Digital System Design with HDL (I)

Lecture 12

Dr. Ming Xu and Dr. Kain Lu Low

Dept of Electrical & Electronic Engineering

XJTLU

Algorithmic State Machine Design

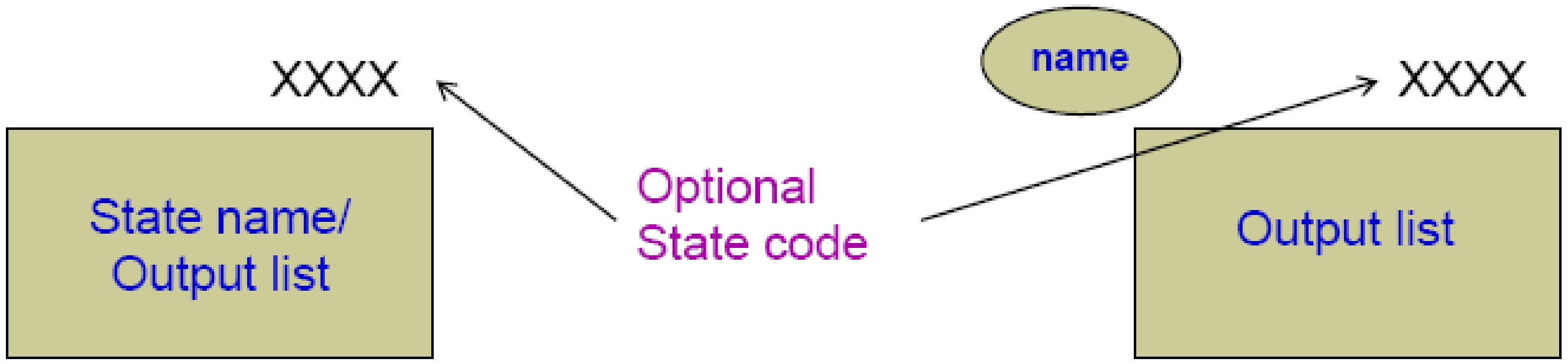
- As an alternative to using state graphs, a **state machine flow chart** or **SM chart** may be used to describe the behaviour of a state machine.
- The ASM chart is a flowchart which resembles to the conventional software flowchart.
- The ASM chart expresses the concept of a **sequence of time intervals** in a precise way.
- The software flowchart describes only the **sequence of events** and not their duration.

States and Clock

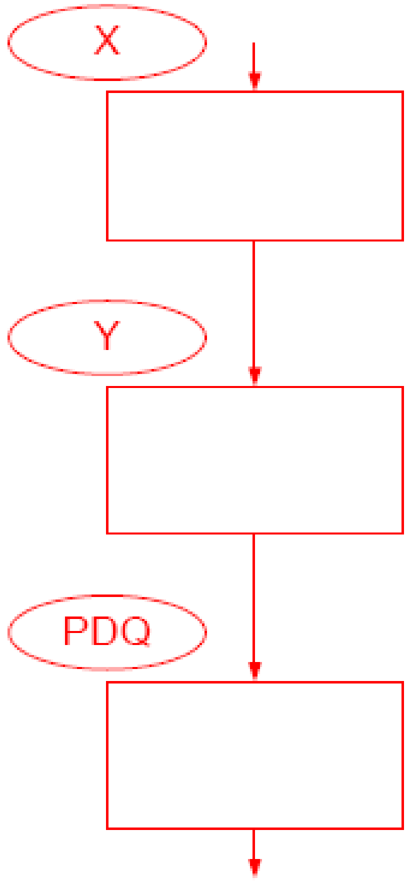
- The algorithmic state machine (ASM) moves through a sequence of states.
- It is the task of the **present state** of the system to:
  1. Produce any required **output signals**.
  2. To use appropriate input information to move, at the proper time, to the **next state**.
- In synchronous systems the state times are determined solely by the *master clock*.

States

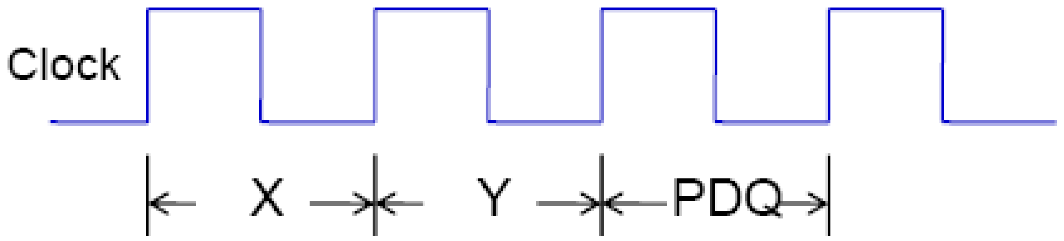
- Each active transition of a clock causes a change of state from the *present* state to the *next* state.
- The symbol for a state is a rectangle with its symbolic name enclosed in a small circle (or oval) at the upper left corner.
- The outputs are written inside the state box.



Sequential ASMs

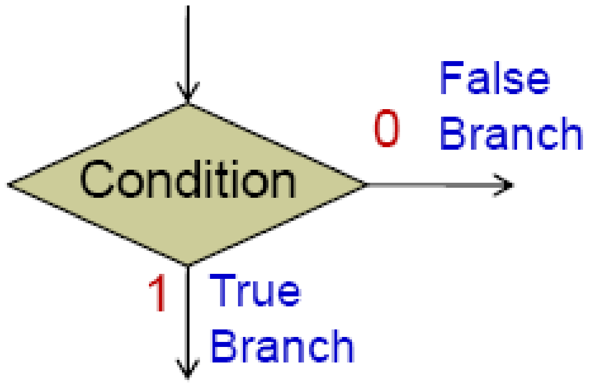


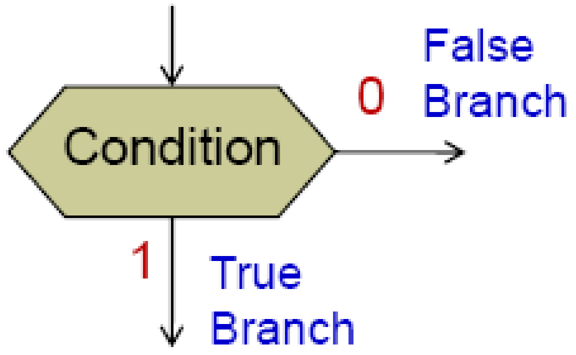
- We could represent a purely sequential algorithm as an ASM chart of a sequence of states.
- The timing diagram for the above sequence of states is as follows.
- You should think of time as rigorously implied in the ASM chart notation.



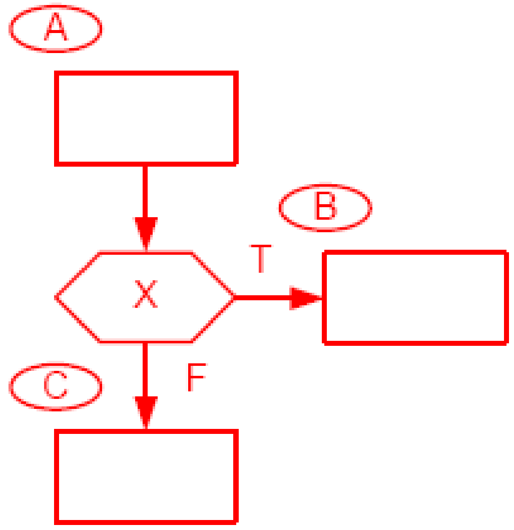
Branches

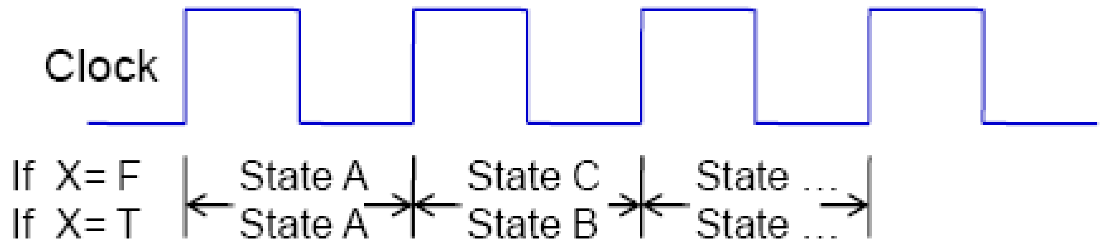
- We need to express conditional branches so that the next state is determined not only by the present state but also by one or more test (status) inputs.
- The symbol is the same as in conventional flowcharts for software: the diamond or diamond-sided rectangle.





Branches –cont.

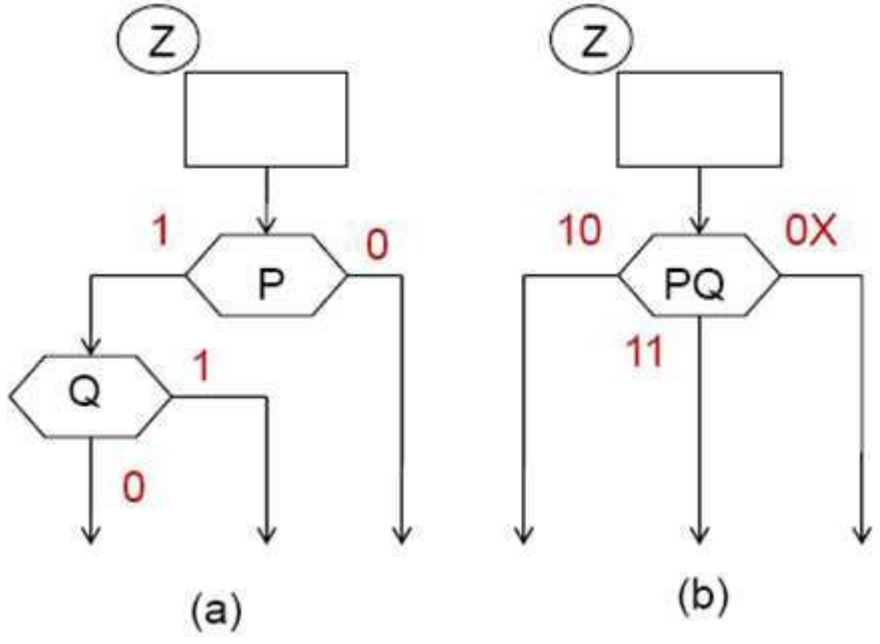




- The decision to jump to either state B or state C is made during state A and the jump occurs at the end of stateA.
- The test does not require a separate clock period, it is done in parallel with the actions of the parent state rectangle and thus is part of the parent state.

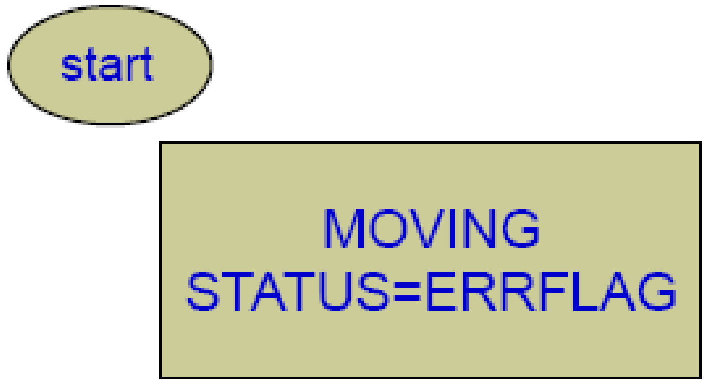
Multi-Way Branches

- We may draw a sequence of diamonds or have more than two paths coming from the same diamond.
- Figure (a) conveys the wrong feeling that the test of variable P is of a higher priority than the test of Q.
- For every valid combination of the input variables, there must be exactly one exit path defined.



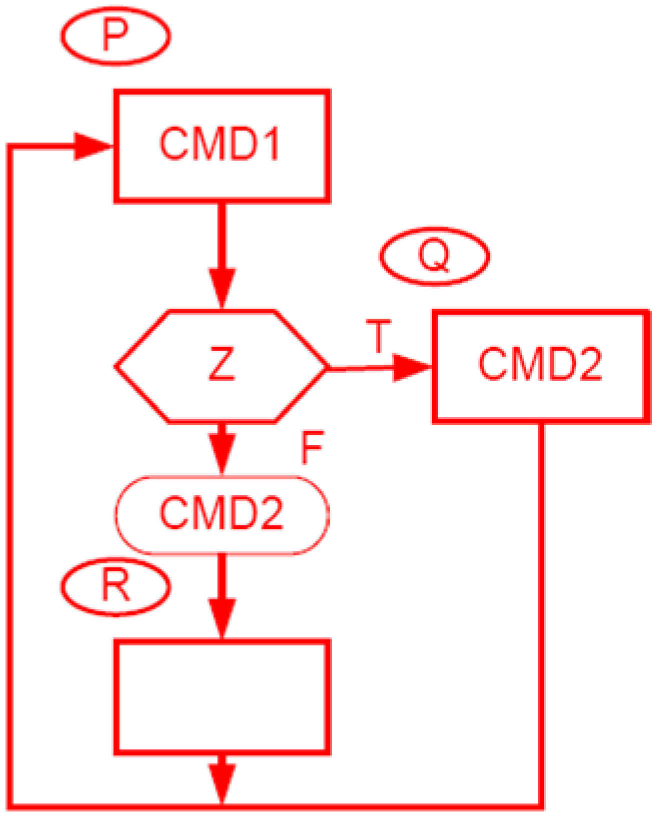
Outputs

- To indicate an output, a command description is placed within the appropriate state rectangle.
- The first line, MOVING, calls for the assertion of the signal MOVING, during the state i.e. MOVING = TRUE.
- The last line means that the output STATUS is to have the value of the variable ERRFLAG(Tor F) during this state.



Conditional Outputs

- Sometimes we want a command to occur only when some other condition exists.
- We call such a command a **conditional output** and specify it with an oval.
- CMD2 will occur for one state time whenever the ASM is in state Q. When in state P, CMD2 will occur if test input Z is false.
- CMD2 is an unconditional output in state Q and a conditional output in state P.

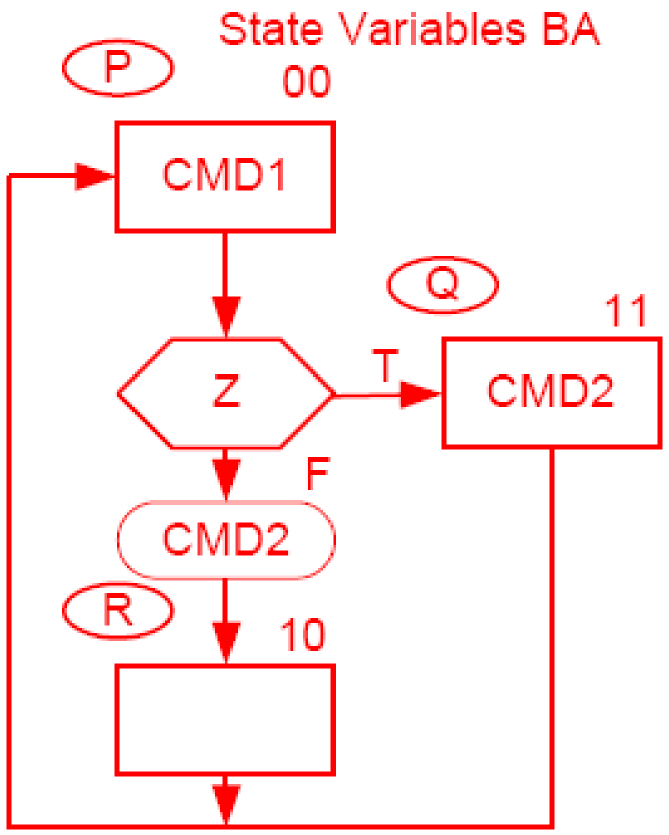


Summary of ASM Symbols

- Test inputs may serve two functions in ASM charts:
  - They may help specify the next state
  - They may control the issuing of conditional outputs.
- Ovals for conditional outputs and diamonds for test inputs belong to the parent state; since the activities occur **concurrently** during the state time.
- A state thus consists of its rectangle, which is always present, and any test diamonds and conditional output ovals associated with that state.
- Unconditional outputs are a function only of the parent state. Conditional outputs depend on both the state and the path within the state.

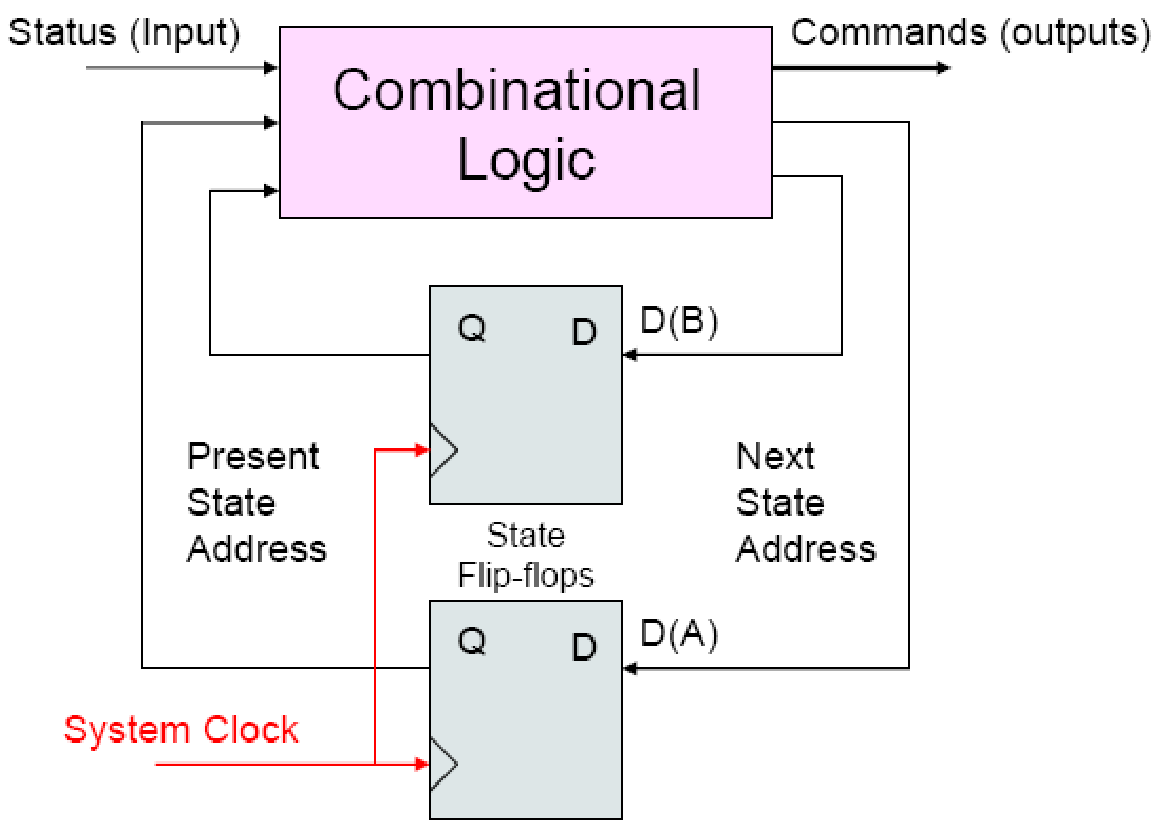
Traditional Design Implementation

- Use flip-flops as state memory (either D, T, or J-K types)
- There are two ways to represent the present state in flip-flop memory
  - Assign a unique binary number to each state (**Encoding Method**).
  - Assign one flip-flop to each state (**One Hot Method**).
- Using the **Encoding method**, two state variables are required for encoding 3 states.
- The state assignment is arbitrary.

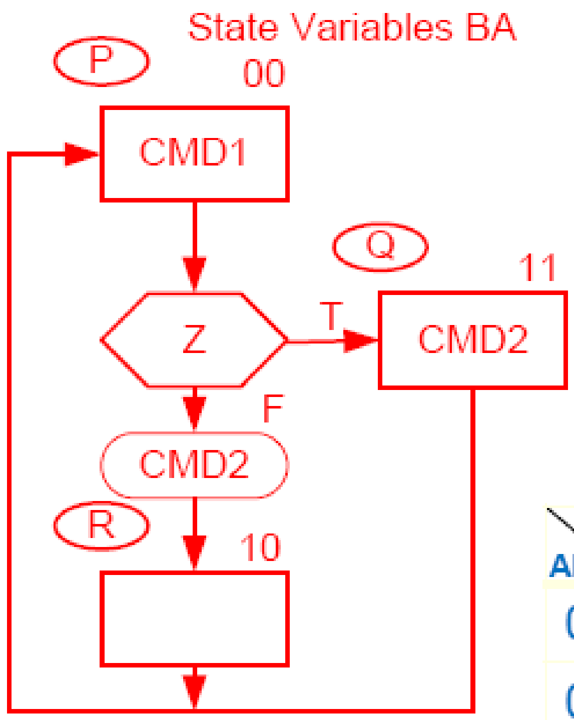


Process Model

Given the present state we need to compute the new state code to load into the state flip flops.



State Transition Table



Present State	Input	Next State	Outputs	
AB	**Z**	**D(A)D(B)**	**CMD1**	**CMD2**
00	0	10	1	1
00	1	11	1	0
11	-	00	0	1
10	-	00	0	0
01	-	-	-	-

CMD1 =A'

AB \ Z	0	1
00	1	1
01	X	X
11	0	0
10	0	0

D(A) = A'

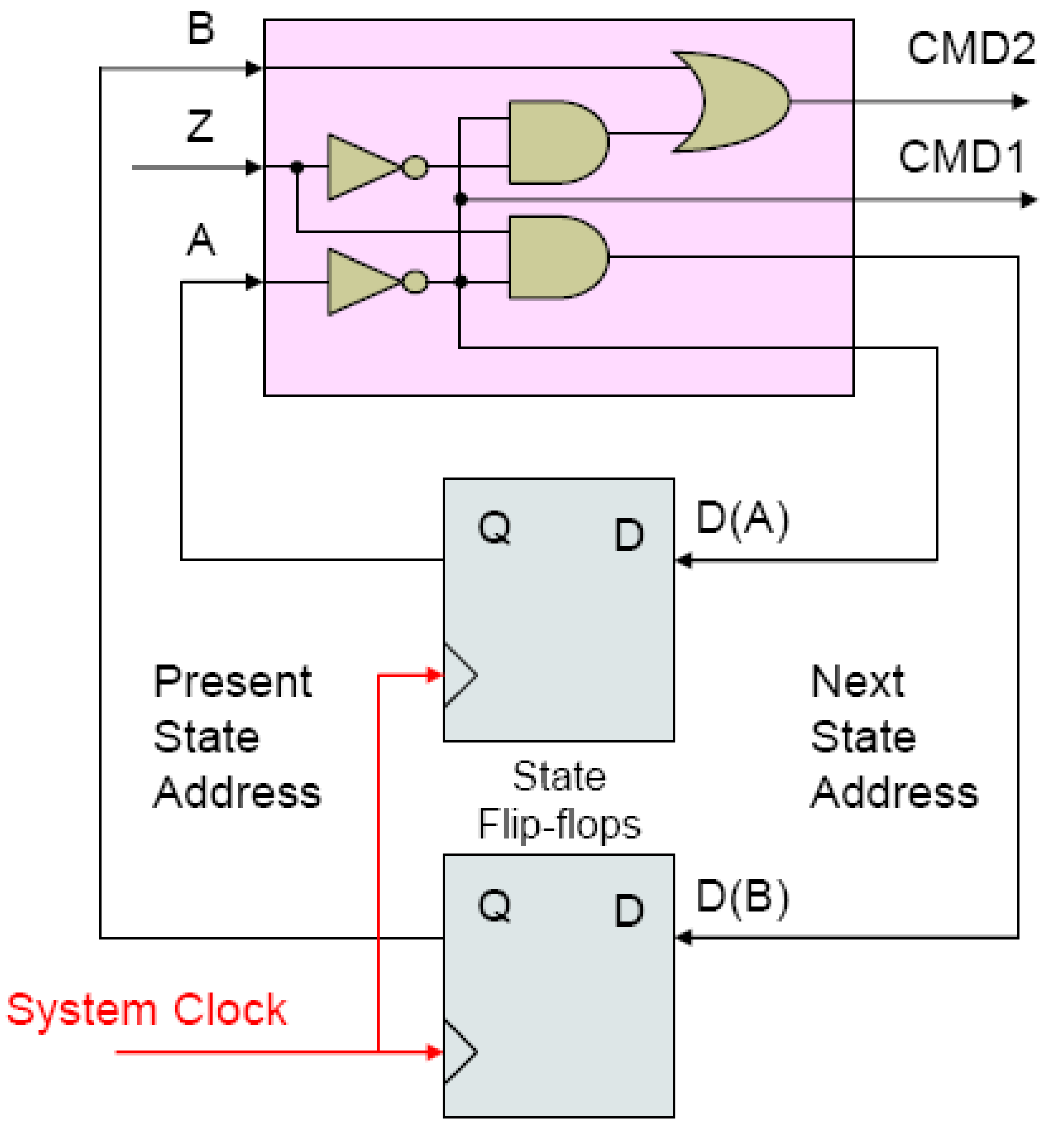
AB \ Z	0	1
00	0	1
01	X	X
11	0	0
10	0	0

D(B) = A'Z

AB \ Z	0	1
00	1	0
01	X	X
11	1	1
10	0	0

CMD2 =B+A'Z'

ASM Implementation



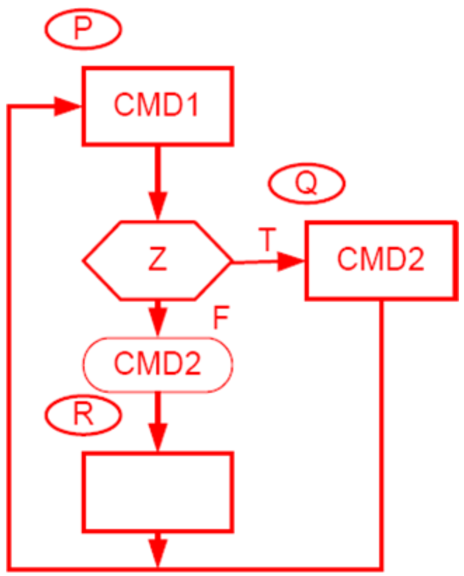
D(A) = A’

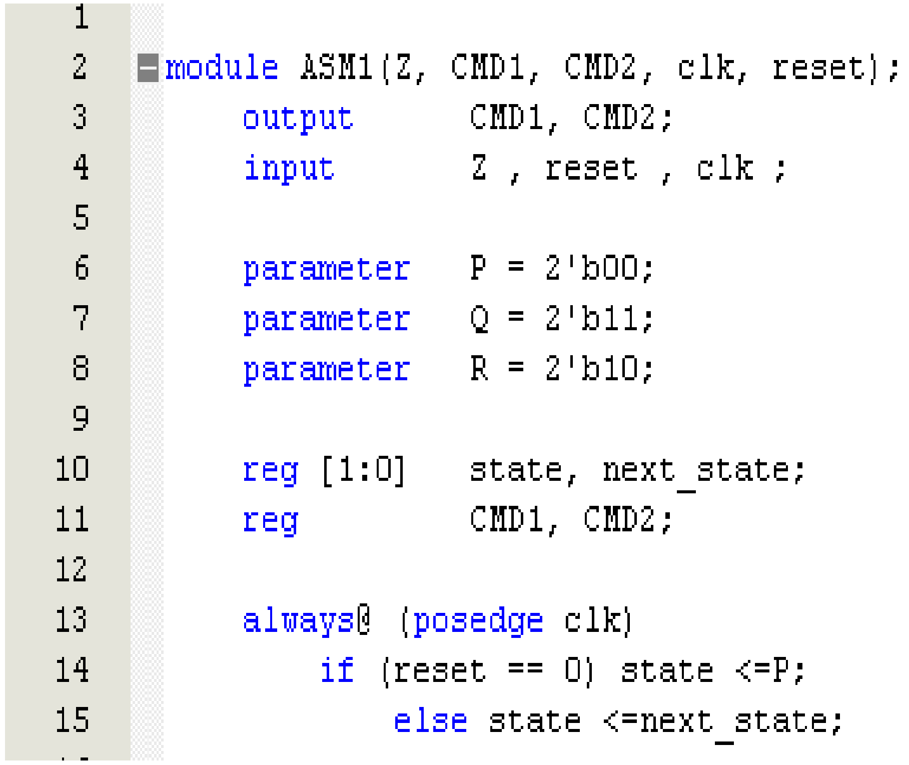
D(B) = A’Z

CMD2 =B+A’Z’

CMD1 =A’

Verilog Code



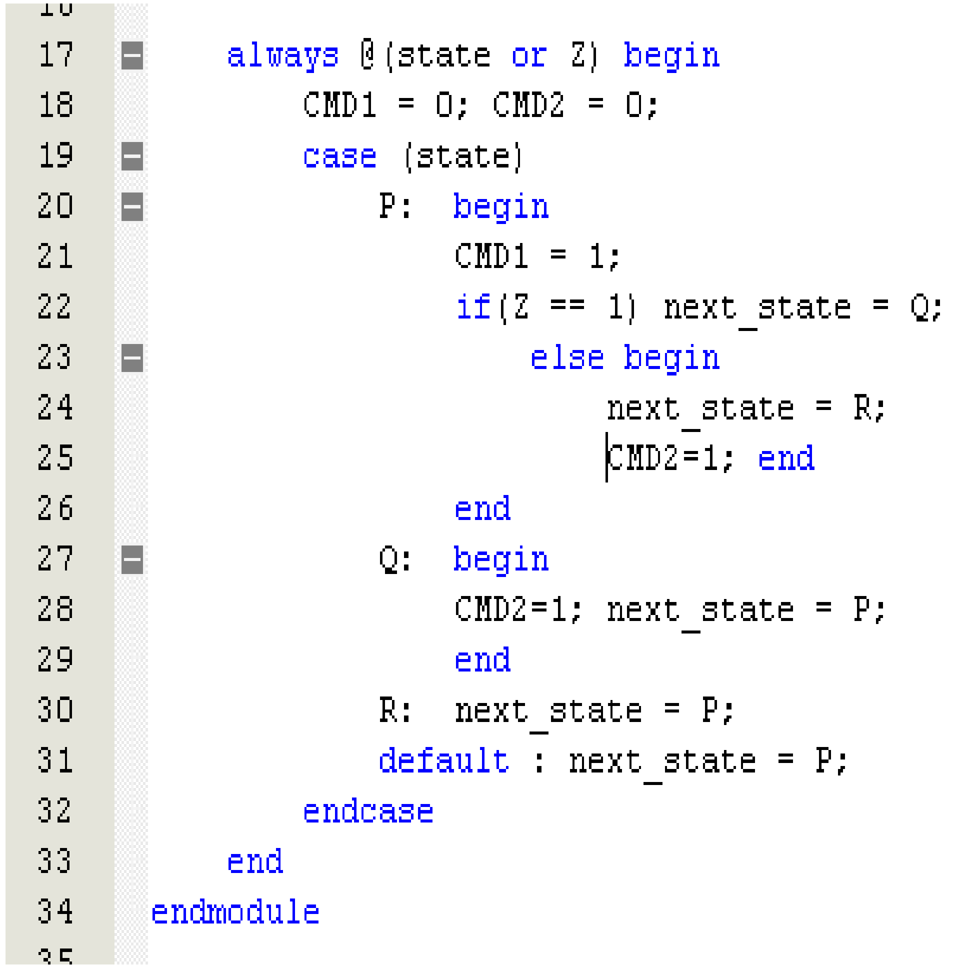


```
module ASM1(Z, CMD1, CMD2, clk, reset);
    output      CMD1, CMD2;
    input       Z , reset , clk ;

    parameter   P = 2'b00;
    parameter   Q = 2'b11;
    parameter   R = 2'b10;

    reg [1:0]   state, next_state;
    reg         CMD1, CMD2;

    always@ (posedge clk)
        if (reset == 0) state <=P;
            else state <=next_state;
```



```
    always @(state or Z) begin
        CMD1 = 0; CMD2 = 0;
        case (state)
            P:  begin
                CMD1 = 1;
                if(Z == 1) next_state = Q;
                    else begin
                        next_state = R;
                        CMD2=1; end
                end
            Q:  begin
                CMD2=1; next_state = P;
                end
            R:  next_state = P;
            default : next_state = P;
        endcase
    end
endmodule
```