Verilog

- Verilog is an alternative language to VHDL for specifying RTL for logic synthesis
- VHDL similar to Ada programming language in syntax
- Verilog similar to C/Pascal programming language
- VHDL more popular with European companies, Verilog more popular with US companies.
- VHDL more 'verbose' than Verilog.
- Verilog and VHDL do RTL modeling equally well.

VHDL vs. Verilog: Process Block

<table>
<thead>
<tr>
<th>VHDL</th>
<th>Verilog</th>
</tr>
</thead>
<tbody>
<tr>
<td>```vhd</td>
<td>```verl</td>
</tr>
<tr>
<td>```vhd</td>
<td>```verl</td>
</tr>
<tr>
<td>```vhd</td>
<td>```verl</td>
</tr>
</tbody>
</table>

VHDL vs. Verilog: Signal Assignment

<table>
<thead>
<tr>
<th>VHDL</th>
<th>Verilog</th>
</tr>
</thead>
<tbody>
<tr>
<td>```vhd</td>
<td>```verl</td>
</tr>
<tr>
<td>```vhd</td>
<td>```verl</td>
</tr>
<tr>
<td>```vhd</td>
<td>```verl</td>
</tr>
</tbody>
</table>

VHDL vs. Verilog: Interface Declaration

<table>
<thead>
<tr>
<th>VHDL</th>
<th>Verilog</th>
</tr>
</thead>
<tbody>
<tr>
<td>```vhd</td>
<td>```verl</td>
</tr>
<tr>
<td>```vhd</td>
<td>```verl</td>
</tr>
<tr>
<td>```vhd</td>
<td>```verl</td>
</tr>
</tbody>
</table>

VHDL vs. Verilog: Busses

<table>
<thead>
<tr>
<th>VHDL</th>
<th>Verilog</th>
</tr>
</thead>
<tbody>
<tr>
<td>```vhd</td>
<td>```verl</td>
</tr>
<tr>
<td>```vhd</td>
<td>```verl</td>
</tr>
<tr>
<td>```vhd</td>
<td>```verl</td>
</tr>
</tbody>
</table>

A Sample Model (RTL), assignment statement

<table>
<thead>
<tr>
<th>Description</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>```vhd</td>
<td>```verl</td>
</tr>
<tr>
<td>```vhd</td>
<td>```verl</td>
</tr>
<tr>
<td>```vhd</td>
<td>```verl</td>
</tr>
<tr>
<td>```vhd</td>
<td>```verl</td>
</tr>
<tr>
<td>```vhd</td>
<td>```verl</td>
</tr>
<tr>
<td>```vhd</td>
<td>```verl</td>
</tr>
<tr>
<td>```vhd</td>
<td>```verl</td>
</tr>
</tbody>
</table>
module majconc (a,b,c,y);
input a,b,c;
output y;
reg y;
always @(a or b or c)
begin
  y = (a & b) | (a & c) | (b & c);
end
endmodule

A Sample Model (RTL), 'always' block

Description

Implementation

must use 'reg' declaration if signal assigned from always block.

Process triggered on any change on signals a,b,c.

module majconc (a,b,c,y);
input a,b,c;
output y;
and I0 (n1, a, b);
and I1 (n2, a, c);
and I2 (n3, b, c);
or I3 (y, n1, n2, n3);
endmodule

A Sample Model (Gate Level Primitives, built into language)

Description

Implementation

Gate primitive name

VHDL does not have the equivalent of gate-level primitives.

Asynchronous vs Synchronous Inputs

reg q;
always @(posedge clk) begin
  if (r) q <= 0;
  else q <= d;
end
Synchronous reset, high true

reg q;
always @(posedge clk or posedge r) begin
  if (r) then q <= 0;
  else q <= d;
end
Asynchronous reset –, high true.

Need 'posedge' on 'r' because Verilog syntax requires if any signals are edge-triggered in event list, all signals must be edge-triggered.

Style suggested by C. Cummins, SNUG 2002

Nonblocking vs blocking assignments

• A nonblocking assignment (<=) samples right hand side (RHS) at beginning of timestep; with the actual assignment (the LHS) taking place at the end of the timestep
  – Works like a signal assignment in VHDL
• A blocking assignment (=) will evaluate the RHS and perform the LHS assignment without interruption from another Verilog statement
  – Works like a variable assignment (:=) in VHDL
• Should use nonblocking assignments in always blocks used to synthesize/simulate sequential logic.

Nonblocking Assignments

module timetest (y1,y2,a,clk);
output y1,y2;
input a,clk;
reg y1,y2;
always @(posedge clk) begin
  y1 <= a;
end
endmodule

with nonblocking assignments, ordering of these always blocks does not affect RTL simulation or synthesized gates.

module timetest (y1,y2,a,clk);
output y1,y2;
input a,clk;
reg y1,y2;
always @(posedge clk) begin
  y1 <= a;
end
always @(posedge clk) begin
  y2 <= y1;
end
endmodule

More on nonblocking assignments
When to use blocking assignments
Use blocking assignments for always blocks that are purely combinational
```verilog
reg y, t1, t2;
always @(a or b or c or d) begin
  t1 = a & b;
  t2 = c & d;
  y = t1 || t2;
end
```

RTL simulation and synthesis results match

Some Rules
- The paper by Cummings lists several rules for writing Verilog in which RTL simulation will match synthesized gate level simulation. The most important of these rules are:
  - Use blocking assignments in `always` blocks that are purely combinational
  - Use only nonblocking assignments in `always` blocks that are either purely sequential or have a mixture of combinational and sequential assignments.
- If you understand the differences between blocking and nonblocking assignments in terms of simulation, then these rules are self-evident.

Verilog Vs. VHDL
- Verilog and VHDL are equivalent for RTL modeling (code that will be synthesized).
- For high level behavioral modeling, VHDL is better
  - Verilog does not have ability to define new data types
  - Other missing features for high level modeling
- Verilog has built-in gate level and transistor level primitives
  - Verilog much better than VHDL at below the RTL level.
- Bottom Line: You should know both!!!!!