

ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

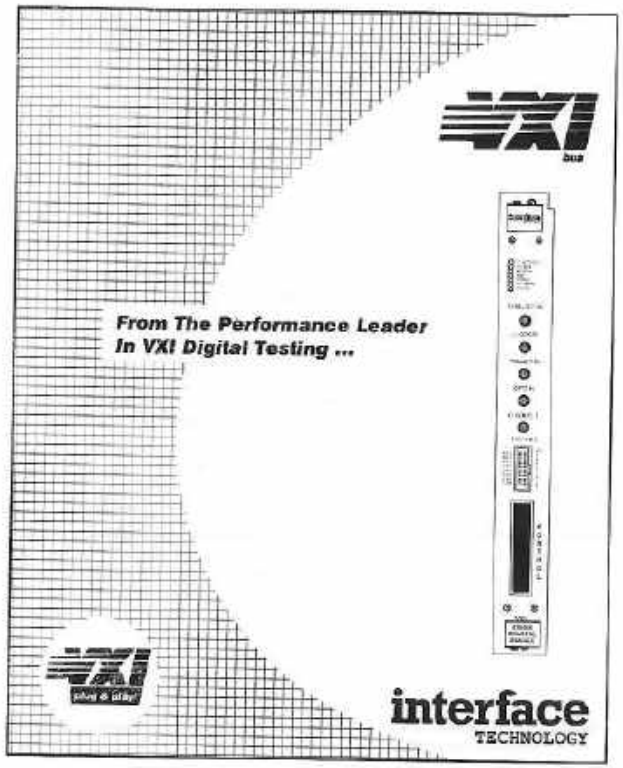
Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Visit our website - *Click HERE*

User's Manual

SR5010 Timing/Control Module



Rev 2.0 Mar 01, 1999
Chg 2.0 Sep 17, 2001

Record of Changes

Change No.	Date of Change	Title or Brief Description	Entered By
Rev 1.0	Apr, 1997	Complete rework and reformat.	Factory
Rev 2.0	Mar 01, 1999	Changes to Chapter 3, *"Programming."*	Factory
Chg 1.0	Oct 15, 1999	pg 3-4. Added paragraph entitled "Stand-Alone Tests Are Preferable".	Factory
Chg 2.0	Sep 17, 2001	pg 1-5. Reformat specifications page.	Factory

Contents

Chapter 1 General Description

About This Manual 1-1
Arrangement of Contents 1-1
Applicability 1-1
Supersedure Notice 1-1
Equipment Description 1-1
VXI Bus Interface 1-2
Real Time Digital Testing 1-2
Powerful Macro Commands Control Test Execution and Data Analysis 1-2
High Performance Response Logic Analysis 1-2
Features 1-2
Controls and Indicators 1-3
LEDs 1-3
Connectors 1-3
Interconnection With I/O Modules 1-4
Specifications 1-5

Chapter 2 Theory of Operation

Block Diagram 2-1
System Processor 2-1
Dynamic RAM 2-1
EPROM 2-1
EEPROM 2-1
VXI Interface 2-1
VME A32 Memory 2-2
Control Processor and Memory 2-2
PLL Interface 2-2
Input Buffer Thresholds 2-2
IT9010M VXI Integrator 2-2
A32 Memory 2-4
200 MHz PLO 2-4
Frequency Resolution 2-4
Frequency Accuracy 2-6
Control Gate Array 2-7
Control Processor 2-7
Control Processor Instructions 2-8
Output (OUTput): 2-8
Start Program/End Program (SProgram and EProgram): 2-9
Start Loop Until/End Loop (SLoopuntil and ELoop): 2-9
Word Loop Until (WLoopuntil): 2-9
Set Condition (SCONDition): 2-10
Set Jump Page (SJMPPage): 2-10

Contents (continued)

Unconditional/Conditional Jump (JMP and CJMP): 2-10
Unconditional/Conditional Jump Subroutine (JSRoutine and CJSRoutine): 2-10
Unconditional/Conditional Return Subroutine (RTSubroutine and CRTSubroutine): 2-11
Clear Error Latch (CLEARError): 2-11
Count (COUNt — {1 - 65535}): 2-12
Response Compares (RCOMpare == TRUE): 2-12
Response Does Not Compare (RCOMpare != TRUE): 2-12
Error Latch Set (LATCherror == TRUE): 2-12
Error Latch Not Set (LATCherror != TRUE): 2-12
System Trigger Has Occurred (STRigger — TRUE): 2-12
Input Flag Match (FRONtpanel && (#h00-#hFF with X's)): 2-13
Input Flag Does Not Match (FRONtpanel &! (#h00-#hFF with X's)): 2-13
Qualifier Match (QUALifword &&(#h00-#hFF)): 2-13
Qualifier Does Not Match (QUALifword &! (#h00-#hFF)): 2-13
Record State Machine 2-14
Filter: 2-14
Record 2-14
CRC 2-14
Advance Sequence 2-15
Jump to Sequence 2-15
Stop 2-16
Control Bus 2-16
Stimulus and Response Control 2-16
Returned Response Status 2-17
Record Control 2-17

Chapter 3 Programming

SCPI Command Syntax 3-1
Basic Programming 3-3
Defining Tests 3-1
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
Selecting the System Clock Source 3-18

Contents (continued)

Selecting the Slope of the External Clock 3-19
Setting the External Clock Threshold Level 3-20
Selecting the System Gate Source 3-21
Setting the External Gate Threshold Level 3-22
Selecting the Polarity of the External Gate 3-23
Selecting the Reference Oscillator Source 3-24
Selecting the System Trigger Source 3-25
Selecting the Slope of the External Trigger 3-26
Setting the External Trigger Threshold Level 3-27
Field Definitions 3-28
Field Definition & Pin Assignment 3-30
Field Deletion 3-36
Selecting the Field Radix 3-37
Field Definition Catalog 3-38
Loading and Querying Test Vectors 3-40
Selecting the Default Stimulus Field 3-43
Loading/Querying Stimulus Patterns 3-44
Clearing Stimulus Patterns 3-47
Enabling the Arrndata Function 3-49
Setting the Arrndata Pattern 3-50
Loading the Stimulus Macro Command Memory 3-52

COMMAND DEFINITIONS 3-56

SProgram[(OUT)] 3-56
EProgram[(OUT)] 3-56
OUTput[(OUT)] 3-56
WLoopuntil[(OUT)](<loop_cond>)) 3-56
SLoopuntil[(OUT](<loop_cond>)) 3-57
ELoop[(OUT)] 3-57
SCONDition[(OUT](<jump_cond>)) 3-58
SJMPPage(Supported Only on 256K and 1M Vector Cards) 3-58
JMP[(OUT](<label_name>)) 3-58
JSRoutine[(OUT](<label_name>)) 3-59
RTSubroutine[(OUT)] 3-59
CJMP[(OUT)(<label_name>)) 3-60
CJSRoutine[(OUT](<label_name>)) 3-60
CRTSubroutine[(OUT)] 3-61
CLEARError[(OUT)] 3-61
OUT 3-62

CONDITION DEFINITIONS 3-63

COUNt == count_value 3-62
RCOMpare == TRUE 3-63
RCOMpare != TRUE 3-63

LATCherror == TRUE 3-63
LATCherror != TRUE 3-63
STRigger == TRUE 3-63
FRONtpanel && match_pattern 3-64
FRONtpanel &!match_pattern 3-64
QUALifword && qual_combination 3-64
QUALifword &! qual_combination 3-64
Redefining Macro Command Label Vectors 3-65
Deleting Macro Command Labels 3-66
Copying Stimulus Macro Commands 3-67
Selecting the Default Record Field 3-69
Loading/Querying Record Patterns 3-70
Clearing Record Patterns 3-73

Trace TMACRO's 3-76

Post Trigger TMACRO Definition 3-78
Sequencing TMACRO Definition 3-82

Run Time Commands 3-85

Setting the Arm Counter 3-86
Initializing the Test Program 3-87
Software Trigger Command 3-88
Test Program Abort Command 3-89
Reading Recorded Data 3-90
Reading Recorded Patterns 3-92
Compare Error Status Query 3-94
Searching Record Memory 3-95

Advanced Programming 3-99

Pattern Editing 3-100

Copy Stimulus Patterns 3-103
Filling Stimulus Memory 3-106
Searching Stimulus Memory 3-110
Copying Record and Response Patterns 3-113
Filling Response Memory 3-116
Searching Response Memory 3-120
I/O Formatting and Timing 3-124
Stimulus Format and Timing 3-126
Clearing Stimulus Format and Timing 3-129
Stimulus Format and Timing Catalog 3-131

Record Sample Mode and Timing 3-133
Clearing Record Sample Mode and Timing 3-136
Expected Compare Offset 3-138
Record Sample Mode and Timing Catalog 3-139
Algorithmic Pattern Generation 3-140

Stimulus Algorithmic Pattern Generation 3-143

Algorithmic Output Command Definitions 3-145
NONAlgorithmic 3-145
INCrement 3-145
DECrement 3-145
XOR 3-145
HOLDData 3-145
HOLDAll 3-145
SLEFTZero 3-146
SLEFTOne 3-146
SLEFTComplement 3-146
RLEFT 3-146
SRIGHTZero 3-146
SRIGHTOne 3-147
SRIGHTComplement 3-147
RRIGHT 3-147
LOADParam 3-147
OUTPUTParam 3-147
Clearing Stimulus Algorithmic Memory 3-149
Copying Stimulus Algorithmic Commands 3-151
Filling Stimulus Algorithmic Memory 3-153

Response Algorithmic Pattern Generation 3-155

Algorithmic Expect Command Definitions 3-157
NONAlgorithmic 3-157
INCrement 3-157
DECrement 3-157
XOR 3-157
HOLDData 3-157
HOLDAll 3-158
SLEFTZero 3-158
SLEFTOne 3-158
SLEFTComplement 3-158
RLEFT 3-158
SRIGHTZero 3-159
SRIGHTOne 3-159
SRIGHTComplement 3-159

RRIGHT 3-159
LOADParam 3-159
OUTPUTParm 3-159
Clearing Response Algorithmic Memory 3-161
Copying Response Algorithmic Commands 3-163
Filling Response Algorithmic Memory 3-165

High Speed Binary Pattern Transfers 3-168

Stimulus Mapped Binary Patterns 3-170
Stimulus Non-Mapped Binary Patterns 3-172
Record Mapped Binary Patterns 3-174
Record Non-Mapped Binary Patterns 3-176

Saving and Loading Tests 3-178

Binary Learn Time-Out 3-180
Learning Binary Tests 3-181

Advanced Record Triggering 3-182

Record Memory Wrapping 3-185
Qualifier Trigger Patterns 3-186
Clearing Qualifier Trigger Patterns 3-188
Qualifier Trigger Combinations 3-189
Clearing Qualifier Trigger Combinations 3-190
Record Filter and Control 3-191
Advancing Trace Sequences 3-193
Jumping to Trace Sequences 3-195
CRC Calculation Control 3-197
Stopping Tests from Trace Sequences 3-199
Clearing Trace Sequences 3-200
Trace Sequences Catalog 3-201
Querying CRC Checksums 3-203

Miscellaneous Commands 3-205

Diagnostics 3-206
Diagnostic Test Execution 3-208
Diagnostic Test Status Query 3-210

Status Queries, Status Interrupts and System Queries 3-212

Operation Interrupt Definition 3-214
Operation Condition Query 3-216
Questionable Status Registers 3-217

Status Preset 3-220
System Error Query 3-221
SCPI Version Query 3-222
System IDN Query 3-223

Bus Master Time-Out 3-224

Bus Master Time-Out 3-225

IEEE 488.2 Commands 3-226
IEEE 488.2 Commands 3-229

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 Expected Responses 4-19
Recording UUT Responses 4-26

Chapter 5
Installation

Installation 5-1

Scope of Chapter 5-1
Unpacking and Inspection 5-1
Installation 5-1
Slot Dependency 5-2
Backplane Jumpers 5-2
Interconnections With Timing/Control Module 5-2
Setting Self-Diagnostics Mode 5-4

(This Page Intentionally Left Blank)

CHAPTER 1

General Description

About This Manual

This manual provides installation and operation information for the Interface Technology SR5010 Timing/Control 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 Programming
- Chapter 4 Programming Examples
- Chapter 5 Installation

Applicability

The information contained in this manual covers a single equipment configuration designated ***SR5010 Timing/Control 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 SR5010 User's Manual Rev 1.0 dated Apr. 1997.

Equipment Description

See Fig. 1-1. The SR5010 Timing/Control Module provides clocking and test sequence control functions for up to 10 SR50xx I/O modules in a single-chassis configuration, and up to 20 I/O modules in a dual chassis configuration. The SR5010 parses and interprets VXI word-serial commands from the Slot-0 Controller and provides overall system setup and test monitoring. The SR5010 also provides real-time control over the test pattern sequencing. With its built-in control processor, the SR5010 is capable of providing sequential or nested program looping, plus conditional or unconditional jumps and subroutines. Overall test timing is provided by an internal, programmable, 400 Hz to 50 MHz frequency synthesized clock source or by external inputs for clocks, gates, test inputs and triggers.

Features

- **Up to 50 MHz Data Rates.**
- **Store Up To 32 Different Test Programs.**
- **Powerful Dual Processor Architecture.**
- **64K Vector Depth.**
- **Multilevel Triggering For Logic Analysis**
- **Message-Based Commands for Easy Test Program Development.**
- **A32 / D32 Binary Transfer for High-Speed Test Program Download.**

The SR5000 module 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.

VXI Bus Interface

Based on the IT9010M industry standard VXI bus interface chip, the SR5010 meets the requirements of VXI Bus Specification Versions 1.3 and 1.4. The SR5010 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 SR50xx I/O modules. The System Processor provides the command power for the SCPI-syntax word serial command structure.

Real Time Digital Testing

The 50 MHz Control Processor provides real time control of the test pattern sequence by controlling nested looping and conditional branching. This capability allows the SR5000 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 SR5000 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.

Controls and Indicators

See Fig. 1-1. All the connectors and LEDs for the SR5010 Timing/Control module are located on the module front panel.

LEDs

There are seven LEDs located at the top of the SR5010 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 SR5010 Module.
- **RUN** (Green) - The SR5000 system is RUNNING.
- **ARMED** (Green) - The SR5000 system is armed and waiting for a system trigger.
- **BUS MAST** (Yellow) - SR5010 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.

Connectors

There are four Input Control Signals, one Output Control Signal and 8 TTL Input Flags on a 16 pin 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.

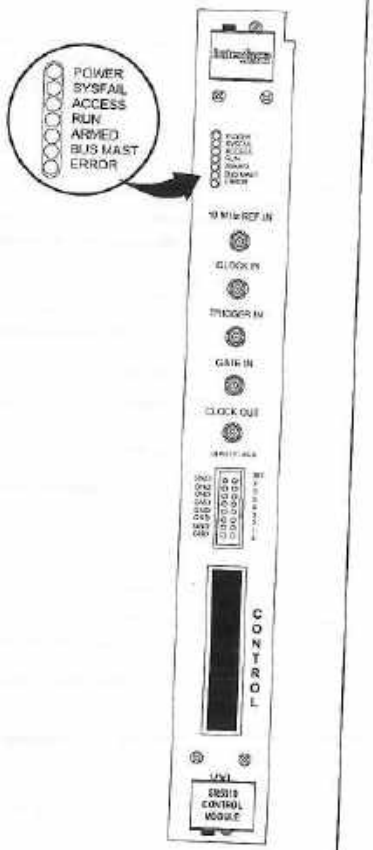


Figure 1-1
SR5010 Timing/Control Module

- **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.

Interconnection With I/O Modules

See Fig. 1-2. Clock and control commands from the SR5010 to the SR50xx I/O Modules is via special factory supplied ribbon cables that connect to the front panel CONTROL connections of the I/O Modules. These cables also return certain real-time status from the I/O Modules back to the SR5010.

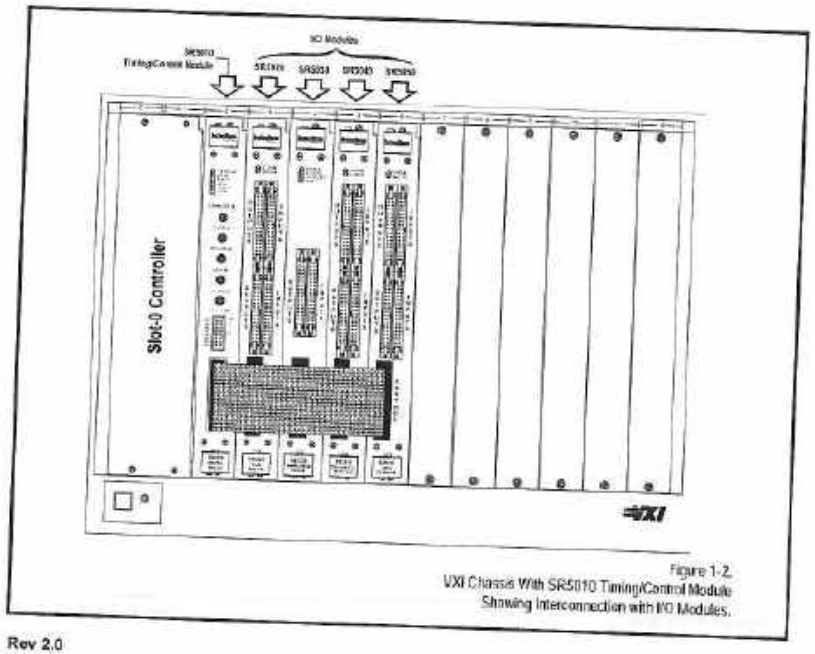


Figure 1-2.
VXI Chassis With SR5010 Timing/Control Module
Showing Interconnection with I/O Modules.

*SR5010 SPECIFICATIONS**

CPU:

System Processor	Motorola 68EC030 @ 25 MHz
Control Processor	50 MHz Custom Gate Array

Internal Clock:

Range	2.2 ns to 20 ns, 400 Hz to 50 MHz
Resolution	≤ 0.005%
Data Output Jitter	10 MHz reference jitter + 100 ps (short term RMS)

External Clock:

Range	DC to 50 MHz
Pulse Width	10 ns, minimum
Active Edge	Rising or falling
Input Voltage	-5.0 to +10.0 volts
Input Threshold	-5.0 to + 4.99 volts in 20 mV steps
Input Impedance	1M ohm

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 Level	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	1M ohm

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	1M ohm

External Input Flags:

Receiver Type	74ACT244
Number	Eight
Active Level	High or low
Input Voltage	Vil = ≤ 0.8 V Vih = ≥ 2.0 V
Input Impedance	10k ohms

Clock Output:

Driver Type	74F244
Output Level	TTL
Duty Cycle	50%
Output Termination	50 ohm, series

VXI Specifications

Interface Compatibility:

Type	Message-based, bus master / servant
Revision	1.3 and 1.4
Size	C-size, single slot
Configuration	Static
Interrupt Level	Programmable 1-7
Triggers	TTLTRG 0-7
Memory	1 MB VME A32 / D32 / D16 / D8

Power Requirements:

+5.0 volts	5.5 A	27.5 W
-5.2 volts	0.8 A	4.16 W
+12 volts	0.2 A	2.40 W
-12 volts	0.2 A	2.40 W
-2.0 volts	0.2 A	0.40 W
Total Power		36.86 W (max.)

Cooling Requirements:

Per-slot Average	37 W (max.)
Airflow	3L / sec @ 0.20 mm water pressure for 10° C temperature rise

Environmental Specifications:

Temperature	Storage = -40° C to +75° C Operating = 0° C to +45° C
Humidity	5% to 95% relative, non-condensing

Weight:

Weight	Approx 3.0 lb

Software Drivers:

National Instruments	LabView
National Instruments	LabWindows/CVI

* Specifications subject to change without notice.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 2

Theory of Operation

Block Diagram

The SR5010 Timing/Control Module provides all system timing and control for the SR50xx I/O Modules. Figure 2-1 is a simplified block diagram of the SR5010's major circuit stages and data paths.

System Processor

The 25 MHz 68EC030 System Processor is used to interpret and execute the SCPI word serial commands, including becoming Bus Master in order to transfer the parsed data to the SR50xx I/O Modules. This processor forms the basic user interface for the SR5000 subsystem and is responsible for the following non-real-time setup, query, and control functions:

- compiling and loading a test
- initializing and arming the hardware
- monitoring a test in progress
- returning test results to the host

The System Processor has access to all of the following subsystems:

Dynamic RAM

This 4 MB RAM is used to store test configuration and management data for the SR5000 subsystem.

EPROM

The SR5000 operating system (OS) is stored in this 256 KB EPROM memory.

EEPROM

This EEPROM is an 8 KB memory used to store calibration information for the SR5010 Timing/Control Module.

VXI Interface

Interface between the SR5010 and the VXIbus consists of a single Interface Technology IT9010M chip. This chip provides all of the configuration registers and message-based device registers required by the VXI specification, as well as control circuitry, arbitration between the VXI bus and the SR5010's local bus, bus master support, A24/A32 register support, and shared memory capability. Read Ready and Write Ready violation detection, data & signal register location monitors, and a multiple query error flag are all provided on-chip. Programmable interrupt level, signal register, offset register, programmable logical address register, and A24/A32 pointers are also provided on-chip. The System Processor has access to all of the registers within the IT9010M chip.

VME A32 Memory

The 1 MB A32 memory provides a high-speed alternative for transferring test setups and data to and from the SR5000 subsystem. Data may be transferred to/from the Slot-0 Controller from/to the A32 memory using D8, D16, or D32 transfers. Data is transferred between the SR5010 and the SR50xx I/O Modules using D32 only.

Control Processor and Memory

The Control Processor controls the generation of test vectors by executing a program stored in the Control RAM. Instructions for this program are called Command Macro (CMACRO) instructions. The System Processor has direct access to registers within the Control Processor, which are used during initialization and operation of the Control Processor. The System Processor also has read/write access to the control processor's instruction memory (Control RAM) whenever the Control Processor is stopped, thus allowing the System Processor to download test sequencing instructions and to perform memory diagnostics.

PLL Interface

The PLL interface allows the system processor to control the frequency of the internal PLL oscillator.

Input Buffer Thresholds

The front panel external clock (CLOCK IN) and trigger (TRIGGER IN) inputs have programmable input thresholds that are adjustable over a range of ±4.99 volts. These are high impedance inputs, greater than 1M ohm.

IT9010M VXI Integrator

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 may be accessed from either the VXIbus or from the 68EC030 System Processor, with all arbitration provided by 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 System Processor to the VXIbus. When a command is received by the SR5010 that requires communication with an SR50xx I/O Module, the System Processor (via the IT9010M) requests to become the VXI Bus Master. When Bus Master is granted, the System Processor carries out the requested operation directly with the register based I/O Module, without further assistance from the Slot-0 Controller, again, through the IT9010M. When the operation is complete, the VXIbus is released and the operational status message is returned to the Slot-0 Controller. With some operations, this process can be quite lengthy, so

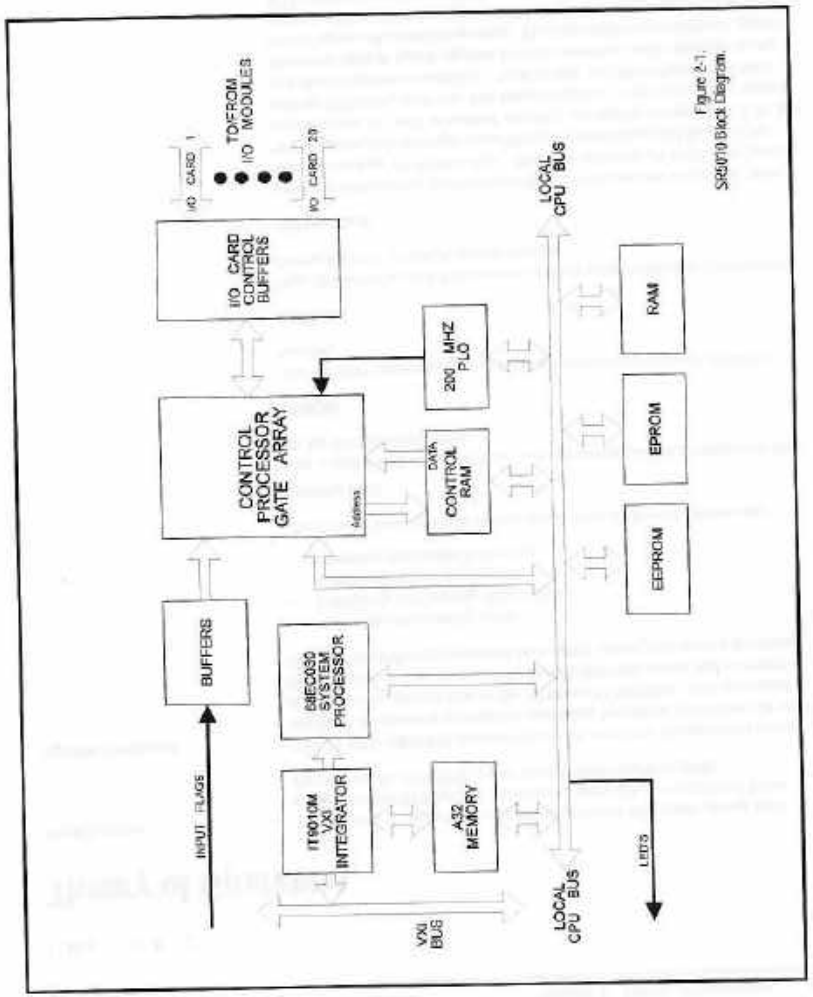


Figure 2-1. SR5010 Block Diagram

care should be taken that a VXIbus transfer not be initiated by the Slot-0 Controller without reasonable assurances that the bus is, or will be, available, else a VXIbus timeout error might occur.

A32 Memory

A 1 MB 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 provides high-speed, binary block transfers to and from the SR5000 subsystem. Two types of transfers are supported with the A32 memory option. In the first type, individual stimulus, response, and record memories on the SR50xx I/O Modules 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 SR5000 subsystem in the form of a binary memory image. Individual tests, or the entire SR5000 subsystem configuration, may be learned with a single command. Data is transferred in binary blocks using a software protocol similar 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 SR5000 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.

200 MHz PLO

The SR5000 system clock is provided by a 100 MHz to 200 MHz PLO stabilized ECL Phase Locked Oscillator (PLO). This PLO is phase locked to a 10 MHz reference and provides the basic timing for all SR5000 tests. The PLO output is divided by four within the control processor and then distributed throughout the system in two phases. Thus, the distributed "system clock" actually operates between 25 MHz and 50 MHz. Logic is provided in the stimulus and response processors located on the I/O modules, 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 381.48 Hz. However, the actual clock rate is software-limited to 400 Hz.

Frequency Resolution

The resolution of the PLO 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

25 MHz to 50 MHz, with a 2.5 kHz resolution. Any test operating in the 25-50.0 MHz range will have a clock resolution of 2.5 kHz (.005%). For test frequencies below 25 MHz, the system clock is divided 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 SR5010 to calculate the correct system clock and divisor values, a pseudo-code example follows:

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 - 25 MHz, the lower bound of the SYSTEM_CLOCK.

STOP_FREQ - 50 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 - 2.5 kHz, the resolution of the SYSTEM_CLOCK.

If the DESIRED_FREQ is less than 25 MHz, this algorithm 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%, the calculation is complete. 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 one is selected. If the DESIRED_FREQ is 25 MHz or greater, the ACTUAL_FREQ is calculated by rounding the DESIRED_FREQ to the nearest 2.5 kHz step (steps 30-31).

For DESIRED_FREQ less than 25 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_STEP)+.05)
8.  TRIAL_CLOCK = START_FREQ + (FREQ_STEPS * steps)
9.  save_div = TRIAL_DIV
10. save_diff = TRIAL_FREQ - DESIRED_FREQ
11. WHILE (TRIAL_CLOCK <= 50 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_STEP + .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 25 MHz

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

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 SR5000 rms output jitter will not exceed 10 MHz reference clock source jitter + 100 ps.

Control Gate Array

See Fig. 2-2. 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. 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 modules 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 modules 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

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 system processor automatically generates a simple default CMACRO program containing a series of output instructions (Do Not Loop, Do Not Branch), 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 (50 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

32K x 56 bit

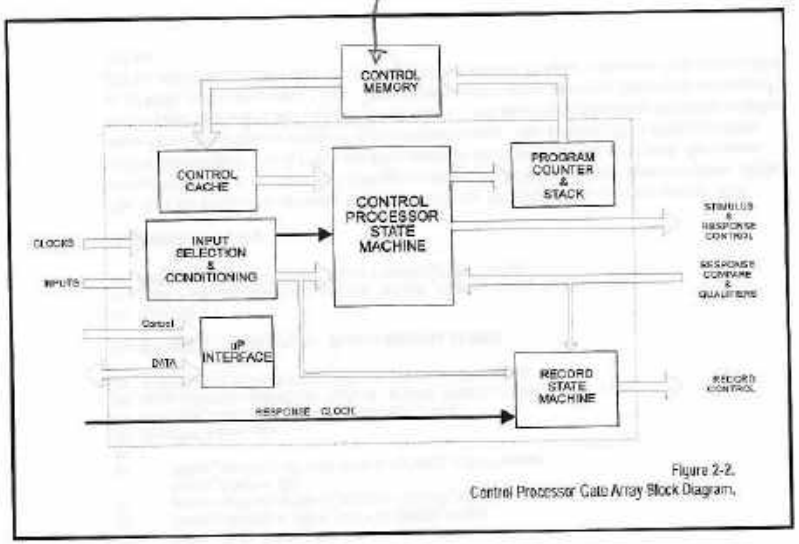


Figure 2-2.
Control Processor Gate Array Block Diagram.

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 32K words (64K 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. 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 the 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 to an odd vector, or five clock cycles for a jump to 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, so a pin using a return-to-zero format to generate a clock would remain active during the jump.

Unconditional/Conditional Jump Subroutine (JSRoutine and CJS Routine):

The *JSRoutine* and *CJSRoutine* instructions cause test execution to branch to the vector specified by a predefined label. The *JSRoutine* vector must be located on a (32 vector + 1) 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 at the state from the previous vector. During a jump, pin formatting remains active, so a pin using a return-to-zero format to generate a clock would remain active during the jump.

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.

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 calling the 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, so a pin using a return-to-zero format to generate a clock would remain active during the jump.

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.

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.

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.

Test Conditions

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.

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 (FRONtpanel && (#h00-#hFF with X's)):

This command provides a match evaluation of the 8 SR5010 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. Like the input flags pattern match condition, the 16-bit literal filed is logically broken into two 8-bit fields. The lower 8-bits (7-0) are used to bitwise compare against the front panel input flags. The upper 8-bits are used to bitwise enable the comparison.

Qualifier Match (QUALifword && (#h00-#hFF)):

The SR5000 supports eight system-wide response input comparators called qualifiers (0-7). Each qualifier can be programmed to compare each bit in an 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.

16 STATES : TRIGGER LEVELS - TRACE SEQUENCES

Record State Machine

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 system's operation. These operations are divided into the following functional areas:

IN32-1 1001 ... 1
REC MEM 1001 ... 1

IN32-1 1001 ... 0
EXP 1101 ... 1
REC 1011 ... 0

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 advance. This is the default condition.
- Advance after a specified number of test cycles.
- Advance after a specified number of response compares.
- Advance after a specified number of response does not compares.
- 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.

Control Bus

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.

The SR5010 Control/Timing module provides buffers and connectors for ten, radially connected, control buses. The control buses support the real-time clocks, status and control signals necessary to operate the I/O modules. Functions that are not real-time are supported via the VXI bus. The functions of these signals is briefly described in the paragraphs that follow.

Stimulus and Response Control

CLK: The CLK TTL clock signal is the main clock used by all gate arrays.

ECL_CLK1 and ECL_CLK2: These signals are balanced ECL versions of CLK and a quadrature version (90 degree phase shifted) of CLK. They form a 4-phase clock for the delay clock generators on the I/O modules.

STIM_EN and C_STIM_EN: These signals gate the system clock. When operating below 25 MHz with the internal clock, the CLK signal continues to operate between 25 MHz and 50 MHz but STIM_EN and C_STIM_EN operate at a slower, more effective frequency. For example, if 20 MHz is selected, CLK is set to 40 MHz and these signals operate at a 50 percent duty cycle.

STIM_CMD(1:0), M_START, M_STOP, M_DONE, START_JSR and LEVEL(2:0): To reduce the number of signals needed to connect the gate arrays, some logic of the control processor (e.g., the 20-bit address counter) is duplicated in a modified form in the stimulus and response gate arrays. These signals are partially encoded by the control processor so that all stimulus and response gate arrays stay in sync with the control gate

array. The stimulus and response gate arrays operate several clock cycles behind the control processor because of system pipelining and propagation delays.

RUNNING: The running signal indicates that the control processor is executing a program.

Returned Response Status

RET_RESP_CLK and RET_RESP_EN: These delayed versions of CLK and C_STIM_EN are returned from the first I/O module to run the record state machine section of the control gate array. Because of system pipelining and propagation delays, the record state machine operates several clock cycles behind the stimulus and response gate arrays.

RESP_COMP and QUALIFIER(0-7): These signals report the result of the response comparison and eight input qualifier words back to the control gate array. These none signals are logically AND'd together with the corresponding signals of the other I/O modules and form the basis of certain conditional instructions in the control processor. These signals are also used by the record state machine to determine the record vectors to be stored and the input vectors to be used for CRC (signature analysis) calculations.

Record Control

CMD_RESP_CLK and CMD_RESP_EN: These further delayed versions of CLK and C_STIM_EN are used by the record section of the response gate array. Because of system pipelining and propagation delays, this section runs several clock cycles behind the record state machine in the control gate array.

SAVE and SEL_DATA: These signals specify if a response input vector is to be saved (recorded) and if the data or the error vector should be saved.

CRC_EN: This signal specifies if a CRC calculation should be performed on a response input vector.

(This Page Intentionally Left Blank)

CHAPTER 3

Programming

SCPI Command Syntax

The SR5000 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 SR5000, 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 SR5000'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;DATA:FIELD D16_09;CLEAR
RECORD:VECTOR 1;COUNT;DATA:FIELD D08_0;CLEAR
RECORD:VECTOR 1;COUNT;DATA:FIELD D16_09;CLEAR,FIELD
D08_00;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 SR5000.

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 3-28*
- *Loading and Querying Test Vectors* ..... *pg 3-40*
- *Trace TMACRO's* ..... *pg 3-76*
- *Run Time Commands* ..... *pg 3-85*
- *Reading Recorded Data* ..... *pg 3-90*

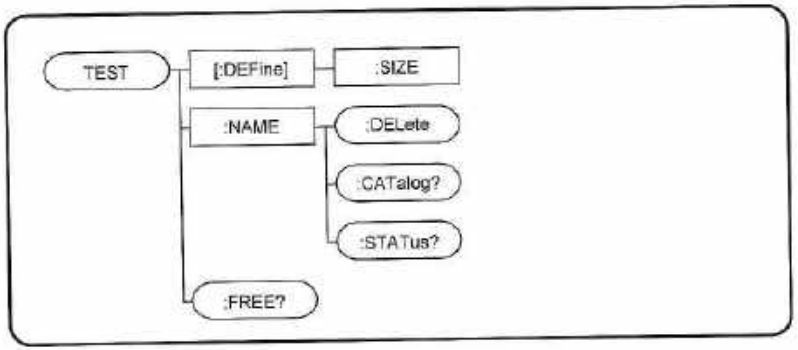
Defining Tests

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 32 unique test programs can be defined within the system at any given time, up to the available SR5000 I/O memory size (64K 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 SR5010 Timing/ Control 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.

Stand-Alone Tests Are Preferable

When creating test programs using any "return-to" data formats, it is advised that you write all tests as complete, stand-alone processes, eliminating any dependencies between the tests. This would imply that each test might need an initialization header or footer to place the UUT in a known state prior to, or after completion of the test patterns. This fits nicely with the recommendation to flush the SR5000 pipeline at the beginning of a test program, prior to generation of any patterns to the UUT. In this way, multiple tests may be executed in sequence without worry as to what state the UUT was left in, or what state the tester was left in. This can greatly simplify test program development and integration.



SR5000 Default Parameters

ITEM	MIN	MAX	DEFAULT	PARAMETERS
Program Loops	1	65,535	1	CONT
Frequency	400 Hz	50 MHz	25 MHz	—
Period	20 ns	2.5 ms	40 ns	—
Clock Source	—	—	INT	INT/EXT
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	—
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 SR5000 I/O cards. The memory is allocated across all SR5000 I/O cards using the same vector locations. SR5000 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 SR5000 is 32, 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.

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}

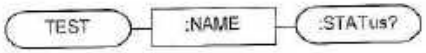
Parameter Definition **name** = Any alphanumeric string, including underscores '_' (max 8 characters).

test_size = (2 - 65500)

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 SR5000 briefly enters the stopped state and then automatically cycles to the idle state. Because of the brief time the SR5000 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)

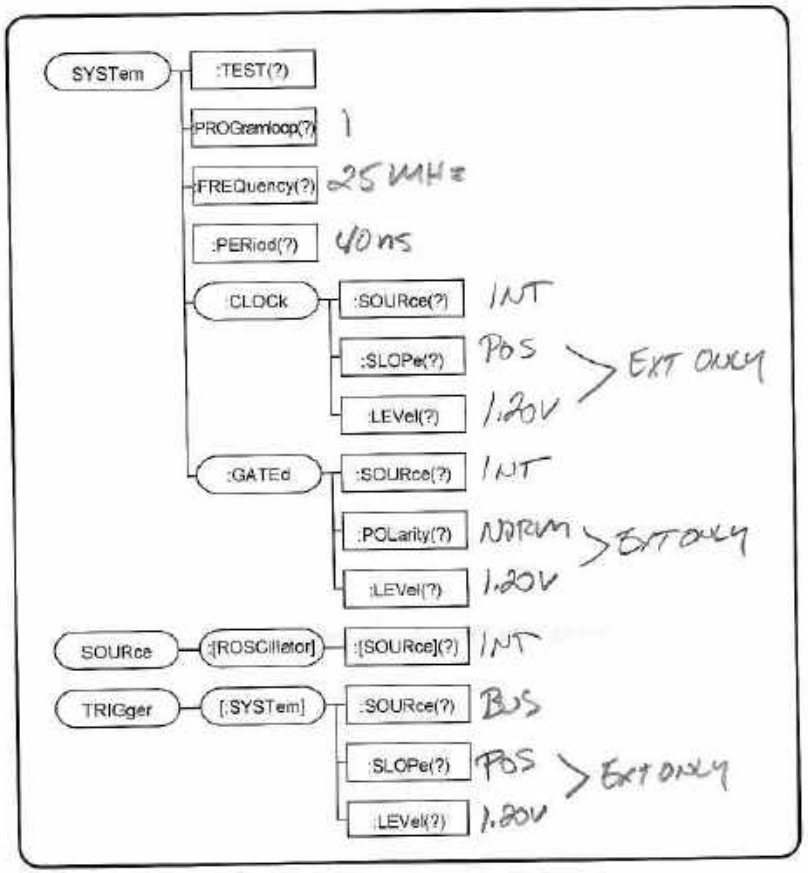
Examples TEST:FREE?
15384

THIS PAGE INTENTIONALLY LEFT BLANK

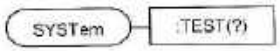
Global Test Parameters

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 on the SR5010 module. Stimulus and expected response data patterns are stored in memory located on the SR5020, SR5030, SR5040, and/or SR5050 modules.



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)

SYSTem — PROGramloop(?)

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)

SYSTem — :FREQuency(?)

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** = (400.00Hz - 50000000.00Hz) 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 = 400 Hz
MAX = 50 MHz
DEFault = 25 MHz

Examples
SYSTEM:FREQUENCY DEFAULT
SYST:FREQ 33.0e6
SYSTEM:FREQ 33MHZ

:FREQuency?

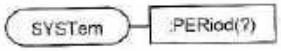
Response hertz

Parameter Definition **hertz** = The clock frequency specified in Hz using scientific notation values from 4.000000e+02 to 5.000000e+07.

Examples
SYSTEM:FREQUENCY?
3.300000e+07

SYST:FREQ?
3.300000e+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** = (20.0 ns to 2.5 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 = 20 ns
MAX = 2.5 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 2.500000e-03 to 2.000000e+08.

Examples
```
SYSTEM:PERIOD?
2.500000e-3
SYST:PER?
2.500000e-3
```


Selecting the System Clock Source (NON-SCPI)



The SYSTem:CLOCk:SOURce command selects the source of the test program clock. The default clock source is the internal clock on the SR5010 Timing/Control Module. The SYSTem :CLOCk:SOURce? query command returns the source of the test program clock

:SOURce <INTernal | EXTernal>

Parameter Definition **INTernal** = (default) The internal clock source from the SR5010 Timing/Control Module.

EXTernal = The user supplied signal into the front panel "CLOCK IN" connector on the SR5010 Timing/Control Module.

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.

Examples
```
SYSTEM:CLOCK:SOURCE INTERNAL
SYST:CLOC:SOUR EXT
```

:SOURce?

Response INT | EXT

Parameter Definition **INT** = The internal clock source from SR5010 Timing/Control Module.
EXT = The user supplied signal into the front panel "CLOCK IN" connector on the SR5010 Timing/Control Module.

Examples
```
SYSTEM:CLOCK:SOURCE?
EXT
SYST:CLOC:SOUR?
INT
```

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



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 SR5010 Timing/Control Module, 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 SR5010 Timing/Control Module. 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 SR5010 Timing/ Conrol Module.

EXTernal = The user supplied signal into the front panel "GATE IN" connector on the SR5010 Timing/Control Module.

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

:SOURce?

Response INT | EXT

Parameter Definition **INT** = The internal gate source from the SR5010 Timing/Control Module.

EXT = The user supplied signal into the front panel "GATE IN" connector on the SR5010 Timing/Control Module.

Examples SYSTEM:GATE:SOURCE?
EXT

SYST:GATE:SOUR?
INT

Setting the External Gate Threshold Level (NON-SCPI)



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 units 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)



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) A "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.

INV = 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?
NORM

SYST:GATE:POL?
INV

Selecting the Reference Oscillator Source (SCPI 19.16.3)



The SOURce:ROSCillator:SOURce command selects the source of the 10 MHz reference for the Phased Lock Loop Oscillator. The internal clock on the SR5010 Timing/Control Module 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 SR5010 Timing/Control Module. 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 SR5010 Timing/Control Module. 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 SR5010 Timing/Control Module.

EXT = The user supplied clock into the front panel "10MHz REF IN" connector on the SR5010 Timing/Control Module.

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 SR5000 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 SR5000 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 SR5000 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 SR5010 Timing/Control Module.
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
```

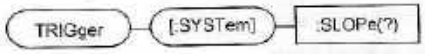
:SOURce?

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 SR5010 Timing/Control Module.
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.

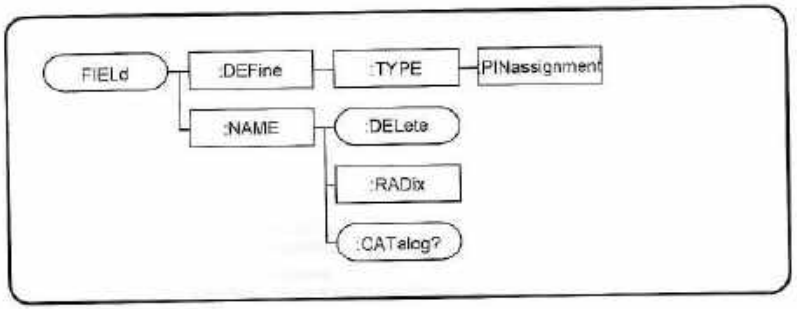
Examples TRIGGER:SYSTEM:LEVEL?
0.200000

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, SR5020 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. The 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 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 reduce, 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 map-

ping 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 I/O module number is determined by its relative position in the VXI chassis. The I/O module with the next higher logical address after the SR5010 is 'C1' (card #1). The I/O module with the next higher addres is 'C2'. This order continues for each I/O module installed.

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

```
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
```

Note

These examples are functionally identical.

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)



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>

Default: 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 modules installed)

pin# = (1 - 32)

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

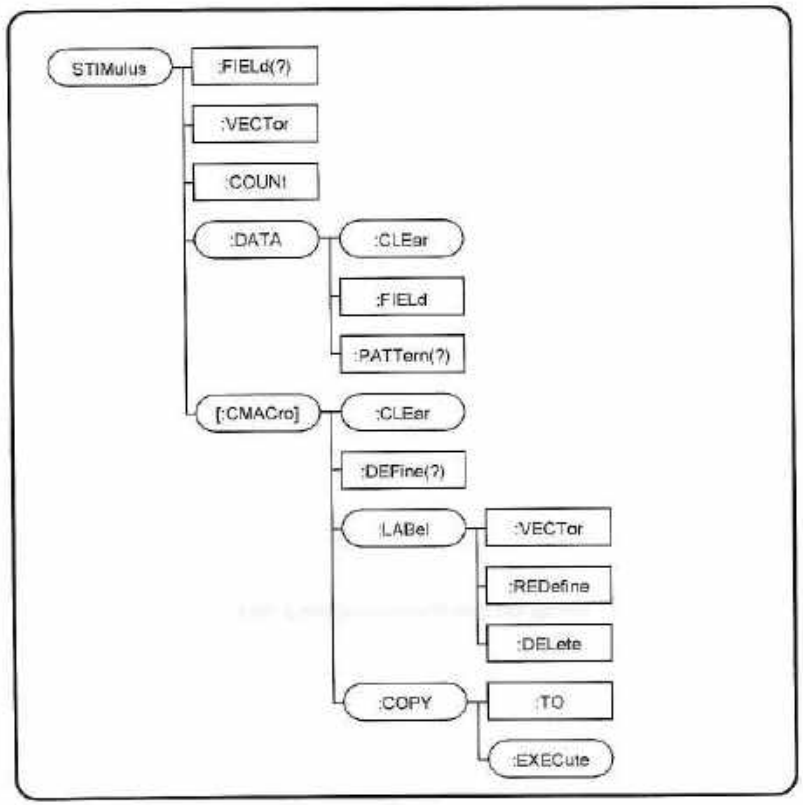
FIEL:NAME ALL:CAT?
MEM_1,OT,HEX,C1P4,C1P3,C1P2,C1P1,C2P4,C2P3,C2P2,C2P1;MEM_2,ED,HEX,C1P4,C1P3,C1P2,C1P1,C2P4,C2P3,C2P2,C2P1

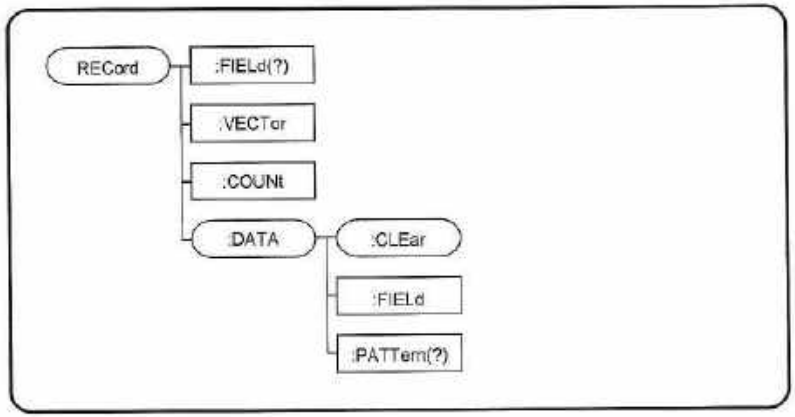
THIS PAGE INTENTIONALLY LEFT BLANK

Loading and Querying Test Vectors

In order to use the SR5000 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 SR5000 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 3-99.


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?

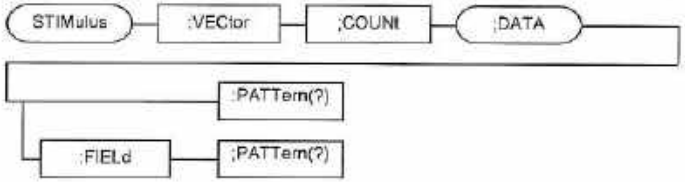
Response name

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,#b0000000001010101,#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).

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.

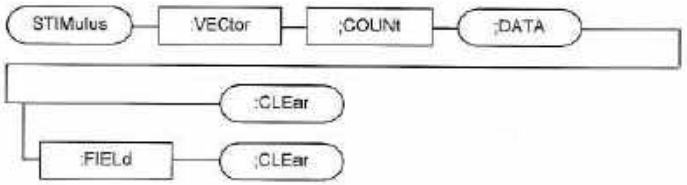
Response data_value{,data_value}

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

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 SR5000 is in an ARMED state, waiting for a trigger. The user-defined data pattern will be held on the stimulus pins until the SR5000 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
```

All commands shown above perform identical functions.

:MODE?

Response 0 | 1

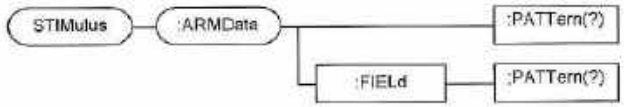
Parameter Definition

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 SR5000 is placed in the ARMED state and waiting for a trigger. The SR5000 is placed in the ARMED state when a.) the INITiate command is received; or b.) when the SR5000 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 SR5000 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).

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,#b0000101011111XX0X0
```


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 SR5000 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 3-1. 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. SProgram [(OUT)]
2. EProgram [(OUT)]
3. OUTput [(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 SR5000 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)
```

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;CMACRO:DEFINE ((LABEL
W_DTACK)WL(OUT(STRI == TRUE)))
STIM:VECT 2;COUN 1;CMACRO: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 && #b10011111)))
```

SJMPPage (Supported Only on 256K and 1M 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
```
STIMULUS:VECTOR 105;COUNT 1;CMACRO:DEFINE (SJMPPAGE(OUT(2)))
STIM:VECT 900,000;COUN 1;CMACRO:DEF (SJMPP(OUT(19)))
STIM:VECT 46,528;COUN 1;CMACRO:DEF (SJMPP(OUT(31)))
```

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 (not currently possible), 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.

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 (not currently possible), 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 (not currently possible), 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 (not currently possible), 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 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;DEF(CJSR(OUT(READ)))
STIM:VECT 255;COUN 1;DEF(CJSR(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)
```

OUT

OUT = A command modifier that allows the current stimulus data vector and output formatting and timing to be enabled.

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 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.

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 ECL input flags located on the SR5010 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 ECL 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 SR5000 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.

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.

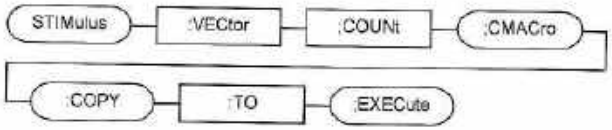
Parameter Definition none

: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 other RECord commands. 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?

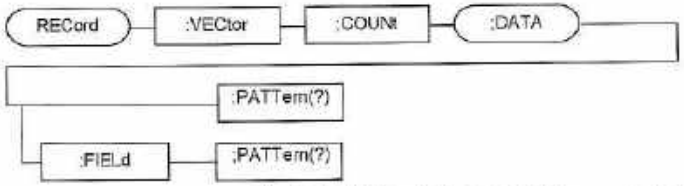
Response name

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 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 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 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. Each field defined can have a different radix format. Leading '0' data characters will be returned.

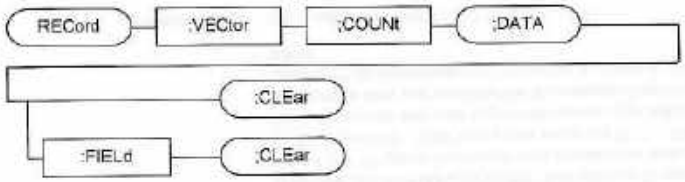
Response data_value{,data_value}

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
```

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

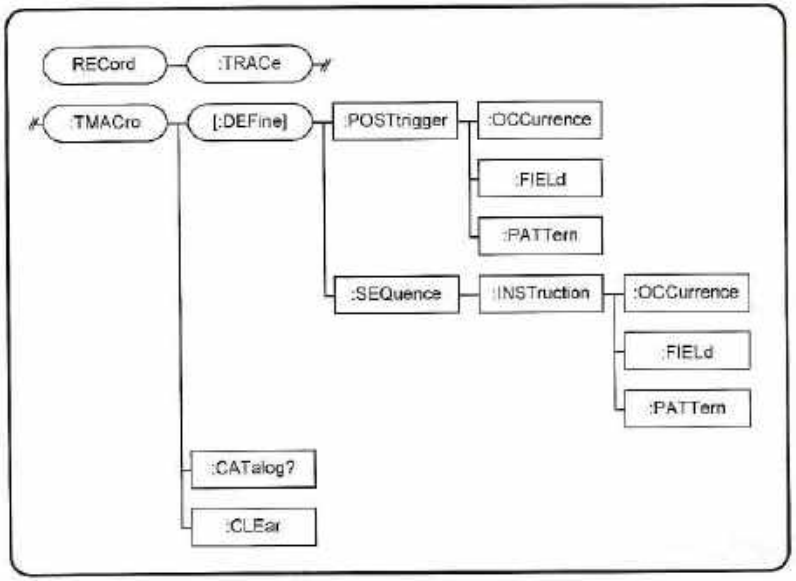
THIS PAGE INTENTIONALLY LEFT BLANK

Trace TMACRO's

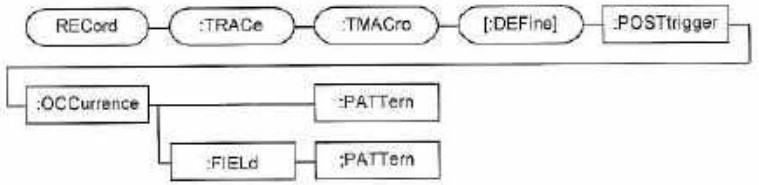
Trace TMACRO's provide a convenient method of triggering the SR5000 record logic for recording data, and are an alternative to record Trace Sequences (see page 3-193). TMACRO's take two forms: Post Trigger and Sequence. Both TMACRO versions actually compile into Trace Sequence functions, so the three 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.



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 record 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 SR5000 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.

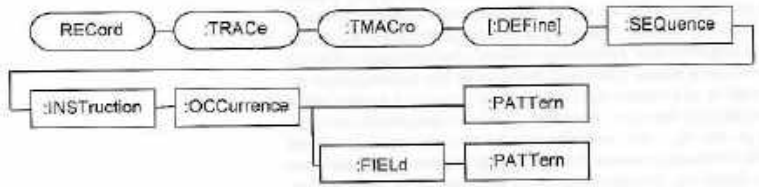
Example 2. REC:TRAC:TMAC:DEF:POST 512:OCC 1;FIELD D15_00;PATTERN #bXX55

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 SR5000 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.

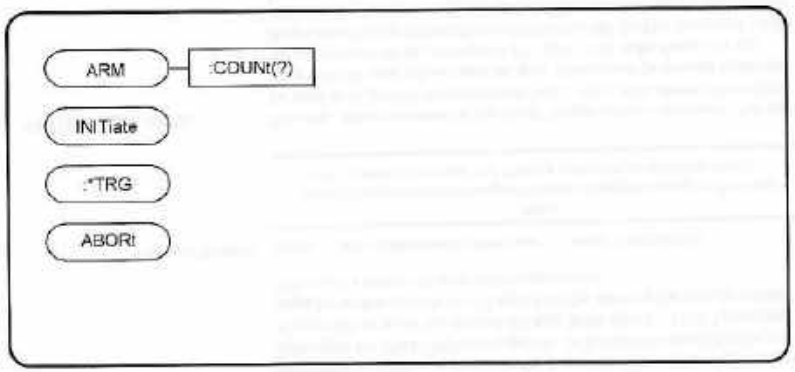
This example is similar to the TMACRO:POSTRIGGER examples and demonstrates a three step record process. Recording will start immediately and continue until the first trigger pattern of hex AAAA is detected, after which all recording will stop. Recording will remain suspended until the trigger pattern 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 SR5000 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
#b10XXXXXXXXXXXXX
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 SR5000 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 SR5000 will be armed. If the arm count is greater than 1, the SR5000 will return to the ARMED state when the test program execution is complete. The SR5000 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 SR5000 will be placed in the IDLE state. The SR5000 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 SR5000 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 SR5000 will then be ready and waiting for a trigger event (hardware signal or software command) to begin the test program execution. The SR5000 test program parameters ***cannot*** be modified or queried while in the ARMED or RUNNING state.

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 SR5000 is in the 'ARMED' state. Additional *TRG commands sent while the SR5000 is 'RUNNING', for the purpose of re-triggering the SR5000, 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 SR5000 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

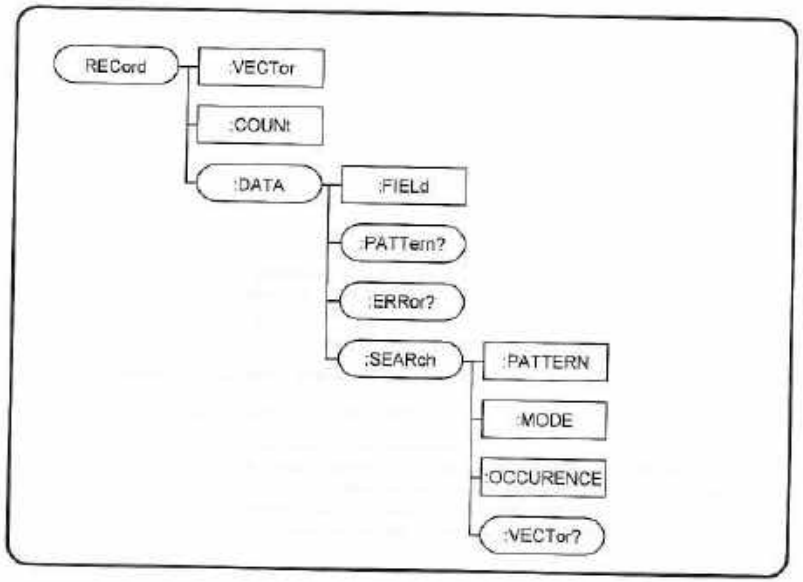
Examples
```
ABORT
ABOR
```


Reading Recorded Data

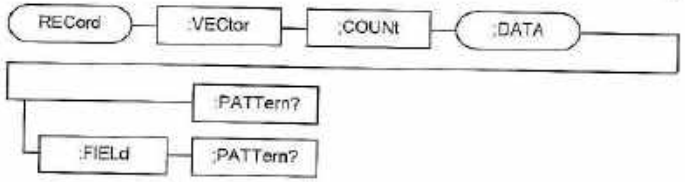
The Recorded Data commands provide access to record memory. Recorded data may be queried only when the SR5000 system has stopped, and only if data was recorded. You may use the TEST:NAME:STATUS? query command to query if, and how many, data samples were recorded. Recorded data may consist of the actual 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.



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 SR5000 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 CLEARError 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

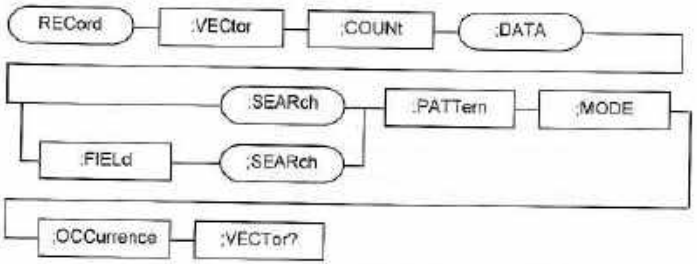
Parameter Definition **0** = No compare error has occurred.

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

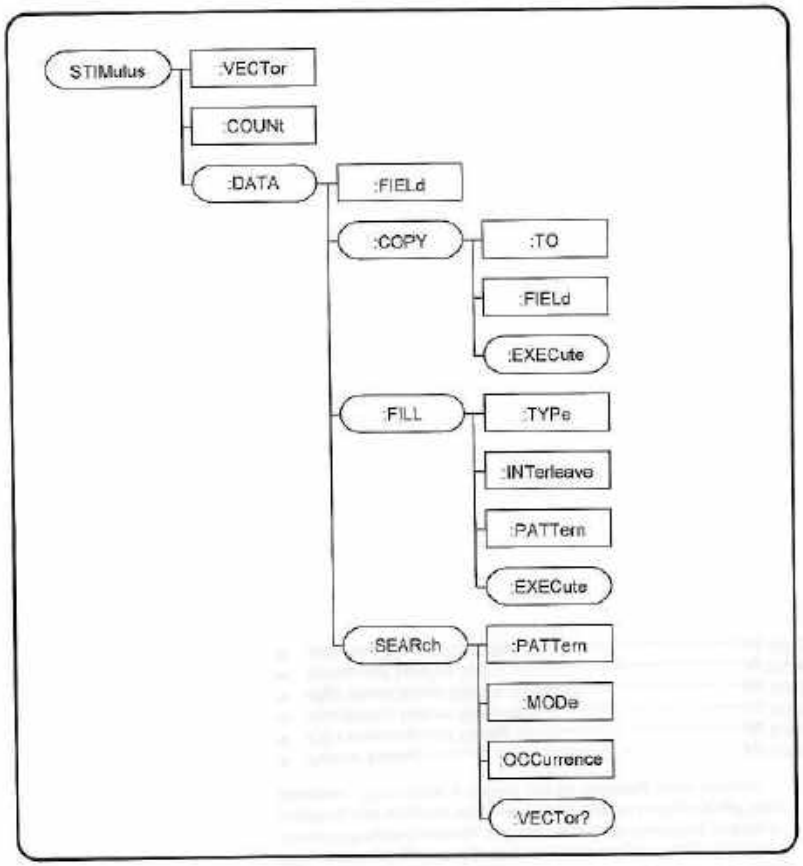
Advanced Programming

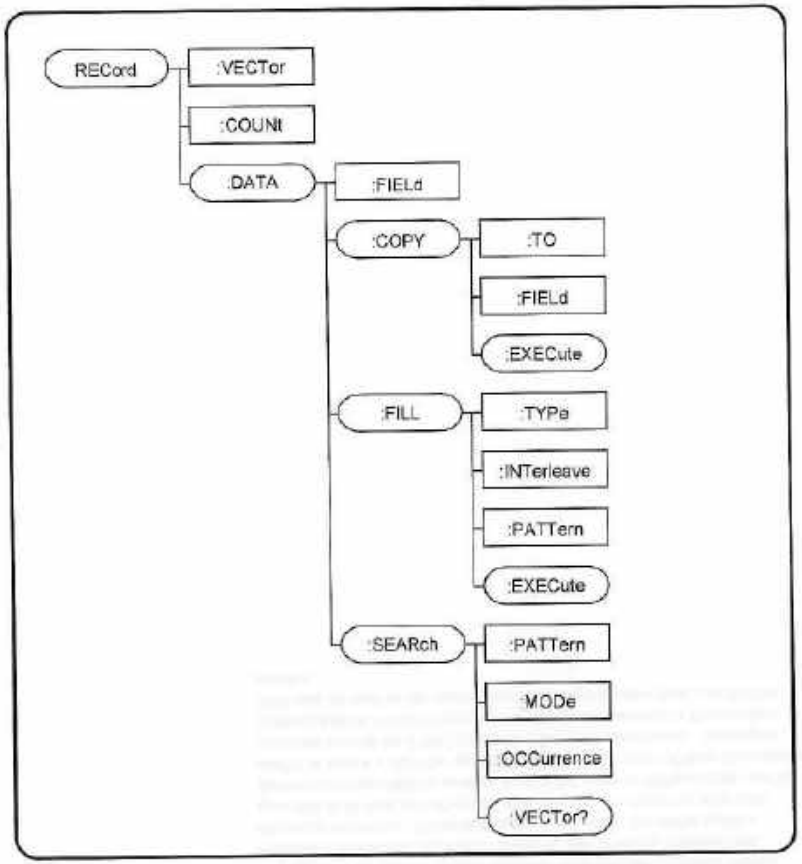
This section includes the more advanced commands required for editing, filling and copying Stimulus/Response patterns, generating Algorithmic Stimulus/Response patterns, defining output data formatting and input sampling, and using the high speed binary pattern load/query and learn functions. This section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

- *Pattern Editing* ..... *pg 3-100*
- *I/O Formatting and Timing* ..... *pg 3-124*
- *Algorithmic Pattern Generation* ..... *pg 3-140*
- *High Speed Binary Pattern Transfers* ..... *pg 3-168*
- *Saving and Loading Tests* ..... *pg 3-178*
- *Advanced Record Triggering* ..... *pg 3-182*

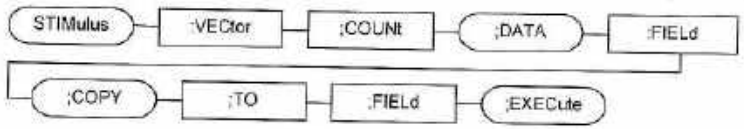
Pattern Editing

The SR5000 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.

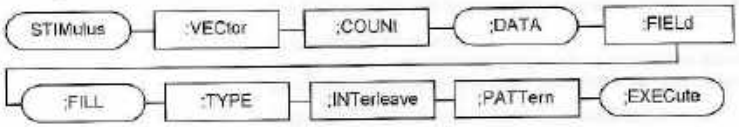
STIMULUS:VECTOR 1;COUNT ALL;DATA:FIELD ADDR;COPY:TO 1,FIELD DATA;EXECUTE

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

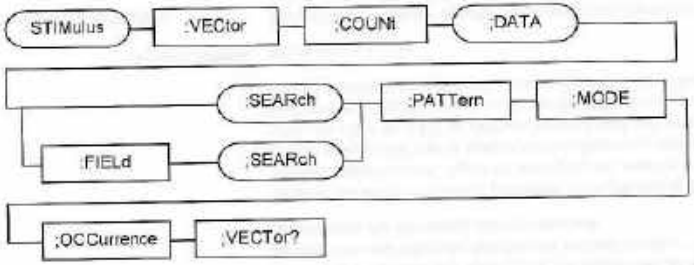
Examples STIMULUS:VECTOR 1;COUNT 100;DATA:FIELD ADDR;FILL:TYPE INCREMENT;INTERLEAVE 1;PATTERN #H0000;EXECUTE

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)

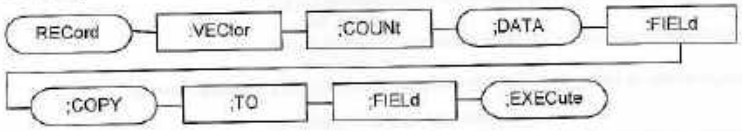
Examples STIMULUS:VECTOR 1;COUNT 100;DATA:FIELD ADDR;
SEARCH:PATTERN #hAAAA;MODE EQ;OCCURRENCE ALL;VECTOR?
24;#h0000AAAA;65,#h0000AAAA;72,#h0000AAAA;90,#h0000AAAA

*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*

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 1000;DATA:FIELD INPUT;COPY:TO 200;FIELD INPUT;EXECUTE

This command copies 1000 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.

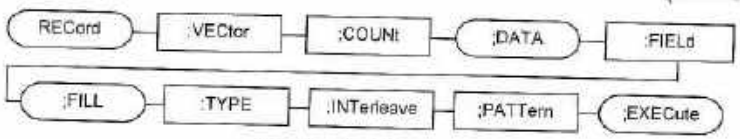
RECORD:VECTOR 1,COUNT ALL;DATA:FIELD INPUT;COPY TO 1;FIELD TST_DATA;EXECUTE

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't care 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 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 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

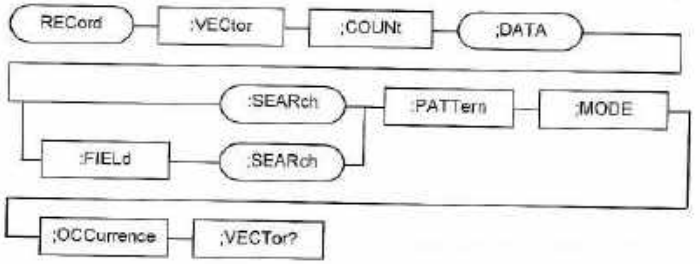
Examples RECORD:VECTOR 1;COUNT 100;DATA:FIELD IN_DATA;FILL:TYPE INCREMENT,INTERLEAVE 2,PATTERN #H0000;EXECUTE

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 SR5020 I/O module contains 18 high resolution timing generator channels which are used to control stimulus edge placement and sample timing. Sixteen 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 on the SR5020. Each group of 8 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.

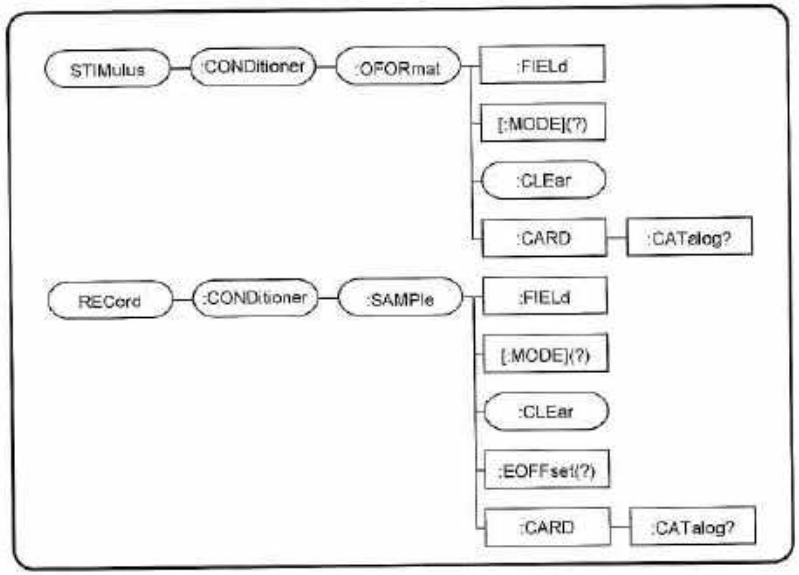
Note

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 SR5000 does not actually allow manipulation of timing generator channels by number.

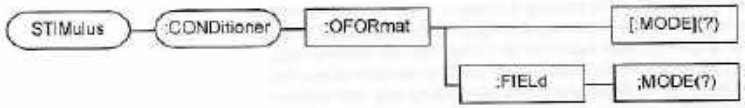
Timing generator channels are always internally paired in groups of 2, meaning that timing generator channels 0 and 1 will always form one pair, timing generator channels 2 and 3 will always form another, and so on. Even numbered timing generator channels are always used to provide delay times, and odd 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 8 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 SR5000'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 SR5000, a command error will be generated and the red "ERROR" LED on the SR5010 front panel will be illuminated. Use the SYSTEM:ERROR? query to read the error condition and clear the LED.



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 - 100ps))
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 - 100ps))
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,5.000000e-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 negative 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 8 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 - 20); up to the number of I/O modules installed in the test system.

ALL = All the SR5020 I/O Modules installed in the SR5000 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 SR5020 module specified.

Parameter Definition

stimulus_pin = C<card_num>P<pin_num>
card_num = (1 - 20)
pin_num = (1-32)
pin_format = < NRZ | RZ | RONE | RTC | RI >
delay_value = (0 - (clk_per - 100ps))
pulse_width = (10ns - (clk_per - 10ns))

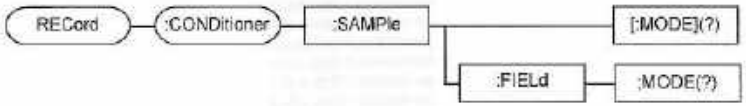
For delay_value and pulse_width, units are in seconds and are always returned in scientific notation. The parameters delay_value and pulse_width are not required to fit within one clock cycle.

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 - 100ps))
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 - 100ps))
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,3.000000e-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 choose 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 SR5020 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.

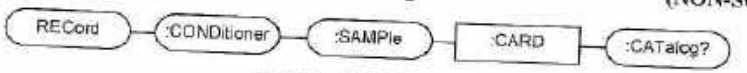
Note
The **FIELd** parameter changes the destination field only for the same command but **does not** change the default field.

: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. Refer to Chapter 5, "Logical Addressing" for information pertaining to SR5020 card numbers.

Parameter Definition

card_num = (1 - 20), up to the number of I/O modules installed in the test system.

ALL = All the I/O Modules installed in the SR5000 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 - 20)
pin_num = (1-32)
sample_mode = < EDGE | WIND >
delay_value = (0 - (clk_per - 100ps))
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 140)

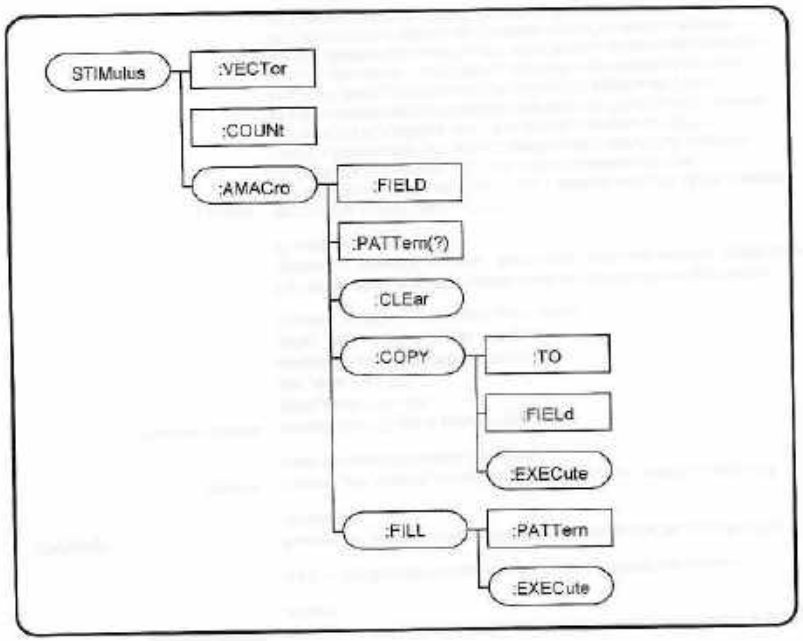
Algorithmic Pattern Generation

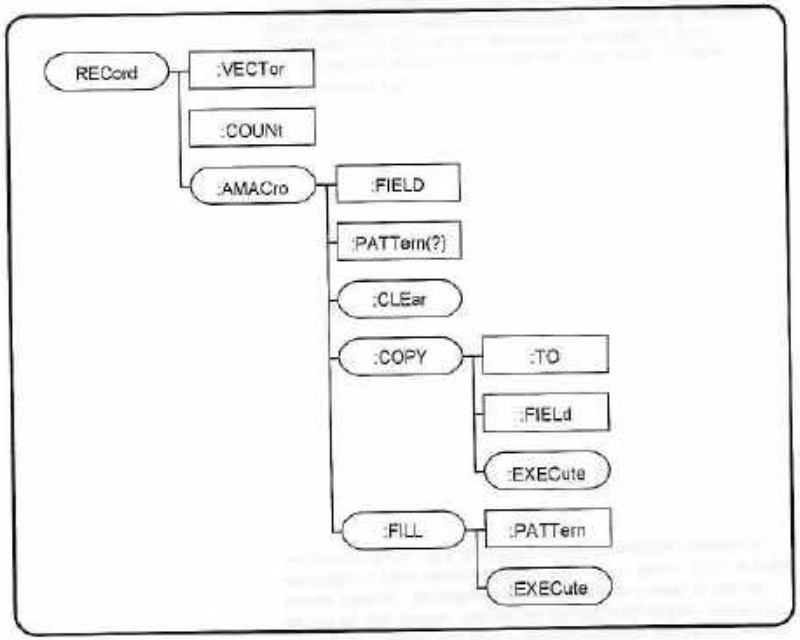
Algorithmic commands allow real-time generation of stimulus patterns and expected responses based on simple functions. Used with CMACRO vector looping commands, algorithmic pattern generation allows extremely long patterns to be generated using very few actual vectors, and, they may be generated at full 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.

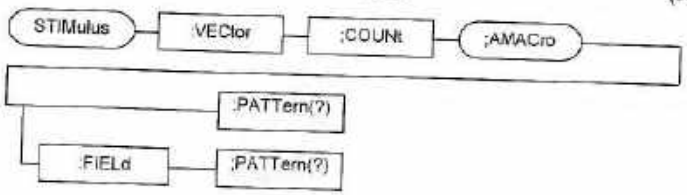
(CONTINUED FROM PAGE 139)

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 SR5010 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}

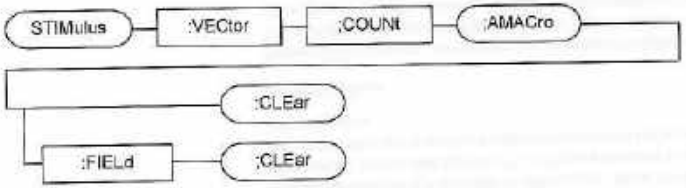
Parameter Definition **algo_macro = < NONA | INC | DEC | XOR | HOLDD | HOLDA | SLEFTZ | SLEFTO | SLEFTC | SRIGHTZ | SRIGHTO | SRIGHTC | RLEFT | RRIGHT | LOADP | OUTPUTP >**

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 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 (£ 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.

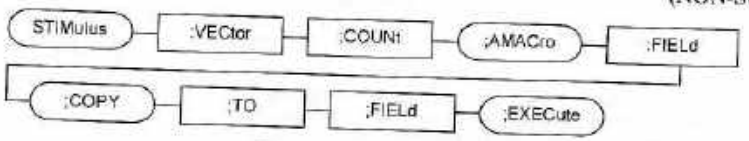
Parameter Definition name = Any alphanumeric string and '_' (max 8 characters).

: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 (£ 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 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 201-300. 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.

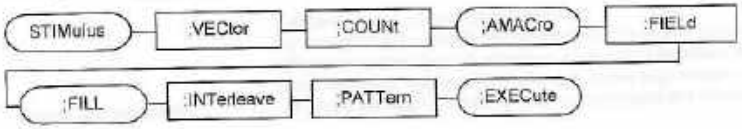
STIMULUS:VECTOR 1;COUNT ALL;AMAC;FIELD ADDR;COPY:TO 1;FIELD DATA;EXECUTE

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

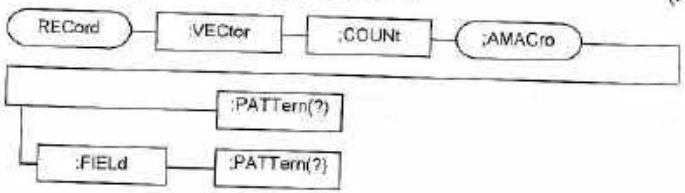
Examples STIMULUS:VECTOR 1;COUNT 100;AMACRO:FIELD ADDR;FILL:INTERLEAVE 1;PATTERN INCREMENT;EXECUTE

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 (£ 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 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 SR5010 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.

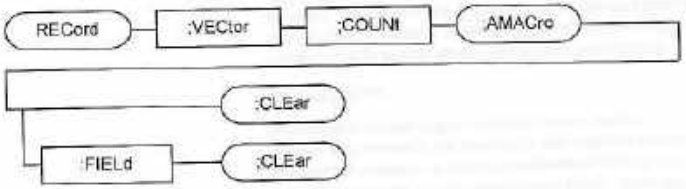
Response alg_macro[,alg_macro]

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 (£ 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 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

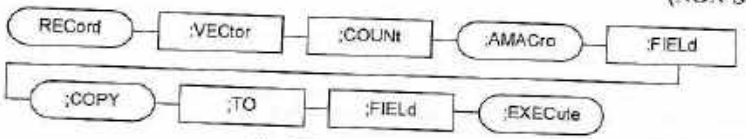
Causes the algorithmic macro commands for the specified field to be cleared to NONA.

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,

and EXECute.

;COPY

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 (1 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 10)

;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 201-300. 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.

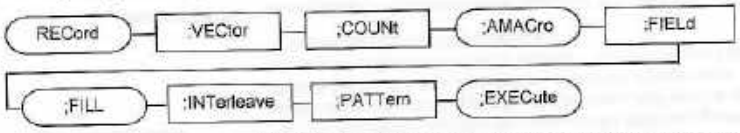
RECORD:VECTOR 1;COUNT ALL;AMAC:FIELD ADDR;COPY:TO 1;FIELD DATA;EXECUTE

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.

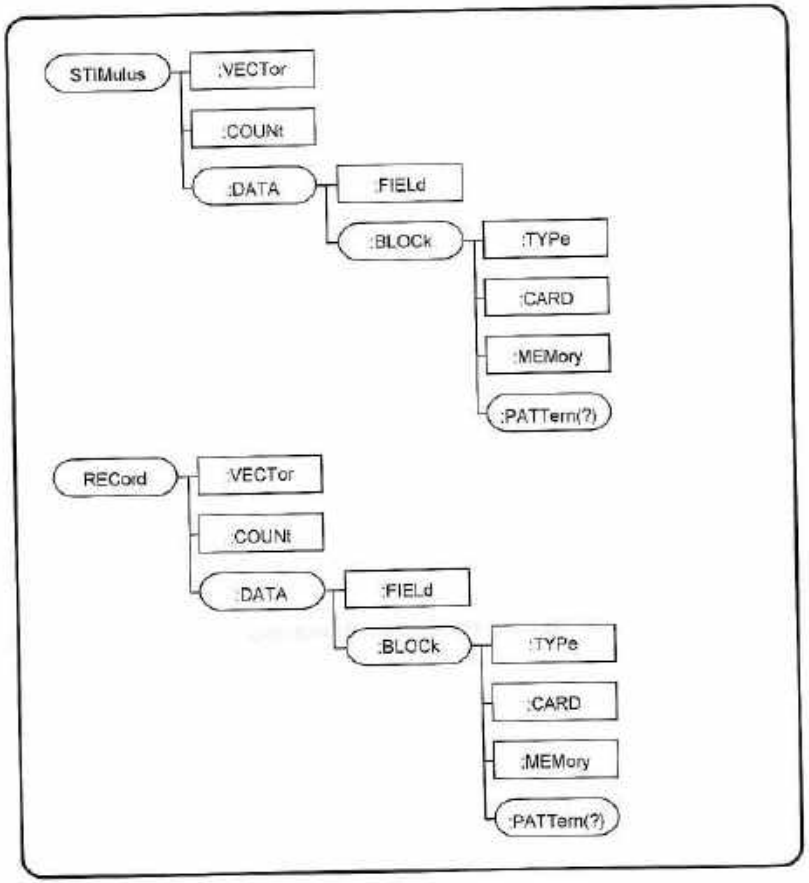
THIS PAGE INTENTIONALLY LEFT BLANK

High Speed Binary Pattern Transfers

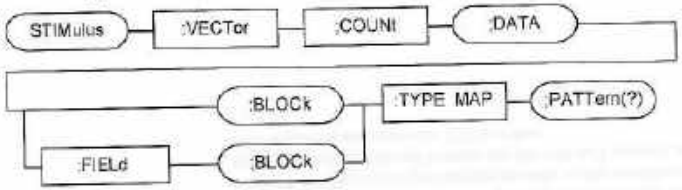
Data patterns may be loaded into memory in binary format instead of using the SCPI command strings. This provides a significant speed advantage over text based transfers for 3 reasons. In ASCII text based transfers, each character transferred requires multiple VXI read/write cycles. This is due to the 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 transfered 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 SR5010 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 SR5010 A32 memory using D8, D16 or D32 transfers. The SR5010 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 SR5010 to transfer the data in its A32 memory to I/O memory. There are two methods for the SR5010 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 SR5010 A32 memory and the I/O module pattern memories use D32 transfers.



Stimulus Mapped Binary Patterns (NON-SCPI)



The STIMulus:;;:BLOCk:TYPE MAP;PATTern command downloads the contents of the SR5010 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 SR5010 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 (≤ 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

: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?

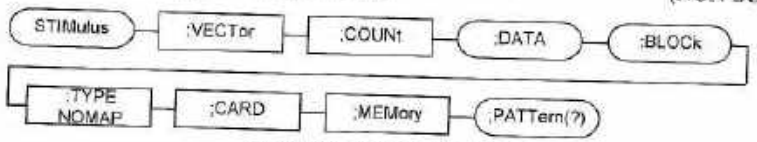
The PATTern string terminates the command and executes the block transfer from the stimulus memory field to the SR5010 A32 memory..

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 SR5010 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 SR5010 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 (≤ 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 SR5010 A32 memory will be transferred (memory write) to "ALL" the I/O cards. If "ALL" I/O cards are selected for transfers to the SR5010 A32 memory (memory read), then the SR5010 A32 memory will contain only the data contents of the last I/O card.

Parameter Definition **card_num** = (1 - 20); up to the maximum number of I/O cards installed in the SR5000 system.

ALL = All I/O cards installed in the SR5000 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?

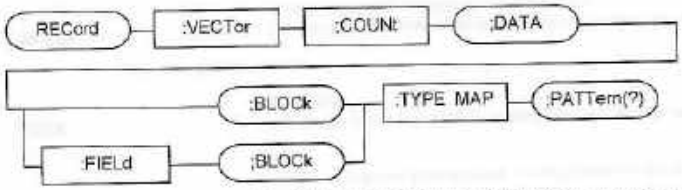
The PATTern string terminates the command and executes the block transfer from the stimulus memory to the SR5010 A32 memory.

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 SR5010 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 SR5010 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 (≤ 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 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 SR5010 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

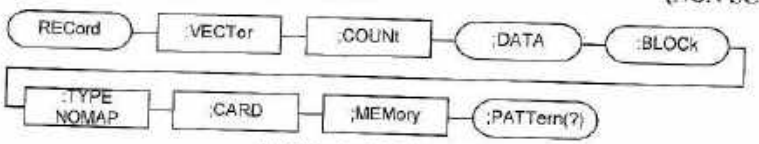
:PATTern?

The PATTern string terminates the command and executes the block transfer from the stimulus memory field to the SR5010 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 SR5010 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 SR5010 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 SR5010 A32 memory will be transferred (memory write) to "ALL" the I/O cards. If "ALL" I/O cards are selected for transfers to the SR5010 A32 memory (memory read) , then the SR5010 A32 memory will contain only the data contents of the last I/O card.

Parameter Definition **card_num** = (1 - 20), up to the maximum number of I/O cards installed in the SR5000 system.

ALL = All I/O cards installed in the SR5000 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 SR5010 A32 memory..

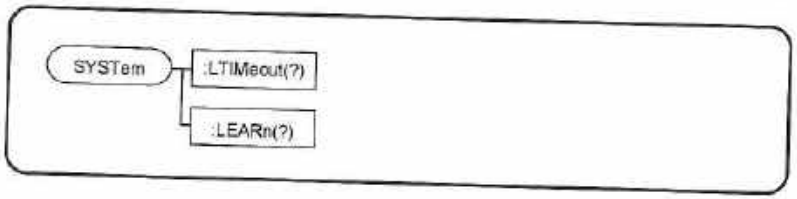
Response none

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?
```

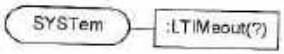

Saving and Loading Tests

Two methods exist to read and save a test from the SR5010 to a slot 0 controller. The first method, which will not be discussed, is to query each individual parameter in the SR5000 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 SR5010 A32 memory. Each time a SCPI command is sent to the SR5010, 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 SR5000 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 SR5010, and the results stored on the SR5010 and the I/O modules. LEARn? query and LEARn allow you to read the compiled test program from the SR5000 system, or write a previously read compiled test back to the SR5000 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.



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 SR5010 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 SR5010 to the slot 0 controller (LEARn?) or from the slot 0 controller to the SR5010 (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 SR5010 to send the current system setup parameters to the slot 0 controller as a binary memory-image file. The SYSTem:LEARn command instructs the SR5010 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 SR5000 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 SR5010 A32 memory by the slot 0 controller, one block for each handshake cycle, to be read by the SR5010.

Parameters none

Examples
```
SYSTEM:LEARN
SYST:LEAR
```

:LEARn? Instructs the SR5000 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 SR5010 A32 memory by the SR5010, one block for each handshake cycle, to be read by the slot 0 controller.

Parameters none

Examples
```
SYSTEM:LEARN?
SYST:LEAR?
```

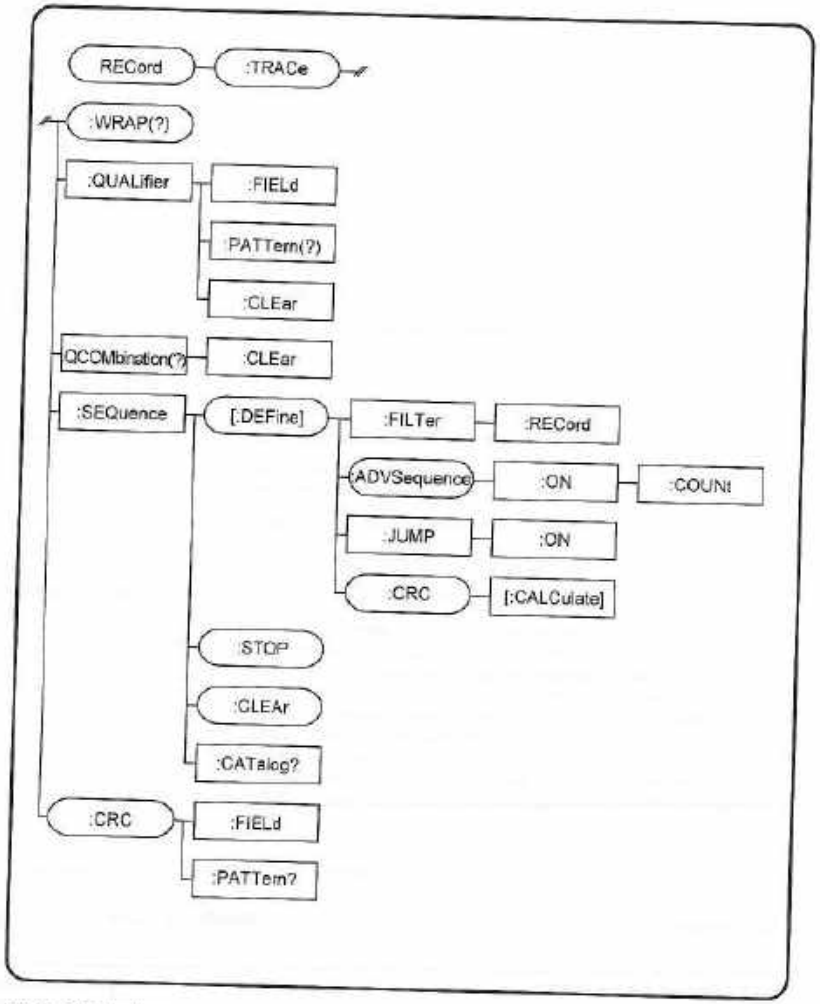

Advanced Record Triggering

The SR5000 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 SR5000 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:POSTrigger 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 SR5000. 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 SR5010 Input Flags for record and CRC control, in addition to the Qualifiers and Qualifier Combinations.



Qualifier Trigger Patterns			
QUAL#	ADDR	DATA	R/W*
1	#h2000	#bXX	#b01
2	#h2000	#bXX	#b10
3	#h7FFF	#bXX	#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	2	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 SR5000 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 #4. Sequence #4 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 SR5000 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 SR5000 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 SR5000 record wrap-around.

Response 0 | 1

Parameters

1 = Record wrap-around is enabled.

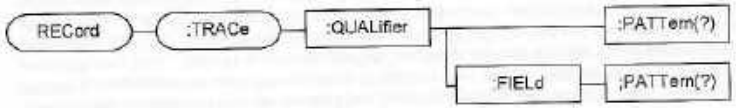
0 = Record wrap-around is disabled.

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).

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)



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

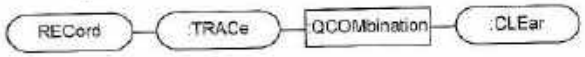
:QCOMbination<qcom_num>? Queries the Qualifier(s) defined for the specified Qcombination.

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 all the Qualifier Combinations of all of the field trigger patterns assigned to it, (QCOM 1-8)

:QCOMbination<qcom_num> The qcom_num parameter is placed immediately after the QCOMbination command without a space between the two. This is followed by a space and the number 8.

Parameters **qcom_num** = (1 - 8)

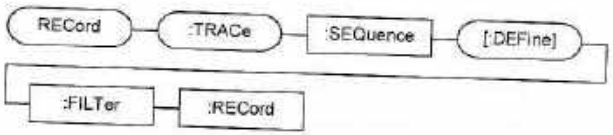
:CLEar Clears the specified Qualifier Combination.

Parameter Definition none

Examples
RECORD:TRACE:QCOMBINATION1 8:CLEAR
REC:TRAC:QCOM1 8: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. 3-182.

: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.

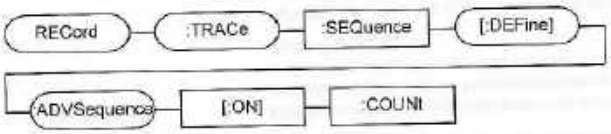
QCOM = Save the specified data to the record memory whenever one or more of the qualifier trigger patterns, as specified by the qualifier combination, evaluates true, i.e., matches the input data from the UUT.

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. 3-182.

:SEQuence <seq_num | ALL>

Parameters

Defines sequence level that the command parameters will apply to.

seq_num = (1 - 16)

ALL = Specifies all 16 sequence levels.

[:DEFine]

Parameter Definition

Provide the path into the sequence definition subsystem. DEFine is the default path, so the DEFine command may be omitted.

none

:ADVSequence

Parameter Definition

Provide the path into the advance sequence definition subsystem.

none

[:ON] <NEVer | CLOCk | COMpare | NCOMpare | QCOM<qcom_num>>

Parameter Definition

Specifies when the selected information is saved to the record memory. The ON command is optional and may be omitted for convenience.

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

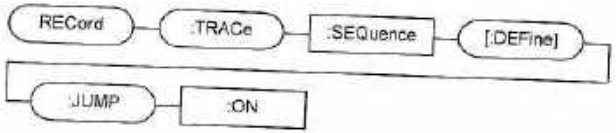
advs_count = (1 - 65535)

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. 3-182.

: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.

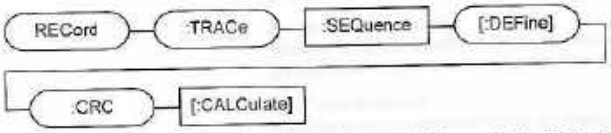
NCOMpare = Advance to the next sequence level after the real-time compare condition evaluates false for the number of cycles specified by the advs_count parameter.

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. 3-182.

: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.

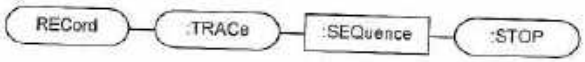
NCOMpare = Calculate the CRC whenever the real-time compare is false. The real-time compare state is the dynamic result of comparing the input data from the UUT to the current vectors expected data pattern.

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 SR5000 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)

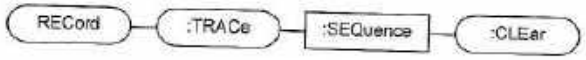
ALL = Specifies all 16 sequence levels.

: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

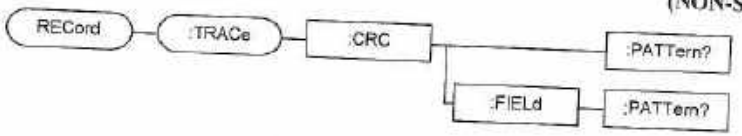
RECORD:TRACE:SEQUENCE 1:CATALOG?
1 FIL DAT QCOM1,CRC NEV,ADVS QCOM1 1,JUMP 1 NEV

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 - 20) - SR50XX module number.
pin# = (1 - 32) - SR50XX pin number.
crc_checksum = #h{(0-F)}[{,#h{(0-F)}}]

Examples RECORD:TRACE:CRC:FIELD DATA;PATTERN?
#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 #H1634,12 C1P12 #HC3D3,11 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

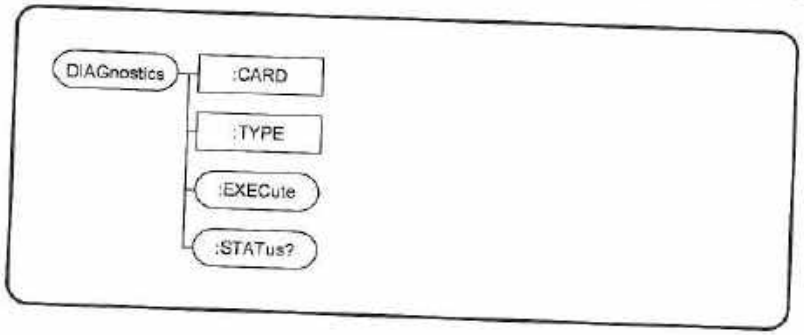
Miscellaneous Commands

This section includes the miscellaneous commands that did not fit anywhere else. These commands provide the ability to run diagnostics tests of the SR5000 system, and query the results of the test, query the status of the SR5000 system, specify the conditions in which the SR5000 will generate a Service Request Interrupt to the Slot-0 Controller, and define the SR5000 Bus Master time-out period. Also included in this section are the 488.2 mandatory commands. This section is divided into the following subsections:

- *SR5000 Diagnostics* ..... *pg. 3-206*
- *Status Queries, Status Interrupts, and System Queries* ..... *pg. 3-212*
- *Bus Master Time-out* ..... *pg. 3-224*
- *IEEE 488.2 Commands* ..... *pg. 3-226*

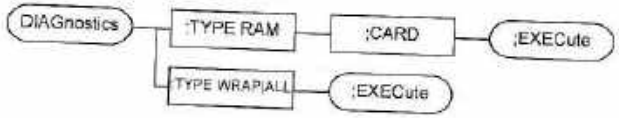
Diagnostics

The SR5000 provides the ability to execute an internal function test called Diagnostics. The diagnostics command allows you to specify which subsystem in the total SR5000 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.



Diagnostic Test Execution

(NON-SCPI)



The DIAGnostics command executes the SR5000 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 SR5010 Timing/Control Module:

- Control Gate Array Register Read/Write test
- Stimulus Command Memory test
- Stimulus Parameter Memory test
- EEPROM Checksum test

The RAM Test performs the following SR50XX I/O Module tests:

- Stimulus Gate Array Register Read/Write tests
- Response Gate Array Register Read/Write tests
- Output Memory test
- Tristate Memory test
- Expected Memory test
- Dontcare Memory test
- Record Memory test
- Stimulus Algorithmic Memory
- Response Algorithmic Memory test
- EEPROM Checksum test

The Wraparound Test executes a comprehensive diagnosis of the SR50XX 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 SR5000 is placed in the RUNNING mode and therefore all SR50XX 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 SR5000 card number will be tested, where "0" is the SR5010 Timing/Control Module and "1" through "20" is the card number of the SR50XX I/O Modules. The card number can also be specified by the literal string "ALL", where all SR5000 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 SR50XX I/O modules will be tested, regardless of the CARD number specified.

Parameter Definition **card_num** = (0 - 20)

ALL = All SR5000 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)

DIAGnostics — :STATus?

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 - 10), 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

Features

- **32 Input and 32 Output Pins Per Module.**
- **64K Vectors per Channel.**
- **RAM-Backed & Algorithmic Pattern Generation.**
- **NRZ, RZ, RONE, RTC and RI Output Data Formats Supported.**
- **16 Stimulus Timing Generators & 2 Response Timing Generators Per Module.**
- **100 ps Edge Placement Resolution.**

Seven I/O Memory Types

The SR5020 I/O Module contains seven separate memory banks, each 64K vectors in depth, for generating stimulus patterns, expected response patterns and recording UUT response data. All memory banks operate at the programmed data rates, up to 50 MHz.

The *Stimulus Memories* consist of the Output, Tristate, and Algorithmic Output Memories. The *Output Memory* contains the actual data patterns to be generated to the UUT. *Tristate Memory* provides tristate control that supports bidirectional I/O. The *Algorithmic Output Memory* determines which algorithmic function will be applied to the output.

The *Response Memories* consist of the Expect, Mask and Algorithmic Expect memories. The *Expect Memory* contains the expected response for the UUT and is the basis for input compare operations. The *Mask Memory* determines which bits in the pattern are to be ignored for input response comparison. *Algorithmic Expect Memory* determines which algorithmic expect function will be used in comparing against the UUT response.

Record Memory is used to store either the UUT response data or the result of the comparison between the UUT response data and the expected response pattern.

Data Formatting with Precise Edge Placement

Stimulus pins may be independently programmed for any of the following data formats: Non Return to Zero (NRZ), Return to Zero (RZ), Return to One (R1), Return to Complement (RC) and Return to Inhibit/Tristate (RI).

Each SR5020 I/O Module contains 16 timing generators for stimulus edge placement and pulse width timing. In each Stimulus Gate Array, logic is provided for the eight output pins to select one of the four delay clocks to assert the output (NRZ mode) or select one of two delay clock pairs to define the assert and deny times for the output (return-to-modes). 100 ps edge placement resolution provides precise UUT timing for bus emulation testing, memory testing, and functional testing.

Controls and Indicators

See Fig. 1-1. All the connectors and LEDs for the SR5020 I/O module are located on the front panel.

LEDs

There are two LEDs located at the top of the SR5020 I/O Module:

- **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. The error code informa-

- TESTS - MAX OF 32 DEFINED TESTS
- I/O MEM SIZE : 64K
- SOIB : TRIG, SYST TRIG, CMACRO SEQ PROGRAMS
  - STIM & RESP PATS : I/O MODULE
- FREQ : 400 Hz to 50MHz, 25MHz DEFAULT
- SYST : FREQ = VECTOR RATE = DATA RATE
- INTERNAL GATE ALWAYS ENABLED
- FIELDS 1) RENUMBERED 2) ALGORITHMIC
  - DELETING FIELDS
    - DEL ALL OUTPUT FORMATTING
    - DEL ALL INPUT SAMPLE TIMING
    - DEL ALL DATA PATTERNS
- TESTS : WHEN DELETED, MEM IS DYN RECALOC TO MAINTAIN CONTIGUOUS FREE BLOCK
- FIELDS DEFINED : ALL MEMORIES SET TO DEFAULT VALUES
- POST-TRIGGER TIMERS - LOSS RECORD PRICE

- TEST SIZE : EVEN #, MAX: 65,500 VECTORS
* TEST 65,500 VECTORS (R/W)
* TEST DEFINE 32 TESTS MAXING 64K 32x TEST:DEF xx:SIZE VECTORS
* CHECK MAX AVAIL & START TEST:FREE?
* CHECK CLK MIN & MAX, DEFAULT
Δ CONTROL RESOLUTION : COULD 90° PHASE SHIFT CLK BE AFFECTING WRITE TES WITH 2-CLK OFFSET?
* TEST EXT GATE TO START/HALT EXEC
* TEST EXT TRIGGER TO START EXEC
  TRIG : SOURCE EXT
Δ CREATE 2 FIELDS, 1) OUT 2) TRI
  - USE DIFFERENT NAMES, SAME CIP32-1
  - DOES SAME AS OT IN 2 STEPS?
* TEST CONTROL MEMORIES, CHANGE CMACRO PROGRAMS CALL 32K x 52-bit MEM BLOCK
Δ FIND DEFAULT CMACRO PROGRAM FOR RAM-BASED OPERATIONS
□ RECORD : 3-153, 3-76, 3-84*
Δ QUERY CMACRO AFTER TEST TO SEE PROGRAM RAN?

THIS PAGE INTENTIONALLY LEFT BLANK

Status Queries, Status Interrupts and System Queries

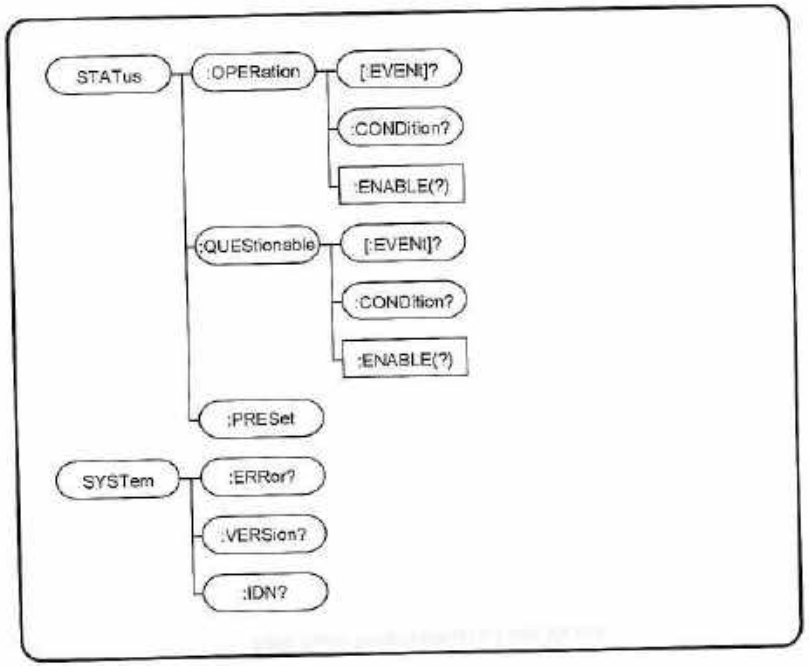
The SR5000 includes the Status Reporting mechanisms described in chapter 11 of IEEE 488.2 and the SCPI-defined Operation Status Register and Questionable Data/Signal Status Register. The SR5000 does not define any functions or conditions for the Questionable Status Register, however, the register reporting 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 SR5000, as well as defining operational events which will cause an interrupt to the slot 0 controller to be generated. The operational condition of the SR5000 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 SR5000 has extensive command error checking built into the command parser. When a command error occurs, the ERROR LED on the front panel of the SR5010 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 SR5000 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 SR5000 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.



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 SR5000. 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.

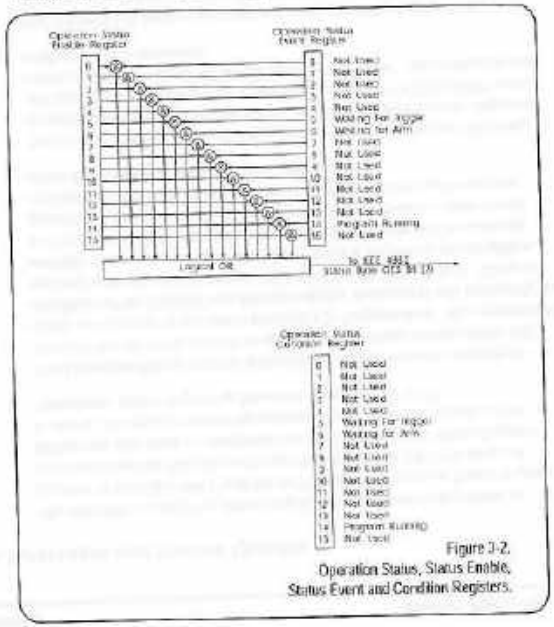


Figure 3-2.
Operation Status, Status Enable, Status Event and Condition Registers.

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 SR5000. 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 SR5000. 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.

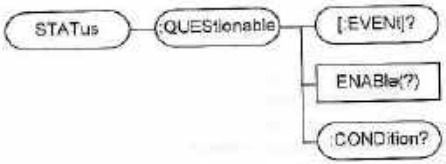
Response cond_reg

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 SR5000; 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 SR5000, but no bits are used.

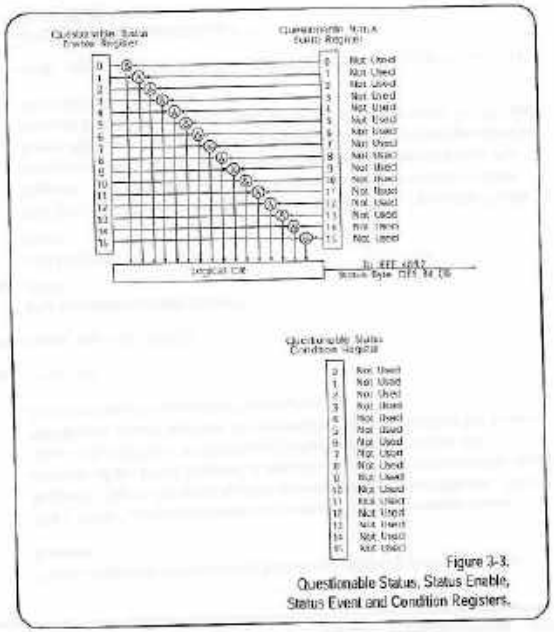


Figure 3-3.
Questionable Status, Status Enable, Status Event and Condition Registers.

:[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.

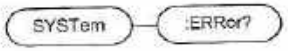
:PRESet

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)



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 SR5010 will illuminate. The error status is returned as an error code number followed by a descriptive string. Refer to Appendix A 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 SR5010.

:ERRor?

Response err_code,err_string

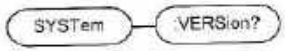
Parameter Definition **err_code** = -(101 - 499)

err_string = See Appendix A 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 SR5000.

: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)

SYSTem — :IDN?

The SYSTem:IDN? query command returns the SR5000 system configuration. The system configuration includes information about the SR5000 including: software revision number, and the model numbers, RAM options, logical addresses, and serial numbers for the SR5010 Timing/Control Module and the SR5000 I/O Modules.

:IDN?

Response sw_rev,SR 5010VXI-50MHz(64K,vme_mem_size)-cc_log_addr-cc_serial_num [{,io_mod_num(io_ram_size)-io_log_addr-io_serial_num

Parameter Definition

sw_rev = XX.X, where XX.X represent the software revision number of the SR5010.

vme_mem_size = (0M | 1M) The SR5010 Timing/Control module A32/D32 VME Memory size.

cc_log_addr = (0 - 255) The SR5010 Timing/Control module VXI logical address.

cc_ser_num = (YYYYYYYYYY-YYYY), where YYYYYYYYYY-YYYY represent the SR5010 Timing/Control module serial number.

io_mod_num = SR 50YYVXI-ZZZIO, where YY is the differentiating model number and ZZZ is the IO logic level type.

io_ram_size = (16K | 64K) The I/O module vector memory size.

io_log_addr = (0 - 255) The I/O module VXI logical address.

io_ser_num = (YYYYYYYYYY-YYYY), where YYYYYYYYYY-YYYY represent the I/O module serial number.

Examples

SYSTEM:IDN?
1,33,SR 5010VXI-50MHz(64K,1M)-014-0293421651-2a05,SR 5020VXI-TTLIO(64K)-023-0293411656-2a09

SYST IDN?
1,33,SR 5010VXI-50MHz(64K,1M)-014-0293421651-2a05,SR 5020VXI-TTLIO(64K)-023-0293411656-2a09,SR 5030VXI-VVTIO(64K)-024-0294121664-2a03

Bus Master Time-Out

Each time the SR5000 parses a command in which the data portion of the command is intended for an I/O module, the SR5010 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 SR5010 must be able to detect when an I/O module is responding to the read and write cycles initiated by the SR5010. 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 SR5000 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 SR5000 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

IEEE 488.2 Commands

The SR5000 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-

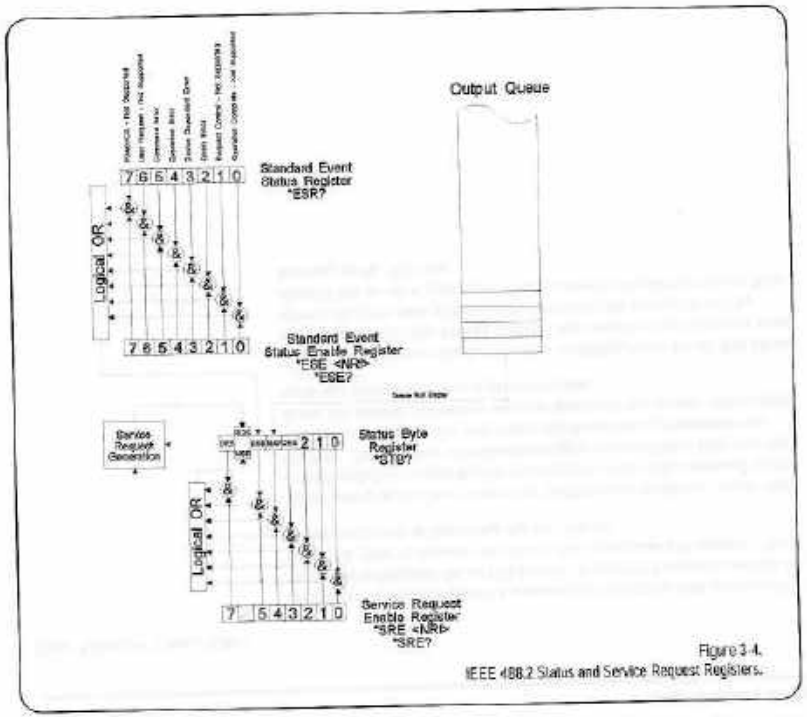


Figure 3-4.
IEEE 488.2 Status and Service Request Registers.

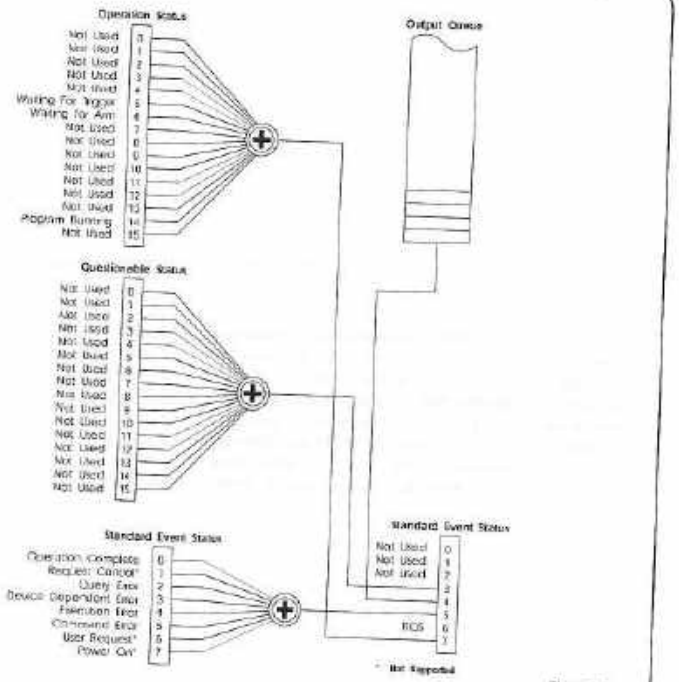
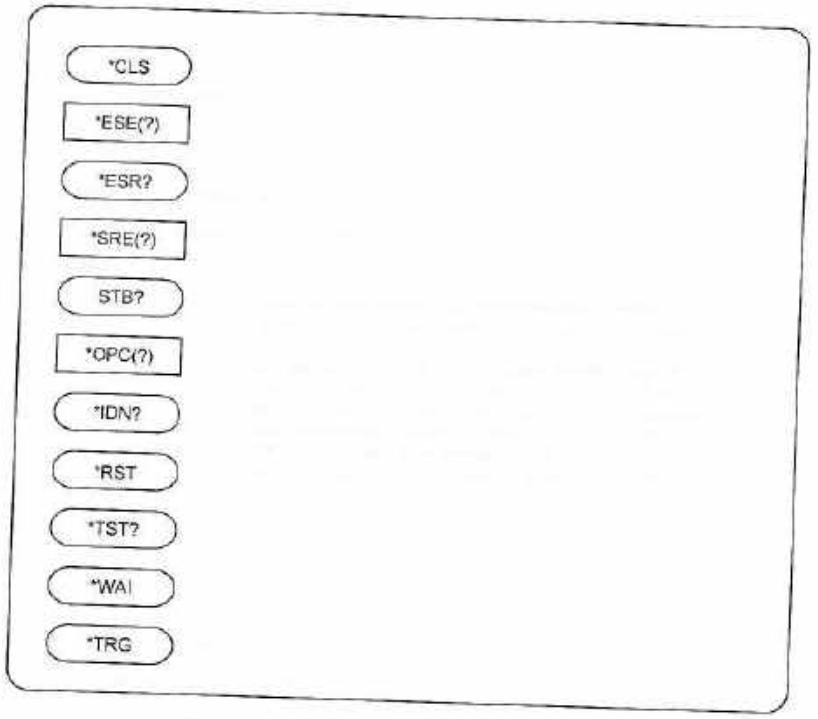


Figure 3-5.
IEEE 488.2 Status and Service Request Registers

ters is required. For this reason, a functional diagram of the 488.2 registers is shown below (figure 3-4), and a dingram 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.



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 SR5000 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 SR5000.

***IDN?**

Response INTERFACE TECHNOLOGY,SR 5010VXI-50MHz(64K,vme_mem_size),cc_serial_num,sw_rev

Parameter Definition **vme_mem_size** = (0M | 1M) The SR5010 Timing/Control module A32/D32 VME Memory size.

cc_ser_num = (YYYYYYYYYY-YYYY), where YYYYYYYYYY-YYYY represent the SR5010 Timing/Control module serial number.

sw_rev = XX.X, where XX.X represent the software revision number of the SR5010.

Example *IDN?
INTERFACE TECHNOLOGY,SR 5010VXI-50MHz(64K,1M),029342165 I-2a05,1.33

*RST

The *RST command resets the SR5000. Any running test will be stopped and the SR5000 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 SR5000. The *WAI is parsed by the SR5000 but no action is taken.

***WAI**

Parameter Definition none

Example *WAI

THIS PAGE INTENTIONALLY LEFT BLANK

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 SR5000, 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 SR5000 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 sub-systems 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 SR5000 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 SR5000 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 SR5000. 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

Qualifier Combinations	1	2	3	4	5	6	7	8

Qualifier Trigger Definitions								
Fields →	1	2	3	4	5	6	7	8

Flow 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 SR5000 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:	OIT
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 SR5000 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 generated from the tables is listed below. To generate the test program defined by this example, the SCPI commands shown below, minus the comments, are sent to the SR5010. 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
```

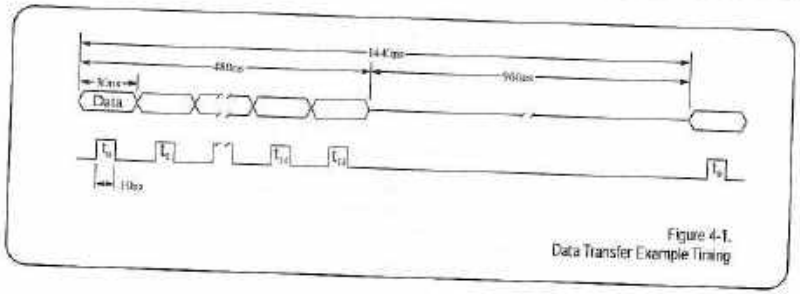


Figure 4-1.
Data Transfer Example Timing

Table 4-7: Data Transfer Test and Field Definitions

Test and Field Definition Worksheet						
Test Definition Parameters						
Name:	TEST_2	Site:	20	Program Loops:	1	10 MHz Ref:	N/C
Clock Source:	N/C	Frequency:	33.33 MHz	Clock Slope:	N/A	Clock Level:	N/A
System Trigger:	EXT	Trigger Slope:	POS	Trigger Level:	2.2V	Arm Data:	OFF
Data Source:	N/C	Alarm 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:	10ns

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				0120	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

SR5000 command macro (CMACRO) programs add another level of flexibility to an SR5000 based test, the ability to loop on patterns and change test program flow. Looping and branching may be done unconditionally or conditionally. The SR5000 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 33.33MHz
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,10.000000NS

/* 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 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,0

/* 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*/
```

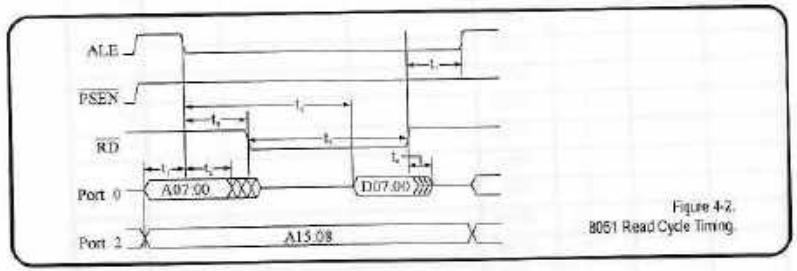


Figure 4-2.
8051 Read Cycle Timing

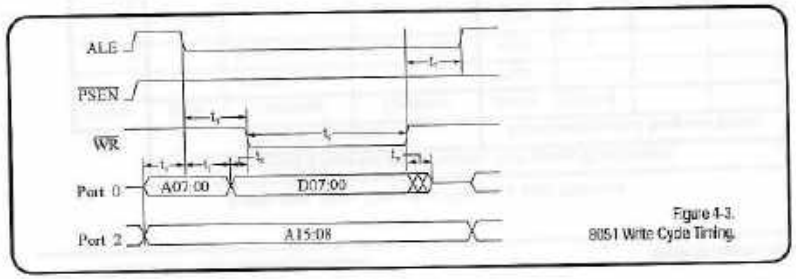


Figure 4-3.
8051 Write Cycle Timing

Table 4-9: 8051 Timing Parameters

Symbol	Parameter	Min	Max	Unit
t1	Address Valid to ALE Low	28		ns
t2	Address Hold After ALE Low	48		ns
t3	ALE Low to RD/WR Low	200	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 SR5000, 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 SR5000 by testing the RAM on a microprocessor based circuit board. A brief discussion of the UUT will help understand the application of the SR5000 to the test.

The UUT is a circuit board based on the Intel 8051 microcontroller. Refer to the read and write timing diagrams, the SR5000 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 SR5000 test rate may be programmed anywhere within the two ranges. In this case, the SR5000 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 SR5000, 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.

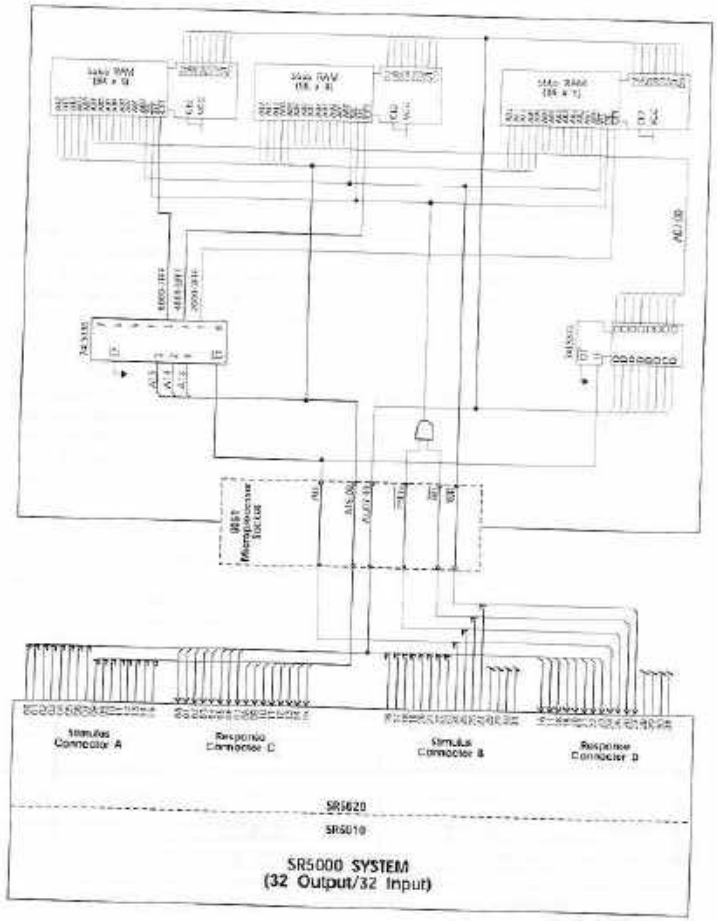


Figure 4-4,
8051 Emulation Wiring Schematic.

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
Data 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: C1P24-17
Arm Pattern:	FF	Format:	RI	Assert:	200.0ns	Pulse Width: 465.0ns
Field Name:	D07_00T	Field Type:	OT	Field Radix:	HEX	Pin List: C1P24-17
Arm Pattern:	FF	Format:	N/C	Assert:	N/A	Pulse Width: N/A
Field Name:	ALE	Field Type:	OT	Field Radix:	BIN	Pin List: C1P25
Arm Pattern:	0	Format:	RZ	Assert:	0.0ns	Pulse Width: 30.0ns
Field Name:	WRO	Field Type:	OT	Field Radix:	BIN	Pin List: C1P28
Arm Pattern:	1	Format:	RONE	Assert:	230.0ns	Pulse Width: 400.0ns
Field Name:	RDO	Field Type:	OT	Field Radix:	BIN	Pin List: C1P27
Arm Pattern:	1	Format:	RONE	Assert:	230.0ns	Pulse Width: 400.0ns
Field Name:	PSENO	Field Type:	OT	Field Radix:	BIN	Pin List: C1P26
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 SR5000 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=90	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 SR5000.

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 SR5000 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.

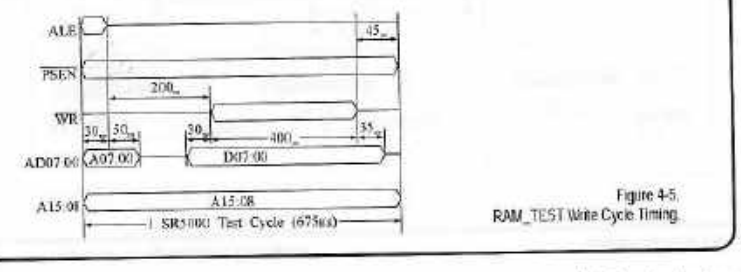


Figure 4-5
RAM_TEST Write Cycle Timing

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 C1P24-17
FIEL:NAME D07_00O:RAD HEX
FIELD:DEF D07_00T:TYPE OT:PIN C1P24-17
FIEL:NAME D07_00T:RAD HEX
FIELD:DEF ALE:TYPE OT:PIN C1P25
FIEL:NAME ALE:RAD BIN
FIELD:DEF WRO:TYPE OT:PIN C1P28
FIEL:NAME WRO:RAD BIN
FIELD:DEF RDO:TYPE OT:PIN C1P27
FIEL:NAME RDO:RAD BIN
FIELD:DEF PSEN:TYPE OT:PIN C1P26
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 (2^5) and beyond.

Finally, the last example demonstrated the process for determining stimulus timing parameters and how to program this information into the SR5000. 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 SR5000 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 SR5020 module allows the definition of 2 *sample times* within the test cycle, and all input functions are tied to one or the other of these sample times. In the case of the *Window Compare* mode, both sample times are used. When both sample clocks are used on a SR5020, *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 SR5000, 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:	C1P24-17
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:	C1P24-17
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:	C1P28
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:	C1P27
Arm Pattern:	N/A	Format:	Sample Edge	Assert:	550.0ns	Pulse Width:	N/A

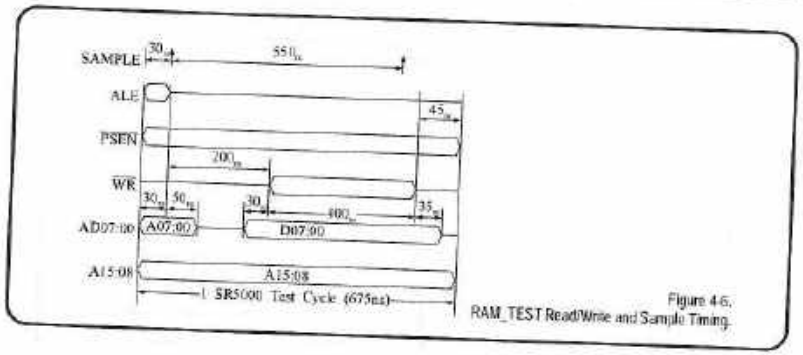


Figure 4-6.
RAM_TEST Read/Write and Sample Timing

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 SR5000 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	IF	NONA	IF
4		StartLoop until	COUNt==96	INC		INC	
5		WordLoop until	COUNt==254	HOLDA		HOLDA	
6		EndLoop		HOLDA		HOLDA	
7		OUT		NONA	IF	NONA	IF
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 C1P24-17
FIEL:NAME D07_00O:RAD HEX
FIELD:DEF D07_00T:TYPE OT:PIN C1P24-17
FIEL:NAME D07_00T:RAD HEX
FIELD:DEF ALE:TYPE OT:PIN C1P25
FIEL:NAME ALE:RAD BIN
FIELD:DEF WRO:TYPE OT:PIN C1P28
FIEL:NAME WRO:RAD BIN
```



```
FIELD:DEF RDO:TYPE OT:PIN C1P27
FIEL:NAME RDO:RAD BIN
FIELD:DEF PSEN:TYPE OT:PIN C1P26
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 C1P24-17
FIEL:NAME D07_00E:RAD HEX
FIELD:DEF D07_00M:TYPE ED:PIN C1P24-17
FIEL:NAME D07_00M:RAD HEX
FIELD:DEF WRE:TYPE ED:PIN C1P28
FIEL:NAME WRE:RAD BIN
FIELD:DEF RDE:TYPE ED:PIN C1P27
FIEL:NAME RDE:RAD BIN

/* Define Stimulus Data Formats and Response Sample Formats */

STIM:COND:OFOR:FIEL A07_00O;MODE R1,0.000000 E-9,80.000000 E-9
STIM:COND:OFOR:FIEL D07_00O;MODE R1,200.000000 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 SR5000 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 SR5000 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, 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 SR5000 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 SR5000 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 SR5000 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:	C1P24-17
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:	C1P24-17
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:	C1P28-27
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:	C1P28-27
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 SR5000 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 Combinations

	1	2	3	4	5	6	7	8
Qualifier 1	*							
Qualifier 2		*						

Qualifier Trigger Definitions

Fields ->	ADDRE	DATAE	WR_RDE
1	0x2000	XX	01
2	0x2000	XX	10
3			
4			
5			
6			
7			
8			

Trace Sequences

	Record	Filter	Record On	DRC	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	CCLOCK	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 SR5000 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 C1P24-17
FIEL:NAME DATA:RAD HEX
FIELD:DEF DATAE:TYPE ED:PIN C1P24-17
FIEL:NAME DATAE:RAD HEX
FIELD:DEF WR_RD:TYPE REC:PIN C1P28-27
FIEL:NAME WR_RD:RAD BIN
FIELD:DEF WR_RDE:TYPE ED:PIN C1P28-27
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.

CHAPTER 5

Installation

Scope of Chapter

This chapter contains instructions for unpacking, inspecting, installing, and checking out the SR5010 Timing/Control Module.

Unpacking and Inspection

Your SR5010 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 SR5010 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. 5-1.

Note

The logical address of the SR5010 Timing/Control Module must be set to a lower value than the logical address of the SR50xx I/O Modules to be controlled. If there is more than one SR5010 in a single VXI chassis, the address of an SR50xx I/O Module located between any two SR5010s will be part of the lower addressed SR5010's system. The SR50xx with the lowest numbered logical address is I/O card #1. The next highest SR50xx I/O Module logical address is I/O card #2. The highest SR50xx I/O Module logical address is the most significant I/O card number. To verify all I/O modules have been recognized by the system, send a "SYSTEM:IDN?" query command.

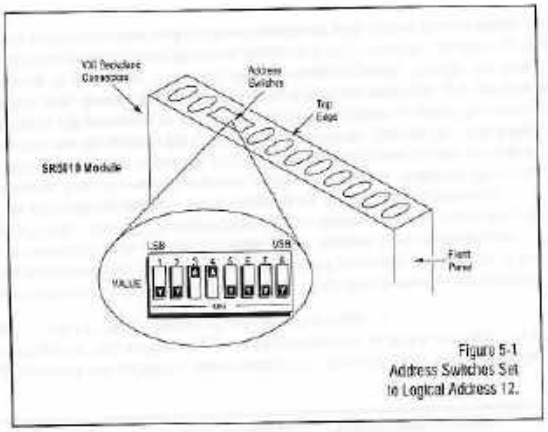


Figure 5-1
Address Switches Set to Logical Address 12.

Slot Dependency

The SR5010 Timing/Control Module may be installed in any available VXI chassis card slot except Slot-0 (occupied by the Slot-0 controller). The SR5000 subsystem uses Bus Master functions to identify the I/O Modules installed. This makes it possible to place the SR5010 in any slot within the VXI chassis, or even in another chassis connected by means of bus extenders such as the MXI bus.

Using Multiple Chassis.

If multiple chassis are used, the SR5010 interconnect cables connecting to the I/O Modules must be lengthened and the SR5000 system must be recalibrated using the longer cable sets. If an existing (single chassis) SR5000 system needs to be expanded into a multi-chassis configuration, it must first be returned to the factory for recalibration. Contact the Interface Technology Customer Service Department to arrange for retrofit and recalibration.

Backplane Jumpers

The SR5010 uses the BG3 and IACK signals from the VXI backplane, therefore these jumpers must be removed from the backplane for the slot that accommodates the SR5010 module. The BG0, BG1, and BG2 are not used on the SR5010, but are simply passed through. The user may remove or install the jumpers for these signals, as required.

Interconnections With Timing/Control Module

To connect the SR5010 Timing/Control Module with the SR50xx I/O Modules, do this (see Fig. 5-2 and 5-3):

1. Remove the side cover from the SR5010 Timing/Control Module.
2. If not already factory installed, thread the required number of SR5010 I/O Module interconnect cables through the access slot located on the front panel of the SR5010 Timing/Control Module (one cable per I/O module).
3. Plug the I/O module interconnect cables into their corresponding connectors located inside the SR5010 (Fig. 5-2).
4. Reinstall the side cover on the SR5010 module; insert the module into the VXI chassis.
5. Install the first SR50xx I/O Module into the VXI slot immediately to the right of the SR5010 Timing/Control module (Fig. 5-3).
6. Connect the free end of the shortest I/O interconnect cable from the SR5010 to the CONTROL connector on the SR50xx I/O module installed immediately to the right of the SR5010 module (Fig. 5-3).
7. Install the next I/O module into the VXI slot immediately to the right of the previously installed I/O module (step 5); plug the next shortest SR50xx I/O interconnect cable into the CONTROL connector of that I/O module.

8. Repeat step 7 for all remaining I/O modules.

9. Tighten the retaining screws at the top and bottom of all SR50x0 I/O modules, and also those of the SR5010 Timing/Control Module.

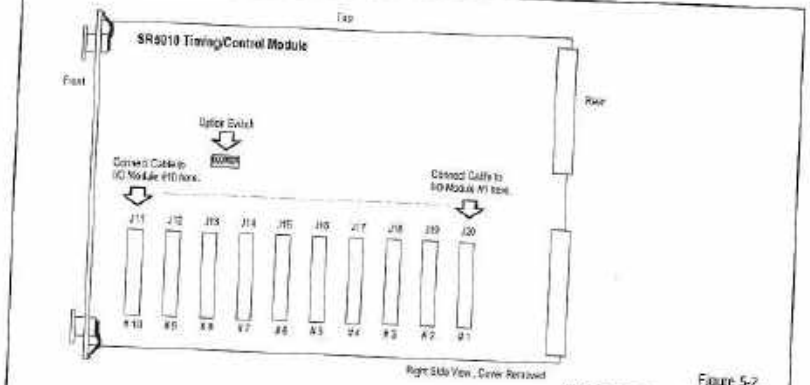
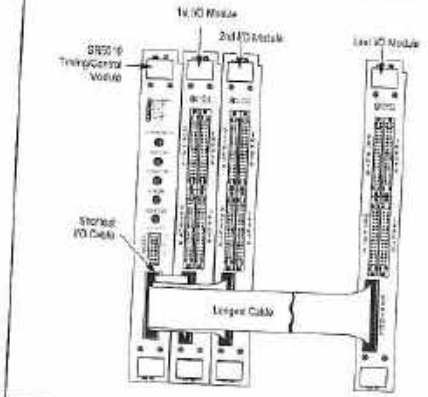


Figure 5-2.
SR5010 I/O Control Connectors.



] CAUTION [

Make these connections only with the cables that are supplied with the I/O modules (p/n 20011673). This cable has a special wiring arrangement that differs from standard flat cable. Failure to observe this precaution may cause erratic behavior and/or damage to the SR5000 subsystem.

Always turn power OFF before connecting or disconnecting interconnect cables. Failure to observe this precaution may result in damage to the SR5000 subsystem.

Figure 5-3
SR5010-to-SR50xx I/O Module Interconnect Cables.

Setting Self-Diagnostics Mode

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 (SW2) on the SR5010 circuit board, see Fig. 5-2. Table 5-1 is a listing of the self-diagnostics tests performed by the SR5010 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.

To set Option Switch SW2 for the type of test desired, do this:

1. Turn system power off; disconnect SR5010 I/O module interconnect cables from CONTROL connectors on front panel of I/O modules, remove SR5010 from VXI chassis.

Note

Do NOT remove SR5010 I/O Module interconnect cables from SR5010.

2. Remove SR5010 module from VXI chassis; remove the side cover from the SR5010.

3. See Fig. 5-2. 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 test.

4. Reinstall the SR5010 side cover; install SR5010 in VXI chassis.

5. Reconnect SR5010 I/O module interconnect cables to I/O modules.

Table 5-1. 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

© Copyright 1999 - 2005. All Rights Reserved

Interface Technology, Inc.
300 S. Lemon Creek Dr., Walnut, CA 91789
Tel: 909/595-8030 - Fax: 909/595-7177
e-mail: info@interfacetech.com - Internet: www.interfacetech.com

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP