דו"ח מסכם פרויקט
Extended Pipeline

נדא אבו סאלח 316151232
חאלד סגיר 207409137

מבוא:

בדו"ח זה אנחנו משתפים את הפרויקט הסופי בקורס שמתמקד בהרחבת המעבד מ5 שלבים ל7 שלבים בהתבסס על ארכיטקטורת ה RISC-V. הדו"ח יכלול גם את חווית הלמידה האישית, עבודת הצוות והאתגרים שניצבו מולנו גם ביצועית וגם טכנית.

רקע:

למדנו במהלך הקורס על הארכיטקטורה הבסיסית של מעבדי ה RISC-V הכוללת 5 שלבים:

IF (Instruction Fetch)	הבאת ההוראה מהזיכרון
ID (Instruction Decode)	שלב פענוח הוראת המכונה וקביעת הפעולות הנדרשות
EXE (Execution)	ביצוע הפעולות הבסיסיות
MEM (Memory Access)	גישה לזיכרון לצורך כתיבה או קריאה
WB (Write Back)	כתיבה של התוצאה לרשומה הרלוונטית

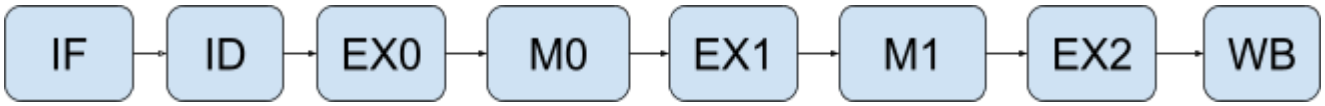
המשימה : הרחבה של המעבד ל7 שלבים

שלב נוסף של MEM ושני שלבים של EXE כאשר שלב ה MEM דרוש לצורך גישה נוספת לזיכרון להכנת הנתונים לשלב הבא וה EXE יתנו לנו לבצע פעולות מתקדמות ומסובכות יותר.

ארכיטקטורה חדשה:

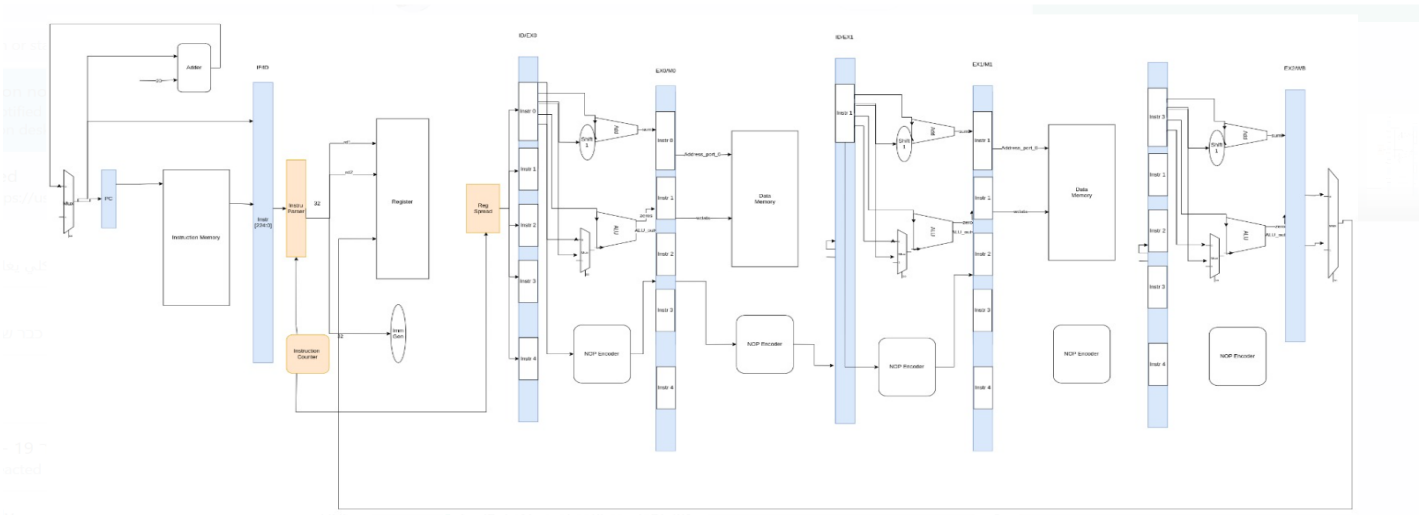
ארכיטקטורה של מעבד עם 7 שלבים הכוללים כעת עבודה עם שני שלבים נוספים של EXE ושני שלבים של MEM.
כל פקודה שנכנסת למעבד הנ"ל תעבור דרך כל השלבים הבאים

IF ⇒ ID ⇒ EXE ⇒ MEM ⇒ EXE ⇒ MEM ⇒ EXE ⇒ WB:



כאשר בשלב הראשון IF - Instruction Fetch המעבד מביא את ההוראה הבאה מהזיכרון
שלב השני ID - Instruction Decode בשלב זה, המעבד מפענח את ההוראה
שלב שלישי EX0 - Execute בשלב זה מתבצעת הפעולה או הפקודה הנדרשת
שלב רביעי M0 -Memory Access בשלב זה המעבד נגש לזיכרון להביא נתונים או אחסון בזיכרון
שלב חמישי EX1 - Execute בשלב זה מתבצע החלק השני של הפקודה
שלב שישי M1 -Memory Access בשלב זה המעבד ניגש שנית לזיכרון להביא נתונים או אחסון בזיכרון
שלב שביעי EX2 - Execute בשלב זה מתבצע החלק השלישי של הפעולה או הפקודה
שלב שמיני WB - Write Back שלב סופי המעבד כותב את תוצאות לרגיסטרים או לזיכרון.

הדיאגרמה מתארת בדיוק איך אנחנו מחלקים את שלבי העבודה של הרגיסטרים תוך כדי ביצוע פעולות מורכבות (דיאגרמה יותר ברורה ומפורטת הוספנו בחלק של הנספחים).



חווית הלמידה:

הלמידה הייתה מאתגרת ומרתקת במיוחד, תהליך ההרחבה עצמו דורש הבנה מעמיקה של הארכיטקטורה של מעבדי RISC-V ואת היכולת ליישם אותה בפועל באמצעות כלי הפיתוח והסימולציה של VIVADO.

אתגרים טכניים:

1) <u>תכנון הארכיטקטורה:</u>

אחד האתגרים הראשוניים היה לתכנן את הארכיטקטורה החדשה כך שתהיה יעילה ומתאימה למטרות ההרחבה. זה כלל קביעת החיבורים בין השלבים החדשים לשלבים הקיימים וגם הוספת רשומות נוספות לרישום.

2) <u>אימות וסימולציה:</u>

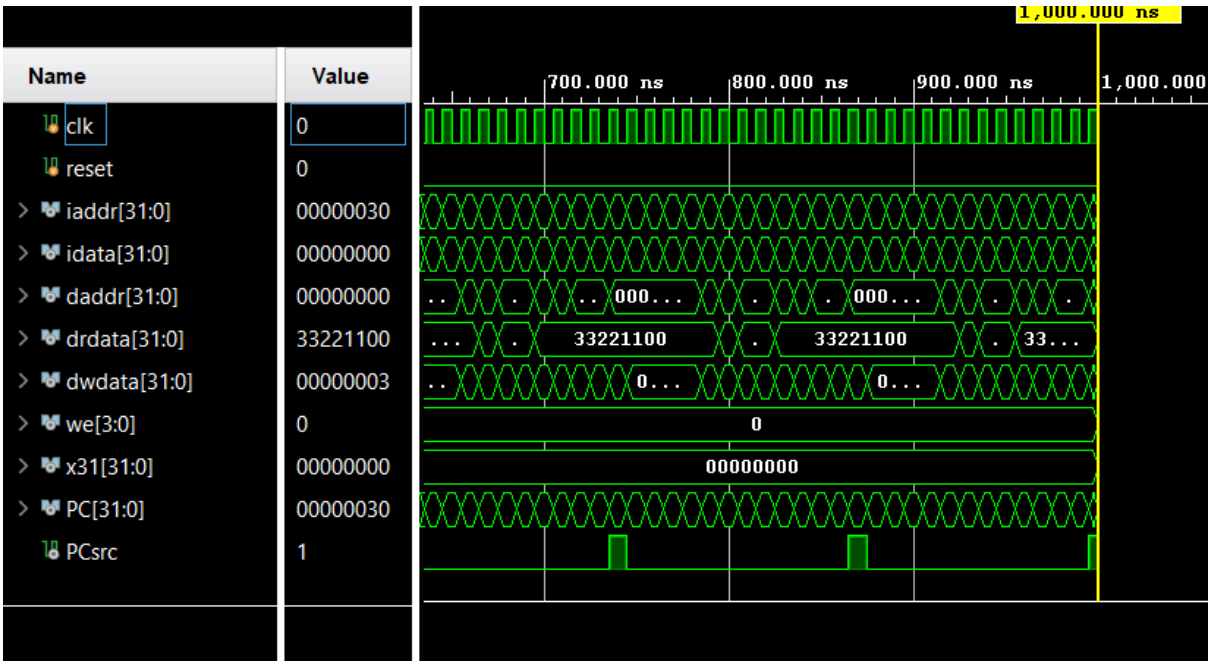
אחרי הפיתוח, היה צורך לבצע אימותים וסימולציות כדי לוודא שהארכיטקטורה החדשה פועלת כמצופה ואין בעיות ביכולת לבצע פונקציות שונות.

3) <u>אינטגרציה עם VIVADO</u>

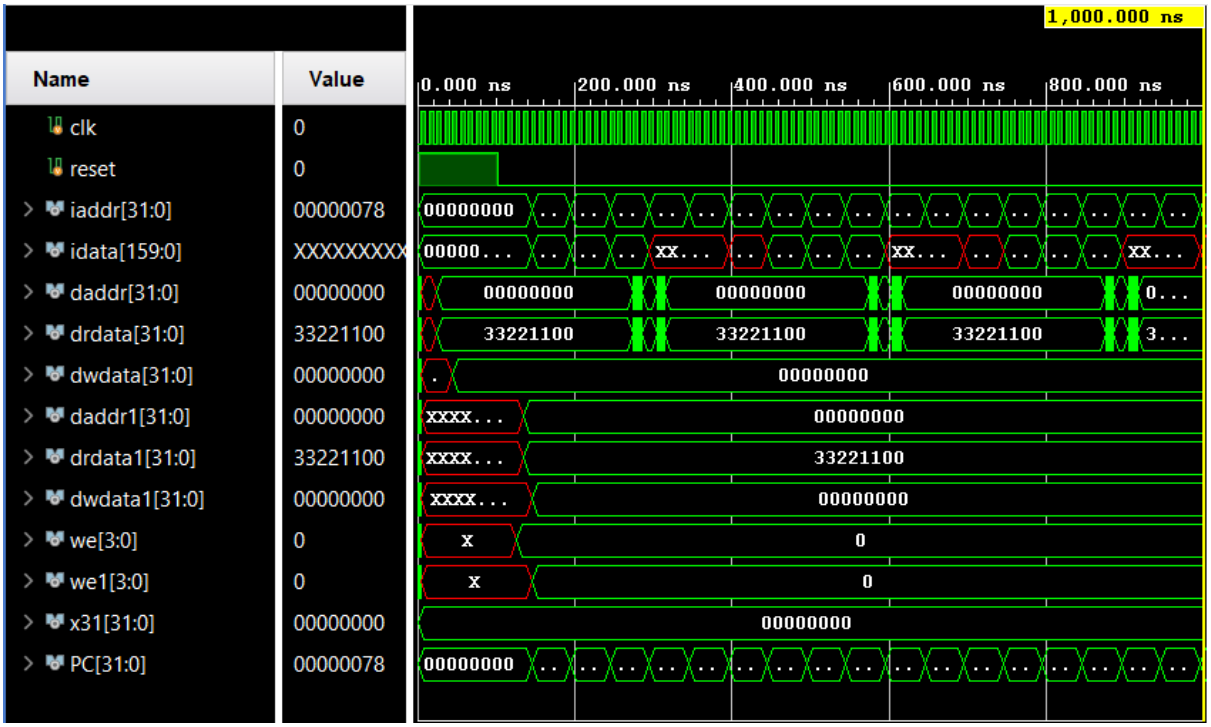
לאחר פיתוח הארכיטקטורה והקוד ב VERILOG היינו צריכים לבצע אנטגרציה של המעבד החדש בשלבי הסימולציה והסינתזה ששם נתקלנו בלא מעט באגים וההרצות בתוכנה היו מכבידות על המחשב.

תוצאות:

simulation 5-stage:

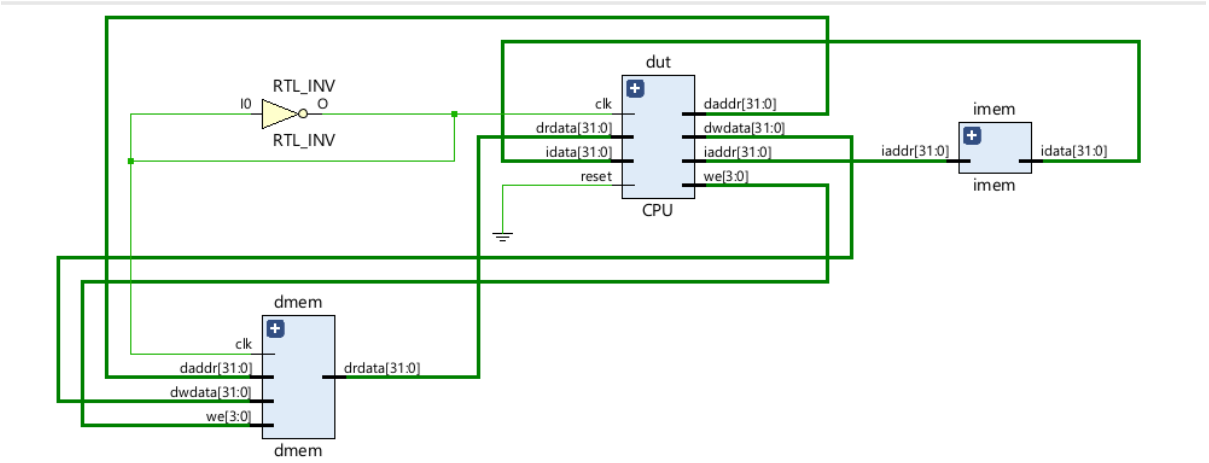


simulation 7-stage:

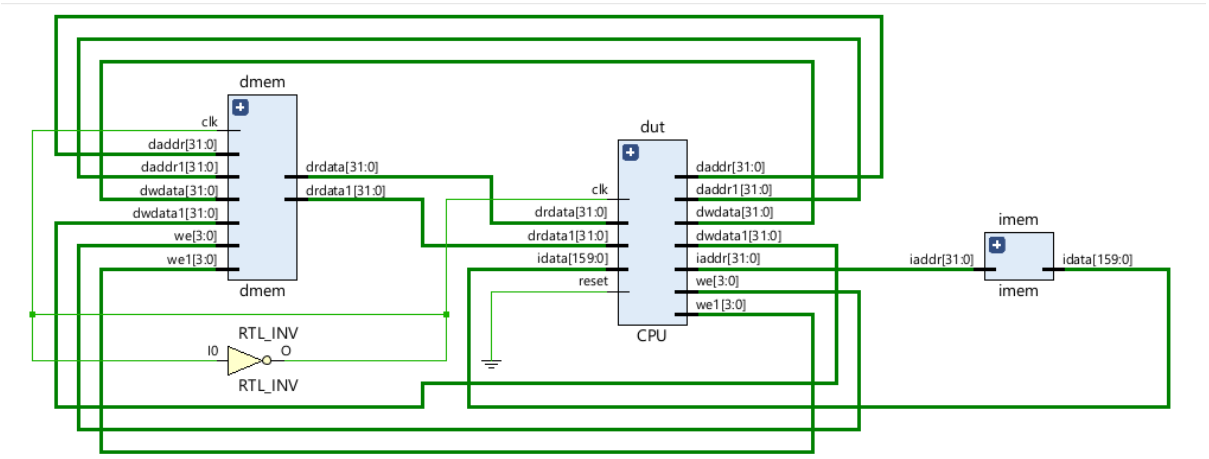


סכימה של אימפלמנטציה - רכיבים:

schematic 5-stage:



schematic 7-stage:



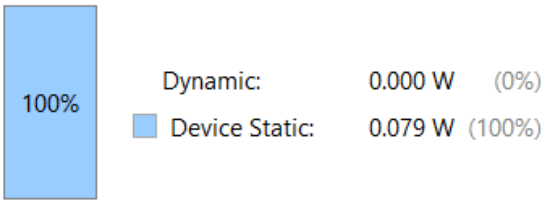
סכימה של סינתזה - שעון פאוור:

schematic 5-stage:

Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	**0.079 W**
Design Power Budget:	**Not Specified**
Process:	typical
Power Budget Margin:	**N/A**
Junction Temperature:	**25.1°C**
Thermal Margin:	59.9°C (31.6 W)
Ambient Temperature:	25.0 °C
Effective ϑJA:	1.9°C/W
Power supplied to off-chip devices:	0 W

On-Chip Power



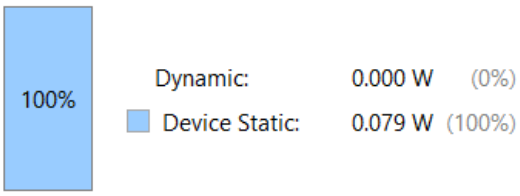
schematic 7-stage:

Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	**0.079 W**
Design Power Budget:	**Not Specified**
Process:	typical
Power Budget Margin:	**N/A**
Junction Temperature:	**25.1°C**
Thermal Margin:	74.9°C (39.4 W)
Ambient Temperature:	25.0 °C
Effective ϑJA:	1.9°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	High

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power



קטעי קוד :

1. Fibonacci

התוכנית מאתחלת את R1 ו-R2 ל-0 ו-1, ומשתמשת בלולאה כדי ליצור ולהדפיס את 10 מספרי פיבונאצ'י הראשונים
מחשבת על ידי הוספת שני האיברים R1 ו-R2,
הלולאה רצה 10 פעמים.

```
ADDI R1, R0, 0
ADDI R2, R0, 1
ADDI R3, R0, 10
ADDI R4, R0, 0

PRINT R1
PRINT R2

LOOP :
    ADD R5, R1, R2
    PRINT R5
    MOVE R1, R2
    MOVE R2, R5
    ADDI R4, R4, 1
    BLT R4, R3, LOOP
```

הקוד אחרי המרה לשפת מכונה:

```
00000013
00001013
00A02013
00000013
00C0006F
00C0006F
00111033
00C0006F
00112033
00112033
00112113
FE2FF06F
```

2. squares of numbers :

התוכנית מאתחלת את R1 ל-1ו-R2 ל 10,נריץ לולאה 10 פעמים בכל איטרציה של הלולאה מחשבים את (R1 * R1) שזה בעצם הריבוע של המספר.

```
ADDI R1, R0, 1
ADDI R2, R0, 10
LOOP :
    MUL R3, R1, R1
    PRINT R3
    ADDI R1, R1, 1
    BLE R1, R2, LOOP
```

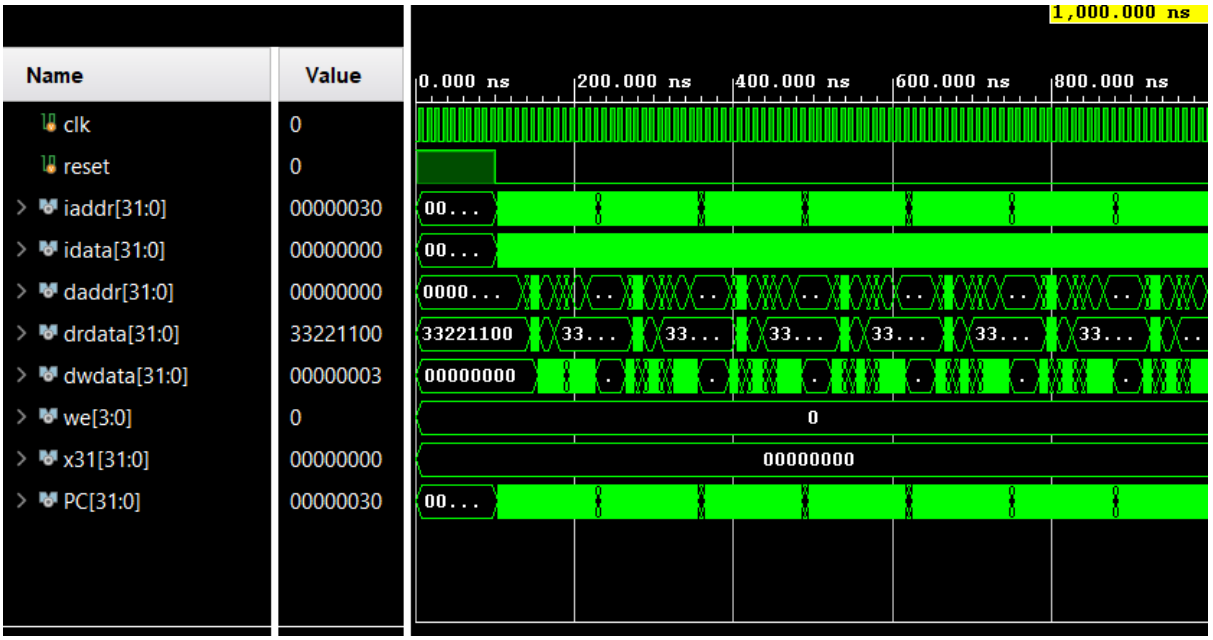
הקוד אחרי המרה לשפת מכונה:

```
00000013
00001013
00110233
00C0026F
00130313
00A30363
```

סימולציה עבור קטעי קוד:

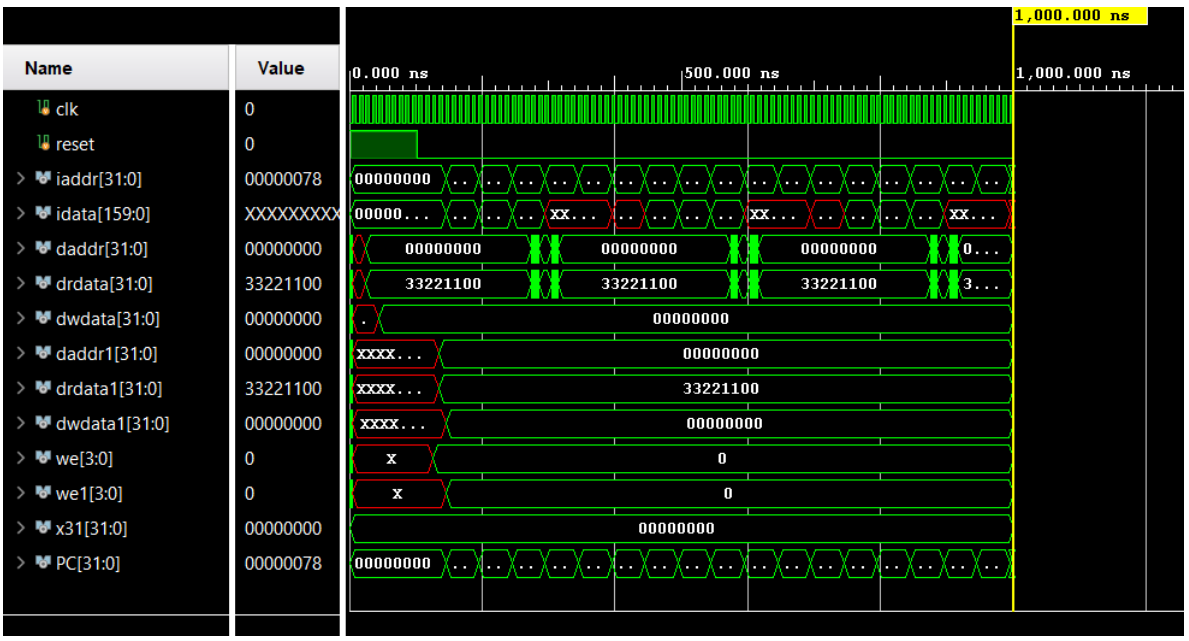
1) Fibonacci:

pipeline 5-stage:




```
INFO: [USF-XSim-96] XSim completed. Design snapshot 'cpu_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:05 ; elapsed = 00:00:07 . Memory (MB): peak = 1325.988 ; gain = 0.000
```

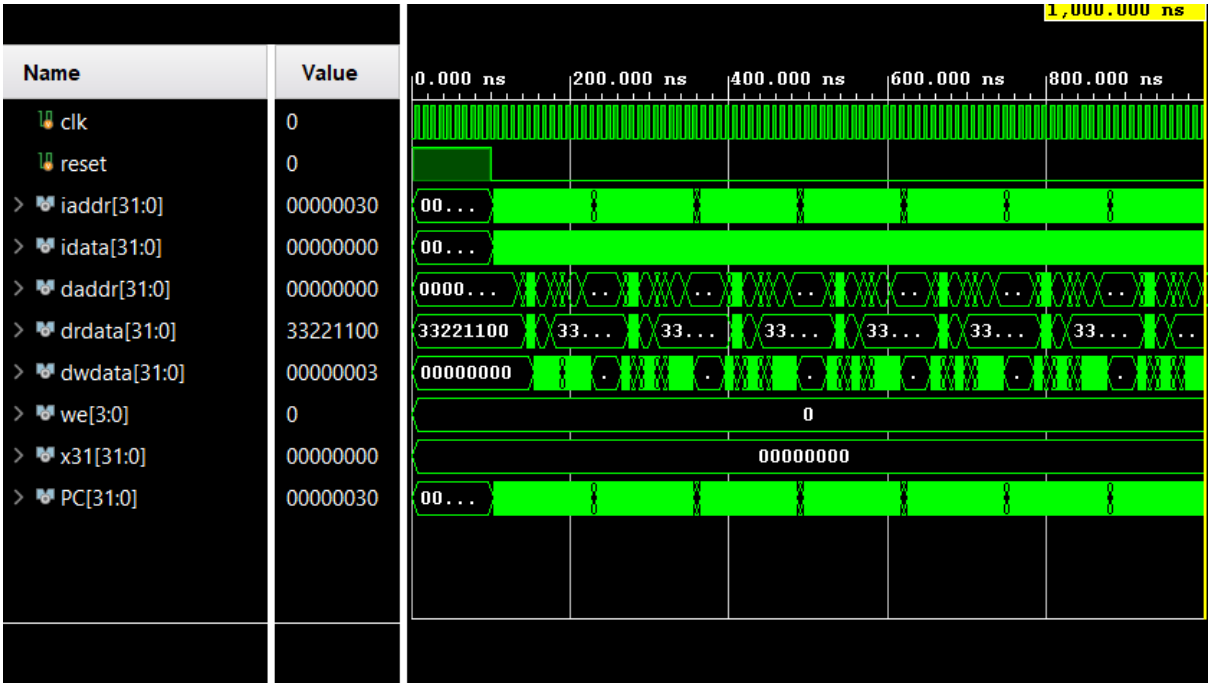
pipeline 7-stage:



```
INFO: [USF-XSim-96] XSim completed. Design snapshot 'cpu_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:12 ; elapsed = 00:00:10 . Memory (MB): peak = 1494.617 ; gain = 39.047
```

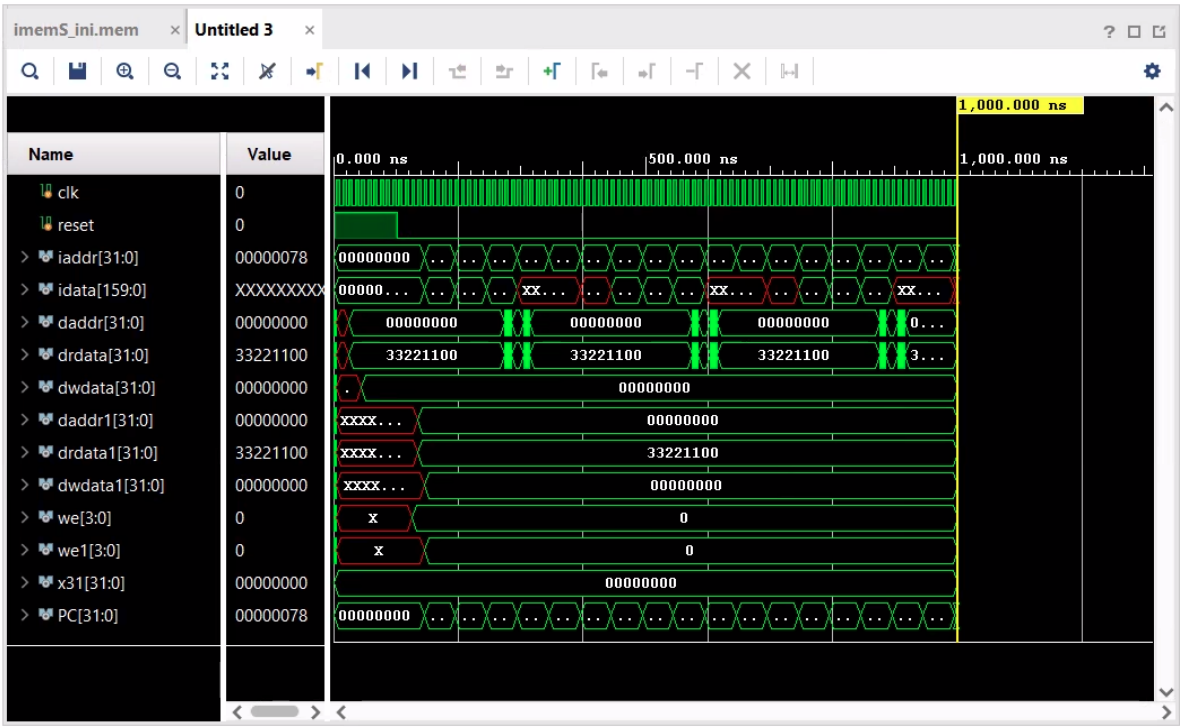
2)squares of numbers:

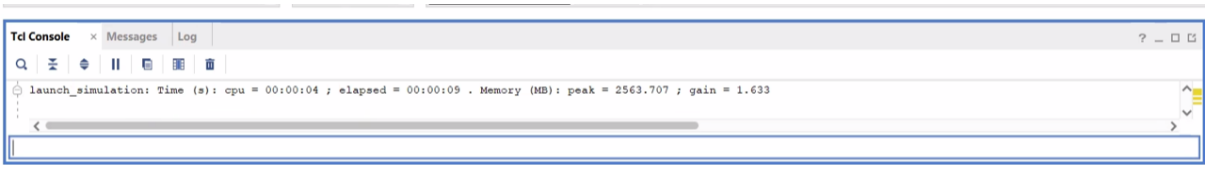
pipeline 5-stage:




```
INFO: [USF-XSim-96] XSim completed. Design snapshot 'cpu_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:11 ; elapsed = 00:00:10 . Memory (MB): peak = 1325.988 ; gain = 22.238
```

pipeline 7-stage:





השוואת התוצאות בין pipeline המקורי ל extended pipeline:

לאחר הרצת קוד פיבונאצ'י גילינו שזמן העיבוד של 5 השלבים היה פחות מזמן העיבוד של המעבד המורחב (7 השלבים) אך עם הרצת הקוד של ריבוע המספרים המעבד של 7 השלבים היה יותר מהיר מבחינת זמן עיבוד.
מבחינת זיכרון בשני הקטעים של הקוד ראינו שמעבד 7 השלבים צורך יותר זיכרון מאשר המעבד המקורי השלבים.
זמן העיבוד במעבד ב-5 שלבים ברוב המקרים (בקטעי קוד פשוטים) אמור להיות יותר מהיר מאשר במעבד ב-7 שלבים. זאת משום שמעבד ב-5 שלבים מבצע פעולות בשלבים קטנים ובאופן סינכרוני יותר. כלומר, בכל שלב יתבצע אירוע אחד ובעקבותיות.

במעבד ב-7 שלבים, כאשר ישנם שלבי MEM ו-EXE נוספים, קיימת אפשרות לבצע פעולות נוספות במקביל. זה יכול להוביל לתקופות שבהן מעבד ב-7 שלבים יוכל להיות יעיל יותר ולסיים יותר מהר ממעבד ב-5 שלבים בתנאים מסוימים.

אך על מנת שזמן העיבוד במעבד ב-7 שלבים יהיה אכן יעיל יותר, ישנו צורך שהתוכניות יהיו מותאמות לשלבים. זה אומר שהמחשב והמעבד ישתמשו באופן יעיל בשלבים הנוספים ויכולת להפעיל פעולות במקביל בצורה שתכניס את המרכיבים החדשים ביעילות לתוך התהליך.

בגלל זאת, הגורמים העיקריים שעלולים לגרום למעבד ב-7 שלבים להיות פחות יעילים ממעבד ב-5 שלבים הם:

1) סגנון הקוד (התוכנית) : ברגע שזה לא מותאם ויש בקוד הרבה פעולות שמחכות זמן רב לביצוע כאשר הן תלויות בפעולות קודמות.
2) התחממות: הרחבת המעבד עם שלבים נוספים עשויה לגרום להתחממות יתרה וצורך במערכת קירור מתקדמת יותר, מה שיגרום להאטה במקרים מסויימים.
3) אוברהד וצריכת יותר זיכרון ומשאבים.

ניתן להגיד שמעבד ב 7 שלבים הוא פוטנציאלית יעיל יותר במיוחד עבור פעולות מורכבות, אך זה דורש התאמה של התכנית וגם שיפורים טכניים נוספים על מנת למקם אותו ביתרון מול מעבד ב-5 שלבים.

מסקנות:

עבודת הצוות הייתה חשובה מאוד במיוחד בשלב של השלמת החומר והרחבה הידע הטכני הנדרש לפרויקט זה, ותוך כדי הפיתוח קיבלנו לניסיון פיתוח מסוג זה שדרש הבנה עמוקה יותר של המעבד ותוך כדי רענון ושיפור מיומנות בקוד של חומרה VERILOG וכמובן התנסות באינטגרציה ופיתוח תוך כדי העבודה עם כלי פיתוח כמו VIVADO.

נספח - דיאגרמה מפורטת: