

הטכניון - מכון טכנולוגי לישראל
הפקולטה להנדסת חשמל



מעבדה בהנדסת חשמל
1א' 044157

ניסוי SV2
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

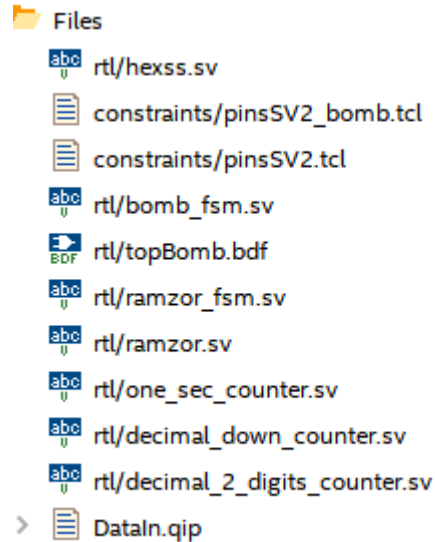
סטודנט	שם פרטי	שם משפחה
1	נועם	אילתה
2	ליאור	דביר

תוכן עניינים

3.....	פתיחת ארכיב	1
3.....	מונה דצימלי יורד 2 ספרות	2
3	2.1 הגדרות – מונה דצימלי יורד	
3.....	Module interface	2.1.1
4.....	Truth table	2.1.2
5	2.2 מונה decimal_2_digits_counter יורד לשתי ספרות	
6.....	Module interface	2.2.1
6.....	Truth table decimal_2_digits_counter	2.2.2
6.....	2.2.3 מימוש המונה לשתי ספרות	
8	2.3 סימולציה	
10.....	רמזור מבוקר	3
10	3.1 הצגת הדרישות	
10.....	Ramzor Module interface	3.1.1
10.....	Ramzor Module Block diagram	3.1.2
11	3.2 RAMZOR_FSM	
11.....	RAMZOR_FSM Module interface	3.2.1
11.....	RAMZOR_FSM bubble diagram	3.2.2
12	3.3 Aux_timer	
12.....	Aux_Timer Module interface	3.3.1
13	3.4 onetens_sec_counter	
13.....	onetens_sec_counter Module interface	3.4.1
13	3.5 קוד SV	
15	3.6 סימולציה	
17.....	פצצה – פרויקטון	4
17	4.1 הגדרות הפצצה	
18	4.2 תכנון הפרויקטון – הפצצה	
19	4.3 תכנון מכונת המצבים – הפצצה	
20	4.4 מימוש מכונת המצבים	
22	4.5 סימולציה של מכונת המצבים	
23	4.6 השלמה של דיאגרמת בלוקים מלאה של הפצצה	
24.....	גיבוי העבודה	5

1 פתיחת ארכיב

צור תיקיה למעבדה זו. הורד מהמודל קובץ ארכיב של המעבדה ופתח אותו לפרויקט בתיקייה שיצרת. ודא תכולת קבצים כזו:



2 מונה דצימלי יורד 2 ספרות

2.1 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 2.1.1

תכננו מונה דצימלי יורד כמתואר בהמשך והוסיפו את הקוד שלכם בקובץ - decimal_down_counter.sv

להלן הכניסות והיציאות של יחידת המונה הדצימלי היורד :

Direction	width	Name
input	1	clk
input	1	resetN
Input	1	ena
Input	1	ena_cnt
Input	1	loadN
input	[3:0]	datain
output	[3:0]	count
output	1	tc

Truth table 2.1.2

יש לתכנן את קוד המונה לפי טבלת האמת הבאה:

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

היציאה האסינכרונית tc מתוארת להלן:

CLK	resetN	ena	ena_cnt	loadN	count[3:0]	tc Async
x	1	x	x	x	4'b0000	1

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-SV 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 // Asynchronous 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
    end
end

```

```

        end else if (count == 4'b0) begin
            count <= 4'h9;
        end else begin
            count <= count - 1'b1;
        end
    end //Synch
end //always

// Asynchronous tc
assign tc = (count == 4'b0 && resetN) ? 1'b1 : 1'b0;

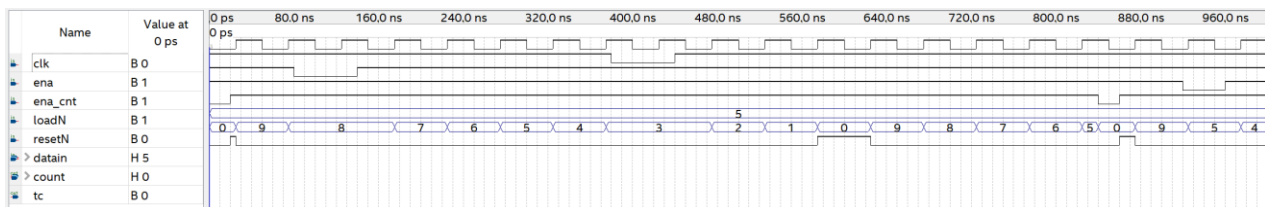
endmodule

```

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

Flow Summary	
<<Filter>>	
Flow Status	Successful - Sun Aug 16 14:38:30 2020
Quartus Prime Version	17.0.0 Build 595 04/25/2017 SJ Lite Edition
Revision Name	SV2Lab
Top-level Entity Name	decimal_down_counter
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	4
Total pins	14
Total virtual pins	0
Total block memory bits	0
Total DSP Blocks	0
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0
Total DLLs	0

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



2.2 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.

- עמוד 5 – ניסוי SV2, דוח הכנה

Module interface 2.2.1

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

Direction	width	Name
input	1	clk
input	1	resetN
Input	1	ena
Input	1	ena_cnt
Input	1	loadN
input	[7:0]	Data_init
Output	[7:0]	Count_out
Output	1	tc

Truth table decimal_2_digits_counter 2.2.2

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

CLK	resetN	ena	ena_cnt	loadN	Data_init[7:0]	Count_out[7:0]	tc Async
x	0	x	x	x	x	8'd0	1'b0
↑	1	x	x	0	Data_init [7:0]	Data_init [7:0]	1'b0
↑	1	0	x	1	x	x	1'b0
↑	1	x	0	1	x	x	1'b0
↑	1	1	1	1	x	Count_out[7:0]-1	1'b0
x	1	x	x	x	x	8'd0	1'b1

2.2.3 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 && resetN) ? 1'b1 : 1'b0;

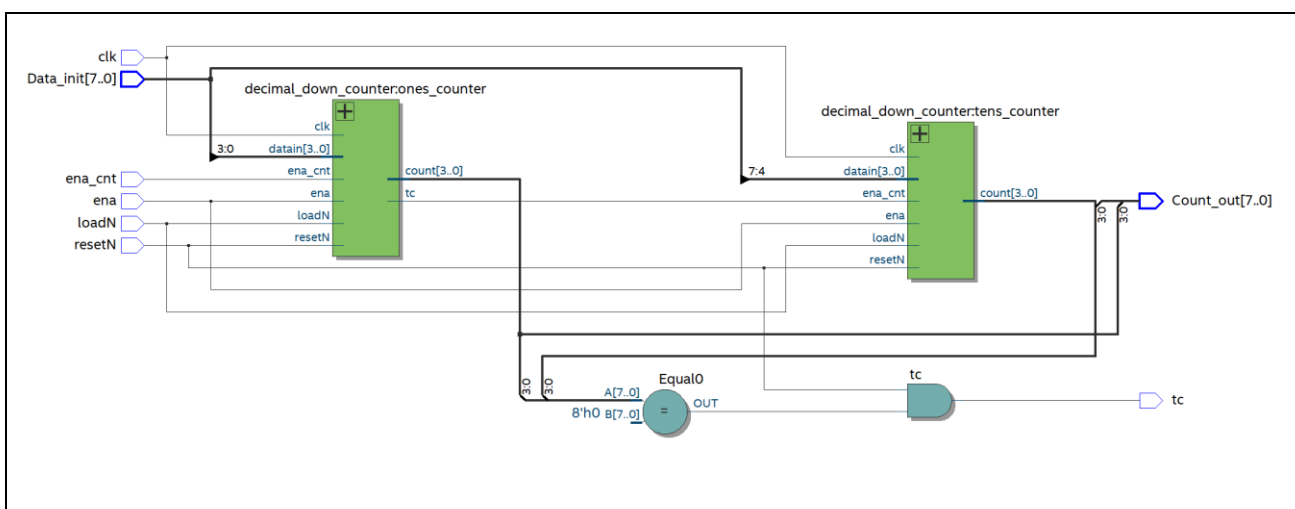
endmodule

```

הרץ סינתזה מוצלחת לקובץ decimal_2_digits_counter.sv וצירף את הסיכום לדו"ח:

Flow Summary	
<<Filter>>	
Flow Status	Successful - Sun Aug 16 15:24:31 2020
Quartus Prime Version	17.0.0 Build 595 04/25/2017 SJ Lite Edition
Revision Name	SV2Lab
Top-level Entity Name	decimal_2_digits_counter
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	8
Total pins	22
Total virtual pins	0
Total block memory bits	0
Total DSP Blocks	0
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0
Total DLLs	0

הצג את המימוש של המונה decimal_2_digits_counter.sv כ- RTL VIEW והוסף אותו לדו"ח:



2.3 סימולציה

הרץ סימולציה מלאה לתוכן decimal_2_digits_counter.sv בה תבדוק את כל הכניסות והיציאות המעניינות ואת כל מקרי הקצה המעניינים.

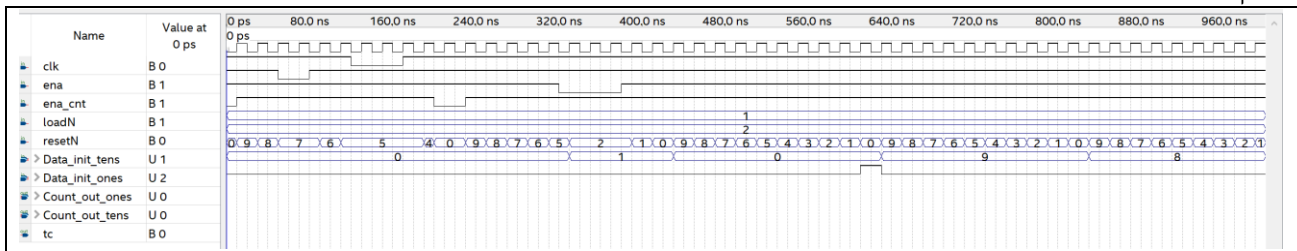
הגדר מה תרצה לבדוק בסימולציה – איזה מצבים מעניינים (המשך למלא את הטבלה)

תוצאות צפויות	מצב
כל היציאות מאותחלות	הפעלת resetN

הספירה נשארת קבועה	ENA_CNT COUNT בלי
הספירה נשארת קבועה	ENA COUNT בלי
הכנסת ערכי Data_in	loadN הפעלת
1 אם הספירה (Count_out) הגיעה ל 0 ו resetN הוא 1, אזרת 0	TC

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

הוסף את תוצאות הסימולציה לדו"ח.



3 רמזור מבוקר

בתרגיל זה עליכם לבנות רמזור באמצעות מכונת מצבים ושימוש במונה 8 Bit שמימשם בסעיף 2 .
בעבודת הכנה זן 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.

3.1 הצגת הדרישות

פעולת הרמזור:

- **מצבי הרמזור הם אדום, אדום_צהוב, ירוק, צהוב.**
- זמני הרמזור בכל מצב:
 - 4.8 שניות באדום
 - 3.6 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.

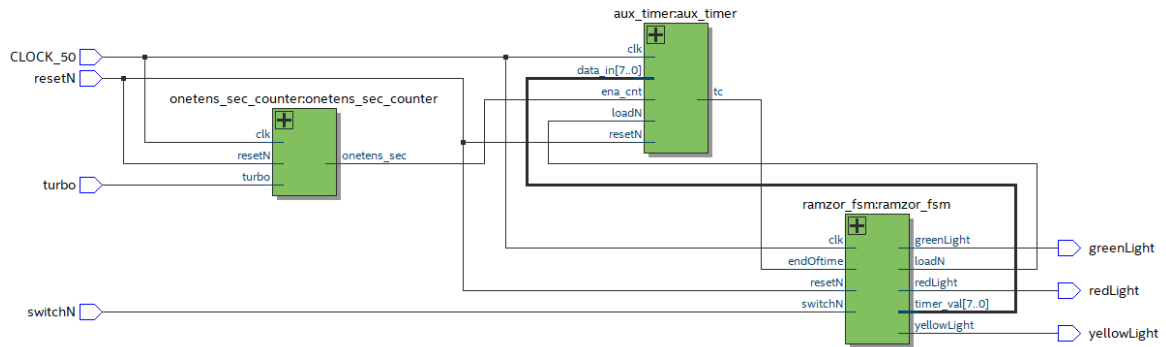
3.1.1 Ramzor Module interface

להלן הכניסות והיציאות של הרמזור:

Direction	width	Name
input	1	CLOCK_50
input	1	resetN
Input	1	turbo
Input	1	switchN
Input	1	loadN
output	1	greenLight
output	1	redLight
output	1	yellowLight

3.1.2 Ramzor Module Block diagram

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



RAMZOR_FSM 3.2

RAMZOR_FSM Module interface 3.2.1

להלן הכניסות והיציאות עבור הבלוק של מכונת המצבים:

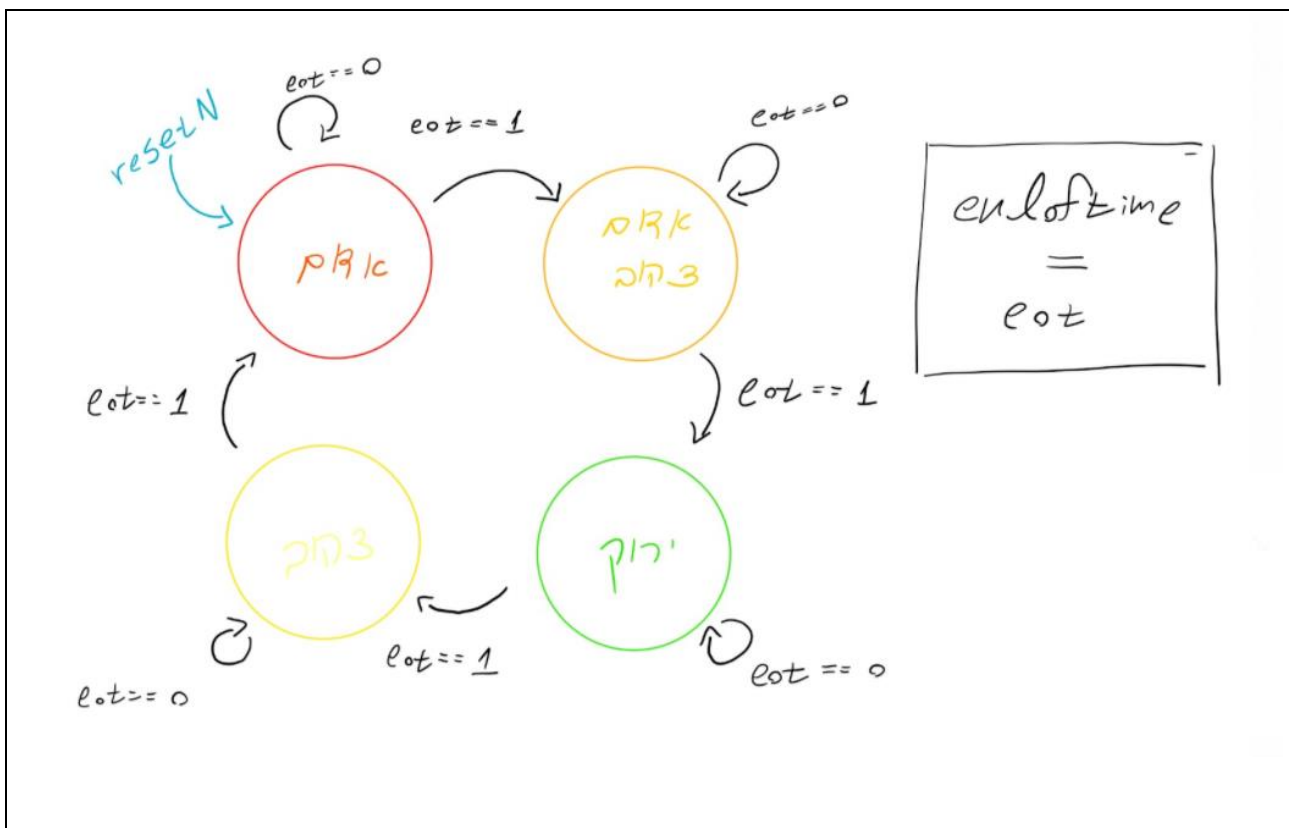
Direction	Type	width	Name
input	logic	1	clk
input	logic	1	resetN
Input	logic	1	switchN
Input	logic	1	cntDn
output	logic	1	greenLight
output	logic	1	redLight
output	logic	1	yellowLight

הפרמטרים:

Parameter	Type	Default value
red_timer	int	48
red_yellow_timer	int	18
green_timer	int	36
yellow_timer	int	18

RAMSOR_FSM bubble diagram 3.2.2

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



Aux_timer 3.3

רכיב זה מבוסס על המונה בקובץ **decimal_down_counter.sv** עם מעט שינויים. רכיב זה מיצר פולס במשך זמן של שעות 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 3.3.1

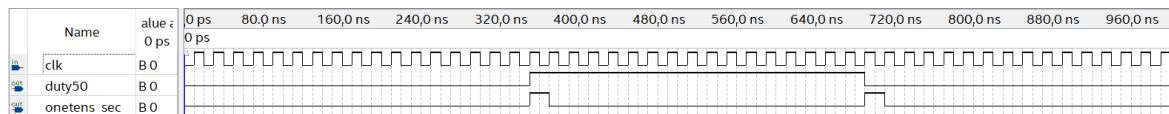
להלן הכניסות והיציאות של מונה הזמן של הרמזור. שימו לב שלשעות בניגוד למונה יש יציאה של bit אח (tc) ללא יציאת ערך המונה:

Direction	width	Name
input	1	clk
input	1	resetN
Input	1	ena_cnt
Input	1	loadN
input	[7:0]	data_in
output	1	tc

3.4 onetens_sec_counter

רכיב זה מייצר פולסים בתדר של 1/10 שניה ברוחב פולסי השעון המהיר CLOCK_50 (או פעם ב-16 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



3.4.1 onetens_sec_counter Module interface

להלן הכניסות והיציאות של המונה

Direction	width	Name	
input	1	clk	
input	1	resetN	
Input	1	turbo	מקצרת מחזור 1/16
output	1	onetens_sec	

3.5 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 according 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
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

```

3.6 סימולציה

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

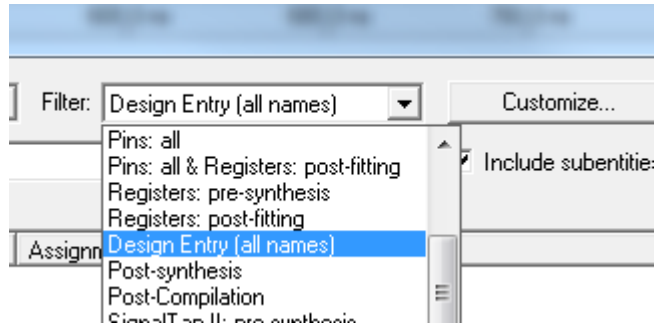
תוצאות צפויות	מצב
כל היציאות מאותחלות	הפעלת resetN
הנורה RED דולקת	מצב Red
הנורה RED והנורה YELLOW דולקות	מצב Red+Yellow
הנורה YELLOW דולקת	מצב Yellow
הנורה GREEN דולקת	מצב Green
מעבר מצב מ RED ל RED_YELLOW	SwitchN למטה במצב RED

בצע סימולציה, לפי ההוראות המופיעות בהמשך.

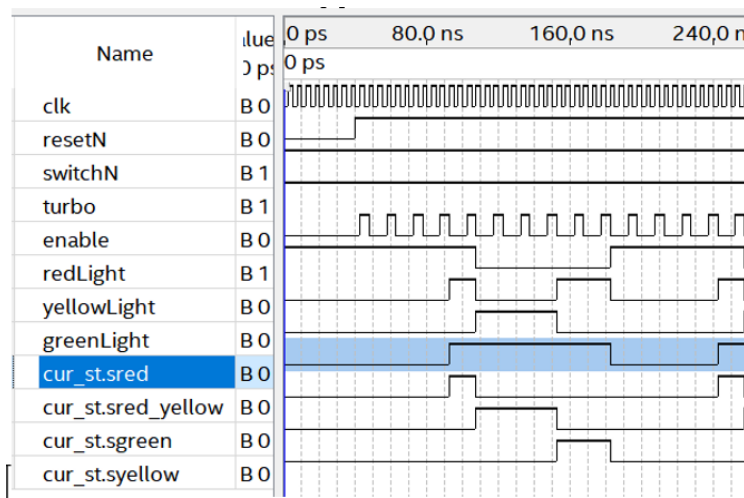
שים לב היות ובמקרה הרמזור יש לספור הרבה עשיריות שניה בכל מצב (54, 41, וכו') מומלץ להגדיר שעון של 1nsec אם עובדים עם אורך חלון של ברירת המחדל 1usec.

הערה: ניתן לראות בסימולציה את מצבי מכונת המצבים על ידי בחירת מסנן Filter:

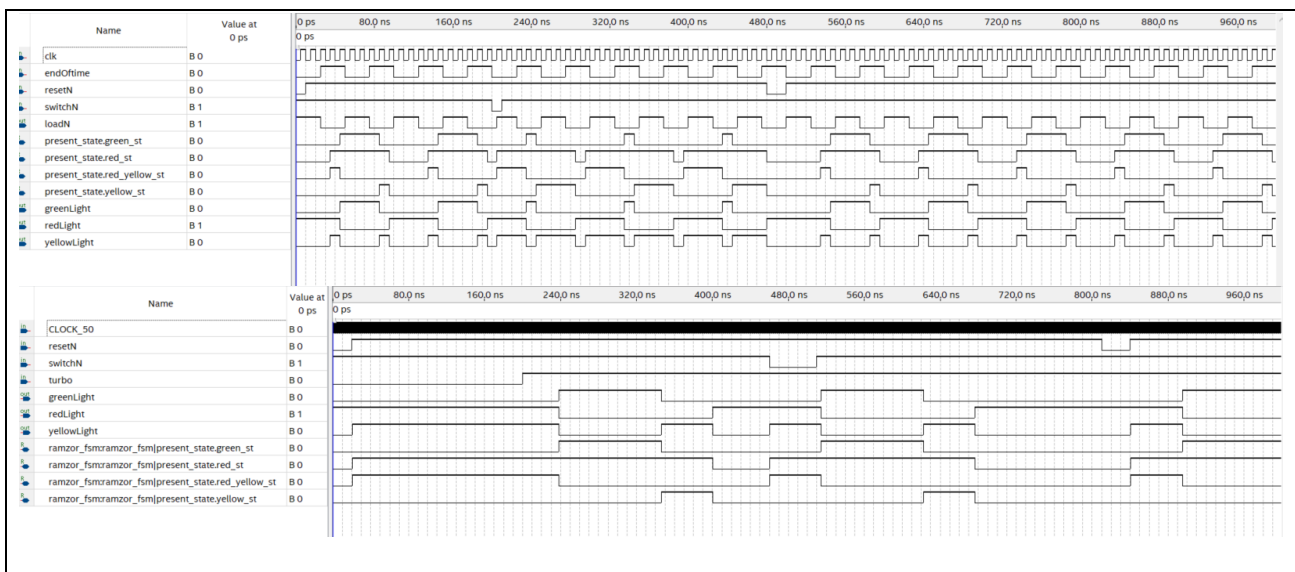
Design entry (all names)



שם 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.



4 פצצה – פרוייקטון

בתרגיל זה יש לבנות מנגנון לפצצה על פי ההגדרות להלן:

4.1 הגדרות הפצצה

1. השעון clk - יש להשתמש בשעון הכרטיס של 50MHz
2. פעולת RESET - אסינכרונית פעילה בנמוך, תאפס את תצוגת והמונים ותמתין ל START
3. פעולת START - תמופה ללחצן ופעילה בנמוך:
- a. בלחיצה על - startN הפצצה רק תטען לזמן הפצצה שהוא -17

שניות

- b. בעזיבת ה- startN הפצצה תמנה אחורה עד לאפס (פיצוץ)
4. כששעון הפצצה יגיע לאפס יהיה הבהוב בתצוגה בתדר של $1/2 \text{ Hz}$ עד ל- RESET חדש
5. פעולת WAIT - תמופה ללחצן ופעילה בנמוך:
- a. בלחיצה על לחצן waitN המניה תעצור כל זמן הלחיצה.
- b. בשחרור הלחצן המניה תמשיך.
- c. הלחיצה יכולה להיות קצרה לכן יש לדגום אותה במכונת מצבים שרצה ב- 50 MHz
6. יש לבנות את הפצצה באמצעות ארבעה מודולים:
- a. מודול של מכונת המצבים (תמומש בעבודת הכנה זו)
- b. מונה שתי ספרות יורד (BCDDN שמומש בסעיף קודם)
- c. מחלק תדר של 1Hz (קיים ממעבדה קודמת)
- d. 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:

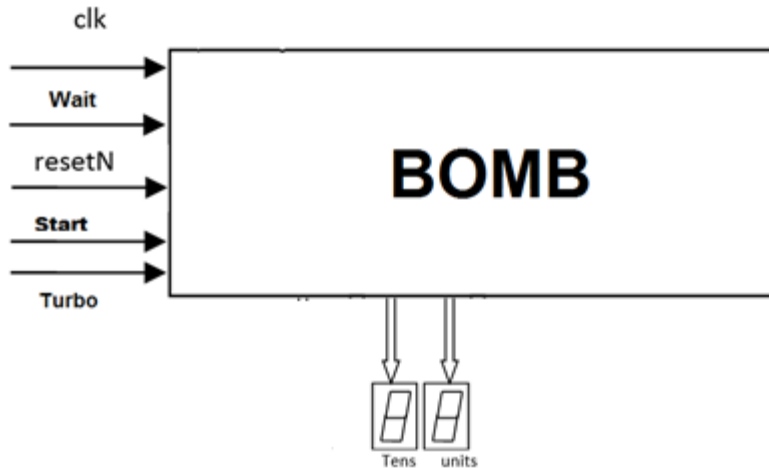
- כניסת שעון clk – השעון שבכרטיס DE10 בתדר 50MHz
- לחצן resetN יכבה את התצוגה. עדיין לא יקרה כלום עד ל- START הראשון
- לחצן startN יטען לשעון את זמן הפיצוץ שנקבע (ערך קבוע) ויפעיל את הספירה מטה
- לחצן waitN - כל לחיצה תעצור את המניה כל זמן הלחיצה
- שימו לב: לחצן לחוץ = 0 משוחרר = 1
- מפסק turbo - יאיץ 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 להוסיף N כסיומת לאותות שהם active Low (לחצנים)



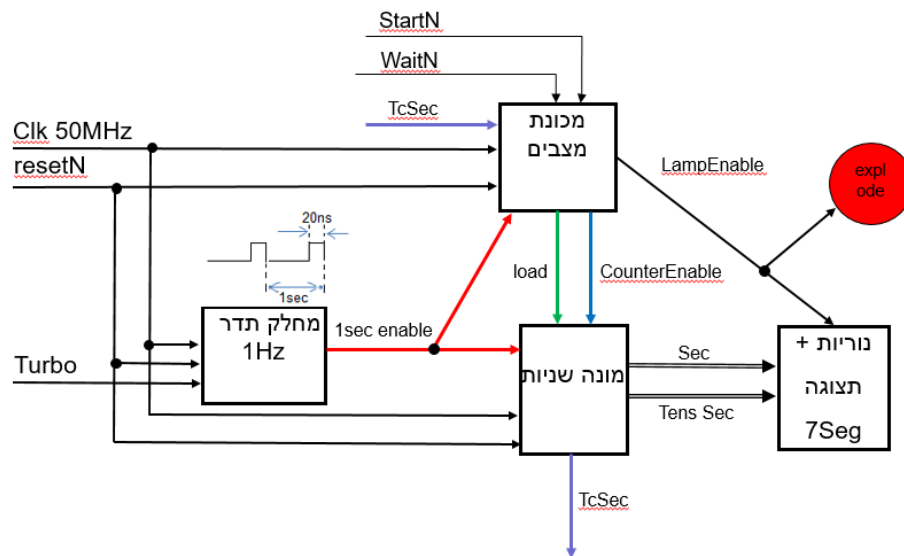


4.2 תכנון הפרויקטון – הפצה

לאחר שהבנת את דרישות התכנן 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.

פרויקטון הפצה – דיאגרמת בלוקים



מלאו את הטבלאות הבאות להגדרת המודולים השונים:

רשום ופרט כל אחד מהמודולים שתמצא למחזור מפרויקטים קודמים

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

רשום ופרט כל אחד מהמודולים שתמצה לממש מאפס

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

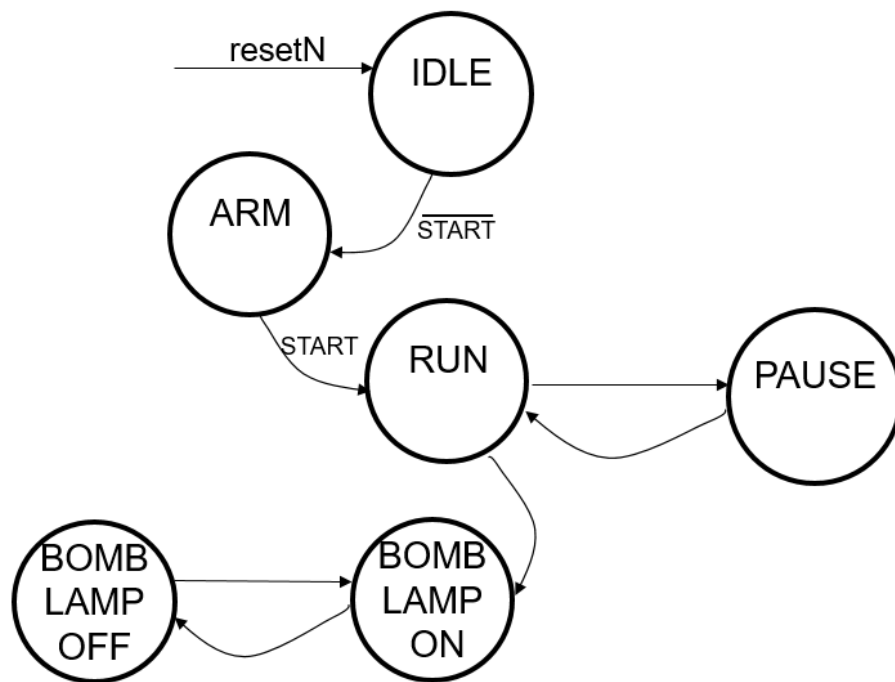
4.3 תכנון מכונת המצבים – הפצצה

בעבודת הכנה זו תבנה את מודול הפצצה (מכונת המצבים שלך) 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 ל 0	לחיצה על waitN מובילה ל PAUSE. בזמן 0 מובילה ל LAMPON
PAUSE	משהה את הספירה	עזיבת לחצן waitN מובילה ל RUN
LAMP ON	מציג 88 ב SEG7	מוביל ל LAMPOFF כעבור שניה
LAMP OFF	אורות כבויים ב SEG7	עובר ל LAMPON כעבור שניה



4.4 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 בתוך 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 ל 17 את המונה
lampEnable	מאפשר את תצוגת SEG7

```

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,       // Flickering the bomb display Onsec 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 ) // Asynchronous 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;
                if (tcSec == 1'b1) // Check if time is over
                    nxtState = SlampOn;
                else if (waitN == 1'b0)

```

```

        nextState = Spause;
    end // run
    Spause: begin
        countEnable = 1'b0;
        if (waitN == 1'b1) // As long as the wait key is pressed it
pauses the timer
            nextState = Srun;
        end // pause
    // The next two states blink the display.
    SlampOn: begin
        lampEnable = 1'b1;
        lampTest = 1'b1;
        if (slowClken == 1'b1)
            nextState = SlampOff;
        end // lampOn
    SlampOff: begin
        lampEnable = 1'b0;
        if (slowClken == 1'b1)
            nextState = SlampOn;
        end // lampOff
    endcase
end // always comb
endmodule

```

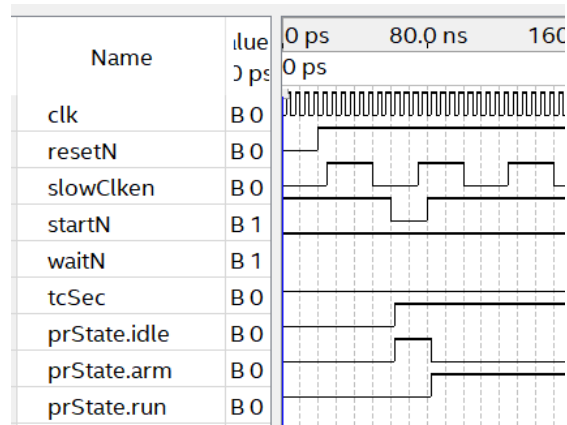
בצע אנליזה מוצלחת לתכן.

4.5 סימולציה של מכונת המצבים

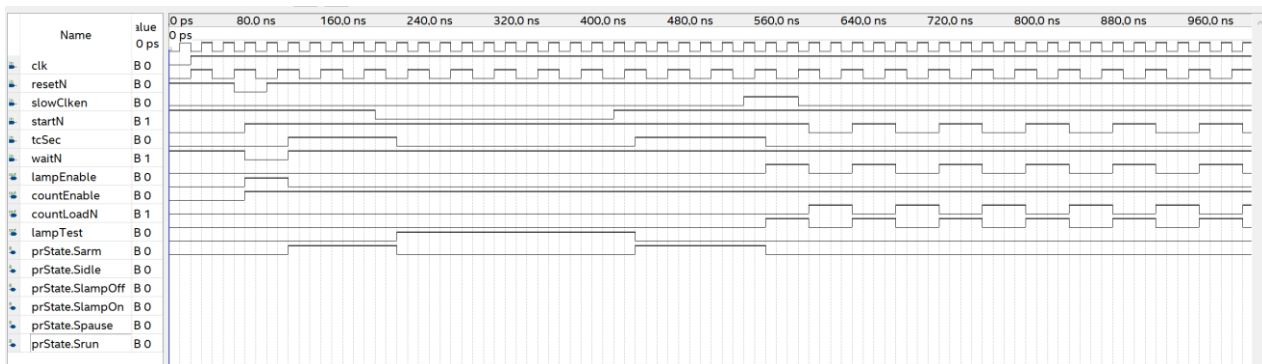
הגדר מה תרצה לבדוק בסימולציה – איזה מצבים מעניינים (המשך למלא את הטבלה)

מצב	תוצאות צפויות
יציאה מ־RESET	מעבר למצב idle
לחיצה על START	מעבר ל־ARM
עזיבת START	מעבר ל־RUN
לחיצה על waitN	מעבר ל־PAUSE
עזיבת waitN	חזרה ל־RUN
גמר המונה	הבהוב על ידי מעבר בין מצבי LAMPON ו־LAMPOFF

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

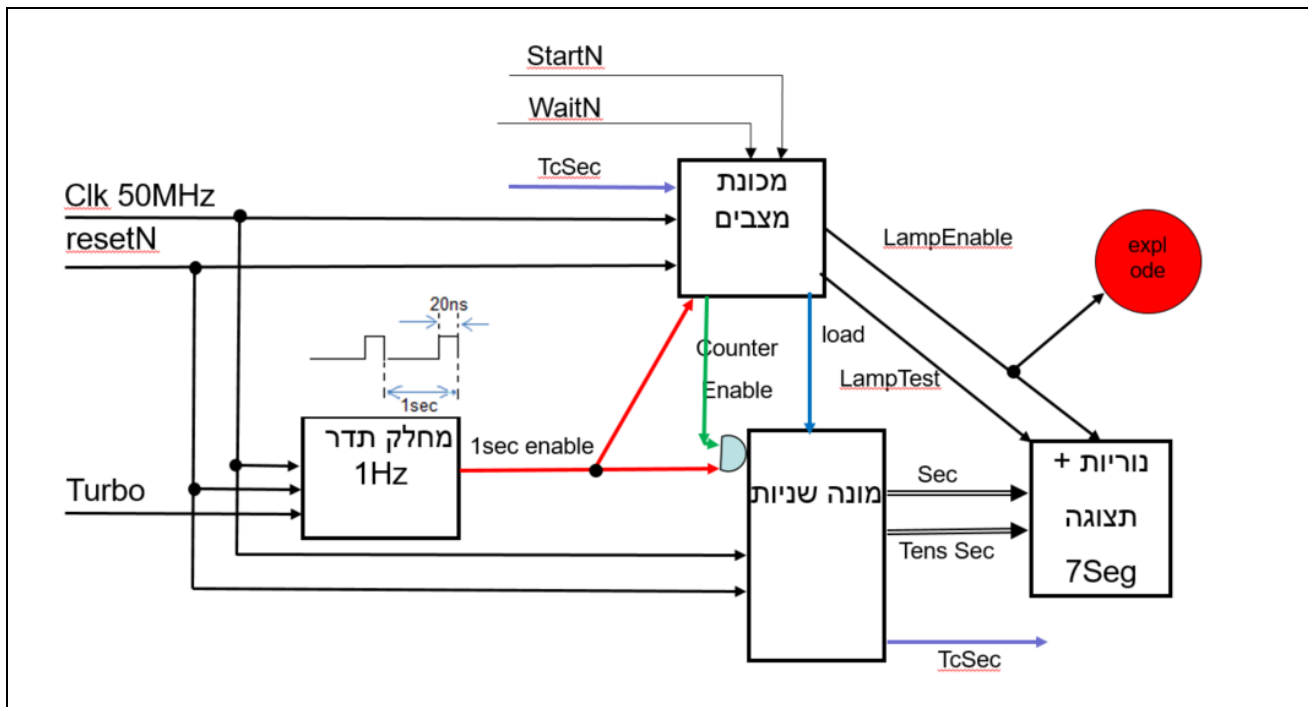


הרץ סימולציה של מכונת המצבים והוסף את תוצאות הסימולציה לדו"ח.



4.6 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).



5 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

שמור דו"ח זה כ- PDF והעלה למודל