ADVANCED CPU ARCHITECTURE AND HARDWARE ACCELERATORS LAB

FINAL PROJECT

MIPS BASED MCU ARCHITECTURE

361.1.4693

2023 SEM B

Michael Grenader 208839845

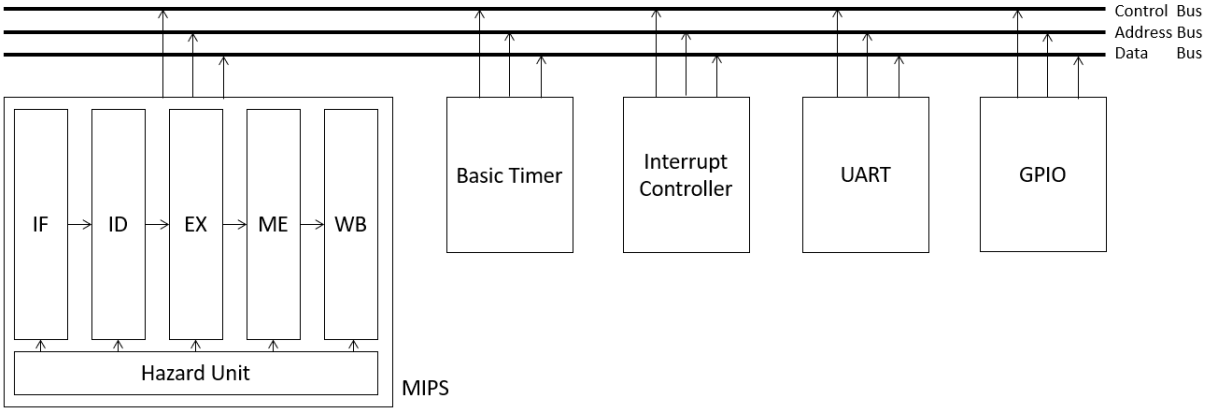
Roy Kislev 206917064

תוכן עניינים

המערכת

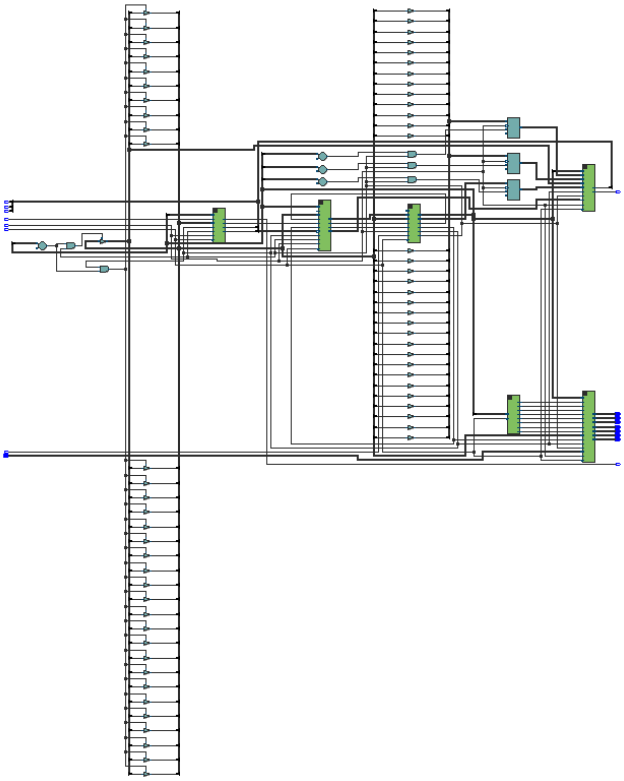
בפרויקט זה נרצה לבנות MCU שמכיל מעבד MIPS מסוג Pipeline בעל 5 דרגות – IF, ID, EX, MEM, WB התומך בטיפול וזיהוי בעיות באמצעות רכיב Hazard detection. יחד עם המעבד נרצה להוסיף פריפריות חומרה נוספות שיעבדו בשיתוף פעולה בניהם בהתאם לצורך. התקשורת בין רכיבי החומרה השונים תהיה באמצעות 3 קווי BUS שיעבירו מידע רלוונטי בין הצורכים לצרכנים.

להלן שרטוט המערכת בדיאגרמת בלוקים –



איור 1 שרטוט מערכת הMCU

להלן דיאגרמת הRTL כללית במודול זה –



איור 2הRTL של הMCU

להלן השימוש בקומבינטוריקה הלוגית במודול זה –

Analysis & Synthesis Resource Usage Summary

<<Filter>>

	Resource	Usage
1	Estimate of Logic utilization (ALMs needed)	1860
2		
3	Combinational ALUT usage for logic	2234
1	-- 7 input functions	9
2	-- 6 input functions	1242
3	-- 5 input functions	365
4	-- 4 input functions	189
5	-- <=3 input functions	429
4		
5	Dedicated logic registers	1715
6		
7	I/O pins	67
8	Total MLAB memory bits	0
9	Total block memory bits	65536
10		
11	Total DSP Blocks	2
12		
13	Maximum fan-out node	clock~input
14	Maximum fan-out	1741
15	Total fan-out	18593
16	Average fan-out	4.48

Analysis & Synthesis Summary

<<Filter>>

Analysis & Synthesis Status	Successful - Mon Aug 7 19:15:15 2023
Quartus Prime Version	21.1.0 Build 842 10/21/2021 SJ Lite Edition
Revision Name	FinalProjetCPU
Top-level Entity Name	MCU
Family	Cyclone V
Logic utilization (in ALMs)	N/A
Total registers	1715
Total pins	67
Total virtual pins	0
Total block memory bits	65,536
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	0
Total DLLs	0

להלן השימוש בקומבינטוריקה הלוגית עבור כל יתר המודולים המצורפים במהלך הדוח –

Analysis & Synthesis Resource Utilization by Entity

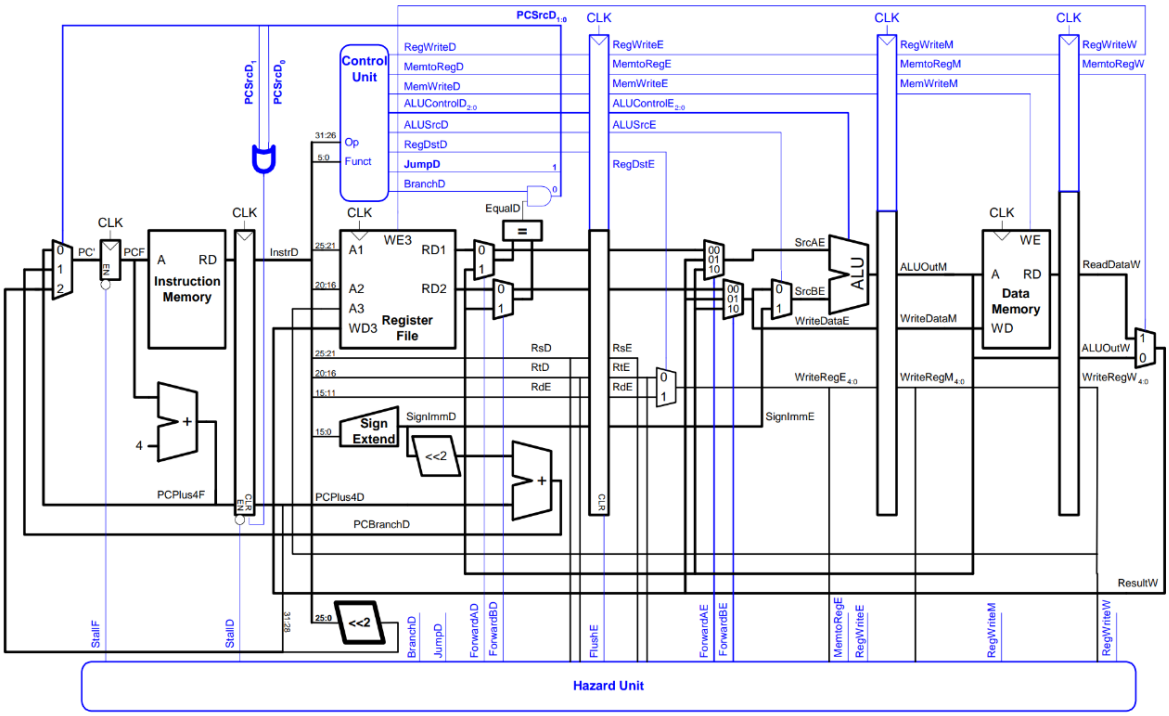
<<Filter>>

	Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers	Block Memory Bits	DSP Blocks	Pins	Virtual Pins	
1	|MCU	2234 (91)	1715 (71)	65536	2	67	0	|MCU
1	|BTIMER:Basic_Timer|	75 (75)	101 (101)	0	0	0	0	|MCU|BTIMER:Basic_Timer
2	|GPIO:IO_interface|	69 (0)	56 (0)	0	0	0	0	|MCU|GPIO:IO_interface
3	|INTERRUPT:Intr_Controller|	113 (113)	15 (15)	0	0	0	0	|MCU|INTERRUPT:Intr_Controlle
4	|MIPS:CPU|	1686 (235)	1337 (337)	65536	2	0	0	|MCU|MIPS:CPU
5	|OptAddrDecoder:OAD|	8 (8)	0 (0)	0	0	0	0	|MCU|OptAddrDecoder:OAD
6	|USART:UsartSupport|	192 (36)	135 (26)	0	0	0	0	|MCU|USART:UsartSupport

איור 3 שימוש בלוגיקה בכל המערכת

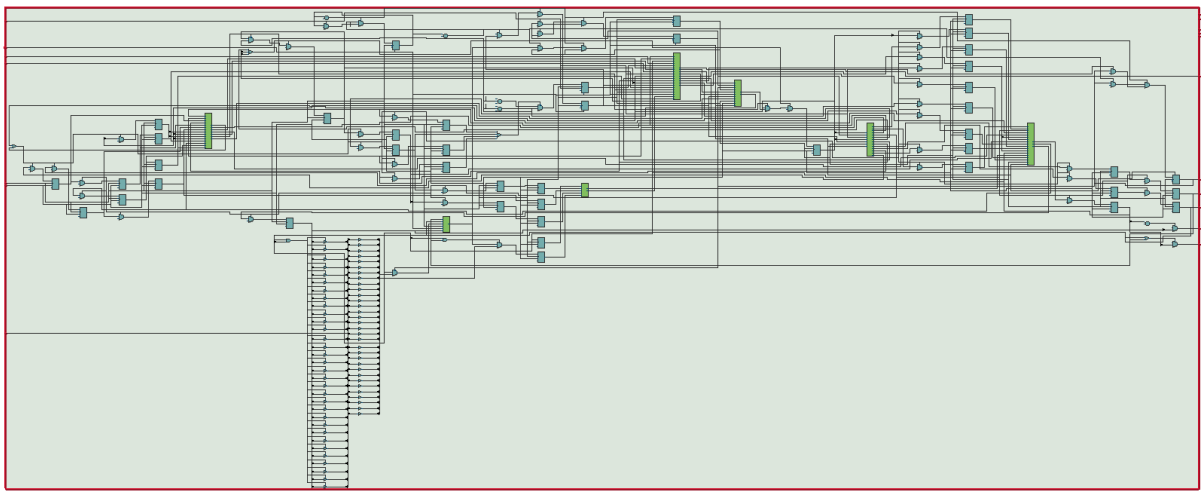
MIPS

רכיב זה הוא CPU של הMCU, כאשר הוא מסוג Pipeline בעל 5 דרגות והוא עובד כמובן לפי וון נוימן לכן יש לו זיכרון של ההוראות וזיכרון של המידע. כמו כן, הוא יודע לבצע forwarding בין השלבים השונים על מנת לחסוך במחזורי שעון. המעבד יודע לתמוך בהוראות המצורפות לתרגיל כך שהוא יודע לבצע את כל התוכניות שמשתמשות אך ורק בהוראות אלו. להלן שרטוט המעבד –



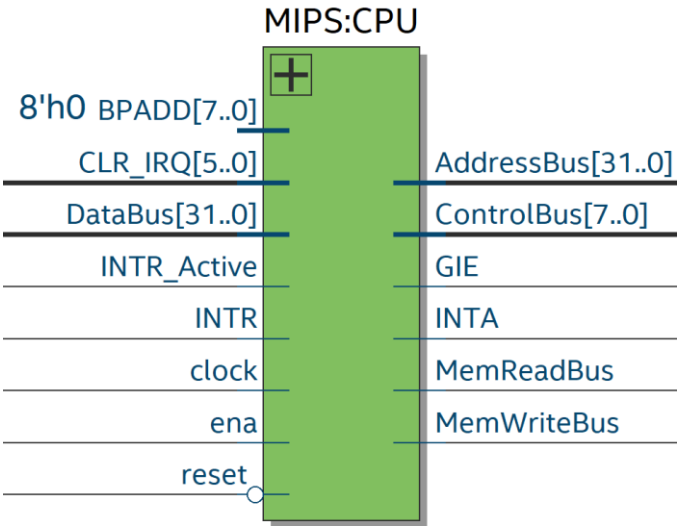
איור 4שרטוט מעבד הCPU

להלן דיאגרמת הRTL כללית במודול זה –



איור 5הRTL של הCPU

תיאור גרפי של המודול –



איור 6 הכניסות והיציאות של הCPU

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

Instruction Decode

בשלב זה נרצה לקחת את הפקודה שהוראה משלב הfetch ובעצם לפרק אותה לפי הסוג של הפקודה באופן הבא –

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)				

איור 7 קידוד הפקודות

כך נכניס את כתובת הרגיסטרים המתאימים לRF שנמצא בשלב הdecode ואך נוכל להוציא את תוכן הרגיסטרים המתאימים.

כמו כן, בשלב זה נבצע את חישוב הbranch והjump אם אחד מהם מתקיים, מפני שבשלב זה כבר נוכל לדעת הן אם התבצע jump (לפי מבנה הפקודה) או branch לאחר השוואת תוכן הרגיסטרים בהנחה ומעודכן (נדאג לזה ב Hazard unit).

כמו כן, בשלב זה יושבת גם יחידת הבקרה שמקבלת את הOp והFunct (פשוט לא מתייחסת לקו זה אם הפקודה אינה מסוג R-type) והיא מוציאה קווי בקרה מתאימים לאופן הפקודה שהתקבלה. לדוגמא, אם התקבלה פקודת חיבור נצטרך להוציא קווים מתאימים לALUSrc, ALUctl, RegWrite וכו'.

Execute

שלב הביצוע, כאן אנחנו מבצעים פקודות לזיכרון כמו חישוב כתובת או ביצוע פעולה מסוימת. זה נעשה באמצעות פקודות אריתמטיות שונות שנעשות ברכיב הALU.

כאמור עבור המעבד single-cycle, בשלב הביצוע מבוצעת גם חישוב הכתובת PC הבאה ופעולת הBranch. בנוסף כאשר עוברים למעבד Pipelined, נראה כי פקודת הbranch מביאה איתי control hazard, המעבד לא יודע מה הפקודה הבאה לבצע עבורה fetch מפני שהחלטת הbranch עוד לא בוצעה בזמן שהפקודה הבאה בוצעה לה fetch. דרך אחת לפתור זאת היא ע"י stall אבל זה יגרום להורדה ביעילות המערכת. דרך אחרת היא לחזות אם פקודת הbranch צריכה לקרות או לא ולבצע את הפקודות בהתאם לחיזוי. ברגע שהחלטת הbranch ידועה, ניתן לבצע flush ולזרוק את הפקודות אם החיזוי שגוי. זה יגרום לקצת שיפור אבל עדיין ביצוע flush להרבה פקודות כאשר מבצעים branch, מוריד את יעילות המערכת.

דרך נוספת והיא הדרך שבחרנו שבה אפשר להוריד את הבזבוז סייקלים של פקודות מבוזבזות במקרה של חיזוי שגוי היא בכך שנבצע את החלטת הbranch בשלב הdecode ביחד עם חישוב הכתובת PC הבאה. ביצוע ההחלטה היא בעצם לבצע השוואה בין שני ערכי רגיסטר. אז בכך שהזזנו את ההשוואה וחישוב הכתובת PC הבאה לשלב אחד קודם, ייעלנו את ביצועי המערכת.

בשלב הביצוע בנוסף אנחנו בודקים האם ישנו צורך לבצע forwarding הנועד לפתור בעיות של data dependencies עבור פקודות R-type וLW/SW.

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

מודול זה אחראי על ניהול החזרת המידע בסוף הpipeline והכתיבה שלו לזיכרון. קיימים הוראות שמבצעות שינוי ברגיסטרים כתוצאה מפעולות על רגיסטרים בALU וכתוצאה מפעולות מהזיכרון, לכן על מודול זה לנהל ולדעת איזה קו להחזיר ולכתוב אותו לרגיסטר.

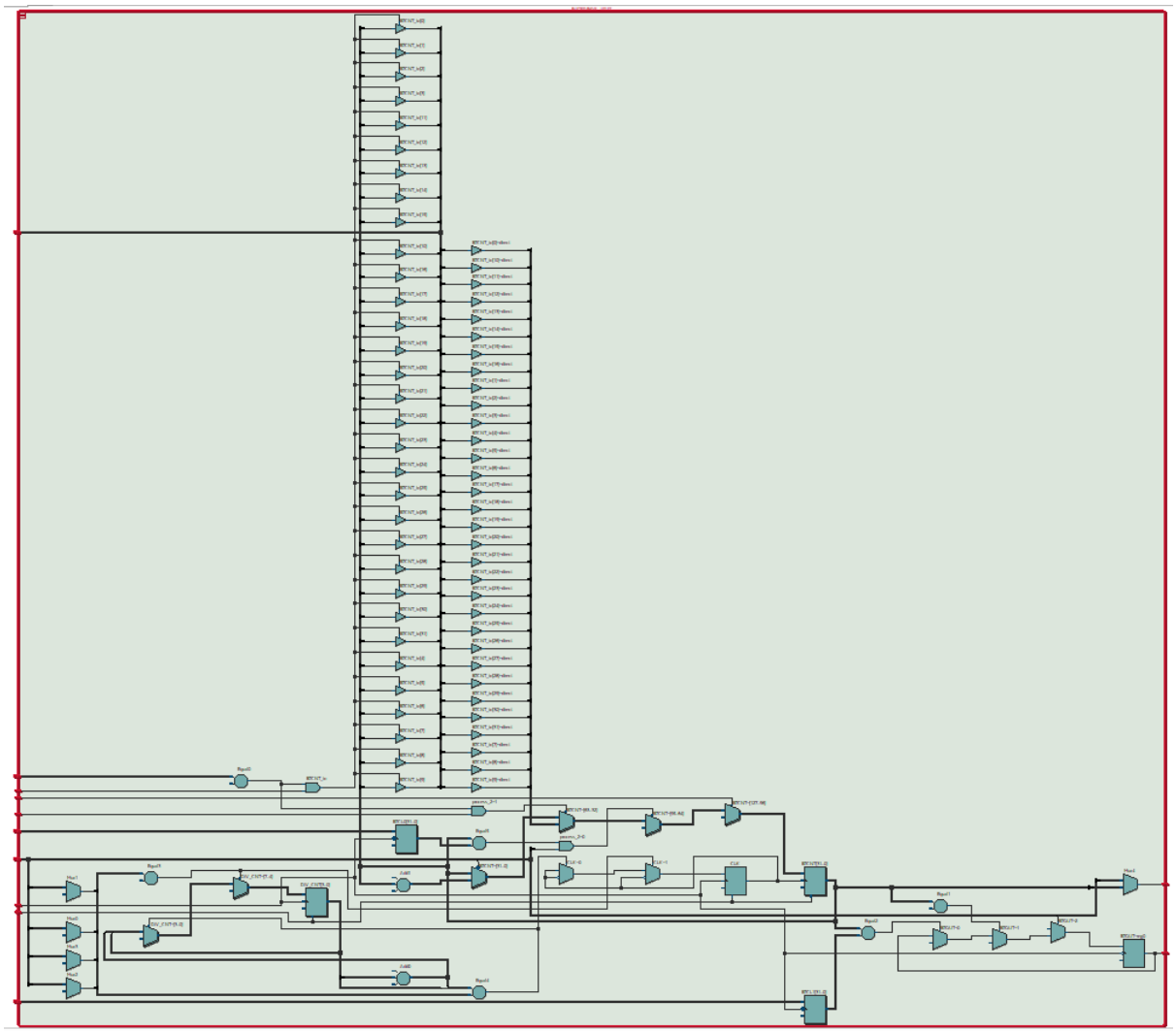
Basic Timer

רכיב פריפריאלי זה אחראי על 2 מטרות.

(1) להיות רכיב חומרה פשוט שסופר על סמך עליות שעון. הוא מכיל רגיסטר בקרה שמכיל הגדרות שונות בהתאם להגדרות המשימה 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. מטרה זו שימושית למשימות הדורשות תזמון כמו אירועים תקופתיים או הפעלת פעולות במרווחי זמן ספציפיים.

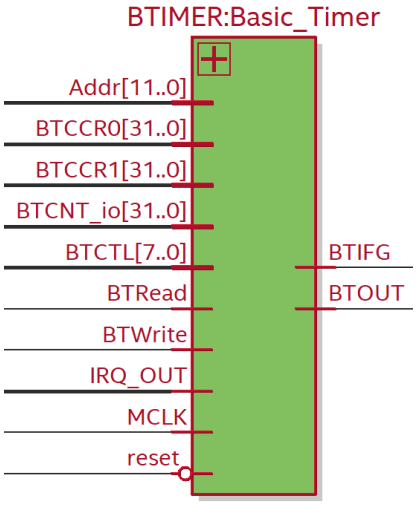
(2) לייצר אות PWM באמצעות הטיימר הבסיסי הזה כך שהטיימר סופר ובאמצעות CCRx ניתן לשלוט על הזמן מחזור של האות ועל הDuty Cycle שלו באמצעות רגיסטרים CCR0 וCCR1 בהתאמה.

להלן דיאגרמת הRTL כללית במודול זה –



איור 8 הRTL של הBasic Timer

תיאור גרפי של המודול –



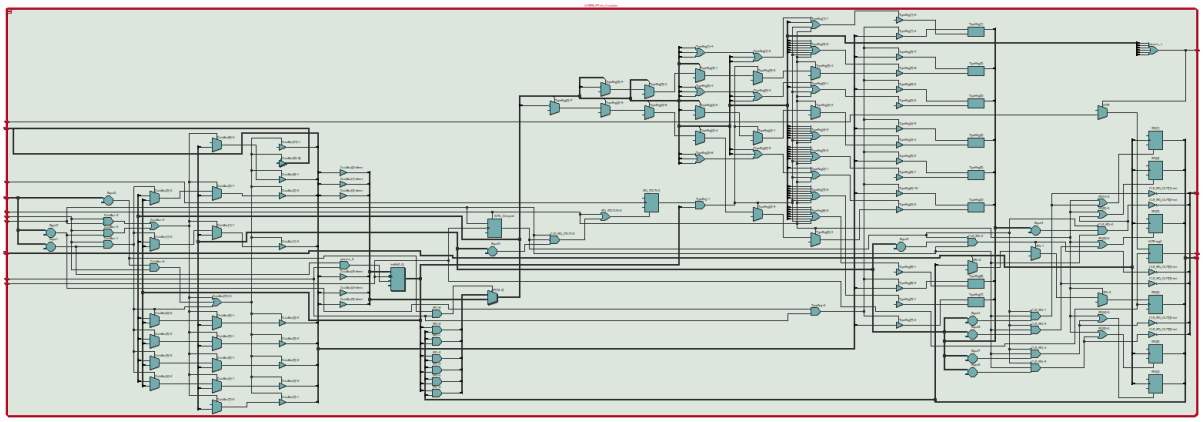
איור 9 הכניסות והיציאות של הBasic Timer

Interrupt Controller

בפרויקט הוספנו תמיכה בפסיקו חיצוניות שבאות מפריפריות החומרה שהוספנו, כך שהם יהיו מתועדפות לפי הסדר הנתון בהגדרת המשימה. פסיקה מתבצעת כאשר אחד מרכיבי החומרה השונים מבקשים לבצע פסיקה אזי הם מעלים סיגנל בשם IRQ שמבקש לבצע פסיקה, כך שבפועל מתבצעת פסיקה רק אם גם הפסיקה הרלוונטית מאופשרת וגם לא קיימת פסיקה נוכחת.

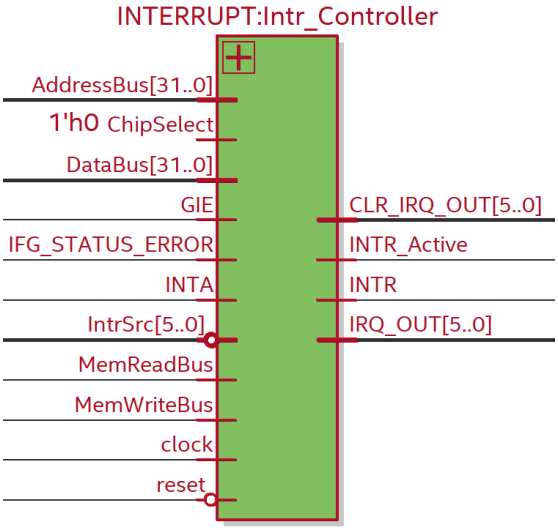
פרוטוקול הכניסה לפסיקה הוא עצירת הPipeline וסיום הפקודות הנותרות, נשים את הכתובת של הISR אליה נרצה לקפוץ על הBUS, נמצא במרחב הזיכרון את הכתובת בפועל אליה נרצה לקפוץ בעזרת $Mem[Type]$ ואז נבצע קפיצה. כמובן שגם נשלח ACK שאכן התקבל הINTR בצד המעבד.

להלן דיאגרמת הRTL כללית במודול זה –



איור 10הRTL של בקר הפסיקות

תיאור גרפי של המודול –

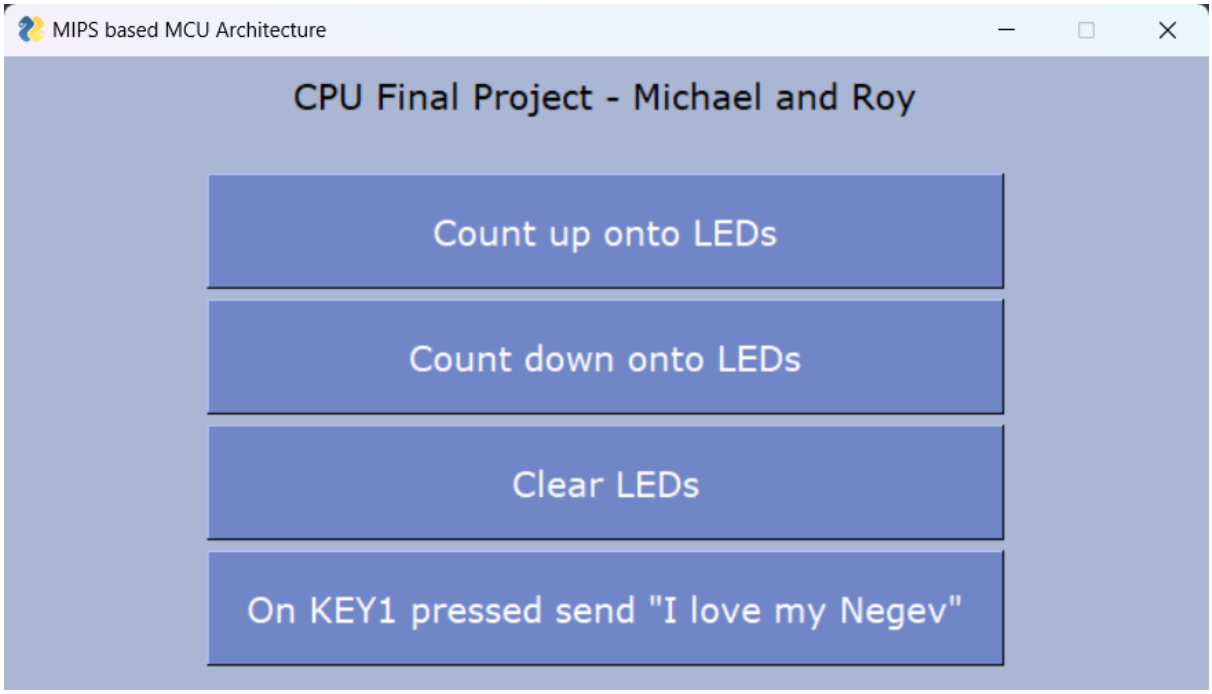


איור 11הכניסות והיציאות של בקר הפסיקות

UART

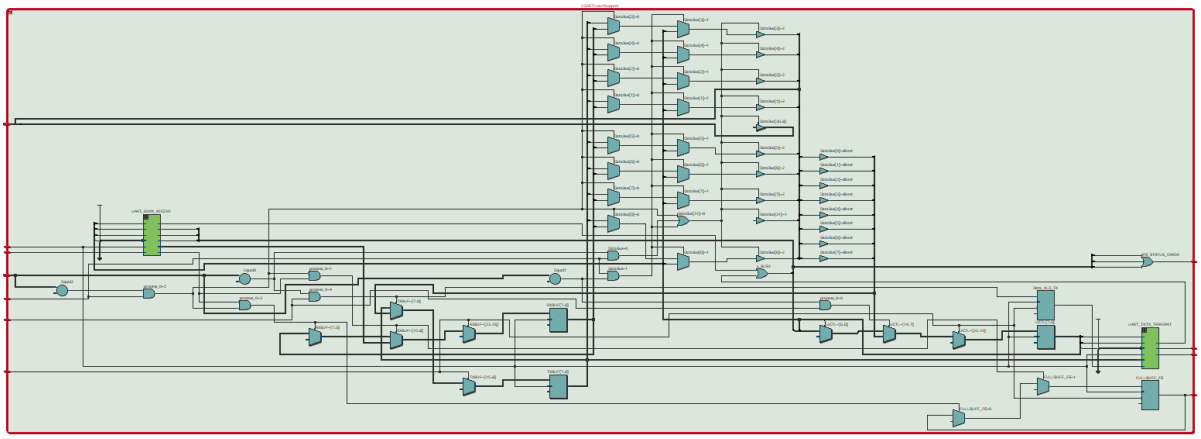
בכדי להעביר מידע מהMCU אל רכיבים אחרים הוספנו את פריפריית UART שניתן באמצעותו להעביר מידע בין הMCU לצד חיצוני באמצעות תקשורת UART ללא הפרעה למעבד, כך שבכל שליחה או קבלה כאשר הBUF מתמלא נבצע פסיקת שליחה או קבלה ובין הפסיקות המעבד יפעל באופן רציף.

בפרויקט ביצענו GUI שמכיל תפריט המבצע את הפעולות השונות, כך שקיים קוד אסמבלי מתאים שבמידה והתקבלה לחיצה אזי הוא מזהה איזה STATE נלחץ באמצעות התוכן שקיים בRXBUF ואז מבצע את הרוטינה המתאימה.
להלן הGUI –



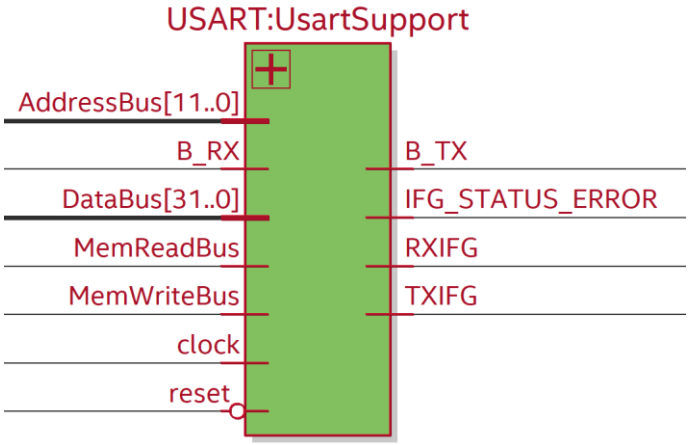
איור 12 הGUI של ממשק ה UART בצד המחשב

להלן דיאגרמת הRTL כללית במודול זה –



איור 13 הRTL של הUART

תיאור גרפי של המודול –

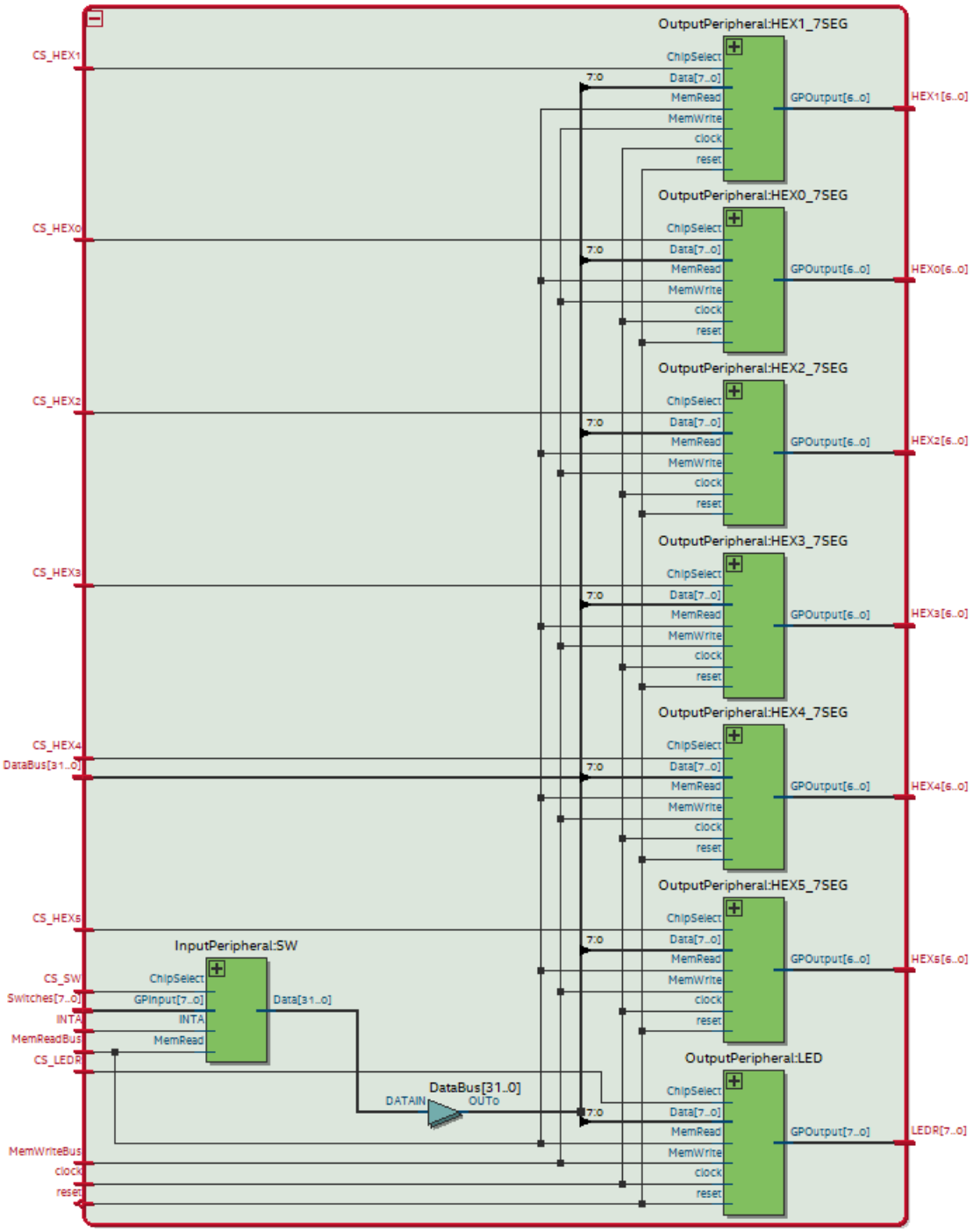


איור 14 הכניסות והיציאות של הUART

GPIO

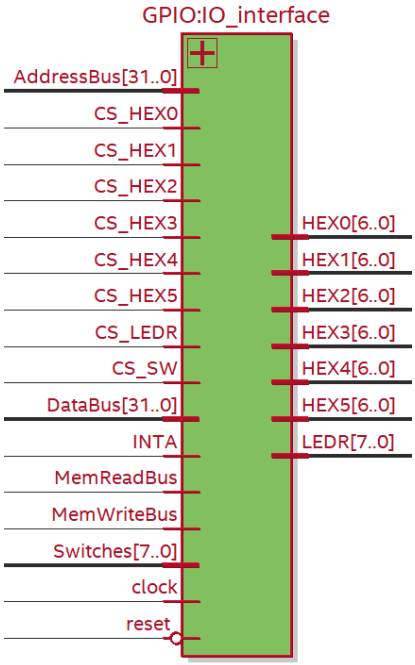
בכדי להוסיף ממשק משתמש, הוספנו ממשק כניסות ויציאות לMCU שיכול להכיל כגון לדים, הקסות, כפתורים וכדומה. הכתיבה והקריאה מרכיבי הGPIO היא דרך הקווי הBUS המצורפים במערכת בהתאם לצורך. כמו כן, ממשק זה כמובן שיכול לתת פסיקות כמו לחיצה על כפתור שתוביל לISR שיבצע רוטינה כלשהי.

להלן דיאגרמת הRTL כללית במודול זה –



איור 15 הRTL של הGPIO

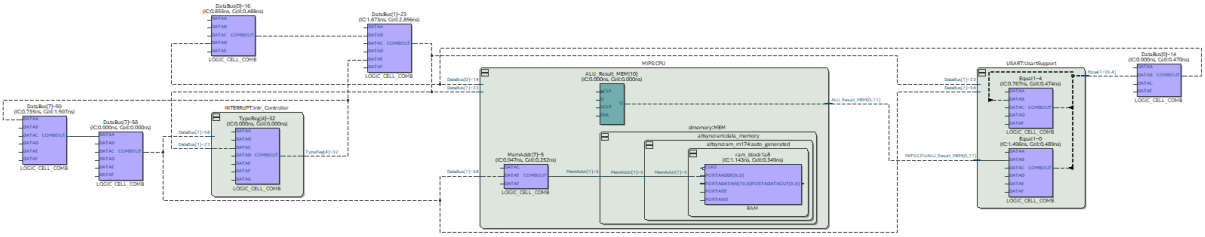
תיאור גרפי של המודול –



איור 16 הכניסות והיציאות של הGPIO

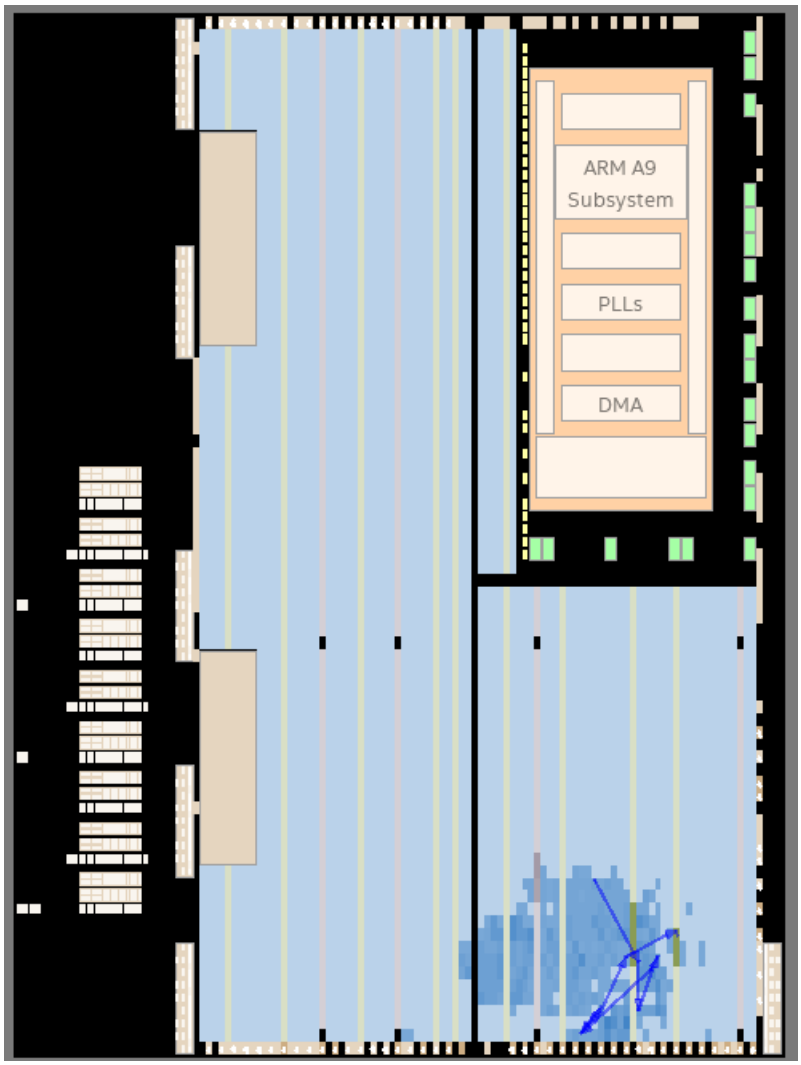
נתיב קריטי

הנתיב הקריטי הוא הנתיב הארוך ביותר של התפשטות הלוגיקה בתוך המעגל בדיגיטלי של עיצוב הMCU שבנינו במהלך הפרויקט. הוא מייצג את הנתיב האיטי ביותר במערכת, כך שנצטרך לקבוע לפיו את תדר השעון המירבי שניתן להשיג עבור תכנון הMCU. להלן הנתיב הקריטי במערכת שלנו –



איור 17 הנתיב הקריטי המערכת בתצורת RTL

כמו גם על הFPGA עצמו –



איור 18 הנתיב הקריטי על גבי פיסת הFPGA

להלן התדר המקסימלי שנקבע כתוצאה מהנתיב הקריטי –

Slow 1100mV 85C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name	Note
1	32.71 MHz	32.71 MHz	clock	

איור 19 התדר המקסימלי במערכת

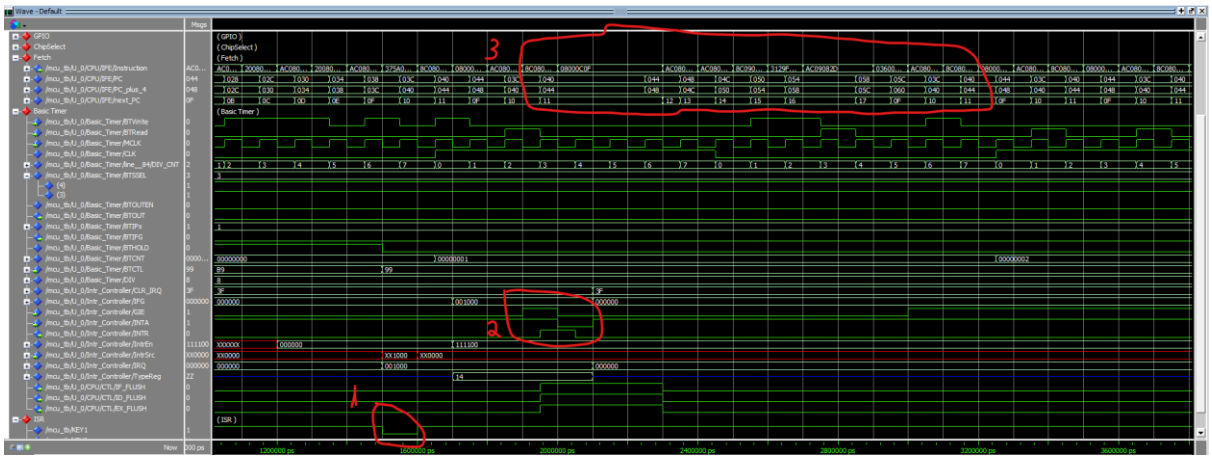
ניתן גם להוסיף PLL על מנת להגדיל את השעון וכך להביא לביצועים מהירים יותר, אך בעקבות הדרישות של הפרויקט שהתוכנית צריכה לפעול בזמנים שנמצאים בטווח מהירות התגובה של האדם האנושי, כך שהיא לא צריכה להיות מהירה. כמו כן גם לא נדרש לבצע חישובים מסובכים שעלולים לקחת הרבה זמן, אזי החלטנו להשאיר את תדר השעון כפי שהוא ללא PLL.

ניתוח תוצאות

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

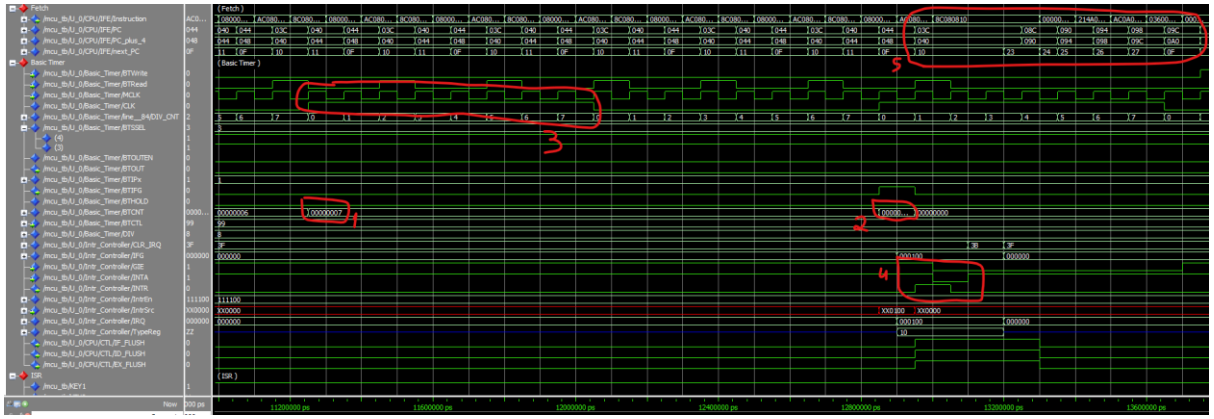
ניתוח גלים בModel SIM

בבדיקה זו נבצע בדיקה עם כמה מודולים כך שנראה שגם מתקיימת פסיקה כאשר נלחץ מקש KEY1 וגם מתקיימת פסיקה כאשר הטיימר מגיע לערך עבורו נבצע פסיקה. ביצענו סימולציה של לחיצה על המקש וקינפגנו בקוד האסמבלי את רגיסטר הבקרה של הטיימר כך שיעלה פסיקה עבור ספירה עד 2^3. להלן תוצאות הגלים –



איור 20 ניתוח גלים מקרה 1 מודלסים

ניתן לראות בעיגול 1 שמבצעת לחיצה על כפתור 1 כך שהערך יורד לנמוך. בעיגול 2 ניתן לראות את קבלת הINTR מצד הCPU עקב לחיצת הכפתור והאישור באמצעות INTA. בעיגול 3 ניתן לראות שהבקר אכן קופץ ויוצא מהלולאה האינסופית וממשיך לבצע את רוטינת השירות של פסיקה של הכפתור KEY1 ואז חוזר חזרה ללולאה האינסופית.



איור 21 ניתוח גלים מקרה 2 מודלסים

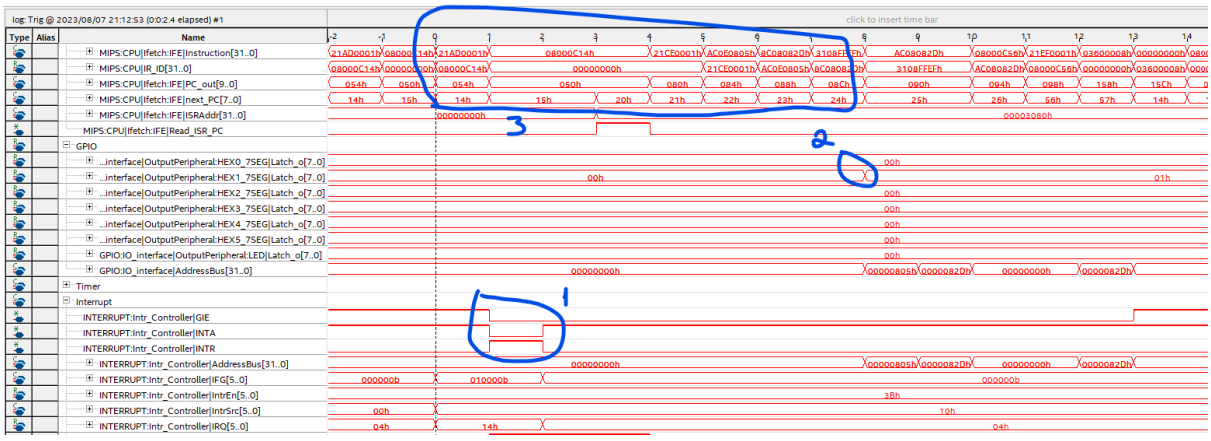
בעיגול 1 ניתן לראות את ערך המניה של הCNT מגיע לערך 7 כך שהוא מתקרב להגעה לפסיקה שעתידה להגיע ב2^3 ואכן בעיגול 2 ניתן לראות שהתקבל הערך 8 שאמור להביא פסיקה שאכן מתקיימת ויוצאת לפועל בעיגול 4. ניתן לראות שאכן התבצעה קפיצה בעיגול מספר 5 לISR המתאים של הפסיקה של הטיימר.

כמו כן, בעיגול מספר 3 ניתן לראות שמניית CNT שאמורה להתבצע בחלוקה של 8, אכן סופרת 8 עליות שעון של השעון המרכזי וכל 8 עליות היא מבצעת היפוך ככה שהיא מתחלקת ב8 כרצוי.

ניתוח גלים בSignal Tap

כעת נבצע בדיקה שהדברים עובדים כמצופה עם קוד האסמבלי המצורף בעבודה שהוא מטפל בפסיקות שונות של כל רכיבים הפרפיאליים, בין היתר גם במודול הUART.

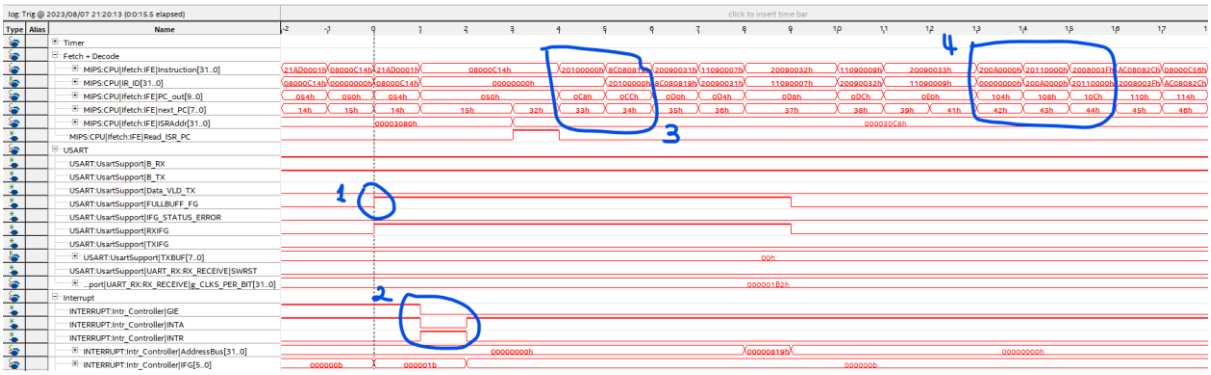
בתור התחלה ביצענו לחיצה על כפתור 2 כאשר במהלך הISR שלו נרצה לעדכן את הערך של HEX1 להיות ערך מניה כאשר מתחיל ב0.



איור 22 ניתוח גלים מקרה Signal Tap 1

ניתן לראות בעיגול 1 שאכן הMCU הצליח לתפוס את הלחיצה ואכן העלה את רוטינת הכניסה לINTR, כמו כן, ניתן לראות בעיגול מספר 3 שערך הPC התעדכן להיות הערך של רוטינת הISR המתאימה. כמובן במהלך הרוטינה העלנו את ערך HEX1 להיות 1 כתוצאה מערך המניה כאשר ניתן לראות זאת בעיגול 2.

כעת נבצע את הבדיקה של הUART במהלכה נלחץ על הכפתור הראשון בGUI המצורף, כלומר מניה למעלה בלדים ונצפה לקבל פסיקה כאשר הבאפר התמלא. להלן הגלים –



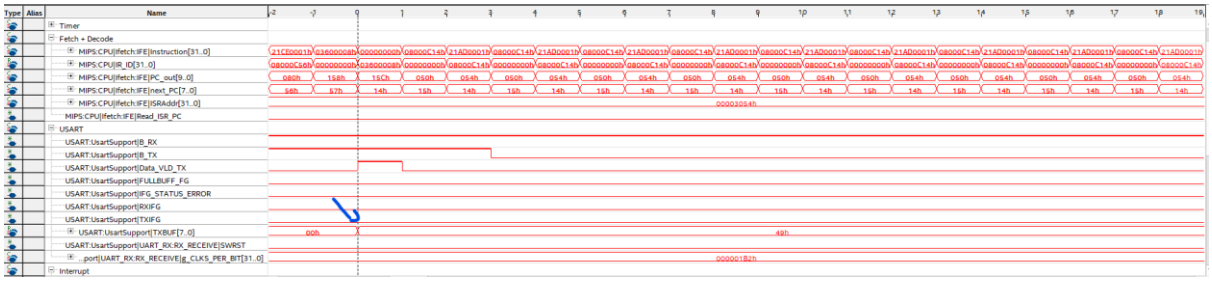
איור 23 ניתוח גלים מקרה Signal Tap 2

אז בעיגול אחד אנחנו רואים כי הבאפר התמלא ולכן הדגל שלו עלה ל-1 ולכן הפסיקה של RX גם כן עלתה ואז בעיגול 2 אכן רואים כי התקבלה פסיקה ובעיגול 3 רואים את הקפיצה לרוטינת שירות של פסיקת RX ובעיגול 4 זה הקפיצה לפונקציה של המנייה מעלה של הלדים :

```
0x200a0000 addi $10,$0,0x00000000 151:        addi $t2, $0, 0        # init timer
0x20110000 addi $17,$0,0x00000000 152:        addi $s1, $0, 0        # 0 when count up (1 when down)
0x2008003f addi $8,$0,0x0000003f  153:        addi $t0,$0,0x3F       # BTIE is enabled
0xac08082c sw $8,0x0000082c($0)   154:        sw   $t0,0x82C
0x08000c56 j 0x00003158           155:        j    LeaveISR
```

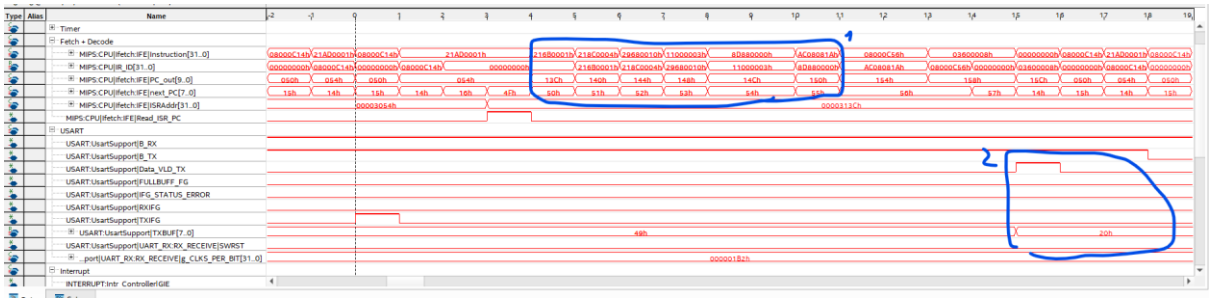
איור 24 קידוד הפקודות בMARS

בבדיקה הבאה, נבדוק את התקשורת חזרה למחשב, ולכן נראה מתי הסיגנל של הDATA_VLD_TX עולה ל-1 האומר שמתחיל שידור חזרה. ואנו רואים בחץ שבבאפר של הTX יש את האות הראשונה של המילה שרוצים לכתוב כלומר 49h שזה האות I, כלומר הוא מוכן לשליחה.



איור 25 ניתוח מקרה Signal Tap 3

במסך הבא, נדגום מתח הIFG של הTX עולה ל-1 וניתן לראות שהוא נכנס לרוטינת שירות של פסיקת TX בעיגול 1 עבור השליחה של האות I ואז בעיגול 2 אנחנו רואים כמו במקרה הקודם עבור המידע הבא שרוצים לשלוח שבמקרה שלנו זה 20 הקסה כלומר האות l.



איור 26 ניתוח מקרה Signal Tap 4