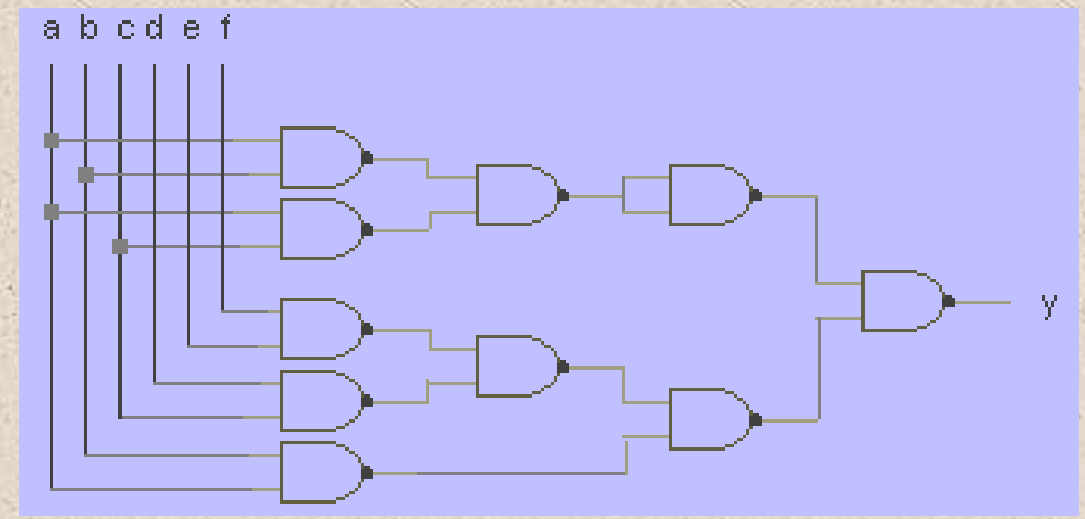
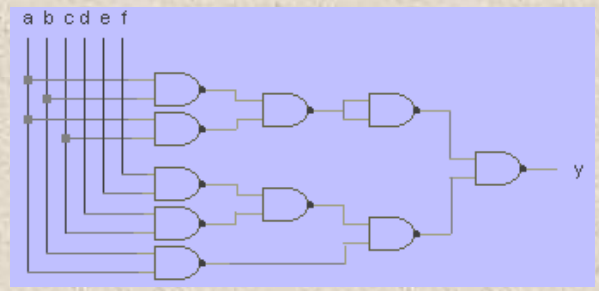
Проектування комбінаційних схем



Комбінаційна схема



Може бути описана параллельними операторами

- булевські рівняння
- **when-else**
- **with-select-when**
- вставка компонента

Може бути описана послідовними операторами

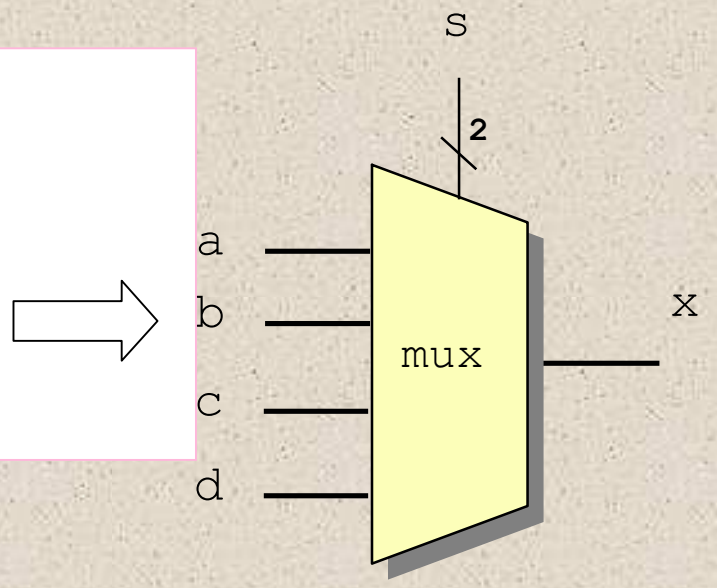
- **if-then-else**
- **case-when**

Комбінаційна схема за булевським рівнянням

булевські рівняння можуть бути використані як в паралельних, так і послідовних операторах присвоювання сигнала.

Мультиплексор 4 до 1 описується так:

```
x <=(a AND NOT(s(1)) AND NOT(s(0)))
     OR(b AND NOT(s(1)) AND s(0))
    OR(c AND s(1) AND NOT(s(0)))
    OR(d AND s(1) AND s(0)) ;
```



Селективне присвоювання сигнала: with-select-when

Присвоювання, що основане на виборці сигнала за допомогою опису **WHEN** повинне бути взаємно виключним.

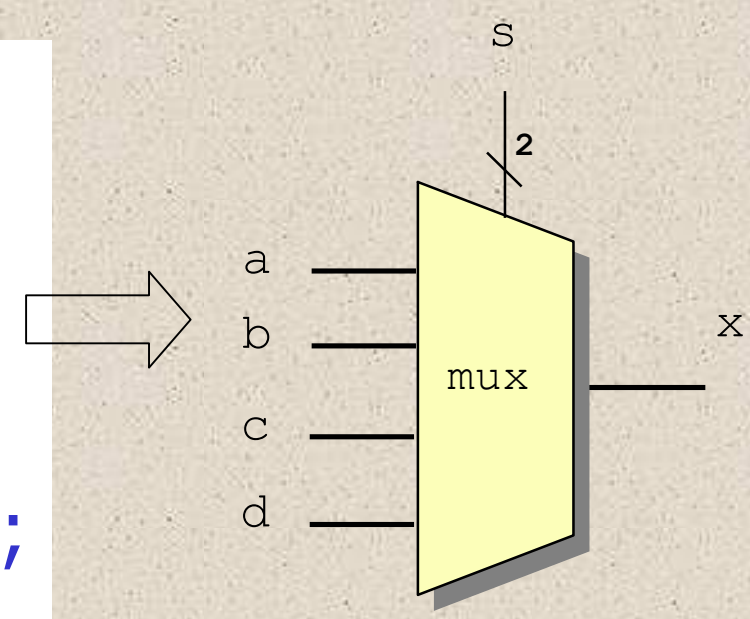
Слід використовувати **WHEN OTHERS** якщо не перелічені всі комбінації аргументів.

```
WITH \селектор\ SELECT
\сигнал\ <= \значення_1\ WHEN \значення_1_селектора\,
            \значення_2\ WHEN \значення_2_селектора\,
                    ...
            \значення_n\ WHEN \значення_n_селектора\,
      \значення_інше\ WHEN OTHERS;
```

Селективне присвоювання сигнала: with-select-when

Той самий мультиплексор 4 -1:

```
with s select
        x <= a when "00" ,
             b when "01" ,
             c when "10" ,
             d when others ;
```



Селективне присвоювання сигнала: with-select-when

Можна використати діапазони значень

```
signal \вибір\ :integer range 0 to 10;
. . .
   with \вибір\ select
                  x <= a when 0 to 3,
                       b when 4 | 6 | 8 ,
                       c when 10 ,
                       d when others ;
```

Умовне присвоювання сигналу: when-else

Сигналу присвоюється значення в залежності від умови.

Умовою може бути вираз.

Пріоритет тому виразу, який зустрічається першим.

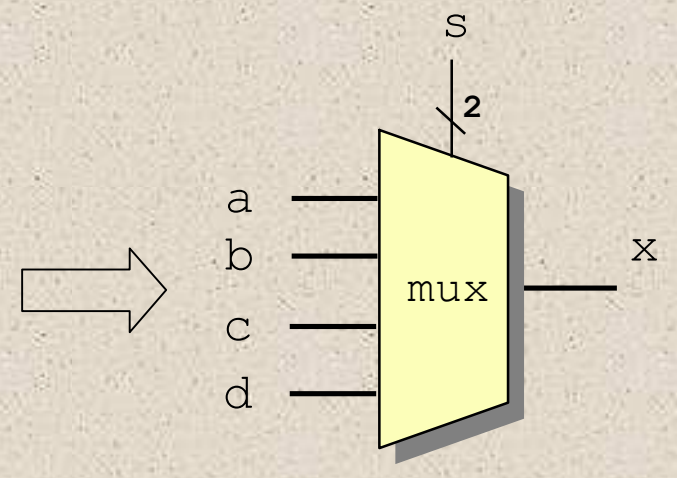
Потрібно закінчувати **ELSE** щоб не було защіпки.

```
\ім'я_сигнала\<= \значення_1\ WHEN \умова1\ ELSE
                \значення_2\ WHEN \умова2\ ELSE
                ...
                \значення_N\ WHEN \умоваN\ ELSE
                \значення_X\;
```

Комбінаційна схема за умовним присвоюванням

Той же мультиплексор 4-1:

```
x <= a when (s = "00") else
     b when (s = "01") else
     c when (s = "10") else
     d ;
```



Комбінаційна схема за умовним присвоюванням

Умова **when** не обов'язково повинна бути взаємно виключною (як в **with-select-when**)

Схема пріоритетного шифратора описується так:

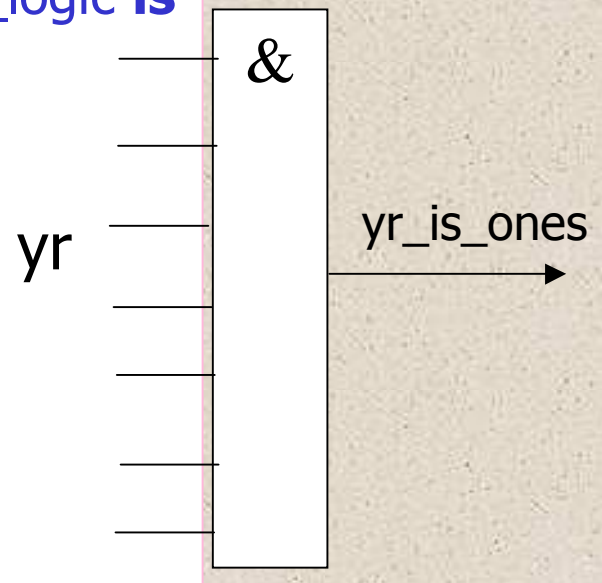
```
j <=  "100" when (a(3) = '1') else
          "011" when (a(2) = '1') else
          "010" when (a(1) = '1') else
          "001" when (a(0) = '1') else
          "000" ;
```

Комбінаційна схема за викликом підпрограми

Підпрограма не повинна вміщувати операторів **wait**.

Приклад функції **логічної редукції** – Логічне І від N входів:

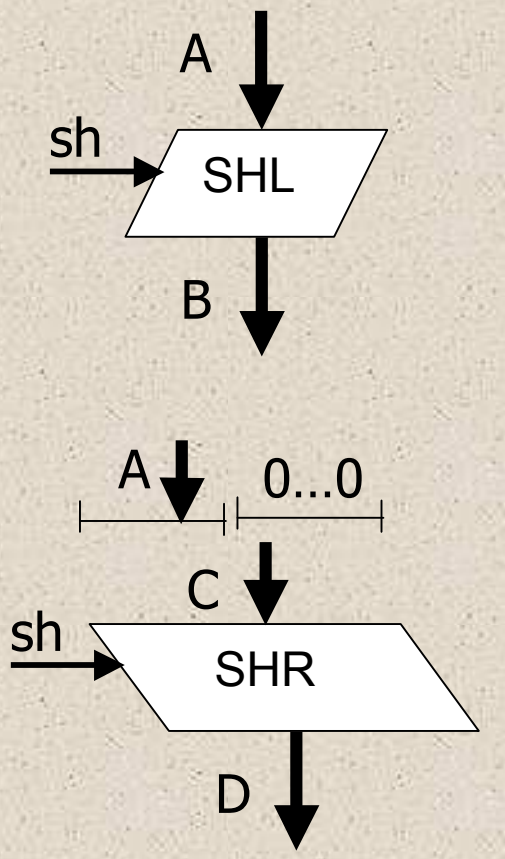
```
Function ANDN(x:std_logic_vector) return std_logic is
        Variable yi:std_logic;
Begin
        yi:='0';
        For I in x'range loop
                yi:=yi and x(I);
        End loop;
Return yi;
End function;
. . .
yr_is_ones<=ANDN(yr);
```



Комбінаційна схема за викликом підпрограми

Схема зсуву

```
Library IEEE;
Use IEEE.STD_LOGIC_1164.all;
. . .
Signal A, B, C: signed(15 downto 0);
Signal C,D: signed(31 downto 0);
Signal sh: signed(15 downto 0);
. . .
B<=SHL(A,sh); -- зсув вліво на sh розрядів
        -- розряди, що висуваються - губляться
C<=A& X"0000" -- розширили розрядну сітку
D<=SHR(C,sh); -- зсув вправо на sh розрядів
                -- без загублених розрядів
```



Комбінаційна схема за послідовним оператором

Послідовні оператори формують тіло процесу.

Порядок послідовних операторів впливає на результат синтезу.

Послідовні оператори : if-then-else

Використовуються для вибору операторів, що будуть виконуватися в даному запуску процесу.

Вибір оснований на результаті булевського виразу від ряду умов.

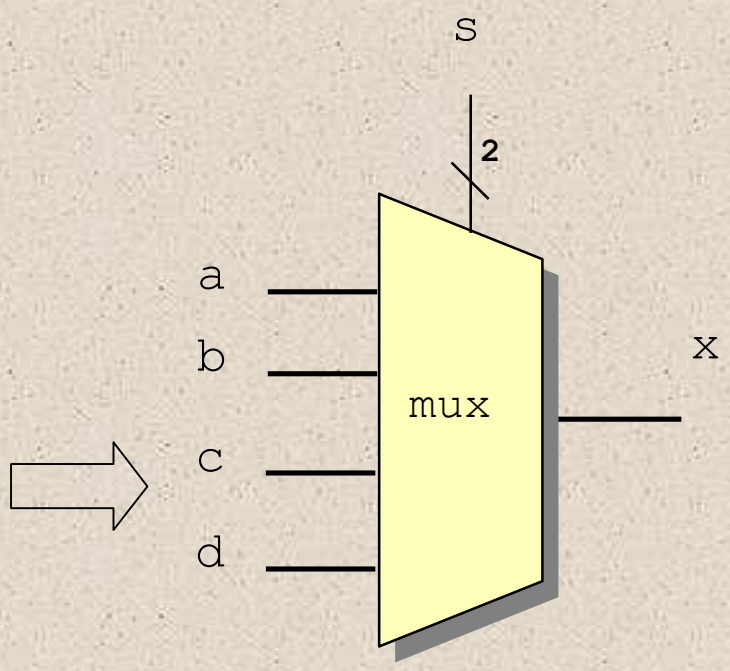
Відсутність **ELSE** породжує **асинхронний триггер.**

```
IF \умова(и)\ THEN
        {оператори};
ELSIF \умова_2\ THEN    -- не обов"язково
        {інші оператори};
ELSE
        {зовсім інші оператори};
END IF ;
```

Послідовні оператори : if-then-else

Мультиплексор 4 -1:

```
mux4_1: process (a, b, c, d, s)
        begin
                if  s = "00" then
                        x <= a ;
                elsif  s = "01" then
                        x <= b ;
                elsif  s = "10" then
                        x <= c ;
                else x
                        <= d ;
                end if;
        end process mux4_1 ;
```



Послідовні оператори : Case-When

```
CASE \селектор\ IS
   WHEN \значення_1_селектора\ =>
        {оператори};        -- 1-й набір операторів
   WHEN \значення_2_селектора\ =>
        {оператори};        -- 2-й набір операторів
...
   WHEN \значення_N_селектора\ =>
        {оператори};        -- N-й набір операторів
    WHEN OTHERS  =>
        {оператори};-- довільні оператори
END CASE ;
```

Послідовні оператори : loop

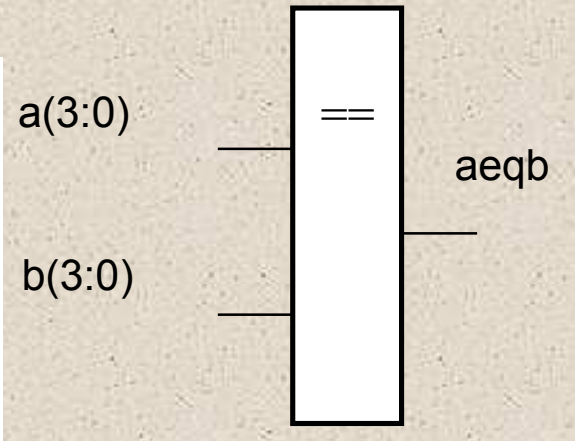
Пріоритетний шифратор

```
SIGNAL s: std_logic_vector(15 downto 0);
SIGNAL prty: std_logic_vector(3 downto 0);
        . . .
priority: PROCESS (s)
        variable n:unsigned(3 downto 0);
BEGIN
        n:="0000";
        for I in s'range loop
            if s(I)='1' then -- знайшли першу 1 і зупинились
                    exit;
            else
                    n:=n+1;
            end if;
        end loop;
        prty<=std_logic_vector(n);
END PROCESS;
```

Приклад: розробка компаратора

Об’ява елемента проекта:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY compare IS PORT (
    a, b: IN std_logic_vector(3 DOWNTO 0);
    aeqb: OUT std_logic);
END compare;
```



- Описати архітектуру в якій сигналу **aeqb** присвоюється 1 якщо **a** дорівнює **b**
- Існує кілька рішень

Три різних рішення

Рішення з **параллельними** операторами з використанням **умовного присвоювання**:

```
ARCHITECTURE arch1_compare OF compare IS
BEGIN
    aeqb <= '1' WHEN a = b ELSE '0';
END arch1_compare;
```

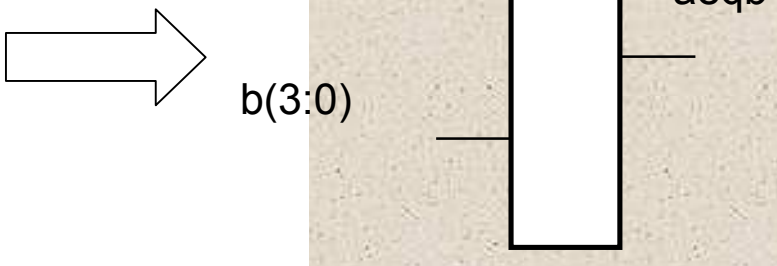
Рішення з **параллельними** операторами з використанням **булевського рівняння**:

```
ARCHITECTURE arch2_compare OF compare IS
BEGIN
    aeqb <= NOT(
        (a(0) XOR b(0)) OR
        (a(1) XOR b(1)) OR
        (a(2) XOR b(2)) OR
        (a(3) XOR b(3)));
END arch2_compare;
```

Три різних рішення

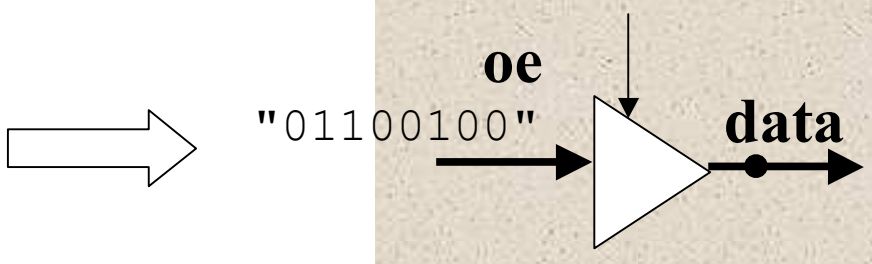
Рішення з використанням **процеса** з **послідовними операторами**:

```
ARCHITECTURE arch_compare OF compare IS
BEGIN
comp: PROCESS (a, b)
    BEGIN
        IF a = b THEN
            aeqb <= '1';
        ELSE
            aeqb <= '0';
        END IF;
    END PROCESS comp;
END arch_compare;
```



Використання логіки з трьома станами

```
ENTITY test_three IS
        PORT(oe  :  IN std_logic;
                 data : OUT std_logic_vector(0 to 7));
END test_three;
ARCHITECTURE archtest_three OF test_three IS
BEGIN
   PROCESS (oe)
   BEGIN
        IF (oe = '1')
          THEN data <= "01100100";
          ELSE data <= "ZZZZZZZZ";
        END IF;
   END PROCESS;
END archtest_three;
```

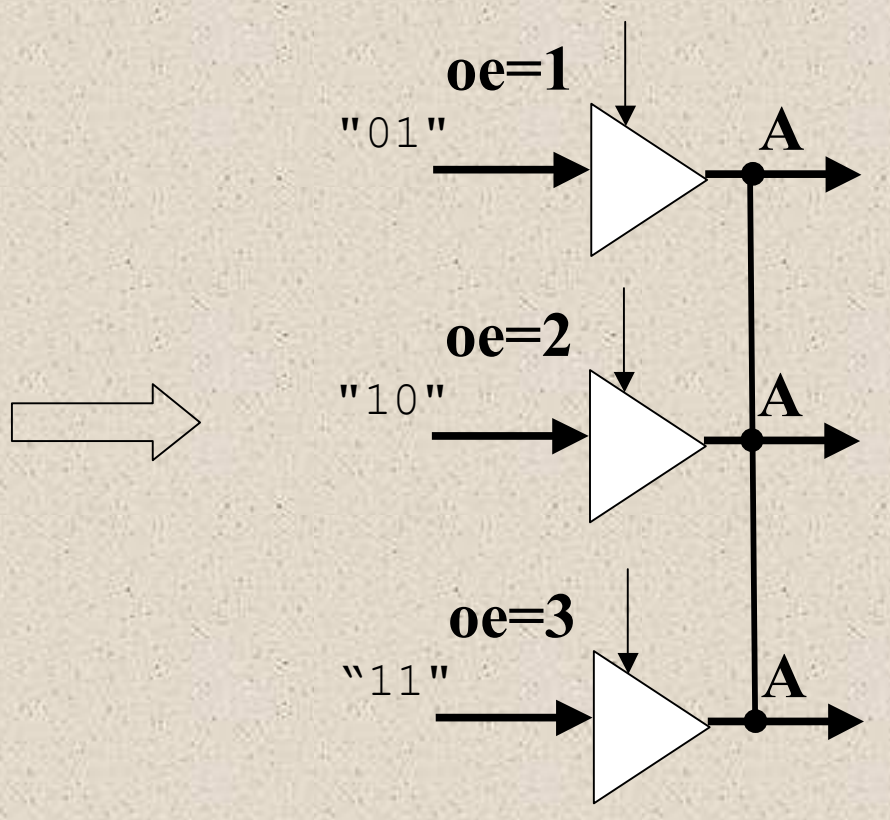


Використання логіки з трьома станами

За допомогою 3-го стану можна 1 сигнал генерувати кількома процесами або паралельними операторами

```
P1:PROCESS (oe) BEGIN
      IF oe = 1  THEN
            A <= "01";
       ELSE
            A <= "ZZ";
      END IF;
   END PROCESS;
P2:PROCESS (oe) BEGIN
      IF oe = 2  THEN
            A <= "10";
       ELSE
            A <= "ZZ";
      END IF;
   END PROCESS;

A<="11" when oe=3 else "ZZ";
```



Довільний стан в Std_Logic

Для поведінкового моделювання часто використовують стан '-' “довільний" .

Цей стан не має еквівалента при синтезі, тому за операторами

```
IF (A="11-0") THEN
        X <= C;
    ELSE
        X <= D;
END IF;
```

завжди буде виконуватись X <= D;

так як A = "11-0" завжди буде false.

Тому **не рекомендуєтья** використовувати стан '-' “довільно" для синтезу.

Довільний стан в Std_Logic

Для попереднього прикладу слід використати функцію **std_match**(a,"string") з пакета IEEE.NUMERIC_STD

Приклад:

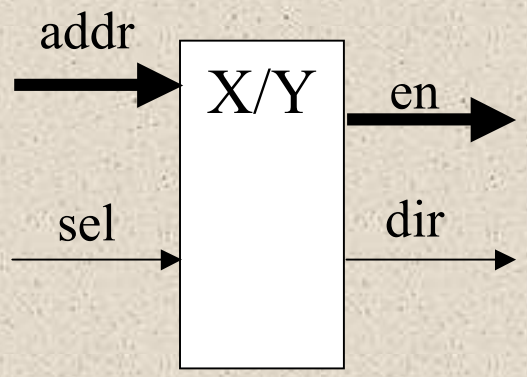
```
. . .
signal A : std_logic_vector (1 to 4) ;
...
IF std_match(A,"11-0") THEN
        X <= C;
ELSE
        X <= D;
END IF ;
```

Використання With-Select для декодування кількох сигналів

```
ARCHITECTURE synt OF DC3 IS
   SIGNAL cntrl : std_logic_vector(2 DOWNTO 0);
   SIGNAL outp : std_logic_vector(0 TO 4);
BEGIN
   cntrl <= addr & sel;

   WITH cntrl SELECT
      outp <= "00100" WHEN "000",
              "10101" WHEN "001",
              "11101" WHEN "010",
              "10101" WHEN "011",
              "10000" WHEN "100",
              "10101" WHEN "101",
              "10111" WHEN "110",
              "10101" WHEN "111",
              "-----" WHEN OTHERS;
   en <= outp(0 TO 3);
   dir <= outp(4);
END synt;
```

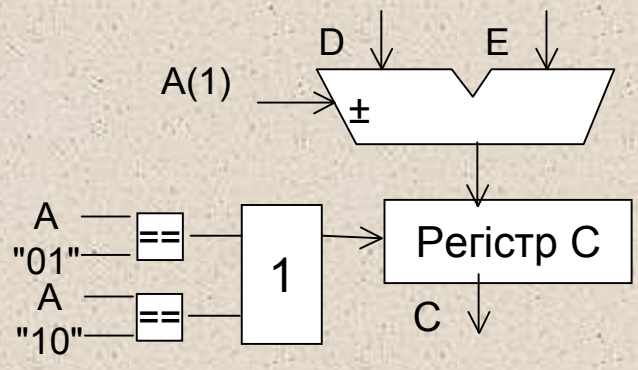
Тут вираз, вирізка або агрегат не припускається



Непередбачувані асинхронні тригери неприпустимі

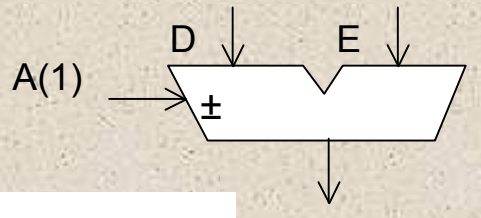
Непередбачуваний тригер,
коли перелічені не всі комбінації вибору

```
Process(A,D,E) begin
   case A is
        when "01"=> C<=D+E;
        when "10"=> C<=D-E;
    end case;
end process;
```



Проектування арифметичних пристроїв

Пакети std_logic_arith, std_logic_signed і std_logic_unsigned



Визначені типи unsigned і signed як вектори елементів типу std_logic

unsigned - позитивні двійкові цілі числа

signed - двійкові цілі числа зі знаком в доповнюючому коді.

Функції '+', '-', '*', abs,

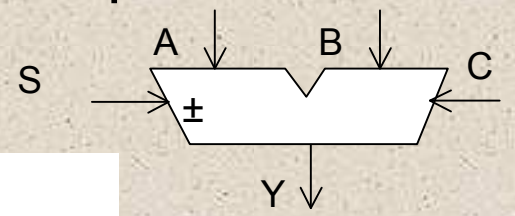
Всі функції порівняння для операндів типу unsigned, signed, small_int і integer

Вектори операндів у виразах можуть мати різні діапазони

Проектування арифметичних пристроїв

Приклад.

Побудувати арифметичний пристрій для операцій A+B, A-B, A+B+C



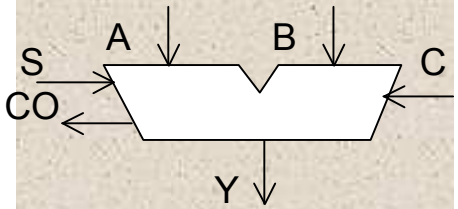

```
Library IEEE;
Use IEEE.STD_LOGIC_1164.all, IEEE.std_logic_arith.all;
Entity SM is port(A,B:in signed(7 downto 0);
                  C:in std_logic;
                  S: in std_logic_vector(1 downto 0);
                  Y: out signed(7 downto 0));
End SM;
Architecture beh of SM is
Begin
    Y<=A+B  When S= "00"   else
          A+B+ C     When S = "01"   else
          A - B  ;
End beh;
```

Проектування арифметичних пристроїв

Приклад.

Побудувати арифметичний пристрій для операцій A+B, A+B+C з видачею переносу зі старшого розряду

```
Library IEEE;
Use IEEE.STD_LOGIC_1164.all, IEEE.std_logic_arith.all;
Entity SM is port(A,B:in std_logic_vector(7 downto 0);
                  C,S:in std_logic;
                  CO: out std_logic;
                  Y: out std_logic_vector(7 downto 0));
End SM;
Architecture beh of SM is
Signal yi: signed (8 downto 0);
Begin
    yi<=signed(SXT(A,9))+signed(B)  When S= '0'  else
          signed(SXT(A,9))+signed(B) + C;
    Y<=CONV_STD_LOGIC_VECTOR(yi,8);
    CO<=yi(8);
End beh;
```



Розширення розрядної сітки до 9

Приклад. Побудувати схему ділення A/B, A<B

```
Signal A,B,Y: unsigned(7 downto 0);
. . .
    process(A,B)
        variable r,t:unsigned(8 downto 0);
    begin
        r:=A(7)&A ;
        for i in 0 to 7 loop
                t:=r - SHR(B, CONV_UNSIGNED(i,3));
                if t(8) = '0' then
                        Y(7-i) <= '1';
                        r:=t;
                else
                        Y(7-i) <= '0';
                end if;
        end loop;
    end process;
```