МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА - Российский технологический университет»

РТУ МИРЭА

Институт Информационных Технологий
Кафедра Вычислительной Техники (ВТ)

ОТЧЁТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 3

«Основы верификации»

по дисциплине

«Схемотехника устройств компьютерных систем»

Выполнил студент группы ИВБО-08-22 — Стецюк В.В.

Принял ассистент кафедры ВТ — Дуксин Н.А.

Практическая работа выполнена «__»_____2024 г.

«Зачтено» «__»_____2024 г.

Москва 2024

АННОТАЦИЯ

Данная работа включает в себя 10 рисунков и 12 листингов. Количество страниц в работе — 28.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В данной практической работе изучаются основные подходы и инструменты для верификации проектов [1-2]. Разрабатываемое устройство представляет из себя конечный автомат устройства, разработанного в прошлых практические, с подключённым набором устройств для ввода/вывода [3]. Для каждого из устройств, входящих в конечное устройство, будет разработано верификационное окружение для проведения тестов и проведены соответствующие тесты. Для конечного устройства так же будет разработано верификационное окружение, а также будет произведена верификация с использованием VIO.

1 СОЗДАНИЕ НЕОБХОДИМЫХ МОДУЛЕЙ

1.1 Создание модуля конечного автомата

Название модуля – «fsm». Модуль обладает входами: «clk» – синхросигнал, «R_I» – сигнал о готовности входных данных, «reset» – сброс, «dataIn» – шестнадцатибитная шина входных данных; выходами: «dataOut» – шестнадцатибитная шина выходных данных, «R_O» – сигнал о готовности выходных данных, «current_state» – трёхбитная шина текущего состояния. У модуль есть регистры «REG_A», «REG_B», «REG_C», «REG_D», в которых сохраняются входные данные, регистр «REG_RES», который хранит результат вычислений, значение которого подаётся на выход «dataOut».

Автомат обладает 5 состояниями «S0», «S1», «S2», «S3», «S4» и «S5», переключается между ними последовательно, останавливаясь в состоянии «S5». В состоянии «S0» значения регистров устанавливаются в 0. В состояниях «S1», «S2», «S3» и «S4» записываются данные в регистры «REG_A», «REG_B», «REG_C», «REG_D» соответственно. В состоянии «S5» в «REG_RES» записывается результат «REG_A - REG_B << REG_C * REG_D» и «R_O» присваивается значение 1. При поступлении сигнала «reset» состояние автомата сбрасывается в «S0».

Код модуля представлен в Листинге 1.1.

Листинг 1.1 – Реализация модуля конечного автомата

```
module fsm(
    input signed [15:0] dataIn,
    input R_I,
    input reset,
    input clk,
    output signed [15:0] dataOut,
    output [2:0] current_state,
    output reg R_O
);
parameter S0 = 0, S1 = 1, S2 = 2, S3 = 3, S4 = 4, S5 = 5;
reg [2:0] state, new_state;
reg signed [15:0] REG_A, REG_B, REG_C, REG_D;
reg signed [15:0] REG_RES;

initial
begin
    state = S0;
```

Продолжение Листинга 1.1

```
    new_state = 0;
    REG_A = 0;
    REG_B = 0;
    REG_RES = 0;
    R_O = 0;
end

always@(posedge clk)
begin
    if (reset)
        state <= S0;
    else
        state <= new_state;
end

always@(posedge clk)
begin
    case(state)
        S0:
        begin
            REG_A <= 0;
            REG_B <= 0;
            REG_C <= 0;
            REG_D <= 0;
            REG_RES <= 16'b0;
            new_state <= S1;
            R_O <= 0;
        end
        S1: if (R_I)
        begin
            REG_A <= dataIn;
            new_state <= S2;
        end
        S2: if (R_I)
        begin
            REG_B <= dataIn;
            new_state <= S3;
        end
        S3: if (R_I)
        begin
            REG_C <= dataIn;
            new_state <= S4;
        end
        S4: if (R_I)
        begin
            REG_D <= dataIn;
            new_state <= S5;
        end
        S5:
        begin
            REG_RES <= REG_A - REG_B << REG_C * REG_D;
            R_O <= 1;
        end
    endcase
end

assign dataOut = REG_RES;
assign current_state = state;

endmodule
```

1.2 Создание модуля верхнего уровня

Для создания модуля верхнего уровня будут использованы дополнительные модули, разработанные ранее в лабораторной работе 1. Код для модулей, описывающих синхронизатор, делитель частоты, фильтр дребезга контактов, счетчик и модуль управления семисегментными индикаторами представлен в листингах 1.2–1.6.

Листинг 1.2 – Реализация модуля синхронизатора

```
module synchronizer(
    input in, clk,
    output out);

reg a, b;

always@(posedge clk)
begin
    b <= a;
    a <= in;
end

assign out = b;

endmodule
```

Листинг 1.3 – Реализация модуля счётчика

```
module counter #(STEP = 1, MODULE = 2)(
    input clk, reset, enable, direction,
    output reg[$clog2(MODULE)-1:0] cnt
);

initial cnt = 0;

always@(posedge clk)
begin
    if (reset)
        cnt <= 0;
    else if (enable)
        cnt <= direction ? (MODULE + cnt - STEP) % MODULE : (cnt + STEP) %
MODULE;
end

endmodule
```

Листинг 1.4 – Реализация модуля делителя частоты

```
module clk_divider #(DIV = 2) (
    input clk,
    output reg clk_div);

wire [$clog2(DIV/2)-1:0] cnt;

counter #(.STEP(1), .MODULE(DIV/2)) cntr(
    .clk(clk),
```

Продолжение Листинга 1.4

```
    .reset(1'b0),
    .enable(1'b1),
    .direction(1'b0),
    .cnt(cnt)
);

initial clk_div = 0;

always@(posedge clk)
    if (cnt == 0)
        clk_div = ~clk_div;

endmodule
```

Листинг 1.5 – Реализация модуля фильтра дребезга кантактов

```
module debouncer #(MODULE = 8) (
    input clk, in_signal, CLOCK_ENABLE,
    output reg out_signal, reg out_signal_enable
 );

wire sync_signal;
wire [$clog2(MODULE)-1:0] counter_res;

synchronizer sync(.in(in_signal), .clk(clk), .out(sync_signal));

counter #(.MODULE(MODULE), .STEP(1)) cntr(
    .clk(clk),
    .reset(sync_signal~^out_signal),
    .enable(CLOCK_ENABLE),
    .direction(1'b0),
    .cnt(counter_res)
);

always@(posedge clk)
begin
    if (&(counter_res) & CLOCK_ENABLE)
        out_signal <= sync_signal;
    out_signal_enable <= &(counter_res) & sync_signal & CLOCK_ENABLE;
end


endmodule
```

Листинг 1.6 – Реализация модуля управления семисегментыми индикаторами

```
module SevenSegmentLED(
    input [7:0] AN_MASK,
    input [31:0] NUMBER,
    input clk,
    input RESET,
    output [7:0] AN,
    output reg[7:0] SEG);

wire[2:0] counter_res;

counter #(.MODULE(8), .STEP(1)) cntr(
    .clk(clk),
    .reset(RESET),
    .enable(1'b1),
    .direction(1'b0),
    .cnt(counter_res)
);
```

Продолжение Листинга 1.6

```
reg [7:0] AN_REG = 0;
assign AN = AN_REG | AN_MASK;

wire [3:0] NUMBER_SPLITTER[0:7];
genvar i;
generate
    for (i = 0; i < 8; i = i + 1)
    begin
        assign NUMBER_SPLITTER[i] = NUMBER[((i+1)*4-1)-:4];
    end
endgenerate

always @(posedge clk)
begin
    if (RESET)
        SEG <= 8'b11111111;
    else
    begin
        case (NUMBER_SPLITTER[counter_res])
            4'h0: SEG <= 8'b11000000;
            4'h1: SEG <= 8'b11111001;
            4'h2: SEG <= 8'b10100100;
            4'h3: SEG <= 8'b10110000;
            4'h4: SEG <= 8'b10011001;
            4'h5: SEG <= 8'b10010010;
            4'h6: SEG <= 8'b10000010;
            4'h7: SEG <= 8'b11111000;
            4'h8: SEG <= 8'b10000000;
            4'h9: SEG <= 8'b10010000;
            4'ha: SEG <= 8'b10001000;
            4'hb: SEG <= 8'b10000011;
            4'hc: SEG <= 8'b11000110;
            4'hd: SEG <= 8'b10100001;
            4'he: SEG <= 8'b10000110;
            4'hf: SEG <= 8'b10001110;
            default: SEG <= 8'b11111111;
        endcase
        AN_REG = ~(8'b1 << counter_res);
    end
end

endmodule
```

Модуль верхнего уровня имеет название «controller». Он обладает следующими портами: шестнадцатибитный входной порт «SWITCHES» - значение, которое вводится с помощью рычажковых переключаетей, входной порт «button_in» - кнопка для разрешения записи, синхросигнал «clk», входной порт «button_reset_in» для сброса значени, выходной порт «AN» – шина разрешающих входов анодов для всех индикаторов, «SEG» - шина катодов для одного индикатора.

Создается экземпляр модуля «debouncer» с названием «dbnc», в единственный параметр которого передается 128. К портам «clk», «in_signal»,

«CLOCK_ENABLE» и «out_signal_enable» подключаются «clk», «button_in», «1'b1» и «button_signal_en» соответственно. Также создается экземпляр модуля «debouncer» с названием «dbncReset», в единственный параметр которого передается 512. К портам «clk», «in_signal», «CLOCK_ENABLE» и «out_signal_enable» подключаются «clk», «RESET», «1'b1» и « reset_signal_en» соответственно.

Создается экземпляр модуля «debouncer» с названием «dbnc», в единственный параметр которого передается 128. К портам «clk», «in_signal», «CLOCK_ENABLE», «out_signal» и «out_signal_enable» подключаются «clk», «button_in», «1'b1», «button_signal» и «button_signal_en» соответственно. Также создается экземпляр модуля «debouncer» с названием «dbnc_reset», в единственный параметр которого передается 128. К портам «clk», «in_signal», «CLOCK_ENABLE», «out_signal» и «out_signal_enable» подключаются «clk», «button_reset_in», «1'b1», «reset_signal» и «reset_signal_en» соответственно.

Создается экземпляр модуля «clk_div» с названием «div», в единственный параметр которого передается 1024. К портам «clk» и «clk_div» подключаются «clk» и «clk_div» соответственно.

Создается экземпляр модуля «SevenSegmentLED» с названием «led». К портам «AN_MASK», «NUMBER», «RESET», «clk», «AN» и «SEG» подключаются «mask», «number», «reset_signal», «clk_div», «AN» и «SEG» соответственно.

Создаётся экземпляр модуля «fsm» с названием «automat» К портам «dataIn», «R_I», «reset», «clk», «dataOut», «R_O» и «current_state» подключаются «SWITCHES», «button_signal_en», «reset_signal», «clk», «dataOut», «R_O» и «state» соответсвенно.

В блоке «always», работающему по переднему фронту, при «R_O», равном 1, в последние 16 бит «NUMBERS» записывается «dataOut», противном случае туда записывается «SWITCHES». В следующие 4 бита «NUMBERS» значение «A_{16}», «B_{16}», «C_{16}», «D_{16}» или «F_{16}» при значениях «state» «S1», «S2», «S3», «S4»,

«S5» соответственно. При «reset_signal», равном 1, выполняется установка значений «NUMBER» в 0 и «AN_MASK» в «8'b11111111».

Код модуля верхнего уровня представлен в Листинге 1.7.

Листинг 1.7 – Реализация модуля управления семисегментыми индикаторами

```
`timescale 1ns / 1ps

module controller(
    input signed [15:0] SWITCHES,
    input button_in, button_reset_in,
    input clk,
    output [7:0] AN,
    output [7:0] SEG
);

wire button_signal, button_signal_en, reset_signal_en, reset_signal, clk_div,
R_O;
reg [7:0] AN_MASK = 8'b11111111;
reg signed [31:0] NUMBER = 0;
wire signed [15:0] dataOut;
parameter S1 = 1, S2 = 2, S3 = 3, S4 = 4, S5 = 5;
wire [2:0] stat;

debouncer #(128) dbnc(
    .clk(clk),
    .in_signal(button_in),
    .CLOCK_ENABLE(1'b1),
    .out_signal(button_signal),
    .out_signal_enable(button_signal_en));

debouncer #(128) dbnc_reset(
    .clk(clk),
    .in_signal(button_reset_in),
    .CLOCK_ENABLE(1'b1),
    .out_signal(reset_signal),
    .out_signal_enable(reset_signal_en));

clk_divider #(1024) div(
    .clk(clk),
    .clk_div(clk_div));

SevenSegmentLED led(
    .AN_MASK(AN_MASK),
    .NUMBER(NUMBER),
    .clk(clk_div),
    .RESET(reset_signal),
    .AN(AN),
    .SEG(SEG));

fsm automat(.dataIn(SWITCHES), .R_I(button_signal_en), .reset(reset_signal),
.clk(clk), .dataOut(dataOut), .R_O(R_O), .current_state(stat));

always@(posedge clk)
begin
    if (R_O)
         NUMBER <= {NUMBER[31:16], dataOut};
    else
        NUMBER <= {NUMBER[31:16], SWITCHES};
    case(stat)
        S1:
```

Продолжение Листинга 1.7

```
            NUMBER[20:16] <= 4'ha;
        S2:
            NUMBER[20:16] <= 4'hb;
        S3:
            NUMBER[20:16] <= 4'hc;
        S4:
            NUMBER[20:16] <= 4'hd;
        S5:
            NUMBER[20:16] <= 4'hf;
    endcase
    AN_MASK <= {AN_MASK[7:5], 5'b0};
    if (reset_signal)
    begin
        NUMBER <= 0;
        AN_MASK <= 8'b11111111;
    end
end

endmodule
```

2 СОЗДАНИЕ ТЕСТОВЫХ МОДУЛЕЙ И ИХ ВЕРИФИКАЦИЯ

2.1 Создание тестов и верификация тестового модуля конечного автомата

Для тестирования были выбраны следующие наборы тестов:

— «a» равное 1101_2, «b» равное 1010_2, «c» равное 1001_2 и «d» равное 0001_2. Эталонным значением на выходе является 11000000000_2. Данный тест проверяет общую работоспособность написанного модуля.

— «a» равное 1111_2, «b» равное 0101_2, «c» равное 0010_2 и «d» равное 0110_2. Эталонным значением на выходе является 1010000000000000_2. Данный тест проверяет корректно ли будет работать модуль при сдвиге на отрицательное число.

— «a» равное 1111_2, «b» равное 0111_2, «c» равное 0011_2 и «d» равное 0100_2. Эталонным значением на выходе является 1000000000000000_2. Данный тест проверяет корректно ли будет работать с произведением отрицательных чисел.

Верификационное окружение для проведения тестов конечного автомата представлено представлен модулем «test_fsm». Объявляются однобитные регистры «clk», «btn», «reset», шестнадцатибитныей регистр «data» и шестнадцатибитныей цепь «res». Так же в нём создаётся экземпляр «fsm», в соответствующие порты которого подключаются все созданные ранее элементы.

В блоке «always» каждые 5 наносекунд регистр «clk» меняет свое значение на противоположное.

Далее блоком «task» с именем «cycle» задана функция теста, который присваивает регистру «data» по порядку значения, переданные ему в качестве аргументов, а также симулирует нажатие на кнопку подтверждения и сброса. В

конце, с помощью функции «$display», выводится информация об операции и её результате.

В блоке «intial» симулируется работа с конечным автоматом. Вызывается функция «cycle» с заранее сформированным набором аргументов, для проверки корректной работоспособности алгоритма.

Код тестового модуля представлен в Листинге 2.1. Результат верификации представлен на Рисунке 2.1.

Листинг 2.1 – Реализация тестового модуля конечного автомата

```
module fsm_test;

reg clk = 0;
reg btn = 0;
reg reset = 0;
reg [15:0] data = 0;
wire [15:0] res;

fsm fsm(
    .R_I(btn),
    .clk(clk),
    .dataIn(data),
    .reset(reset),
    .dataOut(res));

always #5 clk = ~clk;

task cycle;
    input [15:0] an, bn, cn, dn;
begin
    #15;
    data = an;
    btn = 1;
    #15;
    btn = 0;
    #15;
    data = bn;
    btn = 1;
    #15;
    btn = 0;
    #15;
    data = cn;
    btn = 1;
    #15;
    btn = 0;
    #15;
    data = dn;
    btn = 1;
    #15;
    btn = 0;
    #15;
    $display("Результат операции %0b - %0b << %0b * %0b = %0b", an, bn, cn
,dn, res);
    reset = 1;
    #15;
```

Продолжение Листинга 2.1

```
    reset = 0;
end
endtask

initial
begin

    cycle(16'b0000000000001101, 16'b0000000000001010 , 16'b0000000000001001,
16'b0000000000000001);

    cycle(16'b0000000000001111, 16'b0000000000000101 , 16'b0000000000000010,
16'b0000000000000110);

    cycle(16'b0000000000001111, 16'b0000000000000111 , 16'b0000000000000011,
16'b0000000000000100);
    #20

    $stop;

end


endmodule
```

```
Результат операции 1101 - 1010 << 1001 * 1 = 11000000000
Результат операции 1111 - 101 << 10 * 110 = 1010000000000000
Результат операции 1111 - 111 << 11 * 100 = 1000000000000000
```

Рисунок 2.1 – Результат верификации тестового модуля конечного автомата

Значения вычислений совпадают с эталонными, что подтверждает корректность работы модуля конечного автомата.

2.2 Создание и верификация тестового модуля управления семисегментными индикаторами

Для тестирования был выбран следующий набор тестов:

— Тест на отображение для всех возможных вариантов цифр;

— Тест анодной маски пройден.

Верификационное окружение для проведения тестов модуля управления семисегментными индикаторами представлен модулем «testSevenSeg». Объявляются однобитные регистры «clk», «RESET», тридцатидвухбитный регистр «NUMBER», восьмибитный регистр «AN_MASK», восьмибитная цепь «AN» и восьмибитная цепь «CATH». Так же в нём создаётся экземпляр «SevenSegmentLED», в соответствующие порты которого подключаются все

созданные ранее элементы. Также создаются регистры «test_digit_register» и «test_an_mask_register» для записи результатов тестов.

Дальше, функции «get_cath_mask», которая выдаёт значения катодов для числа, переданного в качестве аргумента, «get_an_mask», которая выдаёт значения анодов для индикатора, переданного в качестве аргумента.

Далее блоком «task» с именем «test_seven_segments», на вход которой подаётся анодная маска, проводит тесты, поочередно заполняя «NUMBER» числами от 1_{16} до F_{16}, проверяя корректность отображения чисел.

Далее блоком «task» с именем «test_show_stats» выводятся результаты тестирования по значениям, записанным в регистрах «test_an_register», «test_digit_register» и «test_an_mask_register».

Код тестового модуля представлен в Листинге 2.2. Результат верификации представлен на Рисунке 2.2.

Листинг 2.2 – Реализация тестового модуля семисегментного индикатора

```
`timescale 1ns / 1ps

module testSevenSeg;

reg clk;
initial clk = 0;
always #5 clk <= ~clk;

localparam AN_COUNT = 8;
localparam CATH_COUNT = 8;
localparam DIGIT_SIZE = 4;
localparam DIGIT_COUNT = 16;

reg CE, RESET;
reg [AN_COUNT*DIGIT_SIZE-1:0] NUMBER;
reg [AN_COUNT-1:0] AN_MASK;
initial
begin
    CE = 1;
    RESET = 0;
    NUMBER = {(AN_COUNT*DIGIT_SIZE){1'b0}};
end

wire [AN_COUNT-1:0] AN;
wire [CATH_COUNT-1:0] CATH;

SevenSegmentLED uut (
      .clk(clk),
      .RESET(RESET),
      .NUMBER(NUMBER),
      .AN_MASK(AN_MASK),
      .AN(AN),
```

Продолжение Листинга 2.2

```
      .SEG(CATH)
);

initial
begin
    test_seven_segments(8'b00101100);
    test_show_stats();
end

function [7:0] get_cath_mask;
    input [3:0] number;
begin
      case (number)
        4'h0:  get_cath_mask = 8'b11000000;
        4'h1:  get_cath_mask = 8'b11111001;
        4'h2:  get_cath_mask = 8'b10100100;
        4'h3:  get_cath_mask = 8'b10110000;
        4'h4:  get_cath_mask = 8'b10011001;
        4'h5:  get_cath_mask = 8'b10010010;
        4'h6:  get_cath_mask = 8'b10000010;
        4'h7:  get_cath_mask = 8'b11111000;
        4'h8:  get_cath_mask = 8'b10000000;
        4'h9:  get_cath_mask = 8'b10010000;
        4'ha:  get_cath_mask = 8'b10001000;
        4'hb:  get_cath_mask = 8'b10000011;
        4'hc:  get_cath_mask = 8'b11000110;
        4'hd:  get_cath_mask = 8'b10100001;
        4'he:  get_cath_mask = 8'b10000110;
        4'hf:  get_cath_mask = 8'b10001110;
     default:  get_cath_mask = 8'b11111111;
      endcase
end
endfunction

function [7:0] get_an_mask;
    input [2:0] an_number;
begin
    get_an_mask = ~(8'b1 << an_number);
end
endfunction

reg [DIGIT_COUNT-1:0] test_digit_register;
reg test_an_mask_register;

task test_seven_segments;
    input [AN_COUNT-1:0] mask_value;
reg [3:0] i;
reg [3:0] number;

begin
    $display("\n[%0t]: Тест отображения цифр на индикаторах, принципа работы
динамической индикации и анодной маски.", $time);
    test_digit_register = {DIGIT_COUNT{1'b1}};
    test_an_mask_register = 1'b1;

    AN_MASK = mask_value;
    $display("Битовая маска (AN_MASK): %b", AN_MASK);

    wait(uut.counter_res == AN_COUNT-1);
    @(posedge clk);
    number = 0;
```

Продолжение Листинга 2.2

```
    repeat(DIGIT_COUNT)
    begin
        // Подача числа на входную шину
        for (i = 0; i < AN_COUNT; i = i + 1)
            NUMBER[ ((i+1)*4)-1 -: 4 ] <= number;

        @(posedge clk);
        $display("\n[%0t]: Тест для цифры: %h", $time, number);
        for (i = 0; i < AN_COUNT; i = i + 1)
        begin
            #1;
            $display("Текущий анод: %d", i);

            test_digit_register[number] <= CATH == get_cath_mask(number);
            $display("Ожидаемые сигналы на линии катодов (CATH): %b",
get_cath_mask(number));
            $display("Фактические сигналы на линии катодов (CATH): %b", CATH);

            test_an_mask_register <= AN == (get_an_mask(i) | AN_MASK);
            $display("Ожидаемые сигналы на линии анодов (ПОСЛЕ применения
анодной маски): %b", get_an_mask(i) | AN_MASK);
            $display("Фактические сигналы на линии анодов (ПОСЛЕ применения
анодной маски): %b", AN);

            if (i != AN_COUNT-1)
                @(posedge clk);
        end
        number = number + 1;
    end
end
endtask

task test_show_stats;
    localparam TEST_COUNT = 2;
    integer test_counter, i;
begin
    test_counter = 0;
    $display("\n[%0t]: Результаты тестирования:", $time);
    // Отображение цифры
    if (&(test_digit_register))
    begin
        $display("1. Тест на отображение пройден успешно для всех возможных
вариантов цифр.");
        test_counter = test_counter + 1;
    end
    else begin
        $display("1. Тест на отображение цифр НЕ пройден");
        for (i = 0; i < DIGIT_COUNT; i = i + 1)
            if (!test_digit_register[i])
                $display("Ошибка отображения цифры %d", i);
    end

    // Анодная маска
    if (test_an_mask_register)
    begin
        $display("2. Тест анодной маски пройден успешно.");
        test_counter = test_counter + 1;
    end
    else
        $display("2. Тест анодной маски НЕ пройден.");
```

Продолжение Листинга 2.2

```
    $display("Пройдено тестов: %0d/%0d.", test_counter, TEST_COUNT);
end
endtask

endmodule
```

```
[543000]: Результаты тестирования:
1. Тест на отображение пройден успешно для всех возможных вариантов цифр.
2. Тест анодной маски пройден успешно.
Пройдено тестов: 2/2.
```

Рисунок 2.2 – Результат верификации тестового модуля семисегментного индикатора

2.3 Создание и верификация тестового модуля конечного устройства

Для тестирования был выбран набор тестов аналогичный тестам конечного автомата.

Верификационное окружение для проведения тестов модуля конечного автомата представлен модулем «testbench». Для корректного тестирования значения параметров для модулей «divider» и «debouncer» в «controller» были уменьшены до значения равного четырём. Объявляются однобитные регистры «clk», «button», «button_reset», шестнадцатибитный регистр «NUMBER», восьмибитная цепь «AN» и восьмибитная цепь «SEG». Так же в нём создаётся экземпляр «controller», в соответствующие порты которого подключаются все созданные ранее элементы.

Далее блоки «task» с именами «press_button» и «press_reset_button», которые имитирует дребезг контактов и нажатие кнопок «button» и «reset».

Далее блоком «task» с именем «input_data» с шестнадцатибитными входными параметрами «an», «bn», «cn», «dn», которые постепенно вводятся на «SWITCHES», в конце нажимается «reset».

Код тестового модуля представлен в Листинге 2.3.

Листинг 2.3 – Реализация тестового модуля для конечного устройства

```
`timescale 1ns / 1ps

module testbench();


reg signed [15:0] SWITCHES = 0;
```

Продолжение Листинга 2.3

```
reg clk = 0;
reg button = 0;
reg button_reset = 0;
wire[7:0] AN;
wire[7:0] SEG;


controller cntlr(
    .SWITCHES(SWITCHES),
    .button_in(button),
    .clk(clk),
    .button_reset_in(button_reset),
    .AN(AN),
    .SEG(SEG));

always #5 clk = ~clk;

task press_button;
    begin
        repeat($urandom_range(20, 0))
        begin
            button = $random;
            #3;
        end
        button = 1;
        #80;

        repeat($urandom_range(20, 0))
        begin
            button = $random;
            #3;
        end
        button = 0;
        #80;
    end
endtask

task press_reset_button;
    begin
        repeat($urandom_range(20, 0))
        begin
            button_reset = $random;
            #3;
        end
        button_reset = 1;
        #80;

        repeat($urandom_range(20, 0))
        begin
            button_reset = $random;
            #3;
        end
        button_reset = 0;
        #80;
    end
endtask

task input_data;
    input [15:0] an, bn, cn, dn;
    begin
    SWITCHES = an;
    press_button();
```

Продолжение Листинга 2.3

```
    SWITCHES = bn;
    press_button();

    SWITCHES = cn;
    press_button();

    SWITCHES = dn;
    press_button();
    #10;
    press_reset_button();
    end
endtask

initial
begin
    $srandom(35000);
    input_data(16'b0000000000001101, 16'b0000000000001010 ,
16'b0000000000001001, 16'b0000000000000001);

    input_data(16'b0000000000001111, 16'b0000000000000101 ,
16'b0000000000000010, 16'b0000000000000110);

    input_data(16'b0000000000001111, 16'b0000000000000111 ,
16'b0000000000000011, 16'b0000000000000100);
    $stop;
end

endmodule
```

Результат каждого теста можно проверить по временной диаграмме, рассматривая значение «NUMBER», так как именно оно будет записано в семисигментные индикаторы.

На Рисунке 2.3 представлена временная диаграмма для первого теста.

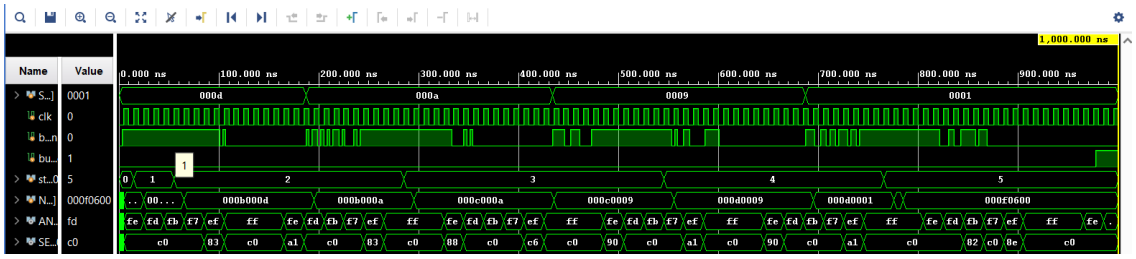

Рисунок 2.3 – Временная диаграмма первого теста

Как видно из рисунка 5 разряд «NUMBERS» корректно отображает переменные, которые вводятся «a», «b», «c» и «d», и «f» результат вычислений. $F = 0600_{16} = 11000000000_2$, что является верным значением для первого теста.

На Рисунке 2.4 представлена временная диаграмма для второго теста.

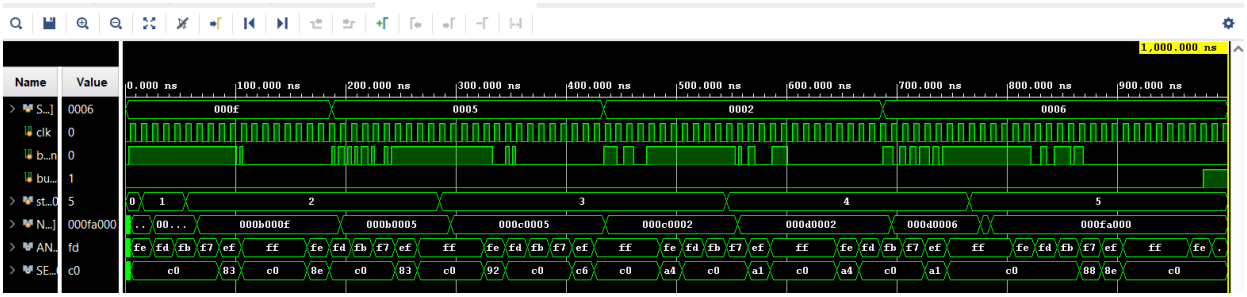

Рисунок 2.4 – Временная диаграмма второго теста

F = $a000_{16}$ = 1010000000000000_2 что является верным значением для второго теста.

На Рисунке 2.5 представлена временная диаграмма для третьего теста.

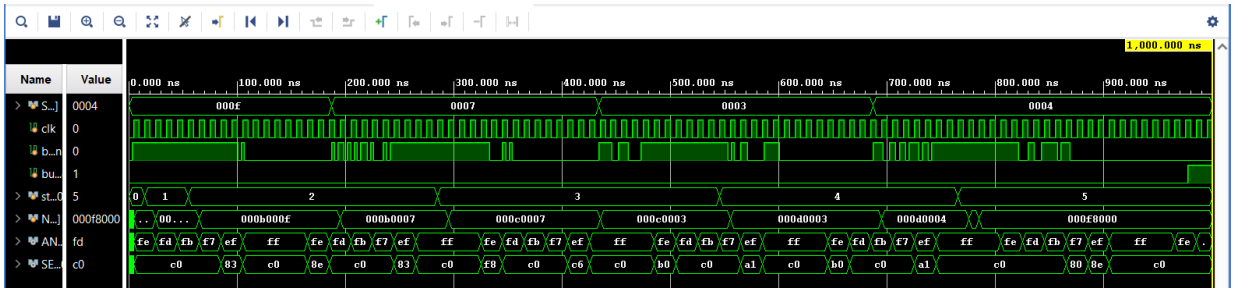

Рисунок 2.5 – Временная диаграмма третьего теста

F = 8000_{16} = 1000000000000000_2, что является верным значением для третьего теста.

3 ДОБАВЛЕНИЕ IP-ЯДРА VIO, СОЗДАНИЕ ФАЙЛА ПРОЕКТНЫХ ОГРАНИЧЕНИЙ И ЗАГРУЗКА ПРОЕКТА НА ОТЛАДОЧНУЮ ПЛАТУ NEXYS A7

Добавлено IP-ядро VIO, обладающее двумя входными портами по восемь бит, и тремя выходными, два из которых однибитные и один шестнадцатибитный. IP-ядро VIO представлено на Рисунке 3.1.

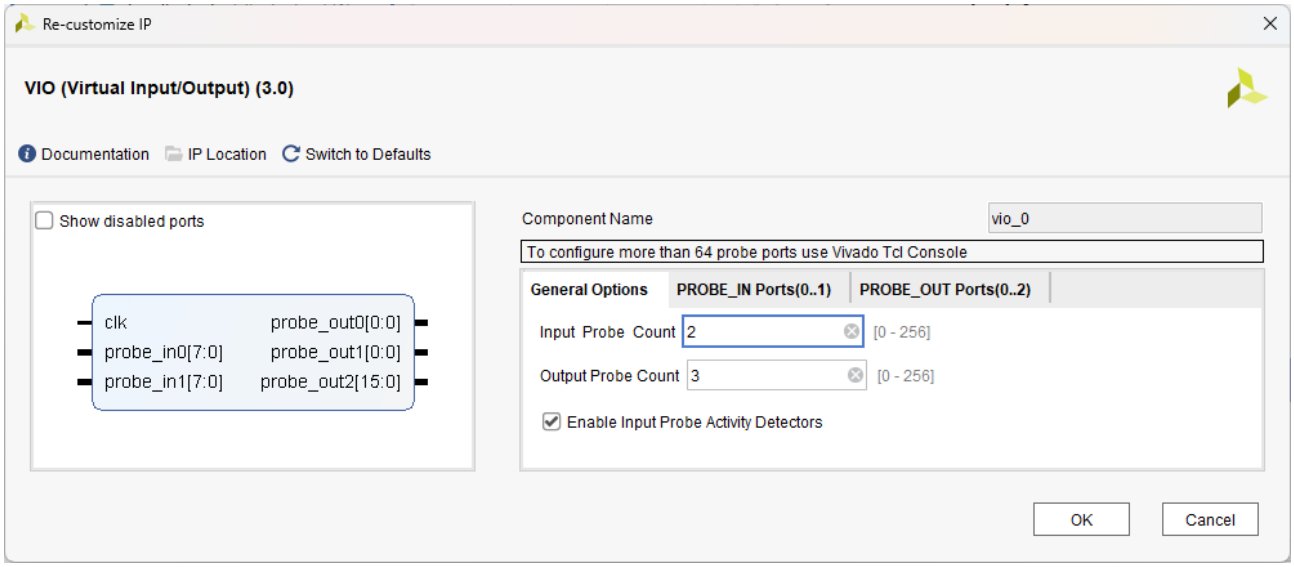

Рисунок 3.1 – IP-ядро VIO

Так же был создан модуль для подключения VIO. Код модуля представлен в Листинге 3.1.

Листинг 3.1 – Модуль VIO

```
`timescale 1ns / 1ps

module vioControl(
    input clk,
    output [7:0] AN,
    output [6:0] SEG
    );

wire button_in, RESET;
wire[15:0] SW;

vio_0 vio (
    .clk(clk),
    .probe_in0(AN),
    .probe_in1(SEG),
    .probe_out0(button_in),
    .probe_out1(RESET),
    .probe_out2(SW)
);

controller controller (
```

Продолжение Листинга 3.1

```
    .clk(clk),
    .button_in(button_in),
    .button_reset_in(RESET),
    .SWITCHES(SW),
    .AN(AN),
    .SEG(SEG)
);

endmodule
```

Содержание файла проектных ограничений представлено в Листинге 3.2.

Листинг 3.2 – Содержимое файла проектных ограничений

```
create_clock -add -name sys_clk -period 10.00 -waveform {0 5} [ get_ports {
clk }]
set_property -dict { IOSTANDARD LVCMOS33 PACKAGE_PIN E3 } [ get_ports { clk }
]

set_property -dict { IOSTANDARD LVCMOS33 PACKAGE_PIN J17 } [ get_ports { AN[0]
} ]
set_property -dict { IOSTANDARD LVCMOS33 PACKAGE_PIN J18 } [ get_ports { AN[1]
} ]
set_property -dict { IOSTANDARD LVCMOS33 PACKAGE_PIN T9 } [ get_ports { AN[2]
} ]
set_property -dict { IOSTANDARD LVCMOS33 PACKAGE_PIN J14 } [ get_ports { AN[3]
} ]
set_property -dict { IOSTANDARD LVCMOS33 PACKAGE_PIN P14 } [ get_ports { AN[4]
} ]
set_property -dict { IOSTANDARD LVCMOS33 PACKAGE_PIN T14 } [ get_ports { AN[5]
} ]
set_property -dict { IOSTANDARD LVCMOS33 PACKAGE_PIN K2 } [ get_ports { AN[6]
} ]
set_property -dict { IOSTANDARD LVCMOS33 PACKAGE_PIN U13 } [ get_ports { AN[7]
} ]

set_property -dict { IOSTANDARD LVCMOS33 PACKAGE_PIN T10 } [ get_ports {
SEG[0] } ]
set_property -dict { IOSTANDARD LVCMOS33 PACKAGE_PIN R10 } [ get_ports {
SEG[1] } ]
set_property -dict { IOSTANDARD LVCMOS33 PACKAGE_PIN K16 } [ get_ports {
SEG[2] } ]
set_property -dict { IOSTANDARD LVCMOS33 PACKAGE_PIN K13 } [ get_ports {
SEG[3] } ]
set_property -dict { IOSTANDARD LVCMOS33 PACKAGE_PIN P15 } [ get_ports {
SEG[4] } ]
set_property -dict { IOSTANDARD LVCMOS33 PACKAGE_PIN T11 } [ get_ports {
SEG[5] } ]
set_property -dict { IOSTANDARD LVCMOS33 PACKAGE_PIN L18 } [ get_ports {
SEG[6] } ]
set_property -dict { IOSTANDARD LVCMOS33 PACKAGE_PIN H15 } [ get_ports {
SEG[7] } ]
```

Проект был загружен на отладочную плату NEXYS A7 и протестирован. На Рисунке 3.2–3.4 представлена фотография платы с введёнными на ней значениями и VIO.

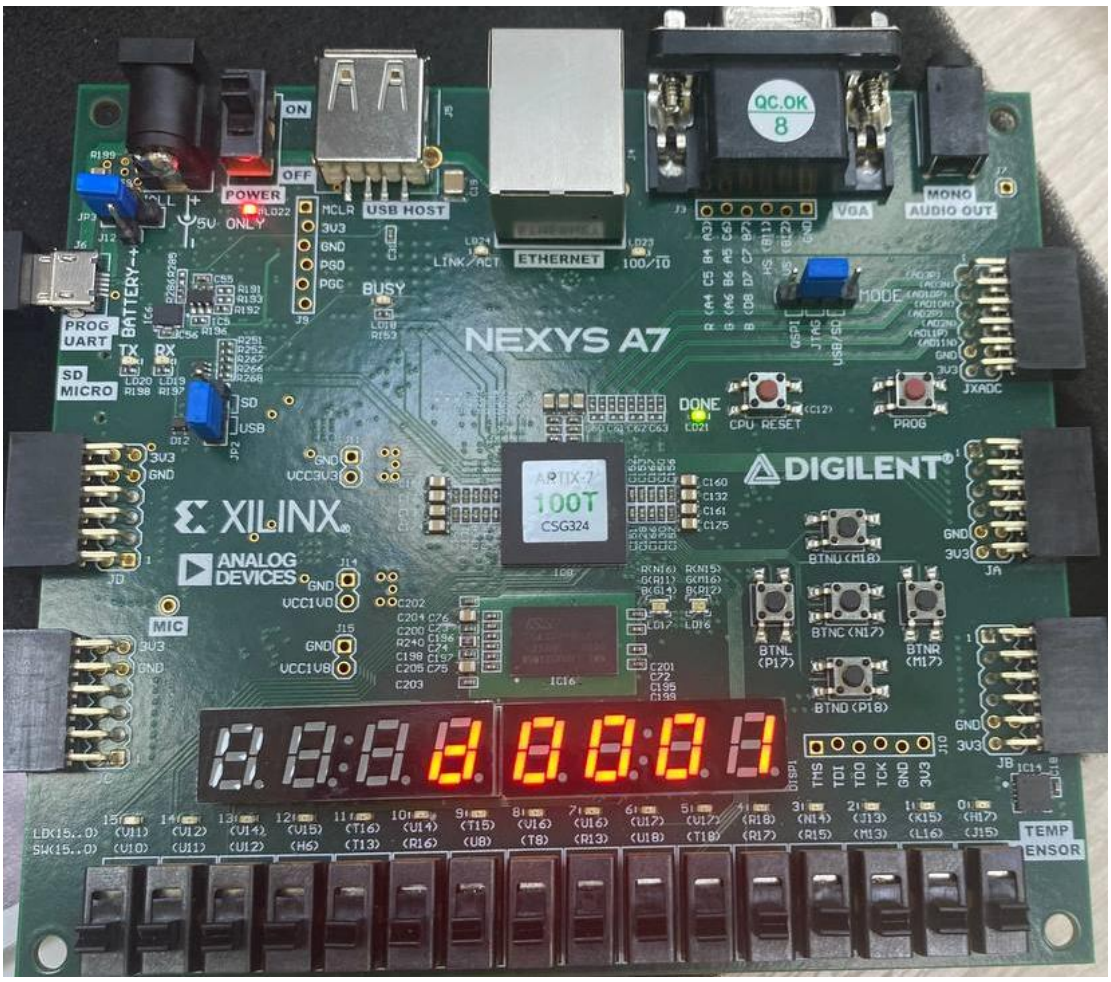

Рисунок 3.2 – Отображение параметра D на плате

hw_vio_1

Name	Value	Activity	Direction	VIO
RESET	[B] 0		Output	hw_vio_1
SW[15:0]	[H] 0001		Output	hw_vio_1
AN_OBUF[7:0]	[H] FF		Input	hw_vio_1
SEG_OBUF[7:0]	[H] C0		Input	hw_vio_1
button_in	[B] 1		Output	hw_vio_1

Рисунок 3.3 – Ввод значения праметра D в VIO

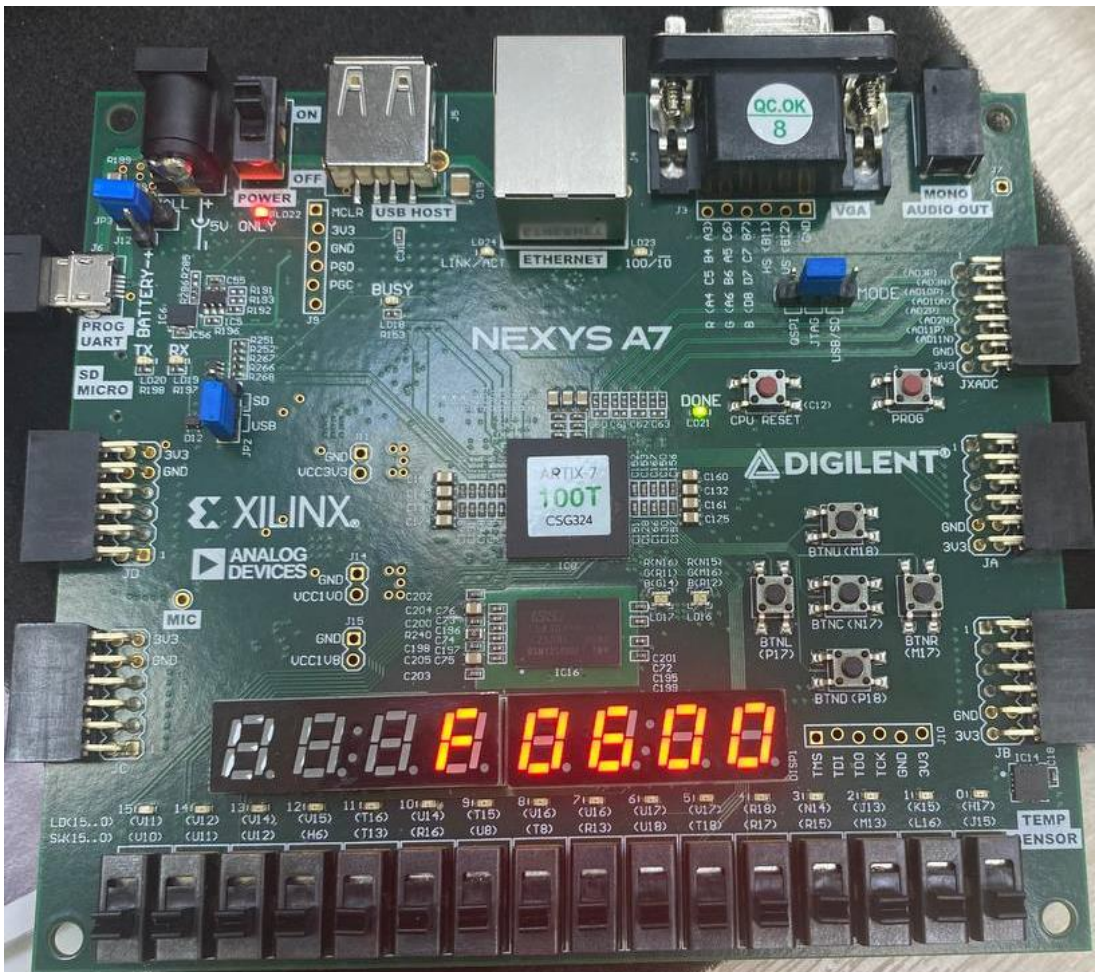

Рисунок 3.4 – Результат вычислений

ЗАКЛЮЧЕНИЕ

В результате выполнения практической работы приобретён навык основных подходов и использования основных инструментов для верификации проектов. Разработано устройство, представляющее из себя конечный автомат устройства, разработанного в прошлых практических, с подключённым набором устройств для ввода/вывода. Для каждого из устройств, входящих в конечное устройство, разработано верификационное окружение для проведения тестов и проведены соответствующие тесты. Для конечного устройства так же разработано верификационное окружение и произведена верификация с использованием VIO.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Методические указания по ПР № 3 — URL: https://online-edu.mirea.ru/mod/resource/view.php?id=413209 (Дата обращения: 12.03.2024).

2. Смирнов С.С. Информатика [Электронный ресурс]: Методические указания по выполнению практических и лабораторных работ / С.С. Смирнов — М., МИРЭА — Российский технологический университет, 2018. — 1 электрон. опт. диск (CD-ROM).

3. Тарасов И.Е. ПЛИС Xilinx. Языки описания аппаратуры VHDL и Verilog, САПР, приемы проектирования. — М.: Горячая линия — Телеком, 2021. — 538 с.: ил.