

הטכניון – מכון טכנולוגי לישראל
הפקולטה להנדסת חשמל



מעבדות 1, ח1

חוברת ניסוי לסטודנט בנושא :

ניפוי תקלות בחומרה (DEBUG)

חומר רקע

QUARTUS 17

הניסוי פותח בחסות המעבדה למערכות ספרתיות מהירות 

גרסה 1.0 (אביב 2018)

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

תוכן עניינים של חומר הרקע - ניפוי תקלות בחומרה (DEBUG)

| | | |
|-------|---|----|
| 1 | הקדמה | 3 |
| 1.1 | מטרות הניסוי | 3 |
| 1.2 | מבנה הניסוי, אופן ביצועו ודרישות מהסטודנט | 3 |
| 2 | הנתח הלוגי ב- Quartus | 4 |
| 2.1 | נתח לוגי חיצוני ופנימי | 4 |
| 2.2 | הכרת מושגים בסיסיים – מספר רמות המתח ותדר הדגימה | 5 |
| 2.3 | מושגים בסיסיים – מבנה זיכרון הדגימה ודרבון (Trigger) | 8 |
| 2.4 | מושגים בסיסיים – הקלטה בזיכרון סיבובי לפני ואחרי הדרבון | 9 |
| 2.5 | מבוא לשימוש ב- SignalTap | 11 |
| 3 | ממשק למקלדת | 12 |
| 3.1 | כללי | 12 |
| 3.1.1 | מבוא ונתונים על ממשק למקלדת ברמת החמרה | 12 |
| 3.1.2 | מבוא ונתונים על ממשק בין מקלדת PS/2 ו Host ברמת ה-bit | 13 |
| 3.1.3 | גילוי שגיאה בעזרת סיבית ה- (odd)parity | 14 |
| 3.1.4 | מבוא לתשדורת של המקלדת ברמת ה-byte | 14 |
| 3.2 | מימוש ממשק למקלדת | 16 |
| 3.2.1 | תכן של פילטר מעביר נמוכים | 17 |
| 3.2.2 | תכן יחידת ה- BITREC | 17 |
| 3.2.3 | תכן יחידת ה- BYTEREC | 19 |
| 4 | מערכת הניסוי לבדיקת החומרה HW_DEBUG עם מכונת RANDOM | 21 |
| 5 | עריכת תוכן של זיכרון וקבועים בזמן אמת ISMCE | 22 |
| 6 | נספח – מימוש ישום ה-CapsLock | 23 |

1 הקדמה

1.1 מטרת הניסוי

- ☐ הכרת יכולות השילוב בין תיאור גרפי וטכסטואלי בכלי הפתוח Quartus
- ☐ תאור מכונת מצבים
- ☐ תאור מממשק למקלדת
- ☐ הכרת כלי עזר בחמרה לניפוי שגיאות: Signal-TAP (נתח לוגי משובץ בחמרה)
- ☐ שימוש ב-Signal-TAP בממשק למקלדת
- ☐ שימוש ב-Signal-TAP בניפוי תקלה

1.2 מבנה הניסוי, אופן ביצועו ודרישות מהסטודנט

הניסוי מתבצע במפגש מעבדה אחד בן 4 שעות ודורש גם זמן הכנה מוקדם לפני המפגש.

לפני מפגש המעבדה יש לקרוא את כל "חומר הרקע" שמופיע בחוברת זו. בנוסף לכך יש לענות על שאלות ההכנה, להכין את הקודים ולהריץ את תרגילי ההכנה. תרגילים אלו מופיעים כולם "בדוח הכנה". בגמר ההכנה לניסוי יהיו בידך כל האלמנטים לביצוע התכן הסופי במעבדה.

דוח ההכנה שתגיש צריך לכלול: תשובות לכל השאלות, תדפיס של כל הקודים, תדפיס מסך של תוצאות הנדרשות.

במפגש המעבדה יתבצע תרגול מודרך באמצעות: Quartus וכרטיס התרגול. בתכן תיצור ותבדוק חמרה שלתוכה הוכנס Logic Analyzer. בנוסף לכך תבדוק חמרה שמשנים בה תוכן של רכיב זיכרון, וקריאת תוכן של זיכרון בזמן פעולת המערכת. בהמשך תהיה לך גם הזדמנות לשלב את כלי הדיבוג החשובים שאותם תכיר בתכן של חמרת הממשק למקלדת PS/2 ובישום נוסף שלו.

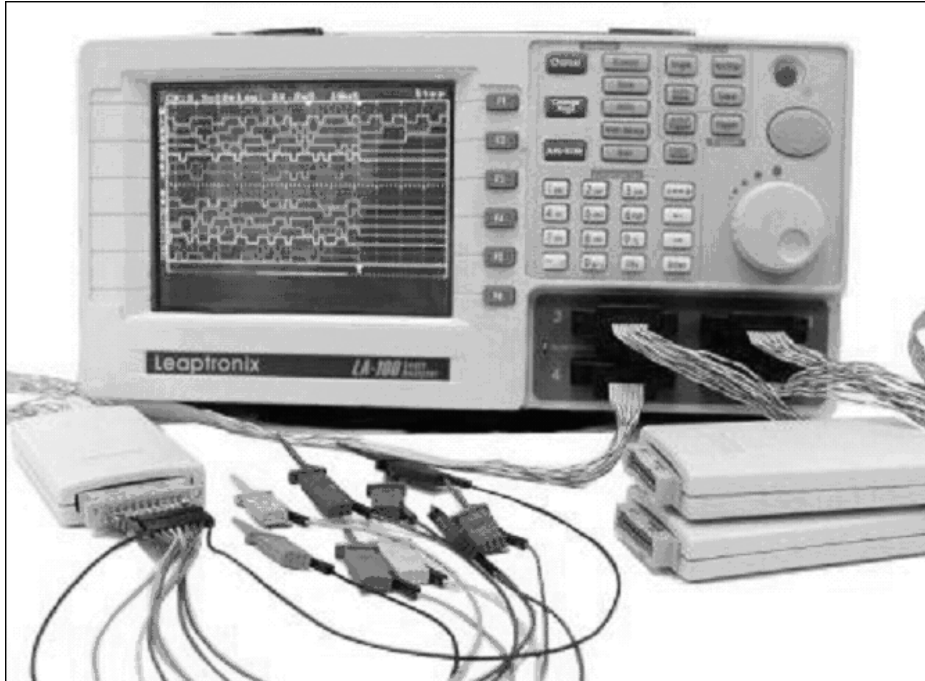
הערה חשובה:

כל הקבצים המוכנים או החצי מוכנים לניסוי זה ימצאו במודל

2 הנתח הלוגי ב- Quartus

2.1 נתח לוגי חיצוני ופנימי

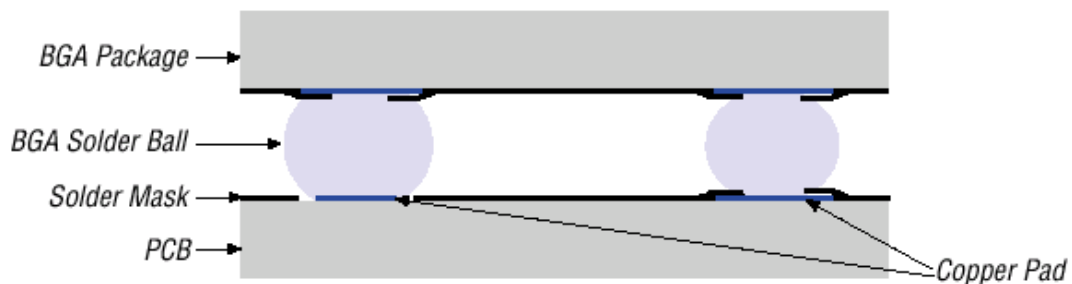
המכשיר שנקרא - Logic Analyzer הוא אחד המכשירים הבסיסיים החשובים ביותר בבדיקה של מערכות ספרתיות. מדובר במכשיר שדומה באופן פעולתו לסקופ (Oscilloscope) בעל זיכרון ומספר רב של ערוצים שמיועד ליישומים ספרתיים. נתח לוגי רגיל הוא בדרך כלל מכשור מעבדתי שמחובר באמצעות הגששים (הפרובים - Probes) שלו להדקים חיצוניים של הרכיבים או להדקי בדיקה מיוחדים של המערכת.



בדיקה של רכיבים מיתכנתים מודרניים באמצעות מכשור חיצוני כמו Logic Analyzer הולכת ונעשית מורכבת. קימות לכך כמה סיבות.

רכיבים מודרניים הולכים ונעשים מורכבים ומספר האותות הפנימיים שלהם, שעשוי להיות בעל עניין, הוא עצום! רוב האותות הפנימיים אינם מחוטים כלל מחוץ לרכיב!

בנוסף לכך הגישה להדקים של הרכיב, שאכן מחוטים החוצה, הולכת ונעשית קשה יותר הן בגלל הממדים הפיסיים הקטנים של הרכיבים והן בגלל סוגי האריזות שבהן נעשה שימוש כיום. הרכיב שבו אנו משתמשים למשל בניסוי זה (5CSXFC6D6F31C6) הוא רכיב באריזה שנקראת Fine Line BGA (השם BGA בא מהמלים Ball Grid Array). זהו רכיב שבתחתית שלו יש 896 הדקים כדורים מאוד קטנים שמולחמים למעגל המודפס!



הרגליים אינן עוברות לצד השני של המעגל המודפס (כמו באריזות DIP ישנות) ואי אפשר לחבר אליהם גששים של Logic Analyzer.

הפתרון של יצרנים של רכיבים מיתכנתים לכל הבעיות הקשות שהועלו כאן היא: לאפשר הכנסה של Logic Analyzer לתוך הרכיב. נתחים לוגיים אלו נקראים Embedded Logic Analyzers או בקצור (ELA). ה- Logic Analyzer שמוכנס לרכיבי Altera נקרא בשם SignalTap.

הנגישות הגבוהה של ה- SignalTap לאותות הפנימיים הרבים של התכן הופכת אותו לכלי דיבוג רב עוצמה !

החמרה של הנתח הלוגי ממומשת באמצעות המשאבים הלוגיים הפנימיים ומשאבי הזיכרון הפנימיים ברכיב. התכן של הנתח והשילוב שלו עם המערכת הנבדקת נעשה באופן אוטומטי באמצעות כלי הפתוח Quartus ובהתאם לדרישות המשתמש. ההורדה של התכן שכולל גם את הנתח הלוגי הפנימי נעשית באמצעות קונפיגורציה רגילה של הרכיב באמצעות שרשרת ה- JTAG.

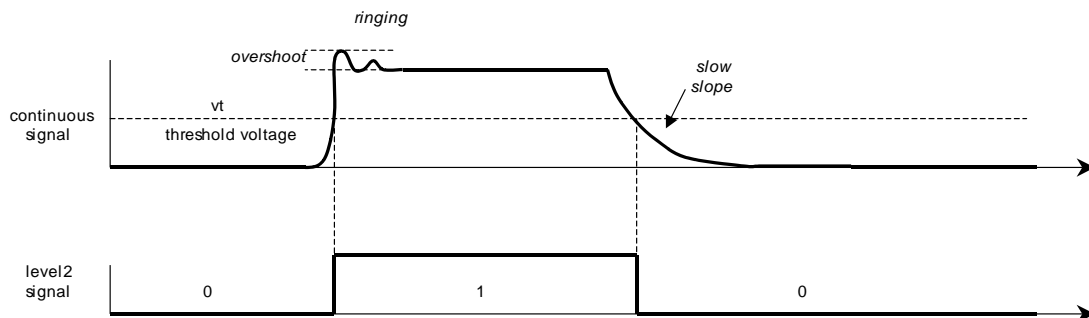
הפעלת הנתח והתקשורת עמו נעשים גם הם באמצעות שרשרת ה- JTAG. גם ההפעלה של הנתח וגם הצגת התוצאות של הנתח נעשים באמצעות תכנת ה- Quartus.

השימוש ב- SignalTap הוא נוח והוא אינו דורש רכישה של מכשור יקר !

בחלקים הבאים נכיר כמה מושגים כלליים ותכונות של מכשירי Logic Analyzer בכלל ושל ה- SignalTap בפרט.

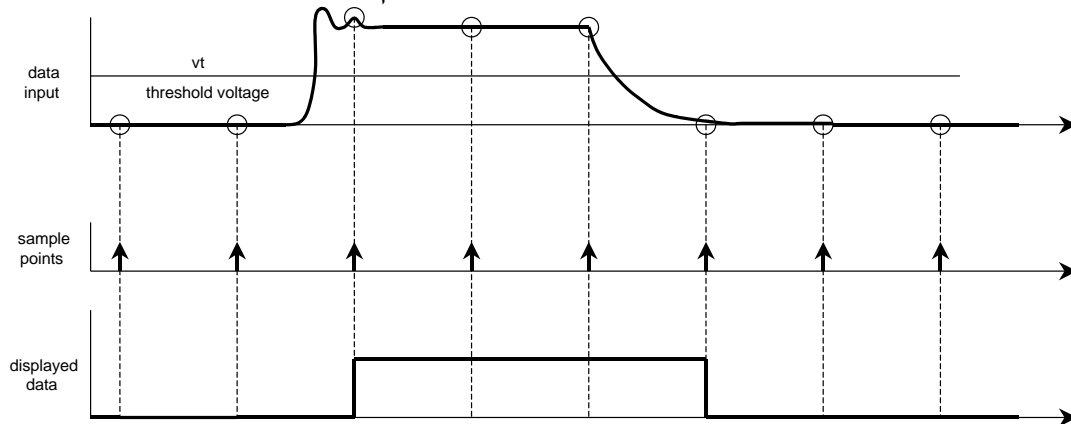
2.2 הכרת מושגים בסיסיים – מספר רמות המתח ותדר הדגימה

הנתח הלוגי (Logic Analyzer) בניגוד לסקופ (Oscilloscope) הוא מכשיר שמציג אותות ספרתיים באחת משתי רמות בלבד: '0' לוגי או '1' לוגי - וללא רמות ביניים נוספות. האירור הבא מציג דוגמה לאות כניסה כל שהוא והדרך שבה הוא מוצג באמצעות הנתח הלוגי כאות בעל שתי רמות מתח בלבד.



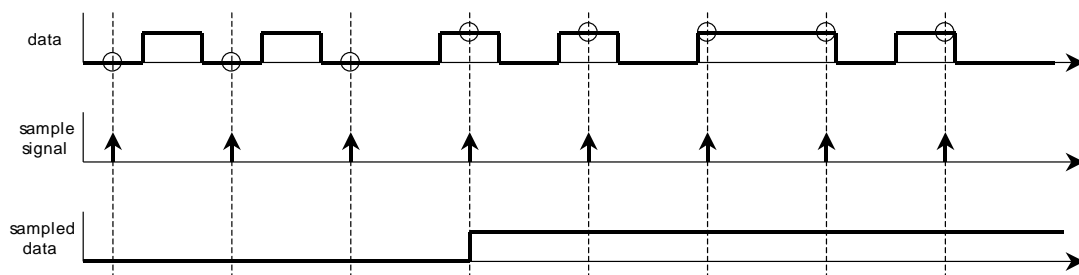
נתחים לוגיים הם מכשירים שאינם מיועדים להצגה של תופעות אנלוגיות כמו - זמני עליה (Transition Times) ושיפועים (Slew Rate or Slope) או תגובת יתר (Over-Shoot) וצלצולים (Ringing).

נתחים לוגיים הם מכשירים שדוגמים (Sample) את אות הכניסה בזמנים קצובים ומקליטים אותו (Acquisition) לזיכרון פנימי של המכשיר. האירור הבא מתאר אות שעבר דגימה באמצעות הנתח הלוגי. אות הדגימה הוא האות המרכזי בדיאגרמה ונקודות הדגימה מתוארים באמצעות חצים אנכיים. רמות המתח הדגומות מתוארות באמצעות עיגולים קטנים שמצוירים על אות הכניסה.



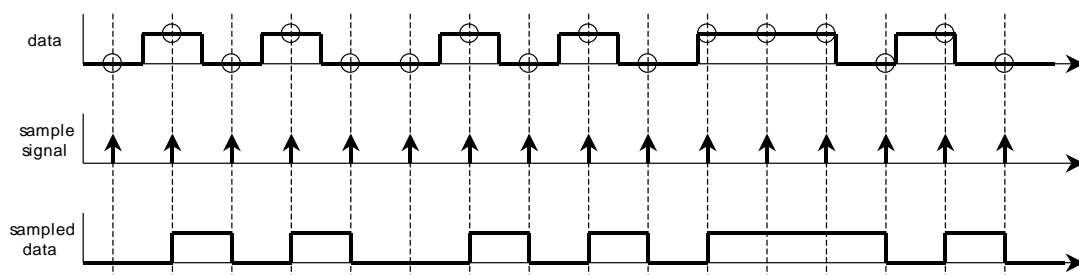
בחלק התחתון של האירור מוצג האופן שבו יוצג האות הדגום באמצעות הנתח הלוגי.

בחירת תדר הדגימה היא חשובה. בציור הבא אנו רואים בשורה הראשונה את שנקרא data שנדגם בקצב דגימה שמוצג באמצעות השורה השניה שבאירור (האות שנקרא sample signal). האות המשוחזר שמוצג על ידי ה- Logic Analyzer מוצג בשורה התחתונה שבציור (sampled data).



מכיוון שקצב הדגימה אינו גבוה מספיק, האות המשוחזר אינו דומה כלל לאות המקורי. מהו קצב הדגימה המינימלי (או זמן הדגימה המכסימלי) שמאפשר שחזור (reconstruction) של האות הנדגם? האם קיים קריטריון דומה למה שמוכר לנו בעולם האנלוגי כתדר Nyquist?

הציור הבא מדגים מקרה גבולי שבו קצב הדגימה מספיק גבוה (זמן הדגימה מספיק נמוך) בכדי לשחזר את האות din שהוצג בציור הקודם.



כפי שאפשר לראות קצב הדגימה הוא כזה, שזמן המחזור שלו מבטיח לפחות דגימה אחת בכל אזור שבו האות הנדגם נמצא ב-Low ודגימה אחת לפחות בכל אזור שבו האות הנדגם נמצא ב-High. אפשר לרשום את הקריטריון באופן הבא:

$$T_{Sample} < \text{Min}(t_{WL})$$

$$T_{Sample} < \text{Min}(t_{WH})$$

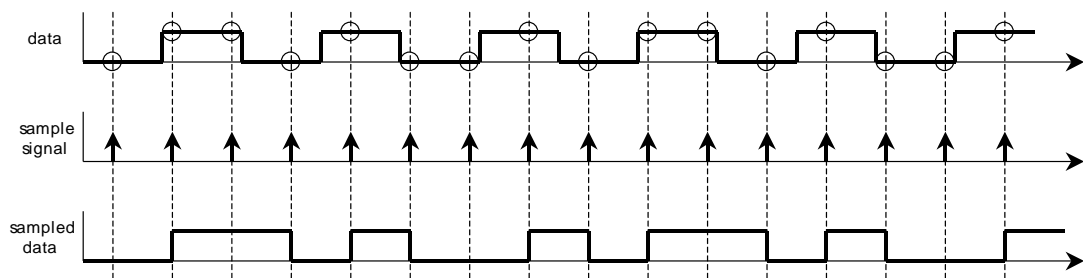
כאשר $\text{Min}(t_{WL})$ ו $\text{Min}(t_{WH})$ מתארים את רוחב הפולס המינימלי ברמות High ו Low בהתאמה של האות הנדגם. קל להראות שזהו במעין קריטריון Nyquist כפי שאנו מכירים אותו במישור התדר.

$$f_{MAX} = \frac{1}{\text{Min}(t_{WL}) + \text{Min}(t_{WH})} < \frac{1}{T_{Sample} + T_{Sample}}$$

או בקיצור :

$$2f_{MAX} < f_{Sample}$$

הקריטריון הנ"ל אמנם מספיק בכדי שנוכל לשחזר את האות מבלי "לפספס" שינויים כלשהם באות, אך בדרך כלל נעדיף לדגום אותות בקצב גבוה יותר (בזמן מחזור קצר יותר) בכדי שהם יראו טוב יותר. הדוגמה הבאה מדגימה למה הכוונה.



קצב הדגימה הוא אמנם קצת גבוה יותר מהמינימום הנדרש בכדי לקיים את הקריטריון הנ"ל. למרות זאת, אם מתבוננים באות המשוחזר נראה שהאות המקורי אינו אות בעל קצב אחיד. כמובן שאין הדבר כך. האות המקורי בדוגמה הזאת הוא אות מחזורי לחלוטין.

בכדי למנוע עיוותים מסוג זה במישור הזמן, בדרך כלל משתמשים בתדר דגימה גבוה יותר (זמן דגימה נמוך יותר). כמובן שככל שתדר הדגימה גבוה יותר מידות הזמן של האות המשוחזר מדויקות יותר. מצד שני אם משתמשים בתדר דגימה גבוה מאוד (זמן דגימה נמוך מאוד) מבזבזים משאבי זיכרון יקרים של ה - Logic Analyzer.

בדרך כלל כאשר משתמשים ב - Logic Analyzer למטרות דיבוג כללי (ולא למטרת מדידות זמן מדויקות) נוהגים להשתמש בזמני דגימה שהם קטנים לפחות פי עשר מזמן הדגימה שהוצג בחלק הקודם :

$$T_{Sample} < \frac{\text{Min}(t_{WL})}{10}$$

$$T_{Sample} < \frac{\text{Min}(t_{WH})}{10}$$

כאשר רוצים לבצע מדידות זמן מדויקות יותר יש להשתמש בזמני דגימה נמוכים יותר (תלוי כמובן באחוז השגיאה במדידה שאותו אנו מוכנים לסבול).

2.3 מושגים בסיסיים – מבנה זיכרון הדגימה ודרבון (Trigger)

נתחים לוגיים הם מכשירים שדוגמים בדרך כלל יותר מסיבית אחת בבת אחת (למשל 32 סיביות). מספר הערוצים (Number of Channels) או רוחב המידע (Data width) של ה-SignalTap יכול לנוע בין סיבית בודדת ל-1024 סיביות.

מספר הדגימות שהמכשיר מסוגל להקליט (Number Of Samples) או עומק הדגימה (Sample Depth) של ה-SignalTap יכול לנוע בין 64 ל-128 K. להלן רשימת כל האפשרויות: 64, 128K, 64K, 32K, 16K, 4K, 2K, 1K, 512, 256, 128.

קצב הדגימה נקבע בדרך כלל באמצעות שעון הדגימה (Trigger Clock). לעתים קרובות זהו פשוט אות השעון הראשי של המערכת, שהוא גם בדרך כלל האות בעל התדר הגבוה ביותר במערכת. כאשר מוכנים לוותר על הדיוק ורוצים לחסוך במשאבי הזיכרון של המערכת משתמשים באות דגימה בעל תדר נמוך יותר מתדר השעון של המערכת.

לדוגמה, כיצד לחשב מה עומק הזכרון שצריכים להקצות ב-Signal Tap, כדי לקלוט קוד שלם של מקש (שמכיל 11 סיביות)? נשתמש בכלל:

$$\text{bits} * \text{time} * \text{frequency} = \text{memory}$$







על פי הכלל הנ"ל נבצע את החישוב באופן הבא:

- זמן מחזור שעון אחד של kbd_clk (זמן של סיבית אחת): 80usec
- מספר הסיביות בקוד של מקש אחד: 11 (עוברת סיבית אחת בכל פולס שעון)
- משך הזמן של קוד שלם של מקש: $80 \times 11 = 880\text{usec}$
- התדר של שעון הדגימה (השעון בלוח התרגול DE2): 50MHz
- מספר הדגימות לקליטת קוד שלם של מקש: $880\text{us} * 50 \text{ MHz} = 44\text{k}$

מסקנה: יש לבחור עומק זיכרון של **64K** ב-Signal Tap.

אות ה-Trigger (אות הדרבון) הוא האות שמורה לנתח הלוגי לעצור את הקלטת האותות הנבחרים לזיכרון ה-RAM. אות ה-Trigger יכול להיות אות חיצוני שמגיע למערכת (Trigger In Signal) או שהוא יכול להיווצר באופן פנימי מהאותות הנדגמים עצמם.

אות ה-Trigger הפנימי נוצר בדרך כלל באמצעות בדיקה של צירוף כל שהוא של הסיביות הנדגמות. בדרך כלל בודקים שכל אחת מהסיביות הנדגמות מקיימת שילוב כל שהוא של אחד מהתנאים האפשריים הבאים:

| | |
|--------------|---|
| Don't Care |  |
| Rising Edge |  |
| Falling Edge |  |
| Either Edge |  |
| High |  |
| Low |  |

נמחיש את מה שנאמר כאן בדוגמה. נניח שאנו קובעים תנאי דרבון (Trigger Condition) הבא על ששת אותות הכניסה - D[5..0]:

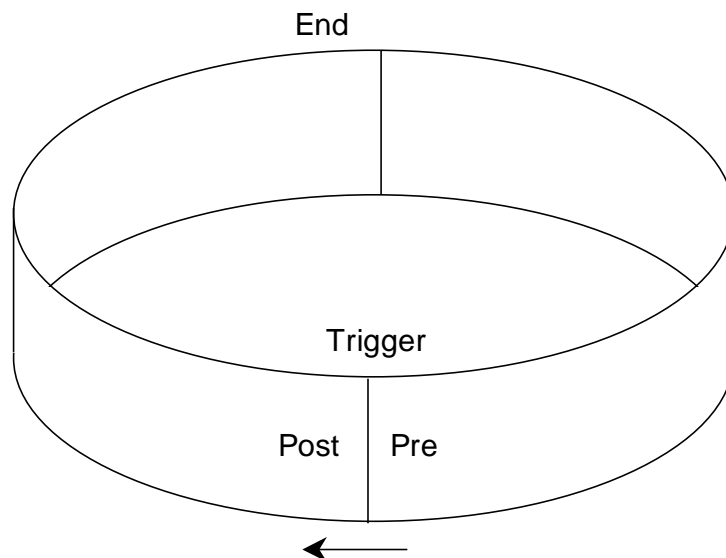
- ☐ עליה (Rising Edge) באות D0
- ☐ האותות D1 ו D2 ימצאו ה - '0' לוגי (Low)
- ☐ האות D3 ימצא ב - '1' לוגי (High)
- ☐ האותות האחרים (D4 ו D5) אינם חשובים (Don't Care)

בכל פעם שהתנאי הנ"ל יתקיים המערכת תעצור את הקליטה והרישום של הדגימות לזיכרון של הנתח הלוגי.

ניתן ליצור תנאי Trigger מורכבים יותר באמצעות כמה מאורעות שאינם מתבצעים בו זמנית. הנתח ממתין לביצוע התנאי הראשון ולאחר מכן הוא ממתיך לביצוע התנאי השני וכו.. עד לביצוע כל התנאים. לכל תנאי כזה קוראים Trigger Level. ב - SignalTap ניתן ליצור עד עשרה Trigger Levels.

2.4 מושגים בסיסיים – הקלטה בזיכרון סיבובי לפני ואחרי הדרבון

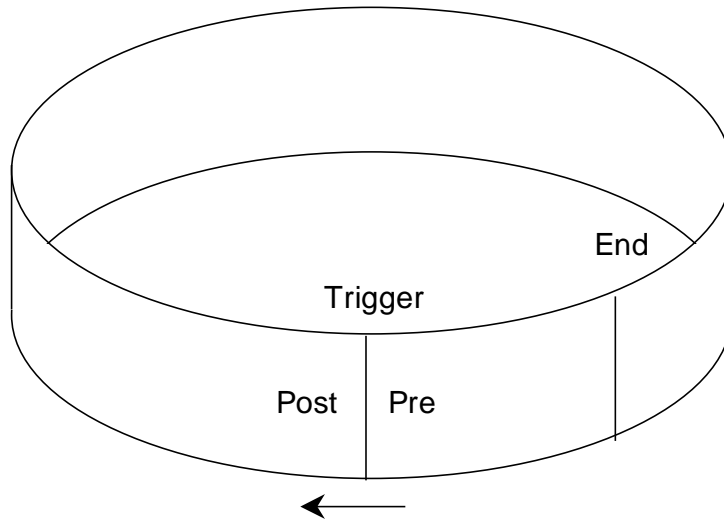
אפשרות חשובה שקיימת במכשירי Logic Analyzer היא להציג את מה שקרה לפני ה - Trigger. רישום של מאורעות שקרו לפני ה - Trigger מתאפשר באמצעות ההקלטה (Acquisition) שנעשית באופן מתמיד גם לפני ה - Trigger לתוך זיכרון סיבובי (Circular Buffer), כאשר הדגימות החדשות "דורסות" ללא הרף דגימות ישנות. מרגע שמתקבל Trigger, ממשיכים לאחסן דגימות רק בחלק מהזיכרון הסיבובי. הציור הבא מדגים מקרה שבו ממלאים חצי מהזיכרון בדגימות חדשות שנעשות לאחר ה - Trigger.



בגמר פעולה זו, חצי מהזיכרון (בצד ימין באיור) מכיל דגימות שנעשו לפני ה - Trigger וחצי מהזיכרון (בצד שמאל באיור) מכיל דגימות שנעשו לאחר ה - Trigger. לצורת עבודה זו קוראים:

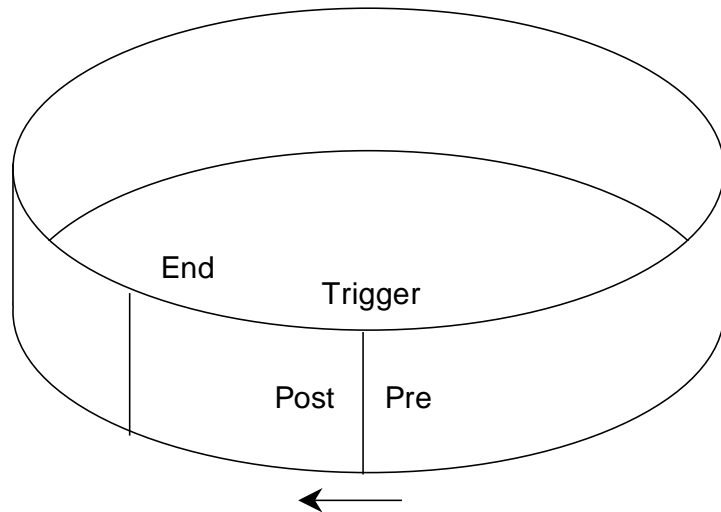
50 % Pre-Trigger / 50% Post-Trigger

הציור הבא מדגים מקרה שבו ממשיכים למלא יותר ממחצית מהזיכרון בדגימות חדשות שנעשות לאחר ה - Trigger.



בצורת עבודה זו רוב הזיכרון יהיה מוקצה להקלטות שנעשו אחרי ה - Trigger.

הציור הבא מדגים מקרה שבו ממשיכים למלא פחות ממחצית מהזיכרון בדגימות חדשות שנעשות לאחר ה - Trigger.

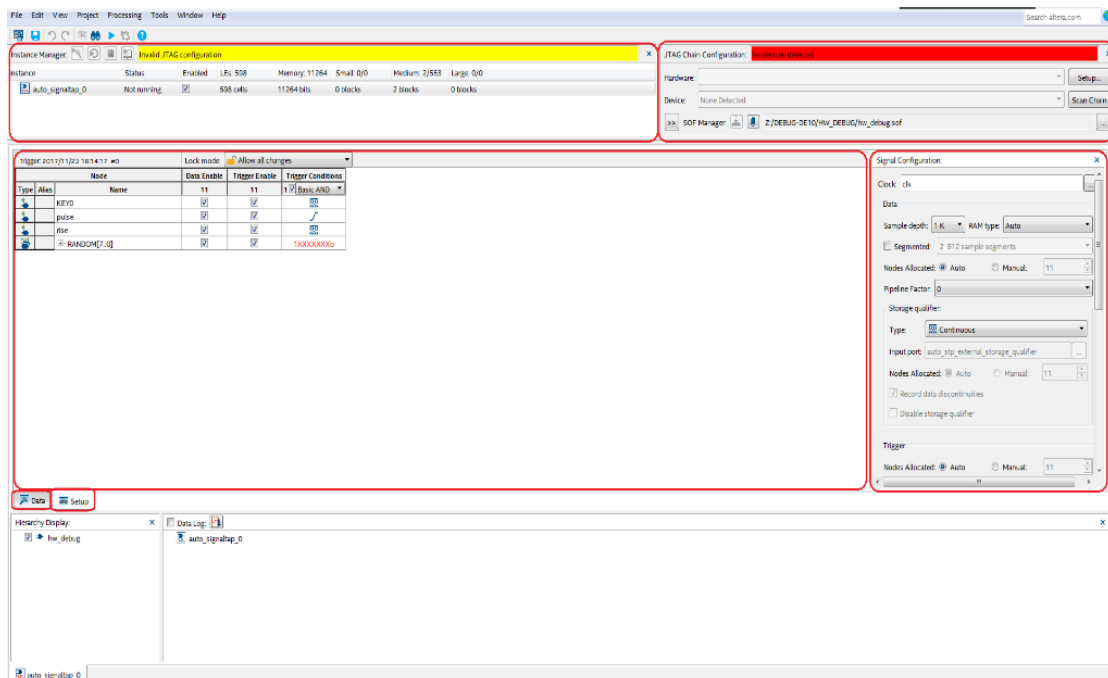


בצורת עבודה זו רוב הזיכרון יהיה מוקצה להקלטות שנעשו לפני ה - Trigger. ב - SignalTap קיימים שלושה מצב עבודה (Trigger Position) שדומים למה שתואר בציורים הנ"ל:

- ☐ מצב - Pre Trigger position (12%Pre trigger / 88% Post-Trigger)
- ☐ מצב - Center Trigger Position (50%Pre trigger / 50% Post-Trigger)
- ☐ מצב - Post Trigger position (88%Pre trigger / 12% Post-Trigger)

2.5 מבוא לשימוש ב - SignalTap

הקונפיגורציה של ה - SignalTap (ההדקים הנבדקים, אות השעון שמשמש לדגימה, מיקום ה - Trigger, מספר הדגימות ועוד ...) נעשים כולם באמצעות Quartus. מידע הזה מאוחסן בקובץ בעל סיומת *.stp.



להלן פירוט קצר על כל אחד מהאזורים המסומנים :

צד ימין עליון : הגדרת הכרטיס והפעלת הצורב.
צד ימין אמצעי : הגדרת השעון, עומק הזכרון ואופן הדגימה.
צד שמאל עליון : אפשרויות ההפעלה ומעקב הביצוע.
צד שמאל אמצעי : הגדרת האותות להקלטה ודברון.

לאחר ביצוע קונפיגורציה יש לקמפל את ה - SignalTap כך שהוא יהיה חלק אינטגרלי של התכן.



את קובץ הצריבה של התכן שכולל את ה - SignalTap מורידים לרכיב באמצעות הדקי ה - JTAG. (USB)



ההפעלה של ה - SignalTap, נעשית באמצעות Quartus, על ידי התפריטים :

Tools -> SignalTAP II Logic Analyzer

התקשורת עם ה - SignalTap נעשית באמצעות ההדקים של מערכת ה - JTAG. (USB)

בניסוי שתבצע במעבדה תכיר באופן מפורט ומעשי את השלבים שאותם יש לבצע על מנת שתוכל להשתמש ב - SignalTap.

3 ממשק למקלדת

3.1 כללי

3.1.1 מבוא ונתונים על ממשק למקלדת ברמת החמרה

בניסוי ניצור ממשק למקלדת PS/2. המקלדת מתחברת בדרך כלל למחשב (Host) או במקרה שלנו ללוח התרגול (DE2) באמצעות כבל הכולל ארבעה חוטים:

- ☐ חוט – VCC (שמגיע מספק מתח במחשב)
- ☐ חוט – GND
- ☐ חוט – שעון של המקלדת (נקרא לו KBD_CLK)
- ☐ חוט – מידע של המקלדת (נקרא לו KBD_DAT)

האות KBD_CLK מופק במקלדת ומשמש ככניסה של ה-Host. האות KBD_DAT כפי שהוא מוגדר בפרוטוקול PS/2 הוא אות דו כיווני שיכול להיות מופק במקלדת אך יכול להיות מופק גם על ידי ה-Host. בניסוי זה נמש על גבי לוח התרגול Host שמסוגל רק לקבל מידע מהמקלדת ואינו שולח חזרה מידע למקלדת. לכן במקרה שלנו האות KBD_DAT הוא אות שמופק במקלדת ומשמש ככניסה ללוח התרגול בלבד.

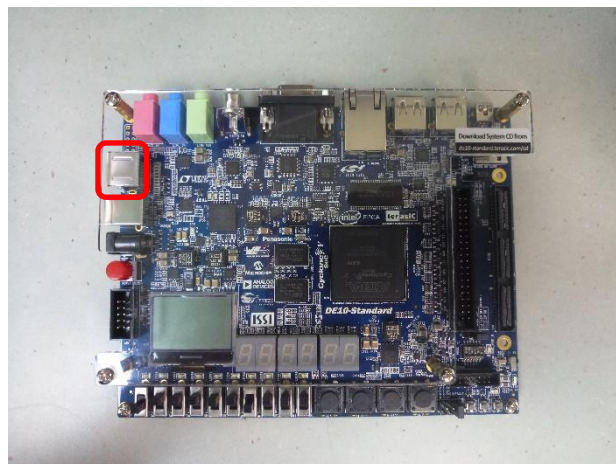
המחבר שבו נעשה שימוש הוא מחבר בעל שישה הדקים (6 pin mini-DIN PS/2 style connector) שנראה כך:



להלן תפקידי ההדקים השונים במחבר:

| Pin on connector | Function |
|------------------|---------------|
| 1 | KBD_DAT |
| 2 | Not Connected |
| 3 | GND |
| 4 | VCC |
| 5 | KBD_CLK |
| 6 | Not Connected |

מחבר זה נמצא בצדו השמאלי העליון של לוח התרגול DE2: ומגיע למחבר בחזית



להלן הקצאת ההדקים של הכניסות הנ"ל מנקודת מבט של רכיב ה- Cyclone – שעל לוח התרגול.

| Signal Name | Name on script | Pin on DE-10 |
|-------------|----------------|--------------|
| KBD_CLK | PS2_CLK | PIN_AB25 |
| KBD_DAT | PS2_DAT | PIN_AA25 |

<http://retired.beyondlogic.org/keyboard/keybrd.htm> ראה גם

3.1.2 מבוא ונתונים על ממשק בין מקלדת PS/2 ו Host ברמת ה-bit

פרוטוקול PS/2 שימושי במקלדות של מחשבי PC.

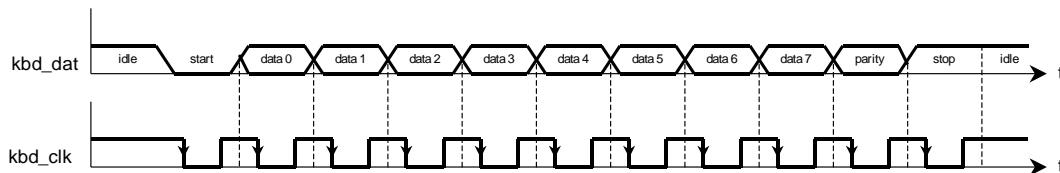
פרוטוקול התשדורת PS/2 הוא פרוטוקול סינכרוני וטורי כלומר מתבצעת העברה של המידע (באמצעות האות KBD_DAT) ביט אחרי ביט ובאופן מתואם לאות שעון (KBD_CLK).

במצב מנוחה, שבו אין תשדורת (Idle או Inactive), המצב הלוגי של KBD_DAT הוא 1'. לוגי. התשדורת מתחילה בסיבית התחלה (Start Bit) שהיא תמיד 0'. לאחר מכן משודר רצף של שמונה סיביות החל מסיבית ה- LSB עד לסיבית ה- MSB.

לאחר מכן משודרת סיבית הזוגיות (Parity), שצריכה ליצור מצב שבו מספר האחדים בתשדורת (כולל סיבית ה- Parity עצמה) הוא תמיד אי-זוגי (Odd Parity).

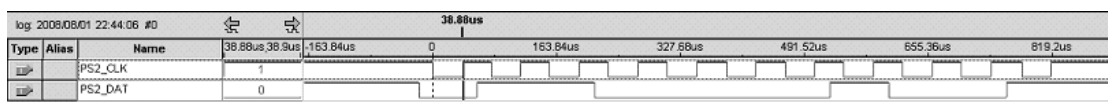
לאחר מכן משודרת סיבית הסיום (Stop Bit), שהיא תמיד 1'. בגמר פעולות אלו המקלדת חוזרת למצב מנוחה, עד לתשדורת הבאה.

האיור הבא מציג דיאגרמת זמנים שמתארת את צורת התשדורת של המידע יחד עם אות השעון של המקלדת.



שים לב שבדיאגרמת הזמנים הנ"ל החלפת המידע נעשית תמיד בזמן שאות השעון הוא 1'. לוגי. בזמן שאות השעון של המקלדת הוא 0', המידע תמיד יציב וניתן לדגום אותו בזמן זה. בתכן שבניסוי זה נדגום את המידע מיד לאחר הירידה באות השעון של המקלדת.

להלן דוגמה להקלטה של אותות ממקלדת אמיתית באמצעות ה- SignalTAP שבו נעסוק בחלק הבא של הניסוי.



3.1.3 גילוי שגיאה בעזרת סיבית ה- (odd)parity

התחנה המשדרת (למשל keyboard) מחשבת את סיבית הזוגיות (PB) באופן הבא:

$$PB_{transmit} = D0 \text{ xor } D1 \text{ xor } D2 \dots \text{ xor } D7$$

התחנה הקולטת בודקת את סיבית הזוגיות (PB) באופן הבא:

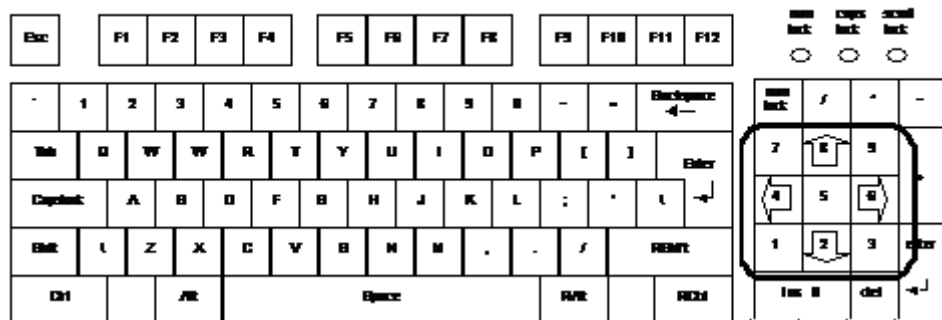
$$PB_{receive} = D0 \text{ xor } D1 \text{ xor } D2 \dots \text{ xor } D7$$

If (PB_{receive} = PB_{transmit}) then O.K.
else ERROR

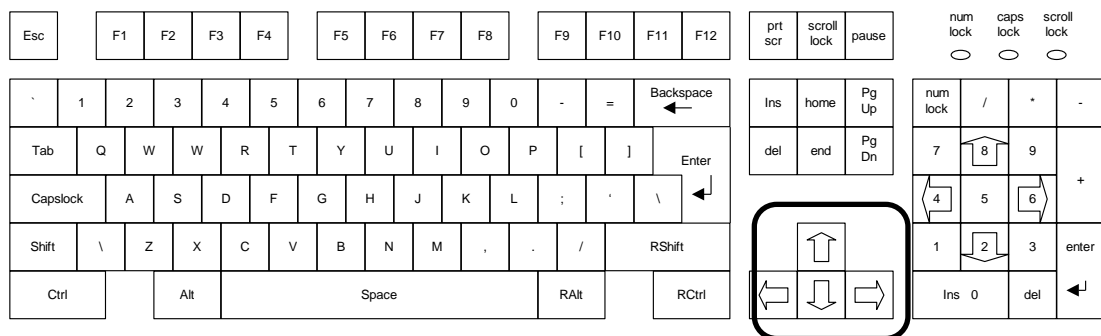
האלגוריתם הנ"ל מגלה שגיאה רק אם חל שינוי במספר אי-זוגי של סיביות ולא מגלה שגיאה אם חל שינוי במספר זוגי של סיביות. בכל מקרה, אין הוא יודע לתקן את השגיאה. קיימים אלגוריתמים מורכבים יותר שבהם מוסיפים סיביות נוספות למידע הנבדק ובעזרתן ניתן לגלות ולתקן מספר גדול יותר של שגיאות.

3.1.4 מבוא לתשדורת של המקלדת ברמת ה- byte

ברמת ה- byte, צריכים לקלוט את ה- bits של השידור ולפרש מצירופים אלו, מהו המקש שנלחץ או נעזב. כדי להדגים זאת, נשתמש למשל בדוגמה של מקשי החצים (Arrows) במקלדת. במקלות ישנות שהיו במחשבי PC ו-XT מקשי החצים נמצאו אך ורק בחלקה הימני של המקלדת.



במקלות מודרניות יותר, שהוכנסו לשימוש בתקופת מחשבי ה- AT, הוספו למקלדת גם מקשי חצים נוספים בחלק התחתון של המקלדת משמאל לקבוצת המקשים הקודמת.



הקשה למשל על מקש החץ שמורה כלפי מטה (Down Arrow), שנמצא באזור הימני שבמקלדת, מפיקה את ה- Byte הבא (בשיטת ייצוג הקסדצימלית):

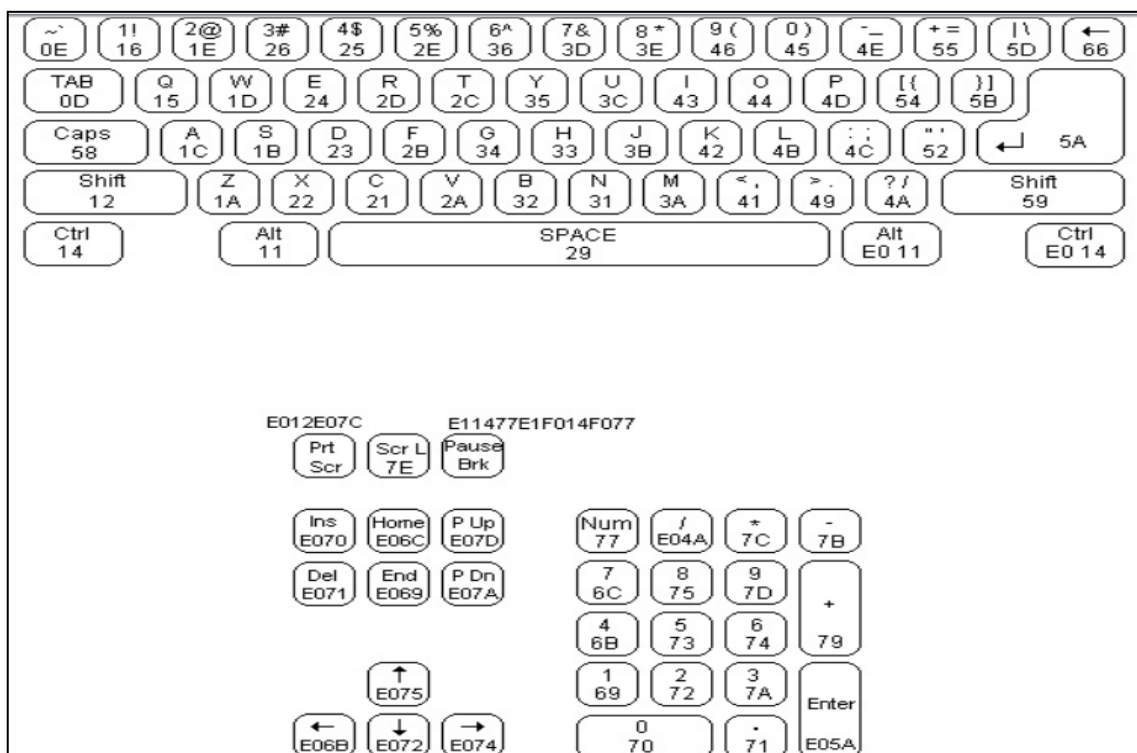
עזיבה של המקש גורמת לייצור byte נוסף שערכו הוא F0 ושמלווה ב – byte נוסף של המקש (72), כלומר מתווספים ל – byte המקורי הנ"ל של המקש שני bytes נוספים.

72 F0-72

ה – byte שערכו הוא F0, משמש תמיד כקידומת של שחרור המקש שנלחץ והוא נקרא בשם Release byte. לצירוף F0-72 בסוף התשדורת הנ"ל קוראים לפעמים גם בשם Break-Code (קוד עזיבה). ל – byte הראשון בתשדורת הנ"ל קוראים לפעמים גם בשם Make-Code (קוד לחיצה).

ערכי קוד של מקשים כדוגמת הערך 72, שהוא הערך של מקש ה – Down Arrow, יכולים לנוע בין הערכים 1 עד 84 עבור המקשים השונים של המקלדת. למשל הערך 75 הוא הערך של המקש Up- Arrow. לשאר הערכים (כמו F0 או E0 וערכים אחרים) יש משמעויות שונות.

להלן נתונים המקשים השונים והקודים שלהם במקלדת בה תשתמש במעבדה.



לחיצה ארוכה על מקש במקלדת, יוצרת פעולת Auto-Repeat, כלומר הקוד של המקש שנלחץ משוכפל מספר פעמים. לדוגמה לחיצה ארוכה על מקש Down Arrow, תגרום לקבלת רצף של כמה bytes שערכם הוא 72. להלן דוגמה לרצף שנוצר מלחיצה ארוכה של מקש זה ועזיבתו.

72 72 72 72 .. F0-72

כפי שאפשר לראות, גם בסופה של לחיצה ארוכה זו נוצר קוד שחרור עם הערכים F0-72.

גם מקש ה – Down Arrow המודרני יותר (שנמצא כאמור קודם בחלק התחתון של המקלדת משמאל לקבוצת המקשים הנ"ל), מפיק תוצאה מספרית 72, אך רצף ה – bytes שנוצר קצת שונה. להלן תוצאת לחיצה ארוכה על מקש זה:

E0-72 E0-72 E0-72 .. E0-F0-72

ה – byte שערכו הוא E0 הוא הקידומת של מקשים שמיצרים Extended-Codes (קוד מורחב). שים לב שהקידומת E0 מופיעה לפני כל הפעלה חוזרת של המקש (Auto Repeat) וגם לפני העזיבה (Release) של המקש.

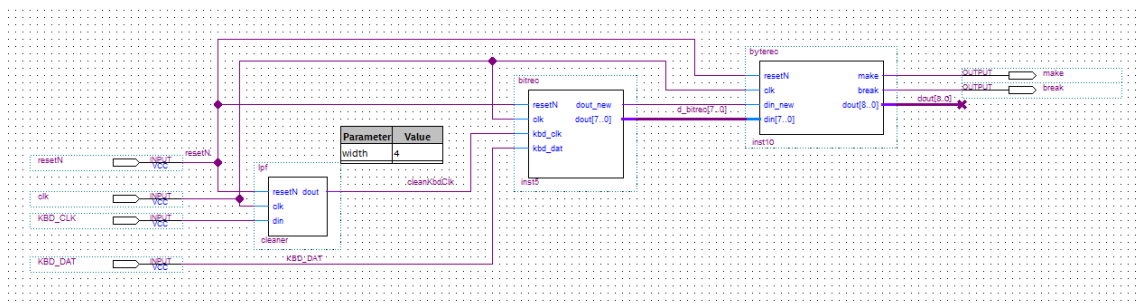
למשל, בהפעלת מקש ה- down arrow אחת מ-4 הסדרות הבאות אפשריות:

| Extended code | Normal code | לחיצה/שחרור |
|---------------|-------------|-------------|
| E0H, 72H | 72H | לחיצת מקש |
| E0H, F0H, 72H | F0H, 72H | שחרור מקש |

הקודים המורחבים אפשרו במשך השנים להוסיף למקלדות מקשים שונים. הוספה כזו נעשתה למשל במעבר ממקלדות XT למקלדות AT וכן כיום במקלדות מודרניות יותר שתומכות בפעולות שונות של Windows. כפי שראינו כבר קודם בדוגמה של מקש ה- Down-Arrow, במקלדות מודרניות, קיימים מקשים בעלי תפקידים זהים שנמצאים **בשני** מקומות שונים במקלדת. הקוד הבסיסי של רוב המקשים הכפולים הללו זהה אך אחד מהם הוא רגיל והשני הוא מורחב. דוגמאות אחרות פרט לארבעת סוגי מקשי החצים הם למשל מקשי הדפדוף (PgDn ו PgUp), מקשי ה- Enter, Ctrl, Alt, Delete, Insert, End, Home ו Enter. השימוש בקודים זהים (רגילים ומורחבים למקשים כפולים) עזר כמובן ליצרנים של ציוד במעבר ממקלדות XT למקלדות AT.

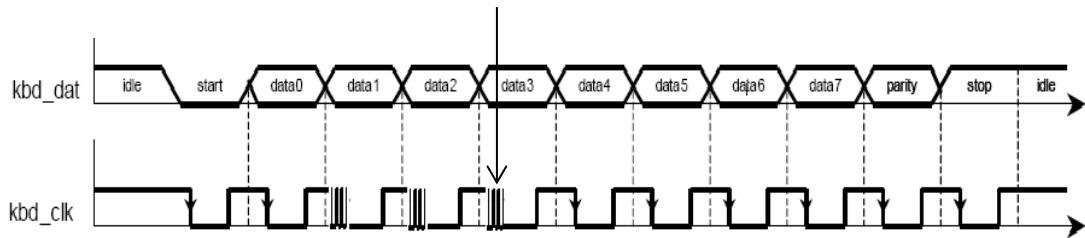
3.2 מימוש ממשק למקלדת

ניתן לממש ממשק חמרה למקלדת באופנים שונים. התכן הסינכרוני שבו בחרנו יאפשר תמיכה ברוב מקשי המקלדת, יהיה מאוד נוח לשימוש, יהיה גמיש וניתן להרחבה ליישומים רבים ויהיה גם קל להבנה. נפרק את התכן של ממשק המקלדת לשלושת החלקים הבאים.



הבלוק השמאלי שנקרא LPF (מסנן מעביר נמוכים), הוא פילטר שמסנן רעשים. אותות המידע והשעון (KBD_CLK ו KBD_DAT) שמגיעים ממקלדות מסוגים שונים, עלולים להיות לפעמים רועשים. מדובר בעלויות וירידות בלתי נקיות שעלולים להיווצר כתוצאה מ Hazards בחמרת היציאה של המקלדת או כתוצאה מהחזרות בחוט המקשר בין המקלדת ולוח התרגול. למרות שתופעות הללו לא תמיד קורות בכל מקלדת ולמרות שלעתים קרובות אפשר ליצור תכן ללא יחידת LPF, מומלץ בכל זאת שלא לקחת סיכון.

היות ובתכן שלנו אנו עומדים להסתמך על העלויות והירידות באות השעון של המקלדת (KBD_CLK), חשוב לסנן עלויות וירידות בלתי רצויות שנגרמות באות הרועש הזה, כמו בדוגמה להלן.



ה – LPF מעביר ליציאה שלו אך ורק אותות שהם עקביים לפחות מספר מסוים של מחזורי שעון. אין צורך לסנן באופן דומה את אות המידע של המקלדת (KBD_DAT), היות ואות זה ידגם בתכן שלנו באזורים שבהם הוא אינו משתנה (מיד לאחר הירידה של האות KBD_CLK).

היחידה שמוצגת במרכז האיור הנ"ל ושנקראת BITREC (מקלט ברמת ה – Bit), היא יחידה שקולטת את המידע הטורי שמגיע מהמקלדת לכניסה KBD_DAT. יחידה זו דוגמת את המידע הטורי מיד לאחר הירידות באות השעון הנקי של המקלדת (האות שמגיע מיחידת ה – LPF). יחידה זו גם ממירה את המידע הטורי למידע במקביל, בודקת את תקינותו (Start Bit, Parity ו Stop Bit) ורק אם המידע תקין, היא מעבירה אותו ליציאה dout. יחידה זו מייצרת גם אות חיווי שנקרא dout_new שמסמן הגעה של מידע חדש ליציאה dout.

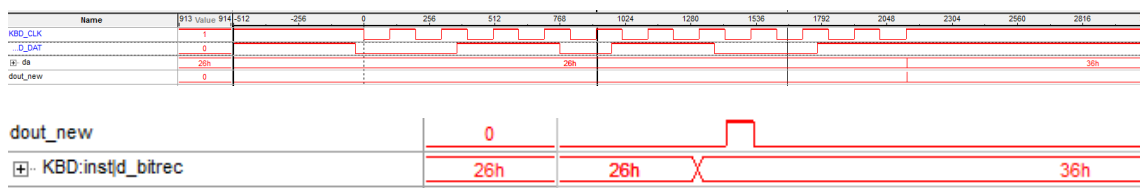
היחידה הימנית שבאיור ושנקראת BYTEREC (מקלט ברמת ה – Byte), מקבלת סדרה של כמה Bytes במקביל מהיחידה הקודמת. הסדרה שמתקבלת מפיקה ביציאה dout קוד שמראה איזה מקש נלחץ (Scan Code). כל הכנסה של כל byte ליחידה זו נעשית כאשר הכניסה din_new נמצאת ב – 1' לוגי. שים לב שהקוד המופק ביציאה dout הוא ברוחב של 9 סיביות. שמונת הסיביות הנמוכות מייצרות את ה – Scan-Code של המקש. הסיבית הגבוהה ביותר (MSB) מראה, האם מדובר במקשים רגילים או במקשים מורחבים (Extended). היחידה הנ"ל גם מפיקה חיווי האם המקש שהקוד שלו מופיע ביציאה **נלחץ** כרגע (make) או **נעזב** כרגע (break).

3.2.1 תכן של פילטר מעביר נמוכים

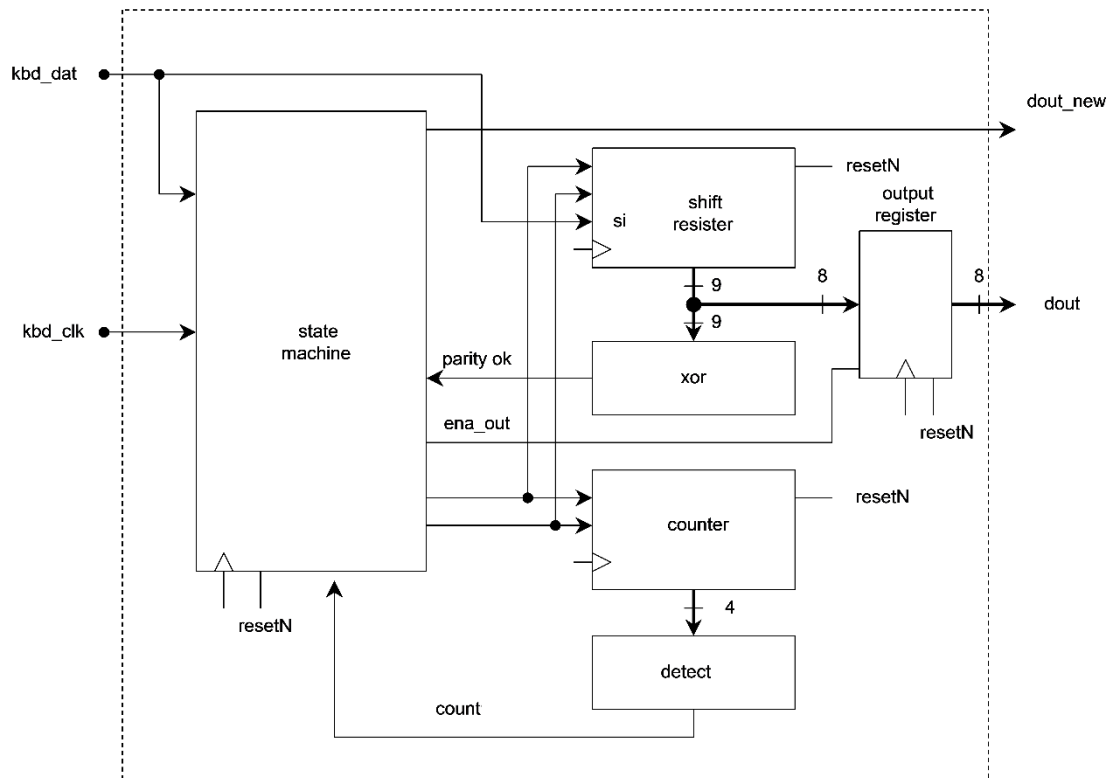
כאמור היחידה LPF מסננת את הרעשים מהקו של שעון המקלדת (KBD_CLK). ניתן לממש מסנן מסוג זה בהרבה דרכים. במודל מופיע קוד לדוגמה

3.2.2 תכן יחידת ה – BITREC

תפקידה של היחידה BITREC שמטפלת בתשדורת הטורית הוא להפיק מהמידע הטורי שמגיע לכניסות kbd_dat ו kbd_clk 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.



שים לב שהאות dout_new פעיל במשך מחזור שעון אחד **לאחר** שהמידע dout עודכן במידע חדש. האיור הבא מתאר את המבנה הפנימי של היחידה ברמת תהליכים עקרוניים (צירופיים וסינכרוניים).



המערכת ממומשת כמכונת VHDL אחת

פונקציונלית היא כוללת רגיסטר הזזה (shift register) ימינה ברוחב X סיביות, שתפקידו הוא לאסוף מאות הכניסה kbd_dat את סיביות המידע וסיבית ה - parity. הסיבית הנמוכה (LSB) של רגיסטר ההזזה מאחסנת בגמר קליטת המידע הטורי את הסיבית הראשונה של התשדורת. הסיבית הגבוהה (MSB) של רגיסטר ההזזה מאחסנת בגמר קליטת המידע הטורי את סיבית ה - Parity של התשדורת.

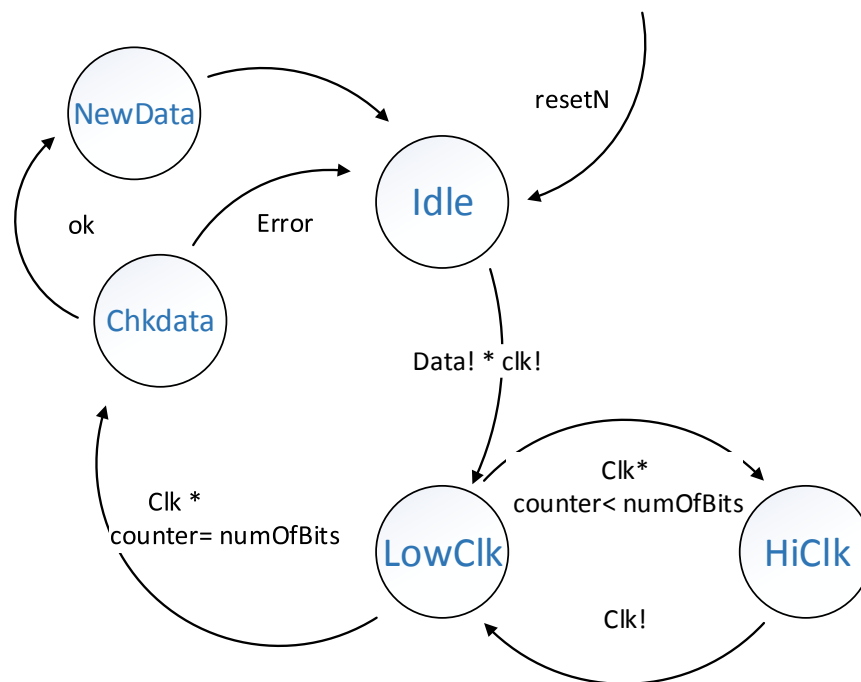
תשעת היציאות במקביל של רגיסטר ההזזה מזינות מערכת צירופית (שבנויה משער XOR) שקובעת האם ה - parity של התשדורת תקין. האות שמופק מיחידה צירופית זו נקרא parity_ok והוא מוזן חזרה לבקר (מכונת המצבים).

שמונה הסיביות הנמוכות של רגיסטר ההזזה (8 מתוך 9) מוזנות לרגיסטר היציאה. בניגוד לאופן פעולתו של רגיסטר ההזזה שהתוכן שלו משתנה במהלך התשדורת, תפקידו של רגיסטר היציאה הוא לאחסן את התוצאה הסופית הנקיה בלבד. רגיסטר זה הוא חלק ממכונת המצבים.

המערכת כוללת גם מונה ברוחב ארבע סיביות שסופר את מספר הסיביות שאוחסנו ברגיסטר ההזזה. היציאות של המונה מזינות את מכונת המצבים.

ליבה של המערכת היא כמובן מכונת מצבים (מסוג Moore) שמשמשת כבקר של היחידה. דיאגרמת המצבים הבאה מתארת את התנהגותה של מערכת זו.

מכונה זו עובדת על שעון 50MHz שהוא מהיר בהרבה משעון המקלדת. לכן הוספנו כניסת CLK_ENABLE שבה ניתן בעתיד להכניס שעון איטי- הדבר ישמש אותנו בזמן הDEBUG, כי גודל הזכרון של ה SIGNAL TAP קטן.



מכונת המצבים מוזנת גם מאות הכניסה kbd_dat ומפיקה את אות יציאה - dout_new.
בתרגיל ההכנה עליכם לחשב מהו NUM_OF_BITS

3.2.3 תכן יחידת ה - BYTEREC

היחידה שמטפלת בתשדורת ברמת ה - byte, קולטת כמה - bits מהיחידה הקודמת BITREC ומפרשת מצירופים אלו, מהו המקש שנלחץ או נעזב. כדי להדגים זאת, נשתמש למשל בדוגמה של מקשי החצים (Arrows) במקלדת כפי שהוסבר החומר הרקע.

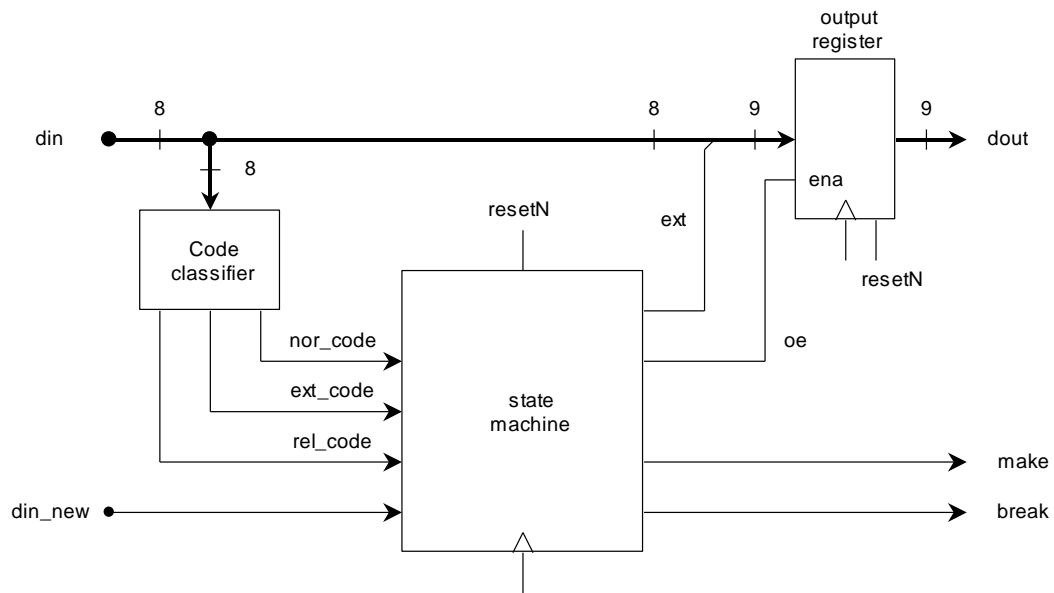
בבפרק הקודם תארנו את אופן פעולתם של מקשי המקלדת (הרגילים והמורחבים) ברמת ה - Byte. כל הקודים שתיארנו היו מבוססים על ערך של Byte בסיסי בודד (כדוגמת הערך 72). רוב מקשי המקלדת פועלים באופן כזה. קיימים כמה מקשים יוצאים מן הכלל כמו PrtSc או Pause שיש להם קודים ארוכים יותר. התכן שלנו לא עוצב לתמוך באופן ישיר במקשים יוצאים מן הכלל אלו.

הקוד שמתאר את יחידת ה BYTEREC הוכן עבורך והוא כולל את הקבצים הבאים:

- ☐ קובץ שמתאר מודל לסימולציה byterec.vhd (בשפת VHDL)
- ☐ קובץ של סמל גרפי לחיווט byterec.bsf

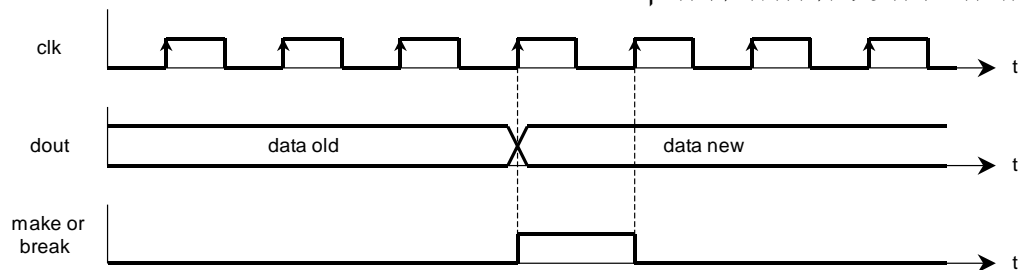
להלן הסבר קצר (תיעוד קצר) על פעולת רכיב ה - byterec שהוכן עבורך.

התכן הפנימי כוללת מערכת צירופית לזיהוי סוג הקוד הנכנס (Code Classifier), מכונת מצבים, ורגיסטר יציאה.



תפקיד ה- Code Classifier – לזהות האם הקוד הוא רגיל (כמו 72 בדוגמה שלנו) קידומת של קוד מורחב (E0) או קידומת של קוד עזיבה (F0). סוגי היציאה מופקים משלושת היציאות: `nor_code`, `ext_code` ו-`rel_code` בהתאמה. רגיסטר היציאה הוא ברוחב 9 סיביות ומורכב משמנה סיביות של אות הכניסה וסיבית גבוהה של אות `ext` (שהיא "1" לוגי כאשר מדובר בקוד מורחב). האפשרות של הרגיסטר נעשה באמצעות אות `oe`.

אות היציאה של יחידה זו נראה כך:



שים לב שהאותות `make` ו-`break` פעילים במשך מחזור שעות אחד מיד לאחר שהמידע `dout` עודכן במידע חדש.

4 מערכת הניסוי לבדיקת החומרה HW_DEBUG עם מכונת RANDOM

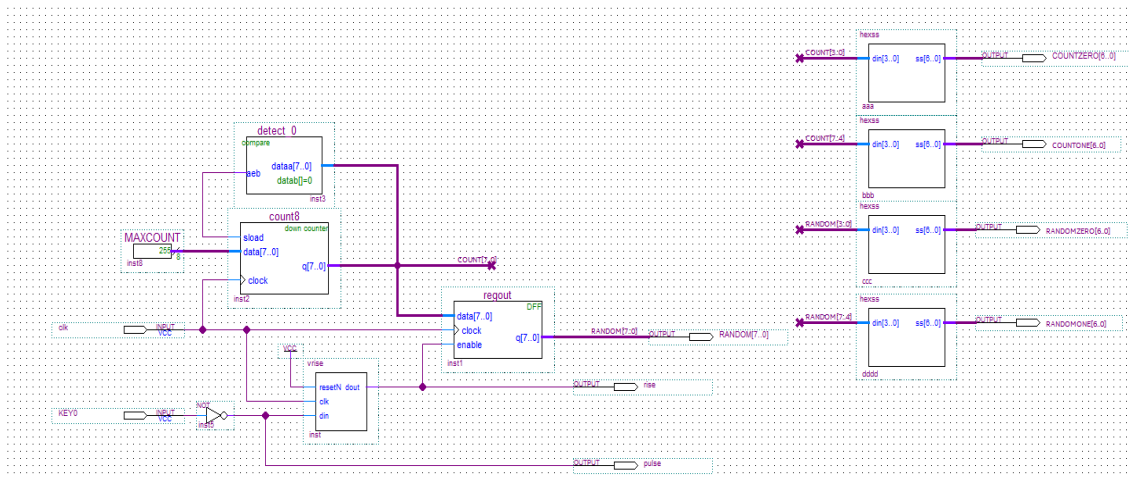
מטרה: במהלך המעבדה יהיה עליך להכין סביבת עבודה שתשמש בניסוי לבדיקת מערכות בחומרה. תבנה סביבת עבודה זו כמכונה ליצירת מספר אקראי RANDOM תוך שימוש בקבצים נתונים, כמסבר להלן.

מערכת ה-RANDOM כוללת מונה בינארי ברוחב שמונה סיביות שנקרא count8 (LPM_COUNTER). זהו מונה שסופר כלפי מטה ובכל פעם שהוא מגיע לספירה 0 המשווה DETECT_0 (LPM_COMPARE) גורם למונה לבצע טעינה סינכרונית במקביל של 255 שמגיע מקבוע MAXCOUNT (LPM_CONSTANT). כניסת השעון של המונה מחוברת לאות השעון הראשי של לוח ה-DE10 שפועל בתדר של 50 MHz.

היציאה של מונה זה נדגמת (Sampled) על ידי רגיסטר שנקרא regout (LPM_REGISTER). הדגימה מתאפשרת בכל פעם שהאות rise נמצא ב-'1' לוגי. מתי מתבצעת הדגימה? בכל פעם שלוחצים על המתג KEY0 בלוח DE10, נוצר פולס שלילי ארוך ביציאת המתג. פולס זה עובר היפוך באמצעות שער NOT כך שהאות שמתקבל ביציאה - pulse הוא פולס חיובי.

האות rise מופק (מיחידה בעלת שם vrise) באמצעות גזירה סינכרונית של העלייה באות pulse (או din), כך שבסופו של דבר, בכל פעם שלוחצים על המתג KEY0 הרגיסטר מאופשר במשך מחזור שעון אחד והרגיסטר מפיק צירוף אקראי כל שהוא ביציאות RANDOM[7..0] (שמחוברות לשמונה נוריות ה-LED הירוקות בלוח).

האיור הבא מתאר את המערכת.



היציאות RANDOM [7..0] הן בעצם יציאות שמפיקות מספרים אקראיים.

5 עריכת תוכן של זיכרון וקבועים בזמן אמת ISMCE

בחלקים הקודמים הצגנו את ה - SignalTap ואת יכולות הדיבוג החזקות שלו. כלי דיבוג חזק נוסף שבו משתמשים בדרך כלל כאשר מדבגים מערכת חמרה ושלעיתים קרובות הוא גם משלים את היכולות של ה - Signal Tap הוא ה - In System Contents Memory Editor או בקצור ISMCE. כלי זה מאפשר לבצע את הפעולות הבאות:

- ☐ להתבונן בזמן אמת בתוכן של זיכרונות וקבועים במערכת החמרה
- ☐ לשנות בזמן אמת תוכן של זיכרונות וקבועים במערכת החמרה

כל הפעולות הנ"ל נעשות בנוחות רבה באמצעות Quartus ומבלי שיהיה צורך לבצע קומפילציה מחודשת של הפרויקט.

כשם שאין צורך במכשור חיצוני יקר על מנת להפעיל את ה - SignalTap, גם ה - ISMCE אינו דורש מכשור וחמרה חיצוניים. לצורך הפעלתו משתמשים ב - Quartus שיוצר קשר עם משאבי החמרה הפנימיים ברכיבי המתוכנתת באמצעות שרשרת ה - JTAG.

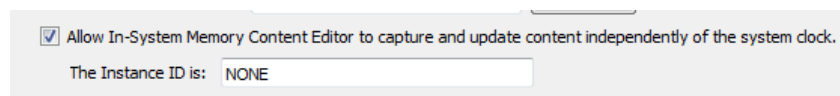
הרכיבים הבאים תומכים באופציה זו:

- ☐ קבוע - LPM_CONSTANT (נראה בניסוי)
- ☐ רכיב זיכרון ROM בעל Port בודד - כמו LPM_ROM
- ☐ רכיב זיכרון RAM בעל Port בודד - כמו LPM_RAM

את יכולת ה - ISMCE ניתן להוסיף לרכיבים מהסוגים הנ"ל באמצעות ה - MegaWizard או באופן ישיר באמצעות שימוש בפרמטר המיוחד שנקרא lpm_hint. תפקידו של פרמטר זה הוא להעביר ל - Quartus כמה מחרוזות תווים שמשמשים כפרמטרים מיוחדים. הטבלה הבאה שמתארת את הפרמטרים המיוחדים הללו, שמועברים כמחרוזות תווים לרכיב תחת lpm_hint:

| שם הפרמטר | משמעות הפרמטר |
|--------------------|--|
| ENABLE_RUNTIME_MOD | אפשרו שינוי ערך הזיכרון על ידי Quartus |
| INSTANCE_NAME | השם של הזיכרון הניתן לשינוי בתוך Quartus |

בכדי ש - ISMCE יופעל על רכיב ספציפי, יש לסמן את האפשרות הזו באשף של הרכיב, בזמן יצירת הרכיב או בזמן מאוחר יותר ע"י לחיצה כפולה עליו.



6 נספח – מימוש ישום ה-CapsLock ללא מכונת מצבים

```

entity caps_lock is
port ( resetN : in std_logic ;
      clk : in std_logic ;
      din : in std_logic_vector (8 downto 0);
      make : in std_logic ;
      break : in std_logic ;
      dout : out std_logic );
end caps_lock;

architecture behavior of caps_lock is
  signal pressed: std_logic;
  signal out_led: std_logic;
begin
  dout <= out_led;
  process ( resetN , clk)
  begin
    if resetN = '0' then
      out_led <= '0';
      pressed <= '0';

    elsif rising_edge(clk) then
      if (din = "001011000") and (make = '1') and (pressed = '0')
then
        pressed <= '1';
        out_led <= not(out_led);
      elsif (din = "001011000") and (break = '1') then
        pressed <= '0';
      end if;
    end if;
  end process;
end architecture;

```

הערה: קוד זה אינו תקני שכן יש בו קבועים "001011000" בגוף הקוד