Example 8.2: Simple FSM #1

Figure 8.4 shows the states diagram of a very simple FSM. The system has two states (stateA and stateB), and must change from one to the other every time d = '1' is received. The desired output is x = a when the machine is in stateA, or x = b when in stateB. The initial (reset) state is stateA.

A VHDL code for this circuit, employing design style #1, is shown below.

```
1  ------------------------------------------
2  ENTITY simple_fsm IS
```

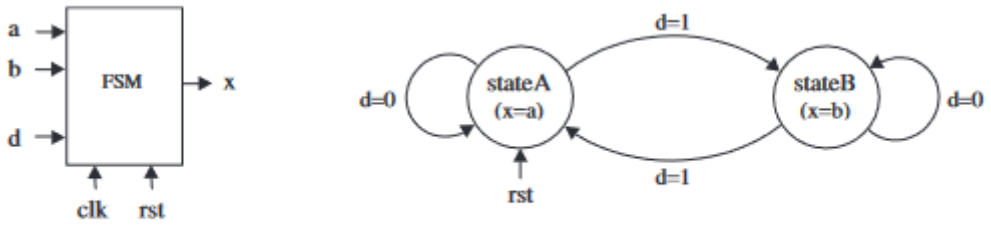

Figure 8.4
State machine of example 8.1.

```
3     PORT ( a, b, d, clk, rst: IN BIT;
4            x: OUT BIT);
5  END simple_fsm;
6  ------------------------------------------
7  ARCHITECTURE simple_fsm OF simple_fsm IS
8     TYPE state IS (stateA, stateB);
9     SIGNAL pr_state, nx_state: state;
10 BEGIN
11    ----- Lower section: ----------------------
12    PROCESS (rst, clk)
13    BEGIN
14       IF (rst='1') THEN
15          pr_state <= stateA;
16       ELSIF (clk'EVENT AND clk='1') THEN
17          pr_state <= nx_state;
18       END IF;
19    END PROCESS;
20    ---------- Upper section: -----------------
21    PROCESS (a, b, d, pr_state)
22    BEGIN
23       CASE pr_state IS
24          WHEN stateA =>
25             x <= a;
26             IF (d='1') THEN nx_state <= stateB;
27             ELSE nx_state <= stateA;
28             END IF;
29          WHEN stateB =>
30             x <= b;
31             IF (d='1') THEN nx_state <= stateA;
32             ELSE nx_state <= stateB;
33             END IF;
34       END CASE;
35    END PROCESS;
36 END simple_fsm;
37 ------------------------------------------
```

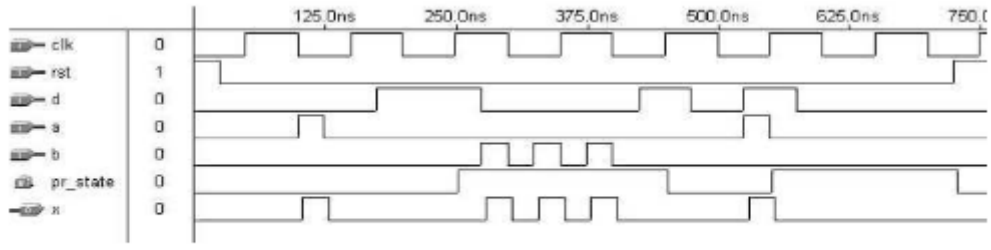

Figure 8.5
Simulation results of example 8.2

Simulation results relative to the code above are shown in figure 8.5. Notice that the circuit works as expected. Indeed, looking at the report files, one will verify that, as expected, only one flip-flop was required to implement this circuit because there are only two states to be encoded. Notice also that the upper section is indeed combinational, for the output (x), which in this case does depend on the inputs (a or b, depending on which state the machine is in), varies when a or b vary, regardless of clk. If a synchronous output were required, then design style #2 should be employed.