulting in highly accurate Ohms, low Voltage, temperature and other sensitive applications requiring low high precision and low noise. For cost sensitive applications, where these issues are not as critical, the SMU4030 offers an economical solution. Though inferior to the SMU4032, the non isolated SMU4030 is a much better switching card compared to similar plug-in products on the market.

When using an SMU2060 series 7-1/2 digit USB DMMs, or similarly accurate DMM, it is necessary to use the Instrumentation quality SMU4032 scanners in order to maintain full measurement accuracy provided by the DMM.

These scanners are software configurable on the fly, and can handle several pre-defined switching configurations, such as *TwoWire*, *FourWire*, *SixWire*, *Universal*, *TwoGroups*, *FourGroups*, *ColdJunction* and *Disabled*. In the *TwoWire* configuration, the scanner is a 35 channel multiplexer. In the *FourWire* configuration, it automatically selects two simultaneous channels, allowing a DMM or other device to connect to four individual lines. In the *SixWire* configuration six wires, or three differential channels are selected simultaneously, providing such applications as six wire guarded in-circuit measurements (see SMU2064 DMM for details). The *Universal* configuration turns the scanner into a pseudo matrix structure, allowing any relay, including channel relays, tree relays and configuration relays, to be closed or open, including multiple relays. The *TwoGroups* and *FourGroups* configurations split the Scanner into several independent Two-Wire multiplexers. When in the *ColdJunction* configuration, the Dbus is connected to the Abus while all other channels are open, to facilitate the measurement of the Cold Junction temperature sensor of the SMU40T module, or a user cold junction sensor.

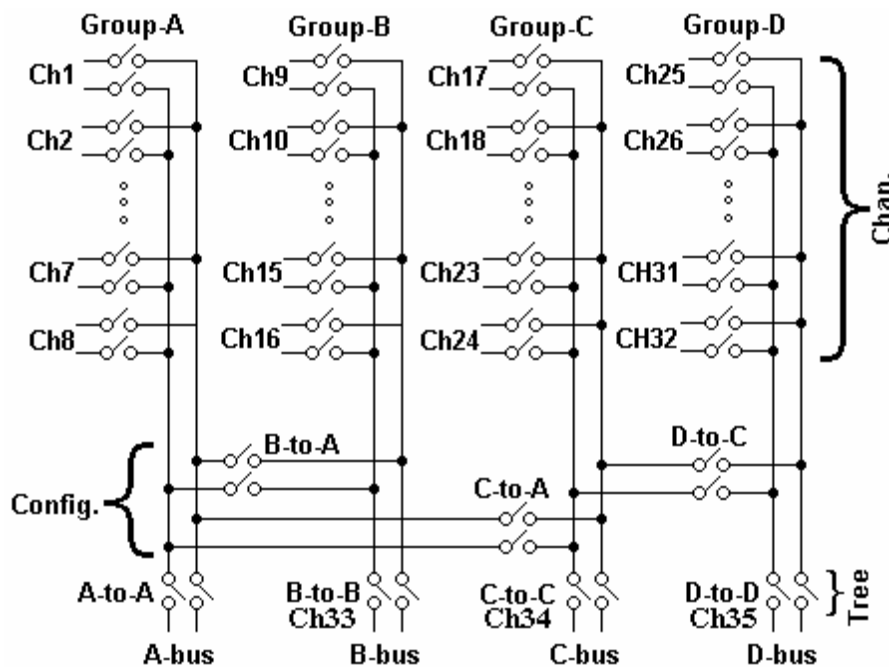


Figure 1-1. Diagram of the SMU4030 and SMU4032 relay scanners.

In the *TwoWire*, *FourWire*, *SixWire*, *TwoGroups*, and *FourGroups* configuration, the switching sequence includes the opening of previously selected channels and then closing of the selected channel(s) relay. In the case where multiple channels are supposed to be closed, such as in *FourWire* configuration, these channels are closed simultaneously so that no additional time beyond the actuation time is consumed. Break-before-make operation is maintained due to an active high speed turns off circuit, which shuts off the energized coil's current very quickly as compared to the turn on time.

1.4 Configuration and Tree Relays

In addition to the channel switching relays, the SMU4032 includes two sets of relays used for configuring of the scanner. The configuration switches facilitate inter-group connections and the Tree relays provide isolation of each of the group's buses from their respective channel relays. These relays are automatically configured on the fly when the scanner receives configuration commands or while a channel is being selected, which requires routing via these relays. In cases where the pre-defined configurations (*TwoWire*, *FourWire*, *SixWire*, *TwoGroups*, and *FourGroups*) do not meet the test needs, select the *Universal* configuration. It allows an unlimited control of any of the channel, configuration and tree relays independently. Care must be taken however, since it does not offer the protection of the predefined configurations. This could result in shorting some "hot" lines, constituting in potential hazard to the user from any thing that is connected to the scanner and from the scanner itself. This could also lead to damaging the scanner or anything connected to it. It may also present an electricution hazard.

1.5 Self Test and Contact Cleaning

In addition to the channel relays, Tree relays and Configuration relays, there are additional relays and circuits to handle the various built-in tests of these Scanners. The user does not have direct control over these components.

The optional test connector provides the means to short each of the cannel relays. This is required in various self tests and cleaning operations. This accessory (SMX40L) is a Female DB-78 position connector which has 35 shorts across all Channels (Ch1Hi shorted to Ch1Lo etc.). The A-Bus is left open to provide access.

2.0 Specifications and Feature Table

Function	SMU4032 Instrumentation	SMU4030
Number of differential channels	35	35
Number of 8:1 groups	four	four
Scanning Arrangement	Four groups of 8:1 differential	Four groups of 8:1 differential
Thermal EMF offset	< 1.0 μ V	10 μ V
Maximum Switching DC Voltage	220 V	120 V
Maximum Switching AC Voltage	250 V	120 V
Maximum Switching Current	1 A	1 A
Maximum Switching Current	1 A	1 A
Typical inter-channel Capacitance	15pf	15pf
Insulation between open contacts	> 1,000 M Ω	> 100 M Ω
Insulation; contacts to coils	> 1,000 M Ω	> 100 M Ω
Insulation; adjacent channels	> 1,000 M Ω	> 100 M Ω
No Load Life	10 ⁸	1X10 ⁷
Loaded Life @ 50Vdc, 0.1A	10 ⁶	3X10 ⁵
Trigger input	√	√
Trigger output	√	√
Thermocouple Cold Junction Capable	√	
Typical Closure Time	4 ms	12 ms
Typical Release Time [1]	2 ms	5 ms
Actuation Time [2]	< 5 ms	15 ms
Actuation Time range	1ms to 850ms in 0.25ms steps	1ms to 850ms in 0.25ms steps
AutoScan Period Range	1ms to 850ms in 0.25ms steps	1ms to 850ms in 0.25ms steps
Available Configurations	2-wire, 4-wire, 6-wire, universal, two-groups, four-groups	2-wire, 4-wire, 6-wire, universal, two-groups, four-groups
Available Scan Groups	A, B, C and D	A, B, C and D
Isolated Relay Coil Drive	√	

[1] Release time is applicable while in Universal Mode.

[2] Actuation time is the time it takes to open any closed relays, and close the selected relay.

Table 2-1. Specifications and Feature table.

2.1 Trigger Inputs and Outputs

2.1.1 External Hardware Trigger Input

Trigger Input line Characteristics

- **Input LED** (nominal 1.7V drop at 1mA) with a series 1k Ω resistor.
- **Input Signal requirements** >2.5 V, < 10 V to activate. < 1V to deactivate.
- **Isolation** Optically Isolated from all other circuitry. Shares a common line with Trigger Output.

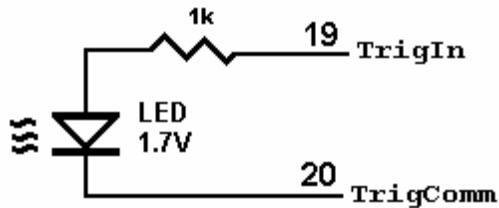


Figure 2-1. Trigger input equivalent circuit.

2.1.2 External Hardware Trigger Output

Output Characteristics

- **Output Circuit** Open collector of a NPN transistor (nominal 0.4V saturation voltage) in series with 1k Ω resistor.
- **Collector Emitter Voltage** < 30 V
- **Output Current** < 4 mA
- **Reverse Voltage** < 7V
- **Isolation** Optically Isolated from all other circuitry. Common line with Trigger Input.

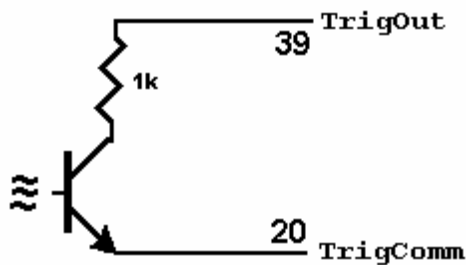


Figure 2-2. Trigger Output Optical Isolator NPN circuit.

2.2 Other Specifications

Hardware Interface	USB Bus
Safety	Designed to IEC 1010-1, Installation Category I.
Temperature Range	0°C to 50°C, operating
Size	Standard USB 5.07" X 8.5"
Power	+5 volts, 450 mA maximum

Note: Signametrics reserves the right to make changes in materials, specifications, product functionality, or accessories without notice.

3.0 Getting Started

After unpacking the Scanner, please inspect for any shipping damage that may have occurred, and report any claims to your transportation carrier.

The Scanner package is shipped with the Scanner module, four floppy disks containing software drivers, user interface panel, and this Operator's manual.

3.1 Setting the Scanner

The SMU4000 series Relay Scanners are USB plug-and-play devices and do not require any switch settings, or any other adjustments to the hardware prior to installation.

3.2 Installing the Software

The first step in the installation process requires that you plug-in the SMU2040/42 into a USB port. Next follow the on the screen for 'New Hardware Found'. The first time you power up with the Scanner installed, the computer plug-and-play will detect the new Scanner and prompt you for a driver. The driver your computer requires is included on the installation CD (SMU4032.inf). Guide the computer to search for device driver on the CD drive. Follow this by running the autorun.exe located on the installation CD. Next run the autorun.exe included on the installation CD. Alternatively remove and reinsert the installaton CD which will invoke the installation automatically.

3.3 Installing the Scanner Module

Warning

To avoid shock hazard, install the Scanner only into a personal computer that has its power line connector connected to an AC receptacle with an Earth Safety ground.

After installation, check to see that no loose wires or ribbon cables infringe upon any of the internal circuits of the Scanner, as this may apply measurement voltages to your computer, causing personal injury and/or damage to your computer!

Caution: Only install the Scanner module with the power turned OFF to the USB chassis !

Use extreme care when plugging the Scanner module(s) into a USB slot.

3.4 Scanner Connector Pin-out

Before using the Scanner, please take a few moments and review this section to understand where the multiplexed and common signals are located in the DB-78 position male connector are located. **Under no circumstances should voltage and current limits exceed the specified values, as personal injury or damage to the instrument, your computer or application may result.**

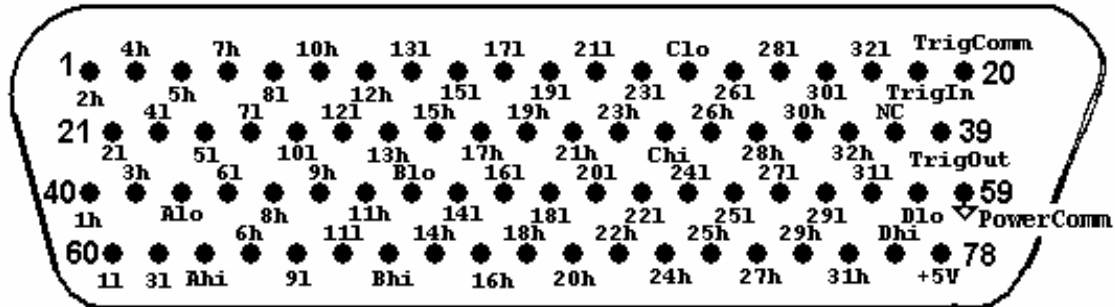


Figure 3-1. The Scanner's DB-78 connector, facing the bracket.

Pin Number	Label	Description	Pin Number	Label	Description
1	Ch2Hi	Channel 2 High	33	CHi	C-bus/Ch34 High
2	Ch4Hi	Channel 4 High	34	Ch26Hi	Channel 26 High
3	Ch5Hi	Channel 5 High	35	Ch28Hi	Channel 28 High
4	Ch7Hi	Channel 7 High	36	Ch30Hi	Channel 30 High
5	Ch8Lo	Channel 8 Low	37	Ch32Hi	Channel 32 High
6	Ch10Hi	Channel 10 High	38	NC	No Connection
7	Ch12Hi	Channel 12 High	39	Trig_Out	Trigger Output
8	Ch13Lo	Channel 13 Low	40	Ch1Hi	Channel 1 High
9	Ch15Lo	Channel 15 Low	41	Ch3Hi	Channel 3 High
10	Ch17Lo	Channel 17 Low	42	ALo	A-bus Low
11	Ch19Lo	Channel 19 Low	43	Ch6Lo	Channel 6 Low
12	Ch21Lo	Channel 21 Low	44	Ch8Hi	Channel 8 High
13	Ch23Lo	Channel 23 Low	45	Ch9Hi	Channel 9 High
14	CLo	C-bus/Ch34 Low	46	Ch11Hi	Channel 11 High
15	Ch26Lo	Channel 26 Low	47	BLo	B-bus/Ch33 Low
16	Ch28Lo	Channel 28 Low	48	Ch14Lo	Channel 14 Low
17	Ch30Lo	Channel 30 Low	49	Ch16Lo	Channel 16 Low
18	Ch32Lo	Channel 32 Low	50	Ch18Lo	Channel 18 Low
19	Trig_IN	Trigger Input	51	Ch20Lo	Channel 20 Low
20	Trig_Com	Trigger common	52	Ch22Lo	Channel 22 Low
21	Ch2Lo	Channel 2 Low	53	Ch24Lo	Channel 24 Low
22	Ch4Lo	Channel 4 Low	54	Ch25Lo	Channel 25 Low
23	Ch5Lo	Channel 5 Low	55	Ch27Lo	Channel 27 Low
24	Ch7Lo	Channel 7 Low	56	Ch29Lo	Channel 29 Low
25	Ch10Lo	Channel 10 Low	57	Ch31Lo	Channel 31 Low
26	Ch12Lo	Channel 12 Low	58	DLo	D-bus/Ch35 Low [1]
27	Ch13Hi	Channel 13 High	59	Comm	Common
28	Ch15Hi	Channel 15 High	60	Ch1Lo	Channel 1 Low
29	Ch17Hi	Channel 17 High	61	Ch3Lo	Channel 3 Low
30	Ch19Hi	Channel 19 High	62	AHi	Channel 32 Low
31	Ch21Hi	Channel 21 High	63	Ch6Hi	Channel 6 High
32	Ch23Hi	Channel 23 High	64	Ch9Lo	Channel 9 Low

Pin Number	Label	Description	Pin Number	Label	Description
65	Ch11Lo	Channel 11 Low	72	Ch24Hi	Channel 24 High
66	BHi	B-bus/Ch33 High	73	Ch25Hi	Channel 25 High
67	Ch14Hi	Channel 14 High	74	Ch27Hi	Channel 27 High
68	Ch16Hi	Channel 16 High	75	Ch29Hi	Channel 29 High
69	Ch18Hi	Channel 18 High	76	Ch31Hi	Channel 31 High
70	Ch20Hi	Channel 20 High	77	DHi	D-Bus/Ch35 High [1]
71	Ch22Hi	Channel 22 High	78	+5V	+5V unregulated supply

[1] D-Bus is not available when using an Isothermal Terminal Block (SMX40T).

Table 3-1. The Scanner's DB-78 connector pin assignments.

ChxHi, ChxLo - These are the channel positive and negative terminals for all group channels, respectively. It starts with Ch1Hi/Lo and ends with Ch32Hi/Lo. Depending on the Scanner configuration, these lines are routed to the bus terminals AHi/Lo, BHi/Lo, CHi/Lo, or DHi/Lo.

AHi, ALo - These are the A-bus positive and negative terminals respectively. The channels positive and negative lines may be routed to these lines depending on Scanner configuration. In the Universal mode, it is also possible to route bus pairs to other bus pairs.

BHi, BLo - These are the B-bus positive and negative terminals respectively. The channels positive and negative lines may be routed to these lines depending on Scanner configuration. In the Universal mode, it is also possible to route bus pairs to other bus pairs. In TwoWire mode these are also Ch33Hi/Ch33Lo lines.

CHi, CLo - These are the C-bus positive and negative terminals respectively. The channels positive and negative lines may be routed to these lines depending on Scanner configuration. In the Universal mode, it is also possible to route bus pairs to other bus pairs. In TwoWire mode these are also Ch34Hi/Ch34Lo lines.

DHi, DLo - These are the D-bus positive and negative terminals respectively. The channel positive and negative lines may be routed to these lines depending on Scanner configuration. In the Universal mode, it is also possible to route bus pairs to other bus pairs. It is important to note that when using the Isothermal Block the DHi and DLo terminals should not be used, and a configuration involving them should not be selected with the exception of the *ColdJunction* configuration which selects the Cold Junction temperature sensor of an SMU40T. Therefore configurations such as *FourGroups* should not be used, and when selecting *Universal* configuration, care must be taken not to include these terminals. Since these lines also serve as Ch35Hi/Ch35Lo, avoid selecting Ch35 when using the Isothermal Block (SMU40T) In *TwoWire* mode.

TrigComm - This is the **TrigIn** and **TrigOut** return terminal. Both of these signals are referenced to it. All three signals associated with the trigger circuitry, **TrigComm**, **TrigIn**, and **TrigOut** are optically isolated from the rest of the terminals in the DB-78 connector.

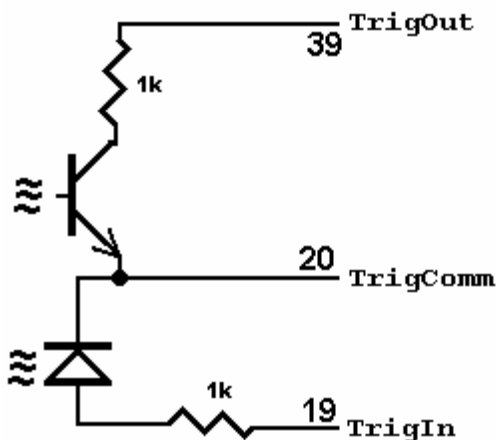


Figure 3-2. The trigger input and output lines are isolated by an optical isolators.

TrigIn - This is the trigger input signal. It requires TTL or CMOS level (at least 2.5V) to activate the trigger input. A series 1k removes the need to add external resistor and allows direct connection to a TTL or CMOS logic source. It is referenced to the **TrigComm** line.

TrigOut - This is the trigger output signal. It is an open collector signal with 1k resistor in series. It will drive a CMOS or TTL line provided it is connected to an appropriate positive supply (from 3V to 10V). It is referenced to the **TRIG_ comm** line.

+5V - This is a 5V supply line. It is designed to support the optional active Isothermal Block. It may also be used to power the **TrigOut** signal. When using it to power **TrigOut**, it is recommended that a 10k be connected from **TrigOut** to it as in Figure 3-3. This supply may vary between 4.7V to 5.7V, and its usage should be limited to no more than 10mA. This supply is isolated from the rest of the signals in the DB-78 connector. This signal is referenced to the **PowerComm** line.

PowerComm - This is the return line for the +5V supply. Since **TrigComm** and **PowerComm** are isolated from each other, they must be connected as in Figure 3-3 if the +5V is used to power **TrigComm**. This supply return is isolated from the rest of the signals in the DB-78 connector.

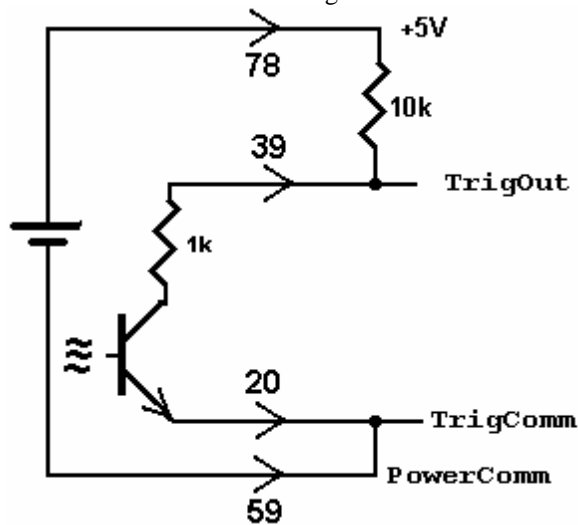


Figure 3-3. Trigger out may use the +5V supply provided for generating logic output.

3.5 Starting the Control Panel

After installing the software, you can verify the installation and gain familiarity with the Scanner by exercising its measurement functions using the Windows based Control Panel. To run the control panel, double click the 'SMU4032.EXE' icon. If you do not hear the relays click, you may have an installation error.

The Control Panel is operated with a mouse. All functions are accessed using the left mouse button.

Note: The SMU4000 front panel powers up in Disabled mode, with all relays open.

3.6 Using the Control Panel

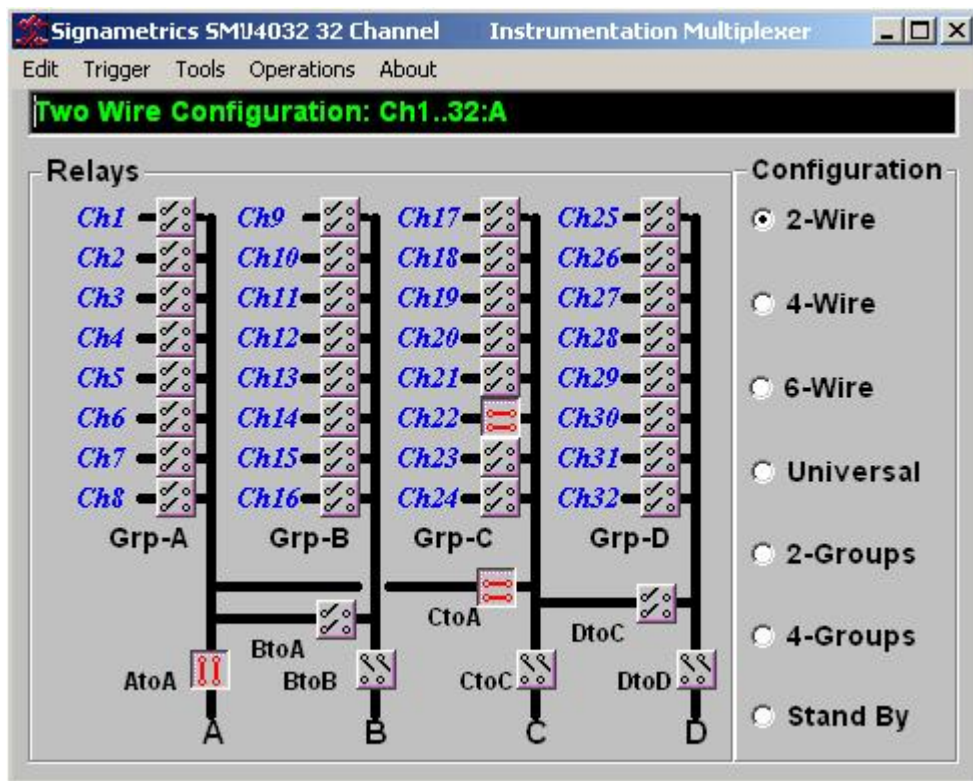
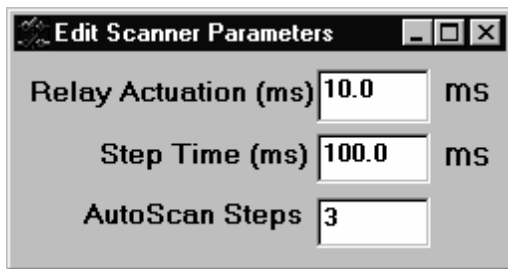


Figure 3-2. The Control Panel for the SMU4032. The three main groups include the relays buttons, the configuration selection options, and the main menu.

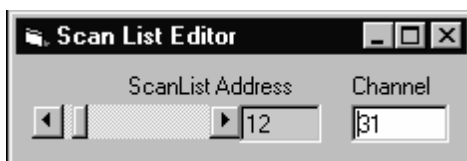
Note: All of the controls described below correspond to their respective software function, which can be invoked within your control software or as objects in a visual programming environment. Using the software command language of the SMU4000 allows additional capabilities and functions that are not included in the control panel above.

Relay Buttons - These buttons are context sensitive. Depending on the selected Configuration, these buttons allow the closing and opening of the various channel relays. In the *Universal* configuration, the user has control over all relays. In all other configurations, several relays may be activated simultaneously to rout the selected channel(s).

Edit Scanner Parameters - This panel provides means to enter the Relay Actuation time and the Step Times, as well as the total number of steps in an AutoScan.



ScanList Editor - This panel provides means to edit the contents of the ScanList.



Configuration Option Checks - This section of the main panel allows the selection of one out of all of the possible pre defined configurations. On power up, the configuration is set to *Disabled*. Possible selections include *Disabled*, *TwoWire*, *FourWire*, *SixWire*, *Universal*, *TwoGroups*, and *FourGroups*.

Edit Menu - This menu item has three selections, setting all parameters to their default value, opening the Edit Scanner Parameter sub-panel, and opening the Scan List Editor sub-panels.

Trigger Menu - This menu item controls the functionality of the scanner's trigger input and output signals. It enables or disabled the Trigger output, and sets the polarity of both Trigger input and output.

Tools Menu - This menu item provides access to some of the special test and cleaning tools. It includes the Self-Cleaning function, the various self-tests, and the shorted input detector.

Operations Menu - This menu item allows running some of the special scanner's operations such as AutoScan, Triggered Scan, Triggered Auto Scan, and the setting and monitoring of the Trigger output and trigger input respectively.

4.0 Scanner Tutorial

Most of the SMU4000 functions are accessible from the control panel described in the previous section. This section describes in detail the Scanner's operations and practices. To gain familiarity with the SMU4000 series Scanners, run the Windows 'SETUP.EXE' to install the software, then run the Scanner control panel to demonstrate its operations.

4.1 Scanner Configurations

The various configurations maybe experimented with using the control panel. Connect to the Scanner's terminals to the appropriate channels and bus lines, making sure not to exceed any of the specifications and limits. Use the control panel's option checks to select the desired operation mode, and the switch buttons to close or open the appropriate channels.

The SCANSetConfig() DLL function sets the Scanner configuration. The amount of time it takes to set a new configuration is equal to the set Actuation time. The Scanner can function as a multiplexer, or as an uncommitted matrix switch. In the *TwoWire*, *FourWire*, *SixWire*, *TwoGroups*, and *FourGroups* configurations the Scanner acts as a multiplexer. This means that when a channel is selected, all other channels in the group are opened prior to closing the selected channel. For instance, in *FourWire* configuration, the response to 'select Ch2) will result in opening of a previously selected channel relays and the closure of Ch2 and Ch18. All this will be done within a single actuation time.

4.1.1 Two Wire Multiplexing

In the *TwoWire* multiplexed configuration, the SMU4000 family first opens all channels, than it connects the Hi and Lo terminals of the selected channel, to the Hi and Lo terminals of the A-bus. Ch1 to Ch35 are routed to the A-bus. Selecting Ch1 will result in Ch1Hi connected to A-bus Hi and Ch1Lo connected to A-bus Lo.

When making very low level DCV measurements (<1 mV), you should use copper wires. A common source of error can come from your test leads, which can introduce tens or hundreds of μ Volts of error due to thermal voltages (Thermal EMF or TEFM). To minimize thermo-voltaic effects, after handling the test leads, wait a while for lead and terminal temperatures to equalize before attempting to make measurements. Signametrics offers several high quality probes that are optimal for low level measurements. It is essential to select the right switching card for the job. Make sure the relay scanner selected has sufficiently low thermal EMF (offset) specification. At 1μ V thermal EMF, the SMU4032 is the best choice for low level voltage and resistance measurements. The Digital Multimeter you select is also important. High end DMMs such as the SMU2042, SMU2044, SMU2064 and Agilent 34420A have the advantage of having an order of magnitude more current, reducing low resistance measurement errors by a factor of 10. The following table quantifies the effect of TEFM on Ohms measurement, using a high quality DMM such as the SMU2042, SMU2044 and SMU2064 in conjunction with a low thermal EMF switches such as the SMU4030 and SMU4032 will result in the lowest possible error. Be aware that this effect is the same for either, 2-Wire or 4-Wire Ohms measurement. The resistance measurement error is equal to the thermal voltage divided by the DMM Ohms test current. In other words, no use in getting a high end DMM while using low end switching card.

SMU2064 Range	Ohms Current	DMM Resolution	Error due to 1μ V EMF [1]	Error due to 10μ V EMF [2]	Error due to 300μ V EMF
24 Ω	10 mA	1 $\mu\Omega$	100 $\mu\Omega$	1 m Ω	30 m Ω
240 Ω	1 mA	10 $\mu\Omega$	1 m Ω	10 m Ω	300 m Ω
2.4 k Ω	1 mA	100 $\mu\Omega$	1 m Ω	10 m Ω	300 m Ω
24 k Ω	100 μ A	1 m Ω	10 m Ω	100 m Ω	3 Ω
240 k Ω	10 μ A	10 m Ω	100 m Ω	1 Ω	30 Ω
2.4 M Ω	1 μ A	100 m Ω	1 Ω	10 Ω	300 Ω
24 M Ω	100 nA	10 Ω	10 Ω	100 Ω	3 k Ω

[1] SMU4032 performance [2] SMU4030 performance

Table 4-1. The Scanner thermal EMF affect on Ohms measurement accuracy.

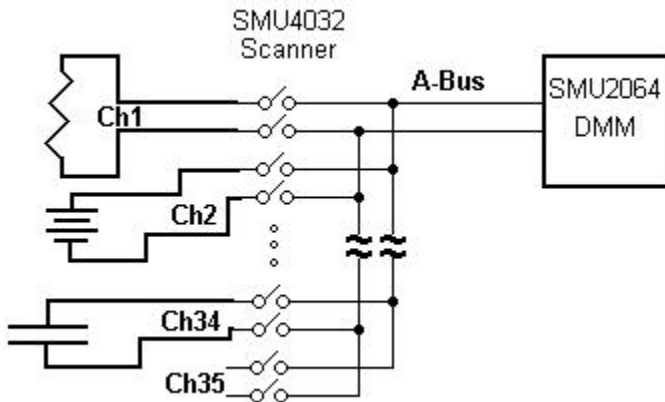


Figure 4-1. 32 Channel 2-Wire measurement application.

4.1.2 Four Wire Multiplexing

In the *FourWire* configuration, the SMU4000 family connects simultaneously two channels. They route Ch1 through Ch16 to the A-bus, and the corresponding channels, Ch17 through Ch32 to the C-bus. To measure a resistor using Kelvin connection, the A-bus can be connected to the DMM source leads and the B-bus to the sense leads. Make sure the polarity of the lines is consistent.

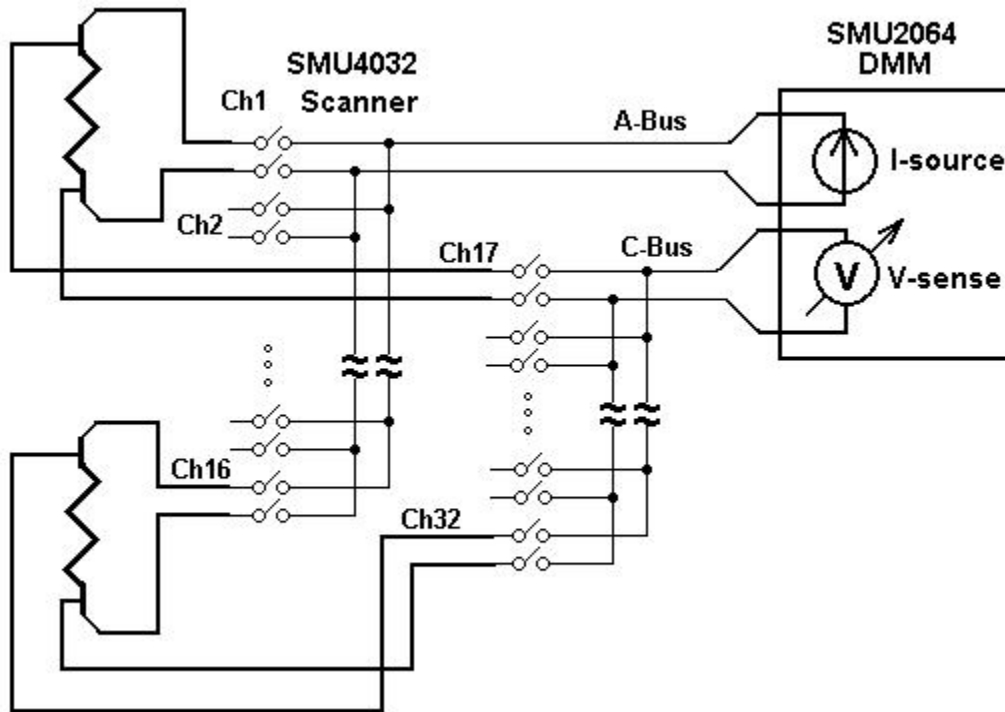


Figure 4-2. To perform 4-Wire resistance measurement with the SMU4032, connect Ch1 and Ch2 to a resistor, maintaining correct polarity.

4.1.3 Six Wire Multiplexing

In *SixWire* configuration, the SMU4030 and SMU4032 simultaneously connect one channel to the A-bus, one to the B-bus and one to the C-bus. Selecting a channel automatically closes three channel relays. For instance, selecting channel 2 results in the opening of all currently closed channel relays, followed by the closure of Ch2, Ch10, and Ch18. Each is routed to the A-bus, B-bus, and C-bus, respectively. To measure a resistor using 6-Wire guarded connection, the A-bus is connected to the DMM source leads, the B-bus to the sense leads and the C-bus to the Guarded point. Make sure the polarity of the lines is consistent. It should be noted that the 'D' group, consisting of Ch31 to Ch40, is available independently to provide additional 10 channels of 2-Wire multiplexing.

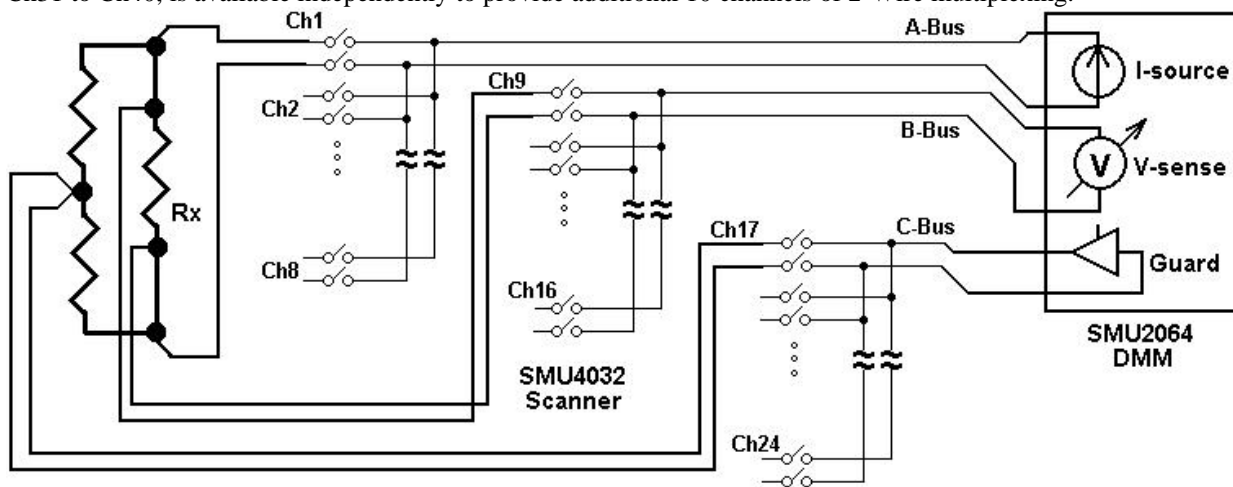


Figure 4-3. In the 6-Wire configuration, the three lead sets are switched simultaneously.

4.1.4 Two Groups Configuration

In the *TwoGroup* configuration, the SMU4000 family provides two independent multiplexing groups. Both groups function as two independent two wire multiplexers. When a channel is selected, any closed channel within that group is opened; next, the selected channel is closed. The SMU4030/32 route Ch1 to Ch16 to the A-bus forming the first group, and Ch17 through Ch32 to the C-bus, forming the second group.

4.1.5 Four Groups Configuration

In the *FourGroup* configuration, the SMU4030/32 provide four independent multiplexing groups. The groups function as independent two wire multiplexers. When a channel is selected, any closed channel within the selected group is opened, then the selected channel is closed. They route Ch1 to Ch8 to the A-bus forming the first group. Ch9 to Ch16 are routed to the B-bus forming the second group. Ch17 to Ch24 are routed to the C-bus forming the third group. Ch25 through Ch32 are routed to the D-bus to form the fourth group.

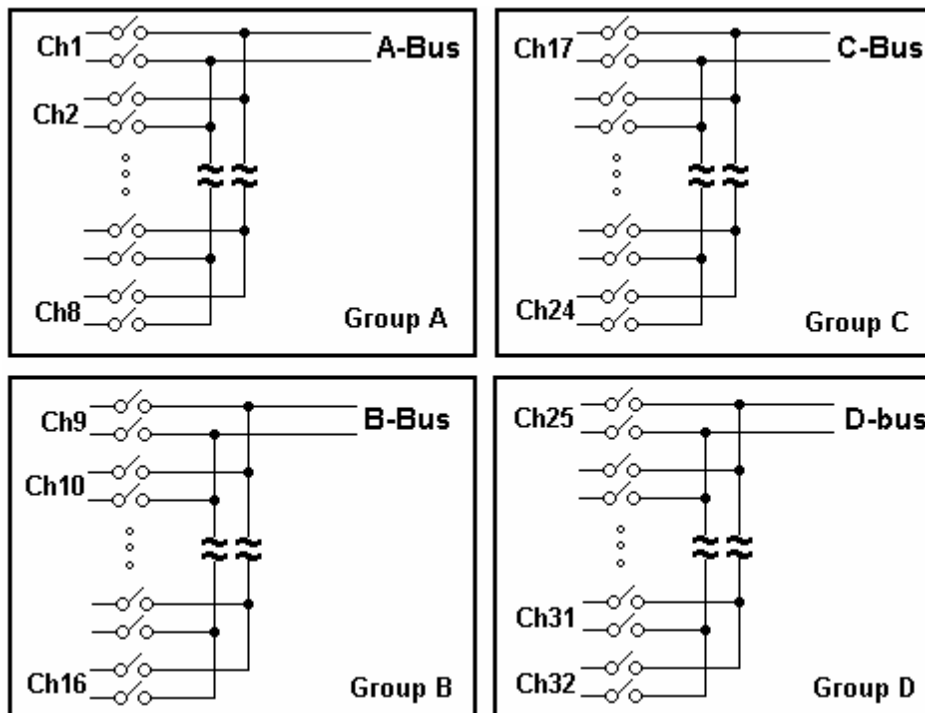


Figure 4-4. In the Four Groups configuration, four independent 2-Wire scanners are available.

4.1.6 Thermocouple ColdJunction test Configuration

Setting the 4030 or 4032 to this configuration connects channel 35 (D-Bus) to the primary measurement terminals, the A-Bus. This provides a measurement path to the optional SMX40T's active temperature sensor which facilitates thermocouple Cold Junction temperature measurement. This is required for making accurate thermocouple measurements. Therefore, while using the SMX40T channel 35 (D-bus) is unavailable as an input. See section 4.2.14 for more information.

4.1.7 Disabled Configuration

In the Disabled configuration, all relays, including channel and configuration relays are open. This is the default configuration, selected on power up, or initialization.

4.2 Scanner Operations

In addition to basic scanner operations such as selecting channels, the SMU4000 series has several built-in operations, which it more versatile and a lot more capable. These operations include various self-diagnostics procedures, contact cleaning, and auto scanning and interfacing to external test and measurement instruments.

4.2.1 Trigger Output

The trigger output line, **TrigOut**, maybe enabled or disabled as well as set for a positive or negative polarity using the **SCANTrigOutState()** command. Under normal operation, the **TrigOut** line is active for the duration of the Actuation time. It maybe used to drive or synchronize other test equipment. If set to positive polarity, a positive edge indicates the selected channel is closed and settled, provided the correct Actuation time is set (use **SCANSetActuationTime()** function to set it). The logic level of TrigOut may be set high or low using the **SCANSetTrigOut()** function. In the following diagram, **TrigOut** level corresponds to the circuit in figure 3.3.

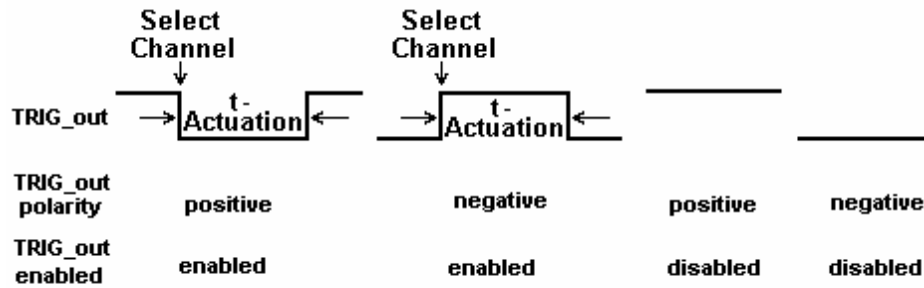


Figure 4-4. The trigger input maybe set for polarity and be enabled or disabled.

4.2.2 Trigger Input

The trigger input line, **TrigIn**, maybe set for a positive or negative polarity using the **SCANTrigInState()** command. Under normal operation, the **TrigIn** line has no effect on the operation of the scanner. It maybe used to synchronize other test equipment such as in the Triggered Auto Scan and Triggered Scan, where an external event initiates scanning operation. The level of the **TrigIn** line maybe monitored using the **SCANGetTrigIn()** command.

4.2.3 Auto Scan Operation

This operation requires that the Scanner be in one of the following configurations: *TwoWire*, *FourWire*, *SixWire*, *TwoGroups*, or *FourGroups*. The Auto Scan operation is a software initiated Scan sequence. Issuing **SCANAutoScan()** command triggers an automatic scanning operation, in which the first channel from the first location in the Scan List table is selected first, followed by a delay equal to t -Step. Channels are selected sequentially from the Scan List. The total number of points in the scan is controlled by the *iPoints* parameter passed by the **SCANAutoScan()** function, which must be a value between 1 and 192. This mode maybe terminated by sending **SCANAbort()** command to the Scanner during the scan. Use the **SCANOpenAllChannel()** function at the end of the scan, if you wish to open the last selected channel. The Trigger output signal can be used to synchronize other instruments to the SMU4000 Scanner. The value of t -Step and t -Actuation can be set to generate a trigger output signal for triggering instruments. For instance, setting the TrigOut polarity to positive sense (as in Figure 4-6) will result in a positive edge to trigger an SMU2044 DMM, after a channel relay is closed and settled.

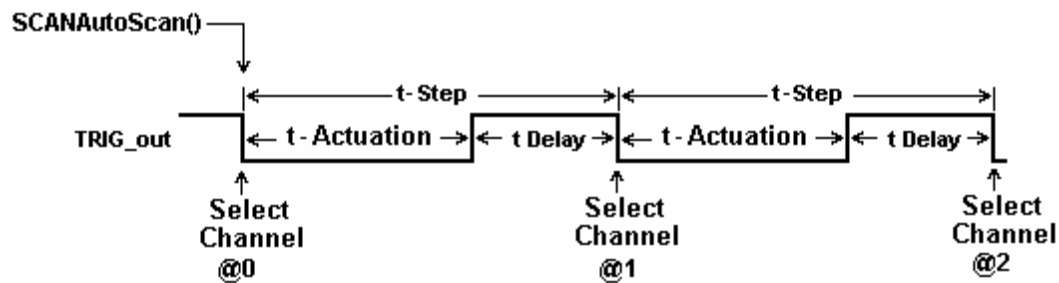


Figure 4-6. AutoScan is a software-initiated channel scanning procedure.

4.2.4 Triggered Auto Scan Operation

This operation requires that the Scanner be in one of the following configurations: *TwoWire*, *FourWire*, *SixWire*, *TwoGroups*, and *FourGroups*. The Triggered Auto Scan operation is a hardware-triggered version of the above Auto Scan operation. The trigger polarity can be selected. The **SCANTrigAutoScan()** function sets the Scanner into the Triggered Auto Scan mode, in which the Scanner waits for a trigger edge to initiate an Auto Scan operation. Once a trigger is received, the first channel from the first location in the Scan List table is selected, followed by a delay equal to t -Step, then the selection of subsequent channels as specified in the Scan List table. The Scanner proceeds to sequentially select a total of *iPoints*. The last parameter is passed to the scanner by the **SCANTrigAutoScan()** function. This value must be between 1 and 192. This may be terminated by sending **SCANAbort()** command to the Scanner. Use the **SCANOpenAllChannel()** function if you wish to open the last selected channel at the end of the scan.

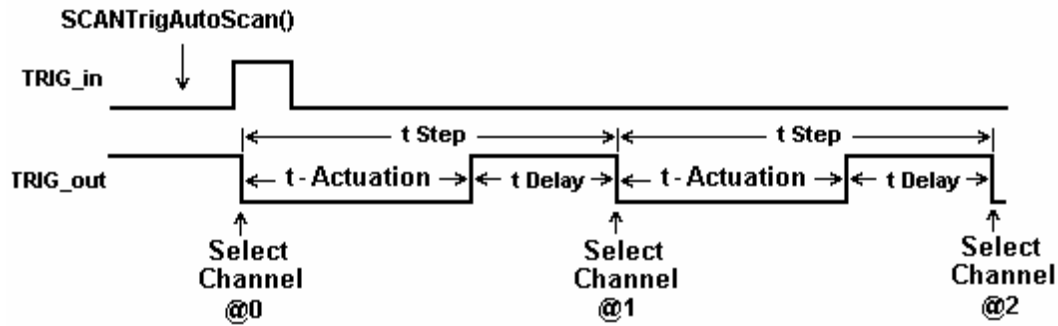


Figure 4-7. Positive edge Triggered AutoScan is a hardware initiated automatic scanning process.

4.2.5 Triggered Scan Operation

This operation requires that the Scanner be in one of the following configurations: *TwoWire*, *FourWire*, *SixWire*, *TwoGroups*, and *FourGroups*. The Triggered Scan operation is a hardware driven scanning process. Each step through the Scan List table is initiated by hardware trigger event. The trigger input signal edge polarity can be selected. The **SCANTrigScan()** function sets the Scanner into the Triggered Scan mode, in which the Scanner expects a total of *iPoints* triggers. Once this command is issued to the Scanner, it waits for the first trigger edge to select the first channel from the first location in the Scan List table. Following the selection of each point, the Scanner waits for *t-Actuation* period, during which triggers are ignored. The Scanner responds to subsequent trigger edges by sequentially selecting channels stored in the Scan List. The total number of points in the scan is controlled by the *iPoints* parameter passed to the scanner by the **SCANTrigScan()** function, which must be a value between 1 and 192. This mode is concluded at the end of the number of points specified, or terminated by sending **SCANAbort()** command to the Scanner. Use the **SCANOpenAllChannel()** function if you wish to open the last selected channel.

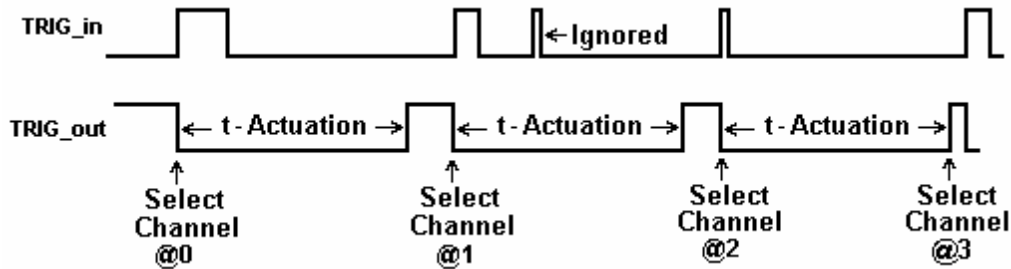


Figure 4-8. Positive edge Triggered Scan operation is a hardware initiated scanning process.

4.2.6 Single Step Scan Operation

This operation requires that the Scanner be in one of the following configurations: *TwoWire*, *FourWire*, *SixWire*, *TwoGroups*, and *FourGroups*. The Single Step Scan operation is similar to the Triggered Scan process with the exception that each step through the Scan List table is initiated by the **SCANStep()** command rather than by hardware trigger event. The **SCANSetupStep()** function prepares the Scanner for this operation by resetting the Scanner's Scan List pointer to point to the first entry of the table. In response to the **SCANStep()** command, the Scanner selects the channel stored in the Scan List location pointed to by the Scan List pointer, then it increments this pointer to point to the next location. The number of points in the scan is controlled by the number of times **SCANStep()** commands is issues since the last **SCANSetupStep()** function was executed. When done using the **SCANStep()** and **SCANSetupStep()** functions, the Scanner is going to remain with the last channel selected. The maximum number of entries in the Scan List is 192. The polled version of **SCANStep()**, **SCANStepCmd()** function maybe used if a polled control is required. Read about the usage of polled operations in the "Polled Type Operations" section. Use the **SCANOpenAllChannel()** function if you wish to open the last selected channel.

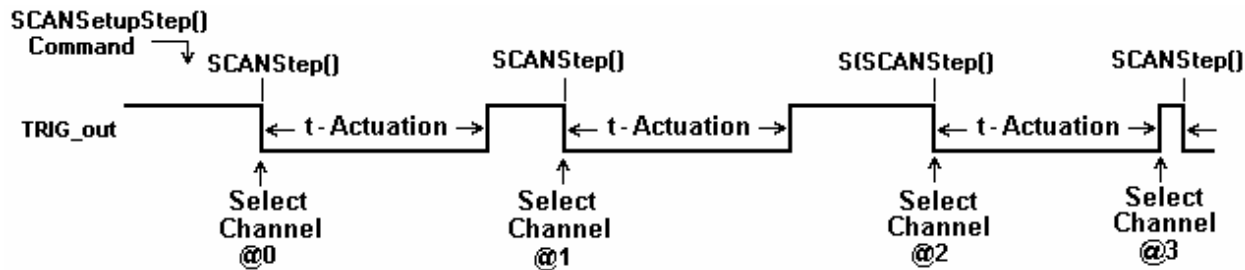


Figure 4-9. Single Step operation requires a `SCANSSetupStep()` then `SCANSStep()` command.

4.2.7 Scan List Operations

The Scan List table resides on board the scanner. It contains 192 locations, allowing a scan made up of up to 192 channels. Two DLL functions provide means to write and read to this table. The table contains channel numbers associated with the scanner channels. The Scanner may have entries with values between 1 and 32, representing the available channel numbers. An entry of zero forces deselection of all channels. The Scan List table may contain repetition, or a scan sequence with multiple selection of a single channel. Entries to the table are context sensitive. The current configuration of the Scanner determines how channel numbers are treated. For instance, an entry of 32 is not reasonable while in *FourWire* configuration. In this configuration, an entry of 1 (Channel 1) will force selection of two channels, Ch1 and Ch17.

All Auto Scan operations read the table sequentially, starting at address 0 and ending at 191. It is important to consider the configuration of the Scanner when setting the table. The table may be written and read any time the Scanner is not busy. Operations using the Scan List include: `SCANAutoScan()`, `SCANSSetupStep()`, `SCANSStep()`, `SCANStepCmd()` and `SCANTrigAutoScan()`.

When several Scanners are connected to form a larger channel count, the use of a zero entry provides for a scan made up of channels from multiple Scanners. This requires that all Scanners included in the scan are loaded with identical table values. The following table shows multiple card scanning operation, assuming all Scanners are in *TwoWire* configuration.

Scanner Number	Scan Table Entries							
	@0	@1	@2	@3	@4	@5	@6	@7
SMU4032 #0	0	0	8	0	0	0	0	1
SMU4032 #1	3	35	0	18	0	0	0	0
SMU4032#2	0	0	0	0	4	5	6	0
Selected scanner#, Ch.	#1, Ch3	#1, Ch35	#0, Ch8	#1, Ch18	#2, Ch4	#2, Ch5	#2, Ch6	#0, Ch1

Figure 4-10. Contents of the Scan List table control Auto Scan sequence.

Use the `SCANSetScanList()` function to write to the Scanner hardware a channel value and `SCANGetScanList()` to read a value.

4.2.8 Locating Shorted Channel

The `SCANGetShortedChannel()` function scans all channels, and returns the first channel which is shorted. This can be used to detect a channel to see if it is the appropriate one, be it at the scanner input connector or at the optional terminal block. It is particularly useful in locating and identifying channel connection at the application end of a wiring harness.

4.2.9 Self Cleaning the Relay Contacts

Using the `SCANCleanRelays()` function, the Scanner can clean each relay contact. It does this using a specially designed on-board stimulus source along with a series of vibrations. This operation causes deposits of contaminants such as polymer deposits as well as oxides to be removed. It also solves a common relay problem involving thin film of insulating deposits, which accumulate particularly on relays which have not been used for a while. It does this by pinching through this film using an on-board high voltage source. Performing this function on a regular basis will improve the scanning system's reliability and repeatability and prolongs contact life. All relays including

Channel relays, Configuration relays and Tree relays are cleaned by this function. The SMX40L test module must be in place in order to perform this operation. This function returns an error if the test connector is absent. Cleaning takes about 10 seconds.

4.2.10 Integrity Test

The Integrity test is a quick verification tool. The **SCANTestChanIntegrity()** function tests the integrity of the specified channel relay, by verifying that the currently set actuation and release times are adequate. The release time is assumed to be 1/2 of the actuation time. This procedure closes the relay, waits for a time equal to the actuation time, then it tests for contact closure on both contacts. Next it opens the channel relay, waits for a period equal to the release time, and verifies that the relay is open.

This test does not include bounce test. Use the **SCANTestChannelRelay()** and **CANTestConfigRelay()** functions for a more comprehensive test. This test does not verify bounce

The test connector must be present in order to carry this operation. If the test connector is not present, this function returns an error. Cleaning takes about 13 seconds.

4.2.11 Self Tests

These comprehensive tests consist of the Channel Relays test and the Configuration Relays test. They provide the confidence of knowing that the Scanner is in good repair, and can continue in its operation. The two tests are applied to a single channel: configuration or tree relay. These tests diagnose excessive bounce, open failure, and short failure. If no failure is detected, these functions measure and return the actuation time for the selected relay. The actuation time measured includes closure time plus bounce time. The results of these tests can be used to fine tune the scanner for maximum switching time performance by either, setting the highest relay's actuation time (using the **SCANSetActuationTime()** function), or setting individual actuation time prior to selecting each channel. The value of the measured actuation time is an indicative of the condition of the selected relay. A value higher than that specified for the Scanner is an indication of relay deterioration, which may require relay replacement. These tests require the test connector.

Channel Relays Test

Each Channel may be tested using the **SCANTestChannelRelay()** function. The Channel parameter can be a value between 1 and 32. If no failure is detected, this function returns the Actuation time for the relay.

Configuration Relays Test

Each Configuration and Tree relay may be tested using the **SCANTestConfigRelay()** function. Configuration relay can be AtoA, BtoA, BtoB/Ch33, CtoA, CtoC/Ch34, DtoC, or DtoD/Ch35. See the *ScanUser.H* file for definitions of these relays. If no failure is detected, this function returns the Actuation time for the relay.

4.2.12 Setting Actuation Time Parameter

The Actuation time includes the time a relay closes and settles. It is made up of both, the Operate time and Bounce time. Each relay has different actuation time, and therefore it is a good idea to keep the default 10ms Actuation time. Alternatively, one can measure all relays, and set the Actuation time to the slowest relay of the Scanner. Actuation time may vary with age, and could be an indication of fatigue of the relay. Configuration and Tree relays may be measured using the **SCANTestConfigRelay()** function and the channel relays using the **SCANTestChannelRelay()** function. Set the Actuation time using **SCANSetActuationTime()** and read it with the **SCANGetActuationTime()** function. The SMU4030 family of Scanners have a special active Release (drop-out time) circuit, which forces all relays to release much faster than operate. For this reason, the Actuation time also includes the release of a currently closed channel, in a break before make multiplexing. In configurations such as the *TwoWire*, *FourWire*, *SixWire*, *TwoGroups*, and *FourGroups*, the Actuation time includes both the release and operate processes. When enabled, the trigger output signal corresponds to the Actuation time. The trigger polarity may be set positive or negative to provide means for triggering external devices when the switching is settled. Further, the Actuation time maybe set to a value higher than necessary for relay actuation to provide delayed trigger. Actuation time may be set to a value between 1ms to 850ms.

4.2.13 Setting Step Time Parameter

This parameter is the Auto Scan period, or step time. It maybe set to a value between 1ms to 850ms. On power up, it defaults to 100ms. It is the channel to channel scan time in Auto Scan. Use **SCANSetStepTime()** to set it and

SCANGetStepTime() to read it. When enabled during Auto Scan, the trigger output signal will have a period equal to the Step Time, and depending on trigger polarity, a negative or positive pulse corresponding to the Actuation time. The trigger can be used to provide means for triggering external devices with the desired delay. For example, consider a case where the Actuation time has been set to 50ms, but the relays actually settle within 10ms. In this scenario, additionally consider a Step time that has been left at its default value of 100ms, along with a trigger output that has been enabled and set for positive sense. With these parameters, the rising edge of the trigger output is delayed by 40ms (50ms – 10ms) from the time the relays are closed and settled, and the scanning speed is 10 channels per second (1/100ms).

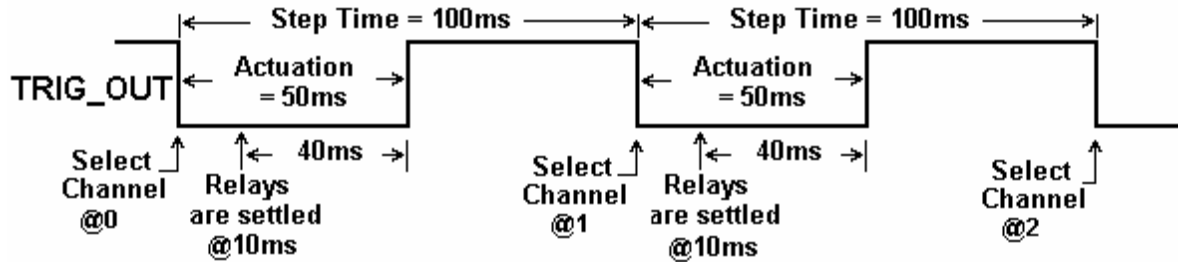


Figure 4-11. Step Time and Actuation Time parameters relations.

4.2.14 Thermocouple Temperature Measurements

When making thermocouple temperature measurements, it is important to use the SMX40T. It is isothermal, meaning that an even temperature is maintained across all terminals. The SMX40T has an active temperature sensor. It is connected to the channel 35 (D-bus), and provides for cold junction (or reference junctions) temperature measurement. When using the SMX40T this channel is not available for anything else. By configuring the 4032 to *ColdJunction*; **SCANSetConfig(ColdJunction)**, channel 35 gets connected to the A-Bus terminals, and can be measured by a Digital Multimeter. Channel 35 may also be selected while the 4032 is configured for *TwoWire*, by an explicit selection; **SCANSelectChannel(35)**.

The temperature of the Isothermal Block, or the, or the temperature where the thermocouples are connected, is required in calculating the thermocouple temperature. Most of the Signametrics Digital Multimeters have a measurement function that is capable of both, measuring and calculating the sensor temperature, as well as linearizing thermocouple measurements. Following the selection of the sensor; **SCANSetConfig(ColdJunction)** use the **DMMReadCJTemp** of the 2040 or 2060 series DMMs to perform the Cold Junction temperature measurement. Once measured, these DMMs store this temperature value for subsequent thermocouple measurements.

4.3 Polled Type Operations

Since the Scanner has its own controller, it is capable of processing operations, such as selecting a channels or executing a long test procedure, without help from the PC. For instance, when the **SCANSelectChannel()** function is issued, the DLL waits for the completion of the operation. It actually waits for the Scanner to complete the operation. In response to the command the Scanner opens the appropriate channels, closes the selected channel, and then waits for Actuation time prior to responding with 'operation complete' to the PC (DLL). When using the **polled** version of the above command, **SCANSelectChannelCmd()**, no waiting takes place. The command is sent to the Scanner and the driver (DLL) does not wait for response. A test program may take advantage of this. It can issue the **polled** command, then perform various other tasks, such as setting a DMM range, then check the Scanner for completion of the polled command by using **SCANReady()**. When the last command returns **TRUE**, it means that the Scanner is ready to accept a new command, and in the case of channel selection, the selected channel is closed and settled. If **SCANReady()** returns **TRUE**, it should not be used again until the next polled command is issued.

The following is a list of all **polled** functions: **SCANSelectChannelCmd()**, **SCANAutoScan()**, **SCANTrigAutoScan()**, **SCANTrigScan()** and **SCANStepCmd()**.

4.4 Interfacing to the SMU2060 USB DMM

The SMU4030 series Scanners are designed to interface to the SMU2060, and SMU2064 Digital Multimeters. The following section describes both, the hardware interface and the software functions used to implement a synchronized operation.

4.4.1 Triggering the SMU2060 DMMs

The SMU2060 and SMU2064 can be triggered to measure selected Scanner channels. The interface requires a connection between the SMU4033/30 TrigComm and Common lines, and two interface wires connected to the SMU type DMM's Trigger input. Once connected, the Scanner can be setup to produce a trigger signal for each relay selection operation. The various auto scanning operations can run independently from the computer, whereby the Scanner selects a list of channels stored in its Scan List table, and the DMM is triggered to take measurements following each channel selection.

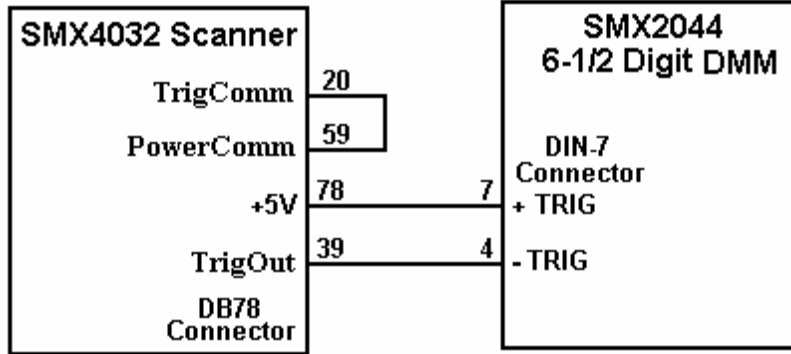


Figure 4-12. Trigger interfacing connection to an SMU2040 USB DMMs.

4.4.2 Multiplexing with the SMU2040 DMMs

For two wire measurements, the SMU204X DMMs must be connected to the A-Bus of the scanner, or to both, the A-Bus and C-Bus for 4-Wire measurements. It is important to consider system-settling time when making measurements. Time delays exist in any measurement system. These delays are contributed by various sources. These include the relay Actuation times, the DMM input settling and wiring burden. The latter is primarily due to capacitance, and will vary with the type of measurement. For instance, when making high value Ohms measurements the DMM current source level could contribute significant delay due to the capacitance charge time. For example, with 1,000pf cable capacitance, the source current of the SMU2044 DMM using the 33MΩ range, is 0.1μA which translates to 33ms ($dt = C \cdot dV/I$). It is also recommended to set the appropriate number of settling measurements for the DMM (a minimum of 4 is recommended).

4.4.3 Interface Commands and Timing

The sequence required for the SMU2040 DMM to make triggered measurements that are generated by the SMU4032, start by setting up the SMU4032. Set the SMU4032 desired configuration, with Trigger Output enabled and positive polarity. Selecting a channel will result in the generate a positive pulse with a duration equal to the Actuation time. This could be generated by one of the scanning operations, or simply by sending channel selection commands to the SMU4032. The SMU2040 must also be set up for triggered readings by using the **DMMSetTrigRead()** command. In the following example, the SMU2040 must send readings during the scan. Since it's on board Fifo is limited to 5 readings, and the DMM must continue to send all readings during the scan, it is therefore important to have a tight loop that reads the measurements fast enough. Refer to Figure 4-12 for proper trigger connection.

```

SCANTriggerOutState(iScan, Enabled, PosEdge)      ' Set trigger output to Positive edge.
iReadings = 10                                    ' Total number of measurements to take
DMMSetTrigRead(iDmm, 4, iReadings, NegEdge)      ' Total of 10 readings and 4 settling readings each
SCANAutoScan(iScan, iSteps)                       ' Start auto scan
For l = 0 to iReadings - 1                          ' read measurements as they come
    While DMMReadTrigVal(iDmm, reading) = NO      ' wait for each reading and store it
        DoEvents
    Wend
Next
SCANOpenAllChannels(iScan)                         ' Good idea to open all channels when done

```

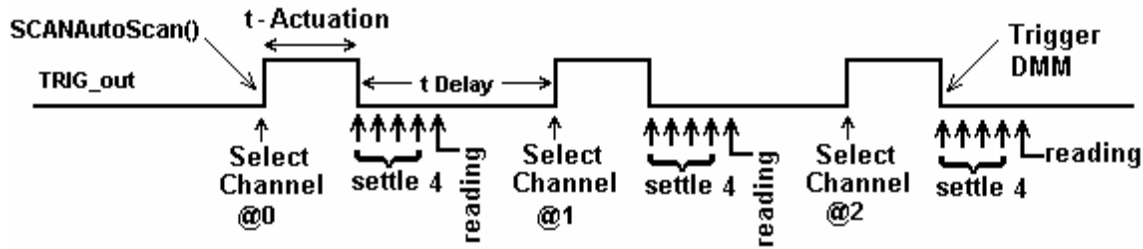


Figure 4-13. Triggered reading process and timing of SMU4032 Scanner and SMU2044 DMMs.

Unlike the previous example, **DMMSetBuffTrigRead()** is not time critical since the DMM saves all measurements to its on-board buffer, which is read after the scan is complete.

```

SCANTriggerOutState(iScan, Enabled, PosEdge)           ' Set trigger output to Positive edge.
iReadings = 50                                         ' Total number of measurements to take
DMMSetBuffTrigRead(iDmm, 4, iReadings, NegEdge)       ' Use 4 settling readings each
SCANAutoScan(iScan, iSteps)                           ' Set off AutoScan
While DMMReady(iDmm) = NO                             ' wait for the DMM to indicate capture
Wend                                                    ' completion of iReadings (available in buffer)
For l = 0 to iReadings -1                             ' read values stored in the buffer
    DMMReadBuffer(iDmm, Buffer(l) )                   ' Store each reading
Next
SCANOpenAllChannels(iScan)                            ' Good idea to open all channels when done
While SCANReady(iScan) = NO                          ' Since AutoScan is a polled operation,
DoEvents                                              ' Make sure Scanner is ready
Wend

```

There are several SMU2040 family commands to be considered for this operation:

DMMSetTrigRead(), **DMMSetBuffTrigRead()**, **DMMReadTrigVal()**, **DMMReady()**, **DMMReadBuffer()** and **DMMReadBufferStr()**. Do not forget to open all channels at the end of the scan, using **SCANOpenAllChannels()**.

Referring to figure 4.13, it is clear that the total time the DMM takes reading must be kept shorter than $t\text{-Delay}$ ($t\text{-Step} - t\text{-Actuation}$), for completion of the measurements prior to the selection of the next channel. That means that $t\text{-Step}$ must be greater than $t\text{-Actuation} + (n\text{Settling} + 1) / (\text{read. per sec.})$

4.5 Single Ended Applications

4.5.1 Point to Point Operation

This may not be obvious, but the SMU4030 and SMU4032 modules can be used for single ended applications. Taking advantage of the Universal mode, which allows any relay to be closed, a point to point switching can be implemented. Such application includes loaded board test commonly used by MDAs. The 32 differential inputs to the multiplexer become 64 independent lines (Ch1Hi, Ch1Lo, Ch2Hi, Ch2Lo, ... CH32Hi, Ch32Lo...) which can be connected to one of the four busses, A, B, C and D. These in turn can be connected to instruments such as the SMU2064 DMM. Further, it is also possible to configure it in such a way that the selected line can be connected to either the positive or the negative terminal of a DMM. Using multiple SMU4032, a high point test system can be constructed. In the following example, the Vlow input of the DMM is connected to both, the Bhi and the Dlo lines of the SMU4032. The Vhigh input of the DMM is connected to both, the Ahi and Clo of the SMU4032. The top eight lines of Figure 4-14 demonstrate a multiple SMU4032 system, where only a single line is connected to either High or Low terminal of an SMU2044 DMM. The following table provides the setting for a few single-ended connections to exemplify this application. The first three lines in the table depict a single SMU4032 usage where measurements between Ch32Lo line and Ch1Hi line are performed. The last two lines demonstrate how to make a measurement in both polarities between Ch1Hi line and Ch1Lo line.

Closed channel relay	Closed Bus relay	Closed Configuration relay	DMM Low terminal (Bhi & Dlo)	DMM High terminal (Ahi & Clo)
Ch1	A-to-A	NONE	NC	Ch1Hi
Ch32	A-to-A	D-to-A	NC	Ch32Hi
Ch1	C-to-C	C-to-A	NC	Ch1Lo
Ch32	C-to-C	D-to-C	NC	Ch32Lo
Ch1	B-to-B	B-to-A	Ch1Hi	NC
Ch32	B-to-B	D-to-A, B-to-A	Ch32Hi	NC
Ch1	D-to-D	D-to-A	Ch1Lo	NC
Ch32	D-to-D		Ch32Lo	NC
Ch1,Ch32	A-to-A ,D-to-D		Ch32Lo	Ch1Hi
Ch1	A-to-A, D-to-D	D-to-A	Ch1Lo	Ch1Hi
Ch1	C-to-C, B-to-B	C-to-A, B-to-A	Ch1Hi	Ch1Lo

Figure 4-14. Point-to-Point selection table. NC means no connection.

4.5.2 Single Ended Selector

By connecting AHi (pin 62) and CLo (pin 14) we can create a 68 channel single ended selector switch. The collector of this switch is pin 62 (Figure 4-14). Ch1 corresponds to Ch1Hi, Ch2 to Ch2Hi ...Ch10 is Ch10Hi etc.. Using Universal mode, make sure to select both, the appropriate channel relay as well as the various configuration relays.

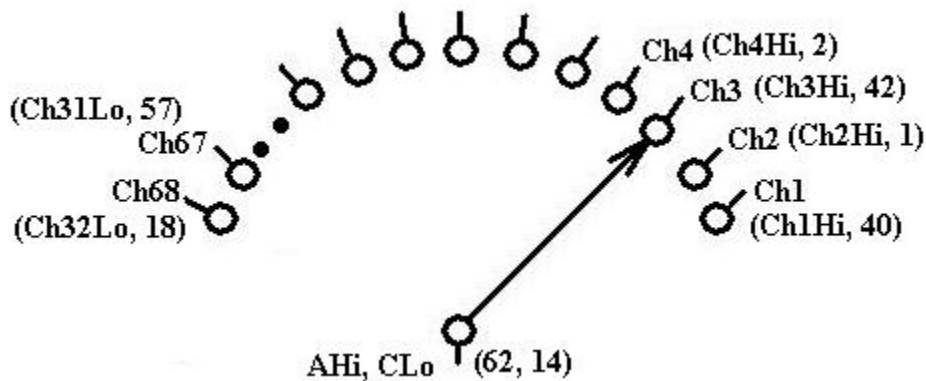


Figure 4-14. Single point switching is enabled by connecting A-bus High and C-bus Low.

5.0 SMU4032 Scanner Family Windows Interface

The windows release supports Windows 9X/ME/2000/XP\Millennium®. The primary means of controlling the scanners is a DLL file that interfaces with the USB bus via a .SYS file. All of these components are placed at the appropriate directories, and registry entries are made automatically during Setup.

5.1 Distribution Files

The main directory of the distribution diskette contains the Microsoft® Windows™ SMU4030 Scanner software. To install this software, enter the command "A:SETUP" in the "Run Program" menu of the Windows File Manager; or double-click on the SETUP.EXE file name from the File Explorer Tool Manager window. Most files on this diskette are compressed, and must be installed using the SETUP program.

The SMU4032 DLL is a protected-mode Microsoft® Windows™ DLL that will control the Signametrics SMU4032 and SMU4030. It is provided with a sample Visual Basic™ control-panel application (GUI) to demonstrate the Scanner and the interface to the DLL. Check the README.TXT file for more information about the files contained on the diskette. Some important files to note are:

<u>File</u>	<u>Description</u>
SMU4032.LIB	The Windows import library. Install in a directory pointed to by your LIB environment variable.
SMU4032.DEF	SM2040 driver DLL module definition file.
SMU4032.DLL	The 32 bit driver DLL. This should be installed either in your working directory, in the Windows system directory, or in a directory on your PATH . The installation program installs this file in your Windows system directory (usually C:\WINDOWS\SYSTEM for Win9X/ME and XP , or at C:\WINNT\SYSTEM32 for Windows 2000).
<u>File</u>	<u>Description</u>
SMU4032.H	Driver header file. Contains the definitions of all the Scanner's function prototypes for the DLL, constant definitions, and error codes. Install in a directory pointed to by your INCLUDE environment variable.
ScanUser.H	Header file containing all of the necessary Scanner's function parameters and configuration definitions to be used with the various functions.
Msvbvm50.dll	Visual Basic run-time interpreter. Usually, install in your C:\WINDOWS\SYSTEM (or equivalent) directory. If it is not already installed, run Msvbvm50.exe for proper extraction and registration.
SMU4032.vbw	Visual Basic project file
SMU4032.frx	Visual Basic binary form file
SMU4032.frm	Visual Basic file with main form
SMU4032.vbp	Visual Basic project file
4032glbl.bas	Visual Basic file with all global Scanner declarations

SMU4032.exe	Visual Basic Scanner control panel executable
Msvcr7.dll	System file. Installs in your C:\WINDOWS\SYSTEM directory.
Windrvr.vxd	Win98/95 Virtual Device Driver. Installs in your C:\WINDOWS\SYSTEM\VMM32 directory.
Windrvr.sys	Win NT/Win 2000 Virtual Device Driver. Installs in your C:\WINNT\SYSTEM32\DRIVERS directory.
SMU4032.INF	Plug-and-Play driver
Install.doc	Installation instructions in MS Word

During initialization (**SCANInit()**), the driver reads various parameters such as Scanner type (SM4020/22/40/42), and serial number etc..

5.2 Using the SMU4032 Driver with C++ or Similar Software

Install the **SMU4032.H** and **ScanUser.H** header files in a directory that will be searched by your C/C++ compiler for header files. These header files are known to work with Microsoft Visual C++™. To compile using Borland, you will need to convert the **SMU4032.DEF** and **SMU4032.LIB** using **ImpDef.exe** and **ImpLib.exe**, provided with the compiler. Install **SMU4032.LIB** in a directory that will be searched by the linker for import libraries. The SMU4032 software must be installed prior to running any executable code. Install the **SMU4032.DLL** in a location where either your program will do a **LoadLibrary** call to load it, or on the **PATH** so that Windows will load the DLL automatically. A common place for the DLL is at C:\WINDOWS\SYSTEM directory for Win9X/ME/XP or at C:\WINNT\SYSTEM32 directory for NT and Windows 2000.

In using the SMU4032 driver, first call **SCANInit** to initialize the scanner hardware and software. This function should only be used once. Call **SCANSetConfig** to set the Scanner to a desired configuration, be it *TwoWire*, *FourWire* etc. The Scanner function constants are defined in the **ScanUser.H** header file, and have names that clearly indicate the function they invoke.

Two functions are provided to set channel-relays, **SCANSelectChannel** and **SCANSetChannelRelay**. The first opens all channels in a group, and closes the specified channel. The second function is usable while the scanner is in the *Universal* configuration, and opens or closes a specified channel relay,.

Most functions accept a Scanner number parameter, which must be set to the **iScan** value when initializing the scanner with the **SCANInit**. For multiple Scanners this **iScan** value will be 0,1,2..n. Most functions return an error code that can be retrieved as a string using **SCANErrStr**.

5.2.1 Multiple Card Operations under Windows

5.2.1.1 Single .EXE operation

Accessing multiple Scanners from a single executable is the most common way for running up to 12 Scanners using the Windows DLL. A combination of several SMU403X can be controlled, as long as the single .EXE (Thread) is used to control all of the units. Make sure that prior to issuing commands to any Scanner, it is initialized using **SCANInit**. The **iScan** parameter is passed with each DLL command to define the unit to be accessed. Since this configuration utilizes the DLL to service all units, it must handle one command at a time. For example, when one Scanner's channel is selected (with a non polled command the DLL must finish the operation prior to addressing another Scanner. For improved performance, one can use the following:

5.2.1.2 Multiple .EXE operation

By having several copies of **SMU4032.DLL**, and renaming them, you may multiple Scanners with separate executables. For instance, having a copy named **SMU4032A.DLL** and one named **SMU4032B.DLL** in C:\WINDOWS\SYSTEM (Win98/95), and having two executable files, **MultiExeA.exe** and **MultiExeB.exe**, each

Signametrics 28

of the executables will run independently, making calls to the respective DLL. This can provide an execution throughput advantage over the method mentioned above. If using VisualBasic, the **MultiExeA.exe** source code should define *iScan* = 0, and **MultiExeB.exe** should define *iScan* = 1. In addition, the first EXE should declare the **SMU4032A.DLL** and the second should declare **SMU4032B.DLL**:

```

/*****
* Exmp4032.C Exmp4032.EXE
*
* A simple Windows .EXE example for demonstrating the SMU4032
* Scanners using "C"
* Sets Configuration to TwoWire, and select a channel.
*
*****
* Make sure SMU4032.lib is included in the libraries. For Microsoft
* Version 4.0 C++ and above, place under 'Source Files' in the
* Workspace, along side with Exmp4032.c
* PROJECT SETTINGS:
*
* /nologo /ML /W3 /GX /O2 /D "WIN32" /D "NDEBUG" /D "_CONSOLE" /D "_MBCS"
* /FR"Release/" /Fp"Release/Exmp4032.pch" /YX /Fo"Release/" /Fd"Release/" /FD /c
*
* Copy both SMU4032.DLL and SMU4032.LIB to the project directory.
*
*****/

// #define WINAPI __stdcall
#include <windows.h>
#include <string.h>
#ifdef _Windows
    #define _WINDOWS
#endif
#include "SMU4032.H"           // functions declarations and error codes.
#include "ScanUser.H"        // All functions, range and rate info and function declarations.
int main(void){
    int I, iScan = 0;         // Address first Scanner in the system
    char Read[16];
    char strMsg[256];
    i = SCANInit(iScan);     // initialize SMU4032
    if(i<0)
        MessageBox(0,"Initialization ERROR !", "Startup SMU4032 DLL",MB_OK);    // Show Error
    SCANSetConfig(iScan,TwoWire);           // Set to TwoWire configuration
    SCANSelectChannel(iScan,32);           // Close Channel 32 relay, and necessary path relays
    MessageBox(0,strMsg, "Channel 32 is closed and settled",MB_OK);           // Show status
    return 0L;
}

```

5.3 Visual Basic Front Panel Application

The Visual Basic front panel application, **SMU4032.EXE**, is an interactive control panel for the SMU4032 Scanners. When it loads it will take a few seconds to initialize before the front panel is displayed.

The push buttons labeled and menus provide means to configure the Scanner and select channels. The panel has several capabilities which are enabled and disabled according to the type of Scanner detected, and the current configuration.

The source code file **GLOBAL.BAS** (in the installed **VisBasic** sub directory) contains the function declarations and the various ranges, rates and other parameters that are required. These definitions are the duplicates of the "C" header files required to write Visual Basic applications which interact with the driver DLL, along with some global variables required for this particular front-panel application.

5.3.1 Visual Basic Simple Application

The following is a very simple panel application for VisualBasic, which is included with the installed software. The two files, Global.Bas and SimplePanel.frm are presented here. This GUI panel contains six objects; a text box to input channel to select, three option buttons to set configuration, a button to apply the channel selection and a text line for displaying errors.

Global.bas module file contents:

```
Option Explicit
'function declarations
Declare Function SCANInit Lib "SMU4032.dll" _
    (ByVal nScanner As Long) As Long
Declare Function SCANSetConfig Lib "SMU4032.DLL" _
    (ByVal nScanner As Long, ByVal Config As Long) As Long
Declare Function SCANErrString Lib "SMU4032.DLL" _
    (ByVal nError As Long, ByVal errString As String, ByVal stringlen As Long) As Long
Declare Function SCANSelectChannel Lib "SMU4032.DLL" _
    (ByVal nScanner As Long, ByVal nChannel As Long) As Long

'Configuration definitions for :
Global Const TwoWire = 0
Global Const FourWire = 7
Global Const SixWire = &H1E
' Since the DLL is written in "C", where TRUE = 1 and FALSE = 0
' and VB has -1 for True, use YES and NO instead of True and False
Global Const YES = 1
Global Const NO = 0

'Error handling code:
Public Sub ShowFault(status As Long)
    Dim errtext As String * 48
    Dim errcode As Long
    Dim start As Single
    If status < 0 Then
        errcode = SCANErrString(status, errtext, 48)
        SimplePanel.TextLine.Text = errtext
        If status <> -3 Then 'if not autorange wait with message
            start = Timer
            While ((Timer - start) < 2#) 'delay for 1.1 seconds
                DoEvents
            Wend
        End If
        SimplePanel.TextLine.Text = ""
    End If
End Sub
```

SimplePanel.frm Form file contents:

```
Option Explicit

Private Sub CommandSelect_Click()
    ShowFault SCANSelectChannel(0, Lng(TextChan.Text)) 'Select the channel
End Sub

Private Sub Form_Load()
    Dim errtext As String * 64
    Dim i, j, k As Long
    i = SCANInit(0)           'Initialize scanner 0
End Sub

Private Sub Option2W_Click()
    ShowFault SCANSetConfig(0, TwoWire)
End Sub

Private Sub Option4W_Click()
    ShowFault SCANSetConfig(0, FourWire)
End Sub
```


Description Initiate AutoScan operation.

#include "SMU4032.H"

int SCANAutoScan(int iScan, int iPoints)

Remarks

Run Auto Scan procedure from the Scanner's stored ScanList. This operation is carried out by the scanner's on-board processor. It uses the Scan List table and timing parameters to perform a complete scan sequence. If necessary, the scan may be terminated by sending the **SCANAbort()** command. The Scanner scans *iPoints* of consecutive locations from the ScanList, starting with the point stored in location 0, and up to location 191 (max of 192 for *iPoints*). Steptime and Actuation time values must be set prior to executing this function. The Scan List must contain channel information for at least *iPoints* prior to using this command. Steptime must be greater than the Actuation time. This function is not applicable for the following configurations: *Universal* and *Disabled*.

Being a polled function, following this command uses **SCANReady()** to test for completion of the operation. No new H/W access command should be issued prior to **SCANReady()** return of **TRUE**. An exception is the **SCANAbor()** command which terminates the current operation.. Following reception of this command, the Scanner hardware enters a busy state. When **SCANReady()** returns **TRUE** do not use it again.

<u>Parameter</u>	<u>Type/Description</u>
<i>iScan</i>	int Identifies the Scanner. Scanners are numbered starting with zero.
<i>iPoints</i>	int The number of points in the scan. This number must be between 1 and 192, inclusive.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
SCAN_OKAY	Operation successfully terminated
Negative Value	Error code.

Example

```
i = SCANAutoScan(0,22); // Scan 22 points;  
while(!SCANReady(0)); // Wait for completion of scan
```

SCANCleanRelays

H/W access Command Polled Command

Description Clean all relays.

#include "SMU4032.H"

int SCANCleanRelays(int iScan)

Remarks

This function in conjunction with the SMX40L module, provides the means to clean all relays. The SMX40L test connector must be present to clean all contacts. Cleaning takes about 10 seconds. Vibrating of the contacts at varying switching patterns, while sourcing a high voltage/low current signal, causes deposits of contaminants as well as oxidations to bounce off or be pinched through, thus resulting in cleaner contacts. Doing this on a regular basis will prolong the contacts' life. Both Channel relays and Configuration relays are cleaned.

<u>Parameter</u>	<u>Type/Description</u>
<i>iScan</i>	int Identifies the Scanner. Scanners are numbered starting with zero.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
SCAN_OKAY	Scanner is OK.
Negative Value	Error

Example

```
I = SCANCleanRelays(0); // Issue clean command (returns in 10s)
```

SCANClose

H/W access Command Polled Command

Description Close the USB bus for the specified Scanner. Not for user applications.

int SCANClose(*int nScan*)

Remarks This function is limited for servicing the Scanner. It has no use in normal operation. See also SCANOpen() function.

<u>Parameter</u>	<u>Type/Description</u>
<i>nScan</i>	int Identifies the Scanner number

Return Value Integer error code.

<u>Value</u>	<u>Meaning</u>
SCAN_OKAY	Operation successfully completed.
Negative Value	Error code

Example

```
int status = SCANClose(0);
```

SCANDelay

H/W access Command Polled Command

Description Wait for a given time.

```
#include "SMU4032.H"
```

```
int SCANDelay(double dTime)
```

Remarks Delay of *dTime* seconds. *dTime* must be a positive double floating point number between 0.0 and 100.0 seconds.

<u>Parameter</u>	<u>Type/Description</u>
<i>dTime</i>	double Delay time in seconds.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
SCAN_OKAY	Operation successfully terminated
Negative Value	Error code

Example

```
SCANDelay(1.2); /* wait for 1.2 Sec */
```

SCANErrString

H/W access Command Polled Command

Description Return the string describing the error.

```
#include "SMU4032.H"
```

```
int SCANErrString(int iErrorCode, LPSTR lpszError, int iBuffLength)
```

Remarks This function returns a string containing the error description, which corresponds to the integer error, code *iErrorCode*. The error string is placed at *lpszError*.

<u>Parameter</u>	<u>Type/Description</u>
<i>iErrorCode</i>	int Error code.
<i>iBuffLength</i>	int The maximum available length of the string buffer
<i>lpszError</i>	LPSTR Points to a buffer (at least 40 characters long) to hold the error string.

Return Value The return value is the length of the error string or one of the following constants.

<u>Value</u>	<u>Meaning</u>
Positive Value	Length of returned error string.
Negative Value	Error code

Example `char cBuf[48];`
 `int length = SCANErrString(-3, cBuf, 48);`

SCANGetActuationTime

H/W access Command Polled Command

Description Return the currently set actuation time.

#include "SMU4032.H"

int SCANGetActuationTime(int *iScan*, double FAR **lpdAct*)

Remarks This function returns a double floating value that is the currently set relay actuation time for the selected scanner.

<u>Parameter</u>	<u>Type/Description</u>
<i>iScan</i>	int Identifies the Scanner. Scanners are numbered starting with zero.
<i>lpdAct</i>	double FAR * Pointer where t-Actuation value is to be saved.

Return Value Integer error code.

<u>Value</u>	<u>Meaning</u>
SCAN_OKAY	Operation successfully completed.
Negative Value	Error code

Example `double FAR tAct; int i = SCANGetActuationTime(0, &tAct);`

SCANGetBusInfo

H/W access Command Polled Command

Description Returns the USB Bus and Slot numbers for the selected SCANNER.

int SCANGetBusInfo(int *nScan*, int **bus*, int **slot*)

Remarks This function reads the USB *bus* and *slot* numbers for the selected Scanner. It provides means to relate the physical location to the *nScan* value by detecting the location of a Scanner in the USB system tree. This function actually scans the hardware rather than look up the information in the registry.

<u>Parameter</u>	<u>Type/Description</u>
<i>nScan</i>	int Identifies the Scanner. These are numbered starting with zero.
<i>bus</i>	int * a pointer to integer at which the bus number is stored (0 to 255)
<i>slot</i>	int * A pointer to an integer where the slot number is stored (0 to 15)

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
SCAN_OKAY	Operation was successful.
Negative number	Error code

Example `int bus, slot; // Find on which bus, and slot the DMM is at
SCANGetCalDate(3, &bus, &slot); // SCANNER#3`

SCANGetConfig

H/W access Command Polled Command

Description Read the current configuration of the scanner.

int SCANGetConfig(int iScan)

Remarks This function reads the current configuration settings from the Scanner hardware. The configurations are defined in the ScanUser.H file.

<u>Parameter</u>	<u>Type/Description</u>
<i>iScan</i>	int Identifies the Scanner. Scanners are numbered starting with zero.

Return Value The return value is an integer configuration code or an error code

<u>Value</u>	<u>Meaning</u>
Positive Value	Configuration code
Negative Value	Error code

Example `int iConfig = SCANGetConfig(0);`

SCANGetGrdVer

H/W access Command Polled Command

Description Get Scanner firmware version.

#include "SMU4032.H"

int SCANGetGrdVer(int iScan)

Remarks This function reads and returns the Scanner's firmware version, which is the on-board Microcontroller's S/W version. This integer should be divided by 10 to get the actual version. i.e., 15 equals version 1.5.

<u>Parameter</u>	<u>Type/Description</u>
<i>iScan</i>	int Identifies the Scanner. Scanners are numbered starting with zero.

Return Value Integer version value or an error code.

<u>Value</u>	<u>Meaning</u>
--------------	----------------

Positive Value Version

Negative Value Error code

Example `firmware_ver = SCANGetGrdVer(0);`

SCANGetHwVer

H/W access Command Polled Command

Description Get the hardware version of the Scanner.

#include "SMU4032.H"

int SCANGetHwVer(**int** *iScan*)

Remarks This function returns the Scanner hardware version. A returned value of 0 corresponds to Rev_, 1 corresponds to Rev_A, 2 to Rev_B etc.

<u>Parameter</u>	<u>Type/Description</u>
<i>iScan</i>	int Identifies the Scanner. Scanners are numbered starting with zero.

Return Value Scanner hardware code or an error code.

<u>Value</u>	<u>Meaning</u>
Positive value	Hardware version code
Negative Value	Error code

Example `int HWVer = SCANGetHwVer(0);`

SCANGetID

H/W access Command Polled Command

Description Get Scanner ID code.

#include "SMU4032.H"

int SCANGetID(**int** *iScan*)

Remarks This function returns the Scanner's identification code. Each Scanner has a unique ID code, which is part of the serial number.

<u>Parameter</u>	<u>Type/Description</u>
<i>iScan</i>	int Identifies the Scanner. Scanners are numbered starting with zero.

Return Value Integer card ID code (serial number) or an error code.

<u>Value</u>	<u>Meaning</u>
--------------	----------------

Positive Value Scanner ID code

Negative Value Error code

Example `int id = SCANGetID(0);`

SCANGetManDate

H/W access Command Polled Command

Description Get Manufacturing date stamp from the Scanner hardware

#include "SMU4032.H"

int SCANGetManDate(int iScan, int *month, int *day, int *year)

Remarks This function returns the Scanner's manufacturing date, which is read from the hardware. The month, day and year are returned as integers. This is used to track the Scanner to a specific manufacturing date code.

<u>Parameter</u>	<u>Type/Description</u>
<i>IScan</i>	int Identifies the Scanner. Scanners are numbered starting with zero.
<i>Month</i>	int * A pointer to an integer where the month is stored
<i>Day</i>	int * A pointer to an integer where the day is stored
<i>Year</i>	int * A pointer to an integer where the year is stored

Return Value Integer error code.

<u>Value</u>	<u>Meaning</u>
SCAN_OKAY	Operation was successful.
Negative Value	Error code

Example `int month, day, year, status
status = SCANGetManDate(0, &month, &day, &year);`

SCANGetScanList

H/W access Command Polled Command

Description Get a value from the on-board Scan List table.

#include "SMU4032.H"

int SCANGetScanList(int iScan, int iAddress)

Remarks This function returns an integer value corresponding to the *iAddress* entry to the Scanner's on-board scan list. This value should be 0 to 35 corresponding to an open channel or channel 1 to 35. This value is written to the table by **SCANSetScanList()**.

<u>Parameter</u>	<u>Type/Description</u>
<i>iScan</i>	int Identifies the Scanner. Scanners are numbered starting with zero.
<i>iAddress</i>	int The address of the scan list to be read.

Return Value Integer error code..

<u>Value</u>	<u>Meaning</u>
SCAN_OKAY	Operation successfully completed.
Negative Value	Error code

Example

```
int ScanLst[192];
ScanLst[3] = SCANGetScanList(0, 3); //Get the 4th entry
```

SCANGetShortedChannel

H/W access Command Polled Command

Description Returns the channel number of the shorted channel.

#include "SMU4032.H"

int SCANGetShortedChannel(int *iScan*)

Remarks This function searches and detects a shorted channel, starting with channel 1. It returns the first channel which is found to be shorted. If none is found, the returned value is 0. A valid returned value must be 0 to 35. 0 indicates no shorted channel is found, 1 to 35 correspond to channels 1 to 35 being shorted. It can be used to identify the connector pins while constructing a harness.

<u>Parameter</u>	<u>Type/Description</u>
<i>iScan</i>	int Identifies the Scanner. Scanners are numbered starting with zero.

Return Value The return value corresponding to the shorted channel or an error code.

<u>Value</u>	<u>Meaning</u>
0	No shorted channel found.
1 to 40	Shorted channel number
Negative Value	Error code

Example

```
int shorted = SCANGetShortedChannel(0);
```

SCANGetStepTime

H/W access Command Polled Command

Description Get the currently set Auto Scan step time.

```
#include "SMU4032.H"
```

```
int SCANGetStepTime(int iScan, double FAR *lpdTstep)
```

Remarks This function returns a double floating value that is the currently set step time.

<u>Parameter</u>	<u>Type/Description</u>
<i>iScan</i>	int Identifies the Scanner. Scanners are numbered starting with zero.
<i>lpdTstep</i>	double FAR * Pointer where the Step time value is to be saved.

Return Value Integer error code..

<u>Value</u>	<u>Meaning</u>
SCAN_OKAY	Operation successfully completed.
Negative Value	Error code

Example

```
double FAR Tstep;  
int status = SCANGetStepTime(0, &Tstep);
```

SCANGetTriggerIn

H/W access Command Polled Command

Description Read the state of the Trigger Input line.

```
#include "SMU4032.H"
```

```
int SCANGetTriggerIn(int iScan)
```

Remarks This function returns the logic state of the Hardware trigger input line to the Scanner.

<u>Parameter</u>	<u>Type/Description</u>
<i>iScan</i>	int Identifies the Scanner. Scanners are numbered starting with zero.

Return Value Scanner hardware code or an error code.

<u>Value</u>	<u>Meaning</u>
0 or Positive value	State of the Trigger input line.
Negative Value	Error code

Example

```
int Trig = SCANGetTriggerIn(0);
```

SCANGetType

H/W access Command Polled Command

Description Get the type of the Scanner.

Signametrics

```
#include "SMU4032.H"
```

```
int SCANGetType(int iScan)
```

Remarks This function returns the Scanner type.

<u>Parameter</u>	<u>Type/Description</u>
<i>iScan</i>	int Identifies the Scanner. Scanners are numbered starting with zero.

Return Value Scanner type Integer or an error code.

<u>Value</u>	<u>Meaning</u>
4030	SMU4030 is at iScan slot
4032	SMU4032 is at iScan slot
Negative Value	Error code

Example

```
int type = SCANGetType(0);
```

SCANGetVer

H/W access Command Polled Command

Description Return the Scanner's DLL software driver version.

```
#include "SMU4032.H"
```

```
int SCANGetVer(double FAR *lpfResult )
```

Remarks This function returns the Scanner's software driver version, which is a double floating value.

<u>Parameter</u>	<u>Type/Description</u>
<i>lpfResult</i>	double FAR * Pointer to the location which holds the version.

Return Value Integer value version code or an error code.

<u>Value</u>	<u>Meaning</u>
SCAN_OKAY	Operation successfully completed.
Negative Value	Error code

Example

```
int status; double d;  
status = SCANGetVer(&d);
```

SCANInit

H/W access Command Polled Command

Description Initialize the scanner.

```
#include "SMU4032.H"
```

```
int SCANInit(int iScan)
```

Remarks This function must be the first function to be executed. It loads the driver and opens the specified Scanner on the USB bus. The first one being 0, the second 1, etc. It also initializes the hardware and software and sets the scanner to Disabled configuration.

<u>Parameter</u>	<u>Type/Description</u>
<i>iScan</i>	int Identifies the Scanner. Scanners are numbered starting with zero.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
SCAN_OKAY	Scanner was initialized successfully.
Negative Value	Error code

Example

```
int i = SCANInit(0); // Initialize the first Scanner
```

SCANIsInitialized

H/W access Command Polled Command

Description Return the active status of the Scanner.

```
#include "SMU4032.H"
```

```
int SCANIsInitialized(int iScan)
```

Remarks This function returns the status of the Scanner. If the scanner was previously initialized, it is an active one, and TRUE is returned. If the scanner was not initialized and is available to be opened, the return value is FALSE, indicating the Scanner maybe initialized and addressed. This function is used for managing multiple scanners in a system.

<u>Parameter</u>	<u>Type/Description</u>
<i>iScan</i>	int Identifies the Scanner. Scanners are numbered starting with zero.

Return Value TRUE, FALSE or an error code.

<u>Value</u>	<u>Meaning</u>
TRUE	Scanner is initialized and active.
FALSE	Scanner is not initialized.
Negative Value	Error code

Example

```
int active = SCANIsInitialzied(0);
```

SCANOpenAllChannels

H/W access Command Polled Command

Description Open all channel relays.

```
#include "SMU4032.H"
```

```
int SCANOpenAllChannels(int iScan)
```

Remarks This function opens all **channel** relays. It does not effect the configuration of the Scanner. It may be used following one of the scanning operations, to make sure all channels are left open. Or it can be used any time when it is necessary to open all channels. This function is not applicable while in *Universal* or *Disabled* configurations. When in the Universal configuration, use the **SCANSetChannelRelay** and **SCANSetConfigRelay** functions to open and close relay. Selecting the *Disabled* or *Universal* configurations using **SCANSetConfig** will open all relays, including configuration relays.

<u>Parameter</u>	<u>Type/Description</u>
<i>iScan</i>	int Identifies the Scanner. Scanners are numbered starting with zero.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
SCAN_OKAY	Operation successfully completed.
Negative Value	Error code

Example

```
int active = SCANOpenAllChannels(0);
```

SCANOpen

H/W access Command Polled Command

Description Open the USB bus for the specified Scanner. Not for user application.

```
int SCANOpen(int nScan)
```

Remarks This function is limited for servicing the Scanner. It has no use in normal operation.. See also **SCANClose ()** function.

<u>Parameter</u>	<u>Type/Description</u>
<i>nScan</i>	int Identifies the Scanner number.

Return Value Integer error code.

<u>Value</u>	<u>Meaning</u>
SCAN_OKAY	Operation successfully completed.

Negative Value Error code

Example `int status = SCANOpen(0);`

SCANReady

H/W access Command Polled Command

Description Return the ready state of the Scanner following a polled operation.

#include "SMU4032.H"

int SCANReady(int iScan)

Remarks Following the completion of long or complex polled commands, the scanner indicates it is ready to accept a new command by issuing a task complete message. Functions requiring the use of the SCANReady() command include; **SCANSelectChannelCmd()**, **SCANAutoScan()**, **SCANTrigAutoScan()**, **SCANTrigScan()**, and **SCANCleanRelays()**. The **SCANReady()** function checks returns TRUE if ready, and FALSE otherwise. Once a TRUE status is returned, the **SCANMMReady()** function should not be used again until a new polled command requires it.

<u>Parameter</u>	<u>Type/Description</u>
<i>iScan</i>	int Identifies the Scanner. Scanners are numbered starting with zero.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
TRUE	(=1) Scanner is done and ready to accept new commands.
FALSE	(=0) Scanner is not ready.
Negative Value	Error code

Example `double Buffer[10];
SCANAutoScan(0,10);
while(! SCANReady(0));
for(i=0;i<10 ; i++) Buffer[i] = SCANReadBuffer(0);`

SCANSelectChannel

H/W access Command Polled Command

Description Select channel command.

#include "SMU4032.H"

#include "ScanUser.H"

int SCANSelectChannel(int iScan, int iChan)

Remarks This is the primary channel selection function for the multiplexed configurations of the scanner. It opens currently closed channel(s) in a configured group and then closes *iChan*. Depending on the current configuration, it will open and close a different number of relays. In *TwoWire* configuration, all relays open prior to closing the *iChan*. If the Scanner is in the *FourWire* configuration, the *iChan* value can be between 1 and 16. In addition to opening the currently closed channels, it will close the two channel relays in accordance with the *FourWire* switching scheme. Configuration relays will also be set as

required for accessing the selected channel. This command is applicable in the following configurations: *TwoWire*, *FourWire*, *SixWire*, *TwoGroups*, and *FourGroups*. It opens and closes all relays within one t-Actuation. Issuing this command will prevent the execution of any other function for a time specified in the currently set Actuation time, t-Actuation.

<u>Parameter</u>	<u>Type/Description</u>
<i>iScan</i>	int Identifies the Scanner. Numbered starting with zero.
<i>iChan</i>	int Channel number Range: 1 to 35 depending on Scanner type and set configuration. 0 causes opening of all channels in a group.

Return Value Integer code.

<u>Value</u>	<u>Meaning</u>
SCAN_OKAY	Operation successfully completed.
Negative Value	Error code

Example

```

SCANSelectChannel(0, 3); // Select a channel
DMMRead(0, @val); // measure input at Channel 3

```

SCANSelectChannelCmd

H/W access Command Polled Command

Description Polled select channel command.

#include "SMU4032.H"

int SCANSelectChannelCmd(int iScan, int iChan)

Remarks This is the polled version of the **SCANSelectChannel** function. It opens all channels in a group, then closes channel number *iChan*. This is a polled function. It sends a command to the Scanner but unlike **SCANSelectChannel**, unlike the **SCANSelectChannel** function it does not wait for completion of the operation. **SCANReady** **must** be used in order to verify the selected channel was closed and the scanner is ready for a new command. The time to ready is dictated by t-Actuation. It is necessary for **SCANReady** to return **TRUE** once for proper operation. For instance, if the actuation time is set to 10ms, and a wait of 100ms it taken following the execution of the **SCANSelectChannelCmd**, it should not be assumed the scanner is ready. **SCANReady** must be used and return **TRUE** to proceed. **SCANReady** not only checks for readiness, but it also clears some registers in preparations for the next command. It is important to note that the **SCANSelectChannelCmd** is context sensitive and depending on the current configuration will function differently. See **SCANSelectChannel** command for details. This command is limited to the following configurations: *TwoWire*, *FourWire*, *SixWire*, *TwoGroups* and *FourGroups*.

<u>Parameter</u>	<u>Type/Description</u>
<i>iScan</i>	int Identifies the Scanner. Numbered starting with zero.
<i>iChan</i>	int Channel number Range: 1 to 40 depending on Scanner type and set configuration

Return Value Integer code.

<u>Value</u>	<u>Meaning</u>
SCAN_OKAY	Operation successfully completed.
Negative Value	Error code

Example

```

SCANSelectChannelCmd(0, 3); // Select a channel.
DMMSetRange(0, 300mV)      //Set DMM to appropriate range
DMMRead(0, &Val);          //Flush a reading or two
While( ! SCANReady(0));    // Wait for Scanner to be ready
DMMRead(0, &reading);      // Read input

```

SCANSetActuationTime

H/W access Command Polled Command

Description Set relay actuation time value

```
#include "SMU4032.H"
```

```
int SCANSetActuationTime(int iScan, double dActuate)
```

Remarks This function sets the actuation time value to the scanner. The actuation time is the time it takes the scanner to select and deselect all relays. The default actuation time is set to 10mS. The allowed range is from 0.25ms to 800ms. The resolution is about 0.25ms. Setting Actuation time below the relay specified Bounce-Time could result in premature measurement with an open or unsettled relay.

<u>Parameter</u>	<u>Type/Description</u>
<i>iScan</i>	int Identifies the Scanner. Numbered starting with zero.
<i>dActuate</i>	double Sets the actuation time. Allowed value is between 0.25ms to 800ms.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
SCAN_OKAY	Function succeeded.
Negative Value	Error code

Example

```
status = SCANSetActuationTime(0, 0.05); // Set to 50ms
```

SCANSetChannelRelay

H/W access Command Polled Command

Description Close or open a channel relay.

```
#include "SMU4032.H"
#include "ScanUser.H"
```

```
int SCANSetChannelRelay(int iScan, int iState, int iChan)
```

Remarks This function provides means to close or open individual channel relays while the Scanner is in the *Universal* configuration. Issuing this command causes a single relay to open or close, with an actuation time of tActuation. In the Universal configuration,

multiple channel relays can be closed. With *iState* set to **OPEN**, open the *iChan* relay. Close it if *iState* is set to **CLOSE**. See ScanUser.H for definitions. See also **SCANSetConfigRelay** function.

<u>Parameter</u>	<u>Type/Description</u>
<i>IScan</i>	int Identifies the Scanner. Numbered starting with zero.
<i>IState</i>	int Closes a channel relay if equal to CLOSE , opens it if OPEN . (OPEN = 'O' or 0X4F and CLOSE = 'C' or 0X43)
<i>IChan</i>	int Channel number Range: 1 to 35 depending on selected configuration. If <i>iState</i> = OPEN and <i>iChan</i> = 0, open all channel relays.

Return Value Integer error code.

<u>Value</u>	<u>Meaning</u>
SCAN_OKAY	Operation successfully completed.
Negative Value	Error code

Example

```
SCANSetChannelRelay(0,OPEN,10); // Open channel 10 relay
for(I=1; I<=9; I++) // Close 1,2...,9
SCANSetChannelRelay(0, CLOSE,I);
```

SCANSetConfig

H/W access Command Polled Command

Description Set the Scanner configuration to one of the predefined modes.

```
#include "SMU4032.H"
#include "ScanUser.H"
```

```
int SCANSetConfig(int iScan, int iConfiguration)
```

Remarks This function sets the Scanner to one of the available configurations. It acts on both, the configuration relays and tree relays to set the Scanner for *TwoWire*, *FourWire*, *SixWire*, *TwoGroups*, *FourGroups*, *Universal*, *ColdJunction* or *Disabled* configurations. All configuration constants are defined in the ScanUser.H file. It consumes t-Actuation to set the configuration.

<u>Parameter</u>	<u>Type/Description</u>
<i>iScan</i>	int Identifies the Scanner. Numbered starting with zero.
<i>iConfiguration</i>	int Configuration to set

Return Value Integer error code.

<u>Value</u>	<u>Meaning</u>
--------------	----------------

SCAN_OKAY Operation successfully completed.

Negative Value Error code

Example `SCANSetConfig(0, TwoWire) // Set scanner to 2-Wire`
 `//configuration`

SCANSetConfigRelay

H/W access Command Polled Command

Description Set one of the configuration relays.

```
#include "SMU4032.H"  
#include "ScanUser.H"
```

```
int SCANSetConfigRelay(int iScan, int iState, int iRelay)
```

Remarks This function opens or closes the selected configuration and tree relays. It is only available while the Scanner is set to the *Universal* configuration. With *iState* set to **CLOSE** the selected relay is closed, while **OPEN** opens it. The Configuration and Tree relays are defined in the ScanUser.H file. They include *AtoA*, *BtoA*, *BtoB*, *CtoA*, *CtoC*, *DtoD*, and *DtoC*. It takes t-Actuation to execute this command. Its channel relays selection counter part is the **SCANSetChannelRelay** function.

<u>Parameter</u>	<u>Type/Description</u>
<i>iScan</i>	int Identifies the Scanner being addressed
<i>iState</i>	int Indicates if relay is to be opened or closed. [OPEN CLOSE]
<i>iRelay</i>	int Identify the configuration/tree relay to act on

Return Value Integer error code.

<u>Value</u>	<u>Meaning</u>
SCAN_OKAY	Operation successfully completed.
Negative Value	Error code

Example `SCANSetConfigRelay(0, CLOSE, CtoA) // Close CtoA relay`

SCANSetScanList

H/W access Command Polled Command

Description Set a single Scan List entry.

```
#include "SMU4032.H"
```

```
int SCANSetScanList(int iScan, int iAddress, int iChannel )
```

Remarks This function writes a single entry to the Scan List table on-board the Scanner. *iAddress* is the location to be written to, and *iChannel* is the channel number to be set. The address can be between 0 and 191, while the channel can be a value between 0 to 32. On power up the contents of the table is not defined. Consider the configuration of the Scanner when writing data to the Scan Table.

<u>Parameter</u>	<u>Type/Description</u>
<i>iScan</i>	int Identifies the Scanner. Scanners are numbered starting with zero.
<i>iAddress</i>	int Identifies the address to write to. It can be a value between 0 and 191.
<i>iChannel</i>	int Channel number maybe a value between 0 to 32 (0 for all open)

Return Value Integer error code.

<u>Value</u>	<u>Meaning</u>
SCAN_OKAY	Operation successfully completed.
Negative Value	Error code

Example

```
double reading; int I;
for(i=0; i<35;i++) SCANSetscanList(0, i-1, i); // Set
locations // 0 to 34 to channels 1 through 35
SCANAutoScan(0,35); // execute a 40 point auto scan
sequence of // Ch1, Ch2, Ch3...Ch35
```

SCANSetStepTime

H/W access Command Polled Command

Description Set auto scan step time

```
#include "SMU4032.H"
```

```
int SCANSetStepTime(int iScan, double dStep)
```

Remarks This function sets the scanner's Step time value. The Step time effects the various auto-scanning operations. It is the dwell time at each channel in a scan sequence. For proper operation the Step time must be greater than the actuation time. The default value is 100mS. The allowed range is from 1ms to 850ms. The resolution is 0.25ms.

<u>Parameter</u>	<u>Type/Description</u>
<i>iScan</i>	int Identifies the Scanner being addressed.
<i>dStep</i>	double Sets the step time. Allowed value is between 1ms to 850ms.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
SCAN_OKAY	Function succeeded.
Negative Value	Error code

Example

```
status = SCANSetStepTime(0, 0.200); // Set step time
// to 200ms
```

SCANSetTriggerOut

H/W access Command Polled Command

Description Set the Scanner Trigger output signal level.

```
#include "SMU4032.H"  
#include "ScanUser.H"
```

```
int SCANSetTriggerOut(int iScan, int iLevel)
```

Remarks This function forces the Scanner's trigger output line to a high or a low level. With *iLevel* set to one (1), the level is set high. Zero (0) forces it low. This setting is independent of the trigger enable line, and may be used as a control line.

<u>Parameter</u>	<u>Type/Description</u>
<i>iScan</i>	int Identifies the Scanner being addressed.
<i>iLevel</i>	int Level indicator. 0 – low, 1 – high.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
SCAN_OKAY	Operation successfully completed.
Negative Value	Error code

Example `status = SCANSetTriggerOut(0,1); //Set trigger line to high level`

SCANSetupStep

H/W access Command Polled Command

Description Set the Scanner for Stepped scanning operation.

```
#include "SMU4032.H"
```

```
int SCANSetupStep(int iScan)
```

Remarks This function prepares the Scanner for a single step scanning operation. See **SCANStep** function for details. It clears the hardware ScanList pointer to point to the first entry of the ScanList table. Step() command increments this pointer.

<u>Parameter</u>	<u>Type/Description</u>
<i>iScan</i>	int Identifies the Scanner being addressed.

Return Value Integer error code.

<u>Value</u>	<u>Meaning</u>
SCAN_OKAY	Operation successfully completed.
Negative Value	Error code

Example `int status = SCANSetupStep(0); // Set for step scan operation`

<u>Parameter</u>	<u>Type/Description</u>
<i>iScan</i>	int Identifies the Scanner being addressed.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
SCAN_OKAY	Operation successfully completed.
Negative Value	Error code

Example

```

SCANStepCmd(0);
DMMSetRange(0,3V) //Set DMM to appropriate range
DMMRead(0, &Val); //Flush a reading or two
While( ! SCANReady(0)); // Wait for Scanner to be ready
DMMRead(0, &reading); // Read input

```

SCANTerminate

H/W access Command Polled Command

Description Terminate Scanner's operation, and removes it from USB configuration.

```
#include "SMU4032.H"
```

```
int SCANTerminate(int iScan)
```

Remarks This function opens all relays of the selected Scanner, then removes it from the USB structure. To use this Scanner again, it is necessary to either, exit the thread, or reinitialize it. It is not necessary to use this function since exiting the thread will automatically terminate the scanner. Following the termination operation, the scanner is not accessible.

<u>Parameter</u>	<u>Type/Description</u>
<i>iScan</i>	int Identifies the Scanner being addressed.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
SCAN_OKAY	Operation successfully completed.
Negative Value	Error code

Example `status = SCANTerminate(0); // Terminate scanner #0`

SCANTestChanIntegrity

H/W access Command Polled Command

Signametrics

Description Test a channel relay for integrity.

```
#include "SMU4032.H"
```

```
int SCANTestChanIntegrity(int iSca, int iChan)
```

Remarks This function tests a single channel relay. The procedure closes a single relay, waits for tActuation, and then checks that both contacts are closed. Next it opens the relay, waits for 1/2 of tActuation, and then checks if the relay is open. A bounce following contact closure and opening is not verified. This test operation is fast, but it is very thorough. Use the **SCANTestChannelRelay** and **SCANTestConfigRelay** operations for a more comprehensive test. The test connector must be in place to perform this test. Following the completion of this test, the scanner is left in the *Disabled* configuration.

<u>Parameter</u>	<u>Type/Description</u>
<i>iScan</i>	int Identifies the Scanner being addressed.
<i>iChan</i>	int Identifies the Channel relay to be tested (1 to 32)

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
SCAN_OKAY	Operation successfully completed.
Negative Value	Error code

Example

```
SCANTestChanIntegrity(0,1); // test Chan1 relay
```

SCANTestChannelRelay

H/W access Command Polled Command

Description Test a channel relay.

```
#include "SMU4032.H"  
#include "ScanUser.H"
```

```
int SCANTestChannelRelay(int iScan, int iChan, double *lpdBounce)
```

Remarks This function tests a single channel relay. Its diagnostics include excessive bounce and open and short failures. If no failure, the Actuation time of the relay, including bounce time, *lpdBounce*, is returned. This value maybe used to fine-tune the Scanner for maximum switching performance by setting the Actuation time using **SCANSetActuationTime**. The Channel parameter can be a value between 1 and 32. The SMU40L test connector is required to execute this procedure. Following the completion of this test, the Scanner is left in the *Disabled* configuration.

<u>Parameter</u>	<u>Type/Description</u>
<i>iScan</i>	int Identifies the Scanner being addressed.

lpdBounce **Double** * Pointer where the bounce time value is to be saved.

iChan **int** Identifies the Channel relay to be tested (1 to 32)

Return Value Integer error code.

Value **Meaning**

SCAN_OKAY Operation successfully completed.

Negative Value Error code

Example `SCANTestChannelRelay(0, 4); // test Chan4`

SCANTestConfigRelay

H/W access Command Polled Command

Description Set the RTD parameters.

```
#include "SMU4032.H"
```

```
#include "ScanUser.H"
```

```
int SCANTestConfigRelay(int iScan, int iConf, double * lpdBounce)
```

Remarks This function tests a single configuration or tree relay. Its diagnostics include excessive bounce and open and short failures. If no failure, the Actuation time of the relay, including bounce time, is returned. This value can be used to fine-tune the Scanner for maximum switching performance by setting the highest Actuation time using **SCANSetActuationTime**. The *iConf* parameter can be set to one of; AtoA, BtoA, BtoB, CtoA, CtoC, DtoC or DtoD. See the *ScanUser.h* file for definitions. The test connector is required to execute this procedure. Following the completion of this test, the Scanner is set to the *Disabled* configuration.

Parameter **Type/Description**

iScan **int** Identifies the Scanner being addressed.

lpdBounce **Double** * Pointer where the bounce time value is to be saved.

iConf **int** Identifies the configuration/tree relay to be tested.

Return Value Integer error code.

Value **Meaning**

SCAN_OKAY Operation successfully completed.

Negative Value Error code

Example `SCANTestConfigRelay(0, BtoA); // test the BtoA relay`

SCANTrigAutoScan

H/W access Command Polled Command

Description Set Scanner for Triggered Auto Scan operation.

```
#include "SMU4032.H"
```

```
int SCANTrigAutoScan(int iScan, int iPoints)
```

Remarks This is the externally triggered version of **SCANAutoScan** operation. Following acceptance of this command, the Scanner enters a wait state, whereby it waits for a trigger edge to start an Auto Scan operation. Trigger edge is whatever was previously selected using the **SCANTriggerOutState**. Being a polled operation, it requires **SCANReady** to return TRUE as indication of completion. Do not issue new command until **SCANReady** returns TRUE. An exception is the **SCANAbort**, which terminates this operation. Keep in mind that the last channel selected is left closed. See **SCANAutoScan** for details.

<u>Parameter</u>	<u>Type/Description</u>
<i>iScan</i>	int Identifies the Scanner being addressed.
<i>iPoints</i>	int Indicates the number of points in the scan.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
SCAN_OKAY	Function succeeded.
Negative Value	Error code

Example `SCANTriggerOutState(0,ENABLED, POSITIVE); //positive edge`
`CANTrigAutoScan(0, 20); // set off 20 point scan.`

SCANTriggerInState

H/W access Command Polled Command

Description Set trigger input state.

```
#include "SMU4032.H"  
#include "ScanUser.H"
```

```
int SCANTriggerInStete(int iScan, int iState, int iEdge)
```

Remarks This function sets the scanner's trigger input state and active edge. Default is DISABLED and POSITIVE. If *iState* is set to ENABLED, the trigger input line is enabled. If *iEdge* is set to POSITIVE, Positive Edge will activate operations. See definitions in ScanUser.H file.

<u>Parameter</u>	<u>Type/Description</u>
<i>iScan</i>	int Identifies the Scanner being addressed.

iState **int** Indicates the enabled state of the trigger input line. May be set to ENABLED or DISABLED.

iEdge **int** Indicates the active trigger edge for the trigger input line. May be set to POSITIVE or to NEGATIVE.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
SCAN_OKAY	Function succeeded.
Negative Value	Error code

Example `SCANTriggerInState(0, ENABLED, POSITIVE) // set Trig-in line`

SCANTriggerOutState

H/W access Command Polled Command

Description Set trigger output state.

```
#include "SMU4032.H"
#include "ScanUser.H"
```

```
int SCANTriggerOutStete(int iScan, int iState, int iEdge)
```

Remarks This function sets the scanner's trigger output line state and polarity. Default is DISABLED and POSITIVE. If *iState* is set to ENABLED, the trigger output is enabled, which reflects in its activity during channel selection and during scanning operations. With *iEdge* set to POSITIVE, the trigger output polarity will be positive, which means that a positive edge on the trigger output line indicates a channel selection ready. See definitions in ScanUser.H file.

<u>Parameter</u>	<u>Type/Description</u>
<i>iScan</i>	int Identifies the Scanner being addressed.
<i>iState</i>	int Indicates the enabled state of the trigger output line. It may be set to ENABLED or DISABLED.
<i>iEdge</i>	int Indicates the active edge for the trigger output line. It may be set to POSITIVE or to NEGATIVE.

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
SCAN_OKAY	Function succeeded.
Negative Value	Error code

Example `SCANTriggerOutState(0, ENABLED, POSITIVE) //set Trig out`

SCANTrigScan

H/W access Command Polled Command

Description Set Scanner for trigger driven step scanning.

```
#include "SMU4032.H"
```

```
int SCANTrigScan(int iScan, int iPoints)
```

Remarks Setup for a hardware-triggered step scanning operation. *iPoints* is the number of points in the scan. The ScanList must be pre-loaded prior to issuing of this command. After receiving this command, the Scanner enters a wait state whereby each selected edge on the trigger input line, selects the next channel from the scan list table. It begins with location 0 of the ScanList and scans a total of *iPoints*. Zero value in scan list opens all channels, which is used for involving multiple scanners in the scanning process. This way a single scan can include points from several scanners. Up to 192 point may be set per scan. This process is completed when *iPoints* have been scanned (*iPoints* trigger edges received). This is a polled operation. It is issued, and then monitored for completion using the **SCANReady**. It requires **SCANReady** to return TRUE as indication of completion. Do not issue new command until **SCANReady** returns TRUE. An exception is the **SCANAbort**, which terminates this operation. See **SCANStep** and **SCANSetupStep** for additional information. Parameters effecting the operation include the set configuration, the Actuation time, and trigger input polarity and enable state.

<u>Parameter</u>	<u>Type/Description</u>
<i>iScan</i>	int Identifies the Scanner being addressed.
<i>iPoints</i>	int The total number of points in the scan

Return Value The return value is one of the following constants.

<u>Value</u>	<u>Meaning</u>
SCAN_OKAY	Operation successfully completed.
Negative Value	Error code

Example `SCANTrigScan(0,150); //150 H/W driver scan operation`

6.0 Accessories

Several accessories are available for the SM4000 relay Scanners, which may be purchased directly from *Signametrics* or one of its distributors or representatives. Please see the 'Accessories' section on our web-site for the current accessories available: <http://www.signametrics.com>

The following D-Subminiature 78 position Female connectors are sourced by a large number of manufactureres, and are all compatible with the SMU4030/32 Scanner.

- *Signametrics* SMU40 - Terminal Block. 32 channel screw terminal block, which plugs into the scanner's 78 pin DB-78 connector. **Note that using the SMU40 limits the voltages to 50V DC and 40V AC.**
- *Signametrics* SMU40T - Isothermal Terminal Block for Thermocouples. Applicable for the SMU4032 model only. 32 channel screw terminal block with reference junction temperature sensor for Thermocouple temperature measurements. Operates in conjunction with the SMU2040, SMU2042, SMU2044, SMU2060, SMU2062 and SMU2064 DMMs. It plugs into the scanner's 78 pin D-78 connector.

Note that using the SMU40T limits the voltages to 50V DC and 40V AC.

- *Signametrics* SMU40L – Loop back test connector. Required for self tests and contact cleaning operations.
- Several manufacturers provide mating connectors for the SMU4030/32 male DB-78 connector. The following are includes some of these Female 78 position mating connector:
 - HD D-Sub 78 Solder cup PN 310498 (<http://www.trynci.com/cat/conn15.htm>)
 - Norcomp Solder pins 180-078-202-031 (from Digikey or www.Norcomp.net)
 - Norcomp Crimp type pins 180-078-272-000 (from Digikey or www.Norcomp.net)
 - Conec Solder cup PN 164-A-17049X
 - Conec Solder pins PN 164-A-17199X and 164-A-17499X
 - Conec 90degree solder pins 164-A-17929X and 164-A-18629X
 - Tyco/AMP Has several types to select from
 - Positronic Has several types to select from
 - Amphenol Has several types to select from

7.0 Warranty and Service

The SMU4000 series Scanners are warranted for a period of one year from the date of purchase. This warrantee does not include relay wear or damage.

If your unit requires repair, contact your *Signametrics* representative. There are no user serviceable parts within the SMU4000 Scanners. Removal of any of the four external shields will invalidate your warranty. For in-warranty repairs, you must obtain a return materials authorization (RMA) from *Signametrics* prior to returning your unit.