

שפת תאור חומרה VHDL

תוכן עניינים

2	מבנה בסיסי של תוכנית
2	דוגמא לשער AND
3	אופרטורים לוגים נוספים
4	דוגמא- רכיב 1 \rightarrow MUX 2
5	תאור התנהגותי של מערכת
5	רכיב 1 \rightarrow MUX 2 בעזרת פקודת when ... else
6	רכיב 1 \rightarrow MUX 2 בעזרת פקודת with ... select
7	משתנה מסוג BIT_VECTOR
7	משתנה מסוג INTEGER
8	סוגי כניסות ויציאות
9	משתנה מרובת ערכים - STD_LOGIC
10	דוגמאות
10	מרבב 4 ל-1
10	צורה 1 – שימוש בשרשור &
10	צורה 2 – שימוש במשתנה s כ- integer
11	מפענח מ-BCD ל-7seg בעזרת פקודת with...select
12	מפענח מבינארי 5 ביטים ל-BCD
13	התהליך - PROCESS
13	תחביר פקודת if else
14	תחביר פקודת case
14	רשימת רגישויות
15	דוגמאות:
15	DFF הפועל בעליית שעון
15	DFF הפועל בעליית שעון עם reset אסינכרוני
16	DFF הפועל בעליית שעון עם reset ו- preset אסינכרוני
16	DFF הפועל בעליית שעון עם reset ו- preset סינכרוני
17	דוגמאות לתכנון מונים
17	מונה מעלה מ- 0 עד 255 עם איפוס סינכרוני
18	מונה מעלה/מטה מ- 0 עד 255 עם איפוס, טעינה ואפשרות סינכרוני
19	מונה מעלה מודולו 200
20	מונה מעלה BCD עם reset אסינכרוני ואפשרות סינכרוני
21	מכונת מצבים
21	דוגמא למכונת Moore
23	דוגמא למכונת Mealy
25	Mealy סינכרוני
27	תכנון היררכי
28	דוגמא – תכנון מסכם מלא – FA

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`

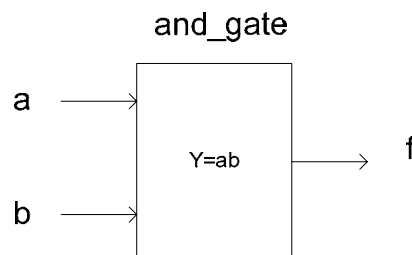
דוגמא לשער AND

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

architecture TWO_INPUTS of AND_GATE is
begin

    Y<=A and B;


end TWO_INPUTS;
```




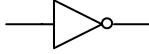
הסבר


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


אופרטורים לוגיים נוספים


 - and


 - or

 - Not

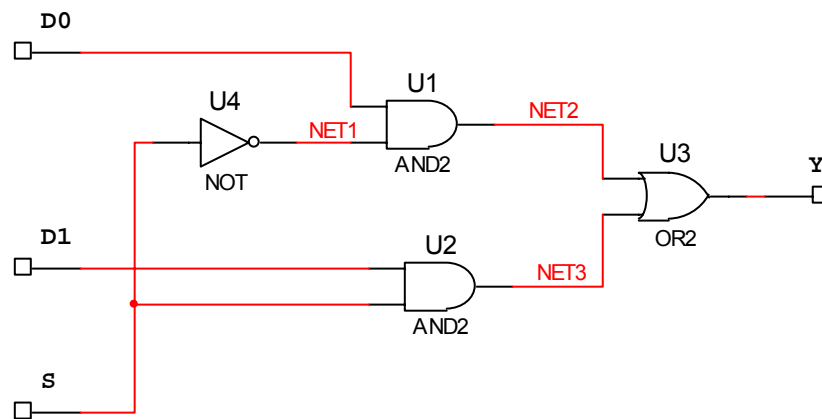
 - Nand

 - nor

 - xor

 - xnor

דוגמא- רכיב 1 → MUX 2



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

```
architecture EQUATION of MUX_2_TO_1 is
begin
```

```
    Y<=((not S)and D0) or (S and D1);
```

```
end EQUATION;
```

ניתן לתאר את אותה המערכת עם שימוש בסיגנלים פנימיים

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

```
architecture EQUATION of MUX_2_TO_1 is
  signal NET1,NET2,NET3:BIT;
```

```
begin
```

```
  NET1<=not S;
  NET2<=NET1 and D0;
  NET3<=S and D1;
  Y<=NET2 or NET3;
```

```
end EQUATION;
```

תאור התנהגותי של מערכת

ניתן לתאר את המערכת הלוגית כמערכת התנהגותית בעזרת 2 פקודות:

- פקודת `when else`

מבנה הפקודה:

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

- פקודת `with select`

מבנה הפקודה

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

נתאר את רכיב $1 \rightarrow 2$ MUX בעזרת שתי הפקודות:

רכיב $1 \rightarrow 2$ MUX בעזרת פקודת `when else`

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

architecture EQUATION of MUX_2_TO_1 is
begin

    Y<= D0 when S='0' else
        D1;

end EQUATION;
```

רכיב 1 → MUX 2 בעזרת פקודת **with select**

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

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

הערה: משתנה מסוג **BIT** מכיל שני ערכים והיא נרשמת בין גרש ימין לגרש שמאל : '0' או '1'

משתנה מסוג BIT_VECTOR

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

דוגמא להגדרה: `Data_1: 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:

```
Data<="11000011";           -- השמה בבינארי
Data<=x"C3";                -- השמה בהקס דצימלי
Data<=(others =>'0');        -- הצבה 0 לכל הסיביות
Data<=('1','1',OTHERS=>'1'); -- הצבה 11 לסיביות 7,6 וכל השאר 1 לוגי
Data(5 downto 2)<="1011";    -- הצבת הערך רק לסיביות 5 עד 2
Data(2)<='1';                -- הצבה 1 לסיבית 2 בלבד
```

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

סוגי כניסות ויציאות

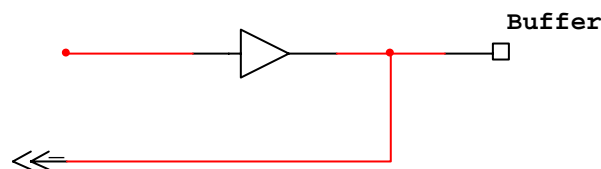
עד כה ראינו כניסה – **IN** הניתן לקריאה בלבד ויציאה – **OUT** הניתן לכתיבה בלבד

קיימים עוד שני סוגי **PORT**

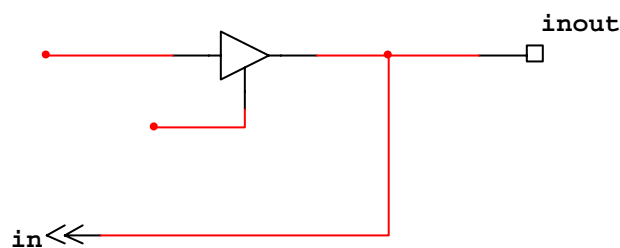
- **Buffer** - הניתן לכתיבה ולקריאה
- **Inout** – קו נתונים דו כיווני הניתן לכתיבה ולקריאה

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**).



בכל זאת ניתן להגדיר **PORT** מסוג **INOUT** כמשתנה **BIT** אבל לא ניתן לקבל עכבה גבוהה.

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 לוגיים:

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 .

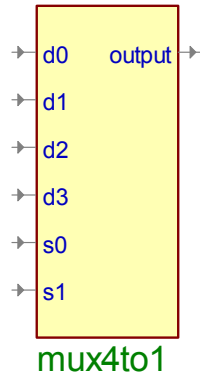
דוגמא:

```
a : in STD_LOGIC_VECTOR(7 DOWNT0 0);
b : in STD_LOGIC_VECTOR(0 to 7);
```

דוגמאות

מרבב 4 ל-1

<u>S1</u>	<u>S0</u>	<u>output</u>
0	0	d0
0	1	d1
1	0	d2
1	1	d3



צורה 1 – שימוש בשרשור &

```

ENTITY mux4to1 IS
    PORT
    (
        d0, d1, d2, d3      : IN BIT;
        s0,s1               : IN BIT;
        output              : OUT BIT
    );
END mux4to1;

ARCHITECTURE maxp1d OF mux4to1 IS
    signal sel:bit_vector(1 downto 0);
BEGIN
    sel<=(s1 & s0);

    WITH (sel) SELECT      -- creates a 4-to-1 multiplexer
        output <=  d0 WHEN "00",
                   d1 WHEN "01",
                   d2 WHEN "10",
                   d3 WHEN "11";

END maxp1d;

```

צורה 2 – שימוש במשתנה s כ- integer

```

ENTITY selsig IS
    PORT
    (
        d0, d1, d2, d3      : IN BIT;
        s                   : IN INTEGER RANGE 0 TO 3;
        output              : OUT BIT
    );
END selsig;

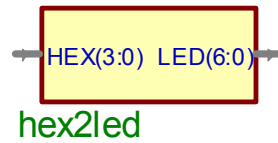
ARCHITECTURE maxp1d OF selsig IS
BEGIN

    WITH s SELECT          -- creates a 4-to-1 multiplexer
        output <=  d0 WHEN 0,
                   d1 WHEN 1,
                   d2 WHEN 2,
                   d3 WHEN 3;

END maxp1d;

```

מפענח מ-BCD ל-7seg בעזרת פקודת with....select



```

library IEEE;
use IEEE.std_logic_1164.all;

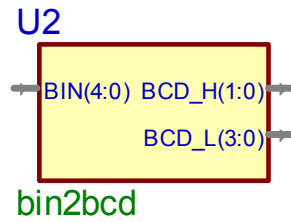
entity hex2led is
    port (
        HEX: in  STD_LOGIC_VECTOR (3 downto 0);
        LED: out STD_LOGIC_VECTOR (6 downto 0)
    );
end hex2led;

--}} End of automatically maintained section

architecture hex2led of hex2led is
    -- segment encoding
    --      0
    --      ---
    --  5 |   | 1
    --      ---  <- 6
    --  4 |   | 2
    --      ---
    --      3
begin
    with HEX select
        LED    <= "1111001" when "0001",  --1
        "0100100" when "0010",  --2
        "0110000" when "0011",  --3
        "0011001" when "0100",  --4
        "0010010" when "0101",  --5
        "0000010" when "0110",  --6
        "1111000" when "0111",  --7
        "0000000" when "1000",  --8
        "0010000" when "1001",  --9
        "0001000" when "1010",  --A
        "0000011" when "1011",  --b
        "1000110" when "1100",  --C
        "0100001" when "1101",  --d
        "0000110" when "1110",  --E
        "0001110" when "1111",  --F
        "1000000" when others;  --0
end hex2led;

```

מפענח מבינארי 5 ביטים ל-BCD



```

--VHDL model for Decoder of 5-bits Binary Numbers to Two BCD Digits
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity bin2bcd is
    port (
        BIN: in integer range 0 to 63;
        BCD_H: out STD_LOGIC_VECTOR (1 downto 0) ;
        BCD_L: out STD_LOGIC_VECTOR (3 downto 0));
end bin2bcd;

architecture BIN2BCD_ARCH of bin2bcd is
begin
    with BIN select
        BCD_L <=  "0001" when 1 | 11 | 21 | 31,
                  "0010" when 2 | 12 | 22,
                  "0011" when 3 | 13 | 23,
                  "0100" when 4 | 14 | 24,
                  "0101" when 5 | 15 | 25,
                  "0110" when 6 | 16 | 26,
                  "0111" when 7 | 17 | 27,
                  "1000" when 8 | 18 | 28,
                  "1001" when 9 | 19 | 29,
                  "0000" when others;

        BCD_H <=  "00" when (BIN < 10) else
                  "01" when (BIN < 20) else
                  "10" when (BIN < 30) else
                  "11";
end BIN2BCD_ARCH;

```

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 <=

התחביר

```
entity ..... is
  port(
    .....;
    .....;
  );
end .....;

architecture ..... of ..... is
begin
  my_label:PROCESS
    variable .... : .....;
    begin
      .....
      .....
    End PROCESS my_label;
end .....;
```

תחביר פקודת if else

```
if (CONDITION1) then
  STATEMENT ;
elsif (CONDITION2) then
  STATEMENT ;
else
  STATEMENT ;
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;
```

רשימת רגישויות

לאחר המלה **PROCESS** ניתן לרשום בסוגריים אותות הנקראים רשימת רגישויות.

רשימת הרגישויות כוללת אירועים באותות כמו: שינוי באות, עליה או ירידת שעון. במידה ויתקיים האירוע התוכנית הנמצאת ב- **PROCESS** תתבצע.

אם לא נרשם אירוע בסוגריים, התהליך מתבצע בכל מקרה .

דוגמאות:

DFF הפועל בעליית שעון

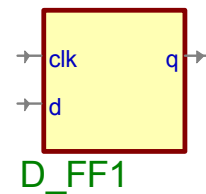
-- Register with active-high Clock

```

ENTITY D_FF1 IS
    PORT
    (
        d, clk      : IN BIT;
        q           : OUT BIT
    );
END D_FF1;

ARCHITECTURE d_ff OF D_FF1 IS
BEGIN
    PROCESS
    BEGIN
        WAIT UNTIL clk = '1';
        q1 <= d;
    END PROCESS;
END d_ff;

```



אם רוצים שה-DFF יעבוד בירידת שעון נרשום את השורה

```
WAIT UNTIL clk = '0';
```

DFF הפועל בעליית שעון עם reset אסינכרוני

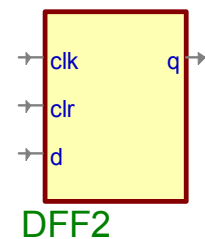
-- Register with active-high Clock & asynchronous Clear

```

ENTITY DFF2 IS
    PORT
    (
        d, clk, clr  : IN BIT;
        q           : OUT BIT
    );
END DFF2;

ARCHITECTURE d_ff OF DFF2 IS
BEGIN
    PROCESS (clk, clr)
    BEGIN
        IF clr = '1' THEN
            q <= '0';
        ELSIF clk'EVENT AND clk = '1' THEN
            q <= d;
        END IF;
    END PROCESS;
END d_ff;

```



DFF הפועל בעליית שעון עם reset ו- preset אסינכרוני

-- Register with active-high Clock & asynchronous Clear & Preset

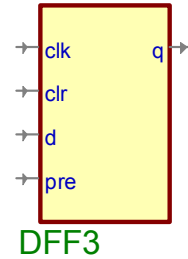
```

ENTITY DFF3 IS
  PORT
    (
      d, clk, clr, pre : IN BIT;
      q                : OUT BIT
    );
END DFF3;

ARCHITECTURE d_ff OF DFF3 IS
BEGIN

  PROCESS (clk, clr, pre)
  BEGIN
    IF clr = '1' THEN
      q <= '0';
    ELSIF pre = '1' THEN
      q <= '1';
    ELSIF clk'EVENT AND clk = '1' THEN
      q <= d;
    END IF;
  END PROCESS;
END d_ff;

```



DFF הפועל בעליית שעון עם reset ו- preset סינכרוני

כדי לקבל שהאיפוס יהיה סינכרוני ב- PROCESS רושמים בסוגריים רק את ה- clk והתנאי הראשון לבדוק את ה- clk

-- Register with active-high Clock & synchronous Clear & Preset

```

ENTITY DFF3 IS
  PORT
    (
      d, clk, clr, pre : IN BIT;
      q                : OUT BIT
    );
END DFF3;

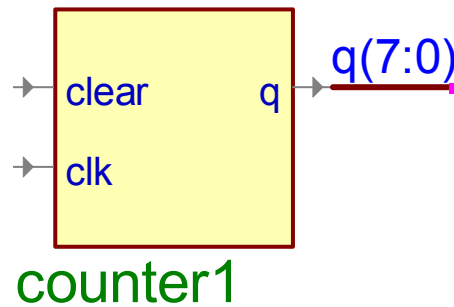
ARCHITECTURE d_ff OF DFF3 IS
BEGIN

  PROCESS (clk)
  BEGIN
    IF clk'EVENT AND clk = '1' THEN
      IF clr = '1' THEN
        q <= '0';
      ELSIF pre = '1' THEN
        q <= '1';
      ELSE
        q <= d;
      END IF;
    END IF;
  END PROCESS;
END d_ff;

```


דוגמאות לתכנון מונים

מונה מעלה מ-0 עד 255 עם איפוס סינכרוני



```

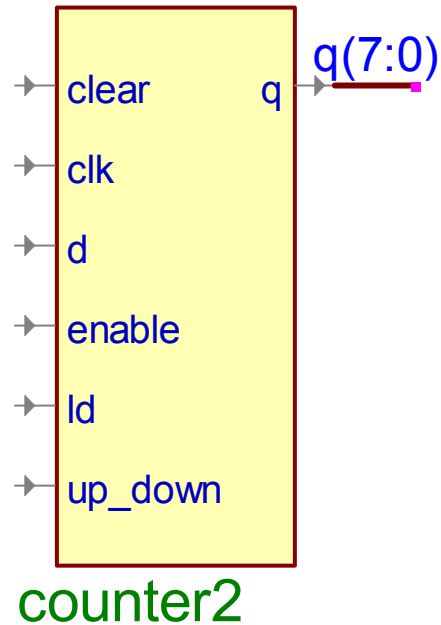
ENTITY counter1 IS
    PORT
    (
        clk      : IN BIT;
        clear    : IN  BIT;
        q        : OUT  INTEGER RANGE 0 TO 255
    );
END counter1;

ARCHITECTURE count OF counter1 IS
BEGIN

    -- A synchronous clear counter
    PROCESS (clk)
        VARIABLE cnt      : INTEGER RANGE 0 TO 255;
    BEGIN
        IF (clk'EVENT AND clk = '1') THEN
            IF clear = '0' THEN
                cnt := 0;
            ELSE
                cnt := cnt + 1;
            END IF;
        END IF;

        q <= cnt;
    END PROCESS;
END count;
  
```

מונה מעלה/מטה מ- 0 עד 255 עם איפוס, טעינה ואפשרות סינכרוני



```

ENTITY counter2 IS
    PORT
    (
        d      : IN    INTEGER RANGE 0 TO 255;
        clk    : IN    BIT;
        clear  : IN    BIT;
        ld     : IN    BIT;
        enable : IN    BIT;
        up_down : IN    BIT;
        q      : OUT   INTEGER RANGE 0 TO 255
    );
END counter2;

ARCHITECTURE count OF counter2 IS
BEGIN
    -- A synchronous clear enable up/down counter
    PROCESS (clk)
        VARIABLE cnt      : INTEGER RANGE 0 TO 255;
        VARIABLE direction : INTEGER;
    BEGIN
        IF (up_down = '1') THEN
            direction := 1;
        ELSE
            direction := -1;
        END IF;

        IF (clk'EVENT AND clk = '1') THEN
            IF clear = '0' THEN

```

```

        cnt := 0;
    ELSE
        IF ld = '0' THEN
            cnt := d;
        ELSE
            IF enable = '1' THEN
                cnt := cnt + direction;
            END IF;
        END IF;
    END IF;
END IF;

q    <= cnt;

END PROCESS;

END count;

```

מונה מעלה מודולו 200

```

ENTITY counter_200 IS

    PORT
    (
        clk      : IN BIT;
        q        : OUT  INTEGER RANGE 0 TO 255
    );

END counter_200;

ARCHITECTURE count OF counter_200 IS
BEGIN

    -- A modulus 200 up counter
    PROCESS (clk)
        VARIABLE cnt      : INTEGER RANGE 0 TO 255;
        CONSTANT  modulus : INTEGER := 200;
    BEGIN
        IF (clk'EVENT AND clk = '1') THEN
            IF cnt = modulus THEN
                cnt := 0;
            ELSE
                cnt := cnt + 1;
            END IF;
        END IF;

        q    <= cnt;

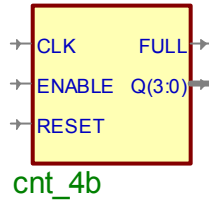
    END PROCESS;

END count;

```

מונה מעלה BCD עם reset אסינכרוני ואפשר סינכרוני

מוצא FULL=1 כאשר ערך המונה 9



```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;  -- integer לדומה ל- integer

entity CNT_4B is
  port (
    CLK: in STD_LOGIC;
    RESET: in STD_LOGIC;
    ENABLE: in STD_LOGIC;
    FULL: out STD_LOGIC;
    Q: out STD_LOGIC_VECTOR (3 downto 0)
  );
end CNT_4B;

architecture CNT_4B of CNT_4B is
  signal Qint: STD_LOGIC_VECTOR(3 downto 0);

begin
  process (CLK, RESET)
  begin
    if RESET = '1' then
      Qint <= (others => '0');
    elsif CLK='1' and CLK'event then
      if ENABLE = '1' then
        if Qint = 9 then
          Qint <= (others => '0');
        else
          Qint <= Qint + 1;
        end if;
      end if;
    end if;
  end process;

  Q <= Qint;

  FULL <= '1' when (Qint = 9) else '0';
end CNT_4B;
  
```

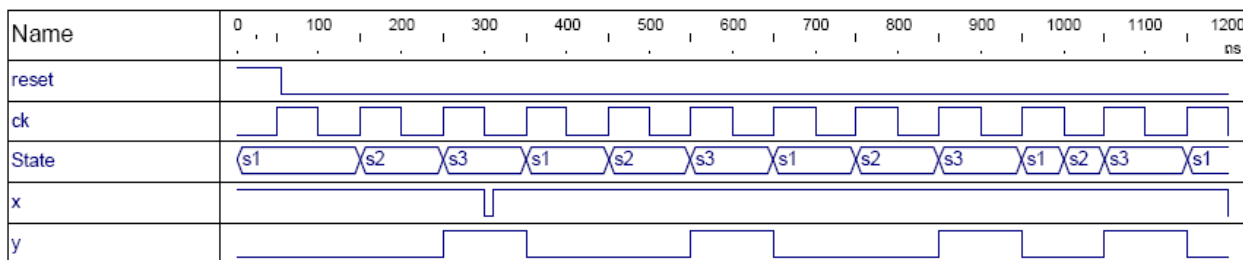
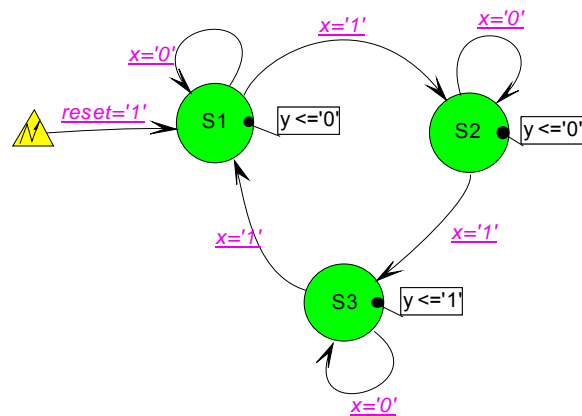
מכונת מצבים

מכונת מצבים הינה אוטומט המשנה את מצבו בכל עליה או ירידת שעון בהתאם למצב הנוכחי, תנאים וערכי הכניסות. מוצא המכונה תלוי במצב המכונה ובכניסות בהתאם לסוג המכונה.

Mealy - מוצא המכונה תלוי במצב המכונה ובכניסה.

Moore - מוצא המכונה תלוי במצב המכונה בלבד.

דוגמא למכונת Moore



בדוגמא המוצא y תלוי במצב המכונה בלבד, בתרשים הזמנים רואים ש-y במצב S3 אינו מושפע מ-x

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

entity moor is
  port (
    ck: in STD_LOGIC;
    reset: in STD_LOGIC;
    x: in STD_LOGIC;
    y: out STD_LOGIC);
end moor;
```

```

architecture moor_arch of moor is

    -- SYMBOLIC ENCODED state machine: State
    type State_type is (
        S1, S2, S3
    );

    signal State: State_type;

begin

    -----
    -- Machine: State
    -----

    State_machine: process (ck, reset)
    begin
        if reset='1' then
            State <= S1;
        elsif ck'event and ck = '1' then

            case State is
                when S1 =>
                    if x='1' then
                        State <= S2;
                    end if;
                when S2 =>
                    if x='1' then
                        State <= S3;
                    end if;
                when S3 =>
                    if x='1' then
                        State <= S1;
                    end if;
                when others =>
                    null;
            end case;
        end if;
    end process;

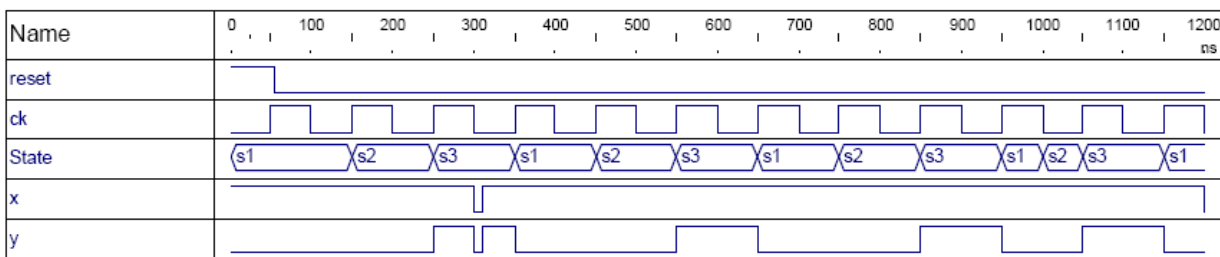
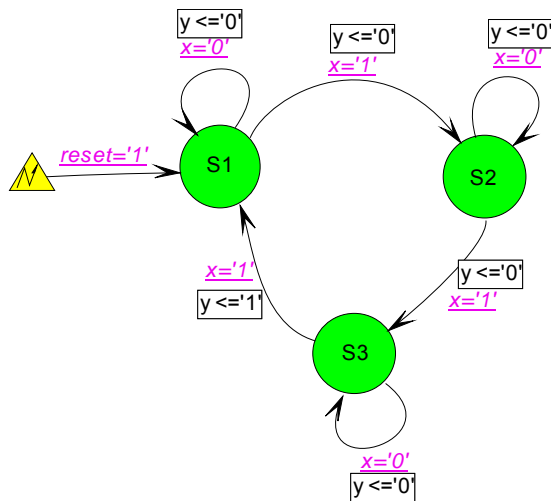
    -----

    process (State)
    begin
        case State is
            when S1 | S2 => y<='0';
            when S3=> y<='1';
        end case;
    end process;

end moor_arch;

```

דוגמא למכונת Mealy



בדוגמא, המוצא y תלוי במצב המכונה ובכניסה x .

אם לדוגמא המכונה נמצאת במצב S3 המוצא y משתנה בתלות ב- x ללא תלות בשעון.

(במקרה שהמכונה היא mealy סינכרוני, המוצא ישתנה בתלות ב- x לאחר שינוי השעון)

```

library IEEE;
use IEEE.std_logic_1164.all;

entity mealy is
  port (
    ck: in STD_LOGIC;
    reset: in STD_LOGIC;
    x: in STD_LOGIC;
    y: out STD_LOGIC);
end mealy;

architecture mealy_arch of mealy is

  -- SYMBOLIC ENCODED state machine: State
  type State_type is (
    S1, S2, S3
  );

```

```

signal State: State_type;

begin

-----
-- Machine: State
-----

State_machine: process (ck, reset)
begin
    if reset='1' then
        State <= S1;
    elsif ck'event and ck = '1' then

        case State is
            when S1 =>
                if x='1' then
                    State <= S2;
                end if;
            when S2 =>
                if x='1' then
                    State <= S3;
                end if;
            when S3 =>
                if x='1' then
                    State <= S1;
                end if;
            when others =>
                null;
            end case;
        end if;
    end process;

-----

process (State,x)
begin
    case State is
        when S1 | S2 => y<='0';
        when S3=>
            if x='1' then
                y<='1';
            else
                y<='0';
            end if;
        end case;
    end process;

end mealy_arch;

```



```

library IEEE;
use IEEE.std_logic_1164.all;

entity mealy_syn is
  port (
    ck: in STD_LOGIC;
    reset: in STD_LOGIC;
    x: in STD_LOGIC;
    y: out STD_LOGIC);
end mealy_syn;

architecture mealy_arch of mealy_syn is

  -- SYMBOLIC ENCODED state machine: State
  type State_type is (
    S1, S2, S3
  );

  signal State: State_type;

begin

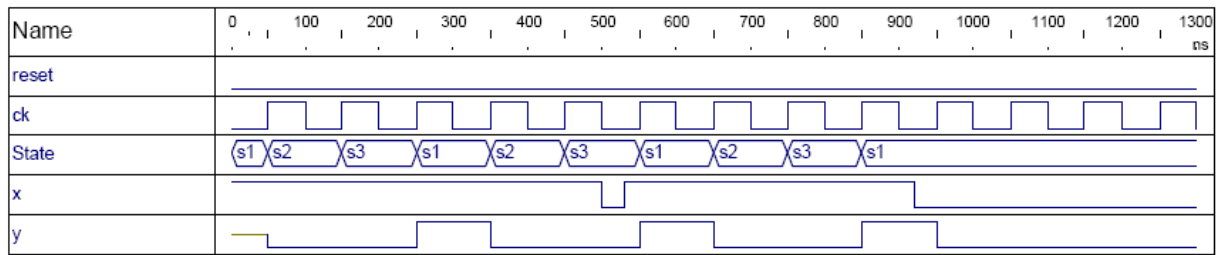
  -----
  -- Machine: State
  -----

  State_machine: process (ck, reset)
  begin
    if reset='1' then
      State <= S1;
    elsif ck'event and ck = '1' then

      case State is
        when S1 =>
          y<='0';
          if x='1' then
            State <= S2;
          end if;
        when S2 =>
          y<='0';
          if x='1' then
            State <= S3;
          end if;
        when S3 =>
          if x='1' then
            y<='1';
            State <= S1;
          else
            y<='0';
          end if;
        when others =>
          null;
        end case;
      end if;
    end process;

  end mealy_arch;

```



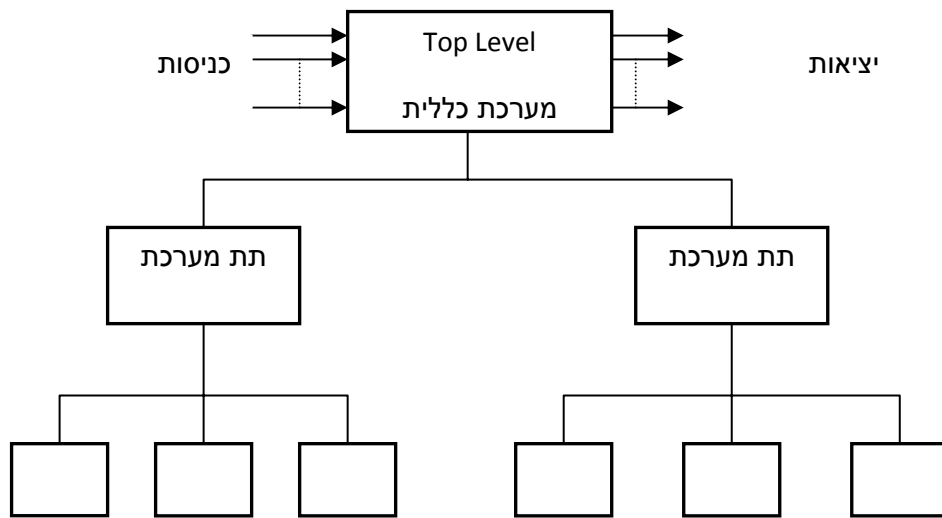
ב- Mealy סינכרוני רואים שהמוצא y משתנה במעבר ממצב S3 ל- S1 (עליית שעון) כאשר הכניסה $x=1$.

אם ב- Moore המוצא תלוי אך ורק במצב, רואים ב-Mealy עבור אותו מצב S1 נוכל לקבל 1 או 0 התלוי בכניסה x .

תכנון היררכי

מאפייני מערכת היררכית:

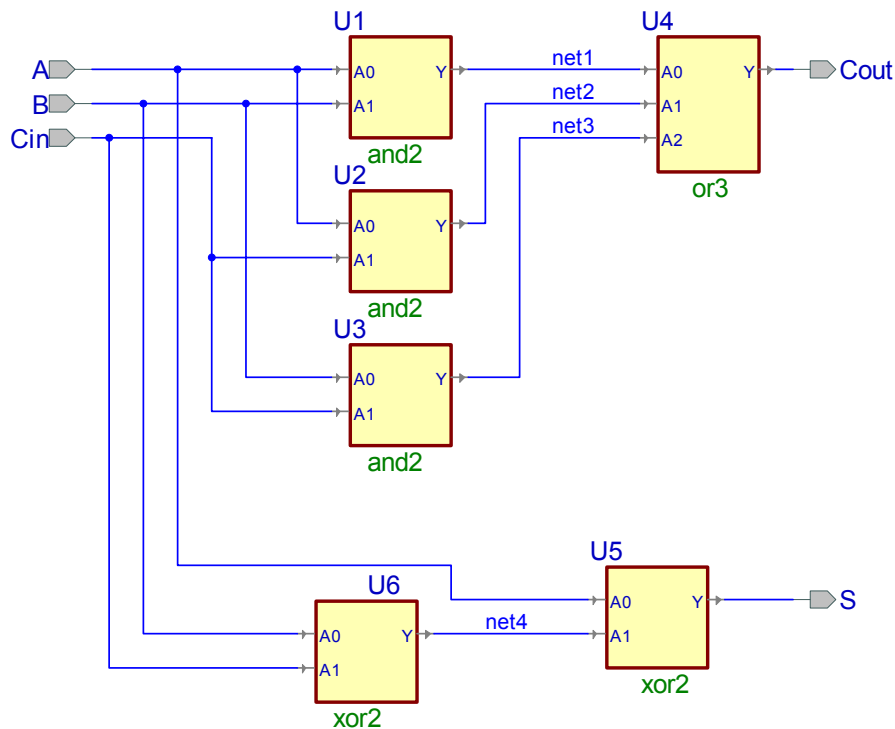
- לכל מערכת כניסות ויציאות – קלט, פלט.
- כל המערכות מורכבות מתת מערכות בצורה הירארכית שהן בעצמן תת מערכות למערכת שמעליהן.



שלבים בתכנון היררכי

1. הגדרת הכניסות ופרוט הדרישות של המערכת המלאה
2. פרוק המערכת לתת מערכות
3. הגדרת הכניסות ופרוט הדרישות של כל תת מערכת
4. כתיבת תוכנית לכל תת מערכת והרצת תוכנית בדיקה וסימולציה
5. חיבור כל תת היחידות
6. הרצת תוכנית בדיקה וסימולציה למערכת המלאה

דוגמא – תכנון מסכם מלא – FA



```
-- Design unit header --
library IEEE;
use IEEE.std_logic_1164.all;

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

architecture f_a of f_a is

  ---- Component declarations ----
  component and2
    port (
      A0 : in STD_LOGIC;
      A1 : in STD_LOGIC;
      Y : out STD_LOGIC
    );
  end component;
  component or3
    port (
      A0 : in STD_LOGIC;
      A1 : in STD_LOGIC;
      A2 : in STD_LOGIC;
      Y : out STD_LOGIC
    );
  end component;
  component xor2
    port (
```

```

        A0 : in STD_LOGIC;
        A1 : in STD_LOGIC;
        Y : out STD_LOGIC
    );
end component;

---- Signal declarations used on the diagram ----
signal net1 : STD_LOGIC;
signal net2 : STD_LOGIC;
signal net3 : STD_LOGIC;
signal net4 : STD_LOGIC;

begin

---- Component instantiations ----

U1 : and2
    port map(
        A0 => A,
        A1 => B,
        Y => net1
    );

U2 : and2
    port map(
        A0 => A,
        A1 => Cin,
        Y => net2
    );

U3 : and2
    port map(
        A0 => B,
        A1 => Cin,
        Y => net3
    );

U4 : or3
    port map(
        A0 => net1,
        A1 => net2,
        A2 => net3,
        Y => Cout
    );

U5 : xor2
    port map(
        A0 => A,
        A1 => net4,
        Y => S
    );

U6 : xor2
    port map(
        A0 => B,
        A1 => Cin,
        Y => net4
    );

end f_a;

```