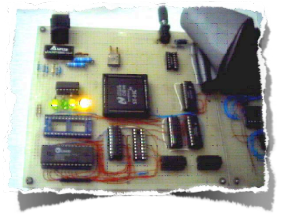


Verilog HDL: Tip & Trick Pitfalls, Fallacies, Good designs

Krerk Piromsopa, Ph.D
Department of Computer Engineering
Chulalongkorn University

Outline & Target

- Outline
  - Revisit basic knowledge of Verilog
  - Emphasis on Design Pitfall
- Goals
  - After this class, if you have a better idea of using Verilog as a language to describe your design, I have succeed.

Basic

- Four-Value Logic
  - 0, 1 , x (unknow), z (high impedance)
- number representation
  - 6'b10_011, 16'hx
  - 6'b_101111 WRONG

Basic

- Parameter is a constant, can only be integer
  - **parameter WIDTH = 32, depth = 1024;**
- Nets, Registers
- bitwise and logic
- comparison ==, === (only w===v)
- case, casex

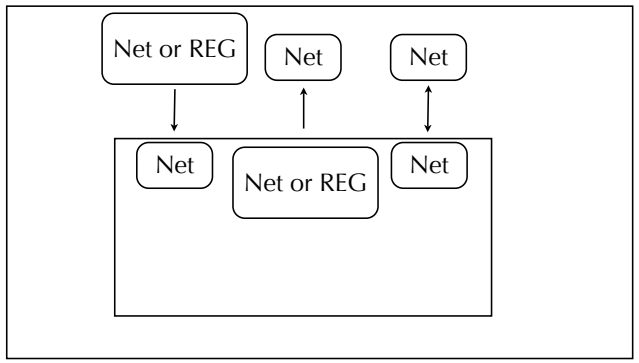
Basic

- Concatenation
  - { cout, sum }
- Replication
  - {8{2'b01}};
-

Basic

- Variable Part-Select (Verilog 2001)
  - sum[K+:3] --- {sum[K+2],sum[K+1],sum[K]}
- Memories
  - **parameter WIDTH = 32, depth = 1024**
  - **reg [WIDTH-1:0] cache [0:depth-1];**
  - **reg [WIDTH-1:0] cache [0: 2**WIDTH -1];** // (Verilog 2001)

Input/Output



Common Notation

Output before input

- Verilog 1995

```
module add
(sum, c_out, a, b);

output sum, c_out;

input a,b;
```

- Verilog 2001

```
module add(
output sum, c_out,
input a,b);
```

User-defined primitives (UDPs)

```
primitive latch(q_out,en, data);
output reg q_out;
input en, data;

table
// en data : state : q_out/next_state
1 1 : ? : 1;
1 0 : ? : 0;
0 ? : ? : -;
x 0 : 0 : -;
x 1 : 1 : -;

endtable

endprimitive
```


Assignment

- Continuous assignment
  - a = b & c;
  - left-handed side must be net.
- Procedural assignment
  - in always
  - left-handed size must be reg.

Event

always @ (***events***) begin

end

Concurrent

- events are synchronization signals or all sensitivity list.
- posedge , negedge

Overriding Parameters

```
module adder (sum,
c_out,a,b);
parameter width=4;
output reg [width-1:0]
sum;
output reg c_out;
input [width-1:0] a,b;
always @(a,b) begin
      {c_out,sum} = a+b;
end
endmodule
```

```
module top();
....
// 8 bit adder
adder  #(8) add8(.....)
// 16 bit adder
adder #(16) add16 (...)

endmodule
```

For

```
input [7:0] A
output reg Valid;
output reg [2:0] Y;
integer i;

always @(A) begin
Valid=0;
Y=3'bX;
for(i=0;i<8;i=i+1)
  if (A(i)) begin
    Valid=1;
    Y=N;
  end
end
```

```
input [7:0] A
output reg Valid;
output reg [2:0] Y;
integer i;
always @(A) begin
 Valid=1;
casex(A)
 8'b1XXX_XXXX : Y=7;
 8'b01XX_XXXX : Y=6;
 8'b001X_XXXX : Y=5;
 8'b0001_XXXX : Y=4;
 8'b0000_1XXX : Y=3;
 8'b0000_01XX : Y=2;
 8'b0000_001X : Y=1;
 8'b0000_0001 : Y=0;
 default: begin
  Valid=0;
  Y=3'bX;
        end
endcase
end
endmodule
```

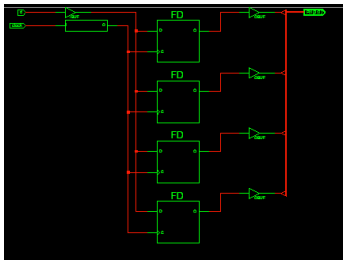

Pitfalls

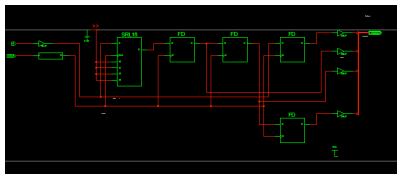
Common Pitfall

- Verilog is not a programming language, but a HDL.
- Don't try to debug a design that doesn't work.
- Don't use Latches or Flip-Flops in combinational circuit.

Blocking vs. NonBlocking

```
module shiftReg1(out,d, clock);
output reg [3:0] out;
input d, clock;
always @(posedge clock) begin
      out[0]=d;
      out[1]=out[0];
      out[2]=out[1];
      out[3]=out[2];
end
endmodule

module shiftReg2(out,d, clock);
output reg [3:0] out;
input d, clock;

always @(posedge clock) begin
      out[3]=out[2];
      out[2]=out[1];
      out[1]=out[0];
      out[0]=d;
end

endmodule
```





Blocking vs. NonBlocking

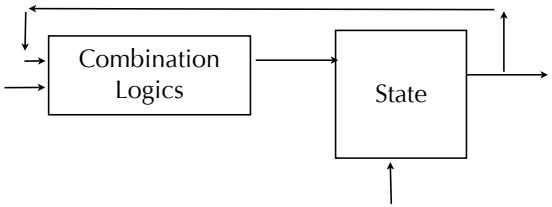
```
module shiftReg3(out,d, clock);
output reg [3:0] out;
input d, clock;

always @(posedge clock) begin
    out[0]<=d;
    out[1]<=out[0];
    out[2]<=out[1];
    out[3]<=out[2];
end

endmodule
```

```
module shiftReg4(out,d, clock);
output reg [3:0] out;
input d, clock;

always @(posedge clock) begin
    out[3]<=out[2];
    out[2]<=out[1];
    out[1]<=out[0];
    out[0]<=d;
end

endmodule
```


FSM Behaviors

- Finite State Machine (FSM)
  - state register
  - combinational logic

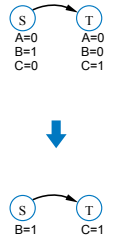
Good vs. Bad




```
always @(posedge clock) begin
   ps=ns;
end

always @([input], ps) begin

end
```
Good

```
always @(posedge clock) begin
   case ...

   ....
   ....
   ...

end
```
Bad

Output Assignment



- FSM outputs should be combinational functions of Current State (Moore) and/or Input (Mealy)
  - No assignment assumes output is a memory. Thus it is not FSM.

```
case State
    S: begin
       B<=1;
    end
    T: begin
       C<=1;
    end
endcase
```

```
case State
    S: begin
      A<=0;
      B<=1;
      C<=0;
    end
    T: begin
       A<=0;
       B<=0;
       C<=1;
    end
endcase
```

```
A<=0;
B<=0;
C<=0;
case State
    S: begin
      B<=1;
    end
    T: begin
      C<=1;
    end
endcase
```

Ambiguous

- Unknown State
  - Always have reset signal
  - Default State
- Unknown Logic
  - Case with default
  - If with else

Implicit Declaration

```
output [3:0] X;

input [3:0] a,b;

assign X=y;

always @(a,b) begin

y=a&b;

end
```


Guideline

- Think as a circuit designer, not a programmer

References

- on-line resources:
- http://www.sutherland-hdl.com/on-line_ref_guide/vlog_ref_top.html
- http://www.engineering.usu.edu/classes/ece/2530/appendb.pdf
- http://www.asic-world.com/verilog/vqref.html
- http://oldeee.see.ed.ac.uk/~gerard/Teach/Verilog/
- Google is your friend.