רועי אלבז 205868011
דור פטשניק 205556533

Final Project

MCU: based on Pipeline MIPS CPU - FPGA

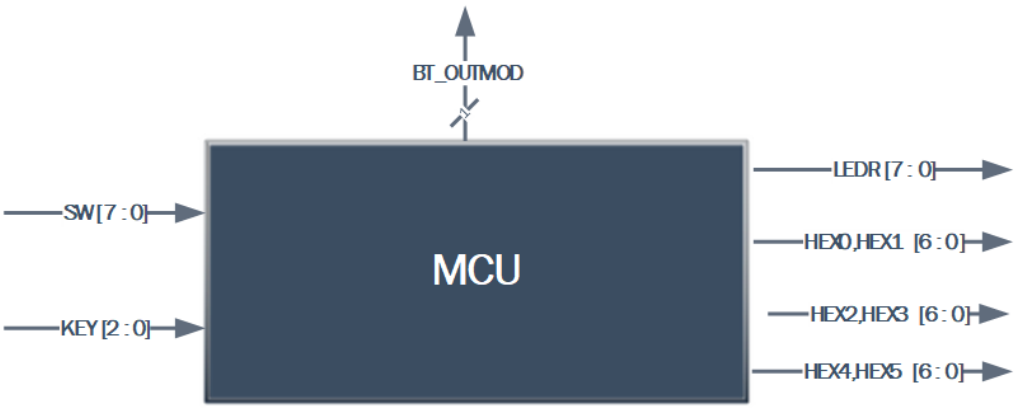
בדו"ח זה נציג את הניתוח מימוש המיקרו בקר שתכננו המבוסס על מעבד מסוג pipelined MIPS

Table of Contents

אופן פעולה

בחלק זה נסביר את אופן הפעולה של הדיזיין שלנו MCU, ונכלול גם את התוספות והשינוים אותם נדרשנו לבצע על מנת לתמוך במעבד מבוסס MIPS Pipeline.

דיאגרמת בלוקים כללית של הרכיב:



המיקרו-בקר שלנו מורכב מהרכיבים הבאים:

- מעבד MIPS מבוסס MIPS Pipeline
- מרכיבי GPIO בעלי יכולת קריאה וכתיבה (קריאה בלבד במקרה של רכיב INPUT)
- BASIC TIMER- בעל יכולת פסיקה בעת OVERFLOW ויציאת אות PWM בעל DUTY CYCLE משתנה.
- INTERRUPT CONTROLLER- בעל יכולת פסיקה עם יכולת PRIORITY של פסיקה ניתנות למיסוך ו-NMI. תומך בפסיקה מלחצנים, מTIMER ובביצוע RESET.

רכיבי GPIO:

מימשנו רכיבים אלה בצורה דומה להגדרת המשימה בתיאור הפרויקט. רכיבים אלה תלוים במודול Chip Select אשר מעביר את קו הבקרה המתאים בהתאם לקידוד של קווי הכתובת הרלוונטים.

רגיסטרים בעלי יכולת כתיבה וקריאה (כגון IFG, BTCNT וכו') מימשנו עם תוספת של בורר בכניסה לרגיסטר שנפתח רק כאשר מתבצעת כתיבה או קריאה מהרכיב ועוצר את פעולתו, על מנת למנוע מצב של MULTIPLE DRIVEN.

INTERRUPT CONTROLLER:

תוכנן כפי שהוגדר בהגדרת המשימה, עם תמיכה בפסיקות עם PRIORITY. מעלה קו בקרה INTR בתחילת פסיקה ומקבל קו בקרה INTA בתחילת פעולת הפסיקה ומסיים את פעולתו כאשר קו זה

חוזר לערך 1 לוגי. כמו כן הוא מחובר באופן בלעדי לכפתור הRESET ומבצע אותו בתור פסיקה. שאר המודולים במערכת מקבלים את הRESET ממודול זה על ידי קו בקרה סינכרוני NMI. כמו כן, בעת RESET אנו מאפסים גם את מוצאי רכיבי הGPIO על ידי ערך הTYPE הנמצא על גבי הDATABUS (ערכו הוא 0 בעת RESET)

MIPS Pipeline:

מעבד הPIPELINE ממעבדה 5 בעל מספר תוספות לתמיכה בפסיקות וGPIO:

1. GPIO – עבודה אך ורק בשלב הDMEMORY.
   a. תוספת של 2 TRISTATES בחיבור לdataBUS: במוצא הזכרון ובכניסת המידע לזכרון. (עבור כתיבה וקריאה)
   b. מניעת כתיבה לזכרון בעת פניה לרכיבי IO.
   c. חיבור קו הכתובת בכניסה לזכרון אל הaddressBUS.
2. INTERRUPT CONTROLLER –
   a. תוספת של MUX לקו הכתובת בכניסה לזכרון על מנת לכתוב את הTYPE בביצוע INTERRUPT.
   b. תוספת של קו EPC המחזיק את הכתובת הנכונה לקפיצה בשלב הEXECUTE. נדרשנו להוסיף לוגיקה לתמיכה בconditional jump אשר משנה את הערך הבא לקפיצה עקב שינוי הערך של הPC.
   c. תמיכה בקפיצה לכתובת הפסיקה על ידי 2 ביטי עזר ISR המבצעים אמולציה של פקודות LW וJAL.
   d. העלאה והורדה של ביט הGIE בתחילת רוטינת פסיקה ובקבלת פקודת RETI.
   e. הוספת STALL ליחידת הHAZARD בעת ביצוע כתיבה לIFG לפני חזרה מרוטינת פסיקה (מונע מצב של כניסה לפסיקה נוספת למרות שהדגל המתאים היה אמור לרדת)

ModelSim

נבצע את הניתוח בתוכנת ModelSim עבור מספר תוכניות שונות, ונבחן את התוצאות.

להלן קומפילציה של הדיזיין שלנו בתוכנת modelsim:

Project - C:/Users/elbaz/OneDrive/Documents/MODELSIM TB/FINAL_PROJECT/PROJECT

Name	Status	Type	Or	Modified
BTIMER.vhd	✓	VHDL	0	07/24/2023 08:00:27 pm
CONTROL.VHD	✓	VHDL	1	07/12/2023 07:50:20 pm
CS_DEC.VHD	✓	VHDL	2	07/16/2023 04:48:48 pm
Decoder.vhd	✓	VHDL	3	07/18/2023 02:47:24 pm
DMEMORY.VHD	✓	VHDL	4	07/24/2023 08:04:49 pm
EXECUTE.VHD	✓	VHDL	5	07/23/2023 07:00:31 pm
FORWARD.vhd	✓	VHDL	6	07/04/2023 03:13:08 am
GPI.VHD	✓	VHDL	7	07/17/2023 11:32:44 pm
GPIO_REG.VHD	✓	VHDL	8	07/18/2023 12:52:22 am
GPO.VHD	✓	VHDL	9	07/23/2023 08:36:07 pm
HAZARD.vhd	✓	VHDL	10	07/24/2023 01:43:06 am
IDECODE.VHD	✓	VHDL	11	07/23/2023 06:59:54 pm
IFETCH.VHD	✓	VHDL	12	07/24/2023 08:04:33 pm
INTERRUPT.vhd	✓	VHDL	13	07/23/2023 10:47:25 pm
MCU.vhd	✓	VHDL	14	07/23/2023 08:38:19 pm
MIPS.vhd	✓	VHDL	15	07/23/2023 07:15:32 pm
TriState.vhd	✓	VHDL	16	07/17/2023 11:28:28 pm
mips_tb_struct.vh...	✓	VHDL	17	07/18/2023 01:20:39 pm
mips_tester_struct...	✓	VHDL	18	07/23/2023 09:54:24 pm

איור 1: קומפילציה בMODELSIM עבור הMCU

בשלב הסימולציה במודל סים נציג 5 טסטים לדוגמא: 2 עבור תקינות הGPIO, 2 עבור ה interrupt controller ועוד בדיקה עבור אות הPWM של השעון.

GPIO:

קוד לדוגמא - Test0:

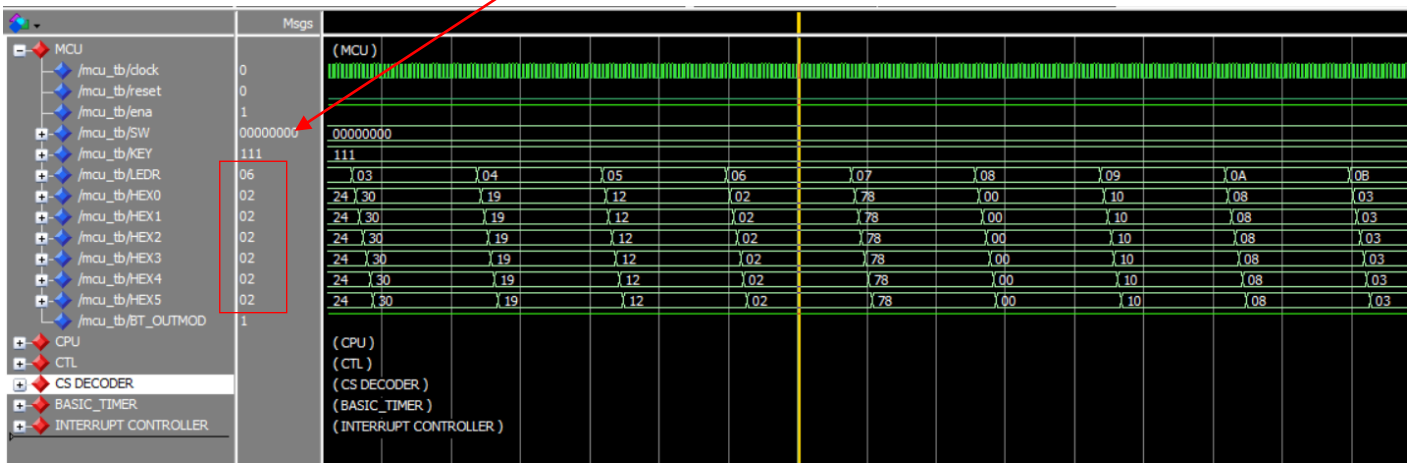
```
.data
        N: .word 0x0004
.text
        lw   $t3,N
        # lw   $t0,0x810 # read the state of PORT_SW[7-0]
        move $t0,$zero  # $t0=0
Loop:
        sw   $t0,0x800 # write to PORT_LEDR[7-0]
        sw   $t0,0x804 # write to PORT_HEX0[7-0]
        sw   $t0,0x805 # write to PORT_HEX1[7-0]
        sw   $t0,0x808 # write to PORT_HEX2[7-0]
        sw   $t0,0x809 # write to PORT_HEX3[7-0]
        sw   $t0,0x80C # write to PORT_HEX4[7-0]
        sw   $t0,0x80D # write to PORT_HEX5[7-0]

        addi $t0,$t0,1 # $t0=$t0+1
        move $t1,$zero  # $t1=0
delay:  addi $t1,$t1,1  # $t1=$t1+1
        slt  $t2,$t1,$t3      #if $t1<N than $t2=1
        beq  $t2,$zero,Loop   #if $t1>=N then go to Loop label
```

איור 2: קוד אסמבלי לדוגמא – test0

בקוד זה אנו בודקים את תקינות רכיבי הGPO: LEDR,HEX. מבצעים מניה מעלה החל מהערך 0 וטוענים אותו ללדים ולנורות הHEX, עם דיליי של N.

בתמונה זו ניתן לראות את אופן הפעולה התקין – מניה מעלה של רכיבי הGPO שלנו. נציין כי ערכי הHEX הן לאחר DECODE על מנת שיתאים לתצוגה על גבי הFPGA ולכן הם שונים מהערכים על הלדים:

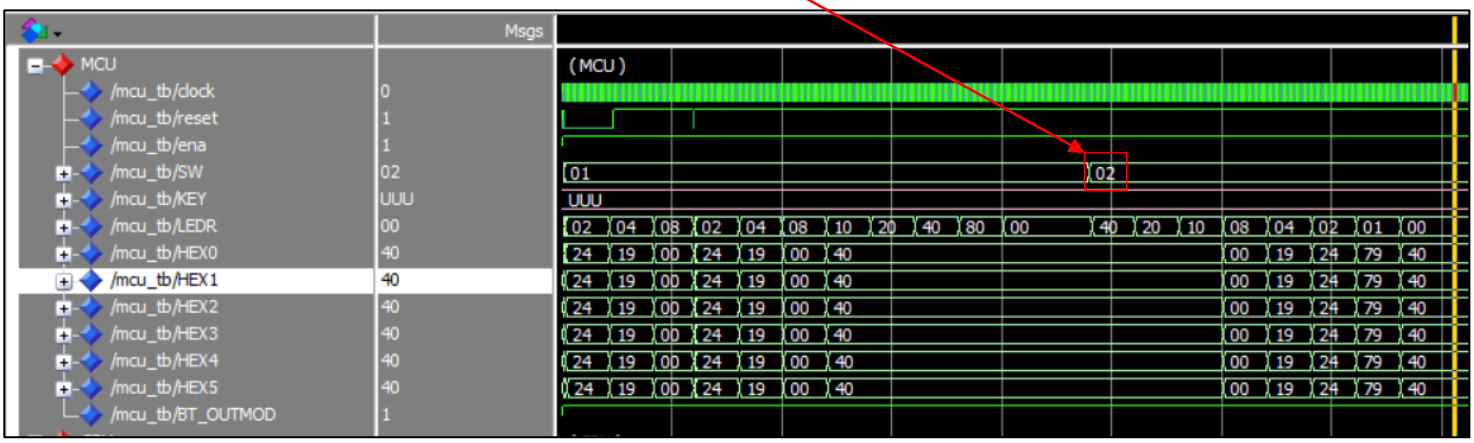


איור 3: ביצוע מניה מעלה של ערכי HEX ו LEDR

קוד לדוגמא – Test2:

בקוד זה אנו בודקים את תקינות רכיבי הGPO: LEDR,HEX ואת רכיב הSWITCHES GPI. אנו מבצעים כפל ב-2 או חלוקה ב-2 של הערך ב$t0 בצורה סריאלית, בהתאם לערך דיליי N. Sw(0)=1 גורר כפל וsw(1)=1 גורר חילוק.

בתמונה זו ניתן לראות את אופן הפעולה התקין – כאשר SW הוא 1 מתבצע כפל ב-2 והצגה של הערך על גבי רכיבי הGPO שלנו. לאחר מכן SW משתנה ל2 ואנו עוברים לביצוע חילוק ב-2.



איור 4: ביצוע כפל\חילוק ב2 והצגה על גבי הLEDR וHEX

:INTERRUPT I/O

:Test2 – קוד לדוגמא

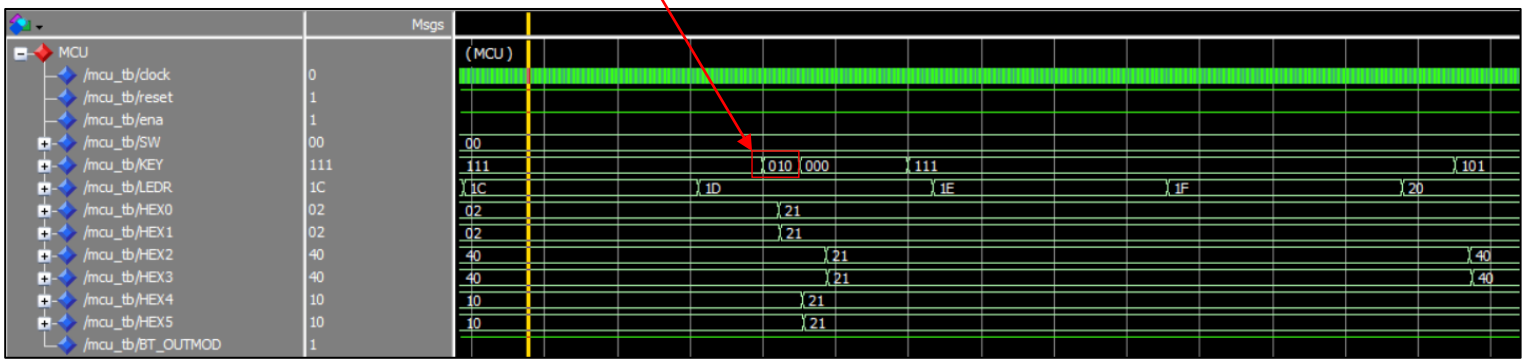
בקוד זה אנו בודקים את תקינותו של בקר הפסיקות אל מול הלחצנים KEY1-3 והBASIC TIMER. בנוסף עבודה עם הLEDR וHEX. בהתחלה מבצעים אתחול של השעון ומאפשרים פסיקות.

בעת פסיקה אמורות להתבצע הפעולות הבאות:

KEY1-3 – כתיבה של ערך הלדים HEX4-5 / HEX2-3 / HEX0-1 בהתאמה.

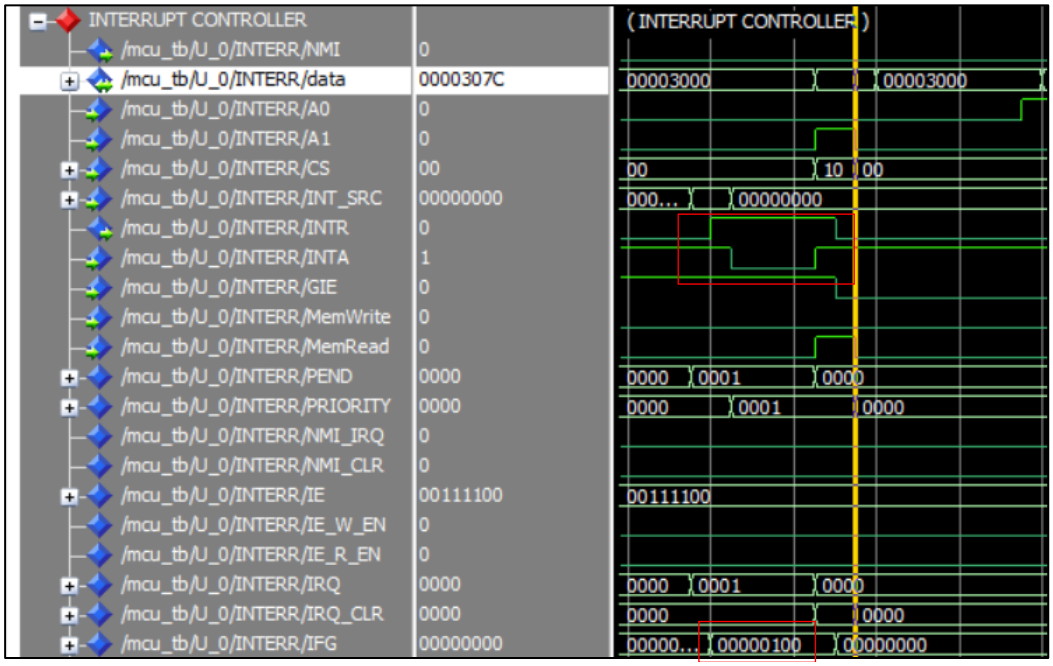
BASIC TIMER – מנייה מעלה, החל מערך הSW וכתיבה של הערך ללדים.

בתמונה זו ניתן לראות את המניה של הלדים עקב פסיקות של השעון ואת עדכון הערכים של הלדים בHEX המתאימים, בהתאם ללחצן שנבחר. בחלק זה ניתן גם להבחין בPRIORITY- KEY0 וKEY2 נלחצו בו זמנית וביצענו פסיקה ל-0 שהוא בעל העדיפות הגבוהה מביניהם.



איור 5: ביצוע פסיקות לחצנים וטיימר והצגה על גבי הLEDR וHEX

בתמונה זו ניתן לראות דוגמא לפעולת הפסיקה במודול הintrerrupt controller. ברגע העליה של INTR יורד ביט INTA ועולה חזרה בסיום הפעולה. כמו כן, בסיומה של הפעולה אנו כותבים לBUS את ערך הtype ומקבלים את ערך כתובת רוטינת הפסיקה מהזכרון. ניתן גם לראות ירידה של GIE.



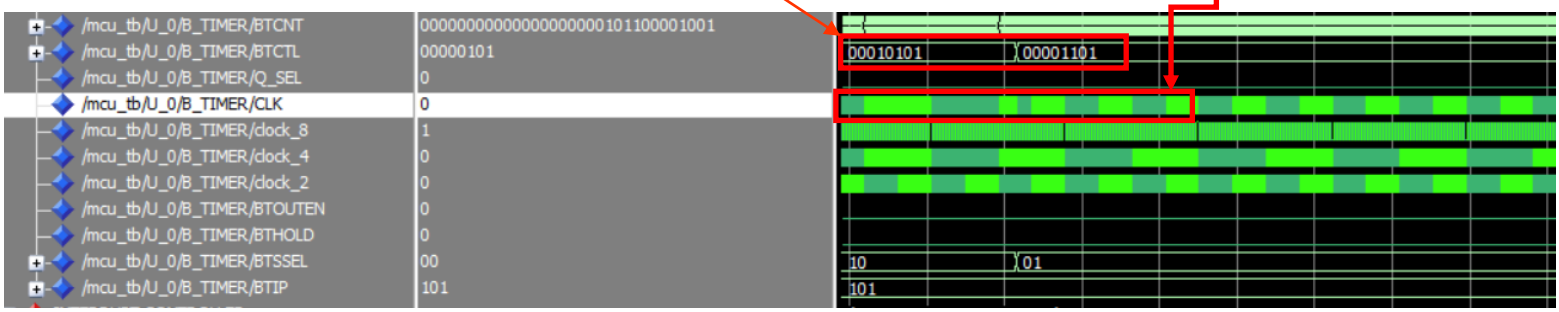
איור 6: ביצוע רוטינת פסיקה

קוד לדוגמא – Test3:

בקוד זה אנו בודקים את תקינותו של בקר הפסיקות אל מול הלחצנים KEY1-3 והBASIC TIMER. בנוסף עבודה עם הLEDR וHEX. בהתחלה מבצעים אתחול של השעון ומאפשרים פסיקות.

בעת לחיצה על הלחצנים, נקנפג מחדש את רגיסטר הבקרה של הטיימר ובכך נשנה את מהירות המנייה של הלדים. הbasic timer מונה ערך ללדים בכל פסיקה.

כאן ניתן לראות את הכניסה לפסיקת אחד הלחצנים ושינוי ערך רגיסטר הבקרה של הטיימר. בעקבות כך שינינו את תדר השעון ואת קצב המניה של הלדים.

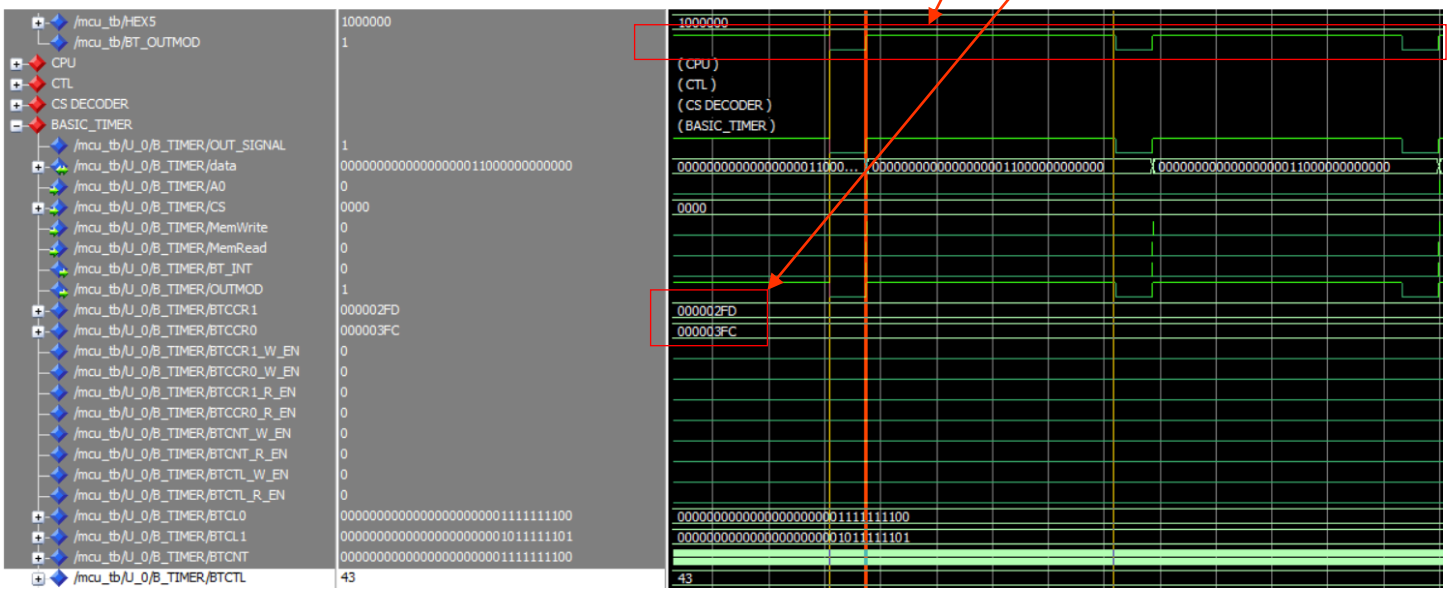


איור 7: שינוי פעולת הטיימר עקב פסיקת לחצן

קוד לדוגמא – OUTMOD:

בקוד זה אנו בודקים את תקינותו של מוד שידור אות הPWM שלנו.

להלן דוגמא לשידור אות PWM עם DUTYCYCLE של כ 88.5%. אות הPWM נקבע על ידי ערכי הרגיסטרים BTCCR0,BTCCR1. מכיוון שאנו עובדים במוד של toggle/set, הDUTYCYCLE נקבע לפי ערך המניה של הטיימר (במקרה שלנו $2^{11} = 2048$) חלקי ההפרש בין הרגיסטרים BTCCR0,BTCCR1 (במקרה שלנו $3FC - 2FD = FF$) ולכן נקבל שב$\frac{7}{8}$ מהמחזור אנו ב1 לוגי, כפי שקיבלנו.

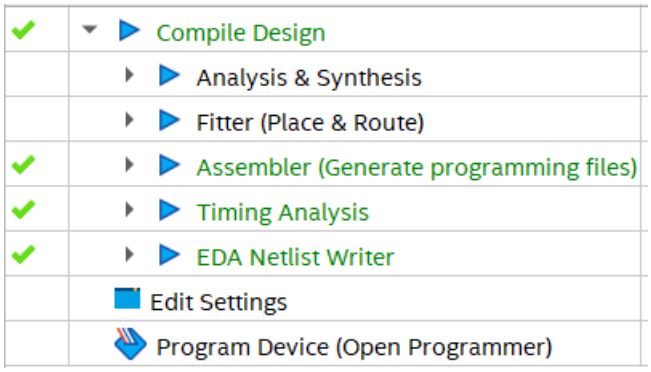


איור 8: אות PWM על ידי הטיימר

Quartus

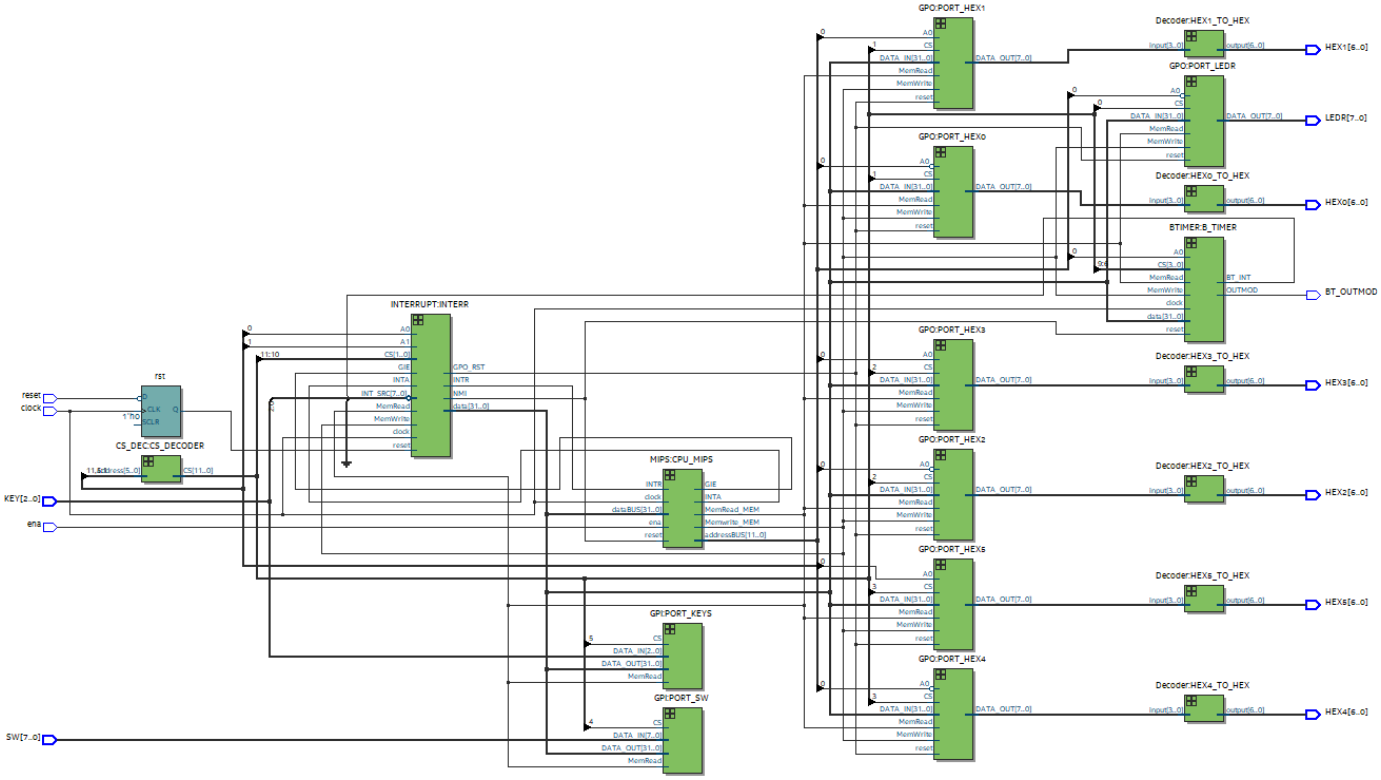
MCU

1. ראשית, קומפילציה של הMCU שלנו:



איור 9: קומפילציה של Quartus ללא הקצאות פינים

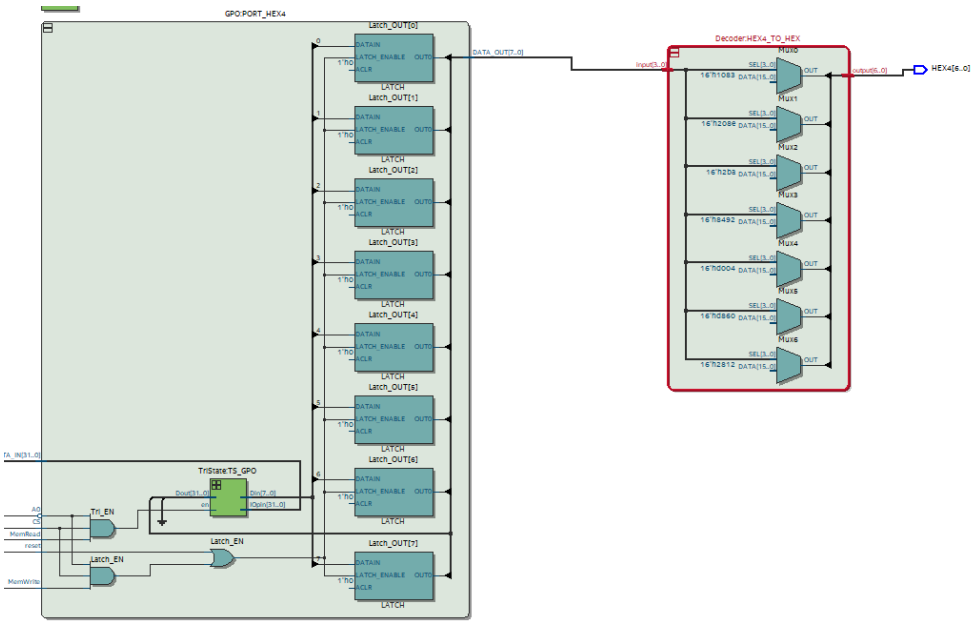
RTL של הדיזיין:



איור 10: MCU - RTL viewer

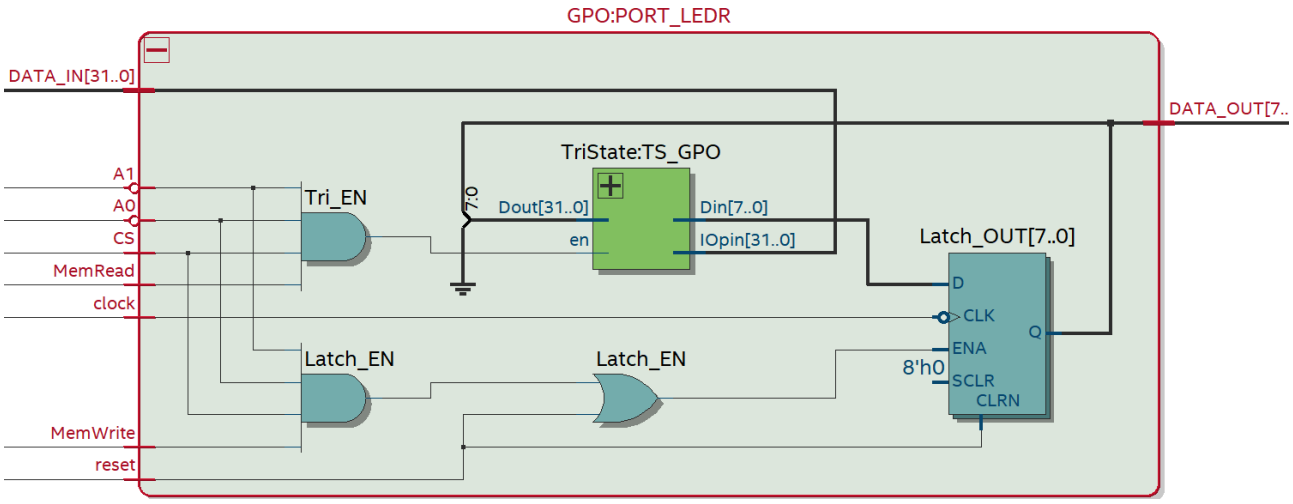
כעת נציג את הRTL של כל אחד מרכיבי הDESIGN.

<u>מבנה כללי של HEX (קיימים 6 כאלה):</u>



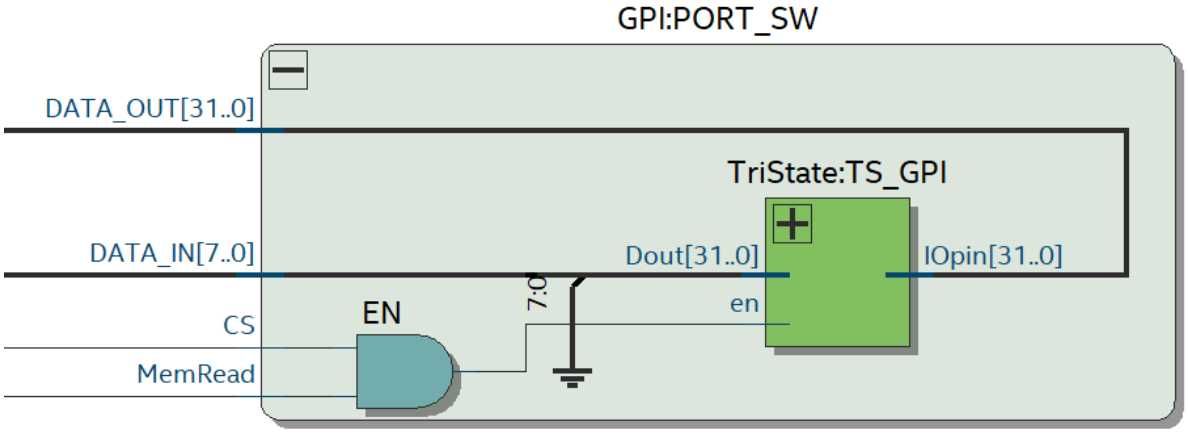
איור 11: HEX - RTL

:(GPO) LEDR



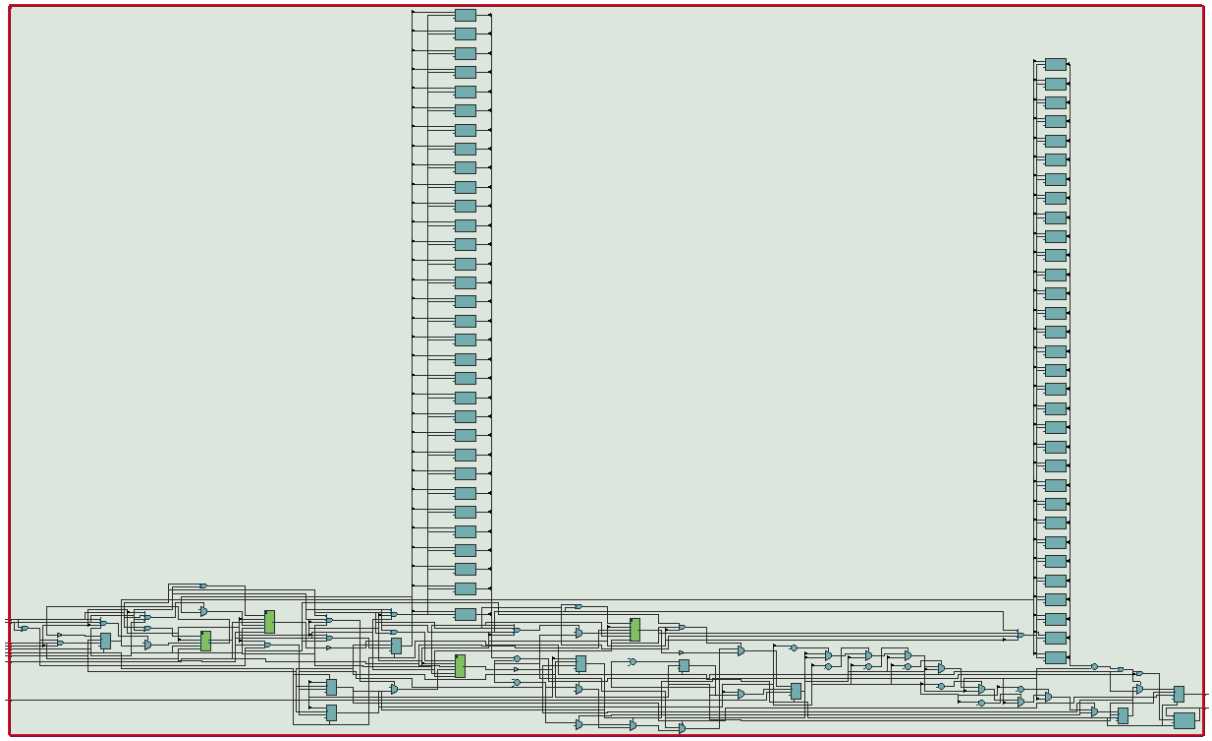
איור 12: RTL - LEDR

(מימוש זהה עם מספר ביטים שונה) :(GPI) SWITCH, KEY



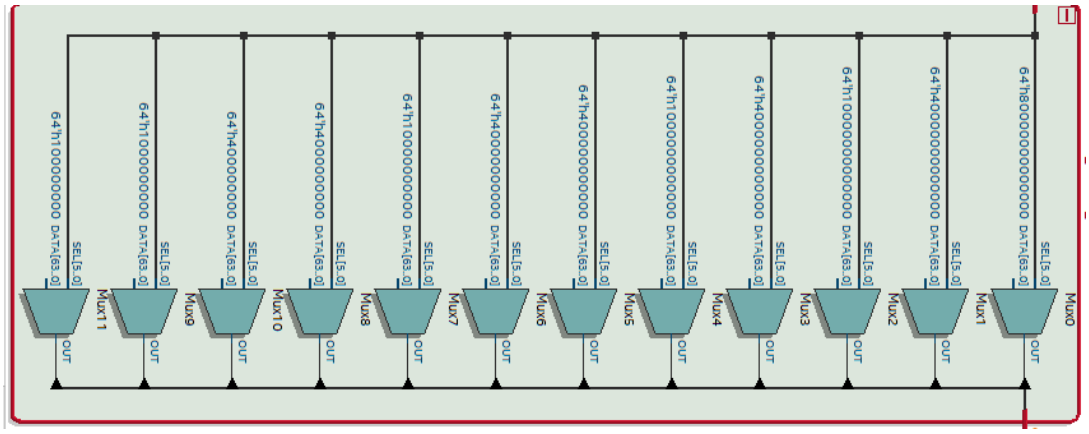
איור 13: RTL – KEYS,SW

:BASIC TIMER



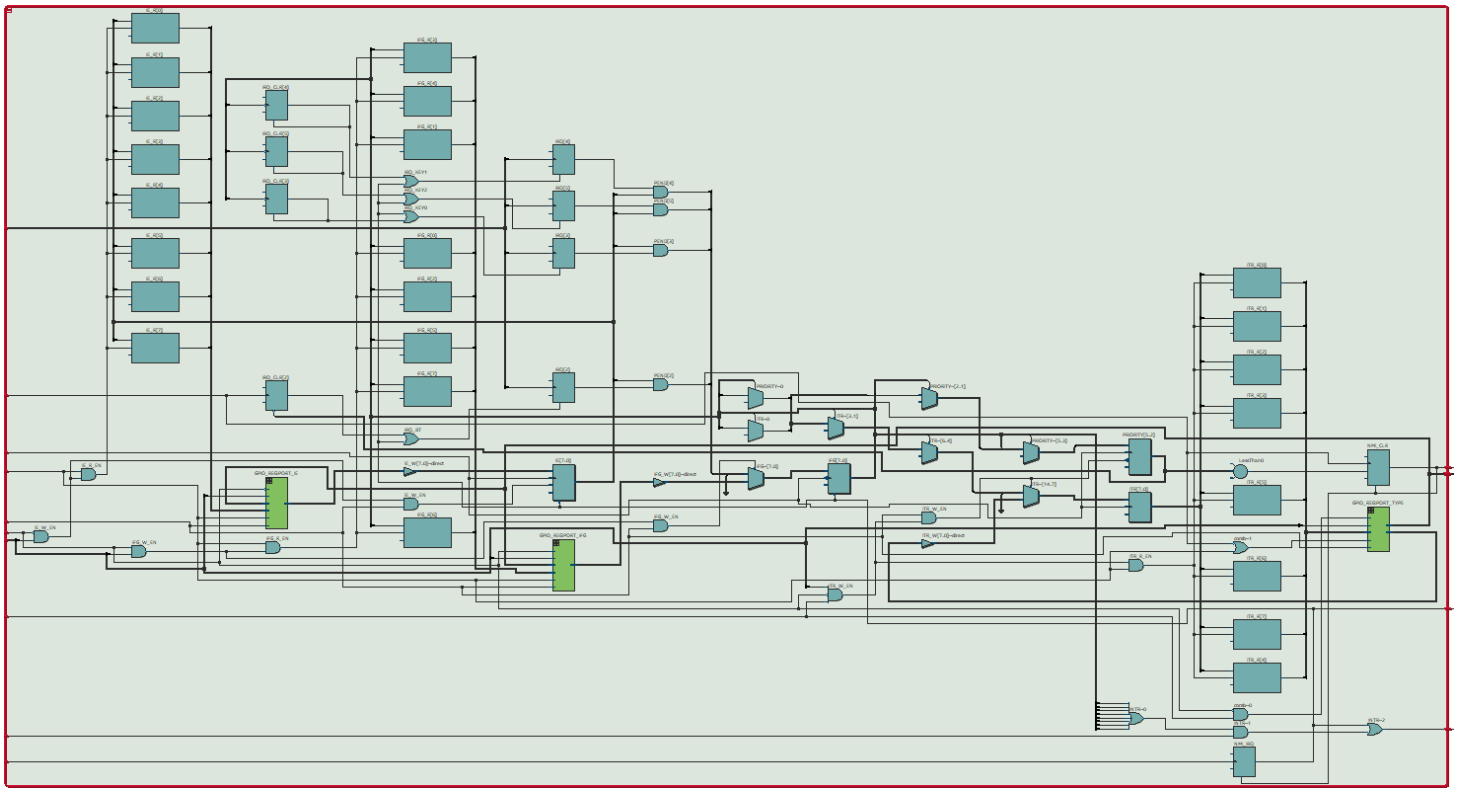
איור 16: BASIC TIMER – RTL

CHIP SELECT DECODER



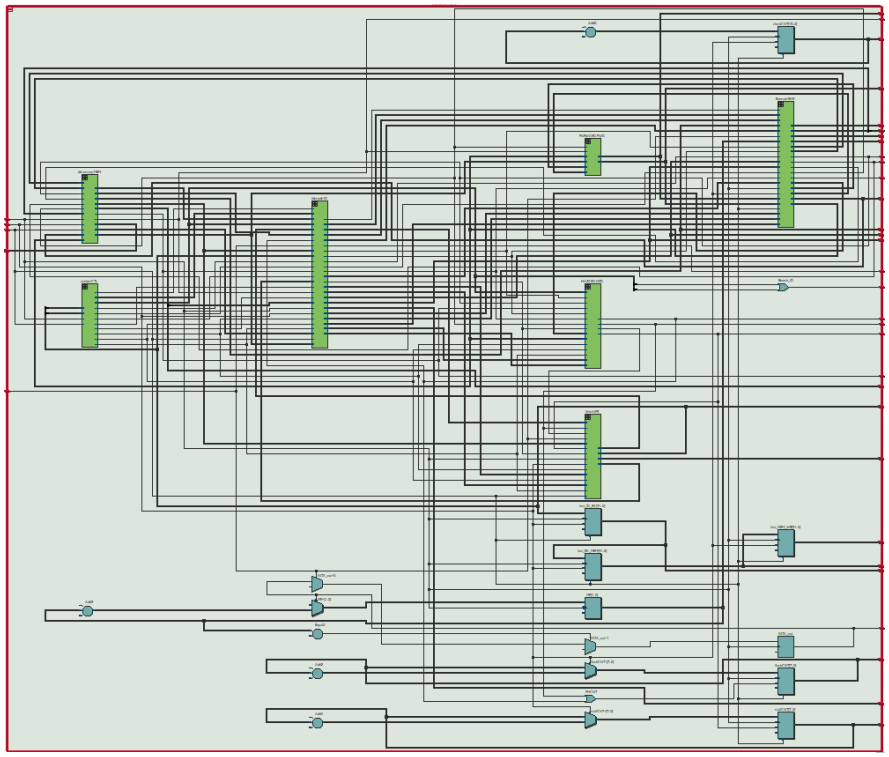
איור 15: CHIP SELECT – RTL

:INTERRUPT CONTROLLER



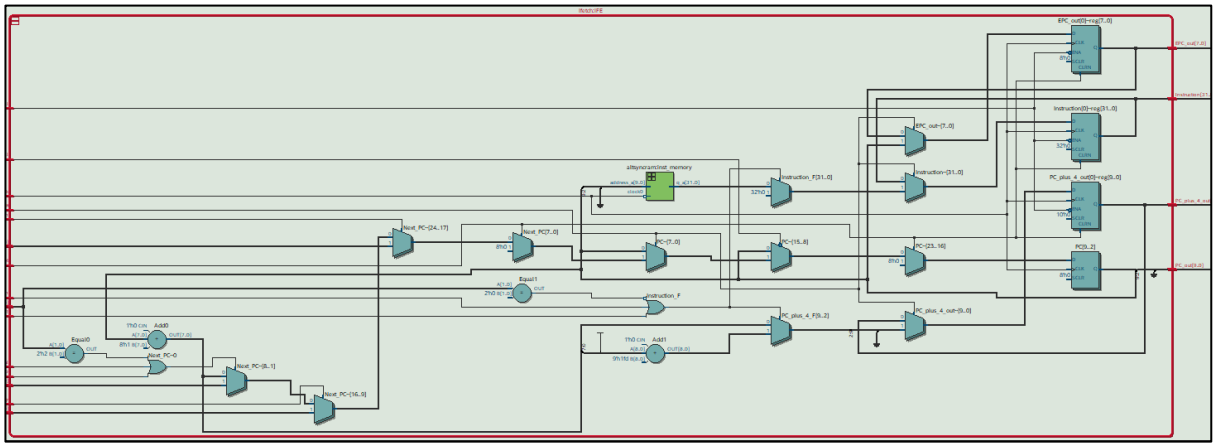
INTERRUPT CONTROLLER – RTL :16 איור

MIPS – TOP ENTITY



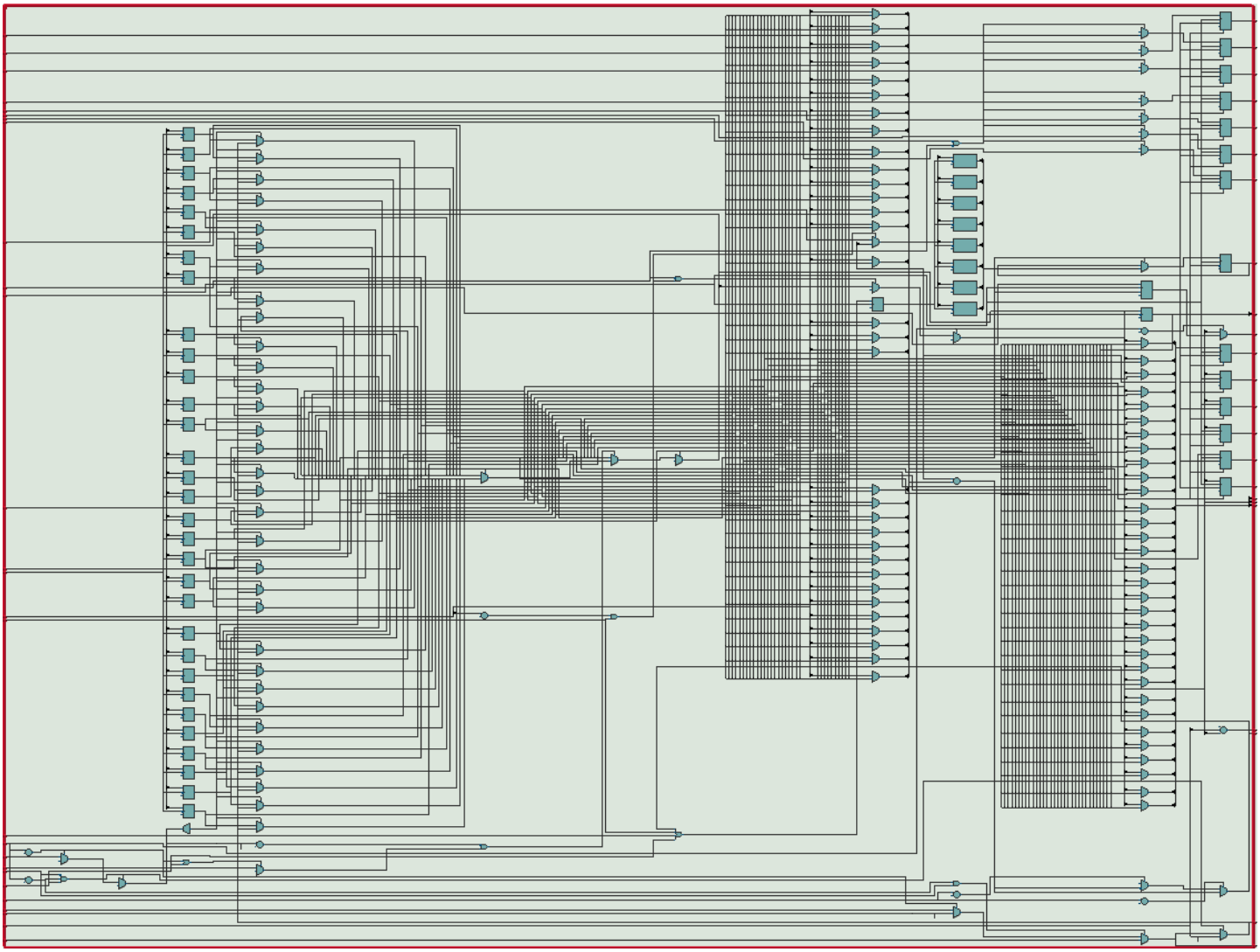
RTL – TOP LEVEL - PIPELINE :17 איור

IFETCH



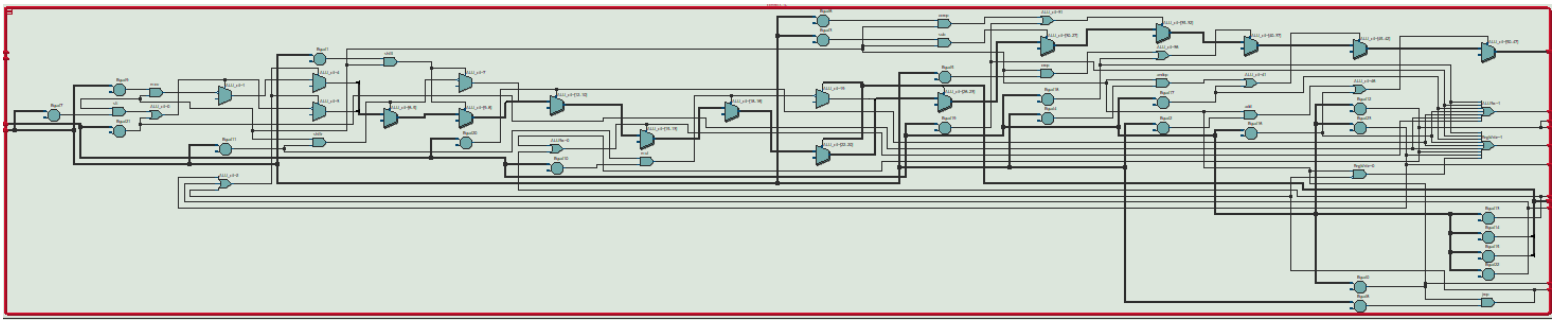
RTL – FETCH - PIPELINE :18 איור

DECODE



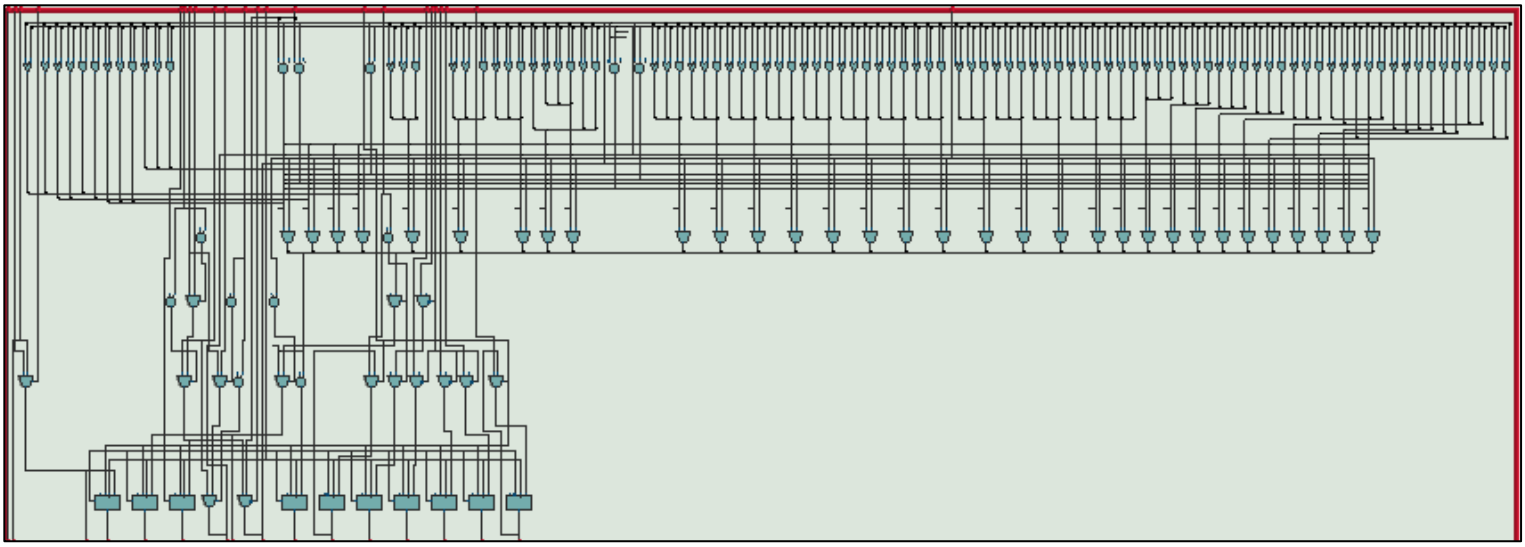
RTL – DECODE- PIPELINE :19 איור

CONTROL



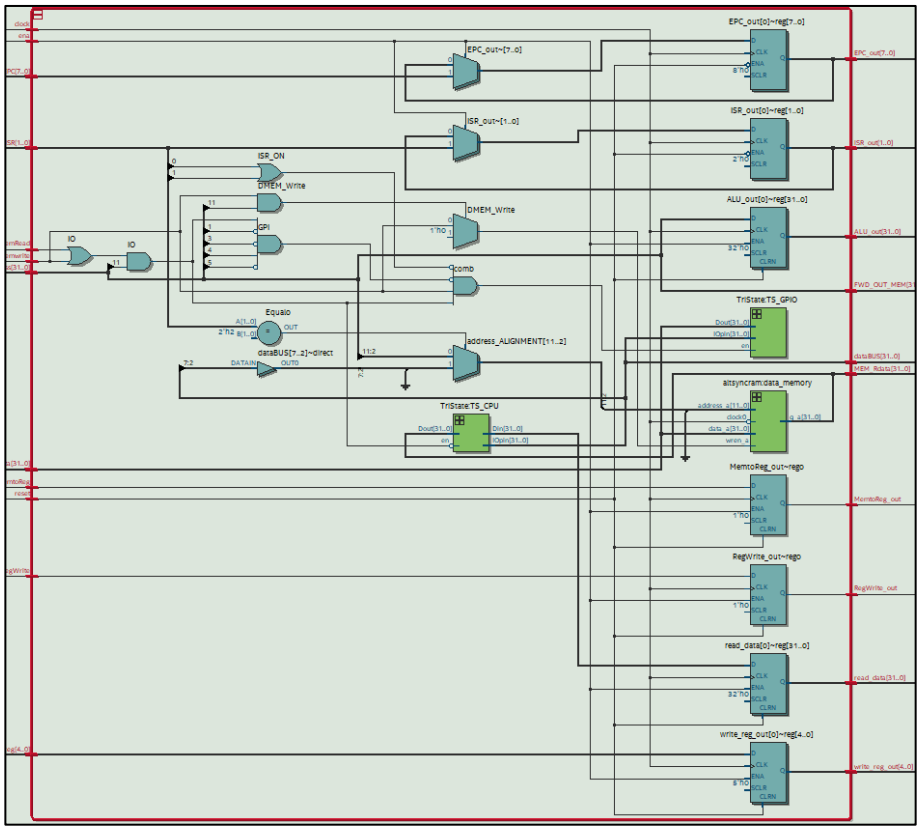
RTL – CONTROL- PIPELINE :20 איור

EXECUTE



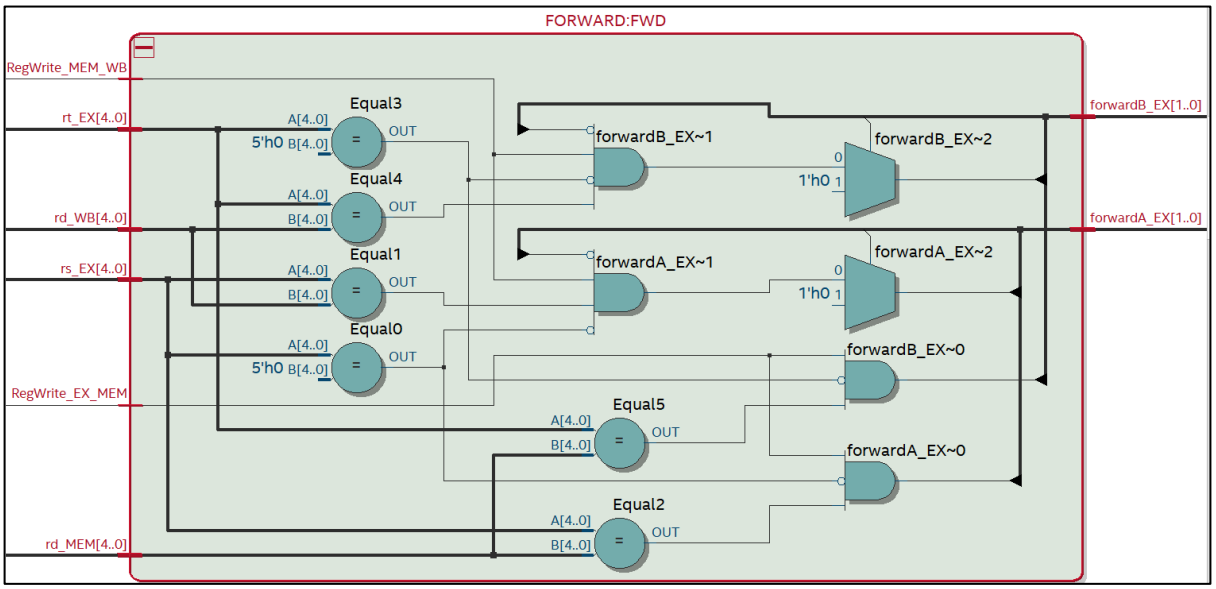
RTL – EXECUTE- PIPELINE :21 איור

MEMORY



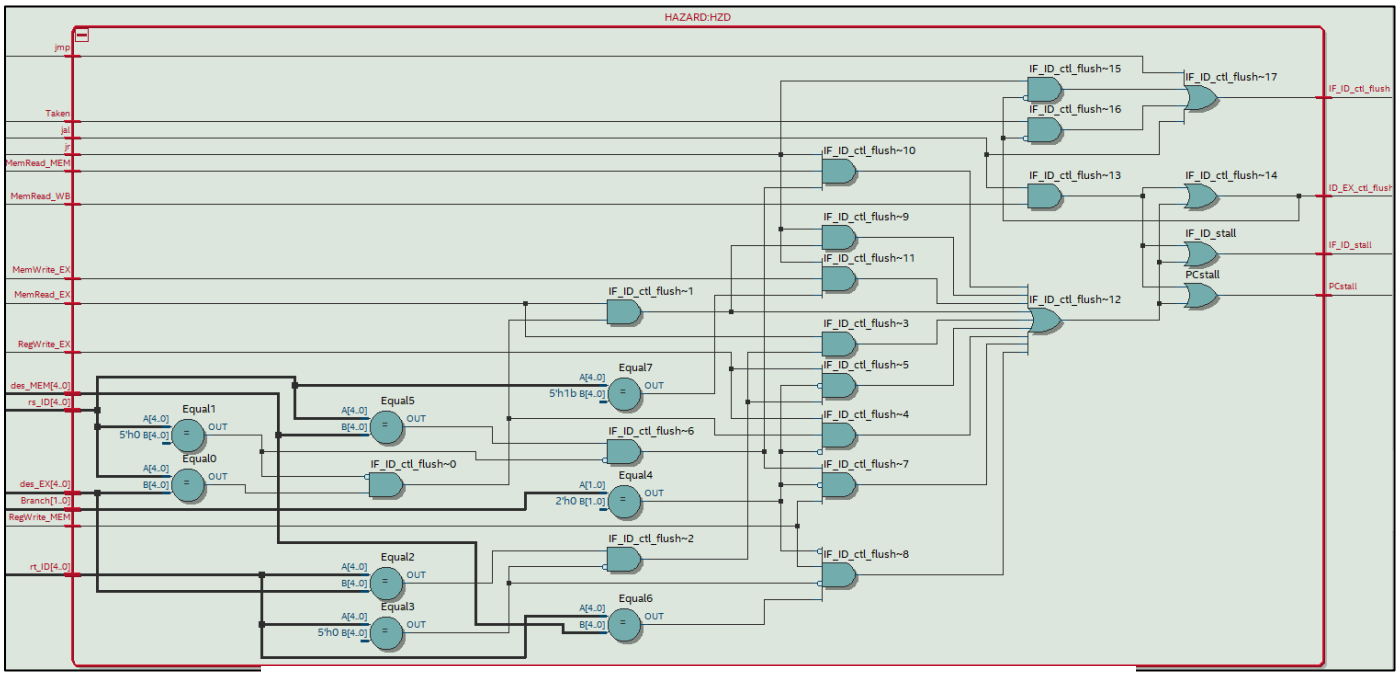
RTL – DMEMORY- PIPELINE :22 איור

FORWARDING UNIT



RTL – FORWARD- PIPELINE :23 איור

HAZARD UNIT



איור 24: RTL – HAZARD - PIPELINE

2. השלב הבא היה למצוא את תדר הפעולה המקסימלי –

	Fmax	Restricted Fmax	Clock Name	Note
1	60.52 MHz	60.52 MHz	clock	

איור 25: תדר הפעולה המקסימלי

תדר זה יקבע את תדר העבודה ולכן נכניס אותו לקובץ הconstraints שלנו:

```
#**************************************************************
# Time Information
#**************************************************************

set_time_format -unit ns -decimal_places 3



#**************************************************************
# Create Clock
#**************************************************************

create_clock -name {clock} -period 20.000 -waveform {0.000 10.000} [get_ports { clock }]
```

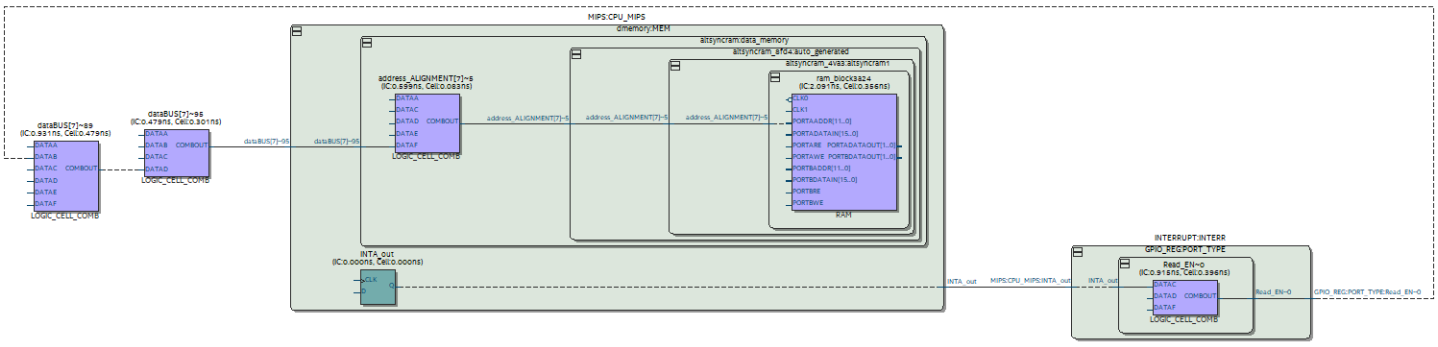
אאיור 26: קובץ SDC המכיל את הCONSTRAINS של השעון

כעת נראה את נתון הlogic usage עבור הדיזיין:

	Resource	Usage
1	Estimate of Logic utilization (ALMs needed)	1846
2		
3	▼ Combinational ALUT usage for logic	2377
1	-- 7 input functions	38
2	-- 6 input functions	1049
3	-- 5 input functions	428
4	-- 4 input functions	239
5	-- <=3 input functions	623
4		
5	Dedicated logic registers	1790
6		
7	I/O pins	65
8	Total MLAB memory bits	0
9	Total block memory bits	163840
10		
11	Total DSP Blocks	2
12		
13	Maximum fan-out node	clock~input
14	Maximum fan-out	1577
15	Total fan-out	20507
16	Average fan-out	4.69

איור 27: MCU - LOGIC USAGE

הנתיב הקריטי של מעבד זה עובר מהCPU אל הinterrupt controller, בחזרה ליחידת ה data memory בCPU דרך הDATABUS. זה מסתדר לנו עם התיאוריה שכן פעולה זו היא כחלק מפנייה לרכיבי GPIO שכידוע בעלי זמן ההשהייה הארוך ביותר והיא מתרחשת כאשר נרצה לטעון את הTYPE בכניסה לזכרון המידע בעת תחילת רוטינת פסיקה. פעולה זו עוברת מספר רב של רכיבים ואף כוללת מעבר דרך הDATABUS:



איור 28: MCU - CRITICAL PATH

3	12.760	6.630					data path
1	6.130	0.000		uTco	1	FF_X36_Y3_N38	MIPS:CPU_MIPS|INTA_out
2	6.130	0.000	RR	CELL	6	FF_X36_Y3_N38	CPU_MIPS|INTA_out|q
3	7.045	0.915	RR	IC	1	LABCELL_X37_Y5_N21	INTERR|PORT_TYPE|Read_EN~(
4	7.441	0.396	RF	CELL	8	LABCELL_X37_Y5_N21	INTERR|PORT_TYPE|Read_EN~(
5	8.372	0.931	FF	IC	1	LABCELL_X40_Y7_N39	dataBUS[7]~89|datab
6	8.851	0.479	FF	CELL	2	LABCELL_X40_Y7_N39	dataBUS[7]~89|combout
7	9.330	0.479	FF	IC	1	LABCELL_X40_Y7_N18	dataBUS[7]~95|datad
8	9.631	0.301	FR	CELL	1	LABCELL_X40_Y7_N18	dataBUS[7]~95|combout
9	10.230	0.599	RR	IC	1	LABCELL_X42_Y6_N24	CPU_MIPS|MEM|address_ALIGN
10	10.313	0.083	RF	CELL	16	LABCELL_X42_Y6_N24	CPU_MIPS|MEM|address_ALIGN
11	12.404	2.091	FF	IC	1	M10K_X26_Y2_N0	CPU_MIPS|MEM|data_memory|
12	12.760	0.356	FF	CELL	0	M10K_X26_Y2_N0	MIPS:CPU_MIPS|dmemory:MEM

איור 29: פירוט הזמנים של הcritical path - MCU

Signal Tap

נרצה לבצע ווריפיקציה של החומרה שיצרנו בעזרת תוכנת ה $Signal\ Tap$ של $Quartus$,

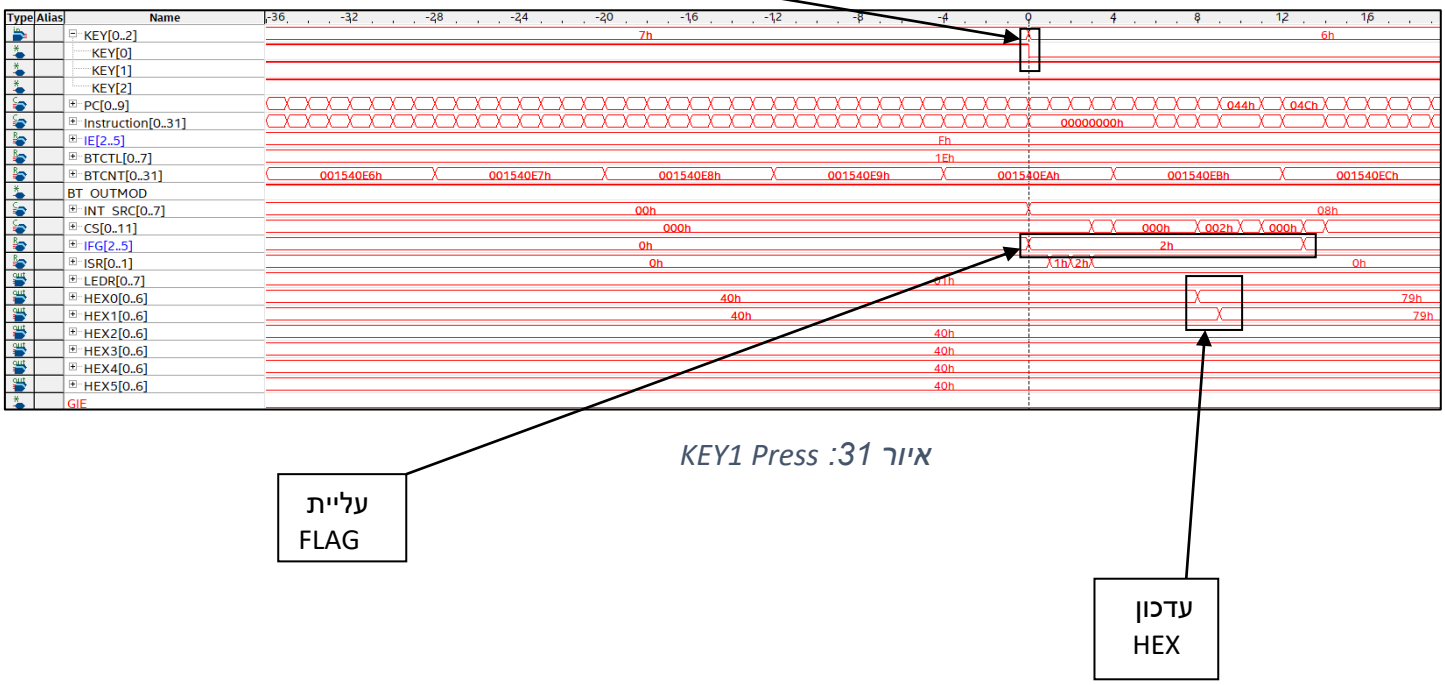
נרצה להציג בזמן אמת את ערכי סיגנלים נבחרים עבור וריפיקציה מיטבית של הדיזיין שלנו. לכידת ערכים בsignal tap תתבצע בעת לחיצה על אחד הלחצנים או בעת הורדת הENA. בfpga נוכל לראות בצורה דינמית את השפעת הפסיקות על המערכת.

טבלת הסיגנלים שנרצה לדגום:

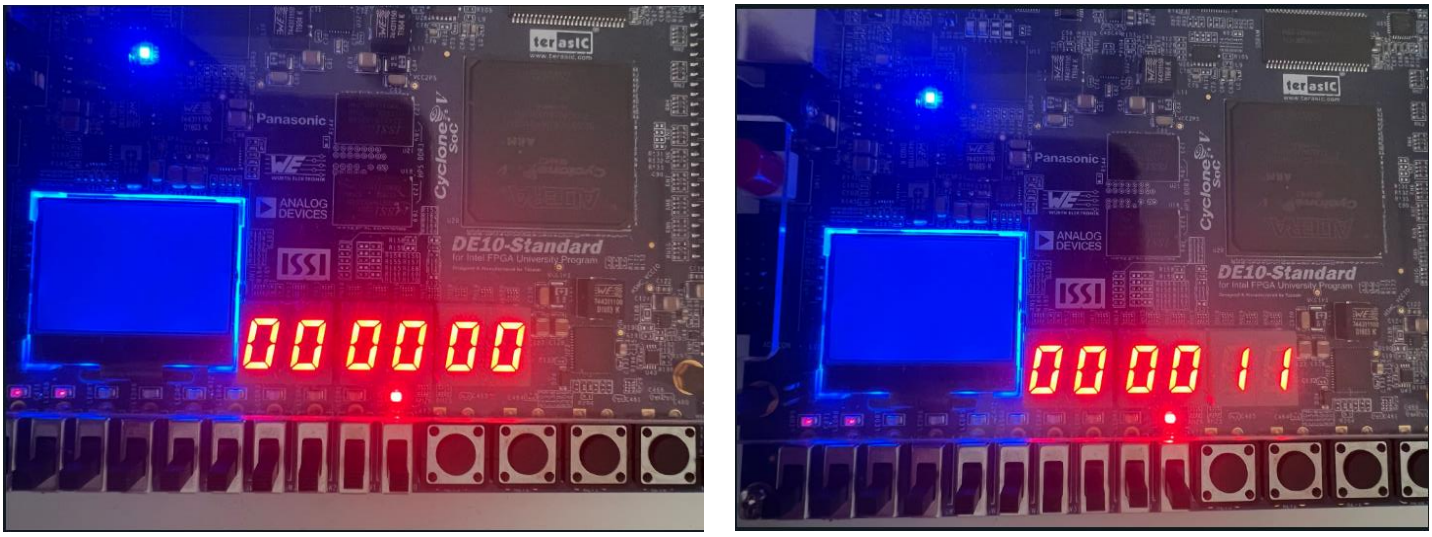
Node			Data Enable	Trigger Enable	Trigger Conditions
Type	Alias	Name	201	201	1 ✓ Basic OR
		reset	✓	✓	↘
		ena	✓	✓	↘
		KEY[0..2]	✓	✓	FFFb (OR)
		KEY[0]	✓	✓	↘
		KEY[1]	✓	✓	↘
		KEY[2]	✓	✓	↘
		SW[0..7] (1)	✓	✓	XXh (OR)
		PC[0..9]	✓	✓	XXXh (OR)
		Instruction[0..31]	✓	✓	XXXXXXXXh (OR)
		IFG[2..5]	✓	✓	Xh (OR)
		IE[2..5]	✓	✓	Xh (OR)
		BTCTL[0..7]	✓	✓	XXh (OR)
		BTCNT[0..31]	✓	✓	XXXXXXXXh (OR)
		BT_OUTMOD	✓	✓	▩
		INT_SRC[0..7]	✓	✓	XXh (OR)
		CS[0..11]	✓	✓	XXXh (OR)
		GIE	✓	✓	▩
		INTR	✓	✓	▩
		INTA	✓	✓	▩
		ISR[0..1]	✓	✓	Xh (OR)
		LEDR[0..7]	✓	✓	XXh (OR)
		HEX0[0..6]	✓	✓	XXh (OR)

איור 30: signal tap signal triggers - MCU

להלן תוצאות דגימה לכידת ערך בHEX לאחר פסיקת לחצן:



איור 31: *KEY1 Press*



איור 32: *FPGA – Before/After Interrupt*

מסקנות

- עבודה לפי סדר פעולות- תכנון של MCU המבוסס על מעבד הsinglecycle ולאחר מכן מעבר לפייפלין. תהליך זה עזר מאוד להבנת אופן הפעולה של המודולים החדשים שנוספו.
- נדרשה הכנה תיאורטית מעמיקה לפני שלב התכנון, בעיקר בנושא של טיפול בפסיקות והממשק בין הCPU לבקר פסיקות. זהו שלב שלא ניתן לדלג עליו מכיוון שהוא מקל מאוד על תהליך התכנון ומאפשר התמודדות פשוטה יותר עם תקלות.
- שינוי אופן עבודת הRESET ומעבר לפסיקת NMI היה חלק חשוב בשלב התכנון שעזר לנו להתמודד עם הצורך באיפוס ערכי הGPO בצורה נכונה.
- נקודה חשובה נוספת שלמדנו במהלך התכנון היא שהפונקציות rising/falling edge מיועדות **לקווי שעון בלבד** ושימוש בהם עבור סיגנלים בתוכנית יוצר בעיה בתפקוד המערכת. התגברנו על הצורך בהם על ידי דגימה של עליה וירידת שעון באמצעות סיגנלי עזר המוקדשים לכך.
- הQUARTUS עובד לפי קריאה של בתים מהזכרון, בעוד ModelSim עובד לפי מילים. הגדרנו זאת באמצעות ביצוע IF GENERATE עבור כל אחת מסביבות העבודה. זוהי נקודה קריטית אשר בזכותה קיבלנו ערכי דגימה זהים בשני המקרים.
- הצלחנו להתגבר על הבעיה בISCME שהייתה קיימת במעבדות קודמות הקשורה לקריאה וכתיבה אל הזכרון בQUARTUS בזמן אמת. יכולת זו עזרה לנו משמעותית בוריפיקציה של המערכת מכיוון שזה חסך לנו את הצורך לבצע קומפלציה כל פעם שרצינו להריץ תוכנית חדשה.