Digital SimCode Reference

Summary

This comprehensive reference describes the Digital SimCode™ language - used to specify simulation models for digital devices. The reference includes in-depth descriptions for each of the constituent functions for the language.

Due to the complexity of digital devices it is generally not practical to simulate them using standard, non-event-driven SPICE instructions. For this reason Altium Designer's Mixed-Signal Circuit Simulator includes a special descriptive language that allows digital devices to be simulated using an extended version of the event-driven XSPICE. This language - used to model digital devices - is called Digital SimCode™.

Tip:

Digital SimCode is a proprietary language - devices created with it are not compatible with other simulators, nor are digital components created for other simulators compatible with Altium Designer's Mixed-Signal Circuit Simulator.

SimCode is a "C like" description language. You use it to define the characteristics and behavior of the device you are modeling. It includes functions to define parameters such as propagation delays, load characteristics, strengths, and so on. The device behavior is defined using truth tables, math functions and conditional control statements, such as **IF .. THEN** statements.
SimCode source files are written as plain ASCII text files and saved with the .txt extension.

> For information on linking a SimCode model to a schematic component, refer to the Creating and Linking a Digital SimCode Model application note.

Beginning a SimCode Model Definition

A SimCode device definition begins with a # xxxx source statement. This statement has the form:
< *func name* > source
where < *func name* > is the name of the simulation function used to identify the model definition block. The model definition block ends with the **EXIT** statement. Therefore, a SimCode model for a particular device takes the general form:
MyDevice source
...
...
...
EXIT;

In the above code fragment, the **.MODEL** statement contained in the .MDL file referenced by the schematic part symbol would call the function MyDevice in order to simulate the device.
A single SimCode text file can contain any number of model definitions.

SimCode Statement Termination Character

The SimCode language uses the semicolon character " ; " to mark the end of a SimCode statement.
In SimCode, a statement may occupy a single line of code, or take up several lines. The termination character marks the termination of the complete SimCode statement and is placed immediately after the last clause of the statement, as shown in the following examples:

```
DELAY Q1 Q2 Q3 Q4 = 10n;
DELAY Q QN =
  CASE (TRAN_LH) : tplh_val
  CASE (TRAN_HL) : tphl_val
END;

data = (E0_1 && (CHANGED(D0) \|\| CHANGED(D1)));
DELAY Q1 Q0 =
  CASE (data && TRAN_LH) : tplh_D_Q
  CASE (data && TRAN_HL) : tphl_D_Q
  CASE (TRAN_LH) : tplh_E_Q
  CASE (TRAN_HL) : tphl_E_Q
END;
```

Including Comments in SimCode Files

You may include comments in a SimCode file by preceding the comments with two "slash" characters //. Everything after the slash characters is ignored. Comments can appear as complete lines, or after a SimCode line, as shown in the following examples:

```
EVENT = (present_time \\+ 1e-6); //return in 1us
DELAY Q QN =
// Case 1
  CASE (TRAN_LH) : tplh_val
//Case 2
  CASE (TRAN_HL) : tphl_val
END;
```

Comments should be placed in SimCode models to make the code more readable and to aid in debugging.

SimCode Language Definition

The following items make up the Digital SimCode language, which is then used to construct a digital device simulation model. Each individual item topic gives full syntax descriptions and examples of usage. In describing the language syntax, the following style is used:

< >	value/variable/pin/expression
[ ]	optional parameter
{ }|{ }	selections (you must choose ONE of these parameters)

Device Definition Functions

Functions used to define device pins, etc.
INPUTS
OUTPUTS
INTEGERS
REALS
PWR_GND_PINS
IO_PAIRS

Device Setup Functions

Functions used to set certain characteristics of the device pins:
VIL_VIH_VALUE
VIL_VIH_PERCENT
VOL_VOH_MIN

Device Test Functions

Functions used to test for any device setup violations which may occur in the circuit:
SUPPLY_MIN_MAX
RECOVER
SETUP_HOLD
WIDTH
FREQUENCY(FMAX)

Output Pin Functions

Functions used to program the output pins of a device:

STATE
STATE_BIT
LEVEL
STRENGTH
TABLE
EXT_TABLE
LOAD
DRIVE
DELAY
NO_CHANGE
EVENT

Expression Operators and Functions

Used in expressions to manipulate data and to make comparisons which control program flow:
OPERATORS
MATH FUNCTIONS
PARAM_SET
PWL_TABLE
SELECT_VALUE
MIN_TYP_MAX
NUMBER
VALUE
CHANGE_TIME
WIDTH_TIME
INSTANCE
CHANGED_xx
READ_DATA

Program Control

Functions used to control the flow of the program:
# xxxx source
IF ... THEN
WHILE ... DO
GOTO
GOSUB
RETURN
EXIT

Text Output

Functions used to display messages during simulation and debugging:
PROMPT
MESSAGE

Debug

Functions used to trace through the execution of the SimCode for debugging purposes:
STEP_ON
STEP_OFF

xxxx source

Syntax

< *func name* > source

Description

Identifies the beginning of the SimCode source function. This statement identifies the SimCode function so that it can be called when it is time to

simulate this device. It must be the first statement of each Digital SimCode device function.

Parameters

< *func name* >		Name of the SimCode function.

Examples

```
//================================
\# MyDevice source
//================================
INPUTS VCC, GND, IN1, IN2;
OUTPUTS VCC_LD, IN1_LD, IN2_LD, OUT;
.
.
.
EXIT;
```

Notes

The Simulator has the ability to read either uncompiled SimCode source models, or compiled SimCode models. The keyword **source** identifies this as a SimCode source model to the Simulator, which automatically compiles the model, on-the-fly, when the simulation is run.
Should you wish for the compiled model information to be written to file, ensure that the **Create compiled SimCode output file** option is enabled in the *Simulation Preferences* dialog. This dialog is accessed by clicking on the **Preferences** button in the *Analyses Setup* dialog (**Design » Simulate » Mixed Sim**). The compiled output is written to an ASCII text file - SimCodeFunctionName simlist.txt - which, by default, is stored in the same directory as the ASCII SimCode source model itself (\Library\Sim). The compiled model information can be extracted to create a compiled SimCode model file (*.SCB).

CHANGE_TIME

Syntax

CHANGE_TIME (< *pin* >)

Description

This function returns a real value that indicates the last time the specified input or output pin changed states.

Parameters

< *pin* >		Input or output pin name.

Examples

T1 = (CHANGE_TIME(INA));

CHANGED_xx

Syntax

CHANGED_xx (< *pin* > [{<}|{<=}|{>}|{>=} < *var/time/value* >])

Description

This function is used to determine if the specified pin has changed state.
The _**xx** that follows the keyword **CHANGED** can be eliminated (to indicate any type of change) or the **xx** can be set to:
LH, LX, HL, HX, XL, XH, LZ, ZL, ZH, ZX, HZ or XZ
to indicate a specific type of change. The optional compare operator (<, <=, >, >=) and < *var/time/value* > would be included to check for a more specific change. If they are not included, the function will return 1 if the pin has changed at the current simulation step.

Parameters

< *pin* >		Input or output pin name.
< *var/time/value* >		Item to which < *pin* > is compared.

Examples

IF (CHANGED_LH(CLK)) THEN ...
IF (CHANGED(DATA < 10n)) THEN ...

DELAY

Syntax 1

DELAY < *output* > [< *output* > ...] = < *delay* >;

Syntax 2

DELAY < *output* > [< *output* > ...] =
CASE (< *conditional exp* >) : < *delay* >
CASE (< *conditional exp* >) : < *delay* >
[**CASE** (< *conditional exp* >) : < *delay* > ...]
END ;

Description

Sets propagation delay to specified outputs. The **DELAY** command is executed once for each pin listed and posts a propagation delay for each pin that has changed its level. The **CASE** option allows more than one < *delay* > to be specified. The < *conditional exp* > then determines which < *delay* > will be used. If a delay is set for a pin that has not changed then the pin will be flagged as NO-CHANGE and the delay will *not* be posted. The < *delay* > can be a real constant, a real variable or a real expression.

Parameters

< *output* >		Name of/variable index to the output pin.
< *conditional exp* >		Conditional expression that determines which delay is used.
< *delay* >		Propagation delay time to the output pin.

Examples

```
DELAY Q1 Q2 Q3 Q4 = 10n;
  DELAY Q QN =
  CASE (TRAN_LH) : tplh_val
  CASE (TRAN_HL) : tphl_val
END;

data = (E0_1 && (CHANGED(D0) \|\| CHANGED(D1)));
DELAY Q1 Q0 =
  CASE (data && TRAN_LH) : tplh_D_Q
  CASE (data && TRAN_HL) : tphl_D_Q
  CASE (TRAN_LH) : tplh_E_Q
  CASE (TRAN_HL) : tphl_E_Q
END;
```

In this example, if data is nonzero and Q1 is changing from High to Low, the tphl_D_Q delay will be posted for Q1. Then, if Q0 is changing from Low to High, the tplh_D_Q delay will be posted for Q0.

Notes

The **DELAY** command must be executed exactly once for each output pin, that is, for each pin declared in the **OUTPUTS** statement which is NOT listed in the **LOAD** or **NO_CHANGE** statements. The order in which the delays are set is based on the order in which these pins are listed in the **DELAY** command (i.e. first pin listed is set first). Each < *conditional exp* > is evaluated in the order it is listed until one expression evaluates TRUE. When this occurs, the < *delay* > value associated with the TRUE expression is posted for the output being set. When using the **CASE** option, at least one < *conditional exp* > should evaluate as TRUE for each output pin listed. If no < *conditional exp* > evaluates to TRUE, the < *delay* > associated with the last **CASE** statement is posted.
In addition to the standard expression functions, the following terms apply *only* to the output pin being set and can be used in the < *conditional exp* >:

- TRAN_LH - low-to-high transition
- TRAN_LX - low-to-other transition
- TRAN_HL - high-to-low transition
- TRAN_HX - high-to-other transition
- TRAN_HZ - high-to-tristate transition
- TRAN_XL - other-to-low transition
- TRAN_XH - other-to-high transition
- TRAN_LZ - low-to-tristate transition
- TRAN_ZL - tristate-to-low transition
- TRAN_ZH - tristate-to-high transition
- TRAN_ZX - tristate-to-other transition
- TRAN_XZ - other-to-tristate transition
- TRAN_XX - other-to-different transition.

If the < *delay* > value is less than or equal to 0.0 a run-time error message will be displayed. Output pins can be specified by using the output pin name or by an integer variable that contains the index of an output pin. Pin names and variables cannot be mixed in the same **DELAY** statement. References to outputs must be either all pin names or all variable names.

DRIVE

Syntax

DRIVE < *output* > [< *output* > ...] =
(v0=< *value* > v1=< *value* > ttlh=< *value* > tthl=< *value* >);

Description

Declares drive characteristics of output pins. The **DRIVE** command is used to declare the output pin's drive characteristics. When the output is set to a LOW state, the output pin is connected to voltage value v0 through resistance rol_param. When the output is set to a HIGH state, the output pin is connected to voltage value v1 through resistance roh_param. The low-to-high transition time is set by ttlh and the high-to-low transition time is set by tthl.

Parameters

< *output* >		Name of or variable index to the output pin.
v0< *value* >		VOL for the output pin. < *value* > can be a real value or variable.
v1< *value* >		VOH for the output pin. < *value* > can be a real value or variable.
ttlh< *value* >		Low-to-high transition time for the output pin. < *value* > can be a real value or variable.
tthl< *value* >		High-to-low transition time for the output pin. < *value* > can be a real value or variable.

Examples

rol_param = (MIN_TYP_MAX(drv_param: 62.5, 43.75, NULL);
roh_param = (MIN_TYP_MAX(drv_param: 262.5, NULL, 52.5);
DRIVE Q QN = (v0=vol_param,v1=voh_param,ttlh=ttlh_val,
tthl=tthl_val);

Notes

Pin names and variables cannot be mixed in the same **DRIVE** statement. References to outputs must be either all pin names or all variable names.
The values used for rol_param should be derived using the data book specs for VOL. This value represents the total saturation resistance of the pull-down structure of the device's output. A standard LS output in the LOW state, for example, sinking 8mA will not exceed 0.5V, typically closer to 0.35V. Therefore:
for typ LOW state drive:
rol_param = VOLtyp / IOLmax
rol_param = 0.35V / 8mA
rol_param = 43.75 ohms.
for min LOW state drive:
rol_param = VOLmax / IOLmax
rol_param = 0.5V / 8mA
rol_param = 62.5 ohms.
The values used for roh_param should be derived using the data book specs for IOS, if available. This value represents the total saturation resistance of the pull-up structure of the device's output. A standard LS output in the HIGH state with the output shorted to ground and Vcc=5.25V will source at least 20mA but not more than 100mA. Therefore:
for min HIGH state drive:
roh_param = VCCmax / IOSmin
roh_param = 5.25V / 20mA
roh_param = 262.5 ohms.
for max HIGH state drive:
roh_param = VCCmax / IOSmax
roh_param = 5.25V / 100mA
roh_param = 52.5 ohms.

EVENT

Syntax

EVENT = ({< *time* >}|{< *expression* >})

Description

Causes a digital event to be posted. In most cases a digital event is posted when one or more INPUT pins for a SimCode model changes state. When the event is processed, the SimCode for the specified event is called and run. This instruction allows a SimCode model to post a digital event at a specified < *time* >. If the specified **EVENT** time is greater than the simulation time (indicated by present_time), then a digital event will be posted. If more than one **EVENT** is posted in a single call to a SimCode model, only the longest **EVENT** < *time* > will be used. This function allows the creation of one-shots and other similar device models.

Parameters

< *time* >		Time at which event should occur.
< *expression* >		Expression indicating time at which event should occur.

Examples

EVENT = (present_time
+ 1e-6); //return in 1us

Notes

If a digital event for a specific SimCode model occurs before an **EVENT** < *time* > posted by that SimCode, the **EVENT** < *time* > must be posted again. For example, if:

- the present simulation time is 1us,
- a SimCode model sets **EVENT** = 2us and
- an INPUT pin in the SimCode model changes state at 1.5us,

then the 2us event must be posted again.

EXIT

Syntax

EXIT

Description

Terminates SimCode execution.

Notes

This is the last line of a SimCode model, but it may also be placed at other locations to abort execution of remaining SimCode.
EXT_TABLE

Syntax

EXT_TABLE < *line* >
< *input pin* > [< *input pin* > ...] < *output pin* > [< *output pin* > ...]
< *input state* > [< *input state* > ...]
< *output state* > [< *output state* > ...];

Description

Sets output logic states based on extended truth table. The **EXT_TABLE** statement is an extended truth table function used to set the level and strength of the specified outputs. **Valid input states are:**

0	low (input voltage is <= vil_param)
1	high (input voltage is >= vih_param)
^	low-to-high-transition

v	high-to-low-transition
X	don't care what input voltage is

Valid output states are:

L	ZERO (set output level to vol_param).
H	ONE (set output level to voh_param).
Z	UNKNOWN (set output level to v3s_param).

It also allows INPUT and/or OUTPUT pin names with optional prefixes to specify the output states. Prefixes are:

	State is the previous state.
~	State is the inverse of the state.
-~	State is the inverse of the previous state.

Output state letters can be followed by a colon and a letter to indicate strength:

s	STRONG (set output to rol_param for L and roh_param for H).
z	HI_IMPEDANCE (set output to r3s_param).

If a strength character is not specified after an output state then STRONG will be used for L and H states and HI_IMPEDANCE will be used for Z states.

Parameters

< *line* >		Variable into which the line number used in the table is placed
< *input pin* >		Name of the input pin
< *output pin* >		Name of the output pin
< *input state* >		State of the individual inputs
< *output state* >		State of the individual outputs based on input conditions

Examples

```
EXT_TABLE tblIndex
PRE CLR CLK DATA Q QN
0 1 X X H L
1 0 X X L H
0 0 X X H H
1 1 \\^ X DATA \~DATA
1 1 X X Q \~Q;
```

This example is representative of 1/2 of a 7474 D type flip-flop. If input pins PRE, CLR, and DATA are all high (>= vih_param) and CLK has a low-to-high transition, Q is set to high (voh_param) and STRONG (roh_param), QN is set to low (vol_param) and STRONG (rol_param) and tblIndex is set to 4.

Notes

Each row is tested sequentially from top to bottom until the input conditions are met. The outputs are set for the first row to meet the input conditions. < *line* > is set to the line number in the table that was used. If no match was made then < *line* > is set to 0. Pin names used to specify output states do not need to be in the table heading. Unlike the **TABLE** statement, input variables are not allowed.

FREQUENCY (FMAX)

Syntax

FREQUENCY (< *input* > [< *input* >...] MIN=< *frequency* > MAX=< *frequency* > ["< *message* >"])

Description

Tests inputs for minimum and maximum frequency violation. The **FREQUENCY** function compares the < *input* > period (the time from one low-to-high edge to the next low-to-high edge) with the reciprocal of the specified < *frequency* > (1/freq). If the time period for the < *input* > is smaller than the reciprocal of the specified MAX frequency or the time period for the < *input* > is greater than the reciprocal of the specified MIN frequency, then a WARNING will be displayed. An optional < *message* > string can be included in the **FREQUENCY** statement which will be output if a WARNING is displayed.

Parameters

< *input* >		Name of or variable index to the input pin under test.
MIN=< *frequency* >		Minimum frequency allowed on the pin under test.
MAX=< *frequency* >		Maximum frequency allowed on the pin under test.
< *message* >		Text string that will be displayed if a warning occurs.

Examples

FREQUENCY(CLK MAX=10MEG "CLK"); //check fmax only

Notes

Data book specifications should be used with this function. Pin and variable names can be mixed in the same **FREQUENCY** statement. Only the first **FREQUENCY** failure for each pin listed will be reported.

GOSUB

Syntax

GOSUB < *label* >;

Description

Jumps to a subroutine in the SimCode. The **GOSUB** instruction is used to perform non-sequential execution of the SimCode. However, unlike the

GOTO statement, SimCode execution will continue from the instruction following the **GOSUB** instruction, when a **RETURN** instruction is encountered.

Parameters

< *label* >		Location in SimCode where program flow resumes.

Examples

```
GOSUB Shift_Left;
.
.
Exit;
Shift_Left:
.
.
RETURN:
```

GOTO

Syntax

GOTO < *label* >;

Description

Jumps to a new location in the SimCode. The **GOTO** instruction is used to perform non-sequential execution of the SimCode.

Parameters

< *label* >		Location in SimCode where program flow resumes.

Examples

```
GOTO Shutdown;
.
.
Shutdown:
.
.
Exit;
```

Notes

Program flow resumes from the location where < *label* >: appears in the SimCode. < *label* > must begin with an alpha character, followed by any number of alpha-numeric characters or the underscore (
_) character. Where < *label* > appears in the code, it must be followed immediately by a colon (:).

IF ... THEN

Syntax 1

IF (< *expression* >) **THEN BEGIN** ... [**ELSE** ...] **END** ;

Syntax 2

IF (< *expression* >) **THEN GOTO** < *label* >;

Description

Conditionally controls flow through the SimCode.
The **IF ... THEN** statement is used to control the flow of the program, based on whether < *expression* > evaluates to true or false. Multiple **IF ... THEN** statements may be nested.

Parameters

< *expression* >		Any expression that can be evaluated as true or false.
< *label* >		Location in SimCode where program flow resumes.

Examples

```
IF (EN) THEN
BEGIN
  STATE Q0 = UNKNOWN;
ELSE
IF (IN2) THEN
BEGIN
  STATE Y2 = ONE;
ELSE
  STATE Y2 = ZERO;
END;
END;
IF (x = \-2) THEN GOTO Do_Neg2;
...
...
Do_Neg2:
...
...
```

Notes

When the **BEGIN . . . ELSE ... END** form of this statement is used and < *expression* > evaluates to true, program flow resumes from the **BEGIN** statement and skips any optional SimCode between the **ELSE** and **END** statements. If < *expression* > evaluates to false, program flow resumes from the optional **ELSE** statement if it exists or after the **END** statement if it does not exist.
When the **GOTO** form of this statement is used and < *expression* > evaluates to true, program flow resumes from the location where < *label* >: appears in the SimCode. < *label* > must begin with an alpha character, followed by any number of alpha-numeric characters or the underscore (_) character. Where < *label* > appears in the code, it must be followed immediately by a colon (:).

INPUTS

Syntax

INPUTS < *input pin* >[, < *input pin* >, ...];

Description

Declares input pins (pins that monitor the circuit). The **INPUTS** data type is used to define the pins which monitor stimulus external to the device. These generally include input, i/o, power and ground pins.

Parameters

< *input pin* >		Name of the input pin.

Examples

INPUTS VCC, GND, PRE, DATA, CLK, CLR;

Notes

Input pin names must begin with a letter and be defined before they are used.

INSTANCE

Syntax

INSTANCE ("< *instance name* >")

Description

Checks if this is the specified device instance. The **INSTANCE** function returns 1 if the present instance of the SimCode device matches the < *instance name* > specified. Otherwise it returns 0.

Parameters

< *instance name* >		Text string indicating instance name.

Examples

```
IF (INSTANCE("AU23")) THEN
BEGIN
    MESSAGE("U23-Q0 = %d", Q0);
END;
```

Notes

A circuit may contain more than one of any given device. During simulation it may be important to know if the device being simulated at this moment is the one you are interested in. This would allow you, for example, to print messages for one specific NAND gate without having to wade through messages for all the other NAND gates as well. The instance name is the device Designator preceded by its SPICE Prefix Character (the letter A).

INTEGERS

Syntax

INTEGERS < *var* >[, < *var* >, ...];

Description

The **INTEGERS** data type is used to define integer variables and arrays.

Parameters

< *var* >		Name of the variable.

Examples

INTEGERS tblIndex, count, data[64];

Notes

Integer variables and arrays must begin with a letter and be defined before they are used. Integer arrays are defined by following the array name with a left bracket ([ ), an integer number which defines the size of the array, and a right bracket ( ]). Integer arrays can be set and/or used in expressions.
The following are reserved SimCode integer variables which do not need to be declared:

Variable	Use	Digital Model Parameter	SPICE Option
tp_param	tplh/hl index	Propagation	TPMNTYMX
tt_param	ttlh/hl index	N/A	TTMNTYMX
ld_param	LOAD index	Loading	LDMNTYMX
drv_param	DRIVE index	Drive	DRVMNTYMX
i_param	ICC index	Current	IMNTYMX
user_param	USER index	User Defined Parameter	USERMNTYMX
warn_param	Warning messages	WARN	SIMWARN
init_sim	Once during SimCode initialization	N/A	N/A
tran_pin	TRAN_xx pin index	N/A	N/A

The first six variables in this list are expected to have a value of 1, 2 or 3. These values represent an index into the min/typ/max arrays:

Value	Represents
1	Index to minimum value.
2	Index to typical value.
3	Index to maximum value.

The **Digital Model Parameter** can be set independently for each digital device using the respective parameters found on the **Parameters** tab of the *Sim Model* dialog. This dialog is accessed by double-clicking on the entry for the simulation model link in the **Models** region of the associated *Component Properties* dialog.
If a **SPICE Option** parameter is set on the **Advanced Options** page of the *Analyses Setup* dialog, that setting will globally override the **Digital Model Parameter** settings for all digital devices. If the variable is set explicitly in the SimCode, that setting will override all other settings.
warn_param can be set to any positive value to conditionally display warning messages for the device. Different levels of warning could be

created by the device programmer, accessed by entering different positive values. The value of init_sim is set 1 during SimCode initialization, otherwise it is set to 0. The value of tran_pin is set to the index of the pin being set during a **DELAY CASE** statement. This index is used to determine which pin the TRAN_xx instruction is applied to.

IO_PAIRS

Syntax

IO_PAIRS (< *ipin :opin* >[, < *ipin :opin* >, ...]);

Description

Declares input/output pin associations for input loading. The **IO_PAIRS** statement defines which of the **INPUTS** pins are associated with which of the **OUTPUTS** pins. This association is used by the **LOAD** statement.

Parameters

< *ipin :opin* >		Pin names of associated input and output pins.

Examples

IO_PAIRS (IN1:IN1_LD, IN2:IN2_LD);
In this example, IN1 and IN2 are INPUTS and IN1_LD and IN2_LD are OUTPUTS. IN1 and IN1_LD both refer to the same physical pin on the device.

Notes

Each physical input pin on a device consists of both an ipin and an opin in SimCode. The opin is required to provide input loading characteristics. This statement can only be used once in the SimCode. Power pins are not listed in the **IO_PAIRS** statement.

LEVEL

Syntax 1

LEVEL < *output* > [< *output* > ...] = (< *expression* >);

Syntax 2

LEVEL < *output* > [< *output* > ...] = {ZERO}|{ONE}|{UNKNOWN};

Description

Sets the level of the output state. The state of an output pin is determined by its level and its strength. Use the **LEVEL** command to set the level of one or more output pins.

< *expression* >	**State**	**Level**
<= vol_param	ZERO	vol_param
>= voh_param	ONE	voh_param

other	UNKNOWN	v3s_param

Parameters

< *output* >		Name of or variable index to the output.
< *expression* >		Any expression compared to VOL or VOH.

Examples

```
LEVEL Q = ONE;
LEVEL Q1 Q2 Q3 Q4 = ZERO;
LEVEL OUT = ((1+2)/3);
code{}
In the last example, OUT will be:
ZERO if vol_param > 1
UNKNOWN if vol_param < 1 and voh_param > 1
ONE if voh_param < 1
```

Notes

Output pins can be specified by using the output pin name or by an integer variable that contains the INDEX of an output pin. Pin and variable names cannot be mixed in the same **LEVEL** statement. References to outputs must be either all pin names or all variable names.

LOAD

Syntax

LOAD < *output* > [< *output* > ...] =
v0=< *value* > r0=< *value* > [v1=< *value* > r1=< *value* >] [io=< *value* >] t=< *value* >);

Description

Declares loading characteristics of input pins. The **LOAD** command is typically used with input or power pins to provide loading for the driving circuit. Since only output pins can provide a load, each input must have a corresponding output. These are assigned using the **IO_PAIRS** statement.
If different loads are required for different inputs, multiple **LOAD** statements may be used. Power pins should be placed in a separate **LOAD** statement which does not include the v1/r1 load or io. Power pins are not included in the **IO_PAIRS** statement. The **IO_PAIRS** statement must be entered before any **LOAD** statements that contain io.

Parameters

< *output* >		Name of or variable index to the output pin.
v0		Load voltage for HIGH state input. < *value* > can be a real constant or a real variable.
r0		Load resistance for HIGH state input. < *value* > can be a real constant or a real variable.
v1		Load voltage for LOW state input. < *value* > can be a real constant or a real variable.
r1		Load resistance for LOW state input. < *value* > can be a real constant or a real variable.

io		Off-state load resistance for unused load. < *value* > must be a real constant
t		Time delay before the load will be applied. < *value* > must be a real constant

Examples

```
r0_val = (MIN_TYP_MAX(ld_param: NULL, NULL, 125k);
r1_val = (MIN_TYP_MAX(ld_param: NULL, NULL, 10.5k);
ricc_val = (MIN_TYP_MAX(ld_param: NULL, 833, 525);
LOAD PRE_LD DATA_LD CLK_LD CLR_LD = (v0=vol_param, r0=r0_val,
v1=voh_param, r1=r1_val, io=1e9, t=1p);
LOAD VCC_LD = (v0=gnd_param, r0=ricc_val, t=1p);
```

Notes

An input load consists of a voltage and a resistance (v0/r0 or v1/r1). The voltage level of the incoming signal determines which load will be used. If the voltage level goes below VIL and remains below VIH, then the input is considered to be in the LOW state and the v1/r1 is applied. If the voltage level goes above VIH and remains above VIL, then the input is considered to be in the HIGH state and the v0/r0 is applied. io is the input state off resistance. The unused load is essentially removed from the circuit by changing its r value to the value specified for io.
The values for v0, r0, v1 and r1 can be either real constants or real variables. The values for io and t must be real constants. Pin names and pin variables cannot be mixed in the same **LOAD** statement. References to outputs must be either all pin names or all variable names.
For input pins, the values used for r0 should be derived using the data book specs for IIH. A standard LS input, for example, will sink a maximum of 20uA at Vin=2.7V. Therefore, if vol_param = 0.2V, then:
for max HIGH state load:
r0 = (Vin
- vol_param) / IIHmax
r0 = (2.7V
- 0.2V) / 20uA
r0 = 125k ohms
The values used for r1 should be derived using the data book specs for IIL. A standard LS input, for example, will source a maximum of 400uA at Vin=0.4V. Therefore, if voh_param = 4.6V then:
for max LOW state load:
r1 = (voh_param
- Vin) / IILmax
r1 = (4.6V
- 0.4V) / 400uA
r1 = 10.5k ohms
For power pins, the value used for r0 should be derived using the data book specs for ICC. For a 74LS151, Icc typ is 6mA at Vcc=5V and Icc max is 10mA at Vcc=5.25V. Therefore:

for Icc typ	:	r0 = 5V / 6mA = 833 ohms
for Icc max	:	r0 = 5.25V / 10mA = 525 ohms

If creating a multiple-parts-per-package device, such as a 74LS00 quad NAND gate, you should adjust the Icc load for the individual parts accordingly.

MATH FUNCTIONS

The following mathematical functions can be used in SimCode models:

Function	Description	Example
POW	power	X= (12 POW(3));
ABS	absolute value	X= (ABS(-12));

SQRT	square-root	X= (SQRT(2));
EXP	exponent	X= (EXP(10));
LOG	natural log	X= (LOG(0.1));
LOG10	log base 10	X= (LOG10(0.1));
SIN	sine	X= (SIN(0.1));
COS	cosine	X= (COS(0.1));
TAN	tangent	X= (TAN(0.1));
ASIN	arc sine	X= (ASIN(0.1));
ACOS	arc cosine	X= (ACOS(0.1));
ATAN	arc tangent	X= (ATAN(0.1));
HSIN	hyperbolic sine	X= (HSIN(0.1));
HCOS	hyperbolic cosine	X= (HCOS(0.1));
HTAN	hyperbolic tangent	X= (HTAN(0.1));

MESSAGE

Syntax

MESSAGE ("< *message* >"[, < *value* / *pin* >...]);

Description

Displays a message without pausing. The **MESSAGE** statement is used to output the information specified by the < *message* > string. It does not interrupt the simulation. The message is displayed in the status window during simulation.

Parameters

< *message* >		Message string including formatting characters as needed.
< *value* >		Variable or constant value.
< *pin* >		Pin name or index to pin variable.

Examples

MESSAGE("device instance= %s",INSTANCE);

Notes

A format string in **MESSAGE** is similar to a format that may be used in a printf statement in C. Valid formatting characters include (but are not limited to):

\t	tab
\n	new line
\r	carriage return
%d	Decimal display for short variable or current input/output state.
%D	Decimal display for short variable or old input/output state.
%x	Hex display for short variable or current input/output state.
%X	Hex display for short variable or old input/output state.
%c	Character display for short variable or current input/output state.
%C	Character display for short variable or old input/output state.
%e	Exponential display for real variable.
%f	Floating point engineering display for real variable.
%g	Short display (%e or %f) for real variable.
%s	String constant display.

The only valid string constants are:

INSTANCE	The present SimCode device instance name.
FUNC	The present SimCode device function name.
FILE	The present SimCode device file name.

MIN_TYP_MAX

Syntax

MIN_TYP_MAX (< *index* >: < *min* >, < *typ* >, < *max* >);

Description

Returns value from **MIN_TYP_MAX** look-up table. The **MIN_TYP_MAX** function is similar to the **SELECT_VALUE** function except that three values/variables must be entered. The keyword "NULL" can be substituted for one or two unknown values. If a predefined integer variable (see INTEGERS) is used as the < *index* >, unknown (NULL) values are calculated from the known values as follows:

Known Values	Formula
< *min* >, < *max* >	typical = (< *max* > + < *min* >) / 2
< *min* > only	typical = (< *min* > / <min scale factor>) maximum = (< *min* > / <min scale factor>) * <max scale factor>
< *typ* > only	minimum = (< *typ* > * <min scale factor>) maximum = (< *typ* > * <max scale factor>)
< *max* > only	minimum = (< *max* > / <max scale factor>) * <min scale factor> typical = (< *max* > / <max scale factor>)

Parameters

< *index* >		Input variable (index to select min (1), typ (2) or max (3) values).
< *min* >		Minimum data book value.
< *typ* >		Typical data book value.
< *max* >		Maximum data book value.

Examples

tplh_val = (MIN_TYP_MAX(tp_param: NULL, 5n, NULL));
In this example, if we assume that the SPICE options PROPMNS and PROPMXS are set to their default values (see Notes), then:
if tp_param = 1 (i.e. return the min value), then tplh_val = 2.5n
if tp_param = 2 (i.e. return the typ value), then tplh_val = 5n
if tp_param = 3 (i.e. return the max value), then tplh_val = 7.5n
ricch_val = (MIN_TYP_MAX(i_param: NULL, 2500, 1250));
In this example, if we assume that the SPICE options CURRENTMNS and CURRENTMXS are set to their default values (see Notes), then:
if i_param = 1 (i.e. return the min value), then ricch_val = 3750
if i_param = 2 (i.e. return the typ value), then ricch_val = 2500
if i_param = 3 (i.e. return the max value), then ricch_val = 1250

Notes

If < *index* > is not one of the predefined variables listed below, then <min scale factor> = 0.5 and <max scale factor> = 1.5.
The <min scale factor> and <max scale factor> for each of these predefined variables can be changed on the **Advanced Options** page of the *Analyses Setup* dialog (**Design » Simulate » Mixed Sim**). The <min scale factor> and <max scale factor> are reversed for ld_param, drv_param and i_param because these parameters control a resistance value rather than a current value (i.e., maximum load equates to minimum resistance.)

Variable	SPICE Option	Parameter Default
tp_param	PROPMNS	<min scale factor> = 0.5
	PROPMXS	<max scale factor> = 1.5
tt_param	TRANMNS	<min scale factor> = 0.5

	TRANMXS	<max scale factor> = 1.5
ld_param	LOADMNS	<min scale factor> = 1.5
	LOADMXS	<max scale factor> = 0.5
drv_param	DRIVEMNS	<min scale factor> = 1.5
	DRIVEMXS	<max scale factor> = 0.5
i_param	CURRENTMNS	<min scale factor> = 1.5
	CURRENTMXS	<max scale factor> = 0.5
vth_param	VTHMNS	<min scale factor> = 0.5
	VTHMXS	<max scale factor> = 1.5
user_param	USERMNS	<min scale factor> = 0.5
	USERMXS	<max scale factor> = 1.5

NO_CHANGE

Syntax

NO_CHANGE < *output* > [< *output* > ...];

Description

Use the **NO_CHANGE** function to indicate no-change for specified output pins. Use this statement on bi-directional pins when the bi-directional pin is being treated as an input.

Parameters

< *output* >		Name of or variable index to the output pin.

Examples

NO_CHANGE Q1 Q2 Q3 Q4;

Notes

Pin names and variables cannot be mixed in the same **NO_CHANGE** statement. References to outputs must be either all pin names or all variable names.

NUMBER

Syntax

NUMBER (<MSB *pin* >, [< *pin* >, ...] <LSB *pin* >);

Description

Returns number based on binary weighted pin states. The **NUMBER** function returns a short integer that represents the decimal value of the binary number represented by the list of < *pin* >. Each bit (represented by a < *pin* >) is set to 1 if the < *pin* > is non-zero, otherwise it is set to 0.

Parameters

< *pin* >		Name of or index to a pin.

Examples

A = (NUMBER(D3,D2,D1,D0));
In this example, if D3 is HIGH, and D2, D1 and D0 are LOW (1000), then A = 8.

Notes

The first < *pin* > in the list represents the most-significant-bit (MSB) and the last < *pin* > in the list represents the least-significant-bit (LSB).

OPERATORS

The following operators can be used in SimCode expressions:

=	Equals (sets a variable or output pin to a value or state).
+	Add
	Subtract
	Multiply
/	Divide
~	Logical not
!	Bitwise complement
&&	AND
||	OR

^^	XOR

Bitwise Operators

&	AND
|	OR
^	XOR
<<	Shift left
>>	Shift right

Relative Comparators

=	Equal
!=	Not equal
<	Less than
<=	Less than or equal to
>	greater than
>=	greater than or equal to

Description

Operators are used to set and manipulate variables and expressions.

Example s

```
clk_twl = (25n);
reg = (reg
+ 1);
vx = (vol_param
- 10m);
C = (A * B);
val = (xval / 2);
X = (A && ~(B));
Y = (!(X)); //if X=1 then Y=FFFFFFFE
A = (X & 1); //if X=1 then A=1, if X=2 then A=0
B = (X | 8); //if X=1 then B=9, if X=2 then B=10
C = (X >> 2); //if X=1 then C=0, if X=2 then C=0
D = (2 >> X); //if X=1 then D=1, if X=2 then D=0
E = (X << 2); //if X=1 then E=4, if X=2 then E=8
F = (2 << X); //if X=1 then F=4, if X=2 then F=8
IF (A >= B) THEN ...
IF ((A < 2) && (B > 3)) THEN ...
IF ((C < 2) || (X > 4)) THEN ...
```

Notes

Expressions must be enclosed within parentheses ( ). Expressions are always evaluated from left to right within parentheses. You should use parentheses to set precedence within an expression. When using the unary operators (logical NOT and bitwise complement) on values, variables, expressions, etc. the values, variables, expressions, etc. must be in parentheses ( ).

OUTPUTS

Syntax

OUTPUTS < *output pin* >[, < *output pin* >, ...];

Description

Declares output pins (pins that drive or load the circuit). The **OUTPUTS** data type is used to define the pins which affect the operation of circuitry external to the device. These generally include input, output, I/O and power pins. Input and power pins are included in this list because their presence constitutes a load on the driving circuitry.

Parameters

< *output pin* >		Name of the output pin.

Example s

OUTPUTS VCC_LD, PRE_LD, DATA_LD, CLK_LD, CLR_LD, QN, Q;

Notes

Output pin names must begin with a letter and be defined before they are used.

PARAM_SET

Syntax

PARAM_SET (< *param var* >)

Description

The **PARAM_SET** function is used to determine if a parameter in the SimCode model definition has been set. It returns 1 if the specified parameter was set (e.g., vil_param=0.8) otherwise it returns 0.

Parameters

< *param var* >		SimCode model definition parameter.

Example s

```
A = PARAM_SET(ld_param);
IF (PARAM_SET(voh_param)) THEN ...
```

Notes

See INTEGERS and REALS for a list of SimCode model definition parameters and their associated variable names.

PROMPT

Syntax

PROMPT ("< *message* >"[, < *value* / *pin* >...]);

Description

The **PROMPT** statement is used to pause simulation and display the information specified by the < *message* > string. The message is displayed in the status window during simulation. The user must click on a button to continue execution of the SimCode.

Parameters

< *message* >		Message string including formatting characters as needed.
< *value* >		Variable or constant value.
< *pin* >		Pin name or index to pin variable.

Examples

PROMPT("input=%d time=%f device=%s", D1, t1, INSTANCE);

Notes

A format string in **PROMPT** is similar to a format that may be used in a printf statement in C. Valid formatting characters include (but are not limited to):

\t	tab
\n	new line
\r	carriage return
%d	Decimal display for short variable or current input/output state.
%D	Decimal display for short variable or old input/output state.
%x	Hex display for short variable or current input/output state.
%X	Hex display for short variable or old input/output state.
%c	Character display for short variable or current input/output state.
%C	Character display for short variable or old input/output state.

%e	Exponential display for real variable.
%f	Floating point engineering display for real variable.
%g	Short display (%e or %f) for real variable.
%s	String constant display.

The only valid string constants are:

INSTANCE	The present SimCode device instance name.
FUNC	The present SimCode device function name.
FILE	The present SimCode device file name.

PWL_TABLE

Syntax

PWL_TABLE (< *IN var* >: < *IN1* >,< *OUT1* >,< *IN2* >,< *OUT2* >[,...< *INn* >,< *OUTn* >])

Description

This piece-wise-linear function is essentially a look-up table. The value of < *IN var* > is used to look up an entry in a table which consists of pairs of values. The first value in each pair is an input compare value and the second value is the corresponding output value. If the < *IN var* > value is less than the first < *IN* > value, the first < *OUT* > value is returned. If the < *IN var* > value is greater than the last < *INn* > value, then the last < *OUTn* > value is returned. Linear interpolation is done between entries according to the formula:
value = (((OUTA-OUTB)/(INA-INB))*(<IN var>-INA)+OUTA)
where < *IN var* > falls between the input compare values INA and INB. The actual output value will fall between output values OUTA and OUTB.

Parameters

< *IN var* >		input variable (integer or real)
< *INx* >		input compare value
< *OUTx* >		output value at < *INx* >

Examples

twh = (PWL_TABLE(var: 5,180n,10,120n,15,80n));
In this example, if var = 10 then twh = 120n and if var = 12 then twh = 104.

Notes

Two or more IN/OUT data value pairs must be entered and the IN values must be entered in ascending order. There is no limit to the maximum number of IN/OUT data pairs that can be entered.

PWR_GND_PINS

Syntax

PWR_GND_PINS (< *pwrpin* >, < *gndpin* >);

Description

The **PWR_GND_PINS** statement defines which of the input pins are power and ground and sets the Power and Ground parameters of the device to absolute voltages as follows:
pwr_param = voltage on < *pwrpin* >
gnd_param = voltage on < *gndpin* >

Parameters

< *pwrpin* >		name of the power pin
< *gndpin* >		name of the ground pin

Examples

PWR_GND_PINS(VCC, GND);

Notes

This statement can only be used once in the SimCode. Only one pin can be defined for power and one for ground.

READ_DATA

Syntax

READ_DATA (< *array* >[, < *array* >, ...])

Description

The **READ_DATA** function opens the file specified by the "data=" parameter in the device's .MODEL statement (in the intermediate linked model file *.mdl) and reads ASCII text data into one or more arrays. The number and type (integer/real) of the values per line that will be read is based on the number and type of array variables that are specified in the function call. The number of data lines read is determined by the number of data lines in the specified file and/or the size of the smallest array in the function call.
The **READ_DATA** function returns the number of lines read. A negative number is returned if an error is encountered:

-1	Invalid file name
-2	Can't find file
-3	Invalid array
-4	Illegal array access
-5	Data type
-6	Expected data value

Parameters

< *array* >		Name of the array into which the value is placed.

Examples

```
MYDEVICE.MDL file:
.MODEL AMYDEVICE XSIMCODE(file="\{MODEL_PATH\}MYDEVICES.SCB"
\+ func=MyDevice data="\{MODEL_PATH\}MYDEVICE.DAT" \{mntymx\})
MYDEVICE.DAT file:
8, 8E-6
9, 9E-6
10, 1E-5
11, 1.1E-5

MyDevice SimCode:
nlines = READ_DATA(int_array, real_array);
```

This example opens a filed called MYDEVICE.DAT. It reads 2 columns of data from the file where the first column contains integer values and the second column contains real values. If the arrays are declared as int_array[3] and real_array[5] then only the first 3 data lines will be read and nlines will be set to 3.

Notes

Multiple values per line in the data file must be separated by commas.
The real values in the data file must be in scientific notation.
The device's .MODEL statement which contains the "data=" parameter must be placed in the device symbol's .MDL file.

REALS

Syntax

REALS < *var* >[, < *var* >, ...];

Description

The **REALS** data type is used to define real variables and arrays.

Parameters

< *var* >		name of the variable

Examples

REALS tplh_val, tphl_val, ricc_val, vbias, values64;

Notes

Real variables and arrays must begin with a letter and be defined before they are used. Real arrays are defined by following the array name with a left bracket ([ ), an integer number which defines the size of the array, and a right bracket ( ]). Real arrays can be set and/or used in expressions.
The following are reserved SimCode real variables which do not need to be declared:

Variable	Use	Digital Model Parameter
vil_param	low input state value	VIL value
vih_param	high input state value	VIH value
vol_param	low output state value	VOL value
voh_param	high output state value	VOH value
v3s_param	tri-state output state value	N/A
rol_param	low output strength value	N/A
roh_param	high output strength value	N/A
r3s_param	tri-state output strength value	N/A
pwr_param	voltage on power pin	PWR value
gnd_param	voltage on ground pin	GND value
present_time	present simulation time	N/A
previous_time	previous simulation time	N/A
sim_temp	circuit operating temperature	N/A (SPICE Option: TEMP)

The **Digital Model Parameter** can be set independently for each digital device using the respective parameters found on the **Parameters** tab of the *Sim Model* dialog. This dialog is accessed by double-clicking on the entry for the simulation model link in the **Models** region of the associated *Component Properties* dialog. Entering a value for any of PWR, GND, VIL, VIH, VOL, and VOH in the *Sim Model* dialog, will override any value specified by the SimCode model.
The values of pwr_param and gnd_param are set each time the **PWR_GND_PINS** statement is executed. The value of present_time and previous_time are set each time the time step changes. The value of sim_temp is the current operating temperature of the circuit which can be set from the SPICE Option " **TEMP** ".

RECOVER

Syntax

RECOVER (< *clk input* > = {LH}|{HL}
< *mr input* > [< *mr input* > ...]
{TREC=< *time* >}|{TRECL=< *time* > TRECH=< *time* >} ["< *message* >"];

Description

Tests inputs for recovery time violations. The **RECOVER** function compares the time difference between a level change (LH or HL) on the < *clk input* > and a level change on the < *mr input* > to a specified test time. **RECOVER** test times are specified jointly using TREC=< *time* > (which sets TRECL and TRECH to the same value) or individually using TRECL=< *time* > and TRECH=< *time* >. If the compare time is less than the specified < *time* > a warning will be displayed during simulation. An optional < *message* > string can be included in the **RECOVER** statement which will be output if a warning is displayed.

Parameters

< *clk input* >		Name of or index to the input clock/reference pin under test
< *mr input* >		Name of or index to the input set/reset pin under test.
TREC		Recovery time for both low and high going < *mr pin* >.
TRECL		Recovery time for low going < *mr pin* >.
TRECH		Recovery time for high going < *mr pin* >.
< *time* >		Specified test time.
< *message* >		Text string that will be displayed if a warning occurs.

Examples

RECOVER(CLK=LH PRE CLR TREC=trec_val
"CLK->PRE or CLR");

Notes

Data book specifications should be used with this function. TRECL=< *time* > and TRECH=< *time* > can be entered in the same **RECOVER** test. The **RECOVER** test will be made only if the state of the < *mr input* > matches the time parameter (TRECL=LOW, TRECH=HIGH) when the < *clk input* > makes the specified transition (LH or HL). For example, if < *clk input* >=LH and TRECL is specified then the < *mr input* > must be LOW when the < *clk input* > goes from LOW to HIGH for a **RECOVER** test to be made.
Pin names and variables can be mixed in the same **RECOVER** statement.

RETURN

Syntax

RETURN

Description

Returns from a subroutine in the SimCode. The **RETURN** instruction is used to return program flow to the instruction that followed the last **GOSUB** instruction.

SELECT_VALUE

Syntax

SELECT_VALUE (< *index* >: < *val* / *pin* / *var* >,< *val* / *pin* / *var* >[,< *val* / *pin* / *var* >,...]);

Description

Returns a value from a simple look-up table. The **SELECT_VALUE** function returns the value of the number or variable indicated by the value of the index variable.

Parameters

< *index* >		input variable (index to < *val* / *pin* / *var* >)
< *val* / *pin* / *var* >		output value, pin or variable

Examples

A = (SELECT_VALUE(B: 16, 8, 4, 2, 1));
In this example, if B = 2 then A = 8 (the 2nd value).

Notes

The number of values and/or variables used is not limited.

SETUP_HOLD

Syntax

SETUP_HOLD (< *clk input* > = {LH}|{HL}
< *data input* > [< *data input* > ...]
{TS=< *time* >}|{TSL=< *time* > TSH=< *time* >}
{TH=< *time* >}|{THL=< *time* > THH=< *time* >} ["< *message* >"];

Description

Tests inputs for setup and hold time violations The **SETUP_HOLD** function compares the time difference between a level change (LH or HL) on the < *clk input* > and a level change on the < *data input* > to a specified test time. SETUP test times are specified jointly using TS=< *time* > (which sets TSL and TSH to the same value) or individually using TSL=< *time* > and TSH=< *time* >. HOLD test times are specified jointly using TH=< *time* > (which sets THL and THH to the same value) or individually using THL=< *time* > and THH=< *time* >. If the compare time is less than the specified < *time* > a WARNING will be displayed. An optional < *message* > string can be included in a **SETUP_HOLD** statement which will be output if a WARNING is displayed.

Parameters

< *clk input* >		Name of or index to the input clock/reference pin under test.
< *data input* >		Name of or index to the input data pin under test.
TS		Setup time for both low and high going < *data input* >.
TSL		Setup time for low going < *data input* >.
TSH		Setup time for high going < *data input* >.
TH		Hold time for both low and high going < *data input* >.
THL		Hold time for high going < *data input* >.
THH		Hold time for low going < *data input* >.
< *time* >		Specified test time.
< *message* >		Text string that will be displayed if a warning occurs.

Examples

SETUP_HOLD(CLK=LH DATA Ts=ts_val Th=th_val "CLK->DATA");

Notes

Data book specifications should be used with this function. TSL=< *time* >, TSH=< *time* >, THL=< *time* > and THH=< *time* > can be entered in the same **SETUP_HOLD** statement. The SETUP and/or HOLD test will be made only if the state of the < *data input* > matches the time parameter (TSL or THL=LOW, TSH or THH=HIGH) when the < *clk input* > makes the specified transition (LH or HL). For example, if < *clk input* >=LH and TSL is specified, then the < *data input* > must be LOW when the < *clk input* > goes from LOW to HIGH for a SETUP test to be made.
Pin names and variables can be mixed in the same **SETUP_HOLD** statement.

STATE

Syntax 1

STATE < *output* > [< *output* >...] = (< *expression* >);

Syntax 2

STATE < *output* > [< *output* >...] = {ZERO}|{ONE}|{UNKNOWN};

Description

The state of an output pin is determined by its level and its strength. The **STATE** command sets the level and strength for one or more output pins or variables. If < *expression* > is less than or equal to vol_param, the output will be set to ZERO. If < *expression* > is greater than or equal to voh_param, the output will be set to ONE. Otherwise, the output will be set to UNKNOWN. The level and strength values are set according to the state:

< *expression* >	**State**	**Level**	**Strength**
<= vol_param	ZERO	vol_param	rol_param
>= voh_param	ONE	voh_param	roh_param
other	UNKNOWN	v3s_param	r3s_param

Parameters

< *output* >		Name of, or variable index to, the output pin.
< *expression* >		Any expression to be compared to VOL or VOH.

Examples

```
STATE Q = ONE;
STATE Q1 Q2 Q3 Q4 = ZERO;
STATE OUT = ((1+2)/3);
In the last example, OUT will be:
ZERO if vol_param > 1
UNKNOWN if vol_param < 1 and voh_param > 1
ONE if voh_param < 1
```

Notes

Output pins can be specified by using the output pin name or by an integer variable that contains the index of an output pin. Pin and variable names cannot be mixed in the same **STATE** command. References to outputs must be either all pin names or all variable names.

STATE_BIT

Syntax

STATE_BIT < *output* > [< *output* > ...] = (< *expression* >);

Description

The state of an output pin is determined by its level and its strength. The **STATE_BIT** command is used to set the level and strength for one or more output pins based on the value of the < *expression* >. The state of the first pin listed is set according to the first (least-significant-bit) of the expression's value, the state of the second pin listed is set according to second bit of the expression's value, and so on. The level and strength values are set by the bit's value:

Bit Value	State	Level	Strength
0	ZERO	vol_param	rol_param
1	ONE	voh_param	roh_param

Parameters

< *output* >		name of or variable index to the output pin
< *expression* >		any expression which can be bitwise matched with the outputs

Examples

STATE_BIT Q1 Q2 Q3 Q4 = (internal_reg);
In this example, if internal_reg = 11 (1011 binary) then Q1 (LSB) = ONE, Q2 = ONE, Q3 = ZERO and Q4 (MSB) = ONE.

Notes

Output pins can be specified by using the output pin name or by an integer variable that contains the index of an output pin. Pin and variable names cannot be mixed in the same **STATE_BIT** statement. References to outputs must be either all pin names or all variable names.
The maximum number of output pins/vars is limited to 16.

STEP_OFF

Syntax

STEP_OFF

Description

Turns off the SimCode trace mode.

STEP_ON

Syntax

STEP_ON

Description

Turns on the SimCode trace mode. This causes the SimCode to display the Program Counter (PC) number and each SimCode instruction before it is executed.

STRENGTH

Syntax 1

STRENGTH < *output* > [< *output* > ...] = (< *expression* >);

Syntax 2

STRENGTH < *output* > [< *output* > ...] = {STRONG}|{HI_IMPEDANCE};

Description

The state of an output pin is determined by its level and its strength. Use the **STRENGTH** command to set the strength of one or more output pins.

Value	State	Strength
STRONG	ZERO	rol_param
STRONG	ONE	roh_param
HI_IMPEDANCE	N/A	r3s_param
< *expression* >	N/A	< *expression* >

Parameters

< *output* >		Name of or variable index to the output pin.
< *expression* >		Any expression to be used directly as a strength.

Notes

Output pins can be specified by using the output pin name or by an integer variable the contains the index of an output pin. Pin and variable names cannot be mixed in the same **STATE** statement. References to outputs must be either all pin names or all variable names.

SUPPLY_MIN_MAX

Syntax

SUPPLY_MIN_MAX (< *min value* >, < *max value* >);

Description

The **SUPPLY_MIN_MAX** function checks the voltage difference between the power and ground pins defined in **PWR_GND_PINS** . If the WARN parameter is set to ON on the **Parameters** tab of the associated *Sim Model* dialog and the voltage difference (pwr_param - gnd_param) is less than < *min value* > or greater than < *max value* > a warning will be displayed during simulation.

Parameters

< *min value* >		Minimum recommended power supply voltage.
< *max value* >		Maximum recommended power supply voltage.

Examples

SUPPLY_MIN_MAX(4.75, 5.25);

Notes

Data book specifications should be used with this function. **PWR_GND_PINS** must be defined to use this function.

TABLE

Syntax

TABLE < *line* >
< *input* > [< *input* > ...] < *output pin* > [< *output pin* > ...]
< *input state* > [< *input state* > ...] < *output state* > [< *output state* > ...];

Description

The **TABLE** statement operates like a truth table to set the level and strength of the specified outputs.
Valid input states are:
0 low (input voltage is <= vil_param).
1 high (input voltage is >= vih_param).
X don't care what input voltage is.
Valid output states are:
L ZERO (set output level to vol_param)
H ONE (set output level to voh_param).
Z UNKNOWN (set output level to v3s_param).
Output state letters can be followed by a colon and a letter to indicate strength:
s STRONG (set output to rol_param for L and roh_param for H).
z HI_IMPEDANCE (set output to r3s_param).
If a strength character is not specified after an output state then STRONG will be used for L and H states and HI_IMPEDANCE will be used for Z states.

Parameters

< *line* >		Variable into which the table line number is placed.

< *input* >		Name of the input pin or variable index to the input pin.
< *output pin* >		Name of the output pin.
< *input state* >		State of the individual inputs.
< *output state* >		State of the individual outputs based on input conditions.

Examples

```
TABLE tblIndex
INA INB OUT
0    0   H
0    1   H
1    0   H
1    1   L;
```

This example is representative of 1/4 of a 7400 2-input NAND gate. If input pins INA and INB are both high (>= vih_param), OUT is set to ZERO (vol_param) and STRONG (rol_param) and tblIndex is set to 4.

Notes

Each row is tested sequentially from top to bottom until the input conditions are met. The outputs are set for the first row to meet the input conditions. The < *line* > is set to the line number in the table that was used. If no match was made then < *line* > is set to 0.
Input pin and variable names cannot be mixed in the same **TABLE** statement. References to inputs must be either all pin names or all variable names.

VALUE

Syntax

VALUE (< *pin* >)

Description

Returns the value of the specified pin. The **VALUE** function returns a real number that indicates the voltage level of the specified pin.

Parameters

< *pin* >		Name of, or index to, a pin.

Examples

```
v = (VALUE(D3));
```

VIL_VIH_PERCENT

Syntax

VIL_VIH_PERCENT (< *vil %* >, < *vih%* >);

Description

VIL and VIH do not use a min/typ/max array to select their values, but must be declared explicitly for each digital device. The **VIL_VIH_PERCENT** statement sets the VIL and VIH parameters of the device to a percentage of the supply voltage as follows:
vil_param = (pwr_param
- gnd_param) * < *vil %* >
vih_param = (pwr_param
- gnd_param) * < *v ih %* >

Parameters

< *vil %* >		Percentage of the supply voltage which defines vil.
< *vih %* >		Percentage of the supply voltage which defines vih.

Examples

VIL_VIH_PERCENT(33, 67);

Notes

PWR_GND_PINS must be defined to use this function.
The % values must be greater than 0 and less than 100.
The vil_param and vih_param values set by **VIL_VIH_PERCENT** are overridden by any values set for the VIL value and VIH value parameters on the **Parameters** tab of the associated *Sim Model* dialog (accessed by double-clicking on the entry for the linked simulation model in the associated *Component Properties* dialog).

VIL_VIH_VALUE

Syntax

VIL_VIH_VALUE (< *vil* >, < *vih* >);

Description

VIL and VIH do not use a min/typ/max array to select their values, but must be declared explicitly for each digital device. The **VIL_VIH_VALUE** statement sets the VIL and VIH parameters of the device to absolute voltages as follows:
vil_param = < *vil* >
vih_param = < *vih* >

Parameters

< *vil* >		Absolute voltage level which defines vil.
< *vih* >		Absolute voltage level which defines vih.

Examples

VIL_VIH_VALUE(1.25, 1.35);

Notes

In order to more accurately model the actual switching characteristics of a digital input, VIL and VIH are not generally set to their specified data

book values. The exception is the case of devices with a specified hysteresis such as the 74LS14. Typically, the hysteresis of a digital device is small, in the order of 100mV, but never 0V.
The vil_param and vih_param values set by **VIL_VIH_VALUE** are overridden by any values set for the VIL value and VIH value parameters on the **Parameters** tab of the associated *Sim Model* dialog (accessed by double-clicking on the entry for the linked simulation model in the associated *Component Properties* dialog).

VOL_VOH_MIN

Syntax

VOL_VOH_MIN (< *vol offset* >, < *voh offset* >, < *min voh-vol* >);

Description

VOL and VOH do not use a min/typ/max array to select their values, but must be declared explicitly for each digital device. The **VOL_VOH_MIN** statement sets the VOL and VOH parameters of the device as follows:
vol_param = gnd_param
+ < *vol offset* >
voh_param = pwr_param
+ < *voh offset* >

Parameters

< *vol offset* >		Voltage offset which must be applied to ground pin voltage to get vol.
< *voh offset* >		Voltage offset which must be applied to power pin voltage to get voh.
< *min voh-vol* >		Minimum allowed difference between voh and vol.

Examples

VOL_VOH_MIN(0.2, -0.4, 0.1);
In this example:
If gnd_param = 0V and pwr_param = 5.0V, then
vol_param = 0.2V and voh_param = 4.6V
If gnd_param = 0V and pwr_param = 0.5V, then
vol_param = 0V and voh_param = 0.1V

Notes

In order to more accurately model the actual characteristics of a digital output, VOH is not generally set to its specified data book value. The reason for this deviation is that data book values for VOH are specified for maximum IOH load. In Digital SimCode, VOL and VOH represent an unloaded output voltage.
PWR_GND_PINS must be defined to use this function.
The vol_param and voh_param values set by VOL_VOH_MIN are overridden by any values set for the VOL value and VOH value parameters on the **Parameters** tab of the associated *Sim Model* dialog (accessed by double-clicking on the entry for the linked simulation model in the associated *Component Properties* dialog). These are offset values rather than absolute voltages. The < *voh offset* > is negative so that when added to pwr_param, the resulting VOH will not be greater than pwr_param. If the difference between the resulting vol_param and voh_param is less than < *min voh-vol* >, then vol_param will be set to the value of gnd_param and voh_param will be set to gnd_param
+ < *min voh-vol* >.

WHILE ... DO

Syntax

WHILE (< *expression* >) **DO BEGIN** ... **END** ;

Description

The **WHILE ... DO** statement is used to loop through a section of SimCode until < *expression* > evaluates to false.

Parameters

< *expression* >		Any expression that can be evaluated as true or false

Examples

```
i = 1;
WHILE (i <= 5) DO
BEGIN
  data[i] = data[i + 1];
  i = i + 1;
END;
```

Notes

Program flow will remain in a loop between the **BEGIN** and **END** statements until < *expression* > evaluates to false, then program flow resumes after the **END** statement.

WIDTH

Syntax

WIDTH (< *input* > [< *input* >...] {TWL=< *time* >}|{TWH=< *time* >} ["< *message* >"];

Description

The **WIDTH** function compares the pulse width on each < *input* > to the specified test WIDTH times. A low level test time is specified using TWL=< *time* > while a high level test time is specified using TWH=< *time* >. If the compare time is less than the specified < *time* > a WARNING will be displayed. An optional < *message* > string can be included in the **WIDTH** statement which will be output if a WARNING is displayed.

Parameters

< *input* >		Name of or variable index to the input pin under test.
TWL		Width of a low going pulse.
TWH		Width of a high going pulse.
< *message* >		Text string that will be displayed if a warning occurs.

Examples

WIDTH(CLK TWL=clk_twl TWH=clk_twh "CLK");
WIDTH(PRE CLR TWL= pre_clr_twl "PRE or CLR");

Notes

Data book specifications should be used with this function. The input pins can be input pin names and/or integer variables that contain an index value to an input pin. Pin names and variables can be mixed in the same **WIDTH** statement.

WIDTH_TIME

Syntax

WIDTH_TIME (< *input* >)

Description

This function returns a real value that indicates the last pulse width encountered on the specified < *input* >.

Parameters

< *input* >		Name of or index to an input pin.

Examples

PW = (WIDTH_TIME(CP2));