CPU Architecture

Final Project Report File

עומר לוקסמבורג 205500390
עילי נוריאל 312538580

מטרת המטלה

מטרת מטלה זו היא ביצוע תכנון, סינתזה וניתוח של CPU פשוט מסוג MIPS, עם שילוב של Mapped I/O, וכמו כן הבנת מבנה הזיכרון של רכיב ה- Cyclone II FPGA. כמו כן הוספת עבודה עם טיימר ובקר פסיקות.

הגדרת תכנון המערכת

ה-CPU אותו תכננו עובד בתצורת Single Cycle, ומבצע פקודות מה-ISA של ה-MIPS.

כמו כן ל-CPU יש register file סטנדרטי של MIPS, וה- top level entity מתוכנן בתצורת structural.

פקודות נוספות (מעבר למטלה 3) שמימשנו הן : JR, JAL והן תוכננו בהתאם לפורמט הפקודות המתואר ב-Table 1 :

Type	-31-			format (bits)		-0-
R	opcode (6)	rs (5)	rt (5)	rd (5)	shamt (5)	funct (6)
I	opcode (6)	rs (5)	rt (5)	immediate (16)		
J	opcode (6)	address (26)				

Table 1 : MIPS Instruction format

התכנון אותו ממשנו תואם לדיאגרמת הבלוקים המתוארת ב-Figure 1 :

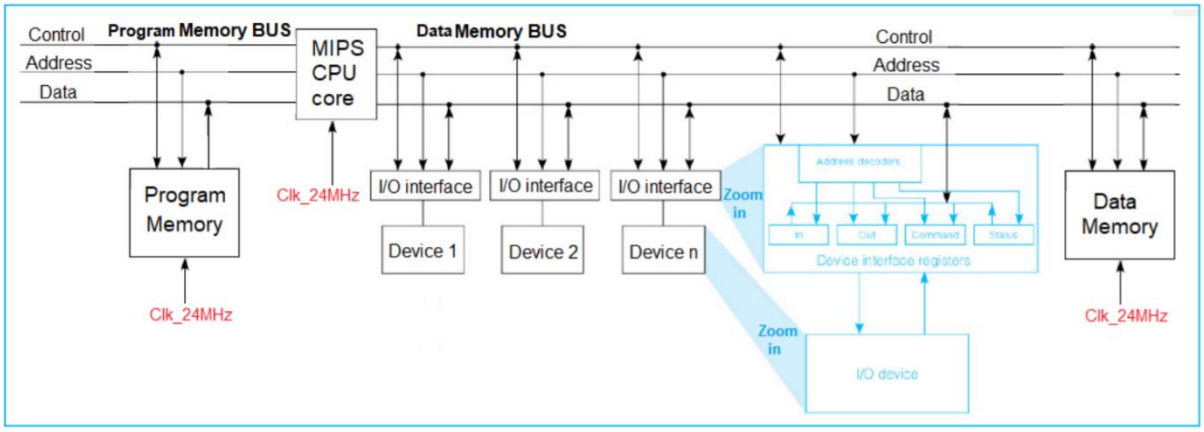


Figure 1 : System architecture

כמו כן התכנון שלנו מותאם ל- Mapped I/O המקושר למרחב הכתובות המתואר ב- Figure 2 :

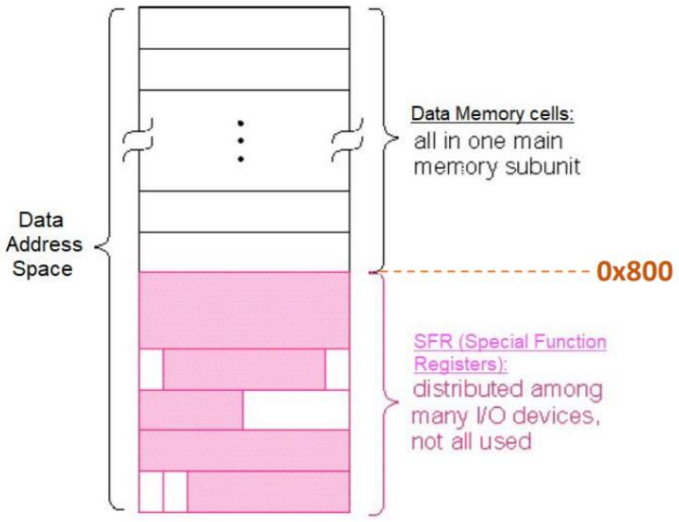


Figure 2: Address Space of a Computer Using Memory Mapped I/O

פריפריות נדרשות

בנוסף לפריפריות שנדרשו מאיתנו במטלה 3 (נורות הלד, המתגים ומסכי הספרות) נדרשנו להוסיף :

1. כפתורים 1-3.
2. Basic Timer - טיימר בסיסי בעל שני רגיסטרים (רגיסטר Control ורגיסטר Counter) שתוכנן בהתאם ל-Figure 3 :

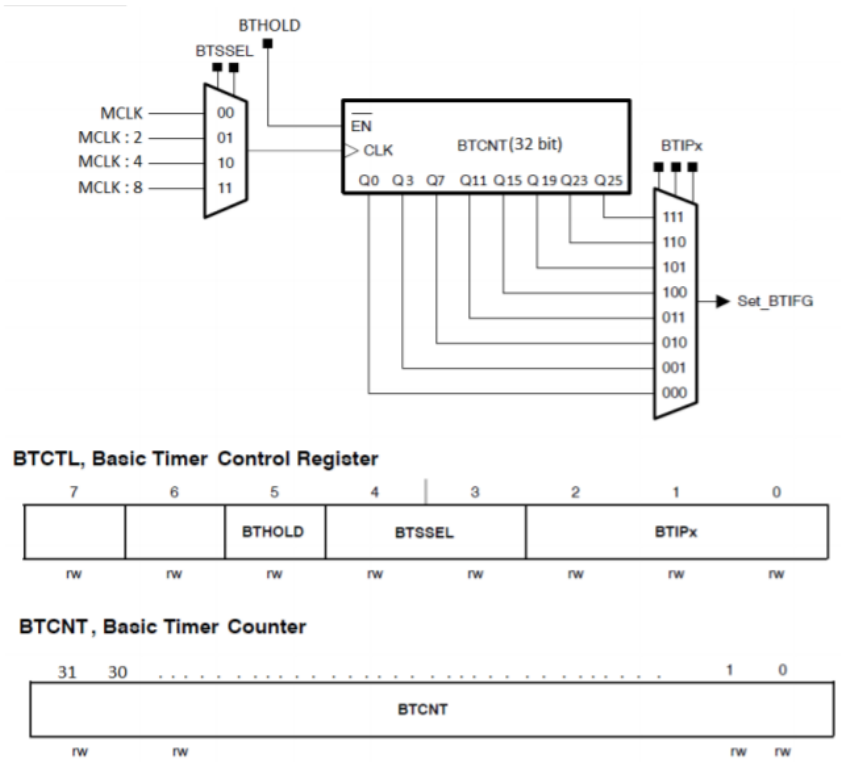


Figure 3: Basic Timer Structure

3. בקר פסיקות – המקבל בקשות לפסיקה מ-4 מקורות (כפתורים והטיימר) ובעל 3 רגיסטרים (רגיסטר Enable, רגיסטר דגלים, ורגיסטר Type לסוג הפסיקה) שתוכנן בהתאם ל-Figure 4:

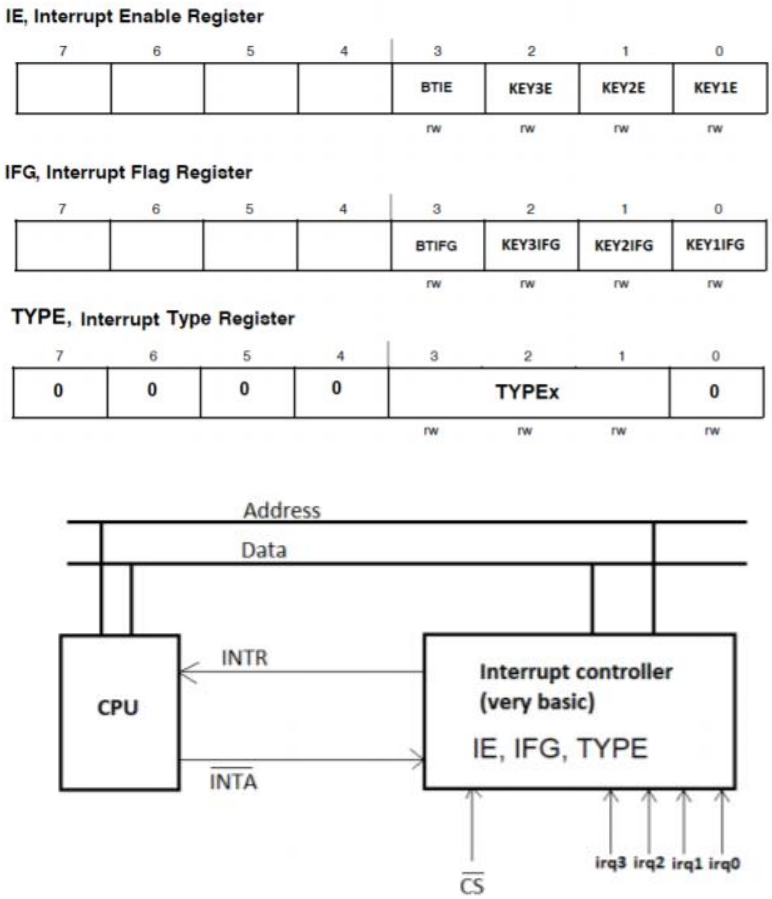


Figure 4: Interrupt Controller Structure

כמו כן תיעדוף בין הפסיקות השונות יתבצע לפי Table 2:

TYPE Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
00h	KEY1	KEY1IFG	Lowest
04h	KEY2	KEY2IFG	
08h	KEY3	KEY3IFG	
0Ch	Basic Timer	BTIFG	Highest

Table 2: Interrupt Controller Priority Table

הערות:

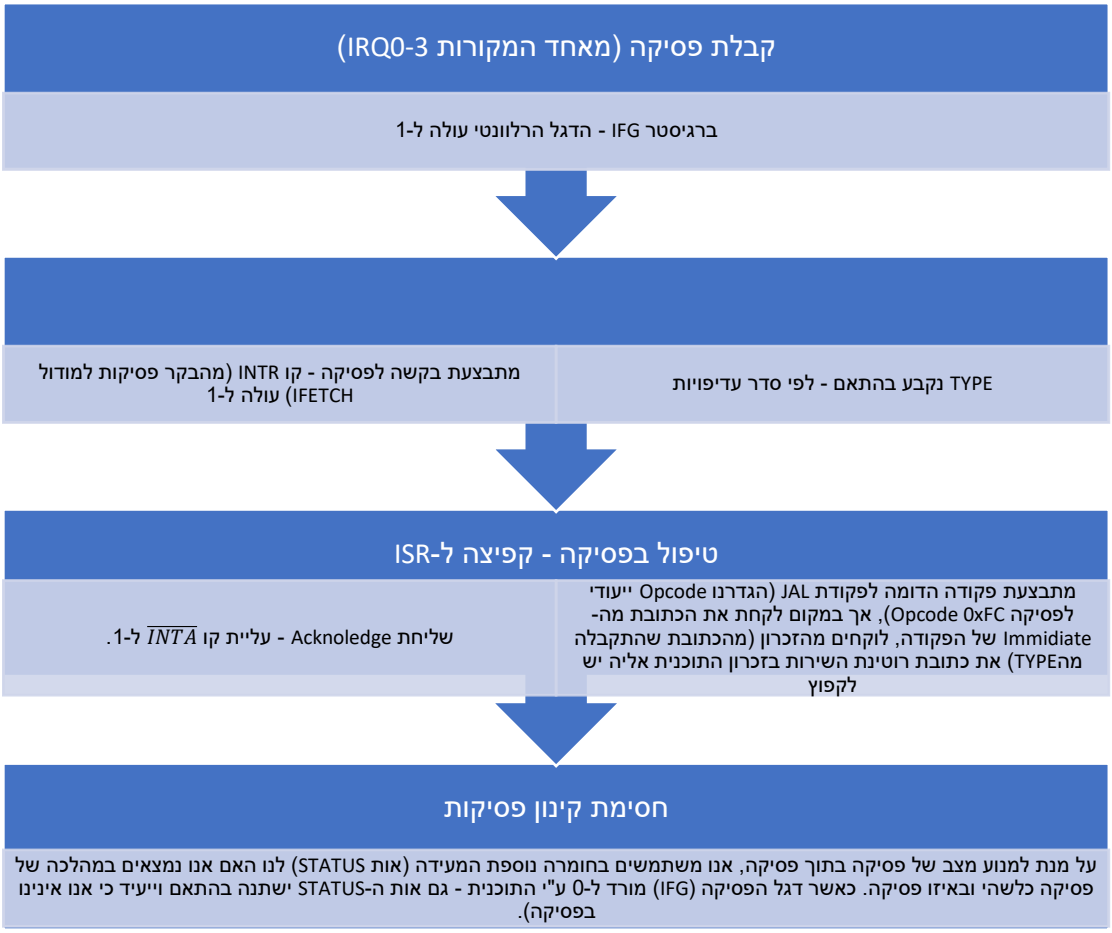
- נדרש מאיתנו לתכנן בקר פסיקות ללא קינון (Nesting), על כן הוספנו גם חומרה המוודאת כי לא מתרחש קינון ומגינה על המערכת במקרה של בקשות פסיקה המתרחשות בו זמנית.

אופן מימוש החומרה הנוספת

1. מימוש הפקודות JR, JAL –
   i. **JR** – בדומה למימוש פקודת JUMP, אנו משתמשים בקווי בקרה ייעודיים (isJR) היוצאים ממודול ה-control ל-IEXECUTE. כאשר מזוהה כי מתבצעת פקודת JR, ניקח את ADD_Result (תוצאה של ALU המחשב Branchים, במקרה שלנו זה ערך כתובת ב-words אליה נקפוץ) הנלקח מרגיסטר הייעד שמצוין ב- Instruction, ונקפוץ אל הכתובת השמורה בו ע"י השמה ל-PC.
   ii. **JAL** – על מנת לבצע פקודה זו, אנו שומרים את כתובת ה-PC אליה אנו נרצה לחזור (PC+4) ברגיסטר מספר 31 ($ra). במקביל אנו מבצעים את פקודת JUMP, כלומר קופצים לערך הכתובת המיידית הנתונה ב-Instruction ב-words.

2. מימוש ה-BasicTimer –
   על מנת לממש את החלוקות הנדרשות מהמודול, הגדרנו מונה באורך 3 ביטים, כך שכל שעון מחולק יהיה תלוי בערך הביט הרלוונטי לו במונה. למשל MCLK2 מקבל את ערכו לפי הביט במיקום ה-0 במונה (כאשר הוא עולה ל-1 גם השעון עולה ל-1 וכו').
   הביט BTIFG – יוצא מהמודול ומתחבר ל-IRQ3 בבקר הפסיקות, עולה ל-1 כאשר נוצר overflow בקבוצת הביטים שנבחרה (לפי BTIPx). לדוגמא אם נבחר ביט Q3, BTIFG יעלה ל-1 כאשר נקבל overflow עבור ביטים Q0-Q3, כלומר כאשר ספרנו $2^4 = 16$ פעמים.

3. מימוש ה-InterruptController – השרשרת המלאה מתוארת בתרשים הזרימה הכחול מטה.
   i. ראשית, הגדרנו למודול קו ENABLE "המאזין" ל-Address Bus, Data Bus ומאפשר את המודול כאשר פונים לכתובת שלו.
   ii. הגדרנו ביט GIE (הביט ברגיסטר $k0 במקום ה-0). כאשר ביט זה שווה ל-0, המודול למעשה מושבת ולא יטופלו פסיקות.
   iii. ניתן לרשום ולקרוא מהרגיסטרים IE, IFG לפי הכתובות שהוגדרו.



בדיקת התכנון והמערכת

ניתוח עבודת המערכת מתבצע ע״י שני ה- test benchים הבאים, שניתנו לנו מראש:

1. תמיכה והרצה תקינה של הקוד בקובץ ה-*test1.asm* שניתן לנו – בדיקת מערך הקריאה לפונקציה, פקודות jal ו-jr:
חיבור 2 מספרים השמורים בזכרון, Y,X, ושולחים את הסכום לפונקציה המחסרת ממנו 4 ומחזירה את הערך. במידה והערך חיובי התוכנית מחזירה אותו, במידה והוא שלילי – היא מחזירה את הסכום המקורי ללא חיסור 4.
2. תמיכה והרצה תקינה של הקוד בקובץ ה-*test2.asm* שניתן לנו – קריאת הערך של המתגים והצגתו באופנים שונים בהתאם למקור הפסיקה:

- Key 1 – מציג את הערך של ה-SW + הערך שהמונה ספר על גבי הLEDים (הירוקים והאדומים).
- Key 2 – מציג את ערך הערך של ה-SW + הערך שהמונה ספר על גבי Hex 0,1.
- Key 3 – מציג את ערך הערך של ה-SW + הערך שהמונה ספר על גבי Hex 0,1.
- BTIFG – הוספת 1 לערך המונה והצגת הערך על גבי הLEDים הירוקים.

הערות:

- הוספנו מנגנון המאפשר מעבר בין תוכנת ה-ModelSim לתוכנת ה-Quartus ללא צורך בשינוי הקוד, כך שה-tester שב-ModelSim מזריק ערך המצביע על כך שאנו בModelSim. זאת ע״י מימוש גנרי למודולים הדורשים זאת (MIPS, IFETCH).

ניתוח עבודת המערכת

- **תדר השעון המקסימלי** שקיבלנו הוא $82\ Mhz$:

Slow Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	82.39 MHz	82.39 MHz	clk_24Mhz	

Figure 5: Fmax analysis

- **ניתוח המסלול הקריטי:**

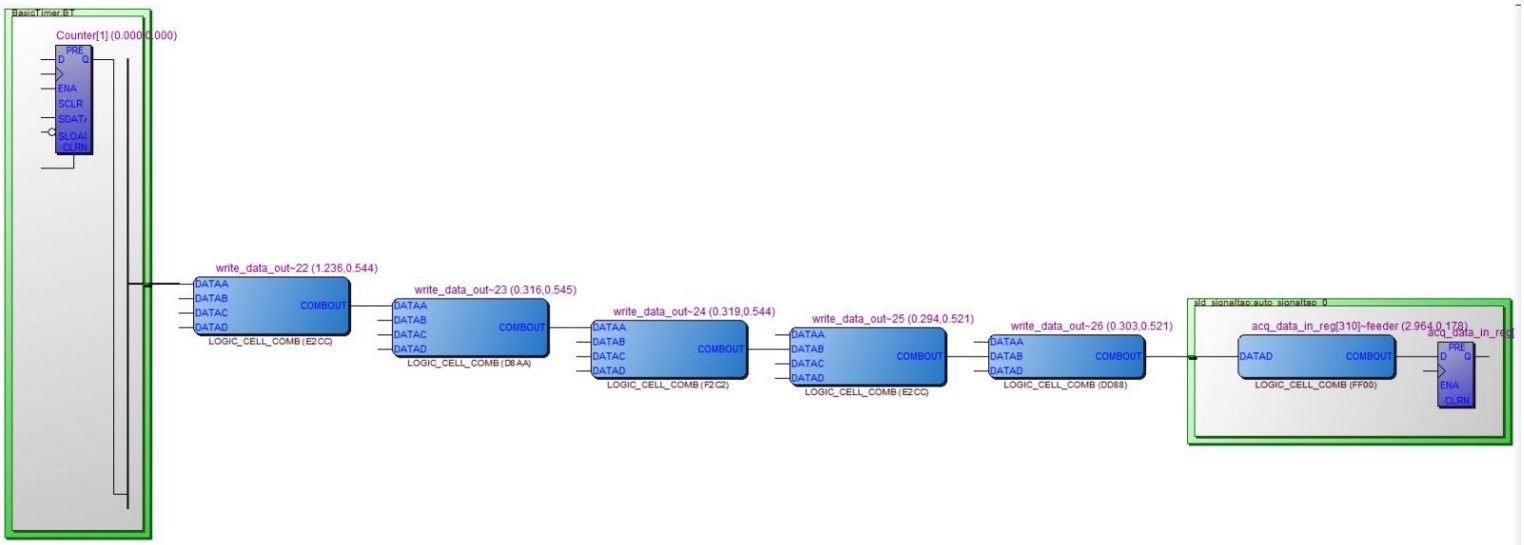


Figure 6: Critical Path

המסלול הקריטי כולל שימוש ב-Counter – שמצריך שמירה של ערכו על מנת לחשב את המצב הבא, וגם חיבור של 2 מספרים באורך של 32 ביטים.

- **ניתוח המסלול המינימלי:**

אנו משערים כי המסלול הכי קצר המערכת הוא של אחת מפעולות ה-bitwise, OR או AND (XOR ככל הנראה מעט ארוכה יותר), זאת מכיוון שהן מבוצעות ע"י מימוש פשוט של שער לוגי וללא פעולות אריתמטיות מסובכות, וכמו כן הן נכתבות לרגיסטר ולא לזיכרון.
באופן כללי – כלל הפעולות מבצעות את 4 השלבים של המערכת (כאשר יש פעולות שמבצעות Write back לרגיסטרים ויש פעולות הכותבות לזיכרון) כלומר, אין פעולה קצרה משמעותית מפני שכולן עוברות בשלבי המערכת כפי שלמדנו על מעבד single cycle.

- **ניתוח השימוש הלוגי:**

של ה-MIPS כולו:

Analysis & Synthesis Summary	
Analysis & Synthesis Status	Successful - Tue Sep 08 17:19:20 2020
Quartus II 64-Bit Version	12.1 Build 177 11/07/2012 SJ Web Editio
Revision Name	Final
Top-level Entity Name	MIPS
Family	Cyclone II
Total logic elements	8,114
Total combinational functions	3,860
Dedicated logic registers	5,462
Total registers	5462
Total pins	235
Total virtual pins	0
Total memory bits	215,808
Embedded Multiplier 9-bit elements	0
Total PLLs	0

Figure 7: MIPS Logic Usage

Fetch:

Analysis & Synthesis Summary	
Analysis & Synthesis Status	Successful - Tue Sep 08 17:28:05 2020
Quartus II 64-Bit Version	12.1 Build 177 11/07/2012 SJ Web Edition
Revision Name	Final
Top-level Entity Name	Ifetch
Family	Cyclone II
Total logic elements	4,956
Total combinational functions	1,532
Dedicated logic registers	4,311
Total registers	4311
Total pins	81
Total virtual pins	0
Total memory bits	101,120
Embedded Multiplier 9-bit elements	0
Total PLLs	0

Figure 8: Ifetch Logic Usage

Decode:

Analysis & Synthesis Summary	
Analysis & Synthesis Status	Successful - Tue Sep 08 17:27:10 2020
Quartus II 64-Bit Version	12.1 Build 177 11/07/2012 SJ Web Edition
Revision Name	Final
Top-level Entity Name	Idecode
Family	Cyclone II
Total logic elements	7,301
Total combinational functions	2,955
Dedicated logic registers	5,294
Total registers	5294
Total pins	209
Total virtual pins	0
Total memory bits	84,736
Embedded Multiplier 9-bit elements	0
Total PLLs	0

Figure 9: Idecode Logic Usage

: Execute

Analysis & Synthesis Summary	
Analysis & Synthesis Status	Successful - Tue Sep 08 17:24:32 2020
Quartus II 64-Bit Version	12.1 Build 177 11/07/2012 SJ Web Edition
Revision Name	Final
Top-level Entity Name	Execute
Family	Cyclone II
Total logic elements	5,406
Total combinational functions	1,983
Dedicated logic registers	4,301
Total registers	4301
Total pins	168
Total virtual pins	0
Total memory bits	84,736
Embedded Multiplier 9-bit elements	0
Total PLLs	0

Figure 10: Execute Logic Usage

: Memory

Analysis & Synthesis Summary	
Analysis & Synthesis Status	Successful - Tue Sep 08 17:25:13 2020
Quartus II 64-Bit Version	12.1 Build 177 11/07/2012 SJ Web Edition
Revision Name	Final
Top-level Entity Name	dmemory
Family	Cyclone II
Total logic elements	4,887
Total combinational functions	1,464
Dedicated logic registers	4,301
Total registers	4301
Total pins	79
Total virtual pins	0
Total memory bits	150,272
Embedded Multiplier 9-bit elements	0
Total PLLs	0

Figure 11: dmemory Logic Usage

: Control

Analysis & Synthesis Summary	
Analysis & Synthesis Status	Successful - Tue Sep 08 17:26:11 2020
Quartus II 64-Bit Version	12.1 Build 177 11/07/2012 SJ Web Edition
Revision Name	Final
Top-level Entity Name	control
Family	Cyclone II
Total logic elements	4,911
Total combinational functions	1,488
Dedicated logic registers	4,301
Total registers	4301
Total pins	26
Total virtual pins	0
Total memory bits	84,736
Embedded Multiplier 9-bit elements	0
Total PLLs	0

Figure 12: control Logic Usage

: BasicTimer

Analysis & Synthesis Summary	
Analysis & Synthesis Status	Successful - Tue Sep 08 17:29:33 2020
Quartus II 64-Bit Version	12.1 Build 177 11/07/2012 SJ Web Edition
Revision Name	Final
Top-level Entity Name	BasicTimer
Family	Cyclone II
Total logic elements	4,938
Total combinational functions	1,510
Dedicated logic registers	4,342
Total registers	4342
Total pins	83
Total virtual pins	0
Total memory bits	84,736
Embedded Multiplier 9-bit elements	0
Total PLLs	0

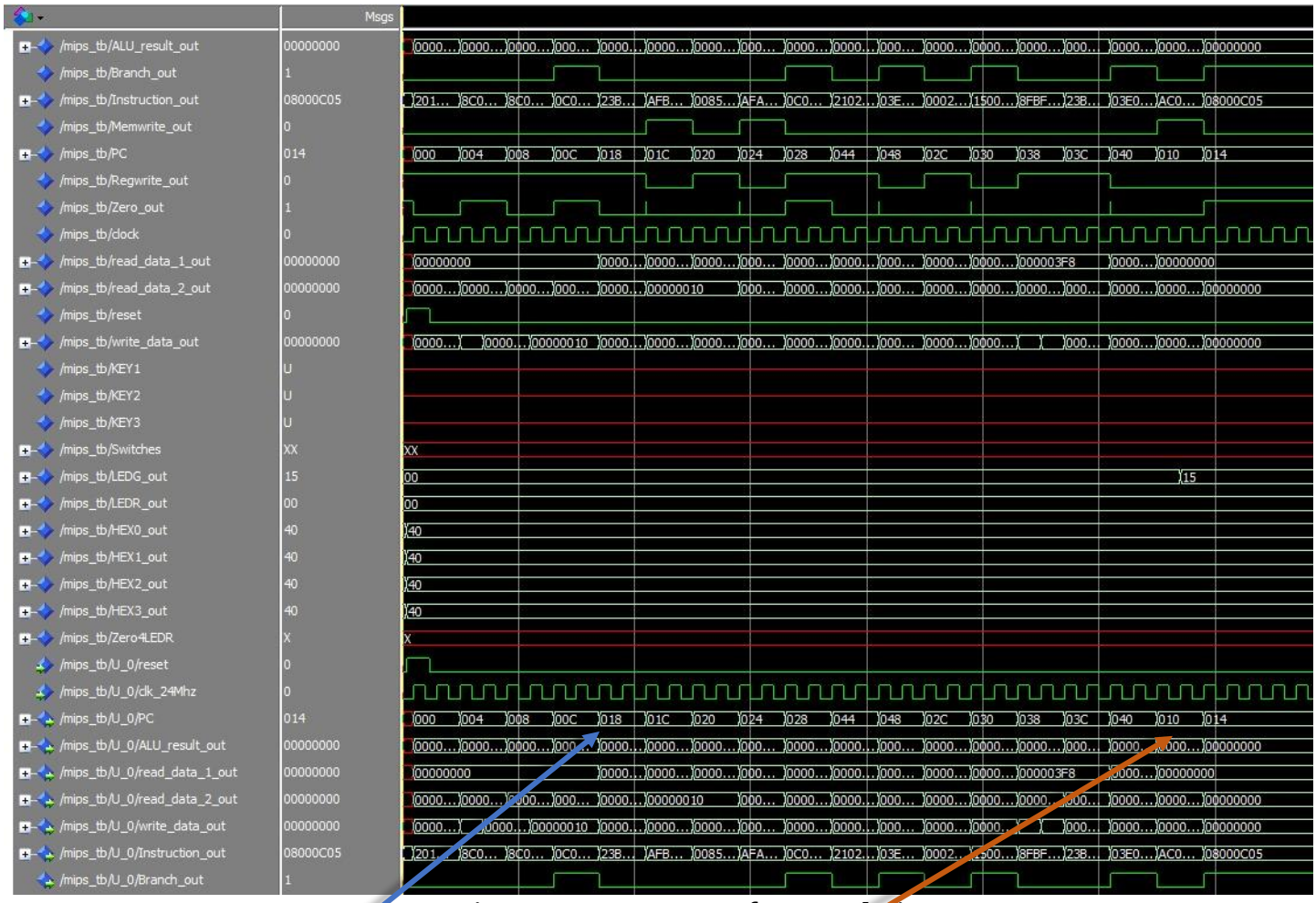
Figure 13: BasicTimer Logic Usage

: InterruptController

Analysis & Synthesis Summary	
Analysis & Synthesis Status	Successful - Tue Sep 08 17:30:30 2020
Quartus II 64-Bit Version	12.1 Build 177 11/07/2012 SJ Web Edition
Revision Name	Final
Top-level Entity Name	InterruptController
Family	Cyclone II
Total logic elements	4,931
Total combinational functions	1,496
Dedicated logic registers	4,325
Total registers	4325
Total pins	53
Total virtual pins	0
Total memory bits	84,736
Embedded Multiplier 9-bit elements	0
Total PLLs	0

Figure 14: InterruptController Logic Usage

- **Wave form**: זוהי תצוגה של הקוד של test1.asm:



Figures 15: test1 Wave form analysis

בשורה 0x300C מתבצעת פקודת JAL לכתובת 0x3018, ואכן ב-Waveform ניתן לראות כי ערך ה-PC קופץ לערך הנדרש

בשורה 0x3040 מתבצעת פקודת JR לכתובת 0x3010, ואכן ב-Waveform ניתן לראות כי ערך ה-PC קופץ לערך הנדרש

Bkpt	Address	Code	Basic		
	0x00003000	0x201d0400	addi $29,$0,0x00000400	20:	addi $sp,$zero,0x400 # $sp=0x400
	0x00003004	0x8c040000	lw $4,0x00000000($0)	22:	lw $a0,x
	0x00003008	0x8c050004	lw $5,0x00000004($0)	23:	lw $a1,y
	0x0000300c	0x0c000c06	jal 0x00003018	24:	jal func
	0x00003010	0xac020800	sw $2,0x00000800($0)	25:	sw $v0,0x800 # write to PORT_LEDG[7-0]
	0x00003014	0x08000c05	j 0x00003014	26: L:	j L
	0x00003018	0x23bdfff8	addi $29,$29,0xffff...	28: func:	addi $sp,$sp,-8
	0x0000301c	0xafbf0000	sw $31,0x00000000($29)	29:	sw $ra,0($sp)
	0x00003020	0x00854020	add $8,$4,$5	30:	add $t0,$a0,$a1
	0x00003024	0xafa80004	sw $8,0x00000004($29)	31:	sw $t0,4($sp)
	0x00003028	0x0c000c11	jal 0x00003044	32:	jal calc
	0x0000302c	0x0002402a	slt $8,$0,$2	33:	slt $t0,$0,$v0
	0x00003030	0x15000001	bne $8,$0,0x00000001	34:	bne $t0,$0,exit
	0x00003034	0x8fa20004	lw $2,0x00000004($29)	35:	lw $v0,4($sp)
	0x00003038	0x8fbf0000	lw $31,0x00000000($29)	36: exit:	lw $ra,0($sp)
	0x0000303c	0x23bd0008	addi $29,$29,0x0000...	37:	addi $sp,$sp,8
	0x00003040	0x03e00008	jr $31	38:	jr $ra
	0x00003044	0x2102fffc	addi $2,$8,0xfffffffc	40: calc:	addi $v0,$t0,-4
	0x00003048	0x03e00008	jr $31	41:	jr $ra

Figures 16: test1 Assembly Code

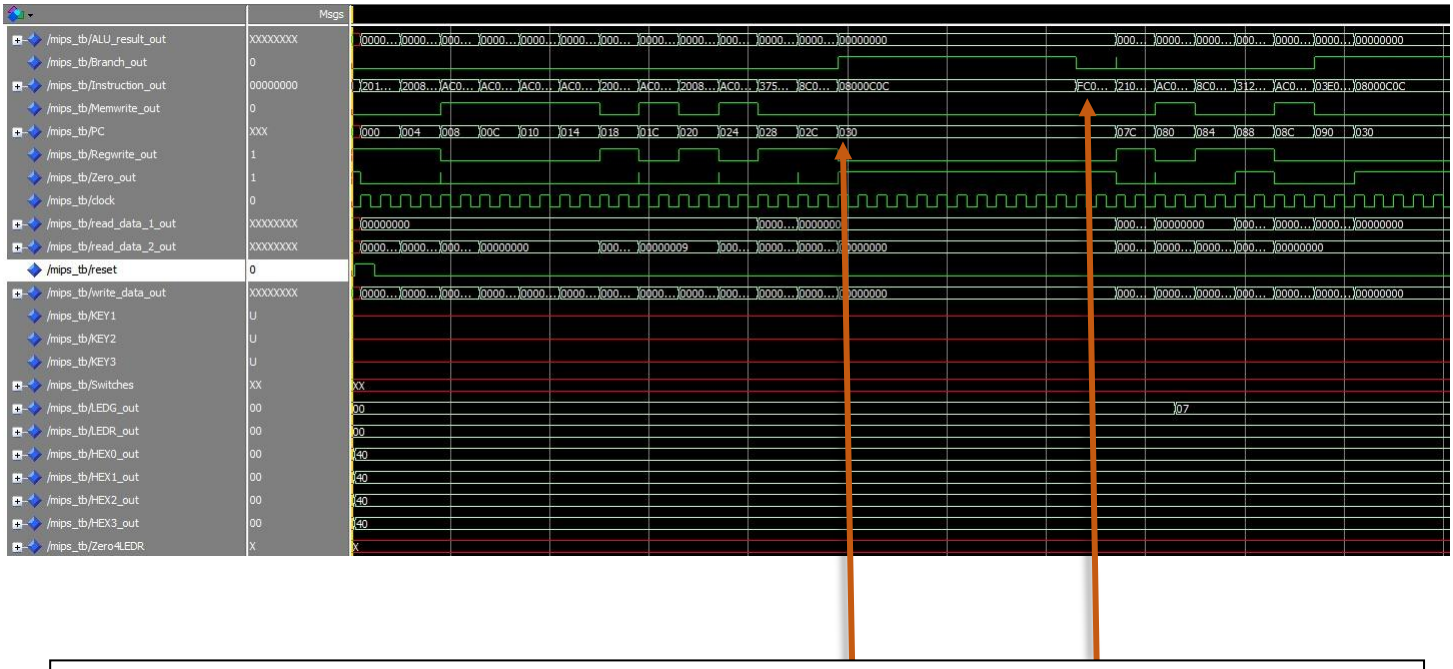
- **Wave form**: זוהי תצוגה של הקוד של test2.asm:

הערות:

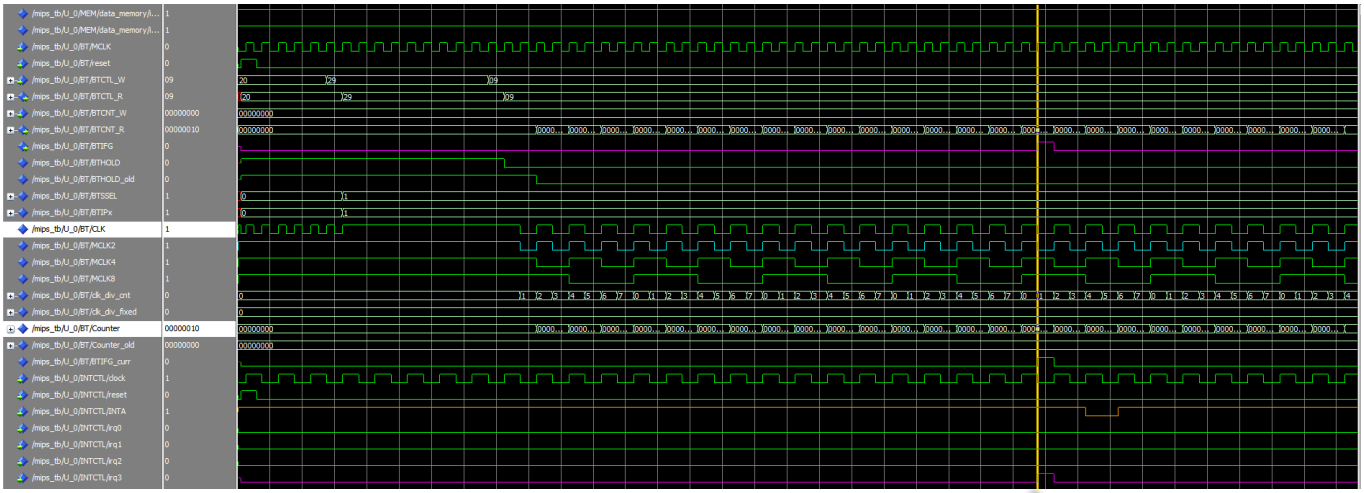
- ערכנו את קוד האסמבלי של test2 לשם הבדיקה – הגדרנו BTIP=1, BTSSEL=1 על מנת שהפסיקות מהטיימר יהיו מהירות יותר (חלוקה קטנה יותר של השעון).
- המקרה המתואר ב- Figure 18 בוחן רק את תפקוד פסיקת הטיימר מאחר וכל הפסיקות זהות ורצינו לבדוק את עבודת שתי הפריפריות החדשות יחדיו, תוך ניצול כלל חלקי המערכת.

Address	Code	Basic		
0x00003000	0x201d0400	addi $29,$0,0x00000400	23:	addi $sp,$zero,0x400 # $sp=0x400
0x00003004	0x20080029	addi $8,$0,0x00000029	24:	addi $t0,$zero,0x29
0x00003008	0xac080820	sw $8,0x00000820($0)	25:	sw $t0,0x820 # BTIP=1, BTSSEL=1, BTHOLD=1
0x0000300c	0xac000824	sw $0,0x00000824($0)	26:	sw $0,0x824 # BTCNT=0
0x00003010	0xac000828	sw $0,0x00000828($0)	27:	sw $0,0x828 # IE=0
0x00003014	0xac00082c	sw $0,0x0000082c($0)	28:	sw $0,0x82C # IFG=0
0x00003018	0x20080009	addi $8,$0,0x00000009	29:	addi $t0,$zero,0x09
0x0000301c	0xac080820	sw $8,0x00000820($0)	30:	sw $t0,0x820 # BTIP=1, BTSSEL=1, BTHOLD=0
0x00003020	0x2008000f	addi $8,$0,0x0000000f	31:	addi $t0,$zero,0x0F
0x00003024	0xac080828	sw $8,0x00000828($0)	32:	sw $t0,0x828 # IE=0x0F
0x00003028	0x375a0001	ori $26,$26,0x00000001	33:	ori $k0,$k0,0x01 # $k0[0] uses as GIE
0x0000302c	0x8c080818	lw $8,0x00000818($0)	35:	lw $t0,0x818 # read the state of PORT_SW[7-0]
0x00003030	0x08000c0c	j 0x00003030	36: L:	j L
0x00003034	0xac080800	sw $8,0x00000800($0)	39:	sw $t0,0x800 # write to PORT_LEDG[7-0]
0x00003038	0xac080804	sw $8,0x00000804($0)	40:	sw $t0,0x804 # write to PORT_LEDR[7-0]
0x0000303c	0x8c09082c	lw $9,0x0000082c($0)	42:	lw $t1,0x82C # read IFG
0x00003040	0x3129fffe	andi $9,$9,0x0000fffe	43:	andi $t1,$t1,0xFFFE
0x00003044	0xac09082c	sw $9,0x0000082c($0)	44:	sw $t1,0x82C # clr KEY2IFG
0x00003048	0x03e00008	jr $31	45:	jr $ra # reti
0x0000304c	0xac080808	sw $8,0x00000808($0)	48:	sw $t0,0x808 # write to PORT_HEX0[7-0]
0x00003050	0xac08080c	sw $8,0x0000080c($0)	49:	sw $t0,0x80C # write to PORT_HEX1[7-0]
0x00003054	0x8c09082c	lw $9,0x0000082c($0)	51:	lw $t1,0x82C # read IFG
0x00003058	0x3129fffd	andi $9,$9,0x0000fffd	52:	andi $t1,$t1,0xFFFD
0x0000305c	0xac09082c	sw $9,0x0000082c($0)	53:	sw $t1,0x82C # clr KEY2IFG
0x00003060	0x03e00008	jr $31	54:	jr $ra # reti
0x00003064	0xac080810	sw $8,0x00000810($0)	57:	sw $t0,0x810 # write to PORT_HEX2[7-0]
0x00003068	0xac080814	sw $8,0x00000814($0)	58:	sw $t0,0x814 # write to PORT_HEX3[7-0]
0x0000306c	0x8c09082c	lw $9,0x0000082c($0)	60:	lw $t1,0x82C # read IFG
0x00003070	0x3129fffb	andi $9,$9,0x0000fffb	61:	andi $t1,$t1,0xFFFB
0x00003074	0xac09082c	sw $9,0x0000082c($0)	62:	sw $t1,0x82C # clr KEY3IFG
0x00003078	0x03e00008	jr $31	63:	jr $ra # reti
0x0000307c	0x21080001	addi $8,$8,0x00000001	65: BT_ISR:	addi $t0,$t0,1 # $t1=$t1+1
0x00003080	0xac080800	sw $8,0x00000800($0)	66:	sw $t0,0x800 # write to PORT_LEDG[7-0]
0x00003084	0x8c09082c	lw $9,0x0000082c($0)	68:	lw $t1,0x82C # read IFG
0x00003088	0x3129fff7	andi $9,$9,0x0000fff7	69:	andi $t1,$t1,0xFFF7
0x0000308c	0xac09082c	sw $9,0x0000082c($0)	70:	sw $t1,0x82C # clr BTIFG
0x00003090	0x03e00008	jr $31	71:	jr $ra # reti

Figures 17: test2 Assembly Code



ניתן לראות כי כל עוד אין פסיקה, התוכנית נשארת על פקודה מספר 0x3030. כאשר מתרחשת פסיקה של הטיימר, אנו מקבלים מה-InterruptController חיווי – ובעקבותיו מתחילה שרשרת הפעולה לטיפול בפסיקה. כחלק משרשרת זו הגדרנו את Opcode מספר 31, שאינו נמצא בשימוש אחר, כ-Opcode יעודי לפסיקות. ניתן לראות כי אכן הפקודה שמתבצעת היא בעלת Opcode 0xFC.

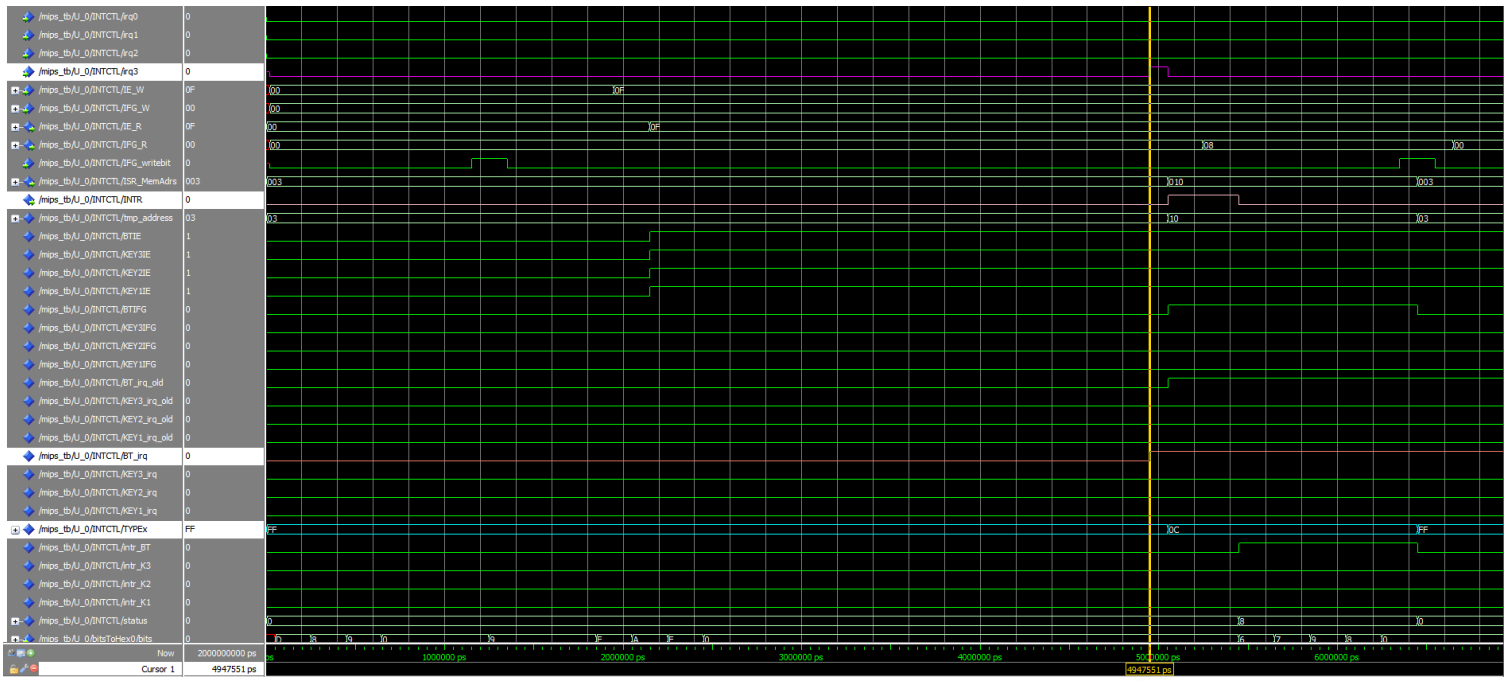


בתכלת – השעון שבחרנו.

בסגול העליון – דגל הפסיקה של הטיימר.

ניתן לראות כי BTIFG עולה ל-1 כאשר ערך ה-Counter מגיע ל-16 $(BTSSEL = 1 \Rightarrow Q3)$.

בכתום – *INTA* – יורד ל-0 מחזור אחד לאחר שקיבלנו פסיקה.



Figures 18-20: test2 Wave form analysis

בסגול– בקשת הפסיקה של הטיימר IRQ3.

ניתן לראות כי intr_BT עולה ל-1 (מתרחש כאשר נכנסים את הפסיקה הרלוונטית) והוא מונע מפסיקות נוספות להתבצע.

- **RTL viewer:**

ניתן להבחין ביחידות לוגיות המטפלות בניתוב הקריאה והכתיבה בין כתובות זיכרון רגילות לבין כתובות זיכרון הממופות ל- I\O:

```
-- IO / MEM
MemWrite_mem <= MemWrite AND (NOT(ALU_Result(11)));          -- Writing to mem / IO
--MemRead_mem    <= MemRead AND (NOT(ALU_Result(11)));
ioWrite      <= MemWrite AND ALU_Result(11);                 -- Writing to IO - see after components
ioRead       <= MemRead AND ALU_Result(11);

-- Added this line to get the switches value into register - memory >= 800h
read_data <= IO_out WHEN ioRead = '1'        -- read_data from IO switches
        ELSE read_data_tmp;                          -- normal read_data from DMEMORY (when ioRead, still has a value)

-- temp signal for ALU_res to address, OR interrupt address:
Address_ALU_res <=
        "00" & ISR_MemAdrs(10 DOWNTO 2) WHEN INTR_synced ='1' AND n=0   -- Interrupt address in memory (modelsim/quartus already taken care)
        ELSE ISR_MemAdrs(10 DOWNTO 2) & "00" WHEN INTR_synced ='1' AND n=1
        ELSE    "00" & ALU_Result(10 DOWNTO 2)  WHEN n=0                -- n=0 - Modelsim
        ELSE    ALU_Result(10 DOWNTO 2) & "00";                         -- n=/=0 - Quartus
                -- copy important signals to output pins for easy
                -- display in Simulator
```

Figure 21: IO/MEM Mapping

ניתן לראות כי הוספנו קווי בקרה בעזרתם אנו יודעים האם פונים ל- I\O.

דיאגרמת ה-RTL המלאה של המערכת

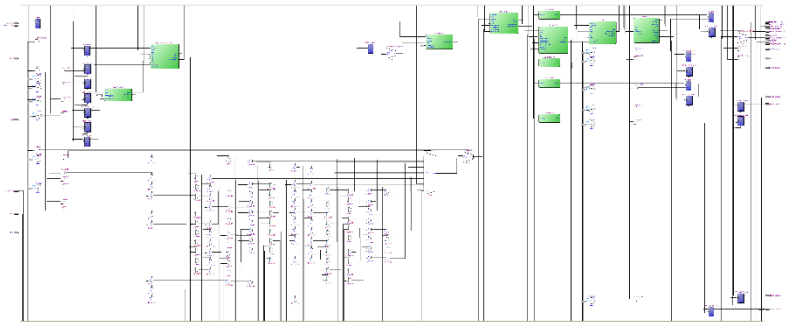


Figure 22: RTL view

להלן הבלוקים המתוארים בדיאגרמת ה-RTL בפירוט גדול יותר – כולל כניסות ויציאות, בהתאם לדיאגרמה המתארת את המערכת ללא הפריפריות:

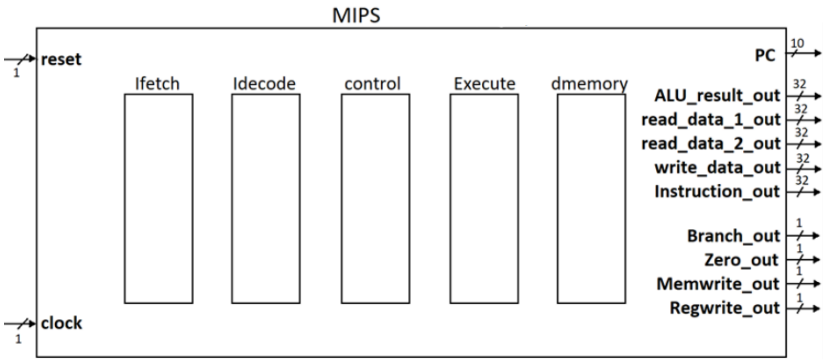


Figure 23: Top Block Diagram

Ifetch

שליפת הפקודה הבאה מהזיכרון בהתאם לקווי הבקרה הנכנסים והוצאת הפקודה הבאה.

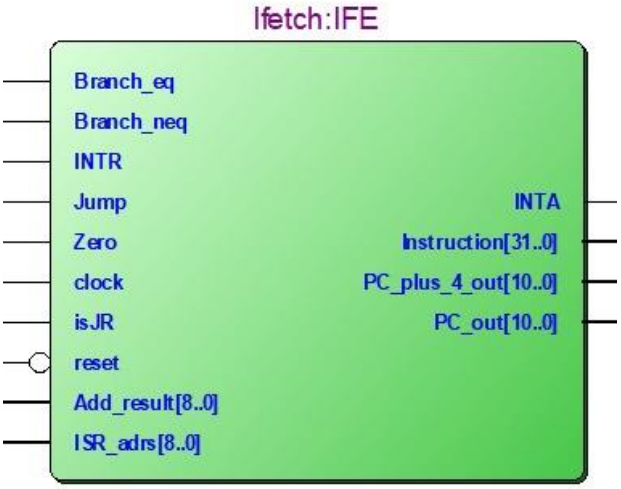


Figure 24: Ifetch block

Idecode

קבלת הפקודה הגולמית מה-Ifetch והוצאת סיגנלי מידע שאותם נצטרך בשביל ביצוע הפקודה.

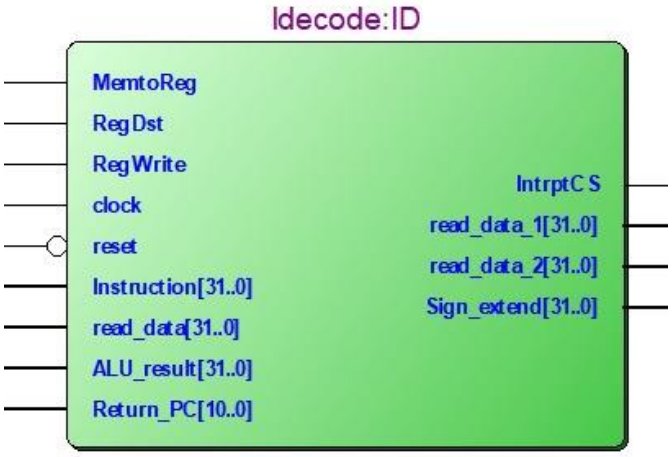


Figure 25: Idecode block

Execute

ביצוע הפקודה עצמה, כלומר פעולה אריתמטית (מתוך אחת מהפעולות הנתמכות במעבד). והוצאת תוצאה ודגלים מתאימים.

Add_Result – תוצאה המייצגת כתובת ולכן הוא רק באורך של שמונה ביטים.

Figure 26: Execute block

dmemory

מבצע קריאה או כתיבה לזכרון, מלבד לכתובות הממופות לרכיבי ה-O\I

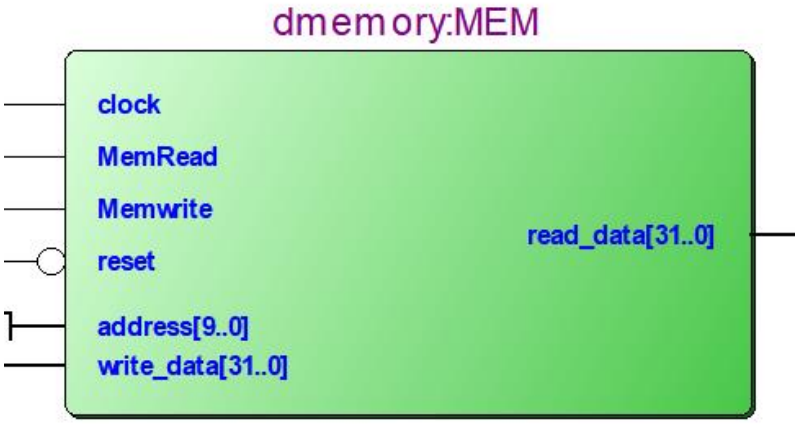


Figure 27: dmemory block

control

הוצאת קווי הבקרה המתאימים לפקודה (לפי ה-opcode המגיע מה-fetch)

Figure 28: control block

BasicTimer

טיימר - רכיב פריפריאלי – הניתן להגדרה עם חלוקות שונות של השעון – ובעל counter פנימי. בעל יכולת פסיקה

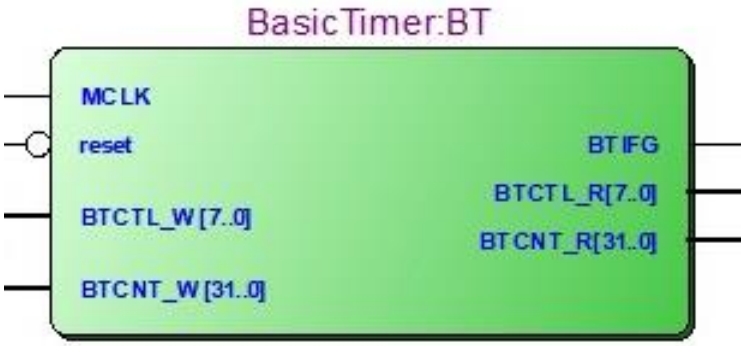


Figure 29: BasicTimer block

InterruptController

בקר פסיקות - רכיב פריפריאלי – בעל יכולת קבלה של פסיקות מ-4 מקורות שונים.

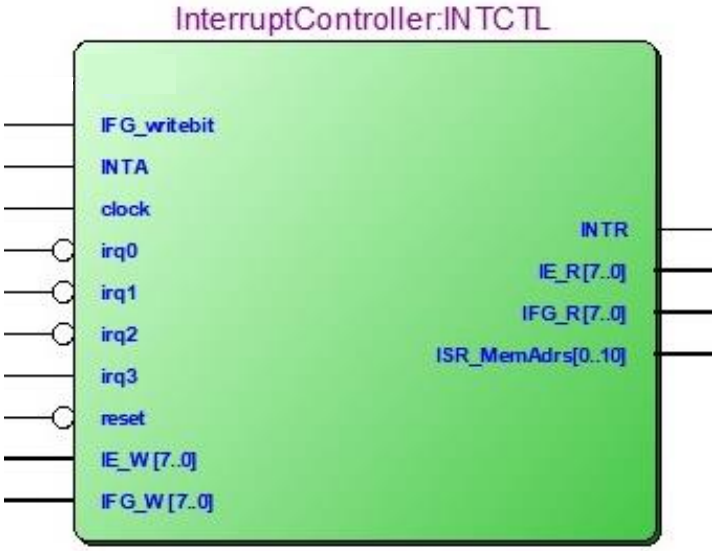


Figure 30: InterruptController block

Port Table : מצורפות טבלאות המתארות את הגודל, כיוון ופונקציונאליות של הקווים היוצאים מכל בלוק במערכת

MIPS			
Port	Direction	Size	Functionality
reset	in	1	
clk_24Mhz	in	1	
PC	out	11	
ALU_result_out	out	32	
read_data_1_out	out	32	
read_data_2_out	out	32	
write_data_out	out	32	
Instruction_out	out	32	
Branch_out	out	1	
Zero_out	out	1	
Memwrite_out	out	1	
Regwrite_out	out	1	
ENABLE	In	1	
KEY1	In	1	
KEY2	In	1	
KEY3	In	1	
Switches	in	8	
LEDG_out	buffer	8	
LEDR_out	buffer	8	
HEX0_out	buffer	7	
HEX1_out	buffer	7	
HEX2_out	buffer	7	
HEX3_out	buffer	7	
HEX4_out	buffer	7	
Zero4LEDR	out	2	To initiate LEDR 8 and 9 to 0 value.

Table 3: MIPS ports

IFETCH			
Port	Direction	Size	Functionality
Instruction	out	32	
PC_plus_4_out	out	11	The next command address
Add_result	in	9	Address value to branch / jump to
Branch_eq	in	1	Control signal indicates BEQ command
Branch_neq	in	1	Control signal indicates BNQ command
Zero	in	1	
Jump	in	1	Control signal indicates Jump command
isJR	In	1	Control signal indicates Jump Register command
PC_out	out	11	
INTR	In	1	
INTA	Out	1	
ISR_adrs	In	9	
clock	in	1	
reset	in	1	

Table 4: Ifetch ports

IDECODE			
Port	Direction	Size	Functionality
read_data_1	out	32	Data #1 to use in execute
read_data_2	out	32	Data #2 to use in execute
Instruction	in	32	
read_data	in	32	
ALU_result	in	32	
RegWrite	in	1	
MemtoReg	in	1	
Reg_dst	in	1	
Sign_extend	out	32	Sign extended immediate
IntrptCS	Out	1	
ReturnPC	In	11	ערך ה-PC שאליו נחזור לאחר ביצוע פסיקה
clock	in	1	
reset	in	1	

Table 5: Idecode ports

IExecute			
Port	Direction	Size	Functionality
Read_data_1	in	32	Data #1 to read from
Read_data_2	in	32	Data #2 to read from
Sign_extend	in	32	Sign extended immediate
Function_opcode	in	6	For “0” opcode
Opcode	in	6	Opcode number
ALUOp	in	2	With Function_opcode, I_opcode – decides which ALU operation will be executed
Jump	in	1	
isJR	in	1	
ALUSrc	in	1	
Zero	out	1	
ALU_Result	out	32	The result of the ALU (shift included)
Add_Result	out	9	
PC_plus_4	in	11	
Clock	in	1	
Reset	in	1	

Table 6: IExecute ports

DMEMORY			
Port	Direction	Size	Functionality
read_data	out	32	Memory data read
address	in	11	Address number in
write_data	in	32	Data to write
MemRead	in	1	ONLY for memory read (no I/O)
Memwrite	in	1	ONLY for memory write (no I/O)
clock	in	1	
reset	in	1	

Table 7: IDmemory ports

CONTROL			
Port	Direction	Size	Functionality
Opcode	in	6	
Fun_op	In	6	
RegDst	out	1	What register to write to
ALUSrc	out	1	Binput selector
MemtoReg	out	1	
RegWrite	out	1	
MemRead	out	1	Read control
MemWrite	out	1	Write control
Branch_eq	out	1	
Branch_neq	out	1	
ALUop	out	2	
Jump	out	1	
isJR	Out	1	
clock	in	1	
reset	in	1	

Table 8: Control ports

BasicTimer			
Port	Direction	Size	Functionality
MCLK	In	1	
reset	In	1	
BTCTL_W	in	1	Basic timer control write to
BTCTL_R	Out	1	Read from
BTCNT_W	in	1	Basic timer counter write to
BTCNT_R	Out	1	Read from
BTIFG	Out	1	Out flag

Table 9: BasicTimer ports

InterruptController			
Port	Direction	Size	Functionality
clock	In	1	
reset	In	1	
INTA	In	1	Interrupt Acknowledge
irq0	In	1	
irq1	In	1	
irq2	In	1	
irq3	In	1	
IE_W	In	8	Interrupt enable write to
IFG_W	In	8	Interrupts flags
IE_R	Out	8	Read from
IFG_R	Out	8	Read from
IFG_writebit	In	1	Notify bit
ISR_MemAdrs	Out	11	Address of ISR in dmemory
INTR	Out	1	Interrupt request

Table 10: InterruptController ports

I/O Memory (ports in MIPS.vhdl file)			
Port	Direction	Size	Functionality
MemWrite_mem	signal	1	‘1’ when Memwrite=’1’ and ALU_result is **not** the address of the I/O.
ioWrite	signal	1	‘1’ when Memwrite=’1’ and ALU_result **is** the address of the I/O.
ioRead	signal	1	‘1’ when MemRead=’1’ and ALU_result **is** the address of the I/O.
read_data	signal	32	Out will be from switches, when ioRead=’1’, else it will be “ DMEMORY - read_data” signal.

Table 11: I/O memory ports

- **Proof of work – Signal Tap Screenshots:**

של test2.asm – עם טריגר המוגדר עבור BTIFG (פסיקת הטיימר)

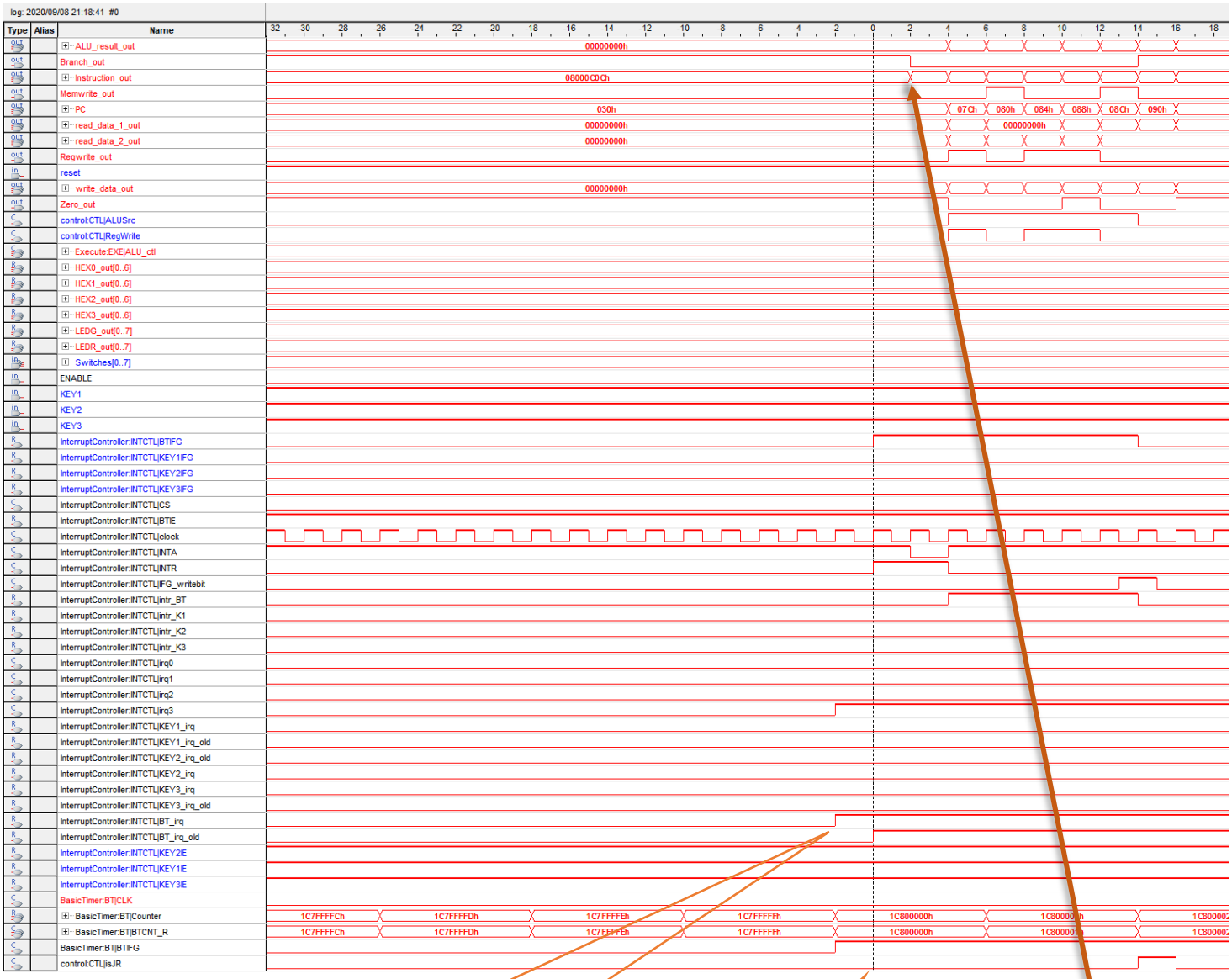


Figure 31: Signal Tap Basic Timer Interrupt

ניתן לראות כי בזמן זה עולה ה-BTIFG (היוצא מהטיימר) ולכן גם BT_irq עולה (בקר הפסיקות מקבל בקשה לפסיקה).

"מתקבלת" הפקודה המלאכותית לטיפול בפסיקה (OPCODE 31)

ניתן לראות כי בזמן זה יוצאת בקשת פסיקה (INTR) מבקר הפסיקות אל ה-MIPS מכיוון ש-BTIFG בבקר הפסיקות עלה.

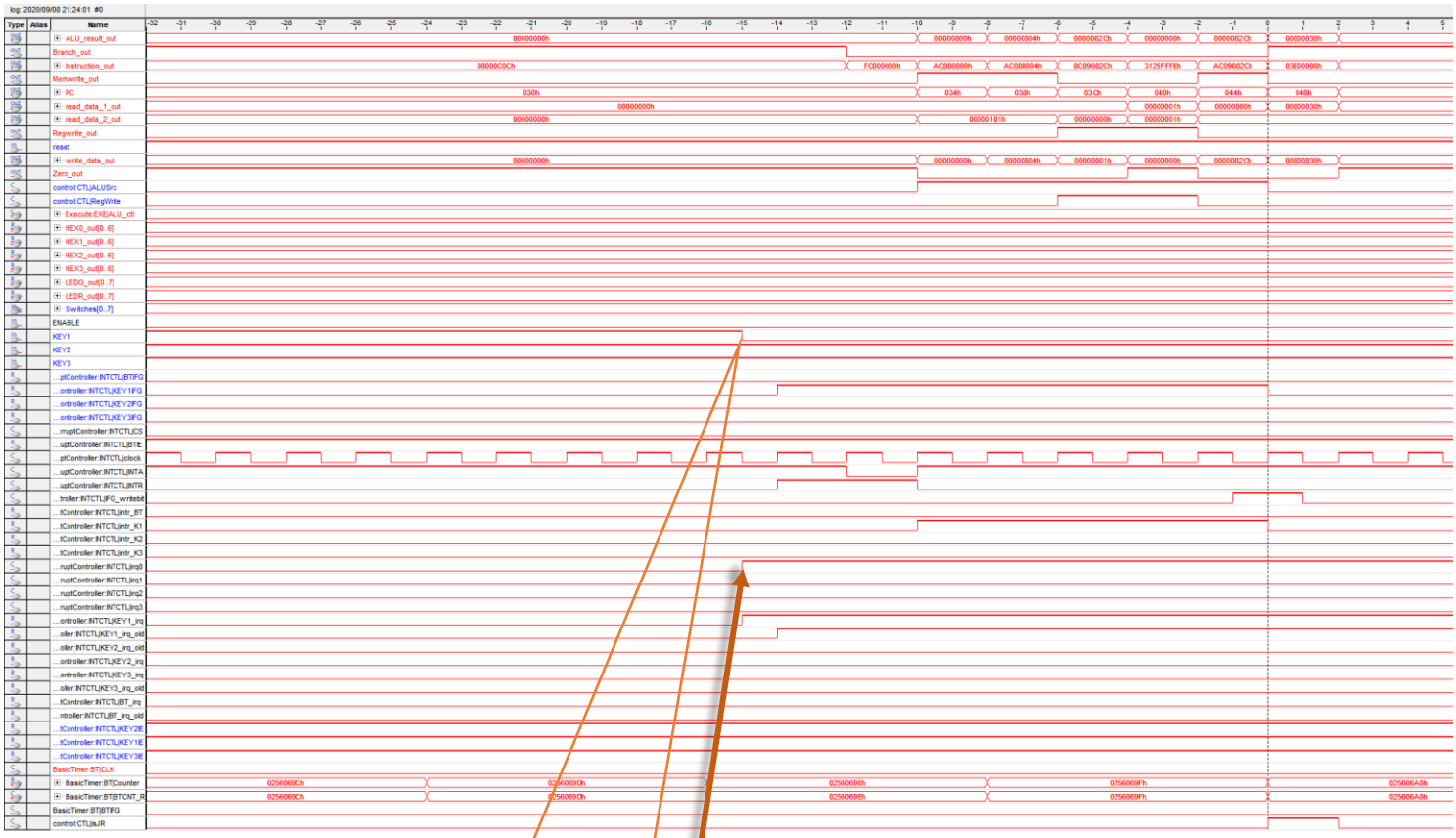


Figure 32: Signal Tap – Key 1 Interrupt

ניתן לראות כי בזמן זה התבצעה לחיצה על KEY1
ולכן גם irq0 עולה