Acceleration of Local Sensitivity Hashing for Genome Assembly

פרויקט #6392
סמסטר חורף 2022
בהנחיית לאוניד יביץ

מגישים:
גל אלבז 209050475
יואב חרוסט 307951335

תוכן עניינים

הקדמה

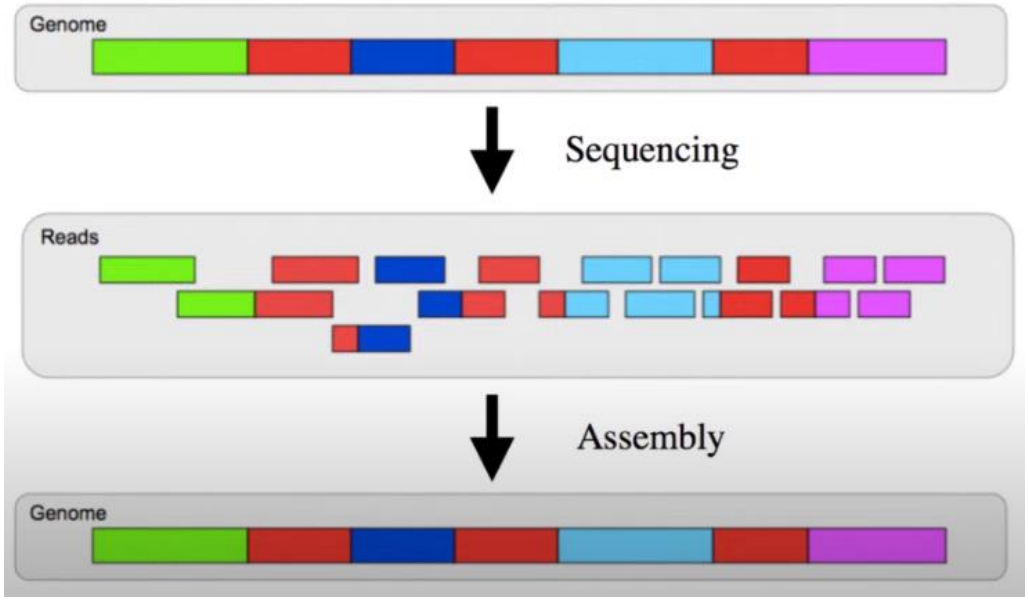
הרכבת הגנום הוא מונח המתאר תהליך בו מתבצע חיבור של מספר רב של רצפים נוקלאוטידים בכדי ליצור את הגנום הכולל.
קיימות שתי גישות עיקריות של הרכבת הגנום:

1. Mapping assembly.
2. De novo assembly.

בפרויקט זה עסקנו בגישה מספר 2 אשר מבצעת את תהליך ההרכבה של גנום מאפס, ללא רפרנס על המידע הגנטי.
זאת בניגוד לגישה מספר 1 אשר לפיה הרכבת הגנום מתבצעת בהתבסס על מידע גנטי ידוע מראש.

הרכבת הגנום בשיטת de novo דורשת עלות חישובית גובהה אשר מהווה צוואר בקבוק משמעותי כאשר ניגשים לבעיות הכוללות גנומים ארוכים.
על מנת להצדיק שימוש בהרכבת גנום מבוססת de novo יש צורך בהאצה של המשימות הגוזלות זמן רב בתהליך.

בפרויקט זה, נחקור גישה הסתברותית המנסה לפתור את צוואר הבקבוק בעזרת אלגוריתם MHAP (MinHash Alignment Process), אשר משתמש בטכניקה של שימוש בפונקציות hash לצורך ייצוג קומפקטי יותר של קריאות הרצף (reads) והשוואתם על מנת לדעת האם ניתן לאחד את הreads לייצוג חופף (contigs) שישמשו ליצירת מפה פיזיקלית המשחזרת את רצף הDNA המקורי.



איור 1
תיאור גנום כולל, פירוקו לreads והרכבת הגנום מחדש על פי אזורים חופפים לcontigs.

מטרות הפרויקט

כאמור, בפרויקט זה תכננו מאיץ חומרתי אשר יפתור את צוואר הבקבוק הנובע מהעלות החישובית של הרכבת הגנום בשיטת de novo.
בפרויקט זה:

- למדנו על אלגוריתם הMinHash שתפקידו להאיץ את תהליך הרכבת הגנום.
- בנינו תכנון חומרתי היררכי של האלגוריתם במטרה להאיץ את חישוב הרכבת הגנום ככל הניתן.
- מימשנו סימולטור תוכנתי בPython אשר מחקה את התנהגות האלגוריתם.
- מימשנו סימולטור חומרתי בSystemVerilog אשר מחקה את התנהגות האלגוריתם.
- בצענו מספר רב של סימולציות ובדקנו התאמה בתוצאות בין שני הסימולטורים המתוארים.
- בצענו סינתזה לתכנון הלוגי שרשמנו, הסינתזה כללה קומפילציה של התכנון והפקת דוחות על המסלול הקריטי, שטח וצריכת ההספק של הרכיב.
- תכנון ומימוש מעגל VLSI.

אלגוריתם MinHash

האלגוריתם ידוע באופן פורמלי בשם **'The AltaVista algorithm'** אשר תוכנן בראשיתו לצורך זיהוי של דפי אינטרנט כפולים.
אם נחשוב על דף אינטרנט כאחד מקריאות הרצף (reads) והמטרה שלנו היא למצוא **reads** אשר דומים זה לזה, קיים הצדקה והיגיון בשימוש באלגוריתם הנ"ל למטרה שלנו.

אלגוריתם MHAP (MinHash Alignment Process) מהווה משערך מהיר של זהות ג'קארד (**Jaccard Similarity**), אשר מוגדרת עבור שני סטים של רצפים S_1, S_2 בקשר:

$$(1)\ J(S_1, S_2) = \frac{|\Gamma(S_1) \cap \Gamma(S_2)|}{|\Gamma(S_1) \cup \Gamma(S_2)|}; 0 \le J \le 1$$

על פי שערוך זה, אנו מחלקים את הרצף הגנטי שלנו לסט של תתי רצפים אשר מכונים כ**kmers** שעליהם מתבצעים מספר פעולות מתמטיות. לבסוף, תוצאת השערוך מחושבת לפי נוסחה מספר **(1)**.

ניתן לראות שעל פי נוסחה מספר **(1)**, זהות ג'קארד שווה לאפס כאשר שני הסטים זרים זה לזה.
באופן דומה, זהות ג'קארד שווה ל1 כאשר שני הסטים זהים זה לזה לחלוטין, אחרת התוצאה היא בין 0 ל1.

השלב הראשון של האלגוריתם

בשלב הראשון של האלגוריתם, מתבצעת חלוקה של הרצף הגנטי הנתון לתתי מחרוזות, **kmers**.
מספר ה**kmers** שכל רצף תורם נתון על ידיי הנוסחה:

$$(2)\ \#kmers = N_c - length(kmer) + 1$$

כאשר מתקיים:

- N_c הוא אורך הרצף המקורי.
- $length(kmer)$ הוא אורך כל תת רצף (פרמטר נתון לבחירה).

בסיום שלב זה, מקבלים סט של ה**kmers** המרכיבים את הרצף המקורי.

לדוגמא, נבצע את השלב הראשון של האלגוריתם על הרצפים הגנטיים הבאים, נקבע שאורך כל **kmers** הוא 3.

$$S_1: CATGGACCGACCAG$$
$$S_2: GCAGTACCGATCGT$$

עם סיום השלב הראשון, יתקבלו הסטים הבאים:

$$S_1 - kmers: [CAT, ATG, TGG, GGA, GAC, ACC, CCG, CGA, GAC, ACC, CCA, CAG]$$
$$S_2 - kmers: [GCA, CAG, AGT, GTA, TAC, ACC, CCG, CGA, GAT, ATC, TCG, CGT]$$

לפי נוסחה מספר 2, אכן התקבלו 12 **kmers** סה"כ.

השלב השני של האלגוריתם

בשלב השני של האלגוריתם, כל **kmer** עובר תרגום לחתימה **(fingerprint)** אשר מיוצגת על ידי מספר שלם.

בפרויקט שלנו, בחרנו לבצע את התרגום הבא:

כידוע, הנוקלאוטידים המרכיבים את חומצת הגרעין הם **A, T, G, C**.

תחילה, בחרנו לייצג כל אות-אות באמצעות ספרה בודדת בבסיס עשרוני:

$$\{A = 0, T = 1, G = 2, C = 3\}$$

לאחר מכן, בצענו המרה של ספרה-ספרה לייצוג בינארי ברוחב 2 ביט:

$$\{0_{10} = 00,\ 1_{10} = 01, 2_{10} = 10, 3_{11} = 11\}$$

בהמשך לדוגמה הקודמת, נבצע מעבר לייצוג הנ"ל עבור הסטים של הkmers שקיבלנו:

$$S_1 - \text{kmers: } [CAT, ATG, TGG, GGA, GAC, ACC, CCG, CGA, GAC, ACC, CCA, CAG]$$

```
C:\Users\Gal\Desktop\projectB\PYTHON\venv\Scripts\python.exe C:/Users/Gal/Desktop/projectB/PYTHON/main.py
The conversion of set number 1 is:
 ['301', '012', '122', '220', '203', '033', '332', '320', '203', '033', '330', '302']
```

איור 2

פירוק הסט S_1 לייצוג אות-אות בבסיס עשרוני

$$S_2 - \text{kmers: } [GCA, CAG, AGT, GTA, TAC, ACC, CCG, CGA, GAT, ATC, TCG, CGT]$$

```
C:\Users\Gal\Desktop\projectB\PYTHON\venv\Scripts\python.exe C:/Users/Gal/Desktop/projectB/PYTHON/main.py
The conversion of set number 2 is:
 ['230', '302', '021', '210', '103', '033', '332', '320', '201', '013', '132', '321']
```

איור 3

פירוק הסט S_2 לייצוג אות-אות בבסיס עשרוני

כעת נעבור לייצוג הבינארי:

```
C:\Users\Gal\Desktop\projectB\PYTHON\venv\Scripts\python.exe C:/Users/Gal/Desktop/projectB/PYTHON/main.py
The conversion of set number 1 is:
 ['301', '012', '122', '220', '203', '033', '332', '320', '203', '033', '330', '302']
The equivalent binary representation is:
 ['110001', '000110', '011010', '101000', '100011', '001111', '111110', '111000', '100011', '001111', '111100', '110010']
```

איור 4

מעבר לתצוגה בינארית אות-אות עבור הסט S_1.

```
C:\Users\Gal\Desktop\projectB\PYTHON\venv\Scripts\python.exe C:/Users/Gal/Desktop/projectB/PYTHON/main.py
The conversion of set number 2 is:
 ['230', '302', '021', '210', '103', '033', '332', '320', '201', '013', '132', '321']
The equivalent binary representation is:
 ['101100', '110010', '001001', '100100', '010011', '001111', '111110', '111000', '100001', '000111', '011110', '111001']
```

איור 5

מעבר לתצוגה בינארית אות-אות עבור הסט S_2.

הסיבה שבגללה בצענו מעבר בין הייצוג הגנטי דרך שני בסיסים היא שראשית רצינו לקבל ייצוג קומפקטי וקריא של הרצף הגנטי באמצעות ספרות בבסיס עשרוני.
לאחר מכן, בצענו המרה של ספרה-ספרה לייצוג בינארי וכך קיבלנו את החתימה (fingerprint) שאיתה אנו עובדים לצורך החישובים.

נשים לב שעבור איור מספר 4, לא מתקיים השוויון הבא עבור הkmer הראשון:

$$(301)_{10} = (110001)_2$$

שכן ההמרה מתבצעת ספרה-ספרה כך שלמעשה מתקיים:

$$(301)_{\mathrm{DnaDecimal}} = (110001)_{\mathrm{DnaBinary}}$$

בעזרת ייצוג זה, הצלחנו לבטא את הערך המספרי של החתימה על ידיי מספר שבהכרח קטן יותר מאשר הייצוג הספרתי-דצימלי.
כלומר:

$$(110001)_{\mathrm{DnaBinary}} = (110001)_2 = (49)_{10} < (301)_{\mathrm{DnaDecimal}}$$

יתרון נוסף של אופן ייצוג זה הוא שהצלחנו לבצע מיפוי של kmers ארוכים לחתימה שמתורגמת למספר עשרוני נמוך שניתן לייצג באמצעות פחות ביטים.

נדגים יתרון זה על הרצף הבא ובחירה של kmers באורך 16.

$$\mathrm{SeqDna} = \mathrm{ATGCGTAGCATAGTACAGTACATGTTACAGTA}$$

הkmer ראשון של רצף זה נתון על ידיי:

$$\mathrm{kmer}[0] = \mathrm{ATGCGTAGCATAGTAC}$$

נעביר לייצוג DnaDecimal כפי שהסברנו:

$$\mathrm{kmer}[0]_{\mathrm{DnaDecimal}} = 0123210230102103$$

במידה ונעבוד עם ייצוג זה כfingerprint שלנו, נקבל ייצוג בינארי באורך 43 ביט!

לעומת זאת, על פי השיטה שלנו, תחילה נעבור לייצוג DnaBinary כפי שהסברנו:

$$\mathrm{kmer}[0]_{\mathrm{DnaBinary}} = 00011011100100101100010010010011$$

במידה ונעבוד עם חתימה זו בתור המספר שלנו, מתקבל כי החתימה מיוצגת בבינארי כמספר ברוחב 32 ביט בלבד.

השלב השלישי של האלגוריתם

בשלב השלישי של האלגוריתם, מעבירים כל אחת מהחתימות שיצרנו דרך N פונקציות hash ידועות מראש, את התוצאה המתקבלת מהפעולה:

$$\mathrm{hash}_i(\mathrm{kmer}_j)$$

כאשר:

$$i \in \{1, N\}$$
$$j \in \{1, N_c - \mathrm{len}(\mathrm{kmer}) + 1\}$$

שומרים ביחד עם אותו הkmer עבור השלב הרביעי באלגוריתם.

לדוגמה, עבור הkmers של הרצפים הגנטיים S_1, S_2 שניתחנו קודם לכן ובחירה של $N = 4$ פונקציות hash מסוימות, נקבל:

Γ_1	Γ_2	Γ_3	Γ_4	
19	14	57	36	CAT
14	57	36	19	ATG
58	37	16	15	TGG
40	23	2	61	GGA
33	28	11	54	GAC
5	48	47	26	ACC
22	1	60	43	CCG
24	7	50	45	CGA
33	28	11	54	GAC
5	48	47	26	ACC
20	3	62	41	CCA
18	13	56	39	CAG

	Γ_1	Γ_2	Γ_3	Γ_4
GCA	36	19	14	57
CAG	18	13	56	39
AGT	11	54	33	28
GTA	44	27	6	49
TAC	49	44	27	6
ACC	5	48	47	26
CCG	22	1	60	43
CGA	24	7	50	45
GAT	35	30	9	52
ATC	13	56	39	18
TCG	54	33	28	11
CGT	27	6	49	44

איור 6

הפעלת פונקציות הhash על כל kmer.

לקוח מתוך המאמר

כל עמודה מתארת את תוצאת פונקציית הhash המתקבלת על הפעלת הפונקציה על fingerprint של כל אחד מהkmers אשר מתוארים בשורות.

השלב הרביעי של האלגוריתם

בשלב הרביעי של האלגוריתם נלקחת התוצאה המינימאלית מכל פונקציית hash, כלומר עבור כל עמודה לוקחים את הערך המינימאלי.

בהמשך לדוגמה המוצגת בשלבים הקודמים, נקבל את תוצאת הביניים הבאה:

Γ_1	Γ_2	Γ_3	Γ_4	
19	14	57	36	CAT
14	57	36	19	ATG
58	37	16	15	TGG
40	23	2	61	GGA
33	28	11	54	GAC
5	48	47	26	ACC
22	1	60	43	CCG
24	7	50	45	CGA
33	28	11	54	GAC
5	48	47	26	ACC
20	3	62	41	CCA
18	13	56	39	CAG

	Γ_1	Γ_2	Γ_3	Γ_4
GCA	36	19	14	57
CAG	18	13	56	39
AGT	11	54	33	28
GTA	44	27	6	49
TAC	49	44	27	6
ACC	5	48	47	26
CCG	22	1	60	43
CGA	24	7	50	45
GAT	35	30	9	52
ATC	13	56	39	18
TCG	54	33	28	11
CGT	27	6	49	44

איור 7

שמירה של הערך המינימאלי מכל עמודה, כלומר תוצאות

החישוב של פונקציית הhash הi.

לקוח מתוך המאמר

התוצאות המינימאליות מסומנות באדום עבור כל עמודה, ביחד איתם מסומן הkmer המתאים אשר הוביל לתוצאה המינימאלית בחישוב.

התוצאות המינימאליות עבור על סט נשמרות במערך חד-מימדי באופן הבא:

$$S_{1_{minArray}} = [5,1,2,15]$$
$$S_{2_{minArray}} = [5,1,6,6]$$

השלב החמישי של האלגוריתם

השלב החמישי והאחרון באלגורית הוא חישוב זהות ג'קארד בהתבסס על תוצאות המינימום שהתקבלו מהשלב הקודם.

על פי נוסחה מספר 1 מתקיים כי מקדם הזהות מחושב לפי:

$$(1)\ J(S_1,S_2) = \frac{|\Gamma(S_1) \cap \Gamma(S_2)|}{|\Gamma(S_1) \cup \Gamma(S_2)|};\ 0 \le J \le 1$$

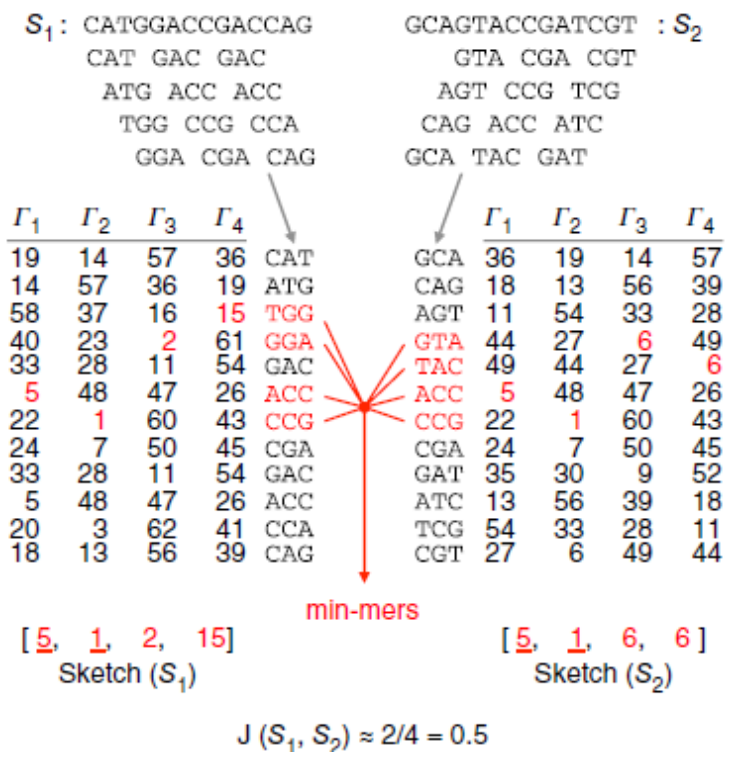
כלומר יש לקחת את החיתוך של הסטים ולחלק בגודל התוצאות שהתקבלו.

בדוגמה שהרצנו עד כה, נקבל כי זהות ג'קארד היא:

$$\begin{cases} S_{1_{minArray}} \cap S_{2_{minArray}} = 2 \text{ elements} \\ S_{1_{minArray}} \cup S_{2_{minArray}} = 4 \text{ elements} \end{cases} \rightarrow J(S_1,S_2) = 0.5$$

על פי מקדם הזהות, נקבל אינדיקציה עד כמה הרצפים הגנטיים זהים.

ניתן לראות את כל השלבים אחד אחריי השני באיור הבא:



איור 8

פעולת האלגוריתם באופן מלא

לקוח מתוך המאמר

הארכיטקטורה

מטרתנו הייתה לדמות את התנהגות האלגוריתם הנ"ל באמצעות שפת תיאור החומרה **SystemVerilog**.
התכנון שלנו כלל חלוקה של האלגוריתם למספר מודולים המסודרים בצורה היררכית כך שכל מודול מטפל בשלב שונה באלגוריתם ומסתמך על הפלט של המודול הקודם לו.

הסתמכנו על עקרונות אופטימיזציה של תכנון באמצעות **pipeline** לצורך האצת החישובים והשגת התוצאה הסופית במספר בודד של מחזורי שעון.

בתכנון שלנו, פעלנו על פי האפיון הבא של כניסות ודרישות המערכת:

1. אורך רצף גנטי מורכב מ64 נוקלאוטידים.
2. אורך **kmer** הוא 16 נוקלאוטידים.
3. שימוש ב8 פונקציות **hash** הפועלות על פי הכלל הבא:

$$\{\mathrm{hash_i}\}_1^8 = (((\mathrm{kmer_i}\ \&\ \mathrm{randA_i})\ |\ \mathrm{randB_i}\ )\ \%\ \mathrm{n})$$

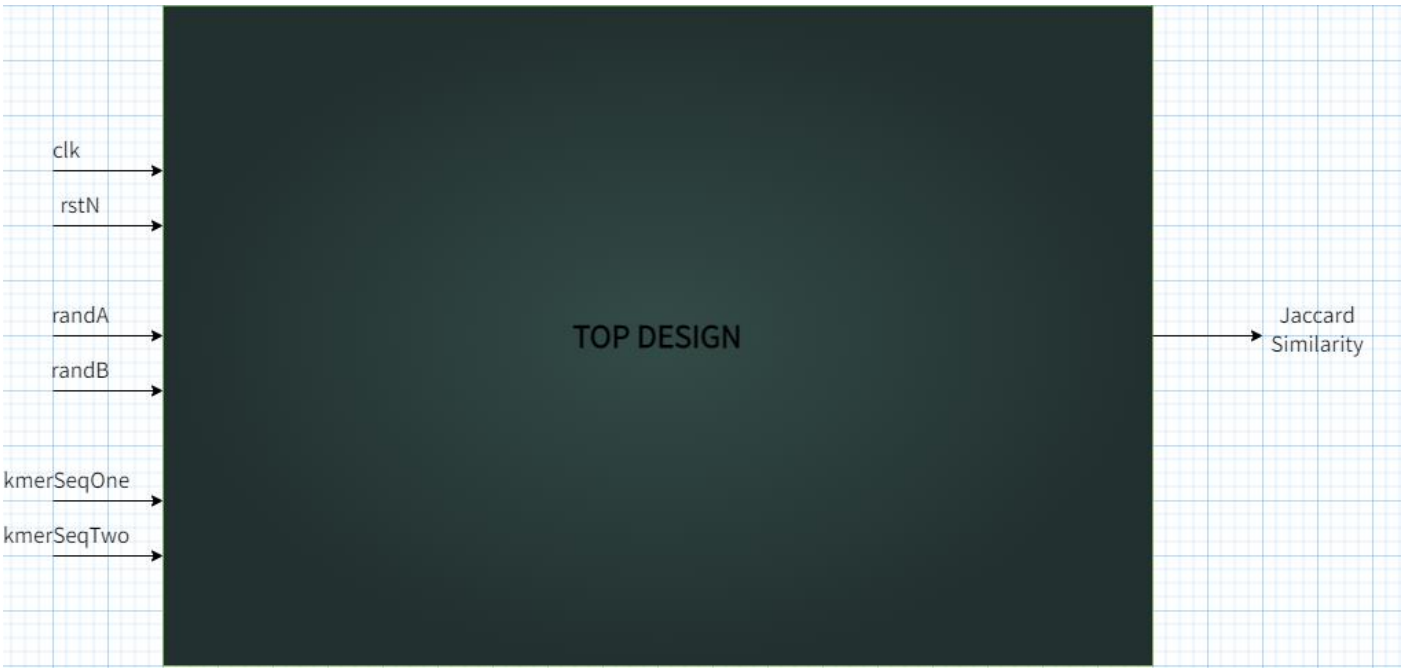
   - n הוא מקדם נרמול ושווה ל255.
   - $\mathrm{randA_i}, \mathrm{randB_i}$ הינם מספרים רנדומליים המתאימים לפונקציית ה**hash** ה**i**.
4. מקדם ג'קארד אינו מנורמל במספר פונקציות ה**hash** שבחרנו לייצוג.

כלומר, מקדם ג'קארד נע בין 0 ל8 כאשר 0 מייצג חוסר זהות ו8 מייצג זהות מוחלטת.

Top-Down Design

נתאר את המערכת שלנו מלמעלה-למטה כך שכל שנתקדם למטה בהיררכיה נתבונן בחלקים קטנים יותר על מנת להבין את המערכת הכוללת.

התכנון ברמה העליונה ביותר מודגם על ידי הסכימה הבאה:

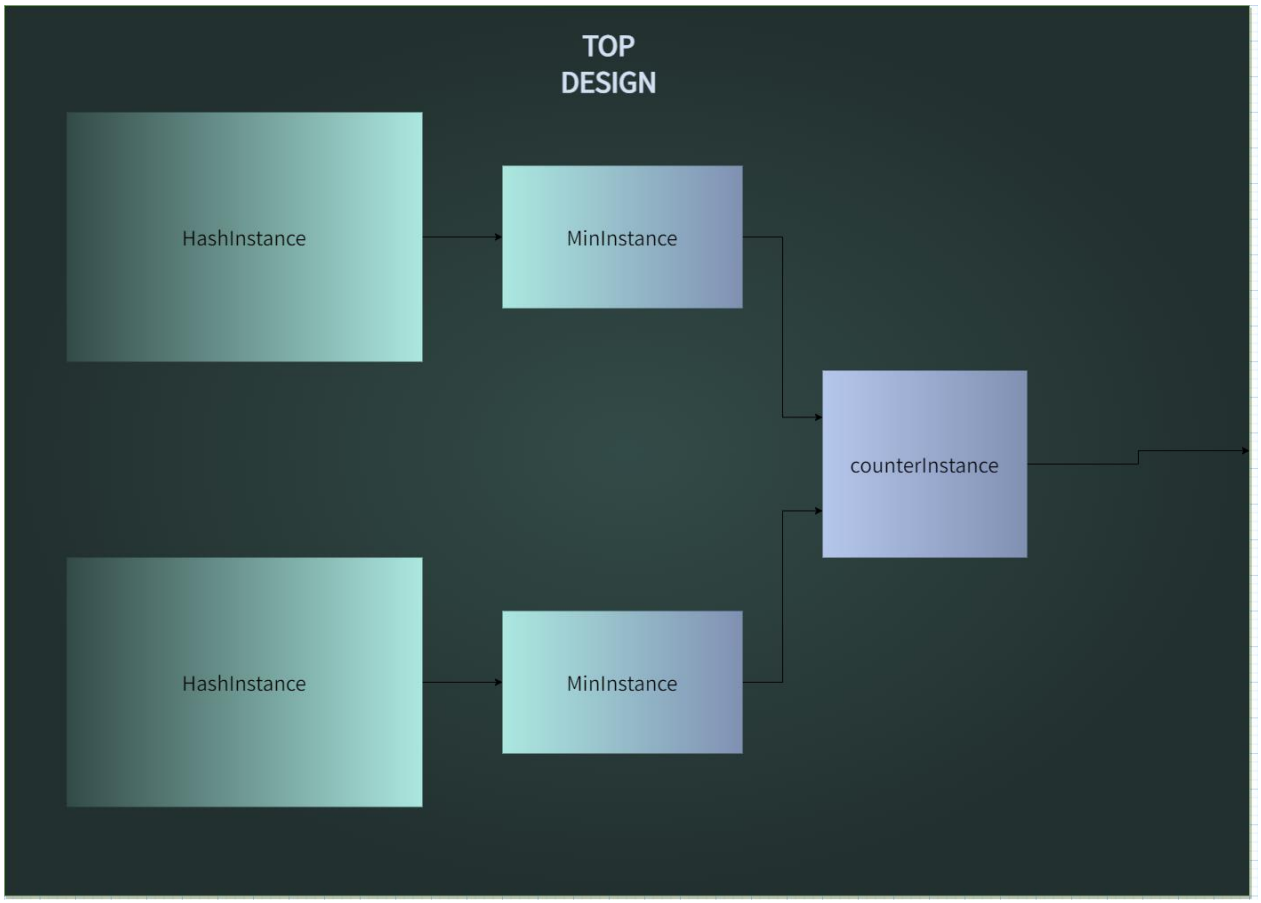


איור 9
סכמת בלוקים מופשטת של ה*Top Design*.

התכנון הנ"ל מכיל את הכניסות והיציאות של המערכת ברמת הרכיב הכולל.
ניתן לראות שהכניסות הם:

- שעון, clk.
- הדק איפוס, rstN.
- זוג מספרים רנדומליים המשמשים לחישוב פונקציות הhash.
- זוג מערכים המכילים את הkmers של הרצפים. המערכים המכילים את הkmers מוחזקים קבועים לאורך כל האלגוריתם, המספרים הרנדומליים משתנים כתלות בעליית שעון ופונקציית הhash הבאה שיש להריץ.

נבצע $zoom - in$ למערכת ונראה את התיאור המופשט הבא:



איור 10

close up על סכמת בלוקים מופשטת של ה*Top Design*.

במבט פנימי יותר ניתן לראות שהמודל הכולל מכיל סך הכל חמישה מודלים:

1. **hashInstance**, מודל אשר מבצע את פעולת ה**hash** על כל רצף גנטי.
   כיוון שיש לנו שני רצפים גנטיים, שכפלנו מודל זה.
2. **minInstance**, מודל אשר מוצא את התוצאה המינימאלית מבין התוצאות המתקבלות מפעולת ה**hash**.
   כיוון שיש לנו שני רצפים גנטיים שמועברים דרך פונקציית ה**hash**, שכפלנו מודל זה.
3. **counterInstance**, מודל אשר מבצע חישוב זהות בין שני התוצאות המינימאליות המתקבלות.

Hash-Instance

נתאר את מודל הHashInstance.
מודל זה אחראי על הפעלת פונקציית הhash על הkmers של הרצף הגנטי, בכל עליית שעון משתנה פונקציית הhash שמפעילים בכך שהמשתנים randA, randB מתחלפים.
כזכור על פי האפיון שלנו, פונקציית הhash שלנו מתוארת על פי הכלל הבא:

$$\{hash_i\}_1^8 = (((kmer_i \ \& \ randA_i) \mid randB_i \ ) \ \% \ n)$$

תיאור הכניסות והיציאות של המודל באופן מופשט נתון על ידי:

איור 11
סכמת בלוקים מופשטת של מודל ה*HashInstance*

ניתן לראות שהכניסות הן:

- שעון, clk.
- הדק איפוס, rstN.
- זוג מספרים רנדומליים המשמשים לחישוב פונקציות הhash.
- מערך המכיל את הkmers של הרצף. המערך מכיל את הkmers כקבועים לאורך כל האלגוריתם.

היציאה היא:

- מערך התוצאות, resHashing.

בכל עליית שעון הערכים של randA, randB משתנים על מנת לבצע את החישוב על החתימות עם פונקציית הhash הבא.
סה"כ עבור 8 פונקציות hash שונות יש צורך ב8 עליות שעון.

Min-Instance

נתאר את מודל הMinInstance.
מודל זה אחראי על קבלת תוצאות החישוב מפונקציית הhash ומציאת הערך המינימאלי.
נשים לב שמודל זה יכול לפעול כראוי רק בעליית השעון השנייה, בה מוכנות תוצאות החישוב של הפעלת פונקציית $hash_1$ על הkmers ועל כן יש להוסיף למודל זה השהיה.

תיאור הכניסות והיציאות של המודל באופן מופשט נתון על ידיי:

איור 12
סכמת בלוקים מופשטת של מודל MinInstance

ניתן לראות שהכניסות הן:

- שעון, clk.
- הדק איפוס, rstN.
- מערך התוצאות, resHashing.

היציאה היא:

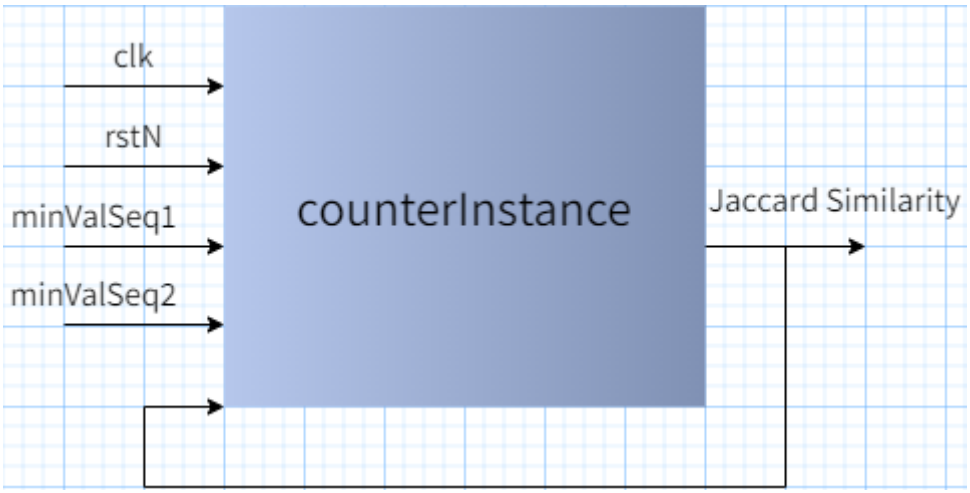
- הערך המינימאלי מבין מערך התוצאות, minVal.

בכל עליית שעון, החל מעליית השעון בפעם השנייה בה המידע תקף, מערך התוצאות מכיל את תוצאות החישוב של הפעלת פונקציית הhash על כל החתימות.
על ידיי מעבר בלולאה על ערכי המערך, מוצאים את המינימום ומעבירים אותו לשלב הבא בצינור.

Counter-Instance

נתאר את מודל ה**CounterInstance**.
מודל זה אחראי על חישוב מקדם ג'קארד כך שבכל עליית שעון, החל מהעלייה השלישית בו המידע תקף בכניסת המודל, מתרחשת בדיקת זהות בין ערכי המינימום מהשלב הקודם עבור הרצפים S_1, S_2.
גם במודל זה הוספנו השהיה על מנת לדגום מידע תקף בלבד ולשמור על נכונות המערכת.

תיאור הכניסות והיציאות של המודל באופן מופשט נתון על ידי:



איור 13
סכמת בלוקים מופשטת של מודל CounterInstance

ניתן לראות שהכניסות הן:

- שעון, **clk**.
- הדק איפוס, **rstN**.
- זוג הערכים המינימליים, **minValSeq1, minValSeq2**.
- מקדם ג'קארד שחושב עד כה.

היציאה היא:

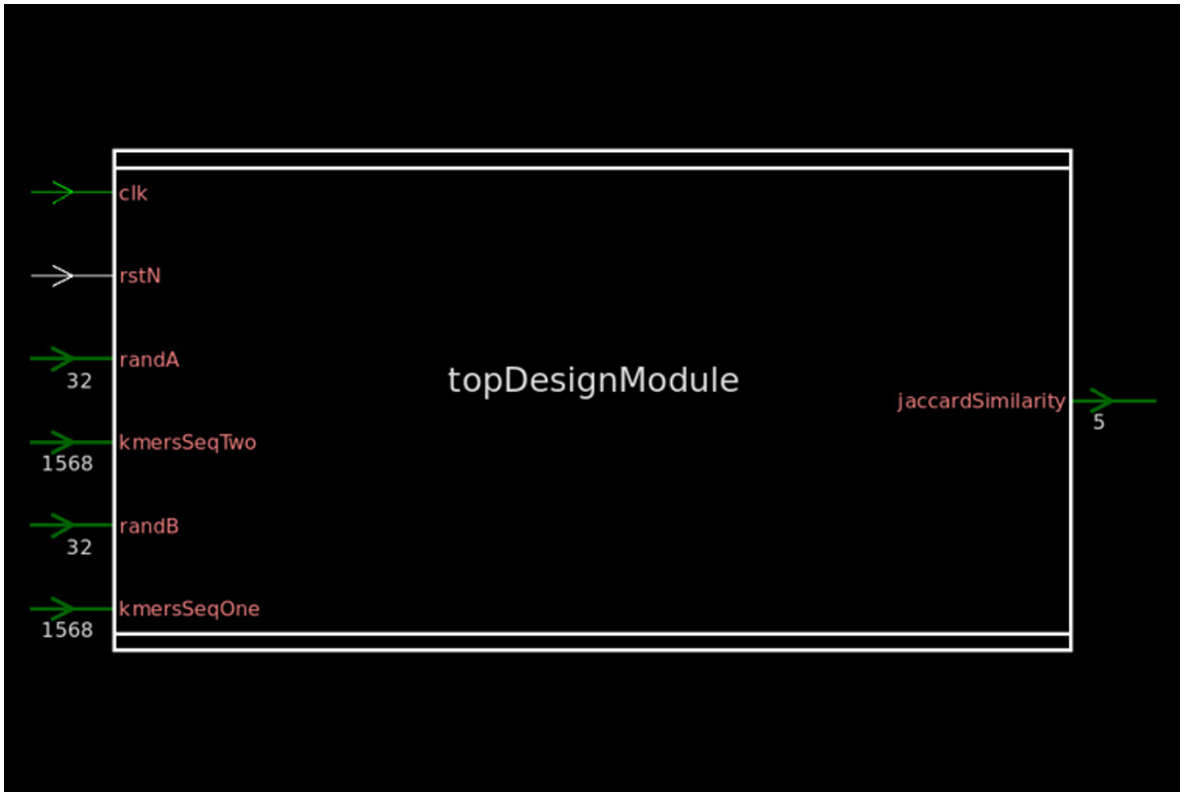
- מקדם ג'קארד, **JaccardSimilarity**.

בכל עליית שעון, החל מעליית השעון בפעם השלישית בה המידע תקף, הכניסות בעלות הערך המינימאלי שהתקבל עבור הפעלת פונקציית ה**hash** על החתימות.
המוצא גדל באחד בכל פעם בו מתקבל כי אכן ערכי המינימום זהים, לבסוף נפלטת תוצאת מקדם ג'קארד הלא מנורמלת.

תיאור סכמתי של כלי הסינתזה

נציג את תיאור מבנה הבלוקים של המודל שתכננו שהתקבל לאחר ביצוע סינתזה על התכנון שלנו.

Top Design



איור 14
סכמת בלוקים לאחר סינתזה של מודל ה$TopDesign$

Hash Module

איור 15
סכמת בלוקים לאחר סינתזה של מודל ה$Hash$ הראשון.

איור 16
סכמת בלוקים לאחר סינתזה של מודל ה$Hash$ השני.

Min Module

איור 17

סכמת בלוקים לאחר סינתזה של מודל הMin הראשון.

Counter Module

איור 19

סכמת בלוקים לאחר סינתזה של מודל ה$Counter$.

דוחות מהסינתזה

Area Report

```
Number of ports:                          27448
Number of nets:                          125379
Number of cells:                         104884
Number of combinational cells:            98927
Number of sequential cells:                 858
Number of macros/black boxes:                 0
Number of buf/inv:                        27276
Number of references:                         5

Combinational area:                151546.000000
Buf/Inv area:                       15458.250000
Noncombinational area:               5866.250000
Macro/Black Box area:                   0.000000
Net Interconnect area:              37179.135665

Total cell area:                   157412.250000
Total area:                        194591.385665
```

Timing Report

```
data required time                                    79.91
data arrival time                                    -79.88
```

איור 21
דו"ח מסלול קריטי

Power Report

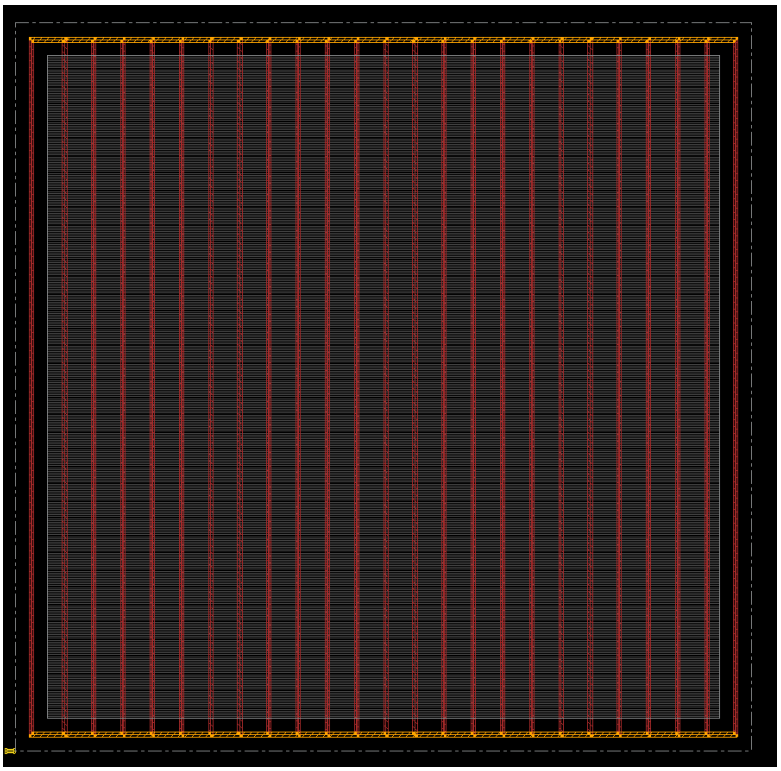
```
Global Operating Voltage = 1.8
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000pf
    Time Units = 1ns
    Dynamic Power Units = 1mW    (derived from V,C,T units)
    Leakage Power Units = 1pW


  Cell Internal Power  =  11.5821 mW   (59%)
  Net Switching Power  =   8.1651 mW   (41%)
                         ---------
Total Dynamic Power    =  19.7473 mW  (100%)

Cell Leakage Power     =   3.8705 uW


                 Internal         Switching           Leakage            Total
Power Group      Power            Power               Power              Power   (   %    )  Attrs
--------------------------------------------------------------------------------------------------
io_pad             0.0000           0.0000            0.0000            0.0000  (   0.00%)
memory             0.0000           0.0000            0.0000            0.0000  (   0.00%)
black_box          0.0000           0.0000            0.0000            0.0000  (   0.00%)
clock_network      0.0000           0.0000            0.0000            0.0000  (   0.00%)
register           0.9548           0.1235        1.3526e+05            1.0784  (   5.46%)
sequential         0.0000           0.0000            0.0000            0.0000  (   0.00%)
combinational     10.6272           8.0416        3.7352e+06           18.6726  (  94.54%)
--------------------------------------------------------------------------------------------------
Total             11.5819 mW        8.1651 mW     3.8705e+06 pW        19.7510 mW
```

איור 22

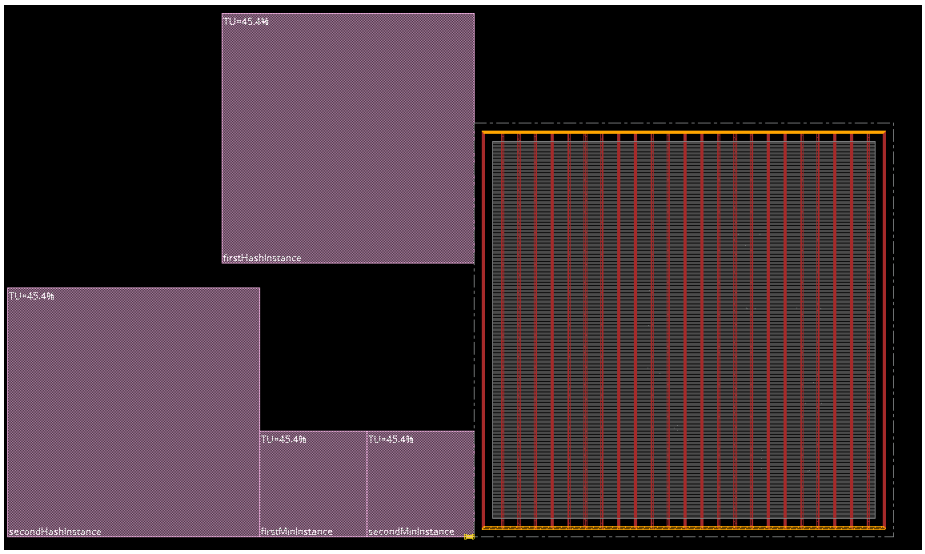
דו"ח צריכת הספק

Layout

Adding Power

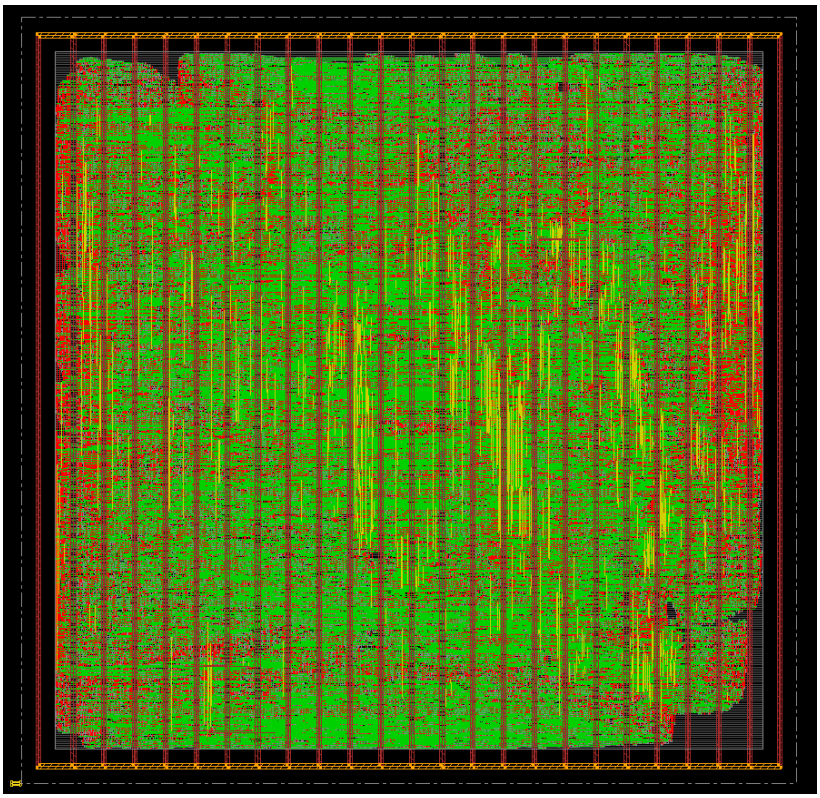


איור 23
הגדרת רשת האספקה.
physical view



איור 24
הגדרת רשת האספקה.
Floorplan view

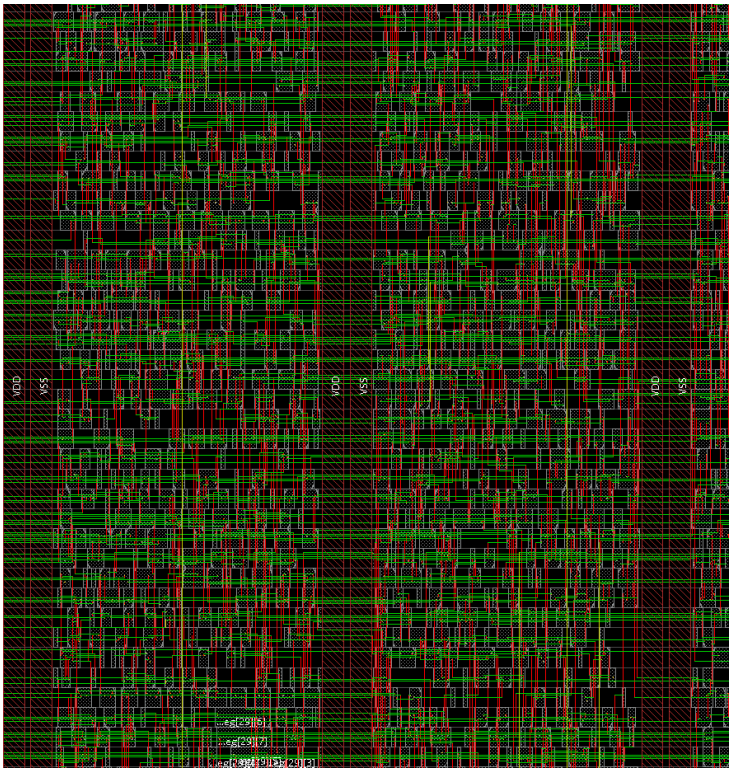
Placing Cells



איור 25

מיקום התאים.

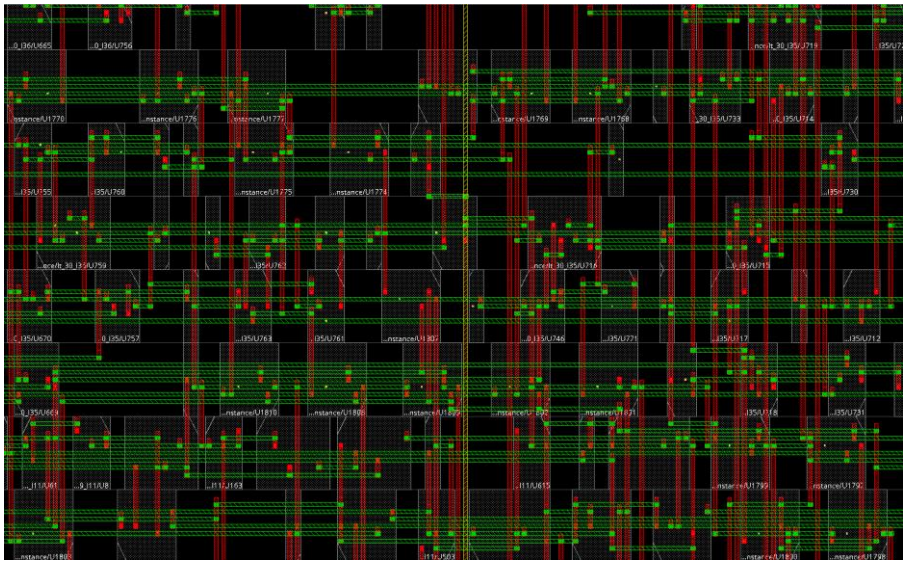
Physical view



איור 26

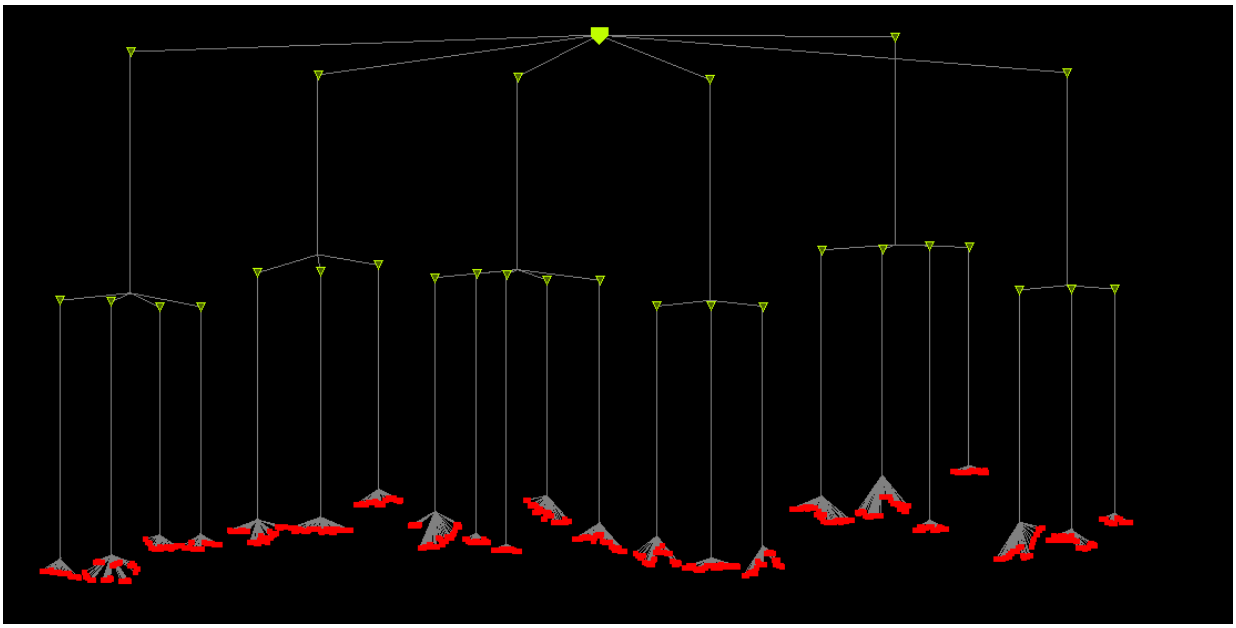
זום על מיקום התאים.

Physical view

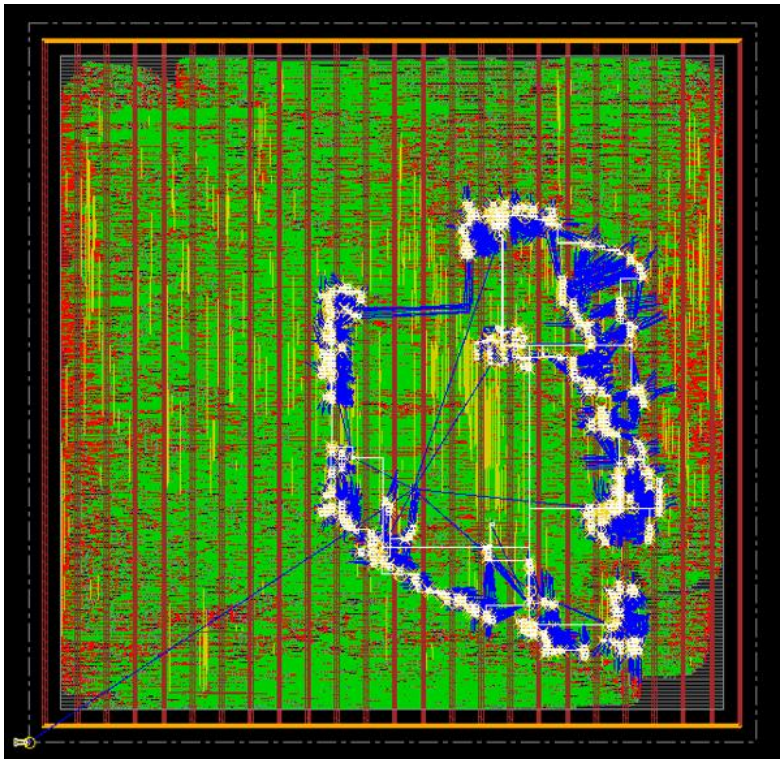


איור 27
זום נוסף על מיקום התאים.
Physical view

Clock Tree Synthesis



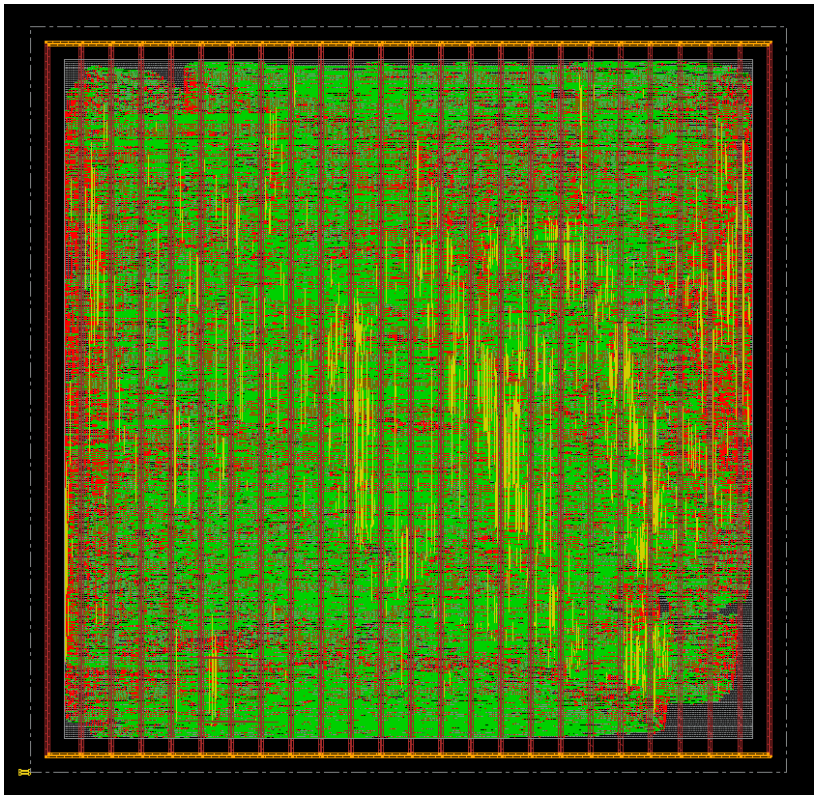
איור 28
עץ שעון מאוזן



איור 29

Physical View לאחר הוספת עץ שעון מאוזן.

Fill Spaces



איור 30

מילוי רווחים בתאי המילוי.

Amoeba View

איור 31
תצוגת Amoeba של הרכיב.

סימולטור תוכנתי

בנינו סימולטור תוכנתי אשר מומש ב$Python$ על מנת לקבל השוואה לתוצאות הסימולציה החומרתית.
הסימולטור הנ"ל מממש את אלגוריתם $MHAP$ אותו הצגנו בסביבת עבודה אשר נתמכת על ידי שפת תכנות עילית.
בנוסף להשוואת התוצאות בין שני הסימולטורים, הסימולטור התוכנתי מקל בכתיבת $TestBench$.

נסביר את השימוש בסימולטור התוכנתי.

1. תחילה יש להגדיר את הקבועים הרצויים להמשך פעולות הסימולטור.
   ניתן להגדיר את הקבועים הבאים:
   - אורך ה$kmer$, K.
   - מקדם הנרמול עבור פונקציית ה$hash$, N.
   - הגדרת הייצוג הדצימלי של כל נוקלאוטיד, מוגדר על ידי מילון.
   - הגדרת הייצוג הבינארי של כל ספרה, מוגדר על ידי מילון.

```
# Defines.
K = 16  # Length of kmer.
N = 255 # A constant used in hash function.
myDNAdictionary = {'A': '0', 'T': '1', 'G': '2', 'C': '3'}
myBinarydictionary = {'0': '00', '1': '01', '2': '10', '3': '11'}
```

איור 32
הגדרת קבועים בסימולטור תוכנתי.

2. יש להגדיר כמחרוזות את הרצפים הגנטיים שעליהם מעוניינים לבצע את החישוב.

```
seqOneDna = "GATCGGATTGCCTAGTGATAAGTCGATGAATGCCGATAGATTTGAGCGGCAAGTTGCGTAGTCG"
seqTwoDna = "GATCGATGTCGAGTCCAGTCCCCCCCCCCCCGATGATGACGGTTGGACCGCCCCCCCCCCCAGTTG"
```

איור 33
הגדרת הרצפים הגנטיים.

3. יש להפעיל את הפונקציה אשר מבצעת המרה של הרצפים הגנטיים לייצוג דצימלי ספרה-ספרה (כפי שהסברנו בעמוד 5), לאחר מכן להפעיל את הפונקציה אשר מוצאת את סט תתי – הרצפים של הרצף הגנטי ולבסוף להמיר כל תת – רצף זה לחתימה בינארית באמצעות פונקציה ייעודית.

```
seqOne = shingleToBinaryToDecimal(shingleForPY(dnaToDecimal(seqOneDna)))
seqTwo = shingleToBinaryToDecimal(shingleForPY(dnaToDecimal(seqTwoDna)))
```

איור 34
ייצוג החתימה של כל $kmer$.

4. יש להגדיר את מערך המספרים הרנדומליים אשר לפיהם מתבצע החישוב של פונקציות ה$hash$.

```
randA = [10323, 2324, 358771, 409712, 94390, 2229481, 123, 1441]
randB = [10091, 1, 233, 76423, 4232409, 57554, 2231130, 1091]
```

איור 35
הגדרת המספרים הרנדומליים.

5. יש להריץ את הסימולציה, לטרמינל תודפס התוצאה של מקדם ג'קארד.

סימולטור חומרתי

רשמנו מספר פונקציות אשר מקלות על מימוש הטסטבנצ'ים והכתיבה החומרתית.

- הפונקציה ***shingleForSV*** מפרקת את ה***kmer*** לתצוגה בינארית ומדפיסה את מערך ה***kmers*** בפורמט המתאים לקוד ה***systemverilog***.
  הפונקציה מקבלת את הרצף הגנטי בייצוג נוקלאוטידי ומשתנה בוליאני אשר מציין האם ההשמה מתבצעת לרצף גנטי מספר 1 או 2.

```python
def shingleForSV(dnaSeq, seqOneFlag):
    shingle_set = []
    for i in range(len(dnaSeq) - K + 1):
        shingle_set.append(dnaSeq[i:i + K])
    if (seqOneFlag):
        for idx, elem in enumerate(shingle_set):
            newElem = invertToBinary(elem)
            print("     kmersSeqOne[" + str(idx) + "]= 32'b" + str(newElem) + ";")
    else:
        for idx, elem in enumerate(shingle_set):
            newElem = invertToBinary(elem)
            print("     kmersSeqTwo[" + str(idx) + "]= 32'b" + str(newElem) + ";")
```

```
Run: main
C:\Users\Gal\Desktop\projectB\PYTHON\venv\Scripts\python.exe "C:/Users/Gal/Desktop/projectB/תוצאות סופיות/codePY/PYTHON/main.py"
     kmersSeqOne[0]= 32'b10000111101000010110111101001001;
     kmersSeqOne[1]= 32'b00011110100001011011110100100110;
     kmersSeqOne[2]= 32'b01111010000101101111010010011000;
     kmersSeqOne[3]= 32'b11101000010110111101001001100001;
     kmersSeqOne[4]= 32'b10100001011011110100100110000100;
     kmersSeqOne[5]= 32'b10000101101111010010011000010000;
     kmersSeqOne[6]= 32'b00010110111101001001100001000010;
     kmersSeqOne[7]= 32'b01011011110100100110000100001001;
     kmersSeqOne[8]= 32'b01101111010010011000010000100111;
```

איור 36

שימוש בפונקציה אשר מדפיסה פורמט מוכן של הkmers ל*testbench*.

- הפונקציה ***printRandomForSV*** מבצעת הדפסה של המספרים הרנדומליים לפורמט מתאים לקוד ה***systemverilog***.

```python
def printRandomForSV(randA, randB):
    print("     randA = ", randA, ";")
    print("     randB = ", randB, ";")
for i in range(8):
    printRandomForSV(randA[i], randB[i])
```

```
Run: main
C:\Users\Gal\Desktop\projectB\PYTHON\venv\Scripts\python.exe "C:/Users/Gal/Desktop/projectB/תוצאות סופיות/codePY/PYTHON/main.py"
     randA =  10323 ;
     randB =  10091 ;
     randA =  2324 ;
     randB =  1 ;
     randA =  358771 ;
     randB =  233 ;
     randA =  409712 ;
     randB =  76423 ;
     randA =  94390 ;
     randB =  4232409 ;
     randA =  2229481 ;
     randB =  57554 ;
     randA =  123 ;
     randB =  2231130 ;
     randA =  1441 ;
```

איור 37

שימוש בפונקציה אשר מדפיסה פורמט מוכן של המספרים הרנדומליים לשימוש ב*testBench*

סימולציות

כעת נדגים את פעולת האלגוריתם על מספר סימולציות שהתבצעו על המימוש החומרתי שבנינו ונשווה את התוצאות לסימולטור התוכנתי.

הסימולטור החומרתי שלנו מגיע לתוצאה סופית בעליית שעון העשירית.

סימולציה מספר 1 – זהות מוחלטת של הרצפים

בסימולציה זו בדקנו את התוצאה המתקבלת עבור מקדם ג'קארד כאשר הרצפים זהים לחלוטין.

ברור שכיוון שהרצפים זהים לחלוטין, מתקיים כי גם ה$kmers$ ובפרט ה$fingerprints$ שלהם זהים ועל כן נצפה לקבל:

$$\text{JaccardSimilarity} = 8$$

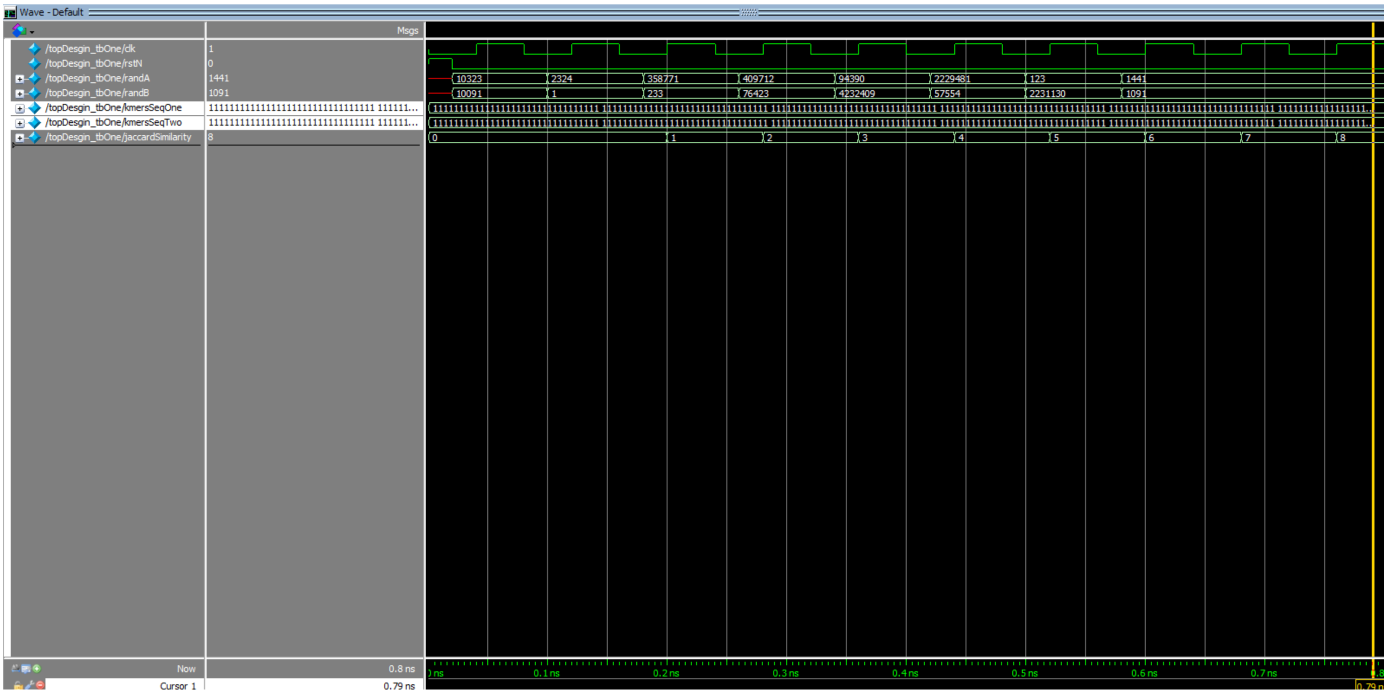
כלומר, מקדם דמיון מקסימלי.

(התוצאה היא 8 ולא 1 שכן ציינו כי לא בצענו נרמול במספר פונקציות ה$hash$)

בחרנו ברצפים הגנטיים הבאים:

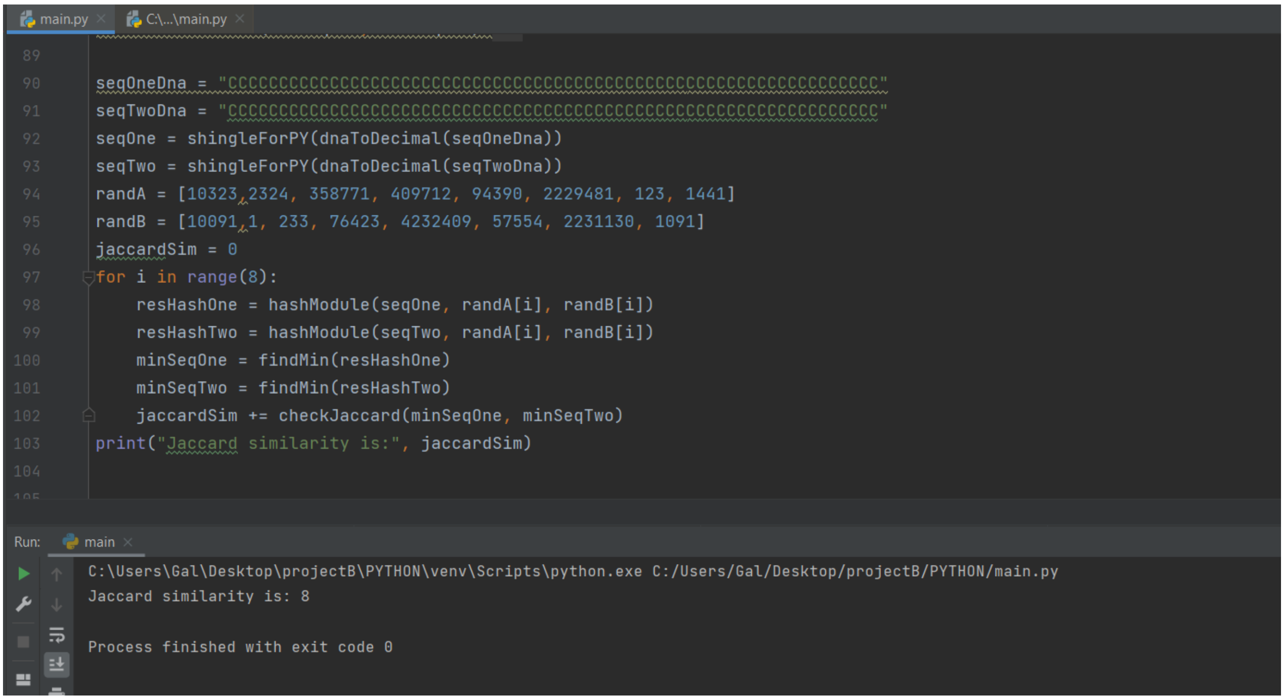
$$S_1 = \text{CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC}$$

$$S_2 = \text{CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC}$$



איור 38

תוצאת הסימולציה החומרתית.



```python
seqOneDna = "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC"
seqTwoDna = "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC"
seqOne = shingleForPY(dnaToDecimal(seqOneDna))
seqTwo = shingleForPY(dnaToDecimal(seqTwoDna))
randA = [10323,2324, 358771, 409712, 94390, 2229481, 123, 1441]
randB = [10091,1, 233, 76423, 4232409, 57554, 2231130, 1091]
jaccardSim = 0
for i in range(8):
    resHashOne = hashModule(seqOne, randA[i], randB[i])
    resHashTwo = hashModule(seqTwo, randA[i], randB[i])
    minSeqOne = findMin(resHashOne)
    minSeqTwo = findMin(resHashTwo)
    jaccardSim += checkJaccard(minSeqOne, minSeqTwo)
print("Jaccard similarity is:", jaccardSim)
```

```
C:\Users\Gal\Desktop\projectB\PYTHON\venv\Scripts\python.exe C:/Users/Gal/Desktop/projectB/PYTHON/main.py
Jaccard similarity is: 8

Process finished with exit code 0
```

איור 39
תוצאת הסימולציה התוכנתית.

אכן קיבלנו בשני המקרים כי מקדם ג'קארד הוא 8, בהתאם לציפיות שלנו.

סימולציה מספר 2 – שוני מוחלט של הרצפים

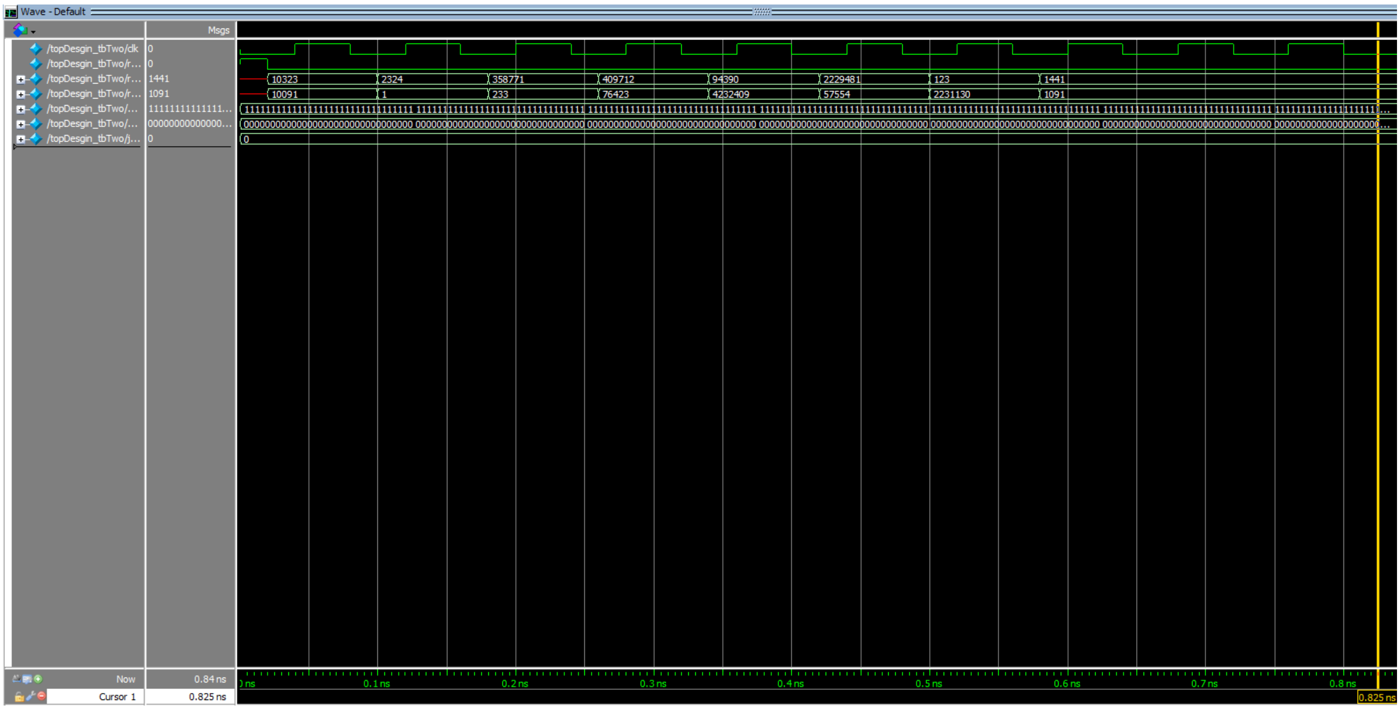
בסימולציה זו בדקנו את התוצאה המתקבלת עבור מקדם ג'קארד כאשר הרצפים שונים לחלוטין.
ברור שכיון שהרצפים שונים לחלוטין, מתקיים כי גם ה$kmers$ ובפרט ה$fingerprints$ שלהם שונים ועל כן נצפה לקבל:

$$\mathrm{JaccardSimilarity} = 0$$

בחרנו בשני הרצפים הגנטיים הבאים:

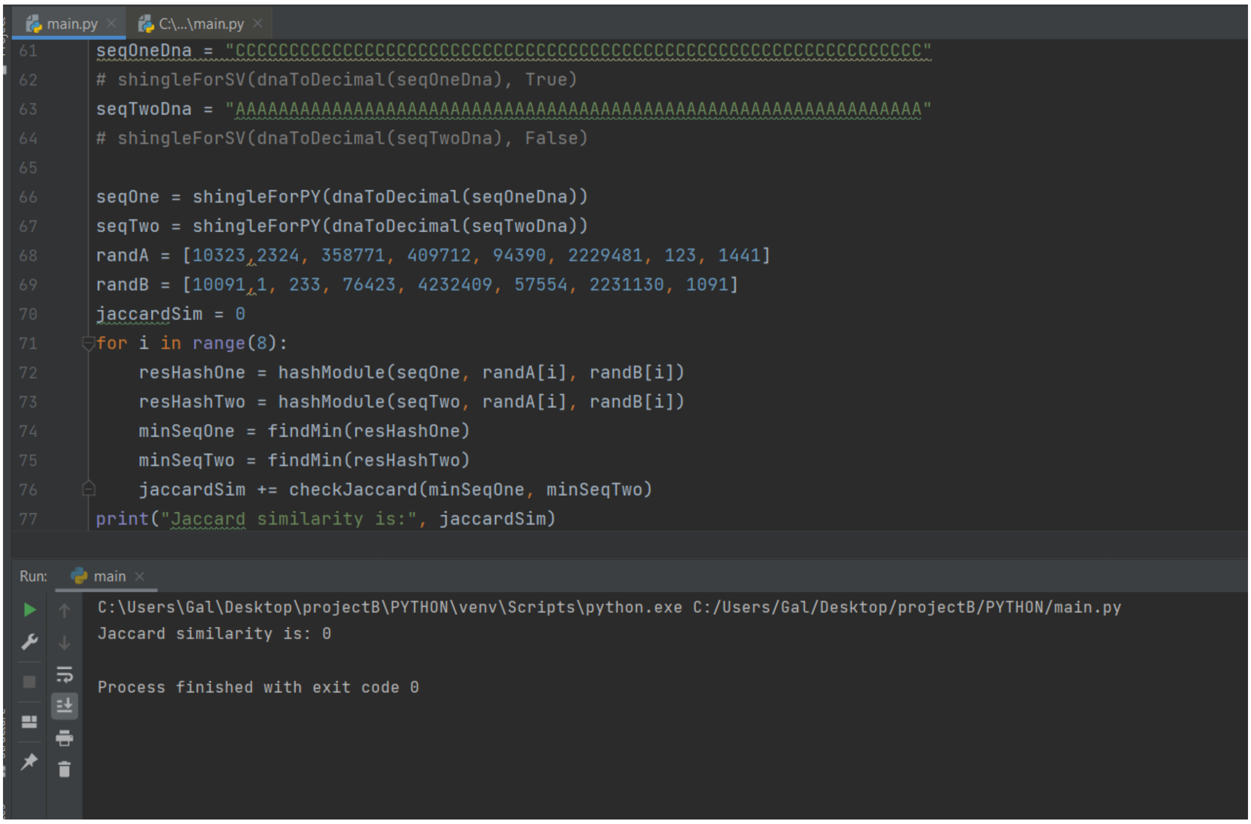
$$S_1 = \mathrm{CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC}$$

$$S_2 = \mathrm{AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA}$$



איור 40

תוצאת הסימולציה החומרתית.



```python
seqOneDna = "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC"
# shingleForSV(dnaToDecimal(seqOneDna), True)
seqTwoDna = "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
# shingleForSV(dnaToDecimal(seqTwoDna), False)

seqOne = shingleForPY(dnaToDecimal(seqOneDna))
seqTwo = shingleForPY(dnaToDecimal(seqTwoDna))
randA = [10323,2324, 358771, 409712, 94390, 2229481, 123, 1441]
randB = [10091,1, 233, 76423, 4232409, 57554, 2231130, 1091]
jaccardSim = 0
for i in range(8):
    resHashOne = hashModule(seqOne, randA[i], randB[i])
    resHashTwo = hashModule(seqTwo, randA[i], randB[i])
    minSeqOne = findMin(resHashOne)
    minSeqTwo = findMin(resHashTwo)
    jaccardSim += checkJaccard(minSeqOne, minSeqTwo)
print("Jaccard similarity is:", jaccardSim)
```

```
C:\Users\Gal\Desktop\projectB\PYTHON\venv\Scripts\python.exe C:/Users/Gal/Desktop/projectB/PYTHON/main.py
Jaccard similarity is: 0

Process finished with exit code 0
```

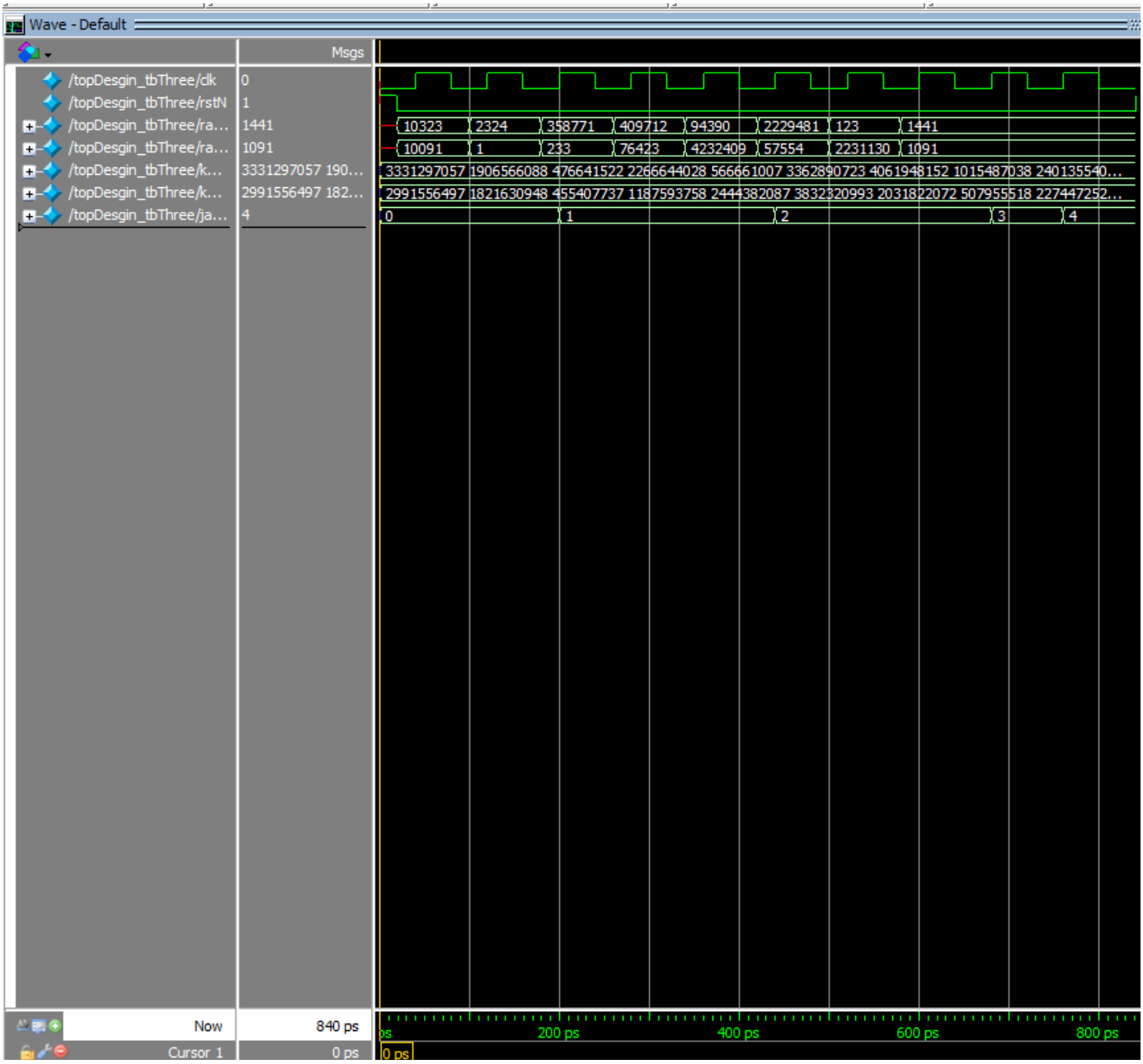
איור 41

תוצאת הסימולציה התוכניתית.

אכן קיבלנו בשני המקרים כי מקדם ג'קארד הוא 0, בהתאם לציפיות שלנו.

סימולציה מספר 3 – רצפים כללים 1

בסימולציה זו בדקנו את התוצאה המתקבלת עבור מקדם ג'קארד כאשר הרצפים בעלי דמיון.
השתמשנו ברצפים גנטיים מורחבים ל64 אותיות מהרצפים שהצגנו בדוגמה בעמוד 5.
עבור הרצפים הנ"ל באורך 14 ו$kmers$ באורך 3 ו4 פונקציות $hash$, מהמאמר שהתבססנו עליו מתקיים כי מקדם הזהות של שני רצפים אלו הוא 2 ללא נרמול.
לכן, נצפה שעבור הגדלה של מספר פונקציות ה$hash$ מקדם הזהות יהיה לפחות 2.



איור 42
תוצאת הסימולציה החומרתי.

```python
seqOneDna = "CATGGACCGACCAGATCATGGACCGACCAGATCATGGACCGACCAGATCATGGACCGACCAGAT"
# shingleForSV(dnaToDecimal(seqOneDna), True)
seqTwoDna = "GCAGTACCGATCGTATGCAGTACCGATCGTATGCAGTACCGATCGTATGCAGTACCGATCGTAT"
# shingleForSV(dnaToDecimal(seqTwoDna), False)
seqOne = shingleToBinaryToDecimal(shingleForPY(dnaToDecimal(seqOneDna)))
seqTwo = shingleToBinaryToDecimal(shingleForPY(dnaToDecimal(seqTwoDna)))
randA = [10323, 2324, 358771, 409712, 94390, 2229481, 123, 1441]
randB = [10091, 1, 233, 76423, 4232409, 57554, 2231130, 1091]
jaccardSim = 0
for i in range(8):
    resHashOne = hashModule(seqOne, randA[i], randB[i])
    resHashTwo = hashModule(seqTwo, randA[i], randB[i])
    minSeqOne = findMin(resHashOne)
    minSeqTwo = findMin(resHashTwo)
    jaccardSim += checkJaccard(minSeqOne, minSeqTwo)
print("Jaccard similarity is:", jaccardSim)
```

```
C:\Users\Gal\Desktop\projectB\PYTHON\venv\Scripts\python.exe C:/Users/Gal/Desktop/projectB/PYTHON/main.py
Jaccard similarity is: 4
```

איור 43

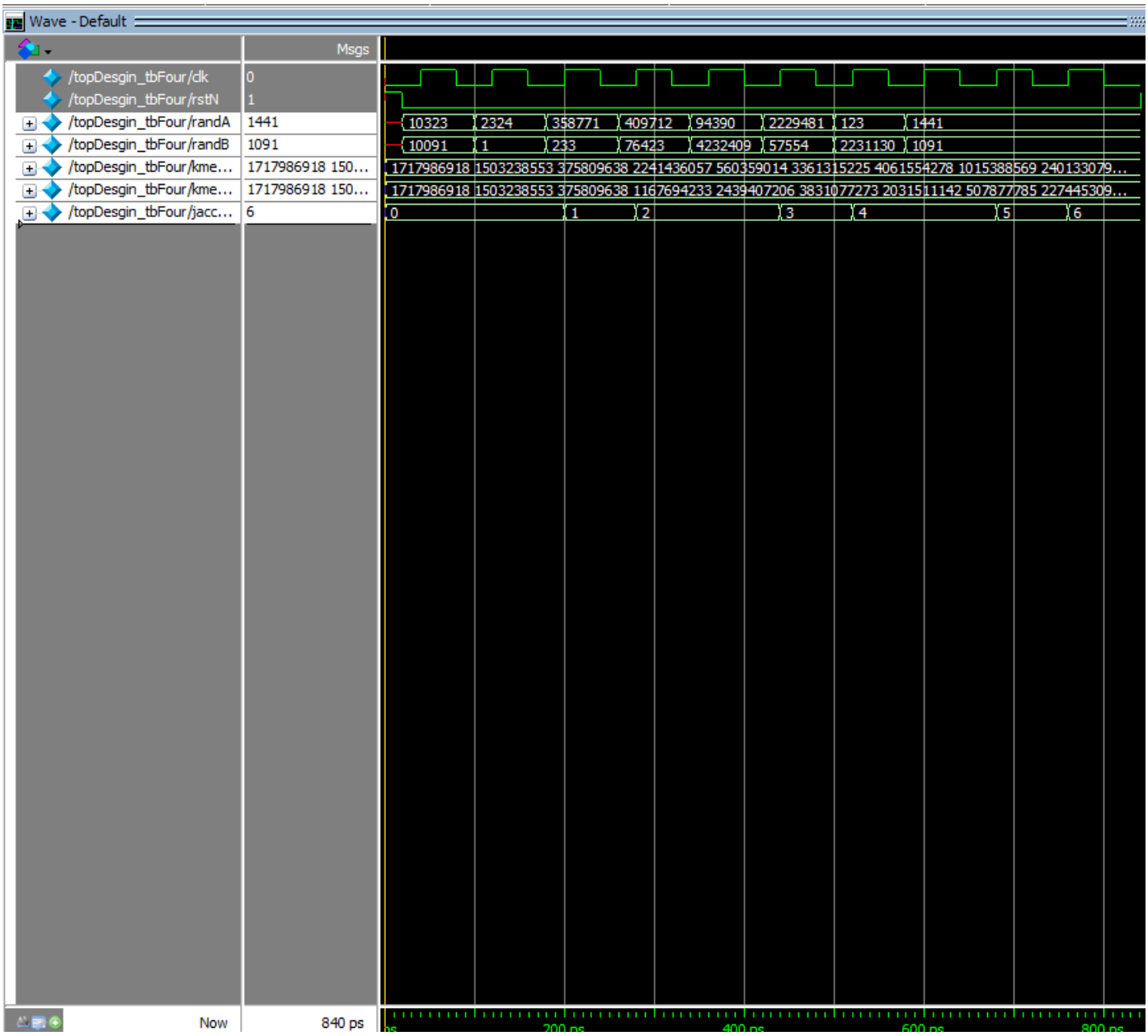
תוצאת הסימולציה התוכנתי.

אכן קיבלנו מקדם זהות זהה עבור שני הסימולציות ובפרט גדול מ2.

ניתן לראות את הרצפים שבחרנו בצורה נוחה בקוד הפיתון.

סימולציה מספר 4 – רצפים כללים 2

בסימולציה זו בדקנו את התוצאה המתקבלת עבור מקדם ג'קארד כאשר הרצפים הוקשו רנדומלית על ידנו כך שהם יהיו באורך 64 אותיות והשתדלנו להתאים בין הלחיצות עבור הרצפים כך שיתקבל מקדם זהות גבוהה.



איור 44

תוצאת הסימולציה החומרתי.

```python
seqOneDna = "CATGGACCGACCAGATCATGGACCGACCAGATGCAGTACCGACCAGATTGTGTGTGTGTGTGTG"
# shingleForSV(dnaToDecimal(seqOneDna), True)
seqTwoDna = "CATGGACCGATCGTATGCAGGACCGATCGTATGCAGTACCGATCGTATTGTGTGTGTGTGTGTG"
# shingleForSV(dnaToDecimal(seqTwoDna), False)
seqOne = shingleToBinaryToDecimal(shingleForPY(dnaToDecimal(seqOneDna)))
seqTwo = shingleToBinaryToDecimal(shingleForPY(dnaToDecimal(seqTwoDna)))
randA = [10323, 2324, 358771, 409712, 94390, 2229481, 123, 1441]
randB = [10091, 1, 233, 76423, 4232409, 57554, 2231130, 1091]
jaccardSim = 0
for i in range(8):
    resHashOne = hashModule(seqOne, randA[i], randB[i])
    resHashTwo = hashModule(seqTwo, randA[i], randB[i])
    minSeqOne = findMin(resHashOne)
    minSeqTwo = findMin(resHashTwo)
    jaccardSim += checkJaccard(minSeqOne, minSeqTwo)
print("Jaccard similarity is:", jaccardSim)
```

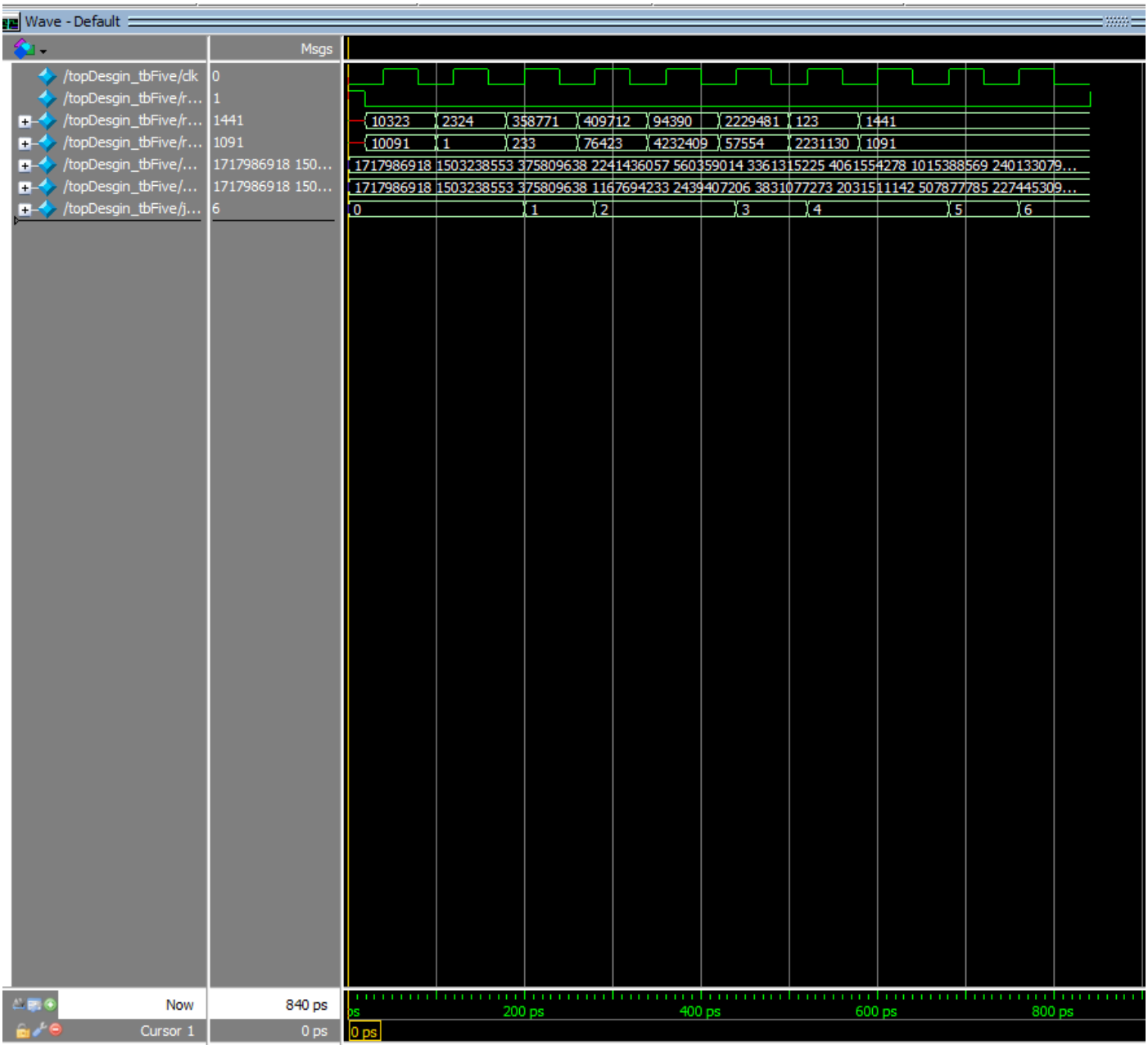
```
C:\Users\Gal\Desktop\projectB\PYTHON\venv\Scripts\python.exe C:/Users/Gal/Desktop/projectB/PYTHON/main.py
Jaccard similarity is: 6

Process finished with exit code 0
```

איור 45
תוצאת הסימולציה התוכנתי.

אכן קיבלנו מקדם זהות זהה עבור שני הסימולציות ובפרט כיוון שציינו כי השתדלנו לבצע הקלדה זהה של הרצפים, קיבלנו מקדם זהות גבוה.
ניתן לראות את הרצפים שבחרנו בצורה נוחה בקוד הפיתון.

סימולציה מספר 5 – רצפים כללים 3

גם בסימולציה זו בדקנו את התוצאה המתקבלת עבור מקדם ג'קארד כאשר הרצפים הוקשו רנדומלית על ידנו כך שהם יהיו באורך 64 אותיות והשתדלנו להתאים בין הלחיצות עבור הרצפים כך שיתקבל מקדם זהות גבוהה.



איור 46
תוצאת הסימולציה החומרתי.

```python
seqOneDna = "GATCGGATTGCCTAGTGATAAGTCGATGAATGCCGATAGATTTGAGCGGCAAGTTGCGTAGTCG"
# shingleForSV(dnaToDecimal(seqOneDna), True)
seqTwoDna = "GATCGATGTCGAGTCCAGTAGTCGTAGTACGATGATGACGGTTGGACCGATTACGTAGTAGTTG"
# shingleForSV(dnaToDecimal(seqTwoDna), False)
seqOne = shingleToBinaryToDecimal(shingleForPY(dnaToDecimal(seqOneDna)))
seqTwo = shingleToBinaryToDecimal(shingleForPY(dnaToDecimal(seqTwoDna)))
randA = [10323, 2324, 358771, 409712, 94390, 2229481, 123, 1441]
randB = [10091, 1, 233, 76423, 4232409, 57554, 2231130, 1091]
jaccardSim = 0
for i in range(8):
    resHashOne = hashModule(seqOne, randA[i], randB[i])
    resHashTwo = hashModule(seqTwo, randA[i], randB[i])
    minSeqOne = findMin(resHashOne)
    minSeqTwo = findMin(resHashTwo)
    jaccardSim += checkJaccard(minSeqOne, minSeqTwo)
print("Jaccard similarity is:", jaccardSim)
```

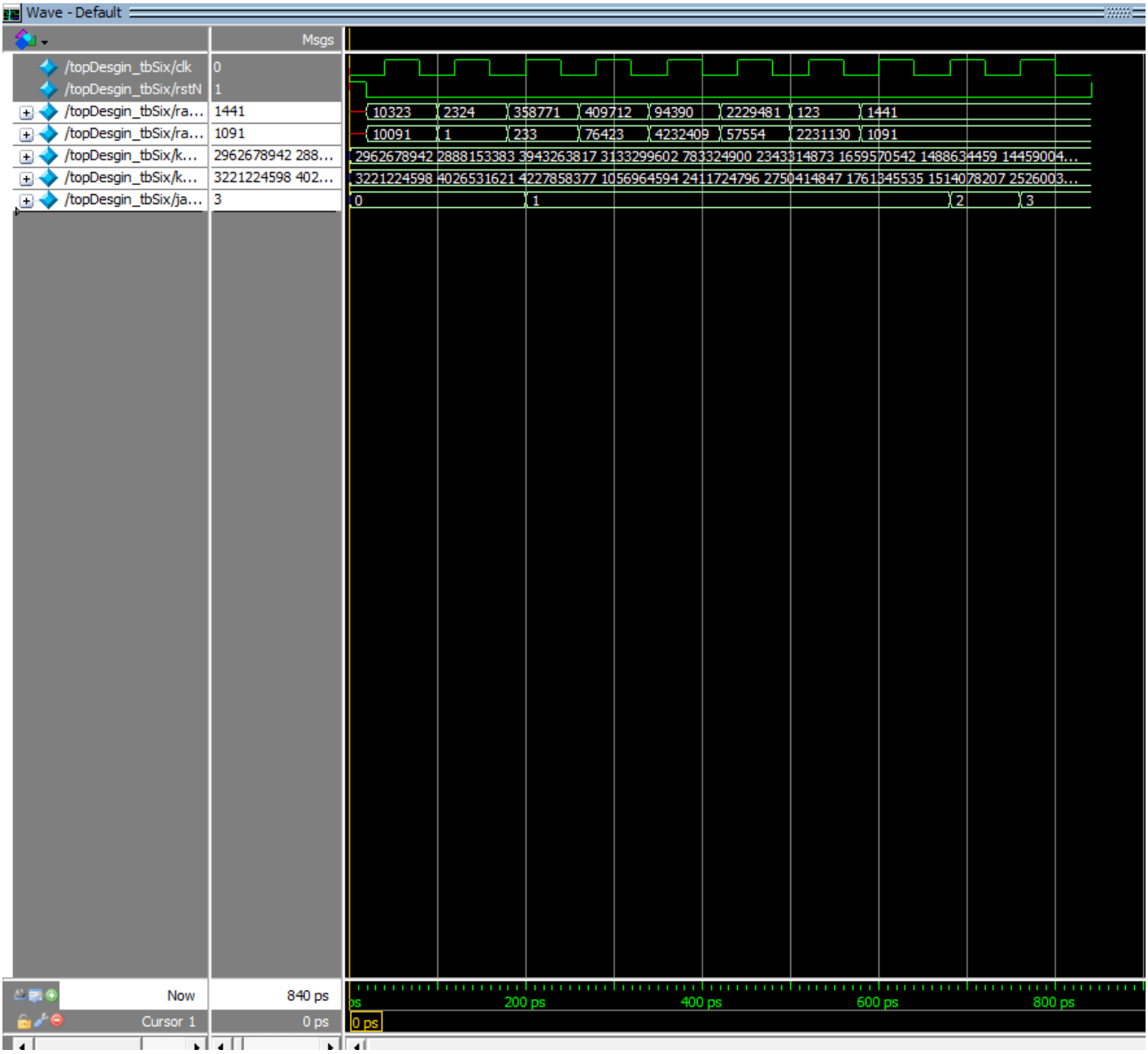
```
C:\Users\Gal\Desktop\projectB\PYTHON\venv\Scripts\python.exe C:/Users/Gal/Desktop/projectB/PYTHON/main.py
Jaccard similarity is: 6

Process finished with exit code 0
```

איור 47
תוצאת הסימולציה התוכנתי.

אכן קיבלנו מקדם זהות זהה עבור שני הסימולציות ובפרט כיוון שציינו כי השתדלנו לבצע הקלדה זהה של הרצפים, קיבלנו מקדם זהות גבוה.
ניתן לראות את הרצפים שבחרנו בצורה נוחה בקוד הפיתון.

סימולציה מספר 6 – רצפים כללים 4

בסימולציה זו בדקנו את התוצאה המתקבלת עבור מקדם ג'קארד כאשר הרצפים הוקשו רנדומלית על ידנו כך שהם יהיו באורך 64 אותיות והשתדלנו לא להתאים בין הלחיצות עבור הרצפים כך שיתקבל מקדם זהות נמוך.



איור 48
תוצאת הסימולציה התוכנתי.

```python
    return newShingle

seqOneDna = "GATCGGATTGCCTAGTGATAAGTCGATGAATGCCGATAGATTTGAGCGGCAAGTTGCGTAGTCG"
# shingleForSV(dnaToDecimal(seqOneDna), True)
seqTwoDna = "GATCGATGTCGAGTCCAGTCCCCCCCCCCCGATGATGACGGTTGGACCGCCCCCCCCCCAGTTG"
# shingleForSV(dnaToDecimal(seqTwoDna), False)
seqOne = shingleToBinaryToDecimal(shingleForPY(dnaToDecimal(seqOneDna)))
seqTwo = shingleToBinaryToDecimal(shingleForPY(dnaToDecimal(seqTwoDna)))
randA = [10323, 2324, 358771, 409712, 94390, 2229481, 123, 1441]
randB = [10091, 1, 233, 76423, 4232409, 57554, 2231130, 1091]
jaccardSim = 0
for i in range(8):
    resHashOne = hashModule(seqOne, randA[i], randB[i])
    resHashTwo = hashModule(seqTwo, randA[i], randB[i])
    minSeqOne = findMin(resHashOne)
    minSeqTwo = findMin(resHashTwo)
    jaccardSim += checkJaccard(minSeqOne, minSeqTwo)
print("Jaccard similarity is:", jaccardSim)
```

```
Run: main
C:\Users\Gal\Desktop\projectB\PYTHON\venv\Scripts\python.exe C:/Users/Gal/Desktop/projectB/PYTHON/main.py
Jaccard similarity is: 3

Process finished with exit code 0
```

איור 49
תוצאת הסימולציה התוכנתי.

אכן קיבלנו מקדם זהות זהה עבור שני הסימולציות ובפרט כיוון שציינו כי השתדלנו לבצע הקלדה שונה של רצפים, קיבלנו מקדם זהות נמוך, שכן מידת הדמיון בין הרצפים נמוכה.
ניתן לראות את הרצפים שבחרנו בצורה נוחה בקוד הפיתון.

סיכום ומסקנות

בפרויקט זה חקרנו את אלגוריתם $MHAP$ לצורך שיפור ביצועי הרכבת גנום בשיטת $de\ novo$ שכאמור בעל עלות חישובית גבוהה.

למדנו את האלגוריתם ויצרנו ייצוג מספרי של רצף גנטי אשר מטיב עם החישובים שלנו ועומד באילוצי אורך של הרכיב שלנו.
הצלחנו לתכנן ארכיטקטורה אשר מוכיחה את יעילות ומהירות האלגוריתם לצורך זיהוי $reads$ והרכבתם במידה והם זהים.

בתכנון שלנו נעזרנו בתכנון אשר משלב היררכיות לצורך פירוק הארכיטקטורה לחתיכות כך שכל אחת אחראית על תפקיד משלה, בכך יצרנו מנגנון מצונר אשר בו כל שלב מסתמך על השלב הקודם.
בנוסף, בנינו סימולטור תוכנתי אשר מהווה אינדיקציה לבדיקה מהירה של תוצאות הסימולטור החומרתי וגילינו התאמה מלאה עבור הסימולציות שבצענו.
יתרה מזאת, הסימולטור התוכנתי כולל בתוכו ממשק נוח לכתיבה קלה של $testbench$ לצורך בדיקת הרכיב החומרתי עבור בדיקות עתידיות, במידת הצורך.

ניתן לראות כי הביצועים המהירים שהשגנו באים על חשבון שטח הרכיב, מדובר בתכנון עם המון כניסות עבור תתי הרצפים ותוצאות החישוב המתמטיות שמועברות בין השלבים השונים.

רפרנס

1. https://www.nature.com/articles/nbt.3238
2. https://en.wikipedia.org/wiki/MinHash
3. https://www.youtube.com/watch?v=5wvGapmA5zM&t=96s&ab_channel=BioinformaticsDotCa
4. https://www.youtube.com/watch?v=l0lZCSU-Gyo&t=1080s&ab_channel=ISCB

תודות

בהזדמנות זו נרצה להודות למנחה לאוניד יביץ ולאחראי המעבדה ל***VLSI*** גואל סמואל על העזרה, ההכוונה וההנחיה לאורך הפרויקט וההזדמנות להכיר ולהתמודד עם תחום חדש.