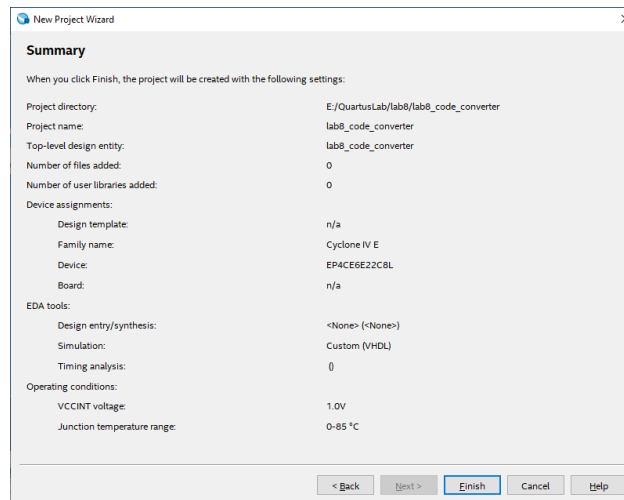


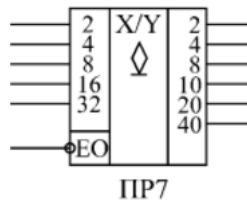
# Лабораторная работа 8 (Преобразователь кода)

☒ Done ☐

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



2. Выбираю свой вариант



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 E0=0
case ({Bin32, Bin16, Bin8, Bin4, Bin2})
    5'b000000: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b0000000; // 0
    5'b000001: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b0000001; // 2
    5'b000010: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b0000010; // 4
    5'b000011: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b0000011; // 6
    5'b000100: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b0000100; // 8
    5'b000101: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b0001000; // 10
    5'b000110: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b0001001; // 12
    5'b000111: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b0001010; // 14
    5'b001000: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b0001011; // 16
    5'b001001: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b0001100; // 18
    5'b001010: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b0010000; // 20
    5'b001011: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b0010001; // 22
    5'b001100: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b0010010; // 24
    5'b001101: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b0010011; // 26
    5'b001110: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b0010100; // 28
    5'b001111: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b0011000; // 30
    5'b010000: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b0011001; // 32
    5'b010001: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b0011010; // 34
    5'b010010: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b0011011; // 36
    5'b010011: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b0011100; // 38
    5'b010100: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b0100000; // 40
    5'b010101: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b0100001; // 42
    5'b010110: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b0100010; // 44
    5'b010111: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b0100011; // 46
    5'b011000: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b0100100; // 48
    5'b011001: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b0100100; // 50
    5'b011010: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b0100101; // 52
    5'b011011: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b0100101; // 54
    5'b011100: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b0100111; // 56
    5'b011101: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b0101100; // 58
    5'b011110: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b0110000; // 60
    5'b011111: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b0110001; // 62
    default: {BCD40, BCD20, BCD10, BCD8, BCD4, BCD2} = 6'b1111111; // 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
        E0 : 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);

```

```

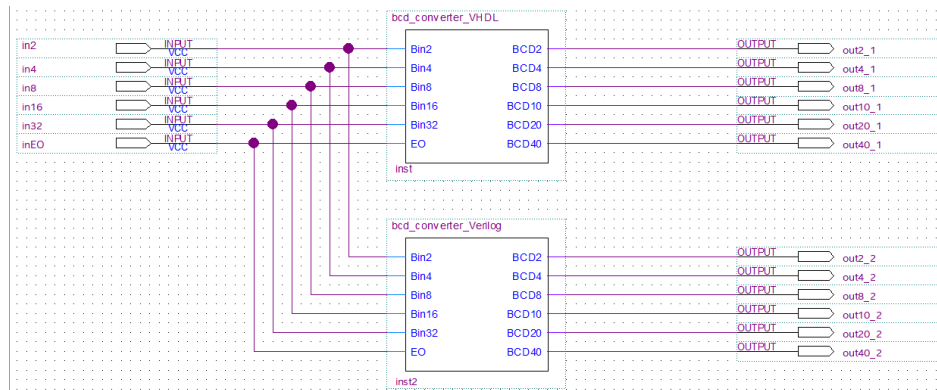
    signal bcd_code : STD_LOGIC_VECTOR(5 downto 0);
begin
    -- Формируем входное двоичное число
    binary_code <= Bin32 & Bin16 & Bin8 & Bin4 & Bin2;

    process (binary_code, E0)
    begin
        if E0 = '1' then
            -- Выходы неактивны при E0=1
            BCD2 <= '1';
            BCD4 <= '1';
            BCD8 <= '1';
            BCD10 <= '1';
            BCD20 <= '1';
            BCD40 <= '1';
        else
            -- Преобразование двоичного кода в BCD при E0=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;

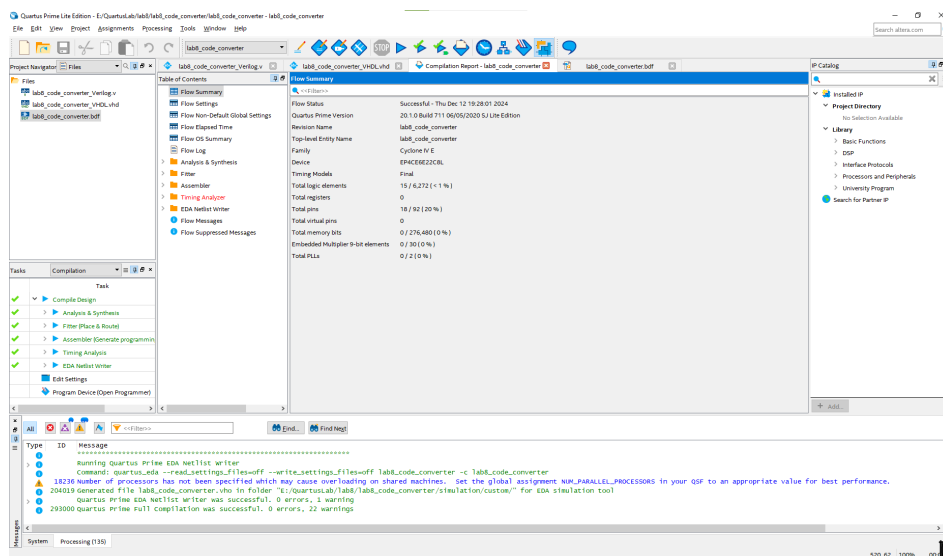
            -- Распределяем значения по выходным битам
            BCD2 <= bcd_code(5);
        end if;
    end process;
end begin;

```

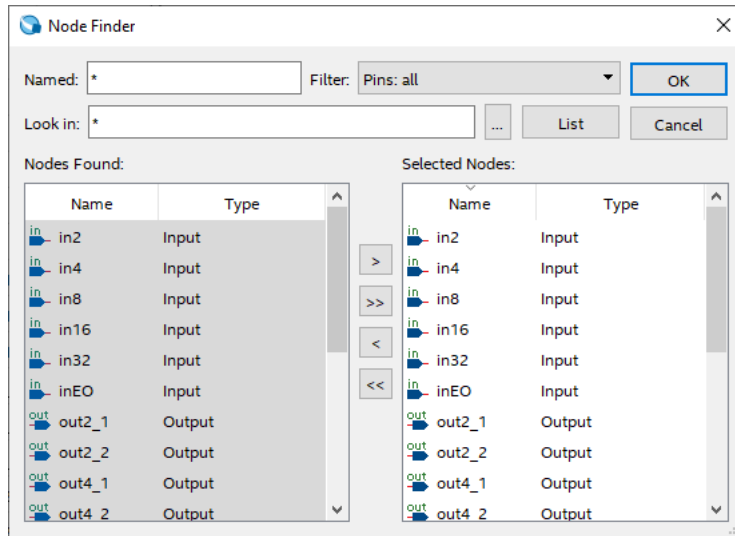
#### 4. Компилируем их и добавляем на Block файл



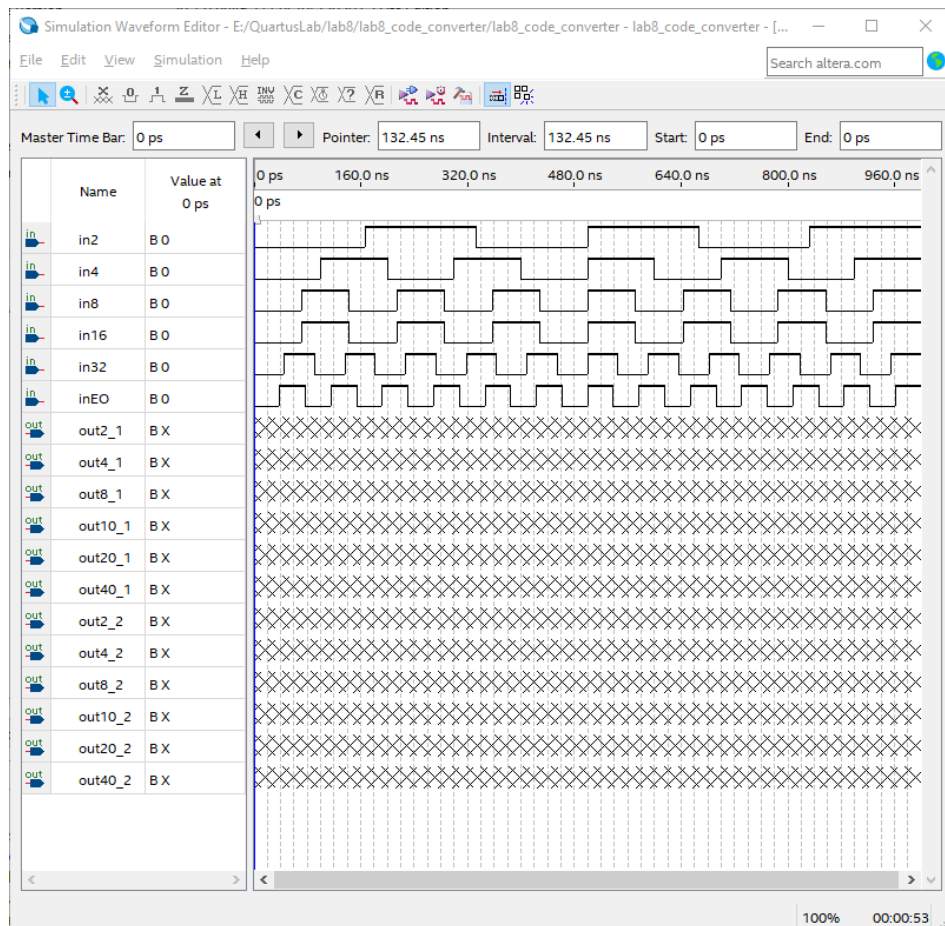
5. Отправляем нашу схему на верхний уровень и запускаем компиляцию проекта, ждем успешного завершения.



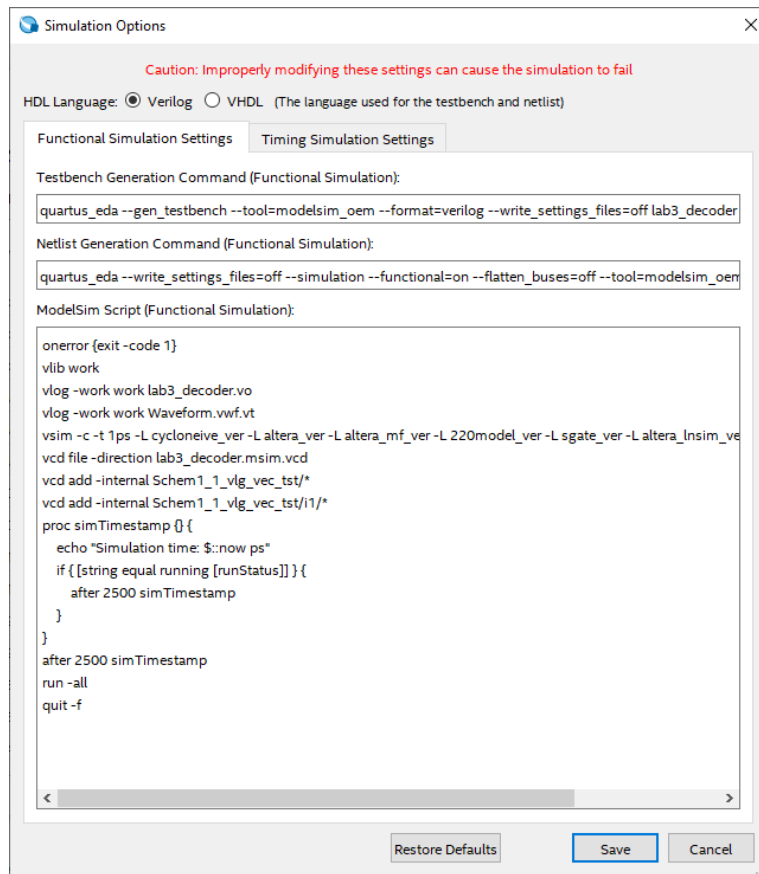
6. На странице добавления узлов в модуляцию ищем все наши узлы и добавляем их



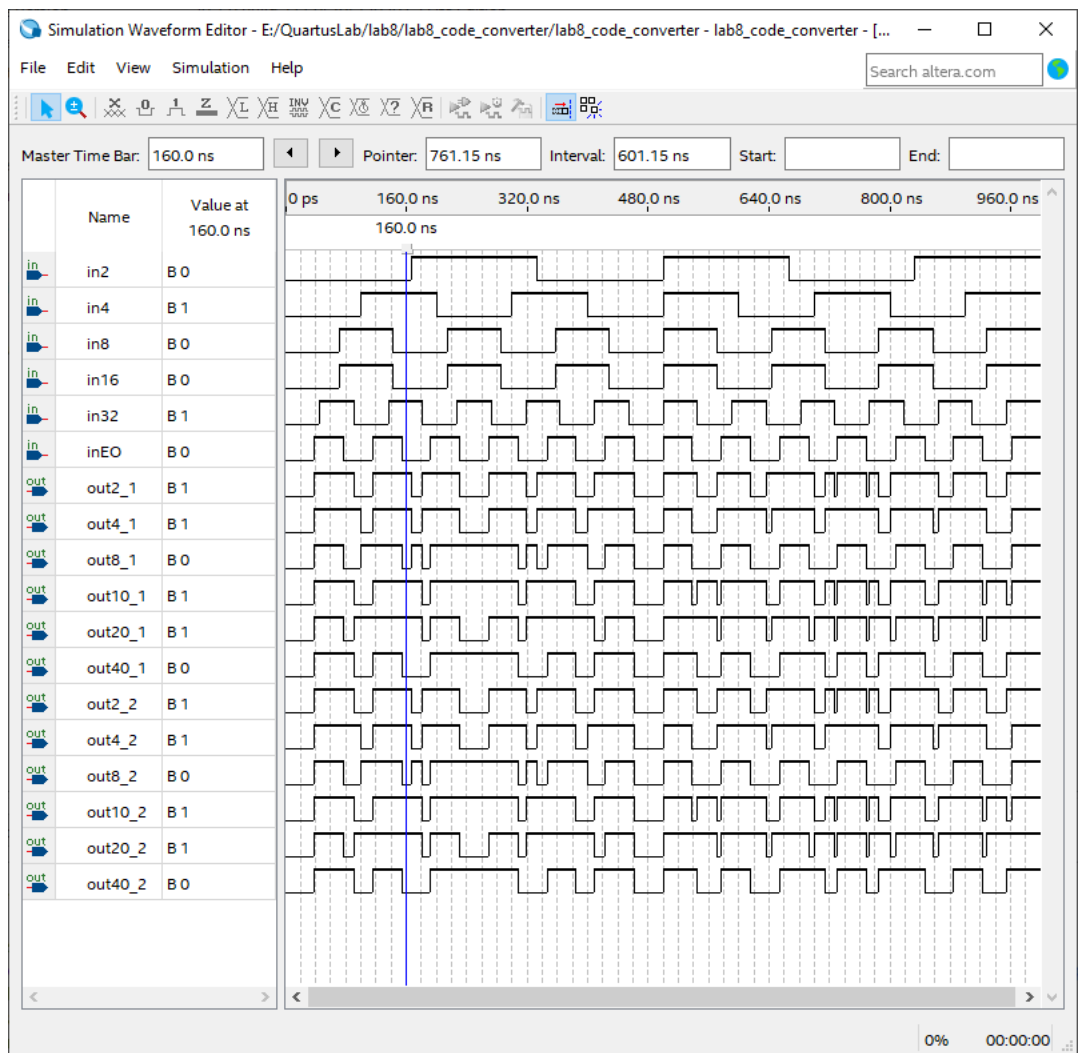
7. Для каждого узла выставляем разную частоту от 3 до 13 MHz



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	1	0	1
0	0	1	1	0	0	0	1

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