

פרויקט גמר - מערכת FPGA ותוכנה

הפקולטה להנדסה

המחלקה להנדסת חשמל ואלקטרוניקה

# מײַץ חומרה לאלגוריתם SHA-256

שם הסטודנט: אלעזר בילינקי

ת.ז: 206838237

שם המנחה: פרופסור יואל רצabi

מרץ 2025

## תקציר

פרויקט זה עוסק בIMPLEMENTATION של גבי כרטיס QYNQ SHA-256 עלbasis Jupyter. תוך שימוש בממשק Vivado. מטרתו המרכזית של הפרויקט היא להציג את חישוב ה-Hash בעולמה חינונית בעולם הדיגיטלי של ימינו, שבו מהירות עיבוד היא משאכ קריטי.

היתרון העיקרי בIMPLEMENTATION חישוב ה-Hash בחומרה טמון בניצול יכולות החישוב המקבילי של רכיבי FPGA. בנייתו לחישוב תוכנות המבוצע באופן סדרתי על מעבדים כליליים (CPU), עיבוד מבוסס חומרה מאפשר ריבוי תהליכי בו-זמןית, מה שmobiel לשיפור ניכר ב מהירות החישוב. בפרויקט זה, תהליך חישוב ה-Hash פועל במספר שלבים מקבילים, מה שמקטין את זמן העיבוד הכלול בהשוויה לפתרונות מבוססי תוכנה.

מעבר ליישום הפרויקט, הפרויקט מדגים כיצד ניתן לנצל טכנולוגיות מתקדמות, דוגמת כרטיסי A-FPGAs ופלטפורמות QYNQ, לטובת ביצוע חישובים מורכבים בצורה יעילה יותר. שילוב של חומרה ותוכנה בסביבה פתוחה כמו QYNQ מאפשר למפתחים ומהנדסים להציג וליעילizar חישובים מגוונים, תוך שימוש בכלים מודרניים לפיתוח חומרה.

באמצעות הפרויקט זה אנו מראים כי שילוב נכון בין חומרה לתוכנה יכול להביא ל垦פיצת מדרגה בביצועים, ולספק פתרון מהיר ויעיל יותר עבור משימות חישוב כבדות.

## הכרת תודה

ברצוני להביע את תודהי והערכתני לכל מי שהלז אוטו, תמרק ושייע לאורך הדרך, הן במהלך תכנון וביצוע הפרויקט הגמר והן לאורך כל תקופה לימודית לתואר.

ראשית, תודה עמוק הלב למשפחתי ולחברי הקרובים. התמיכה הבלתי פוסקת, ההבנה, הדאגה והסבלנות שהענוקתם לי היו עבורי עוגן משמעותית במסע זה.

תודה מיוחדת לפרופסור יואל רצabi על ההנחה המסורה, ההדראה המקצועית, הרחבת הידע ולהלויין הצמוד בכל שלבי הפרויקט והלימודים. תרומתך הייתה חיונית להצלחת, ואני מלא הערכה עלך.

כמו כן, תודה לחבריו ללימודים, אשר היו עבורי שותפים בדרך – על השיתוף, העזרה ההדידית וה נכונות לחלק ידע ומשאבים, שסייעו לי להשלים את הפרויקט על הצד הטוב ביותר.

פרויקט זה אמן מסמן את סיומו של פרק חשוב בחיי סטודנט לתואר ראשון, אך תהליך הלמידה הוא מסע אינסופי. אני נושא עמי את הידע, החוויות והקשרים שצברתי, ויוצא בדרך חדשה, מלא הכרת תודה.

בתודה ובהערכה,  
אלעזר בילינקי

## תוכן עניינים

|         |  |
|---------|--|
| 2.....  | תקציר                                      |
| 2.....  | הכרת תודה                                  |
| 6.....  | 1. פרק המבוא                               |
| 6.....  | 1.1. מבוא תאורטי                           |
| 6.....  | 1.1.1. הקדמה                               |
| 6.....  | 1.1.2. מהי פונקציית גיבוב?                 |
| 6.....  | 1.1.3. שימושים של SHA-256                  |
| 8.....  | 1.2. מימוש של SHA-256                      |
| 8.....  | 1.2.1. מימוש האלגוריתם [3]                 |
| 13..... | 1.2.2. דוגמא למימוש SHA-256 על מחuzeות [4] |
| 16..... | 2. מטרת הפרויקט                            |
| 16..... | 2.1. תקציר                                 |
| 16..... | 2.2. תנאי התכנון                           |
| 17..... | 3. כרטיס PYNNQ                             |
| 17..... | 3.1. כרטיס ה-Z2                            |
| 17..... | 3.2. כלים                                  |
| 19..... | 3.3. אתחול הcartis                         |
| 19..... | 3.4. מחשב המשתמש                           |
| 20..... | 3.5. פרוטוקול AXI                          |
| 23..... | 4. תכנון חלק החומרה של המערכת              |
| 23..... | 4.1. הסבר כללי                             |
| 23..... | 4.2. תהליך הבניה של המערכת                 |
| 24..... | 4.3. הסבר אופן פעולת המערכת בשלבים         |
| 24..... | 4.4. דיאגרמת בלוקים                        |
| 24..... | 4.4.1. דיאגרמת הבלוקים מבט-על              |
| 25..... | 4.4.2. PS – Processing System              |
| 26..... | 4.4.3. DMA                                 |
| 28..... | 4.4.4. Processing System Reset             |
| 29..... | 4.4.5. Connect                             |
| 29..... | 4.4.6. Costume IP                          |
| 30..... | 4.5. תכנון הIP costume                     |

|         |   |       |
|---------|---|-------|
| 30..... | קבוע המערכת ופונקציות שימושיותPKG SHA-256.....                      | 4.5.1 |
| 31..... | מציאת מערך לוח הזמן W.....  | 4.5.2 |
| 31..... | חישוב hashround.....  | 4.5.3 |
| 32..... | בלוק הוחן main.....   | 4.5.4 |
| 37..... | אריזת IP Costume בפרוטוקול AXIS.....                                | 4.6   |
| 38..... | יצירת קבצי bitstream וההוו...<br>5. תכנון חלק התוכנה של המערכת..... | 4.7   |
| 40..... |   | 5     |
| 40..... | יבוא ספריות.....  | 5.1   |
| 41..... | חישוב תוצאה HASH ע"י החומרה.....                                    | 5.2   |
| 41..... | ריפורם המידע.....   | 5.2.1 |
| 42..... | צירבת bitstream והכנת המידע לשילוח DMA.....                         | 5.2.2 |
| 42..... | הקצת מחסניות לשילוח וקבלת המידע.....                                | 5.2.3 |
| 43..... | שליחת המידע לחישוב חומרתי וקבלת התוצאה.....                         | 5.2.4 |
| 43..... | קבלת התוצאה ושמירתה בפורמט נוח להציג.....                           | 5.2.5 |
| 44..... | חישוב תוצאה HASH ע"י התוכנה.....                                    | 5.3   |
| 44..... | תצוגת רשימת הקבצים לחישוב HASH.....                                 | 5.4   |
| 46..... | פונקציית הוחן main.....   | 5.5   |
| 46..... | תצוגת הוכתרת למשתמש.....  | 5.5.1 |
| 47..... | תצוגת האופציות למשתמש וקבלת תגובה.....                              | 5.5.2 |
| 47..... | התנהלות בקבלת מספר מהמשתמש.....                                     | 5.5.3 |
| 48..... | התנהלות בקבלת מל מהמשתמש.....                                       | 5.5.4 |
| 49..... | חישוב והדפסת התוצאה למשתמש.....                                     | 5.5.5 |
| 50..... | השוואה בין התוצאה בחומרה לתוצאה בתוכנה.....                         | 5.5.6 |
| 50..... | קריאה ל main.....   | 5.6   |
| 51..... | 6. תוצאות.....  | 6     |
| 51..... | 6.1. בדיקת המערכת וקבלת תוצאה.....                                  | 6.1   |
| 53..... | 6.2. בדיקת הכנסת קבצים בעלי סימות שונות.....                        | 6.2   |
| 54..... | 6.3. ביצוע החישוב על קבצים בגודלים שונים.....                       | 6.3   |
| 56..... | 6.4. ניתוח התוצאות מסעיף.....                                       | 6.4   |
| 56..... | 6.4.1. השערות שהועלו לגבי התוצאות.....                              | 6.4.1 |
| 56..... | 6.4.2. השערות שספיק מנוע AI לגבי התוצאות.....                       | 6.4.2 |
| 57..... | 6.4.3. בדיקת השערת השעוניים הנפרדים.....                            | 6.4.3 |

|    |       |  |
|----|-------|--|
| 58 | ..... | 6.4.4. בדיקת ההשערה לגבי ה - DMA               |
| 60 | ..... | 6.4.5. מסקנות מהניתוח                          |
| 62 | ..... | 7. סימולציות                                   |
| 62 | ..... | 7.1. בדיקת המחרוזת "abc"                       |
| 64 | ..... | 7.2. הצגת שינוי מערך לוח הזמן W בסימולציה      |
| 65 | ..... | 7.3. הצגת שינוי ה State register               |
| 66 | ..... | 7.4. חישוב word expanded                       |
| 66 | ..... | 7.5. חישוב המצב הבא בכל round                  |
| 66 | ..... | 7.6. סיכום של הסימולציות                       |
| 67 | ..... | 8. מסקנות הפרויקט                              |
| 69 | ..... | 9. קודים מלאים                                 |
| 69 | ..... | 9.1. קוד חומרה                                 |
| 69 | ..... | 9.1.1. Module ה package המכיל קבועים ופונקציות |
| 70 | ..... | 9.1.2. Module ה החישוב של rounds               |
| 71 | ..... | 9.1.3. Module ה החישוב של לוח הזמן W           |
| 72 | ..... | 9.1.4. Main ה Module                           |
| 74 | ..... | 9.1.5. קוד ה Test Bench                        |
| 79 | ..... | 9.2. קוד התוכנה                                |
| 79 | ..... | 9.2.1. קוד הפיתון של הפרויקט                   |
| 82 | ..... | 9.2.2. קוד הפיתון לצורך בניית גרפים לחקירה     |
| 84 | ..... | 9.2.3. קוד לבדיקת תדרי השעונים בכרטיס PYNNQ    |
| 85 | ..... | 9.2.4. מוצא המערכת                             |
| 86 | ..... | 10.ביבליוגרפיה                                 |
| 87 | ..... | 11.רשימות                                      |
| 87 | ..... | 11.1.רשימת אירורים                             |
| 88 | ..... | 11.2.רשימת קטעי קוד                            |
| 88 | ..... | 11.3.רשימת נוסחאות                             |



## 1. פרק המבוא

### 1.1. מבוא תאורטי

#### 1.1.1. הקדמה

פונקציית SHA-256 (קיצור של Secure Hash Algorithm 256-bit) היא אחת מהפונקציות הנפוצות והמאובטחות ביותר בתחום הкриптוגרפיה. הפונקציה היא חלק משפחת אלגוריתמים קרייפטוגרפיים הנקראת "פונקציות גיבוב" (Hash Functions). מטרת פונקציית הגיבוב היא לקחת קלט (מסמך) בגודל משתנה וליצור פלט בגודל קבוע, שבמקרה של SHA-256 הוא בגודל של 256 ביט (קיים סוגים שונים של פונקציות HASH, כאשר בכל סוג - אורך המוצא יהיה בהתאם).

ההערך המתkeletal במווצה כמעט בלתי ניתן לשחזר את הכניסה (כiom לא ידועה דרך לשחזר הכניסה). ברור גם שכיוון שישנם אינסופי כניסה אפשריות, ויש מספר סופי של מוצאים, חיבטים להיות "התנגשויות" (מוצא שותאם לשתי כניסה ויתר). אלא שעד היום עוד לא מצאו אף התנגשות כזו, ולא צפואה להימצא כזו גם בזמן הקרוב (ביכולות המחשב שיש כיום- ייקח שנים רבות ומספר עצום של מחשבים על מנת למצוא התנגשות כזו).

#### 1.1.2. מהי פונקציית גיבוב?

פונקציית גיבוב היא פונקציה המתאימה לכל קלט ערך גיבוב ייחודי בגודל קבוע. התוצאה של הפונקציה נקראת "HASH". פונקציות גיבוב משמשות למספר רב של מטרות במערכות מחשב, כמו אימותות שלמות נתונים, יצירת חתימות דיגיטליות, סיסמאות, הגנה על מידע אישי ועוד. אחת התכונות החשובות ביותר של פונקציית גיבוב היא שהיא חד-כיוונית – כלומר, לא ניתן לשחזר את הקלט מתוך הפלט.

#### 1.1.3. שימושים של SHA-256

ה SHA-256 (Secure Hash Algorithm) היא אחת מהגרסאות של אלגוריתם שנוצר על ידי סוכנות הביטחון הלאומית של ארה"ב (NSA). האלגוריתם מבוסס על עקרונות של חישוב בלוקים ומציע סדרת פעולות של חישובים על קלט הנתונים, אשר תוצאותם היא ערך גיבוב בגודל 256 ביט. כל שינוי קטן בקלט גורם לשינוי מוחלט בפלט. HASH שנוצר על ידי SHA-256 הוא תמיד באורך של 256 ביטים (ללא תלות באורך הכניסה).

**1. אימות נתונים (Data Integrity):** אחד השימושים העיקריים של SHA-256 הוא לוודא שלמות של נתונים. לדוגמה, כאשר אנו מורידים קובץ מסוים מאתר אינטרנט, לעיתים מצורף ערך SHA-256 של הקובץ. באמצעות חישוב HASH של הקובץ שהורד, ניתן להשוות את ה HASH שהתקבל, עם ה HASH הרשמי לוודא שאין שינוי בקובץ. דוגמא לכך ניתן לראות באירוע 1 ובאירוע 2

**2. חתימות דיגיטליות:** כאשר נהנת מסמך דיגיטלי, לעיתים קרובות חישוב האש של המסמך נעשה קודם ומוחתם באמצעות מפתח פרטי של משתמש. החתימה הזאת מוגנת לאדם שאוינו במסמך שיר לו, וגם מספקת לוודא שלא נעשו שינויים במסמך מאז החתימה.

3. **מערכות בלוקצ'ין ומטבעות דיגיטליים:** אחת מהשימושים המפורטים ביותר של SHA-256 הוא בבלוקצ'ינים, במיוחד Bitcoin. בכל בלוק בבלוקצ'ין ישנו חישוב SHA-256 כדי לקבע את הקישוריות של הבלוקים ולזוזה שהנתונים בבלוק לא שונו. זהו אחד מהמרכיבים הקרייפטוגרפיים שמאפשרים את האבטחה של המערכת.
4. **סיסמאות:** גם בתחום האבטחה המקומיית SHA-256 משמש כבסיס לאחסון סיסמות בצוות מאובטחת. במקומם לאחסן את הסיסמה ישירות במסד הנתונים, מחשבים את האשם שלה ומשתמשים בה להשוואה כאשר משתמש מבצע התחרבות.

#### Networking

**Realtek Ethernet Controller Driver**

Version V1168.015.0717.2023 3.52 MB 2023/12/12

Importance : **RECOMMENDED**

Description :  
This package includes Realtek's LAN driver for better ethernet performance by acting as the communication bridge between the LAN dongle and the operating system.

Minimum OS Version :  
Windows 11 64-bit 22H2

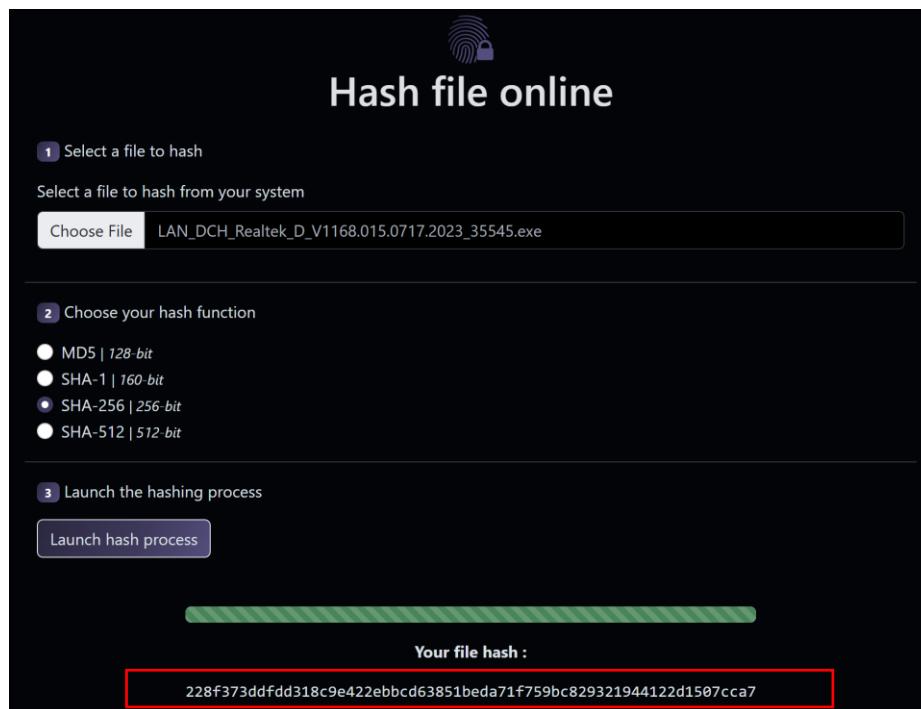
Device List :  
RTL8111H

SHA-256 :  
228f373ddfd318c9e422ebcd63851beda71f759bc829321944122d1507cca7

FAQ :  
[- \[Notebook\] Laptop Frequently Asked Question \(FAQ\)](#)  
[- \[Notebook/Desktop/AIO\] Troubleshooting - How to fix the cable network problems](#)

[SHOW MORE DESCRIPTION ^](#)

אייר 1 – קובץ להודעה המובא יחד עם הערך המתkeletal לפונקציית <sup>1</sup>SHA-256



The screenshot shows the Hash file online interface. Step 1: "Select a file to hash" with a "Choose File" button and the path "LAN\_DCH\_Realtek\_D\_V1168.015.0717.2023\_35545.exe". Step 2: "Choose your hash function" with "SHA-256 | 256-bit" selected. Step 3: "Launch the hashing process" with a "Launch hash process" button. A progress bar is shown below the button. At the bottom, the "Your file hash :" field contains the SHA-256 hash value: "228f373ddfd318c9e422ebcd63851beda71f759bc829321944122d1507cca7", which is also highlighted with a red box.

אייר 2 – בדיקת אותנטיות הקובץ ע"י הכנסתו לשרת המחשבת את ערך SHA-256 המתkeletal ממנו <sup>2</sup>

<sup>1</sup> [https://www.asus.com/laptops/for-home/vivobook/asus-vivobook-pro-15-oled-n6506/helpdesk\\_download?model2Name=N6506MV](https://www.asus.com/laptops/for-home/vivobook/asus-vivobook-pro-15-oled-n6506/helpdesk_download?model2Name=N6506MV)

<sup>2</sup> <https://hash-file.online/>

השוואה בין הערך המוצא באיוור 1 לבין התוצאה המובאת באיוור 2 מראה כי אכן התוצאות זהות וכי הקובץ הינו אותנטי ולא שינו אותו או נגעו אליו.

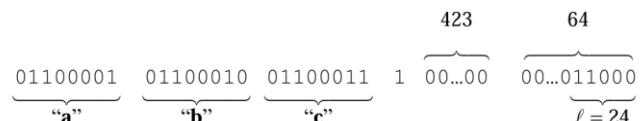
## 1.2. IMPLEMENTATION OF SHA-256

### 1.2.1. IMPLEMENTATION OF THE ALGORITHM [3]

המסמך הרשמי המתאר את פונקציית המימוש של SHA-256 הוא [3] FIPS PUB 180-4 SHA-256. שם ניתן למצוא את כל הנוסחאות והקבועים שיובאו במהלך הסבר על מימוש האלגוריתם. פונקציית-SHA-256 מטבחת בדרך כלל על ידי אלגוריתם המורכב מספר שלבים. התהילה מתחילה בקבלת הקלט (המסמך) ומחלק אותו לבלוקים בגודל של 512 ביט (64 בתים). כל בלוק עובר סדרה של חישובים והפעולות פונקציות קריפטוגרפיות שונות כמו פעולות לוגיות, חיבור, שיפוטים ולולאות. האלגוריתם משתמש גם בטבלאות קבועות שנמצאות בשימוש במהלך החישוב.

שימוש SHA-256 נעשה בצורה צו שכל שינוי קטן בקלט מביא לשינוי מוחלט בהאש. בתהילה, האלגוריתם לא מצריך זיכרון גדול, אך דורש חישובות גבוהה. ניתן לחלק את התהילה למספר חלקים שונים.

1. **חלוקת המידע לבלוקים של 512 bits וריפוד ב'1' ואז באפסים** – בחלק זה המידע נכנס למערכת ומובלק לבלוקים של 512 ביט כל בלוק (בשיטת Big-ending). כאשר נגמר המידע, מוסיפים לו עוד ביט -בקצה- בעל ערך '1', ולבסוף מרפסים באפסים עד להשלמת בלוק של 448. בקצת בלוק זה, מוסיפים עוד 64 ביט ("밀ימ") המבטאים את מספר הביטים במידע המוכנס (לכן ניתן לומר שיש הגבלה על אורך ההודעה המוכנסת, אלא שהגבלה זו היא  $2^{64}$  שזה יוצא:  $18,446,744,073,709,551,615$  (שהה בערך 18 טריליאן בתים של מידע, או בערך 2 טריליאן בתים של מידע)). דוגמא לריפוד ההודעה "abc":



2. **מציאת מערך לוח הזמנים של ההודעות W** – בשלב זה נוצרים 48 מילימ' נוספיםפות, אחרי ה-16 מילימ' של הבלוק (כל מילה היא 32 ביט, ואז  $32^2 = 16$  יוצא הבלוק של 512 ביט כפי שהסביר).

הчисוב ליצירת המילימ' הבאות תלוי כל מילה ב-16 שלפניהם.

נוסחה 1 –  $\sigma_1$

$$\sigma_1(x) = (x \ggg 17) \oplus (x \ggg 19) \oplus (x \gg 10)$$

נוסחה 2 –  $\sigma_0$

$$\sigma_0(x) = (x \ggg 7) \oplus (x \ggg 18) \oplus (x \gg 3)$$

נוסחה 3 –  $W_i^{(j)}$

$$W_i^{(j)} = \sigma_1(W_{i-1}^{(j)}) + W_{i-7}^{(j)} + \sigma_0(W_{i-15}^{(j)}) + W_{i-16}^{(j)}$$

כאשר הסימון »» הוא הZZה סיבובית ימינה (*Right rotate*), והסימון »» הוא הZZה ימינה (כאשר הביטים המוזזים בצד ימין נעלמים ובצד שמאל נכנסים אפסים כמספר ההZZה). הסימון ? בא לסמן את אינדקס המילה (מתוך 64 המילים. מכיוון ש16 המילים הראשונות מגיעות מהמידע הנכנס למערכת, אינדקס זה מתחילה מהמספר 17), והסימון j מסמן את אינדקס הבלוק המדובר (כיוון שהודעה בגודל - או גדולה מ- 448 ביט, מצריכה יותר מבלוק אחד לצורך חישוב ה-*SHA-256* שלו).

3. עדכון המשתנים הפעילים – בפונקציית ה-*SHA* ישנים 8 מילימ (וקטורים של 32 ביט), אשר בכל שלב (*CLK* של השעון) משתנים על פי החישוב הבא:

נוסחה 4 –  $\Sigma_0$

$$\Sigma_0(x) = (x \ggg 2) \oplus (x \ggg 13) \oplus (x \ggg 10)$$

נוסחה 5 –  $\Sigma_1$

$$\Sigma_1(x) = (x \ggg 6) \oplus (x \ggg 11) \oplus (x \ggg 25)$$

נוסחה 6 – Choice

$$Choice(x, y, z) = (x \text{ and } y) \oplus (\bar{x} \text{ and } z)$$

נוסחה 7 – Majority

$$Majority(x, y, z) = (x \text{ and } y) \oplus (x \text{ and } z) \oplus (y \text{ and } z)$$

נוסחה 8 – temp1

$$temp1 = h + \Sigma_1(e) + Choice(e, f, g) + K_i + W_i^{(j)}$$

נוסחה 9 – temp2

$$temp2 = \Sigma_0(a) + Majority(a, b, c)$$

נוסחה 10 – New active variables

$$\begin{aligned}
 a &= temp1 + temp2 \\
 b &= a \\
 c &= b \\
 d &= c \\
 e &= d + temp1 \\
 f &= e \\
 g &= f \\
 h &= g
 \end{aligned}$$

בתחילת התהיל'ר הערכים של שמוות המשתנים ( $a, b, c, d, e, f, g, h$ ) הם ערכים קבועים הידועים מראש, המיצגים את חלק השבר של השורש של שמוות המספרים הראשונים הראשוניים. ניתן להציג אליהם ע"י לקיחת שורש (ריבועי) מ8 המספרים הראשונים הראשוניים, כפול ב $2^{32}$ , להמיר לבסיס הקסדצימלי (עיגול כלפי מטה), ולקחת 8 הספרות הראשונות מהתוצאה (להשmidt את  $MSD$ ). הערכים הינם:

$$\begin{aligned} H_0^{(1)} &= 0x6a09e667 \\ H_0^{(2)} &= 0xbb67ae85 \\ H_0^{(3)} &= 0x3c6ef372 \\ H_0^{(4)} &= 0xa54ff53a \\ H_0^{(5)} &= 0x510e527f \\ H_0^{(6)} &= 0x9b05688c \\ H_0^{(7)} &= 0x1f83d9ab \\ H_0^{(8)} &= 0x5be0cd19 \end{aligned}$$

בהתחלת הפעולה,  $a = H_0^{(1)}$ ,  $b = H_0^{(2)}$  etc. בכל סבב ערכים אלו משתנים כפי שהוסבר (ערכי  $H_0^{(1-8)}$  נשמרים בצד ובשלב הבא יסביר מה עושים איתם). בהגדירה של  $temp1$  המובאת לעליה, ניתן לראות כי מעבר לפונקציות הידועות יש את  $W_i^{(j)}$  ואת  $K_i$ . ערכים אלו הם  $W$  מלאו הזמן שהוסבר בשלב 2 (הו והז' באו לסמן את האינדקס של  $W$  מתוך 64 שהוסברו בשלב 2, ואת האינדקס של  $chunk$  הנוכחי, בהתאם (64 סיבובים הראשונים זהו  $chunk$  (המודול 648 ביט או יותר) מוסיפים עוד הראשון, ואם יש צורך ביותר (ההודעה הינה באורך של 448 ביט או יותר) מוסיפים  $chunks$  (ואיתם 64 סיבובים) לפי הצורך). והוא  $K$  זה 64 קבועים התואימים ל-64 הסיבובים של כל  $chunk$  (ערכים אלו הינם קבועים ואין משתנים אף בשלב). ניתן להציג לערכים אלו ע"י לקיחת שורש שלישי ל-64 המספרים הראשונים הראשוניים, הכפלה ב $2^{32}$ مرة לבסיס הקסדצימלי, ולקיחת 32 ביט התוצאות ( $LSB$ ). ביצוג הקסדצימלי הערכים הם:

```
0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5,
0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,
0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3,
0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,
0xe49b69c1, 0xefbe4786, 0xfc19dc6, 0x240ca1cc,
0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7,
0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,
0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13,
0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,
0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3,
0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5,
0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,
0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208,
0x90beffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2
```

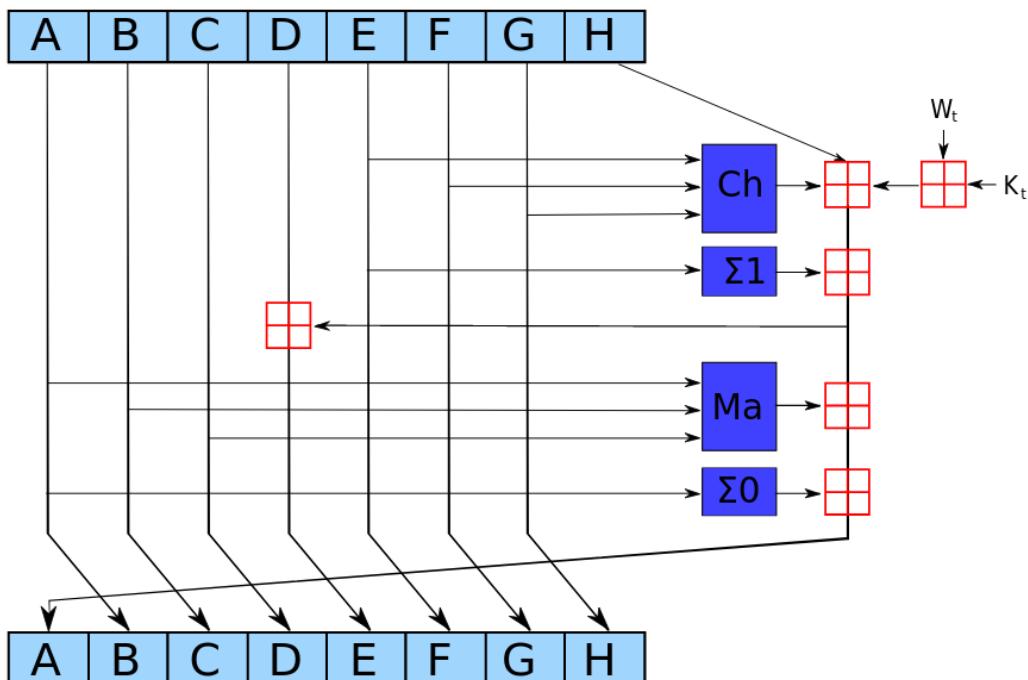
כאשר משמאל לימין מלמעלה למטה, הראשון הוא K1 והאחרון הוא K64.

4. **שלב סיום 64 הסבבים** – בשלב זה, אם ההודעה הייתה בעלת פחות מ448 ביט, השלב האחרון הוא חיבור אריטמטי בין ערכי המשתנים הפעילים ( $a, b, c, d, e, f, g, h$ ) לבין המשתנים ההתחלתיים ( $H_0^{(1-8)}$ ). אחרי החיבור בניהם, המוצא של המערכת (תוצאת HASH) הוא בדיקת שרשרת (concatenation) בין שמות התוצאות. בדיקה מהירה מוכיחה כי אכן המוצא הינו בעל 256 ביט ( $256 = 8^3 \cdot 32$ ).

אם ההודעה הייתה בעלת 448 ביט או יותר (אם יש בדיקת 448 ביט, צריך עוד בלוק מפני שציריך גם להציג 1 לסופם ההודעה וגם 64 ביט שייצגו את אורק ההודעה), השלב הזה הוא דומה לשלב שהוסבר לאם יש פחות מ448 ביט, אלא שבמקרה שתוצאת החיבור תצא לモץ המערכת, החיבור יהיה הערך החדש של המשתנים הפעילים, וגם ערך זה יהיה הערך החדש של המשתנים ההתחלתיים – לצורך הסבב (chunk) הנוכחי. כך בסוף הסבב זהה, הפעולה האחורונה תהיה סכימה עם הערכים החדשניים שהתקבלו (וקח יתקבל הערך החדש  $H_1^{(1-8)}$  אשר יחליף את הערך הקודם  $H_0^{(1-8)}$ ).

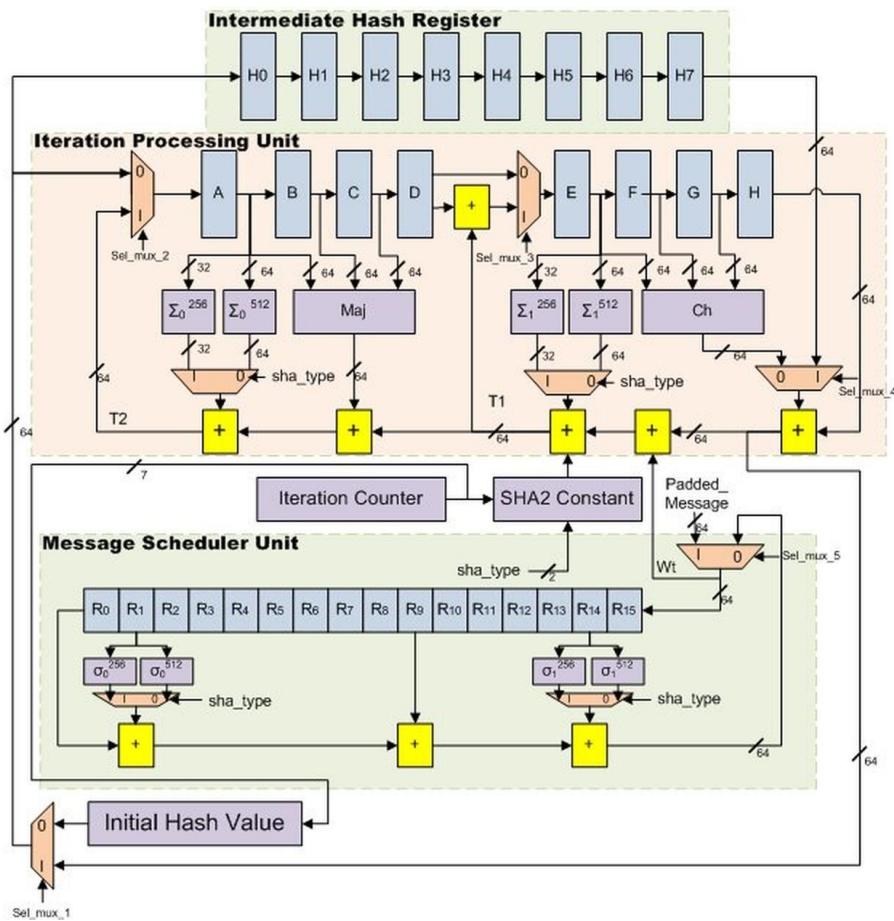
5. במידה וצריך, התהיליך חוזר על עצמו. המידע המוכנס למערכת מסודר ב-512 הביטים הראשונים (16 words) הראשוניים. כאשר אם המידע לא מלא את 447 הביטים- מרפדים ב-1 ואז באפסים כפי שהסבירו), ולאחר מכן נבנה מערך לוח הזמן, המשתנים הפעילים מתעדכנים וועברים 64 סבבים ועוד עד שכל ההודעה עוברת.

באיור 3 ניתן לראות את התיאור הגרפי של נוסחה 8 ונוסחה 10.



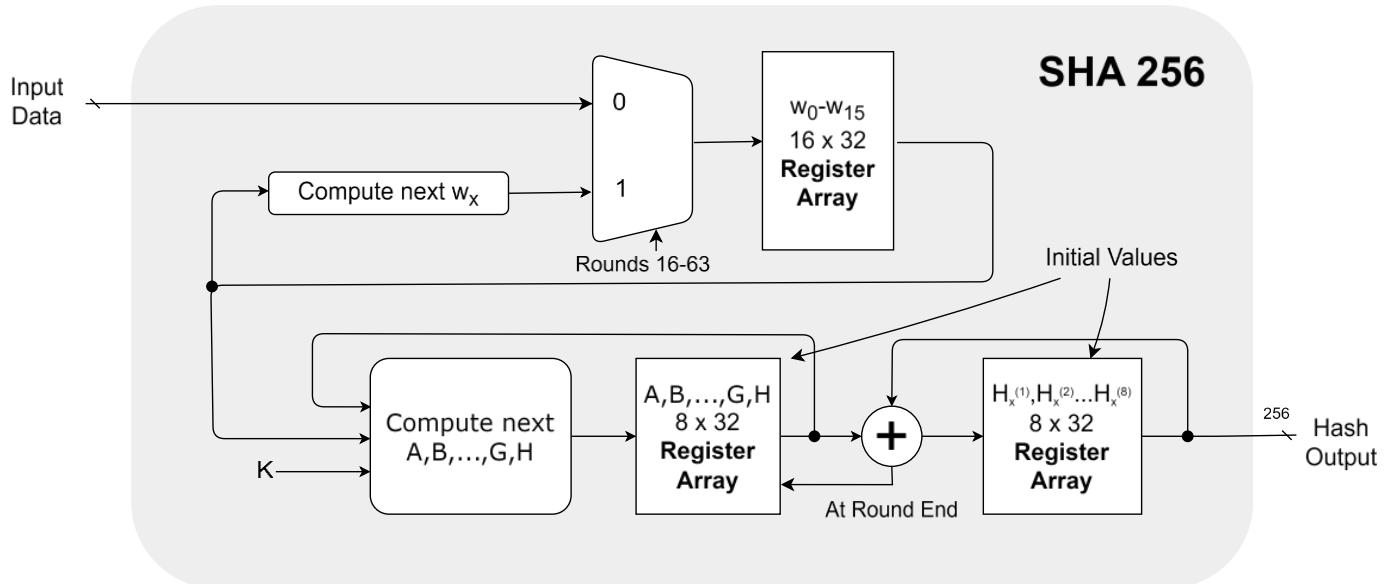
איור 3 – תיאור גרפי של נוסחה 8 ונוסחה 10

באיור 4 מתוארת המערכת השלמה (בשינוי קל המאפשר ביצוע של SHA-512 בנוסף לשא-256 בו אנו מתמקדים).



איור 4 – דיאגרמה המתארת את פעולה SHA המלאה

באיור 5 ניתן לראות את ההליך של ה-SHA-256 בסקירה רחבה יותר (בלי פירוט של התהליכיים והчисובים המתבצעים בפנים).



איור 5 – דיאגרמת בלוקים של תיאור פעולה SHA-256

ניתן לראות בדיאגרמה כי המידע מוכנס (דרך פרוטוקול AX), מה שלא מוזכר בה) לתוך מוקס אשר יקבע מה נכנס לתוך 16 הרגיסטרים של הלוח הזמן (W). ב-16 מילימ' הראשונות, המידע שייכנס הוא המידע המוכנס BUS הכניסה, ובclock הבא המידע שייכנס (לרגיסטר הראשון. וידורס את המידע הנוכחי שם), זה W16 שיתקבל מנוסחה 3. מעורם רגיסטרים זה (W) מתווסף לחישוב המשתנים הפעילים הבאים (כפי שנitinן לראות בנוסחה 8 (זה ה W שם בנוסחה), ובכל clock של השעון הרגיסטרים של המשתנים הפעילים מתעדכנים לערך החדש שהושב. בסיום הבלוק המשנים הפעילים נסכמים עם המשתנים ההתחלתיים ומתקבלים את הערך של התוצאה. אם זהו הבלוק האחרון, המידע יוצא למוצא, ואם לא, אז מתחילה בלוק חדש כאשר הערכים ההתחלתיים עודכנו וכן המשתנים הפעילים (במקום הערכים ההתחלתיים המוגדרים על ידי האלגוריתם (של FIPS [3]) הערכים הם מה שיצא בסיום הבלוק הראשון, וכך ממשיכים לבלוק הבא וחוזר חלילה.

### 1.2.2 דוגמא לIMPLEMENTATION SHA-256 על מחרוזת [4]

באיור 6 ובאיור 7 ניתן לראות IMPLEMENTATION SHA-256 של תהליך SHA-256. ההודעה המוכנסת היא באקס: *abcdabcdecdefdefgefghfhgijhijkjlmklmnlmnynomnopnopq* 448 ביט, ולכן כפי שהסביר, התהליך יצרך שני *chunks*. בראשון ההודעה תיכנס בדיקת בית הראשוניים, ובביט ה-449 (באיור 6, בtemp1 בשורה 14 W), ההודעה מרווחת "1" ולאחריו אפסים עד סוף הבלוק (עד סוף W15). אחרי יש עוד 48 שורות - W63 - W16 בהם נכנס המידע המתקיים מנוסחה 3 ( $W_i^{(j)}$ ). ניתן לראות את המשתנים הפעילים (-*variables*). האתר ממנו נלקחה התמונה מציע פירוט פירוט שלב-שלב על שלבי הניתוח והתהליך. צילום מסך זה נלקח כאשר השלב הוא חישוב המשתנים הפעילים בסיבוב 63. לעומת זאת ב-62, ואת התהליך למציאת *a* ניתן לראות בחלק העליון את ה-*working-variables* של הסיבוב Temp1 ו-*Temp2* ולאחר מכן חישוב של הסכום שלהם ( למציאת *e*) ושל הסיבוב 63 (מציאת *d*). בתחתית הדף ניתן לראות את השלב האחרון של הבלוק הראשון. כפי שהסביר, השלב האחרון הוא חיבור המשתנים הפעילים עם המשתנים ההתחלתיים, וכן עדכון המשתנים ההתחלתיים לצורך הסיבוב הבא.

בצד שמאל של האיור ניתן לראות בזורה מה החישובים הנדרשים לצורך סיבוב זה. מצד ימין מובאים 64 הערכים הקבועים של *K*. כאמור, ערכים אלו הינם קבועים לחלווטן ללא תלות בכלום ונשארים כמו שהם בכל חישוב ובכל החישוב.

באיור 7 התהליך ממשיך ונitinן לראות כי הבלוק מלא בריפוד של אפסים (כיוון שה-1 כבר הוסיף בבלוק הקודם). 64 ביט האחרונים הם 00000000...0000. המספר 111000000 מבטא (בבינארי כMOV) את המספר 448. לאחר מכן יש את ה-*message schedule* כMOV. בתחתית הדף ניתן לראות את השלב האחרון, שהוא חיבור המשתנים הפעילים המתקיים בסוף הסיבוב 64 של *Chunk* יחד עם המשתנים ההתחלתיים (שעודכנו למשתנים הפעילים שהתקבלו בסוף הסיבוב *chunk* הקודם). סכום זה מומר לאקס *hex* ומוחובר (*concatenation*) וכן מתקובלת התוצאה הסופית.

איור 6 - דוגמא למציאת ערך SHA-256 של מחרוזת

Text abcd... Working schedule - 2nd chunk Working Variables K constants

1. Update working variables as:

```
w0 = 00000000000000000000000000000000  
w1 = 00000000000000000000000000000000  
w2 = 00000000000000000000000000000000  
w3 = 00000000000000000000000000000000  
w4 = 00000000000000000000000000000000  
w5 = 00000000000000000000000000000000  
w6 = 00000000000000000000000000000000  
w7 = 00000000000000000000000000000000  
w8 = 00000000000000000000000000000000  
w9 = 00000000000000000000000000000000  
w10 = 00000000000000000000000000000000  
w11 = 00000000000000000000000000000000  
w12 = 00000000000000000000000000000000  
w13 = 00000000000000000000000000000000  
w14 = 00000000000000000000000000000000  
w15 = 00000000000000000000000000000000  
w16 = 00000000000000000000000000000000  
w17 = 00000000000000000000000000000000  
w18 = 00000000000000000000000000000000  
w19 = 00000000000000000000000000000000  
w20 = 00000000000000000000000000000000  
w21 = 00000000000000000000000000000000  
w22 = 00000000000000000000000000000000  
w23 = 00000000000000000000000000000000  
w24 = 00000000000000000000000000000000  
w25 = 00000000000000000000000000000000  
w26 = 00000000000000000000000000000000  
w27 = 00000000000000000000000000000000  
w28 = 00000000000000000000000000000000  
w29 = 00000000000000000000000000000000  
w30 = 00000000000000000000000000000000  
w31 = 00000000000000000000000000000000  
w32 = 00000000000000000000000000000000  
w33 = 00000000000000000000000000000000  
w34 = 00000000000000000000000000000000  
w35 = 00000000000000000000000000000000  
w36 = 00000000000000000000000000000000  
w37 = 00000000000000000000000000000000  
w38 = 00000000000000000000000000000000  
w39 = 00000000000000000000000000000000  
w40 = 00000000000000000000000000000000  
w41 = 00000000000000000000000000000000  
w42 = 00000000000000000000000000000000  
w43 = 00000000000000000000000000000000  
w44 = 00000000000000000000000000000000  
w45 = 00000000000000000000000000000000  
w46 = 00000000000000000000000000000000  
w47 = 00000000000000000000000000000000  
w48 = 00000000000000000000000000000000  
w49 = 00000000000000000000000000000000  
w50 = 00000000000000000000000000000000  
w51 = 00000000000000000000000000000000  
w52 = 00000000000000000000000000000000  
w53 = 00000000000000000000000000000000  
w54 = 00000000000000000000000000000000  
w55 = 00000000000000000000000000000000  
w56 = 00000000000000000000000000000000  
w57 = 00000000000000000000000000000000  
w58 = 00000000000000000000000000000000  
w59 = 00000000000000000000000000000000  
w60 = 00000000000000000000000000000000  
w61 = 00000000000000000000000000000000  
w62 = 00000000000000000000000000000000  
w63 = 10110010000000000000000000000000
```

Where:

```
Temp1 = h + II + Choice + k63 + w63  
Temp2 = II + Majority  
II = (e rightrotate 6) xor  
(e rightrotate 11) xor  
(e rightrotate 25)  
Choice = (e and f) xor ((not e) and g)  
Majority = (a and b) xor (a and c) xor  
(b and c)
```

2. Add the working variables to the current hash value:

```
h0 = h0 + a  
h1 = h1 + b  
h2 = h2 + c  
h3 = h3 + d  
h4 = h4 + e  
h5 = h5 + f  
h6 = h6 + g  
h7 = h7 + h
```

3. Append hash values to get final digest:

```
Sha256 = h0 h1 h2 h3 h4 h5 h6 h7
```

Working schedule - 2nd chunk

Working Variables

K constants

Sha256

איור 7 – דוגמא למציאת ערך ה- $SHA-256$  של מחוזת-2

## Sha256

248d6a61d20638b8e5c026930c3e6039a33ce45964ff2167f6ecedd419db06c1

## 2. מטרת הפרויקט

### 2.1. תקציר

מטרת הפרויקט היא פיתוח מערכת המבצעת חישוב של אלגוריתם לוח SHA-256 באמצעות FPGA מסוג Z2 PYNQ, בשילוב עם מעבד ARM. המערכת מאפשרת העברת נתונים באמצעות משיק AXI4 Lite DMA בין המעבד לבין הלוגיקה של ה-FPGA. החישוב של SHA-256 מבוצע בחומרה. המערכת פועלת באמצעות שילוב של עיבוד כומרה בשפת SystemVerilog ועיבוד תוכנה בשפת Python, ומאפשרת את ביצוע החישוב בצורה מהירה ויעילה על גבי פלטפורמת PYNQ.

הקוד Python יקבל תמונה, יחלץ ממנה את הביטים מהם היא מורכבת, ויסלח אותם לעיבוד כומרה. החומרה תחזיר את הערך HASH התואם למידע אותו היא קיבלה.

המטרה הסופית היא להראות כי החישוב החומרתי מהיר יותר מהחישוב התוכני כתוצאה מהיכולת המקבילה שיש לחומרה על פני תוכנה.

### 2.2. תנאי התכנון

- בשלב הראשון, נלמד האלגוריתם באמצעות מקורות באינטרנט ובאמצעות המסמך הרשמי של FIPS PUB 180-4 [3] המתאר את האלגוריתם ומספק דוגמאות לשימוש בו.
- בשלב הבא תוכנן ועיצוב IP Custom באמצעות התוכנה Vivado O/S של AMD. הבניה והתוכנו עוסכו בצורה צזו שבסכל הזרננות בה ניתן לבצע מספר חישובים במקביל, המערכת תבצע זאת על מנת לחסוך בזמן החישוב.
- בשלב הבא, המערכת תחויב דרך Block design ל DMA אשר יdag לתקשורת בין החלק של החומרה (PL) לחלק של התוכנה הקיים בכרטיס (PS).
- לבסוף, חולץ bitstream מתוך התוכנה (יחד עם קובץ ה .hwh) אשר הועלה – דרך פלטפורמת JUPYTER לכרטיס PYNQ ושם התוכנה (Python) צורבת את הקובץ, ושולחת אותו אליו את המידע של התמונה. המידע יעובד ויישלח בחזרה לתוכנה על מנת שתוצג למשתמש.

## 3. כרטיס ה-PYNQ

### 3.1. כרטיס ה-Z2 PYNQ

כרטיס ה-PYNQ [5] הוא פרויקט קוד פתוח שפותח על ידי חברת AMD, כרטיס ה-Z2 PYNQ הוא רכיב גמ לאחר הייצור, בניגוד ל-ASIC (Application Specific Integrated Circuit) רכיב מיועד לפעולה מסויימת שאינו ניתן לשינוי לאחר ייצורו. הkartis מבוסס על ה-Zynq XC7Z020 של Xilinx ויכול להריץ לוגי שנitin לתכונות (logic programmable - PL) והן מעבד דו-לבתי Cortex-A9. מדובר במערכת על-kartis (SoC) המשלבת בין חומרה ותוכנה.

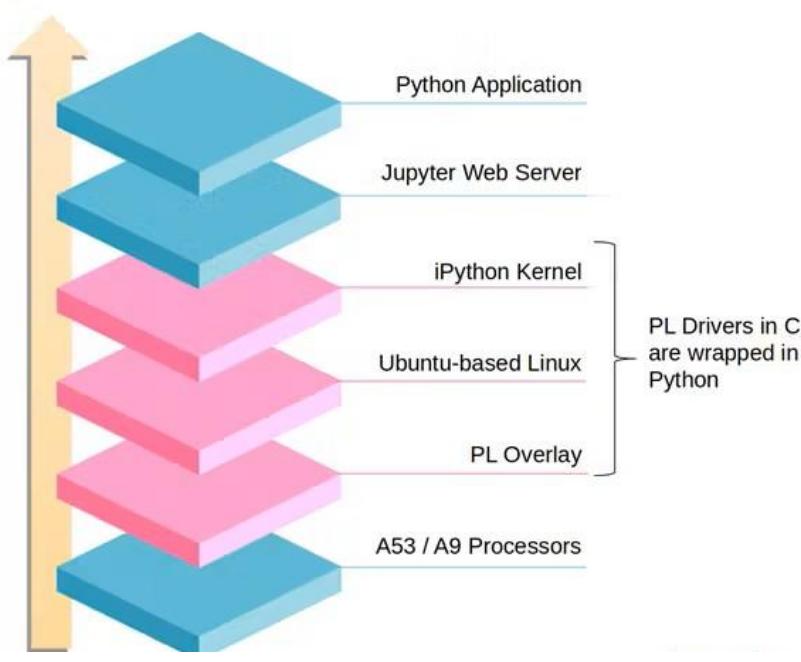
היתרון המרכזי של כרטיס ה-PYNQ הוא יכולת לתוכנן מיגל לוגי בשפת חומרה כמו LogiSim, וליישם אותו על הkartis בקלות, ללא צורך בהתקינה חדשה. לאחר מכן, ניתן להפעיל את הkartis באמצעות שפת Python, כך שהחומרה נתפסת כספרייה Python. שימוש זה מאפשר ניצול יתרונות של כתיבת RTL בשפת חומרה, תוך שימוש בשפת Python על מנת לפתח מערכות אלקטרוניות מורכבות.

### 3.2. כלים

Jupyter Notebook - סביבת מחשב אינטראקטיבית מבוססת דפדפן. מספר סביבות עבודה שבהן משתמשים יכולים לכתוב ולהפעיל קוד Python, שיכל להתmeshק ישירות עם חומרת ה-FPGA.

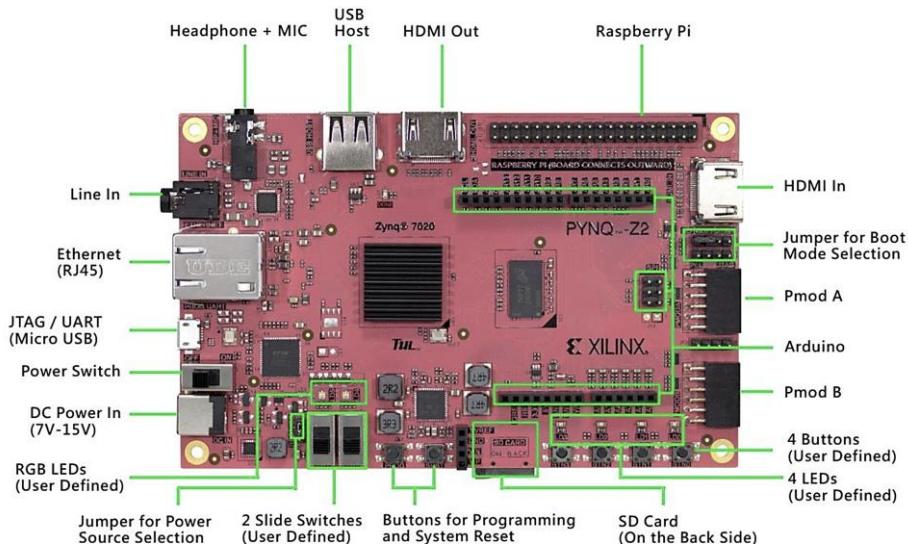
LINUX - הארכיטקטורה בcartis ה-PYNQ מורכבת מיחידת עיבוד PS המכילה ליבות ARM מסווג Cortex-A9

וחידת לוגית הנינתת לתכונות PL. לינוקס מאפשרת ללבית ה-ARM לתקשר עם החלק של ה-PL, ובطיבה אינטראקטיבית חלקה בין יישומי התוכנה לחומרה הבסיסית.



איור 8 – תיאור השכבות והחלקים של כרטיס ה-PYNQ

כאמור, הkartis שעליו מומש הפרויקט הינו kartis Z2 PYNQ שפותח על ידי חברת TUL.



איור 9 – תיאור השכבות והחלקים של כרטיס ה-Z2 PYNQ

ה-ZYNQ הוא קו של יחידות עיבוד היברידיות CPU-FPGA שפותחו על ידי חברת Xilinx, המורכבת משני חלקים עיקריים:

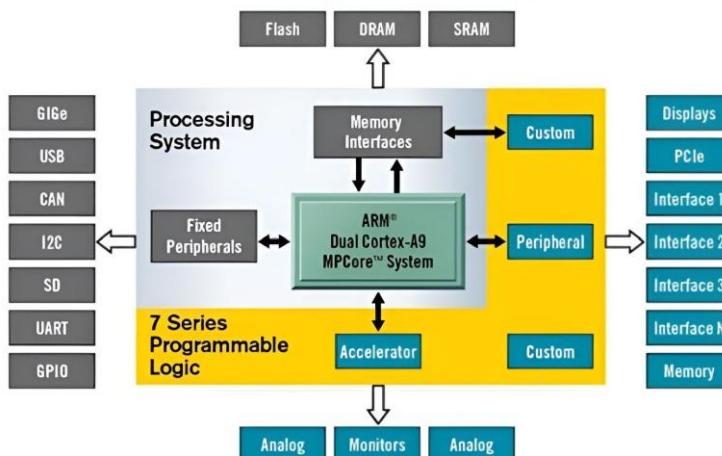
- PS-Processing System •
- PL-Programmable Logic •

ה-PL הוא החלק הנitin לשימוש ותוכנות בכל דרך שבה ניתן להגדיר FPGA רגיל. ה-PS הוא מעבד ליבת כפול מסדרת Cortex-A9.

יחידת ה-PL אחראית על ניהול כל המערכת, ייצור שעוניים בתדרים הנדרשים וניהול הזיכרון. ל-PS נדרש גישה לציכרון DDR ושליטה על הפריפרויות השונות וה-PL באמצעות פרוטוקול AXI4.

כרטיס ה-Z2 PYNQ או בשם הסידורי המדויק XC7Z020-1CLG400C כולל 13,000 פוטו-LOGIC Cells, 6 Look Up Tables (LUTs) ו-8 דלגלגים.

בנוסף 220 יחידות DSP, זיכרון 16 MB DDR3512, זיכרון Quad SPI-Flash16 MB ו-PS 16 MB.

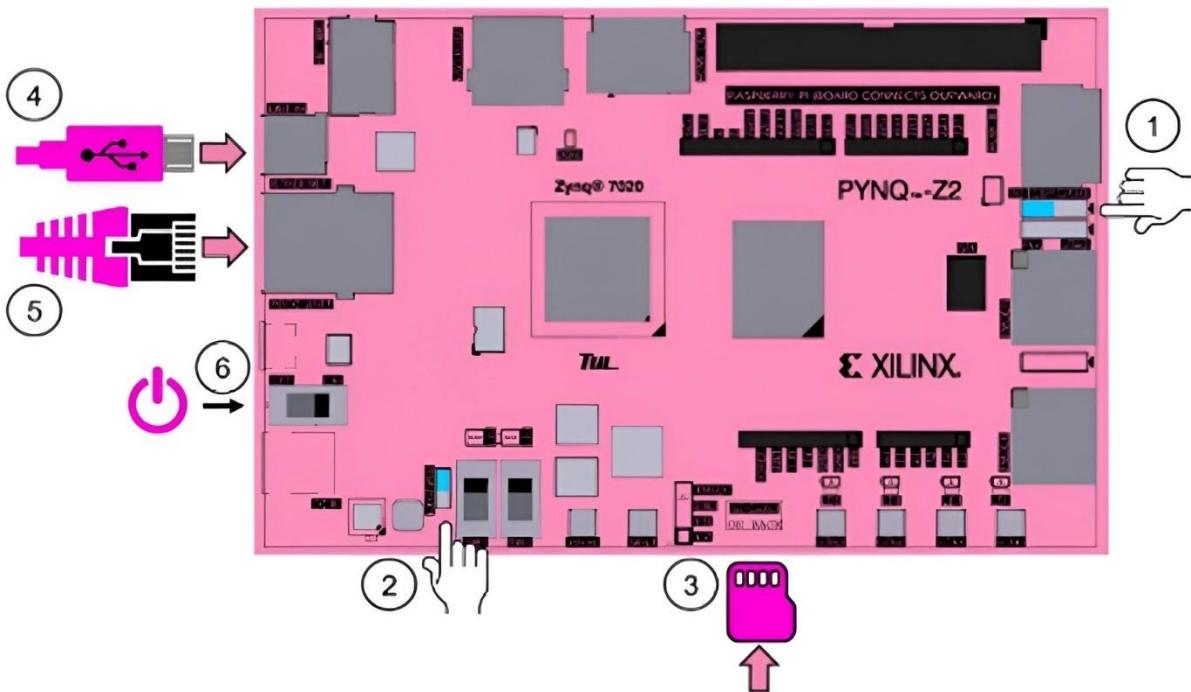


איור 10 – התממשקויות ה-PL וה-PS לפerrיפרויות השונות [6]

### 3.3. אתוחול הכרטייס

כוניסת ה-Z2-PYNQ הוגדר כר שיאותחל (BOOT) מכרטיס SD שעליו הותקנה מערכת הפעלה Linux.

שלבי העבודה:

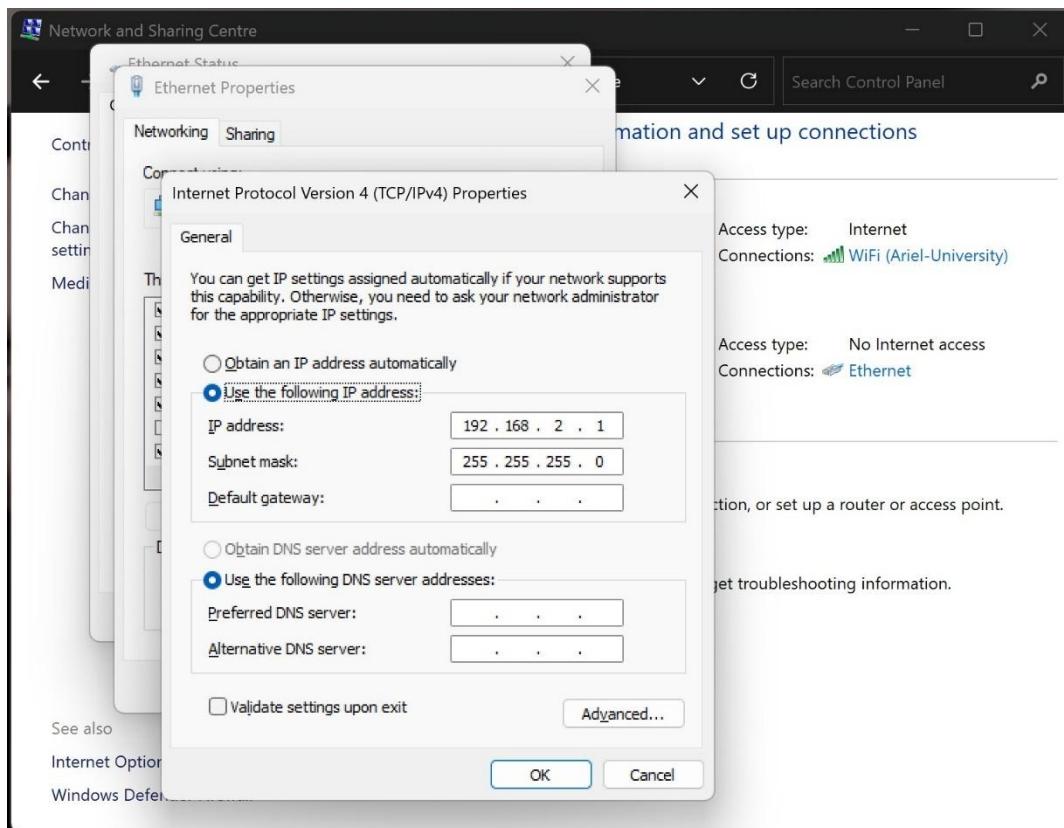


איור 11 – שלבי העבודה להפעלת כרטיס ה-Z2-PYNQ

1. הגדרת הכרטייס לאתחול מכרטיס ה-SD על ידי העברת הג'אמפר למיקום הרלוונטי המסומן בכחול.
2. העברת הג'אמפר למיקום הרלוונטי שמסומן בכחול על מנת לספק מתח לכרטיס מכוניסת ה-*micro-SD*.
3. הכנסת כרטיס SD לסЛОט ה-SD כאשר מותקנת עליו מערכת הפעלה של Linux (image file).
4. חיבור כבל *micro usb* בין הכרטייס למחשב.
5. חיבור כבל רשת (Ethernet) בין הכרטייס לנットב.
6. הפעלת הכרטייס (העברת מתג ההפעלה למצב ON) והמתנה להבהוב נורות ה-LED של גבי הכרטייס.

### 3.4. מחשב המשתמש

מכיוון שלכרטייס ה-Z2-PYNQ יש כתובת IP סטטית: 192.168.2.99, על מנת לאפשר תקשורת העברת נתונים בין מחשב המשתמש לכרטיס ה-Z2-PYNQ על המשתמש להגדיר במחשב שלו באופן ידני כתובת IP סטטית שתתאים לרשת ה-Z2-PYNQ.



איור 12 – הגדרת כתובת IP סטטית במחשב המשתמש

לשם כך יש לעבור על השלבים הבאים:

1. הגדרת כתובת IP סטטית באופן ידני.
2. חיבור כרטיס 2-PYNQ-Z לכניסת Ethernet של המחשב
3. הזנת כתובת URL הבאה: <http://192.168.2.99:9090> בדף האינטרנט.

הגדרת כתובת IP סטטית באופן ידני במערכת הפעלה Windows :

1. יש להכנס ל"הגדרות" => "Ethernet" => "שנה אפשרויות מתאימים"
2. קליק שמאלית על חיבור הרשת המתאים ולהציג עלי "מאפיינים"
3. בחילון חדש לסמן את אופציית הבחירה: "TCP\IPv4 פרוטוקול אינטרנט גרסה 4" ולאחר מכן ללחוץ על "מאפיינים"
4. בחילון חדש שנפתח יש לסמן "קבל כתובת IP הבאה". יש למלא את כתובת ה- IP (באופן רשות מקומית כמו PYNQ. לדוגמה, 192.168.2.1 וכן למלא את המסכת רשות משנה (255.255.255.0).
5. לאחר הגדרת כתובת ה-IP ניתן לתקשר עם הכרטיס דרך סביבת העבודה Jupyter Notebook דרך דף האינטרנט על ידי כתיבת ה-IP של הכרטיס בדף.

### 3.5. פרוטוקול AXI

AXI או בשמו המלא Advanced extensible Interface הוא פרוטוקול עבור מעגלים שפועלים בתדר גבוה ומשתמש לתקשורת בין בלוקים של IP שונים במערכות מושלבים.

### :AXI-Lite

גרסה פשוטה של פרוטוקול AXI. מיועד לחילופי נתונים בתפקה נמוכה. אין תומך בהעברות burst או בכתובות מרובות. משמש בדרך כלל עבור נתונים נתיביים פשוטים.

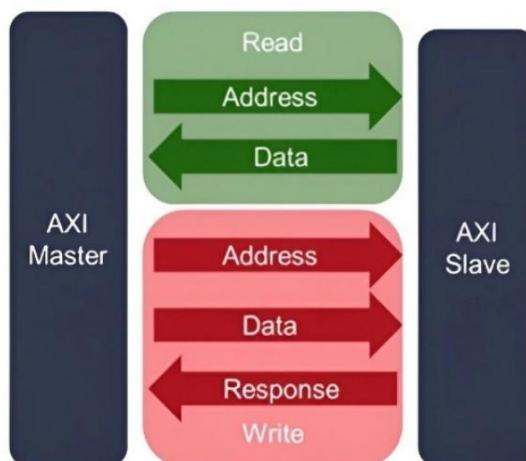
### :AXI-Stream

תת קבוצה של פרוטוקול AXI המותאם להזרמת נתונים. אין שלב של כתובות ובקרה, אלא הוא מתמקד רק בשלב הנתונים.

אידיאלי עבור יישומים כמו הזרמת וידאו או תנועת מנות נתונים שבהם זרימת נתונים רציפה היא קריטית.

### פרוטוקול AXI מגדר בתוכו חמישה ערכאים:

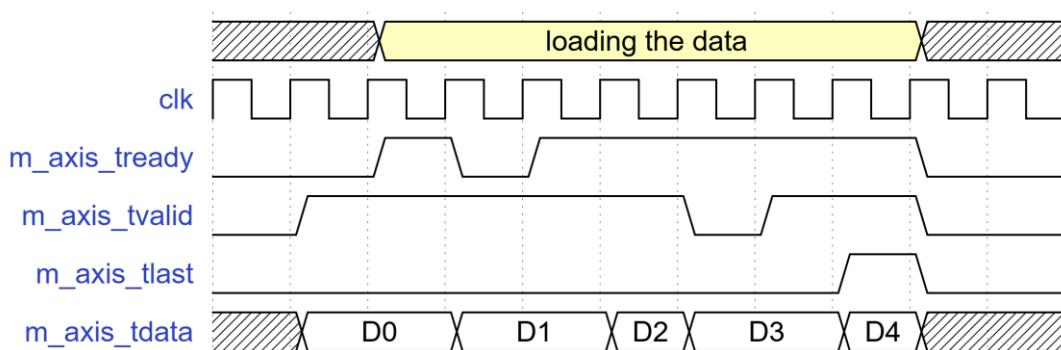
שניים מהערכאים משמשים לקריאה: קריאת כתובות, קריאת נתונים.  
שלושה מהערכאים משמשים לכתיבה: כתיבת כתובות, כתיבת נתונים, כתיבת תגובה.



איור 13 – ערכי קריאה וכטיבה של פרוטוקול AXI

העברה מידע בין SLAVE לMASTER רק מתרחשת רק כאשר גם האות VALID וגם האות Y נמצאים בגביה בזמן עליית שעון.

### AXI handshake



איור 14 – דיאגרמת זמנים של פרוטוקול AXI

#### תיאור כניסה ויציאה ה-MASTER:

- *ACLK* - כניסה שעון
- *N\_RESETN*-אות כניסה איפוס שפועל במנוע
- *M\_AXIS\_TDATA* - מוצא קווי המידע של ה-MASTER.
- *M\_AXIS\_TVALID* - מוצא שמודיע ל-SLAVE שהמידע שנמצא על קווי המידע הוא מידע תקין שיש להתייחס אליו.
- *M\_AXIS\_TLAST* - מוצא שמודיעה ל-SLAVE שהמידע שנמצא כרגע על קווי המידע הוא הנตอน האחרון בחבורה הנตอนים.
- *Y\_AXIS\_TREADY* - כניסה שמגיעה מה-SLAVE ומודיעה ל-MASTER שה-SLAVE מוכן לקבל מידע.

#### תיאור כניסה ויציאה ה-SLAVE:

- *ACLK* - כניסה שעון
- *N\_RESETN*-אות כניסה איפוס שפועל במנוע
- *S\_AXIS\_TDATA* - כניסה של קווי המידע של ה-MASTER שנכנסו לתוך ה-SLAVE
- *S\_AXIS\_TVALID* - כניסה שמגיעה מה-MASTER שאומרת שהמידע שנמצא על קווי המידע הוא מידע תקין שיש להתייחס אליו.
- *S\_AXIS\_TLAST* - כניסה שמודיעה ל-SLAVE שהמידע שנמצא כרגע על קווי המידע הוא הנตอน האחרון בחברת הנตอนים.
- *Y\_AXIS\_TREADY* - מוצא שמגיעה מה-SLAVE ומודיעה ל-MASTER שה-SLAVE מוכן לקבל מידע.

כאשר המידע שבקווי המידע *M\_AXIS\_TDATA* על ה-MASTER לעלות מיד לגובה את אות הבקרה *M\_AXIS\_TVALID* ולא לחכות עד שרגל ה-*S\_AXIS\_TREADY* תעלה גם לגובה.

בנוסף, ברגע ש-*M\_AXIS\_TVALID* מעלת לגובה, הוא לאראשי לרדת אלא עד ש-*S\_AXIS\_TREADY* גם יעלת לגובה ותבצע העברת נתוניים.

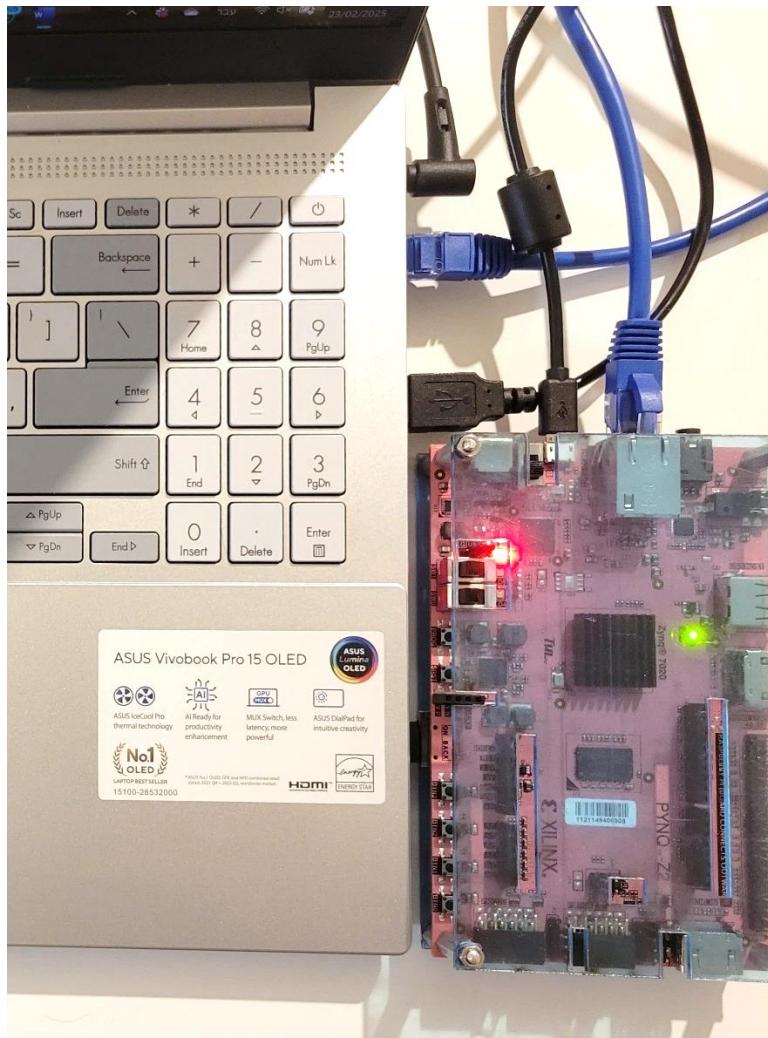
אך במקרה ההופך, ה-SLAVE רשאי להמתין לעליית *TVALID* *M\_AXIS\_TDATA* לפני שהוא מעלה את - *S\_AXIS\_TREADY*

זמן המינימלי להעברת הנתוניים הוא מחזור שעון אחד זהה במקרה *S\_AXIS\_TREADY* *M\_AXIS\_TVALID* ו-*M\_AXIS\_TDATA* עלם לגובה באותו מחזור שעון.

## 4. תכנון חלק החומרה של המערכת

### 4.1. הסבר כללי

המערכת כוללת את כרטיס ה-Z2 QYNQ, כבל USB, מחשב נייד, וכבל Ethernet. כבל ה-USB וcabל-h Ethernet מתחברים מהכרטיס אל המחשב הנייד. הcrcטיס מתחברת לרשת המקומית על ידי כבל Ethernet.



אייר 15 – צייד הפרויקט

באיר ניתן לראות חיבור חשמל מספק הכוח ואת חיבור כבל האינטרנט אשר מיועד לצורך בקרה עם כרטיס ה-QYNQ.

### 4.2. תהליך הבניה של המערכת

בשלב תכנון החומרה, נעשה שימוש נרחב בתוכנת Vivado 2019.1 אשר הותקנה על מחשבו האישי של המגיש. בעזרתה תוכנן ה-IP ופותח תיכון סכמת הבלוקים. לאחר כל שלב זה נעשו סימולציות לבדיקת עבודה תקינה של חלקים המערכת. לאחר הרכבת תיכון הבלוקים, יצירת קובץ bit.

המערכת החומרה שנבנתה מיועדת לשימוש עם כרטיס FPGA מסוג Z2-PYNQ. הכרטיס הוגדר על פי הנחיות הגדרת הכרטיס לשימוש ראשוןי כפי שהוסברו ב3.3. (איתחול הכרטיס).

### 4.3. הסבר אופן פעולה המערכת בשלבים

1. בשלב הראשון, הקוד Python דואג לשילוח המידע המרוףד ל אשר מוכנס בתורו לטור DMA וMOVBR לחומרה לצורך העיבוד. מכיוון שהrifod עצמו לא קורה במקביל לפחות אחת, ומכיון שזה מפשט את המערכת הוחלט לרפוד את המידע כבר בתוכנה (ע"פ שכמובן הדבר ניתן לעשות גם בחומרה באותה מידה).rifod דואג לכך שהמידע יהיה באורך שהוא כפול של 512 ביט (גודל בלוק בודד).rifod הוסבר 1.1.2.1 ("מימוש האלגוריתם"). בקצרה להזכיר- המימוש קורה באמצעותrifod ב' 1' ולאחריו אפסים עד שנשאר 64 ביט מלאה לכפולה של 512 (עד שהמידע כולל ' 1' והאפסים, mod 512 שווה  $512-64=448$ ).
2. בשלב הבא, שכבר קורה בחלק של PL (FPGA) המידע מוכנס בchunks של 32 ביט, ונשמר בתוך מערך של 16 רגיסטרים. כאשר הרגייסטר ה16 מתמלא, הסיגנל tready\_axiss המעיד כי המערכת מוכנה לקבל מידע יורד (Lo) והמערכת ממשיכה בחישוב לוח הזמן W.
3. במקביל לכנית המידע לתוך הרגיסטרים, המידע נכנס לתוך החישוב של המשתנים הפעילים (כפי שהוסבר בחלק של מימוש האלגוריתם). כל chunk משמש לחישוב המשתנים הפעילים הבאים (בתוספת לעוד חישובים והכנסת ערכים קבועים ממערך הקבועים A).
4. כאשר הרגייסטר ה16 מתמלא, הרגייסטר הראשון המושב על ידי נסחה 3, ודורס את המידע שהוא בפנים (כבר אין בו צורך). כך גם מחושב הרגייסטר הבא, וחוזר חלילה לכל  $16-64=48$ . הסבבים הנוגרים עד לחישוב משתני המצב ההתחלתיים החדשים וקבלת הבלוק הבא מהDMA.

**יודגש כי שלב 3 ושלב 4 מבוצעים במקביל** מה שניתן לבצע רק באמצעות חומרה ומהוות ייתרונו של מימוש חומרתי על פניו מימוש תוכני. בעצם המידע המוכנס נשמר ברגיסטר (לצורך חישוב W) וכן נכנס לסכימה לצורך חישוב המשתנים הפעילים הבאים.

5. בשלב הסופי, לאחר שמתקיים סיג널 last\_axiss המעיד על סיום המידע המוכנס, הבלוק האחרון מסיים את 64 הסבבים שלו, והתוכאה יצאת אל התוכנה באמצעות פרוטוקול AXI כפי שהוסבר.
6. בתוכנה המידע מוצג למשתמש, והתוכאה מחושבת דרך התוכנה לצורך השוואת בין שתי הדריכים. הצפי הוא כMOVן לקבל תוצאה טוביה יותר בחומרה מאשר בתוכנה, וזאת מכיוון שבחומרה - כפי שניתן לראות - יש תהליכי המתרחשים במקביל.

### 4.4. דיאגרמת בלוקים

#### 4.4.1. דיאגרמת הבלוקים מבט-על

החוمرة תוכננה ופותחה על גבי כרטיס Z2-PYNQ של חברת LUT, בסביבת עבודה VIVADO 2019.2 של חברת Xilinx.

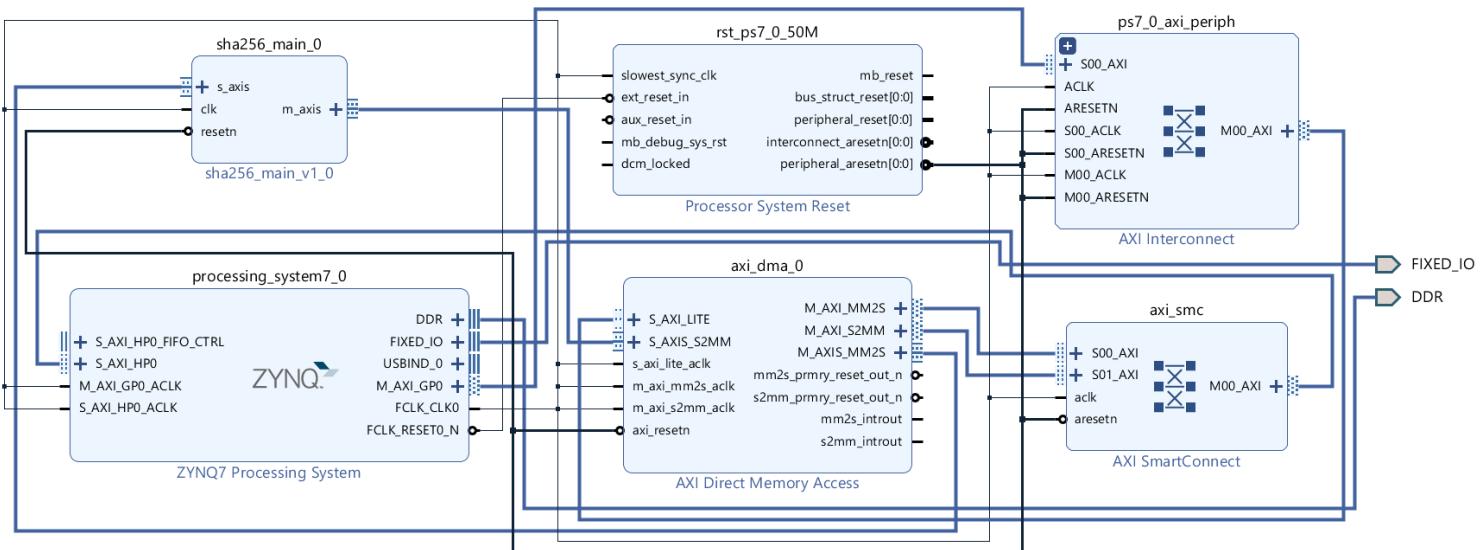
חלוקת המערכת העיקריים מתחלקים לשני קבוצות, IP In Bulit IP ו- Custom IP, המבנימים של Xilinx, IP IP שמכיל IP שנבנה בשבייל מימוש המערכת.

:Built In IP's

- PS-Processing System •
- DMA-Direct Memory Access •

:Custom IP

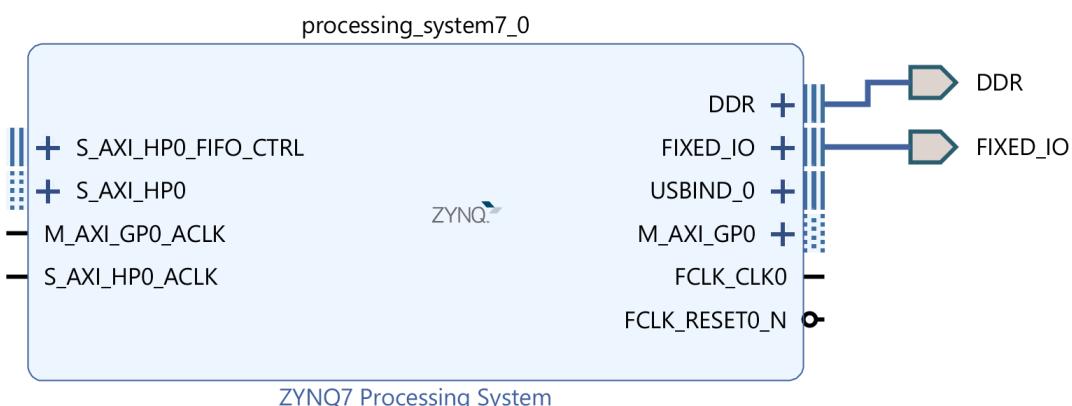
- sha256\_main\_0 •



אייר 16 – דיאגרמת בלקים המתארת את מימוש המערכת

#### PS – Processing System .4.4.2

כרטיס ה-FPGA ZYNQ-7000 הוא בעל שני ליבוטים מסוג ARM-Cortex A9. תפקוד המעבד הוא רחב והוא אחראי בין היתר על ייצור שעוניים בתדר הרצוי, קבלת כניסה מהמשתמש והוצאת מוצאים החוצה למשתמש, והתנהלות מול הזיכרון ברמת קריאה וכתיבה. בנוסף הוא שולט על הפריפרויות השונות באמצעות פרוטוקול AXI.



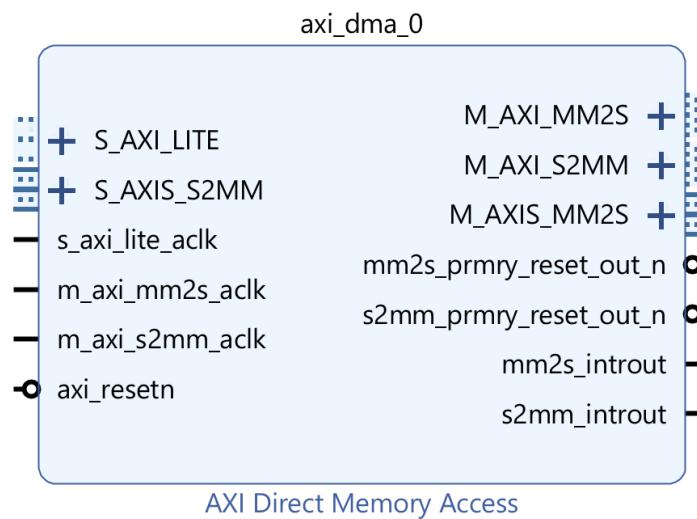
ZYNQ7 Processing System – 17

### כניות ויציאות:

- *FCLK\_CLK0* - אחד מהשעונים שעוברים מה-PS ל-PL. ה-ZYNQ יכול ליצור מספר שעונים נוספים במידת הצורך. הוגדר להיות בתדר של 100MHz.
- *N\_FCLK\_RESET0* - אות ה-*reset* שmagiu ביחד עם אות השעון. אות זה פועל בנמוך ומתחילה את רכיבי המערכת.
- *M\_AXI\_GPO* - זה ה-*Master* General Purpose Master במשק AXI. הוא מאפשר לחלק ה-PS לתקשר עם חלק ה-PL, והוא בדרך כלל מיועד לתעבורות נתונים ב מהירות נמוכה.
- *M\_AXI\_GPO\_ACLK* - כניסה השעון בשבי ה-*General Purpose Master*.
- *S\_AXI\_HPO* - התקן SLAVE בפרוטוקול AXI. משמש כאשר ה-PL רוצה גישה ל זיכרון DDR ב מהירות גבוהה. הוא בעל ביצועים גבוהים.
- *S\_AXI\_HPO\_ACLK* - כניסה שעון שמאפשרת לחלק ה-PL להתmeshק עם הזיכרון ב מהירות גבוהה.
- *S\_AXI\_HPO\_CTRL* - יחידת בקרה שמקושרת עם ה-*HPO\_S*. יכול לשמש כ-buffer בין יחידת ה-PS ויחידת ה-PL.
- *DDR* - הזיכרון של מעבד ה-ZYNQ.
- *FIXED\_IO* - כניות ומוצאים שהם מוגדרים למטרה ספציפית, כגון שעון ו-*reset*.

### DMA .4.4.3

ה-DMA (Direct Memory Access) הוא IP של חברת Achilli שמאפשר לקרוא ולכתוב מהזיכרון ללא התערבות בתהיליך של המעבד. על ידי כך המעבד חופשי לבצע פעולות אחרות בזמן העברת נתונים בין ה-PS ל-PL וכן הביצועים של המערכת משתפים.



DMA-Block – 18

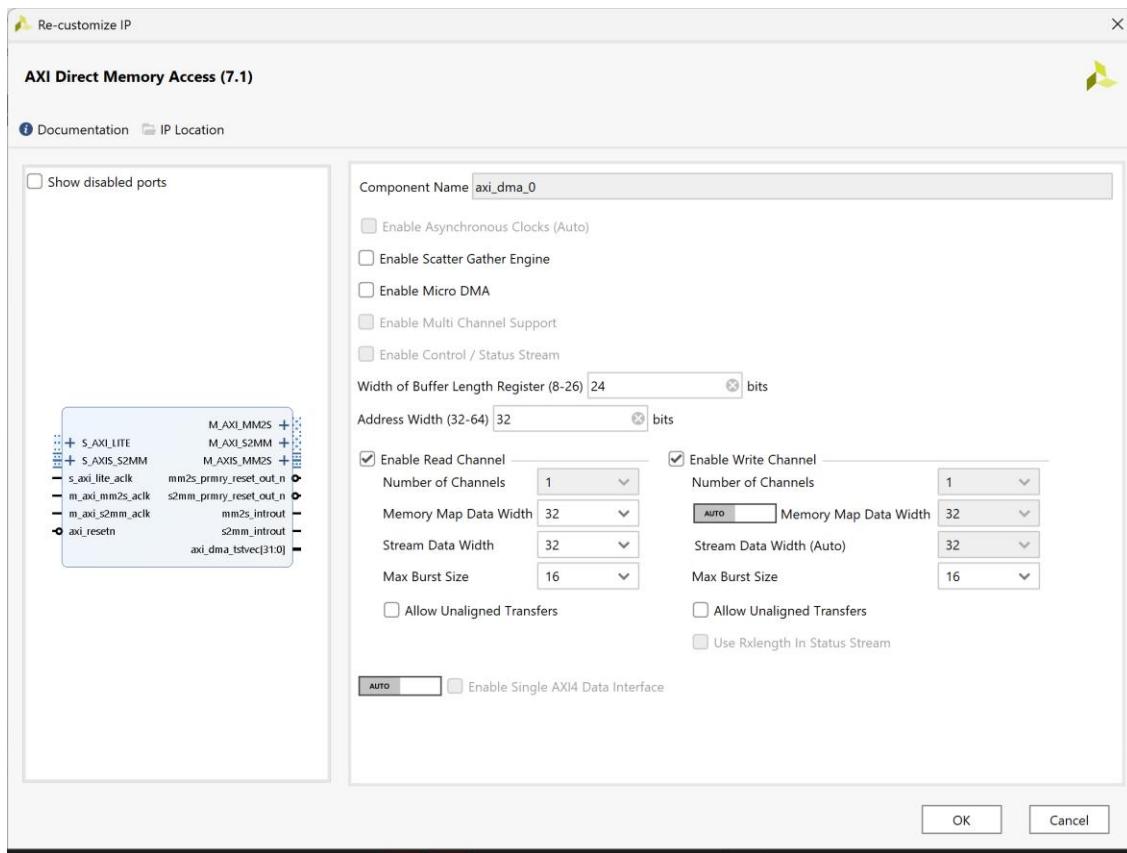
### כניסות ויציאות:

- **S\_AXI\_LITE** - זה ממשך SLAVE בפרוטוקול AXI\_Lite, המשמש לשיליטה ובקרה על DMA, וכן לקביעת תצורתו. באמצעות ממשך זה, המעבד יכול לבצע קריאה וכתיבה לאוגרי-DMA.
- **M\_AXI\_MM2S** - ממשך זה הוא MASTER בפרוטוקול 4AXI. ה-DMA עושה בו שימוש כדי לקרוא נתונים מהזיכרון ולאחר מכן לשולח אותם צרים. תהליך זה מאפשר העברת נתונים ייעילה ומהירה מהזיכרון אל היעד הנדרש.
- **M\_AXI\_S2MM** - גם ממשך זה הוא MASTER בפרוטוקול 4AXI. לאחר שה-DMA מקבל נתונים, הוא משתמש בממשק זה כדי לכתוב את הנתונים חזרה לזכרון. זהו תהליך הפוך ל-MM2S, שבו ה-DMA לוקח נתונים קלט וכותב אותם לזכרון, מה שמבטיח שהנתונים נשמרים בצורה מסודרת ונגישה.
- **S\_AXIS\_S2MM** - זהו ממשך SLAVE בפרוטוקול S\_AXIS, שבו פריפריות חיצונית שלוחות נתונים ל-DMA. ברגע שהנתונים מתקבלים דרך ממשך זה, ה-DMA משתמש בממשק ה-M\_AXI\_S2MM כדי לכתוב את הנתונים הללו לזכרון, מה שמאפשר אינטגרציה חלקה עם רכיבים חיצוניים.
- **M\_AXIS\_MM2S** - ממשך זה הוא MASTER בפרוטוקול S\_AXIS. לאחר שה-DMA קורא נתונים מהזיכרון באמצעות ממשך ה-S\_AXI\_MM2S, הוא שולח את הנתונים הללו צרים דרך ממשך זה אל הפריפריות החיצונית שעובדות בפרוטוקול S\_AXIS. תהליך זה מאפשר העברת נתונים רציפה ויעילה אל רכיבים חיצוניים המתחברים למערכת.

### הגדרת DMA:

ל-DMA יש ארבעה פרמטרים:

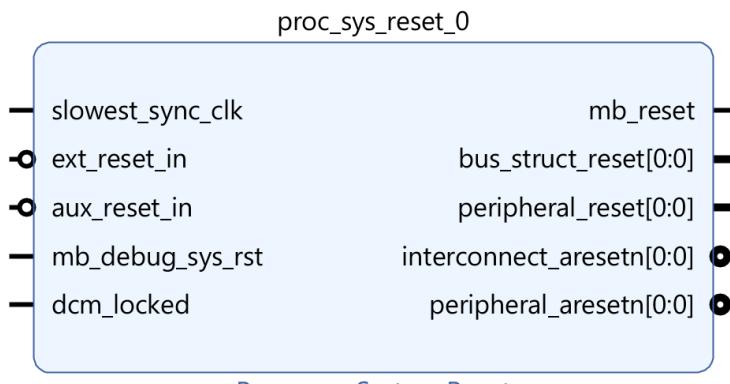
- **Address Width** - מגדר את מרחב הכתובות המשמש את ה-DMA. קובע את טווח הזיכרון אליו ה-DMA יכול לגשת.
- **Memory Map Data Width** - מציין את רוחב אפיק הנתונים של ה-DMA. זה רוחב הנתונים שה-DMA כותב או קורא מהזיכרון.
- **Stream Data Width** - מציין את רוחב הנתונים שה-DMA שלוח או מקבל צרים. הוא צריך להתאים לרוחב הנתונים של ה-IP שלו מהתממשק ה-DMA.
- **Buffer Length Register** - פרמטר זה קובע את רוחב האוגר המציין את אורך הנתונים שיש להעביר. במקרה אחריות, מציין כמה זמן יכולה להיות העברת DMA בודדת במנוחים של בתים. ערך זה נבחר להיות 24 על מנת לאפשר  $byte$   $1 = 16,777,215 - 2^{24}$  מה שיאפשר חישוב HASH של תמונות עד 16.77MB (ניתן גם להכניס ערכים מעל זה, אך מכיוון שהמנוע נועד לניטוח תמונות ותמונה הנחשבת גדולה מגיעה לגודל של 15MB, ההנחה היא כי רוחב באפר זה צריך להספיק).



איור 19 – הגדרת בлок ה-DMA

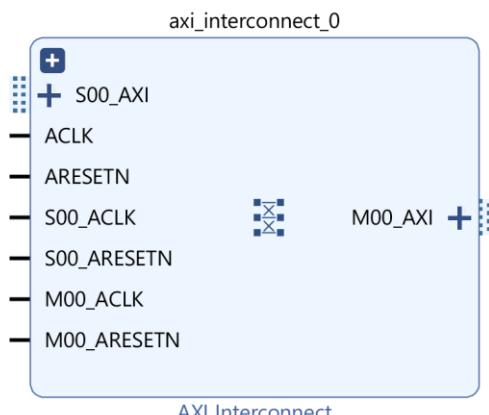
#### Processing System Reset .4.4.4

בלוק זה מטפל בניהול אוטות האיפוס לכל הבלוקים במערכת. הוא מספק בקרת איפוס מרכזית על ידי ניהול סוגים שונים, כגון איפוס תוכנה ואיפוס חומרה. הבלוק מבטיח תיאום בין ה-PL ל-PS במהלך האיפוס, תוך שמירה על תקינות המערכת.

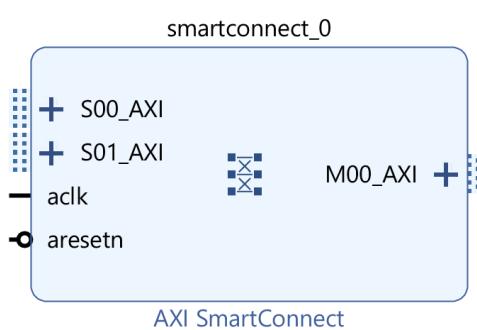


איור 20 – Processing System Reset block – 20

## Connect .4.4.5



איור – 21

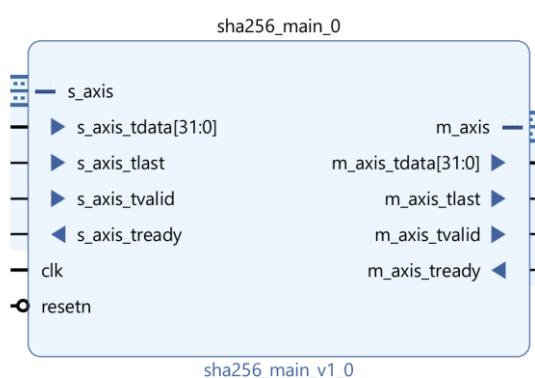


איור – 22

בלוקים אלו משמשים כմמשק חיבור בין רכיבי AXI במערכת. עיקנון הפעולה של בלוקים אלו, הוא לאפשר חיבור דינמי בין רכיבי AXI ללא צורך בקונFIGורציה מורכבת. בלוקים אלו מנהלים את הפקודות בין MASTER ל-SLAVE -ובכך מאפשרים העברת נתונים תקינה בין הרכיבים.

## Costume IP .4.4.6

הבלוק האחרון הוא בלוק ה sha256\_main\_0 השוכן בחלק 4.4.5. בשלב זה ניתן לראותו כboxed blackelogika ששתואר בהמשך, וUMBHOZ ניתן רק לראות את הסיגנלים הגנריים של פרוטוקול AXI (כך הסיגנלים נקראים גם בתוך המודולו sha256\_main כפוי שנייתן יהיה לראות בקטע קוד 4).



איור – 23 – ה sha256\_main\_0 Costume IP שעוצב –

## 4.5. תכנון ה IP costume

חלק זה עוסק בIMPLEMENTATION של IP-costume ש נכתב בשפת SystemVerilog שבבבלי מימוש הפרויקט. לצורך שימוש את החומרה, יש צורך בעבודה בפורוטוקול AXI, על מנת לתקשר עם המחשב כפי שהסביר. על מנת לנצל בצורה מקסימלית את המימוש החומרתי של המערכת, ניתן למקבל מספר תהליכיים. השלב של `hdpsound` והשלב של בניית מערכות לוח הזמן (W) ניתן לבצע בצורה מקבילה, וזאת כיוון שככל סיבוב מצריך את ה W הנוכחי בלבד ולא את הבאים. כך ניתן לשולח מה `Module` הראשי מידע על 16 ה W הקודמים ומספר קווים נוספים מאשר `modulen` אשר יחשב את ה  $W_{i+1}$ , ובמקביל לשולח את ה W ומספר קווים נוספים, `Module` אחר אשר יחשב את השינוי למשתנים הפעילים. ב `Module` המרכזי (`main`) יהיה בлок סינכרוני אשר יdag לՏנckeron של המערכת ולעדכן את השינויים במערכת שייקרו על פי השעון.

### 4.5.1 SHA-256 PKG קבוע המערכת ופונקציות שימושיות

על מנת שנitin יהיה להשתמש בפונקציה בכמה מודולו (Module) שונים מבלתי לבצע כפילותות בקוד (לרשום את אותה הפונקציה בשני מודולו שונים), וכן על מנת שנitin יהיה לאסוף את כל הקבועים שיישמשו אוטנו במקום אחד, נוצר קובץ `package` אשר מכיל את הקבועים ואת הפונקציות. בהמשך, כאשר צריך יהיה לחתת נתונים מקובץ זה למודולו השונים, צריך יהיה לעשות `import` מקובץ זה לנתונים הנדרדים לאוטו מודולו. הפרמטר `standard_initial_state` מייצג את  $H_0^{(1-8)}$  כפי שהסבירו (המשתנים ההתחלתיים הראשוניים). K מייצג את 64 הקבועים בנוסחה Temp1 (נוסחה 8). ולבסוף, הפונקציה `right_rotate` היא פונקציה המבצעת פועלות היזה סיבובית (להזזה רגילה יש אופרטור `BSL`. אך להזזה סיבובית אין, וכיון שהוא מופיע פעמיים לבצע פעולה זו - ועל מנת למנוע כפילותות בקוד- נבניתה פה פונקציה המבצעת פעולה זו).

```

1 `timescale 1ns / 1ps
2 package sha256_pkg;
3
4     parameter logic [31:0] standard_initial_state [7:0] =
5         {32'h6a09e667, 32'hbb67ae85,
6          32'h3c6ef372, 32'ha54ff53a,
7          32'h510e527f, 32'h9b05688c,
8          32'h1f83dab, 32'h5be0cd19};
9
10    parameter logic [31:0] k [64] =
11        {32'h428a2f98, 32'h71374491, 32'hb5c0fbcf, 32'he9b5dba5,
12          32'h3956c25b, 32'h59f111f1, 32'h923f82a4, 32'hab1c5ed5,
13          32'hd807aa98, 32'h12835b01, 32'h243185be, 32'h550c7dc3,
14          32'h72be5d74, 32'h80deb1fe, 32'h9bdc06a7, 32'hcl9bf174,
15          32'he49b69c1, 32'hefbe4786, 32'h0fc19dc6, 32'h240ca1cc,
16          32'h2de92c6f, 32'h4a7484aa, 32'h5cb0a9dc, 32'h76f988da,
17          32'h983e5152, 32'h831c66d, 32'h0b0327c8, 32'hbf597fc7,
18          32'hc6e00bf3, 32'h5da79147, 32'h06ca6351, 32'h14292967,
19          32'h27b70a85, 32'h2elb2138, 32'h4d2c6dfc, 32'h53380d13,
20          32'h650a7354, 32'h766a0abb, 32'h81c2c92e, 32'h92722c85,
21          32'ha2bfe8a1, 32'ha81a664b, 32'hc24b8b70, 32'h76c51a3,
22          32'hd192e819, 32'hd6990624, 32'hf40e3585, 32'h106aa070,
23          32'h19a4c116, 32'h1e376c08, 32'h2748774c, 32'h34b0bcb5,
24          32'h391c0cb3, 32'h4ed8aa4a, 32'h5b9cca4f, 32'h682e6fff3,
25          32'h748f82ee, 32'h78a5636f, 32'h84c87814, 32'h8cc70208,
26          32'h90beffa, 32'ha4506ceb, 32'hbef9a3f7, 32'hc67178f2};
27
28     //right_rotate
29     function automatic logic [31:0] right_rotate(input [31:0] x, input [4:0] n);
30         return x >> n | x << (32 - n);
31     endfunction
32 endpackage

```

קטע קוד 1 - משתנים ופונקציות לצורך ביצוע ה `sha256_pkg`

## 4.5.2. מיציאת מערך לוח הזמינים W.

כאשר מסתירים הבלוק הראשון, המילים החדשות (בW) מסופקות על ידי הבלוק sha256\_expansion. במודולו הראשי (main) ע"י שימוש במקס, המילים נכנסות למערך W מהמודולו הנוכחי, בהתאם לround\_index.

```

1  `timescale 1ns / 1ps
2  import sha256_pkg::right_rotate;
3
4  module sha256_expansion (input [3:0] round_index,
5          input [31:0] w[15:0],
6          output [31:0] expanded_word);
7
8  logic [32-1:0] s0,s1;
9
10 assign s0 = right_rotate(w[(round_index-15)%16], 5'd7) ^
11      right_rotate(w[(round_index-15)%16], 5'd18) ^
12      (w[(round_index-15)%16] >> 2'h3);
13
14 assign s1 = right_rotate(w[(round_index-2)%16], 5'd17) ^
15      right_rotate(w[(round_index-2)%16], 5'd19) ^
16      (w[(round_index-2)%16] >> 4'ha);
17
18 assign expanded_word = (w[round_index] + s0 + w[(round_index-7)%16] + s1);
19 endmodule

```

קטע קוד - sha256\_expansion - 2 - בלוק למציאת מערך לוח הזמינים W ב 15 round\_index >

כפי שניתן לראות, בשלב זה יש צורך בשימוש בפונקציה right\_rotate ולכן בשורה 2 עושים import לפונקציה זו, וכן יש למודולו גישה לפונקציה (כאילו היא כתובה בתוך המודולו). התחליף פשוט במינוח, ובהתאם לround\_index הנוכחי ו-16 הW הקודמים (כיוון שכפי שניתן לראות בסיסה 3, הערך היכן רוחוק של W שצריך על מנת למצוא את המילה הבאה בלוח הזמינים W, זה 16 מילימ אחוריה), ניתן להוציא את המילה הבאה, שנכנסת לモץ expanded\_word. הנוסחאות המומומשות במודולו זה הם: נוסחה 1 נוסחה 2 ונוסחה 3.

## 4.5.3. חישוב ה rounds

במקביל לחישוב מערך לוח הזמינים W, מחושבים הערכים החדשניים של המשתנים הפעילים (אשר יעדכנו בעליית השעון). החישוב לצורך חישוב הקבועים החדשניים מתואר בסיסות 10-4.

הчисוב הוא תהליך קומבינטורי, והערכים מחושבים תמיד (באמצעות שערים לוגיים) בהתאם לכניסות. אלא שהכニסות משתנות בעליית השעון, וכך כל עליית שעון ישר מחושבים הערכים החדשניים של המשתנים הפעילים ומחייבים בכינסה לרגיטרים (אגב, הרגיסטרים עצם משמשים לצורך חישוב החדשניים החדשניים. כך השני קורה וישר אחריו - בהשאה של כל השערים שבדרך - מחייבים המשנים החדשניים).

```

1 `timescale 1ns / 1ps
2 import sha256_pkg::right_rotate;
3 import sha256_pkg::k;
4
5 module sha256_round(input [31:0] word,
6                      input [31:0] state [7:0],
7                      input [5:0] round_index,
8                      output logic [31:0] new_state [7:0]);
9
10 logic [32-1:0] temp1 ,temp2, choice, majority, sum0, sum1;
11
12 /* state[7]=a    state[6]=b    state[5]=c    state[4]=d
13   state[3]=e    state[2]=f    state[1]=g    state[0]=h */
14
15 assign choice = (state[3] & state[2]) ^ (~state[3] & state[1]);
16 assign majority = (state[7] & state[6]) ^ (state[7] & state[5]) ^
17                  (state[6] & state[5]);
18 assign sum0 = right_rotate(state[7], 5'd2) ^ right_rotate(state[7], 5'd13) ^
19                  right_rotate(state[7], 5'd22);
20 assign sum1 = right_rotate(state[3], 5'd6) ^ right_rotate(state[3], 5'd11) ^
21                  right_rotate(state[3], 5'd25);
22 assign temp1 = state[0] + sum1 + choice + k[round_index] + word;
23 assign temp2 = sum0 + majority;
24
25 assign new_state[6:4] = state[7:5]; //assigning b,c,d (to be previous-state's a,b,c)
26 assign new_state[2:0] = state[3:1]; //assigning f,g,h (to be previous-state's e,f,g)
27 assign new_state[7] = temp1 + temp2;
28 assign new_state[3] = temp1 + state[4];
29
30 endmodule

```

קטע קוד 3 - בлок לחישוב המשתנים הפעילים הבאים (לצורך חילמת *round*)

כפי שניתן לראות, הנוסחאות שתוארו בקטע התאורטי מומומשות כאן בצורה מסודרת כך שבמוצא בлок זה - מחכים המשתנים הבאים שהפכו לנוכחים בעליית השעון. הנוסחאות שמומומשות פה הם:  
נוסחה 4 נוסחה 5 נוסחה 6 נוסחה 7 נוסחה 8 נוסחה 9 ונוסחה 10.

#### 4.5.4. בлок ה *main*

בבלוק המין הדבר הראשון שמתואר זה הכניסות והיציאות (ports) של המערכת. בשלב זה ניתן לראות אותה כמעין *black-box* אשר יש לו קווי כניסה וקווי מוצא. בקטע קוד 4 ניתן לראות את הכניסות והיציאות הללו.

```

1 `timescale 1ns / 1ps
2 import sha256_pkg::standard_initial_state;
3
4 module sha256_main( input clk,
5                      input resetn,
6                      input s_axis_tlast,
7                      input s_axis_tvalid,
8                      input [31:0] s_axis_tdata,
9                      input m_axis_tready,
10                     output logic [31:0] m_axis_tdata,
11                     output logic s_axis_tready = 0,
12                     output logic m_axis_tvalid = 0,
13                     output logic m_axis_tlast = 0);

```

קטע קוד 4 - תיאור כניסה ויציאה ה IP/*costume* - *sha256\_main*

על התיאור של הפורטים ניתן לראות את סקאלת הזמן (יותר רלוונטי לסימולציות ולוילדיציה של המערכת). ותחתיו ניתן לראות את הייבוא של המשתנים הפעילים ההתחלתיים (אשר יכנסו בתחילת התהילה לערך משתני המצב הפעילים, וכן למשתני המצב ההתחלתיים). הסיגנלים עצם כפויו שנitin לראות- נקבעו בשמות התואמים לsigmoidים הנדרשים לצורך שימוש פרוטוקול AXI. דבר זה אינו נדרש ונitin לקשר בין שמות מקוריים לבין sigmoidים, אך לצורך נוחות ובahirות הקוד- הכתיבה בוצעה בשמות הגנריים של הפרוטוקול.

בשלב הבא מתוארים sigmoidים הפנימיים של המערכת. sigmoidים אלו לא נראהים מבחן אך הם משמשים לצורך חישוב התוצאה ומלהר השימוש במערכת. ניתן לראות sigmoidים אלו בקטע קוד 5.

```

15 logic [31:0] word, expanded_word, w[15:0];
16 logic [31:0] state[7:0];
17 logic [31:0] new_state[7:0];
18 logic [31:0] initial_state[7:0];
19 logic [5:0] round_index = '0;
20 logic chunk_end = 0;
21 logic [31:0] sha[7:0];
22 logic [2:0] result_ptr = 0;
23 enum logic [1:0] {RECEIVE_DATA, FINISH_CALC, SEND_RESULT} state_t = RECEIVE_DATA;

```

קטע קוד 5 - sigmoidים הפנימיים של המערכת

כפי שניתן לראות בקטע קוד 5, כאן מוגדרים המילה (word) וכן sigmoid המילא (expanded\_word) אשר יחבר למוצא הבלוק expansion. ערך זה בתורו יכנס לword בסבבים 15-0-63 (כאשר בסבבים 15-0 יכנס המידע מהDMA). מוגדרים גם state (רגיסטרים) והnew\_state (קווים). בנוסף יש את round\_index אשר ייחדש המשמש כמספר counter למספר 64 הסיבובים, ולאחריהם מתאפשר ומחihil את הספירה מחדש. chunk\_end מיועד לעזור את הספירה של round\_index עד שהסכמה בסוף 64 הסבבים תסתיים. הרגיסטר sha (בגודל של 256 ביט. 8 רגיסטרים של 32 ביט כל אחד), יכול את תוצאות החישוב הסופית, ויעביר אותה למוצא axis\_tdata\_m על מנת להתאים לפרוטוקול AXI. הptr\_result מציין על האינדקס בתוך sha של המילה הנשלחת כרגע (כפי שניתן לראות, זה 3 ביטים מה שתואם בדיקת המוצא בעל ה-8 רגיסטרים (של 32 ביט כל אחד)). לבסוף, יש את state\_t אשר מכיל את שלושת מצבים המערכת: RECEIVE\_DATA, FINISH\_CALC, SEND\_RESULT (על מנת למש את פרוטוקול AXI המערכת תמש מכונת מצבים).

בתכנון חומרה - קצת בדומה לקוד תוכנה- יש את קוד הבסיס. שהוא יכול למש מספר פונקציות (מקביל למודולים). אך בסוף יש את המין (או כפי שבdag נקרא בחרומרה top). בתוך המין ממומשים שאר הפונקציות המשניות. כך גם כאן, ישנו את בלוק המין. או הsample\_top אשר מממש (instantiates) את המודולו הקודמים שהובאו והסבירו. הכניסות והיציאות של שני המודולו האלו מחוברים לsigmoidים פנימיים בבלוק המין, וכך המידע הנדרש נכנס, עובר את העיבוד הממוחשב באותו מודולו ו יצא במודולו השני. ניתן לראות בקטע קוד 6 את ה instantiation של שני מודולו אלו.

```

25  ↴      //instantiations
26  ↵      // Round calculation
27      sha256_round round (
28          .word(word),
29          .state(state),
30          .round_index(round_index),
31          .new_state(new_state) );
32
33      // Message schedule expansion
34      sha256_expansion expand (
35          .round_index(round_index[3:0]),
36          .w(w),
37          .expanded_word(expanded_word) );

```

קטע קוד 6 – instantiation - sha256\_main – modulen של שני הינדסרים

כפי שניתן לראות, נבחרו שמות דומים על מנת למנוע בלבול. בטור הסוגרים מופיע מה שנכתב למודולו המקורי, ומוחוץ הסוגרים זה הפורט אליו מתחברים.

השורה הבאה היא המוקס שהוזכר, אשר מזקיא לword או את expanded\_word או את s\_axis\_tdata, לפי index.round. ניתן לראות את מימוש המוקס בקטע קוד 7.

```
39  assign word = round_index[5:4]== 2'b00 ? s_axis_tdata : expanded_word;
```

קטע קוד 7 – sha256\_main – מימוש המוקס לצורך קביעת המילה הנוכחית

בשלב הבא נוכנסים לבlok הסינכרוני, המכיל את מכונת המצלבים שהוזכרה בקביעת הסיגנלים. בקטע קוד 8 ניתן לראות את שלב reset הsnsכרוני (פועל בנמור!).

```

41 ↴      always_ff @(posedge clk) begin
42  ↵          if (!resetn) begin
43              state_t <= RECEIVE_DATA;
44              result_ptr <= 0;
45              s_axis_tready <= 0;
46              m_axis_tvalid <= 0;
47              m_axis_tlast <= 0;
48              chunk_end <= 0;
49              round_index <= '0;
50              initial_state <= standard_initial_state;
51              state <= standard_initial_state;
52  ↵      end else begin

```

קטע קוד 8 – תחילת הבלוק snsכרוני - ריסט סיגנל פעיל בנמור

כמובן בשלב זה כל האותות מתאפסים לשלב הבסיסי state ו state (משתני ה state ו state initial\_state) שלם שהוגדר בpkg\_sha256 בקטע קוד 1). בשלב הבא נוכנסים לבlok קבלת המידע. блוק זה גם משתמש כ IDLE כאשר לא מוכנס סיגנל של valid, val, נשארים בבלוק זה ומתחילה לקבל את המידע המוכנס מהמשתמש. במקביל לקבלת המידע, מחושבים המשתנים הפעילים החדשניים וממשיך החישוב שלلوح הזמנים W. נשארים בבלוק זה עד לקבלת סיגנל של DMA. בקטע קוד 9 ניתן לראות את המצב זהה.

```

53 ⌂      case (state_t)
54 ⌂        RECEIVE_DATA: begin
55 ⌂          s_axis_tready <= round_index[5:4] == 2'b00 && round_index != 15;
56 ⌂          if (s_axis_tvalid && s_axis_tready || chunk_end || round_index[5:4] != 2'h0) begin
57 ⌂            if (chunk_end) begin
58 ⌂              foreach (state[i]) begin
59 ⌂                initial_state[i] <= initial_state[i] + state[i];
60 ⌂                state[i] <= initial_state[i] + state[i];
61 ⌂              end
62 ⌂              chunk_end <= 1'b0;
63 ⌂            end else begin
64 ⌂              w[round_index[3:0]] <= word;
65 ⌂              state <= new_state;
66 ⌂              round_index <= round_index + 6'b1;
67 ⌂              if (round_index[5:4] == 2'b00) begin
68 ⌂                if (s_axis_tlast) begin
69 ⌂                  state_t <= FINISH_CALC;
70 ⌂                  s_axis_tready <= 0;
71 ⌂                end
72 ⌂                chunk_end <= 1'b0;
73 ⌂              end else if (round_index == 6'd3) chunk_end <= 1'b1;
74 ⌂            end
75 ⌂          end
76 ⌂        end

```

קטע קוד 9 - מצב ה RECEIVE\_DATA של מוכנת המצביעים

בשורה הראשונה של הקוד ניתן לראות את התיאור המוצב הראשון (RECEIVE\_DATA). לאחריו מתוארת ההתנהגות של הסיגנלים השונים במצב זה. בתחילת התיאור שולח סיגנל של ready לחוצה DMA כל עוד המערכת לא פועלה (sheari אז round\_index==0). כאשר המערכת מתחילה לפעולה (מתקבל s\_axis\_tvalid מהDMA), נכנסים ל'if' ready את שאר הקוד של המצביע. שם עליה לקבלת כל מיליה חדשה עד שב 15 ready יורד וממשיך החישוב.

כאשר מתקבל מהDMA הסיגנל s\_axis\_tlast המערכת עוברת למצב FINISH\_CALC. בקטע קוד 10 ניתן לראות את התיאור של מצב זה.

```

77 ⌂      FINISH_CALC: begin
78 ⌂        if (round_index != 6'd0) begin
79 ⌂          w[round_index[3:0]] <= word;
80 ⌂          state <= new_state;
81 ⌂          round_index <= round_index + 6'b1;
82 ⌂        end else begin
83 ⌂          foreach (sha[i]) sha[i] <= initial_state[i] + state[i];
84 ⌂          m_axis_tdata <= initial_state[7] + state[7];
85 ⌂          m_axis_tvalid <= 1;
86 ⌂          state_t <= SEND_RESULT;
87 ⌂          result_ptr <= 7;
88 ⌂          m_axis_tlast <= 0;
89 ⌂        end
90 ⌂      end

```

קטע קוד 10 - מצב ה FINISH\_CALC של מוכנת המצביעים

במצב זה החישוב ממשיך, אלא שכבר אין קבלת מידע מהמשתמש, ורק מחושבים ה-48 הסבבים שנוטרו אחורי קבלת הבלוק האחרון. בסוף ה-48 סבבים, שМОנת המשתנים הפעילים מסתכמים יחד עם המשתנים הראשוניים (של תחילת הבלוק האחרון), אז נכנסים ל-8 הרגיסטרים של המשתנה sha. בנוסף, הpointer של שליחת המידע בחזרה DMA מתאפס על המספר 7 (לשילוח 8 החלקים של המוצא), והמוצא axis\_tdata מתקבל את הבלוק הראשון (מתוך שמונה) של המוצא יחד עם valid. המוצא הזה ישנה רק בקבלת axis\_tready, במצב הבא של המכונית מצבים.

כאשר axis\_round\_index חוזר למספר 0 (אחרי 63. מכיוון round\_index הוא 6 ביט ולכן אחורי המספר  $63^{p_6}$  מגע  $0^{p_6}$ ), החישוב האחרון (הסכמה) מתבצע, הפוינטර מתאפס ל-7 וועברים למצב השלישי והאחרון של שליחת המידע בחזרה DMA (מצב SEND\_RESULT). בקטע קווד 11 מתואר המצב האחרון שסגור את case, את else (שהRESET נמצא לעלה), ואת הנקודות always block.

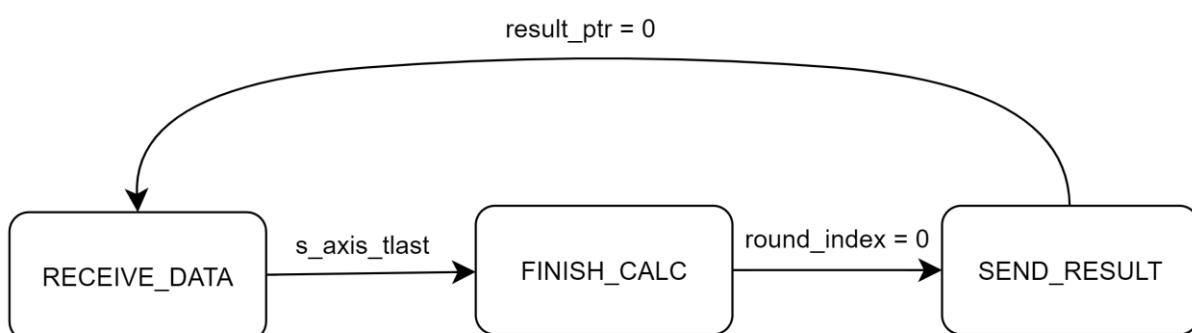
```

SEND_RESULT: begin
    if (m_axis_tvalid && m_axis_tready) begin
        if (result_ptr == 3'd0) begin
            m_axis_tvalid <= 0;
            m_axis_tlast <= 0;
            state_t <= RECEIVE_DATA;
        end else begin
            result_ptr <= result_ptr - 1;
            m_axis_tdata <= sha[result_ptr - 1];
            m_axis_tlast <= (result_ptr - 1 == 3'd0);
            initial_state <= standard_initial_state;
            state <= standard_initial_state;
        end
    end
end
endcase
end
end

```

קטע קווד 11 - sha256\_main - מצב ה SEND\_RESULT של מכונית המצבים

במצב זה הסיגנלים עובדים לפי פרוטוקול AXI והמידע נשלח כאשר axis\_tready מתקבל מוקן. בסיום פעולה המערכת הרגיסטרים של state ו initial\_state מקבלים שניים את הערכים הראשוניים שלהם על מנת להתכוון להפעלה הבאה של המערכת. בסיום שליחת המידע, המצב עובר בחזרה למצב הראשון של RECEIVE\_DATA. באIOR 24 ניתן לראות את דיאגרמה הבולונים המתארת את מכונית המצבים זו.

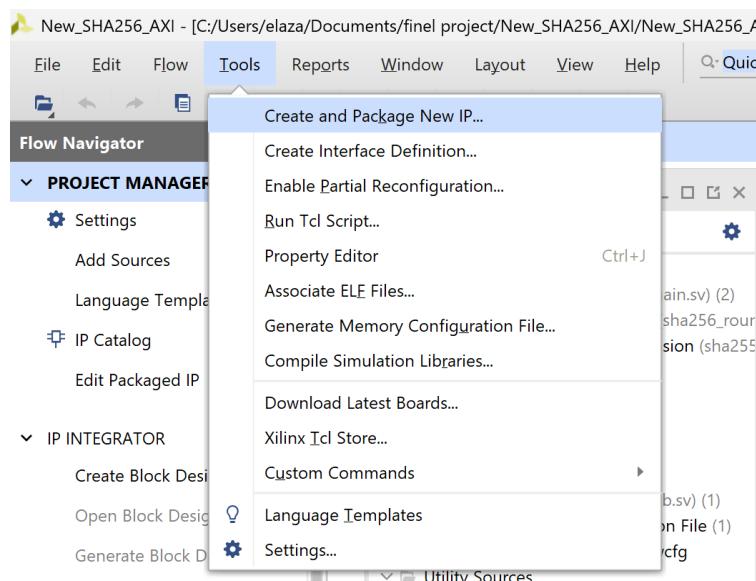


איור 24 – דיאגרמת בלונים של מכונית המצבים

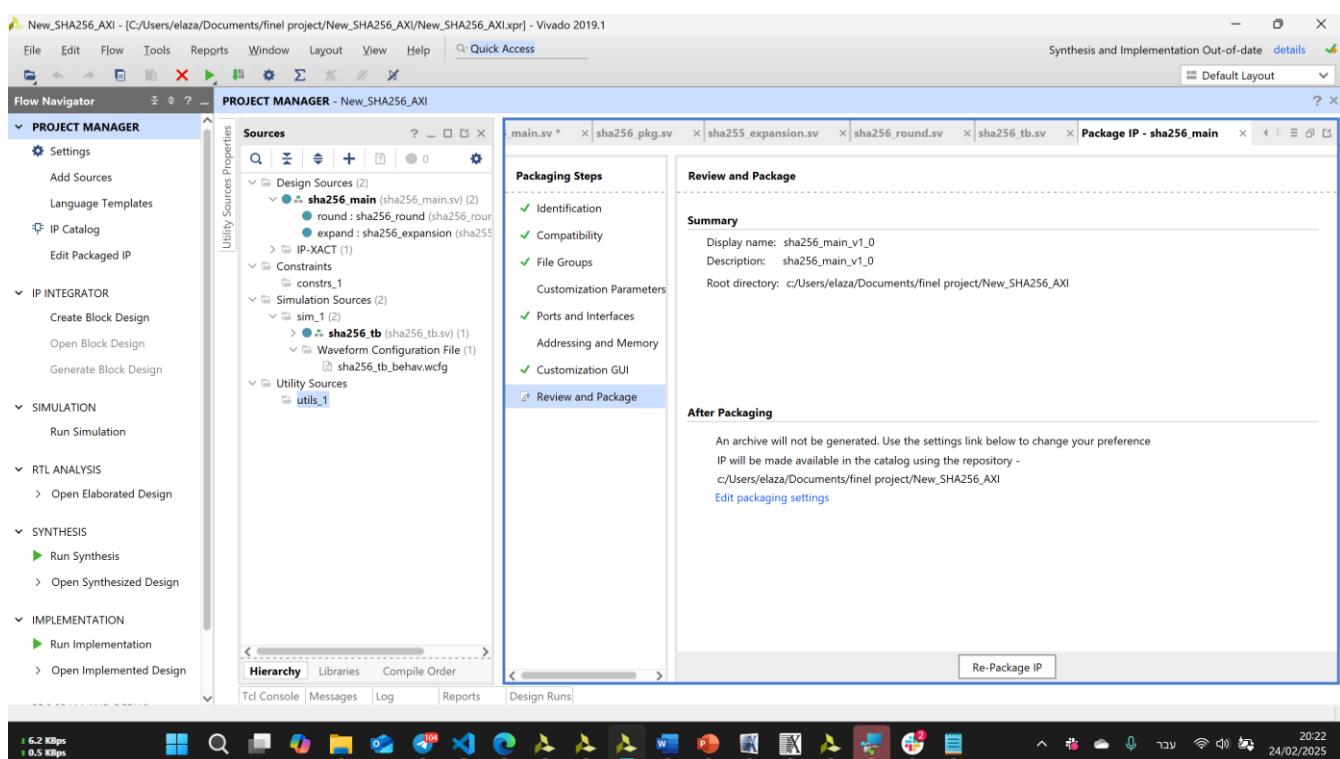
## 4.6. אריזת ה IP בפרוטוקול AXIS Costume

כאמור, על ה IP לעבוד ב프וטוקול AXIS על מנת שיוכל לתקשר עם DMA. הדרך להכניס את ה IP לשירות ה IPs ולהכניס אותו לדיאגרמת הבלוקים (כדי לחבר אותו לDMA) נדרשים הצעדים הבאים.

1. ניתן לראות דוגמא ביצילום מסך באירור 25.
2. לאחר יצירתה נוכל ללחוץ על ה IP Package והIP הוכנס לשירות ה IP של הפרויקט. ניתן לראות שלב זה ביצילום מסך שבאיור 26.



איור 25 – השלב הראשון ביצירת ה IP



איור 26 – השלב השני (והאחרון) ביצירת ה IP  
packaged IP (והاخזור)

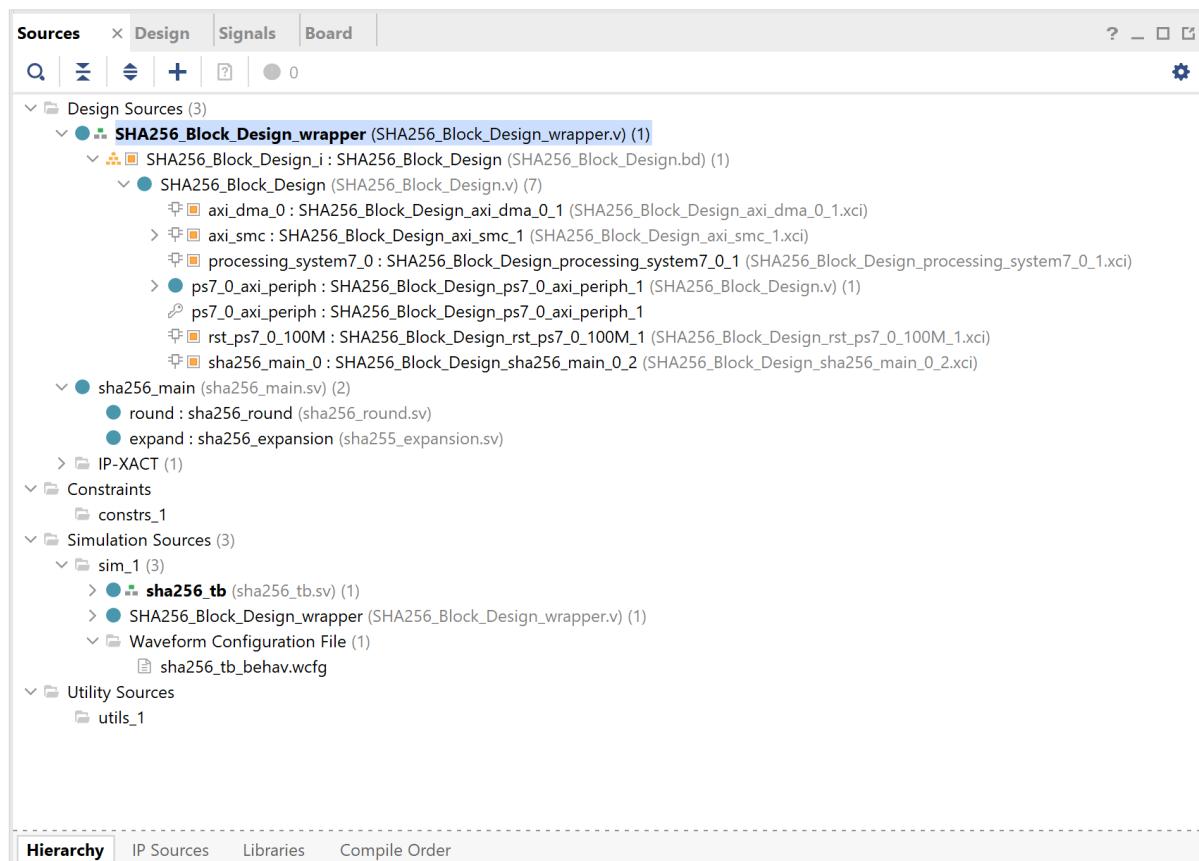
## 4.7. ייצור קבצי bitstream וה hwh

בשלב האחרון, אחרי עיצוב דיאגרמת הבלוקים, יוצרים את קבצי bitstream וה hwh. ניתן לחלק את השלב הזה לחמשה שלבים.

1. Validation - לדיאגרמת הבלוקים על מנת לוודא שהדיאגרמה תקינה (איור 28 ואיור 29).
2. HDL-VIVADO - יוצר מעתפת של שפת חומרה ל-Block Design Wrapper על מנת שייה ניתן להמשיך את תהליך ייצור קובץ bit.
3. Synthesis - מיפוי של קוד ה-HDL לרמת חומרה ושוררים לוגיים (איור 30).
4. Implementation - הטמעת החומרה (עדין לא צריבה) בכרטיס הфизי שבו נעשה שימוש בפרויקט (איור 30).
5. Bitstream - ייצור קובץ bit שאמור להיצרב על הכרטיס על מנת למש את המערכת (איור 31).

קובץ ה HWH-נוצ'ר אוטומטי מעיצוב IP של Vivado ומשמש את כרטיס ה PYNQ ליזיהו אוטומטי של תצורת מערכת ה FPGA.

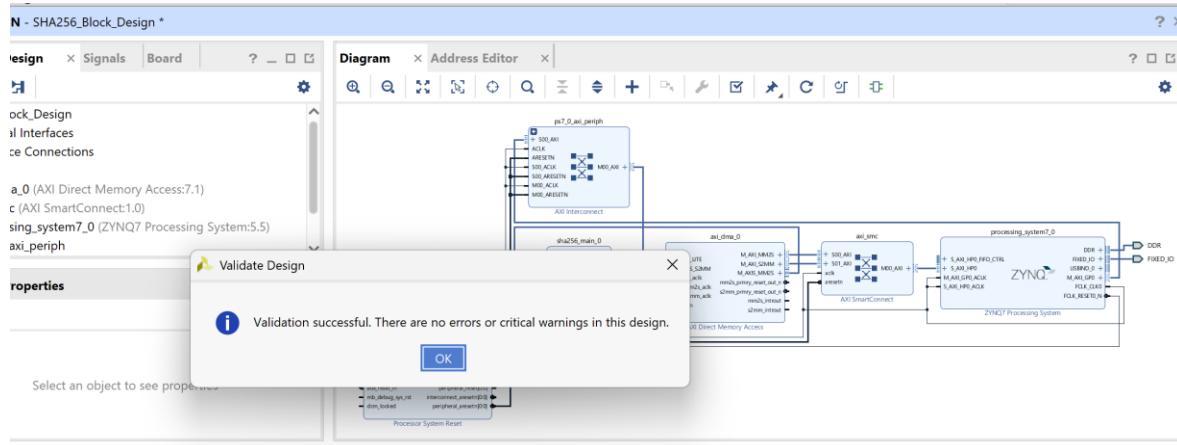
ניתן לראות באיור 27 את היררכיה של המערכת, כולל את הקודים האוטומטיים שמופיעותה התוכנה בעת ייצור Wrapper. באיור 28, איור 30 ואיור 31 ניתן לראות את תהליך הולידציה, ביצוע implementation וקבלת bitstream (וה hwh) שהמשר ייצור על גבי כרטיס PYNNQ (על FPGA).



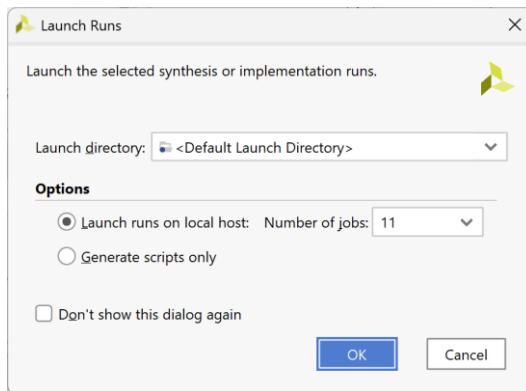
איור 27 – היררכיית המערכת אשר נשלחת למנוע לקבלת bitstream



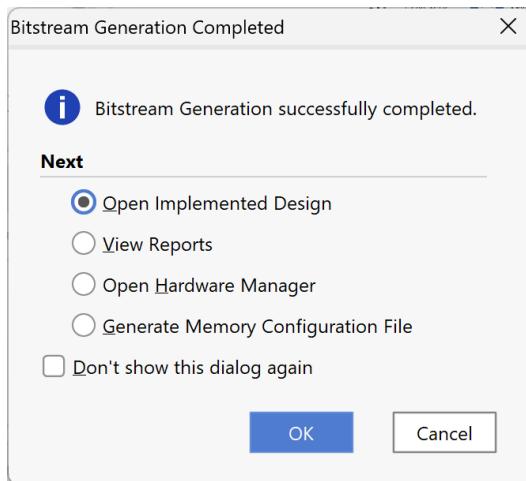
איור 28 – ביצוע וlidציה ל-block design



איור 29 – קבלת אישור כי הולידציה שבסה בהצלחה



איור 30 – הרצת סינטזה וIMPLEMENTATION



איור 31 – קבלת אישור כי bitstream נוצר בהצלחה

## 5. תכנון חלק התוכנה של המערכת

בחלק זה מתואר הקוד שמבצע את חומרת המערכת ובנוסף עושה את פעולות בפני עצמו.

### 5.1. ייבוא ספריות

```

1 from pynq import Overlay
2 from pynq.lib.dma import DMA
3 from pynq import allocate
4 import numpy as np
5 import struct
6 import time
7 from hashlib import sha256
8 import os

```

קטע קוד 12 – ייבוא ספריות פיתוח לצורך הקוד תוכנה

יבוא 3 השורות הראשונות, מגייע מהספריה **PYNQ** (*קיצור של Python on Zynq*), שהוא ספריית Python שנועדה להקל על עבודה עם חומרת **FPGA** (ר"ת של *Field-Programmable Gate Array*), במיוחד על גבי פלטפורמות מבוססות **Zynq** של Xilinx הכוללות מעבד ARM משולב עם לוגיקה ניתנת לתכנות.

- **Overlay** – רכיב זה מאפשר לטעון תוכן חומרה (**bitstream**) אל תוך ה-FPGA באופן DINMI. למעשה, מדובר במימוש של “תיכון חלק” (*Partial Reconfiguration*), שבו ניתן להחליף רכיבי חומרה מסוימים מבלי לכבות את המערכת.
- **DMA (Direct Memory Access)** – מנגנון המאפשר העברת נתונים ישירות בין זיכרון המערכת לבין ה-FPGA, תוך עקיפת המעבד המרכזי (CPU). זהו כלי חיוני להאצת עיבוד נתונים, מכיוון שהוא מפחית את העומס מהמעבד ומשפר את הביצועים על ידי העברת מידע ב מהירות גבוהה.
- **allocate** – פונקציה שמאפשרת להקצות זיכרון משותף (*contiguous memory buffer*) בין ה-CPU ל-FPGA. מאגרי זיכרון אלו הרכחיהם כאשר יש צורך לשתף נתונים בין רכיבי חומרה וביצוע חישובים בצורה יעילה.
- **numpy** – מספקת פעולות מערכיים עיליות החינויוות לטיפול בנתונים הנשלחים אל ומתקבלים מה FPGA.

NumPy מאפשרת עבודה עם מבני נתונים מותאמים לביצועים גבוהים, המספקים:

- **גישה ישירה לזכרון בצורת מערכיים רציפים** – מה שחשוב במיוחד בעת עבודה עם DMA ב-FPGA.
- **תמייה בפורמטים מדויקים של נתונים** – כמו uint32.uint64 או float32.float64, כדי להבטיח תאימות עם רוחב הנתונים של רכיבי החומרה.
- **ביצוע עיבוד מקבילי ויעיל** – ניתן להשתמש בפעולות וקטוריות של NumPy כדי לבצע טרנספורמציות ועיבוד נתונים במהירות לפני שליחתם ל-FPGA או לאחר קבלתם ממנו.

- **struct** – מאפשר להמיר נתונים בין ייצוגים בפייטון (כגון מספרים) לבין פורמט בינהר' קומפקטי. זה נדרש ב-SHA-256 מכיוון שיש צורך לאורך ההודעה בפורמט-big endian (כל תהליך SHA מתרחש בסוף הריפוד, בהתאם לתקן האלגוריתם).
- **time** – משמש למדידת זמן ביצוע של שימושי חומרה לעומת מימושי תוכנה. בדרך כלל, משתמשים ב-*time stamping* באמצעות **time** או **timeit** (בפייטון כדי להשוות את הביצועים, ולמדוד את היתרון של האצת חישובים ב-FPGA לעומת עיבוד תוכנתית על ה-CPU).
- **hashlib** – זאת ספריית פייטון המכילה את פונקציות ההאש השונות (יש עוד סוגים מעבר ל-SHA-256 המדבר בפרויקט זה). מכיוון שאין צורך בכל פונקציות ההאש השונות, מתוך ספרייה זו מיבוא החלק של sha-256 שגורם השוואה בין התוצאה שתתקבל באמצעות החומרה, לבין התוצאה שתתקבל בחישוב באמצעות ספרייה זו (חישוב שכובן יבוצע ב-CPU).
- **os** – משמש לצורך ביצוע פעולות על קבצים, כמו בדיקה אם תיקיות קיימות, קבלת גדי קבצים ועוד. מכיוון שהчисוב יבוצע על קובץ תמונה, והתוכנה "תשלוֹפּ" את הקובץ של התמונה מתוך האחסון של ה-**PyNNQ**, לכן צריך את הספרייה זו.

## 5.2. חישוב תוצאה HASH ע"י החומרה

### 5.2.1. ריפוד המידע

```

9
10 def calculate_sha256.hardware(filepath: str) -> tuple:
11
12     def sha256_pad(message: bytes) -> np.ndarray:
13         length = len(message) * 8
14         message += b'\x80'
15         while (len(message) % 64) != 56:
16             message += b'\x00'
17         message += struct.pack('>Q', length)
18     return np.frombuffer(message, dtype='>u4')

```

קטע קוד 13 – חלק התוכנה האחראי על ריפוד המידע כפי שמוגדר ב프וטוקול SHA

בקטע קוד 13 ניתן לראות בשורה הראשונה את ההגדלה של הפונקציה `calculate_sha256.hardware` אשר תיקרא בפונקציית החה (חלוקת הפונקציות היא לצורך `readability`). בתוך הפונקציה הזו, מוגדרת הפונקציה הדואגת לריפוד המידע.

1. בשלב הראשון מוגדרת הפונקציה (מקבלת אובייקט מסוג `bytes` ומחזירה מערך של `uint64`).
2. בשורה הבאה (שורה 13) מחושב אורך ההודעה (מכפל ב-8 מכיוון שהפונקציה `len` נותרת תוצאה ביחידות של `byte` ובשביל פרוטוקול SHA נדרש אורך ב-`bits` `byte=8`).
3. בשורה 14 מצפיפים (concatenation) בית של '1' 8 בHex זה 1000, והוא אחורי זה בשביל להשלים ל-`byte` ע"מ לקבל 100000000 (10).
4. בשורות 15 ו-16 רצה לוලאה המוסיפה אפסים (rifod) למידע הנכנס עד להישארות של 8 בתים (64 ביט) שם יכנס אורך ההודעה (לפי כללי הריפוד של SHA).

5. בשורה 17 מתווסף למידע 8 הבטים של אורך ההודעה (ביחידות של בית) כדי שיחסב בשורה .(13)

ב struct.pack משמעות הסימונים זה:

- > - מסמן Big-ending

- Q - מסמן מספר שלם, Unsigned, בגודל של 64 בית.

6. בשורה 18, הפונקציה מוחזירה את המידע, מחולק לחתיכות של 32 בית (4 byte).

לטיכום- פונקציה זו:

- מוסיפה 1 בית בסוף ההודעה.

- מוסיפה 0 ים עד שהאורך מתאים (448 מודולו 512 בית, או 56 בית מודולו 64).

- מוסיפה את אורך ההודעה המקורי כמספר 64 בית בפורמט Big-Endian.

- מmirה את הנתונים למלים של 32 בית לצורך עיבוד בSHA-256.

### 5.2.2. צריבת ה-bitstream והכנת המידע לשיליחה DMA

```

19
20     overlay = Overlay("SHA256_BlockDesign_wrapper.bit")
21     dma = overlay.axi_dma_0
22
23     with open(filepath, "rb") as f:
24         data = f.read()
25     input_data = sha256_pad(data)

```

קטע קוד 14 – צריבת ה-bitstream ושמירת התמונה בזיכרון בינארי

בקטע קוד 14 המידע הנמצא בקובץ "SHA256\_BlockDesign\_wrapper" (אשר כל תמצית החלק של החומרה נמצאת שם), נוצרב על גבי FPGA.

ושורה 21 (dma = overlay.axi\_dma\_0) נותנת גישה לבקר ה DMA - (גישה ישירה לזכרון) מתוך העיצוב החומרתי. ה-0\_axi dma הוא שם מופיע ספציפי מתוך העיצוב החומרתי, וה-DMA מאפשר העברת נתונים יعلاה בין זיכרון ה-CPU ל-FPGA ללא התערבות של ה-CPU.

בשורות 23 ו-24 הקוד פותח את הקובץ שנבחר במצב בינארי וקורא את כל תוכנו לזכרון. בשורה 25 הקוד קורא לפונקציית sha256\_pad (שנמצאת בקטע קוד 13) על מנת לעצב את הנתונים בצורה נכונה לפי דרישות SHA-256.

רגיסטר ה input\_data מכיל עכשו את ההודעה המרופדת כמערך PyNum של מילים בגודל 32 סיביות ומוכנה לשיליחה לחומרה לצורך עיבוד.

### 5.2.3. הקצאת מהסניות לשיליחה וקבלת המידע

```

26
27     data_buffer = allocate(shape=(len(input_data),), dtype=np.uint32)
28     output_buffer = allocate(shape=(8,), dtype=np.uint32)
29     np.copyto(data_buffer, input_data)

```

קטע קוד 15 – הקצאת מהסניות זיכרון עבור קלט ופלט המידע

שורה 27 של הקוד יוצרת מהסנית זיכרון מיוחדת לנוטוני הקלט השני לגשת אליהם גם מה-CPU וגם מה-FPGA.

- הצורה (Shape) תואמת לאורך ההודעה (המידע) המרופדת (כיוון `shadata`) עבר כבר את הריפוד בשורה 25 בקטע קוד 14.
  - סוג הנתונים הוא 32 סיביות מסוג `unsigned int` (`np.uint32`)
- בשורה 28 יוצרים מחסנית נוספת עבור פלט החישוב בגודל 8 מכיוון ש SHA-256 מייצר גיבוב בגודל 256 סיביות ( $8 \times 32$  סיביות).
- בשורה 29 מעתיקים את נתוני הקלט `data_input` (המידע שרופד כבר מוקן לשיליחה) לתוך המחסנית שהוקצתה לכך (`data_buffer`).

#### 5.2.4. שליחת המידע לחישוב חומרתי וקבלת התוצאה

בשלב הבא המידע נשלח DMA ל计较 החישוב, ומתקבל בחזרה לתוך הערך `output_buffer`.

```

30
31     start_time = time.time()
32     dma.sendchannel.transfer(data_buffer)
33     dma.sendchannel.wait()
34     dma.recvchannel.transfer(output_buffer)
35     dma.recvchannel.wait()
36     execution_time = time.time() - start_time

```

קטע קוד 16 – שליחת המידע DMA וקבלתה שלו בחזרה מהDMA

בשלב הראשון (שורה 31), מופעל "סטופר" על מנת למדוד את הזמן שהוקח לחומרה לחשב את תוצאת הHASH של המידע המוכנס (פונקציית SHA-256). בשורות הבאות המידע נשלח DMA (שורה 32), בשורה 33 DMAה "מחכה" שהמידע יסימן להישלח, ואחרי זה DMAה DMAה מכוון לחכotta לתגובה מהחומרה (FPGA) של תוצאה HASH. בשורה 35 DMAה DMAה מכחכת להשלמת העברת המידע של התוצאה ובשורה 36 "עוצרם את הסטופר" ומפתחים את זמן הסיום מזמן ההתחלת. התוצאה כמובן נותרת את זמן החישוב (כולל זמן העברת וקבלת המידע מתוך ולתוך DMA), והתוצאה נשמרת במשנה `execution_time`.

#### 5.2.5. קבלת התוצאה ושמירתה בפורמט נוח להציג

```

37
38     sha256_result = ''.join(f'{word:08x}' for word in output_buffer.tolist())
39     return sha256_result, execution_time

```

קטע קוד 17 – שמירת התוצאה בתצורה נוחה להציג ושליחת המידע החוצה מהפונקציה

ממיר את מערך NumPy `output_buffer.tolist()` לרשימה של Python.

(`... for word in f'{word:08x}'`) מעצב כל מילה בגודל 32 סיביות כביטוי הקסדצימלי באורך 8 תווים ומחבר את כלן יחד.

כך המידע הסופי מוצג בHex ונראה כרצף מספרים בHex באורך של 256 ביט (64 ספרות Hex).  
שורה 39 סוגרת הפונקציה `calculate_sha256_hardware` ומחזירה את תוצאה החישוב ואת הזמן שהחישוב לקח.

### 5.3. חישוב תוצאה ה HASH ע"י התוכנה

בשלב הבא, מחושבת התוצאה של ה HASH (ע"י פונקציית SHA-256) בתוכנה (CPU) ע"י פונקציה ייעודית של פיטון, לצורך השוואת המתקבלת מהחומרה וכן לצורך השוואת זמן החישוב. בקטע קוד 18 ניתן לראות את מימוש הפונקציה.

```

40
41 def calculate_sha256_software(filepath: str) -> tuple:
42
43     with open(filepath, "rb") as f:
44         data = f.read()
45
46         start_time = time.time()
47         sha256_result = sha256(data).hexdigest()
48         execution_time = time.time() - start_time
49
50     return sha256_result, execution_time

```

קטע קוד 18 – פונקציה לחישוב תוצאה ה HASH ב CPU ומדידת זמן חישוב זה

הפונקציה מוגדרת בשורה 41 (נקראת calculate\_sha256\_software מסיבות מובנות). בשורות 43 ו 44 הקובץ נפתח והמידע שלו נשמר במשתנה data (זהה לשורות 23 ו 24 בקטע קוד 14). אח"כ בשורות 46 עד 48 התוצאה של ה HASH נמדדת ונשמרת בתוצאה ון נמדד הזמן שהוקח לבצע חישוב זה.

בשורה 50 המידע נשלח החוצה (למיון שיקרא לפונקציה), גם התוצאה של ה HASH וגם הזמן שהוקח לבצע את החישוב.

### 5.4. הצגת רשימת הקבצים לחישוב ה HASH

באיחסון של PYNQ ישנה תיקייה בשם pictures. קבצי התמונות הנמצאים בתוך תיקייה זו יוצגו בשלב זה החוצה למשתמש על מנת שיוכיל לבחור מתוכם את הקובץ הרצוי (במידה והמשמש רוצה לחשב את ה HASH של תמונה מסוימת, עליו להעלות תמונה זו לתיקייה הייעודית pictures). המערכת תומכת בפורמטי הקבצים הבאים:

|      |   |   |
|------|---|---|
| jpg  | . | 1 |
| jpeg | . | 2 |
| png  | . | 3 |
| gif  | . | 4 |
| bmp  | . | 5 |
| tiff | . | 6 |
| webp | . | 7 |

הגבלה על קבצים אלו בלבד היא נטו הצורה בה נכתב הקוד, ואיןנה כלל הגבלה של החומרה על ביצוע חישוב לכל סוג קובץ אחר כמפורט. בקטע קוד 19 ניתן לראות את המימוש של הפונקציה זו.

```

51
52 def list_available_images():
53     pictures_dir = "pictures"
54     if not os.path.exists(pictures_dir):
55         print(f"\nError: '{pictures_dir}' folder not found!")
56         return []
57
58     image_extensions = ['.jpg', '.jpeg', '.png', '.gif', '.bmp', '.tiff', '.webp']
59     all_files = os.listdir(pictures_dir)
60     images = []
61     for file in all_files:
62         file_lower = file.lower()
63         for ext in image_extensions:
64             if file_lower.endswith(ext):
65                 images.append(file)
66                 break
67     if not images:
68         print(f"\nNo image files found in '{pictures_dir}' folder!")
69         return []
70
71     print("\nAvailable images:")
72     for i, img in enumerate(images, 1):
73         filename, extension = os.path.splitext(img)
74         print(f"{i}. {filename}")
75     return images

```

קטע קוד 19 – פונקציה להציג הקבצים שנמצאו לחישוב

בשורה 52 הפונקציה מוגדרת (נקראת `list_available_images`). הפונקציה לא מקבלת כלום ולא מוציאה כלום, מכיוון שכל מה שקרה בה זה הדפסת ערכים לterminal. הסבר שורה שורה של הפונקציה:

1. `pictures_dir = "pictures"` - קבע את שם התקינה בה התוכנית תחפש את התמונות זו ערך קבוע שמצויב על תקינות "pictures" כפי שהוא.
2. `if not os.path.exists(pictures_dir):` - בודק אם תקינות "pictures" קיימת באמצעות הפונקציה `os.path.exists()` זיהוי בדיקה כדי למנוע שגיאות אם התקינה לא קיימת.
3. `print(f"\nError: '{pictures_dir}' folder not found!")` - אם התקינה לא קיימת, תודפס הודעה שגיאה למשתמש
4. `return []` - מחריר רשימה ריקה אם התקינה לא קיימת, כך שהתוכנית לא תקרו, והפונקציה שקרה לה תוכל לטפל בתוצאה הリーיה.
5. `image_extensions = ['.jpg', '.jpeg', '.png', '.gif', '.bmp', '.tiff', '.webp']` - מגדר רשימה של סימנות קבצים שנחשבות לתמונות. כל סימנת מייצגת פורמט תמונה אפשרי (למשל .jpg, .png, .gif וכו').
6. `all_files = os.listdir(pictures_dir)` - מקבל את כל הקבצים והתקינות בתיקיות התמונות.
7. `images = []` - יוצר רשימה ריקה בשם `images` שתשתמש לאחסן שמות הקבצים שזוהו כתרומות. הרשימה הזאת תכיל רק את הקבצים שמסתיימים באחת מהסימנות שהוגדרו קודם.
8. `for file in all_files:` - עובר על כל שם קובץ ברשימה הקבצים `all_files`.

9. () file.lower = file.lower - ממיר את שם הקובץ לאותיות קטנות על מנת לאפשר השוואת הסדרת רגישות לאותיות (כך שניתן להזיהות את הסימות גם אם היא כתובה באותיות גדולות וגם אם היא כתובה באותיות קטנות).

```
for ext in image_extensions: .10
if file_lower.endswith(ext):
    images.append(file)
    break
```

עבור כל קובץ, הפונקציה בודקת אם שם הקובץ מסתיים באחת מהסימות שמצויפות ברשימה image\_extensions. הפונקציה endswith(ext) בודקת אם (שם הקובץ) מסתיים בסימות כלשהי מהרשימה אם הקובץ מסתיים בסימת חוקית (מהרשימה), הוא מתווסף לרשימה images.

11. if not images: - בודק אם הרשימה של התמונות ריקה. זה קורה אם התיקייה קיימת אבל אין בה קבצים מהסימות שהוגדרו ב image\_extensions.

12. print(f"\nNo image files found in '{pictures\_dir}' folder!") - מדפיס הודעה אם לא נמצאו קבצים עם סימת מהרשימה ב image\_extensions (מה שמיידע את המשתמש צריך להוסיף לתיקייה).

13. [] return - מחזיר רשימה ריקה אם לא נמצאו תמונות. כמו במקרה הקודם, זה מאפשר טיפול נכון על ידי הפונקציה שקרה לה

14. (".".join(["Available images: "])) - מדפיס כותרת לרשימה התמונות הזמינות (תו השורה החדשה (הו) מוסיף רוח למען קרייאות טוביה יותר).

15. for i, img in enumerate(images, 1): - עובר על הרשימה של שמות התמונות

- מנת למספר את התמונות מ-1 למשך המשתמש

```
filename, extension = os.path.splitext(img).16
print(f"{i}. {filename}")
```

מדפיס כל תמונה עם מספר לבחירה

17. return images - מחזיר את הרשימה המלאה של שמות הקבצים שנמצאו

## 5.5. פונקציית ה main

בשלב האחרון מוגדרת פונקציית החומרה אשר תרצה ותשמש בפונקציות שהוגדרו עד כה.

### 5.5.1. הציג הוכתרת למשתמש

בשלב הראשון, מוצגת הוכתרת של הפרויקט למשתמש:

```
=====
SHA-256 Hardware Accelerator
```

```
Calculate SHA-256 hash using FPGA acceleration
=====
```

בקטע קוד 20 ניתן לראות את הקוד המבצע את הדפסו המתואר לעיל

```

76
77 def main():
78     print("=" * 80,
79         "                               SHA-256 Hardware Accelerator",
80         "                               Calculate SHA-256 hash using FPGA acceleration",
81     "=" * 80, sep="\n")

```

קטע קוד 20 – הדפסת הברורת למשתמש

בשלב הבא מוצגות למשתמש האופציות שנמצאו בתקייה pictures (כפי שהסביר, הקוד עובר על התמונות הנמצאות בתקייה זו ומציג אותן ממושפרים). אם המשתמש רוצה לחשב את HASH של תמונה מסוימת, עליו להכניס תמונה זו לתקייה pictures.

#### 5.5.2. הציג האופציות למשתמש וקבלת תגובה

בשלב הבא יוצג למשתמש רשימת התמונות הזמיןות (הנמצאות בתקייה הייעודית).

```

82
83     while True:
84         images = list_available_images()
85         if not images:
86             return
87
88         print("\nEnter image name or number (or 'q' to quit)")
89         choice = input("Choice: ").strip()
90
91         if choice.lower() == 'q':
92             print("\nExiting program...")
93             return

```

קטע קוד 21 – הציג האופציות למשתמש וקבלת תגובה

בשלב זה יש לולה אינטסיפית, שתגבורות שונות יוציאו אותנו מהלולה וימשיכו את הקוד. בשורה 84 מודפסות כל האופציות האפשריות (כמו שהסביר בהසבר של קטע קוד 19). אם לא נמצאו תמונות (רשימה ריקה), הפקציה מסיימת את ביצוע התוכנית (return).

אם כן נמצאו תמונות, משתמש מוצג prompt והוא מתבקש להכניס קלט אחד מהקבצים המוצגים לו (או את האינדקס של הבחירה שלו). התוספת () strip() נועדה להסיר רווחים מיוטרים אם בטעות נלחכו אליו ע"י המשתמש. הערך של הבחירה שלו נשמר בchoice.

אם המשתמש לחץ על q המערכת יצאת מהתוכנית (בשורות 91 עד 93).

#### 5.5.3. התנהלות בקבלת מספר מהמשתמש

```

94
95     if choice.isdigit():
96         idx = int(choice) - 1
97         if 0 <= idx < len(images):
98             filename = images[idx]
99         else:
100             print("\nError: Invalid number! Please try again.")
101             continue

```

קטע קוד 22 – התנהלות בקבלת מספר מהמשתמש

• **choice.isdigit()** בודקת אם הקלט הוא מספר באמצעות .

- אם אכן התקבל מספר, הוא מומר לאינטגר, מחוסר ב1 (כי הרשימה מוצגת ממוספרת מ-1).
- אבל רישימות בPython מתחילה מ-0).
- האינדקס נבדק אם הוא בתווך התמונות הזרימנות.
- אם נכון, הוא מוגדר כ filename ועליו יבוצע החישוב.
- אם לא נכון (מחוץ לתחום), מדפסה הודעה שגיאיה וממשיכה לשאול את המשתמש שוב להכניס קלט וכו'.

#### 5.5.4. הדרישות בקבלת מיל מהמשתמש

```

102     else:
103         found = False
104         if choice in images:
105             filename = choice
106             found = True
107         else:
108             for img in images:
109                 name_without_ext, _ = os.path.splitext(img)
110                 if name_without_ext == choice:
111                     filename = img
112                     found = True
113                     break
114             if not found:
115                 print("\nError: Image not found! Please try again.")
116                 continue
117
118             filepath = os.path.join("pictures", filename)
119             break

```

קטע קוד 23 – הדרישות בקבלת מיל מהמשתמש

- ה **else** בתחילת הקוד מתייחס לבדיקה בקטע קוד 22 בשורה 95, אם הקלט הוא מספר. אם לא, הוא חייב להיות מיל (תווים שאינם מספרים).
- בשורה הבאה משתנה found מואתחל לערך false. לאחר מכן נבדק הקלט (choice) אם הוא תואם בדיקן לאחד מהתמונות שנמצאו בתיקייה. אם כן, אז found משתנה לtrue, והמשתנה filename מקבל את הערך של הקובץ זה (שורות 105 ו-106).
- אם choice (בחירה המשתמש) לא תואמת לאחד משמות הקבצים בתיקייה, נערךת בדיקת השוואה מול שמות הקבצים ללא הסויימת שלהם.
- הלולאה רצה על כל אחד מהתמונות, ומורידה את הסויימת שבים של הקובץ (שורה 109).
- אם השם ללא הסויימת תואם לאחד משמות הקבצים, אז found משתנה לtrue, והמשתנה filename מקבל את הערך של הקובץ זה (שורות 111 ו-112), ויוצא מהלולאה (break).
- אם לא, הוא מדפיס הודעה כי התמונה לא נמצאה וכי על המשתמש לנסות שוב.
- בשורה الأخيرة של הקטע קוד (119), מתווסף הpath של הקובץ (תיקית pictures).

### 5.5.5. חישוב והדפסת התוצאה למשתמש

בשלב זהה כבר יש את שם הקובץ ואת כל המידע הנדרש לחישוב, ולכן ניתן להתחיל בתהליך זה.

```

120
121     display_name, _ = os.path.splitext(filename)
122     print(f"\nProcessing {display_name}...")
123
124     print("Calculating hardware hash...")
125     hw_hash, hw_time = calculate_sha256.hardware(filepath)
126
127     print("Calculating software hash...")
128     sw_hash, sw_time = calculate_sha256.software(filepath)
129
130     filesize = os.path.getsize(filepath)
131
132     print(f"""Results:
133     {"=" * 80}
134     File: {os.path.basename(filepath)}
135     Size: {filesize:,} bytes
136
137     Hardware Hash:
138     {hw_hash}
139     Hardware Time: {hw_time:.6f} seconds
140
141     Software Hash:
142     {sw_hash}
143     Software Time: {sw_time:.6f} seconds
144
145     Speedup: {sw_time/hw_time:.2f}x
146     {"=" * 80}"""
147

```

קטע קוד 24 – חישוב וודפסת התוצאה למשתמש

על מנת ליצור ממashing נוח ונעים לעין, הודפסו כל ההודעות בקטע קוד 24 (ולא רק התוצאה והשוואת הזמן).

- בשורה הראשונה והשנייה מוצג למשתמש "... processing" כאשר ה \_\_\_ מכיל את שם הקובץ ללא הסימטת (שורה 121 מורידה את הסימטת של הקובץ).
- בשורות 124 ו-125 מודפס למשתמש כי מתחיל החישוב ו(בשורה 125) hw\_time ו hw\_hash מקבלים את תוצאות החישוב החומרתי ואת הזמן אותו לוקח לבצע את החישוב (המידע מתקיים מהפונקציה שהוגדרה והסבירה בקטע קוד 13, קטע קוד 14, קטע קוד 15, קטע קוד 16, וקטע קוד 17).
- בשורות 127 ו-128 קורא אותו דבר, אלא שכאן המידע נשלח לפונקציית החישוב בתוכנה (לצורך ההשוויה) ונשמר בה hw\_time ו sw\_time.
- לאחר מכן (שורה 130) נשמר filesize גודל התמונה בבייטים (לצורך הצגה למשתמש).
- בשורות 132 עד 146 מודפסות התוצאות למשתמש, כולל הצגה של הקטגוריה המוגדר בזמן החישוב של הCPU חלק, זמן החישוב של FPGA (פי כמה לוקח לתוכנה יותר זמן מאשר לחומרה). הקטגוריה מוצגת כמספר עשרוני עם 2 ספרות אחרי הנקודה.
- בנוסף כמובן מוצגים התוצאות של החישוב בתוכנה וכן התוצאה של החישוב בחומרה, והזמן שהלקח לכך מהרכיבים לבצע את החישוב. הזמן מוצג ביחידות של שנייה, עם שישה ספרות לאחר הנקודה העשרונית, לצורך תצוגה מדויקת של הזמן (עד מיליוןית השנייה).

### 5.5.6. השוואת התוצאה בחרומרה לתוכנה בתוכנה

```

147 if hw_hash == sw_hash:
148     print("✓ Hashes match! Hardware acceleration successful!")
149 else:
150     print("⚠ Warning: Hardware and software hashes don't match!")
151
152 print()

```

קטע קוד 25 – השוואת התוצאה שהתקבלה מה-CPU לתוכנה מה-FPGA

בשלב האחרון בפונקציית הח נערך השוואת התוצאה שהתקבלת בתוכנה לבין התוצאה שהתקבלת ע"י החומרה, על מנת לוודא שacus הטענו חישוב נכון ומדויק. בהתאם לבדיקה מוצגת הודעה למשתמש אם אכן התוצאות זהות או שונות (כਮון שההשתמש יכול לבצע השוואת זו בעצמו מאוחר והוצאות משני החישובים מוצגות לפניו), אך על מנת לחסוך בדיקה זו, ועל מנת להיות יותר נעים לשימוש, התוכנה מבצעת את הבדיקה ומציגת התוצאה למשתמש.

### 5.6. קריאה ל `main`

```

153
154 if __name__ == "__main__":
155     main()

```

קטע קוד 26 – קוד הפיטון הקורא לפונקציה `main`

זהה צורת כתיבה סטנדרטית בPython-שמבצעת את הפונקציה (`main` רק כאשר הקוד רץ ישירות).

בכל קובץ Python יש משתנה מובנה בשם `__name__`. הערך של `__name__` משתנה בהתאם לאופן שבו הקובץ מופעל. כאשר הקובץ מופעל ישירות (כמו `python script.py`) (python script.py) (python script.py)

Python קובע את `__name__` להיות `"__main__"`. במקרה זה, התנאי:  
`:=__main__ == "main"`

אך כאשר הקובץ מיובא כמודול (`import script`) הערך של `__name__` יהיה שם הקובץ ("script" למשל, אם הקובץ נקרא `py.py`). במקרה זה, התנאי לא מתקיים, ולכן `main()` לא יבוצע אוטומטית.

## 6. תוצאות

### 6.1. בדיקת המערכת וקבלת תוצאה

לאחר צירבת ה-stream, נבדקה המערכת. התבילה היא נבדקה באמצעות קוד פשוט אשר לא הכיל מידע, ורק רופד כהגדרת הריפוד, והושווה לערך המתקיים מפונקציית HASH שבספריה שהזוכה (hashlib) של פיטון. התקבל ערך נכון כפי שניתן לראות באIOR 32.

SHA-256 Result: e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855

CPU calculation e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855

AIOR 32 – קבלת תוצאה נכונה בכניסת NALL

בשלב הבא נכתבת התוכנה ושורפה על מנת לאפשר חווית משתמש נוחה ונעימה ככל הנitin. באIOR .34

```
=====
          SHA-256 Hardware Accelerator
          Calculate SHA-256 hash using FPGA acceleration
=====

Available images:
1. window
2. lake
3. PYNQ
4. tree
5. electricity
6. pynq
7. blue water
8. sunset
9. winter
10. rocks

Enter image name or number (or 'q' to quit)

Choice: 
```

AIOR 33 – דוגמא לשימוש במערכת, קבלת קלט מהמשתמש

בשלב זה כפי שהסביר בכתיבת התוכנה, ניתן לראות באIOR 34 הממשק נותן את רשימת הקבצים הנמצאים בתיקייה pictures (באיור 35 ניתן לראות רשימה זו ולהשווות שאכן כל הקבצים זוהו). מתחת לקבצים הוא נותן הודעה למשתמש להכניס שם של קובץ או מספר (האינדקס של התמונה הרצויה), או q (quit). על מנת לאפשר למשתמש לצאת מה prompt הזה ולסיטם את פעולה התוכנה.

בדוגמא הנתונה נבחר הקובץ winter. בשלב הבא מוצג למשתמש הודעות על כך כי החישוב מבוצע (על מנת לתת אינדיקציה למשתמש על הפעולה המתבצעת ברגע זה), ולאחריהם מתתקבלות התוצאות. את השלב של התוצאות ניתן לראות באIOR 34 ושם ניתן לראות את השם של הקובץ הנבחר (winter) בדוגמה זו, את כמות הביטים שיש בקובץ (בסביבות 6.7MB) וכי שניתן לראות גם באIOR 35(ו). ולאחריו את התוצאות של החומרה ושל התוכנה ואת זמני החישוב שלהם.

ניתן לראות את הפרש הזמנים המשמעותי בין חישוב החומרה לחישוב התוכנה. מוצג גם הקטועspeedup ומתקבל שיש האצה משמעותית של פי 2.63.

```

=====
SHA-256 Hardware Accelerator
Calculate SHA-256 hash using FPGA acceleration
=====

Available images:
1. window
2. lake
3. PYNQ
4. tree
5. electricity
6. pynq
7. blue water
8. sunset
9. winter
10. rocks

Enter image name or number (or 'q' to quit)
Choice: winter

Processing winter...
Calculating hardware hash...
Calculating software hash...
Results:
=====
File: winter.jpg
Size: 6,684,662 bytes

Hardware Hash:
6a6ecd3a39c4a288bef619570d581bbac4fd9e22bc828b4a24043b7bc6c9bd1b
Hardware Time: 0.069077 seconds

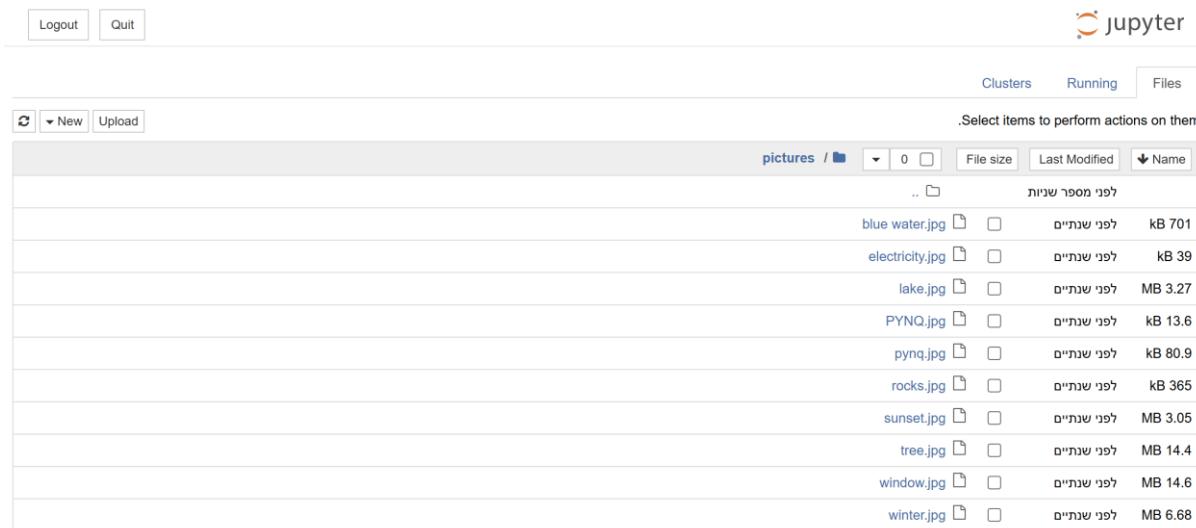
Software Hash:
6a6ecd3a39c4a288bef619570d581bbac4fd9e22bc828b4a24043b7bc6c9bd1b
Software Time: 0.181624 seconds

Speedup: 2.63x
=====
✓ Hashes match! Hardware acceleration successful!

```

איור 34 – דוגמא לשימוש במערכת, קבלת התוצאה

באיר הבא (איור 35) ניתן לראות את הצלום מסך המתעד את רשימת הקבצים הנמצאים בתיקייה **.pictures**.



איור 35 – צילום מסך של תיקייה Pictures בתוכן Jupyter והתוכלה שלה

ניתן לראות כי אכן הרשימה המתקבלת על ידי התוכנה זהה לרשימת הקבצים הנמצאים בתיקייה זו, ומכללים סימנת הנמצאת ברשימה הסימנות המותרונות (במקרה זה, כל הקבצים, לאחריהם כולם מכילים את הסימנת **gqj**).

## 6.2. בדיקת הכנסת קבצים בעלי סיווגים שונים

כפי שהואסביר בהסברים על קוד התוכנה, רק הסיווגים:

- jpg .1
- jpeg .2
- png .3
- gif .4
- bmp .5
- tiff .6
- webp .7

זוהו על ידי התוכנה ווצעו למשתמש לחישוב. באIOR 36 ניתן לראות כי הוכנסו קבצים שונים של תמונות (.jpg, .jpeg) וכן קבצים לא של תמונות (.pdf ו-.docx). הziיפייה היא לקבל אך ורק את הקבצים המסתויימים באחת הסיווגים המותרות.

| Select items to perform actions on them |                         |           |               |                |  |
|---|-------------------------|-----------|---------------|----------------|--|
|   | pictures /              | File size | Last Modified | Name           |  |
| ..                                      |                         |           |               | لפני מספר שנים |  |
|   | electricity.jpg         | kB 39     |               | لפני شנתיים    |  |
|   | Final Project Book.docx | MB 17.7   |               | لפני شנתיים    |  |
|   | Final Project Book.pdf  | MB 4.07   |               | لפני شנתיים    |  |
|   | Mountains.jpeg          | MB 6.29   |               | لפני شנתיים    |  |
|   | volcano.png             | MB 2.7    |               | لפני شנתיים    |  |
|   | winter.jpg              | MB 6.68   |               | لפני شנתיים    |  |

איור 36 – הכנסת קבצים שונים לתיקיות pictures

באIOR 37 ניתן לראות כי אכן הקבצים הנוכנים זוהו, ואילו אלה עם הסיווגת הלא נcona, לא זוהו ולא הוצאו למשתמש לחישוב (בדיוק כפי שהוגדר והתוכנה נכתבה לבצע).

```
=====
SHA-256 Hardware Accelerator
Calculate SHA-256 hash using FPGA acceleration
=====

Available images:
1. electricity
2. Mountains
3. winter
4. volcano

Enter image name or number (or 'q' to quit)

Choice: 
```

איור 37 – הצגת רשימת האירורים התואמת את איור 36

### 6.3. ביצוע החישוב על קבצים בגודלים שונים

על מנת לבדוק את היכולות האצה של המערכת, נבדקו קבצים שונים בגדלים שונים. הציפייה היא לקבל האצה ("יחס  $\frac{T_{sw}}{T_{hw}}$ ") הולכת וגדלה ככל גודל הקובץ הסיבה לכך היא שכלל שהחישוב הולך ונעשה ארוך יותר, היכולת המקובלית של החומרה תלך ותעשה שימושותית יותר. ואילו בקבצים קטנים, הציפייה היא אמם לקלט שיפור, אך פחות משימושתי. בתחילת הוכנסה מחזרות, על מנת לבדוק את האצה בקלט קצר ברמת הכמות ספורה של ביטים. בשלב זה הוכנסה המחרוזת *abc* (כמובן שלא לתוכנה שהוסבה למעלה, שמקבלת ומבצעת את החישוב על תמונות ולא על מחזרות. אלא נכתב קוד ייעודי על מנת לבדוק קלט זה). באירור 38 ניתן לראות את תוצאת החישוב.

```
Enter plaintext data: abc
SHA-256 Result: ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad
Expected Output: ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad
PL Execution Time: 0.003457 seconds
PS Execution Time: 0.000970 seconds
Speedup: 0.28
```

איור 38 – ביצוע חישוב HASH על הקלט *abc*

כפי שניתן לראות, התוצאה יוצאה כי התוכנה מהירה משימושית מהחומרה.

בשלב הבא בוצעו החישובים על קבצים בגודלים הולכים וגדלים, על מנת לעקוב אחרי הקשר בין האצה של החומרה לבין הגודל של המידע הנשלח. באירור 39 ובאיור 40 ניתן לראות את התוצאות לקבצים בגודלים של 39KB וקובץ של 81KB.

```
=====
          SHA-256 Hardware Accelerator
          Calculate SHA-256 hash using FPGA acceleration
=====

Available images:
1. window
2. lake
3. electricity
4. pynq
5. blue water
6. Mountains
7. winter
8. rocks
9. volcano

Enter image name or number (or 'q' to quit)
Choice: 3

Processing electricity...
Calculating hardware hash...
Calculating software hash...
Results:
=====
File: electricity.jpg
Size: 39,033 bytes

Hardware Hash:
c1125f934230d761f228fa69b94769a42ac801f99affed2723a1e6e0ee36283f
Hardware Time: 0.001391 seconds

Software Hash:
c1125f934230d761f228fa69b94769a42ac801f99affed2723a1e6e0ee36283f
Software Time: 0.001049 seconds

Speedup: 0.75x
=====
✓ Hashes match! Hardware acceleration successful!
```

איור 39 – תוצאת החישוב לקובץ 39KB

Results:

```
=====
File: pynq.jpg
Size: 80,907 bytes

Hardware Hash:
18b7323e818816190f978d8cf409f5efbf024b8bf9b8102476418e30aabdba3
Hardware Time: 0.001907 seconds

Software Hash:
18b7323e818816190f978d8cf409f5efbf024b8bf9b8102476418e30aabdba3
Software Time: 0.002090 seconds

Speedup: 1.10x
=====
✓ Hashes match! Hardware acceleration successful!
```

אייר 40 – תוצאות החישוב לקובץ *pynq.jpg* 81KB

בשלב הבא, הוכנסו למנוւ HASH קבצים גדולים יותר. בעליית גודל הקובץ לסדרות ה 365KB של קובץ, כבר ההאצה נעשית משמעותית ונitinן לראות כי מהירות החומרה גדולה במעט פי שטיים מחישוב התוכנה.

Results:

```
=====
File: rocks.jpg
Size: 364,852 bytes

Hardware Hash:
2a7303eeb659c606a93b391776b677853a861e0a6af412ac6607be5cf475c58
Hardware Time: 0.004668 seconds

Software Hash:
2a7303eeb659c606a93b391776b677853a861e0a6af412ac6607be5cf475c58
Software Time: 0.009069 seconds

Speedup: 1.94x
=====
✓ Hashes match! Hardware acceleration successful!
```

אייר 41 – תוצאות החישוב לקובץ *rocks.jpg* 365KB

פה כבר ההאצה משמעותית יותר. בשלב האחרון הוכנסו קבצים בגודל 701KB ו-14.6MB. באיר 42 ניתן לראות כי ההאצה כבר הגיעה ל-2.32. ערך זה הוא כבר משמעותי ומקביד, אך הוא קרוב להאצה שהתקבלה עם קובץ בגודל חצי מזה הנוכחי.

Results:

```
=====
File: blue water.jpg
Size: 701,374 bytes

Hardware Hash:
207c0eaabd36f541cd81c8fba54a40086bfe243bdbcaf97f7a1eb2c32dd3a2c4
Hardware Time: 0.008157 seconds

Software Hash:
207c0eaabd36f541cd81c8fba54a40086bfe243bdbcaf97f7a1eb2c32dd3a2c4
Software Time: 0.018904 seconds

Speedup: 2.32x
=====
✓ Hashes match! Hardware acceleration successful!
```

אייר 42 – תוצאות החישוב לקובץ *blue water.jpg* 701KB

בשלב האחרון גודל הקובץ שהוכנס קפץ ל 14.6MB על מנת לראות מה התוצאה שקבצים גדולים משמעותית יותר.

Results:

```
=====
File: window.jpg
Size: 14,620,205 bytes

Hardware Hash:
2f2e907c5a656eb151e41e9e84317d6d52f87d652c6146d92a94a59ed6f4d64d
Hardware Time: 0.149662 seconds

Software Hash:
2f2e907c5a656eb151e41e9e84317d6d52f87d652c6146d92a94a59ed6f4d64d
Software Time: 0.385297 seconds

Speedup: 2.57x
=====
✓ Hashes match! Hardware acceleration successful!
```

איור 43 – תוצאות החישוב לקובץ window.jpg 14.62MB

כפי שניתן לראות באיור 43 התוצאה של גודל קובץ של 14MB היא קרובה מאוד לתוצאה שהתקבלה בכניסת קובץ בגודל של 701KB !

## 6.4. ניתוח התוצאות מסעיף 6.3

התוצאות שהתקבלו בסעיף 6.3. לכוארה סותרות את החשיבה שמאחורי הפרויקט.

### 6.4.1 השערות שהובילו לגבי התוצאות

שלושה הסברים אפשריים שהובילו המ:

1. התוכנה מבצעת את החישוב בצורה עיליה יותר. פיתרון זה אפשרי והגיוני מכיוון שלא בוצעה חישיבה עמוקה מידי על יעילות החישוב של אלגוריתם SHA-256 בשעת כתיבת התוכנה, אלא יותר על הרכיב נכון של המערכת ותקשורת טוביה וכוננה עם DMA.
2. הבעה היא DMA שיציר מן *buttle-neck* ומונע ביצוע חישוב מהיר יותר של החומרה (התקשורת של DMA לא מהירה מספיק והמערכת צריכה להאט לקבע העברת הנתונים של DMA).
3. הPS והPL של כרטיס PYNNQ רצים עם שני שעוניים שונים.

בשלב הבא הוכנסה הבעה למנוע AI של Claude 3.7 sunnet) על מנת לראות אם הוא ימצא פיתרון הגיוני לבעה.

### 6.4.2 השערות שישפיך מנוע AI לגבי התוצאות

#### ניתוח ביצועים והשוואה בין מימוש תוכנתי (PS) למימוש חומרתי (PL)

במהלך הבדיקות נמצא כי עבור קובצי קLit קטנים (מטהות ל- 70KB), חישוב ה-Hash באמצעות המעבד (PS) היה מהיר יותר בהשוואה למימוש החומרתי (PL). עם זאת, עבור קובצי קLit גדולים יותר, ביצועי FPGA השתפרו באופן משמעותי, והחל מסף של כ-70KB, המימוש החומרתי החל להוביל בגודל קLit של 14MB, התקבלה האצה של פי. 2.6 ~ בהשוואה למימוש התוכנתי.

### ניתוח הסיבות לפער הביצועים עבור קובצי קלט קטנים:

1. **תקורה גבוהה בהעברת נתונים** – ישנה עלות קבועה בהעברת הנתונים בין המעבד (PS) ל-(PL) FPGA וחזרה, בעיקר עקב הצורך בהגדלת משק ה-DMA, הקצאת זיכרון, והעברת המידע דרך ה-Bus AXI. עבור קובצי קלט קטנים, התקורה מהוות חלק משמעותית מזמן העבודה הכללי.
2. **זמן مليוי והתרונות של Pipeline** – השימוש החומרתי מבוסס על ארכיטקטורת Pipeline, המאפשרת עיבוד מקבילי, אך נדרש זמן ראשוני למילוי הציגור (pipeline). כאשר גודל הקלט קטן, זמן זה הופך לדומיננטי.
3. **יתרון קצב השעון של ה-PS** – המעבד המרכזי (PS) פועל בתדר גבוה משמעותית (בין 650MHz ל-1GHz) בהשוואה לימוש ה-(PL) FPGA, אשר פועל בתדר של כ-100MHz כאשר נפח הנתונים קטן ואין אפשרות לנצל מקביליות רחבה, יתרון זה מקנה לשימוש התוכנת ביצועים טובים יותר.

### ניתוח היתרונות של ה-FPGA עבור קובצי קלט גדולים:

1. **צמצום השפעת התקורה** – ככל שגודל הקלט גדול, השפעת התקורה הקיימת פוחתת, ורביית זמן החישוב מנוצל לעיבוד בפועל.
2. **יעול ארכיטקטוני** – הנטיב החומרתי מותאם במיוחד לחישוב ה-Hash, תוך שימוש בחומרה ייעודית המאפשרת פעולות חישוביות ספציפיות.
3. **ניצול אפקטיבי של Pipeline** – לאחר שלב המילוי הראשוני, ה Pipeline מאפשר עיבוד רציף של המידע בקצב של מילה אחת (או יותר) לכל מחזור שעון, מה שמקנה שיפור משמעותי ביצועים.

התוצאות תואמות את הציפיות לימושי חומרה ותוכנה. עבור קובצי קלט קטנים, התקורה והשעון המהיר של ה-PS מknנים לו יתרון ביצועי. עם זאת, עבור קובצי קלט גדולים, בהם ניתן לנצל את המקביליות של ה-FPGA השימוש החומרתי מספק שיפור משמעותי ביצועים. ההאצה של פי 2.6 שהתקבלה בגודל קובץ של 14MB תואמת את הספורות המקצועית, לפיה ניתן לצפות להאצה בטוויה של פי 2 עד פי 10 עבור שימושי Hashing בחומרה.

#### 6.4.3. בדיקת השערת השעונים הנפרדים

ההשערה הראשונה שנבדקה היא ההשערה לגבי השעונים הנפרדים העובדים ל-CPU ו-FPGA. על מנת לבדוק השערה זו, נבדקו מספר אטרים הקשורים בנושא, ובאחד מאטרים אלו (포ורום באתר של PYNN<sup>3</sup>) נמצא קוד Python הנutan (דרך ספרייה של PYNN) את התדרים של השעונים השונים הנמצאים בכרטיס.

באיר 44 ניתן לראות את התוצאה שהתקבלה בהרצת הקוד המופיע באותו פורום. התוצאה מוכיחה כי ההשערה שהעלתה הייתה נכונה ולכן ניתן לומר שהיא תامة לציפייה, לשעון נוספת לפחות לאירועים נוספים סיבות אחרות אפשריות. אך ייחוס זה של שעונים של 6.5 זהו ייחוס לא מבוטל המסביר בצורה טובה את ההאצה היחסית נמוכה.

---

<sup>3</sup> <https://discuss.pyNN.io/t/frequency-of-operation-of-pyNN-z2/2834/3>

```

1 from pynq import Clocks
2
3 print(f'CPU: {Clocks.cpu_mhz:.6f}MHz')
4 print(f'FCLK0: {Clocks.fclk0_mhz:.6f}MHz')
5 print(f'FCLK1: {Clocks.fclk1_mhz:.6f}MHz')
6 print(f'FCLK2: {Clocks.fclk2_mhz:.6f}MHz')
7 print(f'FCLK3: {Clocks.fclk3_mhz:.6f}MHz')

CPU: 650.000000MHz
FCLK0: 100.000000MHz
FCLK1: 100.000000MHz
FCLK2: 100.000000MHz
FCLK3: 100.000000MHz

```

איור 44 – בדיקת מהירות שעון CPU מול שעון FPGA

כפי שניתן לראות, כל שורה מדפיסה את התדר (בדיקה של עד 6 ספרות לאחר הנקודה העשרונית), ביחידות של MHz.

#### 6.4.4. בדיקת ההשערה לגבי ה - DMA

על מנת לבדוק את הגורם לחוסר הלינאריות בהאצה של החומרה מול התוכנה (בדלים קטנים בתוכנה נראית מהירה יותר – ביל' להתחשב בהבדלים בין התדרים של UPC-FPGA מול FPGA) ובגדלים גדולים יותר של מידע ההאצה הולכת וגדלה עד שmag'עה ל"תקרת זכוכית" של בסביבות 2.6 האצה), נדרש רצונות רבות. מספר נסיבות הרואו שהמערכת מביאה תוצאות שונות בזמן, אף על פי שהוא מידע בדיק נשלח בכל פעם. הגורמים לכך הם כנראה שינויים בטמפרטורה ודברים פנימיים הקשורים ל-CPU ו-FPGA.

מה שכן נשאר יחסית קבוע זה הטוווח של התוצאה. אך על מנת לבנות גרפים אמינים ככל האפשר של ההאצה של התוכנה כתלות בגודל המידע (גרף שההיגיון הפשט היה אומר שהוא לינארי, מכיוון שככל שהמידע גדול יותר, ישנו יותר בלוקים. וכל בלוק הוא עוד זמן שהוא FPGA יכול לנצל לחישוב מקבילי מול הCPU).

לען על מנת לשפר את הגرافים כל הودעה (תמונה/טקסט) נשלח ב50Hz, וה ממוצע של 50 הזרחות הוא המידע שנכנס לבניית הgraf.

נכתב קוד תוכנה במיוחד לשלב זה, אשר יוצר קבצי txt בגודלים שונים (, 10, 100, 1000, 10000, 100000, 1000000, 2000000, 4000000, 8000000, 16000000, 23000000 Byte) על מנת ליצור גраф אחד ומין.

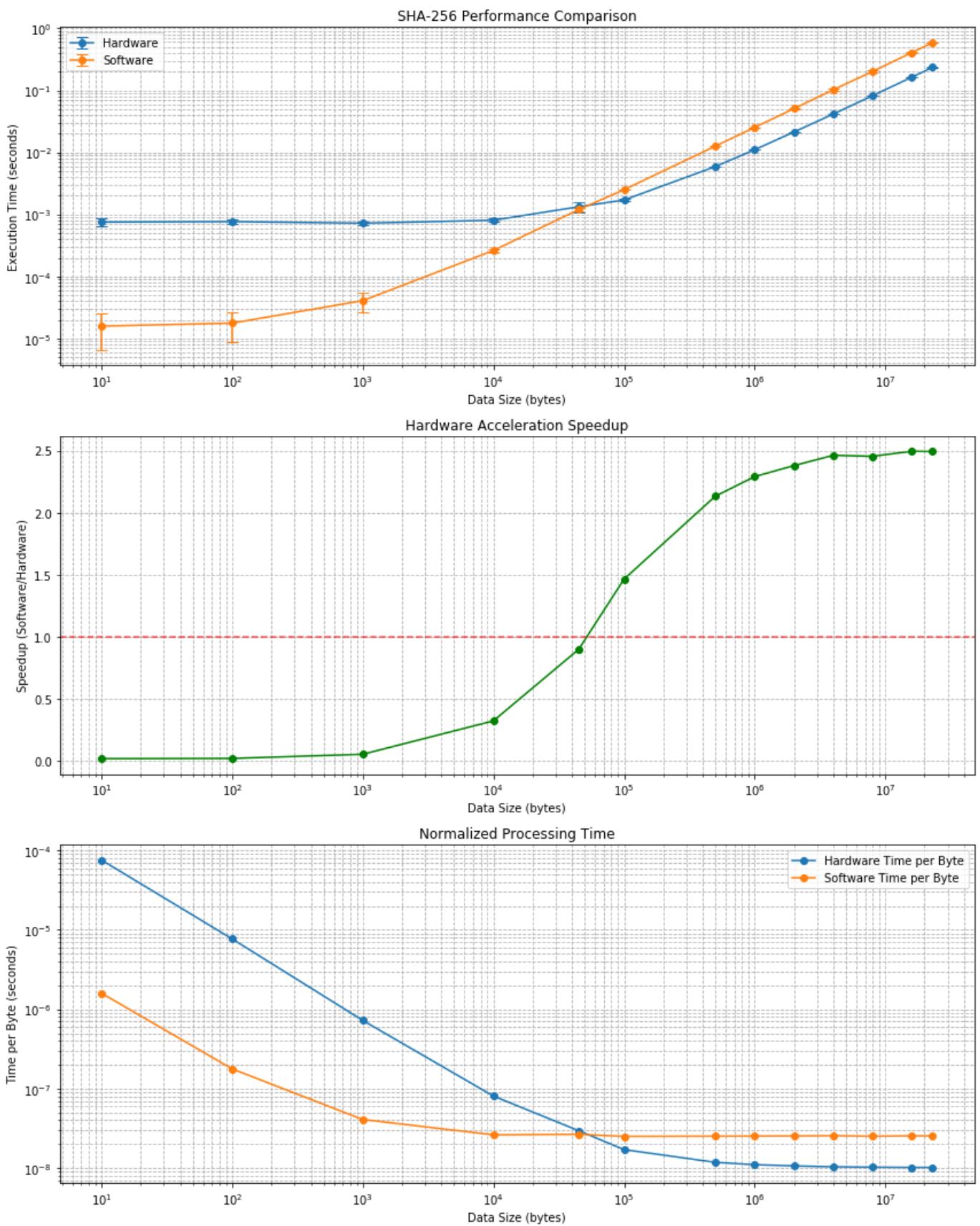
בקבלת הנתונים הוכנו שלושה גרפים שונים, כאשר:

**בgraf הראשון:** ציר ה-X (האפקן) מייצג את גודל הקובץ (בבתים), בסקאלה לוגריתמית. ציר ה-Y (האנכי) מייצג את הזמן הכלול לחישוב ה- SHA-256 (בשניות), גם כן בסקאלה לוגריתמית.

**בgraf השני:** שוב, ציר ה-X הוא גודל הקובץ בbytes (בסקאלה לוגריתמית). ציר ה-Y הוא ה-speedup שהוא היחס  $T_{sw}/T_{hw}$  זמן התוכנה חלקי זמן החומרה.

**בgraf השלישי:** ה-X הוא גודל הקובץ (סקała לוגריתמית). ציר ה-Y הוא זמן לחישוב המחולק בגודל הקובץ – כלומר, "זמן פר ביט (Time per Byte)" גם הוא בסקאלה לוגריתמית.

את שלושת הגרפים ניתן לראות באיר 45.



אייר 45 – שלושת הגרפים המשמשים לנתח התוצאות

### נקודות מהגרף הראשון

- עבר קבצים קטנים, זמן החומרה גדול מזמן התוכנה. זה קורה כי יש "overhead" לא מבוטל של הגדרת DMA, העברת הנתונים אל ומם ה-FPGA, והפעלת החומרה.
- ככל שגודל הקובץ גדול, הזמן הכלול של החומרה גדול בצורה מתונה יותר (כי overhead מתרחש על כמות גדולה יותר של נתונים), ולעומת זאת זמן התוכנה גדול מהר יותר באופן ייחודי.
- לכן, במקרה מסוימת (crossover) החומרה מתחילה להיות מהירה יותר מהתוכנה, והקוו הכהן נעשה נmor יותר מהקו הכתום.

### נקודות מהגרף השני

- בהתאם, עבר גגלי קבצים קטנים, ה-speedup נמור מ-1 (החומרה איטית יותר כי overhead מורגש מאוד).
- במקרה מסוימת (شمוגגת בקו האנכי האדום), הגרף חוצה את הקו 1, ומאותה נקודה ואילך speedup הופך לגדול מ-1 (החומרה משתלמת יותר).
- ככל שהקבצים גדלים מעבר לנקודת זו, היחס הולך וגדל – ככל יותר החומרה נהיה "משתלמת" יותר ויותר ביחס לחישוב התוכנה.

### נקודות מהגרף השלישי

- הגרף הזה מדגיש איך overhead משפיע באופן ייחודי לכל בית.
- עבר קבצים קטנים, רואים שערך הזמן לבית עבר החומרה די גבוה כי overhead הקבוע (הגדרת DMA העברת נתונים וכו') מתחלк על מעט נתונים.
- כשהקבצים גדלים, הזמן לבית עבר החומרה יורך בצורה דרמטית (ה"overhead" מתחלק על יותר בתים), וכך בסופו של דבר הוא נעשה נmor יותר מהתוכנה.
- בתוכנה, לעומת זאת, הזמן לבית יכול להיות יותר יציב (אם כי גם שם יש השפעה מסוימת של גודל הקובץ על ניצול זיכרון/מטרון), אבל לרוב לא צונח באותה דרמטיות.

### 6.4.5. מסקנות מהניתוח

משלשת הגרפים ניתנים לראות כי אכן ההשערה שהעלתה לגבי הזמן מעבר של המידע מה-PS אל ה-PL ובחזרה הוא ממשמעותיו והוא הגורם העיקרי לשינוי הדramtic בין ההאצה של מידע קצר להאצה של מידע ארוך (זמן מעבר "גבילע" בזמן החישוב). וכך על מנת לראות את ההאצה האמיתית של החומרה, נדרש להשתמש בקבצים של מעל 45KB (באIOR 45 נראה כי זה הוא המספר). מתחת לכך המעבר של המידע דרך DMA ("בולע") את היתרונו ומהירותו של המימוש החומרתי. על מנת לבדוק את רמת הסוף הזו בוצעה בדיקה על קובץ בגודל של 45KB ב-1000000 חזרות (לבדיקה אמינה ומדויקת). באIOR 46 ניתן לראות את המוצא על קלט זה.

בנוסף ההשערה לגבי הימצאות שעוניים שונים לחומרה (FPGA) ולתוכנה (CPU), התבררה כנוכנה, וכן גם בהאצה הנמוכה ביותר (מחוזת הייצאת (לאחר RIPOD) באורך של בלוק אחד), התקבלה תוצאה האצה הגדולה מ-1 (אם מנורמלים לפי היחס בין השעוניים של שני הרכיבים).

### ניתוח מספרי

ניתוח של המספרים שניתן לראות בגרף השלישי, מראה כי התקבלה תוצאה בדיקת מצופה, כאשר הזמן של העברת הנתונים דרך DMA נהייה זניח.

אם מתמקדים בחלק הימני של הגרף השלישי באIOR 45 (גודל קובץ הגadol מ 500KB בערך), ניתן לראות את הזמן חישוב לבית בודד מתייצב (גם בחומרה וגם בתוכנה). בחומרה הזמן זהה הוא בערך  $sec^{-8} \cdot 10^{-10}$  ואילו בתוכנה הזמן הוא בערך  $sec^{-8} \cdot 10^{-10} \cdot 2.5$ .

בדיקות מהירה של כמה זמן לוקח החישוב של בלוק בודד (64 בתים) תביא תוצאה של:  $sec^{-8} \cdot 10^{-10} \cdot 64$  בחומרה ותוצאה של  $sec^{-8} \cdot 10^{-10} \cdot 160$  בתוכנה (ב-CPUs).

נורמל לפি מספר מחזוריים של השעון תגלה את יכולות האמיטיות של כל אחד מהרכיבים (VS CPU ו-FPGA). נכפול בתדר ונקבל שהחישוב דרך ה-CPUs **קרה ב-1040 מחזוריים של השעון בערך, ואילו לא-FPGA לך בערך 64 מחזוריים של השעון.**

התוצאה יוצאה בבדיקה מצופה כפי שניתן לראות בסימולציה באIOR 48 עם הוספה של 16 מחזוריים של קבלת המידע מהמשתמש

### סיכום

יתכן מאוד (וסביר להניח) כי קיימים עוד סיבות וגורמים למוגבלה על החומרה, כאשר הצפי היה לקבל האצה משמעותית מאוד. אך כבר שני הסברים אלו מניחים את הדעת והם מסבירים גם את חוסר הלינאריות (בזה אשם ה-DMA וזמן העברת הנתונים בין הצדדים (overhead)), והשעונים השונים (מסביר את התוצאות הנמוכות המתתקבלות בהאצה).

```
Benchmarking 45000 bytes test file (45.00 KB) with 100000 iterations...
Calculating hardware hash...
Calculating software hash...
Results:
=====
File: test_45000_bytes.txt
Size: 45,000 bytes (45.00 KB)

Hardware Hash:
c9f55ae79ad95b0543244315bf305b9750e3cfbc6cde44673299c1e443dddef
Hardware Mean Time: 0.001142 seconds ± 0.000126

Software Hash:
c9f55ae79ad95b0543244315bf305b9750e3cfbc6cde44673299c1e443dddef
Software Mean Time: 0.001138 seconds ± 0.000164

Speedup: 1.00x
=====
✓ Hashes match! Hardware acceleration successful!
```

איור 46 – תוצאות החומרה בקלט בגודל 45KB

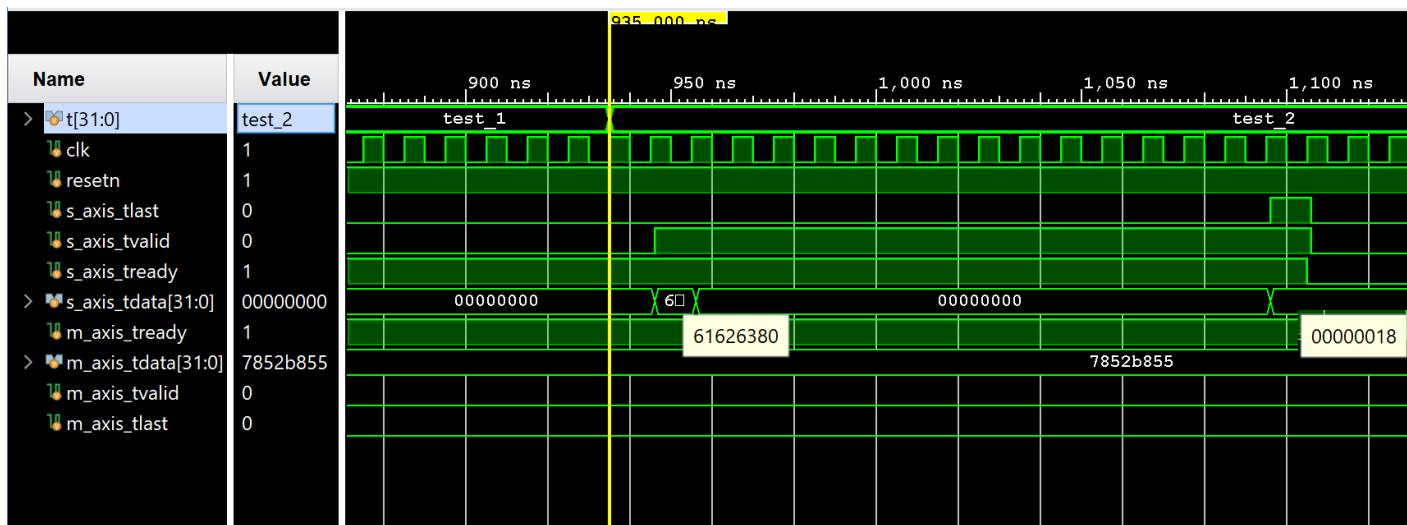
כפי שניתן לראות, בבדיקה עם חזרה של 100000 פעמים איששה את התוצאה שהתקבלה בגרף שacky ערך הסף הוא 45KB. כמובן שלאחריו יש עלייה הדרגתית כפי שניתן לראות באIOR 45, ההאצה הולכת ועולה. אך ערך זה הוא ערך הסף כפי שניתן לראות.

## 7. סימולציה

על מנת לבדוק את החומרה שנכתבה, בצורה נכונה, נכתבת תוכנית test-bench אשר מכילה בדיקה של הכנסות הודעות שונות למערכת – במקרה של DMA בפרוטוקול AXI, ונבדקה תגובת המערכת לכניות האותיות (הנקראות test-bench) בא "לחיקות" את התנהגות DMA).

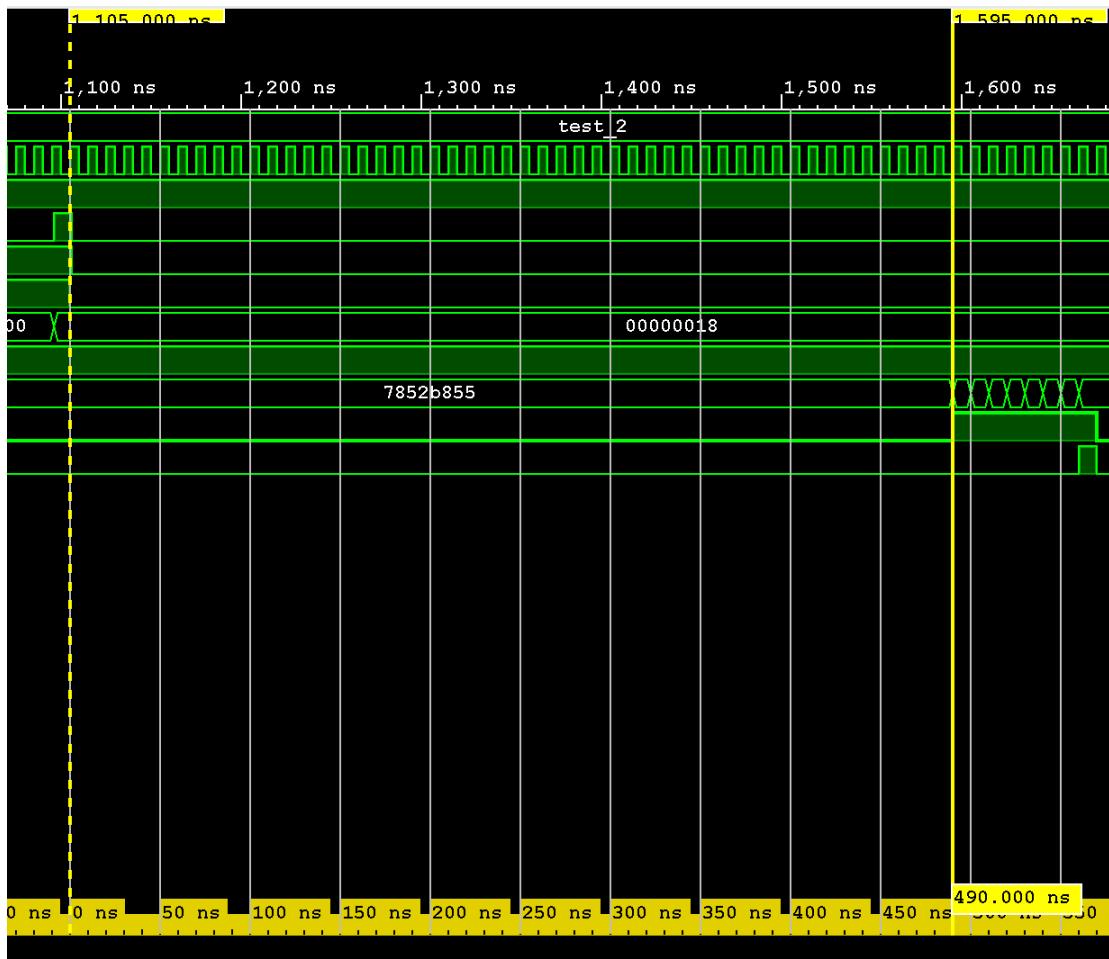
### 7.1. בדיקת המחרוזת "abc"

בדיקת test\_2 היא הכנסת המחרוזת "abc" אשר בתצוגת hexadecimal מקודד "616263". הריפור מוסיף את הביט '1' בסוף מחרוזת זו, ואורך ההודעה הוא 24 ביט (4 ביט לכל ספרה, עם 6 ספרות). הקידוד hexadecimal של 24 זה 18. לכן המידע BUS הכניסה צריך להיות 16 בלוקים של 32 ביט כל אחד, אשר הראשון הוא: 0x61626380, כל השאר חוץ מהאחרון הם אפסים, והבלוק האחרון יהיה 0x00000018. יחד עם הבלוק מידע האחרון, הסיגנל s\_axis\_tlast צריך לעלות. באIOR 47 ניתן לראות תהליך זה שתואר כאן של שליחת המידע.



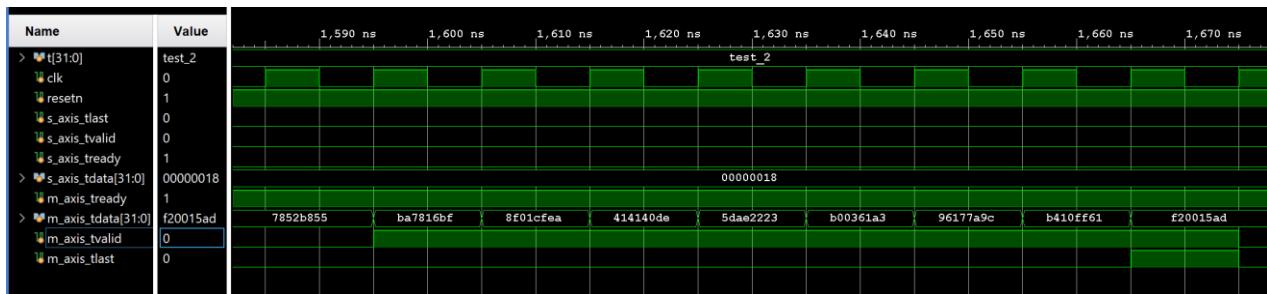
איור 47 –Waveform המציג את שליחת ההודעה "abc" במנגנון AXI

אלו הם 16 סיבבים הראשונים. לאחריהם המידע ממשיר לעבור עיבוד, וכעבור 48 סיבובים (כל סיבוב לוקח.clk אחד), ועוד סיבוב לחישוב הסכימה של המცבים ההתחלתיים עם המשתנים הפעילים (ז"א כעבור 49 clockים של השעון) המידע יהיה מוכן BUS המוצא. באIOR 48 ניתן לראות את זמן החישוב. מכיוון שבסימולציה זו הוצגו רק הקווים הנקנסים אל ה-DUT והקווים היוצאים ממנו (קווי master ו-slaves), לא ניתן לראות את השינויים שהתרחשו בשלב זה. אבל כפי שהואסביר בפרק התאורטי ב-1.2.1, בשלב זה קורים אותם 48 סיבובים.



איור 48 – חישוב התוצאה לוקח בדיק 49 clocks של השעון לאחר עליית `s_axis_tlast`

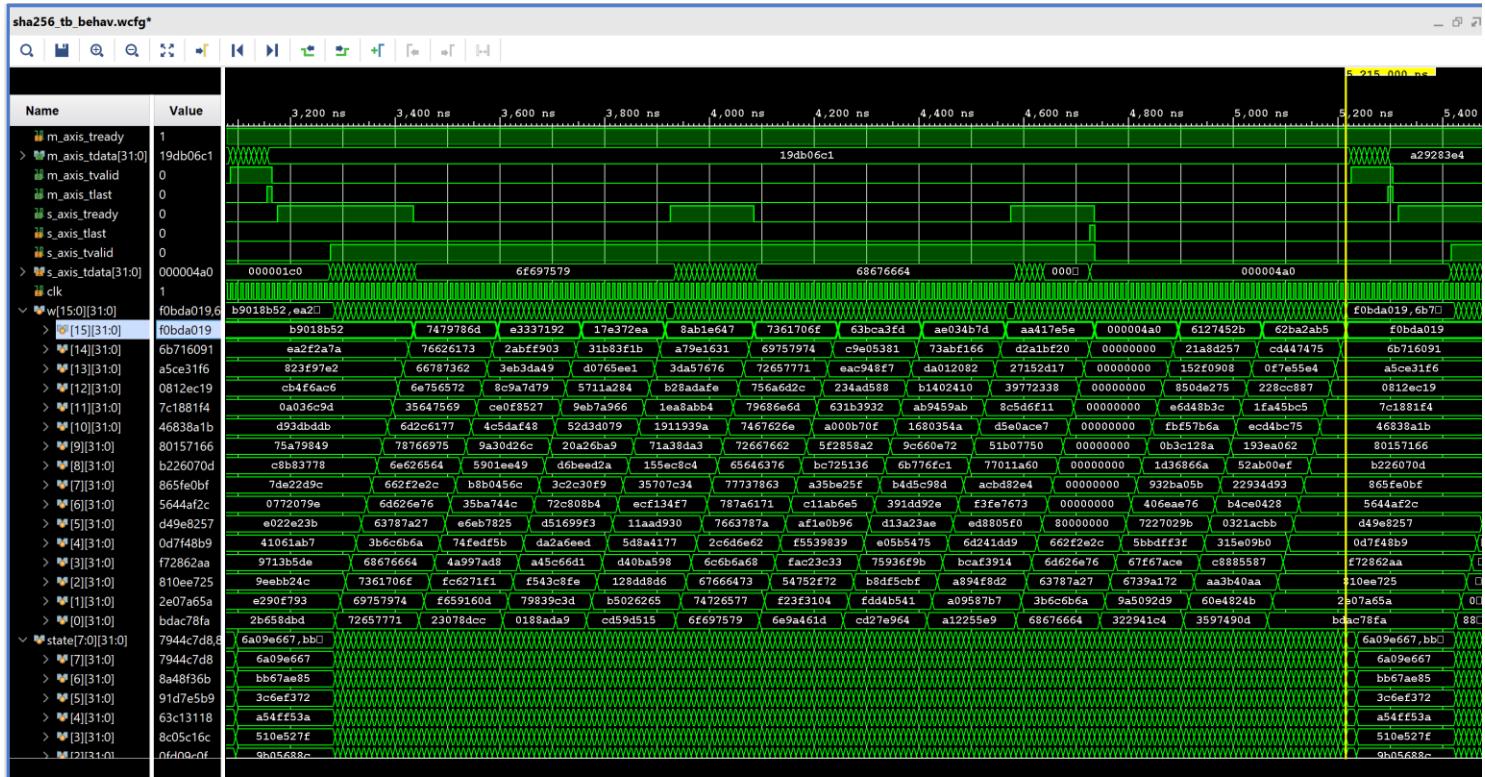
לאחר השלמת החישוב, המידע מוכן במצואו והסיגנל `m_axis_tvalid` עולה ביחד עם המידע במצואו. מכיוון שהטבות (slave-batches) שלוט סיגナル של `test-batch`, המידע ישר מתחלף וועובר לפיסת הbhאשה של המידע. אם הסיגナル `s_axis_tready` מכוון ל-0 בעליית `m_axis_tvalid`, המידע ממשיר לחכotta במצואו ולא מתחלף עד לעליית `m_axis_tready`. תהליך זה של שליחת המידע של HASH בחזור DMA מתואר באיור 49.



איור 49 – שליחת תוצאה HASH על המידע בחזור למשתמש

ניתן לראות את המידע מועבר וועובר בהצלחה בחזרה DMA. ארבעת הסיגנלים התחתונים באיור 49 שייכים לפעולות השליחה בחזרה של תוצאה HASH. ניתן לעקוב אחרי האותיות

## 7.2. הציגת שינוי מערך לוח הזמנים W בסימולציה



איור 50 – סימולציה המציגת את השינוי במערך לוח הזמנים W

בחלק עליון של איור 50 ניתן לראות את הסיגנלים של פרוטוקול AXI לצורך התקשרות עם DMA. הסימולציה מדגימה שליחה של הודעה בגודל 1184 ביט (ההודעה אינה בעלת משמעות). ההודעה היא (בבקסאיד צימלי):

```
72657771_69757974_7361706F_68676664_3B6C6B6A_63787A27_6D626E76_
662F2E2C_6E626564_78766975_6D2C6177_35647569_6E756572_66787362_
76626173_7479786D_6F697579_74726577_67666473_6C6B6A68_2C6D6E62_
7663787A_787A6171_77737863_65646376_72667662_7467626E_79686E6D_
756A6D2C_72657771_69757974_7361706F_68676664_3B6C6B6A_63787A27_
6D626E76_662F2E2C
```

לצורך נוחות הקריאה ההודעה הובאה מחלוקת חלקים של 32 ביט כל אחד (8 ספרות בבקסא).

ניתן לראות כי קצר לפני 3,200ns המרכיב סימנה חישוב קודם, ו-s\_axis\_tready עולה לסמן מוכנות לתחילת פעולה חדשה.

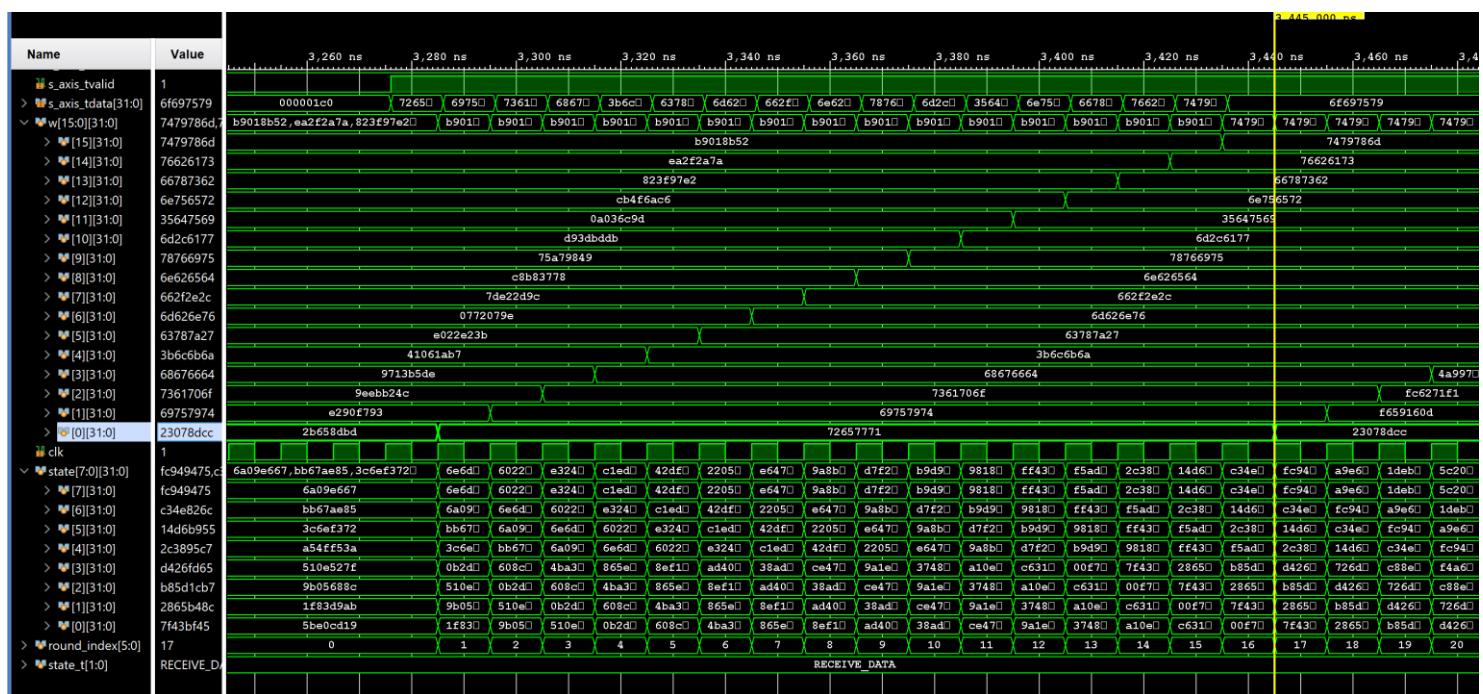
קצת לפני 3,300ns הסיגנל s\_axis\_tvalid עולה והוכנס מידע לא-s\_axis\_tdata. מכיוון שקיים s\_axis\_tready נמצא למעלה (ב'1), המידע מתקבל ומוחלף בשעון הבא. כך המידע מתחלף כל clk של clk של השעון עד לקבלת 16 המילימטר הראשונים של ההודעה. בקבלה המילה ה-16 הסיגנל s\_axis\_tready יורד, מפני שכעת המערכת לא מוכנה לקבל מידע חדש, אלא המידע הנוכחי עובר עיבוד.

בשלב הבא יש 48 מילימטרים שמתקבלים מהמודולו expansion. בזמן 3,450ns ניתן לראות את המילה הראשונה שמתבלט מהexpansion (ב - W[0]).

בסיום 48 הסיבובים של הרחבה של ההודעה, קורית הסכימה של סוף בлок מידע (ה initial state ) הינה עם המשתנים הפעילים בסוף הסיבוב. ניתן לראות כי בזמן זה לא משתנה לוח הזמנים W, אך רגיסטורי המצב (state) כן משתנים. ניתן לראות זאת כ"חור" בשורה המציגת את המערך W.

לאחר מכן מוקם לעלה ומוקן לקבל מידע חדש. כתת ה banch (או במצבים זה יהיה DMA) רואה זאת וממשיך לשולח את המידע (שורת ה s\_axis\_tdata ). אותו תהליך קורה גם לבlok השני וגם השלישי. ניתן לראות בסיום הבלוק השלישי את הסיגנל tlast\_s\_axis\_tdata שמסמן כי נשלח המידע האחרון. בשלב זה המכונת מצבים עוברת למצב FINISH\_CALC. הבלוק האחרון מחושב, ויצא ל DMA בקווים master (ארבעת הסיגנלים העליונים באירור 50).

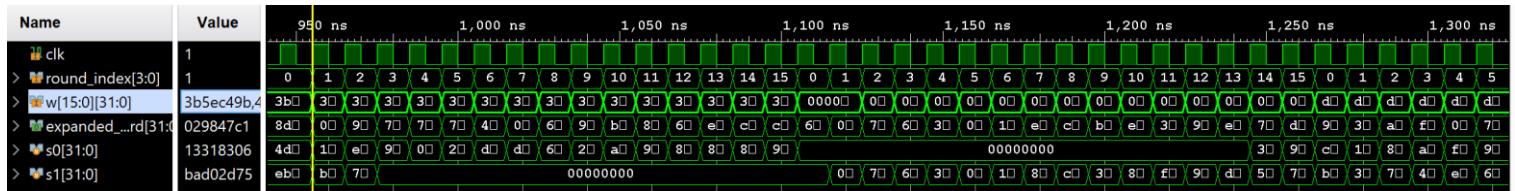
### 7.3. הציגת שינוי ה State register



איור 51 – הציגת שינוי רגיסטורי state

באיור 51 ניתן לראות את תחילת הפעולה מצד שמאל, בה ניתן לראות את ה initial\_state (כפי שנitin לראות ב 1.2.1 (מימוש האלגוריתם)). בהמשך ניתן לראות בכל clk של השעון שינוי ב state registers (ניתן גם לראות כי כעבור 16 clocks של השעון, המידע במערך לוח הזמנים W נדרס על ידי מידע חדש). סיגנל ה round\_index סופר את המידע המתkeletal ואת הסיבובים המחשבבים.

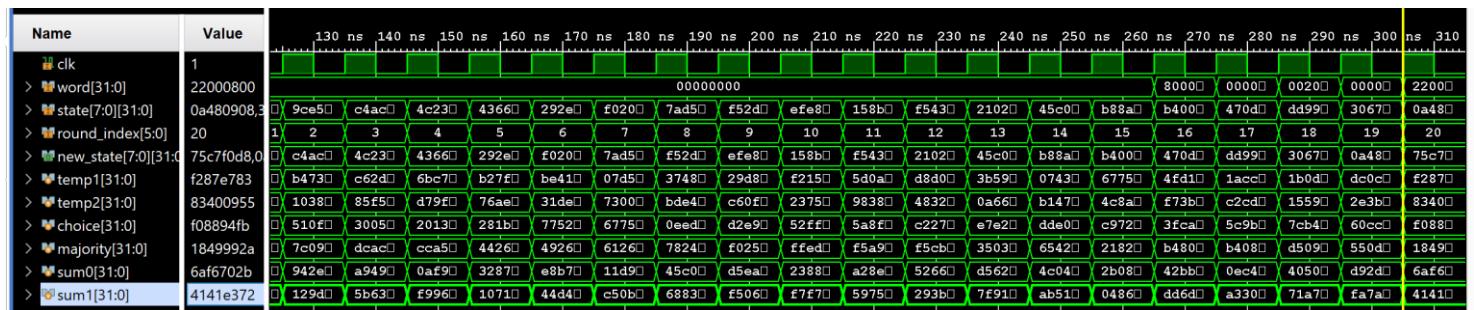
## 7.4. חישוב ה expanded word



איור 52 – חישוב המילה הבאה במספר לוח הזמן

המימוש של קטע קוד 2 מובא באיור 52. ניתן לראות כי בכל `clk` המידע בקווים משתנה. הבלוק הינו קומבינטורי לחלווטין, וכל הסיבה לכך שניתן לראות את השינוי בקצב השעון, זה מכיוון שהסיגנלים שנכנסים כסדרם למודולו הם סינכרוניים (השינוי בהם סינכרוני ולכן השינוי במצב הבלוק הקומבינטורי הינו "סינכרוני"). הבלוק מממש את: נוסחה 1, נוסחה 2 ונוסחה 3.

## 7.5. חישוב המצב הבא בכל round



איור 53 – חישוב המעתנים הפעילים של `round` הבא

בכל `clock` של השעון משתמשים הפעילים לפי נוסחות: נוסחה 4, נוסחה 5, נוסחה 6, נוסחה 7, נוסחה 8, נוסחה 9 ונוסחה 10. ניתן לראות את כל המשתמשים המופיעים בנוסחות אלו. גם פה הבלוק הינו בלוק קומבינטורי לחלווטין, והשינוי הלאוראה סינכרוני נובע רק מההשתנות הסינכרונית של הכניסות לבlok זה.

## 7.6. סיכום של הסימולציות

הסימולציות שהוצגו בפרק זה מסותות רק את המצב הcy פשוט בו DMA מוכן לשילוח המידע תמיד, והמערכת מוכנה על פי יכולת שלה. מקרים שונים, בהם עיקוב של כמה `clocks` של השעון בין כל קבלת מידע של המערכת, וכג"ל לגבי השילוח של התוצאה (`m_axis_tready`) לא נמצאת למעלה קבוע). בוצעו גם הם, ונבחן כי המערכת אכן יציבה ופועלת כמורה.

## 8. מסקנות הפרויקט

### סיכום ומסקנות הפרויקט

#### סיכום הפרויקט

הפרויקט עוסק בפיתוח והטמעה של רכיב IP ייעודי למימוש פונקציית הגיבוב SHA-256 על גבי FPGA. המימוש בוצע באמצעות סביבת סביבת Vivado, תוך שילוב הרכיב בתצורת Block Design, שבו חוברו מודולים קבועים של DMA ומעבד (PS) ZYNQ. לאחר תהליך יצירת Bitstream וקובץ HWH, הקוד נתען על גבי ה-FPGA והופעל דרך פלטפורמת Jupyter Notebook באמצעות Python.

המערכת נבנתה כך שהקוד בפייטון אחראי על ריפוד ההודעה בהתאם לדרישות SHA-256, שליחתה לעיבוד באמצעות DMA, קליטת התוצאה מה-FPGA והשוואה מול חישוב זהה שהתבצע באמצעות ספריית תוכנה בפייטון. מדידות זמן החישוב בשני המקרים אפשרו להשוות את ביצוע ה-FPGA לביוזי ה-CPU וליחס את יחס ההאצה של החומרה ביחס לתוכנה.

#### תוצאות וניתוח

התוצאות שהתקבלו הצביעו על מגמה מעניינת:

1. **לגודל הودעה קטן (עד עשרות קילו בייטים), החומרה הייתה איטית יותר – ככלומר, החישוב ב-CPU היה מהיר יותר בהשוואה ל-FPGA.** הדבר התרברר בכך עד גודל הודעה של 45KB.
2. **ככל שגודל הודעה גדול, ההאצה של ה-FPGA עולה –** כאשר בנפחים גדולים (1MB ומעלה) ההאצה שהושגה הייתה פי 2.4-2.6 בהשוואה ל-CPU.
3. **לאורך כל המדידות, כאשר מנורמלים את זמן החישוב לפי תדר השעון, החומרה תמיד הראתה ביצועים עדיפים על התוכנה –** גם במקרה הגרוע ביותר, כאשר יחס ההאצה היה 0.28 ככלומר, החומרה נראתה איטית פי 3.57 מהתוכנה, בהתחשב בכך שה-CPU רץ בתדר גובה פי 6.5 מה-FPGA, יצא שככל מחזור שעון בודד, החומרה ביצעה את החישוב בצורה מהירה יותר.

ניתוח נוסף העלה מספר גורמים מרכזיים שהופיעו על התוצאות:

- **קצב השעון של החומרה –** שעון ה-FPGA פועל בתדר של 500MHz בעודשה-CPU פועל בתדר של 650MHz ככלומר מהיר בערך פי 6.5. כאשר מנורמלים לפי השעון, הביצועים של ה-FPGA היו טובים יותר בכל מקרה.
- **תקורה בתחלת תהליכי החישוב –** תהליכי ההכנה של המידע ושליחתו ל-FPGA באמצעות DMA מօסיף עיכוב שאיןו קיים בתוכנה, במיוחד מדובר בתנאים קצריים.
- **היתרון של החומרה באידי ביוטי רק במקרים גדולים –** החישוב ב-FPGA הופך לעיל יותר כאשר יש כמות מסוימת של נתונים שמאפשרת למקביליות של החומרה להשתלם על פני הביצוע הסדרתי של CPU. מה שנוצר לאור העובדה שהמעבר של הנתונים בין החומרה לתוכנה (לצורך החישוב וקבלת התוצאה מהחומרה), לוקח זמן משמעותי ביחס לזמן החישוב להודעות קצרות.

## מסקנות עיקריות

1. החומרה מציגה יתרון ברור על פני התוכנה בכל שלבי החישוב, אך התקורה התחילה גורמת לכך שرك עבר גודל נתונים מסוימים היתרון הופך להיות משמעותי.
2. נרמול הביצועים לפי שעון מראה שהחומרה תמיד מהירה יותר מהתוכנה פרט מחרוז שעון – ככלומר, אם ה-FPGA היה פועל בתדר דומה לזה של ה-CPU ההאצה שהייתה מתقبلת הייתה גדולה בהרבה.
3. השפעת תקורת ה-DMA – התקורה של שליחת הנתונים ל-FPGA והחזרתם לתוכנה משפיעה על זמן החישוב הכלול, במיוחד כאשר מעבדים הודיעות קצרות.

### שיפור אפשרי של הביצועים :

- **הגברת תדר השעון של ה-FPGA** כדי לצמצם את הפער מול ה-CPU.
- **הקטנת התקורה של ה-DMA** ע"י אופטימיזציה של פעולת DMA.
- **ኒצול מקבליות גבוהה יותר** כדי לאפשר עיבוד של מספר הודעות במקביל ולמকסם את התפקה.

## סיכום

הפרויקט סיפק תובנות שימושיות בנוגע להאצת חישובי SHA-256 באמצעות חומרה. התוצאות הוכיחו כי החומרה תמיד ביצעה את החישוב מהר יותר מהתוכנה פרט מחרוז שעון, אך היתרון הכלול שלה תלוי בגודל הקולט ובתקורה של המערכת.

בפועל, כאשר מדובר בהודעות קטנות, התקורה עלולה להפוך את התוכנה למהירה יותר במידה ישירה, אך כאשר גודל הנתונים גדול, האצה של ה-FPGA נעשית שימושית. הפרויקט הדגיש את החישיבות של תכנון נכון של חומרה בשילוב עם תקשורת גבוהה, תוך איזון בין התקורה, תדר עבודה וኒצול מקבליות כדי למקסם את הביצועים.

## 9. קודים מלאים

### 9.1. קודי חומרה

#### המכיל קבועים ופונקציות package Module .9.1.1

```

`timescale 1ns / 1ps
package sha256_pkg;

parameter logic [31:0] standard_initial_state[7:0] =
{32'h6a09e667, 32'hbb67ae85,
32'h3c6ef372, 32'ha54ff53a,
32'h510e527f, 32'h9b05688c,
32'h1f83d9ab, 32'h5be0cd19};

parameter logic [31:0] k[64] =
{32'h428a2f98, 32'h71374491, 32'hb5c0fbcf, 32'he9b5dba5,
32'h3956c25b, 32'h59f111f1, 32'h923f82a4, 32'hab1c5ed5,
32'hd807aa98, 32'h12835b01, 32'h243185be, 32'h550c7dc3,
32'h72be5d74, 32'h80deb1fe, 32'h9bdc06a7, 32'hc19bf174,
32'he49b69c1, 32'hefbe4786, 32'h0fc19dc6, 32'h240ca1cc,
32'h2de92c6f, 32'h4a7484aa, 32'h5cb0a9dc, 32'h76f988da,
32'h983e5152, 32'ha831c66d, 32'hb00327c8, 32'hbf597fc7,
32'hc6e00bf3, 32'hd5a79147, 32'h06ca6351, 32'h14292967,
32'h27b70a85, 32'h2e1b2138, 32'h4d2c6dfc, 32'h53380d13,
32'h650a7354, 32'h766a0abb, 32'h81c2c92e, 32'h92722c85,
32'ha2bfe8a1, 32'ha81a664b, 32'hc24b8b70, 32'hc76c51a3,
32'hd192e819, 32'hd6990624, 32'hf40e3585, 32'h106aa070,
32'h19a4c116, 32'h1e376c08, 32'h2748774c, 32'h34b0bcb5,
32'h391c0cb3, 32'h4ed8aa4a, 32'h5b9cca4f, 32'h682e6ff3,
32'h748f82ee, 32'h78a5636f, 32'h84c87814, 32'h8cc70208,
32'h90beffffa, 32'ha4506ceb, 32'hbef9a3f7, 32'hc67178f2};

//right_rotate
function automatic logic [31:0] right_rotate(input [31:0] x, input [4:0] n);
    return x >> n | x << (32 - n);
endfunction

endpackage

```

咎ט קיד – 27 המכיל את הקבועים והפונקציות

## rounds Module .9.1.2

```

`timescale 1ns / 1ps
import sha256_pkg::right_rotate;
import sha256_pkg::k;

module sha256_round(input [31:0] word,
                      input [31:0] state [7:0],
                      input [5:0] round_index,
                      output logic [31:0] new_state [7:0]);

    logic [32-1:0] temp1 ,temp2, choice, majority, sum0, sum1;

/*  state[7]=a      state[6]=b      state[5]=c      state[4]=d
   state[3]=e      state[2]=f      state[1]=g      state[0]=h */

    assign choice = (state[3] & state[2]) ^ (~state[3] & state[1]);
    assign majority = (state[7] & state[6]) ^ (state[7] & state[5]) ^
                      (state[6] & state[5]);
    assign sum0 = right_rotate(state[7], 5'd2) ^ right_rotate(state[7], 5'd13) ^
                  right_rotate(state[7], 5'd22);
    assign sum1 = right_rotate(state[3], 5'd6) ^ right_rotate(state[3], 5'd11) ^
                  right_rotate(state[3], 5'd25);
    assign temp1 = state[0] + sum1 + choice + k[round_index] + word;
    assign temp2 = sum0 + majority;

    assign new_state[6:4] = state[7:5]; //assigning b,c,d (to be previous-state's a,b,c)
    assign new_state[2:0] = state[3:1]; //assigning f,g,h (to be previous-state's e,f,g)
    assign new_state[7] = temp1 + temp2;
    assign new_state[3] = temp1 + state[4];

endmodule

```

קטע קיד 28 – module קומבינטורי האחראי על חישוב המשתנים הפעילים הבאים

### Module .9.1.3 החשבון של לוח הזמנים W

```

`timescale 1ns / 1ps
import sha256_pkg::right_rotate;

module sha256_expansion ( input [3:0] round_index,
                           input [31:0] w[15:0],
                           output logic [31:0] expanded_word);

    logic [32-1:0] s0,s1;

    assign s0 = right_rotate(w[(round_index-15)%16], 5'd7) ^
               right_rotate(w[(round_index-15)%16], 5'd18) ^
               (w[(round_index-15)%16] >> 2'h3);

    assign s1 = right_rotate(w[(round_index-2)%16], 5'd17) ^
               right_rotate(w[(round_index-2)%16], 5'd19) ^
               (w[(round_index-2)%16] >> 4'ha);

    assign expanded_word = (w[round_index] + s0 + w[(round_index-7)%16] + s1);

endmodule

```

קטע קוד 29 האחראי על חישוב המילה הבאה W (מיומוש נסחה 3)

## Main ה Module .9.1.4

```

`timescale 1ns / 1ps
import sha256_pkg::standard_initial_state;

module sha256_main( input clk,
                     input resetn,
                     input s_axis_tlast,
                     input s_axis_tvalid,
                     input [31:0] s_axis_tdata,
                     input m_axis_tready,
                     output logic [31:0] m_axis_tdata,
                     output logic s_axis_tready = 0,
                     output logic m_axis_tvalid = 0,
                     output logic m_axis_tlast = 0);

logic [31:0] word, expanded_word, w[15:0];
logic [31:0] state[7:0];
logic [31:0] new_state[7:0];
logic [31:0] initial_state[7:0];
logic [5:0] round_index = '0;
logic chunk_end = 0;
logic [31:0] sha[7:0];
logic [2:0] result_ptr = 0;
enum logic [1:0] {RECEIVE_DATA, FINISH_CALC, SEND_RESULT} state_t = RECEIVE_DATA;

//instantiations
// Round calculation
sha256_round round (
.word(word),
.state(state),
.round_index(round_index),
.new_state(new_state) );

// Message schedule expansion
sha256_expansion expand (
.round_index(round_index[3:0]),
.w(w),
.expanded_word(expanded_word) );

always_ff @(posedge clk or negedge resetn) begin
  if (!resetn) begin
    state_t <= RECEIVE_DATA;
    result_ptr <= 0;
    s_axis_tready <= 0;
    m_axis_tvalid <= 0;
    m_axis_tlast <= 0;
    chunk_end <= 0;
    round_index <= '0;
    initial_state <= standard_initial_state;
    state <= standard_initial_state;
  end else begin
    case (state_t)
      RECEIVE_DATA: begin
        s_axis_tready <= round_index[5:4] == 2'b00 && round_index != 15;
        if (s_axis_tvalid&&s_axis_tready||chunk_end||round_index[5:4]!=2'h0) begin
          if (chunk_end) begin
            foreach (state[i]) begin
              state[i] = expanded_word;
            end
          end
        end
      end
    endcase
  end
end

```

```

        initial_state[i] <= initial_state[i] + state[i];
        state[i] <= initial_state[i] + state[i];
    end
    chunk_end <= 1'b0;
end else begin
    w[round_index[3:0]] <= word;
    state <= new_state;
    round_index <= round_index + 6'b1;
    if (round_index[5:4] == 2'b00) begin
        if (s_axis_tlast) begin
            state_t <= FINISH_CALC;
            s_axis_tready <= 0;
        end
        chunk_end <= 1'b0;
    end else if (round_index == 6'd3) chunk_end <= 1'b1;
    end
end
FINISH_CALC: begin
    if (round_index != 6'd0) begin
        w[round_index[3:0]] <= word;
        state <= new_state;
        round_index <= round_index + 6'b1;
    end else begin
        foreach (sha[i]) sha[i] <= initial_state[i] + state[i];
        m_axis_tdata <= initial_state[7] + state[7];
        m_axis_tvalid <= 1;
        state_t <= SEND_RESULT;
        result_ptr <= 7;
        m_axis_tlast <= 0;
    end
end
SEND_RESULT: begin
    if (m_axis_tvalid && m_axis_tready) begin
        if (result_ptr == 3'd0) begin
            m_axis_tvalid <= 0;
            m_axis_tlast <= 0;
            state_t <= RECEIVE_DATA;
        end else begin
            result_ptr <= result_ptr - 1;
            m_axis_tdata <= sha[result_ptr - 1];
            m_axis_tlast <= (result_ptr - 1 == 3'd0);
            initial_state <= standard_initial_state;
            state <= standard_initial_state;
        end
    end
end
endcase
end
end
assign word = round_index[5:4]== 2'b00 ? s_axis_tdata : expanded_word;
endmodule

```

קיטוע תייר של חומרה (floorplan) main ו module – 30%

## Test Bench 9.1.5

```

`timescale 1ns / 1ns
module sha256_tb;
/* test cases:
test 1 - NULL
test 2 - abc
test 3 - abcdabcdecdefdefgefghfghighijhijklmklmnlnomnnopnopq
test 4 -
rewqiuytsapohgfd;lkjcxz'mbnvf/,nbedxvium,aw5duinuerfxsbvbastyxmoiuytrewgfds1kj,h,mnbvcxzxaqws
xcedcvrvbvtgbnyhnmujm,rewqiuytsapohgfd;lkjcxz'mbnvf/,,
test 5 -
88866d5a04c2b81f579962b7293928a6a2458381ef4f022fc2ec7a72422b275e0f5588e36c63f371a4ddd72d89308a
6d1a41e5edced3f805720eceaa64f09d21b1059ff90b5b1f5e9ff7f3374da3ded1d47eb9f6562d0bff48974c0234e5b
e5fa1f571c984c5d4dc8edb13d4ffffc20b5009578b782b7f6b029d1b1b4c7726ef8870d1b9130d967e6b170352e3c
1d04579ad3f1207efca01c6d0d4416329e118a66fb0974bac9a026e5511650a4a2a1c10264cf24c38d0c66f3cf3102
e2c5be3e69ffd24fcf964f94dcc329543f8be0b67d3ec91547a61ce928ff1e2d1072c1ab9b499dda9a37847747d001
1f6457f03dafdce9e23ea6bbbf4371eae2b4278f914ac099c428bdac62d9e0e28a8c97df9942a26bb9323b199787e
c001a4fc7d8cda3db0928947ddaa9c73dd1a5a496d6e86adaca8ef5b071bda3269f9a31629d8
test 6 -
d364e8f1545ce324431f92858db5d670dbb90c597149fd94402fbef07d04a3f76e5604c98102eec5adb391582c6758
b85ddd03f53b1696b125c71235cf692dd45f260dd4fe1e19759544655511310ce88581166caa512601073ddceaa9a0
d3608952ecd51bf2a12ed18ad3d8a246c2098d97d8dc762483c49ce8e1ccb4c7ff8721b765046af02a3b44fa8a4ffb
474e3c8dfc121c7a4fcf5cf597b269b8465ed838be2884645a504f251846bd82e8ccdcc7f4296b6995d44fd2b36343
22c119a11abdcff594756536f1d217d65dfcc6e48dfe4976865425f17f95f9b420368ea99df22598c33f49b0a9f669
485e5661682d698fc973c0e1b4627d53fe417e82be13243d29ef5c950f56cb298cedbfffac5899ca76c4e785cf68346
8eb897aca16e0438df074093b0e177e94d707ebece79fe133407a7f48756c5d112f3de2ff50e
test 7 - ... */
logic clk = 0;
logic resetn;
logic start_i;
logic s_axis_tlast;
logic s_axis_tvalid;
logic [31:0] s_axis_tdata;
logic m_axis_tready;
logic [31:0] m_axis_tdata;
logic s_axis_tready, m_axis_tvalid;
logic m_axis_tlast;
logic delayed;
logic [255:0]compression;
logic [63:0] data[130];
logic [63:0] key[16];
typedef enum {test_1,test_2,test_3,test_4,test_5,test_6,test_7} test;
test t;
string test_str, input_massage;

sha256_main uut (.clk(clk),
.resetn(resetn),
.s_axis_tlast(s_axis_tlast),
.s_axis_tvalid(s_axis_tvalid),
.s_axis_tdata(s_axis_tdata),
.m_axis_tready(m_axis_tready),
.m_axis_tdata(m_axis_tdata),
.s_axis_tready(s_axis_tready),
.m_axis_tvalid(m_axis_tvalid),
.m_axis_tlast(m_axis_tlast) );
initial forever #5 clk = ~clk;

task automatic SendBlockData (input [3:0]block_amount);
int unsigned j = 0;
do begin
@(posedge clk) begin #1;
start_i = j == 0;
s_axis_tvalid = 1'b1;
s_axis_tdata = data[j];
end
end
endtask

```

```

        if (j == block_amount*16-1) s_axis_tlast <= 1'b1;
        else s_axis_tlast <= 1'b0;
    end
    if (s_axis_tvalid && s_axis_tready) j++;
end while (j < block_amount*16);
@(posedge clk) begin #1
    s_axis_tlast <= 1'b0;
    s_axis_tvalid <= 1'b0;
    j = 0;
end
wait (m_axis_tvalid);
$display(test_str, "input message: %h", input_message);
$display("Hash Output: ");
do @(posedge clk); while (!m_axis_tready&&m_axis_tvalid); $display("%h", m_axis_tdata);
$display("----expected: %h", compression);
#100;
endtask

task automatic test_vec_1();
test_str = "test 1-"; t = test_1;
input_message = "NULL";
data = '{default: '0};
data[0] = 32'h80000000;
compression = 256'he3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855;
endtask

task automatic test_vec_2();
test_str = "test 2-"; t = test_2;
input_message = "abc";
data = '{default: '0};
data[0] = 32'h61626380;
data[15] = 32'h00000018;
compression = 256'hba7816bf8f01cfeca414140de5dae2223b00361a396177a9cb410ff61f20015ad;
#1; endtask

task automatic test_vec_3();
test_str = "test 3-"; t = test_3;
input_message = "abcdabcdecdefdefgefghfghighijhijklmklmnlnomnopnopq";
data = '{default: '0};
data[0:14] = {'h61626364, 'h62636465, 'h63646566, 'h64656667,
              'h65666768, 'h66676869, 'h6768696A, 'h68696A6B,
              'h696A6B6C, 'h6A6B6C6D, 'h6B6C6D6E, 'h6C6D6E6F,
              'h6D6E6F70, 'h6E6F7071, 'h80000000};
data[31] = 'h000001c0;
compression = 256'h248d6a61d20638b8e5c026930c3e6039a33ce45964ff2167f6ecedd419db06c1;
#1; endtask

task automatic test_vec_4();
test_str = "test 4-"; t = test_4;
input_message
="rewqiuytsapohgfd;lkjcxz'mbnvf/.,nbedxvium,aw5duinuerfxsbvbastyxmoiuytrewgfdslkjh,mnbvcxxzaq
wsxedcvrfvbtgbnyhnmujm,rewqiuytsapohgfd;lkjcxz'mbnvf/,.,";
data = '{default: '0};
data[0:37] = {'h72657771,      'h69757974,      'h7361706f,      'h68676664,
              'h3b6c6b6a,      'h63787a27,      'h6d626e76,      'h662f2e2c,
              'h6e626564,      'h78766975,      'h6d2c6177,      'h35647569,
              'h6e756572,      'h66787362,      'h76626173,      'h7479786d,
              'h6f697579,      'h74726577,      'h67666473,      'h6c6b6a68,

```

```

'h2c6d6e62,      'h7663787a,      'h787a6171,      'h77737863,
'h65646376,      'h72667662,      'h7467626e,      'h79686e6d,
'h756a6d2c,      'h72657771,      'h69757974,      'h7361706f,
'h68676664,      'h3b6c6b6a,      'h63787a27,      'h6d626e76,
'h662f2e2c,      'h80000000};

data[47] = 'h000004a0;
compression = 256'h26d707651fcfe0bcef4e732e3a77a89b312aab802dfdee98fc038036a29283e4;
#1; endtask

task automatic test_vec_5();
  test_str = "test 5-"; t = test_5;
  input_message
= "88866d5a04c2b81f579962b7293928a6a2458381ef4f022fc2ec7a72422b275e0f5588e36c63f371a4ddd72d8930
8a6d1a41e5edced3f805720eceaf09d21b1059ff90b5b1f5e9ff7f3374da3ded1d47eb9f6562d0bff48974c0234e
5be5fa1f571c984c5d4dc8edb13d4fffc20b5009578b782b7f6b029d1b1b4c7726ef8870d1b9130d967e6b170352e
3c1d04579ad3f1207efca01c6d0d4416329e118a66fb0974bac9a026e5511650a4a2a1c10264cf24c38d0c66f3cf31
02e2c5be3e69ffd24fcf964f94dcc329543f8be0b67d3ec91547a61ce928ff1e2d1072c1ab9b499dda9a37847747d0
011f6457f03dafdce9e23ea6bbbfb4371eae2b4278f914ac099c428bdac62d9e0e28a8c97fdf9942a26bb9323b19978
7ec001a4fc7d8cd3db0928947ddaa9c73dd1a5a496d6e86adaca8ef5b071bda3269f9a31629d8";
  data = '{default: '0};
  data[0:80] = {'h88866d5a, 'h04c2b81f, 'h579962b7, 'h293928a6,
    'ha2458381, 'hef4f022f, 'hc2ec7a72, 'h422b275e,
    'h0f5588e3, 'h6c63f371, 'ha4ddd72d, 'h89308a6d,
    'h1a41e5ed, 'hc3d3f805, 'h720eceaf6, 'h4f09d21b,
    'h1059ff90, 'hb5b1f5e9, 'hff7f3374, 'hda3ded1d,
    'h47eb9f65, 'h62d0bff4, 'h8974c023, 'h4e5be5fa,
    'h1f571c98, 'h4c5d4dc8, 'hedbd13d4, 'hfffc20b5,
    'h009578b7, 'h82b7f6b0, 'h29d1b1b4, 'hc7726ef8,
    'h870d1b91, 'h30d967e6, 'hb170352e, 'h3c1d0457,
    'h9ad3f120, 'h7efca01c, 'h6d0d4416, 'h329e118a,
    'h66fb0974, 'hbac9a026, 'he5511650, 'ha4a2a1c1,
    'h0264cf24, 'hc38d0c66, 'hf3cf3102, 'he2c5be3e,
    'h69ffd24f, 'hcf964f94, 'hdcc32954, 'h3f8be0b6,
    'h7d3ec915, 'h47a61ce9, 'h28ff1e2d, 'h1072c1ab,
    'h9b499dda, 'h9a378477, 'h47d0011f, 'h6457f03d,
    'hafdc9e2, 'h3ea6bbbf, 'h4371eae2, 'hb4278f91,
    'h4ac099c4, 'h28bdac62, 'hd9e0e28a, 'h8c97fdf9,
    'h942a26bb, 'h9323b199, 'h787ec001, 'ha4fc7d8c,
    'hda3db092, 'h8947ddaa, 'h9c73dd1a, 'h5a496d6e,
    'h86adaca8, 'hef5b071b, 'hda3269f9, 'ha31629d8,
    'h80000000};

data[95] = 'h00000a00;
compression = 256'h826d606f511f043501117d7ae6cf2e797aaddeb86ecd8be7f59335f32ccb0ac3;
#1; endtask

task automatic test_vec_6();
  test_str = "test 6-"; t = test_6;
  input_message
= "d364e8f1545ce324431f92858db5d670dbb90c597149fd94402fbef07d04a3f76e5604c98102eec5adb391582c67
58b85ddd03f53b1696b125c71235cf692dd45f260dd4fe1e19759544655511310ce88581166caa512601073ddceaa9
a0d3608952ecd51bf2a12ed18ad3d8a246c2098d97d8dc762483c49ce8e1ccb4c7ff8721b765046af02a3b44fa8a4f
fb474e3c8dfc121c7a4fcf5cfb597b269b8465ed838be2884645a504f251846bd82e8ccdc7f4296b6995d44fd2b363
4322c119a11abdcff594756536f1d217d65dfcc6e48dfe4976865425f17f95f9b420368ea99df22598c33f49b0a9f6
69485e5661682d698fc973c0e1b4627d53fe417e82be13243d29ef5c950f56cb298cedbfaf5899ca76c4e785cf683
468eb897aca16e0438df074093b0e177e94d707ebece79fe133407a7f48756c5d112f3de2ff50e";
  data = '{default: '0};
  data[0:80] = {'hd364e8f1, 'h545ce324, 'h431f9285, 'h8db5d670,
    'hd8b90c59, 'h7149fd94, 'h402fbef0, 'h7d04a3f7,
    'h6e5604c9, 'h8102eec5, 'hadb39158, 'h2c6758b8,
    'h5ddd03f5, 'h3b1696b1, 'h25c71235, 'hcf692dd4,
    'h5f260dd4, 'hfe1e1975, 'h95446555, 'h11310ce8,
    'h8581166c, 'haa512601, 'h073ddcea, 'ha9a0d360,
    'h8952ecd5, 'h1bf2a12e, 'hd18ad3d8, 'ha246c209,
    'h8d97d8dc, 'h762483c4, 'h9ce8e1cc, 'hb4c7ff87,
    'h21b76504, 'h6af02a3b, 'h44fa8a4f, 'hfb474e3c,

```

```

'h8dfc121c, 'h7a4fcf5c, 'hf597b269, 'hb8465ed8,
'h38be2884, 'h645a504f, 'h251846bd, 'h82e8ccdc,
'hc7f4296b, 'h6995d44f, 'hd2b36343, 'h22c119a1,
'h1abdcff5, 'h94756536, 'hf1d217d6, 'h5dfcc6e4,
'h8dfe4976, 'h865425f1, 'h7f95f9b4, 'h20368ea9,
'h9df22598, 'hc33f49b0, 'ha9f66948, 'h5e566168,
'h2d698fc9, 'h73c0e1b4, 'h627d53fe, 'h417e82be,
'h13243d29, 'hef5c950f, 'h56cb298c, 'hedbffac5,
'h899ca76c, 'h4e785cf6, 'h83468eb8, 'h97aca16e,
'h0438df07, 'h4093b0e1, 'h77e94d70, 'h7ebece79,
'hfe133407, 'ha7f48756, 'hc5d112f3, 'hde2ff50e,
'h80000000};

data[95] = 'h00000a00;
compression = 256'he23e005624bcc730f352a672e6ddd1500ea787eccd386d71485c7e1c953fb898;
#1; endtask

task automatic test_vec_7();
test_str = "test 7-"; t= test_7;
input_message = "...";
data[0:127] = {'h48705e99, 'h44da44b4, 'hc8138f30, 'h81a001d2, 'hf28acc16, 'h7456b2c7,
    'h7b366bcc, 'h1c814745, 'hcc6114d7, 'h6bcc4243, 'h81b5453e, 'hc7833b7d,
    'he9c2cf0f, 'h8a185def, 'h887b57da, 'h82a2e75e, 'h98b9d534, 'h7259accf,
    'h84b71835, 'h901c930e, 'h6c9540fa, 'h05eb1728, 'ha4d54325, 'h706dc8f1,
    'h35e16958, 'h636bc97f, 'h5a83d0be, 'he45350ad, 'h036be982, 'heeb62a9a,
    'h56acf855, 'hedca3427, 'h293f98ec, 'h45e9e27c, 'h8a7f3580, 'hcc10bc84,
    'h37ac904d, 'h4c61dc6c, 'h26693f13, 'h02a17128, 'hfe3d2a98, 'h21ba3fee,
    'h9c150e24, 'h4f3b1db8, 'h8fc0fe03, 'h27d376c7, 'h9c72de07, 'hb6dac81b,
    'h3b42e603, 'hfec828fc, 'h18ac2551, 'h76bcbc74, 'h940ffb4d, 'h87f63fd2,
    'h41309dd8, 'h115d8128, 'h69b595d6, 'he4fae5ce, 'h87ab40b0, 'h487be1de,
    'hcc024fd3, 'h75b181b6, 'h9c0b462e, 'h1b12e479, 'h0f00ffb8, 'hdc38bd8d,
    'h717c28e8, 'hcd60d41e, 'hc1ffc13c, 'hf8ec829a, 'h366c9fb3, 'hdde23933,
    'hb58aec52, 'hec6afa61, 'h5c86a5a2, 'h0d54dc5d, 'hfb6f1a30, 'h198f8ca7,
    'h95b9720d, 'h5eb3c781, 'he4bb244c, 'h1f78e271, 'hd2317f86, 'h113d42c3,
    'h01bb2259, 'hdf83e504, 'h435c85e4, 'ha66fd789, 'h22fb6e4d, 'h7a4bbe6,
    'h5ae57d3b, 'he0054f77, 'h5a49a15c, 'h52aed5db, 'hc8bba0e1, 'hbefcb173,
    'hfab294fd, 'h07c9d5e7, 'h7d0a686a, 'h829248a2, 'h91835fe4, 'h7af9a651,
    'haa408dab, 'had449d68, 'hf1485c05, 'hcddb719d, 'hc10808ba, 'h00ce786b,
    'h2bd02397, 'h604c9e90, 'h735113eb, 'h7b2de390, 'h91258f0d, 'hd46ad830,
    'hb14825d4, 'h619f26fb, 'hb297b71d, 'h0b9007c0, 'haa58f949, 'h31a1ca12,
    'h61b2f5bc, 'h255c53c7, 'hd230875f, 'hadef9979, 'hae400000, 'h00000000,
    'h00000000, 'h00000f89 };

compression = 256'h0637697e16c69aaaf54c92a0a4338ba4318235f7ca65937fafaf39e5387e06784b;
#1; endtask

task automatic running_tests();
test_vec_1(); SendBlockData(1);
test_vec_2(); SendBlockData(1);
test_vec_3(); SendBlockData(2);
test_vec_4(); SendBlockData(3);
test_vec_5(); SendBlockData(6);
test_vec_6(); SendBlockData(6);
test_vec_7(); SendBlockData(8);
endtask

initial begin
m_axis_tready <= 1;
resetn <= 0; #100
resetn <= 1;
running_tests();
end
endmodule

```

הקוד יצא אורך, אך בכל זאת יסביר בצורה כללית המבנה שלו וצורת הפעולה שלו.

שורה 1 מתארת את סקאלת הזמן. בשפת systemverilog המשמעות של שורה זו היא שבכל מקום שיופיע ב-test-bench הסימן # ולאחריו מספר, הסימן delay הוא והיחידות הם מה שמתיואר בשורה זו (ns).

שורה 2 מגדרה את כוורת המודול (שורה חובה בכל מודולו ב-systemverilog) (שורת 2-3 מתארות את ערכי ה"הודעות" שנכנסו למערכת לחישוב HASH). השורות הבאות מתארות את הסיגנלים (שיתחברו לכניות ולמוצאי ה-DUT).

לאחר מכן עושים instantiation למודולו שעובר את ה-TB (ה-DUT שלנו). לאחר מכן יש את הקוד שרצ באופן קבוע, ואחראי על פועלות השעון של המערכת.

לאחר מכן יש את task (כמו פונקציה בʃפוט תכונות אחרות) האחראי על שליחת המידע בתצורת DMA לדעת. אופן הפעולה זה שיש את המידע השמור במערך data של 80 גjisטרים של 32 ביט כל אחד. הcounter בשם l עובר על האינדקסים של המערך ושולח אותם אחד לדעת (כאשר ההתקדמות לאינדקס הבא מתבצע רק כאשר גם s\_axis\_tvalid וגם s\_axis\_tdata). ישנה גם פונקציית דילוי המחקה את התנהגות DMA במצב בו הוא לא מוכן תמיד למידע, ויש שלבים בהם s יורד. הלולאה רצה על כל הבלוקים של המידע task מקבל את כמות הבלוקים של המידע המוכנס). בשורה 58 ניתן לראות כי הסיגנל tcl-tlast עולה יחד עם כניסה המיליה האחרון. בשורות האחרונות שם מודפס ל-evalConsole התוצאה של חישוב SHA-256 ובשורה מתחת לתוצאה של אותו task הופנקצת חישוב בפייטון (לצורך השוואה).

שורות הבאות בערך מילוט מספר tasks אשר כל אחד מהם קובע ערך שונהpdata (מערך הדאטה שיישלח לחישוב), ובהתאם, ל compression אשר יהוא בבדיקה לערך שיתקבל test\_vec\_1(), test\_vec\_2(), test\_vec\_3(), test\_vec\_4(), test\_vec\_5(), test\_vec\_6(), test\_vec\_7().

לאחר מכן, ב task automatic running\_tests(); זהו task המתחילה את הדאטה לערכים הבדיקה הרצiosa (בדיקות 1-7) ע"י קרייה לפונקציה המתחילה, ולאחר מכן שולחת את הפונקציה לתוך ה-DUT על ידי קרייה task של SendBlockData השולחת את המידע כפי שתואר בשורות 47-93. הפונקציה מקבלת את כמות הבלוקים של המידע, וזה הספרות המוכנסות בתוך הסוגרים(task זה).

ב7 השורות האחרונות של הקוד, מבצעים איפוס(resetn) למערכת לפני תחילת הפעולה להזיה, קובעים כי הערך m מוקן לקבל מידע, וקוראים לפונקציה המתחילה ושולחת את המידע לבדיקה (running\_tests).

השורה الأخيرة סגרת את המודולו של ה-test-bench.

## 9.2. קוד התוכנה

### 9.2.1. קוד הפיתון של הפרויקט

```

from pynq import Overlay
from pynq.lib.dma import DMA
from pynq import allocate
import numpy as np
import struct
import time
from hashlib import sha256
import os

def calculate_sha256.hardware(filepath: str) -> tuple:

    def sha256_pad(message: bytes) -> np.ndarray:
        length = len(message) * 8
        message += b'\x80'
        while (len(message) % 64) != 56:
            message += b'\x00'
        message += struct.pack('>Q', length)
        return np.frombuffer(message, dtype='>u4')

    overlay = Overlay("SHA256_BlockDesign_wrapper.bit")
    dma = overlay.axi_dma_0

    with open(filepath, "rb") as f:
        data = f.read()
    input_data = sha256_pad(data)

    data_buffer = allocate(shape=(len(input_data),), dtype=np.uint32)
    output_buffer = allocate(shape=(8,), dtype=np.uint32)
    np.copyto(data_buffer, input_data)

    start_time = time.time()
    dma.sendchannel.transfer(data_buffer)
    dma.sendchannel.wait()
    dma.recvchannel.transfer(output_buffer)
    dma.recvchannel.wait()
    execution_time = time.time() - start_time

    sha256_result = ''.join(f'{word:08x}' for word in output_buffer.tolist())
    return sha256_result, execution_time

def calculate_sha256.software(filepath: str) -> tuple:

    with open(filepath, "rb") as f:
        data = f.read()

    start_time = time.time()
    sha256_result = sha256(data).hexdigest()
    execution_time = time.time() - start_time

    return sha256_result, execution_time

def list_available_images():
    pictures_dir = "pictures"
    if not os.path.exists(pictures_dir):

```

```

print(f"\nError: '{pictures_dir}' folder not found!")
return []

image_extensions = ['.jpg', '.jpeg', '.png', '.gif', '.bmp', '.tiff', '.webp']
all_files = os.listdir(pictures_dir)
images = []
for file in all_files:
    file_lower = file.lower()
    for ext in image_extensions:
        if file_lower.endswith(ext):
            images.append(file)
            break
if not images:
    print(f"\nNo image files found in '{pictures_dir}' folder!")
    return []

print("\nAvailable images:")
for i, img in enumerate(images, 1):
    filename, extension = os.path.splitext(img)
    print(f"{i}. {filename}")
return images

def main():
    print("=" * 80,
          " SHA-256 Hardware Accelerator",
          " Calculate SHA-256 hash using FPGA acceleration",
          "=" * 80, sep="\n")

    while True:
        images = list_available_images()
        if not images:
            return

        print("\nEnter image name or number (or 'q' to quit)")
        choice = input("Choice: ").strip()

        if choice.lower() == 'q':
            print("\nExiting program...")
            return

        if choice.isdigit():
            idx = int(choice) - 1
            if 0 <= idx < len(images):
                filename = images[idx]
            else:
                print("\nError: Invalid number! Please try again.")
                continue
        else:
            found = False
            if choice in images:
                filename = choice
                found = True
            else:
                for img in images:
                    name_without_ext, _ = os.path.splitext(img)
                    if name_without_ext == choice:
                        filename = img
                        found = True

```

```

        break
if not found:
    print("\nError: Image not found! Please try again.")
    continue

filepath = os.path.join("pictures", filename)
break

display_name, _ = os.path.splitext(filename)
print(f"\nProcessing {display_name}...")

print("Calculating hardware hash...")
hw_hash, hw_time = calculate_sha256.hardware(filepath)

print("Calculating software hash...")
sw_hash, sw_time = calculate_sha256.software(filepath)

filesize = os.path.getsize(filepath)

print(f"""Results:
{ "=" * 80 }
File: {os.path.basename(filepath)}
Size: {filesize:,} bytes

Hardware Hash:
{hw_hash}
Hardware Time: {hw_time:.6f} seconds

Software Hash:
{sw_hash}
Software Time: {sw_time:.6f} seconds

Speedup: {sw_time/hw_time:.2f}x
{ "=" * 80 }""")

if hw_hash == sw_hash:
    print("✓ Hashes match! Hardware acceleration successful!")
else:
    print("⚠ Warning: Hardware and software hashes don't match!")
print()

if __name__ == "__main__":
    main()

```

קטע קיד 32 – קוד הPython האחראי על תקשורת עם המשתמש, ריפוד ושליחת המידע DMA

## 9.2.2. קוד הפייטון לצורך בניית גרפים לחקירה

```

from pynq import Overlay
from pynq.lib.dma import DMA
from pynq import allocate
import numpy as np
import struct
import time
from hashlib import sha256
import os
import matplotlib.pyplot as plt
from statistics import mean, stdev

ITERATIONS = 50

# SHA-256 Padding
def sha256_pad(message: bytes) -> np.ndarray:
    length = len(message) * 8
    message += b'\x80'
    while (len(message) % 64) != 56:
        message += b'\x00'
    message += struct.pack('>Q', length)
    return np.frombuffer(message, dtype='>u4')

# Hardware SHA-256 Calculation
def calculate_sha256.hardware(filepath: str) -> tuple:
    overlay = Overlay("SHA256_Block_Design.bit")
    dma = overlay.axi_dma_0

    with open(filepath, "rb") as f:
        data = f.read()

    input_data = sha256_pad(data)
    data_buffer = allocate(shape=(len(input_data),), dtype=np.uint32)
    output_buffer = allocate(shape=(8,), dtype=np.uint32)

    execution_times = []
    for _ in range(ITERATIONS):
        np.copyto(data_buffer, input_data)
        start_time = time.time()
        dma.sendchannel.transfer(data_buffer)
        dma.sendchannel.wait()
        dma.recvchannel.transfer(output_buffer)
        dma.recvchannel.wait()
        execution_times.append(time.time() - start_time)

    del data_buffer, output_buffer
    return mean(execution_times), stdev(execution_times), len(data)

# Software SHA-256 Calculation
def calculate_sha256.software(filepath: str) -> tuple:
    with open(filepath, "rb") as f:
        data = f.read()

    execution_times = []
    for _ in range(ITERATIONS):
        start_time = time.time()
        sha256(data).hexdigest()

```

```

        execution_times.append(time.time() - start_time)

    return mean(execution_times), stdev(execution_times), len(data)

# Generate Test Files
def create_test_files():
    sizes = [10, 100, 1000, 10000, 45000, 100000, 500000, 1_000_000, 2_000_000,
4_000_000, 8_000_000, 16_000_000, 23_000_000]
    test_dir = "test_files"
    os.makedirs(test_dir, exist_ok=True)

    files = []
    for size in sizes:
        filepath = os.path.join(test_dir, f"test_{size}_bytes.txt")
        if not os.path.exists(filepath) or os.path.getsize(filepath) != size:
            with open(filepath, "wb") as f:
                f.write(b'A' * size)
        files.append(filepath)

    return files

# Benchmark and Plot Results
def benchmark():
    print("Starting calculation")
    files = create_test_files()
    results = []

    for filepath in files:
        hw_mean, hw_std, filesize = calculate_sha256_hardware(filepath)
        sw_mean, sw_std, _ = calculate_sha256_software(filepath)
        speedup = sw_mean / hw_mean
        results.append((filesize, hw_mean, hw_std, sw_mean, sw_std, speedup))

    results.sort()
    sizes, hw_times, hw_errors, sw_times, sw_errors, speedups = zip(*results)

    plt.figure(figsize=(12, 15))

    plt.subplot(3, 1, 1)
    plt.errorbar(sizes, hw_times, yerr=hw_errors, fmt='o-', label='Hardware',
    capsizes=5)
    plt.errorbar(sizes, sw_times, yerr=sw_errors, fmt='o-', label='Software',
    capsizes=5)
    plt.xscale('log')
    plt.yscale('log')
    plt.xlabel('Data Size (bytes)')
    plt.ylabel('Execution Time (seconds)')
    plt.title('SHA-256 Performance Comparison')
    plt.legend()
    plt.grid(True, which="both", ls="--")

    plt.subplot(3, 1, 2)
    plt.plot(sizes, speedups, 'o-', color='green')
    plt.axhline(y=1.0, color='r', linestyle='--', alpha=0.7)
    plt.xscale('log')
    plt.xlabel('Data Size (bytes)')
    plt.ylabel('Speedup (Software/Hardware)')
    plt.title('Hardware Acceleration Speedup')

```

```

plt.grid(True, which="both", ls="--")

plt.subplot(3, 1, 3)
hw_time_per_byte = [t/s for t, s in zip(hw_times, sizes)]
sw_time_per_byte = [t/s for t, s in zip(sw_times, sizes)]

plt.plot(sizes, hw_time_per_byte, 'o-', label='Hardware Time per Byte')
plt.plot(sizes, sw_time_per_byte, 'o-', label='Software Time per Byte')
plt.xscale('log')
plt.yscale('log')
plt.xlabel('Data Size (bytes)')
plt.ylabel('Time per Byte (seconds)')
plt.title('Normalized Processing Time')
plt.legend()
plt.grid(True, which="both", ls="--")

plt.tight_layout()
plt.savefig('sha256_performance.png')
plt.show()

benchmark()
    
```

קטע קוד 33 – קוד פ'יינט לבניית גרפים לחקירה התוצאות

קוד זה לא מוסבר בספר זה, מכיוון שהוא נוצר אך ורק לצורך הבדיקה ויצירת גרפים תואמים למידע. כמובן שניתן פה לעבור על הקוד ולראות את השימוש שלו ואת אופן הפעולה שלו.

הweeneyון של הקוד הוא בדיקת התלות בין זמן העבודה לבין גודל המידע המוכנס. לכן לצורך זה מוכנים ע"י קוד זה מספר קבוע txt בגודלים שונים (מתחילה מביטים בודדים ומגיע לעשרות מגה בייט). קבצים אלו נבדקים במספר איטרציות (על מנת לקבל תוצאה אמינה ככל האפשר), ובסיומו של דבר מוצגים למסך מספר גרפים המוסברים בפרק 6.4.4 (בדיקות ההשערה לגבי ה-DMA).

### 9.2.3. קוד לבדיקת תדרי השעונים בכרטיס ה-PYNQ

```

from pynq import Clocks

print(f'CPU: {Clocks.cpu_mhz:.6f}MHz')
print(f'FCLK0: {Clocks.fclk0_mhz:.6f}MHz')
print(f'FCLK1: {Clocks.fclk1_mhz:.6f}MHz')
print(f'FCLK2: {Clocks.fclk2_mhz:.6f}MHz')
print(f'FCLK3: {Clocks.fclk3_mhz:.6f}MHz')
    
```

קטע קוד 34 – הצגת תדרי השעונים השונים בכרטיס

הקוד מדפיס את תדרי השעונים השונים ביחידות של MHz בדיק ש- 6 ספורות לאחר הנקודה העשוריית.

#### 9.2.4. מוצא המערכת

מוצא המערכת נראה بصورة הבאה, כאשר את הקווים מחליפים שמות הקבצים שנמצאו, הקבוץ הנבחר (והגודל) ולאחריו התוצאות של החומרה ושל התוכנה.

```
=====
          SHA-256 Hardware Accelerator
          Calculate SHA-256 hash using FPGA acceleration
=====

Available images:
1. ____
2. ____
3. ____
4. ____
5. ____
6. ____
7. ____
8. ____
9. ____

Enter image name or number (or 'q' to quit)
Choice: ____

Processing ____...
Calculating hardware hash...
Calculating software hash...
Results:
=====
File: ____.
Size: ____ bytes

Hardware Hash:
_____
Hardware Time: ____ seconds

Software Hash:
_____
Software Time: ____ seconds

Speedup: ____x
=====
✓ Hashes match! Hardware acceleration successful!
```

## .10. ביבליוגרפיה

- [1] U.S. Department of Commerce, "FIPS PUB 180-4 - Secure Hash Standard (SHS)," August 2015. [Online]. Available: <http://dx.doi.org/10.6028/NIST.FIPS.180-4>. [Accessed 25 02 2025].
- [2] dmarman, "SHA256 algorithm explained," [Online]. Available: <https://sha256algorithm.com/>. [Accessed 25 02 2025].
- [3] Xilinx, "tulembedded," [Online]. Available: <https://www.tulembedded.com/FPGA/ProductsPYNQ-Z2.html>. [Accessed 25 02 2025].
- [4] PYNQ, "PYNQ Overlays," [Online]. Available: [https://pynq.readthedocs.io/en/v2.5/pynq\\_overlays.html](https://pynq.readthedocs.io/en/v2.5/pynq_overlays.html). [Accessed 25 02 2025].

## 11. רשימות

### 11.1. רשימת איורים

|          |   |
|----------|---|
| 7 .....  | איור 1 – קובץ להורדה המובא יחד עם הערך המתkeletal מהבנסתו לפונקציית SHA-256 [1]             |
| 7 .....  | איור 2 – בדיקת אוטונומיות הקובץ ע"י הבנסתו למערכת המחשבת את ערך SHA-256 המתkeletal ממנו [2] |
| 11 ..... | איור 3 – תיאור גրפי של נוסחה 8 נוסחה 9 ונוסחה 10.   |
| 12 ..... | איור 4 – דיאגרמה המתארת את פעולה ה- SHA-256 המלאה   |
| 12 ..... | איור 5 – דיאגרמת בלוקים של תיאור פעולה ה- SHA-256   |
| 14 ..... | איור 6 - דוגמא למציאת ערך SHA-256 שלChunk-1 .....   |
| 15 ..... | ...Chunk-2 .....  |
| 17 ..... | איור 7 – דוגמא למציאת ערך SHA-256 שלChunk-2 .....   |
| 18 ..... | ...PYNQ-Z2 .....  |
| 18 ..... | איור 8 – תיאור השכבות והחלקים של ברטיס-ה- PYNQ-PS   |
| 19 ..... | איור 9 – תיאור השכבות והחלקים של ברטיס-ה- Z2 .....  |
| 20 ..... | איור 10 – התממשקות ה- PL- וה- PS- לפירופיות השונות [6]                                      |
| 21 ..... | איור 11 – שלבי העבודה להפעלת ברטיס-ה- PYNQ-PS   |
| 21 ..... | איור 12 – הגדרת כתובות IP סטטיות במחשב המשמש  |
| 21 ..... | איור 13 – ערכוי קרייה וכתיבה של פרוטוקול AXI  |
| 21 ..... | איור 14 – דיאגרמת זמינים של פרוטוקול AXIS   |
| 23 ..... | איור 15 – ציוד הפרוייקט .....   |
| 25 ..... | איור 16 – דיאגרמת בלוקים המתארת את מימוש המערכת .....                                       |
| 25 ..... | ZYNQ7 Processing System – 17 .....  |
| 26 ..... | איור 18 – DMA-Block – 18 .....  |
| 28 ..... | איור 19 – הגדרת בлок ה- DMA-Processing System Reset block – 20 .....                        |
| 28 ..... | ...inter Connect block – 21 .....   |
| 29 ..... | ...Smart Connect block – 22 .....   |
| 29 ..... | איור 23 – ה- IP IP_main_0 שעיצב – 23 .....  |
| 36 ..... | ...Costume .....  |
| 36 ..... | איור 24 – דיאגרמת בלוקים של מבנות המרכיבים .....  |
| 37 ..... | ...Packaged .....   |
| 37 ..... | איור 25 – השלב הראשון ביצירת ה- IP .....  |
| 38 ..... | ...packaged .....   |
| 38 ..... | איור 26 – השלב השני (והאחרון) ביצירת ה- IP .....  |
| 39 ..... | ...bitstream .....  |
| 39 ..... | איור 27 – היררכיית המערכת שהלמה אשר נשלה למנוע לקבלת bitstream .....                        |
| 39 ..... | ...block design .....   |
| 39 ..... | איור 28 – ביצוע ולידציה לה- Bitstream .....   |
| 39 ..... | ...block .....  |
| 39 ..... | איור 29 – קבלת אישור כי הולידציה עברה בהצלחה .....  |
| 39 ..... | ...enerima .....  |
| 39 ..... | איור 30 – הרצת סינטזה ואימפלמנטציה .....  |
| 39 ..... | ...received .....   |
| 39 ..... | איור 31 – קבלת אישור כי ה- Bitstream נוצר בהצלחה .....                                      |
| 51 ..... | ...NALL .....   |
| 51 ..... | איור 32 – קבלת תוצאה נכונה בוכנית NALL .....  |
| 52 ..... | ...packaged .....   |
| 52 ..... | איור 33 – דוגמא לשימוש במערכת, קבלת קלט מהמשתמש .....                                       |
| 52 ..... | ...bitstream .....  |
| 52 ..... | איור 34 – דוגמא לשימוש במערכת, קבלת התוצאה .....  |
| 52 ..... | ...output .....   |
| 53 ..... | ...Pictures .....   |
| 53 ..... | איור 35 – צילום מסך של תיוקית Pictures בתוך ה- PYNQ והתבולה שלה .....                       |
| 53 ..... | ...display .....  |
| 53 ..... | איור 36 – הבנתם קבצים שונים לтиוקית pictures .....  |
| 54 ..... | ...electricity.jpg .....  |
| 54 ..... | איור 37 – הצגת רשימת האיורים התואמת את איור 36 .....  |
| 54 ..... | ...pynq.jpg .....   |
| 55 ..... | ...rocks.jpg .....  |
| 55 ..... | איור 38 – ביצוע חישוב HASH על ה- Bitstream .....  |
| 55 ..... | ...abc .....  |
| 56 ..... | ...blue water.jpg .....   |
| 56 ..... | איור 39 – תוצאה החישוב לקובץ electricity.jpg .....  |
| 58 ..... | ...45KB .....   |
| 59 ..... | איור 40 – תוצאות החישוב לקובץ pynq.jpg .....  |
| 59 ..... | ...81KB .....   |
| 61 ..... | ...rocks.jpg .....  |
| 61 ..... | איור 41 – תוצאות החישוב לקובץ blue water.jpg .....  |
| 61 ..... | ...701KB .....  |
| 62 ..... | ...window.jpg .....   |
| 62 ..... | איור 42 – תוצאות החישוב לקובץ window.jpg .....  |
| 62 ..... | ...14.62MB .....  |
| 64 ..... | איור 43 – בדיקת מהירות שעון ה- CPU מול שעון FPGA .....                                      |
| 64 ..... | ...clock .....  |
| 64 ..... | איור 44 – בדיקת מהירות שעון ה- CPU מול שעון FPGA .....                                      |
| 64 ..... | ...clock .....  |
| 65 ..... | איור 45 – שלושת הגרפים המשמשים לנתחות התוצאות .....   |
| 65 ..... | ...45KB .....   |
| 65 ..... | איור 46 – תואצת החומרה בקלט בגודל .....   |
| 65 ..... | ...45 .....   |
| 67 ..... | איור 47 – המציג את שליחת ההודעה "abc" במנגן AXI .....                                       |
| 67 ..... | ...clocks .....   |
| 68 ..... | ...axis_tlast .....   |
| 68 ..... | איור 48 – חישוב התוצאה לוח ב- DIOK .....  |
| 68 ..... | ...clock .....  |
| 69 ..... | ...tlast .....  |
| 69 ..... | איור 49 – שליחת תוצאה HASH על המדע בחזרה למשתמש .....                                       |
| 69 ..... | ...W .....  |
| 70 ..... | איור 50 – סימולציית המציג את השינוי במערךلوح הזמן .....                                     |
| 70 ..... | ...state .....  |
| 71 ..... | איור 51 – הצגת שינוי רגיסטרו state .....  |

|          |   |
|----------|---|
| 66 ..... | איור 52 – חישוב המילה הבאה במערךلوح הזמן.....       |
| 66 ..... | איור 53 – חישוב המשתנים הפעילים של הround הבא ..... |

## 11.2. רשימה קטעי קוד

|          |  |
|----------|--|
| 30 ..... | קטע קוד 1 - sha256_pkg - מעתנים ופונקציות לצורך יצוא SHA-256                             |
| 31 ..... | קטע קוד 2 - sha256_expansion - בлок למציאת מערךلوح הזמן W ב $15 > round\_index$          |
| 32 ..... | קטע קוד 3 - sha256_round - בлок לחישוב המשתנים הפעילים הבאים (לצורך השלמת round) costume |
| 32 ..... | קטע קוד 4 - sha256_main - תיאור כניסה ויציאה הIP   |
| 33 ..... | קטע קוד 5 - sha256_main - הסיגנלים הפנימיים של המערכת                                    |
| 34 ..... | קטע קוד 6 - sha256_main instantiation של שני modules: expand ו round                     |
| 34 ..... | קטע קוד 7 - sha256_main - מימוש המוקס לצורך קביעת המילה הנוכחית.....                     |
| 34 ..... | קטע קוד 8 - sha256_main - תחילת הבלוק הסינכרוני - ריסט סיגנל פעיל בנמו.                  |
| 35 ..... | קטע קוד 9 - מצב ה RECEIVE_DATA של מבנות המצבים.....                                      |
| 35 ..... | קטע קוד 10 - מצב FINISH_CALC של מבנות המצבים.....  |
| 36 ..... | קטע קוד 11 - מצב SEND_RESULT של מבנות המצבים.....  |
| 40 ..... | קטע קוד 12 – יבוא ספריות פיטון לצורך הקוד תוכנה .....                                    |
| 41 ..... | קטע קוד 13 – חלק התוכנה האחראי על ריפוד המידעafi שמודגר ב프וטוקול SHA.                    |
| 42 ..... | קטע קוד 14 – צריבת ה bitstream ושמירת התמונה כמידע בינהי .....                           |
| 42 ..... | קטע קוד 15 – הקצאת מהשניות זיכרון עבור קלט ופלט המידע.....                               |
| 43 ..... | קטע קוד 16 – שליחת המידע לא DMA וקבלת של בחרה מה DMA.....                                |
| 43 ..... | קטע קוד 17 – שמירת התוצאה בתצורה נוחה להציג ושליחת המידע החוצה מהפונקציה.....            |
| 44 ..... | קטע קוד 18 – פונקציה לחישוב תוצאה HASH בס- CPU ומידת זמן חישוב זה .....                  |
| 45 ..... | קטע קוד 19 – פונקציה להציג הקבצים שנמצאו לחישוב .....                                    |
| 47 ..... | קטע קוד 20 – הדפסת הוכתרת למשתמש .....   |
| 47 ..... | קטע קוד 21 – הציג האופציות למשתמש וקבלת תגובה .....                                      |
| 47 ..... | קטע קוד 22 – התנהלות בקבלת מספר מהמשתמש.....   |
| 48 ..... | קטע קוד 23 – התנהלות בקבלת מל מהמשתמש.....   |
| 49 ..... | קטע קוד 24 – חישוב והדפסה התוצאה למשתמש .....  |
| 50 ..... | קטע קוד 25 – השוואה בין התוצאה שהתקבלה מה-CPU ל贤出א מה-FPGA.....                          |
| 50 ..... | קטע קוד 26 – קוד הפיטון הקיים לפונקציה main .....  |
| 69 ..... | קטע קוד 27 המכיל את הקבאים והפונקציות.....   |
| 70 ..... | קטע קוד 28 קומבינטוריה האחראי על חישוב המשתנים הפעילים הבאים .....                       |
| 71 ..... | קטע קוד 29 על חישוב המילה הבאה W (מימוש נוסחה 3) .....                                   |
| 73 ..... | קטע קוד 30 המodule main של החומרה.....   |
| 77 ..... | קטע קוד 31 Costume IP לשכנתה.....  |
| 81 ..... | קטע קוד 32 – קוד הפיטון האחראי על תקשורת עם המשתמש, ריפוד ושליחת המידע DMA.....          |
| 84 ..... | קטע קוד 33 – קוד פיטון לבניית גרפים לחקר התוצאות .....                                   |
| 84 ..... | קטע קוד 34 – הציג תדרי השעונים השונים בברטיס .....                                       |

## 11.3. רשימה נוסחים

|         |                                       |
|---------|---------------------------------------|
| 8 ..... | נוסחה 1 – $\sigma_1$ .....            |
| 8 ..... | נוסחה 0 – $\sigma_0$ .....            |
| 8 ..... | נוסחה $W_{ij}$ – 3 .....              |
| 9 ..... | נוסחה 4 – $\Sigma_0$ .....            |
| 9 ..... | נוסחה 5 – $\Sigma_1$ .....            |
| 9 ..... | נוסחה 6 – Choice .....                |
| 9 ..... | נוסחה 7 – Majority .....              |
| 9 ..... | נוסחה 8 – temp1 .....                 |
| 9 ..... | נוסחה 9 – temp2 .....                 |
| 9 ..... | נוסחה 10 – New active variables ..... |