מעבדה 1

מעבדה 1

הקדמה

במעבדה זו נלמדת השפה VHDL המתארת חומרה המשמשת לתיאור של מעגלים ספרתיים. בפרט, יושמה מערכת המבצעת מספר מודלים הכוללים מחבר, מחסר, מודל האחראי על פעולות לוגיות בין זוג וקטורים שיפטרים ועוד. בעזרת פקודות הניתנות לALUFN, קו וקטורי הנכנס למערכת, הלה מיישמת כל אחת מהפעולות הנזכרות מעלה על שני וקטורי כניסה אחרים, Y,X.

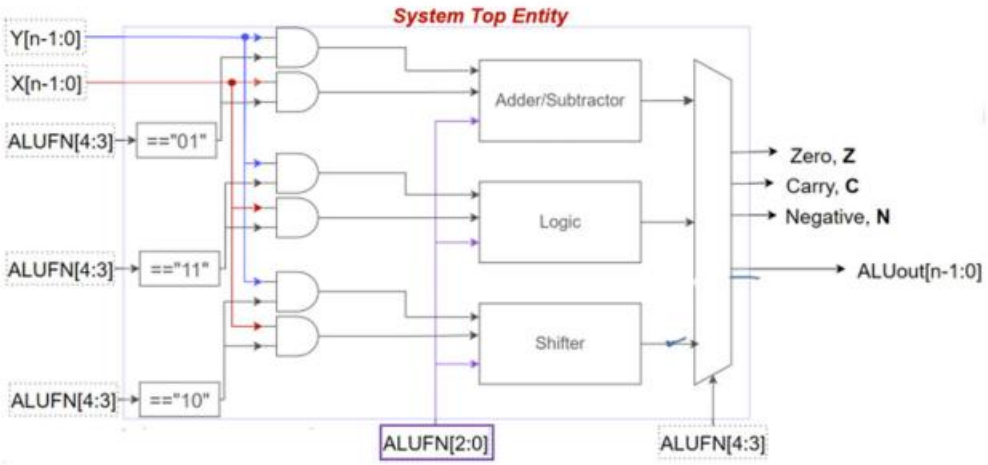


Figure 1 -סכימת המערכת שיושמה

Function Kind	Decimal value	ALUFN	Operation	Note
Arithmetic	8	**01**000	Res=Y+X	
	9	**01**001	Res=Y-X	Used also for compare operation
	10	**01**010	Res=neg(X)	
Shift	16	**10**000	Res=SHL Y,X(k-1 to 0)	Shift Left Y of q≜X(k-1 ..0) times Res=Y(n-1-q…0)#(q@0) $\boldsymbol{When\ k = log_2 n}$
	17	**10**001	Res=SHR Y,X(k-1 to 0)	Shift Right Y of q≜X(k-1 ..0) times Res=(q@0)#Y(n-1...q) $\boldsymbol{When\ k = log_2 n}$
Boolean	24	**11**000	Res=not(Y)	
	25	**11**001	Res=Y or X	
	26	**11**010	Res=Y and X	
	27	**11**011	Res=Y xor X	
	28	**11**100	Res=Y nor X	
	29	**11**101	Res=Y nand X	
	30	**11**111	Res=Y xnor X	

Figure 2 - תיאור מלא לאופן פעולת המערכת

תיאור הקוים הנכנסים למערכת

ישנם שלושה קוים עיקריים אשר נכנסים למערכת:

- וקטור X
- וקטור Y
- ALUFN

Y[n-1:0]

- תיאור: קו כניסה למערכת המייצג וקטור בן nביטים.
- שימוש: וקטור זה מספק את אחד הערכים שישמשו בפעולות האריתמטיות או הלוגיות במודולים הפנימיים של הALU

X[n-1:0]

- תיאור: קו כניסה נוסף למערכת המייצג גם הוא וקטור בן n ביטים.

- שימוש: וקטור זה מספק את הערך השני שישמש בפעולות האריתמטיות או הלוגיות במודולים הפנימיים של הALU.

ALUFN[4:0]

- תיאור: קו כניסה המייצג קוד פקודה בן 5 ביטים המגדיר את הפעולה שה ALU צריכה לבצע.
- שימוש:
  - **ALUFN[4:3]** שני הביטים הגבוהים משמשים לבחירת המודול הפנימי שיבצע את הפעולה.
    - "01" מפעיל את מודול ה Adder/Subtractor
    - "10" מפעיל את מודול ה Shifter
    - "11" מפעיל את מודול ה Logic .
  - **ALUFN[2:0]** שלושת הביטים הנמוכים מגדירים את סוג הפעולה הספציפית בתוך המודול הנבחר. לדוגמה, חיבור, חיסור, הזזה שמאלה או ימינה ופעולות לוגיות.

מודלים

Top

1. **פרמטרים גנריים**:
   - n: גודל וקטור הכניסות והיציאות.
   - k: מספר ביטים לייצוג לוגריתם של n.
   - m: ערך מחושב המבוסס על k.
2. **כניסות**:
   - Y_i, X_i וקטורי כניסות בגודל n.
   - ALUFN_i וקטור כניסה בגודל 5 ביטים המגדיר את הפונקציה האריתמטית/לוגית שתבוצע.
3. **יציאות**:
   - ALUout_o וקטור יציאה בגודל n .
   - Nflag_o, Cflag_o, Zflag_o, Vflag_o דגלי יציאה לציון תוצאות חישוב מיוחדות (שליליות, נשיאה, אפס, וחריגה).

מבנה הארכיטקטורה:

1. **קבועים וסוגים** :

- NUM_OF_MODULES מספר המודולים הפנימיים.
- vector, matrix הגדרות סוגים עבור וקטורים ומטריצות של STD_LOGIC

2. **אותות פנימיים** :

- אותות עבור כניסות ויציאות המודולים הפנימיים.
- אותות עבור דגלי חישוב מיוחדים כמו Vflag_add ו.Vflag_sub

3. **אינסטנציות של תת-מודולים** :

- AdderSub_inst מבצע חיבור/חיסור.
- Shifter_inst מבצע פעולות הזזה.
- Logic_inst מבצע פעולות לוגיות.

4. **היגיון חיווט והפעלה** :

- חיווט כניסות למודולים לפי ערכי ALUFN_i
- קביעת ערכים לדגלים בהתאם לתוצאות החישוב.

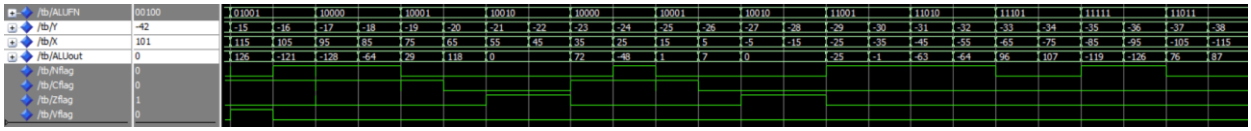
5. **היגיון יציאה** :

- בחירת תוצאת המודול המתאים והעברתה ליציאה ALUout_o
- קביעת ערכים לדגלי היציאה Nflag_o, Zflag_o, Cflag_o, ו- Vflag_o

6. **הערות כלליות**

- כדי לאפשר מינימום אנרגיה בזמן פעולה, רק מצב אחד יכול להיות דולק, וכאשר אף מצב לא דולק ישנם שיערי כניסה שיקבעו כי המערכת כבויה

7. **תוצאות סימולציה**



- אפשר לראות את פעולות המודל בעת כניסות שונות, שהרי הוא עושה פעולות שונות לפי הטבלה בהקדמה.

	ps	delta	/tb/Y	/tb/X	/tb/ALUFN	/tb/ALUout	/tb/Nflag	/tb/Cflag	/tb/Zflag	/tb/Vflag
6	0	+7	11111111	11111111	01000	11111110	1	1	0	0
7	50000	+8	11111110	11110101	01000	11110011	1	1	0	0
8	100000	+6	11111101	11101011	01001	00010010	0	1	0	0
9	150000	+8	11111100	11100001	01001	00011011	0	1	0	0
10	200000	+8	11111011	11010111	01010	00101001	0	0	0	0
11	250000	+6	11111010	11001101	01010	00110011	0	0	0	0
12	300000	+8	11111001	11000011	01000	10111100	1	1	0	0
13	350000	+8	11111000	10111001	01000	10110001	1	1	0	0
14	400000	+8	11110111	10101111	01001	01001000	0	1	0	0
15	450000	+7	11110110	10100101	01001	01010001	0	1	0	0
16	500000	+9	11110101	10011011	01010	01100101	0	0	0	0
17	550000	+6	11110100	10010001	01010	01101111	0	0	0	0
18	600000	+8	11110011	10000111	01000	01111010	0	1	0	1
19	650000	+10	11110010	01111101	01000	01101111	0	1	0	0
20	700000	+9	11110001	01110011	01001	01111110	0	1	0	1
21	750000	+9	11110000	01101001	01001	10000111	1	1	0	0
22	800000	+13	11101111	01011111	10000	10000000	1	1	0	0
23	850000	+11	11101110	01010101	10000	11000000	1	1	0	0
24	900000	+5	11101101	01001011	10001	00011101	0	1	0	0
25	950000	+6	11101100	01000001	10001	01110110	0	0	0	0
26	1000000	+4	11101011	00110111	10010	00000000	0	0	1	0
27	1050000	+1	11101010	00101101	10010	00000000	0	0	1	0
28	1100000	+9	11101001	00100011	10000	01001000	0	1	0	0
29	1150000	+7	11101000	00011001	10000	11010000	1	1	0	0
30	1200000	+5	11100111	00001111	10001	00000001	0	1	0	0
31	1250000	+5	11100110	00000101	10001	00000111	0	0	0	0
32	1300000	+4	11100101	11111011	10010	00000000	0	0	1	0
33	1350000	+1	11100100	11110001	10010	00000000	0	0	1	0
34	1400000	+6	11100011	11100111	11001	11100111	1	0	0	0
35	1450000	+6	11100010	11011101	11001	11111111	1	0	0	0
36	1500000	+6	11100001	11010011	11010	11000001	1	0	0	0
37	1550000	+6	11100000	11001001	11010	11000000	1	0	0	0
38	1600000	+6	11011111	10111111	11101	01100000	0	0	0	0
39	1650000	+6	11011110	10110101	11101	01101011	0	0	0	0
40	1700000	+6	11011101	10101011	11111	10001001	1	0	0	0
41	1750000	+6	11011100	10100001	11111	10000010	1	0	0	0
42	1800000	+6	11011011	10010111	11011	01001100	0	0	0	0
43	1850000	+6	11011010	10001101	11011	01001111	0	0	0	0
44	1900000	+3	11011001	10000011	00100	00000000	0	0	1	0
45	1950000	+1	11011000	01111001	00100	00000000	0	0	1	0
46	2000000	+1	11010111	01101111	00100	00000000	0	0	1	0

	ps	delta	/tb/Y	/tb/X	/tb/ALUFN	/tb/ALUout	/tb/Nflag	/tb/Cflag	/tb/Zflag	/tb/Vflag
6	0	+8	11111111	11111111	01000	11111110	1	1	0	0
7	50000	+8	11111110	11110101	01000	11110011	1	1	0	0
8	100000	+7	11111101	11101011	01001	00010010	0	1	0	0
9	150000	+8	11111100	11100001	01001	00011011	0	1	0	0
10	200000	+8	11111011	11010111	01010	00101001	0	0	0	0
11	250000	+6	11111010	11001101	01010	00110011	0	0	0	0
12	300000	+8	11111001	11000011	01000	10111100	1	1	0	0
13	350000	+8	11111000	10111001	01000	10110001	1	1	0	0
14	400000	+9	11110111	10101111	01001	01001000	0	1	0	0
15	450000	+7	11110110	10100101	01001	01010001	0	1	0	0
16	500000	+9	11110101	10011011	01010	01100101	0	0	0	0
17	550000	+6	11110100	10010001	01010	01101111	0	0	0	0
18	600000	+8	11110011	10000111	01000	01111010	0	1	0	1
19	650000	+10	11110010	01111101	01000	01101111	0	1	0	0
20	700000	+9	11110001	01110011	01001	01111110	0	1	0	1
21	750000	+9	11110000	01101001	01001	10000111	1	1	0	0
22	800000	+9	11101111	01011111	10000	10000000	1	1	0	0
23	850000	+8	11101110	01010101	10000	11000000	1	1	0	0
24	900000	+7	11101101	01001011	10001	00011101	0	1	0	0
25	950000	+8	11101100	01000001	10001	01110110	0	0	0	0
26	1000000	+9	11101011	00110111	10010	00000000	0	0	1	0
27	1050000	+1	11101010	00101101	10010	00000000	0	0	1	0
28	1100000	+9	11101001	00100011	10000	01001000	0	1	0	0
29	1150000	+8	11101000	00011001	10000	11010000	1	1	0	0
30	1200000	+7	11100111	00001111	10001	00000001	0	1	0	0
31	1250000	+7	11100110	00000101	10001	00000111	0	0	0	0
32	1300000	+9	11100101	11111011	10010	00000000	0	0	1	0
33	1350000	+1	11100100	11110001	10010	00000000	0	0	1	0
34	1400000	+5	11100011	11100111	11001	11100111	1	0	0	0
35	1450000	+5	11100010	11011101	11001	11111111	1	0	0	0
36	1500000	+5	11100001	11010011	11010	11000001	1	0	0	0
37	1550000	+5	11100000	11001001	11010	11000000	1	0	0	0
38	1600000	+5	11011111	10111111	11101	01100000	0	0	0	0
39	1650000	+5	11011110	10110101	11101	01101011	0	0	0	0
40	1700000	+5	11011101	10101011	11111	10001001	1	0	0	0
41	1750000	+5	11011100	10100001	11111	10000010	1	0	0	0
42	1800000	+5	11011011	10010111	11011	01001100	0	0	0	0
43	1850000	+5	11011010	10001101	11011	01001111	0	0	0	0
44	1900000	+3	11011001	10000011	00100	00000000	0	0	1	0
45	1950000	+1	11011000	01111001	00100	00000000	0	0	1	0
46	2000000	+1	11010111	01101111	00100	00000000	0	0	1	0

- להלן התוצאות בהשוואה למודל הזהב.

מחבר מחסר ושולל

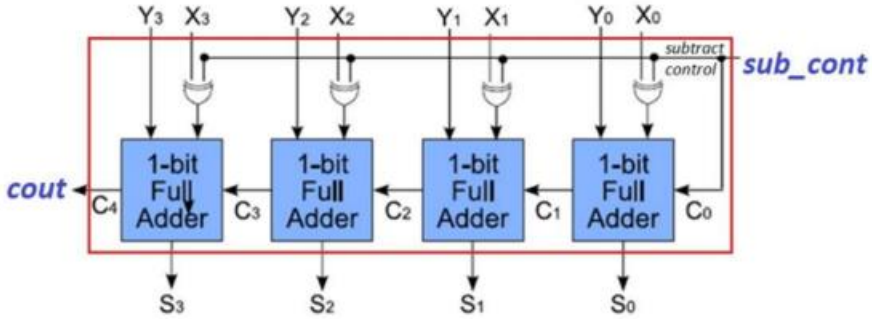


Figure 3- שרטוט של המודל המיושם

1. **פרמטרים גנריים**:
   - n גודל וקטור הכניסות והיציאות, ברירת מחדל היא 8 ביטים.
2. **כניסות**:
   - x_adderSub_in, y_adderSub_in וקטורי כניסות בגודל n ביטים.
   - ALUFN וקטור כניסה בגודל 3 ביטים המגדיר את הפעולה שתבוצע (חיבור או חיסור).
3. **יציאות**:

- adderSub_cout יציאת נשיאה.(Carry Out)
- adderSub_out וקטור יציאה בגודל n ביטים המייצג את תוצאת החישוב.

מבנה הארכיטקטורה:

1. **קבועים וסוגים**:
   - הגדרת סוגים עבור וקטורים של STD_LOGIC
   - אותות פנימיים כגון reg, x_adderSub_xor, y_adderSub_gated ו sub_control.
2. **אותות פנימיים**:
   - sub_control אות המשמש לשליטה האם מתבצע חיסור (ערך '1') או חיבור (ערך '0').
   - x_adderSub_xor וקטור שיכול להיות מושלך עם sub_control עבור פעולת חיסור.
   - y_adderSub_gated וקטור שמאפשר שליטה על הערך של y_adderSub_in על היות 0 במקרים של פעולות לא חוקיות.
3. **תהליך לוגי**:
   - קביעת ערך sub_control בהתאם ל ALUFN
   - קביעת ערך y_adderSub_gated בהתאם לערך ALUFN
   - הפקת וקטור x_adderSub_xor עם אפשרות להיפוך ביטים עבור חיסור.
4. **אינסטנציות של סכום מלא (FA)**
   - **first** אינסטנציה של סכום מלא לביט הראשון עם כניסת נשיאה ראשונית (sub_control)
   - **Rest** אינסטנציות של סכום מלא עבור כל הביטים הנותרים, מקשרים כל ביט עם הנשיאה מהביט הקודם.
5. **היגיון יציאה**:
   - יציאת נשיאה (adderSub_cout) נקבעת מהביט האחרון של הוקטור reg .
6. **תוצאות:**

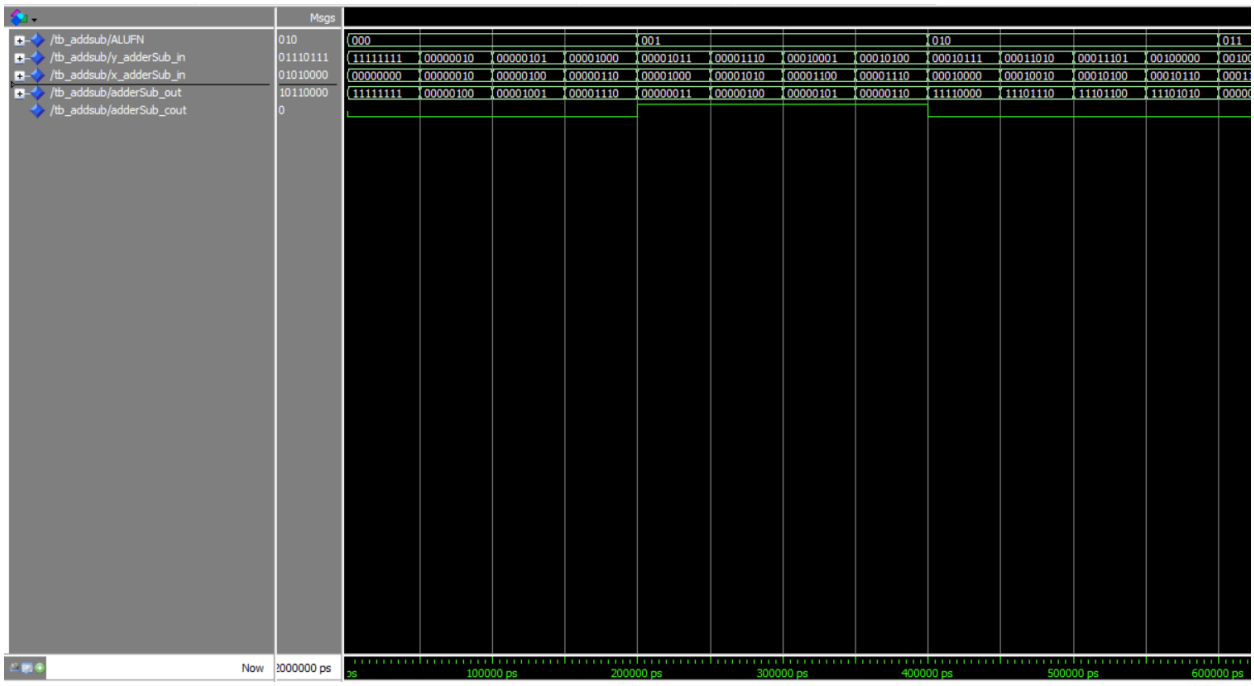


Figure 4 - אינפוטים תקינים של המודל

Figure 5 - אינפוטים לא תקינים של המודל

שיפטר

מבנה כללי:

המודול Shifter מבצע פעולות של הזזה שמאלה (Shift Left) והזזה ימינה (Shift Right) על וקטור כניסה בגודל n ביטים, כאשר כמות ההזזה מוגדרת על ידי וקטור כניסה נוסף בגודל k ביטים.

רכיבי Shifter

1. **פרמטרים גנריים**:
   - n גודל וקטור הכניסות והיציאות, ברירת מחדל היא 8 ביטים.
   - k מספר ביטים לייצוג כמות ההזזה ,((log2(n) ברירת מחדל היא 3 ביטים.
2. **כניסות**:
   - y_Shifter_in וקטור כניסה בגודל nביטים המייצג את הנתונים שיש להזיז.
   - x_Shifter_in וקטור כניסה בגודל kביטים המייצג את כמות ההזזה.
   - ALUFN וקטור כניסה בגודל 3 ביטים המגדיר את סוג הפעולה שתבוצע (הזזה שמאלה או ימינה).
3. **יציאות**:
   - Shifter_cout ביט יציאה המייצג את הביט ש"זז החוצה" במהלך ההזזה.
   - Shifter_out וקטור יציאה בגודל nביטים המייצג את תוצאת ההזזה.

מבנה הארכיטקטורה:

1. **קבועים וסוגים**:
   - הגדרת סוגים עבור וקטורים של std_logic_vector
   - הגדרת מטריצה עבור הוקטורים של הביטים המוזזים שמאלה וימינה.
2. **אותות פנימיים**:
   - Shifter_cout_vector_left, Shifter_cout_vector_rightוקטורים לאיסוף הביטים שנזזו החוצה במהלך ההזזה.
   - row_left, row_right מטריצות עבור התהליך ההזזה שמאלה וימינה בהתאמה.
3. **תהליך לוגי**:
   - **הזזה שמאלה**: כל ביט בוקטור y_Shifter_inמוזז שמאלה, והביט שזז נאסף בוקטור Shifter_cout_vector_left.
   - **הזזה ימינה**: כל ביט בוקטור y_Shifter_inמוזז ימינה, והביט שזז נאסף בוקטור Shifter_cout_vector_right.

4. **הפקת תוצאה ויציאה** :

- **Shifter_out** התוצאה הסופית של ההזזה (שמאלה או ימינה) נקבעת לפי ערך x_Shifter_in ו.ALUFN -
- **Shifter_cout** הביט ש"זז החוצה" במהלך ההזזה נקבע לפי ערך x_Shifter_in ו.ALUFN -

5. **הערות**

- המודל מיושם באופן דומה לסרטוני ההדרכה, בעזרת מטריצת עזר לתוצאה ווקטור עזר לקרי.

6. **תוצאות**

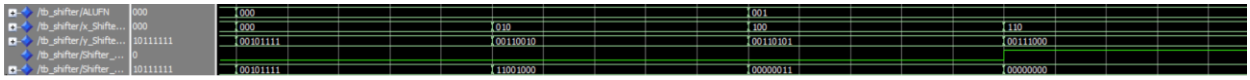


Figure 6 - תוצאות של אינפוטים תקינים המראות גם את הקרי

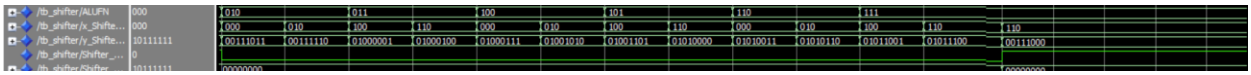


Figure 7 - תוצאות של אינפוטים לא תקינים

Logic

מבנה כללי:

המודול Logicמבצע פעולות לוגיות על שני וקטורי כניסה של nביטים, כאשר סוג הפעולה נקבע על ידי וקטור ALUFN . המודול מסוגל לבצע מגוון פעולות לוגיות כמו NOT, OR, AND, XOR, NOR, NAND, ו.XNOR-

רכיבי Logic

1. **פרמטרים גנריים** :

- n גודל וקטור הכניסות והיציאות, ברירת מחדל היא 8 ביטים.
- OP מספר הפעולות הלוגיות הנתמכות פלוס אחת (עבור תוצאה של אפסים), ברירת מחדל היא 8.

2. **כניסות** :

- x_logic, y_logic וקטורי כניסה בגודל nביטים עליהם מתבצעות הפעולות הלוגיות.
- ALUFN וקטור כניסה בגודל 3 ביטים המגדיר את הפעולה הלוגית שתבוצע.

3. **יציאות** :

- Logic_out וקטור יציאה בגודל nביטים המייצג את תוצאת הפעולה הלוגית.

מבנה הארכיטקטורה:

1. **סוגים ואותות פנימיים** :

- result_matrix מטריצה המייצגת את כל התוצאות האפשריות עבור כל הפעולות הלוגיות הנתמכות.
- output_matrix אות פנימי של המטריצה המכילה את כל תוצאות הפעולות הלוגיות.

2. **חישוב תוצאות** :

- לכל ערך אפשרי של ALUFN(מ-000 עד 111) נקבעת הפעולה הלוגית המתאימה :
  - output_matrix(0)מקבל את תוצאת NOT על y_logic
  - output_matrix(1)מקבל את תוצאת OR בין x_logic ל - y_logic
  - output_matrix(2)מקבל את תוצאת AND בין x_logic ל y_logic -
  - output_matrix(3)מקבל את תוצאת XOR בין x_logic לy_logic -
  - output_matrix(4)מקבל את תוצאת NOR בין x_logic לy_logic
  - output_matrix(5)מקבל את תוצאת NAND בין x_logic לy_logic
  - output_matrix(6)מקבל את תוצאת XNOR בין x_logic לy_logic
  - output_matrix(7) מקבל את וקטור האפסים (ברירת מחדל לפקודות לא חוקיות).

3. **הפקת תוצאה ויציאה** :

- בהתאם לערך של ,ALUFNהמודול בוחר את התוצאה המתאימה מהמטריצה ומעביר אותה ליציאה Logic_out

4. **תוצאות**

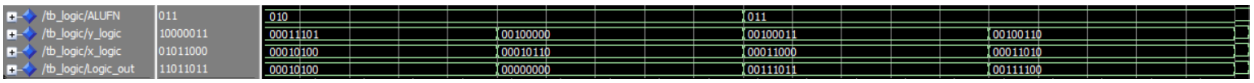


Figure 8 - פעולות לוגיות של and, xor

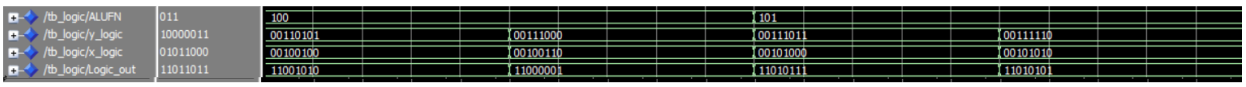


Figure 9 - פעולות לוגיות של nor, nand

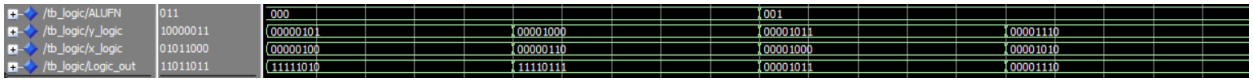


Figure 10 - פעולה לוגית not

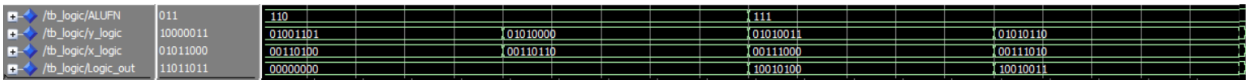


Figure 11 - אינפוטים אסורים עבור nor

סיכום כללי

המבנה המשולב של המודולים מאפשר מימוש יחידה אריתמטית-לוגית (ALU) גמישה ורבת-יכולות, אשר מתאימה לשימוש במערכות דיגיטליות כמו מעבדים ומערכות. כל אחד מהמודולים מבצע פעולה ספציפית בצורה יעילה, והשילוב ביניהם מאפשר לממש פעולות מורכבות בהתאם לצרכים שונים.