SR192 Development System v3.0

User Manual

Document No: 9801110010b
August 30, 2001

Talon Instruments
150 East Arrow Highway
San Dimas, CA 91773

909-599-0690
909-599-6529 Fax

www.taloninst.com
sales@taloninst.com

PRODUCT INFORMATION:

SR192 Development System Software (Referred to as “the Software” or “this Software”)
Copyright © 1998-2001, Serendipity Systems, Inc., All rights reserved.
299 Van Deren
PO Box 10477
Sedona, AZ 86339

1. Limited Liability

IF INSTALLED AND OPERATED AS REQUIRED, THE SOFTWARE SHOULD PERFORM AS DESCRIBED IN THE DOCUMENTATION ENCLOSED HEREWITH. ALL OTHER ASPECTS THE SOFTWARE IS PROVIDED "AS IS" WITHOUT ANY OTHER WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED.

YOU ARE NOT GRANTED ANY IMPLIED WARRANTIES OR MERCHANTABILITY AND/OR FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE SOFTWARE IS WITH YOU EXCEPT AS SET FORTH IN PARAGRAPH 5 BELOW. SHOULD THE SOFTWARE PROVE DEFECTIVE, YOU (AND NOT SSI OR ANY AUTHORIZED SSI DEALER) ASSUME THE ENTIRE RESPONSIBILITY AND COST OF ALL NECESSARY OR INCIDENTAL RESULTS PRODUCED, AS WELL AS ANY DAMAGES OF ANY KIND AND ANY SERVICE, REPAIR OR CORRECTION THAT MAY BE REQUIRED.

SOME STATES DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS AND YOU MAY ALSO HAVE OTHER RIGHTS WHICH VARY FROM STATE TO STATE.

Serendipity Systems, Inc. (SSI hereinafter) does not warrant that any of the functions contained in the Software will meet you requirements or that the operation of the Software will be uninterrupted or error free.

SSI warrants that the distribution media on which the Software is furnished to be free from defects in material or workmanship under its intended use, for a period of ninety (90) days from the date of delivery to you as evidenced by a copy of your receipt.

SSI shall not be responsible for any adverse effects caused by Acts of God or any other cause beyond SSI's reasonable control.

2. Limitation of Remedies

SSI's entire liability and your exclusive remedy shall be limited to the replacement within (30) days, for you (the original acquirer), of any distribution media not meeting SSI's Limited Warranty which is returned to SSI or any authorized dealer with a clearly legible copy of your receipt.

IN NO EVENT WILL SSI BE LIABLE TO YOU FOR ANY DAMAGES, INCLUDING ANY LOST PROFITS, LOST SAVINGS, LOST OPPORTUNITIES, OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE SUCH SOFTWARE, EVEN IF SSI OR AN AUTHORIZED DEALER HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY OTHER PARTY.

SOME STATES DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES SO THE ABOVE LIMITATION OR EXCLUSION MY NOT APPLY TO YOU.

Table of Contents

1. INTRODUCTION 1

1.1 System Architecture 3
1.2 System Operation 4
1.3 SR192 Development Environment 5
1.4 SR192 Hardware 6
1.5 Documentation and Printing 6
1.6 Context-Sensitive Help 6

2. PROJECT BROWSER 7

2.1 Project Validation 8
2.2 Toolbar 8
2.3 Menus 9
2.3.1 File Menu 9
2.3.2 Edit Menu 11
2.3.3 Options Menu 12
2.3.4 Help Menu 13
2.4 Dialogs & Windows 14
2.4.1 New Timing Module Page 14
2.4.2 Debug Log 15

3. SR MODULE CONFIGURATION 17

3.1 Modules 18
3.2 Timing Generators 19
3.3 DAC 20

4. SIGNAL LIST EDITOR 21

4.1 Toolbar 22
4.2 Menus 23
4.2.1 File Menu 23
4.2.2 Edit Menu 24
4.3 Pin Group Control 25
4.3.1 Pin Group Properties 26

5. TIMING SET EDITOR 27

5.1 Toolbar 29
5.2 Menus 30
5.2.1 File Menu 30
5.2.2 Edit Menu 31
5.2.3 Set Menu 32
5.3 Dialogs & Windows 33
5.3.1 Cell Wizard 33
5.3.2 Assert & Return Wizard 34

6. TABLE EDITOR 35

6.1 Error Display 37
6.2 Toolbar 38
6.3 Menus 39
6.3.1 File Menu 39
6.3.2 Import & Export Menu 40
6.3.3 Edit Menu 41
6.3.4 Set Menu 43

6.3.5 Options Menu ... 45
6.3.6 Fill Menu ... 46
6.3.7 Errors Menu ... 47
6.4 Dialogs & Windows ... 48
6.4.1 Import Value Change Dump ... 48
6.4.2 Export Value Change Dump ... 49
6.5 Import & Export File Formats ... 50
6.5.1 Value Change Dump File Format ... 50
6.5.2 Simple Hex File Format ... 52

7. SEQUENCE EDITOR ... 53

7.1 Sequence Operation ... 54
7.2 Sequence List ... 55
7.3 Parameter Pane ... 56
7.4 Menus ... 57
7.4.1 File Menu ... 57
7.4.2 Edit Menu ... 59
7.4.3 Options Menu ... 61
7.5 Dialogs & Windows ... 62
7.5.1 Set Jump Enables ... 62
7.5.2 Remove Name ... 63
7.6 Sequence File Format ... 64

8. TEST MANAGER ... 67

8.1 Operation ... 68
8.2 Sequence Execution ... 69
8.3 Automatic Save/Reload ... 70
8.4 Menus ... 71
8.4.1 File Menu ... 71
8.4.2 SR192 Menu ... 73
8.4.3 Timing Menu ... 74
8.4.4 Options Menu ... 75
8.5 Execution Results ... 76
8.6 Execution Result File Format ... 77

9. PROGRAMMING INTERFACE ... 79

9.1 Execution Result Reporting ... 81
9.2 Driver Updates ... 82
9.3 Error Log File ... 84
9.4 Functions ... 85
9.4.1 ssSrBuildProject ... 86
9.4.2 ssSrExecute ... 87
9.4.3 ssSrExecuteSequence ... 88
9.4.4 ssSrGetConfig ... 89
9.4.5 ssSrGetErrorMessage ... 90
9.4.6 ssSrGetExecResults ... 92
9.4.7 ssSrGetList ... 94
9.4.8 ssSrGetTestResults ... 95
9.4.9 ssSrHalt ... 97
9.4.10 ssSrLearnResponse ... 98
9.4.11 ssSrLoadProject ... 100
9.4.12 ssSrLoadSequence ... 101
9.4.13 ssSrReset ... 102
9.4.14 ssSrSelectSR192 ... 103
9.4.15 ssSrStatus ... 104
9.5 Application Development Environments ... 105

9.5.1 Visual Basic 106
9.5.2 Visual C++ 106
9.5.3 HP-VEE 106
9.5.4 National Instruments LabWindows/CVI 107
9.5.5 National Instruments LabView 108
9.6 EXAMPLES 109
9.6.1 Visual Basic Example 109
9.6.2 Visual C++ Example 111

10. APPENDIX A - VERSION CHANGES 115

11. SALES & SUPPORT 119

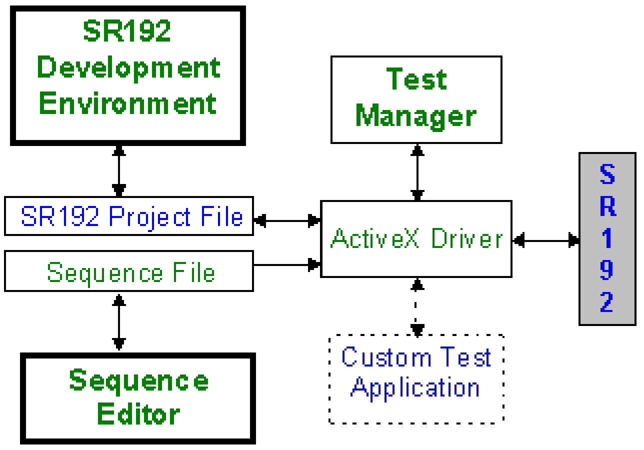
1. Introduction

The Talon SR192 is a modern VXI-based high-performance digital instrument that is used as a digital word generator, bus emulator or as a key component in a digital test system. The SR192 is an extremely adaptable instrument that gets much of its versatility through software control. The SR192 Development System is a collection of software components that provide a comprehensive approach for developing, debugging and maintaining applications for the Talon SR192. Tasks from design to deployment can be accomplished without writing a single line of code.

Coupled with the Talon SR192 hardware, the SR192 Development System provides an application developer with a modern, graphical, open-architecture tool that is applicable to a broad range of operating situations and environments.

1.1 System Architecture

The SR192 Development System includes the SR192 Development Environment, Sequence Editor, ActiveX Driver DLL, Test Manager and SR192 project file. Digital test data is read and written to a project file by the SR192 Development Environment. From there it is loaded and executed on the SR192 hardware by the Test Manager. Alternately, other applications may use the ActiveX Driver DLL to directly control loading and test execution on the SR192.



The SR192 Development Environment is used to visually define and view timing and table information for the Talon SR192. Its advanced user interface design streamlines the process of creating vector patterns and tests. This 32-bit Windows application defines digital test data and timing without cumbersome and arcane programming. In addition, it provides graphical access to the SR192 hardware controls. The SR192 Development Environment writes and reads a project file which stores SR192 application data.

The SR192 Development Environment may be used on a workstation independent of the SR192 hardware. Alternatively, integrated operation with an SR192 is supported when used with the ActiveX Driver DLL and the Test Manager. The SR192 Development Environment is designed to accommodate add-ins for importing test programs from legacy systems, CAD data and simulation data.

The Sequence Editor allows execution sequences to be edited, validated, stored to file and exported as a SCPI file. Sequences are collections of timing set and table pairs (i.e. subsequences) that are executed in a defined order. Their execution order is controlled through conditional jumps and subroutine calls. Each subsequence has a separate count that allows it to be looped up to 32768 times. Building sequence definitions by hand can be tedious and error prone. The Sequence Editor is designed to quickly capture sequence requirements, even while providing a great degree of visibility and flexibility.

The Test Manager is a sample Windows application that uses the ActiveX Driver DLL to handle loading and executing a project file on an SR192. Test results are displayed and several execution options are supported. The Test Manager is written in Visual Basic and its source code is included with the system. The Test Manager may be used as is, customized or simply serve as a program interface example.

The SR192 ActiveX Driver DLL provides high-level software access to the SR192 hardware. It coordinates the loading and execution of a project file on the SR192. It can be used to integrate SR192 control with other instruments or applications. Its programming interface supports development in a wide variety of languages and environments. These include HP-VEE, National Instruments LabWindows CVI, National Instruments LabView, C++, Java and Visual Basic.

The SR192 project file is a repository for SR192 software control settings, timing set and table data. It holds all the information required for a specific operation. This greatly simplifies project development, tracking and maintenance. SR192 project files are identifiable by their file extension (*.SRP).

1.2 System Operation

The SR192 Development System supports several operational scenarios. Its primary role is for the rapid development and deployment of SR192-based test applications. It also functions as a tool for maintaining legacy SR192 test data.

For new development, the SR192 Development Environment is used to create a new project, define table data, define timing set data and specify other SR192 properties. Then the signals used in the table definitions are mapped to the physical SR192 channels. A byproduct of this mapping is the realization of a physical interface design. Once the tables, timing sets and control properties have been defined, the information is stored in an SR192 project file.

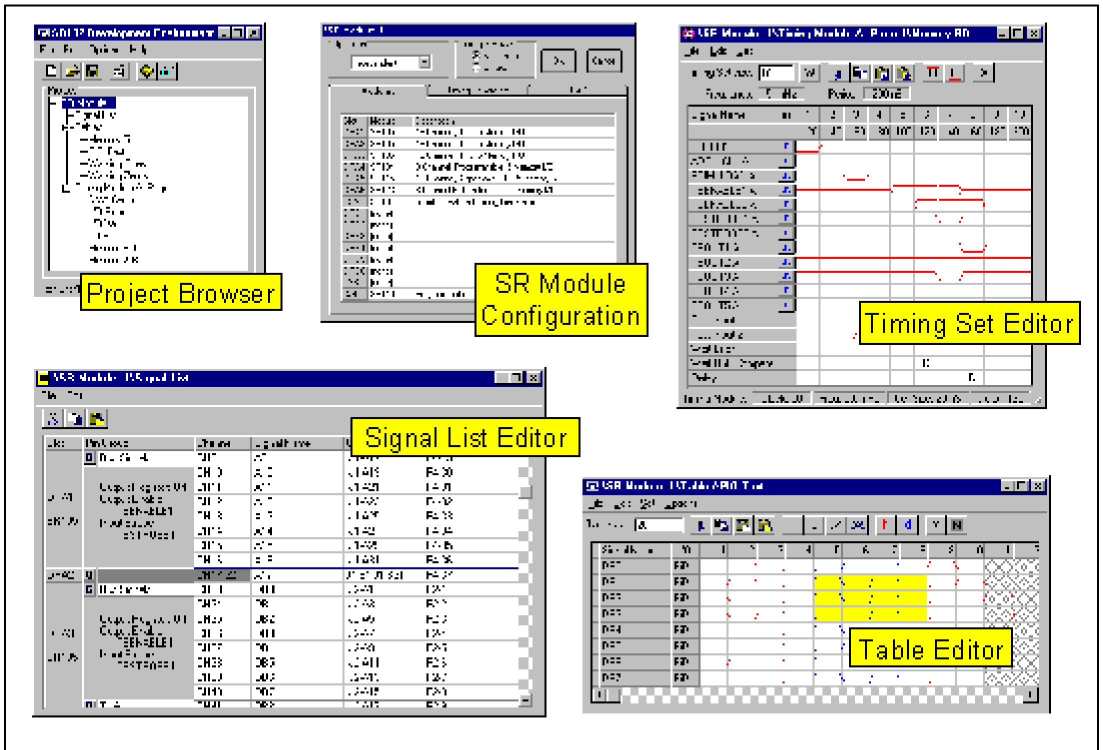
The debug phase begins when the project file is downloaded to the SR192 and executed with the Test Manager. The Test Manager facilitates initial testing and debug by providing capabilities such as running, looping and reporting test results. The integral connection between the SR192 Development Environment and the Test Manager provides a systematic way to edit, download and run a revised project file.

Once debug is complete, the Test Manager can be used by itself for production test. In many circumstances it has sufficient functionality for deployment. If not, the Test Manager source code is provided so that it can be modified to meet specific requirements. Alternatively, the project file can be invoked from a high-level test application, or test executive, through the ActiveX Driver DLL.

Another use for the SR192 Development System is the preservation and maintenance of legacy SR192 test data. The Test Manger can read the contents of an SR192 and store the resulting test information in an SR192 project file. This imported project file can then be opened by the SR192 Development Environment for examination, documentation, archiving or modification.

1.3 SR192 Development Environment

The SR192 Development Environment contains a set of software components that collectively define the operation of an SR192 instrument. Each aspect of an SR192's behavior is supported by one of these components.



SR192 Development Environment

The Project Browser defines the structure of an SR192 project. A hierarchical list shows the relationship between various project elements and allows them to be added, deleted, copied and edited. The project list manages timing sets, data tables, signal lists and multiple SR192 modules.

The SR Module Configuration dialog defines the installed SR192 hardware, timing generator settings and variable voltage levels. This information is typically specified when a project file is first created. Alternatively, the hardware configuration can be "read", by the Test Manager, from an initialized SR192.

The Signal List Editor matches digital signals to physical channels and defines their operating characteristics. It also handles pin group management and cross referencing between SR192 connectors and UUT connector pins. Signals cannot be assigned until pin modules have been specified with the SR Module Configuration dialog.

The Timing Set Editor defines the timing characteristics required for generating digital data. A timing set provides the "pacing" for digital generation. A timing set cycles for each vector of digital data. Within a timing set, stimulus output and response sampling are determined. Additional timing and control signals are also available, as well as a selection of test conditions.

The Table Editor defines the digital data that is output or sampled by an SR192. A data table is matched with a timing set during execution on an SR192. Each digital signal has multiple properties including direction, level and monitoring.

1.4 SR192 Hardware

The hardware component of the system is a Talon Instruments SR192 Bus Emulator/Word Generator. The SR192 is a VXI-based digital stimulus/response instrument housed in a dual slot VXI card. Each SR192 module supports up to 192 digital I/O channels. Each I/O channel is RAM-backed to a depth of 128K and operates at up to 50 MHz. The SR192 has a unique timing set architecture that is conducive for bus emulation.

This hardware versatility, combined with the SR192 Development System tools, provides several operational possibilities. The system can be used as a classic programmable digital word generator, configurable bus emulator or as a high-speed digital test component in a VXI-based test system.

1.5 Documentation and Printing

Sometimes it is necessary to have digital test information included with paper or on-screen documentation. This is easily achieved by using Windows to capture images from the SR192 Development Environment components. This is particularly useful for signal lists, hardware configurations and timing sets. The following describes how to capture images in Windows.

To copy the window or screen contents

> To copy an image of the window that is currently active, press ALT+PRINT SCREEN.
>
> To copy an image of the entire screen, press PRINT SCREEN.

Tip

> To paste the image into a document, click the Edit menu in the document window, and then click Paste.

To directly print a window or dialog contents

Many of the SR192 Development Environment windows and dialogs support printing from a File Menu and/or via shortcut key (**Ctrl-P**). This print option reduces large window images to fit on a single page.

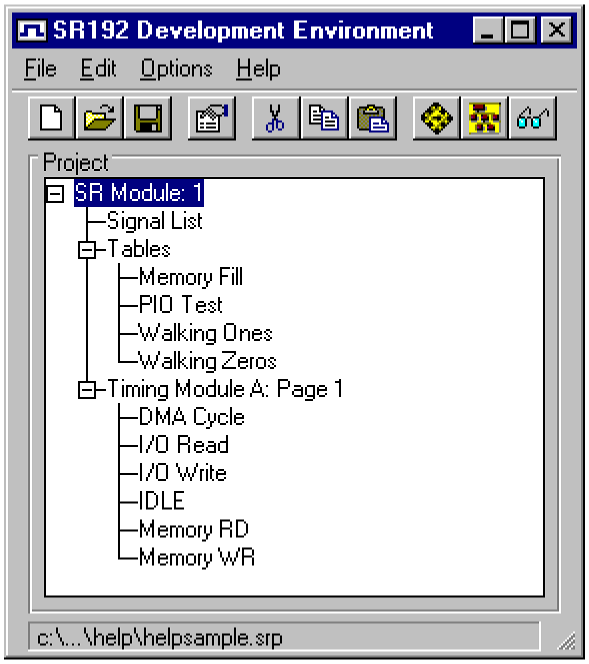
1.6 Context-Sensitive Help

SR192 Development System applications have context-sensitive help for most of their windows and dialog boxes. To access the appropriate help topic, simply press function key **F1**.

> **Note: There are extensive indexing and search capabilities included with the help file (SR192DEV.HLP). These greatly aid locating necessary information.**

2. Project Browser

The Project Browser is the primary interface for the SR192 Development Environment. It is used to load or create a project file, design a project list and select individual items for editing or viewing. The Project Browser is also used to dispatch companion tools and coordinate import and conversion tasks.



The project list is an outline or tree view of the elements contained in the project. A single left mouse click on a branch expands [+] or collapses it [-]. Scroll bars are automatically provided if the list exceeds the size of the pane. Individual elements in the list are selected for editing, copying, deleting or renaming. Elements are added with the Edit Menu or a similar pop-up menu (right mouse click on project list). Placement of new elements is relative to the currently selected item. Note that elements with user defined names are listed in alphabetical order.

Tables, timing sets and complete SR Modules can be copied, cut and pasted via the Edit Menu or toolbar. Elements can be copied and pasted within a single project file or between two project files. Copying to a different project file is most easily accomplished by having two SR192 Development Environments active at the same time.

A project contains SR192 modules, signal lists, data tables and timing sets. Each SR192 module has one signal list and one or more tables and timing sets. Timing sets are further subdivided by timing module (A or B) and page (1 to 4). A selected element is viewed or edited by double clicking the left mouse button, pressing the appropriate toolbar button, or through the Edit Menu.

A status bar at the bottom of the Project Browser displays the name of the current project file. Minimizing the Project Browser causes all associated windows to be hidden.

2.1 Project Validation

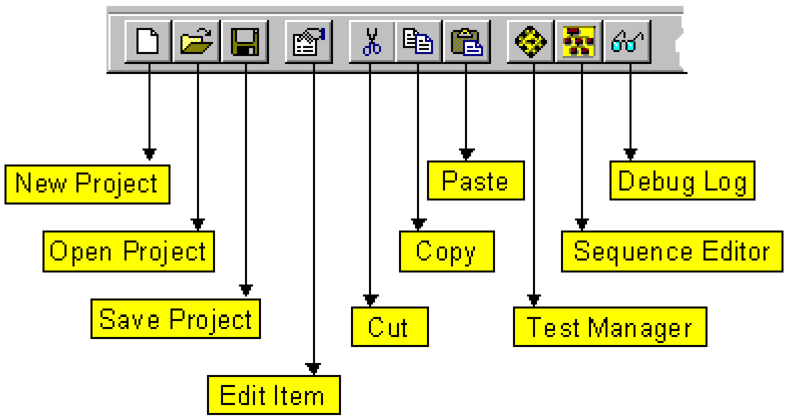
Whenever an SR192 project file is saved, an automatic validation is performed on it. The purpose of the validation is to identify any missing or mismatched elements in the project. This includes missing pin modules, signal names or signal mismatches. Early detection of mismatched signal names can save many hours of hardware debugging.

To avoid warnings about unused signals in a table, preface those signal entries with two forward slashes (e.g. "// Reference Signal" or just "//"). It is often helpful to create unused signal rows for visual spacing or reference.

Validation warnings are displayed on the Debug Log window. Double-clicking a signal warning highlights that signal (if the table is currently displayed).

2.2 Toolbar

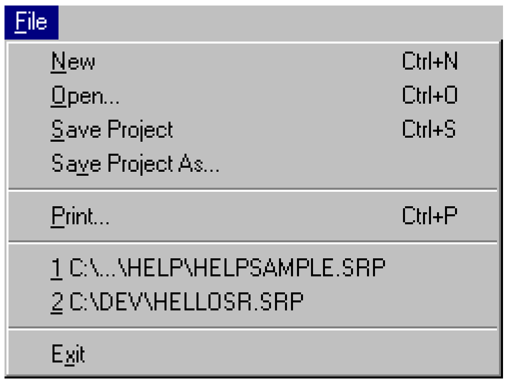
The toolbar on the Project Browser provides quick access to commonly used commands. All of the buttons on the toolbar have corresponding menu entries. Many of them also have shortcut keys that perform the same command. The shortcut keys are shown to the right of their menu entries. Tooltips are displayed when the mouse cursor is held over a toolbar button for approximately two seconds. A tooltip box appears temporarily to identify the button's function.



2.3 Menus

2.3.1 File Menu

The Project Browser File Menu is used to manage the loading and saving of SR192 Project files (*.SRP). With this menu, SR192 project files are created, loaded, saved and renamed. A file history list permits quick reloading of recently accessed project files. The SR192 Development Environment can also be closed from this menu.



Menu Option	**Description**
New	Opens a file browser for naming and placing a new project file. A default configuration is displayed in the project list.
Open…	Opens a file browser for choosing a project file. The chosen file is loaded and displayed in the project list.
Save Project	Updates the project file with the latest editing changes. Project validation is automatically performed when a project is saved.
Save Project As…	Creates a new project file with the latest editing changes. It then becomes the current project file.
Print…	Send an image of the current Project Browser to the printer. Also accomplished by pressing **Ctrl-P**.
File History	This provides a quick selection from the last four project files loaded. The list is updated each time a new file is read. The file names are stored in a state file (SR192DEV.INI).

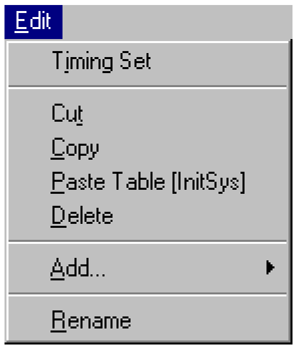
Exit	Close the SR192 Development Environment. If the current project has not been saved, the user is prompted to do so before closing.

When a project file is loaded, or created, a working copy (*.tmp) is made to support interactive editing. When a project is saved, the original file is replaced with the working copy.

Note: If work is lost due to a system crash or power failure, try to recover the working copy of your project file. It is very important to locate and rename it before restarting the SR192 Development Environment. Otherwise, it might be automatically deleted or overwritten.

2.3.2 Edit Menu

The Project Browser Edit Menu is used to manage project contents. It controls editing, adding, deleting and renaming of elements in the project list. It can also be accessed as a pop-up menu (right mouse click on project list). Some menu options are dependent upon which project element is selected. Timing sets, tables and complete SR Modules can be copied and pasted within a single project or between two projects.

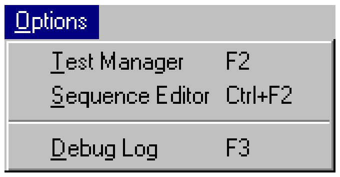


Menu Option	**Description**
<*Selected Item***>**	Activates an editing window for the currently selected item. Also accomplished by double clicking the left mouse button on the item to edit.
Cut	Copy and then delete the selected timing set, table or SR Module.
Copy	Copy the selected timing set, table or SR Module to the paste buffer file (SR192Copy.buf).
Paste <*Item*>	Paste the named item relative to the current selection.
Delete	Removes the selected item from the project list. Some elements cannot be removed.
Add…	Presents a list of elements that can be added to the project list. Placement of new elements is relative to the currently selected item. Tables and timing sets are listed in alphabetical order.
Rename	Allows the current item to be renamed. Some elements (e.g. SR Modules) cannot be renamed. Also accomplished by pressing **Ctrl+R**.

Note: Table and timing set names are truncated to ten characters when they are downloaded to the SR192 hardware. This may affect the behavior of sequences or a programmatic interface.

2.3.3 Options Menu

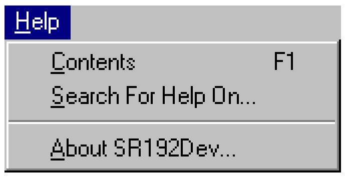
The Project Browser Options Menu is used to access external utilities and logging information. The Test Manager is an external utility for interactive control of an SR192. The Sequence Editor is an external utility for editing execution sequences. They both can be initiated from the Options Menu or executed directly.



Menu Option	Description
Test Manager	Starts the Test Manager utility for interactive control of an SR192. The Test Manager automatically reads the current project file and writes it to the SR192. Function key **F2** also performs this task.
Sequence Editor	Starts the Sequence Editor utility for creating, viewing and modifying execution sequences. Function key **Ctrl+F2** also performs this task.
Debug Log	Allows the user to toggle between the Debug Log window and the Project Browser. Function key **F3** also performs this task.

2.3.4 Help Menu

The Project Browser Help Menu provides access to the SR192 Development Environment help file and version information. The help file is most easily accessed by pressing the **F1** function key.

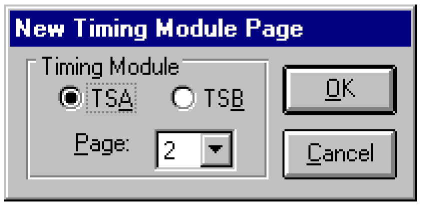


Menu Option	**Description**
Contents	Displays the Contents page of the help file. Function key **F1** also performs this task.
Search for Help On	Displays the Search index of the help file. Use this to locate specific information in the help file.
About SR192Dev…	Displays a dialog containing version information, a serial number and copyright notice.

2.4 Dialogs & Windows

2.4.1 New Timing Module Page

The New Timing Module Page dialog is displayed when adding a timing module page via the Edit Menu. The timing module is selected by a left mouse click next to TSA or TSB. The page value is selected from a drop-down list of 1 to 4. Press the OK button to add the specified page to the project list. Up to 15 timing sets can be defined for each page in a timing module.

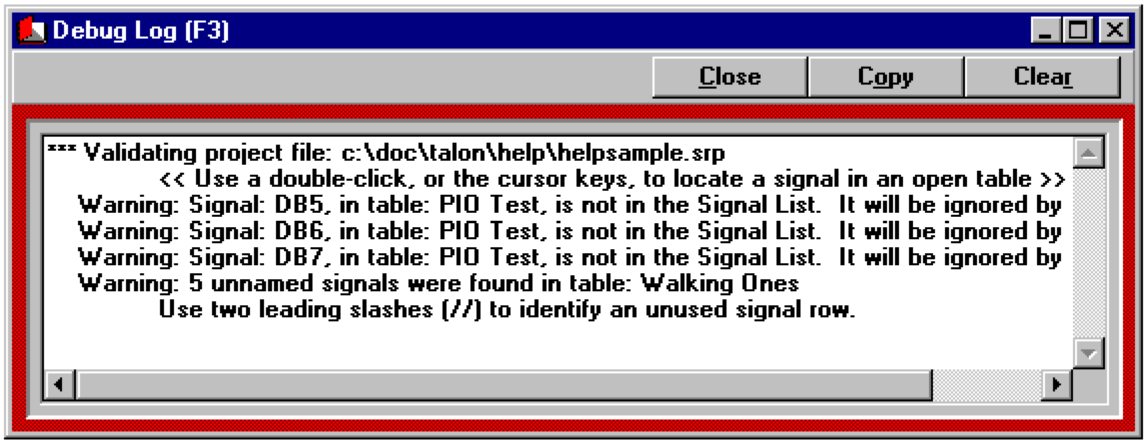


Timing Module: [✓ TSA] TSB

Page: 2

2.4.2 Debug Log

The Debug Log window displays warnings and errors that occur during SR192 Development Environment operations. This includes problems encountered when loading or saving SR192 Project files. The Debug Log window is activated via the Options Menu. Function key **F3** toggles between the Debug Log and the Project Browser window. Scroll bars and key commands (PgUp, PgDn, Home, etc.) are used to move around the window. Push buttons are provided to copy the selected contents to the Windows clipboard or to clear the display.



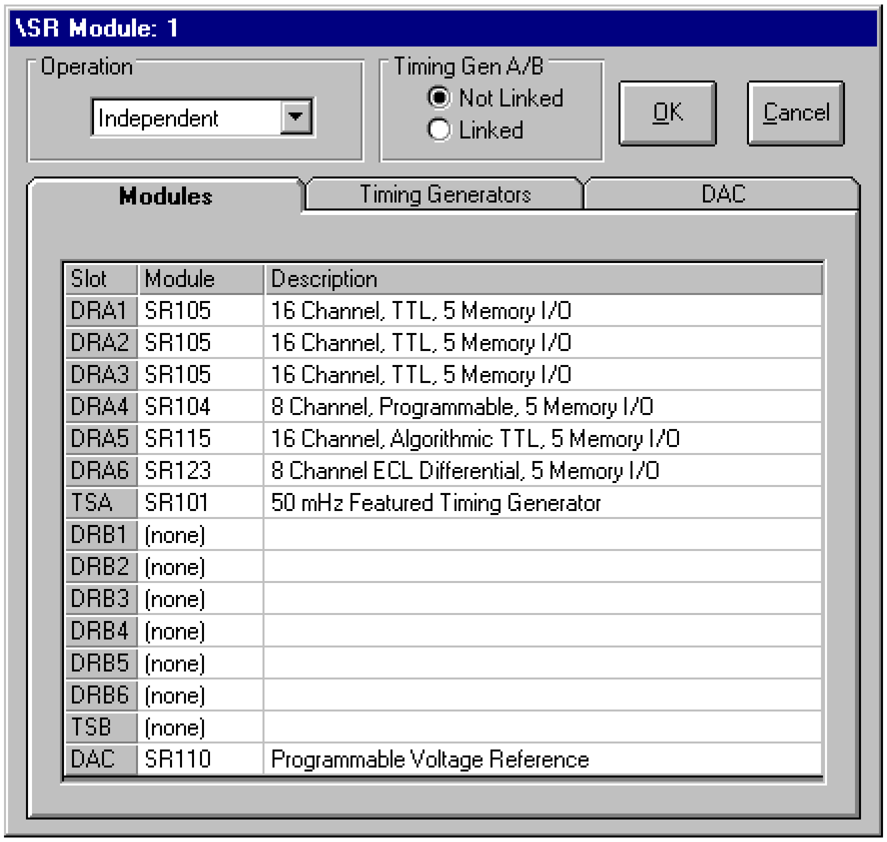
```
*** Validating project file: c:\doc\talon\help\helpsample.srp
          << Use a double-click, or the cursor keys, to locate a signal in an open table >>
   Warning: Signal: DB5, in table: PIO Test, is not in the Signal List.  It will be ignored by
   Warning: Signal: DB6, in table: PIO Test, is not in the Signal List.  It will be ignored by
   Warning: Signal: DB7, in table: PIO Test, is not in the Signal List.  It will be ignored by
   Warning: 5 unnamed signals were found in table: Walking Ones
          Use two leading slashes (//) to identify an unused signal row.
```

All of the information displayed in this window is also written to a log file (SR192Dev.log). Once this file reaches approximately 200K, it is copied to another file (SR192Dev.bak) as a buffer. The buffer file is deleted when the next one is copied over it.

Note: Some error and warning messages respond to a double-click of the left mouse button, or cursor key scrolling. This causes the source of the error, or warning, to be displayed. This is mostly used to highlight signal problems on tables displayed with the Table Editor. These messages are often generated by project validation and VCD importing.

3. SR Module Configuration

The SR Module Configuration dialog is used to define SR192 hardware organization, timing generator settings and voltage levels. General operating parameters are also set with this window. Pressing the OK button saves changes and causes them to be distributed to the other elements of the SR192 Development Environment. This window has to be closed before accessing other parts of the system. Use a shortcut key (**Ctrl-P**) to print this dialog.



Operation: Independent

Timing Gen A/B: [✓ Not Linked] Linked

Slot	Module	Description
DRA1	SR105	16 Channel, TTL, 5 Memory I/O
DRA2	SR105	16 Channel, TTL, 5 Memory I/O
DRA3	SR105	16 Channel, TTL, 5 Memory I/O
DRA4	SR104	8 Channel, Programmable, 5 Memory I/O
DRA5	SR115	16 Channel, Algorithmic TTL, 5 Memory I/O
DRA6	SR123	8 Channel ECL Differential, 5 Memory I/O
TSA	SR101	50 mHz Featured Timing Generator
DRB1	(none)	
DRB2	(none)	
DRB3	(none)	
DRB4	(none)	
DRB5	(none)	
DRB6	(none)	
TSB	(none)	
DAC	SR110	Programmable Voltage Reference

The top of the dialog has a drop-down list for selecting the operating mode of the SR192. This choice configures the SR192 to operate independently of other SR192s, as the master to one or more SR192s, or as a slave to another SR192.

Two radio buttons define whether timing generators A and B will operate linked or unlinked. When unlinked, the two timing generators operate independently of each other. When linked, their operation is synchronized and they execute a shared timing set and data table.

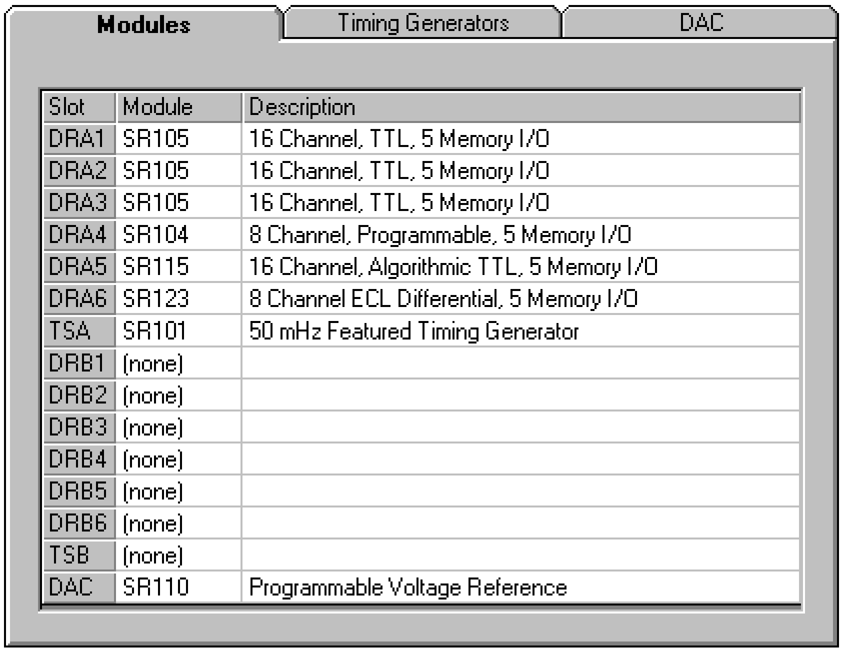
Note: It is best to make the linked/unlinked choice early in development because it affects how tables and timing sets are defined.

Three tabs allow further selection between Modules, Timing Generators and DAC settings. These define installed hardware modules, set general timing behavior and control voltage settings. Installed pin modules need to be defined before signal mapping and control can be specified with the Signal List Editor. A timing generator and clock source needs to be selected in order to enable certain timing attributes of the Timing Set Editor.

3.1 Modules

The Modules tab, of the SR Module Configuration window, is used to define the hardware installed in an SR192. The specified hardware configuration controls several aspects of the SR192 Development Environment. These include signal lists, timing and voltage levels.

An SR192 is populated with up to 12 pin modules (DRA1-6, DRB1-6), two timing generators (TSA, TSB) and a DAC module. Each of these modules contribute to the characteristics of a particular SR192. The wide assortment of pin modules includes 8 and 16 channels; TTL, ECL, differential and variable voltage levels; 3 and 5 memory architectures.



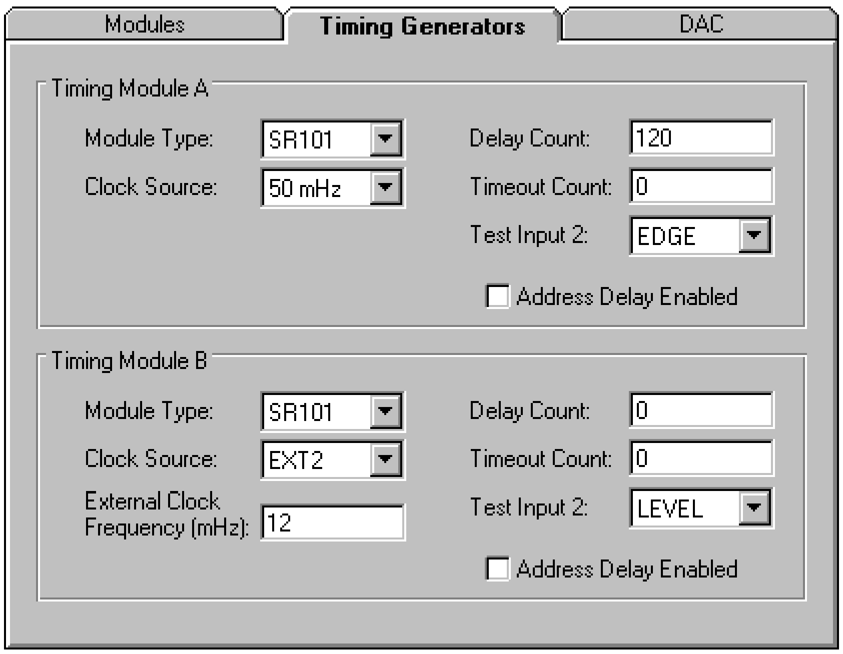
Slot	Module	Description
DRA1	SR105	16 Channel, TTL, 5 Memory I/O
DRA2	SR105	16 Channel, TTL, 5 Memory I/O
DRA3	SR105	16 Channel, TTL, 5 Memory I/O
DRA4	SR104	8 Channel, Programmable, 5 Memory I/O
DRA5	SR115	16 Channel, Algorithmic TTL, 5 Memory I/O
DRA6	SR123	8 Channel ECL Differential, 5 Memory I/O
TSA	SR101	50 mHz Featured Timing Generator
DRB1	(none)	
DRB2	(none)	
DRB3	(none)	
DRB4	(none)	
DRB5	(none)	
DRB6	(none)	
TSB	(none)	
DAC	SR110	Programmable Voltage Reference

A single left mouse click, in the Module column, reveals a drop-down list of the modules appropriate for the particular slot. Once selected, the Description column is updated with information about the module. Timing Generators and DAC modules can also be selected on their corresponding tabs. Use a shortcut key (**Ctrl-P**) to print an image of this tab.

3.2 Timing Generators

The Timing Generators tab, of the SR Module Configuration window, is used to define the timing generator hardware and settings for an SR192. Timing Module A, in slot TSA, controls timing for channels 1-96. Timing Module B, in slot TSB, controls timing for channels 97-192. If the two generators are linked, the settings for Timing Module A are used for all channels in the SR192.

Once the module type is selected from the drop-down list, other parameters can then be set. These settings have a direct effect on the behavior of timing sets associated with the timing module. Use a shortcut key (**Ctrl-P**) to print an image of this tab.



Timing Module A

Module Type: SR101
Clock Source: 50 mHz
Delay Count: 120
Timeout Count: 0
Test Input 2: EDGE
☐ Address Delay Enabled

Timing Module B

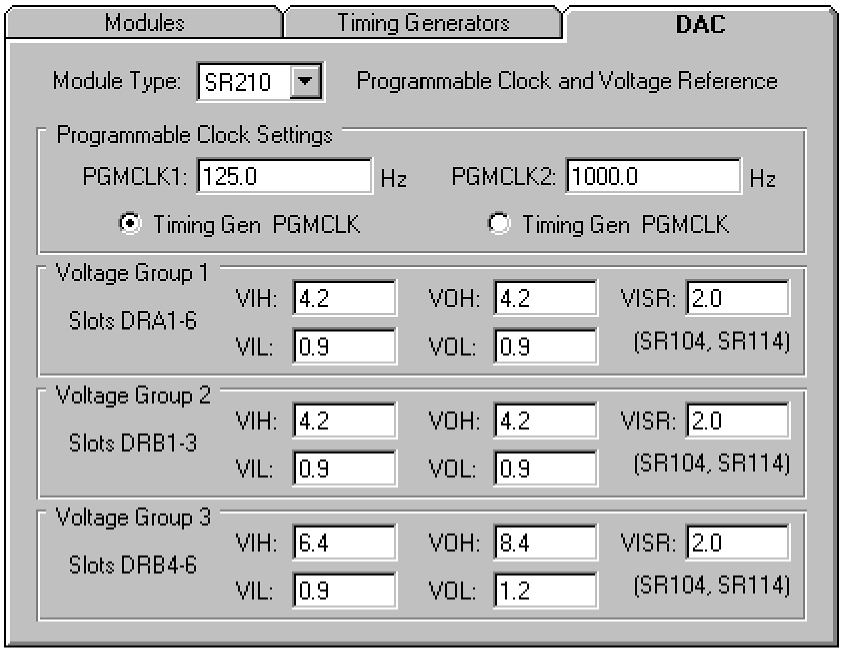
Module Type: SR101
Clock Source: EXT2
External Clock Frequency (mHz): 12
Delay Count: 0
Timeout Count: 0
Test Input 2: LEVEL
☐ Address Delay Enabled

Note: When using an external clock source, enter its frequency (MHz) in the External Clock Frequency field. This allows associated timing sets to be displayed and defined with timing values.

When an SR210 DAC module is present, the Clock Source drop-down list includes a programmable clock option (PGMCLK). The specific programmable clock, and its frequency, is set on the DAC tab.

3.3 DAC

The DAC tab, of the SR Module Configuration window, is used to define programmable voltage and clock references for an SR192. Programmable voltage references are used as drive and sense thresholds by some types of pin modules. A programmable clock can be selected as a clock source by certain types of timing modules. Use a shortcut key (**Ctrl-P**) to print an image of this tab.



Module Type: SR210 — Programmable Clock and Voltage Reference

Programmable Clock Settings

PGMCLK1: 125.0 Hz [✓ Timing Gen PGMCLK]

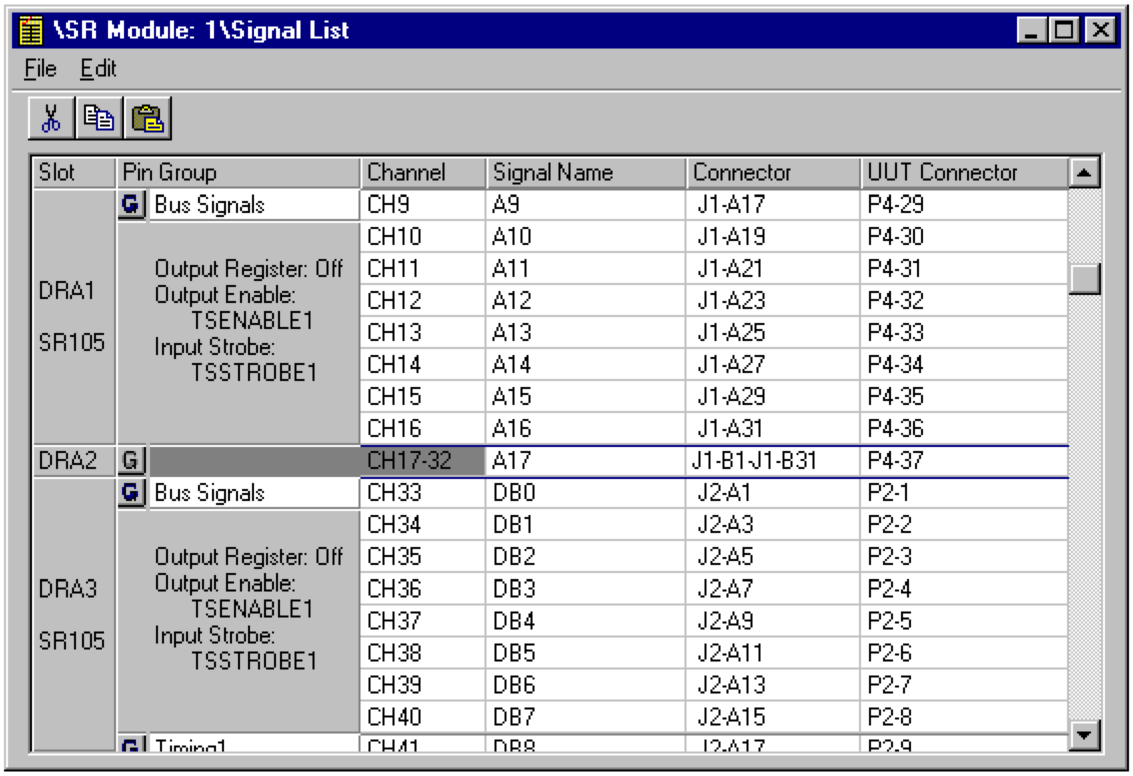
PGMCLK2: 1000.0 Hz Timing Gen PGMCLK

Voltage Group 1 Slots DRA1-6	VIH:	4.2	VOH:	4.2	VISR:	2.0
	VIL:	0.9	VOL:	0.9	(SR104, SR114)	
Voltage Group 2 Slots DRB1-3	VIH:	4.2	VOH:	4.2	VISR:	2.0
	VIL:	0.9	VOL:	0.9	(SR104, SR114)	
Voltage Group 3 Slots DRB4-6	VIH:	6.4	VOH:	8.4	VISR:	2.0
	VIL:	0.9	VOL:	1.2	(SR104, SR114)	

4. Signal List Editor

The Signal List Editor is used to assign signals to physical channels. It also handles the definition of pin groups and the optional entry of UUT connector pins. Signals cannot be assigned until the pin modules have been selected with the SR Module Configuration window.

The behavior and names of digital signals are specified with the Table Editor. It is best to use logical or schematic names for signals rather than physical channel numbers. This isolates the signal data from potential changes in fixtures or instrumentation. The Signal List Editor is then used to match signal names to physical channels. If a data table contains signals that are not present in this list, they are automatically identified during project validation.



Slot	Pin Group	Channel	Signal Name	Connector	UUT Connector
DRA1 SR105	G Bus Signals	CH9	A9	J1-A17	P4-29
	Output Register: Off Output Enable: TSENABLE1 Input Strobe: TSSTROBE1	CH10	A10	J1-A19	P4-30
		CH11	A11	J1-A21	P4-31
		CH12	A12	J1-A23	P4-32
		CH13	A13	J1-A25	P4-33
		CH14	A14	J1-A27	P4-34
		CH15	A15	J1-A29	P4-35
		CH16	A16	J1-A31	P4-36
DRA2	G	CH17-32	A17	J1-B1-J1-B31	P4-37
DRA3 SR105	G Bus Signals	CH33	DB0	J2-A1	P2-1
	Output Register: Off Output Enable: TSENABLE1 Input Strobe: TSSTROBE1	CH34	DB1	J2-A3	P2-2
		CH35	DB2	J2-A5	P2-3
		CH36	DB3	J2-A7	P2-4
		CH37	DB4	J2-A9	P2-5
		CH38	DB5	J2-A11	P2-6
		CH39	DB6	J2-A13	P2-7
		CH40	DB7	J2-A15	P2-8
	G Timing1	CH41	DB8	J2-A17	P2-9

The leftmost column identifies the slot position (e.g. DRA3) and assigned pin module type (e.g. SR105) for a group of channels. Pin modules typically have 8 or 16 channels. If no pin module is assigned to a slot, it is displayed as a single row with a dark gray background.

The next column specifies the pin group associated with a set of 8 channels. Pin groups are a collection of channels (8 to 192) that share behavioral properties (e.g. timing). The specific properties are dependent on the pin module type, but they can include output enables, input strobes and delays. Using named pin groups allows many channels to easily share a common set of operational characteristics.

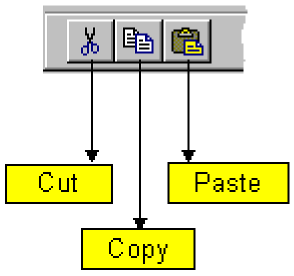
The top row of the Pin Group column has a drop-down list of existing pin groups that are also compatible with the slot's pin module. Alternately, a new pin group name is entered in this location. The remaining rows in the column contain the settings for the specified pin group. Once a pin group is identified, its settings are modified by clicking on the '**G**' button to the left of its name. This activates the Pin Group Control dialog to edit the properties of the pin group.

The Channel column identifies the SR192 channel represented by the row, and the Connector column is its corresponding physical location. The remaining two columns are for user-entered signal names and UUT connectors. The signal names provide the necessary mapping between digital behavior defined in a table and the physical channels of an SR192. The UUT Connector column is provided for additional documentation that could be used when building interface adapters or cable harnesses.

Signal names and UUT connector entries can be cut, copied and pasted from other locations in the SR192 Development Environment, or from many other text sources. A whole column is selected by a left mouse click on its title. Multiple items are selected by holding the left mouse button down and dragging.

4.1 Toolbar

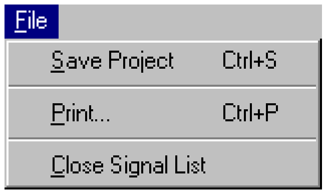
The toolbar, on the Signal List Editor window, provides quick access to commonly used commands. All of the buttons on the toolbar have corresponding menu entries. Many of them also have shortcut keys that perform the same command. The shortcut keys are shown to the right of their menu entries. Tooltips are displayed when the mouse cursor is held over a toolbar button for approximately two seconds. A tooltip box appears temporarily to identify the button's function.



4.2 Menus

4.2.1 File Menu

The File Menu, on the Signal List Editor window, is used to update the project file with the latest changes. This causes all open windows to save their contents to the project file. Periodic saves are recommended as a precaution against editing errors or power failures. The Signal List Editor window is also closed from this menu.

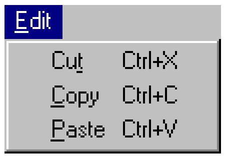


Menu Option	Description
Save Project	Causes all open windows to save their contents to the working project file. Then copies the working project file to the original project file.
Print…	Send an image of the current Signal List Editor to the printer. Also accomplished by pressing **Ctrl-P**.
Close Signal List	Close the Signal List Editor window and update the contents of the working project file.

4.2.2 Edit Menu

The Edit Menu, on the Signal List Editor window, is used to cut, copy and paste signal name and UUT connector information. These operations allow the information to be moved to or from other locations. These locations can be SR192 Development Environment windows or other text sources. This menu can also be accessed as a pop-up menu (right mouse click on the signal grid).

A whole column is selected by a left mouse click on its title. Multiple items are selected by holding the left mouse button down and dragging.

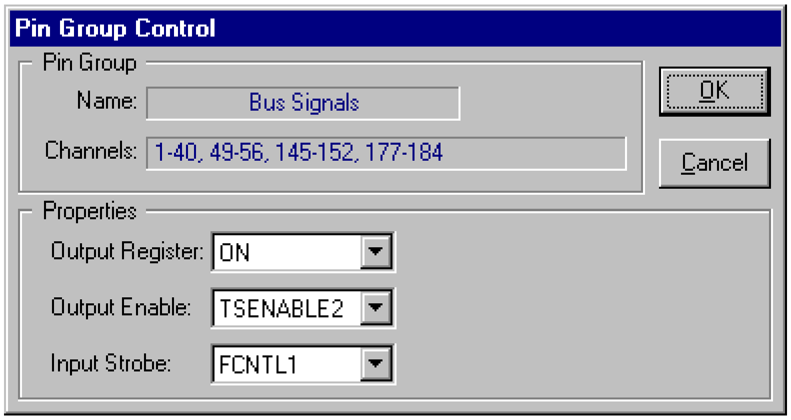


<u>**Menu Option**</u>	<u>**Description**</u>
Cut	Copy the selected rows to the system Clipboard and reset the row contents.
Copy	Copy the contents of the selected rows to the system Clipboard.
Paste	Paste the contents of the system Clipboard into the selected rows.

4.3 Pin Group Control

The Pin Group Control dialog is used to view and edit pin group settings. These are operational parameters that affect signal behavior. SR192 channels are assigned to pin groups with the Signal List Editor. Different types of pin modules have different properties that can be controlled through a pin group. These differences are reflected in the configuration of the Properties section of the Pin Group Control dialog.

Pressing the OK button saves the current settings for the pin group identified in the Name field. The Channels field identifies the SR192 channels that are associated with the referenced pin group. If the channel list exceeds the size of the field, a tooltip box containing the whole list is displayed when the mouse cursor is held over the field. Use a shortcut key (**Ctrl-P**) to print an image of this dialog.

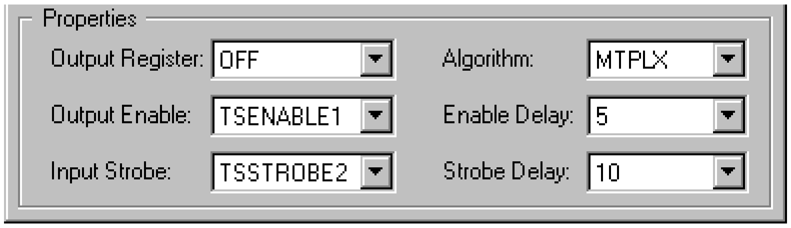


Name: Bus Signals
Channels: 1-40, 49-56, 145-152, 177-184
Output Register: ON
Output Enable: TSENABLE2
Input Strobe: FCNTL1

4.3.1 Pin Group Properties

The Properties section of the Pin Group Control dialog contains the adjustable parameters for a set of SR192 channels. The contents and configuration of the Properties section is dependent upon the type of pin module associated with the channels. Pin modules are assigned to slots/channels with the SR Module Configuration window.

A set of drop-down lists are provided for selecting the operational characteristics of a pin group. Reference the SR192 documentation for further information about pin group properties.



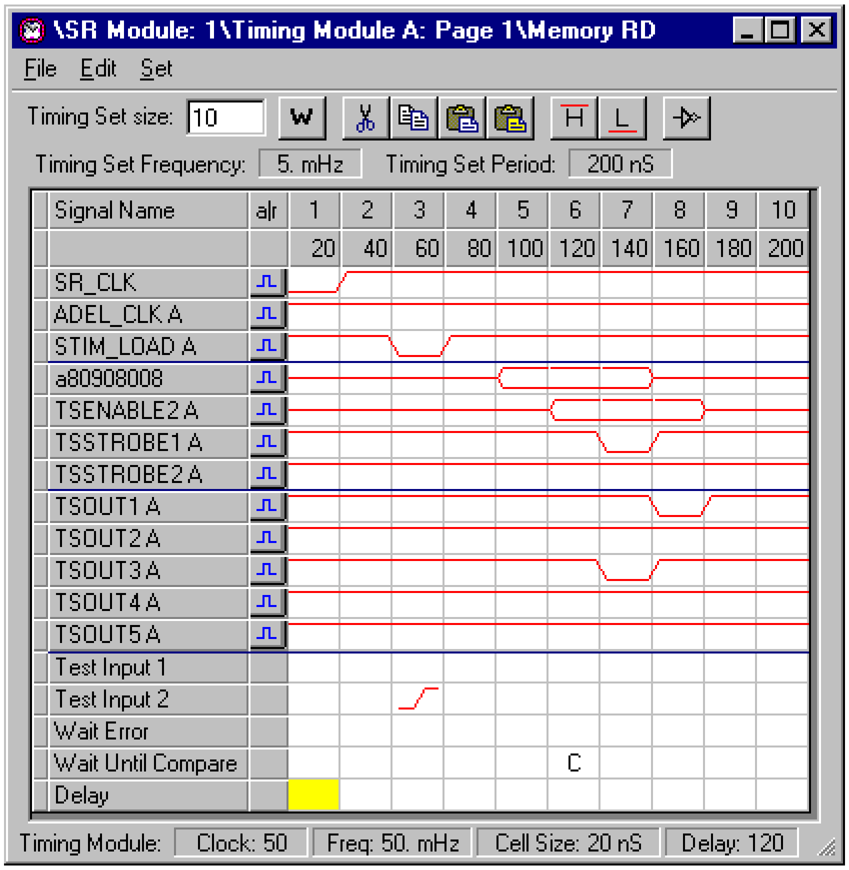
Properties

Output Register:	OFF	Algorithm:	MTPLX
Output Enable:	TSENABLE1	Enable Delay:	5
Input Strobe:	TSSTROBE2	Strobe Delay:	10

5. Timing Set Editor

The Timing Set Editor is used to define the behavior of a timing set. A timing set controls digital data generation when an SR192 is executed. A timing set is cycled for each digital vector that is output. Behavior defined in a timing set controls when stimulus changes occur and when responses are sampled.

A common set of control signals are defined for a timing set. By default they are all initialized to an inactive state (i.e. high) except for SR_CLK. Level changes are easily made by double clicking the left mouse button on a signal cell. Multiple cells are changed by drag selecting them and using the toolbar or Set Menu. A pop-up menu is also available (right mouse click on the signal grid).


\SR Module: 1\Timing Module A: Page 1\Memory RD

Timing Set size: 10

Timing Set Frequency: 5. mHz Timing Set Period: 200 nS

Timing Module: Clock: 50 Freq: 50. mHz Cell Size: 20 nS Delay: 120

Note: If timing generators A and B are linked, the control signals for both are combined into one timing set. The 'B' control signals are displayed with a gray background in order to differentiate them. To ensure proper operation, the 'B' control signals must be set appropriately.

The size (i.e. number of cells) of a timing set is entered in a text box on the toolbar. Pressing the Enter key causes the new size to be displayed. Below the toolbar are fields that show the frequency and period of the timing set. These indicate the rate at which the digital vectors will be generated. Note that the rate also reflects any delay states (**D**) that are specified within the timing set. A Cell Wizard is provided to assist in sizing a timing set based on clock frequency and the desired target frequency or period.

Along the top of the signal grid the columns are labeled with a cell count and time period. Columns are selected by a left click on either of these two top rows. To the left of the signal names is a narrow column that selects a single row with a left click. Multiple rows are selected by holding the left mouse button down and dragging it along this column.

To the right of the signal names is a column of buttons labeled with a blue pulse. Pressing one of these buttons activates the Assert & Return Wizard. This wizard allows a sequence of assert and return times to be numerically entered for an associated signal. This is intended for users that are familiar with defining signals for simulation.

Timing signal names are edited by making a single left click in the appropriate cell. Pressing the Enter key accepts the changes and moves editing to the next lower cell. Signal names can be cut, copied and pasted from other locations in the SR192 Development Environment, or from many other text sources. A whole column is selected by a left mouse click on its title. Multiple items are selected by holding the left mouse button down and dragging.

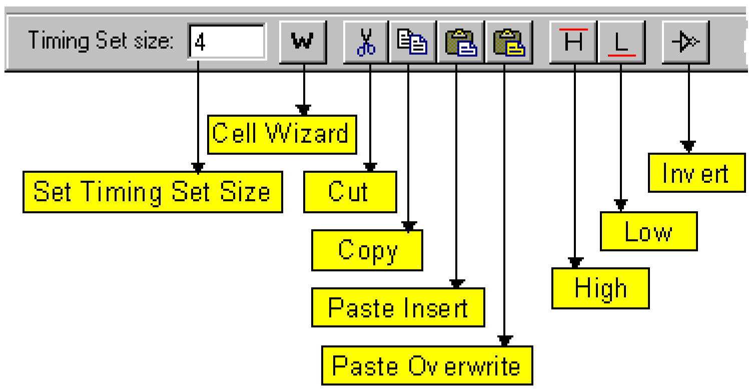
The bottom five rows of the signal grid are test conditions. One of these can be set for each column (i.e. cell) of a timing set. These conditions cause a timing set's execution to pause until an external condition is met (e.g. Test Input 1 at a high level) or a specified amount of time has elapsed (e.g. Delay). The behavior of some of these test conditions is also dependent upon timing generator settings. This includes the delay amount, level or edge triggering for Test Input 2 and an overall timeout count for any of these conditions. Test conditions are easily toggled by double clicking the left mouse button on a cell.

A status bar at the bottom of the window displays the clock source, clock frequency, computed cell size and delay count. These parameters are all defined on the Timing Generators tab of the SR Module Configuration dialog. These fields are left blank if the information is not yet available.

5.1 Toolbar

The toolbar, on the Timing Set Editor window, provides quick access to commonly used commands. It also contains a text box for changing the size (i.e. length) of the timing set.

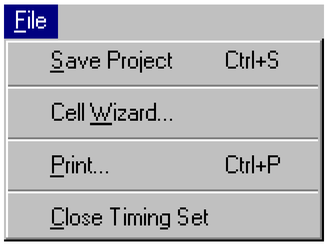
All of the buttons on the toolbar have corresponding menu entries. Many of them also have shortcut keys that perform the same command. The shortcut keys are shown to the right of their menu entries. Tooltips are displayed when the mouse cursor is held over a toolbar button for approximately two seconds. A tooltip box appears temporarily to identify the button's function.



5.2 Menus

5.2.1 File Menu

The File Menu, on the Timing Set Editor window, is used to update the project file with the latest changes. This causes all open windows to save their contents to the project file. Periodic saves are recommended as a precaution against editing errors or power failures. The Timing Set Editor window is also closed from this menu.



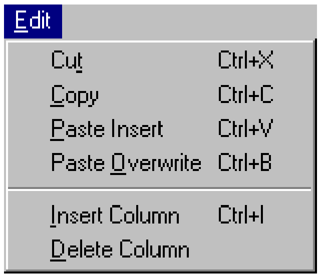
Menu Option	**Description**
Save Project	Causes all open windows to save their contents to the working project file. Then copies the working project file to the original project file.
Cell Wizard	Initiate a Cell Wizard dialog to adjust timing set size based on frequency or period.
Print…	Send an image of the current Timing Set Editor to the printer. Also accomplished by pressing **Ctrl-P**.
Close Timing Set	Close the Timing Set Editor window and update the contents of the working project file.

5.2.2 Edit Menu

The Edit Menu, on the Timing Set Editor window, is used to cut, copy and paste portions of a timing set. This can be used to adjust signal positions or to transfer behavior to another timing set. These operations can also copy all or part of the signal name column to move it to another timing set. Two types of paste operations allow copied signals to replace existing signals (overwrite) or be inserted between existing signals (insert). Only copied timing set information can be pasted in this window. Test conditions are not copied or pasted.

Individual signal cells are selected by a left mouse click. Multiple cells are selected by holding the left mouse button down and dragging. A whole column is selected by a left mouse click on its title. Multiple columns are selected by holding the left mouse button down and dragging it along the title row.

Several toolbar buttons and shortcut keys are provided to more easily initiate these operations. Some operations are also available on the pop-up menu (right mouse click on the signal grid).



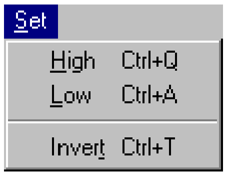
Menu Option	Description
Cut	Copy the selected area to the system Clipboard, remove it and shift left the remaining area to its right.
Copy	Copy the contents of the selected area to the system Clipboard.
Paste Insert	Insert the complete contents of the system Clipboard into the grid, starting at the upper, leftmost selected cell.
Paste Overwrite	Paste the contents of the system Clipboard only into the selected area.
Insert Column	Insert a new, duplicate column to the left of the selected one.
Delete Column	Delete one or more selected columns. A column must be fully selected in order for it to be deleted.

5.2.3 Set Menu

The Set Menu, on the Timing Set Editor window, is used to make selected signals high, low or inverted from their previous state. Signal states are also set by double clicking the left mouse button on an individual signal cell. Many of the signals in a timing set are active during a low state. Test conditions are only changed by double clicking the left mouse button.

Individual signal cells are selected by a left mouse click. Multiple cells are selected by holding the left mouse button down and dragging. A whole column is selected by a left mouse click on its title. Multiple columns are selected by holding the left mouse button down and dragging it along the title row.

Toolbar buttons and shortcut keys are provided to more easily initiate these operations. They are also available on the pop-up menu (right mouse click on the signal grid).

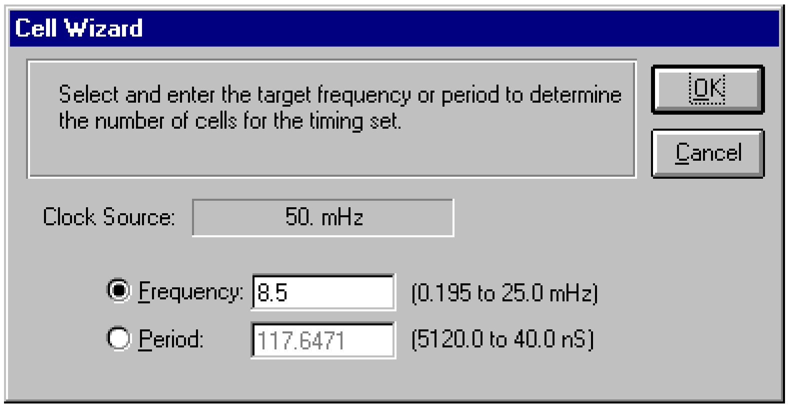


Menu Option	**Description**
High	Set the selected signal cells to a high level.
Low	Set the selected signal cells to a low level.
Invert	Toggle the level of the selected signal cells.

5.3 Dialogs & Windows

5.3.1 Cell Wizard

The Cell Wizard is available from the Timing Set Editor to assist the user in achieving a specific data generation rate. It allows a target frequency, or period, to be entered and it adjusts the size of the timing set to meet that rate. The computation is based on the clock frequency specified for the associated timing generator. If the clock frequency is not yet specified, the Cell Wizard is disabled.



Select and enter the target frequency or period to determine the number of cells for the timing set.

Clock Source: 50. mHz

● Frequency: 8.5 (0.195 to 25.0 mHz)

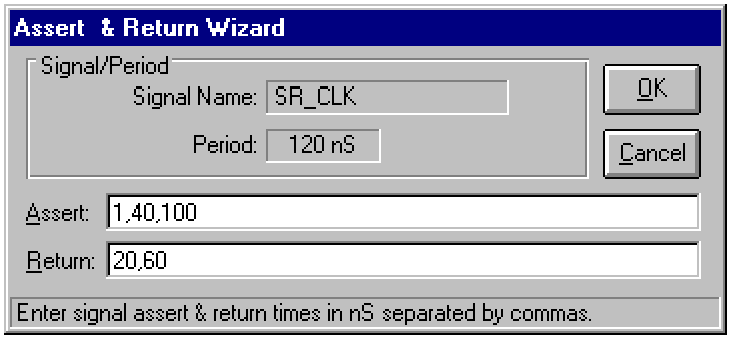
○ Period: 117.6471 (5120.0 to 40.0 nS)

Note that valid timing ranges are displayed to the right of the text entry fields. These ranges are based on the current clock frequency and the range of timing set cells available (2 to 256).

Note: Using the Cell Wizard automatically deletes existing delay states (D) from the timing set.

5.3.2 Assert & Return Wizard

The Assert & Return Wizard is available from the Timing Set Editor to assist the user in specifying timing signal behavior. It uses a simulator-style format for defining when a signal transitions between active and inactive states. Intermediate values are rounded-down to the nearest cell edge.



Assert & Return Wizard

Signal/Period

Signal Name: SR_CLK

Period: 120 nS

Assert: 1,40,100

Return: 20,60

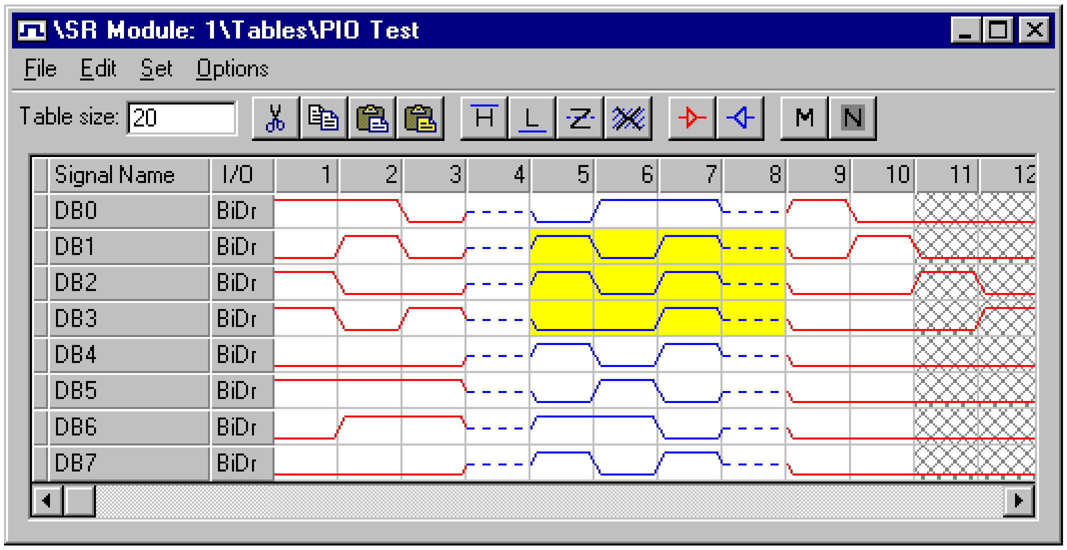
Enter signal assert & return times in nS separated by commas.

Note: Timing signals are active low. Therefore, an assert causes a signal to transition to a low state and a return sets it to a high state.

6. Table Editor

The Table Editor is used to define the behavior of a collection of digital signals. A data table is matched with a timing set when it is executed on an SR192. The timing set is cycled for each digital vector that is output. The signals defined in a table are matched to SR192 channels with the Signal List Editor. Execution errors are automatically highlighted on a table when it is active during Test Manager operations.

By default, a table is set to 32 signals and 100 vectors. Vectors and signals are added or deleted with the Edit Menu. New signals are always initialized to an unknown state. Level changes are easily made by double clicking the left mouse button on a signal cell. Multiple cells are changed by drag selecting them and using the toolbar or Set Menu.



The size (i.e. number of vectors) of a table is entered in a text box on the toolbar. Pressing the Enter key causes the new size to be displayed. Along the top of the signal grid the columns are labeled with a vector count. If the count is too large for the column width, the full count appears when you hold the mouse cursor over the column heading. Columns are selected by a left click on the top row. You can quickly move to a distant vector with the '**Go to**' command on the Options Menu. This is also available on the pop-up menu (right mouse click on signal grid).

Signal conditions are identified by shape and color. Red is a stimulus (drive) signal and blue is a response (sense). The four signal states are high, low, tristate and unknown. The unknown state is used for signals whose behavior is indeterminate during a test. By default all signals are monitored (i.e. tested). Neglected signals are identified by a gray crisscross pattern in the cell.

Signal names are edited by making a single left click in the appropriate cell. Pressing the Enter key accepts the changes and moves editing to the next lower cell. Signal names can be cut, copied and pasted from other locations in the SR192 Development Environment, or from many other text sources. A whole column is selected by a left mouse click on its title. Multiple items are selected by holding the left mouse button down and dragging.

It is best to use logical or schematic names for signals rather than physical channel numbers. This isolates the signal data from potential changes in fixtures or instrumentation. The Signal List

Editor is then used to match signal names to physical channels. When a project file is saved, a validation process is performed that matches table signal names to those on the signal list. If a matching signal is not found, a warning is generated. To avoid warnings on unused signals, preface those entries with two forward slashes (e.g. "// Ref Signal" or just "//"). It is often helpful to create unused signal rows for visual spacing or reference.

Signals are easily repositioned by holding the left mouse button down on the cell to the left of the signal name. The cursor changes to indicate that you are dragging a signal. Release the mouse button to drop the signal in its new position. The signal grid automatically scrolls when you drag the signal near its top or bottom.

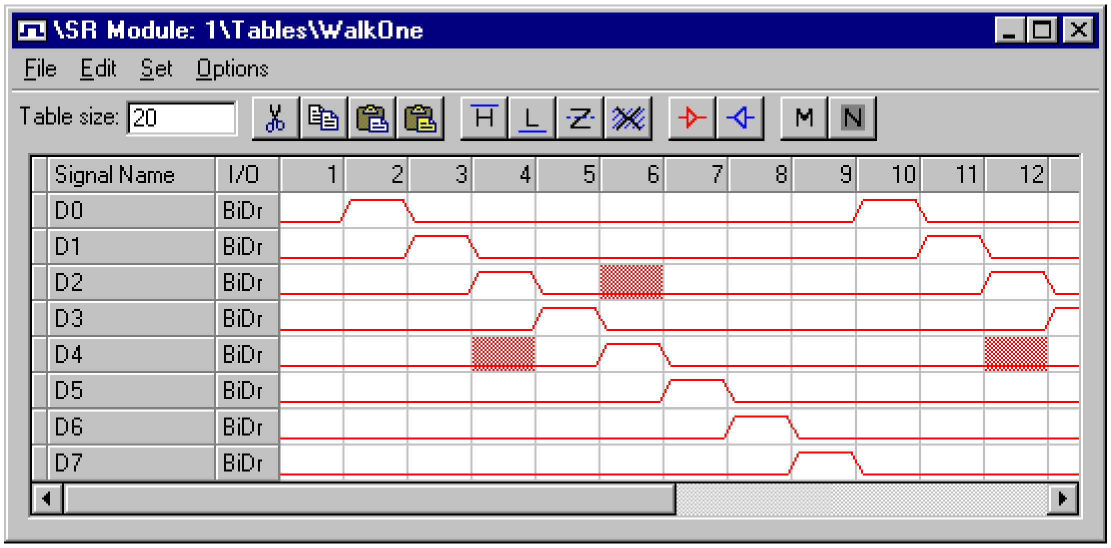
To the right of the signal names is a column of I/O indicators. Double clicking the left mouse button causes these to toggle (e.g. BiDr, Drv, Sens). This defines whether the signal is bidirectional, drive-only or sense-only. Setting the direction of a signal protects it from editing errors and clearly identifies its intended behavior.

A single bidirectional channel is capable of driving and sensing different levels during the same cell. This behavior requires that the drive and sense enables are at different points in the timing set. To define different drive and sense levels for a channel, a **doubled signal** should be created. This is accomplished by creating a drive signal and a sense signal that have the same name. This doubling then allows the drive and sense levels to be defined differently for the same cell.

6.1 Error Display

During test development and debug, the Table Editor automatically displays execution errors detected by the Test Manager. The execution errors are identified by a light red, or pink, background. The Errors submenu has options for finding, clearing and neglecting the highlighted errors.

After the Test Manager runs a test, an execution result file is created. This file contains the project file name and the detected data table failures. Based on this information, the Test Manager looks for an active SR192 Development Environment that has the same project file loaded. If one is found, it is informed that there are applicable execution results to display. Thus, active Table Editor windows are updated following each Test Manager run.



Signal Name	I/O
D0	BiDr
D1	BiDr
D2	BiDr
D3	BiDr
D4	BiDr
D5	BiDr
D6	BiDr
D7	BiDr

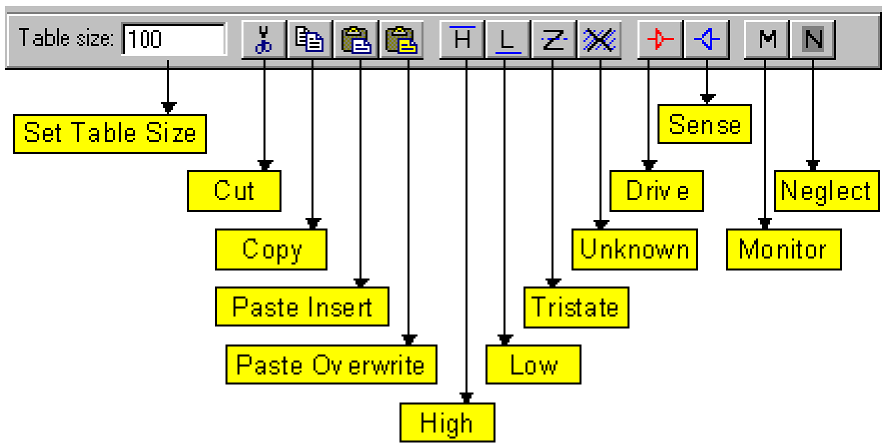
Note: this only displays the number of vector failures defined by the Test Manager's Execution Results dialog. It does not necessarily indicate the full scope of erring vectors.

If the background color used to highlight errors is difficult to discern, it can be modified by editing the application's initialization file (SR192Dev.ini). Change the value for the "ErrorColor" entry to the desired RGB color code. A couple of possible standard colors include cyan (&H808000), light gray (&HC0C0C0), dark gray (&H808080) and light cyan (&HFFFF00). Delete the "ErrorColor" entry to return to the default background color. Be sure to edit the initialization file while the application is closed.

6.2 Toolbar

The toolbar, on the Table Editor window, provides quick access to commonly used commands. It also contains a text box for changing the size (i.e. length) of the table.

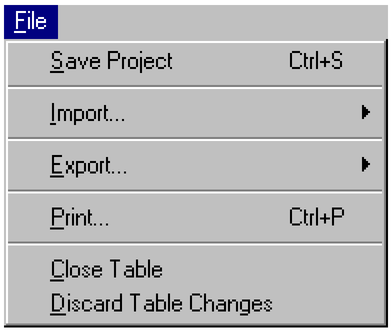
All of the buttons on the toolbar have corresponding menu entries. Many of them also have shortcut keys that perform the same command. The shortcut keys are shown to the right of their menu entries. Tooltips are displayed when the mouse cursor is held over a toolbar button for approximately two seconds. A tooltip box appears temporarily to identify the button's function.



6.3 Menus

6.3.1 File Menu

The File Menu, on the Table Editor window, is used to update the project file with the latest changes. This causes all open windows to save their contents to the project file. Periodic saves are recommended as a precaution against editing errors or power failures. The import and export options permit data to be moved in and out of the table. This allows data to be externally edited or automatically created. The Table Editor window is also closed from this menu. Use the Discard option to close a table without saving it.

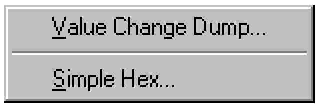


Menu Option	Description
Save Project	Causes all open windows to save their contents to the working project file. Then copies the working project file to the original project file.
Import…	Import to the table, or selected cells, from the specified file type.
Export…	Export the table contents, or selected cells, to the specified file type.
Print…	Send an image of the current Table Editor to the printer.
Close Table	Close the Table Editor window and update the contents of the working project file.
Discard Table Changes	Close the Table Editor window and discard any changes that have been made.

Note: A progress bar is displayed while a large table is being saved. A Cancel button allows the save to be aborted and the Table Editor is subsequently reactivated. Use the Discard option whenever possible with large tables to avoid unnecessary updates to the working project file.

6.3.2 Import & Export Menu

The Import and Export Menus, on the Table Editor window, are used to read and write table data to various file formats. This allows table data to be created or consumed by third party or custom applications. The two formats supported are Value Change Dump (VCD) and a Simple Hex format. VCD is a format that is often used by simulators and waveform viewers. The Simple Hex format is useful because it is easily generated automatically or manually.



Menu Option	Description
Value Change Dump…	Import or export the entire table from/to a VCD formatted file.
Simple Hex…	Import/Export the table, or selected cells, from/to a Simple Hex formatted file.

Note: Large selection areas for Simple Hex import or export are easily achieved. First select one corner of the desired area with a left mouse click; then select the opposing corner with a Shift-left mouse click.

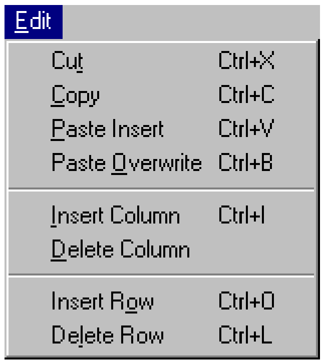
6.3.3 Edit Menu

The Edit Menu, on the Table Editor window, is used to cut, copy and paste portions of a data table. This can be used to adjust signal positions, duplicate behavior or to transfer data to another table. These operations can also copy all or part of the signal name column to move it to another table or to the Signal List Editor.

Two types of paste operations allow copied signals to replace existing signals (overwrite) or be inserted between existing signals (insert). Only copied table information can be pasted in the signal grid. Whole columns, or rows, must be selected before an insert or delete is allowed.

Individual signal cells are selected by a left mouse click. Multiple cells are selected by holding the left mouse button down and dragging. Large selections are quickly achieved by selecting one corner with a left mouse click and the opposing corner with a Shift-left mouse click. A whole column is selected by a left mouse click on its title. Multiple columns are selected by holding the left mouse button down and dragging it along the title row. A single row is selected by a single mouse click in the I/O column.

Several toolbar buttons and shortcut keys are provided to more easily initiate these operations.

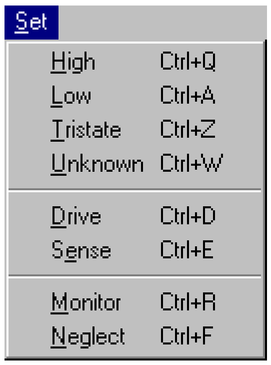


Menu Option	Description
Cut	Copy the selected area to the system Clipboard, remove it and shift left the remaining area to its right.
Copy	Copy the contents of the selected area to the system Clipboard.
Paste Insert	Insert the complete contents of the system Clipboard into the grid, starting at the upper, leftmost selected cell.
Paste Overwrite	Paste the contents of the system Clipboard only into the selected area.
Insert Column	Insert a new, duplicate column to the left of the selected one.
Delete Column	Delete one or more selected columns. A column must be fully selected in order for it to be deleted.
Insert Row	Insert an empty row above the selected one.
Delete Row	Delete the selected row. The signal below is moved up one row and selected. This allows rapid deletion of multiple rows when used with **Ctrl+L**.

6.3.4 Set Menu

The Set Menu, on the Table Editor window, is used to control level, direction and monitoring of digital signals. The set commands operate on the selected signal cells. Signal levels are also set by double clicking the left mouse button on an individual signal cell.

Individual signal cells are selected by a left mouse click. Multiple cells are selected by holding the left mouse button down and dragging. Large selections are quickly achieved by selecting one corner with a left mouse click and the opposing corner with a Shift-left mouse click. A whole column is selected by a left mouse click on its title. Multiple columns are selected by holding the left mouse button down and dragging it along the title row. A single row is selected by a single mouse click in the I/O column. Toolbar buttons and shortcut keys are provided to more easily initiate these operations.



Menu Option	**Description**
High	Set the selected signal cells to a high level.
Low	Set the selected signal cells to a low level.
Tristate	Set the selected signal cells to a mid level (tristate).
Unknown	Set the selected signal cells to an unknown (don't care) state. This becomes tristate/masked in the SR192 hardware.
Drive	Set the selected signal cells to drive (i.e. into the UUT).
Sense	Set the selected signal cells to sense (i.e. read from the UUT).
Monitor	Set the selected signal cells to be monitored (i.e. tested).
Neglect	Set the selected signal cells to be neglected (i.e. not tested).

Note: A progress bar is displayed during a large set operation. A Cancel button allows the operation to be aborted and the Table Editor is subsequently positioned to show the last row modified.

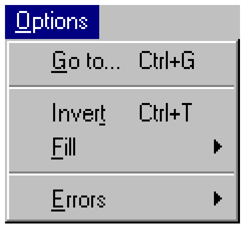
The Unknown state is used for signals whose behavior is indeterminate during a test. Driving channels are automatically set to tristate and sense channels are neglected (i.e. masked). This visually different state allows a user to uniquely identify indeterminate or untestable behavior.

The direction of a signal is changeable if it is designated as bidirectional (BiDr in the I/O column). Otherwise, the direction commands are ignored. Set the I/O column for a signal if you need it to always drive or sense. This protects it from editing errors and clearly identifies its intended behavior. Driving signals are shown in red, sense signals are shown in blue.

By default, all signals are monitored (i.e. tested). Use the Neglect command to disable testing for all or part of a signal's operation. Neglected signals are identified by a gray crisscross pattern in the cell.

6.3.5 Options Menu

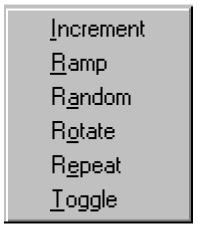
The Options Menu, on the Table Editor window, is used to jump to a different location in the table or change the state of selected signal cells. These commands are also available from a pop-up menu (right mouse click on the signal grid).



Menu Option	Description
Go to…	Prompts for a vector number and shifts the signal grid to display it. Also accomplished by pressing **Ctrl+G**.
Invert	Inverts the state (level) of the selected signal cells. Also accomplished by pressing **Ctrl+T**.
Fill	Activates a submenu with options for setting patterns in a selected block of signal cells.
Errors	Activates a submenu for locating, clearing or neglecting execution errors.

6.3.6 Fill Menu

The Fill Menu, on the Table Editor window, is used to set selected signal cells to a specified pattern. These commands are also available from a pop-up menu (right mouse click on the signal grid).



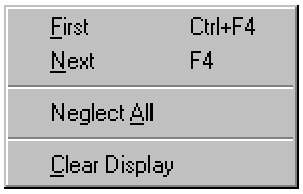
Menu Option	Description
Increment	Set the level of the selected signal cells to an incrementing pattern (topmost signal is least significant). Prompt for increment step.
Ramp	Set the level of the selected signal cells to a ramp pattern.
Random	Set the level of the selected signal cells to a random pattern. Prompt for randomizing seed value.
Rotate	Set the level of the selected signal cells to a rotating pattern based on the leftmost selected column. Prompt for a rotational increment.
Repeat	Set the level of the selected signal cells to match the leftmost selected column.
Toggle	Set the level of the selected signal cells to a toggling (on/off) pattern.

Note: A progress bar is displayed during a large fill operation. A Cancel button allows the operation to be aborted and the Table Editor is subsequently positioned to show the last column modified.

6.3.7 Errors Menu

The Errors Menu, on the Table Editor window, is used to locate, neglect or clear displayed execution errors. The error display correlates execution results with a data table's contents in order to aid test development and debugging. Execution results and display activation are provided by the Test Manager.

When automatically locating an error cell, the cell to the right of the error is selected (i.e. yellow highlight). This provides an additional visual indication of which error has most recently been located. The error navigation commands are also available via shortcut keys (**Ctrl-F4** and **F4**).



Menu Option	Description
First	Shifts the signal grid to display the first execution error. Also accomplished by pressing **Ctrl+F4**.
Next	Shifts the signal grid to display the next sequential execution error. Also accomplished by pressing **F4.**
Neglect All	Sets all error cells to be neglected (i.e. not tested).
Clear Display	Removes error highlighting from table.

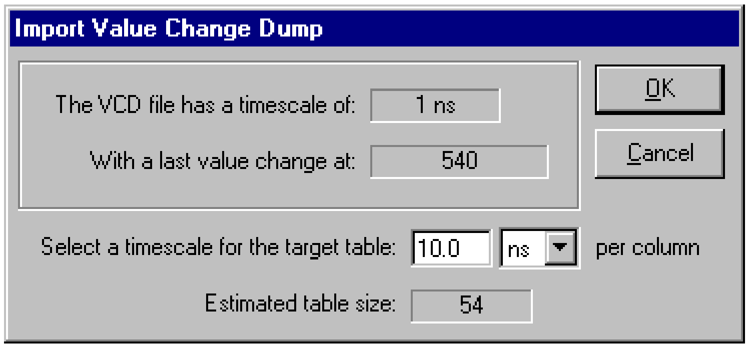
Note: Errors are only displayed for the number of vector failures defined by the Test Manager's Execution Results dialog. Displayed errors do not necessarily indicate the full extent of erring vectors.

6.4 Dialogs & Windows

6.4.1 Import Value Change Dump

When importing a Value Change Dump file, via the Import/Export Menu, this dialog is displayed after an input file is selected. The contents of the VCD file are scanned to determine and display its timescale and length. Based on the scanned information, a suggested scaling value is displayed along with the projected table size.

The default scaling parameter is based on the minimum time between successive value changes in the file. Typically this results in a reduced table size without loss of signal states. If the scale factor causes value changes to be written into the same table location, a warning is written to the Debug Log and the user is notified at the conclusion of the import.


Import Value Change Dump

The VCD file has a timescale of: 1 ns

With a last value change at: 540

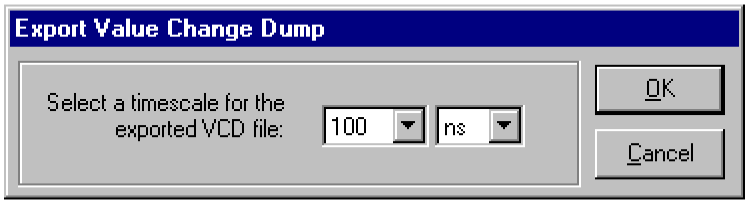
Select a timescale for the target table: 10.0 ns per column

Estimated table size: 54

Note: In the Debug Log window, a double-click of the left mouse button on a VCD warning causes the Table Editor to display the corresponding signal and location. Moving up or down with cursor keys also accomplishes this.

6.4.2 Export Value Change Dump

When exporting a Value Change Dump file, via the Import/Export Menu, this dialog is displayed after an output file is selected. Its purpose is to select a timescale for the VCD file being created.


Export Value Change Dump

Select a timescale for the exported VCD file: 100 ns

Note: Only named signals are exported to a VCD file. Unused signal entries (i.e. "//") are ignored.

6.5 Import & Export File Formats

6.5.1 Value Change Dump File Format

A value change dump (VCD) file contains information about value changes on variables (i.e. signals). It is one of the formats supported for importing and exporting digital data from the Table Editor window. VCD files are often generated, or used, by simulation and waveform viewing applications. Many low-cost or free tools are available for defining or viewing VCD files.

The VCD file format is fully defined in Section 15 of IEEE Standard 1364-1995. It essentially consists of a definitions section and a value change section. The definitions section includes time/date information, VCD writer version, time scale and variable declarations. The value change section lists time increments (e.g. #500) and value changes corresponding to that time.

The VCD format allows a signal to change state within a single time increment (e.g. a momentary pulse or signal spike). This generates an import warning that identifies two states for a signal in the same vector (double-click warning on the Debug Log window to highlight corresponding vector in table).

> **Note: The occurrence of $dump commands can affect the import results. For example, a $dumpoff at the end of a VCD file turns all signals to unknown (X) for the last column of the table. It is always important to examine the contents of a VCD file when questions arise about the import result.**

The following example illustrates the format of a VCD file.

```
$date
      June 26, 2001 10:05:41
$end
$version
      VERILOG-SIMULATOR 1.0a
$end
$timescale
      10 ns
$end
$scope module top $end
$scope module m1 $end
$var trireg 1 *@ net1 $end
$var trireg 1 *# net2 $end
$var trireg 1 *$ net3 $end
$upscope $end
$scope task t1 $end
$var reg 32 (k accumulator[31:0] $end
$var integer 32 {2 index $end
$upscope $end
$upscope $end
$enddefinitions $end
$comment
      Note: $dumpvars was executed at time '#500'.
            All initial values are dumped at this time.
$end

#500
```

```
$dumpvars
x*@
x*#
x*$
bx (k
bx {2
$end
#505
0*@
1*#
1*$
b10zx1110x11100 (k
b1111000101z01x {2
#510
0*$
#520
1*$
#530
0*$
bz (k
#535
$dumpall 0*@ 1*# 0*$
bz (k
b1111000101z01x {2
$end
#540
1*$
#1000
$dumpoff
x*@
x*#
x*$
bx (k
bx {2
$end
#2000
$dumpon
z*@
1*#
0*$
b0 (k
bx {2
$end
#2010
1*$
```

6.5.2 Simple Hex File Format

The Simple Hex file format is used to import and export hexadecimal digital data from the Table Editor window. It is an uncomplicated ASCII format that can be viewed and modified with a variety of utilities and editors. Its straightforward syntax is conducive to automatic generation by CAD, simulation or translation applications.

The first line of a Simple Hex file contains a header that identifies its format. The header is enclosed in square brackets: **[SR192 Simple Hex]**

Each hexadecimal line in the file represents one column of a table. The rightmost bits correspond to the topmost signals. For increased readability, white space and blank lines may be used to separate the hexadecimal data. Comments are preceded by double slashes (//) and they can follow the data or be on a separate line.

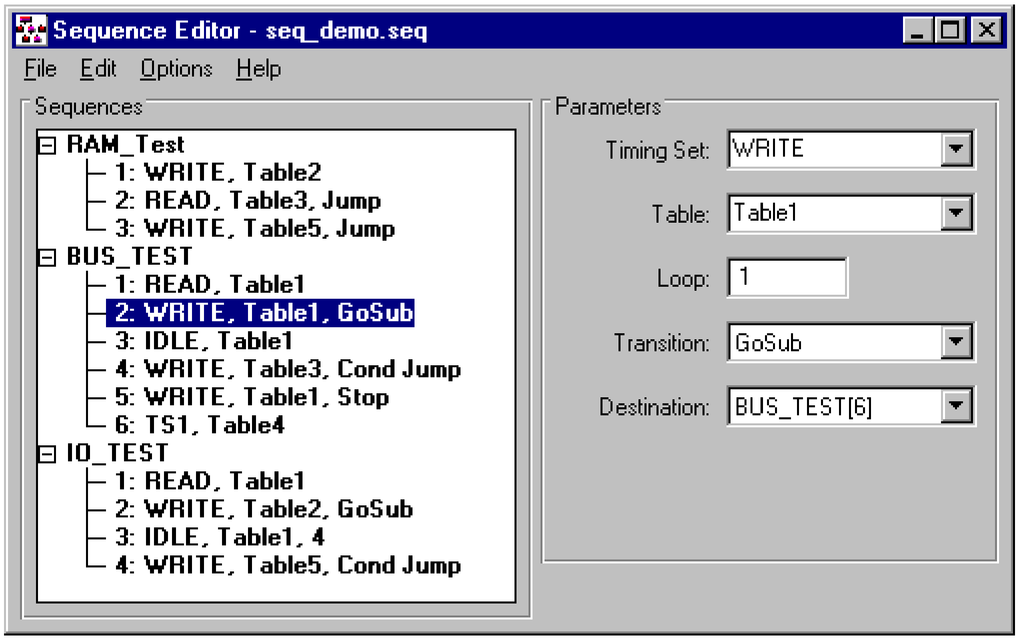
The following is a portion of an exported Simple Hex file with the addition of comments and blank lines:

```
[SR192 Simple Hex]
AC 41 65 96
31 15 AD B7
56 40 59 1B
22 20 9A 7C
ED 8B ED F3
C7 E5 39 D3
0C B4 8E 7B
4B 7D 48 17
ED DF 06 61
B5 0A 6B BD

// Blank lines and comments are allowed
E5 D3 0B 9B
3C 20 1F FF
B2 2F E6 3A
D9 51 33 C5
99 7C 95 87   // comments are allowed here also
F1 96 BD B3
8B 48 AC 69
```

7. Sequence Editor

The Sequence Editor is one of the software components that comprise the SR192 Development System. The Sequence Editor provides an easy-to-use interface for creating and editing execution sequences for the Talon SR192. Sequences are loaded and executed by the Test Manager and programming interface.

Sequences are collections of timing set and table pairs (i.e. subsequences) that are executed in a defined order. Their execution order is controlled through conditional jumps and subroutine calls. Each subsequence has a separate count that allows it to be looped up to 32768 times. Building sequence definitions by hand can be tedious and error prone. The Sequence Editor is designed to quickly capture sequence requirements, even while providing a great degree of visibility and flexibility.



```
RAM_Test
  1: WRITE, Table2
  2: READ, Table3, Jump
  3: WRITE, Table5, Jump
BUS_TEST
  1: READ, Table1
  2: WRITE, Table1, GoSub
  3: IDLE, Table1
  4: WRITE, Table3, Cond Jump
  5: WRITE, Table1, Stop
  6: TS1, Table4
IO_TEST
  1: READ, Table1
  2: WRITE, Table2, GoSub
  3: IDLE, Table1, 4
  4: WRITE, Table5, Cond Jump
```

Timing Set: WRITE
Table: Table1
Loop: 1
Transition: GoSub
Destination: BUS_TEST[6]

The Sequence Editor allows sequences to be edited, validated, stored to file and exported as a SCPI file. Sequences containing one or more subsequences are presented in the Sequence List, a hierarchical list for viewing and editing. Subsequences are added, deleted, copied and pasted in the hierarchical list. Subsequences are each composed of a timing set name, table name, loop count and transition type. A branch (i.e. jump) transition also includes a destination and optional condition code. Defining a subsequence, with the Sequence Editor, is a simple matter of selecting these elements from validated lists in the Parameter Pane. In addition, lists of timing set and table names can be imported from various timing and table files.

When started, the Sequence Editor resumes the position it had during its previous execution. It also automatically reloads the last sequence file that was edited. The Sequence Editor can be moved, resized, minimized and maximized via standard Windows interface operations.

7.1 Sequence Operation

Sequences are a very powerful mechanism for controlling the execution of an SR192. This capability is provided by one or two SR101 Timing Set Modules that are installed in the SR192. This section further describes the sequencing capabilities of an SR101 module.

> **Note: The Sequence Editor only works with SR101 Timing Set Modules. It does not support the limited sequencing of an SR100 Timing Set Module.**

The purpose of sequences is to coordinate the execution of multiple timing set and table pairs. This allows a series of operations to be executed, such as sequential read/write cycles on a bus. With the capability of conditional branching and subroutines, the sequences can generate complex series of patterns and behavior.

Sequences are composed of one or more subsequences. A subsequence represents the execution of a timing set and table pair. For every word in the table the timing set is executed once. Each subsequence can be repeated up to 32768 times.

> **Note: Table and timing set names are truncated to ten characters when they are downloaded to the SR192 hardware. Sequences must reference the truncated names.**

When a subsequence has completed its execution, there is an automatic transition to the next subsequence. During that transition, the **IDLE** timing set is executed. Alternately, a branch (jump) operation can be used to move execution to a different subsequence. A subsequence with a branch operation executes the first word of its corresponding table and then jumps to the destination subsequence.

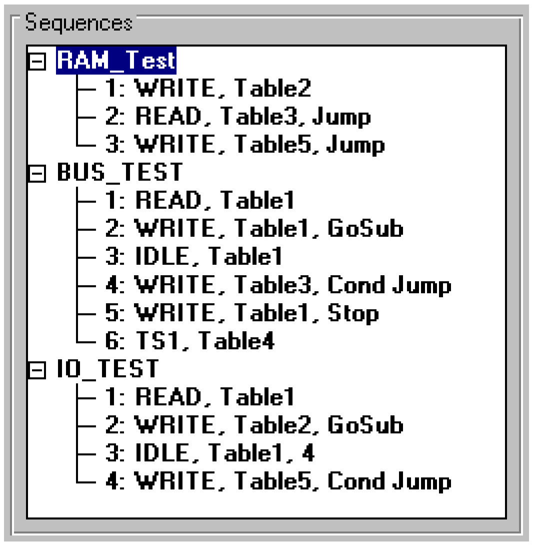
Conditional branching is available for the following test conditions: TSINPUT level, Response Error, Timeout and Jump Enable. Branch conditions are only tested during the execution of words in the table whose jump enable bit is set. The jump enable bit can be set individually for each word in a table (the default is off for all words). This provides additional control over the sequence execution.

Another form of branching is a subroutine call (gosub). A subsequence with a GOSUB executes a single table word and then branches to execute the destination subsequence. Upon completion of the destination subsequence, the next table word of the original subsequence is executed. The destination subsequence is executed after every word of the original subsequence.

Subroutine branching can also be made conditional with the same conditions outlined above for jumps.

7.2 Sequence List

The Sequence List is used to manage the order of sequences and subsequences. It provides a hierarchical view; where each sequence is identified by a name and related subsequences are indented underneath. Subsequences are displayed with Timing Set name, Table name and transition type. If Display Details is checked on the Options menu, subsequence entries also include destination and condition, when appropriate.



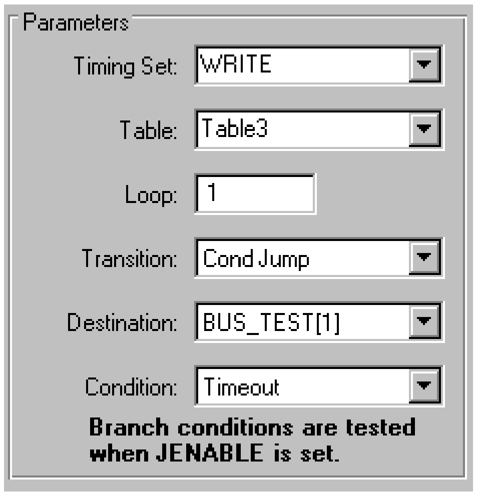
```
Sequences
⊟ RAM_Test
    ├─ 1: WRITE, Table2
    ├─ 2: READ, Table3, Jump
    └─ 3: WRITE, Table5, Jump
⊟ BUS_TEST
    ├─ 1: READ, Table1
    ├─ 2: WRITE, Table1, GoSub
    ├─ 3: IDLE, Table1
    ├─ 4: WRITE, Table3, Cond Jump
    ├─ 5: WRITE, Table1, Stop
    └─ 6: TS1, Table4
⊟ IO_TEST
    ├─ 1: READ, Table1
    ├─ 2: WRITE, Table2, GoSub
    ├─ 3: IDLE, Table1, 4
    └─ 4: WRITE, Table5, Cond Jump
```

Scroll bars appear at the bottom and on the right side of the Sequence List if any text extends beyond those boundaries. Selecting an item in the Sequence List causes its contents to be displayed for editing in the Parameter Pane. Changes made in the Parameter Pane are automatically updated in the Sequence List when Enter is pressed or a selection is made from a drop-down list.

The Edit menu has standard editing operations such as Cut, Copy and Paste. These are used to arrange the sequences and subsequences into their desired order.

7.3 Parameter Pane

The Parameter Pane is used to view and edit the properties associated with a sequence or subsequence. The format of the Parameter Pane presented is dependent upon the item selected in the Sequence List. For a sequence, the Parameter Pane contains a single text box for changing its name. For a subsequence, the Parameter Pane has up to six fields for setting its properties.


Parameters

Timing Set:	WRITE
Table:	Table3
Loop:	1
Transition:	Cond Jump
Destination:	BUS_TEST[1]
Condition:	Timeout

Branch conditions are tested when JENABLE is set.

For a subsequence, the top two fields define the Timing Set and Table names. These are selected from drop-down lists. Drop-down lists are activated by clicking the left mouse button on the down arrow button to the right of the text. These two name lists are loaded by importing names from timing and table files with the File menu. New Table and Timing Set names can also be added to the list by typing them in their respective boxes and pressing Enter. To remove names from these lists, use the Remove Name command on the Edit menu.

The next field defines the number of times to loop the subsequence. This value can be from 1 to 32768. Values outside of this range are automatically converted to the closest limit.

The Transition field defines how the sequence execution will proceed. To continue execution with the following subsequence, select a Next transition. To halt after executing one word from the table, select Stop. Branching to other subsequences is achieved by the Jump and GoSub selections. The Destination field contains a list of all of the subsequences to branch to. The UNKNOWN label is used if a subsequence has not yet been defined, or has been subsequently removed. If a conditional branch is selected, the Condition field appears. Branch conditions include input signals, a response error, a time-out and when allowed by the table's jump enable bit (JEN).

> **Note: Branch conditions are only tested during words (vectors) whose jump enable bit is set. See the Jump Enable dialog for more information on controlling tables' jump enables.**

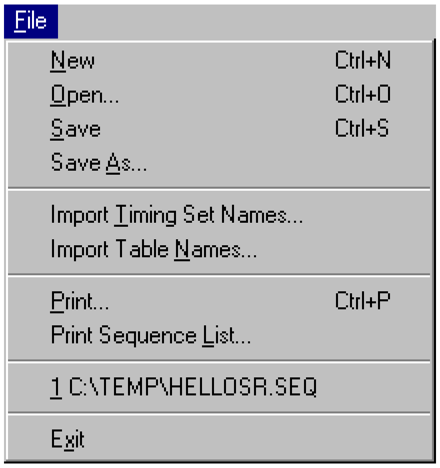
Sequence, timing set and table names can be up to ten alphanumeric and underscore characters. The first character must be an alpha character. The Sequence Editor automatically truncates these names to ten characters and replaces spaces by underscores. If a numeric or underscore is used as the first character, an 'N' is added to the beginning of the name.

7.4 Menus

7.4.1 File Menu

The Sequence Editor File Menu supports selection and management of sequence files, timing set and table lists. It also provides a list of recently accessed sequence files and includes a way to exit the Sequence Editor.

Sequence files are stored in a simple ASCII text format. The imported timing set and table names are used in drop-down lists for configuring subsequences.



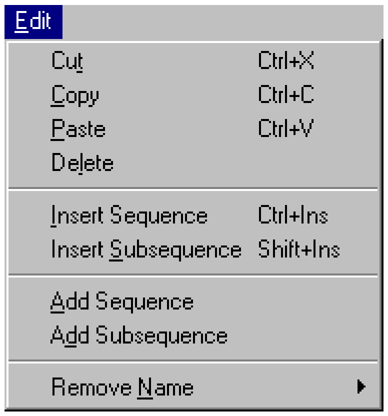
Menu Option	Description
New	Create and open a new sequence file.
Open…	Open an existing sequence (SEQ) file.
Save	Save the loaded sequence file.
Save As…	Save the loaded sequence file with a new name.
Import Timing Set Names…	Load Timing Set list with names extracted from a timing set file.
Import Table Names…	Load Table list with names extracted from a table file.
Print…	Send an image of the Sequence Editor to the printer. Also accomplished by pressing **Ctrl-P**.

Print Sequence List…	Send an image of the Sequence List to the printer.
File History	Open one of the last four sequence files accessed by the Sequence Editor.
Exit	Close the application. If editing changes have been made the operator is prompted to save the sequence file. File history, display options and Window position are saved to SEQEDIT.INI.

7.4.2 Edit Menu

The Sequence Editor Edit Menu allows the user to build and modify sequences. This includes cutting, copying and pasting sequences or subsequences. New items are added or inserted in the hierarchy. Additionally, the Timing Set and Table name lists are accessed from this menu. It is important to keep these lists current in order to best use the validate sequences command. References to unknown tables cause errors when a sequence is downloaded to an SR192.

The copy and paste operations can also move sequences and subsequences between different files. Simply copy a sequence in one file and load another file to paste it into.



Menu Option	Description
Cut	Remove the selected items from the Sequence list and place them in the paste buffer. Cutting a sequence also includes its subsequences.
Copy	Copy the selected items in the Sequence list to a paste buffer. A copy of a sequence includes its subsequences.
Paste	Insert the contents of the paste buffer above the selected item in the Sequence list. This option is disabled when there is nothing in the paste buffer. Also disallowed is pasting a subsequence at the top of the Sequence list.
Delete	Permanently delete the selected items from the Sequence list. Deleting a sequence also deletes all of its subsequences.
Insert Sequence	Insert a new sequence above the selected item in the Sequence list.

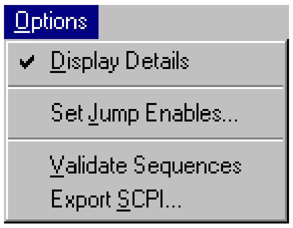
Insert Subsequence	Insert a new subsequence above the selected item in the Sequence list.
Add Sequence	Add a new sequence to the bottom of the Sequence list.
Add Subsequence	Add a new subsequence beneath the selected item in the Sequence list. If the selected item is a sequence, the new subsequence is added to the bottom of its subsequence list.
Remove Name	Select timing set or table names to delete with the Remove Name dialog.

7.4.3 Options Menu

The Sequence Editor Options Menu allows the user to toggle a detail display option, set jump enables for tables, validate sequences and export to a SCPI file.

> **Note: It is no longer necessary to export sequences to a SCPI file because the Test Manager and programming interface support the sequence file format (SEQ).**

Prior to loading an SR192 with sequences, it is highly recommended that the sequences be validated from this menu. Sequence validation includes verifying timing set and table names against imported lists. Jump and GoSub destinations are also verified that they exist.



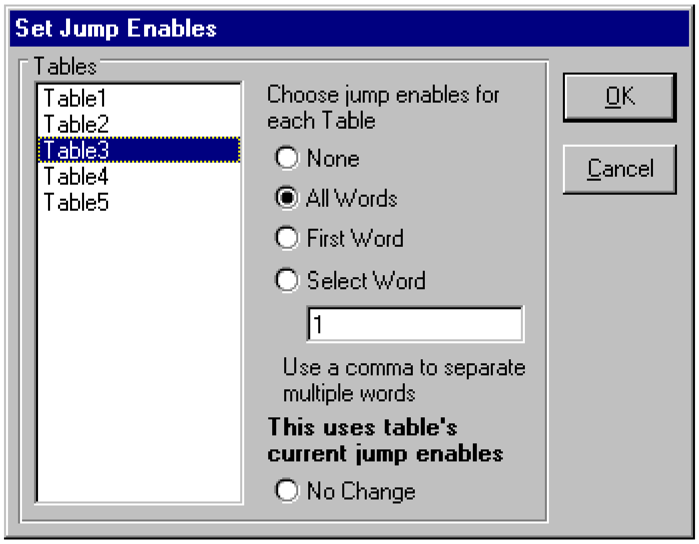
Menu Option	**Description**
Display Details	Select this item to display all subsequence information in the Sequence list. When not selected, subsequence information is displayed in a minimal format.
Set Jump Enables…	Initiates a dialog box for selecting jump enable options for each of the defined tables.
Validate Sequences	Validate timing set and table names against imported lists. Also verify Jump and GoSub destinations.
Export SCPI…	Export sequence definitions to a file of SR192 SCPI commands. Validates sequences prior to export.

When exporting to a SCPI file, a validation check is performed on the sequences and subsequences. If a validation error is encountered, the operator is notified and has the choice of continuing validation or canceling. A SCPI file is not created if there are validation errors. Once the sequences are validated, a file dialog is presented for entering or selecting the SCPI file name. To quit without creating a SCPI file, just cancel the file dialog.

7.5 Dialogs & Windows

7.5.1 Set Jump Enables

The Jump Enable dialog box is for viewing and editing jump enable options. These different options specify which words in a table have the jump enable bit set. During sequence execution, conditional branch conditions are only tested for words whose jump enable bit is set. Use the Remove Name dialog to manage this list of tables because a jump enable for a nonexistent table can cause an error when downloading a sequence file. When the sequences are exported to SCPI, the jump enable options are written at the beginning of the file.



Choose jump enables for each Table

- ○ None
- ● All Words
- ○ First Word
- ○ Select Word

1

Use a comma to separate multiple words

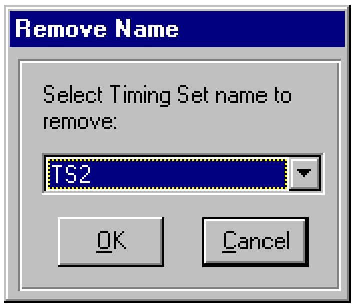
This uses table's current jump enables

- ○ No Change

Jump Enable Option	**Description**
None	Jump enable is not set for any words in the table. Conditional jumps will not occur while this table is executing.
All Words	Every word in the table has its jump enable set.
First Word	The first word of the table has its jump enable set.
Select Word	The specified comma delimited list of words have their jump enables set.
No Change	No change is made to the table's jump enable settings. This allows the table's jump enables to be set elsewhere (e.g. interactively).

7.5.2 Remove Name

The Edit Menu presents the choice of deleting a name from the current Timing Set list or the current Table list. When either option is selected, a Remove Name dialog box appears with a drop-down list of the corresponding names. Pick a name and press the OK button to remove the name from the list.



Names are added to the Timing Set and Table lists by importing them from external files or by entering them in the Parameter Pane. This dialog allows names to be removed that are invalid, misspelled or that no longer apply. These lists are used during validation to ensure that each subsequence references a known timing set and table.

Note: It is very important to keep the table list matched to the tables that will be present in the SR192. Otherwise, a jump enable setting for a nonexistent table can cause a sequence download to fail.

Note: Table and timing set names are truncated to ten characters when they are downloaded to the SR192 hardware. Sequences must reference the truncated names.

7.6 Sequence File Format

Sequence files (SEQ) are created and updated by the Sequence Editor. There is no requirement to edit or display these files directly. The following information is provided for completeness only.

The sequence file format is patterned after Windows' INI files. This is an ASCII text format that can be viewed or printed. Programmatic access to the contents of an INI file is quickly achieved through standard Windows' API functions (WritePrivateProfileString and GetPrivateProfileString).

The format of an INI file consists of sections and entries. A section is denoted by a bracketed name (e.g. [SequenceEditor]). Within a section, entries are labels that are assigned particular values. For example:

```
[SequenceEditor]
Version=1.1
TimingList=IDLE;READ;TS1;TS2;TS3;WRITE;
TableList=Table1,None;Table2,All;Table3;Table4;Table5,First;
```

For a sequence file there is one section devoted to general information and the other sections represent sequences. The sequence sections are named in numerical order (e.g. S1, S2, S3…). Within each sequence section, there is an entry for the sequence name and a numeric list of the subsequences.

Keywords		Description
[SequenceEditor]		Section header for general sequence information.
	Version =	File format version (1.1)
	TimingList =	Semicolon delimited list of timing set names.
	TableList =	Semicolon delimited list of table parameters (see below).
[S1]		Section header for first sequence.
	Name =	Name of the sequence.
	1 =	First subsequence definition.
	2 =	Second subsequence definition.
	:	Additional subsequence definitions
[S2]		Section header for second sequence.
	Name =	Name of the sequence.
	1 =	First subsequence definition.
	2 =	Second subsequence definition.
	:	Additional subsequence definitions

The format of a table list is:

TableList = *Table* [, *JumpEnable* [, *Word*]]; *Table* [, *JumpEnable* [, *Word*]]; …

The *Table* name is followed by an optional *JumpEnable* selection (None, All, First, Select, NoChange). If *JumpEnable* is not present, the default is NoChange. The optional comma-delimited *Word* field defines the vector numbers associated with the jump enable option: Select.

The format of a subsequence definition is:

Number = *TimingSet*; *Table*; *Loop*; *Transition*; *Destination*; *Condition*;

Unspecified parameters are represented by empty fields.

Note: The Sequence Editor does not automatically validate a sequence file before saving it. Consequently, a sequence file may contain invalid data (e.g. an unknown destination).

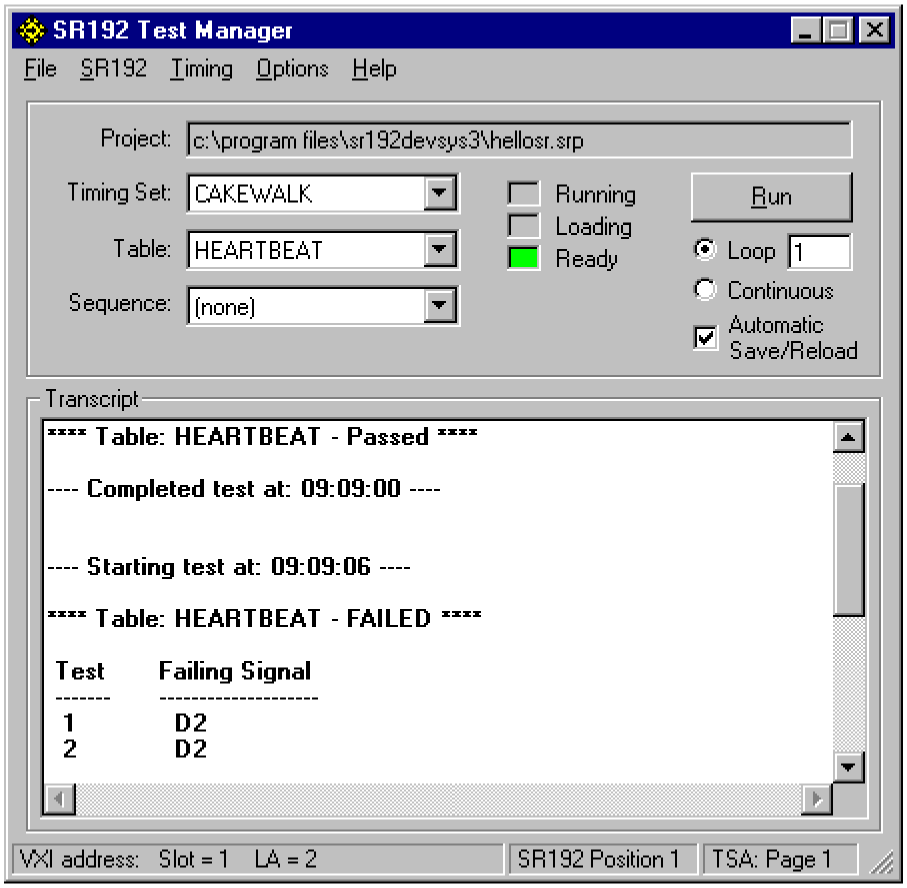
The following is the example sequence file (SEQ_DEMO.SEQ) included with the Sequence Editor:

```
[SequenceEditor]
Version=1.1
TimingList=IDLE;READ;TS1;TS2;TS3;WRITE;
TableList=Table1,Select,2,4,6;Table2;Table3,All;Table4;Table5;

[S1]
Name=RAM_Test
1=WRITE;Table2; 1;Next;;;
2=READ;Table3; 1;Jump;RAM_Test[1];;
3=WRITE;Table5; 1;Jump;RAM_Test[1];;

[S2]
Name=BUS_TEST
1=READ;Table1; 1;Next;;;
2=WRITE;Table1; 1;GoSub;BUS_TEST[6];;
3=IDLE;Table1; 1;Next;;;
4=WRITE;Table3; 1;Jump;BUS_TEST[1];Timeout;
5=WRITE;Table1; 1;Stop;;;
6=TS1;Table4; 1;Next;;;

[S3]
Name=IO_TEST
1=READ;Table1; 1;Next;;;
2=WRITE;Table2; 1;GoSub; IO_TEST[4];;
3=IDLE;Table1; 4;Next;;;
4=WRITE;Table5; 1;Cond Jump;IO_TEST[1];Response Error;
```

8. Test Manager

The Test Manager is one of the software components of the SR192 Development System. The Test Manager handles loading, executing and test result reporting for project files created with the SR192 Development Environment. The Test Manager is used with the SR192 Development Environment during debug and it also runs standalone for production testing. Source code for the Test Manager is provided for customization.

The Test Manager has an easy-to-use interface for loading SR192 project and sequence files, executing timing set/table pairs and reporting test results. The Test Manager also supports reading SR192 test data and writing it to a project file. In addition, the Test Manager has looping and continuous run modes.



Project: c:\program files\sr192devsys3\hellosr.srp
Timing Set: CAKEWALK
Table: HEARTBEAT
Sequence: (none)
Running
Loading
🟩 Ready
[● Loop] 1
Continuous
✓ Automatic Save/Reload

Transcript

```
**** Table: HEARTBEAT - Passed ****

---- Completed test at: 09:09:00 ----


---- Starting test at: 09:09:06 ----

**** Table: HEARTBEAT - FAILED ****

 Test      Failing Signal
 -------   --------------------
  1          D2
  2          D2
```

VXI address: Slot = 1 LA = 2 | SR192 Position 1 | TSA: Page 1

To use the Test Manager simply load a project file, select an appropriate timing set/table pair and press the Run button. Test results are reported in the Transcript window and operating status is indicated by LED-like controls.

8.1 Operation

The Test Manager provides a bridge between the SR192 Development Environment and the SR192 hardware. It coordinates project loading, execution and test result reporting. The following describes the operation of the Test Manager interface.

The Project field shows the current path and name of the loaded SR192 project file. A project file is loaded, or reloaded, with the File Menu. The Timing Set and Table drop-down lists have the timing set and table names of the selected SR192's specified timing generator and page (e.g. TSA: Page 2). The current SR192 and timing generator are selected from the SR192 and Timing menus. The optional Sequence drop-down list has the sequence names from the selected SR192.

Pressing the Run button causes the SR192 to run the selected timing set/table pair or sequence. While running, the button is labeled Stop. Pressing the Stop button immediately halts the SR192. The Stop button is always used to halt the SR192 when it is running in Continuous mode. The Loop value defaults to one, but it can be set up to a count of 9999.

> **Note: A selected sequence has execution priority over a timing set/table pair. Set the sequence to "(none)" in order to execute an individual timing set/table pair.**

When a run is initiated, several checks are performed to ensure that the project and sequence files are current. The exact behavior of these checks is controlled by the Automatic Save/Reload checkbox. Following a run, the Test Manager distributes the execution results to any active SR192 Development Environments for use in their error displays.

The Transcript window is a continuous journal of test activity and test results. Its contents can be cut (**Ctrl-X**), copied (**Ctrl-C**), pasted (**Ctrl-V**) or cleared with the Options Menu. When the Transcript window buffer is full, a portion of the oldest data is deleted. During normal operation, the Transcript window logs test start, Pass/Fail status and displays detailed test failure information. When looping, only the last pass is tested for errors. In Continuous mode, or on early termination while looping, no Pass/Fail analysis is performed because the results would be inconclusive. The Transcript window also displays any error messages reported by the system.

There are three status LED's labeled Running, Loading and Ready. Initially, all three LED's are grayed out. Once an SR192 project file is loaded, the Ready LED turns green. While loading or reading a project file, the Loading LED turns yellow. The Running LED turns yellow while the SR192 is actively running.

The status bar at the bottom of the Test Manager window shows the currently selected SR192's Slot, Logical Address, logical position and Timing Generator/Page.

The Test Manager is written in Visual Basic and its source code is provided (<Install_Dir>\Examples\TestMan\*.*). Control of the SR192 is through the SR192 ActiveX Driver DLL. For production testing, the Test Manager may be used as supplied or customized to meet specific requirements. Alternately, the Test Manager can serve as a nontrivial example of interfacing to the SR192 ActiveX Driver DLL.

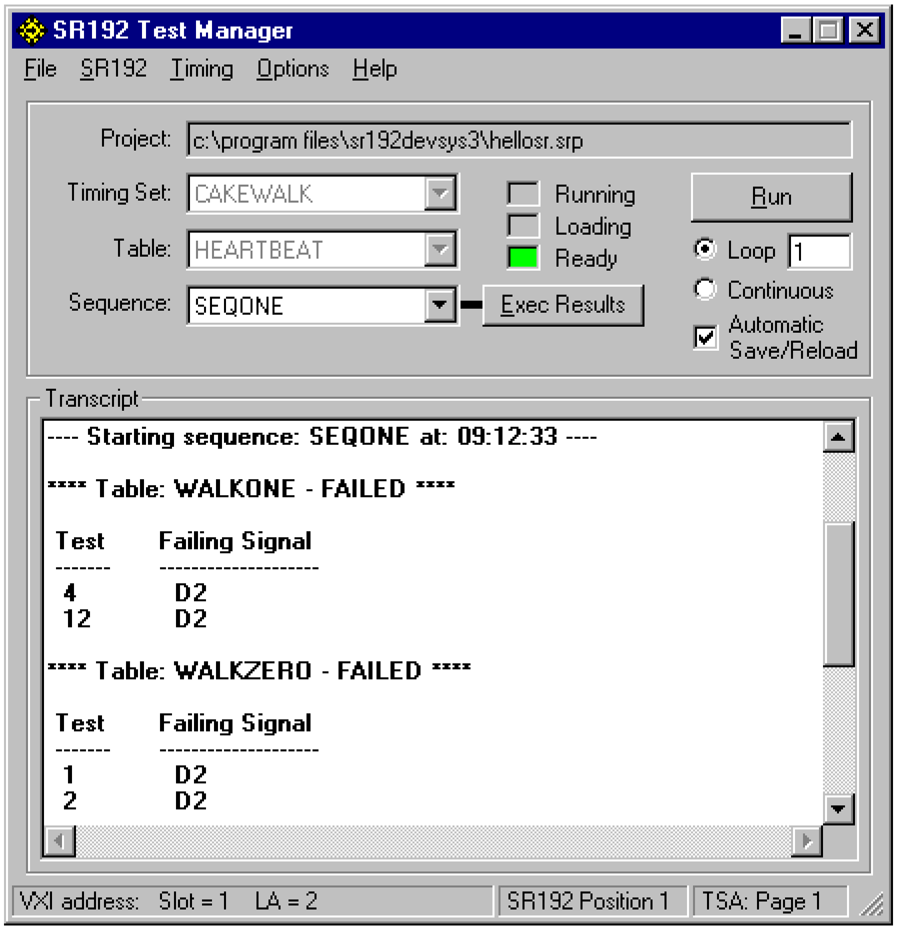
8.2 Sequence Execution

Sequences coordinate the execution of multiple timing set and table pairs. This allows a series of operations to be executed, such as sequential read/write cycles on a bus. With the capability of conditional branching and subroutines, sequences can generate complex series of patterns and behavior.

> **Note: Sequences only work with SR101 Timing Set Modules. The Test Manager does not support the limited sequencing of an SR100 Timing Set Module.**

Once a sequence completes, its execution status (i.e. pass or fail) is determined by checking each of the executed tables for errors. Since sequences can be highly complex, the test developer must define the tables to query after a sequence executes.

When a sequence is selected, on the Test Manager window, the Execution Results ("Exec Results") button becomes visible. The appearance of this button is to emphasis that sequence errors are only reported for the tables selected on the Execution Results dialog.



Project: c:\program files\sr192devsys3\hellosr.srp

Timing Set: CAKEWALK

Table: HEARTBEAT

Sequence: SEQONE

[ ] Running
[ ] Loading
[🟩] Ready

[● Loop] 1
○ Continuous
[✓] Automatic Save/Reload

```
---- Starting sequence: SEQONE at: 09:12:33 ----

**** Table: WALKONE - FAILED ****

 Test      Failing Signal
 -------   --------------------
  4          D2
  12         D2

**** Table: WALKZERO - FAILED ****

 Test      Failing Signal
 -------   --------------------
  1          D2
  2          D2
```

VXI address: Slot = 1   LA = 2 | SR192 Position 1 | TSA: Page 1

8.3 Automatic Save/Reload

The Test Manager monitors the status of project and sequence files in order to ensure that the latest changes have been downloaded to the hardware. This is particularly valuable during development and debug when a rapid edit-load-run cycle can be disrupted by forgetting to save or reload an important change. Equally important is to eliminate the suspicion that an update was missed (Did I…?).

The Automatic Save/Reload checkbox controls the save and reload process. If checked, the Test Manager automatically instigates any required save or reload of project or sequence files. If unchecked, the user is notified when unsaved or modified files are detected. The user then decides whether to save and/or reload the applicable files.

When a run is initiated, the Test Manager first queries active Sequence Editors and SR192 Development Environments to determine if editing changes have been made to the currently loaded sequence or project file. If project changes are detected, the applications are instructed to perform a file save.

As a second step, the Test Manager determines if any files have been modified since the last hardware download. This step is independent of the first because changes could have been manually saved by the operator prior to starting the run. Modified files are detected and downloaded to the SR192.

Note: If a project file is reloaded, the sequence file is also reloaded in order to maintain synchronization with table and timing set locations.

Under certain conditions, it may be best to temporarily disable (i.e. uncheck) the automatic save/reload mechanism. This might occur when a project is partially modified and you are not ready for it to be downloaded during the next run. Or, when the contents of the SR192 have been altered by another application and you want to test the modifications.

Even under normal circumstances it is still valuable to avoid extraneous editing of a project file while debugging with the Test Manager. Any editing of the project file can cause a save and reload to occur. This includes reordering signals, toggling a signal state, examining the contents of a drop-down list or selecting a signal name. To best avoid inadvertent changes to a project, you should use Cancel to close dialog boxes and discard tables when closing the Table Editor.

Even when automatic save/reload is disabled, the user is still notified if a modification is detected and is given the opportunity to download the changes. To force a complete file reload, use the Options Menu to reset the SR192.

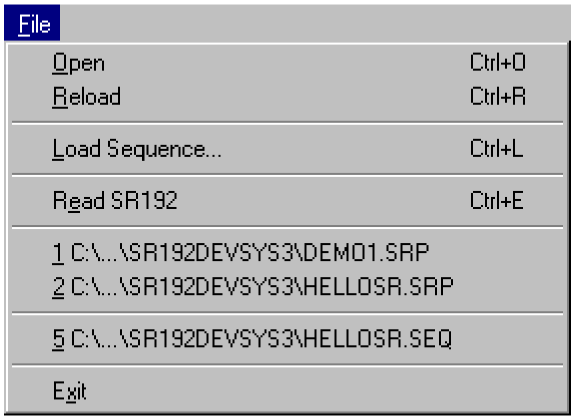
8.4 Menus

8.4.1 File Menu

The Test Manager File Menu supports selection and management of SR192 project files. The read command is particularly useful for identifying SR192 hardware configurations and converting legacy test data to a project file. Sequence files created with the Sequence Editor are downloaded via this menu also.

Note: Sequences are downloaded to the currently selected SR192 and timing generator.

Performing a read on a newly initialized SR192 generates a project file that only contains the hardware configuration. This can be used as a boilerplate when starting a new SR192 project in the SR192 Development Environment. After loading an SR192 with legacy test data, the read command can be used to retrieve it and store it in an SR192 project file.


File
Open Ctrl+O
Reload Ctrl+R
Load Sequence... Ctrl+L
Read SR192 Ctrl+E
1 C:\...\SR192DEVSYS3\DEMO1.SRP
2 C:\...\SR192DEVSYS3\HELLOSR.SRP
5 C:\...\SR192DEVSYS3\HELLOSR.SEQ
Exit

Menu Option	**Description**
Open	Opens a file browser to select a project file to download to the SR192 hardware.
Reload	Reloads current project file and updates the SR192.
Load Sequence…	Select and download a file of sequence definitions to the SR192 hardware.
Read SR192	Uploads the contents of the SR192 hardware and writes it to a project file.
File History	Loads one of the last four project, or sequence, files. The file names are stored in a state file (SR192TM.INI).

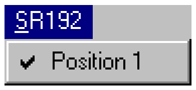
Exit	Closes the Test Manager. The contents of the SR192 are left intact.

Note: Timing sets and tables must be loaded into the SR192 before loading sequences that reference them. Errors are reported if a sequence references a nonexistent timing set or table.

For compatibility with existing sequences, the load sequence operation also supports downloading ASCII SCPI files. This also permits direct control over other aspects of the SR192 hardware via SCPI commands. When sequences are defined in a SCPI file, the load function does not automatically delete existing sequences in the SR192. This can cause multiple-defined parameter errors. To ensure that existing sequences are deleted, include "**SEQ:DEL:ALL**" at the beginning of a SCPI sequence file.

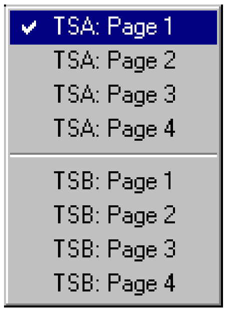
8.4.2 SR192 Menu

The Test Manager SR192 Menu contains a list of detected SR192 modules. If only one SR192 is installed, the menu shows only one entry. If multiple SR192 modules are present, the menu lists each module by logical position with a check next to the currently selected one. Only one module is selected at a time. When a selection is changed, the Timing Set, Table and Sequence drop-down lists are updated to reflect the new setting. The current setting is displayed on the status bar.



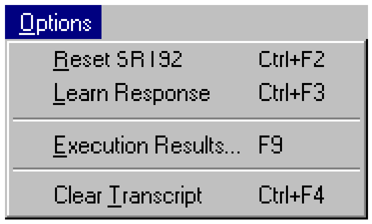
8.4.3 Timing Menu

The Test Manager Timing Menu selects a timing generator and page for operation. Selection is performed by clicking on the desired timing generator page. Once selected, the Timing Set, Table and Sequence drop-down lists are updated with the contents of the specified timing generator page. The current selection is also displayed on the status bar. Items on this menu are only enabled if they are detected in the SR192 hardware.



8.4.4 Options Menu

The Test Manager Options Menu provides additional control over the user interface and SR192 hardware. The Learn Response command is used to update test data with UUT responses recorded during a test execution. Use the Read SR192 command to build a project file from the updated test data. For more information on the learn behavior, see the ssSrLearnResponse function.

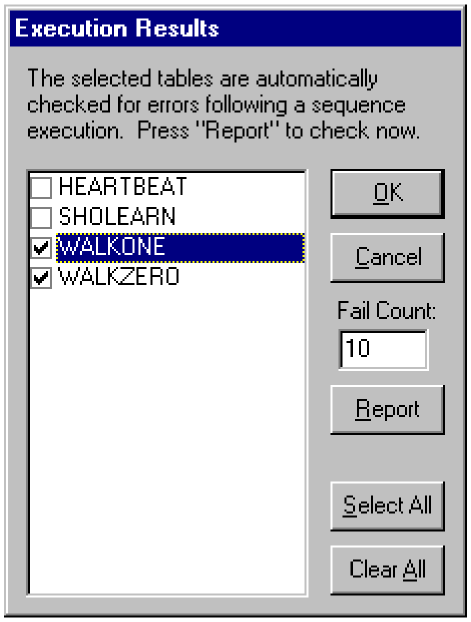


Menu Option	Description
Reset SR192	Causes a hard reset of the SR192. This initializes it to a power-on state. All timing sets, tables, pin groups and sequences are deleted.
Learn Response	Updates selected table with UUT responses recorded during a prior test execution. Only modifies signals that are designated as monitored high or low.
Execution Results…	Activates the Execution Results dialog to select tables for error reporting.
Clear Transcript	Clears the contents of the Transcript window.

8.5 Execution Results

The Test Manager's Execution Results dialog is used to select data tables for error reporting. The selected tables are queried following a sequence execution, or immediately if the Report button is pressed. Since a sequence can cause one or more timing set/table pairs to be executed, it is necessary to select the tables that are to be queried for their test status.

Fail Count defines the number of failing vectors to detect and report. This greatly influences the Error Display, by the Table Editor, and the contents of the execution result file. The goal is to report a sufficient number of errors, for debugging and diagnostic purposes, without expending the time and resources required to report all errors.


Execution Results

The selected tables are automatically checked for errors following a sequence execution. Press "Report" to check now.

- [ ] HEARTBEAT
- [ ] SHOLEARN
- [✓] WALKONE
- [✓] WALKZERO

Fail Count: 10

Use the OK button to accept the changes and close the dialog. The Cancel button discards the changes and closes the dialog. The Select All and Clear All buttons quickly check, or uncheck, all the tables in the list. Use the Report button to immediately report the status of the selected tables. This allows a report to be created without an additional execution.

8.6 Execution Result File Format

The Test Manager creates an execution result file (FAULT.RES) whenever a test is run. For a sequence, this is built based on the tables selected by the Execution Results dialog. This file is used by the SR192 Development Environment to set corresponding Error Displays. There is no need to edit or display these files directly. The following information is provided for completeness only.

The execution result file format is patterned after Windows' INI files. This is an ASCII text format that can be viewed or printed. Programmatic access to the contents of an INI file is quickly achieved through standard Windows' API functions (e.g. GetPrivateProfileString).

The format of an INI file consists of sections and entries. A section is denoted by a bracketed name (e.g. [FaultFile]). Within a section, entries are labels that are assigned particular values. For example:

```
[FaultFile]
Project = c:\program files\sr192devsys3\demo1.srp
```

For an execution result file there is one section that holds the test results for multiple tables. Each table is identified by name prior to the listing of test results. If a table entry is not followed by vector/signal failures, that means the table has no errors to report.

Keywords		**Description**
[FaultFile]		Section header for execution result information.
	Project =	SR192 project file name and path.
	FailCount =	Maximum number of vector failures being reported.
	TableX =	SR Module and table X name (matches window title of Table Editor).
	V1 S1	First failing vector and signal for table X.
	V2 S2	Second failing vector and signal for table X.
	V3 S3	Third failing vector and signal for table X.
	:	Additional failing vectors and signals for table X.
	TableX+1 =	SR Module and table X+1 name.
	V1 S1	First failing vector and signal for table X+1.
	V2 S2	Second failing vector and signal for table X+1.
	V3 S3	Third failing vector and signal for table X+1.
	:	Additional failing vectors and signals for table X+1.

The Test Manager creates this file by intermixing direct file writes with calls to ssSrGetExecResults. The direct file operations supply the necessary INI formatting and project information. The ssSrGetExecResults function appends the vector/signal failure results. The code for creating this file is included with the supplied Test Manager source (<Install_Dir>\Examples\TestMan\*.*).

The following is an example execution result file. Notice that Table1 reports two signal failures for the same vector, and Table2 has no failures to report.

```
[FaultFile]
Project = c:\program files\sr192devsys3\demo1.srp
FailCount = 10
Table1 = \SR Module: 1\Tables\HEARTBEAT
 3 D4
 3 D2
 10 D6
 17 D9
Table2 = \SR Module: 1\Tables\WALKONE
Table3 = \SR Module: 1\Tables\WALKZERO
 4 D2
 6 D4
 12 D2
 14 D4
```

9. Programming Interface

Most production test scenarios involving the SR192 can be handled by the Test Manager. In some cases, small modifications to the Test Manager are all that is required. For that reason the source code for the Test Manager is supplied (<Install_Dir>\Examples\TestMan\*.*). Some testing situations require a custom application or close integration with existing systems and procedures. To meet these requirements, a programming interface is included as part of the SR192 Development System.

The SR192 ActiveX Driver DLL (ssSr192X.DLL) provides a high-level software interface and execution resource for the SR192 hardware. It contains a set of high-level function calls designed specifically to work with SR192 project files created with the SR192 Development Environment. These function calls internally manage SR192 low level operations such as selection, error checking and test result reporting, thereby freeing the application programmer from this burden. Since the ssSr192X.DLL is ActiveX compliant, it may be used by any application development environment that supports ActiveX calling protocol. This includes the latest commercial versions of HP-VEE, National Instruments LabWindows/CVI, National Instruments LabView, Visual C++ and Visual Basic.

The SR192 ActiveX Driver application program interface (API) contains a set of function calls that operate on the currently selected SR192 module. The default module is one (1), the leftmost SR192 module in the VXI chassis. Other modules are selected with the ssSrSelectSR192 function. Returned error values are negative. They can be passed to ssSrGetErrorMessage for translation into a descriptive string.

ActiveX components are object oriented and thereby require object instantiation and reference for the API calls. The name of the SR192 ActiveX Driver object is **ssSr192drv**. This name must be used when creating an instance of the object for accessing the API. Please refer to your application development environment's documentation for the proper instantiation and calling syntax. General guidelines for various popular environments are covered in a later section of this document, and examples are provided in a subdirectory to the SR192 Development System (<Install_Dir>\Examples). Additionally, Visual Basic and Visual C++ examples are included in this document for reference purposes.

The SR192 ActiveX Driver has comprehensive error checking and reporting. A description of the error code and the context that it occurred in is retrieved with the ssSrGetErrorMessage function. In addition, all errors are written to a log file (ssSr192X.LOG**)**. A description of the log file and its entries are outlined in a subsequent section.

The following is an alphabetical list of the SR192 driver functions with a short description of each.

ssSrBuildProject (*projectFile*)
Reads the contents of an SR192 and writes it all to a specified project file.

ssSrExecute (*tgPage*, *timingSetName*, *tableName*, *loopCount*)
Starts execution of the specified timing set and table. Return immediately without waiting for execution to complete.

ssSrExecuteSequence (*tgPage*, *sequenceName*, *loopCount*)
Starts execution of the specified sequence. Return immediately without waiting for execution to complete.

ssSrGetConfig (*iSlot*, *iLa*, *tgCount*, *tgLinked*)
Retrieves from the selected SR192 its slot location, logical address, number of

installed timing generators and whether timing generators A and B are linked.

ssSrGetErrorMessage (*errNum*, *buffer*, *bufSize*)
Fills the specified buffer with a description of the error represented by the specified error value. The description is truncated to *bufSize* value.

ssSrGetExecResults (*tgPage*, *tableName*, *failCount*, *faultFile*, *appendFlag*)
Returns the results of the specified table's execution (1 = Pass, 0 = Fail). If the *failCount* is greater than zero, writes individual signal/vector failures to the specified fault file.

ssSrGetList (*listType*, *tgPage*, *listBuffer*, *bufSize*)
Fills the specified buffer with a comma-delimited list of items from the selected SR192's timing generator page. The list is truncated to *bufSize*.

ssSrGetTestResults (*failCount*, *faultFile*)
Returns the results of the last test execution (1 = Pass, 0 = Fail). If the *failCount* is greater than zero, writes individual signal/vector failures to the specified fault file.

ssSrHalt ( )
Halts the execution of the current SR192. Preserves the current state and contents of the SR192.

ssSrLearnResponse (*tgPage*, *timingSetName*, *tableName*)
Updates specified table with UUT responses recorded during a test execution. Only modifies signals that are designated as monitored.

ssSrLoadProject (*projectFile*)
Loads the SR192 hardware with the contents of the specified project file.

ssSrLoadSequence (*tgPage, sequencetFile*)
Loads the SR192 hardware with the contents of the specified sequence file.

ssSrReset ( )
Resets the current SR192 to a power-up state.

ssSrSelectSR192 (*modulePosition*)
Selects an SR192.

ssSrStatus ( )
Retrieves the status of the current SR192.

The SR192 ActiveX Driver DLL utilizes both VXI Plug and Play (VPP) VISA and the Talon SR192 VPP-4 compliant A32 driver in its implementation. This makes it compatible with any other VPP-compliant application for the Win32 framework.

For operating efficiency, the ActiveX Driver makes significant use of low-level register based commands when communicating with the SR192. In particular, all the table memory accesses are register based A32 read/writes to optimize load/unload times. The ActiveX Driver also employs cache technology to retain SR192 state information in order to reduce VXI bus traffic.

9.1 Execution Result Reporting

Reporting execution results, by the SR192 ActiveX Driver, has been enhanced via the addition of a new function, ssSrGetExecResults. This function is a more general implementation of the original reporting function, ssSrGetTestResults. Specifically, ssSrGetExecResults retrieves execution status for a named table. This allows it to be used following sequence executions (ssSrExecuteSequence) and direct *Plug&Play* driver operations. The original function, ssSrGetTestResults, only retrieves execution status for the table most recently executed by ssSrExecute. This function's focused behavior can cause misleading test results following sequence executions or direct driver operations.

The new ssSrGetExecResults function is optimized and runs faster than the existing ssSrGetTestResults function. For this reason it is recommended that ssSrGetExecResults be used instead of ssSrGetTestResults for execution results reporting in future applications. The ssSrGetTestResults function remains unchanged from previous driver versions. Any existing fielded programs will run without recompilation.

Caveats and Concerns

Here are some things to check for if strange (goofy) errors are reported in the fault results file:

1. *Power Supply and I/O Thresholds:* Make sure that programmable thresholds are set properly and enough delay is programmed in the test application to allow power supplies to cycle and stabilize.

2. *Wait for SR192 to Complete Operation:* Make sure that the SR192 execution is completed before invoking the ssSrGetExecResults function. A good way of doing this is to use the ssSrStatus function in a loop and wait until both Timing Generators A and B have stopped running.

3. *SR192 Hardware Error Count:* The SR192 hardware error counter is global to all tables while the ssSrGetExecResults function only checks the specified table for errors. It is possible to have the hardware error counter set (global indicator) and the ssSrGetExecResults report no errors (local table indicator).

4. *Unused Channels:* In Version 2.11.0008 undeclared channels were not initialized and would be set to power-on default values. In Version 2.11.0009 these channels are programmatically initialized to sense/ignore. Conceivably, this undeclared channel initialization difference between V2.11.009 and earlier versions could cause the UUT to operate differently. This can be avoided and/or corrected by programmatically setting any undeclared critical control signals.

9.2 Driver Updates

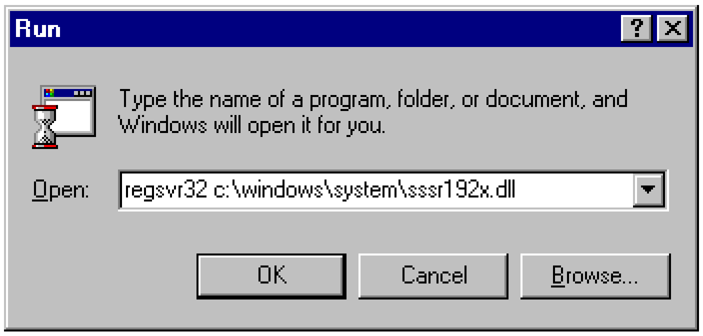
This section describes recent SR192 ActiveX driver updates and the changes that have been implemented in each. It is a good practice to keep your drivers updated in order to avoid diagnosing a problem that has already been solved.

Because the SR192 driver is an ActiveX component, each new version must be registered with the operating system. The following are the steps to take when you receive a new SR192 ActiveX driver.

1. Copy the driver to the Windows System directory. Depending on the specific operating system, this directory may be named differently (e.g. "c:\windows\system\" or "c:\winnt\system32\").

2. Register the driver by executing the following from the Run command on the Start Menu. The directory path must match where the driver was copied in step 1.

```
regsvr32 c:\windows\system\sssr192x.dll
```


Run
?
Type the name of a program, folder, or document, and Windows will open it for you.
Open:
OK
Cancel
Browse...

Open: regsvr32 c:\windows\system\sssr192x.dll

Software Differences Between V2.12.0001 and V2.12.0002

1. *SsSrBuildProject Function:* When reading back the contents of the SR192, and there are split signals present (output and expect memories have different states), and the split signal has imbedded tristate or ignore states, the readback marked the signal incorrectly as bi-directional (BiDr). This was corrected in V2.12.0002.

Software Differences Between V2.12.0000 and V2.12.0001

1. *Large Table Sizes Running under Windows 2000:* Loading large table sizes (28k+) running under Windows 2000 would cause the system to crash. This was corrected in V2.12.0001.

2. *Loading Large Tables on a Fast Computer*: Large tables sometimes failed to load if the computer was faster than the SR192 firmware. This was corrected in V2.12.0001

Software Differences Between V2.11.0009 and V2.12.0000

1. *Tables managed by Timing Generator B corruption:* Under certain conditions when timing Generator A and Timing Generator B are not linked, tables managed by Timing Generator B would be incorrectly loaded. This was corrected in V2.12.0000.

Software Differences Between V2.11.0008 and V2.11.009

1. The ssSrGetExecResults function was added to the ssSr192x.DLL beginning at Version 2.11.0008. Version 2.11.0009 includes the following corrections and improvements.

2. *SR192 Linked TGA and TGB operating mode:* In Version 2.11.0008, signals associated with the I/O channels physically located in DRB01-DRB06 were not evaluated for errors. This is corrected in Version 2.11.0009.

3. *Space Named Signals:* Some users placed empty rows in the SR192 Development System Graphical Table for readability purposes. In Version 2.11.0008 these signals were interpreted as an empty (space) named signal and any errors detected on these channels would be reported in the fault file as a test step only (with no signal) entry. In Version 2.11.0009, empty (space) named signals are not evaluated.

4. *Masking Unused Channels:* In Version 2.11.0008 only channels associated with signals in the SR192 Development System Graphical Table are initialized. The unused (undeclared) channels are not initialized and could trigger a bogus error indication. The power on setting of the SR192 does not guarantee I/O channel masking for errors. In Version 2.11.0009, the ActiveX driver DLL programs all the unused (undeclared) channels to the sense/ignore state when the tables are initially loaded. This masks unused channels evaluation for errors and prevents bogus error reporting.

5. *Empty Table Name:* In Version 2.11.0008 if an empty (space) or invalid table name is passed to the ssSrGetExecResults function, under certain conditions, the function would exit early and report a pass condition (return value of one) without performing test result evaluation. Additional error checking has been added in Version 2.11.0009 to correct this condition.

9.3 Error Log File

The SR192 ActiveX Driver logs all errors to a running journal file named **ssSr192X.LOG**. Entries to the log file are appended and roll over when the file reaches 10k in size. The entries are ASCII text and can be viewed using the Notepad or any text editor of choice. Since only errors are logged, the file is inactive during normal operation, thereby minimizing any operating overhead.

In addition to error entries, a session start and stop time stamp is logged every time the ssSr192X.DLL is loaded and unloaded. This brackets any reported errors to a specific session and its relative time and date.

The messages written to the error log file conform to the following convention:

Error: <SSeCode>**:**<VPPeCode> <Error source> **reported** <"VISA/VPP Driver Error Message"> **while** <operational context description when error occurred>.

An example error message is shown below:

```
Error: BFFC0FFB:BFFF0015 SR192 reported "Timeout expired before operation completed"
          while reading Timing Set directory.
```

The VPPeCodes are errors reported by VISA and the VXI Plug&Play (VPP) Driver. A list of these codes and their descriptions are found in the VPP VISA and VPP driver documentation. The SSeCodes are generated by the SR192 ActiveX Driver and their descriptions follow:

Constant Label	Error Code	Description
SS_ERROR_PJOPEN	(BFFC0FFF)	Error opening/closing project file.
SS_ERROR_PJACCESS	(BFFC0FFE)	Error reading/writing project file.
SS_ERROR_OSFAIL	(BFFC0FFD)	Operating system error.
SS_ERROR_RMFAIL	(BFFC0FFC)	VISA Resource Manager error.
SS_ERROR_SRFAIL	(BFFC0FFB)	SR192 VPP/VISA error.
SS_ERROR_ESCAPE_TIMEOUT	(BFFC0FFA)	Escape timeout expired.
SS_ERROR_SELECT_FAIL	(BFFC0FF9)	Unable to locate SR192 in VXI chassis.
SS_ERROR_SRSYS	(BFFC0FF8)	SR192 SCPI error.
SS_ERROR_TABLEERR	(BFFC0FF7)	SR192 table assignment error.
SS_ERROR_PROGRAM	(BFFC0FF6)	SR192 application error.

9.4 Functions

The following sections describe each of the functions available from the SR192 ActiveX Driver DLL. The descriptions include calling syntax, parameter lists, return values and operation.

9.4.1 ssSrBuildProject

VB	**Function**	*object*.**ssSrBuildProject (** *projectFile* As String **) as Long**
C++	**long**	*object*->**ssSrBuildProject (** BSTR **projectFile* **)**

Parameter	**I/O**	**Description**
projectFile	in	Project file name and path. This must be an existing SR192 project file.

Return Value

0 = Successful operation. Negative values are error codes. To get a description of the error, pass the error code to ssSrGetErrorMessage.

Remarks

This function reads the contents of the selected SR192 and stores this information in the specified *projectFile*. This function can be used to read the hardware configuration of an initialized SR192. This can then serve as a boilerplate for project files in the SR192 Development Environment.

> **Note: This function modifies an existing SR192 project file. To create a project file from the contents of an SR192, use the "Read SR192" command on the Test Manager's File Menu.**

This function can also be used to transfer legacy SR192 test data into a project file for viewing or updating. Since the SR192 hardware does not store signal names internally, uploaded test data has signal names assigned in the form of <PinGroupName>.<SR192ChannelNo>. The exception to this is when the test data originates from the SR192 Development System. For example, signal names are preserved when ssSrLearnResponse is used to capture UUT responses to a project file execution.

9.4.2 ssSrExecute

VB	**Function**	*object*.**ssSrExecute (** *tgPage* As String**,** *timSetName* As String**,** *tableName* As String**,** ByVal *loopCount* As Long **) As Long**
C++	**long**	*object*->**ssSrExecute (** BSTR **tgPage***,** BSTR **timSetName***,** BSTR **tableName***,** long *loopCount* **);**

Parameter	**I/O**	**Description**
tgPage	in	Timing generator/page select. “A1” through “A4” for Timing Generator A pages 1 through 4 and “B1” through “B4” for Timing Generator B pages 1 through 4.
timSetName	in	Name of timing set to run.
tableName	in	Name of table to run.
loopCount	in	Run loop count (1 to 65535). A negative value causes it to loop continuously.

Return Value

0 = Successful operation. Negative values are error codes. To get a description of the error, pass the error code to ssSrGetErrorMessage.

Remarks

Starts execution of the specified timing set and table pair on the currently selected SR192. The loop count specifies the number of times for the cycle to repeat. Returns immediately without waiting for the SR192 to complete. Use the ssSrStatus function to monitor test completion. Once the test is complete, use the ssSrGetExecResults function to retrieve the table's execution status.

If the SR192 is running, this function waits for up to ten seconds for it to complete the previous operation before starting execution.

> **Note: Table and timing set names are truncated to ten characters when they are downloaded to the SR192 hardware. This function must reference the truncated names.**

9.4.3 ssSrExecuteSequence

VB	**Function**	*object*.**ssSrExecuteSequence (** *tgPage* As String**,** *seqName* As String**,** ByVal *loopCount* As Long **) As Long**
C++	**long**	*object*->**ssSrExecuteSequence(** BSTR **tgPage***,** BSTR **seqName***,** long *loopCount* **);**

Parameter	**I/O**	**Description**
tgPage	in	Timing generator/page select. “A1” through “A4” for Timing Generator A pages 1 through 4 and “B1” through “B4” for Timing Generator B pages 1 through 4.
seqName	in	Name of sequence to run.
loopCount	in	Run loop count (1 to 65535). A negative value causes it to loop continuously.

Return Value

0 = Successful operation. Negative values are error codes. To get a description of the error, pass the error code to ssSrGetErrorMessage.

Remarks

Starts execution of the specified sequence on the currently selected SR192. The loop count specifies the number of times for the cycle to repeat. Returns immediately without waiting for the SR192 to complete. Use the ssSrStatus function to monitor test completion. Once the test is complete, use the ssSrGetExecResults function to retrieve execution status for the tables associated with the sequence. Sequences are loaded with the ssSrLoadSequence function.

If the SR192 is running, this function waits for up to ten seconds for it to complete the previous operation before starting execution.

Note: Sequence names are truncated to ten characters when they are downloaded to the SR192 hardware. This function must reference the truncated name.

9.4.4 ssSrGetConfig

VB	**Function**	*object*.**ssSrGetConfig (** *iSlot* As Long**,** *iLa* As Long**,** *tgCount* As Long**,** *tgLinked* As Long **) As Long**
C++	**long**	*object*->**ssSrGetConfig (** long **iSlot***,** long **iLa***,** long **tgCount***,** long **tgLinked* **);**

Parameter	**I/O**	**Description**
iSlot	out	Reports the physical slot location in the VXI chassis.
iLa	out	Reports the VISA logical address.
tgCount	out	Reports the number of timing generators installed in the SR192 (0, 1 or 2).
tgLinked	out	Reports if timing generators A and B are linked. (0 = Not Linked, 1 = Linked)

Return Value

Returns the number of SR192 instruments detected in the VXI chassis. Negative values are error codes. To get a description of the error, pass the error code to ssSrGetErrorMessage.

Remarks

Retrieves from the currently selected SR192 its slot location, logical address, number of installed timing generators and whether timing generators A and B are linked.

9.4.5 ssSrGetErrorMessage

VB	**Function**	*object*.**ssSrGetErrorMessage (** ByVal *errNum* As Long**,** *buf* As String**,** ByVal *bufSize* As Long **) As Long**
C++	**long**	*object*->**ssSrGetErrorMessage (** long *errNum***,** BSTR **buf***,** long *bufSize* **);**

Parameter	**I/O**	**Description**
errNum	in	Error number to evaluate.
buf	out	Output buffer for the error message return.
bufSize	in	The size of the output buffer.

Return Value

0 = Successful operation. Negative values are error codes. To get a description of the error, pass the error code to this function.

Remarks

Fills the specified string buffer with a description of the error represented by *errNum*. The description is truncated to *bufSize*. The SR192 ActiveX driver also maintains a dynamic error description stack. As errors are encountered, their descriptions and context (e.g. "loading sequence") are placed on the stack. These descriptions provide more information about a problem than a simple lookup table of canned error descriptions. To retrieve the top entry of the error stack, use an *errNum* of zero (0). For best results, retrieve all errors from the stack in a code loop. The error stack is empty when a negative one (-1) is returned.

When calling this function from Visual Basic it is recommended to use a fixed length string for *buf.* Otherwise, fill a string to the appropriate size with String$ or Space$.

Error messages are classified into three categories. Errors detected in the SR192 ActiveX Driver (ssSr192X.DLL); errors attributed to the VXI Plug&Play (VPP) driver (tasr192_32.DLL); and errors attributed to the VXI Resource Manager and VISA.

The format of the returned error message conforms to the following convention:

Error: <SSeCode>**:**<VPPeCode> <Error source> **reported** <"VISA/VPP Driver Error Message"> **while** <operational context description when error occurred>.

An example error message is shown below:

Error: BFFC0FFB:BFFF0015 SR192 reported "Timeout expired before operation completed" while reading Timing Set directory.

The VPPeCodes are errors reported by VISA and the VPP Driver. A list of these codes and their descriptions are found in the VPP VISA and VPP driver documentation. The SSeCodes are generated by the SR192 ActiveX Driver and their descriptions follow:

Constant Label	**Error Code**	**Description**
SS_ERROR_PJOPEN	(BFFC0FFF)	Error opening/closing project file.
SS_ERROR_PJACCESS	(BFFC0FFE)	Error reading/writing project file.
SS_ERROR_OSFAIL	(BFFC0FFD)	Operating system error.
SS_ERROR_RMFAIL	(BFFC0FFC)	VISA Resource Manager error.
SS_ERROR_SRFAIL	(BFFC0FFB)	SR192 VPP/VISA error.
SS_ERROR_ESCAPE_TIMEOUT	(BFFC0FFA)	Escape timeout expired.
SS_ERROR_SELECT_FAIL	(BFFC0FF9)	Unable to locate SR192 in VXI chassis.
SS_ERROR_SRSYS	(BFFC0FF8)	SR192 SCPI error.
SS_ERROR_TABLEERR	(BFFC0FF7)	SR192 table assignment error.
SS_ERROR_PROGRAM	(BFFC0FF6)	SR192 application error.
SS_ERROR_VERSION	(BFFC0FF5)	Driver/Firmware version warning.

9.4.6 ssSrGetExecResults

VB	**Function**	*object*.**ssSrGetExecResults** (*tgPage* As String, *tableName* As String, ByVal *failCount* As Long, *faultFile* As String, ByVal *flag* As Long) **As Long**
C++	**long**	*object*-> **ssSrGetExecResults(** BSTR **tgPage***,** BSTR **tabelName*, long *failCount***,** BSTR **faultFile***,** long *flag***);**

<u>**Parameter**</u>	<u>**I/O**</u>	<u>**Description**</u>
tgPage	in	Timing generator/page select. “A1” through “A4” for Timing Generator A pages 1 through 4 and “B1” through “B4” for Timing Generator B pages 1 through 4.
tableName	in	Name of table to scan for test results.
failCount	in	Number of vector failures to report. If *failCount* is zero, do not report any errors in the *faultFile*.
faultFile	in	The file name and path for writing vector/signal failure information.
flag	in	*faultFile* overwrite or append flag. If the flag value is zero (0), this function overwrites any existing data in the *faultfile*. If the flag value is one (1), this function appends the test results to the existing data in the *faultFile*.

Return Value

1 = Test passed, 0 = Test failed. Negative values are error codes. To get a description of the error, pass the error code to ssSrGetErrorMessage.

Remarks

Returns a value for the results of the last test execution for a given timing generator and named table pair. A value of one (1) means the test passed, a value of zero (0) means the test failed. If *failCount* is greater than zero, the individual vector/signal failures are written to the specified *faultFile*. If *failCount* is zero, the failure information is not written; however, pass/fail information is still reported through the return value. Specifying a *failCount* of zero provides a quick pass/fail evaluation.

The fault file is an ASCII file containing detailed failure information about an execution. Previous copies of the fault file are overwritten or appended to depending on the value of the *flag* parameter. If there are no errors to report, an empty fault file is generated. The format of the fault file consists of a failed test vector and signal on each line. The first field is the number of the failed test vector; and the second field is the signal name associated with the failing SR192 channel. Test vectors are unique for each table and are numbered sequentially from one to the table size.

In the following example, failures are detected on SIG1 at test vectors 1, 8, 9, 11 and 25. Failures are detected on SIG2 at test vectors 1, 6 and 11; and on SIG3 at test vector 1, 3 and 11.

```
1  SIG1
 1  SIG2
 1  SIG3
 3  SIG3
 6  SIG2
 8  SIG1
 9  SIG1
 11  SIG1
 11  SIG2
 11  SIG3
 25  SIG1
```

The *failCount* parameter specifies the number of failing vectors to report. If a *failCount* of 3 was specified for the example above, only test vectors 1, 3 and 6 would be reported in the fault file. Note that *failCount* controls the reporting of detected vector errors. This could turn out to be vectors 1, 3 and 6; as in the example above, or vectors 12, 57 and 99.

The *flag* variable is used to control the generation of the fault file data. If *flag* value is zero (0) any existing data in the specified fault file is overwritten with the test results reported by this function. If *flag* value is one (1) the test results are appended to the existing data in the fault file. Append provides a way to combine the test results from multiple tables into a single fault file. This is particularly useful for reporting sequence execution results since sequences typically run multiple tables.

The Test Manager builds an execution result file by intermixing calls to this function and direct file operations. An execution result file is a multi-table fault file augmented by project and table information. The code for doing this is included with the supplied Test Manager source (<Install_Dir>\Examples\TestMan\*.*).

9.4.7 ssSrGetList

VB	**Function**	*object***.ssSrGetList (** *listType* As Long, *tgPage* As String**,** *listBuf* As String**,** ByVal *bufSize* As Long **) As Long**
C++	**long**	*object->***ssSrGetList(** long *listType*, BSTR **tgPage***,** BSTR **listBuf***,** long *bufSize* **);**

Parameter	**I/O**	**Description**
listType	in	Type of list to return: timing set names (0), table names (1) or sequence names (2).
tgPage	in	Timing generator/page select. "A1" through "A4" for Timing Generator A pages 1 through 4 and "B1" through "B4" for Timing Generator B pages 1 through 4.
listBuf	out	Output buffer for comma-delimited list of specified items.
bufSize	in	Size of the output buffer.

Return Value

0 = Successful operation. Negative values are error codes. To get a description of the error, pass the error code to ssSrGetErrorMessage.

Remarks

Fills the specified output buffer with a comma-delimited list of names retrieved from the selected SR192. Truncates the output buffer at *bufSize*.

When calling this function from Visual Basic it is recommended to use a fixed length string for *listBuf.* Otherwise, fill a string to the appropriate size with String$ or Space$.

9.4.8 ssSrGetTestResults

VB	**Function**	*object*.**ssSrGetTestResults (** ByVal *failCount* As Long**,** *faultFile* As String **) As Long**
C++	**long**	*object*->**ssSrGetTestResults (** long *failCount*, BSTR **faultFile* **);**

Parameter	**I/O**	**Description**
failCount	in	Number of test failures to report. If failCount is zero, do not report any errors in the faultFile.
faultFile	in	The file name and path for writing vector/signal failure information.

Return Value

1 = Test passed, 0 = Test failed. Negative values are error codes. To get a description of the error, pass the error code to ssSrGetErrorMessage.

Remarks

Returns a value for the results of the last test execution. A value of 1 means the test passed, a value of 0 means the test failed. If *failCount* is greater than zero, the individual vector/signal failures are written to the specified *faultFile*. If *failCount* is zero, the failure information is not written; however, pass/fail information is still reported through the return value. Specifying a *failCount* of zero provides a quick pass/fail evaluation.

Note: This has been replaced by a faster function (ssSrGetExecResults) that is not as dependent on previous operations. See the section on Execution Result Reporting for more information.

The fault file is an ASCII file containing detailed failure information for the last execution. Previous copies of the fault file are overwritten. If there are no errors to report, an empty fault file is generated. The format of the fault file consists of a failed test vector and signal on each line. The first field is the number of the failed test vector; and the second field is the signal name associated with the failing SR192 channel. Test vectors are unique for each table and are numbered sequentially from one to the table size.

In the following example, failures are detected on SIG1 at test vectors 1, 8, 9, 11 and 25. Failures are detected on SIG2 at test vectors 1, 6 and 11; and on SIG3 at test vector 1, 3 and 11.

```
1  SIG1
1  SIG2
1  SIG3
3  SIG3
6  SIG2
8  SIG1
9  SIG1
11  SIG1
11  SIG2
11  SIG3
25  SIG1
```

The *failCount* parameter specifies the number of **failing** vectors to report. If a *failCount* of 3 was specified for the example above, only test vectors 1, 3 and 6 would be reported in the fault file. Note that *failCount* controls the reporting of detected vector errors. This could turn out to be vectors 1, 3 and 6; as in the example above, or vectors 12, 57 and 99.

9.4.9 ssSrHalt

VB	**Function**	*object*.**ssSrHalt ( ) As Long**
C++	**long**	*object*->**ssSrHalt (** *void* **);**

Parameter	**I/O**	**Description**
(none)		

Return Value

0 = Successful operation. Negative values are error codes. To get a description of the error, pass the error code to ssSrGetErrorMessage.

Remarks

Halts execution of the currently selected SR192. The SR192's contents and settings are preserved. This function can also be used to halt ssSrLoadProject.

9.4.10 ssSrLearnResponse

VB	**Function**	*object*.**ssSrLearnResponse (** *tgPage* as String**,** *timSetName* As String, *tableName* As String **)** As Long
C++	**long**	*object*->**ssSrLearnResponse (** BSTR **tgPage*, BSTR **timSetName*, BSTR **tableName* **);**

Parameter	**I/O**	**Description**
tgPage	in	Timing generator/page select. “A1” through “A4” for Timing Generator A pages 1 through 4 and “B1” through “B4” for Timing Generator B pages 1 through 4.
timSetName	in	Timing set to operate with.
tableName	in	Table to update with learned responses.

Return Value

0 = Successful operation. Negative values are error codes. To get a description of the error, pass the error code to ssSrGetErrorMessage.

Remarks

This function updates the expected states, in the specified table, based on recorded responses to test stimulus. It only updates signals that are monitored for that state (i.e. vector). The stored responses are based on a prior test execution.

The ssSrLearnResponse function provides a way to capture signal responses for monitored channels. Its primary use is to accelerate test development by learning the behavior of a known-good Unit Under Test (UUT). This is accomplished by driving the UUT inputs with test vectors and capturing the UUT responses at its outputs.

The learn process requires the development of stimulus vectors and the assignment of logic levels to response signals on the UUT. Setting response vectors to tristate, neglected or the don’t care state excludes them from being learned. The next step is to run the test vectors against the UUT hardware, followed by an ssSrLearnResponse function call. After verifying repeatability, the test is then saved in a new project file and is subsequently used to test the UUT.

> **Note: It is very important to ensure that the UUT is initialized prior to testing responses. Without initialization, the UUT responses are not predictable or repeatable.**

This entire process can be performed using the SR192 Development System. The SR192 setup and test vectors are defined with the SR192 Development Environment and saved in a project file. The Test Manager then downloads the project file to the SR192, applies the vectors to the UUT and captures the results. The tests can be cycled and tested for errors to establish repeatability. The learned results are then uploaded from the SR192 and saved to a project file. The new, or revised, project file can then be loaded and viewed by the SR192 Development Environment.

Alternately, the functions in the SR192 ActiveX DLL may be used from within a custom application. The necessary sequence of events is to download a project file using ssSrLoadProject, execute it with ssSrExecute, update response states with ssSrLearnResponse, verify test integrity with ssSrGetExecResults and save the modified test data with ssSrBuildProject.

The ssSrLearnResponse function is a post-processing operation that acts on a previous test execution. It is important that a learn be preceded by an ssSrExecute call. Running ssSrLearnResponse twice, without an intermediate test execution, toggles the test data between the initial and learned states. This behavior could be used to restore the initial state of test data.

9.4.11 ssSrLoadProject

VB	**Function**	*object*.**ssSrLoadProject (** *projectFile* As String **) as Long**
C++	**long**	*object*->**ssSrLoadProject (** BSTR **projectFile* **);**

Parameter	**I/O**	**Description**
projectFile	in	SR192 project file name and path.

Return Value

0 = Successful operation. Negative values are error codes. To get a description of the error, pass the error code to ssSrGetErrorMessage.

Remarks

This function loads the contents of the specified SR192 project file into the currently selected SR192. The project file defines which SR192 instruments are affected. Project files are created and viewed with the SR192 Development Environment.

9.4.12 ssSrLoadSequence

VB	**Function**	*object***.ssSrLoadSequence (** *tgPage* as String**,** *seqFile* As String **) as Long**
C++	**long**	*object*->**ssSrLoadSequence (** BSTR **tgPage*, BSTR **seqFile* **);**

Parameter	**I/O**	**Description**
tgPage	in	Timing generator/page select. “A1” through “A4” for Timing Generator A pages 1 through 4 and “B1” through “B4” for Timing Generator B pages 1 through 4.
seqFile	in	SR192 sequence file name and path.

Return Value

0 = Successful operation. Negative values are error codes. To get a description of the error, pass the error code to ssSrGetErrorMessage.

Remarks

This function loads the contents of the specified SR192 sequence file into the currently selected SR192. A sequence file defines executable sequences of timing set and table pairs. Sequences are executed with the ssSrExecuteSequence function. Sequence files are created by the Sequence Editor and they only work with SR101 Timing Set Modules (<u>not</u> SR100).

> **Note: Timing sets and tables must be loaded into the SR192 before loading sequences that reference them. Errors are reported if a sequence references a nonexistent timing set or table.**

This function is also capable of downloading sequences, or other commands, defined by an ASCII SCPI (SCP) file. The file format is line-oriented SCPI commands. For increased readability, blank lines may be used to separate the commands. Comments are preceded by double slashes (//) and they can follow a command or be on a separate line.

> **Note: Existing sequences are not automatically deleted when a SCPI file is downloaded with this function. Consequently, successive loads of the same SCPI sequence file can cause a multiple-defined parameter error. To ensure that existing sequences are deleted, include "**SEQ:DEL:ALL**" at the beginning of a SCPI sequence file.**

9.4.13 ssSrReset

VB	**Function**	*object*.**ssSrReset ( ) As Long**
C++	**long**	*object*->**ssSrReset (** *void* **);**

Parameter	**I/O**	**Description**
(none)		

Return Value

0 = Successful operation. Negative values are error codes. To get a description of the error, pass the error code to ssSrGetErrorMessage.

Remarks

Stops the execution of the currently selected SR192 and resets it to a power-up state. All tables and timing sets are deleted.

9.4.14 ssSrSelectSR192

VB	**Function**	*object*.**ssSrSelectSR192 (** ByVal *modPosition* As Long **) As Long**
C++	**long**	*object*->**ssSrSelectSR192 (** long *modPosition* **);**

Parameter	**I/O**	**Description**
modPosition	in	Relative position to other SR192 modules in VXI chassis.

Return Value

0 = Successful operation. Negative values are error codes. To get a description of the error, pass the error code to ssSrGetErrorMessage.

Remarks

This function selects an SR192 for operation by its relative position in the VXI chassis. Position one (1) is the SR192 instrument closest to slot 0. If the specified module position is invalid, the current selection is unchanged. The default module position is one (1).

The selected SR192 is the instrument that the other functions operate with.

9.4.15 ssSrStatus

VB	**Function**	*object*.**ssSrStatus ( )** As Long
C++	**long**	*object*->**ssSrStatus (** *void* **);**

Parameter	**I/O**	**Description**
(none)		

Return Value

0 = The selected SR192 is ready.
1 = The selected SR192's Timing Generator A is running.
2 = The selected SR192's Timing Generator B is running.
3 = Both Timing Generator A and B are running.

Remarks

Checks the execution status of the currently selected SR192. Use this to watch for the completion of a test following an ssSrExecute command.

9.5 Application Development Environments

There are many application development environments (ADE) that support ActiveX Automation and are able to interface to the SR192 ActiveX Driver DLL (ssSr192X.DLL). Unfortunately each one does so in a slightly different way. The following sections provide general information about interfacing to Visual Basic, Visual C++, HP-VEE, National Instruments LabWindows/CVI and National Instruments LabView. For additional information, refer to the documentation supplied with the specific application development environment. Also, check the examples directory (<Install_Dir>\Examples\) that is installed with the SR192 Development System.

Prerequisites

Before exercising the programming interface, make sure that all of the components of the SR192 Development System, VISA and the Talon SR192 Plug&Play driver (32-bit) are installed and operating properly. A quick check of this is possible by loading and running the example project (HelloSR.SRP) with the Test Manager.

> **Note: Review the section on driver updates to ensure that the latest driver is properly installed and registered.**

9.5.1 Visual Basic

Interfacing to Visual Basic involves declaring the ssSr192X.DLL function prototypes, creating the **ssSr192drv** object and accessing its methods by referencing the instantiated object name. When setting up a Visual Basic application, the ssSr192X DLL must be included as a reference. This is accomplished with the Reference option on the Project menu. A complete Visual Basic example is in the Examples section and the source is included with the SR192 Development System installation.

9.5.2 Visual C++

The easiest way to make the **ssSr192drv** calls accessible to a Visual C++ application (V6) is by using the #import directive with the type library (ssSr192X.tlb). An instance of the **ssSr192drv** object is then created and function calls are made via a pointer to the instantiated object. Sample C++ source code is in the Examples section.

9.5.3 HP-VEE

To access the ssSr192X methods from an HP-VEE application, perform the following steps.

1. From the main menu select **Device>>ActiveX Automation References**. HP-VEE must be in the standard compatibility mode.

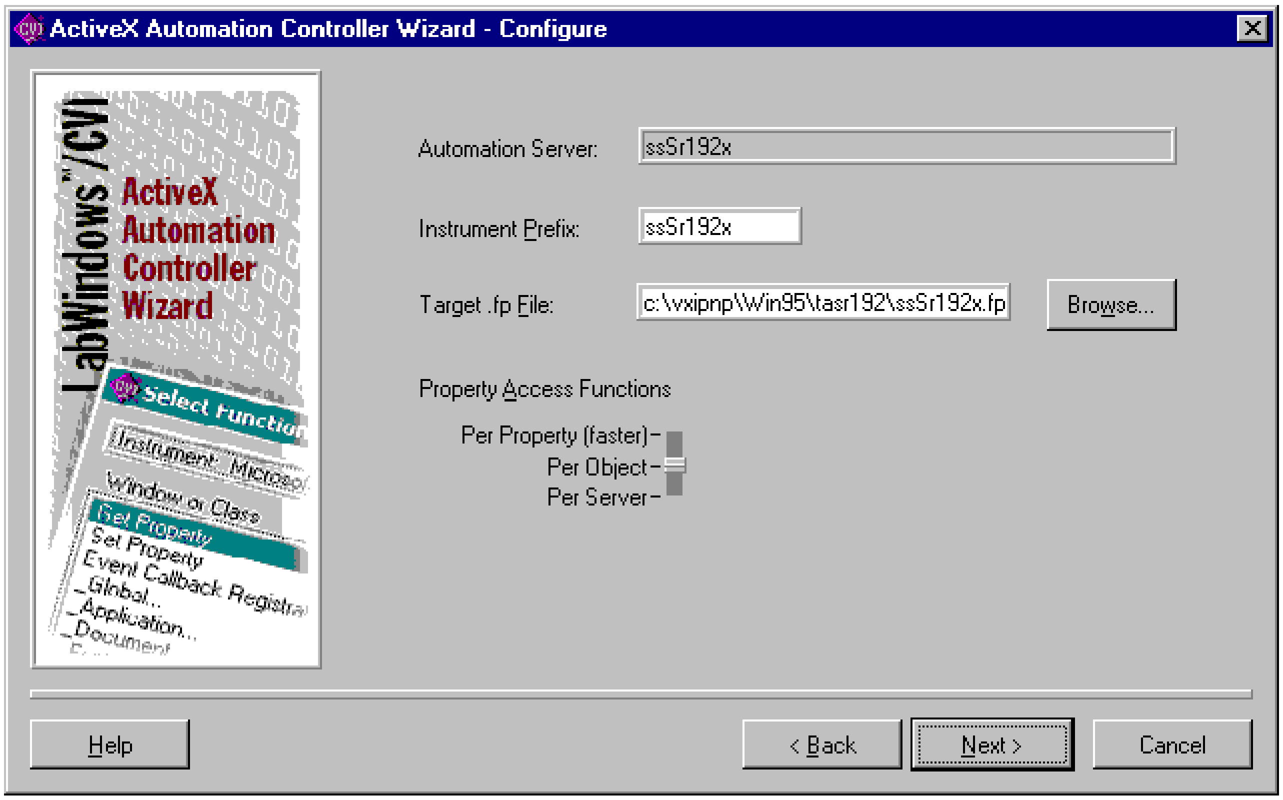
2. Select **References** and select the **ssSr192X.TLB** file.

3. To use an ssSr192X method, perform a **Select>>Function and Object Browser** from the main menu. Pick, place and connect the desired method in the application program.

9.5.4 National Instruments LabWindows/CVI

The LabWindows/CVI software (V5) has a built-in tool to convert the supplied type library file (ssSr192X.TLB) to a Function Panel (FP) file. The conversion tool is accessed from the Tools menu, **Tools>>Create ActiveX Automation Controller**, of LabWindows/CVI. Browse to locate the ssSr192X.TLB file and select Generate to product a function panel file. To use the function panel in an application, select **Instrument>>Load>>ssSr192X.FP** from the main menu.

LabWindows/CVI version 5.5 has additional settings in the **ActiveX Automation Controller Wizard**. For version 5.5, set the "Property Access Functions" selection to the "Per Object" position on the "ActiveX Automation Controller Wizard - Configure" form (see below).


ActiveX Automation Controller Wizard - Configure
LabWindows/CVI
ActiveX Automation Controller Wizard
Select Function
Help
< Back
Next >
Cancel
Browse...

Automation Server: ssSr192x

Instrument Prefix: ssSr192x

Target .fp File: c:\vxipnp\Win95\tasr192\ssSr192x.fp

Property Access Functions

Per Property (faster)
Per Object
Per Server

9.5.5 National Instruments LabView

To access the ssSr192X methods from a LabView application (V5), perform the following steps.

1. Open the LabView diagram window. Make sure that the default palette set is selected. This selection is located in the **Edit>>Select Palette Set>>Default** menu.

2. Open the Functions Palette, if not already open. This selection may be found in the **Window>>Show Functions Palette** menu.

3. From the Functions Palette select **Communication>>ActiveX**. The **ActiveX** palette has five icons. Select the **Automation Open** icon and place it in the diagram window.

4. Right mouse click on the **Automation Open** icon in the diagram window. A pop-up menu will appear. Select the **ActiveX Class>>Browse** entry. Browse, locate and select the type library file (ssSr192X.TLB). This file can be found in the **"c:\Program Files\SR192DevSys"** directory. Once selected, a **"ssSr192drv(ssSr192x.ssSr192drv)"** entry appears in the Objects text pane. Select this entry and click OK. The ActiveX Open icon in the diagram window now has an open handle assignment attached.

5. Next, select an **Invoke Node** icon from the **ActiveX** palette and place it in the diagram window. Open up a pop-up menu by right clicking on the newly placed **Invoke Node** icon. Select the **ActiveX Class>>ssSr192x._ssSr192drv** entry in the pop-up menu. The Automation caption in the **Invoke Node** icon now changes to read **_ssSr192drv**.

6. Make sure that the Finger tool is selected, then left click on the Method field on the **_ssSr192drv Invoke Node** icon. This opens a list of available methods for the ssSr192x ActiveX Automation DLL. Choose a method from this list to transform the **Invoke Node** icon to the selected method with argument fields.

Once the ActiveX Automation icons are placed and specialized, they must be "wired" into the LabView application. This wiring and sequencing of Virtual Instruments (VI) and functions is the standard application development paradigm for LabView.

Keep in mind that all arguments and return values must be appropriately wired to input/output values. Some arguments are passed by reference. These items will have both input and output terminals associated with the entry. Both terminals must be connected for the VI to execute.

9.6 Examples

9.6.1 Visual Basic Example

The following program example shows the minimum code required to load a project file, execute a timing set/table pair and report any test failures. The key program calls are bolded to make them stand out from the comments and error check code. For a more comprehensive application example for a Windows environment, look at the Test Manager source code.

```
'**********************************************************************************************
' modSimpl.bas   v1.1
'   This code module contains a very simple example of interfacing with the
'   ssSr192X driver DLL.  It is intended as a starting point for exploring the
'   driver's functionality.  Consequently, it has no user interface elements or
'   error resolving mechanisms.
'
'   The function prototypes below create a reference between the call
'   and the driver (ssSr192X.DLL).  Alternately, you can include a separate
'   module (ssSr192X.bas) in your project that contains the prototypes.  This
'   file can be found with the Test Manager source code.
'
'   Make sure that the ssSr192drv object is checked in the References window.
'   (The References window selection is on the Project Menu)
'
' Note VB conventions:
'   Comments begin with a single quote;
'   Line continuation is space/underscore/return.
'
'**********************************************************************************************

Option Explicit

' DLL Prototypes in alphabetical order
Declare Function ssSrBuildProject Lib "ssSr192X.dll" (ProjectFile As String) As Long
Declare Function ssSrExecute Lib "ssSr192X.dll" (tgPage As String, _
        TimingSetName As String, TableName As String, ByVal LoopCount As Long) As Long
Declare Function ssSrExecuteSequence Lib "ssSr192X.dll" (tgPage As String, _
        SequenceName As String, ByVal LoopCount As Long) As Long
Declare Function ssSrGetConfig Lib "ssSr192X.dll" _
        (iSlot As Long, iLa As Long, tgCount As Long, tgLinked As Long) As Long
Declare Function ssSrGetErrorMessage Lib "ssSr192X.dll" _
        (ByVal errNum As Long, buffer As String, bufSize As Long) As Long
Declare Function ssSrGetList Lib "ssSr192X.dll" (listSelect As Long, _
        tgPage As String, listBuf As String, ByVal bufSize As Long) As Long
Declare Function ssSrGetTestResults Lib "ssSr192X.dll" _
        (ByVal failCount As Long, faultFile As String) As Long
Declare Function ssSrHalt Lib "ssSr192X.dll" () As Long
Declare Function ssSrLearnResponse Lib "ssSr192X.dll" _
        (tgPage As String, TimingSetName As String, TableName As String) As Long
Declare Function ssSrLoadProject Lib "ssSr192X.dll" _
        (ProjectFile As String) As Long
```

```
Declare Function ssSrLoadSequence Lib "ssSr192X.dll" _
        (tgPage As String, SequenceFile As String) As Long
Declare Function ssSrReset Lib "ssSr192X.dll" () As Long
Declare Function ssSrSelectSR192 Lib "ssSr192X.dll" _
        (ByVal modulePosition As Long) As Long
Declare Function ssSrStatus Lib "ssSr192X.dll" () As Long


'*************************************************************************************
' Main program module.
'  This module loads an SR192 project file, executes a timing set/table pair
'  and checks for errors.  The project file in the example is included with the
'  SR192 Development System and is run using this example code.
'
'  This example was written in VB 5.0.
'    Make sure that the ssSr192drv object is checked in the References
'    window (The References window selection is on the Project Menu).
'*************************************************************************************
'
Sub Main()

Dim status As Long          ' Status variable
Dim drv As ssSr192drv       ' Defines variable "drv" to be a ssSr192drv Object

Set drv = New ssSr192drv        ' Make an instance of SR192 Driver Object

' The following call loads a demo project file into the SR192.  Since no SR192 was
' selected prior to the call, the call automatically defaults to module 1.
status = drv.ssSrLoadProject("c:\program files\sr192DevSys\HelloSR.srp")
If status <> 0 Then
   Debug.Print status         ' Test for error, print error code in debug window
End If

' Start execution.  Use timing generator A, page 1; timing set named "CakeWalk";
' table named "WalkOne" and run for one full cycle.
status = drv.ssSrExecute("A1", "CakeWalk", "WalkOne", 1)
If status <> 0 Then
   Debug.Print status         ' Test for error, print error code in debug window
End If

' Test for error. Report the first 10 errors only and store
' them in a file named "fault.res"
status = drv.ssSrGetExecResults("A1", "WalkOne", 10, "fault.res", 0)
If status = 1 Then Debug.Print "Pass"     ' Print "Pass" in debug window
If status = 0 Then Debug.Print "Fail"     ' Print "Fail" in debug window

Set drv = Nothing             ' Destroy instance of SR Driver Object

End Sub
```

9.6.2 Visual C++ Example

The following source sample shows the minimum code required to load a project file, execute a timing set/table pair and report any test failures. The key program calls are bolded to make them stand out from the comments and error check code. Note that a project file and additional support files are required before this will execute in the Visual C++ (V6) development environment. For a more comprehensive example of a Windows application, look at the Test Manager Visual Basic source code.

```
// MSCPPV6.cpp : Defines the entry point for the console application.
//
//*******************************************************************************
//
//   This code module contains a very simple example of interfacing with the
//   ssSr192X driver DLL.  It is intended as a starting point for exploring the
//   driver's functionality.  Consequently, it has no user interface elements and
//   limited error resolving mechanisms.
//
//   The ssSr192x function references are made accessible by importing a type
//   library (ssSr192X.TLB).
//
//   This code was developed as a console application using Microsoft C++ Version 6.0
//
// Note C++ conventions:
//   Comments begin with a double right slash (//).
//
//********************************************************************************

// This header is automatically generated by console application wizard.
#include "stdafx.h"

// Program includes.
#include <iostream>
#include <string>

// Import ssSr192X.DLL interface type library.
#import "c:\Program Files\Sr192DevSys\ssSr192X.tlb"

// Declare ssSr192X.dll namespace.
using namespace ssSr192x;
using std::cout;
using std::cin;

int main(int argc, char* argv[])
{
    // Local variables
    #define BSIZE 255;
    long status;
    BSTR wStringBuf;
    long bufSize;
    short ccs;
    int anInt;
    long asciBufSize = BSIZE;
    char asciBuf[255];

    // Initialize variables
    bufSize = BSIZE;
    wStringBuf = SysAllocStringLen(NULL,bufSize);

    // First, initialize the COM library; if error, report it and exit.
    if (FAILED(CoInitialize(NULL)))
    {
        cout << "COM Library initialization failed" << "\n";
        return 0;
    }
```

```
//*******************************************************************************
// Main program module.
//  This module loads an SR192 project file, executes a timing set/table pair and
//  checks for errors.  The project file for this example was created by the
//  Visual C++ console application wizard.
//
//  This example was written in Microsoft C++ V6 as a console application. .
//
//*******************************************************************************

    cout << "GO -->>>\n";
    // The try/catch isolates unexpected COM errors from ssSr192x.DLL errors
    try
    {
        // The following makes an instance of the ssSr192drv object and
        // defines variable "pSR" to be a ssSr192drv Object pointer.
        _ssSr192drvPtr pSR(__uuidof(ssSr192drv));

        // Make a UNICODE string containing project file name.
        BSTR fileName;
        fileName = SysAllocString(L"c:\\program files\\sr192DevSys\\HelloSR.srp");

        // The following call loads a demo project file into the SR192.  Since
        // no SR192 was selected prior to the call, the call automatically
        // defaults to module 1.
        status = pSR->ssSrLoadProject(&fileName);

        // Test for load project error. If error, get error message and
        // output to console window.
        if ( status < 0 )
        {
            status = pSR->ssSrGetErrorMessage(status,&wStringBuf,bufSize);
            ccs = SysStringLen(wStringBuf);   // Get string length
            // Convert UNICODE to string
            anInt = WideCharToMultiByte(CP_ACP,0,wStringBuf,ccs,
                                  asciBuf,asciBufSize,NULL,NULL);
            asciBuf[ccs] = NULL;       // Null terminate it
            cout << "Error loading project: " << asciBuf << "\n";
        }

        // Start execution.  Use timing generator A, page 1; timing set "CakeWalk";
        // table named "WalkOne" and run for one full cycle.

        // First, make UNICODE strings of variables.
        BSTR wtgp = SysAllocString(L"A1");
        BSTR wts = SysAllocString(L"CakeWalk");
        BSTR wtable = SysAllocString(L"WalkOne");

        // Run SR
        status = pSR->ssSrExecute(&wtgp, &wts, &wtable, 1);
        if ( status < 0 )
        {
            status = pSR->ssSrGetErrorMessage(status,&wStringBuf,bufSize);
            ccs = SysStringLen(wStringBuf);   // Get string length
            // Convert UNICODE to string.
            anInt = WideCharToMultiByte(CP_ACP,0,wStringBuf,ccs,
                                  asciBuf,asciBufSize,NULL,NULL);
            asciBuf[ccs] = NULL;      // Null terminate it
            cout << "Error SR Execute: " << asciBuf << "\n";
        }
        // Test for error.  Report the first 10 errors only
        // and store them in "fault.res" file.
        BSTR faultFileName = SysAllocString(L"fault.res");
        status = pSR->ssSrGetExecResults(&wtgp, &wtable, 10, &faultFileName, 0);
        if ( status < 0 )
        {
            status = pSR->ssSrGetErrorMessage(status,&wStringBuf,bufSize);
            ccs = SysStringLen(wStringBuf);   // Get string length
            // Convert UNICODE to string.
            anInt = WideCharToMultiByte(CP_ACP,0,wStringBuf,ccs,
                                         asciBuf,asciBufSize,NULL,NULL);
            asciBuf[ccs] = NULL;       // Null terminate it
            cout << "Error GetTestResults: " << asciBuf << "\n";
        }
    }

    // Something went wrong with COM error handler
    catch(const _com_error& Err)
    {
        cout << "COM Error: "<< Err.ErrorMessage() << "\n";
    }

    // End of main body stuff

    // Destroy instance of SR Driver Object
    CoUninitialize();
    cout << "Done!\n";

    return 0;
}
```

10. Appendix A - Version Changes

The following are lists of changes that have occurred from previous versions of the SR192 Development System.

Changes from Version 2.20 to Version 3.0

SR192 Development Environment

1. SR Modules, tables and timing sets can now be copied and pasted within, or between, project files. This also allows SR Modules to be reordered in the project list.

2. Execution errors detected by the Test Manager are now automatically highlighted in the SR192 Development Environment. Errors are highlighted with a colored background on any open, and relevant, tables. Commands are provided for error masking, clearing and navigation.

3. A validation process is now performed when a project is saved. The project is automatically checked for missing modules and unmatched signal names.

4. Project changes are now tracked more accurately and the user is only prompted if actual changes have occurred. A compacting process has been added during a project save. This reduces the size of the project file and improves its overall access.

5. A print option has been added for most windows. This captures an image of the window and sends it to the designated printer. This option is available from the File Menu or via shortcut key (Ctrl-P). Use the shortcut key for windows without a menu bar.

6. The Table Editor now supports Value Change Dump (VCD) import and export. VCD is part of the IEEE 1364-1995 specification. During VCD import, warnings are written to the Debug Log window. Double-clicking a warning on the Debug Log window causes the corresponding signal to be highlighted (if the table is currently displayed).

7. Improvements have been made to the Table Editor for handling large tables. Any lengthy process (e.g. load, save, fill) now includes a progress bar and cancel button. Modifications are now tracked more accurately and a table is only written to disk if an editing change has occurred. To close a table without writing modifications to disk, use the “Discard Table Changes” option on the Table Editor’s File Menu. Canceling a save leaves the original table intact.

8. An automatic tooltip has been added to the column headings of the Table Editor. The tooltip shows the vector number for vectors greater than 9999.

9. New tables are now assigned default signal names (Sig1-Sig32).

10. Pop-up menus have been added to the Signal List and Sequence Editor in response to a right mouse click.

11. Fixed a bug whereas voltage groups were not being stored for SR Modules greater than 1. Also fixed a bug that caused identically named timing sets, on different pages, to be removed when one was deleted by the user.

12. The Table Editor now displays signal direction color more accurately during editing.

Test Manager

1. An “Auto Save/Reload” option has been added to the Test Manager. This permits the Test Manager to automatically determine if the current project or sequence file should be saved and/or reloaded. When this option is inactive, the operator is prompted to save or reload when necessary.

2. The Test Manager can now report errors following a sequence execution. To accomplish this, selected tables are each queried for error conditions. The results of these queries are displayed in the Transcript window and written to a fault result file (fault.res). The format of the fault result file has changed to include SR192 module and table information.

3. The number of errors reported following an execution can now be set for the Test Manager. The “Fail Count” is accessed from the Execution Results window.

4. The Test Manager now supports file history listings for project files (*.srp) and sequence files (*.seq). It has also added a Help menu.

Sequence Editor

1. The Sequence Editor has been modified to save large files more quickly.

2. Two print options have been added for the Sequence Editor window. One option captures an image of the entire window and sends it to the designated printer. The other option prints the visible portion of the Sequence List. These options are available from the File Menu or via shortcut key (Ctrl-P).

SR192 ActiveX Driver (ssSr192X)

1. A modification to the SR192 ActiveX driver has been made to better handle the loading of large tables. This corrects errors that occurred when loading large tables on faster computers (>500 MHz).

2. A new function (ssSrGetExecResults) has been added to the SR192 programming interface. This allows more specific error reporting following sequence executions. It is recommended that this new function should be used instead of ssSrGetTestResults in future applications.

3. Various other enhancements and corrections have been made to the SR192 ActiveX driver.

<u>Changes from Version 2.0 to Version 2.20</u>

1. Three voltage groups are now supported for the SR210 DAC module. Voltage Input Slew Rate (VISR) has also been added to all voltage group settings.

2. The Project Browser now supports deleting timing module B.

3. Several improvements have been made to labeling and display updates. The Timing Set Editor now correctly displays programmable clock values.

4. The Pin Group Properties for an SR214 pin module have been updated to include voltage group mapping.

5. The Test Manager has been modified to display multiple error messages and to allow loop counts up to 9999.

6. The SR192 ActiveX driver has been changed to improve the speed of a Learn Response operation. NOTE: You must set Sense signals to either a high or low state in order for them to be learned correctly. Tristate sense lines are neglected after a Learn Response operation.

7. The SR192 ActiveX driver function, ssSrGetErrorMessage, has been changed to allow retrieval of multiple error messages. Use an error number of zero to request the earliest error from the internal error queue. A value of -1 is returned when the queue is empty. The error queue is reset during primary driver calls (e.g. ssSrReset, ssSrLoadProject, ssSrExecute). The error descriptions now include a delimited error code (e.g. <BFFC0FFB:BFFF0015>).

8. The Program Interface sample code has been expanded to include examples for HP VEE, National Instruments' LabView and Microsoft Visual C++. These are found under the "Examples" subdirectory.

9. The operation of the SR192 ActiveX driver has been updated to handle additional hardware configurations and operations.

Changes from Version 1.0 to Version 2.0

1. The Sequence Editor is now integrated into the SR192 Development System. This includes support for sequences in the Test Manager and the SR192 ActiveX driver.

2. The function set of the SR192 ActiveX driver has been significantly changed. Several functions were combined into one (ssSrGetList) and functions were added to support sequences. Additionally, the parameter data types for some functions were changed to simplify interfacing with a variety of development environments.

3. A selection of block fill operations have been added to the Table Editor.

4. The Table Editor now supports the import and export of a simple hexadecimal formatted ASCII file. This allows digital data to be created or modified in hexadecimal.

5. The Timing Set Editor has been modified to support signal name editing and to reflect delays in the timing set frequency display. Also, the TSENABLE signals now display beginning and ending edges.

6. The documentation has been enhanced to include information on interfacing the SR192 ActiveX driver to a variety of development environments.

7. The SR Module Configuration has been revised to correct certain references, labels and options.

8. A mechanism has been added to the SR192 ActiveX driver that allows a bidirectional channel to have different drive and sense levels during the same cell. This is achieved by creating two signals with the same name in the Table Editor. One signal is set to Drive and the other is set to Sense. This doubled signal can then be set with different drive and sense levels simultaneously.

9. The operation of the SR192 ActiveX driver has been updated to handle additional hardware configurations and operations.

11. Sales & Support

For more information contact:

Talon Instruments

150 East Arrow Highway
San Dimas, CA 91773

909-599-0690
909-599-6529 Fax

sales@taloninst.com

Index

/

// 8, 35

A

Appendix A - Version Changes 115
Application Development Environments 105
Assert & Return Wizard 34
Auto Save/Reload 70

C

Cell Wizard 33
check project 8
Context-Sensitive Help 6
copy tables 11

D

DAC 20
Debug Log 15
Development Environment 5
Documentation and Printing 6
doubled signal 35
Driver Updates 82

E

Error Display 37
Error Log File 84
Error Reporting 81
ErrorColor 37
errors 15
Errors Menu 47
Execution Result File Format 77
Execution Results 76
Export Value Change Dump 49

F

Fail Count 37, 76, 77
file formats 15, 52, 64, 84
fill functions 46

H

help 6
Help Menu 13
hexadecimal 52
HP-VEE 79, 106

I

Import & Export Menu 40
Import Value Change Dump 48
instrument 6
Introduction 1

J

Jump Enables 62

L

LabView 79, 108
LabWindows 79, 107
log file......15, 84

M

Modules 18

N

National Instruments LabView 108
National Instruments LabWindows/CVI 107
New Timing Module Page......14

O

Operation 68

P

Parameter Pane......56
pin group 21, 22
Pin Group Control 25
Pin Group Properties 26
power failure 10
printing 6
Programming Interface 79
Project Browser......7
  Edit Menu 11
  File Menu......9
  Help Menu 13
  Options Menu 12
  Toolbar 8
project file......9, 10
Project Validation......8

R

recover......10
register......82
regsvr32......82
reload......70
Remove Name......63

S

Sales & Support......119
save/reload 70
SCP......61, 71, 101
SCPI file......61, 71, 101
SEQ 57, 64
sequence 53, 54, 88, 94, 101
Sequence Editor 53
  Edit Menu 59
  File Menu......57
  Options Menu......61
Sequence Execution......69

Sequence File Format....64
Sequence List....55
Sequence Operation....54
Signal List Editor....21
  Edit Menu....24
  File Menu....23
  Toolbar....22
Simple Hex File Format....52
SR Module Configuration....17
SR100....54
SR101....53, 54
SR192....6
SR192 Development Environment....5
SR192 Hardware....6
SR192 Menu....73
SR192DEV.LOG....15
SRP....9
ssSr192X....79, 83, 105
ssSrBuildProject....86
ssSrExecute....87
ssSrExecuteSequence....88
ssSrGetConfig....89
ssSrGetErrorMessage....90
ssSrGetExecResults....92
ssSrGetList....94
ssSrGetTableNames....94
ssSrGetTestResults....95
ssSrGetTimingSetNames....94
ssSrHalt....97
ssSrLearnResponse....98, 99
ssSrLoadProject....100
ssSrLoadSequence....101
ssSrReset....102
ssSrSelectSR192....103
ssSrStatus....104
Support....119
System Architecture....3
System Operation....4

T

Table Editor....35
  Edit Menu....41
  File Menu....39
  Fill Menu....46
  Options Menu....45
  Set Menu....43
  Toolbar....38
table list....63, 76
table names....11, 54, 87
Talon Instruments....119
Test Manager....67
  File Menu....71
  Options Menu....75
  SR192 Menu....73
  Timing Menu....74
Timing Generators....19

Timing Menu 74
Timing Set Editor 27
  Edit Menu 31
  File Menu..30
  Set Menu 32
  Toolbar 29
timing set list 54, 63, 87
truncate name..11, 54, 87, 88

U

unknown signal state 43

V

validate project..8
validate sequences 61
Value Change Dump File Format 50
VCD 48, 49, 50
Visual Basic 106
Visual Basic Example 109
Visual C++ 106
Visual C++ Example 111

W

warnings..15
working project file 9