Лабораторная работа 8 (Преобразователь кода)

☑ Done ☐

Цель

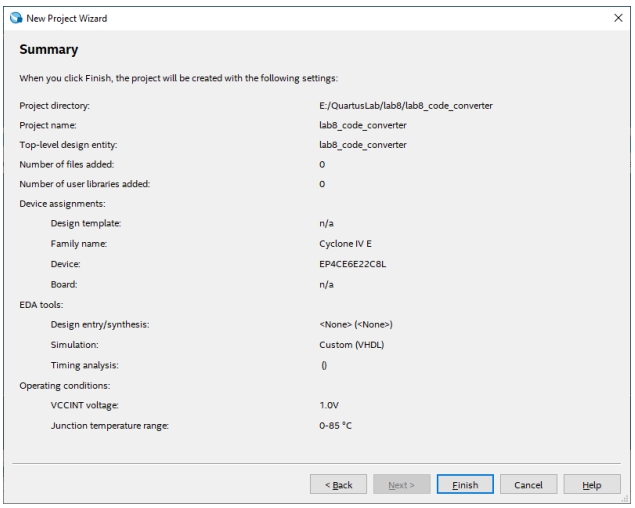
Ознакомиться с примерами преобразователей кодов, их схемами и работой. Написать на языках VHDL и SystemVerylog программу для заданного вариантом преобразователя кодов.

Задание

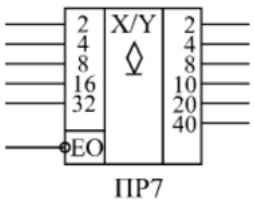
Реализовать на языке VHDL и SystemVerylog преобразователь кодов, предложенный вариантом.

Выполнение

1. Создаем пустой проект с такими же параметрами как и лабораторная №1



2. Выбираю свой вариант



ПР7

3. Создаем `Verilog HDL File`, `Block Diagram/Schematic File`, `VHDL File` и в разделе Verification/Debugging File выбираем `University Program VWF` и заполняем их кодом

```
module bcd_converter_Verilog (
    input wire Bin2, Bin4, Bin8, Bin16, Bin32, // Входные двоичные биты
    input wire EO,                             // Вход разрешения
    output reg BCD2, BCD4, BCD8, BCD10, BCD20, BCD40 // Выходные двоично-десятичные биты
);
```

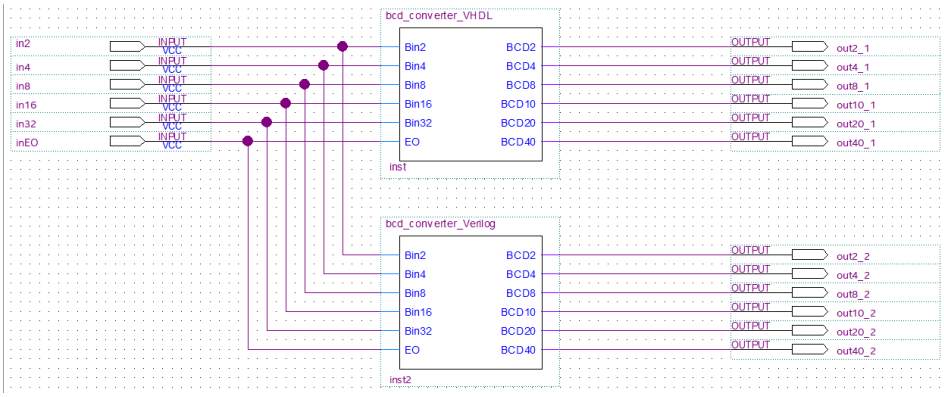
```
    always @(*) begin
        if (EO == 1) begin
            // Выходы неактивны при EO=1
            BCD2 = 1;
            BCD4 = 1;
            BCD8 = 1;
            BCD10 = 1;
            BCD20 = 1;
            BCD40 = 1;
        end else begin
            // Преобразование двоичного кода в BCD при EO=0
                case ({Bin32, Bin16, Bin8, Bin4, Bin2})
                     5'b00000: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b000000; // 0
                     5'b00001: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b000001; // 2
                     5'b00010: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b000010; // 4
                     5'b00011: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b000011; // 6
                     5'b00100: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b000100; // 8
                     5'b00101: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b001000; // 10
                     5'b00110: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b001001; // 12
                     5'b00111: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b001010; // 14
                     5'b01000: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b001011; // 16
                     5'b01001: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b001100; // 18
                     5'b01010: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b010000; // 20
                     5'b01011: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b010001; // 22
                     5'b01100: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b010010; // 24
                     5'b01101: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b010011; // 26
                     5'b01110: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b010100; // 28
                     5'b01111: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b011000; // 30
                     5'b10000: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b011001; // 32
                     5'b10001: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b011010; // 34
                     5'b10010: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b011011; // 36
                     5'b10011: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b011100; // 38
                     5'b10100: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b100000; // 40
                     5'b10101: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b100001; // 42
                     5'b10110: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b100010; // 44
                     5'b10111: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b100011; // 46
                     5'b11000: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b100100; // 48
                     5'b11001: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b101000; // 50
                     5'b11010: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b101001; // 52
                     5'b11011: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b101010; // 54
                     5'b11100: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b101011; // 56
                     5'b11101: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b101100; // 58
                     5'b11110: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b110000; // 60
                     5'b11111: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b110001; // 62
                     default:  {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b111111; // Ошибка
                endcase;
        end
    end
endmodule
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

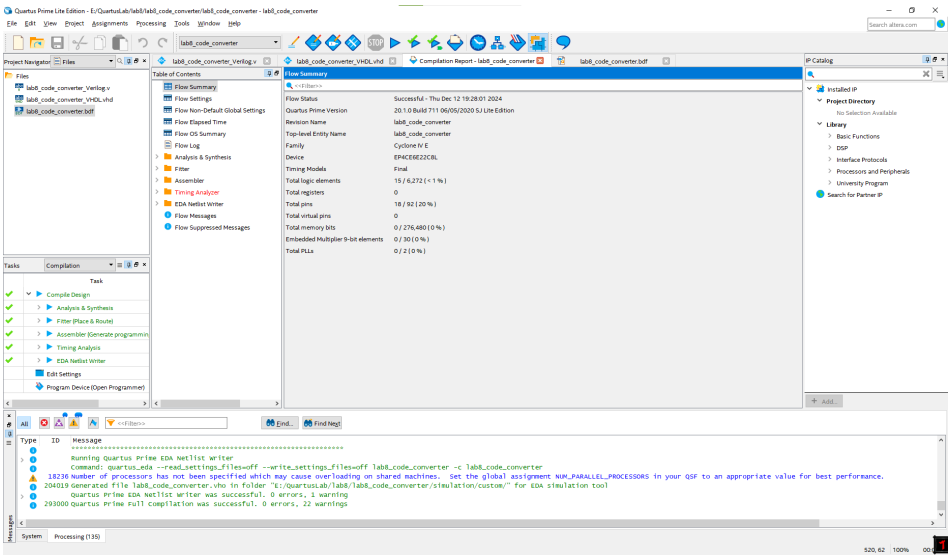
```
entity bcd_converter_VHDL is
    Port (
        Bin2, Bin4, Bin8, Bin16, Bin32 : in STD_LOGIC; -- Входные двоичные биты
        EO : in STD_LOGIC;                             -- Вход разрешения
        BCD2, BCD4, BCD8, BCD10, BCD20, BCD40 : out STD_LOGIC -- Выходные двоично-десятичные бит
    );
end bcd_converter_VHDL;

architecture Behavioral of bcd_converter_VHDL is
    signal binary_code : STD_LOGIC_VECTOR(4 downto 0);
    signal bcd_code : STD_LOGIC_VECTOR(5 downto 0);
begin
    -- Формируем входное двоичное число
    binary_code <= Bin32 & Bin16 & Bin8 & Bin4 & Bin2;

    process (binary_code, EO)
    begin
        if EO = '1' then
            -- Выходы неактивны при EO=1
            BCD2 <= '1';
            BCD4 <= '1';
            BCD8 <= '1';
            BCD10 <= '1';
            BCD20 <= '1';
            BCD40 <= '1';
        else
            -- Преобразование двоичного кода в BCD при EO=0
                case binary_code is
                        when "00000" => bcd_code <= "000000"; -- 0
                        when "00001" => bcd_code <= "000001"; -- 2
                        when "00010" => bcd_code <= "000010"; -- 4
                        when "00011" => bcd_code <= "000011"; -- 6
                        when "00100" => bcd_code <= "000100"; -- 8
                        when "00101" => bcd_code <= "001000"; -- 10
                        when "00110" => bcd_code <= "001001"; -- 12
                        when "00111" => bcd_code <= "001010"; -- 14
                        when "01000" => bcd_code <= "001011"; -- 16
                        when "01001" => bcd_code <= "001100"; -- 18
                        when "01010" => bcd_code <= "010000"; -- 20
                        when "01011" => bcd_code <= "010001"; -- 22
                        when "01100" => bcd_code <= "010010"; -- 24
                        when "01101" => bcd_code <= "010011"; -- 26
                        when "01110" => bcd_code <= "010100"; -- 28
                        when "01111" => bcd_code <= "011000"; -- 30
                        when "10000" => bcd_code <= "011001"; -- 32
                        when "10001" => bcd_code <= "011010"; -- 34
                        when "10010" => bcd_code <= "011011"; -- 36
                        when "10011" => bcd_code <= "011100"; -- 38
                        when "10100" => bcd_code <= "100000"; -- 40
                        when "10101" => bcd_code <= "100001"; -- 42
                        when "10110" => bcd_code <= "100010"; -- 44
                        when "10111" => bcd_code <= "100011"; -- 46
                        when "11000" => bcd_code <= "100100"; -- 48
                        when "11001" => bcd_code <= "101000"; -- 50
```

```
                    when "11010" => bcd_code <= "101001"; -- 52
                    when "11011" => bcd_code <= "101010"; -- 54
                    when "11100" => bcd_code <= "101011"; -- 56
                    when "11101" => bcd_code <= "101100"; -- 58
                    when "11110" => bcd_code <= "110000"; -- 60
                    when "11111" => bcd_code <= "110001"; -- 62
                    when others  => bcd_code <= "111111"; -- Ошибка
                    end case;
            -- Распределяем значения по выходным битам
            BCD2 <= bcd_code(5);
            BCD4 <= bcd_code(4);
            BCD8 <= bcd_code(3);
            BCD10 <= bcd_code(2);
            BCD20 <= bcd_code(1);
            BCD40 <= bcd_code(0);
        end if;
    end process;
end Behavioral;
```

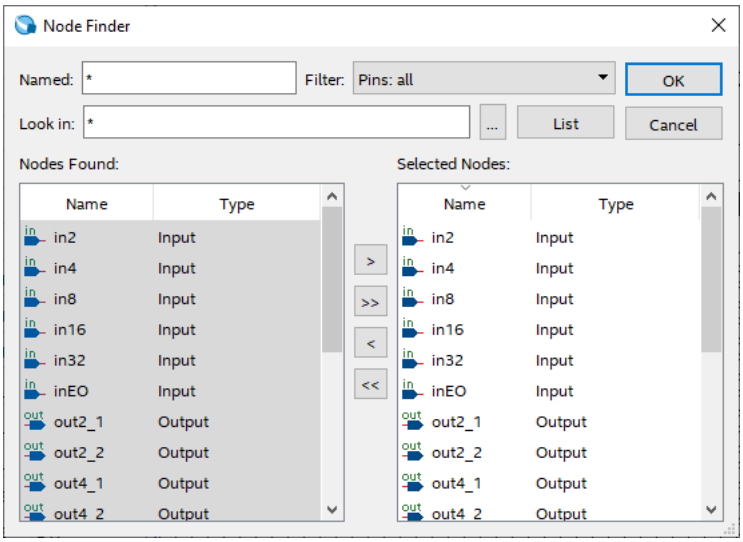
4. Компилируем их и добавляем на Block файл



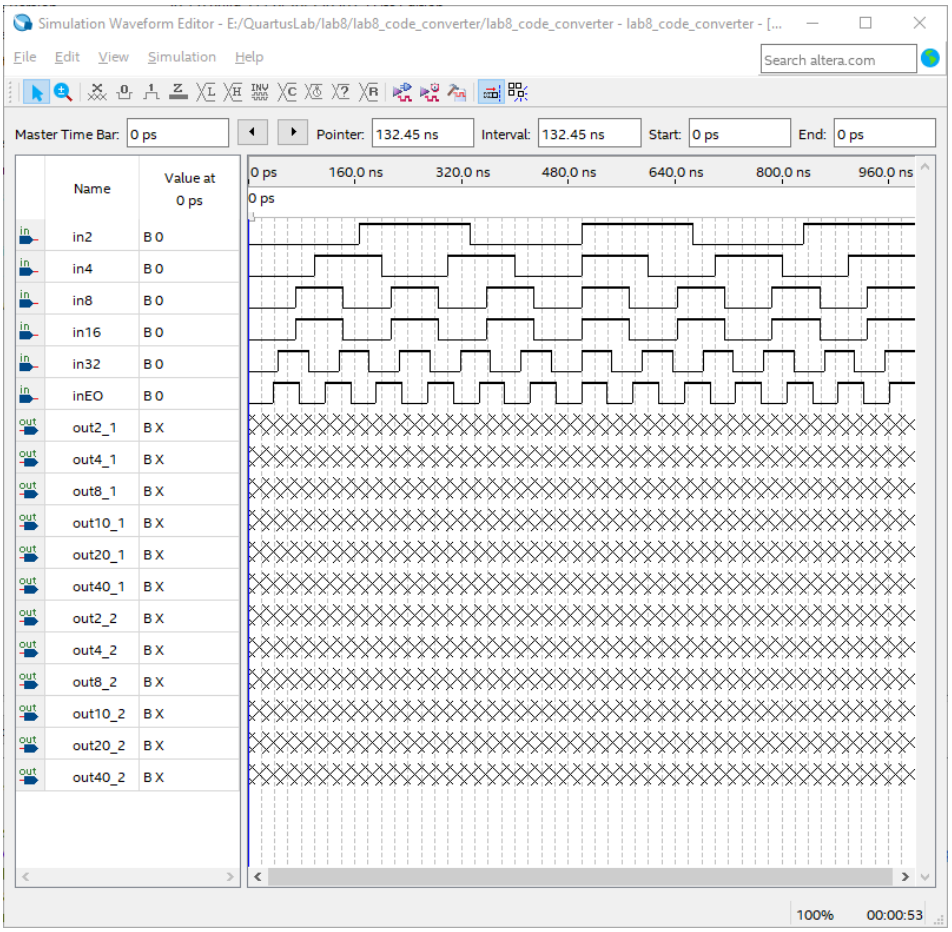
5. Отправляем нашу схему на верхний уровень и запускаем компиляцию проекта, дожидаемся успешного завершения.



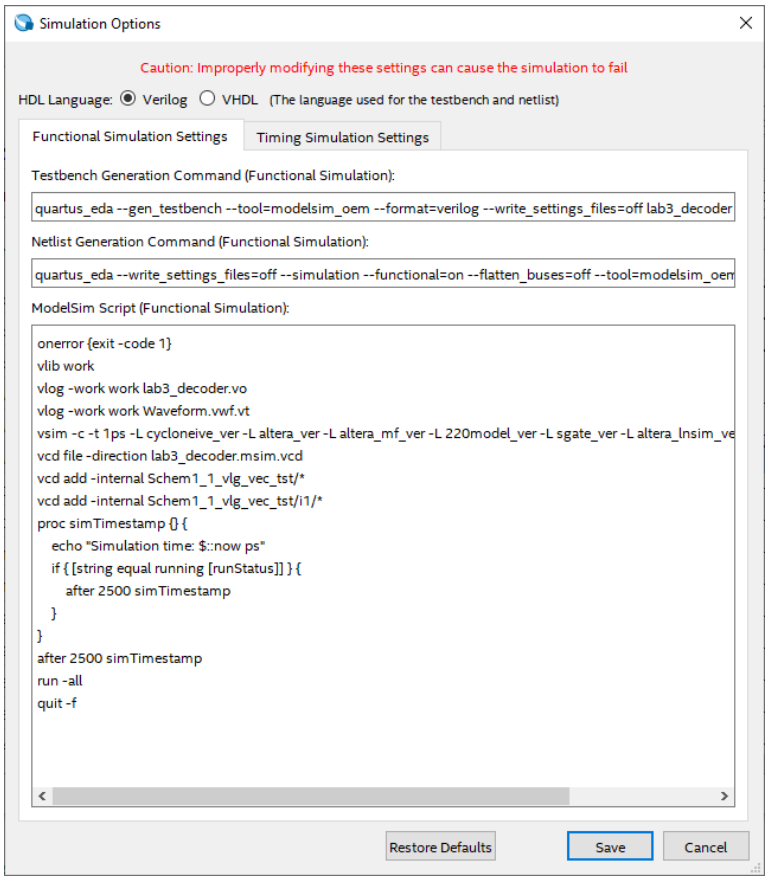
6. На странице добавления узлов в модуляцию ищем все наши узлы и добавляем их



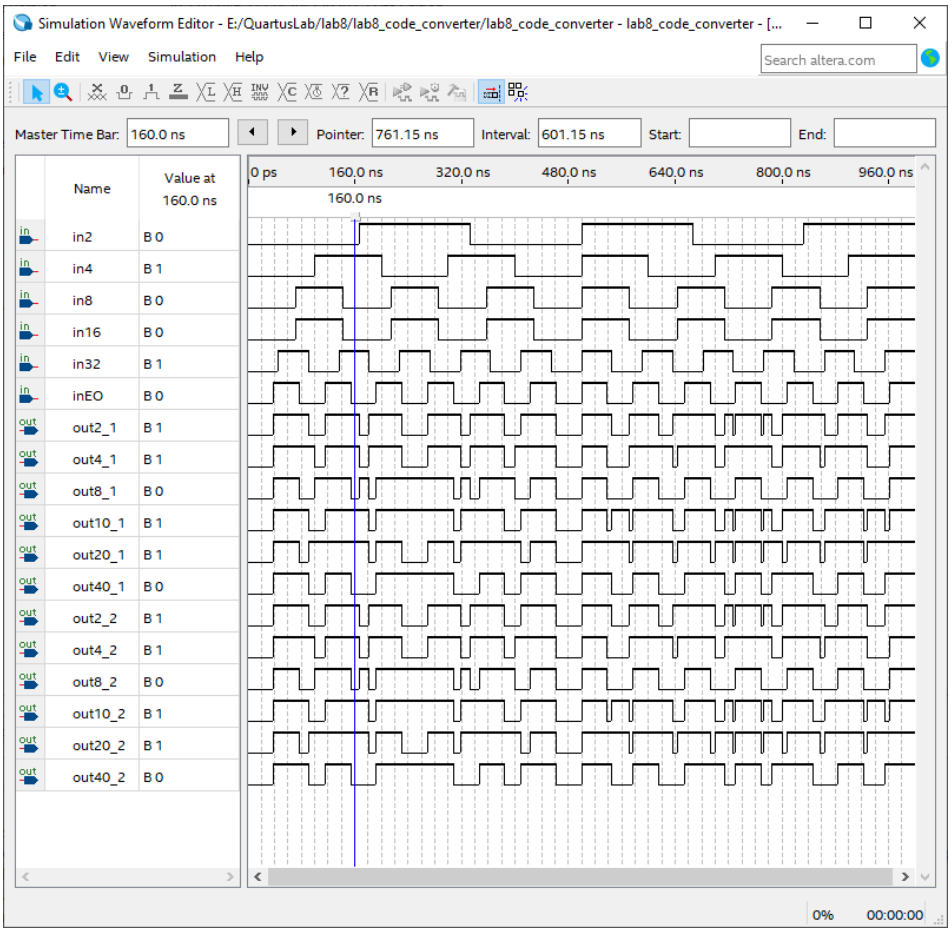
7. Для каждого узла выставляем разную частоту от 3 до 13 MHz



8. Убираем `-novopt` из параметров симуляции



9. Проверяем результат



Входы						Выходы	
-EO	32	16	8	4	2	40	20
1	X	X	X	X	X	1	1
0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0
0	0	0	0	1	1	0	0
0	0	0	1	0	0	0	0
0	0	0	1	0	1	0	0
0	0	0	1	1	0	0	0
0	0	0	1	1	1	0	0
0	0	1	0	0	0	0	0
0	0	1	0	0	1	0	0
0	0	1	0	1	0	0	1
0	0	1	0	1	1	0	1
0	0	1	1	0	0	0	1
0	0	1	1	0	1	0	1
0	0	1	1	1	0	0	1
0	0	1	1	1	1	0	1

Входы						Выходы	
0	1	0	0	0	0	0	1
0	1	0	0	0	1	0	1
0	1	0	0	1	0	0	1
0	1	0	0	1	1	0	1
0	1	0	1	0	0	1	0
0	1	0	1	0	1	1	0
0	1	0	1	1	0	1	0
0	1	0	1	1	1	1	0
0	1	1	0	0	0	1	0
0	1	1	0	0	1	1	0
0	1	1	0	1	0	1	0
0	1	1	0	1	1	1	0
0	1	1	1	0	0	1	0
0	1	1	1	0	1	1	0
0	1	1	1	1	0	1	1
0	1	1	1	1	1	1	1

Вывод

В ходе данной работы мы познакомились с построением преобразователя кодов, а также запрограммировали заданный преобразователь кодов на языках VHDL и SystemVerylog и проверили работу нашего кода с помощью составления схемы и запуска симуляции работы.