Chapter 14
Verilog Hardware Description Language

Procedural Modeling of Clocked Sequential Circuits

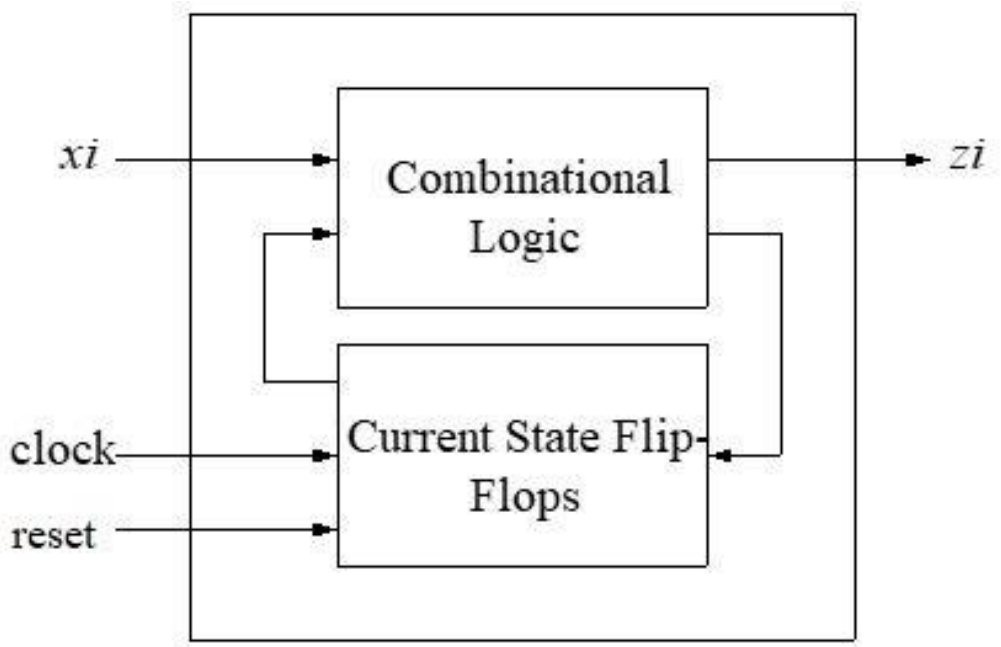

Figure 14.18 Model of a finite state machine.

14.6 Explicit Style of Describing Finite State Machines

In explicit style of describing finite state machines, a case statement is used to specify the actions in each of the machine's states and the transition between states. An example state diagram is shown in Figure 14.21, where six states and their state transitions are shown with one input and three output bits specified. Figure 14.22 shows the Verilog description of this finite state machine.

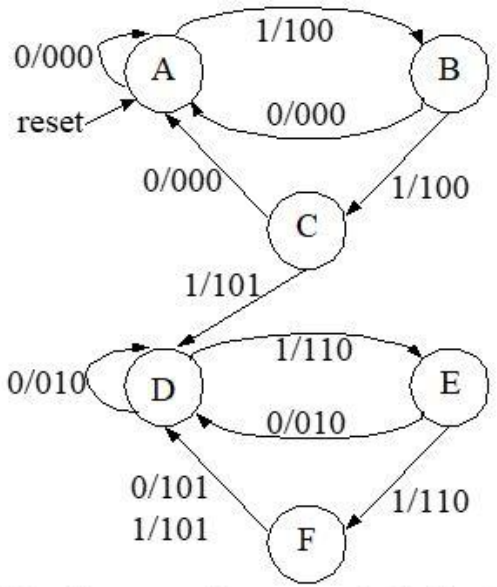

Figure 14.21 State diagram of an example finite state machine.

The first always statement is a description of the combinational output (*out*) and next state (*nextState*) functions. The input set for these functions contains the input *i* and the register *currentState*. Any change on either of these will cause the always statement to be re-evaluated. The single statement within the always is a case statement indicating the actions to be performed in each state. The controlling expression for the case is the state variable (*currentState*). Thus, depending on what state the machine is in, only the specified actions occur. In each case item, the two combinational functions being computed (*out* and *nextState*) are assigned to using conditional operator. In addition, a default case is listed representing the remaining unassigned states. The default sends the machine to state *A* which is equivalent to a reset. By arbitrary choice, *out* is set to don't care in the unassigned states.

This always statement will result in combinational logic, because the sensitivity list contains all of the input set, there are no edge specifiers in the sensitivity list, and for every control path, both of the combinational outputs have been assigned to.

In the second always statement, *reset* is asserted low and will cause the machine to go into state *A*. If *reset* is not asserted, then the normal action of the always will be to load *currentState* with the value of *nextState*, changing the state of the finite state machine on the positive edge of *clock*.

The localparam statement specifies the state assignment for the system. Since these are treated as constants, they cannot be directly overridden by instantiation.

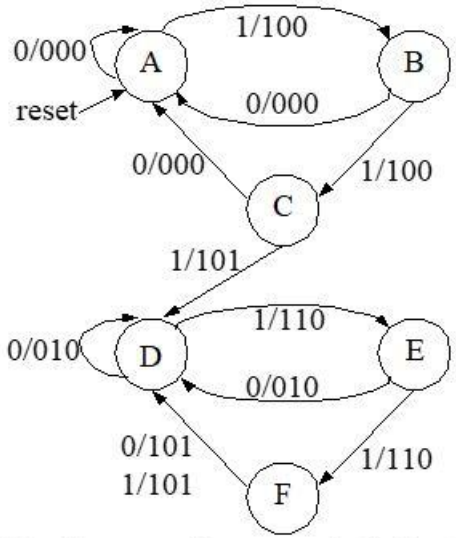

Figure 14.21 State diagram of an example finite state machine.

```
module fsm
    (input              i, clock, reset,
    output reg [2:0]    out);

    reg    [2:0]   currentState, nextState;

    localparam   [2:0]   A = 3'b000,   // The state labels and their assignments
                         B = 3'b001,
                         C = 3'b010,
                         D = 3'b011,
                         E = 3'b100,
                         F = 3'b101;

always @(*)
        case (currentState)
                A:      begin
                                nextState = (i == 0) ? A : B;
                                out = (i == 0) ? 3'b000 : 3'b100;
                        end
                B:      begin
                                nextState = (i == 0) ? A : C;
                                out = (i == 0) ? 3'b000 : 3'b100;
                        end
                C:      begin
                                nextState = (i == 0) ? A : D;
                                out = (i == 0) 3'b000 : 3'b101;
                        end
                D:      begin
                                nextState = (i == 0) : D : E;
                                out = (i == 0) ? 3'b010 : 3'b110;
                        end
                E:      begin
                                nextState = (i == 0) ? D : F;
                                out = (i == 0) ? 3'b010 : 3'b110;
                        end
                F:      begin
                                nextState = D;
                                out = 3'b101;
                        end
                default: begin  // undefined states. Go to state A
                                nextState = A;
                                out = 3'bxxx;
                        end
        endcase

always @ (posedge clock, negedge reset)  // The state register
        if (~reset)
                currentState <= A;  // the reset state
        else
                currentState <= nextState;
endmodule
```