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

Process Block

VHDL:

```
process (siga, sigb)
begin
  ……
 end;
```

Verilog:

```
always @ (siga  or  sigb)
  begin
    ….
  end
```

Both used to specify blocks of logic with multiple inputs/outputs

VHDL vs. Verilog: Signal Assignment

VHDL:

```
signal a, b, c, d: std_logic;
begin
  a <= b and c;
  d < =   (c or b) xor (not (a) and b);
end;
```

VERILOG:

```
wire a,b,c,d;
assign a =  b & c;
assign  d = (c | b) ^ (~a  & b);
```

Declarations for one-bit wires are optional.

Logical operators same as in C.

VHDL vs. Verilog: Interface Declaration

VHDL:

```
entity mux is
  port (  a,b, s: in std_logic;
          y : out std_logic);
architecture a of mux is
  begin
   ……
  end;
```

VERILOG:

```
module mux (a,b,s,y);
 input a,b,s;
 output y;
 ….
 endmodule
```


VHDL vs. Verilog: Busses

VHDL:

```
signal a,c: std_logic_vector(7 downto 0);
  begin
     a(3 downto 0) <= c (7 downto 4);
     c(0) <= '0';
     c<= "00001010";
  end;
```

Verilog:

```
wire [7:0] ;
  begin
    assign  a[3:0] = b[7:4];
     assign a[0] <= 0;
     assign a = 'b0000000;
  end;
```

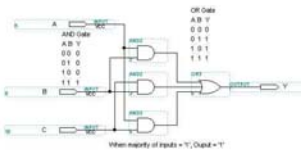
Value specified in binary.

A Sample Model (RTL), assignment statement

Description

```
module majconc (a,b,c,y);
  input a,b,c;
  output y;

  assign y =  (a & b) | (a & c) | (b &c) ;

endmodule
```

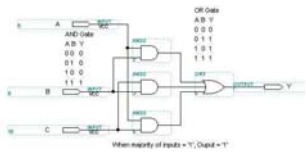
Implementation



A Sample Model (RTL), 'always' block

Description

Implementation



```
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
```

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

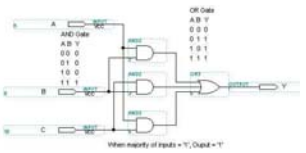
Process triggered on any change on signals a,b,c .

Non blocking assignment

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

Description

Implementation



```
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
```

Gate primitive name

Instance name

Output

Inputs

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;
```

Synchronous reset, high true

```
reg q;

always @(posedge clk or posedge r)
begin
 if ( r) then q <= 0;
 else q <= d;
end
```

Non-blocking assignment

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. Cummings, SNUG 2002

nonblocking vs blocking assignments

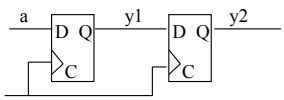
- A nonblocking assignment (<=) samples right hand side (RHS) at beginning of timestep; with the actual assignment (the LHS) taking place at the end of the timestep
  - Works like a signal assignment in VHDL
- A blocking assignment (=) will evaluate the RHS and perform the LHS assignment without interruption 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;
   y2 <= y1;
 end

endmodule
```

Nonblocking assignment

Synthesis results in DFF chain



More on nonblocking assignments

```
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
```

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

When to use blocking assignments

Use blocking assignments for always blocks that are purely combinational

```
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!!!!!