

המעבדה ל – VLSI

מעבדה 2, 3

ארכיטקטורת VLSI להאצת מערכת לומדת ממומשת ב- System Verilog

מהדורה חדשה - הערות נא לשלוח ל-goel@ee

כל הערה תתקבל בברכה!

עדכון אחרון - 11: 57 04/08/2022

נכתב ע"י גואל סמואל

עודכן ע"י מיכאל שלפמן

<https://vlsi.eelabs.technion.ac.il/>

מסמך זה כתוב בלשון זכר ע"מ להקל על הכתיבה אך מתייחס לנשים ולגברים כאחד.
עמכם הסליחה .

הטכניון - מכון טכנולוגי לישראל

הפקולטה להנדסת חשמל



הנחיות בטיחות לסטודנטים במעבדות לאלקטרוניקה

כללי:

תמצית הנחיות בטיחות מובאת לידיעת הסטודנטים כאמצעי למניעת תאונות בעת ביצוע ניסויים ופעילות במעבדות לאלקטרוניקה של הפקולטה להנדסת חשמל. מטרתן להפנות תשומת לב לסיכונים הכרוכים בפעילויות המעבדה, כדי למנוע סבל לאדם ונזק לציוד. אנא קיראו הנחיות אלו בעיון ופעלו בהתאם להן.

מסגרת הבטיחות במעבדה:

- ♦ אין לקיים ניסויים במעבדה ללא קבלת ציון עובר בקורס הבטיחות.
- ♦ לפני התחלת הניסויים יש להתייבב בפני מדריך הניסוי לקבלת הנחיות בטיחות ותדריך ראשוני.
- ♦ אין לקיים ניסויים במעבדה ללא השגחת מדריך.
- ♦ מדריך הניסוי אחראי להסדרים בתחום פעילותכם במעבדה; הטו אוזן קשבת להוראותיו ונהגו על פיהן.

עשו ואל תעשו:

- ♦ יש לידע את המדריך על מצב מסוכן וליקויים במעבדה או בסביבתה הקרובה.
- ♦ לא תיעשה במזיד ובלי סיבה סבירה פעולה העלולה לסכן את הנוכחים במעבדה.
- ♦ אסור להשתמש לרעה בכל אמצעי או התקן שסופק או הותקן במעבדה.
- ♦ היאבקות, קטטה והשתטות אסורים. מעשי קונדס מעוררים לפעמים צחוק אך הם עלולים לגרום לתאונה.

- ♦ אין להשתמש בתוך המעבדה בסמים או במשקאות אלכוהוליים, או להיות תחת השפעתם.
- ♦ אין לעשן במעבדה ואין להכניס דברי מאכל או משקה.
- ♦ יש לכבות מכשירי טלפון ניידים לפני הכניסה למעבדה.
- ♦ בסיום הפעולות יש להשאיר את השולחן נקי ומסודר.

בטיחות חשמל:

- ♦ מדריך הניסוי עבר הכשרה בבטיחות חשמל והינו בעל תעודת חשמלאי בדרגה הנדרשת. היעזרו בו ובגורמים מקצועיים אחרים במעבדה, בעת חירום.
- ♦ בשולחנות המעבדה מותקנים בתי תקע ("שקעים") אשר ציוד המעבדה מוזן מהם. אין להפעיל ציוד המוזן מבית תקע פגום.
- ♦ אין להשתמש בציוד המוזן דרך פתילים ("כבלים גמישים") אשר הבידוד שלהם פגום או אשר התקע שלהם אינו מחוזק כראוי.
- ♦ אסור לתקן או לפרק ציוד חשמלי כולל החלפת נתיכים המותקנים בתוך הציוד; יש להשאיר זאת לטפול הגורם המוסמך.
- ♦ אין לגעת בלוח החשמל המרכזי, אלא בעת חירום וזאת לצורך ניתוק המפסק הראשי.

בטיחות אש, החייאה ועזרה ראשונה:

- ♦ מדריך הניסוי עבר הכשרה בבטיחות אש, החייאה ועזרה ראשונה. העזרו בו ובגורמים מקצועיים אחרים במעבדה, בעת חירום.
- ♦ במעבדה ממוקם מטף כיבוי אש ותיק עזרה ראשונה, זהו את מקומו.
- ♦ אין להפעיל את המטפים ואין להשתמש בציוד העזרה הראשונה, אלא בעת חירום ובמידה והמדריך וגורמים מקצועיים אחרים במעבדה אינם יכולים לפעול.

יציאות חירום:

- ♦ במעבדה ישנה פתח יציאה אחת והיא משמשת כפתח היציאה גם בשעת חירום.
- ♦ בארוע חירום הדורש פינוי, כגון שריפה, יש להתפנות מיד מהמעבדה.

דיווח בעת אירוע חירום:

- ♦ יש לדווח **מידית** למדריך ולצוות המעבדה.
- ♦ המדריך או איש מצוות המעבדה ידווחו מיידית לקצין הביטחון בטלפון; 2740, 2222, נייד; 050-544575. **במידה ואין הם יכולים** לעשות כך, ידווח אחד הסטודנטים לקצין הביטחון.
- ♦ לפי הוראת קצין הביטחון, או כאשר אין יכולת לדווח לקצין הביטחון, יש לדווח, לפי הצורך; משטרה 7-100, מגן דוד אדום 7-101, מכבי אש 7-102 וגורמי בטיחות ו/או ביטחון אחרים. בנוסף לכך יש לדווח ליחידת סגן המנמ"פ לענייני בטיחות; 3033, 2146/7.
- ♦ בהמשך, יש לדווח לאחראי משק ותחזוקה; 4776, 052-419917.
- ♦ לסיום, יש לדווח לאחראי האקדמי; 4661, לעוזר למנהל; 4678, לאחראי ההנדסי; 4668, 4671 ולאחראי האדמיניסטרטיבי; 3276.

תוכן העניינים

2	הנחיות בטיחות לסטודנטים במעבדות לאלקטרוניקה
2	כללי:
2	מסגרת הבטיחות במעבדה:
2	עשו ואל תעשו:
3	בטיחות חשמל:
3	בטיחות אש, החייאה ועזרה ראשונה:
4	יציאות חירום:
4	דיווח בעת אירוע חירום:
6	הקדמה
7	פרק 1 : מבוא קצר לשפת SystemVerilog
8	אינסטנסיאציה – instantiation (הצבה)
8	כתיבת testbench
8	משפט initial
9	דוגמא של מודול וה- testbench שלו
10	פרק 2 : הכנה
16	תכנון הארכיטקטורה
23	דו"ח מכין לניסוי מס' 1
32	דו"ח מכין לניסוי מס' 2
32	מימוש הבקר
35	קוד למימוש מכונת המצבים
38	בהמשך נכין קובץ לבדיקת המכונה
39	בדיקת המערכת השלמה
42	הוספת זיכרונות ה- SRAM
44	פרק 3 : ביצוע הניסוי
44	הערות כלליות לעריכת הדוח:
45	ביצוע ניסוי מס' 1
46	1. מימוש נירון הקונבולוציה
47	2. מימוש נירון ה- FC
48	3. מימוש יחידת ה- Pooling
49	4. סימולציה של יחידת הזיכרון
49	ביצוע ניסוי מס' 2
49	1. סימולציה של הבקר בעזרת Top1
50	2. סימולציה של הבקר ויחידות הזיכרון : Top2 . sv
51	3. הוספת נירונים של שלב הקונבולוציה (Top3) :
53	4. הוספת יחידות ה- Pooling לשלב הקונבולוציה (Top4) :
54	5. הוספת יחידות ה- FC (Top5) :

הקדמה

חוברת זו מהווה תדריך והכנה לניסוי "תכנון ארכיטקטורה למערכת לומדת ממומשת **SystemVerilog**" במעבדה ל- **VLSI**. הניסוי מתבצע על גבי תחנות **Linux** ותוכנות **Cadence** ו- **Synopsys** לתכנון מעגלי **VLSI**.

מטרת הניסוי: עם התקדמות הטכנולוגיה והדרישה לעבודה במהירויות גבוהות, קיימות אפליקציות רבות שדורשות ביצועים של זמן אמת. לעתים קרובות, הרצת האלגוריתם על מחשב כללי אינו מאפשר עבודה בזמן אמת. פתרון אפשרי הוא להיעזר במאיצים כגון **GPU** או **GPGPU**. יש מקרים רבים שגם פתרון זה אינו מספיק. במקרים אלה, הדרך לשיפור הביצועים (תדר עבודה, צריכת הספק, שטח או כל פרמטר אחר) היא לתכנן ולממש רכיב **VLSI** ייעודי להרצת האלגוריתם בחומרה. רכיב מסוג זה יכול לשפר זמן ריצה בשלושה סדרי גודל! הניסוי מתמקד בתכנון חומרה (מאיץ ייעודי) למערכת לומדת. הסטודנט ייחשף לארכיטקטורות השונות ולשיקולים השונים במימוש המאיץ. בסופו של דבר תמומש הארכיטקטורה המתאימה ביותר. היעדים העיקריים של הניסוי הם:

1. התנסות בהבנת ה- **trade-offs** בתהליך פיתוח **ASIC** כלומר מעגל **VLSI** ייעודי עבור אלגוריתם נתון.
2. העמקת הידע בשפת **SystemVerilog**.
3. התנסות בכלים לפיתוח מעגלי **VLSI** של חברת **Cadence**.

מבנה הניסוי:

הניסוי מורכב מ- 2 פגישות. כל פגישה אורכה ארבע שעות. לפני כל ניסוי יש להכין דו"ח מכין בפורמט **pdf**. יש ליצור קובץ **ZIP** (לא **RAR**) המכיל את דו"ח המכין ואת קובצי הקוד שהכנתם ולהעלות אותו ל- **LabAdmin** לפני תחילת הניסוי. ב- **LabAdmin** תוכל למצוא תבניות עבור כל הקבצים שעליך להכין.

חלק א' :

1. הכרה של המבנה הכללי של מערכת לומדת.
2. תכנון, מימוש וסימולציה של נוירונים שונים בשפת **SystemVerilog**.
3. התנסות בעבודה עם זיכרון **SRAM**.

חלק ב' :

1. הבנת ה- **trade-offs** הרבים בתכנון ארכיטקטורת **VLSI**.
2. תכנון, מימוש וסימולציה של בקר כמכונת מצבים.
3. תכנון, מימוש וסימולציה מערכת לומדת לזיהוי צורות.

דרישות הניסוי :

- קריאת חוברת הניסוי בעיון.
- הגשת דו"ח הכנה לניסוי + קוד לפי שאלות מפרק דו"ח הכנה.
- בוחן הכנה לניסוי.
- ביצוע הניסוי על תחנת עבודה.
- הגשת דו"ח סיכום שבועיים לאחר ביצוע חלק ב' של הניסוי.

דרישות דו"ח סיכום :

הגשת :

- דו"ח מכין לשני חלקי הניסוי.
- דו"ח סופי לשני החלקים עם תשובות לכל השאלות שנשאלות במהלך הניסוי.

"הסטודנט מתבקש למלא את טופס המשוב האלקטרוני הנמצא בקישור <http://www2.ee.technion.ac.il/Labs/EELabs/> , הטופס ממולא באופן אונימי. **אנו זקוקים לתגובותיכם על מנת לתקן ולשפר כמו גם לשבח**."

רקע

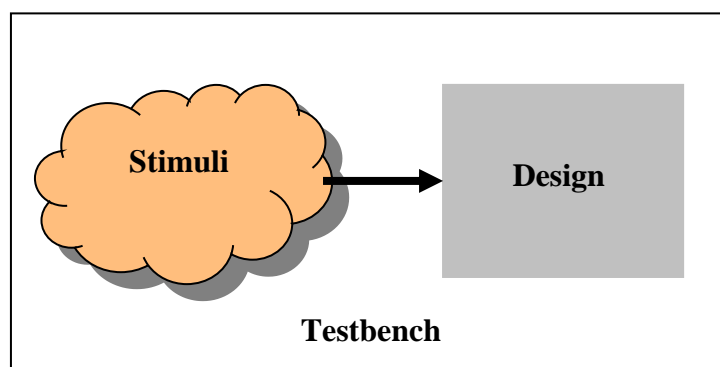
שפות תיאור החומרה (HDL-Hardware Description Language) למיניהן דוגמת **SystemVerilog** ו-**Verilog** פותחו על מנת לעזור למתכנתים לוגיים ליצור ולממש מעגלים במהירות.

כאמור שפת **SystemVerilog** היא שפה לתיאור חומרה שבעזרתה ניתן לתכנן, לממש ולבצע סימולציה של מערכות מורכבות. במגבלות מסוימות ניתן להפוך מימוש **SystemVerilog** למעגל חשמלי באמצעות כלי סינתזה. שפת **SystemVerilog** מאפשרת מימוש ברמת הפשטה גבוהה יותר לעומת **Verilog**. מבנים ופקודות כגון טיפוס **data** חדשים (**logic**, **int**), **enumerated types**, מערכים, משפטי **always** ייעודיים (**always_ff**, **always_comb**) מאפשרים מימוש בקוד יותר קומפקטי. בשפה גם ממשקים ישירים לשפות נוספות כגון **C**, **C++**, **systemC** ועוד.

פרק 1 : מבוא קצר לשפת SystemVerilog

בפרק זה נתאר רק את תכונות השפה שדרושות לביצוע הניסוי. המטרה העיקרית של העבודה עם שפת **HDL** היא מימוש התכנון וסביבת הסימולציה. הסברים נוספים ניתן למצוא ב-

<https://vlsi.eelabs.technion.ac.il/wp-content/uploads/sites/18/2018/05/System-Verilog.pdf>



איור מס' 1

בשפת **SystemVerilog** תכנון רכיב (מודול) ממומש באמצעות משפט **module** שניתן לתאר באופן הבא :

```
module name ( )
  interface // input and output definition
  variables //declare variables and type
  functional behaviour :
    // assign statements
    // always statements
    // instantiation of other module
endmodule
```

המודול מכיל את הגדרת הממשק (כניסות ויציאות), הגדרת משתנים ואת תיאור ההתנהגות הלוגית של הרכיב. ה-**testbench** (איור 1) הוא רכיב עזר ללא כניסות ויציאות המדמה את העולם החיצון של הרכיב הנבדק המהווה תת-רכיב שלו. לכן הוא מכיל הצבה של רכיב התכנון עצמו וקוד (**stimuli**) שיוצר את אותות הכניסה עבור התכנון. הסברים מפורטים יובאו בהמשך. לעתים (גם בניסוי זה) ליחידה **stimuli** בלבד קוראים **testbench**.

אינסטנציאציה – instantiation (הצבה)

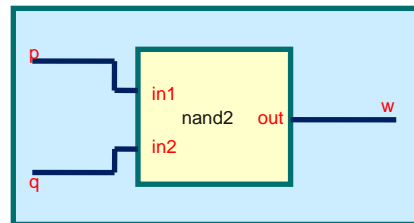
אינסטנציאציה היא הצבת עותק של מודול קיים וחיבורו ללוגיקה הנוספת במודול. לדוגמא שער **and** יכול להיות בנוי משער **nand** ומשער **not** המחוברים ביניהם. להלן דוגמא של משפט אינסטנציאציה:

```
nand nand_U1 (.out(w), .in1(p), .in2(q));
```

המשפט הנ"ל יוצר עותק של יחידה מסוג **nand** בשם **nand_U1** ומחבר את הכניסות והיציאות שלו (**out, in1, in2**) לקווים של היחידה המכילה אותו (**w, p, q**). התוספת **U1** הינה קיצור של **Unit-1** כיוון שעקרונית ניתן להציב יותר ממודול אחד מאותו סוג בתוך מודול אחר. לא חובה לציין את שמות הכניסות והיציאות של ה-**nand**, לדוגמא:

```
nand nand_U1 (w,p,q);
```

במקרה זה החיבור יתבצע לפי סדר רישום האותות, כלומר **w** יחובר לכניסה או ליציאה הראשונה בהגדרה של ה-**nand** וכן הלאה כפי שמופיע באיור הבא:



איור מס' 2

אם במודול שבו מבצעים את האינסטנציאציה מופיעים סיגנלים בשמות זהים לכניסות וליציאות של הרכיב (במקרה הזה **nand**) אז ניתן לבצע אינסטנציאציה מבלי לציין אף שם. לדוגמא:

```
nand U1 (.*) ;
```

ה ".*" מציין שה- pins של ה-**nand** יחוברו לרשתות (חוטים) בעלי שם זהה.

כתיבת testbench

לאחר מימוש מודול צריך לבדוק אותו באמצעות סימולציות. לשם כך יש להכין סביבה פשוטה המכונה **testbench**. הסביבה היא למעשה מודול שמכיל הצבה של המעגל הנבדק בתוספת קוד שיוצר ערכים שמשתנים עם הזמן עבור הכניסות. מקובל לממש את הקוד באמצעות משפטי **always** ו/או **initial** לפי נוחיות המשתמש.

משפט initial

בניגוד למשפט **always** שמתבצע שוב ושוב, משפט **initial** מתבצע רק פעם אחת עם תחילת הסימולציה. משפט **initial** מתחיל בתחילת הסימולציה ויכול להמשיך לאורך כל הסימולציה ועד סופה. בעזרת משפט זה ניתן לקבוע ערכים התחלתיים למשתנים. לדוגמא, בעזרת משפט **always** מחליפים את הערך של אות השעון כל מחזור ובמשפט **initial** קובעים את ערכו ההתחלתי:

ניתן לבצע בלוקי **initial** במקביל ורצוי שלפחות אחד הבלוקים יכיל את המילה **\$finish** אשר גורמת לסיום הסימולציה.


```
initial
begin
    clock = 0;
end
```

```
always
begin
    #5 clock = ~ clock;
end
```

- מסמן השהייה של מספר יחידות זמן. במקרה זה הכלי ימתין 5ns עד לביצוע ההשמה
~ - מסמן היפוך ערך הסיגנל

דוגמא של מודול וה- testbench שלו

```
module arbiter (clock, reset, req_0, req_1, gnt_0, gnt_1);
input clock, reset, req_0, req_1;
output logic gnt_0, gnt_1;
logic gnt_0, gnt_1;
always @ (posedge clock)
    if (reset) begin
        gnt_0 <= 0;
        gnt_1 <= 0;
    end else if (req_0) begin
        gnt_0 <= 1;
        gnt_1 <= 0;
    end else if (req_1) begin
        gnt_0 <= 0;
        gnt_1 <= 1;
    end
end
endmodule
```

משפט זה יוצר flip flop עבור כל משפט השמה.

להלן מימוש ה- testbench של הבורר :

```
module arbiter_test;
    logic clock, reset, req0, req1;
    logic gnt0, gnt1;
    initial
    begin
        reset = 0; clock = 0;
        req0 = 0; req1 = 0;
        #5 reset = 1;
        #15 reset = 0;
        @(posedge clk);
        req0 = 1;
        @(negedge clk);
        {req0, req1} = 2'b11;
        #10 $finish;
    end
```

קביעת ערכים התחלתיים

לאחר השהייה של 5ns

בעלית השעון הבאה...

בירידת השעון הבאה...

```
always
begin
    #5 clock = ~ clock;
end
```

יצירת שעון :

אינסטנציאציה של הבורר :

```
arbiter arb_U1 (.clock (clock), .reset (reset), .req_0 (req0), .req_1 (req1), .gnt_0 (gnt0), .gnt_1 (gnt1));
endmodule
```

חשוב : שים לב לשתי השיטות לקדם את הזמן ב- testbench :

א. #N – מציין שההשמה הבאה מתבצעת אחרי N ננו שניות.

ב. @(posedge clk) ו- @(negedge clk) מציין ההשמה הבאה מתבצעת מיד אחרי ירידת או עלית השעון הבאה.

פרק 2 : הכנה

בתחילת כל אחד משני המפגשים תדרשו להגיש את דו"ח ההכנה הרלוונטי לאותו מפגש כשהוא מכיל פתרונות לכל המטלות השונות – תשובות לשאלות, דיאגרמות וקטעי קוד. את קטעי הקוד יש להכין בנוסף בקבצים נפרדים לפי שמות המודולים השונים ולהעלות אותם ל- labadmin יחד עם דו"ח המכין בקובץ zip.

הערות כלליות :

- בכל **testbench** שמגדיר אות שעון, יש לאתחל את אות השעון בזמן 0 ואת יתר הסיגנלים בזמן 1.
- הקבצים שתבקשו ליצור הם קבצי טקסט רגילים אשר הסימט שלהם היא **.sv**. במקום **.txt**. על הדו"ח המכין להכיל בתוכו את כל קטעי הקוד שבקבצים.
- להכנת הקבצים בסעיף זה, רצוי להשתמש ב- **notepad** (או אפילו טוב יותר ב- **notepad++**) ולא ב- **Word**. לעתים קבצי **Word** מכילים תווים לא רצויים ב- **linux**. שמות הקבצים שעליך להכין מוגדרים בהמשך המסמך.
- יש לצרף את הקוד לדוח בצורה מסודרת ו"קריאה" או כקובצי טקסט נפרדים.
- אין צורך להכין קבצים נוספים. כל יתר הקבצים יינתנו במהלך הניסוי.
- נא להקפיד להגיש קוד מלא ולא רק חתיכות קוד.
- יש להגיש את הדו"ח בפורמט **pdf** ועם שם כמו **part1_shimshon_yovav.pdf**.

רקע כללי

מטרת הניסוי היא תכנון חומרה (מאיץ ייעודי) למערכת לומדת. מערכת לומדת עובדת בשתי פאזות, פאזת הלימוד ופאזת הפעולה (במקרה שלנו נקרא לשלב זה שלב הסיווג). במהלך הניסוי נעסוק בעיקר בשלב הסיווג. אנו נניח ששלב הלימוד התבצע מראש בצורה זו או אחרת. ראשית נתחיל בתיאור מערכת לומדת פשוטה. יחד עם זאת, מומלץ מאוד לצפות בסרטון :

A friendly introduction to Convolutional Neural Networks and Image Recognition

<https://www.youtube.com/watch?v=2-0l7ZB0MmU&t=160s>

המטרה של תיאור זה היא להבין את האלגוריתם של המערכת ובדיוק איזה חישוב עלינו לבצע וכל זה על מנת שנוכל לתכנן חומרה שתבצע את החישוב ביעילות מירבית.

בשלב ראשון נדרש לתכנן מערכת שתהיה מסוגלת לזהות שני תווים "/" ו- "\" (2X2 פיקסלים).

תמונה זאת תזוהה כ- "/"	-1	1
	1	-1
תמונה זאת תזוהה כ- "\"	1	-1
	-1	1

איור מס' 3

נערוך את התמונות בצורה קצת יותר נוחה, כלומר כמערך חד מימדי. נשתמש במערך מסוג זה כאשר נגיע למימוש המערכת.

תמונת ה- "/"

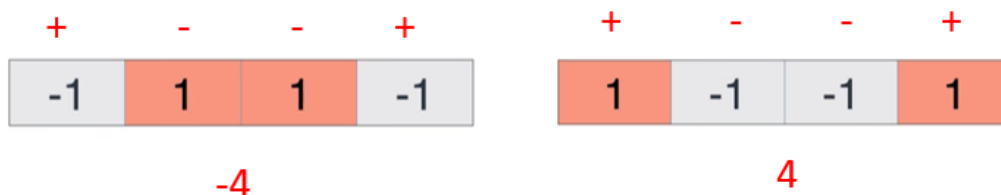
-1	1	1	-1
----	---	---	----

תמונת ה- "\"

1	-1	-1	1
---	----	----	---

איור מס' 4

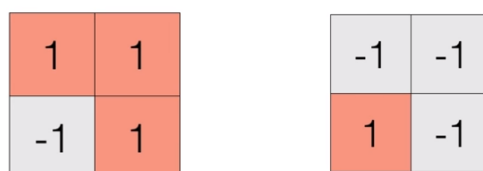
כעת צריך למצוא דרך שתאפשר למחשב להבחין בין שתי הצורות. אם למשל ננקוט בגישה של לחבר או להכפיל יחד את כל אברי המערך, לא נצליח להבדיל ביניהם כי בשני המקרים נקבל את אותה התוצאה. לעומת זאת, אם נחבר את שני האיברים הקיצוניים ונחסיר את שני האמצעיים נקבל את התוצאה הבאה:



איור מס' 5

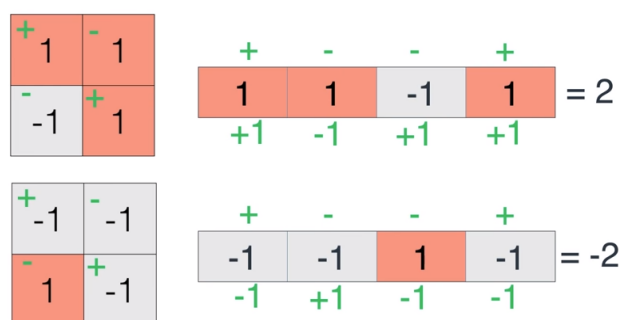
פעולה מסוג זה תאפשר למחשב להבחין בין הצורות. במילים אחרות, אם פעולה זאת מניבה מספר חיובי אז יש לזהות את הצורה כ- " / " אחרת פעולה זאת לא מזהה דבר. לתבנית הספציפית הזאת של '+' (פלוסים) ושל '-' (מינוסים) נקרא "מסנן" (במקרה זה מסנן 1). קל לראות שניתן בקלות לבנות תבנית/מסנן של '+' (פלוסים) ושל '-' (מינוסים) שבאמצעותה ניתן לזהות " \ " (נקרא בהמשך מסנן 2). מילה אחרת למסננים שנשתמש בה בהמשך התיאור היא kernels.

מה לגבי הצורות :



איור מס' 6

אם נפעיל על שתי הצורות את המסנן לזיהוי " \ " נקבל את התוצאה הבאה :



איור מס' 7

במקרה זה, המחשב יזהה את הצורה השמאלית באיור 6 כ- " \ " והמסנן לזיהוי " / " יזהה את הצורה הימנית באיור 6 כ- " / ". ניחוש לא רע בכלל! כלומר שיטה זאת בהחלט עושה רושם טוב. אבל כיצד מחשב יכול להגיע לשיטה המתוארת לבד? כלומר איך מוצאים את המסננים עבור כל מקרה ומקרה?

דרך אחת היא פשוט "ללמוד". כלומר אפשר לתכנת את המחשב לבדוק את כל האפשרויות ולספק למחשב משוב חיובי או שלילי לגבי כל אפשרות. בסוף התהליך המחשב בוחר במסננים הטובים ביותר. זה למעשה תהליך הלמידה. כאמור, במהלך הניסוי, אנו נתכנן מערכת רק לשלב הסיווג.

מסנן א' 1

מסנן א' 2

איור מס' 8 – כל המסננים האפשריים עבור 4 פיקסלים

בדוגמא פשוטה זאת, נדרשת בדיקה של 16 אפשרויות (2 בחזקת 4). קל לראות שעבור מערכות אמיתיות, מספר האפשרויות הוא עצום ומכאן הצורך במאיצים ממומשים בחומרה יעודית. כמובן שבנוסף לזה קיימות שיטות יעילות יותר למציאת המסווג (ראה סרטון). נחזור ונאמר כי ניסוי זה אינו עוסק בשלב הלמידה ומניח כי הלמידה הנדרשת כבר נעשתה מראש והמשקלים האופטימליים לכל מסנן ידועים, אבל עדיין תדרשנה פעולות כדי לטעון את המשקלים הידועים למערכת החומרה כדי שזו תוכל לבצע את פעולותיה.

כעת נסבך את הבעיה קצת. הפעם נרצה לזהות 4 צורות שונות ("X" , "O" , "\ " , "/") בתמונה של 3X3 פיקסלים כאשר כל אחת מהצורות האלו מורכבת מ- " / " ו- "\ " בלבד :

	\		X
	/		O

איור מס' 9

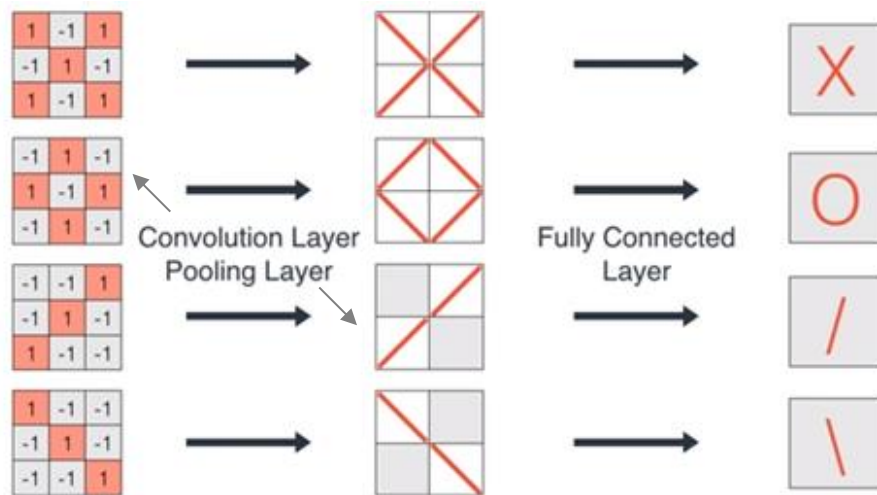
כיצד ניתן לבצע זאת בצורה יעילה? אפשרות אחת כמובן היא להשתמש בשיטה הקודמת, כלומר לנסות לבנות מסנן לתמונה השלמה :

--

איור מס' 10

מסתבר ששיטה זאת מאד לא יעילה ובין השאר מכיוון שקיימים לה 2048 מסננים (2 בחזקת 9). שיטה טובה בהרבה היא השיטה שמנסה לפרק את התמונה לתתי תמונות (של 2X2 פיקסלים) ולנסות לזהות איזו צורה קיימת בכל תת תמונה ולאחר מכן להגיע למסקנה לגבי התמונה השלמה.

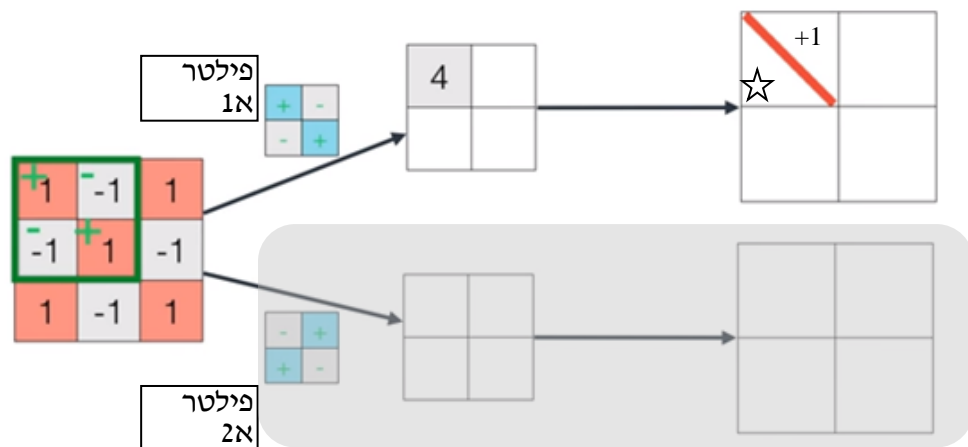
השיטה עובדת בצורה הבאה. כפי שמתואר באיור הבא, בשלב ראשון מנסים לזהות איזו צורה ("\" , "/" או כלום) קיימת בכל "איזור" (2X2) של התמונה הגדולה. בשלב שני מנסים לזהות האם בתמונה הגדולה מצויר "\" , "/" , "O" או "X".



איור מס' 11

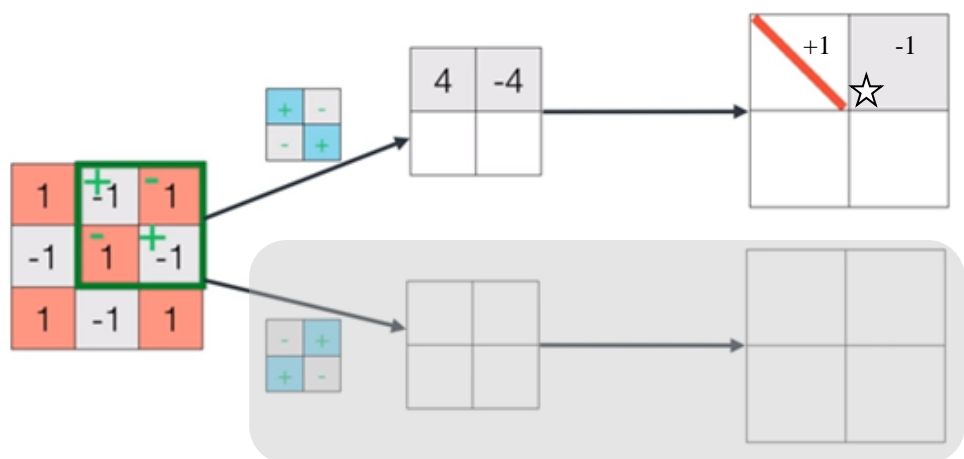
נתמקד בחלק הראשון המורכב משני שלבים, הראשון נקרא שלב הקונבולוציה והשני שלב **pooling** (קיצוץ נתוני ביניים לפי השגת ערך סף של זיהוי תת צורה בתת איזור בתמונה).

לזיהוי שתי צורות "\" ו- "/" דרושים שני פילטרים (מסננים). נתחיל עם הפילטר הראשון (1א). ממקמים את הפילטר באתר הראשון כפי שמופיע באיור מס' 12 במסגרת הירוקה. מבצעים את החישוב. התוצאה המתקבלת היא 4. מכיוון שזה מספר "גדול" (עבר ערך סף מסויים שקבענו) מחליטים שבאתר זה נמצא "\".



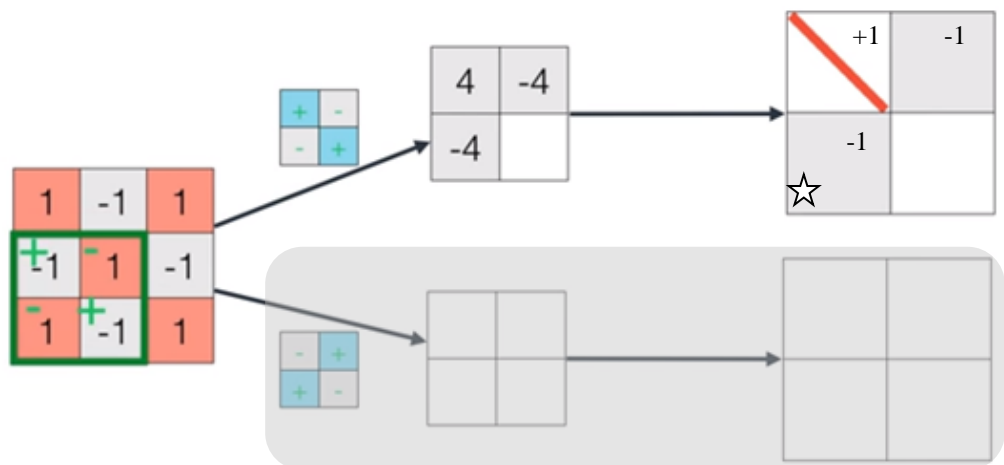
איור מס' 12

עוברים לאתר הבא. הפעם מתקבלת תוצאה -4 מכיוון שזה מספר "קטן" מחליטים שלא נמצא "\" במקום הזה :



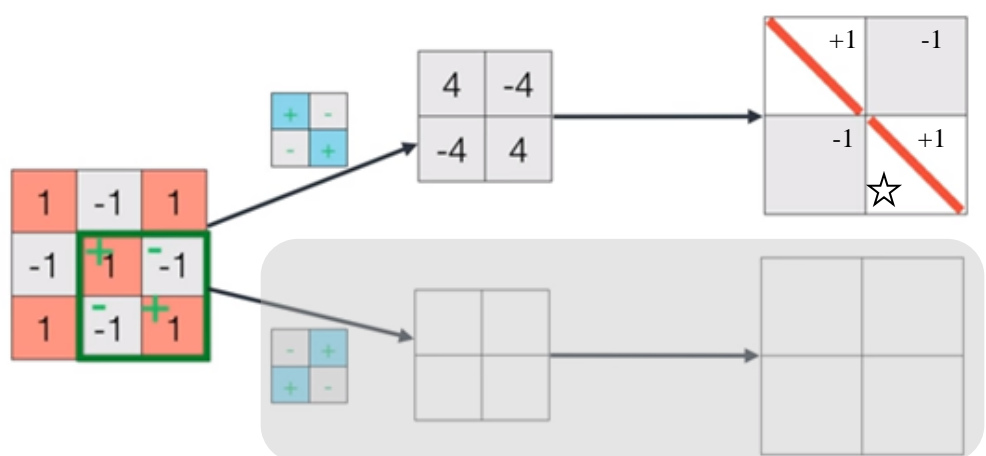
איור מס' 13

עוברים לאתר הבא. גם הפעם מתקבלת תוצאה -4 - לכן מחליטים שוב שאין כלום במקום הזה :



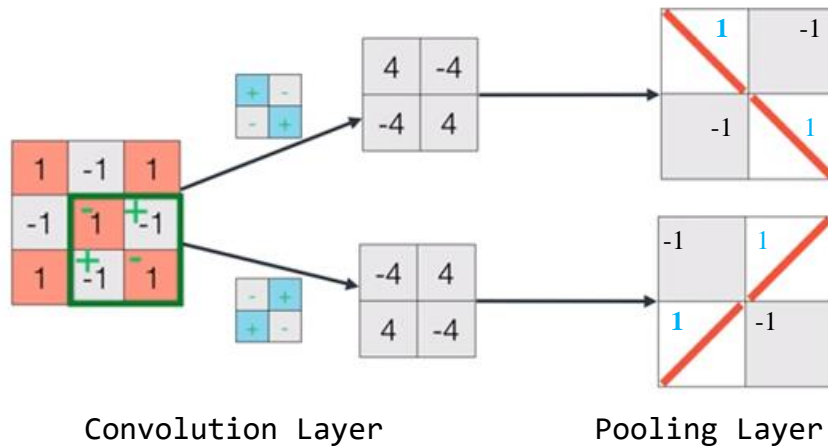
איור מס' 14

באתר האחרון שוב מתקבלת התוצאה 4 - ולכן מחליטים שבאתר זה נמצא " \ " .



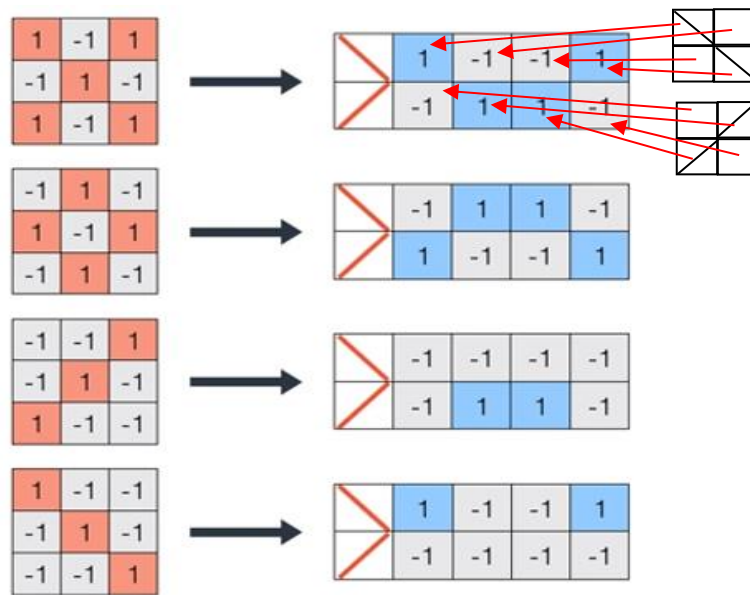
איור מס' 15

כעת חוזרים על כל התהליך עם המסנן השני (א2). מתקבלת התוצאה הבא :



איור מס' 16

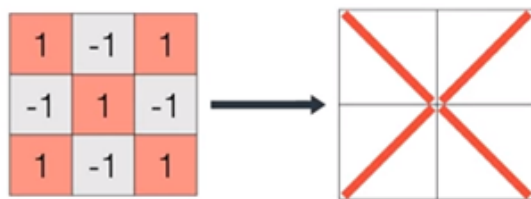
למעשה, לכל צורה ("\" או "/" - ו-), השלב הראשון מניב שתי תוצאות (או מטריצות) – אחת לכל מסנן שבאמצעותן ניתן להסיק לגבי הצורה שבכניסה. להלן סיכום שכל התוצאות שיכולות להתקבל עבור הצורות השונות כאשר שתי המטריצות (2x2 כל אחת) נפרסות לשתי שורות (4x1 כל אחת):



איור מס' 17

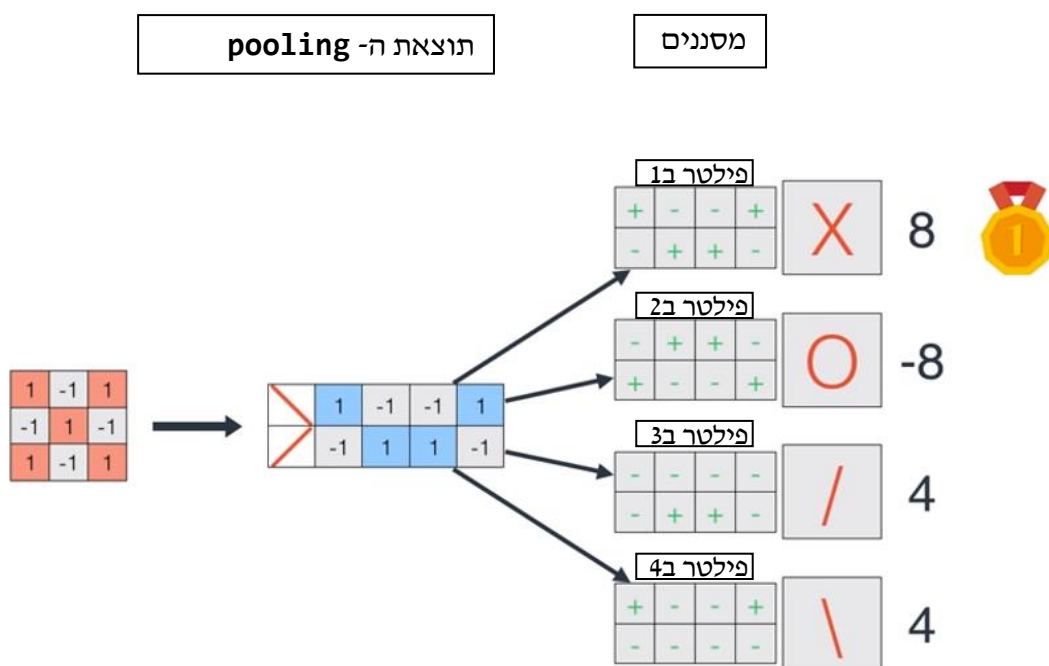
איור 17 מסכם מציאת "\" או "/" בכל אתר של התמונה הגדולה עבור כניסות שונות (השורה הראשונה מעידה על מציאת (1) או אי מציאת (-1) צורה "\" בכל אחד מארבעת האתרים ואילו השורה השנייה מעידה על מציאת (1) או אי מציאת (-1) צורה "/" בכל אחד מארבעת האתרים). זוהי למעשה תוצאת ה- **pooling layer**.

בשלב השני (**Fully Connected Layer**) המחשב משלב את שתי התוצאות הנ"ל על מנת להגיע למסקנה סופית לגבי הצורה שבתמונת הכניסה. בדוגמא הקודמת הצורה שבתמונה הכניסה היא "X". כיצד זה מתבצע ?



איור מס' 18

זהו התפקיד של ה- **Fully Connected Layer** שמכיל אוסף נוסף של (ארבעה) מסננים שבאמצעותם ניתן להחליט על איזו צורה מדובר.



איור מס' 19

כל תוצאה שמתקבלת משלב ה- **pooling** "נבחנת" על ידי ארבעה מסננים כפי שמתואר באיור 19. החישוב שמתבצע הוא סכום האיברים בתוצאת ה- **pooling** לפי הסימנים שבמסנן. מכיוון שהמסנן של ה- "X" מניב את התוצאה הגבוהה ביותר, מזהים את הכניסה כ- "X". גם כאן, תכנון המסננים נעשה בשלב הלימוד שלא נעסוק בו בניסוי זה (למעט הצורך בטעינת ערכי המסננים למערכת).

תכנון הארכיטקטורה

עד כאן האלגוריתם. הפוקוס של הניסוי הוא ללמוד כיצד מתכננים ארכיטקטורת **VLSI** יעילה של מאיץ לאלגוריתם זה. חיוני לבצע את ה- **trade-offs** הנכונים על מנת להגיע לתוצאה טובה. ראשית, חשוב לסכם את כל המשאבים הנדרשים לחישוב:

1. זיכרון לשמירת תמונת הכניסה
2. זיכרון לשמירת המסננים של שלב הקונבולוציה
3. זיכרון לשמירת המסננים (המשקלים) של שלב ה- **Fully Connected**
4. לשלב הקונבולוציה לכל אחד מ-2 המסננים (פילטר א1, א2 - איור 12) דרושות 4 פעולות (לכל חפיפה אפשרית של פילטר 2x2 עם מטריצת התמונה 3x3) מהסוג:

$$\sum_{i=1}^4 h_i x_i$$

כאשר x_i מייצג את הפיקסל ו- h_i את המשקל (הסימן +/-) בתא המתאים במסנן. נקרא ליחידה המבצעת חישוב מסוג זה נוירון.

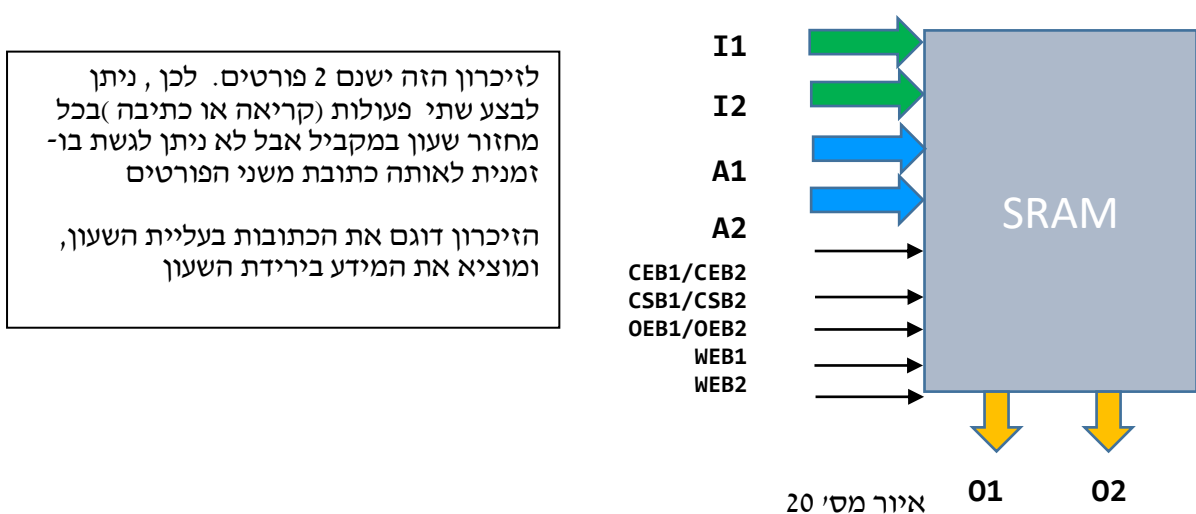
5. לשלב ה-Fully Connected לכל אחת מ-4 הצורות (x, o, /, \ - איור 17) דרושה פעולה מהסוג :

$$\sum_{i=1}^8 w_i x_i$$

כאשר כאן x_i מייצג את הערך המתקבל בכל אחד מהתאים בתוצאת ה-pooling ו- w_i את המשקל (הסימן +/-) בתא המתאים במסנן. כעת מתעוררות מספר שאלות לגבי המימוש :

1. באיזה דיוק (מס' סיביות) לייצג את כל הנתונים?
2. האם להשתמש בזיכרון SRAM או רגיסטרים לשמירת נתוני המסננים (המשקלים)? זיכרון ה-SRAM שברשותינו מאפשר גישה לשני תאי זיכרון בו זמנית (באמצעות שני פורטים), רגיסטרים לעומת זאת, לכמה שרוצים. זיכרון ה-SRAM חסכוני יותר בשטח כאשר מאחסנים יותר מ-1000 סיביות. זיכרון ה-SRAM חסכוני יותר בצריכת הספק. אם יוחלט על SRAM כמה יחידות SRAM דרושות? מה רוחב המילה האופטימלי עבור ה-SRAM? האם להשתמש באותה יחידת זיכרון לכל הנתונים הנ"ל? אפשר לחלק את ה-SRAM אבל לא יותר מידי.
3. האם לבצע את כל החישובים במחזור שעות אחד (חישוב מקבילי) או יותר (חישוב טורי/מצונר)? חישוב מקבילי ידרוש מחזור שעות ארוך יחסי. חישוב מצונר יאפשר שעות מהיר יותר אבל ידרוש יותר שטח (והספק).
4. בכמה נוירונים להשתמש בשלב הקונבולוציה? 1, 2 או 8 (2-מסננים כפול 4-פעולות)? בכמה נוירונים להשתמש בשלב Fully Connected? 1, 2 או 4 (4-צורות)? בכמה יחידות pooling להשתמש? 1, 2

תיאור ה-SRAM



I1 סיגנל DataIn לפורט 1 CEB1/CEB2 אות השעון לפורטים 1-2 פעיל בירידה

I2 סיגנל DataIn לפורט 2 CSB1/CSB2 אות chip select לפורטים 1-2 פעיל באפס
 A1 כתובת לפורט 1 OEB1/OEB2 אות output enable לפורטים 1-2 פעיל באפס
 A2 כתובת לפורט 2 01 סיגנל DataOut לפורט 1
 02 סיגנל DataOut לפורט 2
 WEB1/WEB2 – שליטה של קריאה/כתיבה

דרישות המתכנן :

ה- **trade-offs** השונים יתבצעו בהתאם לדרישות המשתמש. בתרגיל זה הדרישה היא לממש מערכת בעלת **throughput** גבוה ללא הגדלה משמעותית של השטח וצריכת ההספק.

כאמור, למערכת הזאת שתי פאזות של עבודה :

א. פאזת הלימוד – באופן כללי, שלב זה כולל חישוב ואחסון של המסננים האופטימאליים

לכל חלקי המערכת. בניסוי זה שלב הלימוד יכלול רק את אחסון נתוני המסננים בזיכרון.

כלומר אנו מניחים שנתוני המסננים אינם מצויים בזיכרון ה-SRAM מראש אלא במקור

אחר, ולכן לצורך כל סיווג של צורות חדשות יש ראשית לטעון אותם לזיכרון ה-SRAM.

ב. פאזת הסיווג – שלב זה כולל קבלת תמונה חדשה וביצוע הסיווג שלה לאחת מתוך 4

האפשרויות.

אין ספק שקיימות אפשרויות רבות למימוש מערכת שמסוגלת לבצע פעולות אלה. על מנת לצמצם קצת את האפשרויות נקבל מספר החלטות :

א. נחליט לייצג את כל הנתונים באמצעות 8 סיביות.

ב. במערכת המתוארת אין צורך בזיכרון גדול מכיוון שישנם רק שני מסננים בשלב

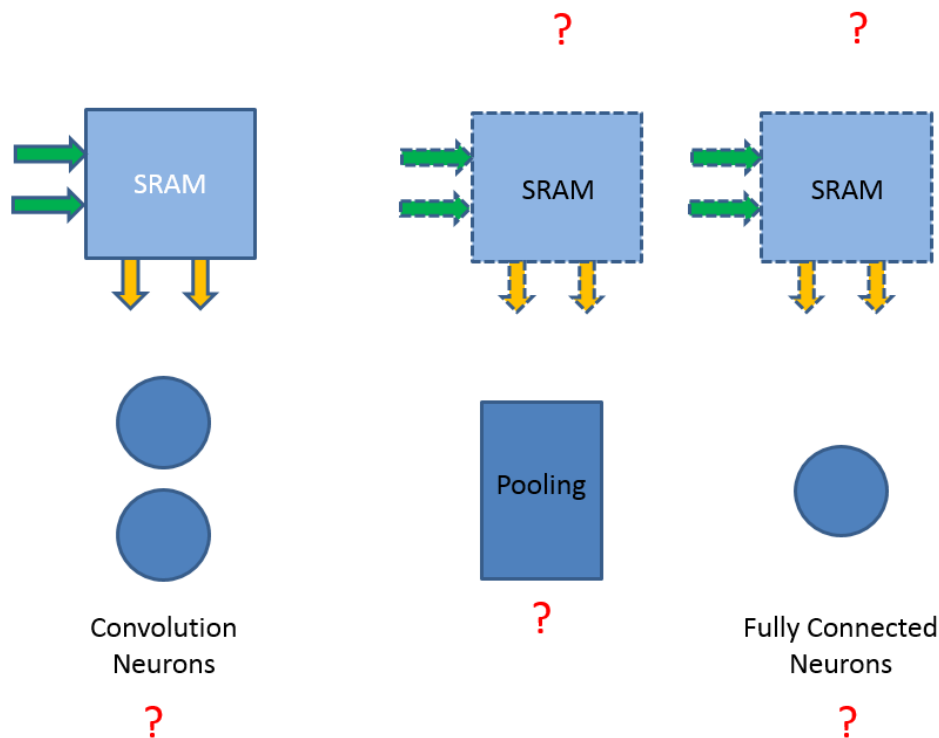
הקונבולוציה (א1, א2) וארבעה בשלב **fully connected** (ב1, ב2, ב3, ב4). ברור

שבמערכת אמיתית יהיו הרבה יותר נתונים ולכן מסיבה זאת ולצורך התרגיל הדידקטי

נחליט לשמור את המידע ב-SRAM ולא ברגיסטרים. נניח גם שהפיקסלים של התמונה

הנבחנת מגיעים מבחוץ ואין צורך לשמור את התמונה במערכת שלנו.

באופן כללי, מבנה המערכת יהיה כפי שמוצג באיור מס' 21 :



איור מס' 21

אנו נרצה לטעון את המשקלים h_i עבור הנוירונים בשלב הקונבולוציה, נרצה לשמור את תוצאת ה-Pooling בזיכרון ונרצה גם לטעון את המשקלים w_i עבור הנוירון(ים) שבשלב ה-Fully Connected.

נגדיר אוסף של שיקולים שיעזרו לנו להחליט על התצורה המתאימה ביותר לצרכים שלנו. המטרה כעת היא למצוא תשובות לשאלות הבאות :

1. מהו המספר האופטימלי של יחידות SRAM? מה רוחב הזיכרון, כלומר, כמה סיביות בכל שורה?
2. בכמה נוירונים להשתמש בשכבת הקונבולוציה ?
3. בכמה נוירונים להשתמש בשכבת ה-Fully Connected ?
4. האם יחידת ה-Pooling תכיל רגיסטרים שיאפשרו חלוקת החישוב לשני שלבי pipeline או האם היא תהיה יחידה קומבינטורית טהורה שתגרום לכל החישוב להתבצע במקביל ?

תכנון מבנה הזיכרון

תכנון הזיכרון גם מאד קשור לצורת המימוש של שכבות הקונבולוציה וה-Fully Connected (FC). ככל שמשתמשים ביותר נוירונים בשכבות אלה יש צורך בהבאת יותר נתונים (משקלי מסננים) מהזיכרון. הרחבת המילים בזיכרון תאפשר קריאה או כתיבה של יותר מידע בכל גישה לזיכרון. (המשקלים הם האיברים שמרכיבים את המסננים).

1. פאזת הלימוד (Learn) דורשת :
 - a. כתיבה (write) של **8 מילים** (4 לכל מסנן) בעלות 8 סיביות כ"א עבור שכבת הקונבולוציה לתוך ה-SRAM.
 - b. כתיבה (write) של 8 מילים בעלות 8 סיביות כ"א עבור כל צורה ששכבת ה-FC תנסה לזהות (4 צורות) לתוך ה-SRAM. כלומר **32 מילים** (במקרה שלנו) בעלות 8 סיביות. השאיפה היא לקצר שלב זה ככל שניתן.
2. פאזת הסיווג (Classify) דורשת :
 - a. אם מממשים את שלב הקונבולוציה בעזרת נוירון בודד אז נדרשת קריאה (read) של 4 משקלים (מסנן אחד) בזמן החישוב מתוך ה-SRAM.
 - b. אם מממשים את שלב הקונבולוציה בעזרת שני נוירונים אז נדרשת קריאה (read) של 8 משקלים בזמן החישוב מתוך ה-SRAM.
 - c. אם מממשים את שלב ה-FC בעזרת נוירון בודד דרוש לקרוא 8 משקלים בעלי 8 סיביות כ"א בכל מחזור שיעון. נוירון זה יפעל במשך 4 מחזורים ויקבל את המשקלים של צורה שונה בכל מחזור.
 - d. אם מממשים את שלב ה-FC בעזרת 4 נוירונים דרוש לקרוא 32 משקלים בעלי 8 סיביות כ"א בכל רגע נתון. נוירונים אלה יפעלו במשך מחזור אחד ויקבלו את המשקלים של כל צורה וצורה בו זמנית.

הערה :

- זכרו את המספרים שהתקבלו לעיל (**8 ו-32 מילים בעלות 8 סיביות**) עבור ההמשך.

השלב של הקונבולוציה דורש מעבר על איזורים רבים של תמונת הכניסה עבור כל מסנן. לכן נבחר להשתמש בנוירון אחד לכל מסנן. שלב זה לוקח מספר מחזורים רבים יחסית ולכן שלב ה-FC יספיק לבצע את כל החישובים גם אם יש בו רק נוירון אחד. לכן נחליט לעבוד עם התצורה הבאה :

2 Convolution Neurons, 1 Fully Connected (FC) Neuron

בתצורה זאת :

- שני הנוירונים בשכבת הקונבולוציה יכולים לעבוד במקביל כ"א עם משקלים שונים.
- כל נוירון בוחן 4 תת אזורים של תמונת הכניסה ולכן דרושים 4 מחזורים להשלמת החישוב.
- שכבת ה-FC לא יכולה להתחיל לעבוד עד להשלמת חישוב הקונבולוציה ולכן יחידת ה-pooling חייבת להכיל רגיסטרים לשמירת כל תוצאות הקונבולוציה.
- סביר להשתמש בשתי יחידות pooling – אחת לכל נוירון שבשכבת הקונבולוציה.

- שכבת ה-FC חייבת לבדוק את תוצאת הקונבולוציה מול 4 צורות. במצב כזה, רצוי לבצע זאת עם נזירון בודד המבצע את החישוב במהלך 4 מחזורי שעות. זה יוצר איזון בין שני חלקי החישוב (Convolution+Pooling vs Fully Connected).
 - בעבודה אופטימלית שתי השכבות עובדות כל הזמן ומבצעות את החישוב ב- 4 מחזורי שעות. כאשר שכבת ה-FC פועלת על תמונה מסוימת, שכבת הקונבולוציה פועלת על התמונה הבאה.
 - כמוכן שכל זה רק בתנאי שכל המשקלים הדרושים לחישוב זמינים!
- נבחן כעת מספר אופציות לגבי מבנה הזיכרון עבור מבנה זה:

חלופה A

One SRAM unit (8 bits wide) in complete system

במקרה הזה, יש לנו רק שני ערוצים (ל-RAM יש שני פורטים) ברוחב של 8 סיביות לביצוע פעולות קריאה וכתיבת של 8 + 32 מילים בעלי 8 סיביות כ"א. לשם כך דרושים 20 + 20 מחזורים.

חלופה B

Two SRAM units (32 bits wide) in system, - one for FC, one for convolution

במקרה זה יש לנו 4 ערוצים בעלי רוחב של 32 סיביות לביצוע פעולות קריאה וכתיבת של 8 + 32 מילים בעלי 8 סיביות כ"א. פעם ניתן לכתוב/לקרוא פי 4 מילים בו זמנית בכל ערוץ וגם ניתן לכתוב/לקרוא לשני ה-RAM ים במקביל, ולכן יידרשו הרבה פחות מחזורים לבצע פעולות אלו.

חלופה C

Three SRAM units (32 bits wide) in system, - two for FC, one for convolution

על מנת לצמצם את זמני הכתיבה של 32 המילים ל-RAM ים של ה-FC אפשר להוסיף יחידת זיכרון נוספת. עם שני RAM ים אפשר לכתוב 32 מילים שני מחזורים בלבד. הכתיבה ל-RAM של convolution מתבצעת במקביל. בכל מחזור classify, יחידת ה-convolution וגם יחידת ה-FC זקוקות 8 מילים כ"א. אפשר לקרוא את כל המידע הזה במחזור בודד.

	# bytes written	# cycles for write	# bytes read	# cycles for read
FC Layer - learn (write)	32	2		
FC Layer - classify (read)			8	1
Convolution Layer - learn (write)	8	1		
Convolution Layer - classify (read)			8	1
Total clock cycles required – learn/classify		2		1

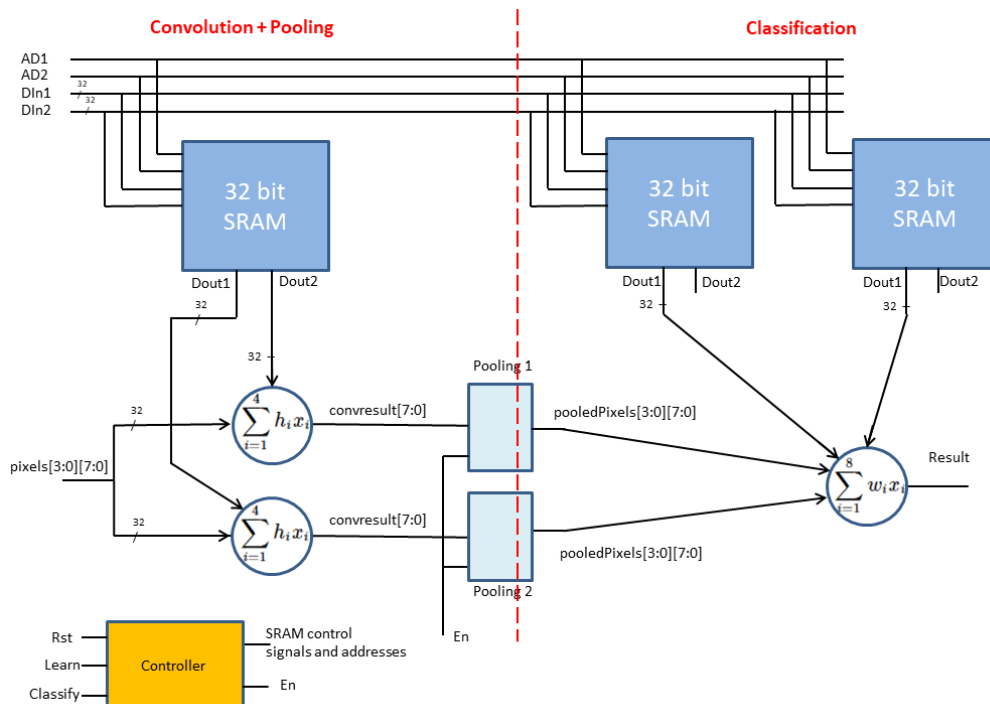
טבלה מס' 1

שאלות: למחשבה

- 1.1 כמה מחזורים דרושים עד שמוזגים את הצורה הראשונה? הסבר!
- 1.2 כמה מחזורים נוספים דרושים עד שמוזגים את הצורה השנייה והשלישית? הסבר!

הארכיטקטורה שמתוארת באיור מס' 22 דורשת את מספר מחזורי השעות הקטן ביותר ולכן נבחר לממש את האופציה הזאת.

הוחלט גם שלא נטפל בשמירה של תמונת הכניסה בזיכרון **SRAM**. התמונה תוגדר ב- **testbench** הכללי וכל תת אזור יועבר למערכת דרך הכניסה **pixels** (4 פיקסלים בני 8 סיביות).



איור מס' 22

- פירוט מלא של כל סיגנלי הבקרה שמפיק הבקר יובא בהמשך.
- במחזור הראשון נבצע **reset** של הבקר.
- במחזור השני והשלישי נבצע טעינה של המסננים והמשקלים בכל הזיכרונות כפי שתואר לעיל. זה למעשה משלים את תהליך הלימוד.
- מיד במחזור הבא נתחיל בשלב הסיווג. במשך 4 מחזורים מגיעים דרך כניסת ה- **pixels** 4 תתי האזורים של התמונה הראשונה אחד אחרי השני.
- כניסת ה- **pixels** מגיעה לשני הנורונים והתוצאות שכל נורון מפיק מגיעות ליחידת- **pooling** שלו ונאגרות במהלך 4 מחזורי שעון.
- עם סיום 4 המחזורים הראשונים, יחידת הקונבולוציה יכולה להתחיל לטפל בתמונה הבאה ובמקביל יחידת ה- **FC** תבצע את הסיווג הסופי לתמונה הראשונה תוך כדי שימוש בתוצאות ה- **pooling** השמורות.
- תיאור מבנה כל היחידות יובא בהמשך.

ייצוג הנתונים :

תמונת הכניסה :

כל הפיקסלים בתמונה יקבלו ערך של 1 או 1- במילה בת 8 סיביות. דוגמא של תמונה שמייצגת את הצורה "X" :

```
InputImage[2] = {8'h01,8'hff,8'h01};
InputImage[1] = {8'hff,8'h01,8'hff};
InputImage[0] = {8'h01,8'hff,8'h01};
```

מסננים של פעולת הקונבולוציה

עבור "\":

1	-1
-1	1

נניח את (-1) כ- ff ואת (+1) כ- 01 באופן הבא :

KR_DATA_I2 = 32'h01ffff01 ;

עבור "/":

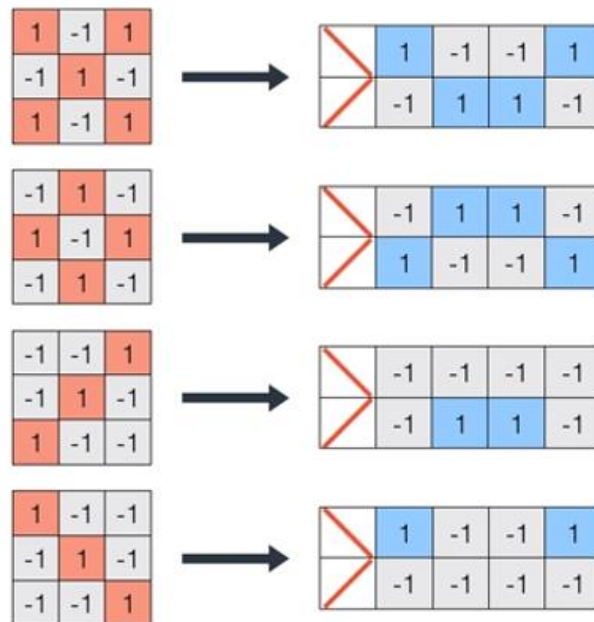
-1	1
1	-1

נקבל באופן דומה :

KR_DATA_I1 = 32'hff0101ff ;

משקלים של יחידת ה-FC

המשקלים של יחידת ה-FC בעלי ערכים (+1) או (-1) שנייצג עם 1 או ff בהתאם. קיימים 8 משקלים לכל צורה. כזכור :



במחזור הראשון נזין את שני הסטים הראשונים :

W1_DATA_I1 = 32'h01ffff01 ;

W2_DATA_I1 = 32'hff0101ff ;

W1_DATA_I2 = 32'hff0101ff ;

W2_DATA_I2 = 32'h01ffff01 ;

ובמחזור השני נזין את שני הסטים הנותרים :

W1_DATA_I1 = 32'hffffffff ;

W2_DATA_I1 = 32'hff0101ff ;

W1_DATA_I2 = 32'h01ffff01 ;

W2_DATA_I2 = 32'hffffffff ;

חשוב : על מנת להקל על הבדיקה :

- יש להגיש את דו"חות המכין כמסמך WORD ולא בכתב יד.
- יש לרשום את השאלה לפני התשובה שלך.

דו"ח מכין לניסוי מס' 1

שאלות רקע

עד כה נתקלתם במספר שאלות למחשבה.
ענו עליהן כעת לצורך הדוח המכין :

חלופה A - One SRAM unit (8 bits wide) in complete system

באיור מס' 22 ניתן לראות שעבור **חלופה C**, כל הערכים של המסננים והמשקלים מגיעים ל- **Convolution Neurons** - ל- **FC Neurons** ביחד באותו מחזור שעון.

- 1.1 האם ניתן לעשות זאת גם עבור חלופה A ? הסבר!
- 1.2 מה עלינו להוסיף לתכנון על מנת לאפשר עבודה עם SRAM בודד בעל רוחב BUS של 8 סיביות?
- 1.3 בנוסף לתוספת שטח האם יש "עלות" נוספת בעבודה עם SRAM בודד ? הסבר.

חלופה B

Two SRAM units (32 bits wide) in system, - one for FC, one for convolution

באיור מס' 22 ניתן לראות שעבור **חלופה C**, כל הערכים של המסננים והמשקלים מגיעים ל- **Convolution Neurons** - ל- **FC Neurons** ביחד באותו מחזור שעון.

- 1.4 האם ניתן לעשות זאת גם עבור חלופה B ? הסבר!
- 1.5 מה עלינו להוסיף לתכנון על מנת לאפשר עבודה עם שתי יחידות SRAM עם בעלי רוחב BUS של 32 סיביות?
- 1.6 בנוסף לתוספת שטח האם יש "עלות" נוספת בעבודה עם SRAM בודד ? הסבר.

חלופה C

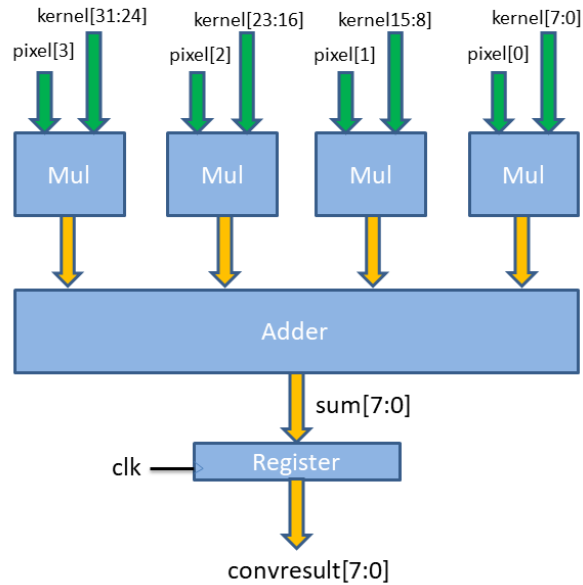
- 1.7 כמה מחזורים דרושים למערכת עד שהיא מזהה את הצורה הראשונה ? הסבר!
- 1.8 כמה מחזורים נוספים דרושים עד שמזהים את הצורה השנייה והשלישית ? הסבר!

בחלק הראשון של הניסוי נממש ונבדוק את כל תתי הבלוקים שדרושים לממש את כל המערכת :

- נזכרון הקונבולוציה
- נזכרון ה- **Fully Connected (FC)**
- יחידת ה- **Pooling**
- זיכרון ה- **SRAM**

מימוש נזכרון הקונבולוציה

באיור הבא מתואר המבנה של הנזכרון בשכבת הקונבולוציה :



1.9 רשום את הקוד הבא בקובץ בשם **cneuron.sv** והשלם את קטע הקוד לחישוב **sum**.
אין להשתמש בלולאת FOR.

```
`timescale 1ns/100fs
module CNeuron (input clk,
  input [31:0] kernel,
  input [3:0] [7:0] pixels,
  output logic [7:0] convResult);

  logic [7:0] sum;
  integer i;
  always_comb
  begin
    // Code for computing sum
  end
  always_ff @(posedge clk)
  begin
    convResult <= sum;
  end
endmodule
```

כאמור, ה- **testbench** זה הקובץ שמכיל את סביבת הסימולציה. קובץ זה מכיל את הקוד שיוצר את אותות הכניסה למעגל וגם אינסטנציאציה של היחידות השונות (כלומר המעגל בבדיקה) שים לב שה- **testbench** בנוי כך שהכניסות משתנות בירידת השעון. למה זה חשוב ?

1.10 רשום קובץ **testbench** בשם **cneuron_test.sv** לבדיקת המימוש. עליך להעתיק את המימוש החלקי המופיע בהמשך ולהוסיף :

a. 2 ערכים שונים עבור **kernel** (32'h01ffff01 ו- 32'hff0101ff)
 4 ערכים שונים עבור **pixels** לכל ערך של **kernel** כאשר הערכים משתנים כל 10ns.
 השתמש בערכים הבאים עבור ה- **pixels** :

```
{8'h01,8'hff,8'hff,8'h01}
{8'hff,8'h01,8'h01,8'hff}
{8'h01,8'h01,8'h01,8'h01}
{8'hff,8'hff,8'hff,8'hff}
```

הערה : ה- **kernel** מכיל 4 הערכים של מסנן הקונבולוציה.

מומלץ להשתמש ב: `@posedge` או `@negedge` לתזמן את משפטי ההשמה. לדוגמא:
`kernel = 32'h01ffff01;`
`pixels = {8'h01,8'hff,8'hff,8'h01} ;`
 - מוסיפים משפט `$finish` כדי להימנע מ-`testbench` אשר אינו מפסיק לעולם.

```
`timescale 1ns/100fs

module cneuron_test;

logic clk;
logic [3:0] [7:0] pixels;
logic [7:0] convResult;
logic [31:0] kernel;

CNeuron I_CNeuron(.clk(clk), .kernel(kernel), .pixels(pixels),
.convResult(convResult));

initial
begin
clk = 1'b0 ;
@negedge clk; kernel = 32'h01fefe01;
@negedge clk; pixels = {8'h01, 8'hff, 8'hff, 8'h01};
// Set remaining values for kernel and pixels
#20 $finish;
end

always
begin
#5 clk = ~clk;
end

endmodule
```

Note how values are assigned on the negedge of the clk. As the clock is 10n and it starts low, the first `@negedge` means at 10n. The second `@negedge clk` means after another 10n => at 20n

מימוש נוירון ה-FC

נוירון כמעט זהה לקודם. ההבדל היחיד הוא כמות הפיקסלים ומשקלים.

1.12 רשום את הקוד הבא בקובץ בשם `fcneuron.sv` והשלם את קטע הקוד לחישוב `sum`.
 שים לב שהפעם במקום `kernel` עם 4 ערכים, יש כאן `weight` שמכיל 8 משקלים.
שוב אין להשתמש בלולאת FOR.

```
`timescale 1ns/100fs

module FCNeuron ( input clk,
input [7:0][7:0] pooledPixelArray,
input [63:0] weight,
output logic [7:0] result);

logic [7:0] sum;
integer i;
```

```

always_comb
begin
    // Code for computing sum
end
always_ff @(posedge clk)
begin
    result <= sum;
end
endmodule

```

1.13 רשום קובץ **testbench** בשם **fcneuron_test.sv** לבדיקת המימוש. עליך להעתיק את המימוש החלקי המופיע בהמשך ולהוסיף: **weight** 4 ערכים שונים עבור

X

```

weight[63:32] = 32'h01ffff01 ;
weight[31: 0] = 32'hff0101ff ;

```

0

```

weight[63:32] = 32'hff0101ff ;
weight[31: 0] = 32'h01ffff01 ;

```

/

```

weight[63:32] = 32'hffffffff ;
weight[31: 0] = 32'hff0101ff ;

```

```

weight[63:32] = 32'h01ffff01 ;
weight[31: 0] = 32'hffffffff ;

```

- 4 ערכים שונים עבור **pooledPixelArray** (כפי שאתה מצפה לקבל לכל צורה).
לדוגמא:

```

pooledPixelArray[0] = 32'h01ffff01 ;

```

- משפט **\$finish** כדי להימנע מ-**testbench** אשר אינו מפסיק לעולם.

```

`timescale 1ns/100fs
`define NoOfKernels 2

module fcneuron_test;

logic clk;
logic [3:0] [7:0] pooledPixelArray[`NoOfKernels-1:0];
logic [63:0] weight;
logic [7:0] result;

FCNeuron I_FCNeuron(.clk(clk),

.pooledPixelArray({pooledPixelArray[0],pooledPixelArray[1]}),
                    .weight(weight),
                    .result(result) );

```

```

initial begin
clk = 1'b0 ;
end

always
begin
//weight for X
@(negedge clk);
//weight for 0
@(negedge clk);
//weight for /
@(negedge clk);
//weight for \
@(negedge clk);
end

initial begin
// pooledPixelFormat X
repeat(4) @(negedge clk);
// pooledPixelFormat '\'
repeat(4) @(negedge clk);
// pooledPixelFormat "0"
repeat(4) @(negedge clk);
// pooledPixelFormat '/'
repeat(4) @(negedge clk);
#20 $finish;
end

always
begin
#5 clk = ~clk;
end
endmodule

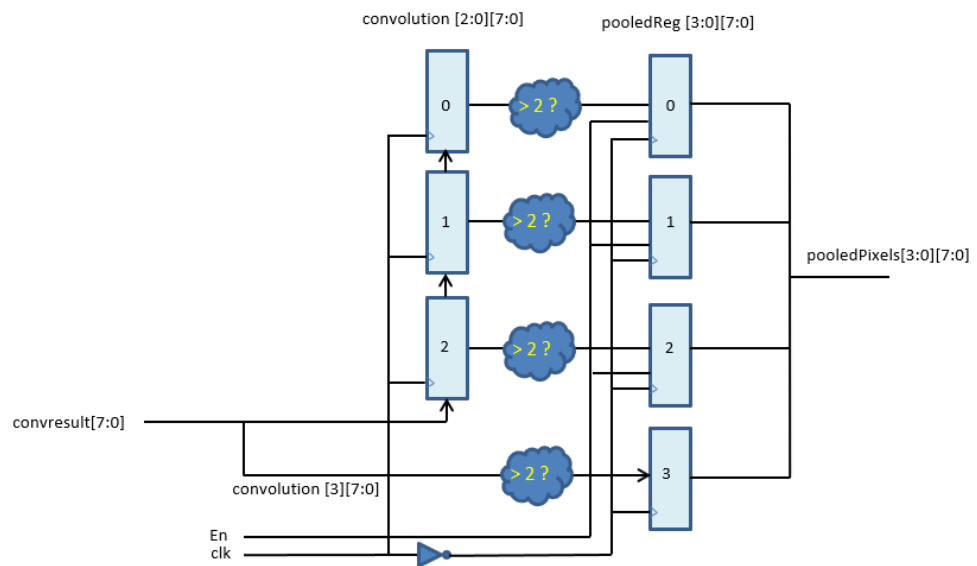
```

חשוב : נזכרנו כי הסיווג מקבל 8 ערכים של `pooledPixels` וכל יחידת `pooling` מייצרת 4 ולכן ב- `testbench` הנ"ל מספקים `pooledPixelFormat[0]` ו- `pooledPixelFormat[1]` ל- `fcneuron` כאשר כ"א מכיל 4 ערכים.

1.14 מדוע המשקלים רשומים במשפט `always` ולא `initial` ?

מימוש יחידת ה- Pooling

- ראשית, נציין שהיחידה אוגרת את שלוש תוצאות הקונבולוציה הראשונות.
- כאשר התוצאה הרביעית מגיעה, היחידה המשווה את כל תוצאות הקונבולוציה עם סף מסוים – 2 במקרה שלנו.
- אם תוצאת הקונבולוציה היא מעל הסף אז היציאה תקבל ערך $1 +$ (כלומר זוהתה הצורה "/" או "\") אחרת יתקבל -1.
- על מנת לפשט את המבנה, תוצאת הקונבולוציה תמיד נכנסת לאותו רגיסטר ומוזז לרגיסטר השכן עם כל עלית שעון.



1.15 רשום את הקוד הבא בקובץ בשם pooling.sv והשלם את קטע הקוד לצבירת תוצאות הקונבולוציה וטעינת הערכים הנכונים ל-pooledReg.

```
`timescale 1ns/100fs

module Pooling ( input clk, En,
                 input [7:0] convResult,
                 output logic [3:0][7:0] pooledPixels);

    integer i;
    logic [2:0] [7:0] convolution;
    logic [3:0] [7:0] pooledReg;

    always_ff @(posedge clk)
    begin
        // Code for accumulating convolution results in "convolution" registers
    end

    always_ff @(negedge clk)
    begin
        begin
            if (En == 1'b1)
            begin
                //Code for loading pooled convolution results into "pooledReg" registers
            end
        end
    end

    assign pooledPixels = pooledReg;
endmodule
```

חשוב : טעינת ה- **pooledReg** מתבצעת עם ירידת השעון על מנת לתאם עדכון ערכים אלה עם קריאת הנתונים מהזיכרונות.

1.16 רשום קובץ **testbench** בשם **pooling_test.sv** לבדיקת המימוש. עליך להעתיק את המימוש החלקי המופיע בהמשך ולהוסיף :

- 6 ערכים נוספים עבור **convResult**. השתמש בערכים :

8'h38 ; 8'h07 ; 8'h73 ; 8'h90 ; 8'h62 ; 8'h84

- משפטים שמאלצים את הערך הנכון בזמן הנכון לאות **En**.

חשוב: יש לעדכן את **En** בעלית השעון על מנת שאות זה יהיה יציב בזמן הדגימה (ירידת השעון).

```
`timescale 1ns/100fs

module pooling_test;

logic clk,En;
logic [7:0] convResult;
logic [3:0][7:0] pooledPixels;

Pooling I_Pooling(.*);

initial
begin
clk = 1'b0 ;
end
initial
begin
//code that sets convResult. For example:
convResult = 8'h31;
@(negedge clk);
convResult = 8'h84;
@(negedge clk);
//add code that assigns 6 more values. One per cycle
#80 $finish;
end

initial
begin
//Code that sets En
end

always
begin
#5 clk = ~clk;
end

endmodule
```

בדיקת יחידת הזיכרון

להלן מימוש חלקי של testbench עבור ה-SRAM :

```
`timescale 1ns/100fs

`define numAddr 5
`define numOut 32
`define wordDepth 32

module dpram32x32_cb_test;

  logic [`numAddr-1:0] A1;
  logic [`numAddr-1:0] A2;
  logic CEB1,WEB1, OEB1, CSB1, CEB2,WEB2, OEB2, CSB2;
  logic [`numOut-1:0] I1,I2;
  logic [`numOut-1:0] O1,O2;

  dpram32x32_cb RAM_U1(.*);

  initial begin
    //initialize CEB1 and CEB2
    @(posedge CEB1);
    //code for SRAM signals
    #30 $finish;
  end

  always
  begin
    // CEB1 code
  end

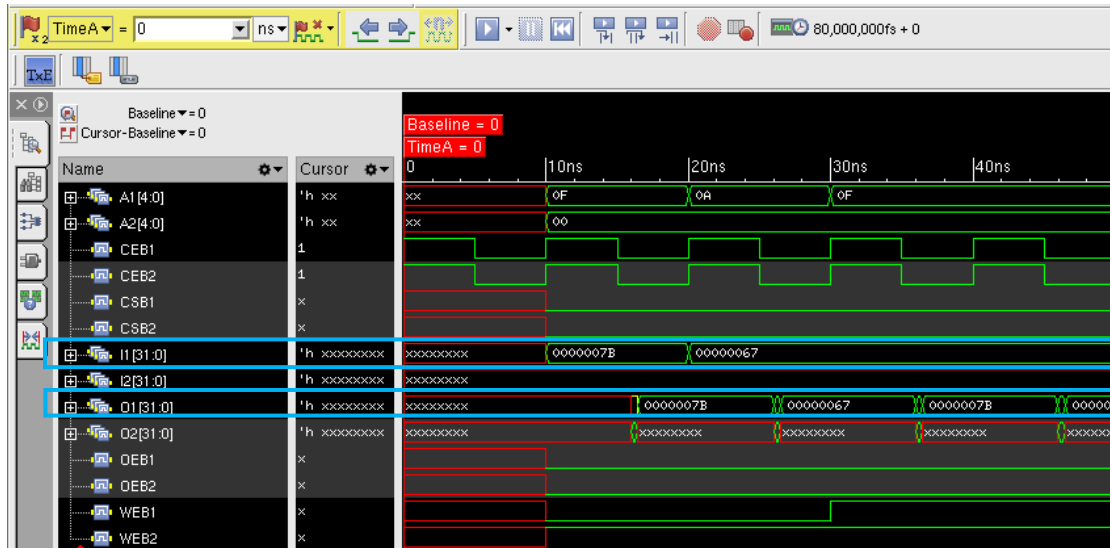
  always
  begin
    // CEB2 code
  end

endmodule
```

1.17 רשום את הקוד הנ"ל בקובץ בשם dpram32x32_cb_test.sv. עליך להוסיף קוד שמבצע את הפעולות הבאות :

1. מגדיר CEB1 ו-CEB2 כשעונים בעלי מחזור של 10ns וערך התחלתי של 0.
2. בעלית השעון השנייה קובע $A1=h0F$ $I1=h7B$ $WEB1=WEB2=0$. זה יגרום לכתיבת h7B בכתובת h0F בירידת השעון הקרובה.
3. בעלית השעון השלישית קובע $A1=h0A$ $I1=h67$. זה יגרום לכתיבת h67 בכתובת h0A בירידת השעון הקרובה.
4. בעלית השעון רביעית קובע $A1=h0F$ $WEB1=1$. זה יגרום לקריאה מכתובת h0F בירידת השעון הקרובה.

העזרו בצורות הגל שבאיור הבא :



הערה - זכרו שה-SRAM בנוי כך שכל הסיגנלים פעילים בנמוך ולא פעילים בגבוה. כך למשל סיגנלי CSB (Chip Select) מבצעים את פעולתם כאשר CSB בנמוך, סיגנלי OEB (Output Enable) מאפשרים מוצא כאשר הם בנמוך וסיגנלי WEB (Write Enable) מגדירים כתיבה כאשר הסיגנלים בנמוך ומאפשרים קריאה כאשר הם בגבוה.

חשוב: מאלצים ערכים לכניסות בעלית שעון, כי הקריאה/כתיבה מתבצעת בירידת שעון.

לסיכום עליך:

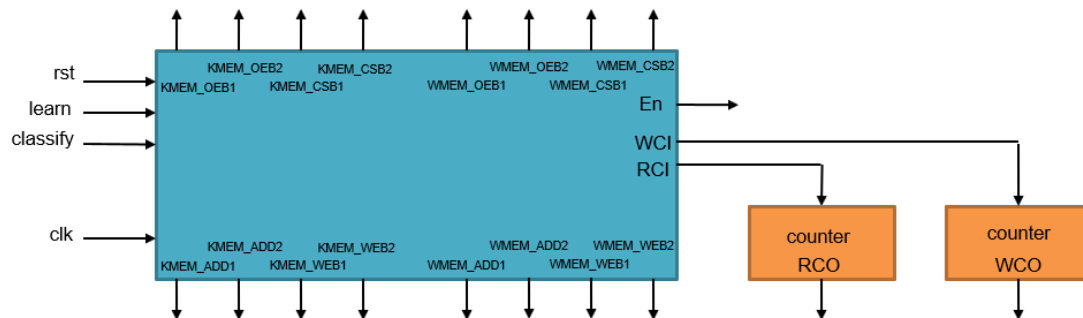
- להשלים את שני משפטי always שהופכים את הערכים של CEB1 ושל CEB2 (השעונים) כל 5ns.
- להשלים את משפט ה-initial כך ש:
- CEB1, CEB2 (שעונים) יקבלו ערכים התחלתיים שווים ל-0.
- CSB1, CSB2, OEB1, OEB2, WEB1, WEB2 יקבלו ערכים קבועים.
- בזמן 15ns (ה-negedge השני) ייכתב h7B לכתובת h0F דרך פורט 1 (כפי שמוסבר בעמ' הקודם).
- בזמן 25ns (ה-negedge השלישי) h67 ייכתב לכתובת 0A דרך פורט 1 (כפי שמוסבר בעמ' הקודם).
- בזמן 35ns (ה-negedge הרביעי) תתבצע קריאה מהכתובת 0F דרך פורט 1 (כפי שמוסבר בעמ' הקודם).
- יש להגדיר את הערך של A2 כ-0.

בנוסף לשיבוץ קטעי הקוד המלאים בדו"ח יש להכין בנוסף את כל קטעי הקוד בקבצים נפרדים לפי שמות המודולים השונים ולהביאם למעבדה.

דו"ח מכין לניסוי מס' 2

החלק זה של הניסוי נתחיל במימוש של הבקר ונמשיך עם הרכבת המערכת כולה.

מימוש הבקר



איור מס' 23 : מבנה הבקר

כניסות הבקר:

clk : אות השעון
rst : אות ה- reset
learn=1 : מסמן שיש לעבור למצב בעלית השעון הבא.
classify=1 : יחד עם learn=0 מסמן שיש לעבור למצב סיווג בעלית השעון הבא.

יציאות הבקר:

ממשק KMEM – לזכרון הקונבולוציה פורט 1

KMEM_ADD1 [5:0] - כתובת
KMEM_OEB1 - אות בקרה - output enable
KMEM_CSB1 - אות בקרה - chip select
KMEM_WEB1 - אות בקרה - write enable

ממשק KMEM – לזכרון הקונבולוציה פורט 2

KMEM_ADD2 [5:0] - כתובת
KMEM_OEB2 - אות בקרה - output enable
KMEM_CSB2 - אות בקרה - chip select
KMEM_WEB2 - אות בקרה - write enable

ממשק WMEM – לזכרון הקונבולוציה פורט 1

WMEM_ADD1 [5:0] - כתובת
WMEM_OEB1 - אות בקרה - output enable
WMEM_CSB1 - אות בקרה - chip select
WMEM_WEB1 - אות בקרה - write enable

ממשק WMEM – לזכרון הקונבולוציה פורט 2

WMEM_ADD2 [5:0] - כתובת
WMEM_OEB2 - אות בקרה - output enable
WMEM_CSB2 - אות בקרה - chip select
WMEM_WEB2 - אות בקרה - write enable

אות עבור יחידת ה-pooling

En - אות בקרה ליחידת ה-pooling

חשוב :

על מנת ליצור את הכתובות של הזיכרונות, אנו נעזר ביציאות של שני מונים (RCO ו-WCO) על מנת למנוע צורך בהגדלת מספר המצבים של המכונה.

אותות פנימיים :

RCI ו-WCI : אותות פנימיים המהווים כניסות למונים. ערכיהם נטענים למונים במצבים מסוימים.

RCO ו-WCO : יציאת המונים.

תזכורת:

כל משקל הוא בגודל של 8 ביט, וכיוון שה-SRAM שומר 32 ביט בכל כתובת של תא זכרון, יש לנו בעצם 4 משקלים בכל כתובת. לכל SRAM ישנם שני ערוצים (פורט 1 ופורט 2) ולכן ניתן לגשת ל-8 משקלים בכל מחזור שעון עבור SRAM בודד.

ניזכר בחלופה C. לפי חלופה זו ברשותינו 3 יחידות SRAM (אחת עבור שלב ה-Convolution ושתיים עבור ה-FC) ולכן ניתן להביא/לספק את המשקלים של שני השלבים במקביל. בשלב ה-FC נדרשים 8 משקלים לצורך זיהוי של כל צורה. כיוון שברשותנו שתי יחידות SRAM ניתן להביא/לספק 16 משקלים תוך מחזור שעון אחד שהם מספיק משקלים עבור שתי צורות. כלומר נדרשים בסך הכל שני מחזורי שעון כדי להביא/לספק את כל המשקלים הנדרשים עבור כל הצורות. לעומת זאת, בשלב ה-Convolution נדרשים 8 משקלים בסך הכל, אותם ניתן להביא/לספק תוך מחזור שעון בודד. לכן נוצר המצב המעניין הבא :

בשלב הלימוד: נרצה לטעון את המשקלים ליחידות ה-SRAM של ה-FC כמה שיותר מהר (2 מחזורים) ומצד שני לאזן בין הזמן הנדרש לכך עבור ה-Convolution ולכן נבחר להשתמש רק בפורט 1 של SRAM של שלב ה-Convolution כדי שגם זה יקח 2 מחזורים.

בשלב הסיווג: עבור ה-FC בכל מחזור נרצה לטעון 8 משקלים שונים (מכתובת אחרת בכל פעם) לצורך זיהוי של כל צורה וצורה בכל מחזור. לעומת זאת, את כל המשקלים (סה"כ 8) של ה-Convolution ניתן לטעון תוך מחזור אחד משתי הכתובות במקביל (דרך פורט 1 ופורט 2). ניתן אפילו להמשיך לעשות זאת מחדש בכל מחזור ומחזור כל עוד נמשך שלב הסיווג.

חשבו על כמות הכתובות הנדרשת עבור גישה למשקלים של שלב ה-convolution ושלב ה-FC. זכרו שניתן לספק כתובות זהות ליחידות ה-SRAM של שלב ה-FC. חשבו איזה מונים כדאי שיהיו בכדי לגשת לכתובות אלו.

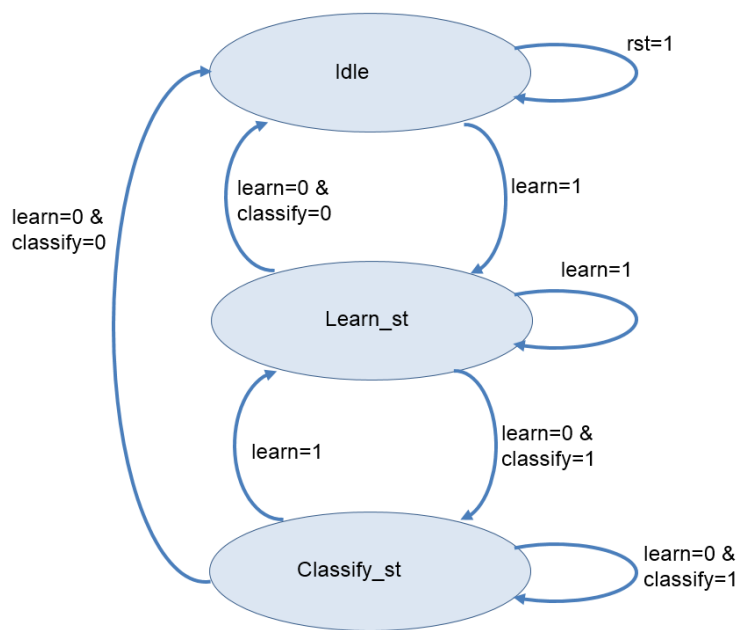
מצבי מכונת המצבים :

Idle_st – מצב סרק

Learn_st – מצב לימוד

Classify_st – מצב סיווג

להלן טבלת המעברים של מכונת המצבים :



rst=1 תמיד מוביל למצב Idle

איור מס' 24 : דיאגרמת המצבים של הבקר

Current State	Inputs			Next_St	Outputs							
	rst	learn	classify		KMEM_WEB1/ KMEM_WEB2	KMEM_ADD1/ KMEM_ADD2	WMEM_WEB1/ WMEM_WEB2	WMEM_ADD1	WMEM_ADD2	WCI	RCI	En
Idle	1	X	X	Idle	X	X	X	X	X	X	X	X
Idle	0	1	X	Learn_st	X	X	X	X	X	X	X	X
Idle	0	0	1	Classify_st	X	X	X	X	X	X	X	X
Learn_st	1	X	X	Idle	X	X	X	X	X	X	X	0
Learn_st	0	1	X	Learn_st	0/1	WCO	0	WCO*2	WCO*2+1	WCO+1	X	0
Learn_st	0	0	1	Classify_st	0/1	WCO	0	WCO*2	WCO*2+1	WCI=0	X	0
Learn_st	0	0	0	Idle	0/1	WCO	0	WCO*2	WCO*2+1	WCI=0	X	0
Classify_st	1	X	X	Idle	X	X	X	X	X	X	X	X
Classify_st	0	1	X	Learn_st	1	0/1	1	RCO	RCO	X	0	X
Classify_st	0	0	1	Classify_st	1	0/1	1	RCO	RCO	X	RCO+1	1*
Classify_st	0	0	0	Idle	X	X	1	RCO	RCO	X	0	0

* רק אם RCO=00

2.1 להלן המימוש החלקי המופיע להלן לקובץ בשם **NeuralNet_cont1.sv**. השלם את המימוש של מכונת המצבים בהתאם לטבלה ודיאגרמת המצבים באיור מס' 24. עליך להשלים את הקוד עבור המצב **Classify_st**.

```
`timescale 1ns/100fs

`define numAddr 5

module NeuralNet_cont (
    input clk, rst, learn, classify,
    output logic [`numAddr-1:0] KMEM_ADD1,
    output logic [`numAddr-1:0] KMEM_ADD2,
    output logic [`numAddr-1:0] WMEM_ADD1,
    output logic [`numAddr-1:0] WMEM_ADD2,
    output logic KMEM_WEB1, KMEM_OEB1, KMEM_CSB1,
    output logic KMEM_WEB2, KMEM_OEB2, KMEM_CSB2,
    output logic WMEM_WEB1, WMEM_OEB1, WMEM_CSB1,
    output logic WMEM_WEB2, WMEM_OEB2, WMEM_CSB2,
    output logic En
);

typedef enum bit[1:0] {Idle_st = 2'b00, Learn_st = 2'b01,
Classify_st = 2'b10} STATE;
STATE CUR_ST;
STATE NEXT_ST;
logic [1:0] RCI, RCO ,WCI, WCO;

always_ff @(posedge clk or posedge rst)
begin
    if (rst == 1) begin
        CUR_ST <= Idle_st;
        RCO    <= 2'b0;
        WCO    <= 2'b0;
    end
    else begin
        CUR_ST <= NEXT_ST;
        RCO    <= RCI;
        WCO    <= WCI;
    end
end
end
```

```

always_comb
begin
    //fixed default values
    KMEM_OEB1 = 1'b0; KMEM_CSB1 = 1'b0;
    KMEM_OEB2 = 1'b0; KMEM_CSB2 = 1'b0;
    WMEM_OEB1 = 1'b0; WMEM_CSB1 = 1'b0;
    WMEM_OEB2 = 1'b0; WMEM_CSB2 = 1'b0;
    //default values
    KMEM_WEB1 = 1'b1;
    KMEM_WEB2 = 1'b1;
    KMEM_ADD1  = 5'b0;
    KMEM_ADD2  = 5'b0;
    WMEM_WEB1 = 1'b1;
    WMEM_WEB2 = 1'b1;
    WMEM_ADD1  = 5'b0;
    WMEM_ADD2  = 5'b0;
    RCI = 2'b0;
    WCI = 2'b0;
    NEXT_ST = Idle_st;

    case(CUR_ST)
        Idle_st:
            begin
                RCI = 2'b0;
                WCI = 2'b0;
                //default - remain in current state:
                NEXT_ST = Idle_st;

                if (learn == 1'b1)
                    begin
                        NEXT_ST = Learn_st;
                    end
                else if (classify == 1'b1)
                    begin
                        NEXT_ST = Classify_st;
                    end
                else
                    NEXT_ST = Idle_st;
            end
    end

```

```

Learn_st:
begin
    // K writes to one port 2 cycles
    // It will write for 2 more cycles because of W but to
different addresses
    // We do it this way to avoid the need for another state
    KMEM_WEB1 = 1'b0;
    KMEM_WEB2 = 1'b1;
    KMEM_ADD1 = {3'b000,WCO};
    KMEM_ADD2 = {3'b000,WCO}; //set so not unknown but not
needed
    // W writes to 2 ports 2 cycles
    WMEM_WEB1 = 1'b0;
    WMEM_WEB2 = 1'b0;
    WMEM_ADD1 = {3'b000,WCO*2};
    WMEM_ADD2 = {3'b000,WCO*2 + 1};
    // default - remain in current state :
    NEXT_ST = Learn_st;
    if (learn == 1'b1)
        begin
            NEXT_ST = Learn_st;
            WCI = WCO + 1;
        end
    else if (classify == 1'b1)
        begin
            NEXT_ST = Classify_st;
            WCI = 2'b0 ;
        end
    else
        begin
            NEXT_ST = Idle_st;
            WCI = 2'b0 ;
        end
    end
end

Classify_st:
begin

    // read from both ports
    // both kernels read and should remain constant - how ? we
remain in this state 4 cycles
    KMEM_WEB1 = 1'b1;
    KMEM_WEB2 = 1'b1;
    //KMEM_ADD1 = count_out;
    //KMEM_ADD2 = count_out+1;
    // Not elegant but should solve problem
    KMEM_ADD1 = 5'b0;
    KMEM_ADD2 = 5'b1;

```

```

//read from one port - repeat for 4 cycles
WMEM_WEB1 = 1'b1;
WMEM_WEB2 = 1'b1; //not necessary
WMEM_ADD1 = {3'b000,RCO};
WMEM_ADD2 = {3'b000,RCO}; //not necessary
// default - remain in current state :
NEXT_ST = Classify_st;

if (learn == 1'b1)
    begin
        //set NEXT_ST and RCI
    end
else if (classify == 1'b1)
    begin
        //set NEXT_ST and increment RCI
    end
else
    begin
        //set NEXT_ST and reset RCI
    end
//set En when RCO has correct value
end

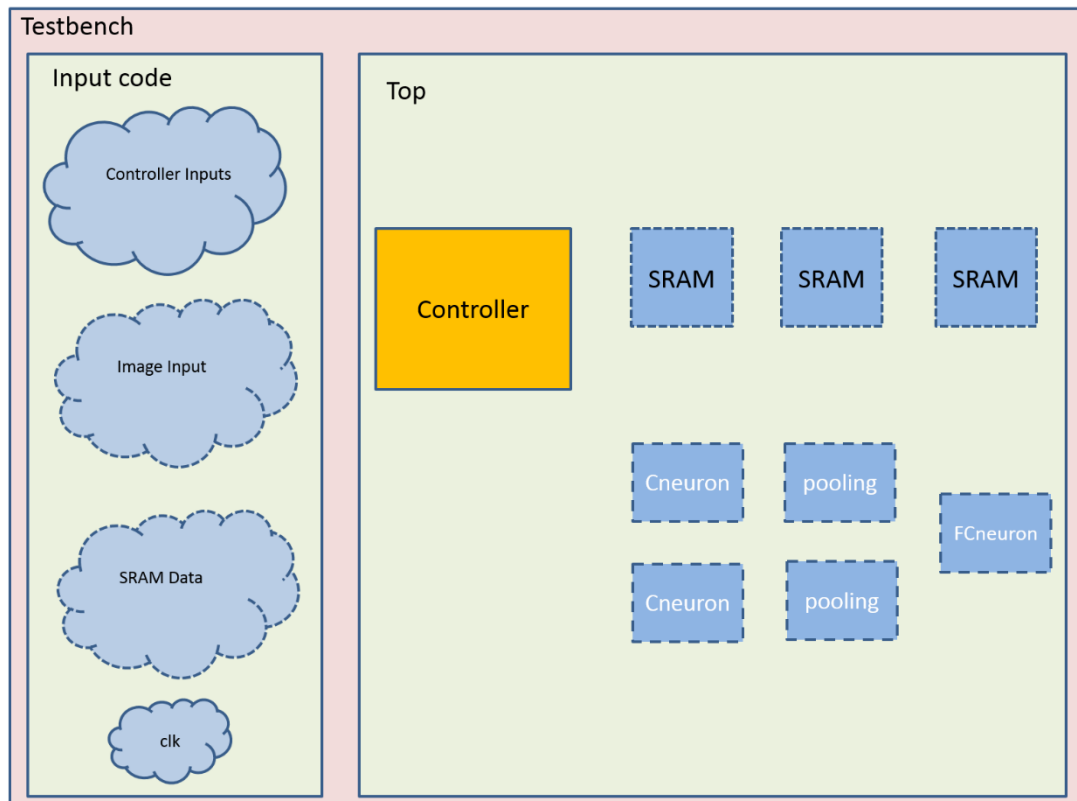
endcase
end

endmodule

```

בהמשך נכין קובץ לבדיקת המכונה.

בדיקת המערכת השלמה תתבצע בשלבים, אבל כבר מסעיף זה נבנה testbench שיתאים למערכת הכוללת את כל חלקיה למרות שבשלב ראשון יחידת ה- Top (איור מס' 25) תכיל רק את הבקר וכל אותות הכניסה שהוא דורש. להלן תיאור מבנה ה- testbench. בסעיף זה לא נכליל את כל הבלוקים שמופיעים בקווים מקווקוים.



איור מס' 25

להלן מימוש ה- testbench :

```

`timescale 1ns/100fs
`define numAddr 5

module Top_test;

logic clk;
logic rst, learn, classify;
logic [`numAddr-1:0] KMEM_ADD1;
logic [`numAddr-1:0] KMEM_ADD2;
logic [`numAddr-1:0] WMEM_ADD1;
logic [`numAddr-1:0] WMEM_ADD2;
logic [2:0] [7:0] InputImage[2:0];
logic [3:0] [7:0] pixels;
logic [7:0] result;
logic KMEM_WEB1, KMEM_OEB1, KMEM_CSB1;
logic KMEM_WEB2, KMEM_OEB2, KMEM_CSB2;
logic WMEM_WEB1, WMEM_OEB1, WMEM_CSB1;
logic WMEM_WEB2, WMEM_OEB2, WMEM_CSB2;
logic [31:0] KR_DATA_I1, KR_DATA_I2, W1_DATA_I1, W1_DATA_I2,
W2_DATA_I1, W2_DATA_I2;

Top Top_U1(.*)

initial
begin
clk      = 1'b0 ;
rst      = 1'b1 ;
learn    = 1'b0 ;
classify = 1'b0 ;
@(negedge clk);
rst      = 1'b0 ;
learn    = 1'b1 ;
repeat(2) @(negedge clk);
learn    = 1'b0 ;
classify = 1'b1 ;
#250 $finish;
end

always
begin
#5 clk = ~clk;
end
// The next line is used for debugging
`include "bin/monitor.sv"
endmodule

```

חשוב!

- בקובץ הנ"ל מוגדרים כל הסיגנלים שיופיעו בתכנון המלא למרות שבסעיף זה נחוצים רק: **rst**, **learn**, **classify** ו-**clk** כלומר הסיגנלים שמתחברים לבקר. הגדרת כל הסיגנלים כבר בשלב זה תחסוך את הצורך בהוספת סיגנלים בכל סעיף וסעיף.

- סיגנלים ששומם מתחיל ב-K הם סיגנלים שמתחברים ל-SRAM של שלב הקונבולוציה.
- סיגנלים ששומם מתחיל ב-W הם סיגנלים שמתחברים ל-SRAM של שלב ה-FC

2.2 רשום את הקוד הנ"ל בקובץ בשם `Top1_test.sv`.
הסבר את המשפט :

```
Top Top_U1(.*);
```

צייר את צורות הגל של : `rst`, `learn`, `classify` ו-`clk` כפי שהם מוגדרים ב-`testbench`.

להלן מימוש ראשון ליחידת ה-`Top`. הבקר הוא הבלוק היחיד שמופיע ב-`Top` :

```
`timescale 1ns/100fs
`define numAddr      5
`define NoOfKernels  2
`define NoOfShapes   4

module Top ( input clk,
             input [3:0] [7:0] pixels,
             input rst, learn, classify,
             input [31:0]  KR_DATA_I1,  KR_DATA_I2,  W1_DATA_I1,  W1_DATA_I2,
             W2_DATA_I1, W2_DATA_I2,
             output logic [7:0] result);

logic [`NoOfKernels-1:0] [7:0] convResult;
logic [3:0] [7:0] pooledPixelArray[`NoOfKernels-1:0];
logic [`numAddr-1:0] KMEM_ADD1;
logic [`numAddr-1:0] KMEM_ADD2;
logic [`numAddr-1:0] WMEM_ADD1;
logic [`numAddr-1:0] WMEM_ADD2;
logic KMEM_WEB1, KMEM_OEB1, KMEM_CSB1;
logic KMEM_WEB2, KMEM_OEB2, KMEM_CSB2;
logic WMEM_WEB1, WMEM_OEB1, WMEM_CSB1;
logic WMEM_WEB2, WMEM_OEB2, WMEM_CSB2;
logic [1:0][31:0] KR_DATA_O;
logic [31:0] W1_DATA_O1, W1_DATA_O2, W2_DATA_O1, W2_DATA_O2;
logic En;

// Add instantiation of NeuralNet_cont. Call the instance NNC_U1

// Example of generate statement (required for exp'2 later section)
// genvar k;
// generate for (k = 0; k <`NoOfKernels ; k = k + 1) begin
// Code to be repeated for each k
// end
// endgenerate

endmodule
```

הערה : `En` – סיגנל בקרה שמתחבר ליחידת ה-`Pooling`

2.3 רשום את הקוד הנ"ל לקובץ בשם **Top1.sv**. הוסף לקוד, הצבה (**instantiation**) של הבקר **NeuralNet_cont**. על שם ה-**instance** להיות **NNC_U1**.

הוספת זיכרונות ה- SRAM

בסעיף זה :

- במידת הצורך קראו שוב את התזכורת בתחיל הפרק של מימוש הבקר לעיל.
- לזיכרון הקונבולוציה נכתבות שתי מילים בעלות 32 סיביות (4 משקלים בעלי 8 סיביות) ולכל אחד מזיכרונות ה-FC נכתבות 4 מילים כפי שמופיע בטבלה מס' 1.
- הכתיבה תתבצע במהלך שני מחזורי שעון כאשר לזיכרון הקונבולוציה יעשה שימוש בפורט אחד בלבד ולזיכרונות ה-FC יעשה שימוש בשני הפורטים.
- המילים שנבחרו בסעיף זה אינן המשקלים האמיתיים אלא ערכים שיעזרו בבדיקת תקינות הכתיבה.
- המידע והכתובות שיש להזין לזיכרונות במהלך הכתיבה מופיעים בטבלה 2.
- לעומת זאת, המידע הנקרא והכתובות שיש להזין לזיכרונות במהלך הקריאה מופיעים בטבלה 3.
- יש לתזמן את המידע בעזרת הסיגנלים שמוגדרים ב-**testbench**. עבור שלב ה-**learn** עליך לספק את המידע וכתובות שמופיעים בטבלה.

KMEM_ADD	Value
0	00000000
1	00000001

Data written to 2 addresses using port 1
Both words(32 bits) read simultaneously using both ports
Output should be : 00000000, 00000001

WMEM_ADD	ValueW1	ValueW2
0	00000010	00000020
1	00000011	00000021
2	00000012	00000022
3	00000013	00000023

Data written to 2 addresses using both ports
0 and 1 addresses written in first cycle
2 and 3 addresses written in second cycle
Both words (32 bits) read simultaneously using port1 of each SRAM
Output of port 2 not used
Over 4 cycles output should be :
00000010, 00000020
00000011, 00000021
00000012, 00000022
00000013, 00000023

טבלה 1

clk	KMEM_ADD1	KMEM_ADD2	KR_DATA_I1	KR_DATA_I2	WMEM_ADD1	WMEM_ADD2	W1_DATA_I1	W1_DATA_I2	W2_DATA_I1	W2_DATA_I2
0	X	X	X	X	X	X	X	X	X	X
1	0	X	00000000	X	0	1	00000010	00000011	00000020	00000021
2	1	X	00000001	X	2	3	00000012	00000013	00000022	00000023

טבלה 2 : כתיבה

clk	KMEM_ADD1	KMEM_ADD1	KR_DATA_O[0]	KR_DATA_O[1]	W1MEM_ADD1	W2MEM_ADD1	W1_DATA_O1	W2_DATA_O1
0	X	X	X	X	X	X	X	X
3	0	1	00000000	00000001	0	0	00000010	00000020
4	0	1	00000000	00000001	1	1	00000011	00000021
5	0	1	00000000	00000001	2	2	00000012	00000022
6	0	1	00000000	00000001	3	3	00000013	00000023
7	0	1	00000000	00000001	0	0	00000010	00000020
8	0	1	00000000	00000001	1	1	00000011	00000021
9	0	1	00000000	00000001	2	2	00000012	00000022
10	0	1	00000000	00000001	3	3	00000013	00000023

טבלה 3 : קריאה

2.4

צור העתקים של **Top1_test.sv**, **Top1.sv** ו- **NeuralNet_cont1.sv** וקרא לקבצים החדשים בשמות **Top2.sv**, **Top2_test.sv** ו- **NeuralNet_cont2.sv** בהתאמה. לקובץ **Top2.sv** הוסף 3 זיכרונות SRAM (אחד לשלב הקונבולוציה ושניים לשלב ה-FC לפי ההערה הבאה).

בסעיף זה ובכל הסעיפים בהמשך, כאשר מוסיפים יחידה למימוש **Top** יש לעשות זאת בעזרת משפט מסוג :

```
nand nand_U1 (.out(w), .in1(p), .in2(q));
```

ולא מסוג :

```
nand nand_U1 (.*);
```

הסיבה לכך היא שאין זהות מלאה בין שמות הממשק של היחידה ושמות הסיגנלים שמתחברים אליהם ב- **testbench**.

2.5 הוסף ל- **Top_test2.sv** משפט **initial** שמספק מידע שייכתב בזיכרונות שנוספו. יש לכתוב לזיכרונות את המידע שמופיע בטבלאות הנ"ל כלומר :

```
repeat(2) @(posedge clk);
KR_DATA_I1 = 32'h0;
W1_DATA_I1 = 32'h10;
W2_DATA_I1 = 32'h20;
W1_DATA_I2 = 32'h11;
W2_DATA_I2 = 32'h21;
@(posedge clk);
KR_DATA_I1 = 32'h1;
W1_DATA_I1 = 32'h12;
W2_DATA_I1 = 32'h22;
W1_DATA_I2 = 32'h13;
W2_DATA_I2 = 32'h23;
```

לפני כל ניסוי יש להכין דו"ח מכין בפורמט **pdf**. יש ליצור קובץ **ZIP** (לא **RAR**) המכיל את דו"ח המכין ואת קובצי הקוד שהכנת ולהעלות אותו ל- **LabAdmin** לפני תחילת הניסוי.

הערות כלליות למהלך העבודה:

- סעיף לביצוע מופיע כ- "–", סעיף לשמירת גרף מופיע כ- 'G11' וסעיף תשובה לשאלה מופיע כ- 'Q11'. מומלץ לשמור את התמונות בהתאם למספר הסעיף, למשל G11.
- הרצת פקודות תתבצע בחלון ה- **terminal** שבאופן טיפוסי יציג בשורה האחרונה טקסט דוגמת זה:

```
[ mlst1@omega81 ~/Shimshon_Yovav ]$ _
```

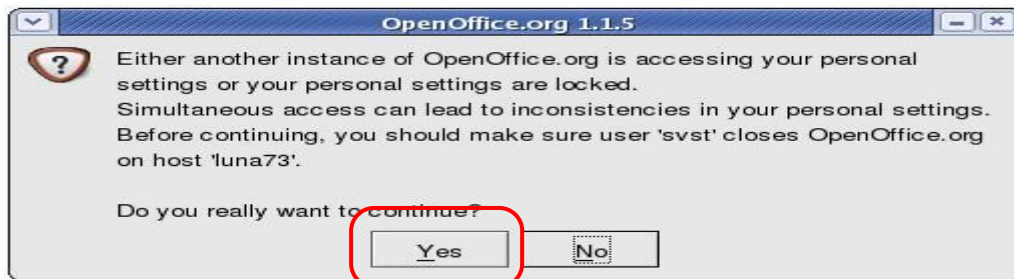
- את הפקודות יש להקליד מימין לו. בהמשך השורה לעיל הנקראת **prompt** תסומן בקיצור כ- '\$'. אין צורך להקליד את הסימן \$. כאשר ה- **prompt** מופיע הוא מסמן שה- **terminal** מוכן לקלוט את הפקודה הבאה. כאשר הוא איננו מופיע זה סימן שה- **terminal** עסוק בפקודה הקודמת.
- כתיבת דוח הניסוי תתבצע במהלך הניסוי באופן ממוחשב בעזרת תבנית שברשותכם. כל הגרפים והסכמות ישולבו בדו"ח זה ללא הצורך בהדפסות כמוסבר בהמשך.

הערות כלליות לעריכת הדוח:

- יש להימנע מפתחת הקובץ ע"י דאבל-קליק כיוון שאז לא ברור מאיזו גרסה נפתח הקובץ ואלול לגרום לקריסת התוכנה.
- לעריכת הדו"ח רשום:

```
$ oowrite name1_name2.odt &
```

- כאשר **name1** ו- **name2** הם שמות הסטודנטים. שים לב ש- **name1_name2.odt** אינו קובץ ריק אלא תבנית עליה מתבסס הדו"ח.
- עריכת הדוח הממוחשב תעשה ע"י מעבד תמלילים של **Open Office**. יתכן ויפתח חלון ההערה הבא:



- לחץ על **Yes** בכדי להמשיך. בדוק שנפתח קובץ לפי שמות הסטודנטים שלכם.

הערות עבור צירוף תמונות/סכמות לדוח:

- ראשית יש צורך ליצור קובץ **jpeg**.
- הפעל את תוכנת **knapshot** בחלון **terminal**.

```
$ ksnapshot &
```

- לחץ על **New Snapshot**.
- בחלון שנפתח בחר ב- **Region** עבור ה- **Capture mode**.
- באמצעות הכפתור השמאלי בחר באיזור של המסך שברצונך להוסיף לדו"ח. ברגע שסיימת, לחץ במקלדת על מקש **Enter** והאיזור המסומן ישמר.
- שמור את התמונה באמצעות **Save As**. בחר בפורמט **jpg** והכנס שם לתמונה עם סיומת **jpg** או לחץ על **Copy to Clipboard** וצרף לדו"ח עם **CNTR V**.



- רצוי להקטין את רוחב חלון ה- waveform כדי שהוא יכנס בצורה יפה יותר לדו"ח.
- הכנס את הסכמה לדו"ח ב- openoffice עם Insert→Picture→From File
- שינוי שפה יעשה ע"י Alt+Shift ושינוי כיוון כתיבה ע"י Ctrl+Shift
- שינוי כיוון הטקסט יעשה ע"י Ctrl+Shift+A ו- Ctrl+Shift+D

חשוב !! עריכת קוד SystemVerilog

העריכה של כל קבצי ה- SystemVerilog תתבצע בעזרת כלי בשם Euclide. כלי זה עוזר מאוד בגילוי ותיקון כל שגיאות ה- syntax. להפעלת הכלי רשום :

```
$ start_veride
```

הוספה של קבצים תתבצע תמיד בתוך הכלי (לרוב עם Save As) ולא בעזרת פקודות linux. חשוב שהני"ל ל יתבצע בספריה :

/users/mlstN/student1_student2 (N=1,2,3 or 4)

ביצוע ניסוי מס' 1

בעזרת ה- script בשם run_sim מריצים סימולציה. על מנת שה- script יעבוד, יש להקפיד על הכללים הבאים בזמן מתן שמות:

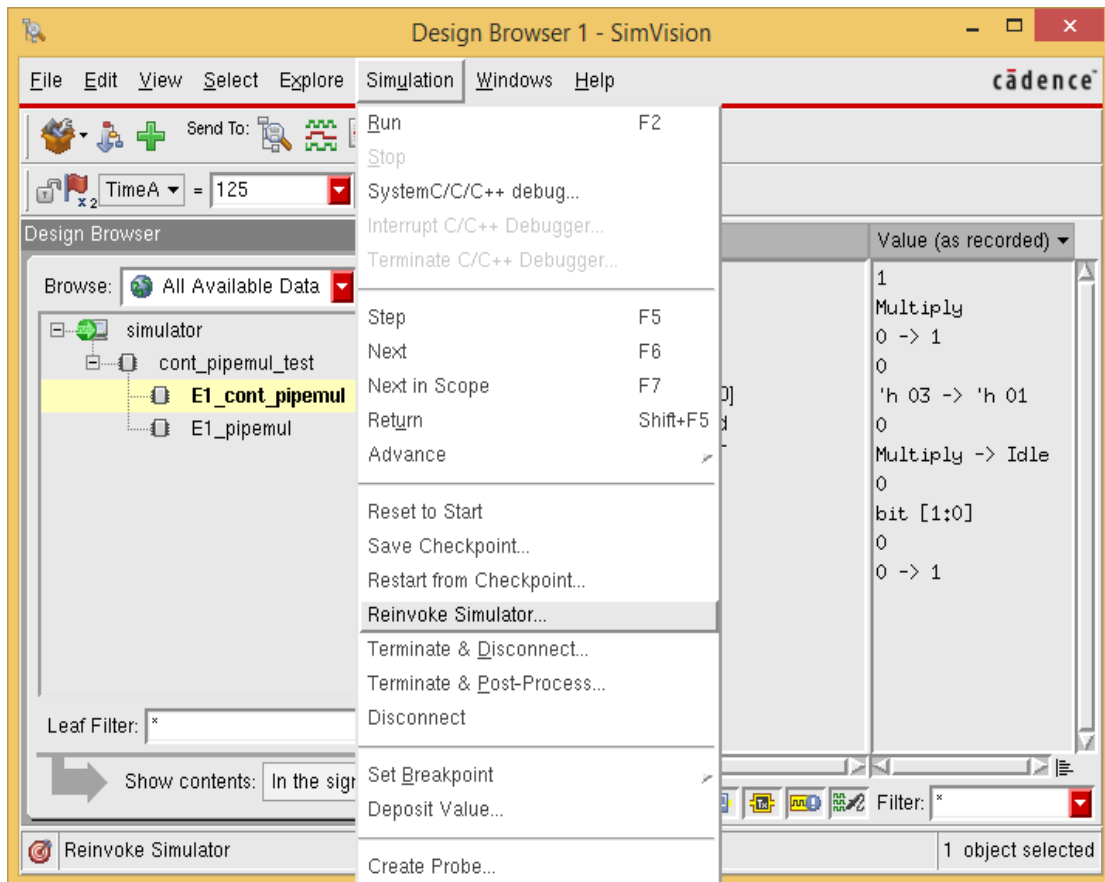
- כאשר שם המודול הוא <design> שם הקובץ הנדרש צריך להיות : <design>.sv
- שם מודול-הבדיקה (testbench) של המודול <design> צריך להיות <design>_test ובהתאם שם הקובץ של מודול-הבדיקה צריך להיות : <design>_test.sv
- <design> הוא שם כלשהו בהתאם לצורך לפי הדוגמה הבאה.
- למשל מודול בשם cneuron ייכתב בקובץ בשם cneuron.sv. ה- testbench יהיה במודול בשם cneuron_test וייכתב בקובץ בשם cneuron_test.sv.
- יש להריץ את הסימולציה עם הפקודה :

```
$ run_sim <design>
```

עבור כל סעיף, חשוב שהני"ל יתבצע בספריה :

/users/mlstN/student1_student2/EXPT (N=1,2,3 or 4)

- ניתן לסמלץ מחדש אם שיניתם את הקוד בעזרת : Simulation → Reinvoke Simulator → Yes



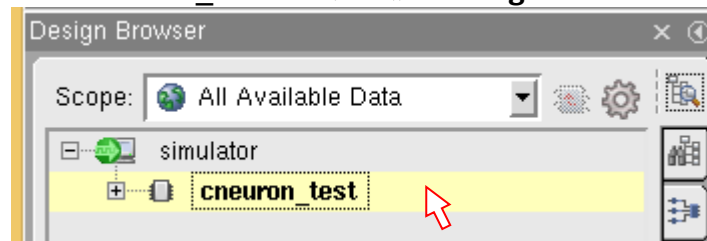
חשוב : במהלך הניסוי הינך מתבקש "להסביר את צורות הגל של הסימולציה". הכוונה היא שעליך לציין את אירועים העיקריים שמראים את נכונות הסימולציה.


1. מימוש נוירון הקונבולוציה


- השתמש בקבצים בשם `cneuron.sv` ו-`cneuron_test.sv` שהכנת. הרץ סימולציה של המודול בעזרת הפקודה :

```
$ run_sim cneuron
```

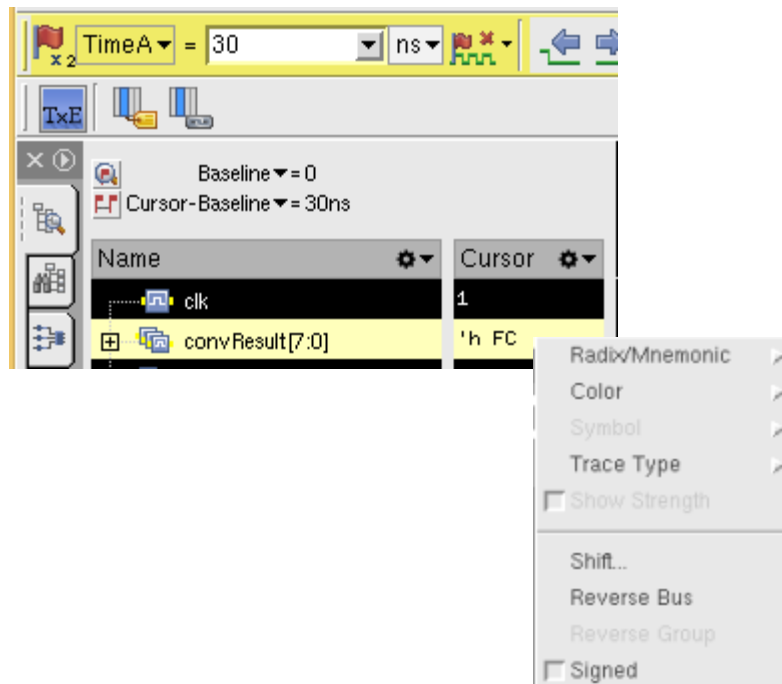
- בחלון של ה- **Design Browser** בחר ביחידה `cneuron_test`



- לחץ על כפתור ה- **Waveform** (). פעולה זאת פותחת חלון עבור צורות הגל של כל הסיגנלים של `cneuron_test`.

- לחץ על הכפתור **Play** () שמופיע בחלק העליון של חלון ה- **Waveform** להרצת ה- `testbench`.

- הבא את הסמן מעל ערך הסיגנל בעמודת **cursor** (במקרה זה 'hFC) לחץ על הכפתור הימני וסמן X במשבצת שליד **signed**.



- G11** : שמור תמונת מסך של צורת הגל וצרף את התמונה לדו"ח.
- Q11** : בדוק את התוצאות המתקבלות. האם הן נכונות ? הסבר.
- שנה בקובץ `cneuron_test.sv` את הערכים של שני ה- `kernels` כך שהערכים יהיו `+5` ו- `-1` במקום `+1` ו- `-1`.
- הרץ את הסימולציה מחדש.
- G12** : שמור תמונת מסך של צורת הגל וצרף את התמונה לדו"ח.
- Q12** : בדוק את התוצאות המתקבלות. האם הן נכונות ? הסבר.
- שנה את המימוש כך שכל החישוב של ה- `sum` יתבצע הפעם באמצעות לולאת `FOR`.
- Q13** : בדוק את התוצאות המתקבלות.
- G13** : הוסף את הקוד של חישוב ה- `sum` לדו"ח (תצלום מסך או הטקסט עצמו).
- G14** : שמור תמונת מסך של צורת הגל וצרף את התמונה לדו"ח.

- הראה את המימוש למדריך.

הערות :

- ניתן לעשות **zoom** ע"י לחיצה על **Ctrl** + גלגלת בעכבר.
- חשוב להצביע על אזורים חשובים בפלטי ההדפסות.
- בסיום כל סימולציה, יש לצאת מהכלי עם




File → **Exit**

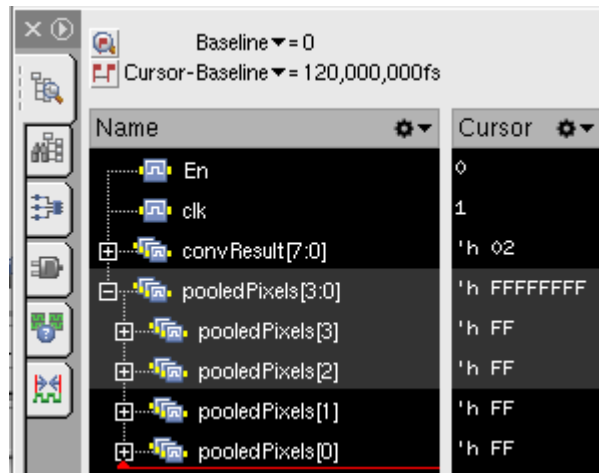
2. מימוש נזירון ה-FC

- השתמש בקובצים בשם `fcneuron.sv` ו- `fcneuron_test.sv` שהכנת. הרץ סימולציה של המודול בעזרת הפקודה :

```
$ run_sim fcneuron
```

- בחלון של ה- **Design Browser** בחר ביחידה `fcneuron_test`

- לחץ על כפתור ה- **Waveform** (). פעולה זאת פותחת חלון עבור צורות הגל של כל הסיגנלים של `fcneuron_test`.
- לחץ על הכפתור **Play** () שמופיע בחלק העליון של חלון ה- **Waveform** להרצת ה- `testbench`.
- לחץ על סימן ה-  ליד סיגנל `pooledPixels` כדי לראות את הערך שבכל תא.





- G21:** שמור תמונת מסך של צורת הגל וצרף את התמונה לדו"ח.
- Q21:** בדוק את התוצאות המתקבלות. האם הן נכונות? הסבר את המשמעות של הערכים שמתקבלים ב- `result`.

3. מימוש יחידת ה- **Pooling**

- השתמש בקבצים בשם `pooling.sv` ו- `pooling_test.sv` שהכנתם. הריצו סימולציה של המודול בעזרת הפקודה:

```
$ run_sim pooling
```



- בחלון של ה- **Design Browser** בחר ביחידה `pooling_test`.
- לחץ על כפתור ה- **Waveform** () פעולה זאת פותחת חלון עבור צורות הגל של כל הסיגנלים של `pooling_test`.
- לחץ על הכפתור **Play** () שמופיע בחלק העליון של חלון ה- **Waveform** להרצת ה- `testbench`.

- G31:** שמור תמונת מסך של צורת הגל וצרף את התמונה לדו"ח.
- Q31:** האם התוצאות המתקבלות ב- `pooledPixel` נכונות? הסבר? הראה שאות ה- `En` עולה בזמן הנכון.
- את `convolution[i]` יש להשוות עם 2. עליך לבצע את ההשוואה פעם עם `convolution[i]` ופעם נוספת עם `signed(convolution[i])`.
- Q32:** רשום את הבדל בין שתי הריצות.
- G32:** שמור תמונת מסך של צורת הגל הנוספת וצרף את התמונה לדו"ח.

4. סימולציה של יחידת הזיכרון

- השתמש בקובץ בשם `dpram32x32_cb.sv` הנתון ובקובץ בשם `dpram32x32_cb_test.sv` שהכנת. הרץ סימולציה של המודול בעזרת הפקודה :

```
$ run_sim2 dpram32x32_cb
```

- בחלון של ה- **Design Browser** בחר ביחידה `dpram32x32_cb_test`.
- לחץ על כפתור ה- **Waveform** (). פעולה זאת פותחת חלון עבור צורות הגל של כל הסיגנלים של `dpram32x32_cb_test`.
- לחץ על הכפתור **Play** () שמופיע בחלק העליון של חלון ה- **Waveform** להרצת ה- `testbench`.

- G41** : שמור תמונת מסך של צורת הגל וצרף את התמונה לדו"ח.
- Q41** : בדוק את התוצאות המתקבלות. הראה בדו"ח שהן נכונות.
- הוסף ל- `testbench` עוד פעולה כך שתבצע קריאה מכתובת `h0A` דרך פורט מס' 1.
- הרץ את הסימולציה שוב.
- G42** : שמור תמונת מסך של צורת הגל וצרף את התמונה לדו"ח.
- Q42** : בדוק את התוצאות. הראה בדו"ח שהקריאה מתבצעת בצורה נכונה?

ביצוע ניסוי מס' 2

- שים לב שבחלק זה של הניסוי שם המודולים הם תמיד **Top** ו- **Top_test**. שמות הקבצים משתנים בכל סעיף.

סימולציה של המערכת השלמה

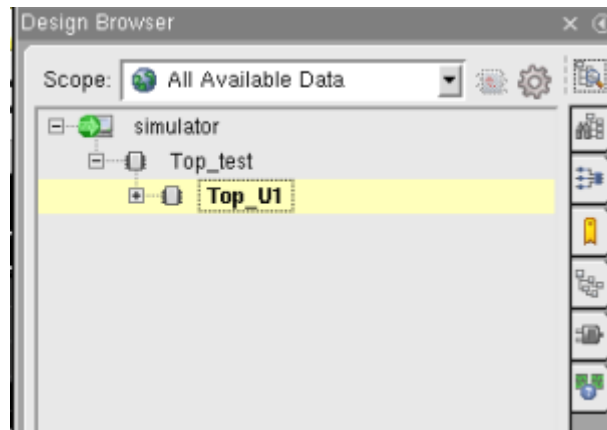
- כאמור כבר מההתחלה התבקשת לבנות **testbench** שיתאים למערכת השלמה אבל בשלב ראשון ה- **testbench** יכול רק את הבקר וכל אותות הכניסה שהוא דורש.

1. סימולציה של הבקר בעזרת Top1

- בסעיף זה נבדוק את נכונות חיבור ה- **testbench** לבקר שמופיע כ- **module** יחיד ב- **Top1**.
- השתמשו בקבצים בשם **Top1.sv** ו- **Top1_test.sv** שהכנתם. הריצו סימולציה של המודול בעזרת הפקודה :

```
$ run_Top1
```

- בחלון של ה- **Design Browser** בחר ביחידה **Top_U1** שנמצאת ב- **Top_test**



- בעמודה הצמודה בחר בסיגנלים הרלוונטיים: אותות ה- `classify`, `learn`, `rst`, `clk` וכן באותות הכתובות, ואותות ה- `WEB` של שלב ה-**Convolution** ושלב ה-**FC שנמצאים בתוך מודול Top_U1**.
- לחץ על כפתור ה- **Waveform** (). פעולה זאת פותחת חלון עבור צורות הגל של כל הסיגנלים של **Top_U1** הרלוונטיים לסעיף זה.
- חזור למסך הקודם ואתר את אות ה- `CUR_ST` שנמצא בהמשך ההיררכיה והוסף גם אותו ל-**Waveform**.
- לחץ על הכפתור **Play** () שמופיע בחלק העליון של חלון ה- **Waveform** להרצת ה-**testbench**.
- **G11** : שמור תמונת מסך של צורת הגל וצרף את התמונה לדו"ח.
- **Q11** : הראה על צורות הגל שראשית עולה `rst`, אחריו `learn` ובסוף `classify`. הראה שמכונת המצבים עוברת דרך המצבים הנכונים. הראה שכל עוד המכונה נשארת במתב `Classify_st` הכתובות של הזיכרון `FC` נכונות. הסבר.
- **Q12** : עליך לעבור על כל אותות ה- `WEB`, `WMEM_ADD1`, `KMEM_ADD1`, `KMEM_ADD2`, `WMEM_ADD2` ואתר את אות ה- `CUR_ST` הכתובות והמצב של המכונה ולוודא ולהסביר מדוע ההתנהגות היא נכונה.
- **הראה את התוצאות למדריך.**

2. סימולציה של הבקר ויחידות הזיכרון: **Top2.sv**

- בסעיף זה נבדוק את נכונות התנהגות שלשת הזיכרונות שהוספת ל- **Top2**. על מנת להקל על הבדיקה נבצע כתיבת המספרים שהופיעו בדו"ח המכין.
- השתמשו בקבצים בשם **Top2.sv** ו- **Top2_test.sv** שהכנתם. הריצו סימולציה של המודול בעזרת הפקודה:

```
$ run_Top2
```

- בחלון של ה- **Design Browser** בחר ביחידה **Top_U1** שנמצאת ב- **Top_test**
- לחץ על כפתור ה- **Waveform** (). פעולה זאת פותחת חלון עבור צורות הגל של כל הסיגנלים של **Top_U1** שב- **Top_test**.
- חזור למסך הקודם ואתר את אות ה- `CUR_ST` שנמצא בהמשך ההיררכיה והוסף גם אותו ל-**Waveform**.
- לחץ על הכפתור **Play** () שמופיע בחלק העליון של חלון ה- **Waveform** להרצת ה-**testbench**.
- לחץ על ה- `+` של האות `KR_DATA_0[1:0]` כדי לראות את שני הבסיסים (**Buses**) בנפרד

- וודא שבתמונה ניתן לקרוא את ערכי ה DATA בבסיס וגם את שמות כל האותות G21 : שמור תמונת מסך של צורת הגל וצרף את התמונה לדו"ח.
 - Q21 : בדוק (והראה על גבי צורות הגל) שכל המספרים נכתבים ונקראים אל ומהזיכרונות בהתאם לצפוי כלומר, בהתאם לטבלאות 1,2 ו-3 בסעיף ההכנה.
- הראה את התוצאות למדריך**

3. הוספת נוירונים של שלב הקונבולוציה (Top3) :

- **הראה את התוצאות למדריך.**
צור גרסה חדשה (3) של כל התכנון באופן הבא.

בעזרת File->Save As שמור את NeuralNet_cont2.sv לקובץ בשם NeuralNet_cont3.sv
בעזרת File->Save As שמור את Top2.sv לקובץ בשם Top3.sv
בעזרת File->Save As שמור את Top2_test.sv לקובץ בשם Top3_test.sv

בשלב הבא תידרש להציב את המודולים של הנוירונים של שלב ה-Convolution בלבד שהכנת בחלק א' של הניסוי. כיוון שעקרונית ישנם נוירונים מרובים, ההצבות לא תכתבנה אחת אחת.

לקובץ Top3.sv הוסף 2 נוירונים של יחידת הקונבולוציה (השתמש במשפט generate).
לקובץ Top3_test.sv שנה את משפט initial שמספק מידע שייכתב לזיכרונות. יש לכתוב לזיכרונות את הערכים הנכונים של המסננים ומשקלים כפי שתוארו לעיל, כלומר :

```
repeat(2) @(posedge clk);
KR_DATA_I1 = 32'h01ffff01 ; //first filter looks for \
W1_DATA_I1 = 32'h01ffff01 ;
W2_DATA_I1 = 32'hff0101ff ;
W1_DATA_I2 = 32'hff0101ff ;
W2_DATA_I2 = 32'h01ffff01 ;
@(posedge clk);
KR_DATA_I1 = 32'hff0101ff ; //second filter looks for /
W1_DATA_I1 = 32'hffffffff ;
W2_DATA_I1 = 32'hff0101ff ;
W1_DATA_I2 = 32'h01ffff01 ;
W2_DATA_I2 = 32'hffffffff ;
```

הוסף משפט initial שמגדיר את 4 התמונות (X, O, \, /) ומעביר 4 תתי הבלוקים לכניסה pixels במהלך 4 מחזורים. לדוגמא, עבור התמונה הראשונה "X" :
(השלימו לבד את השאר)

```
// Shape-1: "X"
repeat(4) @(negedge clk);
InputImage[2] = {8'h01,8'hff,8'h01} ;
InputImage[1] = {8'hff,8'h01,8'hff} ;
InputImage[0] = {8'h01,8'hff,8'h01} ;
pixels = {InputImage[2][2],InputImage[2][1],InputImage[1][2],InputImage[1][1]};
@(negedge clk);
pixels = {InputImage[2][1],InputImage[2][0],InputImage[1][1],InputImage[1][0]};
@(negedge clk);
pixels = {InputImage[1][2],InputImage[1][1],InputImage[0][2],InputImage[0][1]};
@(negedge clk);
```

```

pixels = {InputImage[1][1],InputImage[1][0],InputImage[0][1],InputImage[0][0]};
// Shape-2: "0"
@(negedge clk);
// COMPLETE InputImage... , pixels..., #10..., etc.


// COMPLETE Shapes-3: "/"
// COMPLETE Shapes-4: "\"

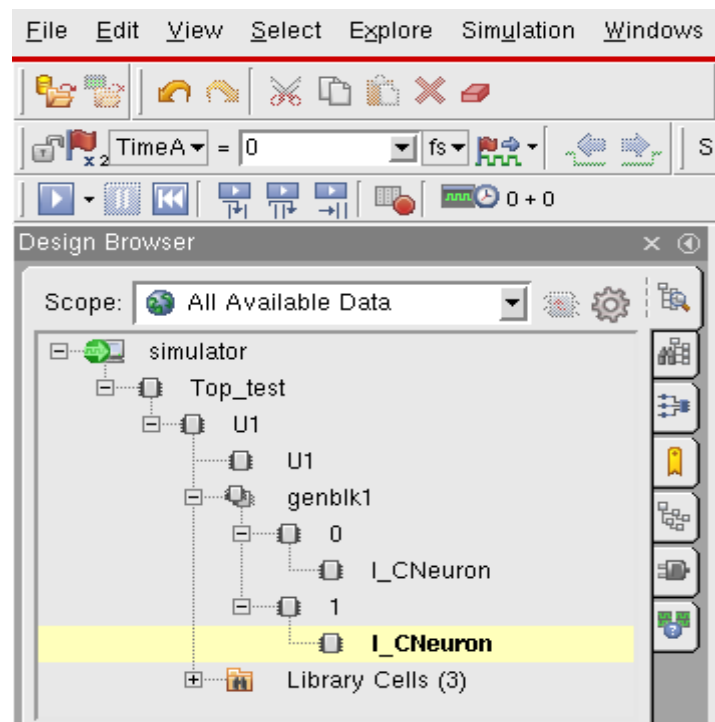
```


- נא לשמור על סדר הכניסות.

- השתמש בקבצים בשם Top3.sv ו-Top3_test.sv שהכנת. הרץ סימולציה של המודול בעזרת הפקודה :

```
$ run_Top3
```

- בחלון של ה- **Design Browser** פתח את מבנה התכנון כפי שמתואר באיור הבא. בחר ביחידה I_CNeuron של יחידה 0. לחץ על כפתור ה- **Waveform** (). חזור על הפעולה עבור ה- I_CNeuron השני. פעולה זאת פותחת חלון עבור צורות הגל של כל הסיגנלים של הנורונים. הוסף גם את InputImage לחלון הגלים.



- לחץ על הכפתור **Play** () שמופיע בחלק העליון של חלון ה- **Waveform** להרצת ה- **testbench**.

G31 : שמור תמונת מסך של צורת הגל וצרף את התמונה לדו"ח.

Q31 : בדוק שכל המספרים תוצאות שני הנורונים בהתאם לצפוי. הסבר את התוצאות המתקבלות עבור שני סיגנלי ה- **sum**.

- הראה את התוצאות למדריך.

4. הוספת יחידות ה- Pooling לשלב הקונבולוציה (Top4):

צור גרסה חדשה (4) של כל התכנון באופן הבא.

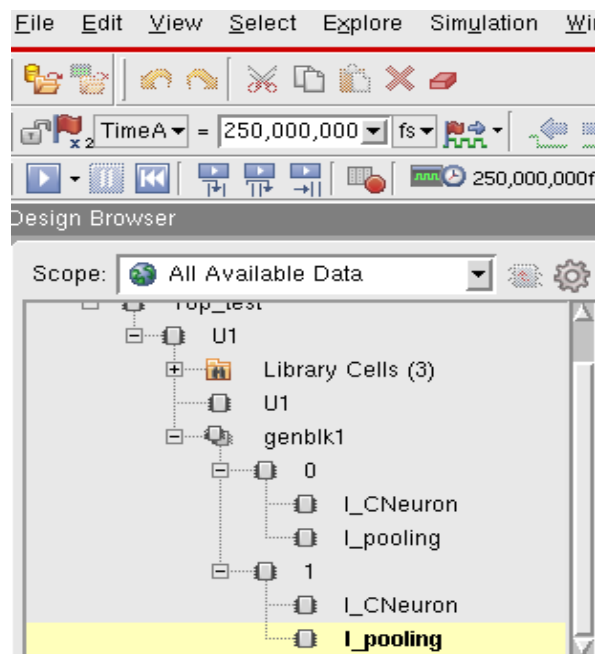
בעזרת **File->Save As** שמור את **NeuralNet_cont3.sv** לקובץ בשם **NeuralNet_cont4.sv**.
בעזרת **File->Save As** שמור את **Top3.sv** לקובץ בשם **Top4.sv**.
בעזרת **File->Save As** שמור את **Top3_test.sv** לקובץ בשם **Top4_test.sv**.

לקובץ **Top4.sv** הוסף 2 יחידות **Pooling** – אחד לכל **cneuron**. יש להוסיף אותם בעזרת אותו משפט **generate** מהסעיף הקודם. חשוב להשתמש בגרסה עם **\$.signed(convolution[i]**

- השתמש בקובצים בשם **Top4.sv** ו- **Top4_test.sv** שהכנת. הרץ סימולציה של המודול בעזרת הפקודה:

```
$ run_Top4
```

- בחלון של ה- **Design Browser** פתח את מבנה התכנון כפי שמתואר באיור הבא. בחר ביחידה **I_pooling** של יחידה 0. לחץ על כפתור ה- **Waveform** (איור 4.1). חזור על הפעולה עבור ה- **I_pooling** השני. פעולה זאת פותחת חלון עבור צורות הגל של כל הסיגנלים של הנוירונים.



- לחץ על כפתור ה- **Waveform** (איור 4.1). פעולה זאת פותחת חלון עבור צורות הגל של כל הסיגנלים של יחידות ה- **pooling**.
- לחץ על הכפתור **Play** (איור 4.2) שמופיע בחלק העליון של חלון ה- **Waveform** להרצת ה- **testbench**.

G41: שמור תמונת מסך של צורת הגל וצרף את התמונה לדו"ח.

Q41: בדוק והראה על גבי צורות הגל שכל תוצאות שני יחידות ה- **pooling** בהתאם לצפוי. (מופיעים כל הערכים של כל הצורות כמו באיור 17 בחומר ההכנה)

- שנה את ערך הסף ביחידת ה- **pooling** ל- 0. הרץ את הסימולציה שוב.

Q42: הסבר כיצד זה משפיע על התנהגות המערכת.

G42 : שמור תמונת מסך של צורת הגל וצרף את התמונה לדו"ח.

- הראה את התוצאות למדריך.
- שחזור את הערך המקורי של הסף (2).

5. הוספת יחידות ה-FC (Top5):

- צור גרסה חדשה (5) של כל התכנון באופן הבא.

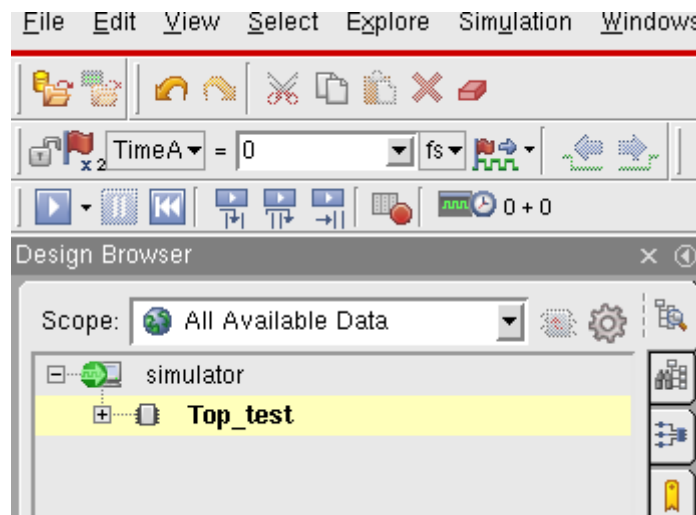
בעזרת File->Save As שמור את NeuralNet_cont4.sv לקובץ בשם NeuralNet_cont5.sv.
בעזרת File->Save As שמור את Top4.sv לקובץ בשם Top5.sv.
בעזרת File->Save As שמור את Top4_test.sv לקובץ בשם Top5_test.sv.


לקובץ Top5.sv הוסף את נגידון הסיווג fcneuron.

- השתמש בקובצים בשם Top5.sv ו- Top5_test.sv שהכנת. הרץ סימולציה של המודול בעזרת הפקודה :

```
$ run_Top5
```

- בחלון של ה- Design Browser בחר ביחידה Top_test



- לחץ על כפתור ה- Waveform () .. פעולה זאת פותחת חלון עבור צורות הגל של כל הסיגנלים של Top_test.

- לחץ על הכפתור Play () שמופיע בחלק העליון של חלון ה- Waveform להרצת ה- testbench.

G51 : שמור תמונת מסך של צורת הגל וצרף את התמונה לדו"ח.

Q51 : בדוק והראה על גבי צורות הגל שכל תוצאות הן בהתאם לצפוי.

- הראה את התוצאות למדריך.

על כל קטעי הקוד שהכנת להופיע בדוח (עם רקע לא כהה) ובנוסף יש להביא את כל הקוד בקבצים נפרדים לפי שמות המודולים השונים למעבדה.