MIPS based MCU Architecture

FINAL PROJECT

MIPS BASED MCU ARCHITECTURE

2024 SEM B

Guy Cohen 207881004

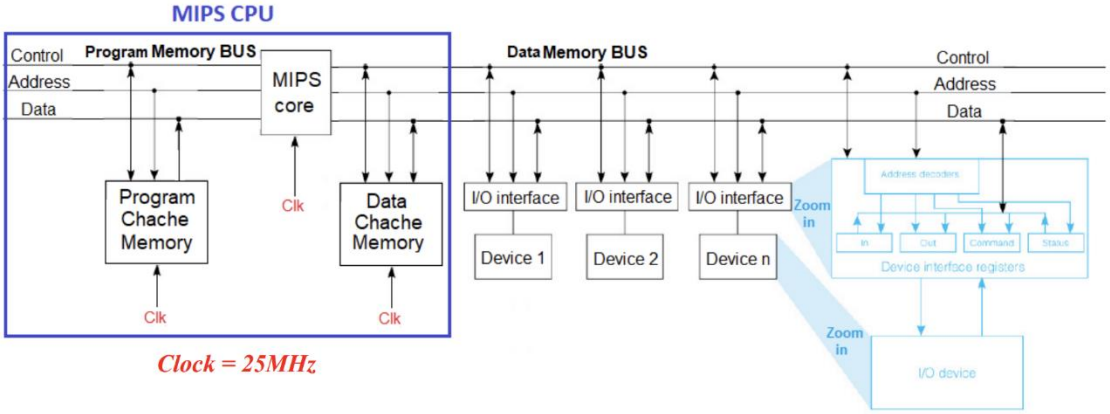
Liav Ben Or 315909390

תוכן עניינים

תיאור המערכת הכללית

בפרויקט זה נרצה לבנות MCU שמכיל מעבד MIPS מסוג Single Cycle המורכב מ5 שלבים – IF, ID, EX, MEM, WB .
התבקשנו להוסיף רכיבי חומרה (Basic Timer, Unsigned Division accelerator, GPIO, Interrupt Controller) נוספות שיעבדו בשיתוף פעולה בניהם בהתאם לצורך. התקשורת בין רכיבי החומרה השונים תהיה באמצעות 3 קווי BUS שיעבירו מידע רלוונטי בין הצורכים לצרכנים.

להלן שרטוט המערכת בדיאגרמת בלוקים –



דיאגרמת הRTL של MCU –

להלן השימוש בקומבינטוריקה הלוגית במודול זה :

Analysis & Synthesis Summary

<<Filter>>

Analysis & Synthesis Status	Successful - Mon ... 2 18:50:13 2024
Quartus Prime Version	21.1.0 Build 842 1...21 SJ Lite Edition
Revision Name	MCU
Top-level Entity Name	MCU
Family	Cyclone V
Logic utilization (in ALMs)	N/A
Total registers	1780
Total pins	65
Total virtual pins	0
Total block memory bits	557,056
Total DSP Blocks	2
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	1
Total DLLs	0

Analysis & Synthesis Resource Usage Summary

<<Filter>>

	Resource	Usage
1	Estimate of Log...n (ALMs needed)	1961
2		
3	▼ Combinational A...usage for logic	2320
1	-- 7 input functions	62
2	-- 6 input functions	1296
3	-- 5 input functions	248
4	-- 4 input functions	366
5	-- <=3 input functions	348
4		
5	Dedicated logic registers	1780
6		
7	I/O pins	65
8	Total MLAB memory bits	0
9	Total block memory bits	557056
10		
11	Total DSP Blocks	2
12		
13	▼ Total PLLs	1
1	-- PLLs	1
14		
15	Maximum fan-out node	PLL:m1|altpll:altp...eneric_pll1_outclk
16	Maximum fan-out	1599
17	Total fan-out	21595
18	Average fan-out	4.98

להלן השימוש בקומבינטוריקה הלוגית עבור כל יתר הרכיבים:

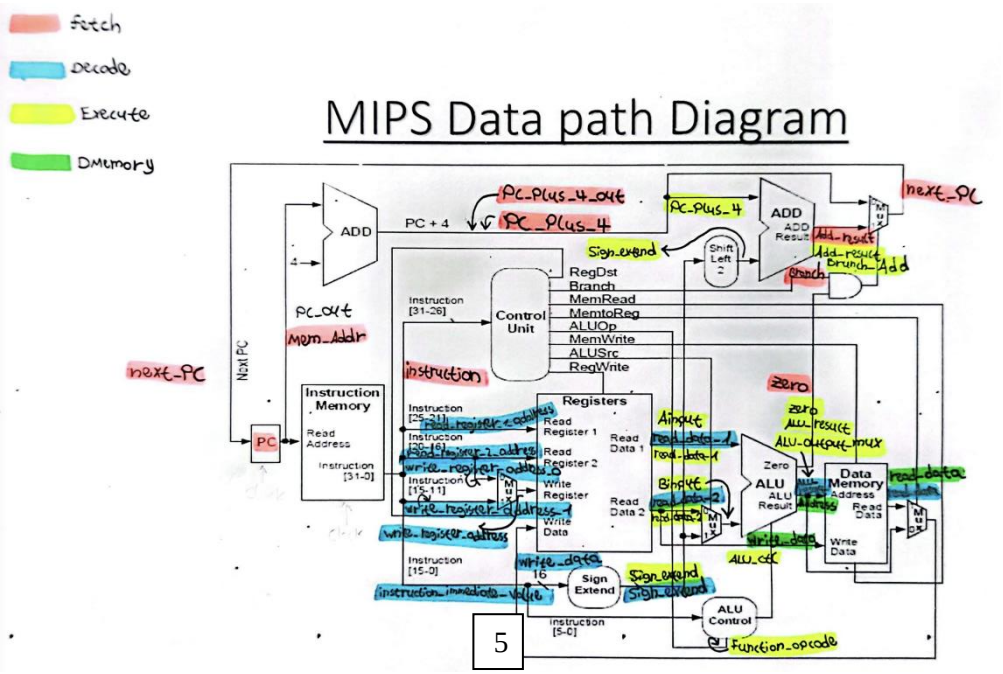
Analysis & Synthesis Resource Utilization by Entity

<<Filter>>

	Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers	Block Mem
1	▼ |MCU	2320 (138)	1780 (200)	557056
1	|BTIMER:Basic_Timer|	75 (75)	102 (102)	0
2	|Divider:div_acc|	113 (113)	105 (105)	0
3	▶ |GPIO:IO_interface|	89 (0)	56 (0)	0
4	|INTERRUPT:Intr_Controller|	132 (132)	14 (14)	0
5	▶ |MIPS:CPU|	1660 (36)	1191 (21)	557056
6	|OptAddrDecoder:OAD|	4 (4)	0 (0)	0
7	▶ |PLL:m1|	0 (0)	0 (0)	0

MIPS

רכיב זה הוא CPU של הMCU, כאשר הוא מסוג Single Cycle. המעבד יודע לתמוך בהוראות המצורפות לתרגיל כך שהוא יודע לבצע את כל התוכניות שמשתמשות אך ורק בהוראות אלו, להלן המעבד עם שמות האותות:



להלן הפעולות:

Arithmetic Instructions			
Instruction	***Example***	***Meaning***	***Comments***
add	add $1,$2,$3	$1=$2+$3	
subtract	sub $1,$2,$3	$1=$2-$3	
add immediate	addi $1,$2,100	$1=$2+100	
Multiply (without overflow)	mul $1,$2,$3	$1=$2*$3	Result is only 32 bits!

Logical Instructions			
Instruction	***Example***	***Meaning***	***Comments***
and	and $1,$2,$3	$1=$2&$3	Bitwise AND
or	or $1,$2,$3	$1=$2|$3	Bitwise OR
xor	xor $1,$2,$3	$1=$2^$3	Bitwise XOR
and immediate	andi $1,$2,100	$1=$2&100	Bitwise AND with immediate value
or immediate	ori $1,$2,100	$1=$2|100	Bitwise OR with immediate value
xor immediate	xori $1,$2,100	$1=$2^100	Bitwise XOR with immediate value
shift left logical	sll $1,$2,10	$1=$2<<10	Shift left by constant number of bits
shift right logical	srl $1,$2,10	$1=$2>>10	Shift right by constant number of bits

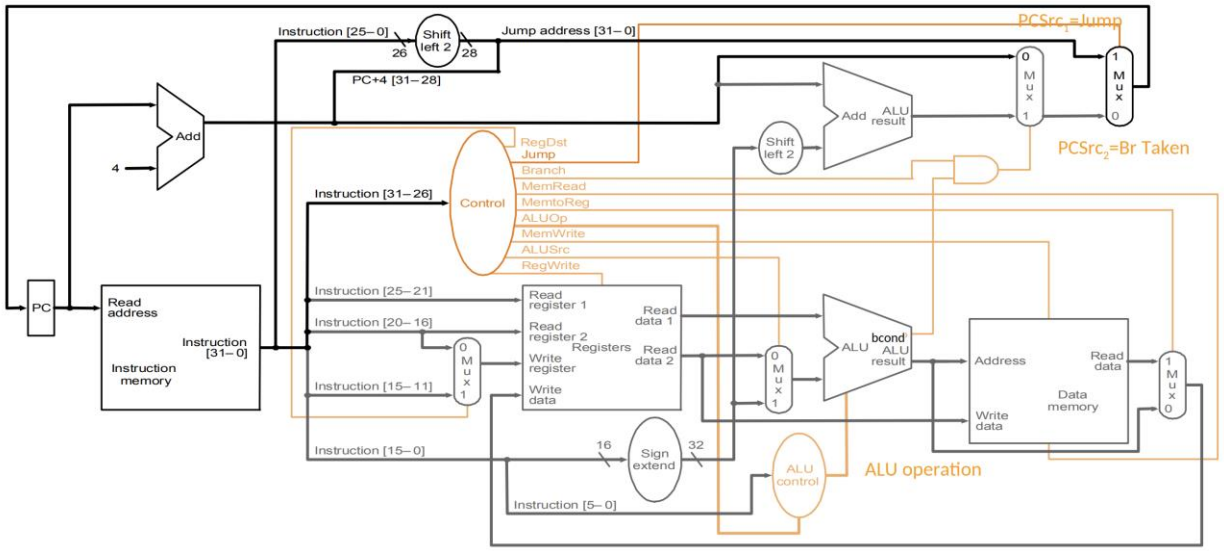
Data Transfer Instructions			
Instruction	***Example***	***Meaning***	***Comments***
move	move $1,$2	$1=$2	*Pseudo-instruction* (provided by *MARS* Assembler, not processor!) Copy from register to register.
load word	Lw $1,100($2)	$1=Memory[$2+100]	
store word	Sw $1,100($2)	Memory[$2+100]=$1	
load upper immediate	lui $1,100	$1=100x$2^{16}$	Load constant into upper 16 bits. Lower 16 bits are set to zero.

Conditional Branch Instructions			
Instruction	***Example***	***Meaning***	***Comments***
branch on equal	`Beq $1,$2,100`	if($1==$2) go to PC+4+100	Test if registers are equal
branch on not equal	`Bne $1,$2,100`	if($1!=$2) go to PC+4+100	Test if registers are not equal

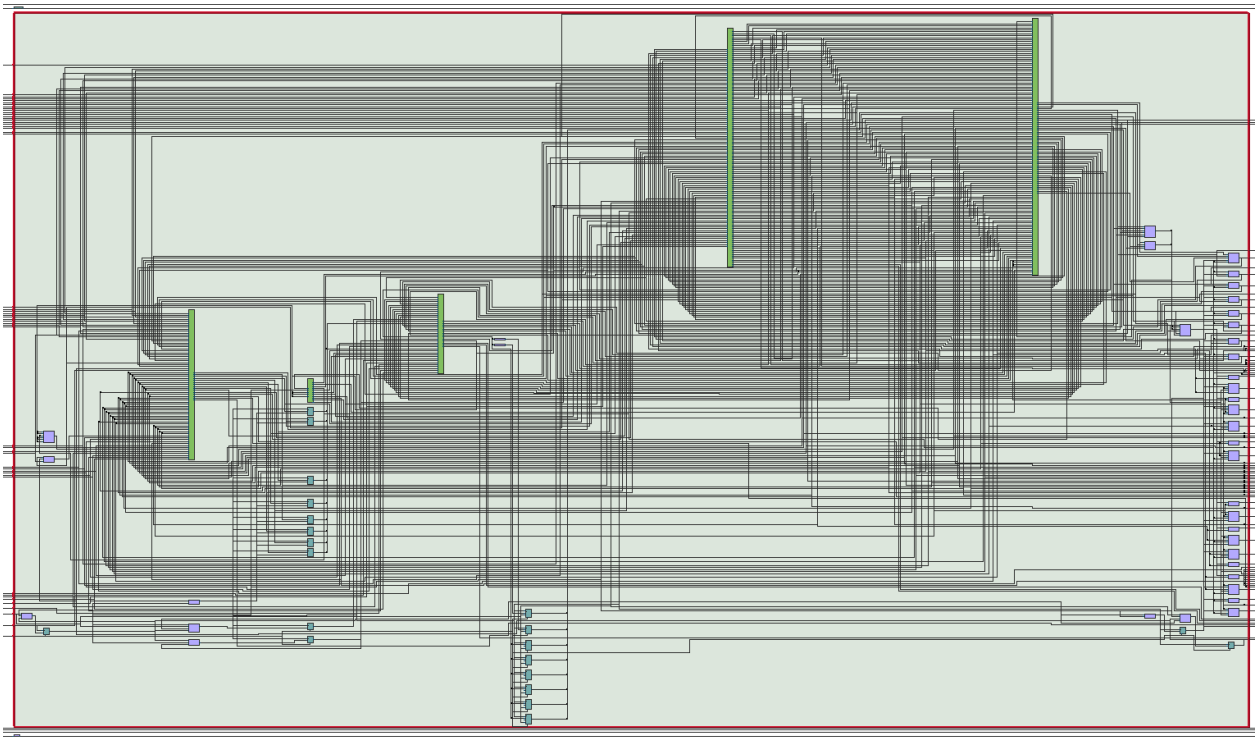
Comparison Instructions			
Instruction	***Example***	***Meaning***	***Comments***
set on less than	`slt $1,$2,$3`	if($2<$3)$1=1; else $1=0	Test if less than. If true, set $1 to 1. Otherwise, set $1 to 0.
set on less than immediate	`Slti $1,$2,100`	if($2<100)$1=1; else $1=0	Test if less than. If true, set $1 to 1. Otherwise, set $1 to 0.

Unconditional Jump Instructions			
Instruction	***Example***	***Meaning***	***Comments***
jump	`j 1000`	go to address 1000	Jump to target address
jump register	`jr $ra`	go to return address stored in $ra	procedure return
jump and link	`jal 1000`	$ra=PC+4; go to procedure call which starts in address 1000	Use when making procedure call. This saves the return address in $ra

להלן שרטוט המעבד –



להלן דיאגרמת הRTL כללית במודול זה:



MIPS:CPU

	ALU_result[1]~29
	ALU_result[2]~53
	ALU_result[3]~57
	ALU_result[4]~21
	ALU_result[5]~25
	ALU_result[6]~42
	ALU_result[7]~46
	ALU_result[8]~49
	ALU_result[9]~32
	ALU_result[10]~35
	ALU_result[11]~38
	AddressBus[0]~0
	AddressBus[1]~3
	AddressBus[2]~6
	AddressBus[3]~7
	AddressBus[4]~1
	AddressBus[5]~2
	AddressBus[9]~4
	AddressBus[10]~5
BTIMER:Basic_Timer:Equal4~0	AddressBus~8
BTIMER:Basic_Timer:Equal4~2	ControlBus[0]~6
DataBus[0]~22	ControlBus[1]~5
DataBus[1]~28	ControlBus[2]~4
DataBus[2]~35	ControlBus[3]~1
DataBus[3]~41	ControlBus[4]~2
DataBus[4]~52	ControlBus[5]~3
DataBus[5]~58	ControlBus[6]~0
DataBus[6]~64	DataBus[0]~1
DataBus[7]~75	DataBus[1]~2
INTERRUPT:Intr_Controller:DataBus[8]~85	DataBus[2]~3
INTERRUPT:Intr_Controller:DataBus[9]~82	DataBus[3]~4
INTERRUPT:Intr_Controller:DataBus[10]~79	DataBus[4]~5
INTERRUPT:Intr_Controller:DataBus[11]~76	DataBus[5]~6
INTERRUPT:Intr_Controller:DataBus[12]~73	DataBus[6]~7
INTERRUPT:Intr_Controller:DataBus[13]~70	DataBus[7]~8
INTERRUPT:Intr_Controller:DataBus[14]~67	DataBus~0
INTERRUPT:Intr_Controller:DataBus[15]~52	Equal1~0
INTERRUPT:Intr_Controller:DataBus[16]~55	Equal5~[1..2]
INTERRUPT:Intr_Controller:DataBus[17]~58	INTA_sig
INTERRUPT:Intr_Controller:DataBus[18]~61	INTR_STATE[1]
INTERRUPT:Intr_Controller:DataBus[19]~64	Idecode:ID:Equal3~1
INTERRUPT:Intr_Controller:DataBus[20]~46	Idecode:ID:Equal4~1
INTERRUPT:Intr_Controller:DataBus[21]~49	Idecode:ID:Equal5~0
INTERRUPT:Intr_Controller:DataBus[22]~103	Idecode:ID:Equal5~3
INTERRUPT:Intr_Controller:DataBus[23]~106	Idecode:ID:Equal6~0
INTERRUPT:Intr_Controller:DataBus[24]~88	Idecode:ID:Equal7~1
INTERRUPT:Intr_Controller:DataBus[25]~91	Idecode:ID:write_data~0
INTERRUPT:Intr_Controller:DataBus[26]~94	Mux32~11
INTERRUPT:Intr_Controller:DataBus[27]~97	Mux33~11
INTERRUPT:Intr_Controller:DataBus[28]~100	Mux34~11
INTERRUPT:Intr_Controller:DataBus[29]~112	Mux35~12
INTERRUPT:Intr_Controller:DataBus[30]~115	Mux36~11
INTERRUPT:Intr_Controller:DataBus[31]~109	Mux37~11
INTERRUPT:Intr_Controller:INTR	Mux38~11
IN62	Mux39~11
OptAddrDecoder:OAD:CS_LEDR~0	Mux40~11
a_internal_jtag:altera_internal_jtag~TCKUTAP	Mux41~11
a_internal_jtag:altera_internal_jtag~TDIUTAP	Mux42~10
clr_reg	Mux43~10
ena~input	Mux44~10
irf_reg[1][0..5]	Mux45~10
irf_reg[2][0..5]	Mux46~10
reset~inputCLKENA0	Mux47~10
reset~input	Mux48~10
splitter_nodes_receive_0[3]	Mux49~10
splitter_nodes_receive_1[3]	Mux50~10
state[3..5]	Mux51~10
state[8]	Mux52~11
virtual_ir_scan_reg	Mux53~10
~GND	Mux54~10
	Mux55~10
	OUT[49..50]
	OUT[53..54]
	OUT30
	PC[2..9]
	altsyncram_dh33:altsyncram1:q_a[29]
	register_array[26][0]
	sld_mod_ram_rom:mgl_prim2:adapted_tdo~0
	sld_mod_ram_rom:mgl_prim2:adapted_tdo~0
	sld_mod_ram_rom:mgl_prim2:ir_loaded_address_reg[0..1]
	sld_mod_ram_rom:mgl_prim2:ir_loaded_address_reg[0]
	sld_mod_ram_rom:mgl_prim2:ir_loaded_address_reg[1..2]
	sld_mod_ram_rom:mgl_prim2:ir_loaded_address_reg[2..3]
	sld_mod_ram_rom:mgl_prim2:ir_loaded_address_reg[3..4]
	sld_mod_ram_rom:mgl_prim2:ir_loaded_address_reg[4..5]
	sld_mod_ram_rom:mgl_prim2:ir_loaded_address_reg[5..6]
	sld_mod_ram_rom:mgl_prim2:ir_loaded_address_reg[6]
	sld_mod_ram_rom:mgl_prim2:is_in_use_reg
	sld_mod_ram_rom:mgl_prim2:is_in_use_reg

Instruction Fetch

בשלב זה נרצה להביא את הפקודה בכתובת הPC מהזיכרון של ההוראות. כמו כן, בחלק זה ישב הטיפול בPC הנבחר, מפני שבארכיטקטורה המצורפת קיימים תמיד מספר אופציות לשינוי הPC –

- נרצה להכניס לPC ערך של כתובת אליה נרצה לבצע jump
- נרצה להכניס לPC ערך של כתובת אליה נרצה לבצע branch
- $PC \leftarrow PC + 4$ כאשר נרצה לקדם את הפקודה הבאה

```
Next_PC   <= X"00" WHEN Reset = '1' ELSE
             ISRAddr(9 DOWNTO 2) WHEN Read_ISR_PC = '1'  ELSE     -- Interrupt!
             jump_address WHEN ((jump = '1' or jump_register = '1') and Read_ISR_PC = '0') ELSE --jump\jal
             Add_result  WHEN ( ( Branch = '1' ) AND ( Zero = '1' ) ) ELSE   --beq
             Add_result  WHEN ( ( Branch_not_equal = '1' ) AND ( Zero = '0' ) ) ELSE  --bne
             PC_plus_4( 9 DOWNTO 2 );
```

Instruction Decode

בשלב זה נרצה לקחת את הפקודה שהוראה משלב הfetch ולהמיר אותה לאחד מסוגי הפקודות שלהלן:

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)				

כך נכניס את כתובת הרגיסטרים המתאימים לRF שנמצא בשלב הdecode ונוכל להוציא את המידע מהרגיסטרים הדרושים.

Execute

בשלב זה מתבצע השלב החישובי של המעבד, עבור הוראות מסוג Rtype, הALU לוקח את הנתונים משני רגיסטרים מבוקשים ומעביר את התוצאה לקו הבקרה Alu_result. עבור פקודות Itype הALU מבצע פכולה אריתמטית עבור נתונים מרגיסטר אחד ביחד עם קבוע המאוחסן ב instruction.

Data Memory

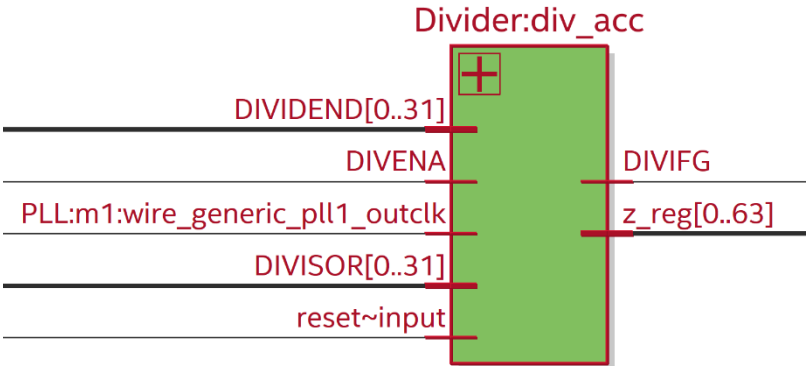
מודול זה אחראי על הכתיבה והקריאה מתוכן הData Memory שזה זיכרון הRAM של המערכת שלנו. אנחנו במשימה השתמשנו בגודל RAM של 2^{12}.

Write Back

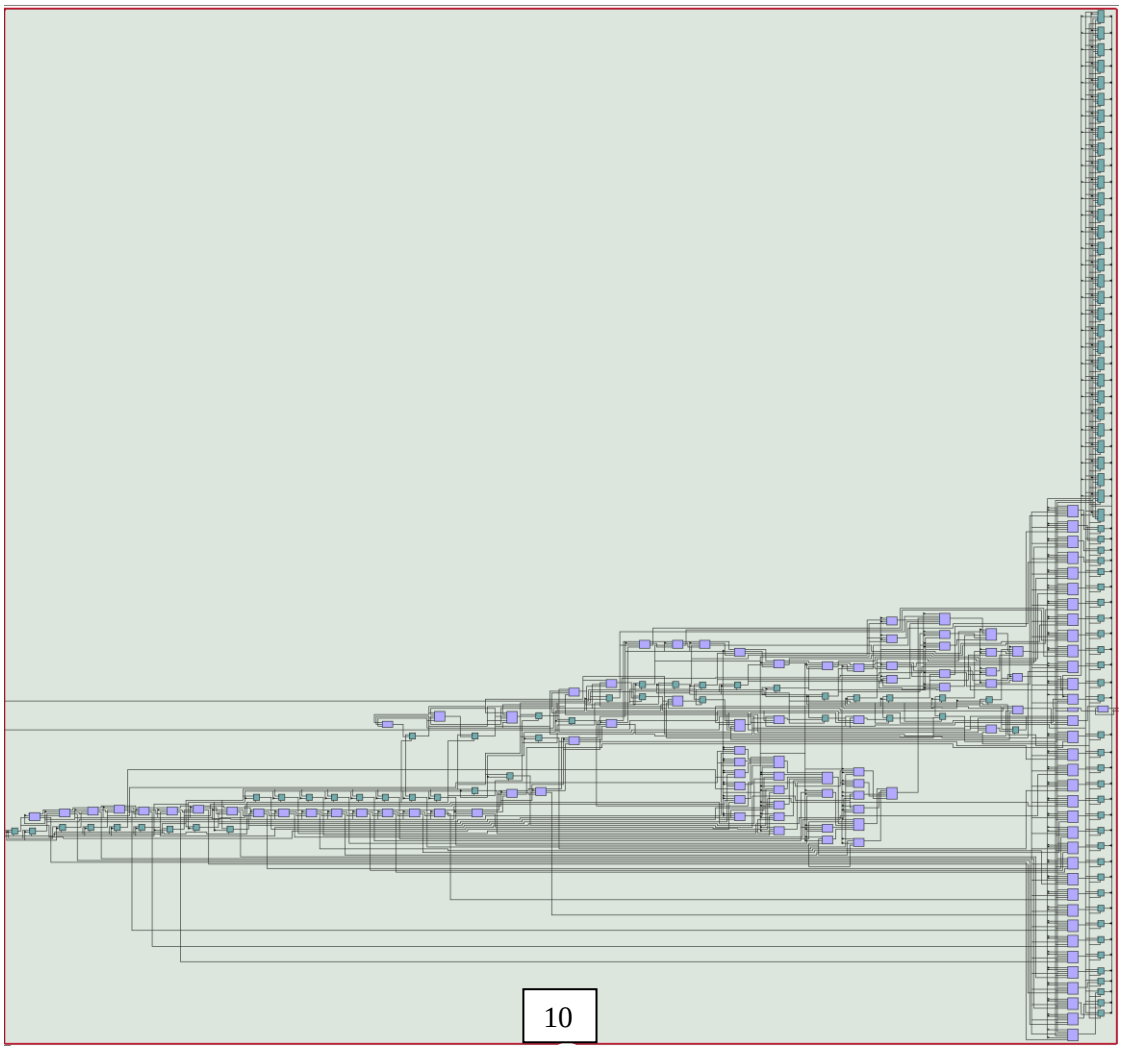
שלב זה מוכל במימוש שלי בתוך הDecode, אשר מקבל את המידע מהALU עבור פעולות Rtype ומה-RAM עבור פעולת LW.

Unsigned Binary Division Multicycle Accelerator

רכיב חומרה זה אחראי על חלוקת מספר המכיל 32 ביטים במספר בעל 32 ביטים, בסיום הפעולה רכיב החומרה טוען לזכרון את תוצאת החלוקה כמספר שלם ואת השארית במקומות המתאימים בזכרון. לאחר 32 מחזורי שעון, רכיב החומרה מעלה דגל DIVIFG אשר מודיע לבקר הפסיקות כי הוא סיים את פעולתו ושיש לבצע את רוטינת הפסיקה בהקדם האפשרי. מימשנו את הרכיב כך שהוא יעבוד רק כאשר הוכנס מספר מחלק אשר שונה מ0 כדי להימנע מexceptions.



RTL:



Basic Timer

תפקידי רכיב חומרה זה:

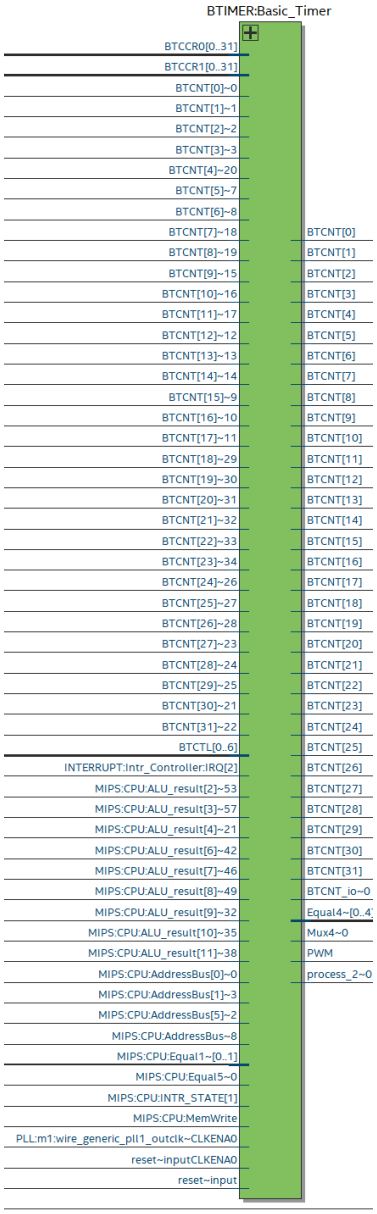
1) רכיב חומרה זה מכיל בין היתר רגיסטר בקרה BTCTL הקובע את תכונות השעון ורגיסטר BTCNT שהוא מייצג את ערך הספירה הנוכחי של הטיימר. כמו כן, קיים ברגיסטר הבקרה ערך BTIP שלפיו הרכיב מעלה דגל כאשר ערך רגיסטר הספירה מגיע לערך מתאים. ניתן להשתמש בדגל זה כדי להפריע לתוכנית הראשית של המיקרו בקר ולבצע ISR מתאים.

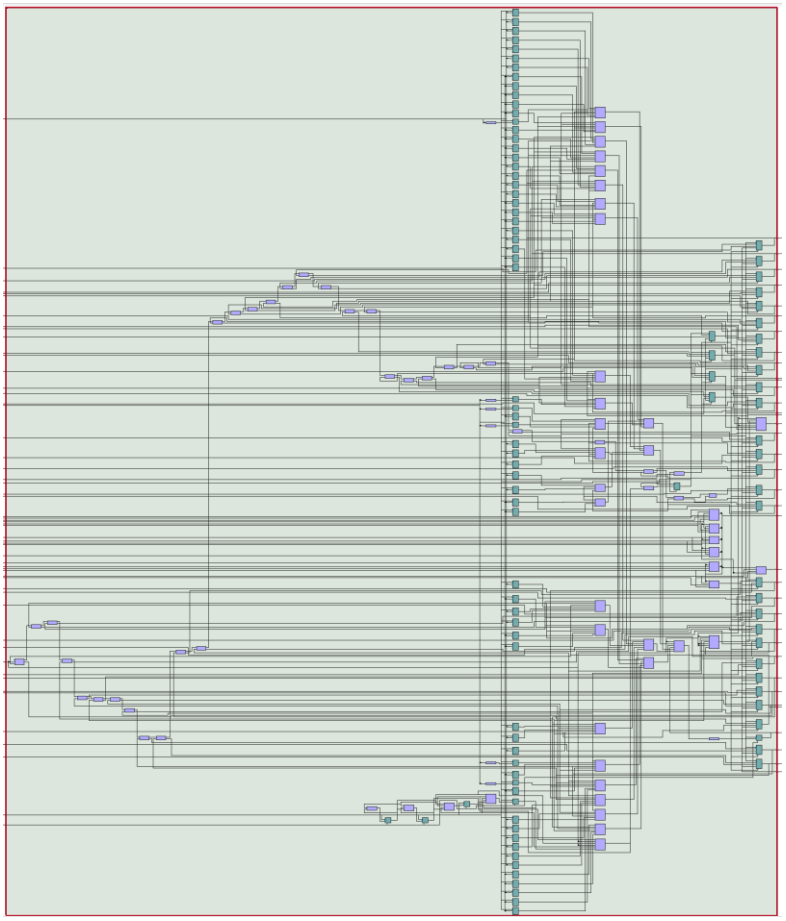
BTCTL, Basic Timer Control Register

7	6	5	4	3	2	1	0
BTOUTMD	BTOUTEN	BTHOLD	BTSSEL		BTIPx		
rw	rw	rw	rw	rw	rw	rw	rw

2) ייצור אות PWM באמצעות הטיימר הבסיסי הזה כך שהטיימר סופר ובאמצעות CCRx ניתן לשלוט על הזמן מחזור של האות ועל הDuty Cycle שלו באמצעות רגיסטרים CCR0 וCCR1 בהתאמה.

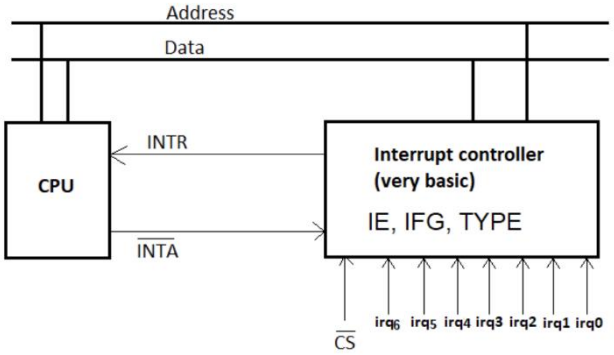
דיאגרמת הRTL כללית במודול זה:

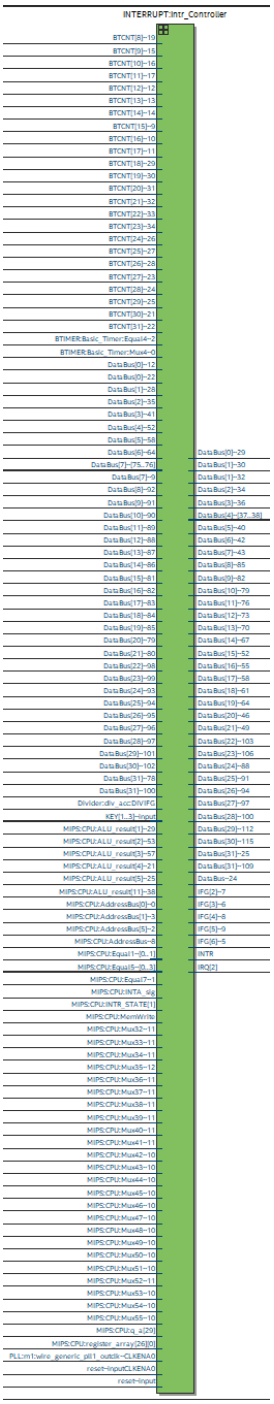




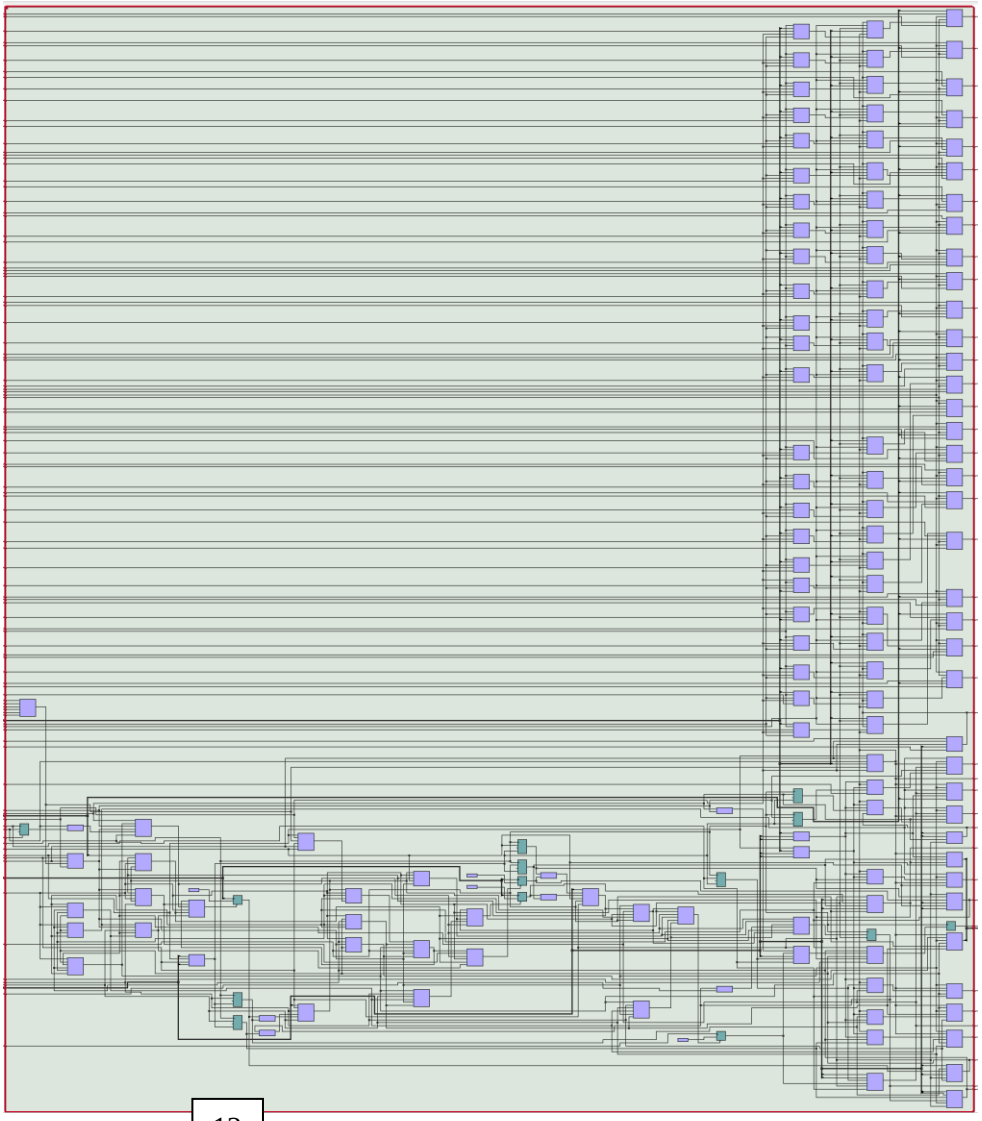
Interrupt Controller

פסיקה מתבצעת כאשר אחד מרכיבי החומרה השונים מבקשים לבצע פסיקה אזי הם מעלים סיגנל בשם IRQ שמבקש לבצע פסיקה, signal בשם intrsrc מכיל את המידע על הרכיבים המבקשים לקפוץ לרוטינת הפסיקה שלהם.
פסיקה מתרחשת כאשר קיים אפשור גלובאלי לפסיקות (GIE) וגם אין פסיקה מעדיפות גבוהה יותר לפניה.





דיאגרמת הRTL של רכיב זה:



GPIO

ממש אשר אחראי על התמיכה ברכיבי IO השונים שנמצאים על גבי שבב הFPGA, רכיבי הIO הם למעשה Memory Mapped IO, להלן הכתובות של הרכיבים בזכרון:

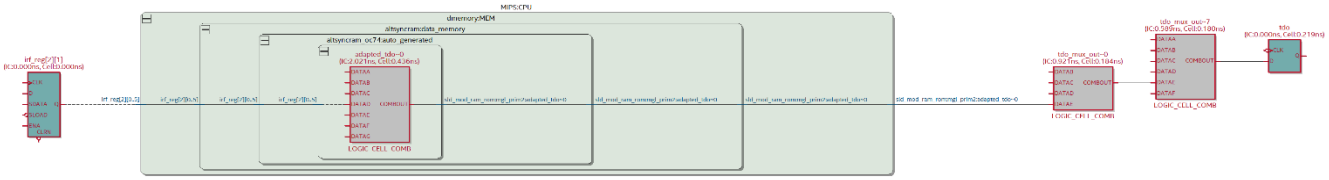
GPIO ***without*** *interrupt capability*

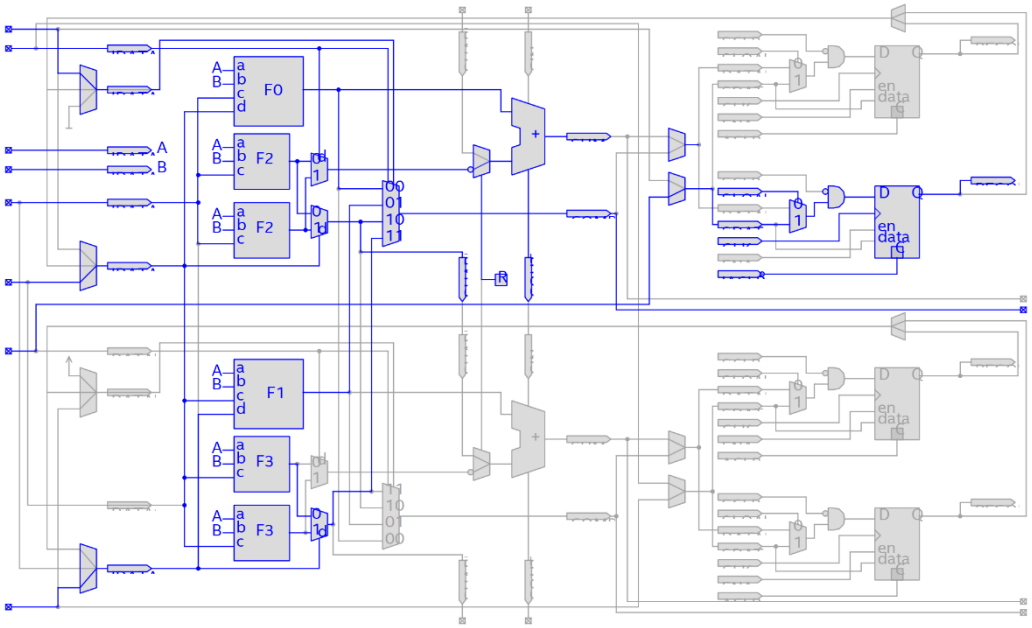
```
#-----------------------------------------------------------------
#           MEMORY Mapped I/O
#-----------------------------------------------------------------
#define PORT_LEDR[7-0] 0x800 - LSB byte (Output Mode)
#-------------------- PORT_HEX0_HEX1 ----------------------------
#define PORT_HEX0[7-0] 0x804 - LSB byte (Output Mode)
#define PORT_HEX1[7-0] 0x805 - LSB byte (Output Mode)
#-------------------- PORT_HEX2_HEX3 ----------------------------
#define PORT_HEX2[7-0] 0x808 - LSB byte (Output Mode)
#define PORT_HEX3[7-0] 0x809 - LSB byte (Output Mode)
#-------------------- PORT_HEX4_HEX5 ----------------------------
#define PORT_HEX4[7-0] 0x80C - LSB byte (Output Mode)
#define PORT_HEX5[7-0] 0x80D - LSB byte (Output Mode)
#-----------------------------------------------------------------
#define PORT_SW[7-0]   0x810 - LSB byte (Input Mode)
#-----------------------------------------------------------------
#define PORT_KEY[3-1]  0x814 - LSB nibble (3 push-buttons - Input Mode)
#-----------------------------------------------------------------
```

למעשה, בעת לחיצה על אחד הלחצנים, רכיב חומרה זה אחראי ע"י קפיצה לרוטינת פסיקה של הלחצנים להעביר מידע לאחד מרכיבים אלו או קבלת מידע מרכיבי קלט כגון מתגים.

נתיב קריטי

הנתיב הקריטי הוא הנתיב הארוך ביותר של התפשטות הלוגיקה בתוך המעגל הדיגיטלי שבנינו. הוא מייצג את הנתיב האיטי ביותר במערכת, כך שנצטרך לקבוע לפיו את תדר השעון המירבי שניתן להשיג עבור תכנון הMCU.
הנתיב הקריטי בפרויקט זה:





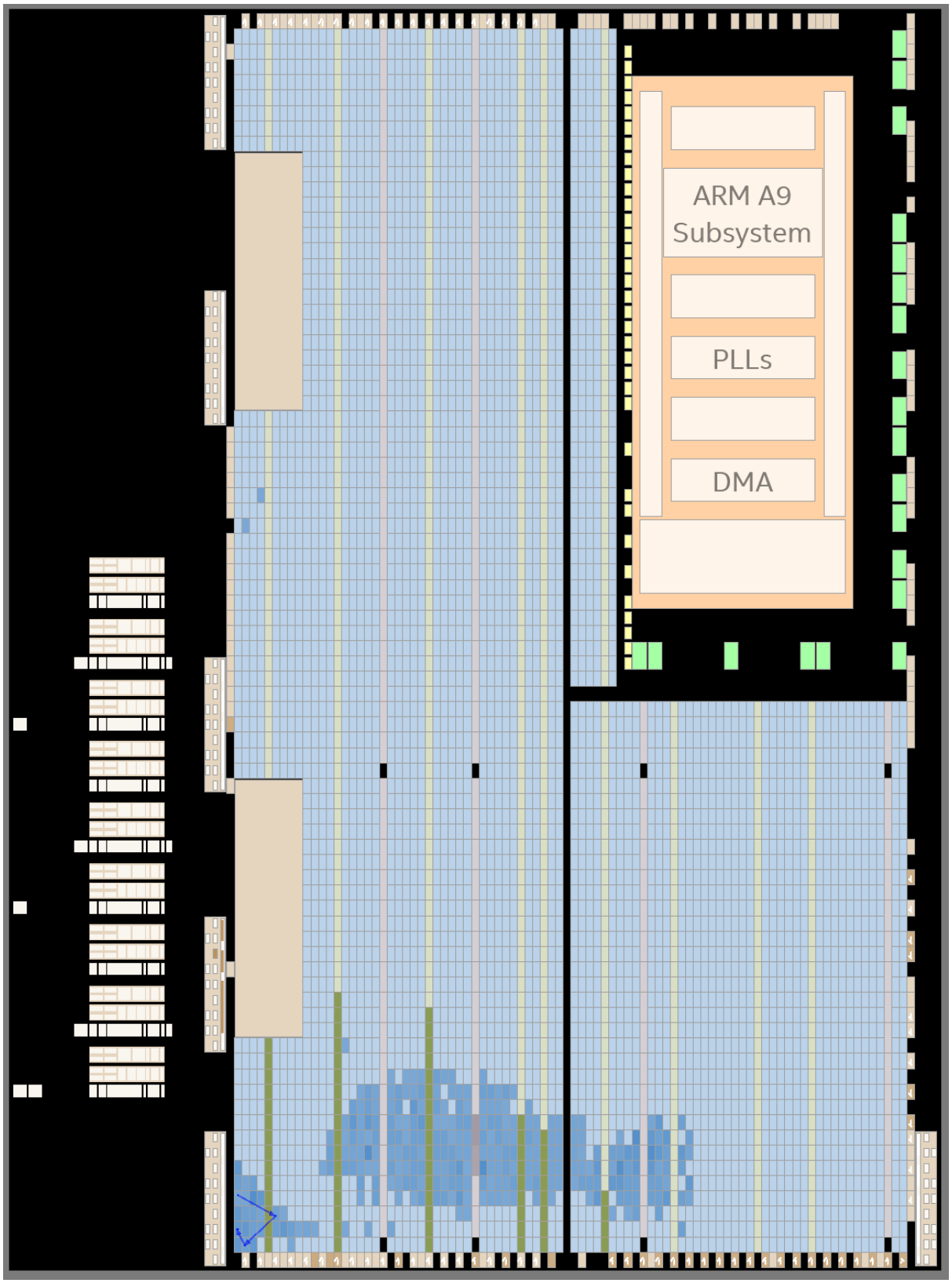
בנוסף על כך, מצאנו **את תדר השעון המקסימלי** עבור המודל שבנינו והוא:

Slow 1100mV 0C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name	Note
1	91.92 MHz	91.92 MHz	alter...d_tck	

הנתיב הקריטי היכן שממוקם על גבי השבב הינו:



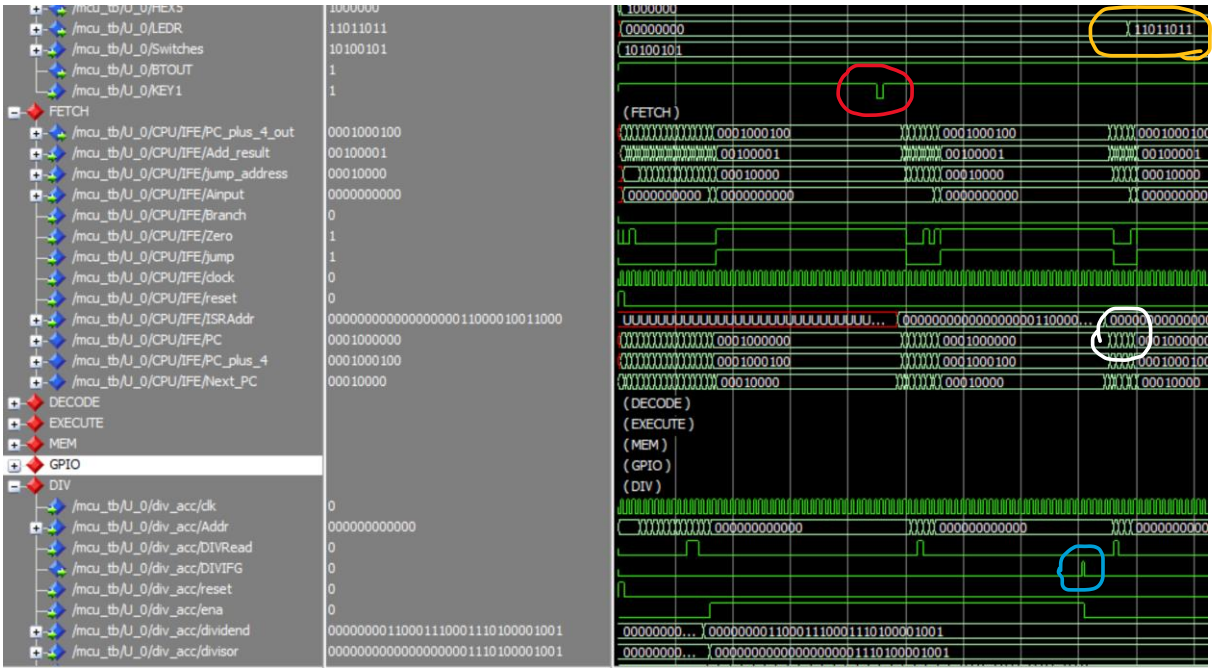
ניתוח תוצאות

בפרק זה נדון בתוצאות המערכת שלנו גם מבחינת ניתוח הזמנים, גם מבחינת סימולציות ותיעוד הגלים בFPGA באמצעות Signal Tap.

ניתוח גלים בModel SIM

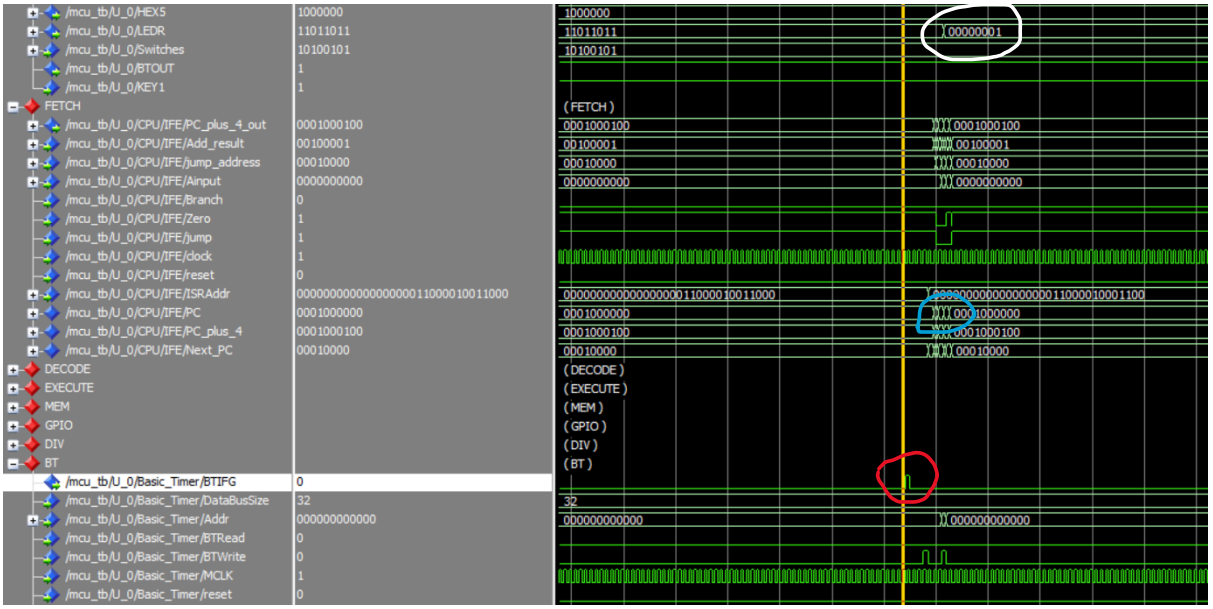
בחלק זה אציג את פעולת הרכיבים כפי שנראים בסימולציית הגלים:

להלן הדוגמא הראשונה:



- בסימון האדום ניתן לראות לחיצה על לחצן מספר 1.
- לחצן אחד טוען שני מספרים למחלק הבינארי (מספר מחלק ומספר מחולק), לאחר 32 מחזורי שעון ניתן לראות בסימון הכחול כי רכיב החומרה מעלה דגל המסמל שהתוצאות זמינות כעת.
- בסימון הלבן ניתן לראות את השינוי בPC כתוצאה מקפיצה לISR של המחלק.
- בסימון הצהוב ניתן לראות את תוצאת החלוקה נכתבת לLEDS.

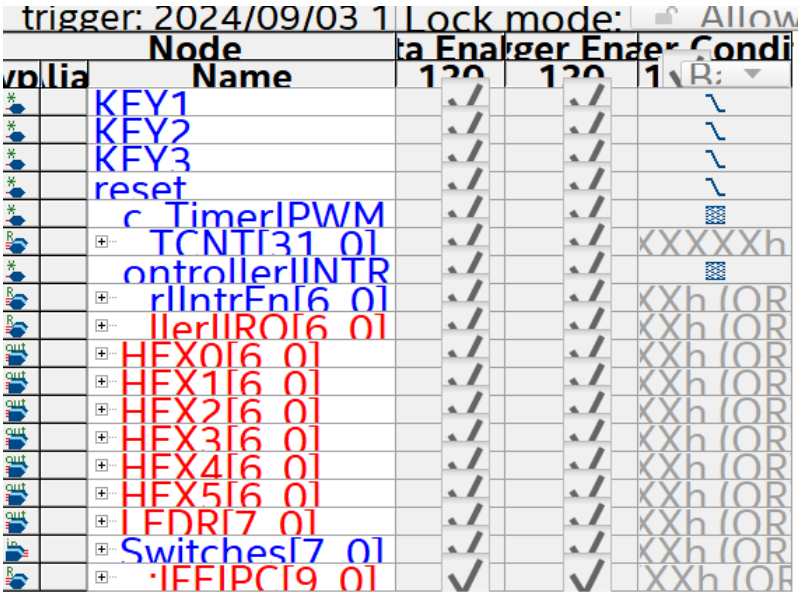
להלן הדוגמא השנייה:



- בסימון האדום ניתן לראות בקשת ה BT לפסיקה בעקבות ערך המנייה אשר הגיעה לערך המתאים לפי BTIP .
- בסימון הכחול ניתן לראות את רגיסטר הPC משתנה לכתובת הISR של השעון החומרתי.
- בסימון הלבן ניתן לראות את ערך הלדים מעודכן לפי ערך המנייה (בבדיקה זו הקוד מבצע מנייה כלפי מעלה על גבי הלדים ע"י פסיקת השעון החומרתי).

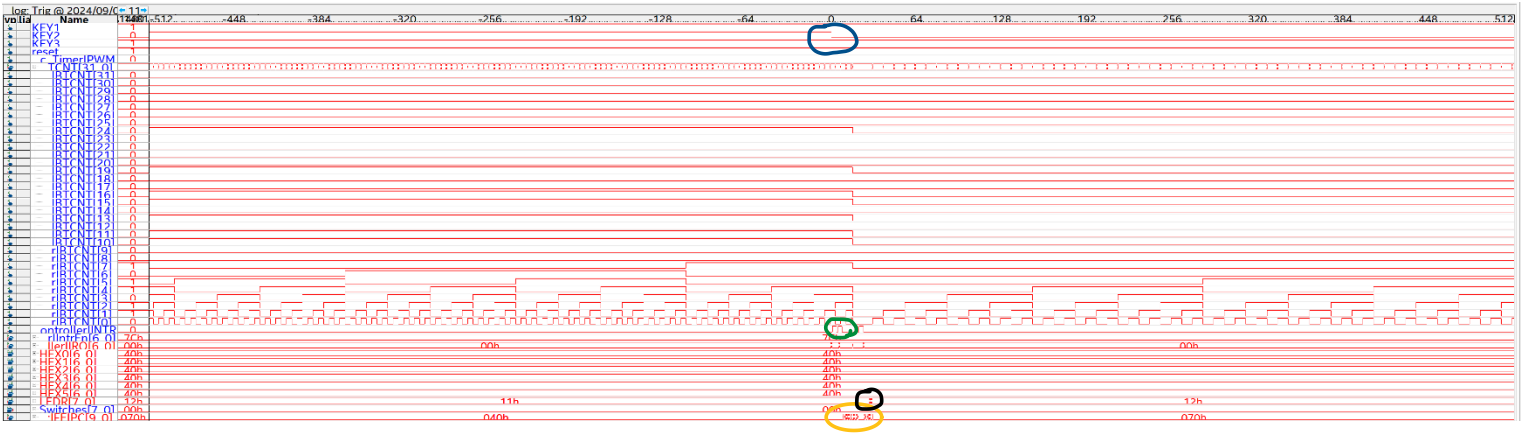
ניתוח גלים בSignal Tap

ראשית אציג את רשימת האותות אשר בחרתי לבחון תחת בדיקה זו:



נשים לב כי ארבעת הלחצנים מקבלים טריגר ב falling edge ולכן בחרתי באפשרות זו, בנוסף האותות תחת קטגוריה Basic Or.

הקוד הצרוב על הבקר בעת הבדיקה מבצע מנייה כלפי מעלה ע"י פסיקת שעון, להלן תוצאות **הדוגמא הראשונה**:



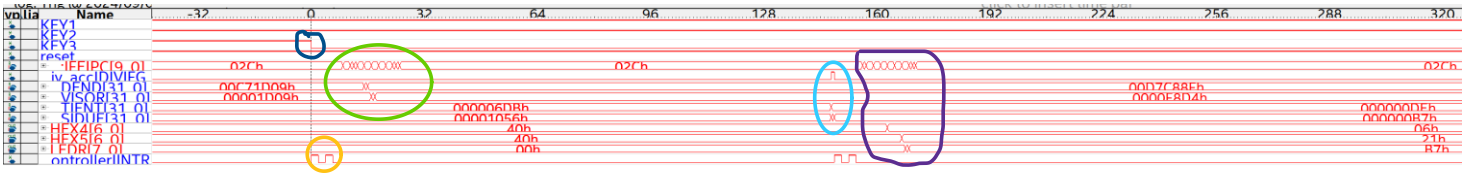
- בסימון הכחול ניתן לראות את הטריגר של לחיצה על לחצן מספר 2.
- בסימון הירוק ניתן לראות את בקשת הפסיקה מועברת לרכיב הint controller על מנת לבצע רוטינת פסיקה של הלחצן.
- בסימון הצהוב ניתן לראות את ביצוע רוטינות השירות של הלחצן והשעון.
- בסימון השחור ניתן לראות לאחר פסיקת השעון את ערך המניה על גבי הלדים עולה ב1.

דוגמא שנייה:
הקוד הצרוב על גבי הבקר מבצע את הדבר הבא:

```
On KEY3 pushing:
---------------
loading DIVIDEND with 8-bit value of y=0x00D7C88F=14,141,583
loading DIVISOR with 8-bit value of x=0x0000F8D4=63,700
Note: y/x={QUOTIENT=0xDE=222 ,RESIDUE=0xB7=183}

On Divider Accelerator interrupt:
--------------------------------
write QUOTIENT low nibble to PORT_HEX4[7-0]
write QUOTIENT high nibble to PORT_HEX5[7-0]
write RESIDUE to PORT_LEDR[7-0]
```

תוצאות הסימולציה:



- בסימון הכחול ניתן לראות לחיצה על לחצן מספר 3.
- בסימון הצהוב מועברת ל int controller בקשת פסיקה מהלחצן.
- בסימון הירוק ניתן לראות ביצוע רוטינת השירות של לחצן 3, במהלכה נטענים ערכי המחלק והמחולק עם הערכים שציינתי מעלה.
- בסימון התכלת, מאיץ החומרה מסיים את פעולת החילוק לאחר 32 מחזורי שעון ומעלה דגל בקשת פסיקה המועבר לint controller.
- בסימון הסגול, רואים קפיצה לרוטינת השירות של מאיץ החומרה, במהלכה תוצאות החילוק נטענים לרכיבי ה IO המתאימים, ניתן לראות שתוצאות החישוב תואמות למצופה כפי שכתבתי מעלה.