CSE1003 Digital LOGIC DESIGN

Verilog HDL(Hardware Description Language)

– Part 1 and 2

Overview of Verilog – Part 1

- **Objectives**
- **Verilog Basics**
  - **Notation**
  - **Keywords & Constructs**
  - **Operators**
- **Types of Descriptions**
  - **Structural**
  - **Dataflow**
    - **Boolean Equations**
    - **Conditions using Binary Combinations**
    - **Conditions using Binary Decisions**
  - **Behavioral & Hierarchical**
  - **Process (Procedural)**

Objectives

- **To become familiar with the hardware description language (HDL) approach to specifying designs**
  - **Be able to read a simple Verilog HDL description**
  - **Be able to write a simple Verilog HDL description using a limited set of syntax and semantics**
  - **Understanding the need for a "hardware view" when reading and writing an HDL**

Verilog Notation - 1

- **Verilog is:**
  - **Case sensitive**
  - **Based on the programming language C**
- **Comments**
  - **Single Line**

    // **[end of line]**
  - **Multiple Line**

    /*

    ***/**
- **List element separator: ,**
- **Statement terminator: ;**

Verilog Notation - 2

- **Binary Values for Constants and Variables**
  - **0**
  - **1**
  - **X,x - Unknown**
  - **Z,z – High impedance state (open circuit)**
- **Constants**
  - **n'b[integer]: 1'b1 = 1, 8'b1 = 000000001, 4'b0101= 0101, 8'bxxxxxxxx, 8'bxxxx = 0000xxxx**
  - **n'h[integer]: 8'hA9 = 10101001, 16'hf1= 0000000011110001**
- **Identifier Examples**
  - **Scalar: A,C,RUN,stop,m,n**
  - **Vector: sel[0:2], f[0:5], ACC[31:0], SUM[15:0], sum[15:0]**

Verilog Keywords & Constructs - 1

- **Keywords are lower case**
- **module – fundamental building block for Verilog designs**
  - **Used to construct design hierarchy**
  - **Cannot be nested**
- **endmodule – ends a module – not a statement => no “;”**
- **Module Declaration**
  - **module *module_name* (*module_port, module_port,* …) ;**
  - **Example: module full_adder (A, B, c_in, c_out, S);**

Verilog Keywords & Constructs - 2

- **Input Declaration**
  - **Scalar**
    - **input** ***list of input identifiers ;***
    - **Example: input A, B, c_in;**
  - **Vector**
    - **input** [*range*] ***list of input identifiers ;***
    - **Example: input[15:0] A, B, data;**
- **Output Declaration**
  - **Scalar Example: output c_out, OV, MINUS;**
  - **Vector Example: output[7:0] ACC, REG_IN, data_out;**

Verilog Keywords & Constructs - 3

- **Primitive Gates**
  - **buf, not, and, or, nand, nor, xor, xnor**
  - **Syntax: *gate_operator instance_identifier* (*output, input_1, input_2, …*)**
  - **Examples:**

```
and A1 (F, A, B); //F = A B
or O1 (w, a, b, c)
   O2 (x, b, c, d, e);    //w=a+b+c,x=b+c+d+e
```

Verilog Operators - 1

- **Bitwise Operators**
  - **~ NOT**
  - **& AND**
  - **| OR**
  - **^ XOR**
  - **^~ or ~^ XNOR**
- **Example:**

```
input[3:0] A, B;
output[3:0] Z ;
assign Z = A | ~B;
```

Verilog Operators - 2

- **Arithmetic Operators**

  +, -, (plus others)

- **Logical & Relational Operators**

  !, &&, | |, = =, !=, >=, <=, >, < (plus others)

- **Concatenation & Replication Operators**

  {identifier_1, identifier_2, …}

  {n{identifier}}

  - **Examples: {REG_IN[6:0],Serial_in},**

  {8 {1'b0}}

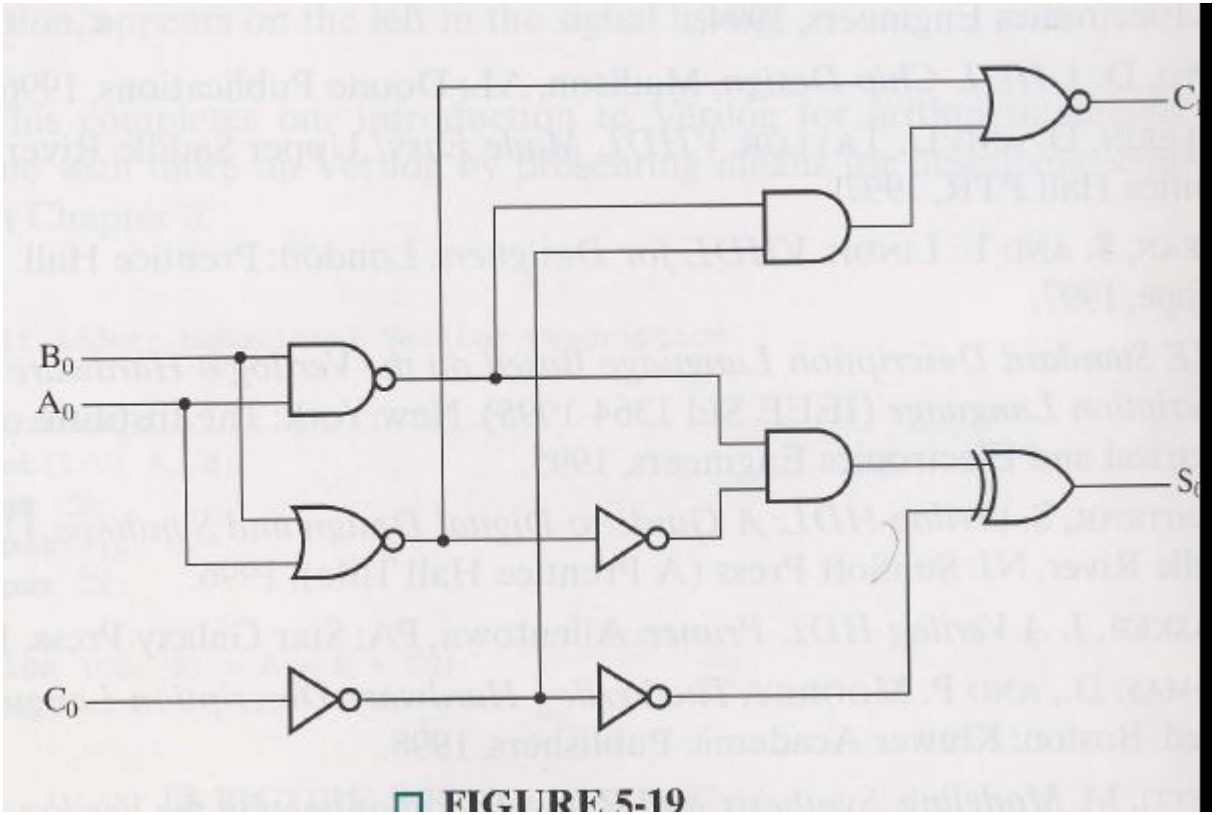


FIGURE 5-19

Structural Verilog

- **Circuits can be described by a netlist as a text alternative to a diagram - Example (See Figure 5-19 in text):**

```
module fig519s (A0, B0, C0, C1, S0);
  input A0, B0, C0;
  output C1, S0;
 //Seven internal wires needed
  wire[1:7] N;
 //Ports on primitive gates listed output port first
  not  G1 (N[3],C0), G2 (N[5],N[2]), G3 (N[6],N[3]);
  nand G4 (N[1],A0,B0);
  nor  G5 (N[2],A0,B0), G6 (C1,N[2],N[4]);
  and  G7 (N[4],N[1],N[3]), G8 (N[7],N[1],N[5]);
  xor  G9 (S0,N[6],N[7]);
endmodule
```

Dataflow Verilog - 1

- **Circuit function can be described by assign statements using Boolean equations (See Figure 5-19 in text):**

```
module fig519d (A0, B0, C0, C1, S0);
  input A0, B0, C0;
  output C1, S0;
  wire[1:2] N;
  assign N[1] = ~(A0 & B0); /*Note: Cannot write ~& for
    NAND */
  assign N[2] = ~(A0 | B0);
  assign C1 = ~((N[1] & ~C0) | N[2]);
  assign S0 = (~N[2] & N[1])^(~(~C0));
endmodule
```

Dataflow Verilog - 2

- **Circuit function can be described by assign statements using the conditional operator with binary combinations as in a truth table (See Figure 4-10 in text):**

```
module fig410dm (A, E_n, D_n);
  input[1:0] A;
  input E_n;
  output[3:0] D_n;
//Conditional: (X) ? Y: Z - if X is true, then Y,else Z
  assign D_n = {4{E_n}}&(
                    (A == 2'b00) ? 4'b1110:
                    (A == 2'b01) ? 4'b1101:
                    (A == 2'b10) ? 4'b1011:
                    (A == 2'b11) ? 4'b0111:
                    4'bxxxx);
endmodule
```

Dataflow Verilog - 3

- **Circuit function can be described by assign statements using the conditional operator for binary decisions on inputs(See Figure 4-10 in text):**

```
module fig410dc (A, E_n, D_n);
  input[1:0] A;
  input E_n;
  output[3:0] D_n;
/* Conditional: (X) ? Y: Z - if X is true, then Y,else Z */
  assign D_n = {4{E_n}} & (A[1] ? (A[0] ? 4'h7 : 4'hB): (A[0] ?
    4'hD : 4'hE));
endmodule
```

Behavioral & Hierarchical Verilog

- **Circuit function can be described by assign statements at higher than the logic level (See Figure 5-8 in text):**

```
module addsub (A, B, R, sub) ;
  input [3:0] A, B ;
  output [3:0] R ;//See Fig. 5-18 for carry out
  input sub ;
  wire [3:0] data_out;
  add A1 (A, data_out, sub, R);
  M1comp C1 (B, data_out, sub);
endmodule
```

Behavioral & Hierarchical Verilog

```
module add (X, Y, C_in, S);
  input [3:0] X, Y;
  input C_in;
  output [3:0] S;
  assign S = X + Y + {3'b0, C_in};
endmodule
module M1comp (data_in, data_out, comp);
  input[3:0] data_in;
  input comp;
  output [3:0] data_out;
  assign data_out = {4{comp}} ^ data_in;
endmodule
```

Overview of Verilog – Part 2

- **Process (Procedural) Description**
- **Verilog Keywords and Constructs**
- **Process Verilog for a Positive Edge-triggered D Flip-flop**
- **Process Verilog for State Diagram for Design Example**
- **Process Verilog for a Combinational Circuit**

Process (Procedural) Description

- So far, we have done dataflow and behavioral Verilog using continuous assignment statements (assign)
- Continuous assignments are limited in the complexity of what can be described
- A process can be viewed as a replacement for a continuous assignment statement that permits much more complex descriptions
- A process uses procedural assignment statements much like those in a typical programming language

Verilog Keywords & Constructs - 1

- **Because of the use of procedural rather than continuous assignment statements, assigned values must be retained over time.**
  - **Register type: reg**
  - **The reg in contrast to wire stores values between executions of the process**
  - **A reg type does not imply hardware storage!**
- **Process types**
  - **initial – executes only once beginning at t = 0.**
  - **always – executes at t = 0 and repeatedly thereafter.**
  - **Timing or event control is exercised over an always process using, for example, the @ followed by an event control statement in ( ).**

Verilog Keywords & Constructs - 2

- **Process begins with begin and ends with end.**
- **The body of the process consists of procedural assignments**
  - **Blocking assignments**
    - **Example: C = A + B;**
    - **Execute sequentially as in a programming language**
  - **Non-blocking assignments**
    - **Example: C <= A + B;**
    - **Evaluate right-hand sides, but do not make any assignment until all right-hand sides evaluated. Execute concurrently unless delays are specified.**

Verilog Keywords & Constructs - 3

- **Conditional constructs**
  - **The if-else**

```
If (condition)
   begin procedural statements end
{else if (condition)
   begin procedural statements end}
else
   begin procedural statements end
```

  - **The case**

```
case expression
   {case expression : statements}
endcase;
```

- **Syntax:** ***gate_operator instance_identifier (output, input_1, input_2, …)***
- **Examples:**

```
and A1 (F, A, B); //F = A B
or O1 (w, a, b, c)
    O2 (x, b, c, d, e);    //w=a+b+c,x=b+c+d+e
```

Examples

```
always
   begin
    B  = A;
    C  = B;
  end
```

- Suppose initially A = 0, B = 1, and C = 2. After execution, B = 0 and C = 0.

```
always
   begin
       B <= A;
       C <= B;
   end
```

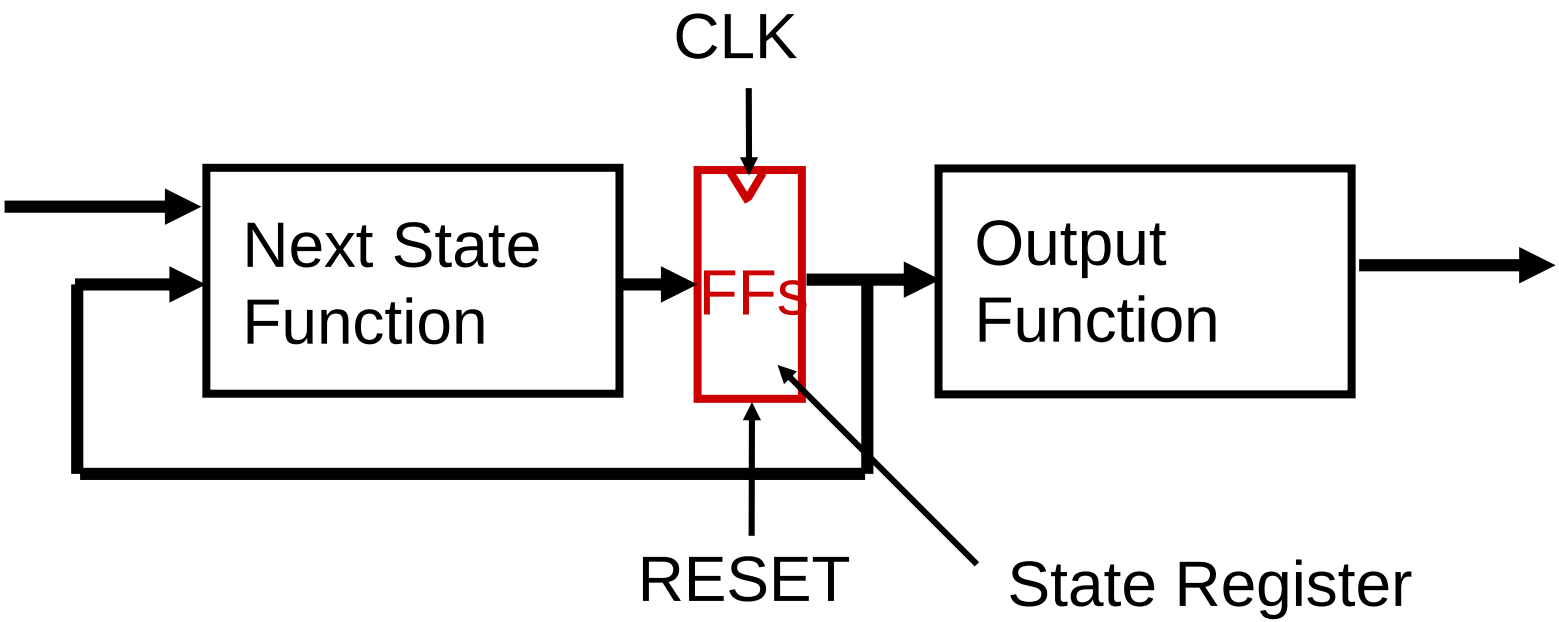
- Suppose initially A = 0, B = 1, and C = 2. After execution, B = 0 and C = 1.

Verilog for Positive Edge-Triggered D Flip-Flop

```
module dff (CLK, RESET, D, Q)
    input CLK, RESET, D;
    output Q;
    reg Q;
    always@ (posedge CLK or posedge RESET)
        begin
            if (RESET)
              Q <= 0;
            else
              Q <= D;
        end
endmodule
```

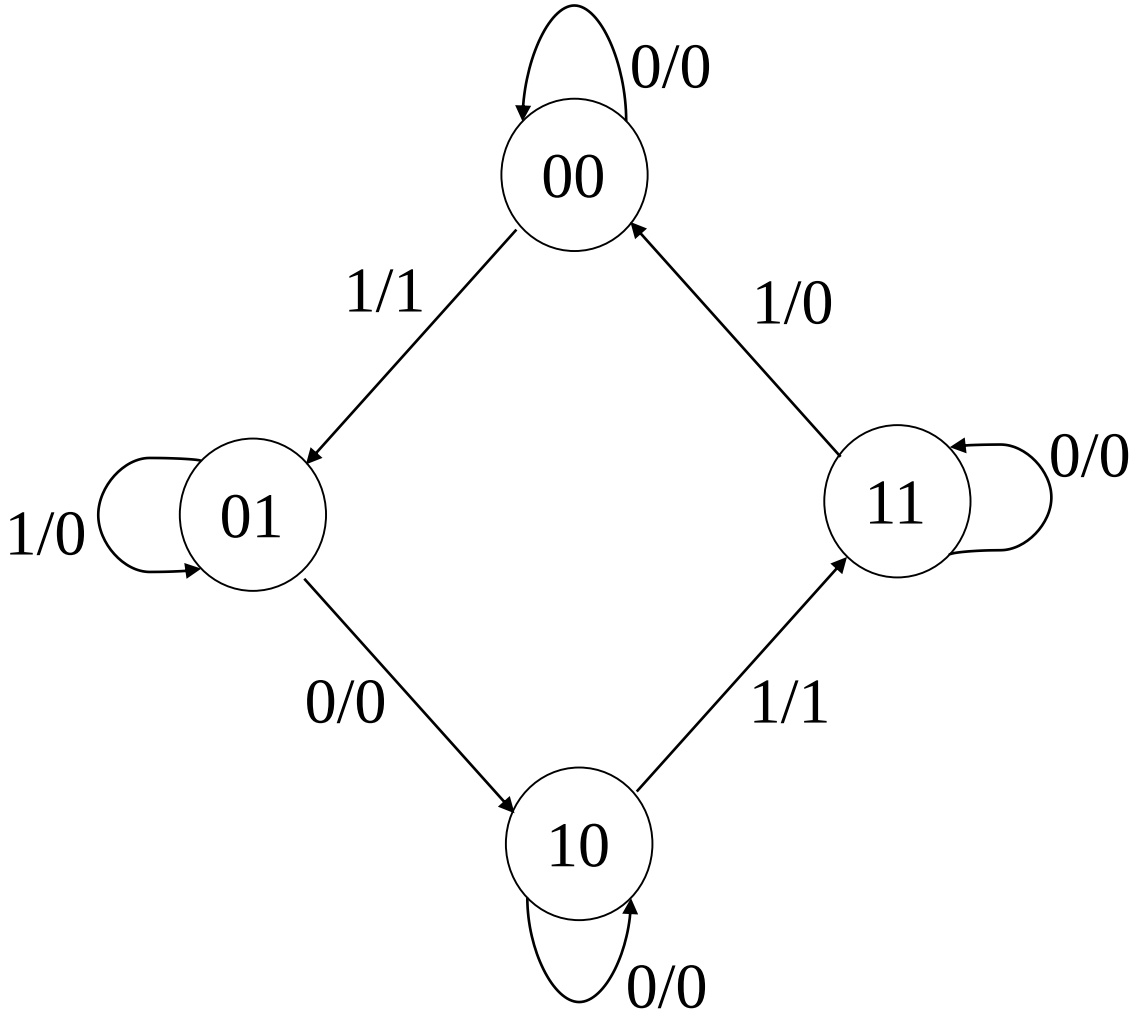
Describing Sequential Circuits

- **There are many different ways to organize models for sequential circuits. We will use a model that corresponds to the following diagram:**



- **A process corresponds to each of the 3 blocks in the diagram.**

State Diagram for Design Example



Verilog for State Diagram for Example

```
module Diag (CLK, RESET, X, Y)
    input CLK, RESET, X;
    output Y;
    reg[1:0] state, next state;
//state register
    always@(posedge CLK or posedge RESET)
          begin
                if (RESET == 1)
                   state <= 2'b00;
                else
                   state <= next_state;
          end
```

Verilog State Diagram (continued)

```
//next state function
    always@(X or state)
        begin
          case (state)
                2'b00: if (X == 1) next_state <= 2'b01;
                     else next_state <= 2'b00;
                2'b01: if (X == 1) next_state <= 2'b01;
                          else next_state <= 2'b10;
                2'b10: if (X == 1) next_state <= 2'b11;
                          else next_state <= 2'b10;
                2'b11: if (X == 1) next_state <= 2'b00;
                     else next_state <= 2'b11;
                default: next_state <= 2'bxx;
        end
```

Verilog State Diagram (continued)

```
//output function
    always@(X or state)
        begin
          case (state)
                2'b00: if (X == 1) Z <= 1'b1;
                     else Z <= 1'b0;
                2'b01: Z <= 0;
                2'b10: if (X == 1) Z <= 1'b1;
                           else Z <= 1'b0;
                2'b11: Z <= 0;
                default: Z <= 1'bx;
        end
endmodule
```