



University of
Massachusetts
Amherst

ECE232: Hardware Organization and Design

Part 3: Verilog Tutorial

<http://www.ecs.umass.edu/ece/ece232/>

Basic Verilog

```
module <module_name> (<module_terminal_list>);  
    <module_terminal_definitions>  
    ...  
    <functionality_of_module>  
    ...  
endmodule
```

Engin 112 Verilog examples:

<http://www.ecs.umass.edu/ece/engin112/labs/lab-E2-F09.html>

<http://www.ecs.umass.edu/ece/engin112/labs/lab-E3-F09.html>

ECE 353 – Verilog Resources

<http://www.ecs.umass.edu/ece353/verilog/verilog.html>

ECE 667 Verilog (on the left side menu):

<http://www.ecs.umass.edu/ece/labs/vlsicad/ece667/ece667.html>

<http://www.asic-world.com/examples/verilog/index.html>

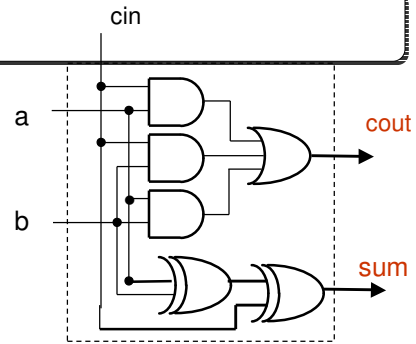
Full Adder

```

module FullAdder(a,b,cin,cout,sum);
    input a, b, cin;    // inputs
    output cout, sum;   // output
    wire w1, w2, w3, w4; // internal nets

    xor #(10) (w1, a, b); // delay time of 10 units
    xor #(10) (sum, w1, cin);
    and #(8) (w2, a, b);
    and #(8) (w3, a, cin);
    and #(8) (w4, b, cin);
    or #(10, 8) (cout, w2, w3, w4); // (rise time of 10, fall 8)
endmodule

```



ECE 232

Verilog tutorial

3

Multiple ways of implementing Full Adder

```

module FullAdder(a,b,cin,sum,cout);
    input a,b,cin;
    output sum, cout;
    reg sum, cout; // registers retain value
    always @ (a or b or cin) // Anytime a or b or cin
                                CHANGE, run the
    process
    begin sum <= a ^ b ^ cin;
        cout <= (a & b) | (a & cin) | (b & cin);
    end
endmodule

```

concurrent assignment

blocking assignment,
non-blocking assignments

ECE 232

Verilog tutorial

4

Ripple Carry Adder

```
4-bit Adder
module adder4(A, B, cin, S, cout);
    input[3:0] A, B;
    input cin;
    output[3:0] S;
    output cout;
    wire c1, c2, c3;
    // 4 instantiated 1-bit Full Adders
    FullAdder fa0(A[0], B[0], cin, c1, S[0]);
    FullAdder fa1(A[1], B[1], c1, c2, S[1]);
    FullAdder fa2(A[2], B[2], c2, c3, S[2]);
    FullAdder fa3(A[3], B[3], c3, cout, S[3]);
endmodule
```

HDL Overview

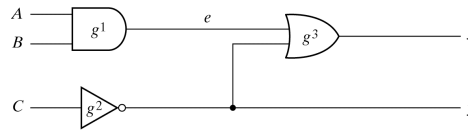
- Hardware description languages (HDL) offer a way to design circuits using text-based descriptions
- HDL describes hardware using keywords and expressions.
 - Representations for common forms
 - » Logic expressions, truth tables, functions, logic gates
 - Any combinational or sequential circuit
- HDLs have two objectives
 - Allow for testing/verification using computer simulation
 - » Includes syntax for timing, delays
 - Allow for synthesis
 - » Synthesizable HDL
 - The two forms often differ !
 - We will use synthesizable subset of verilog
- Two primary hardware description languages
 - VHDL
 - Verilog

Hardware Description Language - Verilog

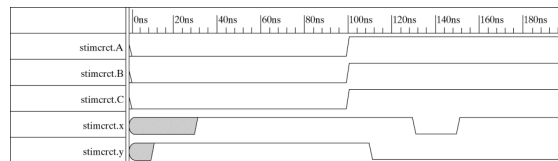
- Represents hardware structure and behavior
- Logic simulation: generates waveforms

//HDL Example 1
//-----

```
module smpl_circuit(A,B,C,x,y);
  input A,B,C;
  output x,y;
  wire e;
  and g1(e,A,B);
  not g2(y,C);
  or g3(x,e,y);
endmodule
```



- Detect errors before fabrication



ECE 232

Verilog tutorial

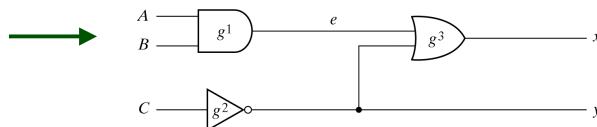
7

Verilog Keywords and Syntax

- Lines that begin with // are comments (ignored by simulation)
- About 100 keywords in total (keywords are case sensitive)
- **module**: Building block in Verilog
- Always terminates with **endmodule**
- **module** followed by circuit name and port list
- Each port is either an **input** or **output**

//HDL Example 2
//-----

```
module smpl_circuit(A,B,C,x,y);
  input A,B,C;
  output x,y;
  wire e;
  and g1(e,A,B);
  not g2(y,C);
  or g3(x,e,y);
endmodule
```



ECE 232

Verilog tutorial

8

Verilog Statements

Verilog has two basic types of statements

1. Concurrent statements (combinational)

(things are happening concurrently, ordering does not matter)

- Gate instantiations
and (z, x, y), **or** (c, a, b), **xor** (S, x, y), etc.
- Continuous assignments
assign Z = x & y; c = a | b; S = x ^ y

2. Procedural statements (sequential)

(executed in the order written in the code)

- **always @** - executed continuously when the event is active
always @ (posedge clock)
- **initial** - executed only once (used in simulation)
- **if then else** statements

ECE 232

Verilog tutorial

9

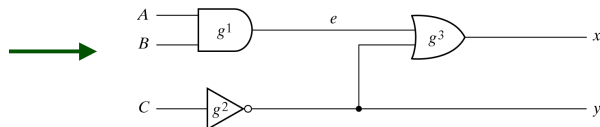
wire and gate-level Keywords

- Example of gate instantiation
 - **wire** defines internal circuit connection
 - Each gate (**and**, **or**, **not**) defined on a separate line
 - Gate I/O includes wires and port values
 - Note: each gate is **instantiated** with a name (e.g., **g1**)

//HDL Example 2

//-----

```
module smpl_circuit(A,B,C,x,y);  
  input A,B,C;  
  output x,y;  
  wire e;  
  and g1(e,A,B);  
  not g2(y,C);  
  or g3(x,e,y);  
endmodule
```



ECE 232

Verilog tutorial

10

Specifying Boolean Expressions

- Example of continuous assignment

```
//HDL Example 3
//-----
//Circuit specified with Boolean equations
```

```
module circuit_bln (x,y,A,B,C,D);
  input A,B,C,D;
  output x,y;
  assign x = A | (B & C) | (~B & C);
  assign y = (~B & C) | (B & ~C & ~D);
endmodule
```

- **assign** keyword used to indicate expression
- Assignment takes place continuously
- Note new symbols specific for Verilog
 - OR -> |
 - AND -> &
 - NOT -> ~

ECE 232

Verilog tutorial

11

User Defined Primitives

```
//HDL Example 4
//-----
//User defined primitive(UDP)
primitive crctp (x,A,B,C);
  output x;
  input A,B,C;
  //Truth table for x(A,B,C) = Minterms (0,2,4,6,7)
  table
  //   A  B  C : x (Note that this is only a comment)
    0  0  0 : 1;
    0  0  1 : 0;
    0  1  0 : 1;
    0  1  1 : 0;
    1  0  0 : 1;
    1  0  1 : 0;
    1  1  0 : 1;
    1  1  1 : 1;
  endtable
endprimitive
```

- Allows definition of truth table
- Only one output is allowed

ECE 232

Verilog tutorial

12

More Verilog Examples - 1

```
//HDL Example 5
//-----
//Dataflow description of a 2-to-4-line decoder
```

```
module decoder_df (A,B,E,D);
  input A,B,E;
  output [0:3] D;
  assign D[0] = ~(~A & ~B & ~E),
         D[1] = ~(~A & B & ~E),
         D[2] = ~(A & ~B & ~E),
         D[3] = ~(A & B & ~E);
endmodule
```

- Combinational functionality
- All assignments take place at the same time
- Note declaration of a **bus**
 - output [0:3] D;

ECE 232

Verilog tutorial

13

More Verilog Examples - 2

```
//HDL Example 6
//-----
//Dataflow description of a 4-bit comparator.
```

```
module mag_comp (A,B,ALTB,AGTB,AEQB);
  input [3:0] A,B;
  output ALTB,AGTB,AEQB;
  assign ALTB = (A < B),
         AGTB = (A > B),
         AEQB = (A == B);
endmodule
```

- Easy to define arithmetic functionality
- Each comparison creates a single bit result
- **Synthesizer** automatically converts RTL description to gate-level description
 - RTL = register transfer level

ECE 232

Verilog tutorial

14

More Verilog Examples - 3

- Example of sequential assignment

```
//HDL Example 7
//-----
//Behavioral description of 2-to-1-line multiplexer
```

```
module mux2x1_bh(A,B,select,OUT);
  input A,B,select;
  output OUT;
  reg OUT;
  always @ (select or A or B)
    if (select == 1) OUT = A;
    else OUT = B;
endmodule
```

- Conditional statements (`if`, `else`) allow for output choices
- `always` keyword used to indicate action based on variable change
- Generally conditional statements lead to multiplexers

ECE 232

Verilog tutorial

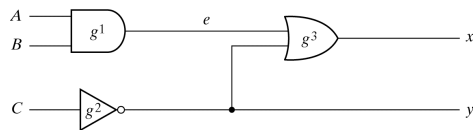
15

Modeling Circuit Delay

- This is for **simulation** only (not for synthesis)
- Timescale directive indicates units of time for simulation
 - `'timescale 1ns / 100ps`
- `#(30)` indicates an input to output delay for gate `g1` of 30 ns
- `#(10)` indicates an input to output delay for gate `g2` of 10 ns

```
//HDL Example 2
//-----
//Description of circuit with delay
```

```
module circuit_with_delay (A,B,C,x,y);
  input A,B,C;
  output x,y;
  wire e;
  and #(30) g1(e,A,B);
  or #(20) g3(x,e,y);
  not #(10) g2(y,C);
endmodule
```



ECE 232

Verilog tutorial

16

Test bench Stimulus - 1

```
//HDL Example 8
//Stimulus for simple circuit
module stimcrct;
  reg A,B,C;
  wire x,y;
  circuit_with_delay cwd(A,B,C,x,y);
  initial
    begin
      A = 1'b0; B = 1'b0; C = 1'b0;
      #100
      A = 1'b1; B = 1'b1; C = 1'b1;
      #100 $finish;
    end
endmodule
//Description of circuit with delay
// NOT synthesizable !
module circuit_with_delay (A,B,C,x,y);
  input A,B,C;
  output x,y;
  wire e;
  and #30 g1(e,A,B);
  or #20 g3(x,e,y);
  not #10 g2(y,C);
endmodule
```

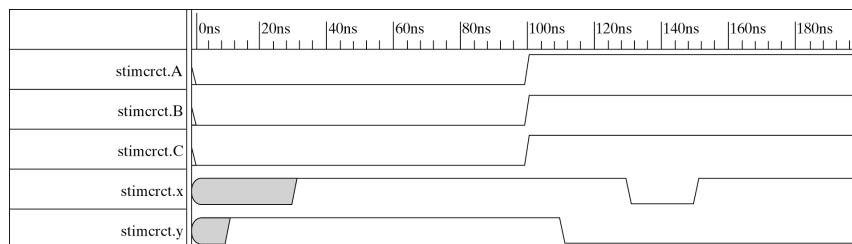
- Module circuit_with_delay is instantiated
- reg keyword indicates that values are stored (driven)
- Stimulus signals are applied sequentially
- \$finish indicates simulation should end
- Result: collection of waveforms

ECE 232

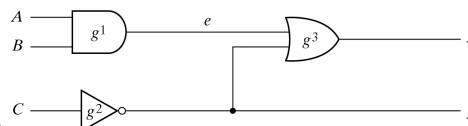
Verilog tutorial

17

Test bench Stimulus - 2



- Timescale directive indicates units of time for simulation
 - `timescale 1ns / 100ps`
- Note that input values change at 100ns
- Shaded area at left indicates output values are undefined



ECE 232

Verilog tutorial

18

Modeling Sequential Elements

module D-latch (D, Clk, Q);

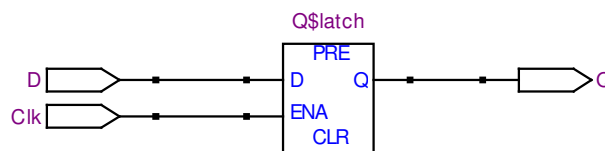
D Latch

input D, Clk;
output Q;
reg Q;

always @(D or Clk)

if (Clk)
Q = D;

endmodule



ECE 232

Verilog tutorial

19

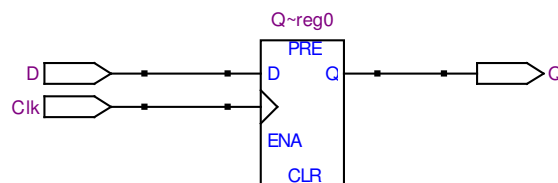
Verilog – D Flip-flop

module D-flipflop (D, Clk, Q);

input D, Clk;
output Q;
reg Q;

always @(posedge Clk)
Q = D;

endmodule



ECE 232

Verilog tutorial

20

Verilog - Blocking Assignment (=)

```
module DFF-blocking(D, Clock, Q1, Q2);
```

```
  input D, Clock;
  output Q1, Q2;
  reg Q1, Q2;
```

```
  always @(posedge Clock)
```

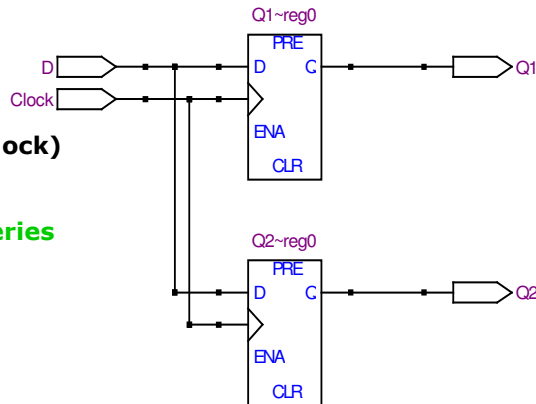
```
  begin
```

```
    // blocking assignment - series
    // execution
```

```
      Q1 = D;
      Q2 = Q1;
```

```
  end
```

```
endmodule
```



ECE 232

Verilog tutorial

21

Verilog – Non-blocking Assignment (<=)

```
module DFF-non-blocking(D, Clock, Q1, Q2);
```

```
  input D, Clock;
  output Q1, Q2;
  reg Q1, Q2;
```

```
  always @(posedge Clock)
```

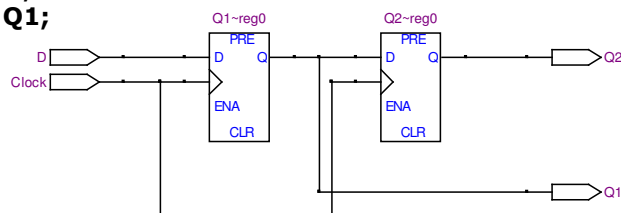
```
  begin
```

```
    // non blocking assignment - can be done in
    // parallel (or any order)
```

```
      Q1 <= D;
      Q2 <= Q1;
```

```
  end
```

```
endmodule
```



ECE 232

Verilog tutorial

22

Verilog – D Flip-flop with Reset

- D flip-flop with asynchronous reset (asserted negative)

```

module dff_reset(D, Clock, Resetn, Q);
  input D, Clock, Resetn;
  output Q;
  reg Q;

```

```

  always @(negedge Resetn or posedge
Clock)

```

```

    if (!Resetn)
      Q <= 0;

```

```

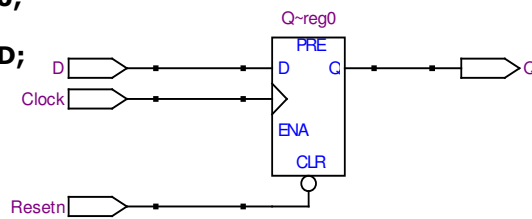
    else
      Q <= D;

```

```

endmodule

```



ECE 232

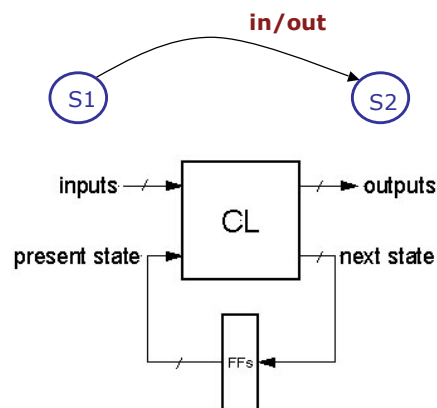
Verilog tutorial

23

Finite State Machines - 1

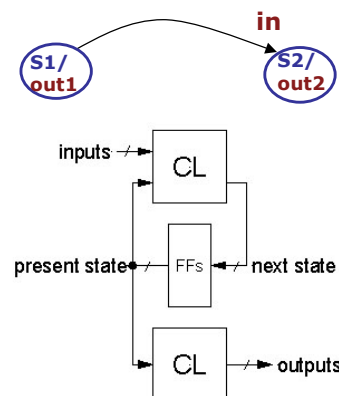
Mealy FSM

- Output based on present state and input
- Output changes during transition



Moore FSM

- Output based on state only
- Output is associated with state



ECE 232

Verilog tutorial

24

Finite State Machines - 2

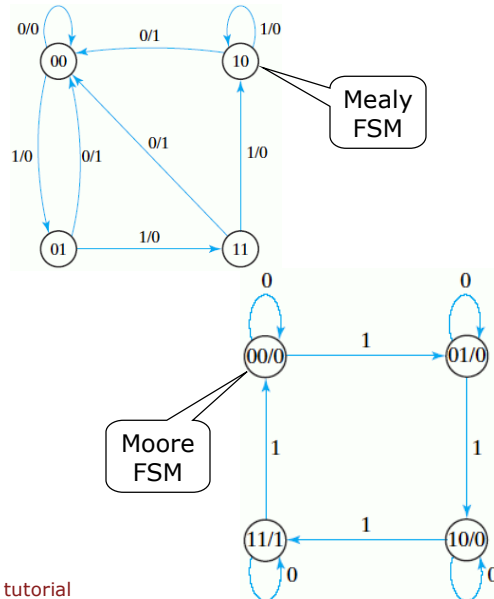
- State diagrams are representations of *Finite State Machines* (FSM)

- Mealy FSM**

- Output depends on input and state
- Output is not synchronized with clock
 - » can have temporarily unstable output

- Moore FSM**

- Output depends only on state



ECE 232

Verilog tutorial

Example 1: Sequence Detector

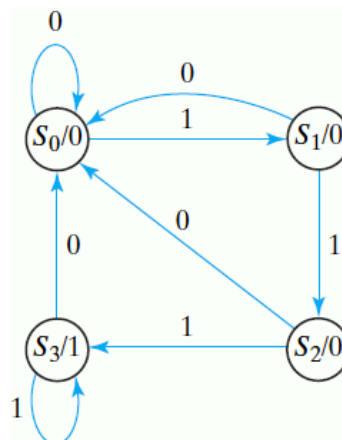
- Circuit specification:**
 - Design a circuit that outputs a 1 when three consecutive 1's have been received as input and 0 otherwise.

- FSM type**

- Moore or Mealy FSM?
 - » Both possible
 - » Chose Moore to simplify diagram

- State diagram:**

- » State S0: zero 1s detected
- » State S1: one 1 detected
- » State S2: two 1s detected
- » State S3: three 1s detected



ECE 232

Verilog tutorial

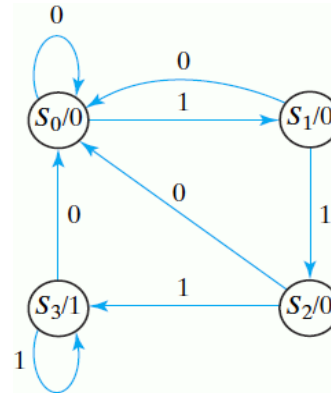
26

Sequence Detector: Verilog (Moore FSM)

```

module seq3_detect_moore(x,clk, y);
// Moore machine for a three-1s sequence detection
input x, clk;
output y;
reg [1:0] state;
parameter S0=2'b00, S1=2'b01, S2=2'b10,
S3=2'b11;
// Define the sequential block
always @(posedge clk)
    case (state)
        S0: if (x) state <= S1;
            else state <= S0;
        S1: if (x) state <= S2;
            else state <= S0;
        S2: if (x) state <= S3;
            else state <= S0;
        S3: if (x) state <= S3;
            else state <= S0;
    endcase
// Define output during S3
assign y = (state == S3);
endmodule

```



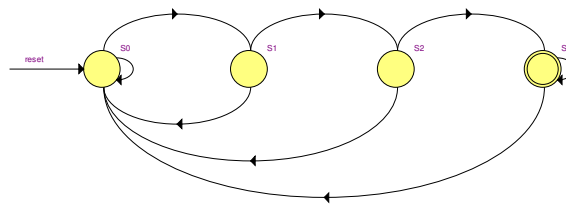
ECE 232

Verilog tutorial

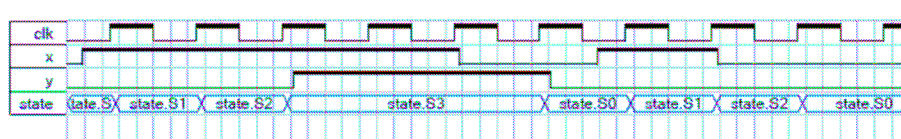
27

Sequence Detector: FSM Synthesis + Simulation

■ Synthesized Moore FSM (Quartus)



■ Simulation results (Quartus)



ECE 232

Verilog tutorial

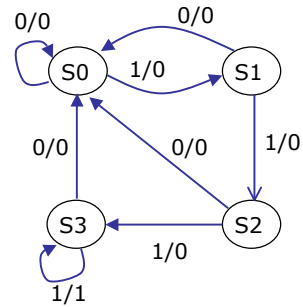
28

Sequence Detector: Verilog (Mealy FSM)

```

module seq3_detect_mealy(x,clk, y);
// Mealy machine for a three-1s sequence detection
input x, clk;
output y;
reg y;
reg [1:0] pstate, nstate; //present and next states
parameter S0=2'b00, S1=2'b01, S2=2'b10, S3=2'b11;
// Next state and output combinational logic
// Use blocking assignments "="
always @(x or pstate)
    case (pstate)
        S0: if (x) begin nstate = S1; y = 0; end
            else begin nstate = S0; y = 0; end
        S1: if (x) begin nstate = S2; y = 0; end
            else begin nstate = S0; y = 0; end
        S2: if (x) begin nstate = S3; y = 0; end
            else begin nstate = S0; y = 0; end
        S3: if (x) begin nstate = S3; y = 1; end
            else begin nstate = S0; y = 0; end
    endcase
// Sequential logic, use nonblocking assignments "<="
always @(posedge clk)
    pstate <= nstate;
endmodule

```



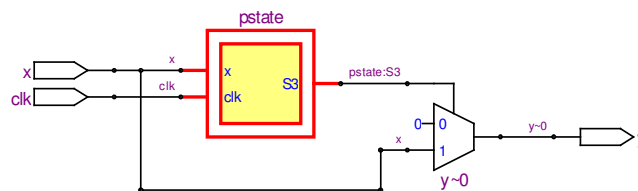
ECE 232

Verilog tutorial

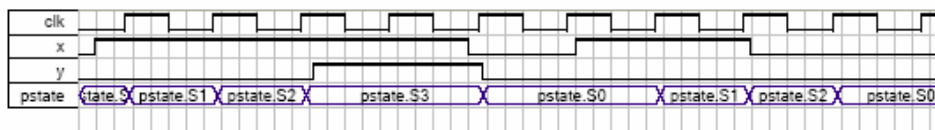
29

Sequence Detector: FSM Synthesis + Simulation

■ Synthesized Mealy FSM (Quartus)



■ Simulation results (Quartus)



ECE 232

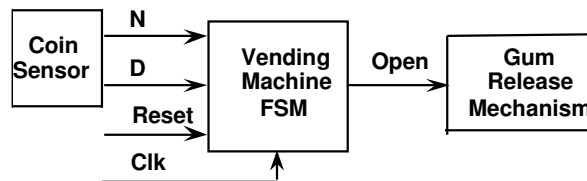
Verilog tutorial

30

Example 2: Vending Machine FSM - 1

Specify the Problem

- Deliver package of gum after 15 cents deposited
- Single coin slot for dimes, nickels
- No change
- Design the FSM using combinational logic and flip flops



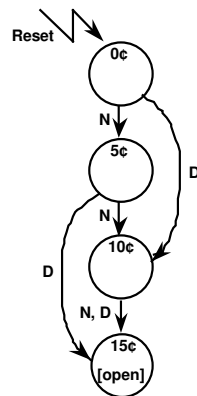
ECE 232

Verilog tutorial

31

Example 2: Vending Machine FSM - 2

State diagram



Reuse states
whenever possible

Present State	Inputs		Next State	Output Open
	D	N		
0¢	0	0	0¢	0
	0	1	5¢	0
	1	0	10¢	0
	1	1	X	X
5¢	0	0	5¢	0
	0	1	10¢	0
	1	0	15¢	0
	1	1	X	X
10¢	0	0	10¢	0
	0	1	15¢	0
	1	0	15¢	0
	1	1	X	X
15¢	X	X	15¢	1

Symbolic State Table

ECE 232

Verilog tutorial

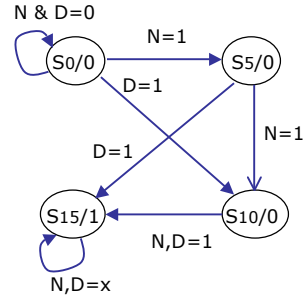
32

Vending Machine: Verilog (Moore FSM)

```
module vending_moore(N, D, clk, reset, open);
// Moore FSM for a vending machine
input N, D, clk, reset;
output open;
reg [1:0] state;
parameter S0=2'b00, S5=2'b01, S10=2'b10,
S15=2'b11;
```

[Synthesizing Moore FSM directly from state diagram](#)

```
// Define the sequential block
always @(posedge reset or posedge clk)
if (reset) state <= S0;
else
case (state)
S0: if (N) state <= S5;
    else if (D) state <= S10;
    else state <= S0;
S5: if (N) state <= S10;
    else if (D) state <= S15;
    else state <= S5;
S10: if (N) state <= S15;
    else if (D) state <= S10;
    else state <= S10;
S15: state <= S15;
endcase
// Define output during S3
assign open = (state == S15);
endmodule
```



ECE 232

Verilog tutorial

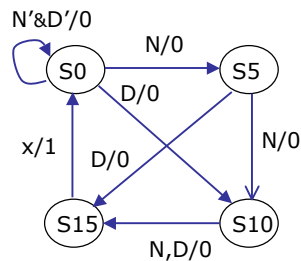
33

Vending Machine: Verilog (Mealy FSM)

```
module vending_mealy(N, D, clk, reset, open);
// Mealy FSM for a vending machine
input N, D, clk, reset;
output open;
reg [1:0] pstate, nstate;
reg open;
parameter S0=2'b00, S5=2'b01, S10=2'b10, S15=2'b11;
```

[Synthesizing Mealy FSM directly from state diagram](#)

```
// Next state and output combinational logic
always @(N or D or pstate or reset)
if (reset)
begin nstate = S0; open = 0; end
else case (pstate)
S0: begin open = 0; if (N) nstate = S5;
    else if (D) nstate = S10;
    else nstate = S0; end
S5: begin open = 0; if (N) nstate = S10;
    else if (D) nstate = S15;
    else nstate = S5; end
S10: if (N | D) begin nstate = S15; open = 0; end
    else begin nstate = S10; open = 0; end
S15: begin nstate = S0; open = 1; end
endcase
// FF logic, use nonblocking assignments "<="
always @(posedge clk)
pstate <= nstate;
endmodule
```



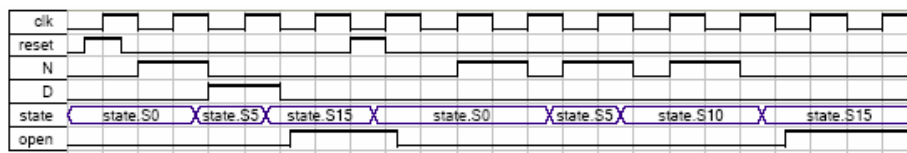
ECE 232

Verilog tutorial

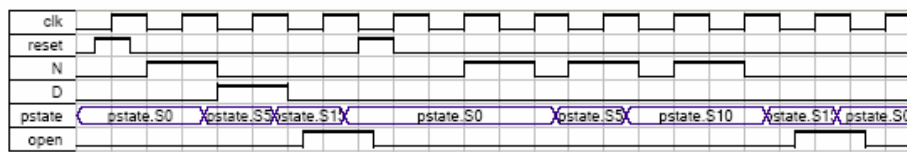
34

Vending Machine: Simulation Results

Moore FSM



Mealy FSM



ECE 232

Verilog tutorial

35

Summary

- Hardware description languages provide a valuable tool for computer engineers
- Any logic circuit (combinational or sequential) can be represented in HDL
- Circuits created using keywords and syntax
- Possible to use timing information
 - Explicit delay (#) is for simulation purposes only
 - Circuits with delays are not synthesizable
- Gate and RTL descriptions are possible
- Verilog is widely used in industry

ECE 232

Verilog tutorial

36