הטכניון – מכון טכנולוגי לישראל
הפקולטה להנדסת חשמל

מעבדה 1א' 044157

ניפוי תקלות בחומרה (DEBUG)
System_Verilog

דו"ח מכין - שאלות ותרגילי הכנה
עם אמולטור למקלדת

HS DSL הניסוי פותח בחסות המעבדה למערכות ספרתיות מהירות

גרסה 3.12 - קיץ 2020

עדכנו : דודי בר-און, ארמנד שוקרון, ליאת שורץ
על פי החוברת המקורית של עמוס זסלבסקי

תאריך הגשת דו"ח ההכנה	18/08/2020
שם המדריך	אלון מזרחי

סטודנט	שם פרטי	שם משפחה
1	ליאור	דביר
2	נועם	אילתה

הנחיה: קובץ זה הוא גם תבנית לדו"ח המכין, יש לשמור ב- PDF ולהגיש במודל.

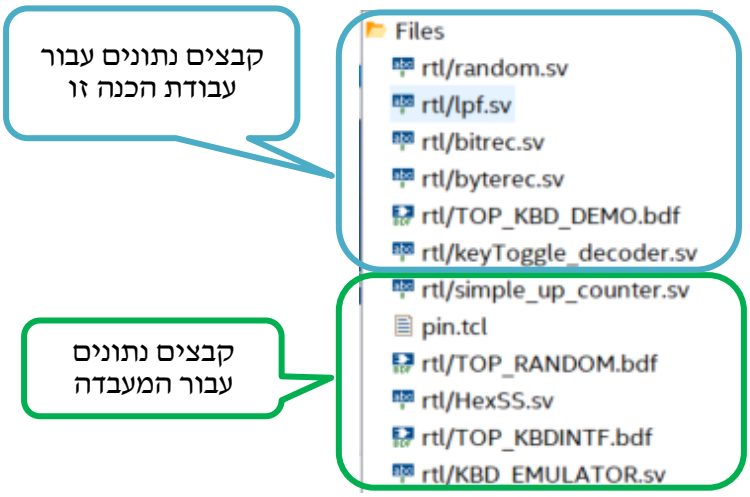
תוכן עניינים של דו"ח מכין DEBUG

1 פתיחת הקבצים לעבודה 2
2 מכונת RANDOM 3
3 ממשק למקלדת 4
3.1 תכן יחידת ה - **BITREC** 4
3.2 סימולציה 9
4 חישוב עומק הזכרון עבור הנתח הלוגי 10
5 מטלת תכן עם מקלדת 10
6 גיבוי העבודה 12

1 פתיחת הקבצים לעבודה

הורד מהמודל קובץ ארכיב של המעבדה ופתח אותו לפרויקט בדיסק שלך.

ודא תכולת קבצים כזו:

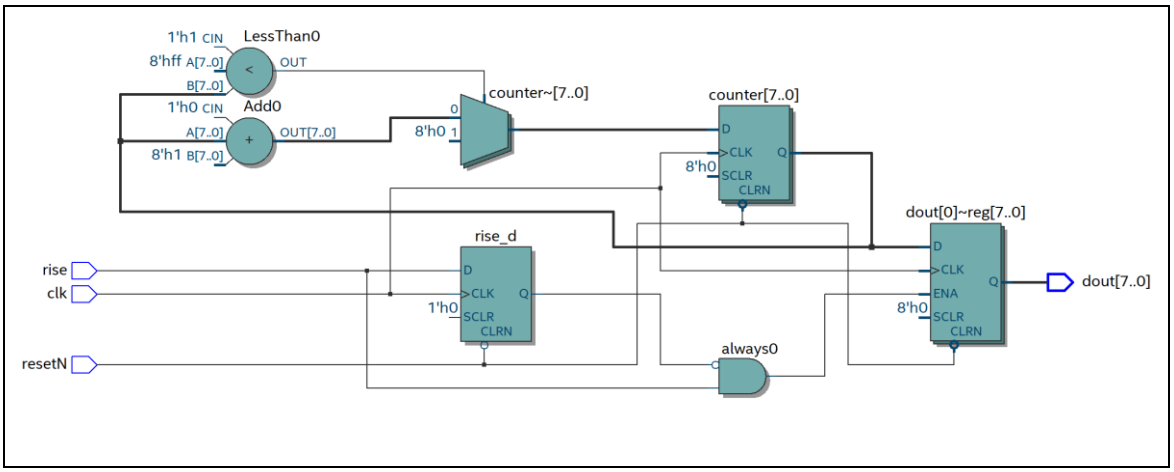


הקבצים המסומנים בכחול הם עבור עבודת הכנה זו.
הקבצים המסומנים בירוק הם עבור המעבדה. הם נתונים לך עכשיו כחלק מהפרויקט שתתחיל אותו כעת בעבודת ההכנה ותמשיך אותו במעבדה.

הערה חשובה: בסיום העבודה יש לשמור את הפרויקט כארכיב QAR במודל וכן להביאו למעבדה יחד עם כל הרכיבים אותם אתה כותב במסגרת עבודת ההכנה. זה יהווה בסיס לעבודתך במעבדה.

2 מכונת RANDOM

נתון לך קובץ random.sv, הממומש מערכת שמייצרת מספרים בצורה אקראית.
פתח אותו ונסה להבין את פעולתו.
הפוך אותו ל-TOP והרץ אנליזה שלו.
הצג את היצוג הגרפי שלו כ- RTL VIEW (Tools -> Netlist Viewers -> RTL Viewer) והוסף אותו לדו"ח.



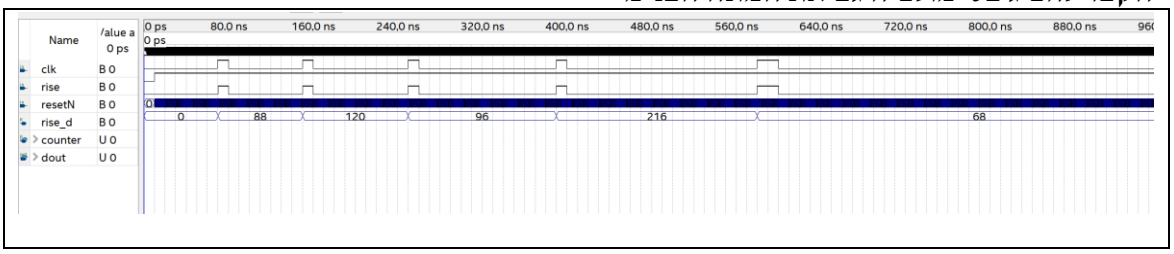
הסבר מדוע היציאה dout[7..0] היא מספר אקראי?

תשובה: הערך יוחזר כתלות בזמן שבו rise עולה מאפס לאחד, כל עוד עליה זו היא אקראית אז נקבל מספר אקראי

כיצד ניתן לשנות את המכונה כך שתגריל מספרים שהם כפולות של 2 בלבד?

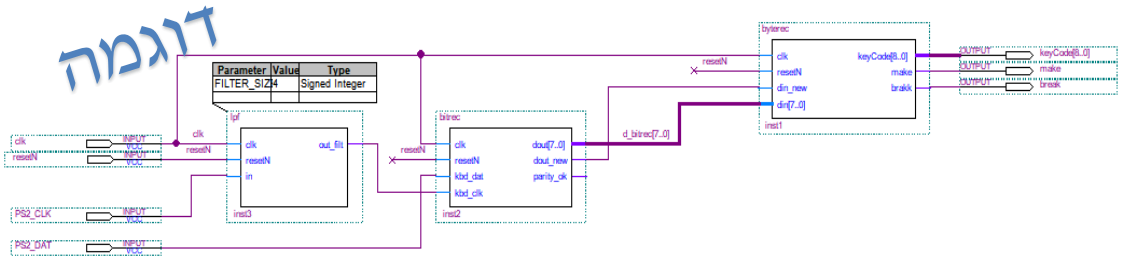
תשובה: נגדיל את count בקפיצות של 2

לבדיקת הפעולה של המודול הרץ סימולציה והוסף את תוצאות הסימולציה לדו"ח. הראה תוצאה אקראית בשני מקרים לפחות.
הקפד להציג בסימולציה גם את המונה הפנימי



3 ממשק למקלדת

כפי שהוסבר בחומר הרקע לניסוי זה, התכן הסינכרוני הבא נבחר למימוש **ממשק חומרה למקלדת**.

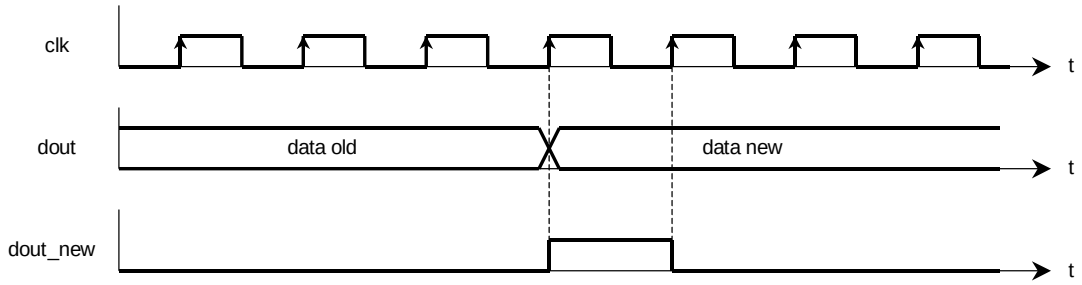


יחידות אלה כתובות בשפת SYS-VERILOG ותשמשנה לבנית הממשק למקלדת במעבדה.
להלן הקבצים הנתונים לך המרכיבים את הממשק למקלדת :

- יחידת מסנן מעביר נמוכים : lpf.sv
- יחידת המקלט ברמת ה – Bit : bitrec.sv - נתון שלך שלה
- יחידת המקלט ברמת ה – Byte : byterec.sv

3.1 תכן יחידת ה - BITREC

רקע למטלה : כמו שהוסבר בחומר הרקע תפקידה של היחידה שמטפלת בתשדורת הטורית, ה-BITREC, הוא להפיק מהמידע הטורי שמגיע לכניסות kbd_clk ו kbd_dat, מידע מקבילי ביציאה dout, יחד עם יציאת חיווי שפעילה למשך מחזור שעון אחד ושנקראת dout_new. דיאגרמת הזמנים הבאה מתארת אותות אלו אחד ביחס לשני וביחס לאות השעון :

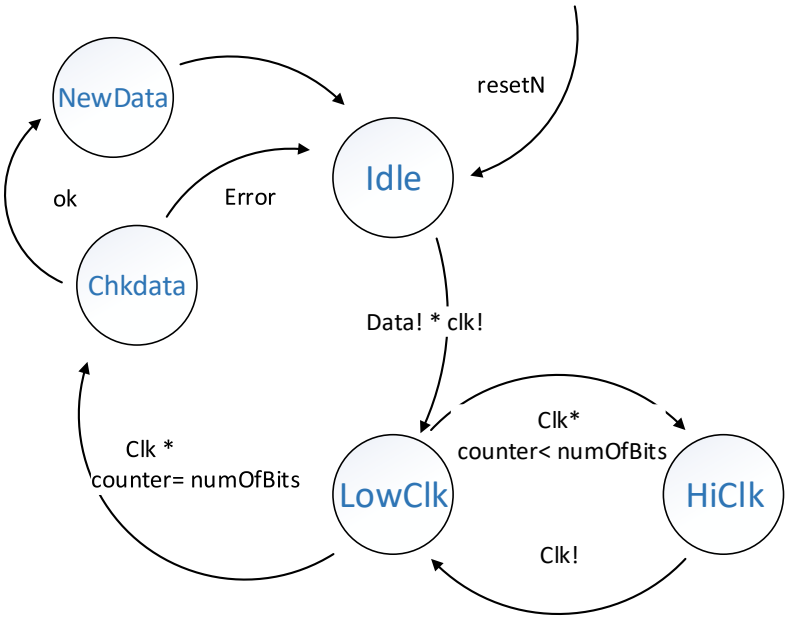


נתון לך הקובץ bitrec.sv שהוא שלד המכיל את כל החלקים הדרושים כפי שהוסבר בחומר הרקע פרט למכונת המצבים.

שים לב! השתמש אך ורק בקובץ הנתון לך כעת במודל ולא בגרסאות אחרות מסמסטרים קודמים!

הוסף לקובץ זה את הקוד של מכונת המצבים, כפי שתתואר להלן, במקומות בקובץ שבהם כתובה ההערה `fill your code please` `&&&&&&&&&&`

מכונת מצבים (מסוג Moore) משמשת כבקר של היחידה. דיאגרמת המצבים הבאה מתארת את התנהגותה.



בדיאגרמה הנ״ל השתמשנו **בקיצורים** הבאים :

- clk מציין את האות Kbd_CLK (ממופה לפין PS2_CLK) בגבוה, ו- clk! בנמוך
- Data מציין את האות Kbd_DAT (ממופה לפין PS2_DAT) בגבוה, ו- Data! בנמוך
- ok מציין את הסיגנל parity_ok במצב true
- Error מציין את הסיגנל parity_ok במצב false
- counter מונה את מספר הביטים של קוד המקש שמגיעים בקו הסריאלי

הדרכה ודרישות:

כתוב קוד המתאר את מכונת המצבים באמצעות תהליך סינכרוני בלבד. פתח את הקובץ **bitrec.sv** מתוך הפרויקט הקיים KBD והגדר אותו כהיררכיה עליונה. הוסף לקובץ את הקוד שלך בלבד בהתאם להנחיות להלן.

שימו לב: במכונה הוכנסה תקלה במכוון
אין צורך לשנות חלקים אחרים משלד הקוד הנתון ב- bitrec.sv!
אם מצאתם את התקלה- אנא אל תדווחו עליה בפורום שאלות ותשובות
וגם אל תספרו לחבריכם, השאירו להם את חווית הגילוי העצמי !

חשב מהו NUM_OF_BITS.
תשובה : 10

בטבלה הבאה מפורטים המצבים שבמכונה והפעולות לביצוע בכל מצב.
מלא את העמודה האחרונה בטבלה לפי הדוגמה שבשורה הראשונה:

שם המצב	פעילות עיקרית	לאיזה מצב עוברים מהמצב הנוכחי ובאילו תנאים – למלא את התאים הריקים
Idle	מאפסים את המונה count. ממתינים לתו חדש : אם יש ירידה באות השעון Kbd_CLK וגם ירידה באות הנתונים Kbd_DAT אז עוברים למצב הבא.	**עוברים ל-** LowClk **עם** ירידה בשעון Kbd_CLK וגם ירידה ב- Kbd_DAT (סימן שמתחיל להגיע תו חדש)
LowClk	זה מצב קבלת הביט. במצ זה ממתינים לאות שעון גבוה כי זה אומר שהביט הבא מגיע.	**עוברים ל-** HiClk

	אם Kbd_CLK גבוה: - משרשרים למקום האחרון ברגיסטר ההזזה shift_reg את הסיבית החדשה שהגיעה מה- Kbd_DAT. `Next_Shift_Reg = {kbd_dat,shift_reg [9:1]};` - בודקים אם מונה הביטים cntr קטן ממספר הביטים. אם כן (טרם הגיעו כל הביטים של המילה) - מקדמים את המונה cntr ב-1 - עוברים למצב HiClk אם לא (המילה השלמה התקבלה) - עוברים למצב בדיקת הנתונים	**עם** עליה בשעון Kbd_CLK וגם אם count קטן מ num_of_bits
HiClk	במצב זה השעון גבוה וממתינים לביט הבא. אם יש ירידה ב- Kbd_CLK זה אומר שמגיע ביט ויש לעבור למצב הבא, קבלת הביט.	**עוברים ל-**LowCLK **עם** ירידת שעון.
ChkData	מצב בו בודקים את נכונות הנתונים. בודקים האם הזוגיות נכונה (1 לוגי). בהתאם לתוצאת הבדיקה עוברים למצב הבא. אם בדיקת הזוגיות (ה- parity_ok) טובה אז מעדכנים את המוצא בתכולת הרגיסטר `Next_Dout = shift_reg[7:0];` ועוברים למצב שבו מודיעים על מילה חדשה. אם הבדיקה לא טובה, עוברים למצב ההתחלתי של המתנה לתו חדש.	**עוברים ל-**NewData **עם** parity_ok הוא 1. **עוברים ל-**Idle **עם** parity_ok הוא 0.
NewData	במצב זה תמיד מודיעים על התו החדש `dout_new = 1'b1 ;` ועוברים מצב	**עוברים ל-**Idle **עם** עליית שעון.

<u>בצע קומפילציה.</u>

```
module bitrec
  (
      input logic clk,
      input logic resetN,
      input logic kbd_dat,
      input logic kbd_clk,
      output  logic [7:0]      dout,
      output  logic dout_new,
      output  logic parity_ok
  ) ;


 enum  logic [2:0] {IDLE_ST, // initial state
                    LOW_CLK_ST, // after clock low
                    HI_CLK_ST, // after clock hi
                    CHK_DATA_ST, // after all bits recieved
                    NEW_DATA_ST // if valid parity announce
new data
                                  }  nxt_st, cur_st /* synthesis
keep = 1 */;
```

```
  logic [3:0] cntr, nextCntr /* synthesis keep = 1 */ ;
  logic [9:0] shift_reg, Next_Shift_Reg  /* synthesis keep = 1 */ ;
  logic [7:0] Next_Dout  /* synthesis keep = 1 */ ;

  localparam NUM_OF_BITS = 4'ha ;  // &&&&&&&&&&&&&&      fill please

      always_ff @(posedge clk or negedge resetN)
      begin: fsm_sync_proc
            if (resetN == 1'b0) begin
                  cur_st <= IDLE_ST ;
                  cntr <= 4'h0  ;
                  shift_reg <= 10'h000 ;
                  dout <= 8'h00 ;
            end
            else begin
                  cur_st <= nxt_st;
                  cntr <= nextCntr;
                  shift_reg <= Next_Shift_Reg;
                  dout <= Next_Dout;

            end
      end // end fsm_sync_proc

always_comb
begin
      // default values
            nxt_st = cur_st ;
            nextCntr = cntr  ;
            Next_Shift_Reg = shift_reg  ;
            Next_Dout = dout ;
            dout_new = 1'b0 ;

      case(cur_st)
                  IDLE_ST: begin
                        nextCntr = 4'h0  ;
                        if( (kbd_clk == 1'b0) && (kbd_dat == 1'b0) )
                              nxt_st = LOW_CLK_ST;
                  end

                  LOW_CLK_ST: begin
                        if (kbd_clk == 1'b1) begin
                              Next_Shift_Reg = {kbd_dat,shift_reg
[9:1]};
                              nextCntr = cntr + 1;
                              if (cntr < NUM_OF_BITS)
                                    nxt_st = HI_CLK_ST;
                              else if (cntr == NUM_OF_BITS)
                                    nxt_st = CHK_DATA_ST;
                        end
                  end

                  HI_CLK_ST: begin
                        if (!kbd_clk)
                              nxt_st = LOW_CLK_ST;
                  end

                  CHK_DATA_ST: begin
                        if (parity_ok) begin
                              nxt_st = NEW_DATA_ST;
                              Next_Dout = shift_reg[7:0];
```

```
                        end else
                                nxt_st = IDLE_ST;
                    end

                    NEW_DATA_ST: begin
                        nxt_st = IDLE_ST;
                        dout_new = 1'b1;
                    end

            endcase

end
// parity Calc
assign parity_ok = shift_reg[8] ^ shift_reg[7] ^ shift_reg[6] ^
shift_reg[5]
        ^ shift_reg[3] ^ shift_reg[2] ^ shift_reg[4] ^ shift_reg[1] ^
shift_reg[0];

endmodule
```

צרף לכאן צילום מסך של תוצאות קומפילציה מוצלחת של המעגל.

Flow Summary

<<Filter>>

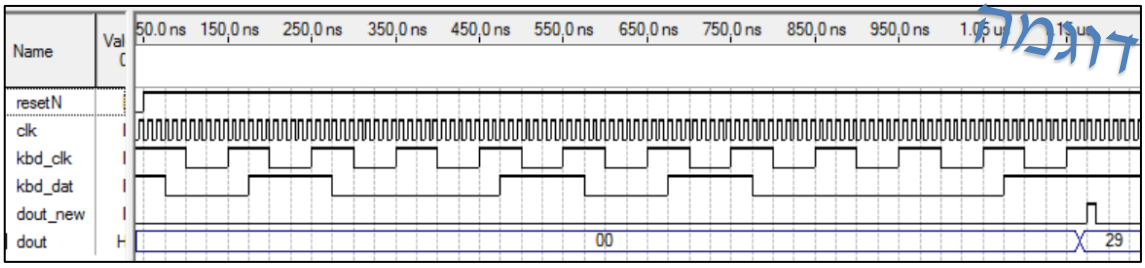
Flow Status	Successful - Tue Aug 18 12:12:19 2020
Quartus Prime Version	17.0.0 Build 595 04/25/2017 SJ Lite Edition
Revision Name	KBDINTF
Top-level Entity Name	random
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	242 / 41,910 (< 1 %)
Total registers	444
Total pins	11 / 499 (2 %)
Total virtual pins	0
Total block memory bits	640 / 5,662,720 (< 1 %)
Total DSP Blocks	0 / 112 (0 %)
Total HSSI RX PCSs	0 / 9 (0 %)
Total HSSI PMA RX Deserializers	0 / 9 (0 %)
Total HSSI TX PCSs	0 / 9 (0 %)
Total HSSI PMA TX Serializers	0 / 9 (0 %)
Total PLLs	0 / 15 (0 %)
Total DLLs	0 / 4 (0 %)

צור SYMBOL של קובץ זה אחרי קומפילציה מוצלחת.

3.2 סימולציה

בצע סימולציה ב- Quartus כדי לדבג את מכונת המצבים שתכננת.

פתח קובץ סימולציה חדש ושרטט את אות הכניסה הבא (עבור כניסת מקש הרווח 29h 01001010001.



הדרכה לסימולציה: מומלץ להגדיר:

- **שעון** מערכת (clk) מהיר פי 10 משעון המקלדת (Kbd_CLK): למשל, קבע בשעון המערכת period=10nsec ובשעון המקלדת period=100nsec.
- **grid** של 25 nsec ושים לב שהשינוי ב- Kbd_DAT מתרחש בזמנים ששעון המקלדת ב- '1' לוגי!
- **משך הסימולציה** כ- End time = 1.5 usec

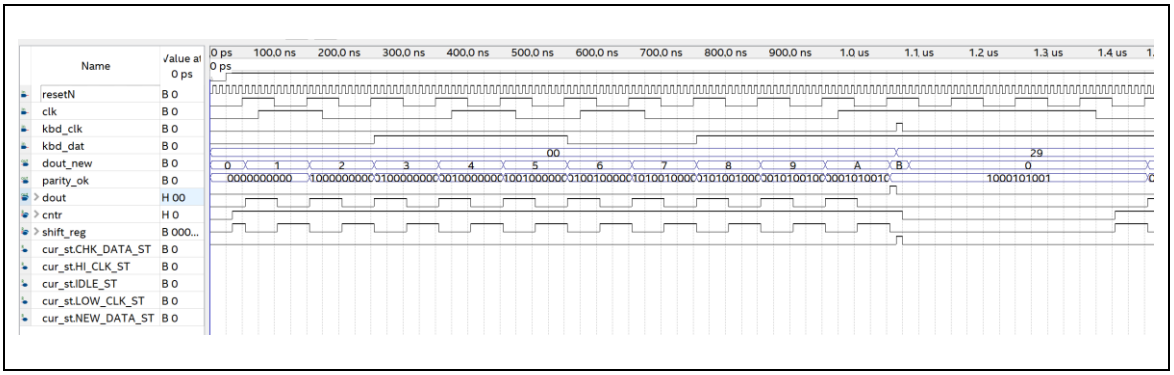
הראה שבסימולציה שלך התוצאות זהות לדוגמה הנתונה להלן. הראה שאם מכניסים רצף טורי של קוד מקש נתון ב- Kbd_DAT, , מתקבל ב- 29H dout מקבילי (הצג אות זה ב- radix hexadecimal) ומתקבל '1' במשך מחזור שעון אחד שמודיע על מקש חדש ב- dout_new אחרי שה- Kbd_CLK האחרון הסתיים (אחרי ה- Stop bit).

חשוב מאד: לביצוע הסימולציה יש להזין אך ורק את אות המבוא KBD_DAT כפי שנתון בדוגמה לעיל!

חשוב להראות בסימולציה גם סיגנלים פנימיים כגון המונה, הSHIFT REGISTER ומכונת המצבים (שורה לכל מצב) , במידת הצורך על מנת שהסימולציה לא תצמצם את המשתנים הוסף להגדרת המשתנים את הפקודה הבאה:

```
logic  [3:0] cntr, nextCntr /* synthesis keep = 1 */ ;
```

צרף לכאן צילום מסך של תוצאות סימולציה מוצלחת.



4 חישוב עומק הזכרון עבור הנתח הלוגי

רקע למטלה: על מנת לדבג את המערכת רוצים לדגום באמצעות הנתח הלוגי את אות המבוא Kbd_DAT של יחידת ה- BITREC בזמן הקשה על מקש כלשהו.

ברוב המקשים קוד המקש מכיל 11 סיביות, אך במקשים מהסוג החדש, הקוד מכיל 11 סיביות נוספות ומחזור שעון הפסקה (למשל הקוד של מקש Down Arrow מהסוג החדש הוא (72 E0). כמו כן, שעון המקלדת Kbd_CLK, שמשמש לסנכרון סיביות הנתונים של Kbd_DAT, עובד בתדר של 12.5 KHz. **לביצוע החישוב היעזר בהסבר המפורט מחומר הרקע**.

חשב מה צריך להיות עומק הזכרון המינימלי בנתח הלוגי הדרוש לקליטת כל הקוד במקרה זה.
חישוב ותשובה:
זמן מחזור = 80 מיקרו-שניות
מספר סיביות בקוד של מקש אחד = 11*1+2 = 23
התדר של שעון הדגימה = 50 מגה הרץ
זמן מחזר * מספר סיביות * תדר = 92000
ולכן נבחן בעומק זכרון של 128K

5 מטלת תכן עם מקלדת

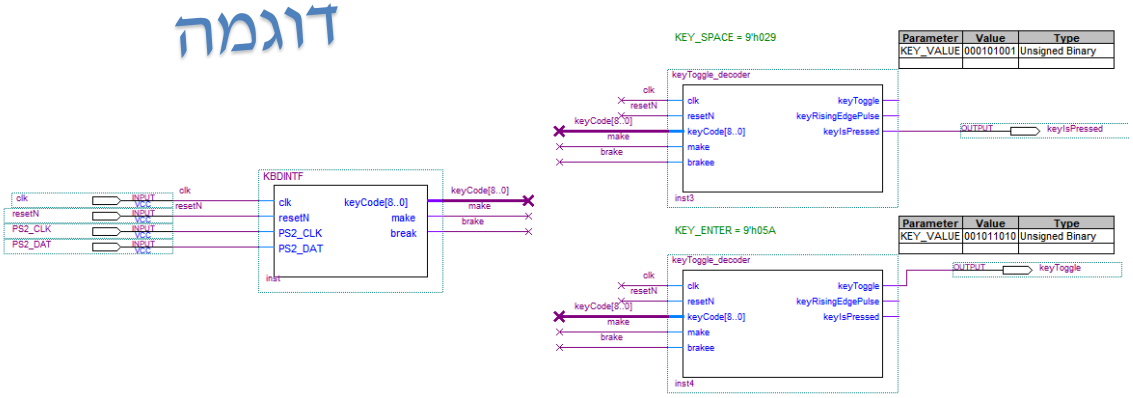
רקע למטלה: בישומים רבים אפשר להשתמש במקלדת לביצוע פעולות שונות, בדומה למפסקים ולחצנים שעל הכרטיס. במטלה זו תלמד איך להשתמש בממשק למקלדת לביצוע פעולות באמצעות מקשים מסוימים.

פתח את הקובץ הגרפי הנתון לך **TOP_KBD_DEMO.bdf**. בישום זה קוד המקש המופק מממשק המקלדת (KBDINTF) מוזן לשני מודולים מאותו סוג:
- מודול keyToggle_decoder שמזהה מקש ספציפי, לפי קוד מקש נתון כפרמטר, ומפיק 3 אותות שונים:

- keyToggle - מחליף מצב כל לחיצה על המקש בין 0 ל-1 לוגי
- keyRisingEdgePulse - גוזר, מוציא פולס צר בתחילת הלחיצה על המקש
- keyIsPressed - מוציא 1 לוגי בכל משך הזמן שהמקש לחוץ

בדוגמה הנתונה משתמשים פעמיים במודול זה, פעם עבור מקש הרווח (עם הפרמטר 9h029) ופעם עבור מקש ה- Enter (עם הפרמטר 9h058).

דוגמה



במטלה זו נתמקד במודול keyToggle_decoder.

פתח את הקובץ keyToggle_decoder וסיים את כתיבת המימוש שלו **במקום בו כתובה ההערה**
&&&&&&&&&& fill your code please

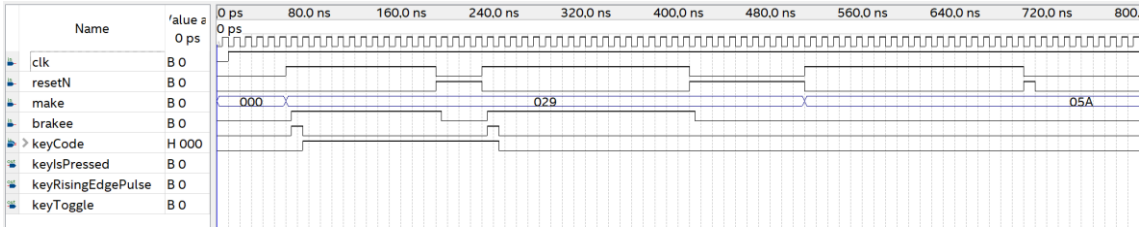
בדוגמה הנתונה מה עושה מקש הרווח ומה עושה מקש ה- Enter?
תשובה:
keyIsPressed 1 עבור מקש הרווח
keyToggle 1 עבור מקש ה Enter

```
module keyToggle_decoder
  (
    input      logic      clk,
       input  logic      resetN,
    input  logic[8:0]     keyCode,
    input  logic      make,
    input  logic      brakee,  // warning "break" is a reserved
SYSVerilog word

    output logic  keyToggle, // toggle this output every time the key
is pressed
    output logic  keyRisingEdgePulse,       //  valid for one clock
after presing the key
    output logic  keyIsPressed // valid while the key is pressed

  ) ;


    parameter KEY_VALUE = 9'h029 ; // space is the default

       logic keyIsPressed_d ; //   _d == delay of one clock

    assign keyRisingEdgePulse = ( keyIsPressed_d == 1'b0 ) && (
keyIsPressed == 1'b1 ) ; // detects a rising edge (change) in the
input


       always_ff @(posedge clk or negedge resetN)
             begin: fsm_sync_proc
                   if (resetN == 1'b0) begin
                         keyIsPressed_d <= 0  ;
                         keyIsPressed  <= 0 ;
                         keyToggle <= 0 ;

                   end
                   else begin
                      if (keyCode  == KEY_VALUE ) begin
                               if (make)
                                     keyIsPressed <= 1'b1;
                               else if (brakee)
                                     keyIsPressed <= 1'b0;
                               else
                                     keyIsPressed <= keyIsPressed;
                         end ;

                         keyIsPressed_d  <= keyIsPressed ; // generate
a delay of one clock
                         keyToggle <= ( keyRisingEdgePulse ) ?
~keyToggle : keyToggle ; // swap on every rising edge

                   end // if
       end // always_ff
```

```
[redacted]

endmodule
```

בצע סימולציה למודול זה (keyToggle_decoder) והראה ששלוש היציאות עובדות נכון עבור לפחות שני מקשים שונים. פעם או פעמיים עם מקש שעובר אותו ופעם עם מקש אחר כלשהוא, וזאת כדי לוודא שהמודול לא מגיב אליו.

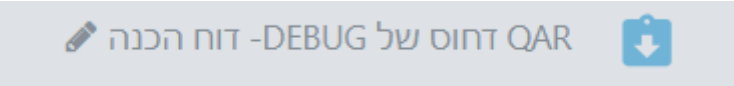
צרף לכאן צילום מסך של תוצאות סימולציה מוצלחת.



6 גיבוי העבודה

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

תגבה את קובץ הארכיב וגם העלה אותו למודל למקום המתאים. במעבדה תמשיך את העבודה על פרויקט זה.



שמור וגבה את הדו"ח שלך רגיל.

שמור את הדוח גם כ- PDF והעלה אותו למודל.

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

מלא את הטופס