Лабораторная работа № 5.
Исследование четырехразрядного сумматора.

Цель работы. Изучение функционирования устройства, позволяющего получить арифметическую сумму двух 4-разрядных двоичных чисел.
Постановка задачи. Устройство должно содержать блок 4-разрядного сумматора, блок регистра результата, блок триггера переноса и блок кодопреобразователя, позволяющего получить вывод числа на сегментный индикатор. Структура программы рассмотрена на упражнении, предшествующем лабораторной работе.

Задание на работу в лаборатории.

1. Создать проект под названием **summ4**. Открыть VHDL файл и записать **Программу 5.1**, отражающую функционирование регистра результата сложения. Сохранить файл с названием **rg4**, установить его старшим в иерархии и откомпилировать.
2. Пользуясь «Приложением 2», получить диаграммы при **интервалах импульса синхронизации – 20нс, для данных – 50нс.**

Программа 5.1.

```
library ieee;
use ieee.std_logic_1164.all;
entity rg4 is
port(d_i: in std_logic_vector(3 downto 0);
clk_i: in std_logic;
d_o: out std_logic_vector(3 downto 0));
end rg4;
architecture BB of rg4 is
begin
process(clk_i)
begin
if(rising_edge(clk_i)) then  d_o<=d_i;
end if;
end process;
end;
```

3. Открыть новый VHDL-файл и записать в него Программу 5.2 для триггера переноса. Сохранить файл с названием **trig**, установить его старшим в иерархии и откомпилировать.
4. Пользуясь «Приложением 2», получить диаграммы при **интервалах**

импульса синхронизации – 20нс, для данных – 50нс.

Программа 5.2.

```
library ieee;
use ieee.std_logic_1164.all;
entity trig is
port(d_i: in std_logic;
clk_i: in std_logic;
d_o: out std_logic);
end trig;
architecture BB of trig is
begin
process(clk_i)
begin
if(rising_edge(clk_i)) then  d_o<=d_i;
end if;
end process;
end;
```

5. Открыть новый VHDL-файл и скопировать в него текст программы для кодопреобразователя из лабораторной работы №2.***В объявлении входов и выходов измените bit_vector на std_logic_vector.*** Сохранить файл под именем coder, установить старшим в иерархии, скомпилировать. Текст файла в данной работе представлен **Программой 5.3.**

Программа 5.3.

```
library ieee;
use ieee.std_logic_1164.all;
entity coder is
port(symb: in std_logic_vector(3 downto 0);
ind: out std_logic_vector(6 downto 0));
end coder;
architecture bbb of coder is
begin
p0:process(symb)
begin
case symb is
when"0000"=>ind<="1000000";
when"0001"=>ind<="1111001";
when"0010"=>ind<="0100100";
when"0011"=>ind<="0110000";
```

```
when"0100"=>ind<="0011001";
when"0101"=>ind<="0010010";
when"0110"=>ind<="0000010";
when"0111"=>ind<="1111000";
when"1000"=>ind<="0000000";
when"1001"=>ind<="0010000";
when"1010"=>ind<="0001000";
when"1011"=>ind<="0000011";
when"1100"=>ind<="1000110";
when"1101"=>ind<="0100001";
when"1110"=>ind<="0000110";
when"1111"=>ind<="0001110";
end case;
end process;
end;
```

6. Открыть новый VHDL-файл и записать в него **Программу 5.4** для полусумматора. Сохранить файл с названием **add1**, установить его старшим в иерархии и откомпилировать.
7. Пользуясь «Приложением 2», получить диаграммы при **интервалах импульса на входе a1 – 20нс, на входе b1 – 30нс.**

Программа 5.4.

```
library ieee;
use ieee.std_logic_1164.all;
entity add1 is
port(a1,b1: in std_logic; c1,s1: out std_logic);
end add1;
architecture ss of add1 is
begin
s1<=(a1) xor (b1);
c1<=(a1) and (b1);
end;
```

8. Открыть новый VHDL-файл и записать в него **Программу 5.5** для 1-разрядного сумматора. Сохранить файл с названием **add11**, установить его старшим в иерархии и откомпилировать.
9. Пользуясь «Приложением 2», получить диаграммы при **интервалах импульса на входе a2 – 20нс, на входе b2 – 30нс, на входе c1 – 50нс.**

Программа 5.5.

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity add11 is
port(a2,b2,c1: in std_logic; c2,s2: out std_logic);
end add11;
architecture ss of add11 is
begin
s2<=((a2) xor (b2)) xor (c1);
c2<=((a2) and (b2)) or (((a2) xor (b2)) and (c1));
end;
```

10. Открыть новый VHDL-файл и записать в него **Программу 5.6** для 4-разрядного сумматора с сохранением результата в регистре, сохранением флага переноса в триггере, и выводом результата на индикаторы. Сохранить файл с названием **summ4**, установить его старшим в иерархии и откомпилировать. **Это файл верхнего уровня.**

Программа 5.6.

```
library ieee;
use ieee.std_logic_1164.all;
entity summ4 is
port(a,b: in std_logic_vector(3 downto 0);
clk: in std_logic;
output: out std_logic_vector(6 downto 0);
car: out std_logic);
end summ4;
architecture vv of summ4 is
component add1
port(a1,b1:in std_logic; c1,s1:out std_logic);
end component;
component add11
port(a2,b2,c1:in std_logic; c2,s2:out std_logic);
end component;
component rg4
port(d_i:in std_logic_vector(3 downto 0);
clk_i:in std_logic;
d_o:out std_logic_vector(3 downto 0));
end component;
component trig
port(d_i:in std_logic;
clk_i:in std_logic;
d_o:out std_logic);
end component;
component coder
port(symb:in std_logic_vector(3 downto 0);
ind: out std_logic_vector(6 downto 0));
end component;
```

```
signal c_in:std_logic_vector(2 downto 0);
signal c: std_logic;
signal s:std_logic_vector(3 downto 0);
signal out_rg4:std_logic_vector(3 downto 0);
begin
p0: add1
port map(a1=>a(0),b1=>b(0),c1=>c_in(0),s1=>s(0));
p1: add11
port map(c1=>c_in(0),a2=>a(1),b2=>b(1),c2=>c_in(1),s2=>s(1));
p2: add11
port map(c1=>c_in(1),a2=>a(2),b2=>b(2),c2=>c_in(2),s2=>s(2));
p3: add11
port map(c1=>c_in(2),a2=>a(3),b2=>b(3),c2=>c,s2=>s(3));
p4: reg
port map(clk_i=>clk,d_i=>s,d_o=>out_reg);
p5: trig
port map(clk_i=>clk,d_i=>c,d_o=>car);
p6: coder
port map(symb=>out_reg, ind=>output);
end;
```

11. Пользуясь «Приложением 3» произвести разводку выводов схемы для работы в макете таким образом, чтобы ввод **числа “a”** осуществлялся с тумблеров **SW9,SW8,SW7,SW6**(**SW9** – старший разряд), ввод **числа“b”** – с тумблеров **SW5,SW4,SW3,SW2**(**SW5** – старший разряд), ввод синхронизации - с кнопки **KEY(0)**. Вывод результата производить на **сегментный индикатор 0-й, вывод значения флага переноса на светодиод – LEDR 9. После компиляции файла планировщика еще раз откомпилируйте файл верхнего уровня!**
12. Пользуясь «Приложением 4» произвести программирование кристалла FPGA макета. На крайнем правом индикаторе должен высветиться «0». Проверить работоспособность схемы. Для этого набирать различные значения чисел **“a”** и **“b”** и нажимать крайнюю правую кнопку, имитируя подачу импульса синхронизации.

    Продемонстрировать работу преподавателю.
    Отчет по работе должен содержать тексты программ и диаграммы работы основных блоков устройства.