אוניברסיטת בן גוריון בנגב

בית הספר להנדסת חשמל ומחשבים

קורס: מעבדת ארכיטקטורת מעבדים מתקדמת ומאיצי חומרה 361.1.4693

FINAL PROJECT

מגישים: רומי לוסטיג 211542089

עומר פינטל 316085554

מדריך: מר רבוא חנניה

תאריך הגשה: 04.09.2024

תוכן עניינים

קורס : מעבדת ארכיטקטורת מעבדים מתקדמת ומאיצי חומרה 361.1.4693 1

1. מבוא : 3

1.1 מטרת הפרוייקט - 3

2. רכיבים במודל MCU : 3

2.1 – MIPS 3

2.2 - GPIO 7

2.3 - DIVIDER 9

2.4 - BASIC TIMER 11

2.5 – INTERRUPT CONTROLLER 12

3. מציאת Fmax, Critical Path : 14

3.1 – Fmax 14

3.2 – Critical Path 14

4. תוצאות : 15

1. מבוא:

1.1. מטרת הפרוייקט -

- מטרת העל של הפרוייקט הינה ליצור מעבד מסוג MIPS, אשר תומך במרבית הפקודות של המעבד הנ"ל (ניתנה רשימה) – ומציג את התוצאות הרלוונטיות, כאשר מתבקש, לרכיבים פריפריאליים. בנוסף, נתבקשנו לתכנן את המעבד כך שיכול לתמוך בפסיקות בעת הצורך. הפסיקות הללו, גם הן קורות באמצעות רכיבים פריפריאליים.
- כאמור, כמו במעבדות שעשינו בעבר: כל מערכת שיצרנו מורכבת מרכיב top כללי, אשר מורכב מרכיבים קטנים יותר, אשר גם הם מתפצלים לרכיבים קטנים (וכן הלאה).
- במעבד שיצרנו, הרכיבים הינם:
  - מערכת MCU, המערכת הכוללת שלנו:
    - רכיב MIPS – המעבד שיצרנו:
      - IFETCH
      - IDECODE
      - EXECUTE
      - DMEMORY
      - CONTROL
    - רכיב פריפריאלי – GPIO:
      - OptAddrDecoder
      - InputPeripheral
      - OutputPeripheral
    - SevenSegDecoder
    - רכיב פריפריאלי – DIVIDER
    - רכיב פריפריאלי – Basic Timer
    - Interrupt Control:
      - Priority Encoder

כל ההסברים על הרכיבים בהמשך.

להוסיף עוד

2. רכיבים במודל MCU:

2.1. MIPS –

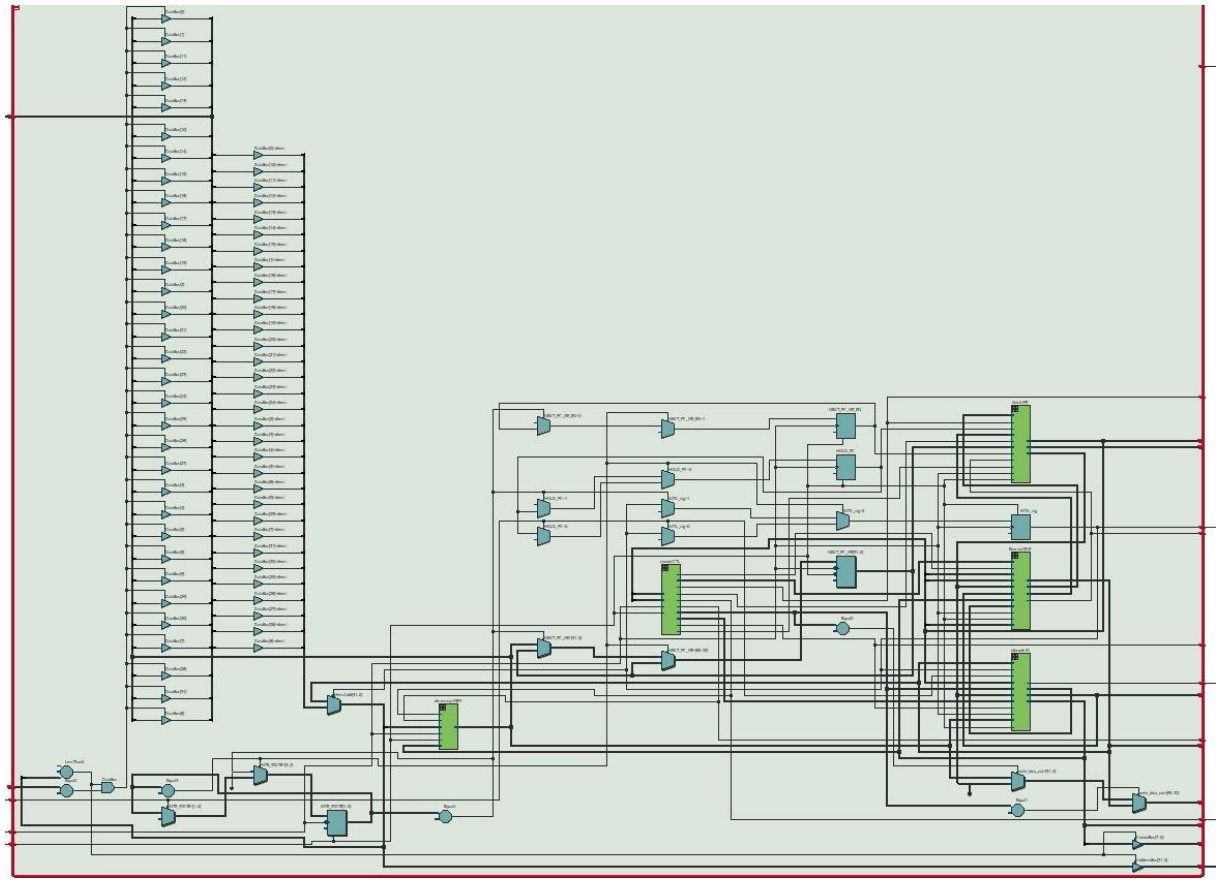
- זהו הרכיב העיקרי של המודל, המבצע את כל ההוראות הדרושות לפרוייקט. הוראות אלו כוללות פקודות אריתמטיות/ לוגיות כגון add, and, sub, mul (המתבצעות בין שני רגיסטרים – R_type /

פעולות המתבצעות עם קבוע כלשהו – I_type), פקודות הקשורות לזיכון: lw, sw, ופקודות הקופצות בין מקומות בקוד (בין אם מותנה: branch, או לא מותנה: j, jal).

- למדנו בתיאוריה כי הפקודות יכולות להתבצע באמצעות רכיבים שונים (לחלק את הארכיטקטורה הכוללת של פקודה – למיקרו ארכיטקטורות), וע"פ הדרישה: המיקרו ארכיטקטורה שהייתה בפרוייקט זה הינה באמצעות המודולים: ifetch, idecode, execute, dmemory, control, כאשר:
  - ifetch – משמש להבאת פקודה חדשה בכל פעם. אנחנו נמצאים בתצורה של מודל פן נוימן, ולכן יש PC (program counter) שעובר על ההוראות לפי סדר של זמנים, בתצורת control flow. בנוסף, כאשר מתבצעת קפיצה בקוד – הכתובת החדשה מתעדכנת והifetch יכול להביא את הפקודה החדשה. דבר זה נעשה באמצעות mux.
  - idecode – משמש ל"קידוד" של הפקודה שהובאה מהifetch, והבנת איזו פקודה מתבצעת בכל פעם. שלב ביניים המקשר בין הבאת הפקודה לבין ביצועה.
  - execute – שלב ביצוע הפעולה. דבר זה אומר: ביצוע פעולות לוגיות / אריתמטיות באם נדרש. העברת כתובות רלוונטיות לBUS (שדרכו עובר מידע) כאשר מדובר בפקודות המשתמשות בכתובות, וכן הלאה. כל הפעולות מתבצעות ברכיב זה.
  - dmemory – רכיב המקבל כתובות וקורא / כותב מהזיכרון של התכנית (גם מידע וגם הוראות) באמצעות port map אותו קיבלנו ממעבדה 5 (שלא נדרשנו לבצע).
  - control – רכיב השולט על כל סיגנלי הבקרה, אותם מדליקים ומכבים בעת הצורך על מנת לבצע את הפקודות כמו שצריך. דוגמה לכך, היא סיגנל Memread, מסוג std_logic אותו אנו מדליקים ('1') ומכבים ('0') כאשר אנו נרצה לקרוא מהזיכרון / לא לקרוא מהזיכרון בהתאמה. נדע מתי נרצה לעשות זאת באמצעות ממשק לרכיב הidecode – שמבהיר לנו מהי הפקודה בכל פעם.

- מימשנו לאורך כל הקודים את הלוגיקה של התיאורה אותה למדנו בקורס המקביל "ארכיטקטורת יחידת מעבדים – תיאוריה".

- להלן הדברים שנתבקשנו להראות לאחר יצירת הקבצים הללו:
  - תוצאות RTL Viewer-

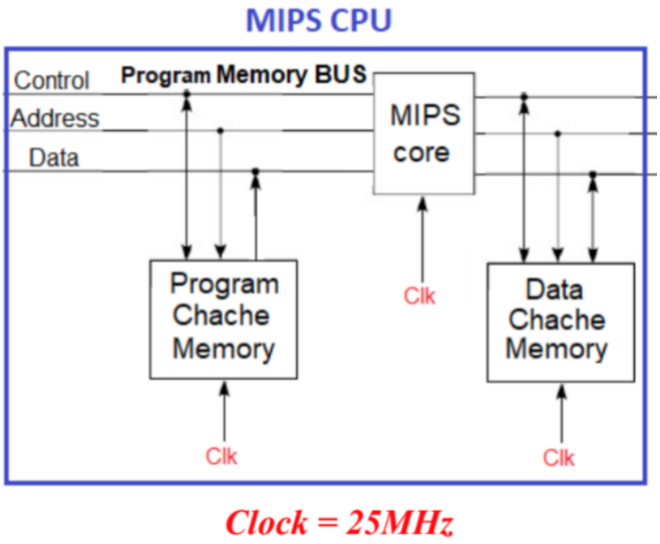


איור 2.1 – RTL Viewer של הMIPS

- הסבר לוגי על המודול-

פועל בתצורת port maps : יש 5 קומפוננטות המבצעות את כל אחד מהשלבים אותם תיארנו מעלה. החיווטים מתבצעים באמצעות port map, כאשר הכתובת נכנסת מהfetch -> הdecode מפרק את הפקודה למה היא ההוראה בפועל-> הcontrol מעלה את קווי הבקרה באמצעות ההוראה שהתקבלה -> הexecute מבצע את ההוראה באמצעות קווי הבקרה -> הmemory קורא / כותב מידע מהזיכרון / לזיכרון בעת הצורך.

- הסבר גרפי-



גרף 2.1 – רכיב הmips כפי שנתבקשנו לבצע

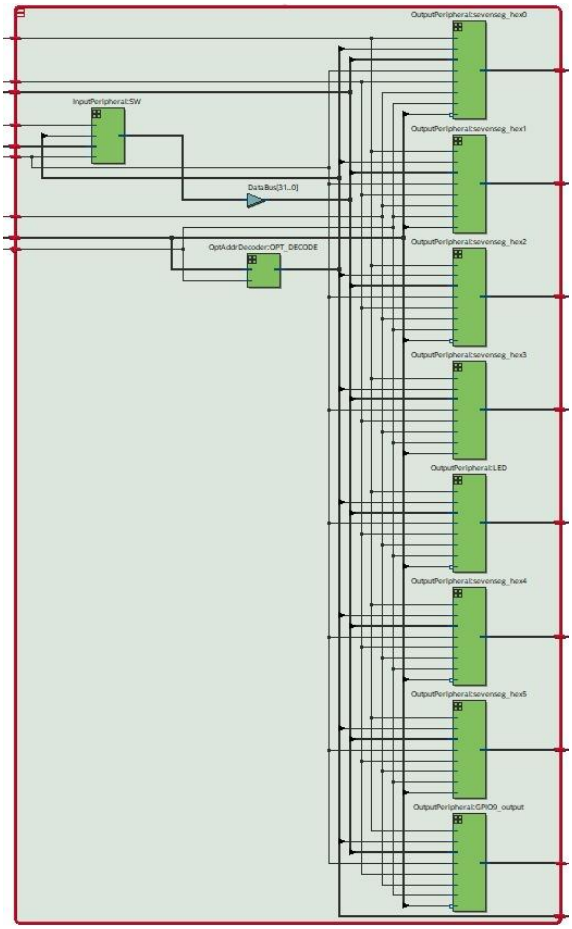
- טבלת ports-

type	size	mode	port
Std_logic	1	in	rst, clk
Std_logic_vector	10	out	PC
Std_logic_vector	32	out	ALU_result_out, read_data_1_out, write_data_out, instruction_out
Std_logic	1	out	branch_out, zero_out, MemWrite_out, Regwrite_out
Std_logic	1	out	MemReadBus
Std_logic	1	out	MemWriteBus
Std_logic_vector	32	out	AddressBus
Std_logic	1	out	GIE
Std_logic	1	in	INTR
Std_logic	1	out	INTA

Std_logic_vector	32	inout	**DataBus**
Std_logic_vector	8	out	**ControlBus**
Std_logic_vector	7	in	**CS_vec**

2.2. GPIO -

- זהו הרכיב הפריפריאלי הראשון, הממשק את הממשק בין הMIPS לבין הרכיבים של הi/o interface – ז"א רכיבי ה - LEDs, Switches, HEX's, KEYs איתם נוכל לראות את הפעולות האריתמטיות / לוגיות שיצרנו, וכן הלאה.
- הGPIO הינו קומפוננט שמכיל מספר מודולים פנימיים – הכוללים input בלבד, וoutput בלבד. זאת משום שלמדנו כי בפרוייקט זה, רכיבי הinput ורכיבי הoutput עובדים באופן בלתי תלוי, ואינם קשורים זה לזה. לכן – יש מודול לכל אחד בנפרד. בנוסף, ע"מ שנוכל להבין מהו הרכיב i/o הרלוונטי – יצרנו encoder שמפענח זאת בכל פעם.
- להלן הדברים שנתבקשנו להראות לאחר יצירת הקבצים הללו :
  - תוצאות RTL Viewer-



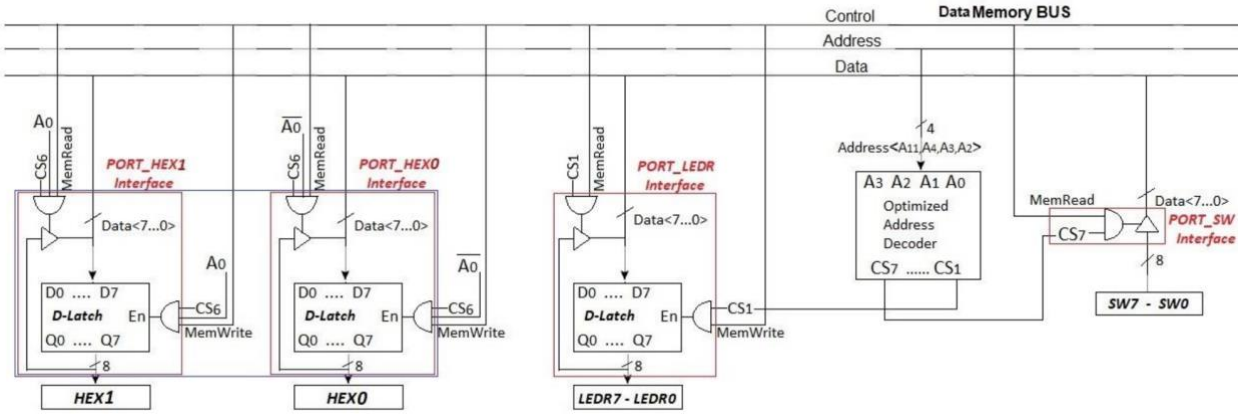
איור 2.2 – RTL Viewer של הGPIO

- הסבר לוגי על המודול-

פועל בתצורת port maps : יש 2 סוגים של קומפוננטות : inputs, outputs – כאשר לכל אחד מהרכיבים הפריפריאליים אנו מבצעים את הport map בהתאם.

נוסף על כך, יש את רכיב הencoder שמפענח מהו הרכיב הרלוונטי, וכאשר יש צורך ברכיב כלשהו – הוא מעלה דגל בוקטור CS (chip select), כלומר מעלה את האיבר ה-i בCS_vec ל'1' כאשר יש צורך בשימוש ברכיב הi.

- הסבר גרפי-



גרף 2.2 – הסבר על מימוש רכיב הGPIO

למעשה מימשנו את כל אחד מהרכיבים בנפרד, וחיברנו ביניהם ע"י שימוש בand, or וכן הלאה כנדרש.

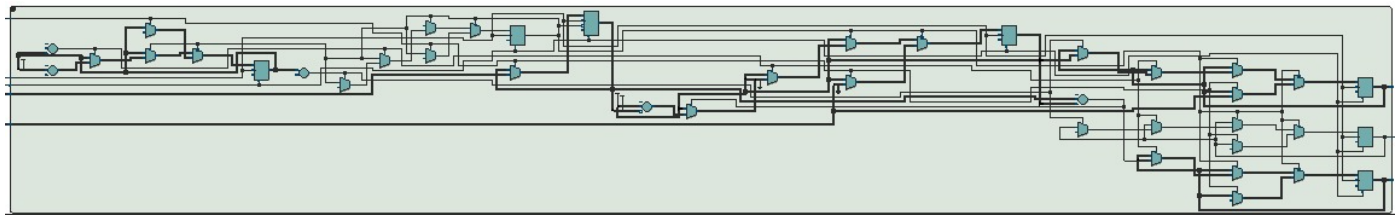
- טבלת ports-

type	size	mode	port
Std_logic	1	in	**ack**
Std_logic	1	in	**MemRead**
Std_logic	1	in	**clk, rst**
Std_logic	1	in	**MemWrite_Contol_Bus**
Std_logic_vector	32	in	**Address_Bus**
Std_logic_vector	32	inout	**DataBus**
Std_logic	1	out	**BTOUT**
Std_logic_vector	7	out	**HEX0, HEX1, HEX2, HEX3, HEX4, HEX5**

Std_logic_vector	8	out	**LEDs**
Std_logic	1	out	**PWM**
Std_logic_vector	8	out	**Switches**
Std_logic_vector	7	out	**CS_vec_out**

2.3. DIVIDER -

- זהו הרכיב הפריפריאלי השני, הממש חילוק של מספר חיובי במספר חיובי אחר כלשהו, ומעלה פסיקה כאשר מסיים לבצע את הפעולה שלו.
- זהו רכיב שפועל באופן בלתי תלוי בMIPS, כלומר אינו חלק מהוראות הISA, ולכן יכול להתבצע בכל פעם כאשר מאפשרים לו לעשות זאת. ע"פ הדרישה, נתבקשנו לאפשר לdivider להתבצע כאשר הרכיב divisor נכתב ע"י היוזר. ע"מ לעשות זאת, יצרנו דגל הנקרא "ena", שנדלק כאשר כתובת המשמשת לאחסון המידע של divisor עוברת בAddressBus – משמע, יש פקודה הכותבת לזיכרון בכתובת זו ע"י היוזר.
- להלן הדברים שנתבקשנו להראות לאחר יצירת הקבצים הללו:
  - תוצאות RTL Viewer-



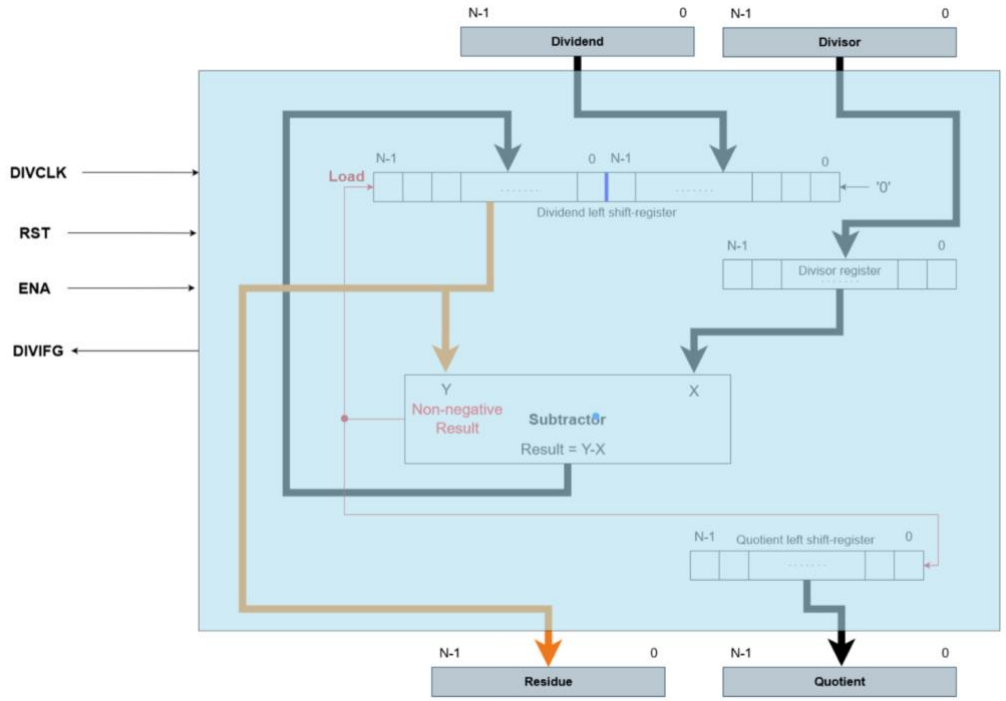
איור 2.3 – RTL Viewer של הdivider

  - הסבר לוגי על המודול-

פועל על פי הלוגיקה שלמדנו עליה במעבדה.

  - הסבר גרפי-

להלן ההסבר שלמדנו עליו:



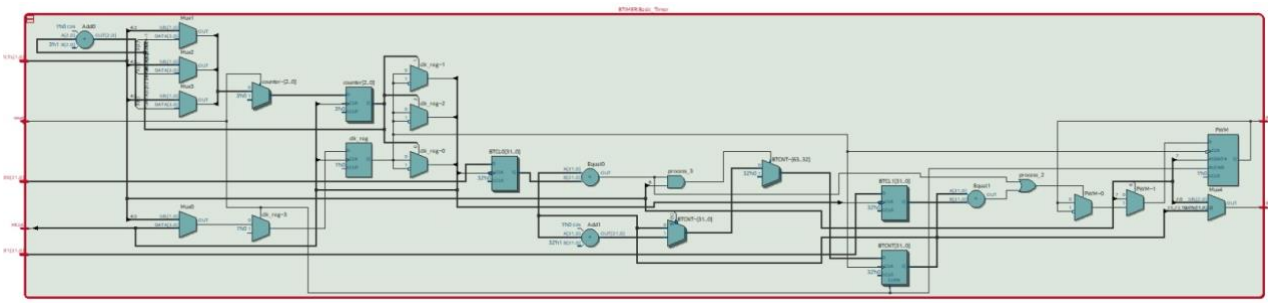
גרף 2.3 – הסבר על לוגיקת רכיב הdivider

- טבלת ports-

type	size	mode	port
Std_logic_vector	32	in	**DIVIDEND**
Std_logic_vector	32	in	**DIVISOR**
Std_logic	1	in	**DIVCLK**
Std_logic	1	in	**RST**
Std_logic	1	in	**ENA**
Std_logic	1	out	**DIVIFG**
Std_logic_vector	32	out	**RESIDUE**
Std_logic_vector	32	out	**QUOTIENT**

2.4. BASIC TIMER -

- זהו הרכיב הפריפריאלי השלישי, הממש פעולה המוציאה אות pwm, ומוציאה פסיקות כאשר נדרש. הרכיב מקבל כניסת שעון ו2 מספרים בגודל של 32 ביטים, ורגיסטר קונטרול.
- הרכיב יודע להשתמש בשעון בו הוא קיבל, או לחלק את התדר שלו ולהשתמש בהם. דבר זה נעשה כתלות בBTSSEL. הרכיב יודע להוציא פסיקה כל 2^x, מחזורים כתלות בBTPI.
- כתלות בשני המספרים שקיבלנו, הרכיב יודע להוציא אות pwm בדיוק לפי הלוגיקה שיצרנו במעבדה 4.
- בנוסף, ישנם ביטי בקרה נוספים לhold, ena, mode.
- להלן הדברים שנתבקשנו להראות לאחר יצירת הקבצים הללו :
  - תוצאות RTL Viewer-



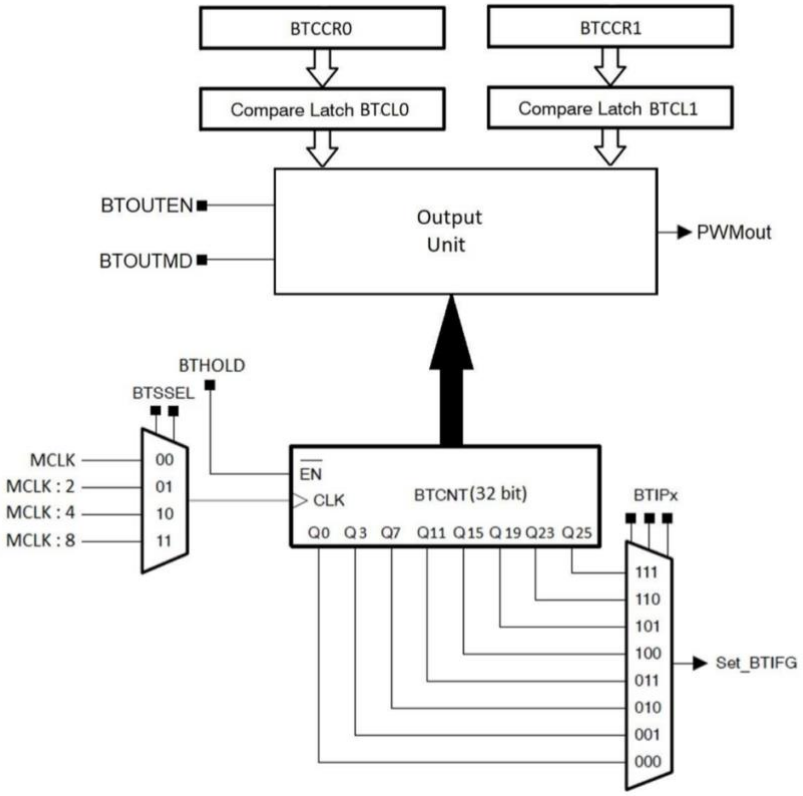
איור 2.4 – RTL Viewer של הbasic timer

  - הסבר לוגי על המודול-

ע״מ לחלק את השעון, על סמך השעון שקיבלנו, השתמשנו במחלק תדר על בסיס דגימת ערכי הביטים של רגיסטר counter. שאר הלוגיקה של מוצא אות הpwm, נעשה כפי שכתבנו עליה בקובץ מעבדה 4.

  - הסבר גרפי-

להלן ההסבר שלמדנו עליו :



גרף 2.4 – שרטוט רכיב הBasic Timer

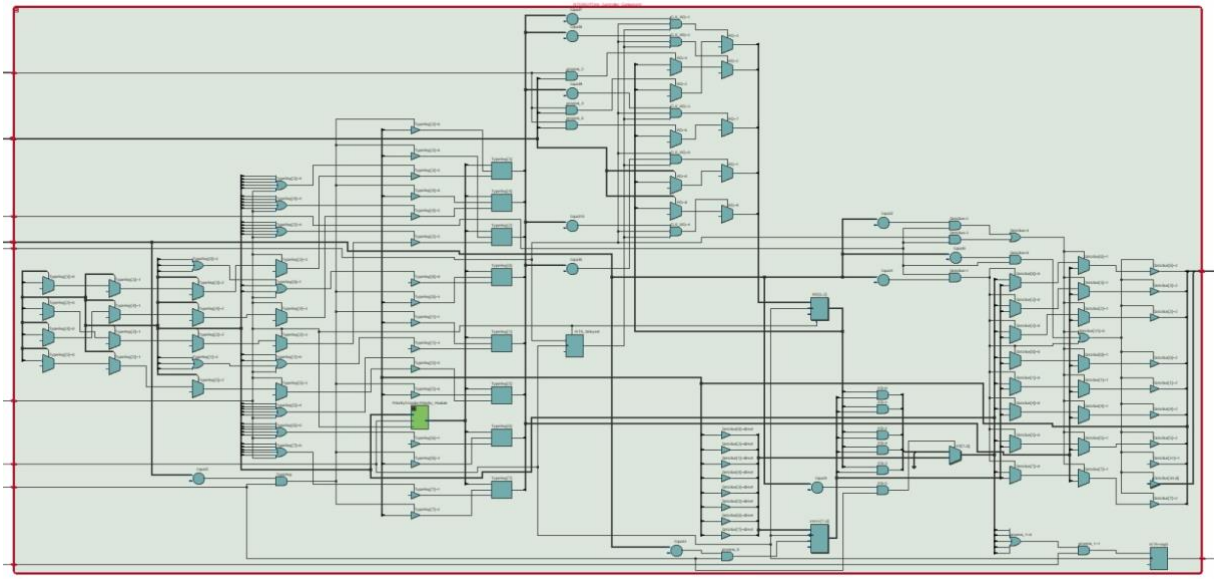
- טבלת ports-

type	size	mode	port
Std_logic	1	in	**MCLK**
Std_logic	1	in	**reset**
Std_logic_vector	8	in	**BTCTL**
Std_logic_vector	32	in	**BTCCR0, BTCCR1**
Std_logic	1	out	**BTIFG**
Std_logic	1	out	**BTOUT**

2.5. INTERRUPT CONTROLLER –

- זהו רכיב המנהל את כל הפסיקות שנתבקשנו ליישם. הפסיקות, שיכולות להגיע מהרכיבים הפריפריאליים (או מרכיבים אחרים בעת הצורך), מתבצעות כך שהקטע קוד "קופץ" לכתובת שונה בזיכרון, בעת פסיקה – ולחזור לאותו מקום בו היה לפני ביצוע הפסיקה.
- פרט לביצוע רכיב זה, היה צורך לתקן את הMIPS ע"מ שיתמוך בפעולה הנ"ל. התמיכה בכך, נעשית באמצעות פקודות jal הקופצת לפסיקה, וjr החוזרת לאותו קטע קוד בסוף פסיקה.

- להלן הדברים שנתבקשנו להראות לאחר יצירת הקבצים הללו :
  - תוצאות RTL Viewer-



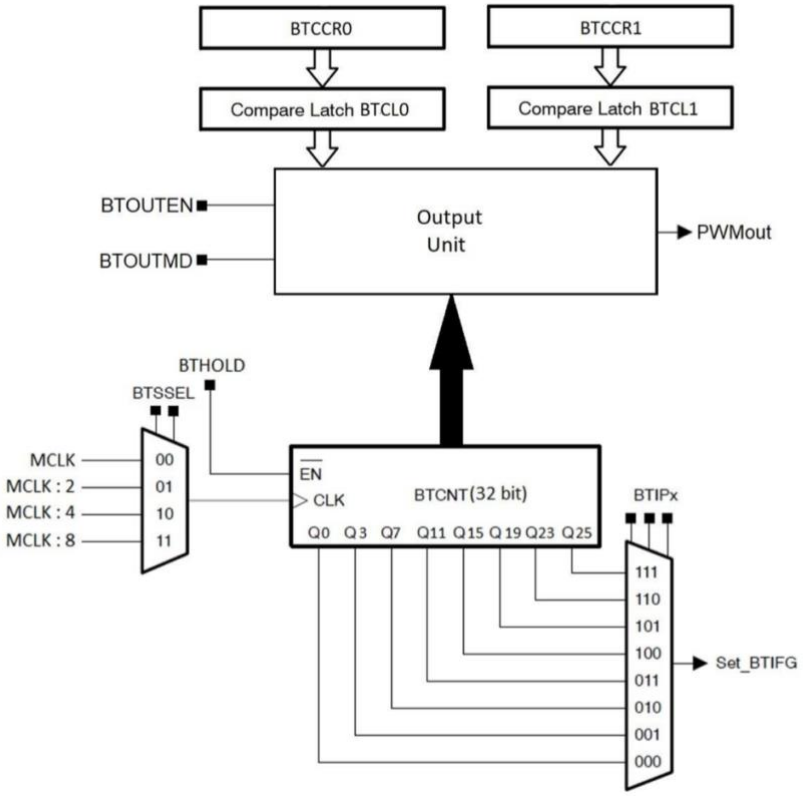
איור 2.5 – RTL Viewerשל הinterrupt controller

  - הסבר לוגי על המודול-

זה להסביר תורה שלמה

  - הסבר גרפי-

להלן ההסבר שלמדנו עליו :



גרף 2.4 – שרטוט רכיב הBasic Timer

- טבלת ports-

type	size	mode	port
Std_logic	1	in	**MCLK**
Std_logic	1	in	**reset**
Std_logic_vector	8	in	**BTCTL**
Std_logic_vector	32	in	**BTCCR0, BTCCR1**
Std_logic	1	out	**BTIFG**
Std_logic	1	out	**BTOUT**

3. מציאת Fmax, Critical Path:

3.1. Fmax –

- להלן התדר המקסימלי:

Slow 1100mV 85C Model Fmax Summary

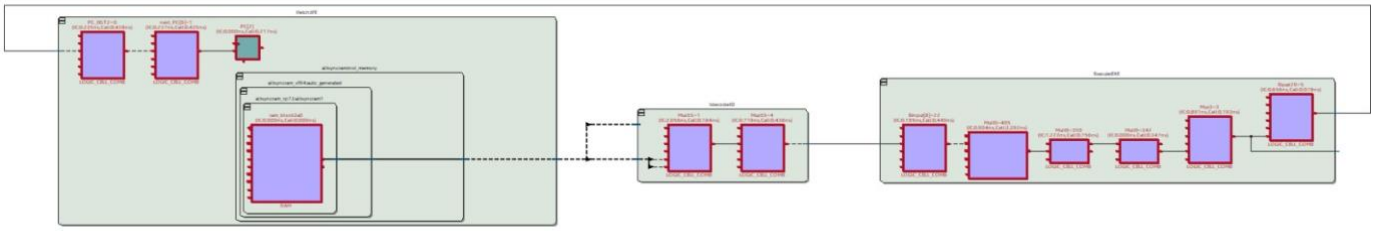
<<Filter>>

	Fmax	Restricted Fmax	Clock Name	Note
1	31.53 MHz	31.53 MHz	pll_clk	
2	92.19 MHz	92.19 MHz	altera_reserved_tck	

טבלה 3.1 – תדר FMAX

3.2. Critical Path –

- להלן הcritical path:

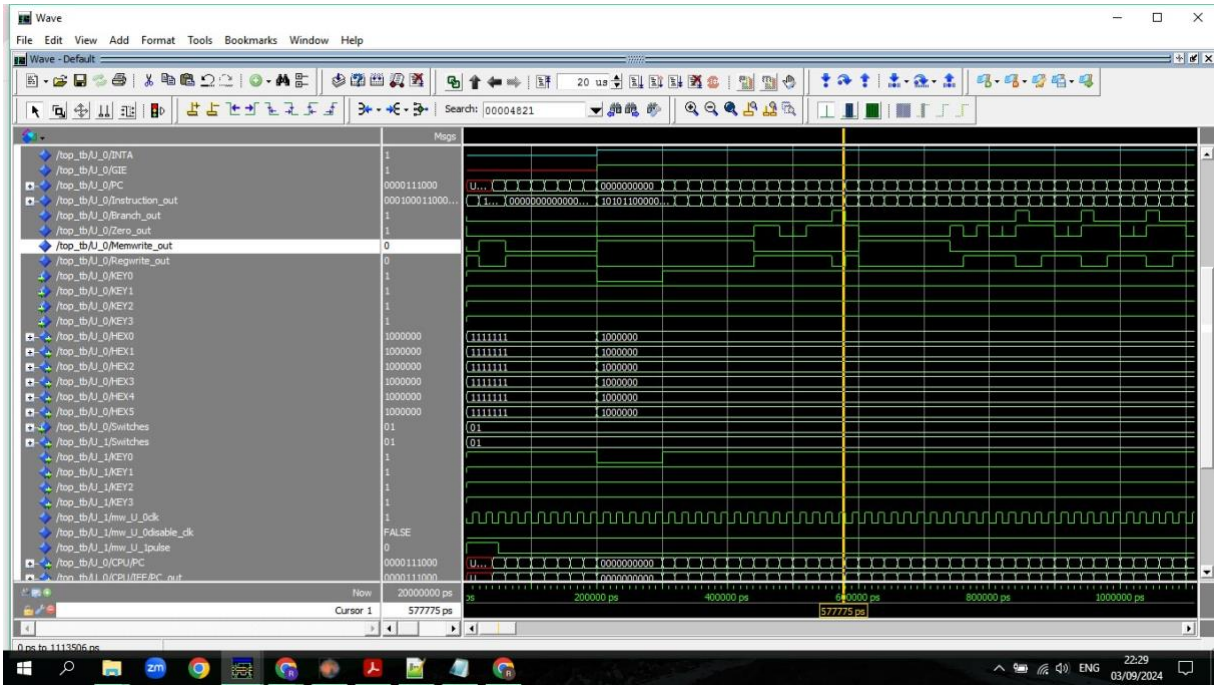


איור 3.2 – critical path

4. תוצאות:

- בדקנו את הרכיבים שלנו גם באמצעות tb כולל של הTOP – של הMCU. בtb, בעזרת הdmemory, המעבד קורא מהזיכרון של הprogram את ההוראות, ובכל פעולת שעון מבצע פעולה אחרת.
- להלן התוצאות –
  - wave:

להלן הwaveform של test1:



  - signal tap:

זהו הsignal tap של test1: