

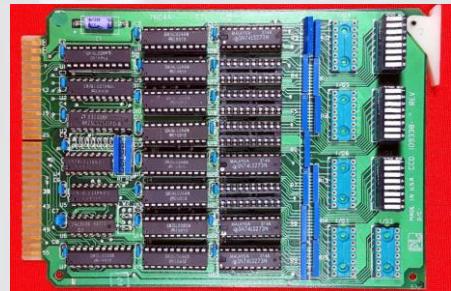
VHDL

אבי חיים

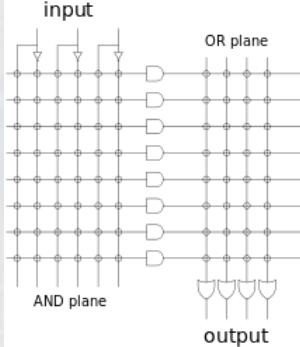


כל הזכויות שמורות

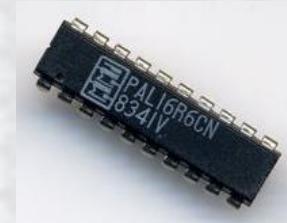
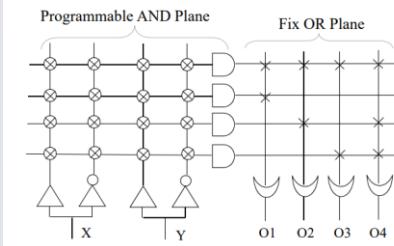
התקנות וההיסטוריה - רכיבים דיגיטליים



Programmable logic array (PLA)



Programmable Array Logic (PAL)



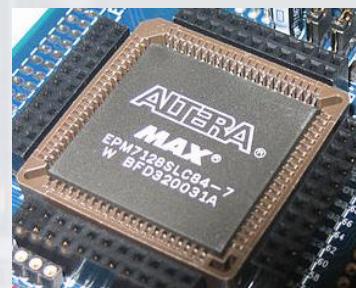
בשנות 78 , רכיב עם תכונות מישור AND



Field-Programmable Gate Array (FPGA)



Complex programmable logic device (CPLD)



Generic array logic (GAL)



בשנות 1985 חברת Lattice Semiconductor המציאה את הרכיב. דומה ל-PAL אבל ניתן למחיקה וצריבה

בעל קיבולת ענקית, מבוסס על זיכרון look-up tables (LUTs) היום מגיע ל מהירות 25Gbps, عشرות אלפי מגה תא זיכרון

בעל קיבולת גדולה, מכיל מספר גדול של PAL

CPLD and FPGA ? למה ?

- שימושים במיוחד בתחוםים הדורשים עיבוד מקבילי, וניתן לבצע בהם כמות גדולה יחסית של פעולות וחישובים במקביל.
- לשיע למעבד הראשי ביצוע פעולות מסוימות, שאינו יכול לבצע בגל מהירות או שהמעבד עסוק ביצוע פעולה.

מגבליות

- סיבוכיות בתכנון לעמודת הקלות יחסית בתכנון עם מעבד ותוכנה רגילה.
- זמן הפיתוח וההרצה הארוך יחסית בהשוואה לתוכנה רגילה.

Field-Programmable Gate Array(FPGA)

- מבוסס על זיכרון טבלאות חיפוש (Look Up Tables)
- لكن תכונות מהיר וניתן לתוכנת מרחוק.
- רכיבים בעלי נפח גדול מאוד.
- שלב התוכנו והפיתוח של המודול האלקטרוני פשוט יותר.

השוואה בין הרכיבים מבחינה קיבולת:

	SPLDs	CPLDs	FPGAs
Equivalent gates	0 ~ 200	200 ~ 12,000	1000 ~ 1,000,000

ההיסטוריה של השפה

VHDL

VHSIC Hardware Description Language

VHSIC- Very High Speed Integrated Circuit

בשנת 1981 החלו לייצר את השפה לתיאור חומרה ספרתית

בשנת 1985 הושלם התיאור הסופי על ידי חברות:

IBM, Texas Instruments, Intermetrics

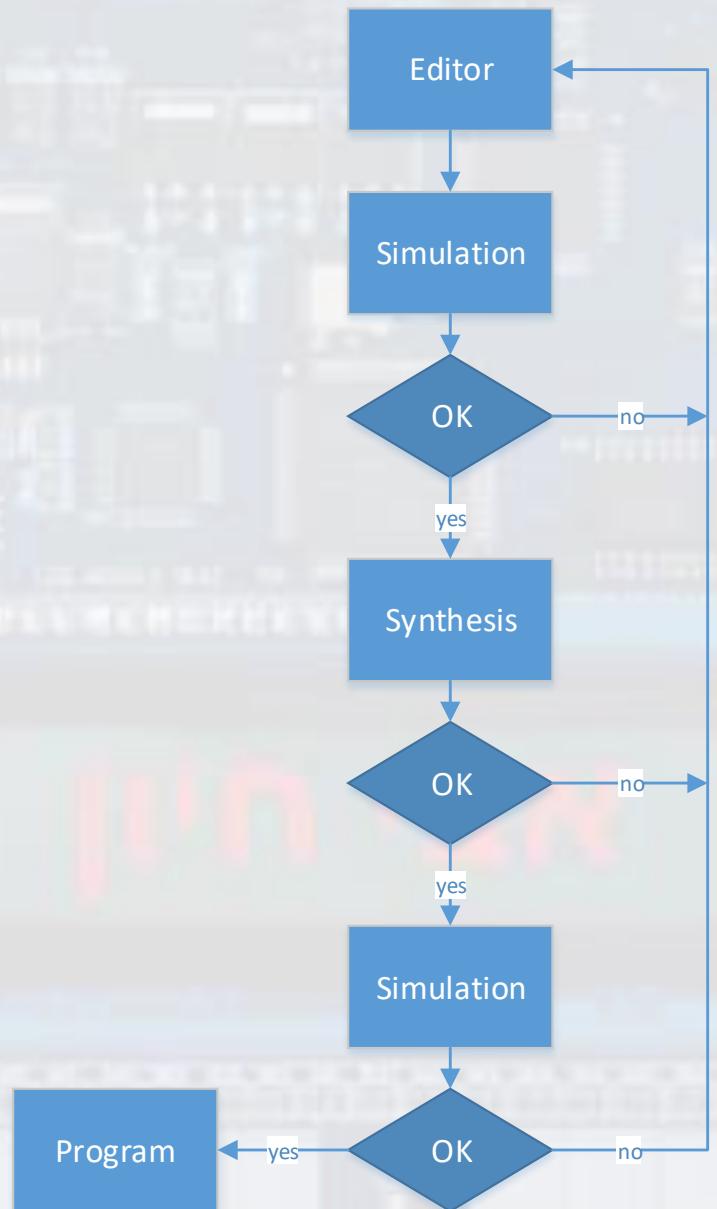
בשנת 1987 נוצר התקן IEEE 1076

בשנים : 1993, 2002, 2008 היו עדכונים

שפות תיאור חומרה השימושות בתעשייה

- **ABEL** – כיומ של חברת Xilinx , שפה למטרת סינטזה בלבד
- **AHDL** – של חברת ALTERA , שפה למטרת סינטזה בלבד.
- **Verilog** – שפה לSimulation וסינטזה, פחות נוקשה אבל עם סיכוי לביעות בתכנון החומרה.
- **VHDL** - שפה לSimulation וסינטזה, בעלת חוקים נוקשים אבל עם סיכוי קטן יותר לביעות בתכנון החומרה.

מהלך התכנון – Design Flows



סימולציה באמצעות תוכנות כמו :
Active HDL, MODELSIM

או סימולציה בסיסית באמצעות כלי סינטזה כמו
ALTERA של חברת QUARTUS

סינטזה וצריבה באמצעות כלי סינטזה כמו:
ALTERA של חברת QUARTUS

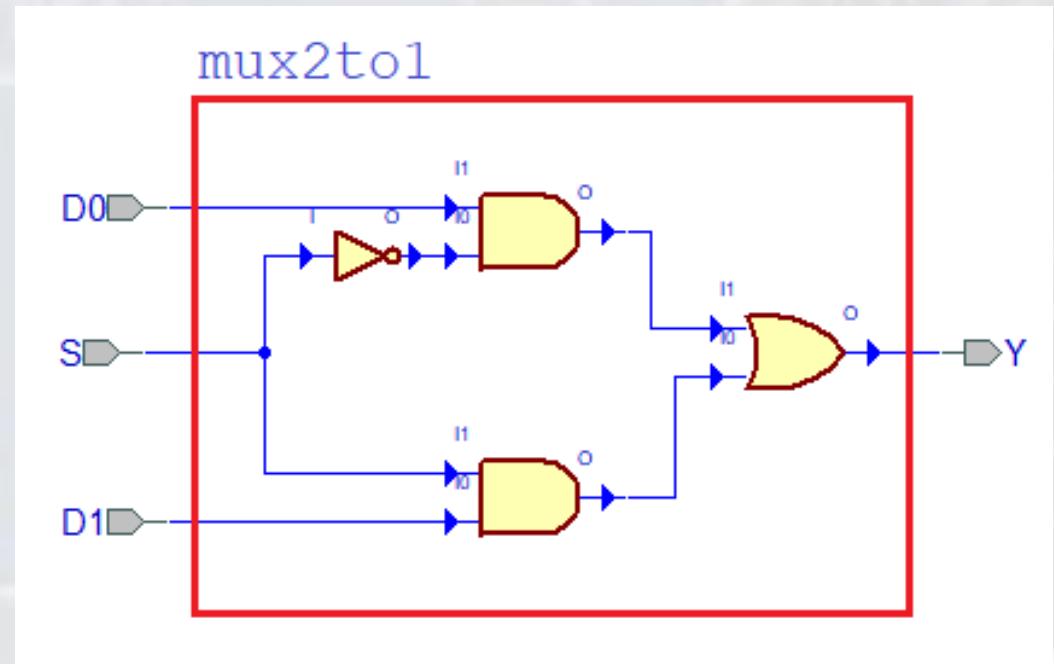
דוגמה ראשונה לכתיבה קוד ב-VHDL

```

entity mux2tol is
  port(D0,D1:in bit;
       S: in bit;
       Y:out bit);
end mux2tol;

architecture arc_mux of mux2tol is
begin
  Y<= (S and D1) or (not S and D0);
end arc_mux;

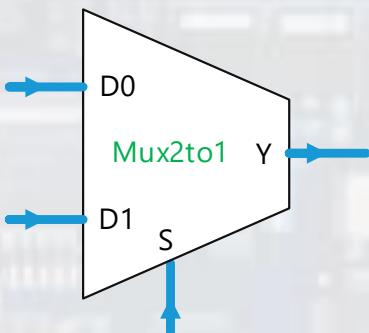
```



- ישות מתאר את המערכת
- מבחר תפקוד המערכת
- מתאר את הדקי המערכת וcieionם
- port

הערה: ב-VHDL אין חשיבות לאותיות קטנות או גדולות

תיאור ה-MUX באופן התנהגותי, עם פקודות שנראות בהמשך



```
entity mux2to1 is
    port(D0,D1:in bit;
          S: in bit;
          Y:out bit);
end mux2to1;

architecture arc_mux of mux2to1 is
begin
    Y<= D1 when S='1' else D0;
end arc_mux;
```

```
entity mux2to1 is
    port(D0,D1:in bit;
          S: in bit;
          Y:out bit);
end mux2to1;

architecture arc_mux of mux2to1 is
begin
    with S select
        Y<= D1 when '1' ,
                    D0 when others ;
end arc_mux;
```

Operators

Miscellanies operators	**	abs	not	
Multiplying operators	* /	Mod	rem	
Signed operator	+	-		
Adding operator	+	-	&	
Shift operator	sll	srl	sla	sra
	rol	ror		
Relational operation	=	/=	<	<=
	>	>=		
Logical operator	and	or	nand	
	nor	xor	xnor	

LOGICAL OPERATORS

Operator	Operand Type פועל על	Result Type מחזיר
not , or , and , nor , nand , xor , xnor	boolean, bit[_vector], std_logic[_vector]	same type

Relational Operators

Operator	Operation	Operand Type	Result Type
=	equality	any type	Boolean
/=	Inequality	any type	Boolean
<	less than	any scalar type	Boolean
<=	less than or equal to	any scalar type	Boolean
>	greater than	any scalar type	Boolean
>=	greater than or equal to	any scalar type	Boolean

Arithmetic Operators

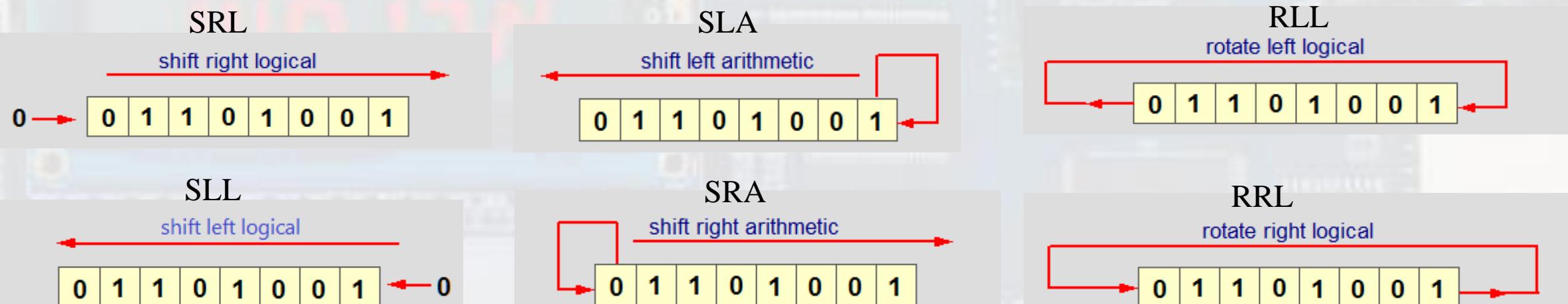
Operator	Operation	Operand Type	Result Type
<code>**</code>	exponential	numeric <code>**</code> integer	numeric
<code>abs</code>	absolute value	numeric	numeric
<code>*</code>	multiplication	numeric <code>*</code> numeric	numeric
<code>/</code>	division	numeric <code>*</code> numeric	numeric
<code>mod</code>	modulus	integer <code>mod</code> integer	integer
<code>rem</code>	remainder	integer <code>rem</code> integer	integer
<code>+</code>	unary plus	<code>+</code> numeric	numeric
<code>-</code>	unary minus	<code>-</code> numeric	numeric
<code>+</code>	addition	numeric <code>+</code> numeric	numeric
<code>-</code>	subtraction	numeric <code>-</code> numeric	numeric

MISCELLANEOUS OPERATORS

Operator	Operation	Operand Type	Result Type
<code>&</code>	concatenation	array or element <code>&</code> array or element	array

Shift Operators

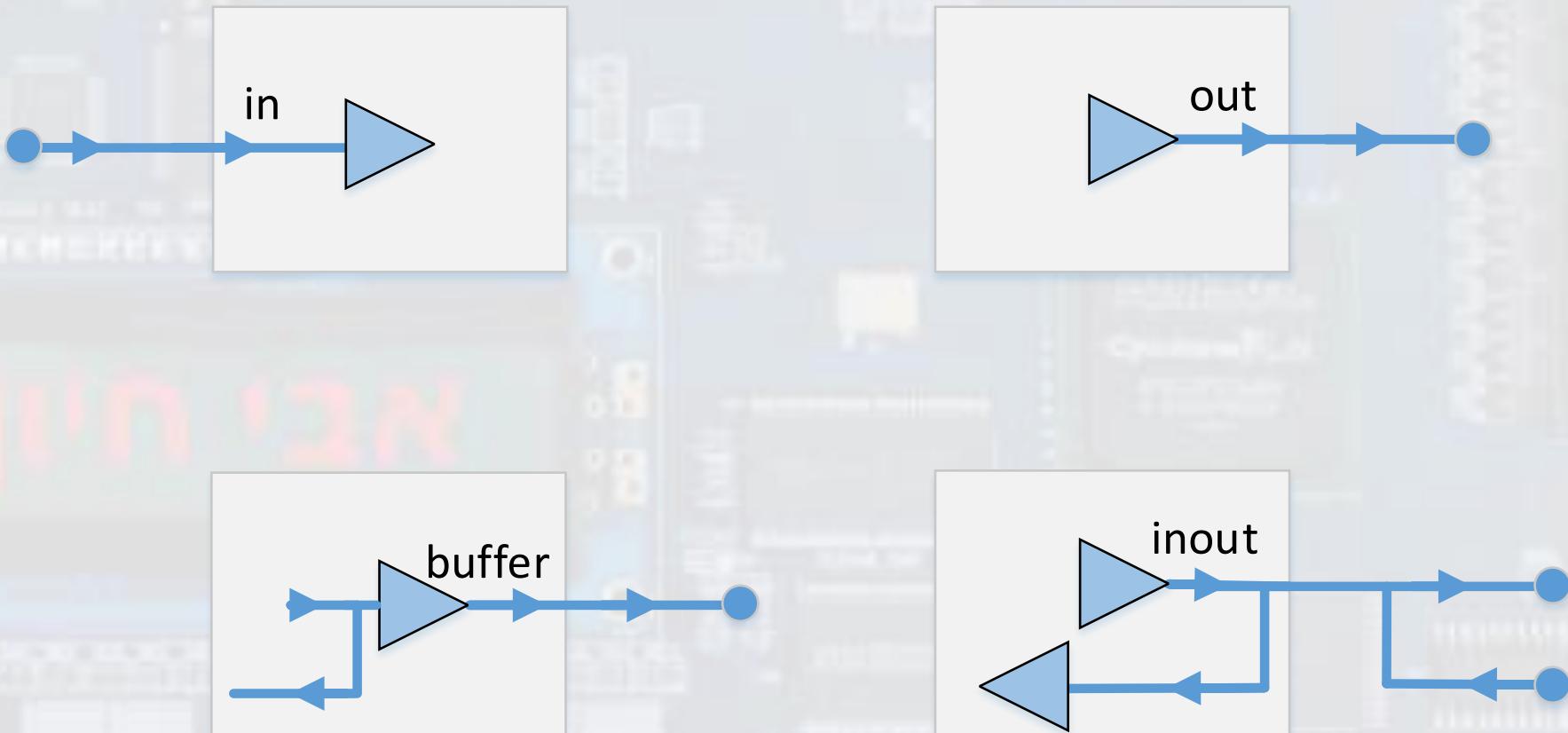
Operator	Operation	Operand Type	Result Type
sll	shift left logical	logical array sll integer	same logical array
srl	shift right logical	logical array srl integer	same logical array
sla	shift left arithmetic	logical array sla integer	same logical array
sra	shift right arithmetic	logical array sra integer	same logical array
rol	rotate left	logical array rol integer	same logical array
ror	rotate right	logical array ror integer	same logical array



חלק מהאופרטורים הנתמכים על ידי הסינთזה

Operator type		Supported synthesizable
Logical	NOT , AND , OR , NAND , NOR , XOR , XNOR	BIT , BIT_VECTOR ,BOOLEAN , STD_LOGIC, STD_LOGIC_VECTOR
Arithmetic	+ , - , * , / , ** , ABS , REM , MOD	Integer , STD_LOGIC_VECTOR (with package std_vector_(un)signed, numeric_std_unsigned)
Comparison	= , /= , > , < , >= , <=	BIT_VECTOR ,BOOLEAN , STD_LOGIC_VECTOR Integer , character , string
Shift	SLL, SRL, SLA, SRA, ROL, ROR	BIT_VECTOR , STD_LOGIC_VECTOR
Concatenation	& (", " , ,)	BIT_VECTOR , STD_LOGIC_VECTOR, string

כיווי אוטות ב- PORT



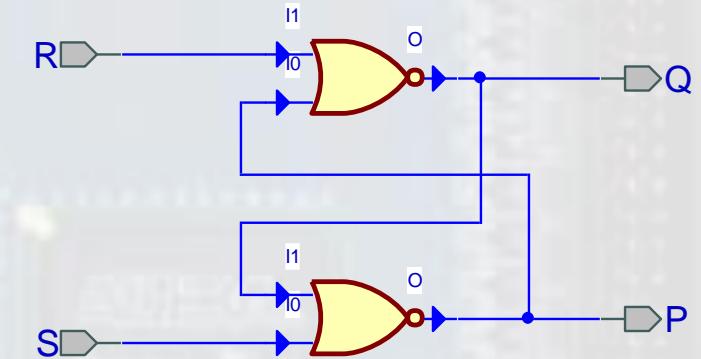
בעה בקריאה מוצא out

```

entity SR_FF is
  port( S, R: in bit;
        Q, P: out bit);
end SR_FF;

architecture arc_SRFF of SR_FF is
begin
  Q <= R nor P;
  P <= S nor Q;
end arc_SRFF;

```



נקבל שגיאה בקומpileציה .

Cannot read output : "P"
Cannot read output : "Q"

מוצא out, לא ניתן לקרוא ולקן הוא לא יכול להופיע בהשמה מצד ימין

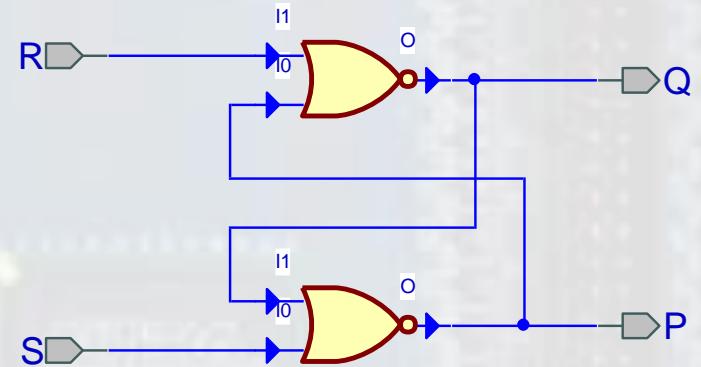
פתרון אפשרי 1 - שימוש בМОץא buffer

```

entity SR_FF is
  port( S, R: in bit;
        Q, P: buffer bit);
end SR_FF;

architecture arc_SRFF of SR_FF is
begin
  Q <= R nor P;
  P <= S nor Q;
end arc_SRFF;

```



קריאה של מוצא מותרת ביציאה
מסוג Buffer ולא out

שימוש ביציאה מסוג buffer מאפשר לקרוא את היציאה והמוצא יכול להיות מצד ימין של ההשמה.

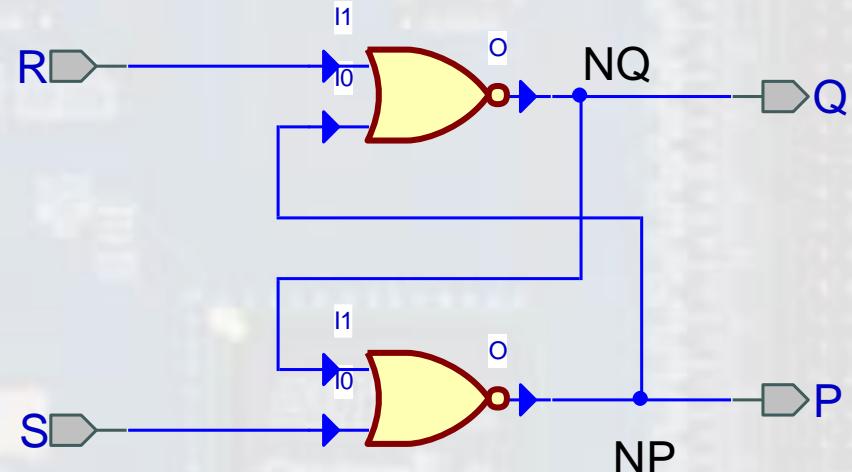
פתרון אפשרי 2 - שימוש באות signal

```

entity SR_FF is
    port( S, R: in bit;
          Q, P: out bit);
end SR_FF;

architecture arc_SRFF of SR_FF is
    signal NQ,NP: bit;
begin
    NQ <= R nor NP;
    NP <= S nor NQ;
    Q <= NQ;
    P <= NP;
end arc_SRFF;

```



Signal נתן לקריאה וכ כתיבה

שימוש ב-signals פנימי מאפשר לחבר בין יחידות שונות בתוך הישות.

Signal מוגדר בין begin וarchitecture

סוגי מידע

סוג bit

מקבל שני ערכים עם סימני בתו' גרש '0' או '1'

דוגמא - `dataBIT : <='0';`

סוג Boolean

מקבל ערכים false או true , כל הסינטזה ממירות ערכים אלה ל - '0' ו- '1'

דוגמא - `data <= true;`

דוגמא עם bit

```
entity big is
    port(a,b:in bit;
        f: out boolean);
end;
```

```
architecture arc_big of big is
begin
    f<= a>b ;
end;
```

דוגמא עם boolean

```
entity big is
    port(a,b:in bit;
        f: out boolean);
end;
```

```
architecture arc_big of big is
begin
    f<= a>b ;
end;
```

סוג character

מקבל ערכים בקוד אסקי , לא ניתן לבצע עליו פועלות לוגיות או אРИתמטיות.

דוגמה – `data <= 'A' ;`

סוג bit_vector

סוג זה, הוא מערך של ביטים

- אפשר לבצע עם סוג זה פועלות לוגיות ולא פועלות אРИתמיות, אלא אם כן תהיה ספריה מתאימה.
- השמה יכולה להיות רק באותו גודל של וקטור.

czherha על אותו מסוג זה לדוגמה:

Signal data1 : bit_vector (7 downto 0) ; -- Bit שמאלית (7) הוא MSB



Signal data2 : bit_vector(0 to 7); -- Bit שמאלית (0) הוא MSB

ניתן לבצע השמה על ביטים בודדים או על חלקים מהווקטור.

ברירת המחדל היא בינהארית עם תווים גרשיים, אבל ניתן לבצע השמה בסיסיים שונים.

איפס ויקטור לדוגמה `Y<=(others => '0');` או בצורה מקוצרת `Y<="00000000";`

את ההשמה הבאה `"01000010"`=> ניתן לבצע במספר דרכים:

Y<=b"01000010";

השמה בבינארי --

Y<=x“42”;

-- השמה בהקסה דצימלי

Y<=o“102”;

השמה באוקטלי --

- השמה על סיביות בודדות:

`Y(7)<='0'; Y(6)<='1'; Y(5)<='0'; Y(4)<='0'; Y(3)<='0'; Y(2)<='0'; Y(1)<='1'; Y(0)<='0';`

- צורה מקוצרת על ידי קבוצה בסוגרים:

`Y<=('0' , '1' , '0' , '0' , '0' , '1' , '0');`

- על ידי הצבה לפי מיקום (הסדר לא חשוב):

`Y<=(7 => '0' , 6 => '1' , 5 => '0' , 4 => '0' , 3 => '0' , 2 => '0' , 1 =>'1' , 0 =>'0');`

- בצורה מקוצרת : (אופרטור | - פועלות או)

`Y<=(6 => '1' , 1 =>'1' , others => '0');`

`Y<=(6 | 1 => '1' , others => '0');`

- בחלוקת

`Y (7 downto 4) <="0100" ;`

`Y (3 downto 0) <="0010" ;`

- שילוב אפשרויות

`Y<=(7 | 0 => '0' , 6 | 1 => '1' , 2 to 3 => '0' ,5 downto 4 => '0');`

השמה עם פועלות האופרטור & (שרשור- concatenation)

נבעט את ההשמה `Y <= "01000010";` באמצעות שרשור
דוגמאות:

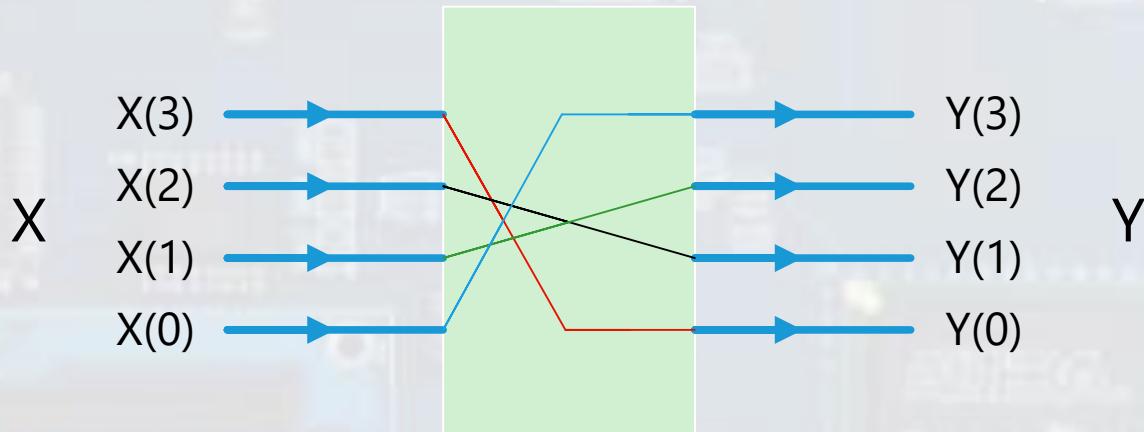
`Y <='0' & '1' & '0' & '0' & '0' & '0' & '1' & '0';`

`Y <='0' & '1' & "0000" & '1' & '0';`

`Y <= "0100" & "001" & '0' ;`

`Y <= x"4" & o"1" & '0' ;`

דוגמה – פועלות על וקטור



$Y(3) \leq X(0); \quad Y(2) \leq X(1); \quad Y(1) \leq X(2); \quad Y(0) \leq X(3);$

סיביות בודדות

$Y \leq (X(0), X(1), X(2), X(3));$

קבוצה

$Y \leq X(0) \& X(1) \& X(2) \& X(3);$

שרשור

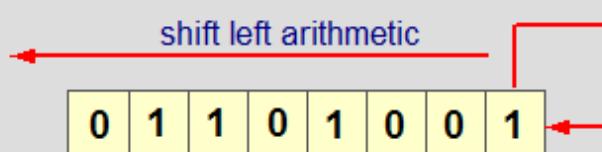
דוגמה פועלות הזרה על וקטור באמצעות שרשור

SRL



$$A \Leftarrow '0' \& B(7 \text{ DOWNTO } 1);$$

SLA



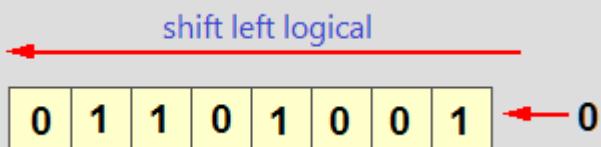
$$A \Leftarrow B(6 \text{ DOWNTO } 0) \& B(0);$$

RLL



$$A \Leftarrow B(6 \text{ DOWNTO } 0) \& B(7);$$

SLL



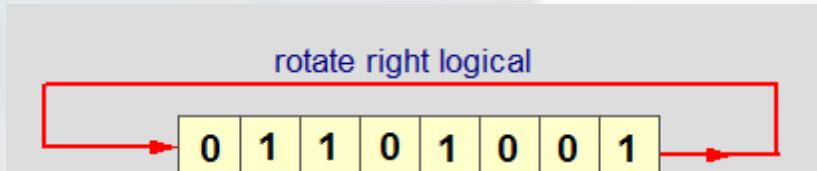
$$A \Leftarrow B(6 \text{ DOWNTO } 0) \& '0';$$

SRA



$$A \Leftarrow B(7) \& B(7 \text{ DOWNTO } 1);$$

RRL



$$A \Leftarrow B(0) \& B(7 \text{ DOWNTO } 1);$$

טוֹג string

מערך של character

הצירה על אות מסווג זה לדוגמה:

Signal message : string(1 to 10); -- **טו 1 תחלתי**

הטו השמאלי מתחילה תמיד מ-
ז<= “Ava Nagila”

סוג integer

אם לא מוגדר הגודל, כל הsynthesizer יקצו לסוג זה, 32 סיביות.

- אפשר לבצע עם סוג זה פעולות אРИתמטיות והשווואה.
- השמה או פעולה אРИתמטית יכולה להיות רק באותן גודל.
- כל הSYMBOLICA יתיחסו לפי הגדרת הגודל ואילו כל הsynthesizer יקצו לפי מספר הסיביות.

דוגמאות

Signal val1 : integer range 0 to 9 ;

כל הsynthesizer יקצו לאות 4 סיביות וכאן המשתנה יכול לקבל ערך 0 עד 15

Signal val1 : integer range -5 to 13 ;

כל הsynthesizer יקצו לאות 5 סיביות בשיטת המשלים ל-2 וכאן המשתנה מקבל ערכים מ-(-16) עד 15

מידע מוגן std_logic

- bit Std_logic מרחיב את ה-bit
- bit_vector Std_logic_vector מרחיב את bit_vector
- כדי להשתמש במידע זה, יש צורך להשתמש בחבילה:

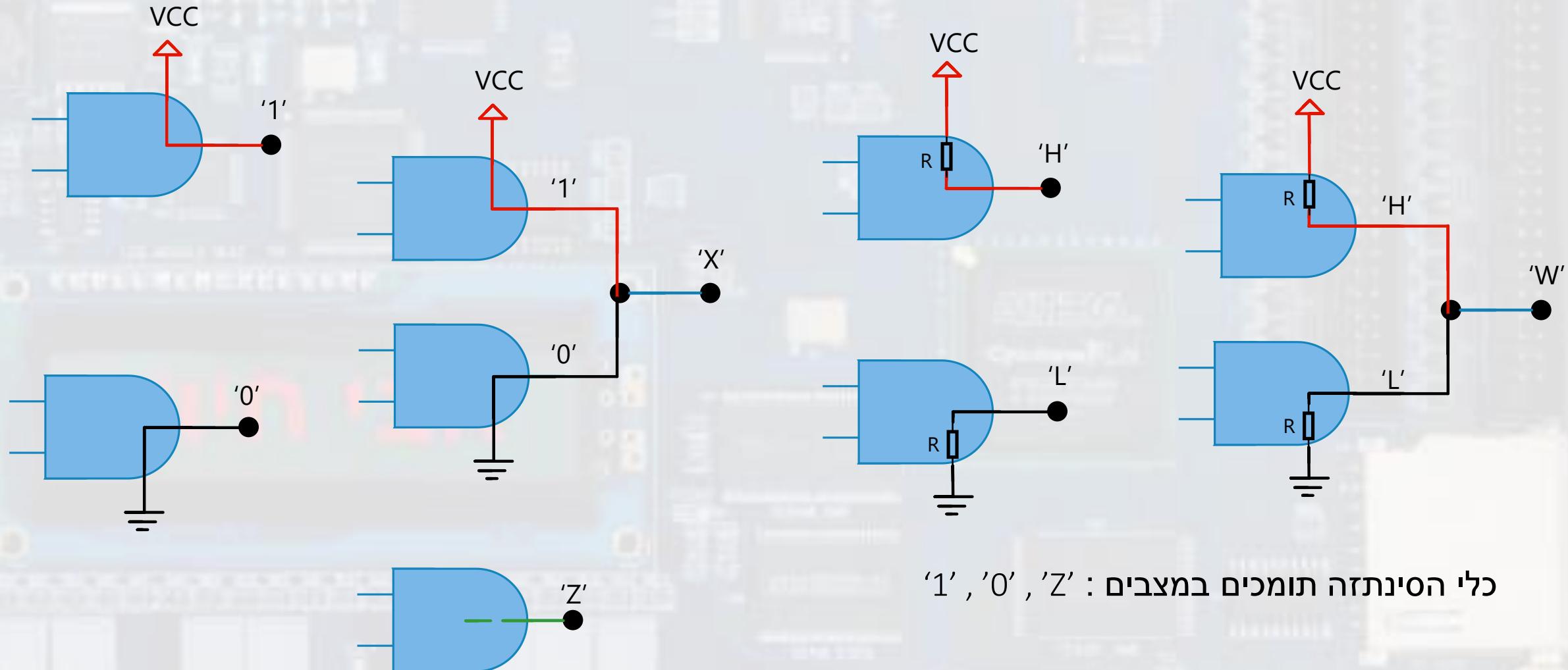
```
library IEEE;  
use IEEE.STD_LOGIC_1164.all;
```

כל הפעולות שפועל על bit ו bit_vector, פועלות גם על סוג מידע זה.

מצבים הלוגיים של סוג המידע std_logic

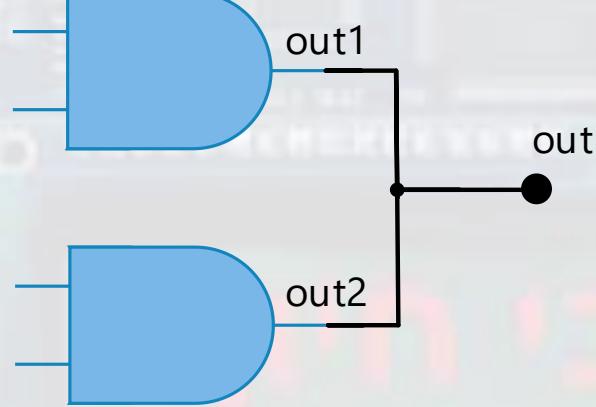
- '1' – 1 לוגי חזק
- '0' – 0 לוגי חזק
- 'Z' – עכבה גבוהה (נתק) , (Z נרשם באות גדולה)
- 'X' – לא ידוע , יכול לקבל כתוצאה מהתנגשות במידע (X נרשם באות גדולה)
- 'L' – '0' לוגי חלש
- 'H' – '1' לוגי חלש
- 'W' – לא ידוע חלש ('X' חלש)
- '-' DON'T CARE – '
- 'U' – ערך לא מאותחל, ערך של משתנה לפני שבוצעה השמה (U נרשם באות גדולה)

תיאור המצבים



כל הסינטזה תומכים במצבים : 'Z', '0', '1',

תרגיל דוגמה



VHDL
אבי היון

out1	out2	out
'0'	'0'	'0'
'0'	'1'	'X'
'0'	'Z'	'0'
'0'	'X'	'X'
'0'	'L'	'0'
'0'	'H'	'0'
'0'	'W'	'0'
'1'	'1'	'1'
'1'	'Z'	'1'
'1'	'X'	'X'
'1'	'L'	'1'
'1'	'H'	'1'
'1'	'W'	'1'

out1	out2	out
'Z'	'Z'	'Z'
'Z'	'L'	'L'
'Z'	'H'	'H'
'Z'	'X'	'X'
'Z'	'W'	'W'
'X'	'L'	'X'
'X'	'H'	'X'
'X'	'X'	'X'
'X'	'W'	'X'
'L'	'L'	'L'
'L'	'H'	'W'
'L'	'W'	'W'
'H'	'H'	'H'
'H'	'W'	'W'
'W'	'W'	'W'

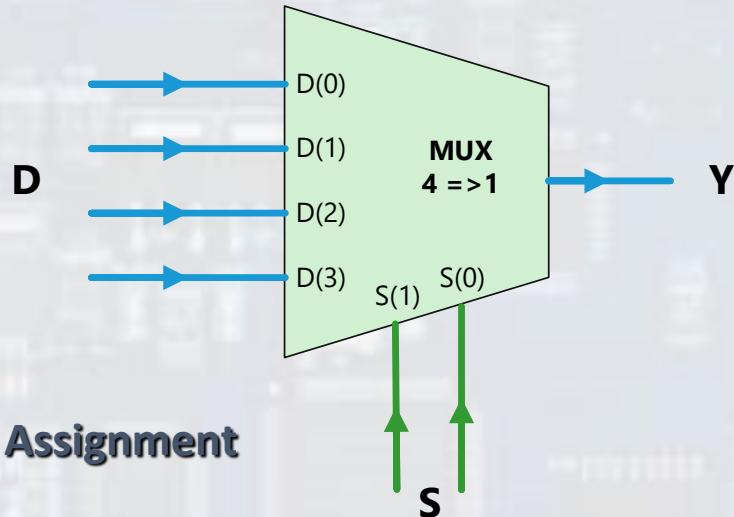
תיאור התנוגותי - התניה מחוץ לתהlixir

השמה מותנת – WHEN-ELSE Conditional Assignment

```
signal_name <= expression_1 when condition_1 else
                    expression_2 when condition_2 else
                    expression_3 when condition_3 else
                    .
                    .
                    expression_n; -- others
```

השמה נבחרת – WITH-SELECT-WHEN Selected Assignment

```
with selection_signal select
signal_name <= value_1 when choice_1 of selection_signal ,
                    value_2 when choice_2 of selection_signal ,
                    value_3 when choice_3 of selection_signal ,
                    .
                    .
                    value_n when others;
```



Conditional Assignment

```

library IEEE;
use IEEE.std_logic_1164.all;

entity mux4to1 is
    port (D :in std_logic_vector (3 downto 0);
          S :in integer range 0 to 3;
          Y :out std_logic );
end mux4to1;

architecture arc_mux of mux4to1 is
begin

    Y<= D(0) WHEN S=0 ELSE
        D(1) WHEN S=1 ELSE
        D(2) WHEN S=2 ELSE
        D(3);

end arc_mux;

```

S	Y
0	D(0)
1	D(1)
2	D(2)
3	D(3)

MUX 4 -> 1 דוגמה:

Selected Assignment

```

library IEEE;
use IEEE.std_logic_1164.all;

entity mux4to1 is
    port (D :in std_logic_vector (3 downto 0);
          S :in integer range 0 to 3;
          Y :out std_logic );
end mux4to1;

architecture arc_mux of mux4to1 is
begin
    with S select
        Y<= D(0) WHEN 0 ,
        D(1) WHEN 1 ,
        D(2) WHEN 2 ,
        D(3) WHEN others ; -- 3

end arc_mux;

```

האם אפשר בפקודה אחת לבצע את רכיב ה-MUX?

```
library IEEE;
use IEEE.std_logic_1164.all;

entity mux4to1 is
    port (D :in std_logic_vector (3 downto 0);
          S :in integer range 0 to 3;
          Y :out std_logic );
end mux4to1;

architecture arc_mux of mux4to1 is
begin

    Y<= D(S);

end arc_mux;
```

```

library IEEE;
use IEEE.std_logic_1164.all;

entity dmux1to4 is
  port (D : in std_logic;
        S : in std_logic_vector (1 downto 0);
        Y : out std_logic_vector (3 downto 0));
end;

```

דרך א' - באמצעות שרשור

```

architecture arc_dmux of dmux1to4 is
begin
  Y<=
    "000" & D when S= "00" else
    "00" & D & '0' when S= "01" else
    '0' & D & "00" when S= "10" else
    D & "000";
end;

```

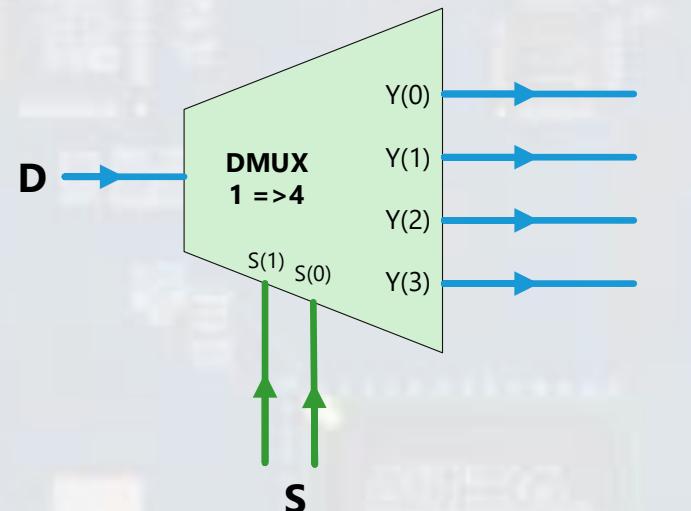
דרך ב' - באמצעות קבוצה

```

architecture arc_dmux of dmux1to4 is
begin
  Y<=
    ('0','0','0',D) when S= "00" else
    ('0','0',D,'0') when S= "01" else
    ('0',D,'0','0') when S= "10" else
    (D,'0','0','0');
end;

```

דוגמה: DMUX 1 to 4



S(1) In	S(0) In	Y(3) Out	Y(2) Out	Y(1) Out	Y(0) Out
0	0	0	0	0	D
0	1	0	0	D	0
1	0	0	D	0	0
1	1	D	0	0	0

דרך ג' - כל מוצא בנפרד

```

architecture arc_dmux of dmux1to4 is
begin
  Y(0)<= D when S= "00" else '0';
  Y(1)<= D when S= "01" else '0';
  Y(2)<= D when S= "10" else '0';
  Y(3)<= D when S= "11" else '0';
end;

```

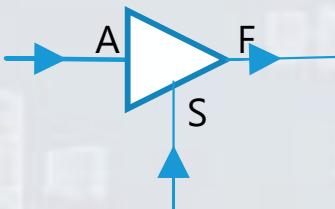
דוגמה: חוץ עם עכבה גבוהה

Conditional Assignment

```
library IEEE;
use IEEE.std_logic_1164.all;

entity BUF is
    port (A,S :in std_logic;
          F :out std_logic );
end ;

architecture arc_buf of BUF is
begin
    F<= A when S='1' else 'Z';
end;
```



S	F
0	'Z'
1	A

Selected Assignment

```
library IEEE;
use IEEE.std_logic_1164.all;

entity BUF is
    port (A,S :in std_logic;
          F :out std_logic );
end ;

architecture arc_buf of BUF is
begin
    with S select
        F<= A when '1' ,
        'Z' when others ;
end;
```

Conditional Assignment

```

entity my_decoder is
  port (A,B,C : in bit;
        F : out bit);
end my_decoder;

architecture arc_deoder of my_decoder is
  signal ABC: bit_vector(2 downto 0);
begin
  ABC <= A & B & C;
  F <= '1' when ABC="000" or ABC="001" or ABC="100" else
    '0';
end arc_deoder;

```

Selected Assignment

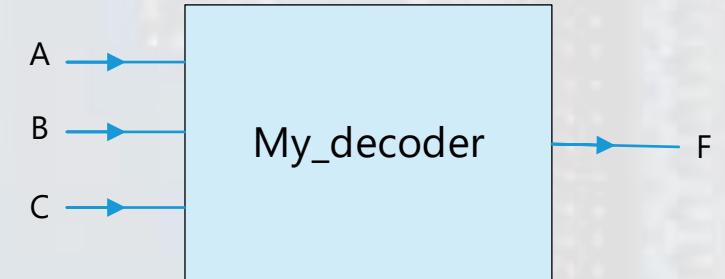
```

entity my_decoder is
  port (A,B,C : in bit;
        F : out bit);
end my_decoder;

architecture arc_deoder of my_decoder is
  signal ABC: bit_vector(2 downto 0);
begin
  ABC <= A & B & C;
  WITH (ABC) SELECT
    F <= '1' when "000" | "001" | "100" ,
      '0' when others;
end arc_deoder;

```

דוגמה: מפענה



	A in	B in	C in	F out
0	0	0	0	1
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	1
5	1	0	1	0
6	1	1	0	0
7	1	1	1	0

דוגמה: אסכם מלא (Full Adder) FA

```

entity FA is
    port(A,B,Ci : in bit;
         S,Co  : out bit);
end;

architecture arc_FA  of FA is
begin
    S<= (A xor B) xor Ci;
    Co <= (A and B) or (A and Ci) or (B and Ci);

end;

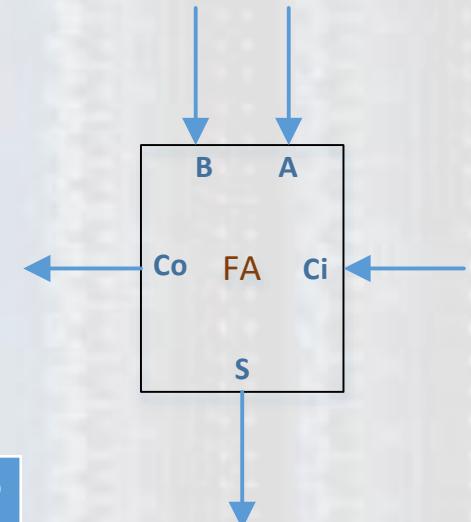
```

$$S = A \oplus B \oplus Ci$$

$$Co = AB + ACi + BCi$$

טבלת אמת

Ci in	A in	B in	S out	Co out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



התהלייר - PROCESS

- התהלייר יכול להיות במצב פעיל (ער) או לא פעיל (ישן).
- לתהלייר יש בדרך כלל רשימת רגיסטרות.
- כאשר אוט ברשימה הרגיסטרות משנה את הערך, התהלייר מתעורר וכל הוצאות העקבות אחרי הרגיסטרות מבוצעות.
- בסוף התהלייר, הוא חוזר למצב ישן.

מבנה התכבירי של התחליך

label: **PROCESS** (sensitivity list)

-- variable declarations

BEGIN

-- sequential statements

END PROCESS **label** ;

- השם **label** והמשתנים **variable** הם אופציונליים.
- התחליך מתבצע, כאשר אחד מהרגישויות משתנה.

דוגמה לשימוש פשוט בתהיליך

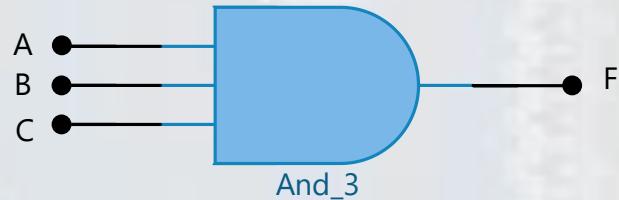
```

library IEEE;
use IEEE.std_logic_1164.all;

entity and_3 is
    port (A,B,C :in std_logic;
          F :out std_logic );
end and_3;

architecture arc_and of and_3 is
begin
    process(A,B,C)
    begin
        F <= A and B and C ;
    end process;
end arc_and;

```



- התהיליך רגיש לשינוי אחד האותות :

A, B, C

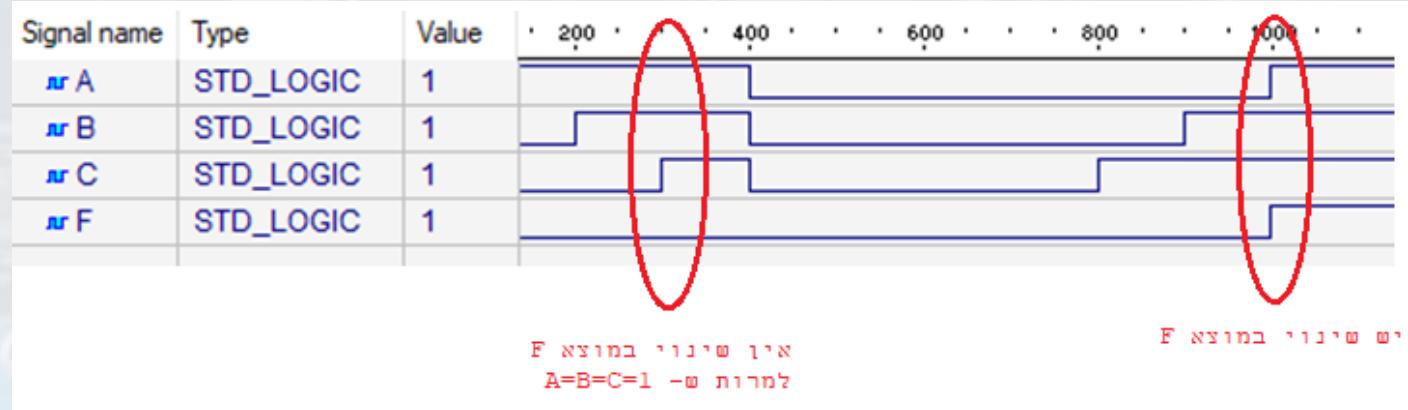
- בדוגמה זו הביטוי ללא PROCESS יבצע את אותה משימה.

מה יקרה אם נסיר בדוגמה את אחת הרגישיות ?

```
library IEEE;  
use IEEE.std_logic_1164.all;  
  
entity and_3 is  
    port (A,B,C :in std_logic;  
          F :out std_logic );  
end and_3;  
  
architecture arc_and of and_3 is  
begin  
    process(A,B)  
    begin  
        F <= A and B and C ;  
    end process;  
end arc_and;
```

המוצא F לא מגייב כאשר הכניסה C משתנה.
התהילר מתיחס ל-C רק כאשר יש שינוי באחת הרגישיות A או B . נוצר מעין זיכרון ל-C.

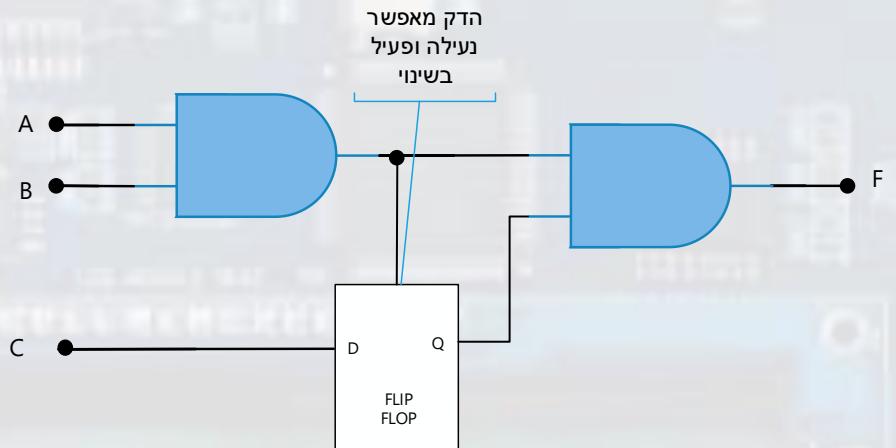
בדוגמה האות C לא מופיע ברשימה הרגישיות,
כלומר הפעולה תתרחש רק אם יהיה שונה
באותות A ו- B



# /tb_and3/A	# /tb_and3/B	# /tb_and3/C	# /tb_and3/F
0	0	0	U
0	0	0	0
1	0	0	0
1	1	0	0
1	1	1	0
0	0	0	0
0	0	1	0
0	1	1	0
1	1	1	0
1	1	1	1

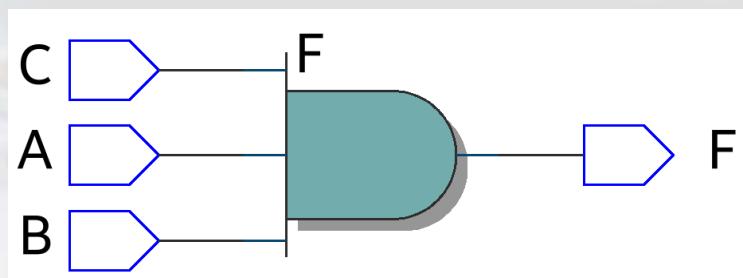
התיחסות כלפי הסימולציה והסינטזה Quartus לבעה

- כלי הסימולציה מתיחסים כאילו נוצר המודול הבא:
 התיחסות ל-C רק כאשר A או B משתנים.
 C נשמר על ידי זיכרון המגיב לשינוי של
 תוצאה פעולה AND של A ו-B



- כלי הסינטזה כמו Quartus מעריכים מבעה זו ומתייחסים כאילו C מופיע ברשימת רגישיות.

חומרה שהתקבלה לאחר RTL (Register Transfer Level)



תיאור התנהוגותי - התניה בתוך התהlixir

- בתוך התהlixir אפשר להשתמש בפסוקי השמה פשוטים (למשל: $F \leftarrow A \text{ and } B$)
- התניות בתוך התהlixir נשות באמצעות הפסוקים הבאים:
 - if-then-else
 - Case-When
- ניתן להשתמש במשתנים variable לחישובי ביןיהם.
- שימוש בולולאות.

תחביר - if-then-else

```
if      condition(s) then  
        STATEMENT1 ;  
  
elsif   condition_2 then  
        STATEMENT2 ;  
        .  
        .  
  
else  
        STATEMENTn ;  
  
end if ;
```

התנאי elsif (בל' e) הוא המשך של התנאי, לאחרת אם משתמשים ב- if , שהוא תנאי חדש צריך שתהיה לו סיום של end if; else

תחביר - Case-When

```
case selection_signal is
    when value_1_of_selection_signal => statements 1
    when value_2_of_selection_signal => statements 2
    .
    .
    when value_N_of_selection_signal => statements N
    when others                      => default action
end case;
```

- האובייקט הנבחר לבדיקה, צריך להיות ערך בודד
- צריך לכסות את כל הערכים האפשריים
- כל ערך חייב להופיע פעם בלבד.

if-then-else

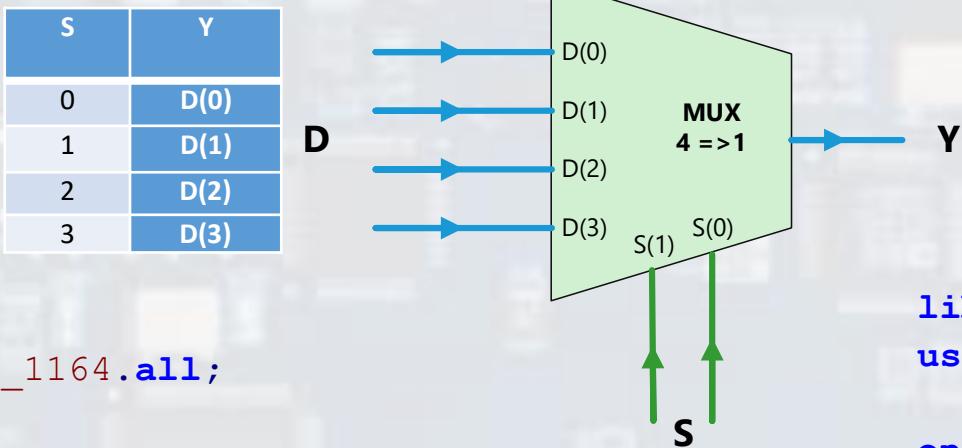
```

library IEEE;
use IEEE.std_logic_1164.all;

entity mux4to1 is
    port (D :in std_logic_vector (3 downto 0);
          S :in integer range 0 to 3;
          Y :out std_logic );
end mux4to1;

architecture arc_mux of mux4to1 is
begin
    process(D,S)
    begin
        if      s=0 then y <= d(0);
        elsif  s=1 then y <= d(1);
        elsif  s=2 then y <= d(2);
        else           y <= d(3);
        end if;
    end process;
end arc_mux;

```

**MUX 4->1 דוגמה:****Case-When**

```

library IEEE;
use IEEE.std_logic_1164.all;

entity mux4to1 is
    port (D :in std_logic_vector (3 downto 0);
          S :in integer range 0 to 3;
          Y :out std_logic );
end mux4to1;

architecture arc_mux of mux4to1 is
begin
    process(D,S)
    begin
        case S is
            when 0 => y <= d(0);
            when 1 => y <= d(1);
            when 2 => y <= d(2);
            when others => y <= d(3); --or 3
        end case;
    end process;
end arc_mux;

```

תיאור מקבילי וסדרתי – בעית שימוש באוט signal בתהיליך

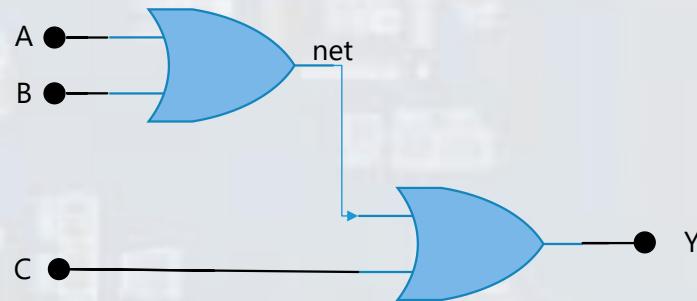
```

library IEEE;
use IEEE.std_logic_1164.all;

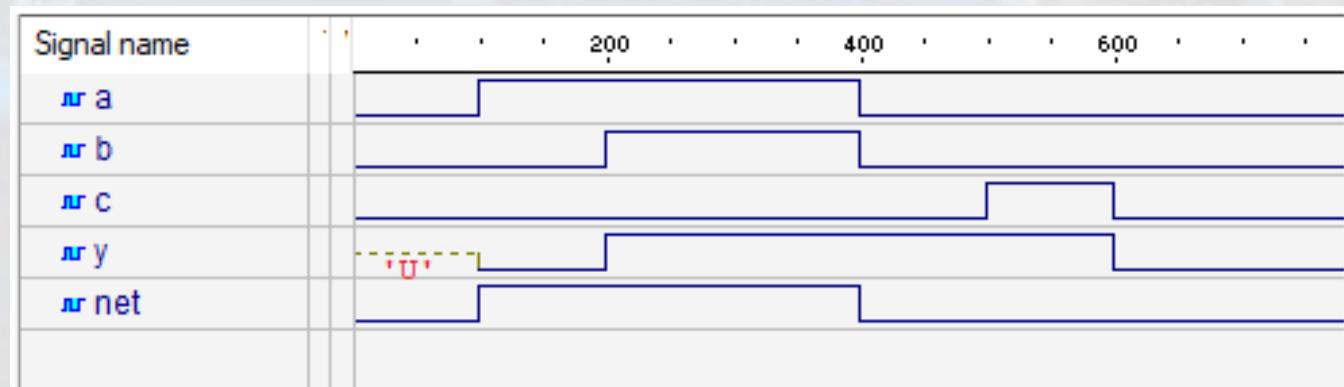
entity test is
    port (A,B,C :in std_logic;
          Y :out std_logic );
end ;

architecture arc of test is
    signal net:std_logic;
begin
    process(A,B,C)
    begin
        net<=A or B;
        Y<= net or C;
    end process;
end;

```



נראה את ה בעיה הדוגמה הבאה,
כפי שמוצגת בסימולציה.



ה בעיה : המוצא Y אינו מגיב מידית לשינוי של A או B במצבה.

סיבה : הפעולות בתוך ה-PROCESS אינן מתבצעות באופן מיידי.

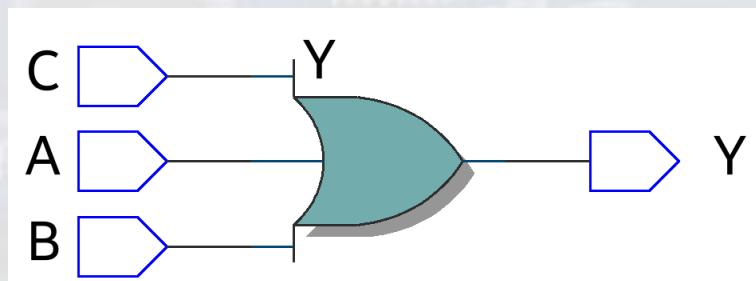
כל השמות נרשומות בזיכרון זמני ורק בסיום התהיליך הן מתבצעות באופן מקבילי (יחד)

לכן נוצר מעין זיכרון לאות net

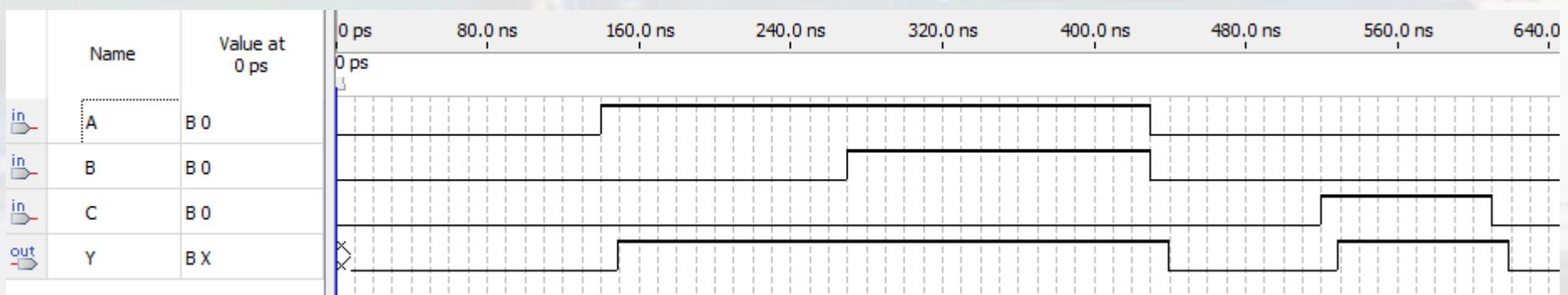
כיצד מתייחסת כל הsynthesizer לבעיה זו

כל הsynthesizer QUARTUS מתעלם מבעיה זו ומבצע את הפעולה, Caino אין את אות net

חומרה שהתקבלה לאחר RTL (Register Transfer Level)



וימולציה המראה תגובה של שער OR רגיל.



פתרון לבעה – שימוש במשתנה variable בתוך התהלייר

```

library IEEE;
use IEEE.std_logic_1164.all;

entity test is
    port (A,B,C :in std_logic;
          Y :out std_logic );
end ;

architecture arc of test is
begin
    process(A,B,C)
        variable net:std_logic;
    begin
        net := A or B;
        Y<= net or C;

    end process;
end;

```

Variable

- הוא משתנה לחישובי ביןים בתוך התהלייר.
- הוא מתרבע מידית בתוך התהלייר.
- מוכר כמשתנה סטטי של אותו תהלייר.
- השמה עם :=

בתוכנית net מקבל מידית את הערך של A or B or C
ומעביר אותו ל- Y

לולאות

- אפשר לבצע מספר גדול של פעולות

לולאת for

label: **for** variable **in** range **loop**

--sequential statements

end loop label ;

דוגמאות

```
for i in 0 to 10 loop
    -- sequential statements
end loop;
```

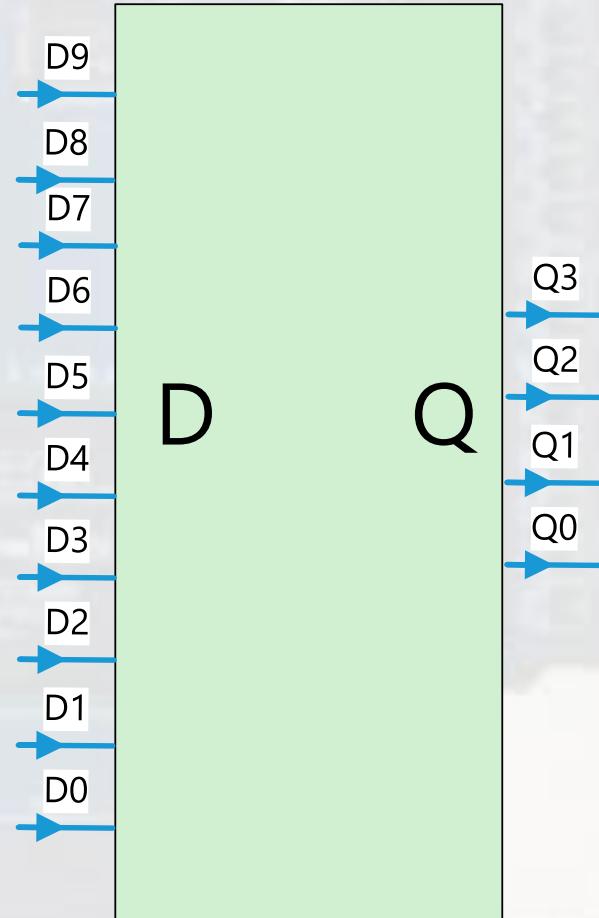
```
for j in 10 downto 2 loop
    -- sequential statements
end loop;
```

דוגמה לתוכנית סופרת את מספר ה-'1' בכניסה באמצעות לולאת for

```

ENTITY count_1 IS
    PORT( d : IN BIT_VECTOR (9 DOWNTO 0);
          q : OUT INTEGER RANGE 0 TO 10);
END ;
ARCHITECTURE arc_count OF count_1 IS
BEGIN
    PROCESS (d)
        VARIABLE num_bits : integer range 0 to 10;
    BEGIN
        num_bits := 0;
        FOR i IN 0 to 9 LOOP
            IF d(i) = '1' THEN
                num_bits := num_bits + 1;
            END IF;
        END LOOP;
        q <= num_bits;
    END PROCESS;
END ;

```



לולאת loop

```
label: loop
    -- sequential statements
    exit when condition;
end loop  label ;
```

```
loop
    A(i) <= '0' ;
    i := i+1;
    exit when    i >10;
end loop;
```

דוגמה

דוגמה לתוכנית סופרת את מספר ה-'1' בכניסה באמצעות לולאת loop

```

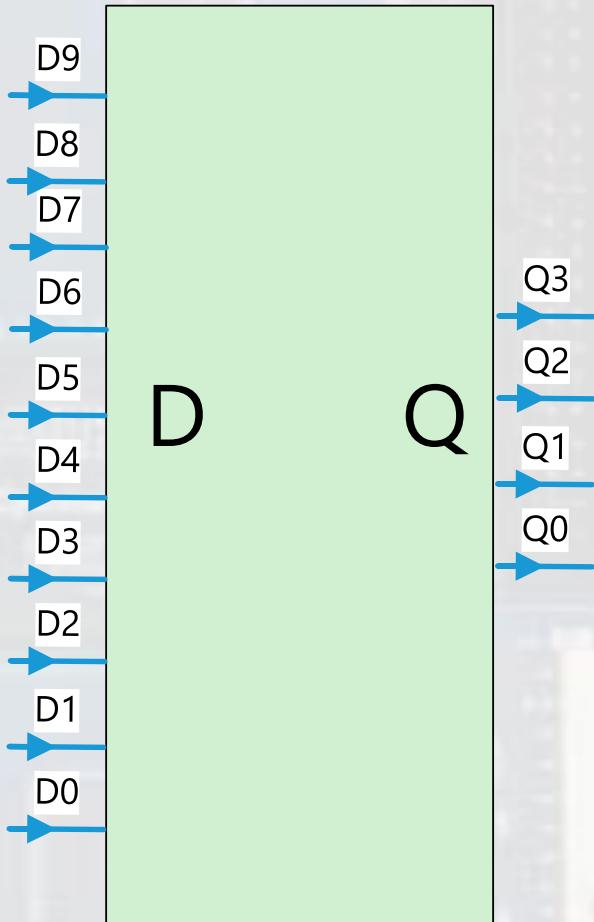
ENTITY loop_1 IS
    PORT( d : IN BIT_VECTOR (9 DOWNTO 0);
          q : OUT INTEGER RANGE 0 TO 10);
END ;

ARCHITECTURE arc_count OF loop_1 IS
BEGIN
    PROCESS (d)
        VARIABLE i, num_bits : integer range 0 to 10;
    BEGIN
        i:=0;
        num_bits:=0;
        LOOP
            IF d(i) = '1' THEN
                num_bits:=num_bits+1;
            END IF;
            i:=i+1;
            exit when (i>9);
        END LOOP;

        q<=num_bits;

    END PROCESS;
END ;

```



ЛОЛАת while

```
label: while condition  
      -- sequential statements  
end loop label ;
```

דוגמה

```
while I < 10 loop  
  A(i) <= '0' ;  
  i := i+1;  
end loop;
```

דוגמה לתוכנית סופרת את מספר ה-'1' בכניסה באמצעות לולאת while

```

ENTITY while_1 IS
    PORT( d : IN BIT_VECTOR (9 DOWNTO 0);
          q : OUT INTEGER RANGE 0 TO 10);
END ;

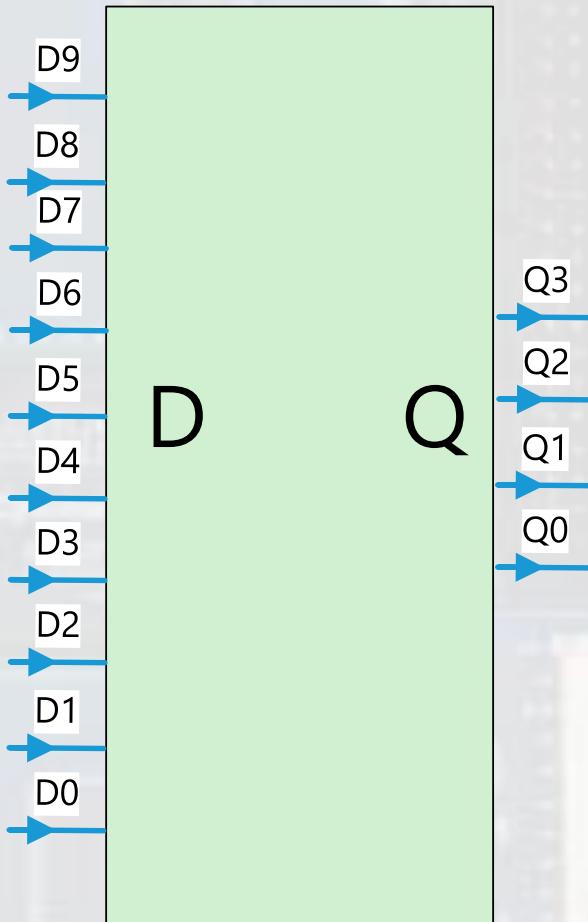
ARCHITECTURE arc_count OF while_1 IS
BEGIN
    PROCESS (d)
        VARIABLE i, num_bits : integer range 0 to 10;
    BEGIN
        i:=0;  num_bits:=0;

        while i<10 LOOP
            IF d(i) = '1' THEN
                num_bits := num_bits+1;
            END IF;
            i:=i+1;
        END LOOP ;

        q<=num_bits;

    END PROCESS;
END ;

```



דוגמה נוספת ללולאה

```

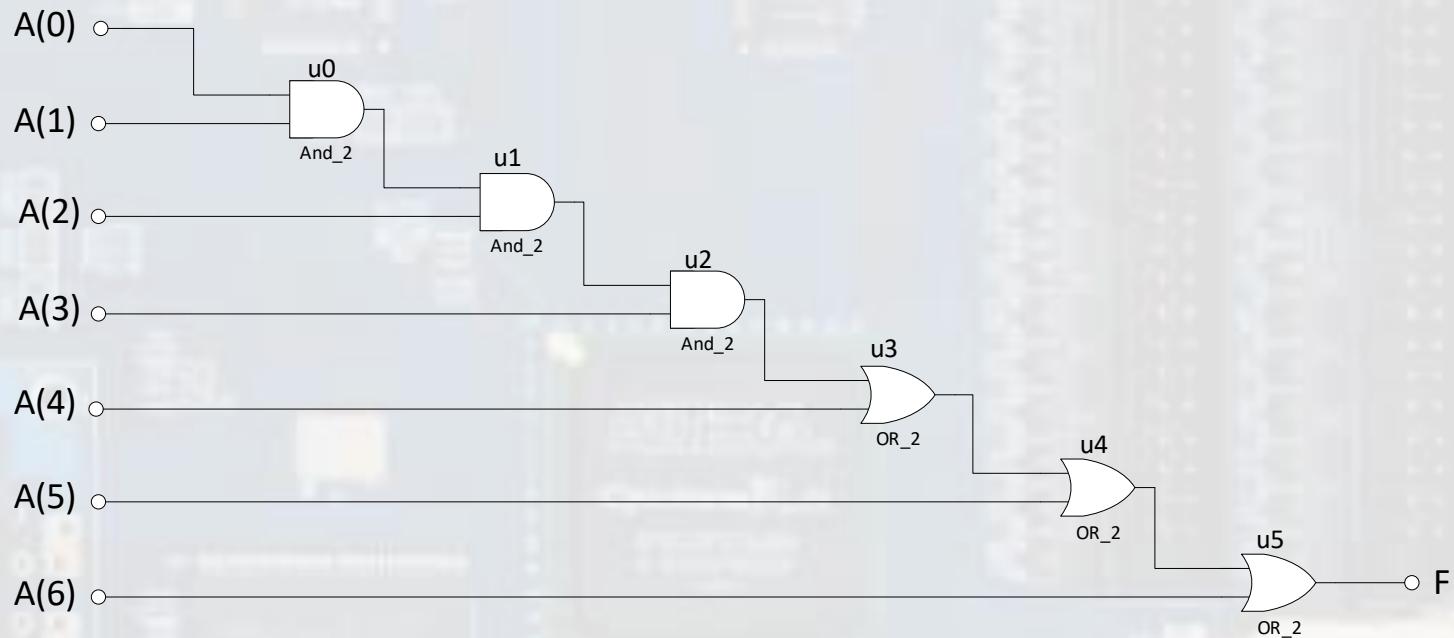
entity and_or is
  port( A : in bit_vector(6 downto 0);
        F : out bit);
END ;

architecture arc of and_or is
begin
  process(A)
    variable temp: bit;
  begin
    temp:=A(0);
    for i in 1 to 6 loop
      if(i < 4) then temp:= temp and A(i);
      else temp:= temp or A(i);
      end if;
    end loop;

    F <= temp;

  end process;
end ;

```



תיאור סינכרוני

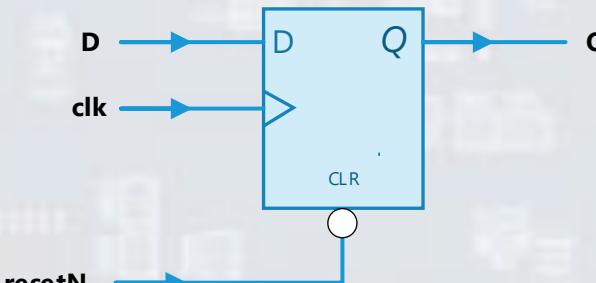
- תיאור סינכרוני של מערכת נעשה עם שימוש באות שעון המגיב לעלייה או ירידת.
- מערכות סינכרוניות יכולות להיות מפליפ-פלופ פשוט ועד למוניים, רגיסטרים ומכונות מצבים המגיבים לשעון.
- שימוש ב- attribute שבו מצמידים את התו 'event' לאות כלשא, גורם לכך שתיקיים תנאי לבדיקת שינוי שינו אותן. למשל `clk'event` הוא תנאי לבדיקת שינוי בשעון `clk`
- תנאי של עליית שעון יכתב: `clk'event and clk='1'`
ירידת שעון: `clk'event and clk='0'`

תיאור של DFF בתוך תהליך

איפוא סינכרוני

```
entity d_ff is
    port (D,clk, resetN :in bit;
          Q :out  bit );
end ;
```

clk	resetN	Q
0	0	0
1	D	



איפוא אסינכרוני

```
entity d_ff is
    port (D,clk, resetN :in bit;
          Q :out  bit );
end ;
```

resetN	clk	Q
0	X	0
1	0	D

```
architecture arc_dff of d_ff is
begin
    process(clk, resetN)
    begin
        if clk'event and clk='1' then
            if resetN ='0' then  Q<='0';
            else      Q<=D;
            end if;
        end if;
    end process;

end;
```

```
entity d_ff is
    port (D,clk, resetN :in bit;
          Q :out  bit );
end ;
```

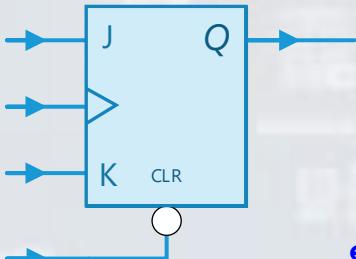


```
architecture arc_dff of d_ff is
begin
    process(clk)
    begin
        if clk'event and clk='1' then
            if resetN = '0' then  Q<='0';
            else      Q<=D;
            end if;
        end if;
    end process;

end;
```

תיאור של JKFF בתוך תהליך

clk	resetN	J	K	Q
0	X	X		0
1	0	0		שומר מצב
1	0	1		0
1	1	0		1
1	1	1		הופך מצב



איפוס סינכרוני

```

entity JK_ff is
    port (J,K,clk, resetN :in bit;
          Q :buffer bit );
end ;

architecture arc_dff of JK_ff is
begin
    process(clk)
    begin
        if clk'event and clk='1' then
            if resetN ='0' then Q<='0';
            elsif J='0' and K='1' then Q<='0';
            elsif J='1' and K='0' then Q<='1';
            elsif J='1' and K='1' then Q<=not Q;
            end if;
        end if;
    end process;
end;

```

resetN	clk	J	K	Q
0	X	X	X	0
1	✓	0	0	שומר מצב
1	✓	0	1	0
1	✓	1	0	1
1	✓	1	1	הופך מצב

איפוס אסינכרוני

```

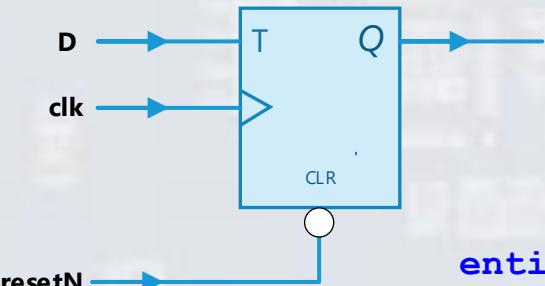
entity JK_ff is
    port (J,K,clk, resetN :in bit;
          Q :buffer bit );
end ;

architecture arc_dff of JK_ff is
begin
    process(clk,resetN)
    begin
        if resetN ='0' then Q<='0';
        elsif clk'event and clk='1' then
            if      J='0' and K='1' then Q<='0';
            elsif J='1' and K='0' then Q<='1';
            elsif J='1' and K='1' then Q<=not Q;
            end if;
        end if;
    end process;
end;

```

תיאור של TFF בתוך תהליך

clk	resetN	T	Q
0	X		0
1	0	שומר מצב	
1	1	הופך מצב	



resetN	clk	T	Q
0	X	X	0
1	↑	0	שומר מצב
1	↑	1	הופך מצב

איפוס סינכרוני

```
entity T_ff is
    port (T,clk, resetN :in bit;
          Q :buffer bit );
end ;

architecture arc_dff of T_ff is
begin
    process(clk)
    begin
        if clk'event and clk='1' then
            if resetN ='0' then Q<='0';
            elsif T='1' then Q<=not Q;
            end if;
        end if;
    end process;
end;
```

איפוס אסינכרוני

```
entity T_ff is
    port (T,clk, resetN :in bit;
          Q :buffer bit );
end ;

architecture arc_dff of T_ff is
begin
    process(clk,resetN)
    begin
        if resetN ='0' then Q <='0';
        elsif clk'event and clk='1' then
            if T ='1' then Q<=not Q;
            end if;
        end if;
    end process;
end;
```

Clock edge detection

- הרגישות לעליית שעון או ירידת שעון מתאימים לאות מסווג bit

```
clk'event and clk='1'
```

```
clk'event and clk='0'
```

מה קורה כאשר האות הוא מסווג `std_logic` ויש בו מצבים נוספים כמו שינוי ממצב 'U', 'Z' או 'H' למצב '1' יפרשו את זה כ שינוי.

(התוצאות שונות של כל הסימולציה והסינטזה לקרה זה , למשל QUARTUS לא יזהה שינוי ממצב Z למצב 1)

פתרון אפשרי לסימולציה הוא לכתוב לעליית שעון את התchapira:

```
if clk'event and clk = '0' = last_value and clk'last_value = '1'
```

פתרון מקובל יותר הוא להשתמש בפונקציות מוכנות המגלות שינוי מ-0 ל-1 או הופך בלבד.

- `if rising_edge(clk)`
- `if falling_edge(clk)`

מוניים וארגוני סינכרוניים

- מונים וארגוני מסונכרים לעליה או ירידת שעון
- בנוספּ קיימים הדקי בקרה כמו : איפוס, טעינה, אפשר ועוד
- הדקי הבדיקה יכולים להיות סינכרוניים(תלוים בשעון) או אסינכרוניים (לא תלויים בשעון)
- יש לשים לב להבדל בין התתייחסות של כל הSIMOLAZA לאות מסוג integer

דוגמה מונה של 4 סיביות הסופר מעלה

```

library IEEE;
use IEEE.std_logic_1164.all;

entity Counter_0_to_15 is
    port(clk ,clear: in std_logic;
          q: buffer integer range 0 to 15);
end;

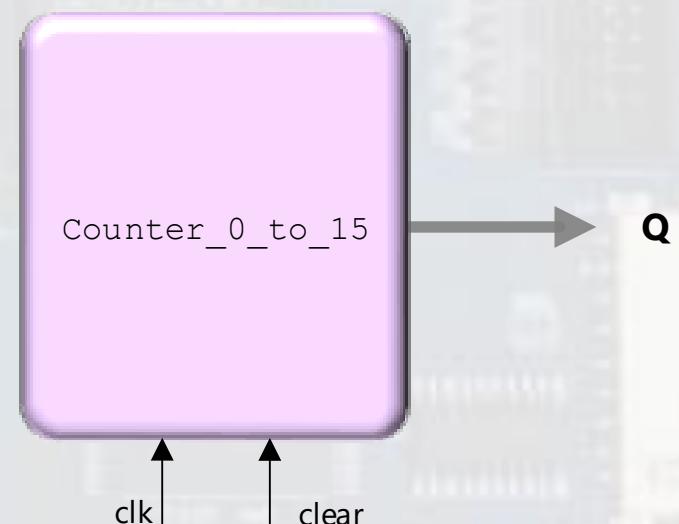
architecture arc_count of Counter_0_to_15 is
begin

    process(clk)
    begin
        if rising_edge(clk) then
            if clear='1' then
                q<=0;
            else
                q<=q+1;
            end if;
        end if;
    end process;
end;

```

בדוגמה נתון מונה עם איפוס
סינכרוני.

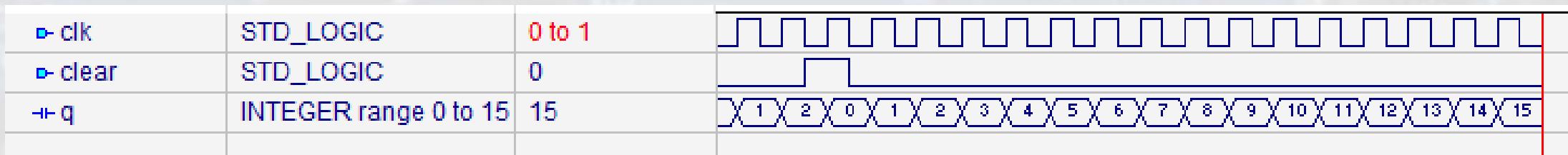
air יתיחסו כל הSIMOLציה
והסינטזה לצורת כתיבה זו



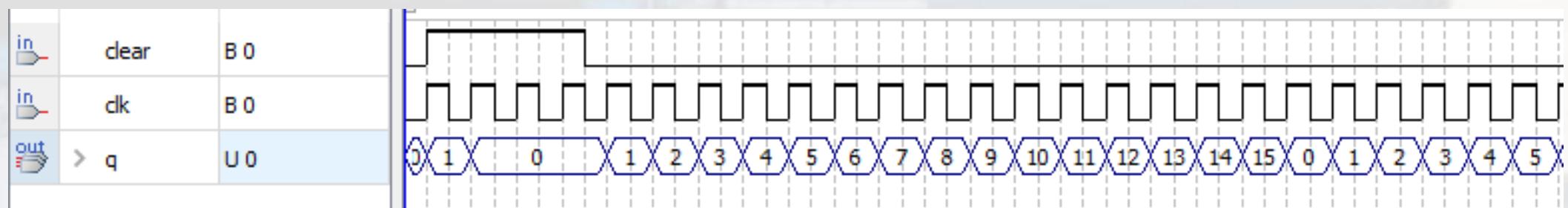
התוצאות של כלי הסימולציה והסינזה למונה

כלי הסימולציה יריצו עד לערך 15 ויציגו הודעה שגיאה

```
# RUNTIME: Fatal Error: RUNTIME_0043 counter.vhd (20): Value 16 out of range (0 to 15).
```



כלי הסינזה QARTUS יקצו 4 סיביות למונח ויקדמו את המונח בצורה מחזורית מ-0 עד 15



פתרון לבעיה

להגדיר את תחום המניה ולאפסו את המונה כאשר הוא יגיע לערך 15

```

library IEEE;
use IEEE.std_logic_1164.all;

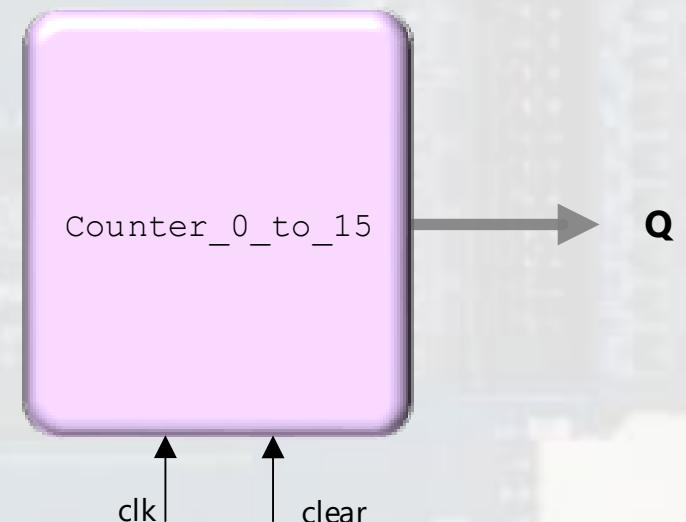
entity Counter_0_to_15 is
    port(clk ,clear: in std_logic;
          q: buffer integer range 0 to 15);
end;

architecture arc_count of Counter_0_to_15 is
begin

process(clk)
begin
    if rising_edge(clk) then
        if clear='1' then
            q <= 0;
        else
            if q < 15 then q <= q+1;
            else q <= 0; end if;
        end if;
    end if;
end process;
end;

```

תנאי להגבלה



דרך נוספת – להגדיר את המוצא מסווג וקטור

אות מסווג וקטור לא ניתן לבצע פעולה אריתמטית, אך יש בעיה לכך אם מונה, אבל אם נצרכ את הchniba : all IEEE.std_logic_unsigned.all;

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity Counter_0_to_15 is
    port(clk ,clear: in std_logic;
         q: buffer std_logic_vector(3 downto 0));
end;

architecture arc_count of Counter_0_to_15 is
begin

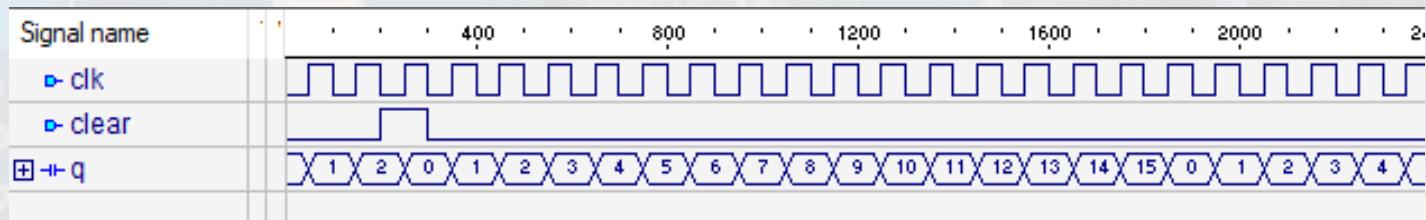
    process(clk)
    begin
        if rising_edge(clk) then

            if clear='1' then
                q <= "0000";
            else
                q <= q+1;
            end if;
        end if;
    end process;
end;

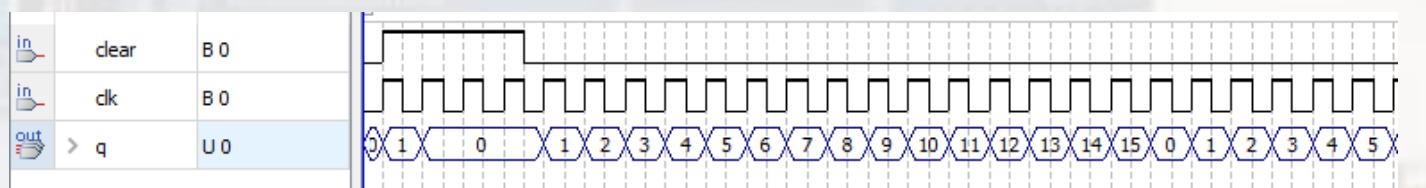
```

נקבל את אותה תוצאה עבור סימולטור וסינטזה

תוצאה סימולטור



תוצאה כלי סינטזה - QUARTUS



דוגמה – מונה BCD מעלה/מטה

```

library IEEE;
use IEEE.std_logic_1164.all;

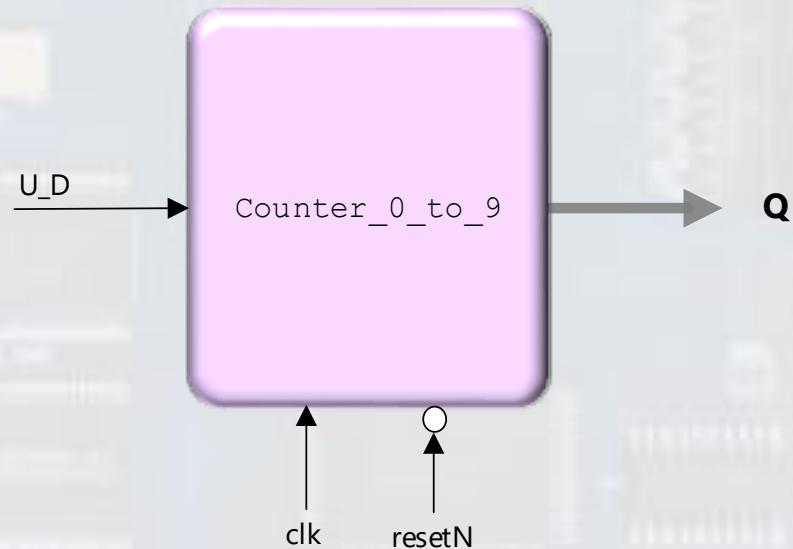
entity Counter_0_to_9 is
    port(clk ,resetN, u_d: in std_logic;
          q: buffer integer range 0 to 9);
end;

architecture arc_count of Counter_0_to_9 is
begin

process(clk)
begin
    if rising_edge(clk) then
        if resetN = '0' then q <= 0;
        elsif u_d = '1' then
            if q = 9 then q <= 0;
            else q <= q+1;
            end if;
        else
            if q = 0 then q <= 9;
            else q <= q-1;
            end if;
        end if;
    end if;
end process;
end;

```

פעולות האוגר			
clk	resetN	U_d	
עליה	0	X	Q=0
עליה	1	1	סופר מעלה מ-0 עד 9 וחזר ל-0
עליה	1	0	סופר מטה מ-9 עד 0 וחזר ל-9



דוגמה - אוגר הזזה מטוררי למקבילי

```

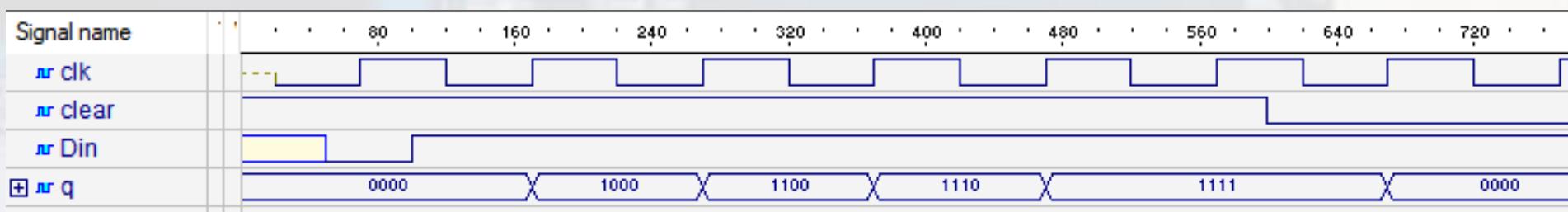
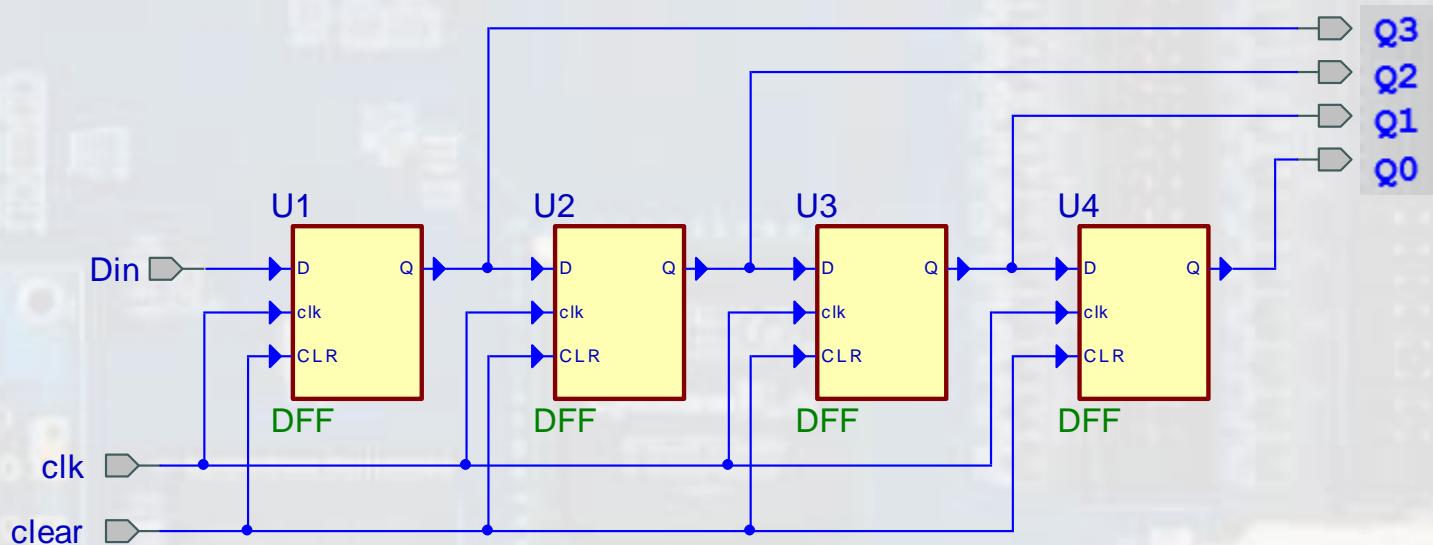
library IEEE;
use IEEE.std_logic_1164.all;

entity Shift4bit is
    port(clk ,clear, Din: in std_logic;
          q: buffer std_logic_vector(3 downto 0));
end;

architecture arc_shift of Shift4bit is
begin

process(clk)
begin
    if rising_edge(clk) then
        if clear='0' then
            q<="0000";
        else
            q<= Din & q(3 downto 1);
        end if;
    end if;
end process;
end;

```



דוגמה - אוגר הזרה ממקביל לטור

```

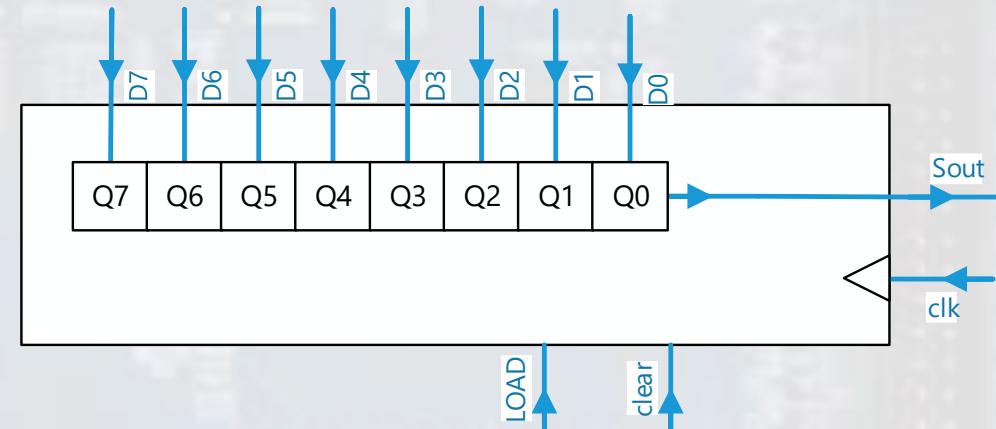
library IEEE;
use IEEE.std_logic_1164.all;

entity ParallelToSerial is
    port(clk ,clear, load: in std_logic;
          Sout : out std_logic;
          D: in std_logic_vector(7 downto 0) );
end;

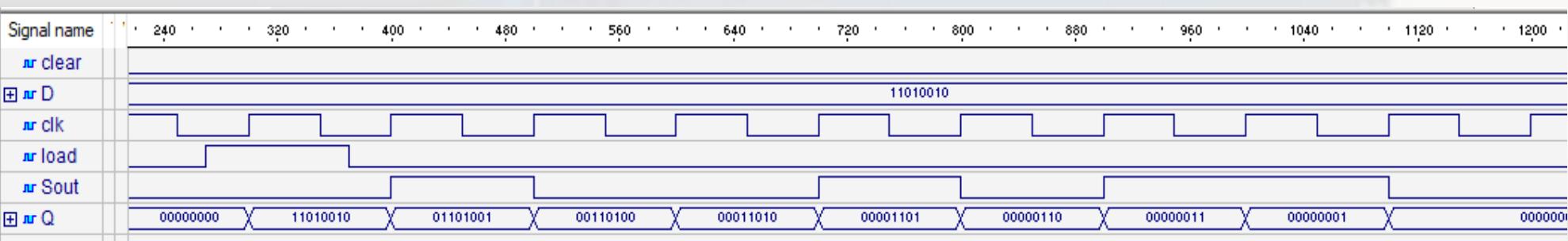
architecture arc_shift of ParallelToSerial is
    signal Q: std_logic_vector(7 downto 0);
begin
    Sout <= Q(0);

    process(clk,clear)
    begin
        if clear='1' then
            Q<=(others => '0');
        elsif rising_edge(clk) then
            if load= '1' then
                Q <= D;
            else
                Q <= '0' & Q(7 downto 1);
            end if;
        end if;
    end process;
end;

```



clear	clk	LOAD	פעולות האוגר
1	X	X	Q=0
0	עליה	1	Q=D
0	עליה	0	הזרה ימינה



דוגמה - גוזר סינכרוני (גלי עליית אחת)

```

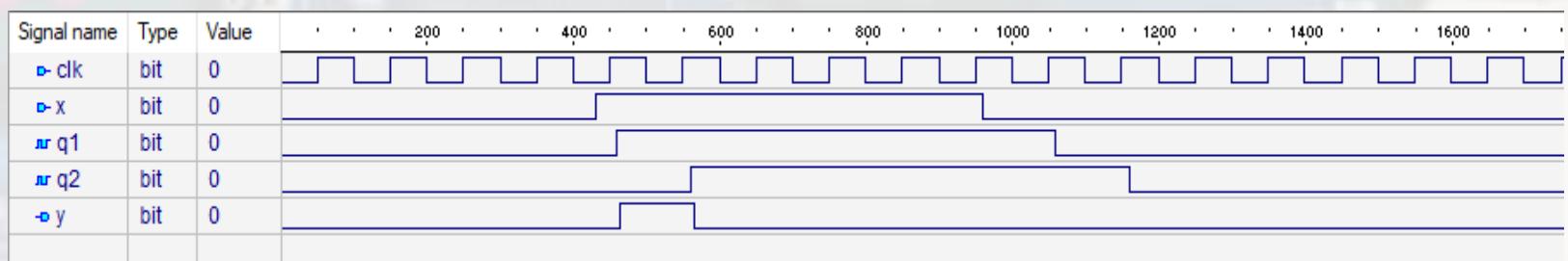
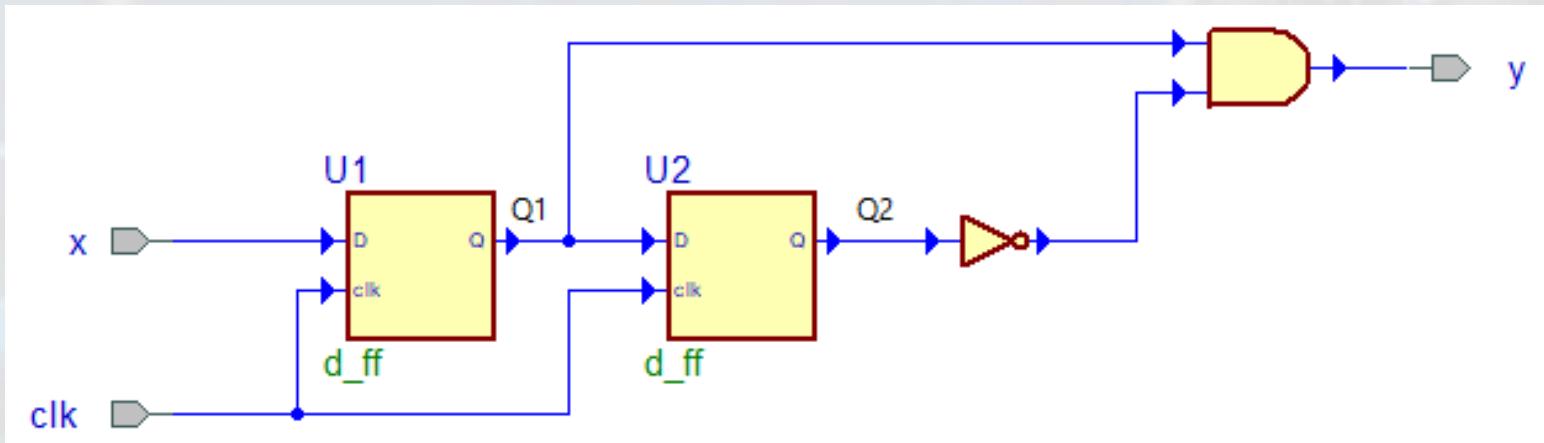
entity RiseDetect is
  port(x, clk: in bit;
        y: out bit);
end  RiseDetect;

architecture arc_rise of  RiseDetect
is
  signal q1,q2:bit;
begin
  process(clk)
  begin
    if clk'event and clk='1' then
      q1 <= x ;
      q2 <= q1 ;
    end if;
  end process;

  y<= q1 and (not q2);

end arc_rise;

```



הזמן לזכור סימולציית השהייה של הרכיבים

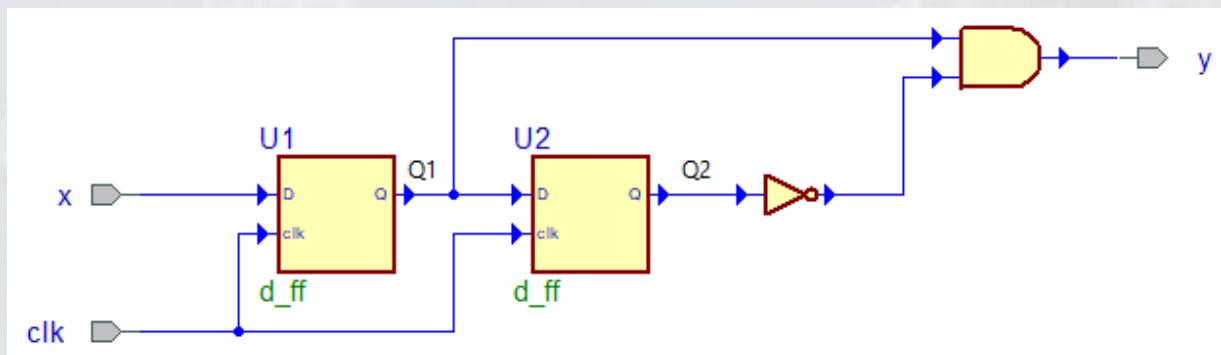
הפרדה בין חלקיים צירופיים לchnckeroniים

חשוב להבדיל בין החלקים הצירופיים לשינכרוניים, כדי להימנע משגיאות.
לדוגמה למעגל הגוזר, אם לא נבצע הפרדה נקבל בעיה.

```
entity RiseDetect2 is
  port(x, clk: in bit;
        y: out bit);
end RiseDetect2;

architecture arc_rise2 of RiseDetect2 is
  signal q1,q2:bit;
begin
  process(clk)
  begin
    if clk'event and clk='1' then
      q1 <= x;
      q2 <= q1;
    end if;
    y<= q1 and (not q2);
  end process;
end arc_rise2;
```

ז' רשום בתוך ה-
PROCESS

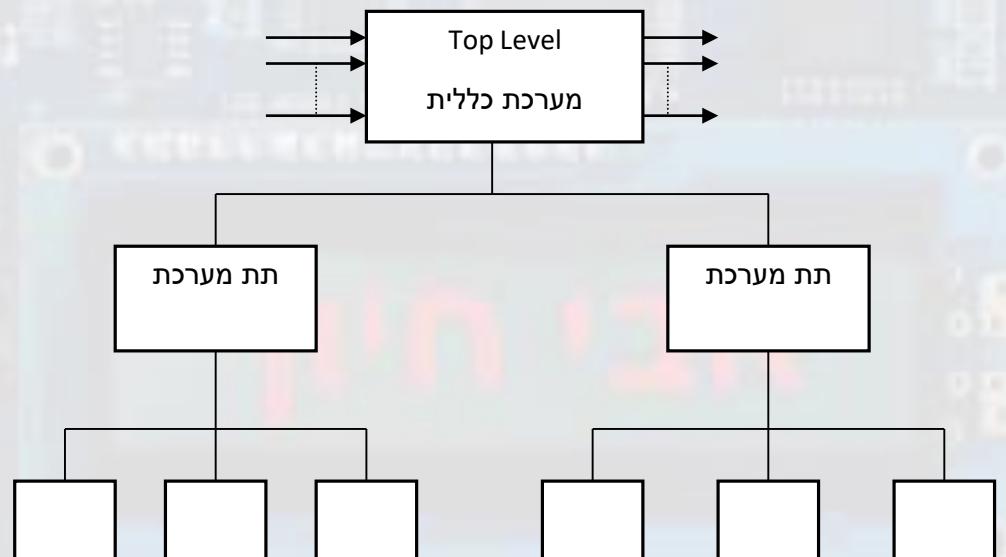


ז' משתנה בירידת
השעון ולא בשינוי
של Q1,Q2

תיאור היררכי (מבני)

מאפייני מערכת היררכית:

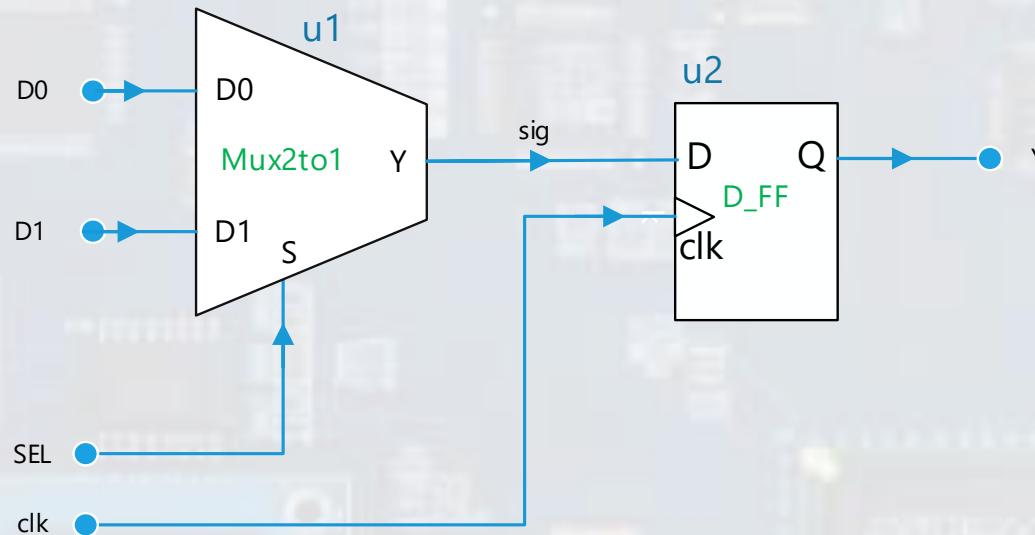
- לכל מערכת כניסה ויציאה – קלט, פלט.
- כל המערכות מורכבות מתח מערכות בצורה היררכית שהן בעצמן תח מערכות למערכת שמעליה.



שלבים בתכנון היררכי

1. הגדרת הכניסות ופרוט הדרישות של המערכת המלאה
2. פרוק המערכת לתח מערכות
3. הגדרת הכניסות ופרוט הדרישות של כל תח מערכות
4. כתיבת תוכנית לכל תח מערכות והרצה תוכנית בדיקה וסימולציה
5. חיבור כל תח היחידות
6. הרצת תוכנית בדיקה וסימולציה למערכת המלאה

דוגמה ראשונה לתוכון היררכי

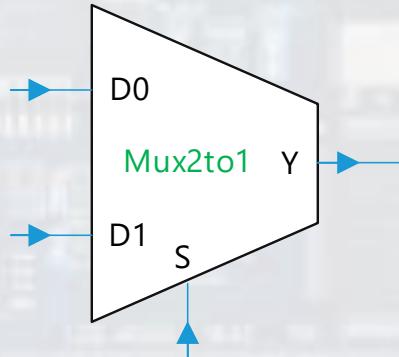


לפני שנחבר את הרכיבים השונים למערכת אחת גדולה - Top Level

1. לדאוג שקיים קוד לכל רכיב שבמערכת, שעבר בדיקה
2. להגדיר entity והכניסות והיציאות של המערכת Top Level
3. להגדיר al signals פנימיים
4. להציג בתוכנה על הרכיבים הפנימיים (Component Declaration)
5. ביצוע ב- architecture, חיבור הרכיבים הפנימיים באמצעות פקודות port map

shellבים

- נכתב קוד לכל רכיב

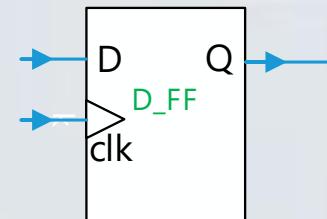


```

library IEEE;
use IEEE.std_logic_1164.all;

entity Mux2to1 is
    port (D :in std_logic_vector (1 downto 0);
          S :in  std_logic ;
          Y :out  std_logic );
end ;

architecture arc_muxb of Mux2to1 is
begin
    Y<= D(0) when S='0' else D(1);
end;
    
```



```

library IEEE;
use IEEE.std_logic_1164.all;

entity D_FF is
    port (D,clk :in std_logic;
          Q :out  std_logic );
end ;

architecture arc_dff of D_FF is
begin
    process(clk)
    begin
        if rising_edge(clk)      then
            Q<=D;
        end if;
    end process;
end;
    
```

תוכנית ראשית

```

library IEEE;
use IEEE.std_logic_1164.all;

entity MuxDFF is
    port (D :in std_logic_vector (1 downto 0);
          SEL,clk :in std_logic;
          Y: out std_logic);
end;

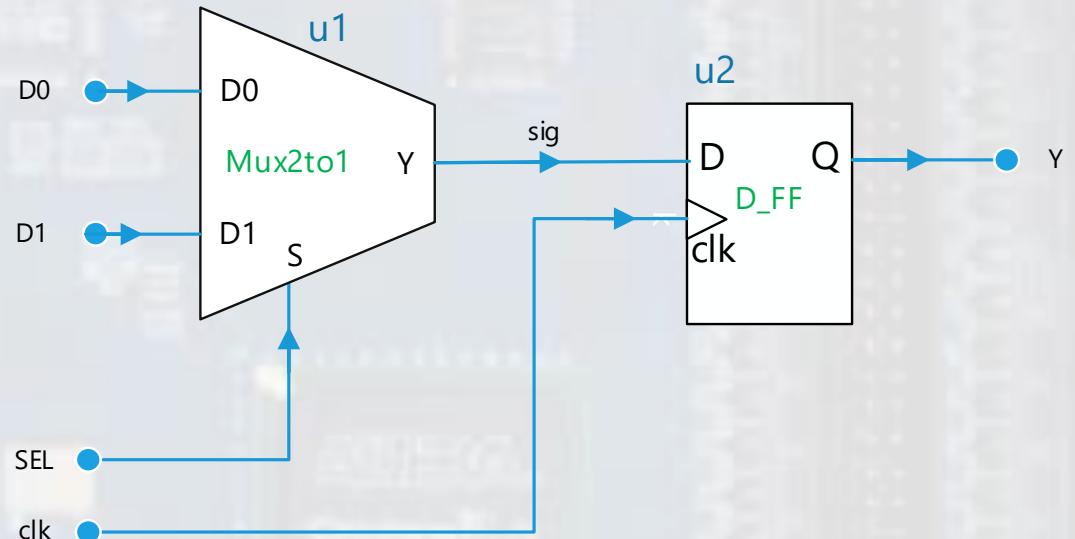
architecture arc_TOP of MuxDFF is
    signal sig: std_logic;

    component Mux2to1
        port (D :in std_logic_vector (1 downto 0);
              S :in std_logic ;
              Y :out std_logic );
    end component;

    component D_FF
        port (D,clk :in std_logic;
              Q :out std_logic );
    end component;

begin
    u1: Mux2to1 port map ( D => D, S => SEL , Y => sig );
    u2: D_FF      port map ( D => sig, clk => clk, Q => Y );
end;

```



אפשר לקצר בפקודות של port map לצורה הבאה:

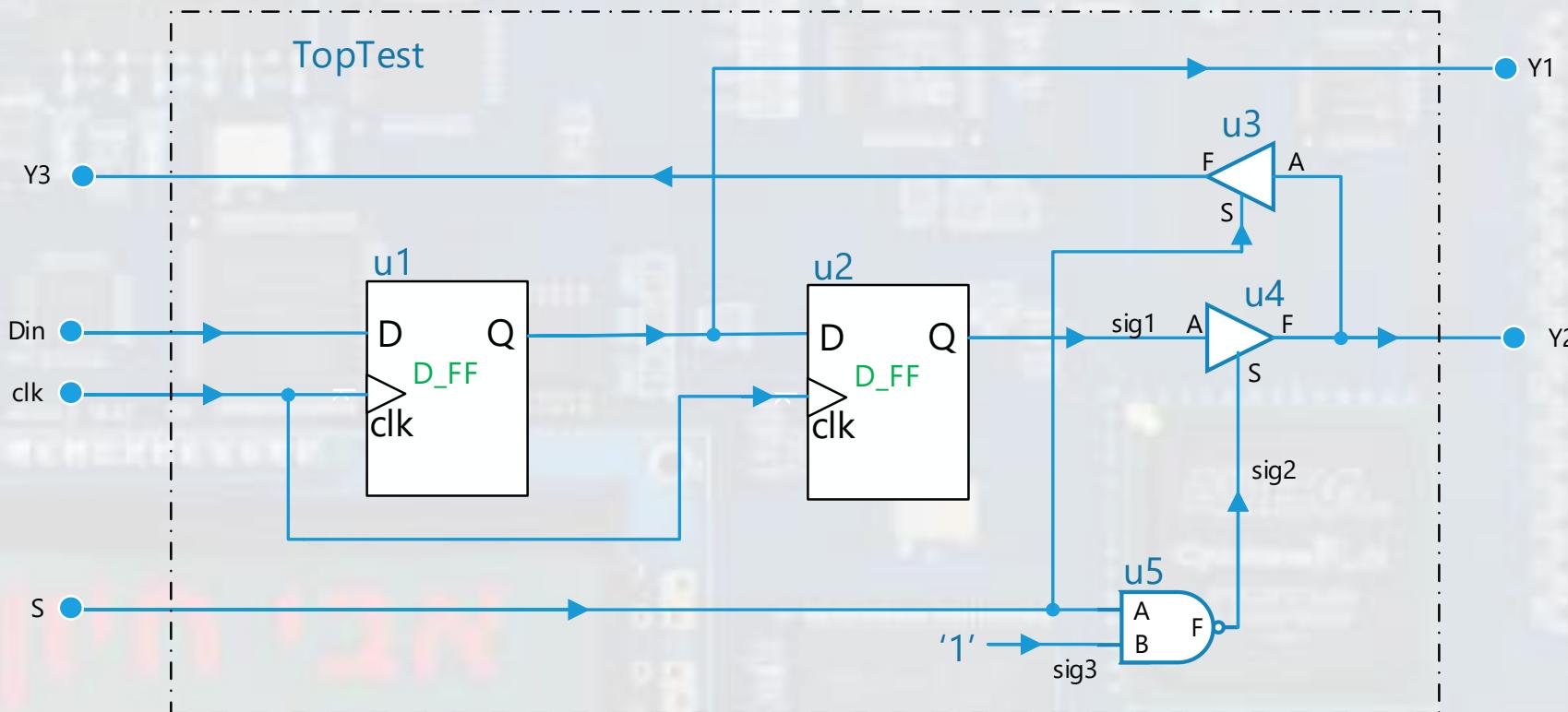
```

u1: Mux2to1 port map (D, SEL ,sig );
u2: D_FF      port map (sig, clk, Y );
;
```

אבל לדאג שסדר החיבור יהיה לפי הסדר שרשום ב-

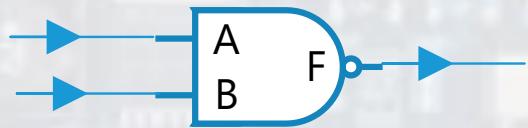
component

דוגמה 2 - מקרים מיוחדים של חיבורים



- 1^ז יוגדר כ- buffer (הדק מוצא הבניתן לקריאה על יד 2^ע), אפשר להגדיר גם כ- **tout** או להגדיר כמוצא **tin** ולהוסיף סיגנל פנימי לאותו הדק.
- 2^ז יוגדר כ-**tout** (יציאה מ- 4^ע וכינוסה ל- 3^ע)
- מוגדרים סיגנלים פנימיים, כולל 3^{טוויס} שאלוי נכתוב פקודת השמה של '1'

1. תוכניות הרכיבים הפנימיים



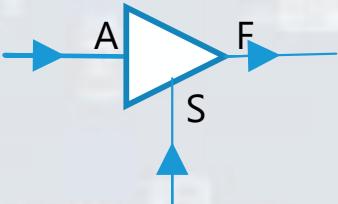
```

library IEEE;
use IEEE.std_logic_1164.all;

entity NAND_2 is
    port (A,B :in std_logic;
          F :out std_logic );
end ;

architecture arc_nand of NAND_2
is
begin
    F<= A nand B;
end;

```



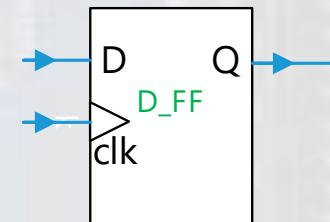
```

library IEEE;
use IEEE.std_logic_1164.all;

entity BUF is
    port (A,S :in std_logic;
          F :out std_logic );
end ;

architecture arc_buf of BUF is
begin
    F<= A when S='1' else 'Z';
end;

```



```

library IEEE;
use IEEE.std_logic_1164.all;

entity D_FF is
    port (D,clk :in std_logic;
          Q :out std_logic );
end ;

architecture arc_dff of D_FF is
begin
    process(clk)
    begin
        if rising_edge(clk) then
            Q<=D;
        end if;
    end process;
end;

```

```

library IEEE;
use IEEE.std_logic_1164.all;

entity TopTest is
    port(Din,clk,s : in std_logic;
        y1 : buffer std_logic;
        y2 : inout std_logic;
        y3 : out std_logic);
end;

architecture arc_top of TopTest is
    signal sig1,sig2,sig3: std_logic;

    component d_ff
        port (D,clk :in std_logic;
              Q :out std_logic );
    end component;

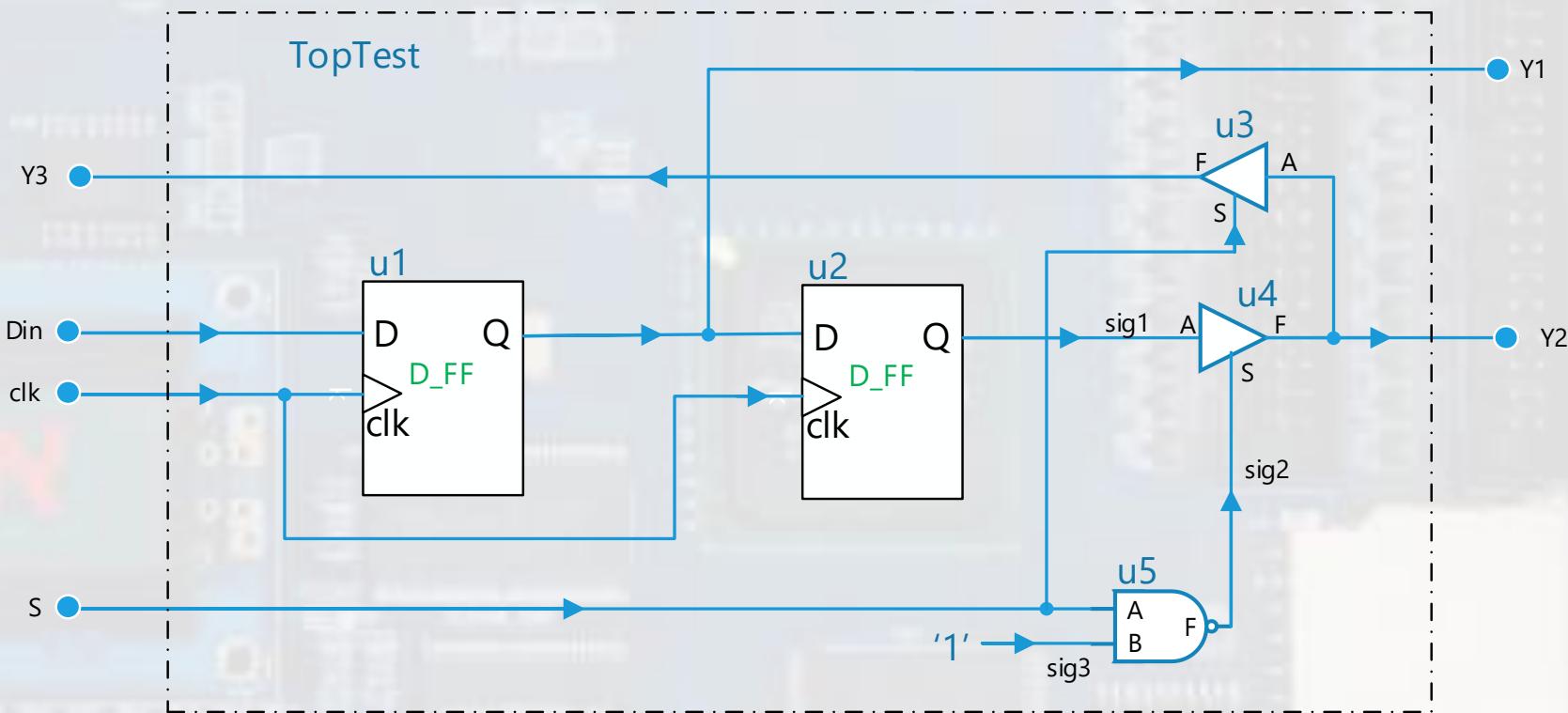
    component BUF
        port (A,S :in std_logic;
              F :out std_logic );
    end component;

    component NAND_2
        port (A,B :in std_logic;
              F :out std_logic );
    end component;

begin
    u1: d_ff  port map (Din, clk, y1);
    u2: d_ff  port map (y1, clk, sig1);
    u3: BUF   port map (y2, S, y3);
    u4: BUF   port map (sig1,sig2, y2);
    u5: NAND_2 port map (S, sig3, sig2);
    sig3<= '1';
end;

```

2. חיבור המרכיבת הראשית



תיאור מבני כללי באמצעות generic

שימושי בתיאור רכיב כללי, היכול לקבל פרמטרים שונים כמו:

- זמן השהייה לצורכי סימולציה.
- רוחב אOTTות הרכיב.

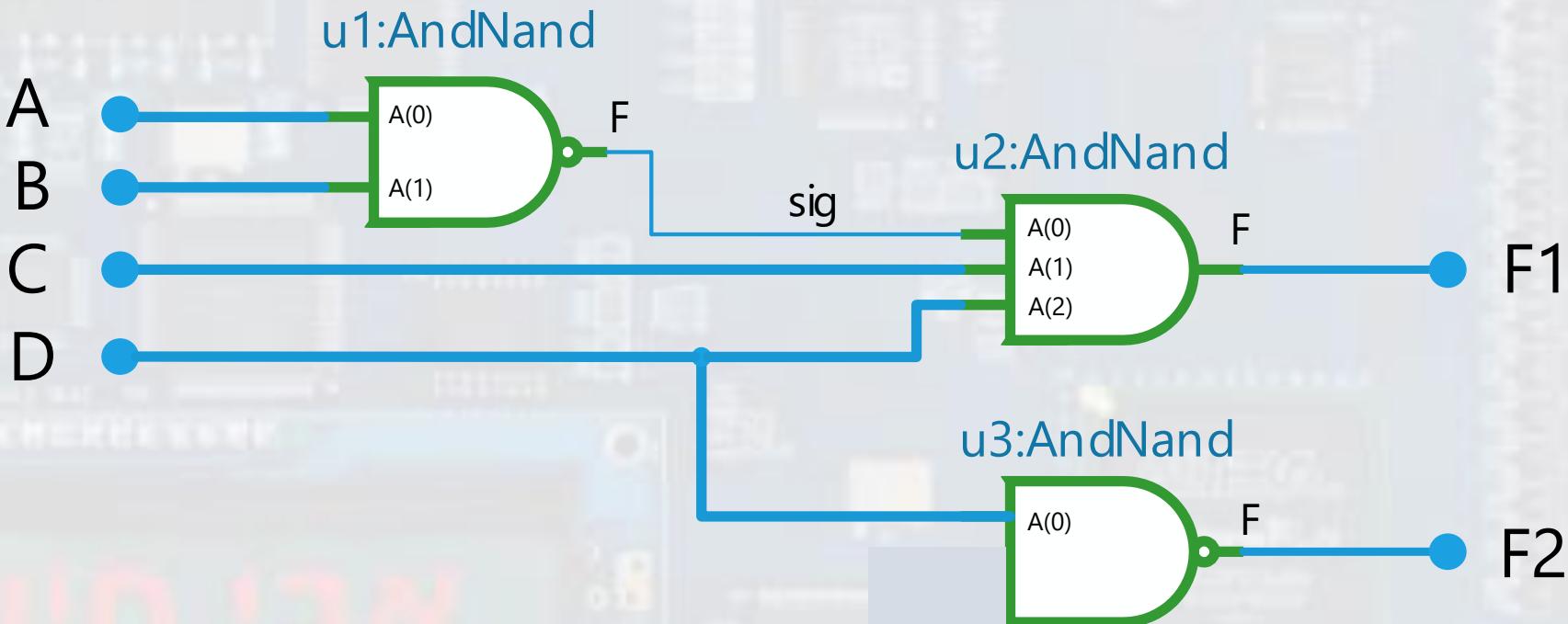
```
entity entity_name is
  generic (generic_list);
  port      (port_list);
end entity_name;

component component_name
  generic (generic_list);
  port (port_list);
end component;

instance_label: component_name
  generic map
  (generic_association_list)
    port map
  (port_association_list);
```

תחבר

דוגמאות



- כל הרכיבים אותו סוג.
- באמצעות generic נגדיר מספר הדקי הכניסה וסוג מוצא (עם היפוך או לא).

```

entity AndNand is
    generic (N : integer :=2;
             type_gate: bit := '0'  ); -- 0 and , 1 nand
    port      (A : in bit_vector(N-1 downto 0);
                F : out bit);
end;

architecture arc_gate of AndNand is
begin
    process(A)
        variable Fout: bit;
    begin
        Fout:='1';
        for i in 0 to N-1 loop
            Fout:=Fout and A(i);
        end loop;
        if type_gate='1' then
            Fout:= not Fout ;
        end if;
        F <= Fout;
    end process;
end;

```

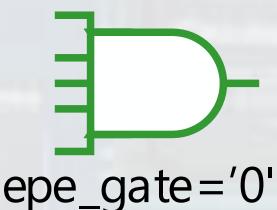
1. נכתב קוד לרכיב האנדי

רכיב AndNand יכול להיות:

.1. שער AND או NAND

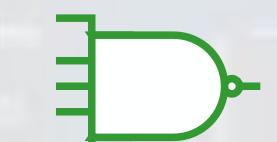
.2. מספר הדקי הכניסה מ-1 עד

למספר N



N כניסה

type_gate='0'



N כניסה

type_gate='1'

```

entity top_AndNand is
    port      (A,B,C,D : in bit;
               F1,F2 : out bit);
end;

architecture arc_top of top_AndNand is
    signal sig: bit;

    component AndNand
        generic (N : integer :=2;
                 type_gate: bit := '0'); -- 0 and , 1 nand
        port      (A : in bit_vector(N-1 downto 0);
                   F : out bit);
    end component;

begin
    u1: AndNand
        generic map (N=>2 ,type_gate =>'1')
        port map ( A(0) => A, A(1) => B, F =>sig );

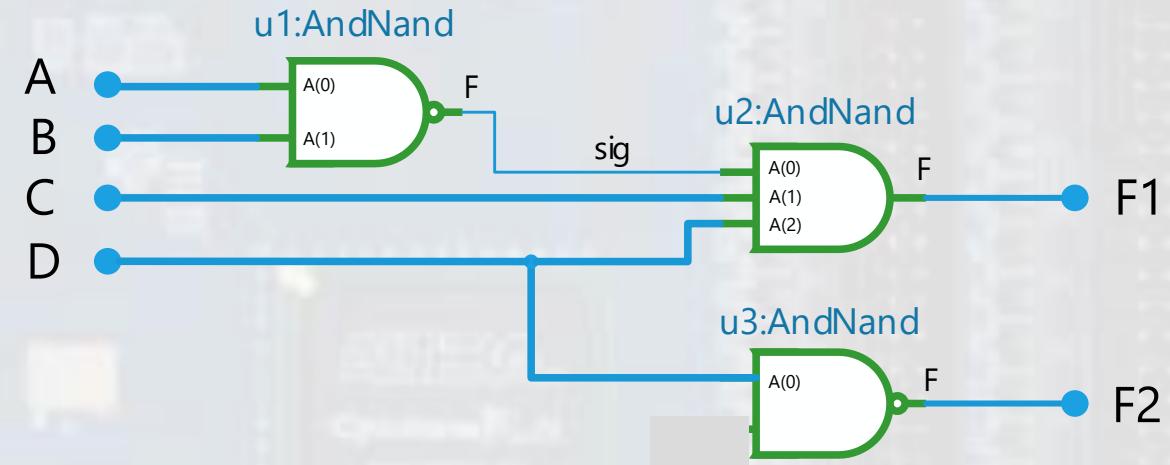
    u2: AndNand
        generic map (N=>3 ,type_gate =>'0')
        port map ( A(0) => sig, A(1) => C, A(2) => D,   F =>F1 );

    u3: AndNand
        generic map (N=>1 ,type_gate =>'1')
        port map ( A(0) => D ,   F =>F2);

end;

```

2. חיבור הכללי



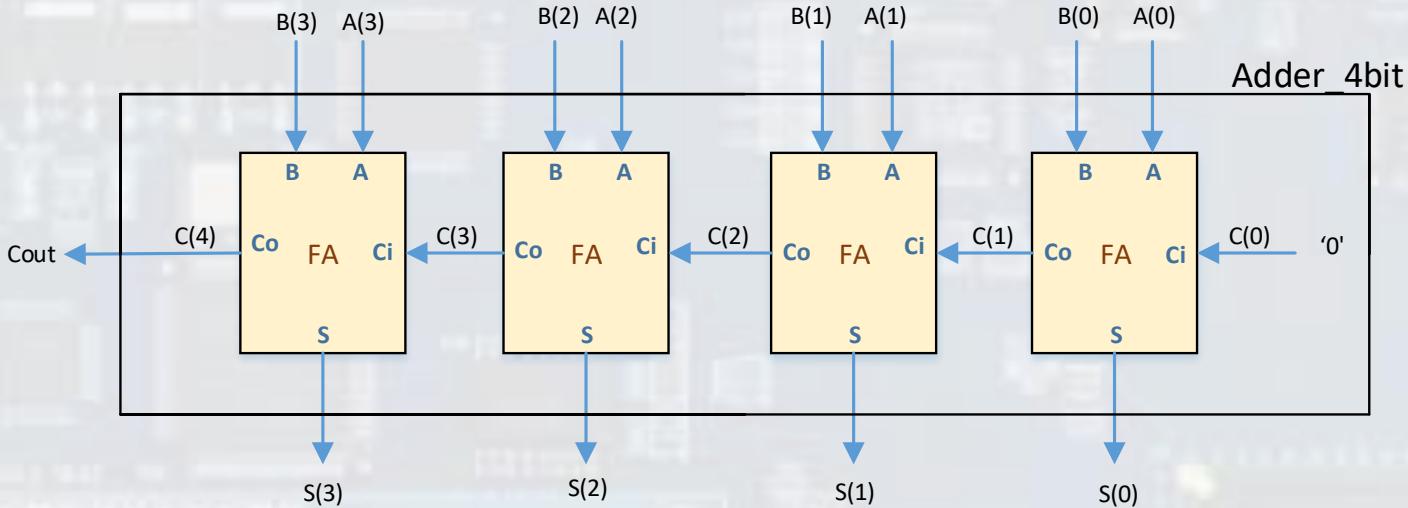
שכפול של תיאור מבני באמצעות פקודת generate

- ניתן לשכפל מבנים באמצעות לולאת for ופקודת generate במקום לרשום שורות רבות.
- השכפול יבוצע כאשר קיימים אותם סוגי רכיבים ונitin להשתמש באותות הכניסה והיציאה זהים עם אינדקס משתנה.
- אותן או הרכיבים שאינם זהים ירשמו בנפרד.
- ניתן לשלב ללולאת for כמו פקודת if

```
label: for parameter in range generate
    concurrent statements
end generate label;
```

תחביר

דוגמת שכפול 1



```

entity FA is
  port(A,B,Ci : in bit;
       S,Co : out bit);
end;

architecture arc_FA of FA is
begin
  S<= (A xor B) xor Ci;
  Co <= (A and B) or (A and Ci) or (B and Ci);

end;

```

- נתון הרכיב FA שאותו נחבר לפי האיר
- כל רכיב FA באיר מחובר לכניות B,A
לייצאה S ולאות C עם אינדקס שימושתנה
- לאות (0) C נחבר את הערך '0'
- מוצא Cout יקבל את הערך של (C(4)

```

entity Adder_4bit is
  port(A,B : in bit_vector(3 downto 0);
       Cout : out bit;
       S: out bit_vector(3 downto 0));
end;

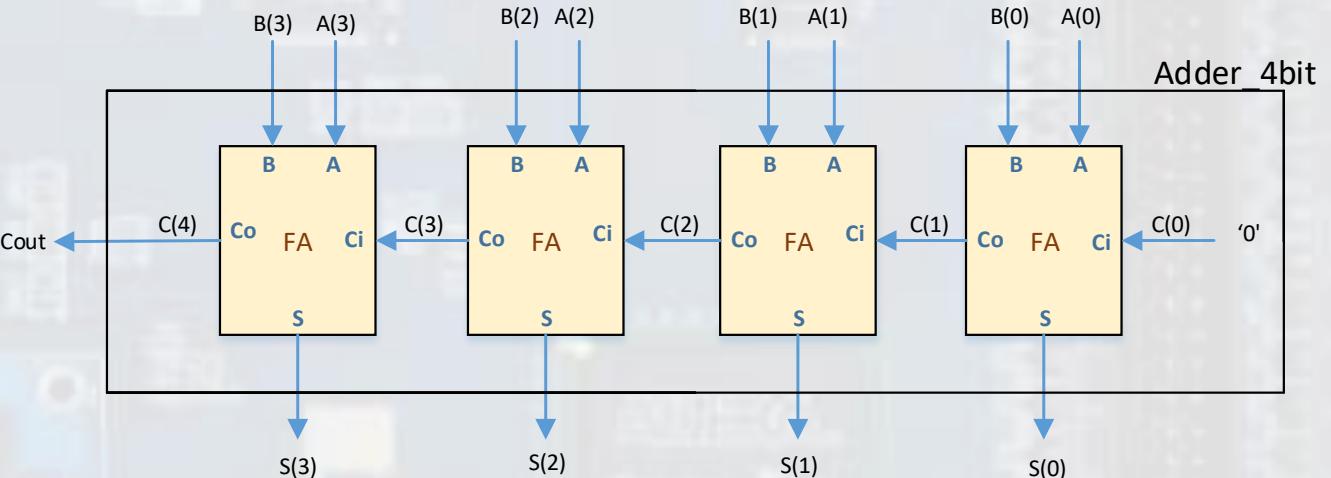
architecture arc_Adder of Adder_4bit is
  signal C: bit_vector(4 downto 0);

  component FA
    port(A,B,Ci: in bit;
         S, Co : out bit);
  end component;

begin
  C(0)<='0';
  Cout<= C(4);

  AdderGen: for i in 0 to 3 generate
    u: FA port map(A=>A(i),B=>B(i),Ci=>C(i),S=>S(i),Co=>C(i+1));
  end generate;
end;

```



פקודה אחת המחליפה את התוכנית הבאה:

```

u0: FA port map(A=>A(0),B=>B(0),Ci=>C(0),S=>S(0),Co=>C(1));
u1: FA port map(A=>A(1),B=>B(1),Ci=>C(1),S=>S(1),Co=>C(2));
u2: FA port map(A=>A(2),B=>B(2),Ci=>C(2),S=>S(2),Co=>C(3));
u3: FA port map(A=>A(3),B=>B(3),Ci=>C(3),S=>S(3),Co=>C(4));

```

```

entity and_or is
  port(A:in bit_vector(6 downto 0);
       F:out bit);
end;

architecture arc of and_or is

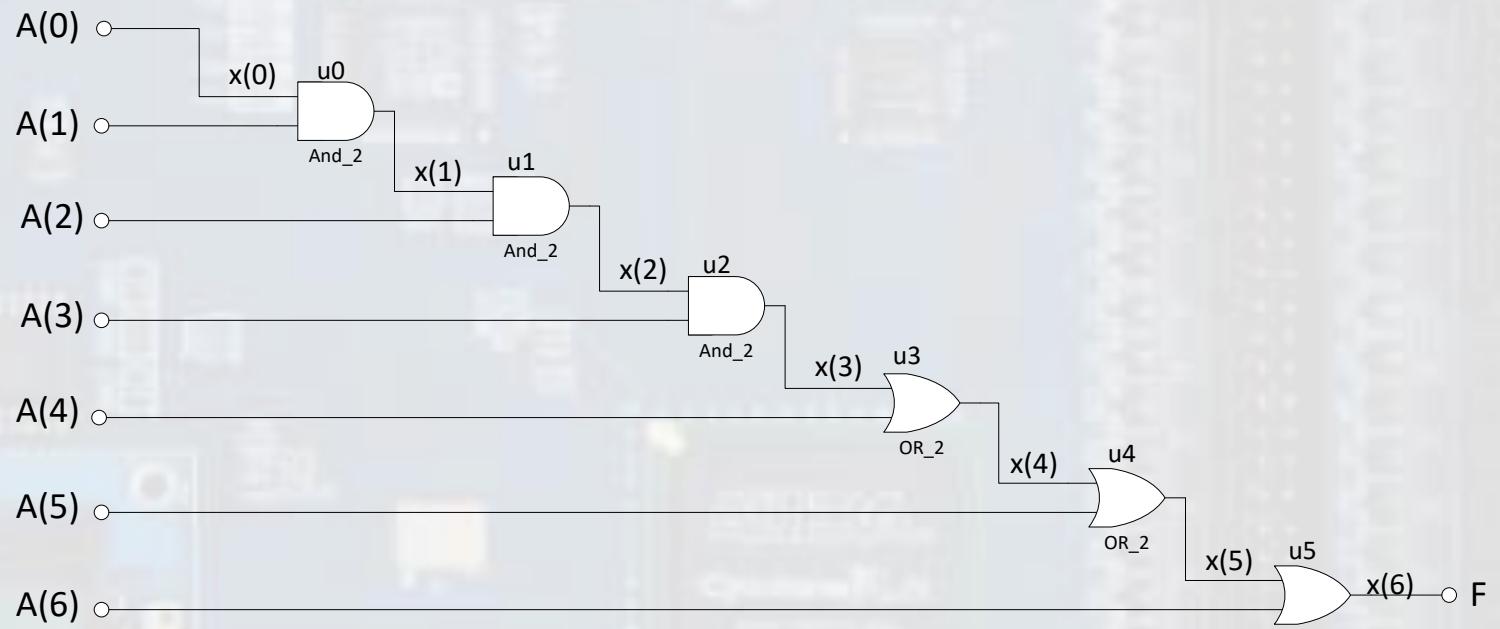
  component and_2
    port(a,b:in bit;
         f:out bit);
  end component;

  component or_2
    port(a,b:in bit;
         f:out bit);
  end component;

  signal x: bit_vector(6 downto 0);
begin
  x(0)<=A(0);
  F<=x(6);
  gen_gate: for i in 0 to 5 generate
    and2: if i<3 generate
      u_and:and_2 port map (A=>x(i),B=>A(i+1),F=>x(i+1));
    end generate and2;
    or2: if i>2 generate
      u_or:or_2 port map (A=>x(i),B=>A(i+1),F=>x(i+1));
    end generate or2;
  end generate gen_gate;
end;

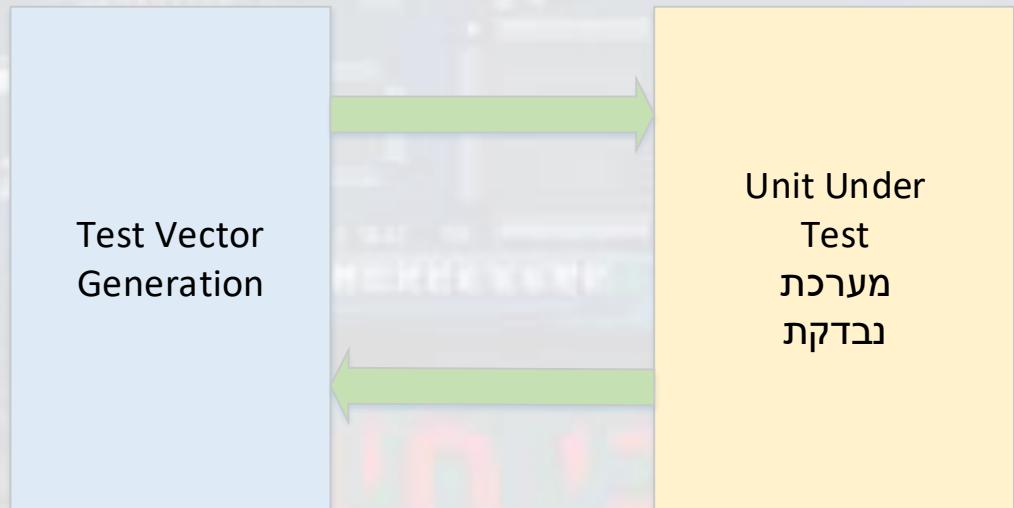
```

דוגמת שכפול 2



סימולציה

Test Bench



תפקיד Test Bench היא לבדוק מערכת על ידי הזרקת אוטומטית אליה ובדיקה התוצאות.

לצורך כך יהיה שימוש:

1. פקודות הודעות למסר – report, assert –
2. פסוקי wait בתוך תהליכיים
3. פסוקים נוספים לחולל אותן של הסימולטור

פקודות מסך

פקודת assert יכולה לשלוח הודעה התראה עבור מצבים שונים.

assert <boolean condition> report <message_string> severity <severity_level>;

- כאשר התנאי הוא false, המחרוזת נשלחת למסך ודרגת החומרה יכולה להיות אחת מ-4 לפי הסדר :
 - note, warning, error, failure
- דרגת החומרה קובעת, רק אם הסימולציה תיעזר ב- failure
- אם נכתבת התנאי יתקיים רק כאשר הוא עבר מצב true/false
- אם בטור התהלייר –בדיקה התנאי false תבוצע בזמן שהפקודה רשומה בתהלייר.

רישום מקוצר לשיליחה למסך

report <message_string>

פקודות wait

wait on – פעולה מתבצעת כאשר אחד האותות משתנים
wait on <signal_name1>, <signal_name2> ...;

דוגמה , המתן לשינוי באותות A או B
wait on A, B ;

wait until – פעולה מתבצעת, כאשר התנאי מתקיים
wait until <condition>;

דוגמה , המתן לעליית שעון
wait until clock'event and clock='1' ;

wait for – השהיית הפעולה
wait for <time_value>;

דוגמה , המתן 100 ננו שניות
wait on 100ns ;

Wait; המתן לצמימות

שילוב של הוראות wait

wait on <signal_name1> until <condition>;

דוגמה

```
process
begin
  wait on A, B until CLK = '1';
end process;
```

התהיליך מתבצע כאשר יש שינוי באחד האותות A או B, אבל רק אם הערך של CLK שווה ל-'1' לוגי.

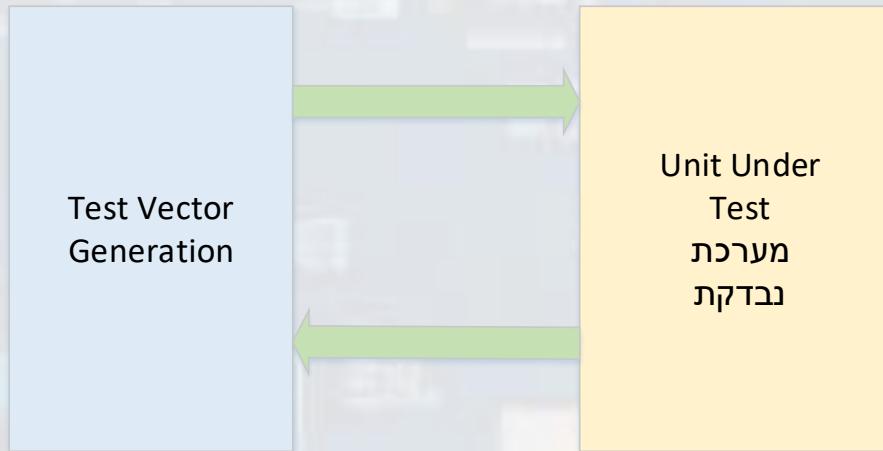
wait on <signal_name1> until <condition> for <time_value>;

דוגמה

```
process
begin
  wait on A, B until CLK = '1' for 100ns;
end process;
```

התהיליך מתבצע כאשר יש שינוי באחד האותות A או B, אבל רק אם הערך של CLK שווה ל-'1' לוגי או שעבר זמן של 100ns.

כתיבת Test Bench



- חיבור בין המערכת הנבדקת לבין המערכת הבודקת
נעשית עם כללי החיבור המבני (port map).
- למערכת הבודקת אין כניסה או יציאות, אלא סיגנלים
בלבד שבהם אנו מחוללים אותן שונות וקוראים את
התגובה מהמערכת הנבדקת.

תוכנית המונה הנבדק

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

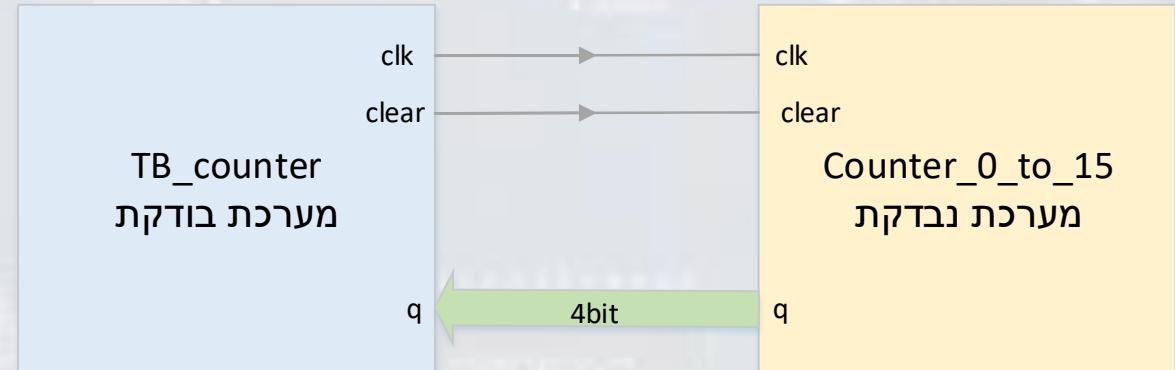
entity Counter_0_to_15 is
    port(clk ,clear: in std_logic;
          q: buffer std_logic_vector(3 downto 0);
end;

architecture arc_count of Counter_0_to_15 is
begin

    process(clk)
    begin
        if rising_edge(clk) then
            if clear='1' then
                q<="0000";
            else
                q<=q+1;
            end if;
        end if;
    end process;
end;

```

דוגמה ראשונה



- בדוגמה ניצר שעון למונח הנוכחי.
- נספַּך אוט איפו
- נקרא את המוצא Q של המונח

תוכנית הבדיקה

```

library IEEE;
use IEEE.std_logic_1164.all;

entity tb_counter is
end;

architecture arc_tb of tb_counter is
    signal clk ,clear :std_logic :='0';
    signal q:  std_logic_vector(3 downto 0);

    component Counter_0_to_15
        port(clk ,clear: in std_logic;
              q: buffer std_logic_vector(3 downto 0));
    end component;

begin
    u: Counter_0_to_15 port map (clk => clk, clear => clear, q=>q);

    clear <= '1', '0' after 163 ns;
    assert (Q /= "1111") report "Q=15" severity note;

process
begin
    clk<= '0';
    wait for 50ns;
    clk<= '1';
    wait for 50ns;
end process;

process
begin
    wait until rising_edge(clk);
    assert not(clear='1') report "clear counter" severity note;
end process;
end;

```

פקודה ישירה המספקת להדק clear='1'
לוגי מזמן 0 עד 163ns וירד ל-'0' לוגי

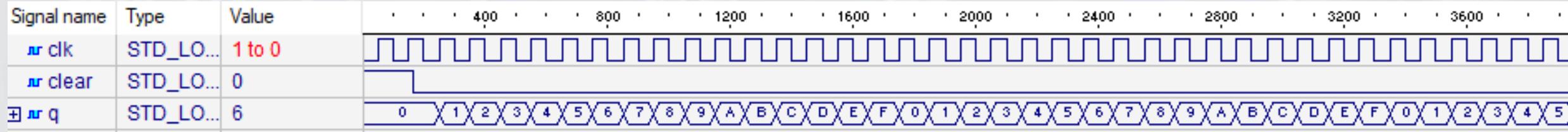
מציאה הودעה Q=15 כאשר Q משתנה לערך
"1111"

מחולל גל מחזורי בתדר 10MHz

בודק בעליית שעון, אם clear='1'
במידה וכן מציג את ההודעה:
clear counter

תוצאות הסימולציה

סימולציה גלים



פלט המסר

```

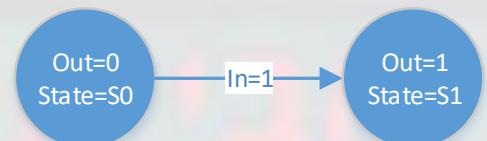
# EXECUTION:: NOTE      : clear counter
# EXECUTION:: Time: 50 ns, Iteration: 1, Instance: /tb_counter, Process: line_31.
# EXECUTION:: NOTE      : clear counter
# EXECUTION:: Time: 150 ns, Iteration: 1, Instance: /tb_counter, Process: line_31.
# EXECUTION:: NOTE      : Q=15
# EXECUTION:: Time: 1650 ns, Iteration: 2, Instance: /tb_counter, Process: line_21.
# EXECUTION:: NOTE      : Q=15
# EXECUTION:: Time: 3250 ns, Iteration: 2, Instance: /tb_counter, Process: line_21.
# KERNEL: stopped at time: 4 us

```

מכונת מצבים – State Machine

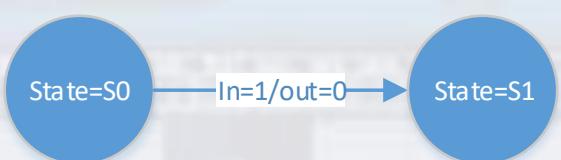
- מכונת מצבים מאפשרת לתוכנן מערכות מורכבות באופן יעיל יותר.
- כל מצב מתאר פעולה של מערכת המפעילה יציאות שונות.
- מעבר במצב למצב תלוי בנסיבות המערכת או בזמן.
- מוצא המערכת תלוי בסוג המכונה:
 1. מכונה מסוג MOORE – מוצא תלוי אך ורק במצב המכונה.

$$\text{out} = f(\text{state})$$



בתרשים רושמים את המוצא בתוך הבועה

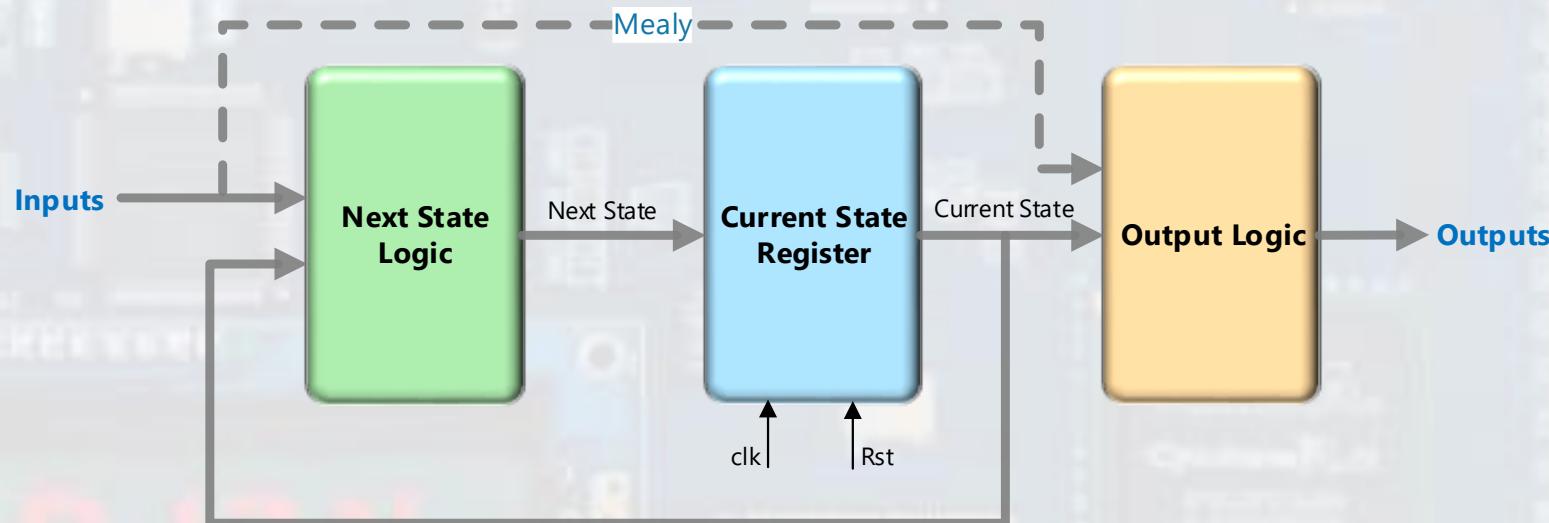
2. מכונה מסוג MEALY – מוצא המערכת תלוי במצב המכונה וגם בנסיבות המערכת.



$$\text{out} = f(\text{in}, \text{state})$$

בתרשים רושמים את המוצא במעבר לבועה הבא

מכונה מצבים



- מצב המכונה **Current State** וועבר למצב הבא **Next State** בסyncron השעון.
- המצב הבא תלוי במצב הנוכחי ובכניסת המערכת.
- מוצא תלוי במצב הנוכחי של המכונה בסוג **Moore** או **Mealy** גם בכניסת המערכת.

```

library IEEE;
use IEEE.std_logic_1164.all;

entity moore110 is
    port ( clk,clear,x: in std_logic;
            y: out std_logic);
end moore110;

architecture arc_moor of moore110 is
    type State_type is (S0,S1, S2, S3);
    signal CurrentState, NextState: State_type;
begin
    process (clk)
    begin
        if rising_edge(clk) then
            if clear='1' then    CurrentState <= S0;
            else    CurrentState <= NextState;
            end if;
        end if;
    end process;

    process(CurrentState, x)
    begin
        case CurrentState is
            when S0 => if x='1' then NextState <= S1;
            else    NextState <= S0; end if;

            when S1 => if x='1' then NextState <= S2;
            else    NextState <= S0; end if;

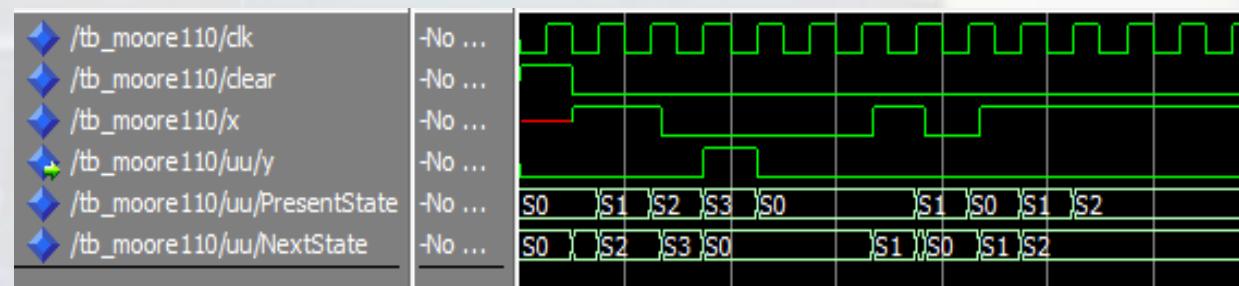
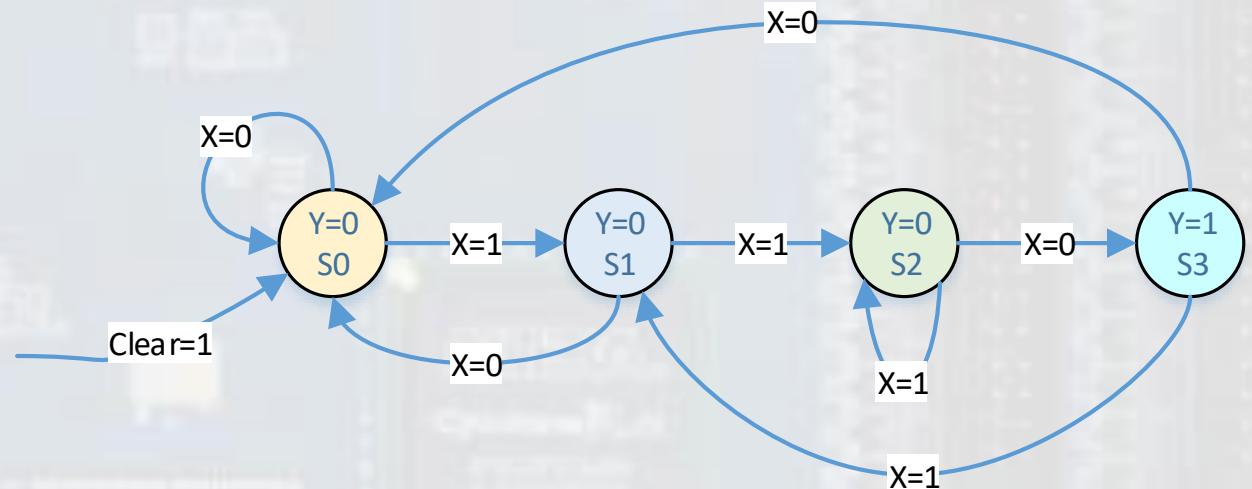
            when S2 => if x='0' then NextState <= S3;
            else    NextState <= S2; end if;

            when S3 => if x='0' then NextState <= S0;
            else    NextState <= S1; end if;
        end case;
    end process;

    y<='1' when CurrentState = s3 else '0';
end arc_moor;

```

דוגמה למכונת Moore המגלת את הסדרה 110



```

library IEEE;
use IEEE.std_logic_1164.all;

entity mealy110 is
    port ( clk,clear,x: in std_logic;
            y: out std_logic);
end mealy110;

architecture arc_mealy of mealy110 is
type State_type is (S0,S1, S2);
signal CurrentState, NextState: State_type;
begin
begin
    process (clk)
    begin
        if rising_edge(clk) then
            if clear='1' then      CurrentState <= S0;
            else      CurrentState <= NextState;
            end if;
        end if;
    end process;

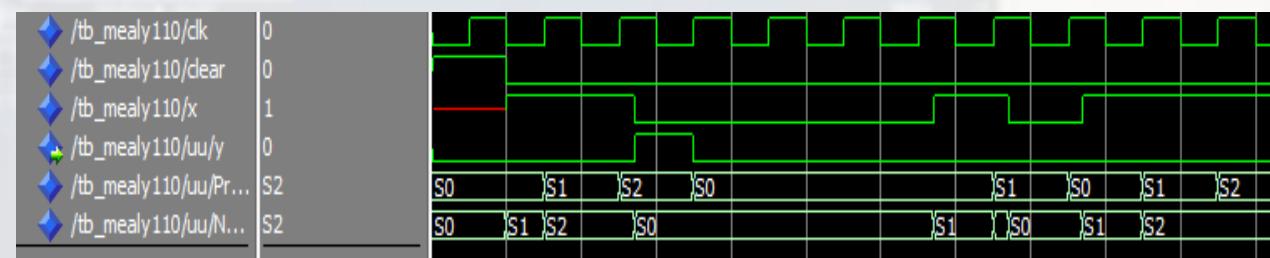
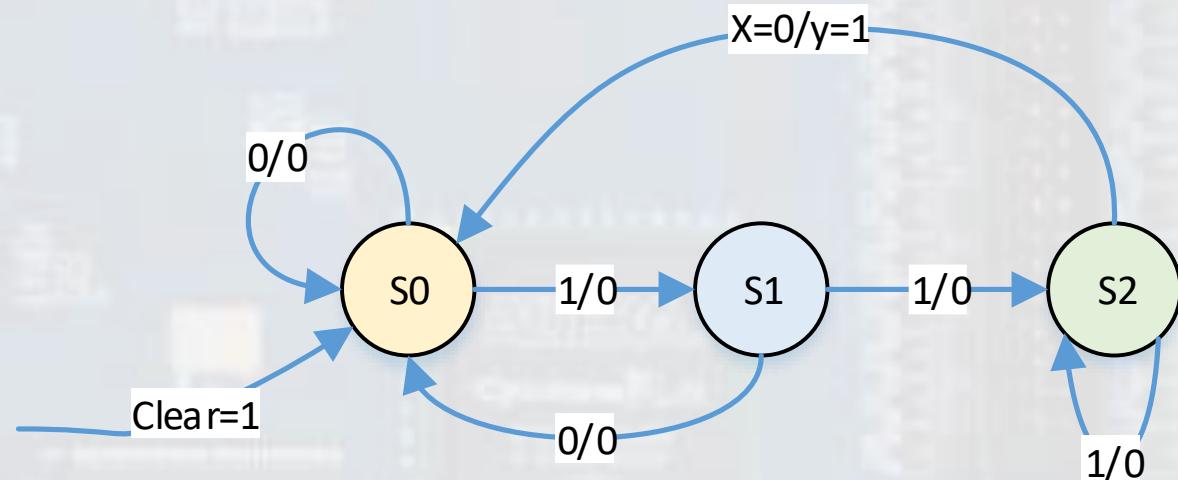
    process(CurrentState, x)
    begin
        case CurrentState is
            when S0 => if x='1' then NextState <= S1;
                         else      NextState <= S0;      end if;

            when S1 => if x='1' then NextState <= S2;
                         else      NextState <= S0;      end if;

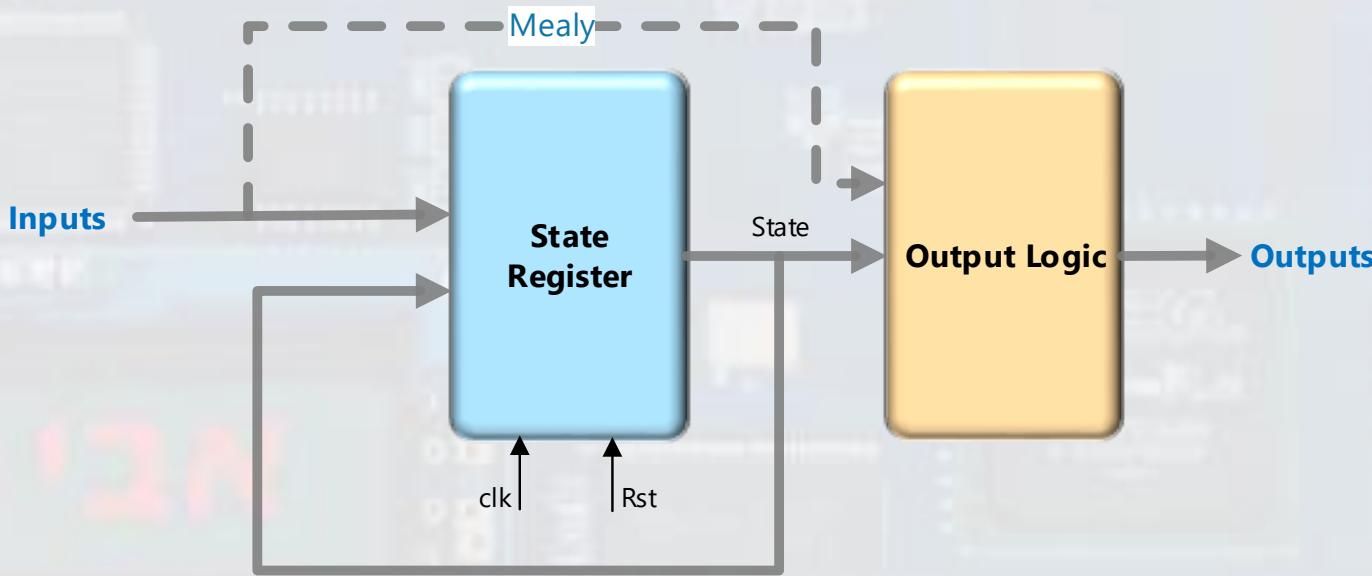
            when S2 => if x='0' then NextState <= S0;
                         else      NextState <= S2;      end if;
        end case;
    end process;
end arc_mealy;

```

דוגמה למכונת Mealy המגלת את הסדרה 110



תיאור המconda עם בלבד Current State עם בלבד



- נגדיר את State כ- **Current State** לשם נוחות.
- ניתן לתאר את המconda גם עם משתנה אחד בלבד.

דוגמה למכונה המגלה 110 באמצעות אחד בלבד

Moore

```
entity moore110 is
  port ( clk,clear,x: in bit;
         y: out bit);
end ;

architecture moor of moore110 is
  type State_type is (S0,S1, S2, S3);
  signal State: State_type;
begin
  process (clk)
  begin
    if clk'event and clk = '1' then
      if clear='1' then    State <= S0;
      else
        case State is
          when S0 => if x='1' then    State <= S1;      end if;

          when S1 => if x='1' then State <= S2;
          else           State <= S0;      end if;

          when S2 =>     if x='0' then State <= S3;      end if;

          when S3 =>     if x='0' then State <= S0;
          else           State <= S1;      end if;
        end case;
      end if;
    end if;
  end process;
  -----
  y<='1' when state=s3 else '0';
end ;
```

Mealy

```
entity AsynMealy110 is
  port (      clk,clear,x: in bit;
              y: out bit);
end ;

architecture mealy of AsynMealy110 is
  type State_type is (S0,S1, S2);
  signal State: State_type;
begin
  process (clk)
  begin
    if clk'event and clk = '1' then
      if clear='1' then    State <= S0;
      else
        case State is
          when S0 => if x='1' then State <= S1; end if;

          when S1 => if x='1' then State <= S2;
          else State <= S0; end if;

          when S2 =>     if x='0' then State <= S0; end if;
        end case;
      end if;
    end if;
  end process;
  -----
  y<='1' when state=s2 and x='0' else '0';
end ;
```

מקורות

- זולבסקי עמו – "לימוד שפת VHDL למולzieה וסינתזה" שורש הוצאה לאור ; מהדורה שנייה , (2012)