Simutest

TECHNICAL PRODUCT DESCRIPTION

Verifier's Basic Features

First, let me highlight some of the basic features.

In a single, easy to use environment, Verifier combines:

- **Simulation Analysis,**
- **Test Program Development and,**
- **Offline Verification.**

In a **Simulation Analysis**, Verifier:

- Compares and reports difference between the same or different simulation runs.
- Displays test vectors as graphical waveforms.
- Analyzes simulation results for compliance with user-defined rules.
- Converts source simulation test vectors into a target simulator format.

In a **Test Program Development**, Verifier:

- Translates source simulation test vectors into the format of a target test system.
- Rule-checks the simulation output for tester compatibility and test methodology.
- Provides a migration path between source and target testers.
- Modifies simulator artifacts through conditioners.

" My former tool was expensive and slow. Verifier gives me both price and productivity.
Pattern conversions were reduced from 24 hours to 30 minutes and, even better, when using PLI playback my designers are now assured that patterns work with first time silicon without expensive debug time or ***rework."***

Mike Perugini,
Test Development & Automation Manager
Silicon Image

In an **Offline Verification**, Verifier:

- Creates test benches from test programs.
- Converts source test system test vectors into a target simulator format.
- Reports discrepancies between the same or different simulation runs.

Verifier's architecture is modular and expandable. So, as your design or test needs change, you can quickly add a simulator or tester interface module. See **Figure 1** for list of supported Simulator and tester modules

Features

With our extensive *Translator* utility, you can easily create test programs from simulation files.

- Rule-based *Analyzer* ensures simulation files' compatibility with your target test system(s).
- Exclusive Playback mode compares the test program to original simulation file.
- Special Verilog simulator uses the tester test bench.
- Built-in Verilog comparison identifies mismatches between simulated and expected results.
- Verifies design and test programs concurrently— all offline without tying up your expensive equipment – before first silicon.

If all this is true, just imagine how Verifier can help you streamline your operation! Read on!

VERIFIER ARCHITECTURE

Examine **Figure 1.** The rectangular boxes represent specific processes. The disk icons represent intermediate data. The arrows show the flow from one process module to the next. You can run each process individually or run them all with one mouse click.

The Simutest Verifier, is a complete offline system to generate, analyze, and debug test programs. Verifier contains the following modules:

- Analyze
- Translate
- Playback
- Simulate
- Compare

Each of these modules is described in the following pages.

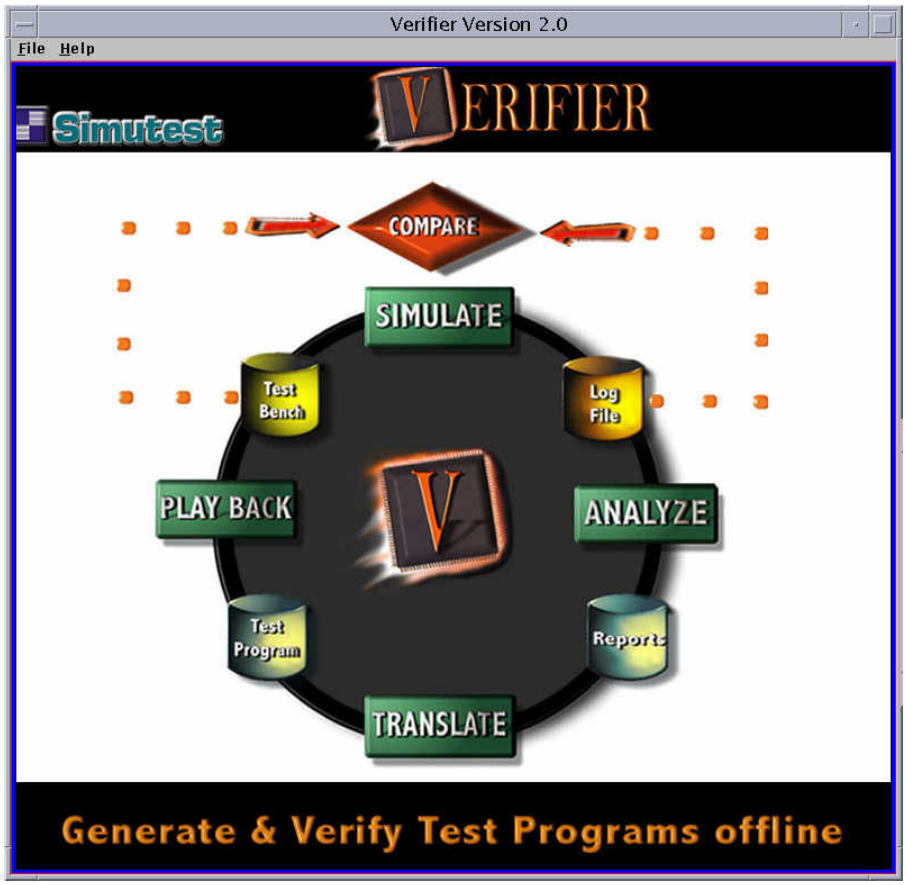


Figure 1. Verifier Architecture

ANALYZE MODULE

The purpose of the *Analyzer* is to verify both design and test programs. The *Analyzer* lets you compare simulated results with your own user-defined timing, test methods, and tester compatibility rules.

Examine **Figure 2**, which shows a typical list of rules.

Datasheet Timings	Test Methodologies	Tester Capabilities
Clock Period	Toggle	Pattern Memory
Set Up Time	Simultaneous Switch	Number of Pins
Hold Time	Signal Format	Maximum Frequency
Pulse Width	Detect Legal State	Number of Edges
Delay	Detect IO State	Number of Timesets
Pin Skew	Tri State Time	Formatting
Stability	IO Align	
	IO Switching	

Figure 2. Rule Description

When to use the *Analyzer*

Designers use the *Analyzer* to verify designs. You can analyze simulated results to detect pin-to-pin timing problems (such as, setup and hold times, glitches, and signal stability).

During test program development, prior to running *Translator* (discussed later), you can evaluate target tester compatibility. For example, simulated output will show you excess timing edges or timing sets that prohibit accurate conversions of simulator-generated test vectors into test programs.

ASIC designers can also use the *Analyzer* to meet foundry-specific test rules. Notice that **Figure 3** shows the files input to the *Analyzer* plus the resulting reports for display and/or printing.

The rule analysis process performs the following steps:

- Imports a simulation file into the SIF Event database,
- Analyzes the SIF Event database using stored information about the target tester and user-specified rules.
- Generates a report showing a violation report as shown in **Figure 3**.

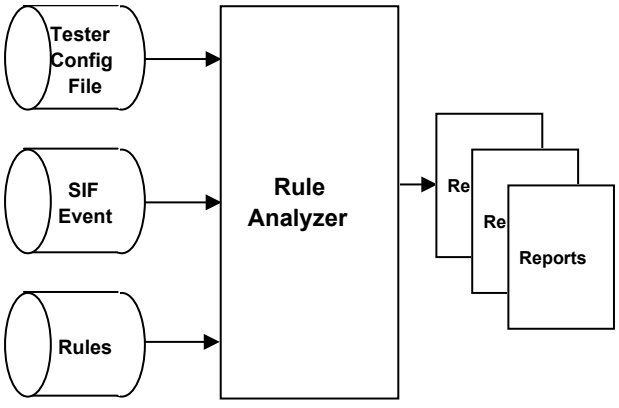


Figure 3. Rule Analysis Process

> *"The software is very reliable and has worked very well at Adaptec."*
>
> — J. C., Product Engineering Manager, Adaptec, Inc.

Analyzer Options

Figure 4 shows the interactive menu that helps you specify the signals, rules and parameters to be used by the *Analyzer*.

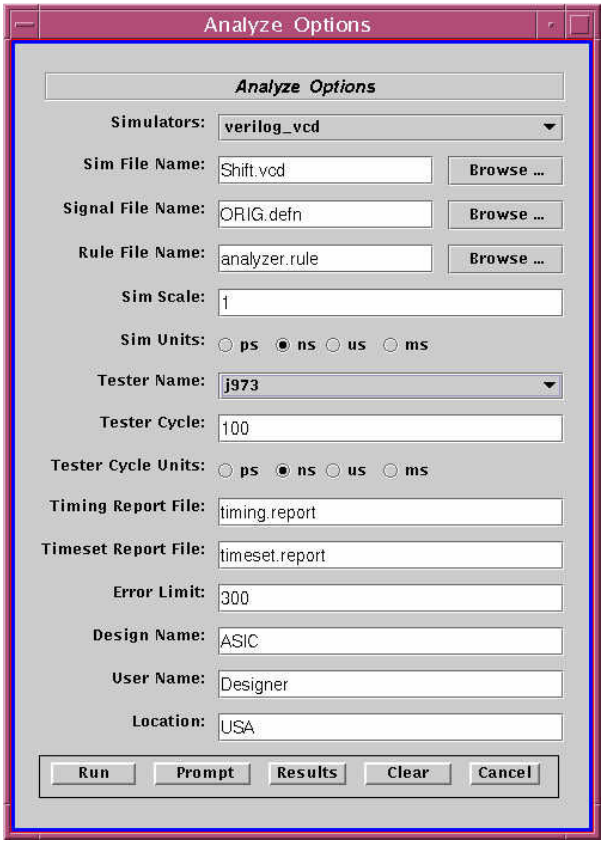


Figure 4. Analyzer Menu

Analyzer Results

Figure 5 shows a detailed rule violation report generated by the *Analyzer*. Study the report for a minute and imagine how this information could be useful for your design efforts.

```
The number of input timing edges exceeds limit *** of target tester ***.

Input signal *** requires more than four timing edges at simulation times *** ***
*** *** within a test cycle. Target tester *** is limited to *** timesets.

Timing edge violation detected for signal *** at simulation time ***, signal ***
at simulation time ***. Edge transition time *** ns relative to TO: test cycle
boundary not within constraints *** ns of target tester ***.

The signal *** is not stable within the defined stable window, period *** ns. The
violation was detected at simulation times *** and ***.

Setup time violation between signal *** and ***. Rule file set up time is ***ns.
Detected timing is ***ns. Violation occurred at simulation times *** and ***.
```

Figure 5. Analyze Reports

TRANSLATE MODULE

Simulator to Tester Translation

The purpose of the *Translate* module is to automatically translate unstructured *event-based* simulation data to structured *cycle-based* data required by ATE systems.

Until *Verifier*, the transfer of design information to production test was difficult at best.

Just give *Verifier* the information about your target tester and *Verifier* automatically translates logic simulation vectors from **more than 30 simulator formats** to optimize test programs for **more than 70 different test systems!**

The *Translator performs* tester-to-tester translations and supports multiple timesets, vector compression, and I/O switching. What's more, it ensures compatibility by performing tester rule checks.

If the *Translator* finds incompatible simulation vectors, you can eliminate these artifacts by sing Verifier's waveform editors, automatic conditioners, and timeset mapping constructs.

Examine **Figure 6.** Notice that the input is a simulator file and the output is a translated test program. Also, notice how the *Analyzer* helps you make sure that the test program is compatible with the target tester.

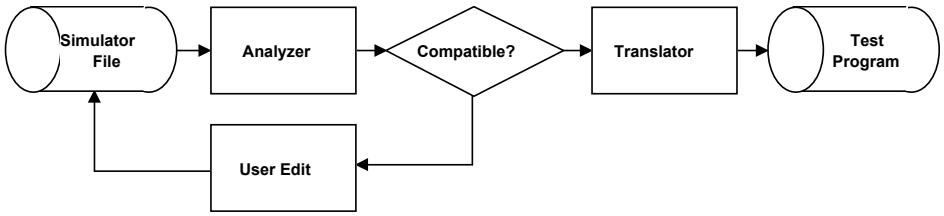


Figure 6. Rule-Based Translation Process

Translator Features

Just look at the standard features of the *Translator* to see how they can help you save time, trouble, and resources.

- Generates pin-definitions, timings, and vectors in the source language of the target test system.
- Displays simulator logic waveforms.
- Performs tester and device timing rule checks during translation.
- Identifies problem signals with their corresponding simulation times.
- Translates multiple simulation files in a single pass.
- Generates optimum timesets by reusing timesets generated from previous files.
- Supports vector compression for repeats, subroutines, and loops.
- Gives you dynamic control over simulation artifacts, i.e., tolerances can be applied to limit spurious timeset and strobe generation for a specified set of signals.

> *"…Within minutes of receiving new simulation vectors in the Verilog format, we are able to convert them into a test program for the HP83000."*
> —Shridhar Dixit, Director Test Engineering, Cisco Systems

Translator Operation

For the more inquiring Engineer, I will now describe the data-flow within translator.Notice that **Figure 7** shows an overview of how the *Source Simulator File* and *Signal Definition File* are translated to the *Target Tester Program*. *This process is automatic and transparent to the user.*

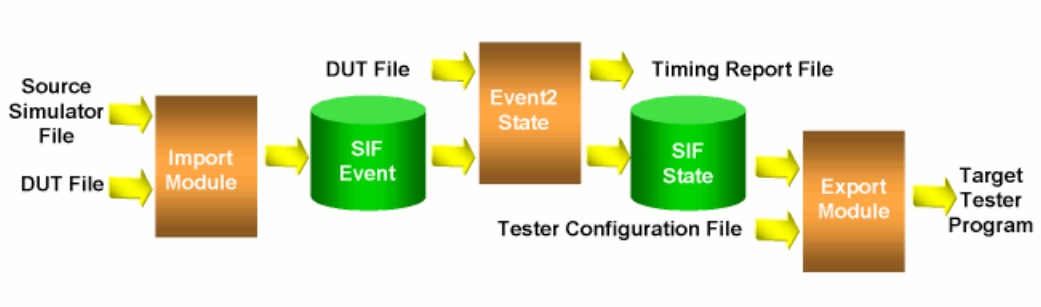


Figure 7. *Translator* Inputs and Outputs

The input files (*Source Simulator* and *Signal Definition*) specify signal definitions and optional timings. (Also **Figure 9** and **Figure 10**) for examples of these files.)

Notice that intermediate output includes SIF Event, SIF State, and various report files. (See **Figures 14-18** for examples of these files.) The *Translator* uses SIF files to produce a tester program in the source language of the target test system. **Outputs include:**

1. Pin definition statements **Figure 11**,
2. Timing and format statements **Figure 12** and,
3. Test patterns **Figure 13**.

You can use the *Translator* in both interactive or batch modes. Besides other options, the *Translator offers* three mechanisms described below:

1. **Auto Translation** — In this mode, *Translator* analyzes the simulation's event data (per pin, per cyclic vector, and across all simulation times) to best fit and reproduce vector data and timings matching the target tester's capabilities. *Translator* analyzes all event transitions and reports any problems (like excessive timing edges) that may prevent generation of accurate test programs. For output signals, you can choose *90% strobe* or *auto-strobe* options.
2. **User Control Translation** — In this mode, you can either supplement or override simulation timings. When you know a simulation file is not compatible with the target test system's physical capabilities, you can specify timings and formats to impose on the simulation timings. Your specified timing/formats will prevail. Use this mode to remove common simulation artifacts, such as minor timing deviations caused by inaccurate simulation models.
3. **Sequence Match Translation** — Mainly, the timing section of the *Signal Definition File* determines this mode. In this mode, you can specify sequence templates (a set of signals and desired signal sequences) that you want the test program to apply. The *Translator* will then compare the simulation event streams to your sequence templates. When it detects a match, it applies your template to the test program. When a match fails, it uses the Auto Translation Mode's vector and timings at the current vector.

The offline simulation environment has fewer restrictions than a physical test system. This difference sometimes requires either re-simulation or modification of simulation output files to remove incompatibilities with the target test system. Verifier lets you use *conditioners* and *user-specified constructs* to make corrections, as follows:

Conditioners

- Use the *Align conditioner* to globally correct timing edge errors caused by irregular edge placement.
- Use the *Skew conditioner* to correct errors caused by un-initialized clocked outputs.

Use the *Filter conditioner* to remove unwanted edges caused by glitches and other simulation anomalies.

User Specified Constructs

The timing section of the *Device Description File* contains constructs to dynamically alter simulation timings or data in order to remove simulation artifacts. These constructs are:

- Use *Deviation* to eliminate spurious timesets by mapping simulated edges to a value you specify.
- Use *Vector Compression* to correct pattern memory depth problems via subroutines and repeats.
- Use *Sequence Template* to correct timing problems by replacing simulated timings and formats with values you specify.

Figure 8 shows the *Translator* Menu.

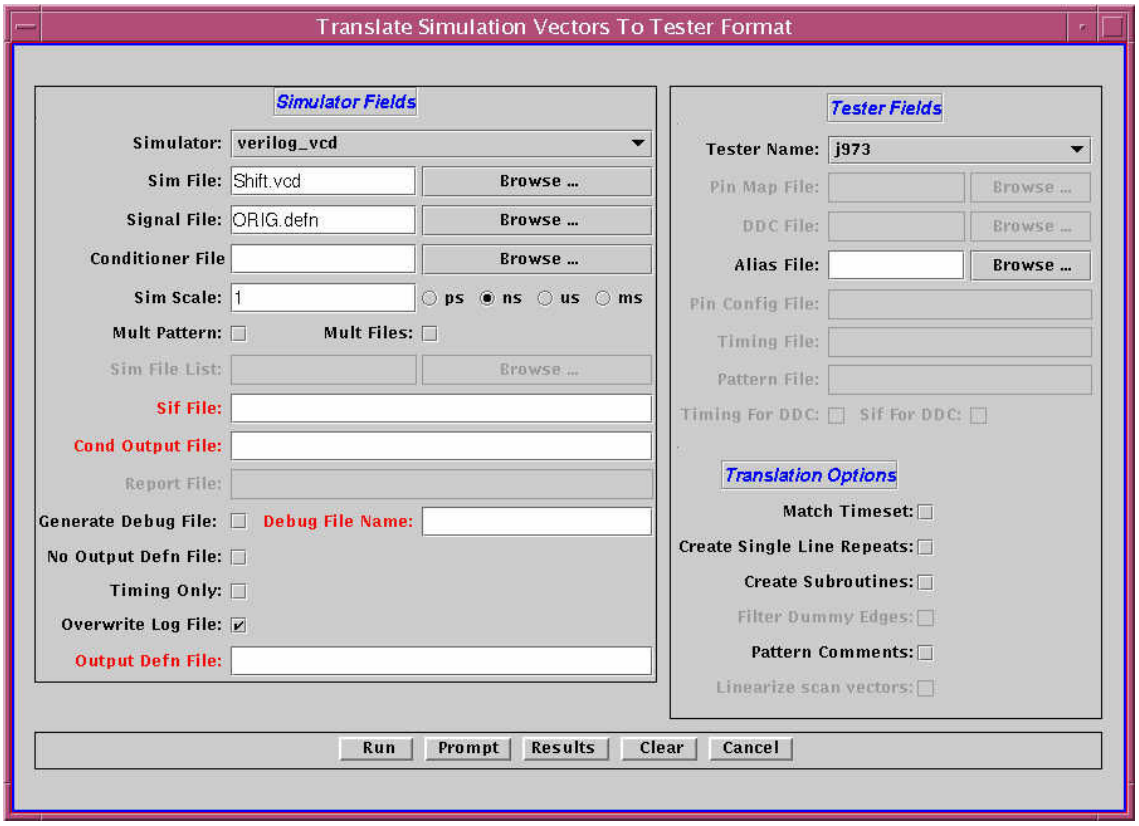


Figure 8. *Translator* Menu

EXAMPLE INPUT FILES

```
$date
    Nov  7, 1999  14:30:32
$end
$version
    VERILOG-XL 2.3
$end
$timescale
     1ns
$end

$scope module test $end
$var wire       1 !     RESPOW_B  $end
$var reg        1 #     INPCLK  $end
..
..
$var tri1       1 4!    BINIO_PT11  $end

$scope module Pin10_inst $end
$var wire       1 ?!    BINIO_PT11_en
$end

$enddefinitions $end
$dumpvars
1!
x"
0#
0?!
$end
#19
1#
#28
0#
...
...
```

Figure 9. Example Verilog

```
begin_simmap
      simstate     isifstate    osifstate
          '1'          '1'          '$'
          '0'          '0'          '$'
          'L'          '$'          'L'
          'H'          '$'          'H'
#end_simmap

begin_signals
# name type(i, o, io;, c) dut-pin(optional)
channel(optional)
            RESPOW_B      i;
            BINIO_PT11    io;
            INPCLK        i;
            BINIO_PT11_en   c;
            ...
end_signals

begin_control_expressions
      BINIO_PT11          BINIO_PT11_en;
end_control_expressions

begin_timing
input_deviation (ALL_INOUTS) 2ns min_edge;
      timeset TS1 ;
      period = 100.000ns ;
      BINIO_PT11     ( STROBE     85ns );
      INPCLK         ( R0 0ns, 50ns );
end_timing
```

Figure 10. Example Signal Definition File

Figure 9 shows a Verilog simulation output file (standard VCD format) that can be used as an input to the translation.

Figure 10 shows the Sim Map, Signal Definition and Timing sections of the signal definition file. Notice the following:

- The *Sim Map section* lets you map simulation characters to the desired SIF state.
- The *Signal Definition section* lets you define signal names and attributes.
- The *Timing section* lets you specify signal timings and formats independent of the simulation. An optional deviation construct lets you set timing tolerances to ignore minor variations in the simulation file that would cause “spurious” timeset switching.

EXAMPLE TARGET TESTER OUTPUT FILES (HP 83000)

```
hp83330,config,0.1
DFPN 10101, "1", (RESPOW_B)
DFPN 10102, "2", (BINIO_PT11)
DFPN 10103, "3", (INPCLK)
CONF I, F160, (RESPOW_B)
CONF IO, F160, (BINIO_PT11)
CONF I, F160, (INPCLK)
```

Figure 11. HP83000 Pin Definition File

This page shows examples of output files for the HP83000.

Figure 11 shows an HP83000 pin definition file, produced by Verifier's translator. Notice that this file contains pin names, I/O attributes plus dutpin and tester channel connections.

```
hp83330, timing, 0.1
PCLK 1,1,50.00,(@)
DCDF 1, "TS0", "0", (RESPOW_B)
WFDF 1, 0, D11,N,N,N,N,N,(RESPOW_B)
WAVE 1, 0, F00,,,,0,(RESPOW_B)
DCDF 1, "TS0", "0", (BINIO_PT11)
DCDF 1, "TS1", "1", (BINIO_PT11)
WFDF 1, 0, FN0,D11,N,N,N,N,(BINIO_PT11)
WFDF 1, 1, FNZ,N,EE1,N,N,N,(BINIO_PT11)
WAVE 1, 0, F00,,,,0,(BINIO_PT11)
WAVE 1, 0, F10,,,,1,(BINIO_PT11)
WAVE 1, 1, ,,FX,,2,(BINIO_PT11)
DCDF 1, "TS0", "0", (INPCLK)
WFDF 1, 0, D11,F00,N,N,N,N,(INPCLK)
WAVE 1, 0, F00,,,,0,(INPCLK)
WAVE 1, 0, F10,,,,1,(INPCLK)
ETIM  1,1,0.,0,0,0,0,0,( RESPOW_B )
ETIM  1,1,0,3,45,0,0,0,( BINIO_PT11 )
```

Figure 12. HP83000 Timing Definition File

Figure 12 shows an HP83000 timing file the *Translator* produced from Verilog simulation results. Notice that *Translator* produces per-pin timing outputs with device cycles and timing edges.

```
hp83330,vector,0.1
DMAS MTST,0,(@)
SREC ACT,(@)
DMAS PARA,304,(@)
DMAS SQPG,5,(@)
WSDM 1,2
SQLB "sample",MAIN,0,4,1
SQPG 0,STVA,0,,(@)
SQPG 1,STSA,,,(@)
SQPG 2,STEA,,,(@)
SQPG 3,GENV,300,,(@)
SQPG 4,STOP,,,(@)
VECD PARA,0,304,(RESPOW B),#9000000152
```

Figure 13. HP83000 Pattern File

Figure 13 shows an HP83000 pattern file that the *Translator* produced from Verilog simulation results. Notice that *Translator* supports comments, loops and repeat constructs.

EXAMPLE INTERMEDIATE OUTPUT FILES

Figure 14 shows *Translator's SIF Event file* as intermediate output. (.) This file contains compressed, raw simulation data, which can be used with Verifier utilities to display, edit, analyze, and compare waveforms.

```
( rawData esubmode.raw
N 1 RESPOW_B  I
N 2 BINIO_PT11 IO
N 3 INPCLK  I
vcd Scale 1.0e-12
#C 0.0 186 119 01 x2 03 ...
#C 19  2 02 13
#C 28  03
…
…
#C 2065050 13
)
```

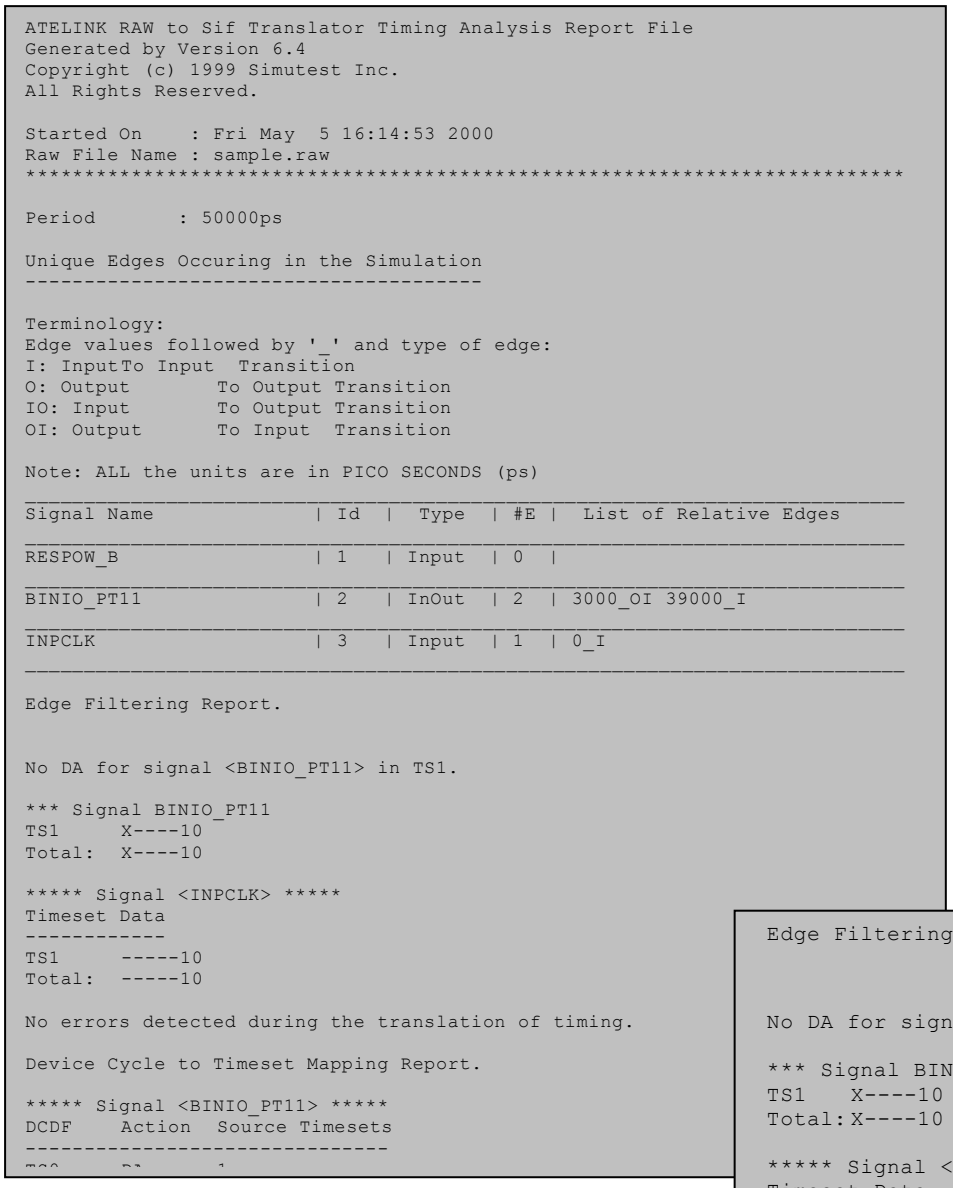
Figure 14. SIF Event File

Figure 15 shows the *Translator's SIF State file* as intermediate output. *Translator* uses this file to produce target tester-specific outputs. Notice that this file contains cyclic data in tester-like format, which supports *loops*, *jumps*, and *repeat* constructs.

```
( SifFile  sample.sif
  ( Sif
   ( TimeScale 1 )
   ( SimulatorInfo
         ( simulatorName VERILOG_VCD
)
         ( simulationValue 0 1 X Z )
         ( simulationMap
            0  0L
            1  1L
            L  0L
            H  1L
         )
   )
   ( Signal 1
         ( RESPOW_B )
         ( SignalType Input )
         ( InitialState 0 )
   )
   ( Signal 2
         ( BINIO_PT11 )
         ( SignalType InOut )
         ( InitialState X )
   )
   ( Signal 3
         ( INPCLK )
         ( SignalType Input )
         ( InitialState 0 )
   )
```

Figure 15. SIF State File

EXAMPLE INTERMEDIATE OUTPUT FILES

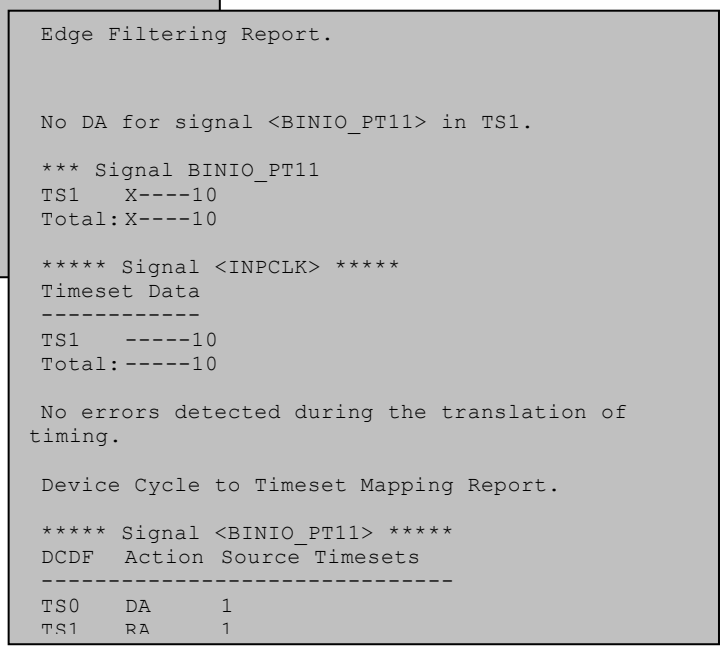


```
ATELINK RAW to Sif Translator Timing Analysis Report File
Generated by Version 6.4
Copyright (c) 1999 Simutest Inc.
All Rights Reserved.

Started On    : Fri May  5 16:14:53 2000
Raw File Name : sample.raw
********************************************************************************

Period       : 50000ps

Unique Edges Occuring in the Simulation
---------------------------------------

Terminology:
Edge values followed by '_' and type of edge:
I: InputTo Input  Transition
O: Output       To Output Transition
IO: Input       To Output Transition
OI: Output      To Input  Transition

Note: ALL the units are in PICO SECONDS (ps)
_____________________________________________________________________________
Signal Name              | Id  |  Type  | #E |  List of Relative Edges
_____________________________________________________________________________
RESPOW_B                 | 1   | Input  | 0  |
_____________________________________________________________________________
BINIO_PT11               | 2   | InOut  | 2  | 3000_OI 39000_I
_____________________________________________________________________________
INPCLK                   | 3   | Input  | 1  | 0_I
_____________________________________________________________________________

Edge Filtering Report.


No DA for signal <BINIO_PT11> in TS1.

*** Signal BINIO_PT11
TS1     X----10
Total:  X----10

***** Signal <INPCLK> *****
Timeset Data
------------
TS1     -----10
Total:  -----10

No errors detected during the translation of timing.

Device Cycle to Timeset Mapping Report.

***** Signal <BINIO_PT11> *****
DCDF    Action  Source Timesets
-------------------------------
```

Figure 16. Timing Report File

Figure 16 shows the *Translator's Timing Report File.* Notice its analysis of per-pin timing edges, cyclic changes, tester timesets, and signal format information.

Figure 17 shows the *Translation Log* for the HP83000. Notice its timings and pattern state association, plus a list of incompatibilities between simulation results and target tester. Also notice the *Translation Log* shows possible remedies for any incompatibilities.



```
Edge Filtering Report.


No DA for signal <BINIO_PT11> in TS1.

*** Signal BINIO_PT11
TS1    X----10
Total: X----10

***** Signal <INPCLK> *****
Timeset Data
------------
TS1    -----10
Total: -----10

No errors detected during the translation of
timing.

Device Cycle to Timeset Mapping Report.

***** Signal <BINIO_PT11> *****
DCDF   Action Source Timesets
-------------------------------
TS0    DA     1
TS1    RA     1
```

Figure 17. Translation Log For HP83000

TEST PROGRAM MIGRATION

(Translation from Source Tester to Target Tester)

Verifier's modular architecture allows you to also convert existing test programs (pin-definition, timings and patterns) from a source tester to a target test system.

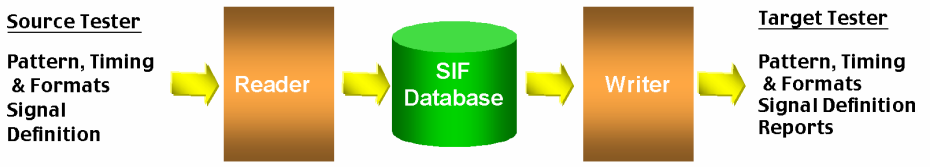


Figure 18. Test Program Migration – Inputs and Outputs

Figure 18 shows how Verifier's *Translator* converts from a Source Tester to a Target Tester. Verifier's Playback module first converts input source files into SIF database.

Verifier's translator module then reads this SIF file and converts it into a target test program.

During the conversion process, incompatibilities between the source test program and target test system is detected, possible remedies are suggested. Verifier's Conditioner module may also be used to correct some of the incompatibilities.

SIMULATOR MIGRATION

(Translation from Source Simulator to Target Simulator)

Figure 19 shows how Verifier's *Translator* converts from a Source Simulator to a Target Simulator.

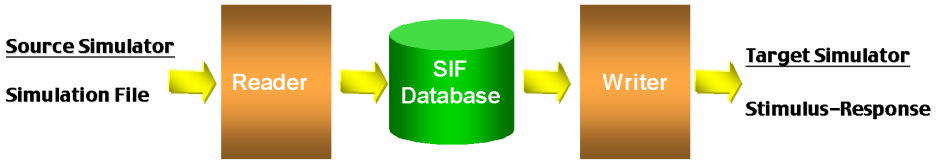


Figure 19. Simulation Program Migration – Inputs and Outputs

In today's ASIC environment, ASIC designers can use several platforms and design tools. Thus, it is often necessary to move design data from one design environment to another for verification or fabrication.

For example, simulation test patterns generated by a system design house may use a third party CAE workstation. The ASIC foundry will need to convert the test vector patterns into its "golden" simulation environment for final simulation and verification.

Verifier's *Translator* accepts simulation vectors from different simulators and imports them into Simutest's SIF Event database. The *Translator*then converts these patterns into the stimulus-response format of the target simulator. Optionally, you can export stimulus only, response only, or both vectors.

This function is only supported in Verifier's batch mode.

> *"The modular architecture of the software also gives us the flexibility to select a different simulator or tester as our needs change."*
> — J. C., Product Eng. Manager, Adaptec, Inc.

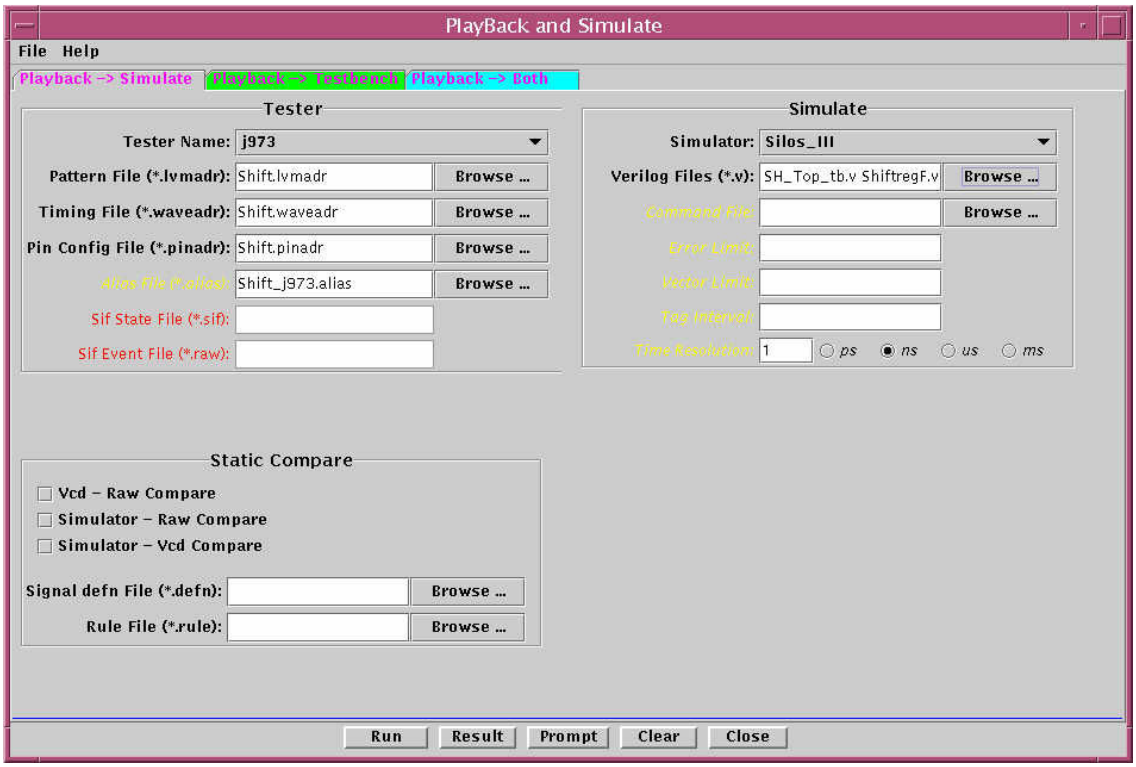


Figure 20. ***Playback*** **Menu**

Figure 20 shows the Menu for setting up *Playback* options.

Figure 21 shows how *Playback* mode converts the final test programs into a testbench. Verifier dynamically compares expected versus simulated results to ensure integrity of original design intent. *Playback* is critical to maintain accuracy of the final test program.

Playback logs all discrepancies to a log file. Also, Verifier displays in a single window, pin-for-pin waveforms for the original simulation file and the test program. (See **Figure 13**) Thus, you can rapidly pinpoint the source of the discrepancy.

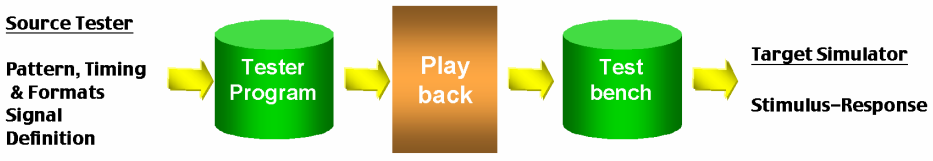


Figure 21. ***Playback*** **Mode**

SIMULATE MODE

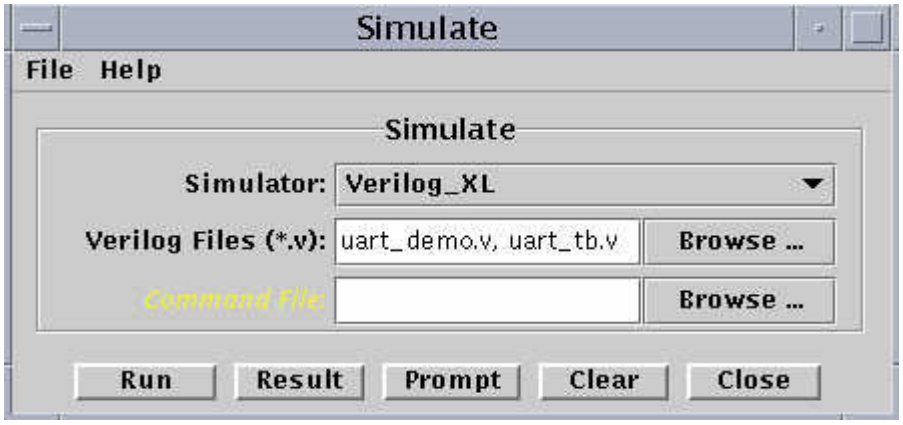
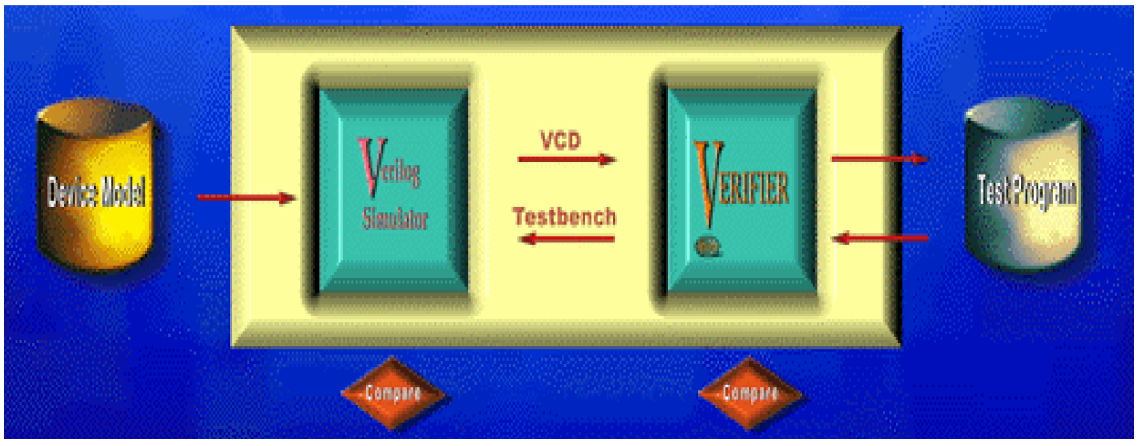


Figure 22 Simulate Menu

Verifier lets you use a Verilog simulator to verify design and test programs concurrently. This powerful tool ensures integrity of the final test program. **Figure 22** shows the Simulate Menu options.

You can invoke the Verilog simulator from the Verifier. The Verilog simulator may be driven by a testbench generated from the test system test program or from your design test-bench.



Device Model to Test Program

Compare Reports

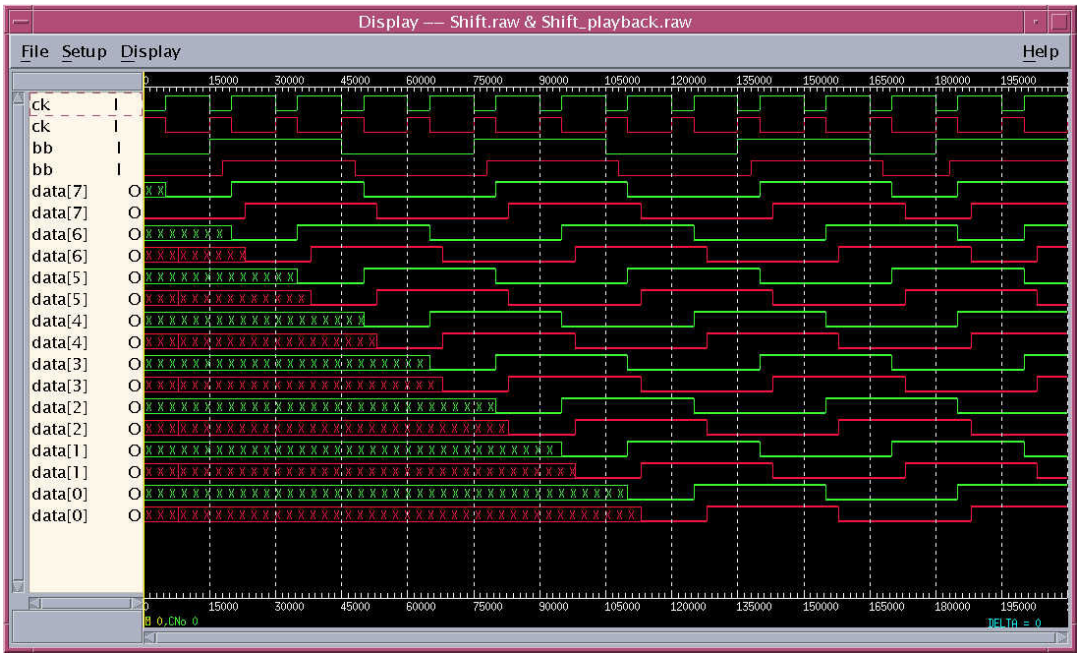


Figure 23. Graphical Report in Waveform Viewer

Textual and graphical reports help you analyze and compare results. Notice in **Figure 23**, the waveform graph report shows both tester and simulator waveforms.

When you choose the text report, Verifier sends all errors to a report file. Notice, in **Figure 24**, that the text report contains detailed error information, including signal name, timestamp, logic state and/or type of transition.

```
*********************************************
*                Report Summary             *
*********************************************
Simulation time scale is: 100 ns
Signal    Total     Total      First    Last
Name      Trans     Errors     Error    Error
---------------------------------------------
y[5]        88        3          400      2900
COUT        5         1          433       433


*********************************************
*              Functional Differences       *
*********************************************
Event Time   Signal   file1   file2

      0        q15      0       1

    400        q0       0       x
               y[5]     0       x
```

```
*********************************************
*          Timing Difference                *
*********************************************
Signal Name     Ram15
      Time in   file1     file2
                433       450
    •           495
                495       500


*********************************************
*          At Strobe Differences            *
*********************************************
Event Time   Signal   file1   file2

      0      q15       0       1

    400      q0        0       x
             y[5]      0       x
```

Figure 24. Comparison Text Reports

SIF DATABASE

Verifier's modular architecture is built around Simutest's Standard Intermediate Format (SIF) database.

The SIF database efficiently stores both simulation and tester data in a simulator-tester independent format. See **Figure 25.** To accurately describe the simulator, the SIF database stores standard CAE & ATE tester and tabular data in three formats:

- **Event format**: unmodified simulation data (compacted).
- **State or cycle-based format**: (including SCAN): tester-oriented cyclic format.
- **Tabular format**: generic "tabular IO" (ASCII) used by proprietary and commercial CAE simulators.

Verifier converts data between SIF database formats in either the batch or interactive mode.

←Converting Between SIF Formats→		
Event Format	**Tabular Format**	**State Format**
Signal Attributes • name, type • time stamp	**Signal Attributes** • name, type • time stamp	**Signal Attributes** • name, type, pin mux • channel, dutpin
		Timings & Formats • per pin timings • tester oriented: nrz, dnrz, R0,R1 • timesets • **pin groups**
Vector Data • logic state • time stamp	**Vector Data** • standard logic states: 0,1,L,H,Z,X	**Vector Data** • linear • subroutine, repeat • loops, jumps • scan • comments • labels • bin or hex radix
	• **Column-oriented test vector data**	

Figure 25. SIF Database Constructs

WAVEFORM DISPLAY

The waveform viewer displays SIF Event, State and Tabular data from either simulators or testers. The features of the waveform display module in **Figure 26**.

Features

- Displays event or cycle based data
- Binary or hex data representation
- Signal groups
- Zoom/pan
- Edge relationship searches
- Measures delta between cursor and marker
- Color waveforms
- Clear and refresh

Figure 26. Waveform Display Features

Verifier is a single environment for design and test tools.

Therefore, both Design Engineers and Test Engineers can access and manipulate the same file.

Notice that Verifier's unique Waveform Viewer (**Figure 17**) makes it easy to spot and correct problems in the simulation file. The Waveform Viewer displays, pin-for-pin, the simulation vector and the actual test program vector.

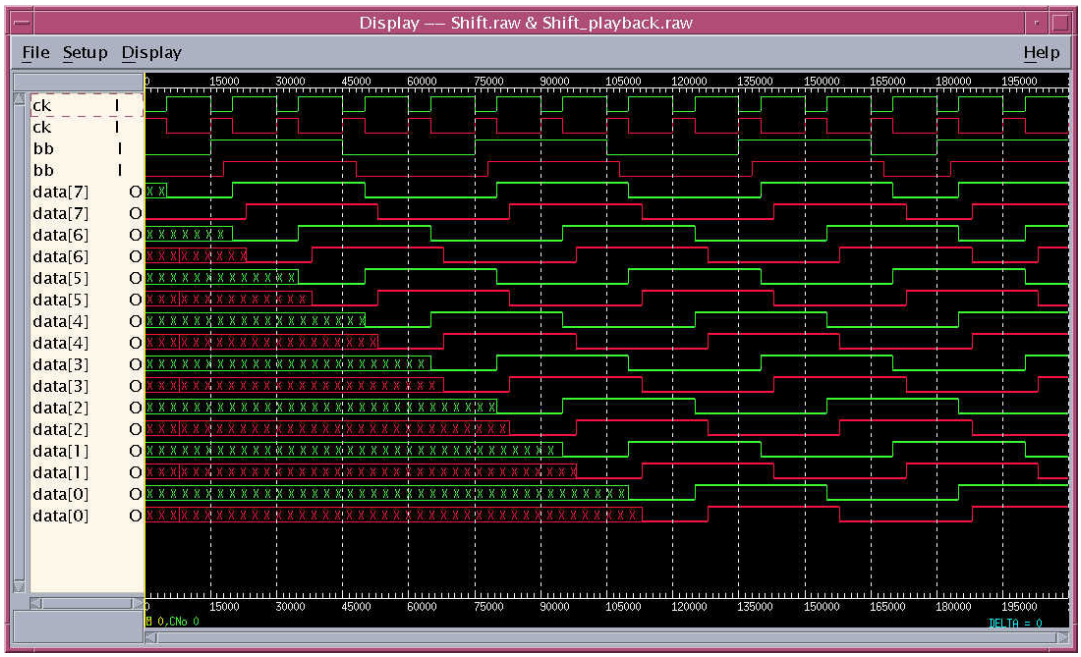


Figure 27. Verifier Waveform Viewer

Verifier's graphical interface can show simulation vectors as waveforms.

SUPPORTED TEST SYSTEMS

Vendor	Test System
Advantest	T3319, T3320, T3340, T3341, T3381
Ando	8000 & 9000 Series
Credence	4060, 6060, 6120, 8256, Vista, DUO, Qurtet, SC212, SC312
GenRad	GR16, GR18
Hewlett-Packard	949X (Mixed Signal),HP82000/83000,A93000, HP16500 Logic Analyzer
IMS	Logic Master, XL-100
LTX	DX90, Syncro, MASTER, Delta Series, Fusion
Megatest	Megaone, Polaris
NexTest	Maverick
Schlumberger	600, 610, SII, SVII, SVIII, S10, S20, S21, S15, S50, ITS9000
Teradyne	J941, J953, J967, J971, J973, A520, A540, A580, J750, Catalyst
TI	V-Series

SUPPORTED CAE SIMULATORS

Vendor	Simulator	Vendor	Simulator
AT&T	Motis	LMSI	Tabular
IEEE 1549.1	STIL	Mentor	Quicksim, LSIM, MASM
Cadence	Verilog-XL, NC Validsim, Rapidsim		FASTSCAN, modelsim
Chip Express	CTV	OKI Semi	Tpl
Avanti	Timemill, Powermill	Racal-Redac	Cadat
Fujitsu	FJTDL	Synopsys	VCS, Tetramax, Testgen
GenRad	Hilo	Simucad Silos,	SILOS III
Tabular I/O	Generic	Teradyne	Lasar
Hitachi	Cliff	TI	TDL
Ikos	IkosSim	Toshiba	TSTL,TSTL2
Intergraph	DLS, Advansym	Viewlogic	Viewsim, VCS
Jedec	JEDEC	Philips/VTI	Trace, VIF
LSI Logic	Lsim	Zycad	Zilos, Mach1000

SUPPORTED PLATFORMS AND OS

Vendor	Operating System
Sun	Solaris 2.6.X, 2.7

SUPPORTED TARGET TESTER LANGUAGE STATEMENTS

Verifier supports the most commonly used tester-specific test language statements used for describing pin definitions, timing, and test vectors. Following is a list of test language statements for some test systems supported by the Verifier product.

Advantest

Supported T33XX Language Statements

Pattern Statements

- Logic value radix - binary.
- IDXI.
- STPS.
- JSR (Flattened).

Pin Definition and Timing Statements

- Pin definitions - i, o, io.
- Pinlist - pin group definitions.
- Format - NRZA, NRZB, NRZC, RZ0, /RZ0, RZZ, /RZZ, FIXH, FIXL, STRB.
- Generators - ACLKn, BCLKn, CCLKn, DREL, DRET, period.
- Timesets - all timings are in constant value or expression form.

Credence

Supported VISTA, DUO, Qurtet Language Statements

SWAV Pattern and Test Program Statements

- Signal - definitions — input, output, bidir, bidir1, power, noconnect.
- Timeset - apply, sense, NRZ, DNRZ, RZ, RTO, Inhibit.
- Pattern - single or double character definitions, sequence numbers, marker statements, repeats, loops.

Hewlett-Packard

Supported HP83/93000 Language Statements

Pattern Statements

- CLKR Clock resolution.
- CMNT Comment statement.
- CNTR Control command.
- GETV Get vector command.
- SLCF Sequencer label configuration command.
- SQPG Sequencer programming command.
- Within this command, Verifier supports the following sequencer instructions:
- GENV - Generate vector instruction.
- STVA - Set parallel vector address instruction.
- STOP - Stop test instruction.
- CTIM - Change timing.
- RPTV - Repeat vector instruction.

Pin Configuration Statements

- CONF Configuration command for the pin type.
- DFPN Define the pin name.

Timing & Format Statements

- DCDF Device cycle definition command.
- ETIM Edge timing command.
- PCLK Pin clock command.
- WAVE Wave command.
- WFDF Waveform definition command.

LTX

Supported Trillium Language Statements

Trillium Pattern Instructions

- BitPattern - binary.
- RPT instruction.
- JSR instruction.
- Vector Label instruction.
- Trillium PASCAL Instructions

- PINDEF - pin definitions — i, o, p, radix bin.
- PINGRP - pin group definitions.
- Marker statements.
- Format definitions.

Supported Envision Language Statements

- Pattern, timing and pin definition in the Envision syntax.

Schlumberger

Supported SENTRY S15 Language Statements

SCM/MDC Pattern Instructions

- BitPattern - binary or hex (Patterns and D/M registers).
- Inc instruction.
- Halt instruction.
- Rpt, ldc and ldf instructions.
- Gonz instruction.
- Ret, retz and retcl instructions.
- Call and callsub instructions (Flattened).
- Subr and subroutine_F memory instructions (Flattened).
- Param_Select instruction (PARAM, PS).
- Vector Label and mark instruction.
- INCLUDE vector files.

PASCAL-15 Instructions

- PINDEF - pin definitions — i, o, p, radix, bin and hex.
- PINGRP - pin group definitions.
- FORMAT_DEF, FORMAT- NRZ, RT0, RTZ, SBC.
- SET_PERIOD_VALUE - t.
- SET_TEG_VALUEs.
- ASSIGN_PIN_TEG.

Supported ITS9000 Language Statements

Pattern Statements

- Logic value radix - binary or hex.
- rpt.
- halt
- call.
- • Vector label.

Pin Definition and Timing Statements

- Pin- form pin definitions - i, o, io, F1 bit.
- Group-form - pin group definitions.
- Essm sequence - NOOP, D0, D1, DF, DF_, DZ, TF, X.
- VECTOR-DEFs- All timings in constant value or expression form.

Teradyne

Supported J953 Language Statements

Pattern Statements

- Logic value radix - binary or hex.
- Rpt.
- Stop.
- Call.
- Vector label.
- Limited vector control.

Timing Statements

- Waveset ZLOAD.
- Format - nr, rl, rh, rc and compare (edge and window).
- Waveset - all timings are in constant value or expression form.

Supported J971,J973 Language Statements

Pattern Statements

- Logic value radix - binary or hex.
- Rpt.
- Stop.
- Call.
- Vector label.
- MACH-4 mode.

Pin Definition and Timing Statements

- Pin- for pin definitions - i, o, io, mcg.
- Group-form - pin group definitions.
- Format - nr, rl, rh, rc, stay, mcg and compare (edge and window).
- Edgesets - all timings are in constant value or expression form.

Partial Customer List	
Adaptec	AMD
Atmel	Chartered Semiconductor
Chip Express	Cisco Systems
Credence Systems Corporation	Cypress Semiconductor
Dallas Semiconductor	Fujitsu Microelectronics, Inc.
Hewlett-Packard	Hitachi Micro Systems
Jet Propulsion Laboratory	LSI Logic
LTX	Motorola
Oki Semiconductor	Samsung Semiconductor
Schlumberger	Siemens Microelectronics
Silicon Graphics	Silicon Magic
Sun Microsystems	Texas Instruments
VLSI Technology	Xilinx
Others . . .	

Customer Quotes

"We were one of the early customers of Simutest when we bought the test automation software products in 1988. We have been using the tools regularly for over seven years now to test our internally designed chips. Simutest worked very closely with us to understand our test environment and then provided us with critical enhancements to the software to fit our needs. The software is very reliable and has worked very well at Adaptec. The modular architecture of the software also gives us the flexibility to select a different simulator or tester as our needs change." — Joe Chan, Product Engineering Manager, Adaptec, Inc.

"We switched to Simutest's test automation software for the new generation of chips. These software tools have allowed us to completely automate design-to-test transfers. Within minutes of receiving new simulation vectors in the Verilog format, we are able to convert them into a test program for the HP83000.

Simutest's technical support is outstanding. Our questions get answered promptly, usually within a few hours; and that is important during crunch times. We are very happy with Simutest and will continue to rely on their products for our future needs." —Shridhar Dixit, Director Test Engineering, Cisco Systems

COMPANY BACKGROUND

Since its founding in1987, Simutest has become a leading supplier of simulation analysis and test development tools. The company sustains growth by providing quality products at reasonable cost and by backing its products with responsive and competent technical support.

Simutest provides the most comprehensive support for simulators and test systems in the industry, supporting more than 30 simulators and 70 test systems.

Simutest, Inc.
4677 Old Ironsides Drive #315
Santa Clara, CA 95054
Tel: (408) 988-4032
Fax: (408) 988-4034

Simutest

Web Site: http://www.simutest.com
E-mail: info@simutest.com