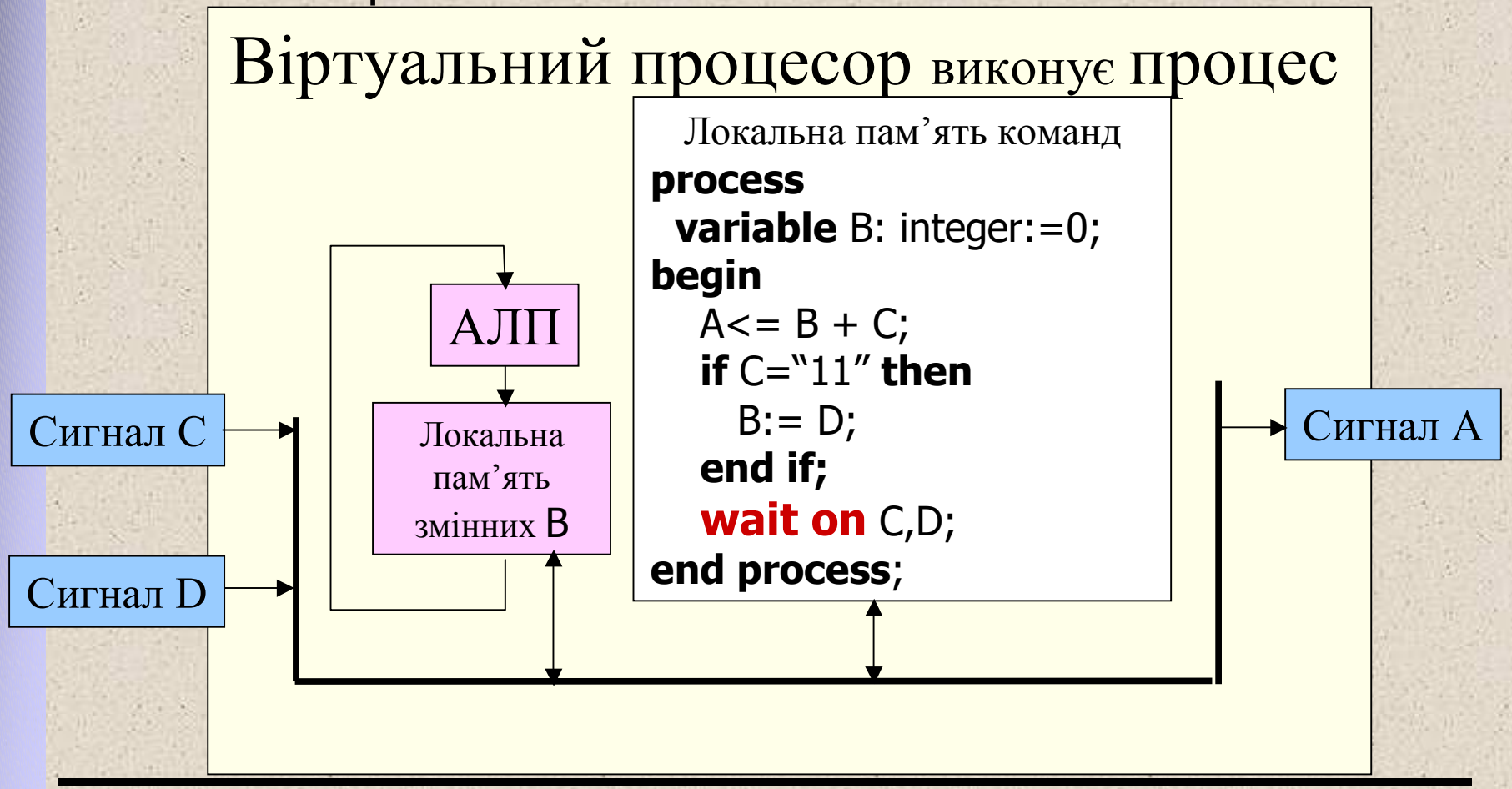
Послідовні оператори VHDL

Процес і послідовні оператори

- Оператор **process** визначає незалежний послідовний процес , який представляє собою поведінку деякої частини проекта.



Процес і послідовні оператори

- Кожен процесс може бути відмічений необов'язковою **міткою**.
- В **декларативній частині** процеса об'являються підпрограми, типи, підтипи, константи, змінні, атрибути, файли, вказівки **use ,** які відносяться тільки до цього процесу.

```
[/мітка/:] process [(список_чутливості)] is
 --декларативна_частина
begin
  --послідовні_присвоювання
end process [/мітка/];
```

Список чутливості

- **Список чутливості** вказує сигнали, зміна яких викликає запуск процеса.
- **Список чутливості** - це компактна форма завдання операторів **wait on** , які є останніми операторами в розділі *послідовні_присвоювання* .
- **Список чутливості** вставляється зразу після слова **process**.
- Процес зі списком чутливості не повинен вміщувати ніяких операторів **wait**. Також якщо в процесі є виклики процедури, то в ній не повинно бути операторів **wait** .

Оператор присвоювання сигналу

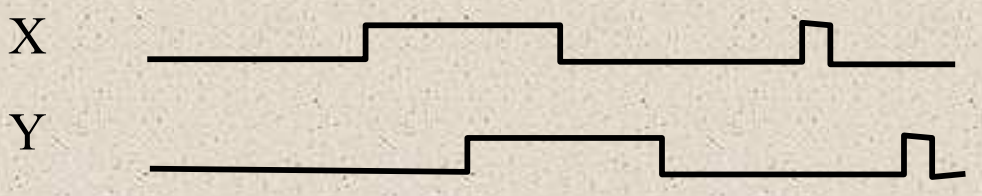
При виконанні процеса сигнали не змінюються зразу. В процесі, коли зустрівся оператор присвоювання, тоді тільки **планується,** що сигналу присвоється значення.

Сигналу **присвоюється** значення в момент досягнення кінця оператора **END PROCESS** (якщо є список чутливості) або оператора **wait.**

```
process (A,B,C) begin
    A <= B or C;  -- A обчислюється, але не присвоюється
    C<= A and B; -- старе значення A приймає участь
    A<= C xor B; -- старе значення C приймає участь
end process;      -- в цей момент присвоюється A і C
```

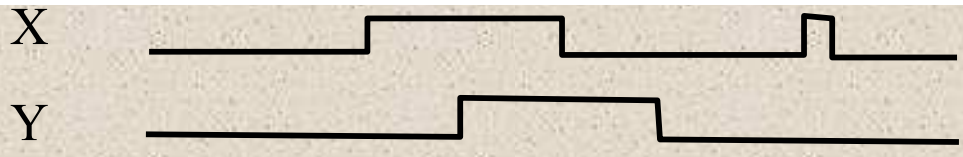
Оператори присвоювання з затримкою

```
Y <= transport X after 5 ns;
```




```
Y <= inertial X after 5 ns;
```

```
Y <= X after 5 ns;
```



Оператори чекання події **wait**

На операторі **wait** процес зупиняється

```
wait on clk, rst ;  -- продовження виконання після
                    -- зміни сигналу clk або rst

wait until rst = '1' ;  -- продовження виконання після
                        -- того, як (rst = '1') = true
wait for td + 15 ns ;  -- продовження виконання після
                       -- затримки на td + 15 наносекунд
                      --після попереднього оператора wait
```

Потім, після того, як буде досягнутий кінець **end process**, виконання продовжується з першого оператора у процесі

Оператор присвоювання змінній

- Змінні зберігають значення таке саме як і сигнал, але можуть бути використані тільки всередині ПРОЦЕСА. Вони не можуть передавати дані між процесами
- На відміну від сигналу, значення, яке присвоєне змінній, доступне негайно

```
process (A,B,C)
  variable c,d: bit:='0';
begin
    d := B or c;       -- d обчислюється і присвоюється,
  -- значення змінної c – з попереднього запуску процеса,
              --  змінна c   не має відношення до сигналу C
    c := d and B;     -- d приймає участь в обчисленні
    A<= c xor B;      -- нове значення c приймає участь
end process;          -- в цей момент присвоюється A
```

Умовні оператори

Оператор **if** виконує приєднані до нього альтернативні послідовності операторів в залежності від булевського значення однієї або декількох умов.

```
if /умова/ then
      --послідовні_оператори
elsif /інша_умова/ then
      --послідовні_оператори
else
       --послідовні_оператори
end if;
```

Умовні оператори

- Оператор **if** керує умовним виконанням інших послідовних операторів. В ньому стоїть принаймні одна булевська умова (після ключового слова). Решта умов вказуються після слів **elsif**. Слово **else** інтерпретується як **elsif** true.
- Умови перевіряються одна за одною поки якась з них не виявиться істинною (true) або не залишиться жодної перевіреної умови.
- Коли умова істинна, то виконується послідовність операторів після слова **then**. Якщо жодна умова не істинна, то керування передається оператору після оператора **if**.

```
if /умова/ then
        --послідовні_оператори
elsif /інша_умова/ then
        --послідовні_оператори
else
       --послідовні_оператори
end if;
```

Умовні оператори

Приклад оператора **If**

```
Signal \код_операції\ :
      Bit_vector ( 1 downto 0);

I3 : if \код_операції\  = “01”  then
            F := \Операнд_1\  + \операнд_2\;
      elsif \код_операції\ = “10”  then
            F := \Операнд_1\  - \операнд_2\;
      else
            F :=  “0000000”;
      end if I3;
```

Умовні оператори

Не слід забувати, що результат виразу умови повинен бути типу **boolean**. Наступний оператор - **неправильний**

```
if a and not b  then
          F := c;
end if ;
```

якщо операнди a і b – не булевські
(на практиці частіше всього це так).
Слід записати **правильно так**:

```
if (a and not b) ='1'  then
          F := c;
end if ;
```

Оператор вибору

Оператор **case** виконує одну з декількох послідовностей операторів в залежності від значення виразу вибору.

```
Case  /вираз_вибору/  is
    when /варіант_вибору1/ => {послідовний оператор}
    {when /варіант_вибору2/ => {послідовний оператор}}
end case;
```

Варіанти вибору не повинні пересікатися.
Всі значення виразу выбору з повної множини його значень повинні бути перелічені в альтернативах **when**.
Якщо це неможливо, то останнім оператором повинен бути оператор альтернативи, що доповнює перелічені значення до повної множини:

```
when others => {оператор};
```

Оператор вибору

Варіант вибору повинен бути того ж типу, що і вираз вибору.
Варіант вибору може включати
-один елемент множини значень виразу,
-діапазон таких значень
- їх перелік,
наприклад,

```
Signal \код\:integer range 1 to 10;
case \код\ is
  when 1=>        y<= 0;
  when 2|3|10 => y<= 1;
  when 5 to 7 => y<= 2;
  when others => null;
end case;
```

Null – оператор, який нічого не виконує

Оператор вибору

Вираз вибору повинен бути простим виразом.

Наприклад, *неправильний* код

```
Signal k1,k2:bit_vector (0 to 1); . . .
case k1&k2 is          -- вираз не простий
  when “0101”=> y<= 0;
  when “1111”=> y<= 1;
  when others => y<= 2;
end case;
```

повинен бути переписаний як

```
  k12<= k1 & k2;
case k12 is
  when “0101”=> y<= 0;
  when “1111”=> y<= 1;
  when others => y<= 2;
end case;
```

Оператор циклу

- Оператор циклу **loop** вміщує послідовність операторів, які треба виконати кілька разів.
- Оператор циклу вміщує ряд умов повторення або кількість ітерацій.

```
Loop_statement ::=
    [/мітка/:] [/схема_ітерації/] loop
                  {послідовність операторів}
    end loop [ /мітка/];

/схема_ітерації/ ::=  while /умова/ |

                for /специфікація_параметрів_loop/

/специфікація_параметрів_loop/ ::=
    /змінна_циклу/ in /дискретний_діапазон/
```

Оператор циклу

Процес описує генератор синхросерії.
Він вміщує цикл без схеми ітерації,
який буде повторюватись безкінечно.

```
Signal Clock : Bit := '0';
……
Clk_1 : process ( Clock)
      begin
        L1 : loop
                Clock <= not Clock after 5 ns;
             end loop L1;
      end process Clk_1;
```

Оператор циклу

Цикл L2 буде повторюватись

поки значення змінної не більше 8.

Коли і прийме значення 9, цикл зупиниться.

```
Shift_2 : process (Input_X)
variable i : POSITIVE := 1;
begin
   L2 : while i <= 8 loop
      Output_X(i) <= Input_x(i) after 5 ns;
      i := i +1;
   end loop L2;
end process Shift_2;
```

Оператор циклу

Параметр count заставить виконуватись цикл 8 разів, поки count буде змінюватись від 1 до 8.

```
Shift_3 : process ( Input_x)
begin
     -- змінну циклу count об'являти не треба
    L3 : for count in 1 to 8 loop
   Output_X(count)<=Input_X(count) after 8 ns;
    end loop L3;
end process Shift_3;
```

Оператор циклу

```
Variable vec:bit_vector(1 to n);
Variable numb:natural;
...
Numb:=0;

for i in 1 to n loop
    next when vec(i)='0';

    numb:=numb+1;  -- підрахунок одиниць в vec

end loop;

for i in 1 to n loop
    exit when vec(i)='0';

    numb:=numb+1;       -- підрахунок нулів перед
        --першою значащою одиницею в vec

end loop;
```

Виклик процедури і повернення з неї

Наприклад, в пакеті IEEE.Math_Real є процедура генерації випадкових чисел:

```
procedure UNIFORM(variable SEED1,SEED2:inout
                                  POSITIVE; variable X:out real);
```

Вона може бути викликана зі зв'язуванням параметрів:

```
variable s1,s2:natural:=12345;
variable Random:real;
…
UNIFORM(X=> Random, SEED1=>s1,SEED2=>s2);
```

Або без зв'язування:

```
UNIFORM(s1,s2, Random);
```

Або загалі без параметрів, але тоді імена фактичних параметрів мають співпадати з іменами формальних параметрів:

```
UNIFORM;
```

Оператори **Assert** і **Report**

Призначені для видачі на консоль текстових повідомлень про хід моделювання. Наприклад, оператор

```
assert a(i)='0'
report "біт" & integer'image(i) &" не дорівнює 0";
```

Викликає на екран дисплея повідомлення:

```
# : ERROR  : біт  1  не дорівнює 0
# : Time: 2000 ns,  Iteration: 0,  TOP instance.
```

Якщо a(i) $\neq$ 0 на 2-й мікросекунді моделювання.

Тут атрибут integer'image(i) повертає строку, в якій ціле і є в строковому вигляді.

Якщо треба видати повідомлення без перевірки умови, то використовують тільки фразу **report** , як наприклад:

```
report "кінець моделювання" severity failure;
```

Тут фраза **severity** означає рівень критичності знайденої помилки і може набути значення з множини: **note, warning, error** і **failure**.