State Machine Editors User Manual
for the HDL Designer Series

Software Version 2010.3

June, 2011

Contractor/manufacturer is:
Mentor Graphics Corporation
8005 S.W. Boeckman Road, Wilsonville, Oregon 97070-7777.
Telephone: 503.685.7000
Toll-Free Telephone: 800.592.2210
Website: www.mentor.com
SupportNet: supportnet.mentor.com/
Send Feedback on Documentation: supportnet.mentor.com/user/feedback_form.cfm

Table of Contents

Chapter 1
State Machines

This chapter is an introduction to the graphical state diagram and algorithmic state machine (ASM) views supported by the HDL Designer Series.

Introduction

A *state machine* represents the control requirements of part or all of a system. Typically, any primitive *block* or *component* in a design hierarchy which represents control behavior can be represented by a graphical state machine view.

A state machine view describes how the parent block or component responds to input conditions and the actions taken in response to these conditions. Synthesizable HDL can be generated automatically from the state machine view.

The *HDL Designer Series* supports state machines drawn as *state diagram*s or as algorithmic state machine (ASM) charts.

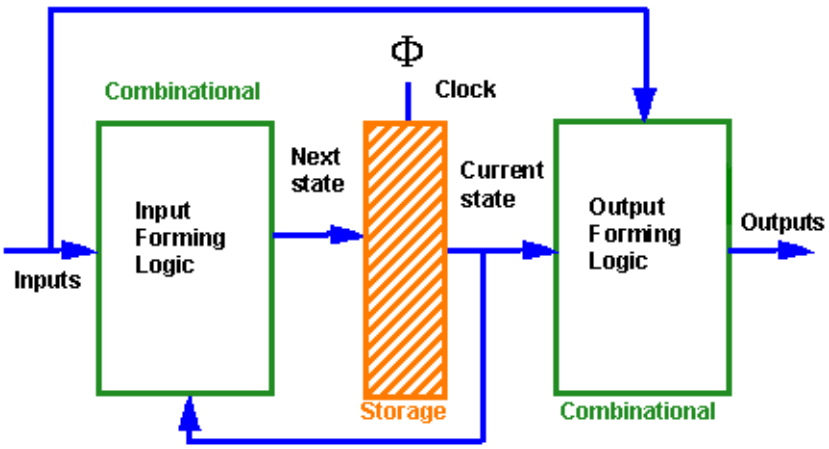
State Diagrams

A classic finite *state machine* (FSM) represents a system in terms of a number of *state*s and the *transition*s between them. The output behavior can be described using a *Mealy notation* or *Moore notation* state machine model.

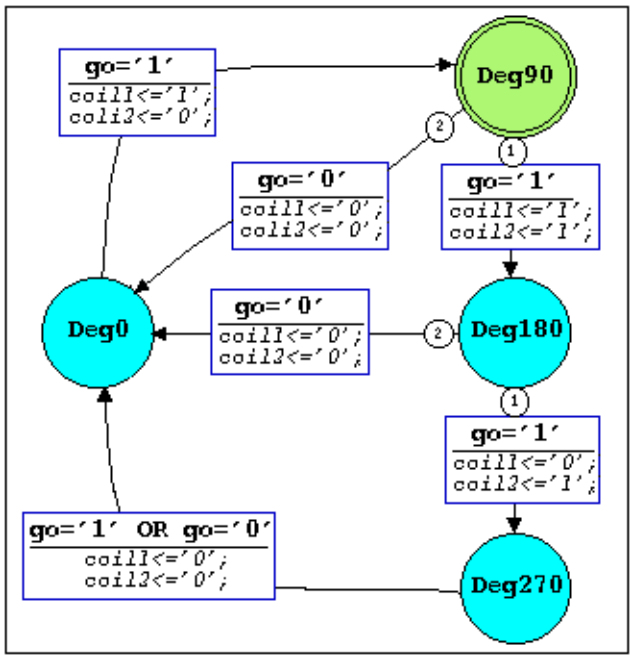
The *state diagram* editor supports state machines with synchronous or asynchronous *action*s. You can specify simple combinatorial output signals or specify registered and clocked outputs and use a choice of state encoding schemes. The way you use these features determines the performance of a state machine design.

Mealy Behavior

The outputs of a Mealy state machine are a function of its current state and inputs. Changing the input has a corresponding affect on the outputs. When an input condition is satisfied, a Mealy state machine performs specified actions, such as changing the values of outputs and the transitions from one state to another.



The following example shows a state diagram for a stepper motor controller drawn using simple states connected by transitions with transition actions.



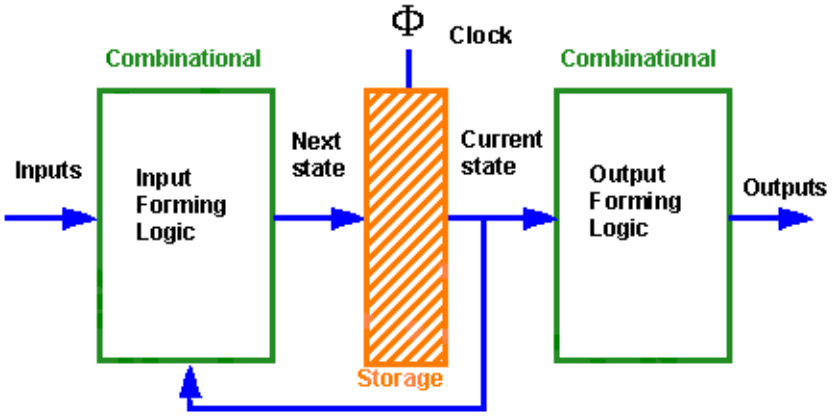
The output actions in this state machine are executed when the conditions associated with the transitions between states are satisfied.

Transitions depend on the conditions that exist on an active clock edge and any input changes that occur between active clock edges cannot cause a change of state.

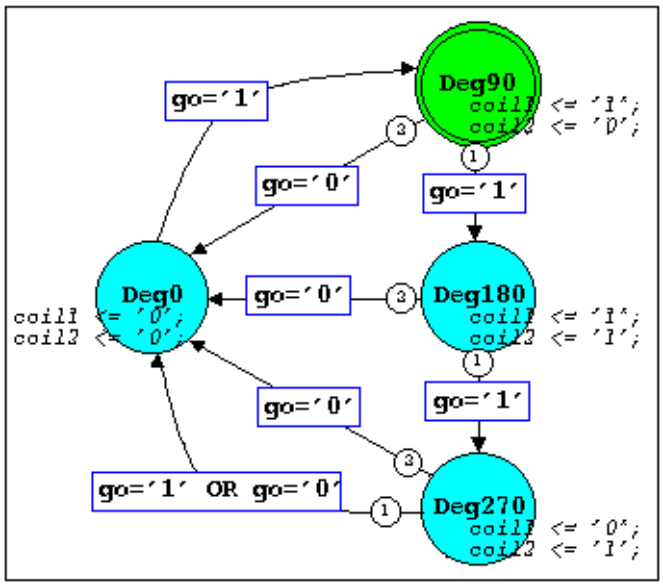
Moore Behavior

The outputs of a Moore state machine are a function of its state only, therefore the outputs only change if the state changes. However, if a Moore state contains an assignment to an input signal,

the state machine has an input dependency and its outputs are a function of both the state and the inputs, that is it will have Mealy behavior.



The following example shows a state diagram for a stepper motor controller drawn using simple states containing state actions and connected by transitions without actions is shown below.



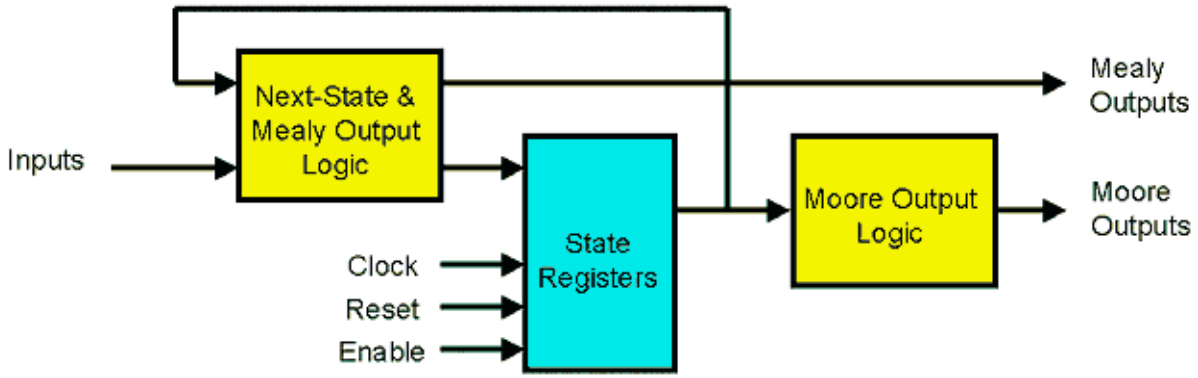
The output actions in this state machine are executed when a state is entered.

Transitions depend on the conditions between the states being satisfied.

Any output changes due to asynchronous inputs occur on an active clock edge and any input changes that occur between active clock edges do not change the outputs.

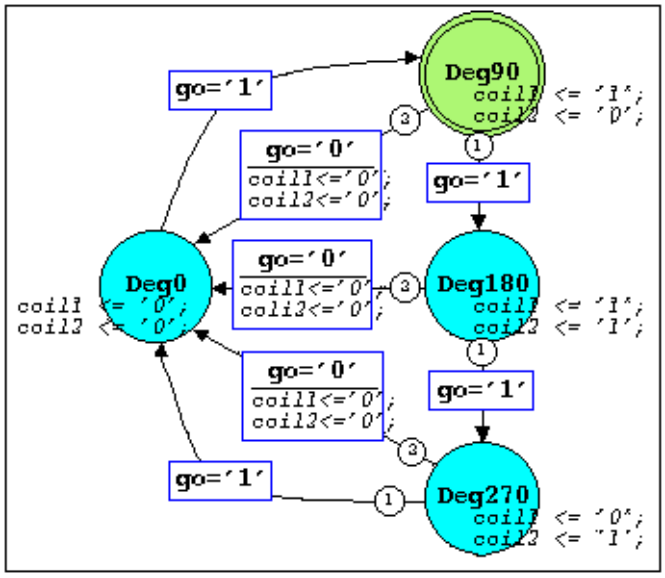
Combined Mealy and Moore Behavior

The following picture summarizes the behavior of a combined Mealy and Moore state machine:



The HDL Designer Series allows you to define a state diagram using strictly Mealy notation (with transition actions between states), Moore notation (with state actions on entry into a state) or to combine the use of transition and state actions. A state diagram may contain any combination of *state actions* and *transition actions*.

The following example shows a state diagram for a stepper motor controller drawn using a combination of state actions and transition actions:

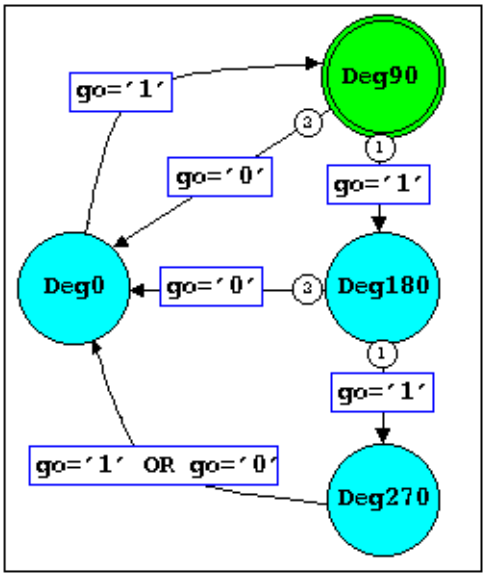


The step operations in this state machine are handled as state actions.

However, the “return to zero” operation is handled by transition actions which ensure an immediate return when the condition *go='0'* is satisfied.

State Variable Definition

The *state variable* is a signal which describes the current state of the state machine.



Constants are defined for the name of each state in this simple stepper motor controller. The state variable can be assigned to an output signal which is set to the bit pattern that corresponds to the current state.

The state variable allows each state to be assigned a unique binary code which can be used to explicitly control the bit pattern for synthesis.

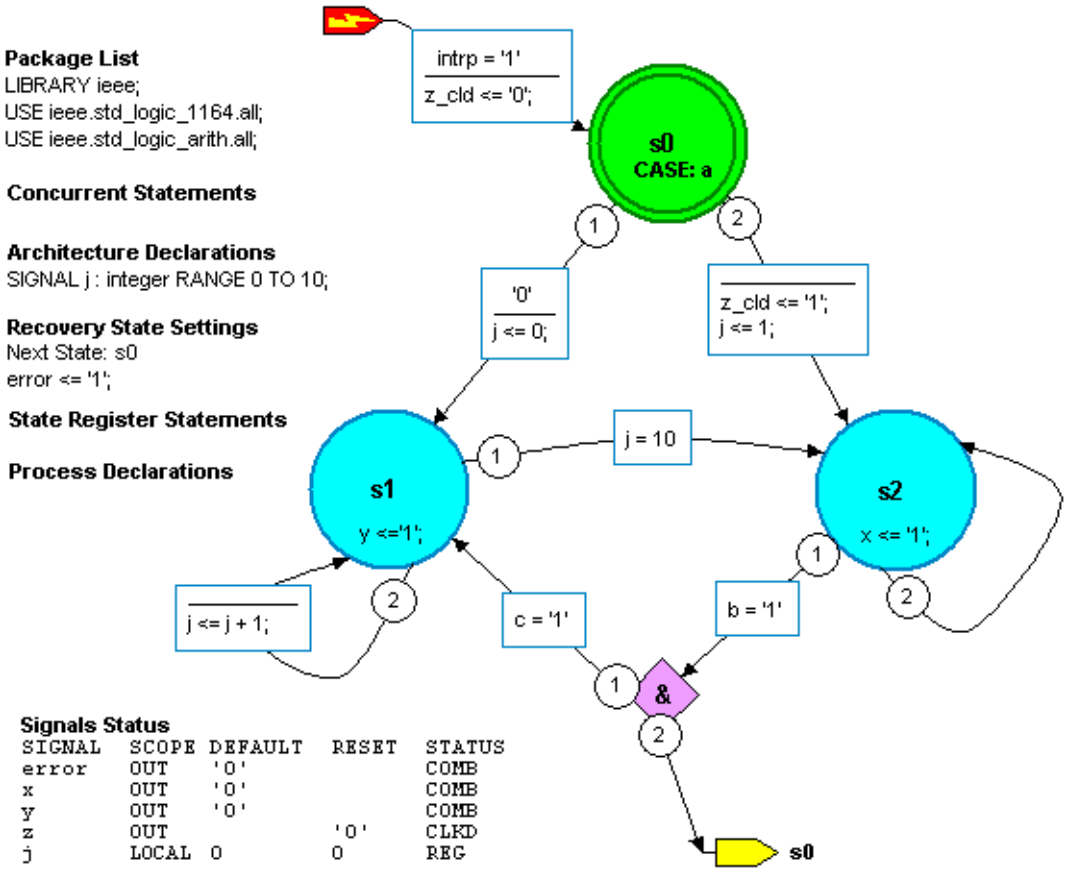
When using VHDL, the state variable is typically an enumerated type definition (defined in a VHDL package) or constants can be used to map state names to values.

For example:

```
SUBTYPE state_variable_type IS std_logic_vector(1 DOWNTO 0);
CONSTANT Deg0 : state_variable_type := "00";
CONSTANT Deg90 : state_variable_type := "10";
CONSTANT Deg180 : state_variable_type := "11";
CONSTANT Deg270 : state_variable_type := "01";
```

State Diagram Example

The following picture shows an example of a simple state diagram with both state and transition actions. This example uses CASE transition decoding between the *s0* and *s2* transitions but IF decoding for all other transitions.



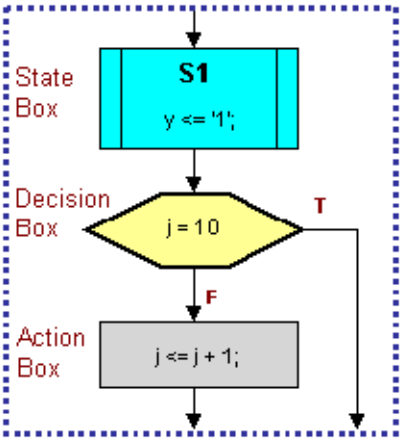
Refer to "State Diagram Notation" on page 30 for more information about the objects that can be used on a state diagram.

Algorithmic State Machines

An algorithmic state machine (ASM) describes the behavior of a system in terms of a defined sequence of operations which produce the required output from the given input data. These sequential operations can be represented using flow chart style notation as an ASM chart.

States are represented by state boxes, conditions by decision or decode boxes and assignments and actions by action boxes.

One state box and any number of decision and action boxes constitute an ASM block:



An ASM block describes the state of the system during one clock pulse interval. It has one entry flow and any number of exit flows. The operations defined in the ASM block are executed during the clock edge transition and the next state is entered as the clock advances.

The ASM chart editor extends this flow between ASM blocks by allowing hierarchical action boxes, hierarchical state boxes and connector objects.

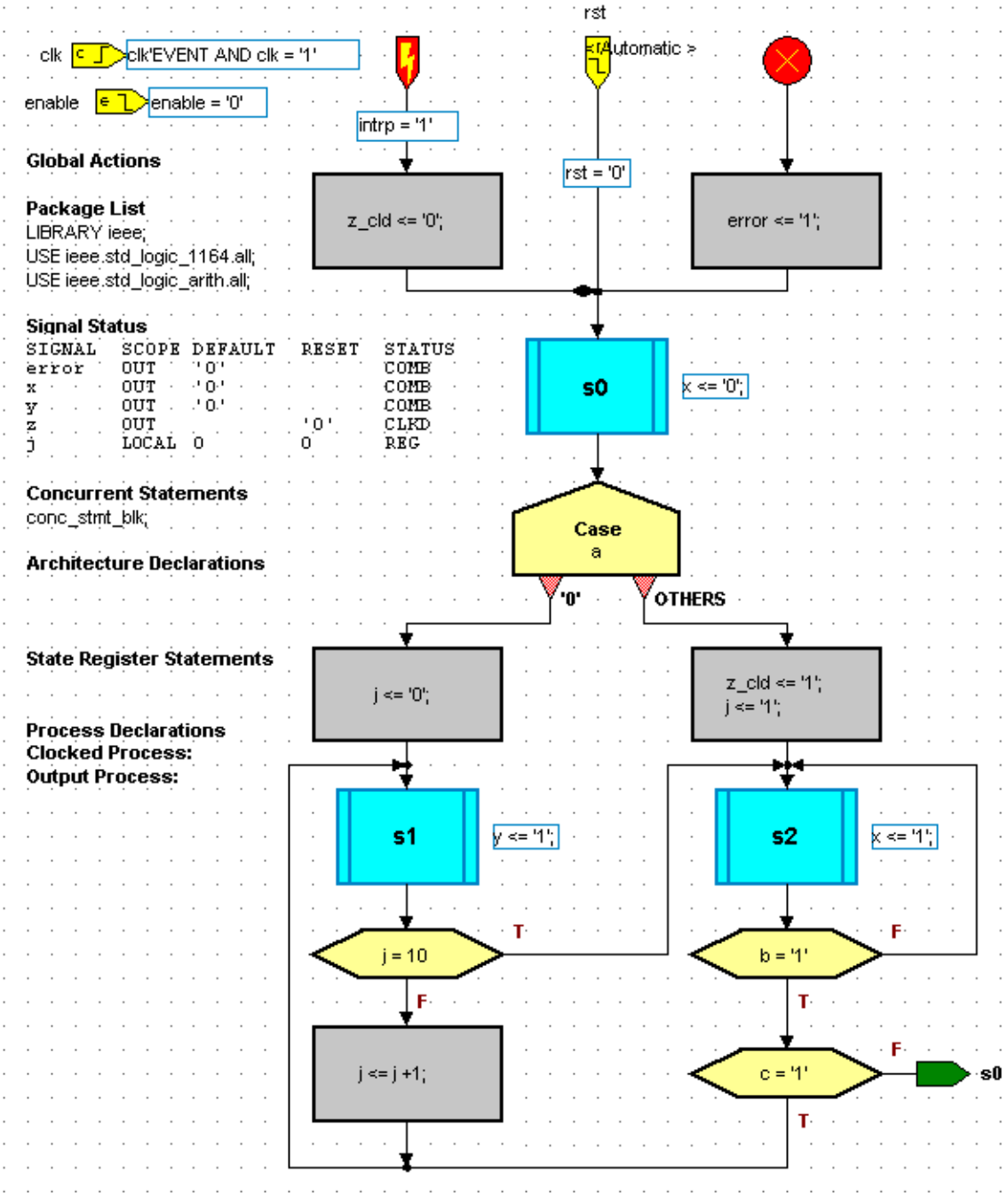
Hierarchical action boxes can be used to represent action logic on a child ASM chart. However, no state boxes can be added to these charts. Decision boxes or decode boxes used on these charts describe the action logic and do not effect the state encoding for the ASM.

A decision box or decode box can be used to decode the next state or for decoding action logic. If all flow branches below a decision or decode box meet, the same state would be reached and the conditions apply only to the action logic.

Hierarchical state boxes can be used to break a large ASM chart into any number of hierarchical ASM charts.

ASM Chart Example

The following picture shows the "State Diagram Example" on page 13 represented by an ASM chart:



Refer to "ASM Chart Notation" on page 86 for more information about the objects that can be used on an ASM chart.

Syntax Notes

The syntax used to define *condition*s, *action*s, *architecture declarations*, *module declarations*, *process declarations* and other HDL statements on a *state diagram* or ASM chart must comply with the language syntax for the *HDL* being used.

Please refer to a VHDL or Verilog language reference manual for full information about HDL syntax.

Conditions, actions, declarations and other statements can be entered as free-format text directly on the diagram or using a dialog box. The syntax is normally checked on entry although you can ignore the syntax warnings or unset the syntax checking preference to allow entry without checking.

You can use an expression builder dialog box to insert ports, signals, values and operators in the correct syntax for the current hardware description language.

A condition is a boolean expression which forms part of an IF or Case statement in the generated HDL. An action or declaration is interpreted as a complete HDL statement and must be terminated by a semi-colon.

Building a HDL Expression

You can display the expression builder dialog box for your active language by using the  button or by choosing **Expression Builder** from the **Edit** menu in a *state diagram*, *ASM chart* or *flow chart*.

The expression builder can be used whenever an input expression (condition) or an output assertion (action) is being edited whether by direct text editing or in a dialog box.

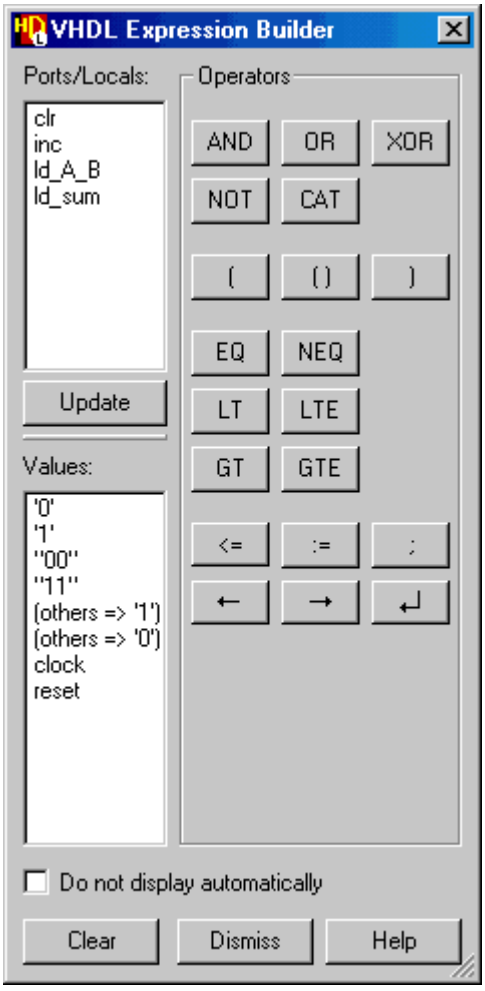
The expression builder provides a palette listing the available ports and locally defined signals together with template values in the syntax required by the active diagram. It also provides buttons which can be used to insert appropriate operators for the active language. (The assignment operators are not shown when editing a condition).

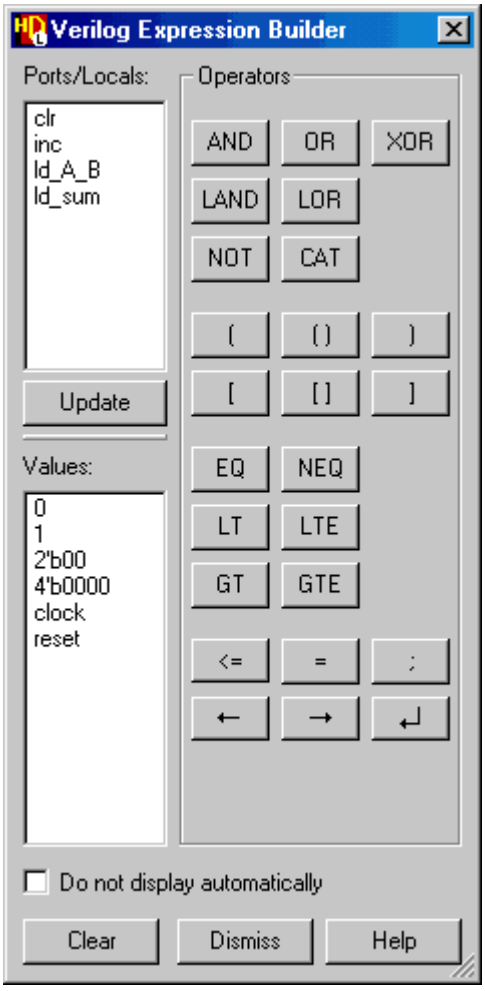
When you are editing a condition, the Ports/Locals list includes all input, bidirectional, clocked output and locally defined signals. When editing actions, it also includes all output and buffer (for VHDL only) signals.

The Values list includes examples of typical values including:

- ‘0’,‘1’, “00” ,“11”, (others => ‘0’) and (others => ‘1’) (when using VHDL)
- 0, 1, 2’b00 and 4’b0000 (when using Verilog)

When you are editing actions the Values list also includes all the assignable signals shown in the Ports/Locals list.





You can build an expression by selecting a signal name in the Ports/Locals list, clicking on an operator button and then selecting a value from the Values list followed by any other required arguments. For example, select the *clr* signal, click the <= assignment operator, select the *'0'* value and then click the **;** operator to enter the action statement:

```
clr <= '1'; (VHDL)
clr <= 1; (Verilog)
```

Complex statements can also be built using the equality and boolean operators. For example, the condition statements:

```
clr = '1' AND inc /= '0' (VHDL)
clr == 1 & inc != 0 (Verilog)
```

You can use the **Ctrl** modifier key to rapidly enter multiple similar expressions such as the following list of signal and variable assignment actions:

```
out2 <= '1';
out3 <= '1';
out4 <= '1';
out5 <= ‘1’;
var0 := '0';
var1 := '0';
var3 := ‘0’;
```

Use the following procedure:

1. Select a default value (which will be used in the assignment statements) from the Values list while holding down the **Ctrl** key. For example: '1' or '0'.
2. Then use **Ctrl**+**Left** mouse click to select each required each entry in the Ports/Locals list.

This procedure generates multiple assignment statements using the default operator, automatically inserting terminating semi-colons and new lines. You can use **Ctrl**+**Left** mouse click again at any time to change the assigned value or assignment type.

You can use the **Update** button to update the list of ports and local signals with any signals added since the expression builder was displayed.

You can use the **Clear** button to clear all text from the field being edited or the **Dismiss** button to hide the expression builder dialog box.

Note

VHDL operators are entered using uppercase if the style option to **Use uppercase VHDL keywords** is set in your preferences or in lowercase if this option is unset.
Note that the expression builder is available when you are editing a non-modal dialog box such as the Object Properties but is not available in a modal dialog box such as Recovery State or CASE Settings.

The expression builder dialog box is normally displayed automatically unless you set the **Do not display automatically** preference in the dialog box. In automatic mode, the expression builder is automatically displayed when a HDL expression is being edited and automatically hidden when the edit is completed

The buttons on the Verilog Expression Builder insert the following operators:

Table 1-1. Verilog Expression Builder

Button	Description	Operator
AND	Bitwise AND operator.	a & b

Table 1-1. Verilog Expression Builder (cont.)

Button	Description	Operator
OR	Bitwise OR operator.	a | b
XOR	Bitwise XOR (exclusive OR) operator.	a ^ b
LAND	Logical AND operator.	a && b
LOR	Logical OR operator.	a || b
NOT	Bitwise negation (NOT) operator.	~a
CAT	Concatenation or aggregate operator.	{a,b}
(	Left parenthesis for grouping.	(a,b)
()	Paired parentheses for grouping around the selected text.	(a,b)
)	Right parenthesis for grouping.	(a,b)
[	Inserts left bracket for element or slice.	[m:n]
[]	Paired bracket for element or slice around the selected text.	[m:n]
]	Right bracket for element or slice.	[m:n]
EQ	Equality operator.	a == b
NEQ	Not equal operator.	a != b
LT	Less than operator.	a < b
LTE	Less than or equal to operator.	a <= b
LT	Greater than operator.	a > b
GTE	Greater than or equal to operator.	a >= b
<=	Non-blocking signal assignment operator.	a <= b
=	Blocking signal assignment operator.	a = b
;	Semi-colon character.	;

Table 1-1. Verilog Expression Builder (cont.)

Button	Description	Operator
←	Deletes the previous character.	
→	Space character.	
↵	New line.	

The buttons on the VHDL Expression Builder insert the following operators:

Table 1-2. VHDL Expression Builder

Button	Description	Operator
AND	Logical or bitwise AND operator.	a AND b
OR	Logical or bitwise OR operator.	a OR b
XOR	Logical or bitwise XOR (exclusive OR) operator.	a XOR b
NOT	Logical or bitwise negation (NOT) operator.	NOT a
CAT	Concatenation operator.	a & b
(	Left parenthesis for grouping, aggregate, element or slice.	(a,b)
()	Paired parentheses for grouping, aggregate, element or slice.	(a,b)
)	Right parenthesis for grouping, aggregate, element or slice.	(a,b)
EQ	Equality operator.	a = b
NEQ	Not equal operator.	a /= b
LT	Less than operator.	a < b
LTE	Less than or equal to operator.	a <= b
GT	Greater than operator.	a > b
GTE	Greater than or equal to operator.	a >= b
<=	Signal assignment operator.	a <= b
:=	Variable assignment operator.	a := b

Table 1-2. VHDL Expression Builder (cont.)

Button	Description	Operator
;	Semi-colon character.	;
←	Deletes the previous character.	
→	Space character.	
↵	New line.	

Condition Syntax

A *condition* is an event (such as a signal changing its value) that may cause a change of state in a state machine.

A condition can be any valid boolean HDL expression and typically has the form:

```
signal relational_operator value
```

where **signal** is an expression containing the name of an input signal to the block representing the state machine on the parent block diagram or IBD view or a locally declared signal in the state machine. When using VHDL, this signal must have a type defined in a VHDL package.

The **relational_operator** can be any valid operator for the current hardware description language including the following simple operators:

VHDL	Verilog	
=	== or ===	equal
/=	!= or !==	not equal

value is typically an integer, real, character, enumerated value (VHDL only), constant (or locally declared constant in a state machine) or any expression which evaluates to one of the above.

When using VHDL, the following values must be specified in a VHDL package for the associated net type:

integer	must be within the bounds constraint of the net type. This value can be a positive or negative number with one or more digits in the range 0 to 9.
real	must be within the bounds of the net type. This value can be a positive or negative number with one or more digits in the range 0 to 9 and a decimal part between 0 to 9.

enumerated value	must be an enumerator of the associated net type.
character	must be a printable character.
constant	must be a non-deferred constant defined in a package or a locally declared constant.

VHDL If conditions are boolean expressions and must result in a true or false value. Typically, this will have the form <name> = <true or false value>. For example: *a = '0'* generates an expression *IF a = '0' THEN*

Verilog If conditions are simply expressions and can be entered directly. For example *a* generates the code *IF a* and *!a* generates the code *IF !a*

Examples of Condition Syntax

The following are examples of some typical VHDL conditions:

```
a ='0'
a = 1.0
a /= red
(a = 1) AND (c = 0)
```

The following are examples of some typical Verilog conditions:

```
!a
a == 1'b1
a != red
(a == 1'b1) && (c == 1'b0)
```

If flow **a** is a VHDL array:

```
a(1) = 0
a(x) = 0
a = (1,0,3)
a(1 to 5) = (3 => 5, others => 0)
```

If flow **b** is a VHDL record (with members **x**, **y**, **z**):

```
b.x = 0
b = (0,0,1)
b = (x => 0, others => 1)
```

A condition entering or leaving a junction on a state diagram is a partial condition. If there are more than one partial conditions between two states, they are combined by performing a logical AND to complete the transition between the states.

Action Syntax

An *action* assigns a value to a signal or variable. Actions can be specified for a *state* or *transition* in a *state diagram* or for an *action box* or *state box* in an *ASM chart*.

An action can be any valid HDL assignment. Every action is a complete HDL statement and must be terminated by the semi-colon *;* character.

When you are using VHDL, a typical action has the format:

```
signal <= value; (signal assignment)
variable := value; (variable assignment)
```

Note that the assignment operators are represented by two characters (**<=** or :=) which must not be separated by a space.

When you are using Verilog, a typical action has the format:

```
signal = value;
```

There are two signal assignment operators in Verilog. The example above shows a blocking assignment (using the **=** operator). You can also specify a non-blocking assignment (using the <= operator) when two or more actions need to be executed concurrently.

signal and **value** have the same syntax as defined for conditions, with the exception that **signal** must be the name of an output signal from the block or component which represents the view on the parent block diagram or IBD view (or the name of a locally declared output signal).

Examples of Action Syntax

The following examples show some typical VHDL action assignments:

```
x <= '1';
a <= "xyz";
x <= x+1;
```

The following examples show some typical Verilog action assignments:

```
x = 1;
a = 10'b0;
```

If flow **a** is a VHDL array:

```
a(1) <= 0;
a(x) <= 0;
a <= (1, 0 3);
a(1 to 5) <= (3 => 5, others => 0);
```

If flow **b** is a VHDL record (with members **x**, **y**, **z**):

```
b.x <= 0;
b <= (0, 0, 1);
b <= (x => 0, others => 1);
```

Declaration Syntax

Signal declarations, constants, variables, comments, procedures, functions or type definitions can be included in an architecture or module declaration. A process declaration may include any of these constructs except for signal declarations.

Typically a VHDL declaration, comprises a keyword (signal, constant or variable) followed by a name, type and value. The specified type must be one of the standard predefined types or a type defined in a VHDL package. An initial value is required when the declaration is a constant but is optional when you declare a signal or variable.

A typical Verilog declaration, comprises a keyword followed by appropriate parameters as given below:

Table 1-3. Verilog Declarations

Keyword	Parameters		
'define	name	value	
parameter	name	value	
reg	range (optional)[1]	name	array (optional)
integer	name	array (optional)	
real	name		
time	name	array (optional)	
wire	range (optional)	name	array (optional)

1. Verilog range and array parameters should be entered in the format [m:n].

Examples of Declaration Syntax

The following examples show some typical VHDL declarations:

```
SIGNAL busb : std_logic_vector (2 DOWNTO 0) := "000";
SIGNAL siga : std_logic;
CONSTANT prd : time := 100 ns;
CONSTANT zero : std_logic_vector := "0000000000";
VARIABLE z: std_logic;
```

The following examples show some typical Verilog declarations:

```
reg  [7:0] busb;
wire siga;
wire [2:0] busc;
parameter prd = 100;
parameter zero = 10'b0;
```

Note

VHDL keywords are usually shown in upper case (for example `CONSTANT`) but Verilog keywords in lower case (for example `parameter`).

Concurrent State Machines

Any number of concurrent state machines with the same interface can be created from within a state diagram or ASM chart.

The package list and any concurrent statements are shared by the concurrent state machines but global actions can be set separately. If there are any other user declarations (interpreted as architecture declarations in VHDL or as module declarations in Verilog) these are also shared by the concurrent state machines.

HDL generation and state encoding characteristics can be set separately for each concurrent state machine.

Each concurrent state machine is given a unique name by adding an integer to the default name (for example, *machine0*, *machine1*, *machine2*..) and identified in the title bar by appending this name and its position in the set of concurrent state machines to the leaf state machine name.

For example, if you create three concurrent state machines for the leaf state machine *DESIGNS\ConcSM\fsm* the resulting set state machines would be identified as follows:

DESIGNS\ConcSM\fsm ['machine0' 1 of 4]
DESIGNS\ConcSM\fsm ['machine1' 2 of 4]
DESIGNS\ConcSM\fsm ['machine2' 3 of 4]
DESIGNS\ConcSM\fsm ['machine3' 4 of 4]

A set of concurrent state machines is treated as a single design object and all the concurrent state machines (including any hierarchical diagrams) are saved when any state diagram is saved.

When HDL is generated for concurrent state machines, separate VHDL processes (or "always" blocks in Verilog) are generated for each machine.

Note

State names must be unique within the set of concurrent state machines.
You cannot create links between concurrent state machines.

Adding a Concurrent State Machine

You can create a concurrent state machine from a state diagram using the **Ctrl**+**F2** shortcut keys or by choosing **Concurrent State Machine** from the **Add** menu.

You can create a concurrent state machine from an ASM chart by choosing **Concurrent ASM** from the **Add** menu.

A new state diagram or ASM chart is created as a new view in the existing window with the same interface as the current diagram.

The package list and any concurrent statements, or status signals list are shared by the concurrent state machines but global actions can be set separately. If there are any other user declarations (which are interpreted as architecture declarations in VHDL or as module declarations in Verilog) these are also shared by the concurrent state machines.

Opening a Concurrent State Machine

You can open an existing concurrent state machine from within a state diagram by choosing **Open Machine** from the **Diagram** menu and selecting from the list of concurrent state machine names.

You can open an existing concurrent state machine from within an ASM chart by choosing **Open ASM** from the **Diagram** menu and selecting from the list of concurrent state machine names.

You can also open a concurrent state machine by selecting it in the *diagram browser*.

Renaming a Concurrent State Machine

You can change the name of the active concurrent state machine by choosing **Rename Concurrent Machine** from the **Diagram** menu to display a Rename dialog box.

You can also rename the active concurrent state machine in the **State Machine** page of the SM Properties dialog box or in the **ASM Diagram** page of the ASM Properties dialog box. Alternatively, you can choose **Rename** from the popup menu when the concurrent state machine name is selected in the *diagram browser* to directly edit the state machine name.

This name is used to uniquely identify concurrent state machines in the generated HDL but can also be specified when there are no concurrent state machines defined. If not specified, the name defaults to the value set in the state machine preferences.

Deleting a Concurrent State Machine

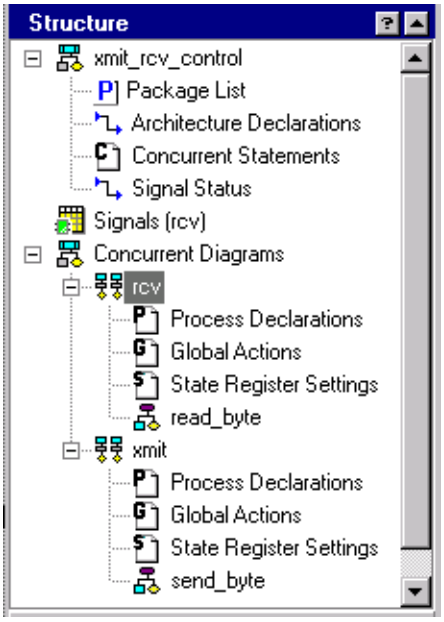
You can delete a concurrent state machine from a set of concurrent state machines by choosing **Delete Machine** or **Delete ASM** from the **Diagram** menu and selecting from the list of concurrent state machine names.

You are prompted for confirmation that the state machine should be deleted. If you delete an open diagram, its window is closed. The titles for all other diagrams in the set of concurrent state machines are updated. However, the *design explorer* view is not updated until you have saved the state machine. Note that you can not delete the last concurrent state machine in the set.

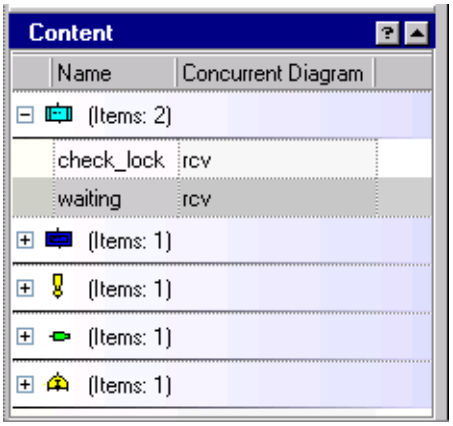
Using the Diagram Browser

You can browse the structure and content of a state machine including any concurrent or hierarchical views and the signals table in the *diagram browser* that is optionally displayed to the left of the main diagram view.

For example, the following picture shows the structure of the ASM chart view for the *xmit_rcv_control* state machine in the *UART* example design:



The following picture shows the content of the *rcv* concurrent state machine which is selected in the Structure pane shown above:



Refer to “The Diagram Browser” in the *Graphical Editors User Manual* for more detailed information about browsing the structure and content of a diagram.

Chapter 2
State Diagram Editor

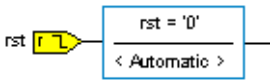
This chapter describes the state diagram editor.

State Diagram Notation

The notation used for objects on a state diagram is shown below:

A *clock point* displays the clock signal name and clock condition in a synchronous state machine. There must be one clock point on a synchronous diagram or none on an asynchronous diagram.

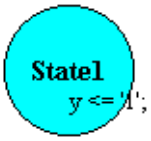
An *enable point* displays an enable signal name and enable condition in a synchronous state machine. There can be one optional enable point on a synchronous diagram or none on an asynchronous diagram.



A *reset point* displays a reset signal name, actions, mode and condition in a synchronous state machine. There can be any number of reset points on a synchronous diagram or none on an asynchronous diagram. A priority is shown if there are more than one reset points with the same mode on the diagram. Each reset point must be connected to a link or state.

A *recovery state point* can be connected to indicate the transition to a recovery state. Recovery actions can be shown on the associated transition.

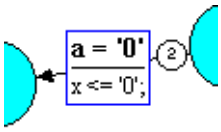
An *interrupt point* is an implicit connection to all states on the diagram. It has an associated interrupt transition which may have a priority if there are more than one interrupt points on the diagram. The interrupt transition condition has priority over all other conditions.



A *simple state* represents observable status that the state machine can exhibit at a point in time. Encoding information is shown if manual encoding is enabled and there may be associated state actions.

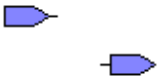
A *wait state* introduces a delay in a synchronous state machine. The delay is specified by an integer or expression that evaluates to a number of clock cycles. Encoding information is shown if manual encoding is enabled and there may be associated state actions.

A *hierarchical state* represents a child state diagram within a hierarchical state machine. A hierarchical state has no associated state actions.



A *transition* is the change of state that occurs when an associated condition is satisfied. Transitions may have transition actions. A transition priority is shown if more than one transition leaves the same state.

A *junction* represents a connection to transitions which are common to more than one state.



The *entry point* and *exit point* are the connections in a child state diagram to the parent hierarchical state.

A named *link* represents a direct transition to the named state or junction.

Pre and Post Global Actions	A statement block listing *global actions* that are always performed at the beginning of the output process for combinatorial signals or after the clock for registered signals.
Concurrent Statements	A statement block containing a list of *concurrent statements* that are included in the generated HDL.
Architecture or Module Declarations	A list of user defined VHDL *architecture declarations* or Verilog *module declarations*.
Signals Status	A table showing the *signals status* of output and locally declared signals.
State Register Statements	A statement block containing a list of statements which are included in the generated HDL as *state register statements*.
Process Declarations	Separate lists of VHDL *process declarations* which are included in the clocked and output processes.

Note

You cannot add a clock point, reset point, enable point or wait state in an asynchronous state machine.

A *clock point* displays the clock signal name and clock condition in a synchronous state machine. There must be one clock point on a synchronous diagram or none on an asynchronous diagram.

An *enable point* displays an enable signal name and enable condition in a synchronous state machine. There can be one optional enable point on a synchronous diagram or none on an asynchronous diagram.

State Diagram Toolbar

The following commands are available from the State Diagram Tools toolbar:

Table 2-1. State Diagram Toolbar

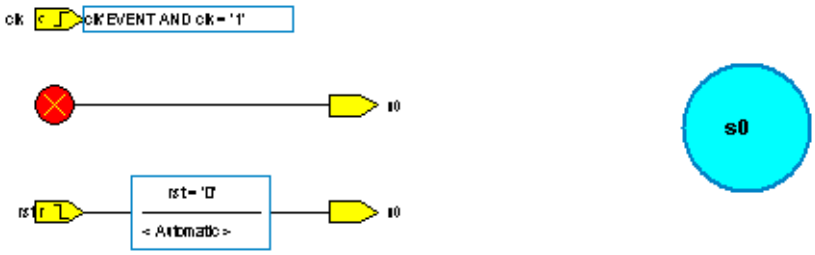
Button	Description
	Select text or objects
	Select text only
	Select objects only
	Add or modify comment text
	Pan the window
	Add a clock point
	Add a reset point
	Add a recovery state point
	Add an enable point
	Add a state
	Add a transition
	Add an interrupt point (not available in child hierarchical diagrams)
	Add a link
	Add a junction
	Add a hierarchical state
	Add a wait state
	Add an entry point (available only in child hierarchical diagrams)
	Add an exit point (available only in child hierarchical diagrams)
	Add a panel

Refer to the *HDL Designer Series User Manual* for general information about toolbars and the HDL Designer Series user interface.

Refer to the *Graphical Editors User Manual* for information about selecting objects, adding comment text, panning the window, adding a panel and additional toolbars which are common to the other graphic editors.

State Machine Initialization

A new *state diagram* is initialized with a default *clock point*, *recovery state point*, *reset point* and *simple state* as shown below:



There must be a single *clock point* on a synchronous state diagram although it can be deleted if you want to make the state diagram asynchronous. The *clock point* is always shown unconnected and the clock signal is used for all state transitions in the state diagram.

The *recovery state point* and *reset point* are connected by *transitions* to *links* which reference the *simple state*.

There can be only one *recovery state point* although it can be deleted if not required. If there is no transition connected to a recovery state point on a synchronous state diagram, the state that is connected to the primary reset is the default recovery state.

You can add one *enable point* and any number of *reset points* on a synchronous state diagram.

Adding Objects on a State Diagram

You can add objects on a *state diagram* using the **Add** menu or one of the buttons in the State Diagram Tools toolbar. Some objects can also be added using a shortcut or mnemonic key. Refer to the **Quick Reference Index** which can be accessed from the Help and Manuals tab of the HDS InfoHub for a list of supported Graphical Editor Shortcut Keys. To open the InfoHub, select **Help and Manuals** from the **Help** menu.

The cursor changes to a cross-hair which allows you to add the object by clicking at the required location on the diagram.

After adding an object, the command normally repeats until you use the **Esc** key (or **Right** mouse button) to terminate the command. However, you can set a preference for the command

to remain active or activate only once and you can toggle this mode for the current command by using the **Ctrl** key.

Adding a Clock Point

You can add a *clock point* in a synchronous state machine when there is no existing clock point by using the  button or by choosing **Clock Point** from the **Add** menu.

Note

There can be only one *clock point* in a synchronous state machine which must be on the top level hierarchical diagram. You can not add a clock point on an asynchronous state machine.

The clock point is added with the default clock signal name *clk*, default *rising* edge and default condition (*clk'EVENT and clk = '1'* for VHDL or *posedge clk* for Verilog).

Tip: Note that the clock edge is indicated by a rising or falling edge on the clock point icon.

You can change the clock signal name (and the condition when a user-specified condition has been entered) by clicking to select the signal name text and clicking again to edit the text in-line.

Alternatively, you can double-click on the clock point, use the  button or choose **Object Properties** from the **Edit** menu, to display the **Clock** page of the SM Object Properties dialog box as described in "Editing Clock Object Properties" on page 49.

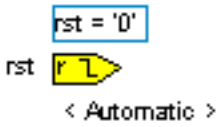
Adding a Reset Point

You can add a *reset point* in a synchronous state machine by using the  button or by choosing **Reset Point** from the **Add** menu.

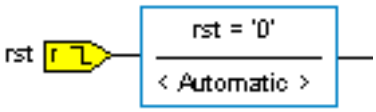
Note

There can be any number of *reset point*s in a synchronous state machine or none in an asynchronous state machine. Reset points can only be added on the top level diagram of a hierarchical state diagram and must be connected (directly or using a link) to a state.

The reset point is added with the default reset signal name *rst* (or *rstN* if reset points already exist on the diagram), *low* reset level (*rst = '0'* for VHDL or *!rst* for Verilog) and *automatic* reset signal actions.



The reset condition and actions are automatically assigned to the transition when the reset is connected to a link or a state.

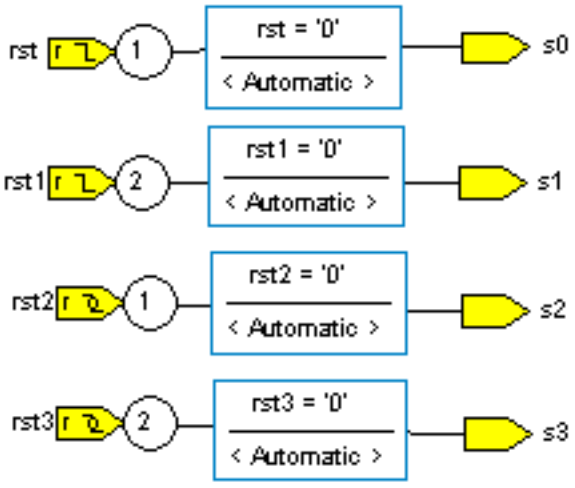


You can change the reset signal name by clicking to select the signal name text and clicking again to edit the text in-line.

Alternatively, you can double-click on a reset point (or the transition attached to a reset point), use the button or choose **Object Properties** from the **Edit** menu, to display the **Resets** page of the SM Object Properties dialog box as described in "Editing Reset Object Properties" on page 50.

You can change the reset mode in the dialog box or by choosing **Synchronous** or **Asynchronous** from the **Reset Mode** cascade in the popup menu.

Where more than one reset with the same mode is defined on the diagram, their evaluation order is determined by the priority. However asynchronous resets take priority over all synchronous resets. For example, the following picture shows two asynchronous resets (*rst* and *rst1*) and two synchronous resets (*rst2* and *rst2*):



Tip: Note that the reset signal level and mode are indicated on the reset point icon.

The reset actions are automatically derived by default from the reset signal status but can be edited directly when specified actions are set in the Object Properties dialog box. The actions syntax is automatically checked for the current hardware description language on entry.

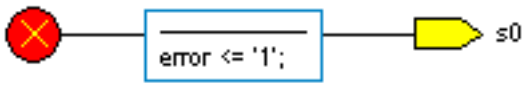
Adding a Recovery State Point

You can add a *recovery state point* to a synchronous state machine using the button or by choosing **Recovery State Point** from the **Add** menu.

Note

There can only be one recovery state point and this command is not available if a recovery state point already exists on the diagram. A recovery state point can only be added on the top level diagram of a hierarchical state diagram. However a separate recovery point can be used for each diagram in a set of concurrent state machines.

The recovery state point should be connected by an unconditional transition to a state (or link that references a state) which is entered when no other valid state is recognized in the next state process or *always* code. The recovery state transition can optionally include recovery actions. These would typically be assignments to signals or variables and Verilog system tasks or VHDL assert and report statements.



If there is no recovery state point, the recovery state is the state connected to the highest priority reset state.

Adding an Enable Point

You can add an *enable point* to a synchronous state diagram using the button or by choosing **Enable Point** from the **Add** menu.

Note

There can only be one enable point and this command is not available if an enable point already exists on the diagram. An enable point can only be added on the top level diagram of a hierarchical state diagram.

The enable signal and condition applies to the entire ASM chart and does not connect to any other objects on the diagram.

You can change the enable signal name (and the condition when a user-specified condition has been entered) by clicking to select the signal name and clicking again to edit the text in-line.

Alternatively, you can double-click on the enable point, use the button or choose **Object Properties** from the **Edit** menu, to display the **Enable** page in the ASM Object Properties dialog box as described in "Editing Enable Object Properties" on page 51.

Adding a State

You can add a *simple state* using the button, **F3** or **S** shortcut key or by choosing **State** from the **Add** menu.

You can add a *wait state* to a state diagram using the button, **F8** or **W** shortcut keys or by choosing **Wait State** from the **Add** menu.

Note

Wait states require a clock signal and cannot be added in an asynchronous state machine.

You can add a *hierarchical state* to a state diagram using the button, **Shift**+**F2** or **H** shortcut keys or by choosing **Hierarchical State** from the **Add** menu.

For example, the following picture shows a simple state (*s1*), wait state (*s2*) and a hierarchical state (*s3*):

A state can have the shape of a circle, ellipse or double circle.

You can change the shape of a simple or wait state by choosing autoshapes from the state popup cascade menu to display the choose shape dialog.

Alternatively, you can double-click on the state, use the button or choose **Object Properties** from the **Edit** menu, to display the **States** page of the SM Object Properties dialog box as described in "Editing State Object Properties" on page 52.

You can change the name of a state (and the state actions or the wait statement for a wait state) by clicking to select the text and clicking again to edit the text in-line.

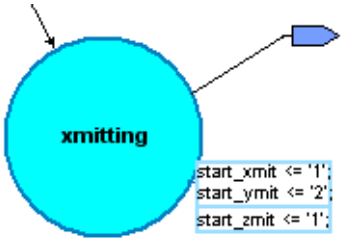
Alternatively, you can double-click on the state, use the  button or choose **Object Properties** from the **Edit** menu, to display the **States** page of the SM Object Properties dialog box as described in "Editing State Object Properties" on page 52.

If you do not change the name of a state, each new state is given a unique name by adding an integer to the default name (for example, *s0*, *s1*, *s2*..). The default base names for new states, default actions and default size can be changed by setting preferences.

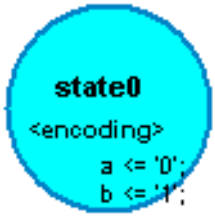
If the state name would overlap the outline, the state is auto-sized to fit the state name.

If default state *action*s are defined in your preferences, these actions are placed below the state name and may overlap the state outline but can be independently moved away from (or into) the state. If you want to contain all your state actions inside the state outline, it may be necessary to resize the object.

Exit and entry *action*s can be defined in your preferences. These actions are also placed below the state and each of the three types of actions is outlined by a blue border.



If **Manual** state encoding mode is enabled in the **Encoding** page of the State Machine Properties dialog box, the default value *<encoding>* is written below the state name. This value can be edited by direct text editing or by using the **States** page of the SM Object Properties dialog box. For example, the following state has encoding enabled and state actions:



Copying State Actions

You can copy an individual action by using the normal **Copy** and **Paste** commands.

You can also copy all the actions associated with a state by selecting a state and using the **Copy** (or **Cut**) option to copy the actions into the paste buffer.

Then select a destination state (or states) and choose the **Paste State Actions** option from the **Paste Special** popup menu.

Adding a Transition

You can add *transition*s using the  button, **F7** or **T** shortcut keys or by choosing **Transition** from the **Add** menu.

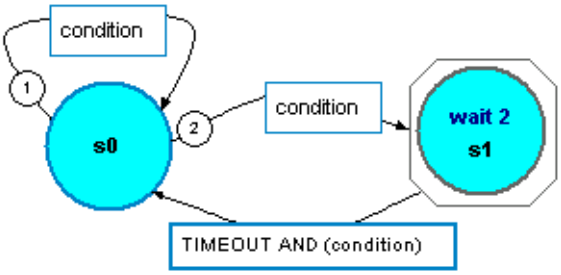
The cursor changes to a cross-hair which allows you to add a transition arc by clicking the **Left** mouse over a source and destination plus any number of *route point*s. The *source* can be any *state*, *junction*, *interrupt point*, *reset point*, *recovery state point* or *entry point* and the *destination* can be any *state*, *junction*, *link* or *exit point*. Note that the arrow head is hidden when a transition arc is attached to a link or exit point.

A loopback transition can be drawn by starting and ending the transition on the same state. (Click twice inside the state to create a loop back without route points or you can specify two or more route points outside the state.)

New transitions are usually added with the default *condition* string *condition* but no *action*s. However, preferences can be set for the default condition and actions strings. By default, *transition arc*s are drawn as curved splines but the default can be changed to orthogonal polylines by setting a preference.

If the origin of a transition is on a wait state, the transition condition defaults to the value *TIMEOUT AND (condition)*.

The following example shows a loopback transition on *s0* and a transition to the wait *state1* with a default TIMEOUT condition on the return transition:



You can change an existing transition condition by clicking on the condition to select the text and clicking again to edit the text. Similarly, you can change the transition actions by clicking on the actions text.

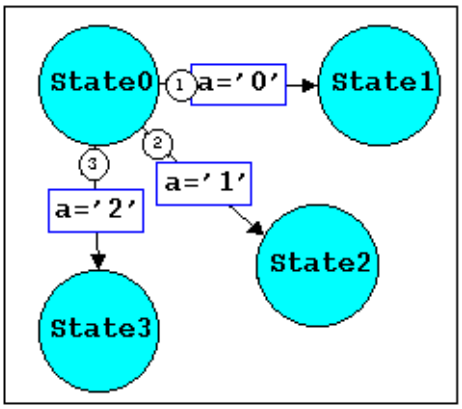
Alternatively, you can double-click on the transition, use the  button or choose **Object Properties** from the **Edit** menu, to display the **Transitions** page in the SM Object Properties dialog box as described in “Editing Transition Object Properties” on page 54.

You can remove route points from transitions by right-clicking a transition and choosing **Remove Route** or **Remove All Routes** options. Alternatively, you can choose **Add a Route** to add a route point to the transition.

Transition Priority

A *transition priority* is assigned to each transition when there are more than one transitions leaving a state. However, no priority is shown when there is only one transition from a state.

If a transition exists with the condition (*others*), it is always evaluated last and no transition priority is displayed. In the following example, the condition *a='0'* is evaluated before the conditions *a='1'* or *a='2'*.



The initial priority is determined by the order in which you add the transitions but can be edited using the Object Properties dialog box or by direct editing on the diagram.

Changing the Direction of a Transition

You can change the direction of one or more selected transitions by choosing **Reverse Direction** from the **Diagram** or popup menu.

Copying Transition Conditions and Actions

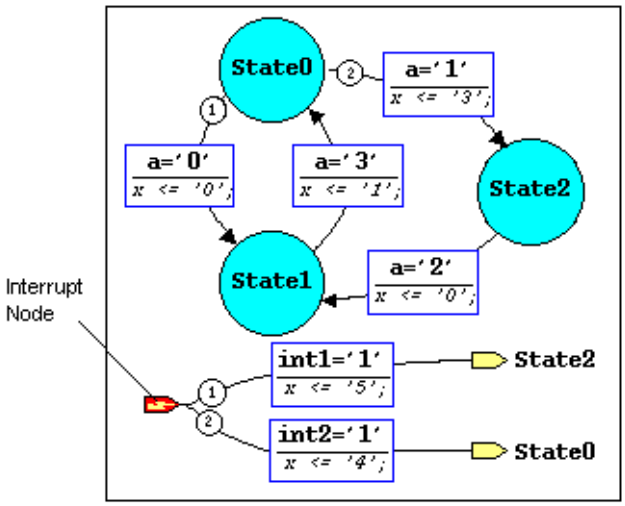
You can copy transition conditions or actions by selecting an individual condition, an action block or the blue box enclosing both the condition and actions and then using the normal **Copy** (or **Cut**) and **Paste** commands.

You can also copy all the conditions and actions associated with a transition by selecting the transition and using the **Copy** (or **Cut**) option to copy the condition and actions into the paste buffer. Then select the destination transition arc (or arcs) and choose **Paste Transition Condition/Actions** from the **Paste Special** popup menu.

Adding an Interrupt Point

You can add *interrupt points* using the  button or **I** shortcut keys or by choosing **Interrupt Point** from the **Add** menu.

A transition from an Interrupt point is a global interrupt which applies to the whole diagram and has priority over all other transitions. In the following example, there are two interrupt conditions from the interrupt point with transition actions which are executed if either of the conditions is satisfied.



Where more than one interrupt is defined on the same diagram, their evaluation order is determined by the transition priority.

Interrupt masking can be achieved by an AND expression combining the condition with a mask signal. This mask signal may control a single interrupt or act as an enable mask for a number of interrupts.

Note

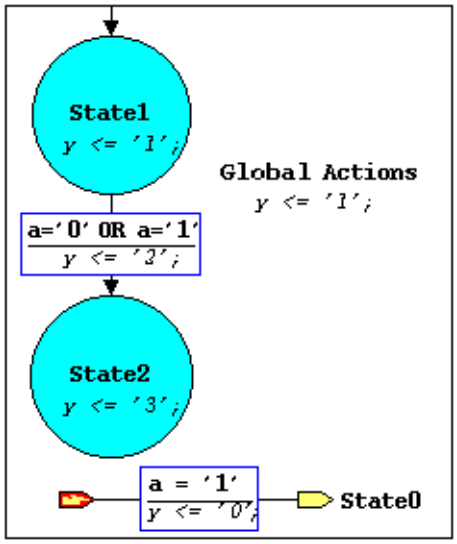
Interrupt points can only be added on the top level diagram of a hierarchical state diagram.

Execution Priority

When there are conflicting actions defined, the following execution priority is observed:

1. Interrupt transition actions
2. Normal transition actions
3. State actions
4. Global actions
5. Default values

In the following example, if *a* is not equal to '*0*' or '*1*', *y* is set to '*1*' by the *State1* actions. If *a* changes to '0', *y* is set to '*2*' by the transition actions.



When a clock edge occurs, *y* is set to '*3*' by the *State2* actions (or reverts to '*1*' if a changes to any value other than '*0*' or '*1*' before the clock edge occurs).

However, if *a* changes to '*1*', the interrupt transition takes priority and *y* is set to '*0*' (and the state changes to *State0* at the next clock edge).

If no conditions are satisfied, the global actions ensure that *y* is set to a known value ('*1*') at the clock edge. Explicit default values can also be set for registered output (and locally declared) signals.

Refer to "Setting Statement Blocks" on page 73 for information about setting global actions and "Setting State Diagram Internal Signal Names" on page 78 for information about setting default values.

Adding a Link

A *link* can be used as a connector to a *state* or a *junction* with the specified name on the same diagram (or another diagram in the same hierarchical state diagram) to avoid long transition arcs.

For example, links are used for the transition conditions shown in the example on the previous page.

You can add links to a state diagram using the  button, **F4** or **L** shortcut keys or by choosing **Link** from the **Add** menu.

Note

You can rotate a link clockwise by choosing **90**, **80**, or **270** from the **Rotate** cascade in the popup menu.

You can change the target of a link by clicking on the current link target to select the text and clicking again to edit the text.

Alternatively, you can double-click on the link, use the button or choose **Object Properties** from the **Edit** or popup menu, to display the **Links** page in the SM Object Properties dialog box as described in "Editing Link Object Properties" on page 56.

Adding a Junction

A *junction* can be used to factor out *condition*s (or *partial condition*s) or actions which are common to more than one *transition*s.

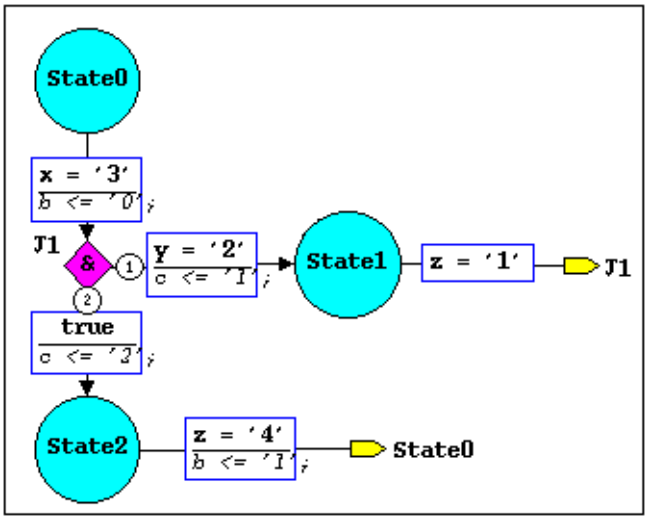
The transitions are replaced by a simpler set of *partial transition*s between the same states. These partial transitions are expanded (using a logical AND, IF or CASE) when HDL is generated for the diagram.

You can add junctions using the button or ***J*** shortcut key or by choosing **Junction** from the **Add** menu.

Junctions have no default name although you can set a default name by setting a preference. You can change the name of a junction by clicking on an existing name to select the text and clicking again to edit the text.

Alternatively, you can double-click on the junction, use the button or choose **Object Properties** from the **Edit** or popup menu to display the **Junctions** page in the SM Object Properties dialog box as described in "Editing Junction Object Properties" on page 58.

In the following example, the condition *(x='3')* which applies to the transitions to both *State1* and *State2* is connected using a junction *J1*.



Links are used to show the return connections to *J1* and *State0*.

Hierarchical State Diagrams

A state diagram may have a number of hierarchical state diagrams. Each child state diagram is represented by a hierarchical state in its parent diagram.

You can open down into a child state diagram by double-clicking on a hierarchical state or by choosing **Open Down** from the **Open** cascade of the **File** menu (or popup menu).

The child state diagram is opened in the existing window. A new child state diagram comprises an *entry point*, a single *state* and an *exit point* connected by *transitions*.

You can edit a hierarchical state diagram in the same way as any other state diagram including more hierarchical states as well as any other state diagram objects (with the exception of interrupt points). Named links can be used between any state (or junction) in the hierarchical state diagram.

Note

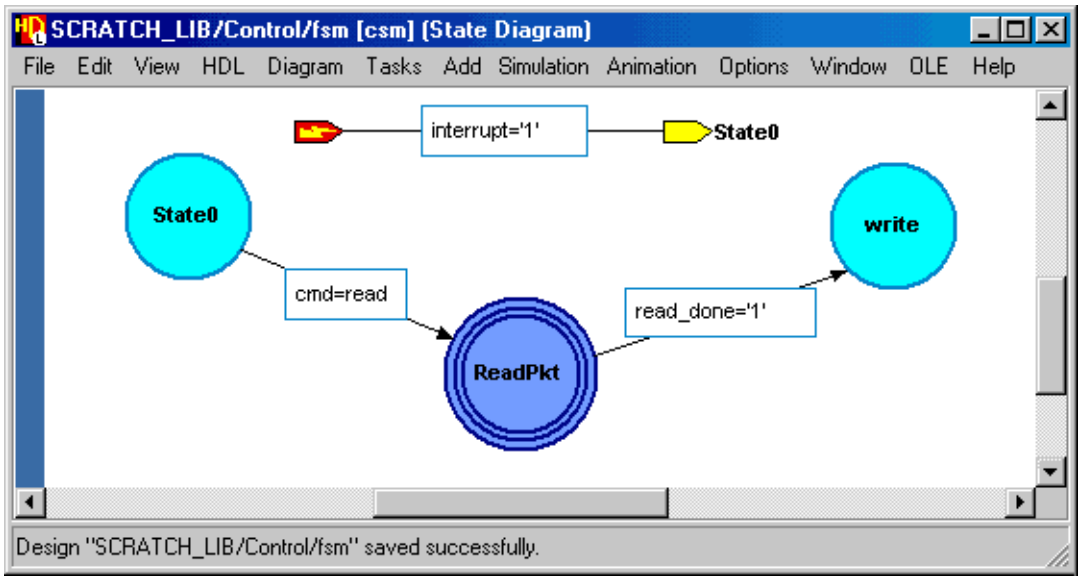
Although links may be used in hierarchical states, exits points should be used to exit from hierarchical states. A hierarchical state should have at least one exit point.

An interrupt point can be added on the top level diagram and is treated as a global interrupt for the whole state diagram.

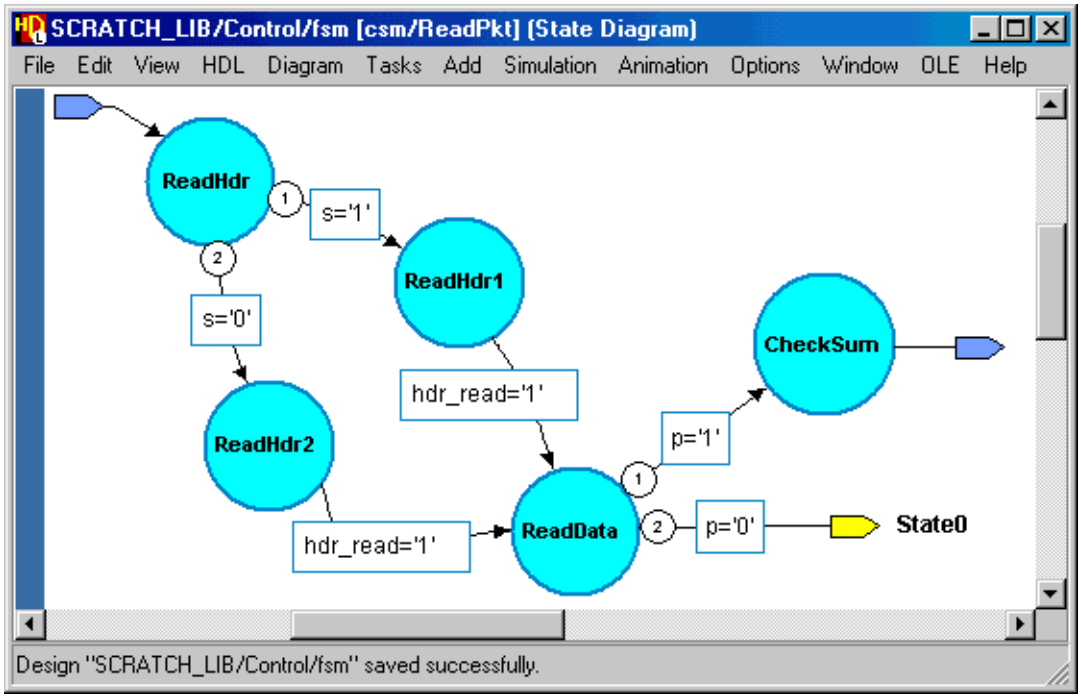
You can choose **Open Up** from the **File** menu or select the name of the parent diagram in the *diagram browser* to open the parent of the currently active state diagram.

Child state diagrams are saved as part of the parent state diagram and named after the parent hierarchical state by adding the parent hierarchical state name to the name of the state diagram.

For example, the parent state diagram in the following picture is named *csm* and the child is named *csm/ReadPkt*:



The child diagram is entered when the entire condition on a transition connected to the hierarchical state is satisfied (*cmd=read* in this example).



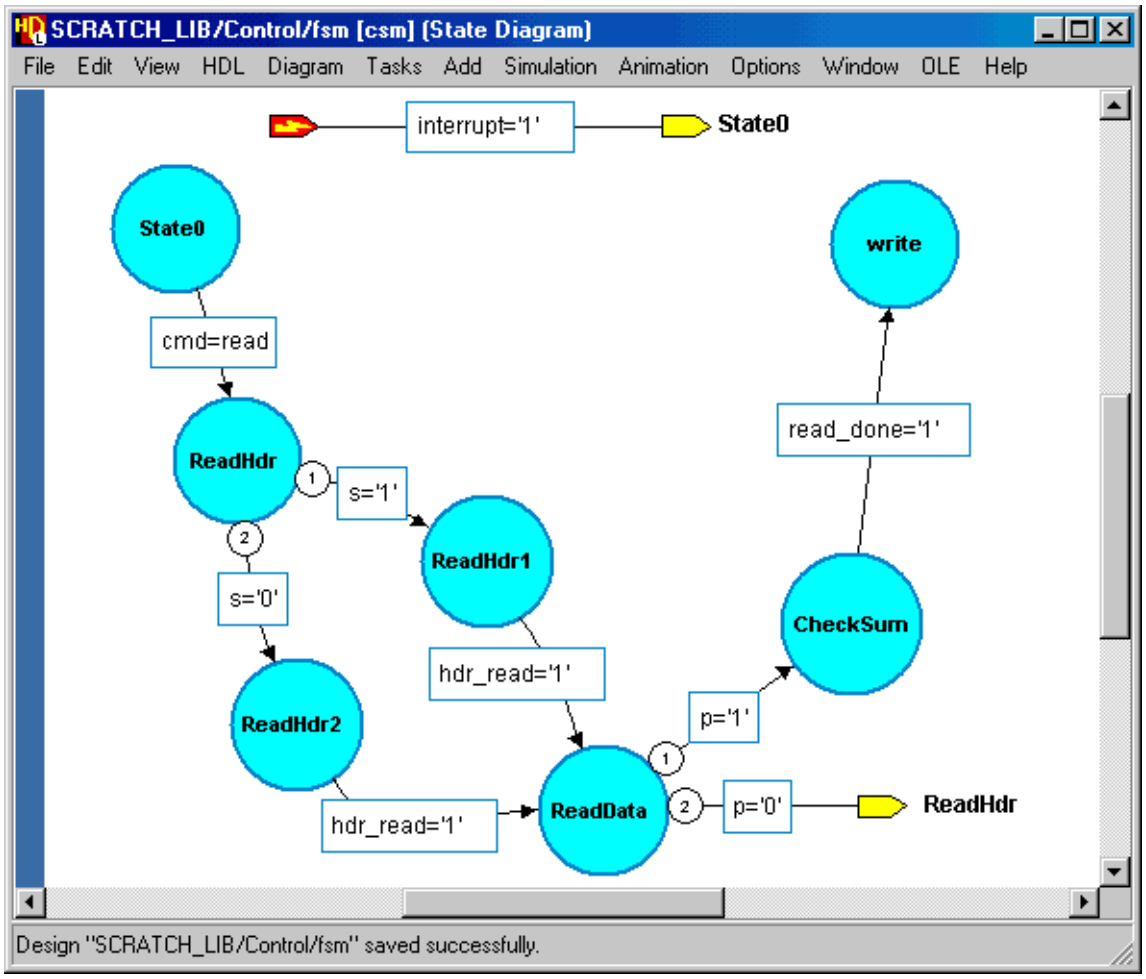
Adding or Removing Hierarchy

You can re-level a state diagram by choosing the **Add Hierarchy** or **Remove Hierarchy** command from the **Re-level** cascade in the **Diagram** or popup menu in a state diagram.

Adding hierarchy replaces the selected states by a new hierarchical state and moves the selected states into the child hierarchical state diagram.

There can be no more than a single state in the selection with transitions entering from the rest of the diagram and there can be no more than a single state with transitions leaving to the rest of the diagram. These connections are represented by transitions to the single entry point and exit point in the child diagram.

Removing hierarchy deletes the selected hierarchical state and replaces it by the objects in the child hierarchical state diagram. For example, the following picture shows the child diagram on the previous page merged into its parent diagram:



The relative placement of the new objects is preserved and centered on the original position of the hierarchical state. They may therefore overlap existing objects on the parent diagram and it may be necessary to re-arrange objects on the diagram.

If the child diagram included other hierarchical states, their hierarchy is retained but can be removed by another re-level operation.

There can be no more than one entry point on the child diagram with no more than one connected transition. This transition must not have any conditions or actions.

There can be no more than one exit point on the child diagram with no more than one connected transition. This transition must not have any conditions or actions.

The entry and exit transitions are each represented by a single connections on the parent diagram.

Adding an Entry Point

An entry point is automatically created when you create a child state diagram (by opening down from its parent hierarchical state) but adding multiple entry points can help reduce diagram complexity.

You can add an *entry point* to a child state diagram in a hierarchical state machine using the  button, **F6** or **E** shortcut keys or by choosing **Entry Point** from the **Add** menu.

Adding an Exit Point

An exit point is automatically created when you create a child state diagram by opening down from its parent hierarchical state but adding multiple entry points can help reduce diagram complexity.

You can add *exit point*s to a child state diagram in a hierarchical state machine using the  button, **Shift**+**F6** or **X** shortcut keys or by choosing **Exit Point** from the **Add** menu.

Each exit point connects to the parent hierarchical state but you can also exit from a diagram using named links that connect to a other state or junction at any level in the hierarchical state machine.

Changing Objects on a State Diagram

You can change an object on a state diagram to another object type by selecting the object (or objects) and using one of the **Change To** options from the **Diagram** menu.

Object	Change To

state	hierarchical state, junction, interrupt point, entry point, link, exit point
hierarchical state	state, junction, interrupt point, link
junction	state, hierarchical state, interrupt point, entry point, link, exit point
interrupt point	state, hierarchical state, junction, link
entry point	state, hierarchical state, junction, link, exit point
link	state, hierarchical state, junction, interrupt point, entry point, exit point
exit point	state, hierarchical state, junction, entry point, link

You cannot change an object if any connected transitions would need to be changed. For example, you cannot change an object which is the source of a transition into an exit point, or a destination object into an entry point.

You cannot change any object to an entry or exit point on a top level state diagram or any object to an interrupt point on a child hierarchical state diagram.

If you change a state which has associated actions to any other object, the state actions are discarded. However, if the new object is a hierarchical state, the state actions are associated with the new default state in the child state diagram.

If you change a hierarchical state, the child state diagram (if it exists) and all its contents are discarded.

The state type (*simple state*, *hierarchical state*, *start state* or *wait state*) can also be changed using the **States** page of the SM Object Properties dialog box.

Adding Other Objects on a State Diagram

You can also add other objects such as a title block, *comment text*, *comment graphics* and *panel*s on a state diagram.

Refer to the *Graphical Editors User Manual* for information about adding these objects and general editing procedures which apply to all the *graphical editor*s.

Editing State Diagram Object Properties

You can edit many properties (including condition expressions and actions) directly on the diagram by clicking to select the text and clicking again to edit the text object.

An expression builder dialog box is automatically displayed when you begin to enter a condition expression or action statement. Refer to "Building a HDL Expression" on page 16 for more information about the expression builder.

The HDL syntax for expressions and actions is automatically checked for the language of the diagram you are using (VHDL or Verilog) although the syntax checking can be disabled by unsetting a preference.

Note

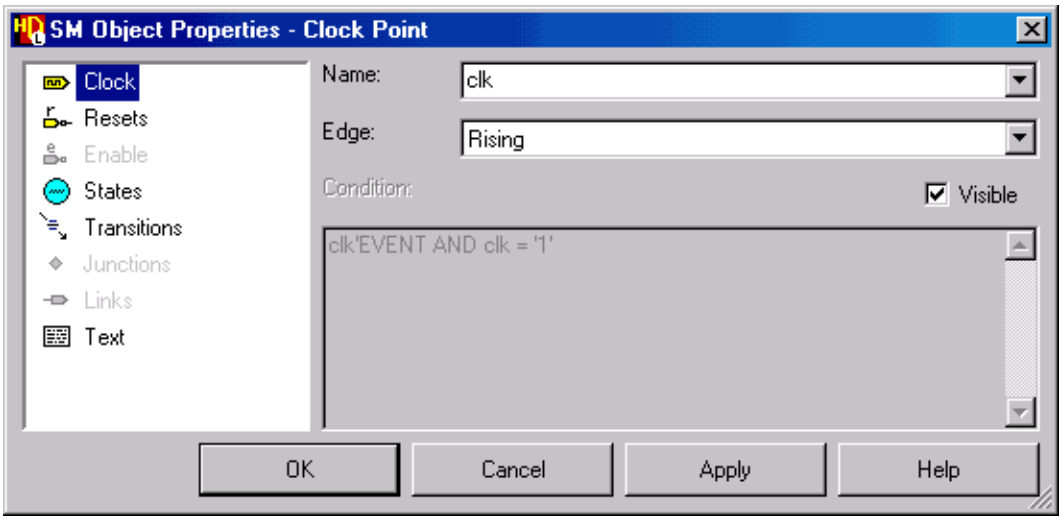
Note that you must include a terminating semi-colon after action statements although line breaks and indents can be used to improve legibility.

You can also edit the properties for an object on a state diagram by double-clicking on the object, using the button or choosing **Object Properties** from the **Edit** menu or popup menu to display an Object Properties dialog box.

The dialog box has separate pages for **Clock**, **Resets**, **Enable**, **States**, **Transitions**, **Junctions**, **Links** and **Text** objects. The editable objects are shown in the left pane of the dialog box. Objects which exist in the current selection set are highlighted in yellow. Objects that are not available in the current selection are shown in dimmed font.

Editing Clock Object Properties

The **Clock** page of the SM Object Properties dialog box allows you to specify the clock signal and set the clock edge sensitivity.



You can choose the clock signal name from a dropdown list of available input signals. Note that any signals starting with *clk* or *clock* take precedence in the list. For a Verilog view, you can choose *Rising* or *Falling* representing *posedge* or *negedge* sensitivity. For a VHDL view, you

can choose *Rising*, *Falling*, *Rising Last*, *Falling Last*, *Rising Edge* or *Falling Edge*. These options generate the following VHDL expressions:

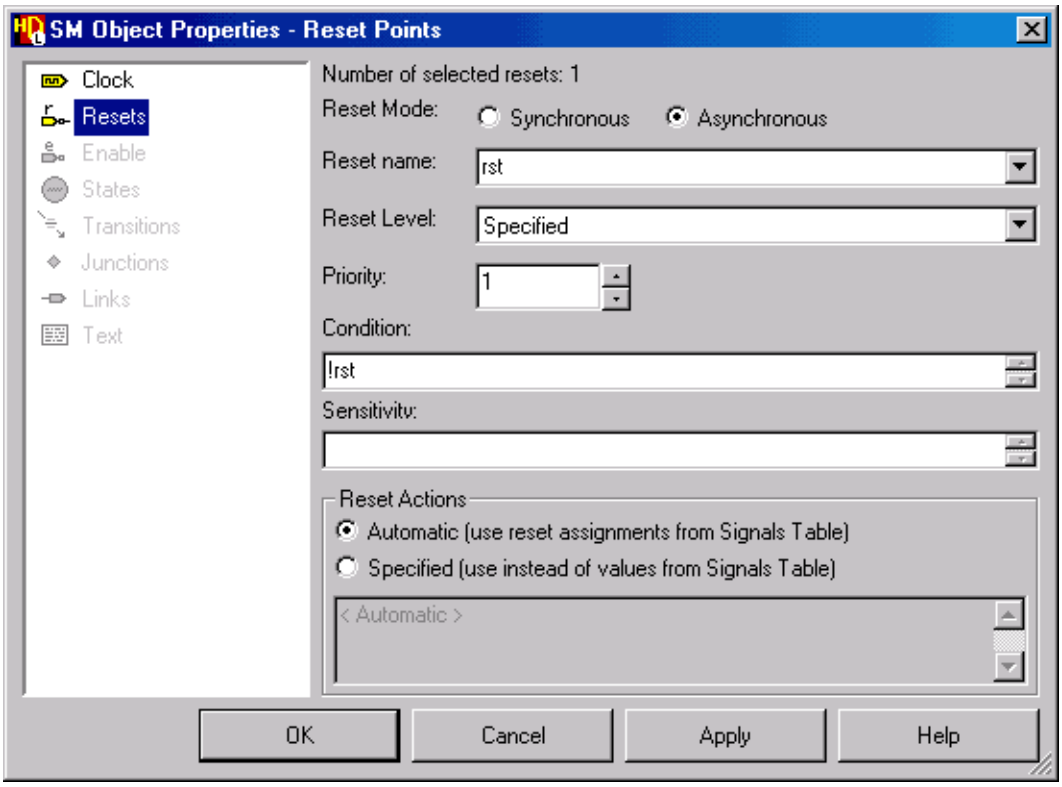
Rising	clk'EVENT AND clk = '1'
Falling	clk'EVENT AND clk = '0'
Rising Last	clk'EVENT AND clk = '1' AND clk'LAST_VALUE = '0'
Falling Last	clk'EVENT AND clk = '0' AND clk'LAST_VALUE = '1'
Rising Edge	rising_edge(clk)
Falling Edge	falling_edge(clk)

Tip: Note that the clock edge is indicated by a rising or falling waveform on the clock point icon.

Alternatively for either language, you can choose *Specify* to enter any other valid edge condition.

Editing Reset Object Properties

The **Resets** page of the SM Object Properties dialog box allows you to specify a synchronous or asynchronous mode reset and specify the reset signal.



You can choose the reset signal name from a dropdown list of available input or locally declared signals. Note that any signals starting with *rst* or *reset* take precedence in the list.

You can specify whether the reset signal is active low, high or when a specified condition is evaluated.

Tip: Note that the signal level and mode are indicated on the reset point icon.

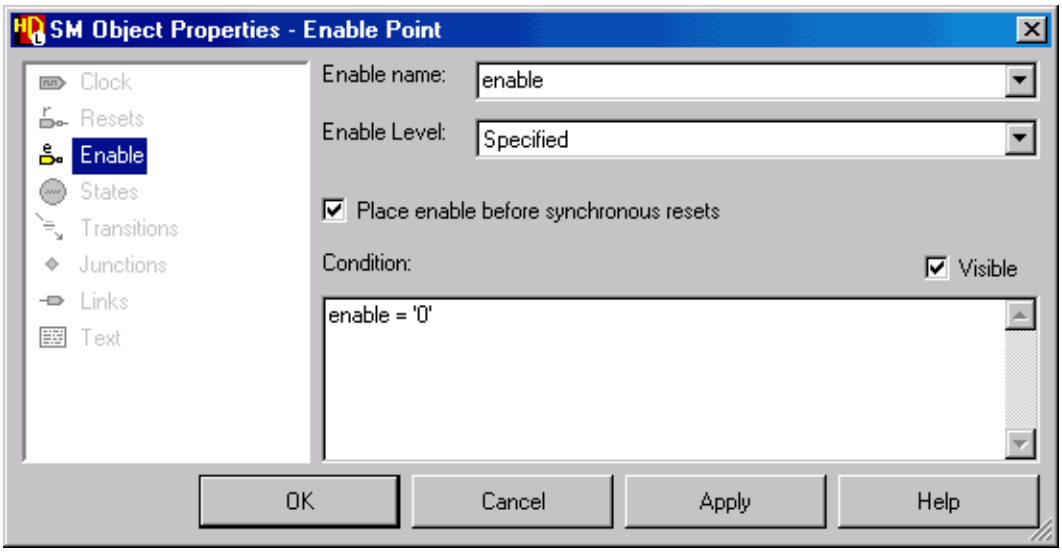
If multiple resets with the same mode are defined on the diagram you also can set the reset priority. However, asynchronous resets take priority over all synchronous resets.

You can optionally specify reset actions. If set to *<Automatic>*, the reset actions are automatically derived from the reset values specified in the signals status table. Refer to "Signals Table" on page 127 for information about setting reset values in the signals status.

If you have specified a Verilog reset condition, you must also specify any additional signals required in the sensitivity list. (Multiple signals should be separated by an *OR* operator.)

Editing Enable Object Properties

The **Enable** page of the SM Object Properties dialog box allows you to specify an enable signal and set the enable signal level.



You can choose the enable signal name from a dropdown list of available input signals. Note that any signals starting with *en* or *enable* take precedence in the list.

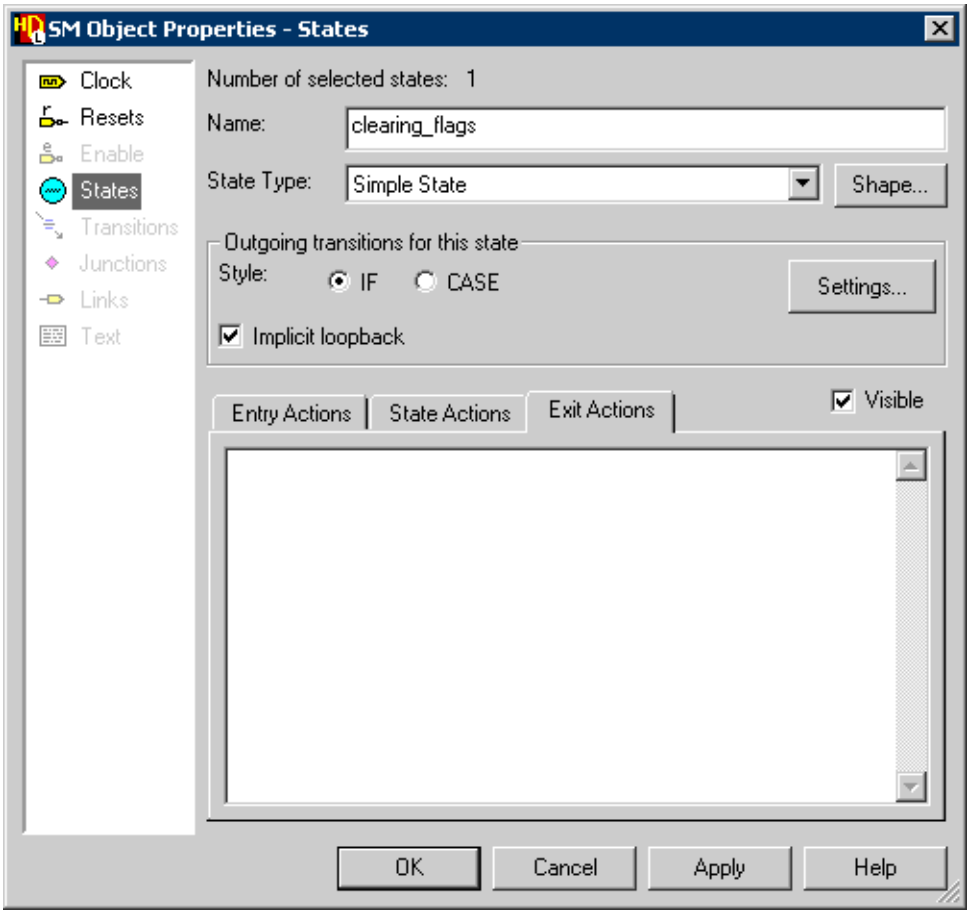
The enable signal can be active low, high or when a specified condition is evaluated.

Tip: Note that the signal level and mode are indicated on the enable point icon.

You can also specify whether the enable should be placed before any synchronous reset signals in the generated HDL. When this option is set, the enable signal is used as a global common condition which holds the state machine in its last state.

Editing State Object Properties

The **States** page of the SM Object Properties dialog box allows you to edit properties for a selected *state* (or states).



The dialog box allows you to change the state name. The state name must be unique and can only be applied to a single selected state. If you change the name of a state, any links to the state or the recovery state specified in the generation properties are automatically updated if they match the name of the state.

You can change one or more selected states to be a *simple state*, *hierarchical state* or *wait state* by choosing from a pulldown list of state types.

If you change a hierarchical state to a non-hierarchical state, the child state diagram (if it exists) and its contents are discarded. However, you can undo this change to recover the hierarchical state and its child state diagram.

If you change a simple state to a hierarchical state, any existing actions are transferred to a default state in the child state diagram.

If a simple state is selected, the **Shape** button is available and you can change the state shape to a circle, double-circle or ellipse. This can be useful if you want to visually identify a state as for example, the reset state in the state machine or to enclose a long state name within the state.

Note

If you change the shape to an ellipse it is initially drawn as a circular ellipse but can be re-shaped by dragging the selection handles when the state is selected on the diagram.

If a wait state is selected, you can enter the number of clock cycles to wait when the wait state is exited using the TIMEOUT condition. The number of clock cycles can be an integer with a value of 2 or greater.

Alternatively, you can enter an expression defined using a local variable, VHDL generic or Verilog parameter. When an expression is entered, a **Settings** button is displayed which can be used to display the Wait State Settings dialog box.

Refer to "Using Wait States" on page 58 for more information about wait states.

If manual state encoding mode is enabled in the **Encoding** page of the State Machine Properties dialog box, an Encoding entry field is disclosed which allows you to enter a binary or decimal constant encoding or enumerated attribute on the state.

Refer to "Setting State Encoding Properties" on page 73 for information about setting state encoding options.

You can choose IF or CASE style for outgoing transitions leaving the selected simple states.

When CASE style is selected, an additional field is available for you to specify the CASE selector expression used by transitions leaving the state and a **Settings** button is available to set additional decode options.

Refer to "Decode Options for CASE Transitions" on page 63 for information about decoding CASE transitions.

You can also specify whether implicit loopback is enabled. When implicit loopback is enabled, a final ELSE or default (when others) branch is automatically included in the generated HDL.

You can add or edit actions defined on the state.

You can also choose whether the actions are visible or hidden on the diagram.

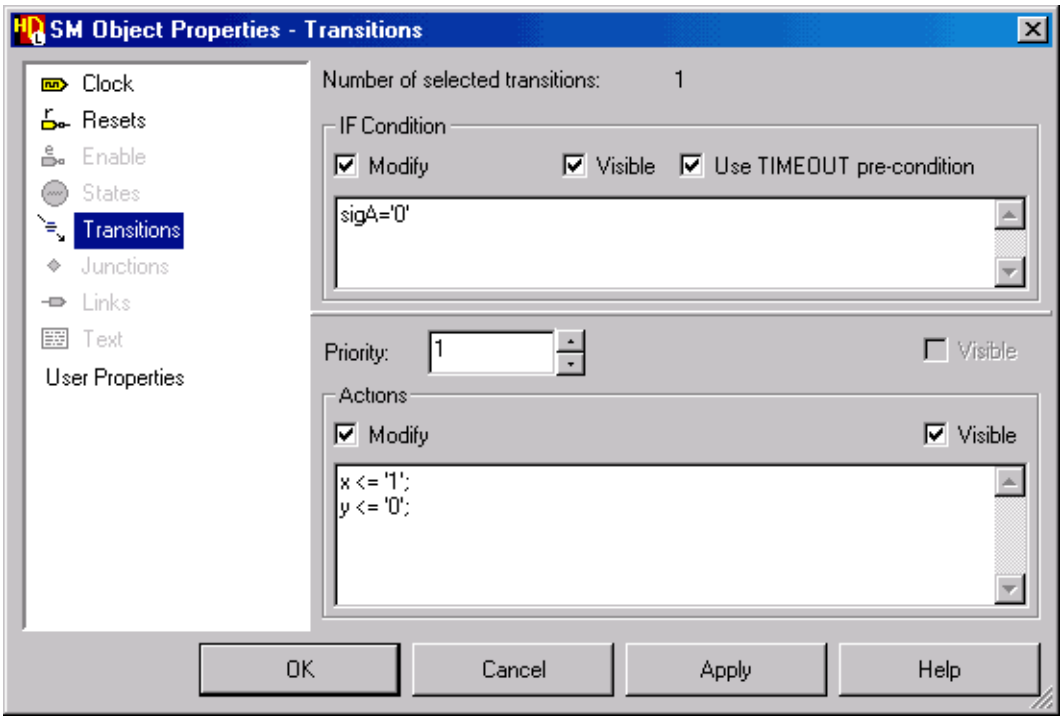
When you enter actions, the HDL syntax is automatically checked for the language of the diagram you are using (VHDL or Verilog).

Note

Note that you must include a terminating semi-colon for each statement although line breaks and indents can be used to improve legibility. State machine syntax checking can be disabled by unsetting a preference.

Editing Transition Object Properties

The **Transitions** page of the SM Object Properties dialog box allows you to edit the properties of a selected *transition* (or transitions).



The dialog box allows you to change the IF condition or CASE branch expression and the actions text for a transition.

Note

The transition style (IF or CASE) is set as an object property of the outgoing state and this determines whether you can enter an IF condition or CASE branch expression in the dialog box.

When any IF style transition is selected, you can enter a condition expression. For example: *sigA= '0'* in VHDL or *~sigA* in Verilog.

If the origin of an IF transition is on a wait state, you can choose to enable a simple *TIMEOUT* condition. If this option is set, the timeout pre-condition is appended to the specified regular

condition (if specified). For example: *TIMEOUT AND (sigA = '0')* in VHDL or *TIMEOUT && (~sigA)* in Verilog.

Refer to "Using Wait States" on page 58 for more information about wait states.

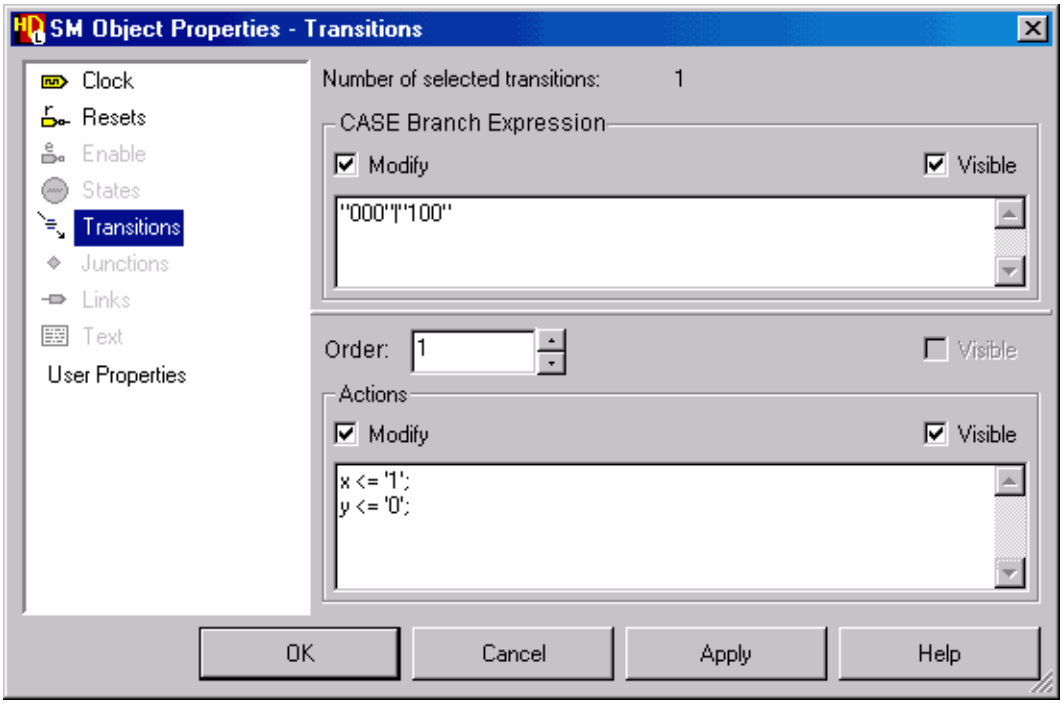
You can also set priority for IF style transitions by specifying the transition priority for a single selected transition. Any other transitions connected to the same origin state are automatically adjusted when you change a transition priority. You can choose whether the priority is visible on the diagram using the adjacent check box.

Note

If no priority is used for a VHDL IF style transition, simple *IF..THEN..END* decoding is used instead of the default *IF..THEN..ELSE* decoding.

If more than one transition is selected, you can use the Modify check box to choose whether the condition is applied to all the selected transitions. A check box determines whether the conditions are visible or hidden on the diagram.

When a CASE style transition is selected, you can enter a CASE branch expression. For example: *"000"|"100"*.



Note that an empty CASE expression is generated as *WHEN OTHERS* in VHDL or as the default choice in Verilog.

You can set the generation order for CASE style transitions by specifying the order for a single selected transition. Any other transitions connected to the same origin state are automatically

adjusted when you change the generation order. You can choose whether the order is visible on the diagram using the adjacent check box.

For VHDL, all CASE style transitions must be mutually exclusive and the transition order has no effect on behavior. However for Verilog, CASE constructs may not be mutually exclusive and the order is significant since the first match in the generated Verilog will be used.

You can add or edit actions defined on an IF or CASE Style transition. Transition actions must be entered using the correct HDL syntax for the language you are using (VHDL or Verilog) including a terminating semi-colon for each statement. However, line breaks and indents can be used to improve legibility on the diagram.

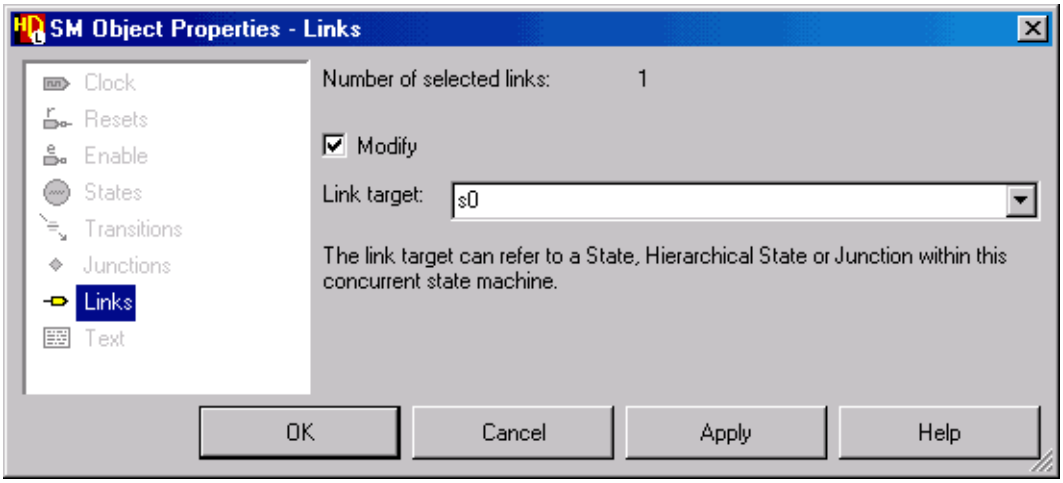
When more than one transition is selected, you can use the Modify check boxes to choose whether the conditions and actions are applied to all the selected transitions. A Visible check box selects whether the actions are displayed or hidden on the diagram.

Note

When you enter an IF condition, CASE branch expression or actions text, the syntax is automatically checked for the hardware description language of the active diagram. However, state machine syntax checking can be disabled by unsetting a preference.

Editing Link Object Properties

The **Links** page in the SM Object Properties dialog box allows you to edit properties for a selected *link* (or links).



The link target should be changed to the name of a state, junction or hierarchical state that you want to be the destination of the link. Note that the default target name for a new link can be set by a preference.

Setting the Recovery State

If the *link* is connected to a *recovery state point*, the drop down list for the link target in the **Links** page of the SM Object Properties dialog box includes all the state names in the active state machine plus the special string <*current_state*>. Alternatively, you can enter an explicit state value such as all X or all Z or any other expression.

Caution

This feature is supported to support state machines which allow state variable values that are not explicitly decoded. However, the state variable must never be allowed to take unknown values for simulation since the only assignments are made in finite state machine code which has a defined set of values. The recovery behavior will either be ignored for synthesis or the synthesis will produce excessive decoding circuitry to work out if the state machine is in any of the many undecoded state values.

A recovery state can be set for If or Case style state machines but is not required for One-Hot style.

The recovery state assignment is generated in the recovery branch (When Others in VHDL or the default branch in Verilog for CASE style and in the final Else for IF style) of the state decode for the *next_state* process or always code.

If there are no assignments in the action text (for example just comments or non-assignment action fragments such as assert or display statements) then these are inserted in the recovery branch for the *next_state* process or always code.

Recovery actions are effectively state actions. If there are any recovery state actions and the actions include assignment statements then the actions are treated in the same way as any other state actions with the appropriate code being placed in the clocked or output process as necessary.

Note that the IF style applies to the *next_state* process or always code only, the state action decoding always uses CASE.

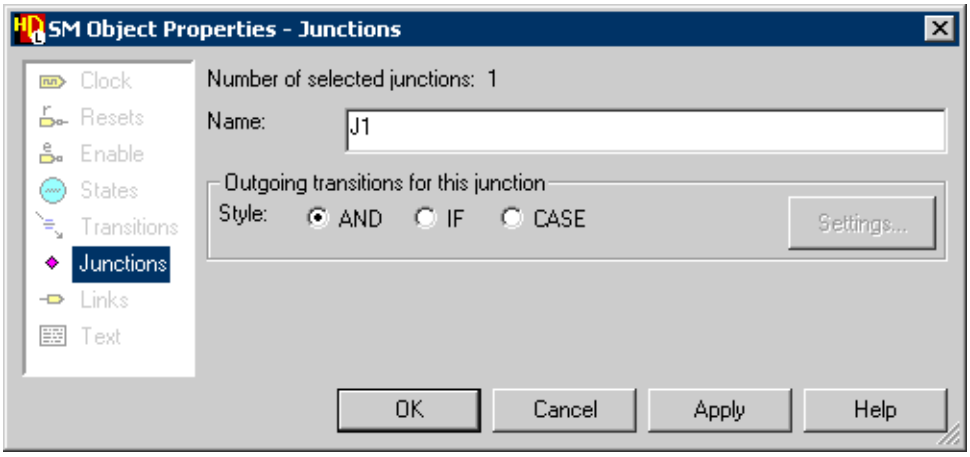
Note

If no recovery state is set when you have set enumerated encoding in the **Encoding** page of the State Machine Properties dialog box and also unset the Default State Assignment generation option, a warning is issued and a default recovery state is automatically set.

The recovery state applies only to state decoding. A "when others" condition can be specified by using a transition with no condition to specify the default branch.

Editing Junction Object Properties

The **Junctions** page in the SM Object Properties dialog box allows you to edit the properties of a selected junction (or junctions).



The dialog box allows you to change the name of the selected junction which cannot be the same as an existing junction or state on the diagram.

You can choose IF, CASE or AND style for outgoing transitions leaving the selected junction.

When CASE style is selected, an additional field is available for you to specify the CASE selector expression used by transitions leaving the state and a **Settings** button is available to set additional decode options.

Refer to "Decode Options for CASE Transitions" on page 63 for information about decoding CASE transitions.

When IF style is selected a **Settings** button is available to set the if statement style used

Using Wait States

A *wait state* can be used to implement a multi-cycle wait in a synchronous state machine.

The number of clock cycles to wait for is specified in the state object properties and is applied when the TIMEOUT pre-condition is used for a transition exiting the wait state.

Note

A wait state can have multiple exit transitions. If the TIMEOUT pre-condition is unset for any of these transitions, the state may be exited via this transition before the timeout has expired.

The TIMEOUT pre-condition can be used on its own to implement a simple delay or if enabled when a regular condition is entered, it is combined with the condition using a logical AND.

A separate timeout signal is generated for each concurrent state machine using the form:

```
<concurrent_machine_name>_timeout
```

Local counter signals which are used by all wait states in the concurrent state machines are generated using the forms:

```
<concurrent_machine_name>_timer
<concurrent_machine_name>_next_timer
```

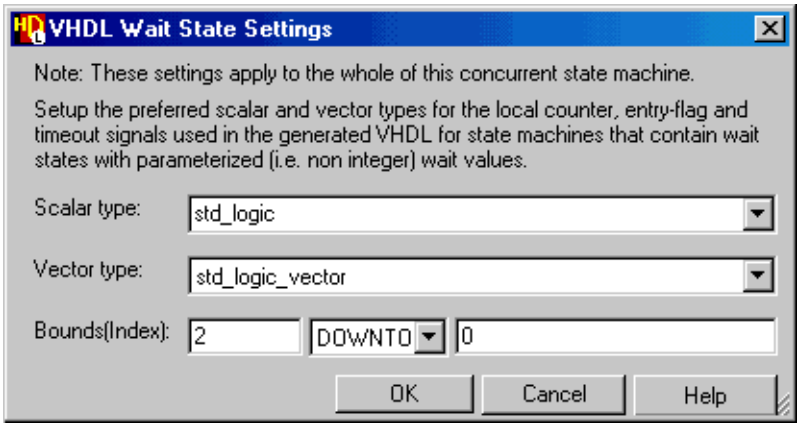
An entry flag is generated for each wait state using the form:

```
<concurrent_machine_name>_to_<state_name>
```

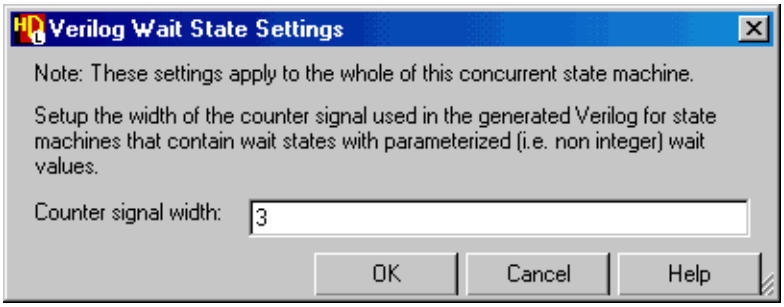
For example, if the concurrent state machine is named *csm2*, the timeout, timer and next timer signals would be *csm2_timeout*, *csm2_timer* and *csm2_next_timer*, and the entry flag for wait state *ws1* would be *csm2_to_ws1*.

If all the wait states in a concurrent state machine use integer wait values, the VHDL signal types default to *std_logic* (or *std_logic_vector*) and have the width required for the largest wait value. If any wait state has a parameterized (non-integer) value, the scalar type, vector type and bounds of the timeout, counter and entry flag signals must be specified.

These settings can be specified in the VHDL Wait State Settings dialog box which is displayed when you use the Wait State **Wait State Settings** button in the **States** page of the SM Object Properties dialog box.



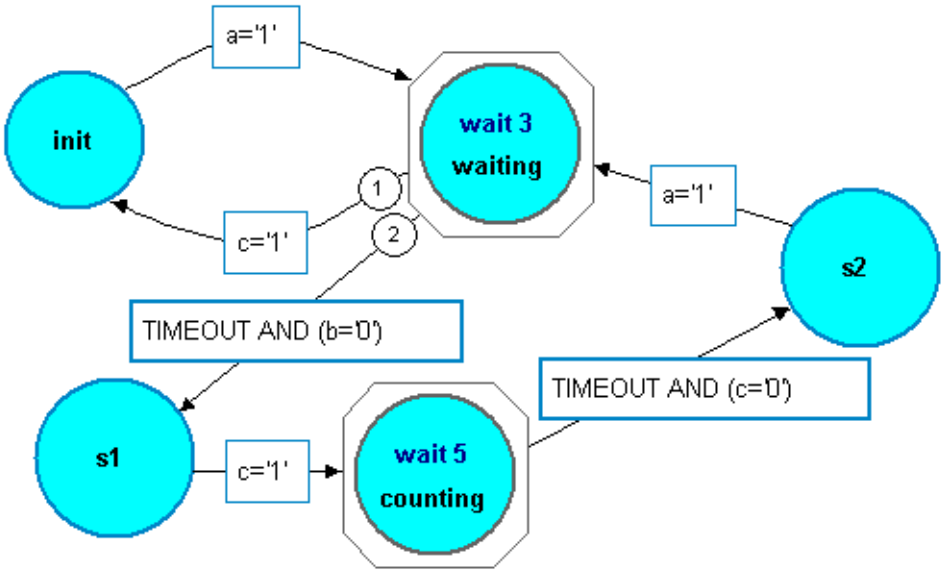
If you are using Verilog, the Verilog Wait State Settings dialog box allows you to specify the width of the counter signal.



The Wait State Settings dialog boxes can also be accessed from the **Generation** page of the State Machine Properties dialog box or you can specify defaults by accessing them from the **Default Settings** page of the State Machine Master Preferences dialog box.

VHDL Wait State Example

The following example shows a simple state diagram using VHDL Wait states:



The wait state logic generated for this state machine is shown below:

```
wait_combo : PROCESS(FSM_timer, FSM_to_waiting, FSM_to_counting)
VARIABLE FSM_temp_timeout : std_logic;

BEGIN

   IF (unsigned(FSM_timer) = 0) THEN
      FSM_temp_timeout := '1';
   ELSE
      FSM_temp_timeout := '0';
```

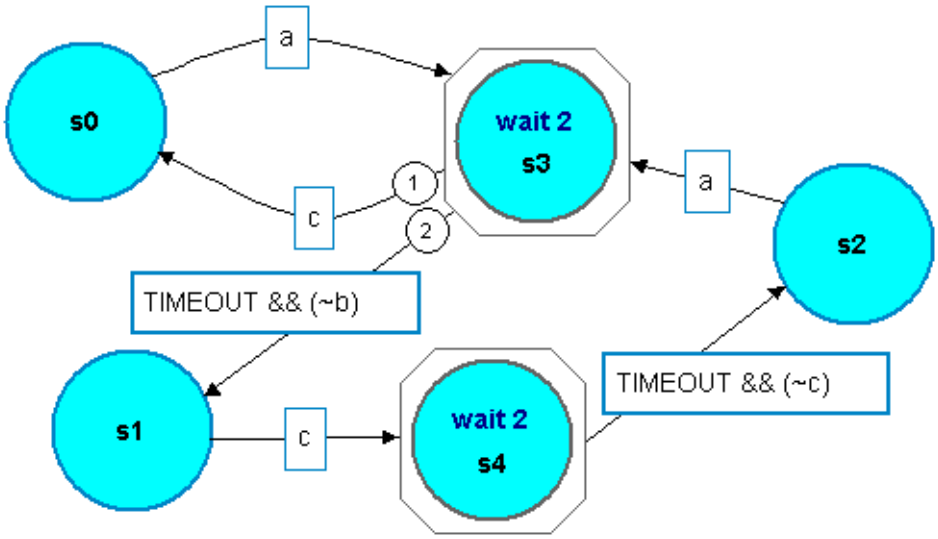
```
   END IF;

   IF (FSM_to_waiting = '1') THEN
      FSM_next_timer <= "010"; --no cycles(3)-1=2
   ELSIF (FSM_to_counting = '1') THEN
      FSM_next_timer <= "100"; --no cycles(5)-1=4
   ELSE
      IF (FSM_temp_timeout = '1') THEN
         FSM_next_timer <= (others => '0');
      ELSE
         FSM_next_timer <= unsigned(FSM_timer) - '1';
      END IF;
   END IF;

   FSM_timeout <= FSM_temp_timeout;

END PROCESS wait_combo
```

The above example applies to a two-process or three-process FSM. The following example shows the wait state logic generated for a single-process FSM:

```
wait_combo : PROCESS(FSM_timer, FSM_to_waiting, FSM_to_counting)
VARIABLE FSM_temp_timeout : std_logic;
 BEGIN
IF (unsigned(FSM_timer) = 0) THEN
FSM_temp_timeout := '1';
ELSE
FSM_temp_timeout := '0';
END IF;
IF (FSM_to_waiting = '1') THEN
FSM_next_timer <= "001"; --no cycles(3)-2=1
FSM_temp_timeout := '0';
ELSIF (FSM_to_counting = '1') THEN
FSM_next_timer <= "011"; --no cycles(5)-2=3
FSM_temp_timeout := '0';
ELSE
IF (FSM_temp_timeout = '1') THEN
FSM_next_timer <= (others => '0');
ELSE
FSM_next_timer <= unsigned(FSM_timer) - '1';
END IF;
END IF;
FSM_timeout <= FSM_temp_timeout;
END PROCESS wait_combo
```

Verilog Wait State Example

The following example shows a simple state diagram using Verilog Wait states:



The wait state logic generated for this state machine is shown below:

```
always @(FSM_timer or FSM_to_waiting or FSM_to_counting)
begin
   FSM_timeout = (FSM_timer == 3'd0);
   if (FSM_to_waiting == 1'b1) begin
      FSM_next_timer = 3'd2;
   end
   else if (FSM_to_counting == 1'b1) begin
      FSM_next_timer = 3'd4;
   end
   else begin
   FSM_next_timer = (FSM_timeout)? 3'd0: (FSM_timer - 3'd1);
   end
end
```

The above example applies to a two-process or three-process FSM. The following example shows the wait state logic generated for a single-process FSM:

```
always @(FSM_timer or FSM_to_waiting or FSM_to_counting)
begin
FSM_timeout = (FSM_timer == 3’d0);
if (FSM_to_waiting == 1’b1) begin
FSM_next_timer = 3’d2;
end
else if (FSM_to_counting == 1’b1) begin
FSM_next_timer = 3’d3;
FSM_timeout = 1'd0;
end
else begin
FSM_next_timer = (FSM_timeout)? 3’d0: (FSM_timer - 3’d1);
end
end
```

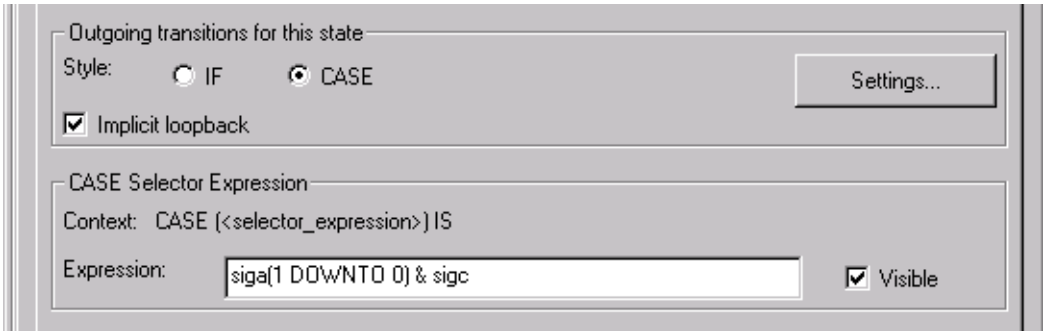
Decode Options for CASE Transitions

A CASE style transition typically produces a faster, parallel, multiplex-based circuit whereas an IF style transition produces a serial, priority decoder-based circuit. Also, more efficient decoding can often be achieved by concatenating together signals of similar type and decoding the resulting concatenated variable rather than each signal individually.

You can use CASE style transitions when the Case option is selected for the HDL style in the **Generation** page of the State Machine Properties dialog box which is described in "Setting State Diagram Generation Properties" on page 68.

When CASE style is selected in the **States** page of the SM Object Properties dialog box, you can specify the CASE selector expression used by transitions leaving the state. You can then enter expressions for each branch as object properties for the transitions leaving the state.

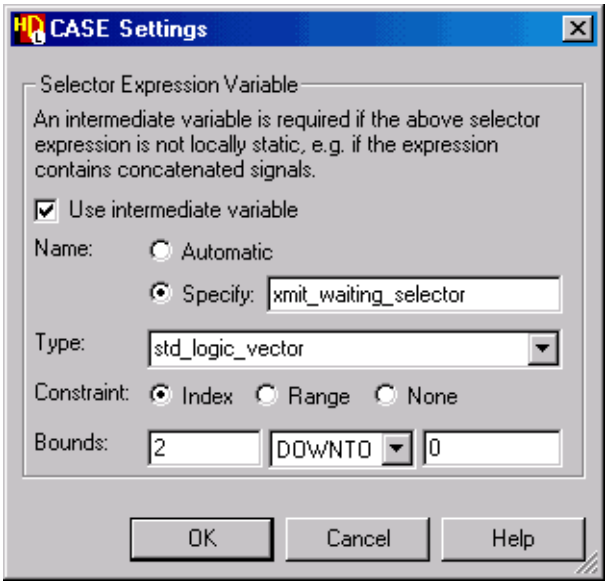
For example, the selector expression in a VHDL design might be *siga(1 DOWNTO 0) & sigc* with branch expressions *"101"* and *"011"* | *"111"* on transitions leaving the state.



You can choose whether the CASE selector expression is displayed or hidden on the diagram. When visible, the selector expression is shown with the prefix `CASE:` under the state name on the state diagram. This prefix must be included if you use direct text editing to edit the selector expression.

You can set additional decode options by using the **Settings** button on the **States** page of the SM Object Properties dialog box.

If you are using VHDL, the following CASE Settings dialog box allows you to specify a selector expression variable:



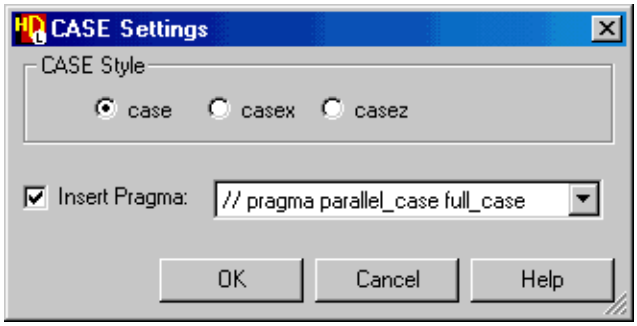
An intermediate variable may be required if the expression is not locally static (for example, if the expression contains concatenated signals).

You can choose to name the variable automatically or specify a variable name which must be a VHDL identifier. If Automatic naming is set, the name is generated using the template: *hds_selN* where *N* is an integer.

A list of standard VHDL types is available in a pulldown list. The variable type should be *std_ulogic_vector* if all the inputs are *std_ulogic* or *std_ulogic_vector*. It should be *std_logic_vector* if all the inputs are scalar and of type *std_logic*.

Otherwise, it should be the same type as the input arrays. The bounds must be sufficient for the size of the concatenated input expressions. Note that when scalar values are concatenated with other scalar values or with array values, the result is always an array value.

If you are using Verilog, you can choose *casex* or *casez* instead of the default bit comparison case style:



You can also choose to insert the following pragmas to specify full case or parallel case statements:

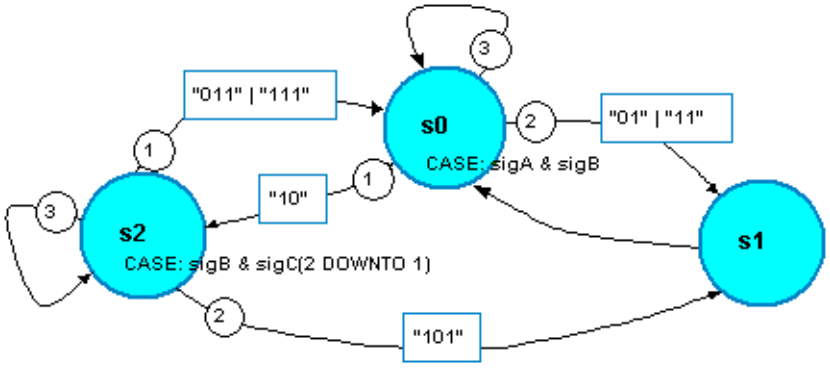
full_case	All possible branches have been specified, any missing branches cannot occur and a default branch need not be generated.
parallel_case	Branches are mutually exclusive.
parallel_case full_case	All possible branches have been specified and are mutually exclusive.

Note

Changes made in the CASE Settings dialog box are not applied to the diagram until you confirm the Object Properties dialog box.

Example of VHDL CASE Decode

The following example shows a simple state machine in which the transitions from states *s0* and *s2* have been defined using CASE selector expressions.



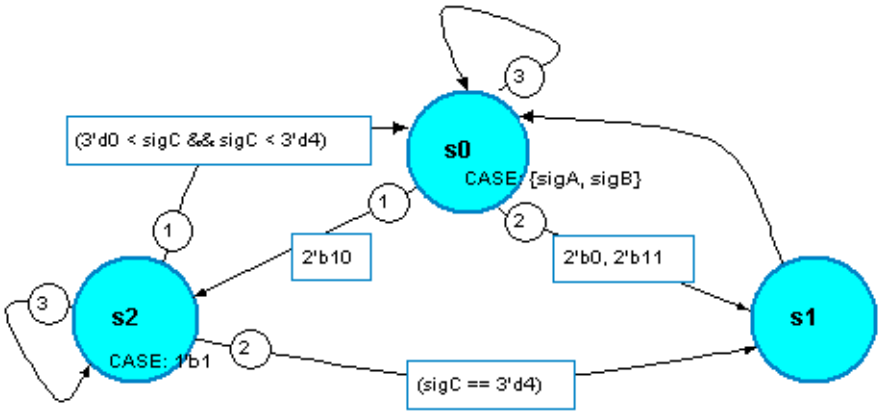
The next state process generated for this state machine is shown below. Notice the use of automatic selector expression variables (*hds_sel0* and *hds_sel1*).

```
nextstate : PROCESS (current_state, sigA, sigB, sigC)
VARIABLE hds_sel0 : std_logic_vector(1 DOWNTO 0);
VARIABLE hds_sel1 : std_logic_vector(2 DOWNTO 0);
BEGIN
   CASE current_state IS
   WHEN s0 =>
      hds_sel0 := sigA & sigB;.
      CASE hds_sel0 IS
         WHEN "10" =>
            next_state <= s2;
         WHEN "01" | "11" =>
            next_state <= s1;
         WHEN OTHERS =>
            next_state <= s0;
```

```
      END CASE;
   WHEN s1 =>
      next_state <= s0;
   WHEN s2 =>
      hds_sel1 := sigB & sigC(2 DOWNTO 1);
      CASE hds_sel1 IS
         WHEN "011" | "111" =>
            next_state <= s0;
         WHEN "101" =>
            next_state <= s1;
         WHEN OTHERS =>
            next_state <= s2;
      END CASE;
   WHEN OTHERS =>
      next_state <= s0;
   END CASE;
END PROCESS nextstate;
```

Example of Verilog CASE Decode

The following example shows a simple state machine in which the transitions from states *s0* and *s2* have been defined using CASE selector expressions.



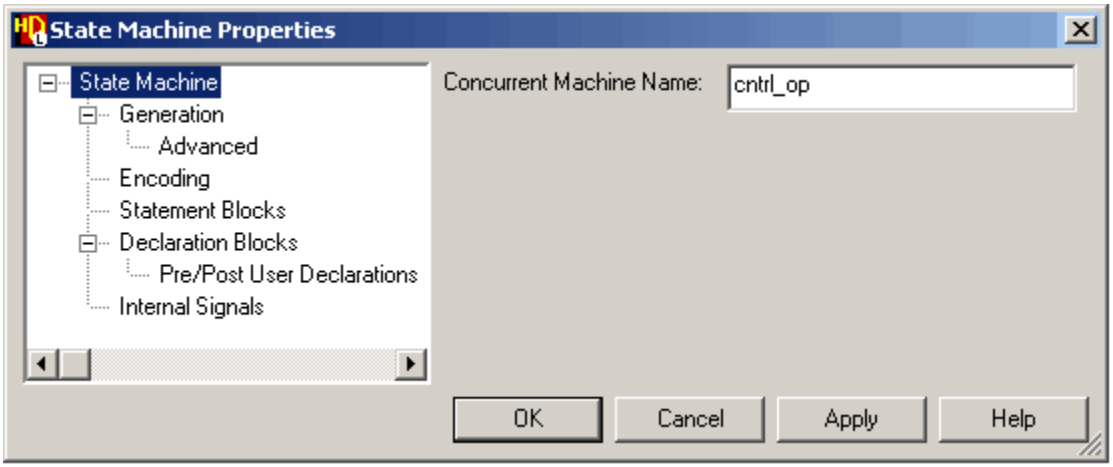
The next state code generated for this state machine is shown below. Notice how the CASE statement is used to match true conditions by setting the selector expression for state *s2* to *1'b1*.

```
always  @(current_state or sigA or sigB or sigC)
begin
   case (current_state)
      s0:
         case({sigA, sigB})
         2'b10:
            next_state = s2;
         2'b0,2'b11:
            next_state = s1;
         default:
            next_state = s0;
         endcase
      s1:
```

```
            next_state = s0;
         s2:
            case(1'b1)
            (3'd0 < sigC && sigC < 3'd4):
               next_state = s0;
            (sigC == 3'd4):
               next_state = s1;
            default:
               next_state = s2;
            endcase
         default: begin
            next_state = s0;
         end
      endcase
   end
```

Setting State Machine Properties

You can set state machine properties for a state diagram by choosing **State Machine Properties** from the **Diagram** or popup menu to display the State Machine Properties dialog box.



The main page of the State Machine Properties dialog box allows you to edit the diagram name of the active concurrent state machine.

You can select additional pages and sub-pages from the left pane in the dialog box:

- The Generation page allows you to set basic properties for HDL generation. A separate sub-page can be used to set Advanced generation properties.
- The Encoding page can be used to specify state machine encoding.
- The Statement Blocks page allows you to specify concurrent statements, state register statements and global actions.
- The Declaration Blocks page allows you to specify architecture, module or process declarations.

- The Internal Signals page allows you to set the prefix or suffix used for internal registered or clocked signals.

The statements, declaration, global actions and signal status are displayed as text objects on the diagram and the dialog can be opened directly by double-clicking over one of these objects.

The generation properties and state machine encoding information are not displayed on the state diagram.

Setting State Diagram Generation Properties

You can set the HDL generation properties for a state diagram in the **Generation**, **Advanced** and **Control** pages of the State Machine Properties dialog box.

Note

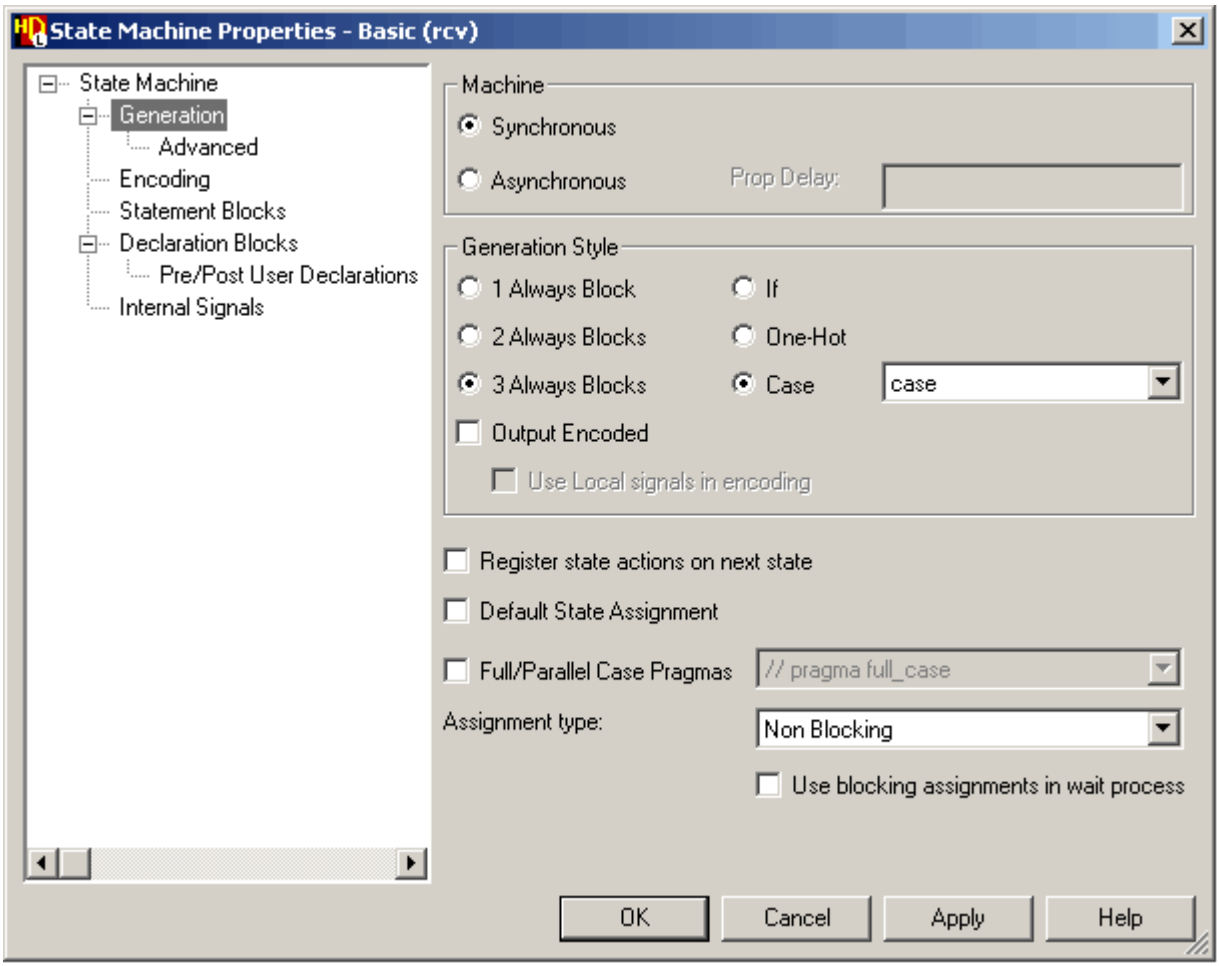
Note that separate generation properties can be specified for each diagram in a set of concurrent state machines.

The **Generation** page allows you to choose a synchronous or asynchronous state machine, set the HDL style, specify the number of Verilog *always* blocks or VHDL processes, set the state encoding type to output encoded and choose whether to register state actions on the next state.

If you set the encoding type to output encoded you can no longer apply any other encoding options on the encoding page of the state machine properties dialog. Also, on setting the output encoded option, you can choose whether you wish to use local signals in encoding or not.

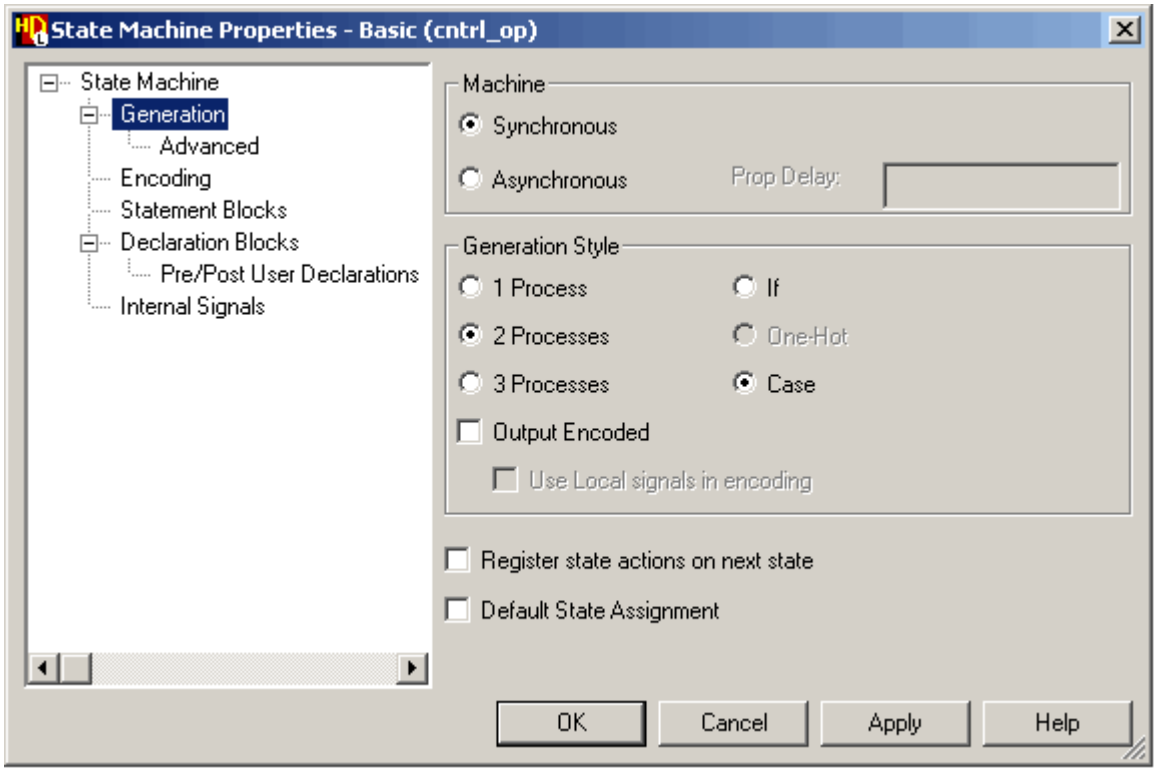
Refer to "Synchronous and Asynchronous State Machines" on page 138 and "HDL Style" on page 138 for more information about state machine types and coding styles.

When you are using Verilog, you can set the assignment type (mixed, blocking or non-blocking) and choose full or parallel case pragmas. You can also set the default state assignment option:



Refer to "Verilog Assignment Type" on page 145 and "Verilog Full/Parallel Case Pragmas" on page 145 for more information.

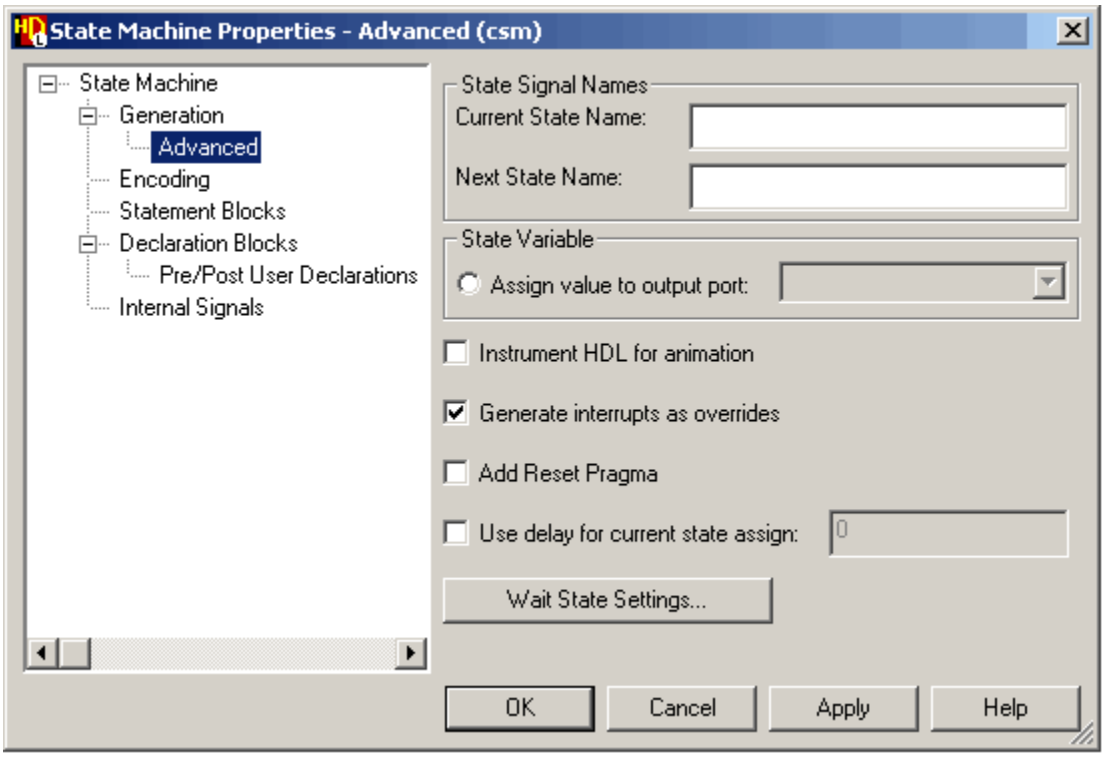
When you are using VHDL and you have selected the Case HDL style, you can also set the default state assignment as described in "VHDL Default State Assignment" on page 144.



Advanced State Diagram Generation Properties

The **Advanced** page of the State Machine Properties dialog box allows you to set additional generation options.

The following picture shows the **Advanced** page when you are using Verilog



These include options to specify alternative names for the reserved state signal names *current_state* and *next_state* used for the state variable in the generated HDL.

You can assign the state variable to the value of an output port selected from a dropdown list of the available ports. When you are using VHDL, you can choose to automatically generate a *type* or you can specify a discrete type for the state variable.

You can choose to instrument the HDL for animation. When this option is enabled, extra code is included in the generated HDL which provides activity information during state diagram animation. The additional code is enclosed between translation control pragmas which ensure that it is ignored by downstream synthesis tools.

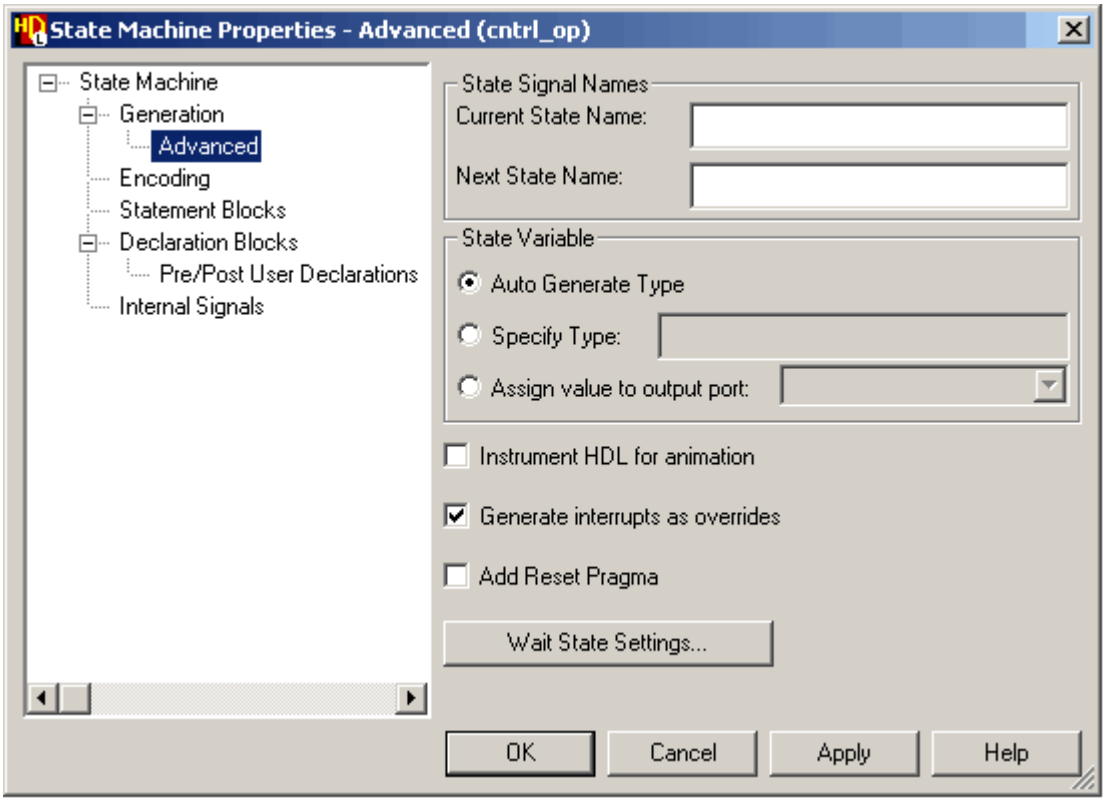
You can choose to generate the HDL for state diagram interrupts as overrides instead of as exclusive if-then-else statements. Refer to “Generate Interrupts as Overrides” on page 143 for more information.

You can optionally set pragmas (*sync_set_reset_local* or *async_set_reset_local*) which identify the name of the currently specified synchronous or asynchronous reset signal. Make sure that the “ATTRIBUTES” package unit in the protected library “synopsys” is included in the package list for the generated code to compile successfully since these are Synopsys attributes which are declared in the package synopsys.ATTRIBUTES (shipped with HDS).

You can choose to use a specified delay for the current state assignment.

You can also specify default wait state settings by using the Wait State Settings... button to display the Wait State Settings dialog box as described in "Using Wait States" on page 58.

The following picture shows the **Advanced** page when you are using VHDL:



Note

If you choose to assign the state variable to a specified output port value (in the **Advanced** page of the SM Object Properties dialog box), and at the same time set the state encoding as Auto (in the **Encoding** page), the following occurs on VHDL generation:

- The state encoding values will be represented in the generated code as constant declarations, and the type of state variable(s) (the current and next state signals) will be the same as the type of the output port (for example "std_logic_vector").

  This is because eventually the state signal will be assigned to the value of the output port, and hence they have to be compatible in type.

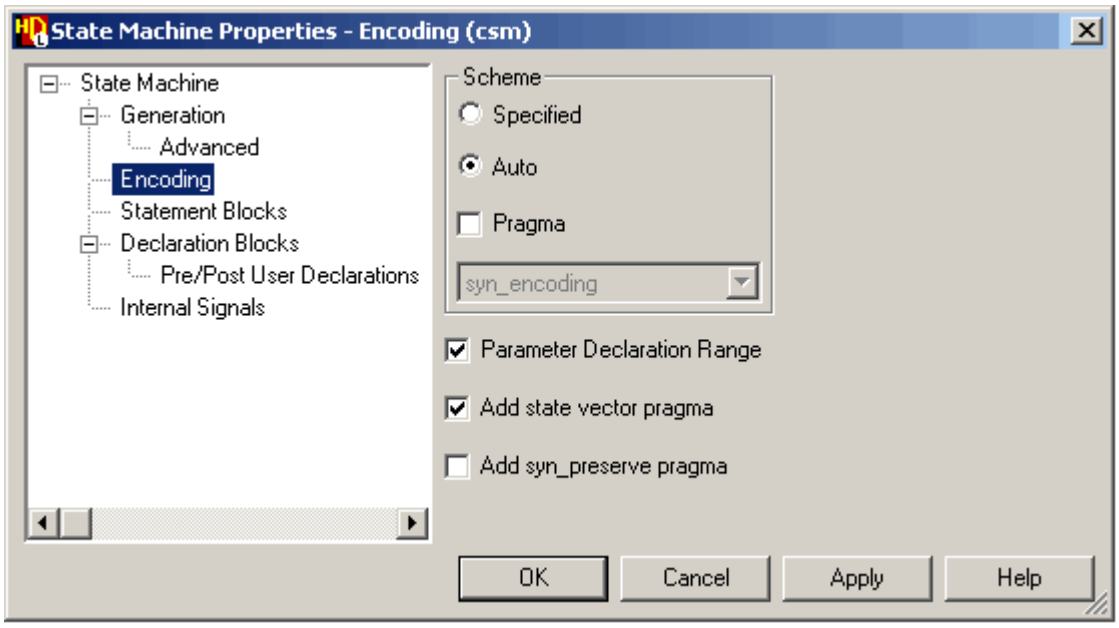
- The state encoding will follow the encoding style you have defined in the **Encoding** page. However, it should be noted that if you have the state encoding set as Auto without specifying an attribute, then the encoding style will be assumed to be "Sequential".

Setting State Encoding Properties

You can set state encoding in the HDL generated for a state diagram in the **Encoding** page of the State Machine Properties dialog box.

You can explicitly specify the state encoding or use an automatic scheme to select the required VHDL attributes or Verilog pragmas. If you are using Verilog, you can also choose whether to use the range in the declaration of the state encoding parameter. You can also add verilog state vector and *syn_preserve* pragmas (refer to "Verilog State Vector Pragmas" on page 145) or VHDL state vector and *syn_preserve* attributes.

For example, the following dialog box is displayed if you are using Verilog:



When a specified encoding scheme is selected for either language, the encoding values can be entered by direct text editing on the state or by using the Encoding field in the **States** page of the State Machine Object Properties dialog box as described in "Editing State Object Properties" on page 52.

Refer to "State Encoding" on page 146 for information about the HDL generated and the use of VHDL attributes or Verilog pragmas and encoding algorithms in automatic or specified encoding schemes.

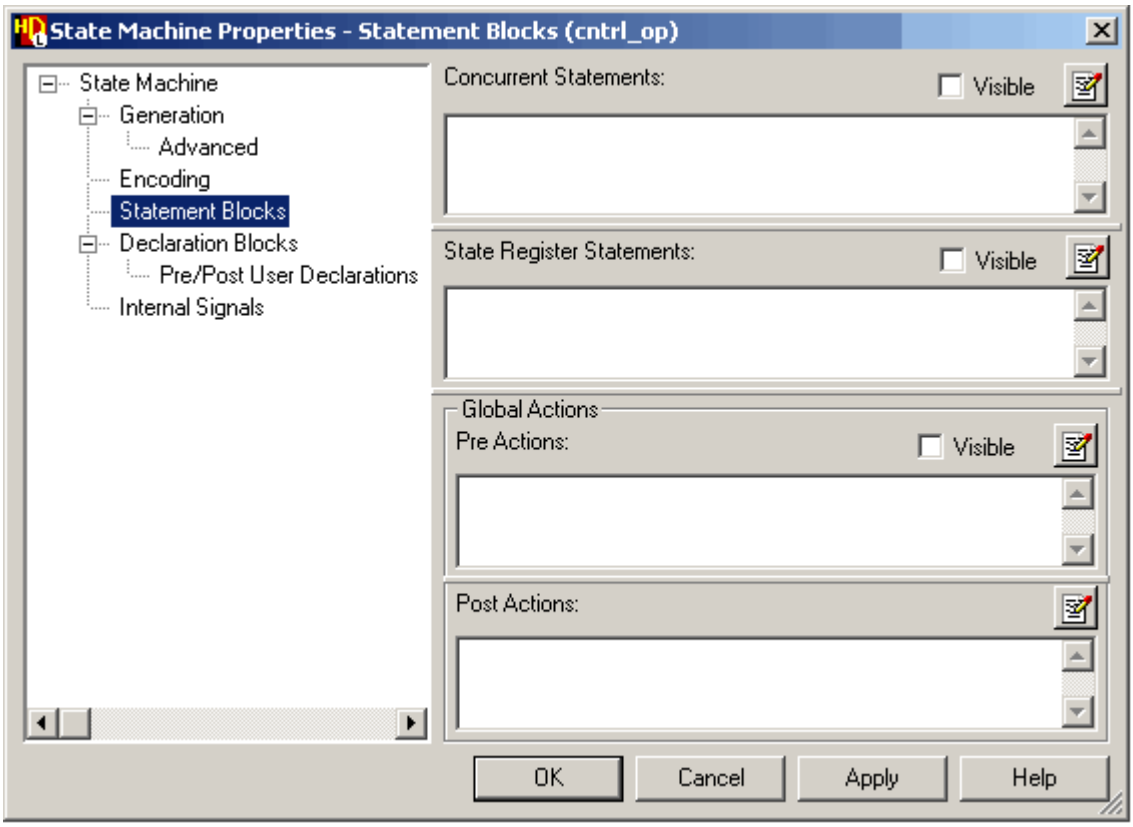
Setting Statement Blocks

You can edit *concurrent statements*, *state register statements* and *global actions* in the **Statement Blocks** page of the State Machine Properties dialog box.

Separate free-format entry boxes are provided for you to add or edit concurrent statements, state register statements and pre- or post- global actions. The edited statements are added to the diagram when you confirm the dialog box.

The syntax is automatically checked on entry for the hardware description language (Verilog or VHDL) of the active diagram. However, state machine syntax checking can be disabled by unsetting a state machine preference.

You can also choose whether the statements are visible or hidden on the diagram (or on the top level state diagram when you are editing a hierarchical state machine).



Concurrent statements are included in the generated HDL at the end of the VHDL architecture or Verilog module and are applied to all diagrams in a set of concurrent state machines. These statements are executed concurrently with all of the processes (or *always* blocks) in the state machine and are typically used for datapath operations, special clocking or assigning individual elements of the state vector to an output.

Note

If you use an output signal in a concurrent statement, you may need to remove the default value assigned to the signal in the *signals status* table.

State register statements are included in the generated HDL before the state decoding statements in the clock process (or *always* code). These statements are inserted instead of the default

assignment to the next state and are typically used to determine when a counter should be incremented or reset and whether to update the state.

Global actions can be used to assign complex actions that are executed on every clock or state change. Note however, that you should use the *signals status* table to assign default values to signals.

Pre Actions are included in the generated HDL just after any default values specified at the beginning of the output process (combinatorial signals). *Post Actions* are included after the clock condition in the clocked process (registered signals) and are executed before any conditional actions.

State register statements and global actions can be specified separately for each diagram in a set of concurrent state machines.

Note

If your default text editor is set to DesignPad, a  button is available which allows you to edit the statement block in DesignPad. You can also edit the statements directly on the diagram by clicking on the action to select the text and clicking again to edit the text.

Setting Declaration Blocks

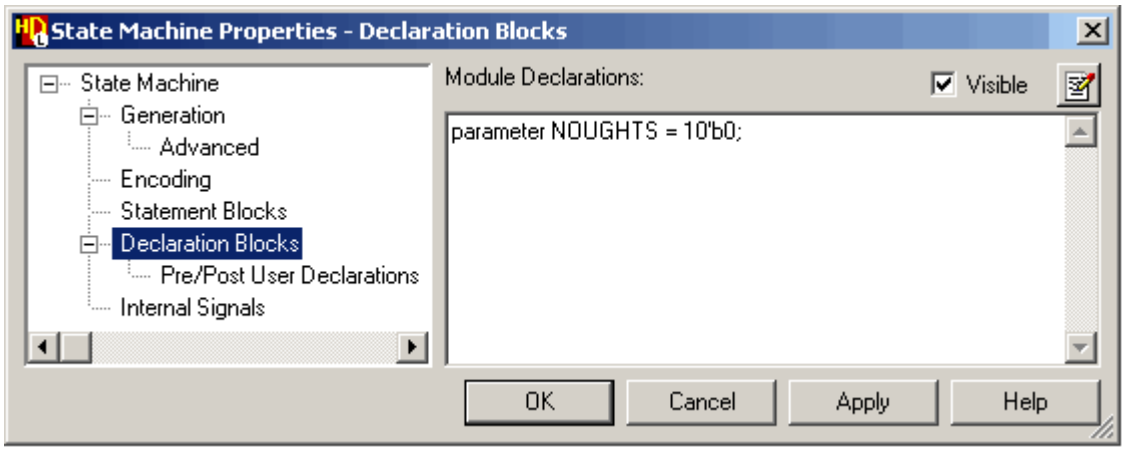
You can edit *module declarations* (when using Verilog) or *architecture declarations* and *process declarations* (when using VHDL) in the **Declaration Blocks** page of the State Machine Properties dialog box.

The dialog box allows you to enter any valid HDL statements for the current hardware description language in a free-format entry box. Signals, constants or variables can be declared and comments, procedures, functions or type definitions can also be included.

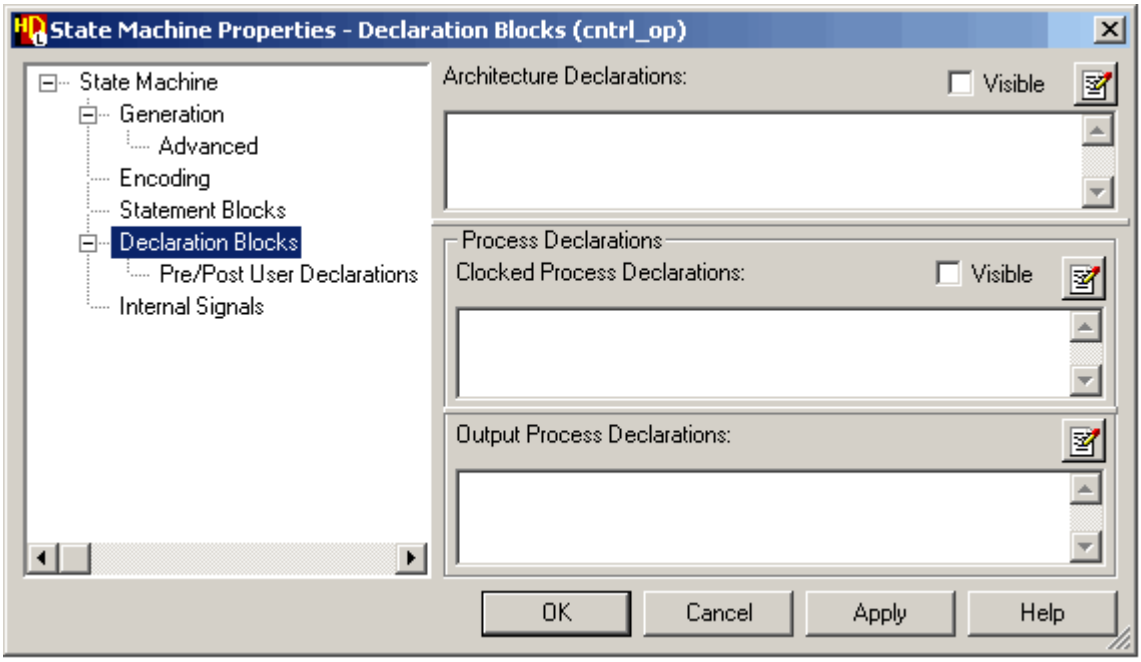
Refer to "Declaration Syntax" on page 24 for information about the syntax used for declarations.

The syntax is checked on entry and the declarations are added as a text object on the state diagram (or the top level state diagram when you are editing a hierarchical state machine) when you confirm the dialog box.

If you are using Verilog, a single entry box is provided for editing module declarations:



If you are using VHDL, the dialog box provides separate free-format entry boxes for you to add or edit architecture declarations, clocked process declarations and output process declarations:



The module or architecture declarations are inserted at the beginning of the VHDL architecture or Verilog module in the generated HDL in the order they are listed and apply to all diagrams in a set of concurrent state machines.

Architecture declarations or module declarations cannot be set when a state diagram is used to define an embedded view in a block diagram or IBD view. However, any declarations required by the embedded view can be set on the parent view.

The process declarations which are placed immediately before the BEGIN statement in the generated VHDL for both the clocked and output processes. Separate process declarations can be specified for each diagram in a set of concurrent state machines.

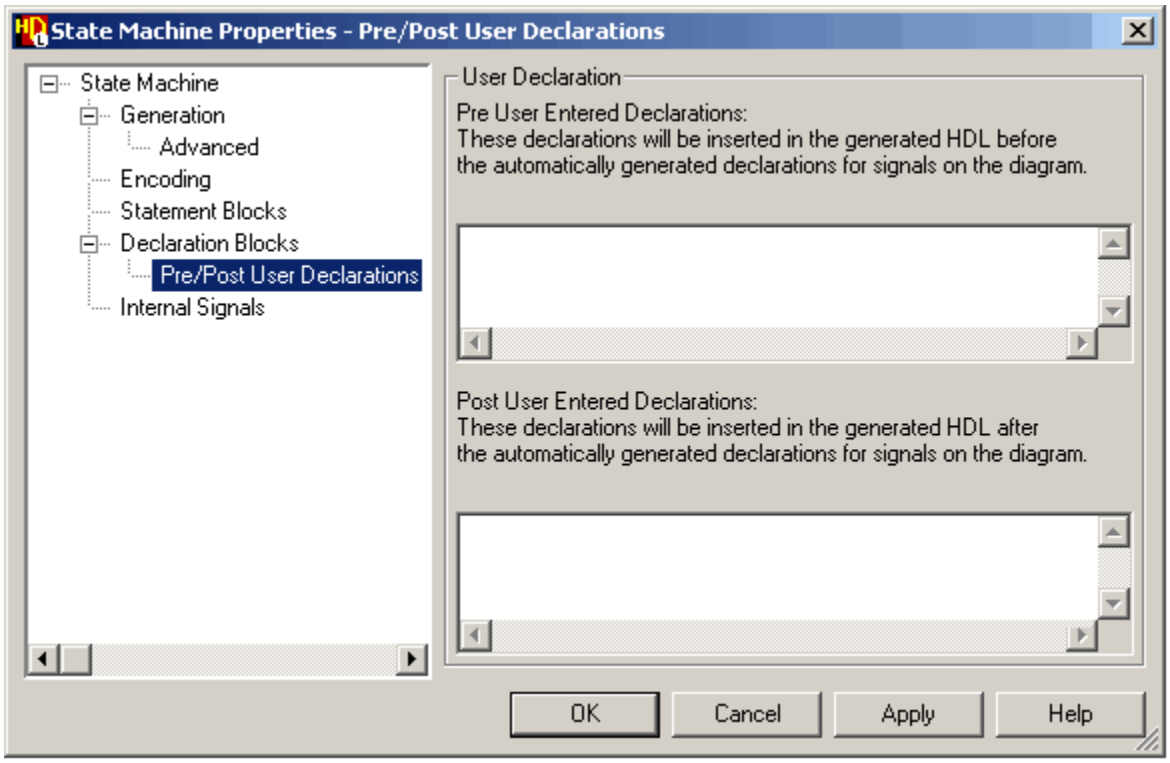
You can choose whether the declarations are visible or hidden on the diagram.

Note

If your default text editor is set to DesignPad, a button is available which allows you to edit the declaration block in DesignPad. You can also edit the declarations directly on the diagram by clicking to select the text and clicking again to edit the text.

Editing Pre/Post User Declarations

The **Pre/Post User Declaration** page of the State Machine Properties dialog box allows you to add or edit pre/post user-defined declarations for a state machine view. You can enter free-format declarations before or after the signal and state type declarations.



Pre user declarations can be referenced by signal declarations or by post user declarations. For example, you could use a pre user declaration for a bus width constant which is referenced by a graphically defined signal declaration.

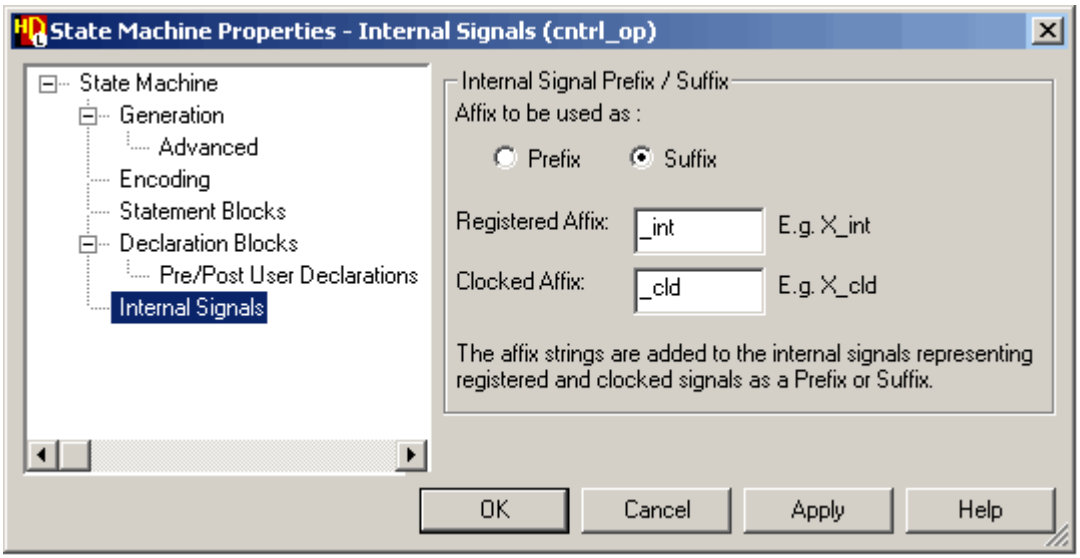
Similarly a post user declaration can reference a signal declared in the graphically defined or pre user declarations.

The syntax is automatically checked when you confirm the dialog box. However, syntax checking can be disabled by unsetting a state machine preference.

Setting State Diagram Internal Signal Names

You can set the status for output or locally declared signals in a state diagram using the "Signals Table" which is described in Chapter 4.

You can set the prefixes and suffixes used to identify registered or clocked internal signals in the **Internal Signals** page of the SM Properties dialog box.



You can set preferences for the internal registered and clocked signal names in the **Default Settings** page of the State Machine Preferences dialog box.

You can set preferences in the **Headers** tabs of the VHDL and Verilog Options dialog boxes to include the generation properties and the contents of the signals status table as comment text in the generated HDL. Refer to "Setting View Headers" in the *HDL Designer Series User Manual* for information about setting VHDL and Verilog header preferences.

Setting State Machine Preferences

You can set state machine preferences by choosing **State Machine** from the **Master Preferences** cascade of the **Options** menu in the *design manager*.

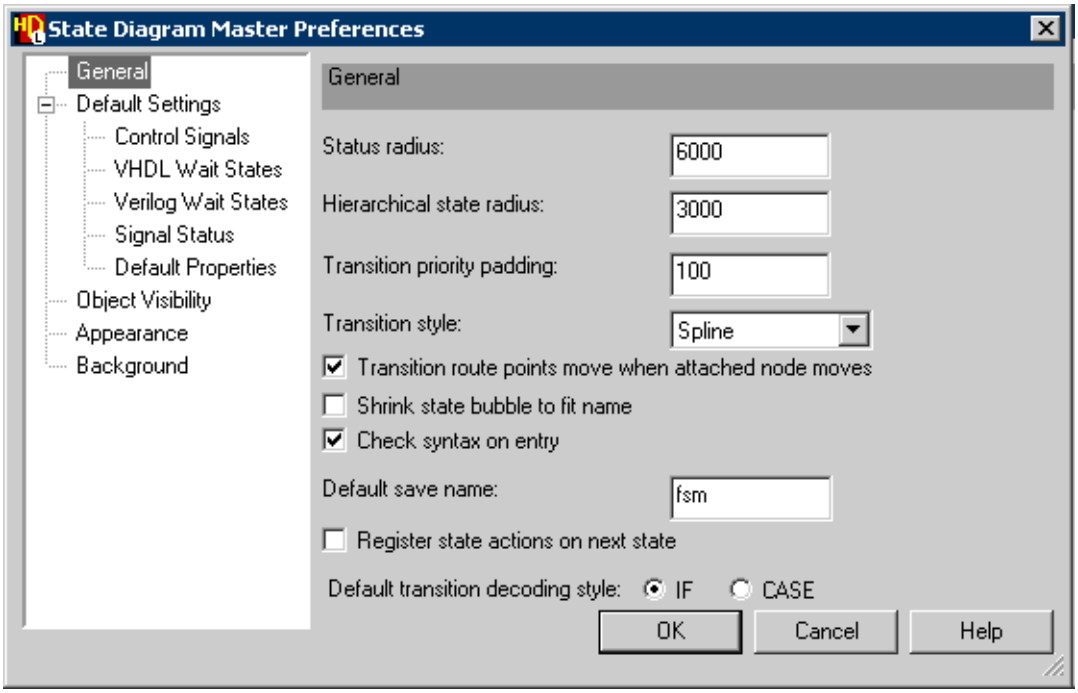
The State Machine Preferences dialog box has separate pages for **General**, **Default Settings**, **Object Visibility**, **Appearance** and **Background** preferences.

Note

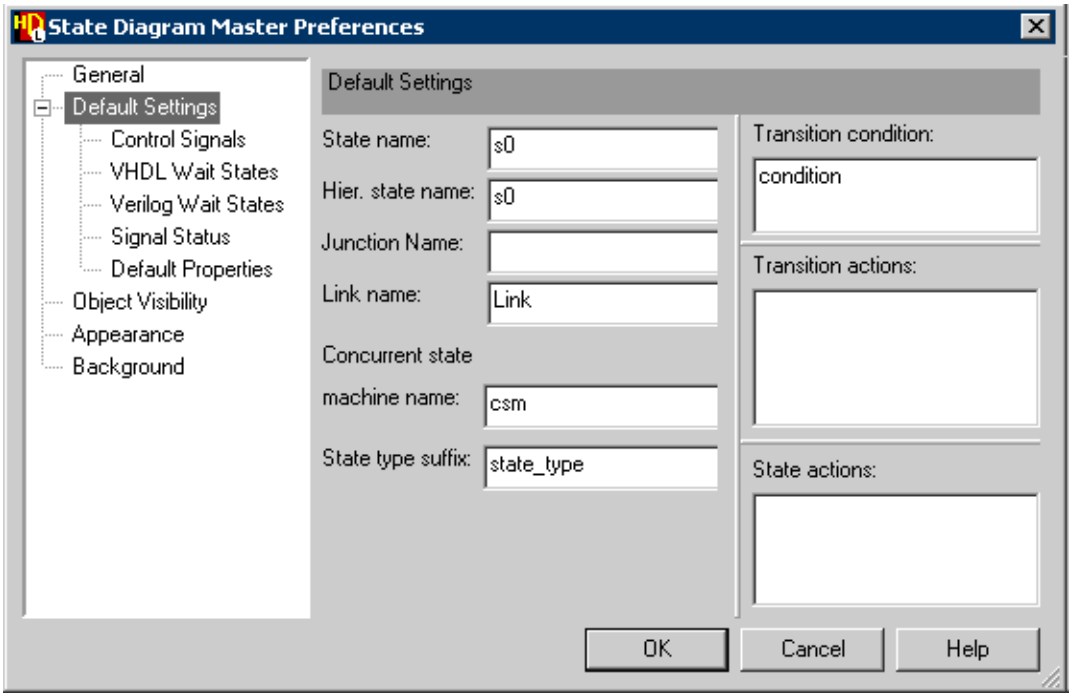
The General, Default and Object Visibility settings take effect on the next state diagram that you create and can only be edited when the dialog box is displayed from the **Master Preferences** cascade in the design manager **Options** menu. These pages are not available when you choose **Diagram Preferences** in a graphic editor window.

The **General** page allows you to set other state machine options:

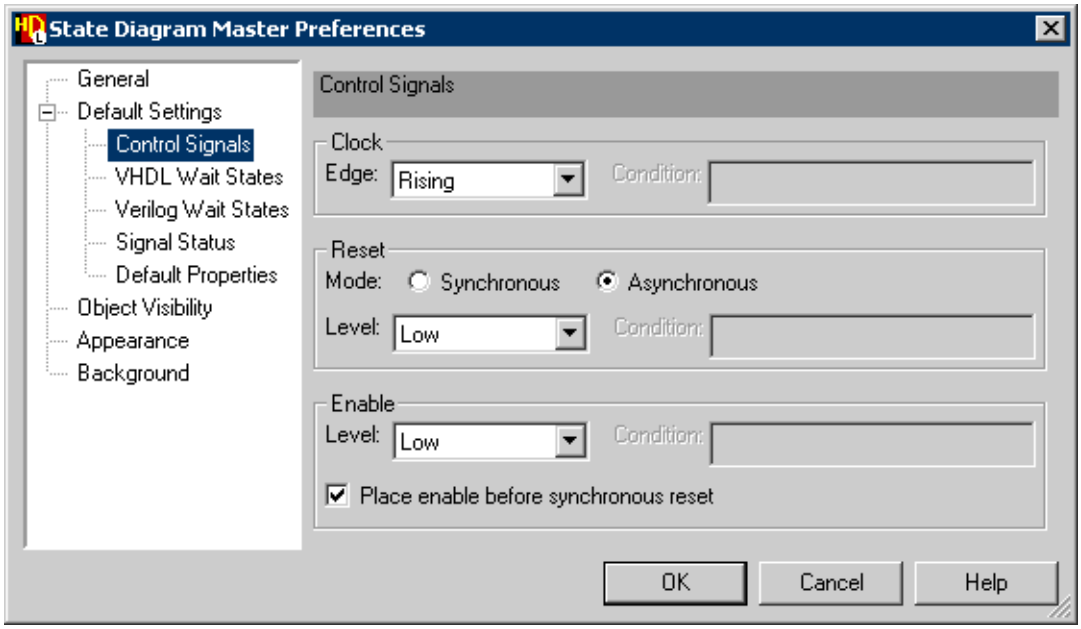
These options include the default radius of states, hierarchical states and transition priorities, the visibility of text objects, the transition style, syntax checking, the default save name for state diagrams, whether to register state actions on the next state and the default transition decoding style.



The **Default Settings** page allows you to set the default names for state diagram objects, transition conditions, transitions actions and state actions:

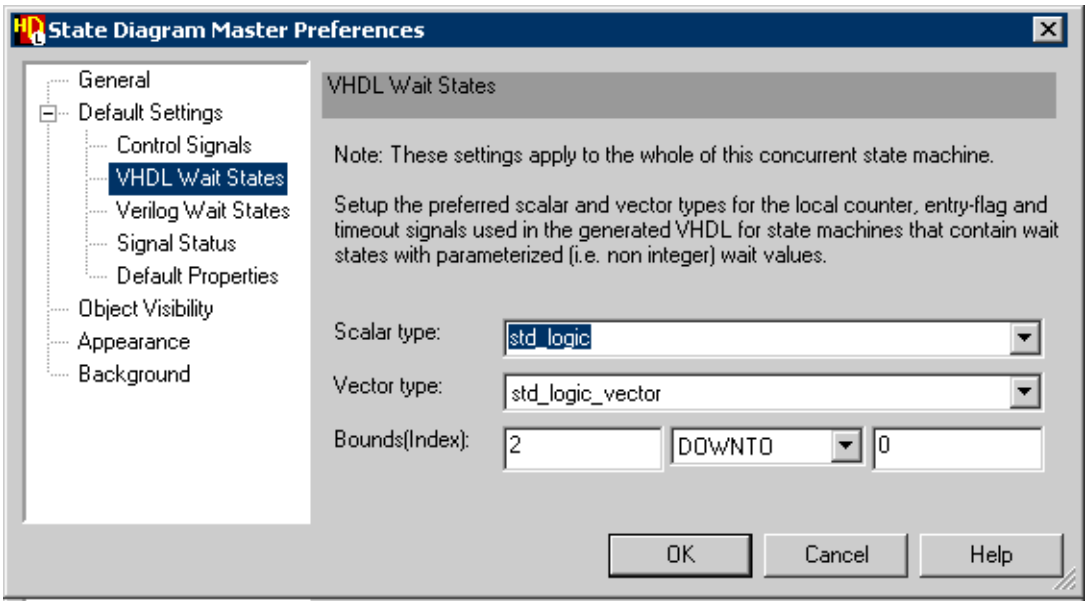


The **Control Signals** sub-page enables you to set the clock edge and clock condition, the reset mode, reset level, and reset condition, as well as the enable level and enable condition.



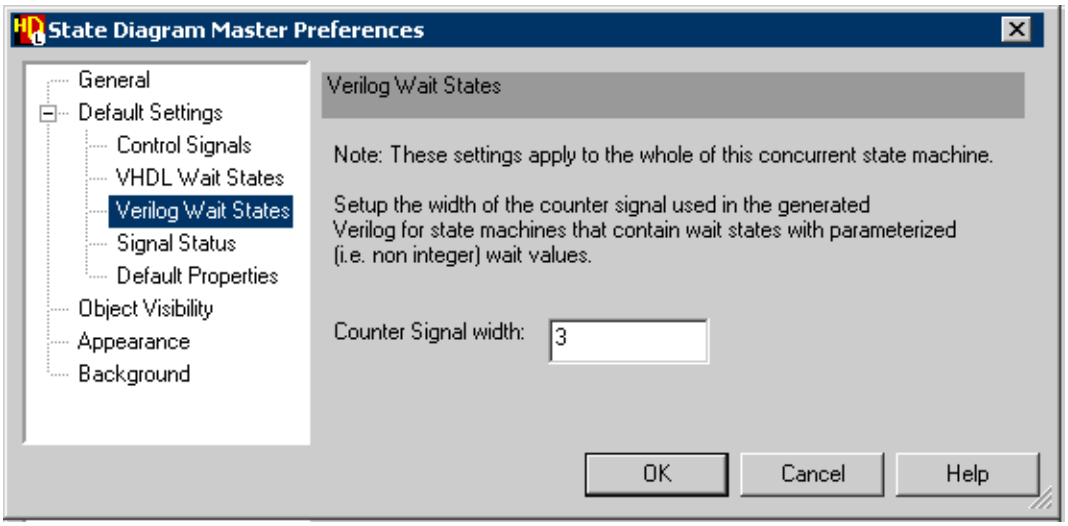
Separate **VHDL Wait States** and **Verilog Wait States** sub-pages can be used to set default options for wait states.

The following picture shows the VHDL sub-page:



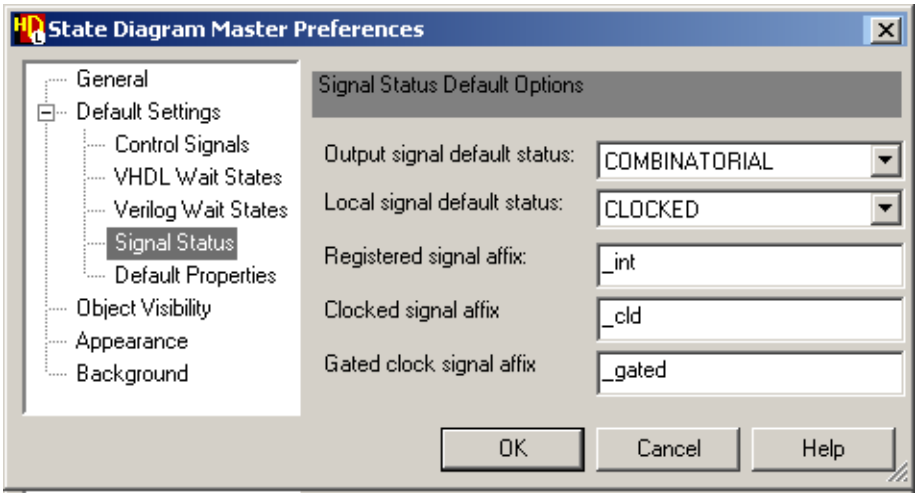
The VHDL sub-page allows you to set the default scalar and vector type of the local counter, entry flag and timeout signals for a non-integer value wait state.

The following picture shows the Verilog sub-page which allows you to set the width of the counter signal for a wait state with a non-integer value:



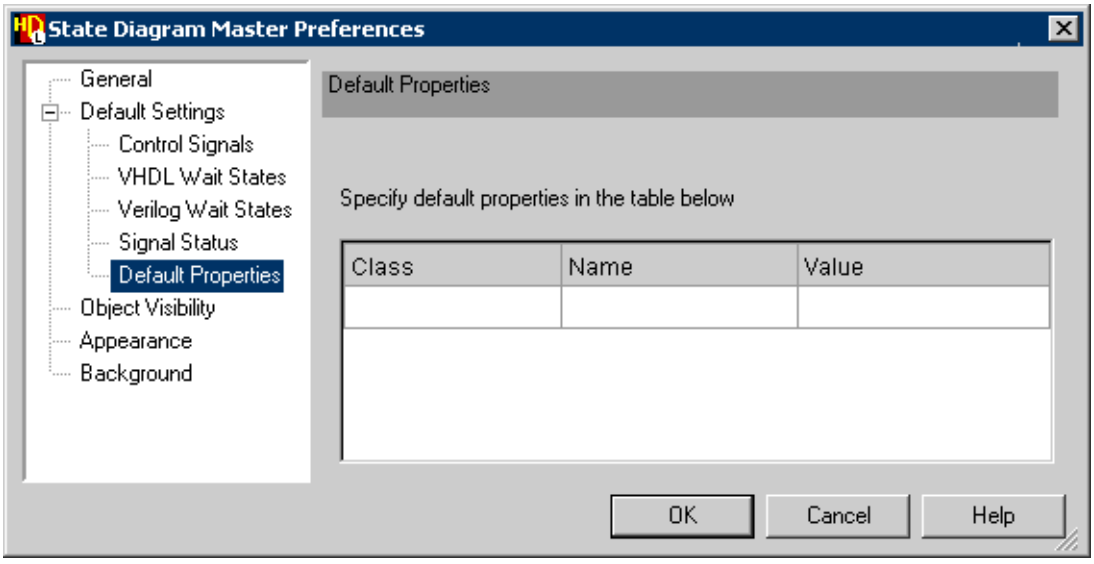
Refer to "Using Wait States" on page 58 for more information about wait states.

A separate **Signal Status Default Options** sub-page allows you to set default options for signal status:



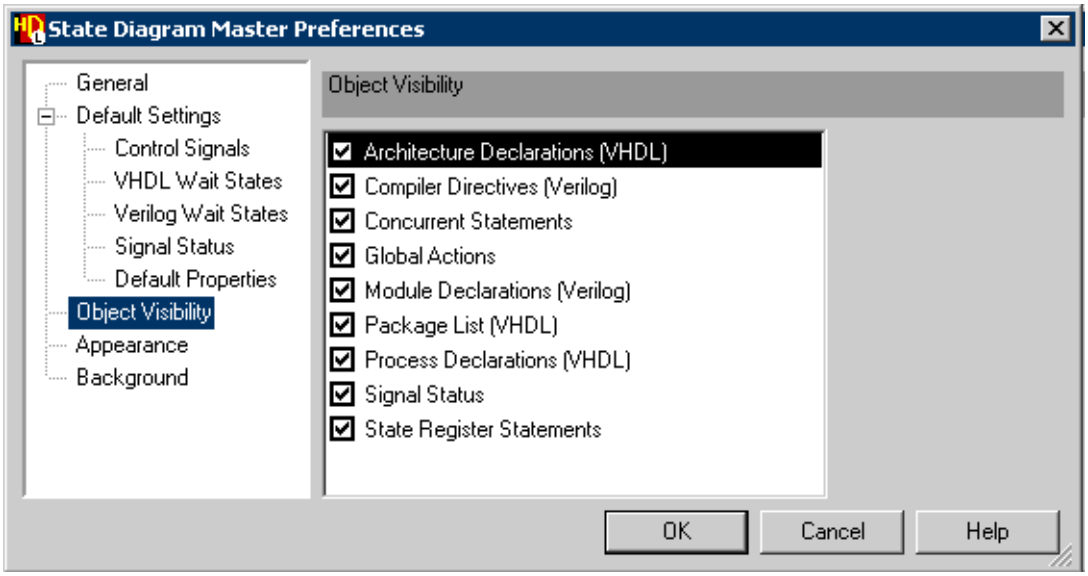
You can also specify the default suffix or prefix used for registered, clocked and gated clock signal names.

You can use the **Default Properties** sub-page to define default user properties for state diagram views:



Refer to the *HDL Designer Series User Manual* for information about “Using View Property Variables”.

The **Object Visibility** page allows you to set the default object visibility for multi-line text objects on the diagram.



Refer to “Changing Text Visibility” in the *Graphical Editors User Manual* for more information.

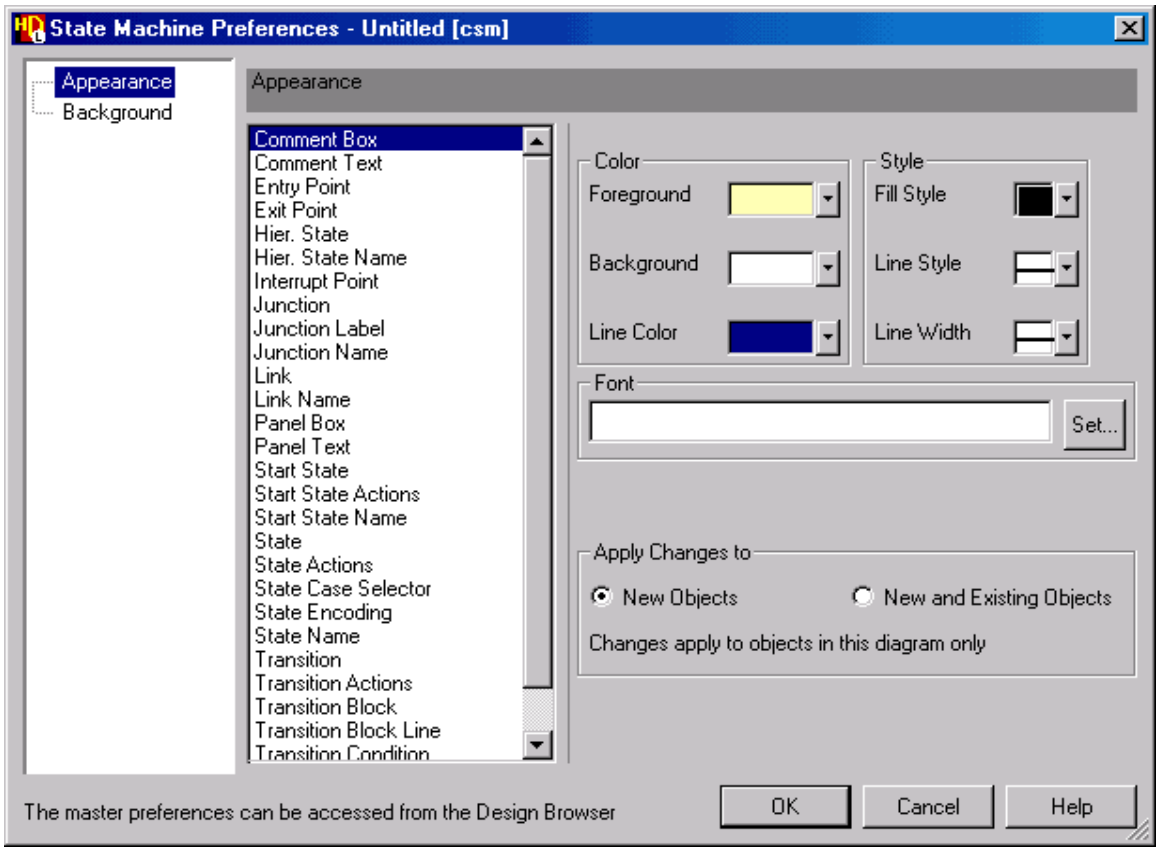
The **Appearance** page allows you to set default visual attributes for individual state diagram objects.

The attributes include the foreground and background colors, line color and style, fill style, line width and the text font. Some attributes may not always be available. For example, line style, width and color attributes are not available for a text object.

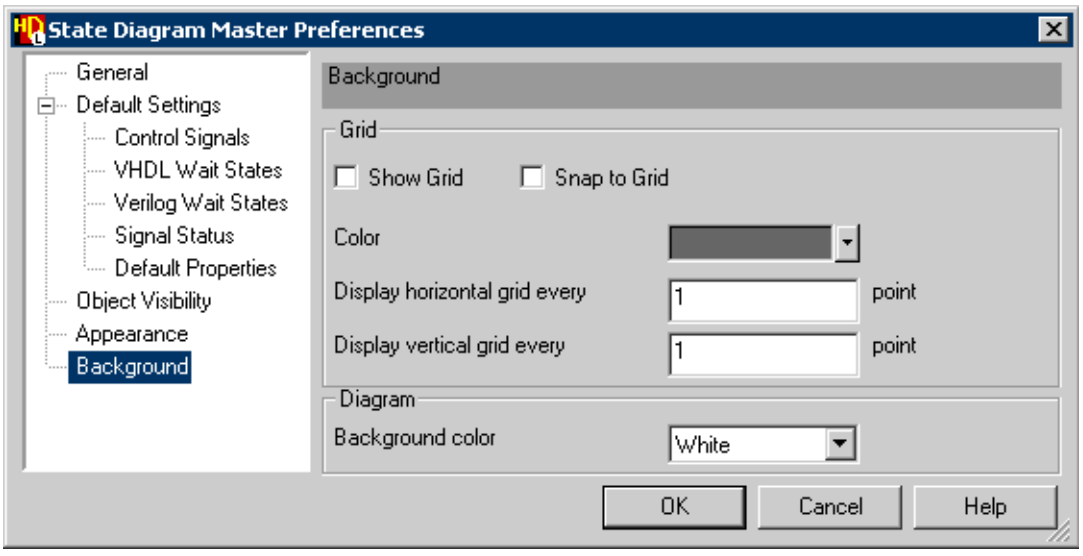
Refer to “Setting Visual Attributes” in the *Graphical Editors User Manual* for more information.

This page can also be edited by choosing **Diagram Preferences** from the **Options** menu in a state diagram. When you edit preferences for the active diagram, the dialog box allows you to choose whether the preferences are applied to new objects or to both new and existing objects in the state machine (including concurrent or hierarchical diagrams).

You can save the appearance preferences set on the active diagram as master preferences by choosing **Update from Diagram** in the **Master Preferences** cascade of the **Options** menu or you can apply the master preferences to the active diagram by choosing **Apply to New Objects** or **Apply to New and Existing Objects**.



The **Background** page allows you to control the diagram background color and grid attributes used by the state diagram editor.



Information about “Setting Background Preferences” is given in the *Graphical Editors User Manual*.

Chapter 3
ASM Chart Editor

This chapter describes the algorithmic state machine (ASM) editor.

ASM Chart Notation

The notation used for the graphical and text objects on an *ASM chart* is shown below:

Table 3-1. ASM Chart Notation

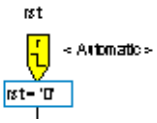
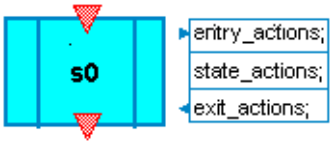
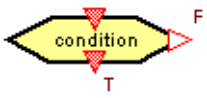
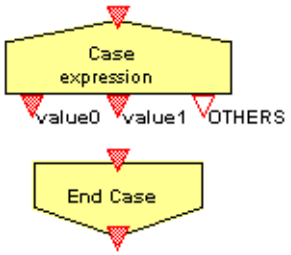
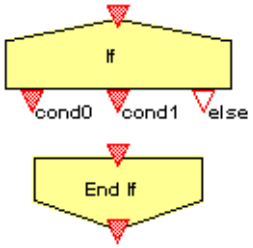
ASM Object	Description
	A *clock point* displays the clock signal name and clock condition. There must be one clock point on the diagram.
	An *enable point* displays the enable signal name and enable condition.
	A *reset point* displays the reset signal name, actions, mode and condition. It may have a priority if there are multiple reset points with the same mode on the diagram. Each reset point must be connected to a state box.
	An *interrupt point* is an implicit connection to all states on the diagram. It has an associated interrupt condition and may have a priority if there are more than one interrupt points on the diagram. Interrupts have priority over all other conditions.
	A *recovery state point* can be connected to indicate the flow to a recovery state.
	An *action box* contains HDL statements which are executed when the box is entered from a *flow*. There must be one input flow and one output flow.
	A *state box* represents observable status that the *ASM* can exhibit at a point in time. Encoding information is shown if manual encoding is enabled and there may be associated entry, state and exit *actions*.
	A *hierarchical action box* represents a child *ASM chart* describing action logic. A hierarchical action box has no associated state actions.
	A *hierarchical state box* represents a child *ASM chart* describing state transitions within a hierarchical ASM. A hierarchical state box has no associated state actions.
	A *decision box* represents if-then-else statements and has two outputs: A True *flow* which is followed when its condition is satisfied or a False *flow* otherwise.

Table 3-1. ASM Chart Notation

ASM Object	Description
	A *link* represents a direct transition to the named *state box*.
	A *case box* represents a CASE statement and has one or more outputs determined by the evaluation of a CASE expression. When used for decoding action logic below a *hierarchical action box*, each Case has an associated End Case object. Any number of other objects can be included in the flow between the Case and End Case.
	An *if decode box* represents an IF statement and has one or more outputs determined by the evaluation of a conditional expression. When used for decoding action logic below a *hierarchical action box*, each If has an associated End If object. Any number of other objects can be included in the flow between the If and End If.
	A *start point* is required on a child *ASM chart* below a *hierarchical action box* or *hierarchical state box*. There can only be one *start point* which is always named *Start*.
	A *end point* is required on a child *ASM chart* below a *hierarchical action box* or *hierarchical state box*. There must be at least one *end point*.
Pre and Post Global Actions	A statement block listing *global actions* that are always performed at the beginning of the output process for combinatorial signals or after the clock for registered signals.
Concurrent Statements	A statement block listing *concurrent statements* that are included in the generated HDL.
Architecture or Module Declarations	A list of user defined VHDL *architecture declarations* or Verilog *module declarations*.
Signals Status	A table showing the *signals status* of output and locally declared signals.
State Register Statements	A statement block listing statements which are included in the generated HDL as *state register statements*.
Clocked and Output Process Declarations	Separate lists of VHDL *process declarations* which are included in the clocked and output processes.

The ▼ icon on ASM chart objects indicates *port*s where flows can be connected. An ▽ icon on a decision box, case box or if decode box indicates an implicit loopback flow.

ASM Chart Toolbar

The following commands are available from the ASM Tools toolbar in the ASM chart editor:

Table 3-2. ASM Chart Toolbar

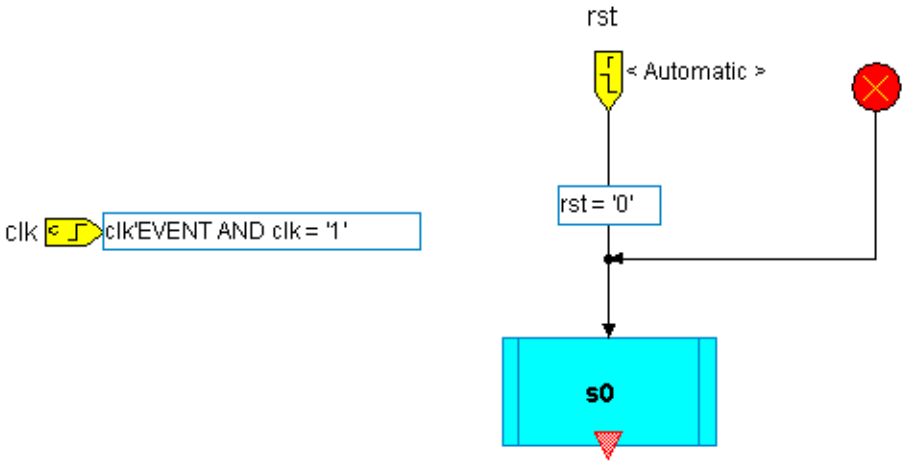
Button	Description
	Select text or objects
	Select text only
	Select objects only
	Add or modify comment text
	Pan the window
	Add an interrupt point
	Add a reset point
	Add a recovery state point
	Add an enable point
	Add an action box
	Add a state box
	Add a hierarchical action box
	Add a hierarchical state box
	Add a decision box
	Add a link
	Add a start point
	Add an end point
	Add a case box
	Add an if decode box
	Add a case or if choice
	Add a flow
	Add a panel

Refer to the *HDL Designer Series User Manual* for general information about toolbars and the HDL Designer Series user interface.

Refer to the *Graphical Editors User Manual* for information about selecting objects, adding comment text, panning the window, adding a panel and additional toolbars which are common to the other graphic editors.

ASM Initialization

A new *ASM chart* is initialized with a default *clock point*, *reset point*, *recovery state point* and *state box* as shown below:



The clock point is always shown unconnected although the clock signal is used for all state transitions in the ASM chart.

The *reset point* and *recovery state point* are connected by *flow*s to the *state box*.

There must be a single *clock point* on an ASM chart which cannot be deleted.

There can be only one *recovery state point* although it can be deleted if not required. If there is no flow connected to a recovery state point, the state box that is connected to the primary reset is the default recovery state.

Any number of *reset point*s, *state box*es or other objects can be added on the diagram.

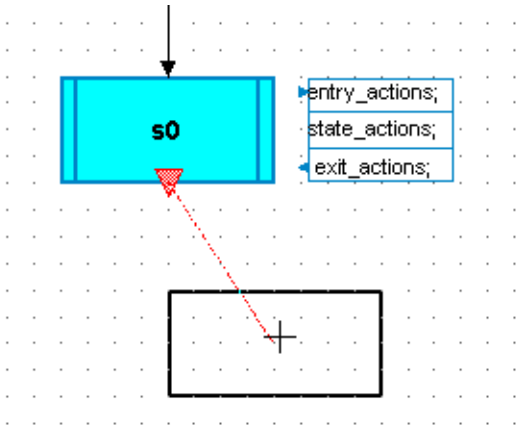
Adding Objects on an ASM Chart

You can add objects on an *ASM chart* using the **Add** menu or one of the buttons in the ASM Chart Tools toolbar.

Some objects can also be added using a shortcut key. Refer to the **Quick Reference Index**, which can be accessed from the Help and Manuals tab of the HDS InfoHub, for a list of supported Graphical Editor Shortcut Keys. To open the InfoHub, select **Help and Manuals** from the **Help** menu.

The cursor changes to a cross-hair which allows you to add the object by clicking at the required location on the diagram.

When you add any object (except an *interrupt point*, *reset point*, *recovery state point* or *start point*), a *flow* is automatically connected to the nearest unconnected *port* on an existing object. The ghosting shows which port the object will connect to. If there are several available ports, the ghost flow snaps between them as you move the cursor.

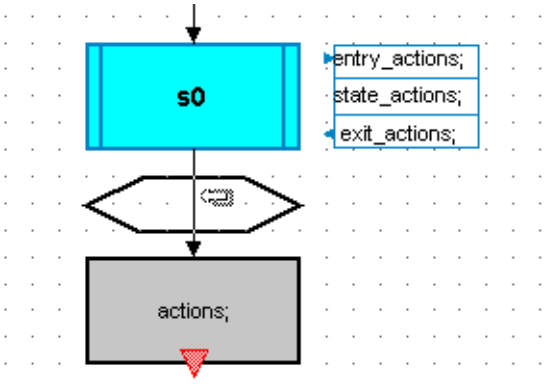


This automatic connection mode can be set or unset by choosing **AutoConnect** from the **Diagram** menu. The current setting is saved as a preference.

After adding an object, the command normally repeats until you use the **Esc** key (or **Right** mouse button) to terminate the command. However, you can set a preference for the command to remain active or activate only once and you can toggle this mode for the current command by using the **Ctrl** key.

If you move the cursor over an existing flow while you are adding an object, the cursor changes to  and the object is inserted into the flow between the existing objects.

The following example shows a decision box being inserted into the flow between a state box and an action box:



If the new object is too close to the object above it, it will automatically snap to a position in free space below the object. Any existing objects below the new object are automatically moved down to make space.

This automatic insertion mode can be set or unset by choosing **AutoInsert** from the **Diagram** menu. The current setting is saved as a preference.

Note

Automatic insert mode works only for vertical downward flows.

If an object is resized so that it overlaps the next object in a flow, the next object is automatically moved down to make space.

Adding an Interrupt Point

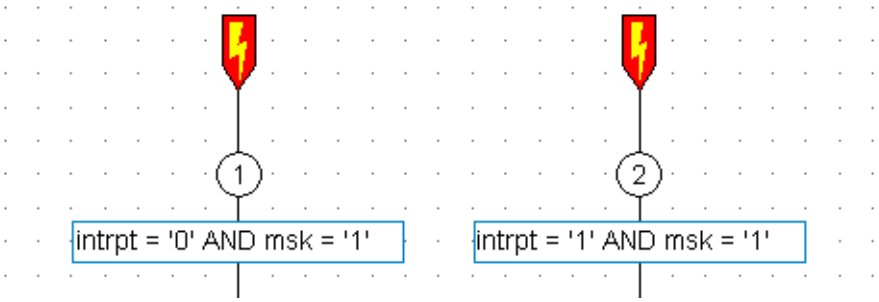
You can add an *interrupt point* to an ASM chart using the button or by choosing **Interrupt Point** from the **Add** menu.

You can change the interrupt condition or priority by clicking to select the text and clicking again to edit the text in-line. The condition syntax is automatically checked for the current hardware description language.

Alternatively, you can double-click on the interrupt point, use the button or choose **Object Properties** from the **Edit** menu, to display the **Interrupts** page in the ASM Object Properties dialog box as described in "Editing Interrupt Object Properties" on page 111.

A transition from an Interrupt point is a global interrupt which applies to the whole diagram and has priority over all other transitions.

Where more than one interrupt is defined on the same diagram, their evaluation order is determined by the priority.



Note

Interrupt points can only be added on the top level diagram of a hierarchical ASM chart.

Adding a Reset Point

You can add a *reset point* to an ASM chart using the button or by choosing **Reset Point** from the **Add** menu.

Note

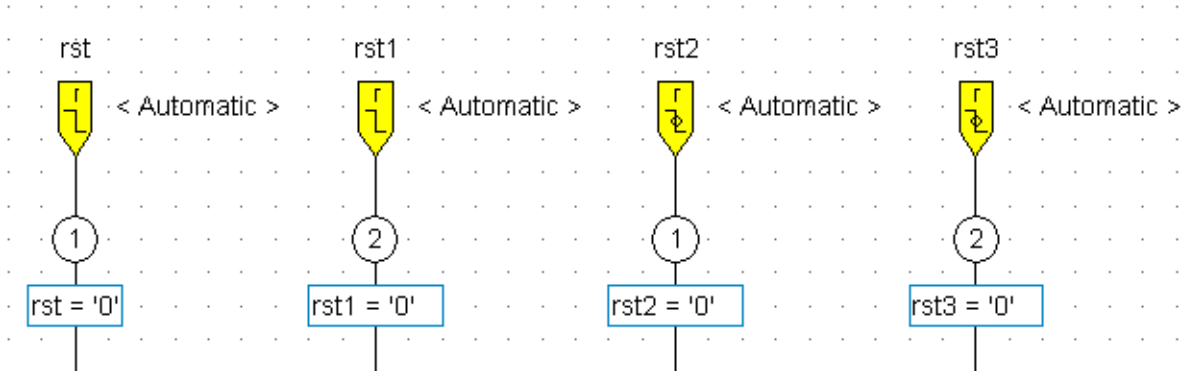
Reset points can only be added on the top level diagram of a hierarchical ASM chart and must be connected (directly or using a link) to a state box.

You can change the reset signal name, condition or priority by clicking to select the text and clicking again to edit the text in-line. The condition syntax is automatically checked for the current hardware description language.

Alternatively, you can double-click on the reset point, use the button or choose **Object Properties** from the **Edit** menu, to display the **Resets** page in the ASM Object Properties dialog box as described in "Editing Reset Object Properties" on page 103.

You can change the reset mode in the dialog box or by choosing **Synchronous** or **Asynchronous** from the **Reset Mode** cascade in the popup menu.

Where more than one reset with the same mode is defined on the diagram, their evaluation order is determined by the priority. However asynchronous resets take priority over all synchronous resets. For example, the following picture shows two asynchronous resets (*rst* and *rst1*) and two synchronous resets (*rst2* and *rst2*):



Tip: Note that the reset signal level and mode are indicated on the reset point icon.

The reset actions are automatically derived by default from the reset signal status but can be edited directly when specified actions are set in the Object Properties dialog box. The actions syntax is automatically checked for the current hardware description language on entry.

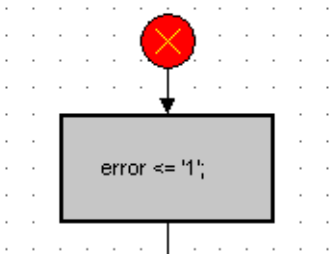
Adding a Recovery State Point

You can add a *recovery state point* to an ASM chart using the button or by choosing **Recovery State Point** from the **Add** menu.

Note

There can only be one recovery state point and this command is not available if a recovery state point already exists on the diagram. A recovery state point can only be added on the top level diagram of a hierarchical ASM chart.

The recovery state point should be connected by a flow to the state box to be entered when no other valid state is recognized in the next state process or *always* code. The flow can optionally include an action box defining recovery actions.



If there is no recovery state point, the recovery state is the state box connected to the highest priority reset state.

Adding an Enable Point

You can add an *enable point* to an ASM chart using the button or by choosing **Enable Point** from the **Add** menu.

Note

There can only be one enable point and this command is not available if an enable point already exists on the diagram. An enable point can only be added on the top level diagram of a hierarchical ASM chart.

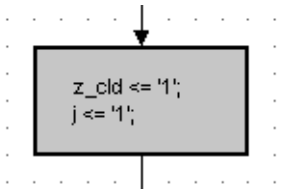
The enable signal and condition applies to the entire ASM chart and does not connect to any other objects on the diagram.

You can change the enable signal name by clicking to select the signal name and clicking again to edit the text in-line.

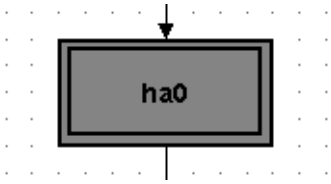
Alternatively, you can double-click on the enable point, use the button or choose **Object Properties** from the **Edit** menu, to display the **Enable** page in the ASM Object Properties dialog box as described in "Editing Enable Object Properties" on page 104.

Adding an Action Box

You can add an *action box* to an ASM chart using the button or by choosing **Action Box** from the **Add** menu.



You can add a *hierarchical action box* to an ASM chart using the button,**Shift**+**F2** shortcut key or by choosing **Hierarchical Action Box** from the **Add** menu.



You can change the enclosed actions (or the name of a hierarchical action box) by clicking to select the text and clicking again to edit the text in-line. The actions syntax is automatically checked for the current hardware description language.

Note

Note that an action box on an ASM chart is automatically resized to contain the enclosed actions or hierarchical action box name.

Alternatively, you can double-click on the action box, use the button or choose **Object Properties** from the **Edit** menu, to display the **Action Boxes** page in the ASM Object Properties dialog box as described in "Editing Action Box Object Properties" on page 105.

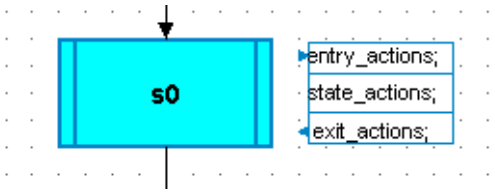
If you do not change the name of a hierarchical action box, each new hierarchical action box is given a unique name by adding an integer to the default name (for example, *ha0*, h*a1*, h*a2*..).

Complex actions can be described graphically using multiple action boxes, decision, case and if decode boxes. These can be on the same diagram or are typically grouped together in a hierarchical ASM chart below a hierarchical action box. A child ASM chart below a hierarchical action box always describes action logic and cannot contain state boxes.

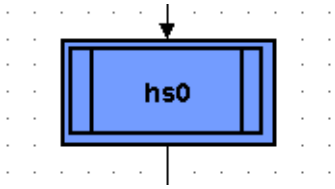
Child diagrams are saved as part of the parent ASM chart and are named after their parent hierarchical action box. However, only the hierarchical ASM charts below state boxes are displayed in the design explorer.

Adding a State Box

You can add an *state box* to an ASM chart using the button, **F3** shortcut key or by choosing **State Box** from the **Add** menu.



You can add a *hierarchical state box* to an ASM chart using the button,**Shift**+**F3** shortcut key or by choosing **Hierarchical State Box** from the **Add** menu.



You can change the name of the state box and the entry, state or exit actions for a non-hierarchical state box by clicking to select the text and clicking again to edit the text in-line. The actions syntax is automatically checked for the current hardware description language.

Note

Note that a state box or state actions text box is automatically resized to contain the enclosed name or actions. The default entry, state and exit actions specified in your preferences are displayed but can be edited or deleted if not required.

Alternatively, you can double-click on the state box, use the button or choose **Object Properties** from the **Edit** menu, to display the **States** page in the ASM Object Properties dialog box as described in "Editing State Object Properties" on page 104.

If you do not change the name of a state box or hierarchical state box, each new state box is given a unique name by adding an integer to the default name (for example, *s0*, *s1*, *s2*.. or *hs0*, hs*1*, hs*2*..).

If manual state encoding is enabled in the **Encoding** page of the ASM Properties dialog box, the default value <*encoding*> is written below the state name. This value can be edited by direct text editing or by using the **States** page of the ASM Object Properties dialog box.

Child diagrams are saved as part of the parent ASM chart and are named after their parent hierarchical state box. These hierarchical ASM charts are displayed in the design explorer.

Adding a Link

A *link* can be used as a connector to a *state box* with the specified name on the same diagram (or another diagram in the same hierarchical ASM).

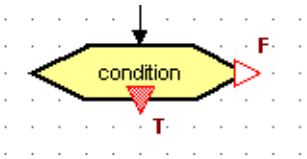
You can add a *link* to an ASM chart using the button, **Shift**+**F5** shortcut key or by choosing **Link** from the **Add** menu.

If there is only one existing state box on the diagram the link target defaults to that state. You can change the link target by clicking to select the text and clicking again to edit the text in-line.

Adding a Decision Box

A *decision box* can be used decode the next state (when the true and false flows end on different states) or for action logic (when both branches end on the same state).

You can add a *decision box* to an ASM chart using the button, **F4** shortcut key or by choosing **Decision Box** from the **Add** menu.



You can change the condition by clicking to select the text and clicking again to edit the text in-line. The decision box is automatically resized to contain the enclosed condition. The syntax is automatically checked for the current hardware description language.

Note

An unconnected *False* port (shown by an unfilled ▽ icon) is assumed to be an implicit loopback connection to the input flow for the previous state. However, you cannot have a loopback flow in an ASM chart used for decoding action logic below a hierarchical action box and both flows must be connected.

Alternatively, you can double-click on the decision box, use the button or choose **Object Properties** from the **Edit** menu, to display the **Decision Boxes** page in the ASM Object Properties dialog box as described in "Editing Decision Box Object Properties" on page 106.

You can move the *True* and *False* ports from a decision box to an alternative vertex by dragging them with the **Left** mouse button or by choosing **Swap True and False** from the **Diagram** or popup menu when a decision box is selected.

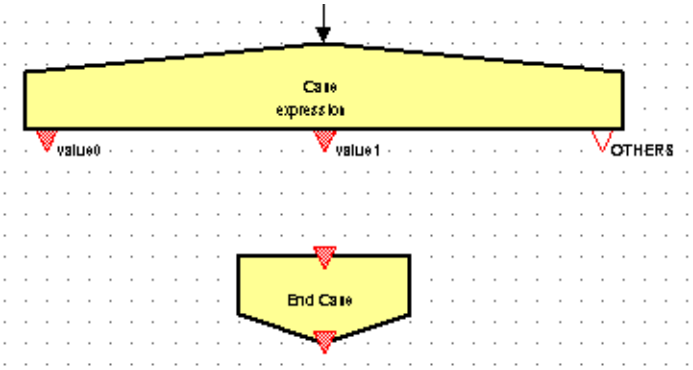
Any number of decision boxes can be nested to support *If...Then...Else* logic. However, multiple conditions can also be implemented by using an *if decode box* box as described in "Adding an If Decode Box" on page 98.

Adding a Case Box

A *case box* can be used decode the next state (when the branches end on different states) or for action logic (when all branches end on the same state).

You can add a *case box* to an ASM chart using the  button, **Shift**+**F4** shortcut key or by choosing **Case Box** from the **Add** menu.

The start case box is added at the cursor location and an associated end case object with the same name is automatically added vertically below.



Any combination of other ASM chart objects (including other case boxes) can be added between the start and end case objects.

The end case is not required for state decoding and can be hidden by choosing **Hide** from the popup menu. However, the end case is required to determine the end points for all possible branches when used for action logic. The end case can be displayed by choosing **Show End Case** from the popup menu for the start case box.

You can add additional ports to the case box by using the  button or **F6** shortcut key or by choosing **Port** from the **Add** menu or **Add Port** from the popup menu when the case box is selected or simply by adding flows with their origin over the start case object.

You can delete a port on a case box by using the **Del** key or by choosing **Delete** from the **Edit** or popup menu while the port is selected.

Note

An unconnected *OTHERS* port (VHDL) or *default* port (Verilog) shown by an unfilled ▽ icon is assumed to be an implicit loopback connection to the input flow for the previous state. However, you cannot have a loopback flow in an ASM chart used for decoding action logic below a hierarchical action box.

You can change the condition expression or port values by clicking on the expression or value to select the text and clicking again to edit the text. The syntax is automatically checked for the current hardware description language.

Alternatively, you can double-click on the case box, use the button or choose **Object Properties** from the **Edit** menu, to display the **Case Boxes** page in the ASM Object Properties dialog box as described in "Editing Case Box Object Properties" on page 109.

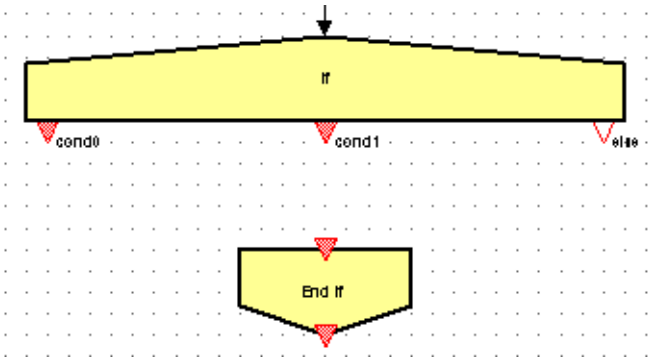
The expression is normally placed inside the start case object but can be moved independently away from (or into) the object. If you want to contain the whole of the expression inside the case box or add ports, it may be necessary to resize the object.

Adding an If Decode Box

A *if decode box* can be used in a similar way to a *case box* to decode the next state (when the branches end on different states) or for action logic (when all branches end on the same state). The if decode box is equivalent to nesting multiple *decision boxes* with much less clutter on the diagram.

You can add an *if decode box* to an ASM chart using the button, **Ctrl**+**F4** shortcut key or by choosing **If Decode Box** from the **Add** menu.

The start if decode box is added at the cursor location and an associated end if object with the same name is automatically added vertically below.



Any combination of other ASM chart objects (including other decode boxes) can be added between the start and end if objects.

The end if is not required for state decoding and can be hidden by choosing **Hide** from the popup menu. However, the end case is required to determine the end points for all possible branches when used for action logic. The end if can be displayed by choosing **Show End If** from the popup menu for the start if decode box.

You can add additional ports to the if decode box by using the button or **F6** shortcut key or by choosing **Port** from the **Add** menu or **Add Port** from the popup menu when the case box is selected or simply by adding flows with their origin over the start if decode object.

You can delete a port on an if decode box by using the **Del** key or by choosing **Delete** from the **Edit** or popup menu while the port is selected. However, an if decode box must have a minimum of three output condition ports.

An unconnected *else* port (shown by an unfilled icon) is assumed to be an implicit loopback connection to the input flow for the previous state. However, you cannot have a loopback flow in an ASM chart used for decoding action logic below a hierarchical action box.

You can change the port expressions by clicking on the expression to select the text and clicking again to edit the text. The syntax is automatically checked for the current hardware description language.

Alternatively, you can double-click on the if decode box, use the button or choose **Object Properties** from the **Edit** menu, to display the **If Decode Boxes** page in the ASM Object Properties dialog box as described in "Editing If Decode Box Object Properties" on page 108.

Adding a Flow

You can add a *flow* to a ASM chart using the button or **F7** shortcut key or by choosing **Flow** from the **Add** menu.

The cursor changes to a cross-hair which allows you to add a flow by clicking the **Left** mouse button with the cursor over a source and destination plus any number of route points.

Flows can only be connected between the connect ports shown by on each ASM chart object. However, you can move the True and False flows from a decision box to an alternative vertex.

Note

You can dynamically create a port on a case box or an if decode box by adding a flow with its origin over the start case or start if object.

A flow cannot originate on another flow but can be terminated on a flow (creating a flow join).

Note that if you delete an object (such as an action box) which has one input flow and one output flow, a flow is automatically connected between the objects immediately above and below the deleted object.

Hierarchical ASM Charts

You can open down to create a *child* hierarchical ASM chart by selecting a *hierarchical state box* or a *hierarchical action box* and double-clicking or **Open Down** from the **Open** cascade of the **File** menu (or popup menu).

You can open down by double-clicking on a hierarchical state box or by choosing **Open Down** from the **Open** cascade of the **File** menu (or popup menu).

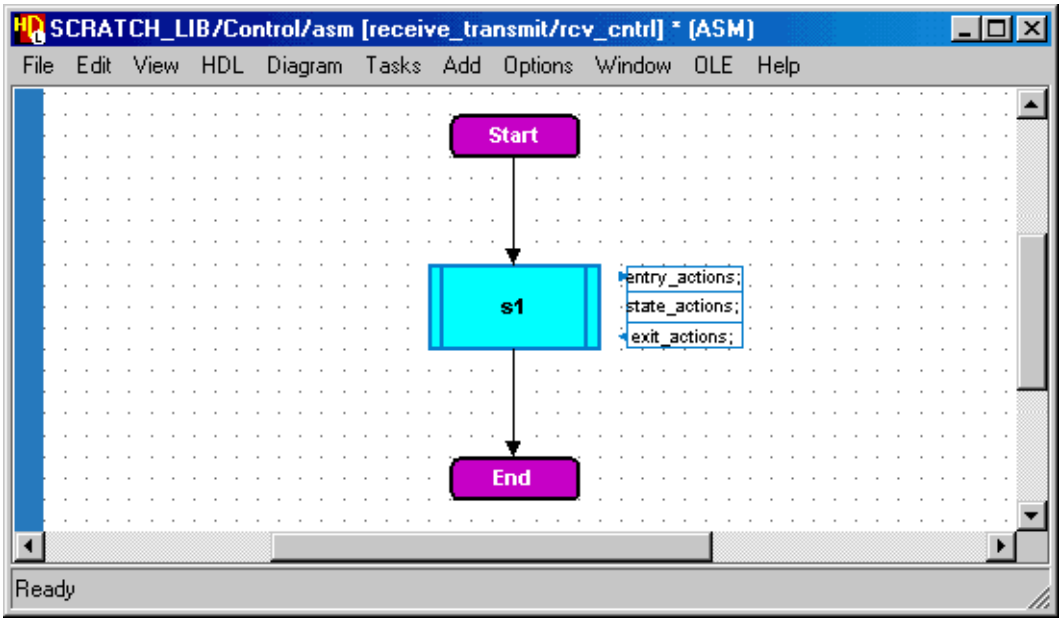
The child diagram is opened in the existing window. A new child ASM chart comprises a *start point*, a single *state box* or *action box* and an *end point* connected by flows.

You can edit a hierarchical diagram in the same way as any other ASM chart including *action boxes*, *decision boxes*, *case box* and *if decode boxes*. However a *state box* or *link* can only be added in the hierarchical diagram below a state box. Named links can be used to access any state box with the specified name in the hierarchical ASM chart.

You can choose **Open Up** from the **File** menu or select the name of the parent diagram in the *diagram browser* to open the parent of the currently active ASM chart.

Child diagrams are saved as part of the parent ASM chart and named after the parent hierarchical state box or hierarchical action box by adding its name to the ASM chart name.

For example, the following child ASM chart is created when you open down from a hierarchical state box named *rcv_cntrl* in the ASM chart named *receive_transmit*:



Adding a Start Point

An *start point* is automatically created when you create a child ASM chart (by opening down from its parent hierarchical state box or hierarchical action box). Only one start point is allowed on a diagram although it can be deleted.

You can add an start point to a child ASM chart using the  button or **Shift**+**F8** shortcut key or by choosing **Start Point** from the **Add** menu.

Adding an End Point

An *end point* is automatically created when you create a child ASM chart by opening down from its parent hierarchical state or hierarchical action box but adding multiple entry points can help reduce diagram complexity.

You can add end points to a child ASM chart using the  button or **F8** shortcut keys or by choosing **End Point** from the **Add** menu.

Each exit point connects to the parent hierarchical state box or action box but you can also exit from a child diagram below a hierarchical state box using named links that connect to the specified state at any level in the hierarchical ASM chart.

Editing ASM Object Properties

You can edit many object properties (including condition expressions and actions) directly on the diagram by clicking to select the text and clicking again to edit the text object.

An expression builder dialog box is automatically displayed when you begin to enter a condition expression or action statement. Refer to "Building a HDL Expression" on page 16 for more information about the expression builder.

The HDL syntax for expressions and actions is automatically checked for the language of the diagram you are using (VHDL or Verilog) although the syntax checking can be disabled by unsetting a preference.

Note

Note that you must include a terminating semi-colon after action statements although line breaks and indents can be used to improve legibility.

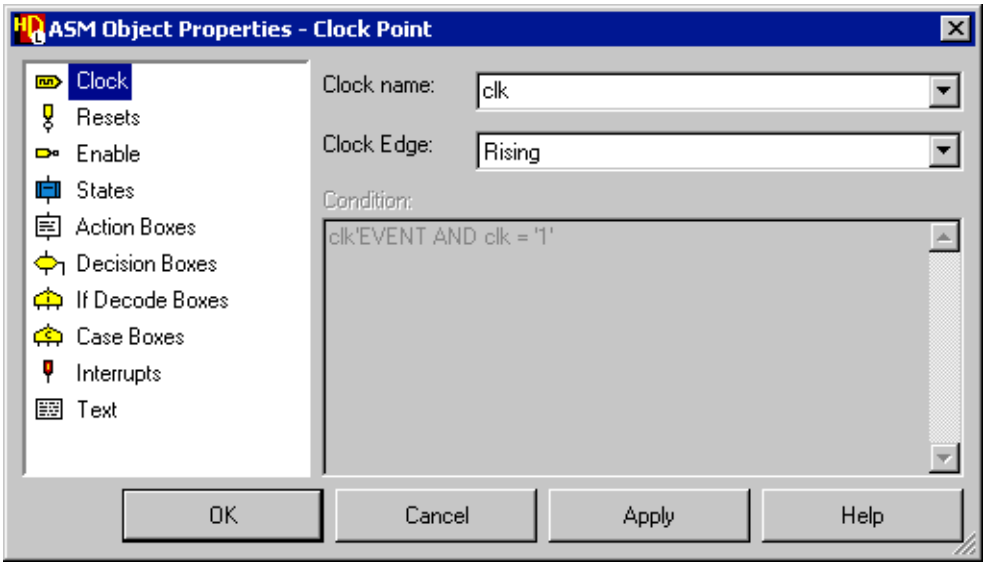
You can also edit the properties for an object on an ASM chart by double-clicking on an object, using the  button or by choosing **Object Properties** from the **Edit** menu or popup menu to display an Object Properties dialog box.

The dialog box has separate pages for **Clock**, **Resets**, **Enable**, **States**, **Action Boxes**, **Decision Boxes**, **If Decode Boxes**, **Case Boxes**, **Interrupts** and **Text** objects.

The editable objects are shown in the left pane of the dialog box. Objects which exist in the current selection set are highlighted in yellow. Objects that are not available in the current selection are shown in dimmed font.

Editing Clock Object Properties

The **Clock** page of the ASM Object Properties dialog box allows you to specify the clock signal and set the clock edge sensitivity.



You can choose the clock signal name from a dropdown list of available input signals. Note that any signals starting with *clk* or *clock* take precedence in the list. For a Verilog view, you can choose *Rising* or *Falling* representing *posedge* or *negedge* sensitivity. For a VHDL view, you can choose *Rising*, *Falling*, *Rising Last*, *Falling Last*, *Rising Edge* or *Falling Edge*. These options generate the following VHDL expressions:

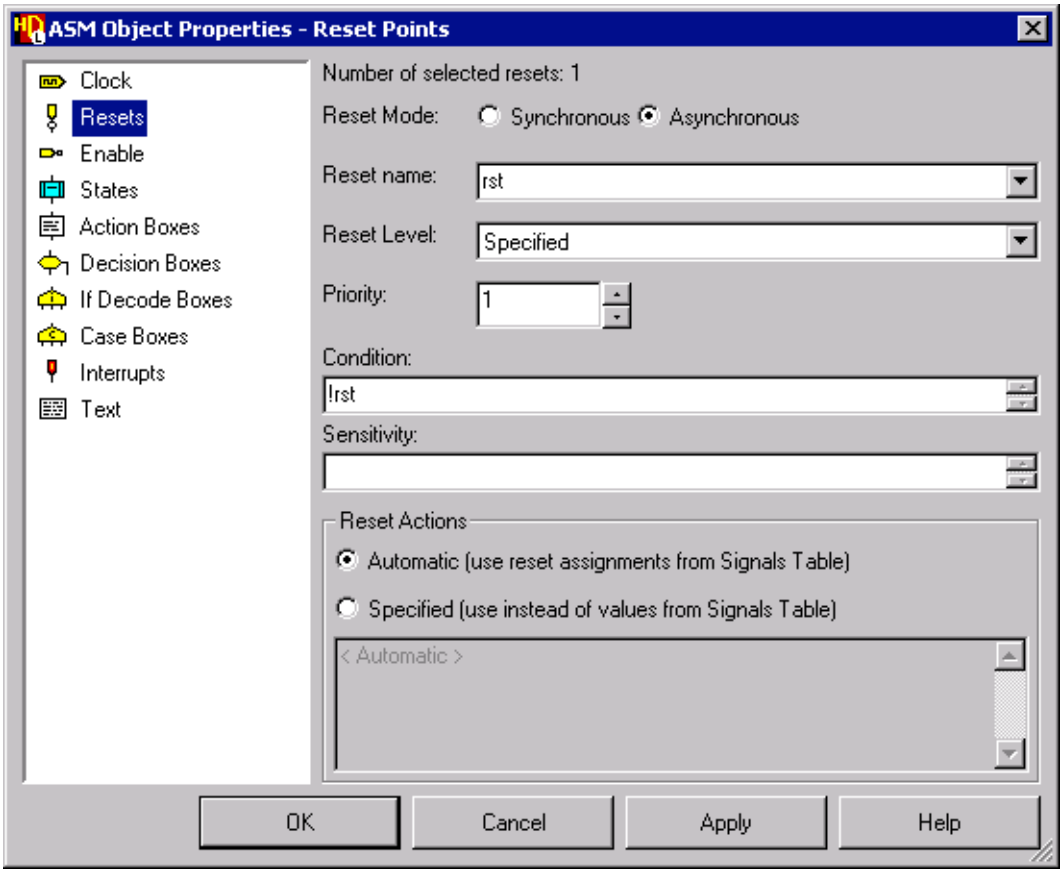
Rising	clk'EVENT AND clk = '1'
Falling	clk'EVENT AND clk = '0'
Rising Last	clk'EVENT AND clk = '1' AND clk'LAST_VALUE = '0'
Falling Last	clk'EVENT AND clk = '0' AND clk'LAST_VALUE = '1'
Rising Edge	rising_edge(clk)
Falling Edge	falling_edge(clk)

Tip: Note that the clock edge is indicated by a rising or falling waveform on the clock point icon.

Alternatively for either language, you can choose *Specify* to enter any other valid edge condition.

Editing Reset Object Properties

The **Resets** page of the ASM Object Properties dialog box allows you to specify a synchronous or asynchronous mode reset and specify the reset signal.



You can choose the reset signal name from a dropdown list of available input signals. Note that any signals starting with *rst* or *reset* take precedence in the list.

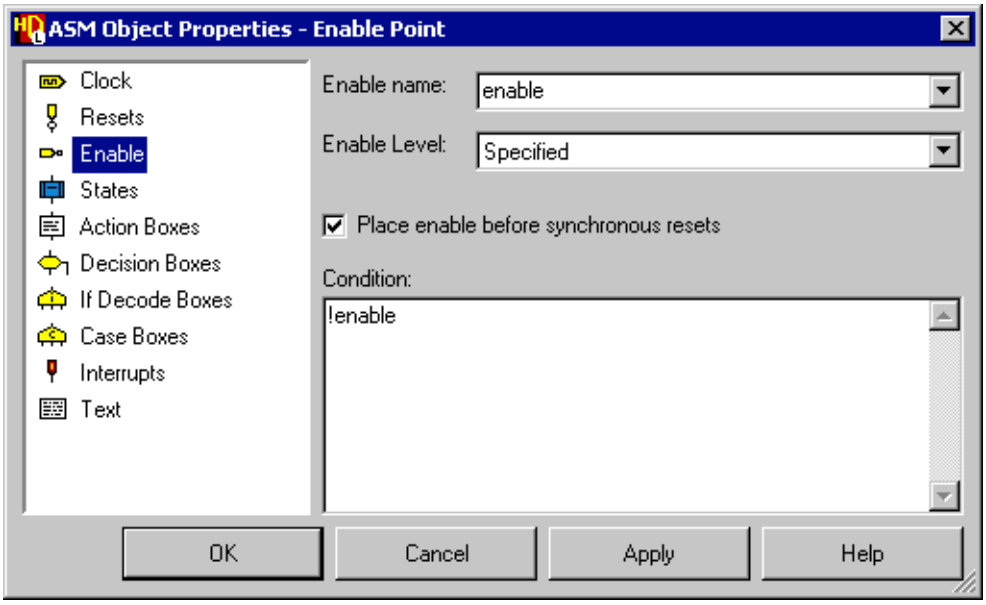
You can specify whether the reset signal is active low, high or when a specified condition is evaluated. If multiple resets with the same mode are defined on the diagram you also can set the reset priority. However, asynchronous resets take priority over all synchronous resets.

You can optionally specify reset actions. If set to <*Automatic*>, the reset actions are automatically derived from the reset values specified in the signals table. Refer to "Signals Table" on page 127 for information about setting reset values in the signals status.

If you have specified a Verilog reset condition, you must also specify any additional signals required in the sensitivity list. (Multiple signals should be separated by an *OR* operator.)

Editing Enable Object Properties

The **Enable** page of the ASM Object Properties dialog box allows you to specify an enable signal and set the enable signal level.



You can choose the enable signal name from a dropdown list of available input signals. Note that any signals starting with *en* or *enable* take precedence in the list.

The enable signal can be active low, high or when a specified condition is evaluated.

You can also specify whether the enable should be placed before any synchronous reset signals in the generated HDL.

Editing State Object Properties

The **States** page of the ASM Object Properties dialog box allows you to specify the properties for the selected state box (or state boxes).

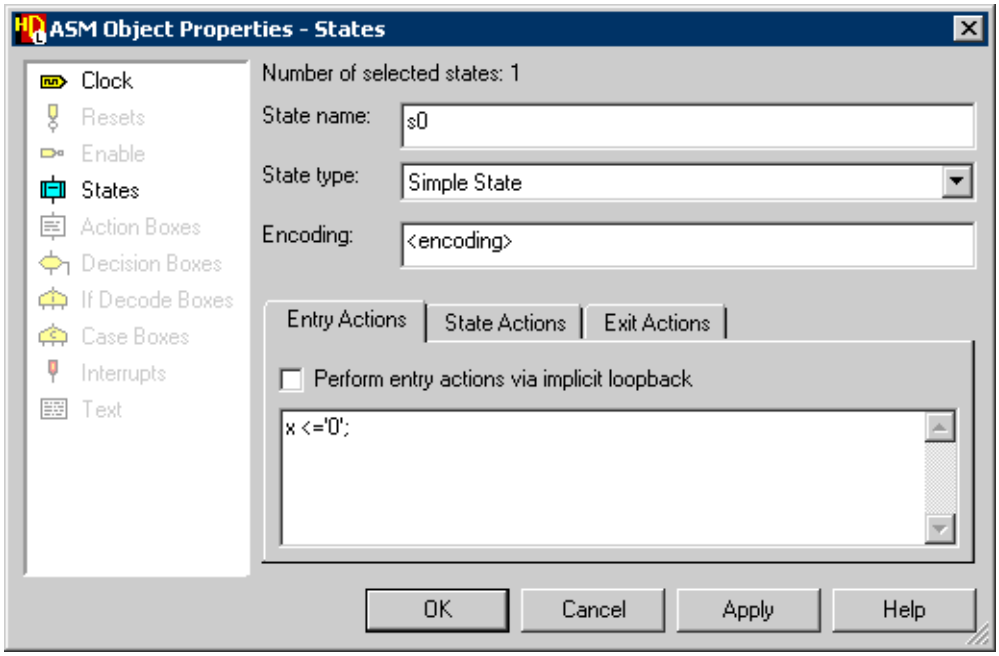
The state name must be unique and can only be applied to a single selected state.

You can change one or more selected states to be a *simple state* or *hierarchical state* by choosing from a pulldown list of state types.

If you change a hierarchical state to a non-hierarchical state, the child ASM chart (if it exists) and its contents are discarded. However, you can undo this change to recover the hierarchical state and its child ASM chart. If you change a simple state box to a hierarchical state, any existing actions are transferred to a default state box in the child ASM chart.

If manual state encoding mode is enabled in the **Encoding** page of the ASM Properties dialog box, an Encoding entry field is disclosed which allows you to enter a binary or decimal constant encoding or enumerated attribute on the state box.

Refer to "Setting ASM Chart Generation Properties" on page 113 for information about setting state encoding options.



You can add or edit entry, state and exit actions defined in the state box. Entry actions are performed on all incoming transitions. However, you can choose whether the entry actions are performed on implicit loopback transitions. Exit actions are performed on all outgoing transitions except implicit loopbacks but are performed for explicit loopbacks (when the loopback connection is made by a flow on the diagram).

If more than one state box is selected, the actions in the dialog box are applied to all the selected state boxes.

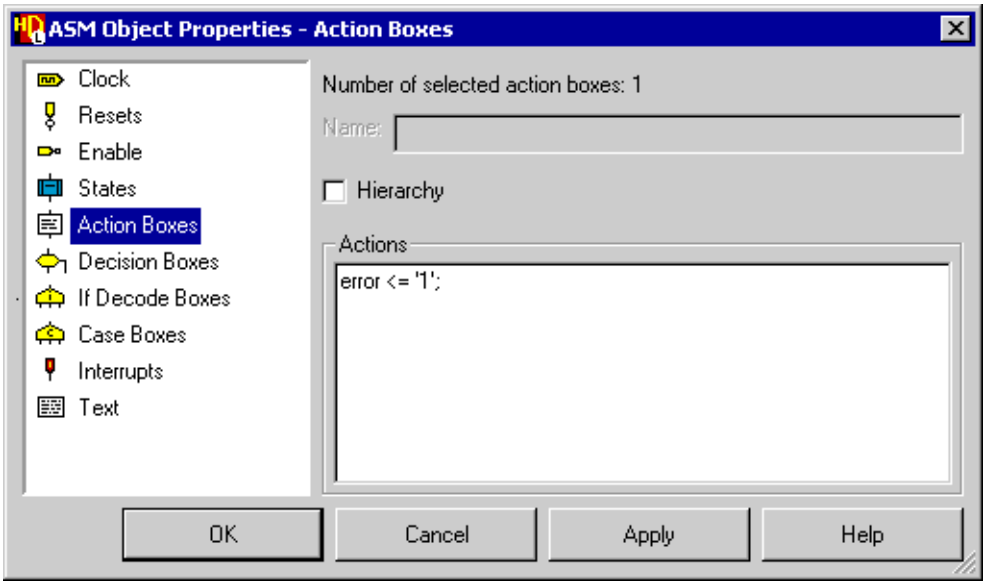
Editing Action Box Object Properties

The **Action Boxes** page of the ASM Object Properties dialog box allows you to specify the name for a hierarchical action box or the action statements for a non-hierarchical action box.

The name must be unique and can only be applied to a single selected hierarchical action box. However, the actions are applied to all selected action boxes.

You can also change the action box hierarchy. If you change a hierarchical action box to a non-hierarchical action box, the child ASM chart (if it exists) and its contents are discarded.

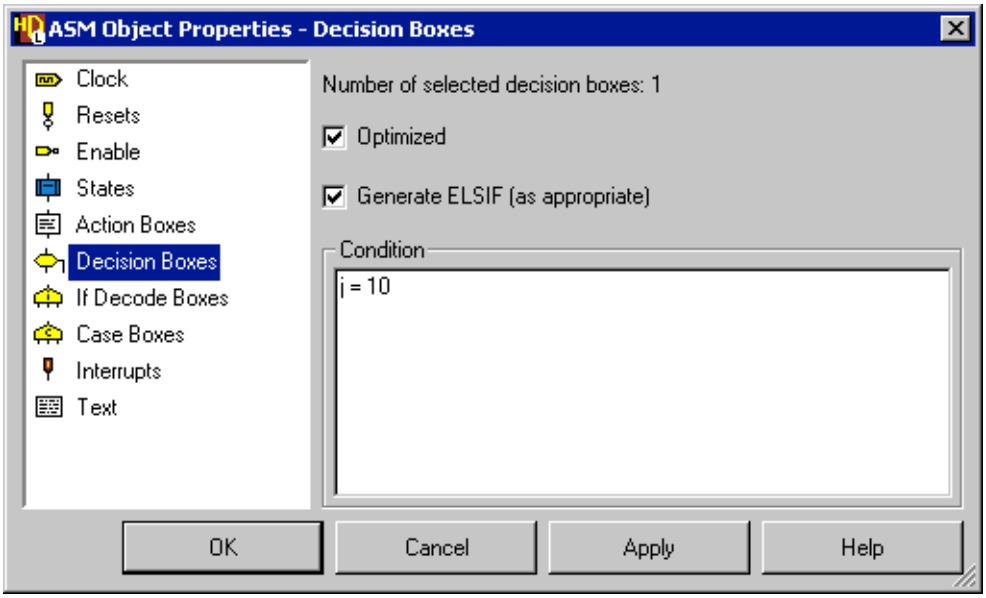
However, you can undo this change to recover the hierarchical action box and its child ASM chart.



If you change an action box to a hierarchical action box, any existing actions are transferred to a default action box in the child ASM chart.

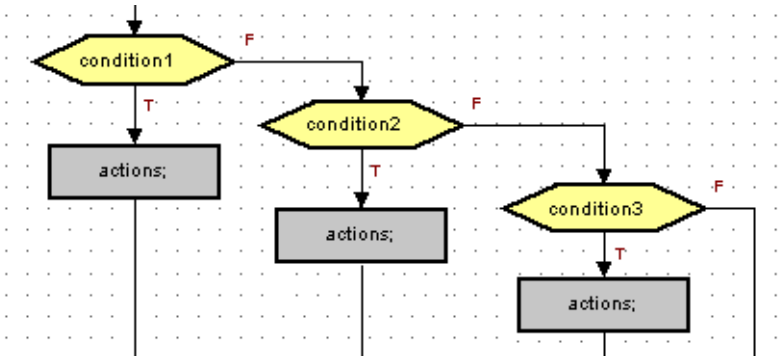
Editing Decision Box Object Properties

The **Decision Boxes** page of the ASM Object Properties dialog box allows you to specify the properties for a decision box.



When the **Optimized** option is set and both branches meet, the condition is assumed to be used for action logic and no state transition is generated in the HDL. However, you can unset the optimize option to generate a transition for each branch. This option is ignored and a transition is generated when the decision branches do not meet. However, the optimize option cannot be unset and a HDL generation error is issued if the branches do not meet when used below a hierarchical action box.

By default if your ASM chart uses nested decision boxes, combined *ELSIF* (VHDL) or *else if* (Verilog) statements are generated for the false branch if the decision box is the first and only statement on the false branch of another decision box. For example, *c2* and *c3* in the following example:



VHDL:

```
IF c1 THEN
    action1;
ELSIF c2 THEN
    action2;
ELSIF c3 THEN
    action3;
END IF;
```

Verilog:

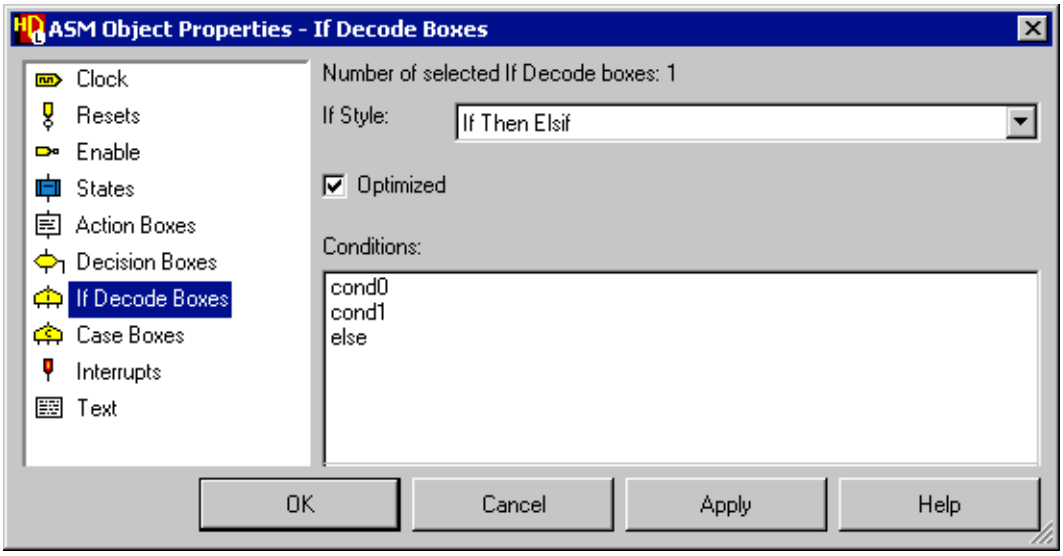
```
if (c1) begin
   action1;
end
else if (c2) begin
   action2;
end
else if (c3) begin
   action3;
end
```

If your downstream tool does not support combined else and if, you can unset this option in the dialog box.

You can add or edit the condition defined in the decision box. If more than one decision box is selected, the condition is applied to all the selected boxes.

Editing If Decode Box Object Properties

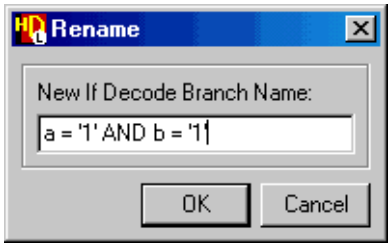
The **If Decode Boxes** page of the ASM Object Properties dialog box allows you to specify the properties for an if decode box.



You can choose whether to use *If Then Else*, *If Then Elsif* or *If Then* statements for nested if decode boxes.

When the **Optimized** option is set and all branches meet, the conditions are assumed to be used for action logic and no state transitions are generated in the HDL. However, you can unset the optimize option to generate a transition for each branch. This option is ignored and a transition is generated when the branches do not meet. However, the optimize option cannot be unset and a HDL generation error is issued if the branches do not meet when used below a hierarchical action box. Note that all branches must meet if an end if is shown on the diagram.

The dialog box also lists the existing port names for each if decode condition branch. These names should be conditional expressions and can be edited by clicking over the existing name in the dialog box to display a Rename dialog box.



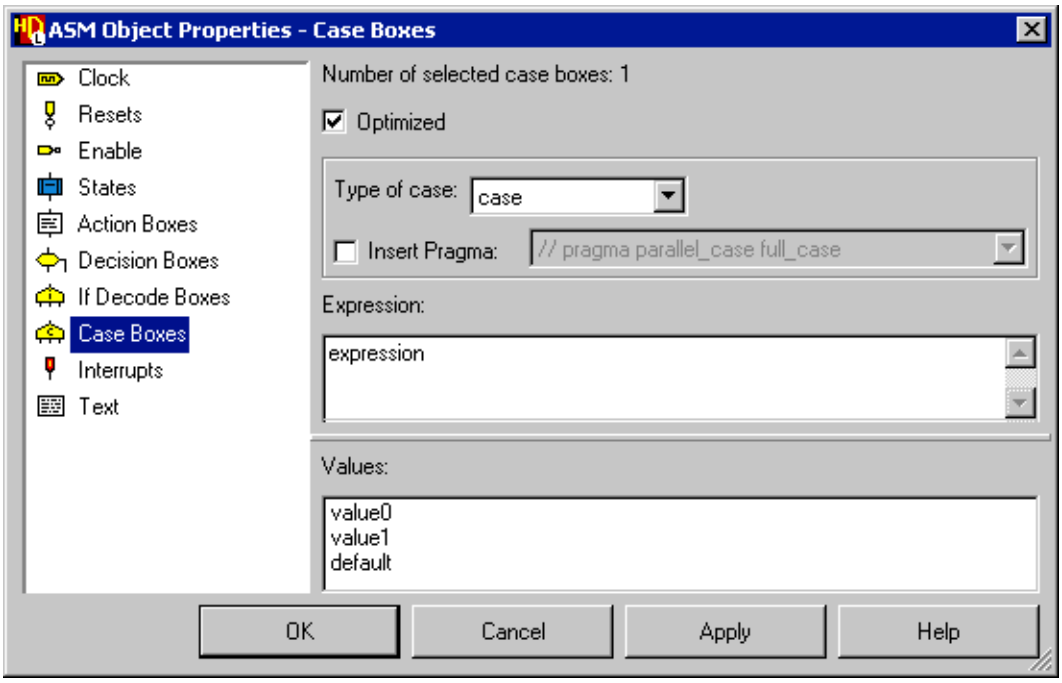
Two default condition ports and an *else* branch are provided by default. Additional ports can be added to the if decode box using the button as described in "Adding an If Decode Box" on page 98.

Editing Case Box Object Properties

The **Case Boxes** page of the ASM Object Properties dialog box allows you to specify the properties for a case decode box.

When the **Optimized** option is set and all branches meet, the case box is assumed to be used for action logic and no state transitions are generated in the HDL. However, you can unset the optimize option to generate a transition for each branch. This option is ignored and a transition is generated when the branches do not meet. However, the optimize option cannot be unset and a HDL generation error is issued if the branches do not meet when used below a hierarchical action box. Note that all branches must meet if an end case is shown on the diagram.

If you are using Verilog, you can choose to use *casex* or *casez* comparison as an alternative to the default bit comparison *case* style.

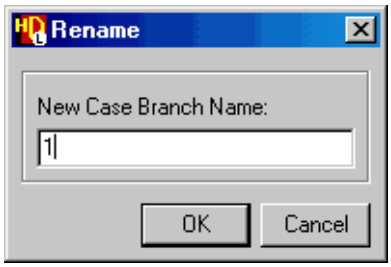


You can also insert the following pragmas to specify full case or parallel case Verilog statements:

full_case	All possible branches have been specified, any missing branches cannot occur and a default branch need not be generated.
parallel_case	Branches are mutually exclusive.
parallel_case full case	All possible branches have been specified and are mutually exclusive.

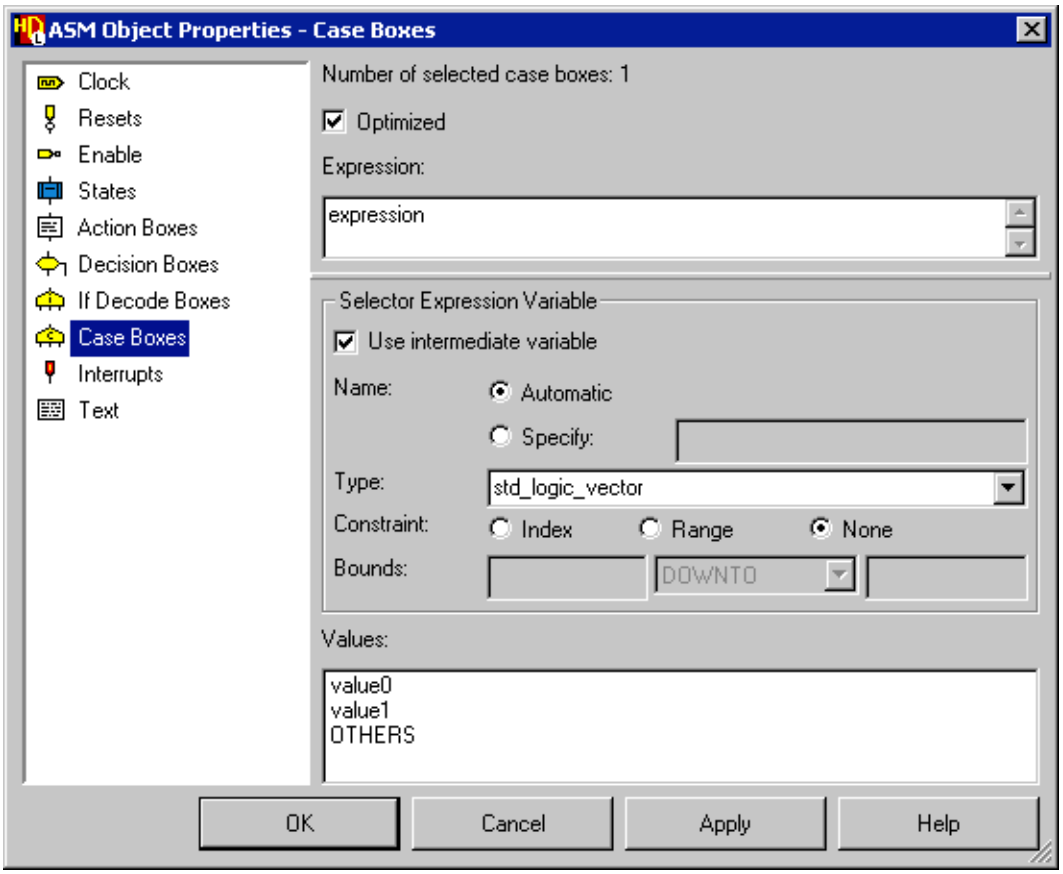
You can specify a case expression which can comprise any valid HDL statements for the current hardware description language (using multiple lines if required).

The dialog box lists the existing port names for each case decode branch which should correspond to the possible values for the evaluated expression. These values can be edited by clicking over the existing name in the dialog box to display a Rename dialog box.



Two default values and a *default* (if you are using Verilog) or *OTHERS* (if you are using VHDL) branch are provided by default. Additional ports can be added to the case box using the button as described in "Adding a Case Box" on page 97.

If you are using VHDL, you can specify an intermediate selector expression variable.



An intermediate variable may be required if the expression is not locally static (for example, if the expression contains concatenated signals).

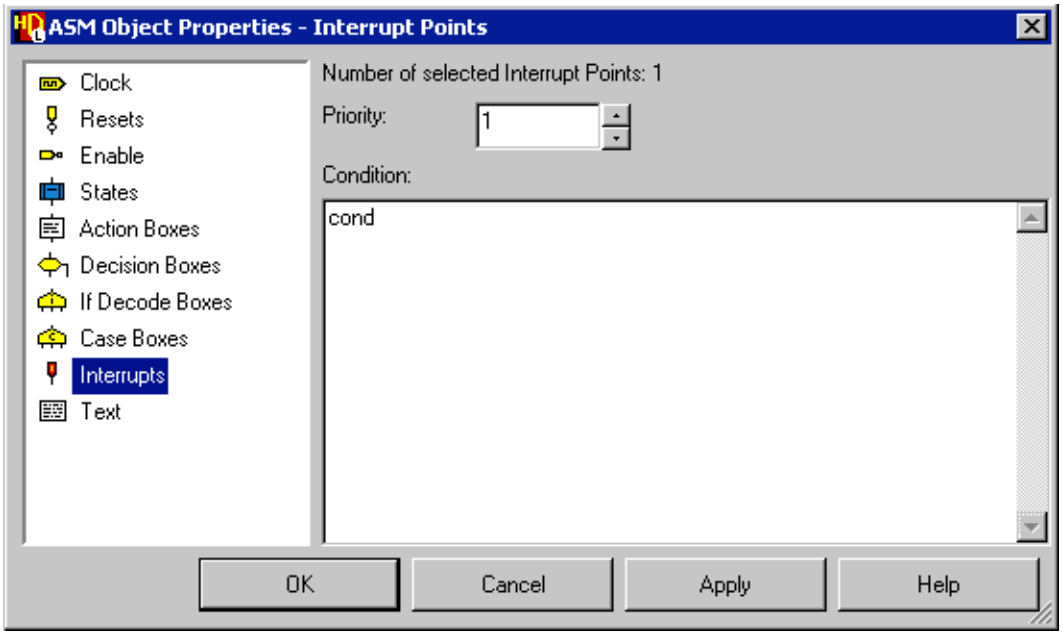
You can choose to name the variable automatically or specify a variable name which must be a VHDL identifier. If Automatic naming is set, the name is generated using the template: *hds_selN* where *N* is an integer.

A list of standard VHDL types is available in a pulldown list. The variable type should be *std_ulogic_vector* if all the inputs are *std_ulogic* or *std_ulogic_vector*. It should be *std_logic_vector* if all the inputs are scalar and of type *std_logic*.

Otherwise, it should be the same type as the input arrays. The bounds must be sufficient for the size of the concatenated input expressions. Note that when scalar values are concatenated with other scalar values or with array values, the result is always an array value.

Editing Interrupt Object Properties

The **Interrupts** page of the ASM Object Properties dialog box allows you to specify the properties for an interrupt point.

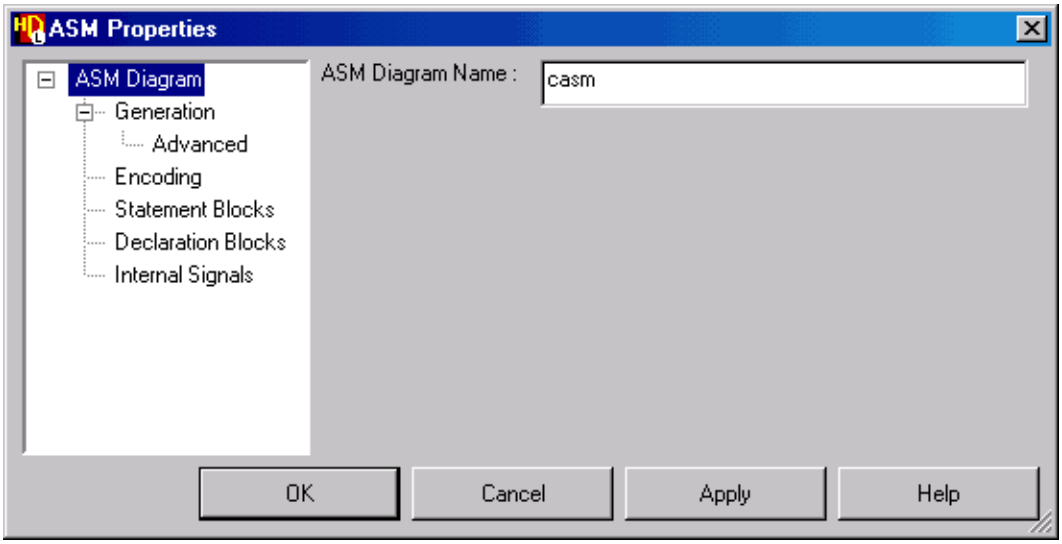


You can change the priority if there are multiple interrupt points on the diagram and specify the interrupt condition which must be unique.

Interrupt masking can be achieved by an AND expression combining the condition with a mask signal. For example, *intrpt = '0' AND msk = '1'*. This mask signal may control a single interrupt or act as an enable mask for a number of interrupts.

Setting ASM Chart Properties

You can set the properties for an ASM chart by choosing **ASM Properties** from the **Diagram** or popup menu to display the ASM Properties dialog box.



The main page of the ASM Properties dialog box allows you to edit the diagram name of the active concurrent state machine.

You can select additional pages and sub-pages from the left pane in the dialog box:

- The Generation page allows you to set basic properties for HDL generation. A separate sub-page can be used to set Advanced generation properties.
- The Encoding page can be used to specify state machine encoding.
- The Statement Blocks page allows you to specify concurrent statements, state register statements and global actions.
- The Declaration Blocks page allows you to specify architecture, module or process declarations.
- The Internal Signals page allows you to set the prefix or suffix used for internal registered or clocked signals.

The statements, declaration, global actions and signal status are displayed as text objects on the diagram and the dialog can be opened directly by double-clicking over one of these objects.

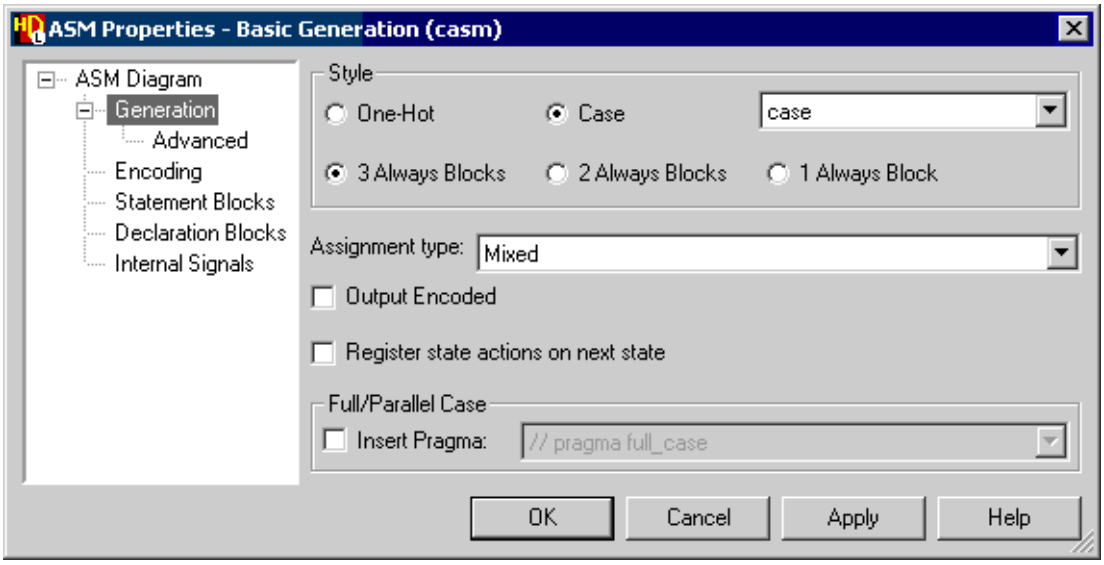
The generation properties and state machine encoding information are not displayed on the ASM chart.

Setting ASM Chart Generation Properties

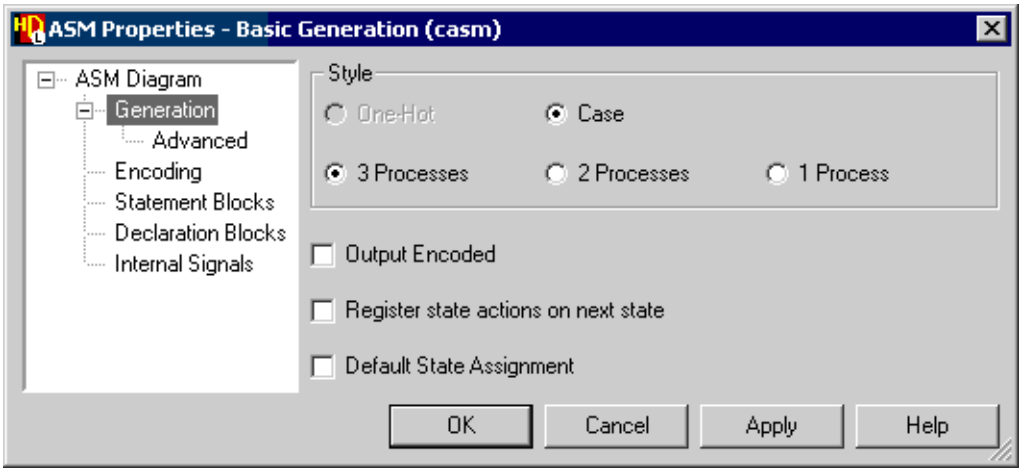
You can set generation properties in the **Generation** and **Advanced** pages of the ASM Properties dialog box.

The **Generation** page allows you to set the HDL style, specify the number of Verilog *always* blocks or VHDL processes and set the state encoding type to output encoded.

When you are using Verilog, this page also allows you to set the assignment type, choose to register state actions on the next state and set full/parallel case pragmas:



When you are using VHDL, you can choose to register state actions on the next state and set the default state assignment.



Note

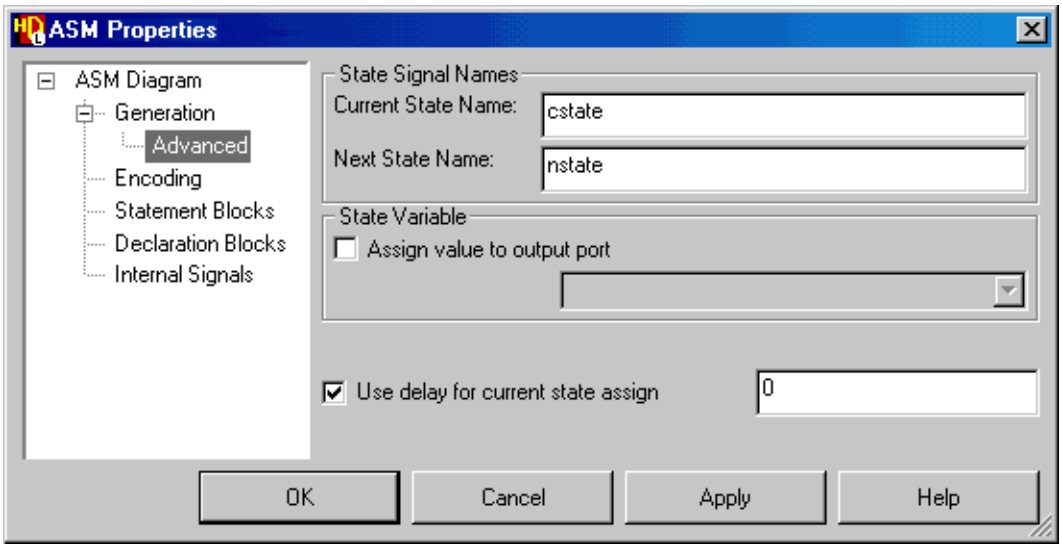
Default state assignment is only available for the Case HDL style.

Advanced Generation Properties

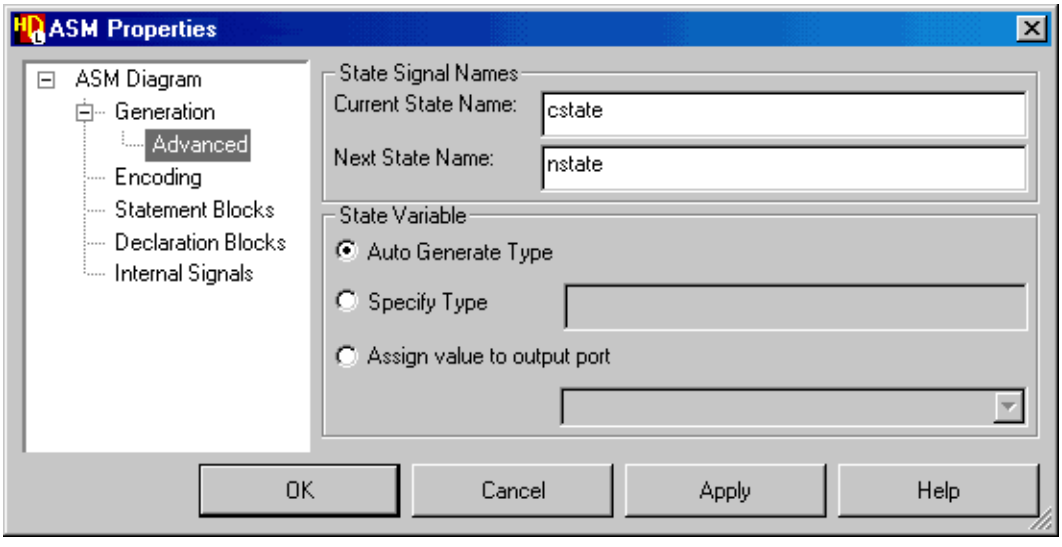
The **Advanced** page allows you to set additional generation properties.

These include the names used for the current state and next state variables. You can also assign the state variable to the value of an output port which can be selected from a dropdown list of the available ports.

When you are using Verilog, you can specify a delay for the current state assignment.



When you are using VHDL, you can choose to automatically generate a *type* or you can specify a discrete type for the state variable:



Refer to Appendix A for more information about the effect of setting HDL Generation Properties on HDL generation.

Note

If you choose to assign the state variable to a specified output port value (in the **Advanced** page of the ASM Object Properties dialog box), and at the same time set the state encoding as Auto (in the **Encoding** page), the following occurs on VHDL generation:

- The state encoding values will be represented in the generated code as constant declarations, and the type of state variable(s) (the current and next state signals) will be the same as the type of the output port (for example “std_logic_vector”).

  This is because eventually the state signal will be assigned to the value of the output port, and hence they have to be compatible in type.

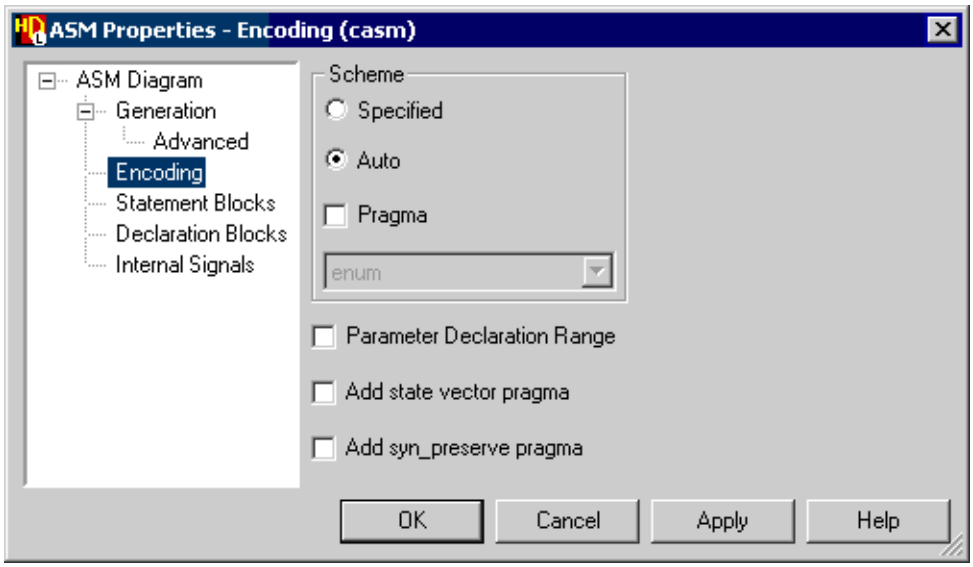
- The state encoding will follow the encoding style you have defined in the **Encoding** page. However, it should be noted that if you have the state encoding set as Auto without specifying an attribute, then the encoding style will be assumed to be “Sequential”.

Setting State Encoding Properties

You can set state machine encoding properties in the **Encoding** page of the ASM Properties dialog box.

The dialog box allows you to explicitly specify the state encoding or use an automatic scheme to select the required VHDL attributes or Verilog pragmas. If you are using Verilog, you can also choose whether to use the range in the declaration of the state encoding parameter. You can also add verilog state vector and *syn_preserve* pragmas or VHDL state vector and *syn_preserve* attributes.

For example, the following dialog box is displayed if you are using Verilog:



When a specified encoding scheme is selected for either language, the encoding values can be entered by direct text editing on the state or by using the *Encoding* field in the **States** page of the ASM Object Properties dialog box as described in "Editing State Object Properties" on page 104.

Refer to "State Encoding" on page 146 for information about the HDL generated and the use of VHDL attributes or Verilog pragmas and encoding algorithms in automatic or specified encoding schemes.

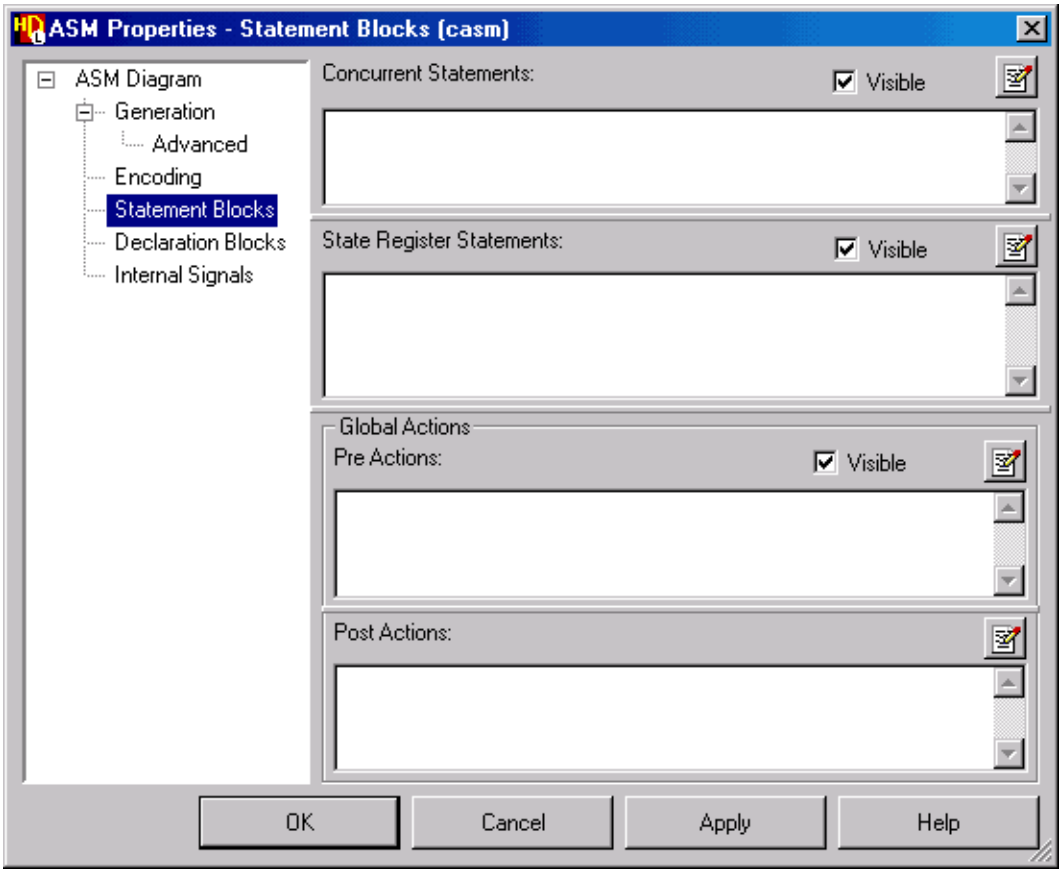
Setting Statement Blocks

You can set *concurrent statements*, *state register statements* and *global actions* in the **Statement Blocks** page of the ASM Properties dialog box.

The dialog box provides separate free-format entry boxes for you to add or edit the concurrent statements, state register statements and for pre and post global actions. The edited statements are added to the diagram when you confirm the dialog box.

The syntax is automatically checked on entry for the hardware description language (Verilog or VHDL) of the active diagram. However, syntax checking can be disabled by unsetting an ASM chart preference.

You can also choose whether the statements are visible or hidden on the diagram (or on the top level ASM chart when you are editing a hierarchical state machine).



Concurrent statements are included in the generated HDL at the end of the VHDL architecture or Verilog module and are applied to all diagrams in a set of concurrent state machines. These statements are executed concurrently with all of the processes (or *always* blocks) in the state machine and are typically used for datapath operations, special clocking or assigning individual elements of the state vector to an output.

Note

If you use an output signal in a concurrent statement, you may need to remove the default value assigned to the signal in the *signals status* table.

State register statements are included in the generated HDL before the state decoding statements in the clock process (or *always* code). These statements are inserted instead of the default assignment to the next state and are typically used to determine when a counter should be incremented or reset and whether to update the state.

Global actions can be used be used to assign complex actions that are executed on every clock or state change. Note however, that you should use the *signals status* table to assign default values to signals.

Pre Actions are included in the generated HDL just after any default values specified at the beginning of the output process (combinatorial signals). *Post Actions* are included after the clock condition in the clocked process (registered signals) and are executed before any conditional actions.

State register statements and global actions can be specified separately for each diagram in a set of concurrent state machines.

Note

If your default text editor is set to DesignPad, a button is available which allows you to edit the statement block in DesignPad. You can also edit the statements directly on the diagram by clicking to select the text and clicking again to edit the text.

Setting Declaration Blocks

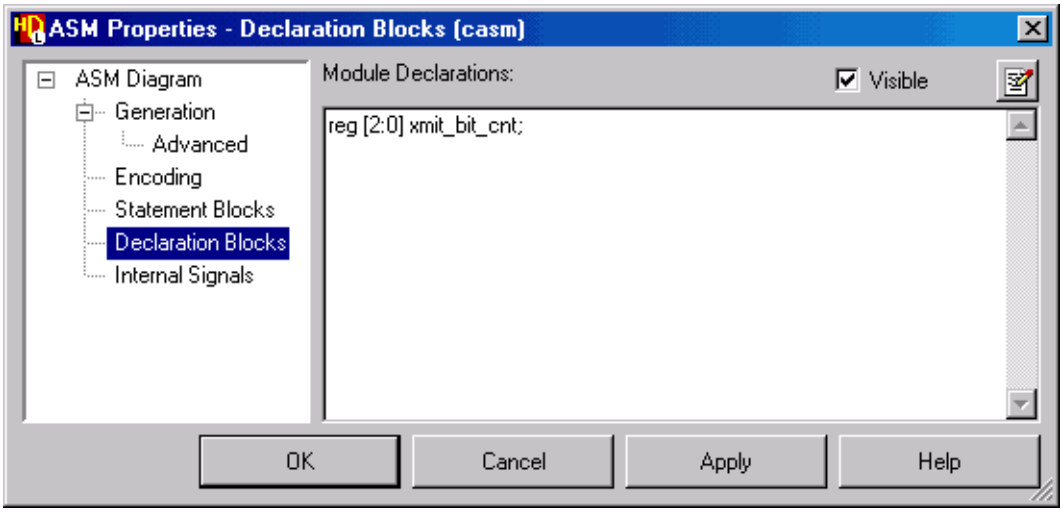
You can set *architecture declarations* (VHDL only) or *module declarations* (Verilog only) and *process declarations* (VHDL only) in the **Declaration Blocks** page of the ASM Properties dialog box.

The dialog box allows you to enter any valid HDL statements for the current hardware description language in a free-format entry box. Signals, constants or variables can be declared and comments, procedures, functions or type definitions can also be included.

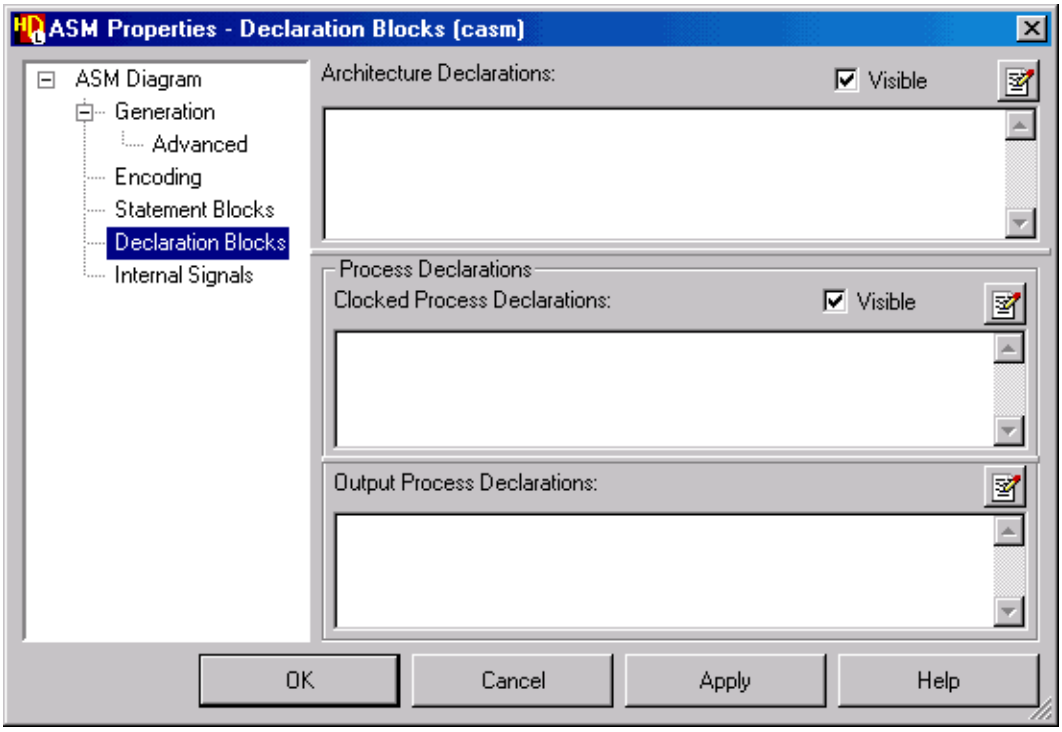
The syntax is automatically checked on entry for the hardware description language (Verilog or VHDL) of the active diagram. However, syntax checking can be disabled by unsetting an ASM chart preference.

Refer to "Declaration Syntax" on page 24 for information about the syntax used for declarations.

If you are using Verilog, a single entry box is provided for editing module declarations:



If you are using VHDL, the dialog box provides separate free-format entry boxes for you to add or edit architecture declarations, clocked process declarations and output process declarations.



You can also choose whether the declarations are visible or hidden on the diagram (or on the top level ASM chart when you are editing a hierarchical state machine).

The architecture declarations or module declarations are inserted at the top of the VHDL architecture or Verilog module in the generated HDL and apply to all diagrams in a set of concurrent state machines.

Architecture declarations or module declarations cannot be set when an ASM chart is used to define an embedded view in a block diagram or IBD view. However, any declarations required by the embedded view can be set on the parent view.

The clocked and output process declarations are placed immediately before the BEGIN statement in the generated VHDL for the clocked and output processes.

Separate process declarations can be specified for each diagram in a set of concurrent state machines.

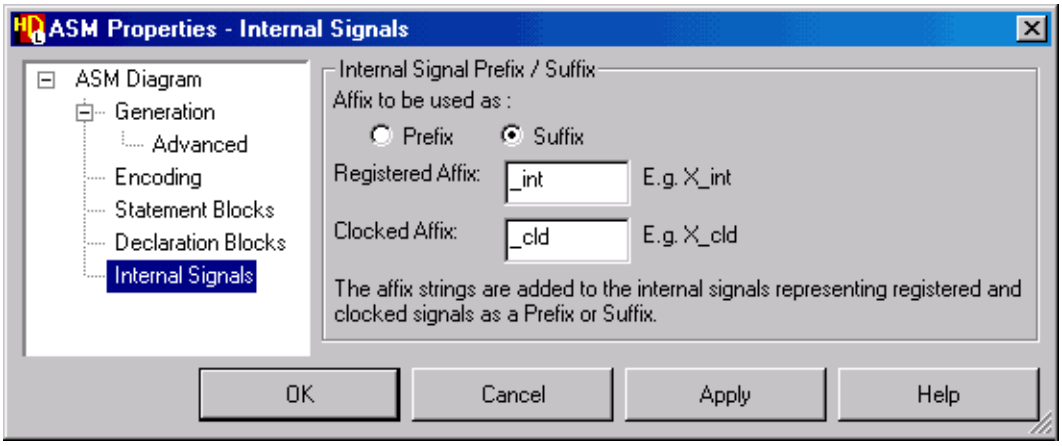
Note

If your default text editor is set to DesignPad, a button is available which allows you to edit the declaration block in DesignPad. You can also edit the declarations directly on the diagram by clicking to select the text and clicking again to edit the text.

Setting ASM Chart Internal Signal Names

You can set the status for output or locally declared signals in an ASM chart using the "Signals Table" which is described in Chapter 4.

You can set the prefixes and suffixes used to identify registered or clocked internal signals in the **Internal Signals** page of the ASM Properties dialog box.



You can set preferences for the internal registered and clocked signal names in the **Miscellaneous** tab of the ASM Preferences dialog box.

Running Design Rule Checks

You can run design rule checks on the active ASM chart by using the button in the HDL Tools toolbar.

The following checks are performed:

- All decision or case branches under hierarchical actions must be connected
- The true branch of all decision boxes must be connected. (The false branch can be unconnected which means it is an implicit loopback.)
- All states must have input and output flows. (This means that you cannot have only links pointing to a state with the input flow unconnected.)
- A reset flow must end on a state, link or hierarchical state.
- All synchronous or asynchronous reset conditions must be unique. (However, it is possible to have the same condition on resets of different modes.)
- An if decode box with "If Then" style cannot have an "else" port. (Since this is the parallel if style and there should be no elsif or else statements.)

Any error messages are reported in the Log window.

Setting ASM Chart Preferences

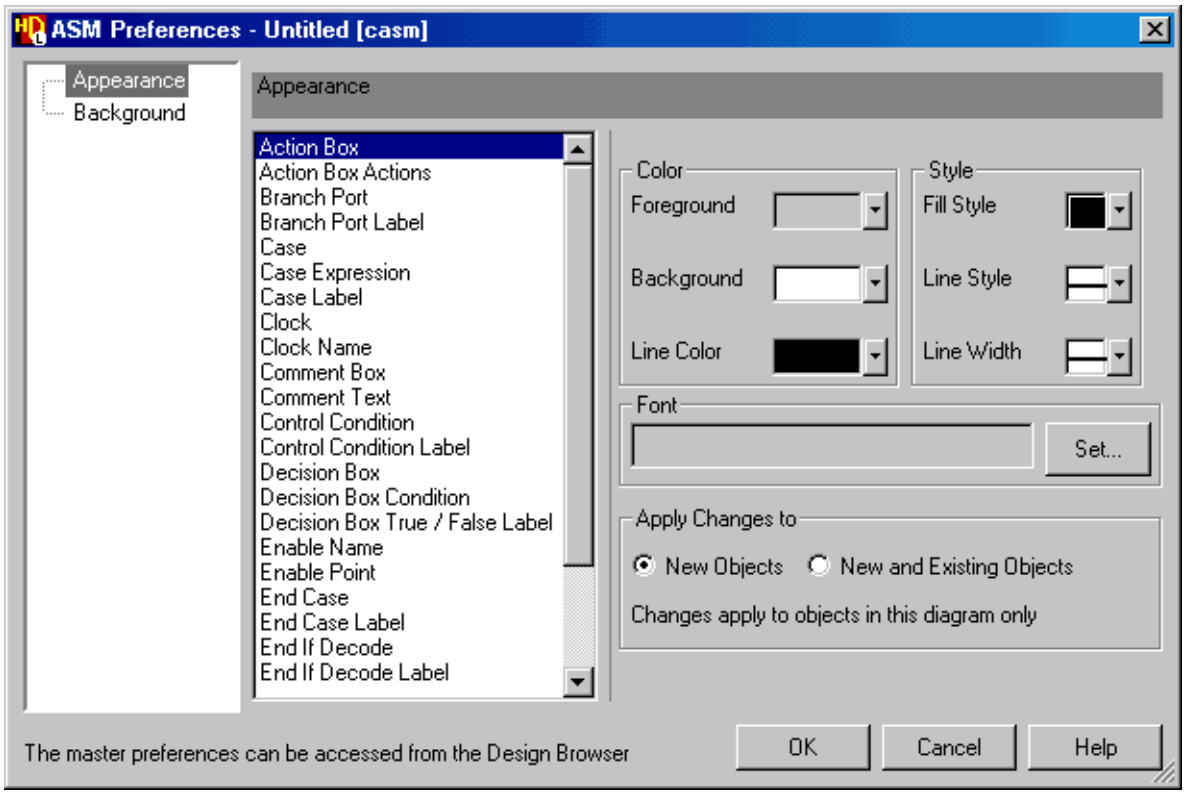
You can set ASM chart preferences by choosing **ASM** from the **Master Preferences** cascade of the **Options** menu in the *design manager*.

The ASM Preferences dialog box has separate pages for **Appearance**, **Default Settings**, **Object Visibility**, **Background** and **General** preferences.

Note

The general preferences, default settings, and object visibility preferences take effect on the next ASM chart that you open and can only be edited when the dialog box is displayed from the **Master Preferences** cascade in the design manager **Options** menu. These pages are not available when you choose **Diagram Preferences** in a graphic editor window.

The **Appearance** page of the ASM Preferences dialog box, which can also be edited by choosing **Diagram Preferences** from the **Options** menu in a state diagram, allows you to set default visual attributes for individual ASM chart objects:



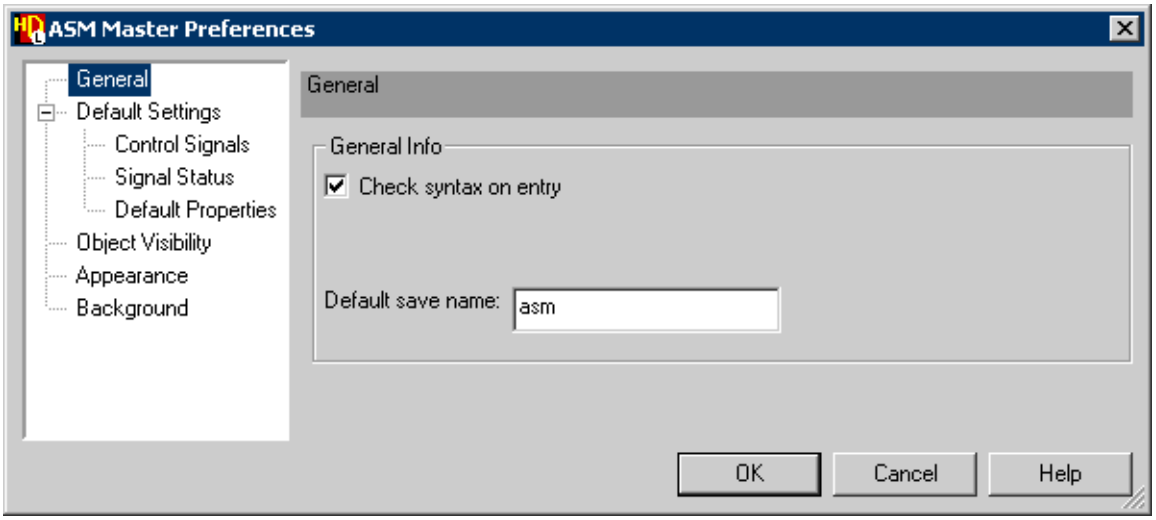
The attributes include the foreground and background colors, line color and style, fill style, line width and the text font. Some attributes may not always be available. For example, line style, width and color attributes are not available for a text object.

Refer to “Setting Visual Attributes” in the *Graphical Editors User Manual* for more information.

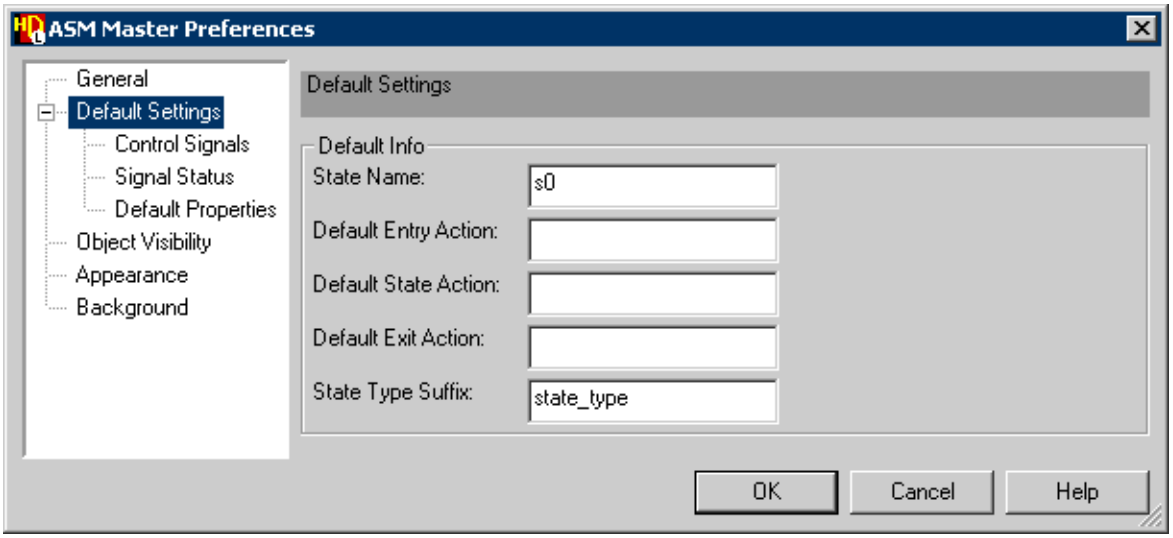
When you edit preferences for the active diagram, the dialog box allows you to choose whether the preferences are applied to new objects or to both new and existing objects in the state machine (including concurrent or hierarchical diagrams).

You can save the appearance preferences set on the active diagram as master preferences by choosing **Update from Diagram** in the **Master Preferences** cascade of the **Options** menu or you can apply the master preferences to the active diagram by choosing **Apply to New Objects** or **Apply to New and Existing Objects**.

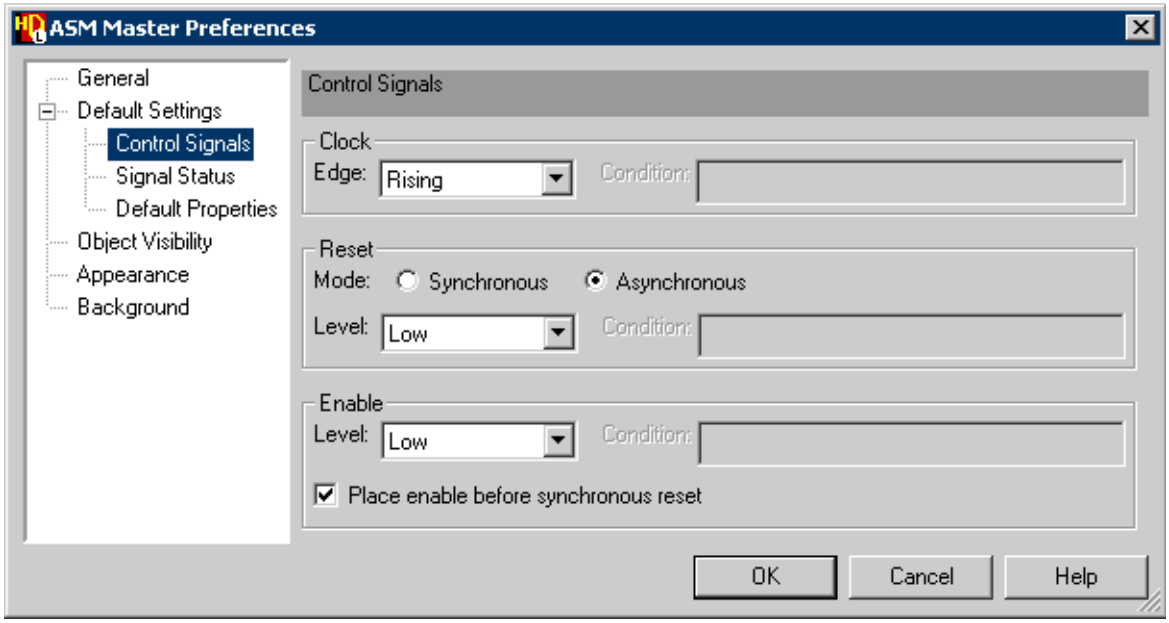
The **General** page allows you to set miscellaneous ASM chart options including whether syntax checking is enabled on entry and the default save name for an algorithmic state machine:



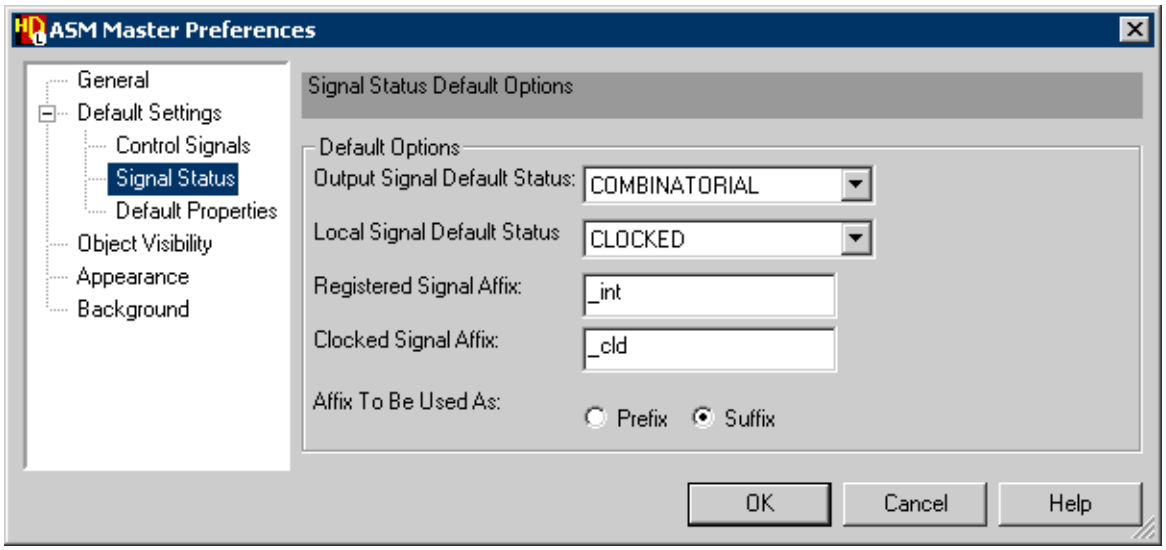
The **Default Settings** page allows you to set ASM chart default values including the default name for a state box and the entry, state and exit actions for a state:



The **Control Signals** sub-page enables you to set the clock edge and clock condition, the reset mode, reset level, and reset condition, as well as the enable level and enable condition.

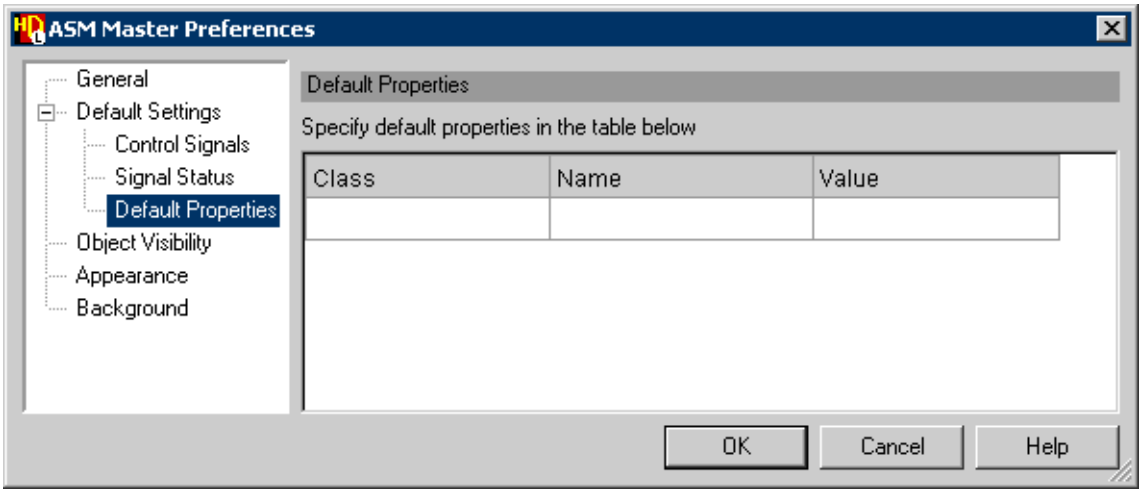


A separate **Signal Status** sub-page allows you to set default options for signal status:



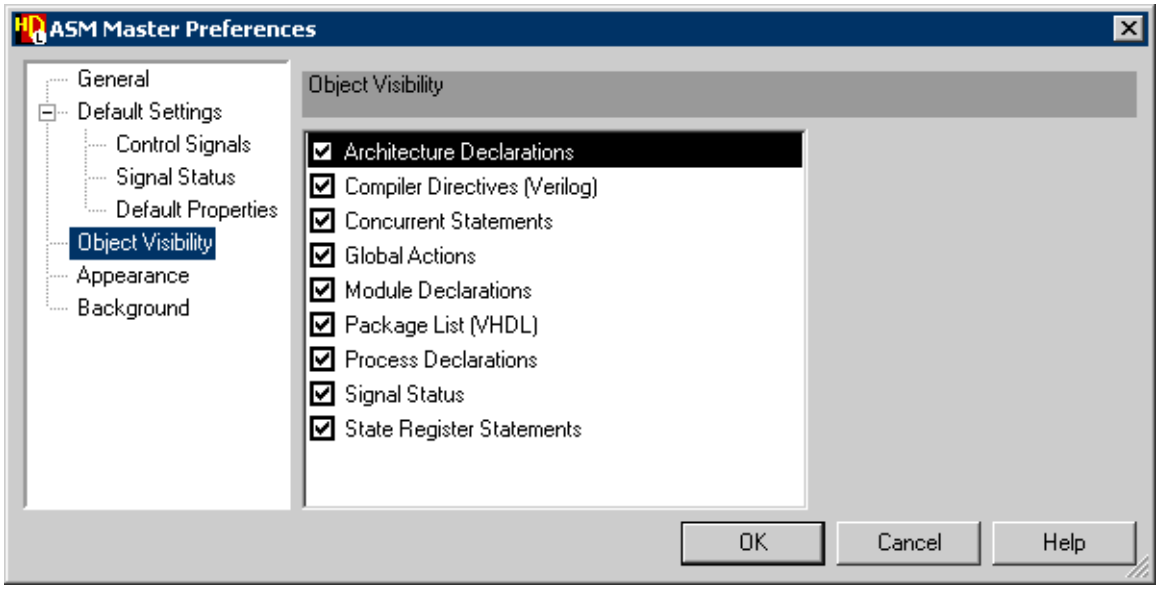
You can also specify the default suffix or prefix used for registered and clocked signal names.

You can use the **Default Properties** sub-page to define default user properties for ASM chart views:



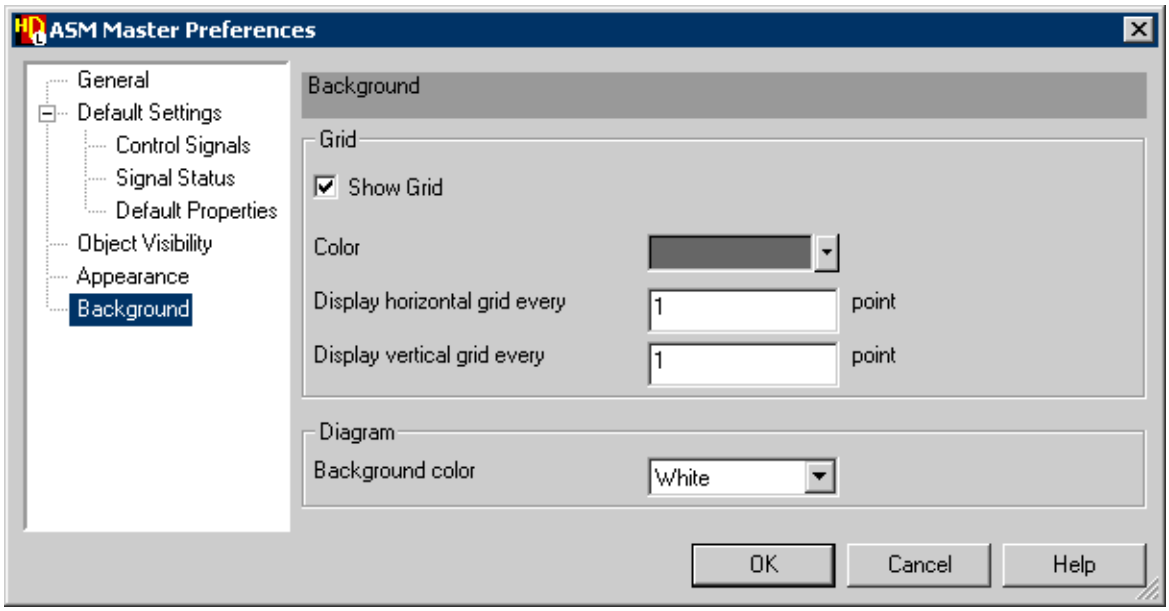
Refer to the *HDL Designer Series User Manual* for information about “Using View Property Variables”.

The **Object Visibility** sub-page allows you to set the default object visibility for multi-line text objects on the ASM chart.



Refer to “Changing Text Visibility” in the *Graphical Editors User Manual* for more information.

The **Background** page allows you to set grid attributes and the background color used in the ASM chart editor.



Information about “Setting Background Preferences” is given in the *Graphical Editors User Manual*.

Chapter 4
Signals Table

This chapter describes the signals table which can be used to view and edit the status of signals used in the state diagram or ASM chart editor.

Displaying the Signals Table

The signal declarations and status associated with each concurrent state diagram or ASM chart can be displayed by double-clicking on the Signals Status on the diagram or by selecting the **Signals** page in the *diagram browser*. Refer to “The Diagram Browser” in the *Graphical Editors User Manual* for information about browsing diagram structure and content.

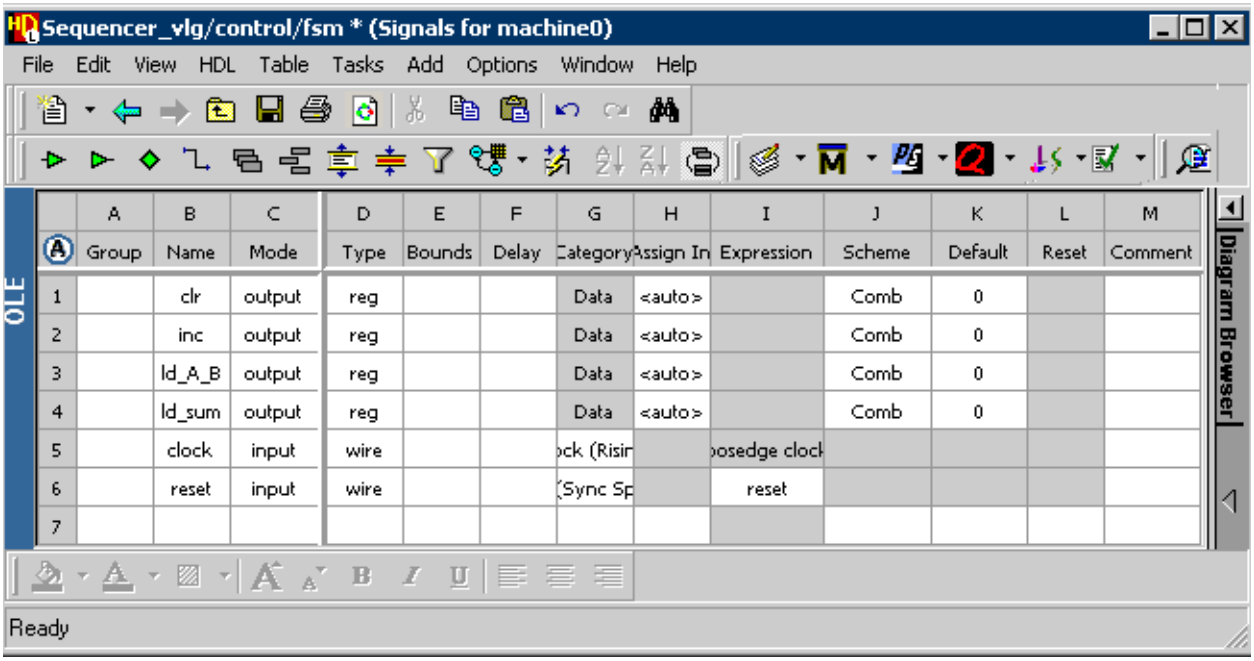
The table is synchronized to show only the signals for the active concurrent view including any ports which have been added to the symbol (if it exists). Any redundant ports are removed and the type, bounds and other properties updated from the symbol.

If you have edited the signals table, the state diagram or ASM chart is synchronized with the signal status. For example, if you clear the enable category for a port, the graphical enable object is removed when you re-display the diagram.

You can display the signals table for any other concurrent state machine by choosing from a drop down list of concurrent state machine names using the  button or in the **Table for Machine** cascade of the **View** menu.

The table has a separate row for each signal defined in the interface plus additional rows for any locally defined signals.

The following example shows the signals table for a concurrent FSM diagram view of the *control* design unit in the *Sequencer_vlg* example design.



	A	B	C	D	E	F	G	H	I	J	K	L	M
	Group	Name	Mode	Type	Bounds	Delay	Category	Assign In	Expression	Scheme	Default	Reset	Comment
1		clr	output	reg			Data	<auto>		Comb	0		
2		inc	output	reg			Data	<auto>		Comb	0		
3		ld_A_B	output	reg			Data	<auto>		Comb	0		
4		ld_sum	output	reg			Data	<auto>		Comb	0		
5		clock	input	wire			ck (Risir		posedge clock				
6		reset	input	wire			(Sync Sp		reset				
7													

Signals Table Notation

Vertical and horizontal scroll bars are available if the signals table does not fit in the current window. However, the header row and the Group, Name and Mode columns are non-scrolling and are always shown.

Refer to "Grouping Signal Rows" on page 134 for information about using the Group column.

Read-only cells displaying information have a grey background. Editable cells have a white background.

Note

Note that you can select an entire row by clicking the row number or an entire column by clicking the column letter. You can also select the entire table by clicking on cell A1. You can resize any cell by dragging the sashes between the columns.

Signal Declaration Columns

The signal declarations for interface ports are displayed in the Name, Mode, Type, Bounds, Delay or Initial and Comment columns.

New signal declarations can be added using an empty row at the bottom of the table.

Name	Port or locally declared signal name.
Mode	Signal mode: input, output, bi-directional, buffer (VHDL only) or local.
Type	VHDL type definition or Verilog net type.
Bounds	Range of the specified type (may use short or long format for VHDL).
Delay	Delay value for a Verilog signal.
Initial	Initial value of a VHDL signal.
Comment	Comment appended to a signal declaration.

Refer to the “Component Interface Views” chapter in the *Graphical Editors User Manual* for more information about port signal declarations.

Signal Status Columns

Signal status is shown using the Category, Assign In, Expression, Scheme, Default and Reset columns.

Category	Input and local signals can be used to specify the clock and enable or any number of resets. Output, bidirectional and buffer ports are read-only and are always set to *Data*.
Assign In	The concurrent state machine in which an output signal is assigned. Defaults to <*auto*>. (Hidden if there is only one concurrent machine.)
Expression	The expression for a clock, enable or reset; or state variable.
Scheme	The clocking scheme used for output and locally declared signals. Output signals can be *Registered*, *Combinatorial* or *Clocked*. Local signals can be *Combinatorial* or *Clocked*. Clocking scheme editing is disabled in the following cases: • Input ports. • INOUT/BUFFER control signal (clk/rst/enable) must always be COMB and nothing else is allowed • Signals with a <none> in the AssignIn cell • Signals assigned in a 1-process concurrent state machine (csm) are always clocked • Output encoded machine with a port of mode out, inout or buffer
Default	The default values for output and local signals. Combinatorial signals used in actions must have default values. However, signals used in concurrent statements do not require default values. If not specified, the value defaults to a value appropriate to the width of the signal (for example, all 0, all 1 or all X).

Reset	The value assigned to a flip-flop on primary reset. A reset value is required for all registered or clocked signals but is not available for combinatorial signals. If not specified, the value defaults to a value appropriate to the width of the signal (for example, all 0, all 1 or all X).

Refer to "Signals Status" on page 150 for information about the effects of registered, combinatorial and clocked signal clocking schemes.

Note

On setting the VHDL generation style as "3 Processes" in the State Machine Properties dialog box, you get three processes describing the next state, output and clocked logic. It is important to note that if the output process to be generated has no value other than the Default value assignment, the output process is not generated. As code optimization, the default assignment is rather generated as a concurrent assignment.

However, this does not mean that concurrent assignments can be directly placed as default assignments; this is not a recommended practice as it may result in syntax errors. In case you need to define concurrent assignments, it is recommended to explicitly add them in the Statement Blocks page of the State Machine Properties dialog box.

Signals Table Toolbars

The following commands are available from the SM Signals Tools or ASM Signals Tools toolbar in the signals table:

Table 4-1. Signals Table Toolbar

Button	Description
	Add an input port
	Add an output port
	Add a bidirectional (inout) port
	Add a buffer port (VHDL only)
	Add a local signal
	Group the selected rows or add a group (in hierarchical mode)
	Ungroup
	Expand all groups
	Collapse all groups
	Toggle Filter
	View the signals table for a specified machine
	Fit the cell width to the contents of the selected cell

Table 4-1. Signals Table Toolbar

Button	Description
	Sort in ascending order
	Sort in descending order
	Toggle between grouped and ungrouped mode

The Standard, HDL Tools and Tasks toolbars are also available in the signals table window. Refer to "Toolbars" in the *Graphical Editors User Manual* for information about the Standard graphical editors toolbar. Refer to "Toolbars" in the *HDL Designer Series User Manual* for information about the HDL Tools and Tasks toolbars.

Adding Port or Local Signal Declarations

You can add ports to a component interface using the **Add** menu or the following buttons in the tabular IO view:

Table 4-2. Adding Port or Local Signal Declarations

Button	Function Key	Description
	**F8**	Add an input port
	**F9**	Add an output port
	**F11**	Add a bidirectional (inout) port
	**F12**	Add a buffer port (VHDL only)

The port is added in the next available row with default name, type and bounds.

Alternatively, you can add ports by entering a declaration directly into the next row of Name, Mode, Type and Bounds cells. The mode defaults to the last mode used or you can choose from a list of available modes: input, output, bidirectional (inout) or buffer (VHDL only).

If you do not change the name of a port, each new port name is made unique by adding an integer to the default name. (For example: *In*, *In1*, *In2*…).

If you add a port whose name suggests it might be a clock, reset, or enable, the port type, mode and category columns are populated with the signal default values. Refer to "Adding Objects on a State Diagram" on page 33.

You can add a declaration for a local signal by using the  button or choosing **Local Signal** from the **Add** menu. A new declaration is added at the bottom of the table with the default name *Local* or *LocalN* (where *N* is automatically incremented if it exists).

Note

Local signals cannot be declared when a state machine is used to define an embedded view in a block diagram or IBD view. However, any local signals required by the embedded view can be declared on the parent view.

You can also add a local signal declaration by entering the new signal name directly in the Name column of the empty row at the bottom of the signals table.

The port or local signal type defaults to the last type used or you can choose from a dropdown list of available types in the Type column. The bounds defaults to the last range used or you can choose from a dropdown list of recently entered ranges in the Bounds column.

A VHDL bounds can be entered in long or short format. The display format can be set by setting or unsetting the **Short Form** option in the **Table** menu.

If you enter a port or signal name followed by a valid bounds constraint, for example, *myport(7 DOWNTO 0)*, the constraint is automatically moved to the Bounds column.

Tip: You can automatically complete a row with default properties by using the  key after entering a port name to move to the name cell in the next row.

You can optionally enter a value in the Initial (VHDL) or Delay (Verilog) and Comment columns. The delay or initial value can be chosen from a dropdown list of recently entered values.

If you enter characters that match characters in an existing entry of the same column, the remaining characters are entered automatically.

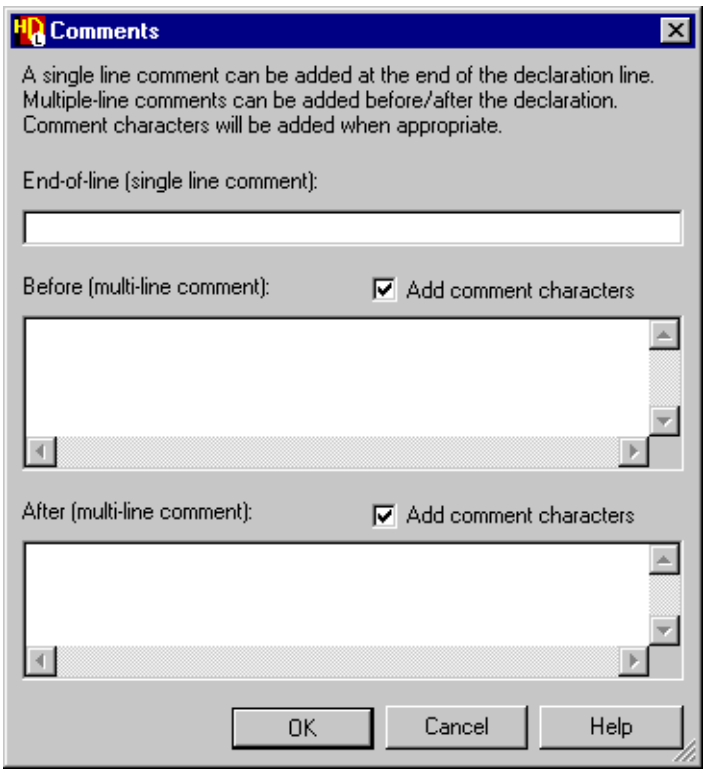
If you have changed port declarations to the signal table, the interface is automatically synchronized when you save the state machine view and any new ports added to the parent view.

Local signals defined in the Architecture/Module declarations in the state machine are shown as read-only rows in the signals table. Refer to "Setting Declaration Blocks" on page 75.

Adding Comments to a Port or Local Signal Declaration

You can add comments to a port or local signal declaration by choosing **Comments** from the popup menu when the declaration row is selected.

A free-format entry Comments dialog box is displayed which allows you to add a single line comment at the end of the declaration or you can enter a multi-line comment to be included before or after the declaration.



Comment characters for the current hardware description language (VHDL or Verilog) are automatically inserted if the **Add comment characters** check box is set.

When this option is unset, the comments must be valid HDL statements and are automatically syntax checked if checking is enabled.

If a declaration is deleted, the corresponding comments are also deleted.

Although multi-line comments can be added using the dialog box, these comments are not displayed in the table. However, end-of-line comments can be edited directly in the Comment column for the local signal declaration row.

Resizing Columns

You can automatically fit a column (or columns) to the width of the text contained in the selected cell (or cells) by using the  button or by choosing **Autofit** from the **Table** or popup menu. If no cells are selected, then all columns in the table are re-sized.

Hiding Columns

You can hide and show columns in the signals table by choosing **Hide Column** from the popup menu or the **Columns** cascade of the **Table** or menu.

If one or more columns are hidden, you can display a dialog box listing the hidden columns by choosing **Show Columns** from the popup or **Table** menu.

Refer to “Hiding Columns” in the *Graphical Editors User Manual* for more information.

Filtering Columns

You can filter the content of columns in the signals table by using the  button or setting the **Filter** option in the **Table** menu. When this option is set, an additional row is added in the non-scrolling area and a dropdown filter menu is available in each column.

You can apply filters to more than one column or set options to match case, match whole words or use regular expressions by choosing **Filter Settings** from the **Table** or popup menu to display the Filter Controls dialog box.

Refer to “Filtering Columns” in the *Graphical Editors User Manual* for more information.

Grouping Signal Rows

You can group rows in the signals table by selecting a row or rows and using the  button or choosing **Group** from the **Add** menu.

The selected rows are added to a new group with the default name *SmGroupN* or *AsmGroupN* (where *N* is automatically incremented if it already exists).

You can also add a group or create a new group by entering a name in the Group column for the ports you want to group.

Note

You can choose from a dropdown list of existing groups. If you type a partial string that matches the name of an existing group the name is automatically completed.

You can remove a group name by selecting a row (or rows) and using the  button or by deleting the name from the Group cell.

If you rename or remove an existing group cell and the group is no longer referenced, you are prompted to delete the old group name.

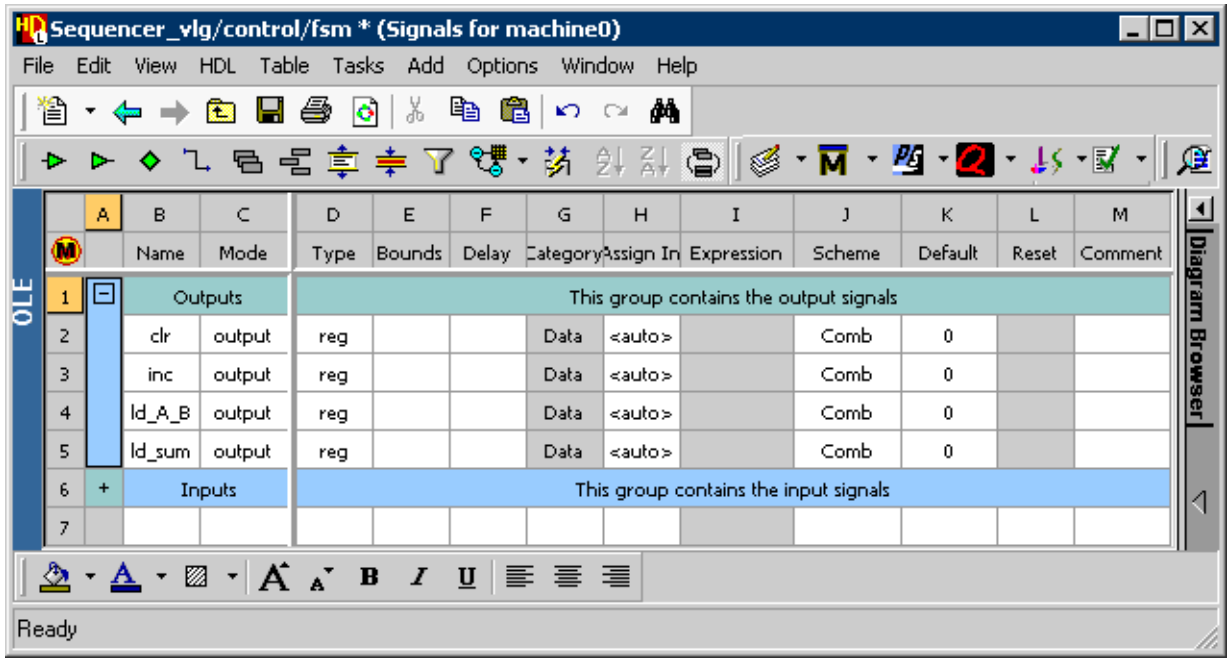
When the signals table includes one or more groups, you can use the  button or set **Show Grouped** in the **Table** menu to toggle between grouped and ungrouped mode.

All rows are displayed normally in flat mode but rows in the same group are shown as a single (but expandable) group in hierarchy mode.

You can expand all the group rows by using the button or choosing **Expand All Groups** from the **Ports** cascade in the **Table** menu. Alternatively, you can expand an individual group by clicking on the icon.

You can collapse all the group rows by using the button or choosing **Collapse All Groups** from the **Ports** cascade in the **Table** menu.

When grouped mode is set, you can enter a comment in the group row as shown for the *Inputs and Outputs* groups in the example below. The *Outputs* group is shown expanded but the *Inputs* group is shown collapsed:



	A	B	C	D	E	F	G	H	I	J	K	L	M
		Name	Mode	Type	Bounds	Delay	Category	Assign In	Expression	Scheme	Default	Reset	Comment
1	−	Outputs		This group contains the output signals									
2		clr	output	reg			Data	<auto>		Comb	0		
3		inc	output	reg			Data	<auto>		Comb	0		
4		ld_A_B	output	reg			Data	<auto>		Comb	0		
5		ld_sum	output	reg			Data	<auto>		Comb	0		
6	+	Inputs		This group contains the input signals									
7													

Note

Groups added in the signals table may be discarded and replaced by the groups defined in the tabular IO view when the table is synchronized with an updated symbol view.

If you delete a group which contains local signals only, they are deleted. If you delete a group which includes ports, only the local signals in the group are deleted.

Sorting Signal Rows

You can sort the rows in a selected column of the signals table in ascending alphanumeric order of the cell data by using the button or choosing **Sort Ascending** from the popup menu or the **Columns** cascade of the **Table** menu.

Alternatively, you can sort the data in descending order by using the  button or by choosing **Sort Descending** from the popup menu or the **Columns** cascade of the **Table** menu.

Editing Signal Status Cells

You can enter data directly by clicking in editable signal status cells or by choosing from a dropdown list of available values.

When an input or local signal is selected, the dropdown list for the Category column includes options for each of the supported clock, reset and enable signals. These options include:

Clock	Rising Falling Specify Risinglast (VHDL only) Risingedge (VHDL only) Fallinglast (VHDL only) Fallingedge (VHDL only)
Reset	Async Low Async High Async Specify Sync Low Sync High Sync Specify
Enable	Low High Specify

If you choose one of the *Specify* options, the Expression column can be used to enter an expression which defines the required clock, reset or enable signal.

Note

Note that only one clock and one enable signal can be specified in each concurrent state machine but you can set any number of reset signals.

When an output or local signal is selected, you can choose which concurrent state machine the signal is assigned in.

If the <*auto*> option is set the concurrent state machine is automatically determined when HDL is generated for the diagram.

If the <*none*> option is set, the clocking scheme, default and reset values cannot be specified.

Appendix A
State Machine HDL Generation

This section describes properties that control how HDL is generated from a graphical *state diagram* or *ASM chart*.

HDL Generation Properties

The HDL generation properties for a state diagram can be set using the **Generation**, **Advanced** and **Control** pages of the State Machine Properties dialog box which is described in "Setting State Diagram Generation Properties" on page 68.

The HDL generation properties for an ASM chart can be set using the **Generation** and **Advanced** pages of the ASM Properties dialog box which is described in "Setting ASM Chart Generation Properties" on page 113.

Synchronous and Asynchronous State Machines

In a synchronous state machine, state transitions occur on an active clock edge which must always be specified. An optional reset signal returns the state diagram to the start state and assigns any specified reset values. A separate enable signal can also be specified. A synchronous state machine should normally be synthesizable.

In an asynchronous state machine, transitions are independent of any clock signal and no clock logic is generated. For an asynchronous state machine, a propagation delay must be specified. The propagation delay is required to avoid a race condition which would prevent the simulator from reaching a stable state and should typically correspond to the minimum resolution time for the simulator. For example, the default minimum simulator resolution for Model*Sim* is *1 ns* when using VHDL or 1 time unit when using Verilog.

The HDL Designer Series supports synchronous and asynchronous state diagrams but all ASM charts are synchronous.

HDL Style

VHDL can be generated as three processes describing the next state, output and clocked logic, two processes which combine the next state and clocked logic or as a single process containing all the assignments. Verilog can be generated as three *always* code blocks describing the next state, output and clocked logic, two *always* blocks which combine the next state and clocked logic or as a single *always* block containing all the assignments.

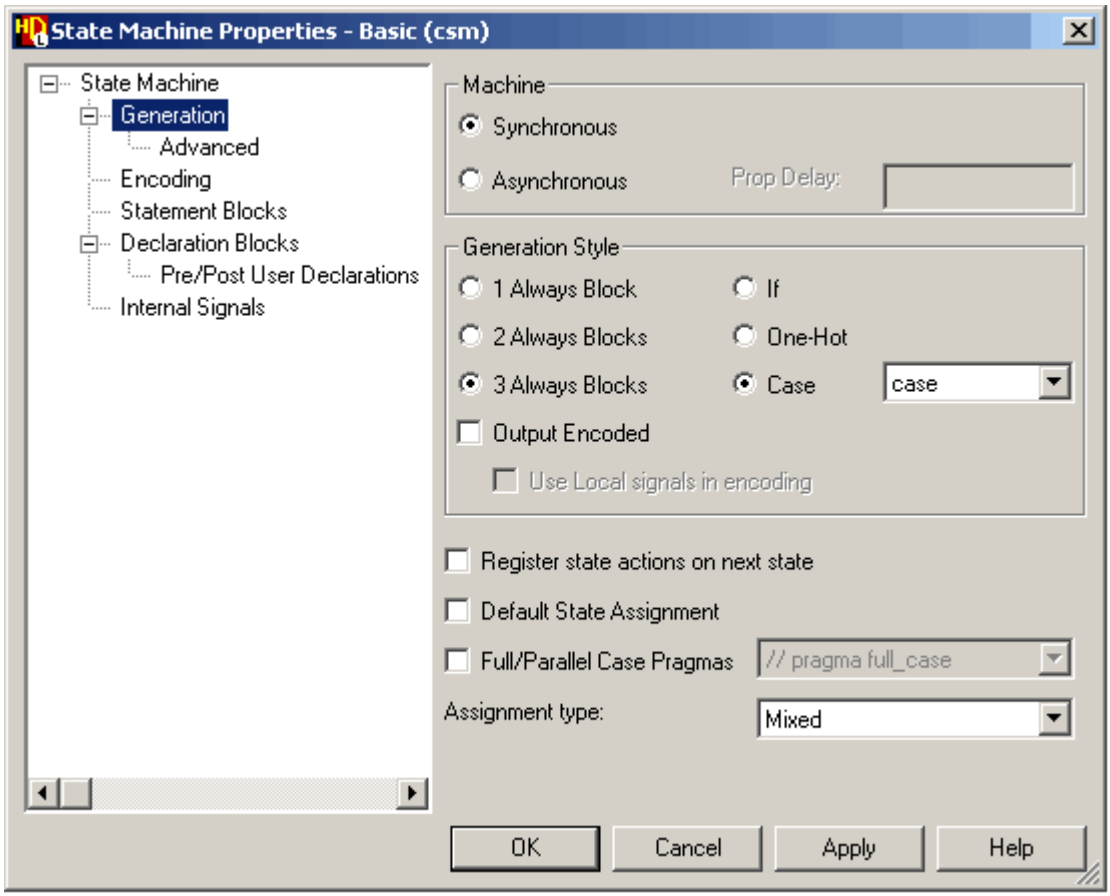
Single process/block generation style

Choosing a single Process/Block generation style will affect the clocking scheme and state variable declarations.

The combination of the next state and output logic will affect the clocking scheme definition as follows:

- The scheme column in the signals table will be hidden as only one clocking scheme is allowed.The Default and Reset columns are enabled for editing. Refer to “Signal Status Columns” on page 129.
- The default assignment type in case of a Verilog design will be Non-blocking if the FSM is synchronous and Blocking if the FSM is asynchronous. Refer to “Setting Declaration Blocks” on page 75.
- The clocked affix will not be assigned to output and internal signals on the state diagram. Refer to “Setting State Diagram Internal Signal Names” on page 78.
- The existence of only one state variable will cancel the option of a default state assignment in VHDL and the next state to the current state assignment in both VHDL

and Verilog. It will also cancel the option of the next_state state variable naming. Refer to page "State Variable" on page 143.



The HDL style chosen determines whether the next state logic is generated using If, One-Hot or Case styles.

When you are using VHDL, the If style uses **IF**...**THEN**....**ELSIF** constructs. The **IF** statement is the logical AND of the state vector value and the transition condition.

For example:

```
IF current_state_name = state0 AND (condition) THEN
   next_state_name <= stateN;
ELSIF...
```

When you are using Verilog, the If style uses **if**...**elsif** constructs. The **if** statement is the logical AND of the state vector value and the transition condition. For example:

```
if current_state_name == state0 && (condition)
   next_state_name <= stateN;
else if...
```

One-Hot style is only available when using VHDL and hard state machine encoding is selected. The One-Hot style defines a constant index for each state and defines a vector type of the same

width as the number of states. The next state logic tests only one bit to determine the current state and sets one bit to change states.

For example:

```
IF current_state_name(state0) = '1' THEN
   IF(condition) THEN
      next_state_name(stateN) <= '1';
   ELSIF...
```

When you are using the One-Hot style in Verilog, the next state logic uses a **case** statement to test the bit which determines the current state and an **if** statement to test the condition, then sets one bit to change states.

For example:

```
case (1'b1)
   current_state_name [state0]:
   if (condition)
      next_state_name[stateN]) = 1'b1;
   else if...
```

The Case style uses a **CASE** statement to decode the state vector. You can use **IF** or **CASE** statements for the transitions.

For example, when using **IF** style transitions in VHDL:

```
CASE current_state_name IS
   WHEN state0 =>
   IF (condition) THEN
      next_state <= stateN;
   ELSIF...
```

Similar code is generated for Verilog, using **case** and **if** statements.

For example, when using **if** style transitions:

```
case current_state_name
   state0:
   if (condition)
      next_state = stateN;
   else if
```

When you are using Verilog in manual state encoding mode, you can also choose to use **casex** or **casez** comparisons as an alternative to bit comparison (**case**).

Output Encoded

The output encoding algorithm uses the outputs and assigned local signals of the state machine to make up the state register. It can be applied to only pure Moore State machines. The algorithm is language independent and can be used with all HDL generation styles.

The following restrictions apply to allow this type of encoding:

- The State machine must be pure Moore style where the outputs/local assignments depend only on the state register and NOT the inputs of the state machine. Therefore outputs/locals must only be assigned in state actions. Mealy transition actions will not be allowed for output assignments. Note that this restriction needs to also apply to local signals which use this encoding scheme.
- Assignment statements used to assign output signals in state actions should be signal assignments in VHDL and blocking assignments in Verilog.
- For Verilog the outputs must only be assigned single bit values (‘0’, ‘1’, or don’t care ‘x’) or strings containing these values.

Tip: Assigned Verilog values should not contain size or base i.e 2’b00 out0 = 2’b00 is not supported.

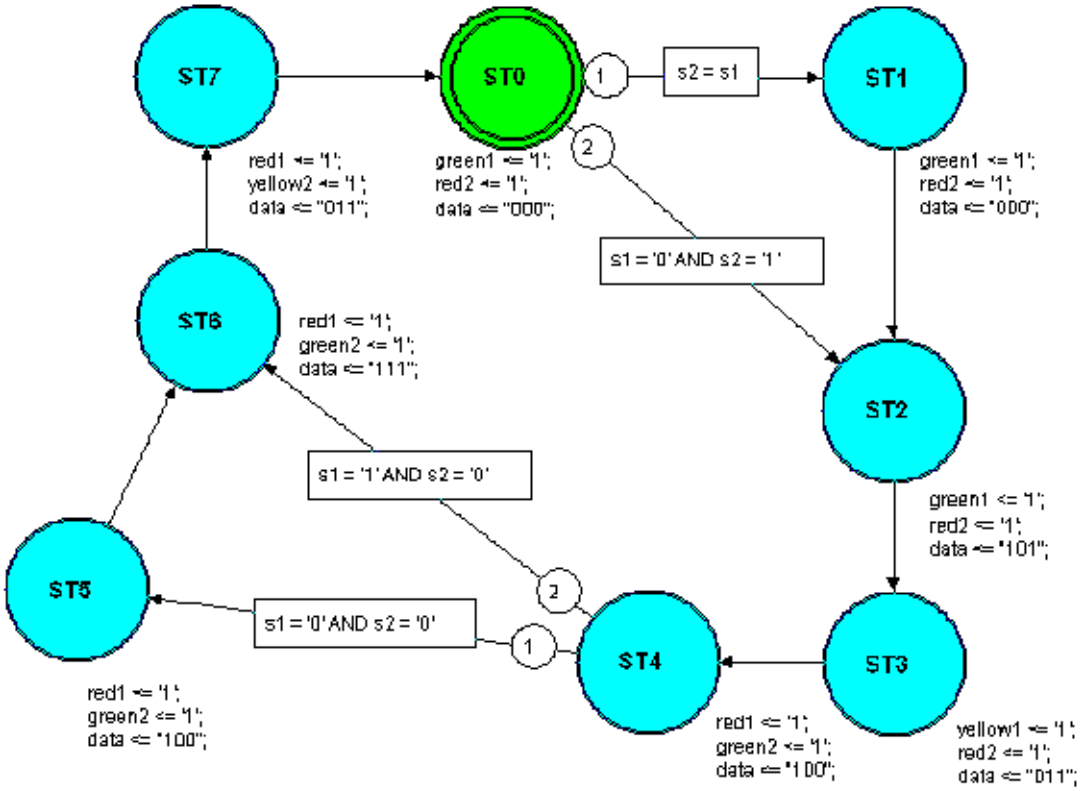
- Output/local signal assignments containing expressions or other signals are not allowed.

  eg out1 <= in1 AND in2
- Signal status clocking scheme is not relevant since the outputs and locals are inherently registered since they are derived directly from the state register. You can use this property to control which Outputs/Locals are directly encoded.
- Global Actions for outputs are not supported.
- All encoded outputs/signals must be of the same type.

The following example shows the state diagram for a simple traffic light controller.

- The default values for all outputs is 0
- S1 and S2 are the road sensors in each direction

- All outputs are scalar except data that is a vector



This is how the states are encoded in VHDL code

```
-- Automatic Output Encoding
   CONSTANT ST0 : STATE_TYPE := "0001100000" ;
   CONSTANT ST1 : STATE_TYPE := "1001100000" ;
   CONSTANT ST2 : STATE_TYPE := "0001100101" ;
   CONSTANT ST3 : STATE_TYPE := "0010100011" ;
   CONSTANT ST4 : STATE_TYPE := "0100001100" ;
   CONSTANT ST5 : STATE_TYPE := "1100001100" ;
   CONSTANT ST6 : STATE_TYPE := "0100001111" ;
   CONSTANT ST7 : STATE_TYPE := "0100010011" ;
```

This is how the states are encoded in Verilog code

```
// State encoding
parameter  // pragma enum current_state_code
          ST0 = 10'b0001100000 ,
          ST1 = 10'b1001100000 ,
          ST2 = 10'b0001100101 ,
          ST3 = 10'b0010100011 ,
          ST4 = 10'b0100001100 ,
          ST5 = 10'b1100001100 ,
          ST6 = 10'b0100001111 ,
          ST7 = 10'b0100010011 ;
```

State Variable

You can choose to assign the state variable which describes the current state of the state machine to the value of an output signal port.

If you are using VHDL, you can choose to automatically generate a *type*. This option generates an enumerated type or if an encoding scheme is set the default type for the currently selected encoding scheme.

If an enumerated type is specified, it must be defined in a *VHDL package* referenced on the diagram. Alternatively, you can specify a discrete type for the state variable.

Hard encoding should be enabled if you want to assign the state variable to an output port or specify a type for the state variable.

Note

If you choose to assign the state variable to a specified output port value (in the **Advanced** page of the SM Object Properties dialog box), and at the same time set the state encoding as Auto (in the **Encoding** page), the following occurs on VHDL generation:

- The state encoding values will be represented in the generated code as constant declarations, and the type of state variable(s) (the current and next state signals) will be the same as the type of the output port (for example “std_logic_vector”).

  This is because eventually the state signal will be assigned to the value of the output port, and hence they have to be compatible in type.

- The state encoding will follow the encoding style you have defined in the **Encoding** page. However, it should be noted that if you have the state encoding set as Auto without specifying an attribute, then the encoding style will be assumed to be “Sequential”.

Generate Interrupts as Overrides

You can choose to generate the HDL for state diagram interrupts as overrides or as exclusive if-then-else statements.

The following example shows the Verilog generated when interrupts are generated as overrides:

```
always @(Rcv_current_state or sample or sin)
begin

   case (Rcv_current_state)
      waiting:
         if (~sin)
            Rcv_next_state <= check_lock;
         else
...
...
```

```
            default: begin
               Rcv_next_state <= waiting;
            end
      endcase

   // Interrupts
   if (IntA) Rcv_next_state <= finish_rcv; -- Interrupt point
```

The following example shows the Verilog generated when interrupts are generated using an explicit if then else statement:

```
   always @(Rcv_current_state or sample or sin)
   begin

      if (IntA) Rcv_next_state <= finish_rcv;
      else
         case (Rcv_current_state)
            waiting:
               if (~sin)
                  Rcv_next_state <= check_lock;
               else
   ...
   ...
            default: begin
               Rcv_next_state <= waiting;
            end
         endcase
      end
```

Register State Actions on Next State

You can choose to register (or clock) state actions on the next state instead of the current state transition. Decoding on the current state adds a one clock cycle delay to the final output which is eliminated if you choose to decode on the next state.

Note

This option is currently only available for a state diagram. It is not supported in the ASM chart properties.

VHDL Default State Assignment

Some synthesis tools require that the next state signal is assigned before the process is entered when you are using VHDL Case HDL style. This can be achieved by setting default state assignment so that a *next_state <= current-state;* statement is inserted at the start of the next state process.

Verilog Assignment Type

You can choose whether blocking assignments (specified by the = operator) or non-blocking assignments (specified by the <= operator) are used in the generated Verilog.

Alternatively, the default option mixes non-blocking assignments in the clocked code with blocking assignments in the next state and output code.

Verilog State Vector Pragmas

You can specify whether pragmas are added around the state vector in a Verilog state machine.

When this option is set, a pragmas are inserted to identify the name and enumeration of the parameter which defines the state variable.

For example:

```
parameter [2:0] // pragma enum current_state_code
          waiting          = 3'd0 ,
          reading_from_reg = 3'd1 ,
          clearing_flags   = 3'd2 ,
          writing_to_reg   = 3'd3 ,
          xmitting         = 3'd4 ;

reg [2:0] /* pragma enum current_state_code */ current_state, next_state ;
// pragma state_vector current_state
```

Verilog Full/Parallel Case Pragmas

Most synthesis tools can detect full and parallel case decoding automatically. However when you are using Verilog, you can choose to add pragmas which explicitly set full, parallel or both full and parallel **case** transitions:

full_case	All possible branches have been specified, any missing branches cannot occur and a default branch need not be generated.
parallel_case	Branches are mutually exclusive.
parallel_case full_case	All possible branches have been specified and are mutually exclusive.

For example a *full_case* pragma can be useful when not all the possible branches have been specified but you know that the unspecified branches cannot occur and you want to prevent the generation of redundant code. Alternatively, a *parallel_case* pragma may be useful to enforce mutually exclusive branches when the synthesis tool is unable to determine this condition.

Either or both pragmas are added at the end of the line containing the **case** statement. For example:

```
case (current_state) //pragma parallel_case full_case
```

State Signal Names

You can specify alternative state signal names for the reserved names *current_state* and *next_state* which are normally used for the state variable in the generated HDL.

Verilog Current State Assignment Delay

If you are using Verilog, you can specify a delay (in time units) between the assignment of the next state (or reset state) to the current state in the clocked *always* block for a synchronous state machine. This option inserts a statement of the following form:

```
if(reset_condition) begin
   current_state_name <= #delay reset_state_name;
   ...
   end
else begin
   current_state_name <= #delay next_state_name;
   ...
```

Note

A state machine which uses this assignment delay is not synthesizable.

State Encoding

The state encoding for a state diagram can be set by selecting the **Encoding** page of the State Machine Properties dialog box as described in “Setting State Encoding Properties” on page 73.

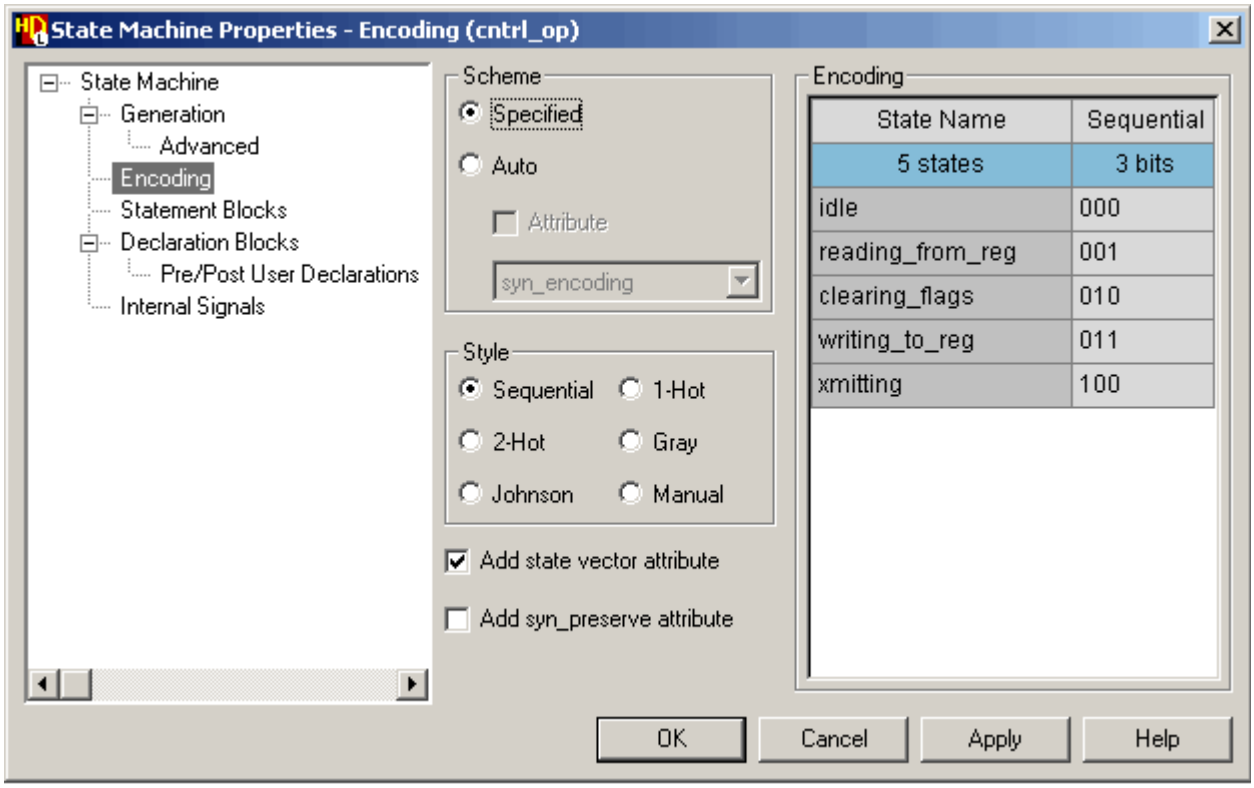
The state encoding for an ASM chart can be set selecting the **Encoding** page of the ASM Properties dialog box as described in “Setting State Encoding Properties” on page 115.

Either of these dialog boxes provide options to set a specified or automatic encoding scheme.

If a specified encoding scheme is selected, the One-Hot HDL style is available when you set generation properties. The One-Hot HDL style can be useful if your synthesis tools do not support automatic state encoding. However, One-Hot HDL style is not recommended for synthesis tools which can perform efficient automatic encoding. Refer to “HDL Style” on page 138 for more information about One-Hot style.

When a specified encoding scheme is selected, you can select from the following encoding style options: *Sequential*, *1-Hot*, *2-Hot*, *Gray*, *Johnson* or *Manual*. The coding for each state is displayed as a table in the dialog box.

For example, the following dialog box is displayed for the *Sequential* style in a VHDL state diagram:



If you choose *Manual*, an empty table allows you to enter any required encoding values. Alternatively, the manual encoding can be directly entered on the states in the state diagram or ASM chart.

The states are listed in the table in ascending alphanumeric order and new states are added at the bottom of the column. However, you can re-order the states in descending order by clicking in the State Name cell header. Each subsequent click reverses the sort order. You can also click in the encoding cell header to sort the table in ascending or descending numeric encoding order.

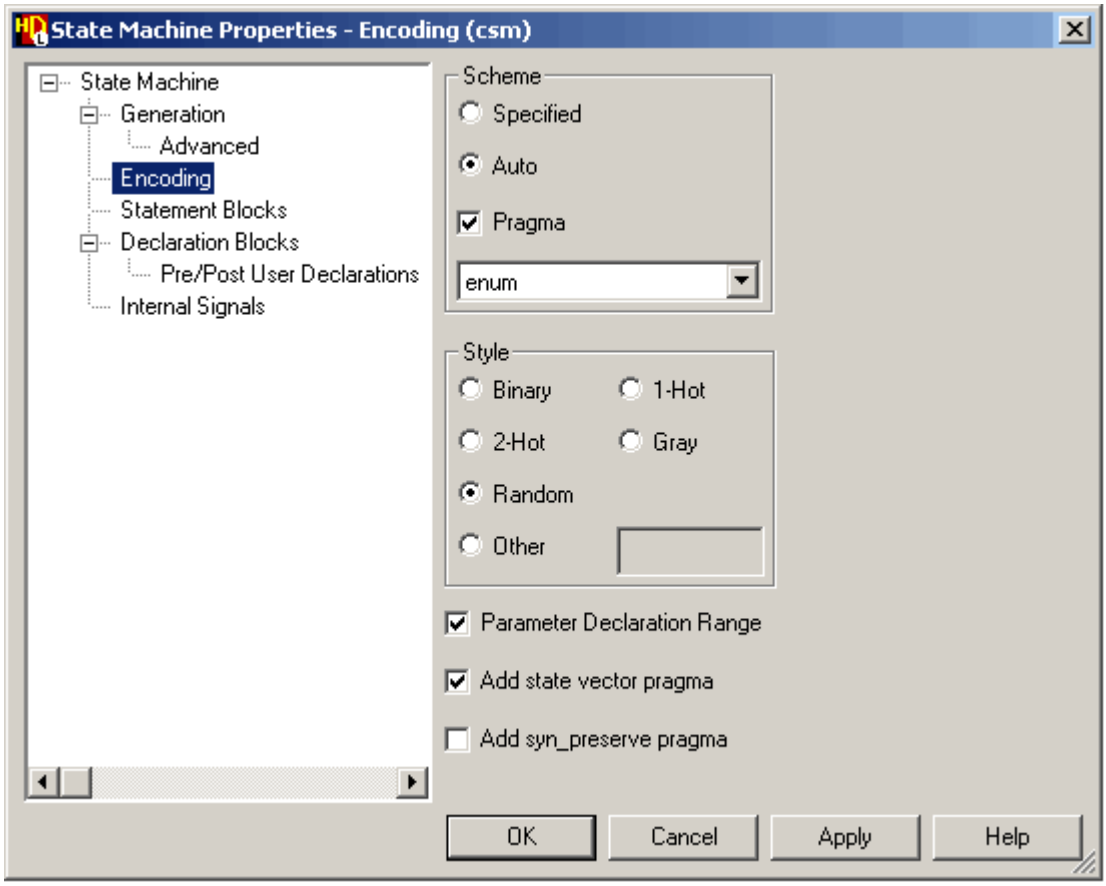
Note that you can change the state associated with each encoding by dragging the state cell with the mouse.

If manual style is selected, you can enter the required encoding for each state directly in the table. You can also copy and paste one or more encoding cells using the **Ctrl**+**C** and the **Ctrl**+**V** keyboard shortcuts.

If you want to explicitly set encoding for some states but allow others to be encoded during synthesis, this can be achieved by using manual encoding with “don’t care” values.

When an automatic scheme is selected, you can set attribute (if you are using VHDL) or pragma (if you are using Verilog) encoding and choose from a dropdown list of VHDL attributes or

Verilog pragmas. For example, the following dialog box shows the *enum* pragma and *Random* encoding algorithm selected in a Verilog state diagram:



If the scheme is set to automatic and the attribute or pragma option is not set, no specific encoding information is included in the generated HDL. This option can be used if you want all specific encoding to be performed during synthesis.

For VHDL, an enumerated type is created for the state variable which contains one enumeration for each state.

For Verilog, a sequential binary parameter value is created which increments by one for each state. This scheme results in the shortest state register length and hence the minimum area. Note however, that binary encoding is not the most efficient scheme for Verilog FPGA designs since FPGA devices typically contain many registers.

Encoding Algorithms

The following table shows examples of the encoding algorithms for each supported style in a state machine with eight states:

Table A-1. Examples of Encoding Algorithms

State Name	Binary	1-Hot	2-Hot	Gray	Johnson	Random
8 states	3 bits	8 bits	5 bits	3 bits	4 bits	3 bits
waiting	000	00000001	00011	000	0000	000
check_locked	001	00000010	00101	001	1000	101
rcv_locked	010	00000100	01001	011	1100	001
read_data	011	00001000	10001	010	1110	100
incr_count	100	00010000	00110	110	1111	010
done_read	101	00100000	01010	111	0111	011
read_stop_bit	110	01000000	10010	101	0011	110
finish_rcv	111	10000000	01100	100	0001	111

Binary or Sequential: Each state is assigned a sequential binary value, incrementing for each state. This algorithm results in the shortest state register length and hence the minimum area. Note however, that binary encoding is not the most efficient scheme for Verilog FPGA designs since FPGA devices typically contain many registers.

1-Hot: The width of the state register is equal to the number of states in the state machine with each bit representing a specific state. This typically results in a fast design although consuming more registers and is commonly used in FPGA designs.

2-Hot: This encoding provides a compromise between area and speed by reducing the number of registers required to represent the state vector. In this case, two bits at a time are used to represent a given state.

Gray: Gray codes are typically used for glitch-free sequential counters since each consecutive code block only requires a single bit change. In general, state machine transitions are more arbitrary resulting in more than one bit change. However, for predominantly sequential state machines, Gray coding should be applied starting with the longest sequential paths first.

Johnson: Johnson encoding typically starts with all zeros and progressively sets each adjacent bit starting with the most significant bit. This continues until all bits are ones. Then the process starts all over again replacing ones with zeros.

Random: Random encoding is a binary code where the values are assigned randomly rather than sequentially. This algorithm results in a minimum state vector width similar to that

achieved with Binary encoding. It is not typically recommended but can be used when the other algorithms fail to give satisfactory results.

VHDL Attribute Encoding

If you are using LeonardoSpectrum or Precision Synthesis, you can enable automatic encoding using a *type_encoding_style* or *type_encoding* attribute.

If you are using Synopsys synthesis tools, you can enable automatic encoding using an *enum_encoding* attribute.

The *type_encoding_style* attribute supports *Binary*, *1-Hot*, *2-Hot*, *Gray*, *Random* or any other specified algorithm.

The *type_encoding* and *enum_encoding* attributes support *Sequential*, *1-Hot*, *2-Hot*, *Gray* and *Johnson* algorithms or you can choose to manually edit the encoding in the dialog box table (or directly on the states in a state diagram or ASM chart).

If you are using the Synplify synthesis tool, you can enable automatic encoding using a *syn_encoding* attribute which supports *Sequential*, *1-Hot*, *Gray* or any other specified algorithm (optionally including the *safe* keyword).

The *type_encoding*, *type_encoding_style*, *enum_encoding* or *syn_encoding* attributes must be declared in a referenced package. Declarations for the LeonardoSpectrum and Precision Synthesis attributes are provided in the *exemplar* package library. The *enum_encoding* and *syn_encoding* attributes should be declared by referencing the appropriate Synopsys or Synplify package.

Verilog Pragma Encoding

If you are using Precision Synthesis, you can enable automatic encoding using a *Sequential*, *1-Hot*, *2-Hot*, *Gray*, *Random* or other specified *enum* pragma.

If you are using the Synplify synthesis tool, you can enable automatic encoding using a *Sequential*, *1-Hot*, *Gray* or other specified *syn_encoding* pragma (optionally including the *safe* keyword).

You can also choose whether to use the range in the declaration of the state encoding parameter.

Signals Status

You can set combinatorial, registered or clocked clocking scheme for output signals and combinatorial or clocked schemes for locally declared signals.

The signals status for a state diagram or ASM chart can be set using the signals table as described in "Signals Table" on page 127.

When a signal set to be registered or clocked, an internal signal is generated by adding a user-specified prefix or suffix to the signal name.

You can set default names for these signals in a state diagram using the State Machine Properties dialog box as described in "Setting State Diagram Internal Signal Names" on page 78 or default names for an ASM chart in the ASM Properties dialog box as described in "Setting ASM Chart Internal Signal Names" on page 120.

Default and Reset Values

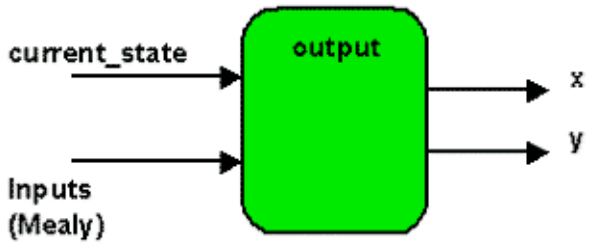
You can optionally set default values and a reset value is required for registered or clocked signals.

Default values ensure that combinatorial signals (and the internal signals generated for registered outputs) are always assigned values to avoid implied latches when the state machine is synthesized.

The reset value is applied to the internal signal for a registered signal or to the actual signal for clocked signals.

Combinatorial Output or Local Signals

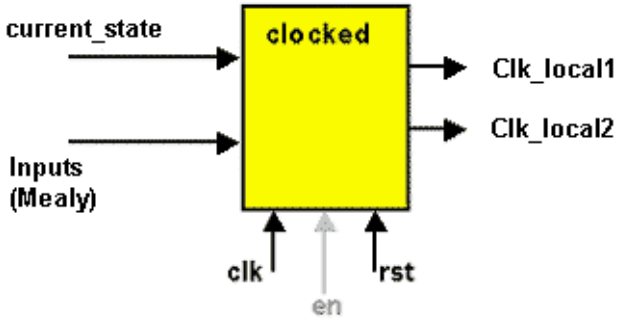
When *Combinatorial* status is set, the original output or locally declared signal is assigned its value in the output code (process or *always* block).



Since there are no registers involved in the output of a combinatorial signal, no reset value is required.

Clocked Local Signals

When *Clocked* status is set for a locally declared signal, the signal is assigned its value in the clocked code. A reset value must be specified and is applied to the original signal.



Since the signal is clocked there is no need for a default value when the design is synthesized although a default can be specified for use in behavioral simulation.

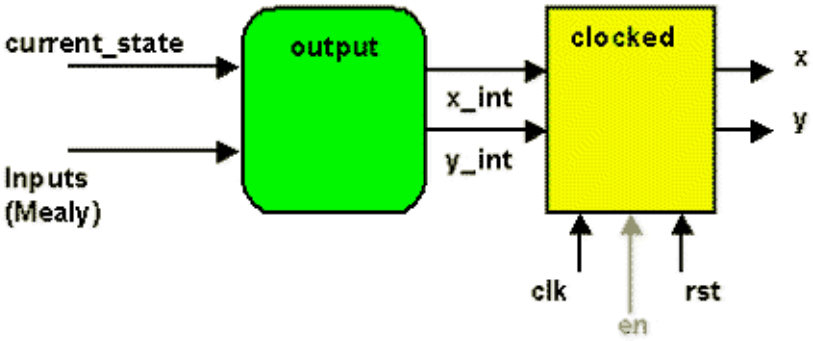
Registered Output Signals

When *Registered* status is set for an output signal, assignment is made to an internal signal using the prefix or suffix specified in the dialog box. For example, *x* is replaced by *x_int*.

All occurrences of the signal are replaced by this internal version on the diagram. This signal is combinatorial (assigned in the output code) and hence requires a default value to avoid latches in synthesis.

The original output signal (for example, *x*) is assigned the value of the internal signal (for example, *x_int*) in the clocked code hence the reset is applied to this original signal.

You can only access the combinatorial internal signal (for example, *x_int*) within the state machine. There is no internal access to the original signal.



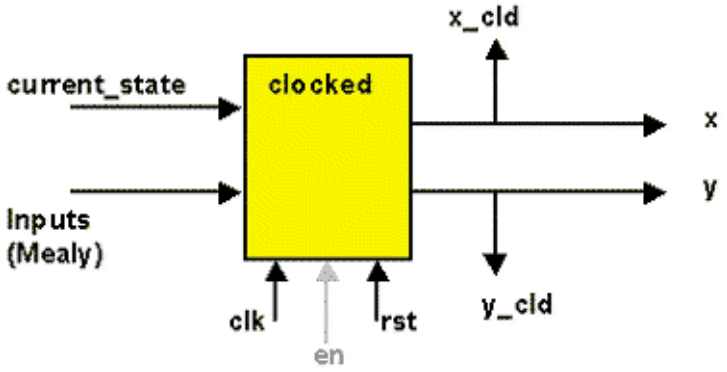
Registered signals should not be used if you intend to use downstream data with the Synopsys FSM compiler.

The FSM compiler processes the core state machine and if registered outputs are required this should be done outside the state machine description.

Clocked Output Signals

When *Clocked* status is set for an output signal, assignment is made to an internal signal using the prefix or suffix specified in the dialog box.

All occurrences of the signal are replaced by this internal version on the diagram. For example, *x* is replaced by *x_cld*. This signal is assigned in the clocked code and hence the reset is applied to this internal signal.



The original output signal (for example, *x*) is assigned the value of the internal signal (for example, *x_int*) in a concurrent assignment.

There is no need for a default value when the design is synthesized although a default can be specified for use in behavioral simulation. If a default value is specified, it is applied to the internal signal. If no default is specified, the previous value is retained.

You can only access the clocked signal (for example, *x_cld*) within the state machine. There is no internal access to the original signal but both always have the same value.

Summary

The following table shows the effect of setting the signal status of output or locally declared signals to combinatorial, registered (for outputs only) or clocked.

The table also shows if any reset is applied to the actual signal or the internal version of the signal.

Scope	Status	Default	Reset	Notes
Output	Combinatorial	No	No	Retains value for simulation. Implied latch in synthesis.
Output	Combinatorial	Yes	No	Active during assignment state or clock cycle before state transition, otherwise default value.
Output	Registered	No	Actual	Retains value for simulation. The state actions are delayed if registered on current state. Implied latch for internal signal in synthesis.
Output	Registered	Yes	Actual	Active during assignment state or one clock cycle for transition, otherwise default value. The state actions are delayed if registered on current state.
Output	Clocked	No	Internal	Retains value for simulation. The state actions are delayed if clocked on current state.
Output	Clocked	Yes	Internal	Active during assignment state or one clock cycle for transition, otherwise default value. The state actions are delayed if clocked on current state.
Local	Combinatorial	No	No	
Local	Combinatorial	Yes	No	
Local	Clocked	No	Actual	Retains value for simulation. The state actions are delayed if clocked on current state.
Local	Clocked	Yes	Actual	Active during assignment state or one clock cycle for transition, otherwise default value. The state actions are delayed if clocked on current state.

Combinatorial outputs (and the internal versions of registered output signals) must be assigned in every possible branch of the state decoding or have default assignments in every branch. Otherwise latches are inferred in synthesis.

Combinatorial outputs are appropriate when the signal does not need retiming or to be registered.

Registered outputs are basically combinatorial with a flip-flop (register) just before the output. These outputs behave like combinatorial signals and the registered signal is not available within the state machine. This type of output can be useful for pipelining when you want to use combinatorial versions of the signals inside the sate machine and just want to retime the outputs.

Clocked outputs are assigned in the clocked process (by definition they are the outputs of flip-flops) and do not need default values to avoid inferring latches. Hence, clocked outputs are more tolerant of missing default values or not being assigned in transitions. Clocked outputs are useful when you want to use combinatorial versions of the signals inside the sate machine as a counter or flag. Clocked outputs are the fail-safe option since they are always clocked and hold their values.

HDL Designer Series Glossary

This glossary defines the standard terminology used in the HDL Designer Series tools.

— A —

action box
A named object on a *flow chart* or *ASM chart* containing *action*s which are executed when the box is entered by a *flow*. Each action box must have one input flow and one output flow. See also *case box*, *decision box* and *wait box*.

action
An operation performed by a *state machine*, *flow chart* or *truth table* which modifies its output signals. In a *state diagram*, there can be *transition actions* executed when an associated *condition* occurs or state actions executed when a *state* is entered. In a flow chart, the actions are executed when a *flow* entering the *action box* is followed. In a truth table, actions are generated from the values assigned to a variable in an output column or can be explicitly entered as additional actions in an unnamed output column. See also *global actions*.

activity trail
A summary of simulation activity (*state*s visited and *transition*s taken) displayed on an animated *state diagram*.

anchor
An anchor attaches a text element to its parent object. For example, the name and *type* of a *signal* in a *block diagram* or the *transition text* and its *transition arc* in a *state diagram*. An anchor is also used to attach a simulation probe to its associated signal.

architecture declarations
User-specified *VHDL* statements which can be entered in a *state diagram*, *flow chart* or *truth table* and are declared for the corresponding *VHDL architecture* in the generated HDL. Architecture declarations are typically used to define local signals or constants. See also *entity declarations* and *process declarations*.

ASIC
ASIC stands for Application Specific Integrated Circuit.

ASM
An algorithmic *state machine* describes the behavior of a system in terms of a defined sequence of operations which produce the required output from the given input data. These sequential operations can be represented using flow chart stye notation as an *ASM chart*.

ASM chart
A graphical representation of an algorithmic state machine which uses flow chart style objects to represent *state*s, *condition*s and *action*s.

asynchronous
An asynchronous process is activated as soon as any of its inputs have any activity on them rather than only being activated on a clock edge. See also *clocking*.

— B —

black box
A view which has HDL translation pragmas set so that it is not analyzed or optimized for synthesis. See also *don't touch*.

black box instance
An instance of a *component* on a *block diagram* or *IBD view* which has no corresponding *design unit*. A black box instance may exist in a partial design which instantiates a view which has not been defined.

block
The representation of a functional object on a *block diagram* or *IBD view*. Also the *design unit* that contains the object definition. A block has a dynamic interface defined by the *signal*s connected to it on the diagram and is typically defined by a *child* block diagram, IBD view, *state diagram*, *flow chart*, truth table or HDL text view. See also *embedded block* and *component*.

block diagram
A diagram editor view which defines a *design unit view* in terms of lower level blocks and *component*s connected by *signal*s. See also *IBD view*.

bottom-up design
The process of designing a system starting from the primitive or leaf-level views and progressing up through parent views until the design is completed. See also *top-down design*.

bounds
The *range* of possible values for a *signal* with integer, floating, enumeration or physical *type*. Also used to specify the index constraint for an array type. A *VHDL* range is normally shown in the format `(15 DOWNTO 0)` or `(0 to 7)`. A *Verilog* range is shown in the format `[15:0]` or `[0:7]`.

breakpoint
A breakpoint can be used to interrupt the progress of a simulation at a specific point in the generated HDL. For example, you could set a breakpoint on a *signal* to interrupt the simulation when the signal changes value or on a *state* to interrupt the simulation when the state is entered.

bundle
A group of *signal*s and/or *bus*es with different *type*s drawn as a composite line on a *block diagram*.

bus
A named vector *signal* with a *type* and *bounds* drawn as a composite line on a *block diagram*. See also *net* and *bundle*.

— C —

case box
A named object which represents a CASE statement on a *flow chart* or *ASM chart*. When used for decoding action logic each Case has an associated End Case object. A case box has one input *flow* and one or more output flows corresponding to the possible values for an evaluated CASE expression. See also *action box*, *decision box*, *if decode box* and *wait box*.

child
A view instantiated below its *parent* in the design hierarchy. A *component* or *block* on a *block diagram* or *IBD view* typically has a child *design unit view* which may be another block diagram, IBD view, *state diagram*, *flow chart*, *truth table* or a *HDL text* view. Also used for the embedded view representing a *hierarchical state* or “hierarchical state box” on page 166 in a hierarchical state machine or a *hierarchical action box* in a hierarchical flow chart or *ASM chart*.

clocked signal
A *signal* in a *state machine* whose value is assigned to an internal signal by the clocked process. This internal signal is continuously assigned to the real output signal. No default value need be specified. Typically used for an internal counter whose value is also required as an output. See also *combinatorial signal* and *registered signal*.

clocking
The timing aspects of behavior can be *asynchronous* or *synchronous* (explicitly clocked).

clock point
An object on an *ASM chart* which displays the clock *signal* name and *condition*. See also *enable point* and reset point.

clone window
A duplicate view of a *graphical editor* window. All select, highlighting and edit operations are made in both windows. However, you can display different parts of the diagram or table in each window.

combinatorial signal
A signal in a *state machine* whose value is directly assigned to the output port. See also *clocked signal* and *registered signal*.

comment graphics
Annotation graphics which can be used for illustration on a *block diagram*, *state diagram*, *flow chart* or *symbol*.

comment text
Annotation text on a *block diagram*, *state diagram*, *flow chart* or *symbol* which can optionally be attached to an object and included as comments or HDL code in the generated HDL for the diagram.

compiled library
A repository within a *library* containing downstream compiled objects usually created by compiling the *HDL* files in a *design data library*.

compiler directive
An instruction to the *Verilog* compiler. Typically used to define library cells or define a macro which controls conditional compilation. Also used to include specified Verilog file or define the simulation time units. The directive is effective from the place it appears in the Verilog code until it is superseded or reset.

complete transition path
The sequence of one or more *partial transition*s going from one *state* to another state (or itself) in a *state machine*. The *condition*s in the transition path are the collection of all the conditions on the individual *transition*s. The *action* in the transition path are the collection of all the actions on the individual transitions plus the actions of the origin state. When tracing the transition path, *link*s are resolved to the referenced *start state*, state or *junction*. See also *partial transition*.

component
A *design unit* that contains a re-usable functional object definition or the instantiation of this object on a *block diagram* or *IBD view*. A component has a fixed interface and may be defined by a child block diagram, IBD view, *state diagram*, *flow chart*, *truth table*, *ModuleWare*, *HDL text*, *external HDL* or *foreign view*. See also *embedded block*, *block* and *port map frame*.

component browser
The component browser is a separate floating window which can be used to browse for components available in the current *library mapping*. Components can be instantiated in an editor view by copy and paste or drag and drop.

concurrent events
Occurrence of two or more events in the same clock cycle.

concurrent statements
Statements which can be entered in a *state diagram*, *flow chart* or *truth table* and are included in the generated HDL at the end of the *VHDL architecture* or *Verilog module*. Concurrent statements are applied to all diagrams in a set of concurrent state machines.

condition
A condition in a *state machine* is a boolean input expression which conforms to *HDL* syntax, and when it evaluates to TRUE, causes a *transition* to occur. The expression usually consists of a signal name, a relational operator and a value. In a flow chart, conditions are used in a *decision box* to determine which output *flow* is followed. In a *truth table*, conditions are generated from

the values assigned to a variable in an input column or can be explicitly entered as additional conditions in an unnamed input column. See also *transition priority*.

configuration

A definition of the *design unit view*s that collectively describe a design by listing the included VHDL entities and architectures. A configuration may also include specification of the values for *VHDL generic*s associated with *component*s in the design. See also *VHDL configuration*.

connectable item

A *node* in a *block diagram*, *flow chart* or *state diagram* that can be the *source* or *destination* of a *signal*, *flow* or *transition*.

current view

The *design unit view* of a *block* or *component* that is currently used. This will be the *default view* unless a loaded *configuration* specifies otherwise.

— D —

decision box

A named object on a *flow chart* or *ASM chart* containing a *condition*. Each decision box has one input *flow* and two output flows (corresponding to the TRUE and FALSE conditions for an IF statement). See also *action box*, *case box* and *wait box*.

default view

The *design unit view* used in hierarchical operations, open commands and HDL generation (unless a loaded *configuration* specifies otherwise). See also *current view*.

design data library

A repository within a *library* containing source design data objects. There are usually different *library mapping*s for *graphical editor* or *HDL text* source views. See also *compiled library*.

design explorer

The *source browser* design explorer windows can be used to browse the content and hierarchy of the source design data using user-defined *viewpoint*s displayed in tree or list format.

design manager

The main *HDL Designer Series* window which is used for library management, data exploration, design flow and version control. The design manager includes a *shortcut bar*, *project manager*, *design explorer*, *side data browser*, *downstream browser*, *task manager* and *template manager*.

design unit

A subdirectory within a *design data library* which is represented by an icon in the *design explorer*. Design units may be *block*s, *component*s or *unknown design unit*s.

design unit view

A description of a *design unit*. Multiple views of *block* or *component* design units can describe alternative implementations. These can include *block diagram*, *IBD view*, *state diagram*, *flow chart*, *truth table* or *HDL text* views.

DesignPad
The built-in *VHDL* and *Verilog* sensitive editor and viewer for *HDL text* views.

destination
The *connectable item* at the end of a *signal*, *transition* or *flow*. See also *source*.

diagram browser
The diagram browser is an optional sub-window which displays the structure and content of the active *diagram editor* view.

diagram editor
An editable *block diagram*, *state diagram*, *flow chart* or *symbol* window which represents a *design unit view* using graphical objects. See also *graphical editor* and *table editor*.

don't touch
A control placed on a *design unit* or *design unit view* which disables specified downstream operations. See also *black box*.

downstream browser
The downstream browser displays the contents of the *compiled library* for the *design data library* currently open in the active *design explorer*. See also *source browser*, *side data browser* and *resource browser*.

downstream only library
A *library* which has *library mapping*s defined only for downstream compiled data.

— E —

embedded block
The representation of an *embedded view* on a *block diagram* or *IBD view* which has a dynamic interface defined by the *signal*s connected to it but unlike a *block* or *component* does not add hierarchy to the design.

embedded view
An embedded view describes concurrent HDL statements on a *block diagram* or *IBD view* and is represented by an *embedded block* which can be defined by a *state diagram*, *flow chart*, *truth table* or *HDL text*.

enable point
An object on an *ASM chart* which displays an enable *signal* name and *condition*. See also *clock point* and *reset point*.

end point
A *flow chart* must have at least one end point which is always named *end*. See also *start point*.

entity declarations
User-specified *VHDL* statements which can be entered as properties in a *symbol* and are added to the corresponding *VHDL entity* declarations in the generated HDL. See also *architecture declarations* and *process declarations*.

entry point
A connector on a *child state diagram* which connects to a *source* in the *parent* state diagram. See also *exit point*.

exit point
A connector on a *child state diagram* which connects to a *destination* in the *parent* state diagram. See also *entry point*.

explicit clock
A *net* on a *block diagram* or *IBD view* which is used as a clock *signal* by the instantiated views of *block*s, *embedded block*s or *component*s. See also *clocking*.

external HDL
A *HDL* description which was not created by a HDL Designer Series tool (for example, user-written *VHDL* or *Verilog*, gate-level HDL models created by synthesis, Inventra, FPGA or 3Soft core models). A port interface must exist for the referenced model as a *VHDL entity* or *Verilog module*. See also *HDL view* and *foreign view*.

— F —

flow
An orthogonal line connecting objects on a *flow chart*. A flow can end on another flow (by creating a *flow join*) but cannot start from a flow.

flow chart
A *diagram editor* view which represents a process in terms of *action box*es, *case box*es, *decision box*es, *wait box*es and *loop*s connected by *flow*s. A flow chart must also contain one *start point* and one or more *end point*s.

flow join
A connection between *flow*s shown as a solid dot where the flows meet.

foreign view
A non-*HDL* description (for example, a C or C++ view) with a registered file type which requires an external HDL generator. See also *external HDL*.

formal
A *signal* or *bus* associated with a *port* on a *component*. Typically, a formal port is connected to an actual signal or bus on the parent view which has the same properties but may have a different name. Formal ports and actual signals with different properties can be connected using a *port map frame*.

FPGA
FPGA stands for Field Programmable Gate Array.

functional primitive
A *block* or *component* that is not further decomposed but fully defined by its own views. However, there may be both a *block diagram* or *IBD view* which describes its behavior in terms of lower level blocks or components and, for example, a *HDL text* view which fully defines its behavior. In this case, the *current view* determines whether the block or component is a functional primitive.

— G —

generate frame
An optional outline which can be used to replicate structure using a FOR frame or conditionally include structure using an IF frame (and ELSE frame in Verilog). Also used in VHDL to cluster concurrent objects using a BLOCK frame.

global actions
Explicit *action* in a *state diagram* or *truth table* which are always performed. In a state machine, global actions are executed on registered signals at an active clock edge or concurrently at a *transition* event on unregistered signals and are used to ensure that default output values are assigned for transitions with no explicit actions defined. See also *state actions* and *transition actions*.

global connector
Any *signal*, *bus*, or *bundle* connected to a global connector is considered to be connected (as an input) to every *block* in the *block diagram* or *IBD view*. It is typically used to connect clock or reset signals.

global net
A global net is a *signal* which can be used on a *block diagram* or *IBD view* but is declared externally in a *VHDL package* or *Verilog include* file. A global net can not be connected to a *block*, external *port* or *global connector*.

graphical editor
An editable window which displays a *diagram editor* or *table editor* view of a *design unit*. See *block diagram*, *IBD view*, *state diagram*, *flow chart*, *symbol*, *truth table* and *tabular IO*.

— H —

HDL
HDL stands for Hardware Description Language and is used in the documentation as a generic term for the *VHDL* or *Verilog* languages. It may also refer to any other language (for example, C) which is being used to describe the behavior of hardware.

HDL2Graphics
HDL2Graphics is a utility program used by *HDL Designer Series* tools to create graphical *block diagram*, *state diagram*, *flow chart* or *IBD view* from source *VHDL* or *Verilog* code.

HDL Author

HDL Author is an advanced environment for *HDL* design which supports design management, HDL text editing using the integrated *DesignPad* text editor, re-usable *ModuleWare* library, version management, and downstream tool interfaces. HDL Author includes *graphical editor*s for maintaining the structure of a design as graphical *block diagram* or *IBD view*s and a *symbol* or *tabular IO* editor for editing *design unit* interfaces. It also includes editors for *state diagram*, *flow chart*, *truth table*, *symbol* and *tabular IO* views which allow an entire design to be represented graphically. A simulation analyzer interface supports error cross-referencing and animation facilities to assist with design de-bug operations.

HDL Designer

The HDL Designer tool includes all the facilities provided by the *HDL Author* tool plus *HDL2Graphics* import which can automatically create editable diagrams from imported HDL code. HDL Designer supports the creation of *block diagram*, *state diagram*, *flow chart* and *IBD view*s.

HDL Designer Series

The HDL Designer Series (HDS) is a family of tools for electronic system design using the *VHDL* and *Verilog* hardware description languages. See also *HDL Detective*, *HDL Author* and *HDL Designer* .

HDL Detective

HDL Detective is the *HDL Designer Series* visualization tool which allows you to import any complete or partial HDL text based design and convert the design into a hierarchy of graphical views. The design structure can be represented as graphical *block diagram*s or *IBD view*s. Primitive leaf-level views can be viewed as block diagram, *state diagram*, *flow chart* or *HDL text* views. A *design manager* can be used to explore the relationship between individual *design unit*s.

HDL text

A textual *HDL* description of a design object. A HDL text *design unit view* may contain structural HDL or define the behavior of a leaf-level *block* or *component design unit*. HDL text may also be used by an *embedded view* on a *block diagram* or *IBD view* to contain concurrent HDL statements which are included in the generated structural code. See also *HDL view*.

HDL text editor

The tool used to edit or view *HDL text* views. The *HDL Designer Series* tools are initially configured to use the built-in *DesignPad* editor but can be set to use many other popular editors.

HDL view

A *design unit view* defined by structural or behavioral *HDL text*. See *Verilog module*, *VHDL entity* and *VHDL architecture*. Also the *VHDL package header* and *VHDL package body* views of a *VHDL package*.

HDM
The Hierarchical Data Model is the internal representation of design data used by the *HDL Designer Series* which allows design objects to be located anywhere in the hierarchy below a physical directory specified in the *library mapping*.

hierarchical action box
The representation on a *flow chart* or *ASM chart* of an embedded *child* diagram which describes *action* logic. See also *action box*.

hierarchical state
The representation on a *state diagram* of an embedded *child* diagram which describes state transitions. See also *simple state*.

hierarchical state box
The representation on an *ASM chart* of an embedded *child* diagram which describes state transitions. See also *state box*.

— I —

IBD view
A *design unit view* described using *Interface-Based Design* which represents the interfaces between instantiated *block*s, *embedded block*s and *component*s as one or more *interconnect table*s showing the *signal* connections between them. See also *block diagram*.

if decode box
A named object which represents an IF statement on an *ASM chart*. When used for decoding action logic each If has an associated End If object. An if decode box has one input *flow* and one or more output flows each corresponding to an evaluated conditional expression. See also *action box*, *case box*,*decision box* and *wait box*.

interconnect cell
A cell at the intersection of a row and a column in an *IBD view*. The interconnect cells specify *port*s connecting *signal*s or *bus*es (defined by the rows) and *block*s, *embedded block*s, *component*s, *external HDL* or *ModuleWare* instances (defined by the columns).

interconnect table
A *table editor* view which represents the connections between one or more *block*s, *embedded block*s, *component*s or *ModuleWare* instances in an *IBD view*. May be abbreviated as ICT.

Interface-Based Design
A methodology which defines the structure of a design in terms of the interfaces between lower level *block*s and *component*s. See also *IBD view*.

interrupt condition
A *condition* associated with a *transition* from an *interrupt point* which applies to every *state* in the *state diagram* and has a higher transition priority than any other transitions.

interrupt point

A *node* on a *state diagram* or *ASM chart* that is implicitly connected to all *state*s on the same diagram. Any *transition* from an interrupt point is treated as an *interrupt condition* from every other state in the diagram. A transition from an interrupt point in the top level diagram is treated as global interrupt condition and applies to all states in a hierarchical state machine. See also *junction* and *entry point*.

— J —

junction

A connector on a *state diagram* that enables a set of *transition*s between *state*s to be replaced by a simpler set of *partial transition*s between the same states. See also *interrupt point* and *entry point*. Also used for a *net connector* joining two *net*s with the same properties on a *block diagram*.

— K —

No entries

— L —

leaf view

An undefined view of a *block* which has been added on a *block diagram* or *IBD view* but has not been defined by a *design unit view*.

library

A repository for source design data and compiled objects that has been assigned a logical name. See also *library mapping*, *regular library*, *protected library* and *downstream only library*.

library mapping

The mapping of a logical *library* name to physical locations. There are typically different mappings for the *design data library* containing *graphical editor* and *HDL text* source views and the *compiled library* containing downstream objects.

link

A connector used on a *state diagram* or *ASM chart* (or between *child* diagrams in the same hierarchy) to avoid long *transition arc*s or *flow*s. A link is implicitly connected to the *state* or *junction*(on a *state diagram*) or to the *state box* (on an *ASM chart*) with the specified name. See also *exit point*.

local declarations

User-specified *Verilog* statements which can be entered as properties for a *flow chart* or *truth table*. These statements are declared at the top of the *always* code in the generated HDL for a truth table. When concurrent flow charts are defined, these declarations are local to each of the individual concurrent flow charts and you can choose whether they are inserted in the *initial* or *always* code. See also *module declarations*.

loop
A loop on a *flow chart* is defined by a start loop and stop loop object connected by a *flow*. A loop is used to repeat a set of sequential statements and can have *Repeat*, *For*, *While* or *Unconditional* control properties.

LPM
A library of parameterizable modules which can be instantiated as *component*s. to implement common gate, arithmetic, storage or pad functions.

— M —

Mealy notation
A Mealy notation *state machine* is defined as a sequential network whose output is a function of both the present *state* and the inputs to the network (*condition*s). In Mealy notation, outputs (*action*) are associated with the *transition*s between states. See also *Moore notation* and *transition actions*.

module declarations
Locally defined *Verilog* statements which can be entered as properties in a *state diagram*, *flow chart*, *truth table* or *symbol* and are declared for the corresponding *Verilog module* in the generated HDL. Module declarations are typically used for 'define, parameter, reg, integer, real, time or wire declarations. See also *local declarations*.

ModuleWare
A library of technology-independent, synthesis-optimized *HDL* generators which can be used to implement many common logic, constant, combinatorial, bit manipulation, arithmetic, register, sequential, memory or primitive functions as instantiated *VHDL* or *Verilog* models.

Moore notation
A Moore notation *state machine* is defined as a sequential network whose outputs (action) are a function of the present *state* only. In Moore notation, actions are associated with the states. See also Mealy notation and *state actions*.

— N —

net
A set of *signal*s or *bus*es which have the same name and *type*. The net represents connections between objects in the design structure and has a value determined by the net's drivers. See also *wire*.

netlist
An ASCII representation of a circuit that lists all of the content of a design and shows how they are interconnected. Typically used for a gate level description as the input to a simulator or place and route tool.

net connector
A net connector can be used on a *block diagram* to join *net*s which have the same properties. It can also be used as an implicit on-page connector between nets with the same properties on the

same diagram or as a dangling connector to terminate nets which are left deliberately unconnected. See also *global connector*, *junction* and *ripper*.

node

A *connectable item* on a *block diagram*, *state diagram*, *ASM chart* or *flow chart*. On a block diagram, it can be a *block*, *embedded block*, *component*, *port map frame*, *global connector*, *port*, *ripper* or *net connector*. On a state diagram, it can be a *state*, *start state*, *hierarchical state*, *junction*, *interrupt point*, *link* and an *entry point* or *exit point* in a *child* hierarchical state diagram. On a flow chart, it can be a *start point*, *action box*, *loop*, *decision box*, *case box*, *wait box* or *end point*.

— O —

object

A general term used for a selectable item or selectable group of closely related items.

object tip

A popup window which displays information about the object under the cursor.

— P —

package list

A list of *VHDL package*s referenced by a *design unit view*. The package list is displayed as a text object on a *block diagram*, *state diagram*, *flow chart* or *symbol*.

panel

A defined and named area on a *block diagram*, *flow chart*, *state diagram* or *symbol* which facilitates viewing or printing the area.

parent

The view immediately above its *child* in the design hierarchy. A *design unit view* appears as a *block* or *component* on its parent *block diagram* or *IBD view*. Also used for the view containing the *hierarchical state* or *hierarchical action box* or *hierarchical state box* representing a hierarchical *state diagram*, *ASM chart* or *flow chart*.

partial condition

The *condition* associated with a *partial transition*.

partial transition

Any *transition* arriving at or leaving a *junction* or *interrupt point* on a *state diagram*. Also the transitions connected to an *entry point* or *exit point* in a *child* hierarchical state diagram. See also *complete transition path*.

polyline

A series of connected straight lines joining one or more points. Polylines may be orthogonal (horizontal and vertical lines only) or may include diagonals. See also *spline*.

port
The external connections for a *design unit* and their representation on a *symbol*, *tabular IO*, *block diagram* or *IBD view*. Also the connections to an instantiated *block*, *embedded block* or *component* on a block diagram or IBD view. The *signal*s connected to *port*s may be inputs, outputs or bidirectional or (for VHDL) buffered. The connection points on objects in an *ASM chart* or *flow chart* are also described as ports.

port map frame
An optional outline around a *component* on a *block diagram* which allows mapping between actual *signal*s on a *block diagram* and *formal port*s which have different properties.

probe
A probe is a text object which can be used to monitor the simulation activity of a *signal* on a *block diagram*. Although a probe can be moved independently, it is permanently attached to its associated signal by an *anchor*.

process declarations
User-specified *VHDL* statements which can be entered on a *flow chart*, *state machine* or *truth table* and are included at the beginning of the corresponding process in the generated HDL. When concurrent flow charts are defined, these declarations are local to each of the individual concurrent flow charts. See also *entity declarations* and *architecture declarations*.

project
The collection of *library mapping* information that the *HDL Designer Series* uses to locate and manage your designs.

project manager
The *source browser* project manager window can be used to set up a *project* and to define, load and configure the *library mapping* for your designs.

protected library
A *library* containing re-usable objects (such as standard VHDL type definitions or shared components) which cannot be edited, generated or compiled.

properties
A mechanism for storing additional information in the data model.

PSL
PSL is a Property Specification Language for the verification of *VHDL* or *Verilog* RTL designs.

— Q —
No entries

— R —

range
The maximum and minimum *bounds* for an integer, floating, physical or enumeration *type*.

recovery state point
A *node* on an *ASM chart* that indicates the *flow* to the recovery *state* used when there is no other valid state assignment.

registered signal
A *signal* in a *state machine* whose value is held as an internal signal which is then assigned to the output port by the clocked process. A default value should be specified to avoid creating latches during synthesis. See also *combinatorial signal* and *clocked signal*.

regular expression
A regular expression is a pattern to be matched against a text string. When found, a string which matches the expression can optionally be replaced by another text string.

regular library
A *library* used for design creation which has *library mapping*s for graphical and HDL text source design objects.

re-level
An operation available in the *state diagram* editor to add or remove hierarchy by moving *state*s into or from a *child* diagram which is represented by a *hierarchical state* on the *parent* diagram.

requirement traceability
The process of tracking a requirement through a design to ensure that it is satisfied.

reset point
A *node* on an *ASM chart* that displays the reset *signal* name and *condition*. See also *clock point* and *enable point*.

resource browser
The resource browser provides a *task manager* for configuring and invoking tasks and a *template manager* for maintaining templates. See also *source browser*, *side data browser* and *downstream browser*.

ripper
A ripper can be used on a *block diagram* to split or combine *net*s which have the same name and *bounds* but represent a different *slice* or element. It can also be used to add or remove nets from a *bundle*. See also *net connector*.

route point
One of a series of points specifying the path of a *net* in a *block diagram* (or a *transition arc* in a *state diagram*). Route points can be connected using *polyline*s or *spline*s.

— S —

selection set
A set of selected objects which are acted on by subsequent operations.

sensitivity list
A list of signals which can be entered in a *flow chart* or *truth table* and are used as the sensitivity list in the generated HDL. The signals defined in the sensitivity list cause the corresponding process to execute when any of the signals changes.

shortcut bar
A customizable control panel which provides shortcuts to *viewpoint*s, *task*s and *ModuleWare component*s.

shortcut key
A keyboard key or key combination that invokes a particular command (also referred to as an accelerator key. See also *toolbar*.

side data
Supplementary source design data (such as EDIF, SDF and document header files) or user data (such as design documents or text files) which is saved with a *design unit view* and can be viewed using the *side data browser*.

side data browser
The *side data* browser displays an expandable indented list showing design and user data associated with the *design unit view* selected in the *design explorer*. See also *source browser*, *resource browser* and *downstream browser*.

signal
A connection or transfer of information between *block*s or *component*s which is represented as a *polyline* or *spline* (with a name and *type*) on a *block diagram*. A set of signals with the same name is called a *net*. See also *bus*.

signals status
A list of the output and locally declared signals in a *state machine* or *ASM chart* which shows the *type* (*VHDL* only), scope (output or local), default value, reset value and status (combinatorial, registered or clocked).

simple state
The representation on a *state diagram* of a *state* which has no *child* state diagram. See also *hierarchical state* and *wait state*.

slice
A slice is used to access a set of contiguous elements within an array type (such as *std_logic_vector*). The left and right limits of the slice must be consistent with the *bounds* of the object.

source
Source design data contained in a *library* as *graphical editor* or *HDL text* views. Also the *connectable item* at the start of a *signal*, *bus*, *transition* or *flow* on a *diagram editor* view. See also *destination*.

source browser
The source browser provides a *project manager* window and any number of *design explorer*s for browsing *source* design objects. See also *side data browser*, *resource browser* and *downstream browser*.

spline
A curved line connecting two or more points. See also *polyline*.

start point
There is one and only one start point in a *flow chart* which is always named *start*. See also *end point*.

start state
The initial *state* of a *state machine*. The start state represents the status of the state machine before any *transition*s occur.

state
A state is a resting mode of a *state machine*. Also the representation of a state on a *state diagram*. Encoding information is shown if manual encoding is enabled and the state may have associated *action*s. See also *hierarchical state*, *simple state*, *start state*, *wait state*, *transition* and *condition*.

state actions
The *action*s associated with a *state* on a *state diagram* which are executed when the state is entered. See also *transition actions* and *global actions*.

state box
A state box is the representation of a *state* on an *ASM chart*. A state box may have associated entry, state and exit *action*s. See also *hierarchical state*.

state diagram
A *diagram editor* representation of a *state machine*. A state diagram typically consists of a number of *state*s, *junction*s, *interrupt point*s or *link*s connected by *transition*s. The diagram may also include text blocks containing *global actions*, *concurrent statements*, *local declarations* and *comment text*. A hierarchical state machine may also include *hierarchical state*s, *entry point*s and *exit point*s.

state machine
A *design unit view* of a *block* or *component* which defines its behavior in terms of a finite state machine (FSM). This is a mathematical model of a system. The system is represented by a finite number of states with a finite number of associated *transition*s between pairs of states. The state machine is represented graphically as a *state diagram*. State machines drawn using *Mealy notation* and *Moore notation* or a mixture of Mealy and Moore notation are supported.

state register statements
User entered statements which can be entered in a *state diagram* and are included in the generated HDL to replace the default state assignment for the *state machine* before the state decode statements at the beginning of a *VHDL* process or *Verilog* always code.

state variable
The name of a *signal* whose value that defines the current *state* of a *state machine*.

status bar
An area at the bottom of the *design manager*, *HDL text editor* or *graphical editor* window that displays information about the current command.

subtree
All objects directly or indirectly below a given object in the design hierarchy.

symbol
A *diagram editor* view which uses graphical objects to define the signal interface of a *component* and its representation when the component is instantiated on a *block diagram*. See also *tabular IO*.

synchronous
A synchronous process is activated on the next explicit clock edge rather than being activated only if any of its inputs are changed. See also *clocking*.

synthesis
The automatic generation of ASIC, FPGA or CPLD designs (circuits) from *HDL* descriptions.

system
Something that performs a specific function or set of functions with defined inputs and outputs. Typically, a self-contained electronic subsystem.

— T —

table editor
An editable *truth table*, *IBD view* or *tabular IO* window which represents a *design unit view* using a tabular matrix of cells. See also *diagram editor* and *graphical editor*.

tabular IO
An alternative *table editor* view showing the interface of a *symbol*.

task
A customizable downstream tool or design flow which can be configured and invoked using the *task manager*.

task manager
The task manager window can be used to create, modify or run a *task*.

template manager
The template manager window can be used to create and modify the templates used for new *graphical editor* or *HDL text* views.

test bench
A test harness which allows a standard set of stimuli to be applied to a design.

toolbar
A group of buttons which provide shortcuts to commonly used commands. The *HDL Designer Series design manager* and *graphical editor* windows typically have several undockable toolbars each supporting a set of related commands. See also *shortcut key*.

tooltip
A small pop-up window that provides descriptive text for a *toolbar* button.

top-down design
The process of designing a system by identifying its major parts, decomposing them into lower level blocks and repeating the process until the desired level of detail is achieved. In electronic design automation, this process is applied to the top-down design of ASIC, FPGA and CPLD circuits using a hardware description language such as *VHDL* or *Verilog*. See also *bottom-up design*.

transition
A change of state within a *state machine*. The transition occurs when an associated *condition* is satisfied. A transition may have associated *transition actions* which are executed when the transition takes place. A transition is represented by a *transition arc* with associated *transition text* in a *state diagram*. See also *transition priority*.

transition actions
The *action* associated with a *transition* in a *state machine* which are executed when the transition occurs. A transition action is the consequence of a *condition*. See also *state actions*.

transition arc
A *polyline* or *spline* representing part of a *transition* between *state*s on a *state diagram*. The direction of the transition is normally shown by an arrow head at its *destination* and the *transition text* is attached to the arc by an anchor.

transition order
The order in which CASE style *transition*s leaving a *state* are generated. CASE style transitions in *VHDL* are mutually exclusive and the order is ignored but the order is significant in *Verilog* since the first match in the generated code is taken.

transition priority
When there are more than one IF style *transition*s leaving a *state*, the associated *condition*s are evaluated in the order of their priority. The transition priority is shown by an integer on the *transition arc* adjacent to the *source* state. However, a *transition* with the *condition* OTHERS is always evaluated last.

transition text
The *condition* text (in a *Moore notation transition*) or the *condition* and *action* text (in a *Mealy notation* transition) which is attached to the *transition arc* by an *anchor*.

truth table
A *table editor* view which represents one or more output signals by the logical state of one or more input signals.

type
Specifies the characteristics and allowed values of a *net*. In VHDL, all *signal*s, *bus*es, variables and constants have a specific *VHDL* type definition which is defined in a *package list*. In *Verilog*, a net may have *wire*, *tri*, *wor*, *trior*, *wand*, *triand*, *tri0*, *tri1*, *supply0*, *supply1*, *reg*, *trireg*, *real*, *integer*, *time* or *realtime* type. The values for a bus may also be limited by a *bounds* constraint.

— U —

universe
The total area available for a diagram.

unknown design unit
A *design unit* which is not defined as a *block*, *component* or *package list*.

unknown design unit view
A *design unit view* representing data that is not defined as a *graphical editor*, *HDL text* or other registered view. Typically contains a text description and is treated as a text view for open, print or other file operations.

user directory
On UNIX, this is the home directory used when you login which contains your startup files and is normally located by the HOME environment variable. On a PC, an application data directory is created when you use a tool for the first time. On Windows NT, this is created in the profiles directory. For example:
C:\Winnt\Profiles\<user>\Application Data\HDL Designer Series
On a Windows XP machine, the application data directory is located below the *Documents and Settings* directory. For example:
C:\Documents and Settings\<user>\Application Data\HDL Designer Series Typically, the user directory will contain your preferences and library mapping files unless you have explicitly saved these files in alternative locations.

— V —

Verilog
A hardware description language (compliant with IEEE standard 1364-1995) that can be used to design, model and simulate electronic circuits. Verilog is a registered trademark of Cadence Design Systems Inc. See also *HDL* and *VHDL*.

Verilog include
A *Verilog* file containing global declarations or other Verilog code which can be included by reference using the `include *compiler directive*.

Verilog module
A *design unit view* of a *block* or *component* which defines its behavior using *Verilog* source code.

Verilog module body
Describes the boundaries and content of a *Verilog* logic block in structural, dataflow and behavioral constructs.

Verilog parameter
A Verilog parameter is a constant value used to parameterize a *Verilog* design description. Verilog parameters are used in a similar way to *VHDL generic*s.

VHDL
VHDL stands for VHSIC (Very High Speed Integrated Circuit) Hardware Description Language. VHDL is a design and modelling language (compliant with IEEE standards 1076-1987, 1076-1993 and 1076-2002) which was specifically created to describe (in machine and human-readable form) the organization and function of digital hardware systems and circuit boards. See also *HDL* and *Verilog*.

VHDL architecture
A *design unit view* of a *block* or *component* which defines its behavior using *VHDL* source code.

VHDL architecture body
Declares the items available inside a *VHDL design entity* and specifies the relationships between inputs and outputs. An architecture body describes the organization and operations performed inside the design entity. You can choose to store the VHDL architecture body in the same file or in a separate file from the *VHDL entity*.

VHDL configuration
A declaration which specifies the *VHDL architecture body* used to define a *VHDL design entity*. See also *configuration*.

VHDL design entity
A VHDL design entity is the primary abstraction level of a *VHDL* hardware model which typically represents a cell, chip, board or subsystem. A VHDL design entity comprises a *VHDL entity* declaration and a *VHDL architecture body*.

VHDL entity
Declares the interface between a *VHDL design entity* and its external environment. An entity declaration contains definitions of inputs to and outputs from the VHDL design entity. VHDL entity declarations can optionally be stored in the same file or a separate file from the associated *VHDL architecture body*.

VHDL generic
A VHDL generic is a constant value used to parameterize a *VHDL* design description. VHDL generics are used in a similar way to *Verilog parameter*s.

VHDL package
A *VHDL* object that contains procedural definitions and declarations used by *design unit view*s of *block*s or *component*s. Typically contains *type* and subtype definitions. Usually comprises a separate *VHDL package header* containing declarations and a *VHDL package body* containing any functions or procedures declared in the package header.

VHDL package body
The part of a *VHDL package* which defines the implementation of objects in the package. It contains data used when the design is evaluated. The package body typically contains constant definitions and function bodies.

VHDL package header
The part of a *VHDL package* which declares the objects defined in the package. It is referenced by *block* and *component* views.

viewpoint
A set of user-defined rules which examine particular aspects of a design.

VITAL
VITAL stands for the *VHDL* Initiative Towards ASIC Libraries which is an IEEE standard (IEEE1076.4) for *ASIC* library design.

— W —

wait box
A named object on a *flow chart* containing a conditional wait statement which controls the delay before an event occurs on a signal in the *sensitivity list*. See also *action box*, *case box* and *decision box*.

wait state
A wait state has similar properties to a *simple state* but introduces a delay of two or more clock cycles.

whisker
A line that extends between a *port* on the boundary of a customized *block* or *component symbol* and the body of the block or component symbol.

wire
A segment of a *net* on a *VHDL* or *Verilog block diagram*. A wire may have *signal* or *bus* style and scalar or vector *type* and should not be confused with the Verilog wire type.

working directory
On UNIX, the directory from which you invoked the application. On a PC, the working directory defaults to the *user directory* or can be set using the **Start In** option when you define the properties for a short cut to your application. Do not set a working directory using the **Start In** shortcut option if you want to use object linking and embedding (OLE) to import objects into a documentation tool as the application will not be able to access library mapping information from this location.

workspace
A working environment which allows common design data to be shared between multiple users. Typically, a project comprises one or more shared workspaces and a private workspace (often described as a sandbox) for each engineer working on the project.

— X —

Xdefaults

A set of resources which can be used to set the default display characteristics on X server window systems.

— Y —

No entries

— Z —

No entries

Index

— T —

— W —

End-User License Agreement

The latest version of the End-User License Agreement is available on-line at:
www.mentor.com/eula

IMPORTANT INFORMATION

USE OF ALL SOFTWARE IS SUBJECT TO LICENSE RESTRICTIONS. CAREFULLY READ THIS LICENSE AGREEMENT BEFORE USING THE PRODUCTS. USE OF SOFTWARE INDICATES CUSTOMER'S COMPLETE AND UNCONDITIONAL ACCEPTANCE OF THE TERMS AND CONDITIONS SET FORTH IN THIS AGREEMENT. ANY ADDITIONAL OR DIFFERENT PURCHASE ORDER TERMS AND CONDITIONS SHALL NOT APPLY.

END-USER LICENSE AGREEMENT ("Agreement")

This is a legal agreement concerning the use of Software (as defined in Section 2) and hardware (collectively "Products") between the company acquiring the Products ("Customer"), and the Mentor Graphics entity that issued the corresponding quotation or, if no quotation was issued, the applicable local Mentor Graphics entity ("Mentor Graphics"). Except for license agreements related to the subject matter of this license agreement which are physically signed by Customer and an authorized representative of Mentor Graphics, this Agreement and the applicable quotation contain the parties' entire understanding relating to the subject matter and supersede all prior or contemporaneous agreements. If Customer does not agree to these terms and conditions, promptly return or, in the case of Software received electronically, certify destruction of Software and all accompanying items within five days after receipt of Software and receive a full refund of any license fee paid.

1. **ORDERS, FEES AND PAYMENT.**

   1.1. To the extent Customer (or if agreed by Mentor Graphics, Customer's appointed third party buying agent) places and Mentor Graphics accepts purchase orders pursuant to this Agreement ("Order(s)"), each Order will constitute a contract between Customer and Mentor Graphics, which shall be governed solely and exclusively by the terms and conditions of this Agreement, any applicable addenda and the applicable quotation, whether or not these documents are referenced on the Order. Any additional or conflicting terms and conditions appearing on an Order will not be effective unless agreed in writing by an authorized representative of Customer and Mentor Graphics.

   1.2. Amounts invoiced will be paid, in the currency specified on the applicable invoice, within 30 days from the date of such invoice. Any past due invoices will be subject to the imposition of interest charges in the amount of one and one-half percent per month or the applicable legal rate currently in effect, whichever is lower. Prices do not include freight, insurance, customs duties, taxes or other similar charges, which Mentor Graphics will state separately in the applicable invoice(s). Unless timely provided with a valid certificate of exemption or other evidence that items are not taxable, Mentor Graphics will invoice Customer for all applicable taxes including, but not limited to, VAT, GST, sales tax and service tax. Customer will make all payments free and clear of, and without reduction for, any withholding or other taxes; any such taxes imposed on payments by Customer hereunder will be Customer's sole responsibility. If Customer appoints a third party to place purchase orders and/or make payments on Customer's behalf, Customer shall be liable for payment under Orders placed by such third party in the event of default.

   1.3. All Products are delivered FCA factory (Incoterms 2000), freight prepaid and invoiced to Customer, except Software delivered electronically, which shall be deemed delivered when made available to Customer for download. Mentor Graphics retains a security interest in all Products delivered under this Agreement, to secure payment of the purchase price of such Products, and Customer agrees to sign any documents that Mentor Graphics determines to be necessary or convenient for use in filing or perfecting such security interest. Mentor Graphics' delivery of Software by electronic means is subject to Customer's provision of both a primary and an alternate e-mail address.

2. **GRANT OF LICENSE.** The software installed, downloaded, or otherwise acquired by Customer under this Agreement, including any updates, modifications, revisions, copies, documentation and design data ("Software") are copyrighted, trade secret and confidential information of Mentor Graphics or its licensors, who maintain exclusive title to all Software and retain all rights not expressly granted by this Agreement. Mentor Graphics grants to Customer, subject to payment of applicable license fees, a nontransferable, nonexclusive license to use Software solely: (a) in machine-readable, object-code form (except as provided in Subsection 5.2); (b) for Customer's internal business purposes; (c) for the term of the license; and (d) on the computer hardware and at the site authorized by Mentor Graphics. A site is restricted to a one-half mile (800 meter) radius. Customer may have Software temporarily used by an employee for telecommuting purposes from locations other than a Customer office, such as the employee's residence, an airport or hotel, provided that such employee's primary place of employment is the site where the Software is authorized for use. Mentor Graphics' standard policies and programs, which vary depending on Software, license fees paid or services purchased, apply to the following: (a) relocation of Software; (b) use of Software, which may be limited, for example, to execution of a single session by a single user on the authorized hardware or for a restricted period of time (such limitations may be technically implemented through the use of authorization codes or similar devices); and (c) support services provided, including eligibility to receive telephone support, updates, modifications, and revisions. For the avoidance of doubt, if Customer requests any change or enhancement to Software, whether in the course of

receiving support or consulting services, evaluating Software, performing beta testing or otherwise, any inventions, product improvements, modifications or developments made by Mentor Graphics (at Mentor Graphics' sole discretion) will be the exclusive property of Mentor Graphics.

3. **ESC SOFTWARE.** If Customer purchases a license to use development or prototyping tools of Mentor Graphics' Embedded Software Channel ("ESC"), Mentor Graphics grants to Customer a nontransferable, nonexclusive license to reproduce and distribute executable files created using ESC compilers, including the ESC run-time libraries distributed with ESC C and C++ compiler Software that are linked into a composite program as an integral part of Customer's compiled computer program, provided that Customer distributes these files only in conjunction with Customer's compiled computer program. Mentor Graphics does NOT grant Customer any right to duplicate, incorporate or embed copies of Mentor Graphics' real-time operating systems or other embedded software products into Customer's products or applications without first signing or otherwise agreeing to a separate agreement with Mentor Graphics for such purpose.

4. **BETA CODE.**

4.1. Portions or all of certain Software may contain code for experimental testing and evaluation ("Beta Code"), which may not be used without Mentor Graphics' explicit authorization. Upon Mentor Graphics' authorization, Mentor Graphics grants to Customer a temporary, nontransferable, nonexclusive license for experimental use to test and evaluate the Beta Code without charge for a limited period of time specified by Mentor Graphics. This grant and Customer's use of the Beta Code shall not be construed as marketing or offering to sell a license to the Beta Code, which Mentor Graphics may choose not to release commercially in any form.

4.2. If Mentor Graphics authorizes Customer to use the Beta Code, Customer agrees to evaluate and test the Beta Code under normal conditions as directed by Mentor Graphics. Customer will contact Mentor Graphics periodically during Customer's use of the Beta Code to discuss any malfunctions or suggested improvements. Upon completion of Customer's evaluation and testing, Customer will send to Mentor Graphics a written evaluation of the Beta Code, including its strengths, weaknesses and recommended improvements.

4.3. Customer agrees to maintain Beta Code in confidence and shall restrict access to the Beta Code, including the methods and concepts utilized therein, solely to those employees and Customer location(s) authorized by Mentor Graphics to perform beta testing. Customer agrees that any written evaluations and all inventions, product improvements, modifications or developments that Mentor Graphics conceived or made during or subsequent to this Agreement, including those based partly or wholly on Customer's feedback, will be the exclusive property of Mentor Graphics. Mentor Graphics will have exclusive rights, title and interest in all such property. The provisions of this Subsection 4.3 shall survive termination of this Agreement.

5. **RESTRICTIONS ON USE.**

5.1. Customer may copy Software only as reasonably necessary to support the authorized use. Each copy must include all notices and legends embedded in Software and affixed to its medium and container as received from Mentor Graphics. All copies shall remain the property of Mentor Graphics or its licensors. Customer shall maintain a record of the number and primary location of all copies of Software, including copies merged with other software, and shall make those records available to Mentor Graphics upon request. Customer shall not make Products available in any form to any person other than Customer's employees and on-site contractors, excluding Mentor Graphics competitors, whose job performance requires access and who are under obligations of confidentiality. Customer shall take appropriate action to protect the confidentiality of Products and ensure that any person permitted access does not disclose or use it except as permitted by this Agreement. Customer shall give Mentor Graphics written notice of any unauthorized disclosure or use of the Products as soon as Customer learns or becomes aware of such unauthorized disclosure or use. Except as otherwise permitted for purposes of interoperability as specified by applicable and mandatory local law, Customer shall not reverse-assemble, reverse-compile, reverse-engineer or in any way derive any source code from Software. Log files, data files, rule files and script files generated by or for the Software (collectively "Files"), including without limitation files containing Standard Verification Rule Format ("SVRF") and Tcl Verification Format ("TVF") which are Mentor Graphics' proprietary syntaxes for expressing process rules, constitute or include confidential information of Mentor Graphics. Customer may share Files with third parties, excluding Mentor Graphics competitors, provided that the confidentiality of such Files is protected by written agreement at least as well as Customer protects other information of a similar nature or importance, but in any case with at least reasonable care. Customer may use Files containing SVRF or TVF only with Mentor Graphics products. Under no circumstances shall Customer use Software or Files or allow their use for the purpose of developing, enhancing or marketing any product that is in any way competitive with Software, or disclose to any third party the results of, or information pertaining to, any benchmark.

5.2. If any Software or portions thereof are provided in source code form, Customer will use the source code only to correct software errors and enhance or modify the Software for the authorized use. Customer shall not disclose or permit disclosure of source code, in whole or in part, including any of its methods or concepts, to anyone except Customer's employees or contractors, excluding Mentor Graphics competitors, with a need to know. Customer shall not copy or compile source code in any manner except to support this authorized use.

5.3. Customer may not assign this Agreement or the rights and duties under it, or relocate, sublicense or otherwise transfer the Products, whether by operation of law or otherwise ("Attempted Transfer"), without Mentor Graphics' prior written consent and payment of Mentor Graphics' then-current applicable relocation and/or transfer fees. Any Attempted Transfer without Mentor Graphics' prior written consent shall be a material breach of this Agreement and may, at Mentor Graphics' option, result in the immediate termination of the Agreement and/or the licenses granted under this Agreement. The terms

of this Agreement, including without limitation the licensing and assignment provisions, shall be binding upon Customer's permitted successors in interest and assigns.

5.4. The provisions of this Section 5 shall survive the termination of this Agreement.

6. **SUPPORT SERVICES.** To the extent Customer purchases support services, Mentor Graphics will provide Customer updates and technical support for the Products, at the Customer site(s) for which support is purchased, in accordance with Mentor Graphics' then current End-User Support Terms located at http://supportnet.mentor.com/about/legal/.

7. **AUTOMATIC CHECK FOR UPDATES; PRIVACY.** Technological measures in Software may communicate with servers of Mentor Graphics or its contractors for the purpose of checking for and notifying the user of updates and to ensure that the Software in use is licensed in compliance with this Agreement. Mentor Graphics will not collect any personally identifiable data in this process and will not disclose any data collected to any third party without the prior written consent of Customer, except to Mentor Graphics' outside attorneys or as may be required by a court of competent jurisdiction.

8. **LIMITED WARRANTY.**

8.1. Mentor Graphics warrants that during the warranty period its standard, generally supported Products, when properly installed, will substantially conform to the functional specifications set forth in the applicable user manual. Mentor Graphics does not warrant that Products will meet Customer's requirements or that operation of Products will be uninterrupted or error free. The warranty period is 90 days starting on the 15th day after delivery or upon installation, whichever first occurs. Customer must notify Mentor Graphics in writing of any nonconformity within the warranty period. For the avoidance of doubt, this warranty applies only to the initial shipment of Software under an Order and does not renew or reset, for example, with the delivery of (a) Software updates or (b) authorization codes or alternate Software under a transaction involving Software re-mix. This warranty shall not be valid if Products have been subject to misuse, unauthorized modification or improper installation. MENTOR GRAPHICS' ENTIRE LIABILITY AND CUSTOMER'S EXCLUSIVE REMEDY SHALL BE, AT MENTOR GRAPHICS' OPTION, EITHER (A) REFUND OF THE PRICE PAID UPON RETURN OF THE PRODUCTS TO MENTOR GRAPHICS OR (B) MODIFICATION OR REPLACEMENT OF THE PRODUCTS THAT DO NOT MEET THIS LIMITED WARRANTY, PROVIDED CUSTOMER HAS OTHERWISE COMPLIED WITH THIS AGREEMENT. MENTOR GRAPHICS MAKES NO WARRANTIES WITH RESPECT TO: (A) SERVICES; (B) PRODUCTS PROVIDED AT NO CHARGE; OR (C) BETA CODE; ALL OF WHICH ARE PROVIDED "AS IS."

8.2. THE WARRANTIES SET FORTH IN THIS SECTION 8 ARE EXCLUSIVE. NEITHER MENTOR GRAPHICS NOR ITS LICENSORS MAKE ANY OTHER WARRANTIES EXPRESS, IMPLIED OR STATUTORY, WITH RESPECT TO PRODUCTS PROVIDED UNDER THIS AGREEMENT. MENTOR GRAPHICS AND ITS LICENSORS SPECIFICALLY DISCLAIM ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY.

9. **LIMITATION OF LIABILITY.** EXCEPT WHERE THIS EXCLUSION OR RESTRICTION OF LIABILITY WOULD BE VOID OR INEFFECTIVE UNDER APPLICABLE LAW, IN NO EVENT SHALL MENTOR GRAPHICS OR ITS LICENSORS BE LIABLE FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES (INCLUDING LOST PROFITS OR SAVINGS) WHETHER BASED ON CONTRACT, TORT OR ANY OTHER LEGAL THEORY, EVEN IF MENTOR GRAPHICS OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT SHALL MENTOR GRAPHICS' OR ITS LICENSORS' LIABILITY UNDER THIS AGREEMENT EXCEED THE AMOUNT RECEIVED FROM CUSTOMER FOR THE HARDWARE, SOFTWARE LICENSE OR SERVICE GIVING RISE TO THE CLAIM. IN THE CASE WHERE NO AMOUNT WAS PAID, MENTOR GRAPHICS AND ITS LICENSORS SHALL HAVE NO LIABILITY FOR ANY DAMAGES WHATSOEVER. THE PROVISIONS OF THIS SECTION 9 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

10. **HAZARDOUS APPLICATIONS.** CUSTOMER ACKNOWLEDGES IT IS SOLELY RESPONSIBLE FOR TESTING ITS PRODUCTS USED IN APPLICATIONS WHERE THE FAILURE OR INACCURACY OF ITS PRODUCTS MIGHT RESULT IN DEATH OR PERSONAL INJURY ("HAZARDOUS APPLICATIONS"). NEITHER MENTOR GRAPHICS NOR ITS LICENSORS SHALL BE LIABLE FOR ANY DAMAGES RESULTING FROM OR IN CONNECTION WITH THE USE OF MENTOR GRAPHICS PRODUCTS IN OR FOR HAZARDOUS APPLICATIONS. THE PROVISIONS OF THIS SECTION 10 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

11. **INDEMNIFICATION.** CUSTOMER AGREES TO INDEMNIFY AND HOLD HARMLESS MENTOR GRAPHICS AND ITS LICENSORS FROM ANY CLAIMS, LOSS, COST, DAMAGE, EXPENSE OR LIABILITY, INCLUDING ATTORNEYS' FEES, ARISING OUT OF OR IN CONNECTION WITH THE USE OF PRODUCTS AS DESCRIBED IN SECTION 10. THE PROVISIONS OF THIS SECTION 11 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

12. **INFRINGEMENT.**

12.1. Mentor Graphics will defend or settle, at its option and expense, any action brought against Customer in the United States, Canada, Japan, or member state of the European Union which alleges that any standard, generally supported Product acquired by Customer hereunder infringes a patent or copyright or misappropriates a trade secret in such jurisdiction. Mentor Graphics will pay costs and damages finally awarded against Customer that are attributable to the action. Customer understands and agrees that as conditions to Mentor Graphics' obligations under this section Customer must: (a) notify Mentor Graphics promptly in writing of the action; (b) provide Mentor Graphics all reasonable information and assistance

to settle or defend the action; and (c) grant Mentor Graphics sole authority and control of the defense or settlement of the action.

12.2. If a claim is made under Subsection 12.1 Mentor Graphics may, at its option and expense, (a) replace or modify the Product so that it becomes noninfringing; (b) procure for Customer the right to continue using the Product; or (c) require the return of the Product and refund to Customer any purchase price or license fee paid, less a reasonable allowance for use.

12.3. Mentor Graphics has no liability to Customer if the action is based upon: (a) the combination of Software or hardware with any product not furnished by Mentor Graphics; (b) the modification of the Product other than by Mentor Graphics; (c) the use of other than a current unaltered release of Software; (d) the use of the Product as part of an infringing process; (e) a product that Customer makes, uses, or sells; (f) any Beta Code or Product provided at no charge; (g) any software provided by Mentor Graphics' licensors who do not provide such indemnification to Mentor Graphics' customers; or (h) infringement by Customer that is deemed willful. In the case of (h), Customer shall reimburse Mentor Graphics for its reasonable attorney fees and other costs related to the action.

12.4. THIS SECTION 12 IS SUBJECT TO SECTION 9 ABOVE AND STATES THE ENTIRE LIABILITY OF MENTOR GRAPHICS AND ITS LICENSORS FOR DEFENSE, SETTLEMENT AND DAMAGES, AND CUSTOMER'S SOLE AND EXCLUSIVE REMEDY, WITH RESPECT TO ANY ALLEGED PATENT OR COPYRIGHT INFRINGEMENT OR TRADE SECRET MISAPPROPRIATION BY ANY PRODUCT PROVIDED UNDER THIS AGREEMENT.

13. **TERMINATION AND EFFECT OF TERMINATION.** If a Software license was provided for limited term use, such license will automatically terminate at the end of the authorized term.

13.1. Mentor Graphics may terminate this Agreement and/or any license granted under this Agreement immediately upon written notice if Customer: (a) exceeds the scope of the license or otherwise fails to comply with the licensing or confidentiality provisions of this Agreement, or (b) becomes insolvent, files a bankruptcy petition, institutes proceedings for liquidation or winding up or enters into an agreement to assign its assets for the benefit of creditors. For any other material breach of any provision of this Agreement, Mentor Graphics may terminate this Agreement and/or any license granted under this Agreement upon 30 days written notice if Customer fails to cure the breach within the 30 day notice period. Termination of this Agreement or any license granted hereunder will not affect Customer's obligation to pay for Products shipped or licenses granted prior to the termination, which amounts shall be payable immediately upon the date of termination.

13.2. Upon termination of this Agreement, the rights and obligations of the parties shall cease except as expressly set forth in this Agreement. Upon termination, Customer shall ensure that all use of the affected Products ceases, and shall return hardware and either return to Mentor Graphics or destroy Software in Customer's possession, including all copies and documentation, and certify in writing to Mentor Graphics within ten business days of the termination date that Customer no longer possesses any of the affected Products or copies of Software in any form.

14. **EXPORT.** The Products provided hereunder are subject to regulation by local laws and United States government agencies, which prohibit export or diversion of certain products and information about the products to certain countries and certain persons. Customer agrees that it will not export Products in any manner without first obtaining all necessary approval from appropriate local and United States government agencies.

15. **U.S. GOVERNMENT LICENSE RIGHTS.** Software was developed entirely at private expense. All Software is commercial computer software within the meaning of the applicable acquisition regulations. Accordingly, pursuant to US FAR 48 CFR 12.212 and DFAR 48 CFR 227.7202, use, duplication and disclosure of the Software by or for the U.S. Government or a U.S. Government subcontractor is subject solely to the terms and conditions set forth in this Agreement, except for provisions which are contrary to applicable mandatory federal laws.

16. **THIRD PARTY BENEFICIARY.** Mentor Graphics Corporation, Mentor Graphics (Ireland) Limited, Microsoft Corporation and other licensors may be third party beneficiaries of this Agreement with the right to enforce the obligations set forth herein.

17. **REVIEW OF LICENSE USAGE.** Customer will monitor the access to and use of Software. With prior written notice and during Customer's normal business hours, Mentor Graphics may engage an internationally recognized accounting firm to review Customer's software monitoring system and records deemed relevant by the internationally recognized accounting firm to confirm Customer's compliance with the terms of this Agreement or U.S. or other local export laws. Such review may include FLEXlm or FLEXnet (or successor product) report log files that Customer shall capture and provide at Mentor Graphics' request. Customer shall make records available in electronic format and shall fully cooperate with data gathering to support the license review. Mentor Graphics shall bear the expense of any such review unless a material non-compliance is revealed. Mentor Graphics shall treat as confidential information all information gained as a result of any request or review and shall only use or disclose such information as required by law or to enforce its rights under this Agreement. The provisions of this Section 17 shall survive the termination of this Agreement.

18. **CONTROLLING LAW, JURISDICTION AND DISPUTE RESOLUTION.** The owners of certain Mentor Graphics intellectual property licensed under this Agreement are located in Ireland and the United States. To promote consistency around the world, disputes shall be resolved as follows: excluding conflict of laws rules, this Agreement shall be governed by and construed under the laws of the State of Oregon, USA, if Customer is located in North or South America, and the laws of Ireland if Customer is located outside of North or South America. All disputes arising out of or in relation to this Agreement shall be submitted to the exclusive jurisdiction of the courts of Portland, Oregon when the laws of Oregon apply, or Dublin, Ireland when the laws of Ireland apply. Notwithstanding the foregoing, all disputes in Asia arising out of or in relation to this Agreement shall be resolved by arbitration in Singapore before a single arbitrator to be appointed by the chairman of the Singapore International

Arbitration Centre ("SIAC") to be conducted in the English language, in accordance with the Arbitration Rules of the SIAC in effect at the time of the dispute, which rules are deemed to be incorporated by reference in this section. This section shall not restrict Mentor Graphics' right to bring an action against Customer in the jurisdiction where Customer's place of business is located. The United Nations Convention on Contracts for the International Sale of Goods does not apply to this Agreement.

19. **SEVERABILITY.** If any provision of this Agreement is held by a court of competent jurisdiction to be void, invalid, unenforceable or illegal, such provision shall be severed from this Agreement and the remaining provisions will remain in full force and effect.

20. **MISCELLANEOUS.** This Agreement contains the parties' entire understanding relating to its subject matter and supersedes all prior or contemporaneous agreements, including but not limited to any purchase order terms and conditions. Some Software may contain code distributed under a third party license agreement that may provide additional rights to Customer. Please see the applicable Software documentation for details. This Agreement may only be modified in writing by authorized representatives of the parties. Waiver of terms or excuse of breach must be in writing and shall not constitute subsequent consent, waiver or excuse.

Rev. 100615, Part No. 246066