Процедури і функції. Пакети.

Структура програми

Підпрограми

Концепція

Можна об’явити в:

- Пакетах
- Процесах
- Архітектурах

Вміщують послідовні оператори

Виконуються послідовно – як у звичайних програмах

Розрізняють: **Функції** - Повертається 1 значення і приймають участь у виразах

Процедури - Повертають одне або більше значень або нічого

Підпрограми

- **Визначення функції**
  - **Ім'я**, формальні параметри, тип , що повертається
  - Послідовність операцій, що виконуються
  - Функція синтезується, якщо її параметри визначаються статично

```
function parity (word: bit_vector) return bit is
variable tmp: bit;
begin
    tmp := '0';
    for i in word'range loop
        tmp := tmp xor word(i);
    end loop;
    return tmp;    -- у функції повинен бути оператор return
end parity;
```

Підпрограми

Виклик функції

```
entity parity _checker is
        port( single_w:  in bit_vector(15 downto 0);
              double_w:  in bit_vector(31 downto 0);
              s_parity: out bit;
              d_parity: out bit);
end parity_checker;

architecture functional of parity_checker is
-- тут повинна бути об'ява функції, якщо не в пакеті
begin
   s_parity <= parity(single_w);-- факт.параметр single_w
   d_parity <= parity(double_w);--факт.параметр double_w
end functional;
```

Підпрограми

Об’ява процедури:

- Ім’я і формальні параметри
- Параметри мають режими вводу-виводу і можуть бути константами, змінними, сигналами
- Неявно заданий напрям in і об’єкт constant

```
procedure parity (word: in bit_vector;
                  signal parity_bit: out bit);
    variable tmp: bit;
    begin
        tmp := '0';
        for i in word'range loop
            tmp := tmp xor word(i);
        end loop;
        parity_bit <= tmp;    -- note this is a signal assignment
    end parity;
```

Підпрограми

Виклик процедури: Замість **формальних параметрів** підставляються **фактичні параметри**

Фактичні параметри за типом повинні відповідати **формальним параметрам**

Виклик процедури синтезується, якщо її параметри визначаються статично

```
entity parity _checker is
        port( single_w:  in bit_vector(15 downto 0);
              double_w:  in bit_ vector(31 downto 0);
              s_parity, d_parity : out bit);
end parity_checker;
architecture procedural  of parity_checker is
-- тут повинна бути об'ява процедури, якщо не в пакеті
begin
        parity(single_w, s_parity);
        parity(double_w, d_parity);
    end procedural;
```

Пакет

Об’ява пакету

-В пакет збирають декларації об’єктів та типів, пов’язаних загальною ознакою.

-Декларації з пакету можна використовувати в різних частинах проектів, посилаючись на цей пакет.

-Багато пакетів стандартизовано

-Декілька пакетів, що служать одній меті, збирають в бібліотеку library.

-Бібліотека, в якій зібрані програми і пакети користувача, неявно має назву WORK.

Синтаксис об’яви пакета:

```
\об’ява пакета\::= package \ідентифікатор\ is
                        {об’ява в пакеті}
                         end [package][\ідентифікатор\];
```

Пакет

Тіло пакету

-Тіло пакету необхідно приводити в парі з об’явою пакета, якщо в тому пакеті об’явлені підпрограми або відкладені константи.

В тілі пакету повинні приводитись повні специфікації процедур і функцій, присвоювання константам, що задекларовані в об’яві цього пакету.

Синтаксис тіла пакета:

\Тіло пакету\::= **package body** \ідентифікатор\ **is**
{об’ява в тілі пакета}
end [**package body**][\ідентифікатор\];

Пакет

Приклад пакету

```
Package funpack is
  function parity (word: bit_vector) return bit;
end funpack;

package body funpack is --власне опис функцій пакету
function parity (word: bit_vector) return bit is
    variable tmp: bit;
    begin
        tmp := '0';
        for i in word'range loop
            tmp := tmp xor word(i);
        end loop;
        return tmp;
    end parity;
end package body funpack;
```

Перезавантаження підпрограм

Будь-яка підпрограма (і оператор) може бути перезавантажена (overloaded)

Викликається така відповідна функція, для якої співпадають типи фактичних і формальних параметрів

```
package my_pack is
 function parity (w: bit_vector)
        return bit;
 function parity (w: std_logic_vector)
        return bit;
function parity (w: unsigned)
        return bit;
end my_pack;

package body my_pack is
-- визначення всіх функцій parity
end my_pack;
```

```
use work.my_pack.all;
architecture example of entity_name
 is
signal word1: bit_vector(31 downto 0);
signal unword: unsigned(15 downto 0);
signal stdbus:
              std_logic_vector(7 downto 0);
signal p1, p2, p3: bit;
begin
  p1 <= parity(word1);
  p2 <= parity(unword);
  p3 <= parity(stdbus);
end example;
```

Перезавантаження підпрограм

Власна версія функції додавання

```
-- додавання 2-х векторів
function "+"(A:bit_vector; B:bit_vector) return bit_vector is
     variable CARRY : bit:='0';
     variable SUM : bit:='0';
     constant SIZE : integer:=A'length;
     variable RESULT : bit_vector(SIZE-1 downto 0);
begin
  for i in A'reverse_range loop
    SUM:=A(i) xor B(i) xor (CARRY);
    CARRY:=(A(i) and B(i)) or ((A(i) or B(i)) and CARRY);
    RESULT(i):=SUM;
  end loop;
  return RESULT;
end;
```

Перетворення типів

VDHL – строго типізована мова і забороняє виконувати присвоєння об’єкта одного типу об’єкту іншого типу, виконувати операції і підпрограми, якщо вони не визначені для даного типу вхідних і вихідних об’єктів.

Часто необхідно або зручно в одній частині проекту оперувати з об’єктами одного типу , а в іншій – з об’єктами іншого типу. Наприклад, виконувати лічбу з цілим типом, а логічні операції – з булевським типом.

Існує 2 шляхи **переходу** від одного типу до іншого ...

Перетворення типів

1)Перехід типу (Type Casting)

“близько зв’язані" типи можуть переходити один в другий (cast from one to another). Це:

- Масиви, вектори однакової довжини з однаковими типами індексів і елементів
- Цілі і дійсні числа (Integers <-> reals)
- Типи bit, boolean а тим більше std_logic не є близько зв’язаними

```
  signal un1, un2: unsigned(7downto 0);
  signal stdul1: std_ulogic_vector(7 downto 0);
  signal int1: integer;
  signal real1;
begin
  stdul1 <= un1; -- type conflict
   int1 <= integer(real1);
  un2 <= "00001111"; -- may be ambiguous
  un2 <= unsigned'("00001111"); -  clearly unsigned
```

Перетворення типів

2) Функції перетворення типів

•Для перетворення операнда одного типу у зовсім інший тип, наприклад, цілого типу у вектор використовують бібліотечну функцію.

•У бібліотеці **IEEE.std_logic_arith** зібрано підпрограми переходу між типами integer та std_logic_vector.

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.std_logic_arith.ALL;

entity count8 is
  port(clk, rst : in std_logic;
  qout : out std_logic_vector(2 downto 0)
  );
end count8;
```

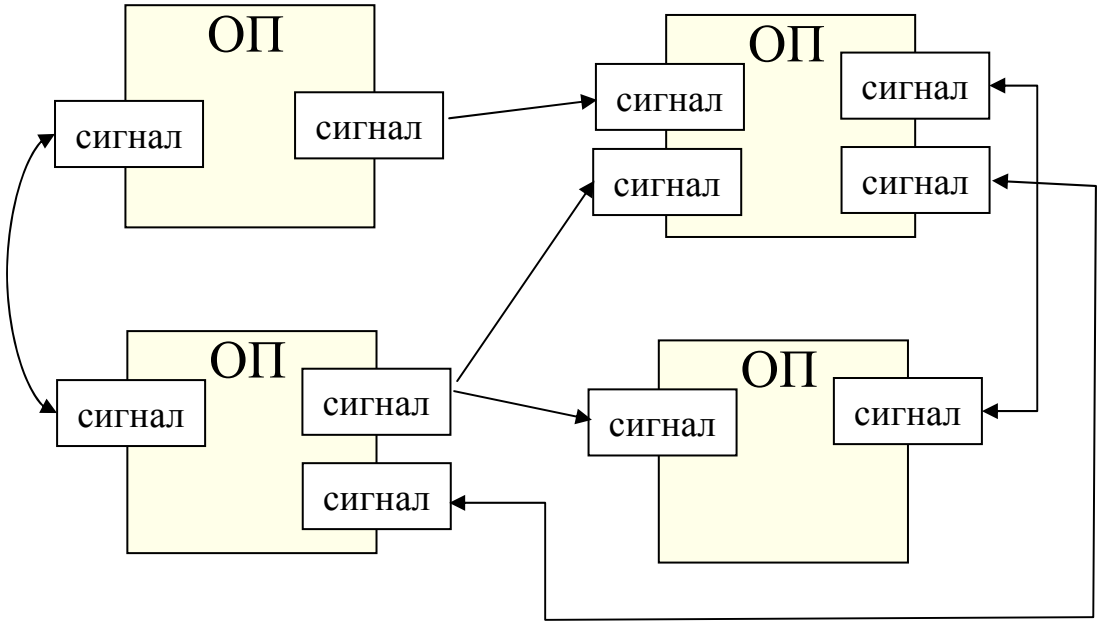
```
Architecture bin of count8 is
 signal count : integer range 0 to 7;
Begin
  count <= 6;
  qout <=conv_std_logic_vector(count,3);
 end bin;
```

Обчислювальна модель для реалізації VHDL

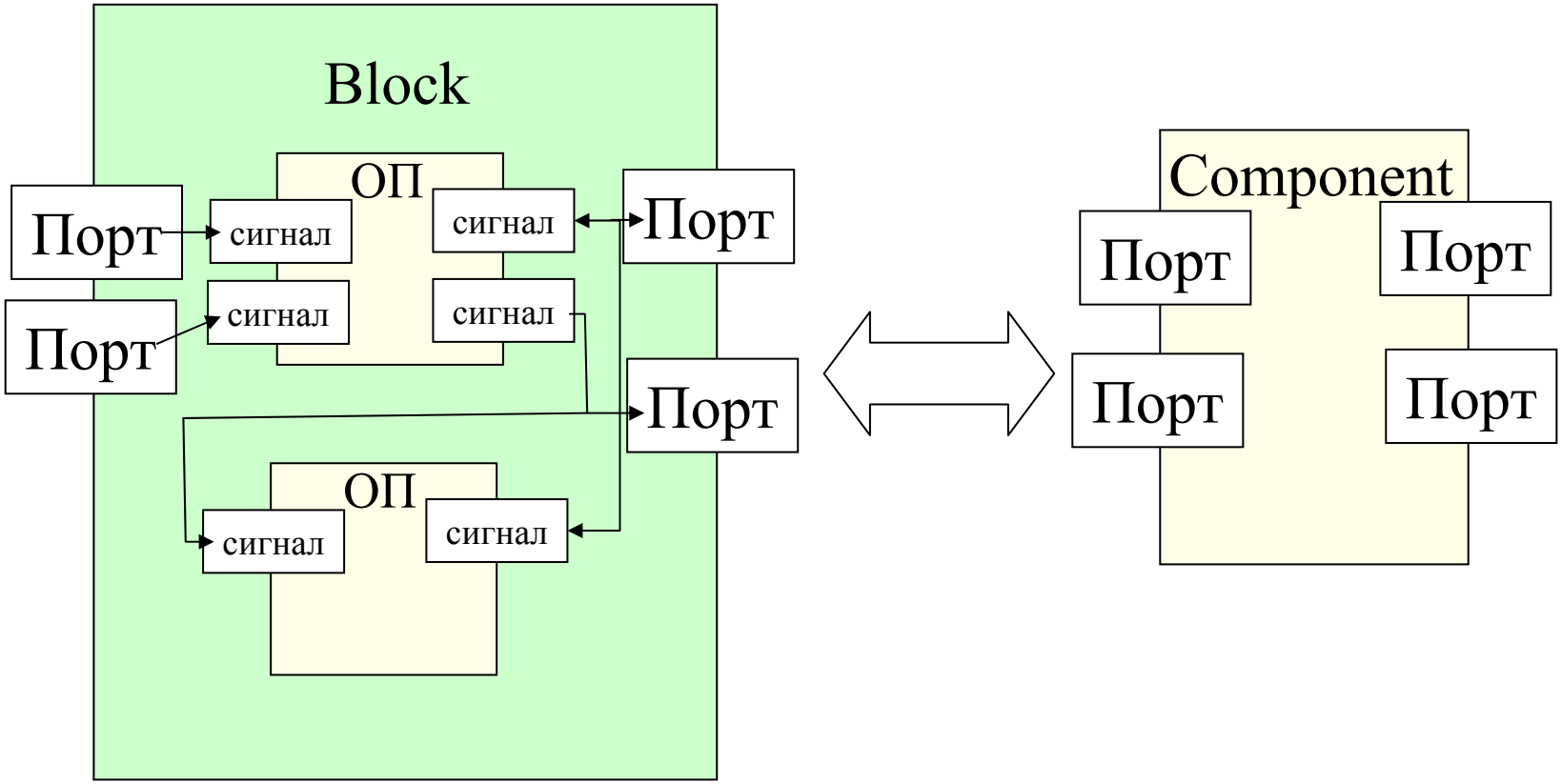
Консоль

Дискова пам’ять

Пам’ять глобальних змінних



Обчислювальна модель для реалізації VHDL



Структура програми

Об’єкт проекта описується згідно синтаксису:

\ Об’єкт проекта\::= [\опис **library**]
[\опис **use**]
\об’ява об’єкта\
\тіло архітектури\
[\об’ява конфігурації\]

Entity і Architecture

Опис проекта на VHDL складається з двох частин:

- декларація **ENTITY**
- опис **ARCHITECTURE**

- ENTITY описує входи- виходи проекта
- ARCHITECTURE описує функціонування проекта
- Для кожної архітектури повинно бути **entity**, тому до проекту звертаються як до пари **ENTITY(ARCHITECTURE)**

Entity - опис інтерфейса

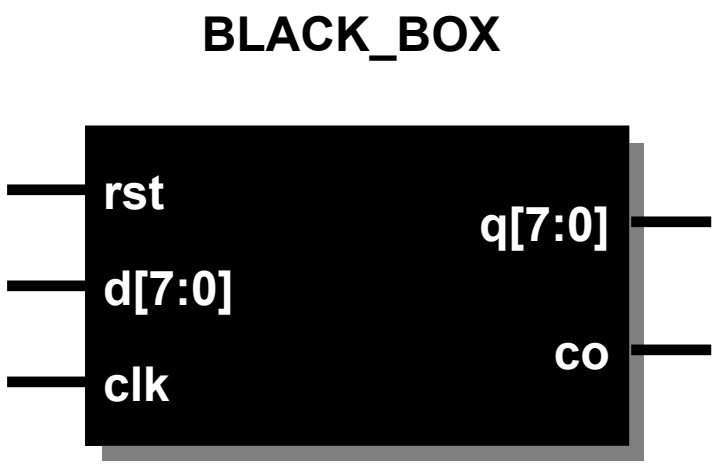
```
entity ENTITY_NAME is
   port (
      <port_declarations>
         );
end ENTITY_NAME;
```

```
architecture ARCH_NAME of ENTITY_NAME is
begin
   <statements>
end ARCH_NAME;
```

Entity - опис інтерфейса

Концепція “чорного ящика”

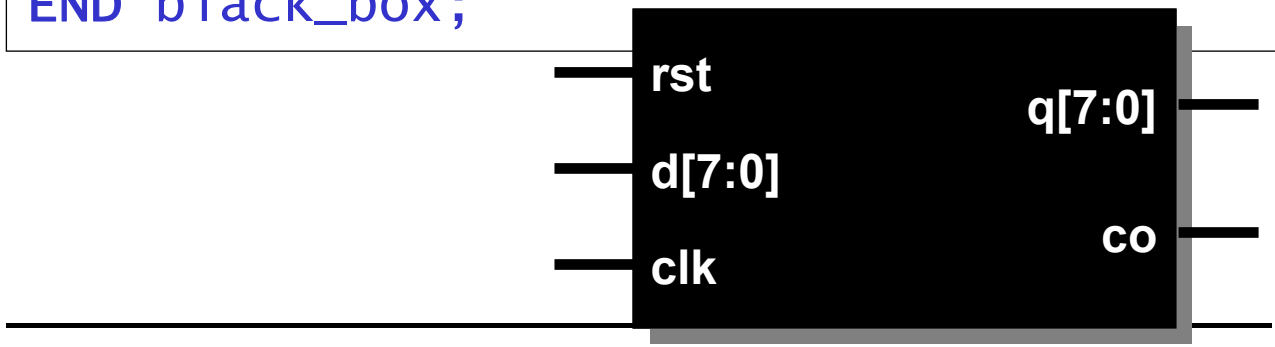
- ENTITY описує інтерфейс чорного ящика (тобто входи-виходи проекту)



Entity - опис інтерфейса

Приклад опису ENTITY “BLACK BOX”

```
ENTITY black_box IS
      PORT (
            clk:  IN  std_logic;
            rst:  IN  std_logic;
            d  :  IN  std_logic_vector(7 DOWNTO 0);
            q  :  OUT std_logic_vector(7 DOWNTO 0);
            co :  OUT std_logic
      );
END black_box;
```



Entity - опис інтерфейса

Структура Entity :

- **entity_name** – будь-яке ім’я
- **generics** - константи для настройки проекта
- **name** - ідентифікатор сигнала-порта
- **mode** - режим (напрямок) вводу-виводу порта
- **type** - вказує тип сигналу - BIT, STD_LOGIC

```
ENTITY entity_name IS
      -- optional generics
      PORT (
            name : -- mode type ;
            ...
      ) ;
END entity_name;
```

Entity - опис інтерфейса

Розділ **entity** вміщує опис портів об’єкта проекта

- через ПОРТИ виконується обмін даними
- під ПОРТАМИ маються на увазі виводи пристроя

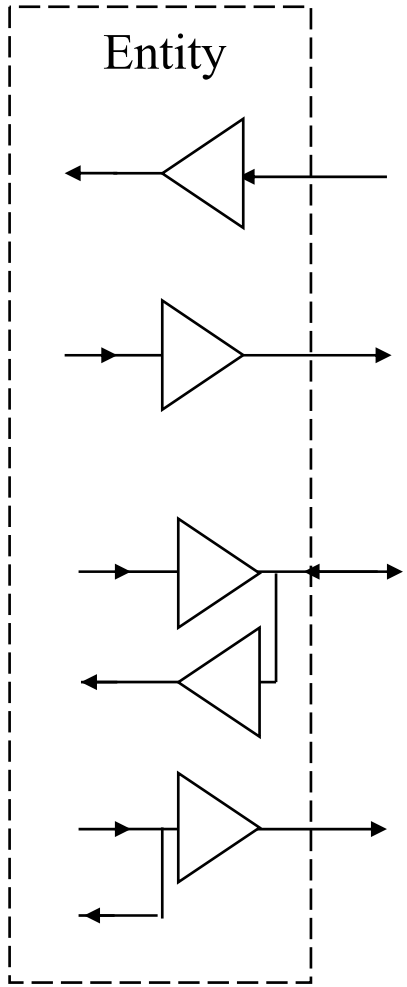
Порти завжди описуються рядом ознак

- Ім’я порту,
- режим порту,
- тип порту

Entity - опис інтерфейса

Режим порту вказує напрямок передачі даних:

- IN дані тільки входять в entity
- OUT дані тільки виходять назовні (і не використовуються всередині)
- INOUT дані передаються в двох напрямках (входять в і виходять з entity)
- BUFFER дані видаються назовні і використовуються всередині



Architecture - опис поведінки моделі

Об’ява архітектури

arch_name – будь-яке ім’я

- **entity_name** – ім’я об’єкта - entity
- допоміжні об’яви **сигналів**
- тіло архітектури - поведінка моделі
- **оператори** описують функціонування архітектури

```
ARCHITECTURE arch_name OF entity_name IS
-- допоміжні об’яви сигналів
BEGIN
      --VHDL -оператори
END arch_name;
```

Architecture - опис поведінки моделі

Архітектура описує поведінку і структуру **entity** - чорного ящика.

Структурний опис

- включення екземплярів (Instantiation – розміщення і з”єднання як в схемі) блоків, які називаються компонентами

```
U1:FDE port map(C=>clk,CE=>ce,D=>d1,Q=>q1);
```

Поведінковий опис і опис потоків даних

- Алгоритмічний опис:

```
IF a = b THEN state <= state5;
```

- Булевські рівняння (так званий, стиль потоків даних):

```
x <= a OR (b AND c);
```

Об’ява конфігурації

\об’ява конфігурації\::=

configuration \ідентифікатор\ **of** \ім’я об’кта\ **is**
for \ім’я архітектури\
{**for** \вказівники вставки компонента\: \ім’я компонента\
\вказівник зв’язування\;
end for;}
end for;
end [**configuration**] [\ідентифікатор\];

\ вказівники вставки компонента \::=\мітка вставки компонента\
{,\мітка вставки компонента\} | **others** | **all**

\вказівник зв’язування\::= **use entity**
\ім’я об’єкта\ [(\ідентифікатор архітектури\)]