The Andrew and Erna Viterbi Faculty of ELECTRICAL & COMPUTER ENGINEERING

Technion Israel Institute of Technology



מעבדה ב- VLSI ספרתי - 0450111

מימוש וסימולציות : דו״ח הכנה

http://www.ee.technion.ac.il/vlsi/

הערות נא לשלוח ל-goel@ee

כל הערה תתקבל בברכה!

עדכון אחרון - 11:07 19/09/2024

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

תוכן העניינים

דו"ח מכין פגישה מס' 1 – תכנון ומימוש.....3
תכנון ומימוש ארכיטקטורת ה- Datapath ...**Error! Bookmark not defined.**
מימוש קוד ה- SystemVerilog ו- Testbenches 11
1. מימוש מערך הרגיסטרים 11
2. מימוש נוירון הקונבולוציה 12
3. מימוש יחידת ה- Pooling 12
4. מימוש נוירון ה- FC 12
5. מימוש הבקר 12
6. בדיקת המערכת השלמה 12

הנחיות כלליות:

לפני הניסוי יש להכין דו״ח מכין בפורמט **pdf**. יש לצור קובץ **ZIP** (לא **RAR**) המכיל את דו״ח המכין ואת קיבצי הקוד שהכנת ולהעלות אותו ל- **Moodle** לפני תחילת הניסוי. ב- **Moodle** תוכל למצוא תבניות עבור כל הקבצים שעליך להכין.

חשוב : לפני הכנת דו״ח המכין חובה לקרוא לפי הסדר:

- חוברת הדרכה על הוראות הבטיחות: **SafetyInstructions.pdf**
- חוברת הסיכום של כל הניסוי : **DigitalSummaryVLSI.pdf**
- החוברת מידע על הפגישה הראשונה: **DigitalDesignInformation.pdf**
- חוברת הדרכה על **SystemVerilog**: **SystemVerilogManual.pdf**
- חוברת הדרכה על רשת הנוירונים: **NeuralNetworkDescription.pdf**
- הסברים על דו"ח מכין של הפגישה הראשונה: **PrepSim.pdf**

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 פגישה מס׳ 1 – תכנון ומימוש

תכנון ומימוש ארכיטקטורת ה- Datapath

המסמך על רשת הנוירונים תיאר את האלגוריתם. המיקוד של החלק הראשון של המעבדה 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 מיצג 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?

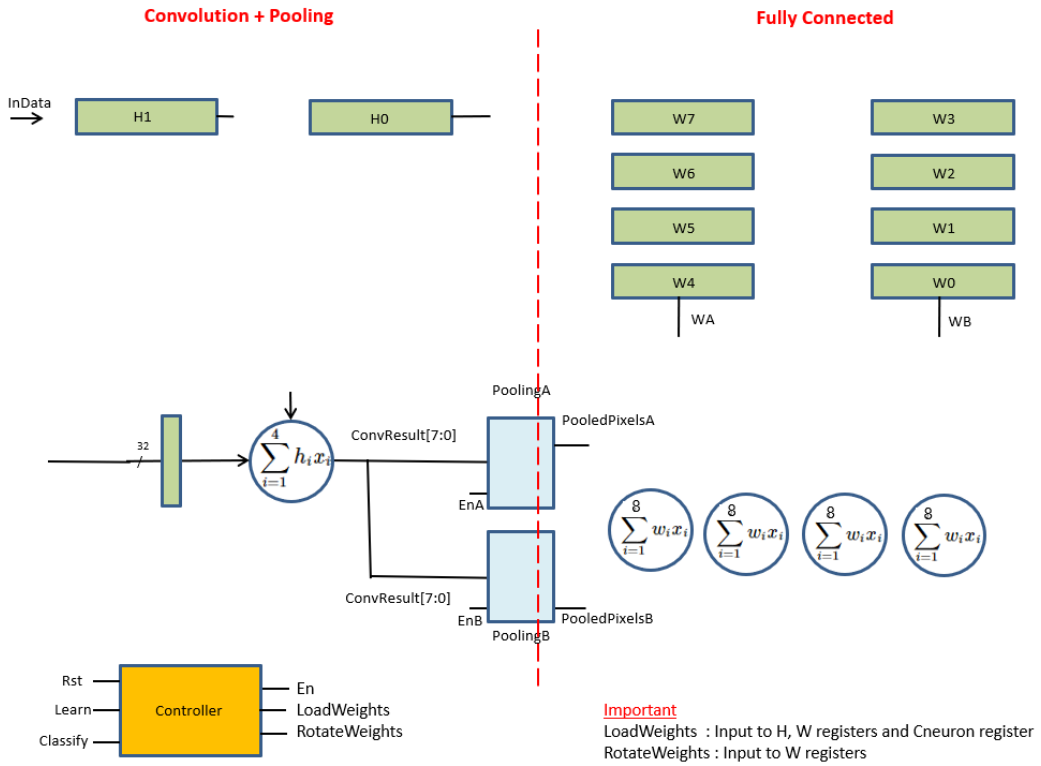
2. האם להשתמש בזיכרון SRAM או רגיסטרים לשמירת נתוני המסננים (המשקלים)?
זיכרון ה- SRAM שברשותנו מאפשר גישה לשני תאי זיכרון בו זמנית (באמצעות שני פורטים), רגיסטרים לעומת זאת, לכמה שרוצים.
זיכרון ה- SRAM חסכוני יותר בשטח כאשר מאחסנים יותר מ- 1000 סיביות. ב- **SRAM** נדרשים כ- 6 טרנזיסטורים לסיבית ברגיסטרים כ- 20.
זיכרון ה- SRAM חסכוני פחות בצריכת הספק.
3. האם לבצע את כל החישובים במחזור שעון אחד (חישוב מקבילי) או יותר (חישוב טורי/מצונר)?
חישוב מקבילי ידרוש מחזור שעון ארוך יחסי.
חישוב מצונר יאפשר שעון מהיר יותר אבל ידרוש יותר שטח (והספק).
4. בכמה נוירונים להשתמש בשלב הקונבולוציה ? 1, 2 או אולי 8 (2-מסננים כפול 4-פעולות)?
בכמה נוירונים להשתמש בשלב `Fully Connected` ? 1, 2 או אולי 4 (4-צורות)?
בכמה יחידות `pooling` להשתמש? 1, 2 ?
5. כמה כניסות ויציאות יהיה לשבב ?

<u>הערה חשובה :</u>
נסמן את מסנן הקונבולוציה ב **Hi**. כל מסנן כזה מורכב מ- **4** משקלים **hi** (כ״א **8** סיביות) ולכן, לכל **32 Hi** סיביות.
כל מסנן של שלב ה- **Fully Connected** מורכב מ- **8** משקלים **wi** (כ״א **8** סיביות) ולכן, לכל מסנן **64** סיביות. מכיוון שנרצה לעבוד עם רגיסטרים בעלי **32** סיביות, נשמור כל מסנן של שלב ה- **Fully Connected** בשני רגיסטרים שנסמן ב **Wi** וב- **Wi+1.**

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 <u>חישוב</u> <u>ואחסון</u> של המסננים האופטימאליים לכל חלקי המערכת. בניסיוי זה שלב הלימוד יכלול רק את <u>אחסון</u> נתוני המסננים בזיכרון. כלומר אנו מניחים שנתוני המסננים אינם מצויים על השבב מראש אלא במקום אחר, ולכן לצורך סיווג של צורות חדשות ראשית יש לטעון אותם לזיכרון.
ב. פאזת הסיווג – שלב זה כולל קבלת תמונה חדשה וביצוע הסיווג שלה לאחת מתוך 4 האפשרויות.
קיימות 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 (זה יכול לתת גמישות מסוימת בעתיד).
ב. עלינו לקלוט ולשמור שני מסננים בשלב הקונבולוציה (**2x32**) וארבעה בשלב **fully connected** (**4x64**). מדובר על **320** סיביות ולכן נשמור אותם ב **10** רגיסטרים בעלי **32** סיביות כ״א. (**H1,H2** ו- **W0..W7**)
ג. ננית גם שהפיקסלים של התמונה הנבחנת מגיעים מחוץ ליחידה ואין צורך לשמור את **כל** התמונה במערכת שלנו.

אלטרנטיבה אחת אפשרית למבנה המערכת יהיה כפי שמוצג באיור מס' 1:



איור מס' 1

נהוג לקלוט מידע המגיע לשבב ברגיסטר. הרגיסטר בכניסה לנוירון הקונבולוציה נועד לקליטת תתי התמונה שמגיעים ממקור חיצוני. פעולות נדרשות מהמערכת:

- טעינת המשקלים **Hi** עבור הנוירונים בשלב הקונבולוציה
- טעינת המשקלים **Wi** בשלב ה-**Fully Connected**
- חישוב שלב הקונבולוציה, שלב ה- **Pooling** ושלב ה-**Fully Connected**.

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

1. כמה כניסות (**input pins**) עבור ה- **Data** יהיו לשבב? כזכור, נדרש לספק לו **10** ערכים של **32** סיביות (עבור המסננים) + **32** סיביות עבור כל תת תמונה שמגיעה לשבב.
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 שתאפשר לכל החישוב להתבצע במקביל?

מספר כניסות ה- Data לשבב

כאמור, עלינו לטעון לשבב **320** סיביות עבור המסננים ולאחר מכן **32** סיביות עבור כל תת תמונה.

שאלות:

1. כמה **input pins** דרושים להכנסת המשקלים לשבב במחזור אחד?
2. אם רוצים להגביל ל 64 את מספר ה- **input pins** שמיועדים להכנסת המשקלים, כיצד ניתן לעשות זאת? כמה מחזורים דרושים להשלמת הכנסת המשקלים במקרה הזה?
3. אנו נראה בהמשך שגודל השבב משמעותית אם נשתמש ב- 64 **input pins** ולכן נחליט לעבוד עם 32 בלבד. כמה מחזורים דרושים להשלמת הכנסת המשקלים במקרה הזה?

תכנון מבנה הזיכרון

תכנון מבנה הזיכרון קשור מאד לצורת המימוש של שכבות הקונבולוציה וה- **Fully Connected (FC)**. ככל שמשתמשים ביותר נוירונים בשכבות אלה יש צורך בקריאת יותר נתונים (משקלי מסננים) מהזיכרון. (המשקלים הם האיברים שמרכיבים את המסננים).

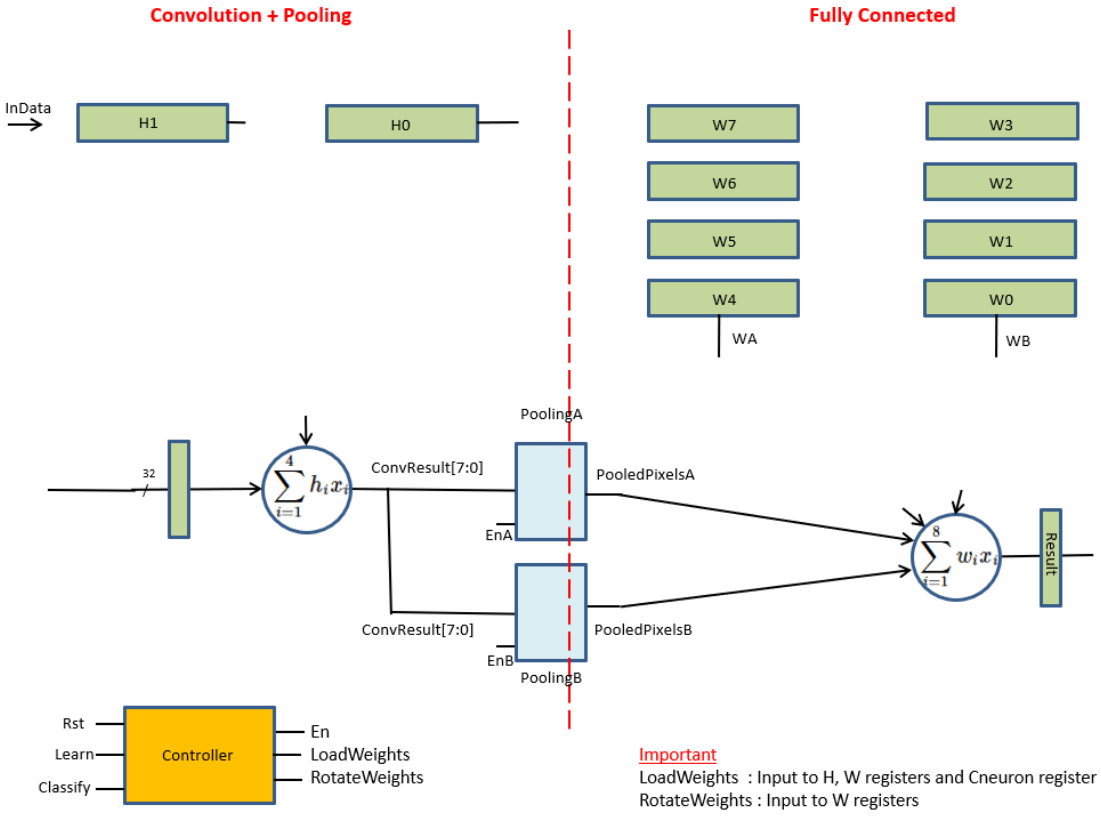
חשוב: הוחלט לייצג את המידע ב- **8** סיביות ולכן **8'bFF = -1** ו- **8'b01 = 1**

1. פאזת הלימוד **(Learn)** דורשת:
   a. כתיבה של H0 ו- H1 בעלות **32** סיביות כ"א עבור שכבת הקונבולוציה.
   b. כתיבה של W7..W0 בעלות **32** כ"א עבור שכבת ה-FC. השאיפה היא לקצר שלב זה ככל שניתן.
2. פאזת הסיווג **(Classify)** דורשת:
   a. אם מממשים את שלב הקונבולוציה בעזרת נוירון בודד אז נדרשת קריאה של מסנן בודד (**32** סיביות) מתוך מערך הרגיסטרים בזמן החישוב.
   b. אם מממשים את שלב הקונבולוציה בעזרת שני נוירונים אז נדרשת קריאה של **2** מסננים מתוך מערך הרגיסטרים בזמן החישוב.
   c. אם מממשים את שלב ה-**FC** בעזרת נוירון בודד נדרשת קריאה של מסנן אחד (**64** סיביות, ז"א **Wi** ו- **Wi+1**) מתוך מערך הרגיסטרים בכל מחזור שעון. נוירון זה יפעל במשך 4 מחזורים ויקבל את המסנן של צורה שונה בכל מחזור.
   d. אם ממשים את שלב ה-**FC** בעזרת 4 נוירונים נדרשת קריאה של ארבעה מסננים (כ"א של **64** סיביות) מתוך מערך הרגיסטרים בו זמנית. נוירונים אלה יפעלו במשך מחזור אחד ויקבלו את המשקלים של כל צורה וצורה בו זמנית.

נבחן כעת מספר אלטרנטיבות:

אופציה מס' 1:

1 Convolution Neurons, 1 Fully Connected (FC) Neuron



איור מס' 2

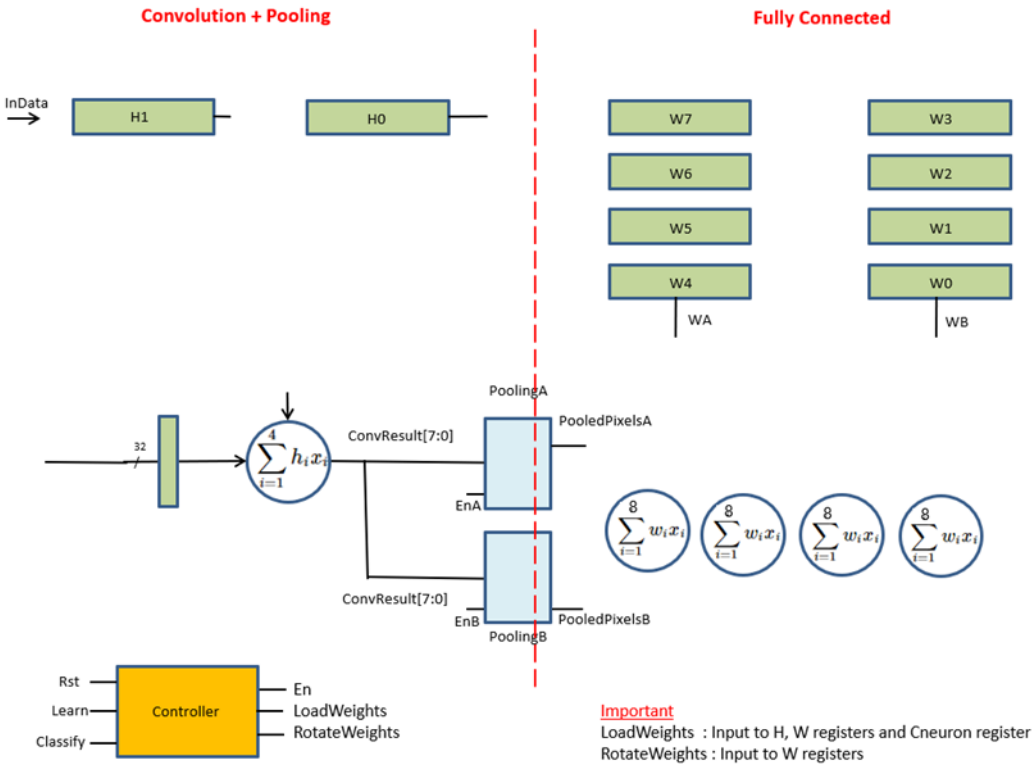
4. השלם את החיבורים של כל הרגיסטרים, כלומר החיבורים בין הרגיסטרים לנוירונים, בין רגיסטר לרגיסטר ובין הכניסה לרגיסטרים (כפי שנעשה באיור 4 בהמשך).

<u>חשוב :</u> מכיוון שיש רק נוירון בודד בשלב ה- **FC**, עלינו לדאוג שבכל מחזור שעון יגיע זוג שונה של **Wi** ליציאות **WA** ו- **WB** על ידי פעולת **SHIFT** כפי שנעשה באיור 4 בהמשך).

5. כמה מחזורים דרושים לביצוע החישוב ?
6. מה הבעיות עם המבנה הזה ?

אופציה מס' 2:

<u>1 Convolution Neurons, 4 Fully Connected (FC) Neuron</u>



איור מס' 3

7. השלם את החיבורים של כל הרגיסטרים, כלומר החיבורים בין הרגיסטרים לנוירונים, בין רגיסטר לרגיסטר ובין הכניסה לרגיסטרים. (כפי שנעשה באיור 4 בהמשך)
8. כמה מחזורים דרושים לביצוע החישוב ?
9. מה הבעיות עם המבנה הזה ?

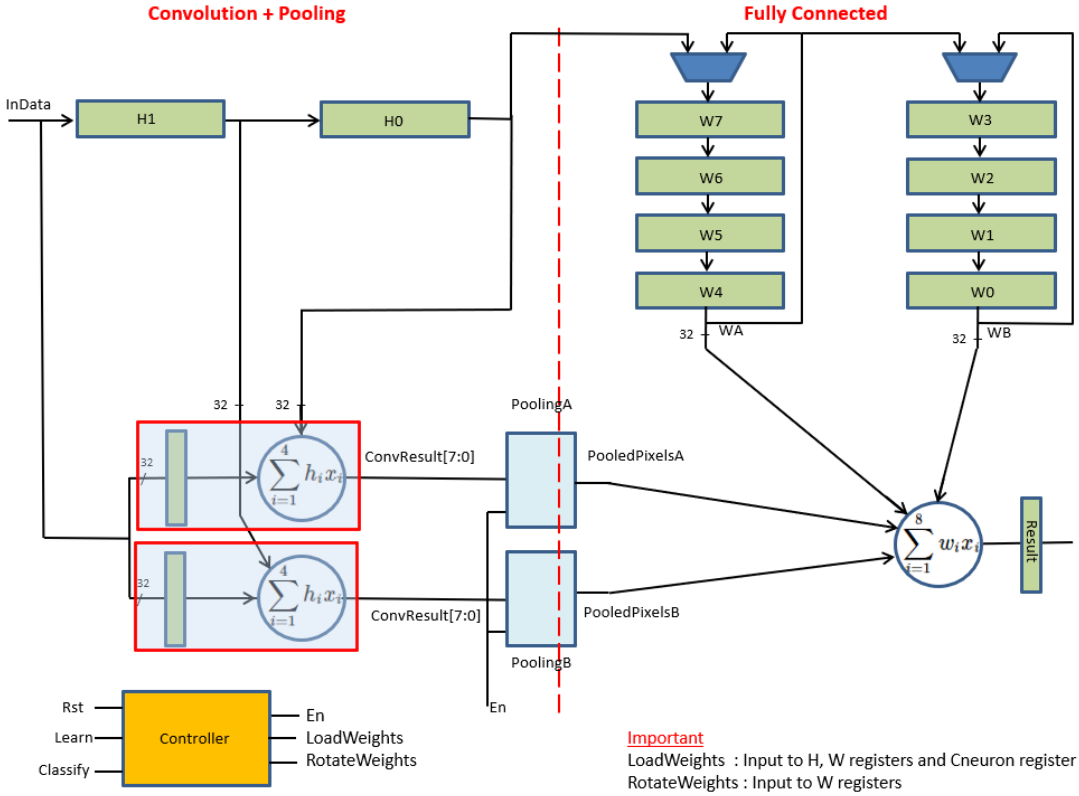
השלב של הקונבולוציה דורש מעבר על 4 איזורים של תמונת הכניסה עבור כל מסנן, ולכן אם נבחר להשתמש בנוירון אחד לכל מסנן (כלומר שני נוירונים), שלב זה ייקח 4 מחזורים להשלים את החישובים עבור כל התמונה. במימוש כזה, שלב ה- **FC** יספיק לבצע את כל ארבעה החישובים גם אם יש בו רק נוירון אחד. לכן נחליט לעבוד עם התצורה הבאה :

אופציה מס' 3:

2 Convolution Neurons, 1 Fully Connected (FC) Neuron

בתצורה זאת :

- שני הנוירונים בשכבת הקונבולוציה יכולים לעבוד במקביל כ"א עם משקלים שונים. המשקלים של שלב זה שמורים ב- **H0** וב- **H1**.
- 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. כאמור, רצוי לבצע זאת עם נוירון בודד המבצע את החישוב במהלך 4 מחזורי שעון. זה יוצר איזון בין שני חלקי החישוב **(Convolution+Pooling vs Fully Connected).**
- המשקלים של שכבת ה- **FC** שמורים ברגיסטרים **W7..W0** לפי המספר באיור 4.
- אם נחליט ששכבת ה- **FC** תבצע השוואה מול 4 הצורות לפי הסדר **\,/,0,X** אז נשמור את המשקלים שמתאימים ל- **X** ב **W0** וב- **W4** וכל היתר בהתאם.
- בשלב ה- **Fully Connected** חשוב לא לאבד את המידע ולכן לאחר השימוש, המשקלים מועברים דרך ה – **muxes** לסוף התור ומתבצע **shift** כלפי מעטה של כל הרגיסטרים.
- בעבודה אופטימלית שתי השכבות עובדות כל הזמן ומבצעות את החישוב שלהן ב- 4 מחזורי שעון. כאשר שכבת ה- **FC** פועלת על תמונה מסוימת, שכבת הקונבולוציה פועלת על התמונה הבאה.
- יש לשים לב שחלק מהחיבורים בין הרגיסטרים בשימוש בזמן הלמידה וחלק בזמן הסיווג כאשר כל השליטה על המערכת בידי ה- **Controller** (הבקר).



איור מס' 4

חשוב : כל זוג W שמייצג תמונה (לדוגמא W0 ו- W4) מורכב מ- 4 + 4 בתים. סימן ה- w_i שבנוירון ה- FC באיור 4 מייצג את משמונה הבתים.

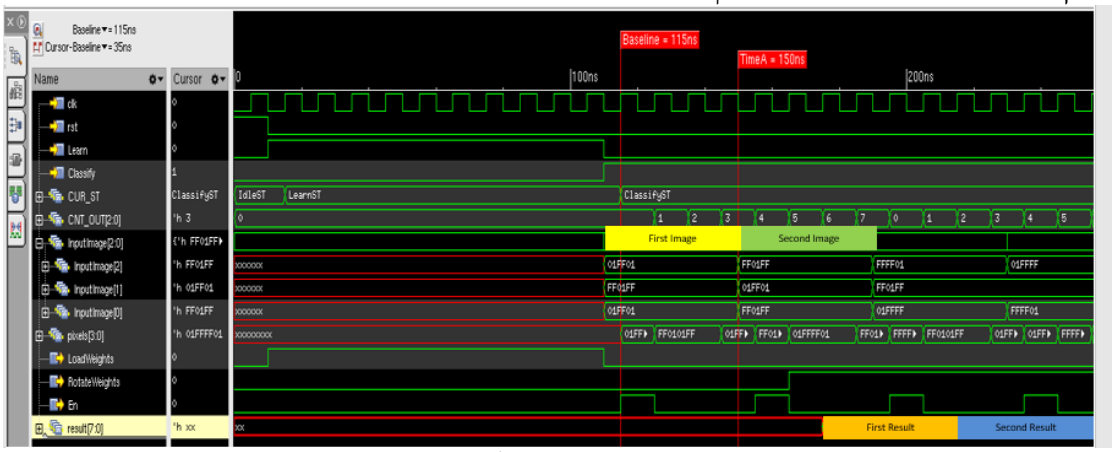
- פירוט מלא של כל סיגנלי הבקרה שמפיק הבקר יובא בהמשך.
- במחזור הראשון נבצע **reset** של הבקר.

- במהלך 10 מחזורים הבאים נבצע טעינה של המסננים **Hi** ו- **Wi** ברגיסטרים המתאימים. זה למעשה משלים את תהליך הלימוד.
- במחזור לאחר מכן נבצע טעינה של תת התמונה ביחידה **CNneuron**, שיכלול רגיסטר, ונתחיל בשלב הסיווג. במשך 4 מחזורים, דרך כניסת ה- **InData**, מגיעים 4 תתי האזורים של התמונה הראשונה אחד אחרי השני.

10. שאלה : הוחלט שלכל **CNeuron** נוסיף רגיסטר בכניסה. האם אפשר היה לחסוך משהו כאן ?

- כניסת ה- **InData** מגיעה לשני הנוירונים והתוצאות שכל נוירון מפיק מגיעות ליחידת- **Pooling** שלו ונאגרות במהלך 4 מחזורי שעון (ראה בהמשך).
- 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 שה- **Pooling** יוצרות.
- תיאור מבנה כל היחידות יובא בהמשך.

להלן דיאגרמת גלים שמציגה את התקדמות החישוב:



איור מס' 5

הערות :

- מחזור אחד של Reset
- 10 מחזורים של טעינת המסננים – נקבע ע"י Learn
- מעבר לשלב הסיווג - נקבע ע"י אות ה- Classify
- pixels : תת התמונה שנכנס ל- CNeuron.
- First Result : תוצאות הזיהוי של התמונה הראשונה

11. כמה מחזורים דרושים למערכת עד שהיא מזהה את הצורה הראשונה ? הסבר!
12. כמה מחזורים נוספים דרושים עד שמזהים את הצורה השנייה והשלישית ? הסבר!

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

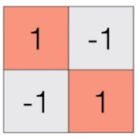
עבור "/" :



נייצג את (**-1**) כ- **ff** ואת (+1) כ- 01 באופן הבא .

```
H0 = 32'hff0101ff ;
```

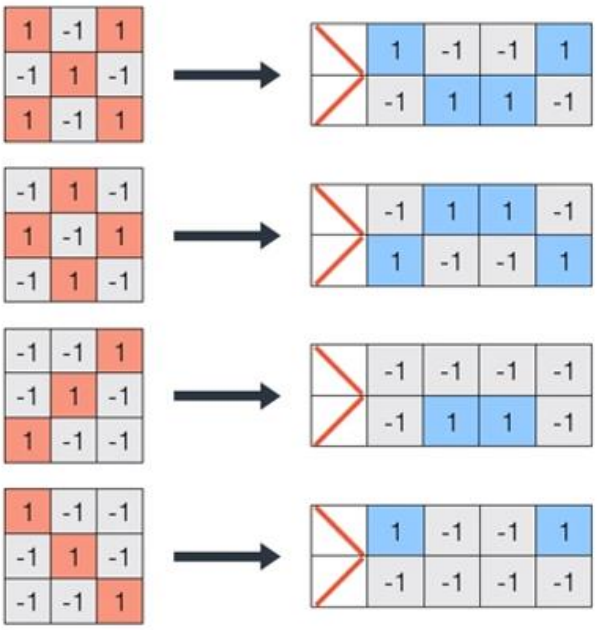
עבור "\" :
נקבל באופן דומה :




```
H1 = 32'h01ffff01 ;
```

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 מציור 17 (בסמסך ה- **NN**) :



```
W4 = 32'h01ffff01 ;  //Part A of X
W0 = 32'hff0101ff ;  //Part B of X

W5 = 32'hff0101ff ;  //Part A of 0
W1 = 32'h01ffff01 ;  //Part B of 0

W6 = 32'hffffffff ;  //Part A of /
W2 = 32'hff0101ff ;  //Part B of /

W7 = 32'h01ffff01 ;  //Part A of \
W3 = 32'hffffffff ;  //Part B of \
```

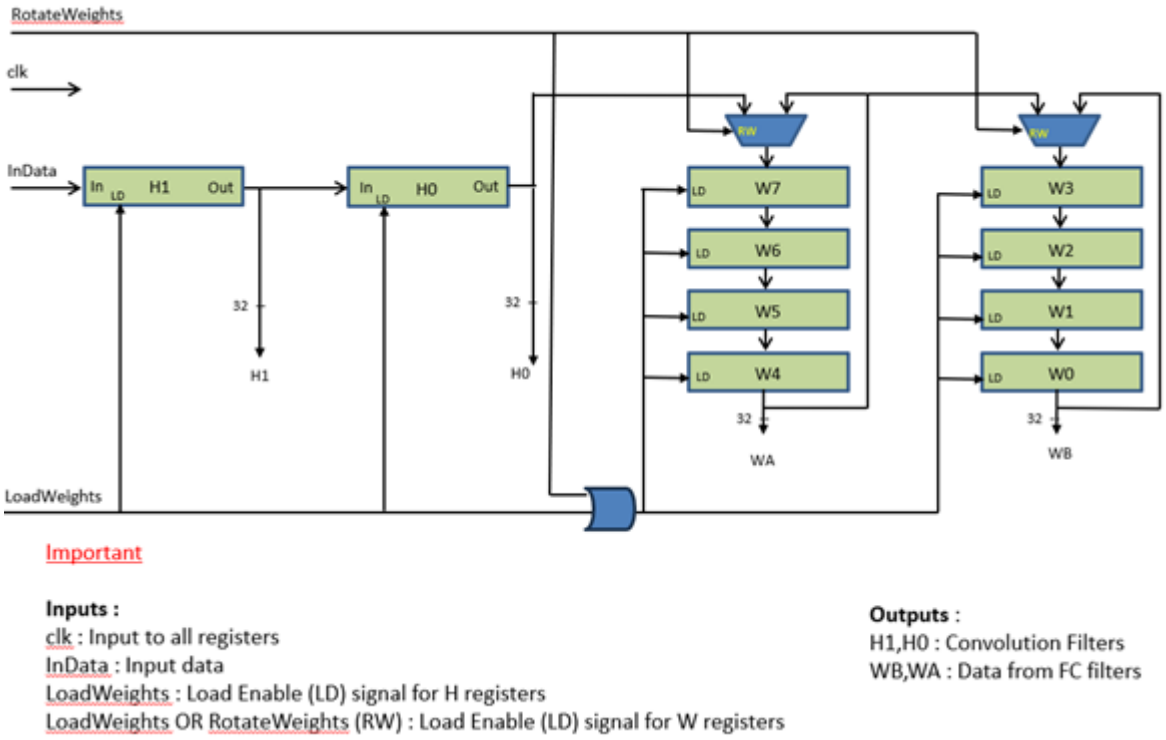
חשוב : בהתאם לאיור 4, בזמן סימולציה יש להזין את המסננים בסדר הנכון כלומר:
W0, W1, W2, W3, W4, W5, W6, W7, H0, H1

```
W0 = 32'hff0101ff ;  //Part B of X
W1 = 32'h01ffff01 ;  //Part B of 0
W2 = 32'hff0101ff ;  //Part B of /
W3 = 32'hffffffff ;  //Part B of \
W4 = 32'h01ffff01 ;  //Part A of X
W5 = 32'hff0101ff ;  //Part A of 0
W6 = 32'hffffffff ;  //Part A of /
W7 = 32'h01ffff01 ;  //Part A of \

H0 = 32'h01ffff01; //right filter looks for \
H1 = 32'hff0101ff; //left filter looks for /
```

כעת נעבור למימוש היחידות שממשים את המערכת.

1. מימוש מערך הרגיסטרים

תפקיד היחידה הזאת היא לשמור ערכי המסננים : **Hi** ו- **Wi.** כאמור טוענים את רגיסטרים אלה מיד אחרי ה- **reset** בשלב ה- **Learn**.
באיור הבא מתואר המבנה של מערך הרגיסטרים :



איור מס' 6

כאמור טוענים את רגיסטרים אלה מיד אחרי ה- **reset** בשלב ה- **Learn**. הערכים יוכנסו דרך **InData.** כאשר אות ה- **LD** של הרגיסטרים גבוהה, כל רגיסטר טוען את המידע שיש לו בכניסה. אות הבקרה **LoadWeights** יוזן לכניסת ה- **LD** של רגיסטרי ה- **H**, ותוצאת ה- **OR** בין אותות הבקרה **LoadWeights** ו- **RotateWeights** יוזן לכניסת ה- **LD** של רגיסטרי ה- **W.** אות הבקרה **RotateWeights** יקבע האם רגיסטרי ה- **W** יבצעו טעינה ראשונית של המשקלים או רוטציה של המשקלים שנטענו בשלב קודם. מהסכמה ניתן לראות שעל סדר הכנסת הנתונים להיות :

```
WB of X
WB of 0
WB of /
WB of \
WA of X
WA of 0
WA of /
WA of \
```

1.1 רשום את הקוד הבא בקובץ בשם **Weights.sv** והשלם את קטעי הקוד טעינת המסננים וביצוע רוטציה בזמן החישוב.

```
module Weights (
    input clk, rst, LoadWeights, RotateWeights,
    input [31:0]  InData,
    output [31:0] H1,H0,
    output [31:0] WA,WB);

    integer i;
    logic [1:0] [31:0] H;
    logic [7:0] [31:0] W;

    always_ff @(posedge clk or posedge rst)
    begin
      if (rst == 1'b1)
        begin
           for (i=0; i<8; i=i+1)
              W[i] <= 32'b0000;
              H[1] <= 32'b0000;
              H[0] <= 32'b0000;
        end
        else
        begin
         if (LoadWeights == 1'b1)
          begin
           //Add code to load H-filter values
          end

         if ((LoadWeights == 1'b1) || (RotateWeights == 1'b1))
          if (RotateWeights == 1'b0)
            begin
             //Add code to load W-filter values
            end
         else
          if (RotateWeights == 1'b1)
           begin
            //Add code to rotate W-filter values
           end
        end

    end

    assign H1 = H[1];
    assign H0 = H[0];
    assign WA = W[4];
    assign WB = W[0];
endmodule
```

כאמור, ה- **testbench** זה הקובץ שמכיל את סביבת הסימולציה. קובץ זה מכיל את הקוד שיוצר את אותות הכניסה למעגל וגם מכיל אינסטנציאציה של המעגל שבבדיקה **(Weights)**.

1.2 שים לב שה-**testbench** בנוי כך שהכניסות משתנות בירידת השעון. למה זה חשוב ?

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

- קוד לקביעת ערך ה- rst ל- 1 במשך המחזור הראשון ולאחר מכן 0.
- קוד לקביעת הערכים הנכונים ל- LoadWeights ו-RotateWeights לטעינת ה- InData במשך 10 מחזורים.
- קוד שמכניס את הערכים 1 עד A (10 בהקסה) לכניסת ה- InData, ערך אחד כל מחזור שעון.

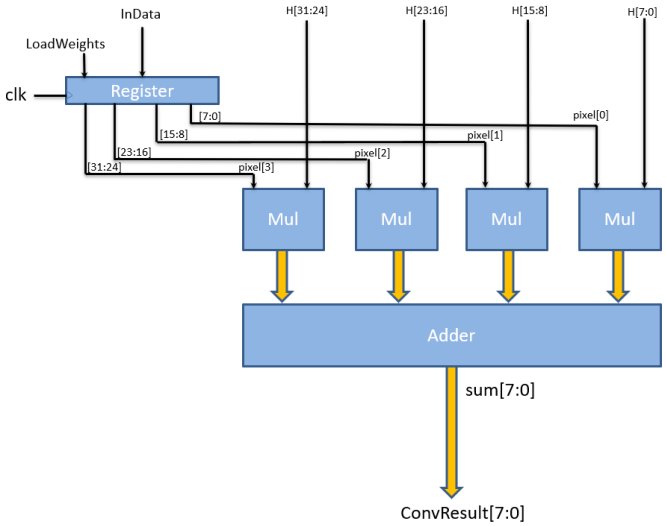
מוסיפים משפט **$finish** כדי להימנע מ-**testbench** אשר אינו מפסיק לעולם.

```
module Weights_test;
logic clk, rst, LoadWeights, RotateWeights;
logic  [31:0] InData;
logic  [31:0] H1,H0;
logic  [31:0] WA,WB;

Weights I_Weights(.*);

always begin
 #5 clk = ~clk;
End

initial begin
clk = 1'b0 ;
//Set the rst value
@(negedge clk);
//Set values of : rst, LoadWeights, RotateWeights
repeat(10) @(negedge clk);
//Set new values of: LoadWeights, RotateWeights
#100 $finish;
end

initial
begin
@(negedge clk);
InData = 32'h00000001 ;
//Set 9 additional values of InData, each at a new negedge
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

2. מימוש נוירון הקונבולוציה

באיור הבא מתואר המבנה של הנוירון בשכבת הקונבולוציה :



איור מס' 7

2.1 רשום את הקוד הבא בקובץ בשם **CNeuron.sv** והשלם את קטע הקוד לחישוב **sum** ללא שימוש ב FOR LOOP.

$$\mathbf{sum = \sum H[i] * pixels[i]}$$

```
module CNeuron (
    input clk, LoadWeights,
    input [31:0] H,
    input [31:0] InData,
    output logic [7:0] ConvResult);

    logic [3:0] [7:0] pixels;
    logic [7:0] sum;
    integer i,j;

    always_ff @(posedge clk)
     begin
       if (LoadWeights  == 1'b0)
         for (i=0; i<4; i=i+1) begin
             pixels[i] <= InData[(i*8)+7 -: 8];
         end
     end

    always_comb
     begin
       // Code for computing sum
     end
     assign ConvResult = sum;
endmodule
```

הערה : שים לב ש- **Indata** נדגם כאשר **LoadWeights=0**.

2.2 הסבר את כל הקוד ב- **for loop**.

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

- בכל מחזור נרצה לדגום את **InData** ולהכפיל אותו ב **H**.
- ערכים נוספים עבור **InData** עבור שני הערכים שונים של **H** כאשר הערכים משתנים כל **10ns**.

השתמש בערכים הבאים עבור ה – **InData** :

```
{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- **H** למעשה מכיל את 4 הערכים של מסנן הקונבולוציה.

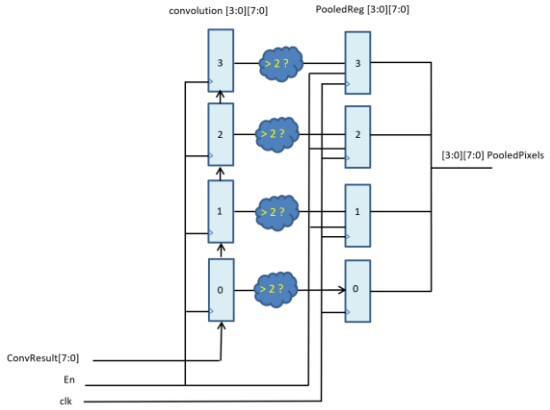
- יש להשתמש ב: **@negedge** לתזמן את משפטי ההשמה. ראה דוגמא בקוד.
- בדוגמא זאת נקבע ש- **LoadWeights** יהיה 0 קבוע כך שהיחידה תחשב בכל מחזור שעון.
- מוסיפים משפט **$finish** כדי להימנע מ-**testbench** אשר אינו מפסיק לעולם.

```
module CNeuron_test;
  logic clk;
  logic LoadWeights;
  logic [3:0] [7:0] InData;
  logic [7:0] ConvResult;
  logic [31:0] H;

  CNeuron I_CNeuron
  (
  .clk(clk),
  .LoadWeights(LoadWeights),
  .H(H),
  .InData(InData),
  .ConvResult(ConvResult));

  always begin
   #5 clk = ~clk;
  end

    initial begin
    clk = 1'b0 ;
    H = 32'h01ffff01;
    LoadWeights = 1'b0 ;
                  InData = {8'h01, 8'hff, 8'hff, 8'h01};
    // Set remaining values for InData for next 3 clock cycles
    @(negedge clk); InData = …
    @(negedge clk); InData = …
    @(negedge clk); InData = …

    @(negedge clk); H = 32'hff0101ff;
                  InData = {8'h01, 8'hff, 8'hff, 8'h01};
   // Set remaining values for InData for next 3 clock cycles
```

```
    #20 $finish;
  end

endmodule
```

3. מימוש יחידת ה- Pooling

- ראשית, נציין שהיחידה אוגרת את שלוש תוצאות הקונבולוציה הראשונות.
- עם הגעת התוצאה הרביעית, היחידה משווה 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.



איור מס׳ 9

3.1 רשום את הקוד הבא בקובץ בשם **Pooling.sv** והשלם את קטע הקוד לצבירת תוצאות הקונבולוציה וטעינת הערכים הנכונים ל- **PooledReg**.

```
module Pooling ( input clk, En,
    input [7:0]  ConvResult,
    output logic [3:0][7:0] PooledPixels);

    integer i;
    logic [3:0] [7:0] convolution;
    logic [3:0] [7:0] PooledReg;

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

```
    always_ff @(posedge clk)
    begin
       if (En == 1'b1)
           //Add code for loading pooled convolution results into "pooledReg" registers
    end

     assign PooledPixels = PooledReg;
endmodule
```

3.2 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

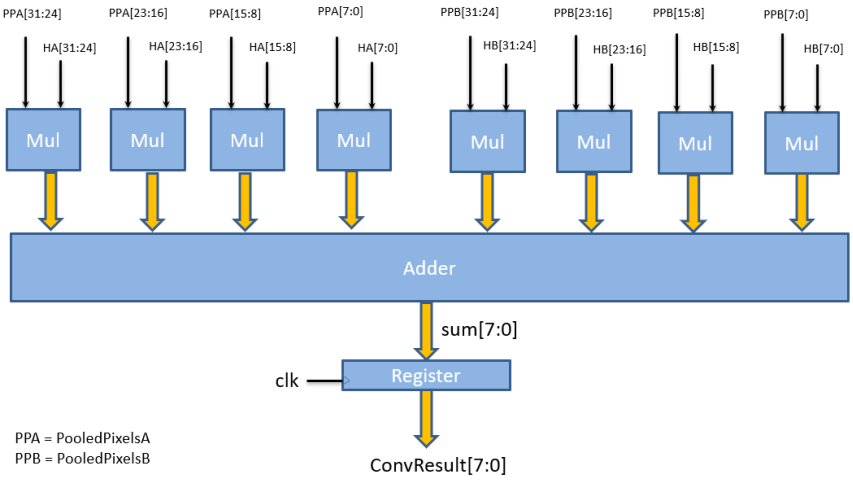
- יש לשים לב להתנהגות/תזמון של אות ה- **En** כך שהשוות של **ConvResult** תתבצע בזמן הנכון.

```
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 for next 6 values of ConvResult
#80 $finish;
end

initial begin
//Enable
En = 1'b0;
repeat(4) @(negedge clk);
En = 1'b1;
@(negedge clk);
En = 1'b0;
// Add code that makes En=1 every 4 clock cycles and the returns
```

```
// to 0 as above.
// Do this for the next 8 clock cycles
end

always begin
#5 clk = ~clk;
end

endmodule
```

4. מימוש נוירון ה- FC

באיור הבא מתואר המבנה של הנוירון בשכבת ה- **FC** :



איור מס׳ 9

4.1 רשום את הקוד הבא בקובץ בשם **FCNeuron.sv** והשלם את קטע הקוד לחישוב **sum**.

```
module FCNeuron ( input clk,
    input [31:0] PooledPixelsA,
    input [31:0] PooledPixelsB,
    input [31:0] WA,WB,
    output logic [7:0] result);

    logic [7:0] sum;
    integer i;

    always_comb
     begin
       // Add code to compute the sum
     end

    always_ff @(posedge clk)
        begin
```

```
            result <= sum;
        end
endmodule
```

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

- 4 ערכים שונים עבור **WA** ו- **WB**:

```
X
WA[31:0] = 32'h01ffff01 ;
WB[31:0] = 32'hff0101ff ;
0
WA[31:0] = 32'hff0101ff ;
WB[31:0] = 32'h01ffff01 ;
/
WA[31:0] = 32'hffffffff ;
WB[31:0] = 32'hff0101ff ;
\
weight[31:0] = 32'h01ffff01 ;
weight[31:0] = 32'hffffffff ;
```

- השלם את שלשת הערכים הנוספים עבור **pooledPixelArrayA** ו- **pooledPixelArrayB** (כפי שאתה מצפה לקבל עבור צורה).
- משפט **$finish** כדי להימנע מ-**testbench** אשר אינו מפסיק לעולם.

בסימולציה הזאת, נרצה לקבל ערכים של **pooledPixelArrayA** ו- **pooledPixelArrayB** עבור צורה מסוימת ובכל מחזור במשך ארבעה מחזורים להכפיל אותם עם ארבעה הסטים האפשריים של **WA** ו- **WB**.

```
module FCNeuron_test;

  logic clk;
  logic [31:0] PooledPixelsA;
  logic [31:0] PooledPixelsB;
  logic [31:0] WA,WB;
  logic [7:0] result;

  FCNeuron I_FCNeuron
  (
  .clk(clk),
  .PooledPixelsA(PooledPixelsA),
  .PooledPixelsB(PooledPixelsB),
  .WA(WA), .WB(WB),
  .result(result)
  );

  initial begin
    clk = 1'b0 ;
```

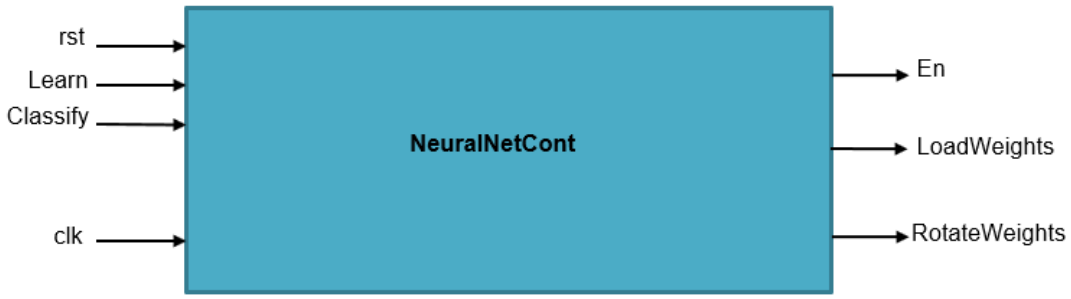
```
  end

  always begin
    #5 clk = ~clk;
  end

  always begin
    //Add WA/WB for X
    @(negedge clk);
    //Add WA/WB for 0
    @(negedge clk);
    //Add WA/WB for /
    @(negedge clk);
    //Add WA/WB for \
    @(negedge clk);;
  end

  initial begin
    // PooledPixelsA/B values for X
    PooledPixelsA = 32'h01ffff01 ;
    PooledPixelsB = 32'hff0101ff ;
    repeat(4) @(negedge clk);
    // Add PooledPixelsA/B values for 0

    repeat(4) @(negedge clk);
    // Add PooledPixelsA/B values for /

    repeat(4) @(negedge clk);
    // Add PooledPixelsA/B values for \

    repeat(4) @(negedge clk);
    #20 $finish;
  end

endmodule
```

4.3 מדוע המשקלים רשומים במשפט **always** ולא **initial** ?

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

כעת נתחיל במימוש של הבקר ונמשיך עם הרכבת המערכת כולה.

5. מימוש הבקר



איור מס׳ 10 : מבנה הבקר

כניסות הבקר:

clk : אות השעון

rst : אות ה- **reset**

Learn : **Learn=1** מסמן שיש לעבור למצב למידה בעלית השעון הבא.

Classify : **Classify=1** יחד עם **Learn=0** מסמן שיש לעבור למצב סיווג בעלית השעון הבא.

יציאות הבקר:

En: אות בקרה ליחידת ה- **Pooling**

LoadWeights: אות לטעינת המשקלים בשלב ה- **Learn**

RotateWeights: אות לביצוע רוטציה של המשקלים בשלב ה- **Classify**

דיאגרמת הגלים הבאה מציגה כיצד עלינו להגדיר את הכניסות **rst**, **Learn** ו- **Classify** ואלו ערכים להפיק בקווי הבקרה **En**, **LoadWeights** ו- **RotateWeights**.

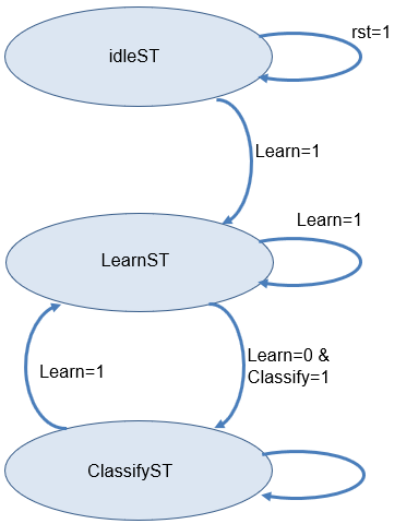
מצבי מכונת המצבים:

IdleST: מצב סרק

LearnST : מצב לימוד

ClassifyST : מצב סיווג

להלן טבלת המעברים של מכונת המצבים:



rst=1 וכל מצב לא מוגדר של Learn ו- Classify תמיד מוביל למצב IdleST

איור מס׳ 11 : דיאגרמת המצבים של הבקר

	Inputs	Next_St	Outputs

Current State	rst	Learn	Classify	CNT_OUT[1:0]		LoadWeights	RotateWeights	En	CNT_IN
IdleST	1	X	X	X	IdleST	X	X	X	0
IdleST	0	1	X	X	LearnST	1	0	X	X
IdleST	0	0	X	X	IdleST	0	0	X	X
LearnST	1	X	X	X	IdleST	0	0	0	0
LearnST	0	1	X	X	LearnST	1	0	0	X
LearnST	0	0	1	X	ClassifyST	0	0	0	X
LearnST	0	0	0	X	IdleST	0	0	0	X
ClassifyST	1	X	X	X	IdleST	0	0	0	0
ClassifyST	0	X	X	0	ClassifyST	0	1 *	1**	CNT_IN = CNT_OUT + 1

הערות חשובות :

* : על **RotateWeights** לעלות ל "1" חמישה מחזורים אחרי כניסה למצב **ClassifyST** כלומר, כאשר יחידית ה- **Pooling** דוגם את הערכים הראשונים.

** : על **En** להיות "1" כל ארבע מחזורים כדי לאפשר לרגיסטרי היציאה של להתעדכן במחזור הנכון, כלומר, כאשר יחידת הקונבולוציה מסיימת לעבור על כל 4 תתי התמונות.

על מנת להקל על יצירת סיגנלים אלו, נשלב בבקר את המונה **CNT**.

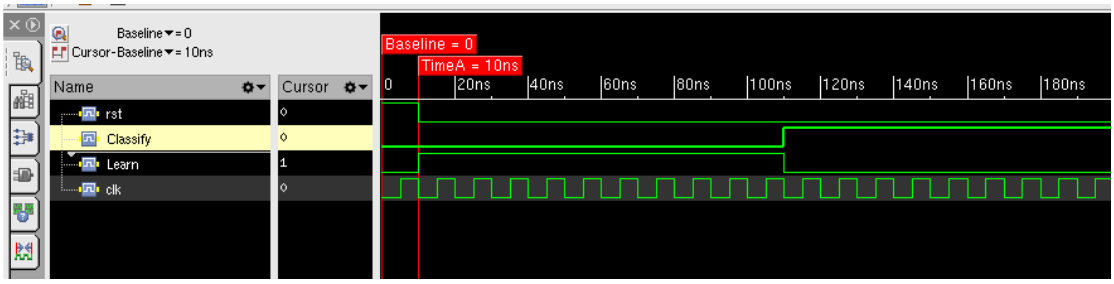
קוד למימוש מכונת המצבים

5.1 להלן המימוש החלקי המופיע להלן לקובץ בשם **NeuralNetCont.sv** . השלם את המימוש של מכונת המצבים בהתאם לטבלה ודיאגראמת המצבים באיור מס' 31. עליך להשלים את הקוד עבור המצב **LearnST**.

```
module NeuralNetCont (
    input clk, rst, Learn, Classify,
    output logic En, LoadWeights, RotateWeights
);

typedef enum bit[2:0] {IdleST = 3'b000, LearnST = 3'b001,
ClassifyST = 'b010} STATE;
STATE CUR_ST;
STATE NEXT_ST;
logic [2:0] CNT_IN, CNT_OUT;

always_ff @(posedge clk or posedge rst)
begin
    if (rst == 1) begin
        CUR_ST <= IdleST;
        CNT_OUT <= 3'b000;
        RotateWeights  <= 1'b0;
    end
    else begin
        CUR_ST <= NEXT_ST;
        CNT_OUT <= CNT_IN;
        if (CNT_OUT == 3'b100)
```

```
            RotateWeights  <= 1'b1;
    end
end

always_comb
begin
//default values
    En  = 1'b0;
    LoadWeights  = 1'b0;
    CNT_IN = 3'b000;
    NEXT_ST = IdleST;

    case(CUR_ST)
     IdleST:
     begin
        if (Learn == 1'b1)
           begin
             NEXT_ST = LearnST;
             LoadWeights  = 1'b1;
           end
     end

     LearnST:
     begin
//Complete the lines in red
          LoadWeights  =
          NEXT_ST =
          if (Learn == 1'b1)
             NEXT_ST =
          else
             if (Classify == 1'b1)
             begin
                NEXT_ST =
                LoadWeights  =
             end
     end

     ClassifyST:
     begin
          NEXT_ST = ClassifyST;
          CNT_IN = CNT_OUT + 3'b001;
          if (CNT_OUT[1:0] == 2'b00)
             En  = 1'b1;
     end
    endcase
end
```

5.2 רשום קובץ **testbench** בשם **NeuralNetCont_test.sv** לבדיקת המימוש. עליך להעתיק את המימוש החלקי המופיע בהמשך ולהוסיף קוד שמייצר את הסיגנלים rst, Classify ו- Learn (לפני משפט ה- $finish) בהתאם לאיור הבא.



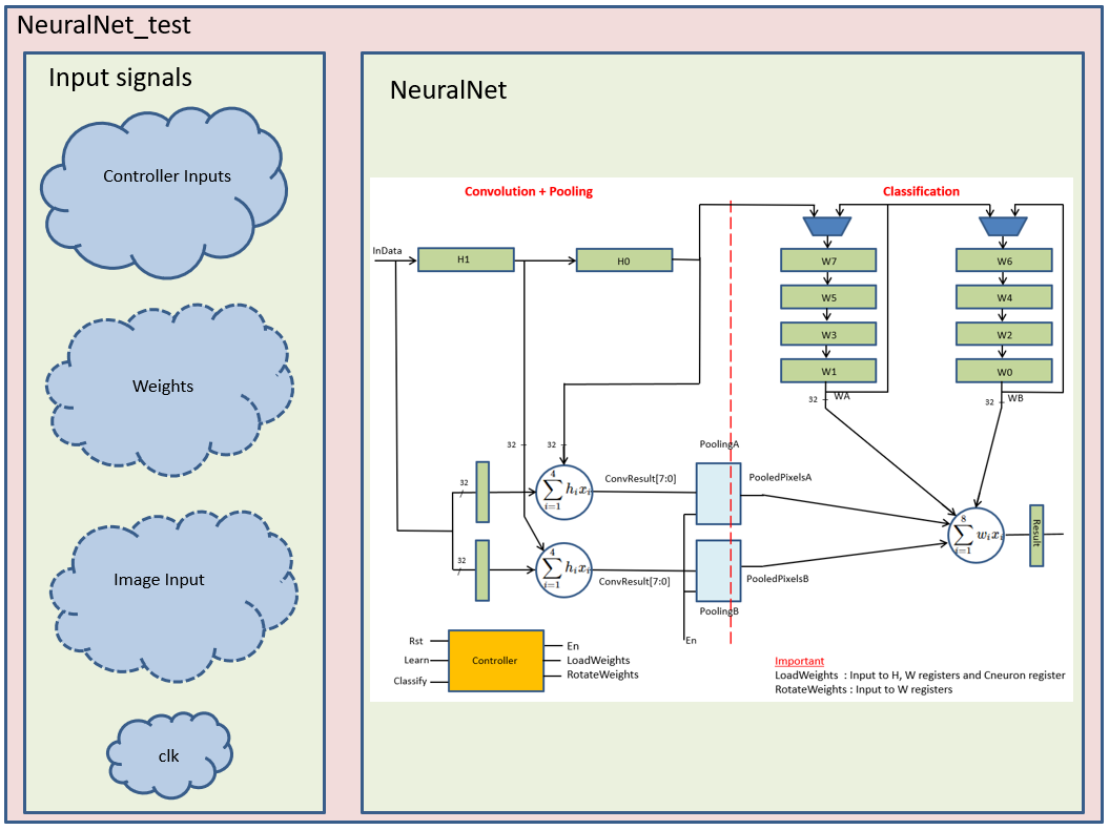

```
module NeuralNetCont_test;

logic clk;
logic rst, Learn, Classify;
logic En, LoadWeights, RotateWeights;

NeuralNetCont NNC_U1(.*);

initial begin
#200 $finish;
end

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

5.3 הסבר את המשפט :

```
NeuralNet NN_U1(.*);
```

6. בדיקת המערכת השלמה

כעת נרכיב את כל המערכת במודול בשם NeuralNet ונבנה סביבו testbench שיאפר לנו לבדוק אותו כפי שמתואר באיור מס׳ 5.



איור מס' 12

6.1 להלן המימוש החלקי המופיע להלן לקובץ בשם **NeuralNet.sv** . עליך להשלים החיבורים של יחידות ה **CNeuron**, **Pooling** ו- **FCNeuron**. העזר בדוגמא של יחידת ה- **Weights** המופיע במימוש הבא.

```
module NeuralNet ( input clk,
                   input rst, Learn, Classify,
                   input [31:0] InData,
                   output logic [7:0] result);
logic [7:0] ConvResult0;
logic [7:0] ConvResult1;
logic [3:0] [7:0] PooledPixelsA;
logic [3:0] [7:0] PooledPixelsB;
logic [31:0] H0;
logic [31:0] H1;
logic [31:0] WA;
logic [31:0] WB;
logic En,LoadWeights,RotateWeights;

//instantiation of NeuralNetCont
NeuralNetCont NNC_U1(.*);
```

```
Weights WTS(.clk(clk), .rst(rst),
.LoadWeights(LoadWeights),.RotateWeights(RotateWeights),
.InData(InData), .H0(H0), .H1(H1), .WA(WA), .WB(WB));

CNeuron CN0(fill in all the ports and their connections);
CNeuron CN1(fill in all the ports and their connections);

Pooling P0(fill in all the ports and their connections);
Pooling P1(fill in all the ports and their connections);

FCNeuron FCN(fill in all the ports and their connections);

endmodule
```

6.2 להלן מימוש חלקי של ה- **testbench**. רשום את הקוד בקובץ בשם **NeuralNet_test.sv**.
עליך להוסיף קוד עבור:

- סיגנלי הכניסה לבקר
- 6 **Weights** הנותרים
- 2 **InputImage** הנותרים

```
module NeuralNet_test;

logic clk;
logic rst, Learn, Classify;
logic [31:0] InData;
logic [2:0] [7:0] InputImage[2:0];
logic [7:0] result;

NeuralNet NN_U1(.*);


initial begin
// Add the code you use for NeuralNetCont_test
#220 $finish;
end

//Input the filter values in the correct order
initial
begin
repeat(1) @(negedge clk);
InData = 32'hff0101ff ; //WB of X
@(negedge clk);
InData = 32'h01ffff01 ; //WB of 0
@(negedge clk);
// Add 2 more WB weights and 4 more WA weight values
@(negedge clk);
InData = 32'h01ffff01 ;  // right filter that looks for \
@(negedge clk);
```

```
InData = 32'hff0101ff ;  // left filter that looks for /
end

initial begin
repeat(11) @(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} ;
InData = {InputImage[2][2],InputImage[2][1],InputImage[1][2],InputImage[1][1]};
@(negedge clk);
InData = {InputImage[2][1],InputImage[2][0],InputImage[1][1],InputImage[1][0]};
@(negedge clk);
InData = {InputImage[1][2],InputImage[1][1],InputImage[0][2],InputImage[0][1]};
@(negedge clk);
InData = {InputImage[1][1],InputImage[1][0],InputImage[0][1],InputImage[0][0]};
@(negedge clk);
InputImage[2] = {8'hff,8'h01,8'hff} ;
InputImage[1] = {8'h01,8'hff,8'h01} ;
InputImage[0] = {8'hff,8'h01,8'hff} ;
InData = {InputImage[2][2],InputImage[2][1],InputImage[1][2],InputImage[1][1]};
@(negedge clk);
InData = {InputImage[2][1],InputImage[2][0],InputImage[1][1],InputImage[1][0]};
@(negedge clk);
InData = {InputImage[1][2],InputImage[1][1],InputImage[0][2],InputImage[0][1]};
@(negedge clk);
InData = {InputImage[1][1],InputImage[1][0],InputImage[0][1],InputImage[0][0]};
@(negedge clk);
// Complete for remaining two InputImage!
@(negedge clk);
repeat(12) @(negedge clk);
$finish;
end

always begin
 #5 clk = ~clk;
end

endmodule
```

לפני כל ניסוי יש להכין דו״ח מכין בפורמט **pdf**. יש לצור קובץ **ZIP** (לא **RAR**) המכיל את דו״ח המכין ואת קיבצי הקוד שהכנת ולהעלות אותו ל- **Moodle** לפני תחילת הניסוי.