Лабораторна робота 4

Використання оператора Generate

1 **Мета лабораторної роботи**:

оволодіти знаннями і навичками по проектуванню арифметико-логічних пристроїв (АЛП) для сучасних компьютерів. Ознайомитись з основами розробки проектів для програмованих логічних інтегральних схем (ПЛІС). Вивчити спосіб проектування логічних схем Generate. Навчитись складати стенд для іспитів і користуватись ним.

Теоретичні відомості

Якщо необхідно кілька разів повторити один або декілька параллельних операторів, то використовують оператор **generate**. Його синтаксис:

```
\оператор generate\::= \мітка\: for \ідентифікатор\ in \діапазон\ generate
                [{\об'ява типу сигналу,константи підпрограми,тощо\}
                        begin]
                            { \параллельний оператор\}
                        end generate [\мітка\];
```

Мітка оператора **generate** необхідна для позначення згенерованої структури, \ідентифікатор\ - це параметр оператора **generate**, а фраза \діапазон\ -діапазон його зміни. Вони мають такі самі синтаксис і семантику, як і в операторі **loop**. В операторі можуть бути вставлені такі самі об'яви, як в декларативній частині тіла архітектури.

На відміну від оператора **loop**, який повторює в циклі один або декілька послідовних операторів, оператор **generate** робить кілька копій параллельних операторів, коли параметр оператора пробігає всі значення з заданого діапазону.

Для того, щоб керувати структурою пристрою, що проектується, використовується умовний оператор **generate**. Його синтаксис:

```
\умовний оператор generate\::= \мітка\: if \булевський вираз\ generate
                [{\об'ява типу сигналу,константи підпрограми,тощо\}
                        begin]
                           { \параллельный оператор\}
                        end generate [\метка\];
```

В залежності від умови, яка задана булевським виразом, оператор вставляє або ні в структуру пристроя вузли, які представлені параллельними операторами. Так як цей булевський вираз впливає на структуру пристрою, він повинен бути статичним.

2. Завдання для лабораторної роботи:

– розробити функціональну схему n-розрядного АЛП, що виконує задані функції;

– змоделювати роботу АЛП.

Результати виконання оформлюються у вигляді звіту (протоколу). Звіт повинен вміщувати:

- опис і рисунок заданого варіанта АЛП,
- хід проектування і схему АЛП,
- графіки сигналів, знятих при іспитах АЛП,
- висновки.

Варіант завдання, крім n, такий самий, як в лабораторній роботі 3. Розрядність n АЛП береться з наступної таблиці.

№ завдання	Розрядність n АЛП	№ завдання	Розрядність n АЛП	№ завдання	Розрядність n АЛП
1	4	8	24	15	12
2	8	9	28	16	16
3	10	10	32	17	18
4	12	11	16	18	20
5	16	12	4	19	24
6	18	13	8	20	28
7	20	14	10	21	32

3. **Виконання роботи**

За основу беруться результати лабораторної роботи 3, а саме – модель одного розряду АЛП. Ця модель сприймається як компонент в новому проекті.

Результуючу VHDL-програму тестують в САПР ActiveHDL.

Для тестів розробляють стенд для іспитів, в якому АЛП, який тестується включений як компонент. Цей стенд можна підготувати за допомогою утіліти Generate Testbench, яку вибирають в меню Tools симулятора ActiveHDL.

Одержані графіки сигналів заносять в протокол лабораторної роботи.

4 **Приклад виконання роботи**

Розглянемо приклад проектування і- го розряду АЛП, що виконує функції

Y = max(A , B) при F=0 , Y = min(A, B) при F=1.

Нехай розрядність n дорівнює 12. При цьому 12-бітний АЛП повинен складатись з 12 таких розрядів. Його структура показана на рис. 1.

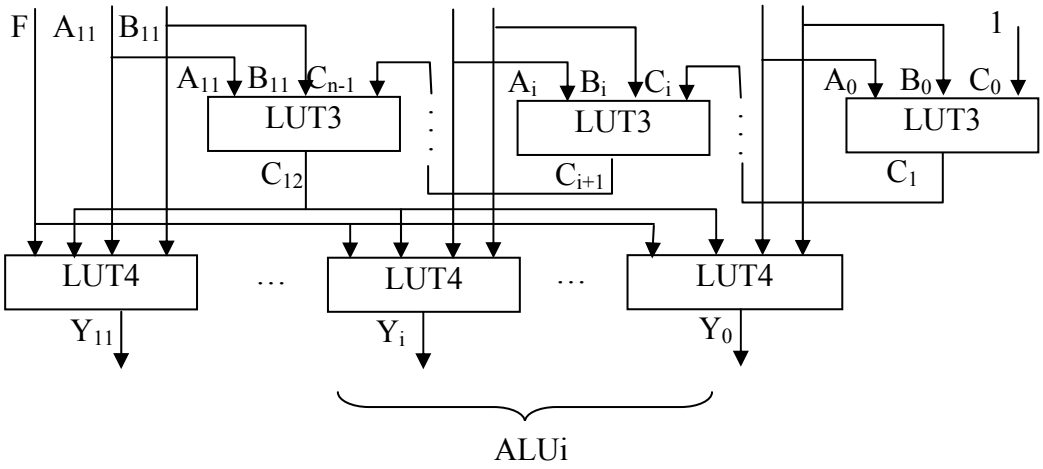

Рис.1. Структура АЛП

Тоді опис моделі багаторозрядного суматора буде наступний

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity ALU12 is
 port( a  : in STD_LOGIC_Vector(11 downto 0);
       b  : in STD_LOGIC_Vector(11 downto 0);
       f  : in STD_LOGIC;
       c12 :out STD_LOGIC; -- вихідний перенос
       y  :out STD_LOGIC_Vector(11 downto 0) );
end ALU12;
```

```
architecture ALU12 of ALU12 is
 component ALUi is
      port( a  : in STD_LOGIC;
            b  : in STD_LOGIC;
            ci : in STD_LOGIC;
            cn : in STD_LOGIC;
            f  : in STD_LOGIC;
            co :out STD_LOGIC;
            y  :out STD_LOGIC  );
 end  component;

 signal    c:STD_LOGIC_Vector(12 downto 0);

begin
 c(0)<='1';
 U_ALU:for i in 0 to 11 generate
      Ui:  ALUi port map( a=>a(i),
            b=>b(i),
            ci=>c(i),
            cn=>c(12),
            f =>f,
            co =>c(i+1),
            y  =>y(i));
 end generate;
end ALU12;
```

Стенд для іспитів можна використати такий:

```
library ieee;
use ieee.std_logic_1164.all;
entity alu12_tb is
end alu12_tb;

architecture TB_ARCHITECTURE of alu12_tb is
 component alu12
 port(
      a : in std_logic_vector(11 downto 0);
      b : in std_logic_vector(11 downto 0);
      f : in std_logic;
      c12 : out std_logic;
      y : out std_logic_vector(11 downto 0) );
 end component;
 signal a : std_logic_vector(11 downto 0);
 signal b : std_logic_vector(11 downto 0);
 signal f : std_logic;
 signal c12 : std_logic;
 signal y : std_logic_vector(11 downto 0);
begin

 -- Unit Under Test port map
 UUT : alu12
      port map (
            a => a,
            b => b,
            f => f,
            c12 => c12,
            y => y
      );
```

```
  -- Add your stimulus here ...
  f<='0','1' after 100 ns;
  a<=X"000", X"111" after 40 ns, X"aaa" after 80 ns,
             X"333" after 120 ns,X"bbb" after 160 ns;
  b<=X"000", X"555" after 20 ns, X"666" after 60 ns,
             X"777" after 100 ns,X"888" after 140 ns;

end TB_ARCHITECTURE;
```

Згенеровані графіки сигналів показані на рис.2.

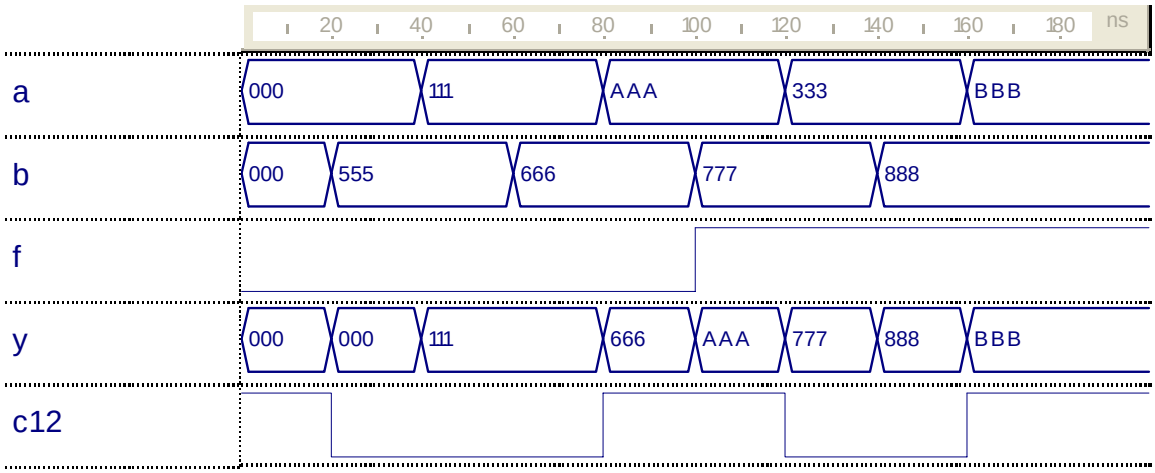

Рис.2. Графіки тестування АЛП

Аналіз графіків показує, що АЛП функціонує корректно – при f=0 вибирається мінімальне число, а при f=1 – максимальне.