

מעבדה בהנדסת חשמל

1א' 044157

**ניסוי DEBUG - ניפוי תקלות בחומרה**

שאלות ודוח הכנה

גרסה 3.5

חורף תשפ"ד 2023-4

|  |  |
| --- | --- |
| שם המדריך | תאריך ההגשה של דוח ההכנה |
| ליאת | 12.02.2024 |

|  |  |  |
| --- | --- | --- |
| סטודנט | שם פרטי | שם משפחה |
| 1 | אמיר | זועבי |
| 2 | עדן | עמוס |

תוכן עניינים של דו"ח הכנה DEBUG

[1 פתיחת הקבצים לעבודה 2](#_Toc155604668)

[2 מכונת RANDOM 3](#_Toc155604669)

[3 ממשק למקלדת 4](#_Toc155604670)

[3.1 תכן יחידת ה - **BITREC** 4](#_Toc155604671)

[3.2 סימולציה 7](#_Toc155604672)

[4 חישוב עומק הזכרון עבור הנתח הלוגי 8](#_Toc155604673)

[5 מטלת תכן עם מקלדת 9](#_Toc155604674)

[6 הכנה למעבדה וגיבוי העבודה 10](#_Toc155604675)

# פתיחת הקבצים לעבודה

**צור תיקיה** למעבדה זו. הורד מהמודל קובץ ארכיב של המעבדה ופתח אותו לפרויקט בתיקייה שיצרת. שים לב לשנות את הPATH שמציע הקוורטוס ל- PATH קצר, שאינו מכיל עברית, רווחים ו/או את הסימן '–' .

**ודא** תכולת קבצים דומה לזו:

קבצים נתונים עבור עבודת הכנה זו

קבצים נתונים עבור המעבדה

**A screenshot of a computer

Description automatically generated**

הקבצים המסומנים בכחול הם עבור עבודת הכנה זו.

הקבצים המסומנים בירוק הם עבור המעבדה. הם נתונים לך עכשיו כחלק מהפרויקט שתתחיל אותו בעבודת ההכנה ותמשיך אותו במעבדה.

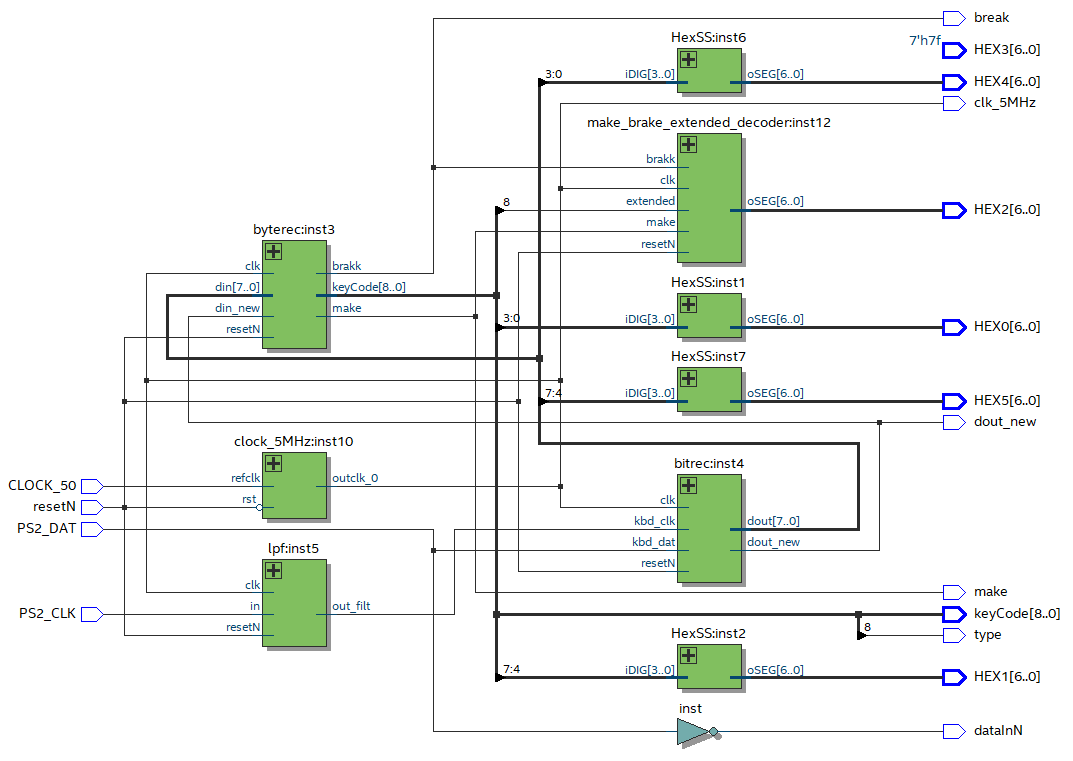
# מכונת RANDOM

**נתון לך קובץ random.sv, המממש 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).



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

תשובה:

היציאה משתנה רק לאחר שהמקש שנלחץ. הפלט הוא הערך של המונה בעליית השעון שבה נקלט בלחיצת. מאחר והפלט תלוי בקלט מהמשתמש, ומאחר ותדר השעון מאוד מהיר, קשה לצפות לערך הבא של המונה מאחר והפלט תלוי במתי המשתמש לחץ על המקש. (יש לנו אנטרופיה כתוצאה מקלט המשתמש)

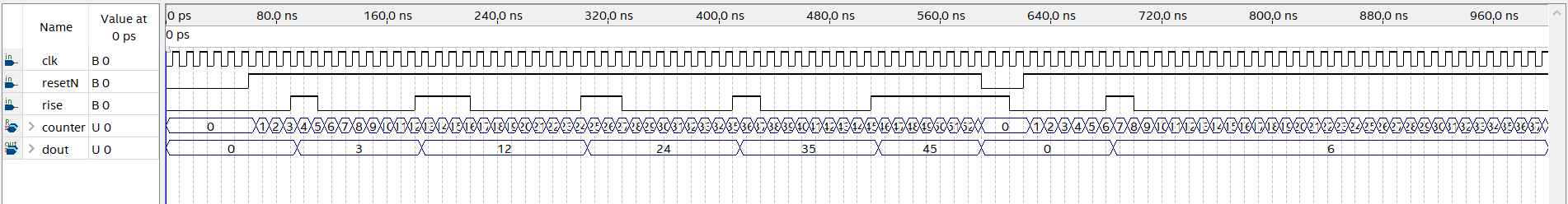
**כיצד ניתן לשנות** את המכונה כך שתגריל מספרים שהם כפולות של 2 בלבד?

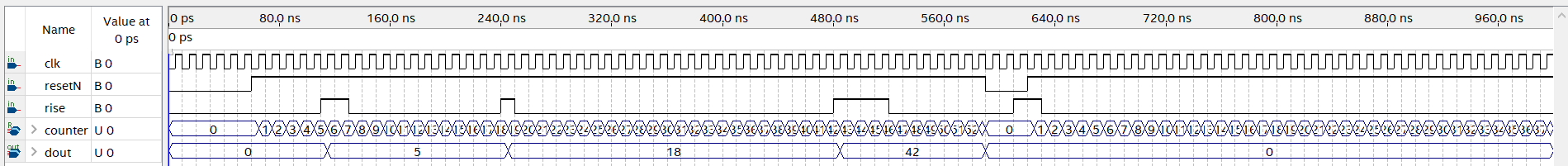
**תשובה:**

**ניתן למשל להגדיל את הערך של count ב 2 במקום ב1 ולהשאיר את שאר הקוד כמו מקודם. הטווח של המספרים יהיה מ0 עד 254 בפלט.**

**הרץ סימולציה** לבדיקת הפעולה של המודול. **הראה** תוצאה אקראית בשני מקרים לפחות.

**הקפד** להציג בסימולציה גם את המונה הפנימי.



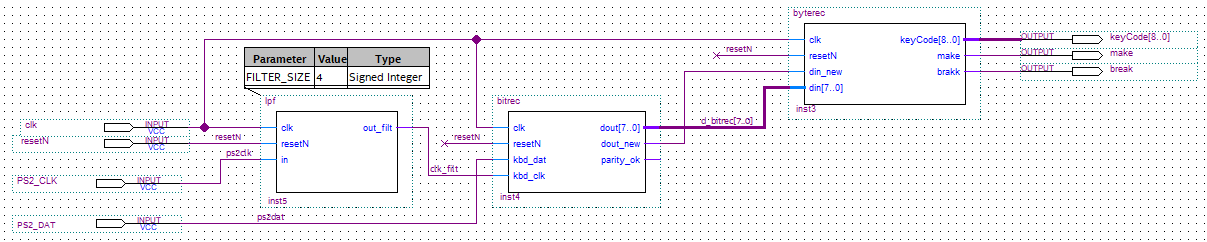


# ממשק למקלדת

**המטרה** בתרגיל זה היא לבנות ממשק למקלדת שיאפשר חיבור מקלדת לכרטיס 10DE. המקלדת תשמש בהמשך להפעלת הפרויקט.

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

דוגמה



יחידות אלה כתובות בשפת SYS-VERILOG ותשמשנה לבנית הממשק למקלדת במעבדה.

להלן הקבצים הנתונים לך שמרכיבים את הממשק למקלדת:

1. יחידת מסנן מעביר נמוכים: lpf.sv – נתונה
2. יחידת המקלט ברמת ה – Bit: bitrec.sv - נתון שלד שלה
3. יחידת המקלט ברמת ה – Byte: byterec.sv – נתונה

## 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 ונקראת dout\_new. דיאגרמת הזמנים הבאה מתארת אותות אלו אחד ביחס לשני וביחס לאות השעון:



|  |
| --- |
| // ===========  LOW\_CLK\_ST**:**  // ===========  **begin**  **if** **(**kbd\_clk **==** 1'b1**)**  **begin**  shift\_reg **<=** **{**kbd\_dat**,** shift\_reg **[**9**:**1**]};** // given in docs  cntr **<=** cntr **+** 4'd1**;**  **if(**cntr **==** NUM\_OF\_BITS**)**  SM\_BITREC **<=** CHK\_DATA\_ST**;**  **else**  SM\_BITREC **<=** HI\_CLK\_ST**;**  **end**  **end** //low\_clk      // ===========  HI\_CLK\_ST**:**  // ===========  **begin**  **if** **(**kbd\_clk **==** 1'b0**)** **begin**  SM\_BITREC **<=** LOW\_CLK\_ST**;**  **end**  **end** //hi\_clk      // ===========  CHK\_DATA\_ST**:**  // ===========  **begin**  **if(**parity\_ok**)** **begin**  dout **<=** shift\_reg**[**7**:**0**];**  SM\_BITREC **<=** NEW\_DATA\_ST**;**  **end**  **else** **begin**  SM\_BITREC **<=** IDLE\_ST**;**  **end**  **end** //chk\_data      // ===========  NEW\_DATA\_ST**:**  // ===========  **begin**  dout\_new **<=** 1'b1**;**  SM\_BITREC **<=** IDLE\_ST**;**  **end** //new\_data        // =======  **default** **:**  // =======  **begin**  SM\_BITREC **<=** IDLE\_ST**;** // bad states recovery  **end** // default    **endcase** |

**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 (זה מה שצריך להיות)
* Error מציין את הסיגנל parity\_ok במצב false
* counter מונה את מספר הביטים של קוד המקש שמגיעים בקו הטורי

**הדרכה ודרישות:**

**כתוב קוד** המתאר את מכונת המצבים באמצעות תהליך סינכרוני בלבד. פתח את הקובץ **bitrec.sv** מתוך הפרויקט הנוכחי 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 שלנו אנא אל תתקנו אותה, אל תדווחו עליה בפורום שאלות ותשובות   
וגם אל תספרו לחבריכם, השאירו להם את חווית הגילוי העצמי!**

**מהו NUM\_OF\_BITS? (לשים לב לגודל הרגיסטר שמקבל את המידע הטורי) הוסף אותו לקוד במקום המתאים.**

**תשובה**: 10. הגודל של השיפט רגיסטר הוא 10. והוא יכיל את שמונת הביטים של הdata, ביט אחד של parity, ואת ה stop bit. ה start bit משומש על מנת לצאת ממצב ה idle.

בטבלה הבאה מפורטים המצבים שבמכונה והפעולות לביצוע בכל מצב.

**מלא את העמודה האחרונה בטבלה לפי הדוגמה שבשורה הראשונה:**

**הערות: - פעולה של המתנה לאות מסויים או ספירה עד ערך מסויים מתורגמת לקוד באמצעות משפטי IF**

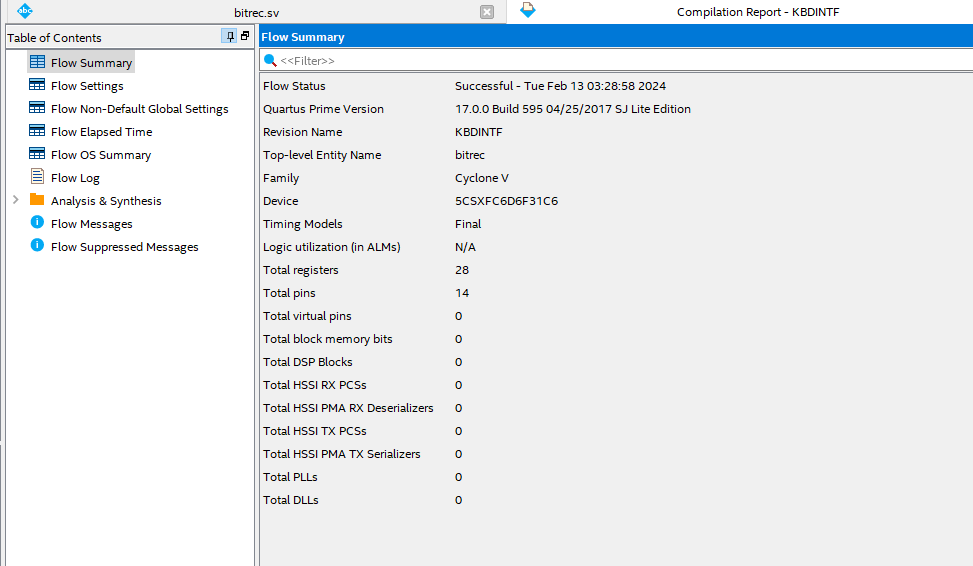
**- אם אין לצאת ממצב מסויים לא צריך לעשות פעולת סרק**

**- היעזרו בדיאגרמת המצבים, בה מופיעים תנאי המעברים, להשלמה של עמודת המעברים בטבלה הבאה**

|  |  |  |
| --- | --- | --- |
| **שם המצב** | **פעילות עיקרית** | **לאיזה מצב עוברים מהמצב הנוכחי ובאילו תנאים – למלא את התאים הריקים** |
| Idle | זהו מצב ההמתנה למילה חדשה. כאן  - מאפסים את המונה count.   - ממתינים למילה חדשה (סימן למילה חדשה – ירידה באותות השעון והנתונים) | **עוברים** ל**-** LowClk  **עם** ירידה באות השעון kbd\_clk וגם ירידה באות kbd\_dat |
| LowClk | זהו מצב קבלת ביט. במצב זה ממתינים לאות שעון גבוה המסמן שהביט הבא מגיע.  אם kbd\_clk גבוה:  - משרשרים למקום האחרון ברגיסטר ההזזה shift\_reg את הסיבית החדשה שהגיעה מה- kbd\_dat.  shift\_reg **=**  **{**kbd\_dat**,** shift\_reg **[**9**:**1**]};**  - בודקים אם מונה הביטים cntr קטן ממספר הביטים. אם כן מקדמים את המונה. | בודקים אם מונה הביטים cntr קטן ממספר הביטים:   - אם כן (טרם הגיעו כל הביטים של המילה) עוברים למצב HiClk, על מנת שנחכה לביט הבא.  - אם לא (המילה השלמה התקבלה) עוברים למצב הבדיקה, שם נבדוק אם ה parity תקין. |
| HiClk | במצב זה השעון גבוה וממתינים לביט הבא (לשעון נמוך). | אם מגיע ביט חדש עוברים בחזרה ל LowClk בשביל לדגום אותו לתוך shift register. |
| ChkData | מצב בו בודקים את נכונות הנתונים על ידי בדיקת הזוגיות parity\_ok.   - אם בדיקת הזוגיות טובה (1 לוגי) מעדכנים את המוצא בתכולת הרגיסטר  dout **=** shift\_reg**[**7**:**0**];** | אם בדיקת הזוגיות טובה עוברים לNewData, שם נוציא כפלט את שמונת הביטים במקביל  - אם הבדיקה לא טובה עוברים לIdle |
| NewData | במצב זה תמיד מודיעים על התו החדש  dout\_new **=** 1'b1 **;** | עוברים לIdle. |

**בצע סינתזה.**

|  |
| --- |
| // (c) Technion IIT, Department of Electrical Engineering 2018  // Written By David Bar-On June 2018  **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 // valid parity laod new data  **}** SM\_BITREC **;** /\* for simulation --> synthesis keep = 1 \*/    **logic** **[**3**:**0**]** cntr**;**  **logic** **[**9**:**0**]** shift\_reg**;**      //-------------------------------------------------------------------------  // &&&&&&&&&&&& please change to the correct number and put in the report  //-------------------------------------------------------------------------    **localparam** NUM\_OF\_BITS **=** 10**;** //???? |1startbit| + [8data + 1parity + 1stopbit ], also not(parity bit) = actual parity  // we dont save the start bit  **always\_ff** **@(posedge** clk **or** **negedge** resetN**)**  **begin:** fsm\_sync\_proc  **if** **(**resetN **==** 1'b0**)** **begin**  SM\_BITREC **<=** IDLE\_ST **;**  cntr **<=** 4'h0 **;**  shift\_reg **<=** 10'h000 **;**  dout **<=** 8'b0 **;**  dout\_new **<=** 1'b0 **;**    **end**  **else** **begin**  **case(** SM\_BITREC **)**    // ===========  IDLE\_ST**:** **begin**  // ===========  dout\_new **<=** 1'b0 **;**  cntr **<=** 4'h0 **;**    **if(** **(**kbd\_clk **==** 1'b0**)** **&&** **(**kbd\_dat **==** 1'b0**)** **)** **begin**  SM\_BITREC **<=** LOW\_CLK\_ST**;**  shift\_reg **<=** 10'h000 **;**  **end**  **end** //idle    //----------------------------------------------------------------  // &&&&&&&&&&&&&& fill your code and paste to the report #1  //----------------------------------------------------------------  //  // ===========  LOW\_CLK\_ST**:**  // ===========  **begin**  **if** **(**kbd\_clk **==** 1'b1**)**  **begin**  shift\_reg **<=** **{**kbd\_dat**,** shift\_reg **[**9**:**1**]};** // given in docs  cntr **<=** cntr **+** 4'd1**;**  **if(**cntr **==** NUM\_OF\_BITS**)**  SM\_BITREC **<=** CHK\_DATA\_ST**;**  **else**  SM\_BITREC **<=** HI\_CLK\_ST**;**  **end**  **end** //low\_clk      // ===========  HI\_CLK\_ST**:**  // ===========  **begin**  **if** **(**kbd\_clk **==** 1'b0**)** **begin**  SM\_BITREC **<=** LOW\_CLK\_ST**;**  **end**  **end** //hi\_clk      // ===========  CHK\_DATA\_ST**:**  // ===========  **begin**  **if(**parity\_ok**)** **begin**  dout **<=** shift\_reg**[**7**:**0**];**  SM\_BITREC **<=** NEW\_DATA\_ST**;**  **end**  **else** **begin**  SM\_BITREC **<=** IDLE\_ST**;**  **end**  **end** //chk\_data      // ===========  NEW\_DATA\_ST**:**  // ===========  **begin**  dout\_new **<=** 1'b1**;**  SM\_BITREC **<=** IDLE\_ST**;**  **end** //new\_data        // =======  **default** **:**  // =======  **begin**  SM\_BITREC **<=** IDLE\_ST**;** // bad states recovery  **end** // default    **endcase**    //-------------------------------------------------------------  // &&&&&&&&&&&&&& end of paste SM to the report #1  //-------------------------------------------------------------  **end** // else clk  **end** // end fsm\_sync\_proc  // 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**[**1**]**/\*<-this is a mistake, should be 4\*/ **^** shift\_reg**[**1**]** **^** shift\_reg**[**0**];**  **endmodule** |

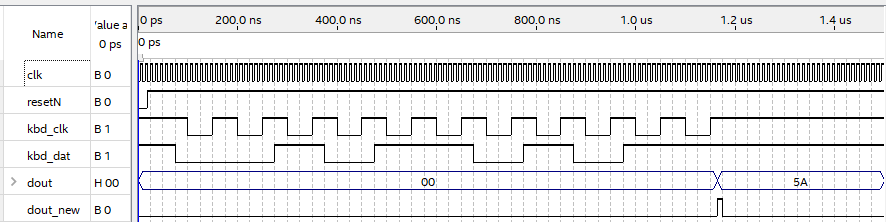


**צור SYMBOL** של קובץ זה אחרי סינתזה מוצלחת.

## סימולציה

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

פתח קובץ סימולציה חדש ושרטט את אות הכניסה הבא (עבור כניסת מקש ה- Enter 5Ah או המילה של 11 סיביות בבינארי 00101101011.



דוגמה

**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 nsec 25 ושים לב שהשינוי ב- kbd\_dat מתרחש בזמנים ששעון המקלדת ב- '1' לוגי!
* **משך הסימולציה**  כ- End time = 1.4 usec

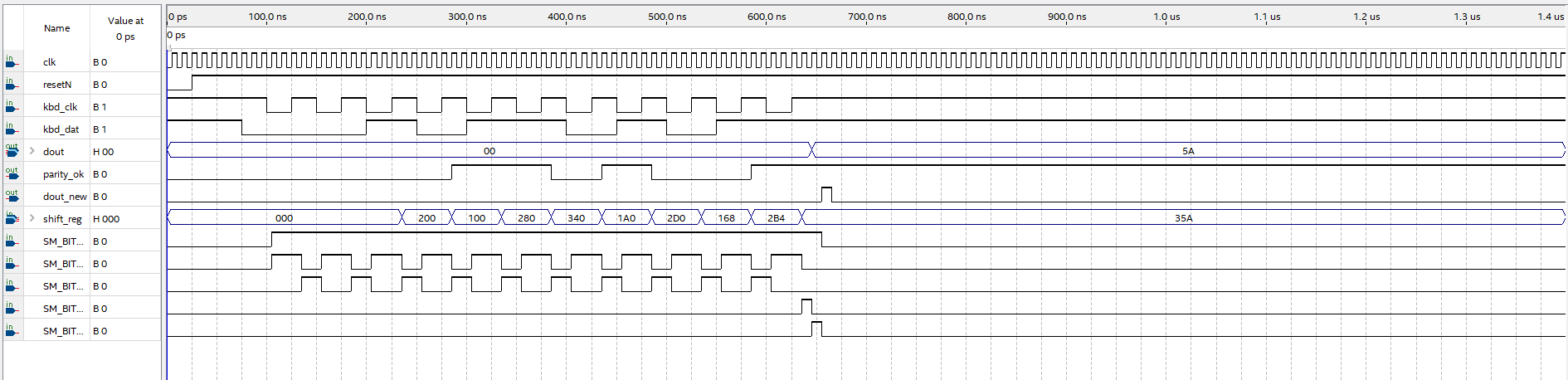
**הראה בסימולציה** שלך תוצאות זהות לדוגמה הנתונה לעיל. הראה שאם מכניסים רצף טורי של קוד מקש נתון ב- kbd\_dat, מתקבל:

* ב- :dout 5AH מקבילי (הצג אות זה ב- radix hexadecimal)
* ב- dout\_new '1' במשך מחזור שעון אחד שמודיע על מקש חדש 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) , שים לב שעל מנת שהסימולציה לא תצמצם את המשתנים יש להוסיף להגדרת המשתנים את הפקודה הבאה, כפי שכבר קיים בקוד:

**logic** **[**3**:**0**]** cntr /\* synthesis keep = 1 \*/ **;**



# 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 Enter מהסוג החדש הוא E0 5A)). כמו כן, שעון המקלדת PS2\_CLK, שמשמש לסנכרון סיביות הנתונים של PS2\_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:**

# 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. בדוגמה כאן משתמשים במקלדת מספרים של 18 מקשים בלבד (ראה הסבר בחומר הרקע).

**פתח** את הקובץ הגרפי הנתון KEYBOARD\_DECODER.bdf .

**הסבר למודול KEYBOARD\_DECODER.bdf**:

בישום זה קוד המקש המופק מממשק המקלדת (KBDINTF) מוזן לשני מודולים מאותו סוג בשם **singleKeyDecoder.sv**, במטרה לעשות פעולות מסויימות באמצעות מקשים מוגדרים של המקלדת.

**במטלה זו נתמקד במודול singleKeyDecoder.sv ונשלים בו קוד חסר**.

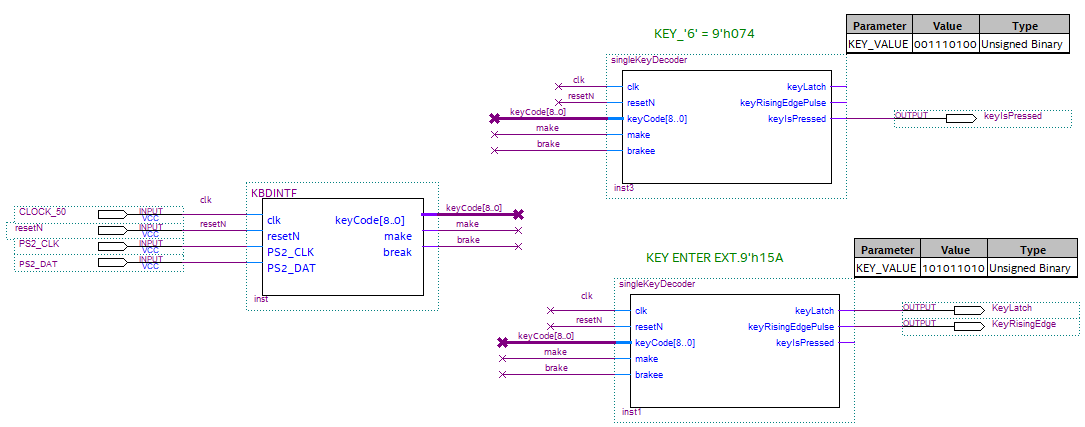
**הסבר למודול singleKeyDecoder.sv**

מודול זה מקבל כניסות מממשק המקלדת – קוד מקש, ואותות make ו- break.

הוא מזהה מקש ספציפי לפי קוד מקש נתון כפרמטר ומפיק 3 אותות שונים:

* keyLatch - מחליף מצב כל לחיצה על המקש בין 0 לוגי 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 "6 חץ ימינה**" (עם הפרמטר 9'h074) ופעם עם מקש **ENTER** (עם הפרמטר 9'h15A).



דוגמה

**פתח** את **הקובץ singleKeyDecoder.sv וסיים** את כתיבת המימוש שלו **במקום בו כתובה ההערה** (העזר בחומר רקע)

**&&&&&&&&&& fill your code please**

**בדוגמה הנתונה מה עושה המקש "6 חץ ימינה" ומה עושה המקש ENTER?**

**תשובה**:

**הוסף** את הקוד שהשלמת לדוח:

החלף בקוד

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

**צרף** להלן צילום מסך של תוצאות סימולציה מוצלחת. אפשר לצרף כמה תמונות, לפי הצורך.

החלף בתוצאות סימולציה

# הכנה למעבדה וגיבוי העבודה

**כהכנה נוספת למעבדה** מומלץ לראות את הסרטונים הנתונים במודל, המסבירים על השימוש בנתח הלוגי, ה- SignalTap.



**שמור דוח זה רגיל וכ- PDF והעלה את קובץ ה- PDF למודל.**

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

שים לב לשנות את השם שמציע הקוורטוס לשם קצר, שאינו מכיל: עברית, רווחים ו/או את הסימן '–' ומכיל את התאריך והשעה של יצירת הארכיב.

**העלה את קובץ הארכיב למודל, כי תצטרך אותו בניסוי, בסיס לעבודתך במעבדה.**

**גבה את הדוח והפרויקט גם באמצעים אחרים.**

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) | |
|  |