

inst.eecs.berkeley.edu/~cs61c
CS61C : Machine Structures

**Lecture 24 –
Verilog II**

2004-10-27



Lecturer PSOE Dan Garcia

www.cs.berkeley.edu/~ddgarcia

Apple's iPod Photo!!! ⇒

The world's best-selling mp3 player just got a lot better... Now it holds 60 GB of music and photos and data. It has a high-res 220x176 screen and A/V features, like a slideshow mode using your music and A/V out directly to your TV. Prices from \$499-\$599, cheaper @ TSW.



apple.com



Review

- Verilog allows both structural and behavioral descriptions, helpful in testing
- Syntax a mixture of C (operators, for, while, if, print) and Ada (begin... end, case...endcase, module ...endmodule)
- Some special features only in Hardware Description Languages
 - # time delay, initial vs. always
- Verilog easier to deal with when thought of as a hardware modeling tool, not as a prog lang.
- Want to Monitor when ports, registers updated
- Verilog can describe everything from single gate to full computer system; you get to design a simple processor



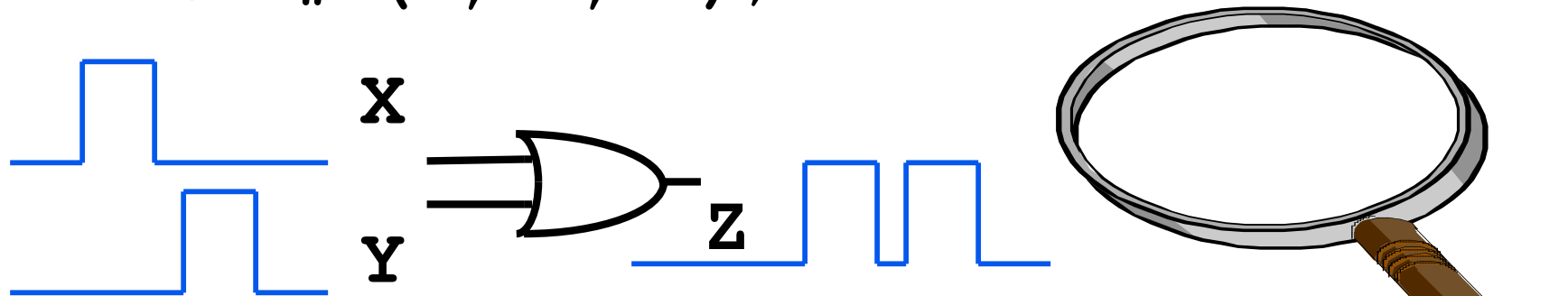
Review: Verilog Pedagogic Quandary

- Using Verilog subset because full Verilog supports advanced concepts that are not described until \geq CS 150, and would be confusing to mention now
- Trying to describe a new language without forcing you to buy a textbook, yet language bigger diff than C vs. Java
- Hence Wawrzynek Verilog tutorial
 - We'll go through most of tutorial together
- Do the tutorials, don't just Read them



Time, variable update, module, & monitor

```
or #2 (z, x, y);
```



- The instant before the rising edge of the clock, all outputs and wires have their OLD values. This includes inputs to flip flops. Therefore, if you change the inputs to a flip flop at a particular rising edge, that change will not be reflected at the output until the NEXT rising edge. This is because when the rising edge occurs, the flip flop still sees the old value.

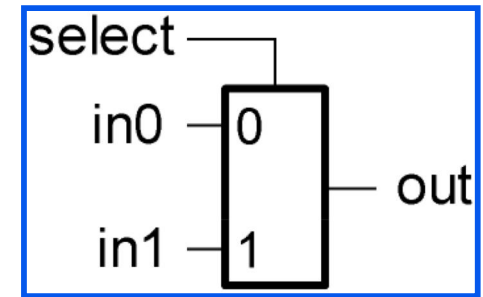
- When Verilog is simulating, time changes, then ports (variables), registers updated, modules invoked, monitor



Example page 6 in Verilog Tutorial

```
// Test bench for 2-input multiplexor.  
// Tests all input combinations.
```

```
module testmux2;  
    reg [2:0] c;  
    wire f;  
    reg expected;  
    mux2 myMux (.select(c[2]), .in0(c[0]),  
                .in1(c[1]), .out(f));  
    initial  
        begin  
            c = 3'b000; expected=1'b0; ...
```



- Verilog constants syntax $N'Bxxx$ where
N is size of constant in bits
B is base: b for binary, h for hex, o for octal
xxx are the digits of the constant



Example page 7 in Verilog Tutorial

```
... begin
    c = 3'b000; expected=1'b0;
    repeat (7)
        begin
            #10 c = c + 3'b001;
            if (c[2]) expected=c[1];
            else expected=c[0];
        end
    end
    #10 $finish;
end
```

- Verilog `if` statement, `for` and `while` loops like C
 - `repeat (n)` loops for n times (restricted `for`)
 - `forever` is an infinite loop
- Can select a bit of variable (`c[0]`)

`$finish` ends simulation



Example page 7 in Verilog Tutorial

```
    . . .
    end
initial
begin
    $display("Test of mux2.");
    $monitor("[select in1 in0]=%b out=%b
    expected=%b time=%d",c, f, expected,
    $time);
    end
endmodule // testmux2
```

• Verilog “printf” statements

- `$display` to print text on command
- `$write` to print text on command, no new line
- `$strobe` prints only at certain times
- `$monitor` prints when any variable updated



Output for example on page 7 (no delay)

Test of mux2.

```
[select in1 in0]=000 out=0 expected=0 time=0
[select in1 in0]=001 out=1 expected=1 time=10
[select in1 in0]=010 out=0 expected=0 time=20
[select in1 in0]=011 out=1 expected=1 time=30
[select in1 in0]=100 out=0 expected=0 time=40
[select in1 in0]=101 out=0 expected=0 time=50
[select in1 in0]=110 out=1 expected=1 time=60
[select in1 in0]=111 out=1 expected=1 time=70
```

**•Note: c output is 3 bits (000 ... 111)
because c declared as 3 bit constant**

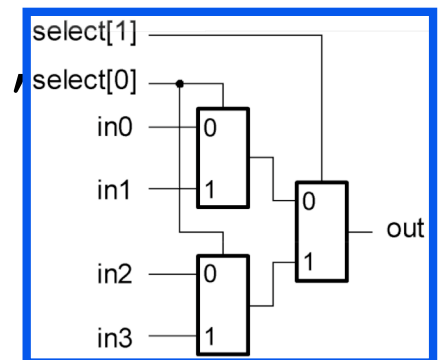
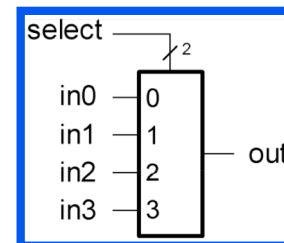


Example page 10 Verilog Tutorial

```
// 4-input multiplexor built from
// 3 2-input multiplexors
module mux4 (in0, in1, in2, in3, select, out);
  input in0,in1,in2,in3;
  input [1:0] select;
  output out;
  wire w0,w1;
  mux2
    m0 (.select(select[0]), .in0(in0),
      .in1(in1), .out(w0)),
    m1 (.select(select[0]), .in0(in2),
      .in1(in3), .out(w1)),
    m2` (.select(select[1]), .in0(w0),
      .in1(w1), .out(out));
endmodule // mux4
```

What is size of ports?

What are m0, m1, m2?



Part of example page 11 Verilog Tutorial

```
...
case (select)
  2'b00: expected = a;
  2'b01: expected = b;
  2'b10: expected = c;
  2'b11: expected = d;
endcase; // case(select)
...
```

- Verilog case statement different from C
- Has optional default case when no match to other cases
- Case very useful in instruction interpretation; case using opcode, each case an instruction



Administrivia

- **We have an updated grading standard for The Float Question (TFQ).**
 - **We offer “free” regrades to anyone who lost points on TFQ.**
 - **“Free” = If that’s your only regrade, we will NOT review the other questions of your entire exam.**
 - **Everyone who lost points on TFQ should take advantage of this**
 - **Just write “FLOAT” on the top-right of the front cover of your exam and return your exam to your TA before Mon @ 4pm.**



Rising and Falling Edges and Verilog

- **Challenge of hardware is when do things change relative to clock?**
 - **Rising clock edge?**
(“positive edge triggered”)
 - **Falling clock edge?**
(“negative edge triggered”)
 - **When reach a logical level?**
(“level sensitive”)
- **Verilog must support any “clocking methodology”**
- **Includes events “posedge”, “negedge” to say when clock edge occur, and “wait” statements for level**



Example page 12 Verilog Tutorial

```
// Behavioral model of 4-bit Register:  
// positive edge-triggered,  
// synchronous active-high reset.  
module reg4 (CLK,Q,D,RST);  
    input [3:0] D;  
    input CLK, RST;  
    output [3:0] Q;  
    reg [3:0] Q;  
    always @ (posedge CLK)  
        if (RST) Q = 0; else Q = D;  
endmodule // reg4
```

- On positive clock edge, either reset, or load with new value from D



Example page 13 Verilog Tutorial

```
...
initial
begin
  CLK = 1'b0;
  forever
    #5 CLK = ~CLK;
end
```

...

- No built in clock in Verilog, so specify one
- Clock CLK above alternates forever in 10 ns period:
5 ns at 0, 5 ns at 1,
5 ns at 0, 5 ns at 1, ...



Example page 14 Verilog Tutorial

```
module add4 (S,A,B) ;
```

- a combinational logic block that forms the sum (S) of the two 4-bit binary numbers (A and B)
- Tutorial doesn't define this, left to the reader
- Write the Verilog for this module in a behavioral style now
 - Assume this addition takes 4 ns



Example page 14 Verilog Tutorial

```
module add4 (S,A,B) ;  
  input [3:0] A, B;  
  output [3:0] S;  
  reg S;  
  always @ (A or B)  
    #4 S = A + B;  
endmodule
```

- a combinational logic block that forms the sum of the two 4-bit binary numbers, taking 4 ns
- Above is behavioral Verilog



Example page 14 Verilog Tutorial

```
// Accumulator
```

```
module acc (CLK,RST,IN,OUT) ;
```

```
input CLK,RST;
```

```
input [3:0] IN;
```

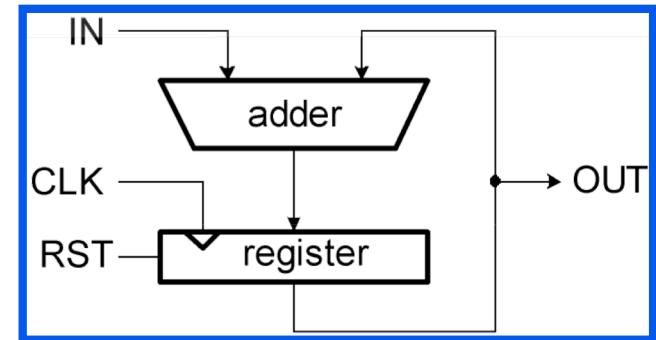
```
output [3:0] OUT;
```

```
wire [3:0] W0;
```

```
add4 myAdd (.S(W0), .A(IN), .B(OUT)) ;
```

```
reg4 myReg (.CLK(CLK), .Q(OUT),  
.D(W0), .RST(RST)) ;
```

```
endmodule // acc
```



- This module uses prior modules, using wire to connect output of adder to input of register



Example page 14 Verilog Tutorial

```
module accTest;
  reg [3:0] IN;
  reg CLK, RST;
  wire [3:0] OUT;
  acc myAcc (.CLK(CLK), .RST(RST),
    .IN(IN), .OUT(OUT));
  initial
  begin
    CLK = 1'b0;
    repeat (20)
      #5 CLK = ~CLK;
  end ...
```

- Clock frequency is ___ GHz?



Example page 14 Verilog Tutorial

...

```
initial
```

```
begin
```

```
#0 RST=1'b1; IN=4'b0001;
```

```
#10 RST=1'b0;
```

```
end
```

```
initial
```

```
$monitor("time=%0d: OUT=%1h",  
$time, OUT);
```

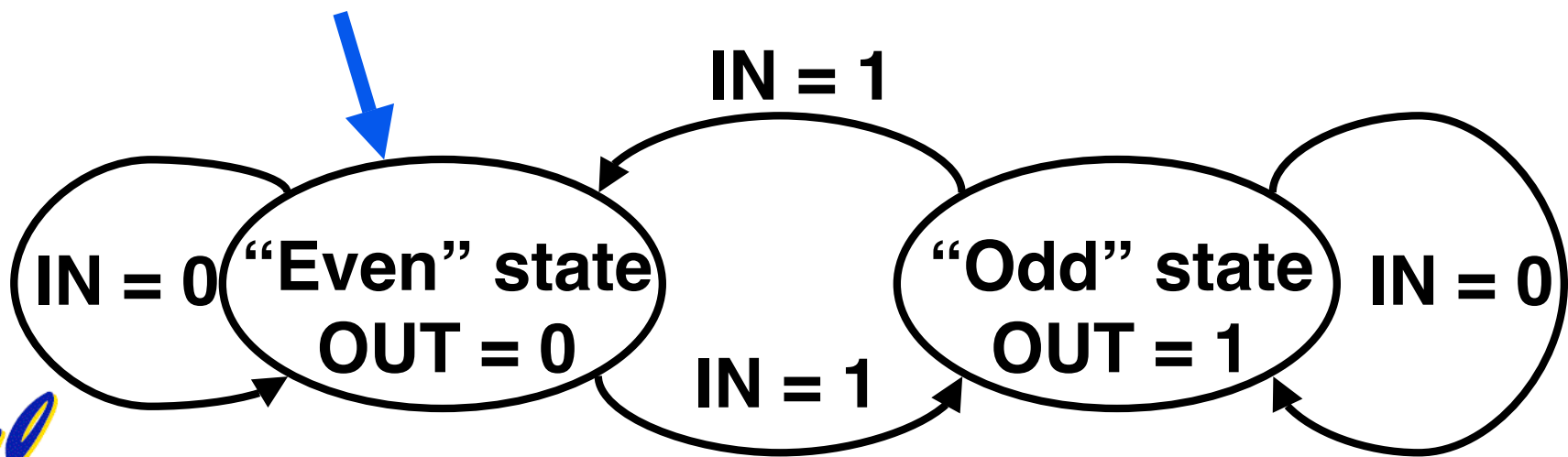
```
endmodule // accTest
```

- What does this initial block do?
- What is output sequence?
- How many lines of output?



Finite State Machines (FSM)

- Tutorial example is bit-serial parity checker
 - computes parity of string of bits sequentially, finding the exclusive-or of all the bits
 - Parity of "1" means that the string has an odd number of 1's



Example page 16, Verilog Tutorial

```
// Behavioral model of D-type flip-flop:  
// positive edge-triggered,  
// synchronous active-high reset.  
module DFF (CLK,Q,D,RST) ;  
    input D;  
    input CLK, RST;  
    output Q;  
    reg Q;  
    always @ (posedge CLK)  
        if (RST) #1 Q = 0; else #1 Q = D;  
endmodule // DFF
```

- Loaded on positive edge of clock

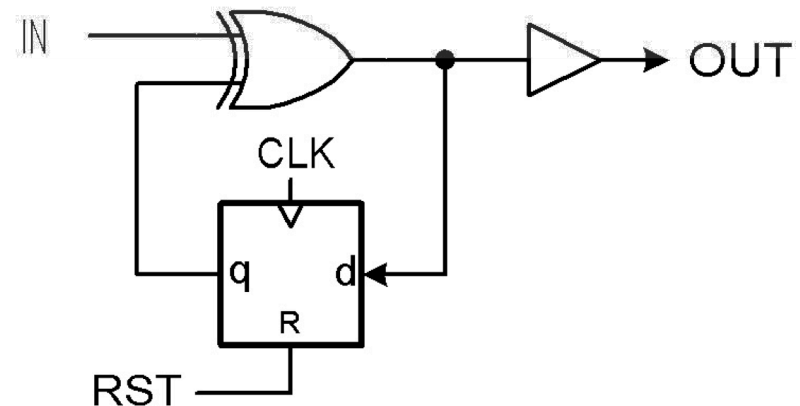


Example page 3, Part III Verilog Tutorial

```
// Structural model of serial parity checker.  
module parityChecker (OUT, IN, CLK, RST);  
    output OUT;  
    input IN;  
    input CLK, RST;  
    wire currentState, nextState;  
    DFF state (.CLK(CLK), .Q(currentState),  
              .D(nextState), .RST(RST));  
    xor (nextState, IN, currentState);  
    buf (OUT, nextState);  
endmodule // parity
```



Verilog doesn't like it
when you feed outputs
back internally...



Peer Instruction

- A. To test FSMs, you need to test every transition from every state**
- B. Modules must be updated with explicit calls to them, ala scheme**
- C. If input not explicitly set, output of CL defaults to '0'**

	ABC
1 :	FFF
2 :	FFT
3 :	FTF
4 :	FTT
5 :	TFF
6 :	TFT
7 :	TF F
8 :	TTT

In conclusion

- Verilog describes hardware in hierarchical fashion, either its structure or its behavior or both
- Time is explicit, unlike C or Java
- Time must be updated for variable change to take effect
- `monitor` print statement like a debugger; `display`, `write`, `strobe` more normal
- Modules activated simply by updates to variables, not explicit calls as in C or Java

