הטכניון - מכון טכנולוגי לישראל
הפקולטה להנדסת חשמל

ניסוי VHDL2 - שאלות ודוח הכנה

גרסה 1.02

קיץ 2018

מחבר: אברהם קפלן, דודי בר-און

על פי חוברת של עמוס זסלבסקי מ 2009

תאריך הגשת דו"ח ההכנה	
שם המדריך	

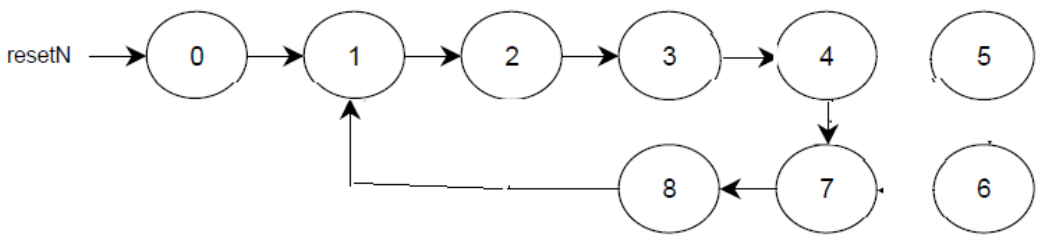
סטודנט	שם פרטי	שם משפחה
1	ברק	זן
2	בועז	טייטלר

שימו לב: הפרויקט שתפתחו ותממשו בסעיפים 2 , 3, 4 להלן, ישמש אתכם במעבדה.
בסיומו, יש לבצע עליו פעולת ARCHIVE ב QUARTUS (כמתואר בפרק 16 של quartus cook book 17 במודל).
את הקובץ המכווץ שתקבלו מפעולה זו יש להעלות במודל ל
QAR דחוס של VHDL2 - דוח הכנה

Contents

1 מונה ציקלי

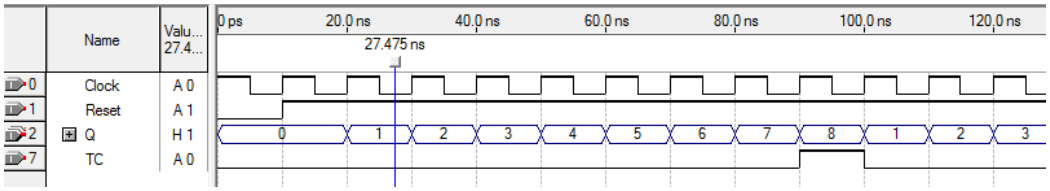
בצע תכן של מונה סינכרוני עולה, שהוא בעל מצבים מ-1 ועד 8 ומדלג על המצבים 5,6



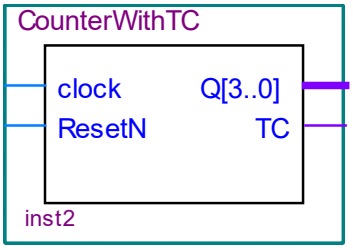
5 ו 6.

יש לממש באמצעות מונה ולא במכונת מצבים.

דוגמה לסימולציה של מונה מ- 1 עד 8 (**שונה** מהתרגיל שלנו) :



המונה מוציא TC אסינכרוני ויציאה סינכרונית של BIT 4



הקפד על תכן סינכרוני נקי למונה לפי הכללים שהוגדרו בחומר רקע לניסוי תוך שימוש בקוד יעיל.

הערה: עליך לזהות את מצב 8 ואז:

- להוציא TC - מיידית (CARRY לספרה הבאה)
- לטעון למונה את המספר -1 בשעון הבא

1.1 מימוש סינכרני של היציאה Q

כתוב תהליך סינכרוני המייצר את היציאות Q ומחזיק משתנה פנימי Qtmp

קוד VHDL

```
process(CLK,RESETN)
 begin
   if rising_edge(CLK) then
      if (RESETN = '0' or unsigned(Qtemp) = 8) then
         Qtemp <= (others => '0');
      elsif (unsigned(Qtemp) = 4 ) then
         Qtemp <= "0111" ;
      else
         Qtemp <= std_logic_vector(unsigned(Qtemp) + 1) ;
      end if;
   end if;
end process;

Q <= Qtemp ;
```

1.2 מימוש אסינכרוני של היציאה - TC

הוסף תהליך **אסינכרוני** המייצר את TC מהמשתנה הפנימי Qtmp

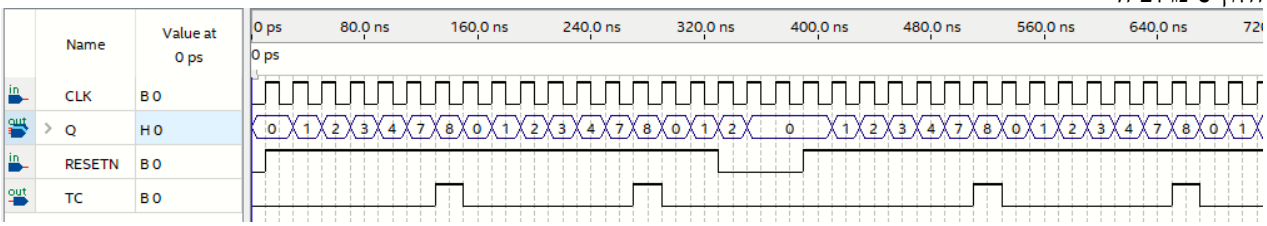
קוד VHDL של התוספת בלבד

```
process(Qtemp)
 begin
   if( Qtemp = 8 ) then
      TC <= '1' ;
   else
      TC <= '0' ;
   end if;
end process;
```

לאחר שהקומפילציה עברה בהצלחה בצע לקובץ זה סימולצית **functional** מלאה בה תבדוק את כל הכניסות והיציאות ואת כל מקרי הקצה, כולל בדיקת הכניסה ResetN באמצע פעולת המונה ולא רק בתחילת הסימולציה (בנגוד לדוגמת הסימולציה למעלה).

יש להקפיד להציג את המספרים בצורה יפה RADIX = HEX ולא בבינארי

חלון סימולציה



1.3 טעינה סינכרונית

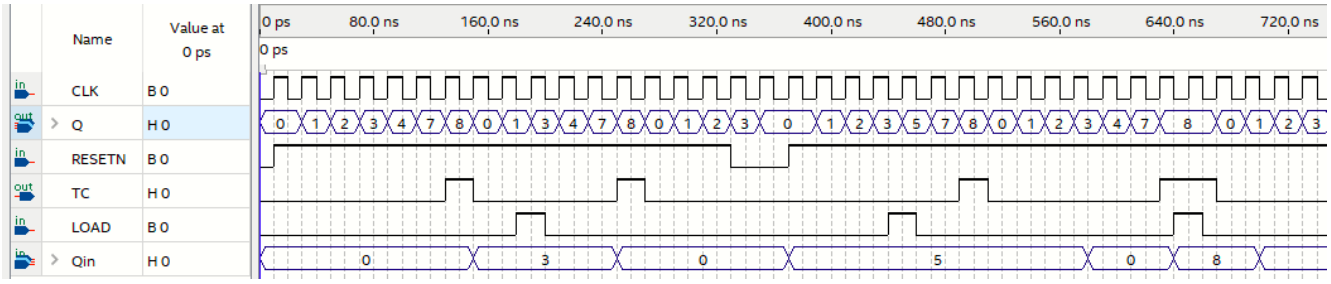
הוסף וקטור כניסה של 4 סיביות וכניסה סינכרונית LOAD (פעילה בגבוה) , המאפשרת לטעון כל מספר חוקי (קרי קטן מ 8) שנבחר למונה

קוד VHDL

```
entity CounterWithTC is
    port ( CLK,RESETN,LOAD : in std_logic ;
           Qin : in std_logic_vector(3 downto 0) ;
           Q : out std_logic_vector(3 downto 0) ;
           TC : out std_logic ) ;
end CounterWithTC ;

architecture arc_CounterWithTC of CounterWithTC is
 signal Qtemp : std_logic_vector(3 downto 0) ;
begin
   process(CLK,RESETN)
    begin
      if rising_edge(CLK) then
         if (RESETN = '0') then  --reset
            Qtemp <= (others => '0');
         elsif (LOAD = '1') then --LOAD Qin
            Qtemp <= Qin ;
         elsif (unsigned(Qtemp) = 8) then --8 (restat)
            Qtemp <= (others => '0');
         elsif (unsigned(Qtemp) = 4 or unsigned(Qtemp) = 5 ) then --skip 4 and 5
            Qtemp <= std_logic_vector(to_unsigned(7,Qtemp'length));
         else
            Qtemp <= std_logic_vector(unsigned(Qtemp) + 1) ;    --inc
         end if;
      end if;
   end process;

   Q <= Qtemp ;
```

לאחר שהקומפילציה עברה בהצלחה בצע לקובץ זה סימולציה **FUNCTIONAL** בה תבדוק כמה מקרי טעינה כולל טעינת 5

חלון סימולציה



2 מונה BCD יורד עד 00

תכנן מונה ציקלי בעל 2 ספרות BCD שסופר בקוד בינארי כלפי **מטה** . ונטען סינכרונית. ניתן להרחיב את המונה מהתרגיל הקודם, כל ספרה היא כמובן ברוחב 4 BIT

הכניסה resetN היא כניסת איפוס א-סינכרונית שפעילה בנמוך. כניסת השעון נקראת clk.
הכניסה ena_cnt היא כניסת אפשור סינכרונית לספירה.

למונה יש 2 יציאות וקטוריות שנקראת countL, countH ויציאה בודדת TC(Terminal Count) שמפיקה '1' לוגי כאשר המונה נמצא במצב האחרון שלו (00) בירידה.

הכניסה LOAD היא כניסת טעינה סינכרונית שפעילה בנמוך. כאשר כניסה זו ב- 0, והשעון בעליה, המספר בכניסות DATA שמיוצגת ב BCD (כל ספרה בנפרד) נטען למונה , ניתן להניח שה DATA חוקית וכל NIBBLE (ארבעה BIT) אינו מכיל מספרים הגדולים מ- 9.

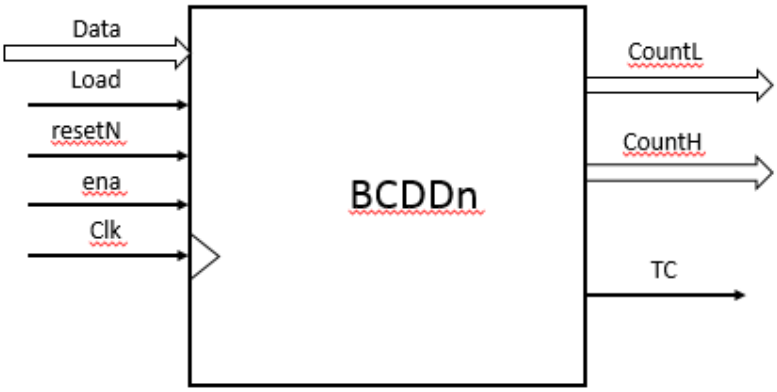
כאשר ספרת האחדות, CountL, מגיעה למצב 0 עשרוני ("0000" בינארי) היא מאותחלת ל- 9 ("1001"), וספרת העשרות, CountH, יורדת באחד באות השעון הבא. כאשר המונה מגיע ל- 00 (עשרוני), הוא :

- מוציא TC

יש להתנות יציאת TC גם ב ena =1.

- מאותחל ל- 99? באות השעון הבא (**נא למלא**).

הציור שלהלן מתאר את כניסות והיציאות של הרכיב



הדרכה:
למעט עבור ה TC עליך לרשום בארכיטקטורה תהליך סינכרוני בלבד . הקפד על כתיבת קוד חוקי שיעבור קומפילציה נקייה.
בנוסף לכך הקפד לבצע תכן סינכרוני שמתאים לכללי התכן הסינכרוני (אל תשתמש למשל ב - Ripple Clock או בתכן א-סינכרוני כל שהוא). קרא לישות ולקובץ שמכיל אותו BCDDN.VHD.

הוצא את ה- TC על ידי תהליך **אסינכרוני**
להזכירכם : אין צורך לאפס תהליך אסינכרוני (אין לו רכיבי זכרון שיש לאתחל).

2.1 קוד VHDL

```
entity bcddn is
    port ( Clk,ena,resetN,LoadN : in std_logic ;
           Data : in std_logic_vector(7 downto 0) ;
           CountL,CountH : out std_logic_vector(3 downto 0) ;
           TC : out std_logic ) ;
end bcddn ;

architecture arc_bcddn of bcddn is
 signal cntL,cntH : std_logic_vector(3 downto 0) ;
begin
   process(Clk,resetN)
    begin
      if rising_edge(CLK) then
         if (resetN = '0') then  --reset
            cntL <= (others => '0');
            cntH <= (others => '0');
         elsif (LoadN = '0') then   --LoadN
            cntL <= Data (3 downto 0) ;
            cntH <= Data (7 downto 4) ;
         else
            if ena = '1' then
               if cntL = 0 then
                  cntL <= "1001" ;
                  if cntH = 0 then
                     cntH <= "1001" ;
                  else
                     cntH <= cntH - 1 ;
                  end if;
               else
                  cntL <= cntL - 1 ;
               end if;
            end if;
         end if;
      end if;
   end process;

   CountL <= cntL ;
   CountH <= cntH ;

   process(cntH, cntL)
    begin
      if( cntL = 0  and cntH = 0 and ena = '1') then
         TC <= '1' ;
      else
         TC <= '0' ;
      end if;
   end process;
```

קמפל את הקובץ bcddn.vhd

העתק דוח קומפילציה

Flow Status	Successful - Sat Aug 11 22:01:56 2018
Quartus Prime Version	17.0.0 Build 595 04/25/2017 SJ Lite Edition
Revision Name	lab_vhdl2
Top-level Entity Name	bcddn
Family	Cyclone V
Device	5CGXFC7C7F23C8
Timing Models	Final
Logic utilization (in ALMs)	10 / 56,480 (< 1 %)
Total registers	8
Total pins	21 / 268 (8 %)
Total virtual pins	0
Total block memory bits	0 / 7,024,640 (0 %)
Total DSP Blocks	0 / 156 (0 %)
Total HSSI RX PCSs	0 / 6 (0 %)
Total HSSI PMA RX Deserializers	0 / 6 (0 %)
Total HSSI TX PCSs	0 / 6 (0 %)
Total HSSI PMA TX Serializers	0 / 6 (0 %)
Total PLLs	0 / 13 (0 %)
Total DLLs	0 / 4 (0 %)

2.2 סימולציה

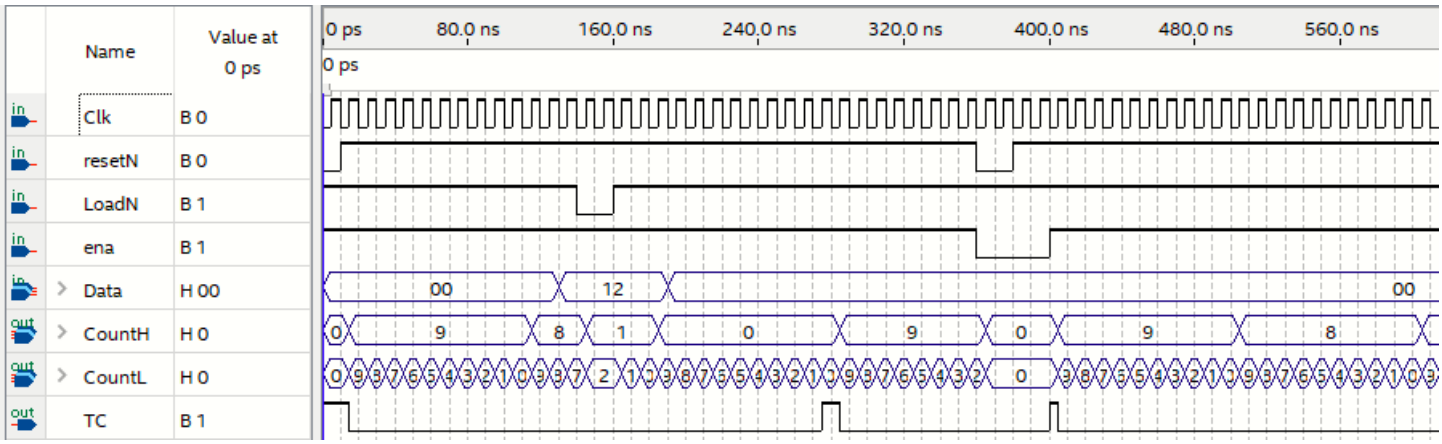
לאחר שהקומפילציה עברה בהצלחה בצע לקובץ זה סימולציה מלאה בה תבדוק את כל הכניסות והיציאות **המעניינות** ואת כל מקרי הקצה **המעניינים**

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

מצב	תוצאות צפויות
יציאה מRESET	כל היציאות מאותחלות
COUNT עם ENABLE	לא משתנה
COUNT בלי ENABLE	יורד ב1
CARRY	נדלק ב00 ותלוי בena
Load	עובד

הקפד להציג את המשתנים בפורמט רלוונטי BIT HEX DECIMAL

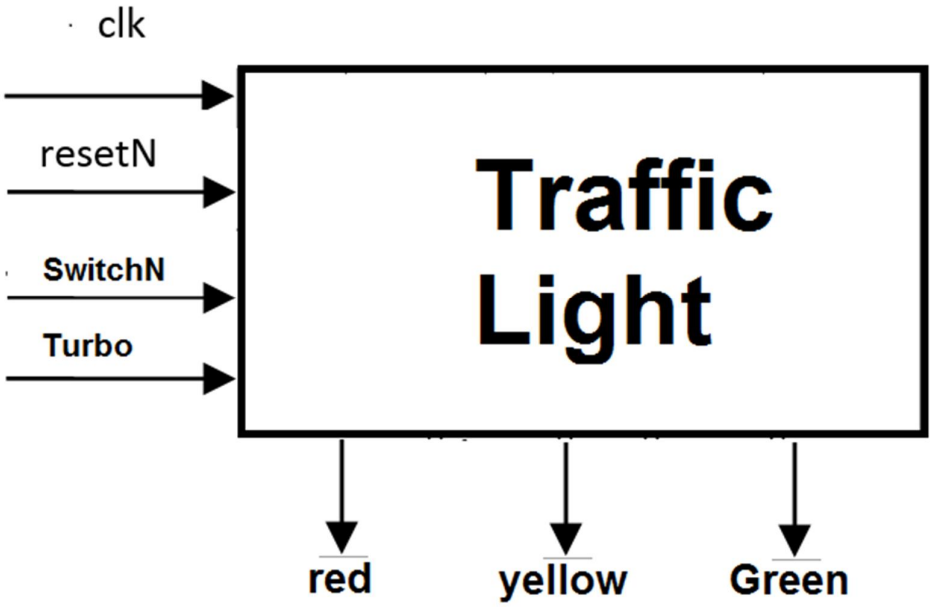
חלון סימולציה



3 מכונת מצבים – רמזור מבוקר

בפרוייקט הקודם פתח TOP חדש, וכתוב מכונת מצבים של רמזור, מצבי הרמזור הם אדום, אדום_צהוב, ירוק, צהוב .
הרמזור יהיה 3.7 שניות באדום .2.9 בירוק ו 1.5 שניות במצבי המעבר (צהוב, אדום-צהוב)
למכונה ינתן שעון מהיר 50MHz
במכונה יוגדר תהליך פנימי שייצר ENABLE של 10Hz ובו תשתמש מכונת המצבים לספירת הזמן
מכונת המצבים תמנה את הזמן על ידי מונה **פנימי**, ותתקדם כשצריך.
לחיצה על לחצן הכניסה SwitchN:

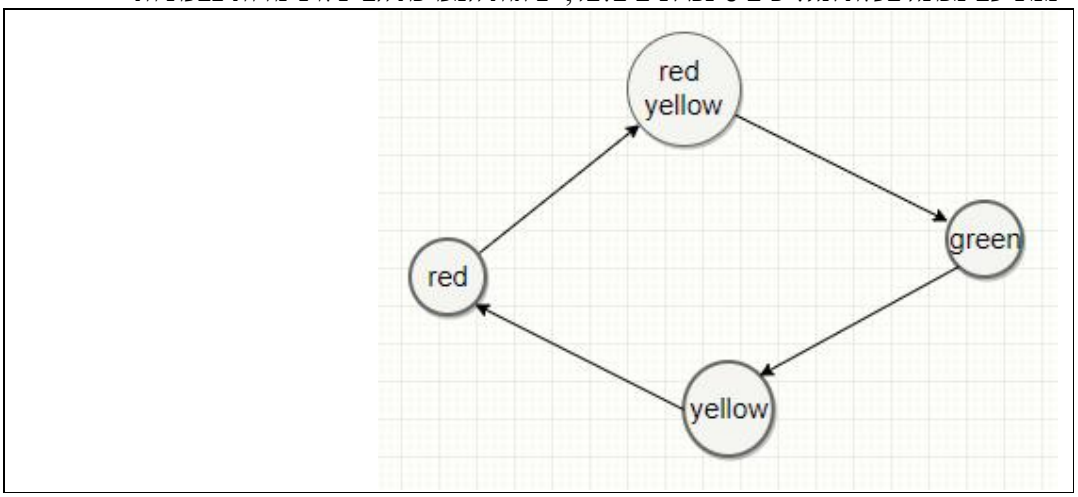
- אם הוא היה במצב אדום תעביר את הרמזור ישירות (בשעון הבא) מאדום לאדום צהוב
- אחרת לא תעשה כלום



הנחייה: יש לנהל מונה סינכרוני **בתוך** מכונת המצבים, המונה יספור מטה וכל פעם שהוא מגיע לאפס עוברים מצב במכונה ומאתחלים אותו לערך חדש.

- יש להשתמש בהגדרת **type** count_state **is(red,yellow...)**
- יש להשתמש ב CONSTANTS ולא במספרים בתוך הקוד
- יש לכתוב נכון את הקבועים למונה הפנימי שיתאימו גם לסימולציה וגם לעבודה בשטח בשינוי יחיד
- כל המכונה תעבוד ב 50MHz
- לרמזור כניסת TURBO באמצעותה אפשר לזרז את פעולת הרמזור פי 10.
  הערה – הכניסה תחובר למחולל ה- 10Hz ותהפוך אותו למונה 100Hz
- כניסת הלחצן בלוגיקה שלילית, 0 כשהוא לחוץ

ממש עם מכונה בעלת **תהליכים סינכרונים בלבד**, יציאות המערכת הם שלוש נורות צבעוניות



תאור המצבים – השלם את המצבים החסרים

שם המצב	פעילות עיקרית	לאיזה מצב עוברים מהמצב הנוכחי ובאילו תנאים – למלא את התאים הריקים
RED	מאתחלים את המונה	לאחר זמן- עוברים לאדום צהוב
RED_YELLOW	מאתחלים את המונה	לאחר זמן – עוברים לירוק
GREEN	מאתחלים את המונה	לאחר זמן – עוברים לצהוב
ORANGE	מאתחלים את המונה	לאחר זמן – עוברים לאדום

3.1 קוד VHDL

```
--Traffic_Light

library ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_unsigned.all ;
use ieee.numeric_std.all;
use ieee.std_logic_arith.all;


entity Traffic_Light is
    port ( clk,resetN,SwitchN,TurboN : in std_logic ;
           Red,Yellow,Green : out std_logic) ;
end Traffic_Light ;

architecture arc_Traffic_Light of Traffic_Light is
 type count_state is(redState, redYellowState, greenState, yellowState);
 signal State : count_state;
 signal MyClk : std_logic;
```

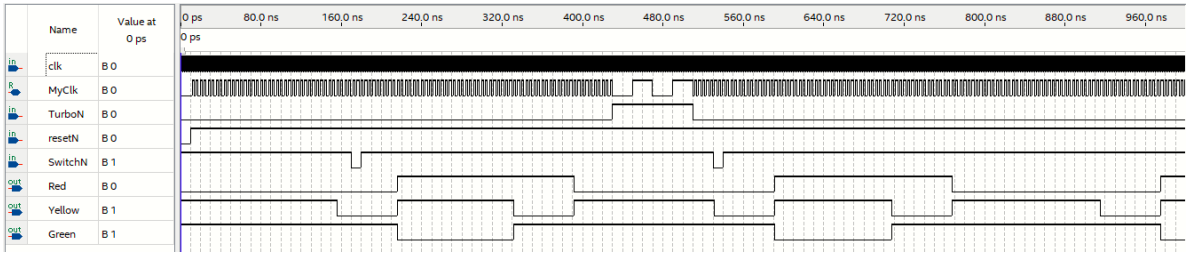
```
begin
--internal clock
   process(Clk, resetN)
      constant TurboSize : integer := 10 ;
      --constant frequencyDivider : integer := 2500000 ;
      constant frequencyDivider : integer := 20 ;
      variable cnt : integer ;
    begin
      if rising_edge(CLK) then
         if resetN = '0' then
            MyClk <= '0' ;
            cnt := 0;
         else
            if TurboN = '0' then
               cnt := cnt + TurboSize;
            else
               cnt := cnt + 1 ;
            end if;
            if cnt >= frequencyDivider then
               cnt := cnt - frequencyDivider;
               MyClk <= not MyClk;
            end if;
         end if;
      end if;
   end process;

   process(MyClk, resetN)
      constant redTime : integer := 37 ;
      constant greenTime : integer := 29 ;
      constant middleTime : integer := 15 ;
      variable count : integer ;
    begin
      if resetN = '0' then
         count := 0;
         State <= redState;
      elsif rising_edge(MyClk) then
         count := count + 1;
         case State is
            when redState =>
               if (count >= redTime) OR (SwitchN = '0') then
                  count := 0;
                  State <= redYellowState;
               end if;
            when redYellowState =>
               if count >= middleTime then
                  count := 0;
                  State <= greenState;
               end if;
            when greenState =>
               if count >= greenTime then
                  count := 0;
                  State <= yellowState;
               end if;
            when yellowState =>
               if count >= middleTime then
                  count := 0;
                  State <= redState;
               end if;
         end case;
      end if;
   end process;

   Red <= '0' when State = redState  or State = redYellowState else '1';
   Yellow <= '0' when State = yellowState  or State = redYellowState else '1';
   Green <= '0' when State = greenState else '1';

end arc_Traffic_Light;
```

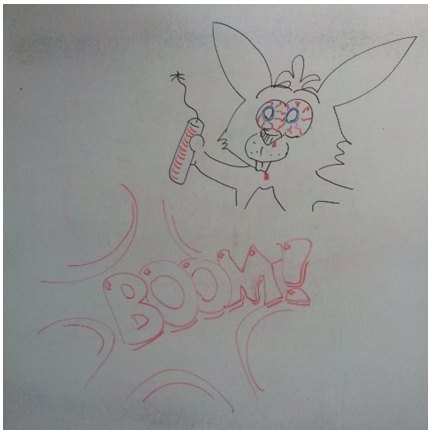
הגדר מה תרצה לבדוק בסימולציה – איזה מצבים מעניינים (המשך למלא את הטבלה)

מצב	תוצאות צפויות
יציאה מRESET	כל היציאות מאותחלות
Switch באדום	עובר לכתום אדום
Switch בצבע אחר	לא עושה כלום
כיבוי טורבו	מאט את התדר פי עשר
לא עושים כלום	כל המצבים עובדים ועוברים לבד

3.2 סימולציה

כתוב והרץ סימולציה של המעגל, שים לב שהשעון צריך להיות מהיר מ- 1Hz (לפחות 5Hz)
חלון(נות) סימולציה





4 פצצה – פרוייקטון

<u>יש לבנות מנגנון לפצצה</u> על פי ההגדרות להלן:

<u>כניסות הפצצה</u>:

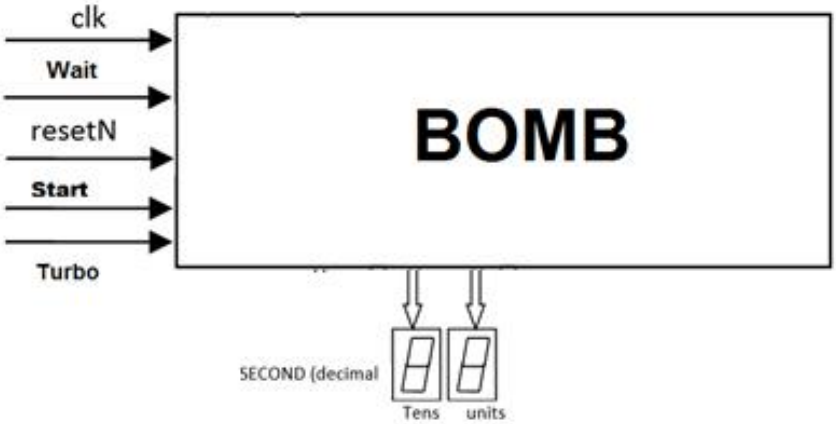
- כניסת clk שבכרטיס DE10 בתדר 50MHz.
- לחצן START יטען לשעון את זמן הפיצוץ שנקבע (קבוע)
- לחצן WAIT כל לחיצה תעצור את המניה כל זמן הלחיצה.
  שימו לב לחצן לחוץ =0 משוחרר = 1
- לחצן resetN יאתחל את התצוגה ל **00** . עדיין לא יקרה כלום עד לSTART הראשון
- טורבו- מאיץ את פעולת השעון פי 10 לצרכי (DEBUG)

<u>יציאות הפצצה</u>:

- תצוגות 7Segments יציגו את הזמן שנותר עד לפיצוץ: בשניות.(עשרות ויחידות)
  כשנגמר הזמן, התצוגות (7 segments) יהבהבו. 00, Blink

<u>פעולת הפצצה</u>

1. ב RESET הפצצה תאפס את התצוגה והמונים ותמתין ל START .
2. בלחיצה על - START הפצצה רק תטען ל זמן הפצצה שהוא -**<u>17 שניות</u>**
3. בעזיבת ה- START הפצצה תמנה אחורה עד לאפס (פיצוץ)
4. האותות RESET, START פעילים בנמוך (ACTIVE LOW).
5. כשעון הפצצה יגיע לאפס יהיה הבהוב בכל התצוגה בתדר של כ- 1/2 Hz. עד ל - RESET חדש
6. בכל לחיצה על לחצן WAIT, המניה תעצור כל זמן הלחיצה. בשחרור הלחצן הפצצה תמשיך לעבוד
7. הלחיצה יכולה להיות קצרה ולכן יש לדגום אותה במכונת מצבים שרצה ב 50 MHz
8. יש לממש את הפצצה באמצעות ארבעה תהליכים:
   a. מכונת המצבים (תמומש בעבודת הכנה זו)
   b. מונה שתי ספרות בקוד VHDL יחיד (מסעיף קודם)
   c. מחלק תדר 1Hz (ממעבדה קודמת)
   d. תצוגה (HEXSS ממעבדה קודמת)



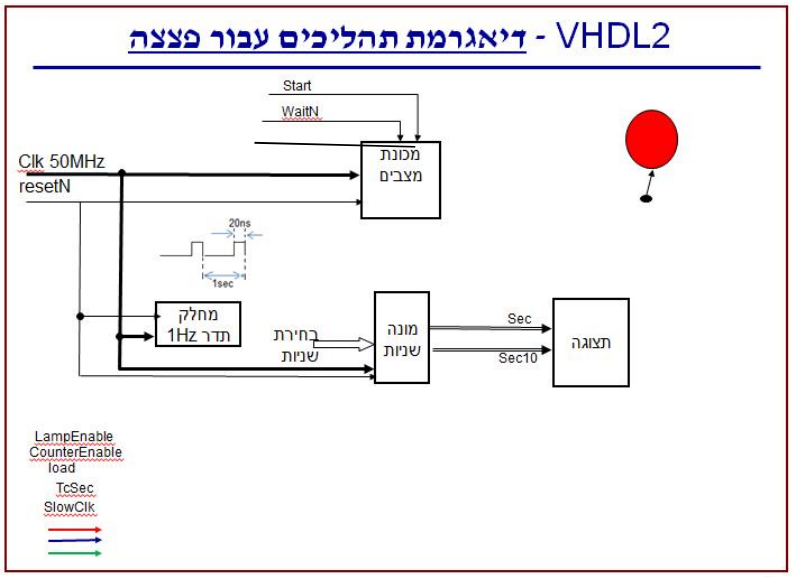
יש להקפיד להוסיף N כסיומת לסיגנלים שהם active Low (לחצנים)

<u>הערה:</u> לצורך ספירת הזמן ניתן להשתמש ביחידה שהכנתם בתרגיל קודם ולשנותה ל-BCDDN_XX. על ידי תיקון הקוד בתוך היחידה

4.1 ארכיטקטורה

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

הסכימה שלהלן מתארת את המודולים המרכיבים את המערכת וחלק מהקשרים ביניהם.



להגדרת המכלולים מלאו את הטבלאות הבאות :

<u>רשום ופרט כל אחד מהתהליכים (מודולים) שתרצה למחזר מפרויקטים קודמים</u>

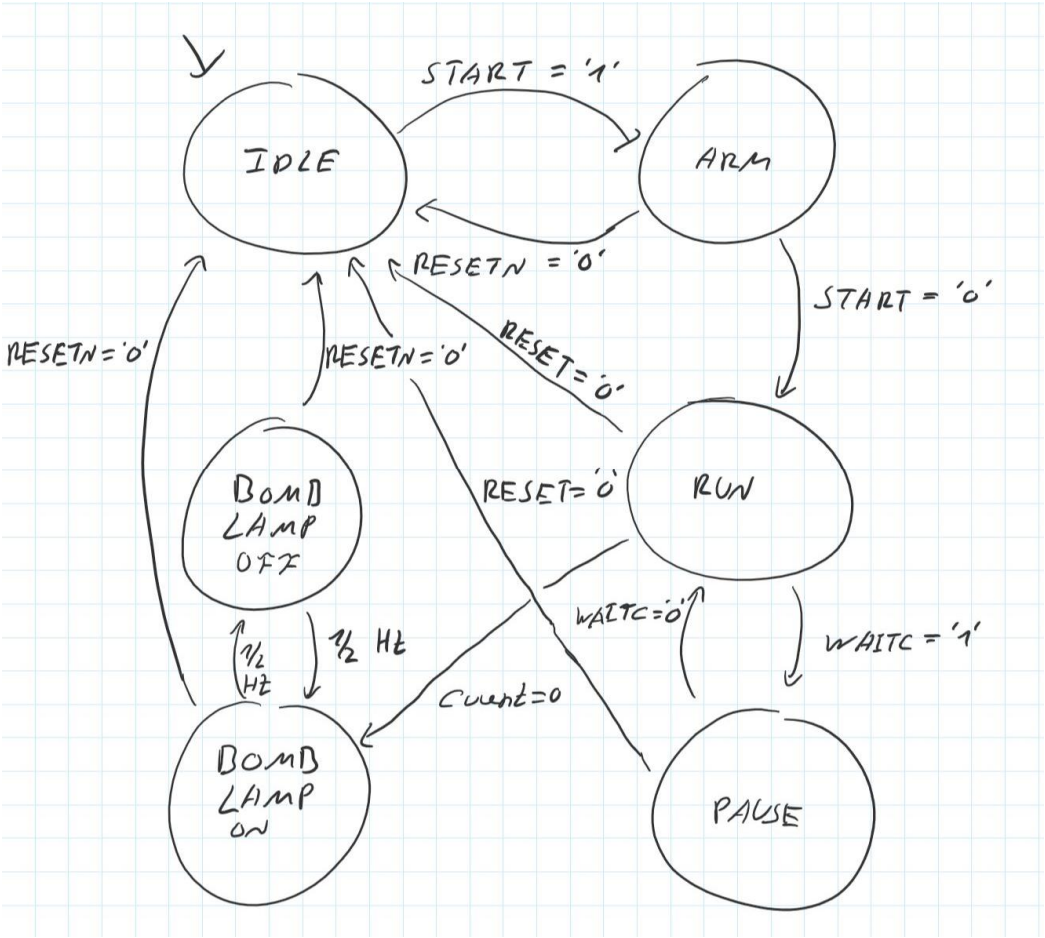
שם	הסבר פעולה	שינויים דרושים
מונה שניות	מונה אחורה שניות	מונה שניות קדימה
HEX SS	תצוגה ספרתית	אין שינויים
מחלק תדר	ממיר אות שעון מ50MHZ ל1HZ	אין שינויים

<u>רשום ופרט כל אחד מהתהליכים שתרצה לממש מ- Scratch</u>

שם	הסבר פעולה	כניסות עיקריות	יציאות עיקריות
מכונת מצבים	ניהול התהליך	Start – כניסה המעידה על התחלת הפעולה WAITC – כניסה המעידה על כך שהמכונה תפסיק להתקדם	

		resetN – כניסה המעידה על איפוס מכונת המצבים Clk – שעון המערכת	

<u>הגדר את מכונת המצבים ושרטט את דיאגרמת מצבים</u> - העזר בשבלונה המוכנה וכתוב את פירוט המעברים .



<u>הוסף בטבלה שלהלן את כל מצבי מכונת המצבים חסרים</u> , כתוב מה עושים בהם 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
RUN	סופר לאחור. בזמן 0 שניות מפוצץ את הפצצה.	Reset נלחץ עוברים ל-IDLE WAIT עוברים ל-WAIT

ARM	טוען את הזמן 17 שניות למונה	Start משתחרר עוברים ל-RUN. Reset נלחץ עוברים ל-IDLE
WAIT	ממתינים עד כניסת ה-wait חוזרת ל-0	עבור שחרור ה-wait חוזרים ל-RUN Reset נלחץ עוברים ל-IDLE
BOMB LAMP ON	מדליק את הנורות	Reset נלחץ עוברים ל-IDLE אחרי 1/2HZ עוברים ל BOMB LAMP OFF
BOMB LAMP OFF	מכבה את הנורות	Reset נלחץ עוברים ל-IDLE אחרי 1/2HZ עוברים ל BOMB LAMP ON

4.2 מימוש מכונת המצבים

השתמש בפרויקט של המונה – פתח קובץ TOP חדש בשם BOMB_SM

כתוב וסמלט את מכונת המצבים - ראה שלד במודל (BombSkeleton.vhd)

שים לב שהמצבים בשלד אינם זהים לחלוטין למצבים שעליך לממש

שם המכלול: מכונת מצבים:

תאור פעולה מקוצר :

מודול אחראי על מעבר בין המצבים ומייצג קומפוננט של ניהול עבור המערכת. המצבים הם אלו המעידים מתי לבצע את הפעולות השונות כגון טעינת הזמן, תחילת ספירת הזמן וכו.

פירוט כניסות:

resetN – מאפס את מכונת המצבים ומחזיר למצב IDLE
START – בלחיצה מהווה אינדיקטור לטעינת הזמן למונה שניות. בשחרור מהווה אינדיקטור לתחילת הספירה לאחור
WaitX – אינדיקטור להכניס את מכונת המצבים למצב השהייה
OneHzPulse – שעון שעולה כל שנייה
TC – מעיד מתי המונה שניות סיים את פעולתו

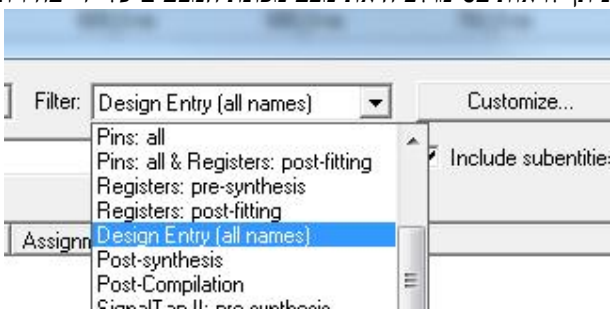
פירוט יציאות:

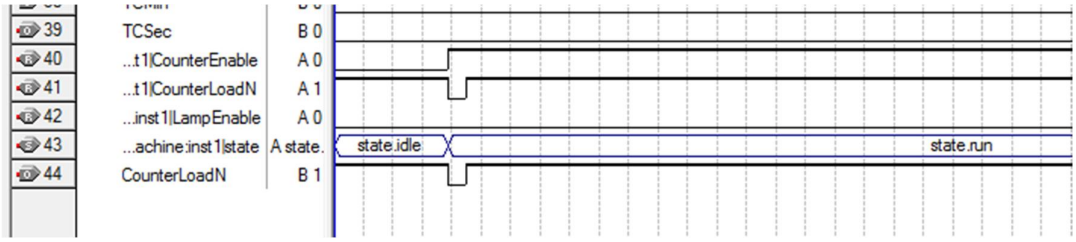
CounterEnable – אות המעיד על תחילת הספירה לאחור
CounterLoadN – מעיד על טעינת ערך חדש למונה שניות
LampEnable – אות המעיד על הדלקת הלדים

```
library ieee ;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;

entity BOMB_SM is
   port (   clk:  in    std_logic;
      resetN:     in    std_logic;
      START:      in    std_logic;
      WaitX:   in std_logic; --X addeed as "wait" might be a reserved word
      OneHzPulse: in std_logic;  -- a narrow pulse every second
      Tc:      in std_logic;   --counter terminal count

      CounterEnable:  out  std_logic;  --enable('1')/disable('0')
      CounterLoadN:  out   std_logic;  --/load ('0')
      LampEnable:    out   std_logic ); -- on('1') off('0')
end BOMB_SM;

architecture arc_BOMB_SM of BOMB_SM is
  type state_type is (idle, arm, run, pause, lamp_on, lamp_off); --enumerated type for state machine
  signal state : state_type;
begin

    process (clk, resetN)
    variable sec_trigger: integer ;
    variable sec_indicator: integer ;
    begin
        if resetN = '0' then
            state <= idle;
            CounterLoadN <= '1' ;
            CounterEnable <= '0';
            LampEnable <= '0';
        elsif (rising_edge(clk)) then
      -- preset all outputs ,override in the state machine if needed
            CounterLoadN <= '1' ;
            CounterEnable <= '0';
            LampEnable <= '0';
         --  fill other commands ;

         if OneHzPulse = '1' AND sec_indicator = 0 then
            sec_indicator := 1;
            sec_trigger := 1;
         elsif OneHzPulse = '0' then
            sec_indicator := 0;
         end if;

            -- Determine the next state synchronously, based on
            -- the current state and the input
         case state is
                when idle=>
                    if start = '1' then
                        state <= arm;
                        CounterLoadN <= '0' ;
```

```
            -- Determine the next state synchronously, based on
            -- the current state and the input
        case state is
                when idle=>
                    if start = '1' then
                        state <= arm;
                        CounterLoadN <= '0' ;
                    end if;
                when arm =>
                    if start = '0' then
                        state <= run;
                        CounterEnable <= '1';
                    end if;
                when run=>
                    if WaitX = '1' then
                        state <= pause;
                        CounterEnable <= '0';
                    elsif Tc = '1' then
                        state <= lamp_on;
                        sec_trigger := 0;
                    else
                        CounterEnable <= '1';
                    end if;
                when pause => --..... to operate 3sec timer
                    if WaitX = '0' then
                        state <= run;
                        CounterEnable <= '1';
                    end if;
                when lamp_on => --.... to operate 1sec timer
                    if sec_trigger = 1 then
                        LampEnable <= '0';
                        state <= lamp_off;
                    else
                        LampEnable <= '1';
                    end if;

                    sec_trigger := 0;
                when lamp_off => --..... to operate 1sec timer
                    if sec_trigger = 1 then
                        LampEnable <= '1';
                        state <= lamp_on;
                    else
                        LampEnable <= '0';
                    end if;

                    sec_trigger := 0;
         end case;
        end if;
    end process;
end arc_BOMB_SM ;
```

ניתן לראות בסימולציה את מצב מכונת המצבים על ידי בחירת: all_names





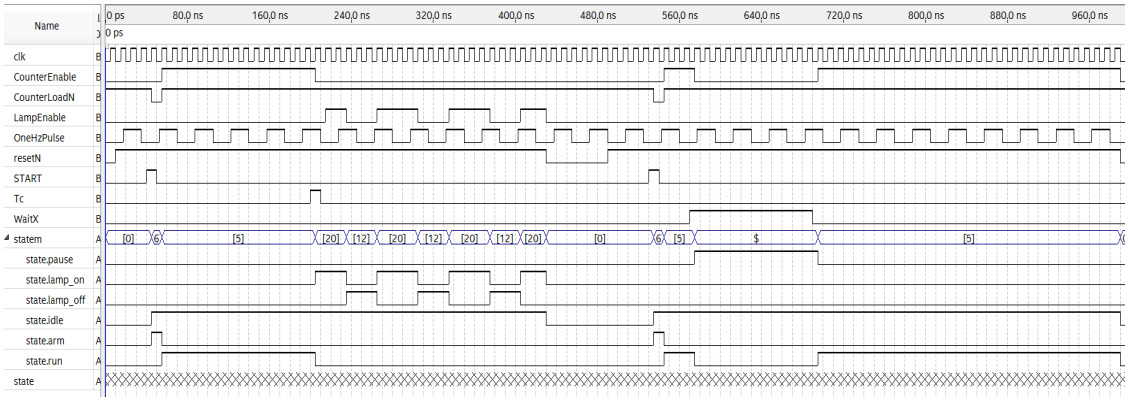
שימו לב שיש לבקש להציג בסימולטור את כל המצבים כל אחד בנפרד, מצב Idle מופיע הפוך (בגלל BUG ופעיל בנמוך. כל השאר פעילים בגבוה)

4.3 סימולציה למכונת המצבים

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

מצב	תוצאות צפויות
יציאה מRESET	כל היציאות מאותחלות
לחיצת START	מעבר למצב ARM וטעינת המונה שניות
עזיבת START	מעבר למצב RUN והתחלת פעולת מונה שניות
לחיצה על waitx	מעבר למצב pause
מצב led_on	עוברים אליו לאחר TC, וממנו עוברים ל-led_off לאחר 1/2HZ
מצב led_off	ממנו עוברים ל-led_on לאחר 1/2HZ

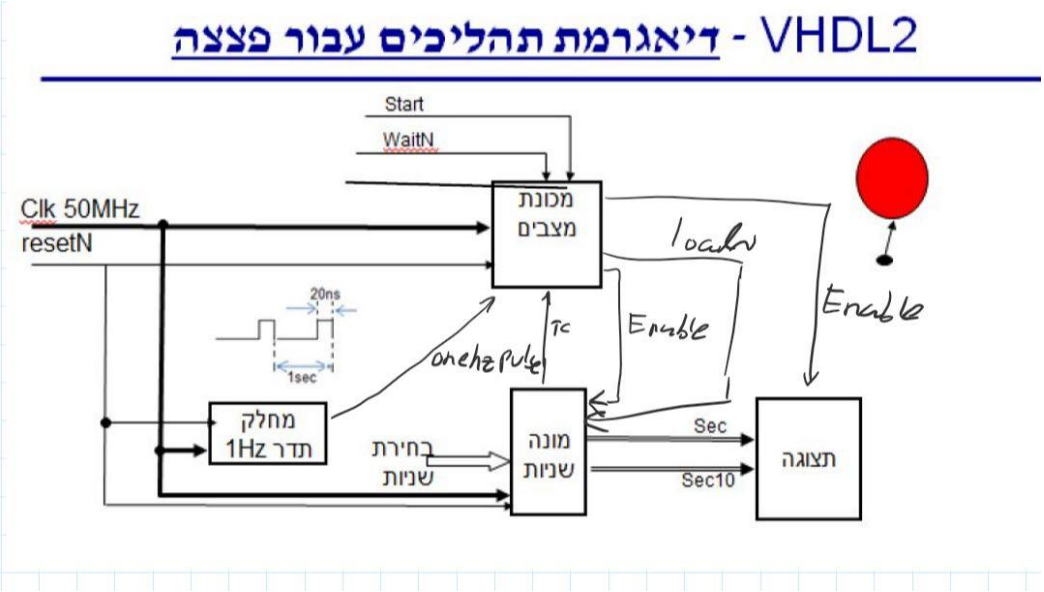
שיב לב לבחור RADIX מתאים לכל תצוגה למש ל HEX ולא בינארי



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

4.4 השלמת דיאגרמת התהליכים.

לאחר שממשת את מכונת המצבים, השלם את דיאגרמת התהליכים (סכימת המלבנים) ע"י הוספת כל הקשרים בין התהליכים.



תזכורת:

לבצע פעולת ARCHIVE במודל על הפרויקט שכתבתם ולהעלות במודל ל

QAR דחוס של VHDL2 - דוח הכנה

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

מלא את הטופס