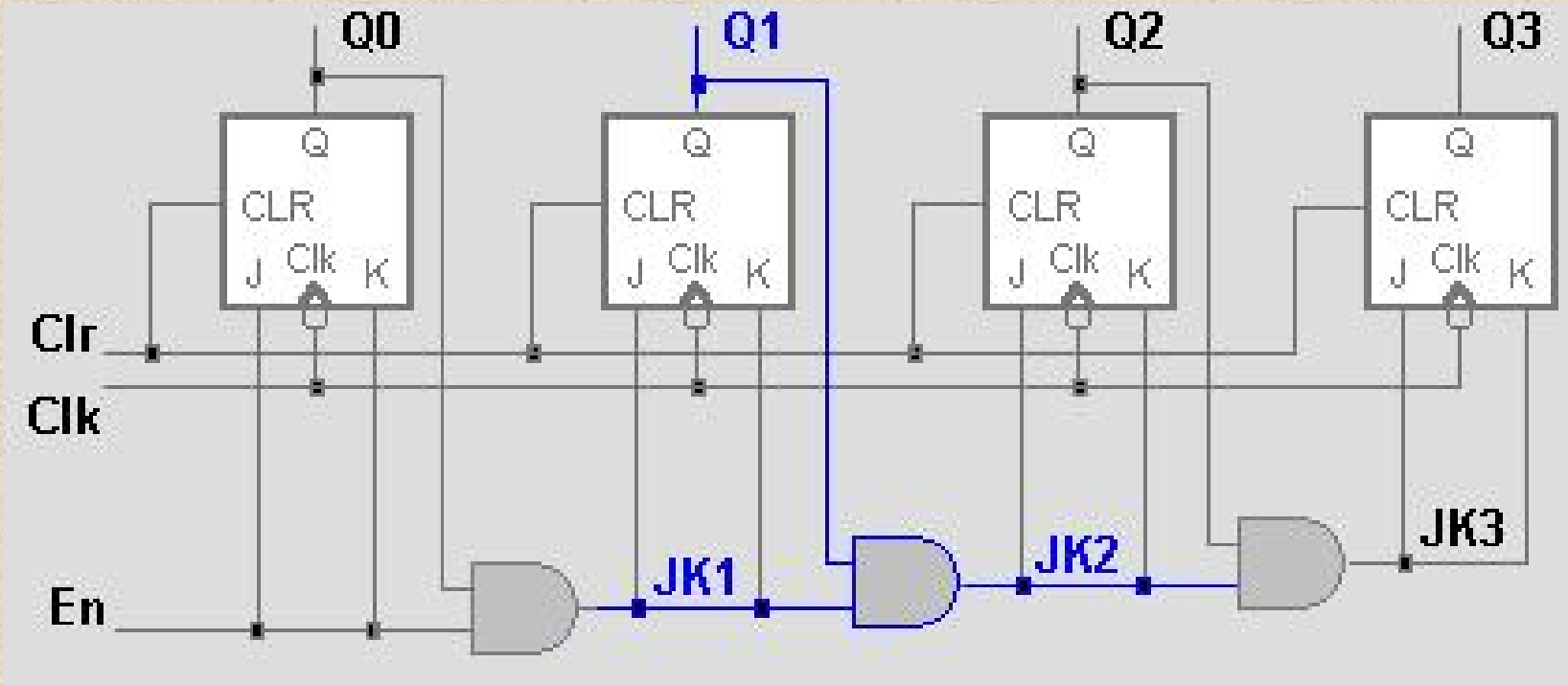
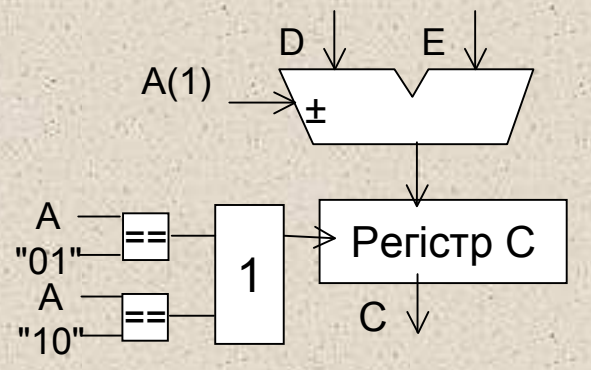
# Проектування схем з пам’яттю



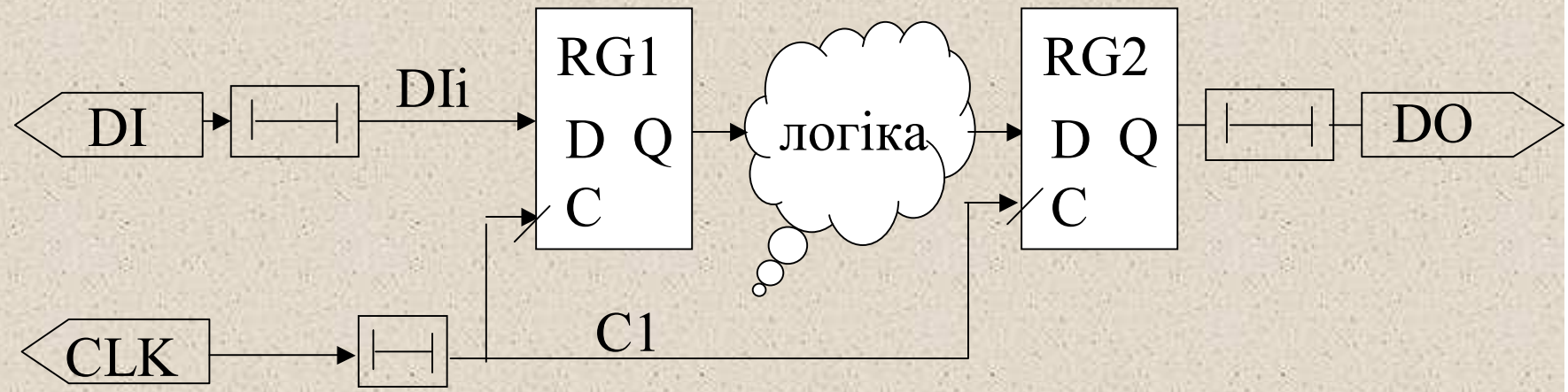
# Непередбачувані асинхронні тригери недопустимі


**Непередбачуваний тригер,**
коли перелічені не всі комбінації вибору

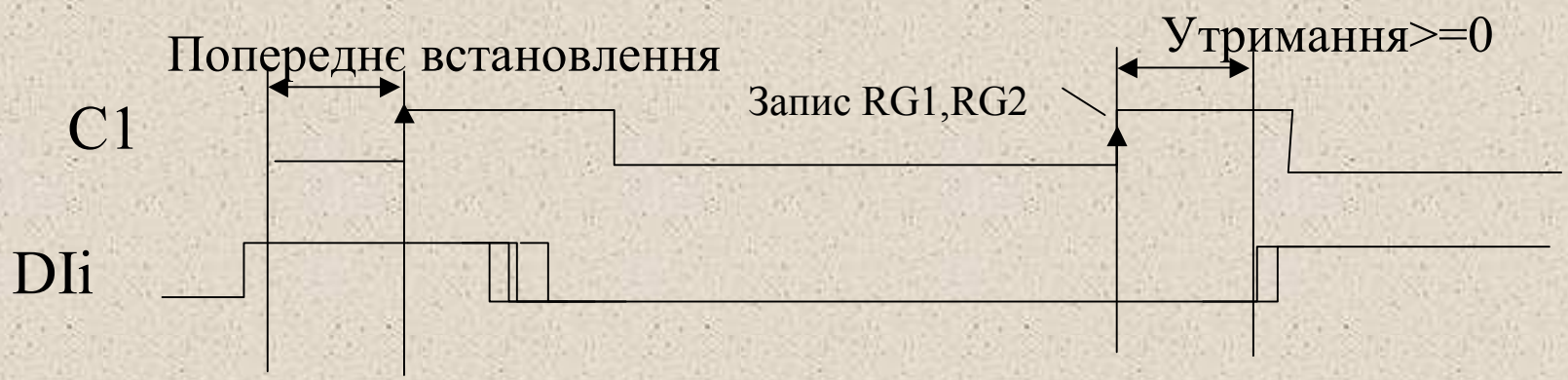
```
Process(A,D,E) begin
   case A is
        when "01"=> C<=D+E;
        when "10"=> C<=D-E;
    end case;
end process;
```



# Принцип однотактної синхронізації







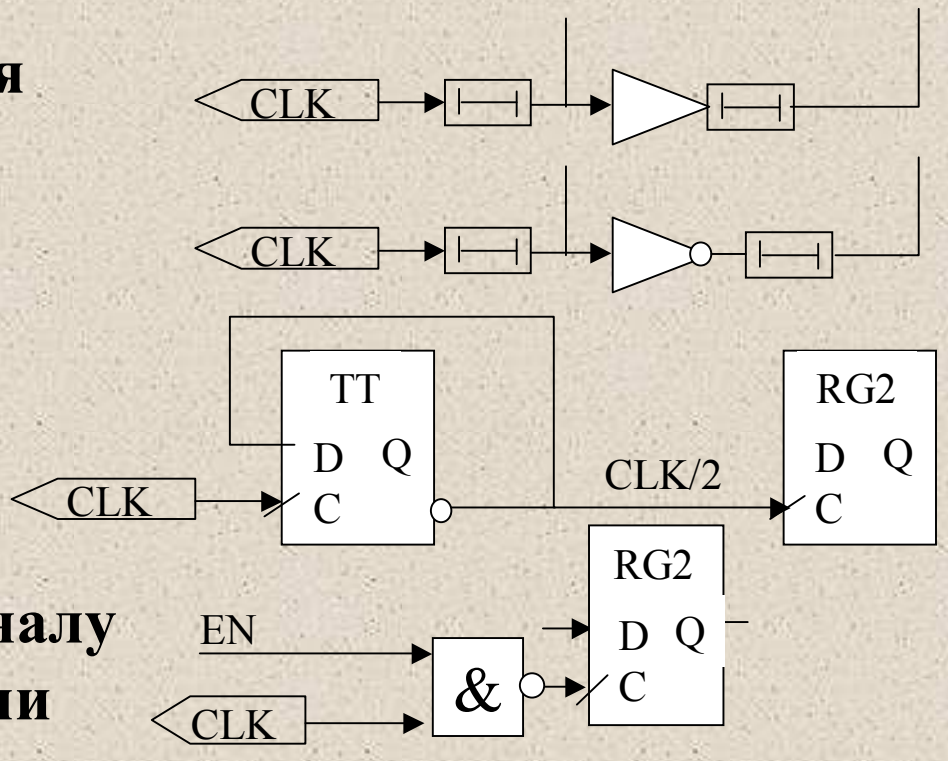
# Перекручування однотактної синхронізації.

**Додаткова буферизація**

**Інвертування**

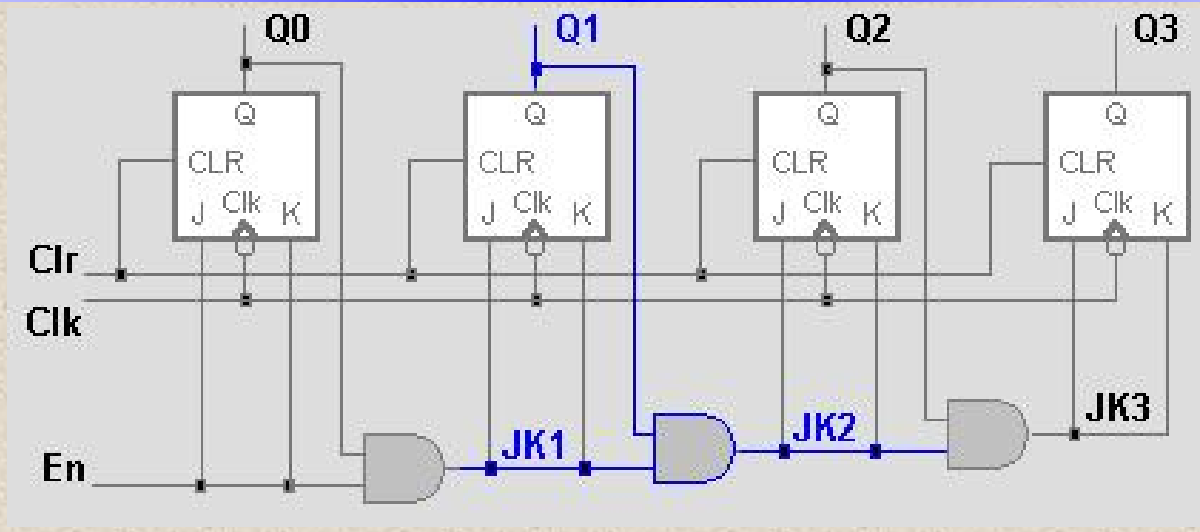
**Внутрішня генерація синхросигналу**

**Змішування синхросигналу з логічними сигналами**




- Треба не допускати викручування синхронізації
- При необхідності використовуйте глобальні буфери і мережі синхронізації
- Бажано зводити кількість синхросерій до 1.

# Синтез схем з пам’яттю




**Два методи створення схем з пам’яттю**

- **Структурний**
  - **вставка компонентів з регістрами**
- **Поведінковий**
  - **Використання процесів з синхросигналом в списку чутливості**

# Регістри в поведінковому VHDL

**Приклад: D-тригер**

```
ENTITY DTRIG IS PORT (d, clk:IN std_logic;
                      q:OUT std_logic);
END DTRIG;
ARCHITECTURE synt OF DTRIG IS
BEGIN
    flipflop: PROCESS (clk)
    BEGIN
        IF rising_edge(clk)
          THEN q <= d;
        END IF;
    END PROCESS flipflop;
END synt;
```

# Регістри в поведінковому VHDL

Компілятор-синтезатор передбачає, що синтезується тригер з виходом **q** через те, що

- Синхросигнал (clk) в списку чутливості
- В процесі знаходиться конструкція **rising_edge**(clk), **falling_edge**(clk) або clk'**event AND** clk='1'

Функції **rising_edge**(clk) і **falling_edge**(clk) означають, що залежне від ціїє умови присвоювання сигналу виконується в момент фронту/спаду синхросигналу

Відсутністо слова "**else**" в операторі "**if-then**" означає, що якщо умова clk'**event** and clk = '1' не виконується ( немає фронту сигналу), то **q** буде зберігати своє значення до наступного присвоювання (що означає схему з памяттю)

# Функції фронту/спаду

Пакет std_logic_1164 визначає дві функції для детектування перепаду синхросигналу

- **rising_edge** (*signal*)
  - аналог (*signal*'**event and** *signal* ='1')
- **falling_edge** (*signal*)
  - аналог (*signal*'**event and** *signal* ='0')

Тригер або регістр:

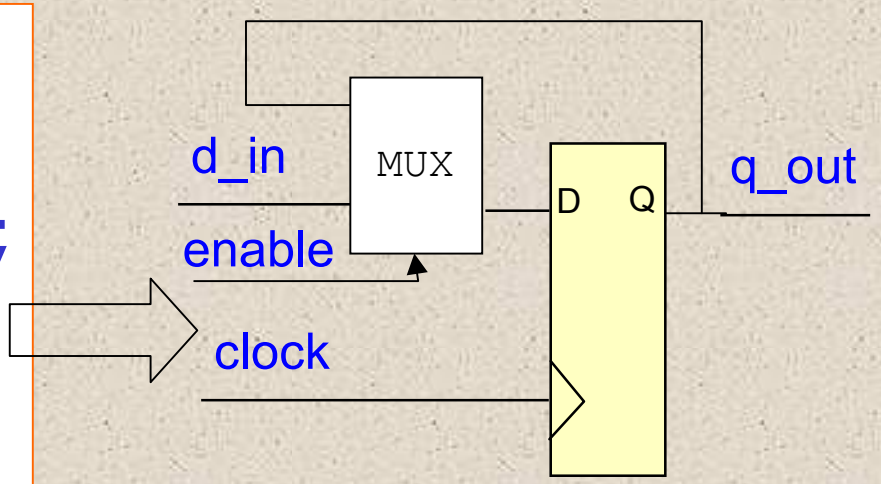
```
Process(clk) begin
        if rising_edge(clk) then
                q <= d;
        end if;
End process;
```

# Оператор WAIT

Це інший метод активізувати процес.

Оператор **WAIT** це послідовний оператор, який зупиняє виконання процесса, поки вказана логічна умова не стане істиною (true)

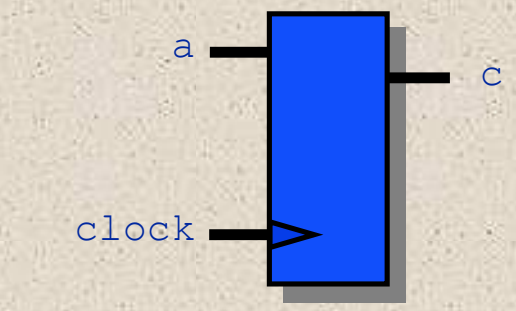
```
sync: PROCESS
BEGIN
    WAIT UNTIL clock='1';
    IF enable='1'
      THEN q_out <= d_in;
    END IF;
END PROCESS sync;
```

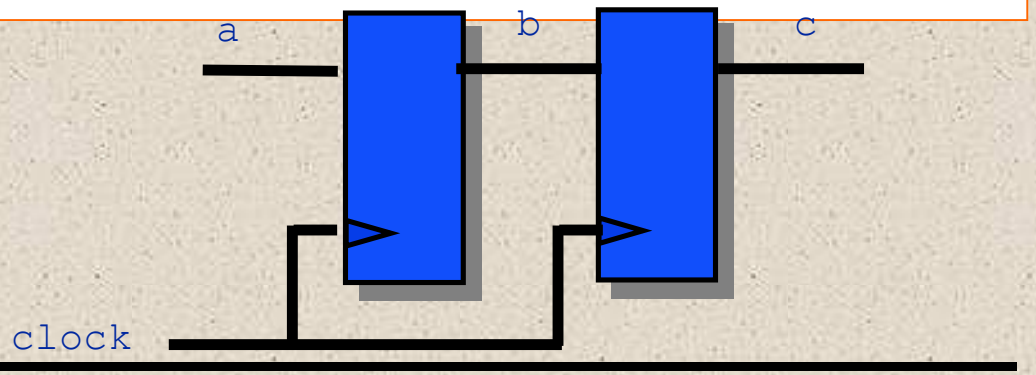


# Присвоювання сигналів в процесі

```
ARCHITECTURE arch_reg OF reg IS
SIGNAL b: std_logic
reg2: PROCESS (clock)  BEGIN
    if clock = '1' and clock'event then;
        b <= SXT(a,16) ;
        c <= b;
    End if;
END PROCESS reg2;
END arch_reg;
```

**Яка зі схем правильна?**



# Присвоювання сигналів в процесі

При виконанні процесу сигнали не змінюються зразу. В процесі, коли зустрівся оператор присвоювання, тоді тільки **планується,** що сигналу буде присвєне значення.

Сигналу присвоюється значення в момент досягнення кінця оператора **END PROCESS** або оператора **WAIT.**

Тому в попередньому прикладі будуть синтезовані 2 триггера (в момент виконання c <= b,  b має старе значення), тобто є 1 лишній триггер.

Во багатьох випадках видалити 1 непотрібний тригер можна використав параллельне присвоювання зовні процеса

# Змінні в процесі

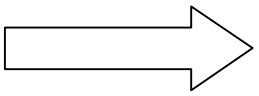
Попередня проблема може бути вирішена за допомогою змінної

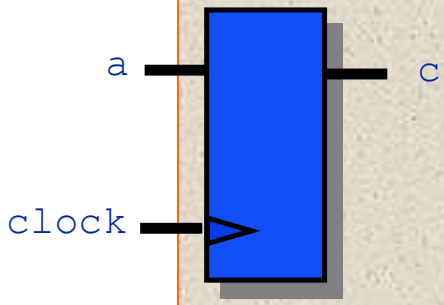
- Змінні зберігають значення таке саме, як і сигнал, але вони можуть бути використані тільки всередині ПРОЦЕСУ. Вони не можуть передавати інформацію між процесами
- На відміну від сигналу, значення, що надане змінній, доступне негайно

# Використання змінної замість сигналу

Рішенння з відображенням в 1 регістр:

```
ARCHITECTURE arch_reg OF reg IS
reg2: PROCESS (clock)
    VARIABLE b: std_logic ;
BEGIN
   if clock = '1' and clock'event then;
        b := SXT(a,16) ;
        c <= b;
   End if;
END PROCESS ;
END arch_reg;
```

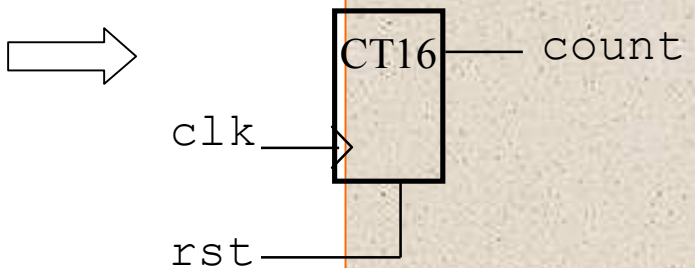




# Процес з регістром (1)

*4-розрядний лічильник з синхронним встановленням в 0*

```
USE IEEE.std_arith.ALL;
...
upcount: PROCESS (clk)
BEGIN
    IF rising_edge(clk) THEN
        IF rst = '1'   THEN
            count <= "0000";
        ELSE
            count <= count + 1;
        END IF;
    END IF;
END PROCESS upcount;
```

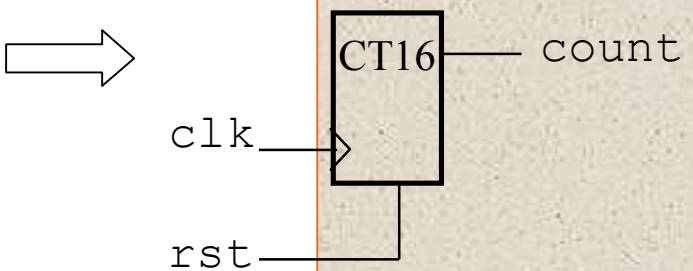



- Цей процес чутливий до змін сигналу “clk”, тобто він буде активним тільки в моменти зміни синхросигналу.

# Процес з регістром(2)

*розрядний лічильник з асинхронним встановленням в 0*

```
USE IEEE.std_arith.ALL;
...
upcount: PROCESS (clk, reset)
BEGIN
    IF reset = '1'  THEN
        count <= x'0';
    ELSIF rising_edge(clk)     THEN
        count <= count + 1;
    END IF;
END PROCESS upcount;
```

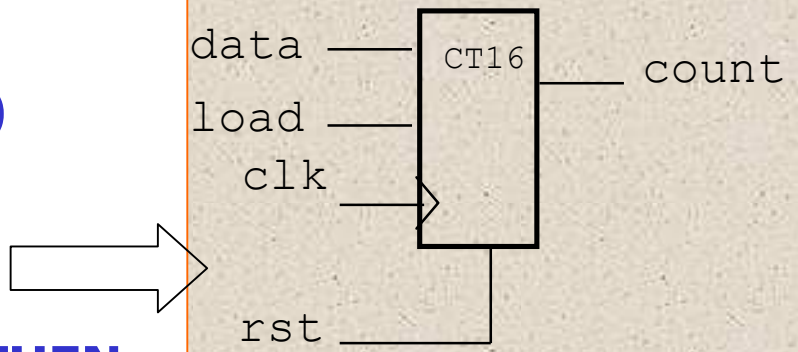



- Цей процес чутливий до змін як сигналу “clk”, так і "reset", тобто він активний в моменти зміни синхросигналу або сигналу встановлення в 0.

# Процес з регістром(3)

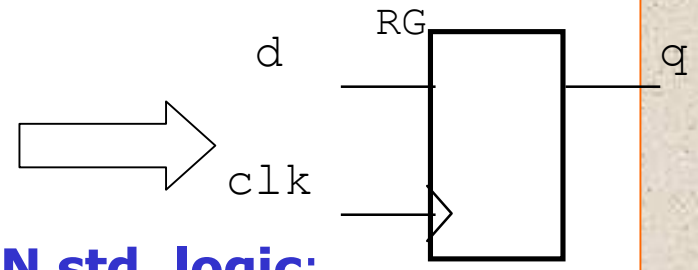
*4-розрядний лічильник з асинхронним встановленням в 0 і початковим встановленням*

```
USE IEEE.std_arith.ALL;
...
upcount: PROCESS (clk, reset)
BEGIN
    IF reset = '1' THEN
        count <= x"0" ;
    ELSIF rising_edge(clk)    THEN
        IF load = '1' THEN
            count <= data;
        ELSE
            count <= count + 1;
        END IF;
    END IF;
END PROCESS upcount;
```



# Вставлення компоненту типу регістр

Приклад: використання бібліотеки макроблоків Virtex

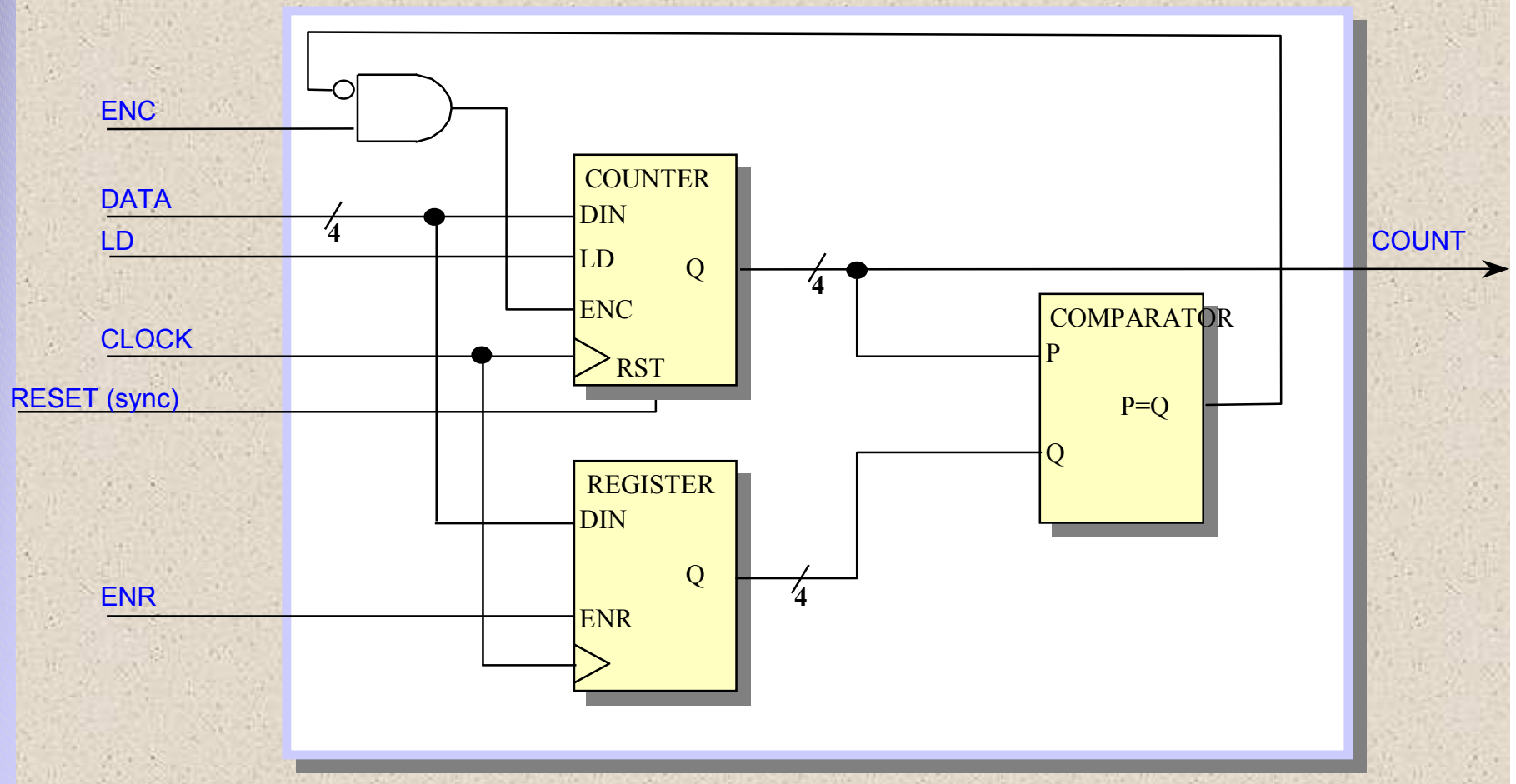



```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE WORK.Virtex.all ;
ENTITY registered IS PORT (clk,ce,rst:IN std_logic;
        d:      IN std_logic_vector(3 DOWNTO 0);
        q:      OUT std_logic _vector(3 DOWNTO 0));
END registered;

ARCHITECTURE archregistered OF registered IS
BEGIN
      REGISTER:FD4RE PORT MAP (c=>clk,r=>rst, ce=>ce,
                d0=>d(0), d1=>d(1), d2=>d(2), d3=>d(3),
                q0=>q(0), q1=>q(1), q2=>q(2), q3=>q(3));
END archregistered;
```

# Приклад

Розробити пару об’єкт- архітектура для наступної схеми:



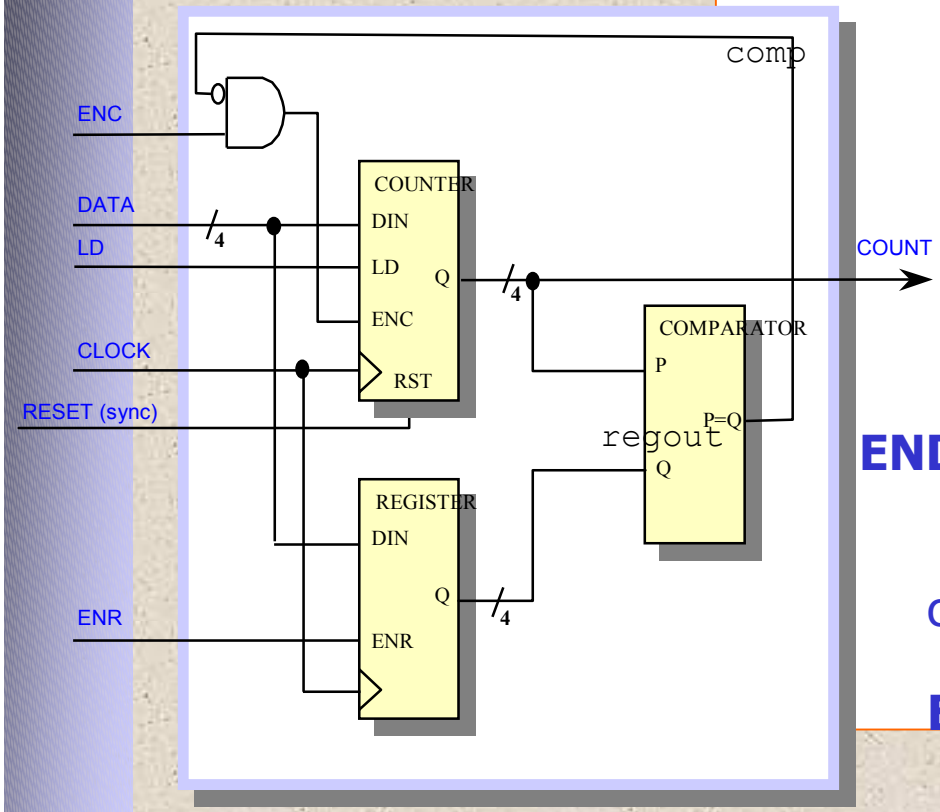
# **Приклад** (продовження)

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL, ieee.std_arith.ALL;
ENTITY ex4 IS PORT (
      clock, reset, enc, enr, ld: IN std_logic;
      data: IN std_logic_vector (3 DOWNTO 0);
      count:BUFFER std_logic_vector(3 DOWNTO 0));
END ex4;

ARCHITECTURE archex4 OF ex4 IS
      SIGNAL comp: std_logic;
      SIGNAL regout: std_logic_vector (3 DOWNTO 0);
BEGIN

reg: PROCESS (clock) --регістр
      BEGIN
            IF RISING_EDGE(clock)   THEN
                  IF enr = '1' THEN
                        regout <= data;
                  END IF;
            END IF;
      END PROCESS reg;
```

# **Приклад** (продовження)




```
cntr: PROCESS (clock)--лічильник
    BEGIN
        IF RISING_EDGE(clock) THEN
            IF reset = '1' THEN
                count <= "0000";
            ELSIF ld = '1' THEN
                count <= data;
            ELSIF enc='1' AND comp='0' THEN
                count <= count + 1;
            END IF;
        END IF;
END PROCESS cntr;

            --компаратор
    comp <= '1' WHEN  regout = count
                ELSE '0';
    END archex;
```