**הטכניון - מכון טכנולוגי לישראל**

**הפקולטה להנדסת חשמל**



מעבדה בהנדסת חשמל

1א' 044157

ניסויSV2

שאלות ודוח הכנה

גרסה 1.5

קיץ 2020

על פי חוברות של עמוס זסלבסקי, 2009

עדכן אלכס קרינשפון + קובי דקל

|  |  |
| --- | --- |
| תאריך כתיבת הדו"ח | 16/8/20 |
| שם המדריך | אלון |

|  |  |  |
| --- | --- | --- |
| סטודנט | שם פרטי | שם משפחה |
| 1 | נועם | אילתה |
| 2 | ליאור | דביר |

**תוכן עניינים**

[1 פתיחת ארכיב 3](#_Toc48227162)

[2 מונה דצימלי יורד 2 ספרות 3](#_Toc48227163)

[2.1 הגדרות – מונה דצימלי יורד 3](#_Toc48227164)

[2.1.1 Module interface 3](#_Toc48227165)

[2.1.2 Truth table 4](#_Toc48227166)

[2.2 מונה decimal\_2\_digits\_counter יורד לשתי ספרות 5](#_Toc48227167)

[2.2.1 Module interface 6](#_Toc48227168)

[2.2.2 Truth table decimal\_2\_digits\_counter 6](#_Toc48227169)

[2.2.3 מימוש המונה לשתי ספרות 6](#_Toc48227170)

[2.3 סימולציה 8](#_Toc48227171)

[3 רמזור מבוקר 10](#_Toc48227172)

[3.1 הצגת הדרישות 10](#_Toc48227173)

[3.1.1 Ramzor Module interface 10](#_Toc48227174)

[3.1.2 Ramzor Module Block diagram 10](#_Toc48227175)

[3.2 RAMZOR\_FSM 11](#_Toc48227176)

[3.2.1 RAMZOR\_FSM Module interface 11](#_Toc48227177)

[3.2.2 RAMSOR\_FSM bubble diagram 11](#_Toc48227178)

[3.3 Aux\_timer 12](#_Toc48227179)

[3.3.1 Aux\_Timer Module interface 12](#_Toc48227180)

[3.4 onetens\_sec\_counter 13](#_Toc48227181)

[3.4.1 onetens\_sec\_counter Module interface 13](#_Toc48227182)

[3.5 קוד SV 13](#_Toc48227183)

[3.6 סימולציה 15](#_Toc48227184)

[4 פצצה – פרוייקטון 17](#_Toc48227185)

[4.1 הגדרות הפצצה 17](#_Toc48227186)

[4.2 תכנון הפרויקטון – הפצצה 18](#_Toc48227187)

[4.3 תכנון מכונת המצבים – הפצצה 19](#_Toc48227188)

[4.4 מימוש מכונת המצבים 20](#_Toc48227189)

[4.5 סימולציה של מכונת המצבים 22](#_Toc48227190)

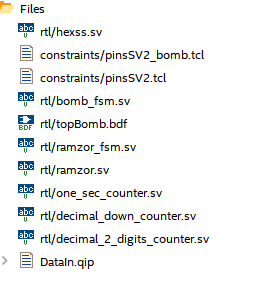
[4.6 השלמה של דיאגרמת בלוקים מלאה של הפצצה 23](#_Toc48227191)

[5 גיבוי העבודה 24](#_Toc48227192)

# פתיחת ארכיב

צור תיקיה למעבדה זו. הורד מהמודל קובץ ארכיב של המעבדה ופתח אותו לפרויקט בתיקייה שיצרת.

ודא תכולת קבצים כזו:



# מונה דצימלי יורד 2 ספרות

## הגדרות – מונה דצימלי יורד

מונה decimal\_down\_counter.sv הינו מונה הסופר מטה מ-9 עד 0

כאשר מגיע ל- 0 הוא מעלה את סיגנל tc למשך מחזור שעון אחד. הדבר מאפשר לשרשר כמה מונים ולבנות מונה בעל כמה ספרות.

### Module interface

תכננו מונה **דצימלי** יורד כמתואר בהמשך והוסיפו את הקוד שלכם בקובץ - decimal\_down\_counter.sv

להלן הכניסות והיציאות של יחידת המונה הדצימלי היורד :

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **width** | **Direction** | |
| **clk** | **1** | **input** |
| **resetN** | **1** | **input** |
| **ena** | **1** | **Input** |
| **ena\_cnt** | **1** | **Input** |
| **loadN** | **1** | **Input** |
| **datain** | **[3:0]** | **input** |
| **count** | **[3:0]** | **output** |
| **tc** | **1** | **output** |

### Truth table

יש לתכנן את קוד המונה לפי טבלת האמת הבאה:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Count[3..0]** | **datain[3:0]** | **loadN** | **ena\_cnt** | **ena** | **resetN**  **(Async)** | | **CLK** |
| Reset | 4'b0000 | x | x | x | x | 0 | **x** | |
| Load | datain[3:0] | datain[3:0] | 0 | x | x | 1 | **↑** | |
|  | previous count | x | 1 | x | 0 | 1 | **↑** | |
|  | previous count | x | 1 | 0 | x | 1 | **↑** | |
| decrement | if (count == 0)  count <= 4'h9  else  count <= count-1; | x | 1 | 1 | 1 | 1 | **↑** | |

היציאה האסינכרונית **tc** מתוארת להלן:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **tc**  **Async** | **count[3:0]** | **loadN** | **ena\_cnt** | **ena** | **resetN** | | **CLK** |
| 1 | 4'b0000 | x | x | x | x | **x** | |

**הערה חשובה!!! בכמה מקבצי השלד הנתונים היה צורך לסגור חלק של הקוד כדי שהקוד יעבור קומפילציה. לכן לפני שמתחילים לכתוב קוד יש להסיר את ההערות המסומנות ב-**

**/\* $$$$$$ remove to fill**

**ולהשלים את הקוד שלכם במקומות המסומנים ב-**

**//fill your code here**

הוסיפו את קוד ה-SVv שלכם לדו"ח:

**module** decimal\_down\_counter

**(**

**input** **logic** clk**,**

**input** **logic** resetN**,**

**input** **logic** ena**,**

**input** **logic** ena\_cnt**,**

**input** **logic** loadN**,**

**input** **logic** **[**3**:**0**]** datain**,**

**output** **logic** **[**3**:**0**]** count**,**

**output** **logic** tc

**);**

// Down counter

**always\_ff** **@(posedge** clk **or** **negedge** resetN**)**

**begin**

**if** **(** **!**resetN **)** **begin** // Asynchronic reset

count **<=** 4'b0**;**

**end**

**else** **begin** // Synchronic logic

**if** **(!**loadN**)** **begin**

count **<=** datain**;**

**end** **else** **if** **(!**ena**)** **begin**

count **<=** count**;**

**end** **else** **if** **(!**ena\_cnt**)** **begin**

count **<=** count**;**

**end** **else** **if** **(**count **==** 4'b0**)** **begin**

count **<=** 4'h9**;**

**end** **else** **begin**

count **<=** count **-** 1'b1**;**

**end**

**end** //Synch

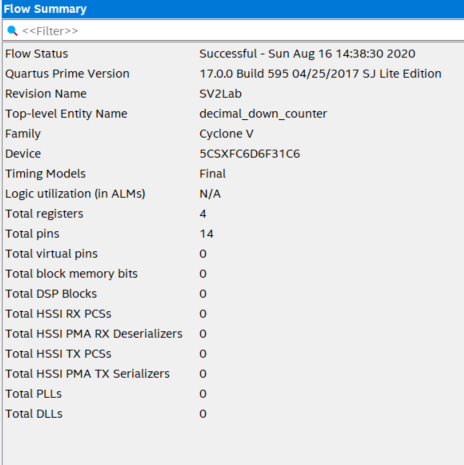
**end** //always

// Asynchronic tc

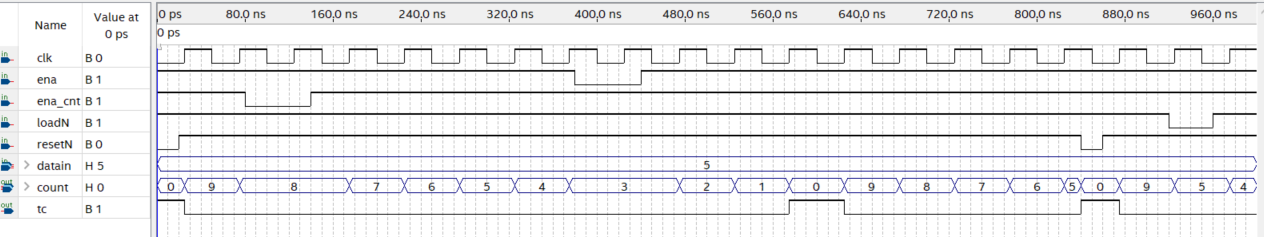
**assign** tc **=** **(**count **==** 4'b0 **)** **?** 1'b1 **:** 1'b0**;**

**endmodule**

בצעו סינתזה מוצלחת למונה ספרה אחת וצרפו את הסיכום לדו"ח:



בצעו סימולציה למונה ספרה אחת וצרפו את התוצאות לדו"ח (כללו את כל המקרים הרלוונטיים):



## מונה decimal\_2\_digits\_counter יורד לשתי ספרות

מונה decimal\_2\_digits\_counter.sv הינו מונה הסופר מ-99 עד 0.

כאשר המונה מגיע ל- 0 מעלה את סיגנל ה-tc ל- 1 למשך מחזור שעון אחד.

מונה זה נבנה על ידי שימוש ב- 2 מונים מסוג: decimal\_down\_counter.sv .

### Module interface

להלן הכניסות והיציאות של יחידת המונה הדצימלי מטה של 2 ספרות:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **width** | **Direction** | |
| **clk** | **1** | **input** |
| **resetN** | **1** | **input** |
| **ena** | **1** | **Input** |
| **ena\_cnt** | **1** | **Input** |
| **loadN** | **1** | **Input** |
| **Data\_init** | **[7:0]** | **input** |
| **Count\_out** | **[7:0]** | **Output** |
| **tc** | **1** | **Output** |

### Truth table decimal\_2\_digits\_counter

יש לתכנן את מונה ה-2 ספרות לפי טבלת האמת הבאה:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **tc Async** | **Count\_out[7:0]** | **Data \_init7:0]** | **loadN** | **ena\_cnt** | **ena** | **resetN** | **CLK** |
| 1’b0 | 8'd0 | x | x | x | x | 0 | **x** |
| 1’b0 | Data\_init [7:0] | Data\_init [7:0] | 0 | x | x | 1 | **↑** |
| 1’b0 | x | x | 1 | x | 0 | 1 | **↑** |
| 1’b0 | x | x | 1 | 0 | x | 1 | **↑** |
| 1’b0 | Count\_out[7:0]-1 | x | 1 | 1 | 1 | 1 | **↑** |
| 1’b1 | 8’d0 | x | x | x | x | x | **x** |

### מימוש המונה לשתי ספרות

ממש את מונה 2 הספרות בקובץ decimal\_2digits\_counter.sv . היעזר בשלד הקיים בפרויקט.

יש לממש את הקוד באופן היררכי כך שיהיה מבוסס על המונה של ספרה אחת מהסעיף הקודם. הכניסה והיציאה של המונה הוא מספר בינארי בין 8 סיביות . 4 הסיביות התחתונות מייצגות את סיפרת היחידות ו- 4 הספרות העליונות מייצגות את סיפרת העשרות.

**module** decimal\_2\_digits\_counter

**(**

**input** **logic** clk**,**

**input** **logic** resetN**,**

**input** **logic** ena**,**

**input** **logic** ena\_cnt**,**

**input** **logic** loadN**,**

**input** **logic** **[**7**:**0**]** Data\_init**,**

**module** decimal\_2\_digits\_counter

**(**

**input** **logic** clk**,**

**input** **logic** resetN**,**

**input** **logic** ena**,**

**input** **logic** ena\_cnt**,**

**input** **logic** loadN**,**

**input** **logic** **[**7**:**0**]** Data\_init**,**

**output** **logic** **[**7**:**0**]** Count\_out**,**

**output** **logic** tc

**);**

**logic** tc\_ones **;**

**logic** tc\_tens **;**

// units (Ones)

decimal\_down\_counter ones\_counter**(**

**.**clk**(**clk**),**

**.**resetN**(**resetN**),**

**.**ena**(**ena**),**

**.**ena\_cnt**(**ena\_cnt**)** **,**

**.**loadN**(**loadN**),**

**.**datain**(**Data\_init**[**3**:**0**]),**

**.**count**(**Count\_out**[**3**:**0**]),**

**.**tc**(**tc\_ones**)**

**);**

// Tens

decimal\_down\_counter tens\_counter**(**

**.**clk**(**clk**),**

**.**resetN**(**resetN**),**

**.**ena**(**ena**),**

**.**ena\_cnt**(**tc\_ones**)** **,**

**.**loadN**(**loadN**),**

**.**datain**(**Data\_init**[**7**:**4**]),**

**.**count**(**Count\_out**[**7**:**4**]),**

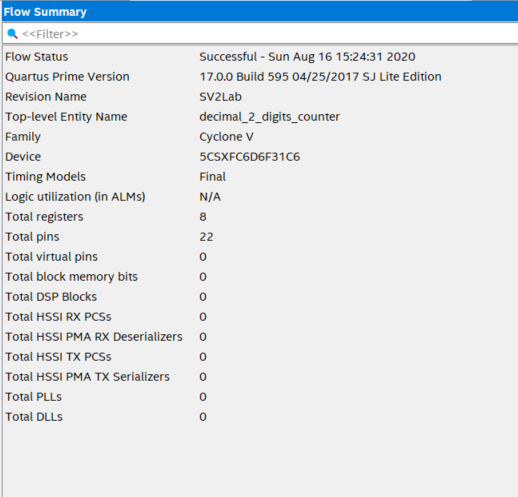
**.**tc**(**tc\_tens**)**

**);**

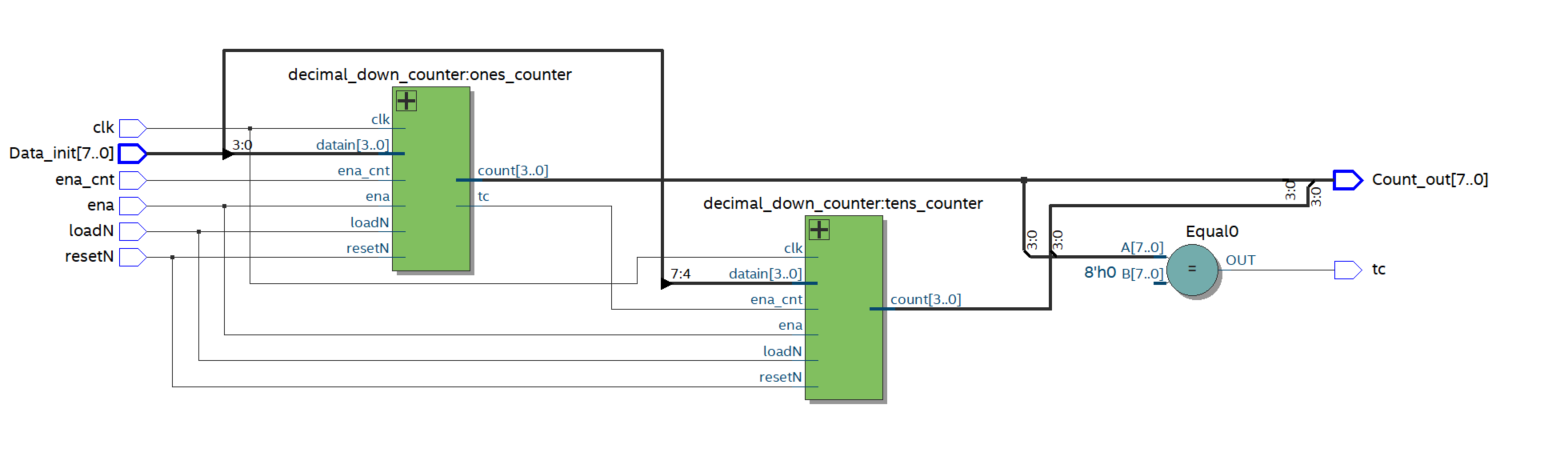
**assign** tc **=** **(**Count\_out **==** 8'b0**)** **?** 1'b1 **:** 1'b0**;**

**endmodule**

הרץ סינתזה מוצלחת לקובץ decimal\_2\_digits\_counter.sv וצרף את הסיכום לדו"ח:



הצג את המימוש של המונה decimal\_2\_digits\_counter.sv כ- RTL VIEW והוסף אותו לדו"ח:



## סימולציה

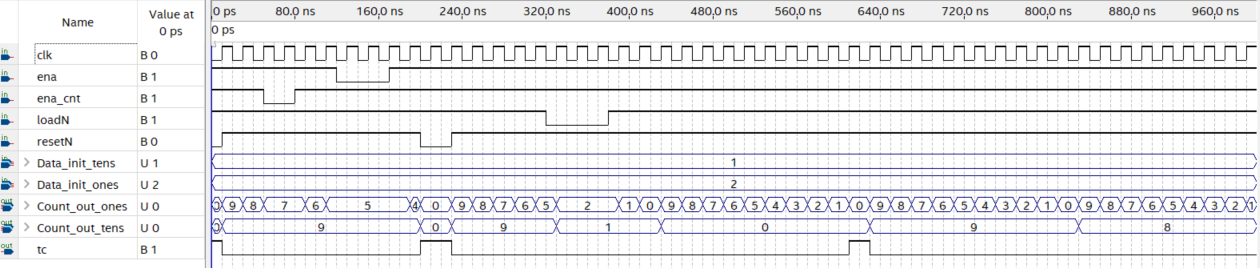
הרץ סימולציה מלאה לתכן **decimal\_2\_digits\_counter.sv** בה תבדוק את כל הכניסות והיציאות **המעניינות** ואת כל מקרי הקצה **המעניינים**.

הגדר מה תרצה לבדוק בסימולציה – איזה מצבים מעניינים (המשך למלא את הטבלה)

|  |  |
| --- | --- |
| **מצב** | **תוצאות צפויות** |
| הפעלת **resetN** | כל היציאות מאותחלות |
| COUNT בלי ENA\_CNT | הספירה נשארת קבועה |
| COUNT בלי ENA | הספירה נשארת קבועה |
| **הפעלת loadN** | הכנסת ערכי Data\_in |
| TC | 1 אם הספירה (Count\_out) הגיעה ל0, אחרת 0 |

הקפד להציג את המשתנים בפורמט הרלוונטי **BIT HEX DECIMAL** .

הוסף את תוצאות הסימולציה לדו"ח.



# רמזור מבוקר

בתרגיל זה עליכם לבנות רמזור באמצעות מכונת מצבים ושימוש במונה Bit 8 שמימשתם בסעיף 2 .

**בעבודת הכנה זו** תתכננו את מכונת המצבים, ותשלבו את מכונת המצבים עם המונה שייטען בזמן המתאים על פי דרישת הזמנים של המצב הבא. כתבו את הקוד שלכם בשלד הקוד הנתון והריצו סימולציה על מנת לבדוק שמה שכתבתם עובד נכון. **במעבדה** תבדקו את הרמזור על הכרטיס.

שימו לב שזמני ההפעלה במצבים השונים של פעולת הרמזור נתונים בעשיריות שניה. במעבדה הקודמת נעזרנו במחלק תדר one\_sec\_counter.sv המייצר פולס של 1Sec שבו נשתמש גם במעבדה זו. על מנת לייצר מנייה של עשיריות שניה נעזר במונה זה ונעדכן בו את חלוקת התדר במטרה לספור עשיריות שניה. לדוגמה 2.5Sec הם 25Tens of Sec)

לצורך כך קחו את הקובץ הישן, שמרו אותו בשם חדש onetens\_sec\_counter.sv ועשו בו את השינויים הנדרשים כך שיפעל בהתאם.

## הצגת הדרישות

**פעולת הרמזור**:

* **מצבי הרמזור הם אדום, אדום\_צהוב, ירוק, צהוב**.
* זמני הרמזור בכל מצב:
  + 4.8 שניות באדום
  + .63 שניות בירוק
  + 1.8 שניות במצבי המעבר (צהוב, אדום-צהוב)
* לרמזור כניסת **TURBO** באמצעותה אפשר לזרז את פעולת הרמזור פי 16.
* לחיצה על לחצן הכניסה **SwitchN** (הלחצן בלוגיקה שלילית, '0' כשהוא לחוץ):
* אם הרמזור במצב אדום, לחיצה על SwitchN תעביר את הרמזור ישירות (בצורה סינכרונית בשעון המהיר הבא) למצב אדום-צהוב.
* אחרת הלחיצה לא תעשה כלום.

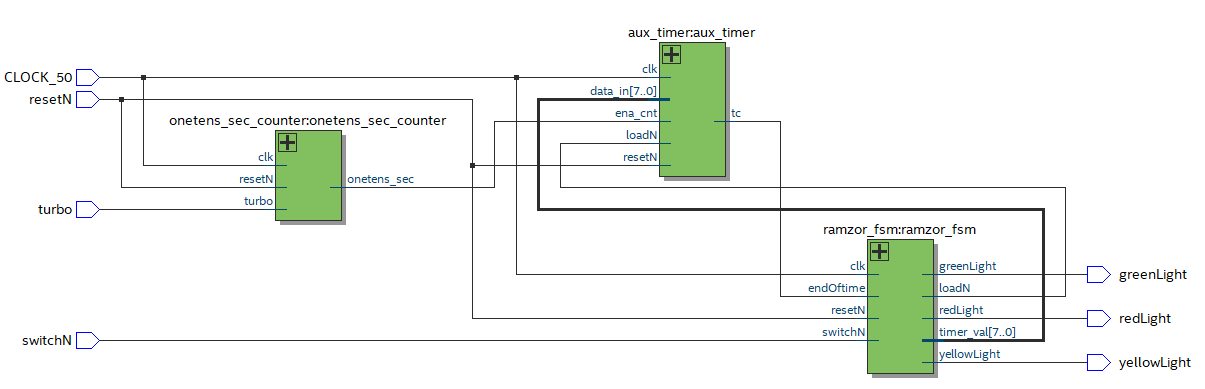
### Ramzor Module interface

להלן הכניסות והיציאות של הרמזור:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **width** | **Direction** | |
| **CLOCK\_50** | **1** | **input** |
| **resetN** | **1** | **input** |
| **turbo** | **1** | **Input** |
| **switchN** | **1** | **Input** |
| **loadN** | **1** | **Input** |
| **greenLight** | **1** | **output** |
| **redLight** | **1** | **output** |
| **yellowLight** | **1** | **output** |

### Ramzor Module Block diagram

להלן מימוש הרמזור על ידי שימוש במכונת המצבים, מונה ומחלק תדר:



## RAMZOR\_FSM

### RAMZOR\_FSM Module interface

להלן הכניסות והיציאות עבור הבלוק של מכונת המצבים:

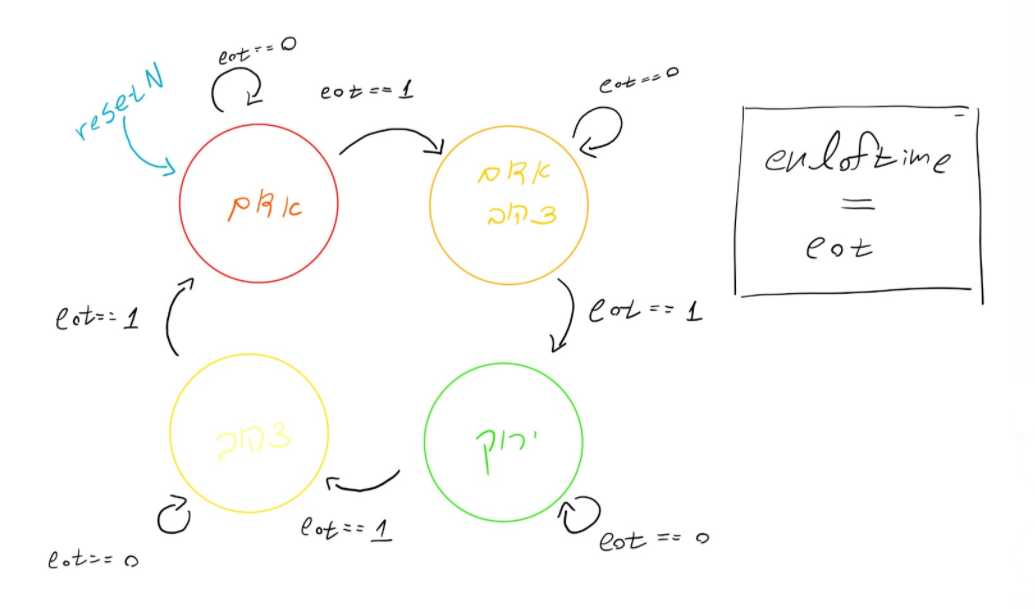
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **width** | **Type** | **Direction** | |
| **clk** | **1** | **logic** | **input** |
| **resetN** | **1** | **logic** | **input** |
| **switchN** | **1** | **logic** | **Input** |
| **cntDn** | **1** | **logic** | **Input** |
| **greenLight** | **1** | **logic** | **output** |
| **redLight** | **1** | **logic** | **output** |
| **yellowLight** | **1** | **logic** | **output** |

הפרמטרים:

|  |  |  |
| --- | --- | --- |
| **Default value** | **Type** | **Parameter** |
| **48** | **int** | **red\_timer** |
| **18** | **int** | **red\_yellow\_timer** |
| **36** | **int** | **green\_timer** |
| **18** | **int** | **yellow\_timer** |

### RAMSOR\_FSM bubble diagram

שרטט דיאגרמה של מכונת המצבים של הרמזור, אפשר בעיפרון ולצלם:



## Aux\_timer

רכיב זה מבוסס על המונה בקובץ **decimal\_down\_counter.sv** עם מעט שינויים. רכיב זה מיצר פולס במשך זמן של שעון אחד (tc) כל פעם שהמונה הפנימי שלו, שהיה טעון לערך התחלתי, ירד לאפס.

נדרש:

1. לשמור את הקובץ הנ"ל בשם aux\_timer,sv בתוך המחיצה rtl\ .
2. לעדכן את המונה הפנימי שלו לפי הטבלה המופיעה בסעיף הבא , וכן את היציאה שלו.

### Aux\_Timer Module interface

להלן הכניסות והיציאות של מונה הזמן של הרמזור. שימו לב שלשעון בניגוד למונה יש יציאה שלbit אח(tc) ללא יציאת ערך המונה :

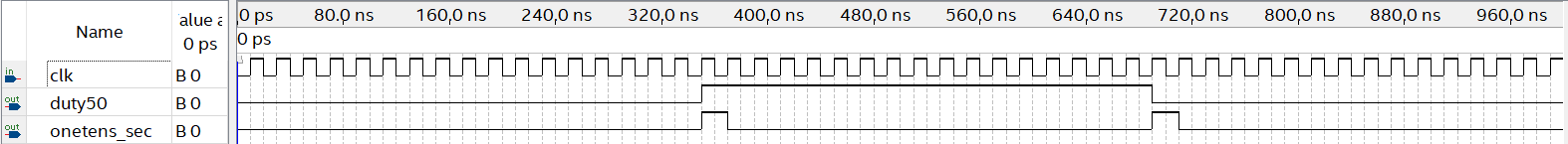
|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **width** | **Direction** | |
| **clk** | **1** | **input** |
| **resetN** | **1** | **input** |
| **ena\_cnt** | **1** | **Input** |
| **loadN** | **1** | **Input** |
| **data\_in** | **[7:0]** | **input** |
| **tc** | **1** | **output** |

## onetens\_sec\_counter

רכיב זה מייצר פולסים בתדר של 1/10 שניה ברוחב פולסי השעון המהיר CLOCK\_50 (או פעם ב16- מחזורי שעון עבור הסימולציות) - וכן פולסים של duty50.

עליכם :

1. לעלות את הקובץ *one\_sec\_counter.sv* נימצא בפרויקט ( אותו הכרנו גם במעבדה SV1 )
2. לשנות את שמו ל *onetens\_sec\_counter.sv* ולשמור אותו במחיצה .\rtl\
3. *לתקנו כך שיתן פולס* פעם בעשירית שניה ( כולל התייחסות לסימולציה והרצה על הכרטיס במעבדה )
4. *להחליף את שמות המשתנים הפנימיים והיציאות מ - oneSec\* ל \*onetensSec* -(חשוב מאוד למען מי שיקרא אותו בעתיד)
5. לצרפו לפרויקט



### onetens\_sec\_counter Module interface

להלן הכניסות והיציאות של המונה

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Name** | **width** | **Direction** | |
|  | **clk** | **1** | **input** |
|  | **resetN** | **1** | **input** |
| **מקצרת מחזור 1/16** | **turbo** | **1** | **Input** |
|  | **onetens\_sec** | **1** | **output** |

## קוד SV

השלם את הקוד שלך בשלד הקוד הנתון של הרמזור (ramzor\_fsm.sv).

בצע סינתזה מוצלחת והוסף את הקוד שלך לדו"ח:

**module** ramzor\_fsm **#** **(** **int** red\_timer **=** 48**,**

**int** red\_yellow\_timer **=** 18**,**

**int** green\_timer **=** 36**,**

**int** yellow\_timer **=** 18

**)**

**(**

**input** **logic** clk**,**

**input** **logic** resetN**,**

**input** **logic** switchN**,**

**input** **logic** endOftime**,** // Time is over . Signal received from the timer

**output** **logic** loadN**,** // Signal to load the timer

**output** **[**7**:**0**]** timer\_val**,** //load the timer value in tens of sec acording to time needed to stay for each state

**output** **logic** redLight**,**

**output** **logic** yellowLight**,**

**output** **logic** greenLight

**);**

**enum** **logic** **[**2**:**0**]** **{**red\_st**,** red\_yellow\_st**,** green\_st**,** yellow\_st**}** present\_state**,** next\_state**;**

// state register

**always\_ff** **@(posedge** clk**,** **negedge** resetN**)**

**if** **(!**resetN**)**

present\_state **<=** red\_st**;**

**else**

present\_state **<=** next\_state**;**

// next state logic

**always\_comb**

**begin**

**case** **(**present\_state**)**

red\_st**:**

**if** **(** endOftime **||** **!**switchN**)** **begin**

next\_state **=** red\_yellow\_st**;**

**end** **else** **begin**

next\_state **=** red\_st**;**

**end**

red\_yellow\_st**:**

**if** **(**endOftime **)** **begin**

next\_state **=** green\_st**;**

**end** **else** **begin**

next\_state **=** red\_yellow\_st**;**

**end**

green\_st**:**

**if** **(**endOftime**)** **begin**

next\_state **=** yellow\_st**;**

**end** **else** **begin**

next\_state **=** green\_st**;**

**end**

yellow\_st**:**

**if** **(**endOftime **)** **begin**

next\_state **=** red\_st**;**

**end** **else** **begin**

next\_state **=** yellow\_st**;**

**end**

**default:** **begin**

next\_state **=** red\_st**;**

**end**

**endcase**

**end**

// Logic for loading the requested time into an auxiliary timer for each one of light state.

**always\_comb** **begin**

**case** **(**present\_state**)**

red\_st**:**

**if** **(** endOftime **||** **!**switchN**)** **begin**

timer\_val **=** red\_yellow\_timer**;**

**end** **else** **begin**

timer\_val **=** red\_timer**;**

**end**

red\_yellow\_st**:**

**if** **(** endOftime **)** **begin**

timer\_val **=** green\_timer**;**

**end** **else** **begin**

timer\_val **=** red\_yellow\_timer**;**

**end**

green\_st**:**

**if** **(** endOftime **)** **begin**

timer\_val **=** yellow\_timer**;**

**end** **else** **begin**

timer\_val **=** green\_timer**;**

**end**

yellow\_st**:**

**if** **(** endOftime **)** **begin**

timer\_val **=** red\_timer**;**

**end** **else** **begin**

timer\_val **=** yellow\_timer**;**

**end**

**default:** **begin**

timer\_val **=** red\_timer**;**

**end**

**endcase**

**end**

**assign** loadN **=** **!(**endOftime **||** **(**present\_state **==** red\_st **&&** **!**switchN**));**

**assign** redLight **=** **(**present\_state **==** red\_st **||** present\_state **==** red\_yellow\_st**)** **?** 1'b1 **:** 1'b0**;**

**assign** yellowLight **=** **(**present\_state **==** yellow\_st **||** present\_state **==** red\_yellow\_st**)** **?** 1'b1 **:** 1'b0**;**

**assign** greenLight **=** **(**present\_state **==** green\_st**)** **?** 1'b1 **:** 1'b0**;**

**endmodule**

## סימולציה

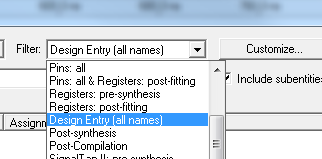
הגדר מה תרצה לבדוק בסימולציה – איזה מצבים מעניינים (המשך למלא את הטבלה)

|  |  |
| --- | --- |
| **מצב** | **תוצאות צפויות** |
| הפעלת **resetN** | כל היציאות מאותחלות |
| מצב Red | הנורה RED דולקת |
| מצב Red+Yellow | הנורה RED והנורה YELLOW דולקות |
| מצב Yellow | הנורה YELLOW דולקת |
| מצב Green | הנורה GREEN דולקת |
| SwitchN למטה במצב RED | מעבר מצב מRED לRED\_YELLOW |

בצע סימולציה, לפי ההוראות המופיעות בהמשך.

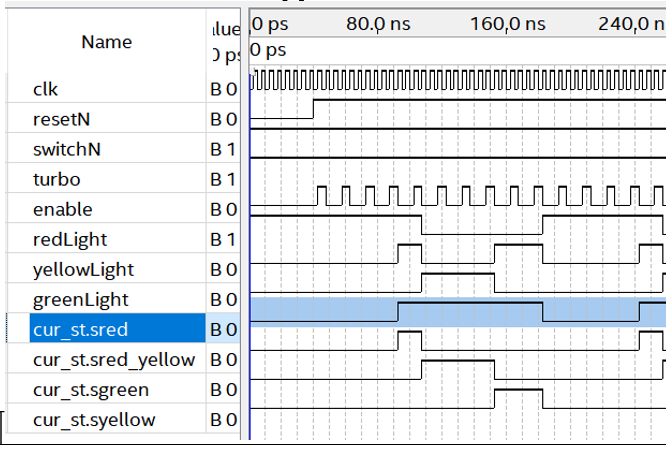
**שים לב היות ובמקרה הרמזור יש לספור הרבה עשיריות שניה בכל מצב (54, 41, וכו') מומלץ להגדיר שעון של 1nsec אם עובדים עם אורך חלון של ברירת המחדל 1usec.**

**הערה**: ניתן לראות בסימולציה את מצבי מכונת המצבים על ידי בחירת **מסנן Filter:  
 Design entry (all names)**

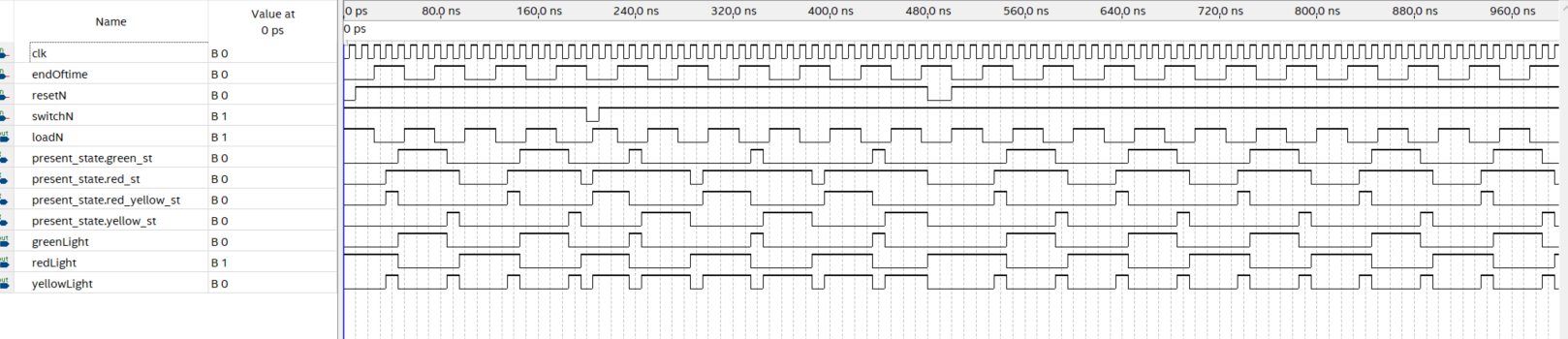
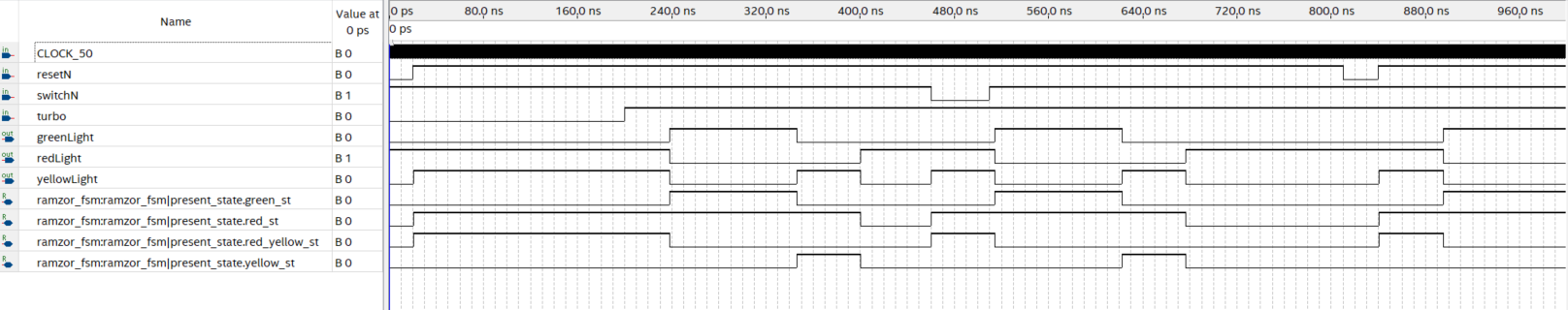


**שים לב** **– יש לבקש להציג בסימולטור** **את כל המצבים כל אחד בנפרד ולא כקבוצה.**

* **המצב הראשון שתגדירו (נניח "אדום") יוצג הפוך (כמו שמוצג מצב idle) כמו בדוגמה להלן.**



הרץ סימולציה והוסף תוצאות סימולציה לדו"ח.

# פצצה – פרוייקטון

**בתרגיל זה יש לבנות מנגנון לפצצה על פי ההגדרות להלן**:

## C:\Users\dudyb\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.Outlook\HWZC6PZ7\20151224_083020.jpgהגדרות הפצצה

1. השעון clk - יש להשתמש בשעון הכרטיס של 50MHz
2. פעולתRESET - אסינכרונית פעילה בנמוך, תאפס את תצוגת והמונים ותמתין ל START
3. פעולת START - תמופה ללחצן ופעילה בנמוך:
   1. בלחיצה על - startN הפצצה רק תטען לזמן הפצצה שהוא -**17 שניות**
   2. בעזיבת ה- startN הפצצה תמנה אחורה עד לאפס (פיצוץ)
4. כששעון הפצצה יגיע לאפס יהיה הבהוב בתצוגה בתדר של  
   1/2 Hz עד ל - RESET חדש
5. פעולת WAIT - תמופה ללחצן ופעילה בנמוך:
   1. בלחיצה על לחצן waitNהמניה תעצור כל זמן הלחיצה.
   2. בשחרור הלחצן המניה תמשיך.
   3. הלחיצה יכולה להיות קצרה לכן יש לדגום אותה במכונת מצבים שרצה ב- 50 MHz
6. יש לבנות את הפצצה באמצעות ארבעה מודולים:
   1. מודול של מכונת המצבים (תמומש בעבודת הכנה זו)
   2. מונה שתי ספרות יורד (BCDDN שמומש בסעיף קודם)
   3. מחלק תדר של 1Hz (קיים ממעבדה קודמת)
   4. תצוגת 7Seg (HEXSS קיים ממעבדה קודמת)
7. לצורך ספירת הזמן ניתן להשתמש ביחידה שהכנתם בתרגיל קודם BCDDN ולטעון לה את הזמן עד להתפוצצות (17 שניות).

**כניסות הפצצה**:

* כניסת שעון clk – השעון שבכרטיס DE10 בתדר 50MHz
* לחצן resetN יכבה את התצוגה. עדיין לא יקרה כלום עד ל- START הראשון
* לחצן startN יטען לשעון את זמן הפיצוץ שנקבע (ערך קבוע) ויפעיל את הספירה מטה
* לחצן waitN - כל לחיצה תעצור את המניה כל זמן הלחיצה

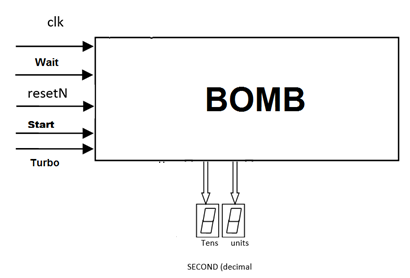
**שימו לב: לחצן לחוץ =0 משוחרר = 1**

* מפסק turbo - יאיץ את פעולת השעון פי 16 לצרכי (DEBUG)

**יציאות הפצצה** יציגו את:

* נוריות ותצוגת 7Seg שמראות את הזמן שנותר עד לפיצוץ בשניות (עשרות ויחידות).
* כשהפצצה מסיימת, התצוגה ((7Seg תהבהב, במצבים 88 וכיבוי לסרוגין.

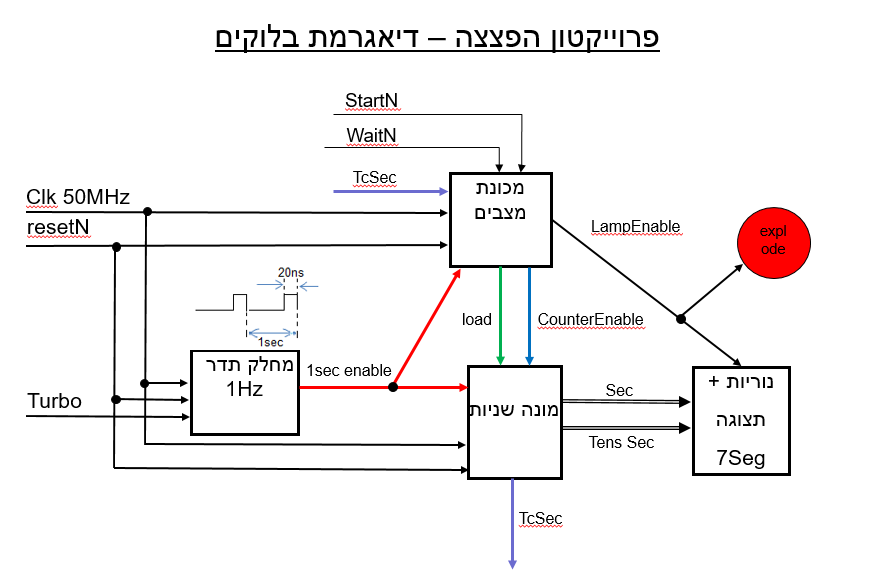
**הערה**: יש להקפיד להוסיףN כסיומת לאותות שהם active Low (לחצנים)



## תכנון הפרויקטון – הפצצה

לאחר שהבנת את דרישות התכן עליך לתכנן כל אחד ואחד מהמודולים, חלקם ימוחזרו מפרויקטים קודמים (בשינויים קלים) ואת חלקם תאלצו לכתוב מבראשית.

דיאגרמת הבלוקים שלהלן מתארת את המודולים המרכיבים את המערכת וחלק מהקשרים ביניהם.



מלאו את הטבלאות הבאות להגדרת המודולים השונים:

**רשום ופרט כל אחד מהמודולים שתרצה למחזר מפרויקטים קודמים**

|  |  |  |
| --- | --- | --- |
| **שינויים דרושים** | **הסבר פעולה** | **שם** |
| Load טוען 17 דצימלי | מונה אחורה שניות | מונה שניות |
| אין צורך בשינויים | תצוגה ספרתית | HEXSS |
| אין צורך בשינויים | מאריך את מחזור השעון | מחלק תדר |

**רשום ופרט כל אחד מהמודולים שתרצה לממש מאפס**

|  |  |  |  |
| --- | --- | --- | --- |
| **יציאות עיקריות** | **כניסות עיקריות** | **הסבר פעולה** | **שם** |
| LampEnable | startN | ניהול התהליך | מכונת מצבים |
| load | waitN |  |  |
| CounterEnable | clk |  |  |
| LampTest | resetN |  |  |
|  | enable |  |  |
|  | TcSec |  |  |

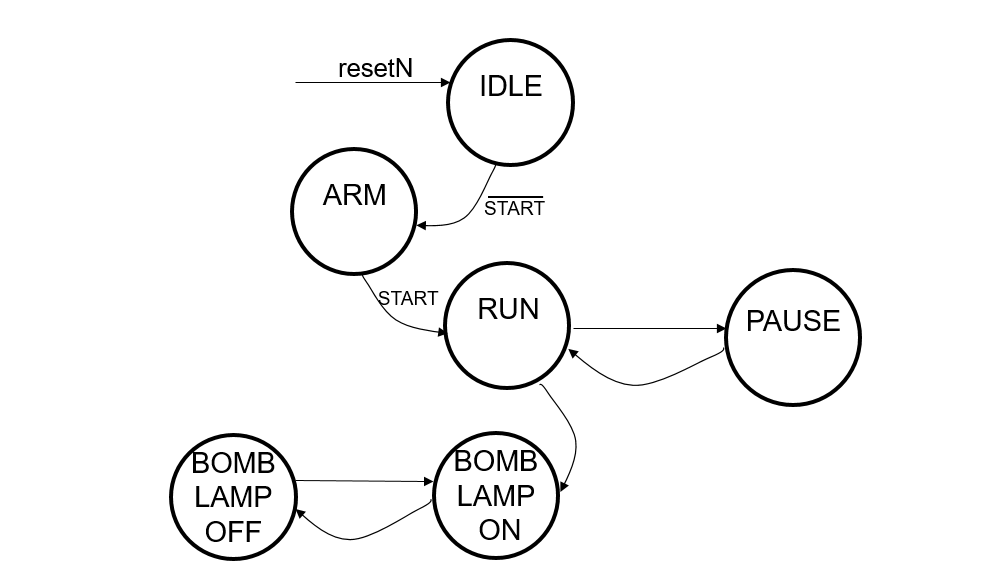
## תכנון מכונת המצבים – הפצצה

**בעבודת הכנה זו** תבנה את מודול הפצצה (מכונת המצבים שלו) לפי השלבים תכנון, השלמת קוד בשלד נתון ובדיקת נכונות הקוד בסימולציה. **במעבדה** תשלים את הפרויקטון, על ידי חיבור המודולים השונים ובדיקתו על הכרטיס.

**הגדר את מכונת המצבים של מודול הפצצה במפורש בטבלה להלן ושרטט את דיאגרמת מצבים** - העזר בשבלונה המוכנה וכתוב את פירוט המעברים.

**השלם בטבלה שלהלן את כל מצבי מכונת המצבים החסרים** ורשום מה עושים בכל מצב ומתי עוברים למצב הבא:

|  |  |  |
| --- | --- | --- |
| **שם המצב** | **פעילות עיקרית** | **לאיזה מצב עוברים מהמצב הנוכחי ובאילו תנאים –** |
| Idle | מאפסים את המונה וממתינים | לחיצה על START מעבירה למצב ARM |
| ARM | טוען 17 שניות למונה יורד | עזיבת לחצן START מעבירה למצב RUN |
| RUN | ספירה אחורנית ל0 | לחיצה על waitN מובילה לPAUSE. בזמן 0 מובילה לLAMPON |
| PAUSE | משהה את הספירה | עזיבת לחצן waitN מובילה RUN |
| LAMP ON | מציג 88 ב7SEG | מוביל לLAMPOFF כעבור שניה |
| LAMP OFF | אורות כבוייים ב7SEG | עובר לLAMPON כעבור שניה |



## מימוש מכונת המצבים

פתח את קובץ השלד של הפצצה הנתון לך (**bomb\_fsm.sv**) והפוך אותו ל- TOP.

השלם את מכונת המצבים על פי השלד הנתון והתכנון שלך.

**שים לב שהמצבים בשלד אינם בהכרח זהים למצבים שעליך לממש, שנה לפי הצורך.**

**שם המכלול: bomb\_fsm**

תאור פעולתו בקצרה:

תשובה: אחראי למעבר מצבים בתכן הפצצה

פירוט כניסות:

|  |  |
| --- | --- |
| **שם כניסה** | **פעולה** |
| Clk | מתאר גל ריבועי שעל פיו מתבצעות פעולות התכן |
| resetN | מחזיר למצב idle |
| startN | מתחיל את פעולת המונה |
| waitN | משהה את פעולת המונה |
| slowClken | נותן חיווי כי עברה שנייה |
| tcSec | נותן חיווי כאשר המונה מגיעה לאפס |

פירוט יציאות:

|  |  |
| --- | --- |
| **שם יציאה** | **פעולה** |
| countEnable | מאפשר פעולת מונה |
| countLoadN | מאתחל ל17 את המונה |
| lampEnable | מאפשר את תצוגת 7SEG |
| lampTest | מדליק את כל הנורות ב7SEG |

**module** bomb\_fsm

**(**

**input** **logic** clk**,**

**input** **logic** resetN**,**

**input** **logic** startN**,** // Start the bomb counter

**input** **logic** waitN**,** // Pause the bomb counter

**input** **logic** slowClken**,** // Flikering the bomb display Onesec ON Onesec OFF

**input** **logic** tcSec**,** // Trigger to explode the bomb

**output** **logic** countEnable**,** // Enable the count down counter

**output** **logic** countLoadN**,** // Load the bomb down count counter.

**output** **logic** lampEnable**,** // Turn the bomb display to ON - Show the number 00

**output** **logic** lampTest

**);**

**enum** **logic** **[**2**:**0**]** **{**Sidle**,** Sarm**,** Srun**,** Spause**,** SlampOn**,** SlampOff**}** prState**,** nxtState**;**

**always** **@(posedge** clk **or** **negedge** resetN**)**

**begin**

**if** **(** **!**resetN **)** // Asynchronic reset

prState **<=** Sidle**;**

**else** // Synchronic logic FSM

prState **<=** nxtState**;**

**end** // always

**always\_comb** // Update next state and outputs

**begin**

nxtState **=** prState**;** // default values

countEnable **=** 1'b0**;**

countLoadN **=** 1'b1**;**

lampEnable **=** 1'b1**;**

lampTest **=** 1'b0**;**

**case** **(**prState**)**

Sidle**:** **begin**

lampEnable **=** 1'b0**;**

**if** **(**startN **==** 1'b0**)**

nxtState **=** Sarm**;**

**end** // idle

Sarm**:** **begin**

countLoadN **=** 1'b0**;**

**if** **(**startN **==** 1'b1**)** //Initiat the bomb when the start key is pressed

nxtState **=** Srun**;**

**end** // arm

Srun**:** **begin**

countEnable **=** 1'b1**;**

countLoadN **=** 1'b0**;**

**if** **(**tcSec **==** 1'b1**)** // Check if time is over

nxtState **=** SlampOn**;**

**else** **if** **(**waitN **==** 1'b0**)**

nxtState **=** Spause**;**

**end** // run

Spause**:** **begin**

countEnable **=** 1'b0**;**

**if** **(**waitN **==** 1'b1**)** // As long as the wait key is pressed it pauses the timer

nxtState **=** Srun**;**

**end** // pause

// The next two states blink the display.

SlampOn**:** **begin**

lampEnable **=** 1'b1**;**

lampTest **=** 1'b1**;**

**if** **(**slowClken **==** 1'b1**)**

nxtState **=** SlampOff**;**

**end** // lampOn

SlampOff**:** **begin**

lampEnable **=** 1'b0**;**

**if** **(**slowClken **==** 1'b1**)**

nxtState **=** SlampOn**;**

**end** // lampOff

**endcase**

**end** // always comb

**endmodule**

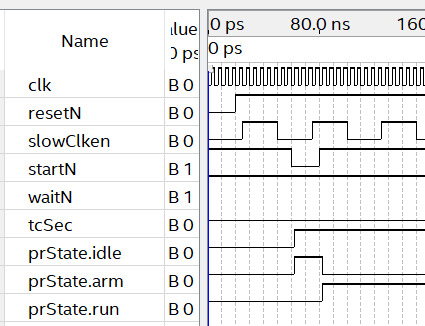
בצע אנליזה מוצלחת לתכן.

## סימולציה של מכונת המצבים

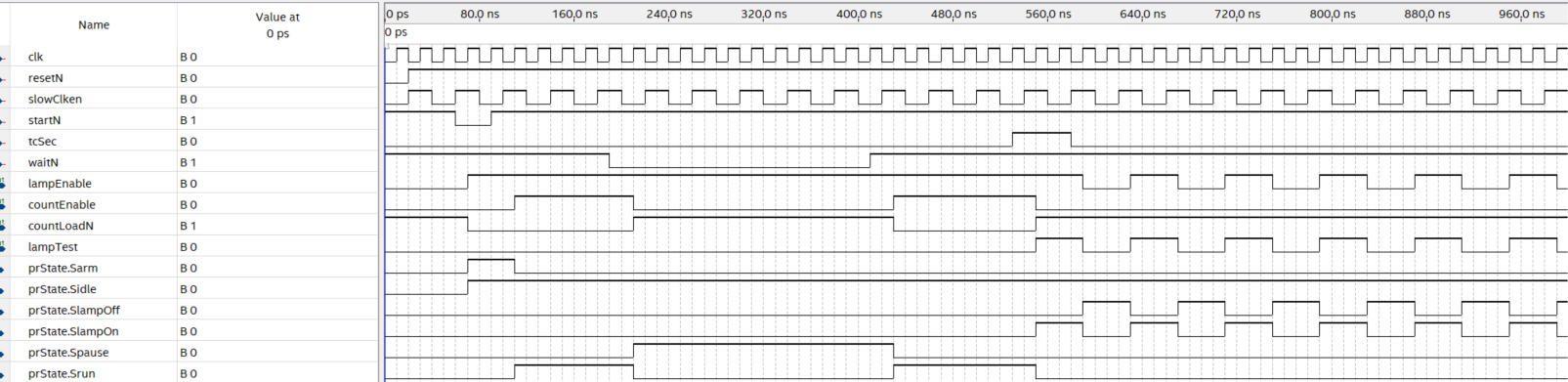
הגדר מה תרצה לבדוק בסימולציה – איזה מצבים מעניינים (המשך למלא את הטבלה)

|  |  |
| --- | --- |
| **מצב** | **תוצאות צפויות** |
| יציאה מRESET | מעבר למצב idle |
| לחיצה על START | מעבר לARM |
| עזיבת START | מעבר לRUN |
| לחיצה על waitN | מעבר לPAUSE |
| עזיבת waitN | חזרה לRUN |
| גמר המונה | הבהוב על ידי מעבר בין מצבי LAMPON וLAMPOFF |

**להזכירך** – **בסימולציה המצב הראשון,Idle , יוצג הפוך** כפעיל בנמוך, יתר המצבים יופיעו נכון, כפעילים בגבוה. יש לבחור כל מצב בנפרד (בשורה אחרת)

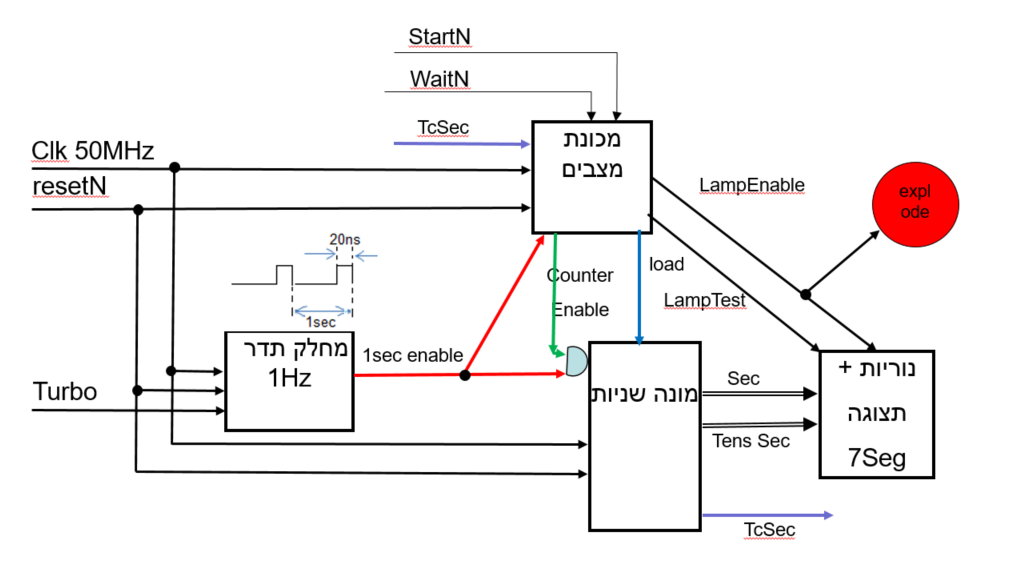


הרץ סימולציה של מכונת המצבים והוסף את תוצאות הסימולציה לדו"ח.



## השלמה של דיאגרמת בלוקים מלאה של הפצצה

לאחר שמימשת את מכונת המצבים והמונה השלם את דיאגרמת הבלוקים הנתונה (השקף הנתון במודל) ע"י הוספת כל הקשרים בין המודולים (אין צורך לבנות שרטוט גרפי בקוורטוס).



# גיבוי העבודה

שמור את הפרויקט רגיל וגם כארכיב (באמצעות Project -> Archive Project).

תגבה את קובץ הארכיב, העלה למודל והבא למעבדה כי תצטרך אותו בניסוי.

**להזכירכם - יש להביא למעבדה את כל הקבצים – כי תשתמשו בהם**

***לאחר שסיימת - לחץ על ה LINK ומלא בבקשה את השאלון המצורף***

|  |
| --- |
|  |
| |  | | --- | | [**מלא את הטופס**](https://docs.google.com/forms/d/1tO1v_J1GNnuBFqGURbuZZzX8uuGXzKp8RdkdBOaakfY/viewform?c=0&w=1&usp=mail_form_link) | |
|  |

שמור דו"ח זה כ-PDF והעלה למודל