

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Безопасности жизнедеятельности

ОТЧЕТ
по лабораторной работе №3
по дисциплине «Проектирование проблемно ориентированных
вычислительных устройств»
Тема: Модульное и иерархическое проектирование

Студенты гр. 1308

Лепов А. В.

Мельник Д. А.

Преподаватель

Гречухин М. Н.

Санкт-Петербург

2024

Цель работы.

Разработать описание устройства в едином модуле с генератором тестового воздействия на базе синтаксиса языка VerilogHDL. Выполнить имплементацию проекта в целевую микросхему программируемой логики.

Задание на лабораторную работу

Часть 1. Проверка устройства средствами, встроенными в файл описания

1. Подготовить в рабочем каталоге файлы `compon.v`, `str_l3.v`, `tb_l3.v`, соответствующие листингам 3.1, 3.2, 3.3, запустить систему моделирования.

2. Создать новый проект и включить в него указанные файлы. Используя окна редактирования, изучить структуру и реализуемые функции программ в предлагаемых файлах.

3. Скорректировать тексты файлов `Str_l3.v` и `Tb_l3.v` таким образом, чтобы воспроизводились три логические функции в соответствии с индивидуальным заданием. При этом желательно, чтобы в программе тестирования одна из введенных таблиц истинности в разделе параметров была искажена по сравнению с заданной для демонстрации правильности работы оператора `tester`.

4. Выполнить компиляцию проектных файлов.

5. Запустить режим моделирования, назначив в качестве вершины проекта модуль `test`. Открыть окна `Objects`, `Processes`, `List`, `Wave`. Выполнить моделирование в течение 1 – 3 циклов изменения тестового воздействия в пошаговом режиме. Наблюдать и записать порядок инициализации процессов и последовательность изменений сигналов в окне `List`.

6. Выполнить моделирование узла в автоматическом режиме. Наблюдать временную диаграмму процесса. В случае обнаружения отклонения результатов от ожидаемых повторить пп. 3, 4 и 6.

Часть 2. Реализация проекта в ПЛИС

1. За основу берется программа, подготовленная по Части 1 практикума. Блок `initial` заменяется двоичным счетчиком, тактируемым внешним сигналом `clock`, а блок монитора удаляется. В качестве счетчика и дешифратора

необходимо использовать модули из библиотеки параметризованных модулей САПР Quartus II, подключение которых выполняется следующим образом.

1.1. Создать в САПР Quartus II проект, вершиной которого является файл test.

1.2. Найти в библиотеке параметризованных модулей (LPM) необходимые функции, поддерживаемые программным обеспечением Quartus II. Обычно они расположены в каталоге ...\\quartus\\libraries\\megafunctions.

Из соответствующего файла *.tdf необходимо извлечь описание портов модуля из раздела subdesign lpm_decode и определить дополнительные параметры настройки. Пример описания дешифратора:

```
(      data[LPM_WIDTH-1..0]          : INPUT = GND;
      enable                        : INPUT = VCC;
      clock, aclr                   : INPUT = GND;
      clken                         : INPUT = VCC;
      eq[LPM_DECODES-1..0]         : OUTPUT;)
```

Параметры настройки экземпляра элемента определяются декларацией defparam. Для необязательных при подключении конкретного экземпляра портов можно оставить значения по умолчанию.

1.3. Описать схему, используя библиотечные компоненты Quartus II, применив оператор вхождения компонента, определив цепи, к которым подключаются порты конкретного экземпляра. В описании подключений рекомендуется использовать сопоставление по имени, при этом неиспользуемые порты примут значения «по умолчанию», указанные в описании.

```
lpm_decode decoder( .data (x_in[2:0]), .eq (y[7:0]));
defparam decoder.lpm_width = 3;
defparam decoder.lpm_decodes = 8;
```

2. Выполнить в САПР Quartus II компиляцию, назначить контакты в соответствии с используемой учебной платой (цоколевка ПЛИС для учебной платы Terasic DE0 приведена в Приложении 4).

3. Осуществить загрузку проекта в учебную плату, наблюдать результаты работы с помощью осциллографа. При настройке осциллографа в режиме работы с внешней синхронизацией рекомендуется вывести на внешние контакты старший разряд счетчика и использовать его в качестве синхросигнала.

Логические функции (Вариант 4)

Z_0 : 10011100

Z_1 : 00111010

Z_2 : 10000010

Выполнение работы

Часть 0. Проверка корректности работы модуля decoder.

Перед началом работы необходимо удостовериться в корректности описания предлагаемого в методических указаниях модуля *decoder*. Для этого необходимо импортировать в проект файл *decode.v*, написать модуль *lab3_0.v*, который будет являться обёрткой для дешифратора: будет подавать на него входные воздействия и выводить его реакции.

Для тестирования составлен тестбенч *lab3_0_tb.v*. Ниже приведены листинги соответствующих модулей с комментариями.

decode.v:

```
//`timescale 1 ns / 1 ns
module decode(x_in, y_out);
    parameter delay=0; // задержка
    parameter n=4; // число входов
    parameter u=8; // число выходов
    input x_in; output y_out; // режимы портов
    wire [n-1:0] x_in;
    reg [u-1:0] y_out; // типы портов
    reg [n:0] i;
    always @ (x_in) // оператор выполняется
                    // при любом изменении x_in
    begin # delay;
        for (i=0; i<u; i=i+1)
            y_out[i] = x_in==i ? 1:0; // в нужный бит выходной шины пишется 1,
                                     // в остальные пишутся нули
        end
    endmodule
```

lab3_0.v:

```
//`timescale 1 ns / 1 ns
module lab3_0( x_in, y_out);
    input x_in; // входная шина номера выхода дешифратора
    output y_out; // выходная шина для дешифратора

    wire [3:0] x_in;
    wire [7:0] y_out; // внутренние связи модуля

    decode decode_inst ( // вставка дешифратора
        .x_in(x_in),      // соединения
        .y_out(y_out)
    );

Endmodule
```

lab3_0_tb.v:

```
// `timescale 1 ns / 1 ns
module lab3_0_tb;
    //Ports
    reg [3:0] x_in; // шина ввода в дешифратор
    wire [7:0] y_out; // шина вывода дешифратора
    reg [3:0] takt; // переменная счётчик для задания тестовых воздействий

    lab3_0 lab3_0_inst ( // вставка модуля-обёртки
        .x_in(x_in),
        .y_out(y_out)
    );

    initial begin
        x_in = 3'bzzz; // инициализация входной шины дешифратора в нулевой момент
        for (takt = 0; takt != 8; takt = takt+1) // перебор входных воздействий
            begin
                #5 x_in = takt; // присвоение входного значения с задержкой 5
            end
        end
    end
endmodule
```

Ниже приведены результаты моделирования дешифратора:

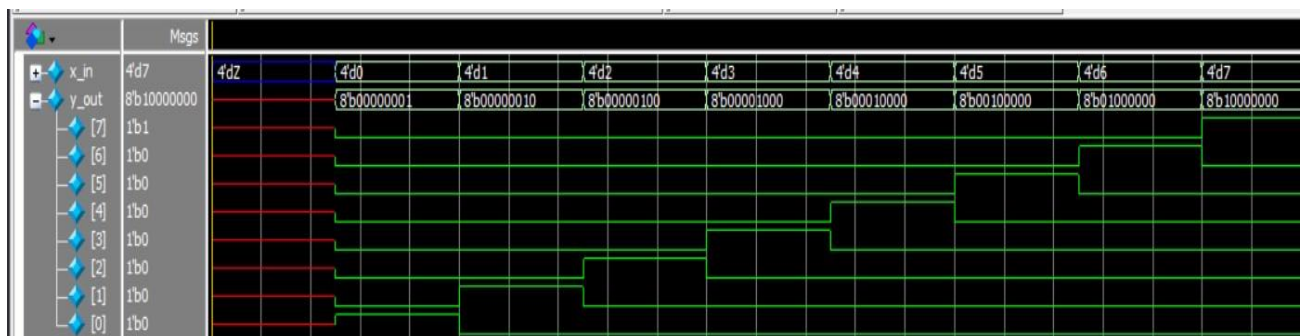


Рисунок 1. Временная диаграмма моделирования дешифратора

Как видно по временной диаграмме, представленный выше дешифратор функционирует корректно: наблюдается «лесенка» из сигналов выходной шины дешифратора при последовательном переборе значений входной шины от 0 до 7.

Часть 1. Реализация логической функции через модуль дешифратора (*decode.v*) и встроенного в Verilog модуля ИЛИ (*or*).

Далее приведено описание комплекса модулей, в которых реализуются три логические функции:

Z_0 : 10011100

Z_1 : 00111010

Z_2 : 10000010

На основе схемы с дешифратором, подобной приведённой ниже:

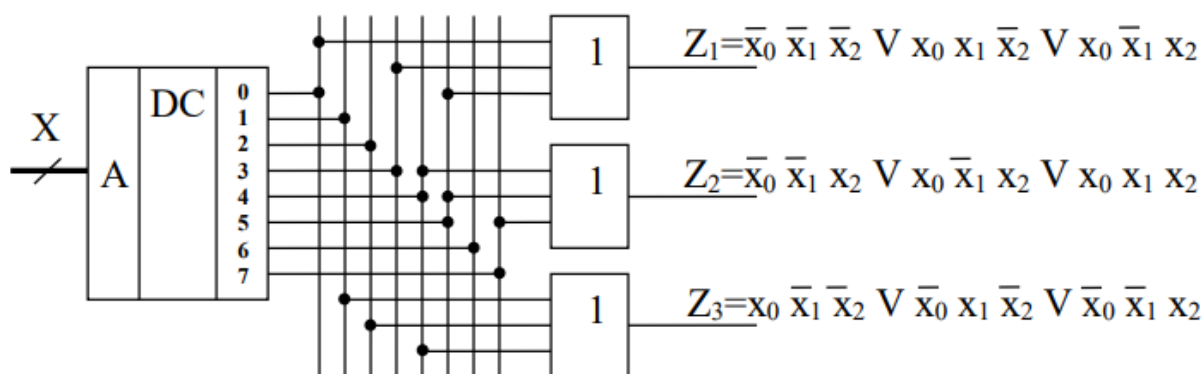


Рисунок 2. Пример реализации трёх логических функций из метод. указаний

Комплекс модулей состоит из ранее описанного модуля дешифратора, основного модуля с описанием логических функций *lab3_1.v* и файл-тестер *lab3_1_tb.v* для проверки корректности описания модуля трёх лог. функций.

Рассмотрим функции, которые необходимо реализовать:

| A_{10} | $A_2 (x_2 x_1 x_0)$ | z_0 | z_1 | z_2 |
|----------|---------------------|-------|-------|-------|
| 0 | 000 | 1 | 0 | 1 |
| 1 | 001 | 0 | 0 | 0 |
| 2 | 010 | 0 | 1 | 0 |
| 3 | 011 | 1 | 1 | 0 |
| 4 | 100 | 1 | 1 | 0 |
| 5 | 101 | 1 | 0 | 0 |
| 6 | 110 | 0 | 1 | 1 |
| 7 | 111 | 0 | 0 | 0 |

Рассмотрев приведённую выше таблицу можно сделать вывод, при каких входных последовательностях функция истинна. Зная набор эти истинных последовательностей, можно выполнить их сборку по ИЛИ. Входные последовательности это и есть номера выходов дешифратора.

Листинг описания модуля lab3_1.v:

```
module lab3_1( x_in, z_0, z_1, z_2);
    input x_in; // входные воздействия на дешифратор
    output z_0,z_1,z_2 ; // выходные значения лог. функций
    wire [3:0] x_in; // шина на 4 бита
    wire z_0,z_1,z_2; // одиночные сигналы
    wire [7:0] y; // внутренние связи модуля
                    // для взаимодействия с выходами дешифратора

    decode # (.n(4), .delay(0)) // вставка модуля дешифратора
        decode_inst(.x_in(x_in), .y_out(y));

    // вставка модулей or
    or or4(z_0, y[0], y[3], y[4], y[5]); // z0 = 10011100

    or or6(z_1, y[2], y[3], y[4], y[6]); // z1 = 00111010

    or or1(z_2, y[0], y[6]); // z2 = 10000010
endmodule
```


Листинг описания модуля-тестера lab3_1_tb.v:

```
module lab3_1_tb;
    parameter delay= 0;

    // образцовые логические функции
    parameter tt4 = 8'b00111001;
    parameter tt6 = 8'b01011100;
    // parameter tt1 = 8'b01000001; // корректная функция
    parameter tt1 = 8'b11111111; //некорректная функция

    reg [3:0] x_in; // переменная для входных воздействий
    wire [7:0] y_out; // внутренняя переменная для состояния дешифратора
    wire z_0, z_1, z_2; // выходы логических функций
    reg [3:0] takt; // переменная счётчик для перебора входных воздействий

    reg chech_point; // точка контроля функции тестером
    reg error_0, error_1, error_2; // сигналы тестера об ошибках
    wire vt_0, vt_1, vt_2; // образцовые значения функций

    lab3_1 lab3_1_inst ( // вставка основного модуля
        .x_in(x_in),
        .z_0(z_0),
        .z_1(z_1),
        .z_2(z_2)
    );

    initial
    begin // формирование тестовых последовательностей
        chech_point=0; // инициализация точки контроля
        for (takt = 0; takt <= 7; takt = takt + 1) // перебор входных воздействий
            begin #10;
                x_in=takt; // установка тестового воздействия
                chech_point= # delay 1; // установка точки контроля в 1
                #3; // на 3 ед. модельного времени
                chech_point=0; // сброс точки контроля
            end
        end

        assign vt_0 = tt4[x_in]; // присвоение образцовых значений функций
        assign vt_1 = tt6[x_in];
        assign vt_2 = tt1[x_in];

        always @ (posedge chech_point) // на передний фронт точки контроля
        begin // проверки тестером
            error_0 = z_0 == vt_0 ? 0:1;
            error_1 = z_1 == vt_1 ? 0:1;
            error_2 = z_2 == vt_2 ? 0:1;
        end
    end

endmodule
```

| Project - T:/MS/Buren2/lab3_0/Lab3_0 | | |
|--------------------------------------|--------|---------|
| Name | Status | Type |
| decode.v | ✓ | Verilog |
| lab3_1_tb.v | ✓ | Verilog |
| lab3_1.v | ✓ | Verilog |

Рисунок 3. Компиляция проекта

Ниже приведены результаты моделирования *lab3_1.v*:

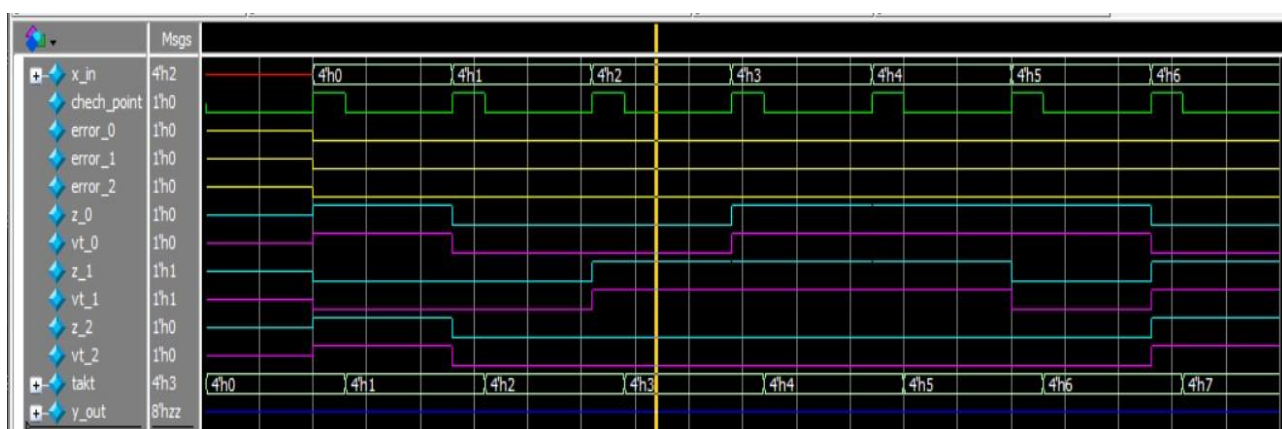


Рисунок 4. Работа тестера при задании корректных образцов функций

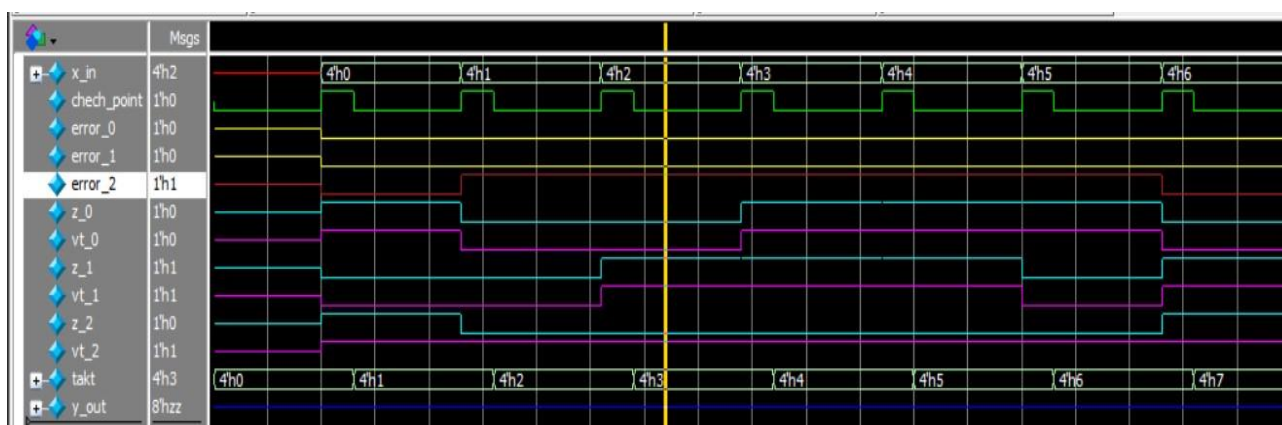


Рисунок 5. Работа тестера при ошибке в образце функции *z_2*

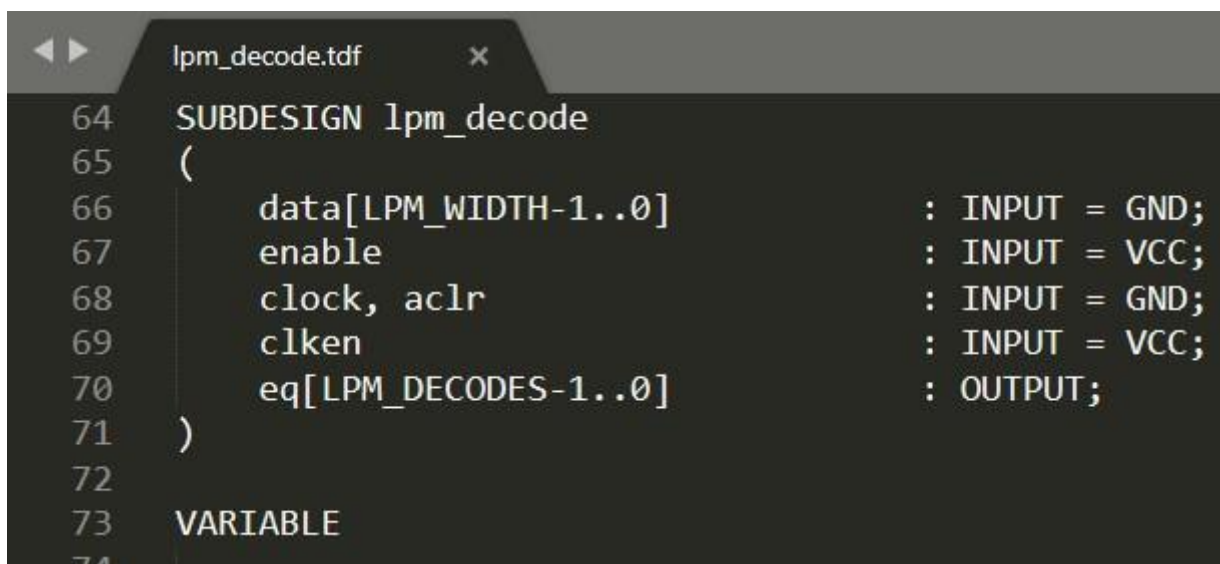
| ns | | /lab3_1_tb/x_in | /lab3_1_tb/z_2 | /lab3_1_tb/vt_1 | /lab3_1_tb/error_0 |
|-------|----|------------------------|-------------------------------|--------------------|--------------------|
| delta | | | /lab3_1_tb/z_1 | /lab3_1_tb/vt_0 | |
| | | /lab3_1_tb/chech_point | /lab3_1_tb/z_0 | lab3_1_tb/error_2 | |
| | | | /lab3_1_tb/vt_2 | /lab3_1_tb/error_1 | |
| 0 | +0 | 4'hx 1'h0 | 1'hx 1'hx 1'hx 1'hx 1'hx 1'hx | 1'hx 1'hx 1'hx | 1'hx 1'hx 1'hx |
| 10 | +0 | 4'h0 1'h0 | 1'hx 1'hx 1'hx 1'hx 1'hx 1'hx | 1'hx 1'hx 1'hx | 1'hx 1'hx 1'hx |
| 10 | +1 | 4'h0 1'h0 | 1'hx 1'hx 1'hx 1'h1 1'h0 1'h1 | 1'hx 1'hx 1'hx | 1'hx 1'hx 1'hx |
| 10 | +2 | 4'h0 1'h1 | 1'hx 1'hx 1'hx 1'h1 1'h0 1'h1 | 1'hx 1'hx 1'hx | 1'hx 1'hx 1'hx |
| 10 | +3 | 4'h0 1'h1 | 1'h1 1'h0 1'h1 1'h1 1'h0 1'h1 | 1'h0 1'h0 1'h0 | 1'h0 1'h0 1'h0 |
| 13 | +0 | 4'h0 1'h0 | 1'h1 1'h0 1'h1 1'h1 1'h0 1'h1 | 1'h0 1'h0 1'h0 | 1'h0 1'h0 1'h0 |
| 23 | +0 | 4'h1 1'h0 | 1'h1 1'h0 1'h1 1'h1 1'h0 1'h1 | 1'h0 1'h0 1'h0 | 1'h0 1'h0 1'h0 |
| 23 | +1 | 4'h1 1'h0 | 1'h1 1'h0 1'h1 1'h1 1'h0 1'h0 | 1'h0 1'h0 1'h0 | 1'h0 1'h0 1'h0 |
| 23 | +2 | 4'h1 1'h1 | 1'h1 1'h0 1'h1 1'h1 1'h0 1'h0 | 1'h0 1'h0 1'h0 | 1'h0 1'h0 1'h0 |
| 23 | +3 | 4'h1 1'h1 | 1'h0 1'h0 1'h0 1'h1 1'h0 1'h0 | 1'h1 1'h0 1'h0 | 1'h1 1'h0 1'h0 |
| 26 | +0 | 4'h1 1'h0 | 1'h0 1'h0 1'h0 1'h1 1'h0 1'h0 | 1'h1 1'h0 1'h0 | 1'h1 1'h0 1'h0 |
| 36 | +0 | 4'h2 1'h0 | 1'h0 1'h0 1'h0 1'h1 1'h1 1'h0 | 1'h1 1'h0 1'h0 | 1'h1 1'h0 1'h0 |
| 36 | +1 | 4'h2 1'h0 | 1'h0 1'h0 1'h0 1'h1 1'h1 1'h0 | 1'h1 1'h0 1'h0 | 1'h1 1'h0 1'h0 |
| 36 | +2 | 4'h2 1'h1 | 1'h0 1'h0 1'h0 1'h1 1'h1 1'h0 | 1'h1 1'h0 1'h0 | 1'h1 1'h0 1'h0 |
| 36 | +3 | 4'h2 1'h1 | 1'h0 1'h1 1'h0 1'h1 1'h1 1'h0 | 1'h1 1'h0 1'h0 | 1'h1 1'h0 1'h0 |
| 39 | +0 | 4'h2 1'h0 | 1'h0 1'h1 1'h0 1'h1 1'h1 1'h0 | 1'h1 1'h0 1'h0 | 1'h1 1'h0 1'h0 |
| 49 | +0 | 4'h3 1'h0 | 1'h0 1'h1 1'h0 1'h1 1'h1 1'h0 | 1'h1 1'h0 1'h0 | 1'h1 1'h0 1'h0 |
| 49 | +1 | 4'h3 1'h0 | 1'h0 1'h1 1'h0 1'h1 1'h1 1'h1 | 1'h1 1'h0 1'h0 | 1'h1 1'h0 1'h0 |
| 49 | +2 | 4'h3 1'h1 | 1'h0 1'h1 1'h0 1'h1 1'h1 1'h1 | 1'h1 1'h0 1'h0 | 1'h1 1'h0 1'h0 |
| 49 | +3 | 4'h3 1'h1 | 1'h0 1'h1 1'h1 1'h1 1'h1 1'h1 | 1'h1 1'h0 1'h0 | 1'h1 1'h0 1'h0 |
| 52 | +0 | 4'h3 1'h0 | 1'h0 1'h1 1'h1 1'h1 1'h1 1'h1 | 1'h1 1'h0 1'h0 | 1'h1 1'h0 1'h0 |
| 62 | +0 | 4'h4 1'h0 | 1'h0 1'h1 1'h1 1'h1 1'h1 1'h1 | 1'h1 1'h0 1'h0 | 1'h1 1'h0 1'h0 |
| 62 | +2 | 4'h4 1'h1 | 1'h0 1'h1 1'h1 1'h1 1'h1 1'h1 | 1'h1 1'h0 1'h0 | 1'h1 1'h0 1'h0 |
| 62 | +3 | 4'h4 1'h1 | 1'h0 1'h1 1'h1 1'h1 1'h1 1'h1 | 1'h1 1'h0 1'h0 | 1'h1 1'h0 1'h0 |
| 65 | +0 | 4'h4 1'h0 | 1'h0 1'h1 1'h1 1'h1 1'h1 1'h1 | 1'h1 1'h0 1'h0 | 1'h1 1'h0 1'h0 |
| 75 | +0 | 4'h5 1'h0 | 1'h0 1'h1 1'h1 1'h1 1'h1 1'h1 | 1'h1 1'h0 1'h0 | 1'h1 1'h0 1'h0 |
| 75 | +1 | 4'h5 1'h0 | 1'h0 1'h1 1'h1 1'h1 1'h0 1'h1 | 1'h1 1'h0 1'h0 | 1'h1 1'h0 1'h0 |
| 75 | +2 | 4'h5 1'h1 | 1'h0 1'h1 1'h1 1'h1 1'h0 1'h1 | 1'h1 1'h0 1'h0 | 1'h1 1'h0 1'h0 |
| 75 | +3 | 4'h5 1'h1 | 1'h0 1'h0 1'h1 1'h1 1'h0 1'h1 | 1'h1 1'h0 1'h0 | 1'h1 1'h0 1'h0 |
| 78 | +0 | 4'h5 1'h0 | 1'h0 1'h0 1'h1 1'h1 1'h0 1'h1 | 1'h1 1'h0 1'h0 | 1'h1 1'h0 1'h0 |
| 88 | +0 | 4'h6 1'h0 | 1'h0 1'h0 1'h1 1'h1 1'h0 1'h1 | 1'h1 1'h0 1'h0 | 1'h1 1'h0 1'h0 |
| 88 | +1 | 4'h6 1'h0 | 1'h0 1'h0 1'h1 1'h1 1'h1 1'h0 | 1'h1 1'h0 1'h0 | 1'h1 1'h0 1'h0 |
| 88 | +2 | 4'h6 1'h1 | 1'h0 1'h0 1'h1 1'h1 1'h1 1'h0 | 1'h1 1'h0 1'h0 | 1'h1 1'h0 1'h0 |
| 88 | +3 | 4'h6 1'h1 | 1'h1 1'h1 1'h0 1'h1 1'h1 1'h0 | 1'h0 1'h0 1'h0 | 1'h0 1'h0 1'h0 |
| 91 | +0 | 4'h6 1'h0 | 1'h1 1'h1 1'h0 1'h1 1'h1 1'h0 | 1'h0 1'h0 1'h0 | 1'h0 1'h0 1'h0 |

Рисунок 6. Окно list при пошаговом моделировании

Часть 2. Макетирование комплекса на ПЛИС

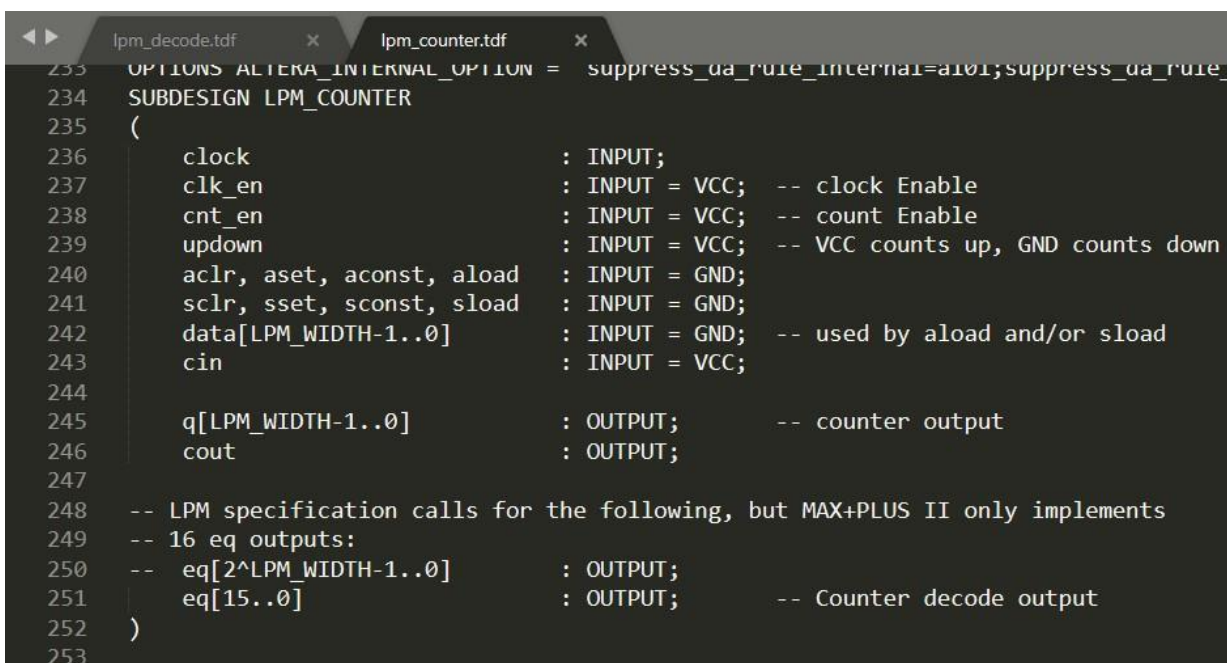
Для проведения макетирования необходимо внести изменения в код описания модуля *lab3_1.v*: заменить самописный модуль дешифратора на библиотечный, ввести тактовый сигнал *clock* и уменьшить его частоту с помощью счётчика.

Для указанных выше изменений необходимо найти описание модулей *lpm_decode* и *lpm_counter* в библиотеке IP блоков САПР Quartus II (*\quartus\libraries\megafunctions\???\.tdf*):



```
64 SUBDESIGN lpm_decode
65 (
66     data[LPM_WIDTH-1..0]      : INPUT = GND;
67     enable                    : INPUT = VCC;
68     clock, aclr               : INPUT = GND;
69     clken                     : INPUT = VCC;
70     eq[LPM_DECODES-1..0]     : OUTPUT;
71 )
72
73 VARIABLE
```

Рисунок 7. Описание блока *lpm_decode*



```
233 OPTIONS ALTERA_INTERNAL_OPTION = suppress_da_rule_internal=a101;suppress_da_rule_
234 SUBDESIGN LPM_COUNTER
235 (
236     clock                : INPUT;
237     clk_en               : INPUT = VCC; -- clock Enable
238     cnt_en               : INPUT = VCC; -- count Enable
239     updown               : INPUT = VCC; -- VCC counts up, GND counts down
240     aclr, aset, aconst, aload : INPUT = GND;
241     sclr, sset, sconst, sload : INPUT = GND;
242     data[LPM_WIDTH-1..0]   : INPUT = GND; -- used by aload and/or sload
243     cin                  : INPUT = VCC;
244
245     q[LPM_WIDTH-1..0]      : OUTPUT; -- counter output
246     cout                  : OUTPUT;
247
248     -- LPM specification calls for the following, but MAX+PLUS II only implements
249     -- 16 eq outputs:
250     -- eq[2^LPM_WIDTH-1..0] : OUTPUT;
251     eq[15..0]             : OUTPUT; -- Counter decode output
252 )
253
```

Рисунок 8. Описание блока *lpm_counter*

Листинг кода описания модуля lab3_2.v для синтеза на ПЛИС:

```
module Lab3(clock, z_0, z_1, z_2);
    input wire clock; // высокочастотный тактовый сигнал
    output wire z_0, z_1, z_2; // выходы функций
    wire [3:0] x_in; // входные воздействия на дешифратор
    wire [7:0] y; // связи с выходами дешифратора
    reg [24:0] clock2; // низкочастотный тактовый сигнал

    // формирование низкочастотного тактового сигнала
    lpm_counter clk1 (
        .clock(clock), // входной тактовый сигнал
        .cnt_en(1'b1), // включение счетчика
        .sclr(1'b0), // сброс счетчика (по умолчанию неактивен)
        .q(clock2) // счётчик для низкочастотного тактового сигнала
    );
    defparam clk1.lpm_width = 25; // установка ширины счётчика q в 25
    // формирование входных воздействий на дешифратор (через переполнение)
    lpm_counter clk2 (
        .clock(clock2[24]), // низкочастотный тактовый сигнал
        .cnt_en(1'b1), // включение счётчика
        .sclr(1'b0), // сброс счётчика (по умолчанию неактивен)
        .q(x_in) // тестовые воздействия (инкрементация)
    );
    defparam clk2.lpm_width = 4; // ширина второго счётчика q в 4

    lpm_decode decoder ( // библиотечный дешифратор
        .data(x_in),
        .eq(y)
    );

    defparam decoder.lpm_width = 3; // ширина входной шины 3 (x2x1x0)
    defparam decoder.lpm_decodes = 8; // ширина выходной шины 8 (0-7)

    or or4(z_0, y[0], y[3], y[4], y[5]); // сборки по ИЛИ
    or or6(z_1, y[2], y[3], y[4], y[6]);
    or or1(z_2, y[0], y[6]);
endmodule
```

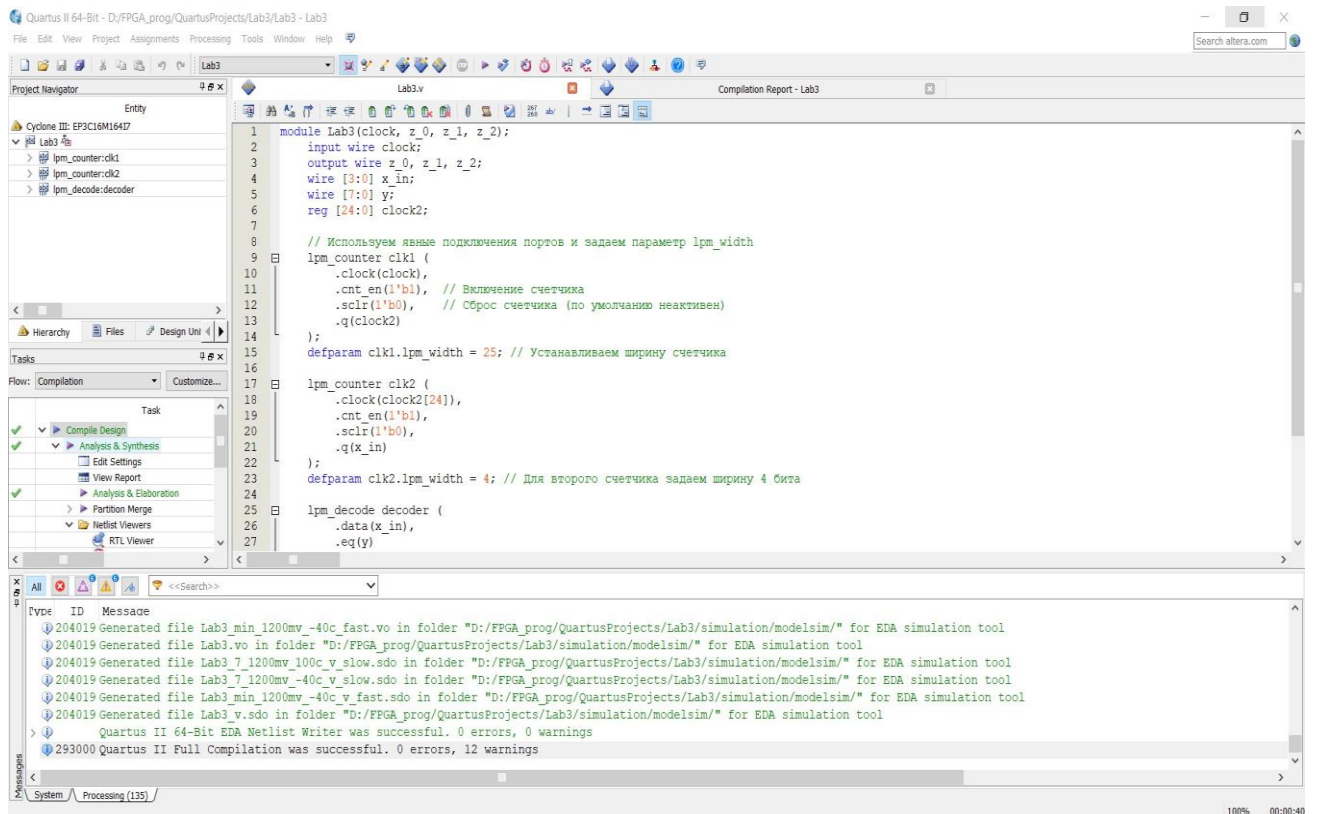


Рисунок 9. Удачная компиляция проекта Quartus II

Вывод

В ходе выполнения работы был изучен процесс разработки устройств на основе модульной архитектуры на языке Verilog. Был рассмотрен метод тестирования модулей средствами специально составленного модуля тестеров.

Были рассмотрены процесс интеграции в модуль библиотечных модулей интеллектуальной собственности САПР Quartus II при реализации макетирования модуля на ПЛИС.