Verilog

- See EE 8999 page for Verilog links.
- Verilog compile command under Model tech is ‘vlog’ on NT, on Unix it is “qvlcom”
- See ~reese/verilog_train for many Verilog examples
- Book "Verilog QuickStart" from Kluwer Academic publishers is a good book, but expensive.

VHDL vs Verilog:

**Process Block**

VHDL:
```
process (siga, sigb)
begin
    ...
end;
```

Verilog:
```
always @ (siga or sigb)
begin
    ...
end
```

**Concurrent signal assignment**:

VHDL:
```
c <= a and b     
```

Verilog:
```
assign c = a & b ;
```

VHDL vs Verilog (cont):

**Signal Delays**

VHDL:
```
a <= transport b after 1 ns;  
```

Verilog:
```
#1 assign a = b;  
```

The ‘#’ operator is the delay operator. # N will delay for N simulation units. Delays can assigned to both inputs and outputs.

VHDL:
```
#1 assign a = #1 b;  
```

Verilog:
```
'b' is delayed by 1 unit, then assigned to 'a', which is then delayed by 1 time unit.
Infinite Loop
always
begin
  c = a & b;
end;
Same as infinite process loop in VHDL.

Clock Generator
VHDL:
  signal clk : std_logic := '0';
  process
  begin
    clk <= not (clk) after clkperiod/2;
    wait on clk;
  end;
Verilog:
  initial clk = 0;
  always #(clkperiod/2) clk = ~ clk;

Verilog Data Types

  bit, can take on values of '1', '0', 'x', 'z'
  integer : 32 bits
  integer a,b;
  reg (register, holds unsigned integers N bits wide)
  reg x, y[7:0], z[0:7];
  x is a 1 bit register, y,z are 8 bit registers. Most significant bit is always left most bit.
  real x, y;
  time t1, t2;
  Time value are 64 bits, units can be set on a per module basis.
Verilog Data Types (cont)

module string1;
reg[8*13 : 1] s;
initial begin
 s = "Hello Verilog";
 $display("The string %s is stored as %h", s, s);
end
endmodule

Strings are stored in registers that hold 8 * the number of characters in the string.

Register with Sync Clear

module reg16t(q, d, clk, clr_n);
input [15:0] d;
input clk, clr_n;
output [15:0] q;
reg [15:0] q;
always @(posedge clk)
 if(clr_n)
  q = #1 d;
 else
  #1 q = 0;
endmodule

Verilog Primitives

- Gates
  - and, nand, or, nor, xor, xnor
- Buffers
  - buf, not, pulldown, pullup, bufif0, notif0, bufif1, notif1
- Transistors
  - nmos, pmos, cmos, (unidirectional switches)
  - nmos, pmos, rmos (strength reduction of unidirectional switches)
  - tran, rtran, tranif0, rtranif0, tranif1, rtranif1 (bi-directional switches with their strength reduction equivalents)
Structural Model

```verbatim
module mux(OUT, A, B, SEL);
output OUT;
input A, B, SEL;
not I5 (sel_n, SEL);  // instance name
and I6 (sel_a, A, SEL);  // output
and I7 (sel_b, sel_n, B);  // inputs
or I4 (OUT, sel_a, sel_b);
endmodule
```

MUX

```
AND, OR, XOR, etc primitives can have any number of inputs.
```

Two Muxes

```verbatim
module mux2(OUT, A, B, SEL);
output [1:0] OUT;
input [1:0] A, B;
input SEL;

mux hi [OUT[1], A[1], B[1], SEL];
mux lo [OUT[0], A[0], B[0], SEL];
endmodule
```
Wires
- Wire declarations are used in structural models to connect instance pins that are not connected to ports
  - wire sum; // one bit wire
  - wire [7:0] d, e, f; // three 8-bit vectors
- A vector is a wire wider than 1 bit
- Different wire types available
  - supply1, supply0 - always '1', always '0' with a strength of 'supply'
  - wand - wired and
  - wor - wired or
  - tri1 - wire with built in pullup
  - tri0 - wire with built in pulldown
  - trireg - storage node for switch-level modeling

Drive Strengths
- 8 different drive strengths
  - 0 to 7, 0 is weakest, 7 is highest
- Drive table
  - 0 (highz0, highz) high impedance
  - 1 (\ldots) small capacitor
  - 2 (\ldots) medium capacitor
  - 3 (weak0, weak1) weak drive
  - 4 (\ldots) large capacitor
  - 5 (pull0, pull1) pull drive
  - 6 (strong0, strong1) strong drive
  - 7 (supply1, supply0) supply drive
- Default drive is ‘strong’.

Using Drive Strengths, Delays in Primitives
- Nand gate with open drain output
  - nand (strong0, highz1) G0 (y, a, b)
- Wimpy buffer
  - buff (weak1, weak0) #6 G1 (out, in)
  - Note that this buffer output has a delay of 6 units
- Delays on primitive outputs can specified as (rise, fall, turnoff).
  - Only use turnoff if output can go to ‘z’ value
  - #(3,2) a(c, w, z); rise/fall delay
- Delays can also have (min:typ:max) values
  - buff8 #2:3:4, 1:2:3, 3:4:5) b2 (f, e, c)
  - Note that each rising, falling, turnoff delay has min, typical, maximum values.
Time Units specified in Module

```
`timescale 1ns / 100ps
module mod1;
    // #1.1 in this module = 1.1 ns
endmodule

`timescale 100ps / 1ps
module mod2;
    // #2 in this module = 200 ps
endmodule;
```

User Defined Primitives (UDP)

Use truth tables to describe module behavior. Below is a MUX description:

```
primitive pmux( y, sel, a, b );
output y;
input sel, a, b;
table
    // s a b : y;
    0 0 ? : 0;
    0 1 ? : 1;
    1 0 ? : 0;
    1 1 ? : 1;
endtable
endprimitive
```

UDP table symbols: '1', '0', 'x' (unknown), '?' (matches 0, 1, x)
If a set of inputs is not covered by a line in the table, output is unknown.

UDP for D Flip-Flop

```
primitive dff(q, clk, d);
output q;
reg q;
input clk, d;
table
    // c d : q : q*
    0 0 : ?: 0;
    r 0 : ?: 1;
    1 ? : ?: 1;
    * : ?: ?: -; // no change on falling clock
    ? * : ?: ?: -; // no change on steady clock
endtable
endprimitive
```

Sequential UDP table symbols: 'r' (rising), 'f' (falling), "?" (any changes), '-.' (output remains unchanged).
UDP misc

- Only 1 output allowed, max of 10 inputs
- Cannot use 'z' in input table
- UDP instances just like module instance declarations
  - output must come first followed by input names
- In a sequential UDP, all transitions that do not affect the output must be specified, or output goes to 'x'
  - Input transitions and their effect on the output must be fully specified

Parameterized MUX

```verilog
module nmux4(a,b,c,d,sel,y);
  // Parameterized n bit wide 4 to 1 mux.
  parameter size = 32; // default to 32 bits
  input [size-1 : 0] a,b,c,d;
  input [1:0] sel;
  output [size-1 :0] y;
  always @ (a or b or c or d or sel)
    case (sel)
      0: y = a;
      1: y = b;
      2: y = c;
      3: y = d;
      default: y = 'bx;
    // will automatically size to fit
    endcase
  endmodule
```

TestBench

```verilog
module test_adder;
  reg [7:0] a,b;
  reg carry_in;
  wire [7:0] sum;
  wire carry_out;
  adder8 dut(carry_out, sum, a,b, carry_in);
  // initial block always executed at time 0, only once
  initial begin
    a = 0; b = 0; carry_in = 0;
    $ 100 if (sum !== 0) begin
      $display("sum is wrong");
      $finish; // 'finish' causes simulator exit
    end
    a = 1; b = 0; carry_in = 0;
    $ 100 if (sum !== 1) begin
      $display("sum is wrong");
      $finish;
    end
    $finish
  end
endmodule
```
Verilog Strengths

- Built in primitives for gate level, switch level modeling
  - UDPs nice, compact method for specifying custom gate behavior
  - Built-in strength system, multi-valued logic system
  - Delay system with rising/falling/turnoff with max/min/typical values
- Also has a defined interface for calling modules written in other programming languages such as ‘C’
  - Helps offset weakness in high level modeling

Verilog Weaknesses

- Not well suited for complex, high level modeling
  - No user defined type definition
  - No concept of libraries, packages, configurations
  - No ‘generate’ statement - can’t build parameterized structural models
  - No complex types above a two-dimensional array

Bottom Line

- Usually a company is either all Verilog or all VHDL
  - Most VLSI companies (US) use Verilog
  - Texas Instruments, Intel use VHDL - most European companies use VHDL
- Model Tech supports mixed Verilog/VHDL models
  - Would be nice to have low level blocks specified in Verilog, high level blocks in VHDL
- Extensions to both Verilog and VHDL for analog simulation (mixed signal) are in the works.