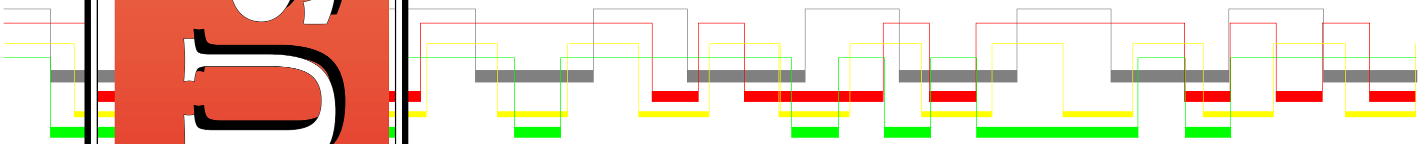
User's Manual

SR2500

VXI Digital Test Subsystem



interface
TECHNOLOGY

SR2500 VXI Digital Test Subsystem

CONTENTS

SR2500 System Overview and Programmer's Guide 1

SR2510 Main Module 2

SR2520 Expansion Module 3

RG2500 Rail Generator 4

SR2520 w/Guided Probe Option 5

Appendices 6

Application & Technical Notes 7

WaveEdit Digital Waveform Editor 8

Interface Technology, Inc.
300 S. Lemon Creek Drive, Ste. A
Walnut, CA 91789

***The Performance Leader
in VXI Digital Testing ...***

interface
TECHNOLOGY

User's Manual

SR2500

VXI Digital Test Subsystem

Includes Coverage of:

- SR2500 Subsystem
- SR2510 Main Module
- SR2520 Expansion Module
- SR2520 w/Guided Probe Option
- RG2500 Rail Generator
- WaveEdit Digital Waveform Editor

interface
TECHNOLOGY

Rev. 05 Jun. '98

Proprietary Notice

This document, and the technical information contained herein, are proprietary of Interface Technology and shall not, without the express written permission of Interface Technology, be used in any form or part to solicit competitive quotations. The information provided herein may be used for operational purposes only, or for the purpose of incorporation into technical specifications or other documents which specify procurement from Interface Technology.

DISCLAIMERS

Interface Technology, Inc. makes no warranty of any kind with regard to this material, including, but not limited to, implied warranties or fitness for a particular use or purpose.

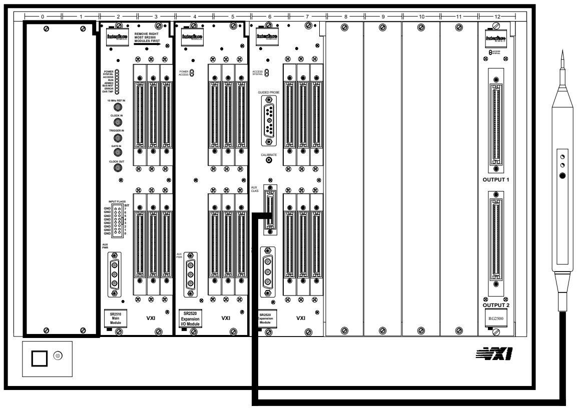
Interface Technology, Inc. shall not be liable for errors contained herein or for incidental or consequential damages in connection with the performance or use of this material.

Interface Technology, Inc. reserves the right to make changes to its products and to the content of this manual without notice.

User's Manual

SR2500 System Overview and Programmer's Guide

From The Performance Leader In VXI Digital Testing ...



interface TECHNOLOGY

Rev. 05 Apr 1998
Change 2 Oct 2000

Contents

Chapter 1
General Description

About This Manual 1-1
Arrangement of Manual 1-1
Arrangement of Contents, This Manual 1-1
Applicability 1-1
Supersedure Notice 1-1
Equipment Description 1-1
SR2510 Main Module 1-2
SR2520 Expansion Module 1-3
SR2520 w/Guided Probe (option) 1-4
RG2500 Rail Generator (option) 1-4

Chapter 2
Test Programming Parameters

Test Programs 2-1
Fields 2-1
Non-Algorithmic Fields 2-2
Algorithmic Fields 2-2
Hardware Fields 2-3
Vector Looping 2-3
Program Branching 2-3
GOTO 2-3
GOSUB 2-4
RETURN 2-4
Data Rates 2-4
VME A32 Dual Access Memory 2-4
Output Data Formatiing 2-4
NRZ (Non-Return-to-Zero) 2-5
RZ (Return-to-Zero) 2-5
RONE (Return-to-One) 2-5
RC (Return-to-Compliment) 2-5
RI (Return-to-Inhibit/Tristate) 2-6
Input Data Formatting 2-6
Edge Mode 2-6
Window Mode 2-6
Memory 2-7
Control Memory 2-7
Stimulus Memory 2-7
Output Memory 2-7
Tristate Memory 2-7
Stimulus Algorithmic Memory
Response Memory 2-8
Record Memory 2-9
Record Memory and Signature Analysis Control 2-9
Programming Languages 2-10

Contents (continued)

Chapter 3
Programming

SCPI Command Syntax 3-1
Basic Programming 3-3
Defining Tests 3-4
Test Definition 3-6
Test Deletion 3-7
Test Definition Catalog 3-8
Test Definition Status Query 3-9
Free Vector Space Query 3-10
Global Test Parameters 3-12
Selecting the Active Test 3-14
Setting the Program Loop Count 3-15
Setting the Test System Frequency 3-16
Setting the Test System Period 3-17
Setting the System Clock Frequency 3-18
Selecting the System Clock Source 3-20
Selecting the Slope of the External Clock 3-21
Setting the External Clock Threshold Level 3-22
Selecting the System Gate Source 3-23
Setting the External Gate Threshold Level 3-24
Selecting the Polarity of the External Gate 3-25
Selecting the Reference Oscillator Source 3-26
Selecting the System Trigger Source 3-27
Selecting the Slope of the External Trigger 3-28
Setting the External Trigger Threshold Level 3-29
Field Definitions 3-30
Field Definition & Pin Assignment 3-32
Field Deletion 3-38
Selecting the Field Radix 3-39
Field Definition Catalog 3-40
Loading and Querying Test Vectors 3-42
Selecting the Default Stimulus Field 3-45
Loading/Querying Stimulus Patterns 3-46
Clearing Stimulus Patterns 3-49
Enabling the Armdata Function 3-51
Setting the Armdata Pattern 3-52
Loading the Stimulus Macro Command Memory 3-54

COMMAND DEFINITIONS 3-58

SProgram[(OUT)] 3-58
EProgram[(OUT)] 3-58
OUTput[(OUT)] 3-58
WLoopuntil([OUT](<loop_cond>)) 3-58

Contents (continued)

SLoopuntil([OUT](<loop_cond>)) 3-59
ELoop[(OUT)] 3-59
SCONDition([OUT](<jump_cond>)) 3-60
SJMPPage(Supported Only on 256K and 1M Vector Cards) 3-60
JMP([OUT](<label_name>)) 3-60
JSRoutine([OUT](<label_name>)) 3-60
RTSubroutine[(OUT] 3-62
CJMP([OUT](<label_name>)) 3-62
CJSRoutine([OUT](<label_name>)) 3-62
CRTSubroutine[(OUT)] 3-63
CLEARError[(OUT)] 3-64

CONDITION DEFINITIONS 3-64

COUNt == count_value 3-64
RCOMpare == TRUE 3-65
RCOMpare != TRUE 3-65
LATCherror == TRUE 3-65
LATCherror != TRUE 3-65
STRigger == TRUE 3-65
FRONtpanel && match_pattern 3-65
FRONtpanel &!match_pattern 3-66
QUALifword && qual_combination 3-66
QUALifword &! qual_combination 3-66
Redefining Macro Command Label Vectors 3-68
Deleting Macro Command Labels 3-69
Copying Stimulus Macro Commands 3-70
Selecting the Default Record Field 3-72
Loading/Querying Record Patterns 3-73
Clearing Record Patterns 3-76

Trace TMACRO's 3-78

Post Trigger TMACRO Definition 3-80
Sequencing TMACRO Definition 3-84

Run Time Commands 3-87

Setting the Arm Counter 3-88
Initializing the Test Program 3-89
Software Trigger Command 3-90
Test Program Abort Command 3-91

Contents (continued)

Reading Recorded Data 3-92

Reading Recorded Patterns.. 3-94
Compare Error Status Query 3-96
Searching Record Memory 3-97

Advanced Programming 3-101

Pattern Editing 3-102

Copy Stimulus Patterns 3-105
Filling Stimulus Memory 3-108
Searching Stimulus Memory 3-112
Copying Record and Response Patterns 3-115
Filling Response Memory 3-118
Searching Response Memory 3-122
I/O Formatting and Timing 3-126
Stimulus Format and Timing 3-128
Clearing Stimulus Format and Timing 3-131
Stimulus Format and Timing Catalog... 3-133
Record Sample Mode and Timing 3-135
Clearing Record Sample Mode and Timing 3-138
Expected Compare Offset 3-140
Record Sample Mode and Timing Catalog 3-141

Algorithmic Pattern Generation 3-142

Stimulus Algorithmic Pattern Generation 3-145
Algorithmic Output Command Definitions 3-147
- NONAlgorithmic 3-147
- INCrement.. 3-147
- DECrement... 3-147
- XOR 3-147
- HOLDData 3-147
- HOLDAll 3-147
- SLEFTZero 3-148
- SLEFTOne 3-148
- SLEFTComplement 3-148
- RLEFT 3-148
- SRIGHTZero.. 3-148
- SRIGHTOne... 3-149
- SRIGHTComplement... 3-149
- RRIGHT... 3-149

Contents (continued)

LOADProgram 3-149
OUTPUTParm 3-149
Clearing Stimulus Algorithmic Memory 3-151
Copying Stimulus Algorithmic Commands 3-153
Filling Stimulus Algorithmic Memory 3-155

Response Algorithmic Pattern Generation 3-157

Algorithmic Expect Command Definitions 3-159
NONAlgorithmic 3-159
INCrement 3-159
DECrement 3-159
XOR 3-159
HOLDData 3-159
HOLDAll 3-159
SLEFTZero 3-160
SLEFTOne 3-160
SLEFTComplement 3-160
RLEFT 3-160
SRIGHTZero 3-160
SRIGHTOne 3-161
SRIGHTComplement 3-161
RRIGHT 3-161
LOADParam 3-161
OUTPUTParm 3-161
Clearing Response Algorithmic Memory 3-163
Copying Response Algorithmic Commands 3-165
Filling Response Algorithmic Memory 3-167

High Speed Binary Pattery Transfers 3-170

Stimulus Mapped Binary Patterns 3-172
Stimulus Non-Mapped Binary Patterns 3-174
Record Mapped Binary Patterns 3-176
Record Non-Mapped Binary Patterns 3-178

Saving and Loading Tests 3-180

Binary Learn Time-Out 3-182
Learning Binary Tests 3-183

Advanced Record Triggering 3-184

Contents (continued)

Record Memory Wrapping 3-187
Qualifier Trigger Patterns 3-188
Clearning Qualifier Trigger Patterns 3-190
Qualifier Trigger Combinations 3-191
Clearing Qualifier Trigger Combinations 3-192
Record Filter and Control 3-193
Advancing Trace Sequences 3-195
Jumping to Trace Sequences 3-197
CRC Calculation Control 3-199
Stopping Tests from Trace Sequences 3-201
Clearing Trace Sequences 3-202
Trace Sequences Catalog 3-203
Querying CRC Checksums 3-205

Miscellaneous Commands 3-207

Diagnostics 3-208
Diagnostic Test Execution 3-210
Diagnostic Test Status Query 3-212

Status Queries, Status Interrupts and System Queries 3-214

Operation Interrupt Definition 3-216
Operation Condition Query 3-218
Questionable Status Registers 3-219
Status Preset 3-222
System Error Query 3-223
SCPI Version Query 3-224
System IDN Query 3-225

Bus Master Time-Out 3-227

Bus Master Time-Out 3-228

Variable Voltage I/O on the SR2500 3-229

Selecting Response Threshold voltage Sets 3-231
Selecting Stimulus Output Voltage Sets 3-232

IEEE 488.2 Commands 3-233

Contents (continued)

IEEE 488.2 Mandatory Commands 3-236

Programming Examples 4-1

Chapter 4
Programming Examples

Program Steps 4-1
RAM-Backed Pattern Generation 4-7
Using CMACROS and Data Formatting 4-10
Generating Algorithmic Stimulus Patterns 4-13
Using Real-Time Compare and Algorithmic Expcted Responses 4-19
Recording UUT Responses 4-26

(THIS PAGE INTENTIONALLY LEFT BLANK)

CHAPTER 1

General Information

About This Manual

This manual provides installation and operation information for the Interface Technology SR2500 VXI Digital Test Subsystem Information contained herein is intended for use by technical personnel involved in the actual installation and operation of the subject equipment.

Arrangement of Manual

This document is comprised of five separate manuals as follows:

- SR2500 User's Manual (overall system manual)
- SR2510 User's Manual
- SR2520 User's Manual
- SR2520 Guided Probe Option User's Manual (option)
- RG2500 Rail Generator User's Manual (option)

Arrangement of Contents, This Manual

Information contained in this manual is arranged in four chapters, as follows:

- Chapter 1 General Information
- Chapter 2 Test Programming Parameters
- Chapter 3 Programming
- Chapter 4 Programming Examples

Applicability

The information contained in this manual covers a single equipment configuration designated ***SR2500 VXI Digital Test Subsystem***. Differences, if any, between this equipment and the actual equipment supplied are covered by Difference Data included at the front of this manual.

Supersedure Notice

This manual supersedes SR2500 User's Manual, Rev.04 and all previous issues of this publication.

Equipment Description

The SR2500 Digital stimulus/response Subsystem provides DC to 25 MHz digital logic patterns for serial and parallel testing of digital semiconductor devices, ASICs, components, circuit boards, assemblies, and other digital devices including complete digital systems. Based on the industry standard VXI architecture, the SR2500 digital subsystem is comprised of one or more dual-slot C-size modules as shown in Figure 1-1.

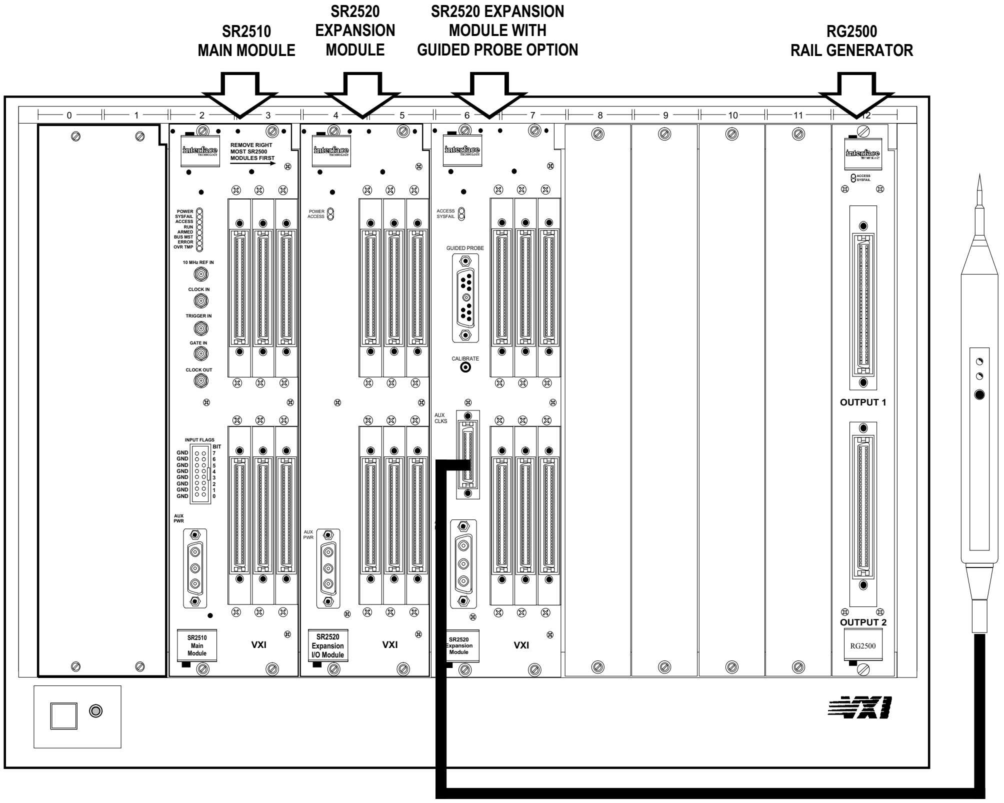


Figure 1-1. SR2500 VXI Digital Test Subsystem.

The major components of the SR2500 VXI Digital Test Subsystem are the SR2510 Main Module and one or more optional modules used to enhance or expand the subsystem. The optional modules include the SR2520 Expansion Module, the RG2500 Rail Generator, and a Guided Probe option for the SR2520 Expansion Module.

SR2510 Main Module

The message-based SR2510 provides clocking and test sequence control functions for all I/O channels within the SR2510 module, and for all SR2520 Expansion Modules as well. The SR2510 consists of a Timing/ Control board, up to three (3) I/O boards, up to six (6) Driver/Receiver boards (two per I/O board) and boards for timing distribution, power distribution and interface logic for any SR2520 expansion modules, see Figures 1-2 and 1-3. The Timing/Control board contains a 25 MHz 68EC030 microprocessor (system processor) that provides the basic user interface to the SR2500 system. The 68EC030 parses and interprets the VXI word-serial commands and provides overall system setup and test monitoring. The SR2510 also contains a custom control processor ASIC that provides real-time control over the test pattern sequencing. The control processor is capable of providing sequential or nested program looping and conditional or unconditional jumps and subroutines. Overall test timing is provided by a programmable 200 Hz to 25 MHz frequency synthesized clock source as well as external inputs for clocks, gates, test inputs and triggers. For additional details, refer to the *SR2510 User's manual.*

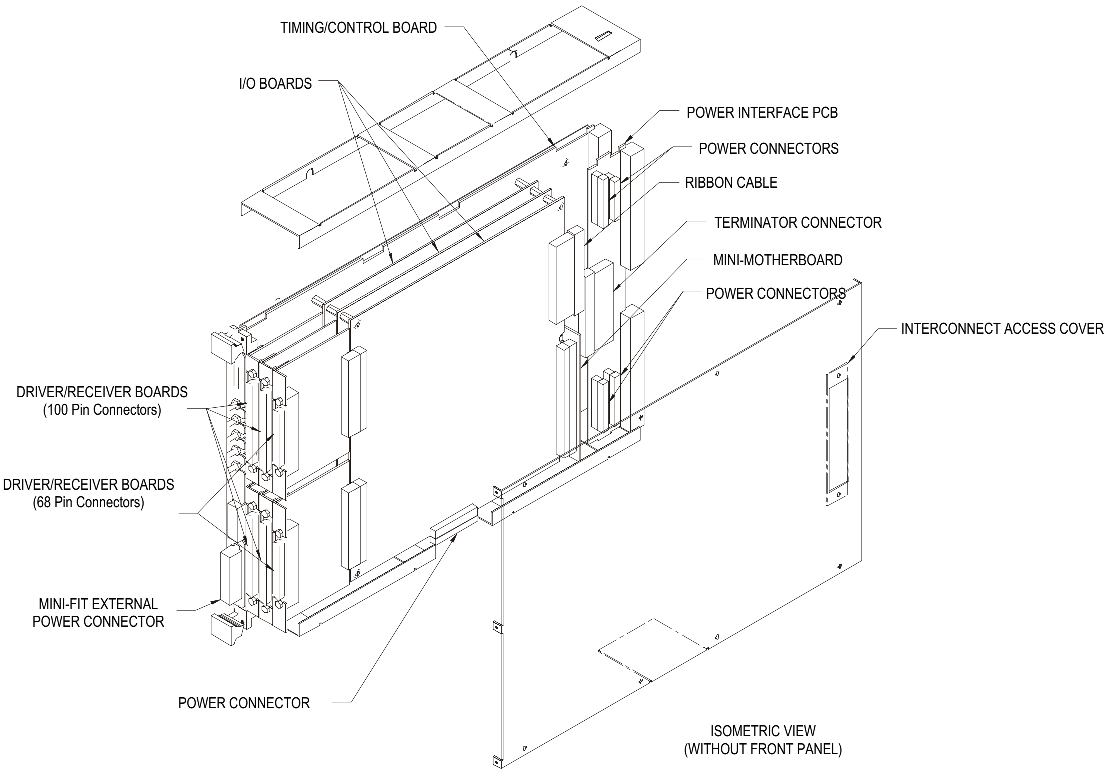


Figure 1-2. SR2510 Main Module, Exploded View Showing Major Components.

SR2520 Expansion Module

(See Fig 1-1) The SR2520 expansion module is a register-based companion to the SR2510 module. Each SR2520 provides an additional 96 I/O channels, and up to five SR2520s (up to 576 channels) may be included in a single SR2500 subsystem. Each SR2520 module has 96 output pins and 96 input pins, except in the case of the Variable Voltage configuration, which has 96 bi-directional I/O pins. By connecting the output and input pins together, 96 I/O channels can be realized. Each I/O channel generates digital stimulus patterns, provides real-time comparison capabilities on the response inputs, and contains logic analyzer type triggering and data recording functions, all at speeds up to 25 MHz.

Except in the case of the Differential ECL configuration, which has no tristate memories, each stimulus pin contains output and tristate memories, allowing bi-directional signal paths. The response pin provides *expected response* and *mask* ("don't care") memories, which generate the expected input pattern used for the real-time comparison. The logic analyzer triggering and recording subsystem allows the recording of either the actual input pattern or the results of the real-time comparison of the expected response pattern and the input pattern (error data). Either may be saved and then later retrieved from the record memory, in much the same way you would use a logic analyzer.

The SR2500 subsystem is designed to operate with any VXI compatible slot-0 controller that supports the word serial protocol. The command set that controls test setup and execution is based on the SCPI-syntax command set.

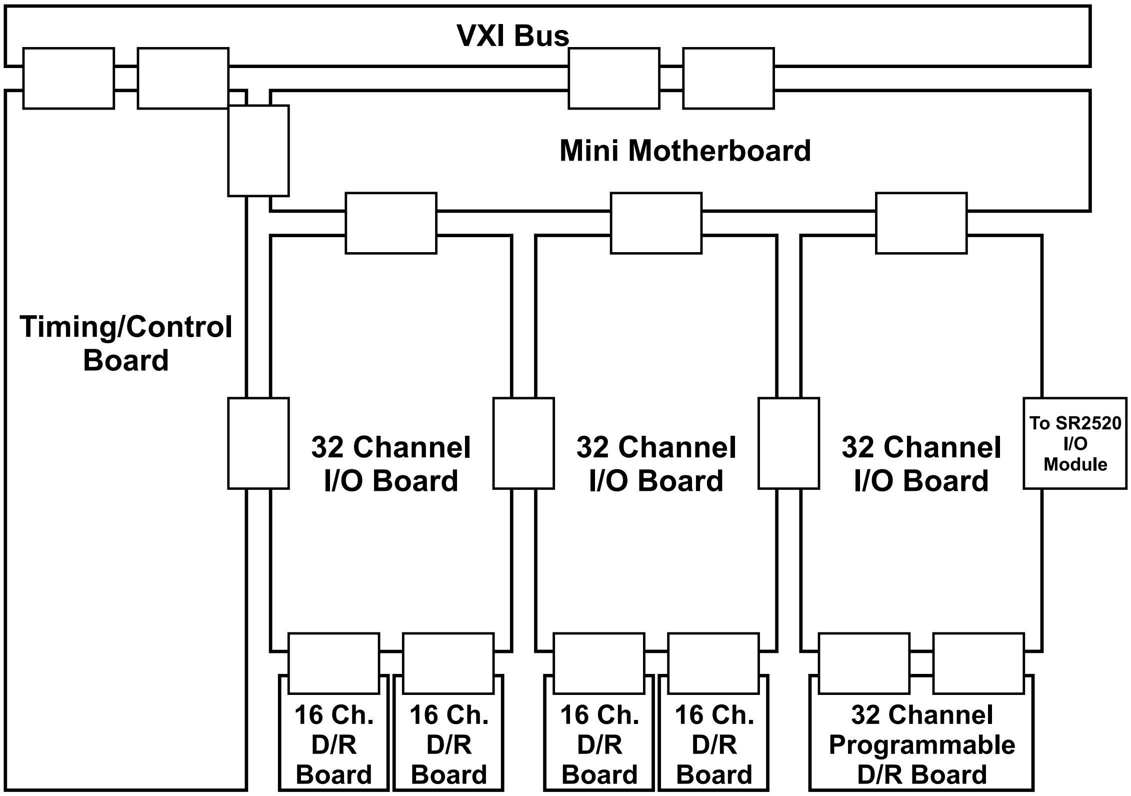


Figure 1-3. Block Diagram, SR2510 Major Components.

SR2520 w/Guided Probe (option)

(see Fig 1-1) Guided Probe is supplied as a factory installed, add-in option to the SR2520 Expansion Module. This option provides the user with the capaiblity to read test points (nodes) on the UUT to determine pass/fail conditions. The guided probe is capable of testing and detecting high, low, and indeterminate states and can also measure analog voltages. Upon determination of the pass/fail state, the guided probe stores the UUT response along with the compare results for later readout. The probe has an active input, which minimizes circuit loading and serves to "condition" the UUT signal before routing it to the guided probe logic. Located on the probe body is an ENTER button used to trigger or continue test execution. For additional details, refer to the *SR2520 Guided Probe Option User's Manual*.

RG2500 Rail Generator (option)

(See Fig 1-1) The RG2500 Rail Generator is a programmable power supply used to provide operating voltages to the SR2510 and SR2520 modules whenever these modules are configured for programmable (variable voltage) operation.

The RG2500 receives operating voltages and control commands from the host computer and supplies either one or two SR2510 or SR2520 modules with eight individual output voltages, each of which is separately programmable over a range of -3.0 to +7.0 Vdc. For additional details, refer to the *RG2500 User's Manual.*

CHAPTER 2

Test Programming Parameters

Test Programs

The SR2500 subsystem is capable of storing multiple user-defined, user-selectable tests in memory. Each test contains complete setup information for test size, field definitions, system timing, input and output formats, record control and stimulus and response data. The maximum tests that may be stored in memory is 128, or until all I/O memory is used up, whichever occurs first. Each time a test is defined, a size (in vectors) must be specified, which causes I/O resources to be allocated for that test. The test size must be an even number of vectors and the actual resources allocated is the size defined plus some internal overhead (up to 32 vectors). Each test retains pointers to its own stimulus/response memory segment, as well as its own unique settings for fields, output format, signal conditioners, etc.

For example, if a test is defined with a size of 100, then 100 I/O vectors are allocated to that test. The available free I/O memory is reduced by 100. This process may be repeated until either 128 tests have been defined, or all of the available I/O memory has been used. If a test is later deleted, then software housekeeping automatically reallocates all of the remaining test resources to keep the unused resources in a sequential address space. The user may select a previously defined test, for execution or modification, via a simple command. Once the user selects a test, system access is directed to the currently active test. The user may also query the system to determine the loaded tests and the status of all defined tests.

Fields

Access to the test's stimulus/response memory is based on "fields." By definition, a field is a logical group of stimulus, response or record pins. Before data can be loaded to or queried from an SR2500 module, the user must create a field. Fields are created by defining:

o a unique field name
o a field type
o the I/O pins associated with that field.

Field width is limited to a maximum of 32 pins. Once a field is defined, the pins associated with that field are not dedicated to only that field. It is possible to define many fields, each containing a pin that was previously defined in another field, or fields. This is very useful for setting up fields and sub-fields. For example, you may define a 4 bit "control" field and a 16 bit data field, and a third field that is 20 bits wide consisting of both control and data. This can be very useful for display purposes, since both control and data may be read with a single command, while retaining the flexibility of manipulating each field independently of the other.

There are three basic field types:

- o stimulus
- o expected response (response)
- o record

There are also sub-types for *stimulus* and *response* fields. Three of the sub-types for *stimulus* and *response* fields are

- o non-algorithmic
- o algorithmic
- o hardware

Non-Algorithmic Fields

Non-algorithmic fields generate their patterns from data stored in RAM. For this reason they are often called RAM-backed patterns. Non-algorithmic fields are very flexible, since pins in a non-algorithmic field can be assigned in any order and any MSB/LSB designation.

Algorithmic Fields

Algorithmic fields generate their patterns algorithmically from a simple set of commands. Test sequences that ordinarily consume excessive amounts of stimulus/response memory can be reduced to just a few algorithmic commands. For example, the "walking bit" data pattern and "incrementing address" patterns often used to functionally test RAM would use proportionately large amounts of stimulus/response memory. The larger the RAM being tested, the more RAM-backed test vectors that would be required. Using algorithmic fields, a single vector of stimulus/ response memory can generate up to 65,536 unique address/data vectors. By placing the algorithmic command within nested loops, literally billions of unique address and data test vectors may be generated using only a few actual vectors.

Because algorithmic patterns are generated in hardware, several restrictions exist concerning how these fields can be assigned. An algorithmic field must consist of byte wide groups of 8, 16, 24 or 32 pins and the MSB/LSB ordering is fixed. None of these restrictions exist for non-algorithmic fields.

One of the algorithmic commands supported allows data to be output from memory, exactly as a RAM-backed field works. This allows mixing of algorithmic and RAM-backed pattern generation on the same pins, but at different test vectors (cycles).

Hardware Fields

Hardware type fields allow loading of data directly to I/O RAM, bypassing the field pin mapping algorithms and the associated processor overhead. By eliminating this overhead, data can be loaded to hardware type fields faster than loading data to other field types. As already discussed above, a pin, or group of pins, may be mapped to multiple fields. So pins may be associated with non-hardware type fields for convenience, and simultaneously associated with hardware type fields for improved performance in querying and loading data patterns.

Vector Looping

A section of stimulus/response memory may be repeatedly looped many times, as determined by a loop count value or by various test conditions. The loops are described as "seamless" because no extra test cycles are required when the program jumps from the bottom of the loop to the top. There are two loop types, *word loops*, which loop on a single test vector, and *start/end loops*, which loop through a range of vectors. Start/end loops may be nested two levels deep, and the nesting must be in a linear sequence of vectors. In other words, with nested start/end loops, it is not allowed to have the first level of looping in the main program sequence, and the second level of looping located within a subroutine. Any number of word loops may be placed within the start/end loop range.

Two other loop structures are provided in the SR2500 subsystem. One is the *program loop*, the other is the *arm count*. With program loops, the entire test program may be repeated up to 65,536 times or continuously. The arm count defines the number of times the SR2500 will re-arm itself and wait for a system trigger. After the trigger, the SR2500 will run through it's entire test sequence as many times as defined with the program loop. Then, assuming the arm count has not been reached, the SR2500 will automatically re-arm itself. Another trigger will cause the test to run again. This process continues until the SR2500 test has run the number of times specified by the arm count value.

Program Branching

The SR2500 test program allows conditional and unconditional test branching. During a *branch*, the data value on the output pins will remain from the test vector immediately prior to the branch instruction. The user has the option of keeping the stimulus data formatting active (useful for keeping clocks active) or suppressing it during the branch, in which case the output pins remain static during the branch. There are three branch types, as follows:

CJMP | JMP

CJMP | JMP allows the test to conditionally or unconditionally branch to any vector location within a test. Test sequencing and pattern generation will continue from the new vector. Jumps to an odd vector require (4) test cycles, while jumps to an even vector requires (5) cycles.

CJSRoutine | JSRoutine

This command allows the test to conditionally or unconditionally branch to a subroutine vector location within a test. Subroutines must start on a 32 vector boundary. Test sequencing and pattern generation will continue from that vector until a return instruction is encountered. After the return, test sequencing and pattern generation continues from the vector immediately after the vector containing the subroutine branch. Subroutines may be nested 8 levels deep and always require (4) test cycles to branch to the subroutine vector.

CRTSubroutine | RTSubroutine

This command initiates a conditional or unconditional return from a subroutine. Returns are not limited to 32 vector boundaries and may reside at any vector location. A *return* to an odd vector location requires (3) test cycles, while a *return* to an even vector requires (4) cycles.

Data Rates

The SR2500 is capable of generating and recording data at rates ranging from 200 Hz to 25 MHz. Using an external clock input supplied to the front panel of the SR2510 module, the SR2500 can support data rates ranging from DC to 25 MHz. By default, the SR2500 is frequency-locked to an internal 10 MHz source. The SR2500 can also be locked to the VXI bus 10 MHz clock (CLK10) or an external 10 MHz reference, which is supplied to the front panel of the SR2510 module.

VME A32 Dual Access Memory

The SR2510 module is configured with 1 MB of dual-access memory, which is mapped to VME A32 address space. This memory is a gateway used for high-speed binary data transfers of stimulus, response and/or record data to and from the SR2500 subsystem. The A32 memory may also be used for high-speed binary block transfers of complete test setups including control, stimulus and response data. The SR2510 module uses the *bus master* mode to transfer the data to or from any SR2520 expansion modules.

Output Data Formatting

Output formatting allows the user to manipulate stimulus output data by impressing a *return-to state* on the output, which is useful when generating high-speed clocks and strobes or serial data streams with a minimum amount of stimulus/response memory.

In a *non-return-to* format, output formatting allows the user to delay (skew) output signals relative to each other up to one full clock period. This aids in meeting or testing UUT setup and hold timing requirements.

Using a *return-to* format, the user can control both the delay and the width of the output format. The delay represents the point at which stimulus data is applied to the output pins, relative to the beginning of the test cycle. The width represents the duration of the output data before the data is returned to the defined *return-to* state. The delay time may be defined at any point within one system clock period. The width parameter must be a minimum of 10 ns and may not exceed one test cycle minus 10 ns. Data format widths may cross cycle boundaries, hence it is permissible to *assert* the state in one cycle, and to *hold* that state into the next cycle. When the internal clock system is used, the delay/width resolution is 12.5% of the system clock. NOTE: The system clock operates in the range of 12.5 MHz to 25.0 MHz. Hence, resolution will vary from 10 ns to 5 ns, respectively. The accuracy of the delay and width settings is 10 ns. When an external clock source is used, the resolution is one-half of the external clock period. The following formats are used to modify output data:

NRZ (Non-Return-to-Zero)

The default mode for output data formatting is NRZ. In this mode, no additional data formatting is impressed on the outputs. The output pins are driven to the defined state after the defined delay time, and remain in that state until the same time in the following cycle, at which point the pins are driven to the newly defined state. The width parameter is not used in NRZ format. True NRZ formatting would place the delay time at 0 ns. Because the NRZ delay time may be placed at any point within the clock cycle, NRZ may also be used for Delayed Non-Return-To-Zero (DNRZ) formats.

RZ (Return-to-Zero)

Return-to-Zero causes the output pins to be driven to the programmed state after the delay time and for the duration of width, and driven to "0" during the delay time and at the end of the width setting. If the tristate control for a pin indicates the pin should be tristated for the test cycle, the tristate control takes priority over the pin formatting and the pin will not be driven to the programmed state, nor return-to-zero.

RONE (Return-to-One)

The return-to-one mode causes the output pins to be driven to the programmed state after the delay time and for the duration of width, and driven to "1" during the delay time and at the end of the width setting. If the tristate control for a pin indicates the pin should be tristated for the test cycle, the tristate control takes priority over the pin formatting and the pin will not be driven to the programmed state, nor return-to-one.

RC (Return-to-Compliment)

The return-to-compliment mode causes the output to be driven to the programmed state after the delay time and driven to its compliment state at the end of the width time. Prior to the assert time (during delay), the outputs are driven to the compliment of the programmed state for the previous cycle. If the tristate control for a pin indicates the pin should be tristated for the test cycle, the tristate control takes priority over the pin formatting and the pin will not be driven to the programmed state, nor return-to-complement.

RI (Return-to-Inhibit/Tristate)

The return-to-inhibit mode causes the output pins to be driven to the programmed state after the delay time and for the duration of width, and tristated during the delay time and at the end of the width setting. If the tristate control for a pin indicates the pin should be tristated for the test cycle, the tristate control takes priority over the pin formatting and the pin will be tristated for the entire test cycle.

Input Data Formatting

The SR2500 provides two methods of sampling and/or comparing the UUT response data:

Edge Mode

In edge mode, UUT response data is latched (sampled) into the input register at the defined time within the test cycle, and compared against the expected UUT response data, masking out any bits indicated by the mask ("don't care") memory. If a compare error is detected, the Error Latch is set. Depending on the record control settings, either the actual UUT response data or the results of the compare -- error data -- are stored in the record memory. Or, if the record control settings so indicate, no information is stored in the record memory.

In the real-time compare mode, the initial expected response comparison may be delayed a total of seven test cycles. This is to compensate for delays external to the SR2500.

Window Mode

The window mode is used to detect glitches. Whereas the edge mode samples the input pins (UUT response) at one instant in time, the window mode in effect samples the inputs over a period of time. The input data must match the expected state and must remain stable for the time duration defined. If the data does not match the expected response, or if the data transitions (glitches) at any time within the window, the result of the response comparison is false (assuming the respective bits are enabled for comparison). If the record memory is programmed to store errors, the bits with detected mismatches (glitches) are set high, even if the initial or final state of the input matches the expected value.

Like the edge compare mode above, the initial expected response window comparison may be delayed a total of seven test cycles, again, to compensate for delays external to the SR2500.

Memory

The SR2500 uses several types of memory to perform specific functions. The SR2500 can be purchased in two sizes of stimulus/response/record memory, providing either 64K or 256K test vector depth.

Control Memory

The control memory provides the sequence instructions (program) for the control processor, which is responsible for the overall vector sequencing of the system. By default, a simple in-line test program of the same length as a defined test is automatically generated. Therefore, the default number of test vectors generated by the program is exactly equal to the length of the test program.

More complex programs may contain looping and conditional or unconditional branches, and may be used to generate many more test vectors than the defined length of the test program. These loop and branch instructions are typically combined with algorithmic pattern generation to produce the desired test vectors.

The stimulus and expected response memories are effectively addressed by the same address counter that drives the control program. Therefore, all stimulus and response fields, whether defined as algorithmic or non-algorithmic, always sequence through the same number of vectors as the control processor. The record memory is addressed in an independent and linear sequence regardless of control program looping and branching.

Stimulus Memory

The stimulus Memories control the generation of stimulus output data. The response memory is addressed by the control processor. For each stimulus pin, the following memory types are available:

Output Memory. The output memory defines the logic states that are driven to the UUT. In the non-algorithmic mode, this data is passed directly from RAM to the output pins, via the stimulus gate array. In the algorithmic mode, the data from the output memory is used as a literal value that may modify the current state of the output pin. One bit of the output memory is assigned for each bit in an output type field (OUT, OT, ALGO and HOUT).

Tristate Memory. Tristate memory determines if the output driver for a given output channel is enabled or disabled. One bit of this memory is assigned for each bit of the tristate type field (TRI, OT and HTRI). The driver is enabled for the cycle if the corresponding tristate bit contains a value of "0". The driver is disabled (tristated) for the cycle if the corresponding tristate bit contains a value of "1".

Stimulus Algorithmic Memory. The stimulus algorithmic memory contains instructions that control the generation of algorithmic output data patterns. Four bits of this memory are used for each byte of algorithmic output type field (ALGO). These four bits determine which of the 16 possible algorithmic operations are to be carried out on that group of eight output bits. Non-algorithmic fields automatically set these bits to "0", which is the equivalent of the non-algorithmic command.

Response Memory. The response memories control the generation of the expected data patterns that are used in real-time compares. The results of the comparison may:

- Be returned at the end of the test
- Be used to control the test program sequence
- Be used to determine starting and stopping of record memory
- Be used to determine starting and stopping of CRC sampling

The response memory is addressed by the control processor. For each input pin, the following response memories are available:

Expect Memory

The expect memory defines the logic state expected to be returned from the UUT. In the non-algorithmic mode, the data in RAM is directly used for the comparison operation. In the algorithmic mode, the data from the expect memory is used as a literal value that may modify the current expected state. One bit of this memory is assigned for each bit in an expect type field (EXP, ED, ALGE and HEXP).

Don't Care Memory

The "don't care" (mask) memory, determines if the UUT response for a given input channel is compared against the expected state of that channel for the current test vector. One bit of this memory is assigned for each bit in a "don't care" type field (DON, ED, HDON). The input value is compared to the expected value if the corresponding bit in the "don't care" memory is programmed with a value of 0. The result of the compare is ignored (masked) if the corresponding "don't care" bit is programmed with a value of 1. When the record error data criteria is selected, a value of 0 is stored to record memory for each bit where the corresponding bit in the "don't care" memory is set to 1.

The "don't care" memory is also used as a mask for enabling signature analysis checksum (CRC) calculations on the individual input pins.

Response Algorithmic Memory

The response algorithmic memory contains instructions that control the generation of algorithmic expected data patterns. Four bits of this memory are used for each byte of expected response type field. These four bits determine which of the 16 possible algorithmic operations are to be carried out on that group of eight expect bits. Non-algorithmic fields automatically set these bits to "0", which is the equivalent of the non-algorithmic command.

Record Memory

The record memory's function is like that of a logic analyzer, i.e., recording data. While the record memory is bundled under the same command subsystem as the response memories, it functions as a wholly separate subsystem from stimulus and response. For this reason it has not been included under the response subsystem in this technical description.

The user may elect to store to record memory in one of two ways ... either the data values returned by the UUT, or the result of a bit-wise comparison between the data from the UUT and the expected data generated by the expect/"don't care" memories (errors). This function is selected with the filter parameter in the Record:Trace command subsystem. In the store error data mode, a value of "1" is stored for each input bit that does not match the expected value. If the input field is operating in the window mode, a value of "1" is stored for each bit in which a pattern mismatch occurs or for each bit where a mismatch or a glitch was detected. If the "don't care" bit is set to "1", a value of "0" is stored regardless of whether a compare mismatch or a glitch occurs.

Record memory may only be loaded as a result of recording UUT responses. It may be queried by the user or used to copy recorded results to expected memory (response learning), but may not be directly loaded by the user.

Unlike the stimulus and response memories, the record memory is addressed and controlled independently of the control processor. The user may selectively store or not store, based on the evaluation of preset trigger conditions and qualifiers. The record operations, as well as the sampling of the CRC (signature analysis) registers, are independently controlled by a 16 level record state machine. This state machine allows for simple or complex triggering capabilities similar to those found in a conventional logic analyzer.

Record Memory and Signature Analysis Control

The SR2500 can perform hardware real-time signature analysis on the data stream of all input pins during qualified cycles. Signature analysis is performed by calculating a 16 bit Cyclic Redundancy Check (CRC) value for each input pin. The results of the CRC calculation are stored in a 16 bit CRC register, one register for each input pin. CRC calculations are performed when enabled by the record control state machine and when the corresponding "don't care" bit is set to "0". The CCITT standard communication polynomial is used to perform the CRC calculations and the CRC value is the 16 bit remainder produced by dividing the input stream by the following polynomial, using Galois field arithmetic:

$$\mathbf{Gx = 1 + x^{5} + x^{12}}$$

Program Languages

The SR2500 programming commands are based on the Standard Commands for Programmable Instruments (SCPI) syntax and are used to set up and query all system functions and execute all run-time controls.

For complete programming instructions, refer to Chapter 3, *"Programming."*

CHAPTER 3

Programming

SCPI Command Syntax

The SR2500 is controlled via a set of word serial commands patterned after the 1993 edition of the Standard Commands for Programmable Instruments (SCPI). To accommodate the robust feature set of the SR2500, many additional commands have been added. Although these additional commands are not defined in SCPI, they do follow the rules and syntax of SCPI commands and provide access to the SR2500's unique features.

SCPI commands are defined in a tree structure starting with a basic command function, called the command root. The tree's functionality is expanded by adding command decriptors, known as command branches. The final command parameter is called a command leaf. With this structure, commands may be logically grouped together based on functionality. Commands defined by SCPI are denoted by the word “SCPI”. The commands that follow the SCPI syntax but are not defined by SCPI are denoted by the word “NON-SCPI.”

SCPI branches and leaf commands use the colon (:) character as a prefix, denoting descent into the command tree by one level. Root commands have no prefix. Some SCPI command paths are quite lengthy, and to avoid typing in the full command path, several short-cuts are available. For example, some SCPI command words are optional. These commands are listed in brackets [ ]. Also, each SCPI command has a long and an abbreviated format. The required abbreviated format is shown in capital letters, while the characters making up the optional long format are shown in lowercase. The following two command strings are identical in functionality:

```
STIMULUS:CMACRO:LABEL:VECTOR 10;REDEFINE START
STIM:LAB:VEC 10;RED START
```

Another shortcut is the semi-colon character (;). Using a semi-colon allows the user to remain at the same level in the command path and enter multiple parameter values, rather than having to re-enter the entire command path for each value entered. In the following example, the first two commands may be replaced by the third example:

```
RECORD:VECTOR 1;COUNT ALL;DATA:FIELD D16_09;CLEAR
RECORD:VECTOR 1;COUNT ALL;DATA:FIELD D08_0;CLEAR
RECORD:VECTOR 1;COUNT ALL;DATA:FIELD D16_09;CLEAR;FIELD
D08_0;CLEAR
```

Table 3-1. SCPI Command Key.

command	Command words take three forms, ROOT, BRANCH, and LEAF. The ROOT is the beginning of a command, i.e. the first word in a command string. Branches are the connecting paths between the ROOT and the LEAF. Branches may or may not have parameters associated with them, or may have a suffix, usually a channel indicator. The LEAF terminates the command string and may or may not have parameters associated with it.
command	Indicates commands which do not have parameters.
command	Indicates commands with parameters.
command(?)	Commands that are followed by a question mark in parenthesis indicate a command format supporting both a command and a command query.
command?	Command strings followed by a question mark without parenthesis indicates a command query only.
UPPERCASE	Command characters displayed in uppercase are required characters.
lowercase	Command characters displayed in lowercase are optional characters.
<required>	Required parameter or suffix.
[option]	Optional command or parameter.
{repeat}	Repeat as many times as required.
(min-max)	The parameter value entered must be within the range of min to max, inclusive.
aaa | bbb	Acceptable choices are aaa OR bbb.
response	Response from SR2500.

Since the parameters FIELD and CLEAR are at the same level within the RECORD command subsystem, the semicolon may be used to omit all of the path command words up to the level where FIELD and CLEAR are specified.

Table 3-1 explains the characters and symbols used in this chapter to represent SCPI Command Syntax.

Chapter 3 is divided into three major sections with each of these further divided into minor sections. For each minor section, commands are presented in the order in which they should be used and not necessarily by subsystem grouping. Each minor section is introduced with a brief description of the commands that will be covered in the section.

Basic Programming

This section is divided into the following sub-sections:

- ***Defining Tests*** ....***pg 3-4***
- ***Global Test Parameters*** ....***pg 3-12***
- ***Field Definitions*** ....***pg 3-30***
- ***Loading and Querying Test Vectors*** ....***pg 3-42***
- ***Trace TMACRO's***...***pg 3-78***
- ***Run Time Commands*** ....***pg 3-87***
- ***Reading Recorded Data*** ....***pg 3-94***

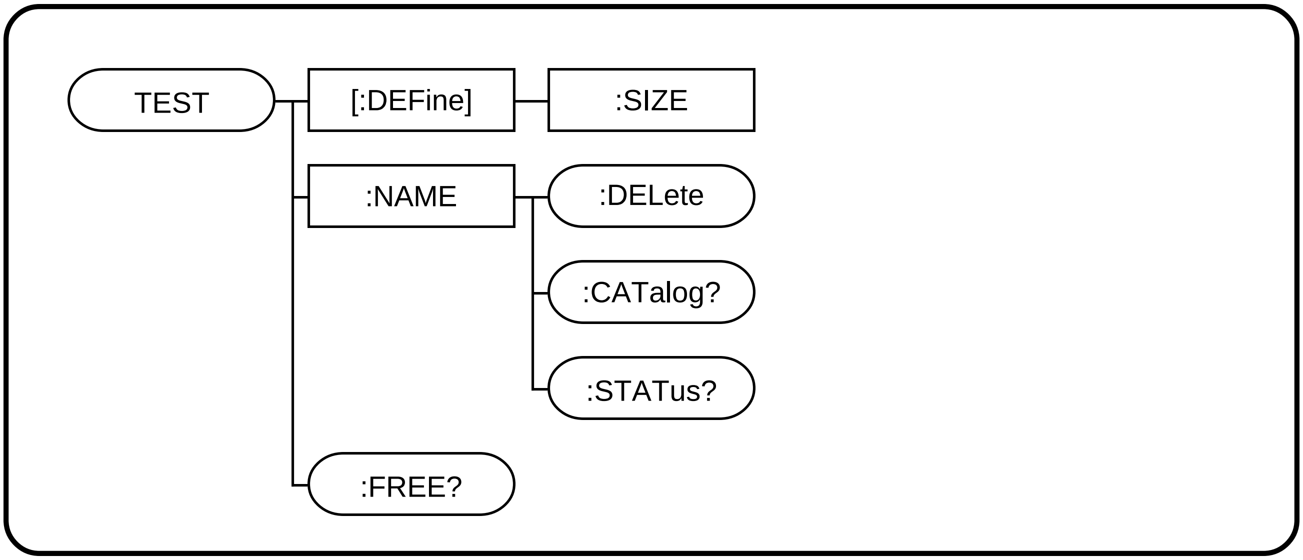
Note

The basic commands required for defining an SR2500 test and entering Stimulus/Response patterns are discussed in the "SR2500 System Manual." You should become thoroughly familiar with those commands and procedures before proceeding with the programming instructions contained in this section.

Defining Tests

The Test subsystem allows the user to allocate available resources to create a test program and to query defined tests to determine available resources. A maximum of 128 unique test programs can be defined within the system at any given time, up to the available SR2500 I/O memory size (256K maximum). Each time a new test is defined, system resources are dynamically allocated. If a defined test is deleted, system resources are dynamically reallocated to maintain a linear block of unused memory.

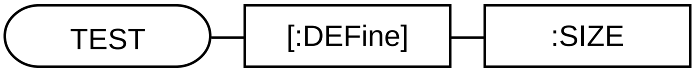
Some parameters, such as test frequency, system trigger and CMACRO sequence programs, are stored in memory located in the SR2510 Timing / Control / I/O Module. Stimulus and expected response data patterns are stored in memory located on the I/O boards. Only one test may be active at a time. Reference the "System" command subsystem for information about activating inactive tests.



SR2500 Default Parameters

ITEM	MIN	MAX	DEFAULT	PARAMETERS
Program Loops	1	65,525	1	CONT
Frequency	200 Hz	25 MHz	25 MHz	---
Period	40 ns	5.0 ms	40 ns	---
Clock Source	---	---	INT	INT/EXT/SSTEP
Clock Slope	---	---	POS	POS/NEG
EXT Clock Threshold Level	-5.00 V	4.99 V	1.20 V	MIN/MAX/DEF
Gate Source	---	---	INT	INT/EXT
EXT Clock Threshold Level	-5.00 V	4.99 V	1.20 V	MIN/MAX/DEF
Gate Polarity	---	---	NORMal	NORM/INV
10 MHz Reference	---	----	INT	INT/EXT/CLK10
System Trigger	---	----	BUS	BUS/EXT/TTLT
EXT Trigger Slope	---	----	POS	POS/NEG
Trigger Level	-5.00 V	4.99 V	1.20 V	MIN/MAX/DEF
Field Radix	---	---	HEX	HEX/BIN
Arm Data Mode	---	---	OFF	ON/OFF
Arm Count	1	1,000,000	1	---
OUTput Field	---	---	All 0's	---
TRIstate Field	---	---	All 1's	---
OT Field	---	---	All X's	---
EXPect Field	---	---	All 0's	---
DONtcare Field	---	---	All 1's	---
ED Field	---	---	All X's	---
RECord Field	---	---	---	---
ALGOutput Field	---	---	Nonalgorithmic	---
ALGExpect Field	---	---	Nonalgorithmic	---

Test Definition (NON-SCPI)



The TEST:DEFINE:SIZE command defines a test and allocates memory on the SR2500 I/O cards. The memory is allocated across all SR2500 I/O cards using the same vector locations. SR2500 I/O card memory is initialized to default values. Every other parameter in the test is also initialized to default values. The maximum number of tests which may be defined in the SR2500 is 128, or until all I/O vectors resources have been used.

[:DEFine] <name>

Parameter Definition **name** = Any alphanumeric string and '_' (max 8 characters).

:SIZE <test_size>

The test size must be an ***even*** number with a minimum value of two (2) vectors and a maximum value of the remaining available free memory. See TEST:FREE? command to query the available free memory.

Note

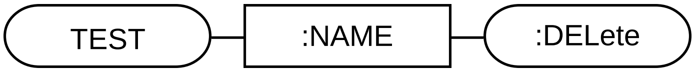
If no tests have been defined, the maximum memory available is 65,500 vectors for a 64K vector card, or 262,108 vectors for a 256K vector card.

Parameter Definition **test_size** = (2 to free_vectors)

Examples

```
TEST:DEFINE MEM_1:SIZE 1000
TEST:DEF MEM_2:SIZE 1000
TEST MEM_3:SIZE 1000
```

Test Deletion (NON-SCPI)



The TEST:NAME:DELETE command deletes a specific test from the test directory. The deletion frees the allocated memory from the I/O cards and shuffles memory to keep all memories contiguous. The physical memory location for each of the remaining tests may change.

:NAME <name | ALL>

Parameter Definition **name** = Any alphanumeric string and ‘_’ (max 8 characters).

ALL = All defined test names.

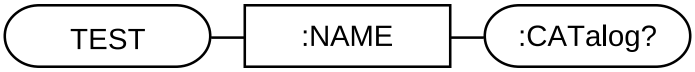
:DELete

Causes the tests specified to be deleted from memory. Memory is reallocated (shuffled) so unused are arranged in a contiguous address space.

Examples

```
TEST:NAME MEM_1:DELETE
TEST:NAME MEM_2:DEL
TEST:NAME ALL:DEL
```

Test Definition Catalog (NON-SCPI)



The TEST:NAME:CATALOG? query command returns the test name and test size parameters of a previously defined test.

:NAME <name | ALL>

Parameter Definition **name** = Any alphanumeric string and '_' (max 8 characters).

ALL = All defined test names.

:CATalog?

Response name test_size{;name test_size}

Parameter Definition **name** = Any alphanumeric string, including underscores '_' (max 8 characters).

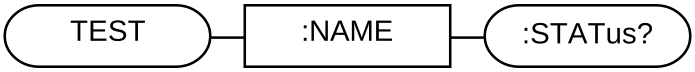
test_size = (2 - 65500) or (2 - 262108)

Examples

TEST:NAME MEM_1:CATALOG?
MEM_1 8192

TEST:NAME ALL:CAT?
MEM_1 8192;MEM_2 4096;MEM_3 2048;MEM_4 1024

Test Definition Status Query



(NON-SCPI)

The TEST:NAME:STATUS? query command returns the status of the specified test.

:NAME <name | ALL>

Parameter Definition

name = Any alphanumeric string, including underscores '_' (max 8 characters).

ALL = All defined test names.

:STATus?

Response

name state,error, stimulus_vector,trace_sequence, vectors_recorded
{;name,state,error,stimulus_vector,trace_sequence,vectors_recorded}

Parameter Definition

name = Any alphanumeric string, including underscores '_' (max 8 characters).

state = IDLE | STOPPED | ARMED | RUNNING

> **IDLE -** The test is idle. The user can set up and query the hardware.
> **STOPPED -** A transitional state. After a test completes, the SR2500 briefly enters the stopped state and then automatically cycles to the idle state. Because of the brief time the SR2500 is in the stopped state, this state will usually not be reported.
> **ARMED -** The test is armed and waiting for a system trigger. The user cannot set up or query the hardware.
> **RUNNING -** The hardware is running. The user cannot set up or query the hardware.

error = 1 for compare errors, 0 for no compare errors

stimulus_vector = current stimulus vector, modulus 2

trace_sequence = current trace sequence as defined by the RECORD:TRACE:SEQUENCE command. This number is invalid when RECORD:TRACE:TMACRO is used.

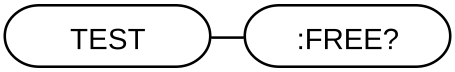
vectors_recorded = total number of vectors recorded. This information is only available after the test has completed.

Examples

TEST:NAME MEM_1:STATUS?
MEM_1 ARMED,0,1000,1,0

TEST:NAME ALL:STAT?
MEM_1 IDLE,0,1000,1,256;MEM_2,RUNNING,0,1000,1,0

Free Vector Space Query (NON-SCPI)



The TEST:FREE? query command returns the amount of free I/O vectors remaining in the system that have not been allocated to defined tests.

:FREE?

Response free_vectors

Parameter Definition The total number of remaining free vectors not allocated to defined tests.

free_vectors = (2 - 65500) or (2 - 262108)

Examples

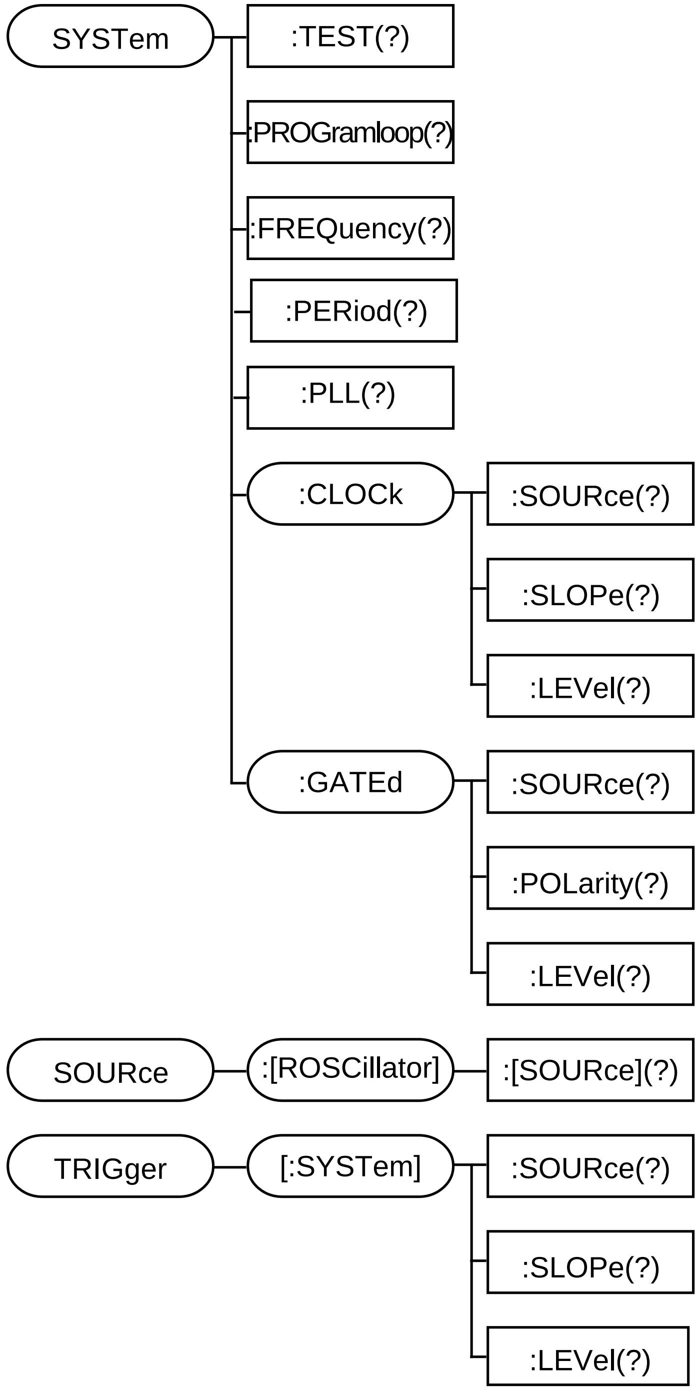
TEST:FREE?
15384

THIS PAGE INTENTIONALLY LEFT BLANK

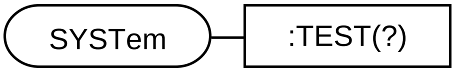
Global Test Parameters

Global Test Parameters, grouped under several command subsystems, allow the user to define system parameters that are unique to a Test Name. Each time a new test is defined, system resources are dynamically allocated and set to their default states. If a defined test is later deleted, system resources are dynamically reallocated to maintain a linear block of unused memory.

Some parameters, such as test frequency, system trigger and CMACRO programs are stored in memory located in the SR2510 module. Stimulus and expected response data patterns are stored in memory located both on the SR2510 and SR2520 modules.


SYSTem
:TEST(?)
:PROGramloop(?)
:FREQuency(?)
:PERiod(?)
:PLL(?)
:CLOCk
:SOURce(?)
:SLOPe(?)
:LEVel(?)
:GATEd
:SOURce(?)
:POLarity(?)
:LEVel(?)
SOURce
:[ROSCillator]
:[SOURce](?)
TRIGger
[:SYSTem]
:SOURce(?)
:SLOPe(?)
:LEVel(?)

Selecting the Active Test (NON-SCPI)



The SYSTem:TEST command activates the specified test name for both editing and test execution. The SYSTem:TEST? query command returns the current active test.

:TEST <name>

Parameter Definition **name** = Any alphanumeric string and '_' (max 8 characters)

Examples SYST:TEST RAM_TEST

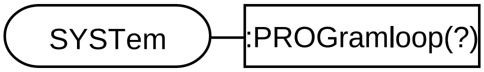
:TEST?

Response name

Parameter Definition **name** = Any alphanumeric string and '_' (max 8 characters)

Examples SYST:TEST?
RAM_TEST

Setting the Program Loop Count (NON-SCPI)



The SYSTem:PROGramloop command defines the number of iterations the test program will execute after each trigger event occurs. The SYSTem:PROGramloop? query command returns the loop count of the active test.

:PROGramloop <count | CONTinuous>

Parameter Definition

count = A numeric value from 1 to 65535. The default value for **count** is 1.

MIN = 1
MAX = 65,535
DEFault = 1

CONTinuous = For continuous looping of test program execution.

Note

If CONTinuous is selected, the ABORt command must be used to stop the test program.

Examples

```
SYSTEM:PROGRAMLOOP CONTINUOUS
SYST:PROG 100
```

:PROGramloop?

Response

count

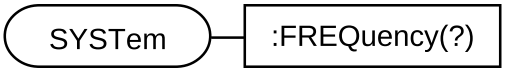
Parameter Definition

count = A numeric value from 1 to 65535 or CONTinuous for continuous looping of test program execution.

Examples

```
SYSTEM:PROGRAMLOOP?
 100

SYST:PROG?
CONT
```

Setting the Test System Frequency (NON-SCPI)



The SYSTem:FREQuency command defines the clock rate of the test program clock. The test program will execute at the specified clock rate also commonly referred to as vector rate or data rate. This command performs the same function as the SYSTem:PERiod command. The system frequency can be specified as a floating point numeric or in scientific notation. The system frequency may also be represented by the literal string MIN, MAX, or DEFault. The SYSTem:FREQuency? query command returns the value of the test program clock.

:FREQuency <hertz | MIN | MAX | DEFault>

Parameter Definition

hertz = (200.00Hz - 25000000.00Hz) Values can be specified as a floating point numeric or in scientific notation using exponential values. Optional Hz, kHz, and MHz suffixes can be used for engineering unit multipliers. The default engineering unit is Hz.

MIN = 200 Hz
MAX = 25 MHz
DEFault = 25 MHz

Examples

```
SYSTEM:FREQUENCY DEFAULT
SYST:FREQ 15.0e6
SYSTEM:FREQ 15MHZ
```

:FREQuency?

Response

hertz

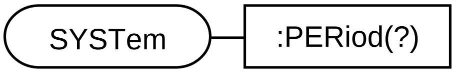
Parameter Definition

hertz = The clock frequency specified in Hz using scientific notation values from 2.000000e+02 to 2.500000e+07.

Examples

```
SYSTEM:FREQUENCY?
 1.500000e+07

SYST:FREQ?
1.500000e+07
```

Setting the Test System Period (NON-SCPI)



The SYSTem:PERiod command sets the clock period of the test program clock. The test program will execute at the specified clock period also commonly referred to as vector rate or data rate. This command performs the same function as the SYSTem:FREQuency command. The system period can be specified as a floating point numeric or in scientific notation. The system period may also be represented by the literal string MIN, MAX, or DEFault. The SYSTem:PERiod? query command returns the period value of the test program clock.

:PERiod <seconds | MIN | MAX | DEFault>

Parameter Definition

seconds = (40.0 ns to 5.0 ms). Values can be specified as a floating point numeric or in scientific notation using exponential values. Optional s, ms, ms, and ns suffixes can be used for engineering unit multipliers. The default engineering unit is s (seconds).

MIN = 40 ns
MAX = 5.0 ms
DEFault = 40 ns

Examples

```
SYSTEM:PERIOD MAX
SYST:PER 2.5e-3
SYSTEM:PER 200ns
```

:PERiod?

Response

seconds

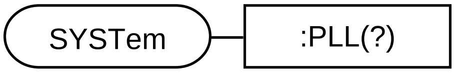
Parameter Definition

seconds = The clock period specified in seconds using scientific notation values from 5.000000e-03 to 4.000000e-08.

Examples

```
SYSTEM:PERIOD?
 2.500000e-3

SYST:PER?
 2.500000e-3
```

Setting the System Clock Frequency (NON-SCPI)



SYSTem:PllFreq,Divider

The PllFreq is the value to set the System Clock to, range 12.5 MHz to 25 MHz where Divider is the value to divide the system clock by range 1 to 65535 (see Chapter 2, *Frequency Resolution* for further details.

SYST:PLL?

Returns the values for the System Clock and the Divider

The following algorithm is used within the SR2510 to calculate the correct SYSTEM_CLOCK and divisor values.

SYSTEM_CLOCK = the actual frequency of the System Clock

DIVIDER = the actual divide-by value (1 - 65,535)

DESIRED_FREQ = the frequency requested by the user, using the SYST:FREQ or SYST:PER commands

START_FREQ = 12.5 MHz, the lower bound of the SYSTEM_CLOCK

STOP_FREQ = 25 MHz, the higher bound of the SYSTEM_CLOCK

TRIAL_DIV = the working divide-by value (1 - 65,535)

TRIAL_CLOCK = the working value for SYSTEM_CLOCK

TRIAL_FREQ = the working value for the vector frequency.

ACTUAL_FREQ = the calculated frequency of the vector cycles,

ACTUAL_FREQ = SYSTEM_CLOCK / DIVIDER

FREQ_STEP = 1.25 kHz, the resolution of the SYSTEM_CLOCK

If the DESIRED_FREQ is less than 12.5 MHz, this algorithim starts by picking the lowest possible TRIAL_DIV (Steps 1-5). Then, it calculates the TRIAL_CLOCK and adjusts it to the nearest FREQ_STEP (Steps 6-8). Then, using the TRIAL_CLOCK and TRIAL_DIV, calculates the TRIAL_FREQ and compares it to the DESIRED_FREQ (steps 13-20). If the difference is less than .001%, we are done (steps 26-29). If not, TRIAL_DIV is incremented by 1 and the process is repeated (steps 21-25). If no TRIAL-FREQ is within .001%, the closest is selected. If the DESIRED_FREQ is 12.5 MHz or greater, the ACTUAL_FREQ is calculated by rounding the DESIRED_FREQ to the nearest 1.25 kHz step (steps 30-21).

For DESIRED_FREQ less than 12.5 MHz.

```
1.  TRIAL_DIV = (INT32)(START_FREQ / DESIRED_FREQ)
2.  WHILE ((TRIAL_DIV * DESIRED_FREQ) < START_FREQ)
3.  {
4.      TRIAL_DIV = TRIAL_DIV + 1
5.  }
6.  TRIAL_CLOCK = TRIAL_DIV * DESIRED_FREQ
7.  steps = (INT32)(((TRIAL_CLOCK - START_FREQ) / FREQ_STEPS) + .05)
8.  TRIAL_CLOCK = START_FREQ + (FREQ + STEP * steps)
9.  save_div = TRIAL DIV
10. save_diff = TRIAL_FREQ - DESIRED_FREQ
11. WHILE (TRIAL_CLOCK <= 25 MHz && TRIAL_DIV < 65536)
12. {
13. TRIAL_FREQ = TRIAL_CLOCK / TRIAL_DIV
14. diff = TRIAL_FREQ - DESIRED_FREQ
15. IF (ABSOLUTE(diff) <= ABSOLUTE(save_diff))
16. {
17.     save_diff = diff
18. save_div = TRIAL_DIV
19. IF (ABSOLUTE(1-(TRIAL_FREQ / DESIRED_FREQ)) <.00001) GOTO STEP 26
20. }
21. TRIAL_DIV = TRIAL_DIV + 1
22. TRIAL_CLOCK = TRIAL_DIV * DESIRED_FREQ
23. steps = (INT32)(((TRIAL_CLOCK - START_FREQ) / FREQ_STEPS) + .05)
24. TRIAL_CLOCK = START_FREQ + (FREQ_STEP) * steps)
25. }
26. DIVIDER = save_div
27. SYSTEM_CLOCK = DIVIDER * DESIRED_FREQ
28. steps = (INT32)(((SYSTEM_CLOCK - START_FREQ) / FREQ_STEP) + .05)
29. SYSTEM_CLOCK = START_FREQ + (FREQ_STEP * steps)
```

For DESIRED_FREQ greater than or equal to 12.5 MHz

```
30. DIVIDER = 1
31. steps = (INT32)(((DESIRED_CLOCK - START_FREQ) / FREQ_STEP) + .05)
32. SYSTEM_CLOCK = START_FREQ + (FREQ_STEP * steps)
```

Example:

```
SYST:PLL 25e6,1000
SYST:PLL 25MHz,1000
SYS:PLL?
2.500000e+07,1000
```

Selecting the System Clock Source (NON-SCPI)

SYSTem
:CLOCk
:SOURce(?)

The SYSTem:CLOCk:SOURce command selects the source of the test program clock. The default clock source is the internal clock on the SR2510 Timing/Control Board. The SYSTem :CLOCk:SOURce? query command returns the source of the test program clock

:SOURce <INTernal | EXTernal | SSTEP>

Parameter Definition

INTernal = (default) The internal clock source from the SR2510 Timing/ Control Board.

EXTernal = The user supplied signal into the front panel "CLOCK IN" connector on the SR2510 Timing/Control Board.

Note

When external clock is selected the test will execute at the rate of the external clock. The SYSTem:FREQuency or SYSTem:PERiod parameters are meaningless and ***cannot*** be modified. Also, the output format for all stimulus fields may be set to any of the available types (NRZ, RZ, RONE, RCOMP, RI), however, edge timing is limited to either the positive or negative edge of the external clock. The same timing restriction applies to edge and window sample clocks. See STIMulus:CONDitioner:OFORmat command for selecting the positive and negative clock edges. See **RECord:CONDitioner:SAMPLE** command for selecting the positive and negative clock edges of response/record fields.

SSTEP = The Single Step function is used to output one vector at a time. The INITiate command is used to advance to the next vector in the sequence.

Examples

SYSTEM:CLOCK:SOURCE INTERNAL
SYST:CLOC:SOUR SSTEP

:SOURce?

Response

INT | EXT | SSTEP

Parameter Definition

INT = The internal clock source from SR2510 Timing/Control Board.
EXT = The user supplied signal into the front panel "CLOCK IN" connector on the SR2510 Timing/Control Board.
SSTEP = The Single Step function is used to output one vector at a time.

Examples

SYSTEM:CLOCK:SOURCE?
EXT

SYST:CLOC:SOUR?
INT

Selecting the Slope of the External Clock (NON-SCPI)

SYSTem
:CLOCk
:SLOPe(?)

The SYSTem:CLOCk:SLOPe command selects the active edge or slope of the External Clock source. This command allows data vectors to be clocked out (output) on either the positive slope (rising edge) or the negative slope (falling edge) of the external clock. The default clock slope is the positive slope of the External Clock. The SYSTem :CLOCk:SLOPe? query command returns the active slope of the External Clock

:SLOPe <POSitive | NEGative>

Parameter Definition

POSitive = (default) Allows vector data to be "clocked out" on the rising edge of the external clock.

NEGative = Allows vector data to be "clocked out" on the falling edge of the external clock.

Note

Selecting the negative slope as the active edge will invert the NRZ delay values for data formatting. See STIMulus:CONDitioner: OFORmat command for selecting the positive and negative clock edges for the NRZ format delays.

Examples

```
SYSTEM:CLOCK:SLOPE POSITIVE
SYST:CLOC:SLOP NEG
```

:SLOPe?

Response POS | NEG

Parameter Definition

POS = Vector data will be "clocked out" on the rising edge of the external clock.

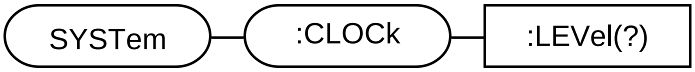
NEG = Vector data will be "clocked out" on the falling edge of the external clock.

Examples

SYSTEM:CLOCK:SLOPE?
POS

SYST:CLOC:SLOP?
NEG

Setting the External Clock Threshold Level (NON-SCPI)



The SYSTem:CLOCk:LEVel command sets the voltage threshold level of the External Clock source. The voltage level can be entered as a floating point numeric or in scientific notation. The voltage threshold may also be represented by the literal string MIN, MAX, or DEFault. The SYSTem:CLOCk:LEVel? query command returns the value of the voltage threshold of the External Clock.

:LEVel:<volts | MIN | MAX | DEFault>

Parameter Definition

volts = (-5.00V to +4.99V) Values can be specified as a floating point numeric or in scientific notation using exponential values. Optional V, MV, and UV suffixes can be used for engineering unit multipliers. The default engineering units is V (volts).

MIN = -5.00V
MAX = 4.99V
DEFault = 1.20V

Examples

SYSTEM:CLOCK:LEVEL 20e-1V
SYST:CLOC:LEV 2.0V

:LEVel?

Response

volts

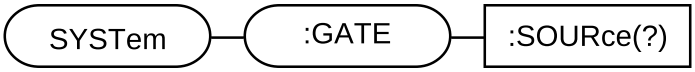
Parameter Definition

volts = The voltage threshold setting specified in volts using scientific notation values from -5.000000e+00 to 4.990000e+00.

Examples

SYSTEM:CLOCK:LEVEL?
2.000000e+00

Selecting the System Gate Source (NON-SCPI)



The SYSTem:GATE:SOURce command selects the source of the test program gate. The default gate source is the ***internal*** gate on the SR2510 Timing/Control Board, which is always enabled. When the ***external*** gate source is selected, the test program clock can be enabled/disabled by the user supplied signal on the "GATE IN" connector on the front panel of the SR2510 Timing/Control Board. While the ***external*** gate is enabled, the test program clock operates normally. When the ***external*** gate is disabled, the test program clock will immediately halt and the logic state of current data vector will be held on the output pins. When the ***external*** gate is once again enabled, the test program will continue its test sequence. The SYSTem:GATE:SOURce? query command returns the source of the test program gate.

:SOURce <INTernal | EXTernal>

Parameter Definition **INTernal** = (default) The internal gate source from the SR2510 Timing/Control Board.

EXTernal = The user supplied signal into the front panel "GATE IN" connector on the SR2510 Timing/Control Board.

Examples SYSTEM:GATE:SOURCE INTERNAL
SYST:GATE:SOUR EXT

:SOURce?

Response INT | EXT

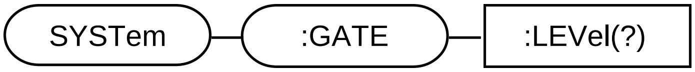
Parameter Definition **INT** = The internal gate source from the SR2510 Timing/Control Board.

EXT = The user supplied signal into the front panel "GATE IN" connector on the SR2510 Timing/Control Board.

Examples SYSTEM:GATE:SOURCE?
EXT

SYST:GATE:SOUR?
INT

Setting the External Gate Threshold Level (NON-SCPI)



SYSTem — :GATE — :LEVel(?)

The SYSTem:GATE:LEVel command sets the input voltage threshold of "GATE IN". The voltage level can be entered as a floating point numeric or in scientific notation. The voltage threshold may also be represented by the literal string MIN, MAX, or DEFault. The SYSTem:GATE:LEVel? query command returns the value of the voltage threshold of the external "GATE IN".

:LEVel:<volts | MIN | MAX | DEFault>

Parameter Definition **volts** = (-5.00V to +4.99V) Values can be specified as a floating point numeric or in scientific notation using exponential values. Optional V, MV, and UV suffixes can be used for engineering unit multipliers. The default engineering unit is V (volts).

MIN = -5.00V
MAX = 4.99V
DEFault = 1.20V

Examples

```
SYSTEM:GATE:LEVEL 20e-1V
SYST:GATE:LEV 2.0V
```

:LEVel?

Response volts

Parameter Definition **volts** = The voltage threshold setting specified in volts using scientific notation values from -5.000000e+00 to 4.990000e+00.

Examples

```
SYSTEM:GATE:LEVEL?
2.000000e+00
```

Selecting the Polarity of the External Gate (NON-SCPI)

SYSTem
:GATE
:POLarity(?)

The SYSTem:GATE:POLarity command selects the active logic level of "GATE IN". The default gate polarity is the NORMal polarity. The SYSTem:GATE:POLarity? query command returns the active logic level of the external "GATE IN" signal.

:POLarity<NORMal | INVerted>

Parameter Definition **NORMal** = (default) A "GATE IN" level above the voltage threshold enables the test program clock, a "GATE IN" level below the voltage threshold disables the test program clock.

INVerted = A "GATE IN" level below the voltage threshold enables the test program clock, a "GATE IN" level above the voltage threshold disables the test program clock.

Examples

```
SYSTEM:GATE:POLARITY NORMAL
SYST:GATE:POL INV
```

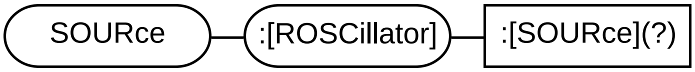
:POLarity?

Response NORM | INV

Parameter Definition **NORM** = (default)

Examples SYST:GATE:POL?
NORM

Selecting the Reference Oscillator Source (SCPI 19.16.3)



SOURce — :[ROSCillator] — :[SOURce](?)

The SOURce:ROSCillator:SOURce command selects the source of the 10 MHz reference for the Phased Lock Loop Oscillator. The internal clock on the SR2510 Timing/Control Board is the default 10 MHz reference source. The SOURce:ROSCillator:SOURce? query command returns the source of the 10 MHz reference.

:SOURce <INTernal | EXTernal |CLK10>

Parameter Definition **INTernal** = (default) The internal 10 MHz reference source on the SR2510 Timing/Control Board. The accuracy of the internal reference is ± 200 ppm with less than 50 ps of short term peak-to-peak jitter. The INTernal clock is the default 10 MHz reference.

EXTernal = The user supplied clock into the front panel "10MHz REF IN" connector on the SR2510 Timing/Control Board. The maximum frequency deviation of the external reference clock must be less than 1% and the short term peak-to-peak jitter must be less than 200 ps.

CLK10 = The CLK10 is a 10 MHz differential ECL clock provided by the Slot-0 and distributed to slots 1-12 on the P2 connector. The CLK10 reference has an accuracy better than ±100 ppm (0.01%) as specified by the VXI Specification.

Examples

```
SOURCE:ROSCILLATOR:SOURCE EXTERNAL
SOUR CLK10
```

:SOURce?

Response INT | EXT | CLK10

Parameter Definition **INT** = The internal 10 MHz reference source from the SR2510 Timing/ Control Board.

EXT = The user supplied clock into the front panel "10MHz REF IN" connector on the SR2510 Timing/Control Board

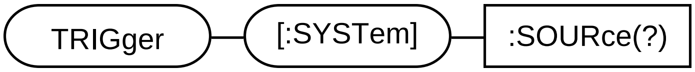
CLK10 = The CLK10 ECL clock provided by the Slot-0 and distributed to slots 1-12 on the P2 connector

Examples

```
SOURCE:ROSCILLATOR:SOURCE?
CLK10
```

```
SOUR?
INT
```

Selecting the System Trigger Source (NON-SCPI)



The TRIGger:SYSTem:SOURce command selects the source of the test system trigger. The test system trigger is used to begin the execution of the active test program. The test system trigger allows the SR2500 test program execution to be synchronized with the VXI backplane or an external trigger signal. The test system trigger may also be used as a loop condition for the single vector word looping (WLoopuntil) and multiple vector word looping (SLoopuntil) CMACRO instructions. See the STIMulus:CMACro:DEFine command for additional information on vector word looping. The TRIGger:SYSTem:SOURce? query command returns the source of the test system trigger.

Note

The SR2500 system must be in the "armed state" before the test system trigger can be activated to begin the test program. See the INITiate command for "arming" the SR2500 system.

:SOURce <BUS | EXTernal | TTLT(0-7)>

Parameter Definition

BUS = (default) The '*TRG' IEEE 488.2 command or the 'TRIG' VXI Word Serial command.
EXTernal = The user supplied signal into the front panel "TRIGGER IN" connector on the SR2510 Timing/Control Board.
TTLT<0-7> = The TTL Trigger lines 0-7 provided on the P2 connector of the VXI Bus.

Examples

TRIGGER:SYSTEM:SOURCE BUS
TRIG:SOURCE EXT
TRIG:SOUR TTLT2

:SOURce?

Response

BUS | EXT | TTLT<0-7>

Parameter Definition

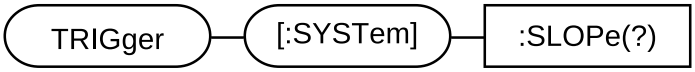
BUS = The '*TRG' IEEE 488.2 command or the 'TRIG' VXI Word Serial command.
EXT = The user supplied signal into the front panel "TRIGGER IN" connector on the SR2510 Timing/Control Board.
TTLT<0-7> = The TTL Trigger lines 0-7 provided on the P2 connector of the VXI Bus.

Examples

TRIGGER:SYSTEM:SOURCE?
EXT

TRIG:SOUR?
TTLT2

Selecting the Slope of the External Trigger (NON-SCPI)



The TRIGger:SYSTem:SLOPe command selects the active edge, or slope, of the External Trigger source. This command allows the test program to begin execution on either the positive slope (rising edge) or the negative slope (falling edge) of the external trigger. The default trigger slope is the positive slope of the external trigger. The TRIGger:SYSTem:SLOPe? query command returns the active slope of the external trigger.

:SLOPe <POSitive | NEGative>

Parameter Definition

POSitive = (default) Allows the test program to be triggered (begin execution) on the rising edge of the external trigger.

NEGative = Allows the test program to be triggered (begin execution) on the falling edge of the external trigger.

Examples

TRIGGER:SYSTEM:SLOPE POSITIVE
TRIG:SLOP NEG

:SLOPe?

Response

POS | NEG

Parameter Definition

POS = The test program will be triggered on the rising edge of the external trigger.

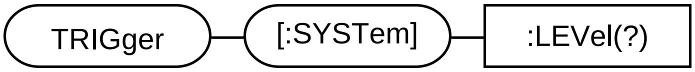
NEG = The test program will be triggered on the falling edge of the external trigger.

Examples

TRIGGER:SYSTEM:SLOPE?
POS

TRIG:SLOP?
NEG

Setting the External Trigger Threshold Level (NON-SCPI)



The TRIGger:SYSTem:LEVel command sets the input voltage threshold of the External Trigger input. The voltage level can be entered as a floating point numeric or in scientific notation. The voltage threshold may also be represented by the literal string MIN, MAX, or DEFault. The TRIGger:SYSTem:LEVel? query command returns the value of the voltage threshold of the external trigger.

:LEVel:<volts | MIN | MAX | DEFault>

Parameter Definition **volts** = (-5.00V to +4.99V) Values can be specified as a floating point numeric or in scientific notation using exponential values. Optional V, MV, and UV suffixes can be used for engineering unit multipliers. The default engineering units is V (volts).

MIN = -5.00V
MAX = 4.99V
DEFault = 1.20V

Examples

```
TRIGGER:SYSTEM:LEVEL 20e-1V
TRIG:LEV 200MV
```

:LEVel?

Response volts

Parameter Definition **volts** = The voltage threshold setting specified in volts using floating point notation values from -5.000000 to 4.990000.

Examples

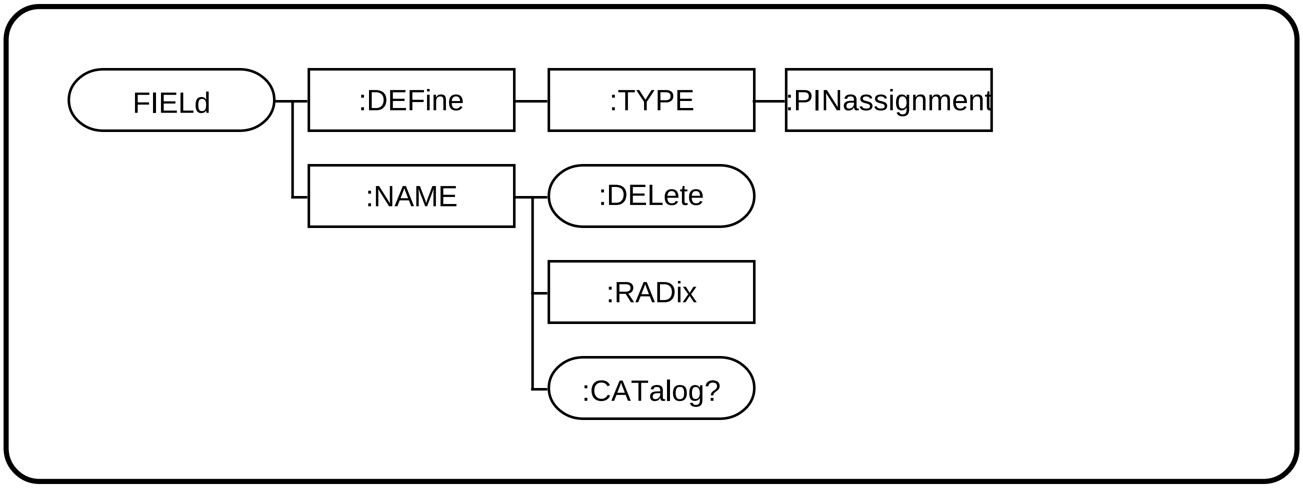
```
TRIGGER:SYSTEM:LEVEL?
0.200000
```

```
TRIG:LEV?
0.200000
```

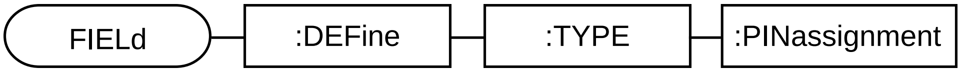
Field Definitions

The Field Definition commands allow the user to create or delete field definitions. A "field" is a mechanism designed to allow test programmers a convenient method of working with channel resources. Loosely defined, a field is a logical grouping of channels. For example, I/O channels that make up an address bus and a data bus would be grouped into two individual fields. A field consists of a name, a type definition and a list of pins. There are two basic field types, RAM backed fields and Algorithmic fields. In a RAM backed field, pins may be defined in any order and across multiple I/O modules. The order in which the pins are defined determines the MSB/LSB order. Pins assigned to an algorithmic field must reside on the same I/O module and MSB/LSB order is predetermined in hardware.

Fields may not be modified once created, with the exception of the field radix. Field radix may be changed at any time. To change a field, you must first delete the field, and then redefine it. Deleting a field has the added effect of deleting all output formatting, input sample timing and data patterns. When a field is first defined, all memories are set to their default values.


FIELd
:DEFine
:TYPE
:PINassignment
:NAME
:DELete
:RADix
:CATalog?

Field Definition & Pin Assignment (NON-SCPI)



The FIELd:DEFine:TYPE:PINassignment command defines the field name, type and channel and pin assignments. Refer to Chapter 1: Introduction for further discussion on Fields.

:DEFine <name>

Parameter Definition **name** = <Any alphanumeric string and '_' (max 8 characters)>

:TYPE <OUTput | TRIstate | OT | EXPected | DONtcare | ED | RECord | ALGOutput | ALGExpected | HOUTput | HTRIstate | HEXPected | HDONtcare | HRECord>

Parameter Definition

OUTput: An Output type field contains RAM backed output patterns which provide stimulus pattern to the UUT. The default state for an output type field is all 0's.

TRIstate: A Tristate type field contains tristate control information for each state of each pin in output memory. For this reason, the pin assignment for a tristate field should have a one-to-one correlation to the pins in the corresponding output field, or Algorithmic Output (ALGO) type field. A '0' bit in a tristate field enables the output pin for the respective vector, and a '1' tristates the output. The default state for Tristate type fields is all 1's.

OT: A composite of the Output and Tristate fields where the user-entered data pattern affects both the Output and Tristate memories. Data loaded or queried to OT type fields may be represented in hex or binary and 'X", where the X represents a bit (binary) or nibble (hex) that is tristated. The default state for OT type fields is X.

EXPected: An Expected field type using Expect memory. An expect type field stores the data used in RAM backed real-time comparisons. Data returning from the UUT is compared to the data stored in the expect type field to determine pass/fail conditions. The default state for an Expect type field is all 0's.

DONtcare: A Dontcare type field holds the bit pattern used to mask out invalid or irrelevant input channels during the real-time compare operation. For this reason, the pin assignment for a Dontcare type field should have a one-to-one correlation to the pins in the corresponding Expect type field, or Algorithmic Expect (ALGE) type field. These mask bits are also used to disable CRC calculations for selected channels. A '0' bit in a Dontcare field enables the compare, or CRC calculation, for the respective vector, and a '1' disables the compare or CRC calculation. The default state for Dontcare type fields is all 1's.

ED: A composite of the Expected and Dontcare fields where the user-entered data pattern affects both the Expected and Dontcare memories. Data loaded or queried to ED type fields may be represented in hex or binary and 'X", where the X represents a bit (binary) or nibble (hex) that is masked out of the compare or CRC calculation. The default state for ED type fields is X.

RECord: Record type fields store the data, or errors, returned by the UUT, when enabled by the record control logic. Record type fields are query only and may only be queried when data is recorded to it. Record controls allow for recording either the data returned by the UUT, as in a Logic Analyzer, or the results of the real-time comparison (error data). The record controls also allow switching between the two record methods during a test.

ALGOutput: Algorithmic Output fields are stimulus fields that generate output patterns algorithmically. Algorithmic patterns are generated real-time according to predetermined instructions, thereby allowing pattern depths many orders of magnitude deeper than traditional RAM backed pattern generators. Also, as these patterns are represented as algorithms, data download is significantly reduced, improving test throughput.

As one of the algorithmic commands is Nonalgorithmic, meaning use data stored in RAM, an algorithmic type field may behave exactly as a RAM backed field. In fact, you may switch between the two modes within the same test. Algorithmic Output type fields may not be combined with Tristate type fields, as in OT type fields, so a separate Tristate field should be created for each ALGO type field. Algorithmic fields default to the Nonalgorithmic instruction.

ALGExpected: Algorithmic Expect fields are response fields that generate expected patterns algorithmically. Algorithmic patterns are generated real-time according to predetermined instructions, thereby allowing pattern depths many orders of magnitude deeper than traditional RAM backed pattern generators. Also, as these patterns are represented as algorithms, data download is significantly reduced, thus improving test throughput.

One of the algorithmic commands is Nonalgorithmic, meaning use data stored in RAM. An algorithmic type field may behave exactly as a RAM backed field. In fact, you may alternate between the two modes within a test. Algorithmic Expect type fields may not be combined with Dontcare type fields, as in ED type fields, so a separate Dontcare field should be created for each ALGE type field. Algorithmic fields default to the Nonalgorithmic instruction.

Note

Any field defined as an algorithmic field type (ALGE and ALGO) must conform to the following algorithmic field rules:

1. Field pins must all reside on the same I/O Module.
2. The pin order must be contiguous (Pin assignment cannot have gaps).
3. The pins must be in groups of eight and must start on pin 32, 24, 16 or 8.
4. The pin numbers must be ordered from high to low (MSB to LSB). An assignment of C1P1-8 is not valid, while C1P8-1 is valid.

HOUTput: A field type of HOUTput is a special hardware mapped Output type field, meaning the pin mapping always follows the hardware MSB to LSB order, and the width of the field must always 32 bits wide. The only valid pin assignments for this type field is CXP32-1, where X represents the I/O board number. Having fields which are pin mapped according to hardware allows the parser to bypass the pin mapping algorithms, which improved the performance of loading and querying data in ASCII format.

HTRIstate: A field type of HTRIstate is a special hardware mapped Tristate type field, meaning the pin mapping always follows the hardware MSB to LSB order, and the width of the field must always be 32 bits wide. The only valid pin assignments for this type field are CXP32-1, where X represents the I/O module number. Having fields which are pin mapped according to hardware allows the parser to bypass the pin mapping algorithms, which improves the performance of loading and querying data in ASCII format.

HEXPected: A field type of HEXPected is a special hardware mapped Expected type field, meaning the pin mapping always follows the hardware MSB to LSB order, and the width of the field must always be 32 bits wide. The only valid pin assignments for this type field are CXP32-1, where X represents the I/O module number. Having fields which are pin mapped according to hardware allows the parser to bypass the pin mapping algorithms, which improves the performance of loading and querying data in ASCII format.

HDONtcare: A field type of HDONtcare is a special hardware mapped DONtcare type field, meaning the pin mapping always follows the hardware MSB to LSB order, and the width of the field must always be 32 bits wide. The only valid pin assignments for this type field are CXP32-1, where X represents the I/O module number. Having fields which are pin mapped according to hardware allows the parser to bypass the pin mapping algorithms, which improves the performance of loading and querying data in ASCII format.

HRECord: A field type of HRECord is a special hardware mapped Record type field, meaning the pin mapping always follows the hardware MSB to LSB order, and the width of the field must always be 32 bits wide. The only valid pin assignments for this type field are CXP32-1, where X represents the I/O module number. Having fields which are pin mapped according to hardware allows the parser to bypass the pin mapping algorithms, which improves the performance of loading and querying data in ASCII format.

:PINassignment <pin_list> | (<chan_list>)

Parameter Definition **pin_list** = <C<card#>P<pin#[-pin#]>[{,C<card#>P<pin#[-pin#]>}>

Pin lists are a simple representation of I/O modules, referenced by card number, followed by a pin number. Multiple pins are delimited by commas, and pin ranges are indicated by using a hyphen '-' character, as illustrated in the examples.

chan_list = <@<card#>!<pin#>[:<card#>!<pin#>][{,
<card#>!<pin#>[:<card#>!<pin#>}]

Channel lists follow the convention defined in the SCPI Syntax and Style document, Volume 1, 1993, Section 8.3.2. Channel lists allow pin definition by I/O board number and pin number, however, where pin lists allow for the definition of pin ranges for a defined I/O board, channel lists allow simultaneous definition of I/O board ranges and pin ranges. The use of the semicolon ':' character implies a range definition.

card# = (1 - number of I/O modules installed)

The card number is determined by its relative position, from left to right, in the SR2500 system. Card 1 is the system left most group of 32 channels. The card number increases as you move to the right.

pin# = (32 - 1)

Assigns the physical I/O pins to the field. The order in which pins are assigned determines the MSB/LSB (Most Significant Bit/Least Significant Bit) order of the bits in the field. The first assigned pin (also the left most entry) represents the MSB, while the last assigned pin (the right most entry) is the LSB. Two formats exist for assigning pins to a field: Pin Lists and Channel Lists (See above).

Fields may overlap one another, or, use some or all of the same pins as already defined fields. This is useful for loading and querying multiple fields simultaneously or including a common signal in multiple fields for reference, such as a clock or other timing signal. For example, you may have 4 discrete fields used for microprocessor control signals, and one additional field combining all 4 discrete signals. The state of each control may be loaded or queried individually, or all at once.

Examples

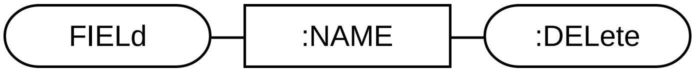
Note

These examples are functionally identical.

```
FIELD:DEFINE ADDR:TYPE OT:PINASSIGNMENT C1P4,C1P3,C1P2,C1P1,
C2P4,C2P3,C2P2,C2P1
FIEL:DEF ADDR:TYPE OT:PIN C1P4-1,C2P4-1
FIEL:DEF ADDR:TYPE OT:PIN (@1!4,1!3,1!2,1!1,2!4,2!3,2!2,2!1)
FIEL:DEF ADDR:TYPE OT:PIN (@1!4:1!1,2!4:2!1)
FIEL:DEF ADDR:TYPE OT:PIN (@1!4:2!1)

FIEL:DEF DATA:TYPE ALGO:PIN C2P32-1
FIEL:DEF DATA_LOW:TYPE OT:PIN C2P16-1
FIEL:DEF DATA_HI:TYPE OT:PIN C2P32-17
FIEL:DEF HDW_DATA:TYPE HOUT:PIN C2P32-1
```

THIS PAGE INTENTIONALLY LEFT BLANK

Field Deletion (NON-SCPI)



The FIELd:NAME:DELete command deletes a previously defined field, or all fields, from the field list.

:NAME <name | ALL>

Parameter Definition **name** = Any alphanumeric string, including underscores '_' (max 8 characters).

ALL = All defined field names.

:DELete

Terminates the command string and causes the specified fields to be deleted from the field list.

Examples

```
FIELD:NAME MEM_1:DELETE
FIEL:NAME ALL:DEL
```

Selecting the Field Radix (NON-SCPI)

FIELd :NAME :RADix

The FIELd:NAME:RADix command sets the specified field’s default radix to Binary or Hexadecimal. The radix is used when loading or querying the data patterns for the named field. When loading data patterns, either hex or binary formats may be used, regardless of the radix setting, by preceding the data with #h or #b prefixes, respectively. See the Stimulus and Record subsystem command sections.

:NAME <name | ALL>

Parameter Definition

name = Any alphanumeric string, including underscores ‘_’ (max 8 characters).

ALL = All defined field names.

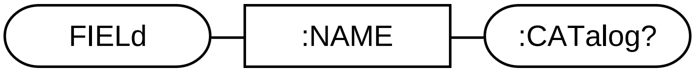
:RADix <HEX | BIN>

<u>Default</u>: HEX

Examples

```
FIELD:NAME MEM_1:RADIX HEX
FIEL:NAME MEM_1:RAD BIN
```

Field Definition Catalog (NON-SCPI)



The FIELd:NAME:CATalog query command returns the parameters of one, or all, previously defined fields.

:NAME <name | ALL>

name = Any alphanumeric string, including underscores '_' (max 8 characters).

Parameter Definition

ALL = All defined field names.

:CATalog?

Response

name,type,radix,pin_list{;name,type,radix,pin_list}

Parameter Definition

name = Any alphanumeric string, including underscores '_' (max 8 characters).

type = OUT | TRI | OT | EXP | DON | ED | REC | ALGO | ALGE | HOUT | HTRI | HEXP | HDON | HREC

radix = HEX | BIN

pin_list = <C<card#>P<pin#>[{,C<card#>P<pin#>}]>

card# = (1 - number of I/O boards installed; up to 18 max. I/O boards are numbered from left to right when facing VXI chassis)

pin# = (1 - 32)

Examples

FIELD:NAME MEM_1: CATALOG?
MEM_1,OT,HEX,C1P4,C1P3,C1P2,C1P1,C2P4,C2P3,C2P2,C2P1

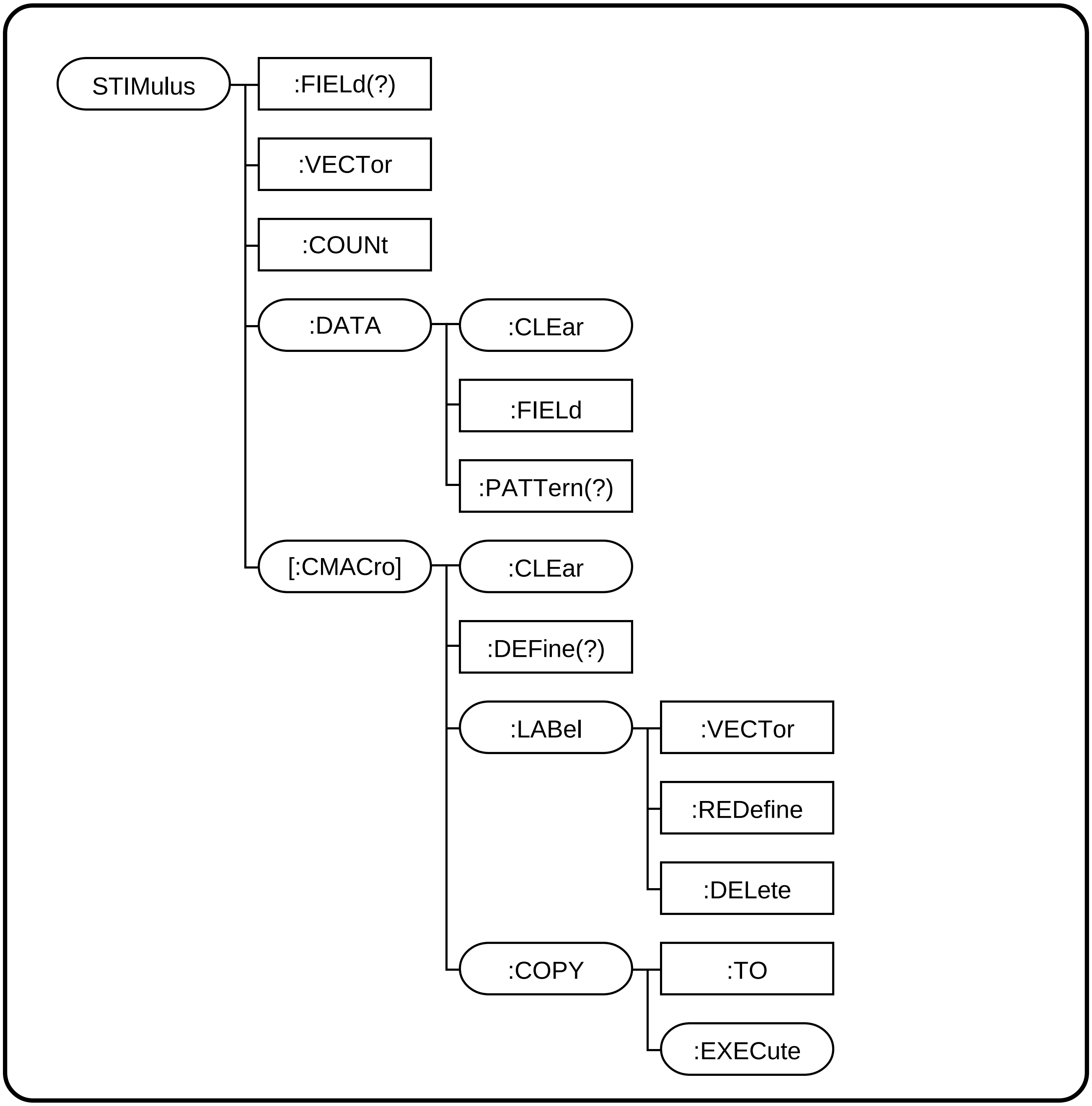
FIEL:NAME ALL: CAT?
MEM_1,OT,HEX,C1P4,C1P3,C1P2,C1P1,C2P4,C2P3,C2P2,C2P1;MEM_2,ED,HEX,C1P4,C1P3,C1P2, C1P1,C2P4,C2P3,C2P2,C2P1

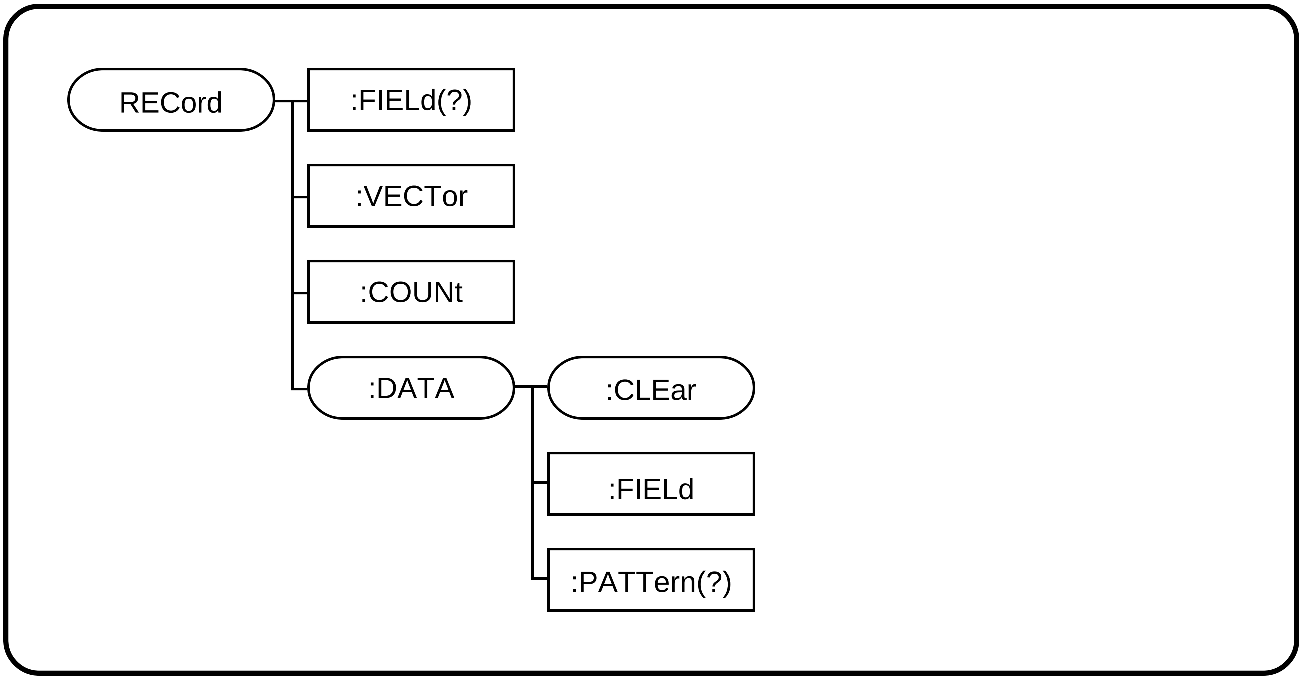
THIS PAGE INTENTIONALLY LEFT BLANK

Loading and Querying Test Vectors

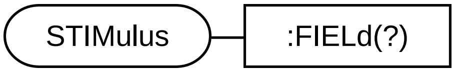
In order to use the SR2500 for testing, you must first load the stimulus and/or response patterns, and (optionally) the CMACRO program. There are two command subsystems that provide access to the SR2500 for loading these parameters. They are the Stimulus subsystem and the Record subsystem. The Record subsystem actually provides access to the Expected Response (response) and Record subsystems. They are grouped under a single subsystem for convenience.

Only the more basic Stimulus and Record commands are detailed in this section. More advance commands for these subsystems are provided in the Advanced Programming section, beginning on pg 3-103.


STIMulus
:FIELd(?)
:VECTor
:COUNt
:DATA
:CLEar
:FIELd
:PATTern(?)
[:CMACro]
:CLEar
:DEFine(?)
:LABel
:VECTor
:REDefine
:DELete
:COPY
:TO
:EXECute


RECord
:FIELd(?)
:VECTor
:COUNt
:DATA
:CLEar
:FIELd
:PATTern(?)

Selecting the Default Stimulus Field (NON-SCPI)



The STIMulus:FIELd command sets the default field for subsequent STIMulus commands. All STIMulus commands that follow will be executed on the default field, unless an alternate field name is specified elsewhere within the STIMulus command. The STIMulus:FIELd? query command returns the name of the default stimulus field.

:FIELd <name>

Parameter Definition **name** = Any alphanumeric string, including underscores '_' (max 8 characters).

Examples STIMULUS:FIELD ADDR
STIM:FIEL ADDR

:FIELd?

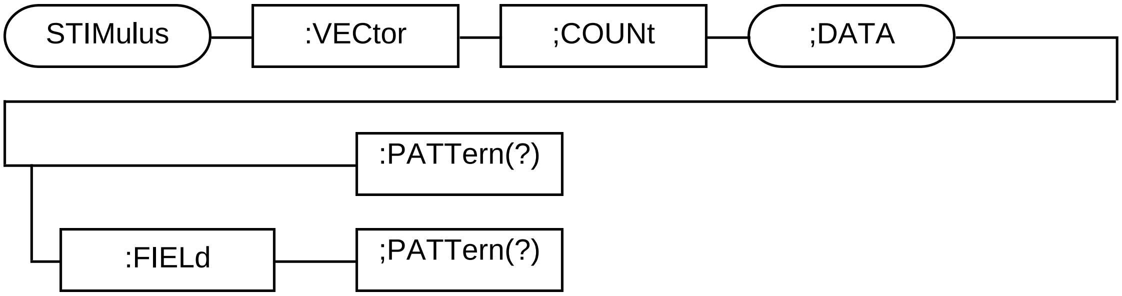
Response name

Parameter Definition **name** = Any alphanumeric string, and '_' (max 8 characters).

Examples STIMULUS:FIELD?
ADDR

STIM:FIEL?
ADDR

Loading/Querying Stimulus Patterns (NON-SCPI)



The STIMulus:VECtor;COUNt;DATA:PATTern command loads output and/or tristate data vectors into the default stimulus memory field. The default memory field is defined by the STIMulus:FIELd command. Valid field types for the STIMulus command are Output (OUT), Tristate (TRI), Output/Tristate (OT), Algorithmic Output (ALGO), Hardware Output, (HOUT), and Hardware Tristate (HTRI). Data will be loaded to the memory field starting at the vector location, specified by the VECtor parameter, and will load the number of vector words specified by the COUNt parameter. The data can be loaded to a destination field other than the default field by using the optional FIELd parameter. The STIMulus:VECtor; COUNt;DATA:PATTern? query command returns the output and/or tristate data vectors from the specified field.

:VECtor <start_vector> The initial vector location where data will start loading (or querying). The starting vector must be within the range of the size of the test (≤ test_size).

Parameter Definition **start_vector** = (1 to test_size)

;COUNt <num_vectors | ALL>

The number of vector words that will be loaded to (or queried from) memory. The number of vectors can also be specified by the literal string "ALL", where "ALL" is equal to the number of vectors from the starting vector location to the last vector in the test. The number of vectors to be loaded/queried must not exceed the last vector in the test. Example: A test is defined to be 100 vectors. The starting destination of memory to be loaded will be at vector location 50. The maximum number of vectors that can be loaded with the same command is 51, where num_vectors = (100-50) + 1 = 51.

Parameter Definition **num_vectors** = (1 to ((test_size - start_vector) + 1))

ALL = All vectors from the start_vector location to the last vector in the test.

;DATA

The DATA command string provides the command path to the PATTern parameter.

Parameter Definition none

:FIELd <name>

The optional FIELd parameter allows the data associated with the same command to be loaded to (or queried from) a destination field other than the default field. If the FIELd parameter option is used, then the FIELd and PATTern(?) parameters must be separated by a semicolon as shown in the example below.

Parameter Definition **name** = Any alphanumeric string and '_' (max 8 characters).

Note

The FIELd parameter changes the destination field only for the same command but **does not** change the default field.

:PATTern <data_value>{,data_value}

The data_value parameter is the actual data that will be loaded to the stimulus memory field. If no radix prefix (#h or #b) is used with the data values, then the data values must be entered in the radix format for the destination field, as defined by the FIELd:NAME: RADix command. If the radix for the destination field is set to HEX, then data can be specified in hexadecimal format (the '#h' prefix is optional) or in binary format if the '#b' prefix is specified. Valid hexadecimal data values are '0' through 'F'. The hexadecimal 'X' character is valid only with Output/Tristate type fields (OT) and represents a tristate condition for that nibble (1 nibble=4 bits).

If the radix for the field is set to BIN, then data can be specified in binary format (the '#b' prefix is optional) or in hexadecimal format if the '#h' prefix is specified. Valid binary data values are '0', '1'. The binary 'X' character is valid only with Output/Tristate type fields (OT) and represents a tristate condition for the corresponding bit position. Leading '0' data characters may be omitted; i.e. '#hF' = '#h000F' and '#b1100' = '#b0000000000001100', for a 16 bit wide field.

Parameter Definition **data_value** = <[#h]{(0-F) | X}> | <[#b]{0 | 1 | X}>

Note

The number of data_value elements must be equal to num_vectors. If a count mismatch occurs, the data will be loaded up to the number of data_value elements or the num_vectors, whichever is less. An error message will be generated.

Examples

STIMULUS:VECTOR 1;COUNT 4;DATA:FIELD ADDR;PATTERN #h00AA, #h0055,#h00AA,#h0055
STIM:VEC 1;COUN 4;DATA:PATT AA,55,AA,55
STIM:VEC 1;COUN 4;DATA:PATT #b0000000010101010,#b0000000001 010101,#b0000000010101010,#b0000000001010101
STIM:VEC 1;COUN 4;DATA:PATT#b10101010,#b01010101,#b10101010, #b01010101

Note

All the commands shown above perform identical functions. The default field is ADDR and is set to HEX radix.

:PATTern?

The data_value parameter is the actual data that will be read from the stimulus memory field. The radix of data_value is determined by the FIELd:NAME:RADix command. If the radix for the field is set to HEX, then data will be returned in hexadecimal format with the '#h' prefix. Valid hexadecimal data values are '0' through 'F'. The hexadecimal 'X' character is valid only with Output/Tristate type fields (OT) and represents a tristate condition for that nibble (1 nibble=4 bits). The hexidecimal '?' character will be displayed when a nibble contains a combination of enabled and tristated stimulus pins

If the radix for the field is set to BIN, then data will be returned in binary format with the '#b' prefix. Valid binary data values are '0', '1'. The binary 'X' character is valid only with Output/Tristate type fields (OT) and represents a tristate condition for the corresponding bit position. Each field defined can have a different radix format. Leading '0' data characters will be returned.

Response data_value{,data_value}

Parameter Definition **data_value** = <#h{(0-F) | X}> | <#b{0 | 1 | X}>

Examples

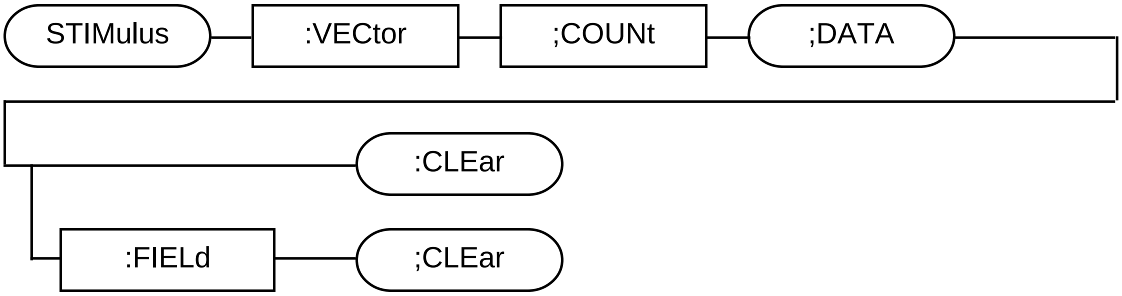
STIMULUS:VECTOR 1;COUNT 4;DATA:FIELD ADDR;PATTERN?
#h00AA,#h0055,#h00AA,#h0055

STIM:VEC 1;COUN 4;DATA:PATT?
#h00AA,#h0055,#h00AA,#h0055

STIM:VEC 1;COUN 2;DATA:PATT?
#b01X00110, #b0011X100

#h?6, #h3?

Clearing Stimulus Patterns (NON-SCPI)



The STIMulus:VECtor;COUNt;DATA:CLEar command clears the data pattern by loading "all zeros" (0's) into the default memory field. If the destination field is an output type (OUT, ALGO, HOUT), the data pattern will be set to all zeros. If the destination field is a tristate type (TRI, HTRI), the data pattern will be set to all enable condition (0's). If the destination field is an output/tristate type (OT), the data pattern will be set to all zeros and enable condition. The default memory field is defined by the STIMulus:FIELd command. Data will be cleared starting at the vector location, specified by the VECtor parameter, and will clear the number of vector words specified by the COUNt parameter. A destination field other than the default field can be cleared by using the optional FIELd parameter.

:VECtor <start_vector> The initial vector location where data will be cleared. The starting vector must be within the range of the size of the test (≤ to test_size).

Parameter Definition **start_vector** = (1 to test_size)

;COUNt <num_vectors> The number of vector memory words that will be cleared. The number of vectors can also be specified by the literal string "ALL", where "ALL" is equal to the number of vectors from the starting vector location to the last vector in the test. The number of vectors to be cleared must not exceed the last vector in the test. Example: A test is defined to be 100 vectors. The starting destination of memory to be cleared will be at vector location 50. The maximum number of vectors that can be cleared with the same command is 51, where num_vectors = (100-50) + 1 = 51.

Parameter Definition **num_vectors** = (1 to ((test_size-start_vector) + 1))

ALL = All vectors from the start_vector location to the last vector in the test.

;DATA The DATA command string provides the command path to the CLEar parameter.

Parameter Definition none

:FIELd <name>

The optional FIELd parameter allows a destination field other than the default field to be specified. The data in the alternate field will be cleared. If the FIELd parameter option is used, then the FIELd and CLEar parameters must be separated by a semicolon as shown in the example below.

Parameter Definition **name** = Any alphanumeric string and '_' (max 8 characters).

Note

The FIELd parameter changes the destination field only for the same command but **does not** change the default field.

:CLEar

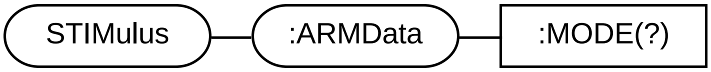
Causes the Output and/or Tristate data patterns for the specified field to be cleared.

Parameter Definition none

Examples

```
STIMULUS:VECTOR 1;COUNT 4;DATA:FIELD ADDR;CLEAR
STIM:VEC 1;COUN 4;DATA:CLE
```

Enabling the Armdata Function (NON-SCPI)



The STIMulus:ARMData:MODE command enables/disables the ARMDATA function. The ARMDATA function allows a user-defined data pattern to be output on the stimulus pins, while the SR2500 is in an ARMED state, waiting for a trigger. The user-defined data pattern will be held on the stimulus pins until the SR2500 is triggered and the test program begins execution. The STIMulus:ARMData:MODE? query command returns the status of the ARMDATA function.

:MODE < mode_num | ON | OFF >

The MODE parameter can be specified as a numeric value or a literal string. The Armdata function is enabled by entering a "non-zero" numeric value for mode_num or by entering the literal string "ON". The Armdata function is disabled by entering a "0" numeric value for mode_num or by entering the literal string "OFF".

Parameter Definition

mode_num = (0 | 1); where "1" value enables the ARMDATA function and "0" value disables the ARMDATA function.

ON = Enables the ARMDATA function.
OFF = Disables the ARMDATA function.
DEFAULT = Off

Examples

```
STIMULUS:ARMDATA:MODE ON
STIM:ARMD:MODE 1
STIM:ARMD:MODE OFF
```

:MODE?

Response

0 | 1

Parameter Definition

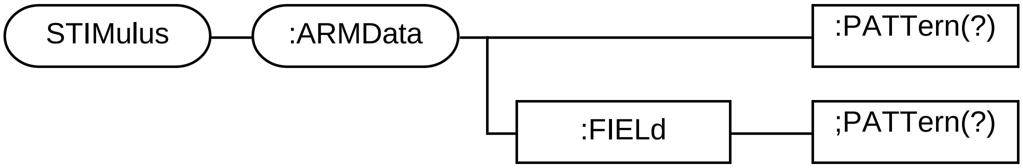
0 = The ARMDATA function is disabled.
1 = The ARMDATA function is enabled.

Examples

STIMULUS:ARMDATA:MODE?
1

STIM:ARMD:MODE?
0

Setting the Armdata Pattern (NON-SCPI)



The STIMulus:ARMData:PATTern command defines the data pattern that will be output to the stimulus pins when the SR2500 is placed in the ARMED state and waiting for a trigger. The SR2500 is placed in the ARMED state when a.) the INITiate command is received; or b.) when the SR2500 has completed a test and the ARM:COUNt has not been fulfilled. Each stimulus field has it's own arm data parameter and can be defined with a unique data pattern. The STIMulus:ARMData:PATTern? query command returns the arm data value for the specified field.

:FIELd <name>

The optional FIELd parameter specifies the stimulus memory field that the arm data pattern will be assigned to (or queried from). Valid field types for the FIELd parameter are Output (OUT), Tristate (TRI), Output/Tristate (OT), Algorithmic Output (ALGO), Hardware Output, (HOUT), and Hardware Tristate (HTRI). If the FIELd parameter option is used, then the FIELd and PATTern(?) parameters must be separated by a semicolon as shown in the example below. If the FIELd parameter is omitted, then the default stimulus field is assumed. The default stimulus field is defined by the STIMulus:FIELd command.

Parameter Definition

name = Any alphanumeric string and ‘_’ (max 8 characters).

Note

The FIELd parameter changes the destination field only for this occurrence of the command, but **does not** change the default field.

:PATTern <arm_data>

The arm_data parameter is the actual data value that will be output to the stimulus field pins while the SR2500 is in the ARMED state. The arm_data can be entered in hexadecimal or binary format by specifying the '#h' or '#b' prefix respectively.

Valid hexadecimal data values are '0' through 'F'. The hexadecimal 'X' character is valid only with Output/Tristate fields (OT) and represents a tristate condition for that nibble (1 nibble=4 bits).

Valid binary data values are '0', '1'. The binary 'X' character is valid only with Output/Tristate fields (OT) and represents a tristate condition for the corresponding bit position. Leading '0' data characters may be omitted; i.e. '#hF' = '#h000F' and '#b1100' = '#b0000000000001100', for a 16 bit example field.

Parameter Definition

arm_data = <#h{(0-F) | X}> | <#b{0 | 1 | X}>

Examples

```
STIMULUS:ARMDATA:FIELD ADDR;PATTERN #h00AA
STIM:ARMD:PATT #hAA
STIM:ARMD:PATT #b0000000010101010
STIM:ARMD:PATT #b10101010
```

Note

All the commands shown above perform identical functions. The default field is ADDR and is set to HEX radix.

:PATTern?

The name of the queried stimulus field and the arm_data pattern for that field will be returned. The radix of data_value returned is determined by the FIELd:NAME:RADix command. If the radix for the field is set to HEX, then the pattern will be returned in hexadecimal format with the '#h' prefix. Valid hexadecimal data values are '0' through 'F'. The hexadecimal 'X' character is valid only with Output/Tristate fields (OT) and represents a tristate condition for that nibble (1 nibble=4 bits). The hexidecimal '?' character will be displayed when a nibble contains a combination of enabled and tristated stimulus pins.

If the radix for the field is set to BIN, then the pattern will be returned in binary format with the '#b' prefix. Valid binary data values are '0', '1'. The binary 'X' character is valid only with Output/Tristate fields (OT) and represents a tristate condition for the corresponding bit position. Leading '0' data characters will be returned.

Response

name, arm_data

Parameter Definition

name = Any alphanumeric string and '_' (max 8 characters).

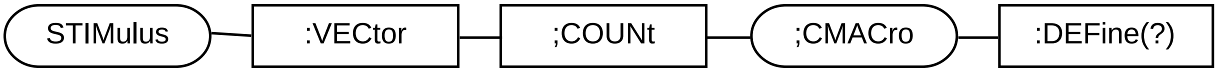
data_value = <#h{(0-F) | X}> | <#b{0 | 1 | X}>

Examples

STIMULUS:ARMDATA:FIELD ADDR;PATTERN?
ADDR,#h00AA

STIM:ARMD:PATT?
IN_DATA,#b00001010111XX0X0

Loading the Stimulus Macro Command Memory (NON-SCPI)



The STIMulus:;;CMACro:DEFine command loads the macro command memory with the command instructions that control the sequence of the stimulus vectors. Each stimulus and response vector has an associated macro command that determines the next vector location to use. The default macro command for each vector location is the OUTput command. The OUTput command executes the current test vector, and then proceeds to the next sequential vector. All test programs must have, as a minimum, a Start Program command (SProgram) and an End Program command (EProgram), where the SProgram command must be the first vector in the test. The EProgram command can be at any vector location, and there can be multiple EProgram commands in a test. If SProgram and EProgram are not specified, then the first and last vector in the test program are defaulted to the SProgram and EProgram commands respectively.

Other macro commands include conditional and unconditional looping and branching such as Single Vector Looping (WLoopuntil), Multiple Vector Looping (SLoopuntil), Jump To Vector (JMP), Jump Subroutine (JSRoutine/CJSRoutine), and Return Subroutine (RTSubroutine/CRTSubroutine). Macro commands will be loaded to memory starting at the vector location, specified by the VECtor parameter, and will load the number of macro commands specified by the COUNt parameter. The STIMulus:;;CMACro:DEFine? query command returns the macro commands for the specified vector location(s).

Note

At least one stimulus type field must be defined before CMACRO instructions may be downloaded. Refer to FIELD:DEFINE:TYPE:PIN.

:VECtor <start_vector> The initial vector location where macro commands will start loading (or querying). The starting vector must be within the range of the size of the test (≤ test_size).

Parameter Definition **start_vector** = (1 to test_size)

;COUNt <num_vectors | ALL>

The number of macro command vectors that will be loaded to (or queried from) memory. The number of macro command vectors can also be specified by the literal string "ALL", where "ALL" is equal to the number of vectors from the starting vector location to the last vector in the test. The number of macro command vectors to be loaded/queried must not exceed the last vector in the test.

Parameter Definition **num_vectors** = (1 to ((test_size-start_vector) + 1)

ALL = All vectors from the start_vector location to the last vector in the test.

[;CMACro] The optional CMACro command string provides the command path to the DEFine string.

Parameter Definition none

:DEFine <([{(LABel <label_name>) | (SUBLabel <label_name>)}] <macro_cmd>) {,([{(LABel <label_name>) | (SUBLabel <label_name>)}] <macro_cmd>)}>

The DEFine command string defines the macro command for each of the specified vector locations. The macro command controls the test sequence by determining the next test vector to be executed, and is analogous to a source code listing for the SR2500 test program. The parenthesis used in the DEFine command string are literal characters and do not represent parameter ranges as described in the Command Syntax Key, Table 5-1. The '< >', '[' and '{' characters are symbols used to represent required, optional, and repetitive parameters respectively.

Each test vector can have up to 10 alphanumeric Labels and/or Sublabels associated with it. Labels and SubLabels are used as destination parameters for vector jumping commands and subroutine branching, respectively. Subroutines must start on (32 vector + 1) boundaries. For example, test vector location 97 can be assigned a sublabel name of "I_O_TEST", which signifies the beginning vector location for a test subroutine. A JSRoutine command can branch to the subroutine by specifying the sublabel name "I_O_TEST". Labels and SubLabels are also useful for documenting test programs.

Note

Multiple macro command vectors can be defined with a single DEFine command string, however, it is **strongly recommended** that each macro command vector be specified by its own DEFine command string so as to simplify documentation and debugging.

Parameter Definition

label_name = Any alphanumeric string and '_' (max 8 characters).

macro_cmd = The following is a list of valid macro commands. 1.

1. SProgram [(OUT)]
2. EProgram [(OUT)]
3. OUTput [(OUT)]
4. WLoopuntil ([OUT] (< loop_cond >))
5. SLoopuntil ([OUT] (< loop_cond >))
6. ELoop [(OUT)]
7. JMP ([OUT] (< label_name >))
8. JSRoutine ([OUT (< label_name >))
9. RTSubroutine [OUT]
10. SCONDition ([OUT] (<jump_cond >))
11. CJMP ([OUT] (< label_name >))
12. CJSRoutine ([OUT] (< label_name >))
13. CRTSubroutine [OUT]
14. CLEARError [(OUT)]

loop_cond = The following is a list of valid conditions to evaluate for determining if a Word Loop or Start/End loop should terminate.

1. COUNt == count_value
2. RCOMpare == TRUE
3. RCOMpare != TRUE
4. LATCherror == TRUE
5. STRIgger == TRUE
6. FRONtpanel && match_pattern
7. FRONtpanel &! match_pattern
8. QUALifword && qual_combination
9. QUALifword &! qual_combination

jump_cond = The following is a list of valid conditions to evaluate for determining if a conditional Jump, Conditional Jump Subroutine or Conditional Return Subroutine should be executed.

1. RCOMpare == TRUE
2. RCOMpare != TRUE
3. LATCherror == TRUE
4. LATCherror != TRUE
5. FRONtpanel && match_pattern
6. FRONtpanel &! match_pattern
7. QUALifword && qual_combination
8. QUALifword &! qual_combination

count_value = (1-65535)

match_pattern = <#h{0-F | X}> | <#b{0 | 1 | X}>

The match_pattern parameter is the 8 bit pattern used to compare against the 8 front panel input flags, and may be represented in hex (#h prefix) or binary (#b prefix). If the hex radix prefix is used, then the valid hexadecimal data values are '0' through 'F', and the hexadecimal 'X' character represents a don't care condition for the corresponding nibble. If the binary radix prefix is used, then valid binary data values are '0', '1', and the binary 'X' character represents a don't care condition for the corresponding bit position.

qual_combination = <#h{0-F}> | <#b{0 | 1}>

The qual_combination parameter is an 8 bit value used to select a combination of the 8 record qualifiers to compare against the input data, and may be represented in hex (#h prefix) or binary (#b prefix). If the hex radix prefix is used, then the valid hexadecimal data values are '0' through 'F'. If the binary radix prefix is used, then valid binary data values are '0' and '1'. Valid examples of qual_combination are #h0A and #b00001010, both of which enable simultaneous comparison against record qualifiers number 2 and 4.

:DEFine?

The DEFine query command string returns the macro command for each of the specified vector locations. The macro command controls the test sequence by determining the next test vector to be executed, and is analogous to a source code listing for the SR2500 test program. The parenthesis used in the DEFine? query command string are literal characters and do not represent parameter ranges as described in the Command Syntax Key, Table 5-1. The '< >', '[' and '{' characters are symbols used to represent required, optional, and repetitive parameters respectively.

Response <([{(LAB <label_name>) | (SUBL <label_name>)}] <macro_cmd>)
{,([{(LAB <label_name>) | (SUBL <label_name>)}] <macro_cmd>)}>

COMMAND DEFINITIONS

SProgram[(OUT)]

The Start Program instruction denotes the beginning of a test program. Only one Start Program instruction is permitted per test, and must be the first instruction in the test, i.e., at vector number one. This instruction requires one clock period to execute.

Examples

```
STIMULUS:VECTOR 1;COUNT 1;CMACRO:DEFINE (SPROGRAM(OUT))
STIM:VECT 1;COUN 1;CMACRO:DEFINE (SP(OUT))
STIM:VECT 1;COUN 1;CMACRO:DEFINE (SP)
```

EProgram[(OUT)]

The End Program instructions denotes the end of a test program. While only one Start Program instruction is permitted per test, any number of End Program instructions are allowed. This instruction requires one clock period to execute.

Examples

```
STIMULUS:VECTOR 10;COUNT 1;CMACRO:DEFINE (EPROGRAM(OUT))
STIM:VECT 32767;COUN 1;CMACRO:DEFINE ((LABEL END)EP(OUT))
STIM:VECT 65500;COUN 1;CMACRO:DEFINE (EP)
```

OUTput[(OUT)]

The Output instruction causes the Control Processor to step to the next vector in the sequence at the end of the test cycle. All Control memory locations are automatically filled with the Output instruction by the System Processor when a test is initially defined. This instruction requires one clock cycle to execute.

Examples

STIMULUS:VECTOR 2;COUNT 1;CMACRO:DEFINE (OUTPUT(OUT))
STIM:VECT 10;COUN 2;CMACRO:DEFINE ((LABEL P_START)OUT),(OUT)
STIM:VECT 1025;COUN 5;CMACRO:DEFINE ((LAB L1)(SUBL L6)OUT),(OUT),(OUT),(OUT),(OUT(~~NOP~~))

WLoopuntil([OUT](<loop_cond>))

The Word Loop Until instruction allows looping at a single vector until the defined condition is detected. If the condition is true, program execution continues at the vector after the Word Loop instruction. If the condition is false, program execution remains at the same vector where the Word Loop instruction is located (the conditions that may be tested by the Word Loop instruction are discussed later). The Word Loop instruction requires one clock period to execute under all conditions and the pattern looping is seamless.

Examples

```
STIMULUS:VECTOR 2;COUNT 1;CMACRO:DEFINE
(WLOOPUNTIL(OUT(COUNT == 100)))
STIM:VECT 2;COUN 1:DEFINE ((LABEL W_DTACK)WL(OUT (STRI ==
TRUE))
STIM:VECT 2;COUN 1:DEFINE (WL((RCOM == TRUE)))
```

SLoopuntil([OUT](<loop_cond>))

The Start Loop Until instruction marks the beginning point of a multi-vector loop. Loop branching is seamless. Although the loop condition is specified by the Start Loop instruction, it is not tested until the corresponding End Loop instruction is executed (see below). If the condition being evaluated is true, the test falls through to the vector after the End Loop instruction. If the condition tested is false, program execution loops back to vector where the Start Loop instruction is located, not to the vector following Start Loop. As a result of the test being performed at the bottom of the loop, the code within a loop will always be executed at least once. Start/End loops may be nested two levels deep and both instructions require one clock period to execute.

Examples

```
STIMULUS:VECTOR 2;COUNT 1;CMACRO:DEFINE
(SLOOPUNTIL(OUT(COUNT == 100)))
STIM:VECT 2;COUN 1:DEFINE ((LAB WRITE)SL(OUT(STRI == TRUE)))
STIM:VECT 200;COUN 1:DEFINE ((LAB READ)SL(OUT(RCOM == TRUE)))
```

ELoop[(OUT)]

The End Loop instruction marks the range (beginning and end) of a multiple vector loop, respectively. Loop branching is seamless. Each instruction requires one clock period to execute under all conditions. Although the loop condition is specified by the Start Loop instruction, it is not tested until the corresponding End Loop instruction is executed. If the condition is true, the test falls through to the vector after the End Loop instruction. If the condition is false, program execution loops back to vector where the Start Loop instruction is located, not to the vector following Start Loop. As a result of the test being performed at the bottom of the loop, the code within a loop will always be executed at least once. Start/End loops may be nested two levels deep. For Start/End loops, the following rules apply:

Note

Failure to observe the following rules may lead to unpredictable results.

1. For every Start/End Loop instruction encountered, the Control Processor must encounter a corresponding End/Start Loop instruction, respectively.
2. If a Jump to Subroutine instruction is executed inside a Start/End loop, the program must eventually return before the End Loop instruction is executed.
3. If nesting Start/End loops, both loops must be in a linear sequence of vectors. It is not permissible to have the first level Start/End loop in

the main program sequence, and have the second level loop in a subroutine. Either both loops must be in the main program sequence, or both loops must be in the subroutine.

Examples

```
STIMULUS:VECTOR 10;COUNT 1;CMACRO:DEFINE (ELOOP(OUT))
STIM:VECT 32767;COUN 1;CMACRO:DEF ((LAB E_WRITE)EL(OUT))
STIM:VECT 65500;COUN 1;CMACRO:DEF ((LAB E_READ)EL)
```

SCONDition([OUT](<jump_cond>))

The Conditional Jump, Conditional Jump to Subroutine and Conditional Return from Subroutine instructions require that the condition being evaluated be previously set with the Set Condition instruction. Failure to do so may lead to unpredictable results. This instruction requires one test cycle to execute.

Examples

```
STIMULUS:VECTOR 10;COUNT 1;CMACRO:DEFINE
(SCONDITION(OUT(RCOMPARE == TRUE)))
STIM:VECT 10;COUN 1;CMACRO:DEF (SCOND(OUT(FRON &&
#b10XXXXXX)))
STIM:VECT 10;COUN 1;CMACRO:DEF (SCOND(OUT(QUAL &&
#b10001111)))
```

SJMPPage (Supported Only on 256K Vector Cards)

If jumps (JMP or CJMP) are performed beyond the first 64K vectors, this instruction must be executed before the jump to specify the jump page. The Set Jump Page instruction accepts a label as a parameter and calculates the specific jump page address. The jump page address is held in the Output and Tristate memories, so any value written to these memories by the user are overwritten during test initialization (INIT). For this vector, the output pins are held with the state and tristate condition from the previous vector. The SJMPP instruction should be used in all tests developed for 256K and 1M vector depth systems even if the current test is not larger than 64K. This instruction must be used in any test loaded, if the total number of vectors in all tests defined is over 64K. Ideally, the SJMPP instructions should be placed just before the corresponding JMP or CJMP instruction.

Examples None

JMP([OUT](<label_name>))

The Jump instructions causes test execution to unconditionally branch to the vector specified. If the vector is not in the current 64K page, the Set Jump Page instruction must have previously been executed. This instruction is not seamless and requires four clock cycles for a jump to an odd vector, or five clock cycles for a jump to an even vector. The jump to address is held in the Output and Tristate memories (for stimulus) and Expect and Dontcare memories (for response), so any value written to these memories by the user are overwritten during test initialization (INIT). During the jump process, the output pins are held with the state and tristate condition from the previous vector and the expect and Dontcare pattern is also held from the previous vector. Pin formatting remains active during the jump, so an output pin which might be generating a clock using a return-to-zero format would remain active during the jump, unless.

Parameter Definition **label_name** = Any alphanumeric string and '_' (max 8 characters). Must have previously been defined with either the LABel or SUBLabel optional parameter in the CMACRO command.

Examples

```
STIM:VECT 500;COUNT 1;CMACRO:DEFINE ((LAB END) EP (OUT))
STIM:VECT 600;COUNT 1;CMACRO:DEFINE ((LAB L1) SL (OUT(STR1==TRUE)))
STIM:VECT 700;COUNT 1;CMACRO:DEFINE ((LAB L6) SL (OUT(RCOM==TRUE)))
```

JSRoutine([OUT](<label_name>))

The Jump-to-Subroutine instructions causes test execution to unconditionally branch to the vector specified. The subroutine vector must be located on a (32 vector +1) boundary. If the vector is not in the current 64K page, the Set Jump Page instruction must have previously been executed. The actual jump to address is held in the Output and Tristate memories (for stimulus) and Expect and Dontcare memories (for response), so any value written to these memories by the user are overwritten during test initialization (INIT). During the jump process, the output pins and the expected response patterns are held with the state from the previous vector. Pin formatting remains active during the jump, so an output pin which might be generating a clock using a return-to-zero format would remain active during the jump.

Jump-to-Subroutine instructions require 4 cycles to execute and may be nested up to eight levels deep, meaning that eight Jump-to-Subroutines, and/or Conditional-Jump-to-Subroutines, may be executed before a Return-from-Subroutine, or Conditional-Return-from-Subroutine, must be performed. All Subroutines must have matching Returns and all subroutines must be completed before the End Program instruction is executed or unpredictable conditions may result. No stack overflow or underflow trapping exists.

Parameter Definition **label_name** = Any alphanumeric string and '_' (max 8 characters). Must have previously been defined with either the LABel or SUBLabel optional parameter in the CMACRO command.

Examples

```
STIM:VECT 129;COUNT 1;CMACRO:DEF((SUBL WRITE)WL(OUT(COUNT==20)))
STIM:VECT 161;COUNT 1;CMACRO:DEF((SUBL READ)WL(OUT(COUNT==20)))
STIM:VECT 321;COUNT 1;CMACRO:DEF((SUBL WRT_RD) WL(OUT
(COUNT==10)))
```

RTSubroutine[(OUT)]

The Return-from-Subroutine instruction causes the address on top of the stack to be popped and program execution to unconditionally resume at the vector after the Jump-to-Subroutine, or Conditional-Jump-to-Subroutine. This instruction is not seamless. It requires three clock periods for a return to an odd address and four clock periods for a return to an even address. Pin formatting remains active during the return, so an output pin which might be generating a clock using a return-to-zero format would remain active.

Examples

```
STIMULUS:VECTOR 10;COUNT 1;CMACRO:DEFINE
(RTSUBROUTINE(OUT)
STIM:VECT 10;COUN 1;CMACRO:DEF (RTS(OUT))
STIM:VECT 255;COUN 1;CMACRO:DEF (RTS)
```

CJMP([OUT](<label_name>))

The Conditional-Jump instructions causes test execution to branch to the vector specified if the defined jump condition evaluates true. The Set Condition command must have previously been executed, and, if the vector is not in the current 64K page, the Set Jump Page instruction must also have been previously executed. This instruction is not seamless and requires four clock cycles for a jump to an odd vector, or five clock cycles for a jump to an even vector. If the conditional jump is not taken, the instruction requires one clock period to execute. The jump to address is held in the Output and Tristate memories (for stimulus) and Expect and Dontcare memories (for response), so any value written to these memories by the user are overwritten during test initialization (INIT). During the jump process, the output pins are held with the state and tristate condition from the previous vector and the expected and Dontcare pattern is also held from the previous vector. Pin formatting remains active during the jump, so an output pin which might be generating a clock using return-to-zero format would remain active during the jump.

Parameter Definition

label_name = Any alphanumeric string and '_' (max 8 characters). Must have previously been defined with either the LABel or SUBLabel optional parameter in the CMACRO command.

Examples

```
STIMULUS:VECTOR 10;COUNT 1;CMACRO:DEFINE (CJMP(OUT(END)))
STIM:VECT 10;COUN 1;CMACRO:DEF (CJMP(L1)
STIM:VECT 255;COUN 1;CMACRO:DEF (CJMP(OUT(L6)))
```

CJSRoutine([OUT](<label_name>))

The Conditional-Jump-to-Subroutine instructions causes test execution to branch to the vector specified if the defined jump condition evaluates true. The Set Condition command must have previously been executed, and the jump vector must be located on a (32 vector +1) boundary. If the vector is not in the current 64K page, the Set Jump Page instruction must also have been previously executed. The jump to address is held in the Output and Tristate memories (for stimulus) and Expect and Dontcare memories (for response), so any value written to these memories by the user are overwritten during test initialization (INIT). During the jump process, the output pins and the expected response patterns are held with the state from the previous vector. Pin formatting remains active during the jump, so an output pin which might be generating a clock using a return-to-zero format would remain active.

Jump-to-Subroutine instructions require 4 cycles to execute and may be nested up to eight levels deep, meaning that eight Jump-to-Subroutines, and/or Conditional-Jump-to-Subroutines, may be executed before a Return-from-Subroutine, or Conditional-Return-from-Subroutine, must be performed. All subroutines must have a matching Returns and all subroutines must be completed before the End Program instruction is executed or unpredictable conditions may result. No stack overflow or underflow trapping exists.

Parameter Definition

label_name = Any alphanumeric string and '_' (max 8 characters). Must have previously been defined with either the LABel or SUBLabel optional parameter in the CMACRO command.

Examples

```
STIMULUS:VECTOR 10;COUNT 1;CMACRO:DEFINE
(CJSROUTINE(OUT(WRITE)))
STIM:VECT 10;COUN 1;CMACRO:DEF(CJSR(OUT(READ)))
STIM:VECT 255;COUN 1;CMACRO:DEF(CJSR(OUT(WRT_RD)))
```

CRTSubroutine[(OUT)]

The Conditional-Return-from-Subroutine instruction causes the address on top of the stack to be popped and program execution to resume at the vector after the Jump-to-Subroutine, or Conditional-Jump-to-Subroutine, if the defined return condition evaluates true. This instruction is not seamless. It requires three clock periods for a return to an odd address and four clock periods for a return to an even address. Pin formatting remains active during the return, so a clock which might be generated using a return-to-zero format would remain active during the return.

Examples

```
STIMULUS:VECTOR 10;COUNT 1;CMACRO:DEFINE
(CRTSUBROUTINE(OUT))
STIM:VECT 10;COUN 1;CMACRO:DEF (CRTS(OUT))
STIM:VECT 255;COUN 1;CMACRO:DEF (CRTS)
```

CLEARError[(OUT)]

The Clear-Error instruction causes the Response Compare Error Latch to be reset. The state of the Response Compare signal is continuously monitored by the system processor. If in any cycle the response input vector does not match the expect vector, for bit locations where the Don’t Care bit is 0, the Response Compare Error Latch is set and remains set until the Clear Error Latch CMACRO is executed. This instruction requires one clock period to execute. If the Response Compare Error condition is still present while this instruction is executed, the latch is immediately set again.

Note

As a by-product of initiating (starting) a test, the Response Compare pipeline is filled with error conditions, and the Error Latch is set indicating an error. To use the Error Latch in a test, the Response Compare pipeline must be flushed and the Error Latch reset. This can be done by defining a vector with the bits of all Dontcare memories set to 1, then loop on that vector for at least 10 cycles. After the loop, the CLEARError instruction must be executed. If this procedure is not followed, the Error Latch will always indicate a response compare error has occurred.

Examples

```
STIMULUS:VECTOR 10;COUNT 1;CMACRO:DEFINE
(CLEARERROR(OUT))
STIM:VECT 10;COUN 1;CMACRO:DEF (CLEARE(OUT))
STIM:VECT 255;COUN 1;CMACRO:DEF (CLEARE)
```

Note

The following CMACRO commands provide an example of how the Response Compare pipeline may be flushed and the Error Latch reset. This example assumes that all Dontcare memories for vectors 1 and 2 are set to '1'. The actual stimulus and response test patterns start with vector 3.

```
STIM:VECT 1;COUN 1;CMACRO:DEF (SP)
STIM:VECT 2;COUN 1;CMACRO:DEF (WL(OUT(COUN==10)))
STIM:VECT 3;COUN 1;CMACRO:DEF (CLEARE)
```

CONDITION DEFINITIONS

COUNt == count_value

This condition evaluates true after the loop has been executed the defined number of times. The loop value may range from 1 to 65,535. This condition may be used with the Start/End Loop and the Word Loop commands, but not with the Set Condition command, which implies that it may not be used with the Conditional Jump, Conditional Jump-To Subroutine or the Conditional Return-From Subroutine.

RCOMpare == TRUE

The Response Compare condition is true when all response input bits match the corresponding Expected Response bits, where the corresponding Don’t Care bits contains a value of 0. Response Compares is a dynamic indication of the results of the input data being compared to the expected response pattern for the current vector only, unlike the Error Latch. This condition may be used with the Start/End Loop, Word Loop and the Set Condition commands.

RCOMpare != TRUE

The Response Does Not Compare condition is true when any of the input bits do not match the corresponding Expected Response bits, where the corresponding Don’t Care bits contain a value of 0. Response Does Not Compare is a dynamic indication of the results of the input data being compared to the expected response pattern for the current vector only, unlike the Error Latch. This condition may be used with the Start/End Loop, Word Loop and the Set Condition commands.

LATCherror == TRUE

The Error Latch Set condition is true if the Response Compare Error Latch is set. The Response Compare Error Latch is set whenever a Response Does Not Compare condition occurs, and will remain set until cleared by the CLEARError instruction. This condition may be used with the Start/End Loop, Word Loop and the Set Condition commands.

LATCherror != TRUE

The Error Latch Not Set condition is true if the Response Compare Error Latch is not set. The Response Compare Error Latch is set whenever a Response Does Not Compare condition occurs, and will remain set until cleared by the CLEARError instruction. This condition may be used with the Set Condition CMACRO only. It is not an option for Word Loop or Start/End Loop commands.

STRIgger == TRUE

This condition is true when the currently selected system trigger event occurs. The trigger may be defined as the IEEE 488.2 *TRG command, a VXI Word Serial Trigger, both of which use the Bus Trigger Source, one of the VXI bus TTL triggers (TTLTRG0-7) or the front panel trigger input. The polarity of the VXI bus TTL trigger and the front panel trigger is normally set to the rising edge, but may be inverted to trigger on the falling edge. The front panel trigger input uses a comparator with a programmable threshold which may be adjustable between ±5.00 Volts. This condition may be used with the Start/End Loop and the Word Loop commands, but may not be used with the Set Condition command.

FRONtpanel && match_pattern

This command provides a match evaluation of the 8 TTL input flags located on the front panel against an 8 bit match pattern. For this condition, the match pattern is represented as either a hex (#h) or binary (#b) value, which includes X's to denote masked inputs. If the match pattern is represented in hex, then an X will mask out the 4 corresponding input flags. The condition is true if any of the enabled front panel input flags match the corresponding compare bit. If a match bit is defined as X, then the corresponding input flag is ignored (will always evaluate false). If all bits are X, then the evaluation is always false. This condition may be used with the Start/End Loop, Word Loop and the Set Condition commands.

FRONtpanel &! match_pattern

The Input Flag Pattern Mismatch condition is true if all of the enabled front panel TTL input flags do not match the corresponding compare bits. The match pattern is represented as either a hex (#h) or binary (#b) value, which includes X's to denote masked inputs. If the match pattern is represented in hex, then an X will mask out the 4 corresponding input flags. This instruction will always evaluate to true if the match pattern is set to all X's. Like the Input Flags Pattern Match condition, the 16 bit literal field is logically broken into two eight-bit fields. The lower eight bits (7-0) are used to bitwise compare against the front panel input flags. The upper eight bits are used to bitwise enable the comparison. This condition may be used with the Start/End Loop, Word Loop and the Set Condition commands.

QUALifword && qual_combination

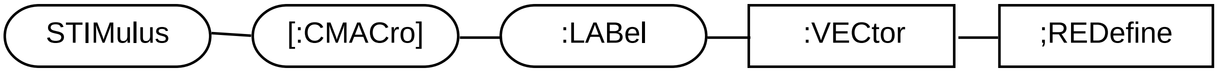
The SR2500 supports eight system-wide response input comparators called Qualifiers (1-8). Each qualifier can be programmed to compare each bit in a record type field against a 1, 0 or Dontcare value. A qualifier is true if all enabled bits match the input pattern. The condition is true if any of the selected qualifiers evaluate true. This condition may be used with the Start/End Loop, Word Loop and the Set Condition commands.

QUALifword &! qual_combination

The Qualifier Mismatch condition is true if none of the selected qualifiers evaluate true. Each qualifier can be programmed to compare each bit in a record type field against a 1, 0 or Dontcare value. A qualifier is true if all enabled bits match the input pattern. This condition may be used with the Start/End Loop, Word Loop and the Set Condition commands.

(THIS PAGE INTENTIONALLY LEFT BLANK)

Redefining Macro Command Label Vectors (NON-SCPI)



The STIMulus:CMACro:LABel:VECtor;REDefine command allows a previously defined Label or Sublabel name to be redefined to another vector. A label may be redefined to any other vector number, while a sublabel may only be redefined to vectors on a (32 vector +1) boundary. Redefining the vector that a label or sublabel is associated with allows jump locations and subroutines to be changed without having to redefine all of the vectors where the jump or subroutine calls are made.

:LABel Provides a path connecting the CMACro command to the LABel command.

Parameter Definition none

;VECtor <vector_num> The new vector number that label_name will be associated with.

Parameter Definition **vector_num** = (1 - test_size)

The new vector number that a Sublabel name will be associated with must be on a (32 vector + 1) boundary - vector 33, 65, 97, and so on.

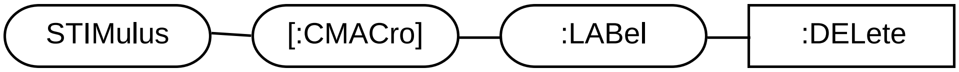
:REDefine <label_name Specifies the Label or Sublabel name that you wish to associate with a new vector number.

Parameter Definition **label_name** = Any alphanumeric string and '_' (max 8 characters). Must have previously been defined with either the LABel or SUBLabel optional parameter in the CMACRO command.

Examples

```
STIMULUS:CMACRO:LABEL:VECTOR 10;REDEFINE START
STIM:CMAC:LAB:VEC 20;RED END
STIM:LAB:VEC 97;RED IO_TEST
```

Deleting Macro Command Labels (NON-SCPI)



The STIMulus:CMACro:LABel:DELete command allows a previously defined Label or Sublabel name to be deleted from the CMACRO program. This command does not delete the label name in the jump or subroutine calls, only the label names associated with the vector.

:LABel

Provides a path connecting the CMACro command to the LABel command.

Parameter Definition

none

:DELete <label_name>

Specifies the Label or Sublabel name that you wish to delete.

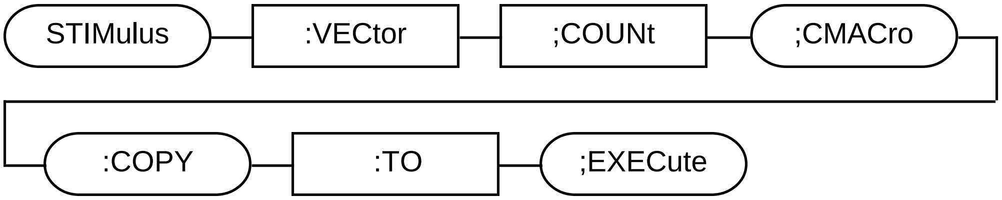
Parameter Definition

label_name = Any alphanumeric string and '_' (max 8 characters). Must have previously been defined with either the LABel or SUBLabel optional parameter in the CMACRO command.

Examples

```
STIMULUS:CMACRO:LABEL:DELETE START
STIM:CMAC:LAB:DEL END
STIM:LAB:DEL IO_TEST
```

Copying Stimulus Macro Commands (NON-SCPI)



The STIMulus::;CMACro:COPY command allows a range of CMACRO instructions to be copied from one location to another. Instructions starting at vector location defined by VECTor, and for the number of vectors defined by COUNt, will be copied to vector location defined by the TO parameter.

:VECtor <start_vector> The initial vector location where macro commands will be copied from.

Parameter Definition **start_vector** = (1 to test_size)

;COUNt <num_vectors | ALL> The number of macro command vectors that will be copied to the new vector location. The number of macro command vectors can also be specified with the literal string "ALL", where "ALL" is equal to the number of vectors from the starting vector location to the last vector in the test. The number of macro command vectors to be copied must not exceed the last vector in the test.

Parameter Definition **num_vectors** = (1 to ((test_size-start_vector) + 1)

ALL = All vectors from the start_vector location to the last vector in the test.

[;CMACro] The optional CMACro command string provides the command path to the DEFine string.

Parameter Definition none

:COPY The COPY command string specifies that a CMACRO copy function will be performed.

Parameter Definition none

:TO <dest_vector> Specifies the starting vector location where the macro commands will be copied.

Parameter Definition **dest_vector** = (1 - test_size)

The range of macro command vectors copied must fit within the range bounded by dest_vector and the end of the test.

:EXECute

Executes the CMACRO copy function.

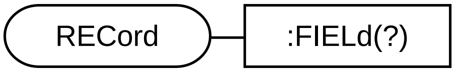
Parameter Definition

none

Examples

```
STIMULUS:VECTOR 12;COUNT 6;CMACRO:COPY:TO 42;EXECUTE
STIM:VEC 33;COUN 16;CMAC:COPY:TO 65;EXEC
```

Selecting the Default Record Field (NON-SCPI)



The RECord:FIELd command sets the default field for subsequent RECord commands. All RECord commands that follow will be executed on the default field, unless a different field name is specified. The RECord:FIELd? query command returns the name of the default field.

:FIELd <name>

Parameter Definition **name** = Any alphanumeric string, including underscores '_' (max 8 characters).

Examples

RECORD:FIELD ADDR
REC:FIEL ADDR

:FIELd?

Response name

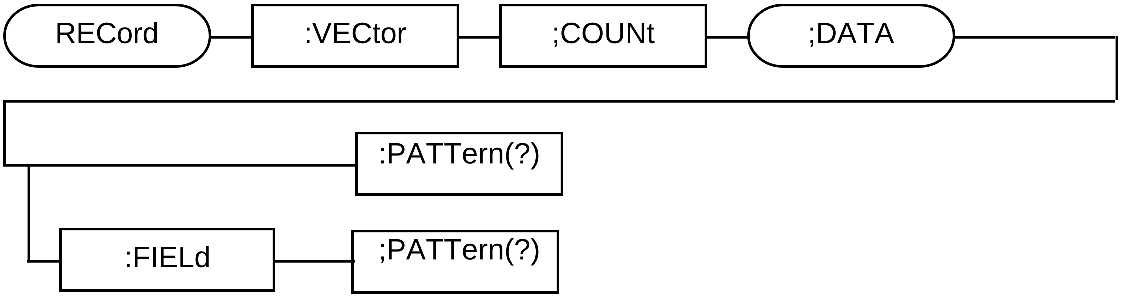
Parameter Definition **name** = Any alphanumeric string, and '_' (max 8 characters).

Examples

RECORD:FIELD?
ADDR

REC:FIEL?
ADDR

Loading/Querying Record Patterns (NON-SCPI)



The RECord:VECtor;COUNt;DATA:PATTern command loads expected response and/or don't care data vectors into the default response memory field. The default memory field is defined by the RECord:FIELd command. Valid field types for the RECord command are Expected (EXP), Dontcare (DON), Expected/Dontcare (ED), Algorithmic Expected (ALGE), Hardware Expected (HEXP), and Hardware Dontcare (HDON). Data will be loaded to the memory field starting at the vector location, specified by the VECtor parameter, and will load the number of vector words specified by the COUNt parameter. The data can be loaded to a destination field other than the default field by using the optional FIELd parameter. The RECord:VECtor;COUNt;DATA:PATTern? query command returns the data vectors from the default field.

Note

Data patterns cannot be loaded to RECord and HRECord type fields. The record memory can only be loaded by UUT input data or UUT compare results. The RECord and HRECord field data can be queried with the RECord:VECtor;COUNt;DATA:PATTern? command. Querying recorded data patterns is discussed in the section titled "Reading Recorded Data".

:VECtor <start_vector>

The initial vector location where data will start loading (or querying). The starting vector must be within the range of the size of the test (≤ test_size).

Parameter Definition

start_vector = (1 to test_size)

;COUNt <num_vectors | ALL>

The number of vector words that will be loaded to (or queried from) memory. The number of vectors can also be specified the literal string "ALL", where "ALL" is equal to the number of vectors from the starting vector location to the last vector in the test. The number of vectors to be loaded/queried must not exceed the last vector in the test. Example: A test is defined to be 100 vectors. The starting destination of memory to be loaded will be at vector location 50. The maximum number of vectors that can be loaded with the same command is 51, where num_vectors = (100-50) + 1 = 51.

Parameter Definition **num_vectors** = (1 to ((test_size-start_vector) + 1)

ALL = All vectors from the start_vector location to the last vector in the test.

;DATA

The DATA command string provides the command path to the PATTern parameter.

Parameter Definition none

:FIELd <name>

The optional FIELd parameter allows the data associated with the same command to be loaded to (or queried from) a destination field other than the default field. If the FIELd parameter option is used, then the FIELd and PATTern(?) parameters must be separated by a semicolon as shown in the example below.

Parameter Definition **name** = Any alphanumeric string and '_' (max 8 characters).

Note

The FIELd parameter changes the destination field only for the same command but **does not** change the default field.

:PATTern <data_value>{,data_value}

The data_value parameter is the actual data that will be loaded to the memory field. If no radix prefix (#h or #b) is used with the data values, then the data values must be entered in the radix format for the destination field as defined by the FIELd:NAME:RADix command. If the radix for the destination field is set to HEX, then data can be specified in hexadecimal format (the '#h' prefix is optional) or in binary format if the '#b' prefix is specified. Valid hexadecimal data values are '0' through 'F'. The hexadecimal 'X' character is valid only with Expected/Dontcare type fields (ED) and represents a don't care condition for that nibble (1 nibble=4 bits).

If the radix for the field is set to BIN, then data can be specified in binary format (the '#b' prefix is optional) or in hexadecimal format if the '#h' prefix is specified. Valid binary data values are '0', '1'. The binary 'X' character is valid only with Expected/Dontcare type fields (ED) and represents a don't care condition for the corresponding bit position. Leading '0' data characters may be omitted as shown in the examples below.

Parameter Definition **data_value** = [#h]{(0-F) | X} | [#b]{0 | 1 | X}

Note

The number of data_value elements must be equal to num_vectors. If a count mismatch occurs, the data will be loaded up to the number of data_value elements or the num_vectors, whichever is less. An error message will be generated.

Examples

RECORD:VECTOR 1;COUNT 4;DATA:FIELD ADDR;PATTERN #H00AA,
#H0055,#H00AA,#H0055
REC:VEC 1;COUN 4;DATA:PATT AA,55,AA,55
REC:VEC 1;COUN 4;DATA:PATT #B0000000010101010,
#B0000000001010101, #B0000000010101010,#B0000000001010101
REC:VEC 1;COUN 4;DATA:PATT #B10101010,
#B01010101,#B10101010,#B01010101

Note

All the commands shown above perform identical functions. The default field is ADDR and is set to HEX radix.

:PATTern?

The data_value parameter is the actual data that will be read from the stimulus memory field. The radix of data_value is determined by the FIELd:NAME:RADix command. If the radix for the field is set to HEX, then data will be returned in hexadecimal format with the '#h' prefix. Valid hexadecimal data values are '0' through 'F'. The hexadecimal 'X' character is valid only with Output/Tristate type fields (OT) and represents a tristate condition for that nibble (1 nibble=4 bits). The hexidecimal '?' character will be displayed when a nibble contains a combination of enabled and don't care expect pins.

If the radix for the field is set to BIN, then data will be returned in binary format with the '#b' prefix. Valid binary data values are '0', '1'. The binary 'X' character is valid only with Output/Tristate type fields (OT) and represents a tristate condition for the corresponding bit position. Each field defined can have a different radix format. Leading '0' data characters will be returned.

Response data_value{,data_value}

Parameter Definition **data_value** = <#h{(0-F) | X?}> | <#b{0 | 1 | X}>

Examples

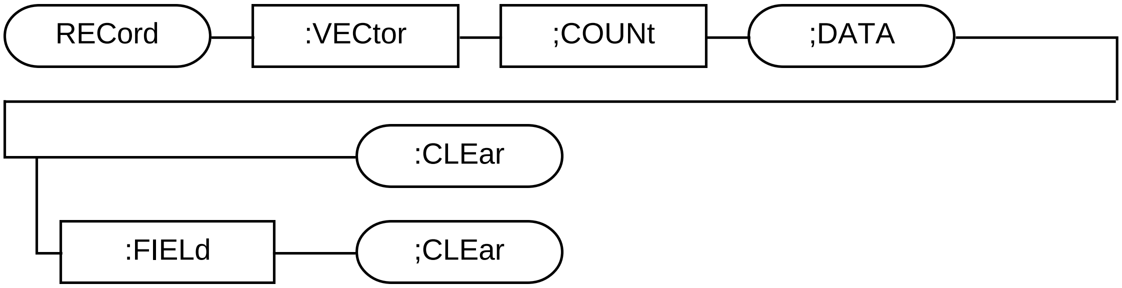
RECORD:VECTOR 1;COUNT 4;DATA:FIELD ADDR;PATTERN?
#h00AA,#h0055,#h00AA,#h0055

REC:VEC 1;COUN 4;DATA:PATT?
#h00AA,#h0055,#h00AA,#h0055

REC:VEC 1;COUN 2;DATA:PATT?
#b01X00110, #b0011X100

#h?6, #h3?

Clearing Record Patterns (NON-SCPI)



The RECord:VECtor;COUNt;DATA:CLEar command clears the response data pattern by loading "all zeros" (0's) into the default memory field. If the destination field is an expected type (EXP, ALGE, HEXP), the expected data pattern will be set to all zeros. If the destination field is a don't care type (DON, HTRI), the don't care pattern will be set to all enable compare condition (0's). If the destination field is an expected/ don't care type (ED), the expected data pattern will be set to all zeros and enable compare condition. The default memory field is defined by the RECord:FIELd command. Data will be cleared starting at the vector location, specified by the VECtor parameter, and will clear the number of vector words specified by the COUNt parameter. A destination field other than the default field can be cleared by using the optional FIELd parameter.

:VECtor <start_vector>

The initial vector location where data will be cleared. The starting vector must be within the range of the size of the test (≤ test_size).

Parameter Definition **start_vector** = (1 to test_size)

;COUNt <num_vectors>

The number of vector memory words that will be cleared. The number of vectors can also be specified the literal string "ALL", where "ALL" is equal to the number of vectors from the starting vector location to the last vector in the test. The number of vectors to be cleared must not exceed the last vector in the test. Example: A test is defined to be 100 vectors. The starting destination of memory to be cleared will be at vector location 50. The maximum number of vectors that can be cleared with the same command is 51, where num_vectors = (100-50) + 1 = 51.

Parameter Definition **num_vectors** = (1 to ((test_size-start_vector) + 1)

ALL = All vectors from the start_vector location to the last vector in the test.

;DATA

The DATA command string provides the command path to the CLEar parameter.

Parameter Definition none

:FIELd <name> The optional FIELd parameter allows a destination field other than the default field to be specified. The expected or don't care data in the alternate field will be cleared. If the FIELd parameter option is used, then the FIELd and CLEar parameters must be separated by a semicolon as shown in the example below.

Parameter Definition **name** = Any alphanumeric string and '_' (max 8 characters).

Note

The FIELd parameter changes the destination field only for the same command but **does not** change the default field.

:CLEar Causes the Expected and/or Don't care memories to be cleared.

Parameter Definition none

Examples

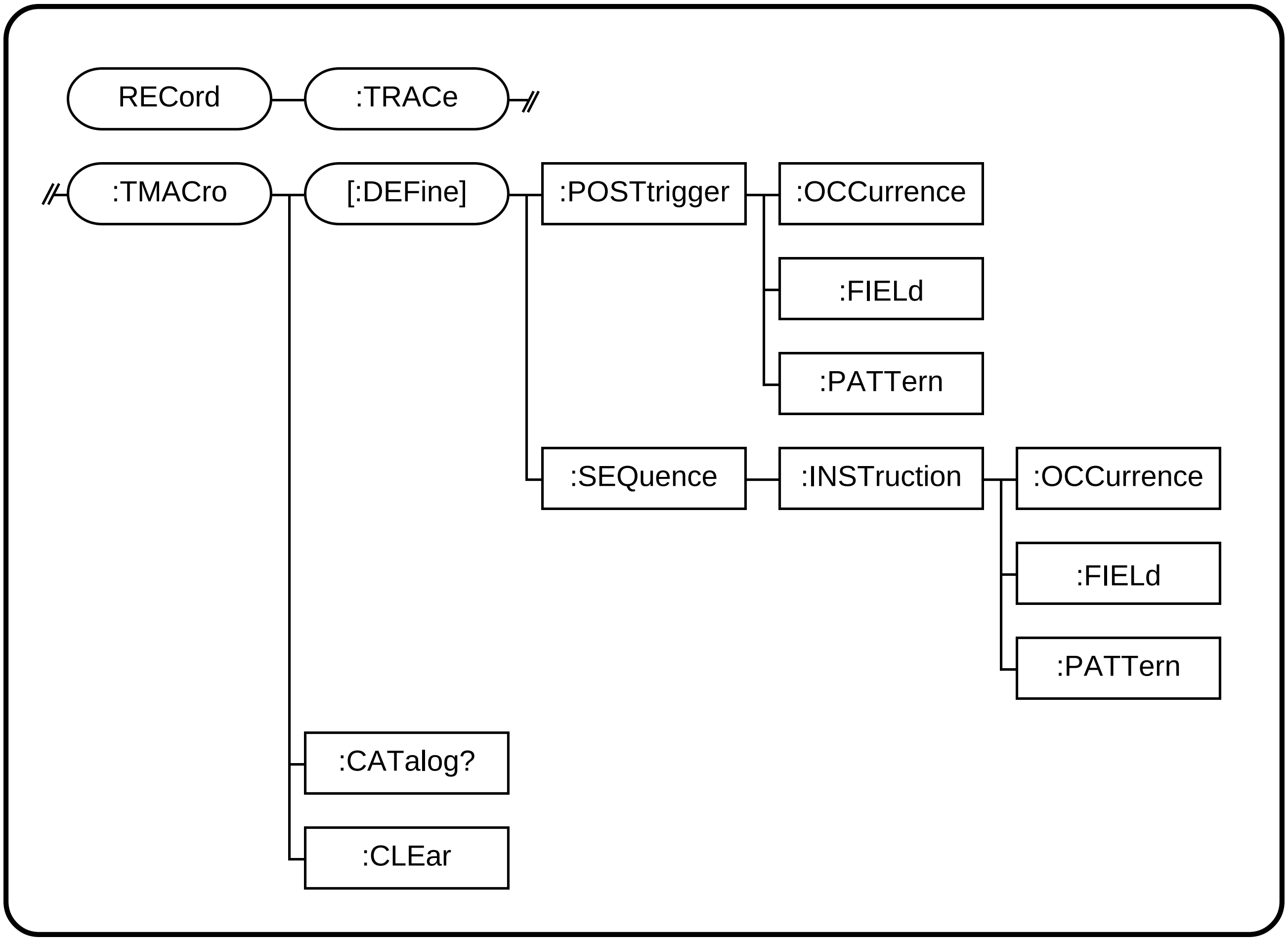
```
RECORD:VECTOR 1;COUNT 4;DATA:FIELD ADDR;CLEAR
REC:VEC 1;COUN 4;DATA:CLE
```

Trace TMACRO's

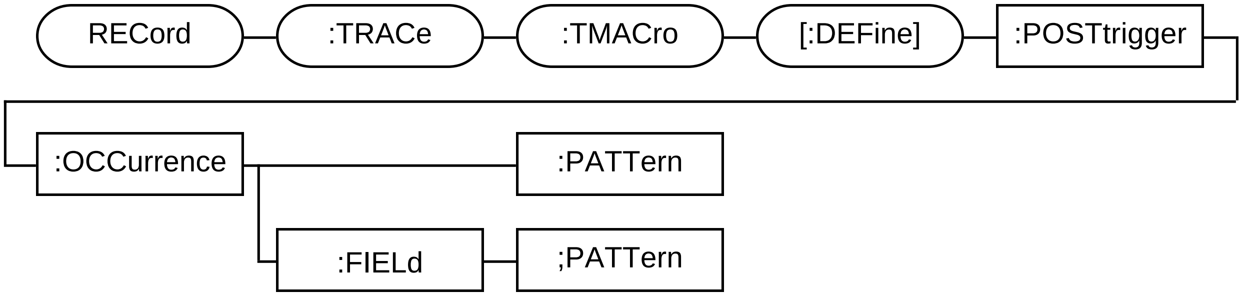
Trace TMACRO's provide a convenient method of triggering the SR2500 record logic for recording data, and are an alternative to record Trace Sequences. TMACRO's take two forms: Post Trigger and Sequence. Both TMACRO versions actually compile into Trace Sequence functions, so the three different approaches are mutually exclusive. Defining any one will overwrite any of the other previously defined trigger processes. The Trace Sequence is automatically set to Record Always for Post Trigger TMACRO's, meaning that even if a trigger pattern is not detected, data will be recorded to record memory. The Trace Sequence "WRAP" parameter is automatically set to on for both Post Trigger and Sequence TMACRO's.

The Post Trigger TMACRO is the simplest, and least flexible method to trigger data recording. Trigger patterns may be defined for a single ED type field. Using the Post-Trigger TMACRO method, you can specify a process where data will be recorded in a Pre Trigger - record all data up to the defined trigger pattern, Center Trigger - record data both before and after the defined trigger pattern, or Post Trigger - record all data after the defined trigger pattern, including the trigger pattern itself.

Sequence TMACRO's allow *Multiple Trigger Sequences* to be defined, with a single unique trigger pattern and action for each sequence level. Like the Post Trigger TMACRO's, Sequence TMACRO's only work with an ED field type, however, you may specify a different ED type field, and pattern, for each sequence level. Sequence TMACRO's are not as flexible as Trace Sequences, but are more flexible than Post Trigger TMACRO's.


RECord
:TRACe
:TMACro
[:DEFine]
:POSTtrigger
:OCCurrence
:FIELd
:PATTern
:SEQuence
:INSTruction
:OCCurrence
:FIELd
:PATTern
:CATalog?
:CLEar

Post Trigger TMACRO Definition (NON-SCPI)



The RECord:TRACe:TMACro:DEFine:POSTrigger command provides a simple method to trigger the record control logic for recording data to memory. Only three parameters need be defined to use the Post Trigger TMACRO - four if the default record field is not of type ED, or not the desired ED type field. All Post Trigger TMACRO parameters must be sent on the same command line. The TMACRO subsystem does not provide controls for using the CRC Signature Analyzer registers.

:TRACe

The TRACe command is a branch that connects the RECord and TMACro commands.

Parameter Definition none

:TMACro

The TMACro command is a branch that connects the TRACe command and either the optional DEFine or the POSTrigger commands.

Parameter Definition none

:DEFine

DEFine is the default branch that connect the TMACro and POSTrigger commands. Since it is the default path, DEFine may be omitted and POSTrigger used directly after TMACro.

Parameter Definition none

:POSTrigger <num_samples>

Defines the number of data samples to save to the record memory, after the defined number of trigger pattern matches (num_triggers) occur. It is possible to perform either Pre Trigger or Center Trigger functions just by changing the number of samples to record after the trigger pattern is detected. For example, if the test size is 1024 and the post trigger sample value is set to 0, then all samples, up to the one where the trigger pattern is detected, will be recorded. This is equivalent to a Pre Trigger. For center trigger, you would set the post trigger parameter to half of the defined test size, and for post trigger, set the parameter to one less than the test size.

Parameter Definition num_samples = (1 to test_size)

:OCCurrence <num_triggers>

Defines the number of trigger pattern matches that must be detected before recording the number of data samples defined by the POSTrigger num_samples parameter.

Parameter Definition num_triggers = (1-65535)

;FIELd <name>

The optional FIELd parameter allows the data associated with the command to be loaded to (or queried from) a destination field other than the default record subsystem field. If the FIELd parameter option is used, then the FIELd and PATTern(?) parameters must be separated by a semicolon instead of colons, as shown in the examples below. The only allowed field types for TMACRO's are type ED. If the default record field is not of type ED, or is not the desired ED type field, then the FIELd parameter **must** be used to specify an ED type field for defining the trigger pattern. Otherwise a command error will be generated.

Parameter Definition **name** = Any alphanumeric string and '_' (max 8 characters).

Note

The FIELd parameter changes the destination field only for the command in which it occurs, but **it does not** change the default field.

:PATTern <data_value>

The data_value parameter is the actual trigger pattern, and mask, that will be used to trigger the record control logic. If no radix prefix (#h or #b) is used with the data values, then the data values must be entered in the radix format defined for the destination ED field. The radix format for the destination field is defined by the FIELd:NAME:RADix command. If the radix for the destination field is set to HEX, then data can be specified in hexadecimal format (the '#h' prefix is optional) or in binary format if the '#b' prefix is specified. Valid hexadecimal data values are '0' through 'F'. For hexadecimal radix fields, the 'X' character represents a don't care condition for that nibble (1 nibble = 4 bits). If the radix for the field is set to BIN, then data can be specified in binary format (the '#b' prefix is optional) or in hexadecimal format if the '#h' prefix is specified. For binary radix fields, the 'X' character represents a don't care condition for the corresponding bit position. Leading '0' data characters may be omitted as shown in the examples below.

Parameter Definition **data_value** = [#h]{(0-F) | X} | #b{0 | 1 | X}

Note

For the following examples, the test size was set to 1024 vectors. The radix defined for field D15_00 is hex and the width is 16 bits.

The first example demonstrates a Pre Trigger record. Recording will start immediately and continue until the trigger pattern of hex AAAA is detected. If the record memory fills up before the trigger pattern is detected, then recording will wrap back to the beginning of memory and continue. Internal processes will always rearrange the record memory so that the oldest sample will be at vector 1, and the most recent sample at vector 1024. If the trigger pattern is never detected, then record memory will hold the last 1024 samples.

Example 1.

```
RECORD:TRACE:TMACRO:DEFINE:POSTRIGGER 1:OCCURRENCE
1;FIELD D15_00;PATTERN #hAAAA
```

The second example demonstrates a Center Trigger record. Recording will start immediately and continue until the trigger pattern of hex 55 is detected on the 8 LSB's. The state of the 8 MSB's are masked out with the 'X' characters. After the trigger pattern is detected an additional 512 samples will be taken, placing the trigger sample at the middle of the sample range. If the record memory fills up before the trigger pattern is detected, or before the additional 512 samples are recorded, then recording will wrap back to the beginning of memory and continue. Internal processes will always rearrange the record memory so that the oldest sample will be at vector 1, and the most recent sample at vector 1024. If the trigger pattern is detected before the record memory is half full, then the record memory will contain all record samples and the trigger sample will not be found in the middle of the recorded data, but at 512 samples before the end. If the SR2500 system completes its test after the trigger pattern is detected, but before the additional 512 samples are taken, then the location of the trigger sample is unknown, but will be somewhere in the last 512 samples. In this case, use the REC:DATA:SEARCH function to locate the trigger word. If the trigger pattern is never detected, then record memory will hold the last 1024 samples.

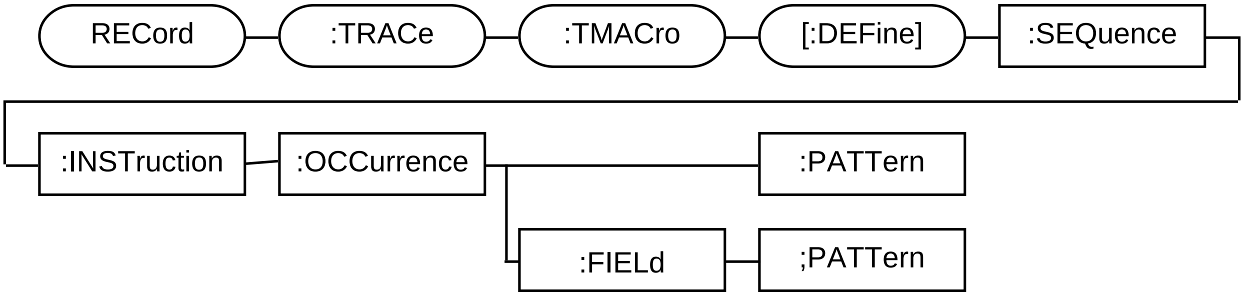
Example 2.

```
REC:TRAC:TMAC:DEF:POST 512:OCC 1;FIELD D15_00;PATTERN #hXX55
```

The last example demonstrates a Post Trigger record. Recording will start immediately and continue until the 5th occurrence of the binary trigger pattern '10' is detected on the 2 most significant bits. The state of the 14 LSB's are masked out with the 'X' characters. After the 5th time the trigger pattern is detected, an additional 1023 samples will be taken. If the record memory fills up before the 5th trigger pattern is detected, or before the additional 1023 samples are recorded, then recording will wrap back to the beginning of memory and continue. Internal processes will always rearrange the record memory so that the oldest sample will be at vector 1, and the most recent sample at vector 1024. If the SR2500 system completes its test after the 5th trigger pattern is detected, but before the additional 1023 samples are taken, then the record memory will hold all data samples and the location of the 5th trigger sample is unknown. In this case, use the REC:DATA:SEARCH function to locate each occurrence of the trigger word. If the trigger pattern was never detected, or detected less than 5 times, then the record memory will hold the last 1024 samples.

Example 3. REC:TRAC:TMAC:DEF:POST 1023:OCC 5;FIELD D15_00;PATTERN #b10XXXXXXXXXXXXXX

Sequence TMACRO Definition (NON-SCPI)



The RECord:TRACe:TMACro:DEFine:SEQuence command provides a more flexible method than the POSTrigger command to trigger the record control logic for recording data to memory. At each sequence level, one of three actions may be taken: Start recording, Continue to the next sequence level, or Stop recording. All Sequence TMACRO parameters, for the specified sequence level, must be sent on the same command line. The TMACRO subsystem does not provide controls for using the CRC Signature Analyzer registers.

:TRACe

The TRACe command is a branch that connects the RECord and TMACro commands.

Parameter Definition none

:TMACro

The TMACro command is a branch that connects the TRACe command and either the optional DEFine or the POSTrigger commands.

Parameter Definition none

:DEFine

DEFine is the default branch that connect the TMACro and POSTrigger commands. Since it is the default path, DEFine may be omitted and SEQuence used directly after TMACro.

Parameter Definition none

:SEQUENCE <num_sequence>

Specifies the sequence number, or level, that this occurrence of the command is defining.

Parameter Definition **num_sequence** = (1 to 8)

:INSTruction <STARt | STOP | CONTinue>

Instructs the record control logic to either start or stop recording samples to record memory upon detection of the trigger pattern defined for this sequence level, or to continue to the next sequence level after the trigger pattern is detected. The sequence level automatically advances to the next level if either STARt or STOP is the defined action and the trigger pattern was detected. Sequence execution will stop after the last defined sequence.

Parameter Definition

STARt = Start storing samples to record memory, including the trigger sample itself, when the trigger pattern specified for this sequence level is detected, then advance to the next sequence level.

STOP = Stop storing samples to record memory when the trigger pattern specified for this sequence level is detected, then advance to the next sequence level. Recording stops after the trigger sample itself is stored to memory.

CONTinue = Continue to the next sequence level after the defined trigger pattern is detected.

:OCCurrence <num_triggers>

Defines the number of trigger pattern matches that must be detected before taking the defined action.

Parameter Definition

num_triggers = (1-65535)

;FIELd <name>

The optional FIELd parameter allows the data associated with the command to be loaded to (or queried from) a destination field other than the default record subsystem field. If the FIELd parameter option is used, then the FIELd and PATTern(?) parameters must be separated by a semicolon instead of colons, as shown in the examples below. The only allowed field types for TMACRO's are type ED. If the default record field is not of type ED, or is not the desired ED type field, then the FIELd parameter **must** be used to specify an ED type field for defining the trigger pattern. Otherwise a command error will be generated.

Parameter Definition

name = Any alphanumeric string and '_' (max 8 characters).

Note

The FIELd parameter changes the destination field only for the command in which it occurs, but **it does not** change the default field.

:PATTern <data_value>

The data_value parameter is the actual trigger pattern, and mask, that will be used to trigger the record control logic. If no radix prefix (#h or #b) is used with the data values, then the data values must be entered in the radix format defined for the destination ED field. The radix format for the destination field is defined by the FIELd:NAME:RADix command. If the radix for the destination field is set to HEX, then data can be specified in hexadecimal format (the '#h' prefix is optional) or in binary format if the '#b' prefix is specified. Valid hexadecimal data values are '0' through 'F'. For hexadecimal radix fields, the 'X' character represents a don't care condition for that nibble (1 nibble = 4 bits). If the radix for the field is set to BIN, then data can be specified in binary format (the '#b' prefix is optional) or in hexadecimal format if the '#h' prefix is specified. For binary radix fields, the 'X' character represents a don't care condition for the corresponding bit position. Leading '0' data characters may be omitted as shown in the examples below.

Parameter Definition **data_value** = [#h]{(0-F) | X} | #b{0 | 1 | X}

Note

For the following example, the test size was set to 1024 vectors. The radix defined for field D15_00 is hex and the width is 16 bits. D15_00 is an ED type field.

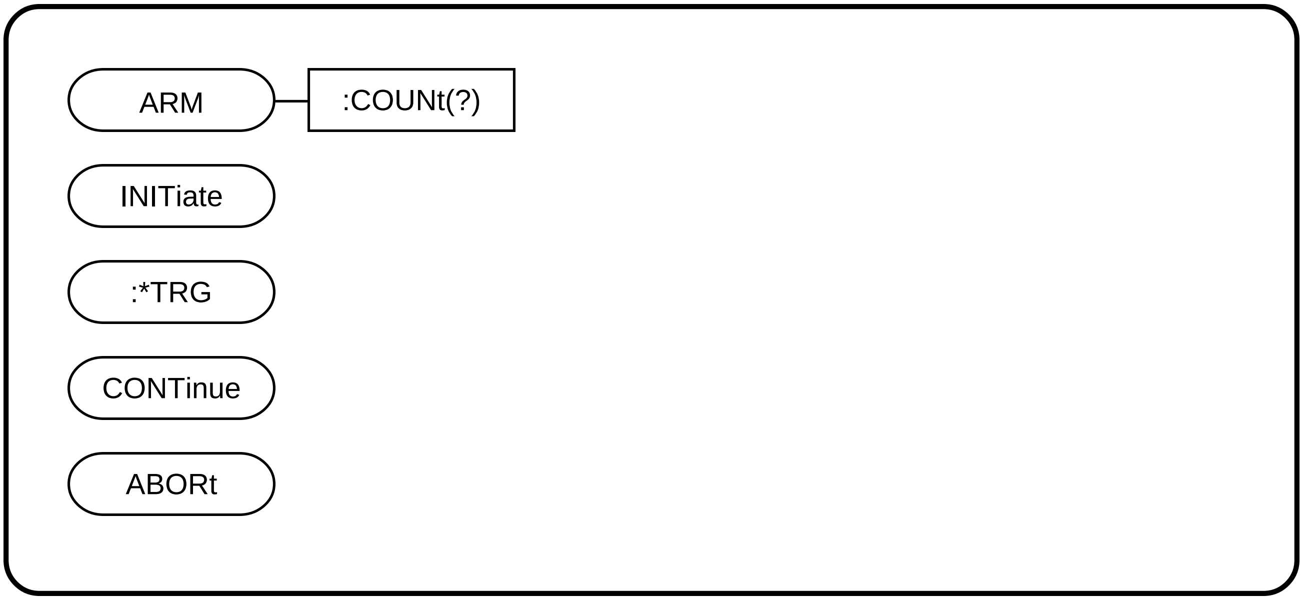
This example is similar to the TMACRO:POSTRIGGER examples and demonstrates a three step record process. Recording will start immediately and continue until the first trigger pattern of hex AAAA is detected, after which all recording will stop. Recording will remain suspended until the trigger pattern of hex 55 is detected on the 8 LSB's. The state of the 8 MSB's are masked out with the 'X' characters. After the second trigger pattern is detected, an additional 511 samples will be recorded. Recording will again be suspended, this time until the 5th occurrence of the binary trigger pattern of '10' is detected on the 2 MSB's. The state of the 14 LSB's are masked out with the 'X' characters. After this third trigger condition has been met, an additional 511 samples will be taken. If the record memory fills up before the first trigger pattern is detected, then recording will wrap back to the beginning of memory and continue. Internal processes will always rearrange the record memory so that the oldest sample will be at vector 1, and the most recent sample at vector 1024. If the SR2500 system completes its test after the first trigger pattern is detected, but before sequence 3/4 and 5/6 samples are taken, then the record memory will hold all data the most recent 1024 samples, and the location of the trigger samples, if any, are unknown. In this case, use the REC:DATA:SEARCH function to locate the trigger samples. If the first trigger pattern is never detected, then record memory will hold the last 1024 samples.

Examples

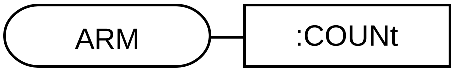
```
REC:TRAC:TMAC:DEF:SEQ 1:INST STAR:OCC 1;FIEL D15_00;PATT
#hXXXX
REC:TRAC:TMAC:DEF:SEQ 2:INST STOP:OCC 1;FIEL D15_00;PATT
#hAAAA
REC:TRAC:TMAC:DEF:SEQ 3:INST START:OCC 1;FIEL D15_00;PATT
#hXX55
REC:TRAC:TMAC:DEF:SEQ 4:INST STOP:OCC 510;FIEL D15_00;PATT
#hXXXX
REC:TRAC:TMAC:DEF:SEQ 5:INST START:OCC 1;FIEL D15_00;PATT
#b10XXXXXXXXXXXXXX
REC:TRAC:TMAC:DEF:SEQ 6:INST STOP:OCC 510;FIEL D15_00;PATT
#hXXXX
```

Run Time Commands

Run-time commands are used to control starting and stopping of SR2500 tests, as well as providing software Triggers and Continues. These commands act upon the currently active test program, as selected with the SYSTEM:TEST command. Only one test may be active and armed or running at any given time.



Setting the Arm Counter (SCPI 24.6)



The ARM:COUNt command sets the number of times the SR2500 will be armed. If the arm count is greater than 1, the SR2500 will return to the ARMED state when the test program execution is complete. The SR2500 is then armed and ready for another trigger to re-execute the test program. The arming sequence will repeat for the number of times equal to the arm count. When the final test execution is complete, the SR2500 will be placed in the IDLE state. The SR2500 can be placed in the IDLE state at any time by using the ABORt command.

:COUNt <arm_count>

Parameter Definition	**arm_count** = (1 - 1,000,000)
Default	1
Examples	ARM:COUNT 200 ARM:COUN 5

Initializing the Test Program (SCPI 24.4)

INITiate

The INITiate command compiles all the test program setup parameters and places the SR2500 in the ARMED state. Compiling may include becoming Bus Master in order to flush the temporary data cache in the SR5010 by writing the data to the appropriate I/O module. The SR2500 will then be ready and waiting for a trigger event (hardware signal or software command) to begin the test program execution. The SR2500 test program parameters ***cannot*** be modified or queried while in the ARMED or RUNNING state. If the Single Step mode is selected for the Clock Source, the INITiate command is used as the Single Step command to advance to the next vector.

Note

Software trigger commands include the IEEE 488.2 "*TRG" command. See Selecting the System Trigger Source (pg. 3-25) for details on the *TRG command.

Parameter Definition

none

Examples

INITIATE
INIT

Software Trigger Command (IEEE 488.2)

*TRG

The *TRG command will trigger the active test program when the trigger source is set to 'BUS' (see the TRIG:SYST:SOUR:BUS command). This command also requires that the INIT command be sent prior, and that the SR2500 is in the 'ARMED' state. Additional *TRG commands sent while the SR2500 is 'RUNNING', for the purpose of re-triggering the SR2500, will be ignored.

When used in conjunction with the WLoopuntil (Word Loop Until) and SLoopuntil (Start Loop Until) commands, the *TRG command allows a test program to continue past the loop. If the loop condition is set to loop until STRI = = TRUE, and the system trigger was defined as 'BUS', the test program will loop on a vector (WLoopuntil) or sequence of vectors (SLoopuntil) until the *TRG command is received. This feature is useful for halting or pausing a test program, yet keeping data and clocks alive.

Parameter Definition none

Examples *TRG

Test Program Abort Command (SCPI 24.5)

ABORt

The ABORt command asynchronously stops the test program in progress and places the SR2500 in the IDLE state. The test program can be ABORTed at any time regardless if the test program is in the ARMED state or RUNNING state. Unless the test is paused in a word loop, it is impossible to predict at which vector the test will actually halt. Once aborted, the test may only be restarted from the beginning.

Parameter Definition none

Examples

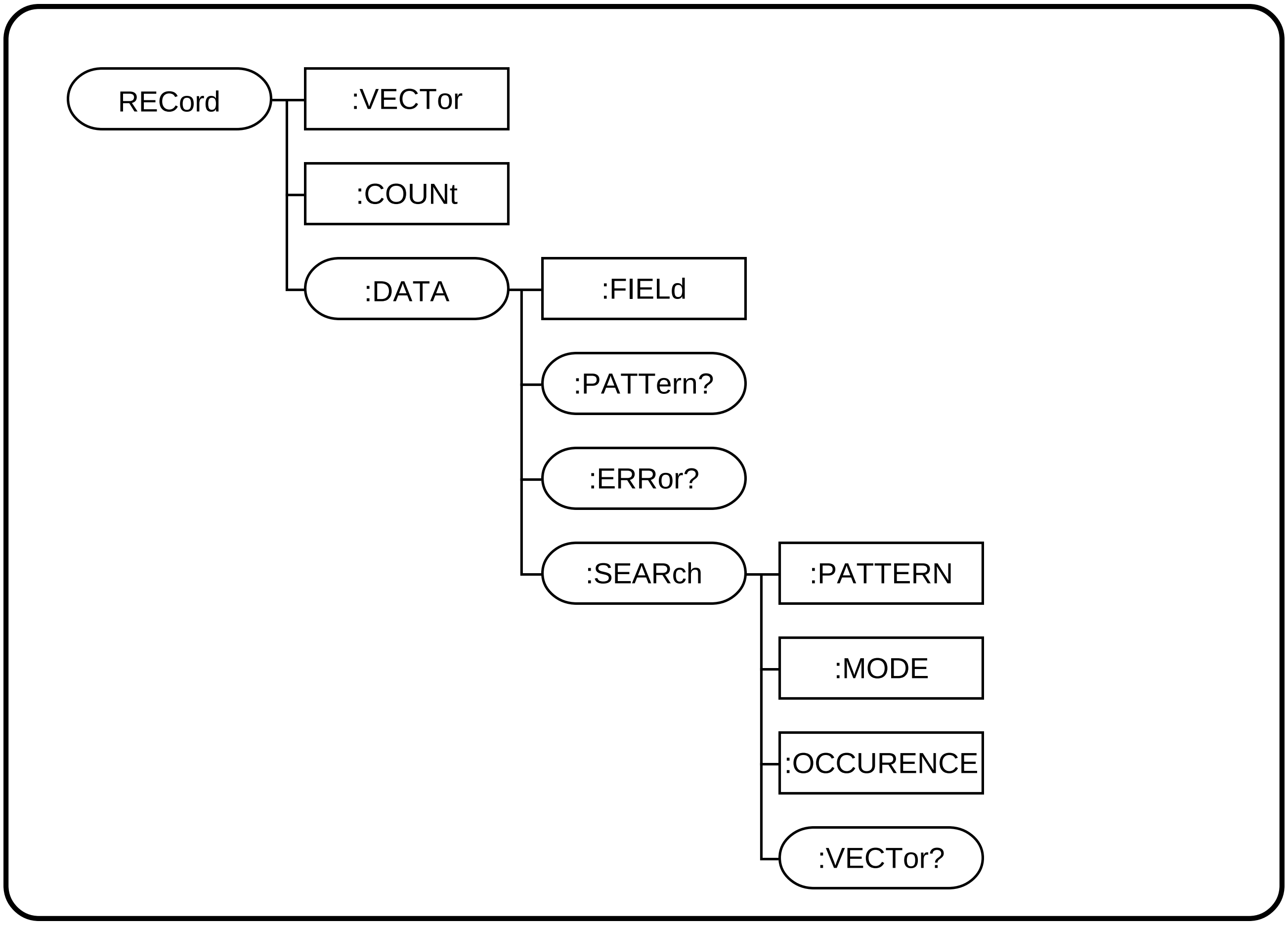
```
ABORT
ABOR
```

Reading Recorded Data

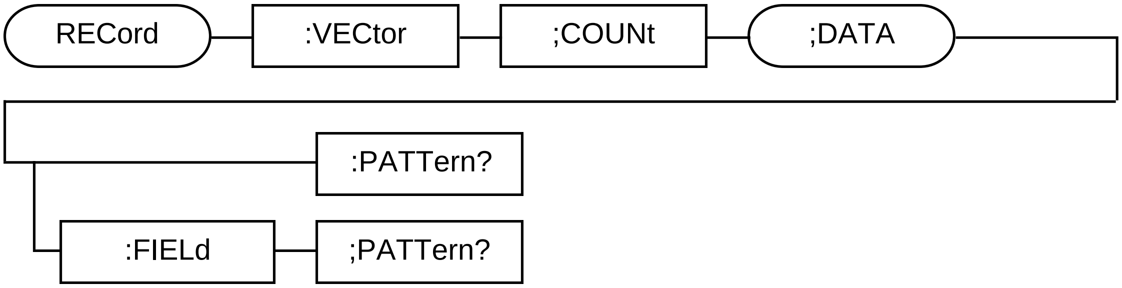
The Recorded Data commands provide access to record memory. Recorded data may be queried only when the SR2500 system has stopped, and only if data was recorded. You may use the TEST:NAME:STATUS? query command to query if, and how many, data samples were recorded. Recorded data may consist of the actual input patterns returned by the UUT, or the results of the real-time compare between the expected response and the data returned by the UUT. It is not possible to load data to any Record type field. For this reason, only the query version of the RECORD subsystem commands are described.

The RECORD:DATA:SEARCH command may be used to search a Record type field for specific data patterns using Equal-To, Not-Equal-To, Greater-Than and Less-Than search parameters. This is useful for finding a record trigger pattern when the exact sample vector location is unknown. It is also useful for locating record vectors where compare error conditions are recorded.

In addition to providing access to the record memory, the state of the Error Latch may be queried. This is a copy of the Error Latch flag returned by the TEST:NAME:STATUS? command and is provided as a more convenient method of determining pass/fail conditions.


RECord
:VECTor
:COUNt
:DATA
:FIELd
:PATTern?
:ERRor?
:SEARch
:PATTERN
:MODE
:OCCURENCE
:VECTor?

Reading Recorded Patterns (NON-SCPI)



The RECord:VECtor;COUNt;DATA:PATTern? query command returns the data vectors from Record memory fields. The data vectors returned will be either input data recorded from the Unit-Under-Test (UUT) or error results of the input data compared to the expected response data. The data vectors that are recorded, whether, input data or error data, is determined by the RECord:TRACe:SEQuence:FILTer command.

Valid field types for the RECord:;;:PATTern? command are Record (REC) and Hardware Record (HREC). Data will be queried from the Record memory field starting at the vector location, specified by the VECtor parameter, and will read the number of vector words specified by the COUNt parameter.

:VECtor <start_vector>

The initial vector location where data will start querying. The starting vector must be within the range of the size of the test (≤ test_size).

Parameter Definition

start_vector = (1 to test_size)

;COUNt <num_vectors | ALL>

The number of vector words that will be queried from Record memory. The number of vectors can also be specified by the literal string "ALL", where "ALL" is equal to the number of vectors from the starting vector location to the last vector in the test. The number of vectors to be queried must not exceed the last vector in the test.

Parameter Definition

num_vectors = (1 to ((test_size-start_vector) + 1)

ALL = All vectors from the start_vector location to the last vector in the test.

;DATA

The DATA command string provides the command path to the FIELd and PATTern parameters.

Parameter Definition

none

:FIELd <name>

The FIELd parameter specifies the record field that data patterns will be queried from. If the FIELd parameter is omitted, then the default memory field is assumed. The default memory field is defined by the RECord:FIELd command.

Parameter Definition

name = Any alphanumeric string and '_' (max 8 characters).

Note

The FIELd parameter changes the destination field only for the same command but **does not** change the default field.

:PATTern?

The data_value parameter is the actual data that will be read from the memory field. The radix of data_value is determined by the FIELd:NAME:RADix command. If the radix for the field is set to HEX, then data will be returned in hexadecimal format with the '#h' prefix. Valid hexadecimal data values are '0' through 'F'. The hexadecimal 'X' character represents a don't care condition for that nibble (1 nibble = 4 bits).

If the radix for the field is set to BIN, then data will be returned in binary format with the '#b' prefix. Valid binary data values are '0', '1'. The binary 'X' character represents a don't care condition for the corresponding bit position. Each field defined can have a different radix format. Leading '0' data characters will be returned.

Response

data_value{,data_value}

Parameter Definition

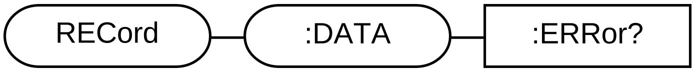
data_value = #h{(0-F) | X} | #b{0 | 1 | X}

Examples

RECORD:VECTOR 1;COUNT 4;DATA:FIELD ADDR;PATTERN?
#h7040,#h002C,#h0000,#h0130

REC:VEC 1;COUN 4;DATA:PATT?
#h7040,#h002C,#h0000,#h0130

Compare Error Status Query (NON-SCPI)



The RECord:DATA:ERRor? query command returns status of the error flag. The error status is a software flag that is "latched" when a compare error occurs. The status of the error flag is valid only when the SR2500 is in a STOPPED or IDLE state. The error status can be queried in the RUNNING state by using the TEST:NAME:STATus? command.

The condition of the error flag is set (ERRor = 1) each time a test is armed with the INITiate command. The Expect/Compare pipeline should be flushed and the error flag cleared at the beginning of each test in which the latched error flag will be used, otherwise the condition of the error flag will remain set and will not accurately portray the true status of the test execution. The STIMulus:CMACro (CLEARError) command is used to clear the condition of the error flag. Refer to the CLEAREerror macro command for additional information.

:DATA

The DATA command string provides the command path to the ERRor string.

Parameter Definition none

:ERRor?

Response 0 | 1

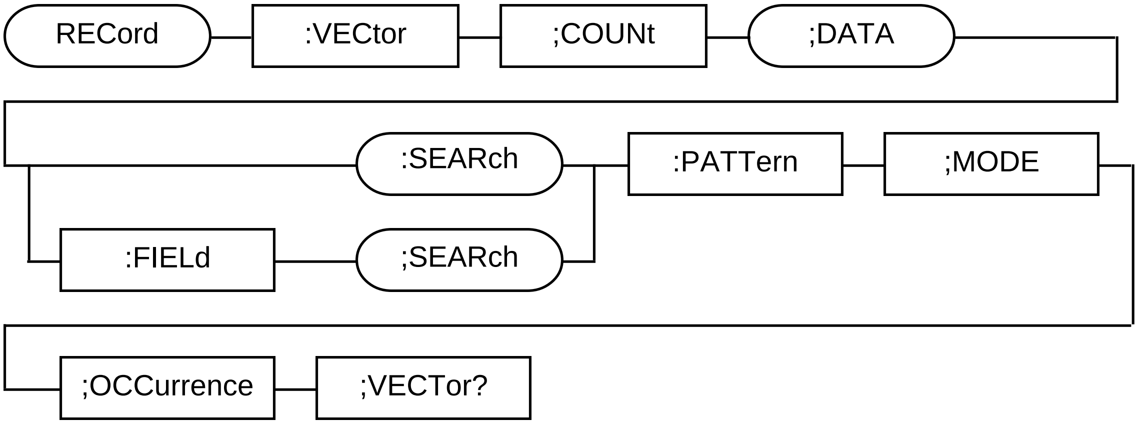
Parameter Definition **0** = No compare error has occurred

1 = A compare error has occurred.

Examples RECORD:DATA:ERROR?
1

REC:DATA:ERR?
0

Searching Record Memory (NON-SCPI)



The RECord:;;DATA:SEARch command searches through the Record Memory for specific pattern matches and returns the vector location and the matching data pattern. This command is useful for searching through the recorded input data for finding vector locations of compare errors or specific data patterns. The RECord:;;DATA:SEARch command will begin searching the record memory, for the specified field, starting at the vector location, specified by the VECtor parameter, and will search through the number of vector words specified by the COUNt parameter. A field other than the default field may be searched by using the optional FIELd parameter. The default memory field is defined by the RECord:FIELd command.

:VECtor <start_vector>

The initial vector location where the RECord:;;DATA:SEARch command will begin searching the record memory . The starting vector must be within the range of the size of the test.

Parameter Definition

start_vector = (1 to test_size)

;COUNt <num_vectors | ALL>

The number of vector words in the record memory that will be searched. The number of vectors can also be specified using the literal string "ALL", where "ALL" is equal to the number of vectors from the starting vector location to the last vector in the test. The number of vectors to be searched must not exceed the last vector in the test.

Parameter Definition

num_vectors = (1 to ((test_size - start_vector) + 1)

ALL = All vectors from the start_vector location to the last vector in the test.

;DATA

The DATA command string provides the command path to the FIELd and SEARch strings.

Parameter Definition none

:FIELd <name>

The FIELd parameter specifies the record memory field that will be searched for a pattern match. Valid field types for the FIELd parameter are Record (REC) and Hardware Record (HREC). If the FIELd parameter is omitted, then the default field is assumed. The default field is defined by the RECord:FIELd command.

Parameter Definition **name** = Any alphanumeric string and '_' (max 8 characters).

Note

The FIELd parameter changes the destination field only for this occurrence of the command, but **does not** change the default field.

;SEARch

The SEARch command string provides the command path to the PATTERN, MODE and OCCurrence parameters.

Parameter Definition none

:PATTern <data_pattern>

The data_pattern parameter is the data pattern that will be searched for in the record memory . If no radix prefix (#h or #b) is used with the data pattern, then the data pattern must be entered in the radix format defined for the record field being searched. The radix format for the record field is defined by the FIELd:NAME:RADix command. If the radix for the record field is set to HEX, then data pattern can be specified in hexadecimal format (the '#h' prefix is optional) or in binary format if the '#b' prefix is specified. Valid hexadecimal data values are '0' through 'F'.

If the radix for the field is set to BIN, then data pattern can be specified in binary format (the '#b' prefix is optional) or in hexadecimal format if the '#h' prefix is specified. Valid binary data values are '0' and '1'. Leading '0' data characters may be omitted as shown in the examples below.

Parameter Definition **data_pattern** = [#h](0-F) | [#b](0 | 1)

:MODE < EQ | NE | GT | LT >

The MODE parameter determines how the record memory vectors will be compared against the data_pattern.

Parameter Definition **EQ** = Compares the record memory for an "equal to" match of the data_pattern.

NE = Compares the record memory for a "not equal to" match of the data_pattern.

GT = Compares the record memory for a "greater than" match of the data_pattern.

LT = Compares the record memory for a "less than" match of the data_pattern.

;OCCurrence <num_match | ALL>

The number of data_pattern match occurrences to search for and return. The SEARch command will terminate the search function when the number of match occurrences has been met or when the last vector in the search range has been searched. This parameter allows the vector locations returned to be limited so as to avoid large data transfers. The number of vectors can also be specified with the literal string "ALL", where "ALL" will return all occurrences of data_pattern matches. The number of occurrences to be returned must not exceed the number of vectors to be searched.

Parameter Definition

num_match = (1 to num_vectors)

ALL = All occurrences of data_pattern matches.

:VECTor?

The VECTor? string terminates the command string and returns the matching vector locations and data patterns.

Response

match_vector, data_pattern{;match_vector, data_pattern}

Parameter Definition

match_vector = (start_vector to test_size)

data_pattern = #hXXXXXXXX, where X = (0 - F)

Examples

```
RECORD:VECTOR 1;COUNT 100;DATA:FIELD ADDR;SEARCH:PATTERN
#hAAAA;MODE EQ;OCCURRENCE ALL;VECTOR?
24,#h0000AAAA;65,#h0000AAAA;72,#h0000AAAA;90,#h0000AAAA
```

*The above command searched the "ADDR" field for **all** occurrences of data values **equal** to #hAAAA, starting at vector location 1 and searching the following 100 vector locations. The search command found four (4) matches at vector locations 24, 65, 72, and 90.*

```
REC:COUN ALL;DATA:SEAR:PATT #h0000;MODE GT;OCC 2;VECT?
1,#h00005B2C;4,#h00002F2A
```

*The above command searched all memory locations of the default field for the **first 2** occurrences of data values **greater than** #h0000. The search command found matches at vector locations 1 and 4 and terminated the search after the first 2 occurrences.*

(This Page Intentionally Left Blank)

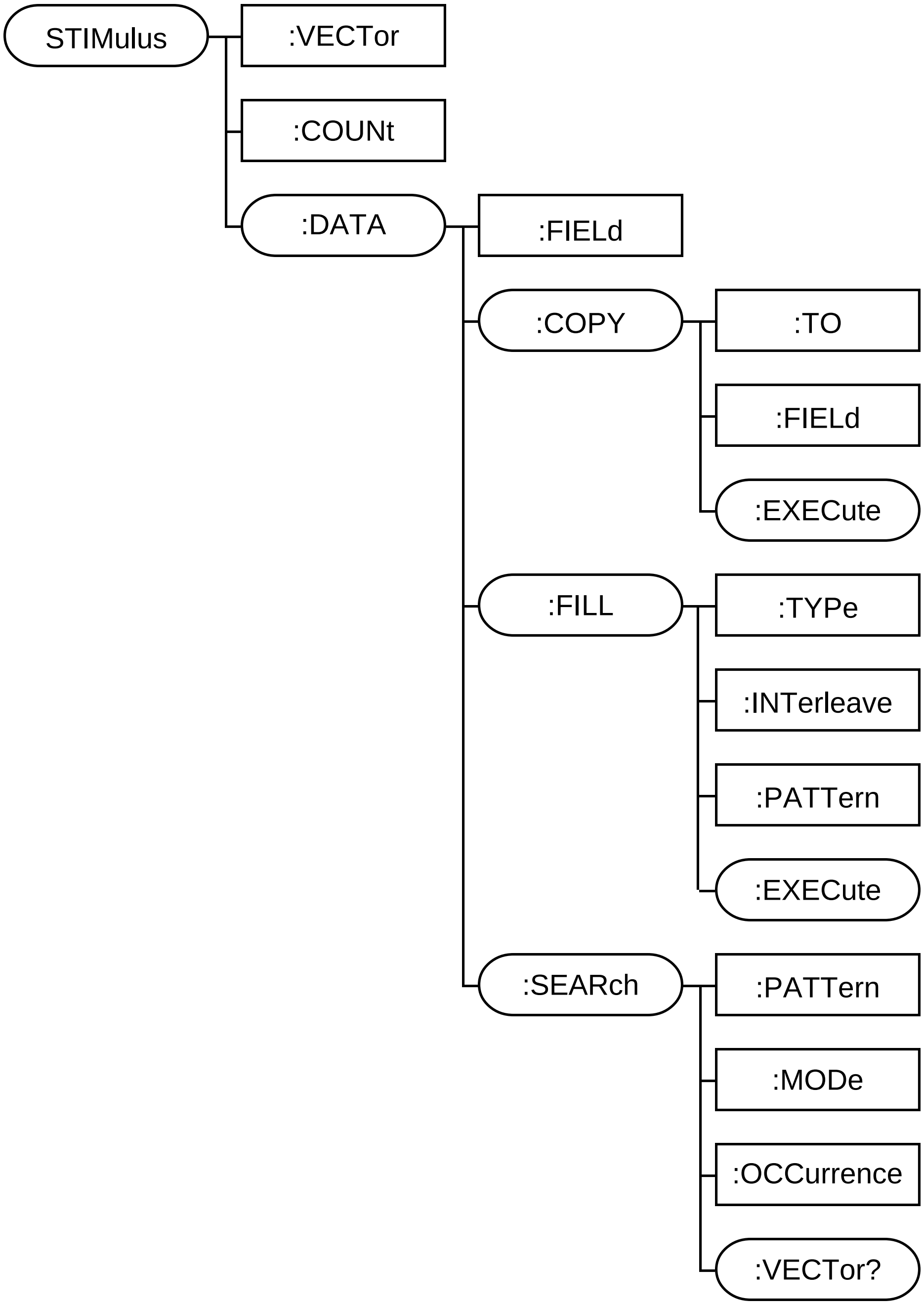
Advanced Programming

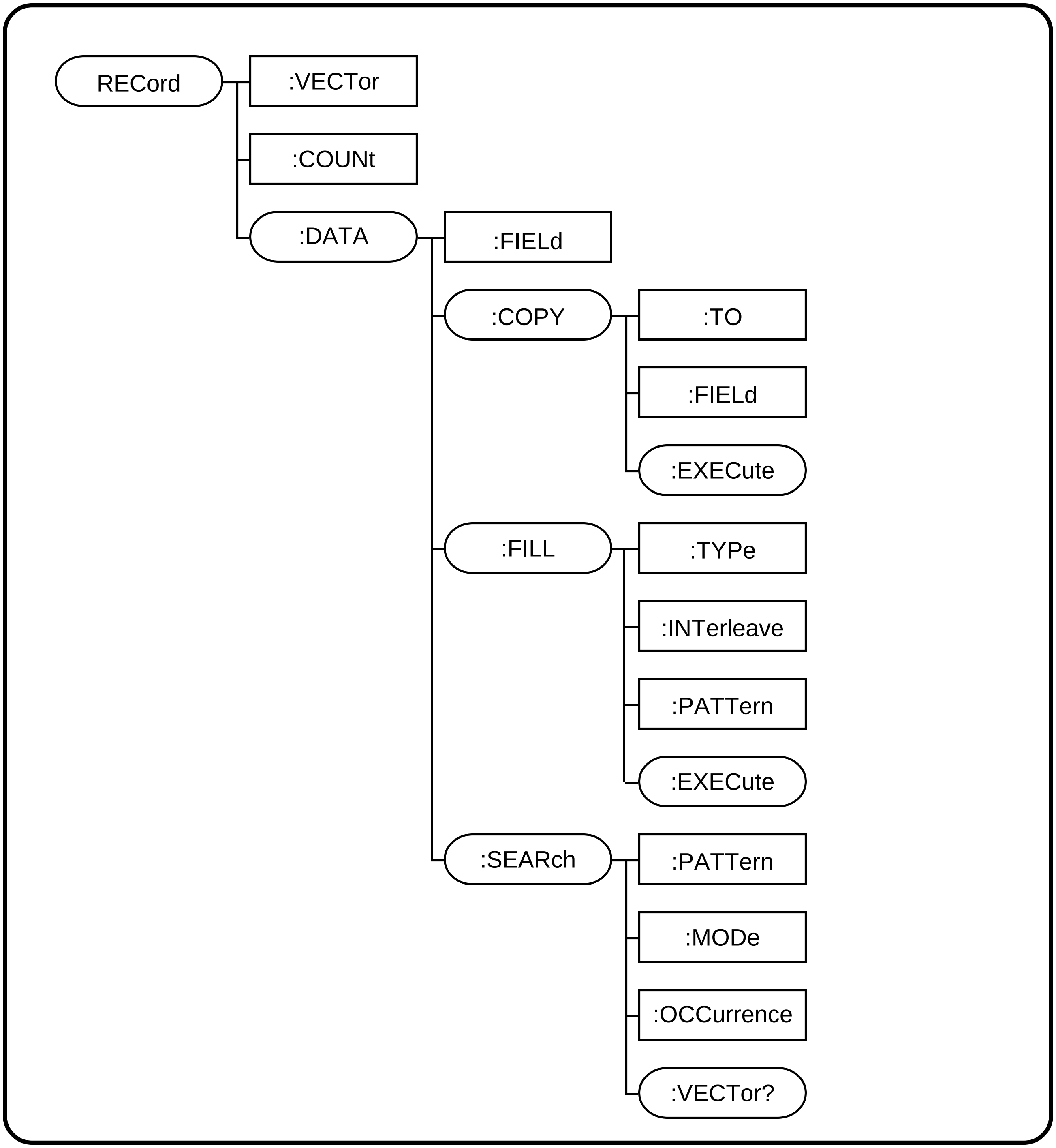
This section includes the more advanced commands required for editing, filling and copying Stimulus/Response patterns, generating Algorithmic Stimulus/Response patterns, defining output data formatting and input sampling, and using the high speed binary pattern load/query and learn functions. This section is divided into the following minor sections:

- ***Pattern Editing* .... *pg 3-102***
- ***I/O Formatting and Timing* .... *pg 3-126***
- ***Algorithmic Pattern Generation* .... *pg 3-142***
- ***High Speed Binary Pattern Transfers* .... *pg 3-170***
- ***Saving and Loading Tests* .... *pg 3-180***
- ***Advanced Record Triggering* .... *pg 3-184***

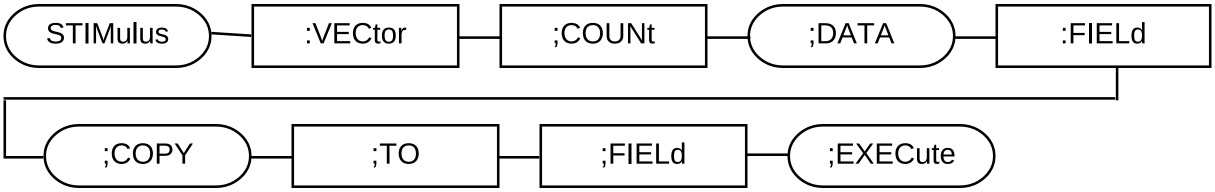
Pattern Editing

The SR2500 has pattern editing functions built into the operating system, which is a convenient, high level means to load or query Stimulus and Response memories. Three functions are provided, the ability to fill a specified field with one of several data patterns, the ability to copy data patterns from one field to another field of the same or different type, and the ability to search a field for specified data patterns. Each of these functions is available in both the STIMULUS and RECORD subsystems. Loading or copying data to a Record type field is prohibited, however, a Record type field may be used as the source field when copying data from one field to another.


STIMulus
:VECTor
:COUNt
:DATA
:FIELd
:COPY
:TO
:FIELd
:EXECute
:FILL
:TYPe
:INTerleave
:PATTern
:EXECute
:SEARch
:PATTern
:MODe
:OCCurrence
:VECTor?


RECord
:VECTor
:COUNt
:DATA
:FIELd
:COPY
:TO
:FIELd
:EXECute
:FILL
:TYPe
:INTerleave
:PATTern
:EXECute
:SEARch
:PATTern
:MODe
:OCCurrence
:VECTor?

Copy Stimulus Patterns (NON-SCPI)



The STIMulus:;;:;COPY command copies output and/or tristate data vectors from a source memory field into a destination memory field. Data will be copied from the source memory field starting at the vector location, specified by the VECtor parameter, and will copy the number of vector words specified by the COUNt parameter. The source field may be any valid Stimulus Field type. The destination field can be Stimulus Field or Record Field type with the exception of the RECord and HRECord Field types. RECord and HRECord fields are "read only" type and can only be written to by the UUT.

:VECtor <source_vector>

The initial vector location in the source memory field where data will be copied from. The starting vector must be within the range of the size of the test (≤ test_size).

Parameter Definition

source_vector = (1 to test_size)

;COUNt <num_vectors | ALL>

The number of vector words that will be copied from the source field to the destination field. The number of vectors can also be specified the literal string "ALL", where "ALL" is equal to the number of vectors from the starting vector location to the last vector in the test. The number of vectors to be copied must not exceed the last vector in the test.

Parameter Definition

num_vectors = (1 to ((test_size-start_vector) + 1)

ALL = All vectors from the start_vector location to the last vector in the test.

;DATA

The DATA command string provides the command path to the FIELd and COPY strings.

Parameter Definition

none

:FIELd <source_name>

The FIELd parameter specifies the stimulus source memory field that data patterns will be copied from. Valid field types for the source FIELd parameter are Output (OUT), Tristate (TRI), Output/Tristate (OT), Algorithmic Output (ALGO), Hardware Output, (HOUT), and Hardware Tristate (HTRI). If the FIELd parameter is omitted, then the default memory field is assumed. The default memory field is defined by the STIMulus:FIELd command.

Parameter Definition **source_name** = Any alphanumeric string and '_' (max 8 characters).

;COPY

The COPY command string provides the command path to TO, FIELd, and EXECute.

Parameter Definition none

:TO <dest_vector>

The initial vector location in the destination memory field where data will be copied to. The destination starting vector must be within the range of the size of the test (≤ test_size).

Parameter Definition **dest_vector** = (1 to test_size)

;FIELd <dest_name>

The FIELd parameter specifies the destination memory field that data patterns will be copied to. All Stimulus and Record field types are valid destination fields, except the REC and HREC field types. RECord fields are "read only" type and can only be written to by sampling the UUT response. Stimulus field types are Output (OUT), Tristate (TRI), Output/ Tristate (OT), Algorithmic Output (ALGO), Hardware Output, (HOUT), and Hardware Tristate (HTRI). Record Field types include Expected (EXP), DontCare (DON), Expected/Dontcare (ED), Algorithmic Expected (ALGE), Hardware Expected (HEXP), and Hardware DontCare (HDON). If the FIELd parameter is omitted, then the default memory field is assumed. The default memory field is defined by the STIMulus:FIELd command.

Parameter Definition **dest_name** = Any alphanumeric string and '_' (max 8 characters).

;EXECute

EXECute terminates the command string and executes the memory COPY command.

Parameter Definition none

Examples

```
STIMULUS:VECTOR 1;COUNT 100;DATA:FIELD ADDR;COPY:TO
200;FIELD ADDR;EXECUTE
```

This command copies 100 data words from vectors 1 - 100 to vectors 200-299. The source and destination field is the "ADDR" field.

```
STIM:VECT 50;COUN 10;DATA:COPY:TO 60;EXECUTE
```

This command copies 10 data words from vectors 50 - 59 to vectors 60 - 69. The source and destination field are the default field as defined by the STIMulus:FIELd command.

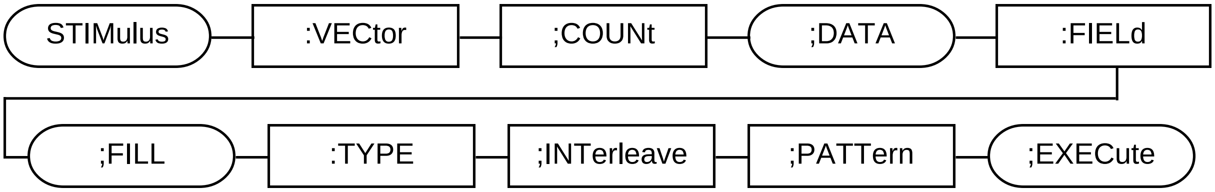
```
STIMULUS:VECTOR 1;COUNT ALL;DATA:FIELD ADDR;COPY:TO
1;FIELD DATA;EXECUTE
```

This command copies all data vectors from the "ADDR" field to the "DATA" field.

STIM:VECT 1;COUN 10;DATA:COPY:TO 11;EXEC;TO 21;EXEC;TO 31;EXEC;TO 41;EXEC

This command defines a block of 10 data words from vectors 1 - 10. This 10 vector block pattern is copied repetitively to vectors 11 - 20, 21 - 30, 31 - 40, and 41 - 50. The source and destination field are the default field as defined by the STIMulus:FIELd command.

Filling Stimulus Memory (NON-SCPI)



The STIMulus:;;:;FILL command loads the output and tristate memories with pre-defined pattern sequences. These pre-defined patterns load the data memory with commonly used data patterns without downloading a large amount of vectors from the Slot 0 Controller. This feature reduces the amount of data pattern programming and minimizes the test program download time. Pattern sequences include Repeat, Increment, Decrement, Complement, Alternate, Walking "1", Walking "0", and Pseudo-Random patterns. Data patterns will be loaded starting at the vector location, specified by the VECtor parameter, and will load the number of vector words specified by the COUNt parameter. The destination field may be any valid Stimulus Field type.

Note

The STIMulus:;;:;FILL command should not be confused with the Algorithmic Command Macros. The STIMulus:;;:;FILL command loads stimulus memory with data vectors. The Algorithmic Command Macros change the output pattern "on-the-fly" during run-time.

:VECtor <start_vector>

The initial vector location in the destination field where data will be start loading. The starting vector must be within the range of the size of the test (≤ test_size).

Parameter Definition

start_vector = (1 to test_size)

;COUNt <num_vectors | ALL>

The number of vector words that will be loaded to memory. The number of vectors can also be specified by the literal string "ALL", where "ALL" is equal to the number of vectors from the starting vector location to the last vector in the test. The number of vectors to be loaded must not exceed the last vector in the test.

Parameter Definition

num_vectors = (1 to ((test_size-start_vector) + 1)

ALL = All vectors from the start_vector location to the last vector in the test.

;DATA

The DATA command string provides the command path to the FIELd and FILL strings.

Parameter Definition none

:FIELd <name>

The FIELd parameter specifies the destination field where data patterns will be loaded to. Valid field types for the source FIELd parameter are Output (OUT), Tristate (TRI), Output/Tristate (OT), Algorithmic Output (ALGO), Hardware Output, (HOUT), and Hardware Tristate (HTRI). If the FIELd parameter is omitted, then the default memory field is assumed. The default memory field is defined by the STIMulus:FIELd command.

Parameter Definition **name** = Any alphanumeric string and '_' (max 8 characters).

;FILL

The FILL command string provides the command path to TYPE, INTerleave, PATTern, and EXECute strings.

Parameter Definition none

:TYPE <REPeat | INCrement | DECrement | COMplement | ALTernate | WLK1 | WLK0 | RANdom>

The type of pattern sequence that will be loaded to the destination memory field.

Parameter Definition **REPeat** = The REPeat parameter fills the memory repetitively with the same data pattern. The repeating data pattern is defined by the PATTern parameter.

INCrement = The INCrement parameter fills the memory with an incrementing data pattern. The initial data value that will begin incrementing is defined by the PATTern parameter.

DECrement = The DECrement parameter fills the memory with an decrementing data pattern. The initial data value that will begin decrementing is defined by the PATTern parameter.

COMplement = The COMplement parameter complements the current data value at each vector location. The PATTern parameter is not required and has no affect on the pattern fill command.

ALTernate = The ALTernate parameter fills the memory with an alternating data pattern. The initial data value that will begin alternating is defined by the PATTern parameter.

WLK1 = The WLK1 parameter fills the memory with a walking "1" data pattern. The "1" pattern will "walk" from LSB to MSB, i.e. #h0001, #h0002, #h0004, #h0008, etc.. The initial bit position that will begin walking is defined by the PATTern parameter. The WLK1 parameter will select the least significant "1" bit position in the PATTern parameter to begin the walking "1" pattern. For example, a initial PATTern data value of #h00F4 will begin walking from bit position 3, since the least significant "1" in #h00F4 is in the 3rd bit position from the LSB. All other "more significant 1s" will be ignored. Therefore, the walking "1" pattern will begin with #h0004 followed by #h0008, #h0010, etc. A PATTern parameter data value of #h0000 will cause all data values to be set to #h0000 and the walking "1" function will not be performed.

WLK0 = The WLK0 parameter fills the memory with a walking "0" data pattern. The "0" pattern will "walk" from LSB to MSB, i.e. #hFFFE, #hFFFD, #hFFFB, #hFFF7, etc.. The initial bit position that will begin walking is defined by the PATTern parameter. The WLK0 parameter will select the least significant "0" bit position in the PATTern parameter to begin the walking "0" pattern. For example, a initial PATTern data value of #h00F3 will begin walking from bit position 3, since the least significant "0" in #h00F3 is in the 3rd bit position from the LSB. All other "more significant 0s" will be ignored. Therefore, the walking "0" pattern will begin with #hFFFB followed by #hFFF7, #hFFEF, etc.

RANdom = The RANdom parameter fills the memory with a pseudo-random data pattern. The seed value that is used to initialize the pseudo-random calculation is determined by the PATTern parameter.

;INTerleave <int_count>

The INTerleave parameter specifies the interval count of the data vector locations to be "filled". For example, if the interleave count is set to two (2), then every other vector will be loaded with the fill function. Likewise, if the interleave count is set to ten (10), then every tenth vector will be loaded with the fill function. The default value for int_count is 1.

A powerful use of the INTerleave parameter is for loading complex data patterns to a multiplexed bus. An application example would be in the case of a multiplexed address/data bus. By setting the int_count to a value of two (2), the address bus memory can be loaded with an incrementing pattern command while the data bus can be loaded with a "checkerboard" pattern using the alternating pattern command.

Another use of the INTerleave parameter is for generating alternating tristated output vectors. Again by setting the int_count to a value of two (2), the tristate field can be loaded with an alternating ones and zeros. This will allow the UUT to alternate between read and write cycles on a bi-directional data bus. Another use for the INTerleave parameter is for loading an initial (or reset) data value to repetitive vector locations.

Parameter Definition **int_count** = (1 - 10)

;PATTern <init_patt>

The PATTern parameter sets the initial data value for the fill function. Refer to each FILL TYPE command for details on the fill function performed on the initial data value. The default init_pattern is #h0.

Parameter Definition **init_patt** = #h{(0-F)} | #b{0 | 1}

;EXECute

EXECute terminates the command string and executes the memory FILL command.

Parameter Definition

none

Examples

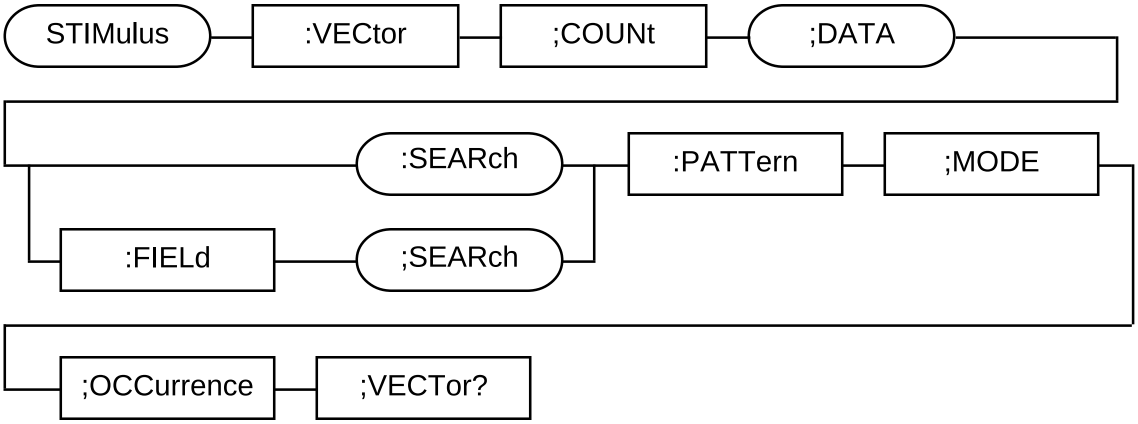
STIMULUS:VECTOR 1;COUNT 100;DATA:FIELD ADDR;FILL:TYPE INCREMENT;INTERLEAVE 1;PATTERN #H0000;EXECUTE

This command fills 100 data words, starting at vector location 1, with an incrementing data pattern starting with an initial data value of #h0000.

STIM:VECT 1;COUN ALL;DATA:FILL:TYPE ALT;PATT #HAAAA;EXECUTE

This command fills all vector locations with an alternating data pattern of #hAAAA and #h5555.

Searching Stimulus Memory (NON-SCPI)



The STIMulus:;;DATA:SEARch command searches through the Output and Tristate memories for specific pattern matches and returns the vector location and the matching data pattern. This command is useful for searching through the output and tristate memories for editing data patterns. The STIMulus:;;DATA:SEARch command will begin searching the specified memory field, starting at the vector location, specified by the VECtor parameter, and will search through the number of vector words specified by the COUNt parameter. A field other than the default field may be searched by using the optional FIELd parameter. The default memory field is defined by the STIMulus:FIELd command.

:VECtor <start_vector>

The initial vector location where the RECord:;;DATA:SEARch command will begin searching the record memory . The starting vector must be within the range of the size of the test.

Parameter Definition **start_vector** = (1 to test_size)

;COUNt <num_vectors | ALL>

The number of vector words in the record memory that will be searched. The number of vectors can also be specified using the literal string "ALL", where "ALL" is equal to the number of vectors from the starting vector location to the last vector in the test. The number of vectors to be searched must not exceed the last vector in the test.

Parameter Definition **num_vectors** = (1 to ((test_size - start_vector) + 1)

ALL = All vectors from the start_vector location to the last vector in the test.

;DATA

The DATA command string provides the command path to the FIELd and SEARch strings.

Parameter Definition none

:FIELd <name>

The FIELd parameter specifies the record memory field that will be searched for a pattern match. Valid field types for the FIELd parameter are Output (OUT), Tristate (TRI), Algorithmic Output (ALGO), Hardware Output, (HOUT), and Hardware Tristate (HTRI). The Output/ Tristate (OT) field type cannot be searched since OT fields consist of a combination of the Output and Tristate memories, and only one memory can be searched at a time. If the FIELd parameter is omitted, then the default field is assumed. The default field is defined by the STIMulus:FIELd command.

Parameter Definition

name = Any alphanumeric string and '_' (max 8 characters).

Note

The FIELd parameter changes the destination field only for this occurrence of the command, but **does not** change the default field.

;SEARch

The SEARch command string provides the command path to the PATTERN, MODE and OCCurrence parameters.

Parameter Definition

none

:PATTern <data_pattern>

The data_pattern parameter is the data pattern that will be searched for in the record memory . If no radix prefix (#h or #b) is used with the data pattern, then the data pattern must be entered in the radix format defined for the record field being searched. The radix format for the record field is defined by the FIELd:NAME:RADix command. If the radix for the record field is set to HEX, then data pattern can be specified in hexadecimal format (the '#h' prefix is optional) or in binary format if the '#b' prefix is specified. Valid hexadecimal data values are '0' through 'F'. If the radix for the field is set to BIN, then data pattern can be specified in binary format (the '#b' prefix is optional) or in hexadecimal format if the '#h' prefix is specified. Valid binary data values are '0' and '1'. Leading '0' data characters may be omitted as shown in the examples below.

Parameter Definition

data_pattern = [#h](0-F) | [#b](0 | 1)

:MODE < EQ | NE | GT | LT >

The MODE parameter determines how the record memory vectors will be compared against the data_pattern.

Parameter Definition

EQ = Compares the record memory for an "equal to" match of the data_pattern.

NE = Compares the record memory for a "not equal to" match of the data_pattern.

GT = Compares the record memory for a "greater than" match of the data_pattern.

LT = Compares the record memory for a "less than" match of the data_pattern.

;OCCurrence <num_match | ALL>

The number of data_pattern match occurrences to search for and return. The SEARch command will terminate the search function when the number of match occurrences has been met or when the last vector in the search range has been searched. This parameter allows the vector locations returned to be limited so as to avoid large data transfers. The number of vectors can also be specified the literal string "ALL", where "ALL" will return all occurrences of data_pattern matches. The number of occurrences to be returned must not exceed the number of vectors to be searched.

Parameter Definition **num_match** = (1 to num_vectors)

ALL = All occurrences of data_pattern matches.

:VECTor?

The VECTor? string terminates the command string and returns the matching vector locations and data patterns.

Response match_vector, data_pattern{;match_vector, data_pattern}

Parameter Definition **match_vector** = (start_vector to test_size)

data_pattern = #hXXXXXXXX, where X = (0 - F)

Examples

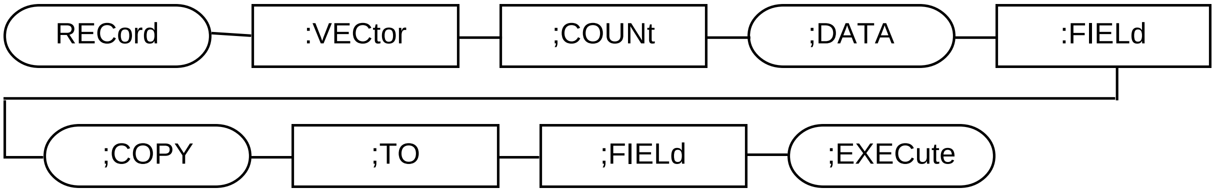
```
STIMULUS:VECTOR 1;COUNT 100;DATA:FIELD ADDR;
SEARCH:PATTERN #hAAAA;MODE EQ;OCCURRENCE ALL;VECTOR?
24,#h0000AAAA;65,#h0000AAAA;72,#h0000AAAA;90,#h0000AAAA
```

The above command searched the "ADDR" field for ***all*** *occurrences of data values* ***equal*** *to #hAAAA, starting at vector location 1 and searching the following 100 vector locations. The search command found four (4) matches at vector locations 24, 65, 72, and 90.*

```
STIM:COUN ALL;DATA:SEAR:PATT #h0000;MODE GT;OCC 2;VECT?
1,#h00005B2C;4,#h00002F2A
```

The above command searched all memory locations of the default field for the ***first 2*** *occurrences of data values* ***greater than*** *#h0000. The search command found matches at vector locations 1 and 4 and terminated the search after the first 2 occurrences.*

Copying Record and Response Patterns (NON-SCPI)



The RECord:;;:;COPY command copies expected response, don't care, and/or record data vectors from a source memory field into a destination memory field. The RECord:;;:;COPY command is very useful for copying Unit-Under-Test (UUT) response data from a "known good UUT" to expected response memory. This is accomplished by taking record data that is stored in a RECord field and copying it to an EXPected response field. Unknown UUTs can then be tested against reference data patterns from the "known good UUT". This is a common method of developing test vectors if a "known good UUT" is available.

Data will be copied from the source memory field starting at the vector location, specified by the VECtor parameter, and will copy the number of vector words specified by the COUNt parameter. The source field may be any valid Record Field type. The destination field can be any Stimulus Field or Record Field type with the exception of the RECord and HRECord Field types. RECord and HRECord fields are "read only" type and can only be written to by the UUT.

:VECtor <source_vector>

The initial vector location in the source memory field where data will be copied from. The starting vector must be within the range of the size of the test (≤ test_size).

Parameter Definition

source_vector = (1 to test_size)

;COUNt <num_vectors | ALL>

The number of vector words that will be copied from the source field to the destination field. The number of vectors can also be specified by the literal string "ALL", where "ALL" is equal to the number of vectors from the starting vector location to the last vector in the test. The number of vectors to be copied must not exceed the last vector in the test.

Parameter Definition

num_vectors = (1 to ((test_size-start_vector) + 1)

ALL = All vectors from the start_vector location to the last vector in the test.

;DATA

The DATA command string provides the command path to the FIELd and COPY strings.

Parameter Definition:

none

:FIELd <source_name>

The FIELd parameter specifies the source memory field that data patterns will be copied from. Valid field types for the source FIELd parameter are Record (REC), Expected Response (EXP), DontCare (DON), Expected/Dontcare (ED), Algorithmic Expected (ALGE), Hardware Expected (HEXP), and Hardware DontCare (HDON). If the FIELd parameter is omitted, then the default memory field is assumed. The default memory field is defined by the RECord:FIELd command.

Parameter Definition **source_name** = Any alphanumeric string and '_' (max 8 characters).

;COPY

The COPY command string provides the command path to TO, FIELd, and EXECute.

Parameter Definition none

:TO <dest_vector>

The initial vector location in the destination memory field where data will be copied to. The destination starting vector must be within the range of the size of the test (≤ test_size).

Parameter Definition **dest_vector** = (1 to test_size)

;FIELd <dest_name>

The FIELd parameter specifies the destination memory field that data patterns will be copied to. All Stimulus and Record field types are valid destination fields, except the REC and HREC field types. RECord fields are "read only" type and can only be written to by the UUT. Stimulus field types are Output (OUT), Tristate (TRI), Output/Tristate (OT), Algorithmic Output (ALGO), Hardware Output, (HOUT), and Hardware Tristate (HTRI). Record Field types include Expected (EXP), DontCare (DON), Expected/Dontcare (ED), Algorithmic Expected (ALGE), Hardware Expected (HEXP), and Hardware DontCare (HDON). If the FIELd parameter is omitted, then the default memory field is assumed. The default memory field is defined by the STIMulus:FIELd command.

Parameter Definition **dest_name** = Any alphanumeric string and '_' (max 8 characters).

;EXECute

EXECute terminates the command string and executes the memory COPY command.

Parameter Definition none

Examples

```
RECORD:VECTOR 1;COUNT 100;DATA:FIELD INPUT;COPY:TO
200;FIELD INPUT;EXECUTE
```

This command copies 100 data words from vectors 1 - 100 to vectors 200-299. The source and destination field is the "INPUT" field.

```
REC:VECT 50;COUN 10;DATA:COPY:TO 60;EXECUTE
```

This command copies 10 data words from vectors 50 - 59 to vectors 60 - 69. The source and destination field are the default field as defined by the RECord:FIELd command.

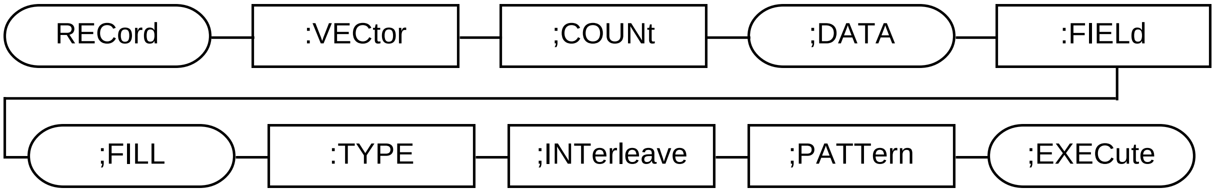
```
RECORD:VECTOR 1;COUNT ALL;DATA:FIELD INPUT;COPY:TO 1;FIELD
TST_DATA;EXECUTE
```

This command copies all data vectors from the "INPUT" field to the "TST_DATA" field.

```
REC:VECT 1;COUN 10;DATA:COPY:TO 11;EXEC;TO 21;EXEC;TO
31;EXEC;TO 41;EXEC
```

This command defines a block of 10 data words from vectors 1 - 10. This 10 vector block pattern is copied repetitively to vectors 11 - 20, 21 - 30, 31 - 40, and 41 - 50. The source and destination field are the default field as defined by the RECord:FIELd command.

Filling Response Memory (NON-SCPI)



The RECord:;;:;FILL command loads the Expected Response and Don'tcare memories with pre-defined pattern sequences. These pre-defined patterns load the data memory with commonly used data patterns without downloading a large amount of vectors from the Slot 0 Controller. This feature reduces the amount of data pattern programming and minimizes the test program download time. Pattern sequences include Repeat, Increment, Decrement, Complement, Alternate, Walking "1", Walking "0", and Pseudo-Random patterns. Data patterns will be loaded starting at the vector location, specified by the VECtor parameter, and will load the number of vector words specified by the COUNt parameter. The destination field may be any valid Record Field type (except REC and HREC field types).

Note

The RECord:;;:;FILL command should not be confused with the Algorithmic Command Macros. The RECord:;;:;FILL command loads response memory with data vectors. The Algorithmic Command Macros change the expected response pattern "on-the-fly" during run-time.

:VECtor <start_vector>

The initial vector location in the destination field where data will start loading. The starting vector must be within the range of the size of the test (≤ test_size).

Parameter Definition

start_vector = (1 to test_size)

;COUNt <num_vectors | ALL>

The number of vector words that will be loaded to memory. The number of vectors can also be specified the literal string "ALL", where "ALL" is equal to the number of vectors from the starting vector location to the last vector in the test. The number of vectors to be loaded must not exceed the last vector in the test.

Parameter Definition

num_vectors = (1 to ((test_size-start_vector) + 1)

ALL = All vectors from the start_vector location to the last vector in the test.

;DATA

The DATA command string provides the command path to the FIELd and FILL strings.

Parameter Definition none

:FIELd <name>

The FIELd parameter specifies the destination field where data patterns will be loaded to. Valid field types for the source FIELd parameter are Expected (EXP), Dontcare (DON), Expect/Dontcare (ED), Algorithmic Expected (ALGE), Hardware Expected, (HEXP), and Hardware Dontcare (HDON). If the FIELd parameter is omitted, then the default memory field is assumed. The default memory field is defined by the RECord:FIELd command.

Parameter Definition **name** = Any alphanumeric string and ‘_’ (max 8 characters).

;FILL

The FILL command string provides the command path to TYPE, INTerleave, PATTern, and EXECute strings.

Parameter Definition none

:TYPE <REPeat | INCrement | DECrement | COMplement | ALTernate | WLK1 | WLK0 | RANdom>

The type of pattern sequence that will be loaded to the destination memory field.

Parameter Definition **REPeat** = The REPeat parameter fills the memory repetitively with the same data pattern. The repeating data pattern is defined by the PATTern parameter.

INCrement = The INCrement parameter fills the memory with an incrementing data pattern. The initial data value that will begin incrementing is defined by the PATTern parameter.

DECrement = The DECrement parameter fills the memory with a decrementing data pattern. The initial data value that will begin decrementing is defined by the PATTern parameter.

COMplement = The COMplement parameter complements the current data value at each vector location. The PATTern parameter is not required and has no affect on the pattern fill command.

ALTernate = The ALTernate parameter fills the memory with an alternating data pattern. The initial data value that will begin alternating is defined by the PATTern parameter.

WLK1 = The WLK1 parameter fills the memory with a walking "1" data pattern. The "1" pattern will "walk" from LSB to MSB, i.e. #h0001, #h0002, #h0004, #h0008, etc.. The initial bit position that will begin walking is defined by the PATTern parameter. The WLK1 parameter will select the least significant "1" bit position in the PATTern parameter to begin the walking "1" pattern. For example, an initial PATTern data value of #h00F4 will begin walking from bit position 3, since the least significant "1" in #h00F4 is in the 3rd bit position from the LSB. All other "more significant 1s" will be ignored. Therefore, the walking "1" pattern will begin with #h0004 followed by #h0008, #h0010, etc. A PATTern parameter data value of #h0000 will cause all data values to be set to #h0000 and the walking "1" function will not be performed.

WLK0 = The WLK0 parameter fills the memory with a walking "0" data pattern. The "0" pattern will "walk" from LSB to MSB, i.e. #hFFFE, #hFFFD, #hFFFB, #hFFF7, etc.. The initial bit position that will begin walking is defined by the PATTern parameter. The WLK0 parameter will select the least significant "0" bit position in the PATTern parameter to begin the walking "0" pattern. For example, a initial PATTern data value of #h00F3 will begin walking from bit position 3, since the least significant "0" in #h00F3 is in the 3rd bit position from the LSB. All other "more significant 0s" will be ignored. Therefore, the walking "0" pattern will begin with #hFFFB followed by #hFFF7, #hFFEF, etc.

RANdom = The RANdom parameter fills the memory with a pseudo-random data pattern. The seed value that is used to initialize the pseudo-random calculation is determined by the PATTern parameter.

;INTerleave <int_count>

The INTerleave parameter specifies the interval count of the data vector locations to be "filled". For example, if the interleave count is set to two (2), then every other vector will be loaded with the fill function. Likewise, if the interleave count is set to ten (10), then every tenth vector will be loaded with the fill function. The default value for int_count is 1.

A powerful use of the INTerleave parameter is for loading complex data patterns to a multiplexed bus. An application example would be in the case of a multiplexed address/data bus. By setting the int_count to a value of two (2), the expected response can be loaded with a "checkerboard" pattern using the alternating pattern command.

Another use of the INTerleave parameter is for generating alternating Dontcare vectors. Again by setting the int_count to a value of two (2), the Dontcare field can be loaded with an alternating ones and zeros for alternating read and write cycles. This will allow the expected response to compare during read cycles and ignore during write cycles.

Parameter Definition

int_count = (1 - 10)

;PATTern <init_patt>

The PATTern parameter sets the initial data value for the fill function. Refer to each FILL TYPE command for details on the fill function performed on the initial data value. The default init_pattern is #h0.

Parameter Definition **init_patt** = #h{(0-F)} | #b{0 | 1}

;EXECute

EXECute terminates the command string and executes the memory FILL command.

Parameter Definition none

Examples

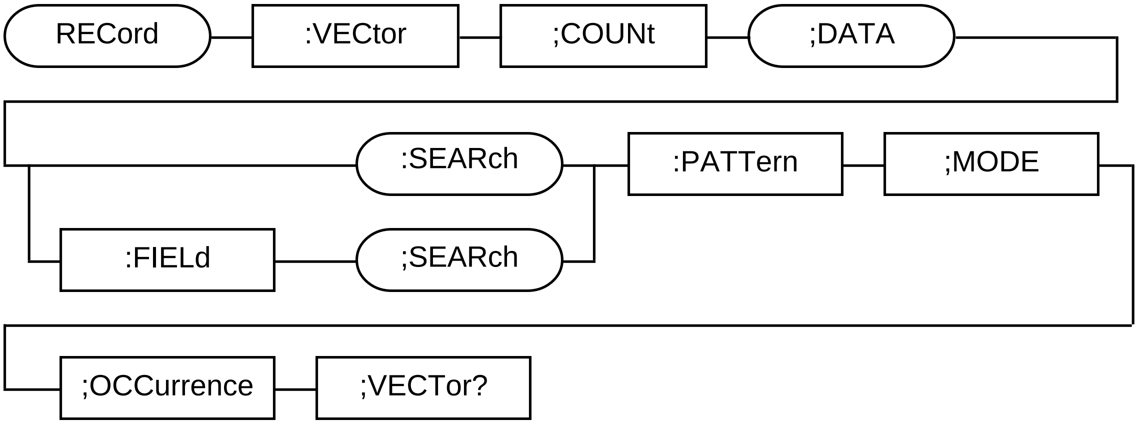
```
RECORD:VECTOR 1;COUNT 100;DATA:FIELD IN_DATA;FILL:TYPE
INCREMENT;INTERLEAVE 2;PATTERN #H0000;EXECUTE
```

This command fills every other vector location, starting at vector location 1, with an incrementing data pattern starting with an initial data value of #h0000.

```
REC:VECT 1;COUN ALL;DATA:FILL:TYPE ALT;PATT
#HAAAA;EXECUTE
```

This command fills all vector locations with an alternating data pattern of #hAAAA and #h5555.

Searching Response Memory (NON-SCPI)



The RECord:;;DATA:SEARch command searches through the Expected or Dontcare memories for specific pattern matches and returns the vector location and the matching data pattern. This command is useful for searching through the expected response memory for editing data patterns. The RECord:;;DATA:SEARch command will begin searching the specified memory field, starting at the vector location, specified by the VECtor parameter, and will search through the number of vector words specified by the COUNt parameter. A field other than the default field may be searched by using the optional FIELd parameter. The default memory field is defined by the RECord:FIELd command.

:VECtor <start_vector>

The initial vector location where the RECord:;;DATA:SEARch command will begin searching the record memory. The starting vector must be within the range of the size of the test.

Parameter Definition **start_vector** = (1 to test_size)

;COUNt <num_vectors | ALL>

The number of vector words in the record memory that will be searched. The number of vectors can also be specified using the literal string "ALL", where "ALL" is equal to the number of vectors from the starting vector location to the last vector in the test. The number of vectors to be searched must not exceed the last vector in the test.

Parameter Definition **num_vectors** = (1 to ((test_size - start_vector) + 1)

ALL = All vectors from the start_vector location to the last vector in the test.

;DATA

The DATA command string provides the command path to the FIELd and SEARch strings.

Parameter Definition none

:FIELd <name>

The FIELd parameter specifies the record memory field that will be searched for a pattern match. Valid field types for the FIELd parameter are Expected (EXP), Dontcare (DON), Algorithmic Expected (ALGE), Hardware Expected, (HEXP), and Hardware Dontcare (HDON). The Expect/Dontcare (ED) field types cannot be searched since ED fields consist of a combination of the Expect and Dontcare memories, and only one memory can be searched at a time. If the FIELd parameter is omitted, then the default field is assumed. The default field is defined by the RECord:FIELd command.

Parameter Definition **name** = Any alphanumeric string and '_' (max 8 characters).

Note

The FIELd parameter changes the destination field only for this occurrence of the command, but **does not** change the default field.

;SEARch

The SEARch command string provides the command path to the PATTERN, MODE and OCCurrence parameters.

Parameter Definition none

:PATTern <data_pattern>

The data_pattern parameter is the data pattern that will be searched for in the record memory. If no radix prefix (#h or #b) is used with the data pattern, then the data pattern must be entered in the radix format defined for the record field being searched. The radix format for the record field is defined by the FIELd:NAME:RADix command. If the radix for the record field is set to HEX, then data pattern can be specified in hexadecimal format (the '#h' prefix is optional) or in binary format if the '#b' prefix is specified. Valid hexadecimal data values are '0' through 'F'. If the radix for the field is set to BIN, then data pattern can be specified in binary format (the '#b' prefix is optional) or in hexadecimal format if the '#h' prefix is specified. Valid binary data values are '0' and '1'. Leading '0' data characters may be omitted as shown in the examples below.

Parameter Definition **data_pattern** = [#h](0-F) | [#b](0 | 1)

:MODE < EQ | NE | GT | LT >

The MODE parameter determines how the record memory vectors will be compared against the data_pattern.

Parameter Definition

EQ = Compares the record memory for an "equal to" match of the data_pattern.
NE = Compares the record memory for a "not equal to" match of the data_pattern.
GT = Compares the record memory for a "greater than" match of the data_pattern.
LT = Compares the record memory for a "less than" match of the data_pattern.

;OCCurrence <num_match | ALL>

The number of data_pattern match occurrences to search for and return. The SEARch command will terminate the search function when the number of match occurrences has been met or when the last vector in the search range has been searched. This parameters allows the vector locations returned to be limited so as to avoid large data transfers. The number of vectors can also be specified the literal string "ALL", where "ALL" will return all occurrences of data_pattern matches. The number of occurrences to be returned must not exceed the number of vectors to be searched.

Parameter Definition **num_match** = (1 to num_vectors)

ALL = All occurrences of data_pattern matches.

:VECTor? The VECTor? string terminates the command string and returns the matching vector locations and data patterns.

Response match_vector, data_pattern{;match_vector, data_pattern}

Parameter Definition **match_vector** = (start_vector to test_size)

data_pattern = #hXXXXXXXX, where X = (0 - F)

Examples

```
RECORD:VECTOR 1;COUNT 100;DATA:FIELD ADDR;SEARCH:PATTERN
#hAAAA;MODE EQ;OCCURRENCE ALL;VECTOR?
24,#h0000AAAA;65,#h0000AAAA;72,#h0000AAAA;90,#h0000AAAA
```

*The above command searched the "ADDR" field for **all** occurrences of data values **equal** to #hAAAA, starting at vector location 1 and searching the following 100 vector locations. The search command found four (4) matches at vector locations 24, 65, 72, and 90.*

```
REC:COUN ALL;DATA:SEAR:PATT #h0000;MODE GT;OCC 2;VECT?
1,#h00005B2C;4,#h00002F2A
```

*The above command searched all memory locations of the default field for the **first 2** occurrences of data values **greater than** #h0000. The search command found matches at vector locations 1 and 4 and terminated the search after the first 2 occurrences.*

THIS PAGE LEFT INTENTIONALLY BLANK

I/O Formatting and Timing

Each SR2510 module contains 6 high resolution timing generators for every 32 I/O channels which are used to control stimulus edge placement and sample timing. Four of these timing generators are used with data formatting controls to provide delay and pulse width timing for the stimulus channels. The remaining 2 timing generator channels are used to define edge and/or window sample timing for all 32 input channels. Each group of 32 output channels share 4 timing generator channels and the 4 channels may be used; to provide 2 delay times, to provide 2 delay time/ pulse width combinations, or to provide one of each. The two sample timing generators may be used to provide 2 edge sample clocks or one compare window.

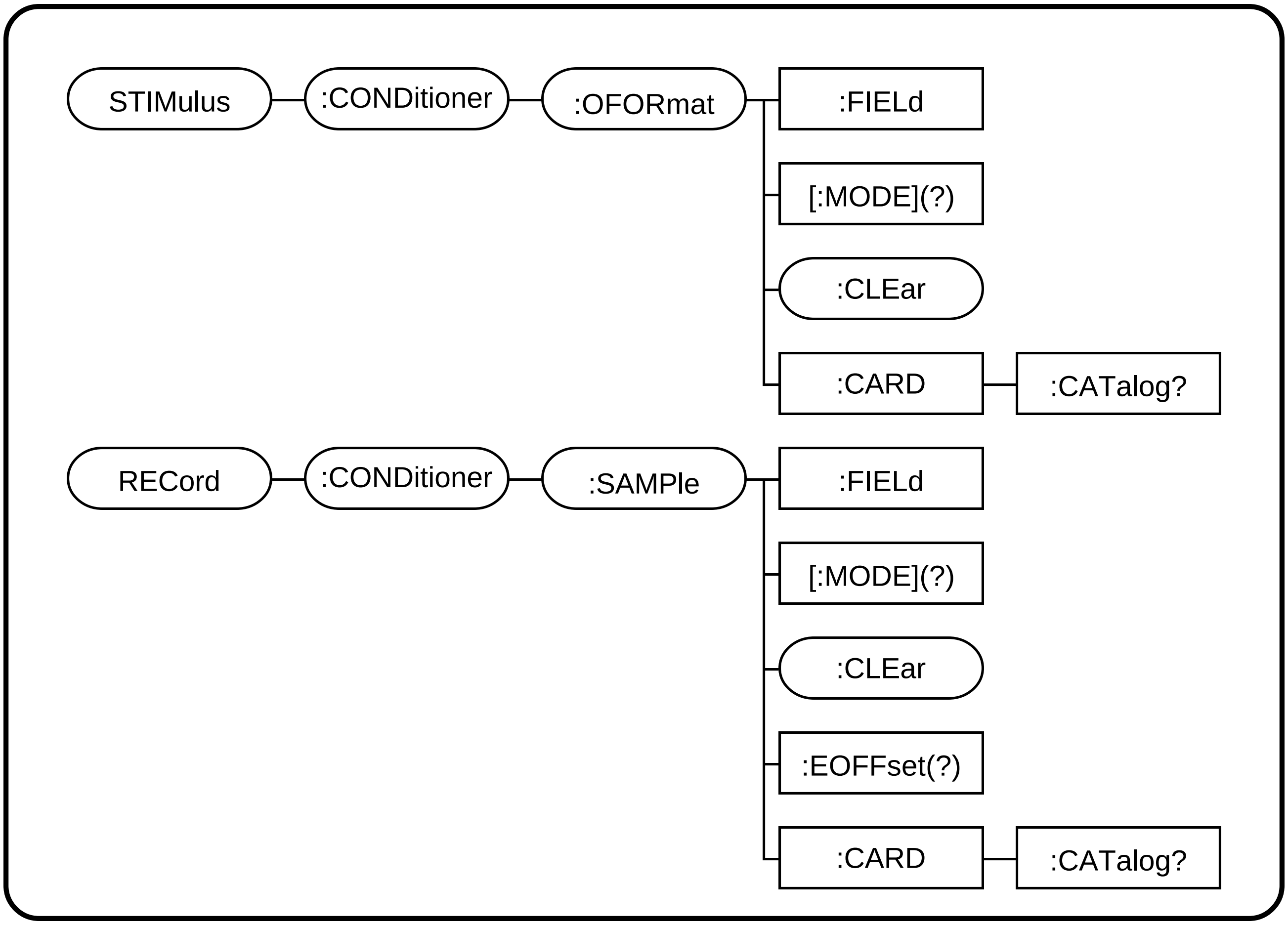
Note

Timing generator channel numbers are used here only for the purpose of describing how timing generator channels are paired, how they function and their restrictions. The SR2500 does not actually allow manipulation of timing generator channels by number.

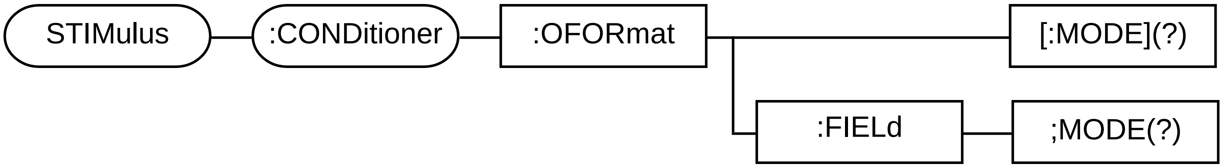
Timing generator channels are always internally paired in groups of 2, meaning that timing generator channels 0 and 1 will always form one pair, timing generator channels 2 and 3 will always form another, and so on. Even numbered timing generator channels are always used to provide delay times, and odd numbered channels are always used for pulse widths, for output formatting, or window widths for sample timing.

When a Non-Return-to-Zero (NRZ) output format is used, the pulse width timing generator channel is unused and unavailable for use elsewhere. The same is not true for the sample timing generator pair. Each of the 2 sample timing generator channels may be used to provide different edge sample times, or both may be used together for window sample. Any timing generator pair may not be placed closer than 10ns apart. However, timing generator channels from different pairs have no placement restrictions. Once a timing generator pair has been used to provide format timing, or sample timing, for a particular group of channels, it is not precluded from being used again, (within the same 32 channel group for stimulus channels), to provide a different format or sample mode, with the same timing parameters, on another group of channels.

For the most part, all of these considerations are taken care of within the SR2500's resource management routines and are transparent to the user. However, if multiple fields overlap, meaning they were defined sharing common pins, then the hardware format and timing for all overlapping pins will be set to the last parameters defined for any one of the fields. When defining two edge sample modes on the same I/O module, the timing parameters must be defined in descending order, i.e., from longest delay to shortest. If any violation is detected by the SR2510, a command error will be generated and the red "ERROR" LED on the SR2510 front panel will be illuminated. Use the SYSTEM:ERROR? query to read the error condition and clear the LED.


STIMulus
:CONDitioner
:OFORmat
:FIELd
[:MODE](?)
:CLEar
:CARD
:CATalog?
RECord
:CONDitioner
:SAMPle
:FIELd
[:MODE](?)
:CLEar
:EOFFset(?)
:CARD
:CATalog?

Stimulus Format and Timing (NON-SCPI)



The STIMulus:CONDitioner:OFORmat command defines the stimulus output format and timing values for the specified stimulus field. If multiple fields overlap, meaning they were defined sharing common pins, then the hardware format and timing for all overlapping pins will be set to the last parameters defined for any one of the fields. See the STIMulus: CONDitioner:OFORmat:CARD: CATalog? command for information about how to query the output format settings for each pin. The STIMulus:CONDitioner:OFORmat :MODE? query command returns the stimulus output format and timing values for the specified field.

:OFORmat

Selects the Output Format path of the STIMULUS:CONDITIONER subsystem.

Parameter Definition

none

[:MODE]

Determines the pin formatting and timing values for the default or specified field. MODE is the default command path. If omitted the parameters associated with MODE are placed after the OFORMAT command.

Parameter Definition

pin_format = < NRZ | RZ | RONE | RTC | RI >
delay_value = (0 - (clk_per - 5ns))
pulse_width = (10ns - (clk_per - 10ns))

For delay_value and pulse_width, values can be specified as a floating point numeric or in scientific notation using exponential values. Optional S, MS, US, and NS suffixes can be used for engineering unit multipliers. The default engineering unit is S (seconds).

NRZ = Non-Return-to-Zero. The data pattern specified for a given test vector will be output at "t_0 + delay_value" within the test cycle, and remain on the output for one full test cycle, i.e., until "t_1 + delay_value".

RZ = Return-to-Zero. The data pattern specified for a given test vector will be output at "t_0 + delay_value" within the test cycle, and remain on the output for pulse_width, after which the output will return to zero.

RONE = Return-to-ONE. The data pattern specified for a given test vector will be output at "t_0 + delay_value" within the test cycle, and remain on the output for pulse_width, after which the output will return to one.

RTC = Return-To-Complement. The data pattern specified for a given test vector will be output at "t_0 + delay_value" within the test cycle, and remain on the output for pulse_width, after which the output will return to its complement state.

RI = Return-to-Inhibit. The data pattern specified for a given test vector will be output at "t_0 + delay_value" within the test cycle, and remain on the output for pulse_width, after which the output will return to a tristate condition.

:FIELd <name | ALL>

The optional FIELd parameter specifies the stimulus type field that the OFORmat parameters will act on. Stimulus field types are Output (OUT), Tristate (TRI), Output/Tristate (OT), Algorithmic Output (ALGO), Hardware Output, (HOUT), and Hardware Tristate (HTRI). If the optional FIELd and MODE parameters are used, then the FIELd and MODE parameters must be separated by a semicolon, as shown. If the FIELd parameter is omitted, then the default stimulus field is used. The default stimulus field is defined by the STIMulus:FIELd command. The field name can also be specified by the literal string "ALL", where "ALL" refers to all stimulus type fields.

Parameter Definition

name = Any alphanumeric string and '_' (max 8 characters).

ALL = All stimulus fields will be set to the same stimulus formatting and timing values.

Note

The FIELd parameter changes the destination field only for the same command but **does not** change the default field.

Examples

```
STIMULUS:CONDITIONER:OFORMAT:FIELD ADDR;MODE NRZ,10ns
STIM:COND:OFOR:FIEL DATA;MODE RI,30ns,50ns
STIM:COND:OFOR RONE,5.0e-8,2.22e-8
```

:MODE?

Returns the output format and timing for the specified field.

Response

name pin_format,delay_value[,pulse_width]

Parameter Definition

name = The specified field name.
pin_format = < UNDEFINED | NRZ | RZ | RONE | RTC | RI >
delay_value = (0 - (clk_per - 5ns))
pulse_width = (10ns - (clk_per - 10ns))

For delay_value and pulse_width, units are in seconds and are always returned in scientific notation.

UNDEFINED = An output format has not been defined for the specified field, or the output format has been cleared.

NRZ = Non-Return-to-Zero.
RZ = Return-to-Zero.
RONE = Return-to-ONE.
RTC = Return-To-Complement.
RI = Return-to-Inhibit.

Examples STIMULUS:CONDITIONER:OFORMAT:FIELD ADDR;MODE?
ADDR NRZ,1.000000e-8

STIM:COND:OFOR:FIEL DATA;MODE?
DATA RTI,3.000000e-8,5.000000e-8

STIM:COND:OFOR?
DEFAULT RONE,5.000000e-8,2.220000e-8

Note

When an external clock is implemented, the selection of positive and negaive clock edges are as follows:

STIMulus:CONDitioner:OFORmat:FIELD<name | all>;MODE <NRZ | RZ | RONE | RTC | RI>, <0 | 1>

0 = positive edge, rising edge of clock

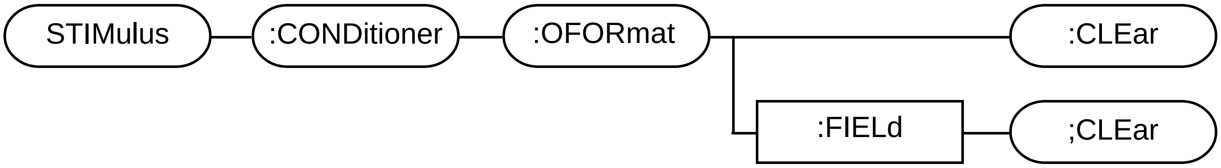
1 = negative edge, falling edge of clock

Examples STIMULUS:CONDITIONER:OFORMAT:FIELD DATA;MODE NRZ, 1
STIM:COND:OFOR:FIELD ADDR; MODE RI, 0

STIMULUS:CONDITIONER:OFORMAT:FIELD DATA;MODE?
DATA NRZ,1.000000E+00

STIM:COND:OFOR:FIELD ADDR; MODE?
ADDR RI,0

Clearing Stimulus Format and Timing (NON-SCPI)



The STIMulus:CONDitioner:OFORmat:CLEar command clears the stimulus output format and timing values for the specified stimulus field. The stimulus output format will be set to UNDEFINED. Clearing a field's output format does not change either the data format nor the timing parameters used by the hardware. But it does free up the timing generator resources so they may be defined with new data format and timing parameters.

If multiple fields share the same timing generator resources, i.e., they share pins in the same 32 pin group and they have identical timing, then in order to free the timing resources, each field must be cleared. If multiple fields overlap, meaning they were defined sharing common pins, then the hardware format and timing for all overlapping pins will be set to the last parameters defined for any one of the fields. See the STIMulus:CONDitioner:OFORmat:CARD: CATalog? command for information about how to query the output format settings for each pin.

:FIELd <name | ALL>

The optional FIELd parameter specifies the stimulus type field that the output format and timing edge placement values will clear. Stimulus field types are Output (OUT), Tristate (TRI), Output/Tristate (OT), Algorithmic Output (ALGO), Hardware Output, (HOUT), and Hardware Tristate (HTRI). If the optional FIELd parameter is used, then the FIELd and CLEar strings must be separated by a semicolon. If the FIELd parameter is omitted, then the default stimulus field is assumed. The default stimulus field is defined by the STIMulus:FIELd command. The field name can also be specified by the literal string "ALL", where "ALL" refers to all stimulus type fields.

Parameter Definition

name = Any alphanumeric string and ‘_’ (max 8 characters).

ALL = All stimulus fields will be cleared.

Note

The FIELd parameter changes the destination field only for the same command but **does not** change the default field.

:CLEar

Terminates the command and executes the clear function.

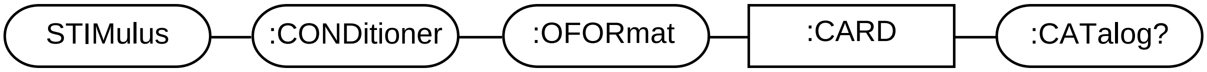
Parameter Definition

none

Examples

```
STIMULUS:COND:OFORMAT:FIELD ADDR;CLEAR
STIM:COND:OFOR:CLE
```

Stimulus Format and Timing Catalog (NON-SCPI)



The STIMulus:CONDitioner:::CATalog? query command returns the last pin formatting information for the specified I/O module. The STIMULUS::CARD:CATALOG? response is not updated after a STIMULUS::OFORMAT:CLEAR command, and will continue to reflect the last defined output format for each output pin. Use the STIMULUS:::MODE? query command to determine if a fields' output formatting is UNDEFINED.

:CARD <card_num | ALL>

The I/O card number to be queried. {See Chapter 5 "Logical Addressing" for information pertaining to I/O card numbers.}

Parameter Definition

card_num = (1 - 18); up to the number of I/O modules installed in the test system.

ALL = All the I/O Modules installed in the SR2500 test system.

:CATalog?

Returns the pin formatting information for the specified I/O module.

Response

{stimulus_pin pin_format,delay_value[,pulse_width]$^{CR}_{LF}$}; repeat 32 times for each module specified.

Parameter Definition

stimulus_pin = C<card_num>P<pin_num>
card_num = (1 - 18)
pin_num = (1-32)
pin_format = < NRZ | RZ | RONE | RTC | RI >
delay_value = (0 - (clk_per - 5ns))
pulse_width = (10ns - (clk_per - 10ns))

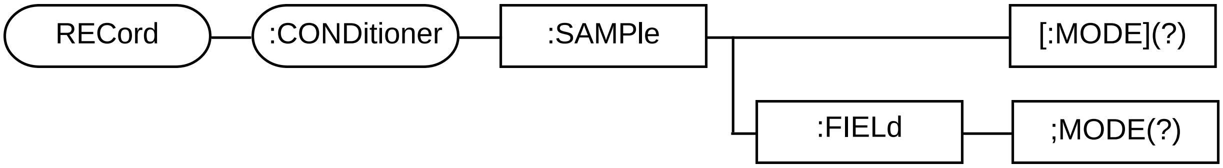
For delay_value and pulse_width, units are in seconds and are always returned in scientific notation. The parameters delay_value and pulse_width are not required to fit within one clock cycle.

Examples

```
STIMULUS:CONDITIONER:OFORMAT:CARD 1:CATALOG?
C1P1 NRZ,1.000000e-08
C1P2 NRZ,1.000000e-08
C1P3 NRZ,1.000000e-08
C1P4 NRZ,1.000000e-08
C1P5 NRZ,1.000000e-08
C1P6 NRZ,1.000000e-08
C1P7 NRZ,1.000000e-08
C1P8 NRZ,1.000000e-08
C1P9 NRZ,1.000000e-08
C1P10 NRZ,1.000000e-08
C1P11 NRZ,1.000000e-08
C1P12 NRZ,1.000000e-08
```

```
C1P13 NRZ,1.000000e-08
C1P14 NRZ,1.000000e-08
C1P15 NRZ,1.000000e-08
C1P16 NRZ,1.000000e-08
C1P17 RZ,1.000000e-08,4.000000e-08
C1P18 RZ,1.000000e-08,4.000000e-08
C1P19 RZ,1.000000e-08,4.000000e-08
C1P20 RZ,1.000000e-08,4.000000e-08
C1P21 RZ,1.000000e-08,4.000000e-08
C1P22 RZ,1.000000e-08,4.000000e-08
C1P23 RZ,1.000000e-08,4.000000e-08
C1P24 RZ,1.000000e-08,4.000000e-08
C1P25 RZ,1.000000e-08,4.000000e-08
C1P26 RZ,1.000000e-08,4.000000e-08
C1P27 RZ,1.000000e-08,4.000000e-08
C1P28 RZ,1.000000e-08,4.000000e-08
C1P29RZ,1.000000e-08,4.000000e-08
C1P30 RZ,1.000000e-08,4.000000e-08
C1P31 RZ,1.000000e-08,4.000000e-08
C1P32 RZ,1.000000e-08,4.000000e-08
```

Record Sample Mode and Timing (NON-SCPI)



The RECord:CONDitioner:SAMPle command defines the sample mode and timing for the specified response field. If multiple fields overlap, meaning they share common pins, then the sample mode and timing for all overlapping pins will be set to the parameters defined by the last command issued. See the RECord:CONDitioner:OFORmat:CARD:CATalog? command for information about how to query the sample mode and timing settings for each pin. The RECord:CONDitioner:SAMPle:MODE? query command returns the sample mode and timing for the specified field.

:SAMPle

Selects the SAMPle path of the RECORD:CONDITIONER subsystem.

Parameter Definition

none

[:MODE]

Defines the sample mode and timing values for the default or specified field. MODE is the default command path. If omitted the parameters associated with MODE are placed after the SAMPle command.

Parameter Definition

sample_mode = < EDGE | WINDow >
delay_value = (0 - (clk_per - 5ns))
window_width = (10ns - (clk_per - 10ns))

For delay_value and window_width, values can be specified as a floating point numeric or in scientific notation using exponential values. Optional S, MS, US, and NS suffixes can be used for engineering unit multipliers. The default engineering unit is S (seconds).

EDGE = Used in Edge Sample and Edge Compare, the input pins specified will be sampled at "t_0 + delay_value" within the test cycle and either stored in the record memory, or compared against the expected response for that vector, as indicated by the RECORD:TRACE controls. If RECORD:TRACE is set to record DATA, then the state of the input pins will be stored in the record memory. If RECORD:TRACE is set to record ERRORS, then for each bit that compares true, a '0' will be stored in the record memory, and for each bit that compares false, a '1' will be stored in the record memory.

WINDow = Used in Window Compare, the input pins specified will be compared against the expected response for the current vector, starting at "t_0 + delay_value" and for a duration of window_width. The WINDOW sample mode is intended for use with real-time compare operations, however, using WINDOW mode to sample data is not precluded.

If RECORD:TRACE is set to record DATA, then the state of the input pins at the end of the window will be stored in the record memory. If RECORD:TRACE is set to record ERRORS, then for each bit that compares true for the duration of the window, a '0' will be stored in the record memory, and for each bit that compares false anytime during the window, a '1' will be stored in the record memory.

:FIELd <name | ALL>

The optional FIELd parameter specifies the record type field that the SAMPle parameters will act on. Record Field types include Expected (EXP), DontCare (DON), Expected/Dontcare (ED), Algorithmic Expected (ALGE), Hardware Expected (HEXP), and Hardware DontCare (HDON). If the optional FIELd parameter is used, then the FIELd and MODE parameters must be separated by a semicolon. If the FIELd parameter is omitted, then the default record field is assumed. The default record field is defined by the RECord:FIELd command. The field name can also be specified by the literal string "ALL", where "ALL" refers to all record type fields.

Parameter Definition

name = Any alphanumeric string and '_' (max 8 characters).

ALL = All record fields will be set to the same sample timing values.

Note

The FIELd parameter changes the destination field only for the same command but **does not** change the default field.

Examples

```
RECORD:CONDITIONER:SAMPLE:FIELD ADDR;MODE EDGE,10ns
REC:COND:SAMP:FIEL DATA;MODE WIND,30ns,50ns
REC:COND:SAMP WIND,5.0e-8,2.22e-8
```

:MODE?

The MODE? query command will return the sample mode and timing for the specified field.

Response

name pin_format,delay_value[,pulse_width]

Parameter Definition

name = The specified field name.
sample_mode = < UNDEFINED | EDGE | WIND >
delay_value = (0 - (clk_per - 5ns))
window_width = (10ns - (clk_per - 10ns))

For delay_value and window_width, units are in seconds and may be entered in integer of floating point format. They may optionally include ns, us and ms suffixes and scientific notation.

UNDEFINED = An sample mode has not been defined for the specified field, or the sample mode has been cleared.

EDGE = Edge Sample/Compare.

WIND = Window Compare.

Examples

RECORD:CONDITIONER:SAMPLE:FIELD ADDR;MODE?
ADDR EDGE,1.000000e-8

REC:COND:SAMP:FIEL DATA;MODE?
DATA WIND,3.000000e-8,5.000000e-8

REC:COND:SAMP?
DEFAULT WIND,5.000000e-8,2.220000e-8

Note

When using an external clock source, edge placement resolution is limited to the fixed edges of that external clock. An option exists to chooe either positive or negative edges for recording, based upon the following command:

REC:COND:SAMPLE:FIELD <name | all>:MODE <edge>, <0 | 1>

0 = positive edge, rising edge of clock
1 = negative edge, falling edge of clock

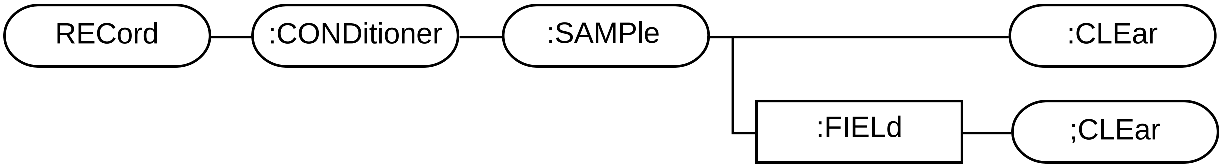
Examples

RECORD:CONDITIONER:SAMPLE:FIELD DATA;MODE EDGE 1
REC:COND:SAMP:FIEL ADDR;MODE EDGE 0

RECORD:CONDITIONER:SAMPLE:FIELD DATA;MODE?
DATA EDGE, 1.000000E+00

REC:COND:SAMP:FIEL ADDR;MODE?
ADDR EDGE, 0.000000E+00

Clearing Record Sample Mode and Timing (NON-SCPI)



The RECord:CONDitioner:SAMPle:CLEar command clears the sample mode and timing values for the specified record field. The sample mode will be set to UNDEFINED. Clearing a field's sample mode does not change either the sample mode nor the timing parameters used by the hardware. But it does free up the timing generator resources so they may be defined with new data format and timing parameters.

If multiple record fields share the same timing generator resources, i.e., they are defined using pins on the same module and they have identical timing, then in order to free the timing resources, each field must be cleared. If multiple fields overlap, meaning they share common pins, then the sample mode and timing for all overlapping pins will be set to the parameters defined by the last command issued. See the RECord:CONDitioner:OFORmat: CARD:CATalog? command for information about how to query the sample mode and timing settings for each pin.

:FIELd <name | ALL>

The optional FIELd parameter specifies the record type field on which the sample mode and timing values will be cleared. Record Field types include Expected (EXP), DontCare (DON), Expected/Dontcare (ED), Algorithmic Expected (ALGE), Hardware Expected (HEXP), and Hardware DontCare (HDON). If the optional FIELd parameter is used, then the FIELd and CLEar strings must be separated by a semicolon. If the FIELd parameter is omitted, then the default record field is assumed. The default record field is defined by the RECord:FIELd command. The field name can also be specified by the literal string "ALL", where "ALL" refers to all record type fields.

Parameter Definition

name = Any alphanumeric string and ‘_’ (max 8 characters).

ALL = All record fields will be cleared.

Note

The FIELd parameter changes the destination field only for the same command but **does not** change the default field.

:CLEar

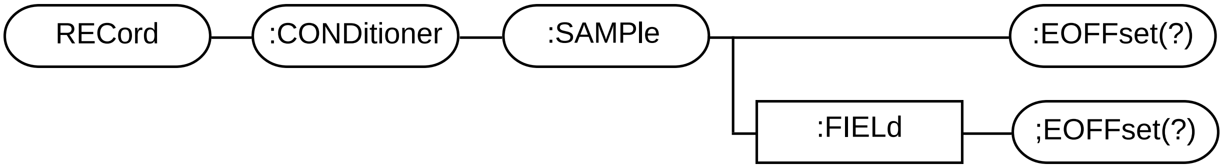
Executes the clear function.

Parameter Definition none

Examples

```
RECORD:CONDITIONER:SAMPLE:FIELD ADDR;CLEAR
REC:COND:SAMP:CLE
```

Expected Compare Offset (NON-SCPI)



The RECord:CONDitioner:SAMPle:EOFFset command allows the Expected data pattern to be offset from the corresponding stimulus vector in order to compensate for propagation and UUT delays. Only a single expect offset may be specified for any group of 8 pins. If a single pin in an 8 pin group is assigned an expect offset, then all pins in the group are assigned the same offset value and those pins are excluded from being defined any other EOFFSET value. To clear a defined expect offset for a field, use the RECord:CONDitioner:SAMPle:CLEar command.

:FIELd <name | ALL>

The optional FIELd parameter specifies the record type field that the expect offset will apply to. Record Field types include Expected (EXP), DontCare (DON), Expected/Dontcare (ED), Algorithmic Expected (ALGE), Hardware Expected (HEXP), and Hardware DontCare (HDON). If the optional FIELd parameter is used, then the FIELd and EOFFset strings must be separated by a semicolon. If the FIELd parameter is omitted, then the default record field is assumed. The default record field is defined by the RECord:FIELd command. The field name can also be specified by the literal string "ALL", where "ALL" refers to all record type fields.

Parameter Definition

name = Any alphanumeric string and '_' (max 8 characters).

ALL = All record fields will be set to the same expect offset value.

Note

The FIELd parameter changes the destination field only for the same command but **does not** change the default field.

:EOFFset <expect_offset>

Defines the number of clock cycles to offset the expected response relative to the stimulus output for the same vector.

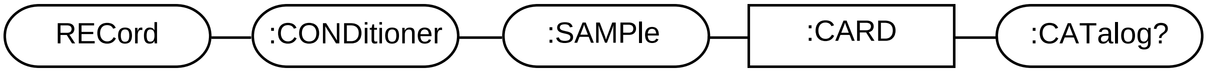
Parameter Definition

expect_offset = (0 - 7)

Examples

```
RECORD:CONDITIONER:SAMPLE:FIELD DATA;EOFFSET 3
REC:COND:SAMP:EOFF 7
```

Record Sample Mode and Timing Catalog (NON-SCPI)



The RECord:CONDitioner:::CATalog? query command returns the last sample mode and timing information for each pin on the specified I/O module. The RECord:::CATalog? response is not updated after a RECord::SAMPle:CLEAR command, and will continue to reflect the last defined sample mode for each pin. Use the RECord:::MODe? query command to determine if a field's output formatting is UNDEFINED.

:CARD <card_num | ALL>

The I/O module number to be queried.

Parameter Definition

card_num = (1 - 18); up to the number of I/O modules installed in the test system.

ALL = All the I/O Modules installed in the SR2500 test system.

:CATalog?

Returns the sample mode and timing information for the specified I/O module.

Response

{record_pin sample_mode,delay_value[,window_width];}; repeat 32 times for each I/O module specified.

Parameter Definition

record_pin = C<card_num>P<pin_num>
card_num = (1 - 18)
pin_num = (1-32)
sample_mode = < EDGE | WIND >
delay_value = (0 - (clk_per - 5ns))
window_width = (10ns - (clk_per - 10ns))

For delay_value and pulse_width, units are in seconds and are always returned in scientific notation. Both delay_value and window_width must fit within one clock cycle.

Examples

```
REC:COND:SAMP:CARD 1:CAT?
```

C1P1 EDGE,1.000000e-08;C1P2 EDGE,1.000000e-08;C1P3 EDGE,1.000000e-08;C1P4 EDGE,1.000000e-08;C1P5 EDGE,1.000000e-08;C1P6 EDGE,1.000000e-08;C1P7 EDGE,1.000000e-08;C1P8 EDGE,1.000000e-08;C1P9 EDGE,1.000000e-08;C1P10 EDGE,1.000000e-08;C1P11 EDGE,1.000000e-08;C1P12 EDGE,1.000000e-08;C1P13 EDGE,1.000000e-08;C1P14 EDGE,1.000000e-08;C1P15 EDGE,1.000000e-08;C1P16 EDGE,1.000000e-08;C1P17 WIND,1.000000e-08,2.000000e-08;C1P18 WIND,1.000000e-08,2.000000e-08;C1P19 WIND,1.000000e-08,2.000000e-08;C1P20 WIND,1.000000e-08,2.000000e-08;C1P21 WIND,1.000000e-08,2.000000e-08;C1P22 WIND,1.000000e-08,2.000000e-

(Continued at bottom of page 144)

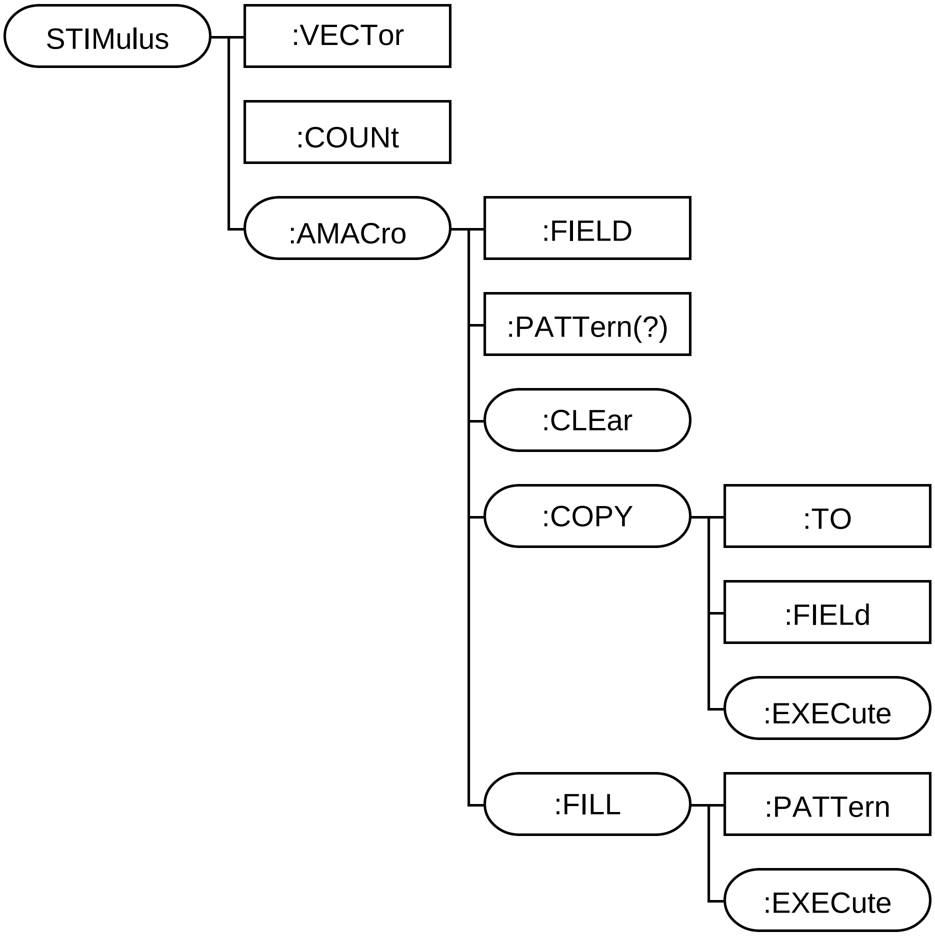
Algorithmic Pattern Generation

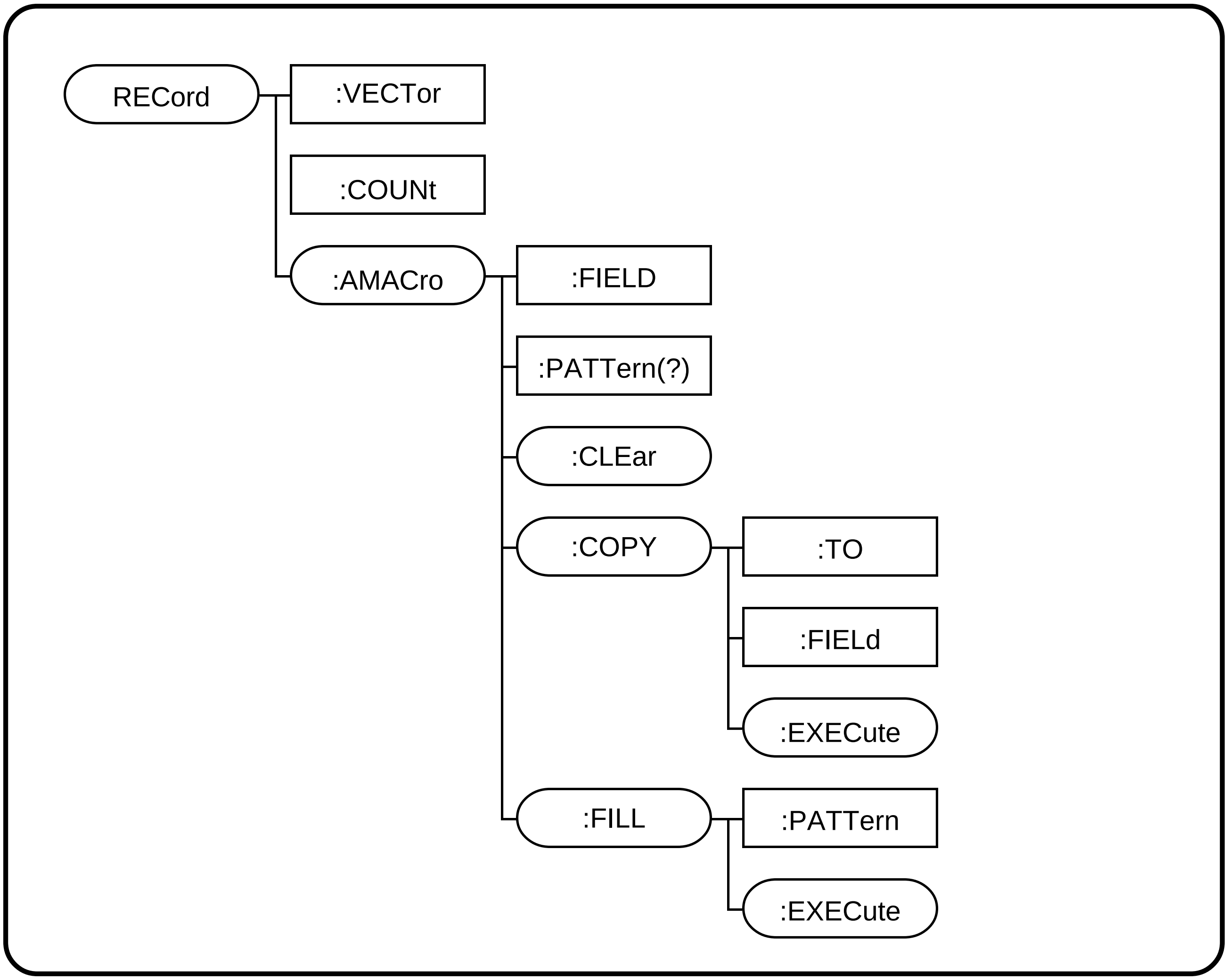
Algorithmic commands allow real-time generation of stimulus patterns and expected responses based on simple functions. Used with CMACRO vector looping commands, algorithmic pattern generation allows extremely long patterns to be generated using very few actual vectors, and, they may be generated at full system speeds. Using nested CMACRO loop counters, literally billions of unique patterns may be generated using less than a dozen commands, without dead-time or gaps between the various loop cycles. This makes algorithmic pattern generation ideal for testing large memory devices or boards.

Each algorithmic field may be programmed with its own commands, including commands that specify the use of Output and Expect memory data as the data pattern. This allows mixing of algorithmic and RAM-backed patterns. Algorithmic commands may be loaded, or queried, discreetly, or entire ranges of commands may be cleared, filled or copied using the CLEAR, FILL and COPY editing commands, respectively.

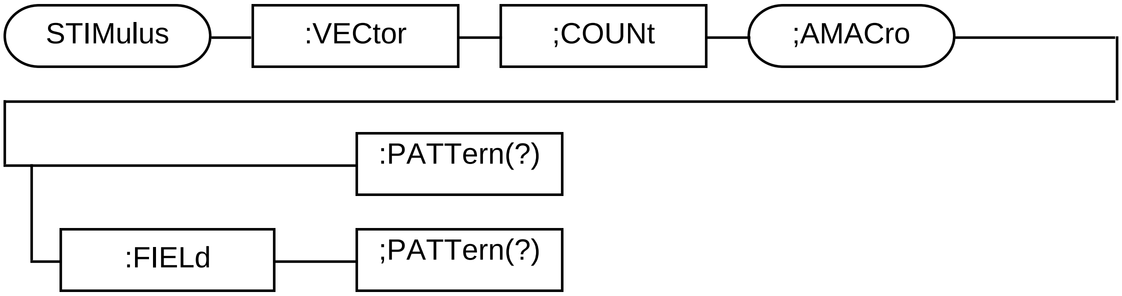
(CONTINUED FROM PAGE 143)

08;C1P23WIND,1.000000e-08,2.000000e-08;C1P24 WIND,1.000000e-08,2.000000e-08;C1P25 WIND,1.000000e-08,2.000000e-08;C1P26 WIND,1.000000e-08,2.000000e-08;C1P27 WIND,1.000000e-08,2.000000e-08;C1P28 WIND,1.000000e-08,2.000000e-08;C1P29 WIND,1.000000e-08,2.000000e-08;C1P30 WIND,1.000000e-08,2.000000e-08;C1P31 WIND,1.000000e-08,2.000000e-08;C1P32 WIND,1.000000e-08,2.000000e-08


STIMulus
:VECTor
:COUNt
:AMACro
:FIELD
:PATTern(?)
:CLEar
:COPY
:TO
:FIELd
:EXECute
:FILL
:PATTern
:EXECute


RECord
:VECTor
:COUNt
:AMACro
:FIELD
:PATTern(?)
:CLEar
:COPY
:TO
:FIELd
:EXECute
:FILL
:PATTern
:EXECute

Stimulus Algorithmic Pattern Generation (NON-SCPI)



The STIMulus:;;AMACro:PATTern command loads algorithmic macro commands to the specified field. The field to be loaded must be defined as an Algorithmic Output (ALGO) field type. Algorithmic commands (patterns) will be loaded to the stimulus field starting at the vector location specified by the VECtor parameter, and will load the number of algorithmic commands specified by the COUNt parameter. The STIMulus:;;AMACro:PATTern? query command returns the algorithmic commands from the specified field for the range defined by the VECTor and COUNt parameters.

:VECtor <start_vector>

The initial vector location where algorithmic macro commands will start loading/querying. The starting vector location must be within the range of the size of the test (≤ test_size).

Parameter Definition

start_vector = (1 to test_size)

;COUNt <num_vectors | ALL>

The number of algorithmic macro commands that will be loaded/queried. The number of macro commands can also be specified by the literal string "ALL", where "ALL" is equal to the number of vectors from the starting vector location to the last vector in the test. The number of macro commands must not exceed the last vector in the test.

Parameter Definition

num_vectors = (1 to ((test_size-start_vector) + 1)

ALL = All vectors from the start_vector location to the last vector in the test.

;AMACro

The AMACro command string provides a path to FIELd and PATTern parameters.

Parameter Definition

none

:FIELd <name>

The optional FIELd parameter specifies the field where the algorithmic macro commands will be loaded to (or queried from). The destination field must be an Algorithmic Output (ALGO) field type. If the FIELd parameter is used, then the FIELd and PATTern parameters must be separated by a semicolon. If the FIELd parameter is omitted, then the default stimulus field is assumed. The default stimulus field is defined by the STIMulus:FIELd command.

Parameter Definition **name** = Any alphanumeric string and '_' (max 8 characters).

Note

The FIELd parameter changes the destination field only for the same command but **does not** change the default field.

:PATTern <alg_macro>{,alg_macro}

Specifies the algorithmic command(s) that will be loaded to the ALGO field. The default algorithmic command is the NONAlgorithmic command The number of alg_macro elements must be equal to num_vectors. If a count mismatch occurs, the macro commands will be loaded up to the number of alg_macro elements provided in the command string, or until the num_vectors count is reached, whichever is less. An error message will be generated and the "ERROR" LED on the front panel of the SR2510 module will be lit. Refer to the SYSTem:ERRor? query command for information about reading command errors.

Parameter Definition **alg_macro** = The following is a list of valid algorithmic macro commands.

1. NONAlgorithmic
2. INCrement
3. DECrement
4. XOR
5. HOLDData
6. HOLDAll
7. SLEFTZero
8. SLEFTOne
9. SLEFTComplement
10. SRIGHTZero
11. SRIGHTOne
12. SRIGHTComplement
13. RLEFT
14. RRIGHT
15. LOADParam
16. OUTPUTParam

Algorithmic Output Command Definitions

NONAlgorithmic

The NONAlgorithmic command allows the Stimulus Gate Arrays to act as a pass through for data from RAM to the output pins. The data which is passed from RAM to output is also used to initialize the algorithmic register. This register may be acted upon by other algorithmic commands to modify the data content.

INCrement

Increment the contents of the algorithmic register and pass the results to the output pins. If algorithmic fields greater than 8 bits are used, multiple gate arrays are interlinked. If an increment instruction causes an overflow in one gate array, the overflow is used as a carry input to the next most significant gate array, thus extending the count up to a maximum of 2^{32} before roll over.

DECrement

Decrement the contents of the algorithmic register and pass the results to the output pins. If algorithmic fields greater than 8 bits are used, multiple gate arrays are interlinked. If a decrement instruction causes an underflow in one gate array, the underflow is used as a borrow input to the next most significant gate array, thus extending the count up to a maximum of 2^{32} before roll over.

XOR

The XOR instruction will perform a bitwise exclusive ORing of the algorithmic register with the contents of Output RAM for that vector. In this case the RAM data acts as a modifier to the algorithmic register, but does not directly load it, thus allowing selective bits of the algorithmic register to be complemented before being passed to the output pins.

HOLDData

Instructs the Stimulus Gate Arrays to hold the state of the algorithmic register from the previous vector on the output pins. This command affects only the output data, tristate control for the vector is still provided from the Tristate RAM.

HOLDAll

The HOLDAll command instructs the Stimulus Gate Arrays to hold the state of the algorithmic register and the tristate control from the previous vector on the output pins. This command is similar to the HOLDData command, however both the output data and tristate control are affected.

SLEFTZero

Shift the contents of the algorithmic register left (LSB to MSB) one bit, fill the LSB with 0 and pass the results to the output pins. If algorithmic fields greater than 8 bits are used, multiple gate arrays are interlinked. In this case, the MSB output of a less significant gate array is used as a LSB input to the next most significant gate array, thus extending the shift to a maximum 32 bits.

SLEFTOne

Shift the contents of the algorithmic register left (LSB to MSB) one bit, fill the LSB with 1 and pass the results to the output pins. If algorithmic fields greater than 8 bits are used, multiple gate arrays are interlinked. In this case, the MSB output of a less significant gate array is used as a LSB input to the next most significant gate array, thus extending the shift to a maximum 32 bits.

SLEFTComplement

Shift the contents of the algorithmic register left (LSB to MSB) one bit, complement the LSB and pass the results to the output pins. If algorithmic fields greater than 8 bits are used, multiple gate arrays are interlinked. In this case, the MSB output of a less significant gate array is used as a LSB input to the next most significant gate array, thus extending the shift to a maximum 32 bits.

RLEFT

Rotate the contents of the algorithmic register left (LSB to MSB) one bit, wrap the MSB to the LSB and pass the results to the output pins. If algorithmic fields greater than 8 bits are used, multiple gate arrays are interlinked. In this case, the MSB output of a less significant gate array is used as a LSB input to the next most significant gate array and the MSB of the most significant gate array is wrapped to the LSB of the least significant gate array, thus extending the rotation to a maximum 32 bits.

SRIGHTZero

Shift the contents of the algorithmic register right (MSB to LSB) one bit, fill the MSB with 0 and pass the results to the output pins. If algorithmic fields greater than 8 bits are used, multiple gate arrays are interlinked. In this case, the LSB output of a more significant gate array is used as a MSB input to the next least significant gate array, thus extending the shift to a maximum 32 bits.

SRIGHTOne

Shift the contents of the algorithmic register right (MSB to LSB) one bit, fill the MSB with 1 and pass the results to the output pins. If algorithmic fields greater than 8 bits are used, multiple gate arrays are interlinked. In this case, the LSB output of a more significant gate array is used as a MSB input to the next least significant gate array, thus extending the shift to a maximum 32 bits.

SRIGHTComplement

Shift the contents of the algorithmic register right (MSB to LSB) one bit, complement the MSB and pass the results to the output pins. If algorithmic fields greater than 8 bits are used, multiple gate arrays are interlinked. In this case, the LSB output of a more significant gate array is used as a MSB input to the next least significant gate array, thus extending the shift to a maximum 32 bits.

RRIGHT

Rotate the contents of the algorithmic register right (MSB to LSB) one bit, wrap the LSB to the MSB and pass the results to the output pins. If algorithmic fields greater than 8 bits are used, multiple gate arrays are interlinked. In this case, the LSB output of a more significant gate array is used as a MSB input to the next least significant gate array and the LSB of the least significant gate array is wrapped to the MSB of the most significant gate array, thus extending the rotation to a maximum 32 bits.

LOADParam

The LOADParm command loads the contents of Output RAM into an algorithmic holding register internal to the Stimulus Gate Array. This value may later be passed to the algorithmic register, and placed on the output pins, using the OUTPUTParam command. This allows test subroutines to be passed a data pattern to use within the routine, such as a starting address for a RAM read or write cycle. Since the data pattern is not actually output until the OUTPUTParam command is executed, the subroutine may run a standard initialization pattern before using the passed pattern.

OUTPUTParm

Passes the data pattern loaded into the algorithmic holding register by the LOADParam command, to the algorithmic register and the output pins. This allows test subroutines to be passed a data pattern to use within the routine, such as an address for a microprocessor bus cycle. Since the data pattern is not actually output until the OUTPUTParam command is executed, the subroutine may run a standard initialization pattern before using the passed pattern.

Examples

```
STIMULUS:VECTOR 1;COUNT 1;AMACRO:FIELD ADDR;PATTERN
INCREMENT
STIM:VEC 1;COUN 1;AMAC:PATT INC
STIM:VEC 1;COUN 4;AMAC:PATT NONA,INC,DEC,XOR
```

:PATTern?

The alg_macro parameter is the algorithmic macro command that will be read from the algorithmic output field. See the STIMulus:;AMACro::PATTern command for a description of the algorithmic macro commands.

Response alg_macro{,alg_macro}

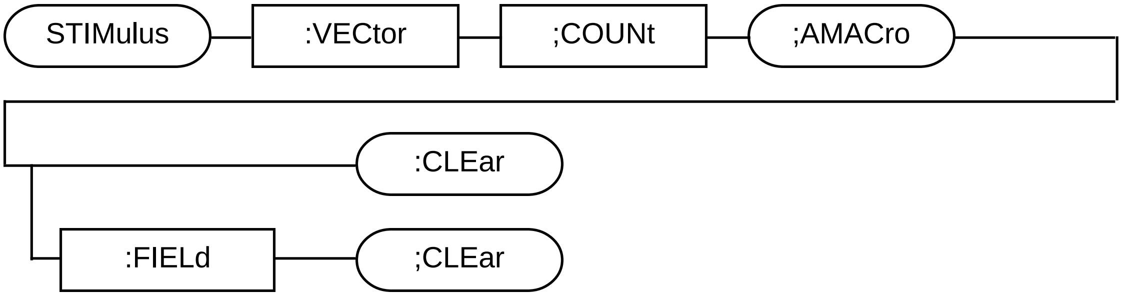
Parameter Definition **algo_macro** = < **NONA | INC | DEC | XOR | HOLDD | HOLDA | SLEFTZ | SLEFTO | SLEFTC | SRIGHTZ | SRIGHTO | SRIGHTC | RLEFT | RRIGHT | LOADP | OUTPUTP** >

Examples

STIMULUS:VECTOR 1;COUNT 4;AMACRO:FIELD ADDR;PATTERN?
NONA,INC,DEC,XOR

STIM:VEC 1;COUN 4;AMAC:PATT?
NONA,INC,HOLDD,INC

Clearing Stimulus Algorithmic Memory (NON-SCPI)



The STIMulus:;;AMACro:CLEar command clears the algorithmic macro command memory by loading all "NONAlgorithmic" commands into the field specified. The field to be cleared must be defined as an Algorithmic Output (ALGO) field type. Algorithmic commands will be cleared starting at the vector location specified by the VECtor parameter, and will clear the number of vectors specified by the COUNt parameter.

:VECtor <start_vector>

The initial vector location where algorithmic macro commands will be cleared. The starting vector location must be within the range of the size of the test (≤ test_size).

Parameter Definition

start_vector = (1 to test_size)

;COUNt <num_vectors | ALL>

The number of algorithmic macro command vectors that will be cleared. The number of algorithmic macro command vectors can also be specified using the literal string "ALL", where "ALL" is equal to the number of macro commands from the starting vector location to the last macro command in the test. The number of macro commands to be cleared must not exceed the last vector in the test.

Parameter Definition

num_vectors = (1 to ((test_size-start_vector) + 1)

ALL = All algorithmic macro commands from the start_vector location to the last command in the test.

;AMACro

The AMACro command string provides the command path to the FIELd and CLEar strings.

Parameter Definition

none

:FIELd <name>

The optional FIELd parameter specifies the field in which the algorithmic macro commands will be cleared. The field must be an Algorithmic Output (ALGO) field type. If the FIELd parameter is used, then the FIELd and CLEar parameters must be separated by a semicolon. If the FIELd parameter is omitted, then the default stimulus field is assumed. The default stimulus field is defined by the STIMulus:FIELd command.

Parameter Definition **name** = Any alphanumeric string and '_' (max 8 characters).

:CLEar

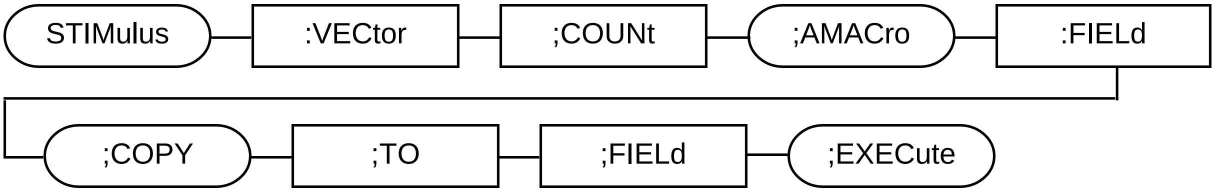
Causes the algorithmic macro commands for the specified field to be cleared to NONA.

Parameter Definition none

Examples

```
STIMULUS:VECTOR 1;COUNT 4;AMACRO:FIELD ADDR;CLEAR
STIM:VEC 1;COUN 4;AMAC:CLE
```

Copying Stimulus Algorithmic Commands (NON-SCPI)



The STIMulus:;;AMACro:;COPY command copies the algorithmic macro commands from a source field into a destination field. AMAcro commands will be copied from the source field starting at the vector location specified by the VECtor parameter, and will copy the number of macro commands specified by the COUNt parameter. The source field must be ALGOutput field type. The destination field can be a ALGOutput or ALGExpected field types.

:VECtor <source_vector>

The initial vector location in the source field where algorithmic macro commands will be copied from. The starting vector must be within the range of the size of the test.

Parameter Definition **source_vector** = (1 to test_size)

;COUNt <num_vectors | ALL>

The number of algorithmic macro commands that will be copied from the source field to the destination field. The number of macro commands can also be specified using the literal string "ALL", where "ALL" is equal to the number of macro commands from the starting vector location to the last vector in the test. The number of macro commands to be copied must not exceed the last vector in the test.

Parameter Definition **num_vectors** = (1 to ((test_size-start_vector) + 1)

ALL = All vectors from the start_vector location to the last vector in the test.

;AMACro

The AMACro command string provides the command path to FIELd and COPY.

Parameter Definition none

:FIELd <source_name>

The FIELd parameter specifies the source ALGO field that algorithmic macro commands will be copied from. The source field must be an Algorithmic Output (ALGO) field type. If the FIELd parameter is omitted, then the default stimulus field is assumed. The default stimulus field is defined by the STIMulus:FIELd command.

Parameter Definition **source_name** = Any alphanumeric string and ‘_’ (max 8 characters).

;COPY

The COPY command string provides the command path to TO, FIELd, and EXECute.

Parameter Definition none

:TO <dest_vector>

The initial vector location in the destination memory field where the algorithmic macro commands will be copied. The destination starting vector must be within the range of the size of the test and must be small enough to allow the entire range of algorithmic commands specified by the source VECTor and COUNt parameters to be transferred.

Parameter Definition **dest_vector** = (1 to test_size)

;FIELd <dest_name>

The FIELd parameter specifies the destination field where the algorithmic macro commands will be copied. The destination field must be an ALGOutput field type. If the FIELd parameter is omitted, then the default stimulus field is assumed. The default stimulus field is defined by the STIMulus:FIELd command.

Parameter Definition **dest_name** = Any alphanumeric string and '_' (max 8 characters).

;EXECute

Executes the algorithmic COPY command.

Parameter Definition none

Examples

STIMULUS:VECTOR 1;COUNT 100;AMACRO:FIELD ADDR;COPY:TO 200;FIELD ADDR;EXECUTE

This command copies 100 algorithmic macro commands from vectors 1 - 100 to vectors 200-299. The source and destination field are the same, the "ADDR" field.

STIM:VECT 50;COUN 10;AMAC:COPY:TO 60;EXECUTE

This command copies 10 macro commands from vectors 50 - 59 to vectors 60 - 69. The source and destination fields are the default field as defined by the STIMulus:FIELd command and assumed to be of type ALGO.

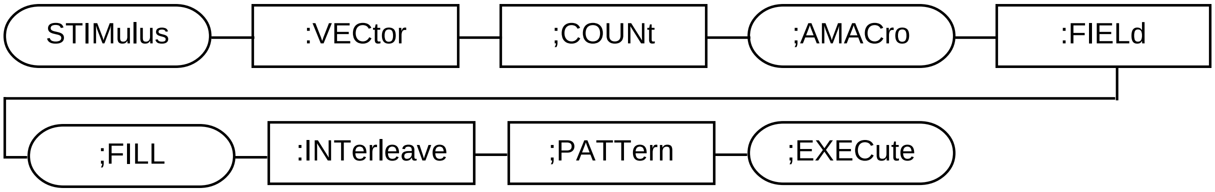
STIMULUS:VECTOR 1;COUNT ALL;AMAC:FIELD ADDR;COPY:TO 1;FIELD DATA;EXECUTE

This command copies all macro commands from the "ADDR" field to the "DATA" field.

STIM:VECT 1;COUN 10;AMAC:COPY:TO 11;EXEC;TO 21;EXEC;TO 31;EXEC;TO 41;EXEC

This command defines a block of 10 algorithmic macro commands from vectors 1 - 10. This macro command block pattern is copied repetitively to vectors 11 - 20, 21 - 30, 31 - 40, and 41 - 50. The source and destination field are the default field as defined by the STIMulus:FIELd command and assumed to be of type ALGO.

Filling Stimulus Algorithmic Memory (NON-SCPI)



The STIMulus:;;AMACro:;FILL command loads the algorithmic macro command memory with a specified macro command. The macro command to be "filled" is specified by the PATTern parameter. The FILL command will begin loading at the vector location specified by the VECtor parameter, and will load the number of macro commands specified by the COUNt parameter

:VECtor <start_vector>

The initial vector location in the destination field where algorithmic macro commands will start loading. The starting vector location must be within the range of the size of the test.

Parameter Definition

start_vector = (1 to test_size)

;COUNt <num_vectors | ALL>

The number of macro commands that will be loaded to memory. The number of macro commands can also be specified using the literal string "ALL", where "ALL" is equal to the number of vectors from the starting vector location to the last vector in the test. The number of vectors to be loaded must not exceed the last vector in the test.

Parameter Definition

num_vectors = (1 to ((test_size-start_vector) + 1)

ALL = All vectors from the start_vector location to the last vector in the test.

;AMACro

The AMACro command string provides the command path to the FIELd and FILL strings.

Parameter Definition

none

:FIELd <name>

The FIELd parameter specifies the destination field where algorithmic command macros will be loaded. The destination field must be an ALGOutput field type. If the FIELd parameter is omitted, then the default stimulus field is assumed. The default stimulus field is defined by the STIMulus:FIELd command.

Parameter Definition

name = Any alphanumeric string and '_' (max 8 characters).

;FILL

The FILL command string provides the command path to INTerleave, PATTern, and EXECute strings.

Parameter Definition

none

:INTerleave <int_count>

The INTerleave parameter specifies the interval count of the algorithmic command vector locations to be "filled". For example, if the interleave count is set to two (2), then every other vector location will be loaded with the algorithmic command specified. Likewise, if the interleave count is set to ten (10), then every tenth vector location will be loaded with the defined command. The default value for int_count is 1.

Parameter Definition **int_count** = (1 - 10)

:PATTern <alg_macro>

The alg_macro parameter is the algorithmic macro command that will be loaded to the destination field. The default algorithmic macro command is the NONAlgorithmic command, which will output the data pattern that is in output memory. See the STIMulus:;AMACro::PATTern command for a description of the algorithmic macro commands.

Parameter Definition **algo_macro** = < **NONAlgorithmic | INCrement | DECrement | XOR | HOLDData | HOLDAll | SLEFTZero | SLEFTOne | SLEFTComplement | SRIGHTZero | SRIGHTOne | SRIGHTComplement | RLEFT | RRIGHT | LOADParam | OUTPUTParam** >

;EXECute

Executes the memory FILL command.

Parameter Definition none

Examples

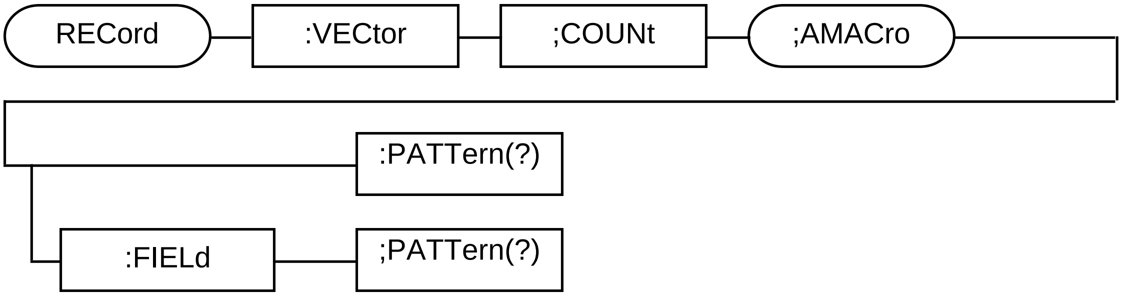
```
STIMULUS:VECTOR 1;COUNT 100;AMACRO:FIELD
ADDR;FILL:INTERLEAVE 1;PATTERN INCREMENT;EXECUTE
```

This command fills 100 vector locations, starting at vector location 1, with the macro command INCREMENT

```
STIM:VECT 1;COUN ALL;AMAC:FILL:PATT NONA;EXECUTE
```

This command fills all vector locations with the macro command NONAlgorithmic.

Response Algorithmic Pattern Generation (NON-SCPI)



The RECord:;;AMACro:PATTern command loads algorithmic macro commands to the specified field. The field to be loaded must be defined as an Algorithmic Expect (ALGE) field type. Algorithmic commands (patterns) will be loaded to the record field starting at the vector location specified by the VECtor parameter, and will load the number of algorithmic commands specified by the COUNt parameter. The RECord:;;AMACro:PATTern? query command returns the algorithmic commands from the specified field for the range defined by the VECTor and COUNt parameters.

:VECtor <start_vector>

The initial vector location where algorithmic macro commands will start loading/querying. The starting vector location must be within the range of the size of the test.

Parameter Definition **start_vector** = (1 to test_size)

;COUNt <num_vectors | ALL>

The number of algorithmic macro commands that will be loaded/queried. The number of macro commands can also be specified by the literal string "ALL", where "ALL" is equal to the number of vectors from the starting vector location to the last vector in the test. The number of macro commands must not exceed the last vector in the test.

Parameter Definition **num_vectors** = (1 to ((test_size-start_vector) + 1)

ALL = All vectors from the start_vector location to the last vector in the test.

;AMACro

The AMACro command string provides a path to FIELd and PATTern parameters.

Parameter Definition none

:FIELd <name>

The optional FIELd parameter specifies the field where the algorithmic macro commands will be loaded to (or queried from). The destination field must be an Algorithmic Expect (ALGE) field type. If the FIELd parameter is used, then the FIELd and PATTern parameters must be separated by a semicolon. If the FIELd parameter is omitted, then the default record field is assumed. The default record field is defined by the RECord:FIELd command.

Parameter Definition **name** = Any alphanumeric string and '_' (max 8 characters).

Note

The FIELd parameter changes the destination field only for the same command but **does not** change the default field.

:PATTern <alg_macro>{,alg_macro}

Specifies the algorithmic command(s) that will be loaded to the ALGE field. The default algorithmic command is the NONAlgorithmic command The number of alg_macro elements must be equal to num_vectors. If a count mismatch occurs, the macro commands will be loaded up to the number of alg_macro elements provided in the command string, or until the num_vectors count is reached, whichever is less. An error message will be generated and the "ERROR" LED on the front panel of the SR2510 module will be lit. Refer to the SYSTem:ERRor? query command for information about reading command errors.

Parameter Definition **alg_macro** = The following is a list of valid algorithmic macro commands.

1. NONAlgorithmic
2. INCrement
3. DECrement
4. XOR
5. HOLDData
6. HOLDAll
7. SLEFTZero
8. SLEFTOne
9. SLEFTComplement
10. SRIGHTZero
11. SRIGHTOne
12. SRIGHTComplement
13. RLEFT
14. RRIGHT
15. LOADParam
16. OUTPUTParam

Algorithmic Expect Command Definitions

NONAlgorithmic

The NONAlgorithmic command allows the Response Gate Arrays to act as a pass through for data from RAM to the expect comparators. The data which is passed from RAM is also used to initialize the algorithmic register. This register may be acted upon by other algorithmic commands to modify the data content.

INCrement

Increment the contents of the algorithmic register and pass the results to the expect comparators. If algorithmic fields greater than 8 bits are used, multiple gate arrays are interlinked. If an increment instruction causes an overflow in one gate array, the overflow is used as a carry input to the next most significant gate array, thus extending the count up to a maximum of 2^{32} before roll over.

DECrement

Decrement the contents of the algorithmic register and pass the results to the expect comparators. If algorithmic fields greater than 8 bits are used, multiple gate arrays are interlinked. If a decrement instruction causes an underflow in one gate array, the underflow is used as a borrow input to the next most significant gate array, thus extending the count up to a maximum of 2^{32} before roll over.

XOR

The XOR instruction will perform a bitwise exclusive ORing of the algorithmic register with the contents of Expect RAM for that vector. In this case the RAM data acts as a modifier to the algorithmic register, but does not directly load it, thus allowing selective bits of the algorithmic register to be complemented before being passed to the expect comparators.

HOLDData

Instructs the Response Gate Arrays to hold the state of the algorithmic register from the previous vector on the expect comparators. This command affects only the expect data, Dontcare masks for the vector are still provided from the Dontcare RAM.

HOLDAll

The HOLDAll command instructs the Response Gate Arrays to hold the state of the algorithmic register and the mask control from the previous vector on the expect comparators. This command is similar to the HOLDData command, however both the expect data and dontcare control are affected.

SLEFTZero

Shift the contents of the algorithmic register left (LSB to MSB) one bit, fill the LSB with 0 and pass the results to the expect comparators. If algorithmic fields greater than 8 bits are used, multiple gate arrays are interlinked. In this case, the MSB output of a less significant gate array is used as a LSB input to the next most significant gate array, thus extending the shift to a maximum 32 bits.

SLEFTOne

Shift the contents of the algorithmic register left (LSB to MSB) one bit, fill the LSB with 1 and pass the results to the expect comparators. If algorithmic fields greater than 8 bits are used, multiple gate arrays are interlinked. In this case, the MSB output of a less significant gate array is used as a LSB input to the next most significant gate array, thus extending the shift to a maximum 32 bits.

SLEFTComplement

Shift the contents of the algorithmic register left (LSB to MSB) one bit, complement the LSB and pass the results to the expect comparators. If algorithmic fields greater than 8 bits are used, multiple gate arrays are interlinked. In this case, the MSB output of a less significant gate array is used as a LSB input to the next most significant gate array, thus extending the shift to a maximum 32 bits.

RLEFT

Rotate the contents of the algorithmic register left (LSB to MSB) one bit, wrap the MSB to the LSB and pass the results to the expect comparators. If algorithmic fields greater than 8 bits are used, multiple gate arrays are interlinked. In this case, the MSB output of a less significant gate array is used as a LSB input to the next most significant gate array and the MSB of the most significant gate array is wrapped to the LSB of the least significant gate array, thus extending the rotation to a maximum 32 bits.

SRIGHTZero

Shift the contents of the algorithmic register right (MSB to LSB) one bit, fill the MSB with 0 and pass the results to the expect comparators. If algorithmic fields greater than 8 bits are used, multiple gate arrays are interlinked. In this case, the LSB output of a more significant gate array is used as a MSB input to the next least significant gate array, thus extending the shift to a maximum 32 bits.

SRIGHTOne

Shift the contents of the algorithmic register right (MSB to LSB) one bit, fill the MSB with 1 and pass the results to the expect comparators. If algorithmic fields greater than 8 bits are used, multiple gate arrays are interlinked. In this case, the LSB output of a more significant gate array is used as a MSB input to the next least significant gate array, thus extending the shift to a maximum 32 bits.

SRIGHTComplement

Shift the contents of the algorithmic register right (MSB to LSB) one bit, complement the MSB and pass the results to the expect comparators. If algorithmic fields greater than 8 bits are used, multiple gate arrays are interlinked. In this case, the LSB output of a more significant gate array is used as a MSB input to the next least significant gate array, thus extending the shift to a maximum 32 bits.

RRIGHT

Rotate the contents of the algorithmic register right (MSB to LSB) one bit, wrap the LSB to the MSB and pass the results to the expect comparators. If algorithmic fields greater than 8 bits are used, multiple gate arrays are interlinked. In this case, the LSB output of a more significant gate array is used as a MSB input to the next least significant gate array and the LSB of the least significant gate array is wrapped to the MSB of the most significant gate array, thus extending the rotation to a maximum 32 bits.

LOADParam

The LOADParm command loads the contents of Expect RAM into an algorithmic holding register internal to the Response Gate Array. This value may later be passed to the algorithmic register, and placed on the expect comparators, using the OUTPUTParam command. This allows test subroutines to be passed a data pattern to use within the routine, such as a starting address for a RAM read or write cycle. Since the data pattern is not actually passed to the expect comparators until the OUTPUTParam command is executed, the subroutine may compare against a standard initialization pattern before using the passed pattern.

OUTPUTParm

Passes the data pattern loaded into the algorithmic holding register by the LOADParam command, to the algorithmic register and the expect comparators. This allows test subroutines to be passed a data pattern to use within the routine, such as a starting address for a microprocessor bus cycle. Since the data pattern is not actually passed to the expect comparators until the OUTPUTParam command is executed, the subroutine may compare against a standard initialization pattern before using the passed pattern.

Examples

```
RECORD:VECTOR 1;COUNT 1;AMACRO:FIELD ADDR;PATTERN  INCREMENT
REC:VEC 1;COUN 1;AMAC:PATT INC
REC:VEC 1;COUN 4;AMAC:PATT NONA,INC,DEC,XOR
```

:PATTern?

The alg_macro parameter is the algorithmic macro command that will be read from the algorithmic expect field. See the RECord:;AMACro::PATTern command for a description of the algorithmic macro commands.

Response

alg_macro{,alg_macro}

Parameter Definition

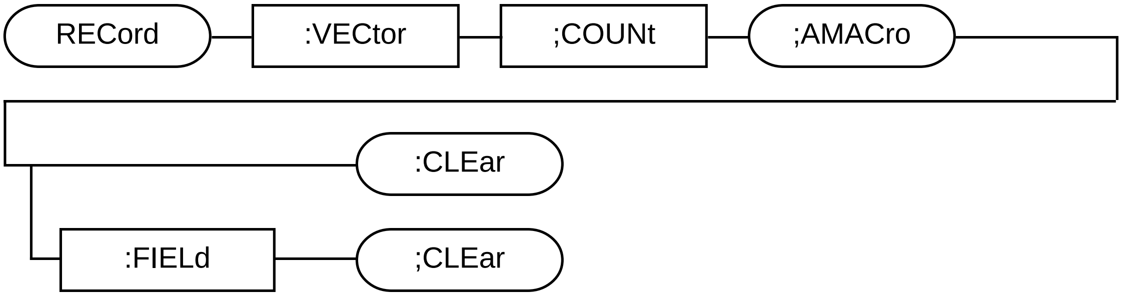
algo_macro = < **NONA | INC | DEC | XOR | HOLDD | HOLDA | SLEFTZ | SLEFTO | SLEFTC | SRIGHTZ | SRIGHTO | SRIGHTC | RLEFT | RRIGHT | LOADP | OUTPUTP** >

Examples

RECORD:VECTOR 1;COUNT 4;AMACRO:FIELD ADDR;PATTERN?
NONA,INC,DEC,XOR

REC:VEC 1;COUN 4;AMAC:PATT?
NONA,INC,HOLDD,INC

Clearing Response Algorithmic Memory (NON-SCPI)



The RECord:;;AMACro:CLEar command clears the algorithmic macro command memory by loading all "NONAlgorithmic" commands into the field specified. The field to be cleared must be defined as an Algorithmic Expect (ALGE) field type. Algorithmic commands will be cleared starting at the vector location specified by the VECtor parameter, and will clear the number of vector specified by the COUNt parameter.

:VECtor <start_vector>

The initial vector location where algorithmic macro commands will be cleared. The starting vector location must be within the range of the size of the test.

Parameter Definition **start_vector** = (1 to test_size)

;COUNt <num_vectors | ALL>

The number of algorithmic macro command vectors that will be cleared. The number of algorithmic macro command vectors can also be specified using the literal string "ALL", where "ALL" is equal to the number of macro commands from the starting vector location to the last macro command in the test. The number of macro commands to be cleared must not exceed the last vector in the test.

Parameter Definition **num_vectors** = (1 to ((test_size-start_vector) + 1)

ALL = All algorithmic macro commands from the start_vector location to the last command in the test.

;AMACro

The AMACro command string provides the command path to the FIELd and CLEar strings.

Parameter Definition none

:FIELd <name>

The optional FIELd parameter specifies the field in which the algorithmic macro commands will be cleared. The field must be an Algorithmic Expect (ALGE) field type. If the FIELd parameter is used, then the FIELd and CLEar parameters must be separated by a semicolon. If the FIELd parameter is omitted, then the default record field is assumed. The default record field is defined by the RECord:FIELd command.

Parameter Definition **name** = Any alphanumeric string and '_' (max 8 characters).

:CLEar

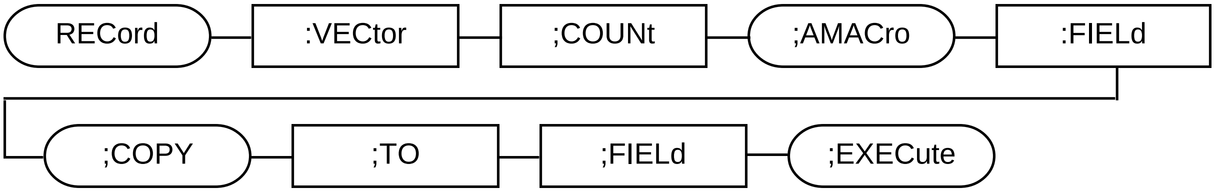
Causes the algorithmic macro commands for the specified field to be cleared to NONA.

Parameter Definition none

Examples

```
RECORD:VECTOR 1;COUNT 4;AMACRO:FIELD ADDR;CLEAR
REC:VEC 1;COUN 4;AMAC:CLE
```

Copying Response Algorithmic Commands (NON-SCPI)



The RECord:;;AMACro:;COPY command copies the algorithmic macro commands from a source field into a destination field. AMAcro commands will be copied from the source field starting at the vector location specified by the VECtor parameter, and will copy the number of macro commands specified by the COUNt parameter. The source field and destination field must be ALGExpected field types.

:VECtor <source_vector>

The initial vector location in the source field where algorithmic macro commands will be copied from. The starting vector must be within the range of the size of the test.

Parameter Definition

source_vector = (1 to test_size)

;COUNt <num_vectors | ALL>

The number of algorithmic macro commands that will be copied from the source field to the destination field. The number of macro commands can also be specified using the literal string "ALL", where "ALL" is equal to the number of macro commands from the starting vector location to the last vector in the test. The number of macro commands to be copied must not exceed the last vector in the test.

Parameter Definition

num_vectors = (1 to ((test_size-start_vector) + 1)

ALL = All vectors from the start_vector location to the last vector in the test.

;AMACro

The AMACro command string provides the command path to FIELd and COPY.

Parameter Definition

none

:FIELd <source_name>

The FIELd parameter specifies the source ALGE field that algorithmic macro commands will be copied from. The source field must be an Algorithmic Expect (ALGE) field type. If the FIELd parameter is omitted, then the default record field is assumed. The default record field is defined by the RECord:FIELd command.

Parameter Definition

source_name = Any alphanumeric string and ‘_’ (max 8 characters).

The COPY command string provides the command path to TO, FIELd,

;COPY

and EXECute.

Parameter Definition

none

:TO <dest_vector>

The initial vector location in the destination field where the algorithmic macro commands will be copied. The destination starting vector must be within the range of the size of the test (£ test_size) and must be small enough to allow the entire range of algorithmic commands specified by the source VECTor and COUNt parameters to be transferred.

Parameter Definition

dest_vector = (1 to test_size)

;FIELd <dest_name>

The FIELd parameter specifies the destination field where the algorithmic macro commands will be copied. The destination field must be an ALGExpected field type. If the FIELd parameter is omitted, then the default record field is assumed. The default record field is defined by the RECord:FIELd command.

Parameter Definition

dest_name = Any alphanumeric string and '_' (max 8 characters).

;EXECute

Executes the algorithmic COPY command.

Parameter Definition

none

Examples

```
RECORD:VECTOR 1;COUNT 100;AMACRO:FIELD ADDR;COPY:TO
200;FIELD ADDR;EXECUTE
```

This command copies 100 algorithmic macro commands from vectors 1 - 100 to vectors 200-299. The source and destination field are the same, the "ADDR" field.

```
REC:VECT 50;COUN 10;AMAC:COPY:TO 60;EXECUTE
```

This command copies 10 macro commands from vectors 50 - 59 to vectors 60 - 69. The source and destination fields are the default field as defined by the RECord:FIELd command and assumed to be of type ALGE.

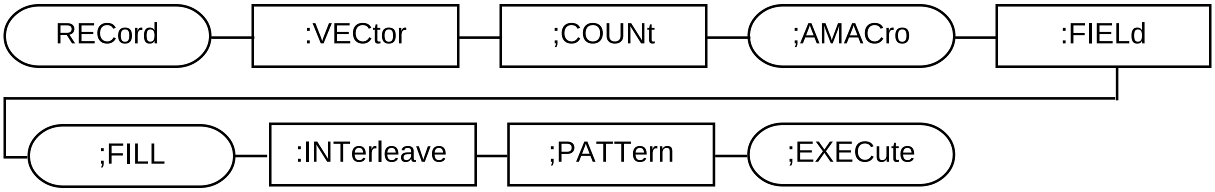
```
RECORD:VECTOR 1;COUNT ALL;AMAC:FIELD ADDR;COPY:TO 1;FIELD
DATA;EXECUTE
```

This command copies all macro commands from the "ADDR" field to the "DATA" field.

```
REC:VECT 1;COUN 10;AMAC:COPY:TO 11;EXEC;TO 21;EXEC;TO
31;EXEC;TO 41;EXEC
```

This command defines a block of 10 algorithmic macro commands from vectors 1 - 10. This macro command block pattern is copied repetitively to vectors 11 - 20, 21 - 30, 31 - 40, and 41 - 50. The source and destination field are the default field as defined by the RECord:FIELd command and assumed to be of type ALGE.

Filling Response Algorithmic Memory (NON-SCPI)



The RECord:;;AMACro:;FILL command loads the algorithmic macro command memory with a specified macro command. The macro command to be "filled" is specified by the PATTern parameter. The FILL command will begin loading at the vector location specified by the VECtor parameter, and will load the number of macro commands specified by the COUNt parameter

:VECtor <start_vector>

The initial vector location in the destination field where algorithmic macro commands will start loading. The starting vector location must be within the range of the size of the test.

Parameter Definition **start_vector** = (1 to test_size)

;COUNt <num_vectors | ALL>

The number of macro commands that will be loaded to memory. The number of macro commands can also be specified using the literal string "ALL", where "ALL" is equal to the number of vectors from the starting vector location to the last vector in the test. The number of vectors to be loaded must not exceed the last vector in the test.

Parameter Definition **num_vectors** = (1 to ((test_size-start_vector) + 1)

ALL = All vectors from the start_vector location to the last vector in the test.

;AMACro

The AMACro command string provides the command path to the FIELd and FILL strings.

Parameter Definition none

:FIELd <name>

The FIELd parameter specifies the destination field where algorithmic command macros will be loaded. The destination field must be an ALGExpect field type. If the FIELd parameter is omitted, then the default record field is assumed. The default record field is defined by the RECord:FIELd command.

Parameter Definition **name** = Any alphanumeric string and '_' (max 8 characters).

;FILL

The FILL command string provides the command path to INTerleave, PATTern, and EXECute strings.

Parameter Definition none

:INTerleave <int_count>

The INTerleave parameter specifies the interval count of the algorithmic command vector locations to be "filled". For example, if the interleave count is set to two (2), then every other vector location will be loaded with the algorithmic command specified. Likewise, if the interleave count is set to ten (10), then every tenth vector location will be loaded with the defined command. The default value for int_count is 1.

Parameter Definition **int_count** = (1 - 10)

:PATTern <alg_macro>

The alg_macro parameter is the algorithmic macro command that will be loaded to the destination field. The default algorithmic macro command is the NONAlgorithmic command, which will use the data pattern that is in expect memory for the real-time compare. See the RECord:;AMACro::PATTern command for a description of the algorithmic macro commands.

Parameter Definition **algo_macro** = < **NONAlgorithmic | INCrement | DECrement | XOR | HOLDData | HOLDAll | SLEFTZero | SLEFTOne | SLEFTComplement | SRIGHTZero | SRIGHTOne | SRIGHTComplement | RLEFT | RRIGHT | LOADParam | OUTPUTParam** >

;EXECute

Executes the memory FILL command.

Parameter Definition none

Examples

```
RECORD:VECTOR 1;COUNT 100;AMACRO:FIELD
ADDR;FILL:INTERLEAVE 1;PATTERN INCREMENT;EXECUTE
```

This command fills 100 vector locations, starting at vector location 1, with the macro command INCREMENT

```
REC:VECT 1;COUN ALL;AMAC:FILL:PATT NONA;EXECUTE
```

This command fills all vector locations with the macro command NONAlgorithmic.

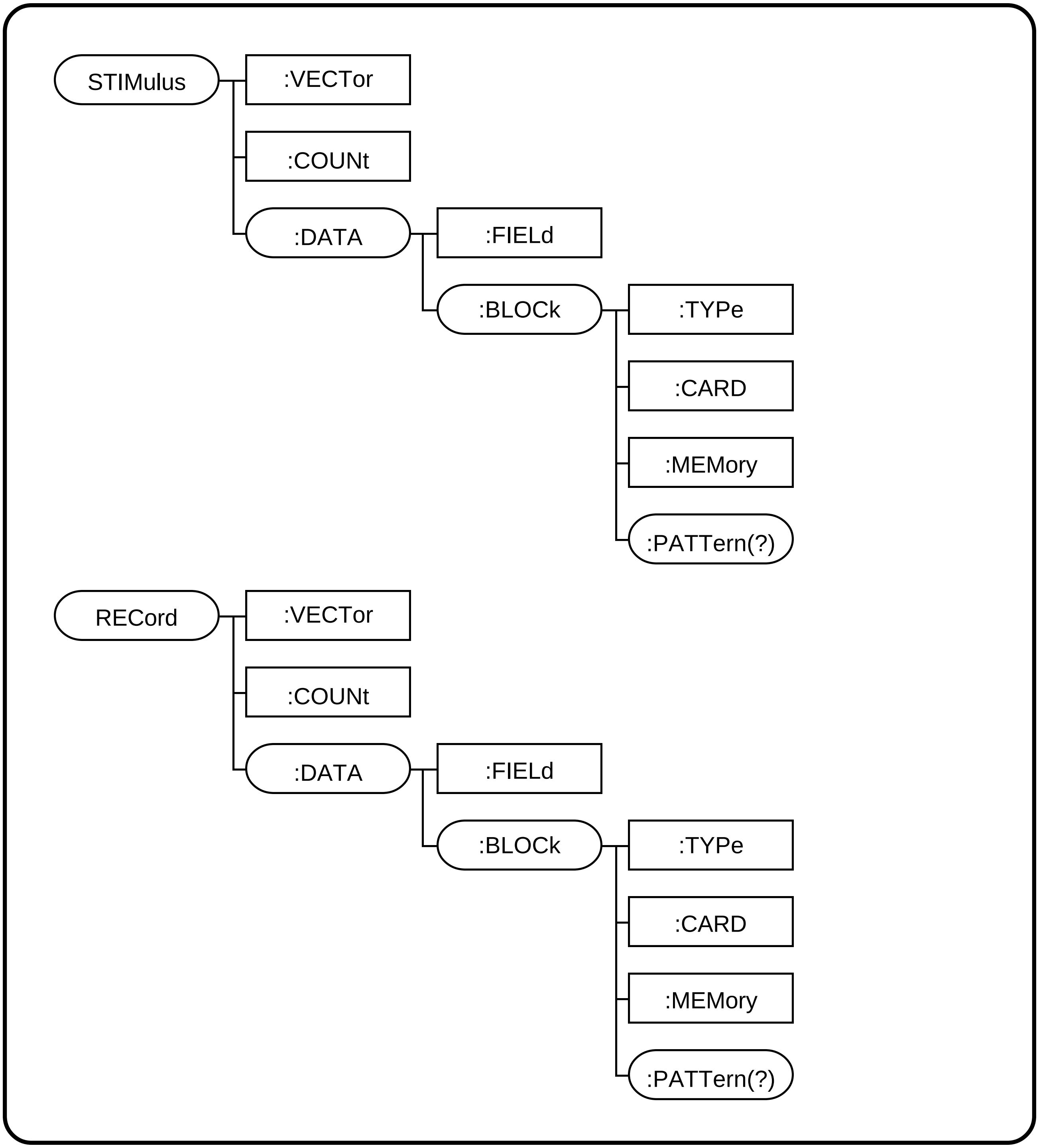
THIS PAGE INTENTIONALLY LEFT BLANK

High Speed Binary Pattern Transfers

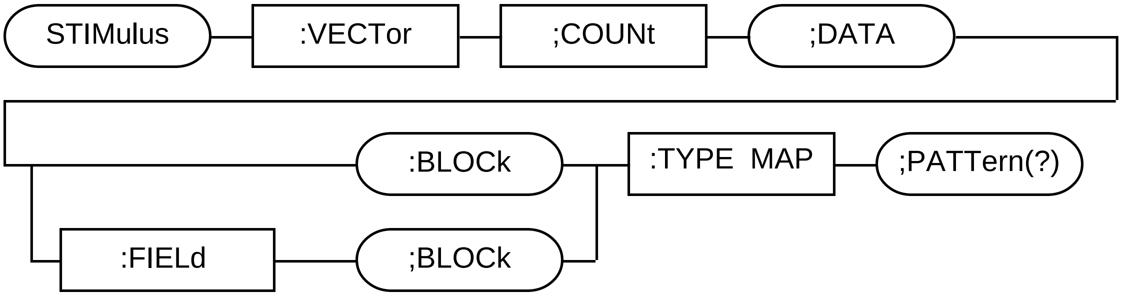
Data patterns may be loaded into memory in binary format instead of using the SCPI command strings. This provides a significant speed advantage over text based transfers for 3 reasons. In ASCII text based transfers, each character transferred requires multiple VXI read/write cycles. This is due to the VXI Word Serial Protocols which implement a multi-cycle handshake for each transfer. The binary transfer is accomplished via direct VME/VXI reads and writes, so a new data pattern is transferred on each bus cycle. Second, each text character transferred represents only a maximum of 4 bits of data. Binary transfers are performed using D32, so each transfer represents a full 32 bits of data. Finally, text transfers must be processed in order to determine where the data will ultimately be sent. Parsing and processing the command can require a significant amount of microprocessor overhead. Binary transfers do not require any parsing or processing.

In order to transfer data in binary format you must first load the 1 MByte A32 memory on the SR2510 Timing/Control Board with the desired data. This process is performed by the slot 0 controller using the memory move function which are typically provided with the slot 0. The data is always represented as 32 bits, regardless of how many bits are actually used, however, the data may be transferred from the slot 0 to the A32 memory using D8, D16 or D32 transfers. The A32 memory is assigned an offset address during the VXI power-up by the slot 0 Resource Management routines. This offset address is where the slot 0 must send the binary data.

Next you would send the SCPI command instruction the SR2510 Timing/ Control Board to transfer the data in its A32 memory to I/O memory. There are two methods for the Timing/Control Board to transfer the data, MAP and NOMAP. The MAP method uses field definitions to determine pin mapping. This requires microprocessor overhead, so is not as fast as the NOMAP method. The NOMAP method transfers data to a specific pattern memory on a specific I/O module. There is no pin mapping process, so the transfers are much faster. All data transferred between the SR2510 A32 memory and the I/O module pattern memories use D32 transfers.


STIMulus
:VECTor
:COUNt
:DATA
:FIELd
:BLOCk
:TYPe
:CARD
:MEMory
:PATTern(?)
RECord
:VECTor
:COUNt
:DATA
:FIELd
:BLOCk
:TYPe
:CARD
:MEMory
:PATTern(?)

Stimulus Mapped Binary Patterns (NON-SCPI)



The STIMulus:;;:BLOCk:TYPE MAP;PATTern command downloads the contents of the SR2510 A32 memory into the specified I/O module stimulus type memory field using the Pin Mapping method. Binary data patterns will be loaded starting at the vector location specified by the VECtor parameter, and will load the number of vector words specified by the COUNt parameter. The STIMulus:;;:BLOCk:TYPE MAP; PATTern? query command uploads the binary data pattern from the specified stimulus memory field into the SR2510 A32 memory using the Pin Mapping method.

:VECtor <start_vector>

The initial vector location where data will start transferring to/from stimulus memory. The starting vector must be within the range of the size of the test.

Parameter Definition **start_vector** = (1 to test_size)

;COUNt <num_vectors | ALL>

The number of vector words that will be transferred to/from memory. The number of vectors can also be specified by the literal string "ALL", where "ALL" is equal to the number of vectors from the starting vector location to the last vector in the test. The number of vectors to be transferred must not exceed the last vector in the test.

Parameter Definition **num_vectors** = (1 to ((test_size - start_vector) + 1))

ALL = All vectors from the start_vector location to the last vector in the test.

;DATA

The DATA command string provides the command path to the BLOCk parameter.

Parameter Definition none

:FIELd <name>

The FIELd parameter specifies the memory field where data patterns will be loaded to/from. Valid field types for the source FIELd parameter are Output (OUT), Tristate (TRI), Algorithmic Output (ALGO), Hardware Output, (HOUT), and Hardware Tristate (HTRI).

Hardware Type fields (HOUT and HTRI) will be downloaded using the same block transfer mode as the NOMAP Block transfer method since Hardware Type fields are unmapped (defined on physical pin-card boundaries). Output/Tristate (OT) type fields are not valid for A32 block transfers.

If the FIELd parameter option is used, then the FIELd and BLOCk parameters must be separated by a semicolon as shown in the example below. If the FIELd parameter is omitted, then the default memory field is assumed. The default memory field is defined by the STIMulus:FIELd command. The FIELd parameter changes the destination field only for the same command but **does not** change the default field.

Parameter Definition **name** = Any alphanumeric string and ‘_’ (max 8 characters).

:BLOCk

The BLOCk command string provides the command path to the TYPE and PATTERN parameters.

Parameter Definition none

:TYPE MAP

The TYPE parameter defines the method that data patterns will be transferred. The MAP option uses field definitions to determine pin mapping.

Parameter Definition **MAP** = Pin Mapping transfer method.

;PATTern

The PATTern string terminates the command and executes the block transfer from A32 memory to the stimulus memory field.

Parameter Definition none

Examples

```
STIMULUS:VECTOR 1;COUNT 1000;DATA:FIELD ADDR;BLOCK:TYPE
MAP;PATTERN
STIM:VECT 1;COUN ALL;DATA:BLOCK:TYPE MAP;PATT
```

:PATTern?

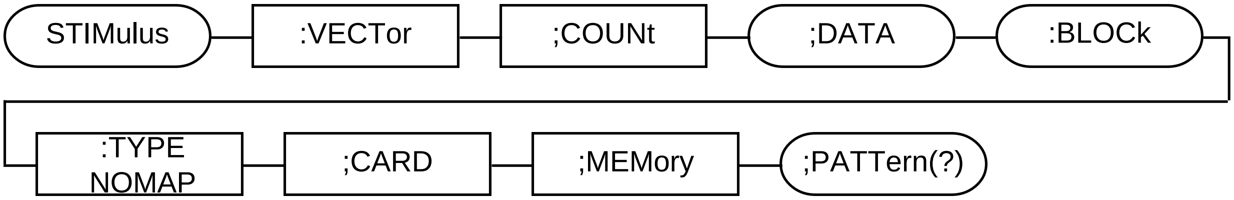
The PATTern string terminates the command and executes the block transfer from the stimulus memory field to the SR2510 A32 memory..

Response none

Examples

```
STIMULUS:VECTOR 1;COUNT 1000;DATA:FIELD ADDR;BLOCK:TYPE
MAP;PATTERN?
STIM:VEC 1;COUN ALL;DATA:BLOCK:TYPE MAP;PATT?
```

Stimulus Non-Mapped Binary Patterns (NON-SCPI)



The STIMulus:;;:BLOCk:TYPE NOMAP;PATTern command downloads the contents of the SR2510 A32 memory into the specified I/O module stimulus type memory using the Hardware Mapping (NOMAP) method. Binary data patterns will be loaded to the Output or Tristate memory starting at the vector location specified by the VECtor parameter, and will load the number of vector words specified by the COUNt parameter.

The STIMulus:;;:BLOCk:TYPE NOMAP; PATTern? query command uploads the binary data pattern from the specified Output or Tristate memory into the SR2510 A32 memory using the Hardware Mapping (NOMAP) method.

:VECtor <start_vector>

The initial vector location where data will start transferring to/from stimulus memory. The starting vector must be within the range of the size of the test.

Parameter Definition **start_vector** = (1 to test_size)

;COUNt <num_vectors | ALL>

The number of vector words that will be transferred to/from memory. The number of vectors can also be specified by the literal string "ALL", where "ALL" is equal to the number of vectors from the starting vector location to the last vector in the test. The number of vectors to be transferred must not exceed the last vector in the test.

Parameter Definition **num_vectors** = (1 to ((test_size - start_vector) + 1))

ALL = All vectors from the start_vector location to the last vector in the test.

;DATA

The DATA command string provides the command path to the BLOCk parameter.

Parameter Definition none

:BLOCk

The BLOCk command string provides the command path to the TYPE ,CARD, MEMory, and PATTERN parameters.

Parameter Definition none

:TYPE NOMAP

The TYPE parameter defines the method that data patterns will be transferred. The NOMAP option uses card number and memory type for download source/destination.

Parameter Definition

NOMAP = Hardware Mapping transfer method.

;CARD <card_num | ALL>

The CARD parameter defines the source or destination I/O module (card) that data patterns will transferred to/from. The card number can also be specified by the literal string "ALL", where the contents of the SR2510 A32 memory will be transferred (memory write) to "ALL" the I/O cards. If "ALL" I/O cards are selected for transfers to the SR2510 A32 memory (memory read) , then the SR2510 A32 memory will contain only the data contents of the last I/O card.

Parameter Definition

card_num = (1 - 18); up to the maximum number of I/O cards installed in the SR2500 system.

ALL = All I/O cards installed in the SR2500 system.

;MEMory <OUTput | TRIstate>

The MEMory parameter defines the source or destination stimulus memory type that data patterns will transferred to/from.

Parameter Definition

OUTput = Output memory.

TRIstate = Tristate memory.

:PATTern

The PATTern string terminates the command and executes the block transfer from A32 memory to the stimulus memory.

Parameter Definition

none

Examples

```
STIMULUS:VECTOR 1;COUNT 1000;DATA:BLOCK:TYPE NOMAP;CARD 1;MEMORY OUTPUT;PATTERN
STIM:VECT 1;COUN ALL;DATA:BLOC:TYPE NOMAP;CARD 1;MEM OUT;PATT
```

:PATTern?

The PATTern string terminates the command and executes the block transfer from the stimulus memory to the SR2510 A32 memory..

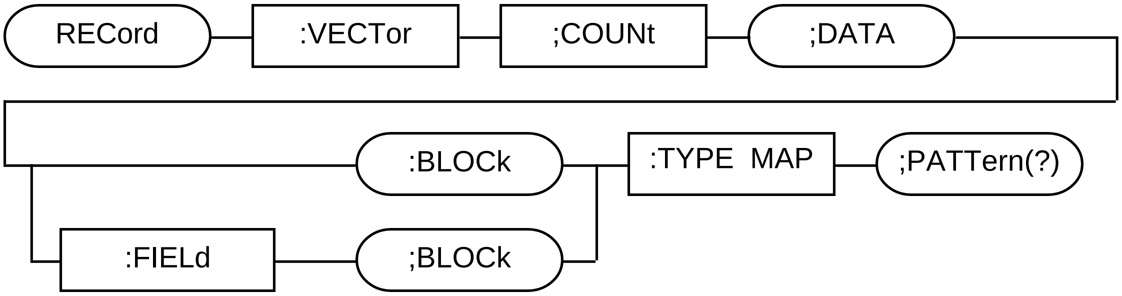
Response

none

Examples

```
STIMULUS:VECTOR 1;COUNT 1000;DATA:BLOCK:TYPE NOMAP;CARD 1;MEMORY OUTPUT;PATTERN?
STIM:VEC 1;COUN ALL;DATA:BLOC:TYPE NOMAP;CARD 1;MEM OUT;PATT?
```

Record Mapped Binary Patterns (NON-SCPI)



The RECord:;;:BLOCk:TYPE MAP;PATTern command downloads the contents of the SR2510 A32 memory into the specified I/O module record type field using the Pin Mapping method. Binary data patterns will be loaded starting at the vector location specified by the VECtor parameter, and will load the number of vector words specified by the COUNt parameter. RECord :;;:BLOCk:TYPE MAP;PATTern? query command uploads the binary data pattern from the specified record memory field into the SR2510 A32 memory using the Pin Mapping method.

:VECtor <start_vector>

The initial vector location where data will start transferring to/from record memory. The starting vector must be within the range of the size of the test.

Parameter Definition **start_vector** = (1 to test_size)

;COUNt <num_vectors | ALL>

The number of vector words that will be transferred to/from record memory. The number of vectors can also be specified by the literal string "ALL", where "ALL" is equal to the number of vectors from the starting vector location to the last vector in the test. The number of vectors to be transferred must not exceed the last vector in the test.

Parameter Definition **num_vectors** = (1 to ((test_size - start_vector) + 1))

ALL = All vectors from the start_vector location to the last vector in the test.

;DATA

The DATA command string provides the command path to the BLOCk parameter.

Parameter Definition none

:FIELd <name>

The FIELd parameter specifies the memory field where data patterns will be loaded to/from. Valid field types for block memory read transfers are Record (REC), Expected (EXP), Dontcare (DON), Algorithmic Expected (ALGE), Hardware Record (HREC), Hardware Expected, (HEXP), and Hardware Dontcare (HDON).

REC and HREC type fields can be uploaded into SR2510 A32 memory, but cannot be downloaded because these field types are 'read only' memory. Hardware type fields (HEXP and HTRI) will be downloaded using the same block transfer mode as the NOMAP Block transfer method since Hardware Type fields are unmapped (defined on physical pin-card boundaries). Expected/Dontcare (ED) type fields are not valid for A32 block transfers.

If the FIELd parameter option is used, then the FIELd and BLOCk parameters must be separated by a semicolon as shown in the example below. If the FIELd parameter is omitted, then the default memory field is assumed. The default memory field is defined by the RECord:FIELd command. The FIELd parameter changes the destination field only for the same command but **does not** change the default field.

Parameter Definition **name** = Any alphanumeric string and '_' (max 8 characters).

:BLOCk

The BLOCk command string provides the command path to the TYPE and PATTERN parameters.

Parameter Definition none

:TYPE MAP

The TYPE parameter defines the method that data patterns will be transferred. The MAP option uses field definitions to determine pin mapping.

Parameter Definition **MAP** = Pin Mapping transfer method.

;PATTern

The PATTern string terminates the command and executes the block transfer from A32 memory to the stimulus memory field.

Parameter Definition none

Examples

```
STIMULUS:VECTOR 1;COUNT 1000;DATA:FIELD ADDR;BLOCK:TYPE
MAP;PATTERN
STIM:VECT 1;COUN ALL;DATA:BLOCK:TYPE MAP;PATT
```

:PATTern?

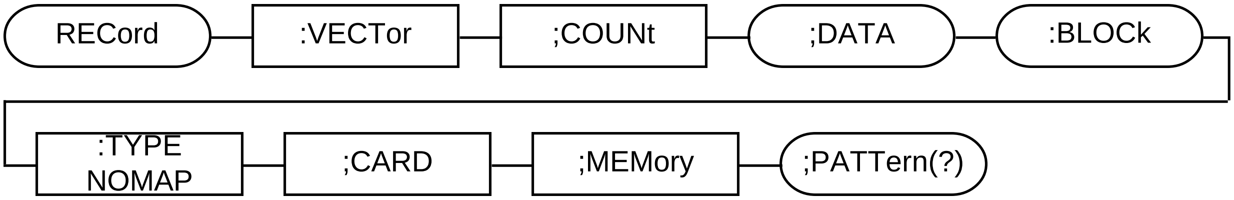
The PATTern string terminates the command and executes the block transfer from the stimulus memory field to the SR2510 A32 memory..

Response none

Examples

```
STIMULUS:VECTOR 1;COUNT 1000;DATA:FIELD ADDR;BLOCK:TYPE
MAP;PATTERN?
STIM:VEC 1;COUN ALL;DATA:BLOCK:TYPE MAP;PATT?
```

Record Non-Mapped Binary Patterns (NON-SCPI)



The RECord:;;:BLOCk:TYPE NOMAP;PATTern command downloads the contents of the SR2510 A32 memory into the specified I/O module record type memory using the Hardware Mapping (NOMAP) method. Binary data patterns will be loaded to the Expect or Dontcare memory starting at the vector location specified by the VECtor parameter, and will load the number of vector words specified by the COUNt parameter.

The STIMulus:;;:BLOCk:TYPE MAP; PATTern? query command uploads the binary data pattern from the specified Expect, Dontcare or Record memory into the SR2510 A32 memory using the Hardware Mapping (NOMAP) method.

:VECtor <start_vector>

The initial vector location where data will start transferring to/from record memory. The starting vector must be within the range of the size of the test (≤ test_size).

Parameter Definition **start_vector** = (1 to test_size)

;COUNt <num_vectors | ALL>

The number of vector words that will be transferred to/from memory. The number of vectors can also be specified by the literal string "ALL", where "ALL" is equal to the number of vectors from the starting vector location to the last vector in the test. The number of vectors to be transferred must not exceed the last vector in the test.

Parameter Definition **num_vectors** = (1 to ((test_size - start_vector) + 1))

ALL = All vectors from the start_vector location to the last vector in the test.

;DATA

The DATA command string provides the command path to the BLOCk parameter.

Parameter Definition none

:BLOCk

The BLOCk command string provides the command path to the TYPE ,CARD, MEMory, and PATTERN parameters.

Parameter Definition none

:TYPE NOMAP

The TYPE parameter defines the method that data patterns will be transferred. The NOMAP option uses card number and memory type for download source/destination.

Parameter Definition **NOMAP** = Hardware Mapping transfer method.

;CARD <card_num | ALL>

The CARD parameter defines the source or destination I/O module (card) that data patterns will transferred to/from. The card number can also be specified by the literal string "ALL", where the contents of the SR2510 A32 memory will be transferred (memory write) to "ALL" the I/O cards. If "ALL" I/O cards are selected for transfers to the SR2510 A32 memory (memory read) , then the SR2510 A32 memory will contain only the data contents of the last I/O card.

Parameter Definition **card_num** = (1 - 18); up to the maximum number of I/O cards installed in the SR2500 system.

ALL = All I/O cards installed in the SR2500 system.

;MEMory <EXPect | DONtcare | RECord>

The MEMory parameter defines the source or destination record memory type that data patterns will transferred to/from.

Parameter Definition **EXPect** = Expect memory.
DONtcare = Dontcare memory.
RECord = Record memory.

:PATTern

The PATTern string terminates the command and executes the block transfer from A32 memory to the record memory.

Parameter Definition none

Examples

```
RECORD:VECTOR 1;COUNT 1000;DATA:BLOCK:TYPE NOMAP;CARD
1;MEMORY EXPECT;PATTERN
REC:VECT 1;COUN ALL;DATA:BLOC:TYPE NOMAP;CARD 1;MEM
EXP;PATT
```

:PATTern?

The PATTern string terminates the command and executes the block transfer from the record memory to the SR2510 A32 memory..

Response none

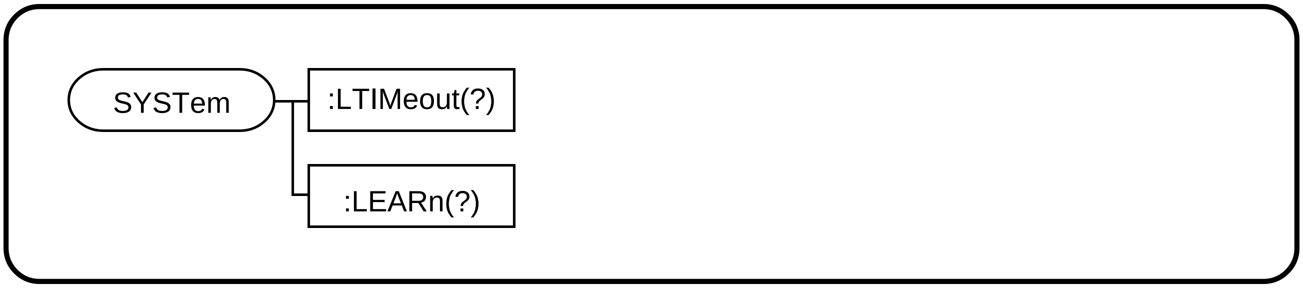
Examples

```
RECORD:VECTOR 1;COUNT 1000;DATA:BLOCK:TYPE NOMAP;CARD
1;MEMORY RECORD;PATTERN?
REC:VEC 1;COUN ALL;DATA:BLOC:TYPE NOMAP;CARD 1;MEM
REC;PATT?
```

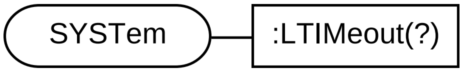
Saving and Loading Tests

Two methods exist to read and save a test from the SR2510 to a slot 0 controller. The first method, which will not be discussed, is to query each individual parameter in the SR2500 system using a SCPI command string, parse the ASCII text response to strip out the relevant information, append the appropriate header to this information, then store the results to a file. The second method is to use the LEARn and LEARn? query commands to read the entire test, via the SR2510 A32 memory. Each time a SCPI command is sent to the SR2510, the system processor must parse the command string in order to determine what action to take. The "action" will, ultimately consists of modifying data at various memory locations within the SR2500 system. In effect, parsing SCPI commands is the same as performing an incremental compile of a SCPI test program. The compiling is taking place on the SR2510, and the results stored on the SR2510 and the I/O modules. LEARn? query and LEARn allow you to read the compiled test program from the SR2500 system, or write a previously read compiled test back to the SR2500 system, respectively.

The LEARn and LEARn? query method provides a significant speed advantage over text based transfers for 3 reasons. First, in text based transfers, each character transferred requires multiple VXI read/write cycles. This is due to the VXI Word Serial Protocols which implement a multi-cycle handshake for each transfer. Using the LEARn and LEARn? query commands, setup and data are read in binary format. The binary transfer is accomplished via direct VME/VXI reads and writes, so new information is transferred on each bus cycle. Second, each text character transferred represents only 4 bits of data. Binary transfers are performed using D32, so each transfer represents a full 32 bits of data. Finally, text transfers must be processed in order to determine where the data will ultimately be sent to or read from. Parsing and processing the command can require a significant amount of microprocessor overhead. Binary transfers do not require any parsing or processing.


SYSTem
:LTIMeout(?)
:LEARn(?)

Binary Learn Time-Out (NON-SCPI)



The SYSTem:LTIMeout command sets the time-out value used in learning system tests. If the transfer of all blocks to or from the SR2510 does not occur in the allotted time, a command error will be generated.

:LTIMout <timeout_value>

This value is the total number of seconds to transfer all blocks of data from the SR2510 to the slot 0 controller (LEARn?) or from the slot 0 controller to the SR2510 (LEARn). The default value is 20 seconds.

Parameter Definition **timeout_value** = (1 to 100)

Examples

SYSTEM:LTIMEOUT 10
SYST:LTIM 1.5e+01

:LTIMout?

Response timeout_value

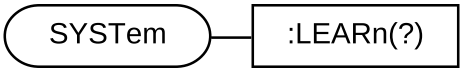
Parameter Definition **timeout_value =** The learn time-out value specified in seconds and represented using scientific notation.

Examples

SYSTEM:LTIMEOUT?
2.000000e+01

SYST:LTIM?
1.500000e+01

Learning Binary Tests (NON-SCPI)



SYSTem :LEARn(?)

The SYSTem:LEARn command instructs the SR2510 to send the current system setup parameters to the slot 0 controller as a binary memory-image file. The SYSTem:LEARn command instructs the SR2510 to receive a previously saved memory-image file from the slot 0 controller. The binary memory-image is sent as multiple blocks of data, each varying in size dependent upon the actual test(s) defined.

:LEARn

Instructs the SR2500 to learn a previously saved system setup. There are no parameters associated with this command. The data content for this command is sent to the SR2510 A32 memory by the slot 0 controller, one block for each handshake cycle, to be read by the SR2510.

Parameters none

Examples

```
SYSTEM:LEARN
SYST:LEAR
```

:LEARn?

Instructs the SR2500 to send the system setup to the slot 0 controller. There are no parameters associated with this command, and none returned. The data content for this command is sent to the SR2510 A32 memory by the SR2510, one block for each handshake cycle, to be read by the slot 0 controller.

Parameters none

Examples

```
SYSTEM:LEARN?
SYST:LEAR?
```

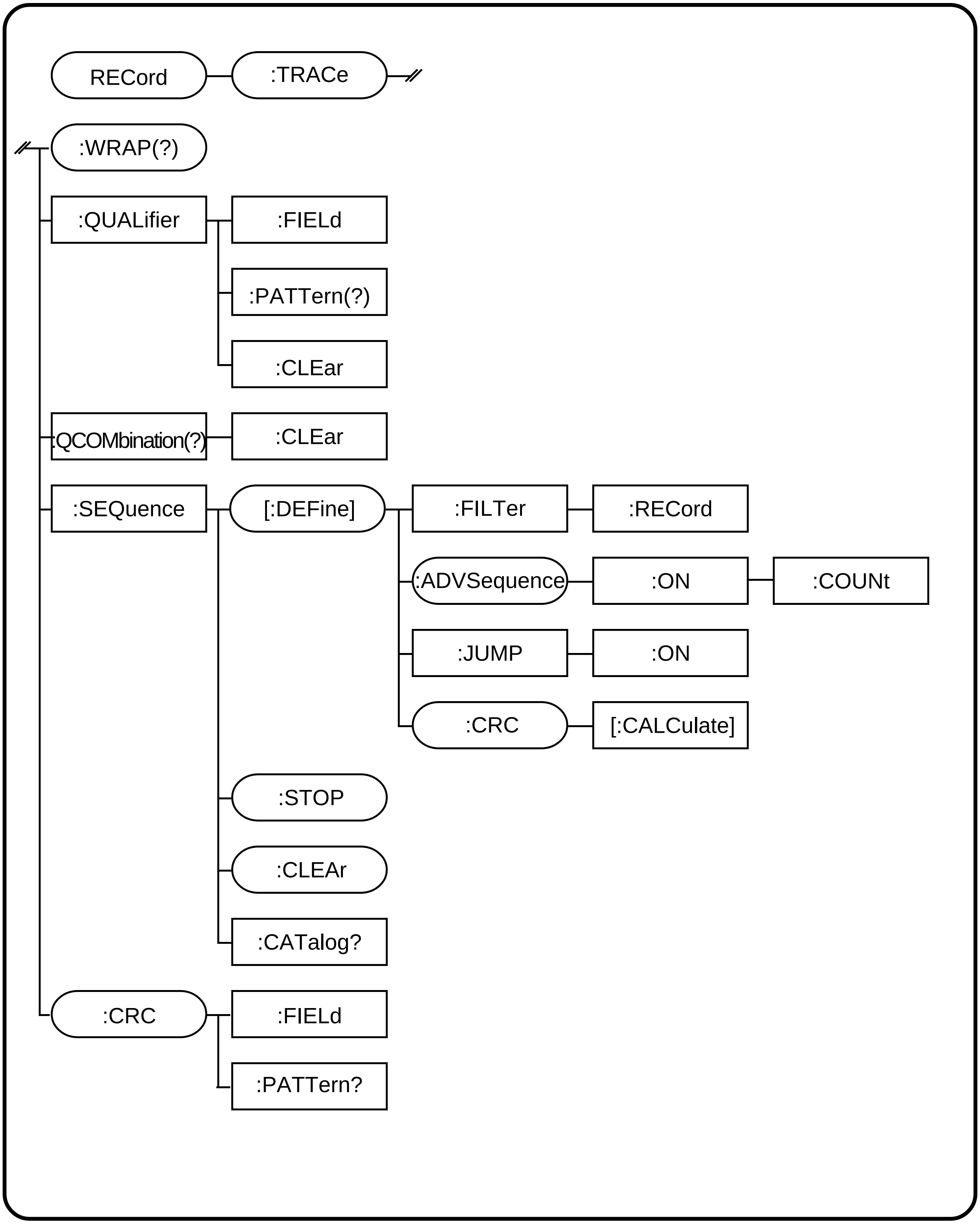
Advanced Record Triggering

The SR2500 employs an advanced record triggering system which controls the type of data stored to record memory, and under what conditions that data is stored. The record triggering logic also controls when input data from the UUT is used in CRC calculations. The SCPI command path which provides access to these control parameters is RECord:TRACe. Trace controls give the SR2500 triggering and recording capabilities very much like a typical Logic Analyzer. A higher level of Trace functions is provided with the Trace Macro commands TMACro:POSTtrigger and TMACro:SEQuence. These functions compile into Trace commands, which may be read using the TRACe:SEQuence:CATalog? query command, and were discussed in section 3.1.

TRACE commands are divided into 3 main subsystems, the Qualifiers (QUAL), the Qualifier Combinations (QCOM) and Sequences (SEQ). Refer to figure 3-1 for a graphic example of how these three systems interact. Qualifiers, simply stated, are trigger match patterns which are compared against data returned by the UUT. Qualifiers are separate from the real-time compare functions, which compare the UUT response with the data stored in the Expect memory. There are 8 qualifier triggers for each record type field defined in the SR2500. A single Qualifier may define trigger patterns for any or all record type fields. When multiple trigger patterns (i.e., field patterns) are defined for a single qualifier, the results of all trigger pattern compares are logically ANDed together. In other words, assume Qualifier 1 was defined with a trigger pattern of #hAA, #hBB and #hCC for fields F1, F2 and F3, respectively. It would require a pattern match of (F1 == #hAA && F2 == #hBB && F3 == #hCC), on the same test cycle, for Qualifier 1 to evaluate TRUE.

Qualifier Combinations are exactly what the name implies, groups of one or more of the 8 Qualifiers Trigger Patterns. Whereas the results of multiple field trigger patterns in a Qualifier are logically ANDed, the results of multiple Qualifiers in a QCOM are logically ORed. To expand on the example above, assume QCOM 1 were defined to consist of Qualifier 1 and Qualifier 2, and that Qualifier 1 was defined with a trigger pattern of #hAA, #hBB and #hCC for fields F1, F2 and F3, respectively, and that Qualifier 2 was defined with a trigger pattern of #hFF, #hEE and #hDD for fields F1, F2 and F3, respectively. It would require a match of (F1 == #hAA && F2 == #hBB && F3 == #hCC) || (F1 == #hFF && F2 == #hEE && F3 == #hDD), for QCOM 1 to evaluate TRUE. Qualifiers and QCOMs are also available for conditional looping and branching evaluation by CMACRO instructions.

The record control functions and CRC control functions are grouped into structures called "Sequences". There are a maximum of 16 sequences for use in controlling the record and CRC processes. Each sequence specifies what to record, when to record it, when to advance to the next sequence level, when to jump out of sequence to a new sequence level, and when the input data will be used in CRC calculations. There is also a global "Stop Test" parameter which allows the record control logic to set a STOP flag when the defined sequence level is reached. The state of this flag is continually polled by the system processor, and when the state indicates a stop condition, the system processor will asynchronously abort the test. Sequences support using the results of the real-time compare and the SR2510 Input Flags for record and CRC control, in addition to the Qualifiers and Qualifier Combinations


RECord
:TRACe
:WRAP(?)
:QUALifier
:FIELd
:PATTern(?)
:CLEAr
:QCOMbination(?)
:CLEAr
:SEQuence
[:DEFine]
:FILTer
:RECord
:ADVSequence
:ON
:COUNt
:JUMP
:ON
:CRC
[:CALCulate]
:STOP
:CLEAr
:CATalog?
:CRC
:FIELd
:PATTern?

Qualifier Trigger Patterns			
QUAL#	ADDR	DATA	R/W*
1	#h2000	#hXX	#b01
2	#h2000	#hXX	#b10
3	#h7FFF	#hXX	#bXX
4			
5			
6			
7			
8			

QCOM	Qualifier Numbers							
	1	2	3	4	5	6	7	8
1	*							
2		*						
3			*					
4								
5								
6								
7								
8								

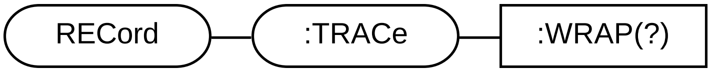
*Read Cycle = #b10; Write Cycle = #b01

Trace Sequences							
	FILTer	RECord	ADVS:ON	COUNt	JUMP:ON	CRC:CAL	STOP
1	Data	QCOM1	QCOM1	1	Never	Never	No
2	Data	Always	QCOM3	1	Never	Never	No
3	Data	Never	QCOM2	1	Never	QCOM2	No
4	Data	Never	QCOM3	1	Never	Always	No
5	Data	Never	Never	1	Never	Never	No
o o o	o o o	o o o	o o o	o o o	o o o	o o o	o o o
16	Data	Never	Never	1	Never	Never	No

Figure 3-1: Trace Qualifiers, Qualifier Combinations and Sequences

This example demonstrates how the Record Trace functions might be used in a RAM test. Assuming the SR2500 is programmed to write data to a block of memory from address 2000 hex to 7FFF, and then read the same addresses back. Trace #1 is set to disable CRC sampling and wait until the trigger pattern defined by Qualifier #1 (QCOM1) is detected on the input pins. When QCOM1 is detected, record one sample and advance to Sequence #2. Recording will be continuous at this level, until QCOM3 evaluates true, indicating the end of the write process. Sequence #3 halts all data recording and waits for QCOM2 to evaluate true. At this time, a single CRC sample is performed, and the trace advances to Sequence #4. At Sequence level 4, CRC sampling is continuous until QCOM3 is again detected, indicating the end of the read process. Detection of QCOM3 causes an advance to Sequence #5. Sequence #5 halts all data recording and CRC sampling, thus ending the record process.

Record Memory Wrapping (NON-SCPI)



The RECord:TRACe:WRAP command turns ON or OFF the record wrap-around feature. When WRAP is set to off, data recording stops when the available record memory is full. In this case the maximum samples is defined by the test_size. If WRAP is set to ON, then recording wraps around to the beginning of record memory when the end of the record memory is reached. Many thousands, or millions, of sample may have been made, however, the record memory only holds the most recent. When recording stops, the SR2500 automatically arranges the contents of the record memory so that the oldest data recorded is located at vector 1, and the most recent data located at the last record vector.

:WRAP <ON | OFF> Instructs the SR2500 to turn ON or turn OFF record wrap-around.

Parameters **ON** = Record wrap-around is enabled.

OFF (default) = Record wrap-around is disabled.

Examples RECORD:TRACE:WRAP ON
REC:TRAC:WRAP ON

:WRAP? Queries the state of the SR2500 record wrap-around.

Response 0 | 1

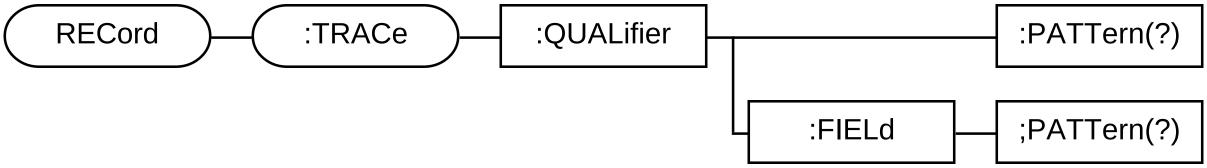
Parameters **1** = Record wrap-around is enabled.

0 = Record wrap-around is disabled.

Examples RECORD:TRACE:WRAP?
1

REC:TRAC:WRAP?
0

Qualifier Trigger Patterns (NON-SCPI)



The RECord:TRACe:QUALifier:PATTern command defines a trigger pattern for the specified field and assigns the field trigger pattern to the specified qualifier. A qualifier may be assigned multiple field trigger patterns. Each added trigger pattern is logically ANDed with the other trigger patterns assigned to the same qualifier.

:QUALifier <qual_num | ALL>

Defines the qualifier to assign the field trigger pattern to.

Parameters

qual_num = (1 - 8)
ALL = Specifies all 8 qualifiers.

:FIELd <name>

The optional FIELd parameter allows the trigger pattern to be loaded to (or queried from) a destination field other than the default record type field. If the FIELd parameter option is used, then the FIELd and PATTern(?) parameters must be separated by a semicolon instead of colons, as shown in the examples below. Valid field types for this command are Expect (EXP), Dontcare (DON), Expect/Dontcare (ED), Algorithmic Expect (ALGE), Hardware Expect (HEXP) and Hardware Dontcare (HDON). For proper operation, Expect and Dontcare field pairs should be used, either as separate fields (EXP/ALGE/HEXP and DON/HDON), or combined (ED).

Note

The FIELd parameter changes the destination field only for the command in which it occurs, but **it does not** change the default field.

Parameter Definition

name = Any alphanumeric string and '_' (max 8 characters).

:PATTern <data_value>

The data_value parameter is the actual trigger pattern, mask (Dontcare) pattern, or trigger and mask pattern, that will be assigned to the qualifier. If no radix prefix (#h or #b) is used with the data values, then the data values must be entered in the radix format defined for the destination field. The radix format for the destination field is defined by the FIELd:NAME:RADix command. If the radix for the destination field is set to HEX, then data can be specified in hexadecimal format (the '#h' prefix is optional) or in binary format if the '#b' prefix is specified. Valid hexadecimal data values are '0' through 'F'. For hexadecimal radix fields, the 'X' character represents a don't care condition for that nibble (1 nibble = 4 bits). If the radix for the field is set to BIN, then data can be specified in binary format (the '#b' prefix is optional) or in hexadecimal format if the '#h' prefix is specified. For binary radix fields, the 'X' character represents a don't care condition for the corresponding bit position. Leading '0' data characters may be omitted as shown in the examples below.

Parameter Definition **data_value** = [#h]{(0-F) | X} | #b{0 | 1 | X}

Examples RECORD:TRACE:QUALIFIER 1:FIELD ADDR;PATTERN #h2000

REC:TRAC:QUAL 1:FIEL DATA;PATT #hXX
REC:TRAC:QUAL 1:FIEL R_W;PATT #b01
REC:TRAC:QUAL 2:FIEL ADDR;PATT #h2000;FIEL DATA;PATT #hXX;FIEL R_W;PATT #b10
REC:TRAC:QUAL 3:FIEL ADDR;PATT #h7FFF;FIEL DATA;PATT #hXX;FIEL R_W;PATT #bXX

:PATTern?

Queries the specified fields trigger pattern for the specified qualifier. The radix of data_value is determined by the FIELd:NAME:RADix command. If the radix for the field is set to HEX, then data will be returned in hexadecimal format with the '#h' prefix. Valid hexadecimal data values are '0' through 'F'. The hexadecimal 'X' character is valid only with Expected/DontCare type fields (ED) and represents a don't care condition for that nibble (1 nibble=4 bits). The hexidecimal '?' character will be displayed when a nibble contains a combination of enabled and don't care expect pins.

If the radix for the field is set to BIN, then data will be returned in binary format with the '#b' prefix. Valid binary data values are '0', '1'. The binary 'X' character is valid only with Expected/DontCare type fields (ED) and represents a don't care condition for the corresponding bit position. Leading '0' data characters will be returned.

Response data_value

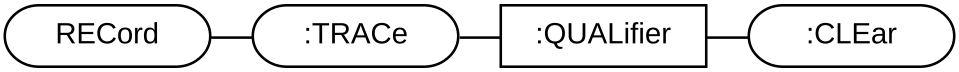
Parameter Definition **data_value** = #h{(0-F) | X} | #b{0 | 1 | X}

Examples RECORD:TRACE:QUALIFIER 1:FIELD DATA;PATTERN?
#hXX

REC:TRAC:QUAL ALL:FIEL ADDR;PATT?
#h2000,#h2000,#h7FFF,#hXXXX,#hXXXX,#hXXXX,#hXXXX,#hXXXX

REC:TRAC:QUAL ALL:FIEL R_W;PATT?
#b01,#b10,#bXX,#bXX,#bXX,#bXX,#bXX,#bXX

Clearing Qualifier Trigger Patterns (NON-SCPI)



The RECord:TRACe:QUALifier:CLEar command clears all of the field trigger patterns assigned to the specified qualifier.

:QUALifier <qual_num | ALL> Defines the qualifier to clear.

Parameters **qual_num** = (1 - 8)

ALL = Specifies all 8 qualifiers.

:CLEar Clears the specified qualifier of all field trigger patterns assigned to it.

Parameter Definition none

Examples

```
RECORD:TRACE:QUALIFIER 1:CLEAR
REC:TRAC:QUAL ALL:CLE
```

Qualifier Trigger Combinations (NON-SCPI)

RECord
:TRACe
:QCOMbination(?)

The RECord:TRACe:QCOMbination command defines the logical OR combination of Qualifiers patterns for record triggering and program control.

:QCOMbination<qcom_num> <qual_num[-qual_num]> [{,<qual_num[-qual_num]>}]

Specifies the Qcombination number and the Qualifier(s) associated with it. The qcom_num parameter is placed immediately after the QCOMbination command without a space between the two. The trigger patterns defined in each of the qualifiers are logically ANDed together, while the qualifiers within the qcombination are logically ORed. Together, they define a traditional Logic Analyzer boolean trigger equation.

A maximum of 8 QCOMbinations may be defined, each consisting of 1 to 8 qualifiers. Qualifiers may be specified in a list format (qualifiers separated by a comma), or as a range of qualifiers (qualifier ranges separated by a '-'). Or, lists and ranges may be mixed as shown in the examples.

Parameters **qcom_num** = (1 - 8)

qual_num = (1 - 8)

Examples

```
RECORD:TRACE:QCOMBINATION1 1
REC:TRAC:QCOM2 1,2,3,7
REC:TRAC:QCOM3 2-5
REC:TRAC:QCOM4 1-3,5,7-8
```

:QCOMbination<qcom_num>?

Queries the Qualifier(s) defined for the specified Qcombination.

Parameters **qcom_num** = (1 - 8)

Response qual_num[{,qual_num}]

Examples

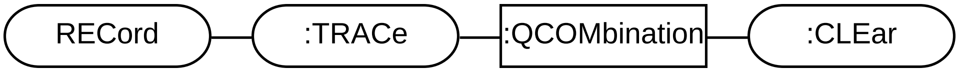
RECORD:TRACE:QCOMBINATION1?
1

REC:TRAC:QCOM2?
1,2,3,7

REC:TRAC:QCOM3?
2,3,4,5

REC:TRAC:QCOM4?
1,2,3,5,7,8

Clearing Qualifier Trigger Combinations (NON-SCPI)



The RECord:TRACe:QCOMbination:CLEar command clears the specified Qualifier Combination of all of the field trigger patterns assigned to it.

:QCOMbination<qcom_num> Defines the Qualifier Combination to clear. The qcom_num parameter is placed immediately after the QCOMbination command without a space between the two.

Parameters **qcom_num** = (1 - 8)

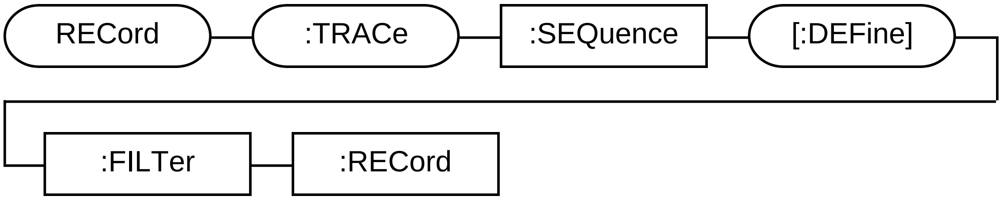
:CLEar Clears the specified Qualifier Combination.

Parameter Definition none

Examples

```
RECORD:TRACE:QCOMBINATION1:CLEAR
REC:TRAC:QCOM2:CLE
```

Record Filter and Control (NON-SCPI)



The RECord:TRACe:SEQuence:DEFine:FILTer:RECord command defines what information to save to record memory and when to save it. For additional information about Trace Sequences, refer to the beginning of the "Advanced Record Triggering" section, pg. 3-186.

:SEQuence <seq_num | ALL>

Defines sequence level that the command parameters will apply to.

Parameters

seq_num = (1 - 16)

ALL = Specifies all 16 sequence levels.

[:DEFine]

Provide the path into the sequence definition subsystem. DEFine is the default path, so the DEFine command may be omitted.

Parameter Definition

none

:FILTer <DATA | ERRor>

Defines what information to save to the record memory, when the RECord conditions are met.

Parameter Definition

DATA = Record the input data from the UUT.
ERRor = Record the results of the real-time compare.
0 = no error
1 = error

:RECord <NEVer | ALWays | COMpare | NCOMpare | QCOM<qcom_num>>

Specifies when the selected information is saved to the record memory.

Parameter Definition

NEVer = Never save information to the record memory.

ALWays = Always save the specified data to the record memory

COMpare = Save the specified data to the record memory whenever the real-time compare is true. The real-time compare state is the dynamic result of comparing the input data from the UUT to the current vectors expected data pattern.

NCOMpare = Save the specified data to the record memory whenever the real-time compare is false. The real-time compare state is the dynamic result of comparing the input data from the UUT to the current vectors expected data pattern.

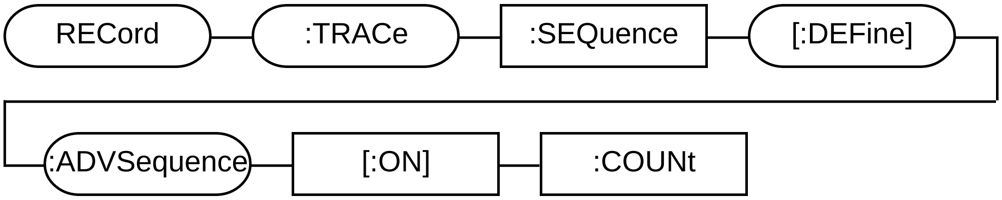
QCOM = Save the specified data to the record memory whenever one or more of the qualifier trigger patterns, as specified by the qualifier combination, evaluates true, i.e., matches the input data from the UUT.

qcom_num = (1 - 8)

Examples

```
RECORD:TRACE:SEQUENCE 1:DEFINE:FILTER DATA:RECORD QCOM1
REC:TRAC:SEQ 2:DEF:FILT DATA:REC ALW
REC:TRAC:SEQ 3:FILT DATA:REC NEV
REC:TRAC:SEQ 4:FILT DATA:REC NEV
```

Advancing Trace Sequences (NON-SCPI)



The RECord:TRACe:SEQuence:DEFine:ADVSequence:ON:COUNt command defines the conditions that must be met in order to advance to the next sequence level. Or, another way to look at it, the command defines how long you will stay at the current sequence level recording the specified data. If both the Advance Sequence and Jump conditions are specified at the same sequence level, and if both conditions are met simultaneously, then the JUMP takes priority and the next trace sequence level executed will be the one specified by the JUMP condition. For additional information about Trace Sequences, refer to "Advanced Record Triggering," pg. 3-186.

:SEQuence <seq_num | ALL> Defines sequence level that the command parameters will apply to.

Parameters **seq_num** = (1 - 16)

ALL = Specifies all 16 sequence levels.

[:DEFine] Provide the path into the sequence definition subsystem. DEFine is the default path, so the DEFine command may be omitted.

Parameter Definition none

:ADVSequence Provide the path into the advance sequence definition subsystem.

Parameter Definition none

[:ON] <NEVer | CLOCk | COMpare | NCOMpare | QCOM<qcom_num>>

Specifies when the selected information is saved to the record memory. The ON command is optional and may be omitted for convenience.

Parameter Definition **NEVer** = Never advance to the next sequence level.

CLOCk = Advance to the next sequence level after the number of clock cycles defined by advs_count.

COMpare = Advance to the next sequence level after the real-time compare condition evaluates true for the number of cycles specified by the advs_count parameter.

NCOMpare = Advance to the next sequence level after the real-time compare condition evaluates false for the number of cycles specified by the advs_count parameter.

QCOM = Advance to the next sequence level after the qualifier combination, as specified by qcom_num, evaluates true for the number of cycles specified by the advs_count parameter.

qcom_num = (1 - 8)

:COUNt <advs_count>

Defines the number of times the ADVS:ON condition must evaluate true before advancing to the next sequence level.

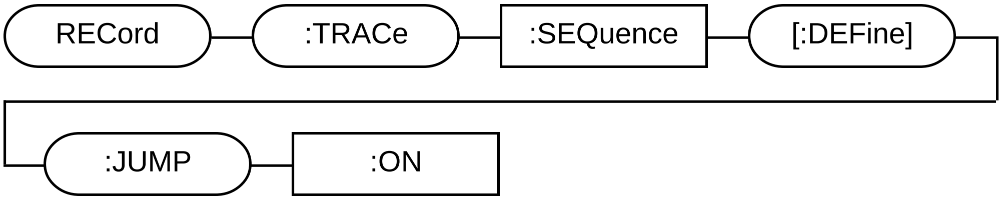
Parameter Definition

advs_count = (1 - 65535)

Examples

```
RECORD:TRACE:SEQUENCE 1:DEFINE:ADVSEQUENCE:ON
QCOM1:COUNT 1
REC:TRAC:SEQ 2:DEF:ADVS QCOM3:COUN 1
REC:TRAC:SEQ 3:ADVS QCOM2:COUN 1
```

Jumping to Trace Sequences (NON-SCPI)



The RECord:TRACe:SEQuence:DEFine:JUMP:ON command defines the conditions that must be met in order to jump to the specified sequence level. If both the Advance Sequence (ADVS) and Jump conditions are specified for the same sequence level, and if both conditions are met simultaneously, then the JUMP takes priority and the next trace sequence level executed will be the one specified by the JUMP condition. For additional information about Trace Sequences, refer to "Advanced Record Triggering," pg. 3-186.

:SEQuence <seq_num | ALL> Defines sequence level that the command parameters will apply to.

Parameters **seq_num** = (1 - 16)

ALL = Specifies all 16 sequence levels.

[:DEFine] Provide the path into the sequence definition subsystem. DEFine is the default path, so the DEFine command may be omitted.

Parameter Definition none

:JUMP <seq_num> Specifies the sequence level to jump to when the jump condition is met.

Parameter Definition **seq_num** = (1 - 16)

:ON <NEVer | COMpare | NCOMpare | QCOM<qcom_num>>

Specifies when the selected information is saved to the record memory.

Parameter Definition **NEVer** = Never advance to the next sequence level.

COMpare = Advance to the next sequence level after the real-time compare condition evaluates true for the number of cycles specified by the advs_count parameter.

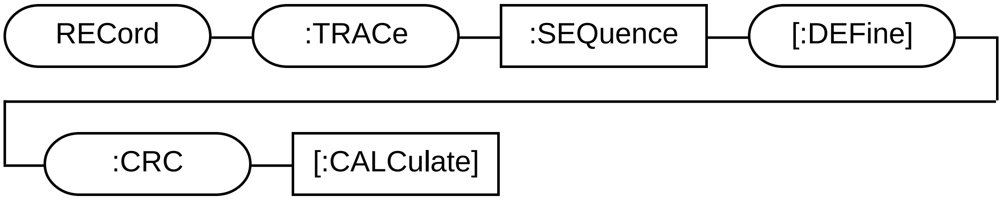
NCOMpare = Advance to the next sequence level after the real-time compare condition evaluates false for the number of cycles specified by the advs_count parameter.

QCOM = Advance to the next sequence level after the qualifier combination, as specified by qcom_num, evaluates true for the number of cycles specified by the advs_count parameter.

Examples **qcom_num** = (1 - 8)

RECORD:TRACE:SEQUENCE 1:DEFINE:JUMP 1:ON QCOM1
REC:TRAC:SEQ 2:DEF:JUMP 1:ON QCOM1
REC:TRAC:SEQ 3:JUMP 1:ON QCOM1

CRC Calculation Control (NON-SCPI)



The RECord:TRACe:SEQuence:DEFine:CRC:CALCulate command defines when the information being returned by the UUT will be used in CRC calculations. For additional information about Trace Sequences, refer to "Advanced Record Triggering" section, pg. 3-186.

:SEQuence <seq_num | ALL> Defines sequence level that the command parameters will apply to.

Parameters **seq_num** = (1 - 16)

ALL = Specifies all 16 sequence levels.

[:DEFine] Provides the path into the sequence definition subsystem. DEFine is the default path, so the DEFine command may be omitted.

Parameter Definition none

:CRC Provide the path into the CRC calculation subsystem.

Parameter Definition none

[:CALCulate] <NEVer | ALWays | COMpare | NCOMpare | QCOM<qcom_num>>

Specifies under what conditions the information being returned by the UUT will be used in a CRC calculation. The CALCulate command is optional and may be omitted for convenience.

Parameter Definition **NEVer** = Never calculate the CRC.

ALWays = Always calculate the CRC.

COMpare = Calculate the CRC whenever the real-time compare is true. The real-time compare state is the dynamic result of comparing the input data from the UUT to the current vectors expected data pattern.

NCOMpare = Calculate the CRC whenever the real-time compare is false. The real-time compare state is the dynamic result of comparing the input data from the UUT to the current vectors expected data pattern.

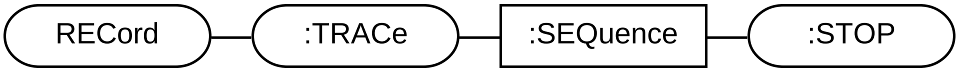
QCOM = Calculate the CRC whenever one or more of the qualifier trigger patterns, as specified by the qualifier combination, evaluates true, i.e., matches the input data from the UUT.

qcom_num = (1 - 8)

Examples

```
RECORD:TRACE:SEQUENCE 1:DEFINE:CRC:CALCULATE QCOM1
REC:TRAC:SEQ 2:DEF:CRC:CALL ALW
REC:TRAC:SEQ 3:CRC:CALC NEV
REC:TRAC:SEQ 4:CRC NEV
```

Stopping Tests from Trace Sequences (NON-SCPI)



The RECord:TRACe:QUALifier:STOP command instructs the SR2500 to stop the currently running test when the trace sequence level specified is reached. The stop process is controlled via software, so stopping a test is not immediate. The amount of vector over-run is indeterminate and depends on the speed of the test that is running. Once the STOP parameter is defined for a sequence level, the only way to clear the stop condition is to clear the sequence level and redefine it.

:SEQuence <seq_num | ALL> Defines sequence level that the command parameters will apply to.

Parameters seq_num = (1 - 16)

ALL = Specifies all 16 sequence levels.

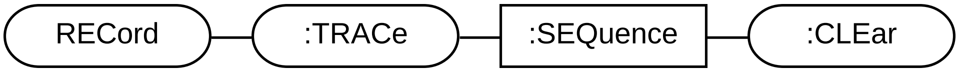
:STOP Sets the stop condition for the sequence level specified.

Parameter Definition none

Examples

```
RECORD:TRACE:SEQUENCE 1:STOP
REC:TRAC:SEQ 2:STOP
```

Clearing Trace Sequences (NON-SCPI)



The RECord:TRACe:SEQuence:CLEar command clears all of the record control parameters for the specified sequence level.

:SEQuence <seq_num | ALL> Defines sequence level that the command parameters will apply to.

Parameters **seq_num** = (1 - 16)

ALL = Specifies all 16 sequence levels.

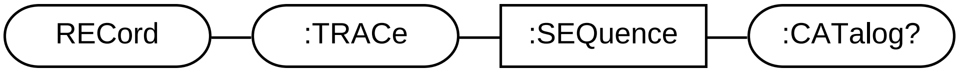
:CLEar Clears the specified sequence level of all record controls defined for it.

Parameter Definition none

Examples

```
RECORD:TRACE:SEQUENCE 1:CLEAR
REC:TRAC:SEQ ALL:CLE
```

Trace Sequences Catalog (NON-SCPI)



The RECord:TRACe:SEQuence:CATalog? query command return the record control parameters for specified sequence level.

:SEQuence <seq_num | ALL> Defines sequence level that the command parameters will apply to.

Parameters **seq_num** = (1 - 16)

ALL = Specifies all 16 sequence levels.

:CATalog? Returns the record control parameters for the specified sequence level. If ALL sequence levels were specified, the response for each level will be separated by a semi-colon ';' character.

Response {seq_num FIL <DAT | ERR> <NEV | ALW | COM | NCOM | QCOM<qcom_num>>,CRC <NEV | ALW | COM | NCOM | QCOM<qcom_num>>,ADVS <NEV | CLOC | COM | NCOM | QCOM<qcom_num>> advs_count,JUMP seq_num <NEV | COM | NCOM | QCOM<qcom_num>>[;]}

Parameter Definition **seq_num** = (1 - 16)

qcom_num = (1 - 8)

advs_count = (1 - 65535)

NEVer = Never record data, calculate CRC, advance to the next sequence sequence level or jump to a new sequence level.

ALWays = Always record data or calculate CRC.

CLOCk = Advance to the next sequence level after the number of clock cycles defined by advs_count.

COMpare = Record data, calculate CRC, advance to the next sequence level or jump to new sequence level whenever the real-time compare is true. The real-time compare state is the dynamic result of comparing the input data from the UUT to the current vectors expected data pattern.

NCOMpare = Record data, calculate CRC, advance to the next sequence level or jump to new sequence level whenever the real-time compare is false. The real-time compare state is the dynamic result of comparing the input data from the UUT to the current vectors expected data pattern.

QCOM = Record data, calculate CRC, advance to the next sequence level or jump to new sequence level whenever one or more of the qualifier trigger patterns, as specified by the qualifier combination, evaluates true, i.e., matches the input data from the UUT.

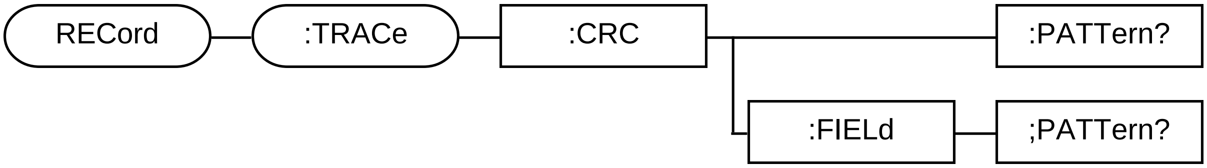
Examples

RECORD:TRACE:SEQUENCE 1:CATALOG?
1 FIL DAT QCOM1,CRC NEV,ADVS QCOM1 1,JUMP 1 NEV

REC:TRAC:SEQ 2:CAT?
2 FIL DAT ALW,CRC NEV,ADVS QCOM3 1,JUMP 1 NEV

REC:TRAC:SEQ ALL:CAT?
1 FIL DAT QCOM1,CRC NEV,ADVS QCOM1 1,JUMP 1 NEV;2 FIL DAT ALW,CRC NEV,ADVS QCOM3 1,JUMP 1 NEV;3 FIL DAT NEV,CRC QCOM2,ADVS QCOM2 1,JUMP 1 NEV;4 FIL DAT NEV,CRC ALW,ADVS QCOM3 1,JUMP 1 NEV;5 FIL DAT NEV,CRC NEV,ADVS NEV 1,JUMP 1 NEV;6 FIL DAT NEV,CRC NEV,ADVS NEV 1,JUMP 1 NEV;7 FIL DAT NEV,CRC NEV,ADVS NEV 1,JUMP 1 NEV;8 FIL DAT NEV,CRC NEV,ADVS NEV 1,JUMP 1 NEV;9 FIL DAT NEV,CRC NEV,ADVS NEV 1,JUMP 1 NEV;10 FIL DAT NEV,CRC NEV,ADVS NEV 1,JUMP 1 NEV;11 FIL DAT NEV,CRC NEV,ADVS NEV 1,JUMP 1 NEV;12 FIL DAT NEV,CRC NEV,ADVS NEV 1,JUMP 1 NEV;13 FIL DAT NEV,CRC NEV,ADVS NEV 1,JUMP 1 NEV;14 FIL DAT NEV,CRC NEV,ADVS NEV 1,JUMP 1 NEV;15 FIL DAT NEV,CRC NEV,ADVS NEV 1,JUMP 1 NEV;16 FIL DAT NEV,CRC NEV,ADVS NEV 1,JUMP 1 NEV

Querying CRC Checksums (NON-SCPI)



The RECord:TRACe:CRC:PATTern? query command returns the CRC checksum for all pins in the specified field.

:CRC

Provides the path into the CRC pattern query subsystem.

Parameters none

:FIELd <name>

The optional FIELd parameter allows the CRC checksums to be queried from a destination field other than the default field. If the FIELd parameter option is used, then the FIELd and PATTern? parameters must be separated by a semicolon, as shown in the examples. The only valid field types for this command are Record (REC) and Hardware Record (HREC).

Parameter Definition **name** = Any alphanumeric string and '_' (max 8 characters).

Note

The FIELd parameter changes the destination field only for the command in which it occurs, but **it does not** change the default field.

:PATTern?

Queries the CRC checksum for all pins in the specified field. The radix of crc_checksum is always set to HEX, regardless of how the field's radix is defined. Refer to the FIELd subsystem in section 3.1.3 for further information about field pin mapping.

Response <crc_num> C<card#>P<pin#> <crc_checksum>[{,<crc_num> C<card#>P<pin#> <crc_checksum>}]

Parameter Definition

crc_num = (1 - 32) - Ordinal position of the pin within the field.
card# = (1 - 18) - SR25XX module number.
pin# = (1 - 32) - SR25XX pin number.
crc_checksum = #h{(0-F)}[{,#h{(0-F)}}]

Examples

RECORD:TRACE:CRC:FIELD DATA;PATTERN?
8 C1P24 #HB980,7 C1P23 #H7A5E,6 C1P22 #HB980,5 C1P21 #H7A5E,4 C1P20 #HB980,3 C1P19 #H7A5E,2 C1P18 #HB980,1 C1P17 #H7A5E

RECORD:TRACE:CRC:FIELD ADDR;PATTERN?
16 C1P16 #HFD7A,15 C1P15 #H7188,14 C1P14 #H7746,13 C1P13 #H1E34,12 C1P12 #HC1D2,11 C1P11 #HE9A0,10 C1P10 #H291E,9 C1P9 #HFFCC,8 C1P8 #HB92A,7 C1P7 #H4CB8,6 C1P6 #H3DF6,5 C1P5 #H7C64,4 C1P4 #H4382,3 C1P3 #HFAD0,2 C1P2 #H15CE,1 C1P1 #HF3FC

THIS PAGE INTENTIONALLY LEFT BLANK

Miscellaneous Commands

This section includes the miscellaneous commands that did not fit anywhere else. These commands provide the ability to run diagnostics tests of the SR2500 system, and query the results of the test, query the status of the SR2500 system, specify the conditions in which the SR2500 will generate a Service Request Interrupt to the Slot-0 Controller, and define the SR2500 Bus Master time-out period. Also included in this section are the 488.2 mandatory commands. This section is divided into the following subsections:

- ***SR2500 Diagnostics .... pg. 3-208***
- ***Status Queries, Status Interrupts, and System Queries........ pg. 3-214***
- ***System IDN Query .... pg. 3-225***
- ***Bus Master Time-out .... pg. 3-227***
- ***Variable Voltage I/O on the SR2500 .... pg. 3-229***
- ***IEEE 488.2 Commands .... pg. 3-233***

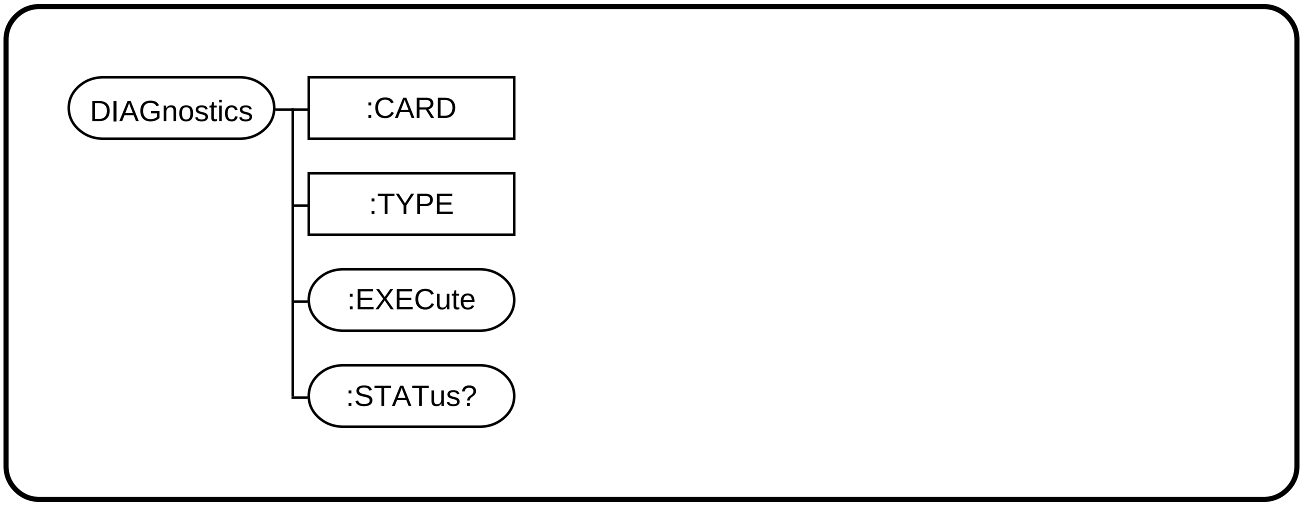
Diagnostics

The SR2500 provides the ability to execute an internal function test called Diagnostics. The diagnostics command allows you to specify which sub-system in the total SR2500 system to test, and what test to perform. After the specified diagnostics test has completed, the diagnostics command sub-system allows the results of the test to be queried, indicating the type of failure detected by the diagnostics test, assuming one existed.

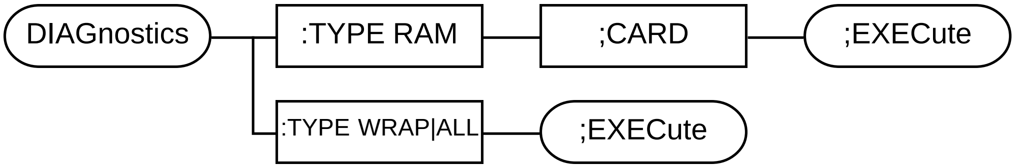
Tests Performed

The table below is a listing of the tests performed by the SR2500 firmware upon receiving the DIAG:EXEC command. The 2nd colomn indicates which tests are run during the RAM section of the diagnostics and the 3rd column indicates which tests are run during the WRAP section.

I/O Cards		
TEST	**RAM Test**	**WRAP Test**
Stimulus Gate Array Test	Yes	No
Response Gate Array Test	Yes	No
Output Memory Test	Yes	No
Tristate Memory Test	Yes	No
Expect Memory Test	Yes	No
Response Memory Test	Yes	No
Algorithmic Memory Test	Yes	No
Test Definition	Yes	No
Field Definition	No	Yes
Fill Command	No	Yes
Record Qualifiers	No	Yes
Record Trace Sequences	No	Yes
Output Stimulus Vectors	No	Yes
Record of Input Vectors	No	Yes
Timing / Control Card		
Command Macro Memory Test	Yes	No
Control Gate Array Test	Yes	No


DIAGnostics
:CARD
:TYPE
:EXECute
:STATus?

Diagnostic Test Execution (NON-SCPI)



The DIAGnostics command executes the SR2500 diagnostic self tests. All defined tests must be deleted prior to executing the diagnostic self tests. There are 2 types of diagnostic tests, RAM Test and Wraparound Test. The RAM Test is a non-intrusive test and ***does not*** generate patterns to the I/O pins.

The RAM Test performs the following test on the SR2510 Timing/Control Board:

o Control Gate Array Register Read/Write test
o Stimulus Command Memory test
o Stimulus Parameter Memory test
o EEPROM Checksum test

The RAM Test performs the following SR25XX I/O Module tests:

o Stimulus Gate Array Register Read/Write tests
o Response Gate Array Register Read/Write tests
o Output Memory test
o Tristate Memory test
o Expected Memory test
o Dontcare Memory test
o Record Memory test
o Stimulus Algorithmic Memory
o Response Algorithmic Memory test
o EEPROM Checksum test

The Wraparound Test executes a comprehensive diagnosis of the SR25XX I/O drivers and receivers. The Wraparound Cables must be installed from the output connectors to the input connectors. By specifying a Wraparound test, a special test is defined internally with a "walking 1" pattern generated on the stimulus output pins and the same pattern expected/ compared against the response input pins. The SR2500 is placed in the RUNNING mode and therefore all I/O Modules will be tested regardless of the I/O card specified.

:TYPE <RAM | WRAP | ALL>

The TYPE parameter specifies the diagnostic test type that will be executed. The diagnostic test type can also be specified by the literal string "ALL", where both the RAM test and the Wraparound test will be executed. If the TYPE parameter is omitted, then "ALL" tests will be executed.

Parameter Definition

RAM = RAM Test will be executed.

WRAP = Wraparound Test will be executed.

ALL = Both the RAM Test and the Wraparound Test will be executed.

;CARD <card_num | ALL>

The optional CARD parameter specifies which SR2500 card number will be tested, where "0" is the SR2510 Timing/Control Board and "1" through "18" is the card number of the SR25XX I/O Modules. The card number can also be specified by the literal string "ALL", where all SR2500 system modules will be tested. If the CARD parameter string is not specified, then all cards will be tested.

Note

If the WRAP test parameter is selected, then all SR2500 I/O modules will be tested, regardless of the CARD number specified.

Parameter Definition

card_num = (0 - 18)

ALL = All SR2500 modules will be tested.

;EXECute

The EXECute command executes the specified diagnostic test.

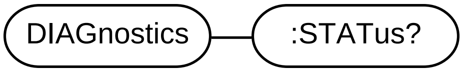
Parameter Definition

none

Examples

```
DIAGNOSTICS:TYPE RAM;CARD 2;EXECUTE
DIAG:TYPE WRAP;EXEC
DIAG:EXEC
```

Diagnostic Test Status Query (NON-SCPI)



The DIAGnostics:STATus? query command returns the results of the last diagnostic test executed.

:STATus?

Response

card_num,fail_type,fail_string,fail_patt[{;card_num,fail_type,fail_string,fail_patt}]

Parameter Definition:

card_num = (0 - 18), the card number where the test failure occurred.

fail_type= (R/W ERROR | UNINITIALIZED | WRAP-AROUND ERROR), the failure type description string.

fail_string= (CONTROL GATE ARRAY | EEPROM | STIM GATE ARRAY 0 | STIM GATE ARRAY 1 | STIM GATE ARRAY 2 | STIM GATE ARRAY 3 | REC GATE ARRAY 0 | REC GATE ARRAY 1 | REC GATE ARRAY 2 | REC GATE ARRAY 3 | NO TRIGGER WORD FOUND | PATTERN), the failure location description string.

fail_patt = (#hXXXXXXXX[{,#hXXXXXXXX}]), where #hXXXXXXXX represents the failing pattern. For R/W ERRORs and WRAP-AROUND ERRORs, two patterns will be returned. The first pattern returned is the expected data and the second pattern is the actual data. For example, the response shown below returns a R/W ERROR failure where the expected pattern was #h000055AA and the actual pattern read was #h00000000.

Examples

DIAGNOSTICS:STATUS?

0,R/W ERROR,CONTROL GATE ARRAY,#h0000055aa,#h00000000;1,R/W ERROR,STIM GATE ARRAY 2,#h000055aa,#h00000000

DIAG:STAT?

1,WRAP-AROUND ERROR,PATTERN,#h00000020,#h000000f0

THIS PAGE INTENTIONALLY LEFT BLANK

Status Queries, Status Interrupts and System Queries

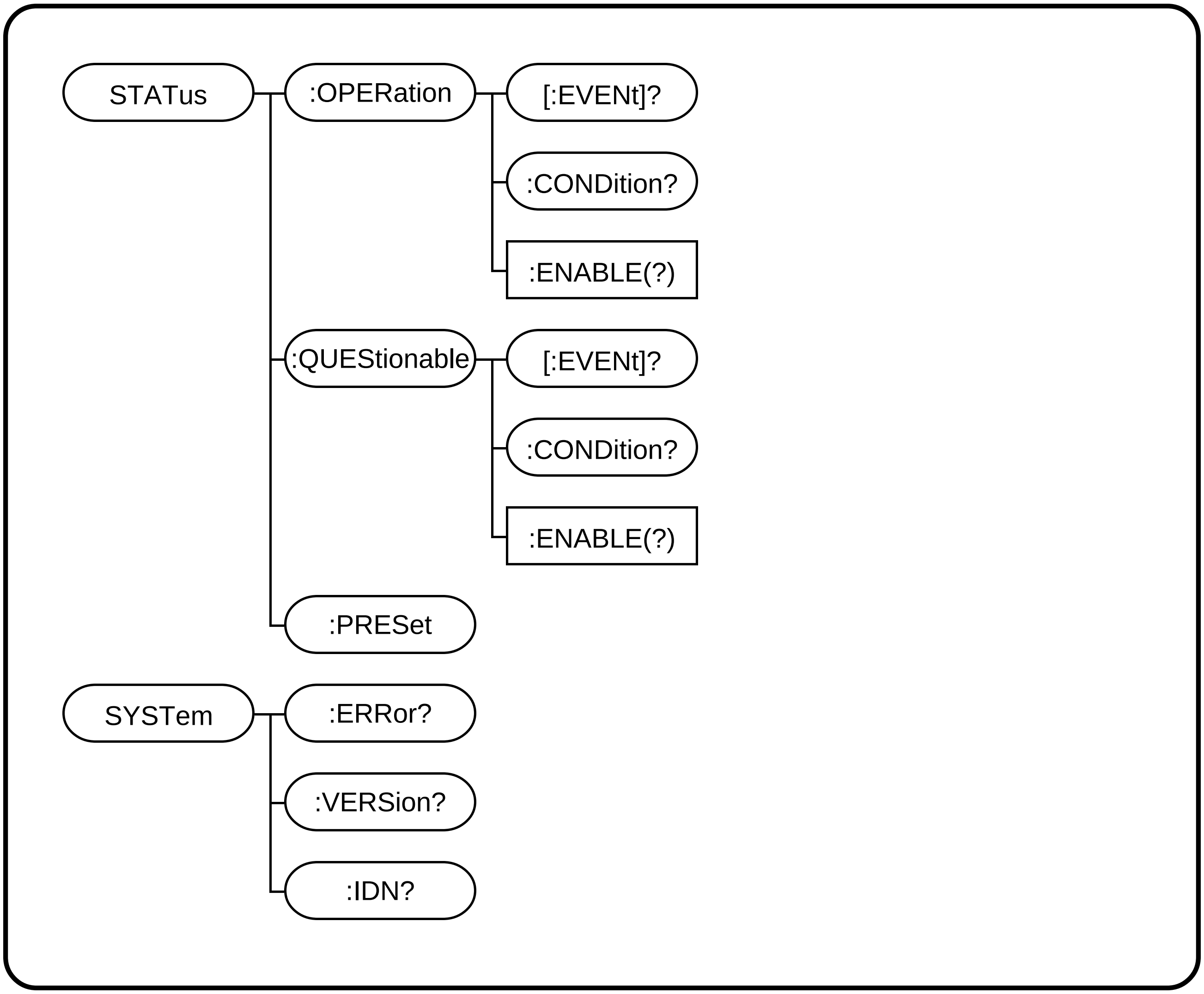
The SR2500 includes the Status Reporting mechanisms described in chapter 11 of IEEE 488.2 and the SCPI-defined Operation Status Register and Questionable Data/Signal Status Register. The SR2500 does not define any functions or conditions for the Questionable Status Register, however, the register reporting commands are included as part of the "Minimum Status Reporting Structure" required by SCPI.

The Operation Status register allows querying the current operational status of the SR2500, as well as defining operational events which will cause an interrupt to the slot 0 controller to be generated. The operational condition of the SR2500 is a dynamic status, meaning it will constantly be updated with the current status. Operation events are latched. When the enabled event occurs (Operation Status Enable Register), the condition will be latched in the Operation Status Event Register and an interrupt generated. The event will remain latched in the operation status event register until the state of the register is queried. Querying the register clears the event and resets the interrupt.

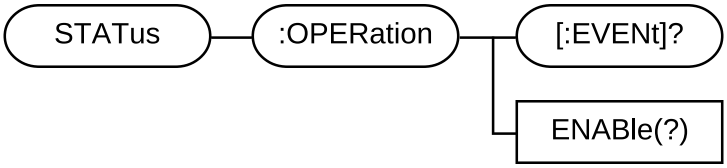
The System Queries defined in this section allow reading the following system parameters; Command Errors, the SCPI syntax version supported and system identification and configuration. The SYSTEM:IDN? command is similar to the IEEE 488.2 command "*IDN?" and returns system configuration information.

The SR2500 has extensive command error checking built into the command parser. When a command error occurs, the ERROR LED on the front panel of the SR2510 will be illuminated, and remain illuminated until the SYSTEM:ERROR? query is executed. Sending this command will extinguish the LED and return the command error that generated the condition.

The SYSTEM:VERSION? number returns the SCPI version supported. It is important to note that the SR2500 follows the SCPI syntax and rules, but most of the commands are not SCPI commands. This is due to the limited number of commands defined by the SCPI language to support digital requirements. Where possible, the SR2500 has used the defined SCPI commands, and each of these SCPI command is indicated by referring to the SCPI paragraph number in which it is defined.


STATus
:OPERation
[:EVENt]?
:CONDition?
:ENABLE(?)
:QUEStionable
[:EVENt]?
:CONDition?
:ENABLE(?)
:PRESet
SYSTem
:ERRor?
:VERSion?
:IDN?

Operation Interrupt Definition (SCPI 20.1 & 20.3)



The Operation Status Registers conform to the IEEE 488.2 specification and are comprised of the Event Register, the Enable Register, and the Condition Register. The Operation Event Register reports the latched operating status conditions for the SR2500. These conditions can be used to generate an interrupt to the Slot 0 by setting the Operation Event Summary Bit (OES Bit 7) in the IEEE 488.2 SRE Status Byte. The Operation Event Summary Bit is set when an enabled Status Operation Event Register bit is set true. The Operation Status Register supports the 'Waiting for Arm', 'Waiting for Trigger', and 'Program Running' status bits.

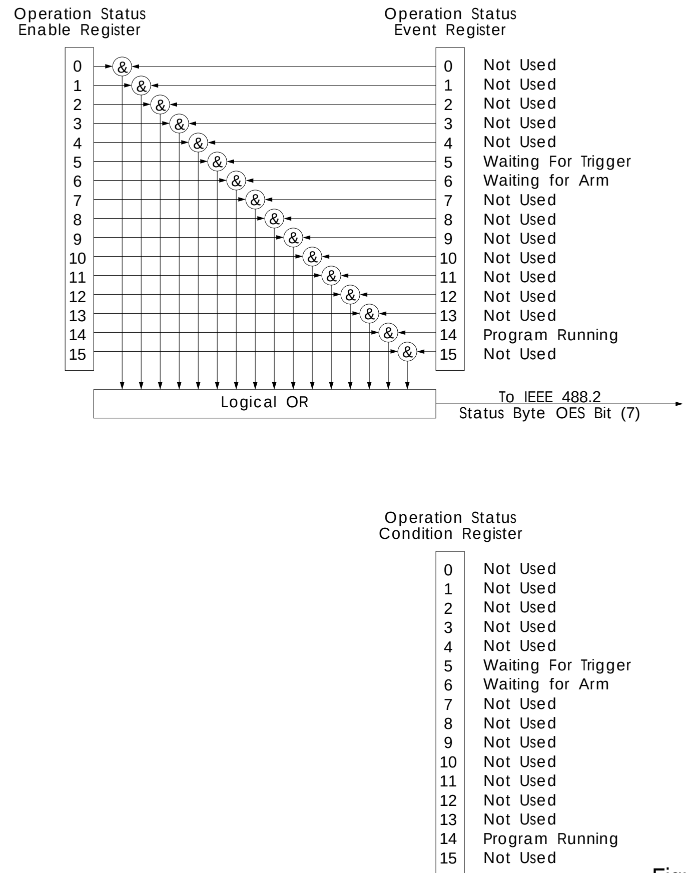


Figure 3-2.
Operation Status, Status Enable,
Status Event and Condition Registers.

The bit definition for the Event Register is identical to the Condition Register.

:[EVENt?]

The EVENt? command returns the contents of the Operation Event Register. The 16 bit event register is returned in decimal format. The contents of the Event Register is latched and may not represent the current state of the SR2500. EVENt is the default command within the OPERation branch and may be omitted for brevity. Reading the Event Register clears the contents of the register.

Response event_reg

Response **event_reg** = (0 - 65535)

Examples

STATUS:OPERATION:EVENT?
16480

STAT:OPER?
16480

:ENABle <enab_reg>

The ENABle command sets the contents of the 16 bit Operation Enable Register. The Enable Register is a mask register used to select which event(s) or bit(s), if any, will be used to set the event status bits in the Status Register. The contents of the Enable Register can be specified in decimal format; or in hexadecimal or binary format by using the '#h' and '#b' prefixes, respectively

Parameter Definition **enab_reg** = ((0 -65535) | (#h0 - #hFFFF) | (#b0 - #b1111111111111111))

Examples

STATUS:OPERATION:ENABLE 0
STAT:OPER:ENAB 0

:ENABle?

The ENABle? command returns the contents of the enable mask for the Operation Event Register. The 16 bit enable register is returned in decimal format.

Response enab_reg

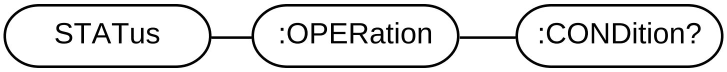
Parameter Definition **enab_reg** = (0 - 65535)

Examples

STATUS:OPERATION:ENABLE?
64

STAT:OPER:ENAB?
64

Operation Condition Query (SCPI 20.2)



The Operation Condition Register contains the current operating status condition for the SR2500. The Operation Condition Register supports the 'Waiting for Arm', 'Waiting for Trigger', and 'Program Running' status bits. The bit definition for the Operation Condition Register is identical to the Operation Event Register.

:CONDition?

The CONDition? command returns the contents of the Operation Condition Register. The 16 bit condition register is returned in decimal format. Reading the Condition Register ***does not*** clear the contents of the register.

Response cond_reg

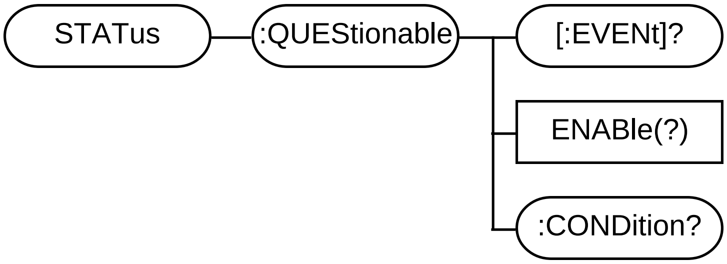
Parameter Definition **cond_reg** = (0 - 65535)

Examples

STATUS:OPERATION:CONDITION?
32

STAT:OPER:COND?
32

Questionable Status Registers (SCPI 20.1 - 20.3)



The Questionable Status Registers are not used by the SR2500; however, since they are required by SCPI, they have been included. The Questionable Status Registers do conform to the IEEE 488.2 specification and are comprised of the Event Register, the Enable Register, and the Condition Register. The Questionable Register commands are parsed by the SR2500, but no bits are used.

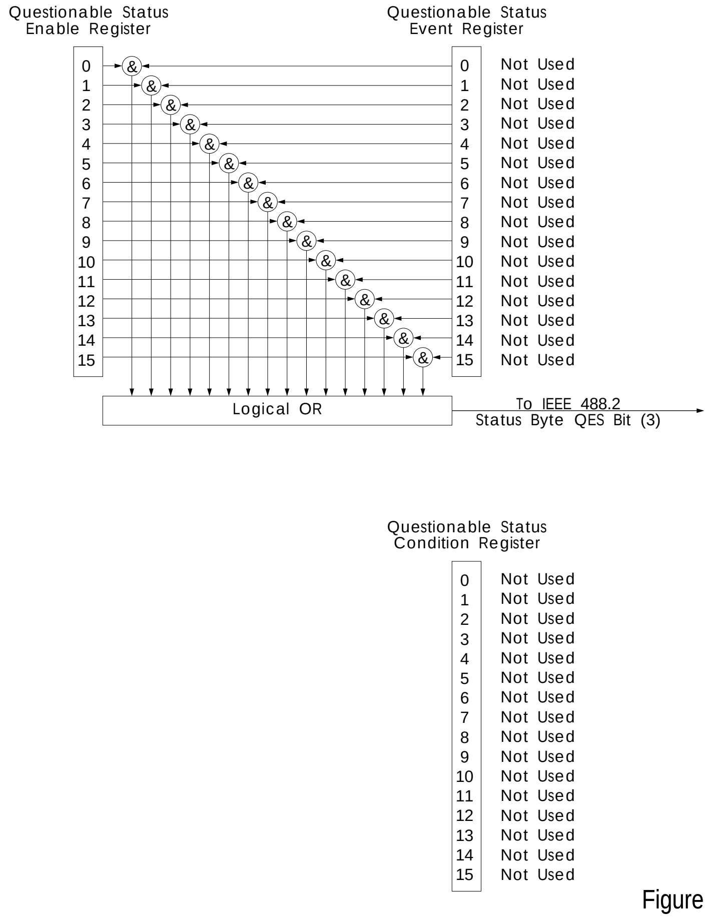


Figure 3-3.
Questionable Status, Status Enable,
Status Event and Condition Registers.

:[EVENt?]

The EVENt? command returns the contents of the Questionable Event Register. The 16 bit event register is returned in decimal format. The contents of the Event Register is always set to '0'. EVENt is the default command within the QUEStionable branch and may be omitted for brevity. Reading the Event Register clears the contents of the register.

Response event_reg

Parameter Definition **event_reg** = 0

Examples

```
STATUS:QUESTIONABLE:EVENT?
0

STAT:QUES?
0
```

:ENABle <enab_reg>

The ENABle command sets the contents of the 16 bit Questionable Enable Register. The contents of the Enable Register can be specified in decimal format; or in hexadecimal or binary format by using the '#h' and '#b' prefixes, respectively

Parameter Definition **enab_reg** = ((0 - 64) | (#h0 - #hFFFF) | (#b0 - #b1111111111111111))

Examples

```
STATUS:QUESTIONABLE:ENABLE 0
STAT:QUES:ENAB 0
```

:ENABle?

The ENABle? command returns the contents of the enable mask for the Questionable Event Register. The 16 bit enable register is returned in decimal format.

Response enab_reg

Parameter Definition **enab_reg** = (0 - 64)

Examples

```
STATUS:QUESTIONABLE:ENABLE?
0

STAT:QUES:ENAB?
0
```

:CONDition?

The CONDition? command returns the contents of the Questionable Condition Register. The 16 bit condition register is returned in decimal format. Reading the Condition Register ***does not*** clear the contents of the register.

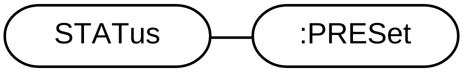
Response cond_reg

Parameter Definition **cond_reg** = 0

Examples

```
STATUS:QUESTIONABLE:CONDITION?
0

STAT:QUES:COND?
0
```

Status Preset (SCPI 20.7)



The STATus:PRESet command sets the Operation Enable Register and the Questionable Enable Register to a preset value of '0'. The Status Preset command configures the SCPI and device-dependent status data structures so that device-dependent events are reported at a higher level through the mandatory part of the status-reporting mechanism.

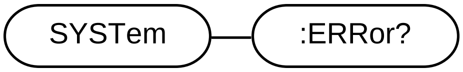
:PRESet

The PRESet command sets the Operation Enable Register and the Questionable Enable Register to a preset value of '0'.

Parameter Definition none

Examples STATUS:PRESET

System Error Query (SCPI 21.7)



SYSTem :ERRor?

The SYSTem:ERRor query command returns the error status of the latched command error. When a command error is generated, the error status is latched into the error register and the error status LED indicator on the front panel of the SR2510 will illuminate. The error status is returned as an error code number followed by a descriptive string. Refer to Appendix F for a list of error codes. Error codes are divided into sections. Section -100 represents command errors. Section -200 represents execution errors. Device-dependent errors are represented by -300 error codes and query errors use -400 numbers. Querying the error status clears the contents of the error register and will turn off the error status LED indicator on the SR2510.

:ERRor?

Response err_code,err_string

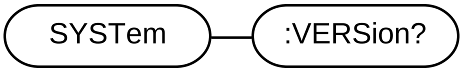
Parameter Definition **err_code** = -(101 - 499)

err_string = See Appendix F for a listing of error codes and error strings.

Examples SYSTEM:ERROR?
0,"No Error"

SYST:ERR?
-103,"Invalid Separator;Semi-colon or colon expected"

SCPI Version Query (SCPI 21.18)



The SYSTem:VERsion? query command returns the Standard Commands for Programmable Instruments (SCPI) revision number supported by the SR2500.

:VERSion?

Response year.version

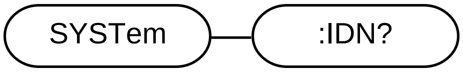
Parameter Definition **year** = YYYY, where YYYY represent the calendar year of the supported version.

version = V, where V is the version number of the supported version.

Examples SYSTEM:VERSION?
1993.0

SYST:VERS?
1993.0

System IDN Query (NON-SCPI)



The SYSTem:IDN? query command returns the SR2500 system configuration. The system configuration incudes information about the SR2500 including: software revision number, RAM options, logical addresses and I/O options installed for the SR2510 module and the SR2520 modules. Information on the RG2500 Rail Generator (if present) is also included.

:IDN?

Response

sw_rev;VXI-SR2520-25MHz(num_io,vec_depth)-cc_log_addr-
SNcc_ser_num;IOCARD1,H-io_type;[IOCARD2,H-
io_type;{IOCARD3,H-io_type,L-io_type;][VXI-SR2520[GP}-
25MHz(num_io,vec_depth)exp_log_addr;[IODCARxH-io_type,L-
io_type[IOCARDx,H-io_type;[IODARDx,H-io_type,L-
io;]]][PROBECARD-n][;VXI-RG2500-rg_num-rg_log_addr]

Parameter Definition

sw-rev = XX.XX where XX.XX represents the software revision number of the SR2510.

num_io = (0 - 3), the number of I/O cards installed in this module. The SR2510 must have at least 1 I/O card.

vec_depth = (64K | 256K), the vector depth of the module, all modules in a single system must be of the same depth.

cc_log_addr = (1 - 255), the SR2510's VXI logical address.

cc_ser_num = (YYYYYYYYYY-YYY), where YYYYYYYYYY-YYY represents the SR2510's serial number.

IOCARDx = (1 - 18), where x is the number of the I/O card. I/O cards are numbered in sequence, starting with IOCARD1, which must be installed in the SR2510. Except for the first I/O card, which must be installed in the 1st space in the SR2510, spaces may be skipped in the system when installing I/O cards.

H-io type = (ECL | TTL1 | TTL2 | TTL3 | TTL4 | CMOS3V | CMOS5V | VVT1 | NONE), this indicates that the high 16-bits (bits 32-17) for this I/O card are of this logic family.

L-io_type = (ECL | TTL1 | TTL2 | TTL3 | TTL4 | CMOS3V | CMOS5V | VVT1 | NONE), this indicates that the low 16-bits (bits 16-1) for this I/O card are of this logic family.

ECL is Differential ECL output.

TTL1 is Fast TTL with no Input term. and no Output Term.

TTL2 is Fast TTL with no Input Term. and 100 Ohm Output Term.

TTL3 is Fast TTL with 220 Ohm Input Term. and no Output Term.

TTL4 is Fast FFT with 220 Ohm Input Term. and 100 Ohm Output Term.

CMOS3V is 3.3 volt CMOS output.

CMOS5V is 5.0 volt CMOS output.

VVT1 is variable voltage output.

NONE is no adapter installed.

exp_log_addr = (2 - 255), the SR2520's VXI logical address.

GP = indicates that the Guided Probe option is installed on the SR2520 module, only 1 Guided Probe option is allowed in an SR2500 system.

PROBE CARD-n = (2 - 19), where n indicates the number of the Guided Probe Card, this will always be the last I/O card in the system. This parameter will only occur in SR2500 systems with the Guided Probe option installed.

rg_num = (1 - 9), the number of the RG2500 Rail Generator.

rg_log_addr = (2 - 255), the RG2500's VXI logical address.

Examples

For the SR2500 system with 1 control module (with 2 I/O cards) and 2 expansion modules (1 with 3 I/O cards, 1 with 3 I/O cards and Guided Probe):

SYSTEM:IDN?
1.07;VXI-SR2510-25MHz(2,256K)-7-SN0123456789-9876;IOCARD1,H-TTL1,L-TTL1;IOCARD2,H-CMOS5V,L-CMOS5V;VXI-SR2520-25MHz(3,356K)-8;IOCARD3,H-ECL,L-ECL;IOCARD4,H-TTL2,L-TTL2;IOCARD5,H-VVT;VXI-SR2520GP-25MHz(3,256K)-9;IOCARD6,H-ECL,L-TTL3;IOCARD7,H-CMOS3V,L-CMOS3V;IOCARD9,H-TTL4,L-NONE;PROBE CARD-9

For an SR2500 system with 1 control module (with 3 I/O cards) and 1 expansion module (with 2 I/O cards):

SYSTEM:IDN?
1.07;VXI-SR2510-25MHz(2,256K)-7-SN0123456789-9876;IOCARD1,H-TTL1,L-TTL1;IOCARD2,H-CMOS5V,L-CMOS5V;IOCARD3,H-TTL1,L-ECL;VXI-SR2520-25MHz(2,256K)-8;IOCARD4,H-ECL,L-ECL;IOCARDS5,H-TTL2,L-TTL2;

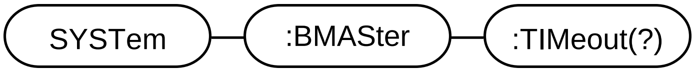
Bus Master Time-Out

Each time the SR2500 parses a command in which the data portion of the command is intended for an I/O module, the SR2510 requests control of the bus in order to transfer the data to the appropriate I/O module. This process is known as becoming the Bus Master.

While acting as the Bus Master, the SR2510 must be able to detect when an I/O module is responding to the read and write cycles initiated by the SR2510. This is done via a handshaking process which is built into the VXI bus structure. If for any reason the handshaking process breaks down, the current bus master must be able abort the current data transfer cycle and return the bus to a quiescent state.

The SR2500 achieves this by having a watchdog timer for all Bus Master operations. If the Bus Master initiated data transfer cycle does not complete in the time period defined for this timer, the bus cycle will be aborted and an error generated. This prevents a handshake failure from hanging up the VXI bus.

Bus Master Time-Out (NON-SCPI)



SYSTem — :BMASter — :TIMeout(?)

The SYSTem:BMASter:TIMeout command will set (or query) the SR2500 Bus Master time-out value.

:TIMeout <bus_master_timeout>

Defines the number of seconds to wait for a Bus Master data transfer cycle to complete, before aborting the bus cycle and generating an error. Values may be specified as a floating point number or in scientific notation. Option S, MS, or US may be used for engineering unit multipliers. The default value is 5.000000e-02.

Parameter Definition **bus_master_timeout** = (1.000000e-3 - 1.000000e+01)

Examples

```
SYSTEM:BMASTER:TIMEOUT 2
SYST:BMAS:TIM 200MS
```

:TIMeout?

Returns the Bus Master time-out value. The Bus Master time-out value is returned in seconds and is represented in scientific notation.

Response bus_master_timeout

Parameter Definition **bus_master_timeout** = (1.000000e-3 - 1.000000e+01)

Examples

SYSTEM:BMASTER:TIMEOUT?
5.000000e-02

SYST:BMAS:TIM?
1.500000e+00

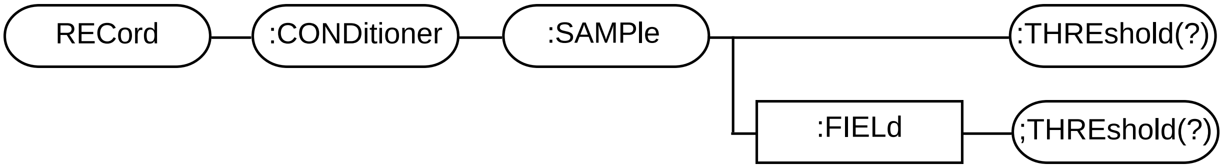
Variable Voltage I/O on the SR2500

The variable voltage I/O cards require externally supplied voltages in 2 sets of 4 voltage each. Each set of 4 voltages consists of a high rail voltage and a low rail voltage used for output, and a high threshold voltage and a low threshold voltage used for input. The high rail voltage will be output on the stimulus pins as a "1". The low rail voltage will be output on the stimulus pins as a "0". The high threshold voltage is used on the input pins to determine when an input should be recorded as a "1"; any input higher than the high threshold voltage is a "1". The low threshold voltage is used on the input pins to determine when an input should be recorded as a "0"; any input lower than the low threshold voltage is a "0". If an input is between the high and low thresholds, it is considered an indeterminate value. Each group of 4 pins (1-4, 5-8, 9-12, 13-16, 17-20, etc.) can select either the A set of 4 voltages or the B set of 4 voltages. Pin groups are assigned the A or B sets by fields; if a field contains more than 1 pin group, then all groups in the field are assigned a voltage set with a single command. If two fields overlap pin groups, the last command issued by the user to select voltage sets will override any previous command. If any pin within a pin group is assigned a voltage set, all pins within the pin group will be assigned the same voltage set. Each field may be queried as to the last voltage set that was selected for that field. When a field is defined, it will default to voltage set A.

Commands for Variable Voltage I/O Cards

RECord	:CONDitioner	:Sample	THREshold(?)
STIMulus	CONDitioner	OFORmat	VOLT?

Selecting Response Threshold Voltage Sets (NON-SCPI)



The RECord:CONDitioner:SAMPle:THREshold command will select the voltage thresholds for all pin groups contained in a field. Because each pin group can have only one voltage set, this command will also have the effect of selecting the high and low rail voltages for the pin groups. If any fields, stimulus or response, overlap, care must be taken to ensure that the pin groups end up with the proper voltage set enabled; the last command to the SR2500 will override any previous commands that selected a voltage set for a pin group.

The RECord:CONDitioner:SAMPle:THREshold? command will return the voltage set selected for the specified field.

:FIELd <name>

The optional FIELd parameter specifies the field where the algorithmic macro commands will be loaded to (or queried from). The destination field must be an Algorithmic Output (ALGO) field type. If the FIELd parameter is used, then the FIELd and THREshold parameters must be separated by a semicolon. If the FIELd parameter is omitted, then the default stimulus field is assumed.

:THREshold

A | B

Examples

REC:COND:SAMPLE:THRES A
RECORD:COND:SAMP:THRE B

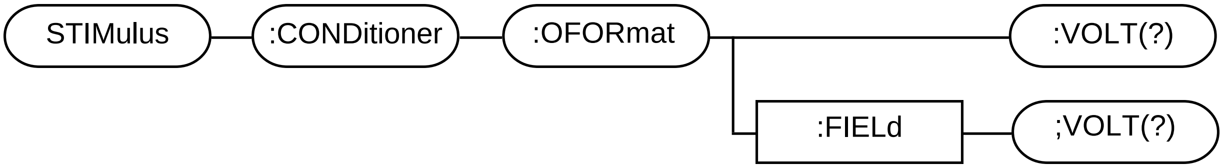
:THREshold?

A | B

Examples

REC:COND:SAMPLE:THRE?

Selecting Stimulus Output Voltage Sets (NON-SCPI)



The STIMulus:CONDitioner:OFORmat:VOLT command will select the voltage output for all pin groups contained in a field. Because each pin group can have only one voltage set, this command will also have the effect of selecting the high and low rail voltages for the pin groups. If any stimulus or response fields overlap, care must be taken to ensure that the pin groups end up with the proper voltage set enabled; the last command issued to the SR2500 will override any previous commands that selected a voltage set for a pin group.

The RECord:CONDitioner:OFORmat:VOLT? command will return the voltage set selected for the specified field.

:FIELd <name>

The optional FIELd parameter specifies the field where the algorithmic macro commands will be loaded to (or queried from). The destination field must be an Algorithmic Output (ALGO) field type. If the FIELd parameter is used, then the FIELd and VOLT parameters must be separated by a semicolon. If the FIELd parameter is omitted, then the default stimulus field is assumed.

:VOLT

A | B

Examples STIM:COND:OFOR:VOLTA

:VOLT?

A | B

Examples STIM:COND:OFORM:VOLT?

IEEE 488.2 Commands

The SR2500 supports the mandatory commands set forth in the IEEE 488.2 specification. The bulk of the mandatory commands utilize a four register set for passing operational information to the system. These registers are the Standard Event Status Register (ESR), Standard Event Status Enable Register (ESE), Status Byte Register (STB) and the Service Request Enable Register (SRE). Together, these register allow certain conditions to generate interrupts to the system Slot 0 Controller, in much the same way that GPIB supports the Service Request (SRQ) function. Many of the commands on the following pages make use of these four registers, so an understanding of the working relationship of these regis-

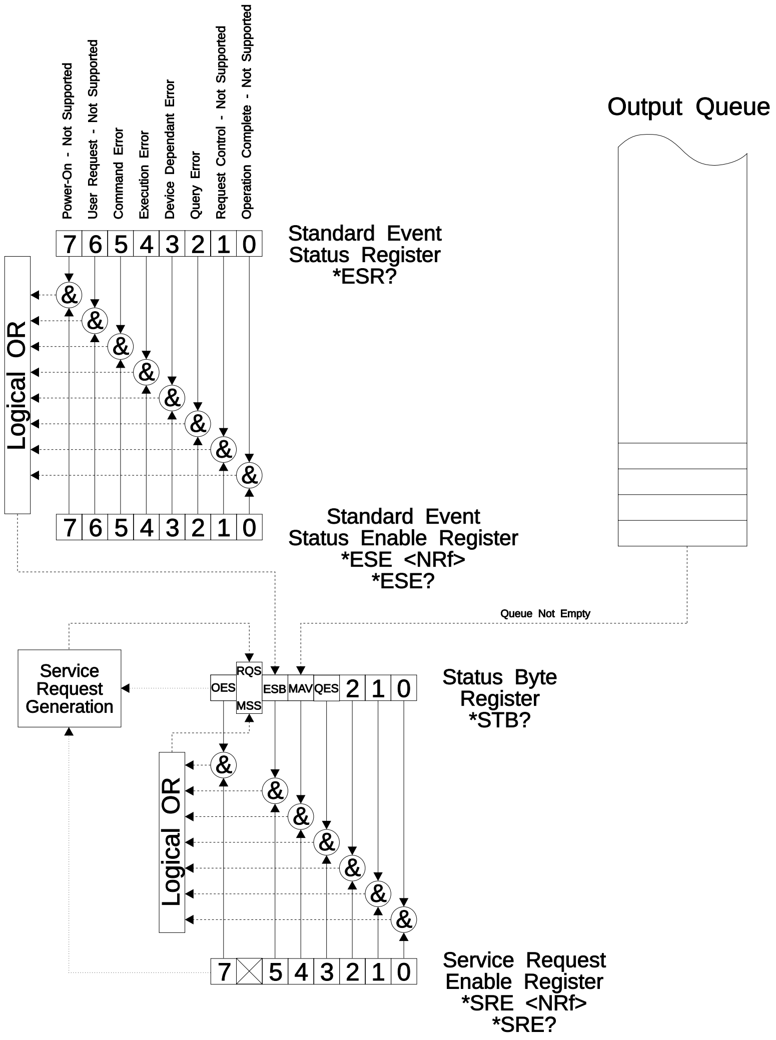


Figure 3-4.
IEEE 488.2 Status and Service Request Registers.

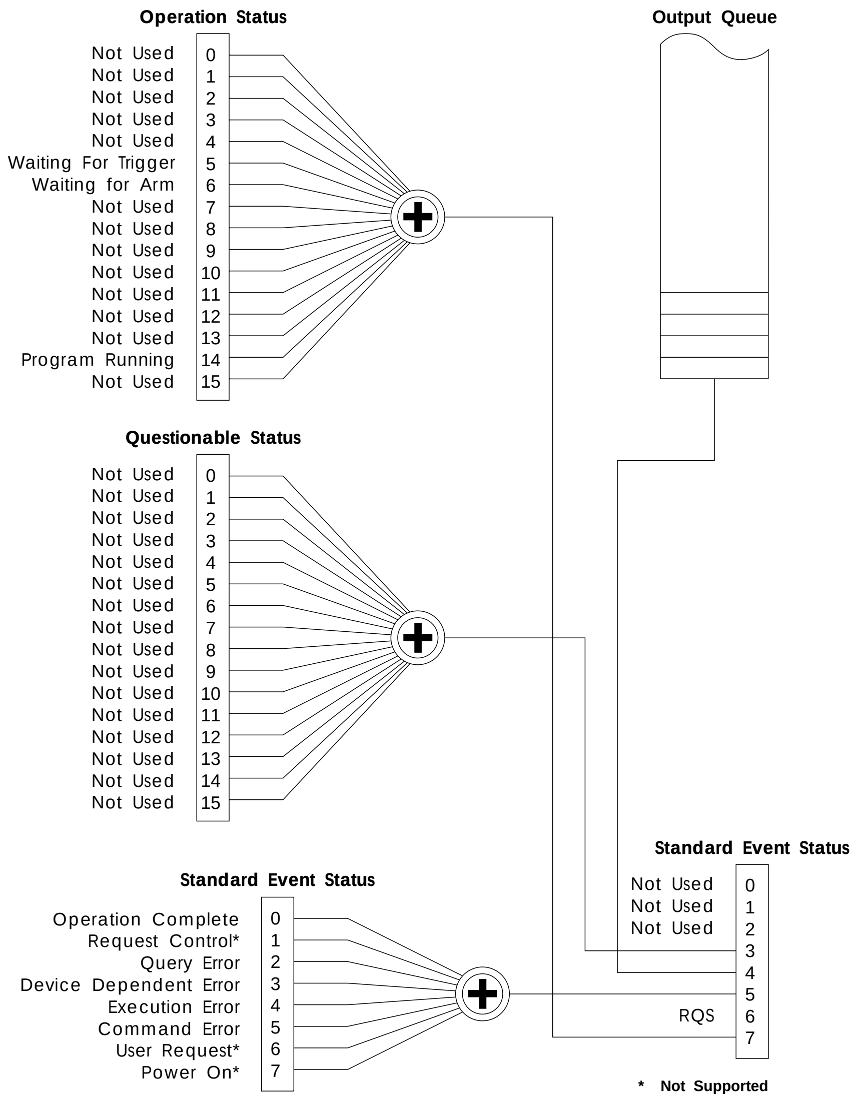
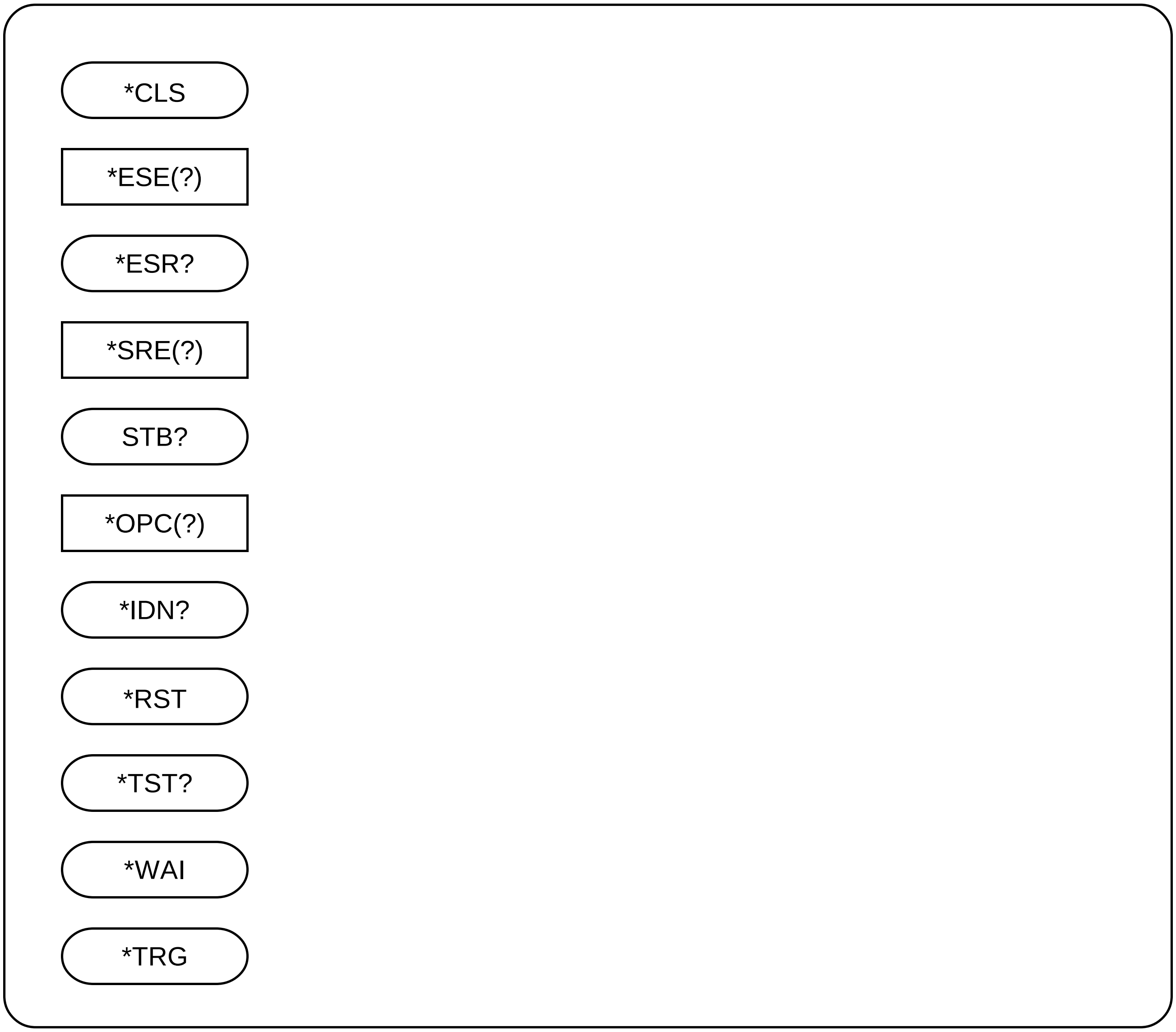


Figure 3-5.
IEEE 488.2 Status and Service Request Registers

ters is required. For this reason, a functional diagram of the 488.2 registers is shown below (figure 3-4), and a diagram of the SCPI Operation and Questionable register connections to the 488.2 status register is shown on the opposite page (figure 3-5). It is also recommended that the user refer to the IEEE 488.2 and the SCPI Syntax and Style manuals for further information.

Some of the 488.2 commands also have parameters associated with them. In all cases the parameters may be entered in either decimal (default format), hexadecimal (#h prefix) or binary (#b prefix) formats.


*CLS
*ESE(?)
*ESR?
*SRE(?)
STB?
*OPC(?)
*IDN?
*RST
*TST?
*WAI
*TRG

IEEE 488.2 Mandatory Commands (IEEE 488.2)

*CLS

The *CLS Clear Status command clears the Operation Status Event Register (OES), the Questionable Status Event Register (QES), the Standard Event Status Register (ESR), and the Status Byte Register (STB). All queues, except the output queue, in the status byte are emptied. The SR2500 is placed into the operation complete - command idle state (OCIS) and operation complete - query idle state (OQIS).

*CLS

Parameter Definition none

Example *CLS

*ESE(?)

The *ESE command sets the contents of the Standard Event Status Enable Register. The 8 bit Enable Register is a mask register used to select which event(s) or bit(s), if any, will be used to set the event status bits in the Standard Event Status Register. Refer to Figure 3-4 for a bit description of the ESE Register. The contents of the Enable Register can be specified in decimal format; or in hexadecimal or binary format by using the '#h' and '#b' prefixes, respectively. The *ESE? query command returns the contents of the Standard Event Status Enable Register.

*ESE <enab_reg>

Parameter Definition **enab_reg** = ((0 - 255) | (#h0 - #hFF) | (#b0 - #b11111111))

Examples *ESE 255

*ESE #b00000001

*ESE?

Response enab_reg

Parameter Definition **enab_reg** = (0 - 255)

Example *ESE?
255

*ESR?

The *ESR query command returns the latched contents of the Standard Event Status Register. The contents of the ESR Register is returned in decimal format. Reading this register clears the latched contents of the ESR Register. Refer to Figure 3-4 for a bit description of the ESR Register.

*ESR?

Response esr_reg

Parameter Definition **esr_reg** = (0 - 255)

Example *ESR?
32

*SRE(?)

The *SRE command sets the contents of the Service Request Enable Register. The 8 bit Enable Register is a mask register used to select which event(s) will cause a generation of a Service Request (SRQ) to the Slot 0 Controller. If an enabled event occurs and interrupt will be sent to the Slot 0. Refer to Figure 3-4 for a bit description of the SRE Register. The contents of the Enable Register can be specified in decimal format; or in hexadecimal or binary format by using the '#h' and '#b' prefixes, respectively. The *SRE? query command returns the contents of the Service Request Enable Register.

*SRE<sre_reg>

Parameter Definition **sre_reg** = ((0 - 255) | (#h0 - #hFF) | (#b0 - #b11111111))

Example *SRE 16

*SRE?

Response sre_reg

Parameter Definition **sre_reg** = (0 - 255)

Example *SRE?
16

*STB?

The *STB? query command returns the contents of the Status Byte Register in decimal format. Refer to Figure 3-4 for a bit description of the STB Register.

*STB?

Parameter Definition stb_reg

stb_reg = (0 - 255)

Example *STB?
64

*OPC(?)

The *OPC Operation Complete command is only used for setting the OPC bit in the Standard Event Status Register when any running test is completed. The *OPC? Operation Complete query command returns an ASCII '1' when the current test is completed. This is not a query of the *OPC command mentioned above. The *OPC? command provides a means of polling the Operation Complete status of the ESR Register without using SRQ.

*OPC

Parameter Definition none

Example *OPC

*OPC?

Response opc_stat

Parameter Definition **opc_stat** = (0 | 1)

Example *OPC?
16

*IDN?

The *IDN? query command returns the identification information for the SR2500.

*IDN?

Response INTERFACE TECHNOLOGY,SR 2510VXI-25MHz(num_io,vec_depth)-cc_log_addr;[VXI-SR2520[GP]-25MHz(num_io,vec_depth)-exp_log_addr;][VXI-RG2500-rg_log_addr;]sw_rev

Parameter Definition **sw_rev** = XX.X, where XX.X represent the software revision number of the SR2510.

vec_depth_addr = (64K | 256K), the vector depth of the module, all modules in a single system must be of the same depth.

cc_log_addr = (1 - 255), the SR2510's VXI logical address.

cc_ser_num = (YYYYYYYYYY-YYYY), where YYYYYYYYYY-YYYY represent the SR2510's serial number.

exp_log_addr = (2 - 255), the SR2520's VXI logical address.

rg_log_addr = (2 - 255), the RG2500's logical address

Example

```
*IDN?
INTERFACE TECHNOLOGY; VXI-SR2510-25MHz(3,256K)-7;VXI-SR2520-25MHz(3,256K-8;VXI-SR2520GP-25MHz(2,256K)-9;1.07
```

*RST

The *RST command resets the SR2500. Any running test will be stopped and the SR2500 will be placed into the operation complete - command idle state (OCIS) and operation complete - query idle state (OQIS). All defined tests will be deleted and all SYSTem parameters to their power-on default condition.

*RST

Parameter Definition none

Example *RST

*TST?

The *TST? query command

*TST?

Response diag_code

Parameter Definition **diag_code** =

Example *TST?

*TRG

The *TRG Software Trigger command will start a test execution. The test must be in the ARMED state and TRIG:SOURce parameter must be set to BUS.

*TRG

Parameter Definition none

Example *TRG

*WAI

The *WAI Wait-to-Continue command is not used by the SR2500. The *WAI is parsed by the SR2500 but no action is taken.

*WAI

Parameter Definition none

Example *WAI

CHAPTER 4

Programming Examples

Program Steps

This chapter deals with the task of programming tests into the system and running them. There are six basic steps to programming a test into the SR2500, which are listed below.

1. Define Test and Global System Parameters
2. Define Stimulus and Expected Response Fields
3. Define Command Macro (CMACRO) Program
4. Load Stimulus and Expected Response Patterns/Algorithms
5. Define Record and CRC Control Parameters
6. Execute the Test

To assist in the definition of tests within the SR2500 system, four work sheets are provided on the following pages (Tables 4-1 to 4-4). Table 4-1 assists in completing steps 1 and 2 above by providing entries for the following parameters:

Define Global System Parameters

- Test Name
- Test Size
- Test Frequency or Period
- System Trigger Source, Slope and Level
- Clock Source, Slope and Level
- 10 MHz Reference Source
- Gate Source, Polarity and Level
- Test Program Loops
- Arm Data Control and Count

Define Stimulus and Expected Response Fields

- Field Name
- Field Type
- Field Pin maps
- Field Radix
- Output Format and Timing
- Sample Format and Timing

Table 4-2 and 4-3 address steps 3 and 4 above, allowing definition of CMACRO test programs and stimulus and response patterns. Since vector sequence control affects both stimulus and response memories, the subsystems are combined into two work sheets with common vector numbers. The parameters defined in tables 4-2 and 4-3 are:

Table 4-1: Test and Field Definition Worksheet

Test and Field Definition Worksheet			
Test Definition Parameters			
Name:	Size:	Program Loops:	10 MHz Ref:
Clock Source:	Frequency:	Clock Slope:	Clock Level:
System Trigger:	Trigger Slope:	Trigger Level:	Arm Data:
Gate Source:	Gate Polarity:	Gate Level:	Arm Count:
Field Definition Parameters			
Field Name:	Field Type:	Field Radix:	Pin List:
Arm Pattern:	Format:	Assert:	Pulse Width:
Field Name:	Field Type:	Field Radix:	Pin List:
Arm Pattern:	Format:	Assert:	Pulse Width:
Field Name:	Field Type:	Field Radix:	Pin List:
Arm Pattern:	Format:	Assert:	Pulse Width:
Field Name:	Field Type:	Field Radix:	Pin List:
Arm Pattern:	Format:	Assert:	Pulse Width:
Field Name:	Field Type:	Field Radix:	Pin List:
Arm Pattern:	Format:	Assert:	Pulse Width:
Field Name:	Field Type:	Field Radix:	Pin List:
Arm Pattern:	Format:	Assert:	Pulse Width:
Field Name:	Field Type:	Field Radix:	Pin List:
Arm Pattern:	Format:	Assert:	Pulse Width:
Field Name:	Field Type:	Field Radix:	Pin List:
Arm Pattern:	Format:	Assert:	Pulse Width:
Field Name:	Field Type:	Field Radix:	Pin List:
Arm Pattern:	Format:	Assert:	Pulse Width:
Field Name:	Field Type:	Field Radix:	Pin List:
Arm Pattern:	Format:	Assert:	Pulse Width:
Field Name:	Field Type:	Field Radix:	Pin List:
Arm Pattern:	Format:	Assert:	Pulse Width:
Field Name:	Field Type:	Field Radix:	Pin List:
Arm Pattern:	Format:	Assert:	Pulse Width:
Field Name:	Field Type:	Field Radix:	Pin List:
Arm Pattern:	Format:	Assert:	Pulse Width:
Comments:			

Define Command Macro (CMACRO) Program

- Labels
- Commands
- Conditions

Load Stimulus and Expected Response Patterns

- Load RAM-Backed Patterns
- Load Algorithmic Instructions

The last work sheet, Table 4-4, assists in the definition of the SR2500 record and CRC controls. The control of the record and CRC sample logic is combined under the TRACE subsystem. Included in the work sheet are the parameters for defining qualifier trigger patterns and trigger combinations. The following items are those defined in Table 4-4:

Define Record and CRC Control Parameters

- Qualifier Trigger Patterns
- Qualifier Trigger Combinations
- Record Filter
- Record Condition
- CRC Sample Condition
- Advance Sequence Condition and Count
- Jump Sequence Condition

These work sheets will be used in the programming examples covered later in this chapter. The examples will start simple and progressively build in function, and complexity, ultimately covering the majority of features and commands of the SR2500 system. Each example will start with an objective, followed by filling in the blanks of a work sheet, and ending with the SCPI commands used to program the SR2500. The examples used are as follows:

4.2 Basic RAM-Backed Pattern Generation
4.3 Using CMACROS for Looping and Branching
4.4 Generating Algorithmic Stimulus Patterns
4.5 Using Real-Time Compare and Algorithmic Expected Responses
4.6 Recording UUT Responses

Examples 3 through 5 will demonstrate testing RAM on a microprocessor based circuit board. A simplified schematic and a wiring diagram are provided to assist in understanding the principles involved in testing the device. Examples 1 and 2 are conceptual examples only and are not based on any specific hardware.

Table 4-2: Test Program and Pattern Definition Worksheet

	Test Program and Pattern Definition Worksheet						
	CMACRO Program			Stimulus/Expected Response Fields			
Vectors	Label	Command	Condition				
Comments:							

Table 4-3: Test Pattern Definition Worksheet

Test Pattern Definition Worksheet (continued)

Stimulus/Expected Response Fields

Comments:

Table 4-4: Record Control Definition Worksheet

Fields ->	Qualifier Trigger Definitions												
	ADDRE	DATAE	WR_RDE										
1													
2													
3													
4													
5													
6													
7													
8													

Qualifier Combinations							
1	2	3	4	5	6	7	8

	Trace Sequences							
	Record Filter	Record On	CRC Sample On	Advance On	Advance Count	Jump On	Jump To	Stop Test
1								
2								
3								
4								
5								
6								
7								
8								
9								
10								
11								
12								
13								
14								
15								
16								

RAM-Backed Pattern Generation

The SR2500 provides a very simple process for generating patterns. In this example only generic test definitions, field definitions and pattern RAM downloading are performed. Default values will be used for all other parameters, including CMACRO program, data formatting and format timing. In this example the real-time compare and data record features will not be used.

Table 4-5: Basic Stimulus Test Definition

Test and Field Definition Worksheet			
Test Definition Parameters			
Name: **TEST_1**	Size: **8**	Program Loops: **1**	10 MHz Ref: **N/C**
Clock Source: **N/C**	Frequency: **25 MHz**	Clock Slope: **N/A**	Clock Level: **N/A**
System Trigger: **Bus**	Trigger Slope: **N/A**	Trigger Level: **N/A**	Arm Data: **Off**
Gate Source: **N/C**	Gate Polarity: **N/C**	Gate Level: **N/C**	Arm Count: **1**
Field Definition Parameters			
Field Name: **ONE**	Field Type: **OT**	Field Radix: **HEX**	Pin List: **C1P8-1**
Arm Pattern: **N/A**	Format: **N/C**	Assert: **N/A**	Pulse Width: **N/A**
Field Name: **TWO**	Field Type: **OT**	Field Radix: **HEX**	Pin List: **C1P16-9**
Arm Pattern: **N/A**	Format: **N/C**	Assert: **N/A**	Pulse Width: **N/A**

Table 4-6: Basic Stimulus Test Patterns

	Test Program and Pattern Definition Worksheet						
	CMACRO Program			**Stimulus/Expected Response Fields**			
Vectors	Label	Command	Condition	ONE	TWO		
1		N/C		00	01		
2		N/C		01	02		
3		N/C		02	04		
4		N/C		03	08		
5		N/C		04	10		
6		N/C		05	20		
7		N/C		06	40		
8		N/C		07	80		

When a SR2500 test is defined, certain test parameters are set to their *Default States*. The stimulus and expected response memories are set to all 0's. The tristate and don'tcare memories are set to tristate all outputs and mask all inputs. The CMACRO program is set to all OUTPUT statements, which instruct the test sequence state machine to cycle through all test vectors in sequential order. In order to generate a simple stimulus pattern, you need only define a few global test parameters, load the output and tristate memories with the appropriate values, and execute the test. Refer to tables 4-5 and 4-6 for the test setup parameters. In this example only two fields are defined. Both fields are 8 bit wide OT type fields (combination of Output and Tristate fields). The test will sequence one time through 8 test vectors at a 25 MHz rate (1 vector every 40ns), and will generate an incrementing pattern on field ONE and a walking 1 pattern on field TWO. Parameters with a N/C indicate "No Change" (default) values. N/A indicates "Not Applicable".

The commented program listing generated from the tables in Figure 1 is listed below. To generate the test program defined by this example, the SCPI commands shown below, minus the comments, are sent to the SR2510. Various methods may be employed for performing this task. All VXI host computers provide function calls for sending ASCII strings to VXI modules. This is one method for loading the test program. Another would be to include the commands in an ASCII text file, and using a function call to send the contents of the file to the specified VXI module. In each case, the end result is the same.

/*** Test Program Listing for TEST_1 *****/**

/* Define Test Parameters */

```
TEST:DEF TEST_1:SIZE 8
SYST:PROG 1
SYST:FREQ 25.00MHz
TRIG:SYST:SOUR BUS
```

/* Define Fields */

```
FIELD:DEF ONE:TYPE OT:PIN C1P8-1
FIELD:NAME ONE:RADIX HEX
FIELD:DEF TWO:TYPE OT:PIN C1P16-9
FIELD:NAME TWO:RADIX HEX
```

/* Load Test Patterns */

```
STIM:FIEL ONE;VEC 1;COUN 8;DATA:PATT 0,1,2,3,4,5,6,7
STIM:FIEL TWO;VEC 1;COUN 8;DATA:PATT 1,2,4,8,10,20,40,80
```

/* Define Run-Time Parameters */

```
ARM:COUN 1
STIM:ARMD:MODE OFF
```

/* Initiate test and trigger */

```
INIT
*TRG
```

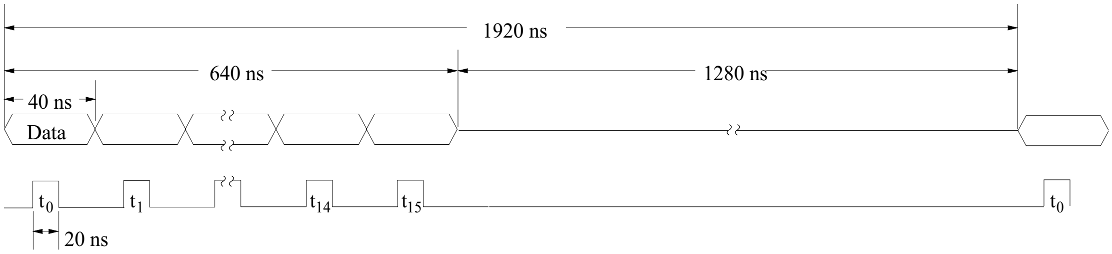


Figure 4-1.
Data Transfer Example Timing

Table 4-7: Data Transfer Test and Field Definitions

Test and Field Definition Worksheet							
Test Definition Parameters							
Name:	**TEST_2**	Size:	**20**	Program Loops:	**1**	10 MHz Ref:	**N/C**
Clock Source:	**N/C**	Frequency:	**25 MHz**	Clock Slope:	**N/A**	Clock Level:	**N/A**
System Trigger:	**EXT**	Trigger Slope:	**POS**	Trigger Level:	**2.2V**	Arm Data:	**OFF**
Gate Source:	**N/C**	Gate Polarity:	**N/C**	Gate Level:	**N/C**	Arm Count:	**1**
Field Definition Parameters							
Field Name:	**DATA**	Field Type:	**OT**	Field Radix:	**HEX**	Pin List:	**C1P16-1**
Arm Pattern:	**0000**	Format:	**N/C**	Assert:	**N/A**	Pulse Width:	**N/A**
Field Name:	**CLOCK**	Field Type:	**OT**	Field Radix:	**HEX**	Pin List:	**C1P17**
Arm Pattern:	**0**	Format:	**RZ**	Assert:	**10ns**	Pulse Width:	**20ns**

A shortcut method to load the data patterns is to use the FILL functions. The examples below show an alternate method of filling field ONE with an incrementing pattern, and field TWO with a walking 1 pattern. While not much of a time saving with only 8 vectors in each field, the saving become significant when thousands of vectors are used.

/* Alternate Method of Loading Memory */

```
STIM:VEC 1;COUN 8;DATA:FIEL ONE;FILL:TYPE INC;PATT 0;EXEC
STIM:VEC 1;COUN 8;DATA:FIEL TWO;FILL:TYPE WLK1;PATT 1;EXEC
```

Table 4-8: Data Transfer Program and Patterns

	Test Program and Pattern Definition Worksheet						
	CMACRO Program			Stimulus/Expected Response Fields			
Vectors	Label	Command	Condition	DATA	CLOCK		
1		StartProgram		F324	1		
2		StartLoop until	Trigger=TRUE	9553	1		
3				0424	1		
4				0424	1		
5				BA1E	1		
6				DA1E	1		
7				14A1	1		
8				7000	1		
9				8591	1		
10				0515	1		
11				0129	1		
12				6891	1		
13				0615	1		
14				8891	1		
15				0122	1		
16				1691	1		
17		WordLoop until	COUNt==32	0000	0		
18		EndLoop		F324	1		
19		EndProgram		0000	0		
20		N/C		N/C	N/C		

Using CMACROS and Data Formatting

SR2500 command macro (CMACRO) programs add another level of flexibility to an SR2500 based test, the ability to loop on patterns and change test program flow. Looping and branching may be done unconditionally or conditionally. The SR2500 also supports 5 data formats which may be applied to any of the stimulus pins.

The next example uses a CMACRO program to produce a burst of high speed signals followed by a long dead time. The burst is meant to simulate an asynchronous transfer of data across a communications bus. A single clock pulse is generated for each 16 bit data word transferred and a frame of data consists of 16 data words. The clock uses a *Return-to-Zero* (RZ) format and is placed in the middle of when the data word is valid. After a frame is transferred the bus goes inactive for 2 frame times. Refer to the timing diagram in figure 4-1 for further details.

A typical method to achieve Mixed High Speed and Low Speed Timing would be to run the test at the highest speed in order to provide the high speed burst, and then pad multiple memory locations with the same data pattern in order to create the static dead time. The disadvantage of this approach is that memory is wasted and programming is complicated. The approach used in this example is to define a unique data pattern for each of the 16 data vectors, and then loop on a single vector for 32 cycles, thus creating the 2 frame dead time using only a single test vector. The test will start upon detection of an external trigger, loop sending the same 16 data words indefinitely, until another external trigger pulse is detected.

Like the previous example, tables 4-7 and 4-8 define the test parameters to generate the required test patterns and the program listing is as follows.

/*** Test Program Listing for TEST_2 *****/**

/* Define Test Parameters */

```
TEST:DEF TEST_2:SIZE 20
SYST:PROG 1
SYST:FREQ 25MHz
TRIG:SYST:SOUR EXT
TRIG:SYST:SLOP POS
TRIG:SYST:LEV +2.20
```

/* Define Fields */

```
FIELD:DEF DATA:TYPE OT:PIN C1P16-1
STIM:ARMD:FIELD DATA;PATT #h0
FIELD:DEF CLOCK:TYPE OT:PIN C1P17
STIM:ARMD:FIELD CLOCK;PATT #h0
STIM:COND:OFOR:FIELD CLOCK;MODE RZ,10.000000NS,20.000000NS
```

/* Define CMACRO Program */

```
STIM:VEC 1;CMAC:DEF (SP(OUT))
STIM:VEC 2;CMAC:DEF (SL(OUT(STRI == TRUE)))
STIM:VEC 3;CMAC:DEF (OUT(OUT))
STIM:VEC 4;CMAC:DEF (OUT(OUT))
STIM:VEC 5;CMAC:DEF (OUT(OUT))
STIM:VEC 6;CMAC:DEF (OUT(OUT))
STIM:VEC 7;CMAC:DEF (OUT(OUT))
STIM:VEC 8;CMAC:DEF (OUT(OUT))
STIM:VEC 9;CMAC:DEF (OUT(OUT))
STIM:VEC 10;CMAC:DEF (OUT(OUT))
STIM:VEC 11;CMAC:DEF (OUT(OUT))
STIM:VEC 12;CMAC:DEF (OUT(OUT))
STIM:VEC 13;CMAC:DEF (OUT(OUT))
STIM:VEC 14;CMAC:DEF (OUT(OUT))
STIM:VEC 15;CMAC:DEF (OUT(OUT))
```

```
STIM:VEC 16;CMAC:DEF (OUT(OUT))
STIM:VEC 17;CMAC:DEF (WL(OUT(COUN == 32)))
STIM:VEC 18;CMAC:DEF (EL(OUT))
STIM:VEC 19;CMAC:DEF (OUT(OUT))
```

/* Load Test Patterns */

```
STIM:FIEL DATA;VEC 1;COUN 19;DATA:PATT F324,9553,424,424,BA1E,DA1E,14A1,
7000,8591,515,129,6891,615,8891,122,1691,0,F32,0
STIM:FIEL CLOCK;VEC 1;COUN 19;DATA:PATT 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1, 0,1,0
```

/* Define Run-Time Parameters */

```
STIM:ARMD:MODE ON
ARM:COUN 1
```

/* Initiate test */

```
INIT
```

/* Wait for External Trigger to start Test */

/* Wait for External Trigger to complete test*/

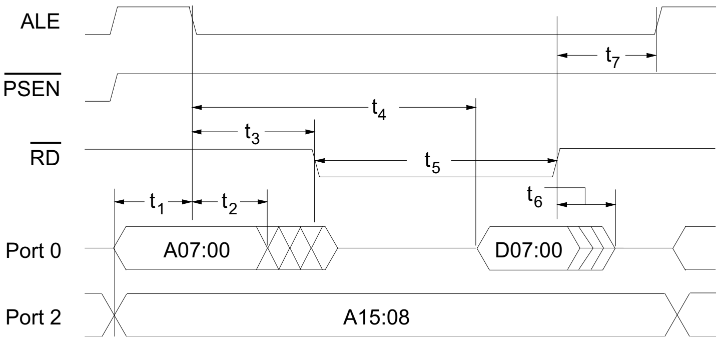


Figure 4-2.
8051 Read Cycle Timing.

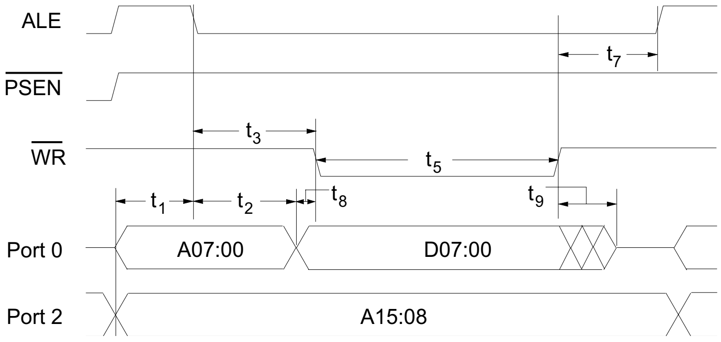


Figure 4-3.
8051 Write Cycle Timing.

Table 4-9: 8051 Timing Parameters

Symbol	Parameter	Min	Max	Unit
t1	Address Valid to ALE Low	45		ns
t2	Address Hold After ALE Low	48		ns
t3	ALE Low to RD/WR Low	225	300	ns
t4	ALE Low to Valid Data In		517	ns
t5	RD/WR Pulse Width	400		ns
t6	Data Float After RD		97	ns
t7	RD/WR High to ALE High	43	123	ns
t8	Data Valid to WR Transition	23		ns
t9	Data Hold After WR	33		ns

Generating Algorithmic Stimulus Patterns

In each of the previous examples, the stimulus patterns were pre-loaded into RAM for output to the UUT. One of the more unique features of the SR2500, and the more useful, is the ability to generate pattern algorithmically. This function is especially useful in bus emulation and RAM test applications, where the data on address and data busses can be represented algorithmically. The following three examples will illustrate this and other advanced features of the SR2500 by testing the RAM on a microprocessor based circuit board. A brief discussion of the UUT will help understand the application of the SR2500 to the test.

The UUT is a circuit board based on the Intel 8051 microcontroller. Refer to the read and write timing diagrams, the SR2500 to UUT interconnect schematic and the timing table (figures 4-2, 4-3 and 4-4, and table 4-9, respectively) for the following discussion. The 8051 utilizes a 16 bit address bus and an 8 bit data bus. The lower 8 address lines are multiplexed with the data bus. When the address is valid on the bus, the Address Latch Enable (ALE) signal will latch the lower address into an address latch. Then the bus is free to either read or write data to the peripheral device, in this case, RAM.

In order to emulate the 8051 cycle timing, you must determine the minimum and maximum cycle times for the read and write cycles. Adding the timing parameters t1 + t3 + t5 + t7 for the read cycle yields 671ns min (1.490 MHz) and 851ns max (1.175 MHz). Adding the timing parameters t1 + t3 + t5 + t7 for the write cycle yields the same values. So to emulate 8051 timing, the SR2500 test rate may be programmed anywhere within the two ranges. In this case, the SR2500 will be programmed with a test rate of 1.481 MHz (675.0ns).

The next step is to determine the field requirements. There are two busses to emulate, plus a handful of control signals. However, keep in mind that the lower 8 address signals are sent on the same bus as the 8 data signals. Since the intent of this test is to use algorithmic pattern generation, three busses are defined in the SR2500, and two of them *Multiplexed* together using the Return-to-Inhibit (Tristate) data formatting (see figure 4-4). In this way, the upper address, lower address and data patterns may be generated using algorithms.

NOT AVAILABLE AT TIME OF PUBLICATION

Figure 4-4.
8051 Emulation Wiring Schematic.

Table 4-10: RAM_TEST Test and Field Definitions

Test and Field Definition Worksheet			
Test Definition Parameters			
Name: **RAM_TEST**	Size: **1024**	Program Loops: **1**	10 MHz Ref: **N/C**
Clock Source: **N/C**	Frequency: **1.5 MHz**	Clock Slope: **N/A**	Clock Level: **N/A**
System Trigger: **Bus**	Trigger Slope: **N/A**	Trigger Level: **N/A**	Arm Data: **ON**
Gate Source: **N/C**	Gate Polarity: **N/A**	Gate Level: **N/A**	Arm Count: **1**
Field Definition Parameters			
Field Name: **A15_08O**	Field Type: **ALGO**	Field Radix: **HEX**	Pin List: **C1P16-9**
Arm Pattern: **FF**	Format: **N/C**	Assert: **N/A**	Pulse Width: **N/A**
Field Name: **A15_08T**	Field Type: **OT**	Field Radix: **HEX**	Pin List: **C1P16-9**
Arm Pattern: **FF**	Format: **N/C**	Assert: **N/A**	Pulse Width: **N/A**
Field Name: **A07_00O**	Field Type: **ALGO**	Field Radix: **HEX**	Pin List: **C1P8-1**
Arm Pattern: **FF**	Format: **RI**	Assert: **0.0ns**	Pulse Width: **80.0ns**
Field Name: **A07_00T**	Field Type: **OT**	Field Radix: **HEX**	Pin List: **C1P8-1**
Arm Pattern: **FF**	Format: **N/C**	Assert: **N/A**	Pulse Width: **N/A**
Field Name: **D07_00O**	Field Type: **ALGO**	Field Radix: **HEX**	Pin List: **C2P24-17**
Arm Pattern: **FF**	Format: **RI**	Assert: **200.0ns**	Pulse Width: **465.0ns**
Field Name: **D07_00T**	Field Type: **OT**	Field Radix: **HEX**	Pin List: **C2P24-17**
Arm Pattern: **FF**	Format: **N/C**	Assert: **N/A**	Pulse Width: **N/A**
Field Name: **ALE**	Field Type: **OT**	Field Radix: **BIN**	Pin List: **C3P25**
Arm Pattern: **0**	Format: **RZ**	Assert: **0.0ns**	Pulse Width: **30.0ns**
Field Name: **WRO**	Field Type: **OT**	Field Radix: **BIN**	Pin List: **C3P28**
Arm Pattern: **1**	Format: **RONE**	Assert: **230.0ns**	Pulse Width: **400.0ns**
Field Name: **RDO**	Field Type: **OT**	Field Radix: **BIN**	Pin List: **C3P27**
Arm Pattern: **1**	Format: **RONE**	Assert: **230.0ns**	Pulse Width: **400.0ns**
Field Name: **PSENO**	Field Type: **OT**	Field Radix: **BIN**	Pin List: **C2P26**
Arm Pattern: **1**	Format: **N/C**	Assert: **N/A**	Pulse Width: **N/A**

Another technique used is to tie the address busses (upper and lower), the RD signal and the WR signal to input pins, in addition to the data bus which is bidirectional by definition, allowing recording of all significant information for debugging failures. This is discussed further in following examples. This example deals with the task of writing data to RAM algorithmically. Refer to table 4-10, 4-11A and 4-11B for SR2500 test definition parameters and pattern definition. For clarity, tables 4-11A and 4-11B are shown on opposite pages so that vector parameters are aligned.

Table 4-11A: RAM_TEST Program and Pattern Definition

	Test Program and Pattern Definition Worksheet							
	CMACRO Program			Stimulus/Expected Response Fields				
Vectors	Label	Command	Condition	A15_08O	A15_08T	A07_00O	A07_00T
1		StartProgram		NONA	1F	NONA	FF
2		StartLoop until	COUNt==96	INC		INC	
3		WordLoop until	COUNt==254	HOLDA		INC	
4		EndLoop		HOLDA		INC	
5		EndProgram		NONA	FF	NONA	FF

When a test of a particular size is defined, stimulus, expected response and record vectors are reserved. Note in this example that the test size is 1024 vectors, yet only 5 vectors are actually used. Later expansions of this test will add data recording to the existing stimulus pattern generation. Defining a test size of 1024 vectors allows recording of up to 1024 samples, even though only 10 or 20 vectors are used in the stimulus and expected response portion of the program. It is important to consider the total test resource requirements when programming the SR2500.

Figure 4-5 depicts the cycle timing programmed into test RAM_TEST. The timing diagram actually shows a single write cycle. The read cycle is identical except that the SR2500 never drives the AD07:00 bus during the data valid time (tristates field D07_00O). Note also that this is the timing for a single vector. All vectors will share the same timing for the duration of the test. Actual data patterns that will be output to the UUT, when valid as defined by the data format parameters (and depicted graphically in figure 4-5), are defined in table 4-11A and 4-11B.

Two fields were defined for the address busses and the data bus. This is because all three busses will be generating algorithmic data. An algorithmic output type field (ALGO) does not provide for tristate control. A second field of type *OT* (combined Output and Tristate memories) was created and mapped to the same pins in order to provide tristate control for these busses. As the default condition of all tristate memories is to disable their respective outputs, not defining a field that provides control of the tristate memory would result in all three busses floating in a high impedance state for the duration of the test.

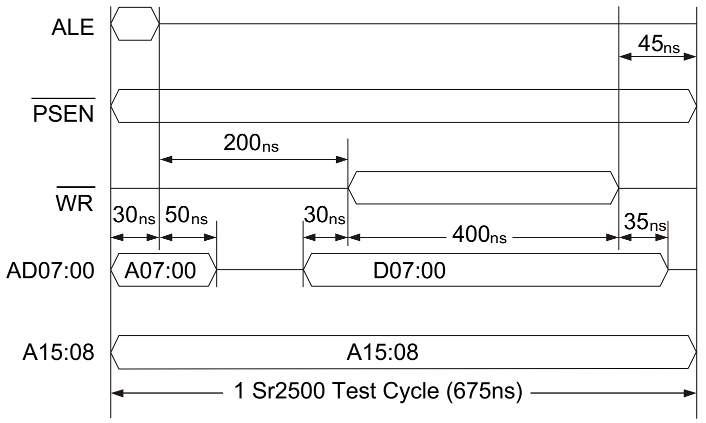


Figure 4-5.
RAM_TEST Write Cycle Timing.

Table 4-11B: RAM_TEST Pattern Definition (continued)

Test Pattern Definition Worksheet (continued)								
Stimulus/Expected Response Fields								
D07_00O	D07_00T	ALE	WRO	RDO	PSEN			
NONA	55	0	1	1	1			
XOR	FF	1	0	1	1			
XOR	FF	1	0	1	1			
XOR	FF	1	0	1	1			
NONA	FF	0	1	1	1			

The CMACRO test program and accompanying data patterns will write a 24K block of data to the RAM. The base address of the RAM is 0x2000. In the test program, the address busses (ALU registers) are initialized to 0x1FFF at vector 1 (0x1F for A15_08O and 0xFF for A07_00O). However, since the ALE strobe is inactive for this vector, the address is not latched into the address latch. The data bus is initialized to 0x55 (D07_00O), but the write strobe is inactive, so data is not written. Vectors 2-4 perform the 24K write process. Vector 2 defines a CMACRO start loop for 96 iterations. The address busses are incremented by one, providing the base RAM address of 0x2000, ALE is strobed high, the data bus ALU register (which was initialized to 0x55) was exclusive ORed with 0xFF, resulting in a pattern of 0xAA, and the write strobe was pulsed causing the 0xAA to be written to address 0x2000.

The CMACRO instruction at vector 3 defines a single vector loop for 254 test cycles. For each of the 254 iterations of vector 3, the lower address bus is incremented once, and the data bus is complemented (exclusive ORed with 0xFF). The net result of vector 3 is an additional 254 write cycles to successive addresses with complementing data.

Vector 4 terminates the loop that was defined at vector 2. One last increment of the lower address bus and one last complement of the data bus are performed. The test then loops back to vector 2, where both address busses are incremented (now at address 0x2100), the data is complemented and the process is repeated. One single pass of vectors 2-4 provide 256 writes of complementing data to successive address locations. Looping through this range of vectors 96 times results in the 24K write cycles (96 x 256 = 24576).

If the address busses were wider, changing only the loop count values at vectors 2 and 3 would accommodate testing larger memories, up to a full 32 bit address bus using the same 3 vectors. Similar techniques could also be employed to test devices with addressing beyond 32 bits. Following is a complete listing to generate test RAM_TEST.

/*** TEST PROGRAM LISTING FOR RAM_TEST *****/**

/* DEFINE TEST PARAMETERS */

```
TEST:DEF RAM_TEST:SIZE 1024
SYST:PROG 1
SYST:FREQ 1.48MHz
TRIG:SYST:SOUR BUS
```

/* DEFINE FIELDS */

```
FIELD:DEF A15_08O:TYPE ALGO:PIN C1P16-9
FIEL:NAME A15_08O:RAD HEX
FIELD:DEF A15_08T:TYPE OT:PIN C1P16-9
FIEL:NAME A15_08T:RAD HEX
FIELD:DEF A07_00O:TYPE ALGO:PIN C1P8-1
FIEL:NAME A07_00O:RAD HEX
FIELD:DEF A07_00T:TYPE OT:PIN C1P8-1
FIEL:NAME A07_00T:RAD HEX
FIELD:DEF D07_00O:TYPE ALGO:PIN C2P24-17
FIEL:NAME D07_00O:RAD HEX
FIELD:DEF D07_00T:TYPE OT:PIN C2P24-17
FIEL:NAME D07_00T:RAD HEX
FIELD:DEF ALE:TYPE OT:PIN C3P25
FIEL:NAME ALE:RAD BIN
FIELD:DEF WRO:TYPE OT:PIN C3P28
FIEL:NAME WRO:RAD BIN
FIELD:DEF RDO:TYPE OT:PIN C3P27
FIEL:NAME RDO:RAD BIN
FIELD:DEF PSEN:TYPE OT:PIN C2P26
FIEL:NAME PSEN:RAD BIN
```

/* DEFINE STIMULUS DATA FORMATS */

```
STIM:COND:OFOR:FIELD A07_00O;MODE RI,0.000000NS,80.000000NS
STIM:COND:OFOR:FIELD D07_00O;MODE RI,200.500000NS,465.000000NS
STIM:COND:OFOR:FIELD ALE;MODE RZ,0.000000NS,30.000000NS
STIM:COND:OFOR:FIELD WRO;MODE RONE,230.000000NS,400.000000NS
STIM:COND:OFOR:FIELD RDO;MODE RONE,230.000000NS,400.000000NS
```

/* DEFINE CMACRO PROGRAM */

```
STIM:VEC 1;CMAC:DEF (SP(OUT))
STIM:VEC 2;CMAC:DEF (SL(OUT(COUN == 96)))
STIM:VEC 3;CMAC:DEF (WL(OUT(COUN == 254)))
STIM:VEC 4;CMAC:DEF (EL(OUT))
STIM:VEC 5;CMAC:DEF (EP(OUT))
```

/* LOAD TEST PATTERNS AND ALGORITHMS */

```
STIM:FIEL A15_08O;VEC 1;COUN 5;AMAC:PATT NONA,INC,HOLDALL,HOLDALL,NONA
STIM:FIEL A15_08T;VEC 1;COUN 5;DATA:PATT 1F,C0,20,20,FF
STIM:FIEL A07_00O;VEC 1;COUN 5;AMAC:PATT NONA,INC,INC,INC,NONA
STIM:FIEL A07_00T;VEC 1;COUN 5;DATA:PATT FF,0,0,0,FF
STIM:FIEL D07_00O;VEC 1;COUN 5;AMAC:PATT NONA,XOR,XOR,XOR,NONA
STIM:FIEL D07_00T;VEC 1;COUN 5;DATA:PATT 55,FF,FF,FF,FF
STIM:FIEL ALE;VEC 1;COUN 5;DATA:PATT 0,1,1,1,0
STIM:FIEL WRO;VEC 1;COUN 5;DATA:PATT 1,0,0,0,1
STIM:FIEL RDO;VEC 1;COUN 5;DATA:PATT 1,1,1,1,1
STIM:FIEL PSENO;VEC 1;COUN 5;DATA:PATT 1,1,1,1,1
```

/* DEFINE RUN-TIME PARAMETERS */

```
STIM:ARMD:MODE ON
STIM:ARMD:FIELD A15_08T;PATT #hFF
STIM:ARMD:FIELD A07_00T;PATT #hFF
STIM:ARMD:FIELD D07_00T;PATT #hFF
STIM:ARMD:FIELD ALE;PATT #h0
STIM:ARMD:FIELD WRO;PATT #h1
STIM:ARMD:FIELD RDO;PATT #h1
STIM:ARMD:FIELD PSEN;PATT #h1
ARM:COUN 1
```

/* INITIATE TEST AND TRIGGER */

```
INIT
*TRG
```

Using Real-Time Compare and Algorithmic Expected Responses

The previous example demonstrated several important concepts. First was how to multiplex two or more data sources onto a common set of pins using the Return-to-Inhibit data format. The example used was to multiplex address and data information onto a common bus, but the same technique could be used to multiplex row address and column address onto a common bus for testing dynamic RAM.

The second important concept introduced in the previous programming example was how using CMACRO looping in combination with algorithmic pattern generation yielded a test that, while only requiring 5 test vectors total, tested 24K of RAM, and that the test could easily have been used to test 4GigaBytes of RAM (2^{32}) and beyond.

Finally, the last example demonstrated the process for determining stimulus timing parameters and how to program this information into the SR2500. Also included were the use of several stimulus data formats, Return-to-Zero, Return-to-One and Return-to-Inhibit (RZ, RONE and RI, respectively).

This program example builds upon the last by adding the *Real-Time Compare* functions into the test. Real-time compare utilizes the Expected Response memories (Expect, Don'tcare and Algorithmic Expect) to perform a hardware comparison between an expected UUT response and the actual UUT response. Response memory vector sequence is controlled via the CMACRO program exactly as is the Stimulus memory, so each vector may have its own unique expected response. Expected responses may also be generated algorithmically, like stimulus patterns. The expected algorithm may be the same as the stimulus algorithm, or different.

Using the real-time compare function of the SR2500 adds two new concepts, in addition to the real-time compare itself. First is *Sample Timing*, and second is use of the *Compare Error Flag*. Anytime an input function is used (record data, real-time compare, or CRC calculation), it is imperative that the input sample time is defined. Each SR2510 module allows the definition of 2 *sample times* within the test cycle, and all input functions are tied to one or the other of those sample times. In the case of the *Window Compare* mode, both sample times are used. When both sample clocks are used on a SR2510, *Sample Time Ordering* becomes important. You must define the later sample time first, followed by the earlier sample time. For the purpose of this program example, the sample times are referred to as *Sample Edges*.

Each time a test is initiated within the SR2500, the internal response pipeline has unknown data left over from the previous test or from the power-on sequence. As such, the Compare Error Flag is asserted, indicating a compare error, even though a sample comparison has not taken place. If it is desired to use the error flag for test control or as a pass/fail indicator, it is necessary to reset the error flag. To reset the error flag, it is also necessary to clock known, error free data into the response pipeline, and issue the CMACRO command to reset the error flag. A recommended procedure is to use the first two vectors in your test to loop for a few test cycles, with all inputs masked, and then reset the error flag upon exit from the loop. The vector at which the error flag is reset becomes the first vector in the test. An example of this procedure is provided in this revision of RAM_TEST.

To add the real-time compare (response) function, several new fields must be defined, one response field for each group of pins where data will be compared or recorded. Refer to table 4-12 for the field definitions being added to the last example. Each of these fields is a response type field. Where the stimulus address and data bus fields were defined as algorithmic output (ALGO) type fields, the response address and data bus fields are defined as algorithmic expect (ALGE) type fields. A similar relationship exists between expect and don'tcare fields as exists between output and tristate fields. An ALGE type field does not have control over the don'tcare, or mask, condition of an input. So, two response field types are defined for the address and data bus fields. This provides the ability to ignore certain pins when performing real-time compare functions, or when calculating CRC values. Note the address fields are sampled at the same time within the test cycle as the generation of the ALE signal, and the data bus, WR strobe and RD strobe are sample when the data on the data bus is valid for either a read or write cycle (refer to table 4-12 and figure 4-6).

Table 4-12: RAM_TEST Response Fields

Field Definition Parameters							
Field Name:	**A15_08E**	Field Type:	**ALGE**	Field Radix:	**HEX**	Pin List:	**C1P16-9**
Arm Pattern:	**N/A**	Format:	**Sample Edge**	Assert:	**30.0ns**	Pulse Width:	**N/A**
Field Name:	**A15_08M**	Field Type:	**ED**	Field Radix:	**HEX**	Pin List:	**C1P16-9**
Arm Pattern:	**N/A**	Format:	**N/C**	Assert:	**N/A**	Pulse Width:	**N/A**
Field Name:	**A07_00E**	Field Type:	**ALGE**	Field Radix:	**HEX**	Pin List:	**C1P8-1**
Arm Pattern:	**N/A**	Format:	**Sample Edge**	Assert:	**30.0ns**	Pulse Width:	**N/A**
Field Name:	**A07_00M**	Field Type:	**ED**	Field Radix:	**HEX**	Pin List:	**C1P8-1**
Arm Pattern:	**N/A**	Format:	**N/C**	Assert:	**N/A**	Pulse Width:	**N/A**
Field Name:	**D07_00E**	Field Type:	**ALGE**	Field Radix:	**HEX**	Pin List:	**C2P24-17**
Arm Pattern:	**N/A**	Format:	**Sample Edge**	Assert:	**550.0ns**	Pulse Width:	**N/A**
Field Name:	**D07_00M**	Field Type:	**ED**	Field Radix:	**HEX**	Pin List:	**C2P24-17**
Arm Pattern:	**N/A**	Format:	**N/C**	Assert:	**N/A**	Pulse Width:	**N/A**
Field Name:	**WRE**	Field Type:	**ED**	Field Radix:	**BIN**	Pin List:	**C3P28**
Arm Pattern:	**N/A**	Format:	**Sample Edge**	Assert:	**550.0ns**	Pulse Width:	**N/A**
Field Name:	**RDE**	Field Type:	**ED**	Field Radix:	**BIN**	Pin List:	**C3P27**
Arm Pattern:	**N/A**	Format:	**Sample Edge**	Assert:	**550.0ns**	Pulse Width:	**N/A**

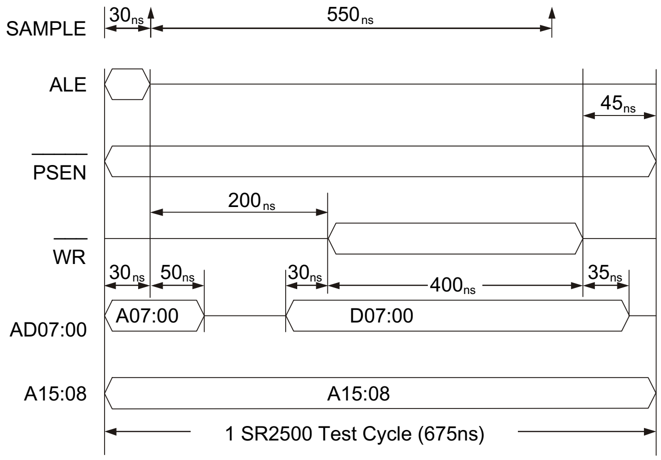


Figure 4-6.
RAM_TEST Read/Write and Sample Timing.

Tables 4-13A and 4-13B show the new CMACRO program, with the pipe line flush procedure inserted at vectors 1, 2 and 3, and the added response field pattern definitions and algorithmic commands. The test has also been extended to include verifying the contents of the address and data busses during the write cycles by defining the stimulus and response patterns to be identical for these vectors, and to reading of the 24K of data that was written in the first part of the test and comparing it to what was written. This will isolate address and data bus failures during the write cycles, and RAM failures during the read cycles. A passed test results in the compare error flag being reset, and a failure is indicated by the error flag being set. Two methods are provided within the SR2500 for reading the status of a tests error flag. Both examples will be provided at the end of the test program listing.

Note

In order to correlate all of the field patterns and algorithmic commands with the CMACRO program, the CMACRO program is listed twice, with each listing correlating to a subset of the total fields defined. There is only ONE (1) CMACRO PROGRAM, and both stimulus and response vector sequencing is controlled by that one CMACRO program. You do not create a separate CMACRO program for stimulus and response fields.

/***** Test Program Listing for RAM_TEST, Rev 2 *****/

/* Define Test Parameters */

```
TEST:DEF RAM_TEST:SIZE 1024
SYST:TEST RAM_TEST
SYST:TEST RAM_TEST
SOUR:ROSC:SOUR INT
SYST:PROG 1
SYST:FREQ 1.481000MHZ
SYST:CLOC:SOUR INT
```

Table 4-13A: RAM_TEST Program and Patterns (rev 2)

	Test Program and Pattern Definition Worksheet						
	CMACRO Program			Stimulus/Expected Response Fields			
Vectors	Label	Command	Condition	A15_08O	A15_08T	A15_08E	A15_08M
1		StartProgram		NONA	FF	NONA	XX
2		WordLoop Until	COUNt==10	NONA	FF	NONA	XX
3		CLEARError		NONA	1F	NONA	1F
4		StartLoop until	COUNt==96	INC		INC	
5		WordLoop until	COUNt==254	HOLDA		HOLDA	
6		EndLoop		HOLDA		HOLDA	
7		OUT		NONA	1F	NONA	1F
8		StartLoop until	COUNt==96	INC		INC	
9		WordLoop until	COUNt==254	HOLDA		HOLDA	
10		EndLoop		HOLDA		HOLDA	
11		EndProgram		NONA	FF	NONA	XX
Vectors	Label	Command	Condition	WRO	WRE	RDO	RDE
1		StartProgram		1	X	1	X
2		WordLoop until	COUNt==10	1	X	1	X
3		CLEARError		1	X	1	X
4		StartLoop until	COUNt==96	0	X	1	X
5		WordLoop until	COUNt==254	0	X	1	X
6		EndLoop		0	X	1	X
7		OUT		1	X	1	X
8		StartLoop until	COUNt==96	1	X	0	X
9		WordLoop until	COUNt==254	1	X	0	X
10		EndLoop		1	X	0	X
11		EndProgram		1	X	1	X

/* Define Fields */

```
FIELD:DEF A15_08O:TYPE ALGO:PIN C1P16-9
FIEL:NAME A15_08O:RAD HEX
FIELD:DEF A15_08T:TYPE OT:PIN C1P16-9
FIEL:NAME A15_08T:RAD HEX
FIELD:DEF A07_00O:TYPE ALGO:PIN C1P8-1
FIEL:NAME A07_00O:RAD HEX
FIELD:DEF A07_00T:TYPE OT:PIN C1P8-1
```

Table 4-13B: RAM_TEST Patterns (rev 2)

Test Pattern Definition Worksheet (continued)								
Stimulus/Expected Response Fields								
A07_00O	A07_00T	A07_00E	A07_00M	D07_00O	D07_00T	D07_00E	D07_00M	
NONA	FF	NONA	XX	NONA	FF	NONA	XX	
NONA	FF	NONA	XX	NONA	FF	NONA	XX	
NONA	FF	NONA	FF	NONA	55	NONA	55	
INC		INC		XOR	FF	XOR	FF	
INC		INC		XOR	FF	XOR	FF	
INC		INC		XOR	FF	XOR	FF	
NONA	FF	NONA	FF	NONA	XX	NONA	55/XX	
INC		INC		NONA	XX	XOR	FF	
INC		INC		NONA	XX	XOR	FF	
INC		INC		NONA	XX	XOR	FF	
NONA	FF	NONA	XX	NONA	FF	NONA	XX	
ALE	PSEN							
0	1							
0	1							
0	1							
1	1							
1	1							
1	1							
0	1							
1	1							
1	1							
1	1							
0	1							

```
FIEL:NAME A07_00T:RAD HEX
FIELD:DEF D07_00O:TYPE ALGO:PIN C2P24-17
FIEL:NAME D07_00O:RAD HEX
FIELD:DEF D07_00T:TYPE OT:PIN C2P24-17
FIEL:NAME D07_00T:RAD HEX
FIELD:DEF ALE:TYPE OT:PIN C3P25
FIEL:NAME ALE:RAD BIN
FIELD:DEF WRO:TYPE OT:PIN C3P28
FIEL:NAME WRO:RAD BIN
```

```
FIELD:DEF RDO:TYPE OT:PIN C3P27
FIEL:NAME RDO:RAD BIN
FIELD:DEF PSEN:TYPE OT:PIN C2P26
FIEL:NAME PSEN:RAD BIN
FIELD:DEF A15_08E:TYPE ALGE:PIN C1P16-9
FIEL:NAME A15_08E:RAD HEX
FIELD:DEF A15_08M:TYPE ED:PIN C1P16-9
FIEL:NAME A15_08M:RAD HEX
FIELD:DEF A07_00E:TYPE ALGE:PIN C1P8-1
FIEL:NAME A07_00E:RAD HEX
FIELD:DEF A07_00M:TYPE ED:PIN C1P8-1
FIEL:NAME A07_00M:RAD HEX
FIELD:DEF D07_00E:TYPE ALGE:PIN C2P24-17
FIEL:NAME D07_00E:RAD HEX
FIELD:DEF D07_00M:TYPE ED:PIN C2P24-17
FIEL:NAME D07_00M:RAD HEX
FIELD:DEF WRE:TYPE ED:PIN C3P28
FIEL:NAME WRE:RAD BIN
FIELD:DEF RDE:TYPE ED:PIN C3P27
FIEL:NAME RDE:RAD BIN
```

/* Define Stimulus Data Formats and Response Sample Formats */

```
STIM:COND:OFOR:FIEL A07_00O;MODE RI,0.000000 E-9,80.000000 E-9
STIM:COND:OFOR:FIEL D07_00O;MODE RI,200.000000 E-9,465.000000 E-9
STIM:COND:OFOR:FIEL ALE;MODE RZ,0.000000 E-9,30.000000 E-9
STIM:COND:OFOR:FIEL WRO;MODE RONE,230.000000 E-9,400.000000 E-9
STIM:COND:OFOR:FIEL RDO;MODE RONE,230.000000 E-9,400.000000 E-9
REC:COND:SAMP:FIEL D07_00E;MODE EDGE,550.000000 E-9
REC:COND:SAMP:FIEL WRE;MODE EDGE,550.000000 E-9
REC:COND:SAMP:FIEL RDE;MODE EDGE,550.000000 E-9
REC:COND:SAMP:FIEL A15_08E;MODE EDGE,30.000000 E-9
REC:COND:SAMP:FIEL A07_00E;MODE EDGE,30.000000 E-9
```

/* Define CMACRO Program */

```
STIM:VEC 1;COUN 1;CMAC:DEF (SP(OUT))
STIM:VEC 2;COUN 1;CMAC:DEF (WL(OUT(COUN == 10)))
STIM:VEC 3;COUN 1;CMAC:DEF (CLEARE(OUT))
STIM:VEC 4;COUN 1;CMAC:DEF (SL(OUT(COUN == 96)))
STIM:VEC 5;COUN 1;CMAC:DEF (WL(OUT(COUN == 254)))
STIM:VEC 6;COUN 1;CMAC:DEF (EL(OUT))
STIM:VEC 7;COUN 1;CMAC:DEF (OUT(OUT))
STIM:VEC 8;COUN 1;CMAC:DEF (SL(OUT(COUN == 96)))
STIM:VEC 9;COUN 1;CMAC:DEF (WL(OUT(COUN == 254)))
STIM:VEC 10;COUN 1;CMAC:DEF (EL(OUT))
STIM:VEC 11;COUN 1;CMAC:DEF (EP(OUT))
```

/* Load Stimulus/Response Data Patterns and Algorithmic Commands */

```
STIM:VEC 1;COUN 11;AMAC:FIEL A15_08O;PATT  NONA, NONA, NONA, INC, HOLDA, HOLDA, NONA, INC,
HOLDA, HOLDA, NONA
STIM:VEC 1;COUN 11;DATA:FIEL A15_08T;PATT  FF, FF, 1F, 00, 00, 00, 1F, 00, 00, 00, FF
REC:VEC 1;COUN 11;AMAC:FIEL A15_08E;PATT  NONA, NONA, NONA, INC, HOLDA, HOLDA, NONA, INC,
HOLDA, HOLDA, NONA
REC:VEC 1;COUN 11;DATA:FIEL A15_08M;PATT  XX, XX, 1F, 00, 00, 00, 1F, 00, 00, 00, XX

STIM:VEC 1;COUN 11;AMAC:FIEL A07_00O;PATT  NONA, NONA, NONA, INC, INC, INC, NONA, INC, INC,
INC, NONA
STIM:VEC 1;COUN 11;DATA:FIEL A07_00T;PATT  FF, FF, FF, 00, 00, 00, FF, 00, 00, 00, FF
REC:VEC 1;COUN 11;AMAC:FIEL A07_00E;PATT  NONA, NONA, NONA, INC, INC, INC, NONA, INC, INC,
INC, NONA
REC:VEC 1;COUN 11;DATA:FIEL A07_00M;PATT  XX, XX, FF, 00, 00, 00, FF, 00, 00, 00, XX

STIM:VEC 1;COUN 11;AMAC:FIEL D07_00O;PATT  NONA, NONA, NONA, XOR, XOR, XOR, NONA, NONA,
NONA, NONA, NONA
STIM:VEC 1;COUN 11;DATA:FIEL D07_00T;PATT  FF, FF, 55, FF, FF, FF, XX, XX, XX, XX, FF
REC:VEC 1;COUN 11;AMAC:FIEL D07_00E;PATT  NONA, NONA, NONA, XOR, XOR, XOR, NONA, XOR, XOR,
XOR, NONA
REC:VEC 1;COUN 11;DATA:FIEL D07_00M;PATT  XX, XX, 55, FF, FF, FF, XX, FF, FF, FF, XX
REC:VEC 7;COUN 1;DATA:FIEL D07_00E;PATT  55
```

Note

At vector 7, the data bus is being tristated for the pending read cycles. The response ALU must be initialized in anticipation of the alternating 0xAA/0x55 pattern being read from the UUT, yet all input pins for the field should be ignored at vector 7. In all other fields this is not a concern as the SR2500 is generating the pattern on the input pins, so the expected pattern can be anticipated. However, the data bus is not being driven by the SR2500 at vector 7. Since the data bus is floating, and their state is unknown, the data bus inputs should be masked. Notice that field D07_00M, vector 7, is masked (XX), yet the pattern of 0x55 is used to initialize the response ALU two program lines later (rec:vec 7;coun 1;data:fiel D07_00E;patt 55). An ALGE (and ALGO) type field is a combined field of algorithmic commands and expect (output) memory. To load pattern memory instead of algorithmic commands, use the word "DATA" in place of "AMAC".

```
STIM:VEC 1;COUN 11;DATA:FIEL ALE;PATT  0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0
STIM:VEC 1;COUN 11;DATA:FIEL WRO;FILL:TYPE REP;PATT 1;EXEC
STIM:VEC 4;COUN 3;DATA:FIEL WRO;FILL:TYPE REP;PATT 0;EXEC
REC:VEC 1;COUN 11;DATA:FIEL WRE;PATT X, X, X, X, X, X, X, X, X, X, X
STIM:VEC 1;COUN 11;DATA:FIEL RDO;FILL:TYPE REP;PATT 1;EXEC
STIM:VEC 8;COUN 3;DATA:FIEL RDO;FILL:TYPE REP;PATT 0;EXEC
REC:VEC 1;COUN 11;DATA:FIEL RDE;PATT X, X, X, X, X, X, X, X, X, X, X
STIM:VEC 1;COUN 11;DATA:FIEL PSEN;FILL:TYPE REP;PATT 1;EXEC
```

/* Define Run-Time Parameters */

```
STIM:ARMD:FIELD A15_08T;PATT #hFF
STIM:ARMD:FIELD A07_00T;PATT #hFF
STIM:ARMD:FIELD D07_00T;PATT #hFF
STIM:ARMD:FIELD ALE;PATT #h0
STIM:ARMD:FIELD WRO;PATT #h1
STIM:ARMD:FIELD RDO;PATT #h1
STIM:ARMD:FIELD PSEN;PATT #h1
```

/* Initiate the Test and Trigger */

```
INIT
*TRG
```

/*** Query the pass/fail state using either of the following *****/**

/* Query the status of the test (includes state of error flag)*/

```
TEST:NAME RAM_TEST:STAT?
```

/* Query the state of the Error Flag only */

```
REC:DATA:ERR?
```

Recording UUT Responses

Using the error flag is one method of determining pass/fail of a device being tested. However, if a failure occurs, the error flag can not indicate the cause of the failure. In this case, it is necessary to record data from the UUT while the test is being performed. Evaluation of the captured data will lead to the failed device.

The SR2500 has great flexibility not only over what information is recorded from the UUT, but in how that information is recorded. This control is provided via the TRACE subsystem. The trace subsystem is, for all practical purposes, an independent logic analyzer packaged with the SR2500 and clocked from a common clock. However, it maintains independent control of what information is stored to the record memory, and under what conditions that information is stored. The trace subsystem also control under what conditions CRC calculations are performed.

This third version of the RAM_TEST program will incorporate the *Trace Functions* to record data from the UUT and enable CRC calculations. The recorded data and CRC signatures may be queried after the test is complete for evaluation, or for comparison to known good signatures. The trace subsystem is used like you would a *Logic Analyzer*. If information from multiple sources must be correlated, then all of that information must be provided to input pins in the SR2500 system and recorded.

In this example, the address and data busses must be recorded in order to pinpoint at what address the RAM failed, if a failure is detected. So, even though an address bus is usually an output only bus, this application will connect the stimulus address pins to response address pins. In order to determine if the failure occurred during the read or write cycle, both the WR and RD control signals are also connected to input pins.

Table 4-14: RAM_TEST Response and Record Fields (rev 3)

Field Definition Parameters			
Field Name: ADDR	Field Type: REC	Field Radix: HEX	Pin List: C1P16-1
Arm Pattern: N/A	Format: N/C	Assert: N/A	Pulse Width: N/A
Field Name: ADDRE	Field Type: ED	Field Radix: HEX	Pin List: C1P16-1
Arm Pattern: N/A	Format: N/C	Assert: N/A	Pulse Width: N/A
Field Name: DATA	Field Type: REC	Field Radix: HEX	Pin List: C2P24-17
Arm Pattern: N/A	Format: N/C	Assert: N/A	Pulse Width: N/A
Field Name: DATAE	Field Type: ED	Field Radix: HEX	Pin List: C2P24-17
Arm Pattern: N/A	Format: N/C	Assert: N/A	Pulse Width: N/A
Field Name: WR_RD	Field Type: REC	Field Radix: BIN	Pin List: C3P28-27
Arm Pattern: N/A	Format: N/C	Assert: N/A	Pulse Width: N/A
Field Name: WR_RDE	Field Type: ED	Field Radix: BIN	Pin List: C3P28-27
Arm Pattern: N/A	Format: N/C	Assert: N/A	Pulse Width: N/A

Six new fields are added to the test, a record and expect address field, a record and expect data field, and a record and expect write/read field. The *Record Fields* are where the data returned by the UUT will be stored, or, more precisely, these fields provide access to the recorded data. Refer to table 4-14. The expect fields provide a more convenient format for defining the address, data and write/read control trigger patterns. Field ADDR is a *Super-set* of the A15_08 and A07_00 fields. By combining both address fields into one, the RAM address may be viewed as one address, not an upper and lower address. The same is true for the WR_RD field. By combining both the write and the read control signals into one field, it is easy to determine the direction of the data flow during the cycle.

The CMACRO program, stimulus patterns and response patterns do not need to be modified. However, the trace parameters do need to be defined. To assist in the definition of the trace parameters, it is first necessary to plan what information to record, and under what conditions. The SR2500 will start recording on the first write cycle (address 0x2000) of the test, and then record 511 additional samples, then wait for the first read cycle (also address 0x2000) of the test and record the read cycle and the next 511 read cycles. This allows you to verify the write and read process is working.

The first step is to determine if Qualifier Triggers will be used. Qualifier triggers are logic record trigger patterns which may be used by the trace subsystem, or by a CMACRO program. Qualifier triggers may be combined together in Qualifier Combinations (QCOM) for use as multi-condition triggers. Example 1 will use 2 qualifier triggers and 2 QCOM's. Qualifier trigger 1 will be looking for address 0x2000 and WR active low. Qualifier trigger 2 will be looking for address 0x2000 and RD active low. Refer to table 4-15.

Table 4-15: Qualifier Trigger, QCOM and Trace Sequence Definitions

	Qualifier Trigger Definitions												
Fields ->	ADDRE	DATAE	WR_RDE										
1	0x2000	XX	01										
2	0x2000	XX	10										
3													
4													
5													
6													
7													
8													

Qualifier Combinations							
1	2	3	4	5	6	7	8
●							
	●						

	Trace Sequences							
	Record Filter	Record On	CRC Sample On	Advance On	Advance Count	Jump On	Jump To	Stop Test
1	DATA	QCOM1	NEVER	QCOM1	1	NEVER	1	NO
2	DATA	ALWAYS	NEVER	CLOCK	511	NEVER	1	NO
3	DATA	QCOM2	NEVER	QCOM2	1	NEVER	1	NO
4	DATA	ALWAYS	NEVER	COLCK	511	NEVER	1	NO
5	DATA	NEVER	NEVER	NEVER	1	NEVER	1	NO
6								
7								
8								
9								
10								
11								
12								
13								
14								
15								
16								

Next, the *Qualifier Triggers* are combined into QCOM's, also in table 4-15. Up to all eight of the qualifier trigger patterns may be used in a single QCOM, or they may be used individually, as in this example. QCOM 1 consists of only qualifier trigger 1, and QCOM 2 consists of only qualifier trigger 2.

The last step is to define the *Trace Sequence Controls* and *CRC Controls.* The first parameter, the Record Filter, defines what to put into the record memory, data or errors. The second parameter, Record On, defines when to record data. Data may be recorded Never, Always, on Real-Time Compare True, on Real-Time Compare False, or on one of the QCOM's. The CRC Sample On parameter defines when the input data will be used in a CRC calculation. The allowable parameters are the same as for the Record On parameter. Advance On defines when you advance from the current sequence number, to the next. Advance On works with the Advance Count, which defines how many times the Advance On condition must be met in order to advance. The Advance On parameters are the same as the CRC Sample On and the Record On. The Jump On parameter is similar to the Advance On parameter in that it also defines a condition in which the current trace sequence will be exited. In this case, though, instead of advancing to the next sequence, you jump to the sequence number defined by the Jump To parameter. There are 16 trace sequence levels. Finally, the Stop parameters tells the SR2500 to halt test execution when that trace sequence level is reached. As the stop flag is polled by the microprocessor, the polling overhead and overhead associated with halting the test dictate that the halt will not be immediate.

In this example, trace sequence 1 is defined to record data, only when QCOM1 matches, never calculate CRC, advance to sequence 2 when QCOM1 matches 1 time, never jump and do not stop. Trace sequence 2 is defined to record data, record always, never calculate CRC, advance to sequence 3 after 511 clock cycles, never jump and do not stop. Trace sequence 3 is defined to record data, only when QCOM2 matches, never calculate CRC, advance to sequence 4 when QCOM2 matches 1 time, never jump and do not stop. Trace sequence 4 is defined to record data, record always, never calculate CRC, advance to sequence 5 after 511 clock cycles, never jump and do not stop. Trace sequence 5 is defined to never record, never sample CRC, never advance and never jump to a new sequence, effectively halting the record process.

The following listing should be added to the program listing for RAM_TEST version 2.

/* New Field Definitions */

```
FIELD:DEF ADDR:TYPE REC:PIN C1P16-1
FIEL:NAME ADDR:RAD HEX
FIELD:DEF ADDRE:TYPE ED:PIN C1P16-1
FIEL:NAME ADDRE:RAD HEX
FIELD:DEF DATA:TYPE REC:PIN C2P24-17
FIEL:NAME DATA:RAD HEX
FIELD:DEF DATAE:TYPE ED:PIN C2P24-17
FIEL:NAME DATAE:RAD HEX
FIELD:DEF WR_RD:TYPE REC:PIN C3P28-27
FIEL:NAME WR_RD:RAD BIN
FIELD:DEF WR_RDE:TYPE ED:PIN C3P28-27
FIEL:NAME WR_RDE:RAD BIN
```

/* Define Qualifier Trigger Patterns */

```
REC:TRAC:QUAL 1:FIEL ADDRE;PATT  #H2000
REC:TRAC:QUAL 1:FIEL DATAE;PATT  #HXX
REC:TRAC:QUAL 1:FIEL WR_RDE;PATT  #H01
REC:TRAC:QUAL 2:FIEL ADDRE;PATT  #H2000
REC:TRAC:QUAL 2:FIEL DATAE;PATT  #HXX
REC:TRAC:QUAL 2:FIEL WR_RDE;PATT  #H10
```

/* Define Qualifier Combinations */

```
REC:TRAC:QCOM1  1
REC:TRAC:QCOM2  2
```

/* Define Trace Sequence Parameters */

```
REC:TRAC:SEQ 1:DEF:FIL DAT:REC QCOM1
REC:TRAC:SEQ 1:DEF:CRC:CAL NEV
REC:TRAC:SEQ 1:DEF:ADVS:ON QCOM1:COUN 1
REC:TRAC:SEQ 1:DEF:JUMP 1:ON NEV
REC:TRAC:SEQ 2:DEF:FIL DAT:REC ALW
REC:TRAC:SEQ 2:DEF:CRC:CAL NEV
REC:TRAC:SEQ 2:DEF:ADVS:ON CLOC:COUN 511
REC:TRAC:SEQ 2:DEF:JUMP 1:ON NEV
REC:TRAC:SEQ 3:DEF:FIL DAT:REC QCOM2
REC:TRAC:SEQ 3:DEF:CRC:CAL NEV
REC:TRAC:SEQ 3:DEF:ADVS:ON QCOM2:COUN 1
REC:TRAC:SEQ 3:DEF:JUMP 1:ON NEV
REC:TRAC:SEQ 4:DEF:FIL DAT:REC ALW
REC:TRAC:SEQ 4:DEF:CRC:CAL NEV
REC:TRAC:SEQ 4:DEF:ADVS:ON CLOC:COUN 511
REC:TRAC:SEQ 4:DEF:JUMP 1:ON NEV
REC:TRAC:SEQ 5:DEF:FIL DAT:REC NEV
REC:TRAC:SEQ 5:DEF:CRC:CAL NEV
REC:TRAC:SEQ 5:DEF:ADVS:ON NEV:COUN 1
REC:TRAC:SEQ 5:DEF:JUMP 1:ON NEV
```

/* Initiate Test and Trigger */

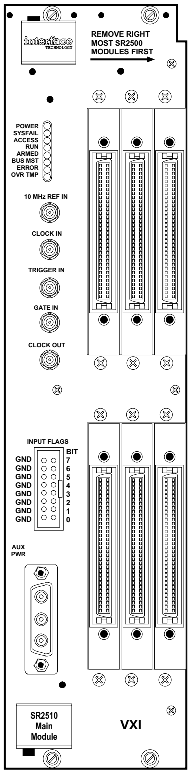
```
INIT
*TRG
```

Once the test has completed, the address data and read/write control values may be queried using the REC:DATA:PATT? query command. Trace parameters may be modified to record on errors, capture errors, enable CRC calculation, or whatever function is appropriate to the diagnostics requirements.

Published by
Interface Technology, Inc.
300 S. Lemon Creek Drive
Walnut, CA 91789

User's Manual

SR2510 Main Module



From The Performance Leader In VXI Digital Testing ...

Rev. 05 Apr 1998
Chg. 09 Sep 2003

SR2510 User's Manual

Record of Changes

Change No.	Date of Change	Title or Brief Description	Entered By
Rev 05	Apr 98	Reformat	Factory
Change 1	Mar 00	Revised external power supply info (pg 3-3); added pinout data for differential TTL (pg 3-18, 3-19).	Factory
Change 2	Mar 00	Added coverage for differential TTL (pg 2-24, 2-25).	Factory
Change 3	May 00	Reformat pages 1-8 thru 1-10 (specifications).	Factory
Change 4	Jun 00	Added LVDS I/O, pgs 1-6, 1-10, 1-11, 2-25, 2-26, 3-18, 3-19	Factory
Change 5	Oct 00	Corrected connector orientaion in Figs 3-9 thru 3-18 and added note explaining relationship of pinout views to instrument illustrations. Added power sequencing note to page 3-3.	Factory
Change 6	Sep 01	Reformatted specifications page, pg 1-8. Deleted pages 1-9 thru 1-12.	Factory
Change 7	May 03	pg. 2-25 ... 1st para., 2nd and 3rd lines ... changed "...-4V to +7V" to "... -3V to +7V"; changed "... -4 to +5.5V" to "... -2.9 to +5.5V."	Factory
Change 8	May 03	Corrected connector pinouts in Fig 3-16 (pins B01 and B02)	Factory
Change 9	Oct 03	Corrected I/O Characteristics table on pg. 1-8. Corrected Figures 2-7 and 2-8; added Fig 2-13. Added pgs 2-27 (3.3 V I/O) and 2-28 (blank). Updated Fig 3-10 and 3-11 to include 3.3 V I/O.	Factory

Contents

Chapter 1
General Information

About This Manual 1-1
  Arrangement of Contentsl 1-1
  Applicability 1-1
  Supersedure Notice 1-1
Equipment Description 1-1
  Timing/Control Board 1-1
  Features 1-2
  I/O Boards 1-2
  VXI Bus Interface 1-2
Real Time Digital Testing 1-3
Power Macro Commands
  Control Test Execution & Data Analysis 1-3
High Performance Response Logic Analysis 1-4
Controls and Indicators 1-5
Timing/Control Connectors 1-6
Interconnection With I/O Modules 1-7
Specifications 1-8

Chapter 2
Theory of Operation

Block Diagram 2-1
Timing/Control Board 2-1
  Dynamic RAM 2-2
  EPROM 2-2
  Flash RAM 2-2
  VXI Interface 2-2
  VME A32 Memory 2-2
  Control Processor and Memory 2-2
  PLO Interface 2-2
  Input Buffer Thresholds 2-2
IT9010M VXI Interface 2-4
VME A32 Memory 2-4
Phase Locked Oscillator (PLO) 2-5
  Frequency Resolution 2-5
  Frequency Accuracy 2-6
Control Gate Array 2-6
  Control Processor 2-6
Record State Machine 2-13
  Filter 2-13
  Record 2-13
  CRC 2-14
  Advance Sequence 2-14
  Jump To Sequence 2-15
  Stop 2-15
I/O Board 2-15

Contents (continued)

Stimulus and Response Memory 2-16
Record Memory 2-16
Delayed Clock Generators 2-17
Stimulus Gate Arrays 2-17
  Algorithmic Stimulus Pattern Generator 2-17
  Stimulus Output Pin Formatter 2-18
Response Gate Arrays
  Expected Response Pattern Generator 2-19
  Response Input Formatter 2-20
  Response Comparator 2-20
  Input Qualifier 2-20
  Record Control 2-20
  CRC Logic 2-20
Algorithmic Commands 2-22
  NONAlgorithmic 2-22
  INCrement 2-22
  DECrement 2-22
  XOR 2-23
  SLEFTZero 2-23
  SLEFTOne 2-23
  SLEFTComplement 2-23
  RLEFT 2-23
  SRIGHTZero 2-23
Driver/Receiver Board 2-24
  TTL Driver/Receiver Logic 2-24
  CMOS Driver/Receiver Logic 2-24
  Differential ECL Driver/Receiver Logic 2-25
  Programmable Driver/Receiver Logic 2-25

Chapter 3 Installation
Scope of Chapter 3-1
Unpacking and Inspection 3-1
Installation 3-1
  Logical Addressing 3-1
Slot Dependency 3-2
Backplane Jumpers 3-2
  SR2510 Main Module 3-2
  SR2520 Expansion Module 3-2
5 Vdc External Power Requirements 3-2
Option Switch 3-4
Option Switch Settings 3-4
  Host Controller Selection 3-4
  Power-On Diagnostic Test (fast/slow) 3-5
  Opterating Protocol, Bus Master/Release-on-Request 3-6

Contents (continued)

Test Post-Processor Speed (fast/slow) 3-6
Boot Mode (EPROM/Flash ROM) 3-7
Main and Expansion Module Interconnect 3-8
Installing I/O Boards 3-10
Required Equipment
Procedure ... Install I/O Bd No.2 3-10
Procedure ... Install I/O Bd No.3 3-11

List of Figures

Fig. 1-1, SR2510 Module With Three I/O Boards
and Six Driver/Receiver Boards 1-3
Fig. 1-2, VXI Chassis Showing SR2510 Main Module
and SR2520 Expansion I/O Module 1-4
Fig. 1-3, LED Indicators 1-5
Fig. 1-4, Connectors 1-6
Fig. 1-5, SR2510 and SR2520 Interconnection, Top View 1-7
Fig. 2-1, SR2510 Block Diagram, Major Components 2-1
Fig. 2-2, SR2510 Timing/Control Board Block Diagram 2-3
Fig.2-3, Control Processor Gate Array Block Diagram 2-8
Fig. 2-4, SR2510 I/O Board Block Diagram 2-16
Fig. 2-5, Stimulus Gate Array Block Diagram 2-18
Fig. 2-6, Response Gate Array 2-19
Fig. 2-7, TTL Single Ended Driver/Receiver (16 per D/R Board)........ 2-26
Fig. 2-8, CMOS Single Ended Driver/Receiver (16 per D/R Board) 2-26
Fig. 2-9, Differential ECL Driver/Receiver (16 per D/R Board) 2-26
Fig. 2-10, Programmable Driver/Receiver (32 per D/R Board) 2-17
Fig. 3-1, Address Switches Set to Logical Address 12(hex) 3-1
Fig. 3-2, Connection of External 5 Vdc Operating Power 3-3
Fig. 3-3, Option Switch SW3 3-5
Fig. 3-4, Interconnect Between SR2510 and SR2520, Top View 3-8
Fig. 3-5, SR2510 and SR2520 Interconnect Connectors 3-9
Fig. 3-6, Cover Screws 3-12
Fig. 3-7, I/O Board Mounting Hardware 3-13
Fig. 3-8a, Buildup of Standoff Spacers at Positions 1-3
for Configurations of One, Two, and Three I/O Boards 3-14
Fig. 3-8b, Buildup of Standoff Spacers at Position 4
for Configurations of One, Two, and Three I/O Boards 3-14
Fig. 3-9, SR2510 Input Flags Connector 3-15
Fig. 3-10, SR2510 Signal Connector Pinouts,
TTL or CMOS, Ch. 00-15 3-16
Fig. 3-11, SR2510 Signal Connector Pinouts,
TTL or CMOS, Ch. 16-31 3-17

Contents (continued)

Fig. 3-12, SR2510 Signal Connector Pinouts,
Differential ECL, Ch. 00-15 3-18
Fig. 3-13, SR2510 Signal Connector Pinouts,
Differential ECL, Ch. 16-31 3-19
Fig. 3-14, SR2510 Signal Connector Pinouts,
Variable Voltage, Ch. 00-31 3-20
Fig. 3-15, SR2510 Rail Voltage Connector Pinouts 3-21
Fig. 3-16, SR2510 Auxiliary Power Connector Pinouts 3-22

CHAPTER 1

General Information

About This Manual

This manual provides installation and operation information for the Interface Technology SR2510 Timing / Control / I/O Module. Information contained herein is intended for use by technical personnel involved in the actual installation and operation of the subject instrument.

Arrangement of Contents

Information contained in this manual is arranged in five chapters, as follows:

- Chapter 1 General Information
- Chapter 2 Theory of Operation
- Chapter 3 Installation

Applicability

The information contained in this manual covers a single equipment configuration designated ***SR2510 Timing / Control / I/O Module***. Differences, if any, between this equipment and the actual equipment supplied are covered by Difference Data included at the front of this manual.

Supersedure Notice

This manual supersedes SR2500 User's Manual, Rev.04 and all previous issues of this publication.

Equipment Description

See Fig.1-1. The SR2510 serves both as the overall SR2500 system timer/ controller, and as the stimulus/response input/output interface with the UUT (Unit Under Test). The major components of the SR2510 include a Timing/Control board, from one, two, or three I/O boards, and up to six Driver/Receiver boards (2 per I/O board). Other components include boards for timing distribution, power distribution and interface logic.

Timing/Control Board.

The SR2510 Timing/Control board provides clocking and test sequence control functions for all I/O boards, both those within the SR2510 module itself, and for any and all additional I/O boards contained in any expansion SR2520 I/O Modules used in the same SR2500 subsystem. The Timing/ Control Board parses and interprets VXI word-serial commands from the Slot-0 Controller and provides overall system setup and test monitoring. It also provides real-time control over the test pattern sequencing. With its built-in control processor, the SR2510 is capable of providing sequential

Features

- **DC to 25 MHz Data Rates.**
- **64K Vector Depth, 256K Optional.**
- **Stimulus / Response / Real-Time Compare / Record.**
- **32, 64, or 96 Inputs and 32, 64, or 96 Outputs in a Dual-Slot VXI Module.**
- **Expandable to 576 Inputs and 576 Outputs in a Single VXI Chassis.**
- **RAM-Backed and Algorithmic Pattern Generation.**
- **Multi-Level Triggering and Advanced Logic Analysis.**
- **Data Formatting with Programmable Edge Placement.**
- **Message-Based SCPI Commands and Software Drivers for Easy Test Program Development.**
- **A32 / D32 Binary Transfer for High-Speed Test Program Download.**
- **Conditional Pattern Looping and Branching for Real time Test Sequence Control.**
- **Multiple Logic Families Supported Through Plug-In Modules.**
- **Software Compatible With Interface Technology's SR5000.**
- **Optional Guided Probe.**

or nested program looping, plus conditional or unconditional jumps and subroutines. Overall test timing is provided by an internal, programmable, 200 Hz to 25 MHz frequency synthesized clock source or by external inputs for clocks, gates, test inputs and triggers.

The Timing/Control board has a dual processor architecture that is optimized for digital testing. The 25 MHz 68030 system processor provides the VXIbus message-based interface to the Slot-0 Controller. The control processor is the real-time digital test engine controlling the conditional test branching, looping, sequencing and logic analysis trigger evaluation.

I/O Boards

The I/O boards within the SR2510 are register-based companions to the message-based Timing/Control board. Each I/O board provides 32 I/O channels. The SR2510 can accommodate up to three I/O boards (up to 96 channels) and up to five SR2520s, each containing up to three I/O boards (96 channels) can be included in a single SR2500 subsystem. Each I/O channel generates digital stimulus patterns, provides real-time comparison capabilities on the response inputs, and contains logic analyzer type triggering and data recording functions, all at speeds up to 25 MHz.

Each stimulus pin contains output and tristate memories, allowing bidirectional signal paths. The response pin provides *expected response* and *mask* ("don't care") memories, which generate the expected input pattern used for the real-time comparison. The logic analyzer triggering and recording subsystem allows the recording of either the actual input pattern or the results of the real-time comparison of the expected response pattern and the input pattern (error data). Either may be saved and then later retrieved from the record memory, in much the same way you would use a logic analyzer.

The SR2510 is designed to operate with any VXI compatible slot-0 controller that supports the word serial protocol. The command set that controls test setup and execution is based on the SCPI-syntax command set.

VXI Bus Interface

Based on the IT9010M industry standard VXI bus interface chip, the SR2510 meets the requirements of VXI Bus Specification Versions 1.3 and 1.4. The SR2510 VXI bus interface receives message-based commands from the Slot-0 Controller, then becomes the VXI Bus Master to pass test parameters and data to the SR2520 I/O modules. The System Processor provides the command power for the SCPI-syntax word serial command structure.

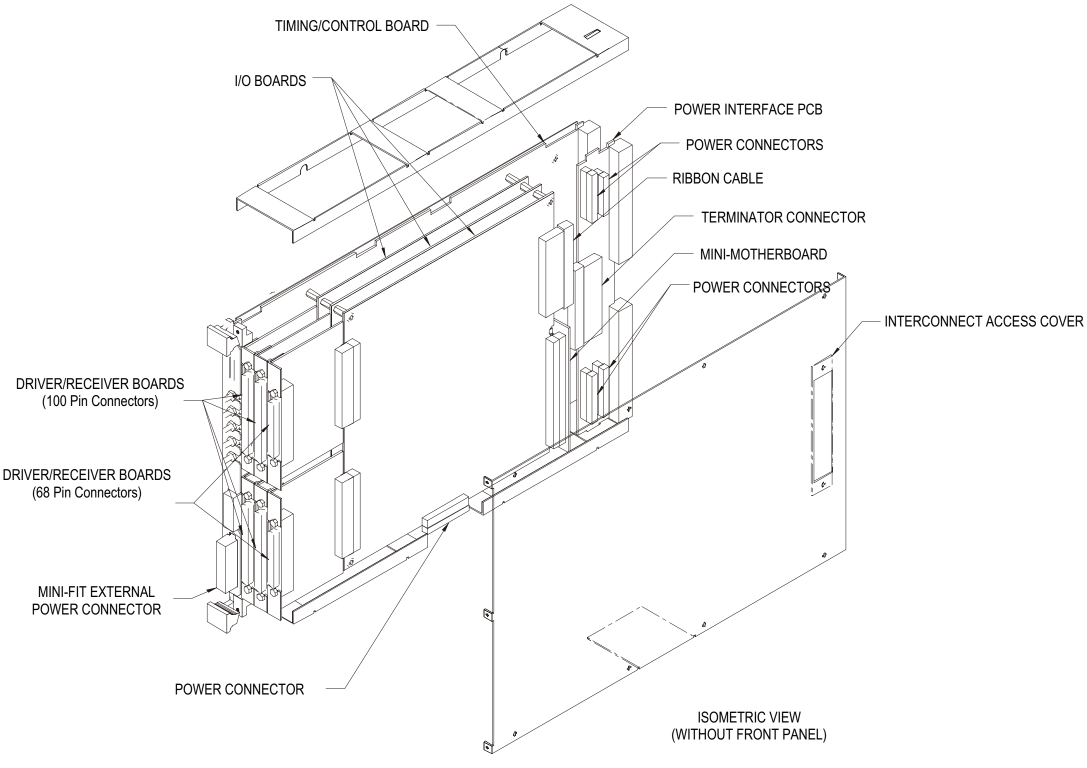


Figure 1-1.
SR2510 Module With Three I/O Boards and Six Driver/Receiver Boards.

Real Time Digital Testing

The 25 MHz Control Processor provides real time control of the test pattern sequence by controlling nested looping and conditional branching. This capability allows the SR2500 to generate stimulus patterns to the UUT, analyze the UUT response patterns, and determine the next test pattern based on test conditions such as expected response pass/fail, loop count, external input flags, response trigger qualifiers, etc.

Powerful Macro Commands Control Test Execution and Data Analysis

Stimulus pattern and response compare sequencing is controlled through a Test Program Macro Command language. The test program language contains over 100 macro command combinations to control the test sequence. All this digital testing capability is performed at full test speed and in real time, therefore, off-loading your Slot-0 Controller from extensive response data analysis.

High Performance Response Logic Analysis

The Record capability of the SR2500 is similar to that of an advanced logic analyzer. For simple logic analysis and recording, ***Trace Macro Commands*** allow you to quickly and easily program pre-trigger, center-trigger, and post-trigger conditions.

The ***Advanced Trace Macro Commands*** provide a higher level of logic analysis performance by providing 16 Trigger Sequences. Each Trigger Sequence can trigger on any combination of up to 8 Response Qualifier Trigger Words. When trigger conditions are met, the trigger action can determine whether UUT response data or UUT compare error data will be recorded to memory. Refer to Chapter 3 *"Programming"* for discussion of these commands.

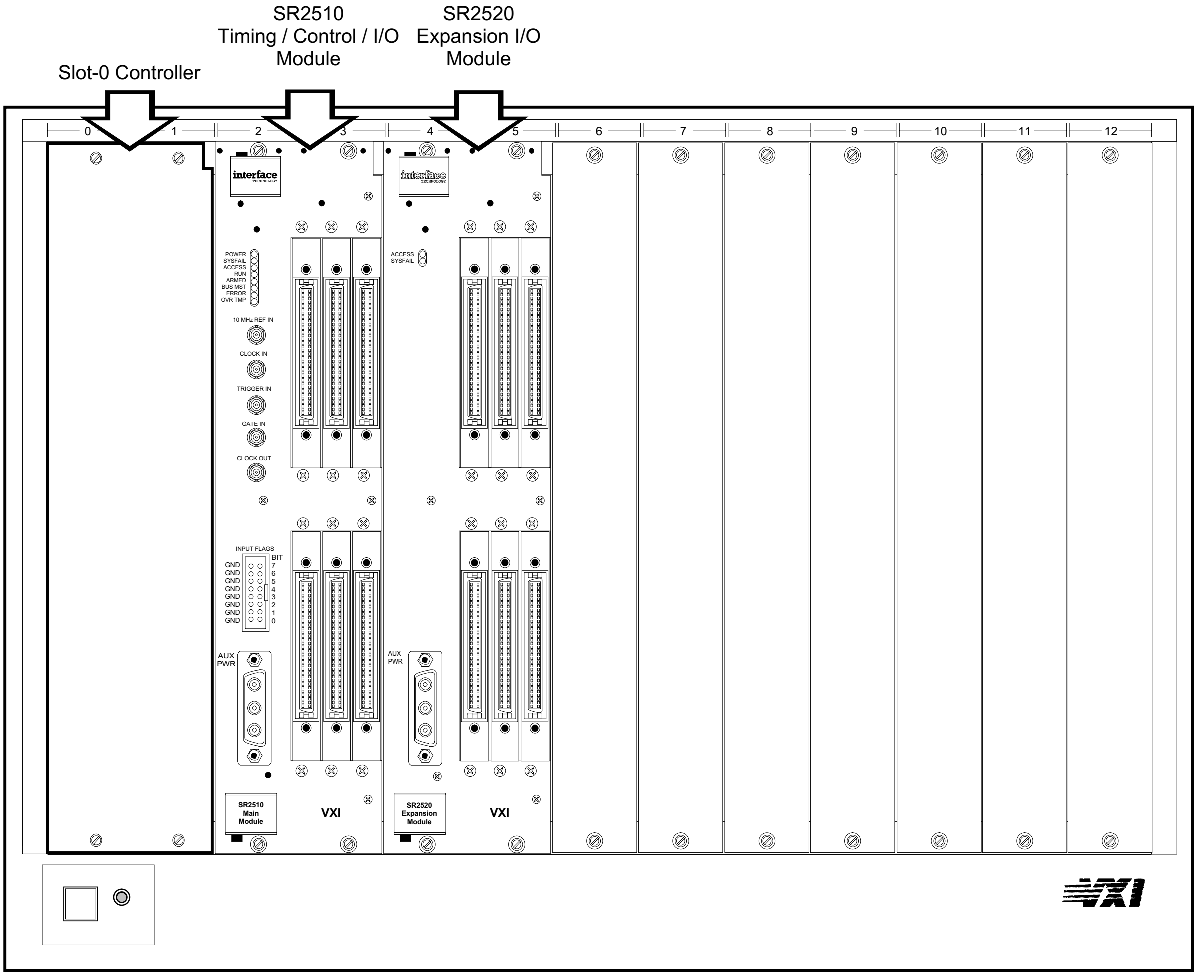


Figure 1-2.
VXI Chassis Showing SR2510 Main Module and SR2520 Expansion Module.

Controls and Indicators

See Fig. 1-3. All the connectors and LEDs for the SR2510 are located on the module front panel.

LEDs

There are eight LEDs located at the top of the SR2510 front panel.

- **POWER** (Green) - The POWER LED is connected to the system reset signal and is lit during normal operation. The LED will turn off during a system reset or if the +5V power supply drops below +4.7V.
- **SYSFAIL** (Red) - The SYSFAIL LED is off during normal operation. During the power-up sequence the LED is lit until the internal self-test passes, or remains lit if the self-test fails. If the self-test fails, error code information stored in the Data Low Register indicates the origin of the self-test failure (See Appendix A of this manual).
- **ACCESS** (Yellow) - Illuminates briefly each time the Slot-0 Controller communicates with the SR2510 Module.
- **RUN** (Green) - The SR2500 system is RUNNING.
- **ARMED** (Green) - The SR2500 system is armed and waiting for a system trigger.
- **BUS MAST** (Yellow) - SR2510 Module is operating in Bus Master mode.
- **ERROR** (Red) - A programming error has occurred. The error status can be queried by sending the "SYSTem:ERRor?" command.
  OVRTMP (Red) - Illuiminates if internal temperature of module reaches the point where operation may become unstable and/or component failure is likely to occur.

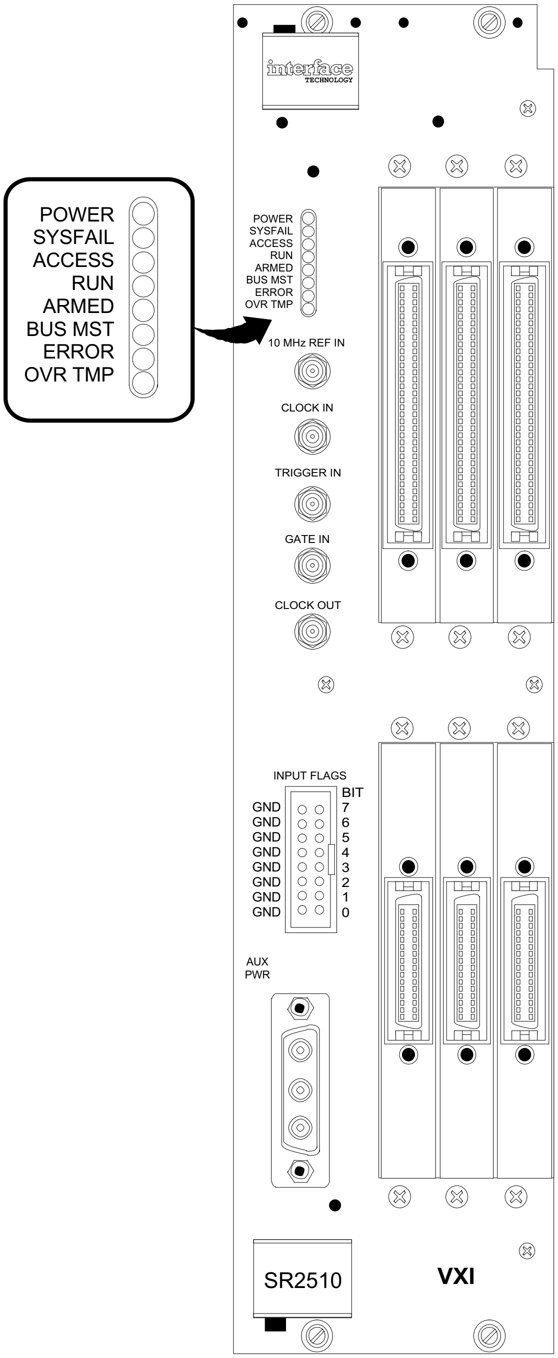


Figure 1-3.
LED Indicators.

Timing/Control Connectors

There are four Input Control Signals, one Output Control Signal and 8 TTL Input Flags on a 16 pin connector, plus an auxiliary power input connector, all located directly below the LEDs.

- **10 MHz REF IN -** Alternate reference source for the internal PLO.
- **CLOCK IN -** An external clock to be used as an alternate test timing source.
- **TRIGGER IN -** An external trigger input used to trigger a test.
- **GATE IN -** An external signal used to enable and disable the system clock.
- **CLOCK OUT -** An output signal providing the undivided system clock.
- **INPUT FLAGS -** The 16-pin Input Flag connector contains eight signal inputs and ground returns, numbered 0-7. These signals are available to control Conditional branching of the CMACRO program sequence and subroutines.
- **AUX PWR -** Connection point for external power when SR2510 is configured with more than one 32-channel I/O Board.

I/O Connectors

Each I/O Board (up to 3) has two I/O connectors. The number of pins, the pin arrangement, and the pin function varies, depending on the type of logic for which the I/O Board is configured (TTL, ECL, CMOS, LVDS or Variable Voltage).

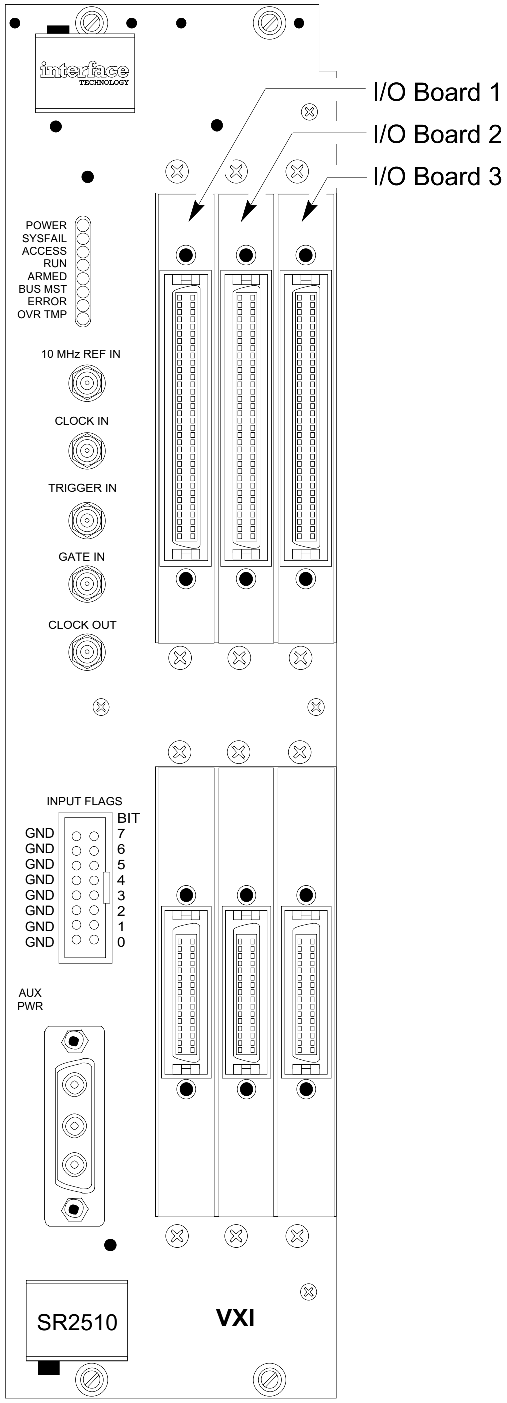


Figure 1-4.
Connectors.

Interconnection With I/O Modules

All interconnections between the SR2510 Timing / Control / I/O Module and SR2520 Expansion I/O Modules are made by means of the VXI backplane, and by a special connector at the side of the module. Interconnections are completed whenever Expansion Modules are added to the system. No additional cabling between modules is required. The second, and subsequent, SR2520 modules are connected in a similar manner. Refer to Chapter 5 *"Installation"* for additional interconnection information.

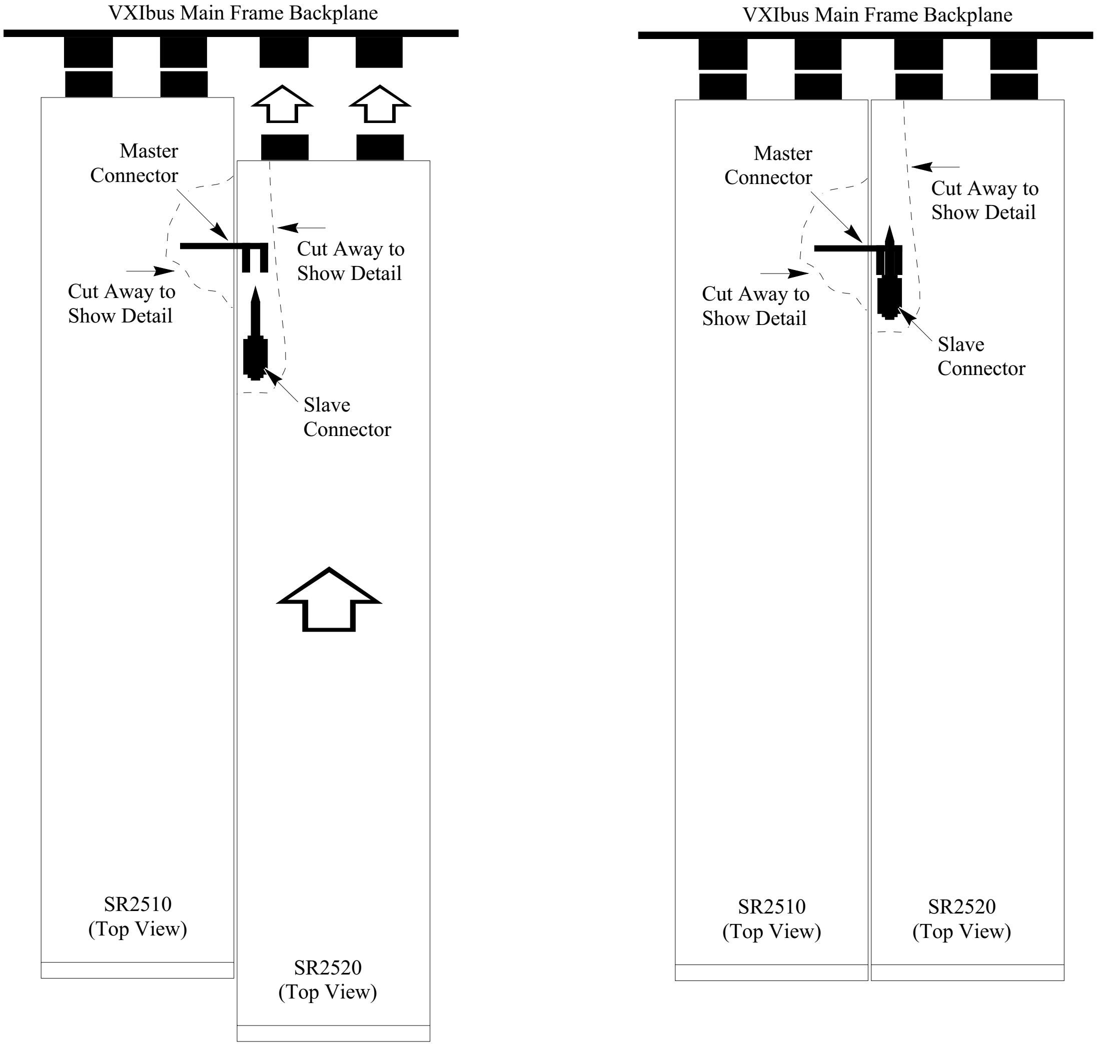


Figure 1-5.
SR2510 and SR2520 Interconnection, Top View.

SR2510 SPECIFICATIONS*

I/O Characteristics:	Differential		Differential		Variable		
	TTL I/O	TTL I/O	ECL I/O	CMOS I/O	Voltage I/O	3.3V Logic I/O	LVDS I/O
Output Drivers							
Type	DS26F31M	74F125	100324	74AC125	-- n/s --	74LVT125	DS90C031
High Voltage (Voh)	3.2V typ	3.4V typ	-1.025V -0.870V^1	4.2V, 24 mA typ	-1.5V to +7.0V^4	3.2V typ	1.14 V typ
Low Voltage (Vol)	0.32V typ	0.55V max	-1.830V -1.620V^1	0.4V, 24 mA typ	-3.0V to + 4.5V^4	0.3V	1.07 V typ
Sink Current	20 mA @ 0.5V	64 mA max	-- n/a --	+24 mA max	50 mA max^2	32 mA max	-- n/a --
Source Current	20 mA @ 0.5 V	15 mA max	-- n/a --	-24 mA max	50 mA max^2	-32mA max	-- n/a --
Output Swing	-- n/a --	-- n/a --	-- n/a --	-- n/a --	0.0V to 11.0V p-p	-- n/a --	-- n/a --
Resolution	-- n/a --	-- n/a --	-- n/a --	-- n/a --	10 mV	-- n/a --	-- n/a --
Absolute Accuracy	-- n/a --	-- n/a --	-- n/a --	-- n/a --	100 mV	-- n/a --	-- n/a --
Abs. Max. Volt. (Hi-Z)	-- n/a --	-- n/a --	-- n/a --	-- n/a --	-3.0V to +7.0V	-- n/a --	-- n/a --
Output Impedance	-- n/a --	100 ohms	-- n/a --	100 ohms	50 ohms	100 ohms	-- n/a --
Input Receivers							
Type	DS26F32M	74ACT244	100325	74ACT244	-- n/s --	74ACT244	DS90C032
Diff. Input Volts (Vth)	0.2V min	-- n/a --	-- n/a --	-- n/a --	-- n/a --	-- n/a --	±200 mV max
Max Input Volts	±5.0V max	+5.0V max	-- n/a --	+5.0V max	-3.0V to +7.0V	+5.0V max	-0.3 to 4.8 V
Input Voltage, high (Vih)	-- n/a --	2.0V min	-1.165V -0.870V^3	2.0V min	-- n/a --	2.0V min	-- n/a --
Input Voltage low, (Vil)	-- n/a --	0.8V max	-1.830V -1.475V^3	0.8V max	-- n/a --	0.8V max	-- n/a --
Input Thrsh, high (Vth)	-- n/a --	-- n/a --	-- n/a --	-- n/a --	-2.9V to +5.5V	-- n/a --	-- n/a --
Input Thrsh, low (Vtl)	-- n/a --	-- n/a --	-- n/a --	-- n/a --	-2.9V to +5.5V	-- n/a --	-- n/a --
Resolution	-- n/a --	-- n/a --	-- n/a --	-- n/a --	10 mV	-- n/a --	-- n/a --
Absolute Accuracy	-- n/a --	-- n/a --	-- n/a --	-- n/a --	100 mV	-- n/a --	-- n/a --
Input Impedance	100 ohms	10k ohms	50 ohms to -2.0V	10k ohms	> 50k ohms	10k ohms	100 ohms

Notes: n/a = not applicable; n/s = not specified; Note 1: Min-Max, Measured with 50 ohm termination to -2.0 V dc bus;
Note 2: Aggregate static source/sink current is 800 mA per 32 channels; Note 3: min-max, single-ended; Note 4: unterminated

CPU:	
System Processor	Motorola 68EC030 @ 25 MHz
Control Processor	25 MHz Custom Gate Array
Internal Clock:	
Range	5.0 ms to 40 ns, 200 Hz to 25 MHz
Resolution	≤ 0.005%
Data Output Jitter	10 MHz reference jitter + 100 ps (short term RMS)
External Clock:	
Range	DC to 25 MHz
Pulse Width	20 ns (minimum)
Active Edge	Rising or falling
Input Voltage	-5.0 to + 10.0 V
Input Threshold	-5.0 to +4.99 V, in 20 mV steps
Input Impedance	1 Megohm
External 10 MHz Ref Input:	
Input Coupling	Capacitor coupled
Input Signal Waveform	Square to sine wave
Input Voltage Level	1-5 V p-p
Input Impedance	High impedance
External Trigger Input:	
Active Edge	High or low
Input Voltage	-5.0 to + 10.0 V
Input Threshold	-5.0 to +4.99 V, in 20 mV steps
Input Impedance	1 Megohm
External Gate Input:	
Active Edge	Rising or falling
Input Voltage	-5.0 to + 10.0 V
Input Threshold	-5.0 to +4.99 V, in 20 mV steps
Input Impedance	1 Megohm
External Input Flags:	
Receiver Type	74ACT244
Number	Eight
Active Level	High or Low
Input Voltage	Vil < 0.8V; Vih > 2.0 V
Input Impedance	10k ohms
Clock Output:	
Driver Type	74F244
Output Level	TTL
Pulse Width	20 ns, minimum
Output Termination	50 ohm, series
I/O Timing:	
Delay Range	1 Test Cycle
Delay Resolution	5-10 ns, depending on frequency
Stimulus Format Clocks	
Resolution	5-10 ns, depending on frequency
Min. Pulse Width	10 ns
Max. Pulse Width	1 Test Cycle - 10 ns
Response Sample Clocks (Edge or Window)	
Resolution	5-10 ns, depending on frequency
Min. Window Width	10 ns
Max. Window Width	1 Test Cycle - 10 ns
Setup Time	10.0 ns, min.
Hold Time	10.0 ns, min.
Skew	± 2 ns (typ) across same type I/O, within single module 3 ± 1 ns (typ), across same type I/O, cumulative, across multiple modules
Data Formats:	
NRZ	Non-Return-to-Zero
RZ	Return to Zero
RONE	Return-to-One
RC	Return-to-Complement
RI	Return-to-Inhibit / Tristate

VXI Specifications

Interface Compatibility:	
SR2510	Message-based, Bus Master/Servant
SR2520	Register-based, Servant
Revision	1.4
Size	C-size, Dual slot
Configuration	Static
Interrupt Level	Programmable 1-7
Triggers	TTLTRG 0-7
Memory(SR2510)	1 MB VME A32/D32/D16/D8 (EO)
Power Requirements: (Note 2)	
+5.0 volts	21.5 A, max.
-5.2 volts	1.0 A, max.
+12.0 volts	0.1 A, max.
-12.0 volts	0.1 A, max.
-2.0 volts	1.0 A, max.

Note 2: Power values specified are with three TTL I/O cards installed.

Cooling Requirements:	
Per Slot Avg.	117 W, maximum per module (Note 2)
Airflow	8 liters / sec per module; 4 liters / sec per slot @ 0.2 mm of water pressure / 10°C temp. rise
Environmental Specifications:	

Temperature	Storage = -40°C to +75°C
	Operating = 0°C to +45°C

Humidity	5% to 95% relative, noncondensing
Software Drivers:	
National Instruments	LabView
National Instruments	LabWindows/CVI

* *Specifications subject to change without notice.*

CHAPTER 2

Theory of Operation

Block Diagram

See Fig 2-1. The major components of the SR2510 are a Timing/Control circuit board, up to three I/O boards, and up to six Driver/Receiver circuit boards (2 for each I/O board). Other components include circuit boards for timing distribution, power distribution and interface logic for SR2520 expansion modules. The major components -- Timing/Control board, I/O boards, and Driver/Receier boards -- are shown in Fig 2-1.

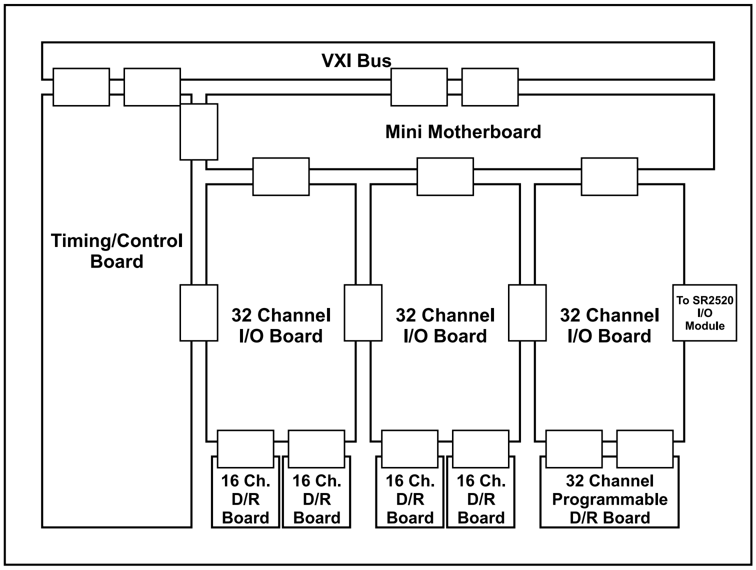


Fig 2-1. SR2510 Block Diagram, Major Components.

Timing/Control Board

See Fig 2-2. The Timing/Control board contains a 25 MHz 68EC030 microprocessor (system processor) that provides the basic user interface to the SR2500 system. The 25 MHz 68EC030 system processor is used to interpret and execute the SCPI word serial commands. It also serves as bus master when transferring data to the SR2520 expansion modules. This processor forms the basic user interface of the SR2500 subsystem and is responsible for non-real-time setup, query, and control functions such as loading and compiling a test, initializing and arming the hardware, monitoring a test in progress and returning test results to the host. The system processor has access to all of the following sub-systems:

Dynamic RAM

This 8 MByte RAM is used to store test configuration and management data for the SR2500 system.

EPROM

The SR2500 operating system is stored in a 512 Kbyte EPROM memory.

Flash RAM

The SR2510 Timing/Control board may be configured with 512 KBytes of flash RAM.

VXI Interface

The SR2510 is a message-based VXI instrument. The VXI interface is based on the Interface Technology IT9010M message-based VXI interface chip. The system processor has access to all of the registers within the IT9010M.

VME A32 Memory

The 1 MByte A32 memory provides a high-speed alternative for transferring test setups and data to/from the SR2500 subsystem. Data can be transferred to/from the slot-0 controller to/from the A32 memory using D8, D16 or D32 transfers. Data is transferred between the SR2510 and SR2520 modules using D32 only.

Control Processor and Memory

The system processor has direct access to registers within the control processor, which are used during initialization and operation of test programs. The system processor also has read/write access to the control processor’s instruction memory when the control processor is stopped, allowing the system processor to download test sequencing instructions and to perform memory diagnostics.

PLO Interface

The PLO interface allows the system processor to control the frequency of the internal PLO oscillator.

Input Buffer Thresholds

The front panel external clock, gate and trigger inputs have programmable input thresholds that are adjustable over a range of ± 4.99 Volts. These are high impedance inputs greater than 1 megohm.

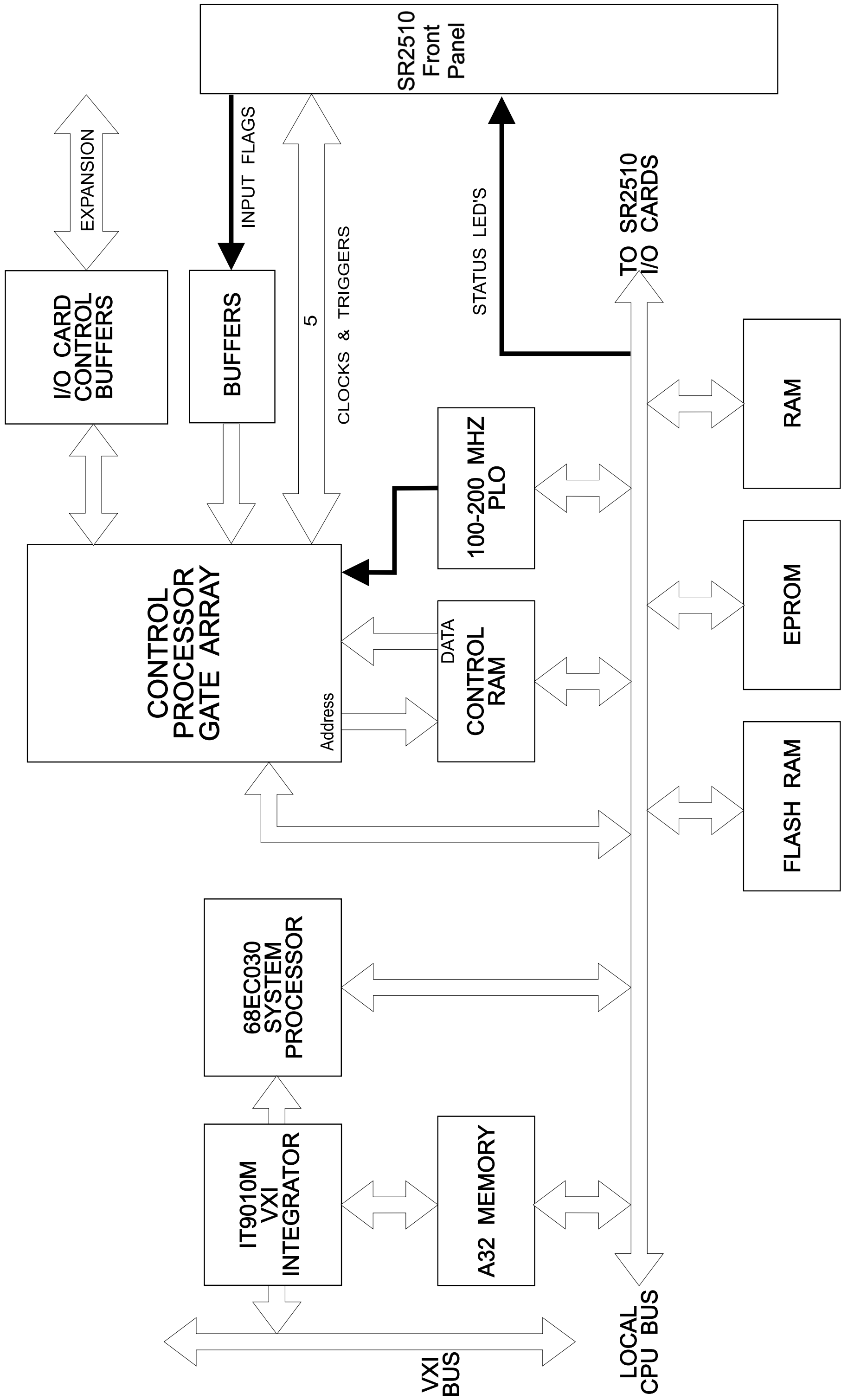


Figure 2-2.
SR2510 Timing / Control Board
Block Diagram.

IT9010M VXI Interface

The SR2510 Timing/Control board uses an Interface Technology IT9010M interface chip to implement the VXI message-based interface and includes bus master, A32/D32 and A32/D16 support. The IT9010M provides all of the low level data transfer signals and protocols, as well as the VXI registers needed for a message-based device. In addition, the IT9010M supports A32 memory access and bus master. The A32 address space can be accessed from either the VXI bus or the 68EC030 processor with all arbitration provided on the IT9010M chip.

Bus master arbitration is also built into the IT9010M. When in the bus master mode, the IT9010M provides a transparent data path from the 68EC030 to the VXI bus. When a command is received by the SR2510 that requires communication with a SR2520 expansion module, the 68EC030, through the IT9010M, requests to become the VXI bus master. When bus mastership is granted, the 68EC030 carries out the requested operation directly with the register-based module without further assistance from the slot-0 controller, again, through the IT9010M. When the operation is complete, the VXI bus is released and the operational status message is updated. This process can be quite lengthy with some operations, so care must be taken that a VXI bus transfer not be initiated by the slot-0 controller without reasonable assurances that the bus will be available. Otherwise, a VXI bus time-out error can occur.

VME A32 Memory

A 1MB block of RAM is mapped to the VME A32 address space and is accessible by either the slot-0 controller or by the 68EC030. The A32 memory is optional and provides high-speed, binary block transfers to and from the SR2500 subsystem. Two types of transfers are supported with the A32 memory option. In the first type of binary transfer, individual stimulus, response, and record memories on the SR2500 I/O boards may be loaded or queried directly with binary data, rather than loading the memories with ASCII text strings. Since the data is binary, no parsing or conversion of the data from ASCII to binary is necessary. Hence, transfer speeds are increased by several orders of magnitude.

The A32 memory may also be used to "learn" the state of the SR2500 subsystem in the form of a binary memory image. Individual tests, or the entire SR2500 subsystem configuration, may be learned with a single command. Data is transferred in binary blocks using a software protocol similar in concept, if not function, to the XON/XOFF protocol. These learned binary images are the compiled equivalent of the SCPI text commands. A learned setup can be sent back to the SR2500 subsystem, resulting in a configuration exactly the same as when the setup was learned. However, since the binary data represents the already compiled SCPI commands, no additional parsing or interpretation takes place. This provides two advantages: 1) The binary image of the complete setup can be loaded at a much faster rate than the high-level ASCII text-based word serial commands; 2) Because the setup and data is in binary format, the test file cannot be modified or viewed, thus adding a measure of security.

Phase Locked Oscillator (PLO)

The SR2500 system clock is provided by a 100 MHz to 200 MHz PLL stabilized ECL Phase Locked Oscillator (PLO). This PLO is phase locked to a 10 MHz reference and provides the basic timing for all SR2500 tests. The PLO output is divided by eight within the control processor and then distributed throughout the system in eight phases. Thus, the distributed system clock actually operates between 12.5 MHz and 25.0 MHz, inclusive. Logic is provided in the stimulus and response processors located on the I/O boards, which can divide this system clock by an integer in the range of 1 to 65,535 to produce the test clock (cycle clock). In theory, the lowest internal clock rate is 190.74 Hz. However, the actual clock rate is software-limited to 200 Hz.

Frequency Resolution

The resolution of the PLL is 10.0 kHz over the range of 100 MHz to 200 MHz. The PLO output is fed into the control processor where it is divided by four, resulting in a system clock that is programmable in the range of 12.5 MHz to 25.0 MHz, with a 1.25 kHz resolution. Any test operating in the 12.5 MHz to 25.0 MHz range will have a clock resolution of 2.50 kHz. For test frequencies below 12.5 MHz, the system clock is divide by an integer in the range of 2 - 65,535. This results in the resolution also being divided by 2 - 65,535. The following algorithm is used within the SR2510 to alculate the correct PLL and divisor values. The $F_{DESIRED}$ parameter is the desired frequency entered by the user. SYSTEM CLOCK is the clock distributed to all I/O boards, and the actual test vector frequency is SYSTEM CLOCK divided by DIVIDE. In the case where $F_{DESIRED}$ is in the range of 12.5 MHz and 25 MHz, the system clock and the test vector frequencies are the same. Test clock resolution is 1.25 kHz divided by DIVIDE.

FREQ = (Unsigned Long Integer)($F_{DESIRED}$ + .5)

DIVIDE = (Long Integer)(24999999.99 / (Double Precision) FREQ + 1

SYSTEM CLOCK = (Double Precision)(FREQ x DIVIDE)

PLO CLOCK = SYSTEM CLOCK x 8

Frequency Accuracy

The absolute frequency accuracy is dependent on the 10 MHz PLO reference source. The internal reference is accurate to ± 300 ppm with less than 100 ps of short term rms jitter. If the VXI reference clock (CLK10) or the front panel 10 MHz reference clock is used, the system assumes the long term accuracy of that reference source. The supplied PLO reference clock source must be a stable and continuous waveform. The maximum frequency deviation, relative to 10 MHz, must not exceed 1.0%. The maximum short term, rms input jitter must not exceed 200 ps, while the SR2500 rms output jitter will not exceed (reference clock source jitter + 100 ps.)

Control Gate Array

The control gate array contains a high-speed sequencer state machine, called the control processor, that controls global generation of stimulus and response test vectors, see Figure 1-5. The control gate array also contains another independent state machine, called the record state machine, which controls the recording of response input data or error vectors, and signature analysis CRC checksums. The control gate array generates certain control signals and clocks that keep the stimulus and response gate arrays on the I/O boards in sync with the system. In addition, the control gate array processes, in real-time, control inputs and clocks that are generated by both the I/O boards and external hardware. These inputs are used in real-time decision based looping and branching and by the record state machine for data recording and CRC sampling.

Control Processor

See Fig 2-3. The control processor controls the generation of test vectors by executing a program out of the control memory. Instructions for this program are called CMACRO's (Command Macros). The address used to fetch data out of the stimulus and response memories is effectively locked to the same address used to fetch the CMACRO instructions. For example, if the CMACRO program causes a word loop at vector number 27 for ten test cycles, then stimulus vector 27 is output for ten cycles and the incoming data is compared against the data stored at response vector 27 for ten cycles also.

The 68EC030 system processor automatically generates a simple default CMACRO program containing a series of output (do not loop, do not branch) instructions, which is sufficient if a simple RAM-backed, non-algorithmic test is desired. The user may download instructions that cause more complex operations to occur. These instructions have two main purposes:

1. To synchronize the test pattern generation with an external event or condition within the UUT.

2. To assist in the generation of algorithmically defined test pattern

sequences that are much longer than would normally be possible with available test memory. These test pattern sequences are typically used to test structured devices, such as a RAM or microprocessor based device.

The control processor obtains its CMACRO instructions from a 32K x 56 bit memory block called the control memory. To achieve the maximum speed (25 MHz) with less expensive memory chips, two vectors are fetched for each memory access. The use of an internal cache controller makes this process transparent to the user. The cache controller is also used to make certain that hardware loops are seamless, meaning no extra time is required when jumping from the bottom of a loop to the top, or when exiting the bottom of the loop. Seamless loops are useful when generating long test sequences without gaps or splices.

The control processor is actually capable of addressing 1M word of memory (2M vectors); however, current memory technology and packaging constraints limit the practical memory size to 128K words (256K vectors).

The control memory is logically divided into a 12 bit instruction field and a 16 bit literal field. The instruction field specifies the control processor operation to take place, while the literal field contains information used by the instruction. For example, if the instruction is a jump command, the literal field specifies the address.

Control Processor Instructions. The control processor can execute the following CMACRO instructions:

Output (OUTput):

The *OUTput* instruction causes the control processor to step to the next sequential vector at the end of its test cycle. All control memory locations are automatically filled with this instruction by the system processor when a test is initially defined. This instruction requires one clock cycle to execute.

Start Program/End Program (SProgram and EProgram):

The *SProgram* and *EProgram* instructions delimit the beginning and end of a test program. Only one *SProgram* instruction is permitted per test, and must be the first instruction in the test, i.e., at vector number one. Any number of *EProgram* instructions are allowed in a test. Each instruction requires one clock period to execute.

Start Loop Until/End Loop (SLoopuntil and ELoop):

The *SLoopuntil* and *ELoop* instructions mark the range (beginning and end) of a multiple vector loop, respectively. Loop branching is seamless.

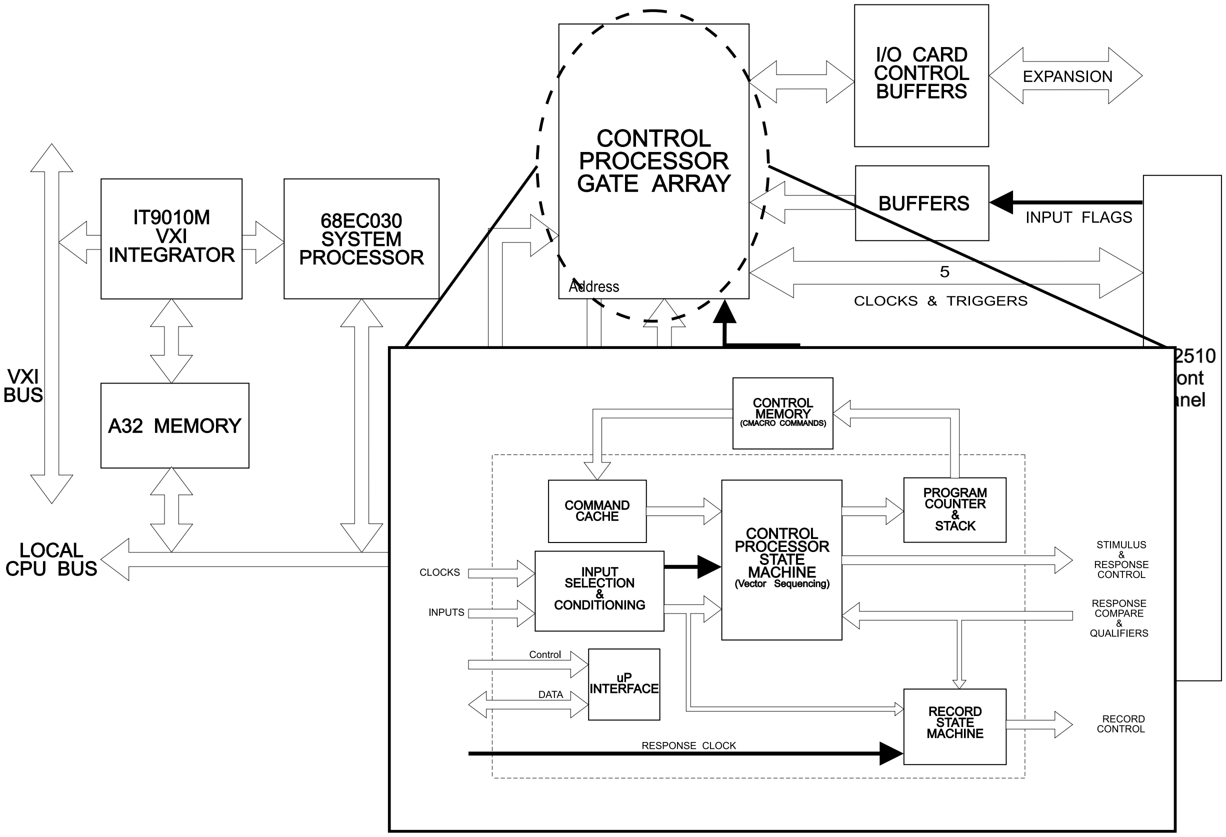


Figure 2-3. Control Processor Gate Array Block Diagram.

Each instruction requires one clock period to execute under all conditions. Although the loop condition is specified by the *SLoopuntil* instruction, it is not tested until the corresponding *ELoop* instruction is executed. If the condition is true, the test falls through to the vector after the *ELoop* instruction. If the condition is false, program execution loops back to vector where the *SLoopuntil* instruction is located, not to the vector following *SLoopuntil*. As a result of the test being performed at the bottom of the loop, the code within a loop will always be executed at least once. The conditions that may be specified by the *SLoopuntil* instruction are discussed later. Start/end loops may be nested two levels deep. For start/end loops, the following rules apply:

Note

Failure to observe the following rules may lead to unpredictable results.

1. For every *SLoopuntil* instruction encountered, the control processor must encounter a corresponding *ELoop* instruction. And for every *ELoop* instruction encountered, the control processor must encounter a corresponding *SLoopuntil* instruction.
2. If a jump to subroutine instruction is executed inside a start/end loop, the program must eventually return before the *ELoop* instruction is executed.
3. If nesting start/end loops, both loops must be in a linear sequence of vectors. It is not permissible to have the first level start/end loop in the main program sequence, and have the second level start/end loop in a subroutine. Either both loops must be in the main program sequence, or both loops must be in the subroutine.

Word Loop Until (WLoopuntil):

The *WLoopuntil* instruction allows looping at a single vector until the defined condition is detected. If the condition is true, program execution continues at the vector after the *WLoopuntil* instruction. If the condition is false, program execution remains at the same vector where the Word Loop instruction is located. The conditions that may be tested by the *WLoopuntil* instruction are discussed later. The *WLoopuntil* instruction requires one clock period to execute under all conditions and the pattern looping is seamless.

Set Condition (SCONDition):

The conditional jump, conditional jump to subroutine and conditional return from subroutine instructions require that the condition being evaluated be set with the *SCONDition* instruction prior to the evaluation. Failure to do so may lead to unpredictable results. This instruction requires one clock period to execute. The conditions that may be specified are discussed later.

Set Jump Page (SJMPPage):

If jumps are performed beyond the first 64K vectors, this instruction must be executed before the jump to specify which of the 64K vector pages to jump to. The *SJMPPage* instruction uses the literal field to specify the page address (0-31). This instruction requires one clock period to execute.

Unconditional/Conditional Jump (JMP and CJMP):

The *JMP* and *CJMP* instructions causes test execution to branch to the vector specified by a predefined label. If the vector is not in the current 64K vector jump page (not currently possible), the *SJMPPage* instruction must have previously been executed. If the jump is conditional, the *SCONDition* command must have previously been executed. This instruction is not seamless and requires four clock cycles for a jump from an odd vector, or five clock cycles for a jump from an even vector. If the *CJMP* is not taken, the instruction requires one clock period to execute. The jump to address is held in the output and tristate memories (for stimulus) and expect and "don't care" memories (for response), so during the jump process, the output pins and the expected response patterns are held with the state and tristate condition from the previous vector. During a jump, pin formatting remains active, optionally, so a pin using a return-to-zero format to generate a clock would remain active during the jump. This would be the same as having an algorithmic HOLD DATA command.

Unconditional/Conditional Jump Subroutine (JSRoutine and CJSRoutine):

The *JSRoutine* and *CJSRoutine* instructions causes test execution to branch to the vector specified by a predefined label. The *JSRoutine* vector must be located on a (32 +1) vector boundary. With a conditional jump, the Set Condition command must have previously been executed. The "jump-to" address is held in the output and tristate memories (for stimulus) and expect and "don't care" memories (for response), so during the jump process, the output pins and the expected response patterns are held with the state from the previous vector. During a jump, pin formatting remains active, optionally, so a pin using a return-to-zero format to generate a clock would remain active during the jump. This functions the same as having an algorithmic HOLD DATA command.

Jump-to-subroutine instructions require 4 cycles to execute if the jump is taken. If the conditional jump is not taken, the instruction requires one cycle to execute. Jump to subroutine instructions may be nested up to eight levels, meaning that eight jump-to-subroutines may be taken before a return-from subroutine must be performed. All jump-to subroutines must have a matching return-from subroutine and all subroutines must be completed before the end program instruction is executed or unpredictable conditions may result. No stack overflow or underflow trapping exists.

Note

As a by-product of initiating (starting) a test, the response compare pipeline is filled with error conditions, and the error latch is set, indicating an error. To use the error latch in a test, the response compare pipeline must be flushed and the error latch reset. This can be done by defining a vector with the value of all "don't care" bits set to "1", then loop on that vector for at least 10 cycles. After the loop, the CLEARError instruction must be executed. If this procedure is not followed, the error latch will always indicate that a response compare error has occurred.

Unconditional/Conditional Return Subroutine (RTSubroutine and CRTSubroutine):

The *RTSubroutine* and *CRTSubroutine* instruction cause the address on top of the stack to be popped and program execution to resume at the vector after the calling jump-to subroutine. If the return-from subroutine is conditional (*CRTSubroutine*), the *SCONDition* command must have previously been executed. This instruction is not seamless. It requires three clock periods for a return from an odd address and four clock periods for a return from an even address. If the conditional return is not taken, the instruction requires one clock period to execute. During a return, pin formatting remains active, optionally, so a pin using a return-to-zero format to generate a clock would remain active during the jump. This functions the same as having an algorithmic HOLD DATA command.

Clear Error Latch (CLEARError):

The *CLEARError* instruction causes the response compare error latch to be reset. The state of the response compare signal is continuously monitored . If in any cycle the response input vector does not match the expect vector for bit locations where the "don’t care" bit is "0", the response compare error latch is set and remains set until the *CLEARError* CMACRO is executed. This instruction requires one clock period to execute. If the response compare error condition is still present while this instruction is executed, the latch is set again during the next clock period. The state of this latch is a control processor test condition, discussed below, and can be queried by the system processor.

Note

Conditions being evaluated for looping and conditional jumping require a 3 cycle + 60 ns setup time. If the "true condition" does not meet the evaluation setup time, the loop will not be exited or the conditional jump will not be taken.

The control processor can execute several conditional instructions. These instructions are used to modify CMACRO program sequencing, in real-time, by performing tests on the following internal and external conditions.

Count (COUNt == (1 - 65535)):

This condition evaluates true after the loop has been executed the defined number of times. The loop value can range from 1 to 65,535. This condition can be used with the start/end loop and the word loop commands, but not with the set condition command, which implies that it may not be used with the conditional jump, conditional gosub or the conditional return from subroutine.

Response Compares (RCOMpare == TRUE):

The response compare condition is true when all response input bits match the corresponding expect bit, where the corresponding "don’t care" bit contains a value of "0". Response compare is a dynamic indication of the results of the input data being compared to the expected response pattern for the current vector only, unlike the error latch.

Response Does Not Compare (RCOMpare != TRUE):

The response does not compare condition is true when any of the input bits do not match the corresponding expect bit, where the corresponding "don’t care" bit contains a value of "0". Response does not compare is a dynamic indication of the results of the input data being compared to the expected response pattern for the current vector only, unlike the error latch.

Error Latch Set (LATCherror == TRUE):

The error latch set condition is true if the response compare error latch is set. The response compare error latch is set whenever a response does not compare condition occurs, and will remain set until cleared by the *CLEARError* instruction.

Error Latch Not Set (LATCherror != TRUE):

The error latch not set condition is true if the response compare error latch is not set. The response compare error latch is set whenever a response does not compare condition occurs, and will remain set until cleared by the *CLEARError* instruction. This instruction may be used with the set condition CMACRO only. It is not an option for word loops or start/end loops.

System Trigger Has Occurred (STRIgger == TRUE):

This condition is true when the currently selected system trigger event occurs. The trigger may be defined as the IEEE 488.2 *TRG command, a word serial trigger, both of which use the bus trigger source, one of the VXI bus TTL triggers (TTLTRG0-7) or the front panel trigger input. The polarity of the VXI bus TTL trigger and the front panel trigger is normally set to the rising edge, but may be inverted to trigger on the falling edge. The front panel trigger input uses a comparator with a programmable threshold that may be adjustable between ± 5.00 volts. This condition may be used with the start/end loop and the word loop commands, but may not be used with the set condition command.

Input Flag Match (FRONtpannel && (#h00-#hFF with X's)):

This command provides a match evaluation of the 8 SR2510 TTL front panel input flags against the 8 bit match pattern. The match pattern is represented as either a hex (#h) or binary (#b) value, which includes X's to denote masked inputs. If the match pattern is represented in hex, then an X will mask out the four corresponding input flags. The condition is true if any of the enabled front panel input flags match the corresponding compare bit. If a match bit is defined as X, then the corresponding input flag is ignored (false). If all bits are X, then evaluation is always false.

Input Flag Does Not Match (FRONtpanel &! (#h00-#hFF with X's)):

As the inverse of the input flags pattern match, the input flag pattern mismatch condition is true if all of the enabled front panel TTL input flags do not match the compare bits. The match pattern is represented as either a hex (#h) or binary (#b) value, which includes X's to denote masked inputs. If the match pattern is represented in hex, then an X will mask out the 4 corresponding input flags. This instruction will always evaluate to true if the match pattern is set to all X's.

Qualifier Match (QUALifword && (#h00-#hFF)):

The SR2500 supports eight system-wide response input comparators called qualifiers (0-7). Each qualifier can be programmed to compare each bit in a expect type field against a "1", "0" or "don't care" value. A qualifier is true if all enabled bits match the input pattern. For example a value of #b00000001 enables qualifier 1 only. A value of #b00000011 or #b00000010 enables qualifiers 1 and 2 or qualifier 2 only, respectively. The condition is true if any of the enabled qualifiers match the pattern on the input pins.

Qualifier Does Not Match (QUALifword &! (#h00 - #hFF)):

As the inverse of the qualifier match condition, this condition is true if none of the enabled qualifiers match the pattern on the input pins. For example a value of #b10000000 enables qualifier 8 only. A value of #b00100100 or #b00010000 enables qualifiers 3 and 6 or qualifier 5 only, respectively.

Record State Machine

See Fig 2-3. The record state machine provides triggering and storing capabilities similar to that of a logic analyzer. The machine contains 16 states or trigger levels, called trace sequences. When a test is started, the record state machine is always initialized to start at trace sequence 1. As the test proceeds, the machine may advance down through the trace levels, or branch out of sequence to different trace sequence levels. At each sequence, controls exist for defining the record systems operation. These operations are divided into the following functional areas:

Filter:

The filter parameter determines the data to be stored in the record memory. The choices are:

- Record the response input data directly from the input pins. This is the default condition.
- Record an error vector in which a bit is set for every input bit that does not match the corresponding expect bit as long the corresponding "don’t care" bit is "0". If the bit does match or the "don't care" bit is "1", a "0" is stored.

Record:

As a subset of the filter parameter, there is another parameter that allows you to define when the filtered data is recorded to memory. The following conditions may be selected:

- **NEVER:** Never store to record memory.
- **ALWAYS:** Store data on each test cycle (default).
- **COMPARE:** Store data only if response compares for the current expected response vector.
- **NCOMPARE:** Store data only if response does not compare for the current expected response vector.
- **QCOMx:** Store data if one or more of the selected input qualifier combinations are true.

CRC:

The CRC parameter determines under which conditions to enable calculation of the signature analysis (CRC). Even when enabled, the CRC calculation is not performed on individual inputs with the "don't care" bit set. The following conditions may be selected:

- **NEVER:** Never calculate a checksum.
- **ALWAYS:** Calculate a checksum on each test cycle (default).
- **COMPARE:** Calculate a checksum only if response compares for the current expected response vector.
- **NCOMPARE:** Calculate a checksum only if response does not compare for the current expected response vector.
- **QCOMx:** Calculate a checksum if one or more of the selected input qualifier combinations are true.

Advance Sequence:

The advance sequence parameter determines the conditions under which the state machine advances from the current trace level to the next trace level. This area is normally used to qualify the input data or advance to a higher trigger state. The next level is the next higher trace sequence number. If both the advance sequence and the jump sequence parameters are used, and if both evaluate true on the same test cycle, then the jump takes priority. The following conditions may be selected for trace sequence advancement:

- **NEVER:** Never advance. This is the default condition.
- **CLOCK:** Advance after a specified number of test cycles.
- **COMPARE:** Advance after a specified number of response compares.
- **NCOMPARE:** Advance after a specified number of response does not compare.
- **QCOMx:** Advance after a specified number of one or more of the selected input qualifier combinations are true.

The advance trace sequence parameter contains a delay counter that may be used to specify the number of times a condition must occur before advancing. The range of this counter is 1 to 65,536 and, by default, is set to 1.

Jump To Sequence:

The jump-to-sequence parameter determines the conditions under which the state machine jumps out of sequence to a new trace level. This section is normally used to disqualify input data and return to a previous or lower trigger state. If both the advance sequence and the jump sequence parameters are used, and if both evaluate true on the same cycle, the jump sequence takes priority. The following conditions may be selected for jumping to a new trace sequence level:

- **NEVER:** Never Jump. This is the default condition.
- **COMPARE:** Jump if response compares.
- **NCOMPARE**: Jump if response does not compare.
- **QCOMx:** Jump if one or more of the selected input qualifier combinations are true.

Stop:

Normally the control processor stops test execution on its own when the program loop has executed the test the specified number of times. However, the stop flag may be set on a given trace sequence so the system is signaled to stop when that sequence is reached. This function sets a flag that is polled by the system processor, which then arbitrarily aborts test execution. Since this is a software procedure, the time the system processor requires to poll and detect the stop flag and then abort the test is unpredictable. Therefore, this function cannot be used as a breakpoint. The control processor is not left in any known state after the abort procedure and cannot be restarted without being re-initialized. Regardless of how the test stops, the user may then query the system to return the recorded data and/or CRC checksums.

The record state machine operates independently of the rest of the system, and contains its own address counter. The number of vectors stored in record memory may not match the number of vectors in the test program.

I/O Board

(Fig 2-4) The I/O board contains the stimulus, response and record logic for 32 channels of output and 32 channels input. Figure 2-4 shows the main components and data paths of this board. The I/O boards installed in the SR2510 module are addressed from the 68030 microprocessor while the I/O boards installed in the SR2520 modules are addressed from the VXI bus as a register-based instrument, (see SR2520 User's Manual for discussion of SR2520 principles of operation).

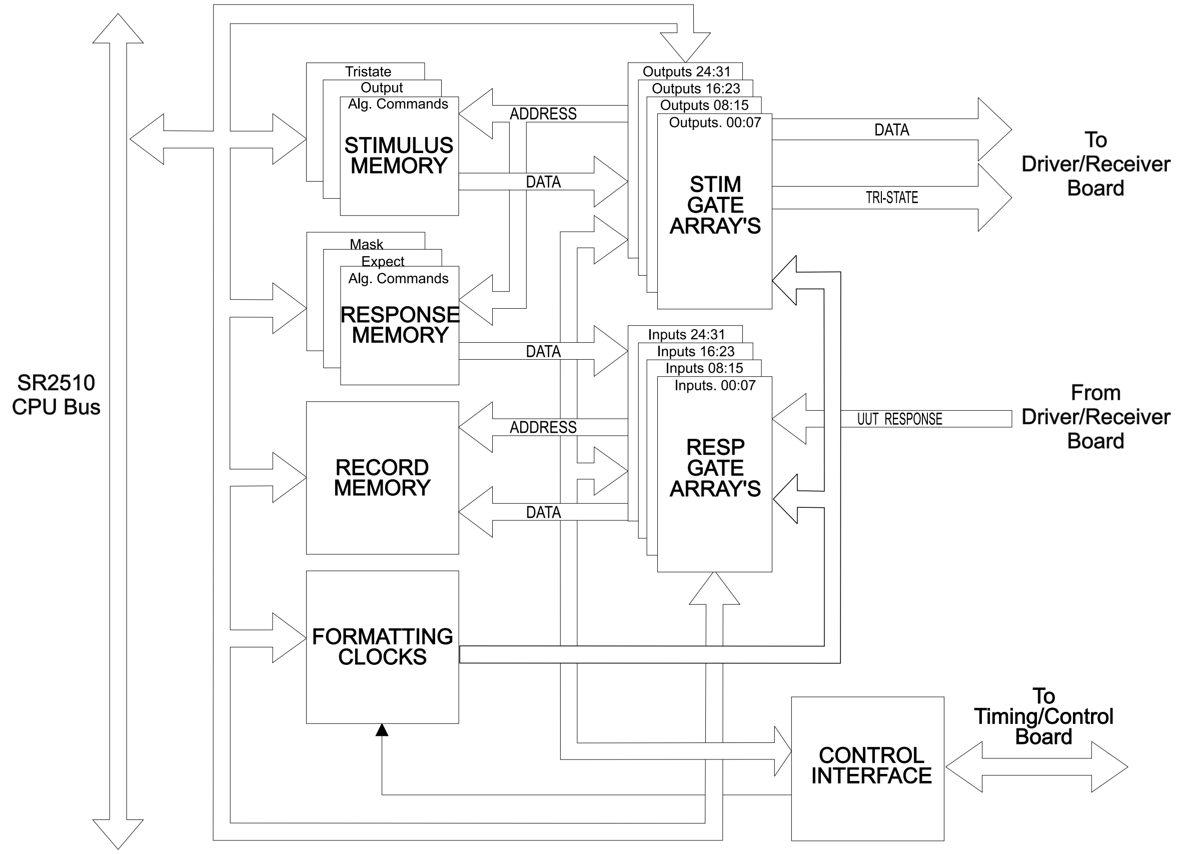


Figure 2-4.
SR2510 I/O Board Block Diagram.

Stimulus and Response Memory

(Fig 2-4) The stimulus and response memory blocks contain data needed to generate the stimulus and expect response data patterns, respectively. The VXI bus can read and write this memory when the control processor is not running. When the control processor is running, access to the VXI bus is blocked and the stimulus and response gate arrays have exclusive read-only access to the memory. The address counters regenerate the control processor address on each stimulus gate array and are used to drive the memory for the pair of stimulus and response gate arrays. Each I/O board contains four of these address buses. However, the buses are effectively locked together with the control processor's address counter.

Record Memory

(Fig 2-4) The record memory stores the data returned by the UUT, or the results of the data returned by the UUT and compared to the data provided by the expected response pattern generator. This is a read only memory for the user and can be read only when the control processor is not running. When the control processor is running, access to the memory by the user is blocked and the response gate arrays have exclusive write-only access to the memory. The record address counters are generated on each response gate array and are effectively locked together. The record address counter is independent of the control processor’s address counter, which controls stimulus and response vector sequencing.

Delayed Clock Generators

As discussed previously, the SR2510 Timing/Control board provides a eight phase system clock, which is distributed throughout the SR2500 subsystem. The actual test vector rate is the system clock divided by an integer in the range of 1 to 65,535. These system clock cycles and phases are available to the stimulus logic to use for the data format delay and width parameters, and they are also available to the response logic to define the edge and window sample timing parameters.

In each stimulus gate array, logic is provided so the output pins, individually or in groups, may select any phase and any cycle of the system clock to assert the output when the NRZ data formatting mode is used. The assert time is synonymous with format delay. Additional logic is provided so that the output pins may select any other phase of any other cycle of the system clock to define the de-assert (deny) times for the return-to data formatting modes. The de-assert time is synonymous with format width.

For example, if the test rate is defined at 25.0 MHz, there is a single system clock for each test vector cycle. Therefore, there are eight phases (points, times) that are available for use with data formatting. The 40 ns test cycle period, divided by the eight available phases, yields an edge placement resolution of 5 ns. If the test rate is defined as 10 MHz, the system clock is set to 20 MHz, then divided by two. This means there are two system clocks for each test vector cycle. Therefore, there are 16 phases (points, times) that are available for use with data formatting. The 100 ns cycle period, divided by the 16 available phases, yields an edge placement resolution of 6.25 ns. Edge placement resolution will always fall within the range of 5-10 ns, regardless of the defined test rate.

The response gate arrays provide similar capabilities for use with edge and window sample modes. Each response input pin can use one system clock phase/cycle for the edge sample mode, or two system clock phase/cycles in the window mode.

Stimulus Gate Arrays

(Fig 2-5). The stimulus gate arrays, in conjunction with the stimulus memories (output, tristate and algorithmic command) form the heart of the SR2500 stimulus pattern generator. Each gate array is an 8 bit wide, high-speed pattern generator and data formatter. Pattern generation is accomplished by outputting the contents of the stimulus RAM directly or by algorithmically generating the data within the gate array using a high-speed ALU state machine. Some gate arrays may be programmed for RAM-backed pattern generation, while other gate arrays on the same card may be programmed for algorithmic pattern generation. While any gate array supports only one type of pattern generation during any test run, one of the algorithmic commands instructs the ALU state machine to pass data directly from RAM to the outputs. This allows mixing of algorithmic and RAM-backed pattern generation on the same pins.

Algorithmic Stimulus Pattern Generator

The pattern generator within the stimulus gate array is a high-speed programmable state machine. Instructions for this state machine are stored in the stimulus algorithmic command memory and instruct the gate array on a test clock-by-clock basis to either load the ALU output register from RAM or to algorithmically modify the contents of the ALU register. The output memory holds the clock-by-clock state of the output pins. The tristate memory holds the clock-by-clock state of the output enable, which allows a pin to be driven by the output memory on one clock cycle and tristated on the next clock cycle, thus achieving a bi-directional pin. In algorithmic mode, data patterns are defined by applying an algorithmic function to the internal ALU register. Multiple stimulus gate arrays may be cascaded together to create 16, 24 or 32 bit wide algorithmic patterns.

Stimulus Output Pin Formatter

Each output channel contains a pin formatter that provides the following data formats: Non Return-to-Zero (NRZ), Return-to-Zero (RZ), Return to One (R1), Return-to-Compliment (RC), and Return-to-Inhibit (RI). The

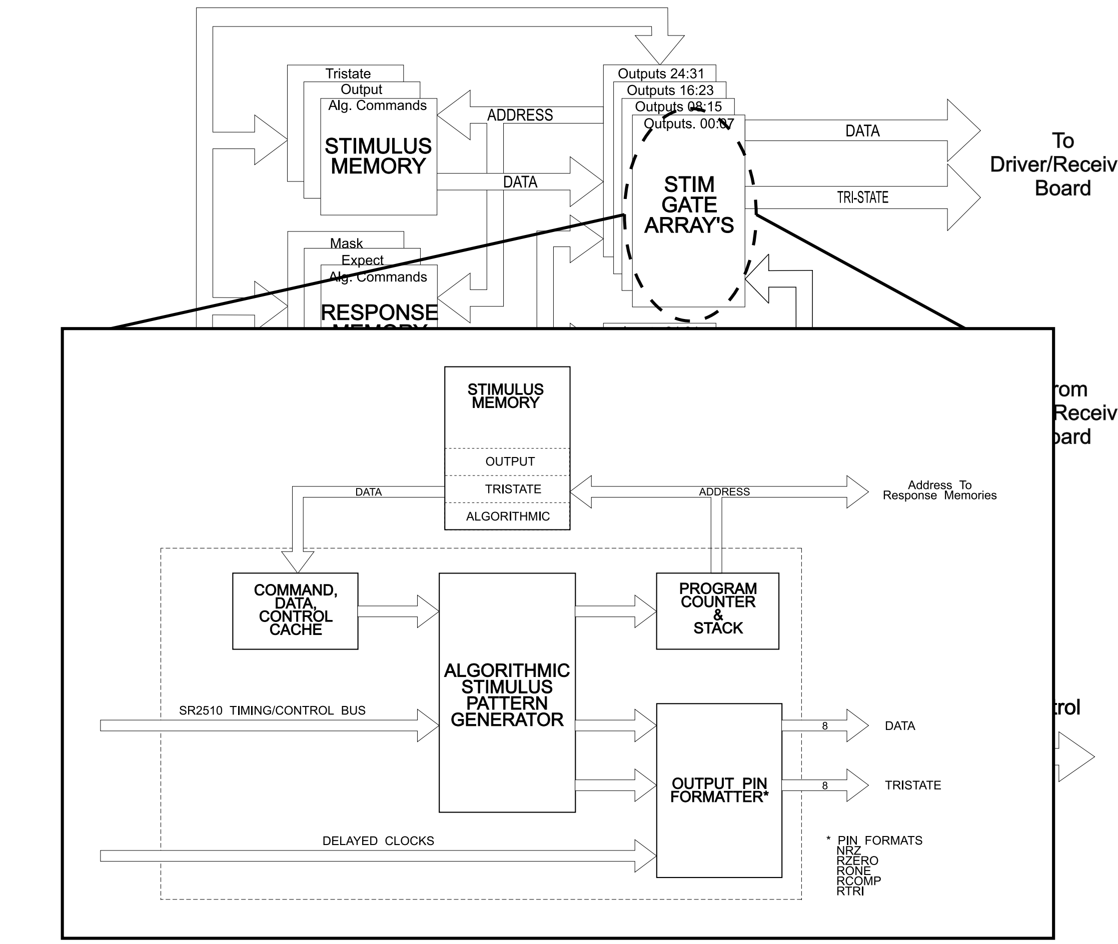


Figure 2-5.
Stimulus Gate Array Block Diagram.

pin formatter section of each gate array can access the available system clock cycle/phase combinations, described in the section on *Delayed Clock Generators*, to define the assert and deny times for the output channels data format.

Response Gate Arrays

(Fig 2-6). The response gate arrays, in conjunction with the response memories (expect, "don't care" and algorithmic command) form the heart of the SR2500 expected response pattern generator, used in real-time compare operations. Each gate array is an 8 bit wide, high-speed pattern generator. Pattern generation is accomplished by outputting the contents of the RAM directly or by algorithmically generating the data within the gate array using a high-speed ALU state machine. Some gate arrays may be programmed for RAM-backed pattern generation, while other gate arrays on the same card may be programmed for algorithmic pattern generation. While any gate array supports only one type of pattern generation during any test run, one of the algorithmic commands instructs the ALU state machine to pass data directly from RAM to the outputs. This effectively allows mixing of algorithmic and RAM-backed pattern generation on the same pins.

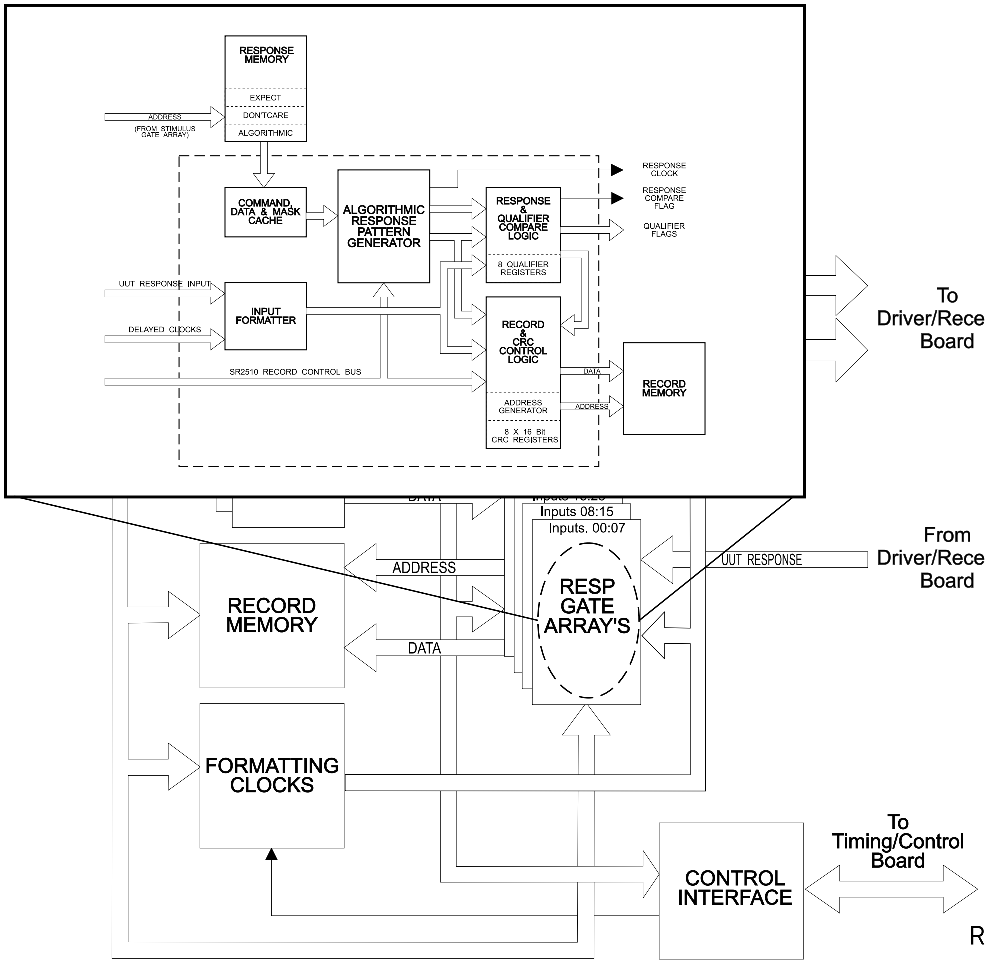


Figure 2-6.
Response Gate Array

Expected Response Pattern Generator

The expected pattern generator within the response gate array is a high-speed, programmable state machine. Instructions for this state machine are stored in the response algorithmic command memory and instruct the gate array on a test clock-by-clock basis to either load the ALU output register from RAM or to algorithmically modify the contents of the ALU register. The expect memory holds the clock-by-clock state of the expected response pattern. The "don't care" memory holds the clock-by-clock state of the compare enable, which allows a pin to be disabled for compare on one clock cycle and enabled for compare on the next clock cycle. In algorithmic mode, data patterns are defined by applying an algorithmic function to the internal ALU register. Multiple gate arrays may be cascaded together to create 16, 24 or 32 bit wide algorithmic patterns.

Response Input Formatter

The response input formatter latches the response data from the UUT and passes it on to the response compare logic and the record control logic.

The pin formatter section of each gate array can access the available system clock cycle/phase combinations, which are used to define the edge and window sample times. In the edge mode, data is sampled at the selected system clock cycle/phase. In the window mode, data must be stable from the time when the window is opened (the first selected system clock cycle/phase combination), to the time the window closes (the second selected system clock cycle/phase combination). Window compare is used for detecting signal glitches.

Response Comparator

The response Comparator logic compares the data latched by the input formatter to the pattern generated by the response pattern generator, and passes the result to the record control logic. A response compare signal is generated for each test cycle, regardless of whether the real-time compare mode is being used. These signals, one from each I/O board, are summed on the SR2510 and used for generation of the error latch, and may also be used for test sequence control decisions based on real-time compare results.

Input Qualifier

The response gate arrays also contain 8 qualifier trigger registers and the qualifier compare logic. All 8 qualifier triggers (qualifiers) are compared to the latched input data that is passed from the input formatter on each test cycle. The results of the qualifier compare are passed back to the SR2510 module. The qualifier compare signals from each I/O board are summed on the SR2510 and used by the record state machine to start and stop data recording, provide filtered data recording, to start and stop CRC sampling. These signals may also be used for test sequence control decisions based on qualifier compare results.

Record Control

The record control logic in the response gate array provides two main functions: to generate record memory addresses and to pass data to the record memory for storing. This logic receives instructions from the SR2510 record state machine, located in the control processor, which determines when to record data and what data to record. After each record operation the record memory address is incremented by one, therefore, all data is recorded in a continuous, linear sequence. Since the record memory is addressed separately from the stimulus and response memories, there is no guarantee that the number of record vectors is the same as the number of stimulus and response vectors. Also, if record data wrapping is enabled, and the number of vectors recorded exceeds the size of the test, the oldest data in the record memory is overwritten. This will continue until the test stops or is aborted. record memory is then rearranged to provide a linear sequence of recorded data from oldest to most recent, accessed from the first vector to the last vector, respectively.

Two types of data may be passed to the record control logic for recording; the UUT response data latched by the input formatter, or the results of the real-time comparison performed in the response comparator. The latter is known as error data, or errors, and is represented as a 0 stored for each bit where the compare matched, and a "1" stored for each bit where the compare did not match. Selecting which data to record may be changed from within the SR2500 test using control structures called trace sequences. As there are 16 levels of trace sequences, this start and stop process of recording data may occur multiple times in a single test, allowing invalid or inappropriate responses to be ignored. For additional information about trace sequences, refer to the Record State Machine section earlier in this chapter.

CRC Logic

Each input pin on the SR2500 I/O board has a 16 bit register and logic used for calculating CRC signatures, all located within the response gate arrays. CRC calculations are controlled from the same trace sequences as are used to control data recording. Based on matching of a trigger condition, CRC calculations may either be enabled or disabled. As there are 16 levels of trace sequences, this start and stop process of calculating CRC signatures may occur multiple times in a single test, allowing invalid or inappropriate samples to be ignored. For additional information about trace sequences, refer to the *Record State Machine* section earlier in this chapter.

For the purpose of signature analysis, each input pin may be thought of as a separate serial channel. So, each SR2500 I/O board has 32 independent signature analysis channels. Enabling or disabling the CRC calculation is performed globally within the SR2500 system using the trace sequences. The "don't care" memory, which is used to enable individual bits for real-time compare, is also used to dynamically enable or disable individual CRC calculations. If CRC calculations are globally enabled, and the individual CRC calculation is enabled ("don't care" bit set to "0"), a CRC calculation is performed. If the individual CRC calculation is disabled ("don't care" memory set to "1"), the CRC calculation is disabled for that channel at that test cycle. When the CRC calculation is enabled, the data passed from the input formatter is used to update the value in the CRC registers based on the CCITT standard communication polynomial used to perform CRC calculations. When disabled, the data passed from the input formatter is ignored by the calculation logic, i.e., no calculation takes place. Data is passed to the CRC logic from the input formatter using the same sample clocks used to record data, so timing for CRC samples is identical to timing for record samples.

Algorithmic Commands

The stimulus and response gate arrays each contain algorithmic pattern generators that generate stimulus and response patterns, respectively. The following list of algorithmic commands are common to both stimulus and response pattern generation.

NONAlgorithmic

The Nonalgorithmic command allows the gate arrays to act as a pass through for data from RAM to the output pins. The data that is passed from RAM to output is also used to initialize the algorithmic register. This register can be acted on by other algorithmic commands to modify the data content programmatically after initialization.

INCrement

Increment the contents of the algorithmic register and pass the results to the output pins. If algorithmic fields greater than 8 bits are used, multiple gate arrays are interlinked. If an increment instruction causes an overflow, the overflow is used as a carry input to the next most significant gate array thus extending the count up to a maximum of 2^{32} before roll over.

DECrement

Decrement the contents of the algorithmic register and pass the results to the output pins. If algorithmic fields greater than 8 bits are used, multiple gate arrays are interlinked. If a decrement instruction causes an underflow, the underflow is used as a borrow input from the next most significant gate array thus extending the count up to a maximum of 2^{32} before roll over.

XOR

The XOR instruction will perform a bit-wise exclusive "ORing" of the algorithmic register with the contents of RAM. In this case the RAM acts as a modifier to the register and does not directly load it. In this way, selective bits of the algorithmic register may be complemented before passed to the output pins.

SLEFTZero

Shift the contents of the algorithmic register left (LSB to MSB) one bit, fill the LSB with "0" and pass the results to the output pins. If algorithmic fields greater than 8 bits are used, multiple gate arrays are interlinked. In this case, the MSB output of a less significant gate array is used as a LSB input to the next most significant gate array, thus extending the shift to a maximum 32 bits.

SLEFTOne

Shift the contents of the algorithmic register left (LSB to MSB) one bit, fill the LSB with "1" and pass the results to the output pins. If algorithmic fields greater than 8 bits are used, multiple gate arrays are interlinked. In this case, the MSB output of a less significant gate array is used as a LSB input to the next most significant gate array, thus extending the shift to a maximum 32 bits.

SLEFTComplement

Shift the contents of the algorithmic register left (LSB to MSB) one bit, complement the LSB and pass the results to the output pins. If algorithmic fields greater than 8 bits are used, multiple gate arrays are interlinked. In this case, the MSB output of a less significant gate array is used as a LSB input to the next most significant gate array, thus extending the shift to a maximum 32 bits.

RLEFT

Rotate the contents of the algorithmic register left (LSB to MSB) one bit, wrap the MSB to the LSB and pass the results to the output pins. If algorithmic fields greater than 8 bits are used, multiple gate arrays are interlinked. In this case, the MSB output of a less significant gate array is used as a LSB input to the next most significant gate array and the MSB of the most significant gate array is wrapped to the LSB of the least significant gate array, thus extending the rotate to a maximum 32 bits.

SRIGHTZero

Shift the contents of the algorithmic register right (MSB to LSB) one bit, fill the MSB with "0" and pass the results to the output pins. If algorithmic fields greater than 8 bits are used, multiple gate arrays are interlinked. In this case, the LSB output of a more significant gate array is used as a MSB input to the next least significant gate array and the LSB of the least significant gate array is wrapped to the MSB of the most significant gate array, thus extending the rotate to a maximum 32 bits.

Driver/Receiver Board

The SR2510 I/O Boards have separate I/O pattern generator boards and driver/receiver boards (D/R boards). Each I/O board provides two connectors of 16 stimulus channels and 16 response channels for connecting to the D/R boards. This means that each I/O board can support two logic families, in groups of 16 channels each. The D/R boards come in four different logic types, allowing the user to configure the SR2500 modules with the specific logic families required for the test system. On the stimulus side, the I/O pattern generator boards provide discrete TTL I/O signals to D/R boards, and the D/R boards translate the TTL I/O signals to the appropriate logic levels. For receiving, the D/R board accepts the UUT response and translates the UUT logic level to the TTL level required by the I/O board.

TTL Driver/Receiver Logic

(Fig 2-7) The TTL D/R board provides 16 channels of single ended TTL to/from the UUT. Separate output and input pins are used (32 signal pins), with a ground return for each signal. Bi-directional signals are supported by connecting the output and input pins together. Each TTL driver (74F125) has a 100 ohm resistor in series with the output. This provides 100 ohm back matched termination as well as additional short circuit and over voltage protection. The receiver (74ACT244) provides 10k pull up/down resistors on it's input.

CMOS Driver/Receiver Logic

(Fig 2-8) The CMOS D/R board provides 16 channels of single ended CMOS to/from the UUT. Separate output and input pins are used (32 signal pins), with a ground return for each signal. Bi-directional signals are supported by connecting the output and input pins together. Each CMOS driver (74ACT125) has a 100 ohm resistor in series with the output. This provides 100 ohm back matched termination as well as additional short circuit and over voltage protection. The receiver (74ACT244) provides 10k pull up/down resistors on it's input.

Differential TTL Driver/Receiver Logic

(Fig 2-9) The Differential TTL D/R board provides 16 channels of differential TTL to/from the UUT. Separate output and input pins are used (32 signal pins). Bi-directional signals are not supported directly on the D/R board, however, 16 tristate control signals are also brought out the differential TTL D/R board. These signals may be used on the UUT, or in a UUT adapter, to provide bi-directional control.

Differential ECL Driver/Receiver Logic

(Fig 2-10) The Differential ECL D/R board provides 16 channels of differential ECL to/from the UUT. Separate output and input pins are used (32 signal pins). Bi-directional signals are not supported directly on the D/R board, however, 16 tristate control signals are also brought out the differential ECL D/R board. These signals may be used on the UUT, or in a UUT adapter, to provide bi-directional control. Each side of the receiver input (100325) provides 50 ohm resistors terminated to -2.0V.

Programmable Driver / Receiver Logic

(Fig 2-11) The programmable, (variable voltage) D/R board (VV D/R) provides 32 bi-directional channels of I/O where the V_{OH} and V_{OL} levels are programmable over a range of -3V to +7V, and the V_{TH} and V_{TL} levels are programmable over a range of -2.9 to +5.5V. The V_{OH}, V_{OL}, V_{TH} and V_{TL} voltages are supplied external to the VV D/R board. Unlike the fixed level D/R boards, the VV D/R does not provide separate output and input pins. All pins are bi-directional signals with a ground return for each signal. The driver (EDGE649) is source terminated with a 50 ohm series resistor, and the receiver (EDGE649) provides a 50 ohm damping resistor in series with its input. The receiver is a dual-threshold part, capable of differentiating between a high input level, a low input level and an indeterminate (tristated) input. Additional logic in the form of a multiplexer and a oscillator are added to the output of each input receiver to allow the SR2500 VV D/R to detect/record if the response was valid or invalid. The truth table in Fig 2-10 indicates the various states that can be detected. If the detected state is other than the state that is tested for, the comparison will fail, the error latch will be set, and the record memory will store a "1" for each enable input bit that failed the test. The states that can be tested are a valid high and a valid low.

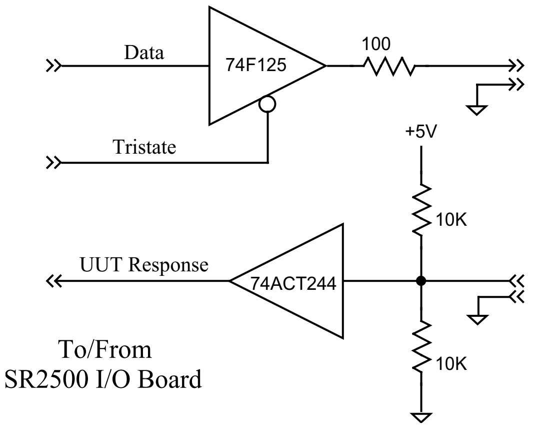


Figure 2-7. TTL Single Ended Driver/Receiver, (16 per D/R Board).

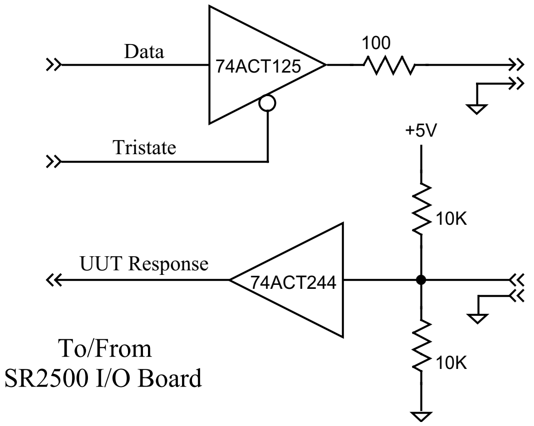


Figure 2-8. CMOS Single Ended Driver/Receiver, (16 per D/R Board).

Output Enables Are In Groups of Four

Tri-state bit 0	enables	bits 0-3
Tri-state bit 4	enables	bits 4-7
Tri-state bit 8	enables	bits 8-11
Tri-state bit 12	enables	bits 12-15
Tri-state bit 16	enables	bits 16-19
Tri-state bit 20	enables	bits 20-23
Tri-state bit 24	enables	bits 24-27
Tri-state bit 28	enables	bits 28-31

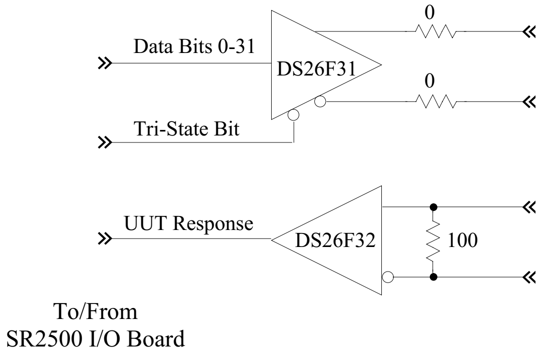


Figure 2-9. Differential TTL Driver/Receiver, (16 per D/R Board).

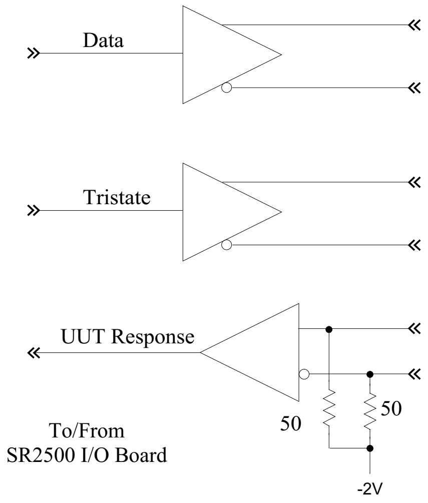
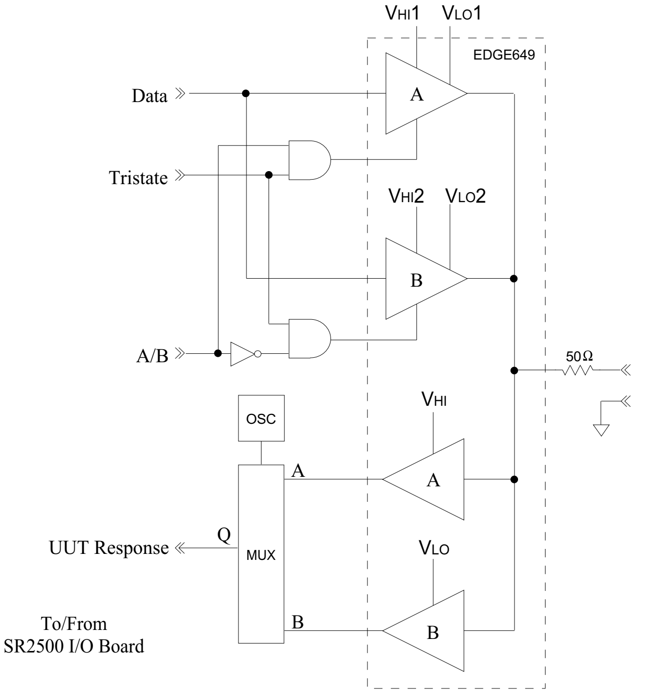


Figure 2-10.
Differential ECL Driver/Receiver,
(16 per D/R Board).

LVDS Driver/Receiver Logic

(Fig 2-12) The LVDS TTL D/R board provides 16 channels of LVDS to/from the UUT. Separate output and input pins are used (32 signal pins). Bi-directional signals are not supported directly on the D/R board, however, 16 tristate control signals are also brought out the LVDS D/R board. These signals may be used on the UUT, or in a UUT adapter, to provide bi-directional control.



A	B	Q
0	0	Low
1	1	High
0	1	Indeterminate
1	0	Invalid

Figure 2-11.
Programmable Driver/Receiver,
(32 per D/R Board).

Output Enables Are In Groups of Four

Tri-state bit 0	enables	bits 0-3
Tri-state bit 4	enables	bits 4-7
Tri-state bit 8	enables	bits 8-11
Tri-state bit 12	enables	bits 12-15
Tri-state bit 16	enables	bits 16-19
Tri-state bit 20	enables	bits 20-23
Tri-state bit 24	enables	bits 24-27
Tri-state bit 28	enables	bits 28-31

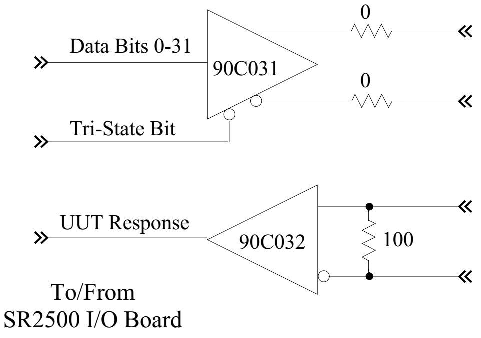


Figure 2-12. LVDS Driver/Receiver,
(16 per D/R Board).

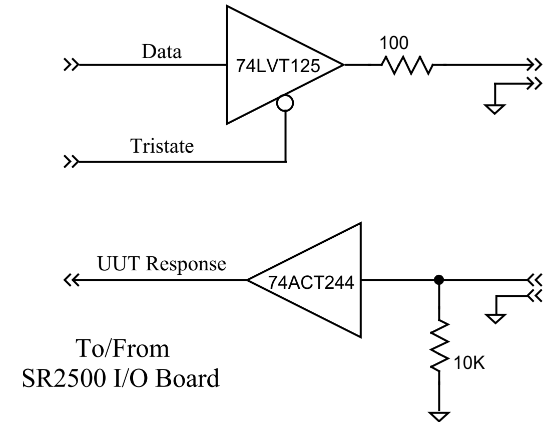


Figure 2-13. 3.3V Single Ended Driver/Receiver, (16 per D/R Board).

3.3 Volt Driver/Receiver Logic

(Fig 2-13) The 3.3V D/R board provides 16 channels of single ended logic to/from the UUT. Separate output and input pins are used (32 signal pins), with a ground return for each signal. Bi-directional signals are supported by connecting the output and input pins together. Each 3.3V driver (74LVT125) has a 100 ohm resistor in series with the output. This provides 100 ohm back matched termination as well as additional short circuit and over voltage protection. The receiver (74ACT244) has 10k pull down resistors on it's input. Both the drivers and receivers are 5V tolerant.

(THIS PAGE INTENTIONALLY LEFT BLANK)

CHAPTER 3

Installation

Scope of Chapter

This chapter contains instructions for unpacking, inspecting, installing, and checking out the SR2510 Main Module.

Unpacking and Inspection

Your SR2510 was thoroughly inspected and tested before shipment from the factory and is ready for immediate operation once all installation procedures have been completed. Carefully remove the instrument from its shipping carton and check for any obvious damage that may have occurred during shipment. If damage is found, report it to the freight carrier immediately. Interface Technology is not liable for damage that may have occurred during transit. Save the shipping carton and all packing material for possible future use.

Installation

Logical Addressing

Before installation, the logical address for the SR2510 Main Module and SR2520 Expansion Modules must be set. Set the address switches according to the requirements of the slot 0 controller. The address switches are numbered from one to eight. Switch 1 corresponds to the least significant bit (LSB) of the logical address. The address is entered in binary, where an ON switch sets the corresponding bit to 0 (Fig 3-1).

Note

The logical addresses of the SR2520 Expansion Modules must be set to a higher value than the logical address of the SR2510 Main Module. If there is more than one SR2510 in a VXI chassis, then the SR2520's with addresses between any 2 SR2510's, will be part of the lower addressed SR2510's system. The SR2520 with the lowest numbered logical address is Expansion Module #1. The next highest SR2520 logical address is Expansion Module #2. The highest SR2520 logical address is the most significant Expansion Module number. To verify all Expansion Modules have been recognized by the system, send a “*IDN?” query command.

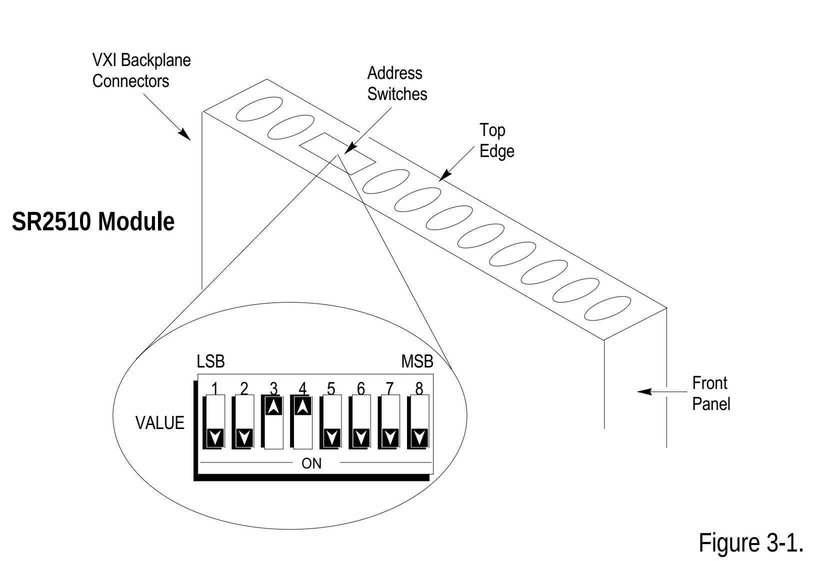


Figure 3-1.
Address Switches Set to Logical Address 12.

Slot Dependency

The SR2510 Module may be installed in any available card slot other than slot-0. SR2520 I/O Modules (if any) must be mounted immediately to the right of the SR2510 Module. The SR2500 uses bus master functions to identify the I/O boards installed in the system, so all SR2500 modules must be located in the same chassis.

Backplane Jumpers

SR2510 Main Module

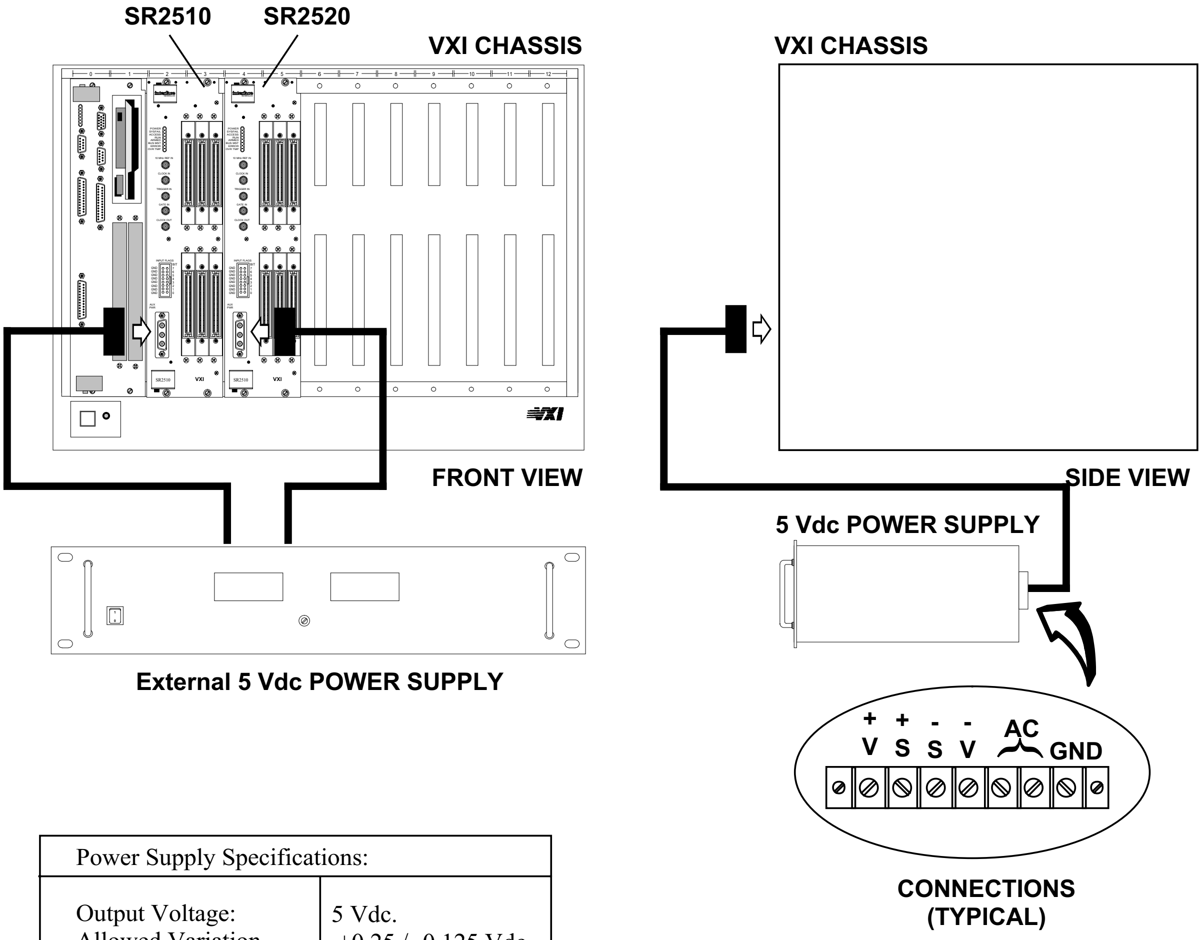
The SR2510 Control Card uses the BG3 and IACK from the VXI backplane; therefore, these jumpers must be removed from the VXI backplane for the first slot that houses the SR2510 Main Module. The BG0, BG1 and BG2 are not used on the Main Module but are passed through. The user may remove or install the jumpers for these signals as required.

SR2520 Expansion Module

The SR2520 Expansion Module does not use any of the IACK or BG3 signals. These signals are passed through. The user may remove or install the jumpers for these VXI slots as required.

5 Vdc External Power Requirements

For SR2510 and SR2520 modules configured with more 32 I/O channels, additional 5 Vdc power is required from an external source. The external power is supplied to the Aux. Power connector located on the module front panel, see Fig. 3-2. The amount of operating current required from the external power supply is directly proportional to the number of modules installed. The SR2510 and SR2520 can each supply enough internal 5 Vdc power to operate up to 32 I/O channels, independently of external power. When more than 32 channels/module are used, approximately 7.5 amperes is required for each additional 32 channels (e.g., a 64 channel module requires 7.5 A from an external 5 Vdc power supply; a 96 channel module requires 15 A).



Power Supply Specifications:	
Output Voltage:	5 Vdc.
Allowed Variation	+0.25 / -0.125 Vdc.
Ouput Current:	30 A (Note 1)
DC Load Ripple/Noise	50 mV
Induced Ripple/Noise	50 mV

Note 1: Amperage indicated is for two (2) modules only. Each additional (fully populated) module (SR2510 or SR2520) requires an additional 15 A (approx) @ 5.0 Vdc.

Power Up Sequencing

Note

It does not matter the order in which the external power supply and the VXI chassis are powered up, as long as both are on and stable when the first SCPI command is sent to the SR2510.

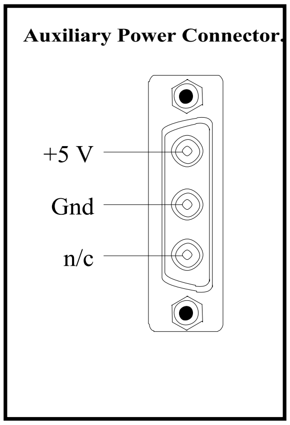


Figure 3-2.
Connection of External 5 Vdc Operating Power.

Option Switch

(Fig 3-3) The Option Switch allows user selection of certain operating parameters, including:

- host cotroller selection
- power-on diagnostics test (fast/slow)
- operating protocol (bus master/release on request)
- test post-processing speed (fast/slow)
- boot mode (EPROM/flash ROM)

Note

The SR2510 is shipped from the factory with all Option Switches set to the OFF (default) position, see Fig 3-3.

Option Switch Settings

To set Option Switch SW3 do this:

1. Turn system power off.

ESD WARNING

Perform the following steps only at an ESD workstation and observe all ESD precautions to avoid damage to the instrument due to Electro-Static Discharge.

Note

Removing the SR2520 modules in the manner indicated is necessary to avoid damage to the interconnect connectors located between the various SR2520s and the SR2510.

2. If there are any SR2520 I/O Modules installed in the VXI chassis, remove all of these modules first, before removing the SR2510. Begin by removing the SR2520 furthest to the right of the SR2510 and continue with the 2nd furthest from the right, the third furthest from the right, etc. until all SR2520s have been removed. Now remove the SR2510 from the VXI chassis.

Note

It is not necessary to remove the protective metal covers from the SR2510 to make changes to the settings of the Option Switch.

Host Controller Selection

Note

This procedures applies to SR2500 Firmware V1.00 (dated 11/22/95) or later.

To use the SR2510 with H-P Slot-0 Controllers, set bit-6 of Option Switch SW3 to ON. Also, when using an H-P controller, always set bit-7 of the Option Switch to ON to select the short power-up diagnostic test.

For use with any other Slot-0 Controller, set bit-6 of Option Switch to OFF (factory default). Set bit-7 of the Option Switch to select either the short (ON) or normal (OFF) power-up diagnostic test, as desired.

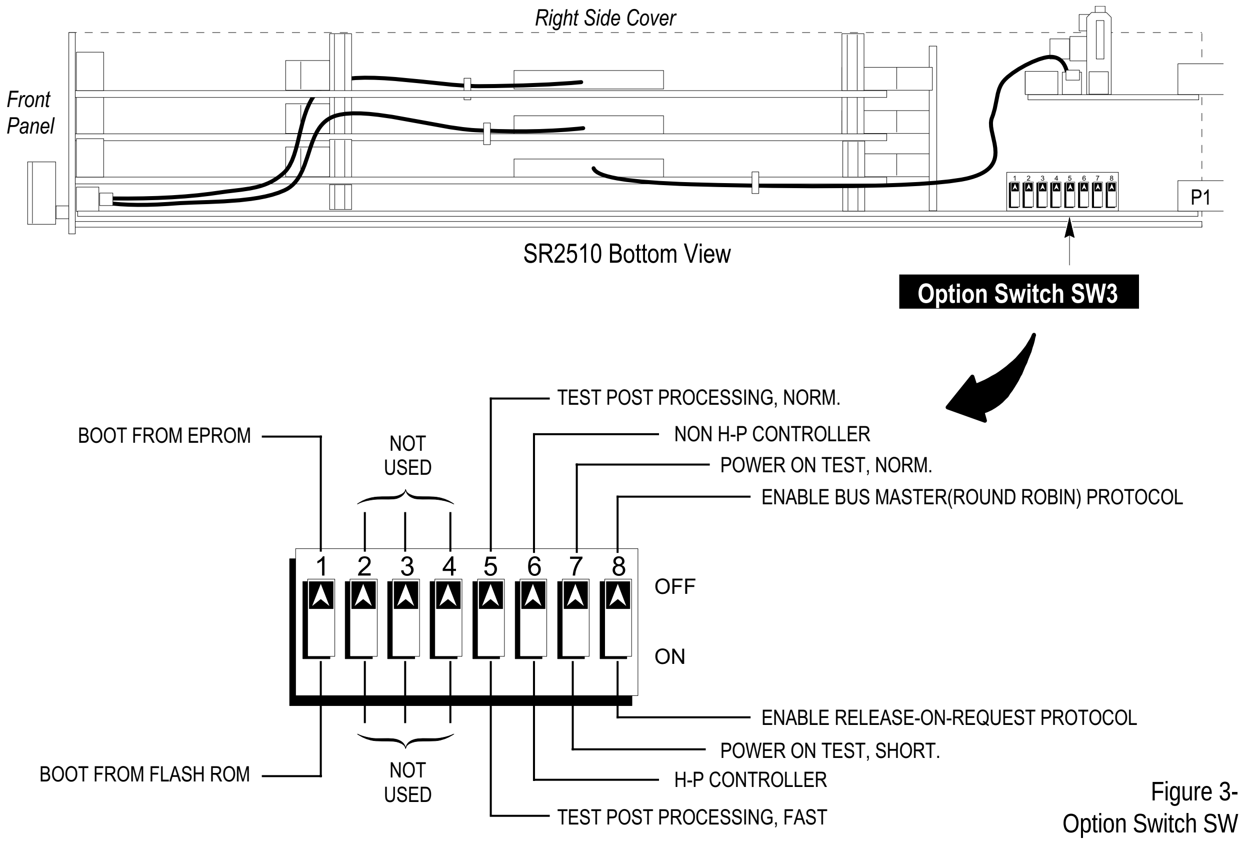


Figure 3-3.
Option Switch SW3.

Note

When using an H-P Slot-0 controller, avoid using the HP command:

VXI:CONF:INF?

Instead, use the following command:

VXI:CONF:DLIST?

Power-On Diagnostic Test (fast/slow)

Two self-diagnostics test modes are available at power-up ... a short test, and a long test. The user may select between either of these tests by selecting the proper setting of the Option Switch (SW3). Table 3-1 is a listing of the self-diagnostics tests performed by the SR2510 firmware at power-up. The first column lists the name of the test; the 2nd and 3rd columns indicate (yes or no) which tests are performed for the long and the short test modes, respectively.

See Fig. 3-3. Set bit-7 of Option Switch SW2 to ON to select the short power-up test; set bit-7 to OFF to select the long (default) test.

Note

When using the SR2510 in conjunction with an H-P Slot-0 Controller, always set the Option Switch for the short diagnostics test, refer to previous procedure for using H-P controllers.

Table 3-1. Self-Diagnostics Tests.

Test Performed	Long Test	Short Test
On-board program DRAM memory tests.	Yes	No
DRAM initialization.	Yes	Yes
VXI shared DRAM memory tests.	Yes	No
Shared DRAM initialization.	Yes	No
ROM checksum.	Yes	Yes
VXI interface chip test.	Yes	Yes
RS-232C / UART test.	Yes	Yes
Control GA access test.	Yes	Yes
SR5000 initialization.	Yes	Yes

Operating Protocol, Bus Master / Release-On-Request

SR2510 modules with firmware version 1.00 and later provide for selecting between Bus Master and Release-On-Request protocol. The SR2510 is shipped from the factory in the Bus Master (round robin) mode ... that is, with bit-8 of Option Switch SW3 set to the OFF (up) position, see Fig 3-3. To select the Release-On-Request protocol, set bit-8 of SW3 to the ON (down) position.

Test Post-Processing Speed (fast/slow)

Note

This procedure only applies to SR2510 modules configured with firmware version 1.34 and later.

To speed up post test processing, the CRC values calculated by the SR2510 can be automatically written to A32 memory, once a test is complete. This function is activated by setting bit-5 of Option Switch SW3 to the ON (down) position, see Fig 3-3. In this mode (fast mode), the 16-bit CRC values for each pin of each card in the SR2500 system is written to A32 memory, starting at the first location. Refer to Table 3-2, which lists details on the locations of each CRC value.

Table 3-2. Storage of CRC Data in A32 Shared Memory.

Comment	Index	Data
Base of Shared Memory	+0	CRC word for Card 1, Pin 1 (16-bits)
	+2	CRC word for Card 1, Pin 2 (16-bits)
	+4	CRC word for Card 1, Pin 3 (16-bits)
	+6	CRC word for Card 1, Pin 4 (16-bits)
	+8	CRC word for Card 1, Pin 5 (16-bits)
	o	o
	o	o
	o	o
	+58	CRC word for Card 1, Pin 30 (16-bits)
	+60	CRC word for Card 1, Pin 31 (16-bits)
End of Card 1	+62	CRC word for Card 1, Pin 32 (16-bits)
Start of Card 2	+64	CRC word for Card 2, Pin 1 (16-bits)
	+66	CRC word for Card 2, Pin 2 (16-bits)
	o	o
	o	o
	o	o
End of Card 2	+126	CRC word for Card 2, Pin 32 (16-bits)
Start of Card 3	+128	CRC word for Card 3, Pin 1 (16-bits)
	o	o
	o	o
	o	o

The index for the CRC word for pin 1 of any card can be calculated usng the formula:

index = [(card number - 1) x 64]

Example:

Find the index for the CRC word for pin 1 of card 3.

index = [(card numer - 1) x 64]
index = [(3 - 1) x 64]
index = 2 x 64
index = 128

Boot Mode (EPROM / Flash ROM)

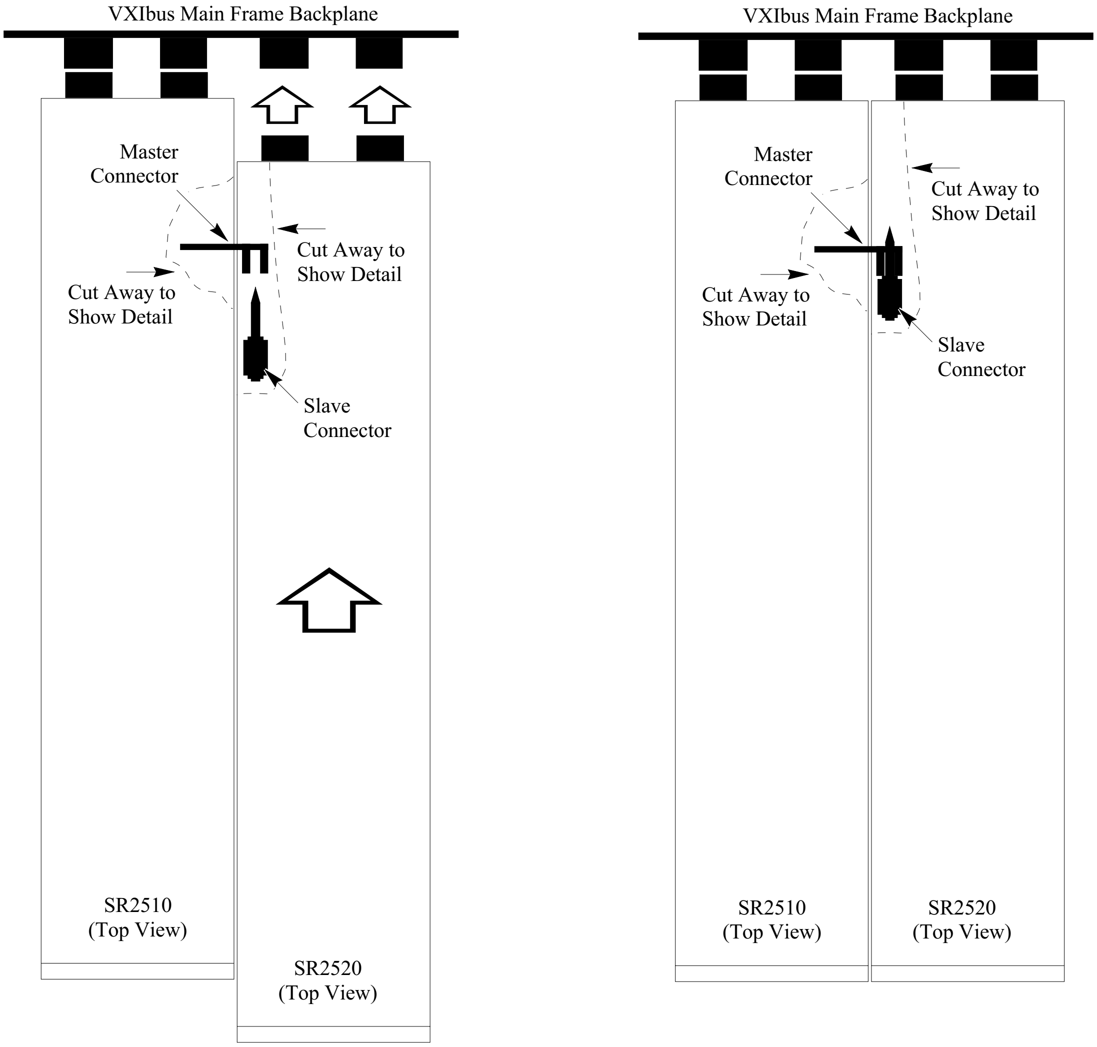
The EPROM boot mode is only used to change (update) the SR2510 firmware. For such operatons, bit-1 of Option Switch SW3 is set to the EPROM (down) position. For all other operating modes, bit-1 of Option Switch SW3 is placed in the Flash ROM (up/default) position, see Fig 3-3.

Note

Instructions for installation are included with firmware updates. This option only applies to SR2510 modules configured with firmware version 2.01 or later.

Main and Expansion Module Interconnect

All interconnections between the SR2510 Main Module and SR2520 Expansion Modules are made by means of the VXI backplane and by a special connector at the side of the module. Interconnections are completed whenever Expansion Modules are added to the system. No additional cabling between modules is required. The second, and subsequent, SR2520 modules are connected in a similar manner.



SR2510 Main Module Fully Mounted in Mainframe; SR2520 Expansion Module Partially Installed.

SR2510 and SR2520 Modules Both Installed in Mainframe; Master and Slave Connectors Mated.

Figure 3-4. Interconnect Between SR2510 and SR2520, Top View.

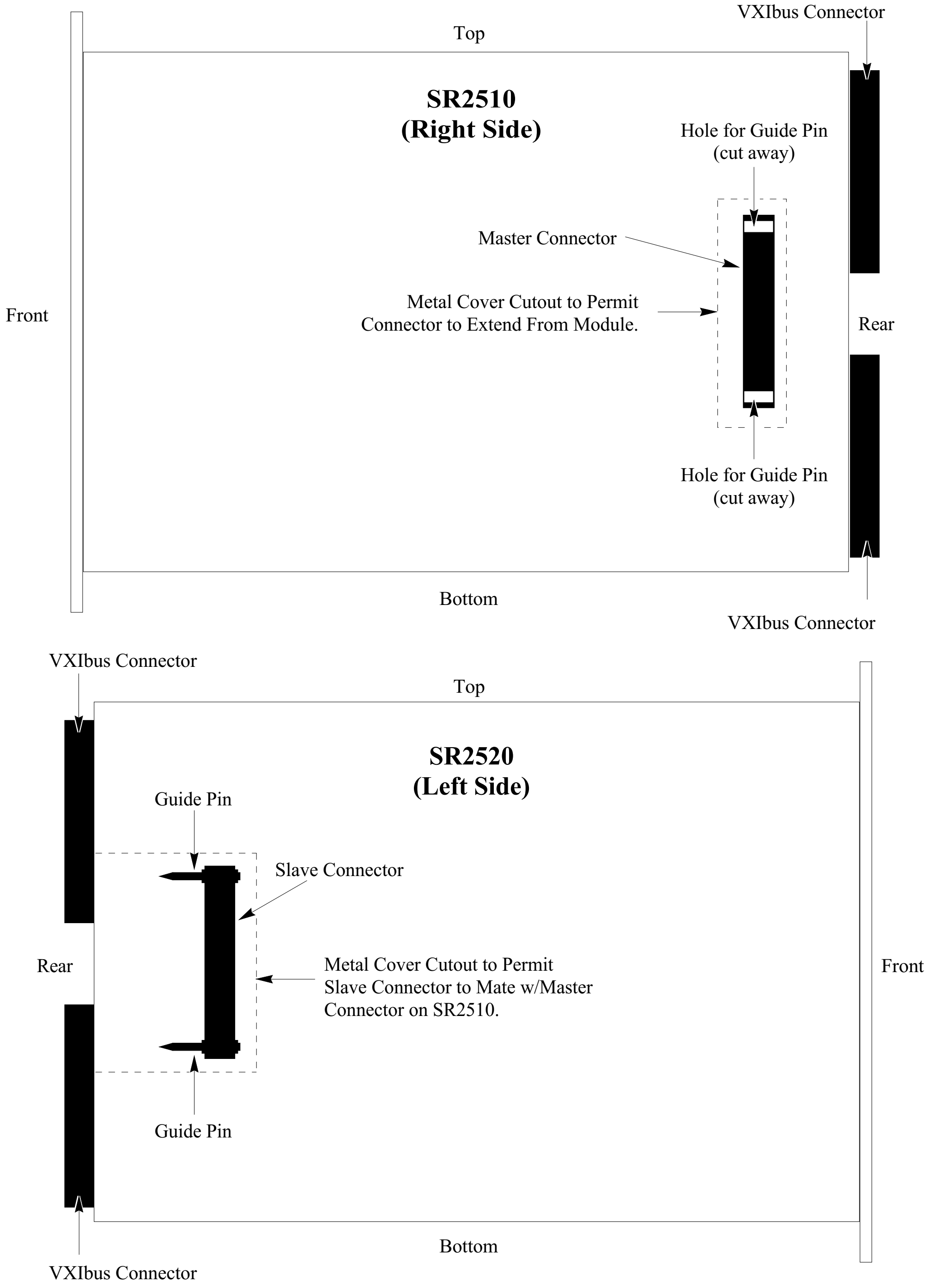


Figure 3-5. SR2510 and SR2520 Interconnect Connectors.

Installing I/O Boards

Each SR2520 modules can contain up to three I/O Boards, each of which provides 32 I/O channels. As shipped from the factory, the SR2510 will contain one, two, or three I/O Boards, depending on the customer's order. If less than three I/O Boards are supplied, cover plates are installed over the unused connector holes in the front panel. Additional I/O Boards can be ordered and installed by the user to expand system capability, at any time.

To install additional I/O Boards, you will need the following tools and materials:

Required Equipment

- screwdriver, No.1 Phillips
- screwdriver, 1/8" blade (pocket type)
- hex nut driver, 3/16"

To install a 2nd I/O Board in an SR2510 module, proceed as follows:

Procedure ... Install I/O Bd No.2

1. Turn VXI chassis power OFF. Disconnect all external cables from front panel of SR2510 module.

CAUTION

If there are SR2520 modules installed in the VXI chassis on the right hand side of the module to which the additional I/O Board is to be installed, these modules must be removed first to avoid damage to the interconnects between modules.

2. Observing the caution above, remove the SR2510 module from the VXI chassis.

3. Place the module on a clean workbench, orient the module to gain access to the right side cover, see Fig 3-6.

4. Remove the 14 #4-40 x .14" Phillips flat head screws securing the cover to the module; 11 screws are located on the side of the module and three more screws are located on the front of the module. The screws to be removed are indicated by the heavy circles in Fig 3-6.

5. Remove the right side cover from the SR2510 module.

6. Remove the 2 cover plates from the front of the module for the I/O Board to be installed ... i.e., cover plates for I/O Board 2 or cover plates for I/O Board 3 are secured to the front panel by four No.4-40 x 1/4" Phillips flat head screws, see Fig 3-7 (note: in Fig 3-7, these screws are shown, blown away from the module, directly to the left of the two Interface Boards).

7. See Fig 3-7 and Fig 3-8. Remove the three spacer stacks at positions 1, 2, and 3, each consisting of two 1/2" and one 7/16" hex spacers (see Fig. 3-8a1). Also remove the No.4-40 x 1/4" Phillips pan head screw, split lock washer and nylon washer at position 4 of the I/O Board (see Fig 3-7 and Fig 3-8b1).
8. See Fig 3-7. Loosen, but do not remove, the three No.4-40 x 1/4" Phillips flat head mounting screws securing the Control / Timing Board to the module front panel. (note: screws indicated by black triangles in Fig 3-7). Also loosen, but do not remove, the two small slotted head retainer screws securing the module latches to the module, see Fig 3-6. The front panel should now swing out slightly, away from the module, to allow access to install the I/O Board.
9. See Fig 3-7.Carefully place the new Expansion I/O Board, with the Interface Board(s) attached, in position inside the module.
10. See Fig 3-7. Connect the ribbon cable to J2; connect the mini-motherboard connector, and attach the power connector at J6 of the newly installed I/O Board.
11. See Fig 3-7 and Fig 3-8a2. Install the three spacer stacks at positions 1, 2, and 3 each consisting of one 1/2" and one 7/16" hex spacers (see Fig 3-8a2). Also install a 7/16" hex spacer and the No.4-40 x 1/4" Phillips pan head screw, split lock washer and nylon washer at position 4 of the I/O Board (see Fig 3-7 and Fig 3-8b2).
12. See Fig 3-7. Install the four No.4-40 x 1/4" flat head Phillips screws securing the Interface Boards of the new I/O Board to the front panel
13. Retighten the three No.4-40 x 1/4" Phillips flat head mounting screws securing the Control / Timing Board to the module front panel that were loosened in step 8. Also retighten the two small slotted head retainer screws securing the module latches to the module that were loosened in step 8.
14. Reinstall the module cover. Reinstall and tighten the 14 mounting screws securing the cover to the module, see Fig 3-6.

Procedure ... Install I/O Bd No.3

The procedure for installing I/O Board No.3 is, essentially, the same as that for installing I/O Board No.2, except for the arrangement of the spacer stacks at positions 1-3 and position 4, as depicted in Figures 3a and 3b, respectively. Use these figures as a guideline to ensure correct I/O Board spacing inside the module.

Note when installing I/O Boards that the power supplied to connector J6 of I/O Board No.1 is taken from the Power Interface Board (see lower part of Fig 3-7), while power supplied to the same connector (J6) of I/O

Boards No.2 and No.3 is taken from the Aux Power Cable going to the Aux Power connector on the module front panel. If I/O Boards No.2 and/or No.3 are not used, the unused connectors of the Aux Power Cable will be tied off and tucked loosely inside the module.

Also note that the ribbon cable (upper part of Fig 3-7) has a separate connector for each of the I/O Boards. Unused connectors on the ribbon cable are left unterminated.

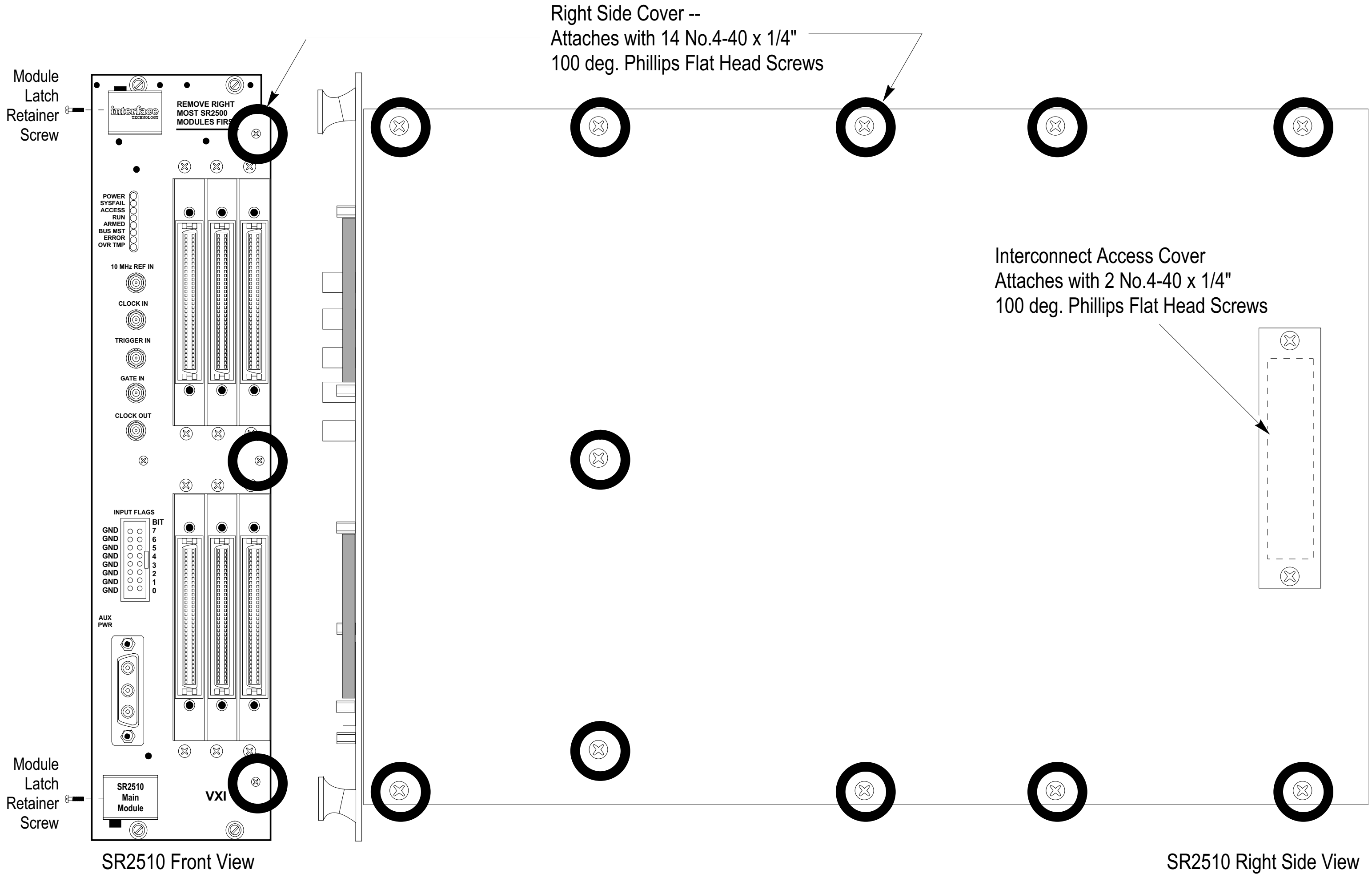

Right Side Cover --
Attaches with 14 No.4-40 x 1/4"
100 deg. Phillips Flat Head Screws
Module Latch Retainer Screw
REMOVE RIGHT MOST SR2500 MODULES FIRS
interface TECHNOLOGY
POWER
SYSFAIL
ACCESS
RUN
ARMED
BUS MST
ERROR
OVR TMP
10 MHz REF IN
CLOCK IN
TRIGGER IN
GATE IN
CLOCK OUT
INPUT FLAGS
BIT
GND
7
6
5
4
3
2
1
0
AUX PWR
SR2510 Main Module
VXI
Module Latch Retainer Screw
Interconnect Access Cover
Attaches with 2 No.4-40 x 1/4"
100 deg. Phillips Flat Head Screws
SR2510 Front View
SR2510 Right Side View

Figure 3-6. Cover Screws

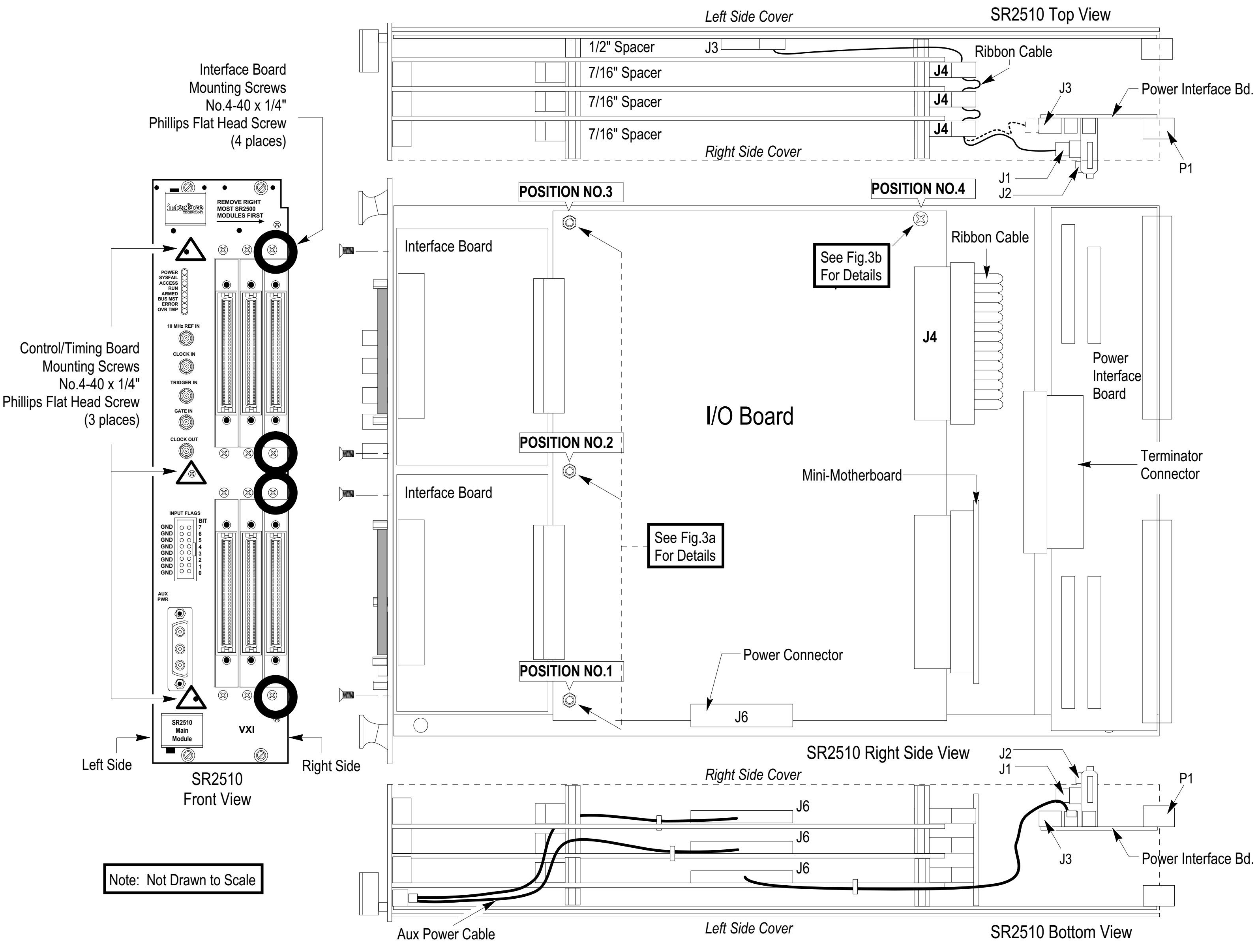
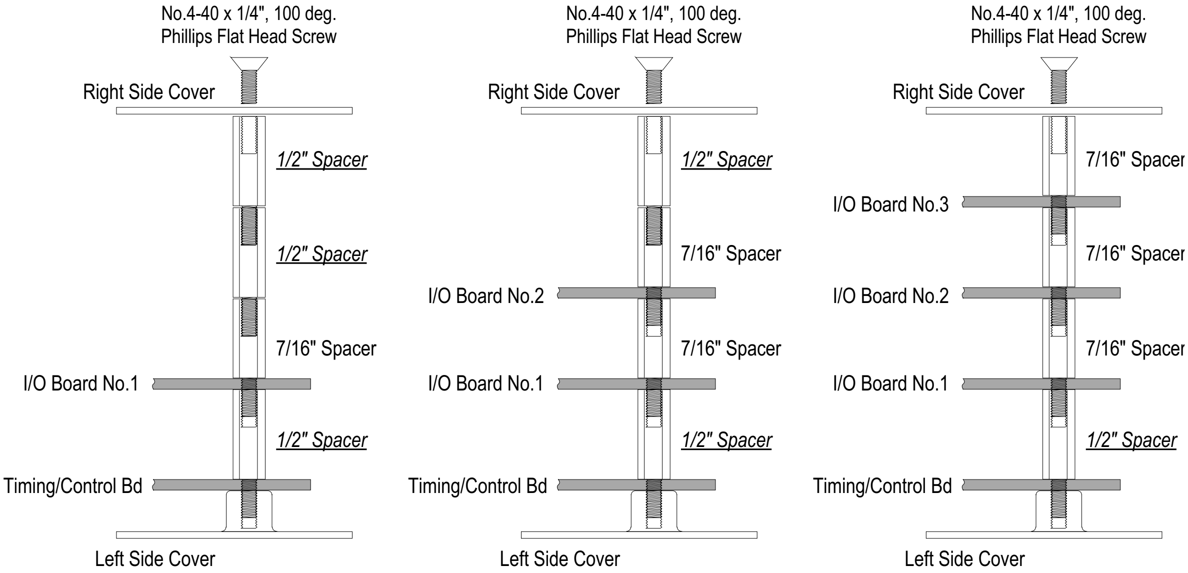
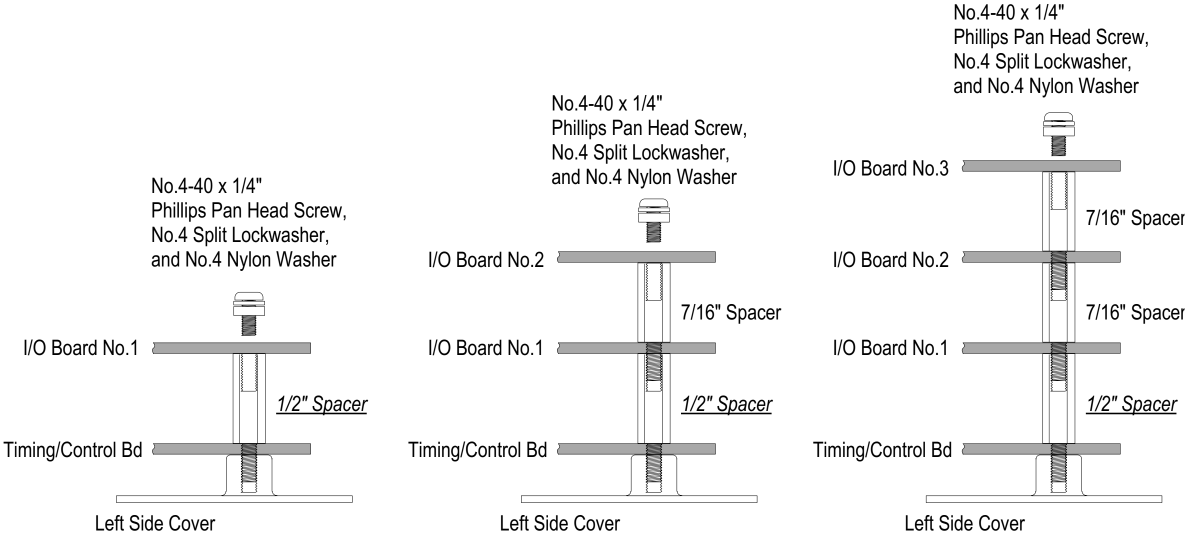


Figure 3-7. I/O Board Mounting Hardware.



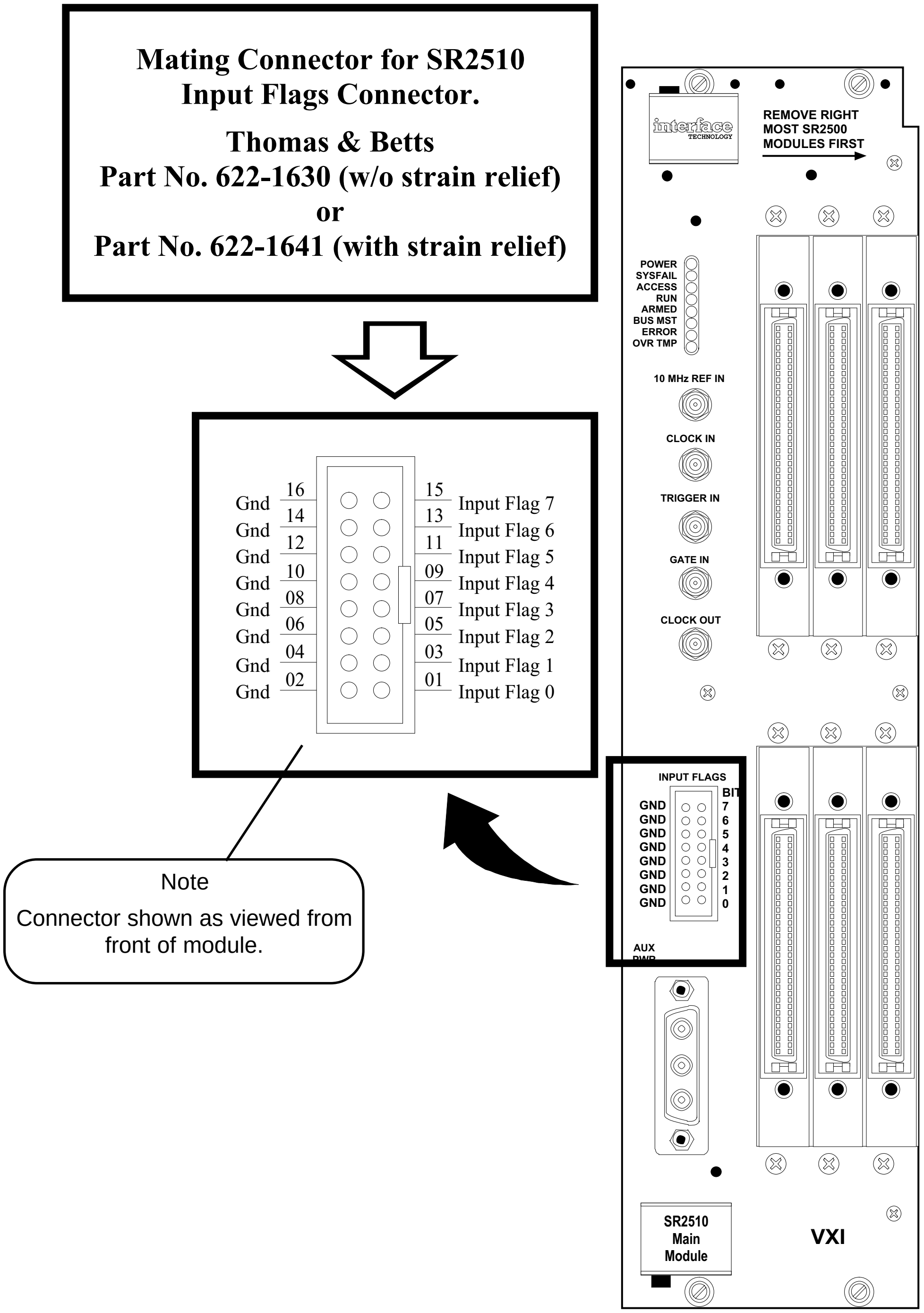
1. One I/O Board. *2. Two I/O Boards.* *3. Three I/O Boards.*

Figure 3-8a.
Buildup of Standoff Spacers at Positions 1-3 for Configurations of One, Two, and Three I/O Boards.



1. One I/O Board. *2. Two I/O Boards.* *3. Three I/O Boards.*

Figure 3-8b.
Buildup of Standoff Spacers at Position 4 for Configurations of One, Two, and Three I/O Boards.



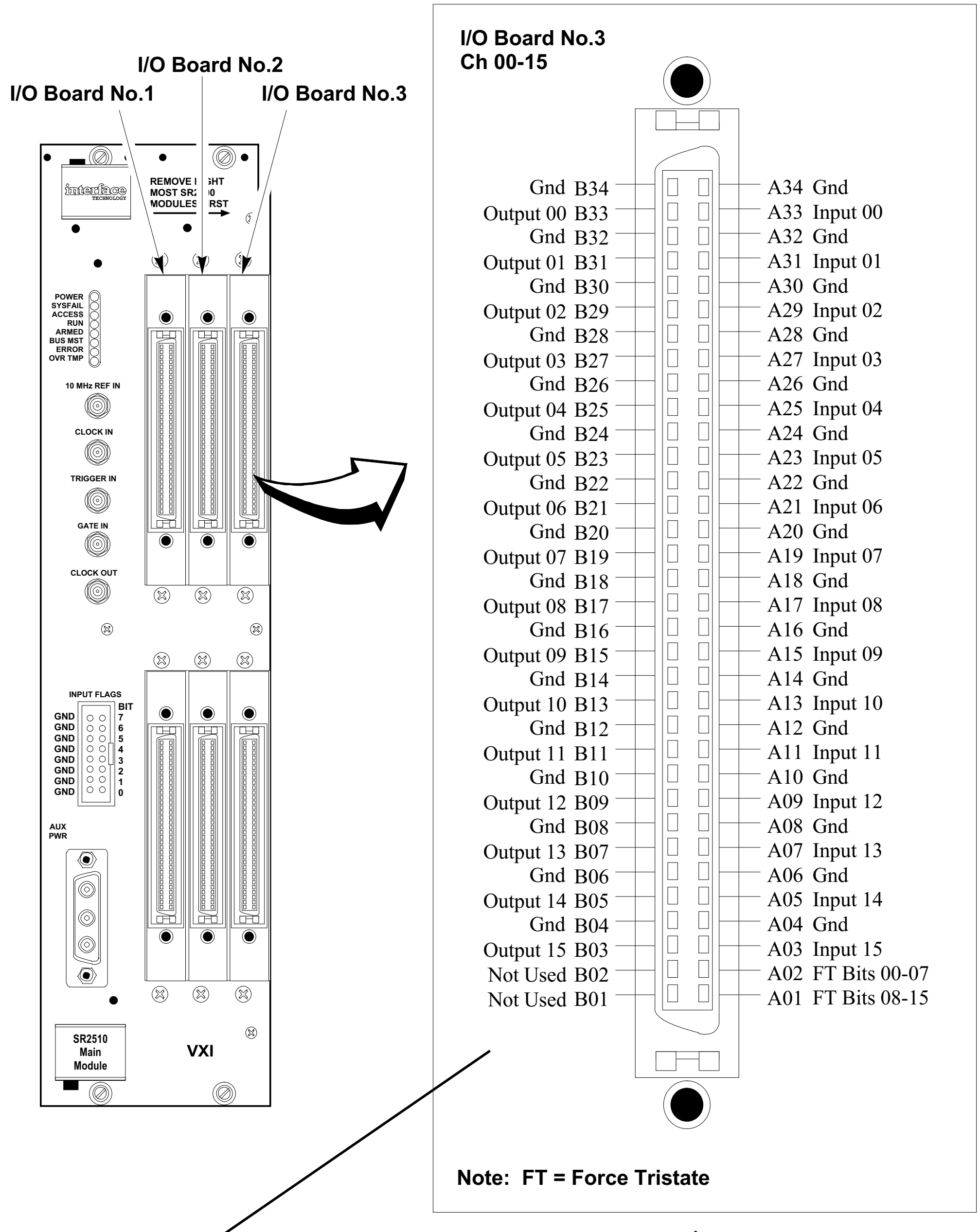
Mating Connector for SR2510 Input Flags Connector.

Thomas & Betts
Part No. 622-1630 (w/o strain relief)
or
Part No. 622-1641 (with strain relief)

Gnd	16	15	Input Flag 7
Gnd	14	13	Input Flag 6
Gnd	12	11	Input Flag 5
Gnd	10	09	Input Flag 4
Gnd	08	07	Input Flag 3
Gnd	06	05	Input Flag 2
Gnd	04	03	Input Flag 1
Gnd	02	01	Input Flag 0

Note
Connector shown as viewed from front of module.

Figure 3-9. SR2510 Input Flags Connector.



Gnd	B34	A34	Gnd
Output 00	B33	A33	Input 00
Gnd	B32	A32	Gnd
Output 01	B31	A31	Input 01
Gnd	B30	A30	Gnd
Output 02	B29	A29	Input 02
Gnd	B28	A28	Gnd
Output 03	B27	A27	Input 03
Gnd	B26	A26	Gnd
Output 04	B25	A25	Input 04
Gnd	B24	A24	Gnd
Output 05	B23	A23	Input 05
Gnd	B22	A22	Gnd
Output 06	B21	A21	Input 06
Gnd	B20	A20	Gnd
Output 07	B19	A19	Input 07
Gnd	B18	A18	Gnd
Output 08	B17	A17	Input 08
Gnd	B16	A16	Gnd
Output 09	B15	A15	Input 09
Gnd	B14	A14	Gnd
Output 10	B13	A13	Input 10
Gnd	B12	A12	Gnd
Output 11	B11	A11	Input 11
Gnd	B10	A10	Gnd
Output 12	B09	A09	Input 12
Gnd	B08	A08	Gnd
Output 13	B07	A07	Input 13
Gnd	B06	A06	Gnd
Output 14	B05	A05	Input 14
Gnd	B04	A04	Gnd
Output 15	B03	A03	Input 15
Not Used	B02	A02	FT Bits 00-07
Not Used	B01	A01	FT Bits 08-15

Note: FT = Force Tristate

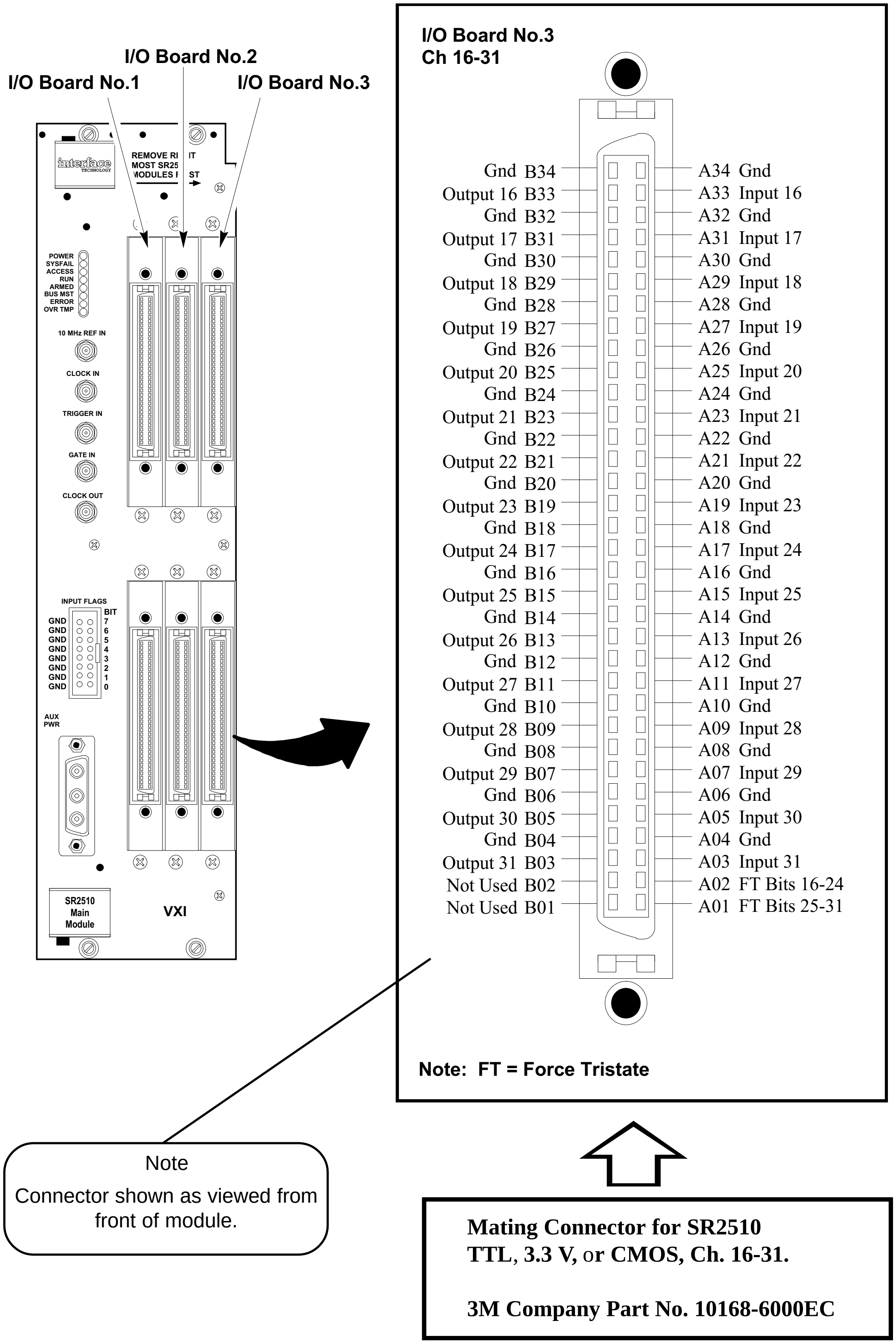
Note

Connector shown as viewed from front of module.

Mating Connector for SR2510
TTL, 3.3 V, or CMOS, Ch. 00-15.

3M Company Part No. 10168-6000EC

Figure 3-10. SR2510 Signal Connector Pinouts, TTL, 3.3 V, or CMOS, Ch. 00-15.



Gnd	B34	A34	Gnd
Output 16	B33	A33	Input 16
Gnd	B32	A32	Gnd
Output 17	B31	A31	Input 17
Gnd	B30	A30	Gnd
Output 18	B29	A29	Input 18
Gnd	B28	A28	Gnd
Output 19	B27	A27	Input 19
Gnd	B26	A26	Gnd
Output 20	B25	A25	Input 20
Gnd	B24	A24	Gnd
Output 21	B23	A23	Input 21
Gnd	B22	A22	Gnd
Output 22	B21	A21	Input 22
Gnd	B20	A20	Gnd
Output 23	B19	A19	Input 23
Gnd	B18	A18	Gnd
Output 24	B17	A17	Input 24
Gnd	B16	A16	Gnd
Output 25	B15	A15	Input 25
Gnd	B14	A14	Gnd
Output 26	B13	A13	Input 26
Gnd	B12	A12	Gnd
Output 27	B11	A11	Input 27
Gnd	B10	A10	Gnd
Output 28	B09	A09	Input 28
Gnd	B08	A08	Gnd
Output 29	B07	A07	Input 29
Gnd	B06	A06	Gnd
Output 30	B05	A05	Input 30
Gnd	B04	A04	Gnd
Output 31	B03	A03	Input 31
Not Used	B02	A02	FT Bits 16-24
Not Used	B01	A01	FT Bits 25-31

Note: FT = Force Tristate

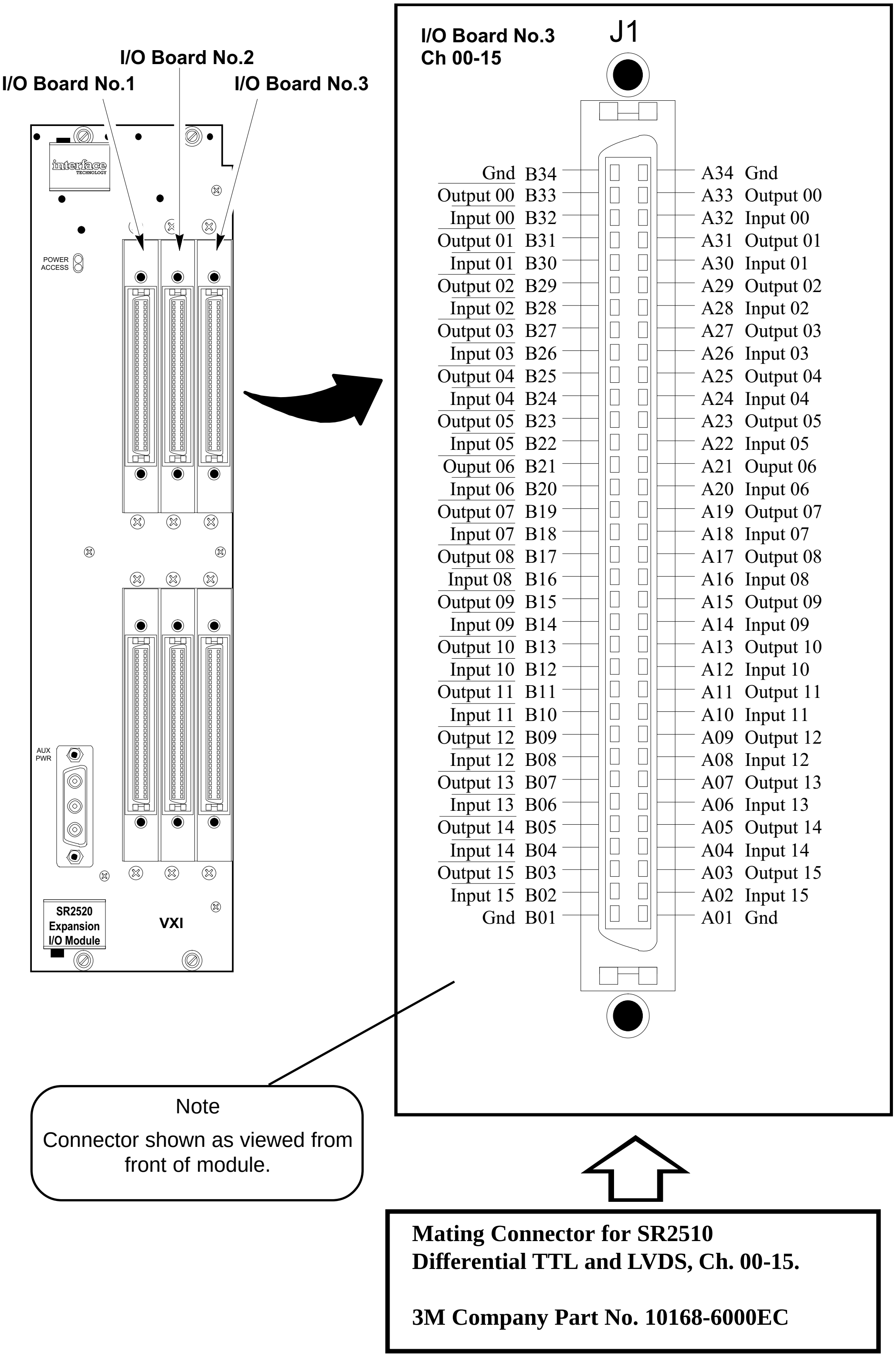
Note

Connector shown as viewed from front of module.

Mating Connector for SR2510
TTL, 3.3 V, or CMOS, Ch. 16-31.

3M Company Part No. 10168-6000EC

Figure 3-11. SR2510 Signal Connector Pinouts, TTL, 3.3 V, or CMOS, Ch. 16-31.



Gnd	B34	A34	Gnd
Output 00	B33	A33	Output 00
Input 00	B32	A32	Input 00
Output 01	B31	A31	Output 01
Input 01	B30	A30	Input 01
Output 02	B29	A29	Output 02
Input 02	B28	A28	Input 02
Output 03	B27	A27	Output 03
Input 03	B26	A26	Input 03
Output 04	B25	A25	Output 04
Input 04	B24	A24	Input 04
Output 05	B23	A23	Output 05
Input 05	B22	A22	Input 05
Ouput 06	B21	A21	Ouput 06
Input 06	B20	A20	Input 06
Output 07	B19	A19	Output 07
Input 07	B18	A18	Input 07
Output 08	B17	A17	Output 08
Input 08	B16	A16	Input 08
Output 09	B15	A15	Output 09
Input 09	B14	A14	Input 09
Output 10	B13	A13	Output 10
Input 10	B12	A12	Input 10
Output 11	B11	A11	Output 11
Input 11	B10	A10	Input 11
Output 12	B09	A09	Output 12
Input 12	B08	A08	Input 12
Output 13	B07	A07	Output 13
Input 13	B06	A06	Input 13
Output 14	B05	A05	Output 14
Input 14	B04	A04	Input 14
Output 15	B03	A03	Output 15
Input 15	B02	A02	Input 15
Gnd	B01	A01	Gnd

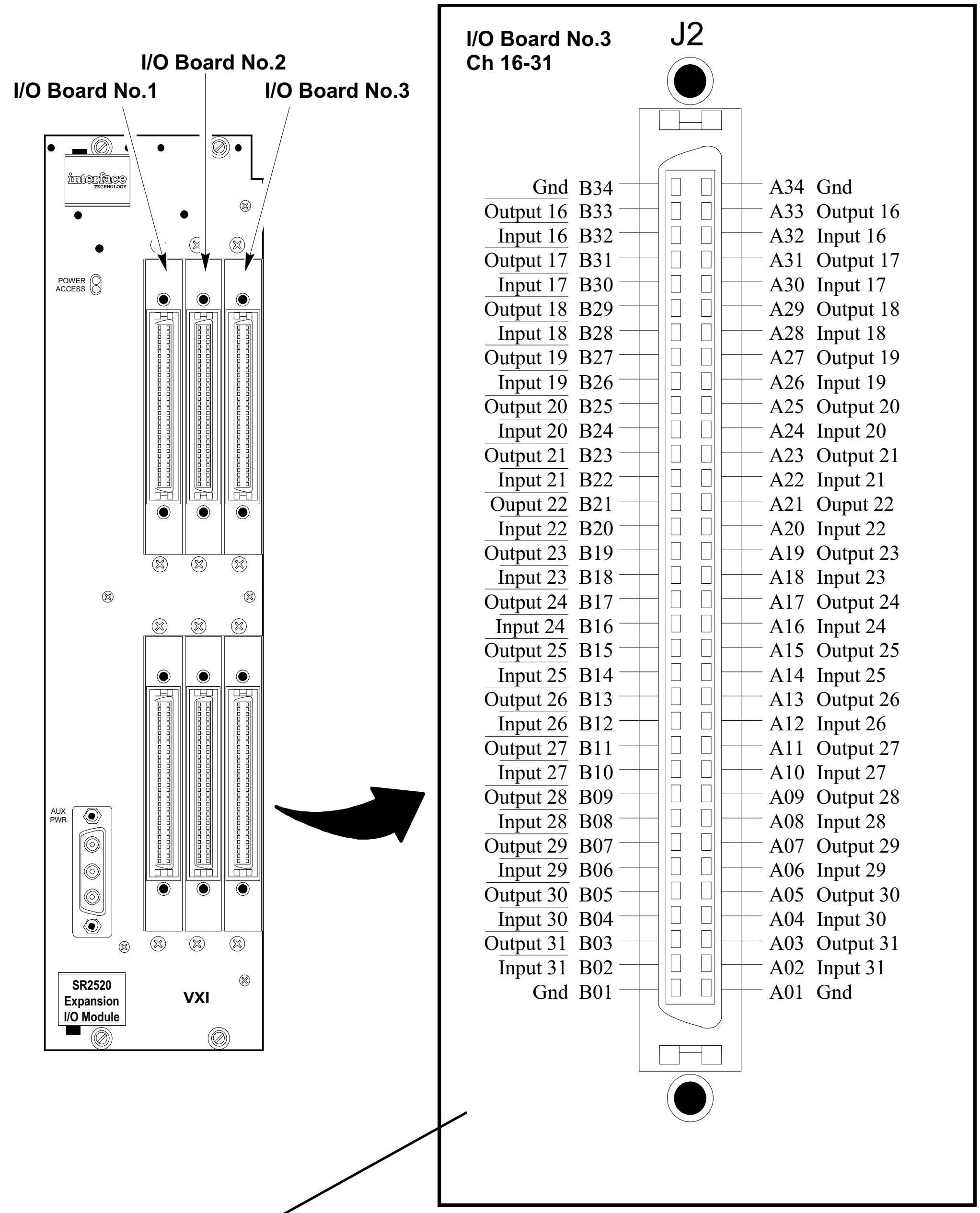
Note

Connector shown as viewed from front of module.

Mating Connector for SR2510 Differential TTL and LVDS, Ch. 00-15.

3M Company Part No. 10168-6000EC

Figure 3-12. SR2510 Signal Connector Pinouts, Differential TTL and LVDS, Ch. 00-15.



Gnd	B34	A34	Gnd
Output 16	B33	A33	Output 16
Input 16	B32	A32	Input 16
Output 17	B31	A31	Output 17
Input 17	B30	A30	Input 17
Output 18	B29	A29	Output 18
Input 18	B28	A28	Input 18
Output 19	B27	A27	Output 19
Input 19	B26	A26	Input 19
Output 20	B25	A25	Output 20
Input 20	B24	A24	Input 20
Output 21	B23	A23	Output 21
Input 21	B22	A22	Input 21
Ouput 22	B21	A21	Ouput 22
Input 22	B20	A20	Input 22
Output 23	B19	A19	Output 23
Input 23	B18	A18	Input 23
Output 24	B17	A17	Output 24
Input 24	B16	A16	Input 24
Output 25	B15	A15	Output 25
Input 25	B14	A14	Input 25
Output 26	B13	A13	Output 26
Input 26	B12	A12	Input 26
Output 27	B11	A11	Output 27
Input 27	B10	A10	Input 27
Output 28	B09	A09	Output 28
Input 28	B08	A08	Input 28
Output 29	B07	A07	Output 29
Input 29	B06	A06	Input 29
Output 30	B05	A05	Output 30
Input 30	B04	A04	Input 30
Output 31	B03	A03	Output 31
Input 31	B02	A02	Input 31
Gnd	B01	A01	Gnd

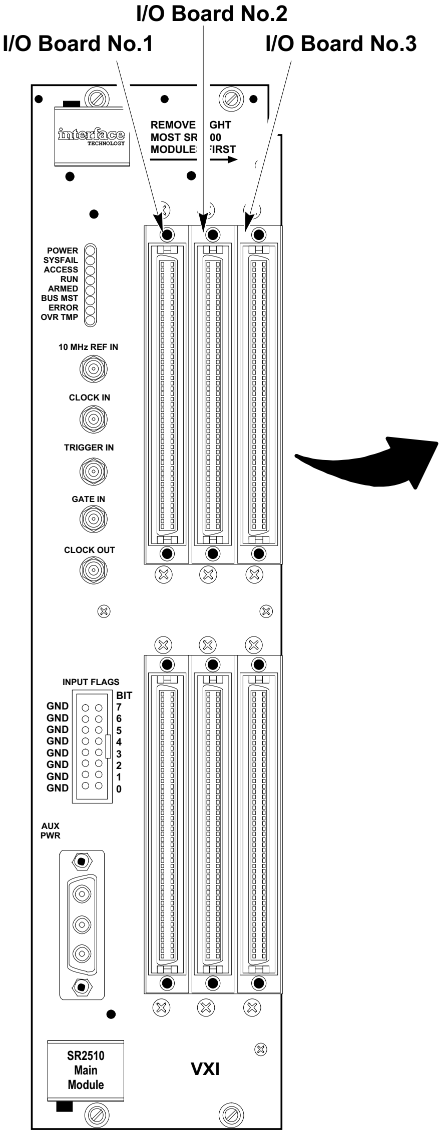
Note

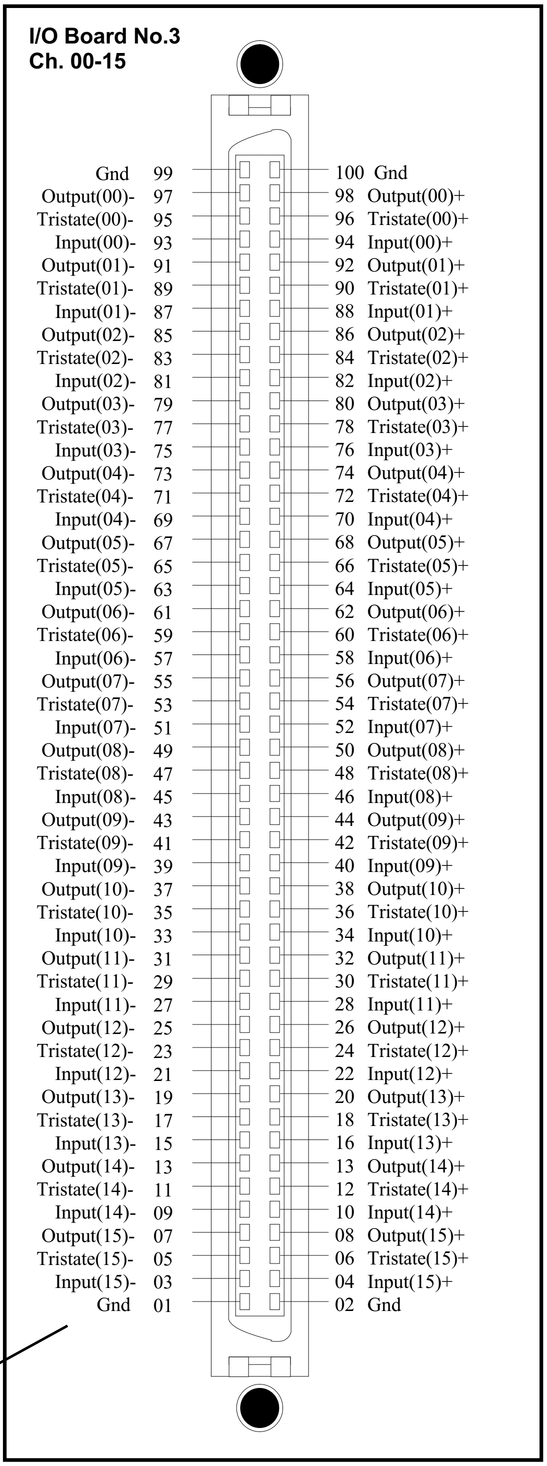
Connector shown as viewed from front of module.

Mating Connector for SR2510 Differential TTL and LVDS, Ch. 16-31.

3M Company Part No. 10168-6000EC

Figure 3-13. SR2510 Signal Connector Pinouts, Differential TTL and LVDS, Ch. 16-31.





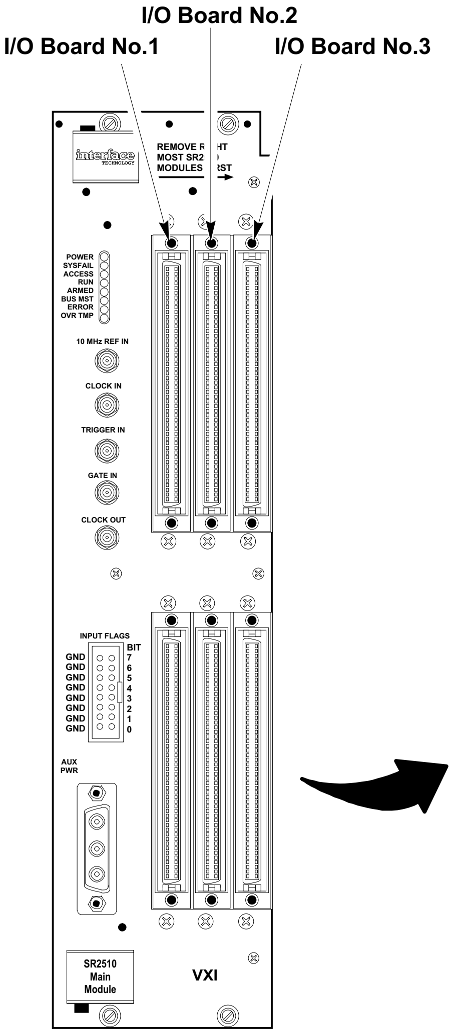
Gnd	99	100	Gnd
Output(00)-	97	98	Output(00)+
Tristate(00)-	95	96	Tristate(00)+
Input(00)-	93	94	Input(00)+
Output(01)-	91	92	Output(01)+
Tristate(01)-	89	90	Tristate(01)+
Input(01)-	87	88	Input(01)+
Output(02)-	85	86	Output(02)+
Tristate(02)-	83	84	Tristate(02)+
Input(02)-	81	82	Input(02)+
Output(03)-	79	80	Output(03)+
Tristate(03)-	77	78	Tristate(03)+
Input(03)-	75	76	Input(03)+
Output(04)-	73	74	Output(04)+
Tristate(04)-	71	72	Tristate(04)+
Input(04)-	69	70	Input(04)+
Output(05)-	67	68	Output(05)+
Tristate(05)-	65	66	Tristate(05)+
Input(05)-	63	64	Input(05)+
Output(06)-	61	62	Output(06)+
Tristate(06)-	59	60	Tristate(06)+
Input(06)-	57	58	Input(06)+
Output(07)-	55	56	Output(07)+
Tristate(07)-	53	54	Tristate(07)+
Input(07)-	51	52	Input(07)+
Output(08)-	49	50	Output(08)+
Tristate(08)-	47	48	Tristate(08)+
Input(08)-	45	46	Input(08)+
Output(09)-	43	44	Output(09)+
Tristate(09)-	41	42	Tristate(09)+
Input(09)-	39	40	Input(09)+
Output(10)-	37	38	Output(10)+
Tristate(10)-	35	36	Tristate(10)+
Input(10)-	33	34	Input(10)+
Output(11)-	31	32	Output(11)+
Tristate(11)-	29	30	Tristate(11)+
Input(11)-	27	28	Input(11)+
Output(12)-	25	26	Output(12)+
Tristate(12)-	23	24	Tristate(12)+
Input(12)-	21	22	Input(12)+
Output(13)-	19	20	Output(13)+
Tristate(13)-	17	18	Tristate(13)+
Input(13)-	15	16	Input(13)+
Output(14)-	13	13	Output(14)+
Tristate(14)-	11	12	Tristate(14)+
Input(14)-	09	10	Input(14)+
Output(15)-	07	08	Output(15)+
Tristate(15)-	05	06	Tristate(15)+
Input(15)-	03	04	Input(15)+
Gnd	01	02	Gnd

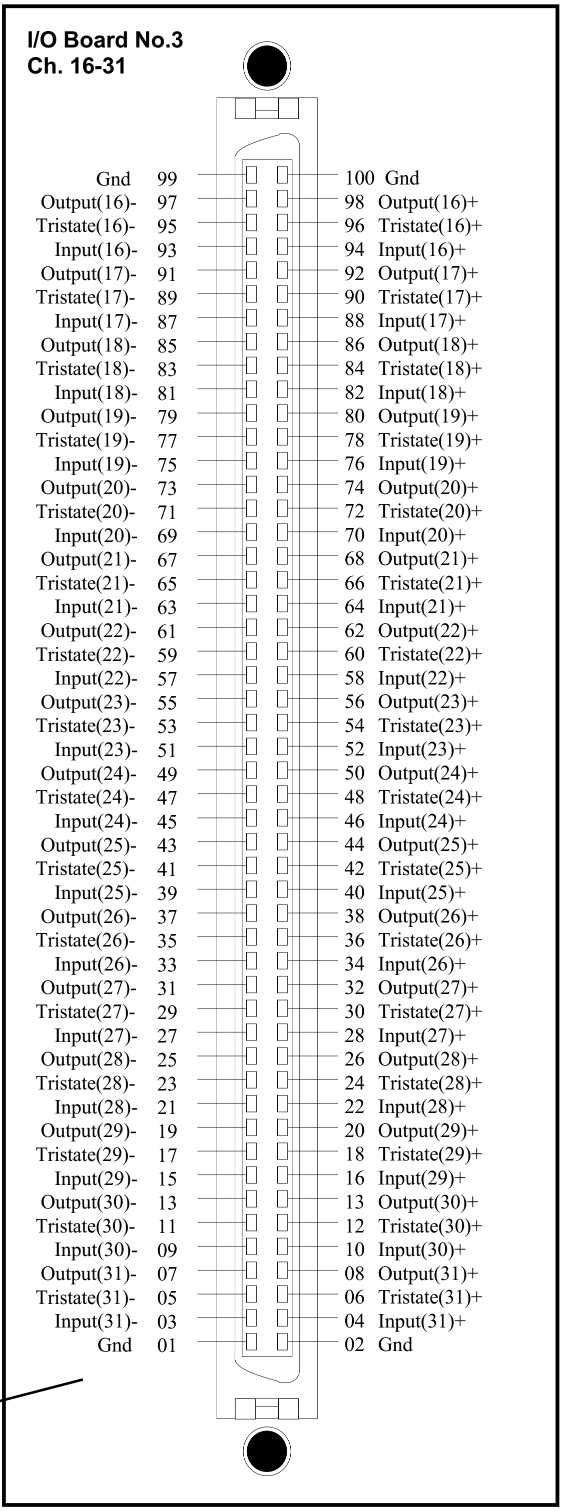
Mating Connector for SR2510
ECL I/O Connector, Ch. 00-15.

3M Company Part No. 101AO-6000EC

Note
Connector shown as viewed from front of module.

Figure 3-14. SR2510 Signal Connector Pinouts, Differential ECL, Ch. 00-15.





I/O Board No.3
Ch. 16-31

Gnd	99	100	Gnd
Output(16)-	97	98	Output(16)+
Tristate(16)-	95	96	Tristate(16)+
Input(16)-	93	94	Input(16)+
Output(17)-	91	92	Output(17)+
Tristate(17)-	89	90	Tristate(17)+
Input(17)-	87	88	Input(17)+
Output(18)-	85	86	Output(18)+
Tristate(18)-	83	84	Tristate(18)+
Input(18)-	81	82	Input(18)+
Output(19)-	79	80	Output(19)+
Tristate(19)-	77	78	Tristate(19)+
Input(19)-	75	76	Input(19)+
Output(20)-	73	74	Output(20)+
Tristate(20)-	71	72	Tristate(20)+
Input(20)-	69	70	Input(20)+
Output(21)-	67	68	Output(21)+
Tristate(21)-	65	66	Tristate(21)+
Input(21)-	63	64	Input(21)+
Output(22)-	61	62	Output(22)+
Tristate(22)-	59	60	Tristate(22)+
Input(22)-	57	58	Input(22)+
Output(23)-	55	56	Output(23)+
Tristate(23)-	53	54	Tristate(23)+
Input(23)-	51	52	Input(23)+
Output(24)-	49	50	Output(24)+
Tristate(24)-	47	48	Tristate(24)+
Input(24)-	45	46	Input(24)+
Output(25)-	43	44	Output(25)+
Tristate(25)-	41	42	Tristate(25)+
Input(25)-	39	40	Input(25)+
Output(26)-	37	38	Output(26)+
Tristate(26)-	35	36	Tristate(26)+
Input(26)-	33	34	Input(26)+
Output(27)-	31	32	Output(27)+
Tristate(27)-	29	30	Tristate(27)+
Input(27)-	27	28	Input(27)+
Output(28)-	25	26	Output(28)+
Tristate(28)-	23	24	Tristate(28)+
Input(28)-	21	22	Input(28)+
Output(29)-	19	20	Output(29)+
Tristate(29)-	17	18	Tristate(29)+
Input(29)-	15	16	Input(29)+
Output(30)-	13	13	Output(30)+
Tristate(30)-	11	12	Tristate(30)+
Input(30)-	09	10	Input(30)+
Output(31)-	07	08	Output(31)+
Tristate(31)-	05	06	Tristate(31)+
Input(31)-	03	04	Input(31)+
Gnd	01	02	Gnd

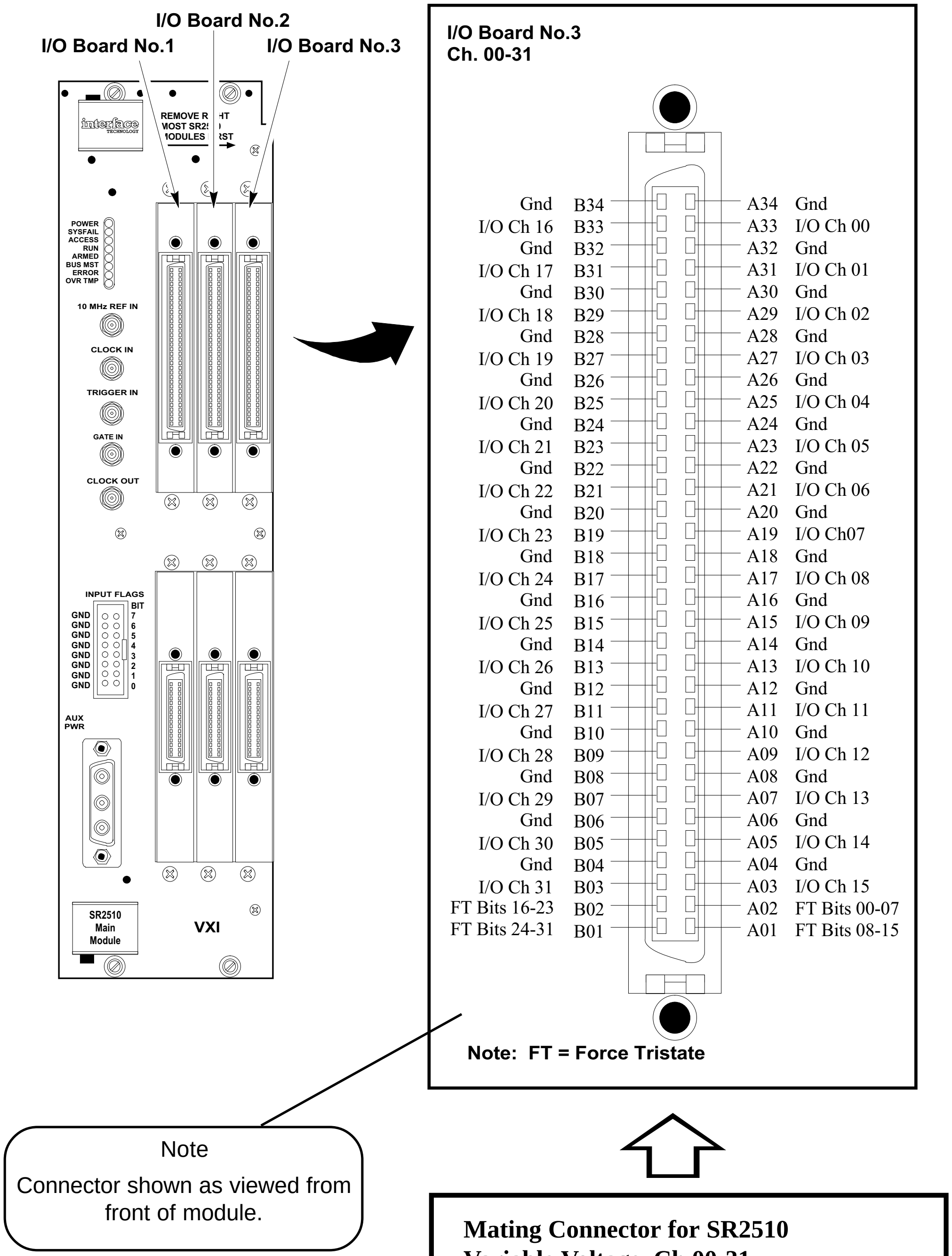
Mating Connector for SR2510 ECL I/O Connector, Ch. 16-31.

3M Company Part No. 101AO-6000EC

Note

Connector shown as viewed from front of module.

Figure 3-15. SR2510 Signal Connector Pinouts, Differential ECL, Ch. 16-31.



Gnd	B34	A34	Gnd
I/O Ch 16	B33	A33	I/O Ch 00
Gnd	B32	A32	Gnd
I/O Ch 17	B31	A31	I/O Ch 01
Gnd	B30	A30	Gnd
I/O Ch 18	B29	A29	I/O Ch 02
Gnd	B28	A28	Gnd
I/O Ch 19	B27	A27	I/O Ch 03
Gnd	B26	A26	Gnd
I/O Ch 20	B25	A25	I/O Ch 04
Gnd	B24	A24	Gnd
I/O Ch 21	B23	A23	I/O Ch 05
Gnd	B22	A22	Gnd
I/O Ch 22	B21	A21	I/O Ch 06
Gnd	B20	A20	Gnd
I/O Ch 23	B19	A19	I/O Ch07
Gnd	B18	A18	Gnd
I/O Ch 24	B17	A17	I/O Ch 08
Gnd	B16	A16	Gnd
I/O Ch 25	B15	A15	I/O Ch 09
Gnd	B14	A14	Gnd
I/O Ch 26	B13	A13	I/O Ch 10
Gnd	B12	A12	Gnd
I/O Ch 27	B11	A11	I/O Ch 11
Gnd	B10	A10	Gnd
I/O Ch 28	B09	A09	I/O Ch 12
Gnd	B08	A08	Gnd
I/O Ch 29	B07	A07	I/O Ch 13
Gnd	B06	A06	Gnd
I/O Ch 30	B05	A05	I/O Ch 14
Gnd	B04	A04	Gnd
I/O Ch 31	B03	A03	I/O Ch 15
FT Bits 16-23	B02	A02	FT Bits 00-07
FT Bits 24-31	B01	A01	FT Bits 08-15

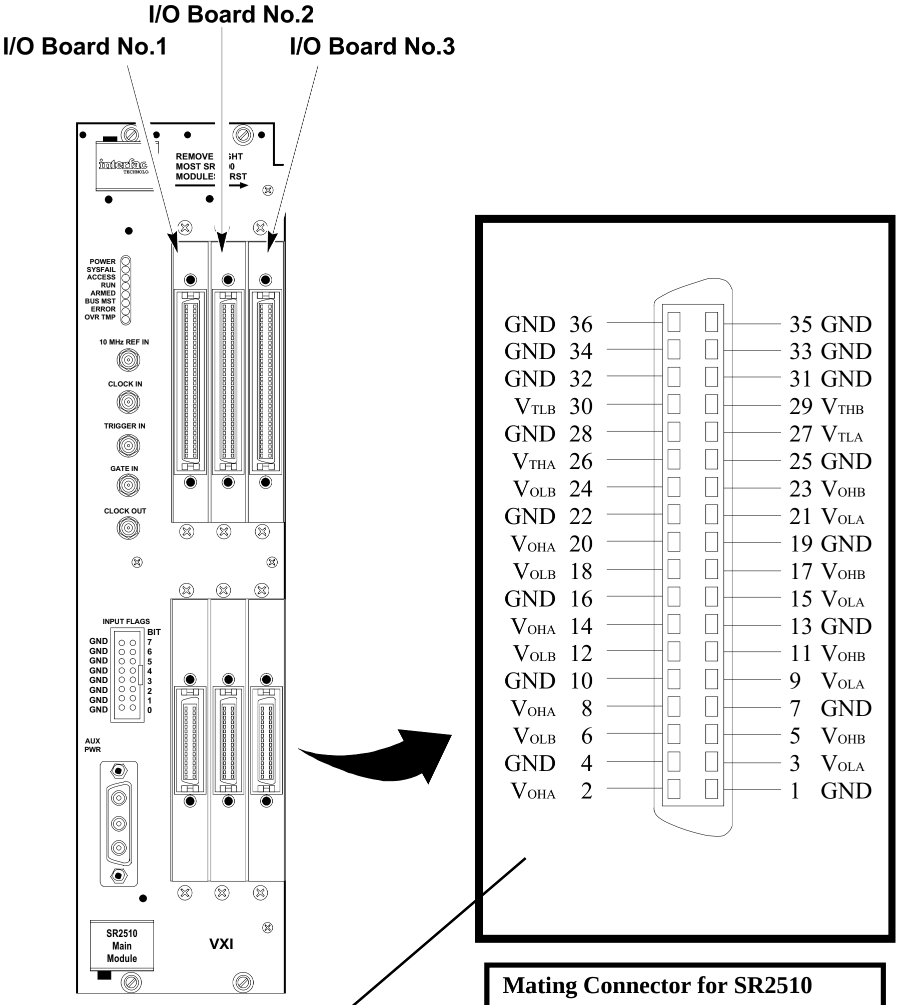
Note: FT = Force Tristate

Note
Connector shown as viewed from front of module.

Mating Connector for SR2510 Variable Voltage, Ch 00-31.

3M Company Part No. 10168-6000EC

Figure 3-16. SR2510 Signal Connector Pinouts, Variable Voltage, Ch. 00-31.



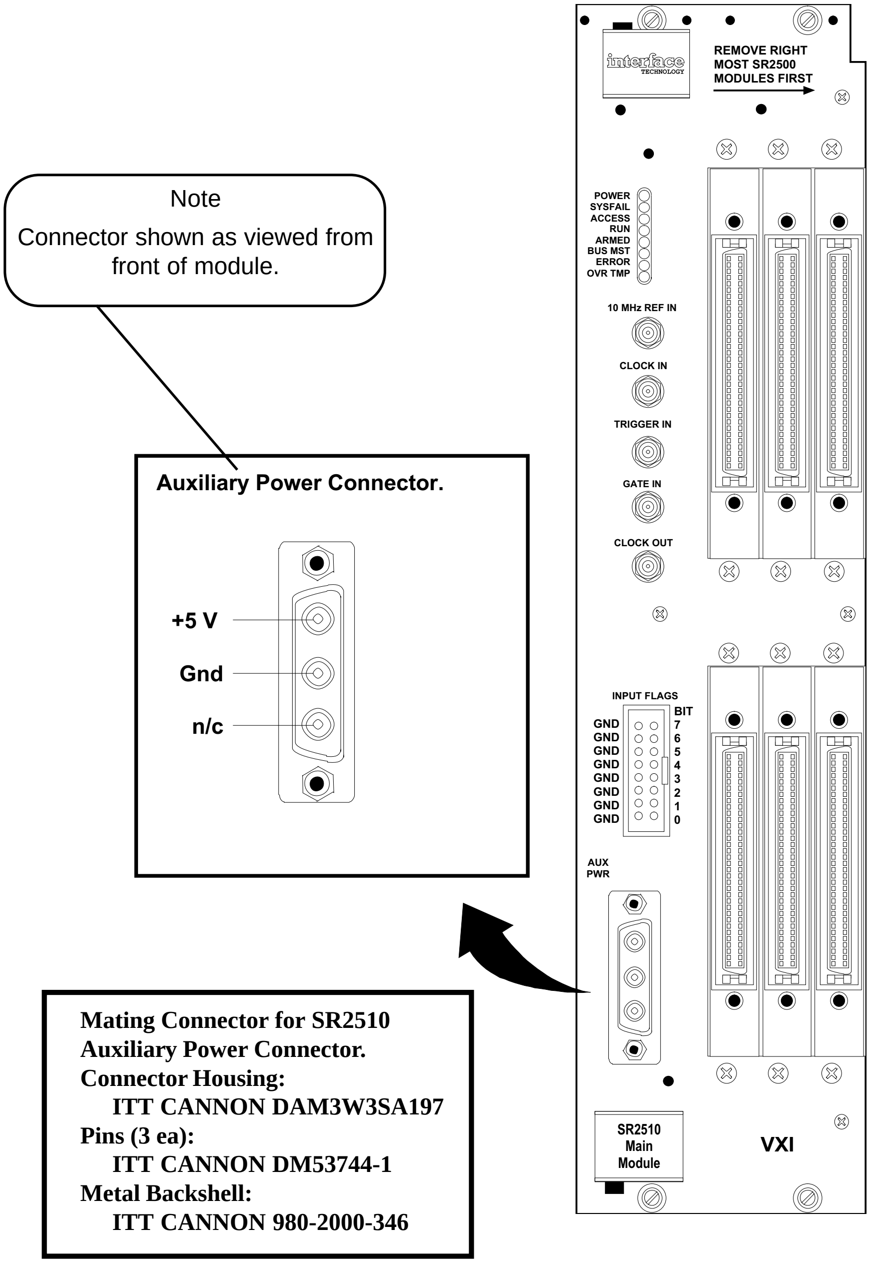
GND	36	35	GND
GND	34	33	GND
GND	32	31	GND
VTLB	30	29	VTHB
GND	28	27	VTLA
VTHA	26	25	GND
VOLB	24	23	VOHB
GND	22	21	VOLA
VOHA	20	19	GND
VOLB	18	17	VOHB
GND	16	15	VOLA
VOHA	14	13	GND
VOLB	12	11	VOHB
GND	10	9	VOLA
VOHA	8	7	GND
VOLB	6	5	VOHB
GND	4	3	VOLA
VOHA	2	1	GND

Mating Connector for SR2510
Rail Voltage Connector.
3M Company
Part No. 10136-6000EC

Note

Connector shown as viewed from front of module.

Figure 3-17. SR2510 Rail Voltage Connector Pinouts.



Note

Connector shown as viewed from front of module.

Mating Connector for SR2510 Auxiliary Power Connector.
Connector Housing:
ITT CANNON DAM3W3SA197
Pins (3 ea):
ITT CANNON DM53744-1
Metal Backshell:
ITT CANNON 980-2000-346

Figure 3-18. SR2510 Auxiliary Power Connector Pinouts.

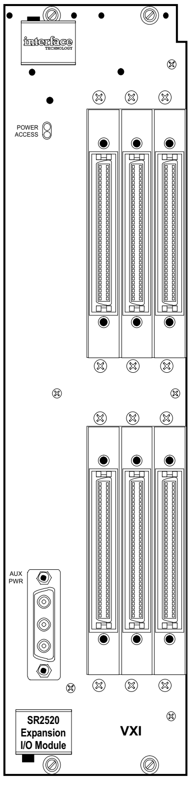
Published by
Interface Technology, Inc.
300 S. Lemon Creek Drive
Walnut, CA 91789

User's Manual

SR2520 Expansion Module

VXI bus

From The Performance Leader In VXI Digital Testing ...



VXI plug & play

interface
TECHNOLOGY

SR2520 User's Manual

Record of Changes

Change No.	Date of Change	Title or Brief Description	Entered By
Rev 05	Apr 98	Reformat	Factory
Change 1	Mar 00	Revised external power supply info (pg 3-3); added pinout data for differential TTL (pg 3-13, 3-14)	Factory
Change 2	Mar 00	Added coverage for differential TTL (pg 2-10 - 2-12).	Factory
Change 3	Jun 00	Added LVDS I/O (pgs 1-6, 1-7, 1-8, 2-11, 2-12, 3-13, 3-14)	Factory
Change 4	Oct 00	Corrected connector orientation in Figs 3-8 thru 3-16 and added note explaining relationship of pinout views to instrument illustrations. Added power sequencing note to page 3-3.	Factory
Change 5	Jun 01	Corrected paragraph wording for LVDS (pg 2-12)	Factory
Change 6	Sep 01	Reformat specifications page, pg 1-6. Deleted pgs 1-7 and 1-8.	Factory
Change 7	May 03	pg 2-11 ... 1st para., lines 2 and 3, changed "... -4V to +7V" to "... -3V to +7V"; changed "... -4 to +5.5V" to "...-2.9 to +5.5V."	Factory
Change 8	May 03	Corrected connector pinouts in Fig 3-14 (pins B01 and B02).	Factory
Change 9	Oct 03	Corrected I/O Characteristics table on pg. 1-6. Corrected Figures 2-4 and 2-5; added Fig 2-10. Added pgs 2-13 (3.3 V I/O) and 2-14 (blank). Updated Fig 3-8 and 3-9 to include 3.3 V I/O.	Factory

Contents

Chapter 1
General Information

About This Manual 1-1
Arrangement of Contents 1-1
Applicability 1-1
Supersedure Notice 1-1
Equipment Description 1-1
Expansion Board 1-1
I/O Boards 1-2
VXI Bus Interface 1-2
Real Time Digital Testing 1-3
Controls and Indicators 1-4
LEDs 1-4
I/O Connections 1-4
Auxiliary Power Connector 1-4
Interconnection With Other SR2500 Modules 1-5
SR2520 Specifications 1-6

Chapter 2
Theory of Operation

I/O Board 2-1
Stimulus and Response Memory 2-2
Record Memory 2-2
Delayed Clock Generators 2-2
Stimulus Gate Arrays 2-3
Algorithmic Stimulus Pattern Generator 2-3
Stimulus Output Pin Formatter 2-4
Response Gate Arrays 2-4
Expected Response Pattern Generator 2-5
Response Input Formatter 2-6
Response Comparator 2-6
Input Qualifier 2-6
Record Control 2-6
CRC Logic 2-7
Algorithmic Commands 2-8
NONAlgorithmic 2-8
INCrement 2-8
DECrement 2-8
XOR 2-8
SLEFTZero 2-8
SLEFTOne 2-9
SLEFTComplement 2-9
RLEFT 2-9
SRIGHTZero 2-9

Contents (continued)

Driver/Receiver Board 2-9
TTL Driver/Receiver Logic 2-10
CMOS Driver/Receiver Logic 2-10
Differential ECL Driver/Receiver Logic 2-10
Programmable Driver/Receiver Logic 2-11

Chapter 3
Installation

Scope of Chapter 3-1
Unpacking and Inspection 3-1
Installation 3-1
Logical Addressing 3-1
Slot Dependency 3-2
Backplane Jumpers 3-2
5 Vdc External Power Requirements 3-2
Main and Expansion Module Interconnect 3-4
Installing I/O Boards 3-6
Required Equipment 3-6
Procedure ... Install I/O Bd No.2 3-6
Procedure ... Install I/O Bd No.3 3-7

List of Figures

Fig. 1-1, SR2520 Module With Three I/O Boards and Six Driver/Receiver Boards 1-2
Fig. 1-2, VXI Chassis Showing SR2510 Main Module and SR2520 Expansion I/O Module 1-3
Fig. 1-3, Connectors and Indicators 1-4
Fig. 1-4, SR2510 and SR2520 Interconnection, Top View 1-5
Fig. 2-1, SR2520 I/O Board Block Diagram 2-1
Fig. 2-2, Stimulus Gate Array Block Diagram 2-4
Fig. 2-3, Response Gate Array 2-5
Fig. 2-4, TTL Single Ended Driver/Receiver (16 per D/R Board) 2-11
Fig. 2-5 CMOS Single Ended Driver/Receiver (16 per D/R Board) 2-11
Fig. 2-6 Differential TTL Driver/Receiver (16 per D/R Board) 2-11
Fig. 2-7 Differential ECL Driver/Receiver (16 per D/R Board) 2-12
Fig. 2-8, Programmable Driver/Receiver (

Fig. 3-1, Address Switches Set to Logical Address 12(hex) 3-1
Fig. 3-2, Connection of External 5 Vdc Operating Power 3-3
Fig. 3-3, Interconnect Between SR2510 and SR2520, Top View 3-4
Fig. 3-4, SR2510 and SR2520 Interconnect Connectors 3-5
Fig. 3-5, Cover Screws 3-8
Fig. 3-6, I/O Board Mounting Hardware 3-9
Fig. 3-7a, Buildup of Standoff Spacers at Positions 1-3 for Configurations of One, Two, and Three I/O Boards 3-10
Fig. 3-7b, Buildup of Standoff Spacers at Position 4

Contents (continued)

Fig. 3-8, SR2520 Signal Connector Pinouts,
TTL or CMOS, Ch. 00-15 3-11
Fig. 3-9, SR2520 Signal Connector Pinouts,
TTL or CMOS, Ch. 16-31 3-12
Fig. 3-10, SR2520 Signal Connector Pinouts,
Differential ECL, Ch. 00-15 3-13
Fig. 3-11, SR2520 Signal Connector Pinouts,
Differential ECL, Ch. 16-31 3-14
Fig. 3-12, SR2520 Signal Connector Pinouts,
Variable Voltage, Ch. 00-31 3-15
Fig. 3-13, SR2510 Rail Voltage Connector Pinouts 3-16
Fig. 3-14, SR2520 Auxiliary Power Connector Pinouts 3-17

(THIS PAGE INTENTIONALLY LEFT BLANK)

CHAPTER 1

General Information

About This Manual

This manual provides installation and operation information for the Interface Technology SR2520 Expansion Module. Information contained herein is intended for use by technical personnel involved in the actual installation and operation of the subject instrument.

Arrangement of Contents

Information contained in this manual is arranged in three chapters, as follows:

- Chapter 1 General Information
- Chapter 2 Theory of Operation
- Chapter 3 Installation

Applicability

The information contained in this manual covers a single equipment configuration designated ***SR2520 Expansion Module***. Differences, if any, between this equipment and the actual equipment supplied are covered by Difference Data included at the front of this manual.

Supersedure Notice

This manual supersedes portions of SR2500 User's Manual, Rev.04 and all previous issues of that publication.

Equipment Description

See Fig 1-1. The SR2520 is an Expansion Module used in conjunction with the SR2510 Main Module, which together comprise the SR2500 Digital Test Subsystem. The major components of the SR2520 include an Expansion Board, one, two, or three I/O boards, and up to six Driver/ Receiver boards (2 per I/O board). Other components include boards for timing distribution, power distribution and interface logic.

Expansion Board.

The SR2520 Expansion Board accepts input from the SR2510 Main Module and distributes clocking and test sequence control functions for all I/O boards.

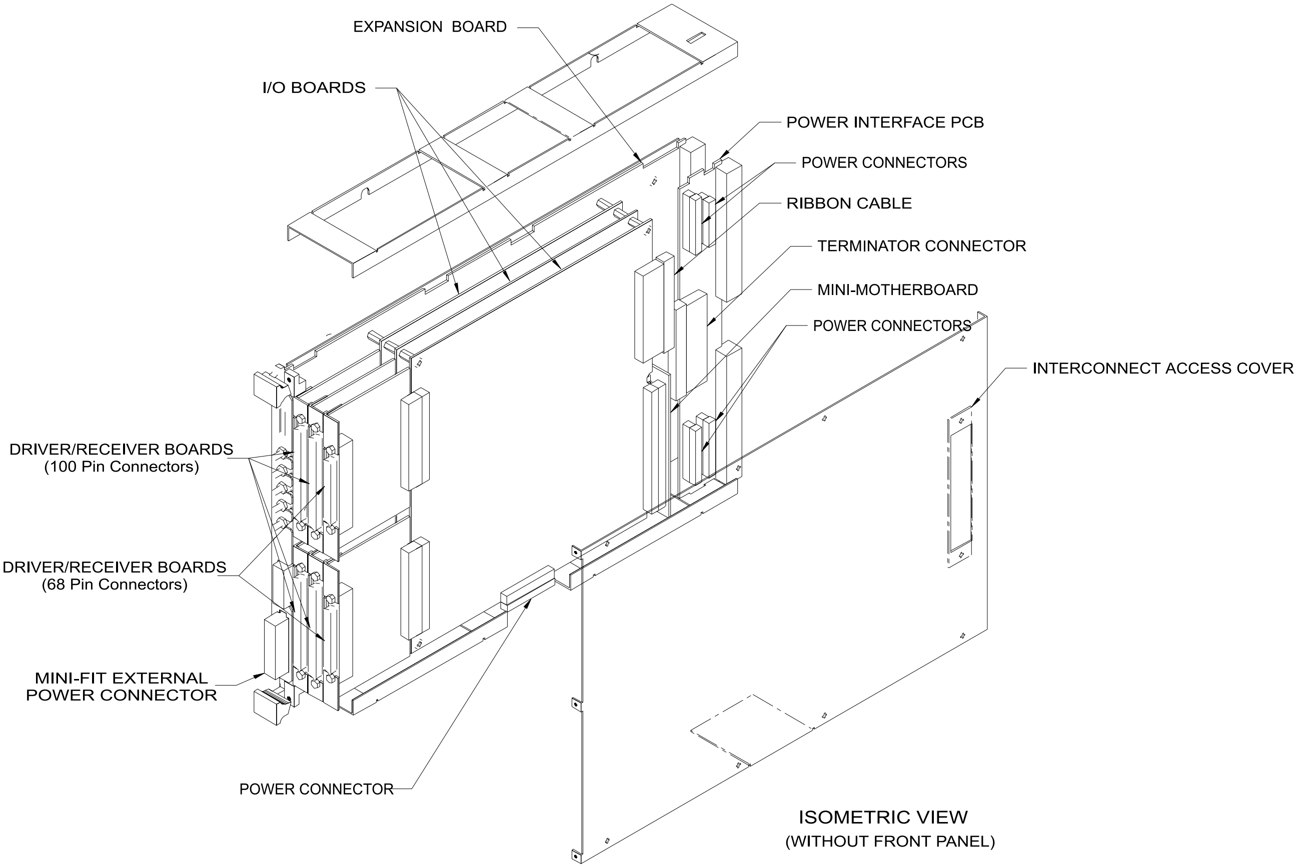

EXPANSION BOARD
I/O BOARDS
POWER INTERFACE PCB
POWER CONNECTORS
RIBBON CABLE
TERMINATOR CONNECTOR
MINI-MOTHERBOARD
POWER CONNECTORS
INTERCONNECT ACCESS COVER
DRIVER/RECEIVER BOARDS (100 Pin Connectors)
DRIVER/RECEIVER BOARDS (68 Pin Connectors)
MINI-FIT EXTERNAL POWER CONNECTOR
POWER CONNECTOR
ISOMETRIC VIEW (WITHOUT FRONT PANEL)

Figure 1-1.
SR2520 Module With Three I/O Boards and Six Driver/Receiver Boards.

I/O Boards

The I/O boards within the SR2520 are register-based. Each I/O board provides 32 I/O channels. The SR2520 can accommodate up to three I/O boards (up to 96 channels) and up to five SR2520s, each containing up to three I/O boards (96 channels) can be included in a single SR2500 subsystem. Each I/O channel generates digital stimulus patterns, provides real-time comparison capabilities on the response inputs, and contains logic analyzer type triggering and data recording functions, all at speeds up to 25 MHz.

Each stimulus pin contains output and tristate memories, allowing bidirectional signal paths. The response pin provides *expected response* and *mask* ("don't care") memories, which generate the expected input pattern used for the real-time comparison. The logic analyzer triggering and recording subsystem allows the recording of either the actual input pattern or the results of the real-time comparison of the expected response pattern and the input pattern (error data). Either may be saved and then later retrieved from the record memory, in much the same way you would use a logic analyzer.

VXI Bus Interface

Based on the IT9010 industry standard VXI bus interface chip, the SR2520 meets the requirements of VXI Bus Specification Versions 1.3 and 1.4. The SR2520 VXI bus interface receives commands, test parameters, data, and timing signals from the SR2510 Main Module.

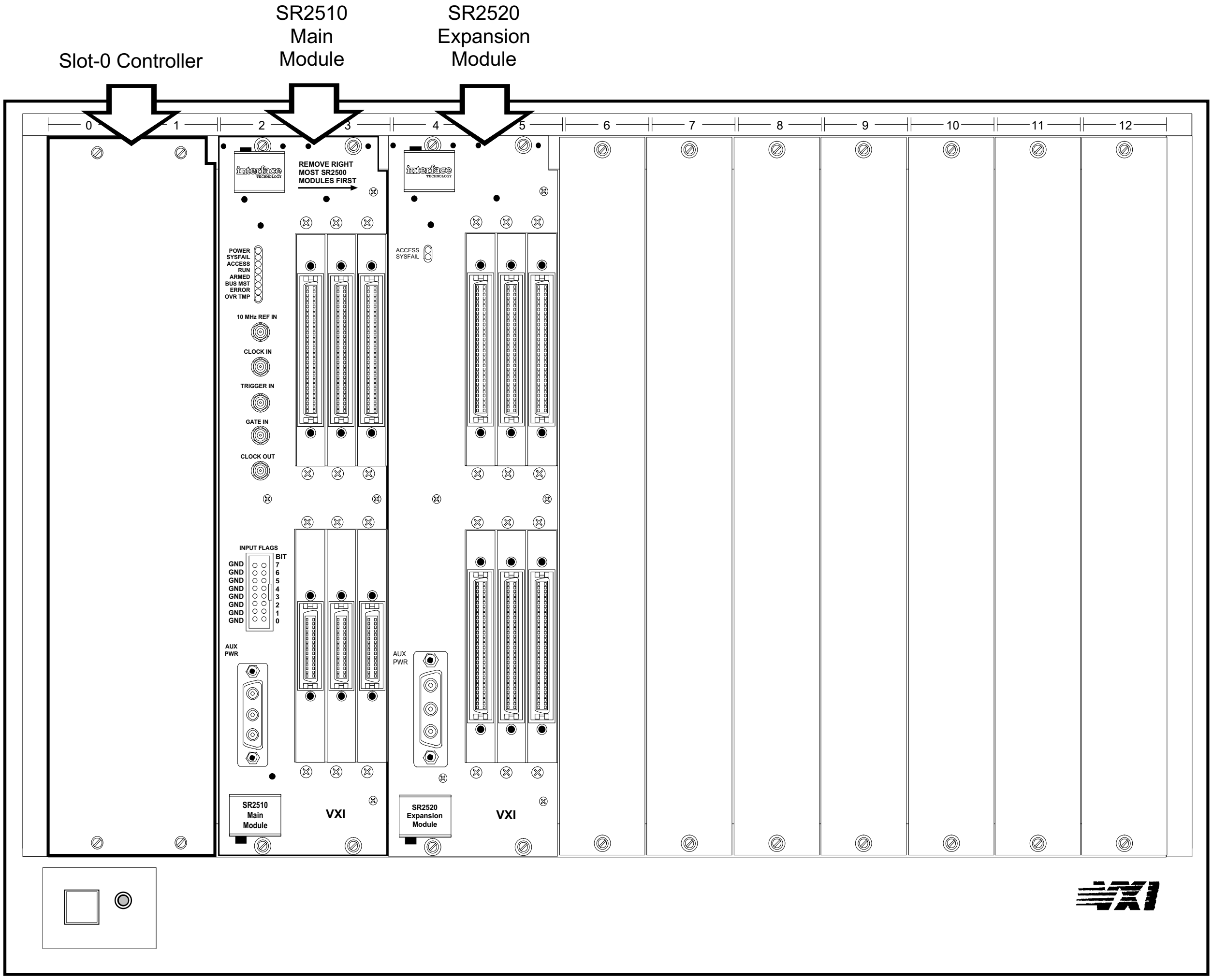


Figure 1-2.
VXI Chassis Showing SR2510 Main Moudle and SR2520 Expansion Module.

Controls and Indicators

See Fig. 1-3. All the connectors and LED indicators for the SR2520 are located on the module front panel.

LEDs

There are two LEDs located at the top of the SR2520 front panel.

- **POWER** (Green) - The POWER LED is connected to the system reset signal and is lit during normal operation. The LED will turn off during a system reset or if the +5V power supply drops below +4.7V.
- **SYSFAIL** (Red) - The SYSFAIL LED is off during normal operation. During the power-up sequence the LED is lit until the internal self-test passes, or remains lit if the self-test fails. If the self-test fails, error code information stored in the Data Low Register indicates the origin of the self-test failure (See Appendix A of this manual).

I/O Connectors

Each I/O Board (up to 3) has two I/O connectors. The number of pins, the pin arrangement, and the pin function varies, depending on the type of logic for which the I/O Board is configured (TTL, ECL, CMOS, or Variable Voltage). Refer to Chapter 3, *Installation* for additional details.

Auxiliary Power Connector

An Aux Pwr connector is provided for connecting an external source of +5 Vdc when the SR2520 is configured with more than one I/O Board. Refer to Chapter 3, *Installation* for additional details.

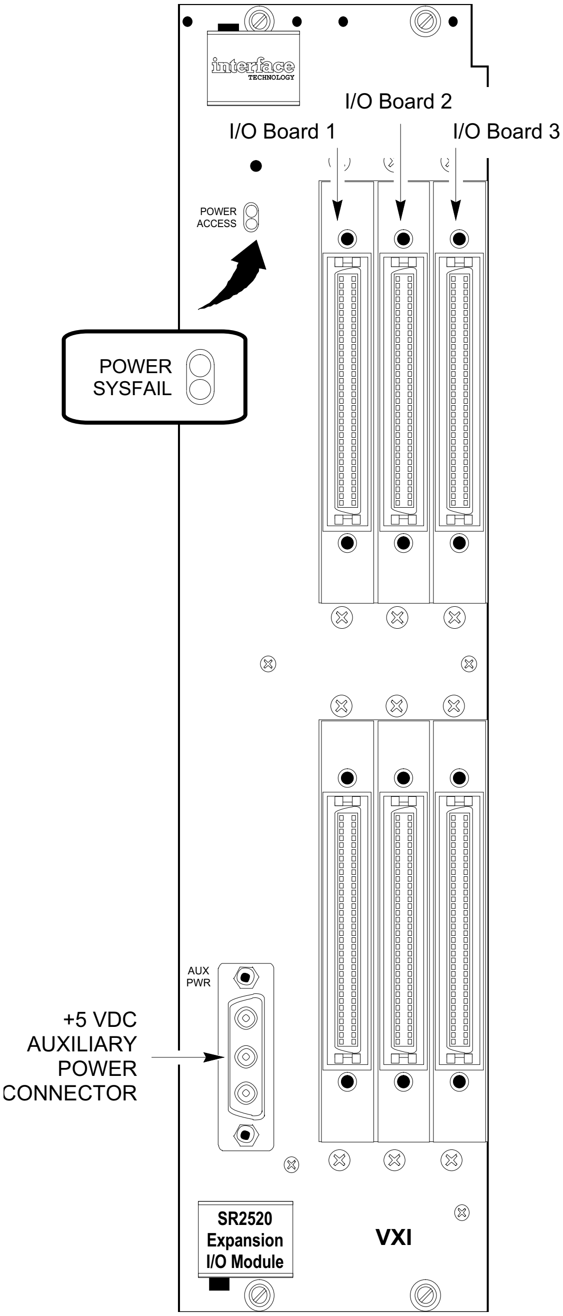


Figure 1-3.
Connectors and Indicators.

Interconnection With Other SR2500 Modules

All interconnections between the SR2520 Expansion Module and other SR2500 modules are made by means of the VXI backplane, and by a special connector at the side of the module. Interconnections are completed whenever Expansion Modules are added to the system. No additional cabling between modules is required. Interconnection between the SR2520 Expansion Module and the SR2510 Main Module is shown below. Refer to Chapter 3 *"Installation"* for additional interconnection information.

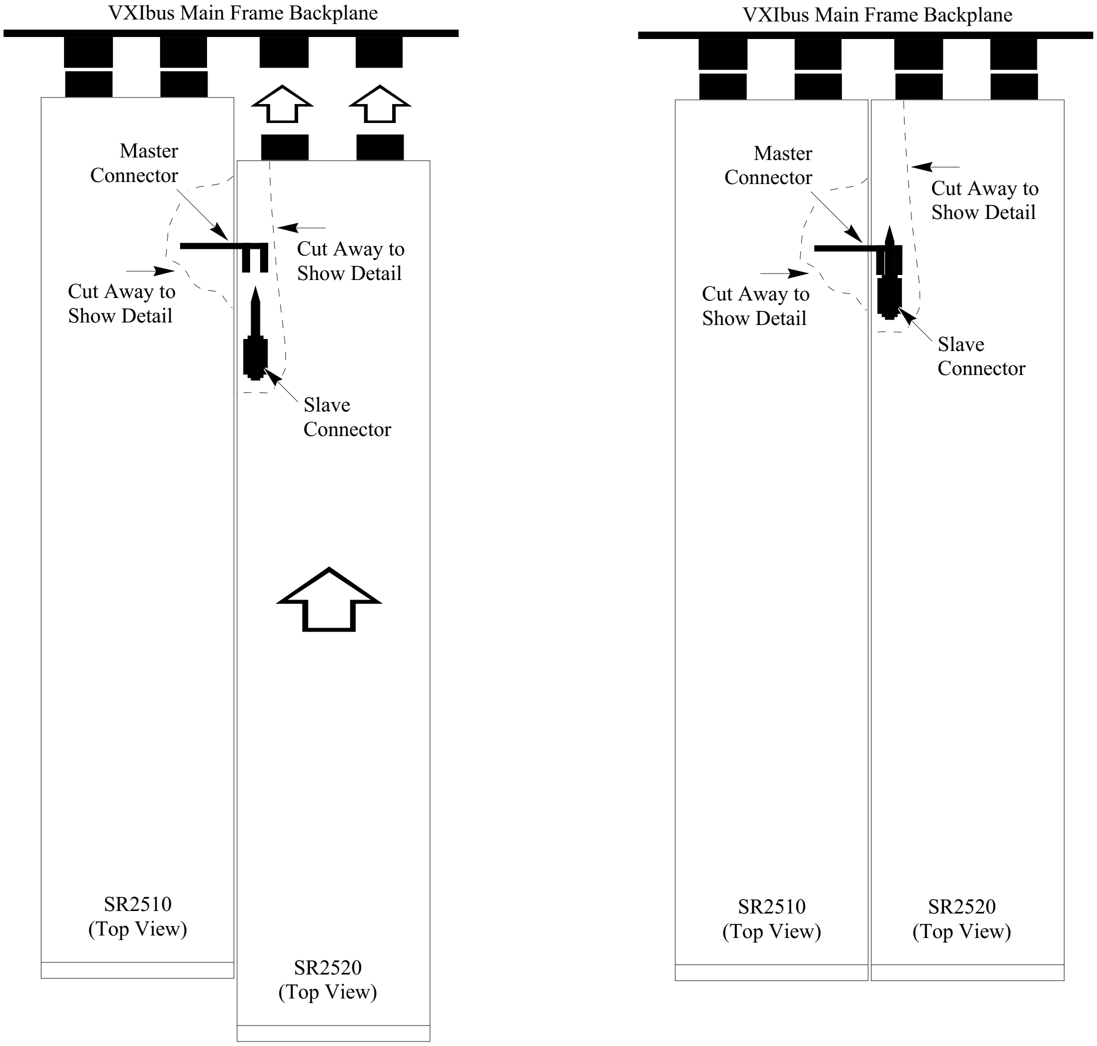


Figure 1-4.
SR2510 and SR2520 Interconnection, Top View.

SR2520 SPECIFICATIONS*

I/O Characteristics:	Differential TTL I/O	TTL I/O	Differential ECL I/O	CMOS I/O	Variable Voltage I/O	3.3V Logic I/O	LVDS I/O
Output Drivers							
Type	DS26F31M	74F125	100324	74AC125	-- n/s --	74LVT125	DS90C031
High Voltage (Voh)	3.2V typ	3.4V typ	-1.025V -0.870V[1]	4.2V, 24 mA typ	-1.5V to +7.0V[4]	3.2V typ	1.14 V typ
Low Voltage (Vol)	0.32V typ	0.55V max	-1.830V -1.620V[1]	0.4V, 24 mA typ	-3.0V to + 4.5V[4]	0.3V	1.07 V typ
Sink Current	20 mA @ 0.5V	64 mA max	-- n/a --	+24 mA max	50 mA max[2]	32 mA max	-- n/a --
Source Current	20 mA @ 0.5 V	15 mA max	-- n/a --	-24 mA max	50 mA max[2]	-32mA max	-- n/a --
Output Swing	-- n/a --	-- n/a --	-- n/a --	-- n/a --	0.0V to 11.0V p-p	-- n/a --	-- n/a --
Resolution	-- n/a --	-- n/a --	-- n/a --	-- n/a --	10 mV	-- n/a --	-- n/a --
Absolute Accuracy	-- n/a --	-- n/a --	-- n/a --	-- n/a --	100 mV	-- n/a --	-- n/a --
Abs. Max. Volt. (Hi-Z)	-- n/a --	-- n/a --	-- n/a --	-- n/a --	-3.0V to +7.0V	-- n/a --	-- n/a --
Output Impedance	-- n/a --	100 ohms	-- n/a --	100 ohms	50 ohms	100 ohms	-- n/a --
Input Receivers							
Type	DS26F32M	74ACT244	100325	74ACT244	-- n/s --	74ACT244	DS90C032
Diff. Input Volts (Vth)	0.2V min	-- n/a --	-- n/a --	-- n/a --	-- n/a --	-- n/a --	±200 mV max
Max Input Volts	±5.0V max	+5.0V max	-- n/a --	+5.0V max	-3.0V to +7.0V	+5.0V max	-0.3 to 4.8 V
Input Voltage, high (Vih)	-- n/a --	2.0V min	-1.165V -0.870V[3]	2.0V min	-- n/a --	2.0V min	-- n/a --
Input Voltage low, (Vil)	-- n/a --	0.8V max	-1.830V -1.475V[3]	0.8V max	-- n/a --	0.8V max	-- n/a --
Input Thrsh, high (Vth)	-- n/a --	-- n/a --	-- n/a --	-- n/a --	-2.9V to +5.5V	-- n/a --	-- n/a --
Input Thrsh, low (Vtl)	-- n/a --	-- n/a --	-- n/a --	-- n/a --	-2.9V to +5.5V	-- n/a --	-- n/a --
Resolution	-- n/a --	-- n/a --	-- n/a --	-- n/a --	10 mV	-- n/a --	-- n/a --
Absolute Accuracy	-- n/a --	-- n/a --	-- n/a --	-- n/a --	100 mV	-- n/a --	-- n/a --
Input Impedance	100 ohms	10k ohms	50 ohms to -2.0V	10k ohms	> 50k ohms	10k ohms	100 ohms

Notes: n/a = not applicable; n/s = not specified; Note 1: Min-Max, Measured with 50 ohm termination to -2.0 V dc bus;
Note 2: Aggregate static source/sink current is 800 mA per 32 channels; Note 3: min-max, single-ended; Note 4: unterminated

Data Formats:

NRZ	Non-Return-to-Zero
RZ	Return to Zero
RONE	Return-to-One
RC	Return-to-Complement
RI	Return-to-Inhibit / Tristate

VXI Specifications

Interface Compatibility:	
SR2520	Register-based, Servant
Revision	1.4
Size	C-size, Dual slot
Configuration	Static
Interrupt Level	Programmable 1-7
Triggers	TTLTRG 0-7
Power Requirements: (Note 2)	
+5.0 volts	21.5 A, max.
-5.2 volts	1.0 A, max.
+12.0 volts	0.1 A, max.
-12.0 volts	0.1 A, max.
-2.0 volts	1.0 A, max.

Note 2: Power values specified are with three TTL I/O cards installed.

Cooling Requirements:	
Per Slot Avg.	117 W, maximum per module (Note 2)
Airflow	8 liters / sec per module; 4 liters / sec per slot @ 0.2 mm of water pressure / 10°C temp. rise
Environmental Specifications:	

Temperature	Storage = -40°C to +75°C
	Operating = 0°C to +45°C

Humidity	5% to 95% relative, noncondensing
Software Drivers:	
National Instruments	LabView
National Instruments	LabWindows/CVI

* *Specifications subject to change without notice.*

CHAPTER 2

Theory of Operation

I/O Board

(Fig 2-1) The I/O board contains the stimulus, response and record logic for 32 channels of output and 32 channels input. Figure 2-1 shows the main components and data paths of this board. The I/O boards installed in the SR2510 module are addressed from the 68030 microprocessor while the I/O boards installed in the SR2520 modules are addressed from the VXI bus as a register-based instrument, (see SR2520 User's Manual for discussion of SR2520 principles of operation).

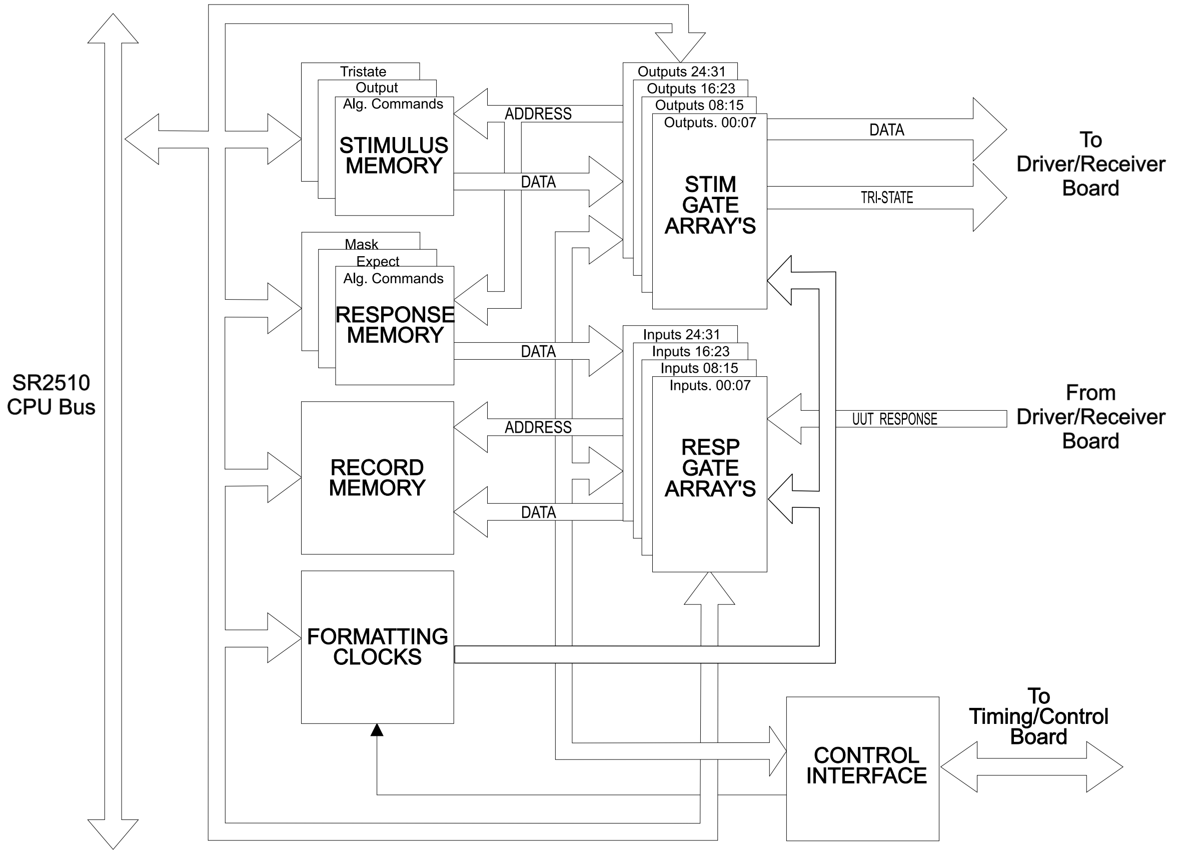


Figure 2-1.
SR2520 I/O Board Block Diagram.

Stimulus and Response Memory

(Fig 2-1) The stimulus and response memory blocks contains data needed to generate the stimulus and expect response data patterns, respectively. The VXI bus can read and write this memory when the control processor is not running. When the control processor is running, access to the VXI bus is blocked and the stimulus and response gate arrays have exclusive read-only access to the memory. The address counters regenerate the control processor address on each stimulus gate array and are used to drive the memory for the pair of stimulus and response gate arrays. Each I/O board contains four of these address buses. However, the buses are effectively locked together with the control processor's address counter.

Record Memory

(Fig 2-1) The record memory stores the data returned by the UUT, or the results of the data returned by the UUT and compared to the data provided by the expected response pattern generator. This is a read only memory for the user and can be read only when the control processor is not running. When the control processor is running, access to the memory by the user is blocked and the response gate arrays have exclusive write-only access to the memory. The record address counters are generated on each response gate array and are effectively locked together. The record address counter is independent of the control processor’s address counter, which controls stimulus and response vector sequencing.

Delayed Clock Generators

The SR2510 Timing/Control board provides a eight phase system clock, which is distributed throughout the SR2500 subsystem. The actual test vector rate is the system clock divided by an integer in the range of 1 to 65,535. These system clock cycles and phases are available to the stimulus logic to use for the data format delay and width parameters, and they are also available to the response logic to define the edge and window sample timing parameters.

In each stimulus gate array, logic is provided so the output pins, individually or in groups, may select any phase and any cycle of the system clock to assert the output when the NRZ data formatting mode is used. The assert time is synonymous with format delay. Additional logic is provided so that the output pins may select any other phase of any other cycle of the system clock to define the de-assert (deny) times for the return-to data formatting modes. The de-assert time is synonymous with format width.

For example, if the test rate is defined at 25.0 MHz, there is a single system clock for each test vector cycle. Therefore, there are eight phases (points, times) that are available for use with data formatting. The 40 ns test cycle period, divided by the eight available phases, yields an edge placement resolution of 5 ns. If the test rate is defined as 10 MHz, the system clock is set to 20 MHz, then divided by two. This means there are two system clocks for each test vector cycle. Therefore, there are 16 phases (points, times) that are available for use with data formatting. The 100 ns cycle period, divided by the 16 available phases, yields an edge placement resolution of 6.25 ns. Edge placement resolution will always fall within the range of 5-10 ns, regardless of the defined test rate.

The response gate arrays provide similar capabilities for use with edge and window sample modes. Each response input pin can use one system clock phase/cycle for the edge sample mode, or two system clock phase/cycles in the window mode.

Stimulus Gate Arrays

(Fig 2-2). The stimulus gate arrays, in conjunction with the stimulus memories (output, tristate and algorithmic command) form the heart of the SR2500 stimulus pattern generator. Each gate array is an 8 bit wide, high-speed pattern generator and data formatter. Pattern generation is accomplished by outputting the contents of the stimulus RAM directly or by algorithmically generating the data within the gate array using a high-speed ALU state machine. Some gate arrays may be programmed for RAM-backed pattern generation, while other gate arrays on the same card may be programmed for algorithmic pattern generation. While any gate array supports only one type of pattern generation during any test run, one of the algorithmic commands instructs the ALU state machine to pass data directly from RAM to the outputs. This allows mixing of algorithmic and RAM-backed pattern generation on the same pins.

Algorithmic Stimulus Pattern Generator

The pattern generator within the stimulus gate array is a high-speed programmable state machine. Instructions for this state machine are stored in the stimulus algorithmic command memory and instruct the gate array on a test clock-by-clock basis to either load the ALU output register from RAM or to algorithmically modify the contents of the ALU register. The output memory holds the clock-by-clock state of the output pins. The tristate memory holds the clock-by-clock state of the output enable, which allows a pin to be driven by the output memory on one clock cycle and tristated on the next clock cycle, thus achieving a bi-directional pin. In algorithmic mode, data patterns are defined by applying an algorithmic function to the internal ALU register. Multiple stimulus gate arrays may be cascaded together to create 16, 24 or 32 bit wide algorithmic patterns.

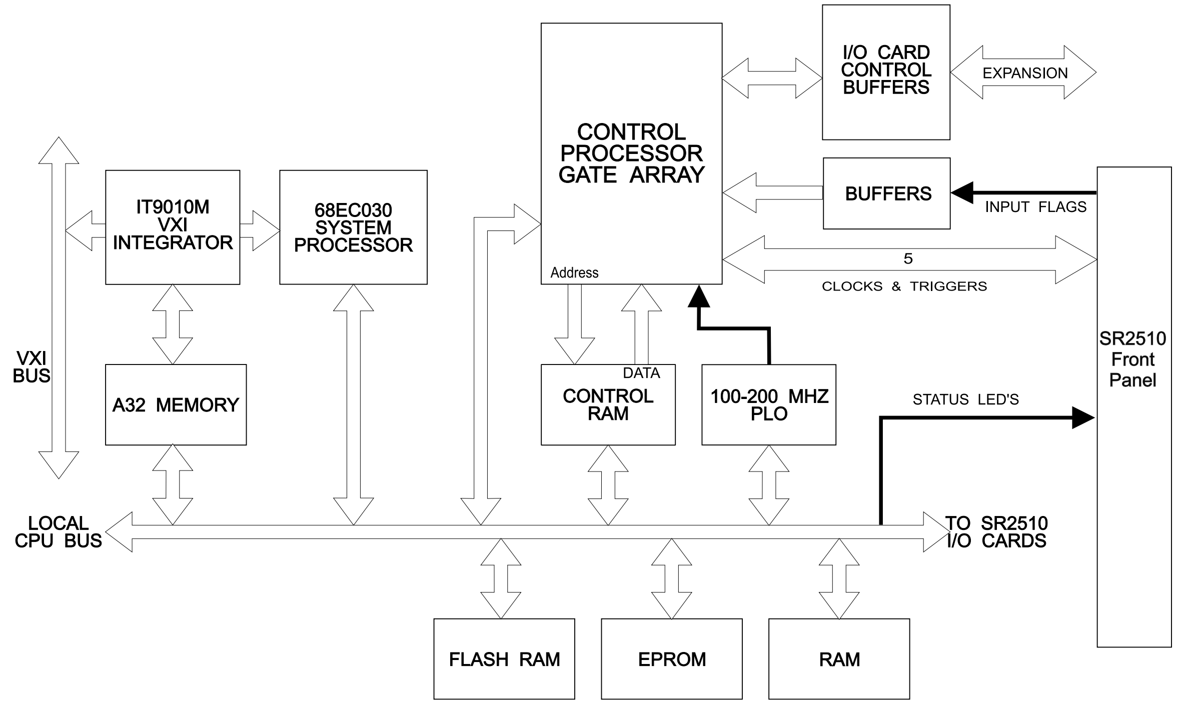


Figure 2-2.
Stimulus Gate Array Block Diagram.

Stimulus Output Pin Formatter

Each output channel contains a pin formatter that provides the following data formats: Non Return-to-Zero (NRZ), Return-to-Zero (RZ), Return to One (R1), Return-to-Compliment (RC), and Return-to-Inhibit (RI). The pin formatter section of each gate array can access the available system clock cycle/phase combinations, described in the section on *Delayed Clock Generators,* to define the assert and deny times for the output channels data format.

Response Gate Arrays

(Fig 2-3). The response gate arrays, in conjunction with the response memories (expect, "don't care" and algorithmic command) form the heart of the SR2500 expected response pattern generator, used in real-time compare operations. Each gate array is an 8 bit wide, high-speed pattern generator. Pattern generation is accomplished by outputting the contents of the RAM directly or by algorithmically generating the data within the gate array using a high-speed ALU state machine. Some gate arrays may be programmed for RAM-

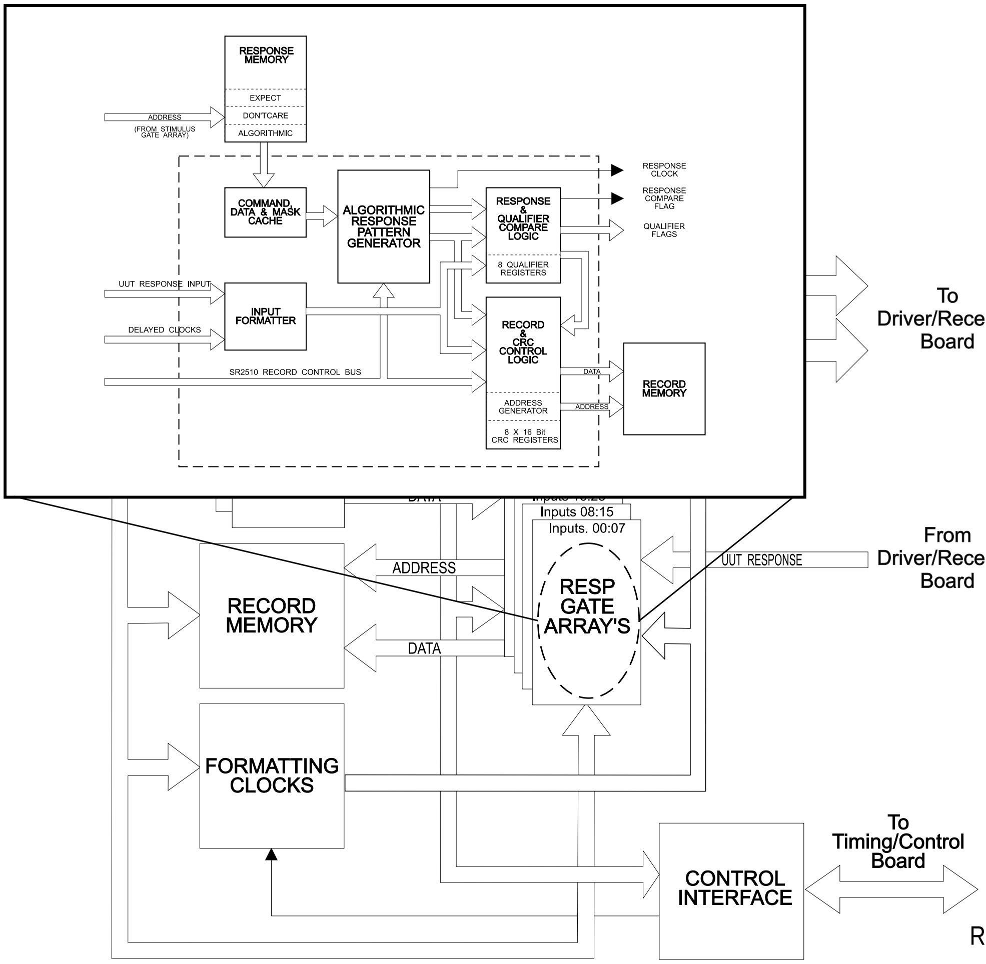


Figure 2-3.
Response Gate Array

backed pattern generation, while other gate arrays on the same card may be programmed for algorithmic pattern generation. While any gate array supports only one type of pattern generation during any test run, one of the algorithmic commands instructs the ALU state machine to pass data directly from RAM to the outputs. This effectively allows mixing of algorithmic and RAM-backed pattern generation on the same pins.

Expected Response Pattern Generator

The expected pattern generator within the response gate array is a high-speed, programmable state machine. Instructions for this state machine are stored in the response algorithmic command memory and instruct the gate array on a test clock-by-clock basis to either load the ALU output register from RAM or to algorithmically modify the contents of the ALU register. The expect memory holds the clock-by-clock state of the expected response pattern. The "don't care" memory holds the clock-by-clock state of the compare enable, which allows a pin to be disabled for compare on one clock cycle and enabled for compare on the next clock cycle. In algorithmic mode, data patterns are defined by applying an algorithmic function to the internal ALU register. Multiple gate arrays may be cascaded together to create 16, 24 or 32 bit wide algorithmic patterns.

Response Input Formatter

The response input formatter latches the response data from the UUT and passes it on to the response compare logic and the record control logic.

The pin formatter section of each gate array can access the available system clock cycle/phase combinations, described previously, which are used to define the edge and window sample times. In the edge mode, data is sampled at the selected system clock cycle/phase. In the window mode, data must be stable from the time when the window is opened (the first selected system clock cycle/phase combination), to the time the window closes (the second selected system clock cycle/phase combination). Window compare is used for detecting signal glitches.

Response Comparator

The response Comparator logic compares the data latched by the input formatter to the pattern generated by the response pattern generator, and passes the result to the record control logic. A response compare signal is generated for each test cycle, regardless of whether the real-time compare mode is being used. These signals, one from each I/O board, are summed on the SR2510 and used for generation of the error latch, and may also be used for test sequence control decisions based on real-time compare results.

Input Qualifier

The response gate arrays also contains 8 qualifier trigger registers and the qualifier compare logic. All 8 qualifier triggers (qualifiers) are compared to the latched input data that is passed from the input formatter on each test cycle. The results of the qualifier compare are passed back to the SR2510 module. The qualifier compare signals from each I/O board are summed on the SR2510 and used by the record state machine to start and stop data recording, provide filtered data recording, to start and stop CRC sampling. These signals may also be used for test sequence control decisions based on qualifier compare results.

Record Control

The record control logic in the response gate array provides two main functions: to generate record memory addresses and to pass data to the record memory for storing. This logic receives instructions from the SR2510 record state machine, located in the control processor, which determines when to record data and what data to record. After each record operation the record memory address is incremented by one, therefore, all data is recorded in a continuous, linear sequence. Since the record memory is addressed separately from the stimulus and response memories, there is no guarantee that the number of record vectors is the same as the number of stimulus and response vectors. Also, if record data wrapping is enabled, and the number of vectors recorded exceeds the size of the test, the oldest data in the record memory is overwritten. This will continue until the test stops or is aborted. record memory is then rearranged to provide a linear sequence of recorded data from oldest to most recent, accessed from the first vector to the last vector, respectively.

Two types of data may be passed to the record control logic for recording; the UUT response data latched by the input formatter, or the results of the real-time comparison performed in the response comparator. The latter is known as error data, or errors, and is represented as a 0 stored for each bit where the compare matched, and a "1" stored for each bit where the compare did not match. Selecting which data to record may be changed from within the SR2500 test using control structures called trace sequences. As there are 16 levels of trace sequences, this start and stop process of recording data may occur multiple times in a single test, allowing invalid or inappropriate responses to be ignored.

CRC Logic

Each input pin on the SR2500 I/O board has a 16 bit register and logic used for calculating CRC signatures, all located within the response gate arrays. CRC calculations are controlled from the same trace sequences as are used to control data recording. Based on matching of a trigger condition, CRC calculations may either be enabled or disabled. As there are 16 levels of trace sequences, this start and stop process of calculating CRC signatures may occur multiple times in a single test, allowing invalid or inappropriate samples to be ignored.

For the purpose of signature analysis, each input pin may be thought of as a separate serial channel. So, each SR2500 I/O board has 32 independent signature analysis channels. Enabling or disabling the CRC calculation is performed globally within the SR2500 system using the trace sequences. The "don't care" memory, which is used to enable individual bits for real-time compare, is also used to dynamically enable or disable individual CRC calculations. If CRC calculations are globally enabled, and the individual CRC calculation is enabled ("don't care" bit set to "0"), a CRC calculation is performed. If the individual CRC calculation is disabled ("don't care" memory set to "1"), the CRC calculation is disabled for that channel at that test cycle. When the CRC calculation is enabled, the data passed from the input formatter is used to update the value in the CRC registers based on the CCITT standard communication polynomial used to perform CRC calculations. When disabled, the data passed from the input formatter is ignored by the calculation logic, i.e., no calculation takes place. Data is passed to the CRC logic from the input formatter using the same sample clocks used to record data, so timing for CRC samples is identical to timing for record samples.

Algorithmic Commands

The stimulus and response gate arrays each contain algorithmic pattern generators that generate stimulus and response patterns, respectively. The following list of algorithmic commands are common to both stimulus and response pattern generation.

NONAlgorithmic

The Nonalgorithmic command allows the gate arrays to act as a pass through for data from RAM to the output pins. The data that is passed from RAM to output is also used to initialize the algorithmic register. This register can be acted on by other algorithmic commands to modify the data content programmatically after initialization.

INCrement

Increment the contents of the algorithmic register and pass the results to the output pins. If algorithmic fields greater than 8 bits are used, multiple gate arrays are interlinked. If an increment instruction causes an overflow, the overflow is used as a carry input to the next most significant gate array thus extending the count up to a maximum of 2^{32} before roll over.

DECrement

Decrement the contents of the algorithmic register and pass the results to the output pins. If algorithmic fields greater than 8 bits are used, multiple gate arrays are interlinked. If a decrement instruction causes an underflow, the underflow is used as a borrow input from the next most significant gate array thus extending the count up to a maximum of 2^{32} before roll over.

XOR

The XOR instruction will perform a bit-wise exclusive "ORing" of the algorithmic register with the contents of RAM. In this case the RAM acts as a modifier to the register and does not directly load it. In this way, selective bits of the algorithmic register may be complemented before passed to the output pins.

SLEFTZero

Shift the contents of the algorithmic register left (LSB to MSB) one bit, fill the LSB with "0" and pass the results to the output pins. If algorithmic fields greater than 8 bits are used, multiple gate arrays are interlinked. In this case, the MSB output of a less significant gate array is used as a LSB input to the next most significant gate array, thus extending the shift to a maximum 32 bits.

SLEFTOne

Shift the contents of the algorithmic register left (LSB to MSB) one bit, fill the LSB with "1" and pass the results to the output pins. If algorithmic fields greater than 8 bits are used, multiple gate arrays are interlinked. In this case, the MSB output of a less significant gate array is used as a LSB input to the next most significant gate array, thus extending the shift to a maximum 32 bits.

SLEFTComplement

Shift the contents of the algorithmic register left (LSB to MSB) one bit, complement the LSB and pass the results to the output pins. If algorithmic fields greater than 8 bits are used, multiple gate arrays are interlinked. In this case, the MSB output of a less significant gate array is used as a LSB input to the next most significant gate array, thus extending the shift to a maximum 32 bits.

RLEFT

Rotate the contents of the algorithmic register left (LSB to MSB) one bit, wrap the MSB to the LSB and pass the results to the output pins. If algorithmic fields greater than 8 bits are used, multiple gate arrays are interlinked. In this case, the MSB output of a less significant gate array is used as a LSB input to the next most significant gate array and the MSB of the most significant gate array is wrapped to the LSB of the least significant gate array, thus extending the rotate to a maximum 32 bits.

SRIGHTZero

Shift the contents of the algorithmic register right (MSB to LSB) one bit, fill the MSB with "0" and pass the results to the output pins. If algorithmic fields greater than 8 bits are used, multiple gate arrays are interlinked. In this case, the LSB output of a more significant gate array is used as a MSB input to the next least significant gate array and the LSB of the least significant gate array is wrapped to the MSB of the most significant gate array, thus extending the rotate to a maximum 32 bits.

Driver/Receiver Board

The SR2510 I/O Boards have separate I/O pattern generator boards and driver/receiver boards (D/R boards). Each I/O board provides two connectors of 16 stimulus channels and 16 response channels for connecting to the D/R boards. This means that each I/O board can support two logic families, in groups of 16 channels each. The D/R boards come in four different logic types, allowing the user to configure the SR2500 modules with the specific logic families required for the test system. On the stimulus side, the I/O pattern generator boards provide discrete TTL I/O signals to D/R boards, and the D/R boards translate the TTL I/O signals to the appropriate logic levels. For receiving, the D/R board accepts the UUT response and translates the UUT logic level to the TTL level required by the I/O board.

TTL Driver/Receiver Logic

(Fig 2-4) The TTL D/R board provides 16 channels of single ended TTL to/from the UUT. Separate output and input pins are used (32 signal pins), with a ground return for each signal. Bi-directional signals are supported by connecting the output and input pins together. Each TTL driver (74F125) has a 100 ohm resistor in series with the output. This provides 100 ohm back matched termination as well as additional short circuit and over voltage protection. The receiver (74ACT244) provides 10k pull up/down resistors on it's input.

CMOS Driver/Receiver Logic

(Fig 2-5) The CMOS D/R board provides 16 channels of single ended CMOS to/from the UUT. Separate output and input pins are used (32 signal pins), with a ground return for each signal. Bi-directional signals are supported by connecting the output and input pins together. Each CMOS driver (74ACT125) has a 100 ohm resistor in series with the output. This provides 100 ohm back matched termination as well as additional short circuit and over voltage protection. The receiver (74ACT244) provides 10k pull up/down resistors on it's input.

Differential TTL Driver/Receiver Logic

(Fig 2-6) The Differential TTL D/R board provides 16 channels of differential TTL to/from the UUT. Separate output and input pins are used (32 signal pins). Bi-directional signals are not supported directly on the D/R board, however, 16 tristate control signals are also brought out the differential TTL D/R board. These signals may be used on the UUT, or in a UUT adapter, to provide bi-directional control.

Differential ECL Driver/Receiver Logic

(Fig 2-7) The Differential ECL D/R board provides 16 channels of differential ECL to/from the UUT. Separate output and input pins are used (32 signal pins). Bi-directional signals are not supported directly on the D/R board, however, 16 tristate control signals are also brought out the differential ECL D/R board. These signals may be used on the UUT, or in a UUT adapter, to provide bi-directional control. Each side of the receiver input (100325) provides 50 ohm resistors terminated to -2.0V.

Programmable Driver / Receiver Logic

(Fig 2-8) The programmable, (variable voltage) D/R board (VV D/R) provides 32 bi-directional channels of I/O where the V_{OH} and V_{OL} levels are programmable over a range of -3V to +7V, and the V_{TH} and V_{TL} levels are programmable over a range of -2.9 to +5.5V. The V_{OH}, V_{OL}, V_{TH} and V_{TL} voltages are supplied external to the VV D/R board. Unlike the fixed level D/R boards, the VV D/R does not provide separate output and input pins. All pins are bi-directional signals with a ground return for each signal. The driver (EDGE649) is source terminated with a 50 ohm series resistor, and the receiver (EDGE649) provides a 50 ohm damping resistor in series with its input. The receiver is a dual-threshold part, capable of differentiating between a high input level, a low input level and an indeterminate (tristated) input. Additional logic in the form of a multiplexer and a oscillator are added to the output of each input receiver to allow the SR2500 VV D/R to detect/record if the response was valid or invalid. The truth table in Fig 2-8 indicates the various states that can be detected. If the detected state is other than the state that is tested for, the comparison will fail, the error latch will be set, and the record memory will store a "1" for each enable input bit that failed the test. The states that can be tested are a valid high and a valid low.

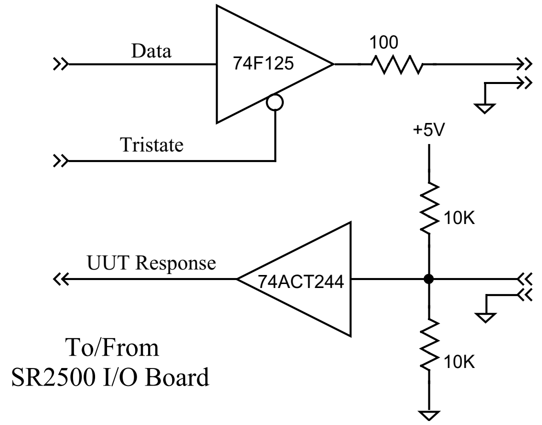


Figure 2-4. TTL Single Ended Driver/Receiver, (16 per D/R Board).

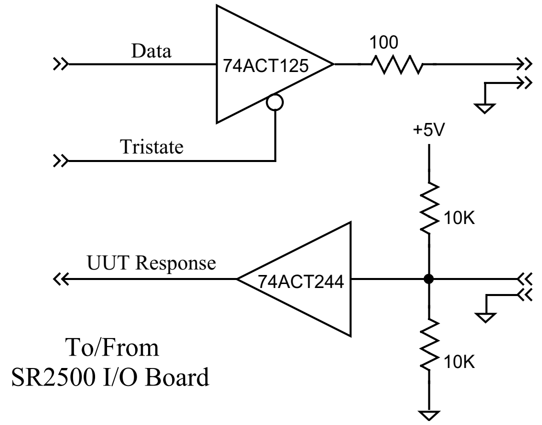


Figure 2-5. CMOS Single Ended Driver/Receiver, (16 per D/R Board).

Output Enables Are In Groups of Four

Tri-state bit 0	enables	bits 0-3
Tri-state bit 4	enables	bits 4-7
Tri-state bit 8	enables	bits 8-11
Tri-state bit 12	enables	bits 12-15
Tri-state bit 16	enables	bits 16-19
Tri-state bit 20	enables	bits 20-23
Tri-state bit 24	enables	bits 24-27
Tri-state bit 28	enables	bits 28-31

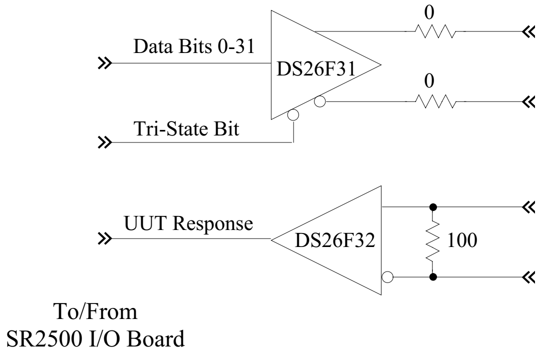


Figure 2-6. Differential TTL Driver/Receiver, (16 per D/R Board).

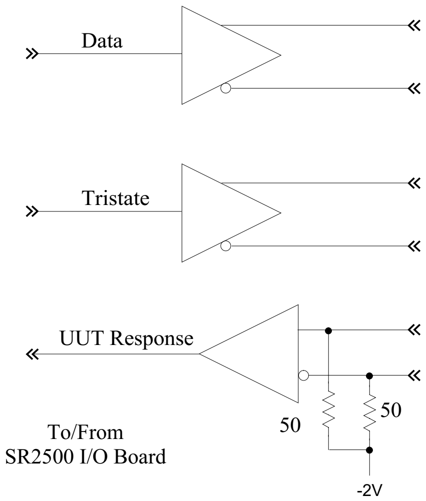
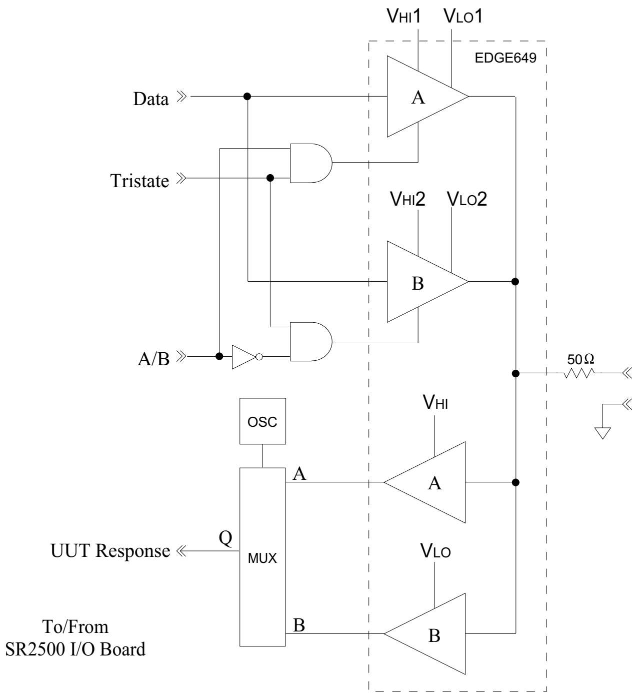


Figure 2-7.
Differential ECL Driver/Receiver,
(16 per D/R Board).

LVDS Driver/Receiver Logic

The LVDS D/R board provides 16 channels of LVDS to/from the UUT. Separate output and input pins are used (32 signal pins). Bidirectional pins are not supported directly on the D/R board, however the user may hardwire the input and outputs signal pins together to have bidirectional capability. Although single bit bidirectional pins are possible, outputs are enabled in groups of 4 (see Fig 2-9).



A	B	Q
0	0	Low
1	1	High
0	1	Indeterminate
1	0	Invalid

Figure 2-8.
Programmable Driver/Receiver,
(32 per D/R Board).

Output Enables Are In Groups of Four

Tri-stare bit 0	enables	bits 0-3
Tri-stare bit 4	enables	bits 4-7
Tri-stare bit 8	enables	bits 8-11
Tri-stare bit 12	enables	bits 12-15
Tri-stare bit 16	enables	bits 16-19
Tri-stare bit 20	enables	bits 20-23
Tri-stare bit 24	enables	bits 24-27
Tri-stare bit 28	enables	bits 28-31

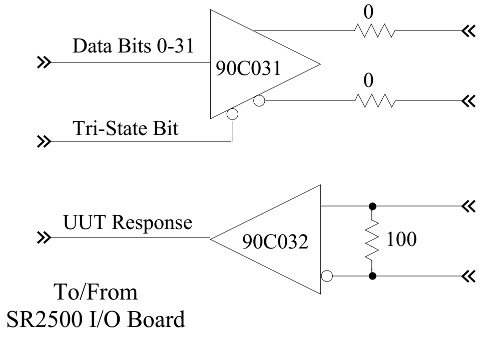


Figure 2-9. LVDS Driver/Receiver,
(16 per D/R Board).

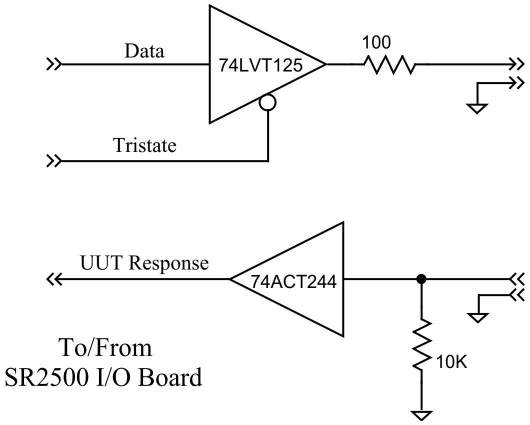


Figure 2-10. 3.3V Single Ended Driver/Receiver, (16 per D/R Board).

3.3 Volt Driver/Receiver Logic

(Fig 2-10) The 3.3V D/R board provides 16 channels of single ended logic to/from the UUT. Separate output and input pins are used (32 signal pins), with a ground return for each signal. Bi-directional signals are supported by connecting the output and input pins together. Each 3.3V driver (74LVT125) has a 100 ohm resistor in series with the output. This provides 100 ohm back matched termination as well as additional short circuit and over voltage protection. The receiver (74ACT244) has 10k pull down resistors on it's input. Both the drivers and receivers are 5V tolerant.

(THIS PAGE INTENTIONALLY LEFT BLANK)

CHAPTER 3

Installation

Scope of Chapter

This chapter contains instructions for unpacking, inspecting, installing, and checking out the SR2520 Expansion Module.

Unpacking and Inspection

Your SR2520 was thoroughly inspected and tested before shipment from the factory and is ready for immediate operation once all installation procedures have been completed. Carefully remove the instrument from its shipping carton and check for any obvious damage that may have occurred during shipment. If damage is found, report it to the freight carrier immediately. Interface Technology is not liable for damage that may have occurred during transit. Save the shipping carton and all packing material for possible future use.

Installation

Logical Addressing

Before installation, the logical address for the SR2520 Expansion Module must be set. Set the address switches according to the requirements of the Slot-0 controller. The address switches are numbered from one to eight. Switch 1 corresponds to the least significant bit (LSB) of the logical address. The address is entered in binary, where an ON switch sets the corresponding bit to 0 (Fig 3-1).

Note

The logical addresses of the SR2520 Expansion Modules must be set to a higher value than the logical address of the SR2510 Main Module. If there is more than one SR2510 in a VXI chassis, then the SR2520's with addresses between any 2 SR2510's, will be part of the lower addressed SR2510's system. The SR2520 with the lowest numbered logical address is Expansion Module #1. The next highest SR2520 logical address is Expansion Module #2. The highest SR2520 logical address is the most significant Expansion Module number. To verify all Expansion Modules have been recognized by the system, send a “*IDN?” query command.

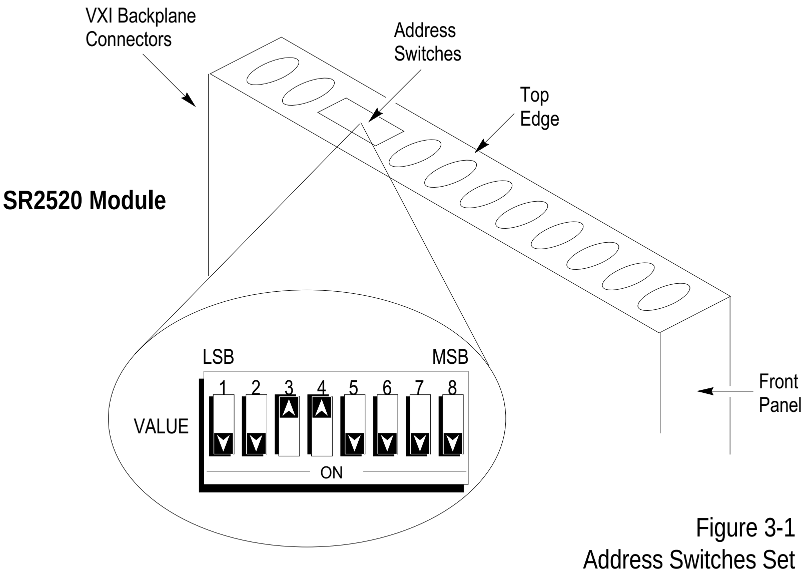


Figure 3-1
Address Switches Set
to Logical Address 12.

Slot Dependency

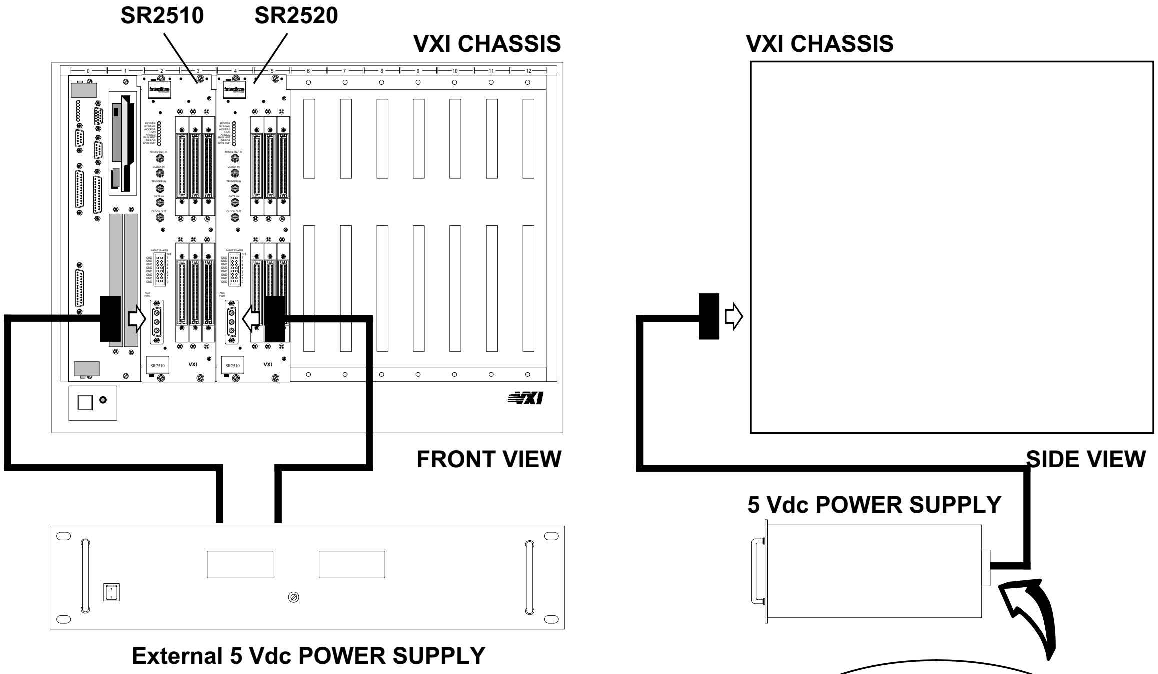
The SR2520 Expansion Module must be mounted immediately to the right of the SR2510 Main Module. The SR2500 uses bus master functions to identify the I/O boards installed in the system, so all SR2500 modules must be located in the same chassis.

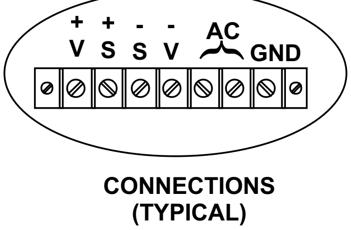
Backplane Jumpers

The SR2520 Expansion Module does not use any of the IACK or BG3 signals. These signals are passed through. The user may remove or install the jumpers for these VXI slots as required.

5 Vdc External Power Requirements

For SR2520 modules configured with more 32 I/O channels, additional 5 Vdc power is required from an external source. The external power is supplied to the Aux. Power connector located on the module front panel, see Fig. 3-2. The amount of operating current required from the external power supply is directly proportional to the number of modules installed. The SR2520 can supply enough internal 5 Vdc power to operate up to 32 I/O channels (one I/O board), independently of external power. When more than 32 channels/module are used, approximately 7.5 amperes is required for each additional 32 channels (e.g., a 64 channel module requires 7.5 A from an external 5 Vdc power supply; a 96 channel module requires 15 A).

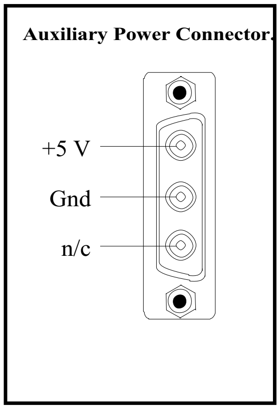




CONNECTIONS
(TYPICAL)

Power Supply Specifications:	
Output Voltage:	5 Vdc.
Allowed Variation	+0.25 / -0.125 Vdc.
Ouput Current:	30 A (Note 1)
DC Load Ripple/Noise	50 mV
Induced Ripple/Noise	50 mV

Note 1: Amperage indicated is for two (2) modules only. Each additional (fully populated) module (SR2510 or SR2520) requires an additional 15 A (approx) @ 5.0 Vdc.



Power Up Sequencing

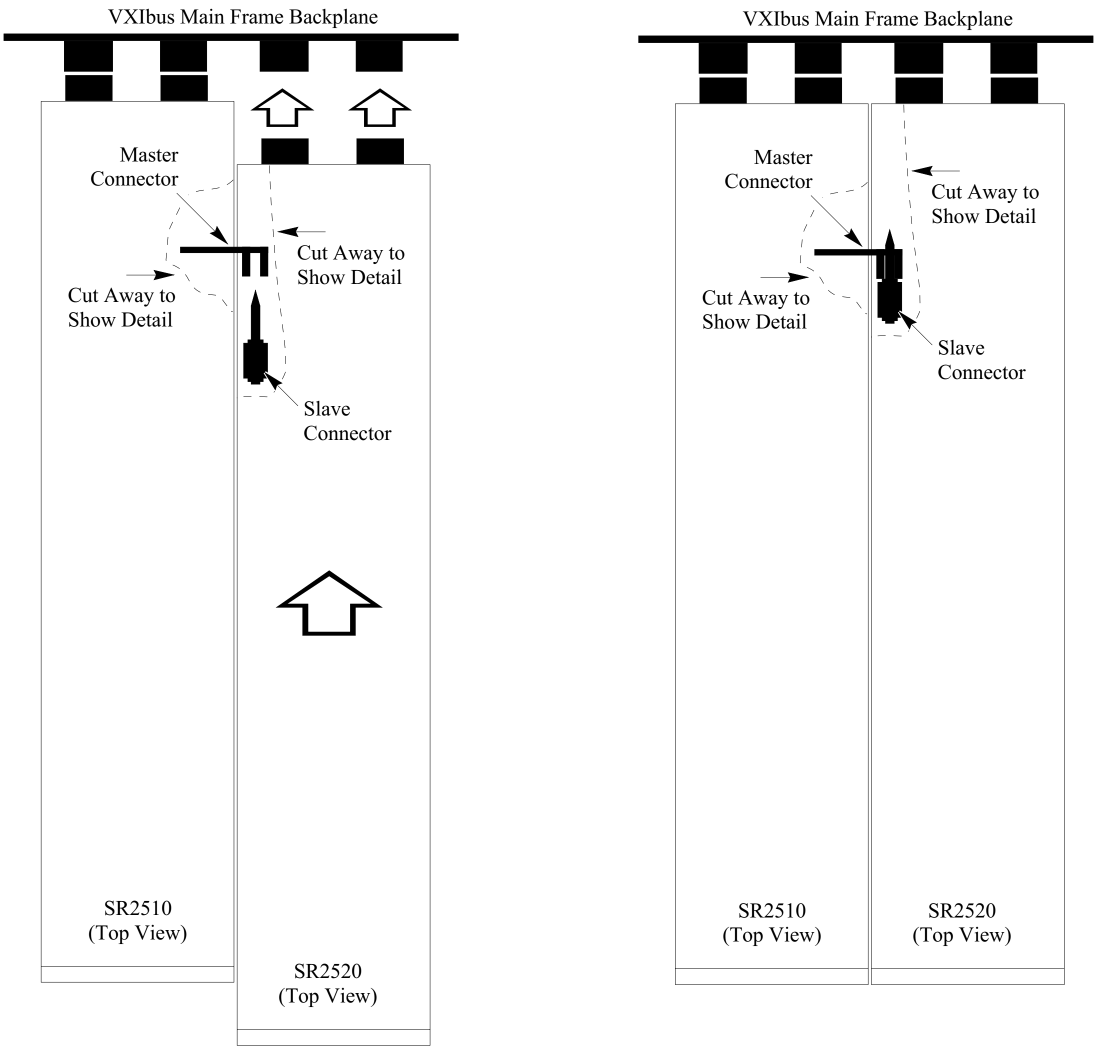
Note

It does not matter the order in which the external power supply and the VXI chassis are powered up, as long as both are on and stable when the first SCPI command is sent to the SR2510.

Figure 3-2.
Connection of External 5 Vdc Operating Power

Main and Expansion Module Interconnect

All interconnections between the SR2510 Main Module and SR2520 Expansion Modules are made by means of the VXI backplane and by a special connector at the side of the module. Interconnections are completed whenever Expansion Modules are added to the system. No additional cabling between modules is required. The second, and subsequent, SR2520 modules are connected in a similar manner.



SR2510 Main Module Fully Mounted in Mainframe; SR2520 Expansion Module Partially Installed.

SR2510 and SR2520 Modules Both Installed in Mainframe; Master and Slave Connectors Mated.

Figure 3-3. Interconnect Between SR2510 and SR2520, Top View.

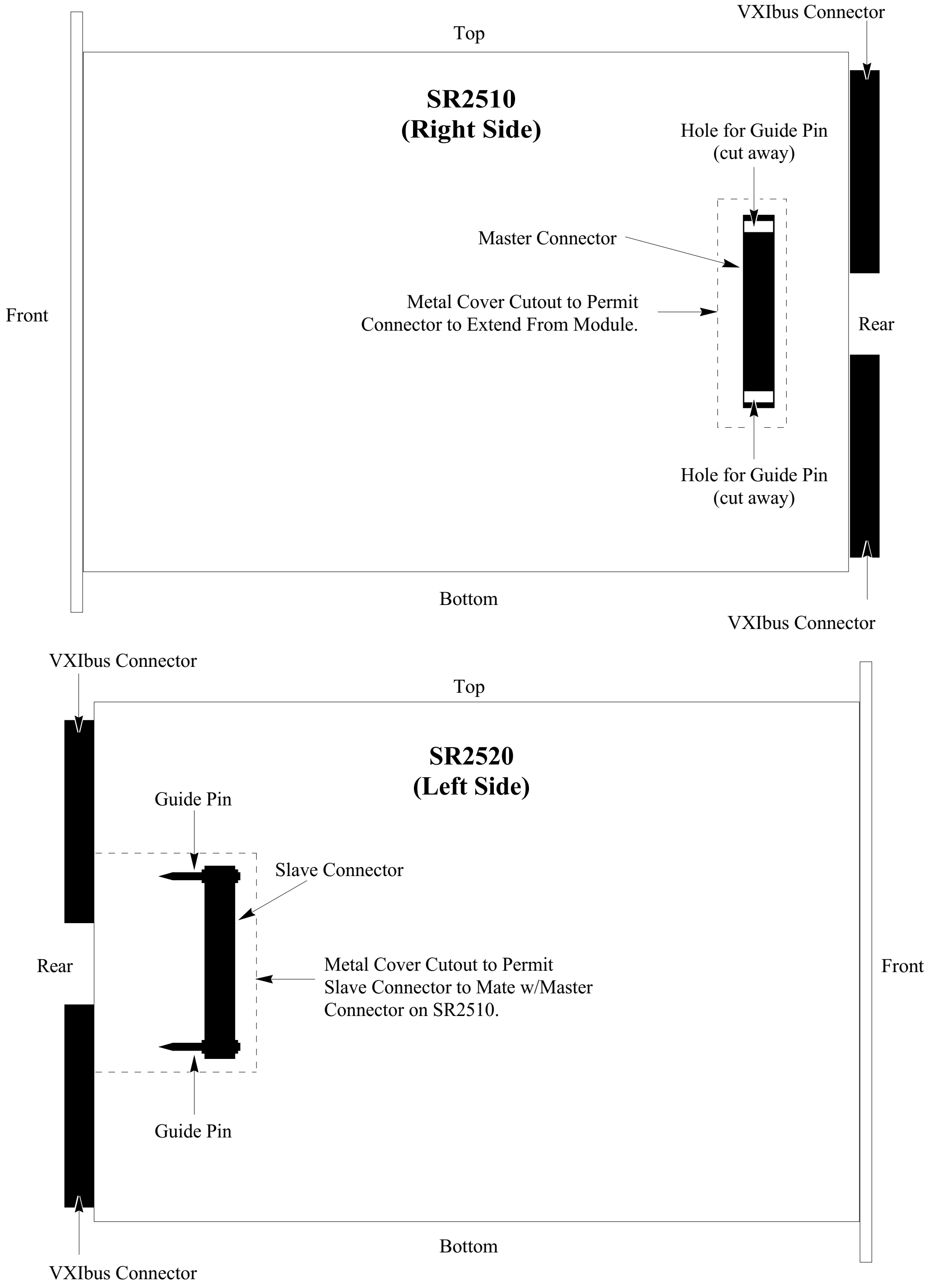


Figure 3-4. SR2510 and SR2520 Interconnect Connectors.

Installing I/O Boards

Each SR2520 modules can contain up to three I/O Boards, each of which provides 32 I/O channels. As shipped from the factory, the SR2520 will contain one, two, or three I/O Boards, depending on the customer's order. If less than three I/O Boards are supplied, cover plates are installed over the unused connector holes in the front panel. Additional I/O Boards can be ordered and installed by the user to expand system capability, at any time.

Required Equipment

To install additional I/O Boards, you will need the following tools and materials:

o screwdriver, No.1 Phillips
o screwdriver, 1/8" blade (pocket type)
o hex nut driver, 3/16"

Procedure ... Install I/O Bd No.2

To install a 2nd I/O Board in an SR2520 module, proceed as follows:

1. Turn VXI chassis power OFF. Disconnect all external cables from front panel of SR2520 module.

CAUTION

If there are SR2520 modules installed in the VXI chassis on the right hand side of the module to which the additional I/O Board is to be installed, these modules must be removed first to avoid damage to the interconnects between modules.

2. Observing the caution above, remove the SR2520 module from the VXI chassis.

3. Place the module on a clean workbench, orient the module to gain access to the right side cover, see Fig 3-5.

4. Remove the 14 #4-40 x .14" Phillips flat head screws securing the cover to the module; 11 screws are located on the side of the module and three more screws are located on the front of the module. The screws to be removed are indicated by the heavy circles in Fig 3-5.

5. Remove the right side cover from the SR2520 module.

6. Remove the 2 cover plates from the front of the module for the I/O Board to be installed ... i.e., cover plates for I/O Board 2 or cover plates for I/O Board 3 are secured to the front panel by four No.4-40 x 1/4" Phillips flat head screws, see Fig 3-6 (note: in Fig 3-6, these screws are shown, blown away from the module, directly to the left of the two Interface Boards).

7. See Fig 3-6 and Fig. 3-7. Remove the three spacer stacks at positions 1, 2, and 3, each consisting of two 1/2" and one 7/16" hex spacers (see Fig. 3-7a1). Also remove the No.4-40 x 1/4" Phillips pan head screw, split lock washer and nylon washer at position 4 of the I/O Board (see Fig 3-6 and Fig 3-7b1).

8. See Fig 3-6. Loosen, but do not remove, the three No.4-40 x 1/4" Phillips flat head mounting screws securing the Interface Board to the module front panel. (note: screws indicated by black triangles in Fig 3-6). Also loosen, but do not remove, the two small slotted head retainer screws securing the module latches to the module, see Fig 3-5. The front panel should now swing out slightly, away from the module, to allow access to install the I/O Board.

9. See Fig 3-6.Carefully place the new Expansion I/O Board, with the Interface Board(s) attached, in position inside the module.

10. See Fig 3-6. Connect the ribbon cable to J2; connect the mini-motherboard connector, and attach the power connector at J6 of the newly installed I/O Board.

11. See Fig 3-6 and Fig 3-7a2. Install the three spacer stacks at positions 1, 2, and 3 each consisting of one 1/2" and one 7/16" hex spacers (see Fig 5-8a2). Also install a 7/16" hex spacer and the No.4-40 x 1/4" Phillips pan head screw, split lock washer and nylon washer at position 4 of the I/O Board (see Fig 3-6 and Fig 3-7b2).

12. See Fig 3-6. Install the four No.4-40 x 1/4" flat head Phillips screws securing the Interface Boards of the new I/O Board to the front panel

13. Retighten the three No.4-40 x 1/4" Phillips flat head mounting screws securing the Interface Board to the module front panel that were loosened in step 8. Also retighten the two small slotted head retainer screws securing the module latches to the module that were loosened in step 8.

14. Reinstall the module cover. Reinstall and tighten the 14 mounting screws securing the cover to the module, see Fig 3-5.

Procedure ... Install I/O Bd No.3

The procedure for installing I/O Board No.3 is, essentially, the same as that for installing I/O Board No.2, except for the arrangement of the spacer stacks at positions 1-3 and position 4, as depicted in Figures 3a and 3b, respectively. Use these figures as a guideline to ensure correct I/O Board spacing inside the module.

Note when installing I/O Boards that the power supplied to connector J6 of I/O Board No.1 is taken from the Power Interface Board (see lower part of Fig 3-6), while power supplied to the same connector (J6) of I/O

Boards No.2 and No.3 is taken from the Aux Power Cable going to the Aux Power connector on the module front panel. If I/O Boards No.2 and/or No.3 are not used, the unused connectors of the Aux Power Cable will be tied off and tucked loosely inside the module.

Also note that the ribbon cable (upper part of Fig 3-6) has a separate connector for each of the I/O Boards. Unused connectors on the ribbon cable are left unterminated.

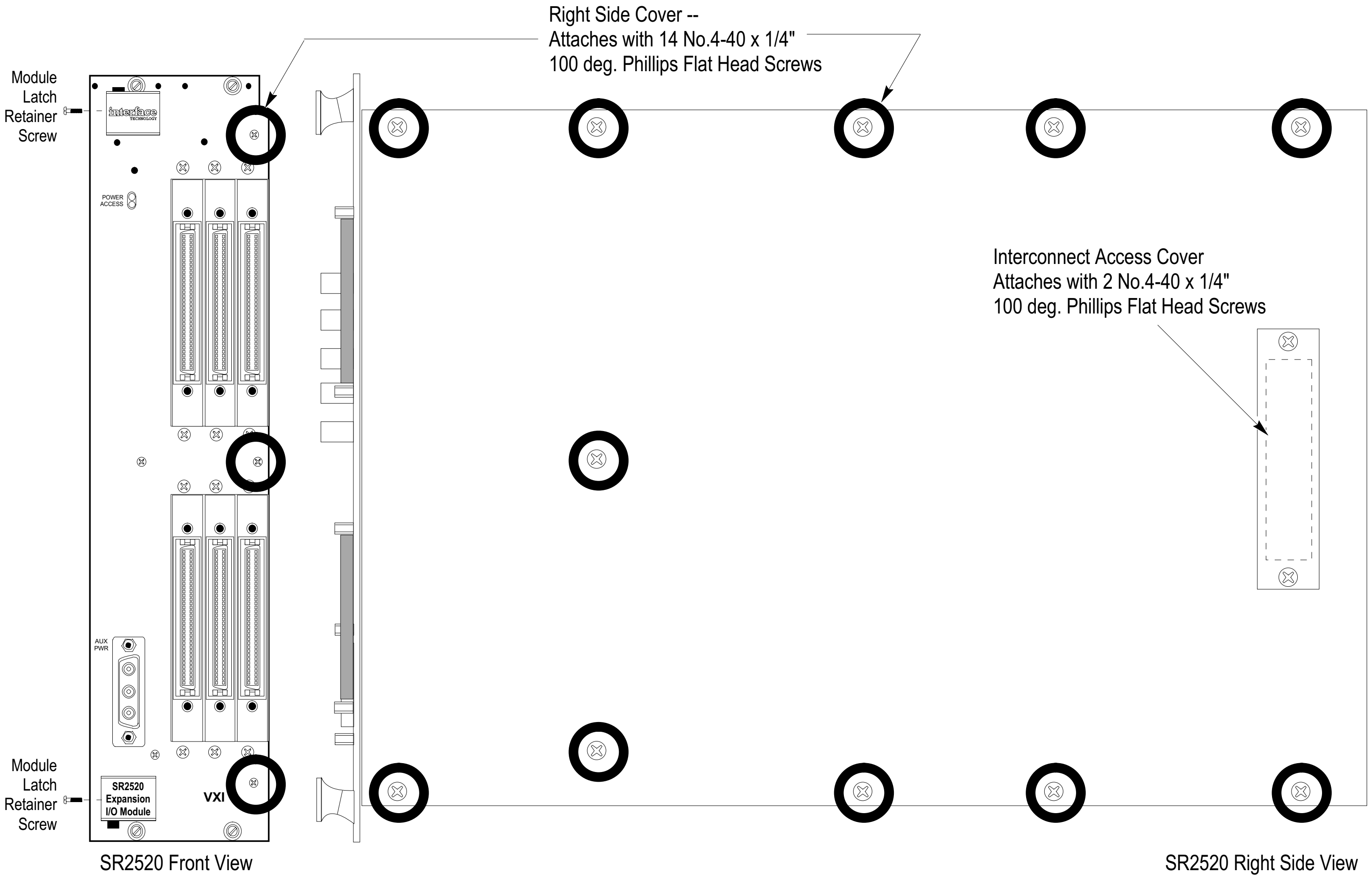

Right Side Cover --
Attaches with 14 No.4-40 x 1/4"
100 deg. Phillips Flat Head Screws
Module Latch Retainer Screw
interface TECHNOLOGY
POWER ACCESS
Interconnect Access Cover
Attaches with 2 No.4-40 x 1/4"
100 deg. Phillips Flat Head Screws
AUX PWR
Module Latch Retainer Screw
SR2520 Expansion I/O Module
VXI
SR2520 Front View
SR2520 Right Side View

Figure 3-5. Cover Screws

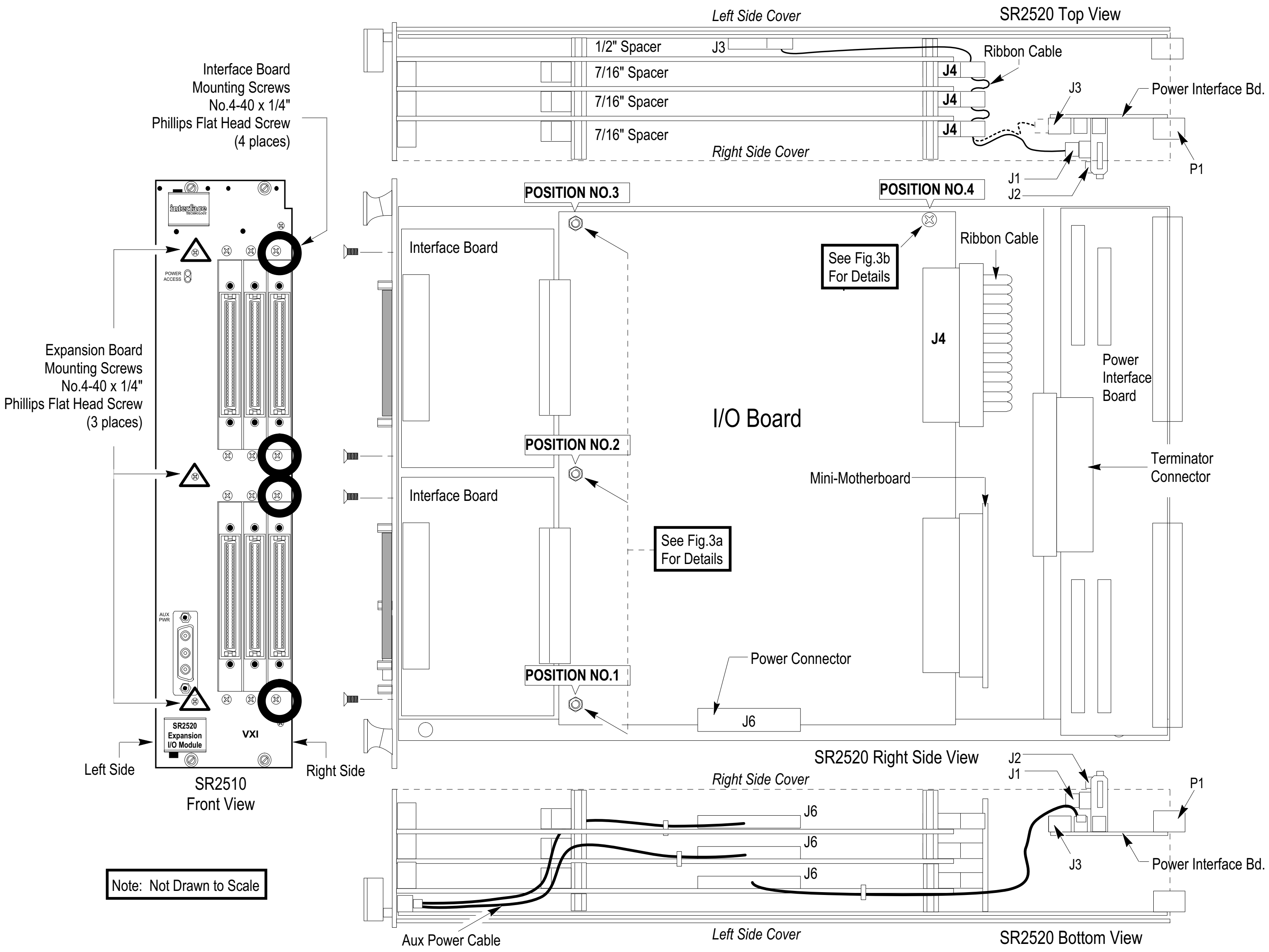
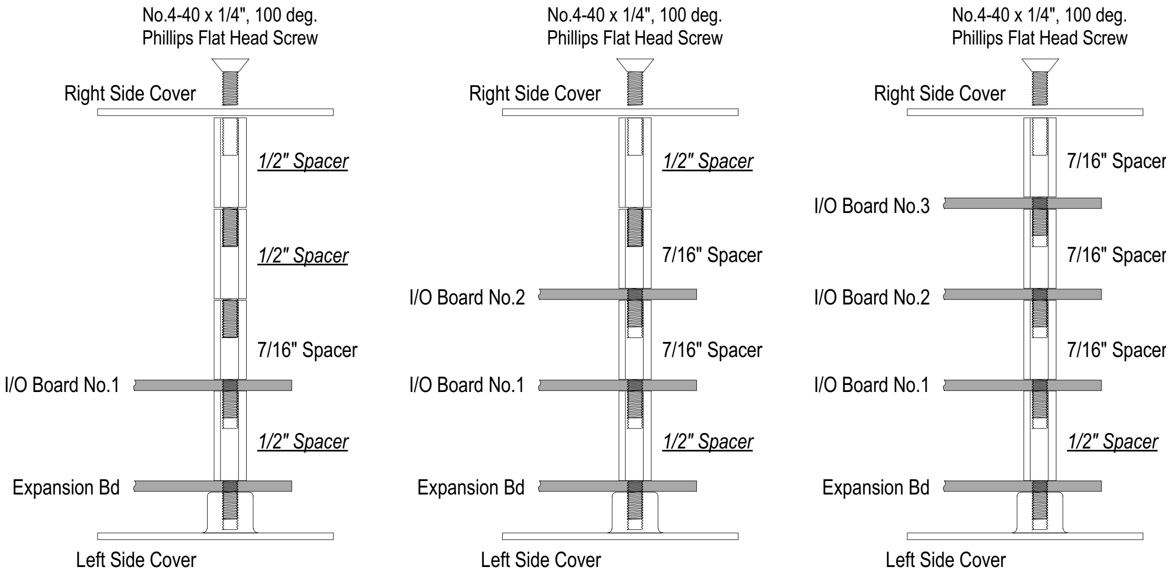
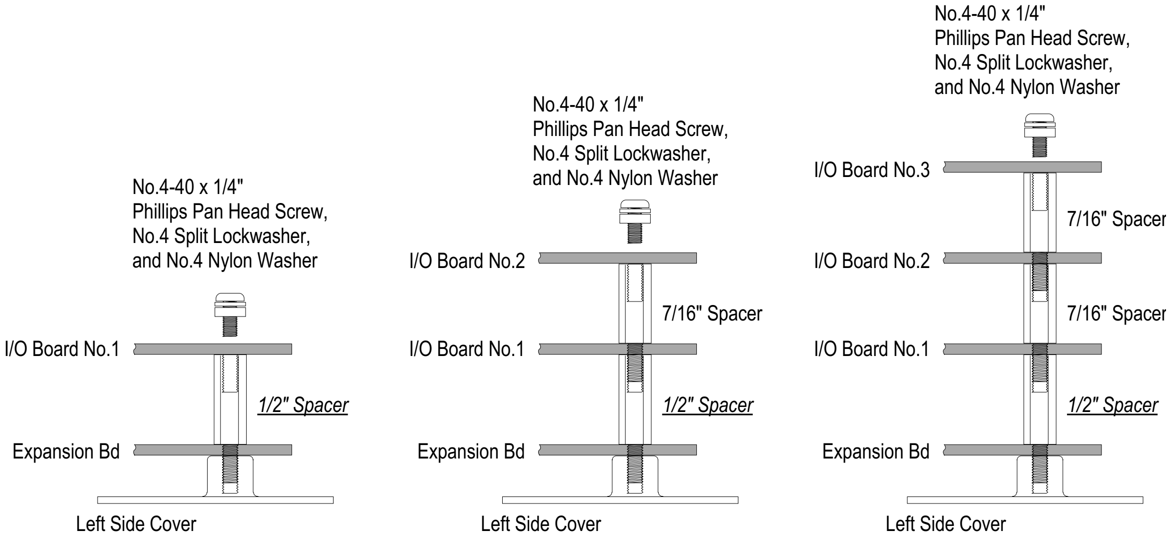


Figure 3-6. I/O Board Mounting Hardware.



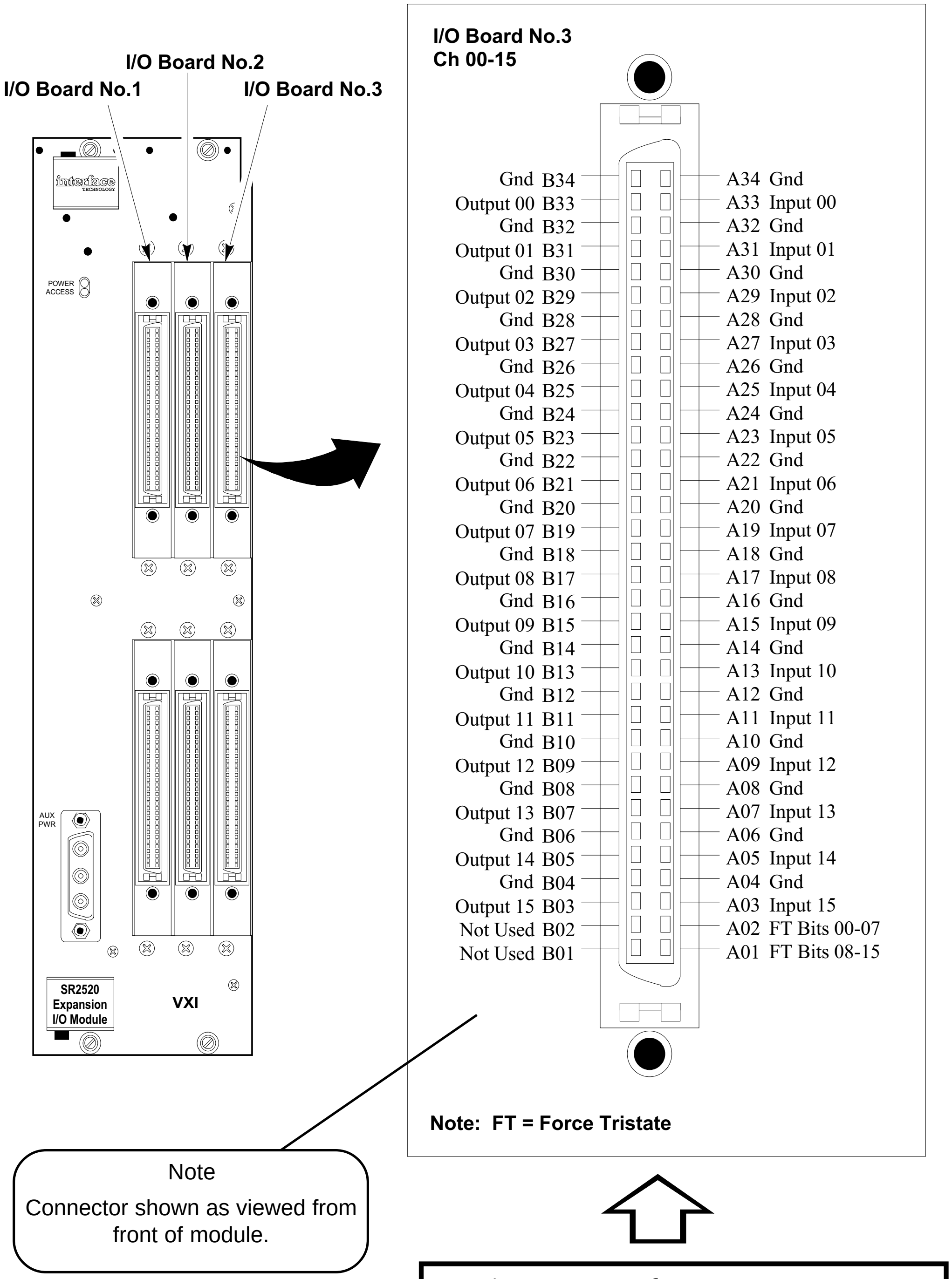
1. One I/O Board. *2. Two I/O Boards.* *3. Three I/O Boards.*

Figure 3-7a.
Buildup of Standoff Spacers at Positions 1-3 for Configurations of One, Two, and Three I/O Boards.



1. One I/O Board. *2. Two I/O Boards.* *3. Three I/O Boards.*

Figure 3-7b.
Buildup of Standoff Spacers at Position 4 for Configurations of One, Two, and Three I/O Boards.



Gnd	B34	A34	Gnd
Output 00	B33	A33	Input 00
Gnd	B32	A32	Gnd
Output 01	B31	A31	Input 01
Gnd	B30	A30	Gnd
Output 02	B29	A29	Input 02
Gnd	B28	A28	Gnd
Output 03	B27	A27	Input 03
Gnd	B26	A26	Gnd
Output 04	B25	A25	Input 04
Gnd	B24	A24	Gnd
Output 05	B23	A23	Input 05
Gnd	B22	A22	Gnd
Output 06	B21	A21	Input 06
Gnd	B20	A20	Gnd
Output 07	B19	A19	Input 07
Gnd	B18	A18	Gnd
Output 08	B17	A17	Input 08
Gnd	B16	A16	Gnd
Output 09	B15	A15	Input 09
Gnd	B14	A14	Gnd
Output 10	B13	A13	Input 10
Gnd	B12	A12	Gnd
Output 11	B11	A11	Input 11
Gnd	B10	A10	Gnd
Output 12	B09	A09	Input 12
Gnd	B08	A08	Gnd
Output 13	B07	A07	Input 13
Gnd	B06	A06	Gnd
Output 14	B05	A05	Input 14
Gnd	B04	A04	Gnd
Output 15	B03	A03	Input 15
Not Used	B02	A02	FT Bits 00-07
Not Used	B01	A01	FT Bits 08-15

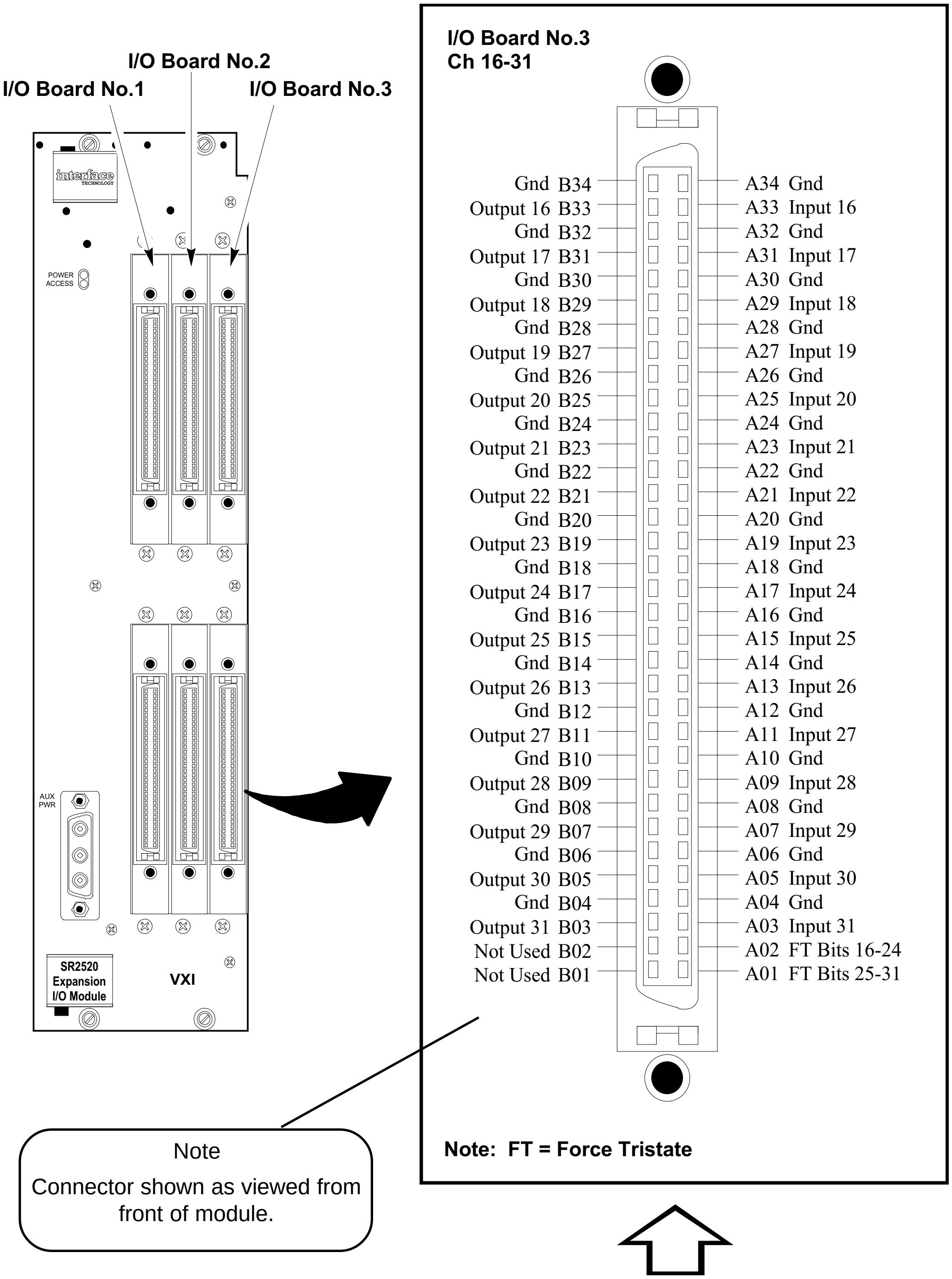
Note: FT = Force Tristate

Note
Connector shown as viewed from front of module.

Mating Connector for SR2520
TTL, 3.3 V, or CMOS, Ch. 00-15.

3M Company Part No. 10168-6000EC

Figure 3-8. SR2510 Signal Connector Pinouts, TTL, 3.3 V, or CMOS, Ch. 00-15.



I/O Board No.3
Ch 16-31

Gnd	B34	A34	Gnd
Output 16	B33	A33	Input 16
Gnd	B32	A32	Gnd
Output 17	B31	A31	Input 17
Gnd	B30	A30	Gnd
Output 18	B29	A29	Input 18
Gnd	B28	A28	Gnd
Output 19	B27	A27	Input 19
Gnd	B26	A26	Gnd
Output 20	B25	A25	Input 20
Gnd	B24	A24	Gnd
Output 21	B23	A23	Input 21
Gnd	B22	A22	Gnd
Output 22	B21	A21	Input 22
Gnd	B20	A20	Gnd
Output 23	B19	A19	Input 23
Gnd	B18	A18	Gnd
Output 24	B17	A17	Input 24
Gnd	B16	A16	Gnd
Output 25	B15	A15	Input 25
Gnd	B14	A14	Gnd
Output 26	B13	A13	Input 26
Gnd	B12	A12	Gnd
Output 27	B11	A11	Input 27
Gnd	B10	A10	Gnd
Output 28	B09	A09	Input 28
Gnd	B08	A08	Gnd
Output 29	B07	A07	Input 29
Gnd	B06	A06	Gnd
Output 30	B05	A05	Input 30
Gnd	B04	A04	Gnd
Output 31	B03	A03	Input 31
Not Used	B02	A02	FT Bits 16-24
Not Used	B01	A01	FT Bits 25-31

Note: FT = Force Tristate

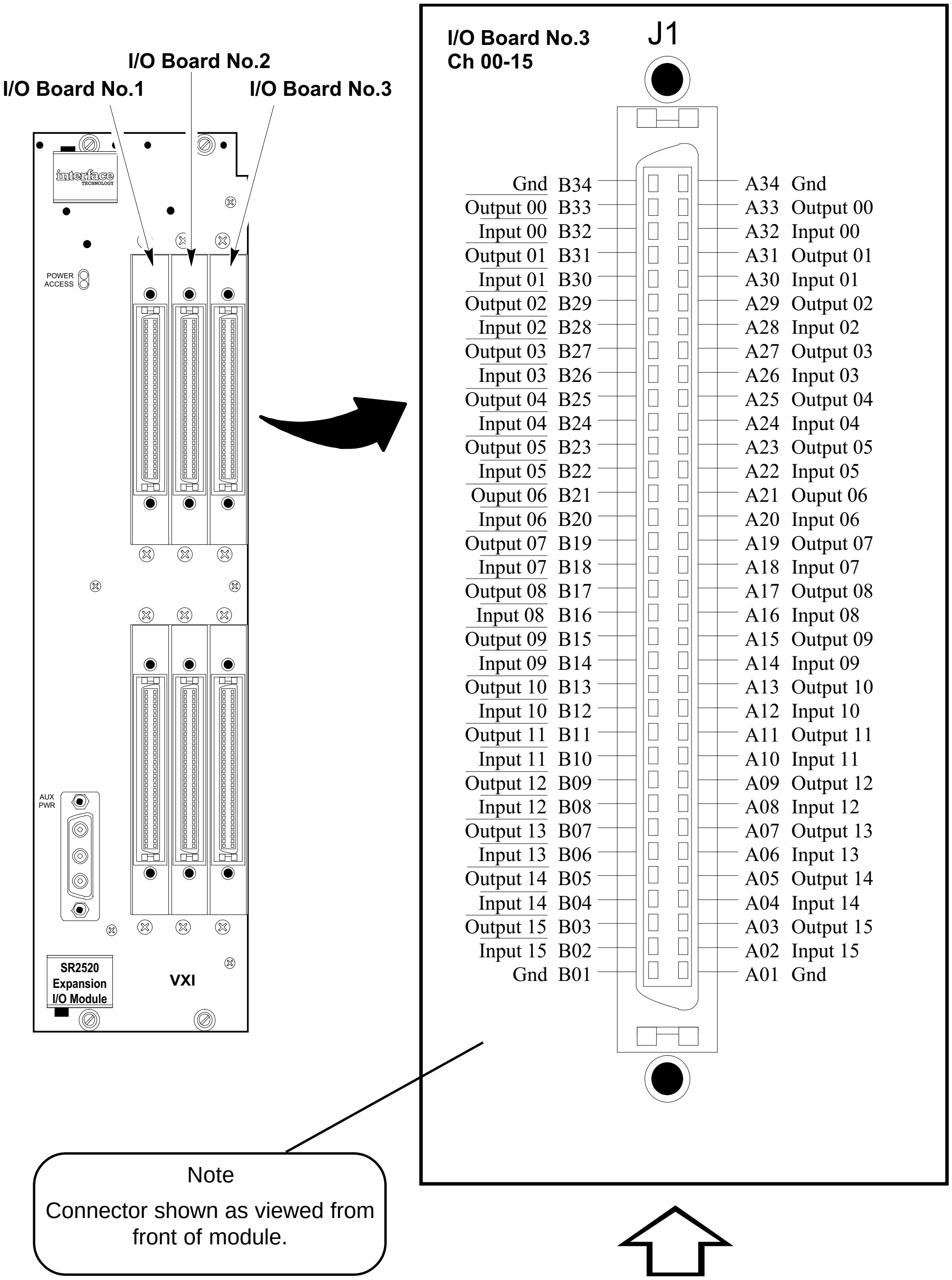
Note

Connector shown as viewed from front of module.

Mating Connector for SR2520
TTL, 3.3 V, or CMOS, Ch. 16-31.

3M Company Part No. 10168-6000EC

Figure 3-9. SR2520 Signal Connector Pinouts, TTL, 3.3 V, or CMOS, Ch. 16-31.



Signal	Pin	Pin	Signal
Gnd	B34	A34	Gnd
Output 00	B33	A33	Output 00
Input 00	B32	A32	Input 00
Output 01	B31	A31	Output 01
Input 01	B30	A30	Input 01
Output 02	B29	A29	Output 02
Input 02	B28	A28	Input 02
Output 03	B27	A27	Output 03
Input 03	B26	A26	Input 03
Output 04	B25	A25	Output 04
Input 04	B24	A24	Input 04
Output 05	B23	A23	Output 05
Input 05	B22	A22	Input 05
Ouput 06	B21	A21	Ouput 06
Input 06	B20	A20	Input 06
Output 07	B19	A19	Output 07
Input 07	B18	A18	Input 07
Output 08	B17	A17	Output 08
Input 08	B16	A16	Input 08
Output 09	B15	A15	Output 09
Input 09	B14	A14	Input 09
Output 10	B13	A13	Output 10
Input 10	B12	A12	Input 10
Output 11	B11	A11	Output 11
Input 11	B10	A10	Input 11
Output 12	B09	A09	Output 12
Input 12	B08	A08	Input 12
Output 13	B07	A07	Output 13
Input 13	B06	A06	Input 13
Output 14	B05	A05	Output 14
Input 14	B04	A04	Input 14
Output 15	B03	A03	Output 15
Input 15	B02	A02	Input 15
Gnd	B01	A01	Gnd

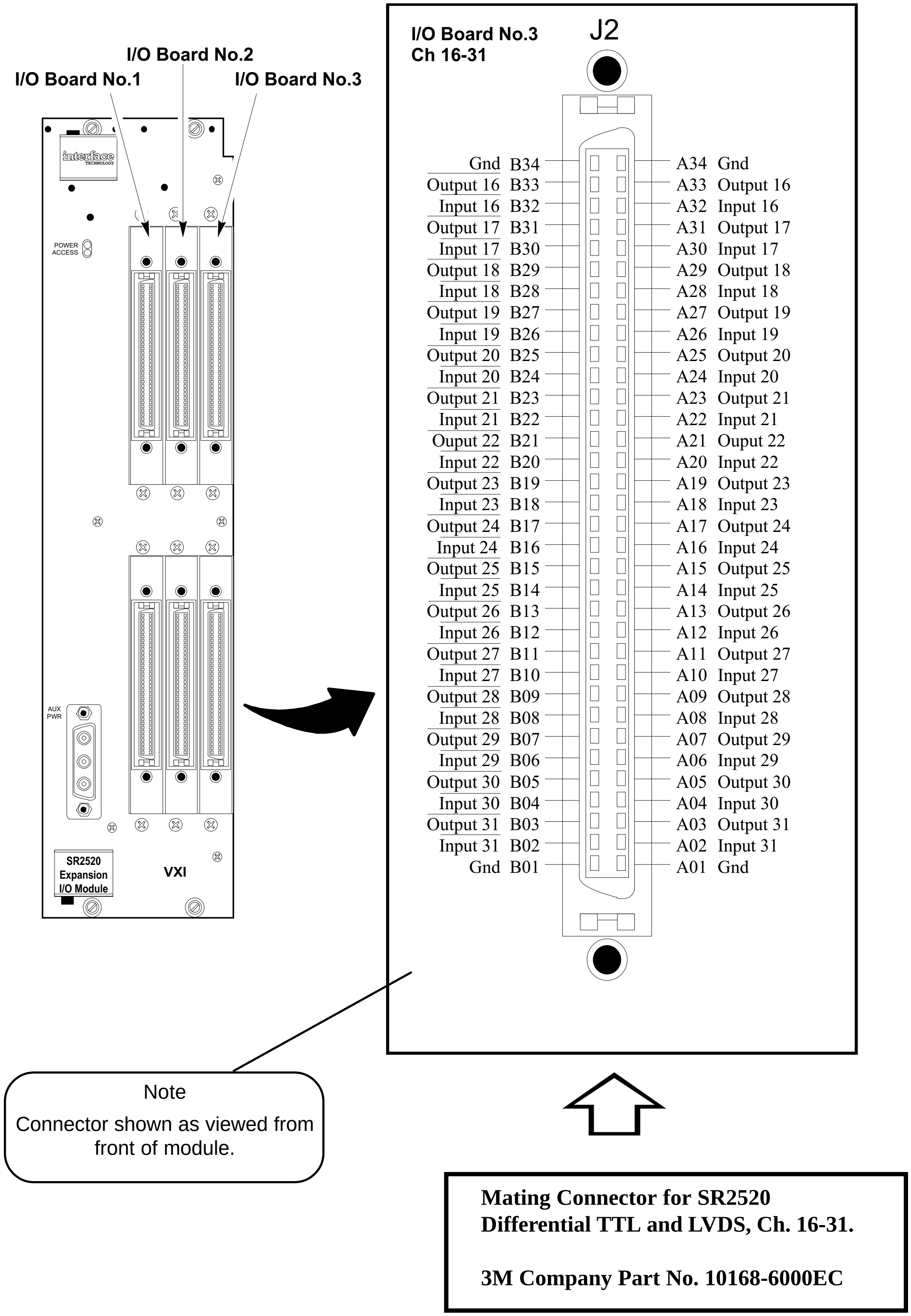
Note

Connector shown as viewed from front of module.

Mating Connector for SR2520 Differential TTL and LVDS, Ch. 00-15.

3M Company Part No. 10168-6000EC

Figure 3-10. SR2520 Signal Connector Pinouts, Differential TTL, and LVDS, Ch. 00-15.



Signal	Pin	Pin	Signal
Gnd	B34	A34	Gnd
Output 16	B33	A33	Output 16
Input 16	B32	A32	Input 16
Output 17	B31	A31	Output 17
Input 17	B30	A30	Input 17
Output 18	B29	A29	Output 18
Input 18	B28	A28	Input 18
Output 19	B27	A27	Output 19
Input 19	B26	A26	Input 19
Output 20	B25	A25	Output 20
Input 20	B24	A24	Input 20
Output 21	B23	A23	Output 21
Input 21	B22	A22	Input 21
Ouput 22	B21	A21	Ouput 22
Input 22	B20	A20	Input 22
Output 23	B19	A19	Output 23
Input 23	B18	A18	Input 23
Output 24	B17	A17	Output 24
Input 24	B16	A16	Input 24
Output 25	B15	A15	Output 25
Input 25	B14	A14	Input 25
Output 26	B13	A13	Output 26
Input 26	B12	A12	Input 26
Output 27	B11	A11	Output 27
Input 27	B10	A10	Input 27
Output 28	B09	A09	Output 28
Input 28	B08	A08	Input 28
Output 29	B07	A07	Output 29
Input 29	B06	A06	Input 29
Output 30	B05	A05	Output 30
Input 30	B04	A04	Input 30
Output 31	B03	A03	Output 31
Input 31	B02	A02	Input 31
Gnd	B01	A01	Gnd

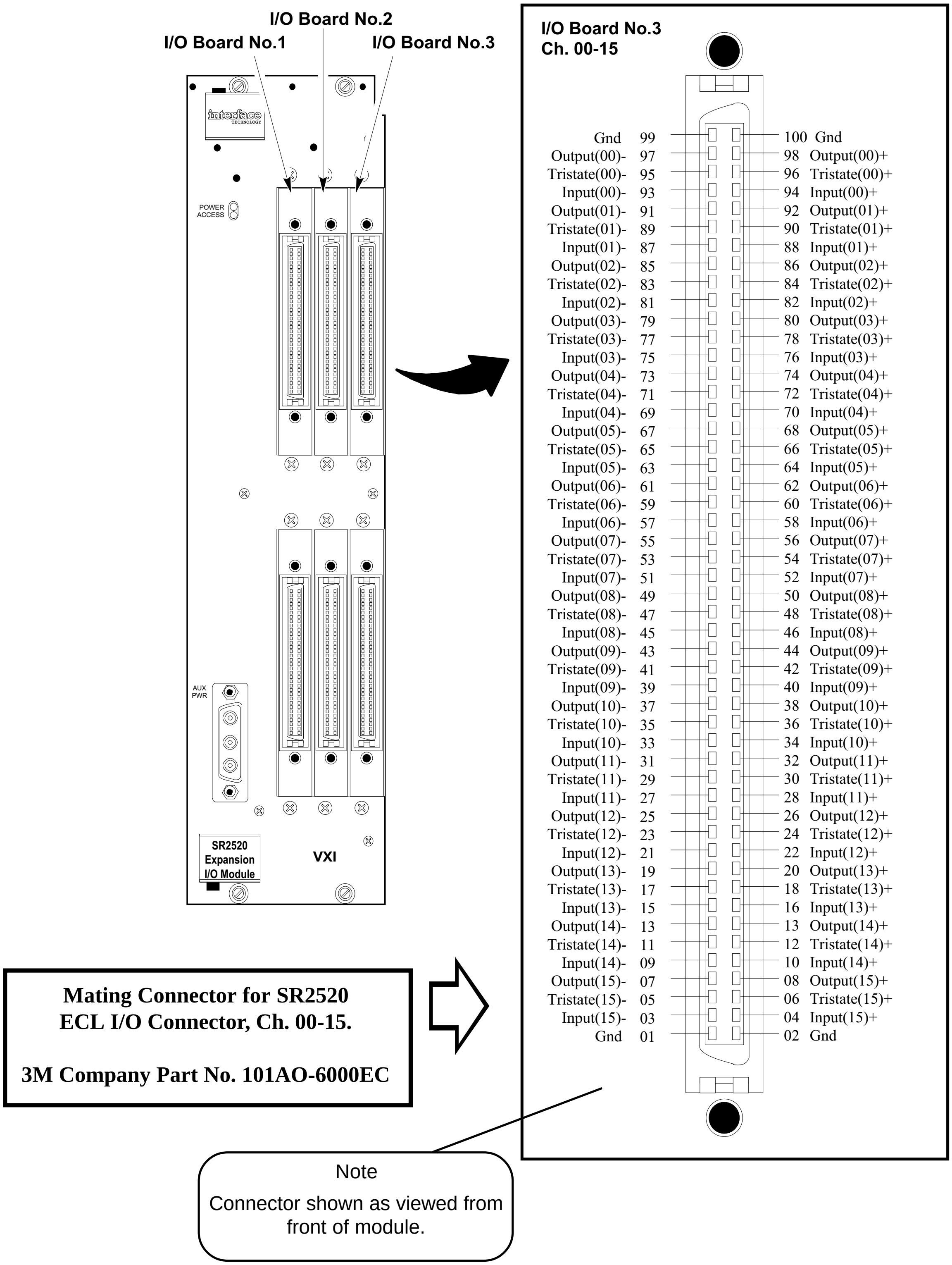
Note

Connector shown as viewed from front of module.

Mating Connector for SR2520 Differential TTL and LVDS, Ch. 16-31.

3M Company Part No. 10168-6000EC

Figure 3-11. SR2520 Signal Connector Pinouts, Differential TTL and LVDS, Ch. 16-31.



Gnd	99	100	Gnd
Output(00)-	97	98	Output(00)+
Tristate(00)-	95	96	Tristate(00)+
Input(00)-	93	94	Input(00)+
Output(01)-	91	92	Output(01)+
Tristate(01)-	89	90	Tristate(01)+
Input(01)-	87	88	Input(01)+
Output(02)-	85	86	Output(02)+
Tristate(02)-	83	84	Tristate(02)+
Input(02)-	81	82	Input(02)+
Output(03)-	79	80	Output(03)+
Tristate(03)-	77	78	Tristate(03)+
Input(03)-	75	76	Input(03)+
Output(04)-	73	74	Output(04)+
Tristate(04)-	71	72	Tristate(04)+
Input(04)-	69	70	Input(04)+
Output(05)-	67	68	Output(05)+
Tristate(05)-	65	66	Tristate(05)+
Input(05)-	63	64	Input(05)+
Output(06)-	61	62	Output(06)+
Tristate(06)-	59	60	Tristate(06)+
Input(06)-	57	58	Input(06)+
Output(07)-	55	56	Output(07)+
Tristate(07)-	53	54	Tristate(07)+
Input(07)-	51	52	Input(07)+
Output(08)-	49	50	Output(08)+
Tristate(08)-	47	48	Tristate(08)+
Input(08)-	45	46	Input(08)+
Output(09)-	43	44	Output(09)+
Tristate(09)-	41	42	Tristate(09)+
Input(09)-	39	40	Input(09)+
Output(10)-	37	38	Output(10)+
Tristate(10)-	35	36	Tristate(10)+
Input(10)-	33	34	Input(10)+
Output(11)-	31	32	Output(11)+
Tristate(11)-	29	30	Tristate(11)+
Input(11)-	27	28	Input(11)+
Output(12)-	25	26	Output(12)+
Tristate(12)-	23	24	Tristate(12)+
Input(12)-	21	22	Input(12)+
Output(13)-	19	20	Output(13)+
Tristate(13)-	17	18	Tristate(13)+
Input(13)-	15	16	Input(13)+
Output(14)-	13	13	Output(14)+
Tristate(14)-	11	12	Tristate(14)+
Input(14)-	09	10	Input(14)+
Output(15)-	07	08	Output(15)+
Tristate(15)-	05	06	Tristate(15)+
Input(15)-	03	04	Input(15)+
Gnd	01	02	Gnd

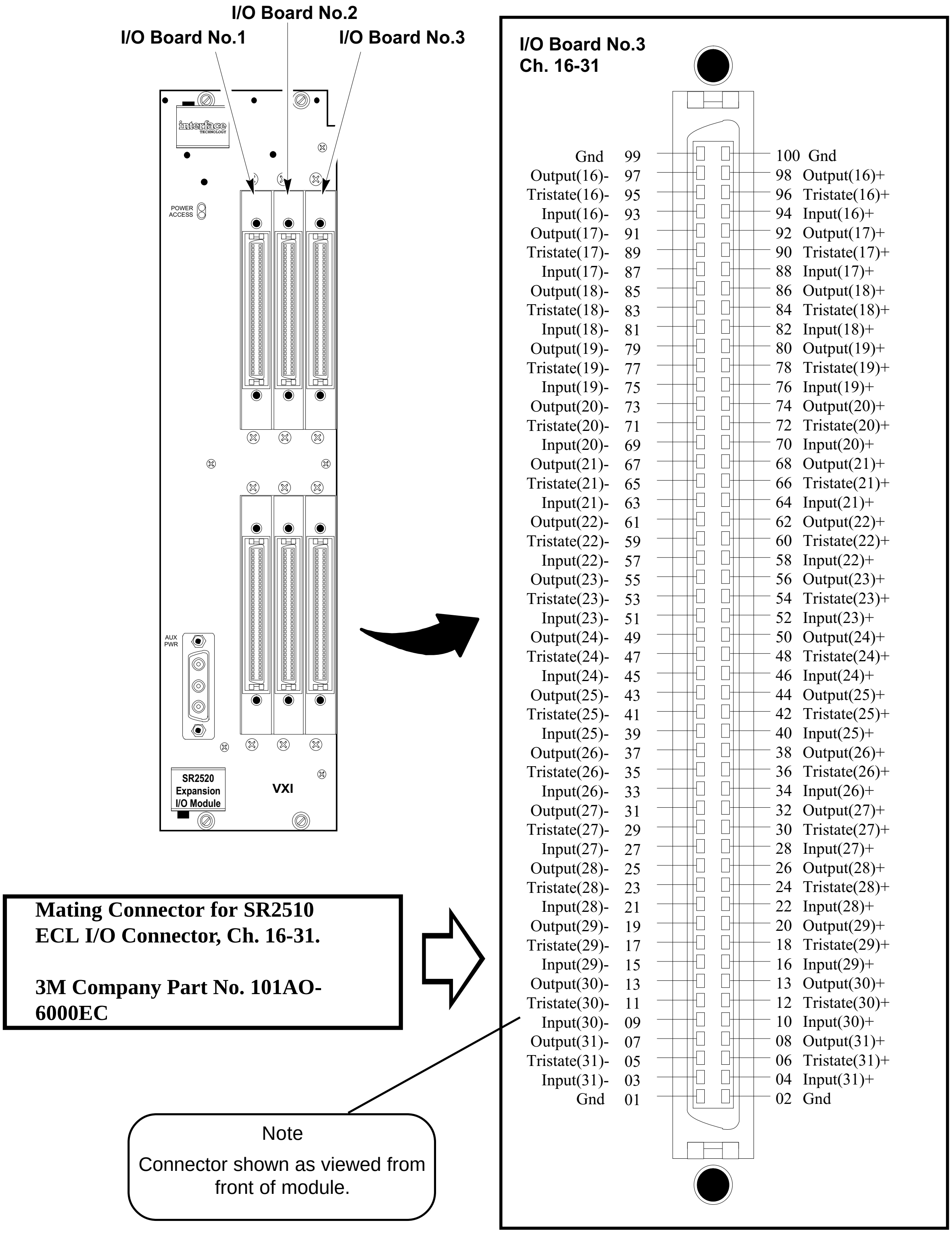
Mating Connector for SR2520
ECL I/O Connector, Ch. 00-15.

3M Company Part No. 101AO-6000EC

Note

Connector shown as viewed from front of module.

Figure 3-12. SR2520 Signal Connector Pinouts, Differential ECL, Ch. 00-15.



Gnd	99	100	Gnd
Output(16)-	97	98	Output(16)+
Tristate(16)-	95	96	Tristate(16)+
Input(16)-	93	94	Input(16)+
Output(17)-	91	92	Output(17)+
Tristate(17)-	89	90	Tristate(17)+
Input(17)-	87	88	Input(17)+
Output(18)-	85	86	Output(18)+
Tristate(18)-	83	84	Tristate(18)+
Input(18)-	81	82	Input(18)+
Output(19)-	79	80	Output(19)+
Tristate(19)-	77	78	Tristate(19)+
Input(19)-	75	76	Input(19)+
Output(20)-	73	74	Output(20)+
Tristate(20)-	71	72	Tristate(20)+
Input(20)-	69	70	Input(20)+
Output(21)-	67	68	Output(21)+
Tristate(21)-	65	66	Tristate(21)+
Input(21)-	63	64	Input(21)+
Output(22)-	61	62	Output(22)+
Tristate(22)-	59	60	Tristate(22)+
Input(22)-	57	58	Input(22)+
Output(23)-	55	56	Output(23)+
Tristate(23)-	53	54	Tristate(23)+
Input(23)-	51	52	Input(23)+
Output(24)-	49	50	Output(24)+
Tristate(24)-	47	48	Tristate(24)+
Input(24)-	45	46	Input(24)+
Output(25)-	43	44	Output(25)+
Tristate(25)-	41	42	Tristate(25)+
Input(25)-	39	40	Input(25)+
Output(26)-	37	38	Output(26)+
Tristate(26)-	35	36	Tristate(26)+
Input(26)-	33	34	Input(26)+
Output(27)-	31	32	Output(27)+
Tristate(27)-	29	30	Tristate(27)+
Input(27)-	27	28	Input(27)+
Output(28)-	25	26	Output(28)+
Tristate(28)-	23	24	Tristate(28)+
Input(28)-	21	22	Input(28)+
Output(29)-	19	20	Output(29)+
Tristate(29)-	17	18	Tristate(29)+
Input(29)-	15	16	Input(29)+
Output(30)-	13	13	Output(30)+
Tristate(30)-	11	12	Tristate(30)+
Input(30)-	09	10	Input(30)+
Output(31)-	07	08	Output(31)+
Tristate(31)-	05	06	Tristate(31)+
Input(31)-	03	04	Input(31)+
Gnd	01	02	Gnd

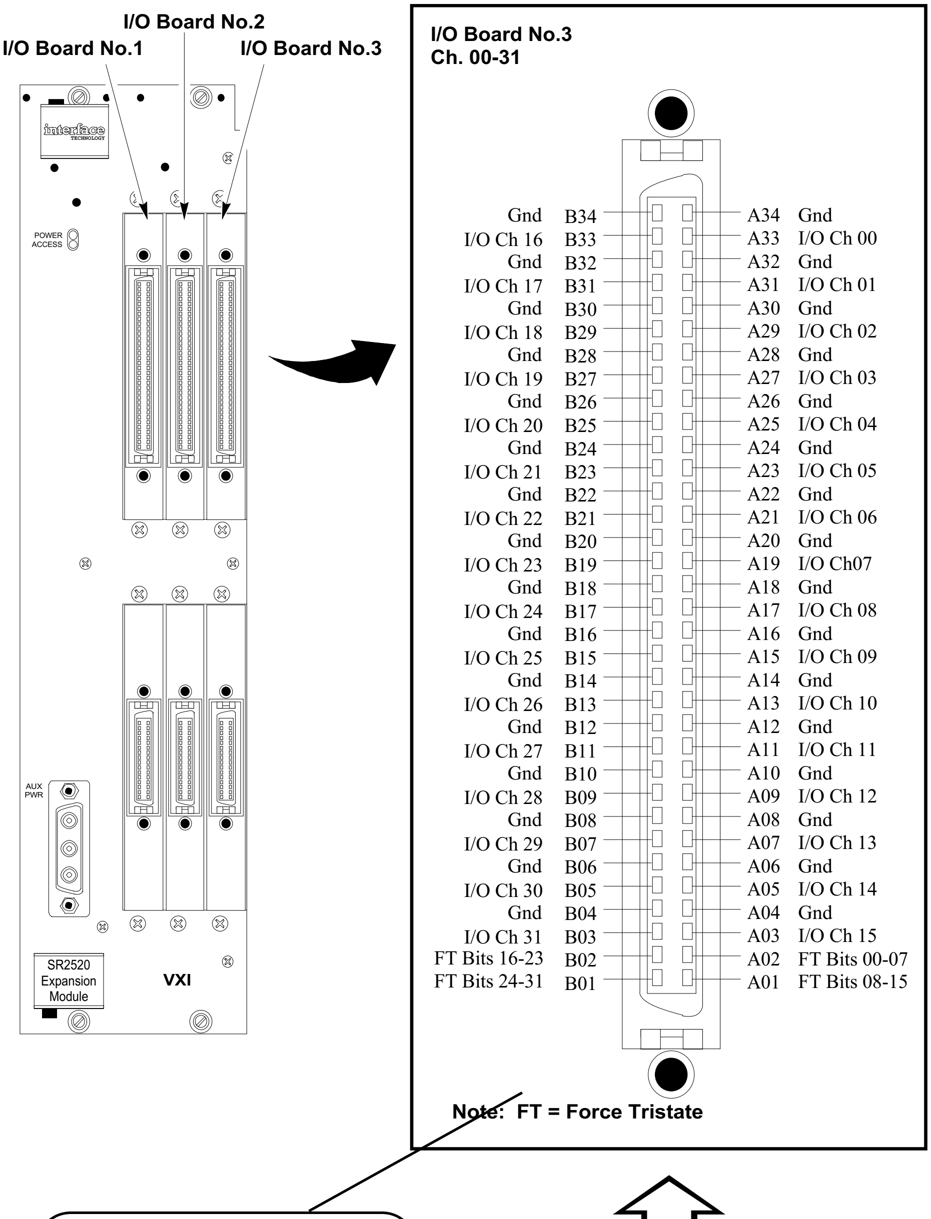
Mating Connector for SR2510 ECL I/O Connector, Ch. 16-31.

3M Company Part No. 101AO-6000EC

Note

Connector shown as viewed from front of module.

Figure 3-13. SR2520 Signal Connector Pinouts, Differential ECL, Ch. 16-31.



Gnd	B34	A34	Gnd
I/O Ch 16	B33	A33	I/O Ch 00
Gnd	B32	A32	Gnd
I/O Ch 17	B31	A31	I/O Ch 01
Gnd	B30	A30	Gnd
I/O Ch 18	B29	A29	I/O Ch 02
Gnd	B28	A28	Gnd
I/O Ch 19	B27	A27	I/O Ch 03
Gnd	B26	A26	Gnd
I/O Ch 20	B25	A25	I/O Ch 04
Gnd	B24	A24	Gnd
I/O Ch 21	B23	A23	I/O Ch 05
Gnd	B22	A22	Gnd
I/O Ch 22	B21	A21	I/O Ch 06
Gnd	B20	A20	Gnd
I/O Ch 23	B19	A19	I/O Ch07
Gnd	B18	A18	Gnd
I/O Ch 24	B17	A17	I/O Ch 08
Gnd	B16	A16	Gnd
I/O Ch 25	B15	A15	I/O Ch 09
Gnd	B14	A14	Gnd
I/O Ch 26	B13	A13	I/O Ch 10
Gnd	B12	A12	Gnd
I/O Ch 27	B11	A11	I/O Ch 11
Gnd	B10	A10	Gnd
I/O Ch 28	B09	A09	I/O Ch 12
Gnd	B08	A08	Gnd
I/O Ch 29	B07	A07	I/O Ch 13
Gnd	B06	A06	Gnd
I/O Ch 30	B05	A05	I/O Ch 14
Gnd	B04	A04	Gnd
I/O Ch 31	B03	A03	I/O Ch 15
FT Bits 16-23	B02	A02	FT Bits 00-07
FT Bits 24-31	B01	A01	FT Bits 08-15

Note: FT = Force Tristate

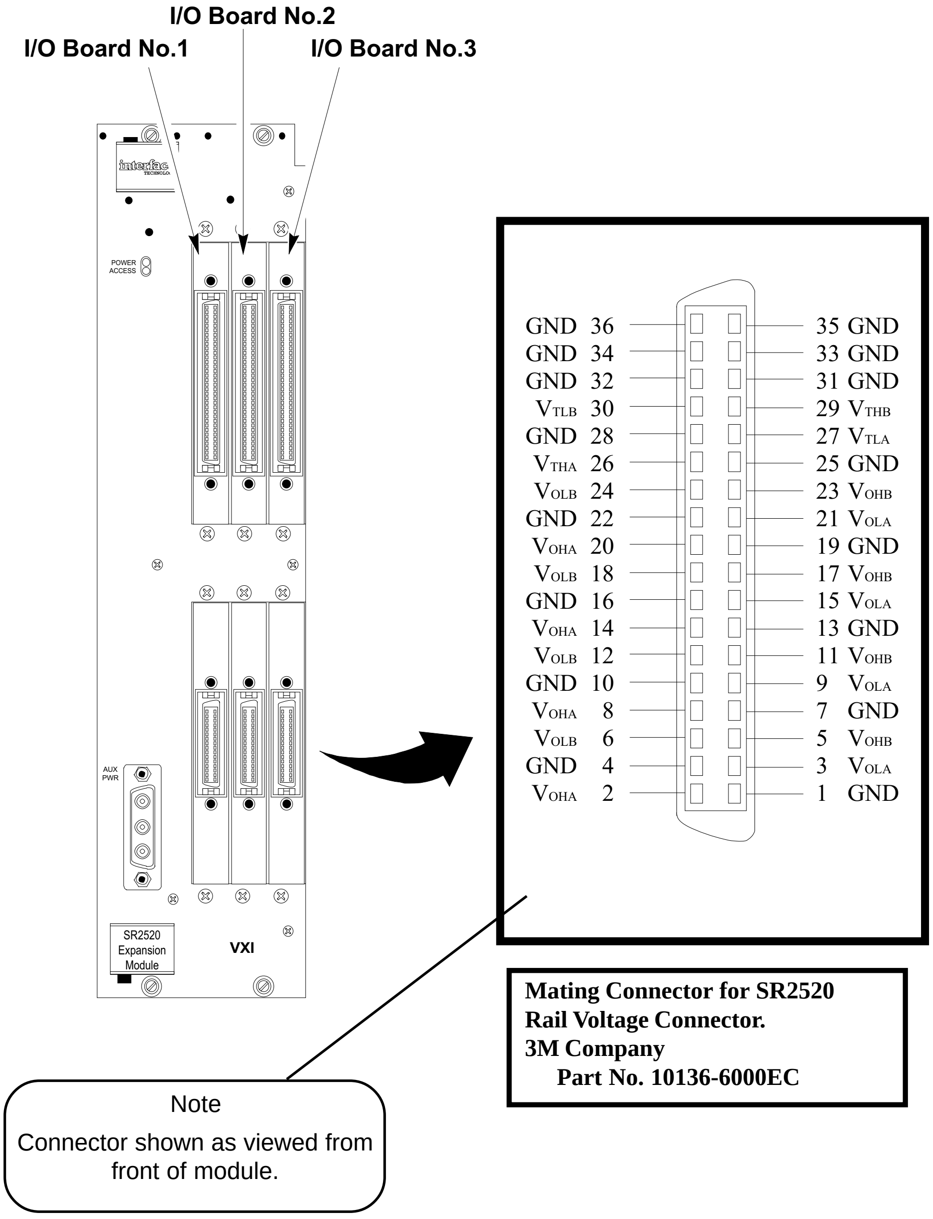
Note

Connector shown as viewed from front of module.

Mating Connector for SR2520 Variable Voltage, Ch 00-31.

3M Company Part No. 10168-6000EC

Figure 3-14. SR2520 Signal Connector Pinouts, Variable Voltage, Ch. 00-31.



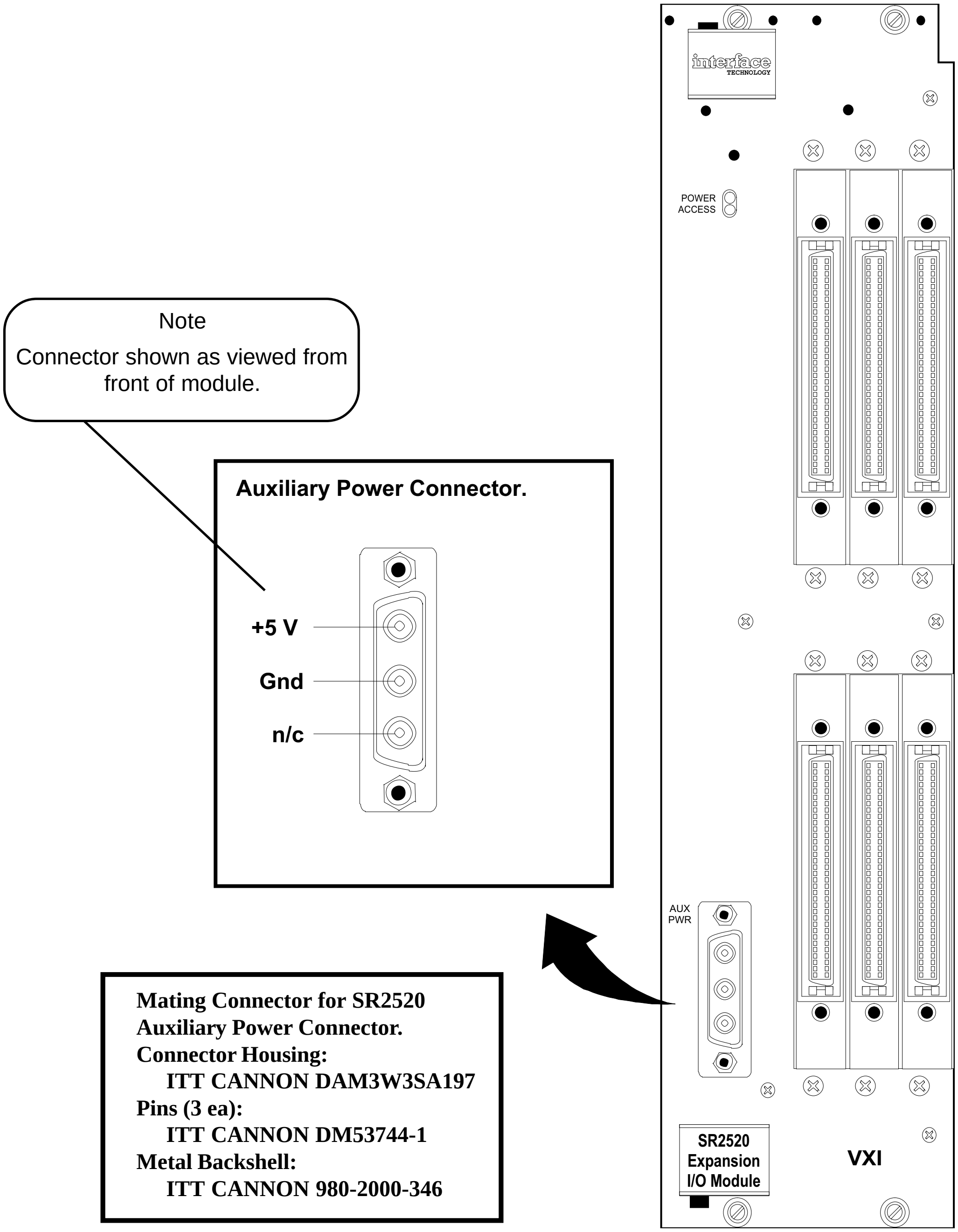
GND	36	35	GND
GND	34	33	GND
GND	32	31	GND
V_{TLB}	30	29	V_{THB}
GND	28	27	V_{TLA}
V_{THA}	26	25	GND
V_{OLB}	24	23	V_{OHB}
GND	22	21	V_{OLA}
V_{OHA}	20	19	GND
V_{OLB}	18	17	V_{OHB}
GND	16	15	V_{OLA}
V_{OHA}	14	13	GND
V_{OLB}	12	11	V_{OHB}
GND	10	9	V_{OLA}
V_{OHA}	8	7	GND
V_{OLB}	6	5	V_{OHB}
GND	4	3	V_{OLA}
V_{OHA}	2	1	GND

Mating Connector for SR2520
Rail Voltage Connector.
3M Company
Part No. 10136-6000EC

Note

Connector shown as viewed from front of module.

Figure 3-15. SR2520 Rail Voltage Connector Pinouts.



Note
Connector shown as viewed from front of module.

Mating Connector for SR2520 Auxiliary Power Connector.
Connector Housing:
ITT CANNON DAM3W3SA197
Pins (3 ea):
ITT CANNON DM53744-1
Metal Backshell:
ITT CANNON 980-2000-346

Figure 3-16. SR2520 Auxiliary Power Connector Pinouts.

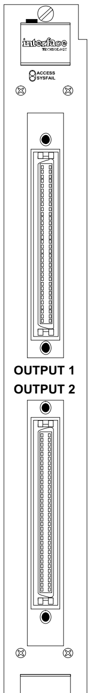
(THIS PAGE INTENTIONALLY LEFT BLANK)

Published by
Interface Technology, Inc.
300 S. Lemon Creek Drive
Walnut, CA 91789

User's Manual

RG2500 Rail Generator

VXI bus



From The Performance Leader In VXI Digital Testing ...

VXI plug & play

interface
TECHNOLOGY

Rev. 05 Apr 1998
Chg. 01 Sep 2001

RG2500 User's Manual

Record of Changes

Change No.	Date of Change	Title or Brief Description	Entered By
Rev A	Jan 97	Preliminary Release (w/o Theory of Operation)	Factory
Rev 02	Apr 97	First Official Issue	Factory
Rev 05	Apr 98	Changed Revision number only, for consistency	Factory
Chg 01	Sep 01	Changed specifications page, pg. 1-4.	Factory
Chg 2	May 03	pg 1-1 .. last para., 4th line ... changed "...-4 to 4.5 volts" to "... +3 to +4.5 volts." pg 1-4, changed specifications under "Output Range". pg 1-3, changed Table 1-1 under "Voltage Range" -- both columns. pg 2-1, changed specifications under "Output Voltages." pgs 3-2, 3-3, 3-5 ... under :HIGH, changed Parameter Definition from "Range from -4 to +5.5" to "Range from -2.9 to +5.5." pg.3-5, under :LOW, changed Parameter Definition from "Range for -4 to 4.5" to "Range from -3.0 to 4.5".	Factory

Contents

Chapter 1, General Information

About This Manual 1-1
Applicability 1-1
Supersedure Notice 1-1
Equipment Description 1-1
Controls and Indicators 1-2
Specifications 1-4

Chapter 2, Theory of Operation

Output Voltages 2-1
Block Diagram 2-1
VXI Interface 2-2
Positive and Negative Boost 2-2
Current Monitor 2-2
Power Filter 2-2
Control Block 2-2
Auto-Cal 2-2

Chapter 3, Programming

Scope of Chapter 3-1
Setting Response High Threshold Voltage 3-2
Setting Response Low Threshold Voltage 3-3
Setting Stimulus High Output Voltage 3-4
Setting Stimulus Low Output Voltage 3-5
Initiating Calibration 3-6
Connecting Voltage Output 3-7
Disconnecting Voltage Output 3-8

Chapter 4, Installation

Scope of Chapter 4-1
Unpacking and Inspection 4-1
Logical Addressing 4-1
Slot Dependencies 4-1
Backplane Jumpers 4-1

List of Figures

Fig. 1-1, RG2500 Control Locations 1-2
Fig. 1-2, RG2500 Rail Generator Showing External Cabling 1-3
Fig. 2-1, RG2500 Simplified Block Diagram 2-3
Fig. 4-1, Address Switches Set to Logical Address 12h 4-1
Fig. 4-2a, Rail Generator Cabling (1 of 2) 4-2
Fig. 4-2b, Rail Generator Cabling (2 of 2) 4-3

List of Tables

Table 1-1, RG2500 Output Voltages 1-3

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1

General Information

About This Manual

This manual provides installation and operation information for the Interface Technology RG2500 Rail Generator. Information contained herein is intended for use by technical personnel involved in the actual installation and operation of the subject instrument.

Arrangement of Contents

Information contained in this manual is arranged in four chapters, as follows:

- Chapter 1 General Information
- Chapter 2 Theory of Operation
- Chapter 4 Programming
- Chapter 5 Installation

Applicability

The information contained in this manual covers a single equipment configuration designated ***RG2500 Rail Generator***. Differences, if any, between this equipment and the actual equipment supplied are covered by Difference Data included at the front of this manual.

Supersedure Notice

This manual supersedes RG2500 User's Manual, Rev. 1.

Equipment Description

See Fig.1-1. The RG2500 Rail Generator is a programmable power supply used to provide operating voltages to the SR2510 Timing / Control / I/O Module and SR2520 I/O Module when these modules are configured for programmable (variable voltage) I/O operation.

The RG2500 receives operating voltages and control commands from the host computer, and supplies one or two SR2500 I/O modules with eight individual output voltages, each of which is separately programmable over a range of -1.5 to 7.0 volts, or -3 to 4.5 volts, as listed in Table 1-1. Two 100-pin output connectors are provided on the RG2500 front panel. Operating voltages are routed from the RG2500 by means of a special breakout cable that splits the output lines from a single RG2500 output connector into three separate cables for use by the SR2500 system. Each of the three cables supplies programmable voltages to one of three 32-channel I/O boards within the SR2510 or SR2520 modules. Pinouts for the RG2500 Rail Generator connectors and the breakout cable are shown in Figure 4-2.

Controls and Indicators

See Fig. 1-1. All connectors and LEDs for the RG2500 Rail Generator are located on the front panel.

LEDs

There are two LEDs located at the top of the RG2500 module.

- **ACCESS** (yellow) Illuminates briefly each time the SR2510 Timing / Control / I/O Module communicates with the RG2500.
- **SYSFAIL** (red) Off during normal operation. During the power-up sequence, this indicator is lit until the internal self-test is complete. The indicator remains lit if the self-test fails.

Connectors

Two 100-pin connectors are provided on the RG2500 front panel. Output voltages from these connectors is listed in Table 1-1. Refer to Chapter 4 for connector pinout information.

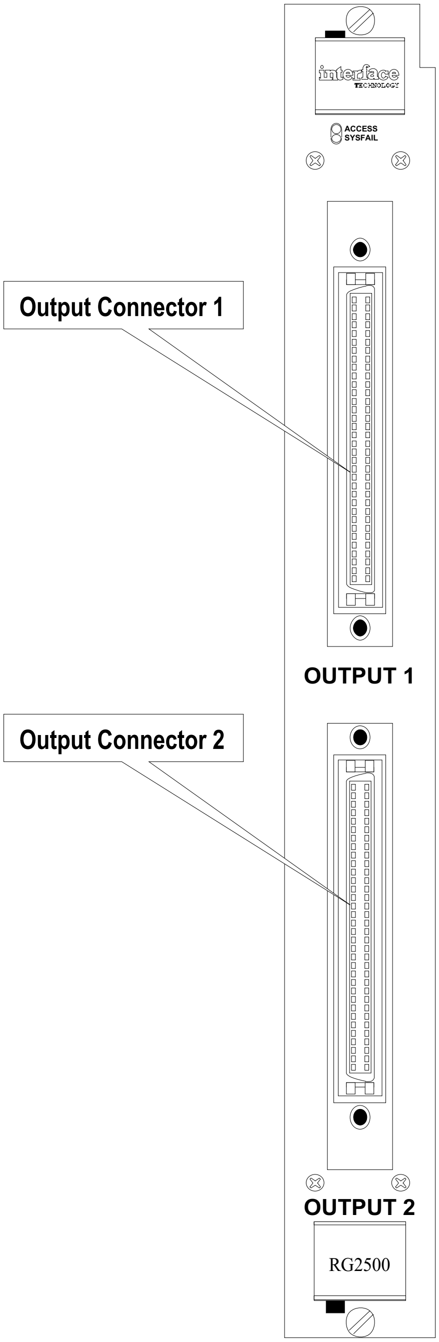


Figure 1-1.
RG2500 Control Locations.

Table 1-1. RG2500 Output Voltages.

Connector 1			Connector 2		
Name	**Description**	**Voltage Range**	**Name**	**Description**	**Voltage Range**
VOHA^{1}	Output Voltage High A	-1.5 V to +7.0 V	VOHA^{2}	Output Voltage High A	-1.5 V to +7.0 V
VOHB^{1}	Output Voltage High B	-1.5 V to +7.0 V	VOHB^{2}	Output Voltage High B	-1.5 V to +7.0 V
VOLA^{1}	Output Voltage Low A	-3.0 V to +4.5 V	VOLA^{2}	Output Voltage Low A	-3.0 V to +4.5 V
VOLB^{1}	Output Voltage Low B	-3.0 V to +4.5 V	VOLB^{2}	Output Voltage Low B	-3.0 V to +4.5 V
VTHA^{1}	Threshold Voltage High A	-2.9 V to +5.5 V	VTHA^{2}	Threshold Voltage High A	-2.9 V to +5.5 V
VTHB^{1}	Threshold Voltage High B	-2.9 V to +5.5 V	VTHB^{2}	Threshold Voltage High B	-2.9 V to +5.5 V
VTLA^{1}	Threshold Voltage Low A	-2.9 V to +5.5 V	VTLA^{2}	Threshold Voltage Low A	-2.9 V to +5.5 V
VTLB^{1}	Threshold Voltage Low B	-2.9 V to +5.5 V	VTLB^{2}	Threshold Voltage Low B	-2.9 V to +5.5 V

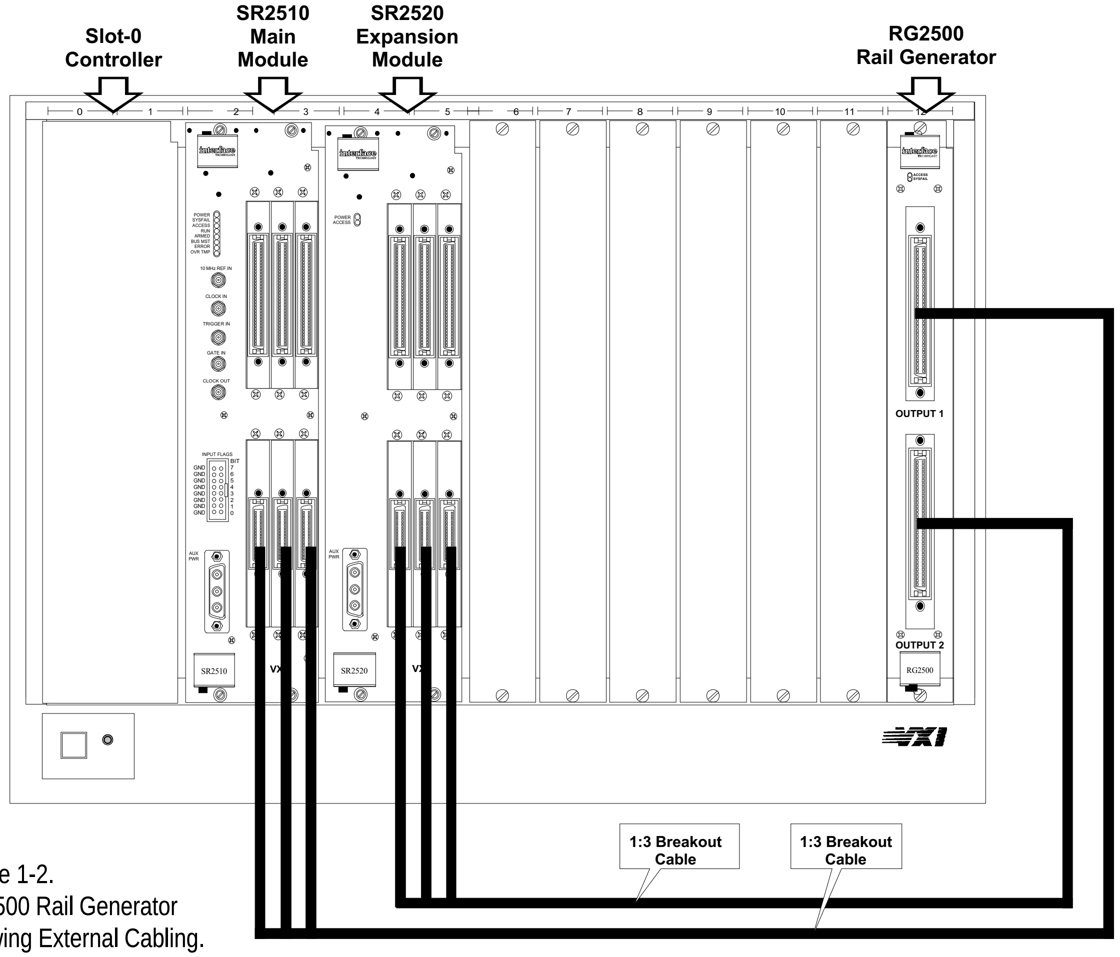


Figure 1-2.
RG2500 Rail Generator
Showing External Cabling.

*RG2500 SPECIFICATIONS**

Output Voltages:

Voltage	Description	Output Range
V_{OHA}	Output Voltage High, A	-1.5 to +7.0 volts
V_{OHB}	Output Voltage High, B	-1.5 to +7.0 volts
V_{OLA}	Output Voltage Low, A	-3.0 to +4.5 volts
V_{OLB}	Output Voltage Low, B	-3.0 to +4.5 volts
V_{THA}	Threshold Voltage High, A	-2.9 to +5.5 volts
V_{THB}	Threshold Voltage High, B	-2.9 to +5.5 volts
V_{TLA}	Threshold Voltage Low, A	-2.9 to +5.5 volts
V_{TLB}	Threshold Voltage Low, B	-2.9 to +5.5 volts

Note: The above voltages are available at each of the two output connectors.

VXI Specifications

Interface Compatibility:

RG2500	Register-based, servant only (controlled by SR2510)
VXI Revision	1.4
Size	C-size, single slot
Configuration	Static

Power Requirements:

+24.0 volts	1.0 A	24W max.
+12.0 volts	1.0 A	12W max.
+5.0 volts	7.0 A	35W max.
-5.2 volts	5.0 A	26W max.
-12.0 volts	1.0 A	12W max.
-24.0 volts	1.0 A	24W max.

Cooling Requirements:

Power	40W max. (15W typical)
Airflow	4L/sec @ 0.2mm water pressure for 10° temperature rise

Environmental Specifications:

Temperature	Storage = -40°C to +75°C
	Operating = 0°C to +45°C

Humidity	5% to 95% relative, noncondensing

* Specifications subject to change without notice.

CHAPTER 2

Theory of Operation

The RG2500 Rail Generator supplies up to 16 independently programmable output voltages to the SR2500 variable voltage module(s). The voltages are supplied via two output connectors (Output 1 and Output 2) located on the front panel of the Rail Generator; each connector supplies four rail voltages and four threshold voltages.

Output Voltages

The ***high rail voltages*** are:

- VOHA^{1} -1.5 to +7.0 volts (connector 1)
- VOHB^{1} -1.5 to +7.0 volts (connector 1)
- VOHA^{2} -1.5 to +7.0 volts (connector 2)
- VOHB^{2} -1.5 to +7.0 volts (connector 2)

The ***low rail voltage***s are:

- VOLA^{1} -3.0 to +4.5 volts (connector 1)
- VOLB^{1} -3.0 to +4.5 volts (connector 1)
- VOLA^{2} -3.0 to +4.5 volts (connector 2)
- VOLB^{2} -3.0 to +4.5 volts (connector 2)

The ***high threshold voltage***s are:

- VTHA^{1} -2.9 to +5.5 volts (connector 1)
- VTHB^{1} -2.9 to +5.5 volts (connector 1)
- VTHA^{2} -2.9 to +5.5 volts (connector 2)
- VTHB^{2} -2.9 to +5.5 volts (connector 2)

The ***low threshold voltages*** are:

- VTLA^{1} -2.9 to +5.5 volts (connector 1)
- VTLB^{1} -2.9 to +5.5 volts (connector 1)
- VTLA^{2} -2.9 to +5.5 volts (connector 2)
- VTLB^{2} -2.9 to +5.5 volts (connector 2)

Block Diagram

See Fig. 2-1. The RG2500 Rail Generator accepts the following input voltages from the VXI chassis: ±24 volts, ±12 volts, -5.2 volts, and +5.0 volts. Switching voltage regulators within the RG2500 convert these input voltages to programmable output voltages, as indicated above. The combined input power to the RG2500 is approximately 130 watts.

VXI Interface

The VXI interface consists of a proprietary ASIC, an address selection switch, and address and data bus buffers that provide a register-based A32/D32 VXI interface between the VXI bus and the RG2500.

During operation, the RG2500 requests a 1 MB block of memory from the resource manager. Although direct (one way) communication from the host computer to the rail generator is possible, all RG2500 commands from the host computer are first sent to the SR2510 Timing / Control / I/O Module where they are parsed. The RG2500 will then, in turn, be programmed by the SR2510 acting as bus master. This eliminates the need for having to learn the register-based memory map of the RG2500 and provides a single programming point within the system ... i.e., the SR2510.

Positive and Negative Boost

These boost regulators are switching regulators that boost the +5 Vdc and -5.2 Vdc to +24 Vdc and -24 Vdc, respectively. These regulators are activated whenever the amount of current drawn from the ±24 Vdc busses approaches 1-ampere.

Current Monitor

The current load of the six VXI power supplies is monitored and all outputs are disabled if excessive current is drawn from any one of the rail outputs. The over-current trip-point is set to approximately 1-ampere. If an over-current condition should occur, the power rails will automatically be restored to normal operation by the internal firmware, when the over-current condition no longer exists.

After an over-current condition has occurred, the firmware checks every 10 seconds to determine whether or not the overload still exists. If so, the power rails will remain off, otherwise if the overload no longer exists the rails are restored to normal operation.

Power Filter

The VXI power supplies are fused and filtered. Fuses are soldered directly to the RG2500 PCB. Fuse ratings are as follows:

- +5 Vdc, fused at 7 A
- +12 Vdc, fused at 1 A
- +24 Vdc, fused at 1 A
- -5.2 Vdc, fused at 5 A
- -12 Vdc, fused at 1 A
- -24 Vdc, fused at 1 A

Control Block

The control block accepts input from the current monitor and shuts down all outputs if an over-current condition occurs.

Auto-Cal

The RG2500 has an A/D converter that monitors output voltages, thus allowing the RS2500 to self-calibrate each of the 16 outputs against an internal on-board precision voltage reference. The Auto-Cal function requires approximately 45 seconds and is initiated by sending the appropriate command to the SR2510 module. During the calibration process,

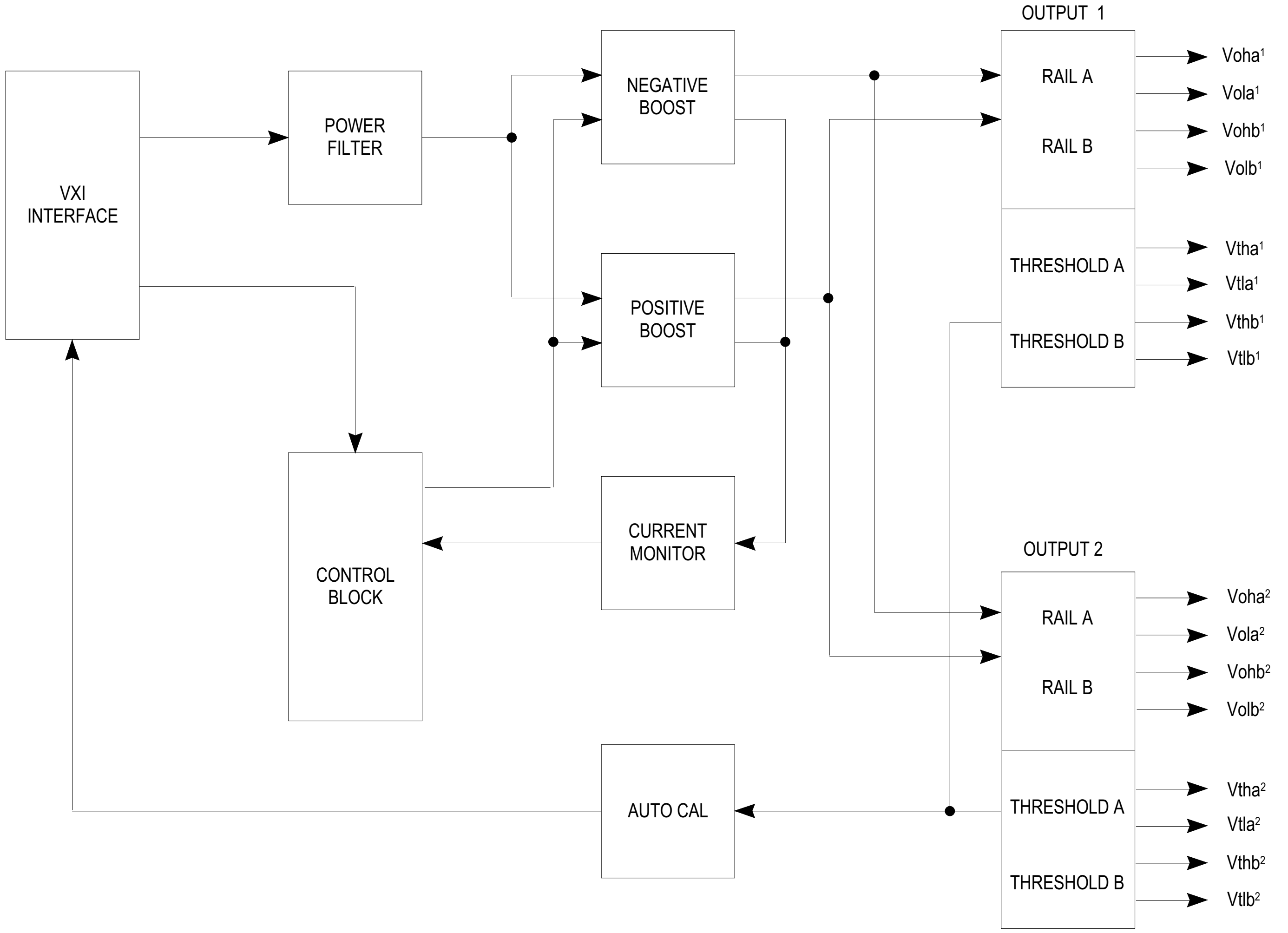


Figure 2-1.
RG2500 Simplified Block Diagram.

communication occurs between the SR2510 module and the RS2500 Rail Generator over the VXI bus. Issuing other VXI bus system commands during this such communication will, unnecessarily, lengthen the calibration time.

Threshold voltages are generated by 12-bit programmable DACs (digital-to-analog converter). The SR2500 output voltages are generated using the DAC outputs to drive push/pull power amplifiers. Each output pin is rated to source/sink 50 mA of current from its respective power amplifier (VOH or VOL).

Note

The SR2500 variable voltage outputs use a high-speed switch to switch between the Voh and Vol rail levels, depending on whether a logic-l or logic-0 is being output.

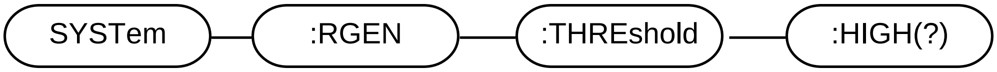
CHAPTER 3

Programming

Scope of Chapter

This chapter contains the programming command routines that are unique to the RG2500 Rail Generator. The routines in this chapter should be used in conjunction with the general programming commands and command routines contained in the SR2500 main manual.

SETTING RESPONSE HIGH THRESHOLD VOLTAGE



SYSTem — :RGEN — :THREshold — :HIGH(?)

The SYST:RGEN:THREshold:HIGH command sets the High Threshold Voltage on the RG2500. Parameters for this command are as follows:

:RGEN Number of Rail Generator

Parameter Definition 1-9 (default 1)

:THREshold Selects which voltage set

Parameter Definition (A1 | A2 | B1 | B2)

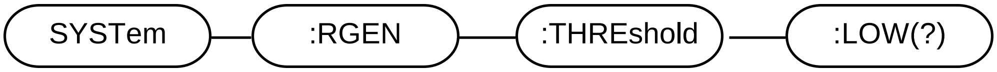
:HIGH

Parameter Definition Range from -2.9 to +5.5

Examples:

```
SYST:RGEN1:THREshold A1:HIGH 2.0
SYST:RGEN1:THREshold A1:HIGH?

2.000000e+00 volts
```

SETTING RESPONSE LOW THRESHOLD VOLTAGE



The SYST:RGEN:THREshold:LOW command sets the Low Threshold Voltage on the RG2500. Parameters for this command are as follows:

:RGEN Number of Rail Generator

Parameter Definition 1-9 (default 1)

:THREshold Selects which voltage set

Parameter Definition (A1 | A2 | B1 | B2)

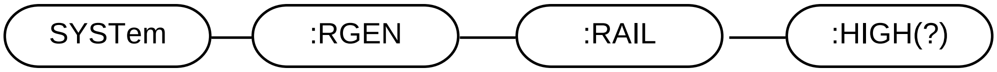
:LOW

Parameter Definition Range from -2.9 to +5.5

Examples

```
SYST:RGEN1:THREshold A1:LOW 0.8
SYST:RGEN1:THREshold A1:LOW?

8.000000e-01 volts
```

SETTING STIMULUS HIGH OUTPUT VOLTAGE



SYSTem — :RGEN — :RAIL — :HIGH(?)

The SYST:RGEN:RAIL:HIGH command sets the High Stimulus Voltage on the RG2500. Parameters for this command are as follows:

:RGEN Number of Rail Generator

Parameter Definition 1-9 (default 1)

:RAIL Selects which voltage set

Parameter Definition (A1 | A2 | B1 | B2)

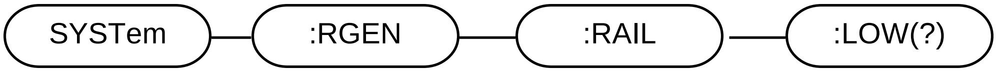
:HIGH

Parameter Definition Range from -1.5 to +7

Examples

```
SYST:RGEN1:RAIL A1:HIGH 5
SYST:RGEN1:RAIL A1:HIGH?

5.000000e+00 volts
```

Setting Stimulus Low Output Voltage



The SYST:RGEN:RAIL:HIGH command sets the Low Stimulus Voltage on the RG2500. Parameters for this command are as follows:

:RGEN Number of Rail Generator

Parameter Definition 1-9 (default 1)

:RAIL Selects which voltage set

Parameter Definition (A1 | A2 | B1 | B2)

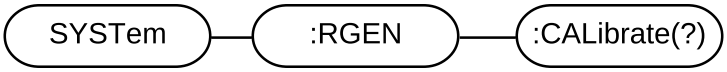
:LOW

Parameter Definition Range from -3.0 to +4.5

Examples

```
SYST:RGEN1:RAIL A1:LOW 0
SYST:RGEN1:RAIL A1:LOW?

0.000000e+00 volts
```

Initiating Calibration



The SYST:RGEN:CALibrate command calibrates the RG2500. Parameters for this command are as follows:

:RGEN Number of Rail Generator

Parameter Definition 1-9 (default 1)

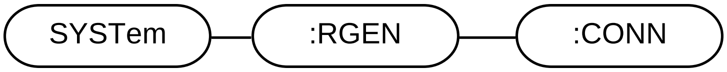
Examples SYST:RGEN 1:CAL
SYST:RGEN 3:CAL?

Response 0 | 1

Note

The SYST:REGEN n:CAL? command will respond with a '1' if the rail generator has been calibrated since power on and with a '0' if it has not been calibrated since power on.

Connecting Voltage Output



The SYST:RGEN:CONN command connects the output voltages on the RG2500 connectors. Parameters for this command are as follows:

:RGEN Number of Rail Generator

Parameter Definition 1-9 (default 1)

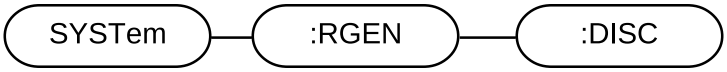
:CONN Number of connector on the Rail Generator

Parameter Definition 1 | 2

Examples

```
SYST:RGEN 3:CONN 1
SYST:RGEN 1:CONN 2
```

DISCONNECTING VOLTAGE OUTPUT



The SYST:RGEN:DISC command disconnects the output voltages on the RG2500 connectors. Parameters for this command are as follows:

:RGEN Number of Rail Generator

Parameter Definition 1-9 (default 1)

:DISC Number of connector on the Rail Generator

Parameter Definition 1 | 2

Examples

```
SYST:RGEN 3:DISC 1
SYST:RGEN 1:DISC 2
```

CHAPTER 4

Installation

Scope of Chapter

This chapter contains instructions for unpacking, inspecting, installing, and checking out the RG2500 Rail Generator.

Unpacking and Inspection

Your RG2500 was thoroughly inspected and tested before shipment from the factory and is ready for immediate operation once all installation procedures have been completed. Carefully remove the instrument from its shipping carton and check for any obvious damage that may have occurred during shipment. If damage is found, report it to the freight carrier immediately. Interface Technology is not liable for damage that may have occurred during transit. Save the shipping carton and all packing material for possible future use.

Logical Addressing

Before installation, the logical address for the RG2500 must be set according to the requirements of the Slot-0 Controller. The address switches are numbered from one to eight. Switch 1 corresponds to the least significant bit (LSB) of the logical address. The address is entered in binary, where an ON switch sets the corresponding bit to "0", see Fig. 4-1.

Slot Dependencies

The RG2500 has no slot dependencies.

Backplane Jumpers

The RG2500 does not use any of the IACK or BG3 signals. These signals are passed through. The user may remove or install the jumpers for this VXI slot, as required.

Note

The logical address of the RG2500 Rail Generator must be set to a higher value than the logical address of the SR2510 (and SR2520, if used) with which it is used in conjunction with.

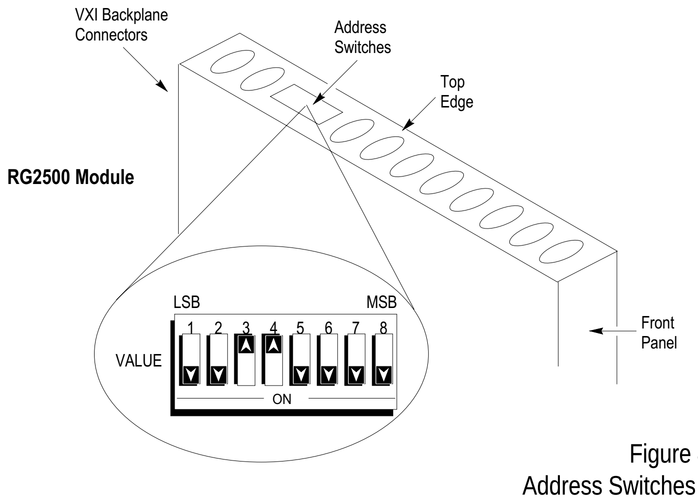


Figure 4-1.
Address Switches Set
to Logical Address 12.

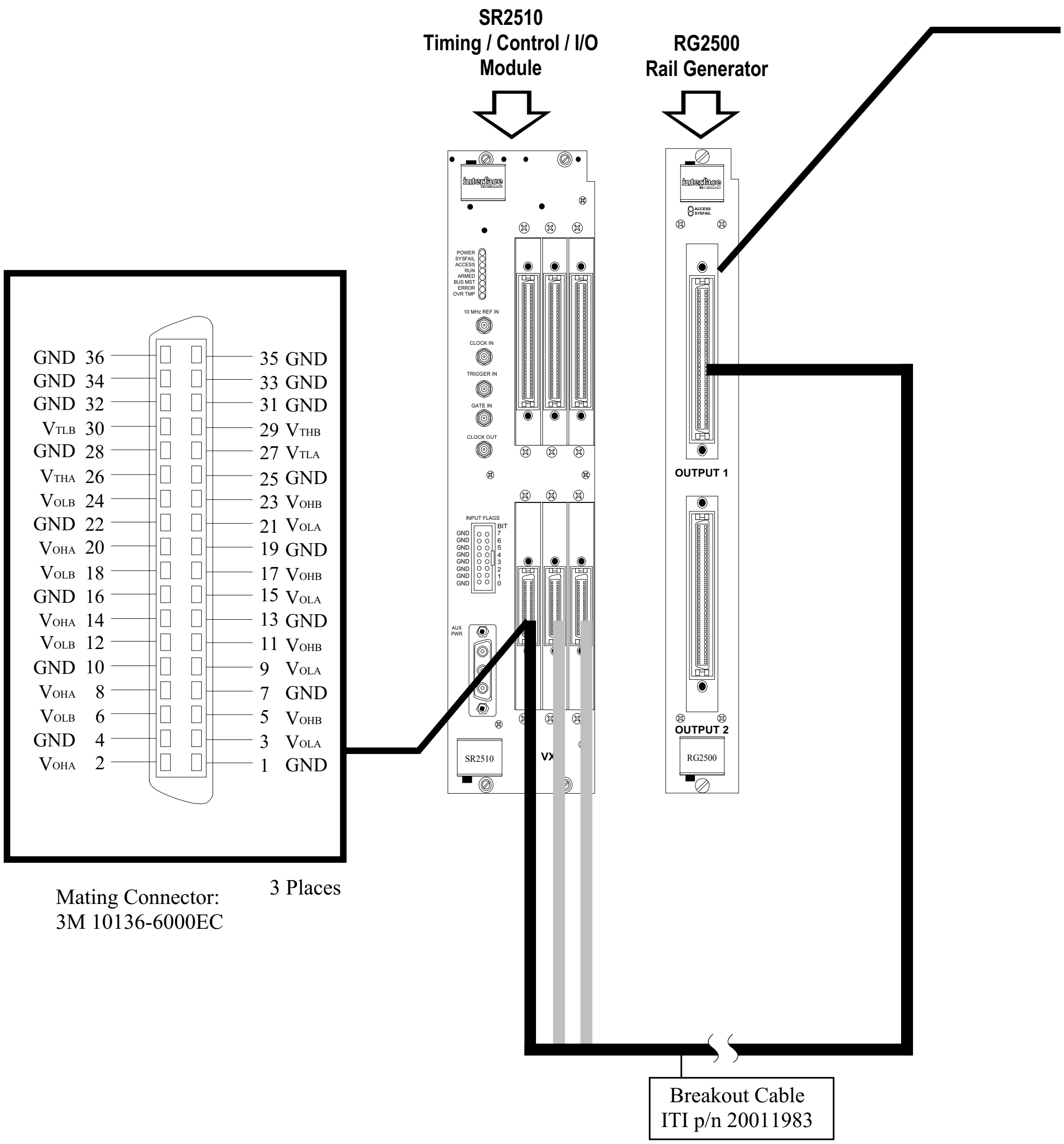


Figure 4-2a.
Rail Generator Cabling (1 of 2).

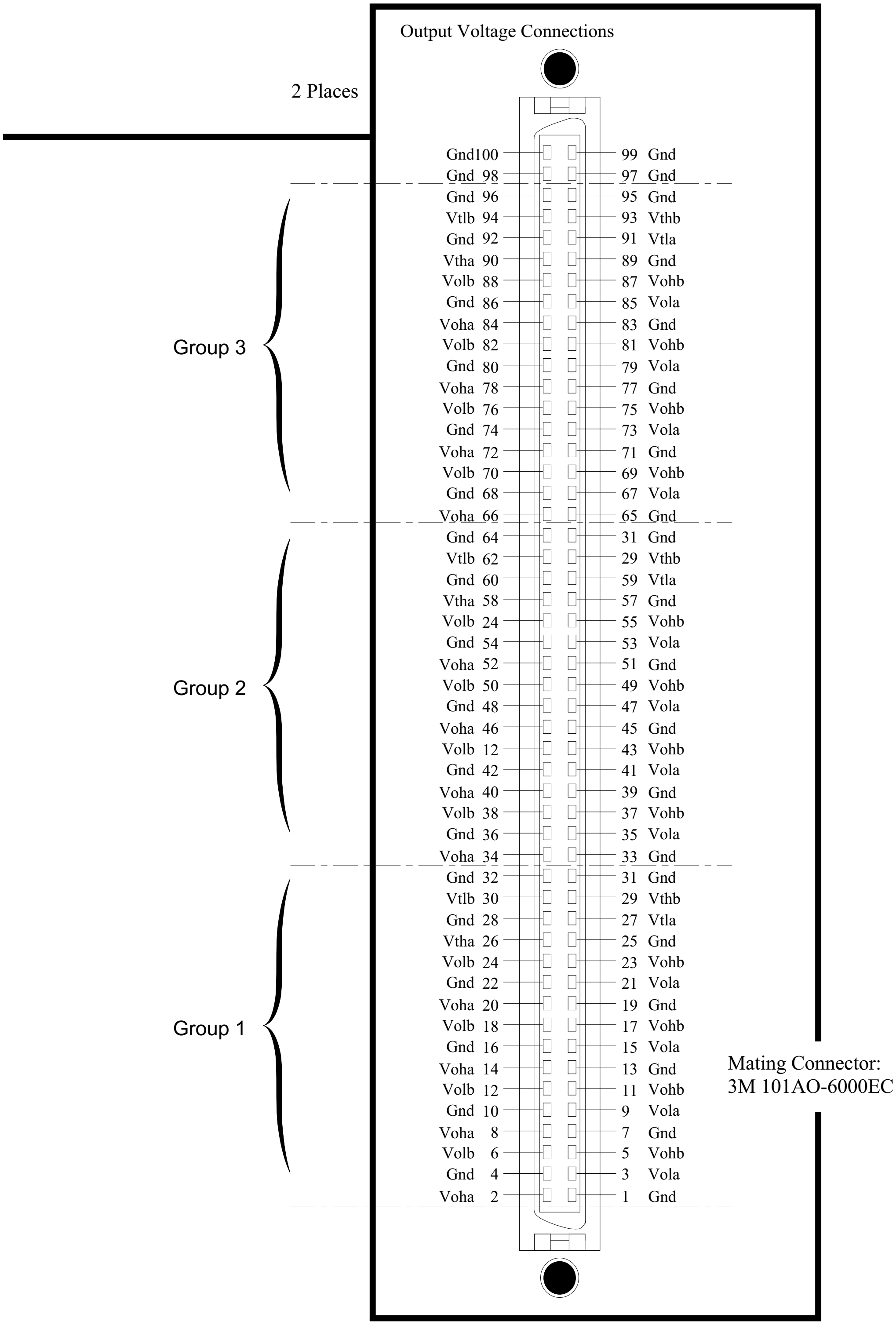


Figure 4-2b.
Rail Generator Cabling (2 of 2).

THIS PAGE INTENTIONALLY LEFT BLANK

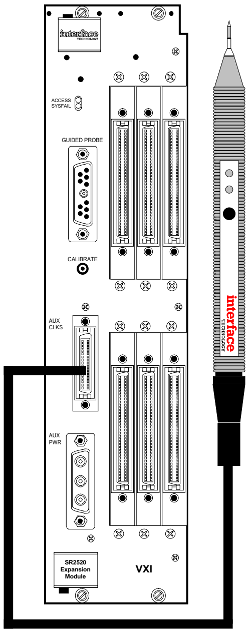
Published by
Interface Technology, Inc.
300 S. Lemon Creek Drive
Walnut, CA 91789

User's Manual

SR2520 w/Guided Probe Option

VXI bus

From The Performance Leader
In VXI Digital Testing ...



VXI plug & play

interface
TECHNOLOGY

Rev. 5.0
April, 1998

SR2520 w/Guided Probe User's Manual

Record of Changes			
Change No.	Date of Change	Title or Brief Description	Entered By
Rev 05	Apr 98	Changed Revision number only, for consistency	Factory

Contents

Chapter 1, General Information

About This Manual 1-1
Arrangement of Contents 1-1
Applicability 1-1
Supersedure Notice 1-1
Equipment Description 1-1
Features 1-2
Controls and Indicators 1-3
Specifications 1-4

Chapter 2, Theory of Operation

General 2-1
Probe Circuitry 2-1
Main Logic 2-1
Extra User Clocks 2-2
Tags 2-2

Chapter 3, Programming

General 3-1
Guided Probe Commands 3-1
SCPI Command Key 3-2
Create and Initialize Guided Probe Fields 3-3
Detecting Probe Switch Position (down/not down) 3-4
Measuring a Voltage With The Probe 3-5
Calibrating the Guided Probe 3-6
Setting Up The Guided Probe 3-7
Clearing The Guided Probe Field 3-8
Settting The High Threshold 3-9
Setting The Low Threshold 3-10
Setting TTL Levels For The Thresholds 3-11
Setting ECL Levels For The GP Thresholds 3-12

Chapter 4, Installation

Scope of Chapter 4-1
Unpacking and Inspection 4-1
Installation 4-1
Logical Addressing 4-1
Slot Dependencies 4-2
Backplane Jumpers 4-2

List of Figures

Fig. 1-1, Guided Probe Add-In Option to SR2520 Module 1-2
Fig. 1-2, Conrols and Indicators 1-3
Fig. 2-1, Functional Block Diagram, Guided Probe Logic 2-3
Fig. 2-2, Functional Diagram, Probe 2-3
Fig. 4-1, Address Switches Set to Logical Address 12h 4-1
Fig. 4-2, SR2520 w/Guided Probe I/O and Aux. Power Pinouts 4-3
Fig. 4-3, SR2520 w/Guided Probe Aux. Clock Pinouts 4-4
Fig. 4-4, Installation of SR2520 w/Guided Probe Module 4-5

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1

General Description

About This Manual

This manual provides installation and operation information for the Interface Technology SR2520 w/ Guided Probe Option. Information contained herein is intended for use by technical personnel involved in the actual installation and operation of the subject instrument.

Arrangement of Contents

Information contained in this manual is arranged in four chapters, as follows:

- Chapter 1 General Information
- Chapter 2 Theory of Operation
- Chapter 3 Programming
- Chapter 4 Installation

Applicability

The information contained in this manual covers a single equipment configuration designated ***SR2520 w/Guided Probe Option***. Differences, if any, between this equipment and the actual equipment supplied are covered by Difference Data included at the front of this manual.

Supersedure Notice

This manual supersedes portions of SR2500GP Guided Probe User's Manual, Rev.03 dated Dec. 96 and all previous issues of that publication.

Equipment Description.

The SR2520 Guided Probe Option provides added capability to read test points (nodes) on the UUT to determine pass/fail conditions. It is capable of testing and detecting high, low and indeterminate states and can also measure analog voltages. Upon determination of the pass/fail state, the guided probe stores the UUT response along with the compare results for later readout. The probe has an active input, which minimizes circuit loading and serves to "condition" the UUT signal before routing it to the guided probe logic in the SR2520. Located on the probe body is an ENTER button (used to trigger or continue test execution)

The SR2500GP Guided Probe is supplied as a factory installed add-in option to the SR2520 Expansion Module, see Figure 1-1.

Features:

- o Dual Threshold Comparators.
- o Initiate Button Located on Probe Body.
- o User Replaceable Probe Tip.
- o Enable/Disable Probe Testing Per Vector.
- o "Learns" Known Good Responses From UUT.
- o Active Input Buffer.
- o Record High, Low, Indeterminate and Error Information per Vector.
- o Record Vector Count/Time Tag.
- o 24-Bit Continuous Cycle Counter.
- o Detect Indeterminate / Float / Tristate Logic Input.
- o Programmable Input Threshold.
- o Hardware Signature Generation (Polynomial CRC).
- o Programmable Sample Strobe/Window.

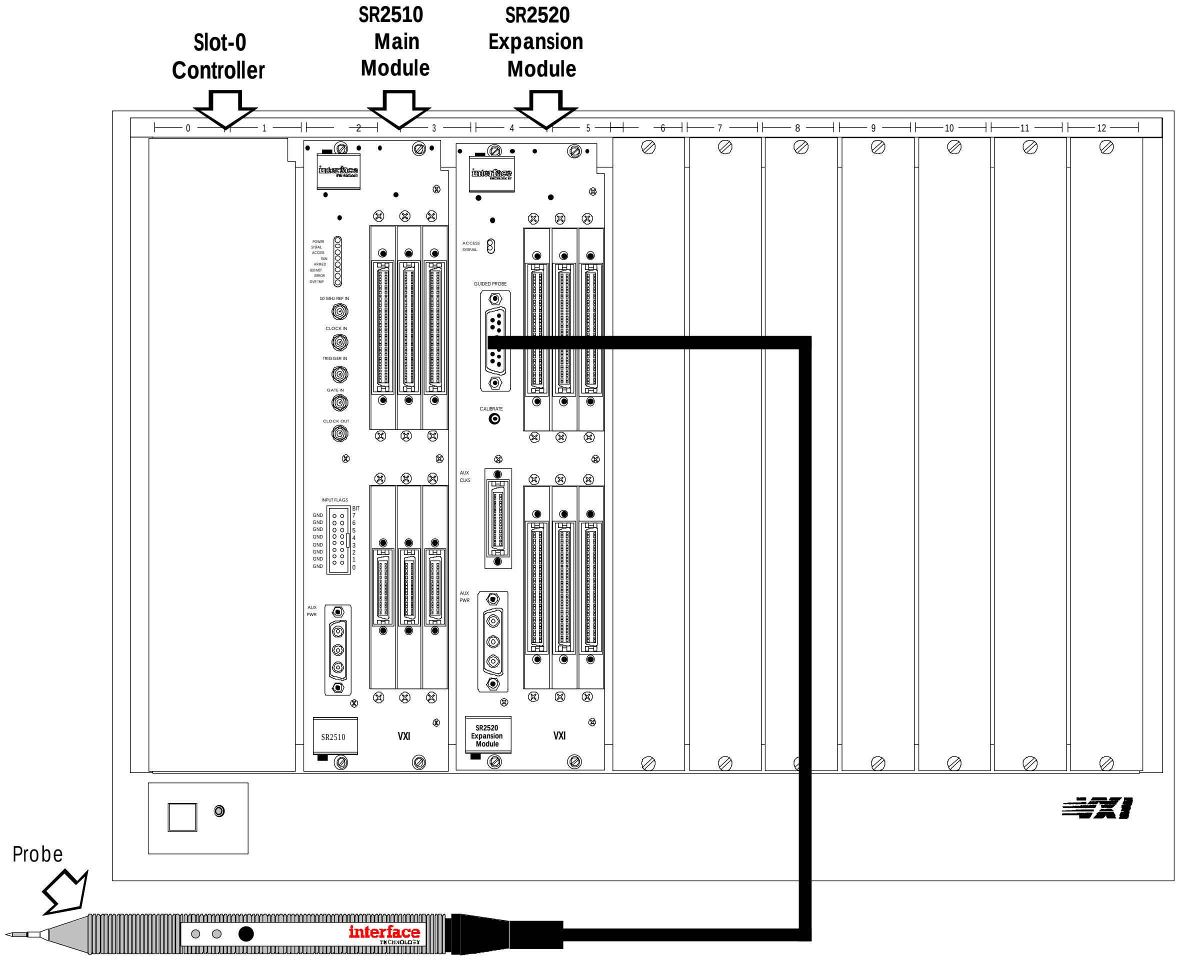


Figure 1-1. Guided Probe Add-In Option to SR2520 Module.

Controls and Indicators

There are no operator controls or adjustments on this instrument ... neither external nor internal. Operator indicators consist of two LED status indicators located on the upper left side of the front panel, and a calibration testpoint for calibrating the guided probe. Indicator function is as follows:

INDICATOR	FUNCTION
ACCESS	LED (yellow) indicator lights whenever SR2520 module is accessed over the VXI backplane.
SYSFAIL	LED (red) indicator lights during the power-up sequence until the internal self-test passes ... or remains lit if it fails.
CALIBRATE	Testpoint supplying calibration voltage to calibrate the guided probe.

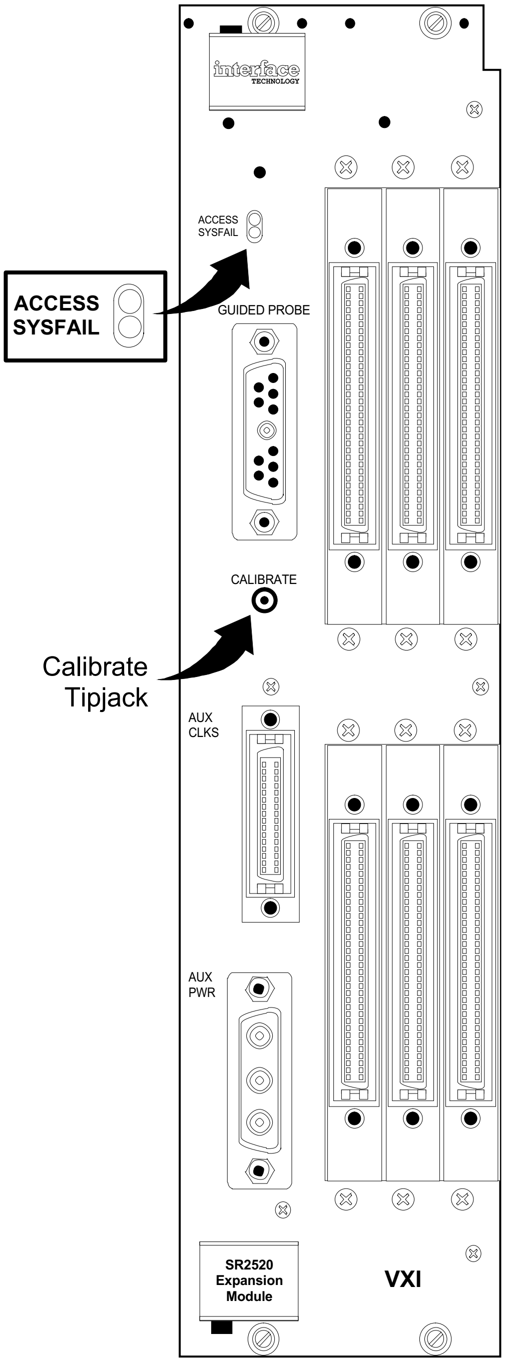


Figure 1-2.
Controls and Indicators.

Specifications

Frequency Range:	DC to 25 MHz.
Minimum Pulse Width:	10 ns.
Modes:	Edge sample / compare. Window compare.
Vector Depth	
Standard:	64K.
Optional:	256K.
Input Impedance:	100k ohms.
Resolution:	12-bits, standard analog measure, ±10.0 volt range.
Memory:	Expect, Mask, Record.
Indicators	
Detect Logic High:	LED (green).
Detect Logic Low:	LED (red).
Overvoltage Protection:	40.0 volts.

Interface Compatibility, SR2520 Add-In Option
(Refer to SR2520 specifications.)

Environmental Limitations

Temperature, Storage	-40C to +75C
Temperature, Operating	0C to +45C
Relative Humidity	5% to 95%, Noncondensing

CHAPTER 2

Theory of Operation

General

The SR2500GP probe measures 7" x 3/4", weighs under 4 oz. and has a tip that is user replaceable. A pushbutton switch on the probe body generates an interrupt to the SR2510 Main Module. The user may define the action to take, based on the probe interrupt switch.

Refer to Figs 3-1 and 3-2. The SR2500GP Guided Probe logic is resident in the SR2520 Expansion Module. It provides the capability to probe nodes on the UUT, measuring state response, and determine pass / fail conditions. The Guided Probe does not detract from the standard feature set of the SR2520, i.e., the Guided Probe does not reduce the available I/O pin count. In fact, the Guided Probe actually adds additional clocks to the basic SR2500 Subsystem.

The Guide Probe can test and detect high, low and indeterminate states, calculate a CRC checksum based on the input steam, detect pulses and make static measurements of analog voltages. Upon determination of the pass / fail state, the Guided Probe will store the information read from the node, the results of the real-time compare between the measured response and the expected response, and information indicating if the node was actively driven or tristated. The probe will also store the state of a 24 bit vector counter (time tag) and a 4 bit user defined tag.

Probe Circuitry

The probe body contains an active input circuit to reduce circuit loading on the test node and to condition the signal prior to passing it to the main Guided Probe logic in the SR2520, see Fig 3-2. Located on the probe body are an activation switch used to trigger or continue test execution and a contact sensing circuit, which indicates through a pair of LEDs that the probe tip is in contact with an active conducting node. The probe receives power for the active components via an interconnect cable, which also passes the conditioned signal back to the Guided Probe main logic board.

Main Logic

The Guided Probe main logic resides on the SR2520 expansion logic board, see Fig 3-1. This board receives programming information from the VXI bus via a register-based interface and timing control signals from the SR2510 Main Module via the master / slave interconnect. The Guided Probe logic consists of input compare logic, memory for storing node data, compare results, midstate data, expected response, mask data, vector and user time tags, and programmable user clocks. It also contains the 16 bit CRC register and provides data format and timing control of the programmable user clocks.

Extra User Clocks

When the Guided Probe option is installed, an additional 20 user clocks are available on the front panel of the SR2520. A fixed, four-phase clock is brought out, each phase to a separate pin. The frequency of these clocks are the same as the test rate programmed into the SR2500 subsystem. An additional 16 programmable clocks are available as well. These signals are properly not clocks at all, but 16 channels of user programmable stimulus. They retain the same characteristics as the other stimulus pins found within the SR2500 subsystem, except that the 16 channels are grouped as two 8 bit fields and output levels are fixed TTL.

The programmable clocks allow the user to define vector states, data formatting and format timing parameters for each channel. The channels may be used to provide clocks, inverted clocks, synchronization pulses, data strobes, etc. Algorithmic pattern functions are supported as well as RAM backed pattern generation, although since the channels are grouped as two 8 bit fields, the algorithmic function is limited to 8 pin groups. Linking the two 8 bit fields to achieve a 16 bit algorithmic field is not possible on these channels.

Tags

Each time a sample is saved to record memory (probe or I/O inputs), a 24 bit vector count time tag and a 4 bit user tag are also stored. The programmer can reset the 24 bit counter to zero at any point within a test sequence. The counter will then increment once for each test cycle. When the count reaches the maximum obtainable within a 24 bit counter, the counter recycles to zero and continues.

The user tags are also controlled by the programmer, and provide an additional level of correlation. For example, the test programmer could set the four bits to 0001 and reset the vector count to zero at the beginning of a UUT initialization routine. Then, during a test of the UUT's I/O ports, the programmer could again reset the vector counter and set the user bits to 0010. All vectors recorded during UUT initialization will show the user bits to contain 0001. And all vectors recorded during the I/O port test will show the user bits to contain 0001. The vector count time tag will indicate the vector cycle each sample was taken, relative to the beginning of each test segment.

The conditioned node signal passed from the Guided Probe to the main probe logic board may also be used to generate a CRC signature based on the incoming data stream. The checksum is generated in hardware, real-time, at whatever test rate the SR2500 subsystem was programmed for. CRC calculations are performed when enabled by the record state machine, and use the CCITT standard communication polynomial to perform the calculation. The CRC signature, or checksum, is the 16 bit remainder produced by dividing the Guided Probe input stream by the following polynomial, using Galois field arithmetic.

$$\mathbf{Gx = x^{16} + x^{12} + x^{5} + 1}$$

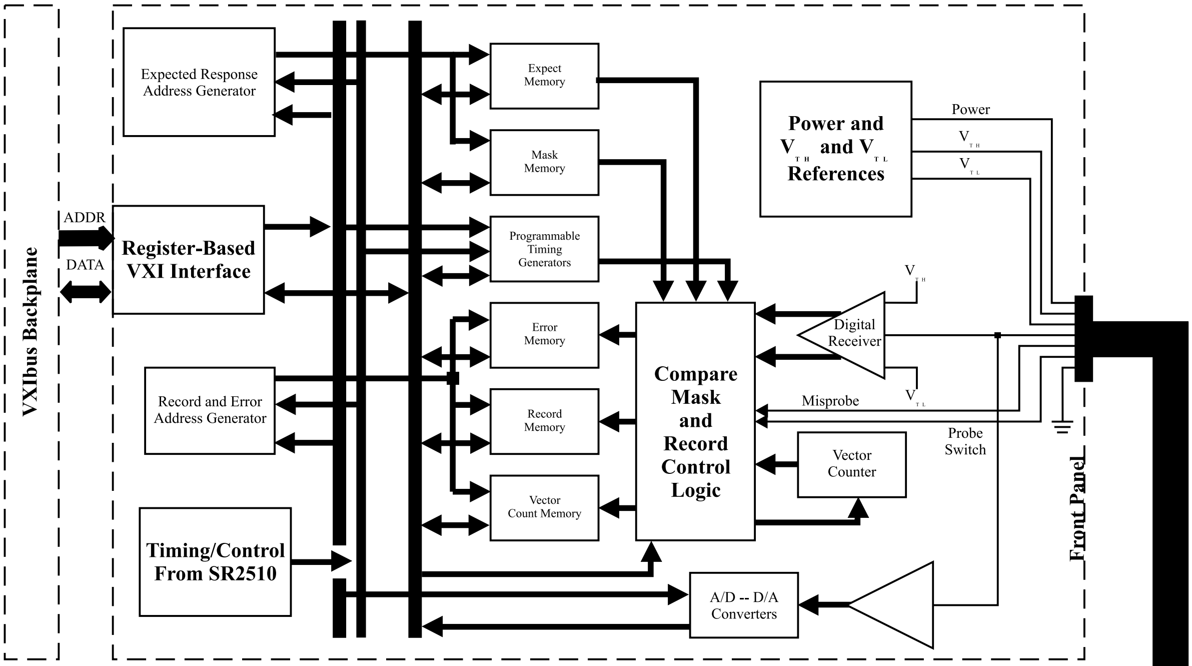

VXIbus Backplane
ADDR
DATA
Expected Response Address Generator
Register-Based VXI Interface
Record and Error Address Generator
Timing/Control From SR2510
Expect Memory
Mask Memory
Programmable Timing Generators
Error Memory
Record Memory
Vector Count Memory
Compare Mask and Record Control Logic
Power and V TH and V TL References
Power
V TH
V TL
Digital Receiver
Misprobe
Vector Counter
Probe Switch
A/D -- D/A Converters
Front Panel

Figure 2-1. Functional Block Diagram, Guided Probe Logic.

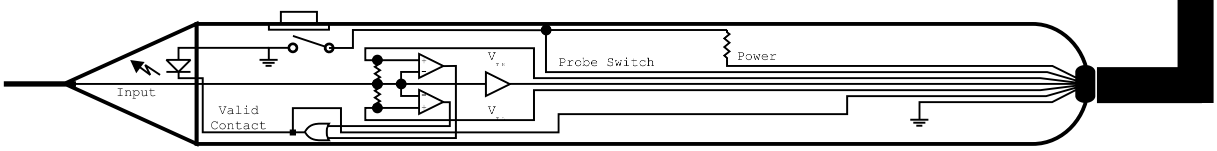


Figure 2-2. Functional Diagram, Probe.

(THIS PAGE LEFT BLANK INTENTIONALLY)

CHAPTER 3

Programming

General

Although the Guided Probe is a special purpose card, it is referenced as if it were any other I/O card. It has no input pins, except for the probe tip, and has only 16 limited purpose outputs called "clocks." A special command is provided to set up the Guide Probe hardware for general use. This command ("FIELd:PROBe:SETup") creates and initializes the necessary fields for using the probe. Other commands have been provided for other probe functions. Data recorded from the probe is accessed in the same way as any recorded data is accessed from the SR2500.

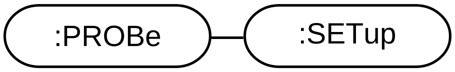
Guided Probe Commands

o Create and Initialize GP Fields
o Detecting Probe Switch Press
o Measuring a Voltage With The Probe
o Calibrating The Guided Probe
o Setting Up The Guided Probe
o Clearing The Guide Probe Fields
o Setting The High Threshold
o Setting The Low Threshold
o Setting TTL Levels For The GP Thresholds
o Setting ECL Levels For The GP Thresholds

SCPI COMMAND KEY

command	Command words take three forms, ROOT, BRANCH, and LEAF. The ROOT is the beginning of a command, i.e. the first word in a command string. Branches are the connecting paths between the ROOT and the LEAF. Branches may or may not have parameters associated with them, or may have a suffix, usually a channel indicator. The LEAF terminates the command string and may or may not have parameters associated with it.
command	Indicates commands which do not have parameters.
command	Indicates commands with parameters.
command(?)	Commands which are followed by a question mark in parenthesis indicate a command format supporting both a command and a command query.
command?	Command strings followed by a question mark without parenthesis indicates a command query only.
UPPERCASE	Command characters displayed in uppercase are required characters.
lowercase	Command characters displayed in lowercase are optional characters.
<required>	Required parameter or suffix.
[option]	Optional command or parameter.
{repeat}	Repeat as many times as required.
(min-max)	The parameter value entered must be within the range of min to max, inclusive.
aaa | bbb	Acceptable choices are aaa OR bbb.
response	Response from SR2500.

FIELDS USED BY THE GUIDED PROBE (NON-SCPI)



The following fields are used by the probe and can be automatically created and initialized by the **FIELd:PROBe:SETup** command.

P_DATA_R Type RECord, pins 30-29. This field will record the raw data from the Guided Probe. A '1' recorded at pin 29 indicates that the probe detected a low condition; a '1' recorded at pin 30 indicates the probe detected a high condition. If both pins record a '0' the probe detected an indeterminate condition.

P_ERR_R Type RECord, pins 32-31. This field will record the error data from the Guided Probe. It will compare the raw data from the probe against an indeterminate condition.

P_UTAG_R Type RECord, pins 28-25. This field is provided to record a user defined vector tag.

P_VTAG_R Type RECord, pins 24-1. This field will record a 24 bit vector tag, that is automatically generated by the SR5000 Guided Probe hardware. It should be used for no other function. This tag will start at #h000000 and increment by one to #hFFFFFF at which point it will roll over and begin again at #h000000.

P_DATA_E Type EXPect, pins 30-29. This field is used to compare against the probe's raw data to ensure that it is recorded. Defaults are all zeros.

P_DATA_M Type DONtcare, pins 30-29. This field is used to enable the compare of the probe's raw data for recording. Defaults to all ones.

P_ERR_E Type EXPect, pins 32-31. This field is used to compare the expected data to the actual data from the probe. It can be filled by using any of the normal methods to write data to a field ... and can also be filled by using the P_DATA_R field to record data from a *known good UUT* and then copying that data to the P_ERR_E field. Defaults are all zeros.

P_ERR_M Type DONtcare, pins 32-31. This field is used to enable the compare of the probe data for the P_ERR_R and P_ERR_E fields. Defaults to ones.

P_UTAG_E Type EXPect, pins 28-25. This field is filled with whatever tag is expected to be recorded in P_UTAG_R.

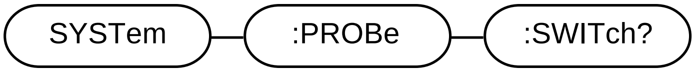
P_UTAG_M Type DONtcare, pins 28-25. This field is used to enable the recording of P_UTAG_R. Defaults to all ones.

P_VTAG_A Type ALGExpect, pins 24-1. This algorithmic field is usd to generate the vector tag in the P_VTAG_R field. Do not write to this field.

P_VTAG_E Type EXPect, pins 8-1. This field is pre-filled with '0s' to allow the recording of the vector tag in the P_VTAG_R field. Do not write to this field.

P_VTAG_M Type DONtcare, pins 8-1. This field is pre-filled with '1s' to allow the recording of the vector tag in the P-VTAG_R field. Do not write to this field.

Detecting Probe Switch Position (down /not down) (NON-SCPI)


SYSTem — :PROBe — :SWITch?

The **SYSTem:PROBe:SWITch?** command will return the current status of the switch on the Guided Probe switch. It will return a '1' if the switch is currently being held down, else it will return a '0'.

Examples SYST:PROBE:SWITCH?
0

SYST:PROB:SWIT?
1

Response *0 | 1*

MEASURING A VOLTAGE WITH THE PROBE (NON-SCPI)

SYSTem — :PROBe — :VOLT?

The **SYSTem:PROBe:VOLT?** command returns the voltage measured at the probe tip. The value returned is always in volts.

Note

If the probe is not in contact with anything, it will return the value of the probe's bias voltage, which is automatically set to midway between the high and low probe threshold voltages.

Examples SYSTEM:PROBE:VOLT?
6.02e0 volts

SYST:PROB:VOLT?
1.47e-3 volts

Response *(-10 to +10) volts.*

Calibrating The Guided Probe (NON-SCPI)

SYSTem — :PROBe — :CALibrate(?)

The **SYSTem:PROBe:CALibrate** command will initiate the calibration of the Guided Probe DACs and ADCs. The probe must be inserted into the Calibration Point, in the front panel of the SR2520 module in order for calibration to be successful.

The SYSTem:PROBe:CALibrate? command will return a '1' if calibration of the Guided Probe has been performed since power on, else it will return a '0' if calibration has not been performed since power on.

Note

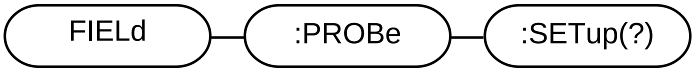
The calibration process may take up to 3 minutes; the probe must be inserted in the SR2520's calibration point during the entire process. Care must be taken not to short the probe tip to the front panel during the calibration process. To verify that calibration has been successful, take a voltage measuement of the calibrating point using the SYST:PROB:VOLT? command. The voltage measured should be 5 volts.

Examples

```
SYSTEM:PROBE:CALIBRATE
SYST:PROB:CAL

SYSTEM:PROBE:CAL?
SYST:PROB:CAL?
```

Response *0 | 1*

SETTING UP THE GUIDED PROBE FIELDS (NON-SCPI)



FIELd :PROBe :SETup(?)

The **FIELd:PROBe:SETup** command will create and initialize the fields used with the Guided Probe hardware (see page 2-3 for details).

The FIELd:PROBe:SETup? will return a '1' if the probe fields have been set up for this test and a '0' if the probe fields are not set up for this test, or if the setup has been cleared.

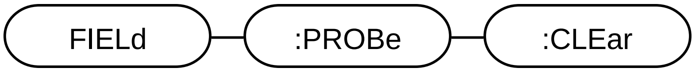
Examples

FIELD:PROBE:SETUP
FIELD:PROBE:SET

FIELD:PROBE:SETUP?
FIELD:PROBE:SET?

Response 0 | 1

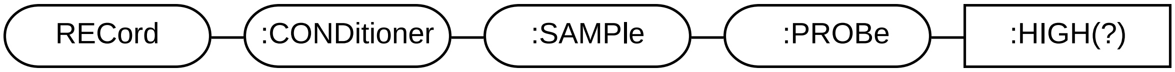
Clearing The Guided Probe Fields (NON-SCPI)



The **FIELd:PROBe:CLEar** command will delete the fields created by the FIELd:PROBe:SETUP command and reset the Probe Setup flag to '0'.

Examples FIELD:PROBE:CLEAR
FIEL:PROB:CLE

SETTING THE HIGH THRESHOLD (NON-SCPI)



The RECord:CONDitioner:SAMPle:PROBe:HIGH command will set the high threshold voltage for the Guided Probe. Any voltage higher than the high threshold will be considered a '1'.

The RECord:CONDitioner:SAMPle:PROBe:HIGH? command will return the current setting of the high threshold voltage for the Guided Probe.

:HIGH < MAX | MIN | DEF | (-9.0 volts to +10.0 volts) >

Parameter Definition

MAX = The maximum voltage for the high threshold (+10.0 volts).

MIN = The minimum voltage for the high threshold (-9 volts).

DEF = The default voltage for the high threshold (+2.0 volts).

Examples

REC:COND:SAMP:PROBE:HIGH MAX
REC:COND:SAMP:PROB:HIGH 3.20 V

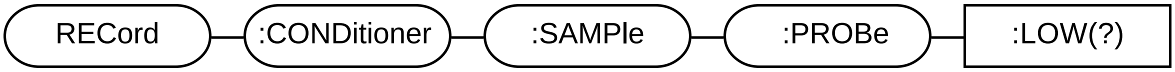
REC:COND:SAMP:PROB:HIGH?

Response

5.0e-1 volts

-2.5 volts

SETTING THE LOW THRESHOLD (NON-SCPI)



RECord—:CONDitioner—:SAMPle—:PROBe—:LOW(?)

The RECord:CONDitioner:SAMPle:PROBe:LOW command will set the low threshold voltage for the Guided Probe. Any voltage lower than the low threshold will be considered a '0'.

The RECord:CONDitioner:SAMPle:PROBe:LOW? command will return the current setting of the low threshold voltage for the Guided Probe.

:LOW < MAX | MIN | DEF | (-10.0 volts to +9.0 volts) >

Parameter Definition

MAX = The maximum voltage for the low threshold (+9.0 volts).

MIN = The minimum voltage for the low threshold (-10 volts).

DEF = The default voltage for the high threshold (+0.8 volts).

Examples

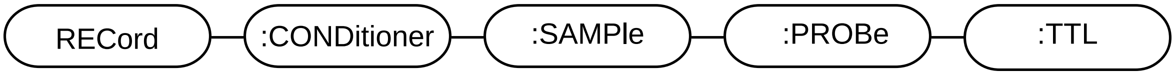
```
REC:COND:SAMP:PROBE:LOW MAX
REC:COND:SAMP:PROB:LOW 3.20 V

REC:COND:SAMP:PROB:LOW?
```

Response

9.000000e+00 volts

3.200000e+00 volts

SETTING TTL LEVELS FOR THE GP THRESHOLDS (NON-SCPI)



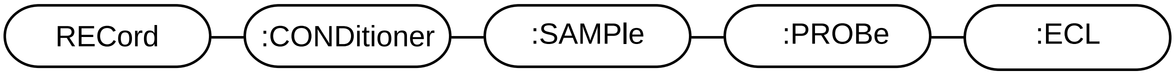
The RECord:CONDitioner:SAMPle:PROBe:TTL command will set the high and low threshold voltages for the Guided Probe to levels suitable for TTL logic. The high threshold will be set to +2.0 volts and the low threshold will be set to +0.8 volts.

Note

These values are identical to the default values for the probe high and low threshold values.

Examples REC:COND:SAMP:PROBE:TTL

SETTING ECL LEVELS FOR THE GP THRESHOLDS (NON-SCPI)



The RECord:CONDitioner:SAMPle:PROBe:ECL command will set the high and low threshold voltages for the Guided Probe to levels suitable for ECL logic. The high threshold will be set to -1.15 volts and the low threshold will be set to -1.49 volts.

Examples REC:COND:SAMP:PROBE:ECL

Flow Chart for Setting Up Guided Probe

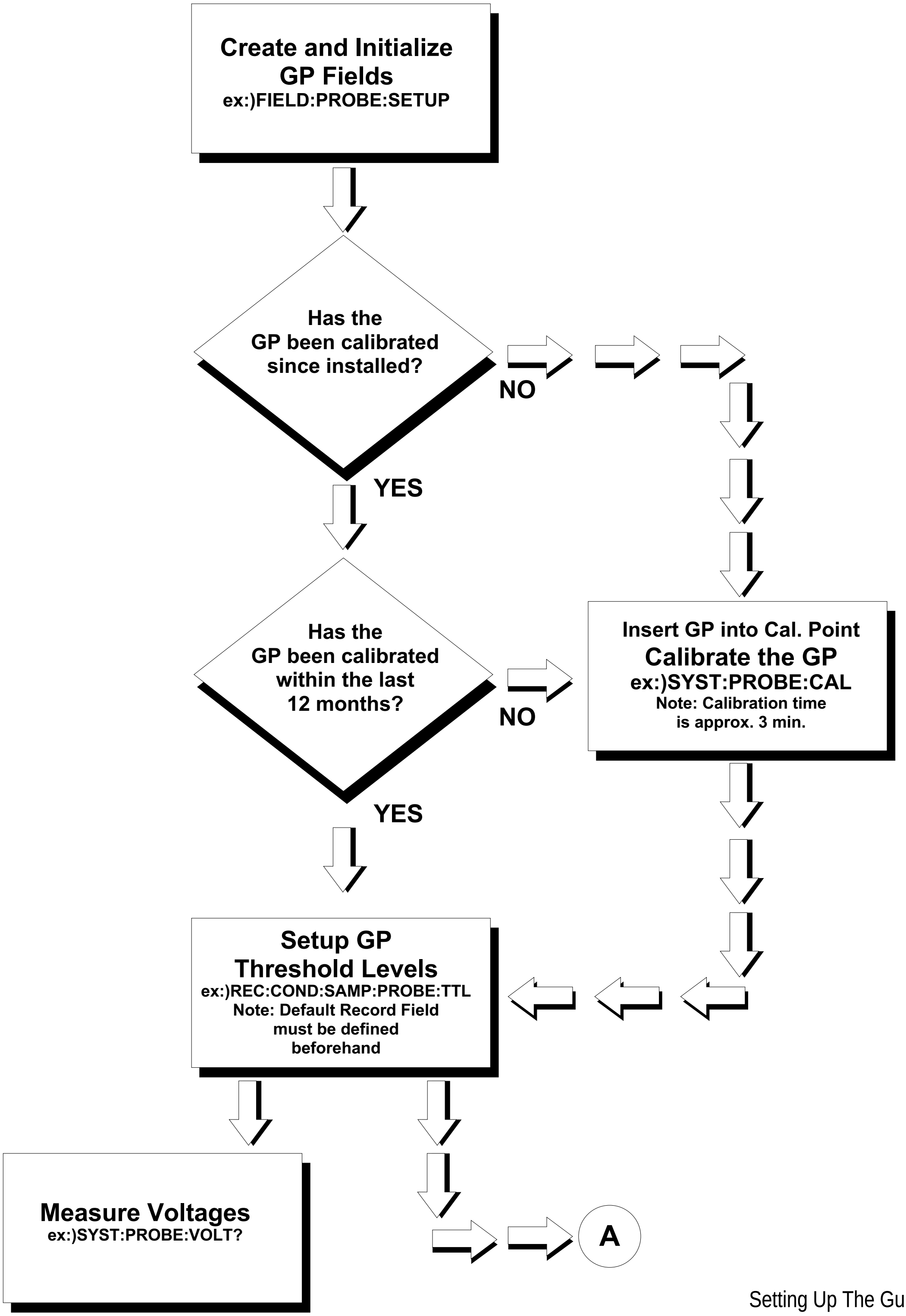


Figure 3-2.
Setting Up The Guided Probe,
(Sheet 1 of 2).

Flow Chart for Setting Up Guided Probe (cont.)

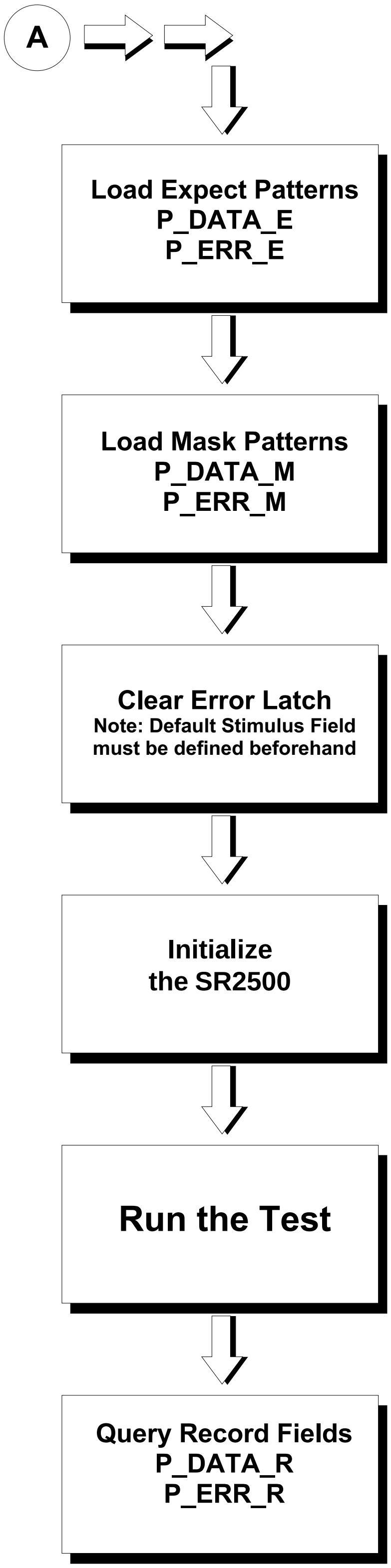


Figure 3-2.
Setting Up The Guided Probe,
(Sheet 2 of 2).

CHAPTER 4

Installation

Scope of Chapter

This chapter contains instructions for unpacking, inspecting, installing, and checking out the SR2520 Expansion Module w/Guided Probe option.

Unpacking and Inspection

Your instrument was thoroughly inspected and tested before shipment from the factory and is ready for immediate operation once all installation procedures have been completed. Carefully remove the instrument from its shipping carton and check for any obvious damage that may have occurred during shipment. If damage is found, report it to the freight carrier immediately. Interface Technology is not liable for damage that may have occurred during transit. Save the shipping carton and all packing material for possible future use.

Installation

Logical Addressing

Before installation, the logical address for the SR2520 w/Guided Probe option must be set. Set the address switches according to the requirements of the slot-0 controller. The address switches are numbered from one to eight. Switch 1 corresponds to the least significant bit (LSB) of the logical address. The address is entered in binary, where an ON switch sets the corresponding bit to 0 (Fig 4-1).

Note

The logical addresses of the SR2520 Expansion Modules must be set to a higher value than the logical address of the SR2510 Main Module. If there is more than one SR2510 in a VXI chassis, then the SR2520's with addresses between any 2 SR2510's, will be part of the lower addressed SR2510's system. The SR2520 with the lowest numbered logical address is Expansion Module #1. The next highest SR2520 logical address is Expansion Module #2. The highest SR2520 logical address is the most significant Expansion Module number. To verify all Expansion Modules have been recognized by the system, send a “*IDN?” query command.

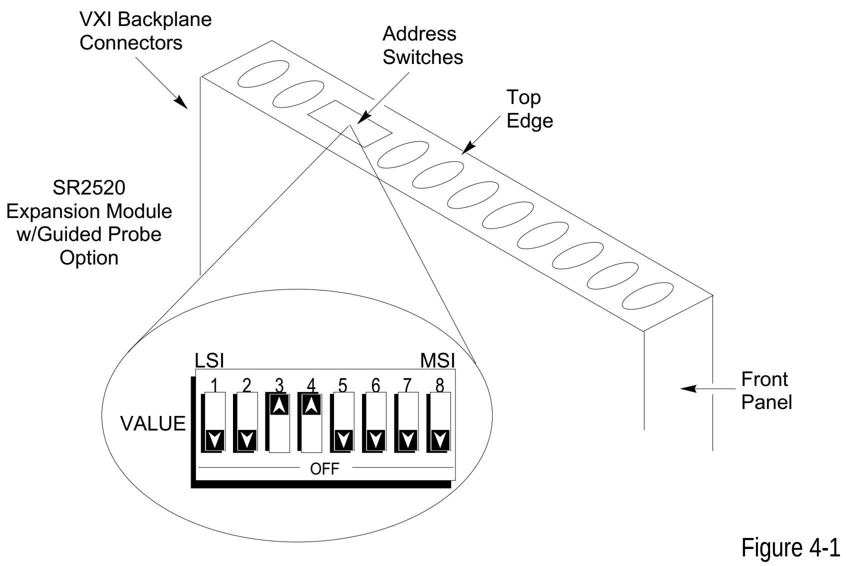


Figure 4-1
Address Switches Set
to Logical Address 12_{hex} .

Logical Addressing

Logical addressing for the SR2500GP Guided Probe module is the same as for the standard SR2520 Expansion Module w/o Guided Probe, refer to SR2500 User's Manual.

Slot Dependencies

The SR2500GP Guided Probe Module (stand alone module) must be installed in theVXI chassis to the right-of the SR2510 Main Module see Fig 4-4.

Backplane Jumpers

The use of backplane jumpers for the SR2500GP Guided Probe module are the same as for the standard SR2520 Expansion Module w/o Guided Probe, refer to the SR2520 User's Manual.

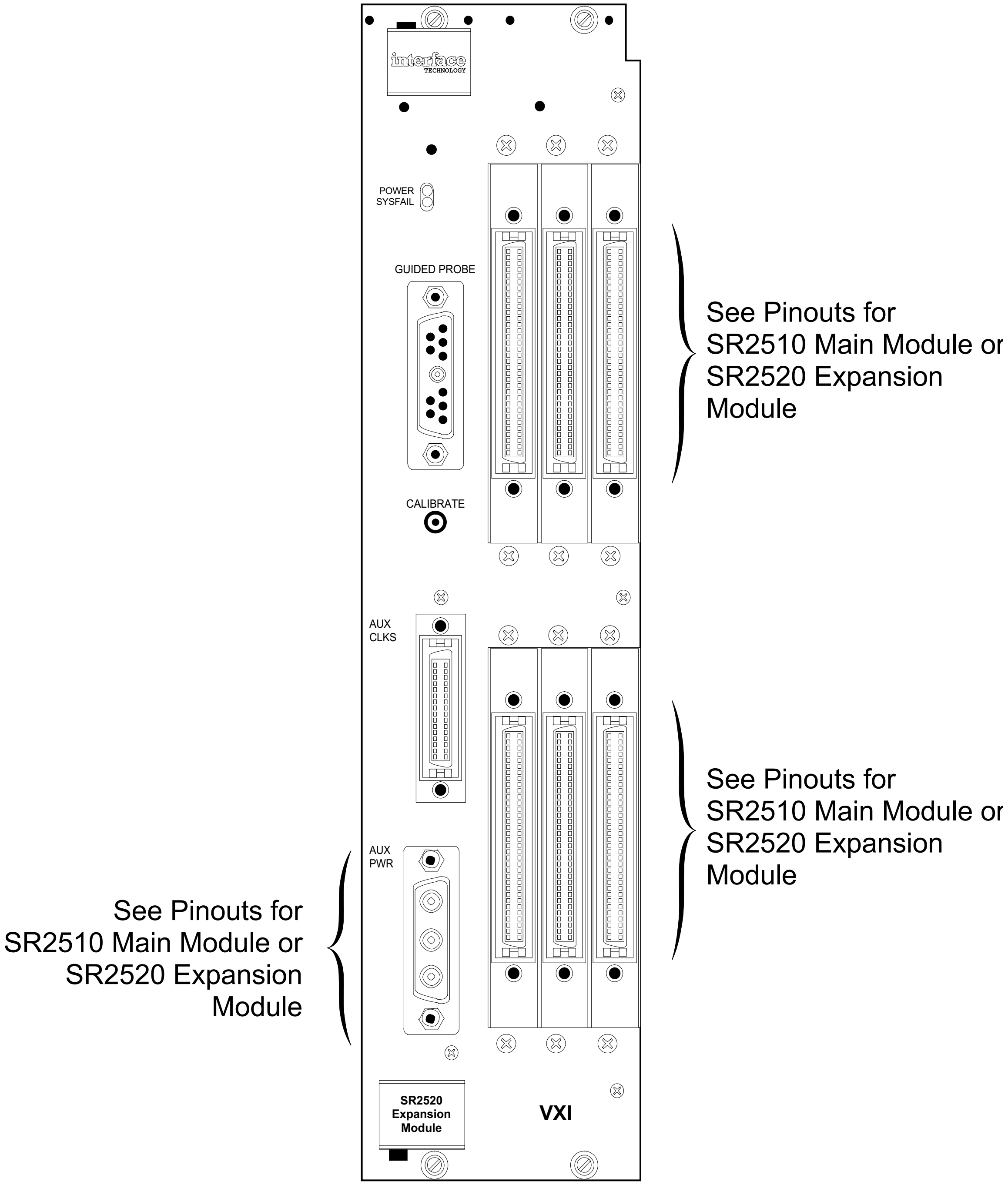
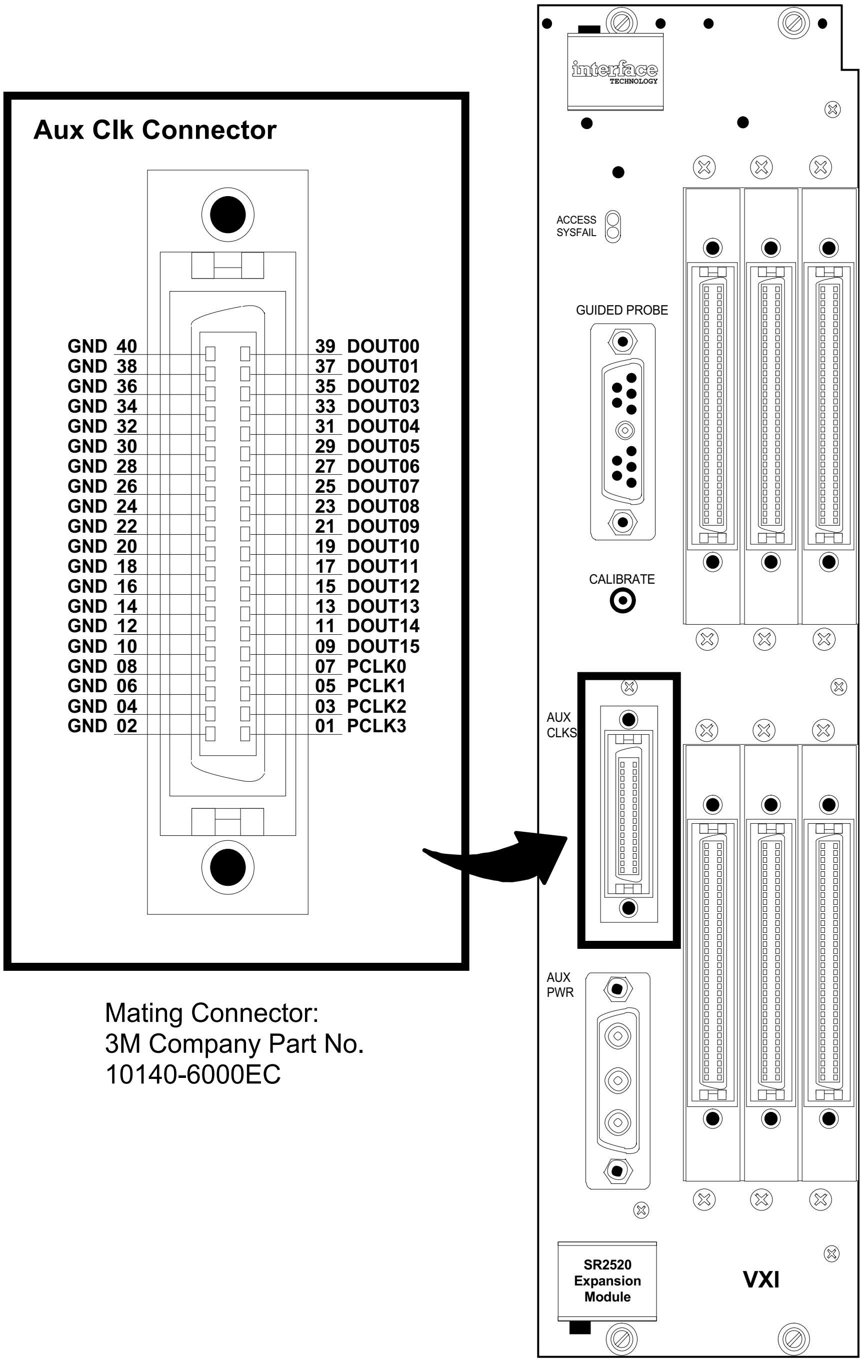


Figure 4-2. SR2520 w/Guided Probe I/O and Aux. Power Pinouts.



Mating Connector:
3M Company Part No.
10140-6000EC

Figure 4-3. SR2520 w/Guided Probe Aux. Clock Pinouts.

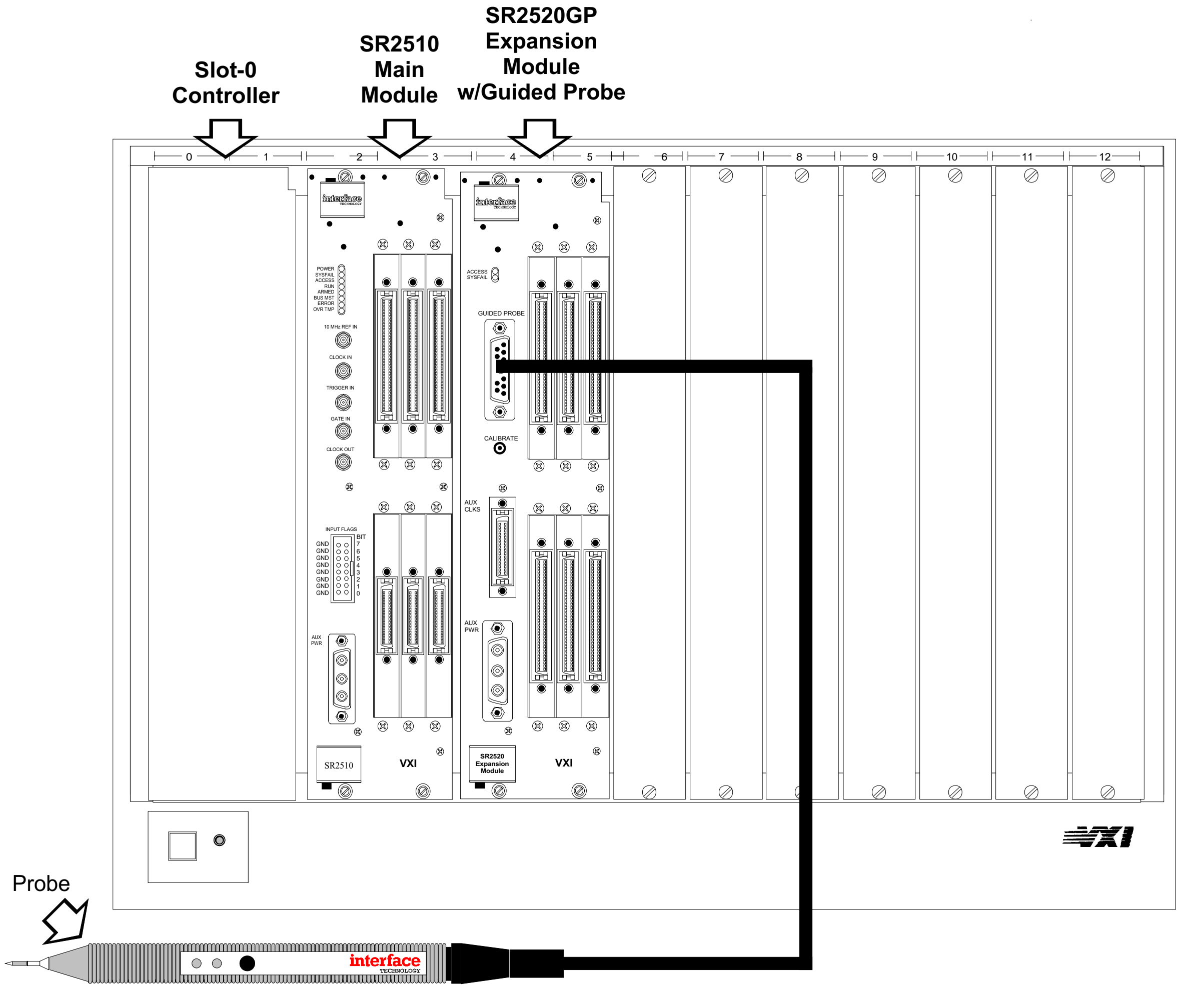


Figure 4-4. Installation of SR2520 w/Guided Probe Module.

(THIS PAGE INTENTIONALLY LEFT BLANK)

Published by
Interface Technology, Inc.
300 S. Lemon Creek Drive
Walnut, CA 91789

APPENDIX A

Error Codes

Scope of Appendix

If the SR2500 self-test fails, the following error code information is written to the VXI datalow register:

Data Low Register Self-Test Error Codes

100 (hex) RAM Test Error in Program RAM on Bank 1
101 (hex) RAM Test Error in Program RAM on Bank 2
102 (hex) RAM Test Error in Program RAM on Bank 3
103 (hex) RAM Test Error in Program RAM on Bank 4
200 (hex) RAM Test Error in Shared RAM, Bank 1, bits 7-0
201 (hex) RAM Test Error in Shared RAM, Bank 1, bits 15-8
202 (hex) RAM Test Error in Shared RAM, Bank 1, bits 23-16
203 (hex) RAM Test Error in Shared RAM, Bank 1, bits 31-24
204 (hex) RAM Test Error in Shared RAM, Bank 2, bits 7-0
205 (hex) RAM Test Error in Shared RAM, Bank 2, bits 15-8
206 (hex) RAM Test Error in Shared RAM, Bank 2, bits 23-16
207 (hex) RAM Test Error in Shared RAM, Bank 2, bits 31-24
300 (hex) IT9010M failed read/write acknowledge test
301 (hex) IT9010M failed read/write pattern test
400 (hex) Shadow RAM failed write/read test
600 (hex) ROM failed checksum test

System Error Codes

Command Errors

0 "No Error"
-101 "Invalid character; Semicolon can't start command"
-103 "Invalid separator; Semicolon or colon expected"
-101 "Invalid character; Syntax error at second colon"
-101 "Invalid character; Syntax error at semicolon following colon"
-101 "Invalid character; Double semicolons not allowed"
-103 "Invalid separator; Asterisk found instead of separator"
-111 "Header separator error; Alpha after 488.2 common cmd invalid"
-101 "Invalid character; Double colons not allowed"
-101 "Invalid character; Colon found but no commands at a lower level"
-101 "Invalid character; Unknown in this context"
-101 "Invalid character' Double semicolons not allowed"
-103 "Invalid separator; Asterisk found instead of separator"
-103 "Invalid separator; Alpha found instead of separator"
-101 "Invalid character; Asterisk found instead of separator"
-158 "String data not allowed; No match found for parameter string"
-113 "Undefined header; A 488.2 common command was expected.
-114 "Header suffix out of range; Number after 488.2 cmd not allowed"

-113 "Undefined header; No match found for command"

-131 "Invalid suffix; Suffix not appropriate"

-113 "Undefined header; Question mark expected"

-101 "Invalid character; Unexpected character found after header"

-113 "Undefined header; Number attached to header not allowed"

-111 "Header separator error; A space separator was expected"

-131 "Invalid suffix; Suffix not appropriate for command"

-144 "Character data too long; Name is maximum of 8 chars"

-103 "Invalid Separator; Comma not found as expected"

-104 "Data type error; PIN LIST syntax <CnPn> or <CnPn-n> not found"

-104 "Data type error; Syntax error on number list parameter"

-104 "Data type error; Invalid Sample Mode, only EDGE or WINDow is allowed"

-104 "Data type error; Invalid OFormat Mode, only NRZ, RZ, RONE, RTC, or RI is allowed"

-104 "Data type error; Invalid field name, only 8 chars long allowed"

-104 "Data type error; Invalid test name, only 8 chars long allowed"

-104 "Data type error; Invalid test name"

-102 "Syntax error; Illegal operator, Command Macro's command does not support Not Equal operator, %s"

-102 "Syntax error; Invalid Command Macro's LABel keyword, %s"

-102 "Syntax error; Command Macro statement must be 'OUTput' or 'NOP' keyword, %s"

-102 "Syntax error; Command Macro statement contains invalid Command keyword, %s"

-102 "Syntax error; Command Macro statement contains incorrect operator, '==' or '<>' are valid, %s"

-102 "Syntax error; Command Macro statement's Right-side Expression must be an 8 bit number, %s"

-102 "Syntax error; Command Macro statement's Right-side Expression must be a 16 bit integer, %s"

-102 "Syntax error; NAME/LABEL must begin with an alpha character."

-102 "Syntax error; NAME/LABEL contains illegal character, only alpha, numeric, and '_' are allowed"

-102 "Syntax error; Unknown Algorithmic Macro command, %s"

-102 "Syntax error; SProgram, EProgram, ELoop, OUTput, CLEARError, RTSubroutine, and CRTSubroutine Command Macros must not contain any parameter, %s\""

-102 "Syntax error; SCONDition Command Macro cannot have COUNt, BUS, and STRIgger as a parameter, %s"

-102 "Syntax error; JMP and JSRoutine Command Macros must have a label in the parameter, %s"

-102 "Syntax error; Illegal Label/Subroutine name, 'ALL' is not a legal Label/Subroutine name."

-102 "Syntax error; SLoop and WLoop Command Macros must have some form of parameter, %s"

-102 "Syntax error; Command Macro must begin with '(', %s"

-102 "Syntax error; Command Macro must end with ')', %s"

-102 "Syntax error; Command Macro is missing ')' after the 'LABel <NAME>, %s"

-102 "Syntax error; Command Macro missing '(' before the '<PARAMETER>, %s"

-102 "Syntax error; Command Macro missing ')' after the '<PARAMETER>', %s"

-102 "Syntax error' Command Macro missing '(' before the '<EXPRESSION>', %s"

-102 "Syntax error; Command Macro missing ')' after the '<EXPRESSION>', %s"

-102 "Syntax error; SCONDition Command Macro must have some form of a parameter, %s"

Execution Errors

-224 "Illegal parameter value; Invalid conversion"

-224 "Illegal parameter value; Invalid base value"

-224 "Illegal parameter value; Undefined parameter"
-224 "Illegal parameter value; Invalid data type"
-222 "Data out of range; Value out of current radix bounds"
-222 "Data out of range; Baud rate not supported"
-222 "Data out of range; Data bits must be 7 or 8"
-222 "Data out of range; Stop bits must be 1 or 2"
-222 "Data out of range; Parity type not supported"
-241 "Hardware missing; Address generates bus/addr exception"
-224 "Illegal parameter value; Valid fill types are REP, INC, DEC, COM, ALT, WLKO, WLK1, or RAN"
-221 "Settings conflict; Undefined field name or field type is not available under this subsystem"
-221 "Settings conflict; Undefined test name"
-221 "Settings conflict; All available fields have been defined"
-221 "Settings conflict; All available tests have been defined"
-225 "Out of memory; Not enough Free vectors available for allocation" -REPEAT
-222 "Data out of range; Valid card numbers are 1-10"
-222 "Data out of range; Valid pin numbers are 1-32 and max number of pins is 32"
-222 "Data out of range; Valid channels are 1-32"
-221 "Settings conflict, No default field is defined"
-221 "Settings conflict; No working test is defined"
-222 "Data out of range; Invalid vector number"
-221 "Settings conflict; Test name is already defined"
-221 "Settings conflict; Field name is already defined"
-222 "Data out of range; Valid trace qual numbers are 1-8 and only up to 8 trace qualifiers can be entered"
-221 "Settings conflict; Not enough items were provided base on the count value."
-222 "Data out of range; Invalid EndLoop (EL) count, 1 and 2 are valid."
-222 "Settings conflict; No SProgram statement found in Command Macro"
-222 "Settings conflict; Too many StartLoop statements before an EndLoop statement at vector %s"
-222 "Settings conflict; Too many EndLoop statements in Command Macro at vector %s"
-222 "Settings conflict; No EProgram statement found in Command Macro"
-222 "Data out of range; Statement's parameter must be 1 to 65535."
-221 "Settings conflict; FIELd TYPE must be OUT or OT with OFORmat"
-222 "Data out of range; OFORmat delay exceeds period"
-222 "Data out of range; OFORmat 'pos/neg pulse width' exceeds 5 ns"
-222 "Data out of range; Max number of OFORmat delays (4) have been used for %s"
-222 "Data out of range; Card number exceeds the current number of SR cards in the system"
-222 "Data out of range; Rate speed must be 400 Hz - 50 MHz"
-222 "Data out of range; Valid group numbers are 1 - 4"
-241 "Hardware missing; IO card does not support Variable Voltage and Variable Threshold settings"
-222 "Data out of range; Value for upper Variable Voltage TTL signals are out of range"
-222 "Data out of range; Value for lower Variable Voltage TTL signals are out of range"
-222 "Data out of range; Value for upper Variable Voltage ECL signals are out of range"
-222 "Data out of range; Value for lower Variable Voltage ECL signals are out of range"
-222 "Data out of range; Value for Variable Voltage Threshold TTL signals are out of range"
-222 "Data out of range; Value for Variable Voltage Threshold ECL signals are out of range"
-241 "Hardware missing; Shared memory option is not installed."
-285 "Program syntax error; Learn encountered invalid format, Learn aborted"

-241 "Hardware missing; Different Control Card was used on the LearnQ command"
-241 "Hardware missing; Different number of I/O cards were used on the LearnQ command"
-285 "Program syntax error; Learn encountered invalid Record header, Learn aborted"
-285 "Program syntax error; Learn command requires additional blocks of data to complete Learn session"
-221 "Settings conflict; Don'tCare syntax is not allowed on single memory fields"
-221 "Settings conflict; FIELd TYPE must be OUT, TRI or OT with ARMData"
-222 "Data out of range; TIMeout STARt or STOP value not valid"
-221 "Settings conflict; FIELd TYPE must not be OT nor ED with BLKVALue"
-254 "Media full; Vectors requested are greater than the Shared Memory size"
-221 "Settings conflict; Invalid pin definition; Algorithmic fields must follow the rule in the manual."
-222 "Settings conflict; Algorithmic field is trying to use pins that are assigned to other algorithmic fields."
-221 "Settings conflict; An algorithmic field must be used."
-221 "Settings conflict; Only 10 labels may be used per vector, %s"
-222 "Data out of range; Command Macro Jump Page statement must be between 1 to 32 (dec)"
-221 "Settings conflict; The label '%s' has already been declared"
-221 "Settings conflict; The label '%s' was not found"
-221 "Settings conflict; The label '%s' for the vector does not exist"
-221 "Settings conflict; A referenced label '%s' cannot be deleted"
-221 "Settings conflict; Command requires a Subroutine Label, %s"
-221 "Settings conflict; Subroutine Labels must begin on 32 vector boundary plus 1, %s"
-222 "Data out of range; SJMPPage Command Macro must have a jump page location of 1-32 (dec), %s"
-222 "Data out of range; Count must be 1 to size of test"
-222 "Data out of range' Test size must end on an even boundary"
-221 "Settings conflict; Command requires a normal Label, %s"
-222 "Data out of range; Test size must be greater than zero"
-222 "Data out of range; Sequence number must be between 1-16"
-222 "Data out of range; Sequence number must be between 1-8"
-221 "Settings conflict; Field name used in Trace Macros no longer exists"
-222 "Data out of range; Trace Qualifier number must be between 1-8"
-221 "Settings conflict; Trace Field type must be of Expect and/or Don'tCare (ED, E, or D) type"
-222 "Data out of range; Trace Qualifier Combination number must be between 1-8"
-221 "Settings conflict; Trace Macro STOP statement can't begin sequence"
-221 "Settings conflict; Trace Macro START statement can't follow START statement"
-221 "Settings conflict; Trace Macro STOP statement can't follow STOP statement"
-221 "Settings conflict; Trace Macro CONT statement can't follow STOP statement"
-222 "Data out of range; Valid occurrence numbers are 1-65535"
-221 "Settings conflict; Trace Macro Field type must be of Expect and Don'tCare (ED) type"
-221 "Settings conflict; STIMulus subsystem must have a field memory type of Output, Tristate, OT, or ALGOutput"
-221 "Settings conflict; RECord subsystem must have a field memory type of Expect, Don'tCare, ED, or ALGExpect"
-241 "Hardware missing; IO Card selected does not exist in the system."
-221 "Settings conflict; Cannot execute a DIAGnostic unless all tests are deleted"
-222 "Data out of range; Valid pin numbers are 1-32 and max number of pins is 320"
-222 "Data out of range; Value out of BASIC mode bounds"
-240 "Hardware error; BUS Master error, unable to gain control of VME BUS within the BUS Master

timeout value"
-222 "Data out of range; Value out of bus master timeout bounds"
-222 "Data out of range; Value out of clock level -5 V to 5 V bounds"
-222 "Data out of range; Value out of gate level -5 V to 5 V bounds"
-222 "Data out of range; Value out of trigger level -5 V to 5 V bounds"
-221 "Settings conflict; Field type must be EXPected, DONtcare, ED, RECord, or ALGExpected"
-222 "Data out of range; Max number of Sample delays (2) have been used for %s"
-221 "Settings conflict; Field type must be OUTput, TRIstate, OT, or ALGOutput"
-222 "Data out of range; Valid count numbers are 1-65535"
-222 "Data out of range; Max RATE for any algorithmic field of 16-bits, 24-bits, or 32-bits is 25 MHz"
-280 "Program error; Timeout during Learn process"
-285 "Program syntax error; Invalid command in shared memory header during Learn process"
-222 "Data out of range; Vector number is out of range"
-241 "Hardware missing; Size of shared memory block exceeds the actual size of the shared memory"
-221 "Settings conflict; Size of the current test(s) plus size of basic test exceeds memory left. Downsize the basic test to meet requirements."
-213 "Init ignored; Only one test can be running at any given time"
-222 "Data out of range; Valid count numbers are 1-1000000"
-285 "Program syntax error; The necessary Learn records were received in an invalid order."
-225 "Out of memory; No more free DRAM memory to execute the command"
-284 "Program currently running; Can't execute command when a test is in 'RUNNING' or 'ARMED' state"

Device Dependent Errors

-350 "Queue overflow; Tail of output string is lost"
-310 "System error; Software bug - error number is out of range"

Query Errors

-410 "Query INTERRUPTED; Previous query output within string was overwritten"
-410 "Query INTERRUPTED; Previous query output lost"
-420 "Query UNTERMINATED; Output buffer was empty"

(THIS PAGE INTENTIONALLY LEFT BLANK)

APPENDIX B

Calibration Verification

1.0 Introduction

This procedure describes a method of verifying the calibration of the Interface Technology SR2500VV module. Standard laboratory test equipment with NIST traceability is used to verify the performance of the SR2500VV module. This procedure does not address the settings and/or interconnection of the standard test equipment.

The performance parameters verified are system clock accuracy and stability, output levels and timing, and input threshold and timing. Procedures are not provided for programmable thresholds of the external clock, or external trigger or gate inputs. This procedure assumes that the SR2500VV module is fully operational and connected to the Rail Generator and that the system has passed the built-in self test. The examples given are for the CARD 1 Variable voltage card. The pin definitions and I/O connectors will have to be changed to verify other Variable Voltage cards in the system.

2.0 Test Equipment Required

Test equipment meeting the following general requirements will be necessary to perform the calibration outlined herein. In general, the actual equipment used should be at least 4X more accurate than the allowable tolerances being measured.

o **2.1 VXI Chassis ...** with a slot 0 controller or embedded computer capable of communicating with the SR2500VV module.

o **2.2 Counter/Timer, Dual Input ...** for measuring frequency accuracy and stability, pulse width, and timing interval accuracy. The counter should have a 500ps (with or without averaging) time interval measurement accuracy. The frequency accuracy should be 312 Hz out of 25 MHz or better. the minimum resolution required is 9 digits per second gate. The **HP5335A** or **HP5370B** meet these requirements.

o **2.3 Digital Voltmeter ...** with DC and True RMS measurement capability. The DC accuracy should be 0.005% on the 10v range with 2.5 mv minimum resolution. The normal mode (ripple) rejection at 1KHz should be 60dB minimum. The **HP3458A** meets these requirements.

o **2.4 DC Power Supply ...** adjustable 0 to 6 v DC voltage source. The output should be setable to within +/- 10mv and should have less then 10mv noise and ripple. The outputs should isolated from chassis ground. The **HP6611C** meets these requirements.

o **2.5 Break Out Cable ...** Connection to the SR2500VV I/O pins is done using a "break-out" cable. This is a mating connector with approximately 3 in. of insulation displacement cable attached. The end of the cable is split apart and each wire stripped back approximately 1/8 in. in order to make connection with the different test equipment probes. Care should be taken to prevent the wires from shorting together during test.

3.0 Verification Procedure

3.1 Overview

The procedures outlined herein are for a standard SR2510VV module with 32 channels. If the SR2510VV has additional channels (up to 96 per module) the tests would be repeated for each block of 32 channels.

The basic sequence of operations will be to verify output driver level accuracy using DC voltage readings. The output waveform timing is then verified using a counter/timer. The input threshold levels are then verified using a DC voltage source. Finally, the SR2500VV outputs are used to verify the input timing.

The verification philosophy is to test a particular voltage parameter at 10%, 50%, and 90% of full scale. This will verify that the calibration for any gain and offset errors is effective. If ranging or gain switching is used, additional readings will be taken after changing the ranging values.

3.2 Output Driver Level Verification

The SR2500VV outputs are connected to the DC meter using the break-out test cable. Refer to figure 1 for the pin location of each output channel. The output drivers are set in volts, and calibrated for no-load conditions. The test sequence will allow measurement of the low data reference level at 10%, 50%, and 90% voltage levels. The data pattern is changed to high and the same settings are verified at the high level. After the output levels are tested statically, they are tested dynamically using an AC RMS meter.

The static test DC meter reading must be within +/- 100mvDC for the low and high reference output levels.

The dynamic output pattern will appear as a 1 KHz square wave with a 6 Volt P-P output, which the RMS must measure as 3.0vAC RMS within +/- 100mV.

Sample Code to Execute the Static Output Test

Note

For all code samples comment lines begin with #!. Lines where a pause is required to verify a meter reading, or wait for operator, begin with #@.

```
*RST

#!        ;define a general purpose test name and output field
TEST:DEF OUT_TEST:SIZE 2
FIELD:DEF ALL_OUTS:TYPE OT:PIN C1P32-1

#!        ;turn all output drivers on using rail A
STIMULUS:CONDITIONER:OFORMAT:VOLT A
SYST:RGEN 1:CONN 1
STIMULUS:COND:FET:CONN

#! FOR n = 0 TO 32 DO

#!        ; force output pins data value to low (0)
STIMULUS:ARMDATA:MODE ON;PATTERN #H00000000

#!        ; set rail A1 low to 10%
SYST:RGEN 1:RAIL A1:HIGH 6.5 ;LOW -2.25
INITIATE

#!        ; measure here for low output= -2.25 +/- .10v
#@
abort

#!        ; set rail A1 low to 50%
SYST:RGEN 1:RAIL A1:LOW .75 V
INITIATE

#!      ; measure here for low output = .75v +/- .10v
#@
abort

#!        ; set rail A1 low to 90%
SYST:RGEN 1:RAIL A1:LOW 3.75V
INITIATE

#!        ; measure here for low output = 3.75 +/- .10v
#@
ABORT

#!        ; force output pins data value to high (1)
STIMULUS:ARMDATA:MODE ON;PATTERN #HFFFFFFFF
```

```
#!        ; set rail A1 high to 10%
SYST:RGEN 1:RAIL A1:LOW -3.0;HIGH –0.65
INITIATE

#!        ; measure here for high output= -0.65 +/- .10v
#@
ABORT

#!        ; set rail A1 high to 50%
SYST:RGEN 1:RAIL A1:HIGH 2.75V
INITIATE

#!      ; measure here for high output= 2.75V +/- .10v
#@
ABORT

#!        ; set rail A1 high to 90%
SYST:RGEN 1:RAIL A1:HIGH 6.15V
INITIATE

#!        ; measure here for high output= 6.15V +/- .10v
#@
ABORT

#!        ; now do the same with the B rail

#!        ;turn all output drivers on using rail B
STIMULUS:CONDITIONER:OFORMAT:VOLT B

#! FOR n = 0 TO 32 DO

#!        ; force output pins data value to low (0)
STIMULUS:ARMDATA:MODE ON;PATTERN #H00000000

#!        ; set rail B1 low to 10%
SYST:RGEN 1:RAIL B1:HIGH 6.5 ;LOW -2.25
INITIATE

#!        ; measure here for low output= -2.25 +/- .10v
#@
ABORT

#!        ; set rail B1 low to 50%
SYST:RGEN 1:RAIL B1:LOW .75 V
INITIATE

#!      ; measure here for low output = .75v +/- .10v
#@
ABORT
```

```
#!        ; set rail B1 low to 90%
SYST:RGEN 1:RAIL B1:LOW 3.75V
INITIATE

#!        ; measure here for low output = 3.75 +/- .10v
#@
ABORT

#!        ; force output pins data value to high (1)
STIMULUS:ARMDATA:MODE ON;PATTERN #HFFFFFFFF

#!        ; set rail B1 high to 10%
SYST:RGEN 1:RAIL B1:LOW -3.0;HIGH –0.65V
INITIATE

#!        ; measure here for high output= -0.65V +/- .10v
#@
ABORT

#!        ; set rail B1 high to 50%
SYST:RGEN 1:RAIL B1:HIGH 2.75V
INITIATE

#!      ; measure here for high output= 2.75V  +/- .10v
#@
ABORT

#!        ; set rail B1 high to 90%
SYST:RGEN 1:RAIL B1:HIGH 6.15V
INITIATE

#!        ; measure here for high output= 6.15V +/- .10v
#@
ABORT

SYSTEM:PROGRAMLOOP CONTINOUS
INITIATE;*TRG

#!      ;pause here for reading,move cable to all 32 outputs in turn
#!      ;stop test when complete
#@

ABORT
#!        ;operator now moves test cable to next output
#!        ;sequence repeats 31 more times.

#! NEXT n
#!       ;turn all output drivers off
SYST:RGEN 1:DISC 1
```

```
#! Now move the rail output cable to OUTPUT 2 on the Rail
#! Generator and repeat the above test substituting A2 for A1
#! and B2 for B1.
#! Also use the command “SYST:RGEN 1:CONN 2” to enable the
#! second connector on the Rail Generator.
#! NOTE: Now that all of the pins on the variable voltage card
#! have been checked, there is no need to recheck all 32 channels
#! for the second rail generator connector.  Just one pin will
#! do.
#!      ;begin dynamic output RMS test here

*RST
TEST:DEF OUT_TEST:SIZE 2

FIELD:DEF ALL_OUTS:TYPE OT:PIN C1P32-1
SYST:FREQ 2.0 KHZ
FIELD:NAME ALL_OUTS:RADIX HEX

STIMULUS:VECTOR 1;COUNT 2;DATA:PATTERN 0,#HFFFFFFFF
STIMULUS:CONDITIONER:OFORMAT:MODE NRZ,0.0 NS
STIMULUS :CONDITIONER:OFORMAT:VOLT A
STIMULUS:COND:FET:CONN
#!      ;SET HIGH TO +3.0 AND LOW TO -3.0

SYST:RGEN 1:RAIL A1:HIGH 3.0V;LOW -3.0V
SYST:RGEN 1:CONN 1

#!      ;RUN TEST AND VERIFY EACH OUTPUT IS 3.0 VRMS +/- 100 MV
#!      ;USING METER RMS AC FUNCTION MOVING TEST CABLE TO
#!      :ALL 32 OUTPUTS WHILE TEST RUNS

SYSTEM:PROGRAMLOOP CONTINUOUS
INITIATE;*TRG

#!      ;ALL 32 PINS SHOULD OUTPUT ALTERNATING 0-1 PATTERN
#!      ;WITH .5 MS PER BIT FOR A 1 KHZ SQURE WAVE EFFECT
#!      ;pause here for reading,move cable to all 32 outputs in turn
#!      ;stop test when complete
#@

ABORT

#!      ;repeat test for b rails

STIMULUS:CONDITIONER:OFORMAT:VOLT B
SYST:RGEN 1:RAIL B1:HIGH 3.0V;LOW -3.0V
SYST:RGEN 1:CONN 1

SYSTEM:PROGRAMLOOP CONTINOUS
```

```
INITIATE;*TRG

#!      ;pause here for reading,move cable to all 32 outputs in turn
#!      ;stop test when complete
#@

ABORT

#!        ;turn all output drivers off
SYST:RGEN 1:DISC 1

#! Now move the rail output cable to OUTPUT 2 on the Rail
#! Generator and repeat the above test substituting A2 for A1
#! and B2 for B1.
#! Also use the command “SYST:RGEN 1:CONN 2” to enable the
#! second connector on the Rail Generator.
#! NOTE: Now that all of the pins on the variable voltage card
#! have been checked, there is no need to recheck all 32 channels
#! for the second rail generator connector. Just one pin will
#! do.
```

3.3 Output Timing Reference Verification

3.3.1 Internal Timing Reference Verification

3.3.1 The Clock Out connector of the SR2500VV is connected to the timer/counter input to measure the internal frequency reference. Measure the frequency of the system clock on channel A with a resolution of 1 Hz (25 ms gate on HP5335A counter). Passing criteria is from 25,012,812 Hz to 24,987,188 Hz. This criteria includes the error of the internal clock source and the counter.

Sample Code to Test the System Clock Frequency

Note

For all code samples comment lines begin with #!. Lines where a pause is required to verify a meter reading, or wait for operator, begin with #@.

```
*RST
TEST:DEF FREQTEST:SIZE 2
SYST:PROG CONT
#! SET PLL TO HIGH END, 25MHZ
FREQ 25MHZ
INIT;*TRG
#! CONNECT FREQUENCY COUNTER TO FRONT PANEL “CLOCK OUT” CONNECTOR
#! MEASURE FREQUENCY, PASSING IS 24,999,375HZ TO 25,000,625HZ
#@
ABORT
#! SET PLL TO LOW END, 12.5MHZ
FREQ 12.5MHZ
INIT;*TRG
#! MEASURE 12,499,687.5HZ TO 12,500,312.5HZ
#@
ABORT
#! NOW VERIFY FREQUENCY DIVIDER, SET FREQUENCY TO 200HZ
FREQ 200HZ
INIT;*TRG
#! MEASURE 199.995HZ TO 200.005HZ
```

3.3.2 Period Clock/Pulse Width Timing Verification

The SR2500VV output test cable from channel 00 is initially connected to the timer/counter start channel (A on HP counters), and then the other 31 outputs will be connected in turn for time interval measurement.

Note

To measure a high or low pulse width on some HP counters, it is necessary to use the common switch to tie the A and B input channels together, then set the start channel trigger edge opposite to the stop channel trigger edge (start-rising, stop-falling for high pulse width, vice versa for low pulse width).

The timer/counter input channels are set for 0.0 volt input threshold level with 50 ohm termination. The test program will output a 12.5 MHz square wave on each output with a +2.0 VDC high level and a -2.0 VDC low level. At the 0 VDC threshold level, the high and low pulse widths should be symmetrical 40 ns nominal bit times. The general procedure is;

1. Measure Output 00 pin signal pulse width using the counter’s time interval function. Using averaging if necessary, the timer resolution should be .1 ns. The pulse width passing criteria is from 33.6 to 46.4 ns for both rising and falling edge pulse widths.

2. Change the data rate to 400 Hz. Measure the pulse width as before with 1 ns resolution. Passing criteria is from 2498752 ns to 2501256 ns.

3. Repeat for all 32 remaining channels.

Sample Program for Pulse Width Timing Verification

```
*RST

TEST:DEF OUT_TEST:SIZE 2
FIELD:DEF ALL_OUTS:TYPE OT:PIN C1P32-1

#! ;set bit rate at 25 MHZ - 40 ns bit width

SYST:FREQ 25 MHZ
FIELD:NAME ALL_OUTS:RADIX HEX
STIMULUS:VECTOR 1;COUNT 2;DATA:PATTERN 0,#HFFFFFFFF
STIMULUS:CONDITIONER:OFORMAT:MODE NRZ,0.0 NS

#! ;set high to +2.0v and low to -2.0v so counter
#! ;trigger input can be at 0v

SYST:RGEN 1:RAIL A1:HIGH 2.0V;LOW -2.0V
SYST:RGEN 1:CONN 1
STIMULUS:COND:FET:CONN
STIMULUS:CONDITIONER:OFORMAT:VOLT A
SYSTEM:PROGRAMLOOP CONTINUOUS
```

```
INITIATE;*TRG

#! ;all 32 pins should output alternating 0-1 pattern
#! ;with 40 ns per bit or a 12.5 MHz square wave
#! ;verify high pulse width and low pulse width within spec
#! ;pause here for reading, move cable to all 32 in turn
#! ;stop test when complete
#@

ABORT

SYST:FREQ 400Hz

INIT;*TRG

#! ;verify high pulse width and low pulse width within spec
#! ;pause here for reading, move cable to all 32 outputs in
#! ;turn
#@

*RST
```

3.3.3 Output Skew

All the output channel edges are measured with respect to a reference (Output 00) and verified to have their output channel delays within the timing skew tolerance of each other on a given SR2500VV module. The test cable pair for the outputs being compared must be matched in length to within 0.25 inch.

The SR2500VV outputs will be set to +2.0 volts and -2.0 volts as in the previous test.

The general test sequence will be;

1. The timer/counter start channel input is connected to the Output 00 connector of the SR2500VV module. The timer/counter stop channel input will be connected to each of the other SR2500VV outputs in turn, starting with Output 01. The timer input thresholds are set to 0.0 volt. The timer is set for .1 ns resolution, using averaging if necessary.

2. Data output pattern is set for all ones with Non-Return to Zero formatting. The system test rate is set for 12.5 MHz. The output assert time delay for Output 00 will be at 0 ns. For Outputs 01-32, the assert time delay will be 10 ns. The 10 ns delay from the reference to the timed channel is to avoid negative time measurements for channel skew.

3. The test pattern is output continuously with the timer/counter measuring the average time interval from start input to stop input using at least a 1000 sample count. This time interval from the Output 00 pin will be used as the reference for other pins. This reading can be stored as the counter reference, or offset, if such a feature is available.

4. Move counter stop bit test connector from Output 01 to next output pin (Output 02) and time interval as before. The time interval shall be within +/- 3 ns of the reference interval from the first pin.

5. Repeat for remaining Outputs 03-32.

Sample Code to Test Output Skew Timing

```
*RST
TEST:DEF OUT_TEST:SIZE 2

#!      ;channel one will be clocked at 0ns, all others at 10ns

FIELD:DEF CLK:TYPE OT:PIN C1P1
FIELD:DEF ALL_OUTS:TYPE OT:PIN C1P32-2

SYST:FREQ 12.5MHz

STIMULUS:VECTOR 1;COUNT 2;DATA:FIELD CLK;PATTERN #H0,#H1
STIMULUS:VECTOR 1;COUNT 2;DATA:FIELD ALL_OUTS;PATTERN #H0,#H7FFFFFFF
STIMULUS:CONDITIONER:OFORMAT:FIELD CLK;MODE NRZ,0NS
STIMULUS:CONDITIONER:OFORMAT:FIELD ALL_OUTS;MODE NRZ,10NS

#!      ;all 32 pins should output alternating 0-1 pattern
#!      ;with 80 ns per bit and varying duty cycle
#!      ; set high to +2.0v and low to -2.0v
#!      ;trigger input can be at 0v

SYST:RGEN 1:RAIL A1:HIGH 2.0V;LOW -2.0V
SYST:RGEN 1:CONN 1
STIMULUS:COND:FET:CONN
STIMULUS:CONDITIONER:OFORMAT:VOLT A

SYSTEM:PROGRAMLOOP CONTINUOUS
INITIATE;*TRG

#!      ;verify channel 1-32 variance from reference within spec
#!      ;pause here for reading, move cable to all 32 in turn
#!      ;stop test when complete
#@

ABORT

*RST
```

3.3.4 Output Edge Placement Timing Verification

The output edge delay calibration is verified by timing the pulse widths and assert delays as they are varied across their programmable range. The timer/counter time interval function is used to measure the pulse width of each channel in turn. Counter input thresholds are again set for 0.0 volts, SR2500VV outputs for +/- 2.0 volts.

The general test sequence is;

1. The timer/counter start channel input is connected to the Output 00 connector of the SR2500VV module. The timer/counter stop channel input will be connected to each of the other SR2500VV outputs in turn, starting with Output 01. The timer input thresholds are set to 0.0 volt. The timer is set for .1 ns resolution, using averaging if necessary.

2. Output data vectors are set for all ones with Return to Zero formatting. The test data rate is set for 6.25 MHz. Assert delay time for the channel 00 signal is set at 0 ns, width is 80 ns.

3. The test is run continuously, and the average delay from channel 00 and the channel under test (from rising to rising edge) is measured using at least 1000 samples. Passing criteria is programmed delay +\- 3.0 ns (nominal +/- width error +/- 2 LSD of timer).

4. Repeat the above assert delay test while using the counter / timer to measure pulse width. Set the counter / timer to measure width from rising edge to falling edge of each channel under test. Verify pulse width of 80ns +\- 3.0 ns (nominal +/- width error +/- 2 LSD of timer) on each channel across the entire assert delay range. From 10ns to 150ns in 10ns steps.

Sample Code to Test Output Edge Placement Timing

```
*RST

TEST:DEF OUT_TEST:SIZE 2
FIELD:DEF CLK:TYPE OT:PIN C1P1
FIELD:DEF ALL_OUTS:TYPE OT:PIN C1P32-2
SYST:FREQ 6.25MHZ

STIMULUS:VECTOR 1;COUNT 2;DATA:FIELD CLK;PATTERN #H1,#H1
STIMULUS:VECTOR 1;COUNT 2;DATA:FIELD ALL_OUTS;PATTERN #H7FFFFFFF,#H7FFFFFFF
STIMULUS:CONDITIONER:OFORMAT:FIELD ALL_OUTS;MODE RZ,10NS,80NS
STIMULUS:CONDITIONER:OFORMAT:FIELD CLK;MODE RZ,0NS,80NS

SYST:RGEN 1:RAIL A1:HIGH 2.0V;LOW -2.0V
SYST:RGEN 1:CONN 1
STIMULUS:COND:FET:CONN
STIMULUS:CONDITIONER:OFORMAT:VOLT A

SYSTEM:PROGRAMLOOP CONTINUOUS

INITIATE;*TRG

#!      ;all 32 pins should output alternating 0-1 pattern
#!      ;with 160 ns per bit time, varying duty cycle
#!      ; set high to +2.0v and low to -2.0v
#!      ;trigger input can be at 0 v


#!      ;measure delay of 10ns, tolerance +/- 3.0  ns
#!      ;of nominal on channel 01, move cable to all 30 other
#!      ;channels in turn.
#!      ;pause here for reading.
#@
ABORT

STIMULUS:CONDITIONER:OFORMAT:FIELD ALL_OUTS;MODE RZ,20NS,80NS
INIT;*TRG

#!      ;measure delay of 20ns, tolerance +/- 3.0  ns
#!      ;of nominal on channel 01, move cable to all 30 other
#!      ;channels in turn.
#!     ;pause here for reading.
#@

ABORT

STIMULUS:CONDITIONER:OFORMAT:FIELD ALL_OUTS;MODE RZ,30NS,80NS
INIT;*TRG
```

```
#!      ;measure delay of 30ns, tolerance +/- 3.0  ns
#!      ;of nominal on channel 01, move cable to all 30 other
#!      ;channels in turn.
#!      ;pause here for reading.
#@

ABORT

STIMULUS:CONDITIONER:OFORMAT:FIELD ALL_OUTS;MODE RZ,40NS,80NS
INIT;*TRG

#!      ;measure delay of 40ns, tolerance +/- 3.0  ns
#!      ;of nominal on channel 01, move cable to all 30 other
#!      ;channels in turn.
#!      ;pause here for reading.
#@

ABORT

STIMULUS:CONDITIONER:OFORMAT:FIELD ALL_OUTS;MODE RZ,50NS,80NS
INIT;*TRG

#!      ;measure delay of 50NS, tolerance +/- 3.0  ns
#!      ;of nominal on channel 01, move cable to all 30 other
#!      ;channels in turn.
#!      ;pause here for reading.
#@

ABORT

STIMULUS:CONDITIONER:OFORMAT:FIELD ALL_OUTS;MODE RZ,60NS,80NS
INIT;*TRG

#!      ;measure delay of 60NS, tolerance +/- 3.0  ns
#!      ;of nominal on channel 01, move cable to all 30 other
#!      ;channels in turn.
#!      ;pause here for reading.
#@

ABORT

STIMULUS:CONDITIONER:OFORMAT:FIELD ALL_OUTS;MODE RZ,70NS,80NS
INIT;*TRG

#!      ;measure delay of 70NS, tolerance +/- 3.0  ns
#!      ;of nominal on channel 01, move cable to all 30 other
#!      ;channels in turn.
#!      ;pause here for reading.
#@
```

```
ABORT

STIMULUS:CONDITIONER:OFORMAT:FIELD ALL_OUTS;MODE RZ,80NS,80NS
INIT;*TRG

#!      ;measure delay of 80NS, tolerance +/- 3.0  ns
#!      ;of nominal on channel 01, move cable to all 30 other
#!      ;channels in turn.
#!      ;pause here for reading.
#@

ABORT

STIMULUS:CONDITIONER:OFORMAT:FIELD ALL_OUTS;MODE RZ,90NS,80NS
INIT;*TRG

#!      ;measure delay of 90NS, tolerance +/- 3.0  ns
#!      ;of nominal on channel 01, move cable to all 30 other
#!      ;channels in turn.
#!      ;pause here for reading.
#@

ABORT

STIMULUS:CONDITIONER:OFORMAT:FIELD ALL_OUTS;MODE RZ,100NS,80NS
INIT;*TRG

#!      ;measure delay of 100NS, tolerance +/- 3.0  ns
#!      ;of nominal on channel 01, move cable to all 30 other
#!      ;channels in turn.
#!      ;pause here for reading.
#@

ABORT

STIMULUS:CONDITIONER:OFORMAT:FIELD ALL_OUTS;MODE RZ,110NS,80NS
INIT;*TRG

#!      ;measure delay of 110NS, tolerance +/- 3.0  ns
#!      ;of nominal on channel 01, move cable to all 30 other
#!      ;channels in turn.
#!      ;pause here for reading.
#@

ABORT

STIMULUS:CONDITIONER:OFORMAT:FIELD ALL_OUTS;MODE RZ,120NS,80NS
INIT;*TRG
```

```
#!      ;measure delay of 120NS, tolerance +/- 3.0  ns
#!      ;of nominal on channel 01, move cable to all 30 other
#!      ;channels in turn.
#!      ;pause here for reading.
#@

ABORT

STIMULUS:CONDITIONER:OFORMAT:FIELD ALL_OUTS;MODE RZ,130NS,80NS
INIT;*TRG
#!      ;measure delay of 130NS, tolerance +/- 3.0  ns
#!      ;of nominal on channel 01, move cable to all 30 other
#!      ;channels in turn.
#!      ;pause here for reading.
#@

ABORT

STIMULUS:CONDITIONER:OFORMAT:FIELD ALL_OUTS;MODE RZ,140NS,80NS
INIT;*TRG

#!      ;measure delay of 140NS, tolerance +/- 3.0  ns
#!      ;of nominal on channel 01, move cable to all 30 other
#!      ;channels in turn.
#!      ;pause here for reading.
#@

ABORT

STIMULUS:CONDITIONER:OFORMAT:FIELD ALL_OUTS;MODE RZ,150NS,80NS
INIT;*TRG

#!      ;measure delay of 150NS, tolerance +/- 3.0  ns
#!      ;of nominal on channel 01, move cable to all 30 other
#!      ;channels in turn.
#!      ;pause here for reading.
#@

ABORT
#!      ;Now set the counter / timer to measure width from rising
#!      ;edge to falling edge of output under test.
#!      ;The pulse outputs should maintain 80ns +/- 3.0 ns
#!      ;while changing the assert delay from 10ns to 150ns
#!      ;in 10ns steps.


STIMULUS:CONDITIONER:OFORMAT:FIELD ALL_OUTS;MODE RZ,10NS,80NS
INIT;*TRG

#!      ;measure pulse width of 80ns, tolerance +/- 3.0  ns
#!      ;of nominal on channel 01, move cable to all 30 other
#!      ;channels in turn.
#!      ;pause here for reading.
#@

ABORT

STIMULUS:CONDITIONER:OFORMAT:FIELD ALL_OUTS;MODE RZ,20NS,80NS
INIT;*TRG

#!      ;measure pulse width of 80ns, tolerance +/- 3.0  ns
#!      ;of nominal on channel 01, move cable to all 30 other
#!      ;channels in turn.
#!      ;pause here for reading.
#@

ABORT

STIMULUS:CONDITIONER:OFORMAT:FIELD ALL_OUTS;MODE RZ,30NS,80NS
INIT;*TRG

#!      ;measure pulse width of 80ns, tolerance +/- 3.0  ns
#!      ;of nominal on channel 01, move cable to all 30 other
#!      ;channels in turn.
#!      ;pause here for reading.
#@

ABORT

STIMULUS:CONDITIONER:OFORMAT:FIELD ALL_OUTS;MODE RZ,40NS,80NS
INIT;*TRG

#!      ;measure pulse width of 80ns, tolerance +/- 3.0  ns
#!      ;of nominal on channel 01, move cable to all 30 other
#!      ;channels in turn.
#!      ;pause here for reading.
#@

ABORT

STIMULUS:CONDITIONER:OFORMAT:FIELD ALL_OUTS;MODE RZ,50NS,80NS
INIT;*TRG

#!      ;measure pulse width of 80ns, tolerance +/- 3.0  ns
#!      ;of nominal on channel 01, move cable to all 30 other
#!      ;channels in turn.
#!      ;pause here for reading.
#@

ABORT
```

```
STIMULUS:CONDITIONER:OFORMAT:FIELD ALL_OUTS;MODE RZ,60NS,80NS
INIT;*TRG

#!      ;measure pulse width of 80ns, tolerance +/- 3.0  ns
#!      ;of nominal on channel 01, move cable to all 30 other
#!      ;channels in turn.
#!      ;pause here for reading.
#@

ABORT

STIMULUS:CONDITIONER:OFORMAT:FIELD ALL_OUTS;MODE RZ,70NS,80NS
INIT;*TRG

#!      ;measure pulse width of 80ns, tolerance +/- 3.0  ns
#!      ;of nominal on channel 01, move cable to all 30 other
#!      ;channels in turn.
#!      ;pause here for reading.
#@

ABORT

STIMULUS:CONDITIONER:OFORMAT:FIELD ALL_OUTS;MODE RZ,80NS,80NS
INIT;*TRG

#!      ;measure pulse width of 80ns, tolerance +/- 3.0  ns
#!      ;of nominal on channel 01, move cable to all 30 other
#!      ;channels in turn.
#!      ;pause here for reading.
#@

ABORT

STIMULUS:CONDITIONER:OFORMAT:FIELD ALL_OUTS;MODE RZ,90NS,80NS
INIT;*TRG

#!      ;measure pulse width of 80ns, tolerance +/- 3.0  ns
#!      ;of nominal on channel 01, move cable to all 30 other
#!      ;channels in turn.
#!      ;pause here for reading.
#@

ABORT

STIMULUS:CONDITIONER:OFORMAT:FIELD ALL_OUTS;MODE RZ,100NS,80NS
INIT;*TRG

#!      ;measure pulse width of 80ns, tolerance +/- 3.0  ns
#!      ;of nominal on channel 01, move cable to all 30 other
#!      ;channels in turn.
#!      ;pause here for reading.
#@

ABORT

STIMULUS:CONDITIONER:OFORMAT:FIELD ALL_OUTS;MODE RZ,110NS,80NS
INIT;*TRG

#!      ;measure pulse width of 80ns, tolerance +/- 3.0  ns
#!      ;of nominal on channel 01, move cable to all 30 other
#!      ;channels in turn.
#!      ;pause here for reading.
#@

ABORT

STIMULUS:CONDITIONER:OFORMAT:FIELD ALL_OUTS;MODE RZ,120NS,80NS
INIT;*TRG

#!      ;measure pulse width of 80ns, tolerance +/- 3.0  ns
#!      ;of nominal on channel 01, move cable to all 30 other
#!      ;channels in turn.
#!      ;pause here for reading.
#@

ABORT

STIMULUS:CONDITIONER:OFORMAT:FIELD ALL_OUTS;MODE RZ,130NS,80NS
INIT;*TRG

#!      ;measure pulse width of 80ns, tolerance +/- 3.0  ns
#!      ;of nominal on channel 01, move cable to all 30 other
#!      ;channels in turn.
#!      ;pause here for reading.
#@

ABORT

STIMULUS:CONDITIONER:OFORMAT:FIELD ALL_OUTS;MODE RZ,140NS,80NS
INIT;*TRG

#!      ;measure pulse width of 80ns, tolerance +/- 3.0  ns
#!      ;of nominal on channel 01, move cable to all 30 other
#!      ;channels in turn.
#!      ;pause here for reading.
#@

ABORT

STIMULUS:CONDITIONER:OFORMAT:FIELD ALL_OUTS;MODE RZ,150NS,80NS
```

```
INIT;*TRG

#!      ;measure pulse width of 80ns, tolerance +/- 3.0  ns
#!      ;of nominal on channel 01, move cable to all 30 other
#!      ;channels in turn.
#!      ;pause here for reading.
#@

ABORT
*RST
```

4.0 Input Threshold Verification

The input thresholds are verified using an adjustable DC voltage source.

The DC voltage source is connected to all 32 channels of the variable voltage card under test, and set to a test value of 10, 50, and 90 percent of the input threshold range. The inputs will be set to record 1’s at a voltage of percent of range (10,50,or 90) minus 100mv (variable voltage receiver accuracy). To record 0’s the rail generator will be set to percent of range (10,50 or 90) plus 100mv (variable voltage receiver accuracy).

The general procedure is;

1. Connect all 32 of the SR2500VV inputs to an adjustable DC voltage source. Set the voltage to -2.06V (10% of range).
2. The Rail Generator and Variable Voltage card will be setup so that the A and B thresholds will be used in alternate 8 bit groups. The A thresholds are set to record 0’s, while the B threshold groups are set to record 1’s. Set Rail Generator Thresholds A high and low to -1.96V , and B Thresholds to -2.16V .
3. Record and examine the recorded data. Returned pattern should be #HFF00FF00. Patterns differing from this indicate bits that have failed.
4. Now the Rail Generator will be setup so that the A thresholds will record 1’s, and the B Thresholds will record 0’s. Set Rail Generator Thresholds A high and low to -2.16V , and B Thresholds high and low to -1.96V .
5. Record and examine the recorded data. Returned pattern should be #H00FF00FF. Patterns differing from this indicate bits that have failed.
6. Repeat for +1.30 Vdc (50%) and 4.66 Vdc (90%).

Sample Code to Verify Input Threshold

```
#!      ;Input threshold level test starts here

*RST
TEST:DEF IN_TEST:SIZE 10
FIELD:DEF ALL_OUTS:TYPE TRI:PIN C1P32-1
FIELD:DEF ALL_INS:TYPE REC:PIN C1P32-1
FIELD:DEF A_INS:TYPE REC:PIN C1P8-1
FIELD:DEF B_INS:TYPE REC:PIN C1P16-9
FIELD:DEF C_INS:TYPE REC:PIN C1P24-17
FIELD:DEF D_INS:TYPE REC:PIN C1P32-25
SYST:RGEN 1:CONN 1
STIM:VECT 1;COUNT ALL;DATA:FIELD ALL_OUTS;FILL:TYPE REP;PATTERN
#HFFFFFFFF;EXEC
SYST:FREQ 12.5MHZ
REC:TRAC:SEQ 1:FILTER DATA:REC ALWAYS
SYST:PROG 1
RECORD:CONDITIONER:SAMPLE:FIELD ALL_INS;MODE EDGE,10NS
RECORD:CONDITIONER:SAMPLE:field A_ins;THRES A
RECORD:CONDITIONER:SAMPLE:field B_ins;THRES B
RECORD:CONDITIONER:SAMPLE:field C_ins;THRES A
RECORD:CONDITIONER:SAMPLE:field D_ins;THRES B

SYST:RGEN 1:THRES A1:LOW -1.96V;HIGH –1.96V
SYST:RGEN 1:THRES B1:LOW –2.16V;HIGH –2.16V

#! set external voltage to –2.06v
#@

INIT;*TRG

#@
#! check for recorded pattern  #HFF00FF00


RECORD:VECTOR 1;COUNT ALL;DATA:FIELD ALL_INS;PATTERN?

#@

SYST:RGEN 1:THRES A1:LOW –2.16V;HIGH –2.16V
SYST:RGEN 1:THRES B1:HIGH –1.96V;LOW –1.96V

INIT;*TRG

#@
#! check for recorded pattern  #H00FF00FF

RECORD:VECTOR 1;COUNT all;DATA:FIELD all_ins;PATTERN?
```

```
#@

SYST:RGEN 1:THRES A1:HIGH +1.40V ;LOW +1.40V
SYST:RGEN 1:THRES B1:HIGH +1.20V ;LOW +1.20V

#! set external voltage to +1.30V
#@

INIT;*TRG

#@
#! check for recorded pattern  #HFF00FF00

RECORD:VECTOR 1;COUNT all;DATA:FIELD ALL_INS;PATTERN?

#@

SYST:RGEN 1:THRES A1:LOW +1.20V ;HIGH +1.20V
SYST:RGEN 1:THRES B1:HIGH +1.40V ;LOW +1.405V
INIT;*TRG

#@
#! check for recorded pattern  #H00FF00FF

RECORD:VECTOR 1;COUNT ALL;DATA:FIELD ALL_INS;PATTERN?

#@

SYST:RGEN 1:THRES A1:HIGH +4.76V;LOW +4.76V
SYST:RGEN 1:THRES B1:HIGH +4.56V;LOW +4.56V

#! set external voltage to +4.66V
#@

INIT;*TRG

#@
#! check for recorded pattern  #HFF00FF00

RECORD:VECTOR 1;COUNT ALL;DATA:FIELD ALL_INS;PATTERN?

#@

SYST:RGEN 1:THRES A1:LOW +4.56V;HIGH +4.56V
SYST:RGEN 1:THRES B1:HIGH +4.76V;LOW +4.76V

INIT;*TRG

#@
#! check for recorded pattern  #H00FF00FF
```

```
   RECORD:VECTOR 1;COUNT ALL;DATA:FIELD ALL_INS;PATTERN?

   #@

   SYST:RGEN 1:DISC 1

   #! now switch cable to rail generator connector 2
   #! and repeat test for connector 2
   #@

   SYST:RGEN 1:CONN 2
   SYST:RGEN 1:THRES A2:LOW –1.96V;HIGH –1.96V
   SYST:RGEN 1:THRES B2:LOW –2.16V;HIGH –2.16V

   #! set external voltage to –2.06V
   #@

   init;*trg

   #@
   #! check for recorded pattern  #HFF00FF00

   RECORD:VECTOR 1;COUNT all;DATA:FIELD ALL_INS;PATTERN?

   #@

   SYST:RGEN 1:THRES A2:LOW –2.16V;HIGH –2.16V
   SYST:RGEN 1:THRES B2:HIGH –1.96V;LOW –1.96V
   INIT;*TRG

   #@
   #! check for recorded pattern  #H00FF00FF

   RECORD:VECTOR 1;COUNT ALL;DATA:FIELD ALL_INS;PATTERN?

   #@

   SYST:RGEN 1:THRES A2:HIGH +1.40V;LOW +1.40V
   SYST:RGEN 1:THRES B2:HIGH +1.20V;LOW +1.20V

   #! set external voltage to +1.30V
   #@

   INIT;*TRG

   #@
   #! check for recorded pattern  #HFF00FF00

RECORD:VECTOR 1;COUNT ALL;DATA:FIELD ALL_INS;PATTERN?
```

```
#@

SYST:RGEN 1:THRES A2:LOW +1.20V ;HIGH +1.20V
SYST:RGEN 1:THRES B2:HIGH +1.40V;LOW +1.40V
INIT;*TRG

#@
#! check for recorded pattern  #H00FF00FF

RECORD:VECTOR 1;COUNT ALL;DATA:FIELD ALL_INS;PATTERN?

#@

SYST:RGEN 1:THRES A2:HIGH +4.76V;LOW +4.76V
SYST:RGEN 1:THRES B2:HIGH +4.56V;LOW +4.56V

#! set external voltage to +4.66V
#@

INIT;*TRG

#@
#! check for recorded pattern  #HFF00FF00

RECORD:VECTOR 1;COUNT ALL;DATA:FIELD ALL_INS;PATTERN?

#@

SYST:RGEN 1:THRES A2:LOW +4.56V ;HIGH +4.56V
SYST:RGEN 1:THRES B2:HIGH +4.76V ;LOW +4.76V
INIT;*TRG

#@

#! check for recorded pattern  #H00FF00FF

RECORD:VECTOR 1;COUNT ALL;DATA:FIELD ALL_INS;PATTERN?

#@
SYST:RGEN 1:DISC 2
*RST
```

4.2 Input Timing Verification

With confidence in the output levels, output timing, and input thresholds established, the outputs can now be used to verify operation of the input timing. The outputs will be set to RZ format and be used to generate a pulse that the inputs will record in edge mode. The assert time of the output pulses will be incremented in 10ns steps across a 160 ns period. The response will be setup to track the moving stimulus. The recorded pattern will then be compared to what is expected. Any deviation from the expected pattern indicates a record timing error.

The general procedure will be;

1. Set output data pattern for all channels to alternating one-zero, with a depth of 10 output vectors. Two 16 bit output fields are set up for RZ format with 30ns width. The two output fields are offset by 10ns from each other. Output high level is set to +2V, and output low level is set to -2V.
2. The test frequency is set for 6.25 MHz, all input thresholds set for 0 volts. Two 16 bit Edge record fields are setup. The edge fields will be offset from the stim fields by 70ns. The data is recorded and the pattern compared to what is expected. Any bad bits indicate a timing problem in the response.
3. Increment stimulus and response delays by 10ns and compare patterns, until the entire 160ns period is covered.

Sample SR2500(VV) Code to Verify Input Timing

```
*RST
TEST:DEF TA:SIZE 10
SYST:RGEN 1:THRES A1:HIGH 0.0;LOW 0.0
SYST:RGEN 1:RAIL A1:HIGH 2.0;LOW -2.0
SYST:RGEN 1:CONN 1
FIELD:DEF FOTALL:TYPE OT:PIN C1P32-1
STIM:COND:OFORMAT:FIELD FOTALL;VOLT A
FIELD:DEF FRECALL:TYPE REC:PIN C1P32-1
REC:COND:SAMPLE:FIELD FRECALL;THRES A
FIELD:DEF FLO:TYPE OT:PIN C1P16-1
FIELD:DEF FHI:TYPE OT:PIN C1P32-17

FIELD:DEF FRECLO:TYPE REC:PIN C1P16-1
FIELD:DEF FRECHI:TYPE REC:PIN C1P32-17
STIM:VECT 1;COUNT ALL;DATA:FIELD FLO;FILL:TYPE ALT;PATTERN #H0000;EXEC
STIM:VECT 1;COUNT ALL;DATA:FIELD FHI;FILL:TYPE ALT;PATTERN #H0000;EXEC
REC:TRAC:SEQ 1:FILTER DATA:REC ALWAYS
REC:COND:SAMP:FIELD FRECLO;EOFF 0
REC:COND:SAMP:FIELD FRECHI;EOFF 0
SYST:PROG 1
SYST:FREQ 6.25MHZ
```

```
STIM:COND:OFORM:FIELD FLO;MODE RZ,00NS,30NS
STIM:COND:OFORM:FIELD FHI;MODE RZ,10NS,30NS
REC:COND:SAMP:FIELD FRECHI;MODE EDGE,80NS
REC:COND:SAMP:FIELD FRECLO;MODE EDGE,70NS
INIT;*TRG

#@
#! Check that test is in idle state so that rec query can be performed

TEST:NAME TA:STAT?

#! check for pattern #h00000000,#hFFFFFFFF,#h00000000,#hFFFFFFFF etc.

REC:VECT 1;COUNT 10;DATA:FIELD FRECALL;PATT?

#@

STIM:COND:OFORM:FIELD FLO;MODE RZ,10NS,30NS
STIM:COND:OFORM:FIELD FHI;MODE RZ,20NS,30NS
REC:COND:SAMP:FIELD FRECHI;MODE EDGE,90NS
REC:COND:SAMP:FIELD FRECLO;MODE EDGE,80NS
INIT;*TRG

#@

TEST:NAME TA:STAT?

#! check for pattern #h00000000,#hFFFFFFFF,#h00000000,#hFFFFFFFF etc.

REC:VECT 1;COUNT 10;DATA:FIELD FRECALL;PATT?

#@

STIM:COND:OFORM:FIELD FLO;MODE RZ,20NS,30NS
STIM:COND:OFORM:FIELD FHI;MODE RZ,30NS,30NS
REC:COND:SAMP:FIELD FRECHI;MODE EDGE,100NS
REC:COND:SAMP:FIELD FRECLO;MODE EDGE,90NS
INIT;*TRG

#@

TEST:NAME TA:STAT?

#! check for pattern #h00000000,#hFFFFFFFF,#h00000000,#hFFFFFFFF etc.

REC:VECT 1;COUNT 10;DATA:FIELD FRECALL;PATT?

#@

STIM:COND:OFORM:FIELD FLO;MODE RZ,30NS,30NS
```

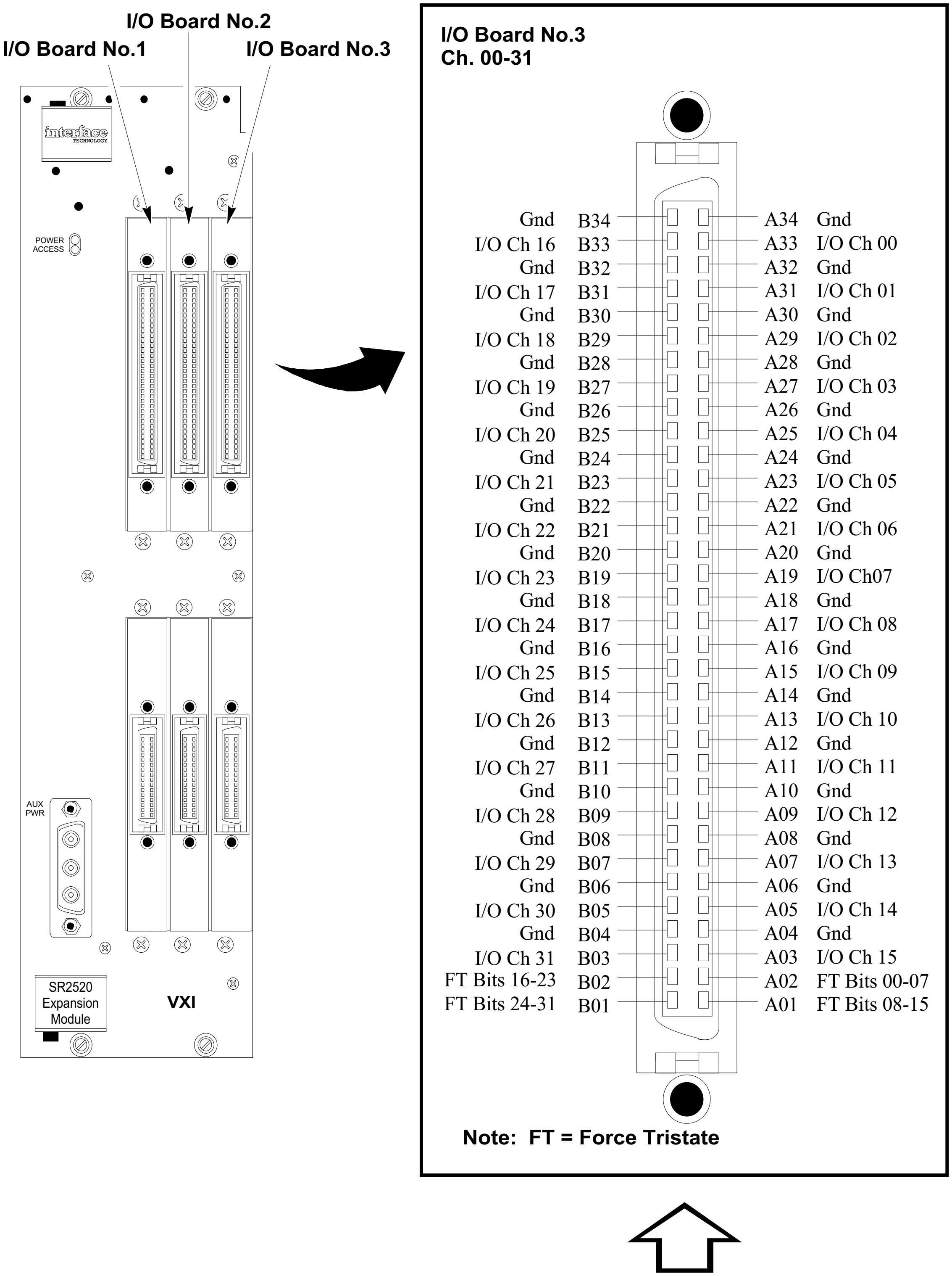
```
STIM:COND:OFORM:FIELD FHI;MODE RZ,40NS,30NS
REC:COND:SAMP:FIELD FRECHI;MODE EDGE,110NS
REC:COND:SAMP:FIELD FRECLO;MODE EDGE,100NS
INIT;*TRG

#@

TEST:NAME TA:STAT?

#! check for pattern #h00000000,#hFFFFFFFF,#h00000000,#hFFFFFFFF etc.

REC:VECT 1;COUNT 10;DATA:FIELD FRECALL;PATT?

#@

STIM:COND:OFORM:FIELD FLO;MODE RZ,40NS,30NS
STIM:COND:OFORM:FIELD FHI;MODE RZ,50NS,30NS
REC:COND:SAMP:FIELD FRECHI;MODE EDGE,120NS
REC:COND:SAMP:FIELD FRECLO;MODE EDGE,110NS
INIT;*TRG

#@

TEST:NAME TA:STAT?

#! check for pattern #h00000000,#hFFFFFFFF,#h00000000,#hFFFFFFFF etc.

REC:VECT 1;COUNT 10;DATA:FIELD FRECALL;PATT?

#@

STIM:COND:OFORM:FIELD FLO;MODE RZ,50NS,30NS
STIM:COND:OFORM:FIELD FHI;MODE RZ,60NS,30NS
REC:COND:SAMP:FIELD FRECHI;MODE EDGE,130NS
REC:COND:SAMP:FIELD FRECLO;MODE EDGE,120NS
INIT;*TRG

#@

TEST:NAME TA:STAT?

#! check for pattern #h00000000,#hFFFFFFFF,#h00000000,#hFFFFFFFF etc.

REC:VECT 1;COUNT 10;DATA:FIELD FRECALL;PATT?

#@

STIM:COND:OFORM:FIELD FLO;MODE RZ,60NS,30NS
STIM:COND:OFORM:FIELD FHI;MODE RZ,70NS,30NS
REC:COND:SAMP:FIELD FRECHI;MODE EDGE,140NS
```

```
REC:COND:SAMP:FIELD FRECLO;MODE EDGE,130NS
INIT;*TRG

#@

TEST:NAME TA:STAT?

#! check for pattern #h00000000,#hFFFFFFFF,#h00000000,#hFFFFFFFF etc.

REC:VECT 1;COUNT 10;DATA:FIELD FRECALL;PATT?

#@

STIM:COND:OFORM:FIELD FLO;MODE RZ,70NS,30NS
STIM:COND:OFORM:FIELD FHI;MODE RZ,80NS,30NS
REC:COND:SAMP:FIELD FRECHI;MODE EDGE,150NS
REC:COND:SAMP:FIELD FRECLO;MODE EDGE,140NS
INIT;*TRG

#@

TEST:NAME TA:STAT?

#! check for pattern #h00000000,#hFFFFFFFF,#h00000000,#hFFFFFFFF etc.

REC:VECT 1;COUNT 10;DATA:FIELD FRECALL;PATT?

#@

STIM:COND:OFORM:FIELD FLO;MODE RZ,80NS,30NS
STIM:COND:OFORM:FIELD FHI;MODE RZ,90NS,30NS
REC:COND:SAMP:FIELD FRECLO;MODE EDGE,150NS
REC:COND:SAMP:FIELD FRECHI;MODE EDGE,00NS;EOFF 1
INIT;*TRG

#@

TEST:NAME TA:STAT?

#! check for pattern #h00000000,#hFFFFFFFF,#h00000000,#hFFFFFFFF etc.


REC:VECT 1;COUNT 10;DATA:FIELD FRECALL;PATT?

#@

STIM:COND:OFORM:FIELD FLO;MODE RZ,90NS,30NS
STIM:COND:OFORM:FIELD FHI;MODE RZ,100NS,30NS
REC:COND:SAMP:FIELD ALL;CLEAR
REC:COND:SAMP:FIELD FRECHI;MODE EDGE,10NS;EOFF 1
```

```
REC:COND:SAMP:FIELD FRECLO;MODE EDGE,00NS;EOFF 1
INIT;*TRG

#@

TEST:NAME TA:STAT?

#! check for pattern #h00000000,#hFFFFFFFF,#h00000000,#hFFFFFFFF etc.

REC:VECT 1;COUNT 10;DATA:FIELD FRECALL;PATT?

#@

STIM:COND:OFORM:FIELD FLO;MODE RZ,100NS,30NS
STIM:COND:OFORM:FIELD FHI;MODE RZ,110NS,30NS
REC:COND:SAMP:FIELD FRECHI;MODE EDGE,20NS
REC:COND:SAMP:FIELD FRECLO;MODE EDGE,10NS
INIT;*TRG

#@

TEST:NAME TA:STAT?

#! check for pattern #h00000000,#hFFFFFFFF,#h00000000,#hFFFFFFFF etc.

REC:VECT 1;COUNT 10;DATA:FIELD FRECALL;PATT?

#@

STIM:COND:OFORM:FIELD FLO;MODE RZ,110NS,30NS
STIM:COND:OFORM:FIELD FHI;MODE RZ,120NS,30NS
REC:COND:SAMP:FIELD FRECHI;MODE EDGE,30NS
REC:COND:SAMP:FIELD FRECLO;MODE EDGE,20NS
INIT;*TRG

#@

TEST:NAME TA:STAT?

#! check for pattern #h00000000,#hFFFFFFFF,#h00000000,#hFFFFFFFF etc.

REC:VECT 1;COUNT 10;DATA:FIELD FRECALL;PATT?

#@

STIM:COND:OFORM:FIELD FLO;MODE RZ,120NS,30NS
STIM:COND:OFORM:FIELD FHI;MODE RZ,130NS,30NS
REC:COND:SAMP:FIELD FRECHI;MODE EDGE,40NS
REC:COND:SAMP:FIELD FRECLO;MODE EDGE,30NS
INIT;*TRG
```

```
#@

TEST:NAME TA:STAT?

#! check for pattern #h00000000,#hFFFFFFFF,#h00000000,#hFFFFFFFF etc.

REC:VECT 1;COUNT 10;DATA:FIELD FRECALL;PATT?

#@

STIM:COND:OFORM:FIELD FLO;MODE RZ,130NS,30NS
STIM:COND:OFORM:FIELD FHI;MODE RZ,140NS,30NS
REC:COND:SAMP:FIELD FRECHI;MODE EDGE,50NS
REC:COND:SAMP:FIELD FRECLO;MODE EDGE,40NS
INIT;*TRG

#@

TEST:NAME TA:STAT?

#! check for pattern #h00000000,#hFFFFFFFF,#h00000000,#hFFFFFFFF etc.

REC:VECT 1;COUNT 10;DATA:FIELD FRECALL;PATT?

#@

STIM:COND:OFORM:FIELD FLO;MODE RZ,140NS,30NS
STIM:COND:OFORM:FIELD FHI;MODE RZ,150NS,30NS
REC:COND:SAMP:FIELD FRECHI;MODE EDGE,60NS
REC:COND:SAMP:FIELD FRECLO;MODE EDGE,50NS
INIT;*TRG

#@

TEST:NAME TA:STAT?

#! check for pattern #h00000000,#hFFFFFFFF,#h00000000,#hFFFFFFFF etc.

REC:VECT 1;COUNT 10;DATA:FIELD FRECALL;PATT?

#@

STIM:COND:OFORM:FIELD FLO;MODE RZ,150NS,30NS
STIM:COND:OFORM:FIELD FHI;MODE RZ,000NS,30NS
REC:COND:SAMP:FIELD FRECHI;MODE EDGE,70NS
REC:COND:SAMP:FIELD FRECLO;MODE EDGE,60NS
INIT;*TRG

#@
```

```
TEST:NAME TA:STAT?

#! check for pattern #h00000000,#hFFFFFFFF,#h00000000,#hFFFFFFFF etc.

REC:VECT 1;COUNT 10;DATA:FIELD FRECALL;PATT?
```

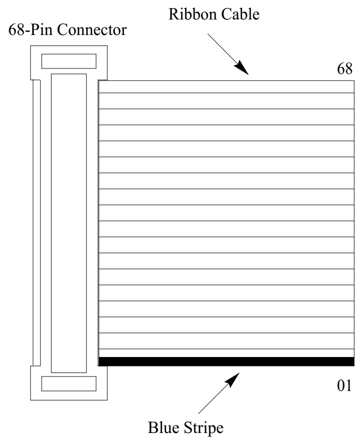


I/O Board No.3
Ch. 00-31

Gnd	B34	A34	Gnd
I/O Ch 16	B33	A33	I/O Ch 00
Gnd	B32	A32	Gnd
I/O Ch 17	B31	A31	I/O Ch 01
Gnd	B30	A30	Gnd
I/O Ch 18	B29	A29	I/O Ch 02
Gnd	B28	A28	Gnd
I/O Ch 19	B27	A27	I/O Ch 03
Gnd	B26	A26	Gnd
I/O Ch 20	B25	A25	I/O Ch 04
Gnd	B24	A24	Gnd
I/O Ch 21	B23	A23	I/O Ch 05
Gnd	B22	A22	Gnd
I/O Ch 22	B21	A21	I/O Ch 06
Gnd	B20	A20	Gnd
I/O Ch 23	B19	A19	I/O Ch07
Gnd	B18	A18	Gnd
I/O Ch 24	B17	A17	I/O Ch 08
Gnd	B16	A16	Gnd
I/O Ch 25	B15	A15	I/O Ch 09
Gnd	B14	A14	Gnd
I/O Ch 26	B13	A13	I/O Ch 10
Gnd	B12	A12	Gnd
I/O Ch 27	B11	A11	I/O Ch 11
Gnd	B10	A10	Gnd
I/O Ch 28	B09	A09	I/O Ch 12
Gnd	B08	A08	Gnd
I/O Ch 29	B07	A07	I/O Ch 13
Gnd	B06	A06	Gnd
I/O Ch 30	B05	A05	I/O Ch 14
Gnd	B04	A04	Gnd
I/O Ch 31	B03	A03	I/O Ch 15
FT Bits 16-23	B02	A02	FT Bits 00-07
FT Bits 24-31	B01	A01	FT Bits 08-15

Note: FT = Force Tristate

Figure 1. I/O Channel Pin Locations.



Wiring Diagram
SR2500 Signal Breakout Cable
(Variable Voltage)

68	Gnd
67	Gnd
66	I/O 00
65	I/O 16
64	Gnd
63	Gnd
62	I/O 01
61	I/O 17
60	Gnd
59	Gnd
58	I/O 02
57	I/O 18
56	Gnd
55	Gnd
54	I/O 03
53	I/O 19
52	Gnd
51	Gnd
50	I/O 04
49	I/O 20
48	Gnd
47	Gnd
46	I/O 05
45	I/O 21
44	Gnd
43	Gnd
42	I/O 06
41	I/O 22
40	Gnd
39	Gnd
38	I/O 07
37	I/O 23
36	Gnd
35	Gnd
34	I/O 08
33	I/O 24
32	Gnd
31	Gnd
30	I/O 09
29	I/O 25
28	Gnd
27	Gnd
26	I/O 10
25	I/O 26
24	Gnd
23	Gnd
22	I/O 11
21	I/O 27
20	Gnd
19	Gnd
18	I/O 12
17	I/O 28
16	Gnd
15	Gnd
14	I/O 13
13	I/O 29
12	Gnd
11	Gnd
10	I/O 14
09	I/O 30
08	Gnd
07	Gnd
06	I/O 15
05	I/O 31
04	Force Tristate 00-07
03	Force Tristate 16-23
02	Force Tristate 08-15
01	Force Tristate 24-31

Figure 2. Breakout Cable.

(THIS PAGE INTENTIONALLY LEFT BLANK)

AppNotes & TechNotes

AppNotes & TechNotes

Note:
This section contains Application Notes and Technical Notes describing the technical details and applications of the subject equipment.

(THIS PAGE LEFT BLANK INTENTIONALLY)

App/Tech Note

Selecting a VXI Test System for Bus Emulation SR2500-01

Purpose and Scope

Bus emulation is one of the most sophisticated applications for digital testing and requires careful consideration when selecting a suitable test system. This application note is intended for the test engineer or engineering manager actually involved with selecting such a test system.

We will begin by discussing the various aspects of our test *problem* which, in this case, is the particular bus emulation test that we wish to perform. Next, we will define the parameters that are important in making the test ... notably the timing parameters with which we must concern ourselves. Finally, we will apply what we have learned to selecting a suitable *solution* to the test problem ... namely, selecting a test system to perform the bus emulation.

The entire process of defining the test and selecting a suitable tester is presented in an easy-to-follow, step-by-step process that is both easy to understand and easy to remember.

Once the principles of bus emulation are understood, we will touch on some of the bonus features that are available on today's modern digital test systems to enhance tester performance and produce better accuracy ... features such as *data formatting, programmable edge placement,* and *algorithmic pattern generation*. We will also touch briefly on tools for evaluating UUT response ... like *real-time compare, signature analysis, guided probe*, and *fault directories*. We will conclude our discussion with a few words on interconnect cabling between the tester and the UUT (unit under test).

The ISA Bus Is An Example

To keep things simple, we’ll limit our discussion of bus emulation to the ISA bus (Industry Standard Architecture). This bus is widely used in personal computers, and is familiar to many test engineers. In the course of our discussion, we will address several of the issues that you are likely to encounter in testing a circuit board designed for use on an ISA bus.

Although the discussion here is limited to the ISA bus architecture, the same principles apply as well to any other type of bus you are apt to encounter.

Step 1 --
Define What Is To Be Tested

In this discussion, we are going to emulate the ISA bus in performing a simple memory read/write subroutine. This subroutine will include two simple operations ...

1. select and latch a desired memory address and ...

2. write data to, or read data from, the selected memory address.

Fig 1 is a timing diagram showing the two operations. The first operation (select and latch a desired memory address) is represented by the top three waveforms:

o LA <23:17>
o BALE
o SA <19:0>

The second operation (writing to or reading data from the selected memory address) is represented by the 4th and 5th waveforms, namely ...

o MEMR* or MEMW*
o SD <15:0>

(note the asterisk (*) after MEMR* and MEMW* that denote these signals use negated logic.)

The bus clock (BCLK) is shown for reference only.

Step 2 --
Determine Pin Requirements

As depicted in Fig 1, the ISA bus has a 16-bit data bus (SD <15:0>), a 20-bit address bus (SA <19:0>), and a 7-bit latchable address bus (LA <23:17>), plus various control signals ... most notably, bus address latch enable (BALE), memory read/memory write (MEMR*/MEMW*) and, of course, the bus clock (BCLK). Just from this information alone, we already know that the tester we select must provide at least 43 channels. We also know that 16 of these channels must be bi-directional to support the bi-directional data bus. The bi-directional channels require two types of memory to provide state-by-state control of the output drivers ... that is, a stimulus memory and a separate tristate control memory. Now, let’s examine the timing parameters.

Step 3 --
Determine Timing Parameters

As the next step in selecting a test system for bus emulation, we must carefully note all of the critical timing parameters involved, both for the memory chip and for the ISA bus. These parameters can be obtained from the ISA bus specification, and from the memory chip manufacturer's technical literature, or from a timing chart such as presented in Fig 1. In the order in which they occur, the parameters of interest are:

1. Bus cycle duration
2. Clock period
3. Address latch (LA <23:17>) deassert time
4. Bus address latch enable (BALE) assert time
5. Address signal (SA<19:0>) assert time
6. Bus address latch enable (BALE) deassert time
7. Data signal (SD<15:0>) assert time
8. MEMR* / MEMW* assert time
9. Address latch (LA <23:17>) assert time
10. MEMR* / MEMW* deassert time
11. Data signal (SD<15:0>) deassert time
12. Address signal (SA<19:0>) deassert time

Bus Cycle Duration and Clock Period.

From our timing chart (Fig. 1), we note that the duration of a single bus cycle is 470 ns. We also note that the duration of the bus clock period (BCLK) is 40 ns ... which equates to a bus frequency of (1 / 40^{-9} = 25^{6} = 25 MHz). We now have two of the 12 timing parameters we need.

Address Timing Parameters.

The first event that occurs when making a memory read/write on the ISA bus is to select the memory address to which data will be written, or from which data will be read. The timing events for this operation involve three signals:

1. memory address latch (LA <23:17>)
2. bus address latch enable (BALE).
3. the address signal (SA <19:0>)

These three timing signals are shown as the top three waveforms of Fig 1.

Address Latch. See Fig 1. The first event that occurs when selecting a memory address is to clear (negate) all of the address latch lines (LA <23:17>). This occurs at time T0 when the seven address latch lines (LA <23:17>) are deasserted, (Fig 1, top waveform.) Next, the bus address latch enable (BALE) line is asserted at T = 50 ns into the bus cycle, (Fig 1, BALE waveform.) BALE pulse width is 60 ns ... that is to say, once asserted BALE remains asserted for 60 ns. During the time BALE is asserted, the address signal (SA <19:0>) is placed on the bus. In more specific terms, SA <19:0> is asserted 30 ns after BALE is asserted ... at the mid point of the BALE pulse width. Our address has now been selected and we are ready to write data to, or read data from, the selected memory.

Memory Read/Write. Now that the desired address has been selected, the next event to occur is the actual memory read or write. By again referring to the timing chart (Fig 1) we see that the time interval between BALE being deasserted and MEMR*/MEMW* being asserted is a relative interval of 10 ns. This is represented by T1 and T9, respectively, both of which depict time in relationship to T0 (i.e., T1=110 ns, T9 = 120 ns).

We now have all of the necessary timing parameters that we need to conduct our bus emulation test, see Table 1.

Step 4 -- Select Test System

With all of our critical timing parameters identified, we are ready to select a suitable test system to perform the bus emulation test. Let’s say we begin by selecting a common, garden variety, VXI digital word generator ... that is, one without data formatting or programmable edge placement. This means that our tester will not provide us with any sort of control over when a particular signal transition occurs, other than during an actual clock transition. Hence, we must force all signal transitions to occur on tester clock boundaries. And, just for argument’s sake, let’s say the data rate of our pattern generator is 25 MHz (clock rate = 1 / 25,000,000 = 40 ns.)

Our digital word generator has a 25 MHz data rate. This means that any signal transition will occur only on the 40 ns clock boundaries ... in other words, only when the clock signal is asserted. Note, for instance, that the timing difference between BALE being deasserted (T1 = 110 ns) and MEMR* / MEMW* being asserted (T9 =120 ns) is only 10 ns. Since 10 ns is less than the 40 ns clock period of our word generator, we must wait for the next clock transition before we can change the state of MEMR* / MEMW* ... that is, we must extend T1 to the next 40 ns boundary at 120 ns (See Figure 2). Since we must wait an additional 30 ns for a clock transition, the overall speed at which bus emulation can be performed is reduced. And, what’s even worse, this effect is compounded every bus cycle so that extending the timing of one signal results in a “ripple effect” on all other signals referenced to it.

Programming state changes and edge timing by just using test vectors will not only degrade bus timing, it will also quickly use up available memory. Note that in Fig 1 there are a total of 12 bus signal transitions. To create these same 12 transitions using a test system without programmable edge placement will require 12 vectors, since we can only change state on a clock boundary. In Fig 2, the basic timing wave form represented by Fig 1

Table 1. ISA Bus Timing Parameters.

Signal	Description	Transition at ...
LA<23:17>	Address Latch Deassert	0 ns
BALE	Bus Address Latch Assert # 1	50 ns
SA <19:0>	Address Signal Assert	80 ns
SD <15:0>	Data Signal Assert	90 ns
BALE	Bus Address Latch Deassert # 1	110 ns
MEMR*/MEMW*	Memory Read / Memory Write Assert	120 ns
LA<23:17>	Address Latch Assert	140 ns
MEMR*/MEMW*	Memory Read / Memory Write Deassert	360 ns
SD <15:0>	Data Signal Deassert	390 ns
BALE	Bus Address Latch Assert #2	400 ns
SA <19:0>	Data Signal Deassert	410 ns
BALE	Bus Address Latch Deassert #2	460 ns
Bus Cycle	Bus Cycle	470 ns
BCLK	Bus Clock (25 MHz)	40 ns

was recreated using the timing resolution of our simple 25 MHz tester that does not have programmable edge placement or data formatting. The best timing resolution at this data rate is 40 ns ... which is the same as the clock rate. In this example, twelve 40 ns vectors must be stored in the tester to complete a single 470 ns ISA bus cycle (12 x 40 ns = or 480 ns.) In this case, two undesirable things occurred ...

1. the length of the normal bus cycle (470 ns) was increased from 470 ns to 480 ns and ...
2. since we must use 12 test vectors for each bus cycle, tester pattern memory depth has been reduced by a factor of 12.

The degrading effect this has on the ability of the tester to test a bus device is quite dramatic; for example, testing a block of memory mapped to a range of addresses on the bus. If 10 test vectors, for example, are required to complete one bus cycle (instead of 12 as in our previous example), then a tester with a 64 K test pattern depth is limited to testing only a (64 K / 10 = 6.4 K) portion of the RAM in any given test. Likewise, to test a 1M RAM would require downloading new address and data patterns into the tester's pattern memory 160 times (i.e., 1,024,000 / 6,400 = 160). Each download can represent a significant proportion of the total time it takes to test the device.

Programmable Edge Placement

A tester with programmable edge placement has the capability to delay the occurrence of each signal on the bus relative to any other signal. This overcomes the problem of having to wait for clock transitions to change signal states by permitting signal transitions to occur at any point within the tester's clock cycle. Hence, the otherwise difficult task of matching setup and hold parameters, relative to strobes and read or write controls, becomes a simple matter of programming.

Figures 1 and 2 illustrate two important criteria in emulating any digital bus, including ISA. These are ...

1. the importance of the timing relationships between the various signals and ...
2. the efficient use of tester resources, specifically the pattern memory.

Data Formatting

Data formatting is the ability to apply a format pattern to a group of pins. The most common digital formats are ...

o Return-to-Zero (RZ)
o Return-to-One (R1)
o Return-to-Inhibit (RINH); also called Return-to-Tristate

Fig 3 shows a bus cycle represented using only one test vector. In this example, the tester we selected for our bus emulation test supports both programmable edge placement and data formatting. A more efficient use of memory is achieved because one ISA bus cycle can be represented by just one test vector. This is possible because signals like BALE, which has two pulses in the middle of the bus cycle, can be represented using an RZ data format with a pulse delay of D1 for the first pulse, measured from the beginning of the test cycle, and a pulse width of P1, followed by another pulse with a delay of D2, (again referenced to T0 of the bus clock) followed by a pulse width of P2. Thus, regardless of the length of the ISA bus cycle, all signals and their phase relationships are easily represented using a single test vector.

The benefits of performing a bus read or write using a single test vector should be obvious. The aforementioned 64 K pattern memory in the tester would now fully test 64 K of RAM. Also, the number of pattern downloads needed to test one megabyte of RAM is reduced from a previous high of 160 down to just 16.

Algorithmic Pattern Generation

In the case of RAM-backed pattern generation, even though the time to test is dramatically reduced by using programmable edge timing and data formatting, it still takes much longer to *load* a RAM test than the time needed to *run* a RAM test. The ideal solution is to test the full address range of the RAM after only a single pattern load. You would still need to load the test program into the tester the first time, but further test loading would be unnecessary. The time to test each 1M of RAM is only 0.5

seconds. With no need to load new test programs to test successive blocks of the RAM, test time is reduced by orders of magnitude. Modern digital testers now provide algorithmic test functions that are especially well suited for testing RAM, or other sequentially addressed locations, on a bus. Algorithmic pattern generation lets you define the desired pattern as an algorithm, or function, instead of as patterns stored in RAM (i.e., RAM-backed patterns). The patterns are generated in real-time, via a high speed state machine, while the test is being executed. To use algorithmic patterns, you initially define the starting address and an incrementing pattern for the address bus, and an equally suitable pattern for the data bus (e.g., an alternating 1's and 0's pattern.) Then, simply repeat the test vector containing the algorithmic commands 1 million times. A full megabyte RAM test is performed using less than ten test vectors. With algorithmic digital testers, you can test very deep memory devices using a small fraction of the testers available pattern memory. Since the whole process can be represented in only a few test vectors, test download time is proportionately reduced, again reducing the time to test the UUT.

Diagnostics Tools for Evaluating UUT Response

Generating stimulus patterns for the ISA bus is only performing half of the overall test. You also need a mechanism for determining how the UUT responds to the test stimulus. Otherwise, it remains uncertain as to whether or not the UUT is operating properly. Historically, the instruments used for evaluating UUT response have been the oscilloscopes and the logic analyzer ... two instruments, the output of which require human interpretation to identify problems. These instruments are better suited to the lab than to the production floor. What is needed on the production floor is automated diagnostics that do not require frequent operator intervention.

Response Recording

Perhaps the most obvious technique for evaluating UUT performance is simply to record the UUT response into a memory specifically reserved for that purpose, much like a conventional logic analyzer. Uploading the captured UUT response data to a host computer for comparison with an expected response can thus determine the *pass* or *fail* status of the UUT. There are, however, two major limitations to this approach ... first, the time it takes to move data to and from the tester's pattern memory and second, the time it takes for the host computer to perform a comparison between the expected good response and the actual response from the UUT. What we need is some method to conduct *real-time comparisons* of expected and actual UUT responses. Fortunately, such a method already exists.

Real-Time Compare

Digital testers sometimes offer *real-time compare* built into the hardware. By loading the known good (or expected) response into the tester and enabling the *real-time compare function*, the pass/fail comparison is performed in real-time as the test proceeds. The results of the test are thus immediately available to the test system, indicating either a pass or fail of the UUT. In addition to determining pass or fail, testers with real-time compare usually provide a choice of recording either the raw response from the UUT, or the results of the compare, which indicates the bit or bits in error. Having this information available aids the diagnostic and trouble-shooting stage of repairing a defective board.

Other response tools exist which aid the test operator in evaluating the functionality of the ISA board, signature analysis, guided probes and fault dictionaries being the more common.

Signature Analysis

When included as a feature of a digital tester, *signature analysis* generates a checksum for each node being probed on the UUT. Since the stimulus pattern provided by the tester is the same for all UUTs being tested, the checksum for each respective node on the UUT should also be the same. If a checksum mismatch occurs, a failure has occurred, and can usually be traced back to the faulty component.

Avoiding Cable Losses

Cable routing and transmission line loss play an important role in delivering a usable signal from the digital tester to the bus being emulated. The longer the interconnect cable between the tester and the UUT, the greater the chance of signal degradation, and the greater the need for properly terminating the transmission line. Some digital testers support only a single type of termination, or a single impedance cable, which may not be suitable for all applications. Coax cables, ribbon cables , and twisted ribbon cables are used successfully in a wide variety of digital test application, including bus emulation.

Coax Cables

Coax cables, with a characteristic impedance of 50 ohms, are routinely used for higher speed logic, but are generally undesirable for TTL due to the added load the termination resistor would place on the TTL driver, and due to the size and weight of the shielded 50 ohm coax cables.

Ribbon Cables

Ribbon cables provide a much denser cable package than coax, and with a characteristic impedance of 100 ohms, require less source current from the driver for the terminating resistor. However, with ribbon cables, each signal must include a ground return to prevent introducing ground noise into the test environment, and provide some channel-to-channel shielding to reduce crosstalk. The most common practice with ribbon cables is to alternate signal and ground conductors across the width of the cable. The tester that supports multiple I/O termination formats will usually deliver better signal quality to the bus without requiring external signal conditioning.

Summary

Selecting a VXI test system for bus emulation is a four step process.

1. Define what is to be tested.
2. Determine pin requirements.
3. Determine timing parameters.
4. Select the test system.

In selecting a test system, you should always select a tester with a pattern rate fast enough to test any UUT on the bus at actual operating speed. Even more important than pattern rate, however, is a tester that provides the features you will need to accurately emulate all of the critical timing parameters of the bus, and one which will provide "on-the-fly" identification of problems and quick problem isolation. Such features include ...

- Programmable edge placement
- Data formatting
- Algorithmic pattern generation
- Response recording with real-time compare
- Signature analysis
- Guided probe (optional)
- Fault directory (optional)

Finally, make sure that cabling between the tester and the UUT is kept short and is properly terminated to minimize signal distortion and transmission loss.

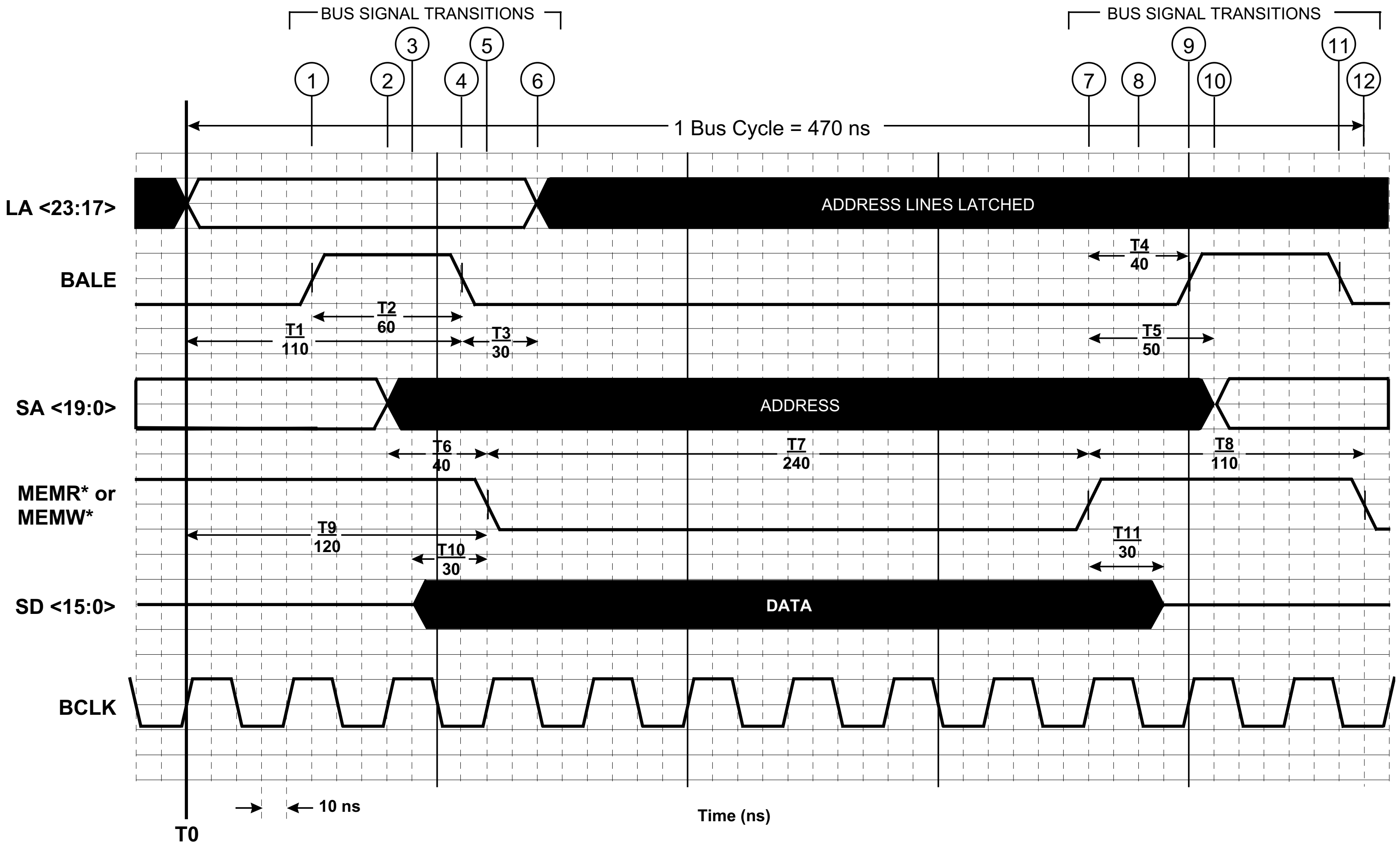


Figure 1.
ISA Bus Memory Read or Write Timing Diagram.

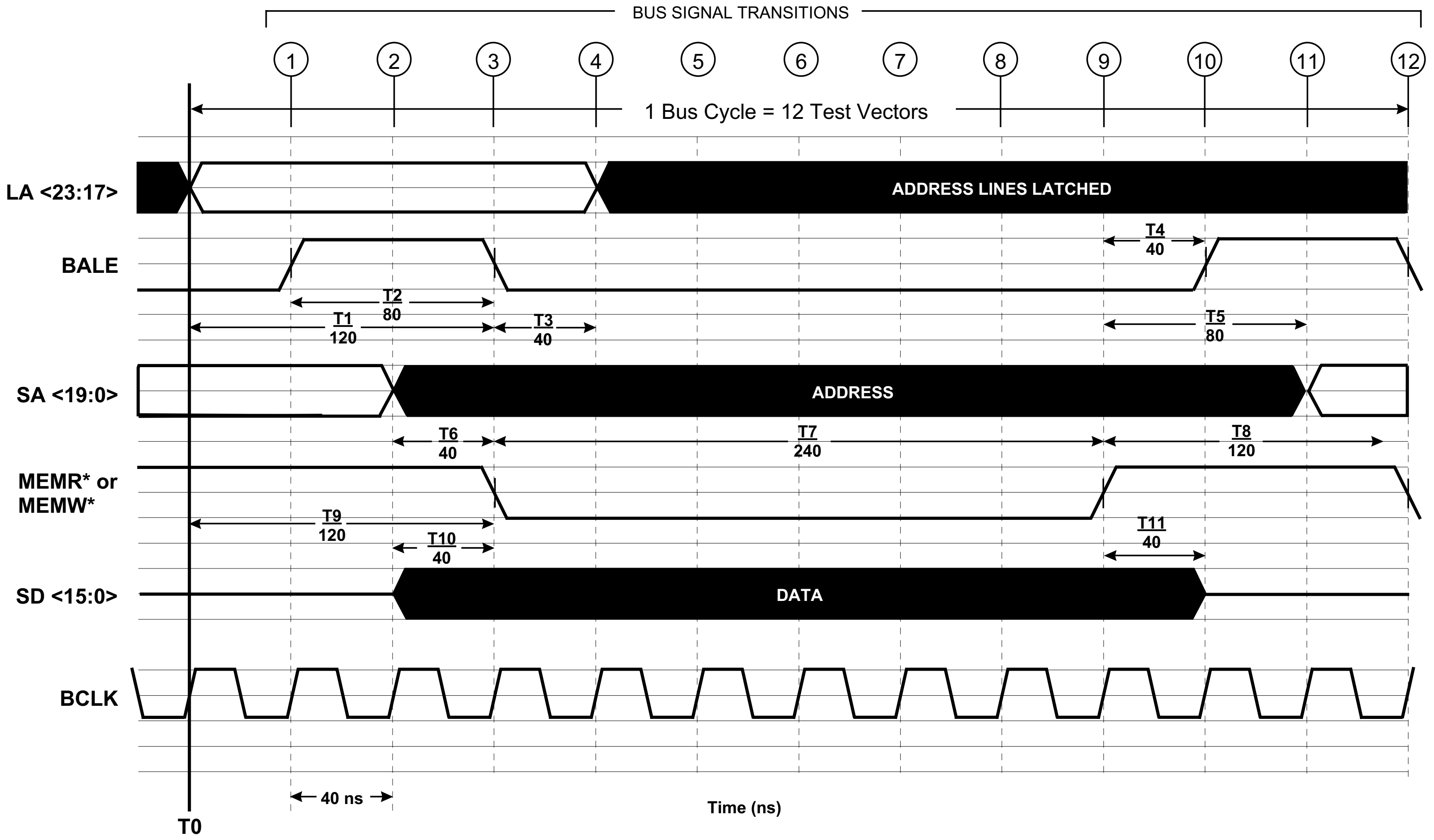


Figure 2.
Bus Emulation Timing Diagram for ISA Bus Read or Write Cycle Showing
Degradation Caused by Using Test System Without Data Formatting or Programmable
Edge Placement Capability (25 MHz Clock).

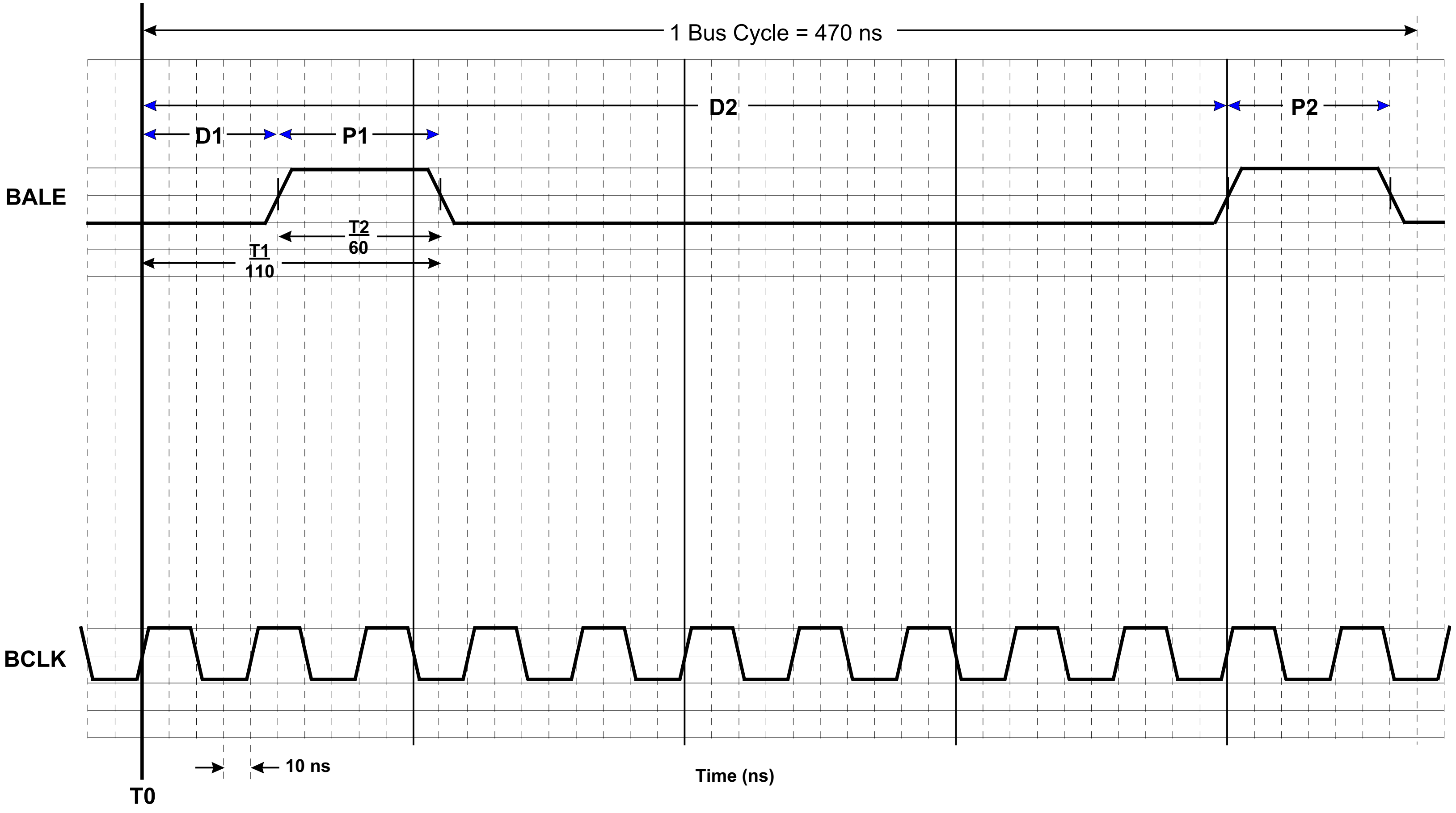


Figure 3.
By Using Programmable Edge Placement and Data Formats,
a Bus Cycle Can be Simulated Using Just One Test Vector.

App/Tech Note

Binary Pattern Transfer SR2500-02

Introduction

The SR2500 can handle both mapped and unmapped binary pattern transfers. Programming commands for each type of transfer are as follows:

Mapped Binary Pattern Transfer

Commands

The commands for the mapped load form of the transfer is:

STIM:VEC <x>;COUNT <y>;DATA:FIEL <name>;BLOC:TYPE MAP;PATT

or

REC:VEC <x>;COUNT <y>;DATA:FIEL <name>;BLOC:TYPE MAP;PATT

where: x is the starting vector, y is the number of vectors to load and name is the fieldname.

For either type of the command, there is no data returned via the Word Serial Protocol path, the data is sent to/from the SR2510 A32 shared memory via the VXI bus. The data in the shared memory is always in 32 bit words; the number of 32 bit words is determined by the COUNT parameter. Regardless of how many bits are in the field being processed, 32 bits are sent to/from the A32 memory. There is no additional handshaking required to move the data to/from the SR2510 A32 cache to the Slot-0. The Slot-0 can determine when the SR2500 is done transferring data to/from the A32 shared memory by checking the DIR bit in the SR2510 Response register. When the DIR bit is set to ONE, the SR2500 has completed the transfer. The Slot-0 may access the A32 shared memory with D8, D16 or D32 transfers, the SR2500 will always use D32 transfers. Fields of data types OT, ED or REC are not valid.

Slot-0 Activity

Pattern Load (PATT)

Before issuing the ....;PATT SCPI command the Slot-0 must load the SR2510 A32 memory with data from a previously saved mapped type pattern transfer. To accomplish this, first locate the address of the shared memory, then use the memory move function provided with the Slot-0. On a National Instruments Embedded controller the functions provided are called, GetDevInfoLong and VXImove. The data must be loaded into the shared memory starting at the first address and continuing for COUNT 32 bit words and should be in Motorola byte order. Once the data has been loaded into the shared memory, issue the SCPI command ...;PATT. When the DIR bit is set by the SR2510, the command has been completed and other SCPI commands may be issued.

Step 1- Load data into SR2510 A32 memory

Step 2- Issue ...PATT SCPI command

Step 3- Wait for DIR

Pattern Save (PATT?)

Issue the ...;PATT? command first, then wait for the DIR bit to be set by the SR2510. When the DIR bit has been set save the data from the SR2510 A32 memory. To save the data first locate the address of the A32 memory, then use the memory move function provided with the Slot-0. On a National Instruments Embedded controller the functions provided are called, GetDevInfoLong and VXImove. The data is available in the A32 memory starting at the first address and continuing for COUNT 32 bit words and is in Motorola byte order.

Step 1- Issue ...;PATT? SCPI command

Step 2- Wait for DIR

Step 3- Save data from SR2510 A32 memory

Non-Mapped Binary Pattern Transfer

Commands

The commands for the mapped load form of the transfer are:

STIM:VEC <x>;COUNT <y>;DATA:BLOC:TYPE NOMAP;CARD <c>;MEM <m>;PATT

or

REC:VEC <x>;COUNT <y>;DATA:BLOC:TYPE NOMAP;CARD <c>;MEM <m>;PATT

where: x is the starting vector, y is the number of vectors to load, c is the card number and m is the memory type.

The commands for the mapped save form of the transfer is:

STIM:VEC <x>;COUNT <y>;DATA:BLOC:TYPE NOMAP;CARD <c>;MEM <m>;PATT?

or

REC:VEC <x>;COUNT <y>;DATA:BLOC:TYPE NOMAP;CARD <c>;MEM <m>;PATT?

where: x is the starting vector, y is the number of vectors to load, c is the card number and m is the memory type.

For either type of the command, there is no data returned via the Word Serial Protocol path, the data is sent to/from the SR2510 A32 shared memory via the VXI bus. The data in the shared memory is always in 32 bit words, the number of 32 bit words is determined by the COUNT parameter. There is no additional handshaking required to move the data to/from the SR2510 A32 cache to the Slot-0. The Slot-0 can determine when the SR2500 is done transferring data to/from the A32 shared memory by checking the DIR bit in the SR2510 Response register. When the DIR bit is set to ONE, the SR2500 has completed the transfer. The Slot-0 may access the A32 shared memory with D8, D16 or D32 transfers, the SR2500 will always use D32 transfers. Only fields of data types OUT, TRI, EXP and DON are valid.

Slot-0 Activity

Pattern Load (PATT)

Before issuing the ....;PATT SCPI command the Slot-0 must load the SR2510 A32 memory with data from a previously saved Nomap type pattern transfer. To accomplish this, first locate the address of the shared memory, then use the memory move function provided with the Slot-0. On an National Instruments Embedded controller the functions provided are called, GetDevInfoLong and VXImove. The data must be loaded into the shared memory starting at the first address and continuing for COUNT 32 bit words and should be in Motorola byte order. Once the data has been loaded into the shared memory, issue the SCPI command ...;PATT. When the DIR bit is set by the SR2510, the command has been completed and other SCPI commands may be issued.

Step 1- Load data into SR2510 A32 memory

Step 2- Issue ...;PATT SCPI command

Step 3- Wait for DIR

Pattern Save (PATT?)

Issue the ...;PATT? command first, then wait for the DIR bit to be set by the SR2510. When the DIR bit has been set save the data from the SR2510 A32 memory. To save the data first locate the address of the A32 memory, then use the memory move function provided with the Slot-0. On a National Instruments Embedded controller the functions provided are called, GetDevInfoLong and VXImove. The data is available in the A32 memory starting at the first address and continuing for COUNT 32 bit words and is in Motorola byte order.

Step 1- Issue ...;PATT? SCPI command

Step 2- Wait for DIR

Step 3- Save data from SR2510 A32 memory

App/Tech Note

Generating Pseudo-Random Bit Streams Using SR2500 Algorithmic and Sequencing Features

SR2500-03

Scope of Coverage

The following information provides an example for generating Pseudo-Random Bit Streams (PRBS) using the SR2500 Algorithmic and Sequencing features.

Algorithmic Functions Used

The SR2500 provides four algorithmic pattern generators on each 32 channel I/O module, which may be linked together to form 16, 24, and 32 bit patterns. Three algorithmic functions used in this example are:

1. Nonalgorithmic (NONA - load pattern from pattern RAM)
2. Shift-Left-Zero-Fill (SLEFTZ - shift all bits left and fill the LSB with zero)
3. Shift-Left-One-Fill (SLEFTO - shift all bits left and fill the LSB with one).

By creating a 24-bit algorithmic field that supports shifting functions, you have a 24-bit shift register of which 23 bits are used.

XOR Function

The Exclusive OR (XOR) function is achieved by monitoring bits 18 and 23 from the output. This is done by physically connecting bit 18 output to bit 18 input, and bit 23 output to bit 23 input, and then monitoring the bit patterns. By monitoring both of these bits, you can change your test sequence flow anytime, based on the combination of these two bits. The *normal* shifting sequence is SLEFTZ. However, whenever bits 18 and 23 are "01" or "10," you branch to a program segment wherein the shifting sequence is SLEFTO, thus creating an XOR function.

Pipeline Latency and Branch Delays

One important consideration that must be taken into account is the SR2500 pipeline latency and branch delays. For each vector, the state generated for all pins must be clocked through the SR2500 output pipeline. This pipeline is three clock cycles plus 60 ns. Also, for the state on the input pins to be evaluated and acted on, the input data must be clocked through the input pipeline. This pipeline is also three clock cycles plus 60 ns. Hence, the round trip takes six clock cycles plus 120 ns. Running at 25 MHz (40 ns period), this translates to a round trip delay of 9 clock cycles (6 cycles of 40 ns plus 120 ns). To be certain that the output bits can be detected, you must wait 9 clock cycles from the time the outputs are changed until they are tested. In addition, a branch takes four clock cycles if branching to an odd vector, and five clock cycles if branching to an even vector. In this program example, each bit in the PRBS takes 16 clock cycles to generate, so the maximum bit rate is (25 MHz / 16), or (40 ns x 16). The table below provides the basic test sequence and pattern generation concepts used in this example.

Vector	JMP/JSR LABEL	CMACRO COMMAND	LOOP/BRANCH CONDITION	ALGORITHMIC PATTERN
0001	---	StartProgram	---	NONA - #h000000
0002	---	WordLoopuntil	COUNt == 10	NONA - #h000000
0003	---	SetCONDition	QUALifier && #b00000011	NONA - #h000000
0004	---	CLEARError	---	NONA - #h000001
0005	---	StartLoopuntil	SystemTRIgger == TRUE	HOLDA
0006	---	WordLoopuntil	COUNt == 10	HOLDA
0007	---	ConditionalJuMP	FILL_1	HOLDA
0008	---	WordLoopuntil	COUNt == 3	HOLDA
0009	---	EndLoop	---	SLEFTZ
0010	---	EndProgram	---	NONA - #h000000
0011	---	OUTput	---	NONA - #h000000
0012	---	StartLoop	COUNt == 1	NONA - #h000000
0013	FILL_1	EndLoop	---	SLEFTO
0014	---	EndProgram	---	NONA - #h000000
0015	---	---	---	NONA - #h000000

A few points regarding this program ... the 10 cycle delay at vector six is to provide time for the output data to work its way through both the output and the input pipelines. The three cycle delay at vector eight is to compensate for the four clock cycles it takes to execute the conditional jump at vector seven. If the jump is not taken, then the single clock cycle of vector seven, plus the three clock cycles at vector eight, provide the same time delay as when the jump path is taken.

Another shortcut you can use in this example concerns looping. To branch from the end of a loop to the beginning takes only a single clock cycle, in contrast to the four or five that a branch requires. However, each StartLoop allows only a single EndLoop. By placing a StartLoop outside the program flow (vector 12), but placing its associated EndLoop within the program flow (vector 13), you wind up with two end loops for the start loop command at vector five. This allows faster execution of the PRBS sequence.

One last note, as previously mentioned, it takes 16 clock cycles to generate a single PRBS output. By changing the COUNt value at vector six, slower bit rates are realized. For example, assume you need to generate PRBS patterns at 100 kHz rates. There are 250 40 ns periods in a single 100 kHz clock cycle. Since the normal sequence through the PRBS requires 16 40 ns cycles, by adding an additional 234 cycle delay to the sequence, a 100 kHz PRBS pattern is generated. Simply add 234 to the 10 cycle delay at vector six, for a total delay of 244 (plus one for the StartLoop at vector five, plus 1 for the EndLoop at vector nine or 13, plus four for the branch or the compensated "nonbranch"). It now requires 250 cycles to generate a single PRBS output.

To run the attached program, you will need to use a loop-back cable to connect bit 18 and 23 outputs to inputs. The program example will mask out any other bits, so you can simplify the cable by looping all 32 channels on the I/O module. The serial output can be tapped off from bit 23's output, or from the unused bit 24, which is bit 23 delayed by one cycle.

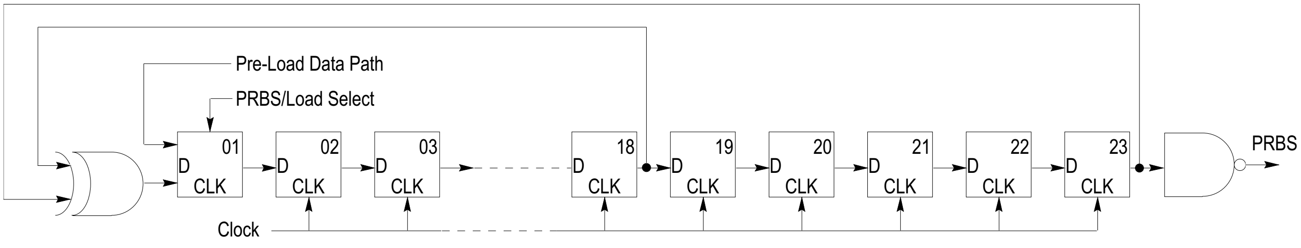
Since the SR2500 is running at 25 MHz, yet the output rate is only 1/16th that speed, you have created some extra fields to qualify when to save sampled data. As programmed, the SR2500 will record all 24 PRBS input bits whenever the sample output bit is set. The sample output bit is set only on the LSEFTZ or SLEFTO algorithmic vectors. The result is a single record sample for each unique PRBS pattern. This program is good for generating serial PRBS patterns as well as parallel PRBS patterns.

Shift Register Creates CCITT Standard PRBS

Several standard PRBSs are in use for multiplexer testing and Consultative Committee on International Telegraphy and Telephony (CCITT) Standard No. 0.151 gives a common example. The figure below shows the block diagram of a circuit that outputs the CCITT sequence.

A 23-stage shift register forms the basis for the circuit. An exclusive OR gate with inuts connected to the outputs of flip-flop 18 and flip-flop 23 generates the input of the first D-type flip-flop. The shift register outputs the PRBS automatically after preloading any bit pattern except 000 0000 0000 0000 0000 0000. The bit sequence produced by this algorithm consists of all different 23-bit-long words except all zeros. Thus, the circuit outputs 223-1 = 8,388,607 different words before the sequence repeats itself.

The 23 flip-flop shift register is the standard most commonly used, although other standards consisting of 7, 10, 11, or 15 flip-flops are in use. Naturally, the higher the number of flip-flops, the more different words are generated and, thus, the more "random" the overall bit sequence. ... *(Test & Measurement Europe/December-January 1996).*



A 23-Stage Shift Register Outputs the PRBS Most Commonly Used.

SR2500 23-Bit PRBS Program Example:

```
TEST:DEF PRBs_23:SIZE 65500
SOUR INTERNAL
SYST:PROG 1
SYST:FREQ 2500000
STYT:CLOC:SOUR INTERNAL
SYST:CLOC:SLOP POS
SYST:CLOC:LEV 1.200000E+0
SYST:GATE:SOUR INT
SYST:GATE:POL NORM
SYST:GATE:LEV 1.200000E+0
TRIG:SOUR BUS
FIEL:DEF MSB_R:TYPE RECORD:PIN C1P23
FIELD:NAME MSB_R:RAD BIN
FIEL:DEF PSRB_0:TYPE ALGOUTPUT:PIN C1P24-1
FIELD:NAME PSRB_O:RAD HEX
STIM:COND:OFOR:FIEL PSRB_0;MODE NRZ, 0.000000E+0
STIM:VEC 1;COUN 20;DATA:FIEL PSRB_O;PATT #h000000, #h000000, #h000000,
#h000001, #h000000, #h000000, #H2C2C2C, #h000000, #h000000, #h000001,
#h000000, #h333333, #h000000, #h000000, #h000000, #h000000, #h000000,
#h000000, #h000000, #h000000
STIM:VEC 1;COUN 20;AMAC:FIEL PSRB_O;PATT NONA, NONA, NONA, NONA, HOLDA, HOLDA,
HOLDA, HOLDA, SLEFTZ, NONA, NONA, NONA, SLEFTO, NONA, NONA, NONA, NONA, NONA,
NONA, NONA
FIEL:DEF PSRB_R:TYPE RECORD:PIN C1P24-1
FIELD:NAME PSRB_R:RAD HEX
FIEL:DEF PSRB_R:TYPE TRISTATE:PIN C1P24-1
FIELD:NAME PSRB_T:RAD HEX
STIM:VEC 1;COUN 20;DATA:FIEL PSRB_T;PATT #h000000, #h000000, #h000000,
#h000001, #h000000, #h000000, #h000000, #h000000, #h000000, #h000001,
#h000000, #h333333, #h000000, #h000000, #h000000, #h000000, #h000000,
#h000000, #h000000, #h000000
FIEL:DEF SAMPLE_O:TYPE OT:PIN C1P26-25
FIELD:NAME SAMPLE_O:RAD BIN
STIM:COND:OFOR:FIEL SAMPLE_O;MODE NRZ, 0.000000E+0
STIM:VEC 1;COUN 20;DATA:FIEL SAMPLE_0;PATT #b00, #b00, #b00, #b00, #b00,
#b00, #b00, #b00, #b01, #b00, #b00, #b00, #b01, #b00, #b00, #b00, #b00,
#b00, #b00, #b00
FIEL:DEF SAMPLE_R:TYPE RECORD:PIN C1P26-25
FIELD:NAME SAMPLE_R:RAD BIN
FIEL:DEF XOR_R:TYPE RECORD:PIN C1P23,C1P18
FIELD:NAME XOR_R:RAD BIN
FIEL:DEF PSRB_E:TYPE ED:PIN C1P24-1
FIELD:NAME PSRB_E:RAD HEX
REC:COND:SAMP:FIEL PSRB_E;MODE EDGE, 1.500000E-8
REC:TRAC:QUAL 1:FIEL PSRB_E;PATT #bX0XXXX1XXXXXXXXXXXXXXXXX
REC:TRAC:QUAL 2:FIEL PSRB_E;PATT #bX1XXXX0XXXXXXXXXXXXXXXXX
REC:TRAC:QUAL 3:FIEL PSRB_E;PATT #hXXXXXX
REC:TRAC:QUAL 4:FIEL PSRB_E;PATT #hXXXXXX
```

```
REC:TRAC:QUAL 6:FIEL PSRB_E;PATT #hXXXXXX
REC:TRAC:QUAL 7:FIEL PSRB_E;PATT #hXXXXXX
REC:TRAC:QUAL 8:FIEL PSRB_E;PATT #hXXXXXX
REC:VEC 1;COUN 20;DATA:FIEL PSRB_E;PATT #hXXXXXX, #hXXXXXX, #hXXXXXX,
#hXXXXXX, #hXXXXXX, #hXXXXXX, #hXXXXXX, #hXXXXXX, #hXXXXXX, #hXXXXXX,
#hXXXXXX, #hXXXXXX, #hXXXXXX, #hXXXXXX, #hXXXXXX, #hXXXXXX, #hXXXXXX,
#hXXXXXX, #hXXXXXX, #hXXXXXX
FIEL:DEF SAMPLE_E:TYPE ED:PIN C1P26-25
FIELD:NAME SAMPLE_E:RAD BIN
REC:COND:SAMP:FIEL SAMPLE_E;MODE EDGE, 1.500000E-8
REC:TRAC:QUAL 1:FIEL SAMPLE_E;PATT #bXX
REC:TRAC:QUAL 2:FIEL SAMPLE_E;PATT #bXX
REC:TRAC:QUAL 3:FIEL SAMPLE_E;PATT #bX1
REC:TRAC:QUAL 4:FIEL SAMPLE_E;PATT #bXX
REC:TRAC:QUAL 5:FIEL SAMPLE_E;PATT #bXX
REC:TRAC:QUAL 6:FIEL SAMPLE_E;PATT #bXX
REC:TRAC:QUAL 7:FIEL SAMPLE_E;PATT #bXX
REC:TRAC:QUAL 8:FIEL SAMPLE_E;PATT #bXX
RED:VEC 1;COUN 20;DATA:FIEL SAMPLE_E;PATT #b0X, #b0X, #b0X, #b0X, #b0X,
#b0X, #b0X, #b0X, #b00, #bXX, #b0X, #b0X, #b00, #b0X, #b0X, #b0X, #b0X,
#b0X, #b0X, #b0X
STIM:VEC 13;COUN 1;CMACRO:DEF ((LAB FILL_1)OUT (OUT)
STIM:VEC 1;COUN 20;CMACRO:DEF (SP (OUT)), (WL (OUT(COUN == 10))), SCOND(OUT(QUAL
&& #b00000011))), (CLEARE (OUT)), (SL (OUT(STRI == TRUE))),
(WL (OUT(COUN== 10))), (CJMP (OUT(FILL_1))), (WL (OUT(COUN == 3))), (EL
(OUT)), (EP (OUT)), (OUT(OUT)), (SL (OUT(COUN == 1))), (EL (OUT)), (EP (OUT)),
(OUT(OUT)), (OUT(OUT)), (OUT(OUT)), (OUT(OUT)), (OUT(OUT)), (OUT(OUT))
REC:TRAC:SEQ 1:DEF:FILT DATA:REC QCOM2
REC:TRAC:SEQ 1:DEF:CRC:CALC NEV
REC:TRAC:SEQ 1:DEF:ADVS:ON NEV:COUN 1
REC:TRAC:SEQ 1:DEF:JUMP 1:ON NEV
REC:TRAC:QCOM1 1, 2
REC:TRAC:QCOM2 3
```

(This Page Intentionally Left Blank)

App/Tech Note

SR2500 Binary Test Load SR2500-04

Scope of Coverage

This AppNote describes the process for learning tests from a properly configured SR2500 subsystem using the Learn Query (LEARN?) command, and also describes the process of transferring the learned data back to the SR2500 subsystem using the Learn (LEARN) command to configure the instrument as before. The procedure presented here assumes the reader is familiar with SR2500 programming using the SCPI message based commands and further assumes that a valid configuration already exists within the SR2500 subsystem prior to learning that configuration.

Note

This AppNote does not provide procedures for creating a binary file from scratch for loading to the SR2500 subsystem using the LEARN command.

Since each Slot-0 controller provides different function calls for accessing A16 and A32 address locations and communicating with VXIbus instruments, this AppNote does not attempt to address programming specifics. Rather, it provides a conceptual procedure in the form of a flowchart with a verbal description of each step. It is assumed that the reader has a basic familiarity of the VXIbus and knowledge of the programming environment of their system.

General

The SR2500 binary LEARN? and LEARN commands make use of 1 MB of memory located on the SR2510 Timing/Control Module. This memory block (cache) is mapped to the VXIbus A32 memory space and may be accessed using D8, D16, or D32 data transfers. There are a few address locations within this A32 memory that are reserved for special functions associated with the LEARN? and LEARN commands. Refer to the following memory map for specific information. Note that the memory map is configured as 16-bit wide data words.

Definitions

Handshake: This word is used to indicate that a valid command ward has been written to A32 base offset +2. It is used by both the Slot-0 controller and by the SR2510.

0xFFF == Valid Command Word, Cache Full
0x000 == Transfer Executed, Cache Empty

Command Word: Indicates if the cache contains the last block of data to transfer or if more data blocks remain. The command word is also used to acknowledge the last block transfer:

0xAC00 == Acknowledge Slot-0 Receipt of Data Block
0x0EAD == More Data Blocks Remain to be Transferred
0xCEAD == Last Data Block to Transfer

Buffer Size: The cache buffer size is stored at a 32-bit Long Word (LWORD) at A32 base offset +4. This indicates the number of BYTES resident in the A32 cache memory. Transfer of data between A32 cache memory and local CPU RAM may be done in one transfer, or may require many transfers, depending on the amount of data being transferred and the size of the buffer allocated in the host.

Cache Start: Data is stored in the A32 cache memory starting at A32 offset +256.

Cache End: Physical end of the A32 cache memory. Note that data stored in the cache does not necessarily occupy the entire cache space.

SR2510 A32 Memory Map

SR2510 A32 Memory Map

A32	Address Description
Base Offset	(16 ------------- Data Bits ------------- 0)
+0	Handshake Word
+2	Command Word
+4	Buffer Size -- Lword (1st 16-bits)
+6	Buffer Size -- Lword (2nd 16-bits)
+8	
o	o
o	o
o	o
+254	
+256	Cache Start (1st 2-bytes)
+258	2nd 2-bytes of data
o	o
o	o
o	o
+1048576	End of Cache

A32 Memory Offset

The address location of an instrument's A32 memory is determined by the Resource Manager (RM) of the VXIbus system. Until the RM assigns an A32 offset for that module, the A32 memory cannot be accessed. Since the user does not predetermine the A32 address for the SR2510 module, any program that makes use of the SR2510's A32 memory must include a routine for determining the A32 base offset value assigned by the RM. The most basic process for determining the A32 base offset is to read the instrument's OFFSET register. After successful completion of the RM program, each instrument that requested A32 address space will have an A32 base address value written to it's OFFSET register by the RM. The OFFSET register is a 16-bit register located at the devices A16 base address plus an offset of six. Knowing the SR2510's logical address means that the OFFSET register may be accessed directly by a program. The formula for determining the absolute address of a modules OFFSET register, based on the modules Logical Address (LA) is as follows:

OFFSET Register Address = ((LOGICAL ADDRESS x 64) + 49152) +6

Therefore, an SR2510 with a LA of 7 would have an OFFSET register address of:

((7 x 64) + 49152) + 6) = 448 + 49152 + 6 = 49606 (0xC1C6 hex)

The value read from address 49606 would be the modules A32 base address.

Another method of acquiring a modules A32 base address is to use function calls within the Slot-0 controller. Some Slot-0 controllers build a table of instrument parameters during the RM process, and keep that information in memory for access by the user, or an application program. This table may be accessed via function calls in order to provide information about a module, such as it's A32 base address. Again, each manufacturer provides their own Slot-o functions, so it is necessary to read the manufacturer's manual to determine which functions are supported.

Flow Charts

The flow charts illustrated in Figures 1 and 2 represent the process that must be used to learn a test configuration from the SR2500 subsystem (LEARN?), and the process for the SR2500 subsystem to learn the previous save configuration back from the host (LEARN), respectively.

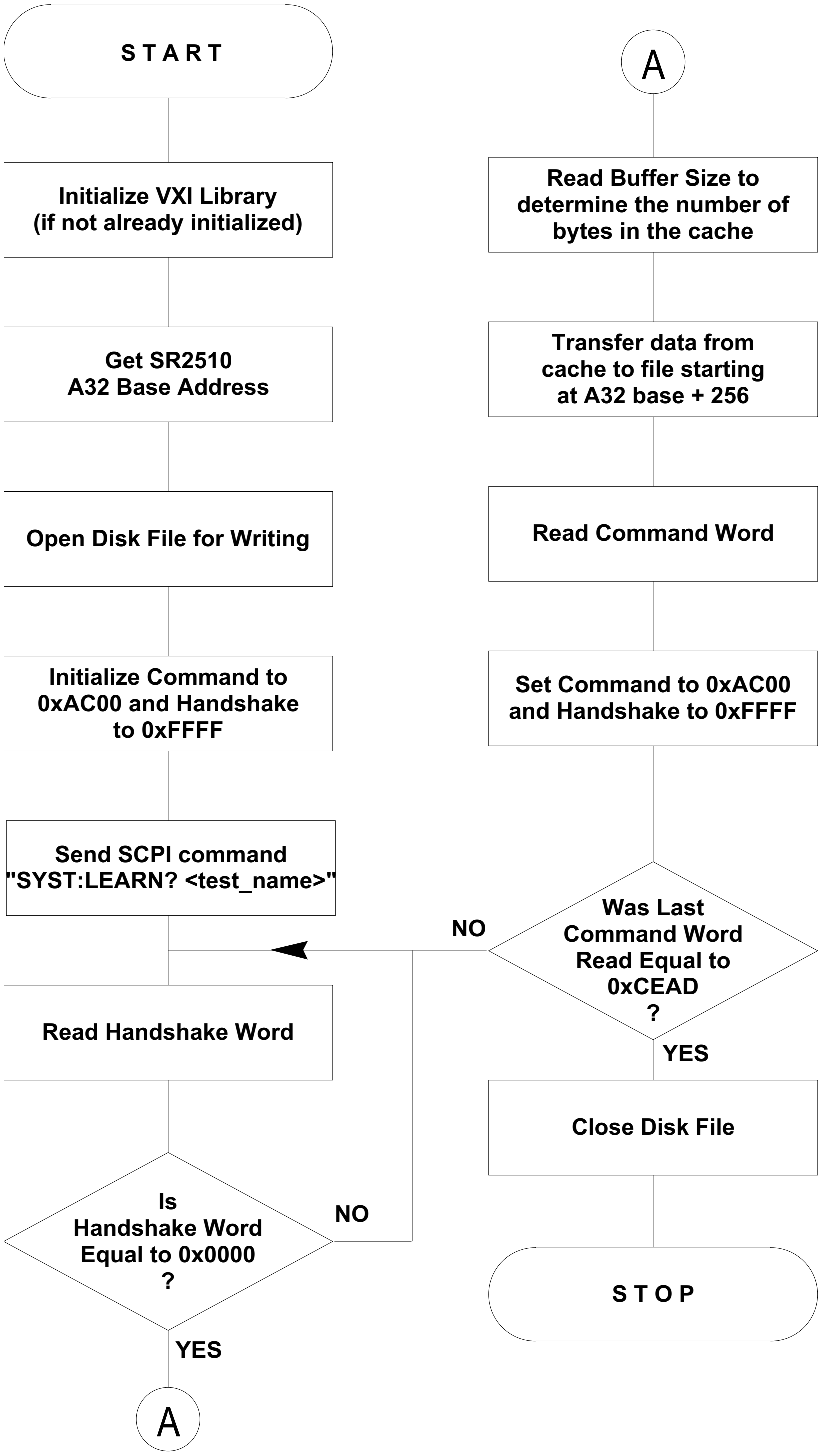

START
Initialize VXI Library (if not already initialized)
Get SR2510 A32 Base Address
Open Disk File for Writing
Initialize Command to 0xAC00 and Handshake to 0xFFFF
Send SCPI command "SYST:LEARN? <test_name>"
Read Handshake Word
Is Handshake Word Equal to 0x0000 ?
NO
YES
A
A
Read Buffer Size to determine the number of bytes in the cache
Transfer data from cache to file starting at A32 base + 256
Read Command Word
Set Command to 0xAC00 and Handshake to 0xFFFF
Was Last Command Word Read Equal to 0xCEAD ?
NO
YES
Close Disk File
STOP

Figure 1.
Learn Query Flowchart.

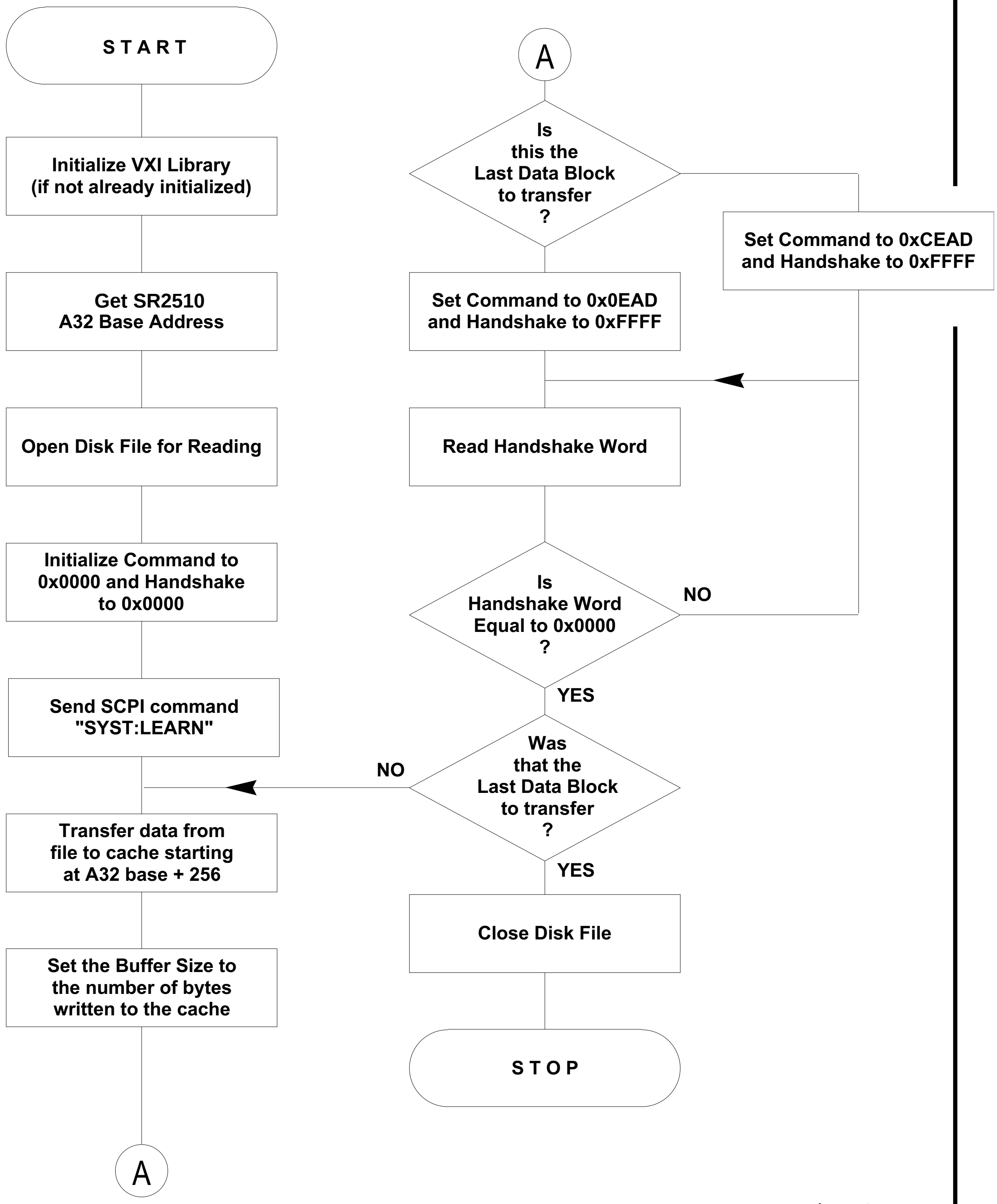

START
Initialize VXI Library (if not already initialized)
Get SR2510 A32 Base Address
Open Disk File for Reading
Initialize Command to 0x0000 and Handshake to 0x0000
Send SCPI command "SYST:LEARN"
Transfer data from file to cache starting at A32 base + 256
Set the Buffer Size to the number of bytes written to the cache
A
A
Is this the Last Data Block to transfer ?
Set Command to 0xCEAD and Handshake to 0xFFFF
Set Command to 0x0EAD and Handshake to 0xFFFF
Read Handshake Word
Is Handshake Word Equal to 0x0000 ?
NO
YES
Was that the Last Data Block to transfer ?
NO
YES
Close Disk File
STOP

Figure 2.
Learn Command Flowchart.

(This Page Intentionally Left Blank)

App/Tech Note

Data Formatting and Edge Timing SR2500-05

Understanding Data Rate

The SR2500 can generate patterns at speeds up to 25 MHz. The term "Data Rate" itself can be confusing, since manufactures often apply their own meaning to the term. Generally, Data Rate is the speed at which data patterns can be generated, and is a function of the clock period. A clock period of 40 ns (25 MHz) means the state for each data channel in the pattern sequence is held for 40 ns before advancing to the next state in the sequence. However, to generate a clock using one of these data channels requires two states ... one high and one low. So the effective rate of a clock generated using a data channel, per the above 25 MHz example, would be 80 ns or 12.5 MHz.

The SR2500 employs data formatting, which allows true 25 MHz data and clock rates, or 50 MHz data rates and 25 MHz clock rates, depending on which definition of data rate you choose to use. Combined with data formatting is programmable edge timing, which allows the user to program channel delay, pulse width, and channel-to-channel skew.

Data Formatting

Data formatting is the ability to apply a format pattern to the data channels for each test vector state, or cycle. The SR2500 supports the following data formats, see Fig 1:

o	NRZ	Non-Return-to-Zero
o	DNRZ	Delayed Non-Return-to-Zero
o	RZ	Return-to-Zero
o	RONE	Return to One
o	RC	Return-to-Complement
o	RI	Return-to-Inhibit (return-to-tristate)

Figure 1 shows test cycles (vector 1, vector 2) and a single data channel with no formatting applied (NRZ) and various data formats applied. The defined output state for the data channel consists of a logic-1 (high) for vector 1 and a logic-0 (low) for vector 2.

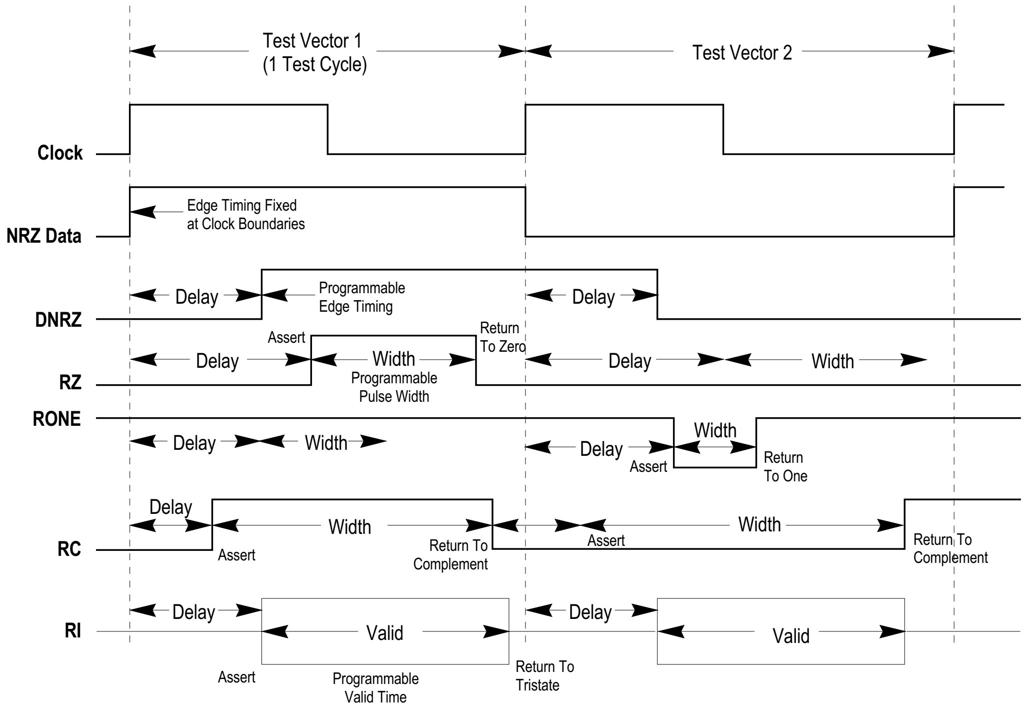


Figure 1.
Data Formats and Edge Timing.

NRZ: NRZ simply means that no formatting, nor edge timing, is applied to the defined output state. The defined output state of the channel will change coincident with the master system clock on a cycle-by-cycle basis. Edge timing for NRZ channels is fixed at 0 ns. In other words, when the system clock starts a new period, the next state will be applied to the output pin.

DNRZ: DNRZ also does not apply any format to the output state, but it does allow a delay to be defined for the channel, referenced to the start of the system clock period. NRZ is the same as DNRZ with a delay of 0 ns. The delay defined applies to every test vector cycle. So, if the DNRZ "delay" is programmed at 40 ns, then the defined state for each test vector will be delayed by 40 ns, relative to the start of the vector cycle. This concept is valid for all data format and edge timing parameters in the SR5000. DNRZ is useful for deskewing channels at the UUT, and providing adequate setup and hold times for data and clock channels.

RZ: A data format of RZ means that during the delay time, and after the width duration, the output state is forced to zero, regardless of the state programmed for the vector. Both the "assert" edge and the "return-to-zero" edge are programmable with up to 5 ns resolution. The minimum width is 10 ns and the maximum width is the clock period -10 ns. Again, the delay and width timing is identical for all test vector cycles. Width may straddle clock boundaries. In other words, assert may be in vector 1 while return-to-zero is in vector 2. Note that in vector 2 the defined state is 0, so RZ formatting is irrelevant. RZ is useful for generating active high strobes and for generating normal polarity clocks.

RONE: RONE (R1) is the opposite of RZ and means that during the delay time, and after the width duration, the output state if forced to one, regardless of the state programmed for the vector. Both the "assert" edge and the "return-to-zero" edge are programmable with up to 5ns resolution. The minimum width is 10 ns and the maximum width is the clock period - 10 ns. Note that in vector 1 the defined state is 1, so RONE formatting is irrelevant. RONE is useful for generating active low strobes and inverted polarity clocks.

RC: A return-to-complement format means the output channel will switch to the complement of the defined state at the end of the width duration. During the delay time, the complement state from the previous vector is held. A RC format will guarantee that there will be a transition for every test cycle, regardless of the defined state. RC is useful for generating Manchester encoded data and generating clocks with phase/polarity shifts.

RI: Return-to-inhibit will force the output to a high impedance (tristate) condition during delay and after width (valid). Not to be confused with dynamic tristate control, which uses the tristate memory to control the state of the driver on a cycle-by-cycle basis, RI allows a signal ... usually a bus ... to be enabled and disabled all within a single test cycle. RI is ideal for driving bidirectional busses (e.g., a data bus), or for multiplexing different pin groups onto a common bus, see Figure 2.

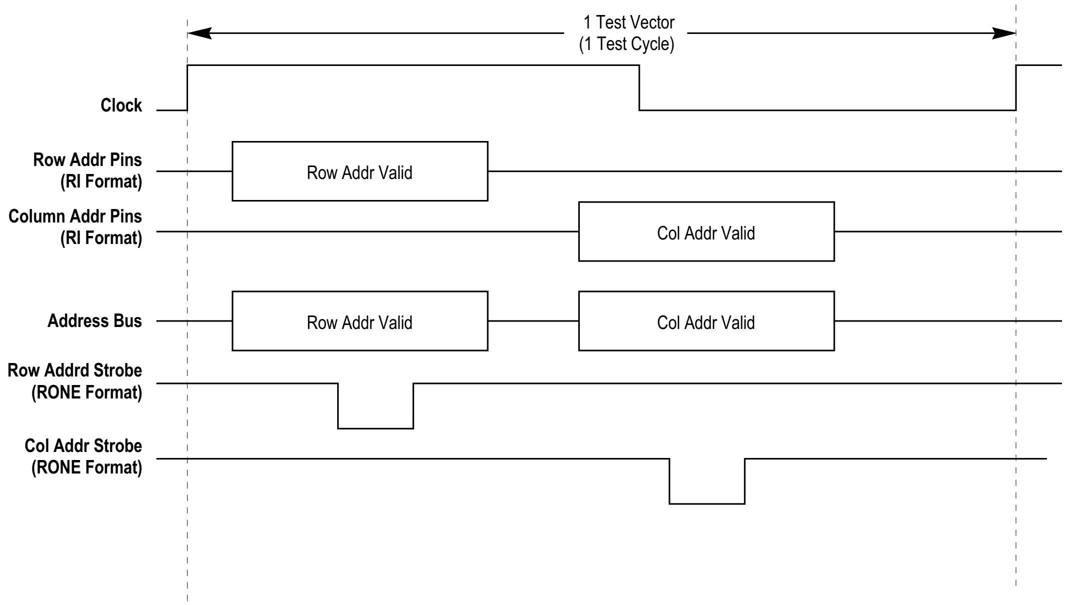


Figure 2.
Dynamic RAM Row/Column
Addressing.

(THIS PAGE INTENTIONALLY LEFT BLANK)

App/Tech Note

Emulating the 8086 Microprocessor For Board-Level Testing SR2500-06

Download this App/Note directly from Interface Technology's website, as follows:

1. Go to Interface Technology's website (www.interfacetech.com).
2. On the main page, click on "Application Notes"
3. On the Application Notes page, click on:

   SR2500-06a
   SR2500-06b

4. Download both sections of the AppNote to your hard drive, then open the self-extracting files the same way you opened this User's Manual file. Note: the password to open this AppNote is the same as that used for the SR2500 manual.

App/Tech Note

Binary Pattern Transfer Times SR2500-07

Introduction

A dynamic digital test instrument often represents the most demanding module in a VXI test system in regards to the amount of data requiring transfer on the backplane and the need for very high bus bandwidth. This is due to the large pattern depth, the wide channel count and the common practice of using multiple banks of memory behind each I/O pin, and the need to move that data quickly. Take for example a system containing 128 bi-directional channels with an output memory, tristate memory, expected pattern reference memory, input mask memory and a record memory behind each pin. If the pattern depth for each pin is 64K deep, this translates to 5MB of data to load all pattern memories.

While the VXI bus is rated for 10MB/S (D8) to 40MB/S (D32) data transfer rates, or even 80MB/S using D64 as defined in the VXI 2.0 specification, slot 0 controllers and instruments often fall far short of these ideals. And when message-based parsing is added into the equation, the transfer rate suffers even greater.

Objective

The challenge for digital test instruments, and other VXI instruments with similar data transfer requirements, is to move the data between components within the system as quickly as possible. Nowhere is this more important than in conditions where test time directly affects profitability.

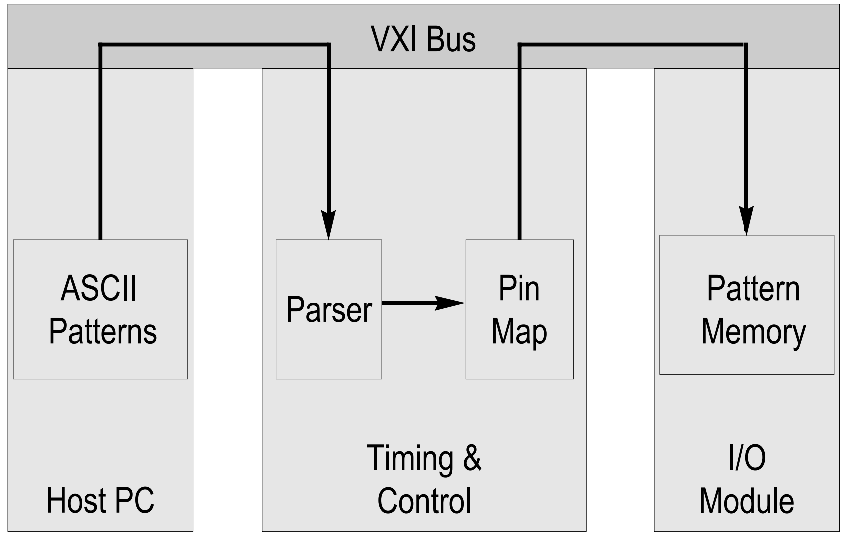
This document will explore the data transfer option available to the SR5000, SR5500 and SR2500 digital test instruments.

Features

While the features discussed in this document apply equally to the SR5000, SR5500 and SR2500, the SR5000 will be used throughout.

The VXI bus uses two primary protocols for transferring data between devices on the bus; they are Register-based and Message-based. Register-based operation mimics the operation of VME modules in that all memories and controls are mapped to an address on the bus, and are accessed via direct memory bus read and bus write operations. Register-based operation is the fastest method of getting or sending information on the VXI bus. However, it requires a detailed knowledge of the memory map of the instrument, so is much more difficult to work with.

Message-based instruments have a microprocessor on board which, among other things, parses high level ASCII text command strings to control the instrument and access memory. The text commands usually follow a convention where the commands use an English-like structure, which makes it easy to read and understand. However, those english commands must be translated into action by the instrument. Parsing the commands takes time and results in slow execution. Also, the protocol for transferring ASCII commands across the VXI bus requires multiple bus cycles to transfer 1 character. So, by its nature, message-based operation is much easier to understand, but also much slower than register-based operation.



Word Serial Transfer of ASCII Command Strings.

The SR5000 bridges the gap between these two protocols by providing message-based operation for test development and debug, and a register-based type of operation for loading finished tests and updating patterns. This document will detail the later function, high speed binary pattern loading and reading.

Presentation

While the SR5000 supports two modes of data pattern transfers, ASCII and Binary, the binary mode itself supports two versions, mapped and nomap. To understand the distinctions it is necessary to discuss the concept of fields and pinmapping.

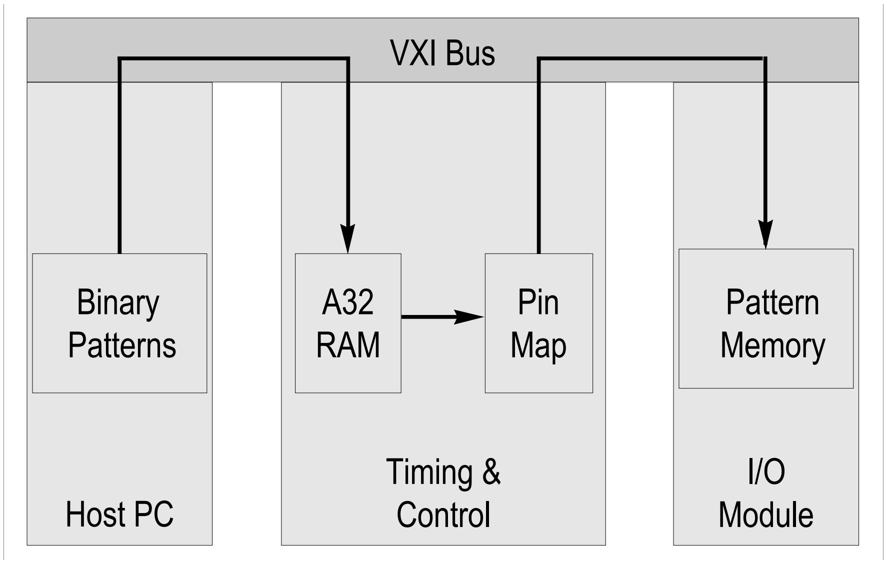
What’s in a Field

A Field is a logical grouping of channels, usually based on function. So all signals associated with an address bus may be included in one field, let’s say ADDR, while the signals associated with a data bus would be in another field, perhaps called DATA.

Pin mapping is the process of routing the patterns stored in the pattern memory to the appropriate pin on the front panel of the SR5000 I/O module. It is typical for signals in a field to be mapped to adjacent pins on a connector. However, this is not necessary. If it makes wiring a fixture easier, the signals in a field may be mapped to pins on different connectors, and even onto different I/O modules.

To Map or Not to Map

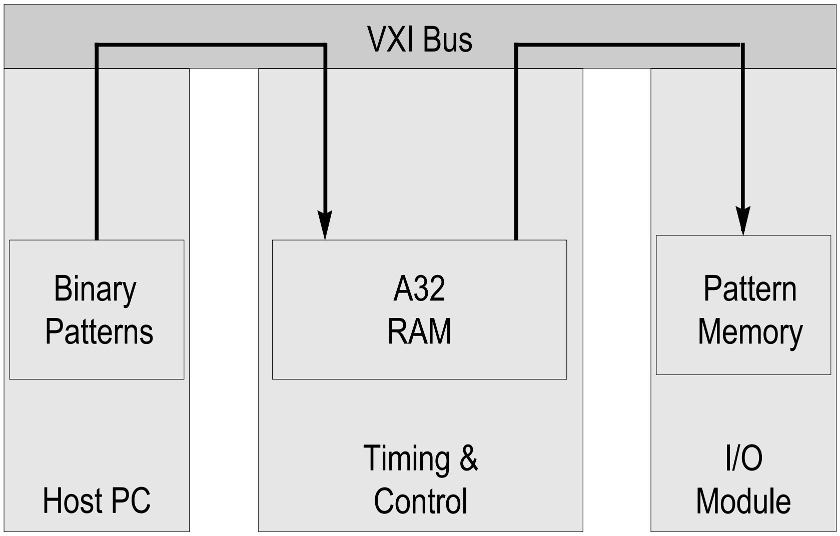
The binary Mapped data transfer uses the pin map defined for a field to route the binary patterns to the appropriate card/connector. The mapping process is controlled by the SR5000's'internal microprocessor. So, while the pattern data is transferred across the VXI bus in binary (i.e., register-based), there is overhead involved in mapping the data once the VXI transfer is complete. Overhead translates to time, so the mapped binary transfer is not the fastest method available.



Mapped Binary Pattern Transfers

The other binary data transfer method bypasses the pin mapping process, eliminating the microprocessor overhead associated with the mapping, and is called the Nomap process. When transferring data patterns using the nomap format, the pin mapping is bypassed. Instead the patterns are loaded directly to the specified pattern memory on the specified card. While the fastest method of getting data patterns into and out of memory, it has drawbacks in other areas.

First, if pins in a field are not arranged in sequential order, you must perform the pinmap shuffle in the host computer. This is often not a major hindrance as PC's are typically several orders of magnitude faster than the processor used in the SR5000. Also, the process is usually done once and saved to disk for future use, unless the data patterns are generated dynamically. In that case an algorithm would need to be developed to accommodate the mapping dynamically, which would likely still be faster than the internal algorithm. Second, the nomap process works ONLY with 32 bit wide patterns. In other words, the SR5000's internal memory structure is arranged in 32 bit wide words. The nomap transfer loads or reads all 32 bits in the designated memory (Output, Tristate, Expect, Mask or Record). Those bits may be part of the same field, different fields, or may not be used at all. Regardless, they are all accessed and transferred across the bus.



Binary Pattern Transfers, sans Pin Mapping

So, What's The Difference?

The best way to illustrate the difference between ASCII pattern loading, Binary Mapped pattern loading and Binary Nomap pattern loading is to give some examples. The table below demonstrates the time required to load a 32 bit wide pattern to one of the pattern memories. This is representative of loading the Output, Tristate, Expect or Mask memory, or reading from the Record memory. To load all of the OTEM memories on a single I/O module, you would multiply the Pattern Load Time by four.

System Configuration	Pattern Load Method	VXI Bus Transfer	Module-to-Module Transfer	Pattern Load Time
50 MHz 486 Embedded	Word Serial	N/C	N/C	01:00.00
50 MHz 486 Embedded	Binary Mapped	800 ms	00:16.90	00:17.70
50 MHz 486 Embedded	Binary NoMap	800 ms	450 ms	00:01.25
550 MHz PIII w/MXI II	Binary NoMap	28 ms	450 ms	470 ms

No attempt was made to calculate the time to transfer the ASCII characters across the VXI bus using the Word Serial method. The load time is the cumulative time for transferring the ASCII characters, parsing the command and loading the pattern memory.

Using the binary pattern load method involves a two-step process; transferring the binary data from the host PC to the A32 cache memory on the timing and control module, and then moving the data from the cache memory to the appropriate pattern memory on the I/O module. The cache on the timing and control module is mapped to A32 address space, so a high-speed block mover operation can be used to load the cache. Once the cache is loaded, a command to the timing and control module instructs that module to become the bus master, read the data from the cache – using the internal microprocessor bus – and transfer the data across the VXI bus to the slave I/O module. The process is reversed to when performing a binary read. There are two interesting facts indicated by the table.

First, the VXI bus data transfer time is the same for Mapped and NoMap operation. The overall time difference between the two methods is due to the overhead of pin mapping. Refer to the Mapped and NoMap times for the 50 MHz 486. The NoMap process is nearly 15 times faster than the Mapped method. The second interesting fact is that the speed of the host PC, and the method of connecting it to the VXI bus, can play a major role in determining overall transfer time. Refer to the 486 NoMap time and the 550 MHz PIII NoMap time. The Module transfer times are the same, while the VXI bus transfer times are very different. The overhead of the slower 486 processor, combined with the slower VXI bus speeds of the older embedded computer combine to reduce the transfer speed by over 2 and a half times. Other factors can affect overall performance, such as the operating system, how well the instrument drivers are written and the number of software layers between the user and the instrument. This illustrates that importance of considering the whole system when determining optimal performance, not just one or two components.

The PIII/MXI II values were generated using a Dell Latitude laptop with a 550 MHz PIII processor, 192MB of RAM and a PCI MXI II installed in a docking station. The system was running under the Microsoft Windows NT operating system and the instrument control was performed using National Instruments LabWindows CVI and the SR5000 VXI Plug & Play drivers distributed with the SR5000. A partial source listing is provided below.

Conclusion

For most instruments designed to operate on the VXI bus, where the volume of bus traffic is minimal, message-based Word Serial operation is adequate. But for instruments, like the SR5000, that push the bandwidth of the bus, Word Serial may become a performance bottleneck. In these situations, instruments that support direct memory reads and writes offer the highest performance. And if the instrument can blend the high level benefits of message-based operation with the high performance benefits of register-based operation, you have the best of both worlds.

Partial Source Listing:

```
      vxi_stat = InitVXIlibrary ();
      sr_stat = itsrXXX0_init (“VXI::2::INSTR”, VI_TRUE, VI_TRUE, &sr5k);
      Cls ();
//  Get the A32 offset address of the Timing Control Module
      sr_stat = GetDevInfo (2, 12, &a32_off);
      data = calloc (65536, 4);
      vxi_stat = VXImove (19, a32_off, 16, (ViUInt32)data, 65500, 4);

//  RESET the instrument
      sr_stat = itsrXXX0_reset (sr5k);

//  Load the TPS program
      sr_stat = itsrXXX0_load_scpi_cmd_file
      (sr5k,”C:\\Vxipnp\\WinNT\\Itsrxxx0\\bench.tps”);

//  Test the FILL INCREMENT function
      sr_stat = itsrXXX0_fill_data
      (sr5k, itsrXXX0_STIMULUS, “F_OUT”, 1, 65500, “#h0”, 1,itsrXXX0_FILL_INCREMENT);

//  Test the BLOCK NOMAP pattern query function
      sr_time[0] = Timer();
      for(i=0;i<100;i++)
            sr_stat = itsrXXX0_send_cmd
            (sr5k, “STIM:VECT 1;COUN ALL;DATA:BLOCK:TYPE NOMAP;CARD 1;MEMORY
            OUTPUT;PATTERN?”);
      sr_time[1] = Timer();
      p_time(i, sr_time[0], sr_time[1], “32 X 65500 NOMAP PATTERN? QUERY”);

//  Test the VXI BLOCK MOVE function
      sr_time[0] = Timer();
      for(i=0;i<1000;i++)
            sr_stat = VXImove (19, a32_off, 16, (ViUInt32)data, 65500, 4);
      sr_time[1] = Timer();
      p_time(i, sr_time[0], sr_time[1], “32 X 65500 Block Move”);

      printf(“Data Pattern Read:\n”);
      for(i=0;i<8;i++)
            printf(“%08X “,data[i]);
      printf(“...\n...”);

      for(i=65492;i<65500;i++)
            printf(“%08X “,data[i]);
      printf(“\n\n”);

//  Place other tests here....

      sr_stat = itsrXXX0_close (sr5k);
      vxi_stat = CloseVXIlibrary ();
      sr_chr = getchar ();
}

void p_time(int i,double t1, double t2, char *s)
{
      printf(“Time to execute %i %s functions:  %f\n”,i,s,t2-t1);
      printf(“Average of each iteration of %s: %f\n\n”,s,(t2-t1)/i);
      t1 = Timer();
      do
            t2 = Timer();
      while(t2 - t1 < 1);
}
```

TPS File:

```
TEST:DEF BENCH:SIZE 65500

SYST:TEST BENCH

FIELD:DEF F_OUT:TYPE OUT:PIN C1P32,C1P31,C1P30,C1P29,C1P28,C1P27,C1P26,C1P25,C1P24,C1P23,
C1P22,C1P21,C1P20,C1P19,C1P18,C1P17,C1P16,C1P15,C1P14,C1P13,C1P12,C1P11,C1P10,C1P9,C1P8,
C1P7,C1P6,C1P5,C1P4,C1P3,C1P2,C1P1
FIELD:DEF F_TRI:TYPE TRI:PIN C1P32,C1P31,C1P30,C1P29,C1P28,C1P27,C1P26,C1P25,C1P24,C1P23,
C1P22,C1P21,C1P20,C1P19,C1P18,C1P17,C1P16,C1P15,C1P14,C1P13,C1P12,C1P11,C1P10,C1P9,C1P8,
C1P7,C1P6,C1P5,C1P4,C1P3,C1P2,C1P1
FIELD:DEF F_EXP:TYPE EXP:PIN C1P32,C1P31,C1P30,C1P29,C1P28,C1P27,C1P26,C1P25,C1P24,C1P23,
C1P22,C1P21,C1P20,C1P19,C1P18,C1P17,C1P16,C1P15,C1P14,C1P13,C1P12,C1P11,C1P10,C1P9,C1P8,
C1P7,C1P6,C1P5,C1P4,C1P3,C1P2,C1P1
FIELD:DEF F_MSK:TYPE DON:PIN C1P32,C1P31,C1P30,C1P29,C1P28,C1P27,C1P26,C1P25,C1P24,C1P23,
C1P22,C1P21,C1P20,C1P19,C1P18,C1P17,C1P16,C1P15,C1P14,C1P13,C1P12,C1P11,C1P10,C1P9,C1P8,
C1P7,C1P6,C1P5,C1P4,C1P3,C1P2,C1P1
FIELD:DEF F_REC:TYPE REC:PIN C1P32,C1P31,C1P30,C1P29,C1P28,C1P27,C1P26,C1P25,C1P24,C1P23,
C1P22,C1P21,C1P20,C1P19,C1P18,C1P17,C1P16,C1P15,C1P14,C1P13,C1P12,C1P11,C1P10,C1P9,C1P8,
C1P7,C1P6,C1P5,C1P4,C1P3,C1P2,C1P1
```

App/Tech Note

Serial EEPROM Test With I2C Bus Emulation SR2500-08

As with most technological advances, the need for increased functionality requires more memory. Preferably, memory that is less expensive and in a smaller package. EEproms help provide the solution to such a demand. Along with the increased functionality for memories comes the need for a bus that will simply communicate with these memories. The I2C (Inter-integrated circuit) which was originally introduced by Phillips for communication between IC's in consumer electronic devices, is the industry leader in serial Eeprom bus protocol. Engineers in various markets including, consumer, automotive, telecom and industrial markets are very familiar with the I2C bus. This document sets out to describe a simplified customer application of how I2C protocol can be emulated with the proper digital test equipment to test the ever-changing market of EEproms.

Serial EEproms and I2C operation

For serial EEproms there are generally two types of bus communications 2-wire or 3-wire. A 2-wire product is utilized in applications that require an I2C bus, noise immunity, or have limited microcontroller I/O pins available. A 3-wire product is utilized in applications that have higher frequency rates than the 2-wire approach or limited protocol requirements.

For the purpose of this document we will discuss the 2-wire approach. I2C protocol is typically the industry leader of communication with 2-wire EEproms.

The two-wire bus is simple in that only the SDA (serial Data) and SCL (serial clock) pins are necessary for bus operation while all other pins are supplementary. The I2C protocol utilizes bi-directional communication between a master and a slave. I2C protocol is defined so that a device that sends data onto the bus is a transmitter and the device that receives the data is a receiver. The bus must be controlled by a master device, which for the purposes of this document, will be Interface Technology's digital stimulus and response subsystem, the SR2500. The master generates the serial clock (SCL), controls the bus direction, and controls the START and STOP conditions for bus communication.

I2C Bus Emulation Requirements?

Although most I2C bus communication is performed with the use of a microcontroller as the master, it is possible, and likely more feasible in a test environment, to use a digital subsystem to emulate the microcontroller. As stated above, the microcontroller has the ability to generate the serial clock, control the bus and data flow (bi-directional), transmit and receive data, and generate START and STOP conditions. Your digital test subsystem should also have the flexibility to provide these functions.

The SR2500 has the capability to generate lengthy test data with data formatting such as NRZ (non-return to zero), RZ (return to zero), RO (return to one), RTC (return to complement), and RTI (return to inhibit) on a per pin basis. Along with multiple timing sets, and looping or branching capability, the SR2500 has the ability to supply continuous clocks or data for the most complex applications. Bi-directional data flow requires the ability to tri-state I/O pins and allow for a master to transmit or receive data. The SR2500 allows per-pin and per vector tri-state control. The ability to provide precise edge placement delays allows that SR2500 to generate START and STOP conditions.

Application

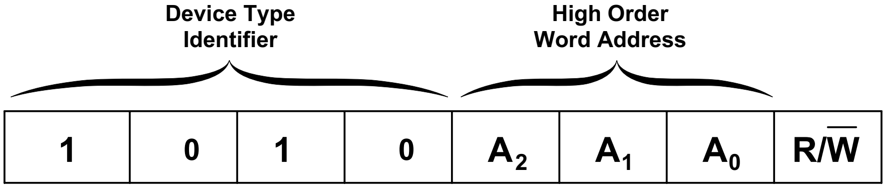
All of the above features of the SR2500 make it possible to emulate a microcontroller to efficiently test an EEprom. Lets look at a simple example of how this could possibly be implemented.

Let’s first expand upon the basic principles of 2-wire serial operation. The common device nomenclature is 24xxx or 85xxxx. For the purpose of this document we well use a X24C16 (2048 x 8 bit) serial EEprom to test.

As stated above, the EEprom communicates using 2-wire I2C protocol. 2-wire I2C protocol utilizes master/slave bi-directional communication. Only the SCL and SDA communication are essential for full Read/Write operation. The SCL input is used to clock all data into an out of the chip. SDA is a bi-directional pin that is used to transfer all data into or out of the chip.

A START condition occurs when SDA transitions from low to high while SCL is high. A STOP condition occurs when SDA transitions from high to low while SCL is high. Both conditions need to observe the proper setup and hold times required by the X24C16 chip. Data is recognized as valid on SDA while SCL is high.

After a start bit, each cycle begins with an eight bit control byte that is to be sent by the master (SR2500). The control byte, or slave address, contains three primary functions, the device identifier, the bank select bits, and the read/write bit. (See figure 1)



Device Type Identifier				High Order Word Address			
1	0	1	0	A_2	A_1	A_0	$R/\overline{W}$

Figure 1.

The most significant four bits of the slave address are the device type identifier, and for the X24C16, this is fixed at 1010[B].

The next three bits of the slave address field are the bank select bits. Toggling of these bits provide access to the eight 256 x 8 banks of memory on the X24C16, where these bits are an extension of the array's address.

The LSB of the control byte is the Read/Write bit, where depending upon the state of the Read/Write bit the X24C16 will perform a read or write operation.

Lets take a look at what a typical example of a byte write cycle would look like.

As stated previously all commands are preceded by a start condition, where a start condition is a high to low transition while SCL is high. We will use the SR2500 to generate the clock (SCL) and data bus (SDA). SCL is a continuous output while SDA is a bi-directional bus (single bit).

Standard SR2500 nomenclature defines a field as a grouping of pins. Since this is a 2 wire application things are simplified in that both fields are defined on single pins. The SR2500 front panel has separate inputs and outputs which can be tied together to form bidirectional channels. Each output on the SR2500 has 3 memory types and each input has 4 memory types. The output memory types consist of OUTput, TRIstate, and OT which is a composite of both the output and tristate memories. OUTput memory types contain RAM backed stimulus patterns that will be passed to the UUT (slave). The TRIstate memory enables or disables the drive function of the RAM backed stimulus patterns. The OT field type is defined so that the end users entered data pattern affects both the OUTput and TRIstate memories. Since SCL is a continuous clock it will be defined as an OT field that will continuously drive data. The input memories are EXPect, DONtcare, REC, and ED which is a composite of the EXPect and DONtcare memories. The EXPect field type stores the data used in RAM backed real time comparison. The DONtcare field types are used to mask (if desired) data that is invalid or irrelavent to the end user. RECord type fields store data or errors that are returned from the UUT. The ED field type is defined so that the end users entered data pattern affects both the

EXPect and DONtcare memories. Since SDA needs to be bidirectional (send and receive data) it will be defined on a single pin using ED and REC field types.). Each pin on the SR2500 has up to 256K of memory behind it, but this example will need less that 200.

The master needs the ability to control data flow of SDA, and in this case, it is the ability to tristate the outputs so that the X24C16 slave can take the bus to generate an acknowledge of successful data transfer. After each data transfer is complete and each acknowledge is returned, a stop condition is sent to the slave to terminate all communication. The STOP condition is a low to high transition of SDA while SCL is high. For simplicity, I have broken the complete write cycle into 10 steps (see figure 2)

Byte Write Cycle

Timing is crucial to provide proper communication between the master and slave. The SR2500 must provide the ability to transition SDA during appropriate states of SCL. Since we have defined the SR2500 system clock to run at 100KHz. Each vector state will be output for 10us. This ensures a 20us clock period for SCL. Using delayed NRZ (DNRZ) formatting, the output of SDA is programmed with a delay of 5us ensuring that SDA's output is driven to the defined state after its 5us delay time. The output pin will remain in that state until the same time in the following cycle, namely 5us and 15us into SCL's period (see figure 3).

Step 1 Generate a START condition by transitioning SDA high to low during a high state of SCL. This is done by the SR2500 by using NRZ (no delay) formatting for SCL and a Delayed NRZ for SDA to transition midway through the high width of SCL's period.

Step 2 After the START condition, communication can begin and must begin with sending the correct slave address, beginning with the correct device type identifier for the X24C16. This is a value of 1010[B].

Step 3 Next are the high order address bits that corresponds to the word address. For this simple example we are using 0000[B].

Step 4 Set the R/W bit low to command the slave to perform a write operation. This completes the setup communication of the slave address.

Step 5 The SR2500 will tristate its outputs so that the slave can send an acknowledge signifying that data transfer was successful. Note the programmed data for the SDA pins at this point is X or "tristate".

Step 6 For a write operation; the X24C16 requires a second address field. This address field is the word address, comprised of eight bits, which provides access to any of the 2048 words in the array. For this example the address is 10101010[B].

Step 7 The SR2500 will tristate its outputs so that the slave can send an acknowledge signifying that data transfer was successful.

Step 8 Upon acknowledge the X24C16 awaits the next 8 bits of data. This is the data to be stored into memory at the particular word address. For this example the data is 00001111[B].

Step 9 The SR2500 will tristate its outputs so that the slave can send an acknowledge signifying that data transfer was successful.

Step 10 Generate a STOP condition by transitioning SDA low to high during a high state of SCL. This terminates all data communication and begins an internal write cycle to the nonvolatile memory. Note - The self-timed write cycle typically takes 5ms.

At the completion of the write cycle and a short delay, a read can be done to return the data written to the specified address. A Random Read of the X24C16 will be shown. Once the STOP command is issued by the master the X24C16 will begin its self-timed write cycle, typically 5ms, and all inputs to the X24C16 are disabled. The master (SR2500) can begin acknowledge polling immediately, which involves issuing a start condition followed by the slave address for a write operation. If the X24C16 is still busy no Acknowledge signal will be returned, but if an acknowledge is returned the slave is ready for the next read or write cycle to begin.

Note

Acknowledge polling is capable with digital test systems, such as the SR2500, that have the ability to conditionally loop or branch. As with most high-speed digital test equipment, waiting for an event to conditionally occur may cause latency issues due to pipeline effects of that instrument. Those latencies have to be taken into consideration while developing test program sets. For simplification purposes, this document will assume that a delay of more than 5ms has been met before the next write/read cycle has begun. To ensure that the delay is met, a 5ms delay can be created using CMACRO control of the SR2500. The SR2500 has per vector *command macro* control. Since the SR2500 system clock is running at 10us per vector, a wordloop on a single vector 600 times, after the write cycle has been completed, will guarantee that at least 6ms has passed. At the end of that programmed delay a read cycle can be performed.

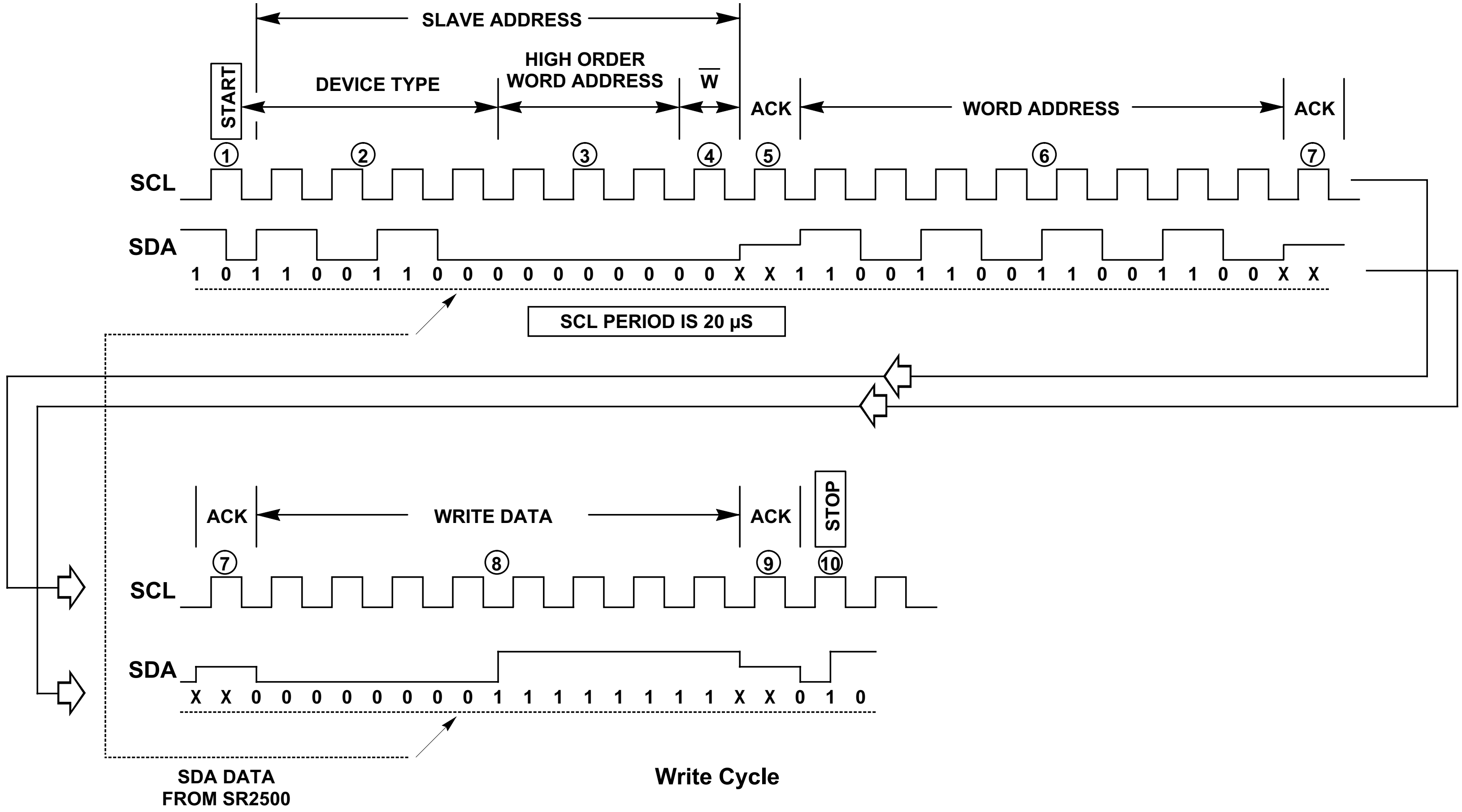

SLAVE ADDRESS
START
DEVICE TYPE
HIGH ORDER
WORD ADDRESS
W̄
ACK
WORD ADDRESS
ACK
1
2
3
4
5
6
7
SCL
SDA
1 0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 X X 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 X X
SCL PERIOD IS 20 μS
ACK
WRITE DATA
ACK
STOP
7
8
9
10
SCL
SDA
X X 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 X X 0 1 0
SDA DATA
FROM SR2500
Write Cycle

Figure 2.

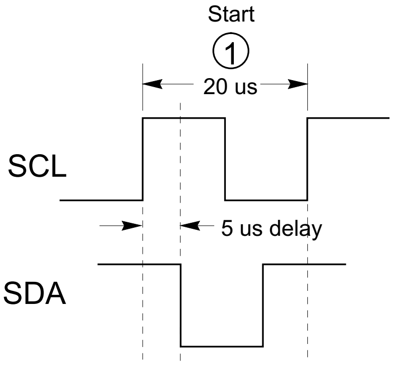


Figure 3.

Random Read Cycle

Random read operations allow the master to access any memory location in a random manner. Prior to issuing the slave address with the read/write bit set high, signifying a read operation, a “dummy write” operation must be performed (see figure 3).

Step 1 Generate a START condition by transitioning SDA high to low during a high state of SCL. This is done by the SR2500 by using NRZ (no delay) formatting for SCL and a Delayed NRZ for SDA to transition midway through the high state of SCL’s period.

Step 2 After the START condition, communication can begin and must begin with sending the correct slave address. The read/write bit of the slave address should be low to start the “dummy” write cycle.

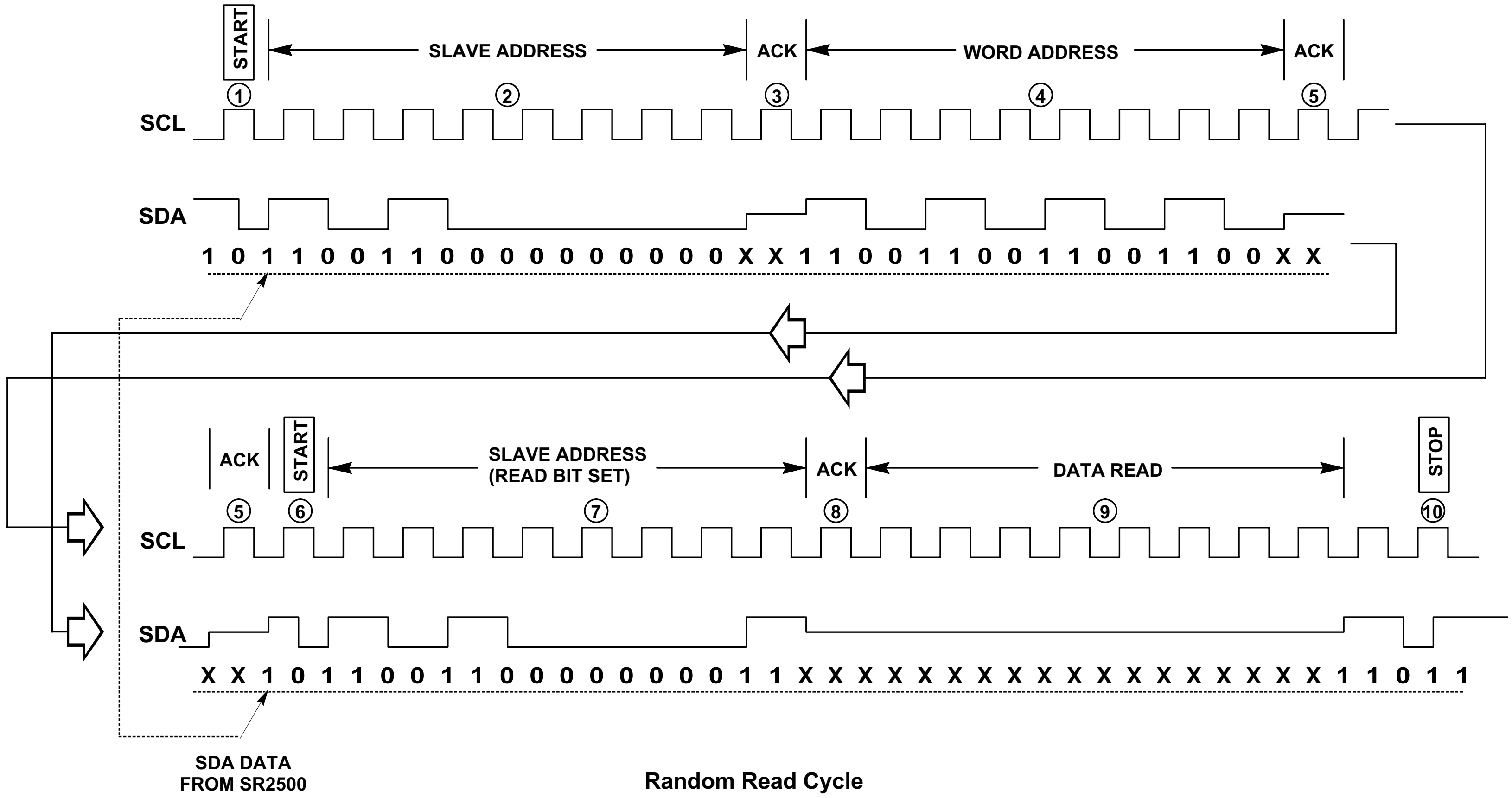
Step 3 The SR2500 will tristate its outputs so that the slave can send an acknowledge signifying that data transfer was successful. Note the programmed data for the SDA pins at this point is X or “tristate”.

Step 4 After the slave address acknowledge, the appropriate word address should be sent to the X24C16 to read the desired location. For this document, the appropriate word address is the same as the address written too in the write cycle above 10101010[B].

Step 5 The SR2500 will again tristate its outputs so that the slave can send an acknowledge signifying that data transfer was successful. Note the programmed data for the SDA pins at this point is X or “tristate”.

Step 6 After the word address acknowledge, the master immediately reissues the start condition to continue with the read cycle.

Step 7 The slave address will again be placed upon the bus, this time with the read bit set high to command a read cycle.


START
SLAVE ADDRESS
ACK
WORD ADDRESS
ACK
1
2
3
4
5
SCL
SDA
1 0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 X X 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 X X
ACK
START
SLAVE ADDRESS
(READ BIT SET)
ACK
DATA READ
STOP
5
6
7
8
9
10
SCL
SDA
X X 1 0 1 1 0 0 1 1 0 0 0 0 0 0 0 1 1 X X X X X X X X X X X X X X X X X X 1 1 0 1 1
SDA DATA
FROM SR2500

Random Read Cycle

Figure 4.

Step 8 The SR2500 will tristate its outputs so that the slave can send an acknowledge signifying that data transfer was successful. Note the programmed data for the SDA pins at this point is X or "tristate".

Step 9 After the slave address acknowledge, the master will continue to tristate its outputs so that the slave can place the byte to be read onto the bus.

Step 10 The read operation is terminated by the master not responding with an acknowledge, and issuing a STOP condition. Generate a STOP condition by transitioning SDA low to high during a high state of SCL. This terminates all data communication.

Conclusion

Digital test is ever changing, as much so as technology is asking for smaller, faster and less expensive products. Flexible digital test equipment can be used in many ways to develop and test these new technologies. The content of this document describes how one particular aspect of EEprom testing can simply be achieved, and although it demonstrates a simple write and read operation, it can be expanded beyond its simple form to thoroughly test this EEprom and multitudes of its kind. The user could very well expand upon this to test the entire memory with various forms of data, as well as include real time comparison and error checking that would execute an event based upon a non-compared acknowledge signal or erred read cycle. Bus emulation and memory test require that the tester can accurately emulate the timing characteristics of that bus, and control/setup the data flow of that memory to be tested.

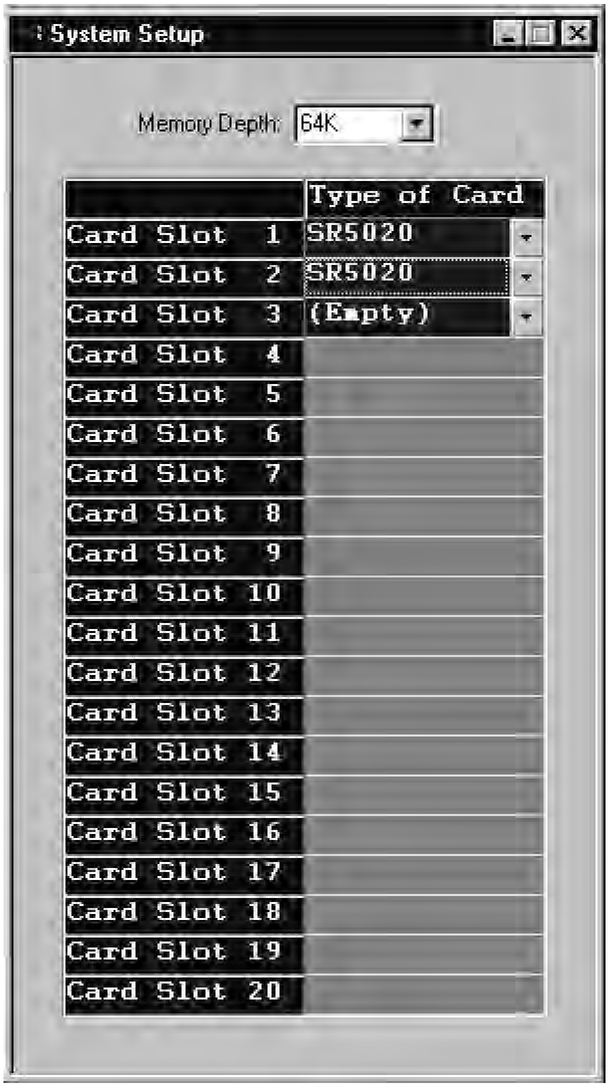
(THIS PAGE INTENTIONALLY LEFT BLANK)

App/Tech Note

Using VisTE Software With the SR2500 SR2500-09

System Setup

When configuring an SR2500 system with 64K of memory use the 5020 I/O module to represent each group of 32 I/O channels. When using an SR2500 system with 256K memory, select the 256K Memory Depth option in VisTE and use the SR5540 I/O module to represent each group of 32 I/O channels, see Fig 1.



System Setup

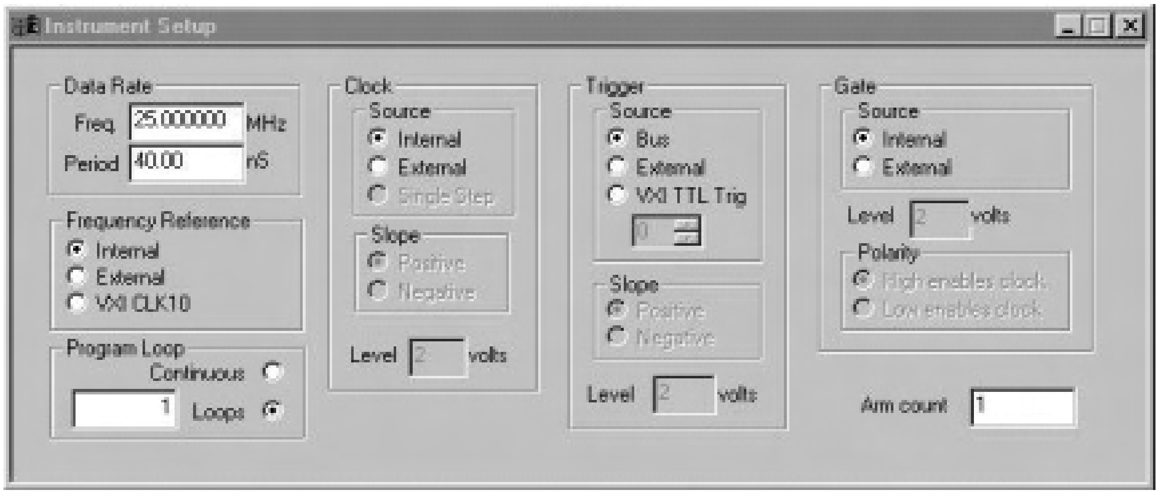
Memory Depth: 64K

	Type of Card
Card Slot 1	SR5020
Card Slot 2	SR5020
Card Slot 3	(Empty)
Card Slot 4	
Card Slot 5	
Card Slot 6	
Card Slot 7	
Card Slot 8	
Card Slot 9	
Card Slot 10	
Card Slot 11	
Card Slot 12	
Card Slot 13	
Card Slot 14	
Card Slot 15	
Card Slot 16	
Card Slot 17	
Card Slot 18	
Card Slot 19	
Card Slot 20	

Figure 1. System Setup.

Instrument Setup

The frequency range of the SR5000 and SR5500 is 400 Hz to 50 MHz. The frequency range of the SR2500 is 200 Hz to 25 MHz. If you need to program the SR2500 between 200 Hz and 400 Hz, the frequency can be changed using the P&P drivers, see Fig 2.



Instrument Setup

Data Rate
Freq. 25.000000 MHz
Period 40.00 nS

Frequency Reference
● Internal
○ External
○ VXI CLK10

Program Loop
Continuous ○
1 Loops ●

Clock
Source
● Internal
○ External
○ Single Step
Slope
● Positive
○ Negative
Level 2 volts

Trigger
Source
● Bus
○ External
○ VXI TTL Trig
0
Slope
● Positive
○ Negative
Level 2 volts

Gate
Source
● Internal
○ External
Level 2 volts
Polarity
● High enables clock
○ Low enables clock

Arm count 1

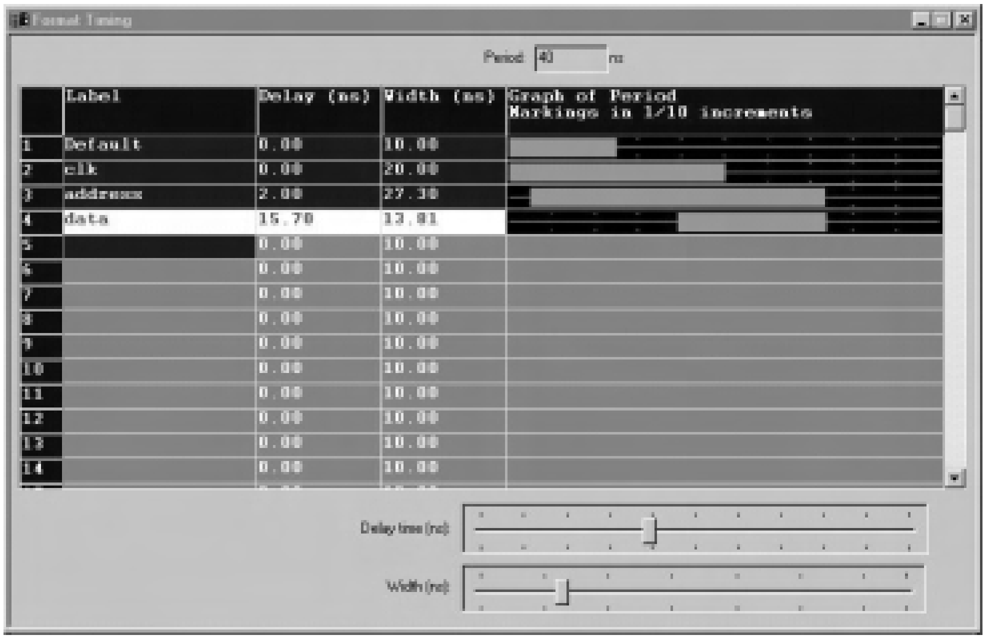
Figure 2. Instrument Setup Screen.

Fields/Format Timing

The SR5000 and SR5500 allow two stimulus timing delays per 8 channels with 100 ps resolution. The SR2500 allows two stimulus timing delays per 32 channels with 5-10 ns resolution, see Fig 3.

Note

The SR2500 will automatically select the closest available timing delay. For example, if you were to enter the timing parameters for the ‘data’ field shown below in figure 3, (delay = 15.7 ns, width = 13.81 ns), the SR2500 would select 15 ns for the delay and 15 ns for the width parameters.



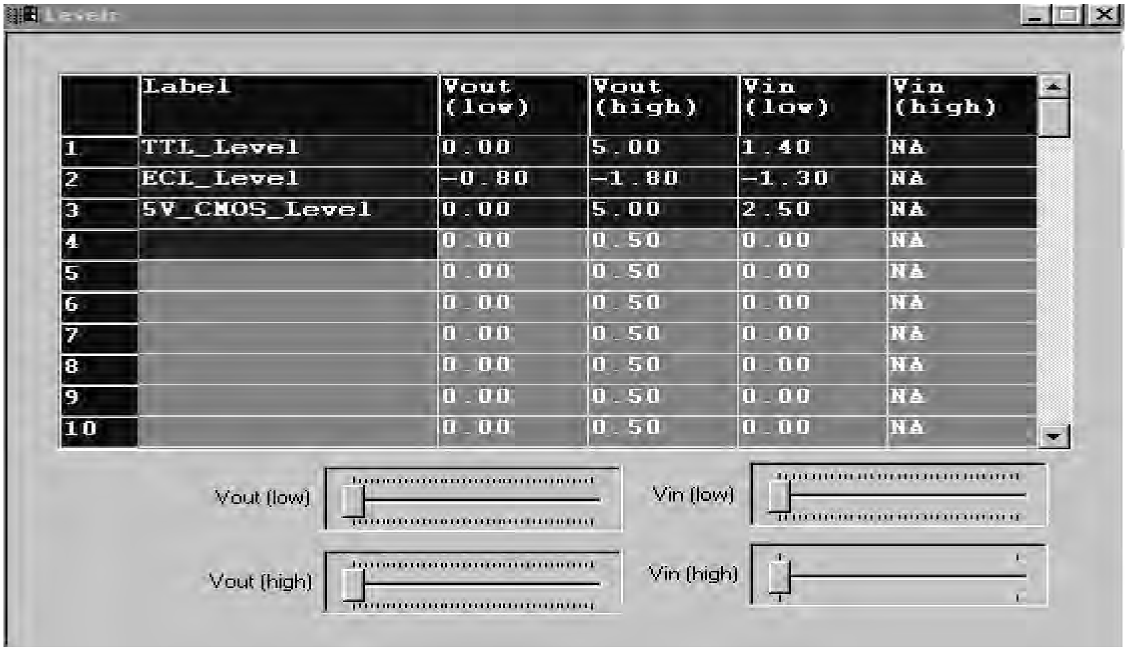
Period: 40 ns

	Label	Delay (ns)	Width (ns)	Graph of Period Markings in 1/10 increments
1	Default	0.00	10.00	
2	clk	0.00	20.00	
3	address	2.00	27.30	
4	data	15.70	13.81	
5		0.00	10.00	
6		0.00	10.00	
7		0.00	10.00	
8		0.00	10.00	
9		0.00	10.00	
10		0.00	10.00	
11		0.00	10.00	
12		0.00	10.00	
13		0.00	10.00	
14		0.00	10.00	

Figure 3. Format Timing Screen.

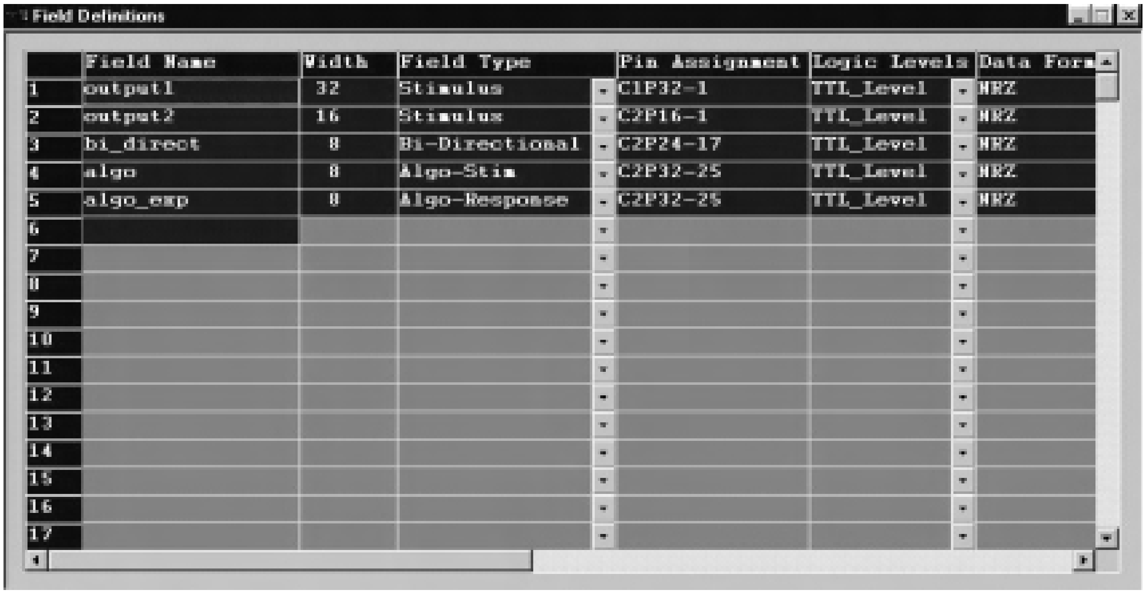
Field/Levels:

In order for proper pinlist match and field definition, the default TTL voltage level must be selected, see Figs 4 and 5.



	Label	Vout (low)	Vout (high)	Vin (low)	Vin (high)
1	TTL_Level	0.00	5.00	1.40	NA
2	ECL_Level	-0.80	-1.80	-1.30	NA
3	5V_CMOS_Level	0.00	5.00	2.50	NA
4		0.00	0.50	0.00	NA
5		0.00	0.50	0.00	NA
6		0.00	0.50	0.00	NA
7		0.00	0.50	0.00	NA
8		0.00	0.50	0.00	NA
9		0.00	0.50	0.00	NA
10		0.00	0.50	0.00	NA

Figure 4. TTL Voltage Level Selection.



Field Definitions

	Field Name	Width	Field Type	Pin Assignment	Logic Levels	Data Form
1	output1	32	Stimulus	C1P32-1	TTL_Level	NRZ
2	output2	16	Stimulus	C2P16-1	TTL_Level	NRZ
3	bi_direct	8	Bi-Directional	C2P24-17	TTL_Level	NRZ
4	algo	8	Algo-Stim	C2P32-25	TTL_Level	NRZ
5	algo_exp	8	Algo-Response	C2P32-25	TTL_Level	NRZ
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						
16						
17						

Figure 5. Field Definitions.

Differential ECL, and Differential TTL SR2500 considerations using VisTE

Special considerations should also be taken with reference to Differential TTL, and Differential ECL modules with respect to tri-state control.

A differential ECL card provides 32 channels of differential ECL to/from the UUT. Bi-directional signals are not supported directly on the board, however, 32 tri-state control signals are also brought out the Differential ECL board. This gives the user the ability to control tri-state conditions directly at the UUT or with the use of external pods. Although the tri-state control is external there is a one to one correlation with the outputs so that programming is performed similar to controlling tri-state conditions with a SR5020 TTL card which VisTE will allow.

A differential TTL card must be handled somewhat different concerning tri-state control. Output enables are controlled in groups of 4. So that,

Tri-state bit 0 (Pin 1) enables bits 0-3 (Pins 1-4)

Tri-state bit 4 (Pin 5) enables bits 4-7 (Pins 5-8)

Tri-state bit 8 (Pin 9) enables bits 8-11 (Pins 9-12)

Tri-state bit 12 (Pin 13) enables bits 12-15 (Pins 13-16)

Tri-state bit 16 (Pin 17) enables bits 16-19 (Pins 17-20)

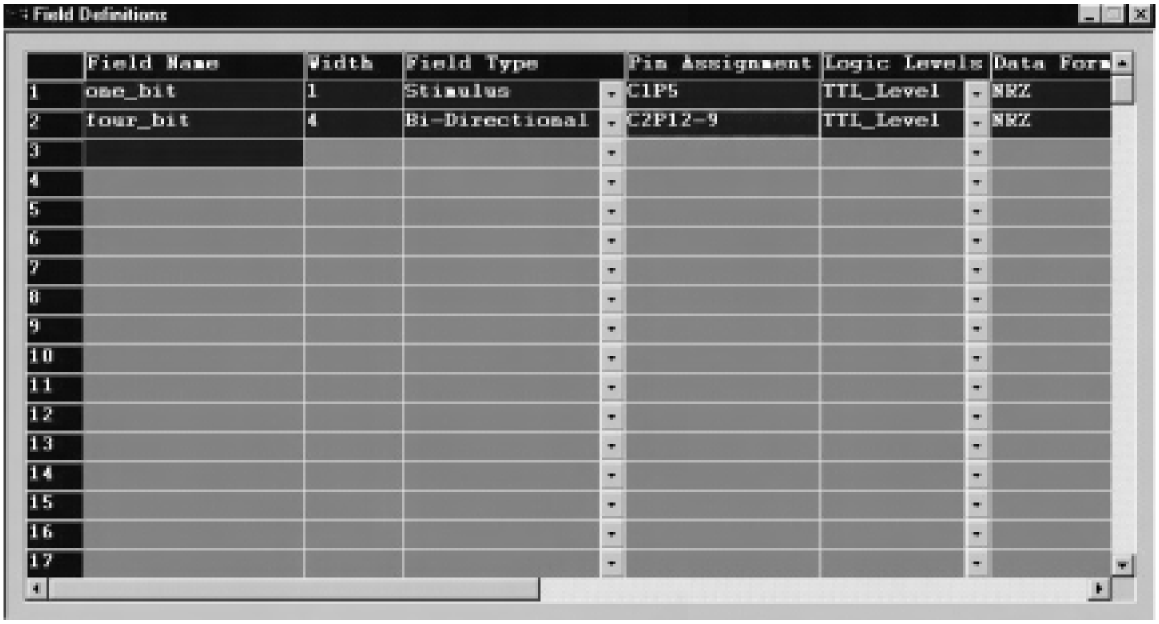
Tri-state bit 20 (Pin 21) enables bits 20-23 (Pins 21-24)

Tri-state bit 24 (Pin 25) enables bits 24-27 (Pins 25-28)

Tri-state bit 28 (Pin 29) enables bits 28-31 (Pins 29-32)

VisTE will automatically generate a tri-state field for each output field that is defined. Field definitions should take consideration that tri-state controls are defined in such a manner to adequately control outputs.

For example, single pin field definitions, if possible, should be defined corresponding to tri-state bit enable boundaries so that when tri-state conditions for those pins are disabled the outputs are enabled and no manipulation outside of VisTE is needed. If that field is not defined upon those boundaries the tri-state condition is not addressed so the default tristate condition will be used, which is to enable the tri-state and disable the outputs. If the pin associated with the tristate control for a groups of four pins is not included in the field definition, then there is no mechanism to enable the outputs for any pin in that group. This is also true considering larger fields that are not defined on 4 bit boundaries. Any tri-state fields that need to be addressed can also be modified within the Plug and Play driver after a test is loaded, see Fig 6.



	Field Name	Width	Field Type	Pin Assignment	Logic Levels	Data Form
1	one_bit	1	Stimulus	C1P5	TTL_Level	NRZ
2	four_bit	4	Bi-Directional	C2P12-9	TTL_Level	NRZ
3						
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						
16						
17						

Figure 6. Field Definitions Screen.

Variable Voltage Considerations using VisTE

When using the SR2500 programmable voltage module a RG25000 rail generator card is used to provide I/O voltages and thresholds. VisTE does not address the RG2500 card, although simple SCPI commands can be sent to the instrument to set up the Rail Generator. Sample SCPI commands to set up all Rail Generator functions are shown below:

***** Setup the rail voltages for output fields *****

SYST:RGEN 1:RAIL A1:HIGH 6.5V;LOW 4.0V

SYST:RGEN 1:RAIL A2:HIGH 5.0V;LOW 2.0V

SYST:RGEN 1:RAIL B1:HIGH 1.5V;LOW –1.5V

SYST:RGEN 1:RAIL B2:HIGH 0.0V;LOW –4.0V

After the levels are setup for the outputs, the input threshold levels can be set.

***** Setup the threshold levels for input fields *****

SYST:RGEN 1:THRES A1:HIGH 5.5V;LOW 5.0V

SYST:RGEN 1:THRES A2:HIGH 4.0V;LOW 3.0V

SYST:RGEN 1:THRES B1:HIGH 0.8V;LOW –0.5V

SYST:RGEN 1:THRES B2:HIGH –1.0V;LOW –3.0V

Once the rail and threshold levels have been set, field definitions can be associated to use the appropriate levels.

***** Apply Stimulus field definitions to defined rail levels *****

```
STIM:COND:OFOR:FIEL DC;VOLT A

STIM:COND:OFOR:FIEL OUTPUT;VOLT B

STIM:COND:OFOR:FIEL BIDIRECT;VOLT A
```

***** Apply Response field definitions to defined threshold levels *****

```
REC:COND:SAMP:FIEL DC_EXPCT;THRES A

REC:COND:SAMP:FIEL INPUT;THRES B

REC:COND:SAMP:FIEL RECORD;THRES A
```

The rail generator supplies 16 independently programmable output voltages to the SR25000 variable voltage module(s). The voltages are supplied via two output connectors (output 1 and output 2) located on the front panel of the Rail Generator; each connector supplies four rail voltages and four threshold voltages. Commands are also needed to connect or disconnect the output voltages on the RG2500 connectors.

***** Connect the output voltages for rail generator 1 *****

```
SYSTEM:RGEN 1:CONN 1
```

***** Disconnect the output voltages for rail generator 1 *****

```
SYSTEM:RGEN 1:DISC 1
```

(THIS PAGE INTENTIONALLY LEFT BLANK)

App/Tech Note

SRAM Soft Error Test System SR2500-10

Introduction

The transition to smaller and smaller micron processor technologies have increased the frequency of soft error rates (SERs) in SRAM devices. Soft error is a natural phenomenon that is caused when a burst of energy, caused by the collision of two atoms, follows a certain path where the semiconductor is trying to measure what is being stored in that cell. Thus, the energy can cause the circuitry to read or write the wrong information.

A method of detecting whether wrong information is corrupting the SRAM device is being utilized by a large semiconductor manufacturer through the use of Interface Technology’s SR2500 Digital Test Subsystems in a Soft Error Tester.

Below describes the configuration of the system and how the Soft Error System is used to detect these rapidly increasing failure rates.

Purpose & Objective

The purpose of the test system described herein is to test memory chips for “soft errors” that occur within the memory chips as a result of external radiation such as low-energy alpha particles, high-energy cosmic particles and thermal neutrons present in the environment. Testing is accomplished by writing a known digital pattern to the chips under test (e.g., a “checkerboard” 10101010 ....1010 pattern) supplied by the SR2510, and then reading the stored pattern back and comparing it, bit-by-bit, with the original written pattern. Soft errors are denoted wherever the bit patterns do not exactly match. The location of the error within the chip is identified and the bit errors tallied and read out as a Soft Error Rate (SER) on a PC screen.

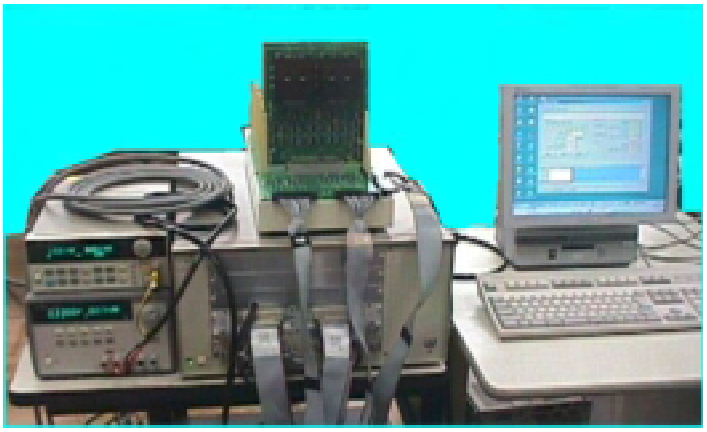


Figure 1. Photo of the Soft Error Test System.

Test Setup

See Fig 3. The SRAM Soft Error Test System can test up to eight memory boards at one time, each containing two MUT's (Memory UnderTest). Each MUT, in turn, contains two types of memory chips ...

o Type 1 — 64K words by 36-bits
o Type 2 — 4K words by 36-bits and 2 blanks.

Write-Read Sequence

Write To Memory.

All 18-bit memories are written simultaneously for both Type 1 and Type2 chips. The same data is loaded into both the lower and higher 18-bit registers for both chips, on the same board, at the same time.

Read From Memory.

Type 1: (64K words by 36-bits) First the lower 18-bit register is read, then the higher 18-bit register is read.

Type 2: (4K words by 36-bits and 2 blanks) The first 18-bit register is read, then the second, third, and fourth 18-bit registers are read.

The read sequence then moves to the next chip.

SR2510 Is Sequence Controller.

Because the SR2500 modules integrate channel density with high speed data rates, multiple types of memory boards can be tested at once. This application includes a mixture of different types of boards; for example Type 1 and Type 2 as mentioned above. The test command sequencing (CMACRO's) handle which type of board is being tested and can be easily modified to account for different combinations. The SR2500 looping and branching capabilities allow for this type of operation. Take for example a test system that would allow a maximum of 4 cards to be tested at once. Combinations possible could be (4) Type 1 cards only, (3) Type 1 cards and (1) Type 2 card, (2) type 1 cards and (2) type 2 cards,etc…. The test script can be written to address the maximum amount ofcards allowable. This type of typical application would usually require the manufacturer to load a test script based upon each new configuration.The overhead in time to load each specific test and its corresponding data, based upon various combinations, takes time and could affect productionor test time. For this application the manufacturer was able to write the test script to include all test scenarios and just modify the correspondingsequence commands (CMACRO's) to "JUMP" or skip commands that are not necessary for the particular combination. The time it takes to modify a few commands, (add or remove Jump commands) , is far less time to reload an entire test scenario. *Refer to SR2500 User's Manual for a description of the CMACRO* commands.

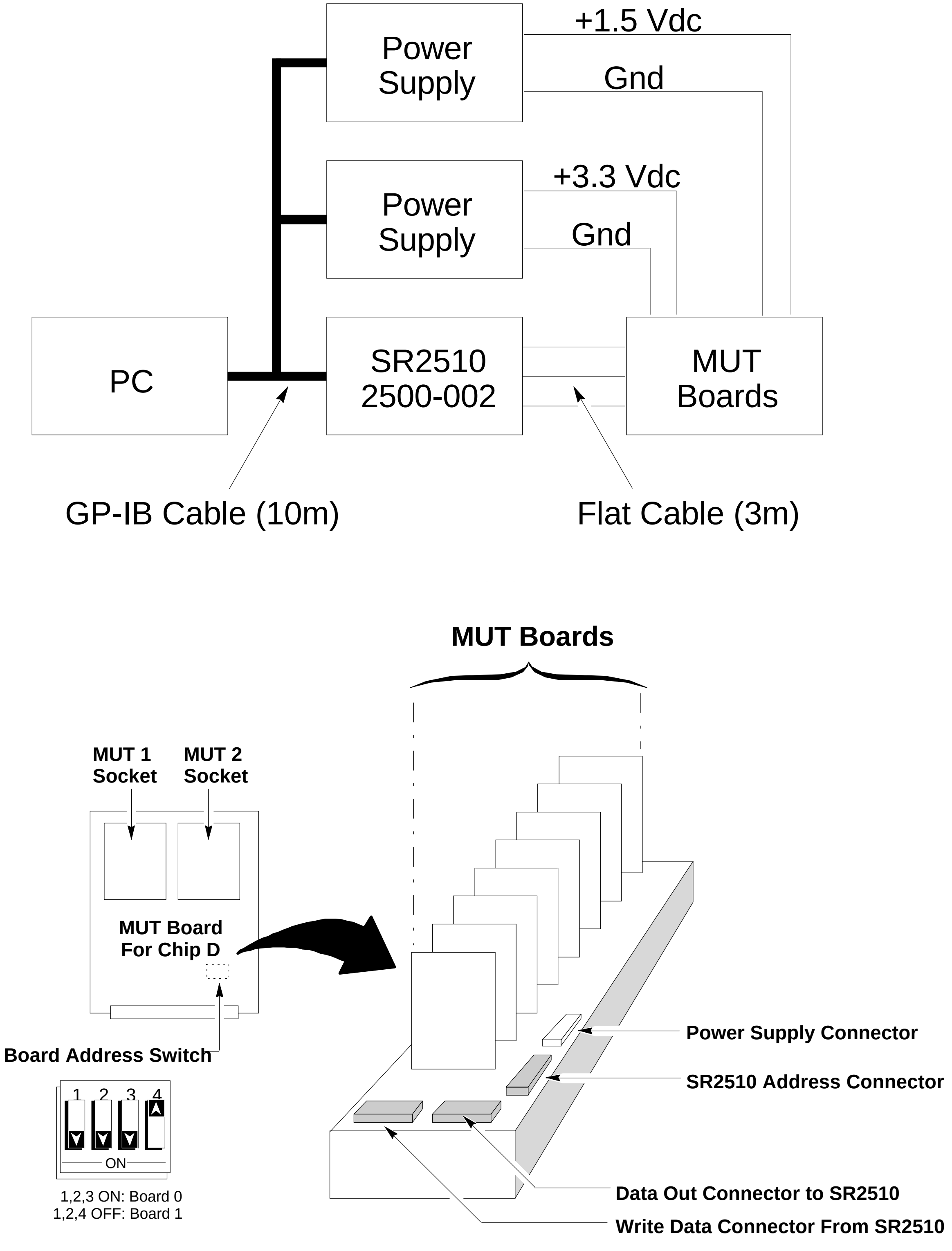


Figure 3. Test Setup Diagram.

SR2510 Program

The actual SR2510 program used to generate the test signal patterns in this test application begins page 5 of the test program. In addition to the checker board pattern, three additional test patterns can be generated with this program ... a reverse checker board pattern, an all “1’s” pattern, and an all “0’s” program. Both the checker board and the reverse checker board patterns are generated with the same CMACRO instruction. The only difference is the pattern. The all “1’s” and all “0’s” patterns are generated using the HOLDD AMACRO (see top of page 17 of test program).

The data and address patterns are created using algorithmic type field definitions. By defining the fields as algorithmic type fields the manufacturer was able to perform algorithmic functions such as INCREMENT the address, and perform an XOR (exclusive OR) function on the data patterns to load, read, or compare the desired data or address. Algorithmic commands (AMACRO’s) allow real-time generation of stimulus and expected responses based upon simple functions. By using the CMACRO looping commands, the SR2510 allows lengthy patterns to be generated with very few actual vectors and at full system speed.. This along with the fact that test load speed is faster due to less command structure makes algorithmic pattern generation ideal for Soft Error Memory Test. *Refer to SR2500 User’s Manual for a description of the AMACRO commands.*

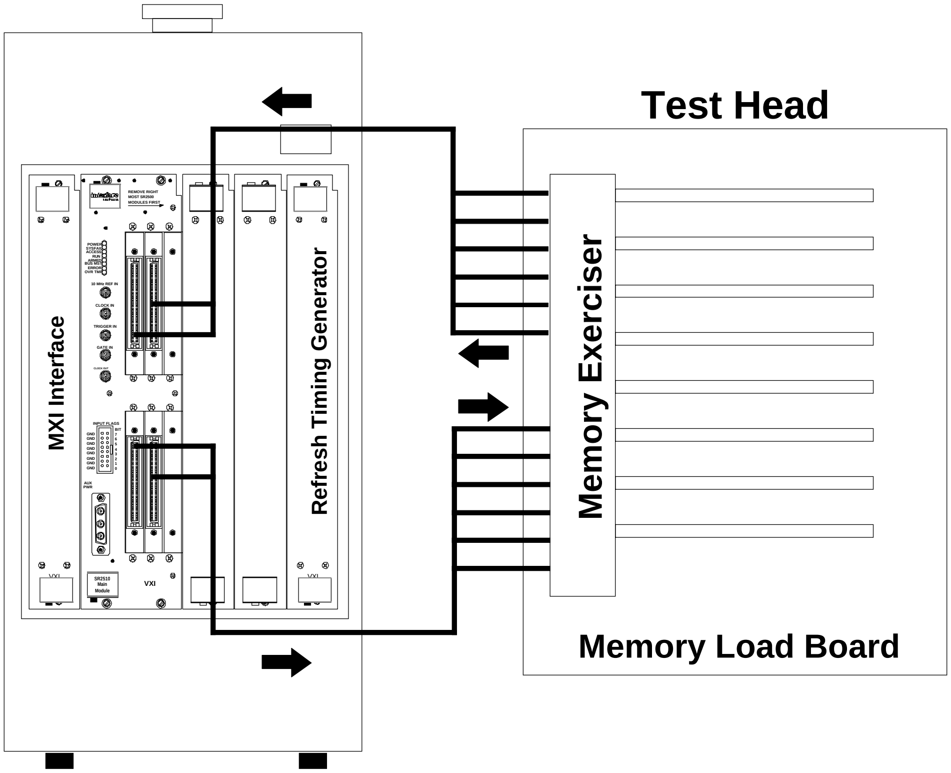


Figure 4. Test Setup Using SR2510.

```
;****SR 2500 PROG 1 REPEAT TEST DATA PATTERN : Checker Board****
;*****CB Normal TEST ******
TEST:DEF CBNRM:SIZE 65500
SYST:TEST CBNRM
SYST:FREQ 5000000
FIEL:DEF ADDRO:TYPE ALGO:PIN C1P24-1
FIELD:NAME ADDRO:RAD HEX
FIEL:DEF ADDRE:TYPE ALGE:PIN C1P24-1
FIELD:NAME ADDRE:RAD HEX
FIEL:DEF ADDROT:TYPE OT:PIN C1P16-1
FIELD:NAME ADDROT:RAD HEX
FIEL:DEF ADDRED:TYPE ED:PIN C1P16-1
FIELD:NAME ADDRED:RAD HEX

FIEL:DEF ADDRR:TYPE RECORD:PIN C1P16-1
FIELD:NAME ADDRR:RAD HEX
FIEL:DEF OSELOT:TYPE OT:PIN C1P17
FIELD:NAME OSELOT:RAD BIN
FIEL:DEF OSELED:TYPE ED:PIN C1P17
FIELD:NAME OSELED:RAD BIN
FIEL:DEF OSELR:TYPE RECORD:PIN C1P17
FIELD:NAME OSELR:RAD BIN

FIEL:DEF MACROOT:TYPE OT:PIN C1P21-18
FIELD:NAME MACROOT:RAD HEX
FIEL:DEF MACROED:TYPE ED:PIN C1P21-18
FIELD:NAME MACROED:RAD HEX
FIEL:DEF MACROR:TYPE RECORD:PIN C1P21-18
FIELD:NAME MACROR:RAD HEX

FIEL:DEF CSELOT:TYPE OT:PIN C1P26-25
FIELD:NAME CSELOT:RAD BIN
FIEL:DEF CSELED:TYPE ED:PIN C1P26-25
FIELD:NAME CSELED:RAD BIN
FIEL:DEF CSELR:TYPE RECORD:PIN C1P26-25
FIELD:NAME CSELR:RAD BIN

FIEL:DEF BDSELOT:TYPE OT:PIN C1P29-27
FIELD:NAME BDSELOT:RAD HEX
FIEL:DEF BDSELED:TYPE ED:PIN C1P29-27
FIELD:NAME BDSELED:RAD HEX
FIEL:DEF BDSELR:TYPE RECORD:PIN C1P29-27
FIELD:NAME BDSELR:RAD HEX

FIEL:DEF RAMCLKOT:TYPE OT:PIN C1P30
```

Note:

For your convenience, this entire test program can be downloaded, in MS-Word format, from the web site at:

http://www.interfacetech.com/appnotes.html.

```
FIELD:NAME RAMCLKOT:RAD BIN
FIEL:DEF RAMCLKED:TYPE ED:PIN C1P30
FIELD:NAME RAMCLKED:RAD BIN.
FIEL:DEF RAMCLKR:TYPE RECORD:PIN C1P30
FIELD:NAME RAMCLKR:RAD BIN

FIEL:DEF FFCLKOT:TYPE OT:PIN C2P30
FIELD:NAME FFCLKOT:RAD BIN
FIEL:DEF FFCLKED:TYPE ED:PIN C2P30
FIELD:NAME FFCLKED:RAD BIN
FIEL:DEF FFCLKR:TYPE RECORD:PIN C2P30
FIELD:NAME FFCLKR:RAD BIN

FIEL:DEF WEBOT:TYPE OT:PIN C1P31
FIELD:NAME WEBOT:RAD BIN
FIEL:DEF WEBED:TYPE ED:PIN C1P31
FIELD:NAME WEBED:RAD BIN
FIEL:DEF WEBR:TYPE RECORD:PIN C1P31
FIELD:NAME WEBR:RAD BIN

FIEL:DEF DATAO:TYPE ALGO:PIN C2P24-1
FIELD:NAME DATAO:RAD HEX
FIEL:DEF DATAOT:TYPE OT:PIN C2P24-1
FIELD:NAME DATAOT:RAD HEX
FIEL:DEF DATAE:TYPE ALGE:PIN C2P24-1
FIELD:NAME DATAE:RAD HEX

FIEL:DEF DATAED:TYPE ED:PIN C2P24-1
FIELD:NAME DATAED:RAD HEX
FIEL:DEF DATAD:TYPE DON:PIN C2P24-1
FIELD:NAME DATAD:RAD HEX
FIEL:DEF DATAR:TYPE RECORD:PIN C2P24-1
FIELD:NAME DATAR:RAD HEX

FIEL:DEF SYNCOT:TYPE OT:PIN C1P32
FIELD:NAME SYNCOT:RAD BIN
FIEL:DEF SYNCED:TYPE ED:PIN C1P32
FIELD:NAME SYNCED:RAD BIN
FIEL:DEF SYNCR:TYPE RECORD:PIN C1P32
FIELD:NAME SYNCR:RAD BIN

;Set timings.
STIM:COND:OFOR:FIEL DATAOT;MODE NRZ,0.000000E-8
STIM:COND:OFOR:FIEL ADDROT;MODE NRZ,0.000000E-8
STIM:COND:OFOR:FIEL RAMCLKOT;MODE RZ,2.500000E-8,6.500000E-8
```

```
STIM:COND:OFOR:FIEL FFCLKOT;MODE RZ,5.000000E-8,4.000000E-8
REC:COND:SAMP:FIEL SYNCED;MODE EDGE,0.950000E-7
REC:COND:SAMP:FIEL DATAED;MODE EDGE,1.500000E-7
REC:COND:SAMP:FIEL ADDRED;MODE EDGE,0.950000E-7
REC:COND:SAMP:FIEL OSELED;MODE EDGE,0.950000E-7
REC:COND:SAMP:FIEL MACROED;MODE EDGE,0.950000E-7
REC:COND:SAMP:FIEL CSELED;MODE EDGE,0.950000E-7.
REC:COND:SAMP:FIEL BDSELED;MODE EDGE,0.950000E-7
REC:COND:SAMP:FIEL RAMCLKED;MODE EDGE,0.500000E-7
REC:COND:SAMP:FIEL FFCLKED;MODE EDGE,0.400000E-7
REC:COND:SAMP:FIEL WEBED;MODE EDGE,0.950000E-7
;CMACRO program commands.
STIM:VEC 1;COUN 1;CMACRO:DEF (SP(OUT))
STIM:VEC 2;COUN 1;CMACRO:DEF (WL(OUT(COUN==10)))
STIM:VEC 3;COUN 1;CMACRO:DEF (CLEARE(OUT))
STIM:VEC 4;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 5;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 6;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 7;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 8;COUN 1;CMACRO:DEF (SL(OUT(COUN==1024)))
STIM:VEC 9;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 10;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 11;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 12;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 13;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 14;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 15;COUN 1;CMACRO:DEF (SL(OUT(COUN==1024)))
STIM:VEC 16;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 17;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 18;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 19;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 20;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 21;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 22;COUN 1;CMACRO:DEF (SL(OUT(COUN==1024)))
STIM:VEC 23;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 24;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 25;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 26;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 27;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 28;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 29;COUN 1;CMACRO:DEF (SL(OUT(COUN==1024)))
STIM:VEC 30;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 31;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 32;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 33;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
```

```
STIM:VEC 34;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 35;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 36;COUN 1;CMACRO:DEF (SL(OUT(COUN==1024)))
STIM:VEC 37;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 38;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 39;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 40;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 41;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 42;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 43;COUN 1;CMACRO:DEF (SL(OUT(COUN==1024)))
STIM:VEC 44;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 45;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 46;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 47;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 48;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 49;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 50;COUN 1;CMACRO:DEF (SL(OUT(COUN==1024)))
STIM:VEC 51;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 52;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 53;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 54;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 55;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 56;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 57;COUN 1;CMACRO:DEF (SL(OUT(COUN==1024)))
STIM:VEC 58;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 59;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 60;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 61;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 62;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 63;COUN 1;CMACRO:DEF ((LAB WC0)OUT(OUT))
STIM:VEC 64;COUN 1;CMACRO:DEF (SL(OUT(COUN==128)))
STIM:VEC 65;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 66;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 67;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 68;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 69;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 70;COUN 1;CMACRO:DEF ((LAB WC1)OUT(OUT))
STIM:VEC 71;COUN 1;CMACRO:DEF (SL(OUT(COUN==128)))
STIM:VEC 72;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 73;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 74;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 75;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 76;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 77;COUN 1;CMACRO:DEF ((LAB WC2)OUT(OUT))
STIM:VEC 78;COUN 1;CMACRO:DEF (SL(OUT(COUN==128)))
```

```
STIM:VEC 79;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 80;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 81;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 82;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 83;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 84;COUN 1;CMACRO:DEF ((LAB WC3)OUT(OUT))
STIM:VEC 85;COUN 1;CMACRO:DEF (SL(OUT(COUN==128)))
STIM:VEC 86;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 87;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 88;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 89;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 90;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 91;COUN 1;CMACRO:DEF ((LAB WC4)OUT(OUT))
STIM:VEC 92;COUN 1;CMACRO:DEF (SL(OUT(COUN==128)))
STIM:VEC 93;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 94;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 95;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 96;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 97;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 98;COUN 1;CMACRO:DEF ((LAB WC5)OUT(OUT))
STIM:VEC 99;COUN 1;CMACRO:DEF (SL(OUT(COUN==128)))
STIM:VEC 100;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 101;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 102;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 103;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 104;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 105;COUN 1;CMACRO:DEF ((LAB WC6)OUT(OUT))
STIM:VEC 106;COUN 1;CMACRO:DEF (SL(OUT(COUN==128)))
STIM:VEC 107;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 108;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 109;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 110;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 111;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 112;COUN 1;CMACRO:DEF ((LAB WC7)OUT(OUT))
STIM:VEC 113;COUN 1;CMACRO:DEF (SL(OUT(COUN==128)))
STIM:VEC 114;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 115;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 116;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 117;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 118;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 119;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 120;COUN 1;CMACRO:DEF ((LAB WEND)WL(NOP(CONTinue==TRUE)))
STIM:VEC 121;COUN 1;CMACRO:DEF ((LAB RA0)OUT(OUT))
STIM:VEC 122;COUN 1;CMACRO:DEF (SL(OUT(COUN==2048)))
STIM:VEC 123;COUN 1;CMACRO:DEF (OUT(OUT))
```

```
STIM:VEC 124;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 125;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 126;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 127;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 128;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 129;COUN 1;CMACRO:DEF (SL(OUT(COUN==2048)))
STIM:VEC 130;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 131;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 132;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 133;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 134;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 135;COUN 1;CMACRO:DEF ((LAB RA1)OUT(OUT))
STIM:VEC 136;COUN 1;CMACRO:DEF (SL(OUT(COUN==2048)))
STIM:VEC 137;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 138;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 139;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 140;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 141;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 142;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 143;COUN 1;CMACRO:DEF (SL(OUT(COUN==2048)))
STIM:VEC 144;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 145;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 146;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 147;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 148;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 149;COUN 1;CMACRO:DEF ((LAB RA2)OUT(OUT))
STIM:VEC 150;COUN 1;CMACRO:DEF (SL(OUT(COUN==2048)))
STIM:VEC 151;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 152;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 153;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 154;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 155;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 156;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 157;COUN 1;CMACRO:DEF (SL(OUT(COUN==2048)))
STIM:VEC 158;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 159;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 160;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 161;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 162;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 163;COUN 1;CMACRO:DEF ((LAB RA3)OUT(OUT))
STIM:VEC 164;COUN 1;CMACRO:DEF (SL(OUT(COUN==2048)))
STIM:VEC 165;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 166;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 167;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 168;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
```

```
STIM:VEC 169;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 170;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 171;COUN 1;CMACRO:DEF (SL(OUT(COUN==2048)))
STIM:VEC 172;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 173;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 174;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 175;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 176;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 177;COUN 1;CMACRO:DEF ((LAB RA4)OUT(OUT))
STIM:VEC 178;COUN 1;CMACRO:DEF (SL(OUT(COUN==2048)))
STIM:VEC 179;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 180;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 181;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 182;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 183;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 184;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 185;COUN 1;CMACRO:DEF (SL(OUT(COUN==2048)))
STIM:VEC 186;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 187;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 188;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 189;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 190;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 191;COUN 1;CMACRO:DEF ((LAB RA5)OUT(OUT))
STIM:VEC 192;COUN 1;CMACRO:DEF (SL(OUT(COUN==2048)))
STIM:VEC 193;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 194;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 195;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 196;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 197;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 198;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 199;COUN 1;CMACRO:DEF (SL(OUT(COUN==2048)))
STIM:VEC 200;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 201;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 202;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 203;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 204;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 205;COUN 1;CMACRO:DEF ((LAB RA6)OUT(OUT))
STIM:VEC 206;COUN 1;CMACRO:DEF (SL(OUT(COUN==2048)))
STIM:VEC 207;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 208;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 209;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 210;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 211;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 212;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 213;COUN 1;CMACRO:DEF (SL(OUT(COUN==2048)))
```

```
STIM:VEC 214;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 215;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 216;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 217;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 218;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 219;COUN 1;CMACRO:DEF ((LAB RA7)OUT(OUT))
STIM:VEC 220;COUN 1;CMACRO:DEF (SL(OUT(COUN==2048)))
STIM:VEC 221;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 222;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 223;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 224;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 225;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 226;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 227;COUN 1;CMACRO:DEF (SL(OUT(COUN==2048)))
STIM:VEC 228;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 229;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 230;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 231;COUN 1;CMACRO:DEF (WL(OUT(COUN==32)))
STIM:VEC 232;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 233;COUN 1;CMACRO:DEF ((LAB RC0)OUT(OUT))
STIM:VEC 234;COUN 1;CMACRO:DEF (SL(OUT(COUN==512)))
STIM:VEC 235;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 236;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 237;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 238;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 239;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 240;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 241;COUN 1;CMACRO:DEF (SL(OUT(COUN==512)))
STIM:VEC 242;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 243;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 244;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 245;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 246;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 247;COUN 1;CMACRO:DEF ((LAB RC1)OUT(OUT))
STIM:VEC 248;COUN 1;CMACRO:DEF (SL(OUT(COUN==512)))
STIM:VEC 249;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 250;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 251;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 252;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 253;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 254;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 255;COUN 1;CMACRO:DEF (SL(OUT(COUN==512)))
STIM:VEC 256;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 257;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 258;COUN 1;CMACRO:DEF (OUT(OUT))
```

```
STIM:VEC 259;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 260;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 261;COUN 1;CMACRO:DEF ((LAB RC2)OUT(OUT))
STIM:VEC 262;COUN 1;CMACRO:DEF (SL(OUT(COUN==512)))
STIM:VEC 263;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 264;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 265;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 266;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 267;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 268;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 269;COUN 1;CMACRO:DEF (SL(OUT(COUN==512)))
STIM:VEC 270;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 271;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 272;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 273;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 274;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 275;COUN 1;CMACRO:DEF ((LAB RC3)OUT(OUT))
STIM:VEC 276;COUN 1;CMACRO:DEF (SL(OUT(COUN==512)))
STIM:VEC 277;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 278;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 279;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 280;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 281;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 282;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 283;COUN 1;CMACRO:DEF (SL(OUT(COUN==512)))
STIM:VEC 284;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 285;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 286;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 287;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 288;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 289;COUN 1;CMACRO:DEF ((LAB RC4)OUT(OUT))
STIM:VEC 290;COUN 1;CMACRO:DEF (SL(OUT(COUN==512)))
STIM:VEC 291;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 292;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 293;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 294;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 295;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 296;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 297;COUN 1;CMACRO:DEF (SL(OUT(COUN==512)))
STIM:VEC 298;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 299;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 300;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 301;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 302;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 303;COUN 1;CMACRO:DEF ((LAB RC5)OUT(OUT))
```

```
STIM:VEC 304;COUN 1;CMACRO:DEF (SL(OUT(COUN==512)))
STIM:VEC 305;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 306;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 307;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 308;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 309;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 310;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 311;COUN 1;CMACRO:DEF (SL(OUT(COUN==512)))
STIM:VEC 312;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 313;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 314;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 315;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 316;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 317;COUN 1;CMACRO:DEF ((LAB RC6)OUT(OUT))
STIM:VEC 318;COUN 1;CMACRO:DEF (SL(OUT(COUN==512)))
STIM:VEC 319;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 320;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 321;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 322;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 323;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 324;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 325;COUN 1;CMACRO:DEF (SL(OUT(COUN==512)))
STIM:VEC 326;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 327;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 328;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 329;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 330;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 331;COUN 1;CMACRO:DEF ((LAB RC7)OUT(OUT))
STIM:VEC 332;COUN 1;CMACRO:DEF (SL(OUT(COUN==512)))
STIM:VEC 333;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 334;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 335;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 336;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 337;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 338;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 339;COUN 1;CMACRO:DEF (SL(OUT(COUN==512)))
STIM:VEC 340;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 341;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 342;COUN 1;CMACRO:DEF (OUT(OUT))
STIM:VEC 343;COUN 1;CMACRO:DEF (WL(OUT(COUN==16)))
STIM:VEC 344;COUN 1;CMACRO:DEF (EL(OUT))
STIM:VEC 345;COUN 1;CMACRO:DEF ((LAB END)EP(NOP))

;Define address pattern
STIM:VEC 8;COUN 6;AMAC:FIEL ADDRO;PATT HOLDA, HOLDA, INC, HOLDA, INC,
HOLDA
STIM:VEC 15;COUN 6;AMAC:FIEL ADDRO;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA
STIM:VEC 22;COUN 6;AMAC:FIEL ADDRO;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA
STIM:VEC 29;COUN 6;AMAC:FIEL ADDRO;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA
STIM:VEC 36;COUN 6;AMAC:FIEL ADDRO;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA
STIM:VEC 43;COUN 6;AMAC:FIEL ADDRO;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA
STIM:VEC 50;COUN 6;AMAC:FIEL ADDRO;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA
STIM:VEC 57;COUN 6;AMAC:FIEL ADDRO;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA
STIM:VEC 63;COUN 1;AMAC:FIEL ADDRO;PATT NONA
STIM:VEC 64;COUN 6;AMAC:FIEL ADDRO;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA
STIM:VEC 71;COUN 6;AMAC:FIEL ADDRO;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA
STIM:VEC 78;COUN 6;AMAC:FIEL ADDRO;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA
STIM:VEC 85;COUN 6;AMAC:FIEL ADDRO;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA
STIM:VEC 92;COUN 6;AMAC:FIEL ADDRO;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA
STIM:VEC 99;COUN 6;AMAC:FIEL ADDRO;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA
STIM:VEC 106;COUN 6;AMAC:FIEL ADDRO;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA
STIM:VEC 113;COUN 6;AMAC:FIEL ADDRO;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA
STIM:VEC 120;COUN 1;AMAC:FIEL ADDRO;PATT HOLDD
STIM:VEC 122;COUN 13;AMAC:FIEL ADDRO;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA, NONA, HOLDA, HOLDA, INC, HOLDA, INC, HOLDA
STIM:VEC 136;COUN 13;AMAC:FIEL ADDRO;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA, NONA, HOLDA, HOLDA, INC, HOLDA, INC, HOLDA
STIM:VEC 150;COUN 13;AMAC:FIEL ADDRO;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA, NONA, HOLDA, HOLDA, INC, HOLDA, INC, HOLDA
STIM:VEC 164;COUN 13;AMAC:FIEL ADDRO;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA, NONA, HOLDA, HOLDA, INC, HOLDA, INC, HOLDA
STIM:VEC 178;COUN 13;AMAC:FIEL ADDRO;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA, NONA, HOLDA, HOLDA, INC, HOLDA, INC, HOLDA
STIM:VEC 192;COUN 13;AMAC:FIEL ADDRO;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA, NONA, HOLDA, HOLDA, INC, HOLDA, INC, HOLDA
```

```
STIM:VEC 206;COUN 13;AMAC:FIEL ADDRO;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA, NONA, HOLDA, HOLDA, INC, HOLDA, INC, HOLDA
STIM:VEC 220;COUN 13;AMAC:FIEL ADDRO;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA, NONA, HOLDA, HOLDA, INC, HOLDA, INC, HOLDA
STIM:VEC 234;COUN 13;AMAC:FIEL ADDRO;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA, NONA, HOLDA, HOLDA, INC, HOLDA, INC, HOLDA
STIM:VEC 248;COUN 13;AMAC:FIEL ADDRO;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA, NONA, HOLDA, HOLDA, INC, HOLDA, INC, HOLDA
STIM:VEC 262;COUN 13;AMAC:FIEL ADDRO;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA, NONA, HOLDA, HOLDA, INC, HOLDA, INC, HOLDA
STIM:VEC 276;COUN 13;AMAC:FIEL ADDRO;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA, NONA, HOLDA, HOLDA, INC, HOLDA, INC, HOLDA
STIM:VEC 290;COUN 13;AMAC:FIEL ADDRO;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA, NONA, HOLDA, HOLDA, INC, HOLDA, INC, HOLDA
STIM:VEC 304;COUN 13;AMAC:FIEL ADDRO;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA, NONA, HOLDA, HOLDA, INC, HOLDA, INC, HOLDA
STIM:VEC 318;COUN 13;AMAC:FIEL ADDRO;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA, NONA, HOLDA, HOLDA, INC, HOLDA, INC, HOLDA
STIM:VEC 332;COUN 13;AMAC:FIEL ADDRO;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA, NONA, HOLDA, HOLDA, INC, HOLDA, INC, HOLDA
STIM:VEC 345;COUN 1;AMAC:FIEL ADDRO;PATT HOLDD

;Define data pattern
STIM:VEC 8;COUN 6;AMAC:FIEL DATAO;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD
STIM:VEC 15;COUN 6;AMAC:FIEL DATAO;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD
STIM:VEC 22;COUN 6;AMAC:FIEL DATAO;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD
STIM:VEC 29;COUN 6;AMAC:FIEL DATAO;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD
STIM:VEC 36;COUN 6;AMAC:FIEL DATAO;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD
STIM:VEC 43;COUN 6;AMAC:FIEL DATAO;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD
STIM:VEC 50;COUN 6;AMAC:FIEL DATAO;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD
STIM:VEC 57;COUN 6;AMAC:FIEL DATAO;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD
STIM:VEC 64;COUN 6;AMAC:FIEL DATAO;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD
STIM:VEC 71;COUN 6;AMAC:FIEL DATAO;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD
STIM:VEC 78;COUN 6;AMAC:FIEL DATAO;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD
```

```
STIM:VEC 85;COUN 6;AMAC:FIEL DATAO;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD
STIM:VEC 92;COUN 6;AMAC:FIEL DATAO;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD
STIM:VEC 99;COUN 6;AMAC:FIEL DATAO;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD
STIM:VEC 106;COUN 6;AMAC:FIEL DATAO;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD
STIM:VEC 113;COUN 6;AMAC:FIEL DATAO;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD
STIM:VEC 120;COUN 1;AMAC:FIEL DATAO;PATT HOLDD
STIM:VEC 122;COUN 13;AMAC:FIEL DATAO;PATT HOLDD, NONA, XOR, NONA,
XOR, HOLDD, NONA, HOLDD, NONA, XOR, NONA, XOR, HOLDD
STIM:VEC 136;COUN 13;AMAC:FIEL DATAO;PATT HOLDD, NONA, XOR, NONA,
XOR, HOLDD, NONA, HOLDD, NONA, XOR, NONA, XOR, HOLDD
STIM:VEC 150;COUN 13;AMAC:FIEL DATAO;PATT HOLDD, NONA, XOR, NONA,
XOR, HOLDD, NONA, HOLDD, NONA, XOR, NONA, XOR, HOLDD
STIM:VEC 164;COUN 13;AMAC:FIEL DATAO;PATT HOLDD, NONA, XOR, NONA,
XOR, HOLDD, NONA, HOLDD, NONA, XOR, NONA, XOR, HOLDD
STIM:VEC 178;COUN 13;AMAC:FIEL DATAO;PATT HOLDD, NONA, XOR, NONA,
XOR, HOLDD, NONA, HOLDD, NONA, XOR, NONA, XOR, HOLDD
STIM:VEC 192;COUN 13;AMAC:FIEL DATAO;PATT HOLDD, NONA, XOR, NONA,
XOR, HOLDD, NONA, HOLDD, NONA, XOR, NONA, XOR, HOLDD
STIM:VEC 206;COUN 13;AMAC:FIEL DATAO;PATT HOLDD, NONA, XOR, NONA,
XOR, HOLDD, NONA, HOLDD, NONA, XOR, NONA, XOR, HOLDD
STIM:VEC 220;COUN 13;AMAC:FIEL DATAO;PATT HOLDD, NONA, XOR, NONA,
XOR, HOLDD, NONA, HOLDD, NONA, XOR, NONA, XOR, HOLDD
STIM:VEC 234;COUN 13;AMAC:FIEL DATAO;PATT HOLDD, NONA, XOR, NONA,
XOR, HOLDD, NONA, HOLDD, NONA, XOR, NONA, XOR, HOLDD
STIM:VEC 248;COUN 13;AMAC:FIEL DATAO;PATT HOLDD, NONA, XOR, NONA,
XOR, HOLDD, NONA, HOLDD, NONA, XOR, NONA, XOR, HOLDD
STIM:VEC 262;COUN 13;AMAC:FIEL DATAO;PATT HOLDD, NONA, XOR, NONA,
XOR, HOLDD, NONA, HOLDD, NONA, XOR, NONA, XOR, HOLDD
STIM:VEC 276;COUN 13;AMAC:FIEL DATAO;PATT HOLDD, NONA, XOR, NONA,
XOR, HOLDD, NONA, HOLDD, NONA, XOR, NONA, XOR, HOLDD
STIM:VEC 290;COUN 13;AMAC:FIEL DATAO;PATT HOLDD, NONA, XOR, NONA,
XOR, HOLDD, NONA, HOLDD, NONA, XOR, NONA, XOR, HOLDD
STIM:VEC 304;COUN 13;AMAC:FIEL DATAO;PATT HOLDD, NONA, XOR, NONA,
XOR, HOLDD, NONA, HOLDD, NONA, XOR, NONA, XOR, HOLDD
STIM:VEC 318;COUN 13;AMAC:FIEL DATAO;PATT HOLDD, NONA, XOR, NONA,
XOR, HOLDD, NONA, HOLDD, NONA, XOR, NONA, XOR, HOLDD
STIM:VEC 332;COUN 13;AMAC:FIEL DATAO;PATT HOLDD, NONA, XOR, NONA,
XOR, HOLDD, NONA, HOLDD, NONA, XOR, NONA, XOR, HOLDD
STIM:VEC 345;COUN 1;AMAC:FIEL DATAO;PATT HOLDD
```

```
;Define address data source (counter initial value)
STIM:VEC 1;COUN 8;DATA:FIEL ADDROT;PATT
#Hxxxx,#hxxxx,#hxxxx,#hxxxx,#hxxxx,#hxxxx,#hxxxx,#hxxxx
STIM:VEC 7;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 10;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 12;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 14;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 17;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 19;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 21;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 24;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 26;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 28;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 31;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 33;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 35;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 38;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 40;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 42;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 45;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 47;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 49;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 52;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 54;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 56;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 59;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 61;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 63;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 66;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 68;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 70;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 73;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 75;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 77;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 80;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 82;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 84;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 87;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 89;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 91;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 94;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 96;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 98;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 101;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 103;COUN 1;DATA:FIEL ADDROT;PATT #H0000
```

```
STIM:VEC 105;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 108;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 110;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 112;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 115;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 117;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 119;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 121;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 124;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 126;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 128;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 131;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 133;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 135;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 138;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 140;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 142;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 145;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 147;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 149;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 152;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 154;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 156;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 159;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 161;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 163;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 166;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 168;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 170;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 173;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 175;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 177;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 180;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 182;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 184;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 187;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 189;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 191;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 194;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 196;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 198;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 201;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 203;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 205;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 208;COUN 1;DATA:FIEL ADDROT;PATT #H0000
```

```
STIM:VEC 210;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 212;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 215;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 217;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 219;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 222;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 224;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 226;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 229;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 231;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 233;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 236;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 238;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 240;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 243;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 245;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 247;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 250;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 252;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 254;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 257;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 259;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 261;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 264;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 266;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 268;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 271;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 273;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 275;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 278;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 280;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 282;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 285;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 287;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 289;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 292;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 294;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 296;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 299;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 301;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 303;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 306;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 308;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 310;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 313;COUN 1;DATA:FIEL ADDROT;PATT #H0000
```

```
STIM:VEC 315;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 317;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 320;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 322;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 324;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 327;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 329;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 331;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 334;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 336;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 338;COUN 1;DATA:FIEL ADDROT;PATT #HFFFF
STIM:VEC 341;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 343;COUN 1;DATA:FIEL ADDROT;PATT #H0000
STIM:VEC 345;COUN 1;DATA:FIEL ADDROT;PATT #H0000

;Define data source (counter initial value)
STIM:VEC 1;COUN 7;DATA:FIEL DATAOT;PATT
#HFFFFFF,#HFFFFFF,#HFFFFFF,#HFFFFFF,#hFFFFFF,#hFFFFFF,#hFFFFFF
STIM:VEC 9;COUN 5;DATA:FIEL DATAOT;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF,#h000000
STIM:VEC 14;COUN 1;DATA:FIEL DATAOT;PATT #HFFFFFF
STIM:VEC 16;COUN 4;DATA:FIEL DATAOT;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
STIM:VEC 21;COUN 1;DATA:FIEL DATAOT;PATT #HFFFFFF
STIM:VEC 23;COUN 4;DATA:FIEL DATAOT;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
STIM:VEC 28;COUN 1;DATA:FIEL DATAOT;PATT #HFFFFFF
STIM:VEC 30;COUN 4;DATA:FIEL DATAOT;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
STIM:VEC 35;COUN 1;DATA:FIEL DATAOT;PATT #HFFFFFF
STIM:VEC 37;COUN 4;DATA:FIEL DATAOT;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
STIM:VEC 42;COUN 1;DATA:FIEL DATAOT;PATT #HFFFFFF
STIM:VEC 44;COUN 4;DATA:FIEL DATAOT;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
STIM:VEC 49;COUN 1;DATA:FIEL DATAOT;PATT #HFFFFFF
STIM:VEC 51;COUN 4;DATA:FIEL DATAOT;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
STIM:VEC 56;COUN 1;DATA:FIEL DATAOT;PATT #HFFFFFF
STIM:VEC 58;COUN 4;DATA:FIEL DATAOT;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
STIM:VEC 63;COUN 1;DATA:FIEL DATAOT;PATT #HFFFFFF
STIM:VEC 65;COUN 4;DATA:FIEL DATAOT;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
STIM:VEC 70;COUN 1;DATA:FIEL DATAOT;PATT #HFFFFFF
```

```
STIM:VEC 72;COUN 4;DATA:FIEL DATAOT;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
STIM:VEC 77;COUN 1;DATA:FIEL DATAOT;PATT #HFFFFFF
STIM:VEC 79;COUN 4;DATA:FIEL DATAOT;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
STIM:VEC 84;COUN 1;DATA:FIEL DATAOT;PATT #HFFFFFF
STIM:VEC 86;COUN 4;DATA:FIEL DATAOT;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
STIM:VEC 91;COUN 1;DATA:FIEL DATAOT;PATT #HFFFFFF
STIM:VEC 93;COUN 4;DATA:FIEL DATAOT;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
STIM:VEC 98;COUN 1;DATA:FIEL DATAOT;PATT #HFFFFFF
STIM:VEC 100;COUN 4;DATA:FIEL DATAOT;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
STIM:VEC 105;COUN 1;DATA:FIEL DATAOT;PATT #HFFFFFF;
STIM:VEC 107;COUN 4;DATA:FIEL DATAOT;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
STIM:VEC 112;COUN 1;DATA:FIEL DATAOT;PATT #HFFFFFF
STIM:VEC 114;COUN 4;DATA:FIEL DATAOT;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
STIM:VEC 121;COUN 1;DATA:FIEL DATAOT;PATT #H000000
STIM:VEC 123;COUN 4;DATA:FIEL DATAOT;PATT
#H000000,#H000000,#H000000,#H000000
STIM:VEC 130;COUN 4;DATA:FIEL DATAOT;PATT
#H000000,#H000000,#H000000,#H000000
STIM:VEC 137;COUN 4;DATA:FIEL DATAOT;PATT
#H000000,#H000000,#H000000,#H000000
STIM:VEC 144;COUN 4;DATA:FIEL DATAOT;PATT
#H000000,#H000000,#H000000,#H000000
STIM:VEC 151;COUN 4;DATA:FIEL DATAOT;PATT
#H000000,#H000000,#H000000,#H000000
STIM:VEC 158;COUN 4;DATA:FIEL DATAOT;PATT
#H000000,#H000000,#H000000,#H000000
STIM:VEC 165;COUN 4;DATA:FIEL DATAOT;PATT
#H000000,#H000000,#H000000,#H000000
STIM:VEC 172;COUN 4;DATA:FIEL DATAOT;PATT
#H000000,#H000000,#H000000,#H000000
STIM:VEC 179;COUN 4;DATA:FIEL DATAOT;PATT
#H000000,#H000000,#H000000,#H000000
STIM:VEC 186;COUN 4;DATA:FIEL DATAOT;PATT
#H000000,#H000000,#H000000,#H000000
STIM:VEC 193;COUN 4;DATA:FIEL DATAOT;PATT
#H000000,#H000000,#H000000,#H000000
STIM:VEC 200;COUN 4;DATA:FIEL DATAOT;PATT
#H000000,#H000000,#H000000,#H000000
```

```
STIM:VEC 207;COUN 4;DATA:FIEL DATAOT;PATT
#H000000,#H000000,#H000000,#H000000
STIM:VEC 214;COUN 4;DATA:FIEL DATAOT;PATT
#H000000,#H000000,#H000000,#H000000
STIM:VEC 221;COUN 4;DATA:FIEL DATAOT;PATT
#H000000,#H000000,#H000000,#H000000
STIM:VEC 228;COUN 4;DATA:FIEL DATAOT;PATT
#H000000,#H000000,#H000000,#H000000
STIM:VEC 235;COUN 4;DATA:FIEL DATAOT;PATT
#H000000,#H000000,#H000000,#H000000
STIM:VEC 242;COUN 4;DATA:FIEL DATAOT;PATT
#H000000,#H000000,#H000000,#H000000
STIM:VEC 249;COUN 4;DATA:FIEL DATAOT;PATT
#H000000,#H000000,#H000000,#H000000
STIM:VEC 256;COUN 4;DATA:FIEL DATAOT;PATT
#H000000,#H000000,#H000000,#H000000
STIM:VEC 263;COUN 4;DATA:FIEL DATAOT;PATT
#H000000,#H000000,#H000000,#H000000
STIM:VEC 270;COUN 4;DATA:FIEL DATAOT;PATT
#H000000,#H000000,#H000000,#H000000
STIM:VEC 277;COUN 4;DATA:FIEL DATAOT;PATT
#H000000,#H000000,#H000000,#H000000
STIM:VEC 284;COUN 4;DATA:FIEL DATAOT;PATT
#H000000,#H000000,#H000000,#H000000
STIM:VEC 291;COUN 4;DATA:FIEL DATAOT;PATT
#H000000,#H000000,#H000000,#H000000
STIM:VEC 298;COUN 4;DATA:FIEL DATAOT;PATT
#H000000,#H000000,#H000000,#H000000
STIM:VEC 305;COUN 4;DATA:FIEL DATAOT;PATT
#H000000,#H000000,#H000000,#H000000
STIM:VEC 312;COUN 4;DATA:FIEL DATAOT;PATT
#H000000,#H000000,#H000000,#H000000
STIM:VEC 319;COUN 4;DATA:FIEL DATAOT;PATT
#H000000,#H000000,#H000000,#H000000
STIM:VEC 326;COUN 4;DATA:FIEL DATAOT;PATT
#H000000,#H000000,#H000000,#H000000
STIM:VEC 333;COUN 4;DATA:FIEL DATAOT;PATT
#H000000,#H000000,#H000000,#H000000
STIM:VEC 340;COUN 4;DATA:FIEL DATAOT;PATT

#H000000,#H000000,#H000000,#H000000
STIM:VEC 1;COUN 120;DATA:FIEL OSELOT;FILL:TYPE REP;INT 1;PATT
#B1;EXEC
STIM:VEC 121;COUN 1;DATA:FIEL OSELOT;PATT #B1
STIM:VEC 124;COUN 1;DATA:FIEL OSELOT;PATT #B0
```

```
STIM:VEC 126;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 128;COUN 1;DATA:FIEL OSELOT;PATT #B1
STIM:VEC 131;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 133;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 135;COUN 1;DATA:FIEL OSELOT;PATT #B1
STIM:VEC 138;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 140;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 142;COUN 1;DATA:FIEL OSELOT;PATT #B1
STIM:VEC 145;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 147;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 149;COUN 1;DATA:FIEL OSELOT;PATT #B1
STIM:VEC 152;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 154;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 156;COUN 1;DATA:FIEL OSELOT;PATT #B1
STIM:VEC 159;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 161;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 163;COUN 1;DATA:FIEL OSELOT;PATT #B1
STIM:VEC 166;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 168;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 170;COUN 1;DATA:FIEL OSELOT;PATT #B1
STIM:VEC 173;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 175;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 177;COUN 1;DATA:FIEL OSELOT;PATT #B1
STIM:VEC 180;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 182;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 184;COUN 1;DATA:FIEL OSELOT;PATT #B1
STIM:VEC 187;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 189;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 191;COUN 1;DATA:FIEL OSELOT;PATT #B1
STIM:VEC 194;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 196;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 198;COUN 1;DATA:FIEL OSELOT;PATT #B1
STIM:VEC 201;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 203;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 205;COUN 1;DATA:FIEL OSELOT;PATT #B1
STIM:VEC 208;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 210;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 212;COUN 1;DATA:FIEL OSELOT;PATT #B1
STIM:VEC 215;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 217;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 219;COUN 1;DATA:FIEL OSELOT;PATT #B1
STIM:VEC 222;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 224;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 226;COUN 1;DATA:FIEL OSELOT;PATT #B1
STIM:VEC 229;COUN 1;DATA:FIEL OSELOT;PATT #B0
```

```
STIM:VEC 231;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 233;COUN 1;DATA:FIEL OSELOT;PATT #B1
STIM:VEC 236;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 238;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 240;COUN 1;DATA:FIEL OSELOT;PATT #B1
STIM:VEC 243;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 245;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 247;COUN 1;DATA:FIEL OSELOT;PATT #B1
STIM:VEC 250;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 252;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 254;COUN 1;DATA:FIEL OSELOT;PATT #B1
STIM:VEC 257;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 259;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 261;COUN 1;DATA:FIEL OSELOT;PATT #B1
STIM:VEC 264;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 266;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 268;COUN 1;DATA:FIEL OSELOT;PATT #B1
STIM:VEC 271;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 273;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 275;COUN 1;DATA:FIEL OSELOT;PATT #B1
STIM:VEC 278;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 280;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 282;COUN 1;DATA:FIEL OSELOT;PATT #B1
STIM:VEC 285;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 287;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 289;COUN 1;DATA:FIEL OSELOT;PATT #B1
STIM:VEC 292;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 294;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 296;COUN 1;DATA:FIEL OSELOT;PATT #B1
STIM:VEC 299;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 301;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 303;COUN 1;DATA:FIEL OSELOT;PATT #B1
STIM:VEC 306;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 308;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 310;COUN 1;DATA:FIEL OSELOT;PATT #B1
STIM:VEC 313;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 315;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 317;COUN 1;DATA:FIEL OSELOT;PATT #B1
STIM:VEC 320;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 322;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 324;COUN 1;DATA:FIEL OSELOT;PATT #B1
STIM:VEC 327;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 329;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 331;COUN 1;DATA:FIEL OSELOT;PATT #B1
STIM:VEC 334;COUN 1;DATA:FIEL OSELOT;PATT #B0
```

```
STIM:VEC 336;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 338;COUN 1;DATA:FIEL OSELOT;PATT #B1
STIM:VEC 341;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 343;COUN 1;DATA:FIEL OSELOT;PATT #B0
STIM:VEC 345;COUN 1;DATA:FIEL OSELOT;PATT #B0

STIM:VEC 7;COUN 7;DATA:FIEL MACROOT;PATT #hF,#h0,#h0,#h0,#h0,#h0,#h0

REC:VEC 7;COUN 7;DATA:FIEL MACROED;PATT #hF,#h0,#h0,#h0,#h0,#h0,#h0

STIM:VEC 1;COUN 7;DATA:FIEL CSELOT;FILL:TYPE REP;INT 1;PATT #B11;EXEC
STIM:VEC 8;COUN 112;DATA:FIEL CSELOT;FILL:TYPE REP;INT 1;PATT
#B00;EXEC
STIM:VEC 120;COUN 1;DATA:FIEL CSELOT;PATT #B11
STIM:VEC 121;COUN 14;DATA:FIEL CSELOT;PATT
#Bxx,#Bxx,#Bxx,#B10,#Bxx,#B10,#Bxx,#Bxx,#Bxx,#Bxx,#B01,#Bxx,#B01,#Bxx
STIM:VEC 135;COUN 14;DATA:FIEL CSELOT;PATT
#Bxx,#Bxx,#Bxx,#B10,#Bxx,#B10,#Bxx,#Bxx,#Bxx,#Bxx,#B01,#Bxx,#B01,#Bxx
STIM:VEC 149;COUN 14;DATA:FIEL CSELOT;PATT
#Bxx,#Bxx,#Bxx,#B10,#Bxx,#B10,#Bxx,#Bxx,#Bxx,#Bxx,#B01,#Bxx,#B01,#Bxx
STIM:VEC 163;COUN 14;DATA:FIEL CSELOT;PATT
#Bxx,#Bxx,#Bxx,#B10,#Bxx,#B10,#Bxx,#Bxx,#Bxx,#Bxx,#B01,#Bxx,#B01,#Bxx
STIM:VEC 177;COUN 14;DATA:FIEL CSELOT;PATT
#Bxx,#Bxx,#Bxx,#B10,#Bxx,#B10,#Bxx,#Bxx,#Bxx,#Bxx,#B01,#Bxx,#B01,#Bxx
STIM:VEC 191;COUN 14;DATA:FIEL CSELOT;PATT
#Bxx,#Bxx,#Bxx,#B10,#Bxx,#B10,#Bxx,#Bxx,#Bxx,#Bxx,#B01,#Bxx,#B01,#Bxx
STIM:VEC 205;COUN 14;DATA:FIEL CSELOT;PATT
#Bxx,#Bxx,#Bxx,#B10,#Bxx,#B10,#Bxx,#Bxx,#Bxx,#Bxx,#B01,#Bxx,#B01,#Bxx
STIM:VEC 219;COUN 14;DATA:FIEL CSELOT;PATT
#Bxx,#Bxx,#Bxx,#B10,#Bxx,#B10,#Bxx,#Bxx,#Bxx,#Bxx,#B01,#Bxx,#B01,#Bxx
STIM:VEC 233;COUN 14;DATA:FIEL CSELOT;PATT
#Bxx,#Bxx,#Bxx,#B10,#Bxx,#B10,#Bxx,#Bxx,#Bxx,#Bxx,#B01,#Bxx,#B01,#Bxx
STIM:VEC 247;COUN 14;DATA:FIEL CSELOT;PATT
#Bxx,#Bxx,#Bxx,#B10,#Bxx,#B10,#Bxx,#Bxx,#Bxx,#Bxx,#B01,#Bxx,#B01,#Bxx
STIM:VEC 261;COUN 14;DATA:FIEL CSELOT;PATT
#Bxx,#Bxx,#Bxx,#B10,#Bxx,#B10,#Bxx,#Bxx,#Bxx,#Bxx,#B01,#Bxx,#B01,#Bxx
STIM:VEC 275;COUN 14;DATA:FIEL CSELOT;PATT
#Bxx,#Bxx,#Bxx,#B10,#Bxx,#B10,#Bxx,#Bxx,#Bxx,#Bxx,#B01,#Bxx,#B01,#Bxx
STIM:VEC 289;COUN 14;DATA:FIEL CSELOT;PATT
#Bxx,#Bxx,#Bxx,#B10,#Bxx,#B10,#Bxx,#Bxx,#Bxx,#Bxx,#B01,#Bxx,#B01,#Bxx
STIM:VEC 303;COUN 14;DATA:FIEL CSELOT;PATT
#Bxx,#Bxx,#Bxx,#B10,#Bxx,#B10,#Bxx,#Bxx,#Bxx,#Bxx,#B01,#Bxx,#B01,#Bxx
STIM:VEC 317;COUN 14;DATA:FIEL CSELOT;PATT
#Bxx,#Bxx,#Bxx,#B10,#Bxx,#B10,#Bxx,#Bxx,#Bxx,#Bxx,#B01,#Bxx,#B01,#Bxx
STIM:VEC 331;COUN 14;DATA:FIEL CSELOT;PATT
```

```
#Bxx,#Bxx,#Bxx,#B10,#Bxx,#B10,#Bxx,#Bxx,#Bxx,#Bxx,#B01,#Bxx,#B01,#Bxx
STIM:VEC 345;COUN 1;DATA:FIEL CSELOT;PATT #Bxx

;Define Board number
STIM:VEC 1;COUN 13;DATA:FIEL BDSELOT;FILL:TYPE REP;INT 1;PATT
#H0;EXEC
STIM:VEC 14;COUN 7;DATA:FIEL BDSELOT;PATT #H1,#H1,#H1,#H1,
#H1,#H1,#H1
STIM:VEC 21;COUN 7;DATA:FIEL BDSELOT;PATT #H2,#H2,#H2,#H2,
#H2,#H2,#H2
STIM:VEC 28;COUN 7;DATA:FIEL BDSELOT;PATT #H3,#H3,#H3,#H3,
#H3,#H3,#H3
STIM:VEC 35;COUN 7;DATA:FIEL BDSELOT;PATT #H4,#H4,#H4,#H4,
#H4,#H4,#H4
STIM:VEC 42;COUN 7;DATA:FIEL BDSELOT;PATT #H5,#H5,#H5,#H5,
#H5,#H5,#H5
STIM:VEC 49;COUN 7;DATA:FIEL BDSELOT;PATT #H6,#H6,#H6,#H6,
#H6,#H6,#H6
STIM:VEC 56;COUN 7;DATA:FIEL BDSELOT;PATT #H7,#H7,#H7,#H7,
#H7,#H7,#H7
STIM:VEC 63;COUN 7;DATA:FIEL BDSELOT;PATT #H0,#H0,#H0,#H0,
#H0,#H0,#H0
STIM:VEC 70;COUN 7;DATA:FIEL BDSELOT;PATT #H1,#H1,#H1,#H1,
#H1,#H1,#H1
STIM:VEC 77;COUN 7;DATA:FIEL BDSELOT;PATT #H2,#H2,#H2,#H2,
#H2,#H2,#H2
STIM:VEC 84;COUN 7;DATA:FIEL BDSELOT;PATT #H3,#H3,#H3,#H3,
#H3,#H3,#H3
STIM:VEC 91;COUN 7;DATA:FIEL BDSELOT;PATT #H4,#H4,#H4,#H4,
#H4,#H4,#H4
STIM:VEC 98;COUN 7;DATA:FIEL BDSELOT;PATT #H5,#H5,#H5,#H5,
#H5,#H5,#H5
STIM:VEC 105;COUN 7;DATA:FIEL BDSELOT;PATT #H6,#H6,#H6,#H6,
#H6,#H6,#H6
STIM:VEC 112;COUN 7;DATA:FIEL BDSELOT;PATT #H7,#H7,#H7,#H7,
#H7,#H7,#H7
STIM:VEC 119;COUN 2;DATA:FIEL BDSELOT;PATT #H0,#H0
STIM:VEC 121;COUN 14;DATA:FIEL BDSELOT;PATT #H0,#H0,#H0,#H0,
#H0,#H0,#H0,#H0,#H0,#H0,#H0,#H0,#H0,#H0
STIM:VEC 135;COUN 14;DATA:FIEL BDSELOT;PATT #H1,#H1,#H1,#H1,
#H1,#H1,#H1,#H1,#H1,#H1,#H1,#H1,#H1,#H1
STIM:VEC 149;COUN 14;DATA:FIEL BDSELOT;PATT #H2,#H2,#H2,#H2,
#H2,#H2,#H2,#H2,#H2,#H2,#H2,#H2,#H2,#H2
STIM:VEC 163;COUN 14;DATA:FIEL BDSELOT;PATT #H3,#H3,#H3,#H3,
#H3,#H3,#H3,#H3,#H3,#H3,#H3,#H3,#H3,#H3
```

```
STIM:VEC 177;COUN 14;DATA:FIEL BDSELOT;PATT #H4,#H4,#H4,#H4,
#H4,#H4,#H4,#H4,#H4,#H4,#H4,#H4,#H4,#H4
STIM:VEC 191;COUN 14;DATA:FIEL BDSELOT;PATT #H5,#H5,#H5,#H5,
#H5,#H5,#H5,#H5,#H5,#H5,#H5,#H5,#H5,#H5
STIM:VEC 205;COUN 14;DATA:FIEL BDSELOT;PATT #H6,#H6,#H6,#H6,
#H6,#H6,#H6,#H6,#H6,#H6,#H6,#H6,#H6,#H6
STIM:VEC 219;COUN 14;DATA:FIEL BDSELOT;PATT #H7,#H7,#H7,#H7,
#H7,#H7,#H7,#H7,#H7,#H7,#H7,#H7,#H7,#H7
STIM:VEC 233;COUN 14;DATA:FIEL BDSELOT;PATT #H0,#H0,#H0,#H0,
#H0,#H0,#H0,#H0,#H0,#H0,#H0,#H0,#H0,#H0
STIM:VEC 247;COUN 14;DATA:FIEL BDSELOT;PATT #H1,#H1,#H1,#H1,
#H1,#H1,#H1,#H1,#H1,#H1,#H1,#H1,#H1,#H1
STIM:VEC 261;COUN 14;DATA:FIEL BDSELOT;PATT #H2,#H2,#H2,#H2,
#H2,#H2,#H2,#H2,#H2,#H2,#H2,#H2,#H2,#H2
STIM:VEC 275;COUN 14;DATA:FIEL BDSELOT;PATT #H3,#H3,#H3,#H3,
#H3,#H3,#H3, #H3,#H3,#H3,#H3,#H3,#H3,#H3
STIM:VEC 289;COUN 14;DATA:FIEL BDSELOT;PATT #H4,#H4,#H4,#H4,
#H4,#H4,#H4,#H4,#H4,#H4,#H4,#H4,#H4,#H4
STIM:VEC 303;COUN 14;DATA:FIEL BDSELOT;PATT #H5,#H5,#H5,#H5,
#H5,#H5,#H5,#H5,#H5,#H5,#H5,#H5,#H5,#H5
STIM:VEC 317;COUN 14;DATA:FIEL BDSELOT;PATT #H6,#H6,#H6,#H6,
#H6,#H6,#H6,#H6,#H6,#H6,#H6,#H6,#H6,#H6
STIM:VEC 331;COUN 14;DATA:FIEL BDSELOT;PATT #H7,#H7,#H7,#H7,
#H7,#H7,#H7,#H7,#H7,#H7,#H7,#H7,#H7,#H7
STIM:VEC 345;COUN 1;DATA:FIEL BDSELOT;PATT #H0

STIM:VEC 1;COUN 7;DATA:FIEL RAMCLKOT;PATT #B0,#B0,#B0,#B0,
#B0,#B0,#B0
STIM:VEC 8;COUN 6;DATA:FIEL RAMCLKOT;PATT #B0,#B0,#B1,#B0, #B1,#B0
STIM:VEC 14;COUN 7;DATA:FIEL RAMCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0
STIM:VEC 21;COUN 7;DATA:FIEL RAMCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0
STIM:VEC 28;COUN 7;DATA:FIEL RAMCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0
STIM:VEC 35;COUN 7;DATA:FIEL RAMCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0
STIM:VEC 42;COUN 7;DATA:FIEL RAMCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0
STIM:VEC 49;COUN 7;DATA:FIEL RAMCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0
STIM:VEC 56;COUN 7;DATA:FIEL RAMCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0
STIM:VEC 63;COUN 7;DATA:FIEL RAMCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0
```

```
STIM:VEC 70;COUN 7;DATA:FIEL RAMCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0
STIM:VEC 77;COUN 7;DATA:FIEL RAMCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0
STIM:VEC 84;COUN 7;DATA:FIEL RAMCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0
STIM:VEC 91;COUN 7;DATA:FIEL RAMCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0
STIM:VEC 98;COUN 7;DATA:FIEL RAMCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0
STIM:VEC 105;COUN 7;DATA:FIEL RAMCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0
STIM:VEC 112;COUN 7;DATA:FIEL RAMCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0
STIM:VEC 119;COUN 2;DATA:FIEL RAMCLKOT;PATT #B0,#B0
STIM:VEC 121;COUN 14;DATA:FIEL RAMCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0,#B0,#B0,#B0,#B1,#B0,#B1,#B0
STIM:VEC 135;COUN 14;DATA:FIEL RAMCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0,#B0,#B0,#B0,#B1,#B0,#B1,#B0
STIM:VEC 149;COUN 14;DATA:FIEL RAMCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0,#B0,#B0,#B0,#B1,#B0,#B1,#B0
STIM:VEC 163;COUN 14;DATA:FIEL RAMCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0,#B0,#B0,#B0,#B1,#B0,#B1,#B0
STIM:VEC 177;COUN 14;DATA:FIEL RAMCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0,#B0,#B0,#B0,#B1,#B0,#B1,#B0
STIM:VEC 191;COUN 14;DATA:FIEL RAMCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0,#B0,#B0,#B0,#B1,#B0,#B1,#B0
STIM:VEC 205;COUN 14;DATA:FIEL RAMCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0,#B0,#B0,#B0,#B1,#B0,#B1,#B0
STIM:VEC 219;COUN 14;DATA:FIEL RAMCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0,#B0,#B0,#B0,#B1,#B0,#B1,#B0
STIM:VEC 233;COUN 14;DATA:FIEL RAMCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0,#B0,#B0,#B0,#B1,#B0,#B1,#B0
STIM:VEC 247;COUN 14;DATA:FIEL RAMCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0,#B0,#B0,#B0,#B1,#B0,#B1,#B0
STIM:VEC 261;COUN 14;DATA:FIEL RAMCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0,#B0,#B0,#B0,#B1,#B0,#B1,#B0
STIM:VEC 275;COUN 14;DATA:FIEL RAMCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0,#B0,#B0,#B0,#B1,#B0,#B1,#B0
STIM:VEC 289;COUN 14;DATA:FIEL RAMCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0,#B0,#B0,#B0,#B1,#B0,#B1,#B0
STIM:VEC 303;COUN 14;DATA:FIEL RAMCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0,#B0,#B0,#B0,#B1,#B0,#B1,#B0
STIM:VEC 317;COUN 14;DATA:FIEL RAMCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0,#B0,#B0,#B0,#B1,#B0,#B1,#B0
```

```
STIM:VEC 331;COUN 14;DATA:FIEL RAMCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0,#B0,#B0,#B0,#B1,#B0,#B1,#B0
STIM:VEC 345;COUN 1;DATA:FIEL RAMCLKOT;PATT #B0

;FF Clock Pattern definition
STIM:VEC 1;COUN 7;DATA:FIEL FFCLKOT;PATT #B0,#B0,#B0,#B0, #B0,#B0,#B0
STIM:VEC 8;COUN 6;DATA:FIEL FFCLKOT;PATT #B0,#B0,#B1,#B0, #B1,#B0
STIM:VEC 14;COUN 7;DATA:FIEL FFCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0
STIM:VEC 21;COUN 7;DATA:FIEL FFCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0
STIM:VEC 28;COUN 7;DATA:FIEL FFCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0
STIM:VEC 35;COUN 7;DATA:FIEL FFCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0
STIM:VEC 42;COUN 7;DATA:FIEL FFCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0
STIM:VEC 49;COUN 7;DATA:FIEL FFCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0
STIM:VEC 56;COUN 7;DATA:FIEL FFCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0
STIM:VEC 63;COUN 7;DATA:FIEL FFCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0
STIM:VEC 70;COUN 7;DATA:FIEL FFCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0
STIM:VEC 77;COUN 7;DATA:FIEL FFCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0
STIM:VEC 84;COUN 7;DATA:FIEL FFCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0
STIM:VEC 91;COUN 7;DATA:FIEL FFCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0
STIM:VEC 98;COUN 7;DATA:FIEL FFCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0
STIM:VEC 105;COUN 7;DATA:FIEL FFCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0
STIM:VEC 112;COUN 7;DATA:FIEL FFCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0
STIM:VEC 119;COUN 2;DATA:FIEL FFCLKOT;PATT #B0,#B0
STIM:VEC 121;COUN 14;DATA:FIEL FFCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0,#B0,#B0,#B0,#B1,#B0,#B1,#B0
STIM:VEC 135;COUN 14;DATA:FIEL FFCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0,#B0,#B0,#B0,#B1,#B0,#B1,#B0
STIM:VEC 149;COUN 14;DATA:FIEL FFCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0,#B0,#B0,#B0,#B1,#B0,#B1,#B0
STIM:VEC 163;COUN 14;DATA:FIEL FFCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0,#B0,#B0,#B0,#B1,#B0,#B1,#B0
STIM:VEC 177;COUN 14;DATA:FIEL FFCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0,#B0,#B0,#B0,#B1,#B0,#B1,#B0
STIM:VEC 191;COUN 14;DATA:FIEL FFCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0,#B0,#B0,#B0,#B1,#B0,#B1,#B0
STIM:VEC 205;COUN 14;DATA:FIEL FFCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0,#B0,#B0,#B0,#B1,#B0,#B1,#B0
STIM:VEC 219;COUN 14;DATA:FIEL FFCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0,#B0,#B0,#B0,#B1,#B0,#B1,#B0
STIM:VEC 233;COUN 14;DATA:FIEL FFCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0,#B0,#B0,#B0,#B1,#B0,#B1,#B0
STIM:VEC 247;COUN 14;DATA:FIEL FFCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0,#B0,#B0,#B0,#B1,#B0,#B1,#B0
STIM:VEC 261;COUN 14;DATA:FIEL FFCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0,#B0,#B0,#B0,#B1,#B0,#B1,#B0
STIM:VEC 275;COUN 14;DATA:FIEL FFCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0,#B0,#B0,#B0,#B1,#B0,#B1,#B0
STIM:VEC 289;COUN 14;DATA:FIEL FFCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0,#B0,#B0,#B0,#B1,#B0,#B1,#B0
STIM:VEC 303;COUN 14;DATA:FIEL FFCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0,#B0,#B0,#B0,#B1,#B0,#B1,#B0
STIM:VEC 317;COUN 14;DATA:FIEL FFCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0,#B0,#B0,#B0,#B1,#B0,#B1,#B0
STIM:VEC 331;COUN 14;DATA:FIEL FFCLKOT;PATT #B0,#B0,#B0,#B1,
#B0,#B1,#B0,#B0,#B0,#B0,#B1,#B0,#B1,#B0
STIM:VEC 345;COUN 1;DATA:FIEL FFCLKOT;PATT #B0
STIM:VEC 1;COUN 6;DATA:FIEL WEBOT;PATT #B1,#B1,#B1,#B1, #B1,#B1
STIM:VEC 7;COUN 14;DATA:FIEL WEBOT;PATT #B1,#B1,#B1,#B0,#B1,#B0,#B1,
#B1,#B1,#B1,#B0,#B1,#B0,#B1
STIM:VEC 21;COUN 14;DATA:FIEL WEBOT;PATT #B1,#B1,#B1,#B0,#B1,#B0,#B1,
#B1,#B1,#B1,#B0,#B1,#B0,#B1
STIM:VEC 35;COUN 14;DATA:FIEL WEBOT;PATT #B1,#B1,#B1,#B0,#B1,#B0,#B1,
#B1,#B1,#B1,#B0,#B1,#B0,#B1
STIM:VEC 49;COUN 14;DATA:FIEL WEBOT;PATT #B1,#B1,#B1,#B0,#B1,#B0,#B1,
#B1,#B1,#B1,#B0,#B1,#B0,#B1
STIM:VEC 63;COUN 14;DATA:FIEL WEBOT;PATT #B1,#B1,#B1,#B0,#B1,#B0,#B1,
#B1,#B1,#B1,#B0,#B1,#B0,#B1
STIM:VEC 77;COUN 14;DATA:FIEL WEBOT;PATT #B1,#B1,#B1,#B0,#B1,#B0,#B1,
#B1,#B1,#B1,#B0,#B1,#B0,#B1
STIM:VEC 91;COUN 14;DATA:FIEL WEBOT;PATT #B1,#B1,#B1,#B0,#B1,#B0,#B1,
#B1,#B1,#B1,#B0,#B1,#B0,#B1
STIM:VEC 105;COUN 14;DATA:FIEL WEBOT;PATT
#B1,#B1,#B1,#B0,#B1,#B0,#B1, #B1,#B1,#B1,#B0,#B1,#B0,#B1
STIM:VEC 119;COUN 2;DATA:FIEL WEBOT;PATT #B1,#B1
STIM:VEC 121;COUN 225;DATA:FIEL WEBOT;FILL:TYPE REP;INT 1;PATT
```

```
#B1;EXEC

;Reference data for the algorithmic address field
REC:VEC 8;COUN 6;AMAC:FIEL ADDRE;PATT HOLDA, HOLDA, INC, HOLDA, INC,
HOLDA
REC:VEC 15;COUN 6;AMAC:FIEL ADDRE;PATT HOLDA, HOLDA, INC, HOLDA, INC,
HOLDA
REC:VEC 22;COUN 6;AMAC:FIEL ADDRE;PATT HOLDA, HOLDA, INC, HOLDA, INC,
HOLDA
REC:VEC 29;COUN 6;AMAC:FIEL ADDRE;PATT HOLDA, HOLDA, INC, HOLDA, INC,
HOLDA
REC:VEC 36;COUN 6;AMAC:FIEL ADDRE;PATT HOLDA, HOLDA, INC, HOLDA, INC,
HOLDA
REC:VEC 43;COUN 6;AMAC:FIEL ADDRE;PATT HOLDA, HOLDA, INC, HOLDA, INC,
HOLDA
REC:VEC 50;COUN 6;AMAC:FIEL ADDRE;PATT HOLDA, HOLDA, INC, HOLDA, INC,
HOLDA
REC:VEC 57;COUN 6;AMAC:FIEL ADDRE;PATT HOLDA, HOLDA, INC, HOLDA, INC,
HOLDA
REC:VEC 63;COUN 1;AMAC:FIEL ADDRE;PATT NONA
REC:VEC 64;COUN 6;AMAC:FIEL ADDRE;PATT HOLDA, HOLDA, INC, HOLDA, INC,
HOLDA
REC:VEC 71;COUN 6;AMAC:FIEL ADDRE;PATT HOLDA, HOLDA, INC, HOLDA, INC,
HOLDA
REC:VEC 78;COUN 6;AMAC:FIEL ADDRE;PATT HOLDA, HOLDA, INC, HOLDA, INC,
HOLDA
REC:VEC 85;COUN 6;AMAC:FIEL ADDRE;PATT HOLDA, HOLDA, INC, HOLDA, INC,
HOLDA
REC:VEC 92;COUN 6;AMAC:FIEL ADDRE;PATT HOLDA, HOLDA, INC, HOLDA, INC,
HOLDA
REC:VEC 99;COUN 6;AMAC:FIEL ADDRE;PATT HOLDA, HOLDA, INC, HOLDA, INC,
HOLDA
REC:VEC 106;COUN 6;AMAC:FIEL ADDRE;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA
REC:VEC 113;COUN 6;AMAC:FIEL ADDRE;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA
REC:VEC 120;COUN 1;AMAC:FIEL ADDRE;PATT HOLDD
REC:VEC 122;COUN 13;AMAC:FIEL ADDRE;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA, NONA, HOLDA, HOLDA, INC, HOLDA, INC, HOLDA
REC:VEC 136;COUN 13;AMAC:FIEL ADDRE;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA, NONA, HOLDA, HOLDA, INC, HOLDA, INC, HOLDA
REC:VEC 150;COUN 13;AMAC:FIEL ADDRE;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA, NONA, HOLDA, HOLDA, INC, HOLDA, INC, HOLDA
REC:VEC 164;COUN 13;AMAC:FIEL ADDRE;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA, NONA, HOLDA, HOLDA, INC, HOLDA, INC, HOLDA
```

```
REC:VEC 178;COUN 13;AMAC:FIEL ADDRE;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA, NONA, HOLDA, HOLDA, INC, HOLDA, INC, HOLDA
REC:VEC 192;COUN 13;AMAC:FIEL ADDRE;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA, NONA, HOLDA, HOLDA, INC, HOLDA, INC, HOLDA
REC:VEC 206;COUN 13;AMAC:FIEL ADDRE;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA, NONA, HOLDA, HOLDA, INC, HOLDA, INC, HOLDA
REC:VEC 220;COUN 13;AMAC:FIEL ADDRE;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA, NONA, HOLDA, HOLDA, INC, HOLDA, INC, HOLDA
REC:VEC 234;COUN 13;AMAC:FIEL ADDRE;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA, NONA, HOLDA, HOLDA, INC, HOLDA, INC, HOLDA
REC:VEC 248;COUN 13;AMAC:FIEL ADDRE;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA, NONA, HOLDA, HOLDA, INC, HOLDA, INC, HOLDA
REC:VEC 262;COUN 13;AMAC:FIEL ADDRE;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA, NONA, HOLDA, HOLDA, INC, HOLDA, INC, HOLDA
REC:VEC 276;COUN 13;AMAC:FIEL ADDRE;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA, NONA, HOLDA, HOLDA, INC, HOLDA, INC, HOLDA
REC:VEC 290;COUN 13;AMAC:FIEL ADDRE;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA, NONA, HOLDA, HOLDA, INC, HOLDA, INC, HOLDA
REC:VEC 304;COUN 13;AMAC:FIEL ADDRE;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA, NONA, HOLDA, HOLDA, INC, HOLDA, INC, HOLDA
REC:VEC 318;COUN 13;AMAC:FIEL ADDRE;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA, NONA, HOLDA, HOLDA, INC, HOLDA, INC, HOLDA
REC:VEC 332;COUN 13;AMAC:FIEL ADDRE;PATT HOLDA, HOLDA, INC, HOLDA,
INC, HOLDA, NONA, HOLDA, HOLDA, INC, HOLDA, INC, HOLDA
REC:VEC 345;COUN 1;AMAC:FIEL ADDRE;PATT HOLDD

;Reference data for the algorithmic data field. Refer to the comment
of STIM:—————:FIEL DATAO;——.
REC:VEC 8;COUN 6;AMAC:FIEL DATAE;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD
REC:VEC 15;COUN 6;AMAC:FIEL DATAE;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD
REC:VEC 22;COUN 6;AMAC:FIEL DATAE;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD
REC:VEC 29;COUN 6;AMAC:FIEL DATAE;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD
REC:VEC 36;COUN 6;AMAC:FIEL DATAE;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD
REC:VEC 43;COUN 6;AMAC:FIEL DATAE;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD
REC:VEC 50;COUN 6;AMAC:FIEL DATAE;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD
REC:VEC 57;COUN 6;AMAC:FIEL DATAE;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD
```

REC:VEC 64;COUN 6;AMAC:FIEL DATAE;PATT HOLDD, NONA, XOR, NONA, XOR,

```
HOLDD
REC:VEC 71;COUN 6;AMAC:FIEL DATAE;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD
REC:VEC 78;COUN 6;AMAC:FIEL DATAE;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD
REC:VEC 85;COUN 6;AMAC:FIEL DATAE;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD
REC:VEC 92;COUN 6;AMAC:FIEL DATAE;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD
REC:VEC 99;COUN 6;AMAC:FIEL DATAE;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD
REC:VEC 106;COUN 6;AMAC:FIEL DATAE;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD
REC:VEC 113;COUN 6;AMAC:FIEL DATAE;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD
REC:VEC 120;COUN 1;AMAC:FIEL DATAE;PATT HOLDD
REC:VEC 122;COUN 13;AMAC:FIEL DATAE;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD, NONA, HOLDD, NONA, XOR, NONA, XOR, HOLDD
REC:VEC 136;COUN 13;AMAC:FIEL DATAE;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD, NONA, HOLDD, NONA, XOR, NONA, XOR, HOLDD
REC:VEC 150;COUN 13;AMAC:FIEL DATAE;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD, NONA, HOLDD, NONA, XOR, NONA, XOR, HOLDD
REC:VEC 164;COUN 13;AMAC:FIEL DATAE;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD, NONA, HOLDD, NONA, XOR, NONA, XOR, HOLDD
REC:VEC 178;COUN 13;AMAC:FIEL DATAE;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD, NONA, HOLDD, NONA, XOR, NONA, XOR, HOLDD
REC:VEC 192;COUN 13;AMAC:FIEL DATAE;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD, NONA, HOLDD, NONA, XOR, NONA, XOR, HOLDD
REC:VEC 206;COUN 13;AMAC:FIEL DATAE;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD, NONA, HOLDD, NONA, XOR, NONA, XOR, HOLDD
REC:VEC 220;COUN 13;AMAC:FIEL DATAE;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD, NONA, HOLDD, NONA, XOR, NONA, XOR, HOLDD
REC:VEC 234;COUN 13;AMAC:FIEL DATAE;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD, NONA, HOLDD, NONA, XOR, NONA, XOR, HOLDD
REC:VEC 248;COUN 13;AMAC:FIEL DATAE;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD, NONA, HOLDD, NONA, XOR, NONA, XOR, HOLDD
REC:VEC 262;COUN 13;AMAC:FIEL DATAE;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD, NONA, HOLDD, NONA, XOR, NONA, XOR, HOLDD
REC:VEC 276;COUN 13;AMAC:FIEL DATAE;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD, NONA, HOLDD, NONA, XOR, NONA, XOR, HOLDD
REC:VEC 290;COUN 13;AMAC:FIEL DATAE;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD, NONA, HOLDD, NONA, XOR, NONA, XOR, HOLDD
REC:VEC 304;COUN 13;AMAC:FIEL DATAE;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD, NONA, HOLDD, NONA, XOR, NONA, XOR, HOLDD
REC:VEC 318;COUN 13;AMAC:FIEL DATAE;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD, NONA, HOLDD, NONA, XOR, NONA, XOR, HOLDD
REC:VEC 332;COUN 13;AMAC:FIEL DATAE;PATT HOLDD, NONA, XOR, NONA, XOR,
HOLDD, NONA, HOLDD, NONA, XOR, NONA, XOR, HOLDD
REC:VEC 345;COUN 1;AMAC:FIEL DATAE;PATT HOLDD

;Define address Reference data source (counter initial value)
REC:VEC 1;COUN 8;DATA:FIEL ADDRED;PATT
#Hxxxx,#hxxxx,#hxxxx,#hxxxx,#hxxxx,#hxxxx,#hxxxx,#hxxxx
REC:VEC 7;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 10;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 12;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 14;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 17;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 19;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 21;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 24;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 26;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 28;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 31;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 33;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 35;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 38;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 40;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 42;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 45;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 47;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 49;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 52;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 54;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 56;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 59;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 61;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 63;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 66;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 68;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 70;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 73;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 75;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 77;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 80;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 82;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 84;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 87;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 89;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 91;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
```

```
REC:VEC 94;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 96;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 98;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 101;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 103;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 105;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 108;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 110;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 112;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 115;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 117;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 119;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 121;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 124;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 126;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 128;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 131;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 133;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 135;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 138;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 140;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 142;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 145;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 147;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 149;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 152;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 154;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 156;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 159;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 161;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 163;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 166;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 168;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 170;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 173;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 175;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 177;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 180;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 182;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 184;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 187;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 189;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 191;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 194;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 196;COUN 1;DATA:FIEL ADDRED;PATT #H0000
```

```
REC:VEC 198;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 201;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 203;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 205;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 208;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 210;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 212;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 215;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 217;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 219;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 222;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 224;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 226;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 229;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 231;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 233;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 236;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 238;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 240;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 243;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 245;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 247;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 250;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 252;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 254;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 257;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 259;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 261;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 264;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 266;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 268;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 271;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 273;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 275;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 278;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 280;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 282;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 285;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 287;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 289;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 292;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 294;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 296;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 299;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 301;COUN 1;DATA:FIEL ADDRED;PATT #H0000
```

```
REC:VEC 303;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 306;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 308;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 310;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 313;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 315;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 317;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 320;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 322;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 324;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 327;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 329;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 331;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 334;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 336;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 338;COUN 1;DATA:FIEL ADDRED;PATT #HFFFF
REC:VEC 341;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 343;COUN 1;DATA:FIEL ADDRED;PATT #H0000
REC:VEC 345;COUN 1;DATA:FIEL ADDRED;PATT #H0000

;Define data source (counter initial value)
REC:VEC 123;COUN 4;DATA:FIEL DATAED;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
REC:VEC 128;COUN 1;DATA:FIEL DATAED;PATT #HFFFFFF
REC:VEC 130;COUN 4;DATA:FIEL DATAED;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
REC:VEC 135;COUN 1;DATA:FIEL DATAED;PATT #HFFFFFF
REC:VEC 137;COUN 4;DATA:FIEL DATAED;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
REC:VEC 142;COUN 1;DATA:FIEL DATAED;PATT #HFFFFFF
REC:VEC 144;COUN 4;DATA:FIEL DATAED;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
REC:VEC 149;COUN 1;DATA:FIEL DATAED;PATT #HFFFFFF
REC:VEC 151;COUN 4;DATA:FIEL DATAED;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
REC:VEC 156;COUN 1;DATA:FIEL DATAED;PATT #HFFFFFF
REC:VEC 158;COUN 4;DATA:FIEL DATAED;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
REC:VEC 163;COUN 1;DATA:FIEL DATAED;PATT #HFFFFFF
REC:VEC 165;COUN 4;DATA:FIEL DATAED;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF;
REC:VEC 170;COUN 1;DATA:FIEL DATAED;PATT #HFFFFFF
REC:VEC 172;COUN 4;DATA:FIEL DATAED;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
REC:VEC 177;COUN 1;DATA:FIEL DATAED;PATT #HFFFFFF
```

```
REC:VEC 179;COUN 4;DATA:FIEL DATAED;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
REC:VEC 184;COUN 1;DATA:FIEL DATAED;PATT #HFFFFFF
REC:VEC 186;COUN 4;DATA:FIEL DATAED;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
REC:VEC 191;COUN 1;DATA:FIEL DATAED;PATT #HFFFFFF
REC:VEC 193;COUN 4;DATA:FIEL DATAED;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF;
REC:VEC 198;COUN 1;DATA:FIEL DATAED;PATT #HFFFFFF
REC:VEC 200;COUN 4;DATA:FIEL DATAED;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
REC:VEC 205;COUN 1;DATA:FIEL DATAED;PATT #HFFFFFF
REC:VEC 207;COUN 4;DATA:FIEL DATAED;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
REC:VEC 212;COUN 1;DATA:FIEL DATAED;PATT #HFFFFFF
REC:VEC 214;COUN 4;DATA:FIEL DATAED;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
REC:VEC 219;COUN 1;DATA:FIEL DATAED;PATT #HFFFFFF
REC:VEC 221;COUN 4;DATA:FIEL DATAED;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
REC:VEC 226;COUN 1;DATA:FIEL DATAED;PATT #HFFFFFF
REC:VEC 228;COUN 4;DATA:FIEL DATAED;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
REC:VEC 233;COUN 1;DATA:FIEL DATAED;PATT #HFFFFFF
REC:VEC 235;COUN 4;DATA:FIEL DATAED;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
REC:VEC 240;COUN 1;DATA:FIEL DATAED;PATT #HFFFFFF
REC:VEC 242;COUN 4;DATA:FIEL DATAED;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
REC:VEC 247;COUN 1;DATA:FIEL DATAED;PATT #HFFFFFF
REC:VEC 249;COUN 4;DATA:FIEL DATAED;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
REC:VEC 254;COUN 1;DATA:FIEL DATAED;PATT #HFFFFFF
REC:VEC 256;COUN 4;DATA:FIEL DATAED;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
REC:VEC 261;COUN 1;DATA:FIEL DATAED;PATT #HFFFFFF
REC:VEC 263;COUN 4;DATA:FIEL DATAED;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
REC:VEC 268;COUN 1;DATA:FIEL DATAED;PATT #HFFFFFF
REC:VEC 270;COUN 4;DATA:FIEL DATAED;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
REC:VEC 275;COUN 1;DATA:FIEL DATAED;PATT #HFFFFFF
REC:VEC 277;COUN 4;DATA:FIEL DATAED;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
REC:VEC 282;COUN 1;DATA:FIEL DATAED;PATT #HFFFFFF
```

```
REC:VEC 284;COUN 4;DATA:FIEL DATAED;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
REC:VEC 289;COUN 1;DATA:FIEL DATAED;PATT #HFFFFFF
REC:VEC 291;COUN 4;DATA:FIEL DATAED;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
REC:VEC 296;COUN 1;DATA:FIEL DATAED;PATT #HFFFFFF
REC:VEC 298;COUN 4;DATA:FIEL DATAED;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
REC:VEC 303;COUN 1;DATA:FIEL DATAED;PATT #HFFFFFF
REC:VEC 305;COUN 4;DATA:FIEL DATAED;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
REC:VEC 310;COUN 1;DATA:FIEL DATAED;PATT #HFFFFFF
REC:VEC 312;COUN 4;DATA:FIEL DATAED;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
REC:VEC 317;COUN 1;DATA:FIEL DATAED;PATT #HFFFFFF
REC:VEC 319;COUN 4;DATA:FIEL DATAED;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
REC:VEC 324;COUN 1;DATA:FIEL DATAED;PATT #HFFFFFF
REC:VEC 326;COUN 4;DATA:FIEL DATAED;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
REC:VEC 331;COUN 1;DATA:FIEL DATAED;PATT #HFFFFFF
REC:VEC 333;COUN 4;DATA:FIEL DATAED;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF
REC:VEC 338;COUN 1;DATA:FIEL DATAED;PATT #HFFFFFF
REC:VEC 340;COUN 4;DATA:FIEL DATAED;PATT
#H015555,#H03FFFF,#H02AAAA,#H03FFFF

;Following lines are to define DATAD field.
REC:VEC 121;COUN 6;DATA:FIEL DATAD;PATT
#HFFFFFF,#HFFFFFF,#HFFFFFF,#HFC0000,#HFFFFFF,#HFC0000
REC:VEC 128;COUN 1;DATA:FIEL DATAD;PATT #HFFFFFF
REC:VEC 130;COUN 4;DATA:FIEL DATAD;PATT
#HFFFFFF,#HFC0000,#HFFFFFF,#HFC0000
REC:VEC 135;COUN 6;DATA:FIEL DATAD;PATT
#HFFFFFF,#HFFFFFF,#HFFFFFF,#HFC0000,#HFFFFFF,#HFC0000
REC:VEC 142;COUN 1;DATA:FIEL DATAD;PATT #HFFFFFF
REC:VEC 144;COUN 4;DATA:FIEL DATAD;PATT
#HFFFFFF,#HFC0000,#HFFFFFF,#HFC0000
REC:VEC 149;COUN 6;DATA:FIEL DATAD;PATT
#HFFFFFF,#HFFFFFF,#HFFFFFF,#HFC0000,#HFFFFFF,#HFC0000
REC:VEC 156;COUN 1;DATA:FIEL DATAD;PATT #HFFFFFF
REC:VEC 158;COUN 4;DATA:FIEL DATAD;PATT
#HFFFFFF,#HFC0000,#HFFFFFF,#HFC0000
REC:VEC 163;COUN 6;DATA:FIEL DATAD;PATT
#HFFFFFF,#HFFFFFF,#HFFFFFF,#HFC0000,#HFFFFFF,#HFC0000
```

```
REC:VEC 170;COUN 1;DATA:FIEL DATAD;PATT #HFFFFFF
REC:VEC 172;COUN 4;DATA:FIEL DATAD;PATT
#HFFFFFF,#HFC0000,#HFFFFFF,#HFC0000
REC:VEC 177;COUN 6;DATA:FIEL DATAD;PATT
#HFFFFFF,#HFFFFFF,#HFFFFFF,#HFC0000,#HFFFFFF,#HFC0000
REC:VEC 184;COUN 1;DATA:FIEL DATAD;PATT #HFFFFFF
REC:VEC 186;COUN 4;DATA:FIEL DATAD;PATT
#HFFFFFF,#HFC0000,#HFFFFFF,#HFC0000
REC:VEC 191;COUN 6;DATA:FIEL DATAD;PATT
#HFFFFFF,#HFFFFFF,#HFFFFFF,#HFC0000,#HFFFFFF,#HFC0000
REC:VEC 198;COUN 1;DATA:FIEL DATAD;PATT #HFFFFFF
REC:VEC 200;COUN 4;DATA:FIEL DATAD;PATT
#HFFFFFF,#HFC0000,#HFFFFFF,#HFC0000
REC:VEC 205;COUN 6;DATA:FIEL DATAD;PATT
#HFFFFFF,#HFFFFFF,#HFFFFFF,#HFC0000,#HFFFFFF,#HFC0000
REC:VEC 212;COUN 1;DATA:FIEL DATAD;PATT #HFFFFFF
REC:VEC 214;COUN 4;DATA:FIEL DATAD;PATT
#HFFFFFF,#HFC0000,#HFFFFFF,#HFC0000
REC:VEC 219;COUN 6;DATA:FIEL DATAD;PATT
#HFFFFFF,#HFFFFFF,#HFFFFFF,#HFC0000,#HFFFFFF,#HFC0000
REC:VEC 226;COUN 1;DATA:FIEL DATAD;PATT #HFFFFFF
REC:VEC 228;COUN 4;DATA:FIEL DATAD;PATT
#HFFFFFF,#HFC0000,#HFFFFFF,#HFC0000
REC:VEC 233;COUN 6;DATA:FIEL DATAD;PATT
#HFFFFFF,#HFFFFFF,#HFFFFFF,#HFC0000,#HFFFFFF,#HFC0000
REC:VEC 240;COUN 1;DATA:FIEL DATAD;PATT #HFFFFFF
REC:VEC 242;COUN 4;DATA:FIEL DATAD;PATT
#HFFFFFF,#HFC0000,#HFFFFFF,#HFC0000
REC:VEC 247;COUN 6;DATA:FIEL DATAD;PATT
#HFFFFFF,#HFFFFFF,#HFFFFFF,#HFC0000,#HFFFFFF,#HFC0000
REC:VEC 254;COUN 1;DATA:FIEL DATAD;PATT #HFFFFFF
REC:VEC 256;COUN 4;DATA:FIEL DATAD;PATT
#HFFFFFF,#HFC0000,#HFFFFFF,#HFC0000
REC:VEC 261;COUN 6;DATA:FIEL DATAD;PATT
#HFFFFFF,#HFFFFFF,#HFFFFFF,#HFC0000,#HFFFFFF,#HFC0000
REC:VEC 268;COUN 1;DATA:FIEL DATAD;PATT #HFFFFFF
REC:VEC 270;COUN 4;DATA:FIEL DATAD;PATT
#HFFFFFF,#HFC0000,#HFFFFFF,#HFC0000
REC:VEC 275;COUN 6;DATA:FIEL DATAD;PATT
#HFFFFFF,#HFFFFFF,#HFFFFFF,#HFC0000,#HFFFFFF,#HFC0000
REC:VEC 282;COUN 1;DATA:FIEL DATAD;PATT #HFFFFFF
REC:VEC 284;COUN 4;DATA:FIEL DATAD;PATT
#HFFFFFF,#HFC0000,#HFFFFFF,#HFC0000
REC:VEC 289;COUN 6;DATA:FIEL DATAD;PATT
#HFFFFFF,#HFFFFFF,#HFFFFFF,#HFC0000,#HFFFFFF,#HFC0000
```

```
REC:VEC 296;COUN 1;DATA:FIEL DATAD;PATT #HFFFFFF
REC:VEC 298;COUN 4;DATA:FIEL DATAD;PATT
#HFFFFFF,#HFC0000,#HFFFFFF,#HFC0000
REC:VEC 303;COUN 6;DATA:FIEL DATAD;PATT
#HFFFFFF,#HFFFFFF,#HFFFFFF,#HFC0000,#HFFFFFF,#HFC0000
REC:VEC 310;COUN 1;DATA:FIEL DATAD;PATT #HFFFFFF
REC:VEC 312;COUN 4;DATA:FIEL DATAD;PATT
#HFFFFFF,#HFC0000,#HFFFFFF,#HFC0000
REC:VEC 317;COUN 6;DATA:FIEL DATAD;PATT
#HFFFFFF,#HFFFFFF,#HFFFFFF,#HFC0000,#HFFFFFF,#HFC0000
REC:VEC 324;COUN 1;DATA:FIEL DATAD;PATT #HFFFFFF
REC:VEC 326;COUN 4;DATA:FIEL DATAD;PATT
#HFFFFFF,#HFC0000,#HFFFFFF,#HFC0000
REC:VEC 331;COUN 6;DATA:FIEL DATAD;PATT
#HFFFFFF,#HFFFFFF,#HFFFFFF,#HFC0000,#HFFFFFF,#HFC0000
REC:VEC 338;COUN 1;DATA:FIEL DATAD;PATT #HFFFFFF
REC:VEC 340;COUN 4;DATA:FIEL DATAD;PATT
#HFFFFFF,#HFC0000,#HFFFFFF,#HFC0000

;Following lines are to define OSELED field.
REC:VEC 1;COUN 120;DATA:FIEL OSELED;FILL:TYPE REP;INT 1;PATT #B1;EXEC
REC:VEC 121;COUN 1;DATA:FIEL OSELED;PATT #B1
REC:VEC 124;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 126;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 128;COUN 1;DATA:FIEL OSELED;PATT #B1
REC:VEC 131;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 133;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 135;COUN 1;DATA:FIEL OSELED;PATT #B1
REC:VEC 138;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 140;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 142;COUN 1;DATA:FIEL OSELED;PATT #B1
REC:VEC 145;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 147;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 149;COUN 1;DATA:FIEL OSELED;PATT #B1
REC:VEC 152;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 154;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 156;COUN 1;DATA:FIEL OSELED;PATT #B1
REC:VEC 159;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 161;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 163;COUN 1;DATA:FIEL OSELED;PATT #B1
REC:VEC 166;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 168;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 170;COUN 1;DATA:FIEL OSELED;PATT #B1
REC:VEC 173;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 175;COUN 1;DATA:FIEL OSELED;PATT #B0
```

```
REC:VEC 177;COUN 1;DATA:FIEL OSELED;PATT #B1
REC:VEC 180;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 182;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 184;COUN 1;DATA:FIEL OSELED;PATT #B1
REC:VEC 187;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 189;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 191;COUN 1;DATA:FIEL OSELED;PATT #B1
REC:VEC 194;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 196;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 198;COUN 1;DATA:FIEL OSELED;PATT #B1
REC:VEC 201;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 203;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 205;COUN 1;DATA:FIEL OSELED;PATT #B1
REC:VEC 208;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 210;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 212;COUN 1;DATA:FIEL OSELED;PATT #B1
REC:VEC 215;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 217;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 219;COUN 1;DATA:FIEL OSELED;PATT #B1
REC:VEC 222;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 224;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 226;COUN 1;DATA:FIEL OSELED;PATT #B1
REC:VEC 229;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 231;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 233;COUN 1;DATA:FIEL OSELED;PATT #B1
REC:VEC 236;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 238;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 240;COUN 1;DATA:FIEL OSELED;PATT #B1
REC:VEC 243;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 245;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 247;COUN 1;DATA:FIEL OSELED;PATT #B1
REC:VEC 250;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 252;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 254;COUN 1;DATA:FIEL OSELED;PATT #B1
REC:VEC 257;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 259;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 261;COUN 1;DATA:FIEL OSELED;PATT #B1
REC:VEC 264;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 266;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 268;COUN 1;DATA:FIEL OSELED;PATT #B1
REC:VEC 271;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 273;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 275;COUN 1;DATA:FIEL OSELED;PATT #B1
REC:VEC 278;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 280;COUN 1;DATA:FIEL OSELED;PATT #B0
```

```
REC:VEC 282;COUN 1;DATA:FIEL OSELED;PATT #B1
REC:VEC 285;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 287;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 289;COUN 1;DATA:FIEL OSELED;PATT #B1
REC:VEC 292;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 294;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 296;COUN 1;DATA:FIEL OSELED;PATT #B1
REC:VEC 299;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 301;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 303;COUN 1;DATA:FIEL OSELED;PATT #B1
REC:VEC 306;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 308;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 310;COUN 1;DATA:FIEL OSELED;PATT #B1
REC:VEC 313;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 315;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 317;COUN 1;DATA:FIEL OSELED;PATT #B1
REC:VEC 320;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 322;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 324;COUN 1;DATA:FIEL OSELED;PATT #B1
REC:VEC 327;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 329;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 331;COUN 1;DATA:FIEL OSELED;PATT #B1
REC:VEC 334;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 336;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 338;COUN 1;DATA:FIEL OSELED;PATT #B1
REC:VEC 341;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 343;COUN 1;DATA:FIEL OSELED;PATT #B0
REC:VEC 345;COUN 1;DATA:FIEL OSELED;PATT #B0

;Define CSELED field

REC:VEC 8;COUN 112;DATA:FIEL CSELED;FILL:TYPE REP;INT 1;PATT
#B00;EXEC
REC:VEC 120;COUN 1;DATA:FIEL CSELED;PATT #BXX
REC:VEC 121;COUN 14;DATA:FIEL CSELED;PATT
#Bxx,#Bxx,#Bxx,#B10,#Bxx,#B10,#Bxx,#Bxx,#Bxx,#Bxx,#B01,#Bxx,#B01,#Bxx
REC:VEC 135;COUN 14;DATA:FIEL CSELED;PATT
#Bxx,#Bxx,#Bxx,#B10,#Bxx,#B10,#Bxx,#Bxx,#Bxx,#Bxx,#B01,#Bxx,#B01,#Bxx
REC:VEC 149;COUN 14;DATA:FIEL CSELED;PATT
#Bxx,#Bxx,#Bxx,#B10,#Bxx,#B10,#Bxx,#Bxx,#Bxx,#Bxx,#B01,#Bxx,#B01,#Bxx
REC:VEC 163;COUN 14;DATA:FIEL CSELED;PATT
#Bxx,#Bxx,#Bxx,#B10,#Bxx,#B10,#Bxx,#Bxx,#Bxx,#Bxx,#B01,#Bxx,#B01,#Bxx
REC:VEC 177;COUN 14;DATA:FIEL CSELED;PATT
#Bxx,#Bxx,#Bxx,#B10,#Bxx,#B10,#Bxx,#Bxx,#Bxx,#Bxx,#B01,#Bxx,#B01,#Bxx
REC:VEC 191;COUN 14;DATA:FIEL CSELED;PATT
```

```
#Bxx,#Bxx,#Bxx,#B10,#Bxx,#B10,#Bxx,#Bxx,#Bxx,#Bxx,#B01,#Bxx,#B01,#Bxx
REC:VEC 205;COUN 14;DATA:FIEL CSELED;PATT
#Bxx,#Bxx,#Bxx,#B10,#Bxx,#B10,#Bxx,#Bxx,#Bxx,#Bxx,#B01,#Bxx,#B01,#Bxx
REC:VEC 219;COUN 14;DATA:FIEL CSELED;PATT
#Bxx,#Bxx,#Bxx,#B10,#Bxx,#B10,#Bxx,#Bxx,#Bxx,#Bxx,#B01,#Bxx,#B01,#Bxx
REC:VEC 233;COUN 14;DATA:FIEL CSELED;PATT
#Bxx,#Bxx,#Bxx,#B10,#Bxx,#B10,#Bxx,#Bxx,#Bxx,#Bxx,#B01,#Bxx,#B01,#Bxx
REC:VEC 247;COUN 14;DATA:FIEL CSELED;PATT
#Bxx,#Bxx,#Bxx,#B10,#Bxx,#B10,#Bxx,#Bxx,#Bxx,#Bxx,#B01,#Bxx,#B01,#Bxx
REC:VEC 261;COUN 14;DATA:FIEL CSELED;PATT
#Bxx,#Bxx,#Bxx,#B10,#Bxx,#B10,#Bxx,#Bxx,#Bxx,#Bxx,#B01,#Bxx,#B01,#Bxx
REC:VEC 275;COUN 14;DATA:FIEL CSELED;PATT
#Bxx,#Bxx,#Bxx,#B10,#Bxx,#B10,#Bxx,#Bxx,#Bxx,#Bxx,#B01,#Bxx,#B01,#Bxx
REC:VEC 289;COUN 14;DATA:FIEL CSELED;PATT
#Bxx,#Bxx,#Bxx,#B10,#Bxx,#B10,#Bxx,#Bxx,#Bxx,#Bxx,#B01,#Bxx,#B01,#Bxx
REC:VEC 303;COUN 14;DATA:FIEL CSELED;PATT
#Bxx,#Bxx,#Bxx,#B10,#Bxx,#B10,#Bxx,#Bxx,#Bxx,#Bxx,#B01,#Bxx,#B01,#Bxx
REC:VEC 317;COUN 14;DATA:FIEL CSELED;PATT
#Bxx,#Bxx,#Bxx,#B10,#Bxx,#B10,#Bxx,#Bxx,#Bxx,#Bxx,#B01,#Bxx,#B01,#Bxx
REC:VEC 331;COUN 14;DATA:FIEL CSELED;PATT
#Bxx,#Bxx,#Bxx,#B10,#Bxx,#B10,#Bxx,#Bxx,#Bxx,#Bxx,#B01,#Bxx,#B01,#Bxx
REC:VEC 345;COUN 1;DATA:FIEL CSELED;PATT #Bxx

;Define BDSELED field.
REC:VEC 1;COUN 13;DATA:FIEL BDSELED;FILL:TYPE REP;INT 1;PATT #H0;EXEC
REC:VEC 14;COUN 7;DATA:FIEL BDSELED;PATT #H1,#H1,#H1,#H1, #H1,#H1,#H1
REC:VEC 21;COUN 7;DATA:FIEL BDSELED;PATT #H2,#H2,#H2,#H2, #H2,#H2,#H2
REC:VEC 28;COUN 7;DATA:FIEL BDSELED;PATT #H3,#H3,#H3,#H3, #H3,#H3,#H3
REC:VEC 35;COUN 7;DATA:FIEL BDSELED;PATT #H4,#H4,#H4,#H4, #H4,#H4,#H4
REC:VEC 42;COUN 7;DATA:FIEL BDSELED;PATT #H5,#H5,#H5,#H5, #H5,#H5,#H5
REC:VEC 49;COUN 7;DATA:FIEL BDSELED;PATT #H6,#H6,#H6,#H6, #H6,#H6,#H6
REC:VEC 56;COUN 7;DATA:FIEL BDSELED;PATT #H7,#H7,#H7,#H7, #H7,#H7,#H7
REC:VEC 63;COUN 7;DATA:FIEL BDSELED;PATT #H0,#H0,#H0,#H0, #H0,#H0,#H0
REC:VEC 70;COUN 7;DATA:FIEL BDSELED;PATT #H1,#H1,#H1,#H1, #H1,#H1,#H1
REC:VEC 77;COUN 7;DATA:FIEL BDSELED;PATT #H2,#H2,#H2,#H2, #H2,#H2,#H2
REC:VEC 84;COUN 7;DATA:FIEL BDSELED;PATT #H3,#H3,#H3,#H3, #H3,#H3,#H3
REC:VEC 91;COUN 7;DATA:FIEL BDSELED;PATT #H4,#H4,#H4,#H4, #H4,#H4,#H4
REC:VEC 98;COUN 7;DATA:FIEL BDSELED;PATT #H5,#H5,#H5,#H5, #H5,#H5,#H5
REC:VEC 105;COUN 7;DATA:FIEL BDSELED;PATT #H6,#H6,#H6,#H6,
#H6,#H6,#H6
REC:VEC 112;COUN 7;DATA:FIEL BDSELED;PATT #H7,#H7,#H7,#H7,
#H7,#H7,#H7
REC:VEC 119;COUN 2;DATA:FIEL BDSELED;PATT #H0,#H0
REC:VEC 121;COUN 14;DATA:FIEL BDSELED;PATT #H0,#H0,#H0,#H0,
#H0,#H0,#H0,#H0,#H0,#H0,#H0,#H0,#H0,#H0
```

```
REC:VEC 135;COUN 14;DATA:FIEL BDSELED;PATT #H1,#H1,#H1,#H1,
#H1,#H1,#H1,#H1,#H1,#H1,#H1,#H1,#H1,#H1
REC:VEC 149;COUN 14;DATA:FIEL BDSELED;PATT #H2,#H2,#H2,#H2,
#H2,#H2,#H2,#H2,#H2,#H2,#H2,#H2,#H2,#H2
REC:VEC 163;COUN 14;DATA:FIEL BDSELED;PATT #H3,#H3,#H3,#H3,
#H3,#H3,#H3,#H3,#H3,#H3,#H3,#H3,#H3,#H3
REC:VEC 177;COUN 14;DATA:FIEL BDSELED;PATT #H4,#H4,#H4,#H4,
#H4,#H4,#H4,#H4,#H4,#H4,#H4,#H4,#H4,#H4
REC:VEC 191;COUN 14;DATA:FIEL BDSELED;PATT #H5,#H5,#H5,#H5,
#H5,#H5,#H5,#H5,#H5,#H5,#H5,#H5,#H5,#H5
REC:VEC 205;COUN 14;DATA:FIEL BDSELED;PATT #H6,#H6,#H6,#H6,
#H6,#H6,#H6,#H6,#H6,#H6,#H6,#H6,#H6,#H6
REC:VEC 219;COUN 14;DATA:FIEL BDSELED;PATT #H7,#H7,#H7,#H7,
#H7,#H7,#H7,#H7,#H7,#H7,#H7,#H7,#H7,#H7
REC:VEC 233;COUN 14;DATA:FIEL BDSELED;PATT #H0,#H0,#H0,#H0,
#H0,#H0,#H0,#H0,#H0,#H0,#H0,#H0,#H0,#H0
REC:VEC 247;COUN 14;DATA:FIEL BDSELED;PATT #H1,#H1,#H1,#H1,
#H1,#H1,#H1,#H1,#H1,#H1,#H1,#H1,#H1,#H1
REC:VEC 261;COUN 14;DATA:FIEL BDSELED;PATT #H2,#H2,#H2,#H2,
#H2,#H2,#H2,#H2,#H2,#H2,#H2,#H2,#H2,#H2
REC:VEC 275;COUN 14;DATA:FIEL BDSELED;PATT #H3,#H3,#H3,#H3,
#H3,#H3,#H3,#H3,#H3,#H3,#H3,#H3,#H3,#H3
REC:VEC 289;COUN 14;DATA:FIEL BDSELED;PATT #H4,#H4,#H4,#H4,
#H4,#H4,#H4,#H4,#H4,#H4,#H4,#H4,#H4,#H4
REC:VEC 303;COUN 14;DATA:FIEL BDSELED;PATT #H5,#H5,#H5,#H5,
#H5,#H5,#H5,#H5,#H5,#H5,#H5,#H5,#H5,#H5
REC:VEC 317;COUN 14;DATA:FIEL BDSELED;PATT #H6,#H6,#H6,#H6,
#H6,#H6,#H6,#H6,#H6,#H6,#H6,#H6,#H6,#H6
REC:VEC 331;COUN 14;DATA:FIEL BDSELED;PATT #H7,#H7,#H7,#H7,
#H7,#H7,#H7,#H7,#H7,#H7,#H7,#H7,#H7,#H7
REC:VEC 345;COUN 1;DATA:FIEL BDSELED;PATT #H0

;Define WEBED field
REC:VEC 1;COUN 6;DATA:FIEL WEBED;PATT #B1,#B1,#B1,#B1, #B1,#B1
REC:VEC 7;COUN 14;DATA:FIEL WEBED;PATT #B1,#B1,#B1,#B0,#B1,#B0,#B1,
#B1,#B1,#B1,#B0,#B1,#B0,#B1
REC:VEC 21;COUN 14;DATA:FIEL WEBED;PATT #B1,#B1,#B1,#B0,#B1,#B0,#B1,
#B1,#B1,#B1,#B0,#B1,#B0,#B1
REC:VEC 35;COUN 14;DATA:FIEL WEBED;PATT #B1,#B1,#B1,#B0,#B1,#B0,#B1,
#B1,#B1,#B1,#B0,#B1,#B0,#B1
REC:VEC 49;COUN 14;DATA:FIEL WEBED;PATT #B1,#B1,#B1,#B0,#B1,#B0,#B1,
#B1,#B1,#B1,#B0,#B1,#B0,#B1
REC:VEC 63;COUN 14;DATA:FIEL WEBED;PATT #B1,#B1,#B1,#B0,#B1,#B0,#B1,
#B1,#B1,#B1,#B0,#B1,#B0,#B1
REC:VEC 77;COUN 14;DATA:FIEL WEBED;PATT #B1,#B1,#B1,#B0,#B1,#B0,#B1,
#B1,#B1,#B1,#B0,#B1,#B0,#B1
REC:VEC 91;COUN 14;DATA:FIEL WEBED;PATT #B1,#B1,#B1,#B0,#B1,#B0,#B1,
#B1,#B1,#B1,#B0,#B1,#B0,#B1
REC:VEC 105;COUN 14;DATA:FIEL WEBED;PATT #B1,#B1,#B1,#B0,#B1,#B0,#B1,
#B1,#B1,#B1,#B0,#B1,#B0,#B1
REC:VEC 119;COUN 2;DATA:FIEL WEBED;PATT #B1,#B1
REC:VEC 121;COUN 225;DATA:FIEL WEBED;FILL:TYPE REP;INT 1;PATT
#B1;EXEC

;Define SYNCOT field for debug purpose
;STIM:VEC 1;COUN 345;DATA:FIEL SYNCOT;FILL:TYPE REP;INT 1;PATT
#B0;EXEC ;
;STIM:VEC 1;COUN 1;DATA:FIEL SYNCOT;PATT #B1
;STIM:VEC 317;COUN 1;DATA:FIEL SYNCOT;PATT #B1

;Define SYNCED field for debug purpose
;REC:VEC 1;COUN 345;DATA:FIEL SYNCED;FILL:TYPE REP;INT 1;PATT
#B0;EXEC
;REC:VEC 1;COUN 1;DATA:FIEL SYNCED;PATT #B1
;REC:VEC 317;COUN 1;DATA:FIEL SYNCED;PATT #B1

REC:TRAC:QUAL 1:FIEL DATAED;PATT #hxxxxxx
REC:TRAC:QUAL 1:FIEL ADDRED;PATT #h0000
REC:TRAC:QUAL 1:FIEL OSELED;PATT #b0
REC:TRAC:QUAL 1:FIEL CSELED;PATT #B10
REC:TRAC:QUAL 1:FIEL BDSELED;PATT #hx
REC:TRAC:QUAL 1:FIEL WEBED;PATT #B1
REC:TRAC:QUAL 1:FIEL SYNCED;PATT #Bx
REC:TRAC:QCOM1 1
REC:TRAC:SEQ 1:DEF:FILT DATA:REC NCOM
REC:TRAC:SEQ 1:DEF:CRC:CALC NEV
REC:TRAC:SEQ 1:DEF:ADVS:ON QCOM1:COUN 1
REC:TRAC:SEQ 1:DEF:JUMP 1:ON NEV
REC:TRAC:SEQ 2:DEF:FILT DATA:REC NCOM
REC:TRAC:SEQ 2:DEF:CRC:CALC NEV
REC:TRAC:SEQ 2:DEF:ADVS:ON NCOM:COUN 65500
REC:TRAC:SEQ 2:DEF:JUMP 1:ON NEV
```

(THIS PAGE LEFT BLANK INTENTIONALLY)

App/Tech Note

TRACE Recording SR2500-11

Note:

AppNote SR2500-11 is downloadable from the following web link:

http://www.interfacetech.com/appnotes.html

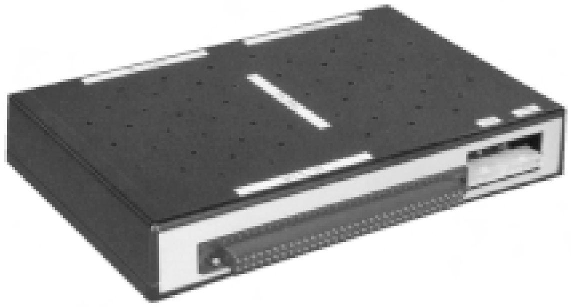
(THIS PAGE LEFT BLANK INTENTIONALLY)

Published by
Interface Technology, Inc.
300 S. Lemon Creek Drive
Walnut, CA 91789

User's Manual

SR2500-100 TTL Pod

The Performance Leader In
Test & Measurement Instrumentation



interface
TECHNOLOGY

SR2500-100 TTL Pod

Record of Changes

Change No.	Date of Change	Title or Brief Description	Entered By
Rev NC	Sep 01	Initial Release	Factory
Rev 01	May 02	Reformat manual	Factory

CHAPTER 1

General Information

Description

The SR2500 TTL Pod provides remote TTL drive capability for the SR2500 in applications where the Device Under Test (DUT) can not reliably drive the transmission cable to the SR2500. This may be due to weak drive currents, unterminated DUT drivers, or other reasons. And because the pod can be located close to the device under test, interconnect cables can be kept to a minimum length. This provides the added benefit of reducing noise, crosstalk and ringing associated with long, unterminated cables.

Each TTL Pod provides 32 bi-directional channels, each with independent Output, Tristate, Expect, Don’t Care (Mask) and Record memories provided by the SR2500 Digital Test Subsystem. Each channel is capable of driving an output high, low or tristating the output (high impedance) while simultaneously monitoring or recording the state of the channel.

The Pod is designed to interface to the Differential ECL (DECL) I/O of the SR2500. The DECL I/O provides both the drive state and the tristate control signals to the pod, and receives translated response signals from the DUT. Only the DECL I/O option for of the SR2500 provides all of the signals necessary to interface to the TTL Pod. No other SR2500 I/O interface should be used.

Installation

The SR2500 is a C2 VXI module (C-Size, Dual Slot). Follow the instructions in the SR2500 manual for installing the SR2500 into its VXI chassis.

See Fig 1. The SR2500 TTL Pod connects to the SR2500 front panel via the two 100 pin cables provided with the pod. Each cable provides sixteen channels of output, tristate and response signals. The top connector on the SR2500 DECL I/O connects to the pod connector labeled CH 0-15. The bottom SR2500 DECL connector is cabled to the pod connector labeled CH 16-31

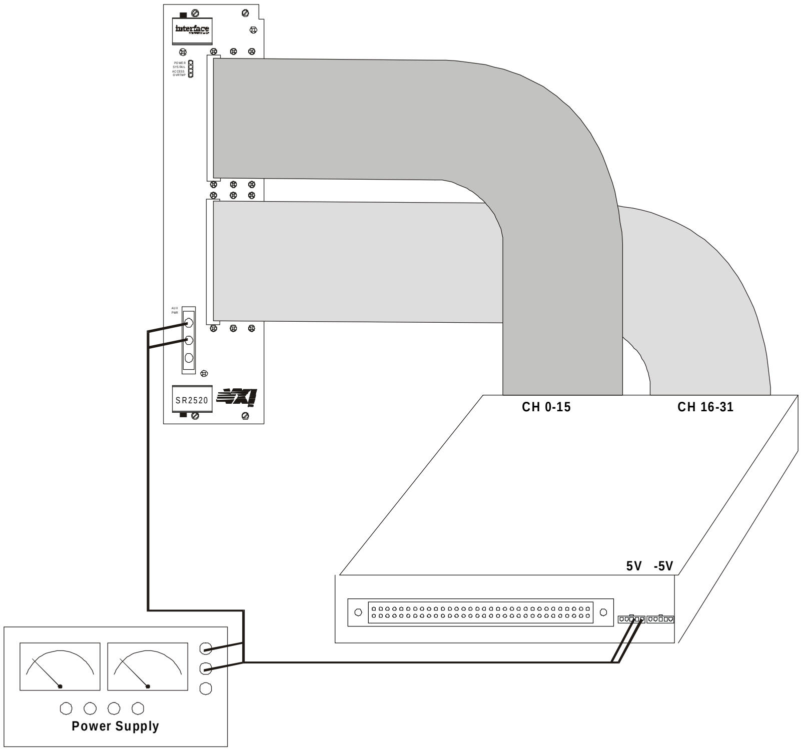


Figure 1. Pod Installation.

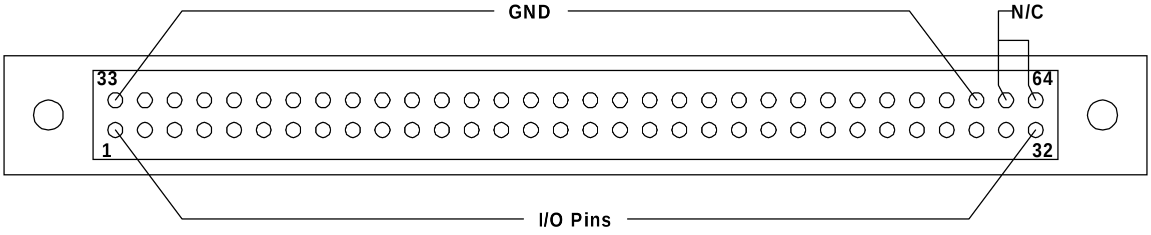


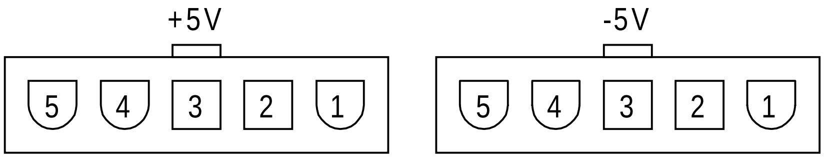
Figure 2. Front Panel view of the TTL Pod I/O Connector.

SR2500-100 TTL Pod I/O Pinouts.

Pin	Channel	Pin	Channel	Pin	Channel	Pin	Channel
1	00	9	08	17	16	25	24
2	01	10	09	18	17	26	25
3	02	11	10	19	18	27	26
4	03	12	11	20	19	28	27
5	04	13	12	21	20	29	28
6	05	14	13	22	21	30	29
7	06	15	14	23	22	31	30
8	07	16	15	24	23	32	31

External +5 Volt Power

The TTL Pod requires an external +5V, 5A power source. The power is brought into the pod via a Molex connector, part number 39-30-3055. An on-board DC-DC power converter provides the –5V power for the SR2500 DECL interface. A second 39-30-3055 Molex connector provides access to the –5V power for checking proper operation of the DC-DC converter. The pinout for the +5V and –5V connectors are shown in Fig 3.



1 VCC
2 GND
3 VCC Sense
4 GND Sense
5 N/C

1 GND
2 VEE
3 N/C
4 GND Sense
5 VEE Sense

The mating assembly for the +5V and –5V connectors require one Molex part number 39-01-4050, two Molex part number 44476-3112, two Molex part number 39-00-0039 and one Molex 15-04-0211.

Figure 3. Pinout for +5 V and -5 V Connectors.

CAUTION

Do NOT connect an external –5V power source to the –5V connector as damage may result. The connector is provided for measurement only.

Principles of Operation

The SR2500 TTL Pod converts the differential ECL outputs of the SR2500 into TTL outputs for the device under test. And it accepts TTL from the device under test and converts it into differential ECL for the return to the SR2500. The SR2500 provides both data output and tristate control to the pod, which in turn provides bi-directional capability for the DUT.

A DC-DC converter provides the –5V power required by the DECL translators. The DECL receivers (SR2500 Output and Tristate) are terminated via 50-ohm resistors through a 118-ohm resistor to VEE.

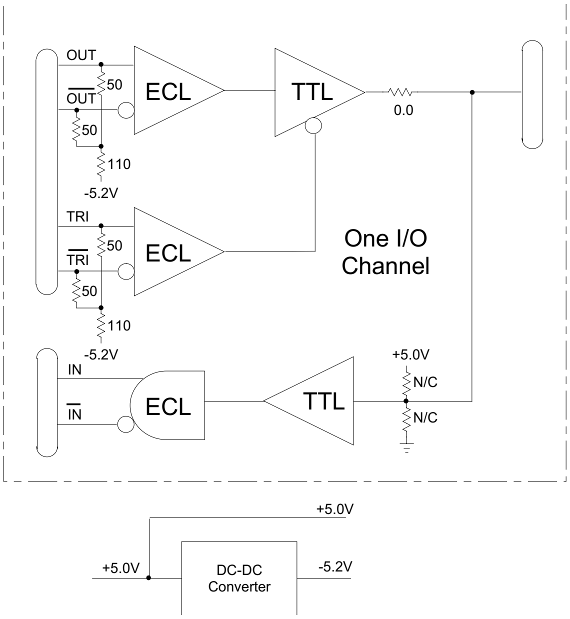


Figure 4. Pod Electronics, Simplified Schematic.

The pod does not re-clock any of the signals provided by the SR2500, instead the signals are passed through the pod to the DUT. Were the outputs to be re-clocked, then all formatting and timing parameters associated with a signal would be lost. Utilizing the pass-through approach, the full capabilities of the SR2500 are maintained.

Propagation Delay

Because there is propagation delay associated with the conversion from DECL to TTL, and from TTL to DECL, utilizing the TTL Pod will affect the operation of an SR2500 test program. The calculated propagation delay for the pod ranges from 18ns to 45ns, including propagation through the cables. Actual delay averages at 32ns. These are the propagation delays for the pod and SR2500-to-pod cables only. The DUT and the DUT cables introduce additional delays.

If using the SR2500/Pod combination to perform real-time compare of the DUT response with an expected DUT response, all propagation delays must be compensated for. Otherwise, it is possible for erroneous error conditions to be generated. For example, assume the SR2500 were operating at 25 MHz (40 ns period), and the round trip propagation delay from the SR2500, through the pod, through the DUT, back through the pod and returning to the SR2500 were 55 ns. By the time the DUT responds to the first SR2500 test pattern (vector #1), the SR2500 has moved to the second test vector. The SR2500 would then be comparing the DUT response to vector #1 with the expected response stored at vector #2. The SR2500 provides three methods for dealing with large propagation delays.

The first, and least desirable method is to manually offset the expected response by an appropriate number of vectors. In the example cited above, you would store the expected response to stimulus vector #1 at response vector #2, and the expected response to stimulus vector #2 at response vector #3. This can lead to confusion in the creation and maintenance of your test program.

The next approach is valid if the round trip propagation delay is well within the vector cycle time of the test program. In this case, you would simply adjust the sample delay to a point in time where the DUT response is guaranteed to be valid. Using the above example, if the round trip delay were 55 ns, but the test rate were 10 MHz (100 ns), you could set the sample delay to 95 ns and be assured of catching the DUT response. However, this approach works only when the DUT response is guaranteed to be received within the same test cycle as the stimulus pattern that initiated the response.

The last method takes advantage of the SR2500’s Expect Offset parameter. The expect offset function delays the comparison of the DUT response with the expected response by one or more full clock cycles. In effect, the expect offset feeds the expected response pattern stored in the SR2500’s expect memory through a FIFO before using it to compare against the DUT response. This FIFO can be from 0 to 7 clock cycles in length, allowing compensation for delays up to 7 full test cycles. The sample delay may still be used to add an additional delay of almost one full clock cycle resulting in the ability to compensate for nearly eight full clock cycles of propagation delay.

Programming:
Programming an SR2500 with the TTL Pod installed is identical to programming the SR2500 with standard TTL outputs, except for the effects of propagation delays, as discussed above. Channel groups can be created as Output type fields, Tristate type fields, OT (Output/Tristate)

type fields, Expect type fields, Don’t Care (Mask) type fields, ED (Expect/ Don’t’ Care) type fields and Record type fields; the same as for SR2500 systems without pods.

If separate Output and Tristate type fields are defined for a field, then the output will be tristated for any bit at any time where a “1” is stored in the tristate field. If a combined OT type field is used, then an “X” value will cause a bit to be tristated.

Wrap-Around Test

Because the TTL Pod connects the TTL driver and the TTL receiver internally, a simple wrap-around test can easily be created and executed to validate proper operation of the pod. The attached SCPI test program can be used to perform a wrap-around test on a single pod connected to channel card #1 on the SR2500. This program can also serve as a guide for creating other test programs.

The test was written to operate at 25 MHz. Since the pod delay is in the order of 30-40 ns, approaching the clock period of the test, the sample time was set to 20 ns and an expect offset of 1 cycle was programmed for the response fields.

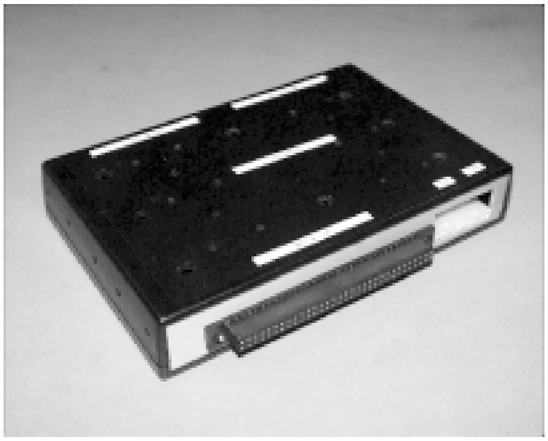
The test utilizes the real-time compare capability of the SR2500 to check operation of the pod. If the pod is connected and operating properly, the error state will be “0” – compare good – after the test completes. If there are any errors, the error state will be “1” – compare failed. The error state does not differentiate between one compare failure, or many. The record Trace function can be used to qualify that only compare failed cycles are recorded. This provides an easy method for determining the number of failures and which bits failed.

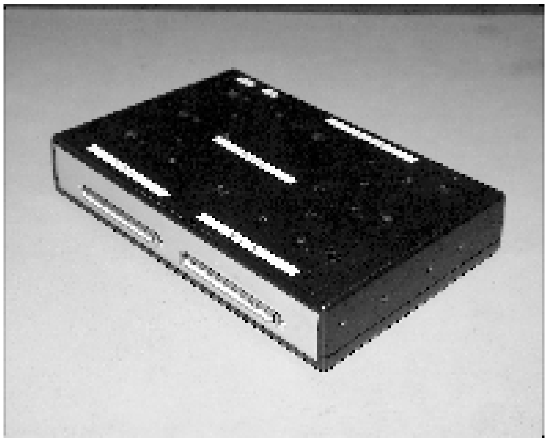
Since the test uses the error flag, it is necessary to “flush the pipeline” of the SR2500 at the beginning of the test. This guarantees that the SR2500 is in a known state prior to the actual test patterns being generated, and DUT responses evaluated. Refer to the SR2500 manual for additional information about pipeline flushing. The test pattern is a simple walking one pattern starting at test vector #4.

```
REC:VEC 1;COUN 36;DATA:FIEL RESP;PATT #bXXXXXXXXXXXXXXXXXXX

TEST:DEF TTL_POD:SIZE 64

SYST:TEST TTL_POD

SYST:PROG 1;FREQ 2.500000e+07;:BMAS:TIM 5.000000e-02;:SOUR:ROSC:SOUR INT

SYST:CLOCK:SOUR INT;SLOPE POS;LEVEL 1.200000e+00;:SYST:GATE:SOUR INT;POL
NORM;LEVEL 1.200000e+00

TRIG:SYST:SOUR BUS;SLOP POS;LEVEL 1.200000e+00

FIELD:DEF STIM:TYPE OT:PIN
C1P32,C1P31,C1P30,C1P29,C1P28,C1P27,C1P26,C1P25,C1P24,C1P23,C1P22,C1P21,C1P20,C1P19,C1P18,C1P17,

C1P16,C1P15,C1P14,C1P13,C1P12,C1P11,C1P10,C1P9,C1P8,C1P7,C1P6,C1P5,C1P4,C1P3,C1P2,C1P1

FIELD:DEF RESP:TYPE ED:PIN
C1P32,C1P31,C1P30,C1P29,C1P28,C1P27,C1P26,C1P25,C1P24,C1P23,C1P22,C1P21,C1P20,C1P19,C1P18,C1P17,C1P16,C1P15,C1P14,

C1P13,C1P12,C1P11,C1P10,C1P9,C1P8,C1P7,C1P6,C1P5,C1P4,C1P3,C1P2,C1P1

FIELD:DEF REC:TYPE REC:PIN
C1P32,C1P31,C1P30,C1P29,C1P28,C1P27,C1P26,C1P25,C1P24,C1P23,C1P22,C1P21,C1P20,C1P19,C1P18,C1P17,C1P16,C1P15,

C1P14,C1P13,C1P12,C1P11,C1P10,C1P9,C1P8,C1P7,C1P6,C1P5,C1P4,C1P3,C1P2,C1P1

REC:COND:SAMP:FIEL RESP;MODE EDGE,2.000000e-08;FIEL RESP;EOFF 1

STIM:COND:OFOR:FIEL STIM;MODE NRZ,0.000000e+00

REC:COND:SAMP:FIEL REC;MODE EDGE,2.000000e-08;FIEL REC;EOFF 1

STIM:VEC 1;COUN 36;DATA:FIEL STIM;PATT
#b00000000000000000000000000000000,#b00000000000000000000000000000000,#b000000000000000000000

00000000000,#b00000000000000000000000000000001,#b00000000000000000000000000000010,#b000000000

00000000000000000000100,#b00000000000000000000000000001000,#b00000000000000000000000000010000

,#b00000000000000000000000000100000,#b00000000000000000000000001000000,#b00000000000000000000

000010000000,#b00000000000000000000000100000000,#b00000000000000000000001000000000,#b00000000

000000000000010000000000,#b00000000000000000000100000000000,#b0000000000000000000100000000000

0,#b00000000000000000010000000000000,#b00000000000000000100000000000000,#b0000000000000000100

0000000000000,#b00000000000000010000000000000000,#b00000000000000100000000000000000,#b0000000

0000001000000000000000000,#b00000000000010000000000000000000,#b000000000001000000000000000000

00,#b00000000001000000000000000000000,#b00000000010000000000000000000000,#b000000001000000000

00000000000000,#b00000001000000000000000000000000,#b00000010000000000000000000000000,#b000001

00000000000000000000000000,#b00001000000000000000000000000000,#b00010000000000000000000000000

000,#b00100000000000000000000000000000,#b01000000000000000000000000000000,#b10000000000000000

000000000000000,#bXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
REC:VEC 1;COUN 36;DATA:FIEL RESP;PATT
#bXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX,#bXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX,#b0000000000000000000
0000000000000,#b00000000000000000000000000000001,#b00000000000000000000000000000010,#b00000
000000000000000000000000100,#b00000000000000000000000000001000,#b00000000000000000000000000
010000,#b00000000000000000000000000100000,#b00000000000000000000000001000000,#b000000000000
00000000000010000000,#b00000000000000000000000100000000,#b00000000000000000000001000000000,
#b00000000000000000000010000000000,#b00000000000000000000100000000000,#b0000000000000000000
1000000000000,#b00000000000000000010000000000000,#b00000000000000000100000000000000,#b00000
000000000001000000000000000,#b00000000000000010000000000000000,#b00000000000000100000000000
000000,#b00000000000001000000000000000000,#b00000000000010000000000000000000,#b000000000001
00000000000000000000,#b00000000001000000000000000000000,#b00000000010000000000000000000000,
#b00000000100000000000000000000000,#b00000001000000000000000000000000,#b0000001000000000000
0000000000000,#b00000100000000000000000000000000,#b00001000000000000000000000000000,#b00010
000000000000000000000000000,#b00100000000000000000000000000000,#b01000000000000000000000000
000000,#b10000000000000000000000000000000,#bXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
STIM:VEC 1;COUN 36;CMAC:DEF (SP (OUT)),(WL (OUT(COUN == 10))),(CLEARE (OUT)),(SL (OUT(COUN ==
10000))),(OUT (OUT)),(OUT (OUT)),(OUT (OUT)),(OUT (OUT)),(OUT (OUT)),(OUT (OUT)),(OUT (OUT)),(OUT
(OUT)),(OUT (OUT)),(OUT (OUT)),(OUT (OUT)),(OUT (OUT)),(OUT (OUT)),(OUT (OUT)),(OUT (OUT)),(OUT
(OUT)),(OUT (OUT)),(OUT (OUT)),(OUT (OUT)),(OUT (OUT)),(OUT (OUT)),(OUT (OUT)),(OUT (OUT)),(OUT
(OUT)),(OUT (OUT)),(OUT (OUT)),(OUT (OUT)),(OUT (OUT)),(OUT (OUT)),(OUT (OUT)),(EL (OUT)),(EP
(OUT))
```





*SR2500-100 TTL POD SPECIFICATIONS**

I/O Chanels:

Output Channels

Channels	32
Driver	74F125
Voh	2.0V min., 3.3V typ.
Vol	0.42V typ., 0.55V max.
Ioh	-15 mA, max
Iol	64 mA, max

Input Channels

Channels	32
Receiver	74F244
Vih	2.0V min.
Vil	0.8V max.
Iih	20uA, max
Iil	-1.6mA, max

Power

Voltage	5.0V
Current	5.0A

Dimensions

Length	9.1"
Width	6.2"
Height	1.5"

Environmental

Operating Temperature	0º C to +40º C

Mating Connector (Virgina Panel)

Connector Housing	510-108-101
Male Pins (64)	610-110-108

Power Connector (Molex)

Connector Housing	39-01-4050
Male Pins (2)	44476-3112
Male Pins (2)	39-00-0039
Polarizing Key	15-04-0211

(THIS PAGE INTENTIONALLY LEFT BLANK)

Published by
Interface Technology, Inc.
300 S. Lemon Creek Drive,
Walnut, CA 91789