Verilog Examples

October 18, 2010

Structural Description of a Full Adder

```
module full_adder(x,y,cin,s,cout);
input x,y,cin;
output s,cout;
wire s1,c1,c2,c3;
xor(s1,x,y);
xor(s,cin,s1);
and(c1,x,y);
and(c2,y,cin);
and(c3,x,cin);
or(cout,c1,c2,c3);
endmodule
```

Hierarchical description of a Full Adder

```
//Gate Level description of Half Adder
module half_adder(x,y,s,c);
input x,y;
output s,c;
xor(s,x,y);
and(c,x,y);
endmodule
```

Full Adder

```
module full_adder(x,y,cin,s,cout);
input x,y,cin;
output s,cout;
wire s1,c1,c2;
half_adder ha1(x,y,s1,c1);
half_adder ha2(cin,s1,s,c2);
or(cout,c1,c2);
endmodule
```

Four bit Full adder

```
module four_bit_adder(x,y,cin,sum,cout);
input [3:0] x,y;
input cin;
output[3:0] sum;
output cout;
wire c1,c2,c3;
//fourbit adder body
full_adder fa1(x[0],y[0],cin,sum[0],c1);
full_adder fa2(x[1],y[1],c1,sum[1],c2);
full_adder fa3(x[2],y[2],c2,sum[2],c3);
full_adder fa4(x[3],y[3],c3,sum[3],cout);
endmodule
```

3-Input AND Gate Multi-bit signal inputs

```
module three_and_gate(a, out) ;
input [2:0] a ;
output out ;
wire out ;
assign out = a[0]&a[1]&a[2] ;
endmodule
```

3-Input OR Gate Multi-bit signal inputs

```
module three_or_gate(a, out) ;
input [2:0] a ;
output out ;
wire out ;
assign out =a[0]|a[1]|a[2] ;
endmodule
```

Modules can be hierarchical 3 INPUT NAND

```
module three_nand_gate_non_hierarch(a, b, c,
out) ;
input a, b, c ;
output out ;
wire out ;
assign out =∽ (a&b&c) ;
endmodule
```

Modules can be hierarchical 3 INPUT NAND

```
module three_nand_gate_hierarch(a, b, c, out) ;
input a, b, c ;
output out ;
wire out, temp;
three_and_gate(a, b, c, temp);
assign out =∽ temp ;
endmodule
```

Days in a Month

```
module DaysInMonth(month, days) ;
input [3:0] month ;
output [4:0] days ;
reg [4:0] days ;
always @(*)
begin
case(month)
2: days = 5'd28 ;
4,6,9,11: days = 5'd30 ;
default: days = 5'd31 ;
endcase
end
endmodule
```

Decoder

```
// a - binary input (n bits wide)
// b - one hot output (m bits wide)
module Dec(a, b) ;
parameter n=2 ;
parameter m=4 ;
input [n-1:0] a ;
output [m-1:0] b ;
wire [m-1:0] b = 1«a ;
endmodule
```

Prime Number function

```
module Prime(in, isprime) ;
input [2:0] in ;
output isprime ;
wire [7:0] b ;
// instantiate a 3 to 8 decoder
Dec #(3,8) d1(in,b) ;
// compute the output as the OR of minterms
wire isprime = b[1]|b[2]|b[3]|b[5]|b[7];
endmodule
```

Testing a Design

```
module Dec_Test();
reg[1:0] in;
wire[3:0] out;
Dec #(2,4) dec1(in,out);
initial begin
in = 2'b00;
#5
$display("With input %b, output is %b", in, out);
in = 2'b01;
```

Testing a Design

```
#5
$display("With input %b, output is %b", in, out);
in = 2'b10;
#5
$display("With input %b, output is %b", in, out);
in = 2'b11;
#5
$display("With input %b, output is %b", in, out);
$stop;
end
endmodule
```

Four bit Adder -Data Flow Description

```
module adder4(A,B,Cin,SUM,Cout);
input [3:0] A,B;
input Cin;
output [3:0] SUM;
output Cout;
assign {Cout,SUM}=A+B+Cin;
endmodule
```

Behavioral Description 2 to 1 line multiplexer

```
module mux2to1(mout,a,b,s);
output mout;
input a,b,s;
reg mout;

always @(a or b or s)
    if (s==1) mout=a;
    else mout=b;

endmodule
```

Behavioral Description of 2 to 4 Decoder

```
module dec2x4(xin,yout,enable);
input [1:0] xin; input enable;
output[3:0] yout;
reg[3:0] yout;

always @(xin or enable)
  begin
  if(enable==1)
  case(xin)
        2'b00: yout=4'b0001;
        2'b01: yout=4'b0010;
        2'b10: yout=4'b0100;
        2'b11: yout=4'b1000;
  endcase
  else
  yout=4'b0000;
  end

endmodule
```

Writing a Test Bench

Use **initial** and **always** to generate inputs for the unit you are testing.

```
initial
begin // if more than one statement it must be put begin ...end block
A=0; B=0;
# 10 A=1;
# 10 A=0; B=1;
end
initial
begin
D=3'b000;
repeat(7) // repeat following statement 7 times
# 10 D=D+3'b001;
end
```

Testing module

module test_module_name;
//Declare local **reg** and **wire** identifiers.
//Instatiate the design module under test
//Generate stimulus (inputs to design module) using **initial** and **always**.
// Display output response.(Display on screen or print)
endmodule

Display and Control of Testing

$display– display value of variables
$monitor– display variables whenever a value changes during simulation
$time– display simulation time
$finish– terminate simulation
Examples:
$display("%d %b %b",A,B,C);
$monitor($time„"%b %b %h" , s1,s2,outvalue);

Testbench example Test mux2to1

```
module testmux2to1;
wire tout;
reg tA,tB,tselect;
parameter stoptime=50;
mux2to1 mux1(tout,tA,tB,tselect);
initial #stoptime $finish;
initial begin
tselect=1;tA=0; tB=1;
#10 tA=1; tB=0;
#10 tselect=0;
#10 tA=0;tB=1;
end
```

Testbench example Test mux2to1- Cont

```
initial begin $display(" time select A B mout");
$monitor("time=%d select=%b A=%b B=%
mout=%b",$time,tselect,tA,tB,tout);
end
endmodule
```