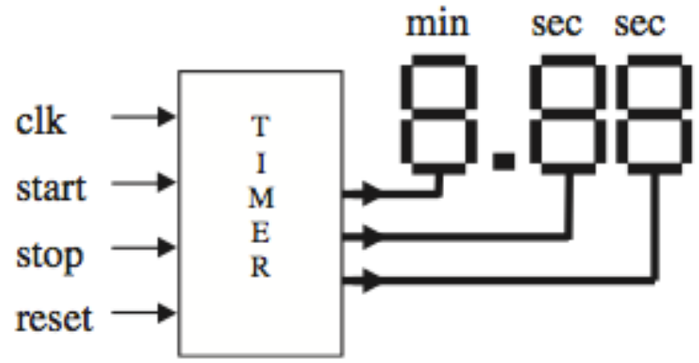
CPE 462
VHDL: Simulation and Synthesis

Topic #07 - c) Finite State Machines in Spartan-3E

Roadmap for next classes

- This week: FSM exercise (Today), Packages (W) and Components (F)
  - HW: Hardware FSM (wednesday)
- Next week (Dec. 5, 7 and 9): Function and Procedures
  - HW: Package and Components (monday), Functions (friday)
- Final week (Dec. 12 and 14): Interesting VHDL System Designs (E.g: Random Numbers and Neural Networks)
  - HW: Procedures (monday)
  - Project Exam: Will be given out on Dec. 12 and its due on Dec. 16 at 11:59pm.

Our exercise



- Implement in our FPGA boards a timer capable of running from 0min:00sec to 9min:59sec.
- The circuit must have only two switches, one which must perform the start/stop and the other which performs the circuit reset.
- The outputs must be seven segment display (SSD) coded.
- The input CLK is the reliable 50MHz clock signal.
- It is a requirement for this assignment to use a FSM to address the SSD time division multiplexing issues of our FPGA board.

7-Segment Displays (SSD)

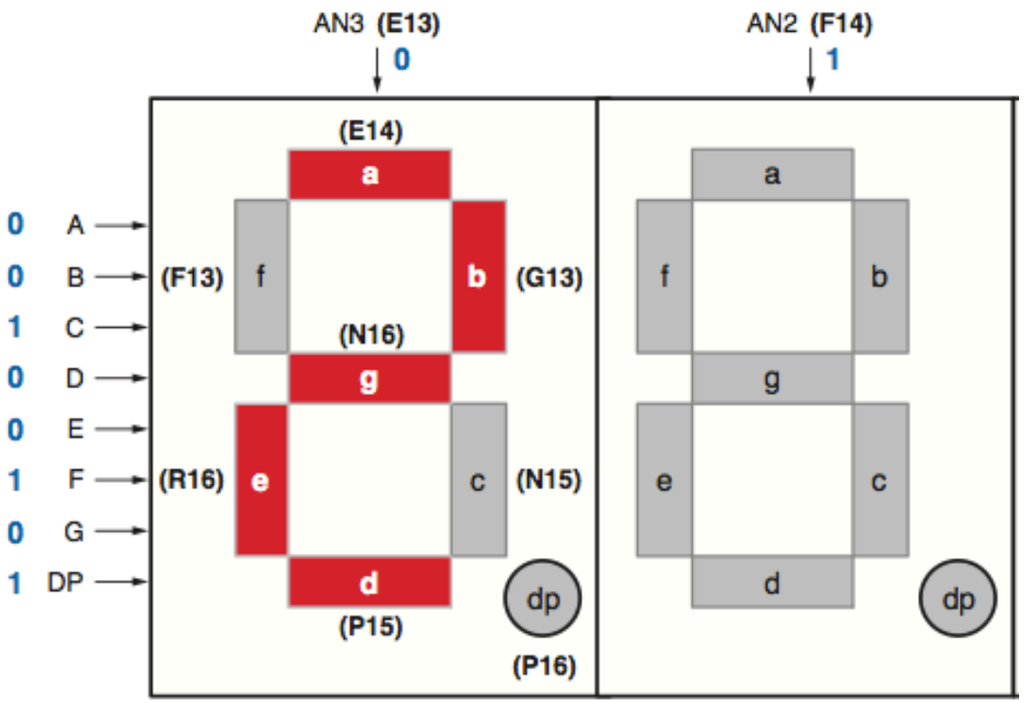


Table 3-1: **FPGA Connections to Seven-Segment Display (Active Low)**

Segment	FPGA Pin
A	E14
B	G13
C	N15
D	P15
E	R16
F	F13
G	N16
DP	P16

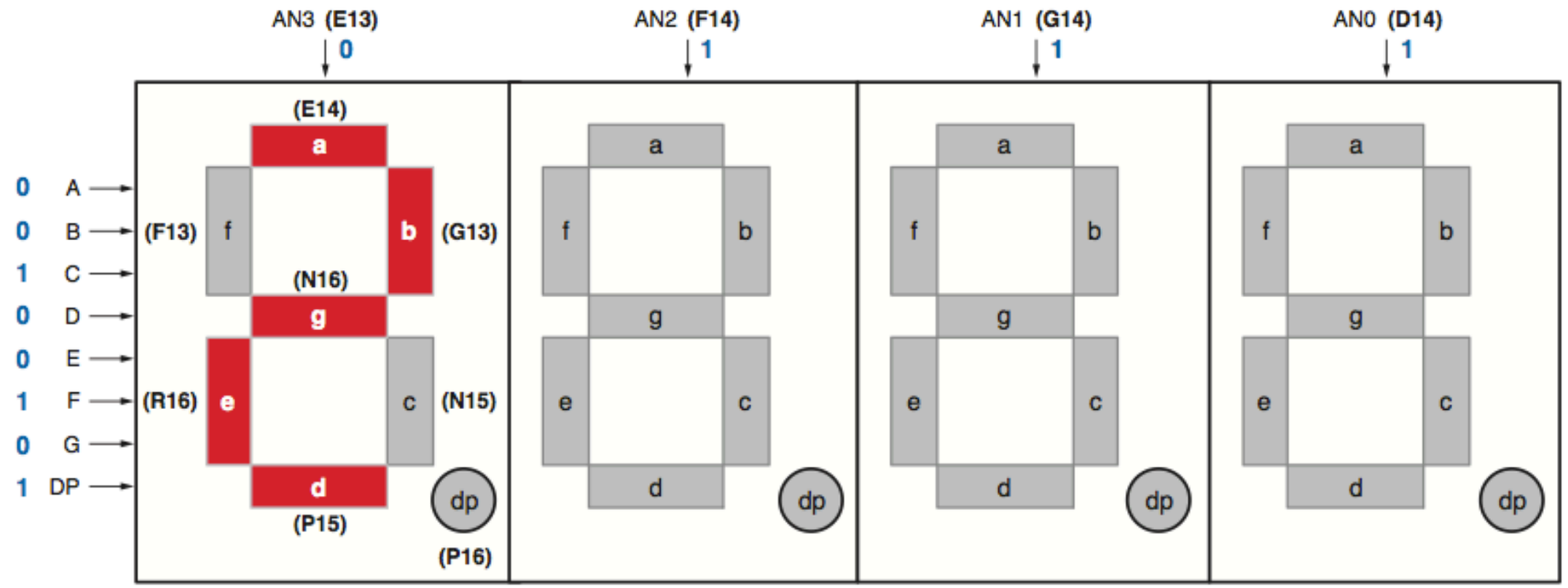
Table 3-2: **Digit Enable (Anode Control) Signals (Active Low)**

Anode Control	AN3	AN2	AN1	AN0
FPGA Pin	E13	F14	G14	D14

Using the Spartan3 seven segment display

The seven segment display are time-multiplexed.

This means, with the same pins, we can address each of the four seven-segment displays.



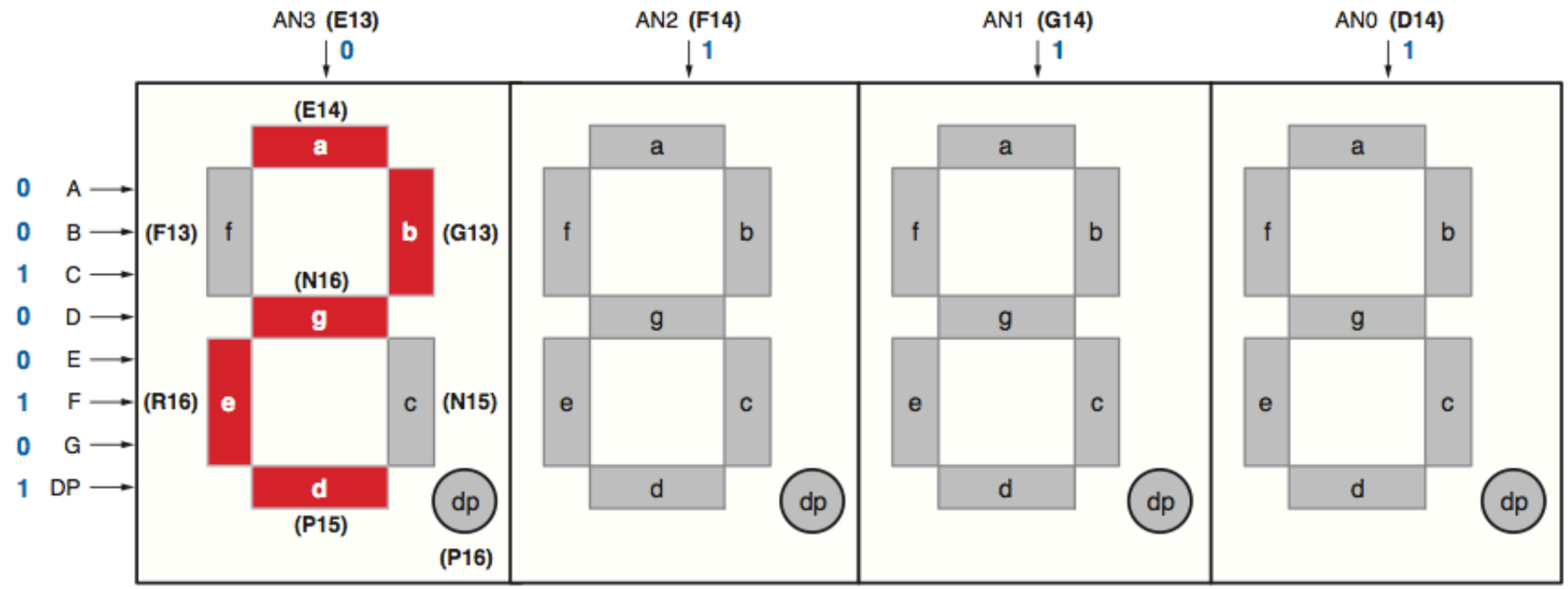
Using SSD

If I want to write 2322 to all four SSD elements I must.

a) Set the AN(3 downto 0) to 0111, which will select the first element

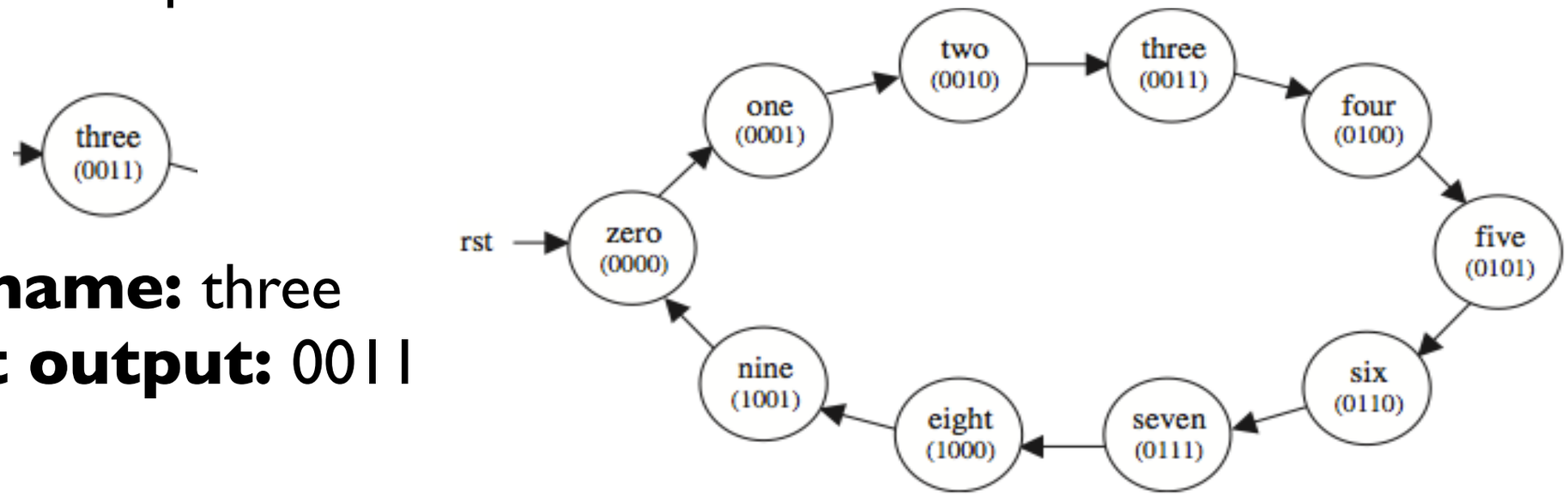
b) Write to the SSD the following bits LED(7 downto 0) 00100101

c) Wait a little bit, and execute steps a) & b) with different values, in order to address different elements and write different numbers!



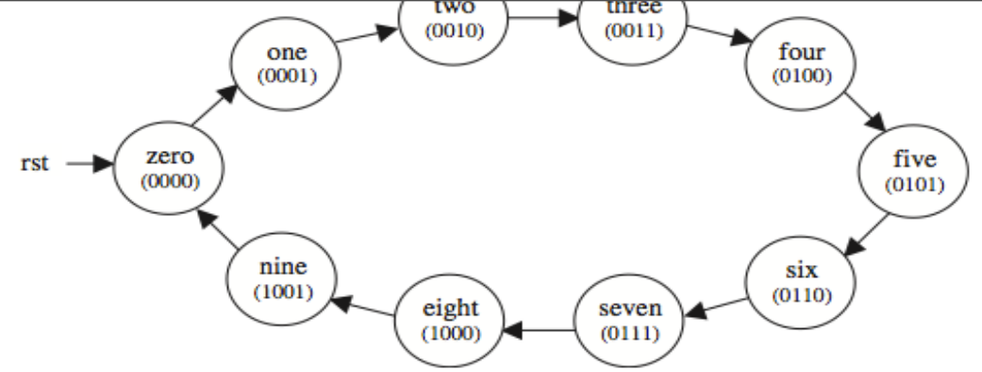
Using the SSD with a FSM

- Approach is very similar to our BCD counter example we discussed on previous class.



State name: three
Circuit output: 0011

- At every rising-edge clock cycle it will change its state.
- When each state changes, the circuit will return a new output value.
- When **rst** is pressed, the present state will now be state zero.




```
1  ------------------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ------------------------------------------------
5  ENTITY counter IS
6     PORT ( clk, rst: IN STD_LOGIC;
7            count: OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
8  END counter;
9  ------------------------------------------------
10 ARCHITECTURE state_machine OF counter IS
11    TYPE state IS (zero, one, two, three, four,
12       five, six, seven, eight, nine);
13    SIGNAL pr_state, nx_state: state;
14 BEGIN
15    ------------- Lower section: -----------------
16    PROCESS (rst, clk)
17    BEGIN
18       IF (rst='1') THEN
19          pr_state <= zero;
20       ELSIF (clk'EVENT AND clk='1') THEN
21          pr_state <= nx_state;
22       END IF;
23    END PROCESS;
24    ------------- Upper section: -----------------
25    PROCESS (pr_state)
26    BEGIN
27       CASE pr_state IS
28          WHEN zero =>
29             count <= "0000";
30             nx_state <= one;
31          WHEN one =>
32             count <= "0001";
33             nx_state <= two;
34          WHEN two =>
35             count <= "0010";
36             nx_state <= three;
37          WHEN three =>
38             count <= "0011";
39             nx_state <= four;
40          WHEN four =>
41             count <= "0100";
42             nx_state <= five;
43          WHEN five =>
44             count <= "0101";
45             nx_state <= six;
46          WHEN six =>
47             count <= "0110";
48             nx_state <= seven;
49          WHEN seven =>
50             count <= "0111";
51             nx_state <= eight;
52          WHEN eight =>
53             count <= "1000";
54             nx_state <= nine;
55          WHEN nine =>
56             count <= "1001";
57             nx_state <= zero;
58       END CASE;
59    END PROCESS;
60 END state_machine;
61 ------------------------------------------------
```

SSD with FSM

State name, which correspond to a particular SSD

...of course you choose the LED values you want to display

1st SSD

An[3:0]=0111
LED[7:0]=10101010

2nd SSD

An[3:0]=1011
LED[7:0]=10101010

3rd SSD

An[3:0]=1101
LED[7:0]=10101010

4th SSD

An[3:0]=1110
LED[7:0]=10101010

Every time a slower clock (much less than 50MHz) goes high, change state.

The LED circuit outputs should be defined by external signals.

Here are some counter modifications you must to in order to get your SSD FSM to work.

```
1  ----------------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ----------------------------------------------
5  ENTITY counter IS
6     PORT ( clk, rst: IN STD_LOGIC;
7            count: OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
8  END counter;
9  ----------------------------------------------
10 ARCHITECTURE state_machine OF counter IS
11    TYPE state IS (zero, one, two, three, four,
12       five, six, seven, eight, nine);
13    SIGNAL pr_state, nx_state: state;
14 BEGIN
15    ------------ Lower section: ----------------
16    PROCESS (rst, clk)
17    BEGIN
18       IF (rst='1') THEN
19          pr_state <= zero;
20       ELSIF (clk'EVENT AND clk='1') THEN
21          pr_state <= nx_state;
22       END IF;
23    END PROCESS;
24    ------------ Upper section: ----------------
25    PROCESS (pr_state)
26    BEGIN
27       CASE pr_state IS
28          WHEN zero =>
29             count <= "0000";
30             nx_state <= one;
31          WHEN one =>
32             count <= "0001";
33             nx_state <= two;
34          WHEN two =>
35             count <= "0010";
36             nx_state <= three;
37          WHEN three =>
38             count <= "0011";
39             nx_state <= four;
40          WHEN four =>
41             count <= "0100";
42             nx_state <= five;
43          WHEN five =>
44             count <= "0101";
45             nx_state <= six;
46          WHEN six =>
47             count <= "0110";
48             nx_state <= seven;
49          WHEN seven =>
50             count <= "0111";
51             nx_state <= eight;
52          WHEN eight =>
53             count <= "1000";
54             nx_state <= nine;
55          WHEN nine =>
56             count <= "1001";
57             nx_state <= zero;
58       END CASE;
59    END PROCESS;
60 END state_machine;
61 ----------------------------------------------
```

Use your own clock, which is defined elsewhere. **You must find a good SSD clock frequency.**

Instead of just count, you need two outputs LED[7:0] and AN[3:0].

Make sure transition is correct (zero and not four).

You do not need all these states... only four!

My suggestion

1) Have two different slower clocks: One that counts the time, and another that cycles between each each of the four SSD.

2) Simulate this behavior in activeHDL with a clock of 2Hz (similar to what we've done on the exam) and test the circuit behavior.

3) Once you are confident your FSM works go into hardware and change the appropriate timer settings.

4) Before you try the timer exercise in hardware, you may want to display a simple counter using the SSD.