Computer Aided Design

Instructor: Dr.Beitollahi

Solutions: Homework 4

Topic: Advanced VHDL & FPGA

Lectures: 7-8-9-10-11

Erfan Hemati
Helia Shamszadeh
Elham Gholami

Theory:

Q4) Design a Finite State Machine (FSM) for a smart home lighting system that can turn the lights on/off, dim, and brighten. (10 points)

برای این سیستم با توجه به خواسته سوال ۵ حالت در نظر می‌گیریم:

خاموش، روشن، کم‌نور، پرنور و خطا (حالت خطا اجباری نیست)

رابطه حالت‌ها به صورت زیر است:

- **خاموش: چراغ‌ها خاموش هستند و منتظر فرمان روشن اند و بعد از روشن شدن می‌توانند کم‌نور یا پرنور شوند.**

  خروجی: (۰) ۰۰ ۰

- **روشن: چراغ‌ها روشن اند، می‌توانند کم‌نور، پرنور یا خاموش شوند.**

  خروجی: (۱)۰۱ ۰

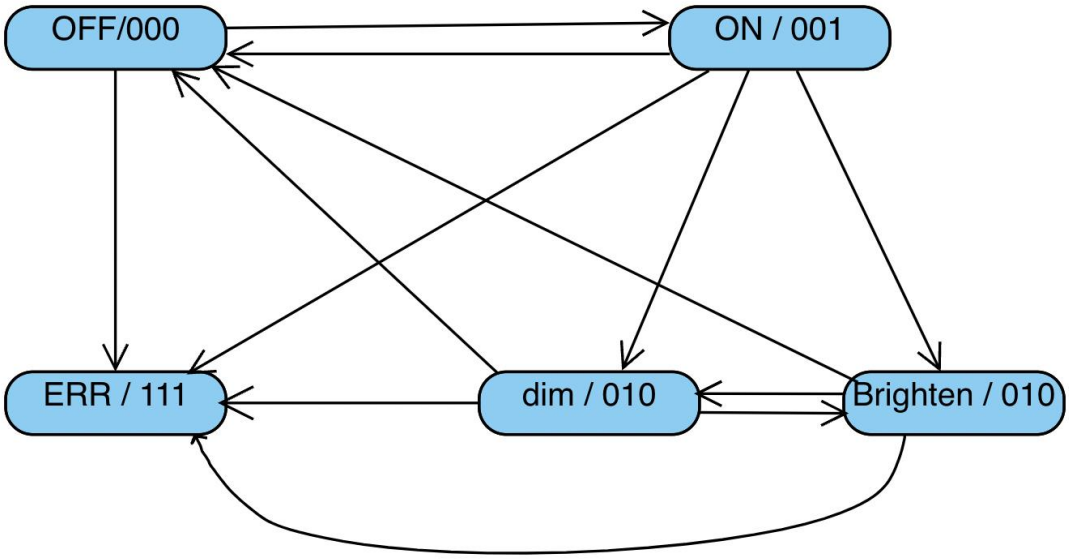
- **کم نور: چراغ‌ها کماکان روشن اما کم‌نور اند که می‌توانند خاموش یا پرنور شوند.**

  خروجی: (۲) ۱۰ ۰

- **پرنور: چراغ‌ها کماکان روشن اما پرنور اند که می‌توانند خاموش یا کم‌نور شوند.**

  خروجی: (۳)۱۱ ۰

- **خطا: هرگونه مشکل در سیستم ما را به این وضعیت می‌برد تا در نهایت یک پیغام خطا نشان داده شود. (۱-) ۱۱۱**



نکته: خروجی این سیستم درواقع تعیین کننده ولتاژ ورودی به لامپ‌ها است، اما برای ساده‌سازی می‌توان خروجی‌ها را ۵ عدد در نظر گرفت. در مرحله بعد سیستمی که بر اساس این FSM کار می‌کند می‌داند هر عدد به چه ولتاژی وصل می‌شود.

نکته: حالت دیگر رسم این استیت‌ها این است که روشن بودن یک استیت برای یک روشنایی متوسط در نظر گرفته شود، یعنی سیستم هنگام خاموشی بتواند به وضعیت کم‌نور، پرنور یا نور متوسط برود. در صورتی که منطق رسم شما این باشد، درست است. اما خواسته اصلی سوال در نظر گرفتن وضعیت روشن، به عنوان وضعیتی است که پس از آن روشنایی سیستم تنظیم شود.

Q5) Implement these two functions on a single PAL with minimum number of resources. (10 points)
(*hint:* You can use S_0 to construct S_1)

$$S_0 = \overline{A_1}\,\overline{A_0}\,B_0\,B_1 + \overline{A_1}\,\overline{A_0}\,B_0\,\overline{B_1} + \overline{A_1}\,A_0\,\overline{B_0}B_1 + \overline{A_1}\,A_0\,\overline{B_0}\,\overline{B_1} + A_1\,\overline{A_0}\,B_0\,B_1 + A_1\,\overline{A_0}\,B_0\,\overline{B_1} + A_1\,A_0\,\overline{B_0}\,B_1 + A_1\,A_0\,\overline{B_0}\,\overline{B_1}$$

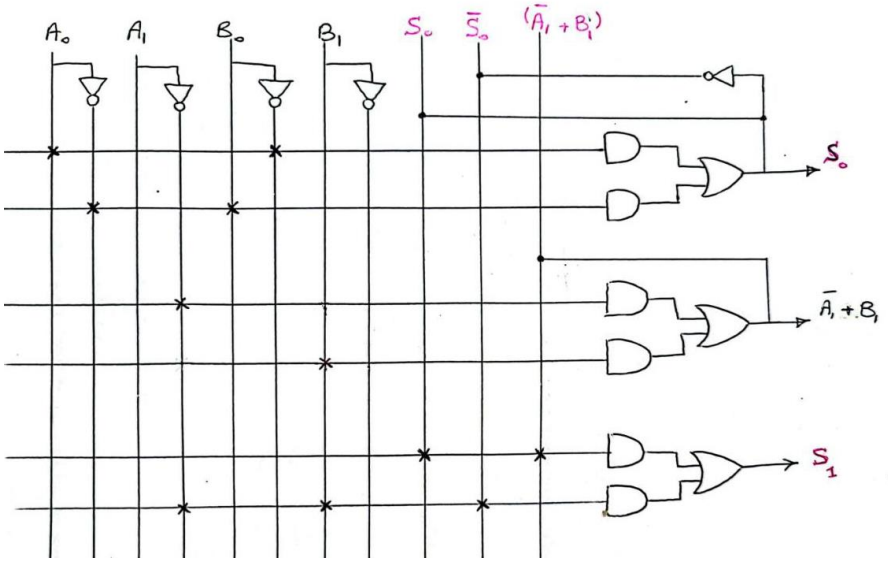
$$= \overline{A_1 A_0} B_0 + \overline{A_1} A_0 \overline{B_0} + A_1 \overline{A_0} B_0 + A_1 A_0 \overline{B_0}$$

$$= \overline{A_0} B_0 + A_0 \overline{B_0}$$

$$S_1 = A_0\,\overline{B_0}\,(A_1 + \overline{B_1}) + \overline{A_0}\,B_0\,(\overline{B_1} + A_1) + \overline{A_1}\,B_1\,(\overline{A_0}\,\overline{B_0} + A_0\,B_0)$$

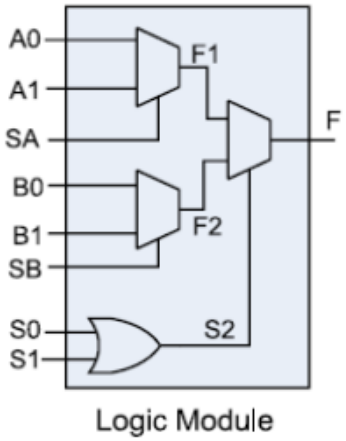
$$= (A_1 + \overline{B_1})(\overline{A_0} B_0 + A_0 \overline{B_0}) + (\overline{A_1} B_1)(\overline{A_0 B_0} + A_0 B_0)$$

$$= (A_1 + \overline{B_1})\, S_0 + (\overline{A_1} B_1)\, \overline{S_0}$$



Q6) Implement the following function on the given logic module. (15 points)

$$F = (A.B) + (B'.C) + D$$



- Expand *F* based on *B* (using Shannon's expansion theorem):
  - $F = B \cdot (A + D) + B' \cdot (C + D) = B \cdot F2 + B' \cdot F1$
  - $F2 = A + D = A + (A'.\ D) = (A \cdot 1) + (A' \cdot D)$
  - $F1 = C + D = C + (C'.D) = (C \cdot 1) + (C' \cdot D)$

- Connect *A*, *B*, *C* to the select lines:
  - $S2 = B$
  - $SA = C$
  - $SB = A$

- '1' and *D* are the inputs of the MUXes:
  - $S0 =$ '0', $S1 = B$
  - $A0 = D,\ A1 =$ '1'
  - $B0 = D,\ B1 =$ '1'

Q7) Using the programmable logic device shown below, implement a circuit in which input is a 3-bit number *x* and its output is *x* + 3. (15 points)
Put a cross in the circles or fill up the circles where the connection should be made.

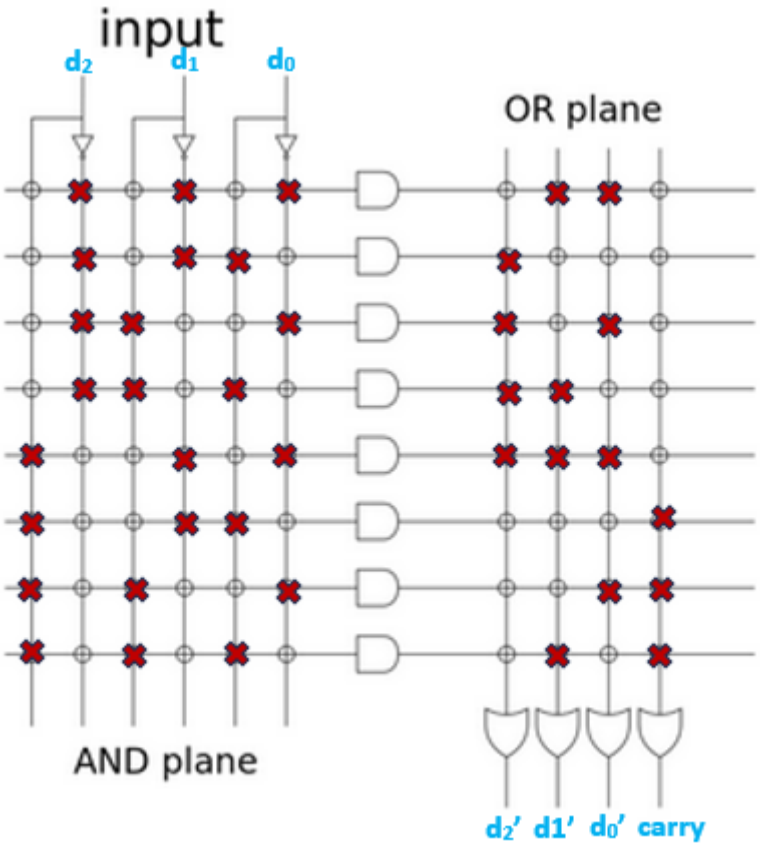
d_2	d_1	d_0	d_2'	d_1'	d_0'	carry
0	0	0	0	1	1	0
0	0	1	1	0	0	0
0	1	0	1	0	1	0
0	1	1	1	1	0	0
1	0	0	1	1	1	0
1	0	1	0	0	0	1
1	1	0	0	0	1	1
1	1	1	0	1	0	1

$\rightarrow d_2' = \bar{d_2}\,\bar{d_1}\,d_0 + \bar{d_2}\,d_1\,\bar{d_0} + \bar{d_2}\,d_1\,d_0 + d_2\,\bar{d_1}\,\bar{d_0} = m(1, 2, 3, 4)$

$\rightarrow d_1' = \bar{d_2}\,\bar{d_1}\,\bar{d_0} + \bar{d_2}\,d_1\,d_0 + d_2\,\bar{d_1}\,\bar{d_0} + d_2\,d_1\,d_0 = m(0, 3, 4, 7)$

$\rightarrow d_0' = \bar{d_2}\,\bar{d_1}\,\bar{d_0} + \bar{d_2}\,d_1\,\bar{d_0} + d_2\,\bar{d_1}\,\bar{d_0} + d_2\,d_1\,\bar{d_0} = m(0, 2, 4, 6)$

$\rightarrow carry = d_2\,\bar{d_1}\,d_0 + d_2\,d_1\,\bar{d_0} + d_2\,d_1\,d_0 = m(5, 6, 7)$



Implementation:

Q1) Consider the *Carry Ripple Adder* given below. Design a FAU (Full Adder Unit), to be used as a COMPONENT. Write a code for the complete *Carry Ripple Adder* containing instantiations of FAU. Compile and simulate the synthesized circuit. (15 points)

- Sample VHDL code is provided in the Github repository.

Q2) Write a function capable of converting an INTEGER to a STD_LOGIC_VECTOR value. Call it *conv_std_logic*( ). Then write an application example, containing a call to your function, in order to test it. Construct a solution containing the function in the main code itself. (15 points)

- Sample VHDL code is provided in the Github repository.

Q3) Design a password system such that its inputs are *0*, *1* and *Reset*. The system should recognize the input pattern *0110* as the correct password and, in this case, change the output state of the lock to *1* (*Unlocked*). If the input pattern is incorrect or the Reset button is pressed, the system should return to the initial state and change the output to 0 (*Locked*). The states and transitions between them should be clearly defined in the code. (20 points)

Pseudocode:

```
Initialize state to RESET
Initialize input buffer to empty

Function check_logic(buffer):
   If buffer equals "0110":
      Return UNLOCK
   Else:
      Return RESET

Main loop:
   While True:
      If reset_button_pressed:
         state = RESET
         output = LOCKED
         input_buffer = empty
      Else:
         If new_digit_received:
```

Append new digit to input_buffer

If input_buffer contains 4 digits:
state = check_password(input_buffer)
If state is UNLOCK:
output = UNLOCKED
Else:
output = LOCKED
input_buffer = empty

Wait for next clock cycle

End loop

- Inputs : 0, 1, reset(-1)
- output: lock(0) unlock(1)
- By any wrong digit you go to mistake states(s0x, s01x, s011x)