A logic network can be modeled using continuous assignments or switches and logic gates. Gates and continuous assignments serve different modeling purposes and it is important to appreciate the differences between them in order to achieve the right balance between accuracy and efficiency in Verilog-XL. Modeling with switches and logic gates has the following advantages:

- Gates provide a much closer one to one mapping between the actual circuit and the network model.
- There is no continuous assignment equivalent to the bidirectional transfer gate.

A limitation of those nets declared with the keyword **vectored** affects gates and switches as well as continuous assignments. Individual bits of vectored nets cannot be driven; thus, gates and switches can only drive scalar output nets. If you declare a multi-bit net as **vectored** and you drive individual bits of it, Verilog-XL will display a compilation error message. If you do not declare a multi-bit net as **vectored**, Verilog-XL handles it as a vector except in the following cases. A multi-bit net is handled as a scalar if:

- part of the vector is driven by a gate or switch.
- part of the vector is assigned a value with a continuous assignment.
In Verilog-XL, gate and switch level modeling is superior to continuous assignment modeling for the following two reasons:

1. Because gates and switches have fixed functions, Verilog-XL can optimize its data structure so as to reduce the amount of memory needed to simulate large circuits.
2. For a random network of nets, it is likely that the use of gates and switches for modeling gives a shorter simulation run time than the use of continuous assignments.

6.1 Gate and Switch Declaration Syntax

A gate or switch declaration names a gate or switch type and specifies its output signal strengths and delays. It contains one or more gate instances. Gate instances include an optional instance name and a required terminal connection list. The terminal connection list specifies how the gate or switch connects to other components in the model. All the instances contained in a gate or switch declaration have the same output strengths and delays.
Gate and Switch Level Modeling
Gate and Switch Declaration Syntax

Syntax 6-1 presents the gate or switch declaration syntax.

```
<gate_declaration>
   ::=<GATETYPE><drive_strength>?<delay>?<gate_instance>
      <,<gate_instance>>* ;

<GATETYPE> is one of the following keywords:
   and nand or nor xor xnor buf bufif0 bufif1 not notif0 notif1
   pulldown pullup nmos rnmos pmos rpmos cmos rcmos tran
   rtran tranif0 tranif1 rtranif0 rtranif1

<drive_strength>
   ::= ( <STRENGTH0> , <STRENGTH1> )
   ||= ( <STRENGTH1> , <STRENGTH0> )

<delay>
   ::= # <number>
   ||= # <identifier>
   ||= # ( <mintypmax_expression> <,<mintypmax_expression>>? <,<mintypmax_expression>>? )

<gate_instance>
   ::= <name_of_gate_instance>? ( <terminal> <,<terminal>>* )

<name_of_gate_instance>
   ::= <IDENTIFIER>

<terminal>
   ::= <IDENTIFIER>
   ||= <expression>
```

Syntax 6-1: Syntax for gate instantiation

This section describes the following parts of a gate or switch declaration:

- the keyword that names the type of gate or switch primitive
- the drive strength specification
- the delay specification
- the identifier that names each gate or switch instance in gate or switch declarations
- the terminal connection list in primitive gate or switch instances
The gate type specification

A gate declaration begins with the <GATETYPE> keyword. The keyword specifies the gate or switch primitive that is used by the instances that follow in the declaration. Table 6-1 lists the keywords that can begin a gate or switch declaration.

<table>
<thead>
<tr>
<th>Gate Type Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>and</td>
</tr>
</tbody>
</table>

Table 6-1: Keywords for the <GATETYPE> syntax item

Explanations of the keywords in Table 6-1 begin in Section 6.2.

The drive strength specification

The drive strength specifications specify the strengths of the values on the output terminals of the instances in the gate declaration. It is possible to specify the strength of the output signals from the gate primitives in Table 6-2.

<table>
<thead>
<tr>
<th>Gate Types That Support Driving Strength</th>
</tr>
</thead>
<tbody>
<tr>
<td>and</td>
</tr>
</tbody>
</table>

Table 6-2: Gate types that accept strength specifications

The drive strength specification in Syntax 6-1 has two parts. A gate declaration must contain both parts or no parts, with the exception of pullup and pulldown sources. One of the parts specifies the strength of signals with a value of 1, and the other specifies the strength of signals with a value of 0.
The STRENGTH1 specification, which specifies the strength of an output signal with a value of 1, is one of the following keywords:

```
supply1 strong1 pull1 weak1 highz1
```

Specifying highz1 causes the gate to output a logic value of Z in place of a 1.

The STRENGTH0 specification, which specifies the strength of an output signal with a value of 0, is one of the following:

```
supply0 strong0 pull0 weak0 highz0
```

Specifying highz0 causes the gate to output a logic value of Z in place of a 0.

The strength specifications must follow the gate type keyword and precede any delay specification. The STRENGTH0 specification can precede or follow the STRENGTH1 specification. In the absence of a strength specification, the instances have the default strengths strong1 and strong0.

The strength specifications, (highz0, highz1) and (highz1, highz0), are invalid and produce the following compiler error message:

```
Error! Illegal strength specification
```

The following example shows a drive strength specification in a declaration of an open collector nor gate:

```
nor(highz1, strong0)(out1, in1, in2);
```

In this example, the nor gate outputs a Z in place of a 1.

Sections 6.10 through 6.15 discuss logic strength modeling in more detail.

### The delay specification

The delay specifies the propagation delay through the gates and switches in a declaration. Gates and switches in declarations with no delay specification have no propagation delay. A delay specification can contain up to three delay values, depending on its gate type. Section 6.2 begins discussions of each type of gate that detail the applicable delays. Section 6.16 discusses delays in more detail. The pullup and pulldown source declarations do not include delay specifications.
The primitive instance identifier

The `<IDENTIFIER>` in Syntax 6-1 is an optional name given to a gate or switch instance. The name is useful in tracing the operation of the circuit during debugging. Verilog-XL can generate names for unnamed gate instances in the source description. See Section 12.6 for information about automatic naming. Compiler directives discussed in Section 6.17 remove optional gate and net names to reduce virtual memory requirements at simulation time.

Primitive instance connection list

The `<terminal>`s at the end of Syntax 6-1 are the terminal list. The terminal list describes how the gate or switch connects to the rest of the model. The gate or switch type limits these expressions. The output or bidirectional terminals always come first in the terminal list, followed by the input terminals.

6.2 and, nand, nor, or, xor, and xnor Gates

Declarations of these gates begin with one of these keywords:

```
and    nand    nor    or    xor    xnor
```

The delay specification can be zero, one, or two delays. If there is no delay, there is no delay through the gate. One delay specifies the delays for all output transitions. If the specification contains two delays, the first delay determines the rise delay, the second delay determines the fall delay, and the smaller of the two delays applies to transitions to X.

These six gates have one output and one or more inputs. The first terminal in the terminal list connects to the gate’s output and all other terminals connect to its inputs.
The truth tables for these gates, showing the result of two input values, appear in Table 6-3.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>x</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>and</td>
<td>0</td>
<td>1</td>
<td>x</td>
<td>z</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>x</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>z</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>x</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>nand</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>x</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>z</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>x</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>or</td>
<td>0</td>
<td>1</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>1</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>z</td>
<td>x</td>
<td>1</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>x</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>nor</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>x</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>0</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>z</td>
<td>x</td>
<td>0</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>x</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>xor</td>
<td>0</td>
<td>1</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>z</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>x</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>xnor</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>x</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>z</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

Table 6-3: Logic tables for and, nand, or, nor, xor, and xnor gates

Versions of these six gates having more than two inputs behave identically with cascaded 2-input gates in producing logic results, but the number of inputs does not alter propagation delays.

The following example declares a two input and gate:

```
and (out, in1, in2);
```

The inputs are in1 and in2. The output is out.
6.3 **buf and not Gates**

Declarations of these gates begin with one of the following keywords:

```
buf
not
```

The delay specification can be zero, one, or two delays. If there is no delay, there is no delay through the gate. One delay specifies the delays for all output transitions. If the specification contains two delays, the first delay determines the rise delay, the second delay determines the fall delay, and the smaller of the two delays applies to transitions to X.

These two gates have one input and one or more outputs. The last terminal in the terminal list connects to the gate’s input, and the other terminals connect outputs.

Truth tables for versions of these gates with one input and one output appear in Table 6-4.

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>z</td>
<td>x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>z</td>
<td>x</td>
</tr>
</tbody>
</table>

*Table 6-4: Logic tables for buf and not gates*

The following example declares a two output buf:

```plaintext
buf (out1, out2, in);
```

The input is `in`. The outputs are `out1` and `out2`. 
6.4 bufif1, bufif0, notif1, and notif0 Gates

Declarations of these gates begin with one of the following keywords:

bufif0  bufif1  notif1  notif0

A strength specification follows the keyword and a delay specification follows the strength specification. The next item is the optional identifier. A terminal list completes the declaration.

These four gates model three-state drivers. In addition to values of 1 and 0, these gates output Z.

The delay specification can be zero, one, two, or three delays. If there is no delay, there is no delay through the gate. One delay specifies the delay of all transitions. If the specification contains two delays, the first delay determines the rise delay, the second delay determines the fall delay, and the smaller of the two delays specifies the delay of transitions to X and Z. If the specification contains three delays, the first delay determines the rise delay, the second delay determines the fall delay, the third delay determines the delay of transitions to Z, and the smallest of the three delays applies to transitions to X.

Some combinations of data input values and control input values cause these gates to output either of two values, without a preference for either value. These gates’ logic tables include two symbols representing such unknown results. The symbol L represents a result which has a value of 0 or Z. The symbol H represents a result which has a value of 1 or Z. Delays on transitions to H or L are the same as delays on transitions to X.

These four gates have one output, one data input, and one control input. The first terminal in the terminal list connects to the output, the second connects to the data input, and the third connects to the control input.
Table 6-5 presents these gates' logic tables:

<table>
<thead>
<tr>
<th></th>
<th>bufif0 CONTROL</th>
<th></th>
<th>bufif1 CONTROL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 1 x z</td>
<td></td>
<td>0 1 x z</td>
</tr>
<tr>
<td>D</td>
<td>0  z L L</td>
<td>D</td>
<td>0   z L L</td>
</tr>
<tr>
<td>A</td>
<td>1  z H H</td>
<td>A</td>
<td>1   z H H</td>
</tr>
<tr>
<td>TA</td>
<td>x  x z x x</td>
<td>TA</td>
<td>x   x z x x</td>
</tr>
<tr>
<td>z</td>
<td>x  x z x x</td>
<td>z</td>
<td>x   x z x x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>notif0 CONTROL</th>
<th></th>
<th>notif1 CONTROL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 1 x z</td>
<td></td>
<td>0 1 x z</td>
</tr>
<tr>
<td>D</td>
<td>0 1 z H H</td>
<td>D</td>
<td>0   z L L</td>
</tr>
<tr>
<td>A</td>
<td>1 0 z L L</td>
<td>A</td>
<td>1   z 0 L L</td>
</tr>
<tr>
<td>TA</td>
<td>x  x z x x</td>
<td>TA</td>
<td>x   x z x x</td>
</tr>
<tr>
<td>z</td>
<td>x  x z x x</td>
<td>z</td>
<td>x   x z x x</td>
</tr>
</tbody>
</table>

*Table 6-5: Logic tables for bufif0, bufif1, notif0, and notif1 gates*

The following example declares a bufif1:

```plaintext
bufif1 (outw, inw, controlw);
```

The output is `outw`, the input is `inw`, and the control is `controlw`.

### 6.5 MOS Switches

Models of MOS networks consist largely of the following four primitive types:

```
nmos  pmos  rnmos  rpmos
```
The `pmos` keyword stands for PMOS transistor and the `nmos` keyword stands for NMOS transistor. PMOS and NMOS transistors have relatively low impedance between their sources and drains when they conduct. The `rpmos` keyword stands for resistive PMOS transistor and the `rnmos` keyword stands for resistive NMOS transistor. Resistive PMOS and resistive NMOS transistors have significantly higher impedance between their sources and drains when they conduct than PMOS and NMOS transistors have. The load devices in static MOS networks are examples of `rpmos` and `rnmos` gates. These four gate types are unidirectional channels for data similar to the `bufif` gates.

Declarations of these gates begin with one of the following keywords:

```
  pmos  nmos  rpmos  rnmos
```

A delay specification follows the keyword. The next item is the optional identifier. A terminal list completes the declaration.

The delay specification can be zero, one, two, or three delays. If there is no delay, there is no delay through the switch. A single delay determines the delay of all output transitions. If the specification contains two delays, the first delay determines the rise delay, the second delay determines the fall delay, and the smaller of the two delays specifies the delay of transitions to `Z` and `X`. If there are three delays, the first delay specifies the rise delay, the second delay specifies the fall delay, the third delay determines the delay of transitions to `Z`, and the smallest of the three delays applies to transitions to `X`. Delays on transitions to `H` and `L` are the same as delays on transitions to `X`.

These four switches have one output, one data input, and one control input. The first terminal in the terminal list connects to the output, the second terminal connects to the data input, and the third terminal connects to the control input.

The `nmos` and `pmos` switches pass signals from their inputs and through their outputs with a change in the signals' strengths in only one case, discussed in Section 6.13. The `rnmos` and `rpmos` gates reduce the strength of signals that propagate through them, as discussed in Section 6.14.

Some combinations of data input values and control input values cause these switches to output either of two values, without a preference for either value. These switches' logic tables include two symbols representing such unknown results. The symbol `L` represents a result which has a value of 0 or `Z`. The symbol `H` represents a result which has a value of 1 or `Z`. 

```
Table 6-6 presents these switches' logic tables:

<table>
<thead>
<tr>
<th>pmos</th>
<th>CONTROL</th>
<th>nmos</th>
<th>CONTROL</th>
</tr>
</thead>
<tbody>
<tr>
<td>rpmos</td>
<td>0 1 x z</td>
<td>rpmos</td>
<td>0 1 x z</td>
</tr>
<tr>
<td>D A T</td>
<td>0 z L L</td>
<td>D A T</td>
<td>z 0 L L</td>
</tr>
<tr>
<td>x z</td>
<td>1 z H H</td>
<td>x z</td>
<td>z 1 H H</td>
</tr>
<tr>
<td>z z</td>
<td>x z</td>
<td>z z</td>
<td>z z z z</td>
</tr>
<tr>
<td>z z z z</td>
<td></td>
<td>z z z z</td>
<td></td>
</tr>
</tbody>
</table>

Table 6-6: Logic tables for pmos, rpmos, nmos, and rnmos gates

The following example declares a pmos switch:

```plaintext
pmos (out, data, control);
```

The output is `out`, the data input is `data`, and the control input is `control`.

### 6.6 Bidirectional Pass Switches

Declarations of bidirectional switches begin with one of the following keywords:

- `tran`
- `tranif1`
- `tranif0`
- `rtran`
- `rtranif1`
- `rtranif0`

A delay specification follows the keywords in declarations of `tranif1`, `tranif0`, `rtranif1`, and `rtranif0`; the `tran` and `rtran` devices do not take delays. The next item is the optional identifier. A terminal list completes the declaration.

The delay specifications for `tranif1`, `tranif0`, `rtranif1`, and `rtranif0` devices can be zero, one, or two delays. If there is no delay, the device has no turn-on or turn-off delay. If the specification contains one delay, that delay determines both turn-on and turn-off delays. If there are two delays, the first delay specifies the turn-on delay, and the second delay specifies the turn-off delay.
These six devices do not delay signals propagating through them. When these devices are turned off they block signals, and when they are turned on they pass signals.

The tranif1, tranif0, rtranif1, and rtranif0 devices have three items in their terminal lists. Two are bidirectional terminals that conduct signals to and from the devices, and the other terminal connects to a control input. The terminals connected to inouts precede the terminal connected to the control input in the terminal list.

The tran and rtran devices have terminal lists containing two bidirectional terminals.

The bidirectional terminals of all six of these devices connect only to scalar nets or bit-selects of expanded vector nets.

The tran, tranif0, and tranif1 devices pass signals with an alteration in their strength in only one case, discussed in Section 6.13. The rtran, rtranif0, and rtranif1 devices reduce the strength of signals passing through them according to rules discussed in Section 6.14.

The following example declares a tranif1:

```
tranif1 (inout1, inout2, control);
```

The bidirectional terminals are inout1 and inout2. The control input is control.

### 6.7 cmos Gates

Declarations of these gates begins with one of these keywords:

```
cmos  rcmos
```

The delay specification can be zero, one, two, or three delays. If there is no delay, there is no delay through the gate. A single delay specifies the delay for all transitions. If the specification contains two delays, the first delay determines the rise delay, the second delay determines the fall delay, and the smaller of the two delays is the delay of transitions to Z and X. If the specification contains three delays, the first delay controls rise delays, the second delay controls fall delays, the third delay controls transitions to Z, and the smallest of the three delays applies to transitions to X. Delays in transitions to H or L are the same as delays in transitions to X.
The \texttt{cmos} and \texttt{rcmos} gates have a data input, a data output, and two control inputs. In the terminal list, the first terminal connects to the data output, the second connects to the data input, the third connects to the n-channel control input, and the last connects to the p-channel control input.

The \texttt{cmos} gate passes signals with an alteration in their strength in only one case, discussed in Section 6.13. The \texttt{rcmos} gate reduces the strength of signals passing through it according to rules that appear in Section 6.14.

The \texttt{cmos} gate is the combination of a \texttt{pmos} gate and an \texttt{nmos} gate. The \texttt{rcmos} gate is the combination of an \texttt{rpmos} gate and an \texttt{rnmos} gate. The combined gates in these configurations share data input and data output terminals, but they have separate control inputs. These combined configurations simulate more efficiently than the equivalent networks of two gates.

The equivalence of the \texttt{cmos} gate to the pairing of an \texttt{nmos} gate and a \texttt{pmos} gate is detailed in the following explanation:

\begin{verbatim}
cmos (w, datain, ncontrol, pcontrol);
\end{verbatim}

is equivalent to:

\begin{verbatim}
nmos (w, datain, ncontrol);
\end{verbatim}

\begin{verbatim}
pmos (w, datain, pcontrol);
\end{verbatim}

\section*{6.8 \textbf{pullup and pulldown} Sources}

Declarations of these sources begin with one of the following keywords:

\begin{verbatim}
pullup pulldown
\end{verbatim}

A strength specification follows the keyword and an optional identifier follows the strength specification. A terminal list completes the declaration.

A \texttt{pullup} source places a logic value of 1 on the nets listed in its terminal list. A \texttt{pulldown} source places a logic value of 0 on the nets listed in its terminal list. The signals that these sources place on nets have pull strength in the absence of a strength specification. There are no delay specifications for these sources because the signals they place on nets continue throughout simulation without variation.
The following example declares two pullup instances:

```plaintext
pullup (strong1, strong0)(neta),(netb);
```

In this example, one gate instance drives neta, the other drives netb.

### 6.9 Implicit Net Declarations

Including a previously unused identifier in a terminal list implicitly declares a new net of the `wire` type with zero delay.

If the `wire` type is unsuitable for implicitly declared nets, the compiler directive `default_nettype` can change the type acquired by implicitly declared nets.

The following is the directive’s syntax:

```plaintext`
default_nettype <type_of_net>
```

The first character in the directive is an accent grave.

The `<type_of_net>` can be one of the following net types:

<table>
<thead>
<tr>
<th>Net Type</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>wire</td>
<td>tri</td>
</tr>
<tr>
<td></td>
<td>tri0</td>
</tr>
<tr>
<td>wand</td>
<td>triand</td>
</tr>
<tr>
<td></td>
<td>tri1</td>
</tr>
<tr>
<td>wor</td>
<td>trior</td>
</tr>
<tr>
<td></td>
<td>trireg</td>
</tr>
</tbody>
</table>

This directive must occur outside of module definitions. All the modules between any two `default_nettype` directives are affected by the first `default_nettype` directive. The effect of the directive crosses source file boundaries in the order in which they appear on the command line. The `resetall` compiler directive ends the effect of a preceding `default_nettype` directive. A source description can contain any number of these directives. Implicit nets are of type `wire` in the absence of a `default_nettype` directive.
Each implicitly declared net must connect to one or more of the following:

- gate output
- tranif bidirectional terminal
- module output port

If an implicitly declared net does not connect to one of the listed items, the compiler produces an error message with this form:

"warning! implicit wire (<name>) has no fanin"

If nothing drives a net, Verilog-XL assigns a value of Z to the net.

6.10 Logic Strength Modeling

The Verilog HDL provides for accurate modeling of signal contention, bidirectional pass gates, resistive MOS devices, dynamic MOS, charge sharing, and other technology dependent network configurations by allowing scalar net signal values to have a full range of unknown values and different levels of strength or combinations of levels of strength. This multiple level logic strength modeling resolves combinations of signals into known or unknown values to represent the behavior of hardware with maximum accuracy.

A strength specification has two components:

1. the strength of the 0 portion of the net value, designated <STRENGTH0> in Syntax 6-1
2. the strength of the 1 portion of the net value, designated <STRENGTH1> in Syntax 6-1

Despite this division of the strength specification, it is helpful to consider strength as a property occupying regions of a continuum in order to predict the results of combinations of signals.
Gate and Switch Level Modeling
Logic Strength Modeling

Table 6-7 demonstrates the continuum of strengths. The left column lists the keywords that specify strength levels of trireg or gate output. The middle column in Table 6-7 shows relative strength levels correlated with the keywords. The abbreviations Verilog-XL reports are in the right column in Table 6-7.

<table>
<thead>
<tr>
<th>strength name</th>
<th>strength level</th>
<th>abbreviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>supply0</td>
<td>7</td>
<td>Su0</td>
</tr>
<tr>
<td>strong0</td>
<td>6</td>
<td>St0</td>
</tr>
<tr>
<td>pull0</td>
<td>5</td>
<td>Pu0</td>
</tr>
<tr>
<td>large0</td>
<td>4</td>
<td>La0</td>
</tr>
<tr>
<td>weak0</td>
<td>3</td>
<td>We0</td>
</tr>
<tr>
<td>medium0</td>
<td>2</td>
<td>Me0</td>
</tr>
<tr>
<td>small0</td>
<td>1</td>
<td>Sm0</td>
</tr>
<tr>
<td>highz0</td>
<td>0</td>
<td>HiZ0</td>
</tr>
<tr>
<td>highz1</td>
<td>0</td>
<td>HiZ1</td>
</tr>
<tr>
<td>small1</td>
<td>1</td>
<td>Sm1</td>
</tr>
<tr>
<td>medium1</td>
<td>2</td>
<td>Me1</td>
</tr>
<tr>
<td>weak1</td>
<td>3</td>
<td>We1</td>
</tr>
<tr>
<td>large1</td>
<td>4</td>
<td>La1</td>
</tr>
<tr>
<td>pull1</td>
<td>5</td>
<td>Pul</td>
</tr>
<tr>
<td>strong1</td>
<td>6</td>
<td>St1</td>
</tr>
<tr>
<td>supply1</td>
<td>7</td>
<td>Su1</td>
</tr>
</tbody>
</table>

*Table 6-7: Strength levels for scalar net signal values*

In the preceding table, there are four driving strengths:

- supply
- strong
- pull
- weak

Signals with driving strengths propagate from gate outputs and continuous assignment outputs.
In the preceding table, there are three charge storage strengths:

large  medium  small

Signals with the charge storage strengths originate in the trireg net type.

It is possible to think of the strengths of signals in the preceding table as locations on the scale in Figure 6-1.

<table>
<thead>
<tr>
<th>0 strength</th>
<th>1 strength</th>
</tr>
</thead>
<tbody>
<tr>
<td>Su0 6 5 4 3 2 1 0</td>
<td>St0 0 1 2 3 4 5 6 7</td>
</tr>
<tr>
<td>Pu0 La0 We0 Me0 Sm0 HiZ0</td>
<td>Pu1 St1 Su1</td>
</tr>
</tbody>
</table>

*Figure 6-1: Scale of strengths*

Discussions of signal combinations later in this document will employ graphics similar to Figure 6-1.

A net signal can have one or more strength levels associated with it. If a net signal value is known, its strength levels are all in either the 0 strength part of the scale represented by Figure 6-1, or they are all in its 1 strength part. If a net signal value is unknown, it has strength levels in both the 0 strength and the 1 strength parts. A signal with a value of Z has a strength level only in the HiZ0 or HiZ1 subdivisions of the scale.

### 6.11
#### Strengths and Values of Combined Signals

In addition to a value, a signal has either a single unambiguous strength level or it has an ambiguous strength, consisting of more than one level. When signals combine, their strengths and values determine the strength and value of the resulting signal in accord with the principles in the four sections that follow.

### 6.11.1
#### Combined Signals of Unambiguous Strength

This section deals with combinations of signals in which each signal has a known value and a single strength level.
If two signals of unequal strength combine in a wired net configuration, the stronger signal is the result. This case appears in Figure 6-2.

In Figure 6-2, the numbers in parentheses indicate the relative strengths of the signals. The combination of a pull 1 and a strong 0 results in a strong 0, which is the stronger of the two signals. The combination of two signals of like value results in the same value with the greater of the two strengths.

The combination of signals identical in strength and value results in the same signal.

The combination of signals with unlike values and the same strength has three possible results. Two of the results occur in the presence of wired logic and the third occurs in its absence. Section 6.11.4 discusses wired logic. The result in the absence of wired logic is the subject of the first figure in the next section.
6.11.2
Ambiguous Strengths: Sources and Combinations

The classifications of signals possessing ambiguous strengths are the following:

- signals with known values and multiple strength levels
- signals with a value of \( X \), which have strength levels consisting of subdivisions of both the strength 1 and the strength 0 parts of the scale of strengths in Figure 6-1
- signals with a value of \( L \), which have strength levels that consist of high impedance joined with strength levels in the 0 strength part of the scale of strengths in Figure 6-1
- signals with a value of \( H \), which have strength levels that consist of high impedance joined with strength levels in the 1 strength part of the scale of strengths in Figure 6-1

Many configurations can produce signals of ambiguous strength. When two signals of equal strength and opposite value combine, the result has a value of \( X \) and the strength levels of both signals and all the smaller strength levels. Figure 6-3 shows the combination of a weak signal with a value of 1 and a weak signal with a value of 0 yielding a signal with weak strength and a value of \( X \).

![Figure 6-3: Combination of signals of equal strength and opposite values](image)

This signal is described in Figure 6-4.

![Figure 6-4: Weak X signal strength](image)
An ambiguous signal strength can be a range of possible values. An example is the strength of the output from the tristate drivers with unknown control inputs in Figure 6-5.

![Figure 6-5: Bufifs with control inputs of X](image)

The output of the `bufif1` in Figure 6-5 is a strong H, composed of the range of values described in Figure 6-6.

<table>
<thead>
<tr>
<th>0 strength</th>
<th>1 strength</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 6 5 4 3 2 1 0</td>
<td>0 1 2 3 4 5 6 7</td>
</tr>
<tr>
<td>Su0 St0 Pu0 La0 We0 Me0 Sm0 HiZ0</td>
<td>HiZ1 Sml Mel Wel Lal Pul St1 Su1</td>
</tr>
</tbody>
</table>

![Figure 6-6: Strong H range of values](image)

The output of the `bufif0` in Figure 6-5 is a weak L, composed of the range of values described in Figure 6-7.
The combination of two signals of ambiguous strength results in a signal of ambiguous strength. The resulting signal has a range of strength levels that includes the strength levels in its component signals. The combination of outputs from two tristate drivers with unknown control inputs, shown in Figure 6-8, is an example.

In Figure 6-8, the combination of signals of ambiguous strengths produces a range which includes the extremes of the signals and all the strengths between them, as described in Figure 6-9.
The result is an X because values of both H and L are being driven onto the output net with ambiguous strengths. The number 35, which precedes the X, is a concatenation of two digits. The first is the digit 3, which corresponds to the highest strength level for the result’s value of 0. The second digit, 5, corresponds to the highest strength level for the result’s value of 1.

Switch networks can produce a range of strengths of the same value, such as the signals from the upper and lower configurations in Figure 6-10.

![Figure 6-9: An unknown signal's range of strengths](image)

![Figure 6-10: Ambiguous strengths from switch networks](image)
In Figure 6-10, the upper combination of a register, a gate controlled by a register of unspecified value, and a pullup produces a signal with a value of 1 and a range of strengths (651) described in Figure 6-11.

In Figure 6-10, replacing the pullup in the lower configuration with a supply0 would change the range of the result to the range (StX) described in Figure 6-14.
The range in Figure 6-14 is strong \( X \), because it is unknown and both of its components’ extremes are strong. The extreme of the output of the lower configuration is strong because the lower \texttt{pmos} reduces the strength of the \texttt{supply0} signal. Section 6.13 discusses this modeling feature.

Logic gates produce results with ambiguous strengths as well as tristate drivers. Such a case appears in Figure 6-15.

In Figure 6-15, register \( b \) has an unspecified value, so its input to the upper and gate is strong \( X \). The upper and gate has a strength specification including \texttt{highz0}. The signal from the upper and gate is a strong \( H \) composed of the values described in Figure 6-16.
HiZ0 is part of the result, because the strength specification for the gate in question specified that strength for an output with a value of 0. A strength specification other than high impedance for the 0 value output results in a gate output of X. The output of the lower and gate is a weak 0 described in Figure 6-17.

<table>
<thead>
<tr>
<th>0 strength</th>
<th>1 strength</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Su0</td>
<td>7 HiZ1</td>
</tr>
<tr>
<td>6 St0</td>
<td>6 Sm1</td>
</tr>
<tr>
<td>5 Pu0</td>
<td>5 Me1</td>
</tr>
<tr>
<td>4 La0</td>
<td>4 We1</td>
</tr>
<tr>
<td>3 Me0</td>
<td>3 Sm0</td>
</tr>
<tr>
<td>2 We0</td>
<td>2 HiZ0</td>
</tr>
<tr>
<td>1 Sm0</td>
<td>1 Me0</td>
</tr>
<tr>
<td>0 HiZ0</td>
<td>0 Su0</td>
</tr>
</tbody>
</table>

*Figure 6-17: Weak 0*

When the signals combine, the result is the range (36X) described in Figure 6-18.

<table>
<thead>
<tr>
<th>0 strength</th>
<th>1 strength</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Su0</td>
<td>7 HiZ1</td>
</tr>
<tr>
<td>6 St0</td>
<td>6 Sm1</td>
</tr>
<tr>
<td>5 Pu0</td>
<td>5 Me1</td>
</tr>
<tr>
<td>4 La0</td>
<td>4 We1</td>
</tr>
<tr>
<td>3 Me0</td>
<td>3 Sm0</td>
</tr>
<tr>
<td>2 We0</td>
<td>2 HiZ0</td>
</tr>
<tr>
<td>1 Sm0</td>
<td>1 Me0</td>
</tr>
<tr>
<td>0 HiZ0</td>
<td>0 Su0</td>
</tr>
</tbody>
</table>

*Figure 6-18: Ambiguous strength in combined gate signals*

This figure presents the combination of an ambiguous signal and an unambiguous signal. Such combinations are the topic of Section 6.11.3.

### 6.11.3 Ambiguous Strength Signals and Unambiguous Signals

The combination of a signal with unambiguous strength and known value with another signal of ambiguous strength presents several possible cases. To understand a set of rules governing this type of combination, it is necessary to consider the strength levels of the ambiguous strength signal separately from each other and relative to the unambiguous strength signal. When a signal of known value and unambiguous strength combines with a component of a signal of ambiguous strength, these are the effects:

**Rule 1:**

The strength levels of the ambiguous strength signal that are greater than the strength level of the unambiguous signal remain in the result.
Rule 2:
The strength levels of the ambiguous strength signal that are smaller than or equal to the strength level of the unambiguous signal disappear from the result, subject to Rule 3.

Rule 3:
If the operation of Rule 1 and Rule 2 results in a gap in strength levels because the signals are of opposite value, the signals in the gap are part of the result.

The following figures show some applications of the rules.

Figure 6-19: Elimination of strength levels

In Figure 6-19, the strength levels in the ambiguous strength signal that are smaller than or equal to the strength level of the unambiguous strength signal disappear from the result, demonstrating Rule 2.
In Figure 6-20, Rule 1, Rule 2, and Rule 3 apply. The strength levels of the ambiguous strength signal that are of opposite value and lesser strength than the unambiguous strength signal disappear from the result. The strength levels in the ambiguous strength signal that are less than the strength level of the unambiguous strength signal, and of the same value, disappear from the result. The strength level of the unambiguous strength signal and the greater extreme of the ambiguous strength signal define a range in the result.
In Figure 6-21, Rule 1 and Rule 2 apply. The strength levels in the ambiguous strength signal that are less than the strength level of the unambiguous strength signal disappear from the result. The strength level of the unambiguous strength signal and the strength level at the greater extreme of the ambiguous strength signal define a range in the result.
In Figure 6-22, Rule 1, Rule 2, and Rule 3 apply. The greater extreme of the range of strengths for the ambiguous strength signal is larger than the strength level of the unambiguous strength signal. The result is a range defined by the greatest strength in the range of the ambiguous strength signal and by the strength level of the unambiguous strength signal.

### 6.11.4 Wired Logic Net Types

The net types triand, wand, trior, and wor resolve conflicts when multiple drivers are at the same level of strength. These net types resolve signal values by treating signals as inputs of logic functions.
For example, consider the combination of two signals of unambiguous strength in Figure 6-23.

The combination of the signals in Figure 6-23, using wired AND logic, produces a result with the same value as the result produced by an AND gate with the two signals' values as its inputs. The combination of signals using wired OR logic produces a result with the same value as the result produced by an OR gate with the two signals' values as its inputs. The strength of the result is the same as the strength of the combined signals in both cases. If the value of the upper signal changes so that both signals in Figure 6-23 possess a value of 1, then the results of both types of logic have a value of 1.

When ambiguous strength signals combine in wired logic, it is necessary to consider the results of all combinations of each of the strength levels in the first signal with each of the strength levels in the second signal, as shown in Figure 6-24.
Gate and Switch Level Modeling
Strengths and Values of Combined Signals

The combinations of strength levels for AND logic appear in the following chart:

<table>
<thead>
<tr>
<th>Signal 1</th>
<th>Signal 2</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strength</td>
<td>Value</td>
<td>Strength</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>5</td>
</tr>
</tbody>
</table>

The result is the following signal:

The combinations of strength levels for OR logic appear in the following chart:

<table>
<thead>
<tr>
<th>Signal 1</th>
<th>Signal 2</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strength</td>
<td>Value</td>
<td>Strength</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>5</td>
</tr>
</tbody>
</table>

The result is the following signal:

Figure 6-24: Wired logic and ambiguous strengths
6.12 Mnemonic Format

Trace messages giving signal strength information are compatible with the %v format option in the $display system task. See Section 21.2 for more information on this mnemonic strength notation.

6.13 Strength Reduction by Non-Resistive Devices

The nmos, pmos, and cmos gates pass through the strength from the data input to the output, except that a supply strength is reduced to a strong strength.

The tran, tranif0, and tranif1 gates do not affect signal strength across the bidirectional terminals, except that a supply strength is reduced to a strong strength.

6.14 Strength Reduction by Resistive Devices

The rnmos, rpmos, rcmos, rtran, rtranif1, and rtranif0 devices reduce the strength of signals that pass through them according to Table 6-8.

<table>
<thead>
<tr>
<th>input strength</th>
<th>reduced strength</th>
</tr>
</thead>
<tbody>
<tr>
<td>supply drive</td>
<td>pull drive</td>
</tr>
<tr>
<td>strong drive</td>
<td>pull drive</td>
</tr>
<tr>
<td>pull drive</td>
<td>weak drive</td>
</tr>
<tr>
<td>weak drive</td>
<td>medium capacitor</td>
</tr>
<tr>
<td>large capacitor</td>
<td>medium capacitor</td>
</tr>
<tr>
<td>medium capacitor</td>
<td>small capacitor</td>
</tr>
<tr>
<td>small capacitor</td>
<td>small capacitor</td>
</tr>
<tr>
<td>high impedance</td>
<td>high impedance</td>
</tr>
</tbody>
</table>

Table 6-8: Strength reduction rules
6.15 Strengths of Net Types

The tri0, tri1, supply0, and supply1 net types generate signals with specific strength levels. The trireg declaration can specify either of two signal strength levels other than a default strength level.

6.15.1 tri0 and tri1 Net Strengths

The tri0 net type models a net connected to a resistive pulldown device. Its signal has a value of 0 and a pull strength in the absence of an overriding source. The tri1 net type models a net connected to a resistive pullup device: its signal has a value of 1 and a pull strength in the absence of an overriding source.

6.15.2 trireg Strength

The trireg net type models charge storage nodes. The strength of the drive resulting from a trireg net that is in the charge storage state (that is, a driver charged the net and then went to high impedance) is one of these three strengths: large, medium, or small. The specific strength associated with a particular trireg net is specified by the user in the net declaration. The default is medium. The syntax of this specification is described in Section 3.4.1.

6.15.3 supply0 and supply1 Net Strengths

The supply0 net type models ground connections. The supply1 net type models connections to power supplies. The supply0 and supply1 net types have supply driving strengths.

6.16 Gate and Net Delays

Gate and net delays provide a means of accurately describing delays through a circuit. The gate delays specify the signal propagation delay from any gate input to the gate output. Up to three values per output can be specified. The descriptions in this chapter of each gate type give the rules for which gates can take how many delays—see Section 6.2 through Section 6.7.
Net delays refer to the time it takes from any driver on the net changing value to the time when the net value is updated and propagated further. Up to three delay values per net can be specified.

**Please note:** Verilog-XL treats two nets connected by a bidirectional switch as one net and simulates the delays on both nets in parallel.

For both gates and nets, the default delay is zero when no delay specification is given. When one delay value is given, then this value is used for all propagation delays associated with the gate or net. The following is an example of a delay specification with one delay:

```verilog
and #(10) (out, in1, in2);
```

The following is an example of a delay specification with two delays:

```verilog
and #(10, 12) (out, in1, in2);
```

When two delays are given, the first specifies the rise delay and the second specifies the fall delay. The delay when the signal changes to high impedance or to unknown is the lesser of the two delay values.

The following is an example of a delay specification with three delays:

```verilog
and #(10, 12, 11) (out, in1, in2);
```

For a three delay specification:

- the first delay refers to the transition to the 1 value (rise delay)
- the second delay refers to the transition to the 0 value (fall delay)
- the third delay refers to the transition to the high impedance value

When a value changes to the unknown (X) value, the delay is the smallest of the three delays.
Table 6-9 summarizes the from-to propagation delay choice for the two and three delay specifications.

<table>
<thead>
<tr>
<th>from value:</th>
<th>to value:</th>
<th>delay used if there are:</th>
<th>2 delays</th>
<th>3 delays</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>d1</td>
<td>d1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>x</td>
<td>min(d1, d2)</td>
<td>min(d1, d2, d3)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>z</td>
<td>min(d1, d2)</td>
<td>d3</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>d2</td>
<td>d2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td>min(d1, d2)</td>
<td>min(d1, d2, d3)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>z</td>
<td>min(d1, d2)</td>
<td>d3</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>0</td>
<td>d2</td>
<td>d2</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>1</td>
<td>d1</td>
<td>d1</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>z</td>
<td>min(d1, d2)</td>
<td>d3</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>0</td>
<td>d2</td>
<td>d2</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>1</td>
<td>d1</td>
<td>d1</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>x</td>
<td>min(d1, d2)</td>
<td>min(d1, d2, d3)</td>
<td></td>
</tr>
</tbody>
</table>

*Table 6-9: Rules for propagation delays*

The following example specifies a simple latch module with tri-state outputs, where individual delays are given to the gates. The propagation delay from the primary inputs to the outputs of the module will be cumulative, and depends on the signal path through the network.
Gate and Switch Level Modeling
Gate and Net Delays

Example 6-1: Using delay values

6.16.1 min/typ/max Delays

The syntax for delays on gate primitives (including user-defined primitives), nets, and continuous assignments allows three values each for the rising, falling, and turn-off delays. The minimum, typical, and maximum values for each are specified as constant expressions separated by colons. The following example shows min/typ/max values for rising, falling, and turn-off delays:

```verbatim
module tri_latch(qout, nqout, clock, data, enable);
    output qout, nqout;
    input clock, data, enable;
    tri qout, nqout;
    not #5
        (ndata, data);
    nand #(3, 5)
        (wa, data, clock),
        (wb, ndata, clock);
    nand #(12, 15)
        (q, nq, wa),
        (nq, q, wb);
    bufif1 #(3, 7, 13)
        q_drive (qout, q, enable),
        nq_drive (nqout, nq, enable);
endmodule
```

Example 6-2: Syntax example for delay expressions

```verbatim
module iobuf(io1, io2, dir);
    bufif0 #(5:7:9, 8:10:12, 15:18:21) (io1, io2, dir);
    bufif1 #(6:8:10, 5:7:9, 13:17:19) (io2, io1, dir);
endmodule
```
The syntax for delay controls in procedural statements also allows minimum, typical, and maximum values. These are specified by expressions separated by colons. Example 6-3 illustrates this concept.

```
parameter
  min_hi = 97, typ_hi = 100, max_hi = 107;
reg clk;
always
  begin
    #(95:100:105) clk = 1;
    #(min_hi:typ_hi:max_hi) clk = 0;
  end
```

**Example 6-3: Delay controls in procedural statements**

The delay used during simulation will be one of the three—either minimum, typical, or maximum. One delay choice is used throughout a simulation run; it cannot be changed dynamically.

Selection of which delays will be used is done using one of three command options. The `+maxdelays` option selects all of the maximum delays; the `+typdelays` option selects all of the typical delays; the `+mindelays` option selects all of the minimum delays. For example, the following command line runs Verilog-XL with only the values specified for the maximum delay:

```
verilog source1.v +maxdelays
```

**Please note:** If only one delay is specified, then Verilog-XL uses it regardless of whether minimum, typical, or maximum delays are selected. If more than one delay is desired, then all three delays must be specified; for example, it is not possible to specify minimum and maximum without typical.

**CAUTION**

There is currently no syntax checking on plus command options. Be very careful in specifying them to avoid confusing results. If you misspell "maxdelays", "mindelays" or "typdelays", the option will be ignored.
6.16.2 trireg Net Charge Decay

Like all nets, a trireg declaration’s delay specification can contain up to three delays. The first two delays specify the simulation time that elapses in a transition to the 1 and 0 logic states when the trireg is driven to these states by a driver. The third delay specifies the charge decay time instead of the time that elapses in a transition to the z logic state. The charge decay time specifies the simulation time that elapses between when a trireg’s drivers turn off and when its stored charge can no longer be determined.

A trireg needs no turn-off delay specification because a trireg never makes a transition to the z logic state. When a trireg’s drivers make transitions from the 1, 0, or x logic states to off, the trireg retains the previous 1, 0, or x logic state that was on its drivers. The z value does not propagate from a trireg’s drivers to a trireg. A trireg can only hold a z logic state when z is the trireg’s initial logic state or when it is forced to the z state with a force statement.

A delay specification for charge decay models a charge storage node that is not ideal, a charge storage node whose charge leaks out through its surrounding devices and connections.

This section describes the charge decay process and the delay specification for charge decay.

The charge decay process

Charge decay is the cause of transition of a 1 or 0 that is stored in a trireg to an unknown value (x) after a specified number of time units. The charge decay time is that specified number of time units.

The charge decay process begins when the trireg’s drivers turn off and the trireg starts to hold charge. The charge decay process ends under the following two conditions:

1. The specified number of time units elapse and the trireg makes a transition from 1 or 0 to x.
2. The trireg’s drivers turn on and propagate a 1, 0 or x into the trireg.

When charge decay causes a trireg’s value to change to x, Verilog-XL issues a warning message. This message takes the following form:

```
Warning! Time = simulation_time: Charge on node hierarchical_name_of_trireg has decayed [Verilog-DECAY]
"source_file_name", line_number: trireg_identifier
```

You can tell Verilog-XL not to issue this warning with the $disable_warnings system task.
The delay specification for charge decay time

The third delay in a trireg declaration specifies the charge decay time. A three-valued delay specification in a trireg declaration has the following form:

```
#(d1, d2, d3)
// three delays —
// (rising_delay, falling_delay, charge_decay_time)
```

The specification in a trireg declaration of the charge decay time must be preceded by a rise and fall delay specification. The following example shows a specification of the charge decay time in a trireg declaration:

```
trireg (large) #(0,0,50) cap1;
```

This example declares a trireg with the identifier cap1. This trireg stores a large charge. The delay specifications for the rise delay is 0, the fall delay is 0, and the charge decay time specification is 50 time units.

Please note: A charge decay time is not a propagation delay like a rising delay or a falling delay. A charge decay time greater than 0 does not prevent the acceleration of the trireg.

Example 6-4 presents a source description file that contains a trireg declaration with a charge decay time specification. Figure 6-25 assists you in reading the source description file.

*Figure 6-25: This figure accompanies the example below*
module capacitor;
reg data, gate;

trireg (large) #(0,0,50) cap1;
nmos nmos1 (cap1, data, gate);

initial
begin
  $monitor("%0d data = %v gate = %v cap1 = %v",
           $time, data, gate, cap1);
  data = 1;
  gate = 1;
#10 gate = 0;
#30 gate = 1;
#10 gate = 0;
#100 $finish;
end
endmodule

Example 6-4: trireg with a charge decay
Example 6-5 shows the simulation results of the model in Example 6-4.

<table>
<thead>
<tr>
<th>Time</th>
<th>Data</th>
<th>Gate</th>
<th>Cap1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>St1</td>
<td>St1</td>
<td>St1</td>
</tr>
<tr>
<td>10</td>
<td>St1</td>
<td>St0</td>
<td>La1</td>
</tr>
<tr>
<td>40</td>
<td>St1</td>
<td>St1</td>
<td>St1</td>
</tr>
<tr>
<td>50</td>
<td>St1</td>
<td>St0</td>
<td>La1</td>
</tr>
</tbody>
</table>

Warning! Time = 100: Charge on node capacitor.cap1 has decayed [Verilog-DECAY]

"trireg1.v", 4: cap1
100 data = St1 gate = St0 cap1 = LaX

Example 6-5: Charge decay simulation results

The results show the following sequence of events:

1. At simulation time 0, data drives a strong 1 into trireg cap1.
2. At simulation time 10, gate’s value changes to 0, disconnecting trireg cap1 from data; trireg cap1 enters the capacitive state, storing its value of 1 with a large strength. The charge decay process begins for trireg cap1: its value is scheduled to change to x at simulation time 60.
3. At simulation time 40, gate’s value changes to 1, connecting trireg cap1 to data; trireg cap1 enters the driven state, and data drives a strong 1 into trireg cap1. The charge decay process stops for trireg cap1 because it is no longer in the capacitive state.
4. At simulation time 50, reg gate’s value changes to 0, disconnecting trireg cap1 from reg data again; trireg cap1 enters the capacitive state, storing its value of 1 with a large strength. The charge decay process begins again for trireg cap1; its value is scheduled to change to x at simulation time 100.
5. At simulation time 100, the charge decay process changes the stored value in trireg cap1 from 1 to x.

Please note: Specifying a charge decay time can affect performance. You may see a performance degradation caused by specifying trireg charge decay time in a design—such as a dynamic circuit, whose triregs frequently enter the capacitive state.
6.17 Gate and Net Name Removal

Four compiler directives have been provided that control the removal of gate and/or net names in order to reduce the virtual memory requirements at the gate and switch level. The names are removed from the second and all subsequent module instances so that removing gate and net names saves the most memory in designs containing gate-level modules that are instantiated many times.

The compiler directives are the following:

'\texttt{remove\_gatenames}'

'\texttt{no\_remove\_gatenames}'

'\texttt{remove\_netnames}'

'\texttt{no\_remove\_netnames}'

The first two directives control the removal of gate names, and the latter two control the removal of net names. For both controls, the default is to NOT remove the names.

These directives can only be specified outside modules. The control applies to all modules following a directive until the end of the source description (going across source files if necessary) or until another of these directives is given or until a '\texttt{reset\_all}' directive is given. Any number of these compiler directives can be given in a source description.

The removal of gate names is more useful than the removal of net names because gate names at the present are only used for the tracing of value changes across the gates.

Net names cannot be removed if they have been referenced in a hierarchical name. An example of a hierarchical referencing would be a monitoring task, or nets that will need to be referenced interactively.

As shown in the following partial description, all gate names from modules a and b, and net names from all the instances of module b are removed.
Example 6-6: Gate and net name removal

Note that it is not possible to selectively remove the gate and/or net names from particular instances of a module.