Description: technion-logoטכניון – מכון טכנולוגי לישראל

**תכן לוגי  
תרגיל רטוב 1#**

מגישים:

****

**אור פרפרה 205758345**

****

**עידו גליל 204408272**

**חלק יבש**

**שאלה 1**

הסבירו את ההבדל בין X ו Z

x זה don’t care – מצב בו ערך ההכנסה לא משנה עבור פעולת המערכת ולכן ניתן להכניס אליו כל ערך לוגי והמערכת עדיין תפעל כהלכה.

z זה מצב לא תקין במערכת הנוצר מתכנון לקוי של המערכת ועשוי להתפרש כ-0 או 1, אך עשוי לגרום למערכת לא לפעול כמצופה.

מלאו את טבלאות האמת הבאות

שער and

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| z | x | 1 | 0 |  |
| 0 | 0 | 0 | 0 | 0 |
| x | x | 1 | 0 | 1 |
| x | x | x | 0 | x |
| x | x | x | 0 | z |

שער or

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| z | x | 1 | 0 |  |
| x | x | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |
| x | x | 1 | x | x |
| x | x | 1 | x | z |

**שאלה 2**

נניח כי a = reg[3:0], אילו ערכי a יחשבו כערך אמת (if (a)) ?

כל ערך a שהוא לא וקטור אפסים (כל וקטור מלבד (0,0,0,0) )

הסבירו את ההבדל בין האופרטור == לבין האופרטור ===

האופרטור '==' הוא אופרטור דו מקומי שמשווה רק בין ערכי 0 ו-1, אך אם אחד מהאופרנדים הוא z או x, ערך ההחזרה יהיה תמיד 0. לעומתו, האופרטור '===' יודע להשוות בין כל 4 "המצבים הלוגיים" (0,1,x,z), כך שבמידה ושני האופרנדים הם מאותו מצב לוגי ערך ההחזרה יהיה '1', ו-'0' אחרת. לא ניתן באמת לסנתז את האופרטור השני לרכיב אמיתי, והוא נועד רק עבור בדיקות.

הסבירו את ההבדל בין האופרטור reduction OR לבין האופרטור bit-wise OR

האופרטור reduction-OR הוא אונרי, ומבצע על כל הביטים של האופרנד שלו את פעולת OR, ומחזיר את התוצאה כביט בודד. האופרטור bitwise-OR הוא בינארי, ומבצע פעולת OR בין שני האופרנדים ביט מול ביט בהתאמה (במיקום), ובמידה ויש אי התאמה בין אורכי האופרנדים, מתבצעת השלמה של ערכי '0' משמאלו של האופרנד הקצר יותר. ערך ההחזרה הוא וקטור באורך האופרנד הארוך עם תוצאות ה-OR שהתקבלו.

הסבירו את ההבדל בין האופרטור הבינארי & לבין האופרטור הבינארי &&

האופרטור הבינארי & הוא bitwise-AND, ומבצע פעולת AND בין שני האופרנדים ביט מול ביט בהתאמה במיקום, ומחזיר וקטור עם תוצאות ה-AND שהתקבלו.

האופרטור && מבצע פעולת AND בין שני ערכים בוליאנים (כאשר כל אופרנד שהוא וקטור '0' באורך כלשהו מתפרש כערך שקר ('0'), וכל השאר כערכי אמת ('1')) ומחזיר ערך בוליאני '0' או '1' כתוצאה.

**שאלה 3**

בכל אחת מהשורות הבאות מלאו בהתאמה reg / wire / both

|  |  |
| --- | --- |
| **wire** | left-hand side of an assign statement |
| **both** | right-hand side of an assign statement |
| **reg** | left-hand side of an = or <= sign in an always@ block |
| **both** | right-hand side of an = or <= sign in an always@ block |
| **both** | can be connected to the input port of a module instantiation |
| **wire** | can be connected to the output port of a module instantiation |
| **both** | can be used as outputs within an actual module declaration |
| **wire** | can be used as inputs within an actual module declaration |

**שאלה 4**

הסבירו את ההבדל ביןBlocking לבין Nonblocking assignmentותנו דוגמה בה שימוש בכל אחת יביא לתוצאה שונה

Blocking assignment –מבצעת פעולת השמה לתוך האופרטור ברגע שהפעולה הקודמת הסתיימה. לדוגמא: במידה וישנם כמה פקודות ברצף של השמה, קודם הפקודה הראשונה מתבצעת ואז השניה וכו'...

always @(posedge clk) begin

Test\_1 =1'b1

Test\_2=Test\_1

end

בדוגמא זו, לtest\_2 יכנס הערך 1 אחרי מחזור שעון 1.

Nonblocking assignment –מבצעת השמה לתוך האופרטור באותו רגע ולא מחכה לסיום שאר הפעולות.

בהמשך לדוגמא הקודמת –

always @(posedge clk) begin

Test\_1<=1'b1

Test\_2<=Test\_1

end

כאן יכנס לtest\_1 הערך 1 ול-test\_2 הערך הישן של test\_1 במחזור הראשון, ובמחזור השני יכנס הערך '1' ל-test\_2.

**חלק רטוב**

**בתרגיל זה נממש את גרסת מור של מכונת המצבים מתרגול #1 (שקפים 28-32)**

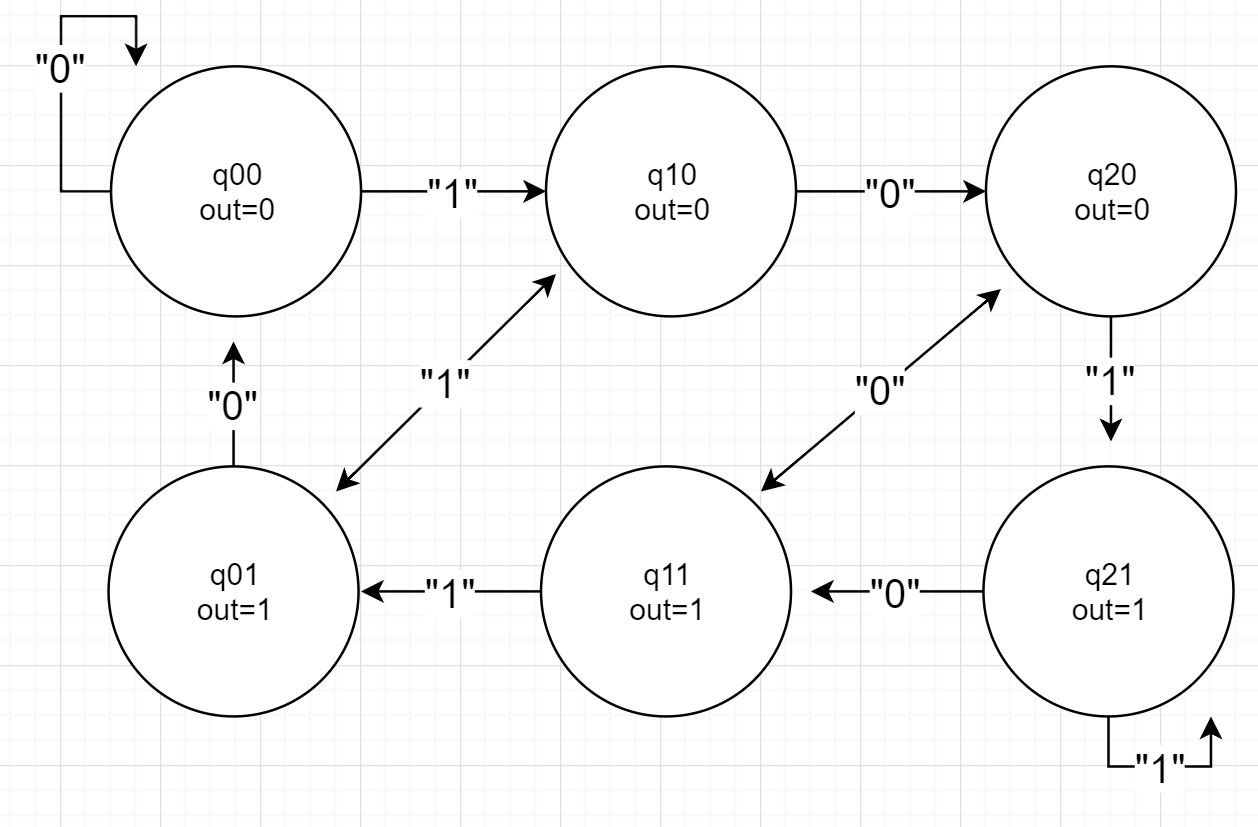
* המערכת מקבלת מספר טבעי X ומשדרת מספר טבעי Y כך ש  
* הקלט והפלט בינאריים, בכל מחזור שעון נקלט ונפלט ביט אחד, החל מהMSB.
* הLSB של Y משודר מחזור שעון אחד לאחר המחזור בו נקלט הLSB של X (מכונת מור).
* למערכת ישנה כניסת reset, כאשר reset==1 מכונת המצבים צריכה לחזור למצבה ההתחלתי.

**הוראות נוספות:**

* אין לשנות את הכניסות או היציאות למודול.
* אין לשנות את שם המודול או שם הקובץ.
* בהגשה האלקטרונית יש להגיש רק את הקובץ my\_fsm.
* **יש לצרף לחלק היבש ציור של מכונת המצבים וכן את הקוד שכתבתם מודפס.**
* מדריך להורדתModelSim נמצא באתר הקורס של סמסטר אביב 17.
* עבור העבודה עם ModelSim מומלץ לראות את הוידאו בקישור   
  <https://www.youtube.com/watch?v=Z8whdGa7RtY>
* שימו לב שמספיק להוריד רק את ModelSim, לא צריך את Quartus או device support.
* בכתיבת הקוד ניתן ומומלץ להיעזר בtemplate הבא :  
  <https://www.altera.com/support/support-resources/design-examples/design-software/verilog/ver_statem.html>
* לשם בדיקת המודול שכתבתם קיבלתם את הקובץ test\_fsm.v עם טסט בסיסי בלבד, מומלץ לכתוב טסטים מלאים שיוודאו נכונות עבור כל קלט.

**חלק רטוב:**

**דיאגרמת מצבים:**



**קוד מודפס:**

module my\_fsm (

clock ,

reset ,

in ,

out

);

input clock, reset, in;

output out;

wire clock, reset, in;

reg out;

reg [2:0] state = 0;

always @(state)

begin

case (state)

0: out=0;

1: out=1;

2: out=0;

3: out=1;

4: out=1;

5: out=0;

default: out=0;

endcase

end

always@(posedge clock or posedge reset) begin

if (reset == 1) begin

out=0;

state=0;

end

else begin

case(state)

0:

if(in==0)

state=0;

else

state=2;

1:

if(in==0)

state=0;

else

state=2;

2:

if(in==0)

state=5;

else

state=1;

3:

if(in==0)

state=5;

else

state=1;

4:

if(in==0)

state=3;

else

state=4;

5:

if(in==0)

state=3;

else

state=4;

endcase

end

end

endmodule