



# דו"ח מסכם לפרויקט גמר קורס

"מעבדת ארכיטקטורת מעבדים מתקדמת ומאיצי חומרה"

361-1-4693

הפקולטה למדעי ההנדסה –

מעבדת ארכיטקטורת מעבדים מתקדמת ומאיצי חומרה

עבורה: רכוא חנניה

מגייסים:

אורן שור – 316365352

טל שווורצברג – 316581537



## תוכן עניינים

3	-	מטרת הפROYIKט והסביר על המערכת	.1
10	-	וריפיקציה פונקציונלית ליבת <i>PipelineMIPS</i>	.2
17	-	<i>GPIO</i>	.3
22	-	<i>Interrupt Controller</i>	.4
23	-	<i>Basic Timer</i>	.5
29	-	<i>UART</i>	.6



## 1. הסבר על המערכת

בפרויקט זה, התבוננו למש מעבד עם ליבת **MIPS**, התומך במיפוי זכרן **0/I** ובקבלת פסיקות חייזניות וניהול רוטינות פסיקה. בחרנו למש את סעיף הבונוס המשלב ליבת **Pipelined MIPS CPU** במקום ליבת **Single Cycle**. ליבת ה-*Pipeline* ממומשת ע"י ארכיטקטורת **MIPS** בעלת 5 שלבים התומכת בזיהוי וטיפול בעיות של **Data Dependencies** תוך שימוש ביחידות **Forwarding Unit** ו-**Hazard detection**. את המערכת הכללית שנבנו צרפנו ובדקו בשכבה ה- **Altera FPGA board DE10**. המשלב רכיבי פריפריה שונים כגון כפתורים, לדימ, מסכי *segments* – 7 ועוד.

### MCU

קובץ ה-MCU משמש כ"מעטפת" המכילה את ארבעת המודולים הבאים:  
*PipelinedMIPS, GPIO, InterruptController, BasicTimer*  
בהתאם לאיור הבא:

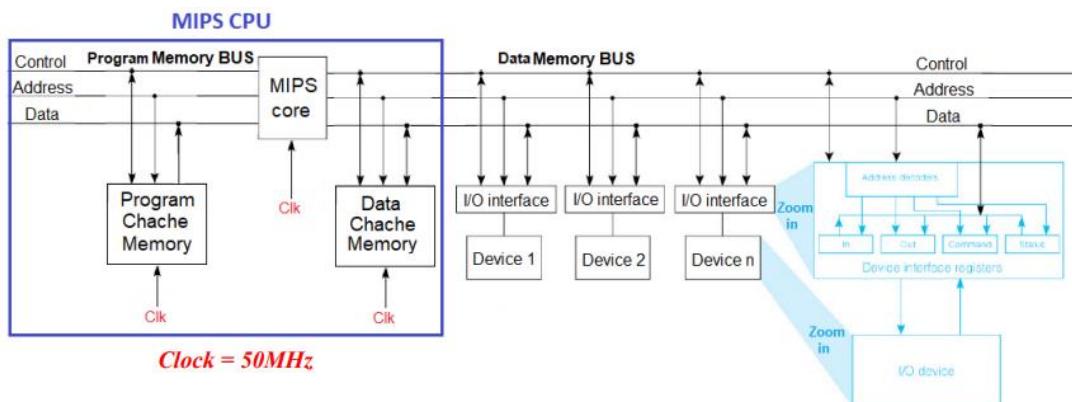


Figure 1: MCU System Architecture

קובץ זה מכיל בנוסף במסותף הכלול 3 תתי באסים: מידע, כתובות וקרה. הbasics המשותף הינו אמצעי התקשרות שבין ליבת הAKER לרכיבים הפריפריאליים.



## PipelinedMIPS

קובץ ה-S-Block PipelinedMIPS מושם כ"מעטפת" המכילה את שבעת המודולים הבאים:

*IFETCH, IDECODE, EXECUTE, DMEMORY,  
CONTROL, Forwarding, HazardDetection*

בהתאם לאיור הבא:

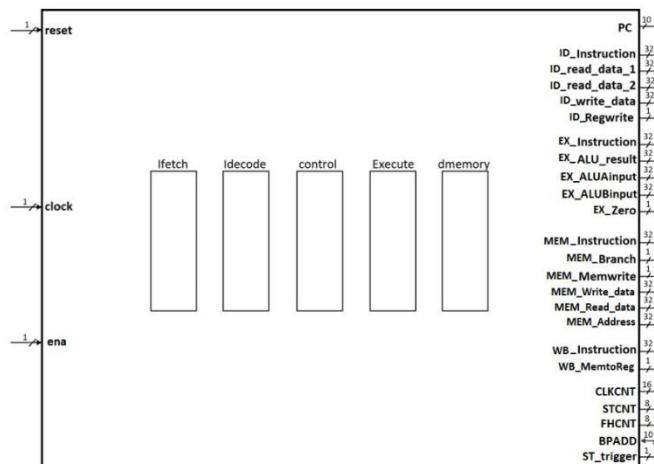


Figure 2: Pipelined MIPS architecture top entity

הכניסות של המודול הן:

סיגナル *reset*, המאפס את ערך ה *PC* בשלב *Fetch*.

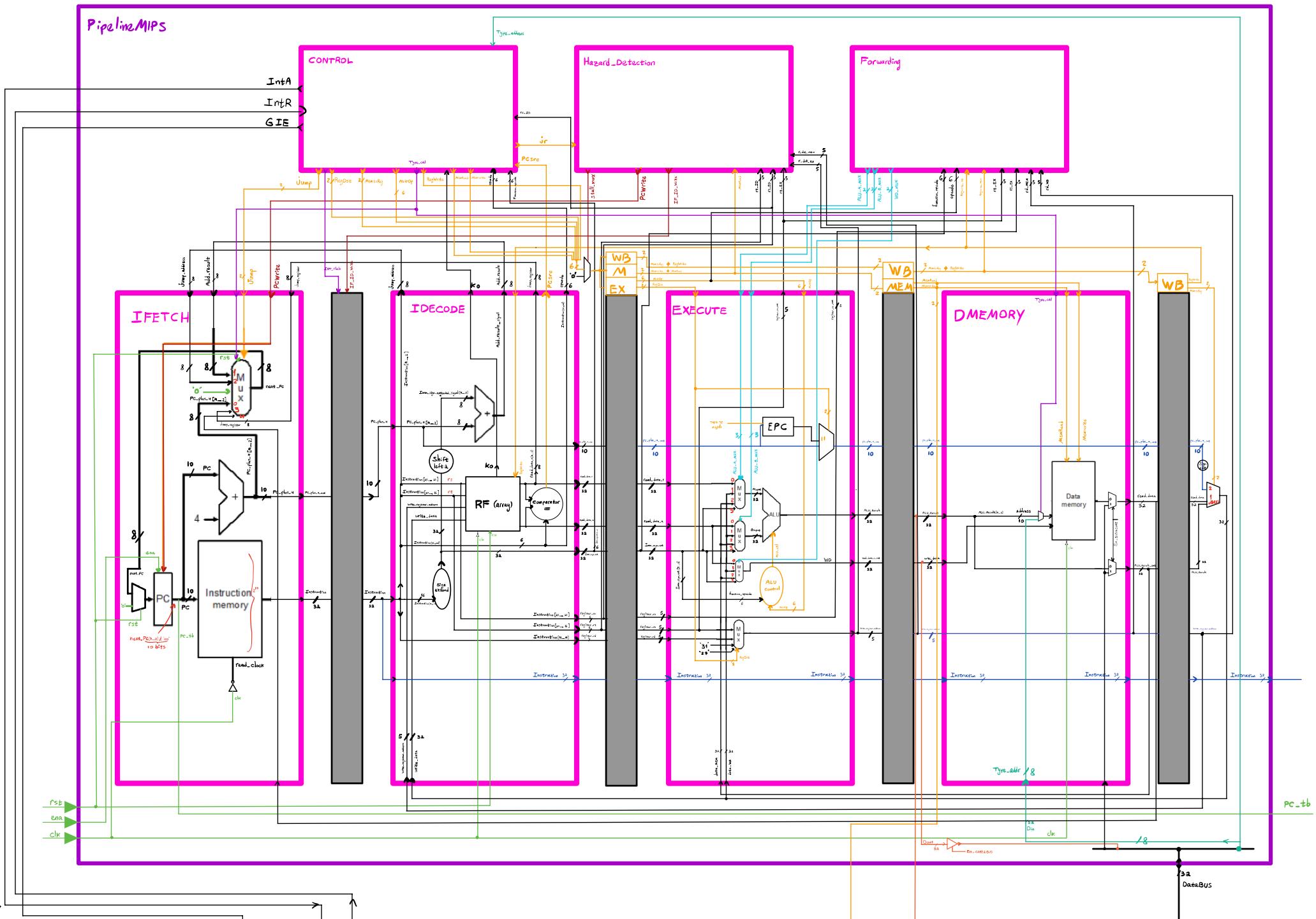
סיגナル *clock* המהווה את אות השעון הנכנס לרכיבים הסינכרוניים בכלל שכבות המערכת.

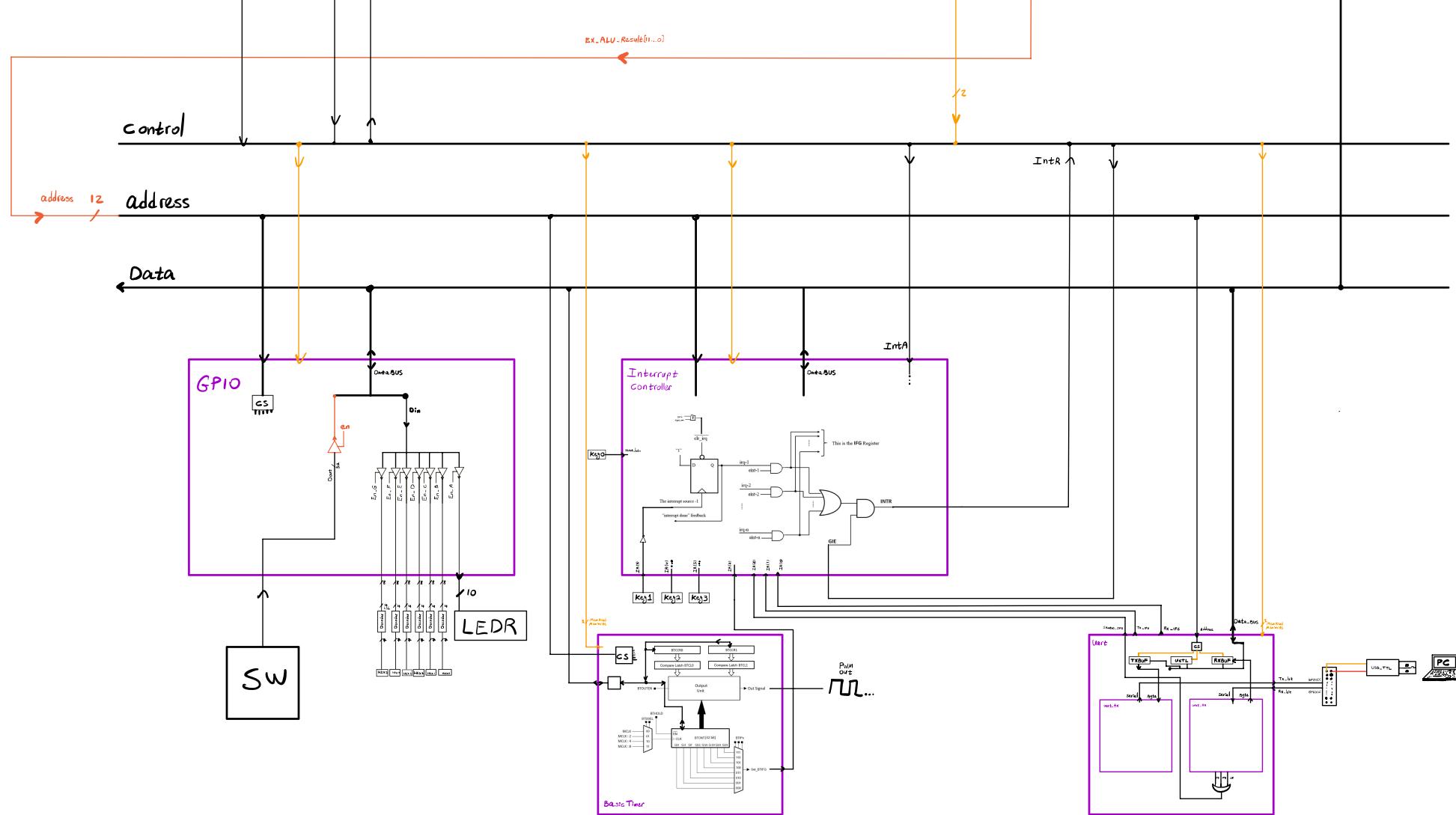
סיגナル *ena* אשר מאפשר העברת מידע משכבה לשכבה.

מצאי המודול הם רכבים ומשמשים לניתוח המערכת, בין אם בשלב הוריפיקציה הפורקציונלית בתוכנת ה-*Modelsim*, ובין אם בשלב הוריפיקציה החשמלית דרך ה-*Quartus Signal Tap* שבתוכנת ה-*Quartus*.

בשני העמודים הבאים מתואר שרטוט מלא של מודול ה-**MCU** עם כל קוי המידע ואוטות הבקרה הקיימים במערכת, הכולל בתוכו בין היתר את ליבת ה-**PipelinedMIPS**, ואת המודולים הפריפריאלים כגון *SPI*, בקר פסיקות וטיימר בסיסי:

## PipelineMIPS







**דיאגרמת בלוקים (Technology Map Viewer – Top Fitting)  
ותצוגת :RTL Viewer**

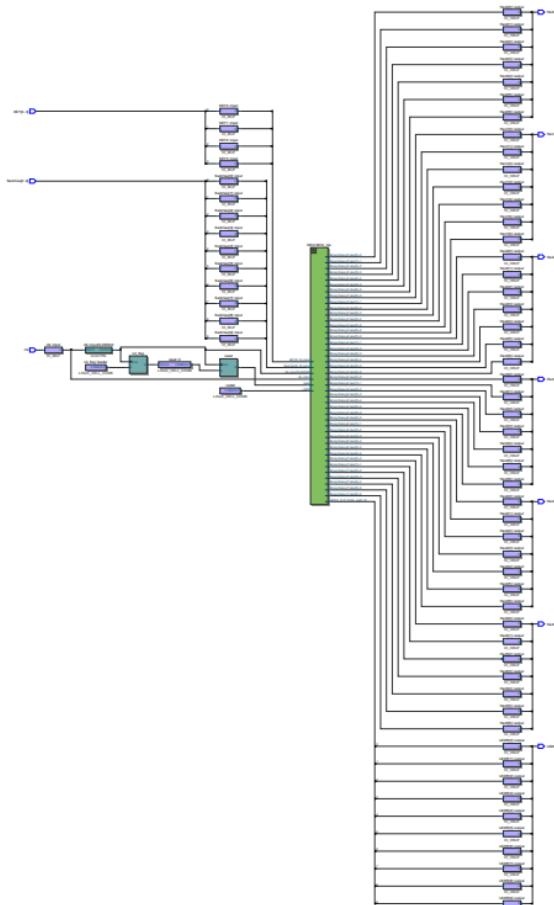


Figure 3: Technology Map Viewer – Top Fitting

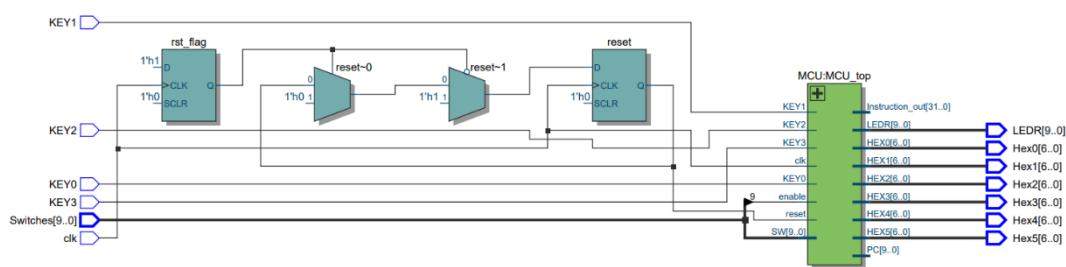


Figure 4: RTL Viewer



דיאגרמות בלוקים עבור ליבת ה-  
*PipelinedMIPS* : (Technology Map Viewer – Top Fitting)

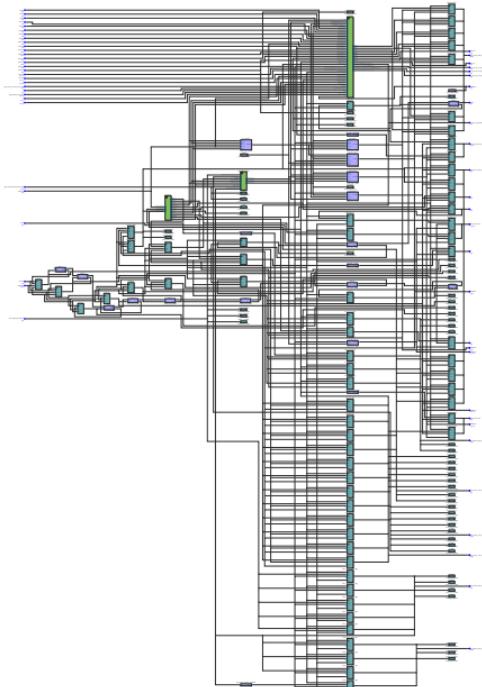


Figure 5: Control, Hazard, Fetch

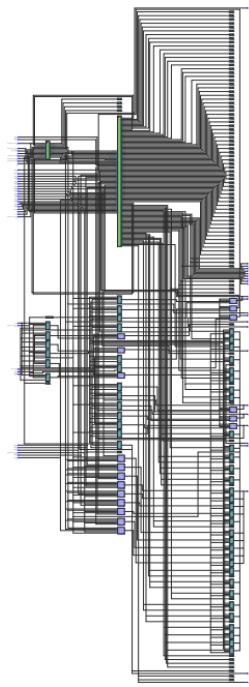


Figure 6: Execute

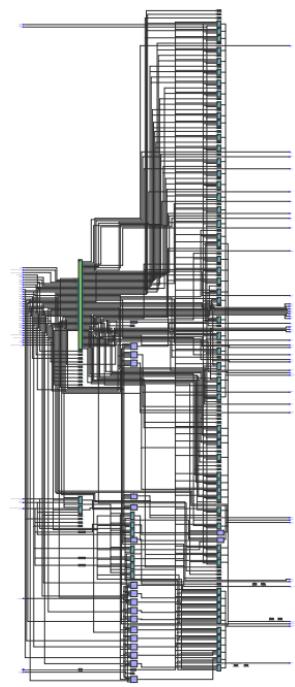


Figure 7: Decode

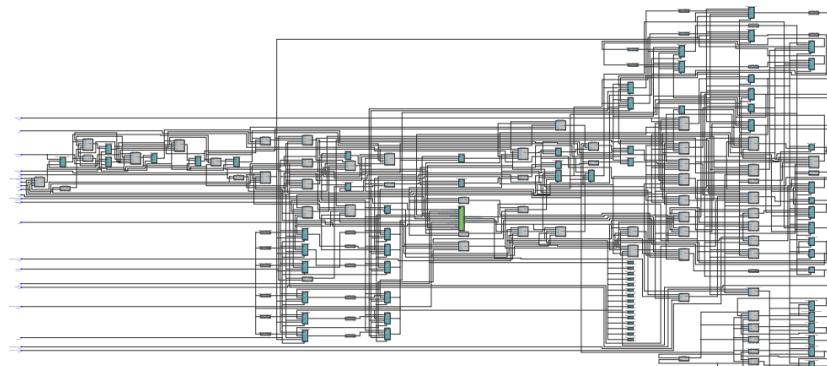


Figure 8: Dmemory



## שימוש ברכיבים חומרתיים – Logic Usage

Analysis & Synthesis Summary		
<<Filter>>		
Analysis & Synthesis Status	Successful - Fri Jul 21 17:59:40 2023	
Quartus Prime Version	22.1std.1 Build 917 02/14/2023 SC Lite Edition	
Revision Name	FinalProjectQuartus	
Top-level Entity Name	QuartusTop	
Family	Cyclone V	
Logic utilization (in ALMs)	N/A	
Total registers	1459	
Total pins	67	
Total virtual pins	0	
Total block memory bits	98,304	
Total DSP Blocks	1	
Total HSSI RX PCSS	0	
Total HSSI PMA RX Deserializers	0	
Total HSSI TX PCSS	0	
Total HSSI PMA TX Serializers	0	
Total PLLs	0	
Total DLLs	0	

Figure 9: MCU Synthesis Summary

Analysis & Synthesis Resource Usage Summary		
Resource	Usage	
1 Estimate of Logic utilization (ALMs needed)	1480	
2		
3 Combinational ALUT usage for logic	1719	
4 -- 7 input functions	10	
5 -- 6 input functions	1006	
6 -- 5 input functions	243	
7 -- 4 input functions	209	
8 -- <=3 input functions	251	
9 Dedicated logic registers	1459	
10 I/O pins	67	
11 Total MLAB memory bits	0	
12 Total block memory bits	98304	
13 Total DSP Blocks	1	
14 Maximum fan-out node	clk<input>	
15 Maximum fan-out	1519	
16 Total fan-out	15578	
	Average fan-out	4.61

Figure 10: MCU Usage – Resource Usage Summary

## מסלול קריטי – Critical Path ותדר שעון מקסימלי:

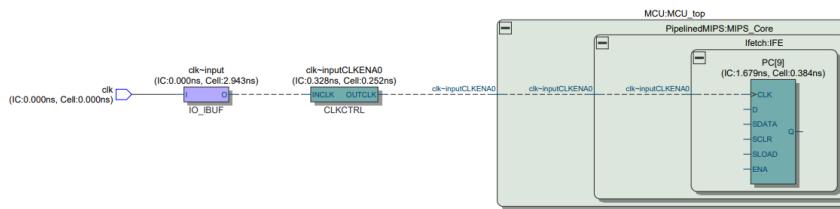


Figure 11: Top Critical Path

ובאופן מפורט:

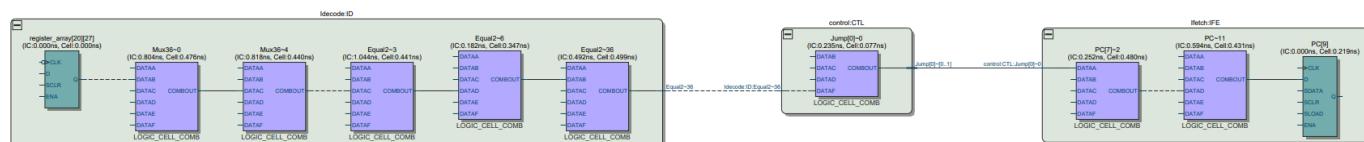


Figure 12: Top Critical Path – Chosen Path

ניתן לראות כי הנטייב הארוך ביותר הינו המסלול המתחילה בקובץ הרגיסטרים, אל תור הקומפרטורי, אל קו הבקרה *jump* הנכנס ליחידת הבקרה, אל תור ה-*mux*.  
בוחר את הכתובת לקפיצה שנכנסת לרגיסטר ה-PC שכשלב ה-*Fetch*.  
נתיב קריטי זה מכתיב את תדר השעון המקסימלי של המערכת, כפי שנראה בהמשך.

Slow 1100mV 85C Model Fmax Summary				
	Fmax	Restricted Fmax	Clock Name	Note
1	58.01 MHz	58.01 MHz	clk	

Figure 13: MCU maximal frequency,  $f_{max}$



## :Functional Verification – Pipeline Core .2

על מנת לבדוק את תקינות ליבת ה-*Pipeline*, ביצענו את הסימולציה להלן:  
הרכנו מכנית אסםכלי בשפת *MIPS* המחברת שתי מטריצות נתונות לתוך מטריצה  
יעד. את הקוד הנתון בשפת C תרגמנו לשפת אסםכלי *MIPS*:

```
#define M 4
void addMats(int Mat1[M][M], int Mat2[M][M], int resMat[M][M]){
    define it yourself ...
}

void main(){ //int=32bit
    int Mat1[M][M]={{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16}};
    int Mat2[M][M]={{13,14,15,16},{9,10,11,12},{5,6,7,8},{1,2,3,4}};
    int resMat[M][M];

    addMats(Mat1,Mat2,resMat); // resMat = Mat1 + Mat2
}
```

Figure 14: Matrices addition C program



```
.data
Mat1: .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16
Mat2: .word 13, 14, 15, 16, 9, 10, 11, 12, 5, 6, 7, 8, 1, 2, 3, 4
resMat: .space 64 # 4*4*4

.text
# Function: addMats
addMats:

    la $t0, Mat1      # Move Mat1 address to $t0
    la $t1, Mat2      # Move Mat2 address to $t1
    la $t2, resMat    # Move resMat address to $t2
    lui $t3, 0          # Loop index i

outer_loop:
    lui $t4, 0          # Reset index j
    inner_loop:
        lw $s0, 0($t0)    # Load Mat1[i][j]
        lw $s1, 0($t1)    # Load Mat2[i][j]
        add $s2, $s0, $s1  # Add Mat1[i][j] and Mat2[i][j]
        sw $s2, 0($t2)    # Store the result in resMat[i][j]

        addi $t0, $t0, 4    # Move to the next element of Mat1
        addi $t1, $t1, 4    # Move to the next element of Mat2
        addi $t2, $t2, 4    # Move to the next element of resMat

        addi $t4, $t4, 1    # Increment index j
        slti $t5, $t4, 4    # Check if j < M
        bne $t5, $zero, inner_loop
        addi $t3, $t3, 1    # Increment index i
        slti $t6, $t3, 4    # Check if i < M
        bne $t6, $zero, outer_loop
        nop
```

Figure 15: Matrices addition MIPS program

תרגומם קוד האסםכלי לפקודות לאחר ביצוע הרצתה:  
(מרחיב הכתובות של סגמנט המידע מתחילה מכתובת 0, בעוד שסגמנט הורכנית מ-3000h).

Text Segment				Source
Bkpt	Address	Code	Basic	
	0x00003000	0x20080000 addi \$8,\$0,0x00000000	10: la \$t0, Mat1 # Move Mat1 address to \$t0	
	0x00003004	0x20090040 addi \$9,\$0,0x00000040	11: la \$t1, Mat2 # Move Mat2 address to \$t1	
	0x00003008	0x200a0080 addi \$10,\$0,0x00000080	12: la \$t2, resMat # Move resMat address to \$t2	
	0x0000300c	0x3c0b0000 lui \$11,0x00000000	13: lui \$t3, 0 # Loop index i	
	0x00003010	0x3c0c0000 lui \$12,0x00000000	16: lui \$t4, 0 # Reset index j	
	0x00003014	0x8d100000 lw \$16,0x00000000(\$8)	18: lw \$s0, 0(\$t0) # Load Mat1[i][j]	
	0x00003018	0x8d310000 lw \$17,0x00000000(\$9)	19: lw \$s1, 0(\$t1) # Load Mat2[i][j]	
	0x0000301c	0x2119020 addi \$18,\$16,\$17	20: add \$s2, \$s0, \$s1 # Add Mat1[i][j] and Mat2[i][j]	
	0x00003020	0xad520000 sw \$18,0x00000000(\$10)	21: sw \$s2, 0(\$t2) # Store the result in resMat[i][j]	
	0x00003024	0x21080004 addi \$8,\$8,0x00000004	23: addi \$t0, \$t0, 4 # Move to the next element of Mat1	
	0x00003028	0x21290004 addi \$9,\$9,0x00000004	24: addi \$t1, \$t1, 4 # Move to the next element of Mat2	
	0x0000302c	0x214a0004 addi \$10,\$10,0x0000... 25:	addi \$t2, \$t2, 4 # Move to the next element of resMat	
	0x00003030	0x218c0001 addi \$12,\$12,0x0000...	27: addi \$t4, \$t4, 1 # Increment index j	
	0x00003034	0x298d0004 slti \$13,\$12,0x0000...	28: slti \$t5, \$t4, 4 # Check if j < M	
	0x00003038	0x15a0fff6 bne \$13,\$0,0xfffffffff6	29: bne \$t5, \$zero, inner_loop	
	0x0000303c	0x216b0001 addi \$11,\$11,0x0000...	31: addi \$t3, \$t3, 1 # Increment index i	
	0x00003040	0x296e0004 slti \$14,\$11,0x0000...	32: slti \$t6, \$t3, 4 # Check if i < M	
	0x00003044	0x15c0fff2 bne \$14,\$0,0xfffffffff2	33: bne \$t6, \$zero, outer_loop	
	0x00003048	0x00000000 nop	34: nop	

Figure 16: Matrices addition MIPS program – Text Segment (MARS)

Data Segment									
Address	Value (+0)	Value (+1)	Value (+2)	Value (+3)	Value (+4)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00000000	0x00000001	0x00000002	0x00000003	0x00000004	0x00000005	0x00000006	0x00000007	0x00000008	
0x00000020	0x00000009	0x0000000a	0x0000000b	0x0000000c	0x0000000d	0x0000000e	0x0000000f	0x00000010	
0x00000040	0x0000000d	0x0000000e	0x0000000f	0x00000010	0x00000009	0x0000000a	0x0000000b	0x0000000c	
0x00000060	0x00000005	0x00000006	0x00000007	0x00000008	0x00000001	0x00000002	0x00000003	0x00000004	
0x00000080	0x0000000e	0x00000010	0x00000012	0x00000014	0x0000000e	0x00000010	0x00000012	0x00000014	
0x000000a0	0x0000000e	0x00000010	0x00000012	0x00000014	0x0000000e	0x00000010	0x00000012	0x00000014	
0x000000c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	
0x000000e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	

MatA  
MatB  
ResMat

Figure 17: Matrices addition MIPS program - Data Segment (MARS)

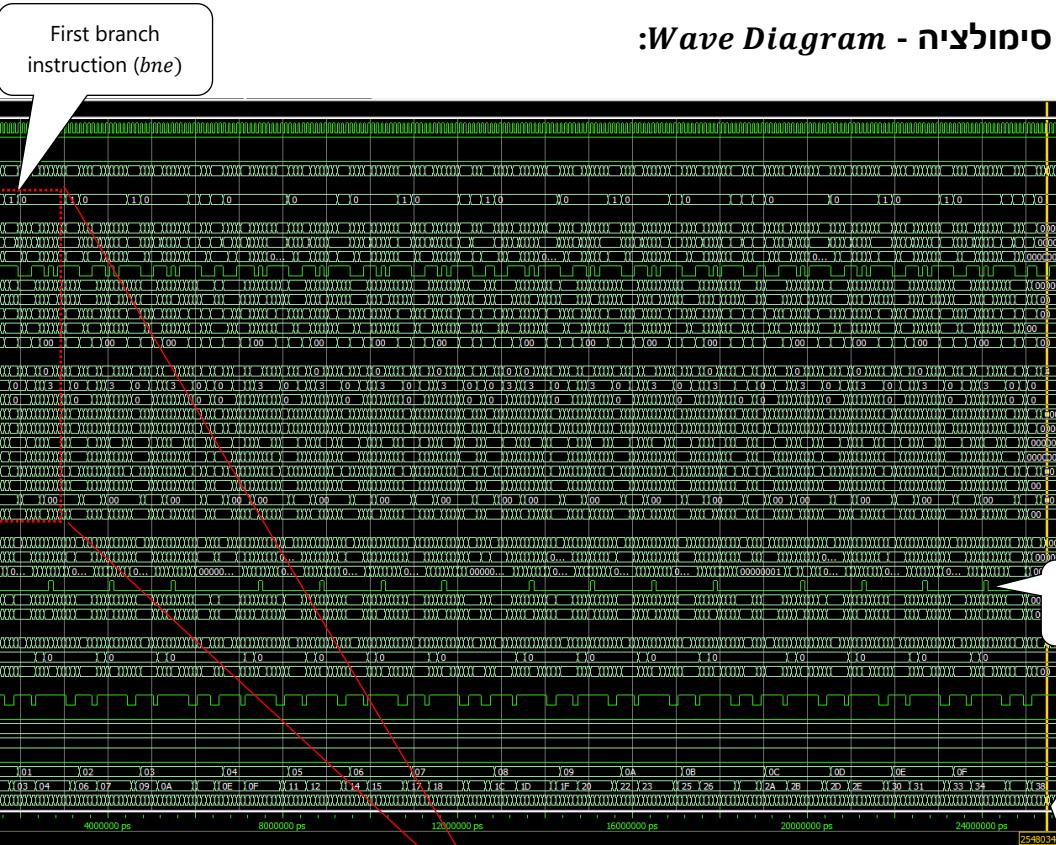


Figure 18: Modelsim Simulation – wave diagram

נתקדם בפקודת ההסתעפות הראשונה (bne).

ניתן לראות באירור 11 כי פקודת ההסתעפות תליה בפקודת *slti* שלפניה (הគותבת לרגיסטר 13 שבוינו התנאי לקפיצה). בנוסף פקודת *slti* תליה גם היא בפקודת *addi* שלפניה, המשנה את ערך רגיסטר 12. על כן, במקרה זה מותבאים גם פקודת *addi* (מפקודת *Data forwarding* לפקודת *slti*), וגם *stall* כפול, המבטיח שפקודת *bne* תשתמש בערך רגיסטר 13 המעודכן.

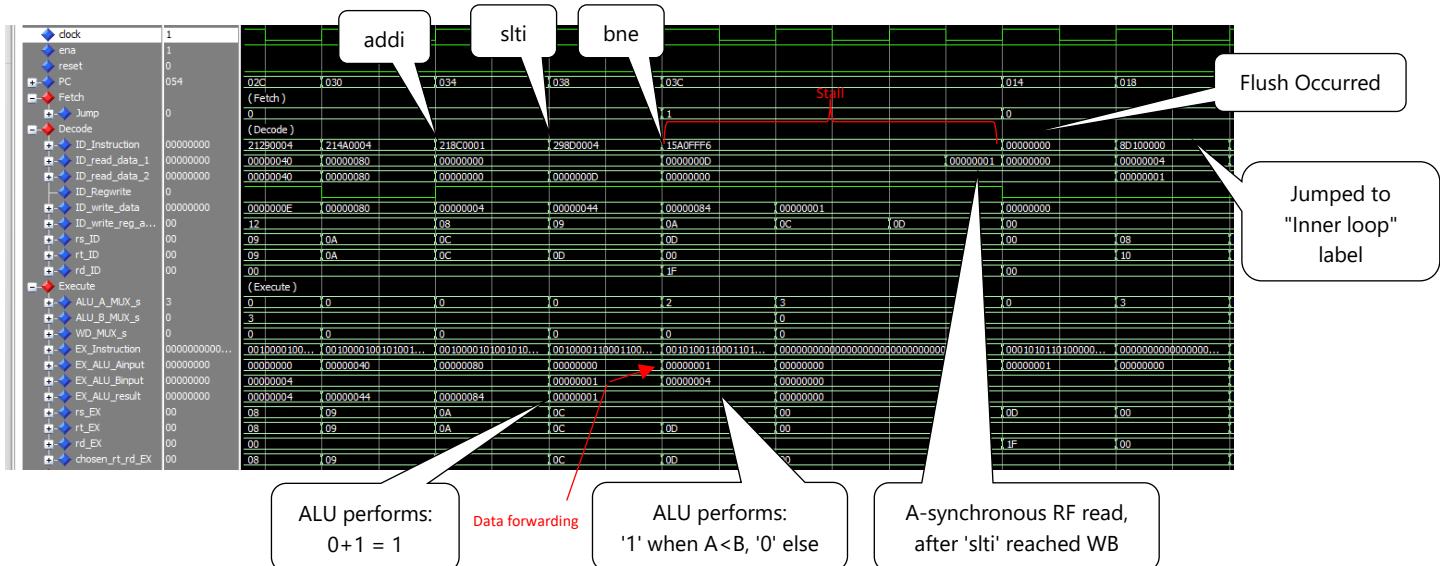


Figure 19: Modelsim Simulation – wave diagram (zoomed)



### סימולציה - *Memory List*

לאחר ריצת הסימולציה, נתבונן בזיכרון התכנית אליו נכתבת מטריצת היעד:

00000047	00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000	00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000	ResMat
0000003f	00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000	00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000	
00000037	00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000	00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000	
0000002f	00000014 00000012 00000010 0000000E 00000014 00000012 00000010 0000000E	00000014 00000012 00000010 0000000E 00000014 00000012 00000010 0000000E	
00000027	00000014 00000012 00000010 0000000E 00000014 00000012 00000010 0000000E	00000014 00000012 00000010 0000000E 00000014 00000012 00000010 0000000E	
0000001f	00000004 00000003 00000002 00000001 00000008 00000007 00000006 00000005	00000004 00000003 00000002 00000001 00000008 00000007 00000006 00000005	MatB
00000017	0000000C 0000000B 0000000A 00000009 00000010 0000000F 0000000E 0000000D	0000000C 0000000B 0000000A 00000009 00000010 0000000F 0000000E 0000000D	
0000000f	00000010 0000000F 0000000E 0000000D 0000000C 0000000B 0000000A 00000009	00000010 0000000F 0000000E 0000000D 0000000C 0000000B 0000000A 00000009	MatA
00000007	00000008 00000007 00000006 00000005 00000004 00000003 00000002 00000001	00000008 00000007 00000006 00000005 00000004 00000003 00000002 00000001	

Figure 20: Modelsim Simulation – Memory List

### בדיקה ביצועים – *Performance Test Case*

בשלב זה, ברכינו לבודק את הביצועים והפונקציונליות של המערכת הדיגיטלית שבנו.

בשלב הסימולציה ראיינו כי התכנית ביצעה: 15 פקודות *Flush*, 56 פקודות *stall* ו-253 מחזורי שעון עד לסיום התכנית. לפי הנוסחה שלמדנו, ה-IPC של התכנית מחושב באופן הבא:

$$IPC = \frac{CLKCNT - (STCNT + 4 + FLCNT \cdot dept)}{CLKCNT} = \frac{253 - (56 + 4 + 15 \cdot 1)}{253} = \frac{178}{253} = 0.7035573$$

$$CPI = \frac{1}{IPC} = 1.421348$$

כאשר  $dept = 1$  מייצג את עומק הפיפליין הנמוך בפקודת *Flush* (כיוון שמיינשנו את ה-*roper* בשלב *Decode*, הוראת *Flush* מאפשרת רק את שלב *Comparator*).

### תדר שעון $f_{max}$ מקסימלי:

באופן כללי, ראיינו כי תדר השעון המקסימלי מושפע מזמן ההשהיה של הרכיבים השונים המשתתפים בمعالג. הקשר בין זמן השעון המינימלי לזמן ההשהיה נתון

$$f_{max} = \frac{1}{T_{min}} \text{ ומכאן שהתדר המקסימלי: } T_{min} = t_{cq} + \max_i \{t_{pd(CL_i)}\} + t_{su} \text{ ע"י:}$$

Slow 1100mV 85C Model Fmax Summary				
	Fmax	restricted Fmax	clock Name	Note
1	111.1 MHz	111.1 MHz	alte..._tck	

Figure 21: PipelinedMIPS maximal frequency,  $f_{max}$



### :Electrical Verification – Signal Tap

בשלב זה, צרכנו את התכנית לשכט-*FPGA* וביצענו סימולציה *Signal Tap* בזמני אמת את עריכם של הסיגנלים בשכט ה-*DE10*, ע"י שימוש במשאים פנימיים כמו בלוקי זכרון ורכיבים לוגיים.

הגדכנו את הסיגנל *ST\_Trigger* כתנאי ליצירת טריגר לסימולציה, קבענו את עריכם של המתגים על גבי השכט לצורך קביעת נקודת עצירה, לביצוע השהיה אחורי חיבור של ארבעת האיברים הראשונים במטריצה (סוף הלולאה הפנימית), ולהצנו על כפטור 0*PB0* על מנת להתחילה את התכנית. (בנוסף הגדכנו שבעלית סיגנל ה-*ST\_Trigger* ה-*en*, יתאפשר קו ה-*en* ובכך יפסיק את ריצת התכנית לאחר הגיעו לנקודת ה-*Breakpoint*).

### שלב הצריבה לשכט – Code Programming

בסיום הקומpileציה ולאחר שמצאנו את התדר המקסימלי, כתבנו קוד – *QuartusTop* המקשר בין שככת ה-*MIPS Pipeline* לרכיבים הפריפריאליים – לחץ 0*KEY0* שאחראי על ביצוע *Reset* לתוכנית (איפוס כתובת ה-*PC*), – מערך מתגים 7*SW* – 0*WS*, שאחראים על קביעת כתובת *Breakpoint* למערכת, – שאליה תרוץ התכנית ותעוצר (בקבלת סיגנל טריגר *ST\_Trigger*).

לאחר ביצוע הקומpileציה לקובץ, ביצענו *Pin Assignment* וצרכנו את הקוד ע"י מצלב *Programmer*.



להלן תוצאות הסימולציה על גבי חלון *Signal Tap*, עבור התכנית *AddMat*:

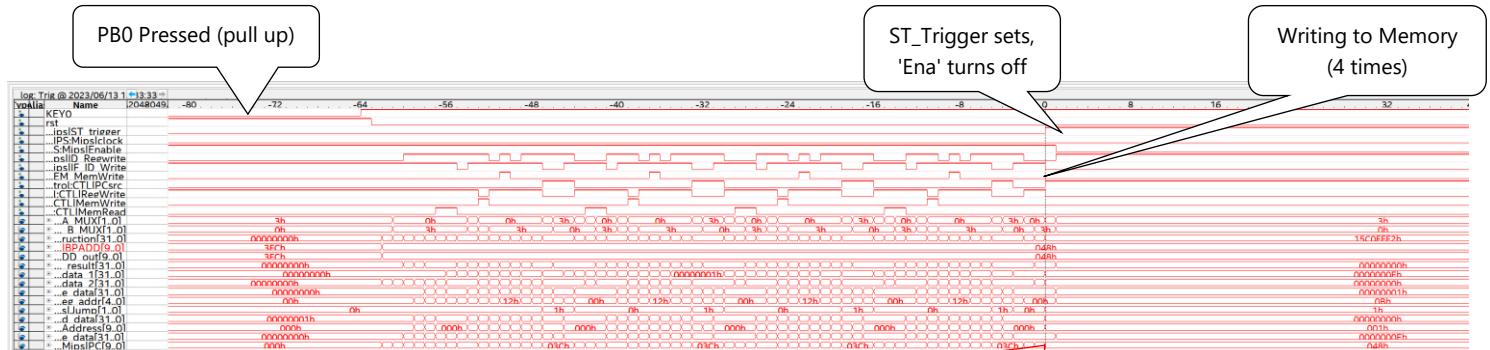


Figure 22: Signal Tap simulation

Zoom In

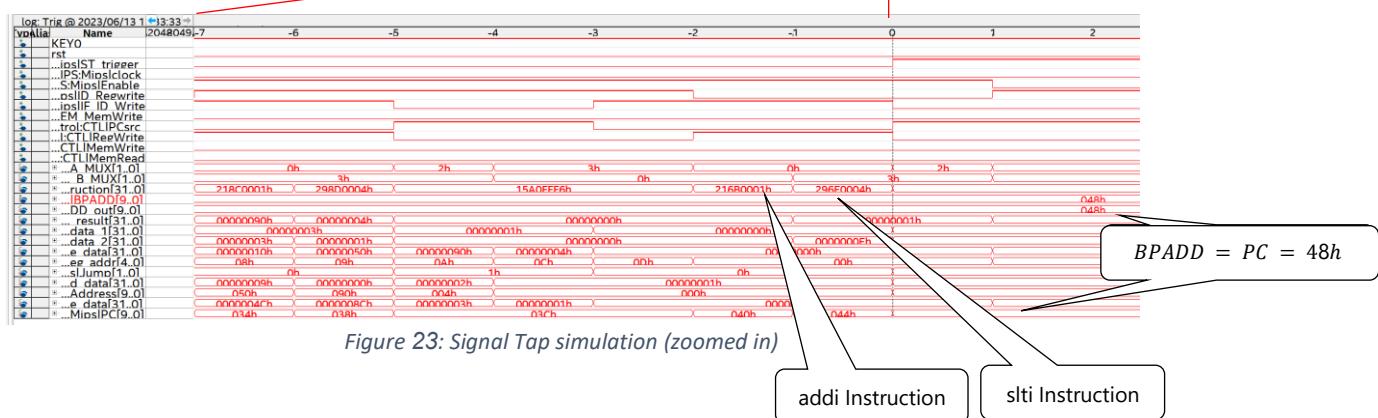


Figure 23: Signal Tap simulation (zoomed in)



## בדיקות ביצועים – Performance Test Case

בSIMOLCZIA שביצענו ב-*Modelsim* ראיינו כי:

$$IPC = 0.7035573 \Rightarrow CPI = \frac{1}{IPC} = 1.421348$$

נחשב את ה-*IPC* המתקבל בסימולציה ה-*Signal Tap*. להלן אירור המתרחשת חישוב ה-*IPC* על ידי סיבוב הריצה שלמה וביצוע מלא של הפקת *AddMat*:



Figure 24: Full Signal Tap simulation

Zoom in

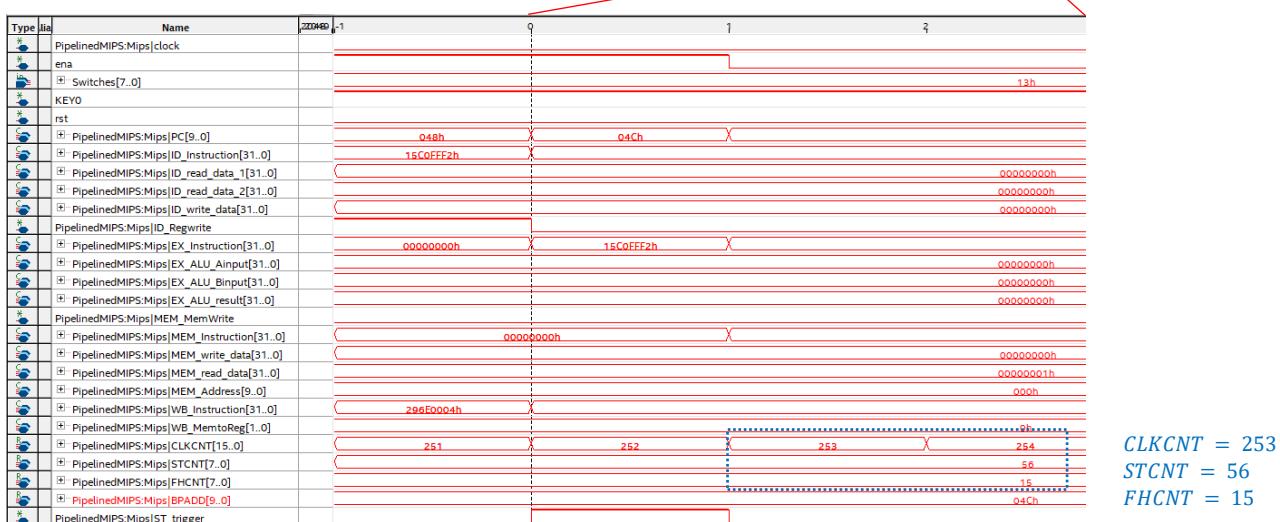


Figure 25: Full Signal Tap simulation (zoomed in)

נתמך

בריגיסטרי המניה מיד לאחר עליית אות הטריאגר, ונשים לב כי קיבלנו את אותו הערכיהם הצפויים משלב הוריפיקציה הפונקציונלית ב-*ModelSim*.

כלומר גם כאן קיבל כי

$$IPC = 0.7035573 \Rightarrow CPI = \frac{1}{IPC} = 1.421348$$



## Theoretical IPC Calculation

נסתכל כעת על תכנית MIPS המחברת שתי מטריצות. נאתר ונסמן את כל השורות בהן יש תלות בין הפקודות (*Data/Control Dependencies*)

```
.data
Mat1: .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16
Mat2: .word 13, 14, 15, 16, 9, 10, 11, 12, 5, 6, 7, 8, 1, 2, 3, 4
resMat: .space 64 # 4*4*4

.text
# Function: addMats
addMats:

    la $t0, Mat1      # Move Mat1 address to $t0
    la $t1, Mat2      # Move Mat2 address to $t1
    la $t2, resMat    # Move resMat address to $t2
    lui $t3, 0          # Loop index i

outer_loop:
    lui $t4, 0          # Reset index j
    inner_loop:
        lw $s0, 0($t0)    # Load Mat1[i][j]
        lw $s1, 0($t1)    # Load Mat2[i][j]
        add $s2, $s0, $s1  # Add Mat1[i][j] and Mat2[i][j]
        sw $s2, 0($t2)    # Store the result in resMat[i][j]

        addi $t0, $t0, 4   # Move to the next element of Mat1
        addi $t1, $t1, 4   # Move to the next element of Mat2
        addi $t2, $t2, 4   # Move to the next element of resMat

        addi $t4, $t4, 1   # Increment index j
        slti $t5, $t4, 4   # Check if j < M
        bne $t5, $zero, inner_loop

        addi $t3, $t3, 1   # Increment index i
        slti $t6, $t3, 4   # Check if i < M
        bne $t6, $zero, outer_loop
    nop
```

**RAW**  
במקרה זה אין צורך ביביצוע Forwarding שפקודות הושם מתבצעות לאחר הכתיבת לרגיטרים.

**RAW**  
במקרה זה פקודת *add*-*add* משתמשת ברגיסטרים שונים ונכתב אליו מיד ערך עדכני "lw" בפקודת *load*. *lw* במקורה זה נבעץ *forwarding* בודד ואחריו *stall* לבני פקודת *add*.

**RAW**  
במקרים אלו פקודת *bne* משתמשת ברגיסטר שטרם נכתב אליו מיד ערך עדכני ".slti" ע"פ קוווטה *tslti*. בתום *tslti*, תליה גם היא בפקודת *addi* ב*forwarding* *stall* ל*add* *tslti*, *bne*, *flush*-*flush*, *bne* וס"מ-ב*flush* בשל ההזלהה בפקודת ההסתעופה.

Figure 26: MIPS program – Theoretical IPC analysis

בסה"כ ניתן לראות כי בכל לולאה פנימית מתבצעות 3 פעולות *stall*. לאחר 4 לולאות פנימיות מתבצעות עוד 3 פעולות *stall* של הלולאה החיצונית.

הלולאה החיצונית מתבצעת 4 פעמים גם כן. לכן בסה"כ:

$$STCNT = \left( \frac{\text{inner}}{(1+2) \cdot 4} + \frac{\text{outer}}{2} \right) \cdot \frac{\text{outer}}{4} = 56$$

כמות פעולות *flush* : בכל לולאה פנימית (פרט לאחרונה) מתבצעת פעולה *Flush*, ובכל לולאה חיצונית (פרט לאחרונה) מתבצעת פעולה *flush* נוספת. בסה"כ:

$$FHCNT = \left( \frac{\text{inner}}{1 \cdot 3} + \frac{\text{outer}}{1} \right) \cdot \frac{\text{outer}}{3} + \frac{\text{last inner}}{3} = 15$$

כמות מחזורי השעון הכלולות:

$$CLKCNT = 4 + (1 + (10 \cdot 4) + 3) \cdot 4 + 1 + \frac{ST}{56} + \frac{FH}{15} + 1 = 253$$

לכן בסה"כ נקבל *IPC* זהה לשתי הסימולציות שבייצעו.



### :GPIO . 3

בפרויקט זה נדרשנו לתמוך בMUX עם חומרת *GPIO*, אשר משתמש באופן וירטואלי במרחב הכתובות  $0x800$  ומעלה כמתואר להלן:

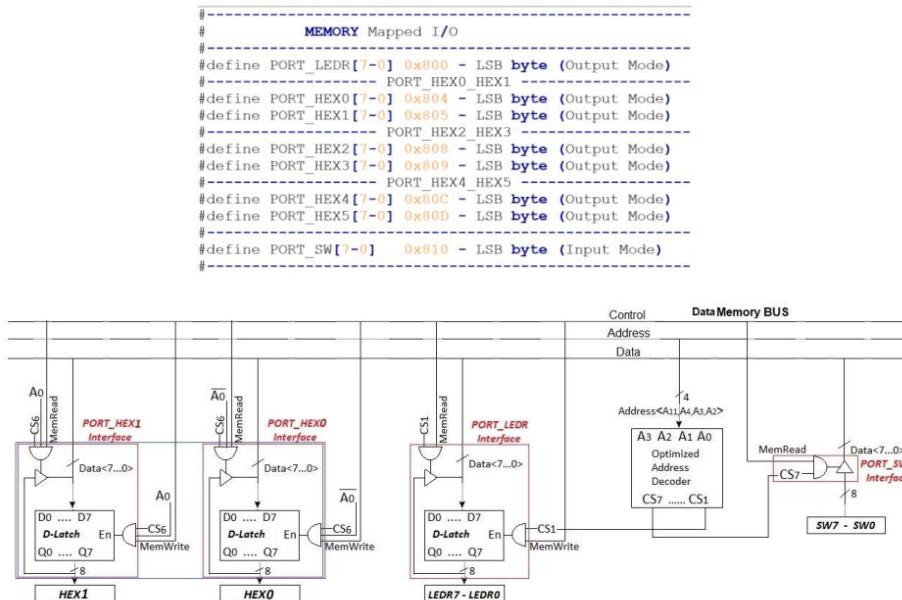


Figure 27: Primitive GPIO peripheral connection using Memory Mapped I/O approach

הרכיבים המוגדרים כפלט של המערכת הם הלדים ותצוגות *Hex* – 7, HEX1, HEX0, LEDR7-LED0. מערך פסיקות חיצונית (הינו מערך מתגים *SW7* – *SW0*) מושם למערכת (שאינו במסגרת פסיקות חיצונית) הינו מערך מתגים *SW7* – *SW0*. שני תנים לקריאה, ואחראים בין היתר (בפעולת *8WS*) על קביעת כתובות למערכת, שאליה תרוץ התכנית ותעוזר (בקבלת סיגנל טריגר *Breakpoint* או *ST\_Trigger*).

במודול *MCU* הגדרנו את מתג *9WS* כאחראי על הפעלת קו הבקרה *enable* שמאפשר את ריצת התכנית.

התקשורת שבין מודול זה לבין ליבת *MIPS* נעשתה על גבי הבאס המשותף. כאשר בשלב ה-*EX* חושבה כתובת השיכת למרחב הכתובות של *h-0*, *GPIO*, בשלב ה-*MEM* תבוצע עקיפה של זכרון המידע והערך שעיל הבאס יהיה הערך הרלוונטי. בפקודות הכתובות ל-*LED7* או ל-*LED0* (*9WS* כගו  $0x804$  ו-*t0* \$t0, מודול *h-0* קבע מיהו הרכיב הפריפריאלי שיקבל את המידע ע"י בחירת ה-*Chip Select* – *CS* – המתאים).

מודול זה פשוט למימוש מאחר שאינו מעכ卜 את רצף התכנית ואין משפיע על הפקודות השונות הנמצאות בשלבי הפעילין.



## סימולציות (Signal Tap) וריפיקציה פונקציונלית (Wave Diagram)

בשלב זה הרצינו את ארבעת קבוצי הבדיקה הנתוניים עבור בדיקת מודול ה-GPIO.

Test0: תכנית זו כותבת בלאלה ערכים עולמים לכל רכיבי ה-GPIO.

זמן השהייה הכתיבה (משתנה  $N$  של הלולאה) נתון לשינוי ונבחר לערך  $4 = N$  גם עבור ה-Modelsim וגם עבור Signal Tap (על מנת לראות את השינויים ב-STP):

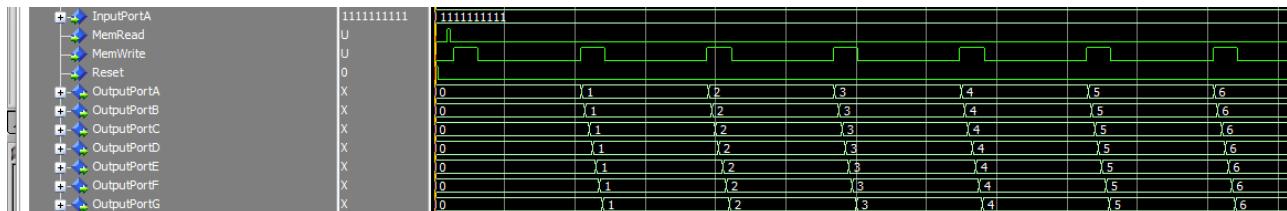


Figure 28: Test0 – Wave diagram

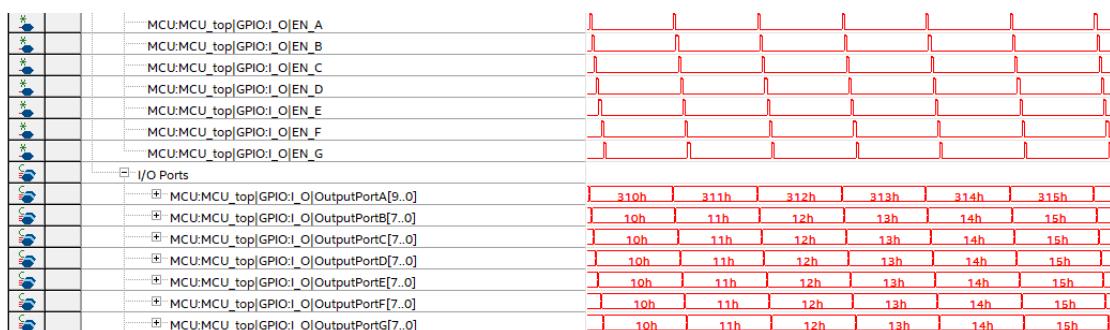


Figure 29: Test0 – Signal Tap

**Test1:** תכנית זו בזקמת את מצב המתגים. אם מtag 0 דולק, אז תבצע מניה מעלה על גבי רכיבי ה-*GPIO*. אחרת, אם מtag 1 דולק תבצע מניה מטה. עבור קובץ זה והלאה, הגדרנו את זמן השהיית הכתיבה לערך  $4 = N$  עبور ה- *Modelsim* ולערך  $0xB71B00 = N$  עبور ה-*Signal Tap* (על מנת לראות שינויים בעין).

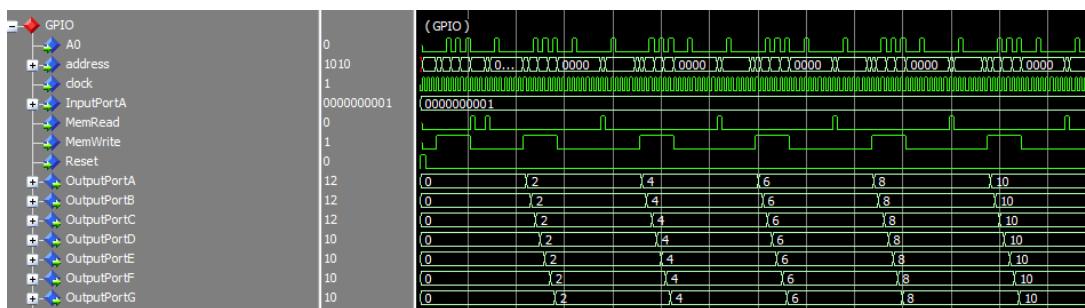


Figure 30: Test1 – Wave diagram (increment)

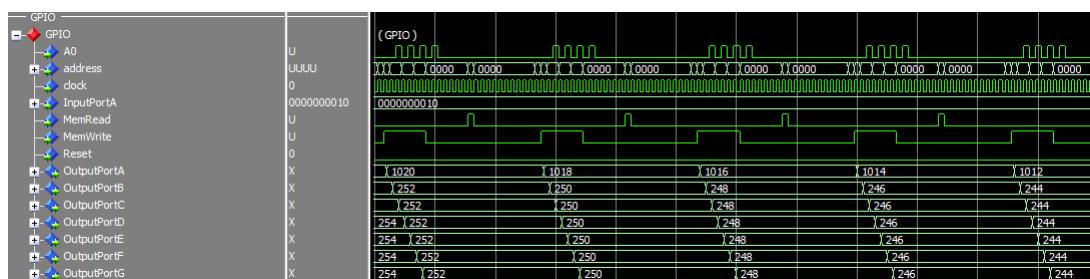


Figure 31: Test 1 – Wave diagram (decrement)

**בתיקית Auxilary Material** שבתקיכת הפרויקט מצורפים סרטונים המעידים על ביצוע תקין של התכנית על גבי שכב ה-*FPGA* עבור קבץ הבדיקה השונים.



תכנית זו בודקת את מצב המתגים. אם מtag 0 דולק, אז תבוצע הכפלת ב-2 על גבי רכיבי ה-GPI0. אחרת, אם מtag 1 דולק, אז תבוצע חילוקה ב-2:

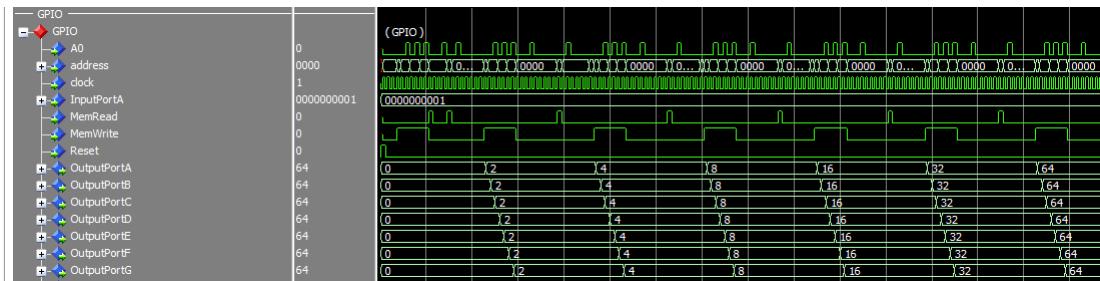


Figure 32: Test2 – Wave diagram (sll)

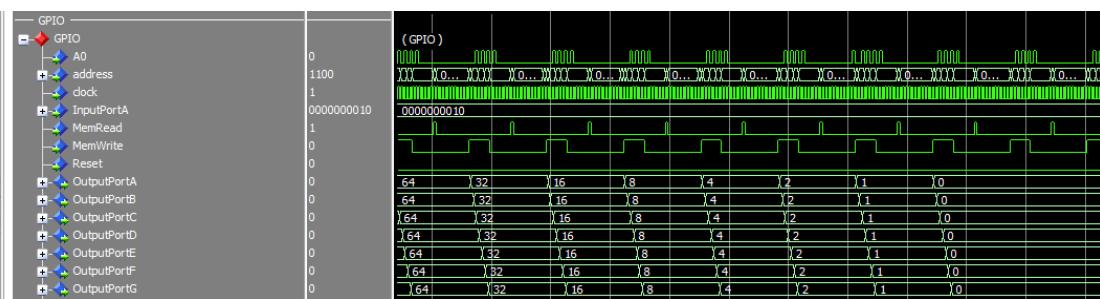


Figure 33: Test2 – Wave diagram (srl)



תכנית זו בודקת את מצב המתגים. אם מותג 0 דולק, אז תבוצע הוספה ב-1  
ואז הכפלת ב-2 על גבי רכיבי ה-GPIO. אחרת, אם מותג 1 דולק, אז יתבצע חיסור ב-1  
ואז הכפלת ב-2:

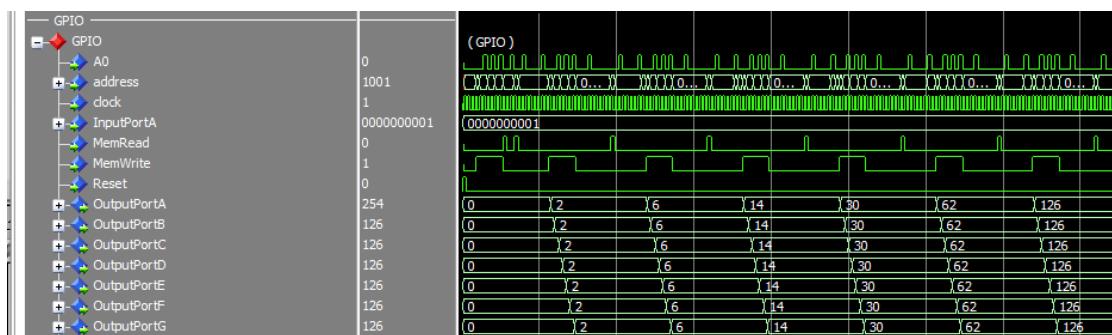


Figure 34: Test3 – Wave diagram (add + sll)

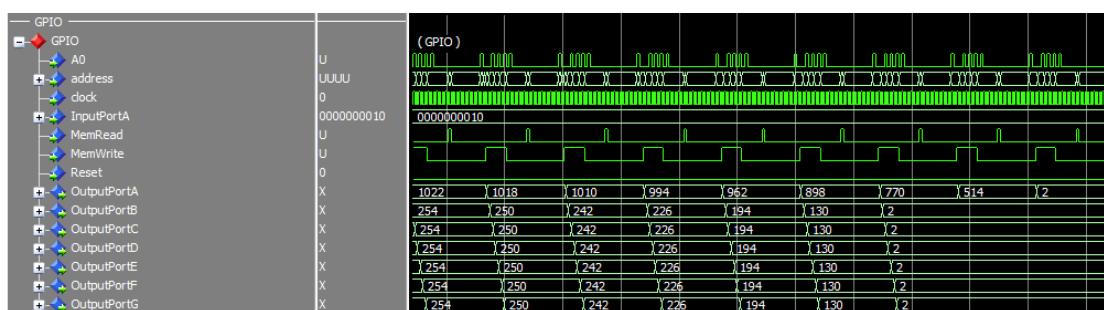


Figure 35: Test3 – Wave diagram (sub + sll)



#### :Interrupt Controller .4

בפרויקט זה נדרשנו לתרום בקבלה פסיקות חיוניות, הנשלטות ומתחודפות על ידי מודול הנקרא *Interrupt Controller*.  
למודול זה קיימים מספר רגיסטרים הנמצאים באופן יירטואלי במרחב הכתובות אשר משתמשת באופן יירטואלי במרחב הכתובות המקורי להלן:

```

#-----#
#define PORT_KEY[3-1] 0x814 - LSB nibble (3 push-buttons - Input Mode)
#
#-----#
#define UCTL          0x818 - Byte
#define RXBF          0x819 - Byte
#define TXBF          0x81A - Byte
#
#-----#
#define BTCTL         0x81C - LSB byte
#define BTCNT         0x820 - Word
#define BTCCR0        0x824 - Word
#define BTCCR1        0x828 - Word
#
#-----#
#define IE            0x82C - LSB byte
#define IFG           0x82D - LSB byte
#define TYPE          0x82E - LSB byte

```

Figure 36: Memory Mapping – with Interrupt Peripherals

מקורות הפסיקה:

- לחץ 0 KEY שאחראי על ביצוע *Reset* לתוכנית (קפיצה לככובת תחילת התוכנית),
- לחצנים 3 KEY1 ~ KEY3 שאחראים על ייצור פסיקות והפעלת רוטינות פסיקה ייעודיות המוגדרות מראש בתוכנית.
- *Basic Timer* – מודול נוסף שיוצג בהמשך אשר אחראי על ייצור אותן מוחזרים המשמשים הן כמקור פסיקה והן כאות *WAKE* חיוני.
- לטימר זה קיימת רוטינת פסיקה ייעודית המוגדרת מראש בתוכנית.
- *UART* (בונוס) – פסיקות המתקבלות דרך תקשורת סריאלית.

התקשורת שבין מודול זה לבין ליבת *MIPS* נעשתה על גבי הבאס המשותף. כתיבה לריגיסטר ה-*IC* נועשית באופן דומה לכתיבה לרכיבי *GPIO* ע"י ביצוע פעולות *sw, wa* למרחב הזיכרון המתאים.

קבלה פסיקות אסינכרוניות מתבצעות באופן הבא:

1. אורם פסיקה מתבצע (לחיצת כפתור/מנית הטימר הגעה לערך החדש וכו').  
בשלב זה קורא ה-*INTR* עליה לאחד ומתריע ל-*CPU* על בקשה פסיקה.
2. ה-*CPU* מבצע פעולות הכנה לפני תחילת הטיפול בפסיקה:  
- ביצוע *stall* עד שהפקודה בשלב ה-*EX* תסיים את הכתיבה לזכרון.  
- ביצוע *Flush* לשלבים שקדמו לשלב *EX*  
- איפוס רגיסטר ה-*GIE* (ביט ה-*LSB* של רגיסטר 0)
- העלאת ערך *readType* לבאס, המצביע על מקום רוטינת הפסיקה הרלוונטית.  
- גיבוי הערך לחזירה ברגיסטר 1 *k* וקפיצה לככובת של רוטינת הפסיקה.
3. ה-*CPU* מבצע את רוטינת הפסיקה
4. בהגעה לפקודת *reti* מתבצע *JR* לרגיסטר 1 *k*. בנוסף ערך רגיסטר *GIE* חוזר ל-1  
לטובת המשך קבלת פסיקות.



### **Basic Timer: .5**

הינו מודול אשר אחראי על ייצור אוטות מחזוריים המשמשים הן כמקור פסיקה והן כאות PWM חיצוני.  
למודול קיימים מספר רגיסטרים:

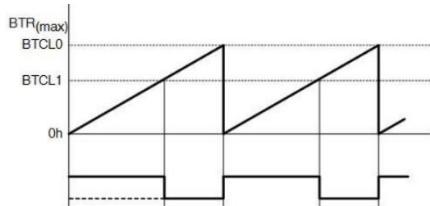


Figure 37: Output Mode: Reset/Set

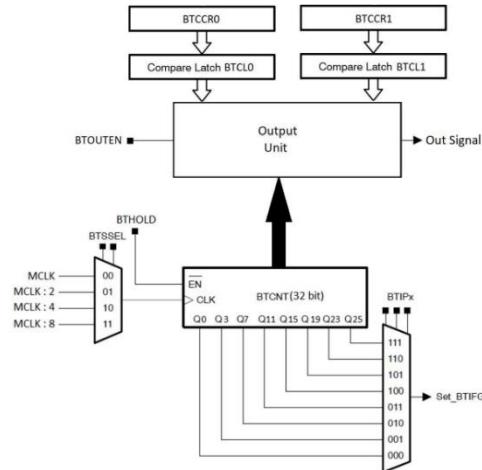


Figure 38: Basic Timer Module

- *BTCTL* – זה רגיסטר השליטה, המכיל את ביטי הבקרה השונים, כגון *BTHOLD, BTSEL, BTIP*.

- *BTCNT* – זה רגיסטר המניע בגודל 32 ביט אשר ערכו גדול בכל עליית שעון (הניתן לבחירה) כאשר *BTHOLD* קבוע. ניתן לדגם באופן פרטני את הביטים השונים של הרגיסטר לקבלת פסיקות בתדר שונה, על פי שינוי ערך ה-*BTIP*.

ה-*C-D* של אות PWM היוצא, *Out Signal*.  
על רצף של הרגיסטר נקבע ישירות ע"י המתכנת כחלק מהקונפוג של הטימר.

בהתגובה של *BTCNT* לערך המעטפות, ערך המוצא של ה-*PWM* מתהפר, כאשר הערך מגיע ל-*CCR0* ערך *BTCNT* מותאפס (ראה איור 37)



## סימולציות (Signal Tap), וריפיקציה פונקציונלית (Wave Diagram)

Test1: תכנית זו בסיסית ואינה משלבת רותיניות פסיקה.

התכוית מגדרה  $9 = X, 16 = Y$  ובעזרת קרייה למספר פונקציות מבצעת את הפסודו הבא:

$$LEDs \leftarrow (X + Y - 4 > 0 ? X + Y - 4 : X + 4)$$



Figure 39:  $LED = 00010101 = 0x15$

Test2: תכנית זו מקנפת טיימר המבצע מניה למשתנה בכל פסיקת שעון. את ערך המשתנה אנו רושמים לתוצאות Hex על פי הכפטור המתאים שנלחש.

- בROTINYNT הפסיקה של כפטור 1 נכתב את הערך של המשתנה **hex1**.
- עבור כפטור 2 נכתב **hex2, hex3**
- עבור כפטור 3 נכתב **hex4, hex5**

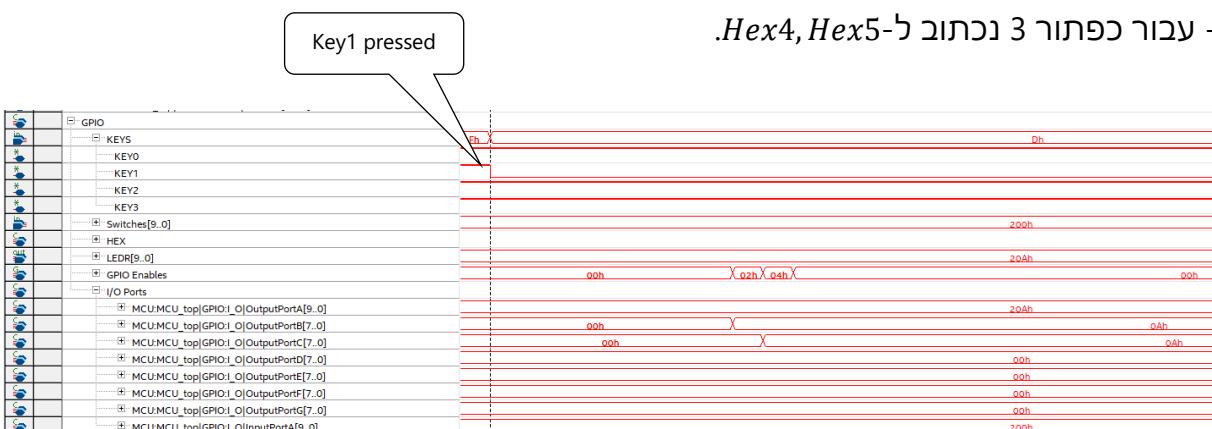


Figure 40: writing 'A' to hex0 and hex1 on KEY1 press – Signal Tap (GPIO)

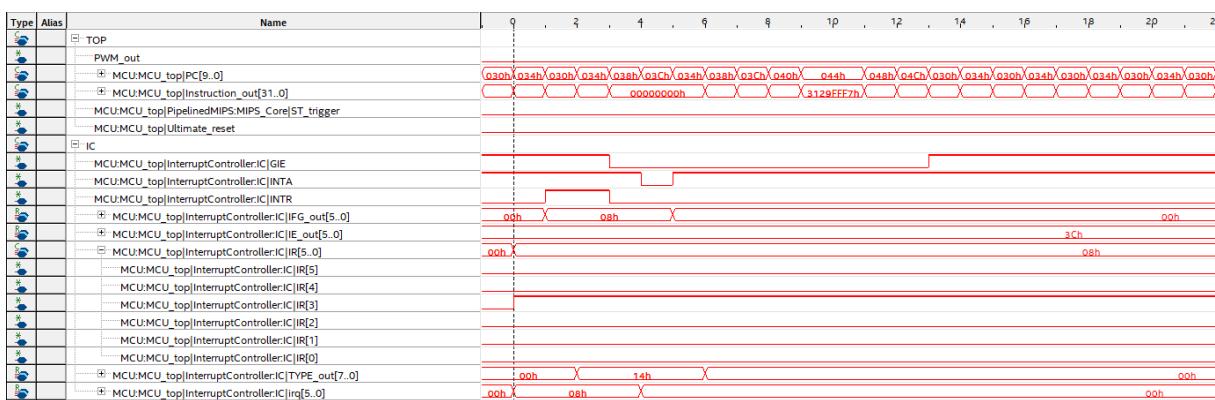


Figure 41: writing 'A' to hex0 and hex1 on KEY1 press – Signal Tap (IC)

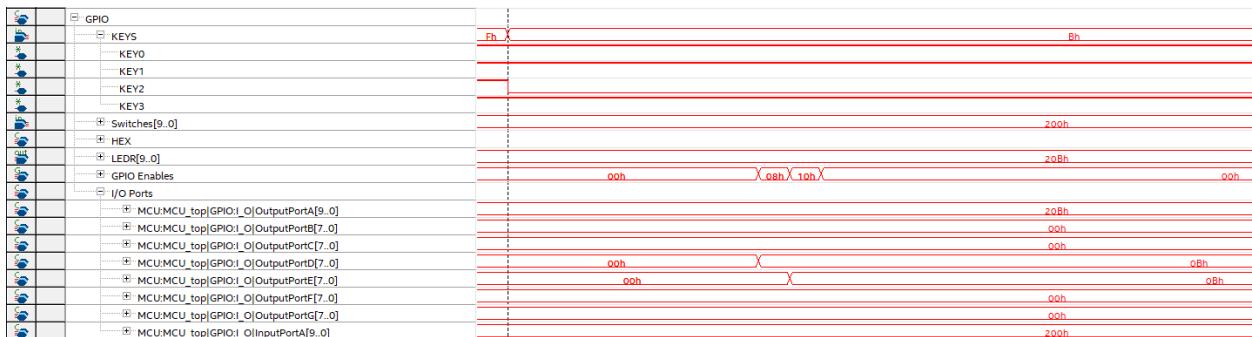


Figure 42: writing 'B' to hex0 and hex1 on KEY2 press – Signal Tap (GPIO)

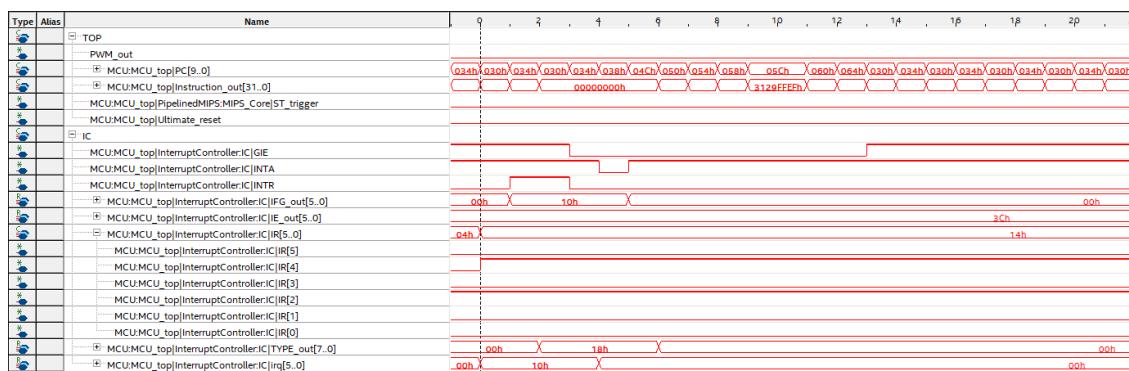


Figure 43: writing 'B' to hex0 and hex1 on KEY2 press – Signal Tap (IC)

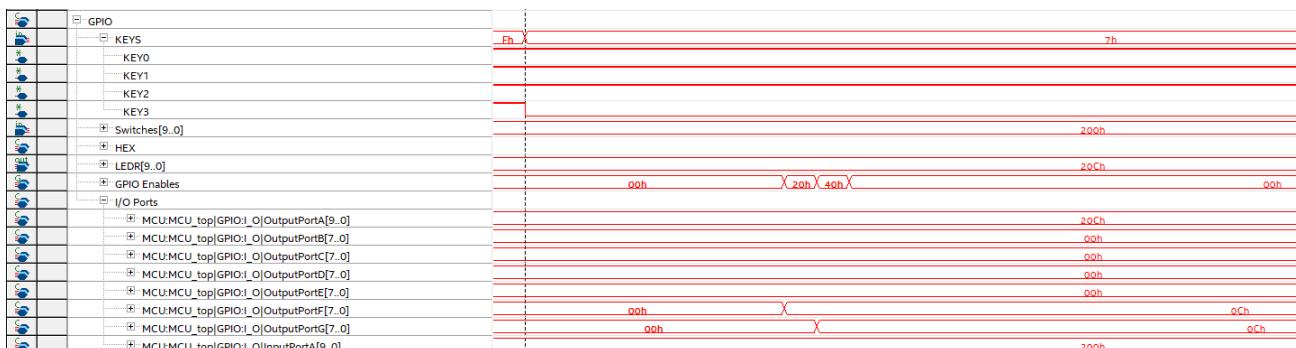


Figure 44: writing 'C' to hex0 and hex1 on KEY3 press – Signal Tap (GPIO)

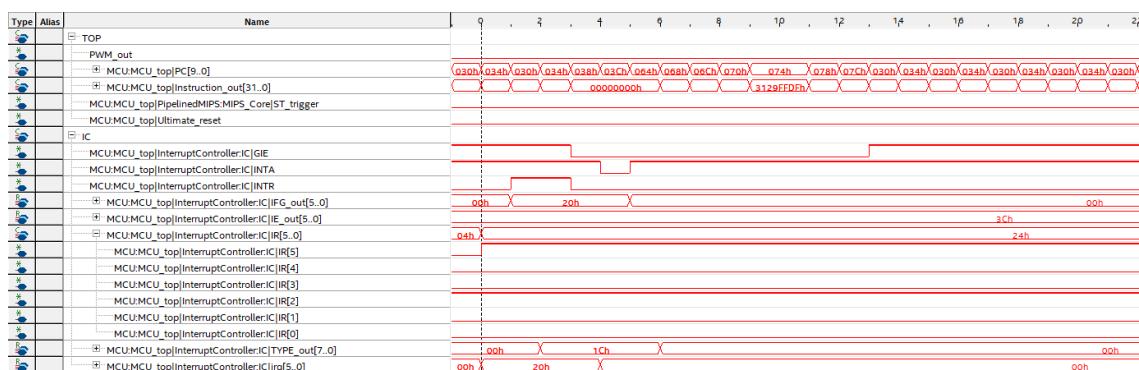


Figure 45: writing 'C' to hex0 and hex1 on KEY3 press – Signal Tap (IC)



**Test3:** בתכנית זו מתחכמת מניה מעלה על גבי הלדים עבור תדר שעון נתון:  
 לחיצה על כפתור 3 תגדיר את התדר שעון הగבורה ביותר ( $Clock = 50MHz$ )  
 לחיצה על כפתור 2 תחלק ב-2 את תדר השעון ( $Clock = 25MHz$ )  
 לחיצה על כפתור 1 תחלק ב-4 את תדר השעון ( $Clock = 12.5MHz$ )  
 לחיצה על כפתור 0 תבצע *Reset* לתכנית.

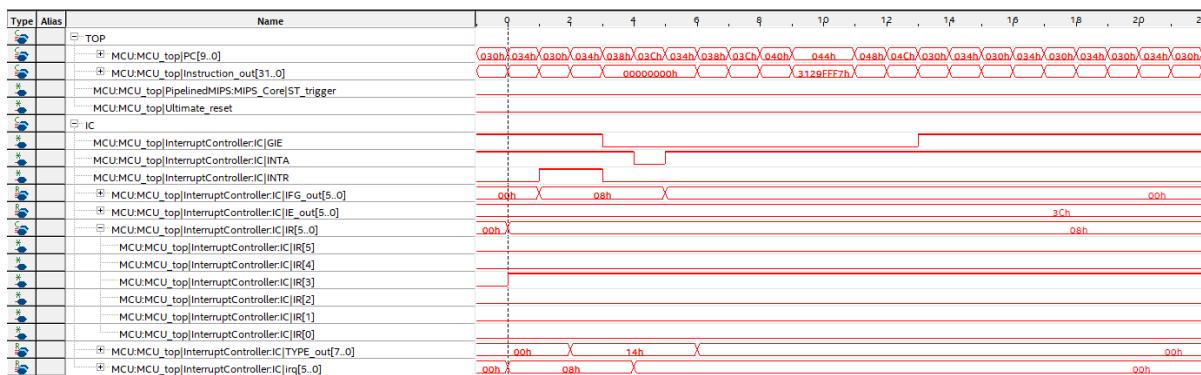


Figure 46: KEY1 IC – Signal Tap

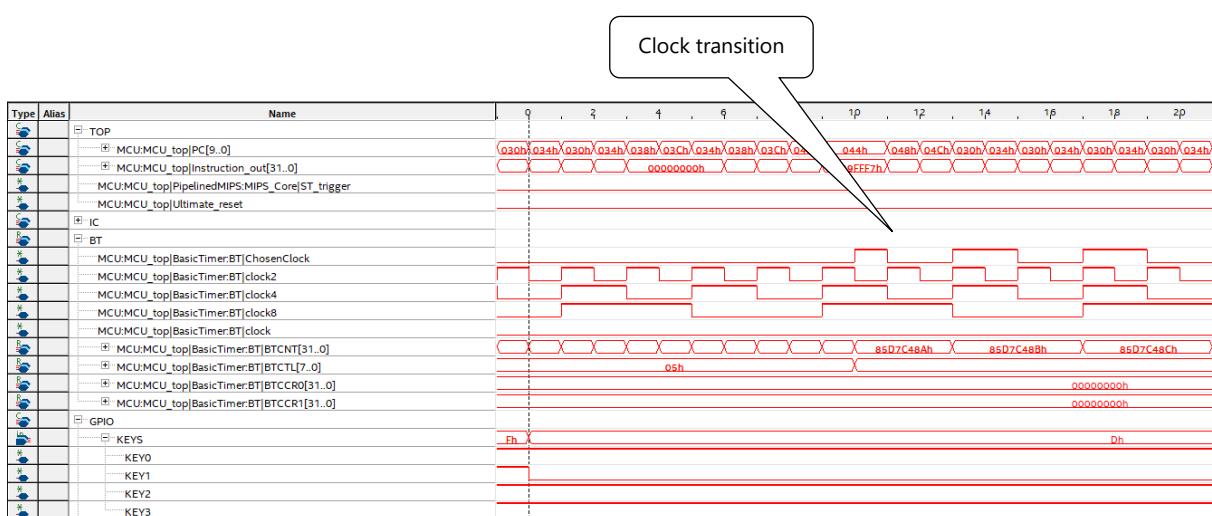


Figure 47: KEY1 ISR – Signal Tap

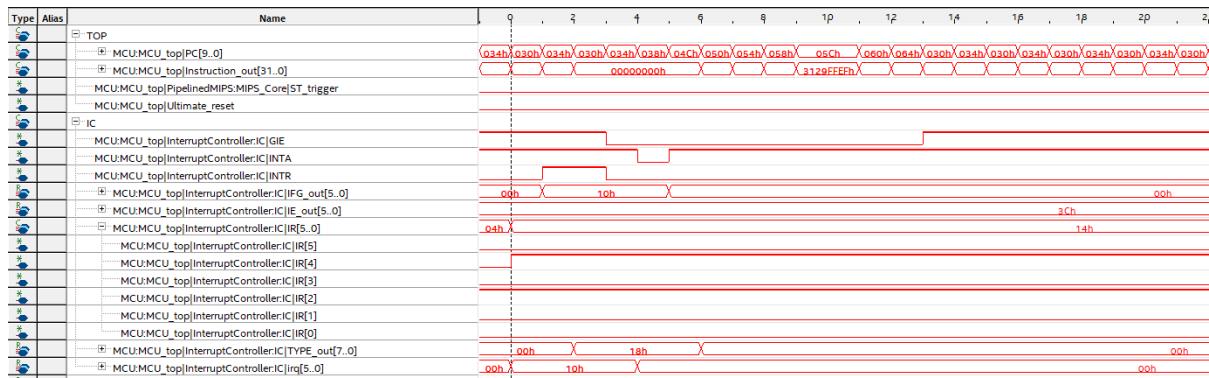


Figure 48: KEY2 IC – Signal Tap



Figure 49: KEY2 ISR – Signal Tap

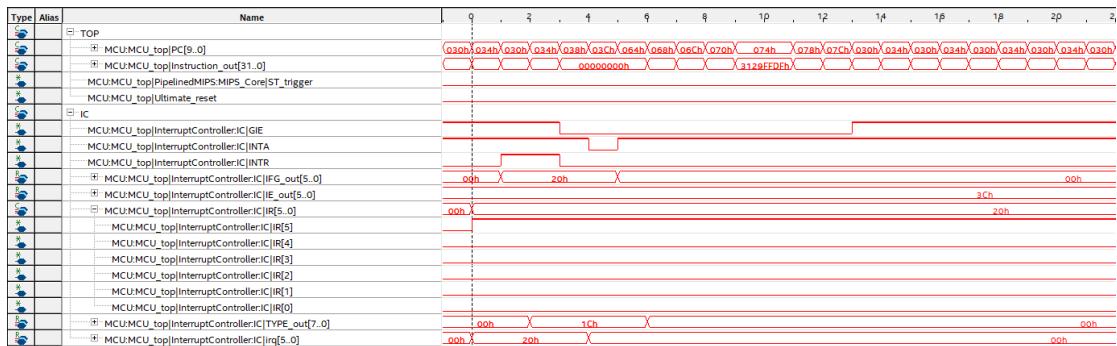


Figure 50: KEY3 IC – Signal Tap

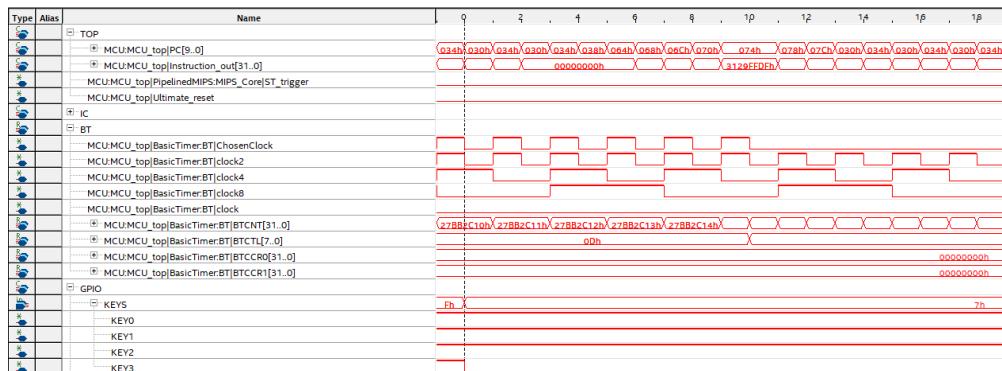


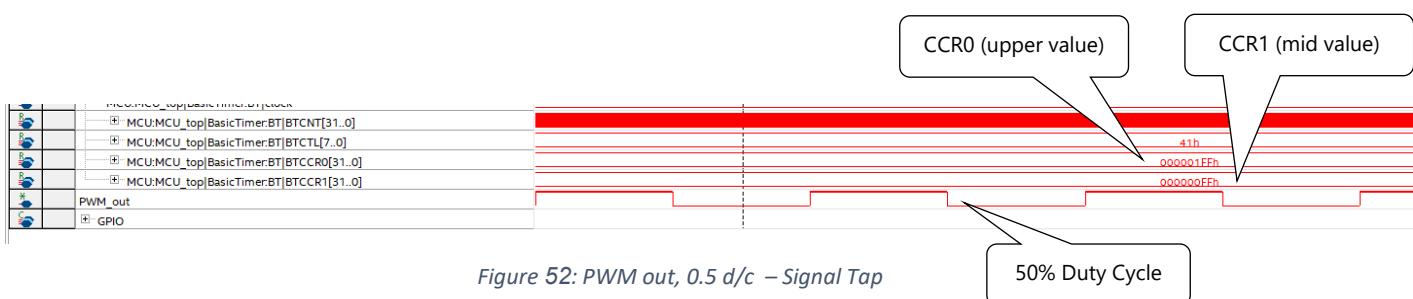
Figure 51: KEY3 ISR – Signal Tap



## סימולציה PWM:

תכנית נוספת שכתבנו היא תכנית המKENGAט טימר לצורך הוצאה גל ריבועי עם דיטוי סיקל מוגדר מראש ממודול *Basic Timer*.  
ערך ה-*D*.*N* קבוע ע"פ הערך שנקבע לריגיסטרי המעתפת *CCR0*, *CCR1* בנוסף להוצאה גל ריבועי, הגדרנו שבכל פסיקת שעון נטען את ערכי המתגים על פנוי כל הפלטים (*HEX*, *LEDs*).

בסימולציה הבאות על פניו *Signal Tap* ניתן לראות את ערכי המעתפות ואת האות הריבועי המתאים שיוצא מה-*MCU*:



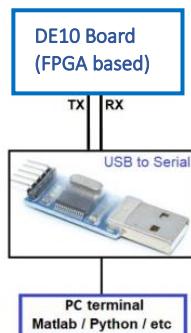


Figure 54: UART connection between FPGA and PC side

## UART .6

*UART* הינו מודול אשר אחראי על שליחת וקבלת מידע ותקשורת סריאלית בין צד מחשב לבין שבב ה-*FPGA*. התקשרות מתבצעת באמצעות מתאם *TTL USB* המחבר באופן ישיר לפיני ה-*GPIO* של ה-*FPGA*.

למודול קיימים מספר רגיסטרים:

- *UCTL* – זהו רגיסטר השליטה, המכיל את ביטי הבקרה השונים, כגון: בית אפשר, בית זוגיות, בית עבר קצב שליחה, בית זמינות וביטים המסמנים שגיהה.

- *RXBUF* – זהו באפר שניtan לקריאה ע"י המשתמש, המכיל את הביטים האחרונים שהתקבלו מרגיסטר ההזזה, שקיבל מידע מרגל *AX* באופן טורי. קריאה מהבאפר מאפשרת את ביטי השגיהה ואת דגל *IFG1* עבור קבלת המידע. בכל פעם שהميدע תקין ומזמן בריגיסטר זה, נקבל פסיקת *AX* ונ Kapoor לרווחת הפסיקה המתאימה.

- *TXBUF* – זהו באפר שניtan לכתיבה ע"י המשתמש, ומחזק את המידע שמחכה לעבר לרגיסטר ההזזה ולהשליך דרך דרגל *AX* באופן טורי. כתיבה לבאפר מאפשרת את דגל *IFG1* עבור שליחת המידע. לאחר טיענת מידע לרגיסטר זה, נקבל פסיקת *AX* ונ Kapoor לרווחת הפסיקה המתאימה.

את רגיסטרי ההזזה וההמרה בין מידע טורי לוקטור ביצענו בעזרת *FSM* בשני תת-המודול. שלבי ה-*FSM* הם: המתנה, קבלת בית התחלת, קבלת בית מידע, קבלת בית סיום, *cleanup*.

כאשר *PENA* דולק, כלומר אנו מצפים לקבל בית נוסף של זוגיות, נוסף עוד מצב למוכנות המצביעים עבור בדיקת הזוגיות.



## סימולציות (Signal Tap), וריפיקציה פונקציונלית (Wave Diagram):

על מנת לבדוק את תקינות המודול, כתבנו קובץ *tb\_UART\_rx* שבודק באופן פרטני את ייחidot ה-*RX, TX*.

הבדיקה הראשונה שלחצת את המידע "A5" לAKER באופן טורי, בית ביט (בעזרת רגיסטר זהה), ללא/עם הוספת בית זוגיות:

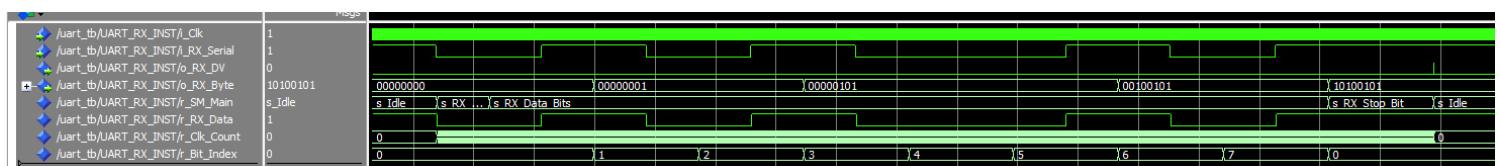


Figure 55: Receiving byte 'A5' through RX – Wave diagram (without parity bit)

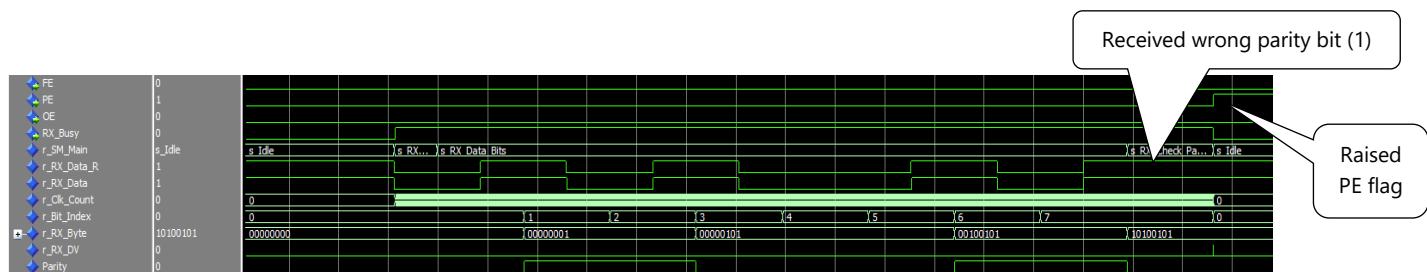


Figure 56: Receiving byte 'A5' through RX – Wave diagram (with parity bit)

הבדיקה השנייה שביבינו גורמת לAKER לשלח את המידע "AB" דרך רgel ה-AX, באופן טורי ע"י רגיסטר זהה ומוכנת המצביעים, ללא/עם הוספת בית זוגיות:

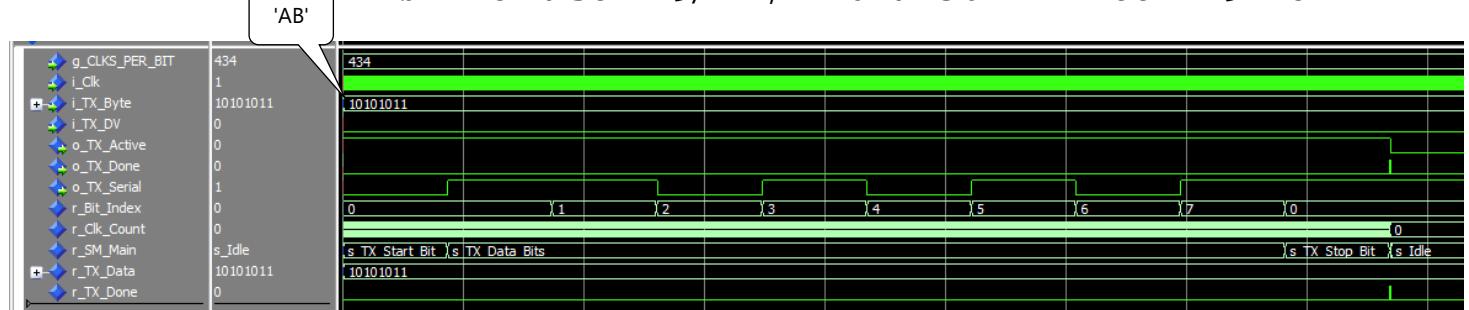


Figure 57: Transmitting byte 'AB' through TX – Wave diagram (without parity bit)

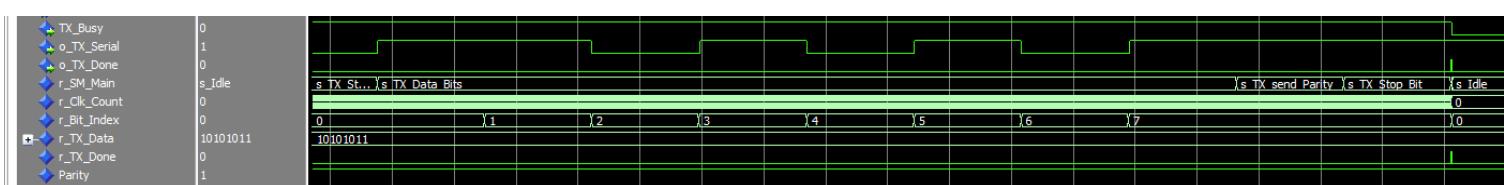


Figure 58: Transmitting byte 'AB' through TX – Wave diagram (with parity bit)

לאחר מכן ביצעו סימולציות נוספות בודקות את התקינות הכללת של מודול ה-*UART*. לשם כך צרבנו את קובץ הבדיקה *Test5*, המשלב שליחת/קבלת מידע דרך המודול ודרך בקר הפסיקות:

Printing  
result to  
LEDs

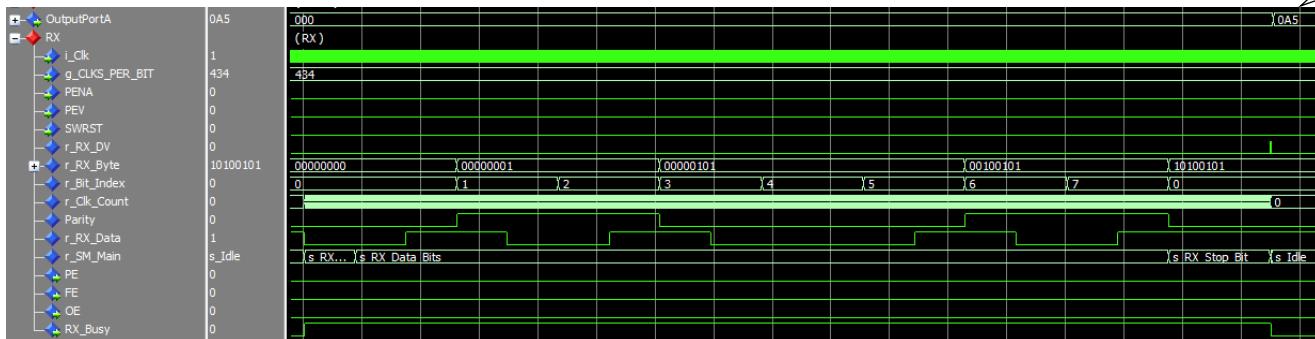


Figure 59: Receiving byte 'A5' through RX – Wave diagram (without parity bit)

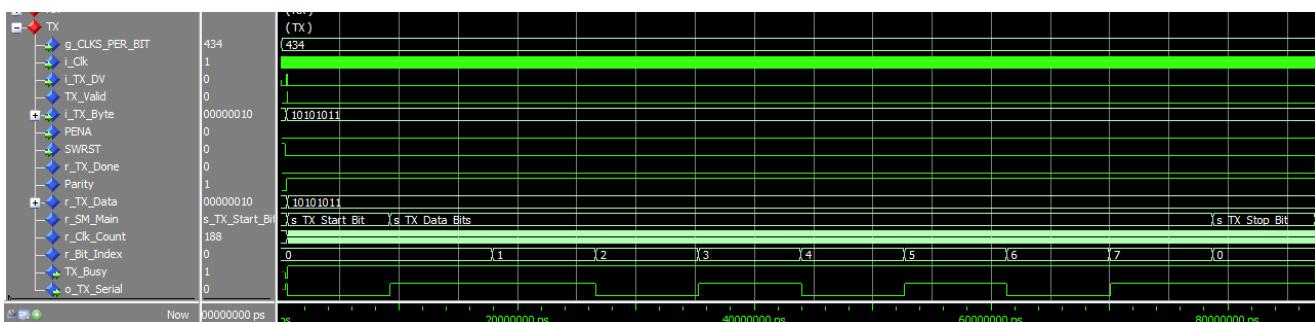


Figure 60: Transmitting byte 'AB' through TX – Wave diagram (without parity bit)

בקובץ ה-*tb* הפעילנו את כפתור 1, אח"כ את כפתורים 2,3 ייחדי, אח"כ קיבלנו את המידע "A5" דרך פסיקת *RX*, ולבסוף כפתור 0 המאטחל את התכנית.

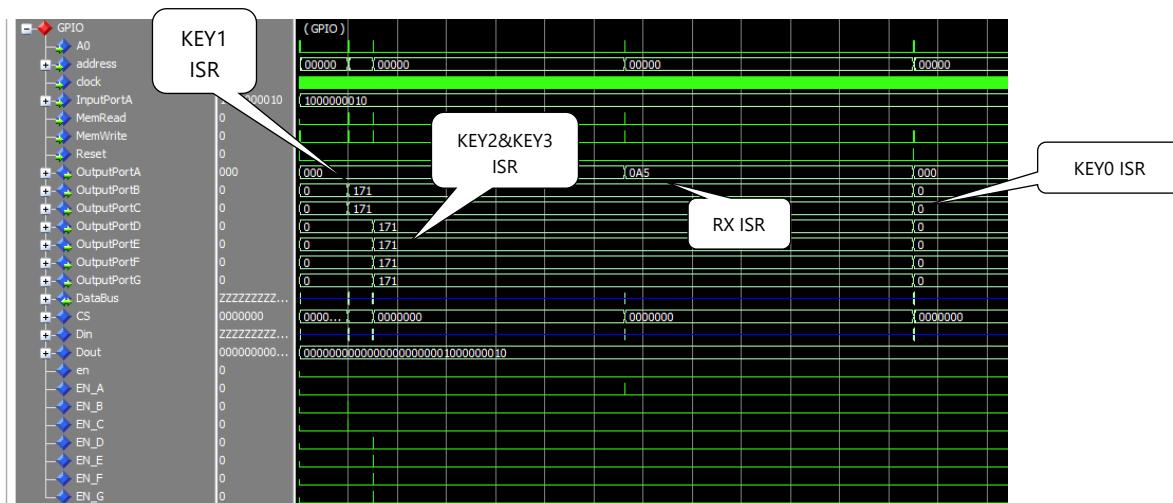


Figure 61: Test5 GPIO – Wave diagram



### Signal Tap

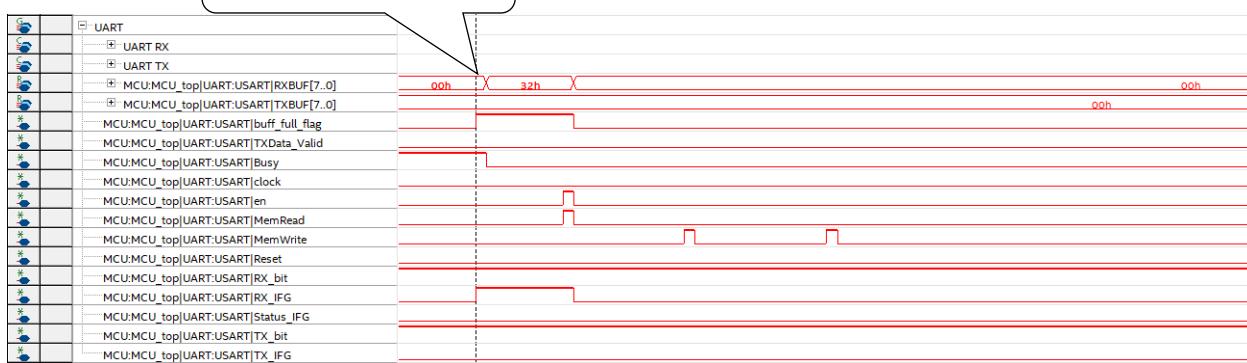


Figure 62: Receiving byte '32' through RX – Signal Tap

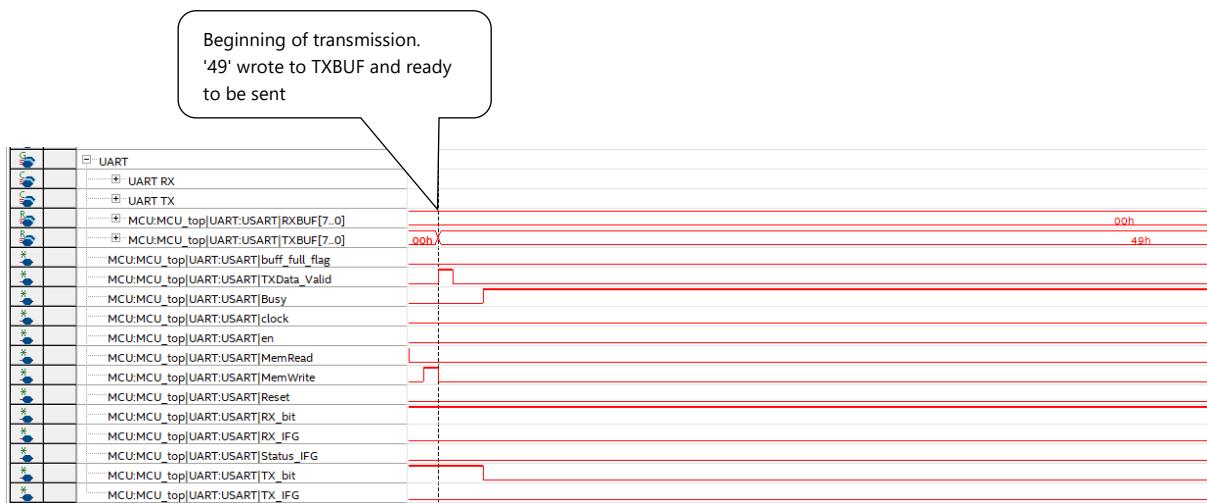


Figure 63: Transmitting byte '49' through TX – Signal Tap



לצורך הבדיקה המסכמת, כתבו קוד *asm* המממש את התפরיט הבא:

- Menu
1. Count up from 0x00 onto LEDG with delay ~0.5sec
  2. Count down from 0xFF onto LEDR with delay ~0.5sec
  3. Clear all LEDs
  4. On each KEY1 pressed, send the message "I love my Negev"
  5. Show Menu

כאשר המעבר בין הפקנציות בתפরיט מתבצע בעזרת *GUI* לצד המחשב (*Python*):

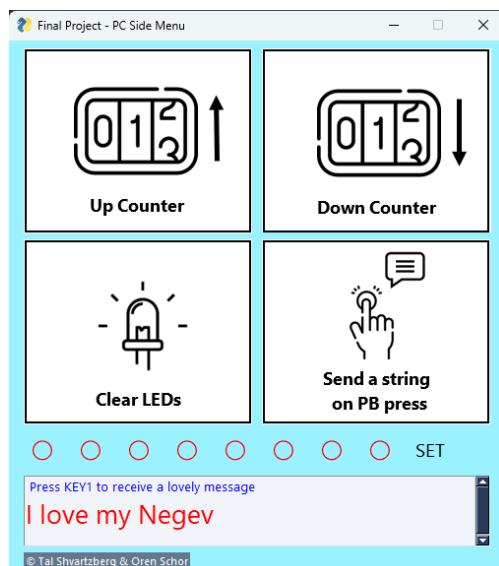


Figure 64: PC side menu, developed using PySimpleGUI

**Auxiliary Material** סרטון מלא המציג את תקינות הממשק מצורף בתיקייה שבתיקית הפרויקט.