CHAPTER 6

TIMING VERIFIER

Reference Manual

6.1 INTRODUCTION

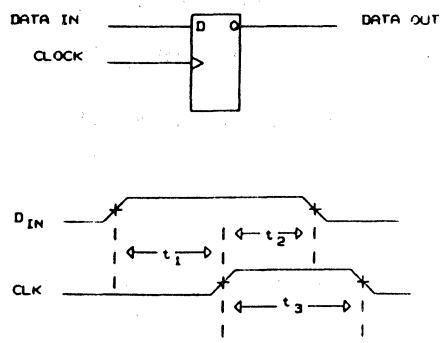
The Timing Verifier represents a new approach to verification of timing constraints of large digital systems. The Timing Verifier uses an algorithm which is computationally efficient and complete. Furthermore, the Timing Verifier does not require test inputs (such as a logic simulator) and works directly from the output of the Compiler. Thus, timing verification is done using only the designer's original set of drawings.

The Timing Verifier allows the verification of entire designs or of designs section by section. Verification of portions of a design means that small pieces of a design may be verified to save computation time. Similarly, a design need not be complete to be verified - verification can proceed on the finished sections. Designers can check their own pieces of a system on a daily basis, getting continuous feedback on its correctness as work proceeds. Verification of an entire system can be done when the pieces are known to be correct.

WHAT IS TIMING VERIFICATION?

Digital systems are composed of components and their interconnections, or wires which convey signals from one component to another. In general, when a signal on the input of a component changes, some time later the signal on the output changes. The wire connected to this output then conveys the signal to the input of other components, again after some delay. Because of variations in construction, the delay time of components and wires varies.

At certain places in a system - data inputs of registers, and external interfaces for example - a signal must assume its value at a certain time. Thus, if a path to such a place is too long or too short, the system may yield an incorrect result. For example, consider a circuit consisting of a D register:



For real devices, t1 must be longer than some critical time - the setup time of the register ts - or the device may malfunction. If the data delay is long, t1 may shrink below ts, violating the setup time spec of the register. Similarly, t2 must be longer than the hold time of the register, th. If the data delay is short, the hold time specification of the register may be violated. Finally, the width of the clock pulse, t3 must exceed some minimum time, or correct operation of the register is not guaranteed.

As a second example, consider a memory interface with a data bus and a data out valid signal:

```
D<0> ----------<J21>        D<4> ----------<J25>
D<1> ----------<J22>        D<5> ----------<J26>
D<2> ----------<J23>        D<6> ----------<J27>
D<3> ----------<J24>        D<7> ----------<J28>

DOVAL ------------<J30>
```

There will typically be some specification that data must be ready some period of time before DOVAL becomes true. If this setup time is not met, systems connected to the memory may malfunction.

The Timing Verifier checks that there are no timing violations of these two types in the design. That is, at all points in a design:

- Component constraints (setup, hold, pulse width, etc.) are observed.
- All interface specifications provided by the designer are met.

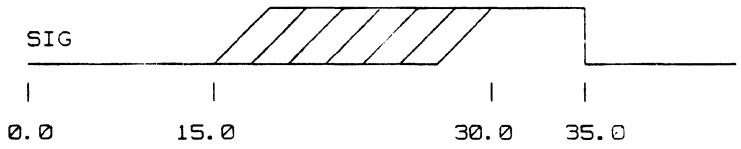
Timing constraint verification is based on minimum and maximum propagation delays of circuit components, their set-up times, hold times and pulse width constraints, wire delays and interface specifications.

6.2 TIMING VERIFIER OPERATION

The Timing Verifier operates in two phases. First, it computes the value history of every signal in the system over one clock period. Then, it checks that the signals meet the timing constraints of the components and interfaces.

6.3 SIGNALS IN THE TIMING VERIFIER

The Timing Verifier represents the behavior of a signal over time textually. For example, a signal SIG is shown as a waveform and in Timing Verifier text format:



SIG: 0:0.0, R:15.0, 1:30.0, F:35.0, 0:35.0

The evolution of a signal over a clock is called its value history.

A basic assumption of the Timing Verifier is that the circuit to be verified has periodic behavior. That is, given a circuit and a set of input stimulus, there is some state of the circuit S and some time T, such that starting the circuit in state S, applying the inputs and simulating for time T, the circuit returns to state S. (By state of a circuit, we mean the value history of each signal in the design.) Synchronous sequential circuits, and strictly combinational circuits both have this property.

In general, the design is simulated using an 8 value logic system:

1. 0 -- signal is 0 or false

2. 1 -- signal is 1 or true

3. S -- signal is stable, that is either 0 or 1

4. R -- signal is rising - going from 0 to 1

5. F -- signal is falling - going from 1 to 0

6. C -- signal is undergoing a transition of unknown direction

7. U -- nothing is known about the signal

8. Z -- the signal is high impedance

The truth table for an AND gate in this logic system is:

AND	0	1	S	R	F	C	U	Z
0	0	0	0	0	0	0	0	0
1	0	1	S	R	F	C	U	U
S	0	S	S	R	F	C	U	U
R	0	R	R	R	C	C	U	U
F	0	F	F	C	F	C	U	U
C	0	C	C	C	C	C	U	U
U	0	U	U	U	U	U	U	U
Z	0	U	U	U	U	U	U	U

The intent of this approach is to preserve the logic behavior of the circuit when actual values are known, and in other cases, to represent a signal as stable (0,1 or S), or undergoing a transition (R, F or C). In most cases, this information is sufficient for timing verification. For example, to verify the setup and hold time of a register, you only need to know when the D input is undergoing a transition and when it is stable with respect to the clock input. The actual value (0 or 1) on the D input is not important. Similarly, you do not need to know the output value (0 or 1) of the register. The output of a register changes only during a short interval after it is clocked and otherwise it is stable. Using S means that the contents of registers and memories do not have to be specified, greatly reducing the amount of time the designer has to spend preparing inputs for the verifier. Also, using S exponentialy reduces the number of states that must be simulated to verify the timing behavior of the circuit. For example, a k-bit counter that contains the value SSSSS....SS

(k times) has only one state, not 2**k states. Rarely does a circuit's timing depend on the actual value in the counter, but merely how long after the counter is clocked it takes for the outputs to stop undergoing transitions.

There are some cases where modelling the signals in a circuit as stable or undergoing a transition is not adequate, the actual (0, 1) behavior is necessary. The Timing Verifier has a mechanism called case analysis for handling these situations.

In addition to the eight values described above, a signal may have one of three strengths, HARD, SOFT and UNDRIVEN. In effect the Timing Verifier does a kind of twenty-four state simulation. Signal strengths are discussed in detail in section "Signal Drive Strengths".

6.4 SCALD SIGNALS AND THE TIMING VERIFIER

The Timing Verifier needs certain kinds of information about signals in the system being checked. For example, interface specifications for those output signals that are to be checked must be provided. In general there are three kinds of information that the Timing Verifier extracts from signals - timing behavior, delays and special evaluation rules.

Timing Behavior (Signal Assertions)

The Timing Verifier initially sets all undriven signals of the circuit to "S" (stable) and all others to "U" (undefined). Often systems will have many undriven inputs set to "S" in this manner and consequently will not exhibit meaningful timing behavior. Some examples of this are:

1. Primary inputs to the system. For example the design may be a controller which "talks" on some standard bus interface. If all the interface (input) signals are always stable, the controller will not operate.

2. Partial designs. One of the most important aspects of the Timing Verifier is the ability to verify the timing of partial designs. (A partial design may be either one that is incomplete, or a piece of an entire system that was extracted for separate timing verification.) In a partial design, signals that have not been generated yet will be undriven.

3. Clock Signals. In large systems it is often convenient to defer the design of the logic for complex multiphase clocks to near the end of the design cycle. These signals will therefore be undriven, even though their

timing behavior is known. Furthermore, a synchronous system will not do anything unless these clock lines are driven.

For these signals assertions, which are simply part of the signal name, can be added. Assertions define the value history of signals when the history is not determined by a driving device. Assertions are discussed in detail below.

DELAYS

Timing verification requires the modelling of both component and interconnect delays. Component delays are specified inside Timing Verifier library components. Interconnect delays on the other hand are specified as delays associated with wires.

A delay is associated with a wire in one of several ways:

1. The designer may place a delay property on a signal. If this is done that particular instance of the signal has the specified delay.

2. The Timing Verifier will read a list of delays typically computed by some physical design subsystem. A list element associates a delay with an input pin. Thus the delay on each stub of a signal that drives multiple loads may be specified.

3. The Timing Verifier can use its delay estimator to calculate an estimated delay based on the number of loads and the size of the loads.

4. If none of these delays is specified, the Timing Verifier will use a default delay value that is specified when the Timing Verifier is run (which may be zero).

TUNED SIGNALS AND GATED CLOCKS (EVALUATION DIRECTIVES)

High-speed digital system often use clocks which have been tuned in order to compensate for delays in the system. A means for describing signals that will be adjusted to have some particular timing behavior independent of circuit delays is necessary for complete timing verification. Evaluation Directives provide these descriptions.

A related complication occurs in systems that use gated clocks. The system functions correctly only if the gating signal properly "envelopes" the clock for all variations in

circuit delays. Evaluation directives are used to direct the Timing Verifier checks for correct timing behavior of this type signal as well.

6.5 SIGNAL ASSERTIONS

In order for the Verifier to produce meaningful results, the designer must specify the value history of all input signals to the design and all interface signals that are to be checked.

Value histories are specified using assertions which are simply part of a SCALD signal name. The general form is:

<assertion> ::= ! <clock period> <assertion type> <time specifier> <explicit skew>

<assertion type> ::= C | P | S | D

The assertions recognized by the Verifier are:

1. C -- this indicates that the signal is a clock signal. Together with the <time specifier> and <explicit skew> this determines the 0, 1 behavior of the signal. If no <explicit skew> is given, this assertion will use the skew specified by the CLOCK_SKEW directive.

2. P -- this indicates that the signal is a precision clock signal. The P assertion is identical to the C assertion except that when no <explicit skew> is given, it uses the skew specified by the PREC_CLOCK_SKEW directive.

3. S -- together with the <time specifier> and <explicit skew> this determines the stable, changing behavior of the signal. This is used to specify an initial value history for a signal. During the course of verification, should the computed value be different than the specified value, the computed value will replace it. If no <explicit skew> is given, then the <time specifier> is assumed to be exact, and no skew is added to the signal.

4. D -- together with the <time specifier> and <explicit skew> this determines the stable, changing behavior of the signal. This assertion is the same as the S assertion except that the value history specified is never changed during the course of verification. The use of the D assertion on signals in feed back paths which are broken by latches can significantly speed up the execution of the Timing Verifier.

<clock period> is a time (in nanoseconds). This overrides the CLOCK_PERIOD in the directives file for a particular signal. The clock period has to be a sub-multiple of the system clock period. For example, with the CLOCK_PERIOD and CLOCK_INTERVALS set to 100, the following assertions can be given:

Signal Assertion	Equivalent To
SIG !50 P0-25	SIG !P 0-25, 50-75
SIG !25 C5-10	SIG !C 5-10, 30-35, 55-60, 80-85

A <time specifier> is used to describe time intervals. The <explicit skew> allows uncertainty or skew to be specified about the <time specifier>. The detailed syntax is:

<time specifier> ::= <time interval> |
<time interval>, <time specifier>

<time interval> ::= <time in clk units> | <time period> |
<pulse>

<time period> ::= <time in clk units> - <time in clk units>

<pulse> ::= <time in clk units>+<time in nsec>

<explicit skew> ::= | (<negative skew> , <positive skew>)

<negative skew> ::= -<time in nsec> | <time in nsec>

<positive skew> ::= +<time in nsec> | <time in nsec>

<time in clk units> ::= <integer> | <fixed point number>

<time in nsec> ::= <integer> | <fixed point number>

All three types of <time interval>s specify signal behavior in terms of evenly spaced sub-intervals of a global clock. A clock unit is equal to one of these sub-intervals. In the examples below the clock period is assumed to be 100 nsec and is divided into 8 even sub-periods of 12.5 nsec each. The CLOCK_SKEW directive is -2 nsec to +2 nsec, and the PREC_CLOCK_SKEW directive is set to 0.

Each of the types of <time specifier>s is shown below:

<time in clock units>

This form specifies a pulse whose width is one sub_period long. The signal is asserted the number of indicated sub-periods from the begining of the cycle.

CLK!P 4	0:0.0, 1:50.0, 0:62.5
CLK!P 4 (-2,5). . .	0:0.0, R:48.0, 1:55.0, F:60.5, 0:67.5
CLK!C 2,5*	1:0.0, F:23.0, 0:27.0, R:35.5, 1:39.5, F: 60.5, 0:64.5, R: 73.0, 1:77.0
CLK!P2.2,5.7 . . .	0:0.0, 1:27.5, 0:40.0, 1:71.3, 0:83.8

Note that low-asserted clocks, such as the third example above, take the value "0" when the signal is asserted.

<time period>

A <time period> is a pulse whose leading and trailing edge is specified.

CLK!P 0-2.0 . . .	1:0.0, 0:25.0
CLK!P 1-4,6.0-7 . .	0:0.0, 1:12.5, 0:50.0, 1:75.0, 0:87.5
SIG!S1-4,6-7.3 . .	C:0.0, S:12.5, C:50.0, S:75.0, C:91.3
SIG !S2-4 (-1,5) . .	C:0.0, S:30.0, C:49.0

<pulse>

This form is used to specify a signal whose start time is specified relative to the clock sub-periods, but whose width is specified in absolute units.

SIG!S2+11.3 . . .	C:0.0, S:25.0, C:36.3
-CLK!P3+9.0,4+10.0 .	1:0.0, 0:37.5, 1:46.5, 0:50.0, 1:60.0

The time before the plus symbol gives the time of the leading edge of the pulse in clock units, and the time after the plus symbol gives the width of the pulse in absolute time units (nanoseconds). This allows for pulses to be specified where the width doesn't scale with the cycle time of the circuit.

ADVANCED USE OF ASSERTIONS

During simulation, if a driven signal does not meet its assertion specification, an error is reported in the Timing Verifier output file. Assertions have obvious uses as interface specifications of signals, and as pseudo-drivers in partially complete designs. They may also be used to create abstract models.

An abstract model of a part P consists of a body drawing and an abstract timing model. The abstract timing model is constructed soley of buffers -- all input signals are received by buffers and the output signals driven by buffers. The output of each receiver buffer has a local signal with a timing assertion on it matching the input timing spec of the corresponding pin of P. The input of each drive buffer has a local signal with a timing assertion on it matching the output timing spec of corresponding output pin of P. This approach can be expanded to have small amounts of logic in the abstract model to achieve more complex logic or timing behavior as required.

One capability of the Timing Verifier enhances the power of timing assertions. Assertions may be specified in the CASE analysis file rather than on the print. This facilitates experimenting with assertions. See the section on Timing Verifier Case Analysis for details.

6.6 DELAY PROPERTIES

Delay properties are used to model the delays of a circuit's interconnections. Delay properties are simply SCALD signal properties:

```
<delay property> ::= \ <property name > <value>
<value>          ::= = ' <time interval specifier> '
<time interval specifier>
                 ::= <delay range> |
                     <rising range> , <falling range>
<rising range>   ::= <delay range>
<falling range>  ::= <delay range>
<delay range>    ::= <delay> |
                     <min delay> - <max delay>
<delay>          ::= <time>
<min delay>      ::= <time>
<max delay>      ::= <time>
<time>           ::= <integer> | <fixed point number>
```

Verifier delay properties indicate that the signal is to be delayed (by the time indicated by the <time interval specifier>) with respect to the signal source. The most general form of a delay gives a minimum and maximum rising

delay and a minimum and maximum falling delay. If only one range is given, then the rising and falling delays are assumed to be the same.

The general delay properties recognized by the Verifier are:

1. WIRE_DELAY -- this type of delay is a simple wire delay property. It value may be over-ridden by the physical design subsystem, and also may be set to zero by certain evaluation directives.

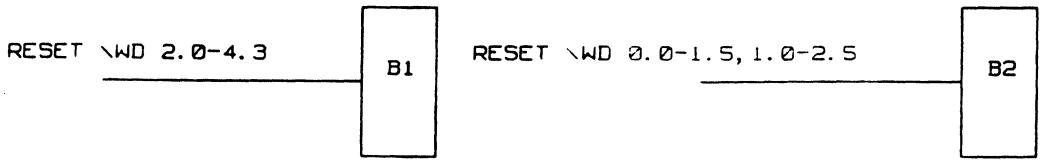
2. CHIP_DELAY -- this type of delay is used primarily with Verifier library models. It is just like the WIRE_DELAY, except that it is not changeable by the physical design subsystem and a different set of evaluation directives set it to zero.

3. CLOCK_DELAY -- this delay is not affected by any evaluation directives and cannot be overridden by the physical design subsystem. Its primary use is to describe a tuned clock which is adjusted to have some delay with respect to another logical version of the clock.

All delay properties are the same except in the way evaluation directives and the physical design subsystem operate on them.

To shorten signals and increase readability, the compiler has predefined text macros for these delay properties: the strings "WD", "CD", and "CKD" respectively. The user may of course use either the full property name or the associated text macro interchangeably. All examples will be in terms of the predefined macros. When using the predefined text macros the equal sign and quotes should be omitted, just giving the delay range after the macro name.

All Verifier delay properties are pin properties. That is, the delay is applied at each pin to which the wire with the signal name containing the delay property is attached. A pin connected to the same signal that lacks a delay property is not delayed. For example consider a drawing with two bodies that use a signal RESET:



Then the behavior of RESET is:

```
RESET  .  .  .   0:0.0, 1:10.0, 0:20.0
                 ( at the driver )
RESET  .  .  .   0.0.0, R:12.0, 1:14.3,
                 F:22.0, 0:24.3 ( at B1 )
RESET  .  .  .   0.0.0, R:10.0, 1:11.5,
                 F:21.0, 0:22.5 ( at B2 )
```

Delay properties are handled this way so that systems where delays are different on different "stubs" of a net may be modelled.

6.7 EVALUATION DIRECTIVES

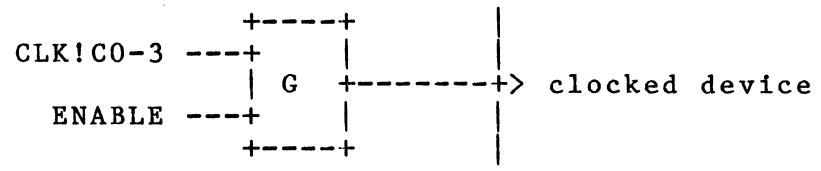
Evaluation directives are used for two purposes:

1. To facilitate the verification of designs that use tuned clocks. This is done by providing a description of how clock signals are tuned.

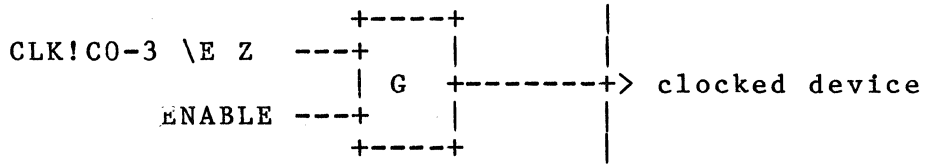
2. To facilitate the verification of designs that use "gated" clocks. Evaluation directives are defined that direct the Verifier to ensure that gating signals properly "envelope" clock signals.

EVALUATION DIRECTIVES FOR CLOCK TUNING

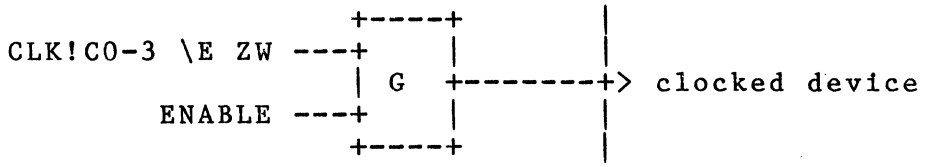
High performance designs often require the adjustment of clocks to compensate for circuit delays. A typical example is shown below:



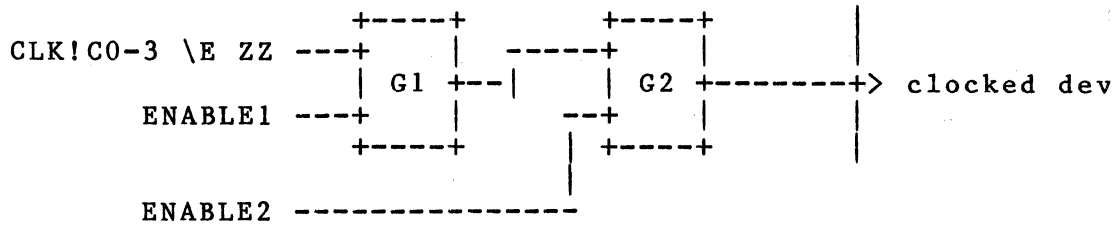
This is a typical circuit where a clocked device is conditionally clocked depending on whether the enable is asserted or not. Speed constraints may require that the signal CLK!CO-3 be generated so that the effective delay of the gate "G" and its top input wire is zero. An alternate way of viewing this situation is that the top input signal to gate G is generated so that the gates output signal is asserted from period 0 to 3 (when enabled). We indicate this to the Verifier using the evaluation directive "Z". A typical example is shown below:



The second evaluation directive for tuning is 'W', which says to set the minimum wire delay to zero, and to substract the minimum delay from the maximum delay. The 'W' evaluation directive is just used to zero out the minimum wire delay of the last wire on a clock path. For example, evaluation directives may be composed:



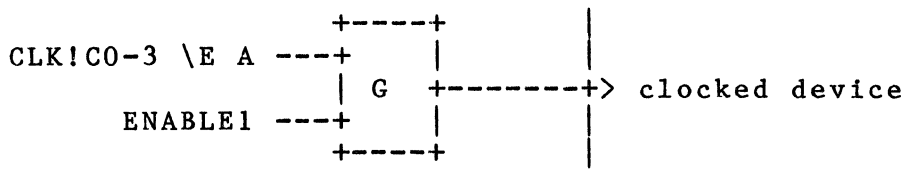
The 'ZW' means to treat the circuit as if the input wire delay, the delay of gate G and the minimum delay of the output wire delay is zero. Tuning directives may be combined to zero multiple levels of gating between the clocked signal and the clocked device:



If the clock is tuned with respect to the output of G2 the evaluation directive 'ZZ' is used -- the first "Z" sets the G1 and its input wire to zero, the second "Z" sets G2 and its input wire to zero. If the clock is tuned at the clock device's clock input the evaluation directive 'ZZW' should be used.

EVALUATION DIRECTIVES FOR CLOCK GATING

Correct performance of a digital system using gated clocks, requires that the gating signal be stable during the on-time (asserted time) of the clock. The evaluation directive 'A' is used to check this:



The 'A' indicates that ENABLE1 must be stable (S or 0 or 1) when that signal is controlling the gate:

- If G is an AND gate, ENABLE1 must be stable when CLK!CO-3 is high.
- If G is an OR gate, ENABLE1 must be stable when CLK!CO-3 is low.
- If G is any other kind of logic element, it is ignored.

If the signals do not meet the conditions specified an error is generated in the Verifier output report.

The 'A' directive may be used only on AND and OR gates where one input of the gate is driven by a clock signal (a signal with a 'C' or 'P' assertion).

TUNED AND GATED CLOCKS

Use the directive 'H' to verify designs with clocks that are both tuned and gated. The directive 'H' causes the Timing Verifier to zero the wire and the gate, and also to check that the enabling signal(s) is stable when the clock enables the gate.

MULTILEVEL COMPONENT DEFINITION

When a component is defined with multiple levels of primitives, it is desired that the evaluation directives refer to the entire path through the component, rather than to a single primitive that the component is made up of. If the component definition is a single level drawing, then the Timing Verifier automatically causes the evaluation directive string to count all of the primitives as one element. A user can also put the body property 'KEEPDIRECTIVE' on a primitive which will cause it to propagate the entire evaluation string through it, rather than taking the first evaluation letter off of it. This property is useful if a hierarchical definition for a component is used and the evaluation directives only want to increment once when going through the component.

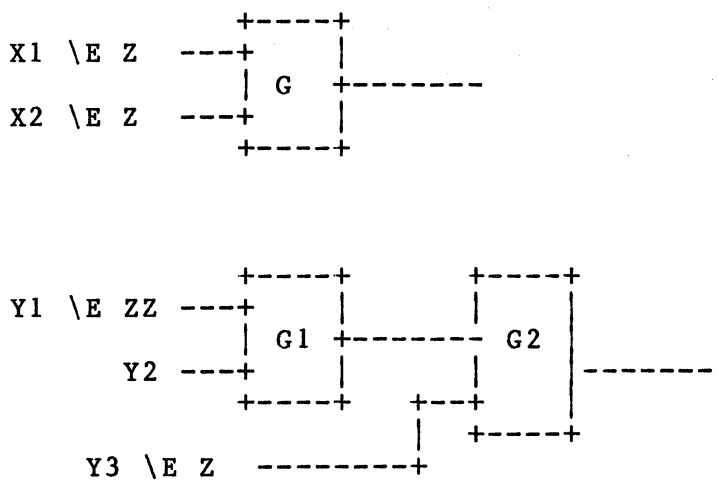
SUMMARY OF EVALUATION DIRECTIVES

Five evaluation directives are recognized by the Timing Verifier:

- W -- sets the minimum delay of the wire to zero and subtracts the minimum delay from the maximum delay.
- Z -- sets the wire delay and the gate delay to zero.
- A -- checks that the non-clock input(s) to a gate is stable when the clock input is enabling the gate. Directs the Timing Verifier to ignore all the inputs to the gate except the one with the I assertion.
- H -- sets the wire delay and the gate delay to zero and check that the non-clock input(s) to a gate is stable when the clock input is enabling the gate.
- I -- directs the Timing Verifier to ignore all the inputs to the gate except the one with the I assertion. The output of the gate is simply the input signal (with the assertion) delayed by the propagation delay of the gate. This directive may be used on any gate type but only one input to the gate may have an I assertion.

RESTRICTIONS

An evaluation directive may be applied to only one input of a gate. The diagram illustrates an unacceptable condition.



Timing Verifier Primitives

6.8 MODELLING COMPONENTS IN THE TIMING VERIFIER

Timing Verifier models are simply logic diagrams constructed from a specific set of parts called Timing Verifier primitives. Timing models may be hierarchical. If they are, the leaf drawings must be in terms of this parts set.

All Timing Verifier primitives may have an optional body property, TRANSITION, which takes the values SMOOTH or GLITCHY. The simulation of some primitives is modified based on this parameters. (Details are given below.) In addition, all Timing Verifier primitives have bubbleable pins. This feature allows negative edge triggering of latches, buffers to become inverters, etc.

The truth tables for the Timing Verifier primitives are given below. In the case where more than one entry applies to a given set of input conditions, the first entry will take precedence.

A complete list of the primitives is given below:

2 OR	2-input SIZE wide OR gate
3 OR	3-input SIZE wide OR gate
4 OR	4-input SIZE wide OR gate
5 OR	5-input SIZE wide OR gate
6 OR	6-input SIZE wide OR gate
7 OR	7-input SIZE wide OR gate
8 OR	8-input SIZE wide OR gate
2 AND	2-input SIZE wide AND gate
3 AND	3-input SIZE wide AND gate
4 AND	4-input SIZE wide AND gate
5 AND	5-input SIZE wide AND gate
6 AND	6-input SIZE wide AND gate
7 AND	7-input SIZE wide AND gate
8 AND	8-input SIZE wide AND gate
2 CHG	2-input SIZE wide CHANGE gate
3 CHG	3-input SIZE wide CHANGE gate
4 CHG	4-input SIZE wide CHANGE gate
5 CHG	5-input SIZE wide CHANGE gate
6 CHG	6-input SIZE wide CHANGE gate
7 CHG	7-input SIZE wide CHANGE gate
8 CHG	8-input SIZE wide CHANGE gate
XOR	2-input SIZE wide XOR gate
BUF	1-input SIZE wide BUFFER gate

OR	SIZE inputs to single bit output OR gate
AND	SIZE inputs to single bit output AND gate
CHG	SIZE inputs to single bit output CHANGE gate
THRESHOLD	1-input SIZE wide threshold gate
IDENTITY	1-input SIZE wide identity gate
RES	1-input SIZE wide resistor
TS BUF	SIZE wide tri-state driver with enable
LATCH	SIZE wide latch with enable
LATCH RS	SIZE wide latch with enable and asynchrous set and reset
REG	SIZE wide rising-edge triggered register
REG RS	SIZE wide rising-edge triggered register with asynchrous set and reset
2 MUX	SIZE wide 2-input multiplexer
4 MUX	SIZE wide 4-input multiplexer
8 MUX	SIZE wide 8-input multiplexer
SETUP HOLD	SIZE wide rising-edge setup and hold checker
SETUP RISE HOLD FALL	SIZE wide rising-edge setup and falling-edge hold checker
EDGE TO EDGE	SIZE wide rising-edge to rising-edge skew checker
MIN PULSE WIDTH	SIZE wide minimum pulse width checker
TRANSMISSION GATE	SIZE wide bi-directional transmission gate

AND, OR, CHANGE and XOR FUNCTIONS

The truth tables for the AND, OR, CHANGE(CHG), and XOR functions are given in the following tables:

AND	0	1	S	R	F	C	U	Z
0	0	0	0	0	0	0	0	0
1	0	1	S	R	F	C	U	U
S	0	S	S	R	F	C	U	U
R	0	R	R	R	C	C	U	U
F	0	F	F	C	F	C	U	U
C	0	C	C	C	C	C	U	U
U	0	U	U	U	U	U	U	U
Z	0	U	U	U	U	U	U	U

OR	0	1	S	R	F	C	U	Z
0	0	1	S	R	F	C	U	U
1	1	1	1	1	1	1	1	1
S	S	1	S	R	F	C	U	U
R	R	1	R	R	C	C	U	U
F	F	1	F	C	F	C	U	U
C	C	1	C	C	C	C	U	U
U	U	1	U	U	U	U	U	U
Z	U	1	U	U	U	U	U	U

CHG	0	1	S	R	F	C	U	Z
0	S	S	S	C	C	C	U	U
1	S	S	S	C	C	C	U	U
S	S	S	S	C	C	C	U	U
R	C	C	C	C	C	C	U	U
F	C	C	C	C	C	C	U	U
C	C	C	C	C	C	C	U	U
U	U	U	U	U	U	U	U	U
Z	U	U	U	U	U	U	U	U

XOR	0	1	S	R	F	C	U	Z
0	0	1	S	R	F	C	U	U
1	1	0	S	F	R	C	U	U
S	S	S	S	C	C	C	U	U
R	R	F	C	C	C	C	U	U
F	F	R	C	C	C	C	U	U
C	C	C	C	C	C	C	U	U
U	U	U	U	U	U	U	U	U
Z	U	U	U	U	U	U	U	U

These parts are simulated in the same way for TRANSITION = SMOOTH and TRANSITION = GLITCHY.

TS BUF and TS BUS FUNCTIONS

The truth tables for the TS BUF primitive and related TS BUS are given in the following tables.

TS BUF	ENABLE INPUT 0	1	S	R	F	C	U	Z
0	Z	0	U	C	C	C	U	U
1	Z	1	U	C	C	C	U	U
S	Z	S	U	C	C	C	U	U
R	Z	R	U	C	C	C	U	U
F	Z	F	U	C	C	C	U	U
C	Z	C	U	C	C	C	U	U
U	Z	U	U	U	U	U	U	U
Z	Z	U	U	U	U	U	U	U

DATA INPUT

(tri-state mode)

TS BUF	ENABLE INPUT 0	1	S	R	F	C	U	Z
0	Z	0	0	C	C	C	U	U
1	Z	1	1	C	C	C	U	U
S	Z	S	S	C	C	C	U	U
R	Z	R	R	C	C	C	U	U
F	Z	F	F	C	C	C	U	U
C	Z	C	C	C	C	C	U	U
U	Z	U	U	U	U	U	U	U
Z	Z	U	U	U	U	U	U	U

DATA INPUT

(dot-or mode)

TS BUS I2 \ I1	0	1	S	R	F	C	U	Z
0	0	U	U	U	F	U	U	0
1	U	1	U	R	U	U	U	1
S	U	U	U	U	U	U	U	S
R	U	R	U	R	U	U	U	R
F	F	U	U	U	F	U	U	F
C	U	U	U	U	U	U	U	C
U	U	U	U	U	U	U	U	U
Z	0	1	S	R	F	C	U	Z

(tri-state mode)

TS BUS I2 \ I1	0	1	S	R	F	C	U	Z
0	0	S	S	R	F	C	U	0
1	S	1	S	R	F	C	U	1
S	S	S	S	C	C	C	U	S
R	R	R	C	R	C	C	U	R
F	F	F	C	C	F	C	U	F
C	C	C	C	C	C	C	U	C
U	U	U	U	U	U	U	U	U
Z	0	1	S	R	F	C	U	Z

(dot-or mode)

These parts are simulated in the same way for TRANSITION = SMOOTH and TRANSITION = GLITCHY.

DOT GATES

The Timing Verifier simulates multiple driven nets (buses) by inserting a gate in the network. All the drivers of the bus are reconnected to the gate's inputs. The output of the gate drives all the inputs on the bus. If the bus is dot-or (dot-and), the inserted gate is an OR (AND) gate. If the bus is a tri-state bus, the inserted gate is a TS BUS with one of the two logic functions shown below.

Note that both the TS BUF and the TS BUS have two modes of operation. The mode used for simulation depends on whether the value of the TS_BUS_TYPE directive in the Verifier command file is "DOT_TS" or "DOT_OR". (See the Timing Verifier Directives Summary in this chapter of the manual).

BUF AND THRESHOLD FUNCTIONS

The truth tables for the BUF and THRESHOLD primitives are given in the following tables:

	BUF	OUTPUT
INPUT	0	0
	1	1
	S	S
	R	R
	F	F
	C	C
	U	U
	Z	U

	THRESHOLD	OUTPUT
INPUT	0	C
	1	1
	S	C
	R	C
	F	C
	C	C
	U	U
	Z	U

These parts are simulated in the same way for TRANSITION = SMOOTH and TRANSITION = GLITCHY.

RES AND IDENTITY FUNCTIONS

The truth tables for the RES and IDENTITY primitives are given in the following tables:

	RES	OUTPUT
INPUT	0	0
	1	1
	S	S
	R	R
	F	F
	C	C
	U	U
	Z	Z

	IDENTITY	OUTPUT
INPUT	0	0
	1	1
	S	S
	R	R
	F	F
	C	C
	U	U
	Z	Z

These parts are simulated in the same way for TRANSITION = SMOOTH and TRANSITION = GLITCHY.

LATCH PRIMITIVE

The LATCH primitive has a DATA and EN input. Note: If EN is bubbled the iverse of the chart should be followed.

LATCH:

EN	LASTOUTPUT	DATA	OUTPUT
0	{0,1,S}	X	{0,1,S}
0	{R,F,C,U,Z}	X	S
1	X	{0,1,S,R,F,C}	{0,1,S,R,F,C}
1	X	{U,Z}	{U,U}
R	= DATA	{0,1,U,Z}	{0,1,U,U}
R	= DATA	S	S
	If no input transition since EN was last 1 or R and the latch is being simulated SMOOTH.		
R	= DATA	all other cond.	C

R	<> DATA	{U,Z}	U
R	0	{1,S}	R
R	1	{0,S}	F
R	{R,F,C,U,Z}	{0,1,S}	C
R	{R,1}	R	R
R	{F,0}	F	F
R	all other conditions		C
F	= DATA	{0,1,S,U,Z}	{0,1,S,U,U}
F	= DATA	{R,F,C}	C
F	X	{U,Z}	{U,U}
F	0	{1,S}	R
F	1	{0,S}	F
F	C	{0,1,S}	{0,1,S}
F	{R,1}	R	R
F	{F,0}	R	F
F	all other conditions		C
S	= DATA	X	LASTOUTPUT
S	<> DATA	{0,1,S}	S
S	1	R	R
S	0	F	F
S	<> 1	R	C
S	<> 0	F	C
S	X	C	C
S	all other inputs		U
C	X	{U,Z}	{U,U}
C	= DATA	{0,1,S,R,F,C}	{0,1,S,R,F,C}

C	all other inputs	C
Z	X	X	U
U	X	X	U

If the INPUT undergoes a transition while the latch is closing, then a setup/hold time violation has occured. Under these conditions the latch is simulated in one of three ways depending on the value of LATCH-ERR-MODEL: LATCH_ERR_MODEL = OPEN; LATCH_ERR_MODEL = CLOSED or LATCH_ERR_MODEL = CONSERVATIVE;

LATCH_ERR_MODEL = OPEN:

LASTEN	LASTOUTPUT	DATA	OUTPUT
F	X	{U,Z}	{U,U}
F	0	{0,1,S}	R
F	1	{0,1,S}	F
F	C	{0,1,S}	{0,1,S}
F	{S,R,F,C,Z}	{0,1,S}	C
F	{R,1}	R	R
F	{F,0}	F	F
F	all other conditions		C

LATCH_ERR_MODEL = CLOSED:

LASTEN	LASTOUTPUT	DATA	OUTPUT
F	{0,1,S}	X	{0,1,S}
F	{R,F,C,U,Z}	X	S

LATCH_ERR_MODEL = CONSERVATIVE:

LASTEN	LASTOUTPUT	DATA	OUTPUT
F	X	{U,Z}	{U,U}
F	0	{0,1,S}	R
F	1	{0,1,S}	F
F	{S,R,F,C,,U}	{0,1,S}	C
F	{R,1}	R	R
F	{F,0}	F	F
F	all other conditions		C

The body property TRANSITION determines whether the output of the LATCH primitive should change when it is enabled, even if the input has not changed. When the body property TRANSITION = GLITCHY is attached to a LATCH primitive, the output of the LATCH will always change even if the input remains stable. If TRANSITION = SMOOTH is attached, or no TRANSITION property is attached, the output of the LATCH will not change if the input is always stable.

The LATCH RS primitive is the same as the LATCH except that it also has asynchrous RESET and SET inputs. First the LATCH output is computed for the current input values, then the SET RESET function is applied to the outputs. The SET RESET function is described in the next section.

SET RESET FUNCTION

The SET RESET function is composed with the LATCH function to form a LATCH RS and the REG function to form a REG RS. It is not directly accessible as a Timing Verifier Primitive. The SET RESET function is different for TRANSITION = SMOOTH and GLITCHY. The function inherits its TRANSITION property from the LATCH RS or REG RS of which it is a part.

GLITCHY:

R	S	OLDOUTPUT	NEWOUTPUT
0	0	X	OLDOUTPUT
0	X	1	1
0	1	<> 1	1
0	<> {0,1}	<> 1	CHG(OLDOUTPUT,S)
X	0	0	0
1	0	X	0
<>{0,1}	0	X	CHG(OLDOUTPUT,R)
X	all other cases		CHG(OLDOUTPUT,R,S)

where CH is the change function defined on Page 6-21.

SMOOTH:

R	S	OLDOUTPUT	NEWOUTPUT
0	0	X	OLDOUTPUT
0	X	1	1
0	1	X	1
0	R	0	R
0	<> {1,R}	X	CH(OLDOUTPUT,S)
X	0	0	0
1	0	X	0
R	0	0	F
<>{1,R}	0	X	CH(OLDOUTPUT,R,S)
{1,R}	F	{0,1,S}	{0,F,F,}
{1,R}	F	<> {0,1,S}	CH(OLDOUTPUT,R,S)
F	{1,R}	{0,1,S}	{R,1,R}
F	{1,R}	<> {0,1,S}	CH(OLDOUTPUT,R,S)

REG FUNCTION

The REG primitive implements a rising edge triggered register.

CLOCK	LASTCLOCK	INPUT	LASTOUTPUT	NEXTOUTPUT
1	0	{0,1}	{0.1}	LASTOUTPUT
1	0	{1,R}	{0,R}	R
1	0	{0,F}	{1,F}	F
1	0	S	S	S
If the REG is SMOOTH and there were no input transitions.				
1	1	X	<>{0,1,S}	S
1	1	X	= {0,1,S}	LASTOUTPUT
1	S	X	X	LASTOUTPUT
1	R	{0,1}	S	LASTOUTPUT
1	F	X	<>{0,1,S}	S
1	{C,U,Z}	INPUT =	LASTOUTPUT	LASTOUTPUT
1	{C,U,Z}	INPUT <>	LASTOUTPUT	S
{C,R}	X	[0,1]	[0,1]	LASTOUTPUT
{C,R}	X	{1,R}	{0,R}	R
{C,R}	X	{0,F}	{1,F}	F
{C,R}	X	S	S	S
If the REG is SMOOTH and there were no input transitions.				
{0,S,F}	X	X	<>{0,1,S}	S
{0,S,F}	X	X	={0,1,S}	LASTOUTPUT
{U,Z}	X	X	X	U

The body property TRANSITION determines whether the output of the REG primitive should change when it is clocked, even if the input has not changed. When the body property TRANSITION = GLITCHY is attached to a REG primitive, the output of the REG will always change even if the input remains stable. If TRANSITION = SMOOTH is

attached, or no TRANSITION property is attached, the output of the REG will not change if the input is always stable.

The REG RS primitive is the same as the REG except that it also has asynchrous R and S inputs. First the REG output is computed for the current input values, then the SET RESET function is applied to the output.

THE 2, 4 AND 8 MUX FUNCTIONS

The 2 MUX, 4 MUX, and 8 MUX primitives implement 2-input, 4-input, and 8-input multiplexers. If any of the select inputs on these multiplexers has a known value of 0 or 1, then only the possibly selected data inputs will be looked at when calculating the output value. If more than one data input might be selected, the output value is calculated by using the CHANGE function on the set of selected data inputs.

If the N MUX has no TRANSITION property or TRANSITION = GLITCHY, then any input transition causes an output transition of the appropriate slope. If TRANSITION = SMOOTH, then if the output state before and after an input transition is the same, there is no output transition.

SETUP HOLD FUNCTION

The SETUP HOLD primitive has a clock and data input. It will generate an error message in the output listing if the data input is not stable from SETUP nsec's before the rising edge of the clock until HOLD nsec's after the rising edge of the clock. SETUP and HOLD are timing parameters given to this primitive. This primitive is normally used to check the set-up and hold times of registers and latches. This primitive has an optional enable input, which if specified, turns the checking on and off. If the enable input is any value other than ZERO, then checking is enabled. If checking is enabled anytime during the rising edge of the clock input, then checking will be done for that edge.

SETUP RISE HOLD FALL FUNCTION

The SETUP RISE HOLD FALL primitive has a clock and data input. It will generate an error message in the output listing if the data input is not stable from SETUP nsec's before the rising edge of the clock, while the clock is rising, while the clock is true, during the falling edge of the clock, until HOLD nsec's after the falling edge of the clock. SETUP and HOLD are timing parameters given to this primitive. This primitive is normally used to check the set-up and hold times of data being written into memories.

This primitive has an optional enable input which can be used to turn off checking. If the enable input is given, then any value other than ZERO will cause checking to be enabled. If checking is enabled anywhere between the beginning of the rising edge to the end of the falling edge, then checking will be done for that clock pulse.

EDGE TO EDGE FUNCTION

The EDGE TO EDGE primitive has two clock inputs, CK1 and CK2. It checks that the beginning of a RISING edge on CK2 is at least a minimum delay from the end of a RISING edge on CK1 and that the end of a RISING edge on CK2 is no more than a maximum delay from the beginning of a RISING edge on CK1. The delay parameter is used to specify the minimum and maximum delays used. Only rising delays are used. If there is no edge on CK2, then no error message will be generated. This primitive has an optional enable input, which if specified, turns the checking on and off. If the enable input is any value other than ZERO, then checking is enabled. If checking is enabled anytime during the rising edge of CK1, then the checking will be done for that edge.

MIN PULSE WIDTH

The MIN PULSE WIDTH primitive has one data input. It has two timing parameters LOW and HIGH. It checks that its data input has no pulses on it that are low for less than LOW nsec's, and that it has no pulses on it that are high for less than HIGH nsec's. This primitive has an optional enable input, which if specified, turns the checking on and off. If the enable input is any value other than ZERO, then checking is enabled. If checking is enabled anytime during a given pulse, then the width of that pulse is checked.

TRANSMISSION GATE

The TRANSMISSION GATE primitive has an enable input EN, and two bi-directional pins T1 and T2. If the enable input is ZERO, then both T1 and T2 are set to high-impedence. If EN is ONE, then T1 and T2 are tied together using the same function as the tri-state bus (TS BUS), which is defined on Page 5-22.

BUBBLING OF PRIMITIVE PINS

Each input and output of every primitive may be "bubbled" independently. (See Graphics Editor, BUBBLE command.) When this is done, it is as if an inverting buffer were inserted between the signal (input or output) and the primitive itself. The characteristics of the primitive

itself are not changed in any way. This is useful for creating inverting buffers (by bubbling the input or output of a BUF), nand gates. nor gates, negative edge triggered registers, etc.

The use of a bubbled input on a MIN PULSE WIDTH primitive is a good example of the statement that the primitive itself is unchanged. In order to check a low asserted signal (e.g., CK) to make sure that it is low for at least 20.0 nsec one may use a MIN PULSE WIDTH primitive with a bubbled input and a HIGH=20.0 property.

6.9 SIGNAL STRENGTHS IN THE TIMING VERIFIER

The output of a Timing Verifier primitive may assume one of three strengths, HARD, SOFT or UNDRIVEN. Strengths are required to correctly model circuit nodes that have multiple outputs on them when those outputs have different drive capabilities. A typical example of this is a tristate bus that is pulled-up with a resistor. When none of the tristate drivers are on, the bus should be in the one state. When a single driver drives the bus to zero, the bus should assume the zero state. Thus we need some way of modelling the fact that the resistor output is weaker than a bus driver output.

By default, the output of all devices except RES, IDENTITY and wire gates is HARD. The output of a resistor primitive is SOFT, unless the input to the resistor is UNDRIVEN, then the output is UNDRIVEN. The output of the IDENTITY primitive is the same as its input. The output strength of a wire gate is the same as the strongest input strength.

The default output strength of a primitive may be specified by attaching to it a body property STRENGTH which takes the values HARD, SOFT and UNDRIVEN. All primitives except the resistor and identity gate and wire gates ignore the strengths of their input signals. The function of a resistor and identity gate was described above. The functions of the dot gates are shown in the tables below.

The tables have four indices. Indices 1 and 3 are the strength and value of the first input, indices 2 and 4 are the strength and value of the second input.

DOT OR

HARD,HARD,X0,X0	X0	HARD,HARD,X0,X1	X1
HARD,HARD,X0,Xs	Xs	HARD,HARD,X0,Xz	X0
HARD,HARD,X0,Xc	Xc	HARD,HARD,X0,Xr	Xr
HARD,HARD,X0,Xf	Xf	HARD,HARD,X0,Xu	Xu

HARD,HARD,X1,X0	X1	HARD,HARD,X1,X1	X1
HARD,HARD,X1,Xs	X1	HARD,HARD,X1,Xz	X1
HARD,HARD,X1,Xc	X1	HARD,HARD,X1,Xr	X1
HARD,HARD,X1,Xf	X1	HARD,HARD,X1,Xu	X1
HARD,HARD,Xs,X0	Xs	HARD,HARD,Xs,X1	X1
HARD,HARD,Xs,Xs	Xs	HARD,HARD,Xs,Xz	Xs
HARD,HARD,Xs,Xc	Xc	HARD,HARD,Xs,Xr	Xr
HARD,HARD,Xs,Xf	Xf	HARD,HARD,Xs,Xu	Xu
HARD,HARD,Xc,X0	Xc	HARD,HARD,Xc,X1	X1
HARD,HARD,Xc,Xs	Xc	HARD,HARD,Xc,Xz	Xc
HARD,HARD,Xc,Xc	Xc	HARD,HARD,Xc,Xr	Xc
HARD,HARD,Xc,Xf	Xc	HARD,HARD,Xc,Xu	Xu
HARD,HARD,Xr,X0	Xr	HARD,HARD,Xr,X1	X1
HARD,HARD,Xr,Xs	Xr	HARD,HARD,Xr,Xz	Xr
HARD,HARD,Xr,Xc	Xc	HARD,HARD,Xr,Xr	Xr
HARD,HARD,Xr,Xf	Xc	HARD,HARD,Xr,Xu	Xu
HARD,HARD,Xf,X0	Xf	HARD,HARD,Xf,X1	X1
HARD,HARD,Xf,Xs	Xf	HARD,HARD,Xf,Xz	Xf
HARD,HARD,Xf,Xc	Xc	HARD,HARD,Xf,Xr	Xc
HARD,HARD,Xf,Xf	Xf	HARD,HARD,Xf,Xu	Xu
HARD,HARD,Xz,X0	X0	HARD,HARD,Xz,X1	X1
HARD,HARD,Xz,Xs	Xs	HARD,HARD,Xz,Xz	Xz
HARD,HARD,Xz,Xc	Xc	HARD,HARD,Xz,Xr	Xr
HARD,HARD,Xz,Xf	Xf	HARD,HARD,Xz,Xu	Xu
HARD,HARD,Xu,X0	Xu	HARD,HARD,Xu,X1	X1
HARD,HARD,Xu,Xs	Xu	HARD,HARD,Xu,Xz	Xu
HARD,HARD,Xu,Xc	Xu	HARD,HARD,Xu,Xr	Xu
HARD,HARD,Xu,Xf	Xu	HARD,HARD,Xu,Xu	Xu
HARD,SOFT,X0,X0	X0	HARD,SOFT,X0,X1	X1
HARD,SOFT,X0,Xs	Xs	HARD,SOFT,X0,Xz	X0
HARD,SOFT,X0,Xc	Xc	HARD,SOFT,X0,Xr	Xr
HARD,SOFT,X0,Xf	Xf	HARD,SOFT,X0,Xu	Xu
HARD,SOFT,X1,X0	X1	HARD,SOFT,X1,X1	X1
HARD,SOFT,X1,Xs	X1	HARD,SOFT,X1,Xz	X1
HARD,SOFT,X1,Xc	X1	HARD,SOFT,X1,Xr	X1
HARD,SOFT,X1,Xf	X1	HARD,SOFT,X1,Xu	X1
HARD,SOFT,Xs,X0	Xs	HARD,SOFT,Xs,X1	X1
HARD,SOFT,Xs,Xs	Xs	HARD,SOFT,Xs,Xz	Xs
HARD,SOFT,Xs,Xc	Xc	HARD,SOFT,Xs,Xr	Xr
HARD,SOFT,Xs,Xf	Xf	HARD,SOFT,Xs,Xu	Xu
HARD,SOFT,Xc,X0	Xc	HARD,SOFT,Xc,X1	X1
HARD,SOFT,Xc,Xs	Xc	HARD,SOFT,Xc,Xz	Xc
HARD,SOFT,Xc,Xc	Xc	HARD,SOFT,Xc,Xr	Xc
HARD,SOFT,Xc,Xf	Xc	HARD,SOFT,Xc,Xu	Xu
HARD,SOFT,Xr,X0	Xr	HARD,SOFT,Xr,X1	X1
HARD,SOFT,Xr,Xs	Xr	HARD,SOFT,Xr,Xz	Xr
HARD,SOFT,Xr,Xc	Xr	HARD,SOFT,Xr,Xr	Xr
HARD,SOFT,Xr,Xf	Xr	HARD,SOFT,Xr,Xu	Xu
HARD,SOFT,Xf,X0	Xf	HARD,SOFT,Xf,X1	X1
HARD,SOFT,Xf,Xs	Xf	HARD,SOFT,Xf,Xz	Xf
HARD,SOFT,Xf,Xc	Xf	HARD,SOFT,Xf,Xr	Xf
HARD,SOFT,Xf,Xf	Xf	HARD,SOFT,Xf,Xu	Xu

HARD,SOFT,Xz,X0	X0	HARD,SOFT,Xz,X1	X1
HARD,SOFT,Xz,Xs	Xs	HARD,SOFT,Xz,Xz	Xz
HARD,SOFT,Xz,Xc	Xc	HARD,SOFT,Xz,Xr	Xr
HARD,SOFT,Xz,Xf	Xf	HARD,SOFT,Xz,Xu	Xu
HARD,SOFT,Xu,X0	Xu	HARD,SOFT,Xu,X1	X1
HARD,SOFT,Xu,Xs	Xu	HARD,SOFT,Xu,Xz	Xu
HARD,SOFT,Xu,Xc	Xu	HARD,SOFT,Xu,Xr	Xu
HARD,SOFT,Xu,Xf	Xu	HARD,SOFT,Xu,Xu	Xu

HARD,UNDRIVEN,X0,X0	X0	HARD,UNDRIVEN,X0,X1	X1
HARD,UNDRIVEN,X0,Xs	Xs	HARD,UNDRIVEN,X0,Xz	X0
HARD,UNDRIVEN,X0,Xc	Xc	HARD,UNDRIVEN,X0,Xr	Xr
HARD,UNDRIVEN,X0,Xf	Xf	HARD,UNDRIVEN,X0,Xu	Xu
HARD,UNDRIVEN,X1,X0	X1	HARD,UNDRIVEN,X1,X1	X1
HARD,UNDRIVEN,X1,Xs	X1	HARD,UNDRIVEN,X1,Xz	X1
HARD,UNDRIVEN,X1,Xc	X1	HARD,UNDRIVEN,X1,Xr	X1
HARD,UNDRIVEN,X1,Xf	X1	HARD,UNDRIVEN,X1,Xu	X1
HARD,UNDRIVEN,Xs,X0	Xs	HARD,UNDRIVEN,Xs,X1	Xs
HARD,UNDRIVEN,Xs,Xs	Xs	HARD,UNDRIVEN,Xs,Xz	Xs
HARD,UNDRIVEN,Xs,Xc	Xs	HARD,UNDRIVEN,Xs,Xr	Xs
HARD,UNDRIVEN,Xs,Xf	Xs	HARD,UNDRIVEN,Xs,Xu	Xu
HARD,UNDRIVEN,Xc,X0	Xc	HARD,UNDRIVEN,Xc,X1	Xc
HARD,UNDRIVEN,Xc,Xs	Xc	HARD,UNDRIVEN,Xc,Xz	Xc
HARD,UNDRIVEN,Xc,Xc	Xc	HARD,UNDRIVEN,Xc,Xr	Xc
HARD,UNDRIVEN,Xc,Xf	Xc	HARD,UNDRIVEN,Xc,Xu	Xc
HARD,UNDRIVEN,Xr,X0	Xr	HARD,UNDRIVEN,Xr,X1	X1
HARD,UNDRIVEN,Xr,Xs	Xr	HARD,UNDRIVEN,Xr,Xz	Xr
HARD,UNDRIVEN,Xr,Xc	Xr	HARD,UNDRIVEN,Xr,Xr	Xr
HARD,UNDRIVEN,Xr,Xf	Xr	HARD,UNDRIVEN,Xr,Xu	Xr
HARD,UNDRIVEN,Xf,X0	Xf	HARD,UNDRIVEN,Xf,X1	X1
HARD,UNDRIVEN,Xf,Xs	Xf	HARD,UNDRIVEN,Xf,Xz	Xf
HARD,UNDRIVEN,Xf,Xc	Xf	HARD,UNDRIVEN,Xf,Xr	Xf
HARD,UNDRIVEN,Xf,Xf	Xf	HARD,UNDRIVEN,Xf,Xu	Xf
HARD,UNDRIVEN,Xz,X0	X0	HARD,UNDRIVEN,Xz,X1	X1
HARD,UNDRIVEN,Xz,Xs	Xs	HARD,UNDRIVEN,Xz,Xz	Xz
HARD,UNDRIVEN,Xz,Xc	Xc	HARD,UNDRIVEN,Xz,Xr	Xr
HARD,UNDRIVEN,Xz,Xf	Xf	HARD,UNDRIVEN,Xz,Xu	Xu
HARD,UNDRIVEN,Xu,X0	Xu	HARD,UNDRIVEN,Xu,X1	Xu
HARD,UNDRIVEN,Xu,Xs	Xu	HARD,UNDRIVEN,Xu,Xz	Xu
HARD,UNDRIVEN,Xu,Xc	Xu	HARD,UNDRIVEN,Xu,Xr	Xu
HARD,UNDRIVEN,Xu,Xf	Xu	HARD,UNDRIVEN,Xu,Xu	Xu

The DOT OR function for strength 1 SOFT, and strength 2 HARD is obtained by transposing the values of the strength 2 SOFT, and strength 1 HARD table.

The DOT OR function for strength 1 SOFT, and strength 2 SOFT is identical to the strength 1 HARD, and strength 2 HARD table.

The DOT OR function for strength 1 SOFT, and strength 2 UNDRIVEN is identical to the strength 1 HARD, and strength 2 UNDRIVEN table.

The DOT OR function for strength 1 UNDRIVEN, and strength 2 HARD is is obtained by transposing the values of the strength 1 HARD, and strength 2 UNDRIVEN table.

The DOT OR function for strength 1 UNDRIVEN, and strength 2 SOFT is is obtained by transposing the values of the strength 1 SOFT, and strength 2 UNDRIVEN table.

UNDRIVEN,UNDRIVEN,X0,X0	X0	UNDRIVEN,UNDRIVEN,X0,X1	Xs
UNDRIVEN,UNDRIVEN,X0,Xs	Xs	UNDRIVEN,UNDRIVEN,X0,Xz	X0
UNDRIVEN,UNDRIVEN,X0,Xc	Xs	UNDRIVEN,UNDRIVEN,X0,Xr	Xs
UNDRIVEN,UNDRIVEN,X0,Xf	Xs	UNDRIVEN,UNDRIVEN,X0,Xu	Xu
UNDRIVEN,UNDRIVEN,X1,X0	Xs	UNDRIVEN,UNDRIVEN,X1,X1	X1
UNDRIVEN,UNDRIVEN,X1,Xs	Xs	UNDRIVEN,UNDRIVEN,X1,Xz	X1
UNDRIVEN,UNDRIVEN,X1,Xc	Xs	UNDRIVEN,UNDRIVEN,X1,Xr	Xs
UNDRIVEN,UNDRIVEN,X1,Xf	Xs	UNDRIVEN,UNDRIVEN,X1,Xu	Xu
UNDRIVEN,UNDRIVEN,Xs,X0	Xs	UNDRIVEN,UNDRIVEN,Xs,X1	Xs
UNDRIVEN,UNDRIVEN,Xs,Xs	Xs	UNDRIVEN,UNDRIVEN,Xs,Xz	Xs
UNDRIVEN,UNDRIVEN,Xs,Xc	Xs	UNDRIVEN,UNDRIVEN,Xs,Xr	Xs
UNDRIVEN,UNDRIVEN,Xs,Xf	Xs	UNDRIVEN,UNDRIVEN,Xs,Xu	Xu
UNDRIVEN,UNDRIVEN,Xc,X0	Xs	UNDRIVEN,UNDRIVEN,Xc,X1	Xs
UNDRIVEN,UNDRIVEN,Xc,Xs	Xs	UNDRIVEN,UNDRIVEN,Xc,Xz	Xs
UNDRIVEN,UNDRIVEN,Xc,Xc	Xs	UNDRIVEN,UNDRIVEN,Xc,Xr	Xs
UNDRIVEN,UNDRIVEN,Xc,Xf	Xs	UNDRIVEN,UNDRIVEN,Xc,Xu	Xu
UNDRIVEN,UNDRIVEN,Xr,X0	Xs	UNDRIVEN,UNDRIVEN,Xr,X1	Xs
UNDRIVEN,UNDRIVEN,Xr,Xs	Xs	UNDRIVEN,UNDRIVEN,Xr,Xz	Xs
UNDRIVEN,UNDRIVEN,Xr,Xc	Xs	UNDRIVEN,UNDRIVEN,Xr,Xr	Xs
UNDRIVEN,UNDRIVEN,Xr,Xf	Xs	UNDRIVEN,UNDRIVEN,Xr,Xu	Xu
UNDRIVEN,UNDRIVEN,Xf,X0	Xs	UNDRIVEN,UNDRIVEN,Xf,X1	Xs
UNDRIVEN,UNDRIVEN,Xf,Xs	Xs	UNDRIVEN,UNDRIVEN,Xf,Xz	Xs
UNDRIVEN,UNDRIVEN,Xf,Xc	Xs	UNDRIVEN,UNDRIVEN,Xf,Xr	Xs
UNDRIVEN,UNDRIVEN,Xf,Xf	Xs	UNDRIVEN,UNDRIVEN,Xf,Xu	Xu
UNDRIVEN,UNDRIVEN,Xz,X0	X0	UNDRIVEN,UNDRIVEN,Xz,X1	X1
UNDRIVEN,UNDRIVEN,Xz,Xs	Xs	UNDRIVEN,UNDRIVEN,Xz,Xz	Xz
UNDRIVEN,UNDRIVEN,Xz,Xc	Xs	UNDRIVEN,UNDRIVEN,Xz,Xr	Xs
UNDRIVEN,UNDRIVEN,Xz,Xf	Xs	UNDRIVEN,UNDRIVEN,Xz,Xu	Xu
UNDRIVEN,UNDRIVEN,Xu,X0	Xu	UNDRIVEN,UNDRIVEN,Xu,X1	Xu
UNDRIVEN,UNDRIVEN,Xu,Xs	Xu	UNDRIVEN,UNDRIVEN,Xu,Xz	Xu
UNDRIVEN,UNDRIVEN,Xu,Xc	Xu	UNDRIVEN,UNDRIVEN,Xu,Xr	Xu
UNDRIVEN,UNDRIVEN,Xu,Xf	Xu	UNDRIVEN,UNDRIVEN,Xu,Xu	Xu

DOT AND

HARD,HARD,X0,X0	X0	HARD,HARD,X0,X1	X0
HARD,HARD,X0,Xs	X0	HARD,HARD,X0,Xz	X0
HARD,HARD,X0,Xc	X0	HARD,HARD,X0,Xr	X0
HARD,HARD,X0,Xf	X0	HARD,HARD,X0,Xu	X0
HARD,HARD,X1,X0	X0	HARD,HARD,X1,X1	X1

HARD,HARD,Xl,Xs	Xs	HARD,HARD,Xl,Xz	Xl
HARD,HARD,Xl,Xc	Xc	HARD,HARD,Xl,Xr	Xr
HARD,HARD,Xl,Xf	Xf	HARD,HARD,Xl,Xu	Xu
HARD,HARD,Xs,X0	X0	HARD,HARD,Xs,Xl	Xs
HARD,HARD,Xs,Xs	Xs	HARD,HARD,Xs,Xz	Xs
HARD,HARD,Xs,Xc	Xc	HARD,HARD,Xs,Xr	Xr
HARD,HARD,Xs,Xf	Xf	HARD,HARD,Xs,Xu	Xu
HARD,HARD,Xc,X0	X0	HARD,HARD,Xc,Xl	Xc
HARD,HARD,Xc,Xs	Xc	HARD,HARD,Xc,Xz	Xc
HARD,HARD,Xc,Xc	Xc	HARD,HARD,Xc,Xr	Xc
HARD,HARD,Xc,Xf	Xf	HARD,HARD,Xc,Xu	Xu
HARD,HARD,Xr,X0	X0	HARD,HARD,Xr,Xl	Xr
HARD,HARD,Xr,Xs	Xr	HARD,HARD,Xr,Xz	Xr
HARD,HARD,Xr,Xc	Xc	HARD,HARD,Xr,Xr	Xr
HARD,HARD,Xr,Xf	Xc	HARD,HARD,Xr,Xu	Xu;
HARD,HARD,Xf,X0	X0	HARD,HARD,Xf,Xl	Xf
HARD,HARD,Xf,Xs	Xf	HARD,HARD,Xf,Xz	Xf
HARD,HARD,Xf,Xc	Xc	HARD,HARD,Xf,Xr	Xc
HARD,HARD,Xf,Xf	Xf	HARD,HARD,Xf,Xu	Xu;
HARD,HARD,Xz,X0	X0	HARD,HARD,Xz,Xl	Xl
HARD,HARD,Xz,Xs	Xs	HARD,HARD,Xz,Xz	Xz
HARD,HARD,Xz,Xc	Xc	HARD,HARD,Xz,Xr	Xr
HARD,HARD,Xz,Xf	Xf	HARD,HARD,Xz,Xu	Xu;
HARD,HARD,Xu,X0	X0	HARD,HARD,Xu,Xl	Xu
HARD,HARD,Xu,Xs	Xu	HARD,HARD,Xu,Xz	Xu
HARD,HARD,Xu,Xc	Xu	HARD,HARD,Xu,Xr	Xu
HARD,HARD,Xu,Xf	Xu	HARD,HARD,Xu,Xu	Xu

HARD,SOFT,X0,X0	X0	HARD,SOFT,X0,Xl	X0
HARD,SOFT,X0,Xs	X0	HARD,SOFT,X0,Xz	X0
HARD,SOFT,X0,Xc	X0	HARD,SOFT,X0,Xr	X0
HARD,SOFT,X0,Xf	X0	HARD,SOFT,X0,Xu	X0
HARD,SOFT,Xl,X0	X0	HARD,SOFT,Xl,Xl	Xl
HARD,SOFT,Xl,Xs	Xs	HARD,SOFT,Xl,Xz	Xl
HARD,SOFT,Xl,Xc	Xc	HARD,SOFT,Xl,Xr	Xr
HARD,SOFT,Xl,Xf	Xf	HARD,SOFT,Xl,Xu	Xu
HARD,SOFT,Xs,X0	X0	HARD,SOFT,Xs,Xl	Xs
HARD,SOFT,Xs,Xs	Xs	HARD,SOFT,Xs,Xz	Xs
HARD,SOFT,Xs,Xc	Xs	HARD,SOFT,Xs,Xr	Xr
HARD,SOFT,Xs,Xf	Xs	HARD,SOFT,Xs,Xu	Xu
HARD,SOFT,Xc,X0	X0	HARD,SOFT,Xc,Xl	Xc
HARD,SOFT,Xc,Xs	Xc	HARD,SOFT,Xc,Xz	Xc
HARD,SOFT,Xc,Xc	Xc	HARD,SOFT,Xc,Xr	Xc
HARD,SOFT,Xc,Xf	Xf	HARD,SOFT,Xc,Xu	Xu
HARD,SOFT,Xr,X0	X0	HARD,SOFT,Xr,Xl	Xr
HARD,SOFT,Xr,Xs	Xr	HARD,SOFT,Xr,Xz	Xr
HARD,SOFT,Xr,Xc	Xc	HARD,SOFT,Xr,Xr	Xr
HARD,SOFT,Xr,Xf	Xc	HARD,SOFT,Xr,Xu	Xu;
HARD,SOFT,Xf,X0	X0	HARD,SOFT,Xf,Xl	Xf
HARD,SOFT,Xf,Xs	Xf	HARD,SOFT,Xf,Xz	Xf
HARD,SOFT,Xf,Xc	Xc	HARD,SOFT,Xf,Xr	Xc
HARD,SOFT,Xf,Xf	Xf	HARD,SOFT,Xf,Xu	Xu;

HARD,SOFT,Xz,X0	X0	HARD,SOFT,Xz,X1	X1
HARD,SOFT,Xz,Xs	Xs	HARD,SOFT,Xz,Xz	Xz
HARD,SOFT,Xz,Xc	Xc	HARD,SOFT,Xz,Xr	Xr
HARD,SOFT,Xz,Xf	Xf	HARD,SOFT,Xz,Xu	Xu;
HARD,SOFT,Xu,X0	X0	HARD,SOFT,Xu,X1	Xu
HARD,SOFT,Xu,Xs	Xu	HARD,SOFT,Xu,Xz	Xu
HARD,SOFT,Xu,Xc	Xu	HARD,SOFT,Xu,Xr	Xu
HARD,SOFT,Xu,Xf	Xu	HARD,SOFT,Xu,Xu	Xu

HARD,UNDRIVEN,X0,X0	X0	HARD,UNDRIVEN,X0,X1	X0
HARD,UNDRIVEN,X0,Xs	X0	HARD,UNDRIVEN,X0,Xz	X0
HARD,UNDRIVEN,X0,Xc	X0	HARD,UNDRIVEN,X0,Xr	X0
HARD,UNDRIVEN,X0,Xf	X0	HARD,UNDRIVEN,X0,Xu	X0
HARD,UNDRIVEN,X1,X0	X0	HARD,UNDRIVEN,X1,X1	X1
HARD,UNDRIVEN,X1,Xs	Xs	HARD,UNDRIVEN,X1,Xz	X1
HARD,UNDRIVEN,X1,Xc	Xc	HARD,UNDRIVEN,X1,Xr	Xs
HARD,UNDRIVEN,X1,Xf	Xf	HARD,UNDRIVEN,X1,Xu	Xu
HARD,UNDRIVEN,Xs,X0	Xs	HARD,UNDRIVEN,Xs,X1	Xs
HARD,UNDRIVEN,Xs,Xs	Xs	HARD,UNDRIVEN,Xs,Xz	Xs
HARD,UNDRIVEN,Xs,Xc	Xs	HARD,UNDRIVEN,Xs,Xr	Xs
HARD,UNDRIVEN,Xs,Xf	Xs	HARD,UNDRIVEN,Xs,Xu	Xs
HARD,UNDRIVEN,Xc,X0	Xc	HARD,UNDRIVEN,Xc,X1	Xc
HARD,UNDRIVEN,Xc,Xs	Xc	HARD,UNDRIVEN,Xc,Xz	Xc
HARD,UNDRIVEN,Xc,Xc	Xc	HARD,UNDRIVEN,Xc,Xr	Xc
HARD,UNDRIVEN,Xc,Xf	Xc	HARD,UNDRIVEN,Xc,Xu	Xc
HARD,UNDRIVEN,Xr,X0	X0	HARD,UNDRIVEN,Xr,X1	Xr
HARD,UNDRIVEN,Xr,Xs	Xr	HARD,UNDRIVEN,Xr,Xz	Xr
HARD,UNDRIVEN,Xr,Xc	Xr	HARD,UNDRIVEN,Xr,Xr	Xr
HARD,UNDRIVEN,Xr,Xf	Xr	HARD,UNDRIVEN,Xr,Xu	Xr;
HARD,UNDRIVEN,Xf,X0	Xf	HARD,UNDRIVEN,Xf,X1	Xf
HARD,UNDRIVEN,Xf,Xs	Xf	HARD,UNDRIVEN,Xf,Xz	Xf
HARD,UNDRIVEN,Xf,Xc	Xf	HARD,UNDRIVEN,Xf,Xr	Xf
HARD,UNDRIVEN,Xf,Xf	Xf	HARD,UNDRIVEN,Xf,Xu	Xf;
HARD,UNDRIVEN,Xz,X0	X0	HARD,UNDRIVEN,Xz,X1	X1
HARD,UNDRIVEN,Xz,Xs	Xs	HARD,UNDRIVEN,Xz,Xz	Xz
HARD,UNDRIVEN,Xz,Xc	Xc	HARD,UNDRIVEN,Xz,Xr	Xs
HARD,UNDRIVEN,Xz,Xf	Xf	HARD,UNDRIVEN,Xz,Xu	Xu;
HARD,UNDRIVEN,Xu,X0	Xu	HARD,UNDRIVEN,Xu,X1	Xu
HARD,UNDRIVEN,Xu,Xs	Xu	HARD,UNDRIVEN,Xu,Xz	Xu
HARD,UNDRIVEN,Xu,Xc	Xu	HARD,UNDRIVEN,Xu,Xr	Xu
HARD,UNDRIVEN,Xu,Xf	Xu	HARD,UNDRIVEN,Xu,Xu	Xu

The DOT AND function for strength 1 SOFT, and strength 2 HARD is obtained by transposing the values of the strength 2 SOFT, and strength 1 HARD table.

The DOT AND function for strength 1 SOFT, and strength 2

SOFT is identical to the strength 1 HARD, and strength 2 HARD table.

The DOT AND function for strength 1 SOFT, and strength 2 UNDRIVEN is identical to the strength 1 HARD, and strength 2 UNDRIVEN table.

The DOT AND function for strength 1 UNDRIVEN, and strength 2 HARD is is obtained by transposing the values of the strength 1 HARD, and strength 2 UNDRIVEN table.

The DOT AND function for strength 1 UNDRIVEN, and strength 2 SOFT is is obtained by transposing the values of the strength 1 SOFT, and strength 2 UNDRIVEN table.

UNDRIVEN,UNDRIVEN,X0,X0	Xs
UNDRIVEN,UNDRIVEN,X0,X1	X0
UNDRIVEN,UNDRIVEN,X0,Xs	Xs
UNDRIVEN,UNDRIVEN,X0,Xz	X0
UNDRIVEN,UNDRIVEN,X0,Xc	Xs
UNDRIVEN,UNDRIVEN,X0,Xr	X0
UNDRIVEN,UNDRIVEN,X0,Xf	Xs
UNDRIVEN,UNDRIVEN,X0,Xu	X0
UNDRIVEN,UNDRIVEN,X1,X0	X0
UNDRIVEN,UNDRIVEN,X1,X1	X1
UNDRIVEN,UNDRIVEN,X1,Xs	Xs
UNDRIVEN,UNDRIVEN,X1,Xz	X1
UNDRIVEN,UNDRIVEN,X1,Xc	Xs
UNDRIVEN,UNDRIVEN,X1,Xr	Xs
UNDRIVEN,UNDRIVEN,X1,Xf	Xs
UNDRIVEN,UNDRIVEN,X1,Xu	Xu
UNDRIVEN,UNDRIVEN,Xs,X0	X0
UNDRIVEN,UNDRIVEN,Xs,X1	Xs
UNDRIVEN,UNDRIVEN,Xs,Xs	Xs
UNDRIVEN,UNDRIVEN,Xs,Xz	Xs
UNDRIVEN,UNDRIVEN,Xs,Xc	Xs
UNDRIVEN,UNDRIVEN,Xs,Xr	Xs
UNDRIVEN,UNDRIVEN,Xs,Xf	Xs
UNDRIVEN,UNDRIVEN,Xs,Xu	Xu
UNDRIVEN,UNDRIVEN,Xc,X0	X0
UNDRIVEN,UNDRIVEN,Xc,X1	Xs
UNDRIVEN,UNDRIVEN,Xc,Xs	Xs
UNDRIVEN,UNDRIVEN,Xc,Xz	Xs
UNDRIVEN,UNDRIVEN,Xc,Xc	Xs
UNDRIVEN,UNDRIVEN,Xc,Xr	Xs
UNDRIVEN,UNDRIVEN,Xc,Xf	Xs
UNDRIVEN,UNDRIVEN,Xc,Xu	Xu
UNDRIVEN,UNDRIVEN,Xr,X0	X0
UNDRIVEN,UNDRIVEN,Xr,X1	Xs
UNDRIVEN,UNDRIVEN,Xr,Xs	Xs
UNDRIVEN,UNDRIVEN,Xr,Xz	Xs
UNDRIVEN,UNDRIVEN,Xr,Xc	Xs

UNDRIVEN,UNDRIVEN,Xr,Xr	Xs
UNDRIVEN,UNDRIVEN,Xr,Xf	Xs
UNDRIVEN,UNDRIVEN,Xr,Xu	Xu;
UNDRIVEN,UNDRIVEN,Xf,X0	Xs
UNDRIVEN,UNDRIVEN,Xf,X1	Xs
UNDRIVEN,UNDRIVEN,Xf,Xs	Xs
UNDRIVEN,UNDRIVEN,Xf,Xz	Xs
UNDRIVEN,UNDRIVEN,Xf,Xc	Xs
UNDRIVEN,UNDRIVEN,Xf,Xr	Xs
UNDRIVEN,UNDRIVEN,Xf,Xf	Xs
UNDRIVEN,UNDRIVEN,Xf,Xu	Xu;
UNDRIVEN,UNDRIVEN,Xz,X0	X0
UNDRIVEN,UNDRIVEN,Xz,X1	X1
UNDRIVEN,UNDRIVEN,Xz,Xs	Xs
UNDRIVEN,UNDRIVEN,Xz,Xz	Xz
UNDRIVEN,UNDRIVEN,Xz,Xc	Xs
UNDRIVEN,UNDRIVEN,Xz,Xr	Xs
UNDRIVEN,UNDRIVEN,Xz,Xf	Xs
UNDRIVEN,UNDRIVEN,Xz,Xu	Xu;
UNDRIVEN,UNDRIVEN,Xu,X0	Xu
UNDRIVEN,UNDRIVEN,Xu,X1	Xu
UNDRIVEN,UNDRIVEN,Xu,Xs	Xu
UNDRIVEN,UNDRIVEN,Xu,Xz	Xu
UNDRIVEN,UNDRIVEN,Xu,Xc	Xu
UNDRIVEN,UNDRIVEN,Xu,Xr	Xu
UNDRIVEN,UNDRIVEN,Xu,Xf	Xu
UNDRIVEN,UNDRIVEN,Xu,Xu	Xu

TS BUS

HARD,HARD,X0,X0	X0	HARD,HARD,X0,X1	Xu
HARD,HARD,X0,Xs	Xu	HARD,HARD,X0,Xz	X0
HARD,HARD,X0,Xc	Xc	HARD,HARD,X0,Xr	Xc
HARD,HARD,X0,Xf	Xf	HARD,HARD,X0,Xu	Xu
HARD,HARD,X1,X0	Xu	HARD,HARD,X1,X1	X1
HARD,HARD,X1,Xs	Xu	HARD,HARD,X1,Xz	X1
HARD,HARD,X1,Xc	Xc	HARD,HARD,X1,Xr	Xr
HARD,HARD,X1,Xf	Xc	HARD,HARD,X1,Xu	Xu
HARD,HARD,Xs,X0	Xu	HARD,HARD,Xs,X1	Xu
HARD,HARD,Xs,Xs	Xs	HARD,HARD,Xs,Xz	Xs
HARD,HARD,Xs,Xc	Xc	HARD,HARD,Xs,Xr	Xc
HARD,HARD,Xs,Xf	Xc	HARD,HARD,Xs,Xu	Xu
HARD,HARD,Xc,X0	Xc	HARD,HARD,Xc,X1	Xc
HARD,HARD,Xc,Xs	Xc	HARD,HARD,Xc,Xz	Xc
HARD,HARD,Xc,Xc	Xc	HARD,HARD,Xc,Xr	Xc
HARD,HARD,Xc,Xf	Xc	HARD,HARD,Xc,Xu	Xu
HARD,HARD,Xr,X0	Xc	HARD,HARD,Xr,X1	Xr
HARD,HARD,Xr,Xs	Xc	HARD,HARD,Xr,Xz	Xr
HARD,HARD,Xr,Xc	Xc	HARD,HARD,Xr,Xr	Xr
HARD,HARD,Xr,Xf	Xc	HARD,HARD,Xr,Xu	Xu;
HARD,HARD,Xf,X0	Xf	HARD,HARD,Xf,X1	Xc
HARD,HARD,Xf,Xs	Xc	HARD,HARD,Xf,Xz	Xf

HARD,HARD,Xf,Xc	Xc	HARD,HARD,Xf,Xr	Xc
HARD,HARD,Xf,Xf	Xf	HARD,HARD,Xf,Xu	Xu;
HARD,HARD,Xz,X0	X0	HARD,HARD,Xz,X1	X1
HARD,HARD,Xz,Xs	Xs	HARD,HARD,Xz,Xz	Xz
HARD,HARD,Xz,Xc	Xc	HARD,HARD,Xz,Xr	Xr
HARD,HARD,Xz,Xf	Xf	HARD,HARD,Xz,Xu	Xu;
HARD,HARD,Xu,X0	Xu	HARD,HARD,Xu,X1	Xu
HARD,HARD,Xu,Xs	Xu	HARD,HARD,Xu,Xz	Xu
HARD,HARD,Xu,Xc	Xu	HARD,HARD,Xu,Xr	Xu
HARD,HARD,Xu,Xf	Xu	HARD,HARD,Xu,Xu	Xu

HARD,SOFT,X0,X0	X0	HARD,SOFT,X0,X1	X0
HARD,SOFT,X0,Xs	X0	HARD,SOFT,X0,Xz	X0
HARD,SOFT,X0,Xc	X0	HARD,SOFT,X0,Xr	X0
HARD,SOFT,X0,Xf	X0	HARD,SOFT,X0,Xu	X0
HARD,SOFT,X1,X0	X1	HARD,SOFT,X1,X1	X1
HARD,SOFT,X1,Xs	X1	HARD,SOFT,X1,Xz	X1
HARD,SOFT,X1,Xc	X1	HARD,SOFT,X1,Xr	X1
HARD,SOFT,X1,Xf	X1	HARD,SOFT,X1,Xu	X1
HARD,SOFT,Xs,X0	Xs	HARD,SOFT,Xs,X1	Xs
HARD,SOFT,Xs,Xs	Xs	HARD,SOFT,Xs,Xz	Xs
HARD,SOFT,Xs,Xc	Xs	HARD,SOFT,Xs,Xr	Xs
HARD,SOFT,Xs,Xf	Xs	HARD,SOFT,Xs,Xu	Xs
HARD,SOFT,Xc,X0	Xc	HARD,SOFT,Xc,X1	Xc
HARD,SOFT,Xc,Xs	Xc	HARD,SOFT,Xc,Xz	Xc
HARD,SOFT,Xc,Xc	Xc	HARD,SOFT,Xc,Xr	Xc
HARD,SOFT,Xc,Xf	Xc	HARD,SOFT,Xc,Xu	Xc
HARD,SOFT,Xr,X0	Xr	HARD,SOFT,Xr,X1	Xr
HARD,SOFT,Xr,Xs	Xr	HARD,SOFT,Xr,Xz	Xr
HARD,SOFT,Xr,Xc	Xr	HARD,SOFT,Xr,Xr	Xr
HARD,SOFT,Xr,Xf	Xr	HARD,SOFT,Xr,Xu	Xr;
HARD,SOFT,Xf,X0	Xf	HARD,SOFT,Xf,X1	Xf
HARD,SOFT,Xf,Xs	Xf	HARD,SOFT,Xf,Xz	Xf
HARD,SOFT,Xf,Xc	Xf	HARD,SOFT,Xf,Xr	Xf
HARD,SOFT,Xf,Xf	Xf	HARD,SOFT,Xf,Xu	Xf;
HARD,SOFT,Xz,X0	X0	HARD,SOFT,Xz,X1	X1
HARD,SOFT,Xz,Xs	Xs	HARD,SOFT,Xz,Xz	Xz
HARD,SOFT,Xz,Xc	Xc	HARD,SOFT,Xz,Xr	Xr
HARD,SOFT,Xz,Xf	Xf	HARD,SOFT,Xz,Xu	Xu;
HARD,SOFT,Xu,X0	Xu	HARD,SOFT,Xu,X1	Xu
HARD,SOFT,Xu,Xs	Xu	HARD,SOFT,Xu,Xz	Xu
HARD,SOFT,Xu,Xc	Xu	HARD,SOFT,Xu,Xr	Xu
HARD,SOFT,Xu,Xf	Xu	HARD,SOFT,Xu,Xu	Xu

The TS BUS function for strength 1 HARD, and strength 2 UNDRIVEN is identical to the strength 1 HARD, and strength 2 SOFT table.

The TS BUS function for strength 1 SOFT, and strength 2

HARD is obtained by transposing the values of the strength 1 HARD, and strength 2 SOFT table.

The TS BUS function for strength 1 SOFT, and strength 2 SOFT is identical to the strength 1 HARD, and strength 2 HARD table.

The TS BUS function for strength 1 SOFT, and strength 2 UNDRIVEN is identical to the strength 1 HARD, and strength 2 UNDRIVEN table.

The TS BUS function for strength 1 UNDRIVEN, and strength 2 HARD is is obtained by transposing the values of the strength 1 HARD, and strength 2 UNDRIVEN table.

The TS BUS function for strength 1 UNDRIVEN, and strength 2 SOFT is is obtained by transposing the values of the strength 1 SOFT, and strength 2 UNDRIVEN table.

UNDRIVEN,UNDRIVEN,X0,X0	X0	UNDRIVEN,UNDRIVEN,X0,X1	Xs
UNDRIVEN,UNDRIVEN,X0,Xs	Xs	UNDRIVEN,UNDRIVEN,X0,Xz	X0
UNDRIVEN,UNDRIVEN,X0,Xc	Xs	UNDRIVEN,UNDRIVEN,X0,Xr	Xs
UNDRIVEN,UNDRIVEN,X0,Xf	Xs	UNDRIVEN,UNDRIVEN,X0,Xu	Xs
UNDRIVEN,UNDRIVEN,X1,X0	Xs	UNDRIVEN,UNDRIVEN,X1,X1	X1
UNDRIVEN,UNDRIVEN,X1,Xs	Xs	UNDRIVEN,UNDRIVEN,X1,Xz	X1
UNDRIVEN,UNDRIVEN,X1,Xc	Xs	UNDRIVEN,UNDRIVEN,X1,Xr	Xs
UNDRIVEN,UNDRIVEN,X1,Xf	Xs	UNDRIVEN,UNDRIVEN,X1,Xu	Xu
UNDRIVEN,UNDRIVEN,Xs,X0	Xs	UNDRIVEN,UNDRIVEN,Xs,X1	Xs
UNDRIVEN,UNDRIVEN,Xs,Xs	Xs	UNDRIVEN,UNDRIVEN,Xs,Xz	Xs
UNDRIVEN,UNDRIVEN,Xs,Xc	Xs	UNDRIVEN,UNDRIVEN,Xs,Xr	Xs
UNDRIVEN,UNDRIVEN,Xs,Xf	Xs	UNDRIVEN,UNDRIVEN,Xs,Xu	Xu
UNDRIVEN,UNDRIVEN,Xc,X0	Xs	UNDRIVEN,UNDRIVEN,Xc,X1	Xs
UNDRIVEN,UNDRIVEN,Xc,Xs	Xs	UNDRIVEN,UNDRIVEN,Xc,Xz	Xc
UNDRIVEN,UNDRIVEN,Xc,Xc	Xs	UNDRIVEN,UNDRIVEN,Xc,Xr	Xs
UNDRIVEN,UNDRIVEN,Xc,Xf	Xs	UNDRIVEN,UNDRIVEN,Xc,Xu	Xu
UNDRIVEN,UNDRIVEN,Xr,X0	Xs	UNDRIVEN,UNDRIVEN,Xr,X1	Xs
UNDRIVEN,UNDRIVEN,Xr,Xs	Xs	UNDRIVEN,UNDRIVEN,Xr,Xz	Xr
UNDRIVEN,UNDRIVEN,Xr,Xc	Xs	UNDRIVEN,UNDRIVEN,Xr,Xr	Xr
UNDRIVEN,UNDRIVEN,Xr,Xf	Xs	UNDRIVEN,UNDRIVEN,Xr,Xu	Xu
UNDRIVEN,UNDRIVEN,Xf,X0	Xs	UNDRIVEN,UNDRIVEN,Xf,X1	Xs
UNDRIVEN,UNDRIVEN,Xf,Xs	Xs	UNDRIVEN,UNDRIVEN,Xf,Xz	Xf
UNDRIVEN,UNDRIVEN,Xf,Xc	Xs	UNDRIVEN,UNDRIVEN,Xf,Xr	Xs
UNDRIVEN,UNDRIVEN,Xf,Xf	Xf	UNDRIVEN,UNDRIVEN,Xf,Xu	Xu
UNDRIVEN,UNDRIVEN,Xz,X0	X0	UNDRIVEN,UNDRIVEN,Xz,X1	X1
UNDRIVEN,UNDRIVEN,Xz,Xs	Xs	UNDRIVEN,UNDRIVEN,Xz,Xz	Xz
UNDRIVEN,UNDRIVEN,Xz,Xc	Xc	UNDRIVEN,UNDRIVEN,Xz,Xr	Xr
UNDRIVEN,UNDRIVEN,Xz,Xf	Xf	UNDRIVEN,UNDRIVEN,Xz,Xu	Xu
UNDRIVEN,UNDRIVEN,Xu,X0	Xu	UNDRIVEN,UNDRIVEN,Xu,X1	Xu
UNDRIVEN,UNDRIVEN,Xu,Xs	Xu	UNDRIVEN,UNDRIVEN,Xu,Xz	Xu
UNDRIVEN,UNDRIVEN,Xu,Xc	Xu	UNDRIVEN,UNDRIVEN,Xu,Xr	Xu
UNDRIVEN,UNDRIVEN,Xu,Xf	Xu	UNDRIVEN,UNDRIVEN,Xu,Xu	Xu

TS OR BUS

HARD,HARD,X0,X0	X0	HARD,HARD,X0,X1	X1
HARD,HARD,X0,Xs	Xs	HARD,HARD,X0,Xz	X0
HARD,HARD,X0,Xc	Xc	HARD,HARD,X0,Xr	Xr
HARD,HARD,X0,Xf	Xf	HARD,HARD,X0,Xu	Xu
HARD,HARD,X1,X0	Xs	HARD,HARD,X1,X1	X1
HARD,HARD,X1,Xs	Xs	HARD,HARD,X1,Xz	X1
HARD,HARD,X1,Xc	Xc	HARD,HARD,X1,Xr	X1
HARD,HARD,X1,Xf	Xc	HARD,HARD,X1,Xu	X1
HARD,HARD,Xs,X0	Xs	HARD,HARD,Xs,X1	Xs
HARD,HARD,Xs,Xs	Xs	HARD,HARD,Xs,Xz	Xs
HARD,HARD,Xs,Xc	Xc	HARD,HARD,Xs,Xr	Xc
HARD,HARD,Xs,Xf	Xc	HARD,HARD,Xs,Xu	Xu
HARD,HARD,Xc,X0	Xc	HARD,HARD,Xc,X1	Xc
HARD,HARD,Xc,Xs	Xc	HARD,HARD,Xc,Xz	Xc
HARD,HARD,Xc,Xc	Xc	HARD,HARD,Xc,Xr	Xc
HARD,HARD,Xc,Xf	Xc	HARD,HARD,Xc,Xu	Xu
HARD,HARD,Xr,X0	Xc	HARD,HARD,Xr,X1	Xc
HARD,HARD,Xr,Xs	Xc	HARD,HARD,Xr,Xz	Xr
HARD,HARD,Xr,Xc	Xc	HARD,HARD,Xr,Xr	Xr
HARD,HARD,Xr,Xf	Xc	HARD,HARD,Xr,Xu	Xu
HARD,HARD,Xf,X0	Xc	HARD,HARD,Xf,X1	Xc
HARD,HARD,Xf,Xs	Xc	HARD,HARD,Xf,Xz	Xf
HARD,HARD,Xf,Xc	Xc	HARD,HARD,Xf,Xr	Xc
HARD,HARD,Xf,Xf	Xf	HARD,HARD,Xf,Xu	Xu
HARD,HARD,Xz,X0	X0	HARD,HARD,Xz,X1	X1
HARD,HARD,Xz,Xs	Xs	HARD,HARD,Xz,Xz	Xz
HARD,HARD,Xz,Xc	Xc	HARD,HARD,Xz,Xr	Xr
HARD,HARD,Xz,Xf	Xf	HARD,HARD,Xz,Xu	Xu
HARD,HARD,Xu,X0	Xu	HARD,HARD,Xu,X1	Xu
HARD,HARD,Xu,Xs	Xu	HARD,HARD,Xu,Xz	Xu
HARD,HARD,Xu,Xc	Xu	HARD,HARD,Xu,Xr	Xu
HARD,HARD,Xu,Xf	Xu	HARD,HARD,Xu,Xu	Xu

HARD,SOFT,X0,X0	X0	HARD,SOFT,X0,X1	X0
HARD,SOFT,X0,Xs	X0	HARD,SOFT,X0,Xz	X0
HARD,SOFT,X0,Xc	X0	HARD,SOFT,X0,Xr	X0
HARD,SOFT,X0,Xf	X0	HARD,SOFT,X0,Xu	X0
HARD,SOFT,X1,X0	X1	HARD,SOFT,X1,X1	X1
HARD,SOFT,X1,Xs	X1	HARD,SOFT,X1,Xz	X1
HARD,SOFT,X1,Xc	X1	HARD,SOFT,X1,Xr	X1
HARD,SOFT,X1,Xf	X1	HARD,SOFT,X1,Xu	X1
HARD,SOFT,Xs,X0	Xs	HARD,SOFT,Xs,X1	Xs
HARD,SOFT,Xs,Xs	Xs	HARD,SOFT,Xs,Xz	Xs
HARD,SOFT,Xs,Xc	Xs	HARD,SOFT,Xs,Xr	Xs
HARD,SOFT,Xs,Xf	Xs	HARD,SOFT,Xs,Xu	Xs
HARD,SOFT,Xc,X0	Xc	HARD,SOFT,Xc,X1	Xc
HARD,SOFT,Xc,Xs	Xc	HARD,SOFT,Xc,Xz	Xc
HARD,SOFT,Xc,Xc	Xc	HARD,SOFT,Xc,Xr	Xc

HARD,SOFT,Xc,Xf	Xc	HARD,SOFT,Xc,Xu	Xc
HARD,SOFT,Xr,X0	Xr	HARD,SOFT,Xr,X1	Xr
HARD,SOFT,Xr,Xs	Xr	HARD,SOFT,Xr,Xz	Xr
HARD,SOFT,Xr,Xc	Xr	HARD,SOFT,Xr,Xr	Xr
HARD,SOFT,Xr,Xf	Xr	HARD,SOFT,Xr,Xu	Xr
HARD,SOFT,Xf,X0	Xf	HARD,SOFT,Xf,X1	Xf
HARD,SOFT,Xf,Xs	Xf	HARD,SOFT,Xf,Xz	Xf
HARD,SOFT,Xf,Xc	Xf	HARD,SOFT,Xf,Xr	Xf
HARD,SOFT,Xf,Xf	Xf	HARD,SOFT,Xf,Xu	Xf
HARD,SOFT,Xz,X0	X0	HARD,SOFT,Xz,X1	X1
HARD,SOFT,Xz,Xs	Xs	HARD,SOFT,Xz,Xz	Xz
HARD,SOFT,Xz,Xc	Xc	HARD,SOFT,Xz,Xr	Xr
HARD,SOFT,Xz,Xf	Xf	HARD,SOFT,Xz,Xu	Xu
HARD,SOFT,Xu,X0	Xu	HARD,SOFT,Xu,X1	X1
HARD,SOFT,Xu,Xs	Xu	HARD,SOFT,Xu,Xz	Xu
HARD,SOFT,Xu,Xc	Xu	HARD,SOFT,Xu,Xr	Xu
HARD,SOFT,Xu,Xf	Xu	HARD,SOFT,Xu,Xu	Xu

The TS OR BUS function for strength 1 HARD, and strength 2 UNDRIVEN is identical to the strength 1 HARD, and strength 2 SOFT table.

The TS OR BUS function for strength 1 SOFT, and strength 2 HARD is obtained by transposing the values of the strength 1 HARD, and strength 2 SOFT table.

The TS OR BUS function for strength 1 SOFT, and strength 2 SOFT is identical to the strength 1 HARD, and strength 2 HARD table.

The TS OR BUS function for strength 1 SOFT, and strength 2 UNDRIVEN is identical to the strength 1 HARD, and strength 2 UNDRIVEN table.

The TS OR BUS function for strength 1 UNDRIVEN, and strength 2 HARD is is obtained by transposing the values of the strength 1 HARD, and strength 2 UNDRIVEN table.

The TS OR BUS function for strength 1 UNDRIVEN, and strength 2 SOFT is is obtained by transposing the values of the strength 1 SOFT, and strength 2 UNDRIVEN table.

UNDRIVEN,UNDRIVEN,X0,X0	X0
UNDRIVEN,UNDRIVEN,X0,X1	Xs
UNDRIVEN,UNDRIVEN,X0,Xs	Xs
UNDRIVEN,UNDRIVEN,X0,Xz	X0
UNDRIVEN,UNDRIVEN,X0,Xc	Xs
UNDRIVEN,UNDRIVEN,X0,Xr	Xs
UNDRIVEN,UNDRIVEN,X0,Xf	Xs
UNDRIVEN,UNDRIVEN,X0,Xu	Xu
UNDRIVEN,UNDRIVEN,X1,X0	Xs

UNDRIVEN,UNDRIVEN,X1,X1	X1
UNDRIVEN,UNDRIVEN,X1,Xs	Xs
UNDRIVEN,UNDRIVEN,X1,Xz	X1
UNDRIVEN,UNDRIVEN,X1,Xc	Xs
UNDRIVEN,UNDRIVEN,X1,Xr	Xs
UNDRIVEN,UNDRIVEN,X1,Xf	Xs
UNDRIVEN,UNDRIVEN,X1,Xu	Xu
UNDRIVEN,UNDRIVEN,Xs,X0	Xs
UNDRIVEN,UNDRIVEN,Xs,X1	Xs
UNDRIVEN,UNDRIVEN,Xs,Xs	Xs
UNDRIVEN,UNDRIVEN,Xs,Xz	Xs
UNDRIVEN,UNDRIVEN,Xs,Xc	Xs
UNDRIVEN,UNDRIVEN,Xs,Xr	Xs
UNDRIVEN,UNDRIVEN,Xs,Xf	Xs
UNDRIVEN,UNDRIVEN,Xs,Xu	Xu
UNDRIVEN,UNDRIVEN,Xc,X0	Xs
UNDRIVEN,UNDRIVEN,Xc,X1	Xs
UNDRIVEN,UNDRIVEN,Xc,Xs	Xs
UNDRIVEN,UNDRIVEN,Xc,Xz	Xs
UNDRIVEN,UNDRIVEN,Xc,Xc	Xs
UNDRIVEN,UNDRIVEN,Xc,Xr	Xs
UNDRIVEN,UNDRIVEN,Xc,Xf	Xs
UNDRIVEN,UNDRIVEN,Xc,Xu	Xu
UNDRIVEN,UNDRIVEN,Xr,X0	Xs
UNDRIVEN,UNDRIVEN,Xr,X1	Xs
UNDRIVEN,UNDRIVEN,Xr,Xs	Xs
UNDRIVEN,UNDRIVEN,Xr,Xz	Xs
UNDRIVEN,UNDRIVEN,Xr,Xc	Xs
UNDRIVEN,UNDRIVEN,Xr,Xr	Xs
UNDRIVEN,UNDRIVEN,Xr,Xf	Xs
UNDRIVEN,UNDRIVEN,Xr,Xu	Xu
UNDRIVEN,UNDRIVEN,Xf,X0	Xs
UNDRIVEN,UNDRIVEN,Xf,X1	Xs
UNDRIVEN,UNDRIVEN,Xf,Xs	Xs
UNDRIVEN,UNDRIVEN,Xf,Xz	Xs
UNDRIVEN,UNDRIVEN,Xf,Xc	Xs
UNDRIVEN,UNDRIVEN,Xf,Xr	Xs
UNDRIVEN,UNDRIVEN,Xf,Xf	Xs
UNDRIVEN,UNDRIVEN,Xf,Xu	Xu
UNDRIVEN,UNDRIVEN,Xz,X0	X0
UNDRIVEN,UNDRIVEN,Xz,X1	X1

UNDRIVEN,UNDRIVEN,Xz,Xs	Xs
UNDRIVEN,UNDRIVEN,Xz,Xz	Xz
UNDRIVEN,UNDRIVEN,Xz,Xc	Xs
UNDRIVEN,UNDRIVEN,Xz,Xr	Xs
UNDRIVEN,UNDRIVEN,Xz,Xf	Xs
UNDRIVEN,UNDRIVEN,Xz,Xu	Xu
UNDRIVEN,UNDRIVEN,Xu,X0	Xu
UNDRIVEN,UNDRIVEN,Xu,X1	Xu
UNDRIVEN,UNDRIVEN,Xu,Xs	Xu
UNDRIVEN,UNDRIVEN,Xu,Xz	Xu
UNDRIVEN,UNDRIVEN,Xu,Xc	Xu
UNDRIVEN,UNDRIVEN,Xu,Xr	Xu
UNDRIVEN,UNDRIVEN,Xu,Xf	Xu
UNDRIVEN,UNDRIVEN,Xu,Xu	Xu

Timing Verifier Output Format

6.10 INTRODUCTION

The Timing Verifier provides a single output listing which contains several kinds of information:

1. Errors detected during the reading of the input files. These errors are syntax errors and indicate that one of the inputs to the Timing Verifier is not properly formed.

2. Errors detected while the Timing Verifier is running. These errors fall into two classes: improper inputs and internal Timing Verifier errors.

   The errors reported by the Timing Verifier will guide the user to correct his design. Internal errors are clearly distinguished and should be reported to VALID immediately.

3. A list of the value history of every output signal in the design.

4. A list of timing violations.

The output listing is described in detail using the annotated Timing Verifier output on the following pages.

6.11 SAMPLE RUN OF THE TIMING VERIFIER

First the Timing Verifier is run with the following directives (VERIFIER.CMD) file:

```
CLOCK_PERIOD  200.0;
CLOCK_INTERVALS  10;
CLOCK_SKEW  0.0;
PREC_CLOCK_SKEW  0.0;
WIRE_DELAY  0.0;
BIT_ORDERING RIGHT_TO_LEFT;
end.
```

This produces the following output listing (line numbers have been added for reference - they do not normally appear in the output file):

```
 1)  Reading Compiler Expansion File ...
 2) Primitive = TSBUF
 3)     42)
 4)              SIZE='8';
 5)  ERROR #     1  undefined primitive type
 6)  ERROR #     2  Unknown primitive in wire list ignored
 7)         2  errors detected.
 8) (00:00:16.74)
 9)
10)  Reading directives file ...
11)    863)  BIT_ORDERING RIGHT_TO_LEFT;
12)
13)  ERROR #     3  Unknown option specified
14)      One error detected.
15) (00:00:00.40)
16)
17)  Verifier Directives:
18)      CLOCK_PERIOD 200.0;
19)      CLOCK_INTERVALS 10 (20.0ns);
20)      CLOCK_SKEW 0.0;
21)      PRECISION_CLOCK_SKEW 0.0;
22)      MINIMUM_WIRE_DELAY 0.0;
23)      MAXIMUM_WIRE_DELAY 0.0;
24)      RISE_FALL_ANALYSIS ON;
25)      DOT_TYPE DOT_AND;
26)      MAX_ERRORS 10
27)      LIST OFF;
28)
29)  Reading bus delay file ...
30)      No errors detected.
31) (00:00:00.24)
32)
33)  Initializing signals ...
34)      No errors detected.
35) (00:00:01.32)
36)
37)  Reading Case Specification ...
38)      1) END.
39)      No errors detected.
40)  Doing timing analysis ...
41)  Circuit Evaluation completed
42)  Total number of evaluation passes:     10
```

```
 43)  Total number of events processed:      83
 44) (00:00:02.77)
 45)
 46)  Set-up, Hold and Minimum Pulse Width Errors ....
 47)
 48)  Doing error analysis ...
 49) (00:00:00.30)
 50)
 51)  Values of all signals
 52)
 53)   NC     .    .    .    .    .    .    .   S:0.0, C:46.0, S:80.0
 54)  (SHF .00.15P)UNNAMED$1$2 AND$2P$T     .   S:0.0, C:46.0, S:80.0
 55)  (SHF .00.16P)UNNAMED$1$2 AND$2P$T     .   S:0.0, C:50.0, S:95.0
 56)  (SHF .00.2P)UNNAMED$1$2 AND$2P$T      .   S:0.0
 57)  (SHF .157.10P) NC       .    .    .    .   S:0.0
 58)  (SHF .157.10P)UNNAMED$1$2 AND$1P$I0   .   S:0.0
 59)  (SHF .157.10P)UNNAMED$1$2 AND$1P$I1   .   1:0.0
 60)  (SHF .157.10P)UNNAMED$1$2 MUX$4P$S    .   S:0.0
 61)  (SHF .157.11P) NC       .    .    .    .   S:0.0
 62)  (SHF .157.11P)UNNAMED$1$2 AND$1P$I0   .   S:0.0
 63)  (SHF .157.11P)UNNAMED$1$2 AND$1P$I1   .   1:0.0
 64)  (SHF .157.11P)UNNAMED$1$2 MUX$4P$S    .   S:0.0
 65)  (SHF .157.12P) NC       .    .    .    .   S:0.0
 66)  (SHF .157.12P)UNNAMED$1$2 AND$1P$I0   .   S:0.0
 67)  (SHF .157.12P)UNNAMED$1$2 AND$1P$I1   .   1:0.0
 68)  (SHF .157.12P)UNNAMED$1$2 MUX$4P$S    .   S:0.0
 69)  (SHF .157.14P) NC       .    .    .    .   S:0.0
 70)  (SHF .157.14P)UNNAMED$1$2 AND$1P$I0   .   S:0.0, C:46.0, S:80.0
 71)  (SHF .157.14P)UNNAMED$1$2 AND$1P$I1   .   1:0.0
 72)  (SHF .157.14P)UNNAMED$1$2 MUX$4P$S    .   S:0.0
 73)  (SHF .157.5P) NC 4      .    .    .    .   S:0.0
 74)  (SHF .157.5P)UNNAMED$1$2 AND$1P$I0    .   S:0.0
 75)  (SHF .157.5P)UNNAMED$1$2 AND$1P$I1    .   1:0.0
 76)  (SHF .157.5P)UNNAMED$1$2 MUX$4P$S     .   S:0.0
 77)  (SHF .157.6P) NC 3      .    .    .    .   S:0.0
 78)  (SHF .157.6P)UNNAMED$1$2 AND$1P$I0    .   S:0.0
 79)  (SHF .157.6P)UNNAMED$1$2 AND$1P$I1    .   1:0.0
 80)  (SHF .157.6P)UNNAMED$1$2 MUX$4P$S     .   S:0.0
 81)  (SHF .157.7P) NC 2      .    .    .    .   S:0.0
 82)  (SHF .157.7P)UNNAMED$1$2 AND$1P$I0    .   S:0.0
 83)  (SHF .157.7P)UNNAMED$1$2 AND$1P$I1    .   1:0.0
 84)  (SHF .157.7P)UNNAMED$1$2 MUX$4P$S     .   S:0.0
 85)  (SHF .157.8P) NC 1      .    .    .    .   S:0.0
 86)  (SHF .157.8P)UNNAMED$1$2 AND$1P$I0    .   S:0.0, C:54.0, S:110.0
 87)  (SHF .157.8P)UNNAMED$1$2 AND$1P$I1    .   1:0.0
 88)  (SHF .157.8P)UNNAMED$1$2 MUX$4P$S     .   S:0.0
 89)  (SHF .157.9P) NC 0      .    .    .    .   S:0.0
 90)  (SHF .157.9P)UNNAMED$1$2 AND$1P$I0    .   S:0.0
 91)  (SHF .157.9P)UNNAMED$1$2 AND$1P$I1    .   1:0.0
 92)  (SHF .157.9P)UNNAMED$1$2 MUX$4P$S     .   S:0.0
 93)  (SHF .374.4P)UNNAMED$1$REG$5P$T<7..0> .   S:0.0, C:41.0, S:48.0
 94)  (SHF .74.13P)UNNAMED$1$BUF$1P$I       .   S:0.0, C:40.0, S:40.0
 95)  (SHF .74.13P)UNNAMED$1$IBUF$11P$T     .   0:0.0
 96)  (SHF .74.13P)UNNAMED$1$IBUF$3P$T      .   0:0.0
 97)  (SHF .374.4P)UNNAMED$1$BUF$3P$T       .   0:0.0
```

```
 98)  UNNAMED$1$LS00$15P$Y  . . . .  S:0.0, C:50.0, S:95.0
 99)  UNNAMED$1$LS00$16P$A  . . . .  S:0.0
100)  0      . . . . . . .  0:0.0
101)  1      . . . . . . .  1:0.0
102)  CARRY    . . . . . .  S:0.0, C:46.0, S:80.0
103)  CLOCK !C 2-4  . . . .  0:0.0, R:40.0, 1:40.0,
                                   F:80.0, 0:80.0
104)  LSB IN     . . . . .  S:0.0, C:54.0, S:110.0
105)  SERIAL DATA IN  . . .  S:0.0
106)  SHIFT      . . . . .  S:0.0
107)  SHIFT/ROTATE  . . . .  S:0.0
108)  UN$1$8MERGE$3P$A  . .  S:0.0
109)  UN$1$8MERGE$3P$B  . .  S:0.0
110)  UN$1$8MERGE$3P$C  . .  S:0.0
111)  UN$1$8MERGE$3P$D  . .  S:0.0
112)  UN$1$8MERGE$3P$E  . .  S:0.0
113)  UN$1$8MERGE$3P$F  . .  S:0.0
114)  UN$1$8MERGE$3P$G  . .  S:0.0
115)  UN$1$8MERGE$3P$H  . .  S:0.0
116)  UN$1$LS00$2P$B    . .  S:0.0
117)  UN$1$LS157$10P$Y  . .  S:0.0
118)  UN$1$LS157$11P$Y  . .  S:0.0
119)  UN$1$LS157$12P$Y  . .  S:0.0
120)  UN$1$LS157$14P$Y  . .  S:0.0, C:50.0, S:94.0
121)  UN$1$LS157$5P$Y   . .  S:0.0
122)  UN$1$LS157$6P$Y   . .  S:0.0
123)  UN$1$LS157$7P$Y   . .  S:0.0
124)  UN$1$LS157$8P$Y   . .  S:0.0, C:58.0, S:124.0
125)  UN$1$LS157$9P$Y   . .  S:0.0
126)
127)  Signals not meeting their stable assertions
128)
129)  All done
130)      3 runtime errors detected.
131)  Start time   = 22:52:42.25
132)  Ending time  = 22:53:05.44
133)  Elapsed time = 00:00:23.19
```

1. Lines 1-8 -- The expansion file was read. One undefined primitive type was detected. The correct primitive name is TS BUF, not TSBUF. Reading the expansion file took 16.7 seconds of elapsed time.

2. Line 10 -- The directives file is being read.

3. Lines 11-15 -- An error was detected in the directives file. An unknown option was specified. This is corrected by replacing the directive with one that is defined in the SCALD III Timing Verifier Directives document. The directives file was read in .4 seconds.

4. Lines 17-27 -- The Timing Verifier directives are listed. These are the directives in effect for this running of the Timing Verifier.

5. Lines 29-31 -- The bus delay file for this example is empty so it is not too surprising that no errors were detected when they were read in.

6. Lines 33-35 -- Signals are initialized. During this phase errors such as multiply driven nets, and overlapping signal assertion intervals are reported.

7. Lines 37-39 -- The case file is empty for this example so no errors were detected.

8. Lines 40-44 -- A summary of the simluation phase of the timing verifier is reported. In this particular case a total of 10 simulations of the entire network were required. However, the Timing Verifier is event driven, and only simulates an object if its inputs have changed. A total of 83 objects (gates, registers etc.) were simulated -- approximately 8 per pass.

9. Lines 46-49 -- All the set-up, hold and minimum pulse width errors are listed next. In this case there were none.

10. Lines 51-125 -- All the signals in the design and their value histories are listed next. A value history is a list of the form: <time>: <value>, <time>: value ... <time>: <value>. A signal assumes the value indicated at the time indicated. The last value in the list is the value of the signal to the end of the clock period. So for example, the signal LSB IN is stable at time 0 ns, and remains stable until time 54 ns, then the signal is changing. The signal is changing from time 54 ns until time 110 ns. Then it goes stable and remains stable until 200 ns (the end of the clock period).

11. Lines 127 -- Any signals that have stable assertions and are driven are checked to see that the actual value history of the signal (as determined by the circuit) is more conservative than the assertion. That is the signal is actually stable whenever the assertion says it is (and maybe at other times too). In this example, no signals had stable assertions, so there were not violations.

12. Lines 130-132 -- The total number of runtime errors (all errors other than timing errors) detected by the Timing Verifier and the total elapsed time for the job are reported.

6.12 SECOND TRY

The drawing is corrected to use the correct part type -- TS BUF and the command file is fixed:

```
CLOCK_PERIOD  200.0;
CLOCK_INTERVALS  10;
CLOCK_SKEW  0.0;
PREC_CLOCK_SKEW  0.0;
WIRE_DELAY  0.0;
end.
```

This produces the following correct output listing:

```
 1)  Reading Compiler Expansion File ...
 2)       No errors detected.
 3) (00:00:16.36)
 4)
 5)  Reading directives file ...
 6)       No errors detected.
 7) (00:00:00.32)
 8)
 9)  Verifier Directives:
10)       CLOCK_PERIOD 200.0;
11)       CLOCK_INTERVALS 10 (20.0ns);
12)       CLOCK_SKEW 0.0;
13)       PRECISION_CLOCK_SKEW 0.0;
14)       MINIMUM_WIRE_DELAY 0.0;
15)       MAXIMUM_WIRE_DELAY 0.0;
16)       RISE_FALL_ANALYSIS ON;
17)       DOT_TYPE DOT_AND;
18)       MAX_ERRORS 10
19)       LIST OFF;
20)
21)  Reading bus delay file ...
22)       No errors detected.
23) (00:00:00.26)
24)
25)  Initializing signals ...
26)       No errors detected.
27) (00:00:01.37)
28)
29)  Reading Case Specification ...
30)       1) END.
31)       No errors detected.
32)  Doing timing analysis ...
33)  Circuit Evaluation completed
34)  Total number of evaluation passes:      11
35)  Total number of events processed:       86
36) (00:00:03.29)
37)
38)  Set-up, Hold and Minimum Pulse Width Errors ....
39)
40)  Doing error analysis ...
41) (00:00:00.30)
42)
```

```
43)  Values of all signals
44)
45)   NC      .     .     .     .     .     .     S:0.0, C:46.0, S:80.0
46)  (SHF .00.15P)UN$1$2 AND$2P$T      .     S:0.0, C:46.0, S:80.0
47)  (SHF .00.16P)UN$1$2 AND$2P$T      .     S:0.0, C:50.0, S:95.0
48)  (SHF .00.2P)UN$1$2 AND$2P$T       .     S:0.0
49)  (SHF .157.10P) NC       .     .     .     S:0.0
50)  (SHF .157.10P)UN$1$2 AND$1P$I0    .     S:0.0
51)  (SHF .157.10P)UN$1$2 AND$1P$I1    .     1:0.0
52)  (SHF .157.10P)UN$1$2 MUX$4P$S     .     S:0.0
53)  (SHF .157.11P) NC       .     .     .     S:0.0
54)  (SHF .157.11P)UN$1$2 AND$1P$I0    .     S:0.0
55)  (SHF .157.11P)UN$1$2 AND$1P$I1    .     1:0.0
56)  (SHF .157.11P)UN$1$2 MUX$4P$S     .     S:0.0
57)  (SHF .157.12P) NC       .     .     .     S:0.0
58)  (SHF .157.12P)UN$1$2 AND$1P$I0    .     S:0.0
59)  (SHF .157.12P)UN$1$2 AND$1P$I1    .     1:0.0
60)  (SHF .157.12P)UN$1$2 MUX$4P$S     .     S:0.0
61)  (SHF .157.14P) NC       .     .     .     S:0.0
62)  (SHF .157.14P)UN$1$2 AND$1P$I0    .     S:0.0, C:46.0, S:80.0
63)  (SHF .157.14P)UN$1$2 AND$1P$I1    .     1:0.0
64)  (SHF .157.14P)UN$1$2 MUX$4P$S     .     S:0.0
65)  (SHF .157.5P) NC 4      .     .     .     S:0.0
66)  (SHF .157.5P)UN$1$2 AND$1P$I0     .     S:0.0
67)  (SHF .157.5P)UN$1$2 AND$1P$I1     .     1:0.0
68)  (SHF .157.5P)UN$1$2 MUX$4P$S      .     S:0.0
69)  (SHF .157.6P) NC 3      .     .     .     S:0.0
70)  (SHF .157.6P)UN$1$2 AND$1P$I0     .     S:0.0
71)  (SHF .157.6P)UN$1$2 AND$1P$I1     .     1:0.0
72)  (SHF .157.6P)UN$1$2 MUX$4P$S      .     S:0.0
73)  (SHF .157.7P) NC 2      .     .     .     S:0.0
74)  (SHF .157.7P)UN$1$2 AND$1P$I0     .     S:0.0
75)  (SHF .157.7P)UN$1$2 AND$1P$I1     .     1:0.0
76)  (SHF .157.7P)UN$1$2 MUX$4P$S      .     S:0.0
77)  (SHF .157.8P) NC 1      .     .     .     S:0.0
78)  (SHF .157.8P)UN$1$2 AND$1P$I0     .     S:0.0, C:54.0, S:110.0
79)  (SHF .157.8P)UN$1$2 AND$1P$I1     .     1:0.0
80)  (SHF .157.8P)UN$1$2 MUX$4P$S      .     S:0.0
81)  (SHF .157.9P) NC 0      .     .     .     S:0.0
82)  (SHF .157.9P)UN$1$2 AND$1P$I0     .     S:0.0
83)  (SHF .157.9P)UN$1$2 AND$1P$I1     .     1:0.0
84)  (SHF .157.9P)UN$1$2 MUX$4P$S      .     S:0.0
85)  (SHF .374.4P)UN$1$REG$5P$T<7..0>  .     S:0.0, C:41.0, S:48.0
86)  (SHF .74.13P)UN$1$BUF$1P$I        .     S:0.0, C:40.0, S:40.0
87)  (SHF .74.13P)UN$1$IBUF$11P$T      .     0:0.0
88)  (SHF .74.13P)UN$1$IBUF$3P$T       .     0:0.0
89)  (SHF .374.4P)UN$1$BUF$3P$T        .     0:0.0
90)  UN$1$LS00$15P$Y         .     .     .     S:0.0, C:50.0, S:95.0
91)  UN$1$LS00$16P$A         .     .     .     S:0.0
92)  0       .     .     .     .     .     .     0:0.0
93)  1       .     .     .     .     .     .     1:0.0
94)  CARRY       .     .     .     .     .     S:0.0, C:46.0, S:80.0
95)  CLOCK !C 2-4      .     .     .     .     0:0.0, R:40.0, 1:40.0,
                                                      F:80.0, 0:80.0
96)  LSB IN        .     .     .           .     S:0.0, C:54.0, S:110.0
```

```
 97)  SERIAL DATA IN    .    .    .    .    S:0.0
 98)  SHIFT             .    .    .    .    S:0.0
 99)  SHIFT/ROTATE      .    .    .    .    S:0.0
100)  UN$1$8MERGE$1P$A       .    .    .    S:0.0, C:47.0, S:68.0
101)  UN$1$8MERGE$1P$B       .    .    .    S:0.0, C:47.0, S:68.0
102)  UN$1$8MERGE$1P$C       .    .    .    S:0.0, C:47.0, S:68.0
103)  UN$1$8MERGE$1P$D       .    .    .    S:0.0, C:47.0, S:68.0
104)  UN$1$8MERGE$1P$E       .    .    .    S:0.0, C:47.0, S:68.0
105)  UN$1$8MERGE$1P$F       .    .    .    S:0.0, C:47.0, S:68.0
106)  UN$1$8MERGE$1P$G       .    .    .    S:0.0, C:47.0, S:68.0
107)  UN$1$8MERGE$1P$H       .    .    .    S:0.0, C:47.0, S:68.0
108)  UN$1$8MERGE$3P$A       .    .    .    S:0.0
109)  UN$1$8MERGE$3P$B       .    .    .    S:0.0
110)  UN$1$8MERGE$3P$C       .    .    .    S:0.0
111)  UN$1$8MERGE$3P$D       .    .    .    S:0.0
112)  UN$1$8MERGE$3P$E       .    .    .    S:0.0
113)  UN$1$8MERGE$3P$F       .    .    .    S:0.0
114)  UN$1$8MERGE$3P$G       .    .    .    S:0.0
115)  UN$1$8MERGE$3P$H       .    .    .    S:0.0
116)  UN$1$LS00$2P$B         .    .    .    S:0.0
117)  UN$1$LS157$10P$Y       .    .    .    S:0.0
118)  UN$1$LS157$11P$Y       .    .    .    S:0.0
119)  UN$1$LS157$12P$Y       .    .    .    S:0.0
120)  UN$1$LS157$14P$Y       .    .    .    S:0.0, C:50.0, S:94.0
121)  UN$1$LS157$5P$Y        .    .    .    S:0.0
122)  UN$1$LS157$6P$Y        .    .    .    S:0.0
123)  UN$1$LS157$7P$Y        .    .    .    S:0.0
124)  UN$1$LS157$8P$Y        .    .    .    S:0.0, C:58.0, S:124.0
125)  UN$1$LS157$9P$Y        .    .    .    S:0.0
126)
127)  Signals not meeting their stable assertions
128)
129)  All done
130)      No runtime errors detected.
131)  Start time   = 23:43:05.84
132)  Ending time  = 23:43:29.27
133)  Elapsed time = 00:00:23.43
```

6.13 THIRD TRY

Now that we have some confidence in the design, we will try one more run. This time a clock period of 100 ns is tried by modifying the directives file. (Verifier.cmd on VAX and S-32, verifier cmd on IBM.) Also we want to feed the CARRY signal of this system to another system which requires that it be stable from 0 ns to 30 ns and from 60 ns to 90 ns in the cycle. To check that the CARRY signal is stable we add the appropriate assertion on CARRY! S0-3,6-9. Running the Timing Verifier we get the following output listing.

```
 1)  Reading Compiler Expansion File ...
 2)      No errors detected.
 3) (00:00:18.37)
 4)
 5)  Reading directives file ...
 6)      No errors detected.
 7) (00:00:00.30)
 8)
 9)  Verifier Directives:
10)      CLOCK_PERIOD 100.0;
11)      CLOCK_INTERVALS 10 (10.0ns);
12)      CLOCK_SKEW 0.0;
13)      PRECISION_CLOCK_SKEW 0.0;
14)      MINIMUM_WIRE_DELAY 0.0;
15)      MAXIMUM_WIRE_DELAY 0.0;
16)      RISE_FALL_ANALYSIS ON;
17)      DOT_TYPE DOT_AND;
18)      MAX_ERRORS 10
19)      LIST OFF;
20)
21)  Reading bus delay file ...
22)      No errors detected.
23) (00:00:00.23)
24)
25)  Initializing signals ...
26)      No errors detected.
27) (00:00:01.53)
28)
29)  Reading Case Specification ...
30)      1) END.
31)      No errors detected.
32)  Doing timing analysis ...
33)  Circuit Evaluation completed
34)  Total number of evaluation passes:     11
35)  Total number of events processed:     102
36) (00:00:04.22)
37)
38)  Set-up, Hold and Minimum Pulse Width Errors ....
39)
40)  Doing error analysis ...
41)
42)  Minimum pulse width violated by signal; Minimum HIGH = 25.0,
```

```
43)                                                Minimum LOW = 15.0
44)  current location string is "(SHF .374.4P MPW1P)"
45)  CLOCK !C 2-4                    (+0.1)       0:0.0, 1:20.0, 0:40.0
46)
47)  Setup or Hold Time Error at 20.0; Setup Time = 20.0, Hold Time = 0.0
48)  current location string is "(SHF .374.4P SUH2P)"
49)  CK INPUT    = CLOCK !C 2-4                0:0.0, R:20.0, 1:20.0, F:40.0,
                                                                      0:40.0
50)  DATA INPUT = UNNAMED$1$LS157$8P$Y         C:0.0, S:4.0, C:38.0
51) (00:00:00.40)
52)
53)  Values of all signals
54)
55)   NC      .     .     .     .     .     .    S:0.0, C:26.0, S:60.0
56)  (SHF .00.15P)UN$1$2 AND$2P$T       .     .    S:0.0, C:26.0, S:60.0
57)  (SHF .00.16P)UN$1$2 AND$2P$T       .     .    S:0.0, C:30.0, S:75.0
58)  (SHF .00.2P)UN$1$2 AND$2P$T        .     .    S:0.0
59)  (SHF .157.10P) NC          .     .     .    S:0.0
60)  (SHF .157.10P)UN$1$2 AND$1P$I0     .     .    S:0.0
61)  (SHF .157.10P)UN$1$2 AND$1P$I1     .     .    1:0.0
62)  (SHF .157.10P)UN$1$2 MUX$4P$S      .     .    S:0.0
63)  (SHF .157.11P) NC          .     .     .    S:0.0
64)  (SHF .157.11P)UN$1$2 AND$1P$I0     .     .    S:0.0
65)  (SHF .157.11P)UN$1$2 AND$1P$I1     .     .    1:0.0
66)  (SHF .157.11P)UN$1$2 MUX$4P$S      .     .    S:0.0
67)  (SHF .157.12P) NC          .     .     .    S:0.0
68)  (SHF .157.12P)UN$1$2 AND$1P$I0     .     .    S:0.0
69)  (SHF .157.12P)UN$1$2 AND$1P$I1     .     .    1:0.0
70)  (SHF .157.12P)UN$1$2 MUX$4P$S      .     .    S:0.0
71)  (SHF .157.14P) NC          .     .     .    S:0.0
72)  (SHF .157.14P)UN$1$2 AND$1P$I0     .     .    S:0.0, C:26.0, S:60.0
73)  (SHF .157.14P)UN$1$2 AND$1P$I1     .     .    1:0.0
74)  (SHF .157.14P)UN$1$2 MUX$4P$S      .     .    S:0.0
75)  (SHF .157.5P) NC 4         .     .     .    S:0.0
76)  (SHF .157.5P)UN$1$2 AND$1P$I0      .     .    S:0.0
77)  (SHF .157.5P)UN$1$2 AND$1P$I1      .     .    1:0.0
78)  (SHF .157.5P)UN$1$2 MUX$4P$S       .     .    S:0.0
79)  (SHF .157.6P) NC 3         .     .     .    S:0.0
80)  (SHF .157.6P)UN$1$2 AND$1P$I0      .     .    S:0.0
81)  (SHF .157.6P)UN$1$2 AND$1P$I1      .     .    1:0.0
82)  (SHF .157.6P)UN$1$2 MUX$4P$S       .     .    S:0.0
83)  (SHF .157.7P) NC 2         .     .     .    S:0.0
84)  (SHF .157.7P)UN$1$2 AND$1P$I0      .     .    S:0.0
85)  (SHF .157.7P)UN$1$2 AND$1P$I1      .     .    1:0.0
86)  (SHF .157.7P)UN$1$2 MUX$4P$S       .     .    S:0.0
87)  (SHF .157.8P) NC 1         .     .     .    S:0.0
88)  (SHF .157.8P)UN$1$2 AND$1P$I0      .     .    S:0.0, C:34.0, S:90.0
89)  (SHF .157.8P)UN$1$2 AND$1P$I1      .     .    1:0.0
90)  (SHF .157.8P)UN$1$2 MUX$4P$S       .     .    S:0.0
91)  (SHF .157.9P) NC 0         .     .     .    S:0.0
92)  (SHF .157.9P)UN$1$2 AND$1P$I0      .     .    S:0.0
93)  (SHF .157.9P)UN$1$2 AND$1P$I1      .     .    1:0.0
94)  (SHF .157.9P)UN$1$2 MUX$4P$S       .     .    S:0.0
95)  (SHF .374.4P)UN$1$REG$5P$T<7..0>   .     .    S:0.0, C:21.0, S:28.0
96)  (SHF .74.13P)UN$1$BUF$1P$I         .     .    S:0.0, C:20.0, S:20.0
```

```
 97)  (SHF .74.13P)UN$1$IBUF$11P$T      .    0:0.0
 98)  (SHF .74.13P)UN$1$IBUF$3P$T       .    0:0.0
 99)  *(SHF .374.4P)UN$1$BUF$3P$T       .    0:0.0
100)  *UN$1$LS00$15P$Y          .  .  .    S:0.0, C:30.0, S:75.0
101)  *UN$1$LS00$16P$A          .  .  .    S:0.0
102)  0          .  .  .  .  .  .  .  .    0:0.0
103)  1          .  .  .  .  .  .  .  .    1:0.0
104)  CARRY!S 0-3,6-9           .  .  .    S:0.0, C:26.0, S:60.0
105)  CLOCK !C 2-4         .    .  .  .    0:0.0, R:20.0, 1:20.0,
                                                F:40.0, 0:40.0
106)  LSB IN          .    .    .  .  .    S:0.0, C:34.0, S:90.0
107)  SERIAL DATA IN  .    .    .  .  .    S:0.0
108)  SHIFT           .    .    .  .  .    S:0.0
109)  SHIFT/ROTATE    .    .    .  .  .    S:0.0
110)  UN$1$8MERGE$1P$A          .  .  .    S:0.0, C:27.0, S:48.0
111)  UN$1$8MERGE$1P$B          .  .  .    S:0.0, C:27.0, S:48.0
112)  UN$1$8MERGE$1P$C          .  .  .    S:0.0, C:27.0, S:48.0
113)  UN$1$8MERGE$1P$D          .  .  .    S:0.0, C:27.0, S:48.0
114)  UN$1$8MERGE$1P$E          .  .  .    S:0.0, C:27.0, S:48.0
115)  UN$1$8MERGE$1P$F          .  .  .    S:0.0, C:27.0, S:48.0
116)  UN$1$8MERGE$1P$G          .  .  .    S:0.0, C:27.0, S:48.0
117)  UN$1$8MERGE$1P$H          .  .  .    S:0.0, C:27.0, S:48.0
118)  UN$1$8MERGE$3P$A          .  .  .    S:0.0
119)  UN$1$8MERGE$3P$B          .  .  .    S:0.0
120)  UN$1$8MERGE$3P$C          .  .  .    S:0.0
121)  UN$1$8MERGE$3P$D          .  .  .    S:0.0
122)  UN$1$8MERGE$3P$E          .  .  .    S:0.0
123)  UN$1$8MERGE$3P$F          .  .  .    S:0.0
124)  UN$1$8MERGE$3P$G          .  .  .    S:0.0
125)  UN$1$8MERGE$3P$H          .  .  .    S:0.0
126)  UN$1$LS00$2P$B       .    .  .  .    S:0.0
127)  UN$1$LS157$10P$Y          .  .  .    S:0.0
128)  UN$1$LS157$11P$Y          .  .  .    S:0.0
129)  UN$1$LS157$12P$Y          .  .  .    S:0.0
130)  UN$1$LS157$14P$Y          .  .  .    S:0.0, C:30.0, S:74.0
131)  UN$1$LS157$5P$Y           .  .  .    S:0.0
132)  UN$1$LS157$6P$Y           .  .  .    S:0.0
133)  UN$1$LS157$7P$Y           .  .  .    S:0.0
134)  UN$1$LS157$8P$Y           .  .  .    C:0.0, S:4.0, C:38.0
135)  UN$1$LS157$9P$Y           .  .  .    S:0.0
136)
137)  Signals not meeting their stable assertions
138)  CARRY!S 0-3,6-9           .  .  .    S:0.0, C:26.0, S:60.0
139)
140)  All done
141)      No runtime errors detected.
142)  Start time   = 23:41:01.40
143)  Ending time  = 23:41:28.01
144)  Elapsed time = 00:00:26.61
```

Notice that now there are a variety of timing errors.

1. Lines 42-45 -- The clock signal violates the minimum pulse width requirements of the LS374 at location 4P of drawing SHF. The minimum required high time is 25 ns. The signal however, is high from 20 to 40 ns or 20 ns (with a skew of -0.0, +0.1 ns).

2. Lines 47-51 -- The same LS374 has a setup violation at time 20 ns. At 20 ns the register is clocked, yet the data input to the register has been stable for only 16

ns prior to the rising edge - 4 ns short of the devices setup time. The actual hold time is 18 ns which is sufficient.

3. Lines 137-138 -- The CARRY signal fails to meet the interface specification. Its assertion requires that it be stable at time 30 ns, but the Timing Verifier shows that it is changing 4 ns too soon.

Timing Verifier Directives Summary

6.14 INTRODUCTION

Timing Verifier directives are used to specify certain things about a design that effect how the Timing Verifier interprets and processes the Compiler output expansion. Directives are passed to the Timing Verifier in two ways. Directives may be placed in a text file (which is bound to the logical device OPTFILE when the Timing Verifier is run). Second, the directives may be specified as properties on a TIME DIRECTIVES body in some drawing of the expansion. If both a text file and a TIME DIRECTIVES body property specify a directive value, the values in the text file will dominate.

6.15 TIMING VERIFIER DIRECTIVES

Each of the directives is described below. The Timing Verifier directives and their parameters are not case sensitive. An example Timing Verifier directives file is given at the end.

CLOCK_PERIOD

Used to set the period of the clock used by the Timing Verifier. Any signal with a "C", "P", "S", or "D" name property (e.g. MASTER CLK!C 0-3) has its behavior specified in terms of this period which is in units of nanoseconds:

```
CLOCK_PERIOD  56.0;      set the clock period to
                         56 ns.
```

If unspecified, the Timing Verifier sets the period to 100 ns.

CLOCK_INTERVALS

This directive sets the number of evenly spaced sub-periods within the clock period. For example, if there are 8 sub-periods and the period of the clock is 100 ns then MASTER CLK!C 0-2 is high from time 0 ns to time 25 ns and low from 25ns to 100ns.

```
CLOCK_PERIOD  100.0;   sets the clock period to
                       100 ns.

CLOCK_INTERVALS  20;   divide the clock into 20
                       pieces.
```

The signal MASTER CLK !C 0-10,15-20 would be high from 0 ns to 50ns and high from 75.0 ns to 100 ns and zero otherwise -- MASTER CLK !C 0-10,15-20 = 1:0.0, 0:50.0, 1:75.0 .

If unspecified the clock is divided into 8 sub-periods.

CLOCK_SKEW

This directive sets the uncertainty in the time at which edges in signals with a "C" name property occur. CLOCK_SKEW is skew from the nominal time (in nanoseconds):

```
CLOCK_PERIOD  100.0;   sets the clock period to
                       100 ns.

CLOCK_INTERVALS  20;   divide the clock into 20
                       pieces.

CLOCK_SKEW  0.1;       skew is -0.1, +0.1 ns
                       from nominal.
```

The signal MASTER CLK !C 0-10,15-20 above would become:

```
1:0.0, F:49.9, 0:50.1, R:74.9, 1:75.1
```

If unspecified the CLOCK_SKEW is 0 ns.

DEFAULT_DRIVE

This directive defines the default drive to be used by the delay estimator when no DRIVE body property is given for a primitive.

```
DEFAULT_DRIVE  0.5-1.2,0.4-1.0
```

If this directive is missing, the default drive is 0.

DELAY_ESTIMATOR

This directive is used to turn the delay estimator on or off.

```
DELAY_ESTIMATOR  ON;     turn the delay estimator on
```

If this directive is missing, the default is to leave the delay estimator off.

DOT_TYPE

Correctly defined timing models have pin properties on dottable outputs. These properties inform the Verifier what logic function is performed when such outputs are connected together. If the properties are missing, or outputs with inconsistent logic function specifications are dotted, the Verifier needs to make some choice for the bus. This choice is specified with the DOT_TYPE directive.

```
DOT_TYPE  DOT_OR;    unspecified buses are dot OR
          DOT_AND;   unspecified buses are dot
                     AND
          DOT_TS;    unspecified buses are
                     tri-state
```

If this directive is missing, unspecified buses are simulated DOT_AND.

LATCH_ERR_MODEL

This directive changes the model used for latches. There are three models for latches, OPEN, CLOSED, and CONSERVATIVE. The differences between these models is defined in the definition of the latch model.

```
LATCH_ERR_MODEL CLOSED;    changes the model used for
                           latches to the closed model.
```

If this directive is missing, the default model to use for latches is CONSERVATIVE.

LIST

This directive takes a list of options separated by commas that control the output listing. These options are:

1. BY_NAME/NOBY_NAME (default BY_NAME) BY_NAME means that signals are to be listed sorted by name instead of by path name. NOBY_NAME results in the old format, with signals sorted by path name. Signals with unique names will be listed by name only, and signals with multiple path names will be listed with each path name indented:

```
CLRINIT !C 0+1.0
  (TST123221 .123.10P) .    .    1:0.0, 0:1.0
  (TST123221 .221.9P)  .    .    1:0.0, 0:1.0
```

   This eliminates many superfluous path names from the signal listing.

2. CHIP/NOCHIP (default NOCHIP) CHIP means that local signals of timing models should be listed.

3. DOT/NODOT (default NODOT) DOT means that the signals generated automatically for the inputs of wire-dots should be listed.

4. HISTOGRAM/NOHISTOGRAM (default NOHISTOGRAM) HISTOGRAM means to generate the histograms on timing error statistics.

5. NC/NONC (default NONC) NC means that the "not connected" signals should be listed.

6. TRAN_INPUT/NOTRAN_INPUT (default NOTRAN_INPUT) Transmission gates have two bi-directional pins, T1 and T2. TRAN_INPUT means to list the value that bi-directional transmission gates see at their pins T1 and T2 that the other drivers on those nets have generated. This feature is useful in debugging complicated circuits with a lot of bi-directional transistors in them.

7. UNNAMED/NOUNNAMED (default NOUNNAMED) UNNAMED means that unnamed signals should be listed in the output listing.

8. VIOLATIONS/NOVIOLATIONS (default NOVIOLATIONS) VIOLATIONS will cause a report of all types of timing violations to be printed. The violations will be sorted in order of descending severity:

Setup Time Violations

Violation	Time	Error #	Primitive
52.0	509.5	11	(MAR TIM11P SUH4P)
30.1	546.0	22	(MAR TIM11P SUH14P)
17.5	546.0	20	(MAR TIM11P SUH4P)
7.0	583.0	12	(MAR TIM11P SUH4P)
2.7	6.9	15	(MAR TIM11P SUH14P)
2.7	6.9	9	(MAR TIM11P SUH4P)
2.6	595.0	17	(MAR TIM11P SUH4P)

Included in this report are: Setup and Hold violations, Low and High pulse width violations, and Minimum and Maximum Edge-to-Edge timing violations.

An example LIST directive is:

```
LIST UNNAMED, NODOT, VIOLATIONS,
     HISTOGRAM, NOBY_NAME, CHIP;
```

MAX_ERRORS

If more than MAX_ERRORS occur during the reading of the input files to the Timing Verifier, verification is aborted.

```
MAX_ERRORS  2;   abort this run if there are
                 more than two errors
```

The Timing Verifier will be aborted if MAX_ERRORS is reached. (If unspecified, then MAX_ERRORS is zero.) However, this does not include fatal errors.

MAX_EVAL_PASSES

If more than MAX_EVAL_PASSES occur during the simulation of the design then verification is aborted.

```
MAX_EVAL_PASSES  50;   abort this run if there
                       are more than fifty passes.
```

If unspecified then MAX_EVAL_PASSES is 200.

MAX_EXP_ERRORS

If more than MAX_EXP_ERRORS are detected in the expansion file then verification is aborted.

```
MAX_EXP_ERRORS  4;    abort this run if there are
                      more than four errors
```

If unspecified then MAX_EXP_ERRORS is 0;

NC_SIGNALS

This directive tells what value NC (non-connected) inputs should be set to. There are 5 possible values, 0, 1, S, ASSERTED, and DEASSERTED. The values 0 and 1 set all non-connected inputs to either 0 or 1, ignoring any bubbled inputs. For example, 0 will make a bubbled input true, and a non-bubbled input false. A value of S will set all non-connected inputs to stable. The value ASSERTED will set all inputs to true (i.e., bubbled inputs to zero, and non-bubbled inputs to one), and DEASSERTED will set all inputs to false. For ECL circuits, DEASSERTED is generally the right thing to do because of the resistor pull-downs on the inputs.

```
NC_SIGNALS DEASSERTED;  set non-connected inputs to
                        deasserted
```

If unspecified, the default value is S.

PREC_CLOCK_SKEW

The directive PREC_CLOCK_SKEW is identical to CLOCK_SKEW except it affects only signals with "P" name properties. If unspecified the PREC_CLOCK_SKEW is 0 ns.

PRINT_WIDTH

This directive tells the Timing Verifier how many

columns may be used in the TVLST file. The output is formatted according to this specification.

```
PRINT_WIDTH 80;    format output for an 80 column
                   display.
            132;
```

If unspecified, the width is 132. (Only 80 and 132 are permitted.)

PULSE_FILTER

This directive turns a pulse filter on that is used by the register and latch model. If a pulse has a large amount of skew, such that the skew is larger than the width of a pulse, then the two edges skew together, causing a changing value to occur. This directive makes the register and latch models recognize this case, and causes it handle it as if the leading edge of the pulse was extended through the changing value.

```
PULSE_FILTER  on;
```

If unspecified, this directive is OFF.

RECONV_FANOUT

This directive tells the Timing Verifier it should analyze the circuit to understand reconvergent fanout, which will eliminate false error messages that are currently generated for circuits that count on correlated signal skews to work.

Reconvergent fanout is where a signal fans out from a common point in a circuit through different paths, which then reconverge at some other point. When these signals come together at a primitive like a setup-hold checker, the skew which is common to them needs to be subtracted out before the check is done, or else false error messages may be generated.

One of the most common cases of this is a shift register. Skew on the clock to the register is common to all of the bits of the register and needs to be subtracted out before checking the setup and hold

times of the shift bits.

Other common terms for reconvergent fanout are correlated skews and common ambiguity.

```
RECONV_FANOUT   ON; do analysis to understand correlated
                    signal skews.
```

If unspecified, the analysis is not done.

RISE_FALL_ANAL

This directive tells the Timing Verifier if it should analyze cascades of inverting logic (with asymetric rise and fall delays) to obtain correct behavior, taking into account the difference between the rising and falling delays, even when the value of the signal is not known (i.e. for stable/changing behavior.)

```
RISE_FALL_ANAL  ON; do analysis exploiting
                    different rise and fall
                    delays for stable/changing
                    values of signals.
```

If unspecified, the analysis is done.

RISE_FALL_MODELS

This directive tells the Timing Verifier if it should analyze cascades of inverting logic (with asymetric rise and fall delays) when the value of the signal is 0, 1, RISE, or FALL. If this directive is turned off, then RISE_FALL_ANAL is also turned off.

```
RISE_FALL_MODELS ON;  do analysis exploiting
                      different rise and fall
                      delays
```

If unspecified, the analysis is done.

SET_MIN_DELAYS

This option allows all minimum delays above the specified value to be set to that value. This

directive is useful when there is concern that the minimum delays along a path may be longer than the allowed maximum delay for a path, causing an error to go undetected.

```
SET_MIN_DELAYS 5.0;    set all minimum delays greater
                       than 5.0 nsec to 5.0 nsec
```

If unspecified, then this feature is turned off.

TIMING_DIAGRAMS

If set, then the Timing Verifier outputs a file that is used to automatically generate Timing Diagrams.

```
TIMING_DIAGRAMS ON;    generate timing diagram
                       output file
```

If unspecified, then this feature is turned off.

TS_BUS_TYPE

The Timing Verifier has two modes for simulating tri-state buses: the true tri-state function and a modified DOT_OR function. The need for the second mode is to handle designs with stable/changing behavior on the enables of the tri-state driver.

```
TS_BUS_TYPE DOT_OR;    do TS buses in forgiving
                       mode
            DOT_TS;    do TS buses exactly
```

The actual logic functions performed by the bus and the TS BUF for the two simulation modes are described earlier in this section of the manual.

If unspecified, then tri-state buses are simulated DOT_TS.

USE_DRAWING_WD

This directive tells the Timing Verifier that it should use any wire delays specified in the drawings. This directive should normally be turned off when using the delay estimator, because any delays in the

drawings will be added to delays calculated by the delay estimator.

```
USE_DRAWING_WD OFF;    turn off use of wire delays
                       specified in drawings
```

If unspecified, this directive is on.

WIRE_DELAY

The default wire delays in a design are specified using these directives.

```
WIRE_DELAY  0.1-2.0;   unless otherwise specified,
                       wires have between 0.1 ns
                       and 2 ns delay
```

If unspecified, the minimum default wire delay is 0 ns and the maximum default wire delay is 0 ns.

WIRE_ESTIMATE

To estimate wire delay, the number of stops on each net is counted. The number of stops is converted to equivalent loads by table lookup using a table specified by this directive. WIRE_ESTIMATE takes an argument list of fixed point numbers and an optional FAMILY specification. A net with j stops receives a wire delay estimate given by the jth number in the list. The family specification allows for a number of different WIRE_ESTIMATE tables to be used in the same Timing Verification run. If a FAMILY body property is given on a primitive, then the WIRE_ESTIMATE table with the same FAMILY specification will be used. If no FAMILY body property is given on a primitive, then the WIRE_ESTIMATE table without a FAMILY specification will be used. An example set of WIRE_ESTIMATE directives are given below:

```
WIRE_ESTIMATE 1.0, 2.0, 3.0, 4.0;
WIRE_ESTIMATE ECL: 0.5, 1.0, 2.0, 3.0;
WIRE_ESTIMATE TTL: 1.0, 2.0, 3.1, 4.0;
WIRE_ESTIMATE ON_GATE_ARRAY: 0.3, 0.6, 1.0, 1.3;
WIRE_ESTIMATE BET_GATE_ARRAY: 1.0, 2.0, 3.1, 4.5;
```

If unspecified, the default is 0.0;

6.16 AN EXAMPLE OF A TIMING VERIFIER DIRECTIVES FILE

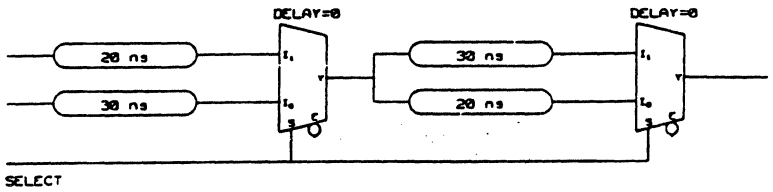
The following gives an example of a Timing Verifier directives file. This file can be created with a text editor. The Timing Verifier does not pay any attention to the end-of-line or to multiple spaces. The letter case of the directives is unimportant. Comments may not be placed in the file. The comments in the example below are only for purposes of documentation. Note that all Timing Verifier directives must be separated by a ';' and the file must end with an 'end.'.

```
CLOCK_PERIOD 132.0;          set the clock period to 132 ns
CLOCK_INTERVALS 12;          clock has 12 intervals of 11 ns
CLOCK_SKEW 1.0;              the clock skew is +/- 1.0 ns
PREC_CLOCK_SKEW 0.1;         precision clock skew is +/- 0.1 ns
WIRE_DELAY 0.0-2.0;          wires may be 0.0 to 2.0 ns long
DOT_TYPE DOT_AND;            non-tri-state busses are dot ANDs
END.                         marks end of the file, note "."
```

Timing Verifier Case Analysis

6.17 INTRODUCTION

Some digital systems are designed so that data dependent delays are exploited. That is, although paths through the system exist that are very long, they are never used. For example consider the contrived example:



The Timing Verifier will assume that the SELECT signal is stable and that the delay through the system is 60 ns. However, simulating this system twice, once with SELECT set and once with it cleared shows that the actual delay is 50 ns. To timing verify systems like this, the Timing Verifier has a mechanism called case analysis.

A case specifies a list of signals, and for each signal a value to substitute for those intervals when the signal is stable (0, 1, or S). Case analysis proceeds by verifying the circuit in the usual manner, except that signals specified in the case file use the specified value instead of the stable value. For example, consider the select signal above:

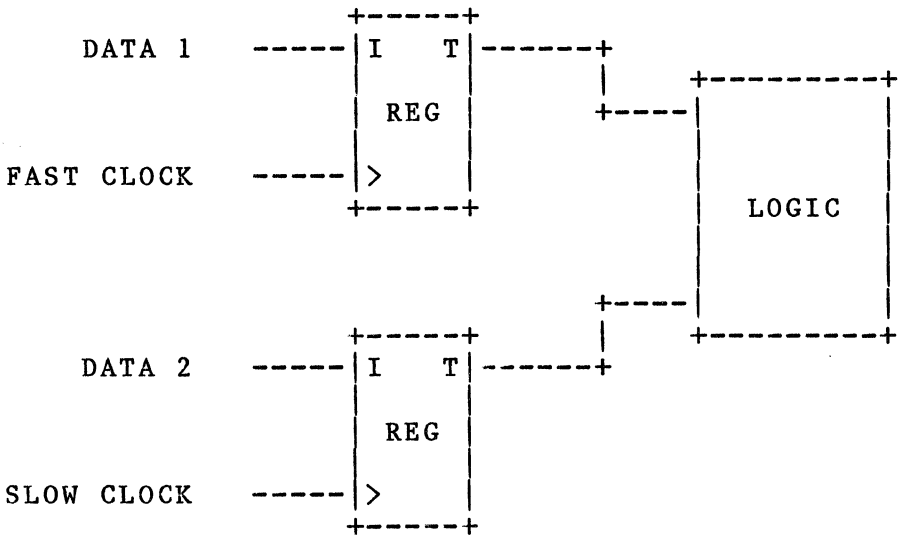
Without case analysis it might have the value:

SELECT S:0.0, C:20.0, S:21.1

Using case analysis, assuming the case specified that SELECT be set to 0:

SELECT 0:0.0, C:20.0, 0:21.1

As a second example of the use of case analysis, consider the following circuit:



Suppose that there are hundreds of transitions on FAST CLOCK for every transition on SLOW CLOCK. This could be modeled by using a very long period for analysis (the period of SLOW CLOCK) and specifying the full behavior of FAST CLOCK over that period. This is time consuming and will result in much redundant information.

A better way to handle this kind of circuit is to consider that there are only two cases of interest. The first case is that time when SLOW CLOCK remains stable during a period of FAST CLOCK. (i.e., only the upper register is being clocked). The second case is when both registers are being clocked.

To do this, a clock assertion is used on FAST CLOCK:

```
FAST CLOCK !C 0-5 ...  1:0.0, 0:50.0
```

(assuming 10 clock intervals per clock period with a 100.0 ns period). If the behavior of SLOW CLOCK is unspecified (assuming that it is undriven as well) then:

```
SLOW CLOCK .........  S:0.0
```

is stable throughout the period of analysis. The two cases of interest may be specified:

```
'SLOW CLOCK' = '!S 0-10';
'SLOW CLOCK' = '!C 0-5';
```

in the case file as described below.

6.18 CASE SPECIFICATION

For any run of the Timing Verifier an arbitrary number of cases may be done. All of the cases are in a file (whose logical name is CASEFILE). Each case is a list of signals and the value that they are to assume.

CASE FILE SYNTAX

The syntax for the case file is:

```
<case file>::= <case list> END. | END.

<case list>::= <case>; | <case>; <case list>

<case>      ::= <signal assignment list> | ;

<signal assignment list>
            ::= <signal assignment> |
                <signal assignment>, <signal assignment list>

<signal assignment>
            ::= <signal name> = <value>

<value>     ::=  '0' | '1'
```

Signal names consisting of only alphanumeric characters and "_" need not be quoted, all other signal names must be quoted. Signal names must be entered in Valid canonical syntax. (That is, all low asserted signals specified with a "_" to the left of the signal. All high asserted signals contain no assertion or negation symbols.)

Signal assignments may refer to scalar signals, complete buses, subranges of a bus, or individual bits of a bus. For example, if the signal DATA<15..0> occurs in the design, then DATA<15..0>, DATA<5>, DATA<3..1>, or DATA<11..4> may appear in the case file. If subscripts are specified, they must be outside the quoted signal name.

EXAMPLE CASE FILE

Assume that a design contains the signals SELECT A1, SELECT B2, CLOCK_RATE and DATA<3..0> . Then a sample case file is:

```
'SELECT A1'  = '0';
;           ;
'SELECT A1'  = '1',
'-SELECT B2'  = '0',
CLOCK_RATE    = '1';
-DATA = '0';
'SELECT A1' = '1',
'-DATA'<3..0> = '1';
eNd.
```

Notice that case files are not case sensitive.

The case file specifies five cases are to be run.

1. The first case sets SELECT A1 to 0 during its stable intervals.

2. The second case does not alter any of the signals. It is equivalent to running the Timing Verifier with no case file.

3. The third case sets SELECT A1 to 1, SELECT B2 to 0 and CLOCK_RATE to 1 during each stable interval.

4. The fourth case sets the bus DATA<3..0> to 1 during its stable interval.

5. The last case sets the bus DATA<3..0> to 1 during its stable intervals and sets SELECT A1 to 1 during its stable intervals.

6.19 USE OF THE CASE FILE FOR TIMING EXPERIMENTS

Often the user wishes to try a number different assertions on a signal or group of signals, to test timing, or discover the correct resetting sequence for his circuit. It is time consuming to do this by modifying the drawings. To facilitate this kind of experimentation, timing assertions may be associated with a signal in the case file.

This is done with an extension to the signal assignment syntax:

```
<signal assignment> ::= <signal name> = <value> |
                        <signal name> = '<timing assertion>'
```

where <timing assertion> is as described earlier in this chapter.

Associating a timing assertion with a signal in the case file is identical to associating the assertion with the signal on the print. Note that if the original signal had a timing assertion as part of its name, the timing assertion must appear as part of the name in the signal assignment.

As an example, consider a design with the signals, RESET, INIT!C 0-3, DATA<15..0> and ID!S 2-4<3..0> in it. Then the following is a correct case file:

```
;

-RESET = '0',
'INIT!C 0-3' = '!P 2-4(-2,+5)',
DATA<15..0> = '!S 14-17,19';

'-RESET' = '!C 0-4',
'ID!S 2-4' = '!D 2-4';

END.
```

An extension to the case file syntax allows specifying a signal's period as different from that specified by the CLOCK_PERIOD directive. The period may be included as part of the timing assertion between the ! character and the assertion character, C, P, S, or D. With CLOCK_PERIOD = 100ns, and CLOCK_INTERVALS = 50, the following case file is correct:

```
'CLKA' = '!25C 10-20
'CLKB' = '!10C 1-2';
END.
```

In this example, CLKA would have a period of 50ns, and would be low for the first 20ns, high for 20ns, and low for 10ns. CLB would have a period of 20ns, be low for 1ns, high for 1ns, and low for 18ns. The signal's period must divide evenly into the number of CLOCK_ INTERVALS.

Wire Delays and the Timing Verifier

6.20 INTRODUCTION

The SCALD Timing Verifier models circuits that contain both component and wire delays. Wire delays are specified in four ways:

1. Simple estimated delays -- these are placed on the logic drawings by the designer.

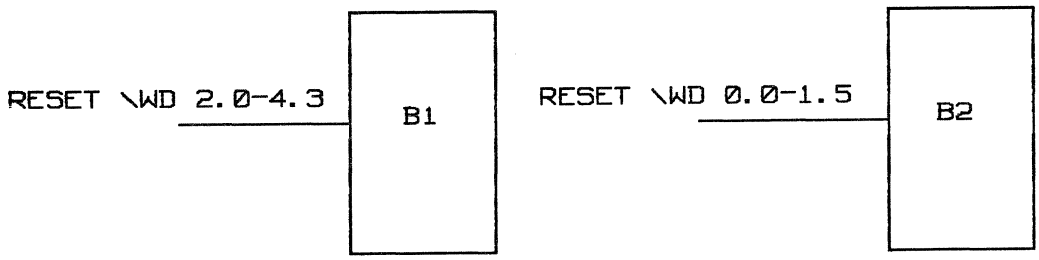
2. Estimated delays that are dependent upon the interconnect delay and the size of the load. This method is described in detail in the following section, SCALD Timing Verifier Delay Estimator.

3. Calculated delays -- these delays are computed by the physical design system and fed back to the Timing Verifier.

4. Default values -- if neither estimated or calculated delays are provided then default values may be specified.

6.21 ESTIMATED WIRE DELAYS

Estimated wire delays are placed on the drawings by the designer using the general signal property WD. An example is included here:



Then the behavior of RESET is:

```
RESET  .  .  .   0:0.0, 1:10.0, 0:20.0 (at the driver)
RESET  .  .  .   0.0.0, R:12.0, 1:14.3, F:22.0, 0:24.3
                 (at B1)
RESET  .  .  .   0.0.0, R:10.0, 1:11.5, F:20.0, 0:21.5
                 (at B2)
```

Delay properties are handled this way so that systems where delays are different on different "stubs" of a net may be modelled.

6.22 CALCULATED WIRE DELAYS

The Timing Verifier will read a file which associates delays with input pins of the components in the system. These delays are applied to the signal driving that pin before the component is simulated. In this fashion, a delay may be associated with each stub on a net. This file is typically generated by a delay calculator which is part of the user's physical design sub-system.

The format of this file is a list. Each element of the list consists of a signal name, a list of the path names of the components that the signal drives and a delay for each pin:

```
'SIGNAL 1' <5..0> :
       '(SYS ALU MUX)' = '2.3-3.4',
       '(SYS REG)' = '0.2-1.7,0.1-1.2';

'SIGNAL 2' <7..5> :
       '(SYS SHIFTER)' = '0.0-1.1';

'SIGNAL 2' <4..1> :
       '(SYS SHIFTER)' = '0.5-3.1';

'SIGNAL 2' <0> :
       '(SYS SHIFTER)' = '0.5-3.1';
end.
```

If no bit numbers are given after a signal name, then the delay will apply to all of the bits of a bus.

WIRE DELAY FILE FORMAT

The detailed syntax for the wire delay file is:

```
<delay file> ::=  END. | <delay list>; END.

<delay list> ::= <signal delay list>; <delay list>

<signal delay list> ::= <signal name> : <stop delay list>
<stop delay list> ::= <stop delay>; |
                      <stop delay>, <stop delay list>

<stop delay> ::= <quoted path name> = <quoted rise/fall
                                       range>

<signal name> ::= <quoted signal name> |
                  <quoted signal name> < <bit range> >
<quoted signal name> ::= ' <signal name> '
<bit range> ::= <bit number> | <bit number> .. <bit number>
```

<bit number> ::= <integer>

<quoted path name> ::= ' <path name> '

<quoted rise/fall range>
::= ' <delay> ' | ' <delay range> ' |
' <rise delay range> , <fall delay range> '

<rise delay range> ::= <min delay> - <max delay>

<fall delay range> ::= <min delay> - <max delay>

<delay range> ::= <min delay> - <max delay>

<delay> ::= <fixed-point number>

<min delay> ::= <fixed-point number>

<max delay> ::= <fixed-point number>

To simplify feeding back of wire delays, a <path name> in the <stop delay list> may be a unique left substring of the actual path name.

SCALD Timing Verifier Delay Estimator

6.23 DELAY ESTIMATION

In many technologies, the time required for the output of a device to reach its loads is affected by both the interconnect delay and the size of the load:

```
Tr = Tdr + Kr(load on the net) + Tir
Tf = Tdf + Kf(load on the net) + Tif    where
```

```
Tdr   is the device's rise delay
Kr    is a device-specific constant related to changes in
      the output's rise time as a function of device loading
Tir   is the rising edge delay due to wires

Tdf   is the devices fall delay
Kf    is a device specific constant related to changes in
      the outputs fall time as a function of device loading
Tif   is the falling edge delay due to wires
```

```
Note: Tdr, Tdf, Kr, Kf are device specific;
      (load on the net) is net specific; and
      Tir, Tif  input specific.
      All quantities can assume both a min and max value.
```

We can lump the net loading term and interconnect term of the delay. Then the delay due to all interconnection effects can be modeled as an input specific wire delay. If interconnection delays are computed (or estimated) this way by the physical design sub-system, and then fed back to the Timing Verifier as wire delays (the DELAY.DAT file), we obtain an accurate timing representation of the system. Early in the design cycle however, it may be impractical to provide such detailed delay information -- estimators are required.

The Timing Verifier can use estimated wire delays provided by the designer on his print set, or use a global default value for wire delays. (See Wire Delays and the Timing Verifier.) This is often adequate if the signal delay due to loading effects is small.

If the loading effect is not small, the Timing Verifier has a more accurate delay estimator that takes into account static load, and provides a wire delay estimate based on the number of stops (inputs and outputs) on the net. This delay estimate is added to the basic device delay (RISE, FALL, or DELAY parameter on the body) and overides any other specified wire delays.

Tr(estimated) = Tdr + Kr(loads on the net + wire delay)
Tf(estimated) = Tdf + Kf(loads on the net + wire delay)

where constants Tdr, Kr, Tdf, Kf are as above.

The load term is a weighted sum of inputs and outputs on the net which approximates the true capacitive and DC load on the net. The wire delay is estimated by counting the number of stops on the net and converting stops into load equivalents.

COMPUTING NET DEPENDENT DELAYS

The Timing Verifier estimates the net delay on each net in six steps:

1. The load is estimated by taking a weighted sum of the inputs and outputs on the net.

2. The number of stops on the net is counted.

3. The number of stops is converted to an interconnection delay estimate (in units of load equivalents) by table look-up.

4. An effective net load is computed by adding the interconnect and load estimates.

5. The effective net loading is multiplied by the drive constants (Kr and Kf) of the drivers of the net to obtain rise and fall delays due to net loading.

6. These delays are added to the drivers zero-load parameters (Tdr and Tdf).

Counting Loads

Counting the inputs and outputs on a net is complicated by the presence of TIMES parameters and dots.

If an input pin is part of a device with a TIMES parameter of Ni on it, the input is counted Ni times. If an output pin of a device has a times parameter of No on it, it is counted once and the number of inputs on the net is divided by No. If several outputs are dotted together the previous rules apply with two changes. The number of outputs on the net is the sum of all the outputs ignoring their times parameters. The number of inputs on the net is counted as before and then divided by the smallest TIMES

parameter of any device with an output on the net.

If phantom gates are used, they are collapsed to an explicit dot for the counting procedure.

Finally, the user may place an optional pin property, LOAD_FACTOR, on any pin. LOAD_FACTOR takes a fixed point number as a value. If LOAD_FACTOR is specified, a pin is counted LOAD_FACTOR times (or LOAD_FACTOR Ni times if it is an input pin and a TIMES parameter of Ni is present) rather than once in the above counting procedure.

Estimating Wire Delays

To estimate wire delay, the number of stops on each net is counted. If phantom gates are used, they are collapsed into an explicit dot for the stop counting process. The number of stops is converted to equivalent loads by table lookup using a table specified with the Timing Verifier directive WIRE_ESTIMATE. WIRE_ESTIMATE takes an argument list of fixed point numbers and an optional FAMILY specification. A net with j stops receives a wire delay estimate given by the jth number in the list. The family specification allows for a number of different WIRE_ESTIMATE tables to be used in the same Timing Verification run. If a FAMILY body property is given on a primitive, then the WIRE_ESTIMATE table with the same FAMILY specification will be used. If no FAMILY body property is given on a primitive, then the WIRE_ESTIMATE table without a FAMILY specification will be used. An example set of WIRE_ESTIMATE directives are given below:

```
WIRE_ESTIMATE 1.0, 2.0, 3.0, 4.0;
WIRE_ESTIMATE ECL: 0.5, 1.0, 2.0, 3.0;
WIRE_ESTIMATE TTL: 1.0, 2.0, 3.1, 4.0;
WIRE_ESTIMATE ON_GATE_ARRAY: 0.3, 0.6, 1.0, 1.3;
WIRE_ESTIMATE BET_GATE_ARRAY: 1.0, 2.0, 3.1, 4.5;
```

Computing Load Dependent Net Delays

Each Timing Verifier primitive within a component model, that drives an output pin of the part being modelled, can have an optional drive constant body property, DRIVE. This property takes a pair of fixed point ranges as a value, the first number is the driver's Kr factor, the second its Kf factor. A range is required to specify both a minimum and maximum value. If no DRIVE property is specified, Kr and Kf are set to a global default. This default is specified in the directives file with the directive DEFAULT_DRIVE. If this is unspecified as well, Kr and Kf are set to 0-0. If only one fixed point number is provided,

Kr and Kf are both set to the value.

After the effective net loading has been computed for a device's output net, the device's output delays (DELAY, or RISE/FALL) are adjusted on a bit-by-bit basis, by the time obtained by multiplying its drive constant(s) by each output bit's effective net loading.

Load estimation can be disabled using the DELAY_ESTIMATOR directive.

USING LOAD DELAY ESTIMATION

Using load estimation entails the following:

1. Load estimation is turned on and off with the load estimator directive: DELAY_ESTIMATOR { ON | OFF }. OFF is the default value.

2. Specifying drive constants (Kr, and Kf). This can be done in two ways:

   a) By attaching a the DRIVE body property to each Timing Verifier primitive whose output is to display load-dependent behavior:

      <drive> ::= DRIVE = <rise and fall delay> |
                  DRIVE = <rise delay>,<fall delay>

      <rise and fall delay> ::= <delay> | <min delay>-<max delay>

      <rise delay> ::= <delay> | <min delay>-<max delay>

      <fall delay> ::= <delay> | <min delay>-<max delay>

      <min delay> ::= <delay>

      <max delay> ::= <delay>

      <delay> ::= <fixed point number>

   b) If no body property is specified, a default value is used. It is set with the default drive directive:

      DEFAULT_DRIVE <rise and fall delay factor> or
      DEFAULT_DRIVE <rise delay factor>,<fall delay factor>

      the default value for DEFAULT_DRIVE is 0.

3. Specifying pin loading. This is done by attaching the LOAD_FACTOR property to a pin of the Timing Verifier primitive that connects to the interface signal that represents the pin of the part whose load is to be specified.

<pin load> ::= LOAD_FACTOR = <fixed point number>

If no property is specified, a default LOAD_FACTOR of 1 is used.

4. Specifying a conversion table from stops to load equivalents. This is done with the WIRE_ESTIMATE directive:

<wire estimate directive> ::= WIRE_ESTIMATE <wire estimate options>

<wire estimate options> ::= <list of delays> |
<family> : <list of delays>

<list of delays> ::= <fixed point number> |
<fixed point number>,<list of delays>

<family> ::= <identifier up to 16 characters long>
where
a net with j stops receives a delay estimate given by the jth fixed point number in the list. This list may have a maximum of 100 entries, and runs with over 100 stops will use the last entry.

NOTES ON THE COUNTING MODELS

This scheme for counting stops and loads on a net is independent of the actual wiring of a net. In two significant cases this results in delay estimates that are too large.

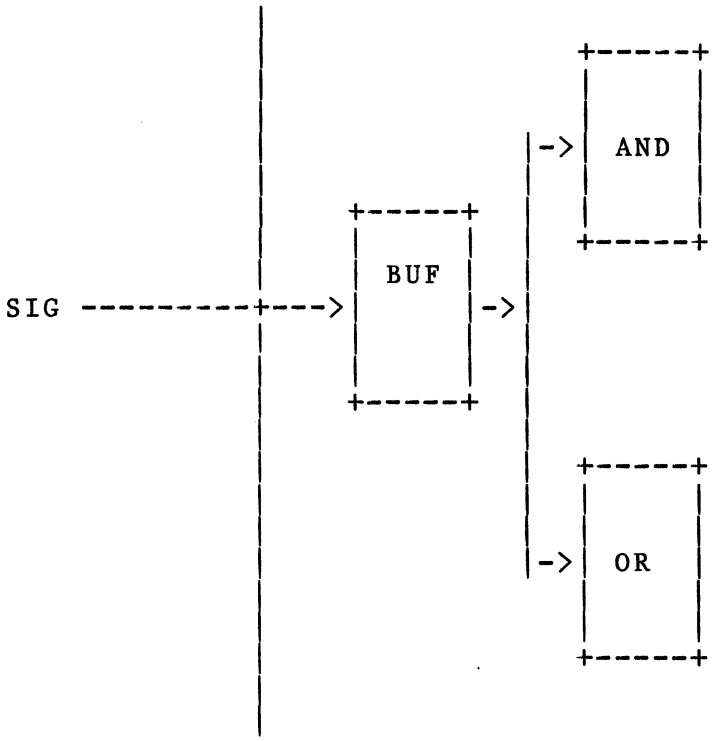
Drivers with TIMES parameters, especially feeding phantom gates, are often wired with a careful partitioning and placement of the loads. The estimation scheme does not take this into account. It assumes a load that results from an even partitioning of the loads into a number of pieces equal to the driver with the smallest TIMES parameter.

Physical parts often have common input pins which are modelled as separate pins. For example two SIZE=4B, 74LS374s might be driven by the same clock signal and hence be allocated to the same package. However, since two logical parts appear on the prints, two 74LS374 clock pins

will appear on the net instead of one.

In order to ensure correct counting of loads, timing models should only connect one primitive (not counting checker bodies) to each interface signal. This can always be accomplished using zero delay non-inverting buffers, as shown below:

Higher Level Drawing Timing Model



6.24 INTERACTION OF WIRE DELAYS WITH DELAY ESTIMATOR

The delays calculated by the delay estimator are used to adjust the delays of the driving components, and as such will be added to wire delays specified in the drawings or fed back wire delays specified in the wire delay file. The directive USE_DRAWING_WD allows the user to control whether the wire delays specified in the drawings will be used or not. In general, any fed back wire delays will override any wire delays specified in the drawings. It is suggested that USE_DRAWING_WD normally be turned off when the DELAY_ESTIMATOR is on.

Timing Diagram Plotter (PLOTTIME)

6.25 INTRODUCTION

The timing diagram plotting program PLOTTIME converts the tabular output of the Timing Verifier or the Logic Simulator to conventional timing diagrams for display by the Graphics Editor and for hardcopy output through the Graphics Editor's hardcopy facility. The Plottime program runs exclusively on the S-32 cluster controller.

6.26 DIRECTIVES FILE

The Plottime program requires a directives file (td.cmd) to define the input file to be plotted, the SCALD directory and file name of the output or "plot" file, and the timing diagram display parameters (e.g., number of signals displayed, nanoseconds per centimeter, etc.). A typical td.cmd directives file looks like this:

```
DIRECTORY 'MY_SCALD_DIRECTORY';
INPUT 'MY_INPUT_FILE';
OUTPUT 'MY_TIMING_DIAGRAM.MY_EXTENSION';
NS_PER_INCH = 20;
NS_PER_TICK = 10;
SIGNALS_PER_PAGE = 15;
END.
```

Referring to the above directives file, each entry is on a single line and must be terminated by a semicolon; an "END." statement is required following the last entry. The individual entries are defined as follows:

- DIRECTORY -- the name of the SCALD directory to contain the output (plot) file; if this entry is omitted, the program PLOTTIME will abort.

- INPUT -- the name of the input file to be plotted from the Timing Verifier or Logic Simulator; if this entry is omitted, the file "plotsig.dat" is used by default. Note that the name of the Timing Verifier output file is always "plotsig.dat"; the default name of the Logic Simulator output file is "plotsig.dat" or the file name specified in the Plot command.

- OUTPUT -- the name of the plot file used by the Graphics Editor to display the timing diagram. If an output (plot) file is not specified, the file "timing.timing" is written to the corresponding SCALD directory as the default timing drawing. Note that a

version or page number extension is ignored in the file name.

- NS_PER_INCH -- the number of nanoseconds represented by an inch of timing diagram display. Note that a NS_PER_CM entry alternately can be used.

- NS_PER_TICK -- the number of nanoseconds per tick mark on the timing diagram.

- SIGNALS_PER_PAGE -- the number of timing signals plotted per page. The Plottime program plots all signals within a design; when the number of signals to be plotted exceeds the SIGNALS_PER_PAGE entry, a set of plot files (pages) are generated (i.e., TIMING.TIMING.1.1, TIMING.TIMING.1.2, etc.).

6.27 EDITING THE TABULAR INPUT

The input file specified in the Plottime directives file is the tabular output ASCII file generated by the Timing Verifier or Logic Simulator (i.e., plotsig.dat). This file can be edited with one of the UNIX text editors (e.g., vi) prior to running Plottime in order to remove any unwanted signals from the timing diagram. The first part of the file defines the labels that appear across the bottom of the timing diagram. The actual timing signal descriptions follow the labels and are defined as a sequence of state-time values. The timing signals generated by the Timing Verifier are listed in alphabetical order; timing signals generated by the Logic Simulator are listed in the order in which they are "opened" in the Waveforms mode and, if specified, their display position (row). The signal name is enclosed in single quotes at the beginning of a line; the signal description can be any number of lines and is terminated by a semicolon. The following example shows a typical timing signal description.

```
'CLOCK SYNC 1'0:0.0,1:10.0,0:50.0,1:60.0;
```

6.28 CREATING TIMING DIAGRAMS

In order to create a timing diagram using Plottime, the drawing must be compiled (for TIME or SIM) and the Timing Verifier or Logic Simulator must be run to generate the tabular output file (plotsig.dat). Remember that when setting up the Timing Verifier's directive file (verifier.cmd), the TIMING_DIAGRAMS flag must be set to "ON." When running the Logic Simulator, the PLot command is used to create the tabular output file (see Simulator

commands in Chapter 7).

If the Timing Verifier or Logic Simulator is run on the host (VAX or IBM), the tabular output file must be transferred to the cluster controller using the file copy utility (see Chapter 12).

Next, edit (or create) the "td.cmd" directives file. Note that if the timing diagram(s) are to fit on a B size (11x17) page, the total length of the timing diagrams should be 10 inches (e.g., to display a 200 nanosecond waveform, the NS_PER_INCH entry would be set to 20). The total length of a plot should not exceed 30 inches. The maximum number of signals that can be plotted on a B size page is 15.

After the corresponding tabular output file and the Plottime directives file are created, run the Plottime program (from UNIX) by entering:

plottime

after the UNIX prompt and then pressing RETURN. Plottime will create the required number of pages of the timing drawing and will automatically update the drawing name in the SCALD directory (default name "timing.timing").

Once the drawing file has been created, enter the Graphics Editor and display (edit) the drawing file.

NOTE

The Plottime program automatically updates the SCALD directory. However, if the Graphics Editor is running (open) when the Plottime program is run, the directory currently referenced by the Graphics Editor (and not the updated directory) will be used. To update the SCALD directory without exiting the Graphics Editor, the following sequence of Graphics Editor commands is used:

```
IGNORE MY_SCALD_DIRECTORY
USE MY_SCALD_DIRECTORY
```

When the number of timing signals exceeds the SIGNALS_PER_PAGE directive, the Graphics Editor displays the first page of the timing diagram (default file name timing.timing.1.1). To display a subsequent page, enter

EDIT ...n

where "n" is the page number to be displayed. Note that the timing diagrams are conventional drawings and can be manipulated using the Graphics Editor (e.g., a timing signal can be grouped and then moved or deleted). The Graphics Editor's HArdcopy command is used to plot the displayed timing diagram.

6.29 TIMING VERIFIER ERROR MESSAGES

Error messages are generated by the Timing Verifier for a large number of erroneous conditions. Each occurrence of a specific type or "class" of error is assigned a sequential number. This number, the error class, the specific error number, and a brief description of the error are written to the Timing Verifier's list file (tvlst.dat). The sections that follow define the classes of errors and describe each error and its probable cause.

6.30 CLASSES OF ERRORS

Errors detected by the Timing Verifier fall into one of the following three classes:

- Syntax Errors
  Syntax errors are typographical errors or violations in the specified form of a character string and are detected when the Timing verifier is searching any of its four input files.

- Timing Errors
  Timing errors are detected by the checker primitives. Common timing errors include setup and hold violations and minimum pulse width violations.

- Runtime Errors
  Runtime errors occur after the input files are read and while the Timing Verifier is processing the design. An error that is not a syntax or a timing error is a runtime error.

6.31 FORMAT OF MESSAGES

Syntax, timing, and runtime error messages are formatted in the list file as follows:

#(n) Syntax error(m): <message>

#(n) Timing error(m): <message>

#(n) Runtime error(m): <message>

In the above formats, "n" is a running count of the number of occurrences of the corresponding class of error, "m" is the actual error number, and <message> is a brief description of the error. For example

#287 Syntax error(22): String length exceeded

indicates that the this entry is the 287th syntax error and that this specific error is error #22, "string length exceeded" (i.e., a character string in excess of 255 was encountered).

Following each error message entry may be several lines that describe the location of the error (e.g., the drawing, the body on the drawing, or the pin on the body where the error was detected). This information is included to assist the designer in finding and correcting the problem.

Syntax, timing, and runtime error messages are counted separately. At the end of the verification run, the total number of errors for each class is reported. For example:

12 syntax errors detected.
No timing errors detected.
One runtime error detected.

6.31 SUMMARY OF THE MESSAGES BY NUMBER

The remainder of this section contains an ordered (by error number) description of each Timing Verifier error message. Included with the descriptions are suggestions on the probable cause of the error and how to recover from the error.

Each error number is listed with one of the following:

- Syntax error message text.
- Timing error message text.
- Runtime error message text.
- "Unused" (the error message number is available for future use).
- "Reserved" (the error number is used for debugging or other Valid internal operations).

Error #0: Unimplemented error message

An error of undetermined type has been detected.

Syntax error #1: Expected identifier

This error is generated whenever the Verifier is expecting an identifier (a string of letters, digits, or '_' starting with a letter) and finds some other data. Identifiers are used as names in properties, text macros, and as operands for Verifier directives. The Verifier prints the input line along with a pointer to the position in the line where the problem was detected.

Syntax error #2: Expected =

This error is generated whenever the Verifier is expecting an equal (=) and finds some other data. Equals are used in many places: between property names and values, and in expressions. The Verifier prints the portion of the input line it read before it encountered the error.

Syntax error #3: Reserved.

Syntax error #4: Reserved.

Syntax error #5: Reserved.

Syntax error #6: Reserved.

Syntax error #7: Expected)

This error is generated whenever the Verifier is expecting a right parenthesis ( )) and finds some other character. The Verifier prints the portion of the input line it read before the error was encountered.

Syntax error #8: Expected ,

This error is generated whenever the Verifier is expecting a comma (,) and finds some other data. Commas are used to separate elements in lists and are required, for example, in specifying options to the LIST command. The Verifier prints the portion of the input line it read before the error was encountered.

Syntax error #9: Reserved.

Syntax error #10: Expected <

This error is generated whenever the Verifier is expecting a less than character (<) and finds some other character. The Verifier prints the portion of the input line read before the error was encountered.

Syntax error #11: Expected >

This error is generated whenever the Verifier is expecting a less than character (>) and finds some other character. The Verifier prints the portion of the input line read before the error was encountered.

Syntax error #12: Expected ;

This error is generated whenever the Verifier is expecting a semicolon (;) and finds some other character. The Verifier prints the portion of the input line read before the error was encountered.

Syntax error #13: Expected :

This error is generated whenever the Verifier is expecting a colon (:) and finds some other character. The Verifier prints the portion of the input line read before the error was encountered.

Syntax error #14: Reserved.

Syntax error #15: Expected (

This error is generated whenever the Verifier is expecting a left parenthesis (( ) and finds some other character. The Verifier prints the portion of the input line read before the error was encountered.

Syntax error #16: Reserved.

Error #17: Unused.

Error #18: Unused.

Error #19: Unused.

Syntax error #20: Unmatched closing comment character

This error is generated when the Verifier encounters a closing comment character (}) without a matching starting comment character ({). The Verifier prints the portion of the input line read before the error was encountered. Either this symbol is extraneous or the beginning of the comment was never specified. If the symbol really is extraneous, the Verifier continues with no further errors. If it isn't, bogus errors will probably have been generated as the Verifier tried to read the text of the comment.

Syntax error #21: Nested comments not allowed

Comments within comments are not allowed in Timing Verifier input files. This error is generated if input of the form:

```
{ This is a comment { This is a nested comment }}
```

is encountered.

Syntax error #22: String length exceeded

This error is generated as the Verifier is reading a string and finds that the string is too long. Strings are limited to 255 characters. The Verifier prints the portion of the input line read before the error was encountered. The string is truncated at the current position and the Verifier reads until it finds the closing quote or the end of the input line. Make the string shorter!

Syntax error #23: Illegal character found

This error is generated when the Verifier finds an illegal character in an input file. All non-printing characters except TAB are illegal. The Verifier prints the portion of the input line read before the error was encountered. Remove the character.

Syntax error #24: Expression value overflow

This error is generated when the Verifier evaluates an expression whose value overflows. The Verifier prints the portion of the input line read before the error was encountered. An overflow does not cause the Verifier to abort; it assigns the value 0 to the result (unless it knows a more reasonable value) and continues with the verification.

Syntax error #25: Reserved.

Error #26: Unused.

Error #27: Unused.

Error #28: Unused.

Error #29: Unused.

Syntax error #30: Reserved.

Error #31: Unused.

Syntax error #32: Non-printing character found

This error is detected when the Verifier is reading characters from an input file and a non-printing character is encountered (non-printing characters are not permitted). The Verifier prints the portion of the input line read before the error was encountered.

Syntax error #33: Expected a string

This error is detected when the Verifier is expecting a string (a quoted sequence of printing characters) and finds some other data. The Verifier prints the portion of the input line read before the error was encountered.

Syntax error #34: Comment not closed before end of input

This error is detected when the Verifier does not find the end of a comment before the end of the file. A comment begins with a "{" character and ends with a "}" character. The Verifier prints the portion of the input line read before the error was encountered.

Error #35: Unused.

Syntax error #36: Reserved.

Syntax error #37: Expected .

This error is generated when the Verifier is expecting a period (.) and finds some other character. The Verifier prints the portion of the input line read before the error was encountered. This error is most commonly caused by omitting the '.' following the END at the end of the directives, case, or wire delay file.

Error #38: Unused.

Syntax error #39: Reserved.

Syntax error #40: Expected END

This error is generated when the Verifier reaches what it expects to be the end of a file and no END is found. An END must be present at the end of the directives, case, and delay files. The Verifier prints the portion of the input line read before the error was encountered. The END is used to inform the Verifier that the file is complete and that it isn't unfinished or missing some text.

Syntax error #41: Reserved.

Error #42: Unused.

Error #43: Unused.

Error #44: Unused.

Error #45: Unused.

Error #46: Unused.

Error #47: Unused.

Error #48: Unused.

Syntax error #52: Reserved.

Syntax error #53: Reserved.

Syntax error #54: Reserved.

Syntax error #55: Reserved.

Error #56: Unused.

Syntax error #57: Reserved.

Syntax error #58: Reserved.

Error #59: Unused.

Syntax error #60: Reserved.

Syntax error #61: Reserved.

Error #62: Unused.

Error #63: Unused.

Error #64: Unused.

Error #65: Unused.

Error #66: Unused.

Syntax error #69: Reserved.

Syntax error #70: Reserved.

Error #71: Unused.

Syntax error #72: Reserved.

Syntax error #73: Reserved.

Syntax error #74: Reserved.

Syntax error #75: Reserved.

Syntax error #76: Reserved.

Syntax error #77: Reserved.

Syntax error #78: Reserved.

Syntax error #79: Reserved.

Syntax error #80: Output digits must be 0, 1, 2, or 3

This error will occur if an illegal numerical value is given to the OUTPUT_DIGITS directive. Legal values are 0 (display nanoseconds) through 3 (display picoseconds.) See the directives section for more information.

Error #81: Unused.

Error #82: Unused.

Error #83: Unused.

Error #84: Unused.

Syntax error #85: Reserved.

Runtime error #86: Reserved.

Syntax error #87: Reserved.

Error #88: Unused.

Syntax error #89: Reserved.

Syntax error #90: Reserved.

Syntax error #91: Reserved.

Error #92: Unused.

Syntax error #93: Reserved.

Error #94: Unused.

Error #95: Unused.

Error #96: Unused.

Syntax error #97: Reserved.

Syntax error #98: Reserved.

Runtime error #99: Reserved.

Runtime error #100: Assertion check failure: save Log File.

This error is generated whenever the Verifier discovers some internal data problem. The Verifier is constantly checking to make sure that its internal data is consistent. If it detects some problem, this message is generated. Contact Valid Logic Systems for a work around and/or corrections. This message indicates an internal Verifier error and usually cannot be fixed by the user. Save the data that caused the error as it will be very helpful in finding the problem. It is very important that the TVLOG file be saved (at a minimum). Valid may also request any of the input or output files for the Verifier. Try to be ready to reproduce the problem for the Service Engineer.

Runtime error #101: Cannot open compiler output (CMPEXP)

This error is generated when the Verifier cannot find the compiler output file, CMPEXP.DAT (CMPEXP DATA on the IBM). This file must be in the directory (mini-disk on the IBM) where you are running the Timing Verifier. See the SCALD Compiler documentation in Chapter 5 for information on how to compile your drawing for the Timing Verifier.

Runtime error #102: Compiler expansion file has wrong type

This error occurs when an expansion file is found, but the drawing has not been compiled for the Timing Verifier. Possibly the drawing was compiled for the SCALD Logic Simulator. Check the compiler directives file (compiler.cmd) and read the SCALD Compiler documentation.

Syntax error #103: Number too large

This error is generated when the Verifier detects an integer value that is larger than 99999. This could occur when reading any of the input files: the compiler expansion file, the directives file, the case file, or the wire delay feedback file.

Syntax error #104: Illegal character in number

This error is generated when the Verifier finds a character other than a digit or a decimal point in a number. This could occur when reading any of the input files: the compiler expansion file, the directives file, the case file, or the wire delay feedback file.

Syntax error #105: EOF encountered

The end of an input file (EOF) was found prematurely. This means before "END." appeared in the file.

Syntax error #106: Reserved.

Runtime error #107: Reserved.

Syntax error #108: Continuation character not at EOL.

The Verifier found a line in an input file that was not ended properly. A string was being read, and it contained a newline character. Strings that extend past character 80 in an input file should have a continuation character (~) to indicate that they continue on the next input line.

Syntax error #109: String too long

This error is generated when the Verifier finds a string that extends over 255 characters. Make the string shorter.

Syntax error #110: Bad delimiter

This error occurs when the Verifier is expecting some delimiter (such as a double quote ending a string), and finds a different one. The expected delimiter is printed with the portion of the input line read before

the error was encountered.

Syntax error #111: Expected quoted string

This error is generated when the Verifier encounters something other than a string in quotes when it is expecting a quoted string. The portion of the input line read before the error was encountered is printed out.

Runtime error #112: Reserved.

Syntax error #113: Invalid width of signal

This error occurs if the Verifier computes a signal that has a period the is not equal to the CLOCK_PERIOD. This is a serious error, and if it ever occurs, should be reported immediately.

Syntax error #114: Reserved.

Syntax error #115: Multiple values given for signal

This error is generated if more than one value is given for a signal in one case in the case file. This obviously makes no sense, so remove the incorrect case specification.

Runtime error #116: Max number of evaluation passes executed

If the circuit does not converge within the number of evaluation passes specified by the MAX_EVAL_PASSES directive (which currently defaults to 200), this error is generated. Many evaluation passes may be required for circuits with feedback loops in them.

Runtime error #117: Resistor connected to constants at both ends

This error occurs when the Verifier tries to orient the resistors in the design. A resistor connected to a constant signal (1 or 0) at both ends is not allowed for the Timing Verifier and usually indicates a design error.

Runtime error #118: Resistor driven at both ends

The Verifier generates this error when it finds a resistor that has primitive outputs attached to both ends. This resistor cannot be oriented. The verification run continues, but the resistor is ignored. Change the circuit so that all resistors are driven at only one end.

Runtime error #119: Part not orientable

This error is generated if a resistor cannot be oriented. Each resistor must have unique, unambiguous input and output sides; they are not allowed to be truly bidirectional.

Runtime error #120: The following parts are unorientable

This error is generated if more than one resistor cannot be oriented; see Runtime error #119 above.

Syntax error #121: Max time is smaller than min time

This error is generated by the Verifier whenever it finds a maximum time that is less than the corresponding minimum time. Specifying DELAY=5.0-4.0, for example, would cause this error.

Syntax error #122: Single time variable expected, not range

This error occurs when a minimum-to-maximum range (min-max) is specified where only a single time value is expected. Examples: setup times, hold times, minimum pulse widths, etc.

Syntax error #123: Reserved.

Syntax error #124: Illegal transition type specified

This error in generated when the Verifier finds a Transition Type other than SMOOTH or GLITCHY in the expansion file.

Syntax error #125: Illegal strength type specified

This error in generated when the Verifier finds a strength other than SOFT or HARD in the expansion file.

Syntax error #126: Illegal character in evaluation string

Evaluation characters must be either: A, I, E, Z, H, or W. See the section of the documentation on Evaluation strings.

Syntax error #127: Bit numbers specified are out of range

This error is generated when bit subscripts specified in the case file do not agree with the signal width found in the expansion file. If a signal has bits <5..2> in the design, any specification in the case file for that signal may only refer to bits 5 through 2 or to the whole signal.

Syntax error #128: Illegal character in signal list

This error is generated by the Verifier if it finds extraneous or incorrect input in the properties connected to a signal in the expansion file. The various property names must be spelled correctly, and the other elements such as equal signs (=) and semicolons (;) must be in the proper places.

Syntax error #129: Time range given for clock delay

The value of a CLOCK_DELAY property must be a single value, not a minimum-maximum range.

Syntax error #130: Undefined pin

This error is generated if an incorrect pin name is found for a Timing Verifier primitive in the expansion file. The correct pin names are documented in the section on Verifier primitives.

Syntax error #131: No signal passed to parameter

This error in generated if there is no signal bound to a pin of a Timing Verifier primitive in the expansion file. If this happens, without any Compiler errors, it is indicative of a SCALD Compiler bug.

Syntax error #132: Missing parameter

This error is generated if a required pin of a primitive is not found in the expansion file. If optional pins such as enables on checker primitives do not exist, this error will not be generated. This is indicative of either a library or a SCALD compiler bug.

Runtime error #133: Incorrect width parameter passed to formal

This error is generated when a signal and the pin to which it is connected are found to have different widths in the expansion file. This should have caused an error to be generated by the SCALD Compiler.

Syntax error #134: Illegal value given to boolean option

Boolean Verifier directives must be given either the value TRUE or the value FALSE. Any other value will generate this error.

Syntax error #135: Illegal value given to on/off option

Verifier directives such as RECONV_FANOUT and DELAY_ESTIMATOR require either the value ON or the value OFF. Any other value will generate this error.

Syntax error #136: Unknown dot type specified

The legal values for the DOT_TYPE directive are: DOT_OR, DOT_AND, and DOT_TS. Any other value will generate this error.

Syntax error #137: Expected bit ordering specifier

This error is generated if the BIT_ORDERING directive is read, and the value assigned is not either RIGHT_TO_LEFT or LEFT_TO_RIGHT.

Syntax error #138: Too many entries given in wire estimate list

The maximum number of wire estimates that can be specified in a wire estimate list is currently 100. Any additional estimates will be ignored and will cause this error to be generated.

Syntax error #139: Unknown option given

This error is generated if an unknown (illegal or undefined) Timing Verifier directive is specified in the directives file. It is also generated if the value of the DELAY_MODEL directive is not one of the legal values: MIN, MAX, MIN/MAX, RISE/FALL, or a combination of the legal values. This is most likely caused by a spelling error.

Syntax error #140: Unknown syntax specification

Signal specifications have up to five parts: the property specifier, the assertion specifier, the subscript specifier, the name specifier, and the negation specifier. If the values given for the SYNTAX specification are anything else, this error will be generated.

Syntax error #141: Invalid clock period specified

If the CLOCK_PERIOD is specified as less than one nanosecond, this error will be generated, and the CLOCK_PERIOD set to the default, 100 nanoseconds.

Syntax error #142: Invalid number of clock intervals specified

If the number of CLOCK_INTERVALS specified to the CLOCK_INTERVALS directive is less than one or greater than 10000 times the CLOCK_PERIOD, this error will be generated.

Syntax error #143: Invalid tri-state bus type

The only legal values for the TS_BUS_TYPE directive are DOT_OR and DOT_TS. This error is generated if any other value is specified. The value is then set to the default, DOT_TS, or to DOT_OR if a previous TS_BUS_TYPE directive had the value DOT_OR.

Syntax error #144: NC_SIGNALS set to illegal value

The legal values for the NC_SIGNALS directive are: 0, 1, S, ASSERTED, and DEASSERTED. Using any other value will generate this value and cause the value S (stable) to be used.

Syntax error #145: PULSE_EDGE_CORR must be between 0 and 1

Legal values for the PULSE_EDGE_CORR directive range between 0 and 1. See the directives summary for more information. If an illegal value is used that generated this error, the value 1 will be used as a default.

Syntax error #146: Print width invalid

Valid values for the PRINT_WIDTH directive are from 80 to 132, inclusive. Specifying other values will generate this error. When this occurs, the value will default to 132. See the PRINT_WIDTH directive for more information.

Syntax error #147: Invalid number of passes specified

Specifying a value less than one for the MAX_EVAL_PASSES directive will generate this error. Values less than one are meaningless, so the value will default to 200 if this error in encountered.

Syntax error #148: Expected FILE_TYPE

This error is generated when the Verifier finds a file called CMPEXP.DAT but the first line in the file is not "FILE_TYPE" (the first line of a compiler expansion file for the Verifier must be either
"FILE_TYPE=CMP_EXPANSION;" or
"FILE_TYPE=TIME_EXPANSION;"). Make sure the proper expansion file is in your current directory, and that the drawing was compiled for TIME.

Syntax error #149: Unknown primitive

This error is generated by the Verifier when it reads a primitive from the expansion file that has an unknown type. This should only happen if the expansion file is edited by hand, and a primitive's name is changed accidently. The primitive will be ignored.

Syntax error #150: Expected "END_PRIMITIVE"

This is another error that should only be caused by erroneous hand editing of an expansion file. Every primitive in the expansion has the keyword, "END_PRIMITIVE" at the end of its description. This can be hard to recover from, and in some cases can cause many extraneous errors to be generated.

Syntax error #151: Unknown block type in expansion file

This is another error that should not ever be generated unless an incorrectly hand-edited expansion file is used. Legal block types are: DIRECTIVES, TIME, PRIMITIVE, and END.

Runtime error #152: Reserved.

Timing error #153: Edge to Edge timing violation

This error is generated by a Timing Verifier Edge to Edge checker primitive. This indicates a timing error in the design. See the documentation section on checker primitives for more information on this and the other Timing errors.

Error #154: Unused.

Error #155: Unused.

Timing error #156: Setup time violation

This error is generated by a Timing Verifier Setup Hold checker primitive. This indicates a timing error in the design. See the documentation section on checker primitives for more information on this and the other Timing errors.

Timing error #157: Hold time violation

This error is generated by a Timing Verifier Setup Hold checker primitive and indicates a timing error in the design. See the documentation section on checker primitives for more information on this and the other Timing errors.

Timing error #158: Setup/Hold time violation

This error is generated by a Timing Verifier Setup Hold checker primitive and indicates a timing error in the design. See the documentation section on checker primitives for more information on this and the other Timing errors.

Timing error #159: Minimum pulse width timing violation

This error is generated by a Timing Verifier Minimum Pulse Width checker primitive and indicates a timing error in the design. See the documentation section on checker primitives for more information on this and the other Timing errors.

Timing error #160: Delay is greater than CLOCK_PERIOD

Before doing the actual verification, the Timing Verifier checks that all delays are less than the CLOCK_PERIOD. All delays that are greater than the CLOCK_PERIOD are flagged with this warning error message. (This is a feature only available in release 7.5 and later releases). As always, greater delays are used modulo the CLOCK_PERIOD during the verification. That is, with CLOCK_PERIOD set to 102 ns, a delay of 104.2 ns will use 2.2ns (104.2 mod 102) as a delay.

Syntax error #161: Too many entries given in load coefficient table

This error is generated when more than 100 entries are given for a LOAD_COEFF table. See the section on non-linear delay estimation for more information.

Error #162: Unused.

Syntax error #163: Illegal latch directive

The legal values for the LATCH_ERR_MODEL directive are: OPEN, CLOSED, and CONSERVATIVE. Specifying any other value will generate this error.

Syntax error #164: Reserved (debug).

Runtime error #165: Multiple evaluation directives on primitive

The use of evaluation directives is restricted to only ONE pin of a primitive. See the section Evaluation Directives For Clock Tuning for more information.

Timing error #166: Input changing while clock is asserted

This error indicates that the conditions required by an "A" evaluation directive have not been met by the circuit. While the clock input to an AND gate or an OR gate is asserted, the other input is not stable. See the section Evaluation Directives For Clock Gating for more information.

Syntax error #167: Reserved.

Runtime error #168: Wire-tie error - mixed and/or

This error is generated by the Verifier when some illegal combination of wire-and and wire-or logic is used on a signal. See the sections Dot Gate Primitives, Signal Strengths in the Timing Verifier, and DOT_TYPE directive for more information.

Syntax error #169: Illegal value given

This error is generated if a value given in the case file is illegal. Legal values are 0, 1, S, and clock assertions. See the section on Case Analysis for more information.

Syntax error #170: Invalid casefile syntax

This error is generated if the proper case file syntax if not used. In particular, signal names and values must be enclosed in single quotes (').

Syntax error #171: Case signal not used in network

This error is generated if a signal found in the case file is not found in the design. This type of erroneous case specification is ignored.

Syntax error #172: Reserved.

Syntax error #173: Expected ; or ,

This error is generated if a case specification is not ended properly. Ending a specification with a comma, (,) means that more specifications will follow for the current case. Ending a specification with a semicolon (;) means that the current specification is the last for the current case. See the Case Analysis section for the exact syntax and more information.

Syntax error #174: Signal not found - delay spec ignored

This error is generated if a signal is found in the wire delay feedback file (delay.dat) that is not found in the design. The signal is printed out, and the specification is ignored. Usually this indicates a spelling error.

Syntax error #175: Signal does not drive pin, delay ignored

This error is generated if a signal is found in the wire delay feedback file (delay.dat) with an incorrect path name. See the section Calculated Wire Delays for the exact syntax required and for more information.

Runtime error #176: Dotted signal name too long

This error is generated as a warning when a signal name and it path name are too long to be concatenated to form a dotted signal name. The limit on signal name lengths is 255 characters. The Verifier picks an intelligent substitute and prints out that substitute name.

Runtime error #177: Too many outputs are wire-tied together

Currently, only 1000 primitive outputs may be wire-tied together. If this error occurs, and there is not an error in the drawing, please report the problem and the size will be increased.

Syntax error #178: Error in timing assertion

This error is generated if any of a number of signal name syntax errors is detected. The exact error is printed out with the signal in question. If possible, an intelligent substitute or default is used.

Syntax error #179: Illegal List option

This error refers to the LIST directive which has many possible options of the form <option> and NO<option>. See the LIST directive documentation in the Timing Verifier Directives Summary.

Runtime error #180: No option file or TIME_DIRECTIVES block

This error is generated when the Timing Verifier cannot find any directives. The file verifier.cmd does not exist in the current directory AND there is no TIME_DIRECTIVES block in the expansion file. Either create a file called verifier.cmd and put any desired directives from the Timing Verifier Directives Summary into it or add a TIME_DIRECTIVES body from the Standard library (with the desired directives) to your schematic.

Syntax error #181: Verification aborted-expansion file errors

This error is generated when more errors are detected while reading in the expansion file than the value given to the MAX_EXP_ERRORS directive. See the Timing Verifier Directives Summary for information on the MAX_EXP_ERRORS directive.

Runtime error #182: Cannot open file for write

This error is generated by the Verifier when it cannot open the monitor (screen output) file. Check for space on the disk, no write access to the current directory, etc.

Runtime error #183: Reserved.

Runtime error #184: Verification aborted-too many input errors

This error is generated when more errors are detected while reading in the input files (not just the expansion file) than the value given in the MAX_ERRORS directive. See the Timing Verifier Directives Summary for information on the MAX_ERRORS directive.

Runtime error #185: Illegal evaluation modes

This is an internal error that should not occur. If it does, please note the two evaluation modes, save the list and log files, and notify a Valid Service Engineer.

Runtime error #186: Wire table already defined, redefining

This error in generated when more than one wire estimate list is given for a single family. In the directives file or the TIME_DIRECTIVES block, more than one WIRE_ESTIMATE directive was found with either no family specification, or the same family specification.

Runtime error #187: Undefined wire delay table given

This error is generated when a FAMILY specification is given for a primitive, but a wire estimate list for that family is not in the directives file or in the TIME_DIRECTIVES block of the drawing.

Runtime error #188: Undefined load coefficient table given

This error is generated when a FAMILY specification is given for a primitive, but a load coefficient list for that family is not in the directives file or in the TIME_DIRECTIVES block of the drawing.

Runtime error #189: Load table already defined, redefining

This error in generated when more than one load coefficient list is given for a single family. In the directives file or the TIME_DIRECTIVES block, more than one LOAD_COEFF directive was found with either no family specification, or the same family specification.

Error #190: Unused.

Error #191: Unused.

Error #192: Unused.

Runtime error #193: No name string in print_signal_formatted

This error is an internal error that should not occur. If it does, and is repeatable, please save the input and output files and report the problem to Valid.

Runtime error #194: Invalid margin in print_signal_formatted

This error is an internal error that should not occur. If it does, and is repeatable, please save the input and output files and report the problem to Valid.

Errors #195 through #200: Unused.

6.33 GLOSSARY

The following is a short glossary of the terms used in the SCALD Timing Verifier documentation. It is not intended to be comprehensive since most of the terms are described in detail elsewhere within this chapter. A short description of each term is given along with a reference to the chapter or section where it is more fully described.

6.34 TERMS

ASSERTION

A timing assertion is a statement about the behavior of a signal during the execution of the system of which it is a part. Assertions are named or general signal properties that specify the stable/changing, or in the case of clock signals (see below) the zero/one behavior of a signal. (See Signal Assertions in this chapter.)

BASE SIGNAL

The SCALD language allows the user to associate several different signal names with a single electrical node. The base signal is the name which the verifier uses to refer to a signal node. It will be one of the (possibly many) names that the user has associated with the node. (See Chapter 4, "SCALD III Language".)

CASE

Sometimes circuits have timing behavior that depends on the actual logic values of certain data signals (not just whether they are stable or changing). The Timing Verifier supports case analysis to check these systems. A particular assignment of zero/one values to a set of signals in a design for a verifier run is called a CASE. (See Timing Verifier Case Analysis in this chapter.)

DIRECTIVE

See Evaluation Directive.

EVALUATION DIRECTIVE

An evaluation directive is a general signal property that directs the Timing Verifier to treat the signal it is associated with specially. Evaluation directives are provided to verify designs that contain tuned or gated clock signals. (See Timing Verifier Directives in this chapter.)

CLOCK

A clock is a signal with a clock assertion. Signals with clock assertions are special in two ways. First, assertions specify zero/one (not stable/changing) behavior on clocks. Second, evaluation directives may be applied to clocks. (See Timing Verifier Signal Syntax.)

CLOCK SKEW

The Timing Verifier assumes that clocks have some kind of skew over the entire system. Clock skew is a single value (fixed point number) that denotes a symmetric uncertainty in the time of occurrence of an edge in a clock signal. The Timing Verifier allows two skews to be specified in a system, CLOCK SKEW and PRECision CLOCK SKEW. These skews differ only in the details of the clock assertion syntax (see Timing Verifier Signal Syntax).

IDENTIFIER

An identifier is a name made up of letters (A-Z), digits (0-9), and underscores (_) starting with a letter. The case (whether upper or lower) makes no difference; the compiler upper-shifts all letters on input. An identifier must start with a letter so that the compiler can tell the difference between a name and a number (since numbers may have alphabetic characters following them to make them easier to read, for instance, SIZE = 3Bits). All the names the compiler expects (except for signals and macros) must be identifiers. These names include those for text macros, structures, properties, and parameters.

PATHNAME

The path from the root macro down the tree to some other macro is given a name to distinguish it from the other paths in the tree. The path name is crucially important in identifying macro and signal instances and guarantees that signal names will be unique. See the compiler overview for a further description.

PRECISION CLOCK SKEW

(See CLOCK SKEW above.)

SCALAR

A scalar is a single bit signal that is not a part of a vector signal. Normally, a signal has labeled bits. That is, its bits are given numbers to identify them as, for example:

FOO<2> SNARF<0..4> WHOOPS<56, 59>

In each of these signals, the bits have been numbered to distinguish them from other bits that are part of the signals FOO, SNARF, and WHOOPS. A scalar signal, on the other hand, has only one bit. There is no need to give it a number to distinguish it; it is always unique. Scalar signals never have bit subscripts since the subscript's only function is to number the bits of the signal.

TEXT MACRO

A text macro is a symbolic reference to some string of characters that replaces the text macro name wherever it is used. Text macros are defined in a DEFINE body within a macro drawing. Text macros are used to provide a shorthand notation for commonly used items (such as signal properties) or to allow easy changes of fundamental values (by naming the value, the value can be easily changed). Text macros are described in Chapter 4.

TIMING VIOLATION

A timing violation is a point in time in a signal's value history where the setup time, hold time, or pulse width requirements of some part driven by the signal are not met or the signal's value is inconsistent with its timing assertion.

VALUE HISTORY

The value history of a signal is the ordered list of values (0, 1, S, R, F, C, U and Z) and durations of those values that a signal assumes during system operation.

VECTOR

A vector is a signal representing more than one bit (such as a bus) or representing a portion of a base signal that represents more than one bit. A vector is always given a bit subscript; the presence of a subscript makes a signal a vector. When a signal refers to a portion of a base signal (signal definition) that is a vector, it must be specified as a vector even if the signal refers to only one bit of the vector. The reason for this is that the bits of a multibit signal are referred to by number. When a signal refers to a portion of a signal that contains many bits, the bits of interest must be specified by number. See also SCALAR and "Signal Name Syntax" in Chapter 4.