Overview of Chapter 4

- Part 1: Types of Sequential Circuits
  - Storage Elements
  - Latches and Flip-Flops
- Part 2: Sequential Circuit Analysis
  - State Tables and State Diagrams
- Part 3: Sequential Circuit Design
  - Assignment of State Codes
  - Implementation
- Part 4: HDL Representation

HDL Representation: Verilog – Part 2

- Process (Procedural) Description
- Verilog Keywords and Constructs
- Process Verilog for a Positive Edge-triggered D Flip-flop
- Process Verilog for Figure 4-23
- 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 descriptive complexity
- 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.
  - Multiple non-blocking assignments 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;
    ```

Examples

```verilog
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

```verilog
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:

![Next State Function](image)

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

Verilog for Figure 4-23 State Diagram

```verilog
module fig423 (CLK, RESET, X, Y)
    input CLK, RESET, X;
    output Y;
    reg[1:0] state, next_state;
    reg[1:0] state_register;
    always@ (posedge CLK or posedge RESET)
        begin
            if (RESET == 1)
                state <= 2'b00;
            else
                state <= next_state;
        end
    //next state function
    always @ (X or state)
        begin
            case (state)
                2'b00: if (X == 1) next_state <= 2'b10;
                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;
            endcase
        end
endmodule
```
Verilog State Diagram (continued)

```verilog
// 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;
  endcase
endmodule
```