

Architectural Simulator for Object Oriented Processor

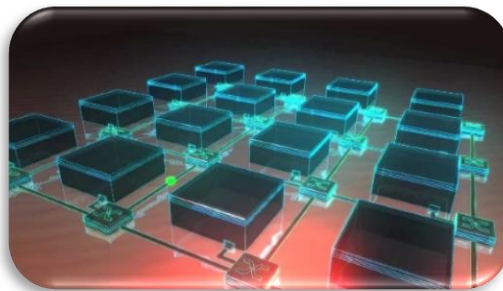
Technion – Israel Institute of Technology

גדיר עואיסה 316571934

אזדיהאר גיוסי 313380164

שם המנחה: אמנון סטאניסלאבסקי

2021/2022



תוכן עניינים

3	הקדמה
4	רקע טכני
5	תיאור כללי של המערכת
6	המטרה של הפרוייקט
7	מבני נתונים
10	אלגוריתם מימוש הסימולטור
11	מפת החום
13	טסטים לבדיקה
16	הסבר וסיכום
17	אתגרים
18	Simulator User Guide
22	מקורות

הקדמה

בחוברת הזו מתואר מימוש של סימולטור למערכת דמויית NOC (Network on a chip) שהיא מורכבת מיחידות וראוטרים המהווים תחנות בין כל יחידה ויחידה וגם מתואר את החיבור ביניהם על מנת שיתקיים התנאי של NOC שבו יש מספר ריבועי של ראוטרים וכל ראوتر מחובר לכל היותר ליחידה אחת.

המשתמש יגדיר את המערכת שכוללת : מספר הרכיבים והחיבור ביניהם לפי תנאים מסויימים שיוסברו בהמשך, ואחרי הסימולציה יוצרו מפות חום שבהם יופיעו את צווארי הבקבוק של הביצועים.

בהתחלה יוסברו את המושגים החשובים שלפיהם תוגדר המערכת, אחר כך יוגדר האלגוריתם שבונה את המערכת ואת מהלך הסימולציה, מה קורה בכל שלב ושלב ואיך המערכת תעבוד.

בסוף תוגדר הנוסחא של חישוב מפות החום וסיכום של התוצאות הסופיות והאם הם תואמים למה שציפינו מקודם.

הפרוייקט נלקח מפקולטה להנדסת חשמל עם הנחיית אמנון סטאניסלאבסקי.



רקע טכני

- 1- **VLSI : Very Large Scale Integration** , הוא תהליך של שילוב מספר רב של רכיבים אלקטרוניים ממוזערים ליצירת מעגל משולב של רכיבים רבים – בעיקר טרנזיסטורים על גבי שבב יחיד.
- 2- **OOPc Simulator**: מעבד OOPc מטרתו לבטל את הצורך במערכת הפעלה על ידי מתן תשתית לניהול תהליכים וזיכרון ללא מערכת הפעלה כזו. היא עושה זאת על ידי הטמעת הקצאה דינמית של זיכרון בחומרה, הוספת מספר הוראות למערך ההוראות, הוספת מצב כתובת חדש והוספת מספר מודולי חומרה ליצירה וניהול של חוטים.
- 3- **NOC**: קיצור של Network on a chip, רשתות-על-שבב מהוות טכנולוגיה מתפתחת למערכות-על-שבב, שיכולה להפיק תועלת משמעותית מהטכניקות של ארכיטקטורות מערכות רשת, ליתר דיוק מהחלפת ארכיטקטורות. שיטות אדריכלות ותכנון מעגלים הן קריטיות בתכנון ובהערכה של ארכיטקטורות NoC, בהתחשב בתלות החזקה של NoCs בטכנולוגיית יישום ובדרישות של ארכיטקטורות NoC לביצועים גבוהים בנוסף לצריכת חשמל נמוכה.
- 4- **SOC : System on a Chip**, מערכת על שבב היא מעגל משולב (המכונה גם "שבב") המשלב את כל או רוב הרכיבים של מחשב או אחר מערכת אלקטרונית. רכיבים אלה כוללים כמעט תמיד יחידת עיבוד מרכזית (CPU), זיכרון, יציאות קלט/פלט ואחסון משני, לרוב לצד רכיבים אחרים כגון מודמי רדיו ויחידת עיבוד גרפית (GPU) - הכל על מצע בודד או שבב אחד [הוא עשוי להכיל פונקציות עיבוד אותות דיגיטליות, אנלוגיות, מעורבות ולעיתים קרובות בתדר רדיו (אחרת הוא נחשב רק למעבד יישומים).
- 5- **Unit**: יחידה במערכת שהיא קופסא שחורה ויכולה להיות מעבד, התקן קלט פלט, זיכרון cache וכו'.
- 6- **Router**: התקן רשת המעביר חבילות נתונים בין רשתות מחשבים. בפרוייקט שלנו הוא תחנה בין כל יחידה ויחידה שמקבל ומעביר חבילות.
- 7- **X-Y algorithm**: האלגוריתם שבו בחרנו את המסלול של העברת החבילות בין רכיבי המערכת השונים. כך שבהתחלה החבילה תעבור על כל קורדינטות X של המסלול עד שתגיע ליעד ואחר כך על קורדינטות Y.

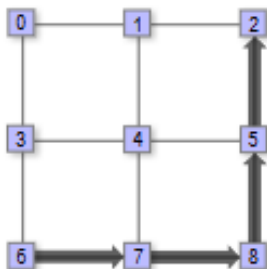
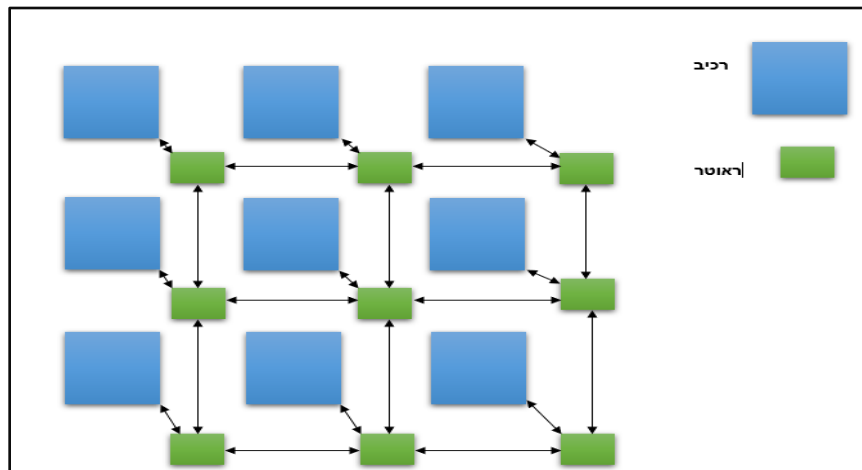


Figure 1: Algorithm XY example, a packet sent from router 6 to router 2

תיאור כללי של המערכת



המערכת OOPc בנויה מחיבור רואטרים ויחידות Units, ניתן לראות דוגמה למערכת ב Figure2.

Figure 2: example of an OOPc system topology

רכיב (יחידה - UNIT) במערכת הוא חומרה אשר מקבלת/שולחת חבילות לרכיבים אחרים בתוך המערכת (למשל מעבד), לכל רכיב מותאם ראוטר אשר דרכו עובר המידע, הרכיבים מחוברים ישירות לראוטרים והראוטרים מחוברים עם ראוטרים אחרים כך שחבילה עוברת בין שני רכיבים שונים דרך לפחות שני ראוטרים, דבר שתלוי בחיבור של המערכת ובמסלול שנבחר להעברת החבילה.

לא חשוב המימוש הפנימי של הרכיב עצמו במקרה הזה, הביצועים של המערכת יחושבו לפי קריטריונים שתלויים בכמות החבילות המועברות בין רכיבים שונים ולא בתוך הרכיב עצמות, לכן לאורך כל הסימולציה ניתן להסתכל עליו כקופסה שחורה, ועל זה לא משנה מבחינתנו אם היחידה היא מעבד, התקן קלט/פלט או זירון קאש וכולי.

רכיב הרואטר אחראי על העברת החבילות לunit היעד, אם החבילה אינה מיועדת לרכיב ה unit שמחובר לראוטר תועבר החבילה לראוטר אחד, שמחובר לראוטר המטפל בחבילה, על הסלול לunit היעד.

המבנה הכללי ואופן החיבור בין רכיבים שונים במערכת יוגדר על ידי המשתמש, בנוסף המשתמש גם כן אחראי להגדיר את כמות המידע המועברת על כל מסלול בין שתי יחידות unit במערכת.

המטרה של הפרויקט

כמעט בכל המעבדים המסחריים המודרניים - מערכת הפעלה משמשת למעקב אחר השרשרת השונים, הקצאת זיכרון, זיכרון וירטואלי ועוד. מערכת ההפעלה היא שכבת תוכנה בין החומרה לתוכנת המשתמש והיא בדרך כלל מוסיפה כוח ועיכוב למערכת.

מטרת הפרויקט היא לדמות את ההשפעה של שינוי מספר רכיבי ארכיטקטורה על הביצועים הכוללים של המעבד. פרמטרים כאלה כוללים: כמות הרכיבים, כמות החוטים בו-זמנית שיכולה לרוץ במערכת, הקצב של קבלת ושליחת חבילות בין רכיבים שונים במערכת, צריך לפתח מערכת שבה ניתן לשנות את הפרמטרים הללו ולאחר מכן לדמות אותם. יש ליצור מפת חום כדי למצוא את צווארי הבקבוק בביצועים.

הציפיות שלנו:

- 1- שינוי מספר רכיבים במערכת ישפיע על מספר החבילות הכולל שעובר במערכת.
- 2- שינוי מספר החוטים ישפיע על מהירות העברת החבילות במערכת כך שכל שמספר החוטים יגדל, מספר החבילות שיגיעו ליעד גם כן יגדל.
- 3- שינוי קצב שליחת חבילות בין כל שני רכיבים ישפיע על מספר החבילות שמתקבלות בסוף סימולציה רק על שני הרכיבים.
- 4- מספר מחזורי השעון של משך הסימולציה ישפיע על העומס של המערכת, כך שכל שהמערכת רצה יותר זמן, מספר החבילות הנתקעות בכל ראוטר יגדל יותר.

מבני נתונים

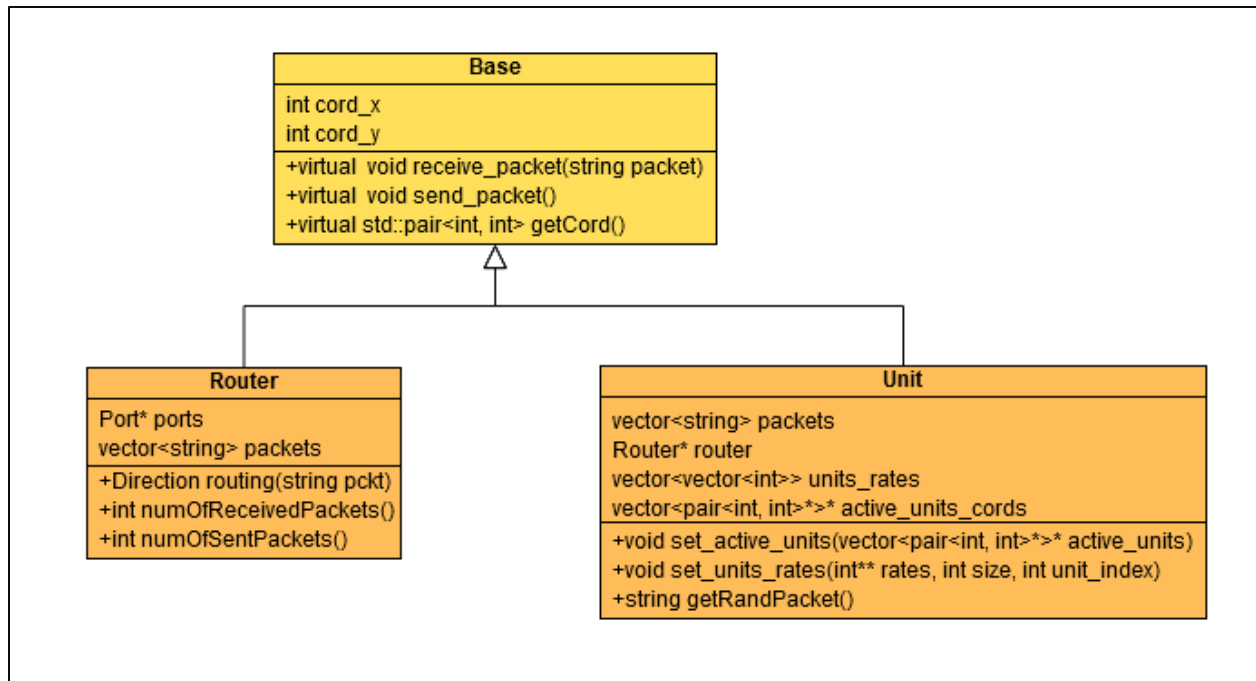


Figure 3: Router and Unit classes

1- **Class Router** : מחלקה שמתארת ראוטר אשר יהיה מחובר לרכיבים במערכת.

המחלקה תכיל:

- המזהה של הראוטר
 - רשימת החיבורים שלו עם רכיבים שונים במערכת
 - רשימות של חבילות על כל קו חיבור של הראוטר- נשים לב שראוטר יכול לבצע פעולה אחת בכל מחזור שעון, ולכן יכול להיות שיש חבילות שממתינות להעברה דרכו.
 - לכל ראוטר כזה יש לכל היותר 5 חיבורים, 4 מהם חיבורים עם ראوترים אחרים לפי ארבע הכיוונים (תלוי במיקום שלו על הטופולוגיה). והחיבור החמישי הוא בין ראוטר ליחידה, החיבור הזה הוא תלוי בבחירת המשתמש ובמספר היחידות/רכיבים Units במערכת.
- הסבר על הפונקציות העיקריות של המבנה:

- `Direction routing(string pkt)`: הפונקציה מקבלת את החבילה ולפי יעד החבילה מחזירה element מסוג `Direction`, הערכים האפשריים ל element הם: `{ up, down, left, right, unit }`.
- נשתמש בפונקציה על מנת למצוא את הראוטר על מסלול בין שתי יחידות במערכת שנשלח אליו את החבילה אם החבילה אינה מיועדת ל unit של הראוטר.
- `int numReceivedPackets()`: מחזירה את מספר החבילות שקיבל הראוטר.

- `int numOfSentPackets()` : מחזירה את מספר החבילות ששלח לראוטר ליעד.
- `void receive_packet(string packet)` : הפונקציה מקבלת את החבילה ושומרת אותה בוקטור של חבילות.
- `void send_packet()` : הפונקציה שולפת חבילה מהוקטור של חבילות, עובדים לפי FIFO, קוראת לפונקציית routing ומעבירה את החבילה או לרכיב המתאים.
- הפונקציות `numOfSentPackets` ו- `numOfReceivedPackets` משתמשים בהן בחישובים הקשורים למפת חום, לחישוב העומס על הראוטר.

2- **Class Unit** : מחלקה שמתארת רכיב במערכת שאינו ראוטר. כל רכיב כזה מחובר לראוטר אחד ויחיד. המחלקה מכילה :

- המזהה של הרכיב
- המזהה הראוטר שמחובר אליו
- רשימה לחבילות שממתינות לביצוע על ידי רכיב זה - נשים לב שרק חבילה אחת ניתנת לביצוע בכל מחזור שעון.
- הסבר על הפונקציות העיקריות של המבנה :

- `set_active_units(vector<pair<int, int>*> active_units)` : הפונקציה מקבלת את הקורדינאטות של כל ה `units` במערכת, ושומרת אותם בוקטור. נשתמש בוקטור בפונקציה `getRandPacket`, כיוון שיש רואטרים במערכת שאינם מחוברים ל `units`, ולכן זה יעזור בשליחת חבילות.
- `void set_units_rates(int** rates, int size, int unit_index)` : הפונקציה מקבלת מטריצה של קצבים, מספר היחידות שיכולות לשלוח ליחידה כלשהי אשר המיקום שלה הוא `unit_index`, הפונקציה מחפשת את הקצבים המתאימים שלפיהם היחידה מקבלת חבילות מרכיבים אחרים במערכת וממלאת במיקום המתאים.
- `string getRandPacket()` : הפונקציה מקבלת את הקורדינאטות של היחידות אשר מוגדרות כיחידות פעילות כלומר יכולות להיות יעד של חבילה וכן יכולות הקצב המוגדר עבור היחידה המתאימה אינו אפס, הפונקציה בוחרת בצורה רנדומלית אחת זוג של קורדינאטות שמתאים לאיזשהי יחידה והיחידה הזו נבחרת כיעד של החבילה, הפונקציה מחזירה מחרוזת שמכילה את הקורדינאטות של חבילת היעד.
- נשים לב שלכל יחידה יש קצב מסויים שבו היא יכולה לקבל חבילות מיחידות אחרות, ולכן הקצב הזה נחשב כסיכוי של בחירה ליחידה כלשהי, למשל אם יחידה (x,y) צריכה לקבל כסיכוי 0.3 חבילות מיחידה `u1`, 0.2 מיחידה `u2` ו- 0.5 מיחידה `u3`. אז בבחירה הרנדומלית שעשינו הסיכוי שכל יחידה מאלה תתקבל מתאים בדיוק לסיכוי של קבלת חבילה מאותה יחידה.
- `void receive_packet(string packet)` : הפונקציה מקבלת את החבילה ושומרת אותה בוקטור של חבילות.

- `void send_packet()` : הפונקציה קוראת לפונקציית `getRandPacket` ומעבירה את החבילה לרואטר המחובר אליה ע"י קריאה לפונקציית `receive_packet` של הרואטר.

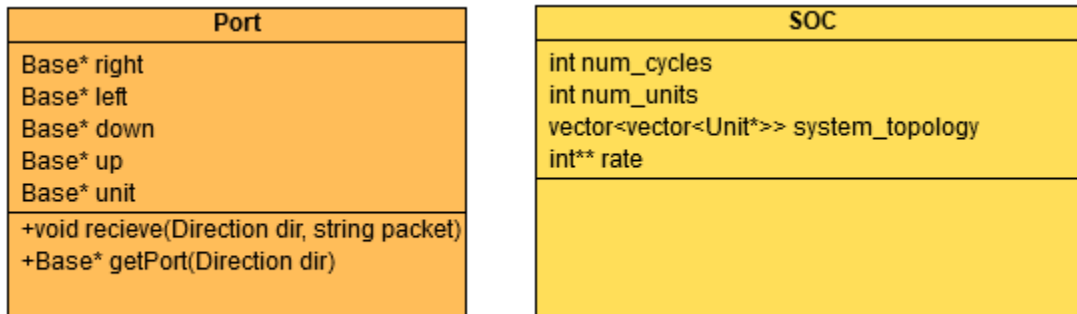


Figure 4: SOC and Port classes

3- Port : מחלקה שמתארת את החיבור של הרואטרים במערכת, לכל רואטר לפי חיבורו במערכת מוגדרים את השכנים שלו ואת היחידה שתלויה בו אם יש כאלה. כך שמתוארים בצורה הבאה : Right, Left, Up, Down, Unit הפונקציות :

- `void recieve(Direction dir, string packet)` : הפונקציה קוראת לפונקציה של `receive_packet` של הרואטר/unit המתאימה לפי `dir` שהוא `element` של `direction`.
- `Base* getPort(Direction dir)` : מחזירה את הרכיב המתאים ל `dir` שמחובר לרואטר.

4- Class SOC (System on a Chip) : מכיל :

- הטופולוגיה של המערכת אשר תוגדר לפי בחירת המשתמש
- קצב קבלת חבילות בין כל שני רכיבים במערכת
- משך הסימולציה.

אלגוריתם מימוש וריצת הסימולטור

שלבי האלגוריתם:

אפשר לחלק את אלגוריתם ריצת הסימולטור לשלושה שלבים, השלב הראשון הוא קבלת המידע מהמשתמש, השלב השני הוא הרצת הסימולציה והשלב האחרון הוא יצירת מפות חום.

שלב 1 – קבלת המידע מהמשתמש :

א. הוא בוחר את אופן החיבור של המערכת שמבוסס על NOC topology - המשתמש בוחר מספר של יחידות ולפי הבחירה הזו מותאם מספר ריבועי של ראוטרים הגדול שווה ממה שהוכנס כקלט על מנת שיתקיים תנאי ה-NOC.

ב. קובע את קצב קבלת ההודעות בין כל שני רכיבים.

ג. משך הסימולציה.

ד. היחידה המרכזית – Main Unit - המקבלת חבילות מבחוץ בכל מחזור שעון.

כמו שהוסבר מקודם המערכת בנויה בצורת NOC ולכן מספר הראוטרים יהיה מרובע לעומת הזאת, יש למשתמש את האפשרות להכניס מספר כלשהו של יחידות, ולכן ייתכנו ויש במערכת ראוטרים שאינם מחוברים בכלל לאיזשהי יחידה, ואת הראוטרים האלה ייעזרו לנו בשמירה על מאפייני המערכת ועל בחירת המסלול לפי אלגוריתם X-Y.

ה. רצה פונקציית מעטפת שבונה את המערכת ע"פ בחירות המשתמש.

שלב 2 – הרצת הסימולציה:

חבילות מתחילות להתקבל לראוטר של היחידה המרכזית שנקבעת על ידי המשתמש מאיזשהו התקן חיצוני וכן בכל יחידה Unit מיצרות פנימית חבילות רנדומליות ונשלחות לראוטר המחובר ליחידה שמעביר אותן לראוטר אחר על המסלול ליחידת היעד.

- אופן בחירת המסלול במערכת הוא לפי אלגוריתם X-Y שבו החבילה תועבר על ציר X עד שנגיע לקורדינטת היעד, ואחר כך תועבר על ציר Y עד שתגיע ליעד.

במהלך כל מחזור שעון יש כמה פעולות שמתבצעות :

- חבילה מתקבלת מבחוץ לתוך הראוטר של היחידה המרכזית.
- בדיקת תוכן החבילה ושליחתה ליעד אם זה ליחידה שתלוייה בראוטר עצמו או לראוטר אחר במסלול של יחידת היעד
- הראוטרים מקבלים חבילות מכל מהרוטרים השכנים להם ובודקים את התוכן ושולחים כמו בצעד הקודם.

שלב 3 – יצירת מפות החום:

ייצור שתי מפות חום ע"מ למצוא צווארי בקבוק של הביצועים. בסה"כ יש לנו שתי מפות חום אחת של ראוטרים ואחת של יחידות, עוד על איך מיצרים את מפות החום בפרק הבא.

מפת החום

כל יחידת זמן מחולקת לשני שלבים: החבילות שמועברות על הקווים (בין הראוטרים), החבילות שמבוצעות בתוך היחידות. הדרך שבה תיווצר מפת החום, היא כדלהלן:

במפת החום הראשונה נסכום את מס החבילות שנשלחות בין כל שני רכיבים במערכת ונחלק אותם במספר הרכיבים שנמצאים במערכת ונקבל את העומס הממוצע על כל הרכיב, במפת החום תהיה לנו תמונה של המערכת שמכילה את הרכיבים לפי החיבור שלהם במערכת וכל רכיב יוצבע בצבע מתאים לפי מרחק העומס עליו. (נשים לב שגם לכל רכיב יש ממוצע לעומס עליו לפי הקצבים בינו ובין כל רכיב אחר).

המפה השנייה תוגדר באופן דומה אבל הפעם בראוטר עוברות חבילות שאינם מגיעות לרכיב המתאים לראוטר אבל הם "סתם" עוברות במסלול הזה ולכן צריך להוסיף את זה לממוצע שאנחנו מחשבים.

נסכום את מס' ההודעות שמתקבלות בכל רכיב ואת מס' ההודעות שנשלחות מכל רכיב, ונחלק כל סכום במס' הרכיבים שיש במערכת. כלומר, נמצא את הממוצע של שליחת וקבלת הודעות לכל רכיב.

במפה יש שלושה צבעים (כחול, כתום ואדום) שמתארים את העומס על המערכת, הבחירה של הגדרת עומס כפי שנראה בהמשך תלויה במספר מחזורי שעון וגם כן בראוטרים שתלויות בהם יחידות.

מפה של היחידות:

לכל יחידה u יש את שני הערכים `num_of_recieved_packets` אשר מצביע על מספר החבילות שהגיעו ליחידה במהלך הסימולציה, ואת `num_of_stuck_packets` אשר מצביע על מספר החבילות שנתקעות בתוך הראוטרים במערכת וצריכים להגיע יחדה תוך מספר סופי של מחזורי שעון.

מפה של הראוטרים:

עבור ראוטר r כלשהו צריך לחשב את מספר החבילות הנתקעות בראוטר הזה בסוף הסימולציה, שזה מתאים למספר החבילות הכולל שהתקבל במהלך הריצה פחות מספר החבילות שנשלחו הלאה. או פשוט מספר החבילות שנשארו בסוף בתוך הרשימה שמתאימה לכל ראוטר.

OOPc Simulator

#Units:

#Threads:

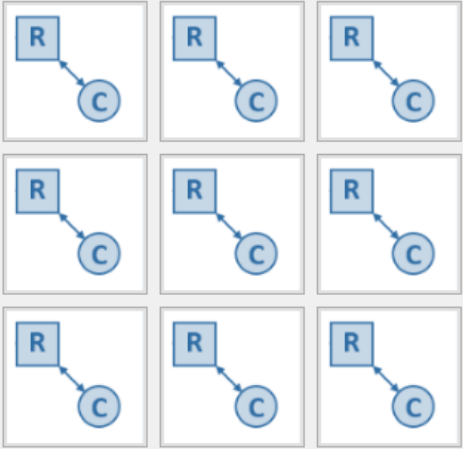
Total Simulation Time:
 Cycles

Choose Units

Build System

Start Simulation

Show Heat Map



טסטים לבדיקה

בשני הטסטים הבאים משנים את מספר החוטים עם מספר קבוע של יחידות בכל פעם.
משך מחזור השעון בכל הטסטים הוא 1000 מחזורי שעון, והיחידה המרכזית היא התלוייה בראוטר (0,0).
תוצאות הריצה בכל טסט המספרים מתארים את מספר החבילות שנתקע בכל ראוטר.

טסט 1:

מערכת הבנויה מ:

1. 4 units

2. 4 routers

תוצאות הריצה:

	10	8	6	4	2	1	
ראוטר(0,0)	1812	1468	1635	1762	1981	1986	
ראוטר(0,1)	880	747	836	741	985	975	
ראוטר(1,0)	1072	7	1247	975	800	823	
ראוטר(1,1)	738	529	795	311	925	1024	

טסט 2:

מערכת הבנויה מ:

1. 6 units

2. 9 routers

תוצאות הריצה:

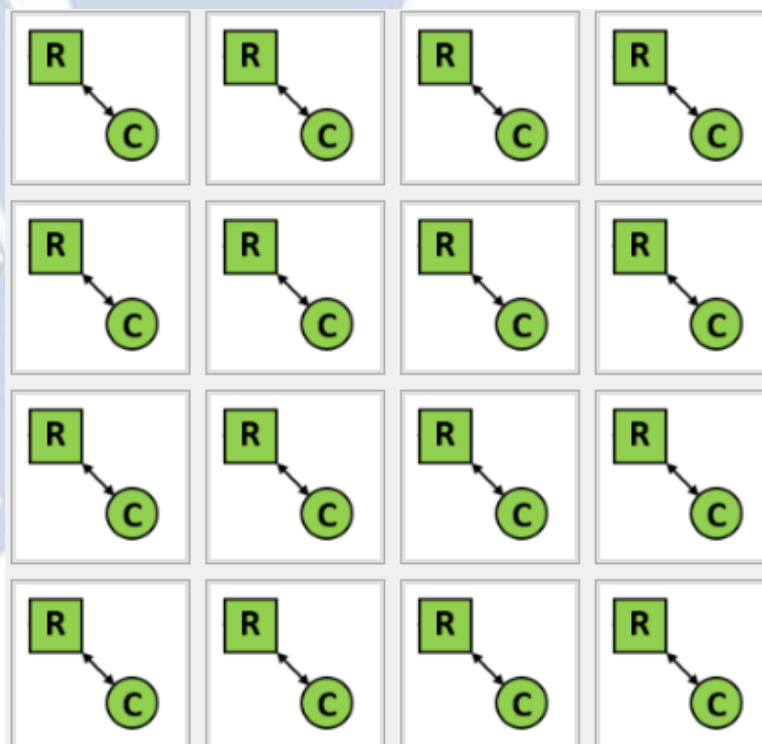
	8	6	4	2	1	
ראוטר(0,0)	1991	1555	1843	1934	1951	
ראוטר(0,1)	5	3	1	2	0	
ראוטר(0,2)	996	873	974	985	1000	
ראוטר(1,0)	756	1091	545	1028	978	
ראוטר(1,1)	1068	177	1030	841	1000	
ראוטר(1,2)	1	3	21	15	2	
ראוטר(2,0)	1000	997	1260	482	1027	
ראוטר(2,1)	19	8	67	310	1	
ראוטר(2,2)	993	1120	966	836	1000	

אם משנים את מספר היחידות עם מספר קבוע של החוטים (3 חוטים) נקבל:

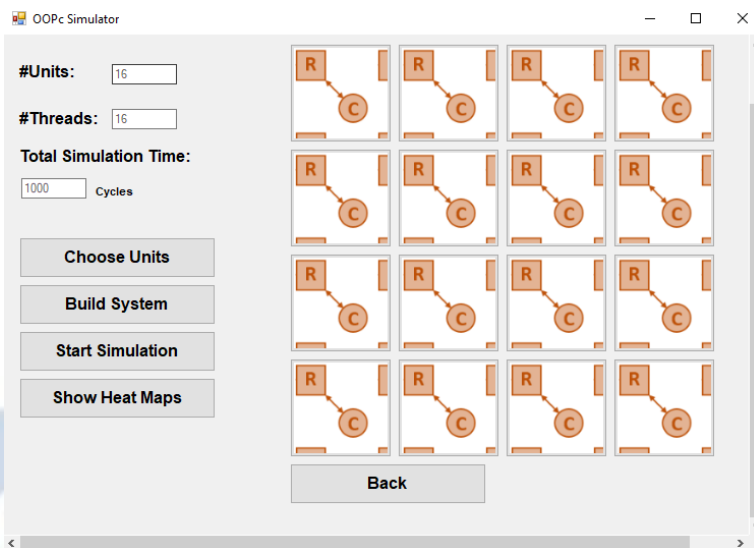
9	6	4	2	
1997	1997	1984	958	ראוטר(0,0)
1001	0	1001	0	ראוטר(0,1)
998	0	אין	אין	ראוטר(0,2)
1025	1003	935	3	ראוטר(1,0)
979	1014	1003	0	ראוטר(1,1)
997	985	אין	אין	ראוטר(1,2)
1004	3	אין	אין	ראוטר(2,0)
997	1080	אין	אין	ראוטר(2,1)
1000	867	אין	אין	ראוטר(2,2)

טסט 3:

טסט גדול של 16 יחידות ו- 16 חוטים עם 1000 מחזורי שיעון.

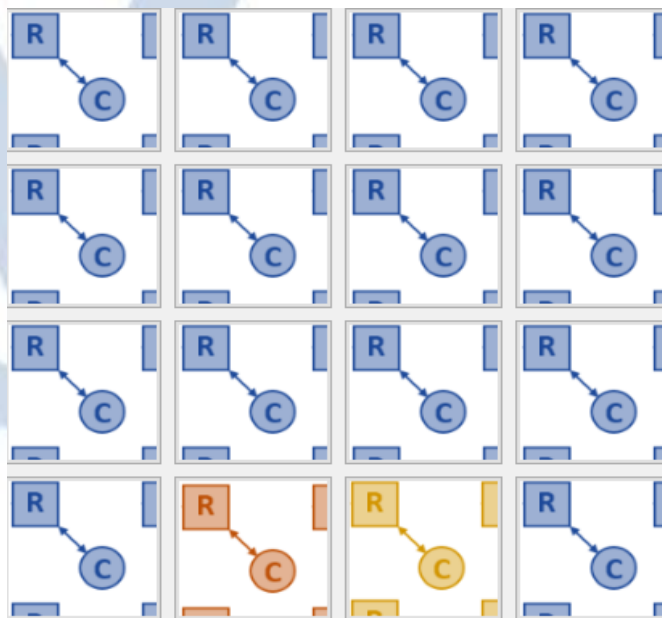


מפת חום של הראוטרים:



כמו שרואים בכל הראוטרים נתקעות מספר גדול של חבילות.

מפה של היחידות:



רק שתי יחידות נתקע בהם מספר גדול של חבילות כך שאחת בצבע כתום והשנייה בצבע אדום, בשאר יש מספר סביר של חבילות נתקעות.

הסבר וסיכום

בשני הטסטים הראשונים קבענו מספר כלשהו של יחידות במערכת ושינינו את מספר החוטים, מספר היחידות נבחרים כך שפעם נקבל מערכת מלאה כאשר בוחרים 4 יחידות, ובטסט השני בחרנו 6 יחידות כך שבמערכת הבנויה יהיו 9 ראוטרים והבחירה של ה-6 מתוך ה-9 היא חופשית למשתמש.

בשני הטסטים נשים לב כי כמות החבילות שמתקבלות בראוטרים שתלוי ביחידה גדולות יותר מאשר כמות החבילות שמתקבלות בראוטרים החופשיים וזה נכון לכל מספר חוטים שנבחר.

וזה הגיוני כי הראוטרים החופשיים מהווים תחנות להעברת חבילות על מסלול $X-Y$ בין מקור ויעד. לעומת זאת בראוטרים האחרים מתקבלות חבילות רנדומליות בכל מחזור שעון בנוסף לחבילות שמתקבלות על המסלולים בין כל יחידה ויחידה.

נשים לב כי אין כל כך הבדל מהותי בין כמות החבילות שמתקבלות בראוטרים לא משנה מה מספר החוטים שנבחר.

בטסט האחרון בחרנו במספר ממוצע של חוטים (3 חוטים) כמו שנאמר קודם לא משנה המספר שנבחר, שינינו את מספר היחידות בכל פעם וכראה כמעט רק בראוטרים שתלויים ביחידות התקבלו מספר גדול של חבילות.

נשים לב שבכל הטסטים הראוטר (0,0) שהיחידה שלו נבחרת להיות היחידה המרכזית התקבלו הכי הרבה חבילות כי בנוסף לכך שהראוטר מקבל חבילה רנדומלית כל מחזור שעון גם הוא מקבל חבילה מבחוץ בכל מחזור.

לסיכום:

- 1- מספר הרכיבים במערכת משפיע על מספר החבילות הכולל שעובר במערכת, ככל שיש יותר רכיבים עוברות יותר חבילות.
- 2- שינוי מספר החוטים לא השפיע על מהירותה של המערכת, כך שלא משנה כמה חוטים ירוצו בו זמנית העומס על המערכת נשאר כמו שהוא.
- 3- שינוי הקצב של החבילות העוברות בין כל שני רכיבים לא משפיע על העומס של המערכת.
- 4- מספר מחזורי השעון של משך הסימולציה משפיע על העומס של המערכת, כך שכל שהמערכת רצה יותר זמן, מספר החבילות הנתקעות בכל ראוטר יגדל יותר.

אתגרים

במהלך העבודה על הפרוייקט נתקלנו בכמה אתגרים החל מההבנה של הפרוייקט ואיך צריך לחתק את הביע לתת בעיות עד לסיום המימוש הכולל מימוש של הסימולטור ומימוש של ה-GUI שמאפשר למשתמש הבנה יותר עמוקה למה משמש הפרוייקט הזה.

להלן חלק מהאתגרים שנתקלנו בהם:

- 1- הבנה עמוקה של המושגים החשובים על מנת שנוכל לממש את הסימולטור בצורה המתאימה, וזה דרש ממנו לחפש באינטרנט ולצפות בכמה סרטונים שבאמצעותן צברנו את המידע המתאים.
 - 2- החלק של ה-GUI שזה היה החלק המאתגר ביותר, החלק הזה דרש ממנו לעבור כמה תרגולים כדי להבין איך לממש את זה. בהתחלה התחלנו לעבוד על CLION אבל בגלל כמה סיבוכים של ה-GUI שם החלטנו בסוף לממש את הסימולטור ב-VISUAL STUDIO.
 - איזדהאר עבדה על החלק המאתגר ביותר ב-GUI שזה מאפשר למשתמש ליצור בזמן ריצה את המערכת כולל תמונות שמשקפות את הצורה האמיתית של המערכת כמו שהמשתמש יבקש. גדיר עבדה יותר על החלק שבו ניצור את מפת החום.
 - 3- הוספת mutexes ו-locks על מנת שנקבל multithreading system.
 - 4- המימוש הכללי של הסימולטור שכולל בעיקר החיבור בין ה-GUI ובין המימוש של ה-backend כך שהכל יעבוד כמו שצריך, וכל לחיצה של המשתמש יהיה מתאים בדיוק לפעולה שצריך שתבצע בכל שלב ושלב.
 - 5- mutex לא נתמך ב-visual studio בזמן קומפילציה עם clr, הפתרון היה ליצור קובץ ++c - מכבים את התמיכה ב-clr - שם יצרנו מחלקה שיש לה את הפונקציונליות הנדרשת.
- בסוף רוצים לציין שכל האתגרים האלה ממש עזרו לנו להבין יותר מה אנחנו עושים וגם צברנו ידע חשוב שיעזור לנו בהמשך.

Simulator User Guide

When the simulator is first opened, the simulator window will appear. As shown in Figure 1, the simulator window includes several boxes and several buttons that appear after completing each step of the simulation. The simulator window also includes a layout of the system topology that will be displayed at the right side of the simulator window.

1, **2**, **3** : is where the user decides the number of Units, Threads in the system, and the total simulation time.

4 : this button displays a layout of the total routers in the system, the user decides which routers are connected to units.

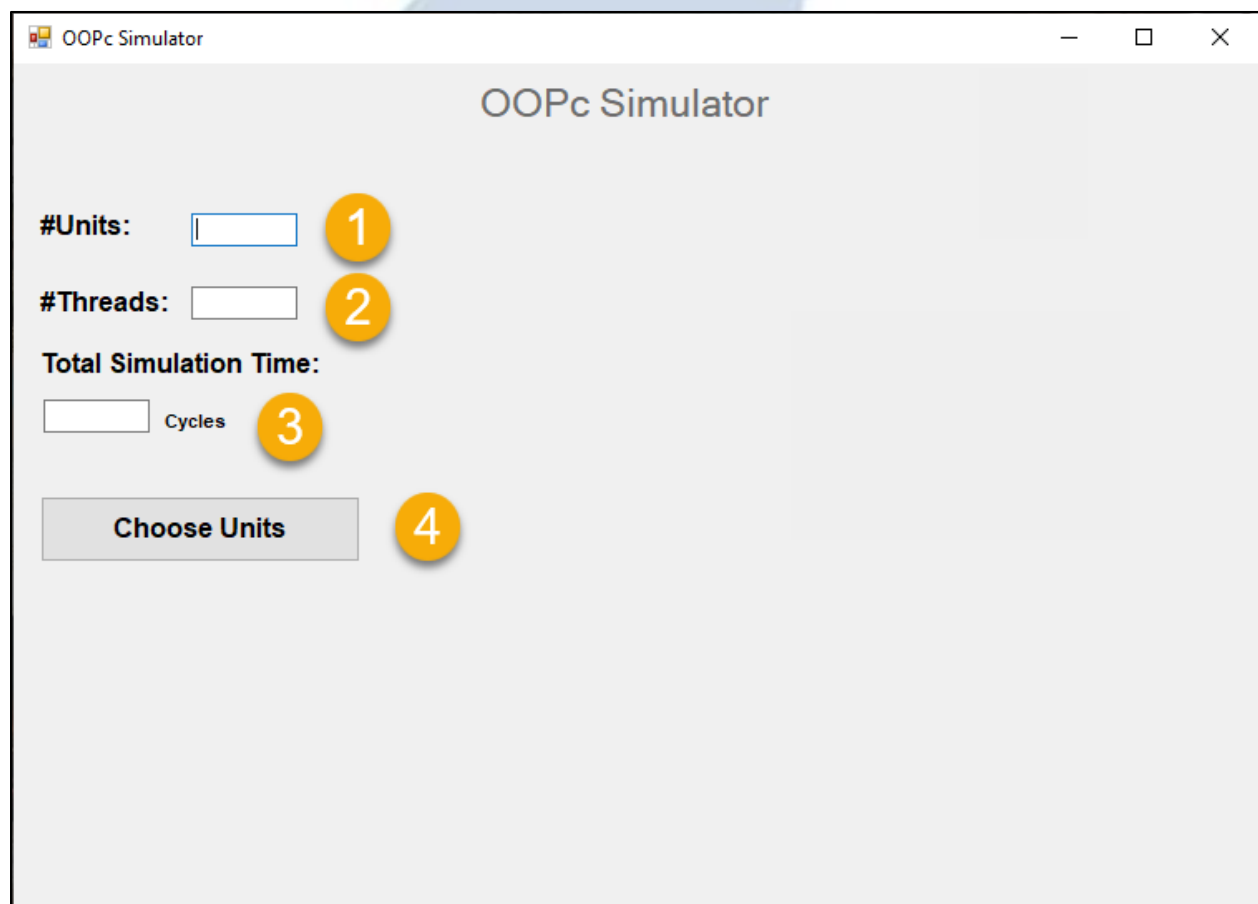


Figure 1 The Simulator Display Window

In Figure 2 **5** is the layout of the routers in the system, the routers are numbered from 0 to the (number of units -1). The user can select which router is connected to a unit by clicking on the box with the number of the router.

6 this button builds the system based on the information and the topology the user decided.

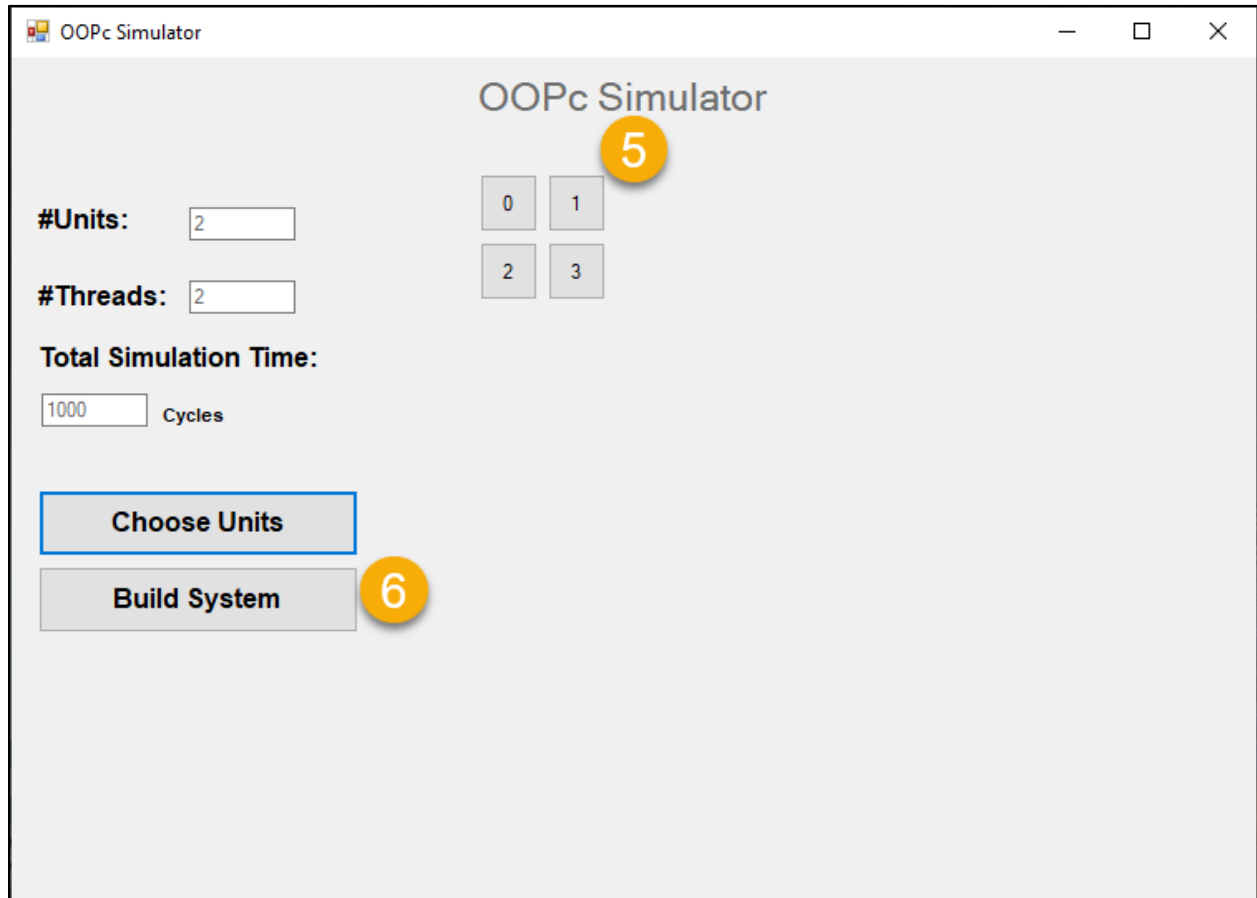


Figure 2 Build System Button and routers layout

In Figure 3 **7** is the layout of the system topology after clicking on the Build system button.

8 The start simulation button runs the simulation.

Each router\ router and unit in **7** are a button, clicking on the button open a window, the window in Figure 4, where the user needs to give more information regarding the units in the system.

In Figure 4, **A** is where the user enters the rate of received packets on the route between two units

in the system. **B** is a checkbox, if the user checks the box, then the units is set as the main unit in the system. The main unit is the units which receives packets from outside the system and sends it the destination unit inside the system.

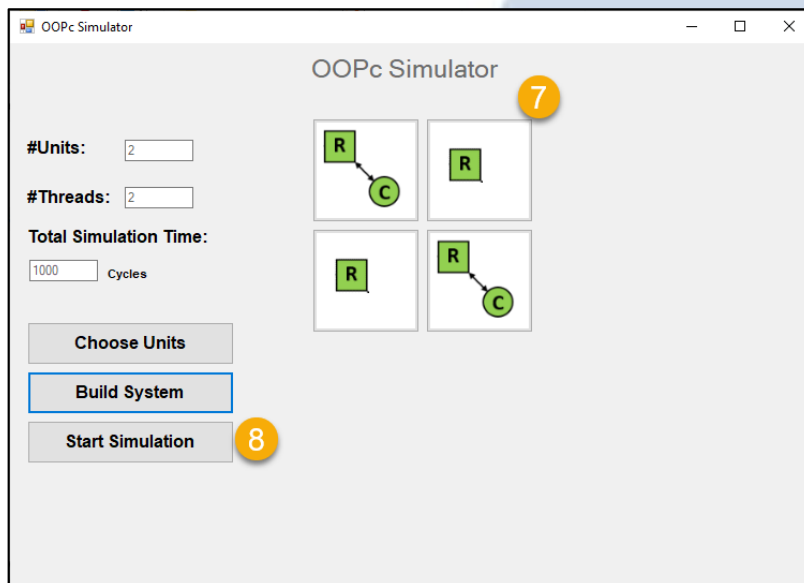


Figure 3 Start Simulation Button and System layout

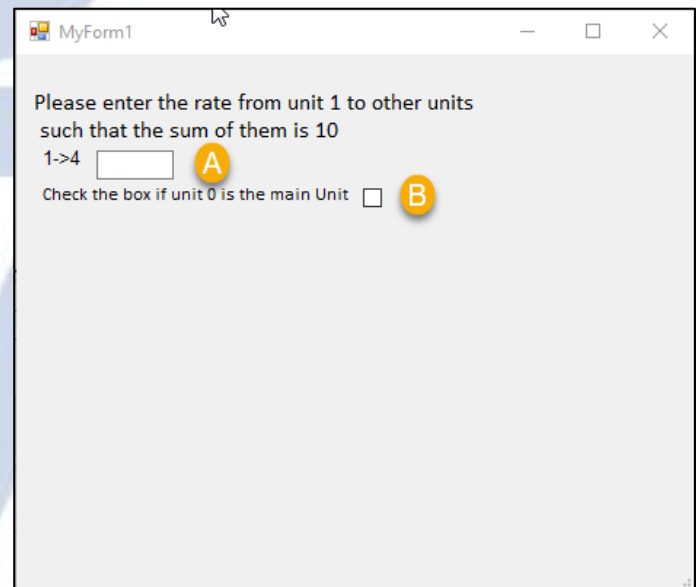


Figure 4 Units

In Figure 5 **9** is the heat map button, this button appears after the simulation ends. Clicking on the button takes us to Figure 6.

In Figure 6 the user has two options for a heatmap, one for routers and another one for units.

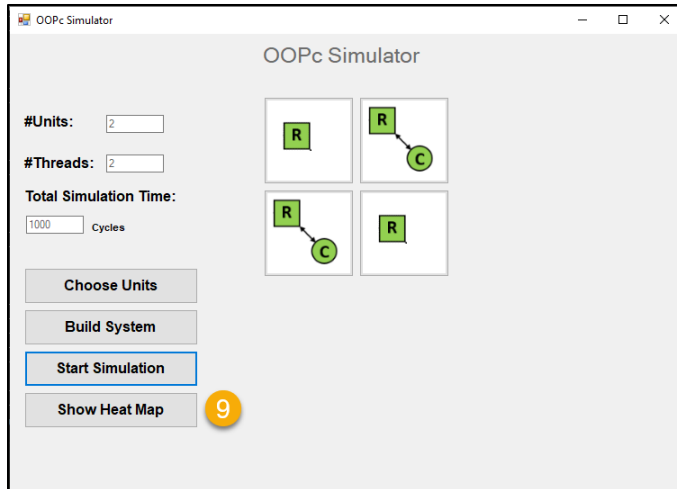


Figure 5 HeatMap Button

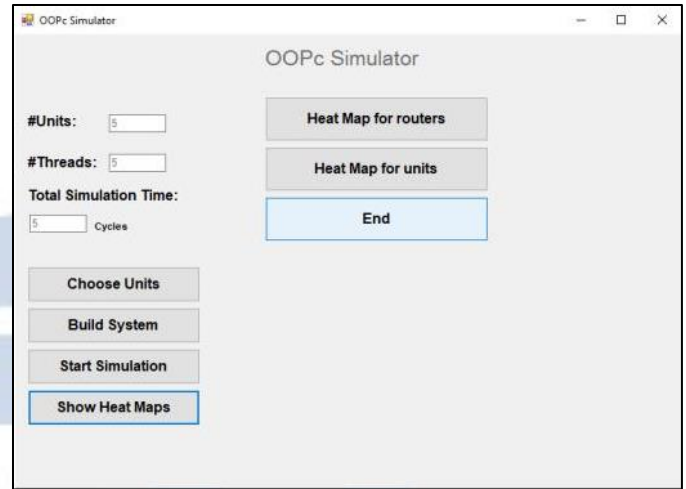


Figure 6 HeatMap options

In Figure 7 appears the heatmap, the color of each router is based on load of the packets stuck in the router after the simulation ended.

Blue: is for light load.

Orange: is for medium load.

Red: is for heavy load.

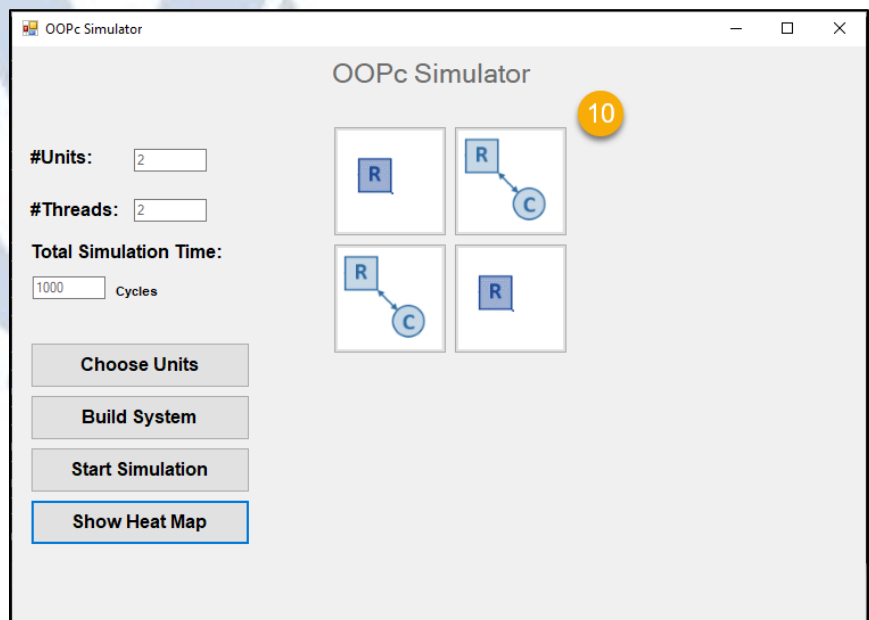


Figure 7 HeatMap layout

מקורות

- 1- NOC definition:
<https://www.sciencedirect.com/topics/engineering/network-on-chip>
- 2- VLSI definition:
<https://he.wikipedia.org/wiki/VLSI>
- 3- SOC definition:
https://en.wikipedia.org/wiki/System_on_a_chip
- 4- Router definition:
[https://en.wikipedia.org/wiki/Router_\(computing\)](https://en.wikipedia.org/wiki/Router_(computing))
- 5- X – Y algorithm:
<https://www.interscience.in/cgi/viewcontent.cgi?article=1384&context=ijcct>
- 6- OOPc simulator:
<https://vlsi.eelabs.technion.ac.il/projects/architectural-simulator-for-object-oriented-processor/>

