תקציר VHDL

מבנה בסיסי של תוכנית

מעגל דיגיטלי מתואר כקופסא עם כניסות ויציאות.

קופסא – מתוארת בשפה כישות entity

כניסה הפשוטה ביותר in מסוג **BIT** (0 או 1 לוגי)

יציאה הפשוטה ביותר out מסוג **BIT** (0 או 1 לוגי)

תפקוד המתאר את הקשר בין הכניסות והיציאות מוגדר ע"י המלה ARCHITECTURE

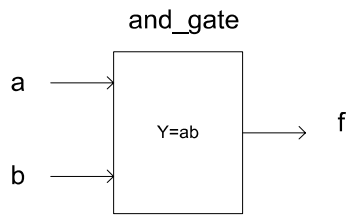
דוגמא לשער AND

```
entity AND_GATE  is
        port(A,B : in BIT;
                f: out BIT);
end AND_GATE;

architecture TWO_INPUTS of AND_GATE  is
begin

    f<=A and B;

end TWO_INPUTS;
```



<u>הסבר</u>

- שם הקופסא - entity
- מבואות ומוצאים בתוך הסוגריים של port, אחרי סוגריים – נקודה פסיק.
- סיום של entity end ושם AND_GATE (אפשר בלי שם) וסיום עם נקודה פסיק.
- לחלק התפקודי שם , בדוגמא נבחר TWO_INPUTS
- לאחר begin הפעולה עם השמה =<
- סיום עם המילה end ושם TWO_INPUTS (אפשר בלי שם) וסיום עם נקודה פסיק.

סוגי כניסות ויציאות

IN - כניסה הניתן לקריאה בלבד.

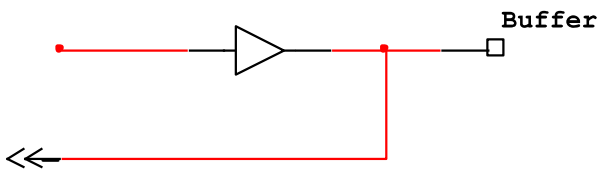
OUT - יציאה – הניתן לכתיבה בלבד.

קיימים עוד שני סוגי -PORT

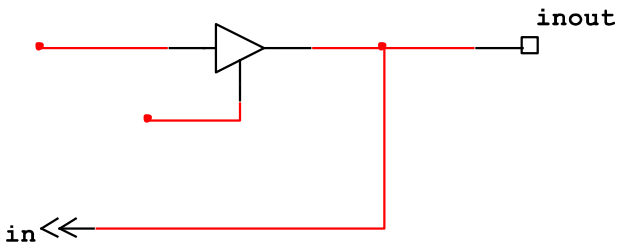
- Buffer - מוצא הניתן לכתיבה ולקריאה
- Inout – קו נתונים דו כווני הניתן לכתיבה וקריאה

מה ההבדל בין סוגי ה- PORT

Buffer - זהו PORT יציאה בלבד ולא ניתן לחבר אליו רכיב קלט, הקריאה מהיציאה היא לצורך פנימי בלבד לשם התניות או חיבור לכניסה פנימית .



Inout – זהו PORT יציאה וכניסה וניתן לחבר אליו רכיב קלט או פלט, כאשר הוא במצב של קלט, צריך לדאוג לעכבה גבוהה של הפלט הפנימי (אם משתמשים בעכבה גבוהה צריך להשתמש במשתנה מסוג **STD_LOGIC**).



משתנים

משתנה מסוג BIT

משתנה המכיל שני '0' או '1' ערכים והיא נרשמת בין גרש ימין לגרש שמאל .

משתנה מסוג BIT_VECTOR

מערך חד ממדי של משתנים מסוג **BIT** הניתן לכתיבה וקריאה, גם במיעון סיבית.

דוגמא להגדרה: Data :in **bit_vector** (7 downto 0)

בדוגמא מוגדר וקטור בגודל 8 סיביות

Data(7)	Data(6)	Data(5)	Data(4)	Data(3)	Data(2)	Data(1)	Data(0)
							LSB

דוגמאות להשמה:

```
Y<="11000011";              -- השמה בבינארי
Y<=x"C3";                   -- השמה בהקסה דצימלי
Y<=(others =>'0');          -- הצבה 0 לכל הסיביות
Y<=('1','1',OTHERS=>'0');   -- הצבה 11 לסיביות 7,6 וכל השאר 0 לוגי
Y<=(A, B ,OTHERS=>'0');     -- הצבה ערך של A לסיבית 7,ערך של B לסיבית 6 וכל השאר 0
Y(5 downto 2)<="1011";      -- הצבת הערך רק לסיביות 5 עד 2
Y(2)<='1';                  -- הצבה 1 לסיבית 2 בלבד
Y<=A xor B;                 -- פעולה לוגית , A,B הם מאותו סוג וגודל כמו Y
```

משתנה מסוג INTEGER

בגודל עד 32 סיביות ומסוגל לקבל ערך בטווח: 2147483648- עד 2147483647

דוגמאות להגדרת המשתנה:

```
Data_1 : in integer range  12 downto 0;      -- 4bit
Data_2 :  in integer range 0 to 255;         -- 8bit
```

כללים לגבי המשתנה

- נכתב ללא גרש או גרשיים
- שימושי לפעולות אריתמטיות ולא לוגיות
- התוכנה מקצה מספר סיביות לפי התחום

<u>השמה למשתנה מסוג INTEGER</u>

```
Y<=25;       -- השמת ערך עשרוני
Y<=A + B;    -- פעולה אריתמטית , A,B משתנים מאותו סוג כמו Y
Y<=Y+1;      -- קידום ב-1
```

משתנה מרובת ערכים - STD_LOGIC

משתנה מוגדר תחת הספרייה – STD_LOGIC_1164

ההצהרה בתחילת התוכנית:

```
library IEEE;

use IEEE.STD_LOGIC_1164.all;
```

למשתנה מסוג זה יש 9 מצבים לוגים:

1. '1' לוגי
2. '0' לוגי
3. 'Z' – עכבה גבוהה (Z נרשם באות גדולה)
4. 'X' – לא ידוע , יכול להתקבל כתוצאה מהתנגשות במידע (X נרשם באות גדולה)
5. '-' – DON'T CARE אפשר לתת ערך זה למשתנה
6. 'U' – ערך לא מאותחל, ערך של משתנה לפני שבוצעה השמה (U נרשם באות גדולה)
7. 'L' – '0' לוגי חלש , ערך הקרוב לסף המוגדר כ- 0 לוגי
8. 'H' – '1' לוגי חלש , ערך הקרוב לסף המוגדר כ- 1 לוגי
9. 'W' – לא ידוע חלש ('X' חלש)

אופן הרישום

- הגדרה של **BIT** - במקום **BIT** רושמים **STD_LOGIC**

דוגמא:

```
port(  D0,D1 : in  STD_LOGIC;
        S : in  STD_LOGIC;
        Y : out  STD_LOGIC);
```

- וקטור מורכב ממערך של ביטים מסוג **STD_LOGIC** ונרשם בדומה **BIT_VECTOR** רגיל .

דוגמא:

```
a : in  STD_LOGIC_VECTOR (7 DOWNTO 0);

b : in  STD_LOGIC_VECTOR (0 to 7);
```

אופרטורים

אופרטור	הסבר
=	equivalence – שווה
/=	non-equivalence - שונה
<	less than -קטן מ
<=	less than or equal -קטן או שווה
>	greater than - גדול
>=	greater than or equal - גדול או שווה
+	Addition -חיבור
-	subtraction -חיסור
*	multiplication - כפל
/	division - חילוק
|	או
&	concatenation - שרשור
and or nand nor xor xnor not	אופרטורים לוגים

דרבון באות – עליה או ירידת האות

האם עליית שעון

```
if clk'EVENT AND clk ='1'
```

המתן לעליית שעון

```
wait until clk'EVENT AND clk ='1';
```

האם ירידת שעון

```
if clk'EVENT AND clk ='0'
```

המתן לירידת שעון

```
wait until clk'EVENT AND clk ='0';
```

כאשר עובדים עם אותות std_logic ניתן להשתמש גם :

if rising_edge(CLK) - האם עליית שעון מ- '0' ל- '1'

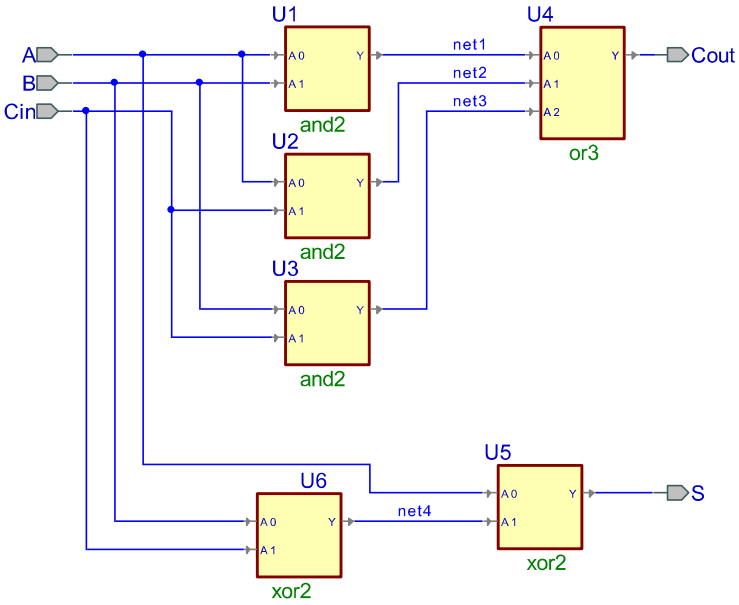
if falling_edge(CLK) - האם ירידת שעון מ- '1' ל- '0'

(נבדק רק השינוי 0 ו-1 למרות שיש עוד מצבים X,Z,,,)

תכנון היררכי

חיבור מודולים מוכנים- component אחד לשני

דוגמא – תכנון מסכם מלא – FA




```
entity f_a is
  port( A,B,Cin:     in bit;
        Cout,S: out bit);
end f_a;

architecture f_a of f_a is

---- Component declarations
component and_2
  port ( A0,A1 : in bit;
         Y  : out bit);
end component;

component or_3
  port ( A0,A1,A2 : in bit;
         Y :  out bit );
end component;

component xor_2
  port ( A0,A1 : in bit;
         Y :  out bit);
end component;

---- Signal declarations used on the diagram ----
signal net1,net2,net3,net4 : bit;
begin
----  Component instantiations  ----

U1 : and_2
  port map( A0 => A, A1 => B, Y  => net1 );

U2 : and_2
  port map( A0 => A, A1 => Cin, Y  => net2 );

U3 : and_2
  port map( A0 => B, A1 => Cin, Y  => net3 );

U4 : or_3
  port map( A0 => net1, A1 => net2, A2 => net3, Y  => Cout );

U5 : xor_2
  port map( A0 => A, A1 => net4,  Y  => S );

U6 : xor_2
  port map( A0 => B, A1 => Cin, Y  => net4 );

end f_a;
```

פקודות

פקודת when .... else

```
var    <=        value_1  when condition_1 else
                 value_2  when condition_2 else
                 value_3  when condition_3 else
                 .
                 .
                 value_n;
```

דוגמא רכיב 1 → MUX 2

```
entity  MUX_2_TO_1  is
        port(    D0,D1,S : in BIT;
                      Y : out BIT );
end  MUX_2_TO_1;


architecture  arc_Mux  of  MUX_2_TO_1  is
begin
        Y<= D0 when S='0' else  D1;
end   arc_Mux  ;
```

פקודת with .... select

```
with expression select
var    <=        value_1  when choice_1,
                 value_2  when choice_2,
                 value_3  when choice_3,
                 .
                 .
                 value_n  when others;
```

דוגמא רכיב 1 → MUX 2

```
entity MUX_2_TO_1 is
        port(
                D0,D1,S : in BIT;
                     Y : out BIT);
end MUX_2_TO_1;

architecture arc_Mux  of MUX_2_TO_1 is
begin
        with  S select
        Y<= D0 when '0',
        D1 when others;

end arc_Mux  ;
```

התהליך - PROCESS

- מאפשר כתיבה של בקרה טורית
- ניתן להשתמש בפקודות – LOOP, CASE, IF-ELSE
- ניתן להשתמש במשתני עזר פנימיים הנקראים VARIABLE והם מוגדרים בין PROCESS לבין begin . ההצבה במשתנה מתבצעת ע"י := ולא =>
- רגישויות -נרשם לאחר המלה PROCESS , כוללת אירועים באותות כמו: שינוי, עליה או ירידת שעון

התחביר

```
architecture ........... of ........... is
begin
        my_lable: PROCESS( רגישויות ,,,)
                variable .... : ........;
                begin
                ........
                 End PROCESS my_lable;
end .........;
```

תחביר פקודת if else

```
if (CONDITION1)     then  STATEMENT1 ;
elsif (CONDITION2) then  STATEMENT2 ;
else    STATEMENT3 ;
end if;
```

דוגמא:

```
if (S=0 and S=1)    then F <= A;   -- האפשרויות 0 וגם 1
elsif (S=3 or S=5) then F <= B; -- האפשרויות 3 או 5
else                     F <= C;
end if;
```

תחביר פקודת case

```
case  EXPRESSION  is
    when CHOICES1 =>  STATEMENTS1;
    when CHOICES2 =>  STATEMENTS2;
    when OTHERS   =>  STATEMENTS3;
end case;
```

דוגמא:

```
case SELECTOR is
    when  0 to 2 =>       -- האפשרויות 0,1,2
       F <= A;
    when 3 | 5   =>       -- האפשרויות 3 או 5
       F <= B;
    when OTHERS  =>
       F <= C;
end case;
```