МИНОБРНАУКИ РОССИИ

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

РТУ МИРЭА

Институт Информационных Технологий
Кафедра Вычислительной Техники (ВТ)

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 3

«Построение синхронного цифрового автомата»

по дисциплине

«Схемотехника устройств компьютерных систем»

Выполнил студент группы Туктаров Т.А.

ИВБО-11-23

Принял ассистент кафедры ВТ Дуксина И.И.

Лабораторная работа выполнена «__»_____2025 г.

«Зачтено» «_»_____2025 г.

Москва 2025

АННОТАЦИЯ

Данная работа включает в себя 13 рисунков, 13 листингов. Количество страниц в работе — 29.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Все комбинационные схемы обладают следующей особенностью: они не имеют эффекта запоминания, т.е. в каждый момент времени значение на выходе меняется в зависимости от значений на входах. Обозначив множество значений логической функции, которая реализована посредством комбинационной схемы, как B и A – множество значений параметров функции [1], можно сказать, что b(t) = f(a(t)), где t – некоторый фиксированный дискретный момент времени. Иными словами, вне зависимости от предыдущего значения функции, новое значение формируется исключительно на основе входных данных.

При переходе от комбинационных схем к цифровым автоматам [2], встаёт вопрос о возможности запоминания информации при изменении входных значений. Пусть будут введены следующие обозначения:

1. A – множество входных символов.
2. B – множество выходных символов.

Для описания цифрового автомата необходимо добавить ещё одно конечное множество, называемое множеством состояний. Под состоянием будет пониматься некоторый «снимок» стабильных значений, составляющих некого объекта, в данном случае - цифрового автомата.

Пусть такое множество будет обозначено как Q. Введение этого множества позволит сохранять «историю» изменения значений, иными словами, разницу в поведении устройства (схемы) при, возможно, одинаковых входных данных можно будет обозначить, используя множество состояний совместно с вышеописанными множествами.

1 ХОД РАБОТЫ

1.1 Постановка задачи

Разработать верификационное окружение для цифрового устройства, реализованного на языке описания аппаратуры Verilog HDL. Провести моделирование и верификацию работы устройства согласно выданному варианту. Реализовать тестовый модуль, позволяющий проверить корректность работы проектируемого устройства на различных входных данных. Проанализировать роль верификации на этапах проектирования цифровых систем. Вариант: Субквадратичная сортировка массива

1.2 Системная модель

1.2.1 Создание алгоритма

Для данной задачи алгоритм с точки зрения достаточно высокого уровня абстракции будет представлен ниже:

1. Ввести размер массива
2. Если размер больше 8, то вывести код ошибки
3. Отсортировать массив сортировкой слияния
4. Вывести массив

1.2.2 Блок-схема алгоритма

На данном этапе, согласно алгоритму, требуется разработать блок-схему. Блок-схема алгоритма представлена на Рисунке 1.1.

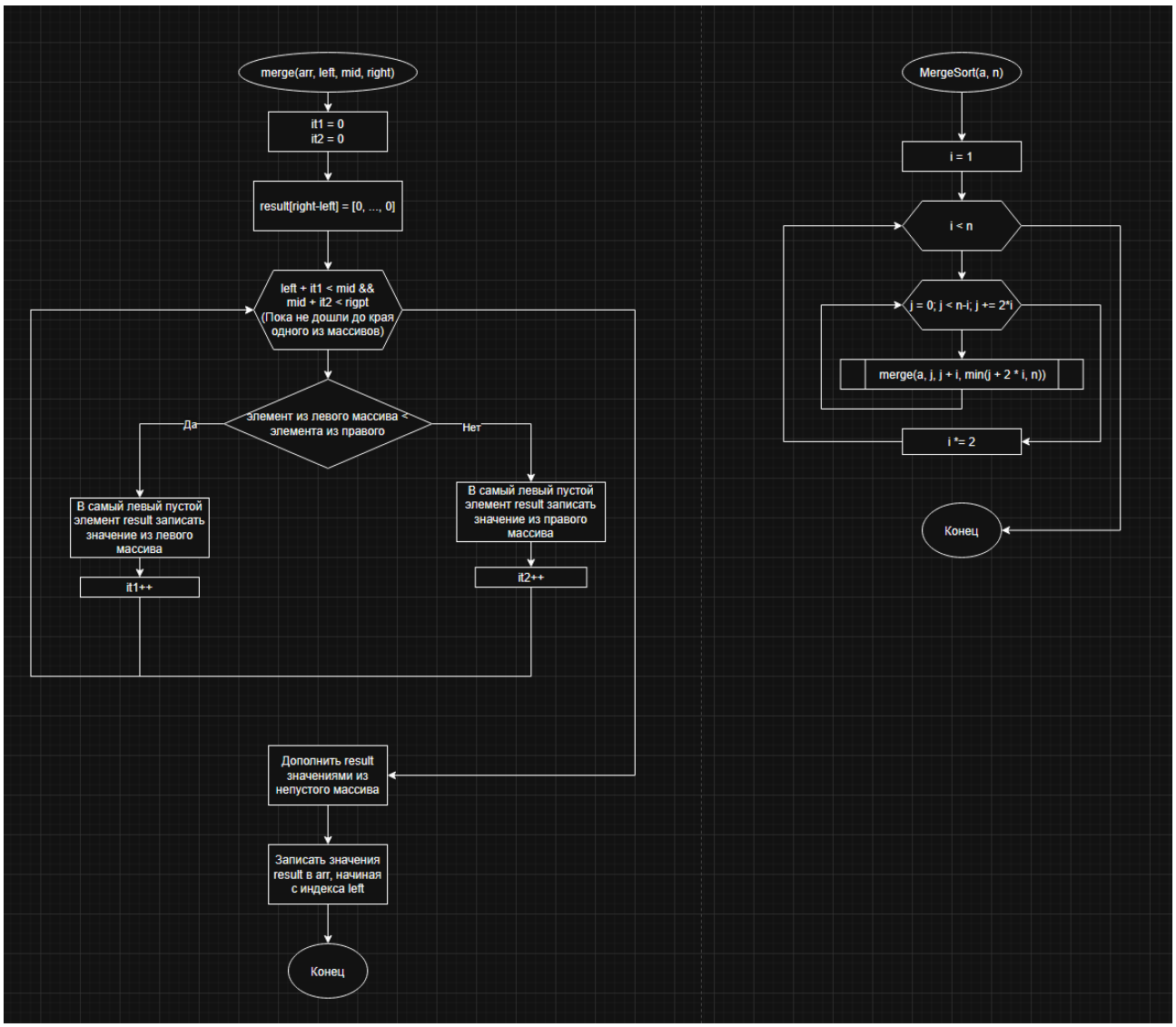

Рисунок 1.1 — Блок-схема алгоритма

1.2.3 Программная реализация системной модели

Программная реализация для поставленной задачи согласно системной модели представлена в Листинге 1.1.

Листинг 1.1 — Программная реализация на языке Python

```
def merge(arr, left, mid, right):
    it1 = 0
    it2 = 0

    result = [0 for _ in range(right - left)]

    while left + it1 < mid and mid + it2 < right:
        if arr[left + it1] < arr[mid + it2]:
            result[it1 + it2] = arr[left + it1]
            it1 += 1
        else:
            result[it1 + it2] = arr[mid + it2]
            it2 += 1
```

Продолжение листинга 1.1

```
    flag = False

    for i in range(it1, mid - left):
        result[i + it2] = arr[left + i]
        it1 = i
        flag = True

    if flag: it1 += 1

    flag = False

    for i in range(it2, right - mid):
        result[it1 + i] = arr[mid + i]
        it2 = i
        flag = True

    if flag: it2 += 1

    for i in range(it1 + it2):
        arr[left + i] = result[i]
    return result

def mergeSortIterative(a):
    if len(a) == 0:
        return [-1]
    i = 1
    n = len(a)
    while i < n:
        for j in range(0, n-i, 2 * i):
            merge(a, j, j + i, min(j + 2 * i, n))
        i *= 2
```

1.2.4 Тестовое покрытие системной модели

Результаты тестирования представлены в листинге 1.2. Таким образом, были сформированы эталонные наборы значений.

Листинг 1.2 — Результаты работы программной реализации

```
n: = 3 | arr: [7, 3, 1] | sorted: [1, 3, 7] | error code:  0
n: = 5 | arr: [7, 3, 1, 2] | sorted: [1, 2, 3, 7] | error code:  0
n: = 9 | arr: [] | sorted: [] | error code:  1
```

1.3 RTL-модель

1.3.1 Блок-схема работы цифрового автомата

Исходя из рассмотренного выше алгоритма требуется сформировать необходимый набор узлов. Составим блок-схему работы устройства. Реализация блок-схемы представлена на Рисунках 1.2 и 1.3

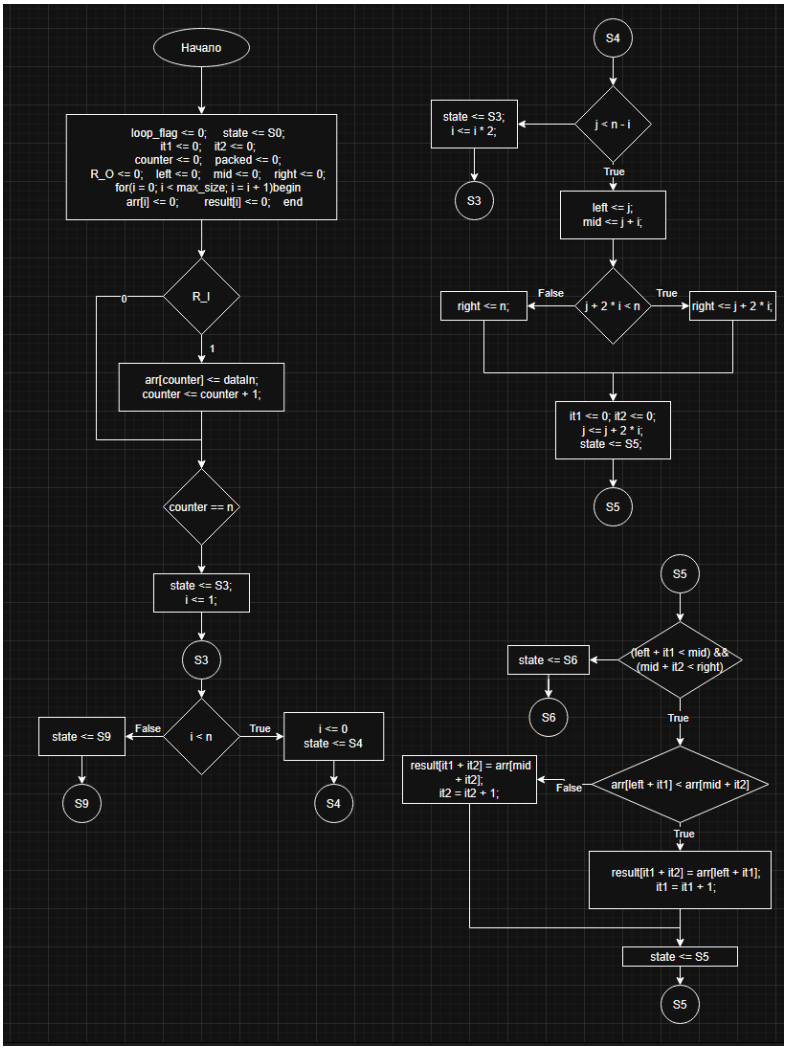

Рисунок 1.2 — Блок-схема работы цифрового автомата

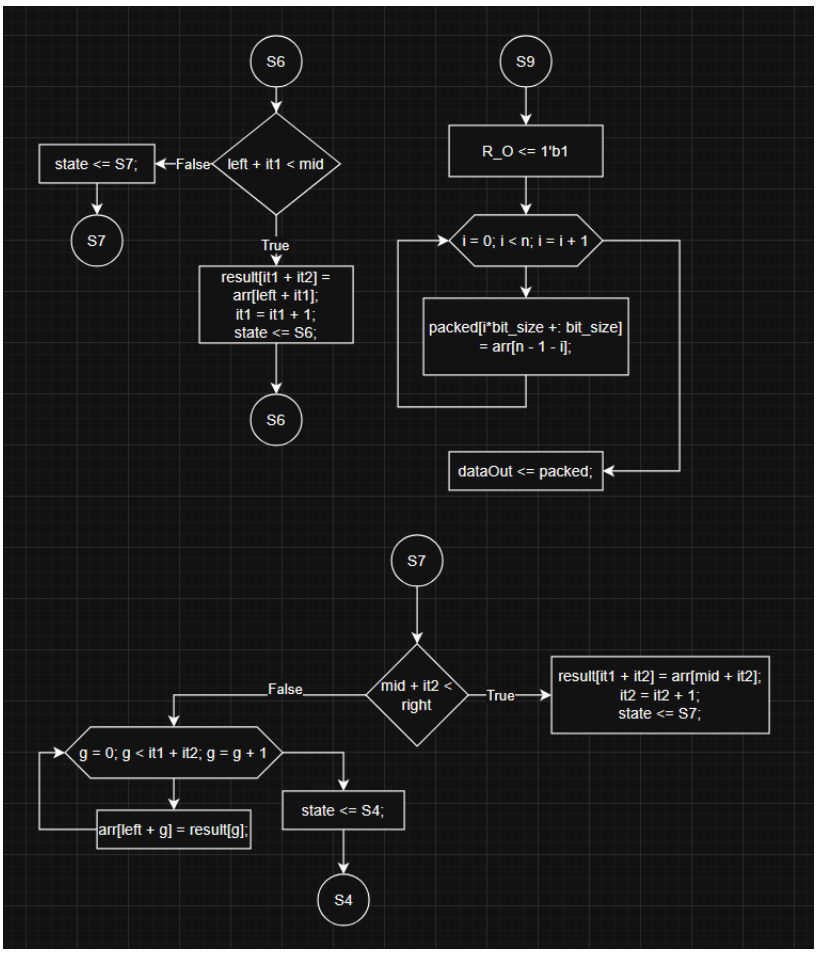

Рисунок 1.2 — Блок-схема работы цифрового автомата

1.3.2 Описание автомата на Verilog HDL

Исходный код представлен в Листинге 1.3.

Листинг 1.3 — Программная реализация автомата на Verilog HDL

```
module fsm#(max_size = 16, bit_size = 16)
(
    input [bit_size-1:0] n,
    input [bit_size-1:0] dataIn,
    input R_I,
    input reset,
    input clk,
    output reg [bit_size * max_size - 1: 0] dataOut,
    output reg R_O
    );

localparam S0 = 4'b0000, S1 = 4'b0001, S2 = 4'b0010, S3 = 4'b0011, S4 = 4'b0100,
S5 = 4'b0101, S6 = 4'b0110, S7 = 4'b0111, S8 = 4'b1000, S9 = 4'b1001;

reg [bit_size-1:0] arr [0:max_size-1]; // Массив

reg [bit_size * max_size - 1:0] flattered_arr; // Норм массив блэать.

reg [3:0] state; // состояния

reg [bit_size - 1:0] counter; // счетчик индекса массива

integer i;
integer j;
integer k;
integer g;

reg [bit_size - 1:0] it1;
reg [bit_size - 1:0] it2;

reg [bit_size - 1:0] result [0:max_size - 1]; // массив для хранения временных
отсортированных данных.

reg [1:0] z;

reg [bit_size * max_size - 1: 0] packed;

reg loop_flag;

reg [bit_size - 1:0] left;
reg [bit_size - 1:0] mid;
reg [bit_size - 1:0] right;
initial
begin
    loop_flag <= 0;
    state <= S0;
    // n <= 0;
    it1 <= 0;
    it2 <= 0;
    counter <= 0;
    packed <= 0;
    R_O <= 0;

    left <= 0;
    mid <= 0;
    right <= 0;
```

Продолжение листинга 1.3

```
    for(i = 0; i < max_size; i = i + 1)begin
        arr[i] <= 0;
        result[i] <= 0;
    end
end

always@(posedge clk)
begin
    if(reset)begin
        state <= S0;
    end
    else
    case(state)
        S0:
        begin
            // n <= 0;
            it1 <= 0;
            it2 <= 0;
            counter <= 0;
            R_O <= 0;
            dataOut <= 0;
            loop_flag <= 0;
            for(i = 0; i < max_size; i = i + 1)begin
                arr[i] = 0;
                result[i] = 0;
            end
            state <= S2;
        end
        S2: // ввод массива
        begin
            if(R_I) begin
                arr[counter] <= dataIn;
                counter <= counter + 1;
            end
            if(counter == n)begin
                state <= S3;
                i <= 1;
            end
        end // сортировка

        S3: begin
            if(i < n)begin
                j <= 0;
                state <= S4;
            end else state <= S9;
        end

        S4:begin
            if(j < n - i)begin
                left <= j;
                mid <= j + i;
                if(j + 2 * i < n)
                    right <= j + 2 * i;
                else
                    right <= n;
                it1 <= 0;
                it2 <= 0;
                $display(" i = %0h, j = %0h", i, j);
                j <= j + 2 * i;
                state <= S5;
            end else begin
                state <= S3;
```

Продолжение листинга 1.3

```
                i <= i * 2;
            end
        end

        S5: begin
            if((left + it1 < mid) && (mid + it2 < right))begin
                if(arr[left + it1] < arr[mid + it2])begin
                    result[it1 + it2] = arr[left + it1];
                    it1 = it1 + 1;
                end
                else begin
                    result[it1 + it2] = arr[mid + it2];
                    it2 = it2 + 1;
                end
                state <= S5;
            end else
                state <= S6;
        end

        S6:begin
            if(left + it1 < mid)begin
                result[it1 + it2] = arr[left + it1];
                it1 = it1 + 1;
                state <= S6;
            end else state <= S7;
        end

        S7:begin
            if(mid + it2 < right)begin
                result[it1 + it2] = arr[mid + it2];
                it2 = it2 + 1;
                state <= S7;
            end else state <= S8;
        end

        S8:begin
            for(g = 0; g < it1 + it2; g = g + 1)begin
                arr[left + g] = result[g];
            end
            state <= S4;
        end
        S9: begin
            R_O <= 1'b1;
            for (i = 0; i < max_size; i = i + 1) begin
                packed[i*bit_size +: bit_size] = arr[n - 1 - i];  // "+:" -
part-select
            end
            dataOut <= packed;
        end
    endcase
end
endmodule
```

1.3.3 Этап тестирования

На данном этапе основной целью является получение значений, идентичных эталонным, а также проверка различных особенностей работы

самой схемы. Пример тестового модуля представлен в Листинге 1.4. Результат симуляции представлен на Рисунке 1.3. В ходе тестирования были получены ответы, идентичные эталонным, а сброс автомата в действительности привёл автомат в начальное положение. [4]

Рисунок 1.3 — Временная диаграмма работы устройства

1.4 Реализация основных модулей

1.4.1 Реализация синхронизатора

Опишем синхронизатор в модулес synchro. Реализация представлена в Листинге 1.5.

Листинг 1.5 — Модуль synchro.v

```
module synchro(
 input in_signal,clk,
 output q
 );
 reg new_signal = 1'bx;reg last_signal = 1'bx;wire lq;
 dtrigger tr1(.C(clk),.D(new_signal),
 .en(1'b1),.q(lq));
 dtrigger tr2(.C(clk),.D(last_signal),.en(1'b1),.q(q));
 always @(posedge clk) begin new_signal=in_signal;
 last_signal=lq;end
endmodule
```

1.4.2 Реализация счётчика

Опишем счётчик в модулес count. Реализация представлена в Листинге 1.6.

Листинг 1.6— Модуль count.v

```
`timescale 1ns / 1ps
module count# (step = 1, mod = 16) (
    input clk,
    input dir,
    input RE,
    input CE,
    output reg [$clog2(mod) -1:0] out
);

initial out  = 0;
always@(posedge clk or posedge RE)
begin
    if (RE)
        out <= 0;
    else if (CE) begin
        if (dir == 0)
            out <= (out + step) % mod;
        else
            out <= (out - step) % mod;
    end
end

endmodule
```

1.4.3 Реализация триггера

Опишем d-trigger в модулес dtrigger. Реализация представлена в Листинге 1.7.

Листинг 1.7 — Модуль dtrigger.v

```
`timescale 1ns / 1ps
module dtrigger(
input wire C,
 input wire D,
 input wire en,
 output reg q
 );

initial begin
q<=0;
end

always @(posedge C) begin
  if(en) begin
    q <= D;
  end
  end

endmodule
```

1.4.4 Реализация делителя частоты

Опишем параметризированный делитель частоты в модулес clk_divider. Реализация представлена в Листинге 1.8.

Листинг 1.8 — Модуль clk_divider.v

```
`timescale 1ns / 1ps
module clk_divider#(div = 2) (
 input clk,
 output reg clk_div
 );
reg l;
wire [8:0] out;
count #(.step(1), .mod(div/2)) cntr(
 .clk(clk),
 .RE(1'b0),
 .CE(1'b1),
 .dir(1'b0),
 .out(out)
);
initial clk_div = 0;
always@(posedge clk)begin
 if (out ==0 && l==1)
 clk_div = ~clk_div;
  l<=1;
  end

endmodule
```

1.4.5 Реализация модуля управления семисегментными индикаторами

Опишем модуль управления семисегментными индикаторами в модуле SevenSegmentLED. Реализация представлена в Листинге 1.9.

Листинг 1.9 — Модуль SevenSegmentLED.v

```
module SevenSegmentLED(
 input [7:0] AN_MASK,
 input [31:0] NUMBER,
 input clk,
 output [7:0] AN,
 output reg[7:0] SEG);
wire[2:0] counter_res;
count #(.mod(8), .step(1)) cntr(
 .clk(clk),
 .RE(1'b0),
 .CE(1'b1),
 .dir(1'b0),
 .out(counter_res)
);

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
 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
endmodule
```

1.4.6 Реализация модуля фильтра дребезга контактов

Опишем фильтр дребезга контактов в модулес filtercon. Реализация представлена в Листинге 1.10.

Листинг 1.10 — Модуль filtercon.v

```
`timescale 1ns / 1ps

module filtercon #(mode = 2)(
 input in_signal,
 input clock_enable,
 input clk,
 output wire out_signal,
 output out_signal_enable,
 output wire [1:0] q_count
 );

 wire out_sync;
 wire out_first_and;
 wire out_second_and;
 wire out_third_and;
synchro syn(
 .in_signal(in_signal),
 .clk(clk),
 .q(out_sync)
 );

 count cs(
 .clk(clk),
 .RE(out_sync~^out_signal),
 .dir(0),
 .CE(clock_enable),
 .out(q_count)
 );

 dtrigger dt1(
 .C(clk),
 .D(out_sync),
 .en(out_second_and),
 .q(out_signal)
 );
 dtrigger dt2(
 .C(clk),
 .D(out_third_and),
 .en(1'b1),
 .q(out_signal_enable)
 );

   assign out_first_and = &q_count;
   assign out_second_and = out_first_and & clock_enable;
   assign out_third_and = out_second_and & out_sync;
endmodule
```

1.4.7 Реализация модуля верхнего уровня

Опишем модуль верхнего уровня в модуле main. Реализация представлена в Листинге 1.11.

Листинг 1.11 — Модуль main.v

```
`timescale 1ns / 1ps
module main(
    input [3:0] SWITCHES,
    input button_in, clk, button_reset_in,
    output [7:0] AN,
    output [6:0] SEG,
    output reg [31:0] NUMBER,
    output wire button_en,
    output wire button_reset_en,
    output reg error_output
);

    reg[7:0] AN_MASK = 8'b11111111;
    reg [3:0] dataIn;
    wire [15:0] dataOut;
    reg [3:0] n; // размер массива
    reg flag;
    reg error_flag;
    wire R_O;
    reg R_I;

    initial begin
        NUMBER <= 0;
        dataIn <= 0;
        flag <= 1'b1;
        R_I <= 0;
        error_flag <= 0;
    end
    filtercon #(128) dbnc(
        .clk(clk),
        .in_signal(button_in),
        .clock_enable(1'b1),
        .out_signal(button_signal),
        .out_signal_enable(button_signal_en)
    );

    filtercon #(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),
```

Продолжение листинга 1.11

```
        .AN(AN),
        .SEG(SEG)
    );

    fsm #(4, 4)fsm1 (
        .n(n),
        .clk(clk),
        .reset(reset_signal),
        .R_I(R_I), // todo button_signal_en
        .dataIn(dataIn),
        .dataOut(dataOut),
        .R_O(R_O) // todo
    );
    integer i;
    always@(posedge clk)
    begin
        if (reset_signal)
        begin
            NUMBER <= 0;
            AN_MASK <= 8'b11111111;
            flag <= 1'b1;
            error_flag <= 0;
            error_output <= 0;
        end
        else if(!error_flag)
            if (button_signal_en)
                begin
                    if(flag)begin
                        flag <= 0;
                        n <= SWITCHES;
                        if(SWITCHES > 8)begin
                            error_output <= 1'b1;
                            error_flag <= 1'b1;
                        end
                    end
                    else begin
                        if(!R_O)begin
                            R_I <= 1'b1;
                            dataIn <= SWITCHES;
                            NUMBER <= {NUMBER[27:0], SWITCHES};
                            AN_MASK <= {AN_MASK[6:0], 1'b0};
                        end
                    end
                end
            else begin
                if(R_O)begin
                    NUMBER <= 0;
                    NUMBER <= dataOut;
                    AN_MASK <= 8'b11111111;
                    AN_MASK = AN_MASK << n;
                end
                R_I <= 0;
            end
    end
endmodule
```

1.5 Создание и верификация тестового модуля верхнего уровня

Для тестирования был выбран набор тестов аналогичный тестам конечного автомата. Код тестового модуля представлен в Листинге 1.12.

Листинг 1.12 — Модуль testbench.v

```
`timescale 1ns / 1ps
module testbench2;
    reg[3:0] SWITCHES = 0;
    reg clk = 0;
    reg button = 0;
    reg button_reset = 0;
    wire[7:0] AN;
    wire[6:0] SEG;
    wire[31:0] NUMBER;
    wire button_en;
    wire button_reset_en;
    main cntlr(
        .SWITCHES(SWITCHES),
        .button_in(button),
        .clk(clk),
        .button_reset_in(button_reset),
        .AN(AN),
        .NUMBER(NUMBER),
        .SEG(SEG),
        .button_en(button_en),
        .button_reset_en(button_reset_en)
    );
    always #5 clk = ~clk;
    task button_press;
    begin
        repeat($urandom_range(50,0))
        begin
            button = $random;
            #3;
        end
        button = 1;
        #1000;

        repeat($urandom_range(50,0))
        begin
            button = $random;
            #3;
        end
        button = 0;
        #8000;
    end
    endtask

    task button_reset_press;
    begin
        repeat($urandom_range(50,0))
        begin
            button_reset = $random;
            #3;
        end
        button_reset = 1;
```

Продолжение листинга 1.12

```
        #1000;

        repeat($urandom_range(50,0))
        begin
            button_reset = $random;
            #3;
        end
        button_reset = 0;
        #8000;
    end
    endtask

    initial
    begin
        #4000
        $srandom(35000);

        SWITCHES = 4'd4;
        button_press;

        SWITCHES = 4'd7;
        button_press;

        SWITCHES = 4'd3;
        button_press;

        SWITCHES = 4'd1;
        button_press;

        SWITCHES = 4'd2;
        button_press;
    end
endmodule
```

Результат каждого теста можно проверить по временной диаграмме, рассматривая значение «NUMBER», так как именно оно будет записано в семисегментные индикаторы. [5]

На Рисунке 1.5 представлена временная диаграмма для первого теста.

Рисунок 1.5 — Временная диаграмма первого теста

На Рисунке 1.6 представлена временная диаграмма для второго теста.

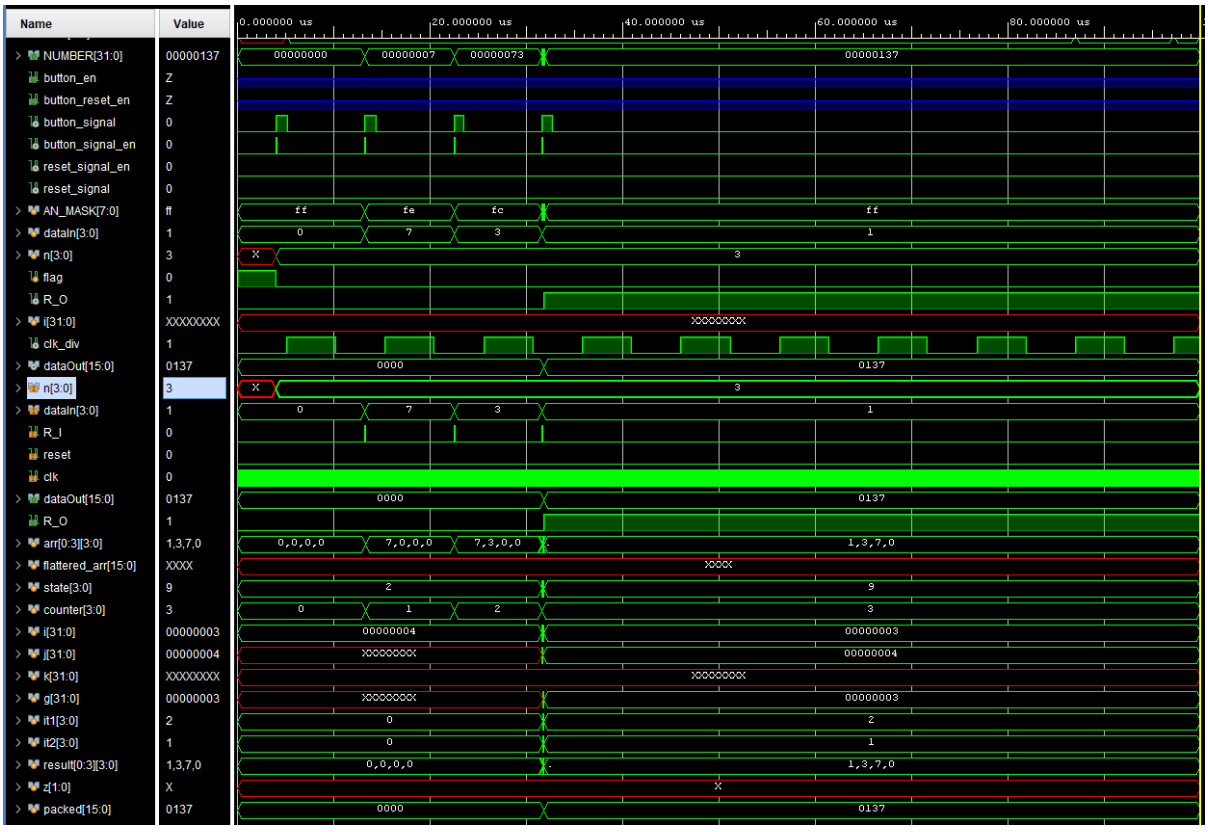

Рисунок 1.6 — Временная диаграмма второго теста

На Рисунке 1.7 представлена временная диаграмма для третьего теста.

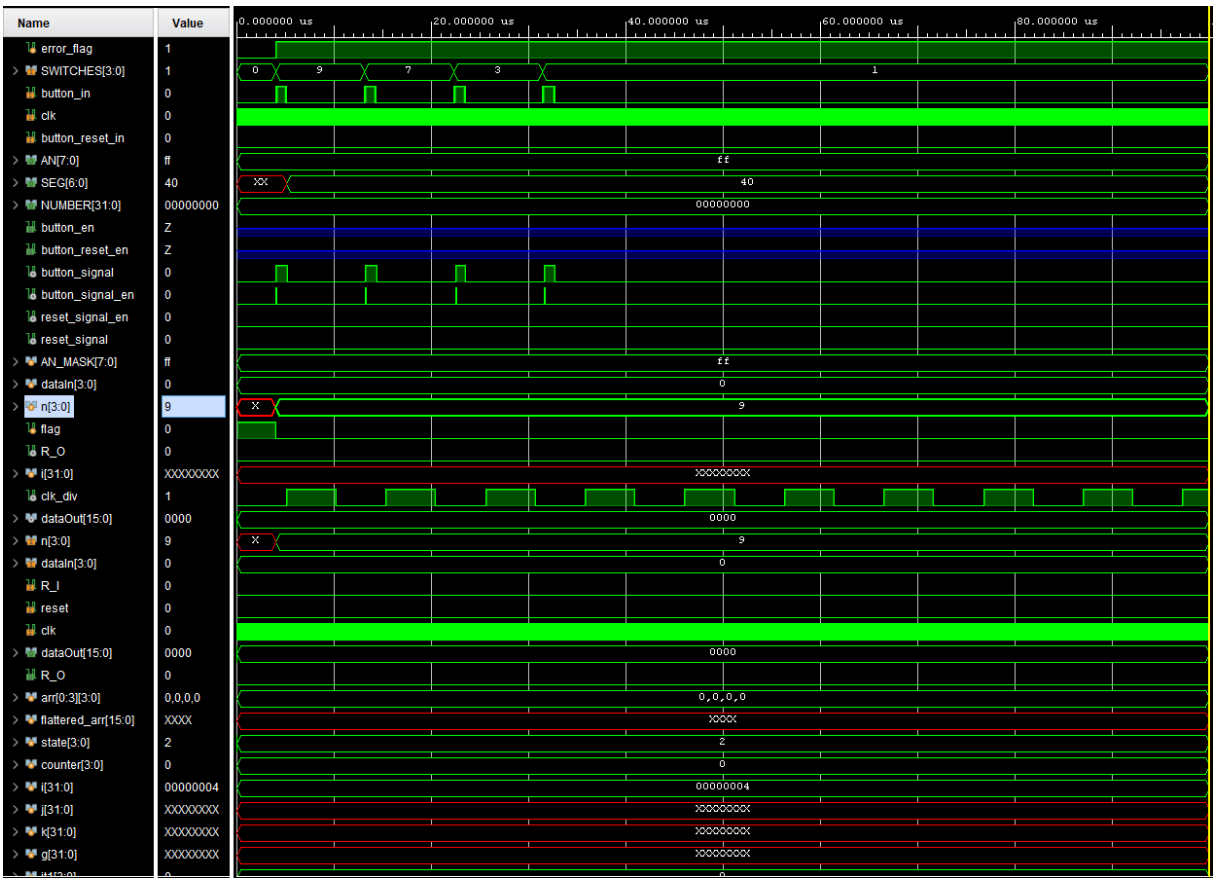

Рисунок 1.7 — Временная диаграмма третьего теста

1.6 Создание файла проектных ограничений и загрузка проекта на отладочную плату nexys a7

Содержание файла проектных ограничений представлено в Листинге 1.13.

Листинг 1.13 — Содержимое файла проектных ограничений

```
create_clock -period 10.000 -name sys_clk -waveform {0.000 5.000} -add
[get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property PACKAGE_PIN E3 [get_ports clk]
#кнопки
set_property PACKAGE_PIN N17 [get_ports button_in]
set_property IOSTANDARD LVCMOS33 [get_ports button_in]
set_property PACKAGE_PIN M18 [get_ports button_reset_in]
set_property IOSTANDARD LVCMOS33 [get_ports button_reset_in]
#свичи
set_property IOSTANDARD LVCMOS33 [get_ports {SWITCHES[0]}]
set_property PACKAGE_PIN J15 [get_ports {SWITCHES[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SWITCHES[1]}]
```

Продолжение листинга 1.13

```
set_property PACKAGE_PIN L16 [get_ports {SWITCHES[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SWITCHES[2]}]
set_property PACKAGE_PIN M13 [get_ports {SWITCHES[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SWITCHES[3]}]
set_property PACKAGE_PIN R15 [get_ports {SWITCHES[3]}]
#аноды
set_property IOSTANDARD LVCMOS33 [get_ports {AN[0]}]
set_property PACKAGE_PIN J17 [get_ports {AN[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[1]}]
set_property PACKAGE_PIN J18 [get_ports {AN[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[2]}]
set_property PACKAGE_PIN T9 [get_ports {AN[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[3]}]
set_property PACKAGE_PIN J14 [get_ports {AN[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[4]}]
set_property PACKAGE_PIN P14 [get_ports {AN[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[5]}]
set_property PACKAGE_PIN T14 [get_ports {AN[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[6]}]
set_property PACKAGE_PIN K2 [get_ports {AN[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[7]}]
set_property PACKAGE_PIN U13 [get_ports {AN[7]}]
#катоды
set_property IOSTANDARD LVCMOS33 [get_ports {SEG[0]}]
set_property PACKAGE_PIN T10 [get_ports {SEG[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEG[1]}]
set_property PACKAGE_PIN R10 [get_ports {SEG[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEG[2]}]
set_property PACKAGE_PIN K16 [get_ports {SEG[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEG[3]}]
set_property PACKAGE_PIN K13 [get_ports {SEG[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEG[4]}]
set_property PACKAGE_PIN P15 [get_ports {SEG[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEG[5]}]
set_property PACKAGE_PIN T11 [get_ports {SEG[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEG[6]}]
set_property PACKAGE_PIN L18 [get_ports {SEG[6]}]
#леды
set_property IOSTANDARD LVCMOS33 [get_ports {error_output}]
set_property PACKAGE_PIN N15 [get_ports {error_output}]
```

Проект был загружен на отладочную плату NEXYS A7 и протестирован. На Рисунках 1.7–1.13.

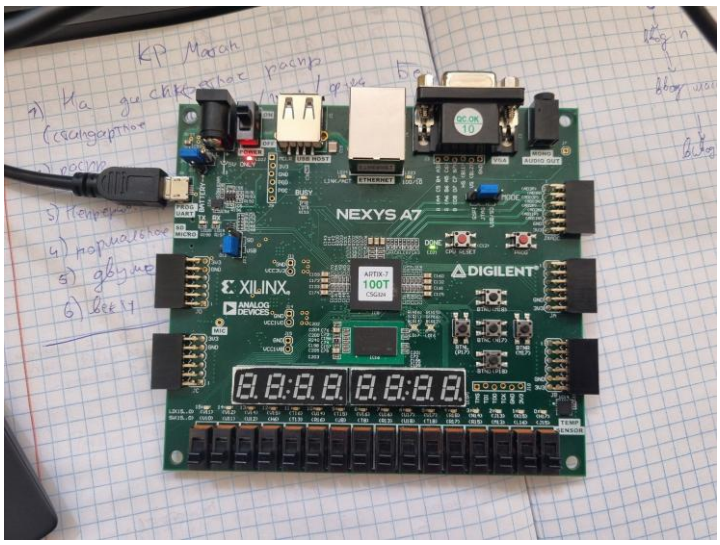

Рисунок 1.7 — Начальное состояние платы

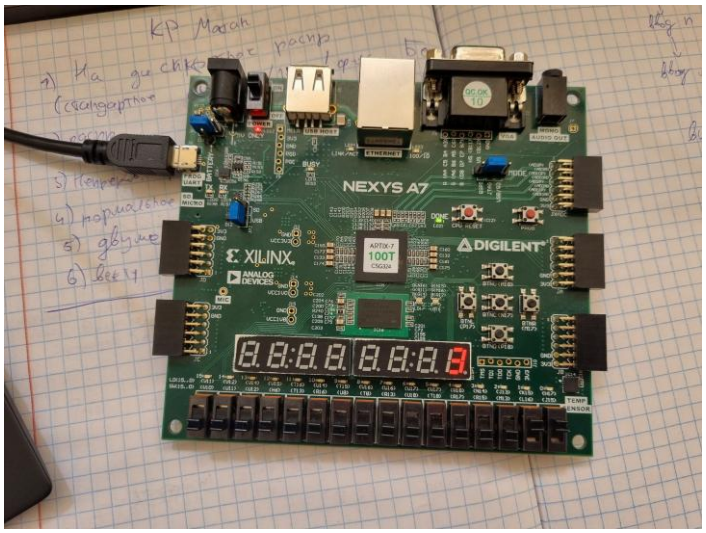

Рисунок 1.8 — Запись размера массива

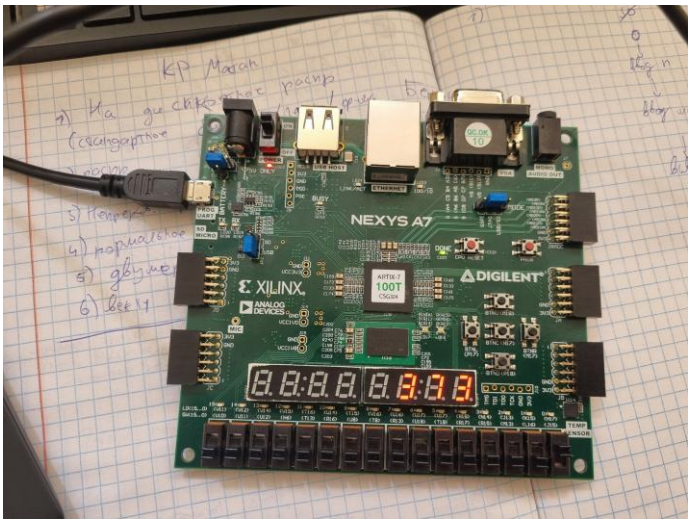

Рисунок 1.9 — Запись массива

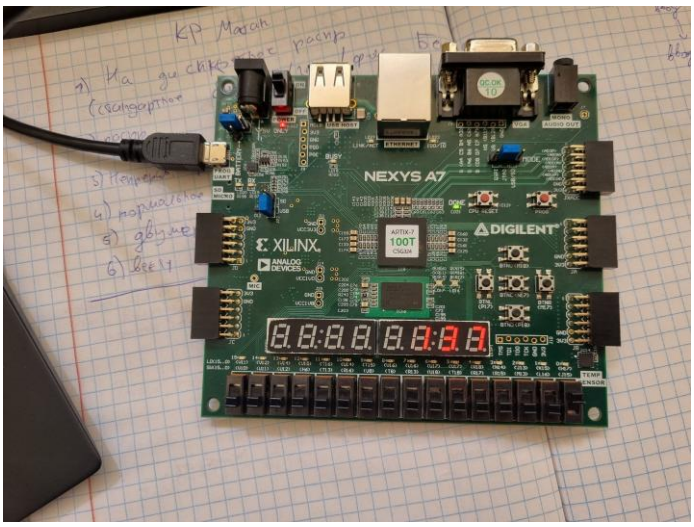

Рисунок 1.10 — Результат теста 2

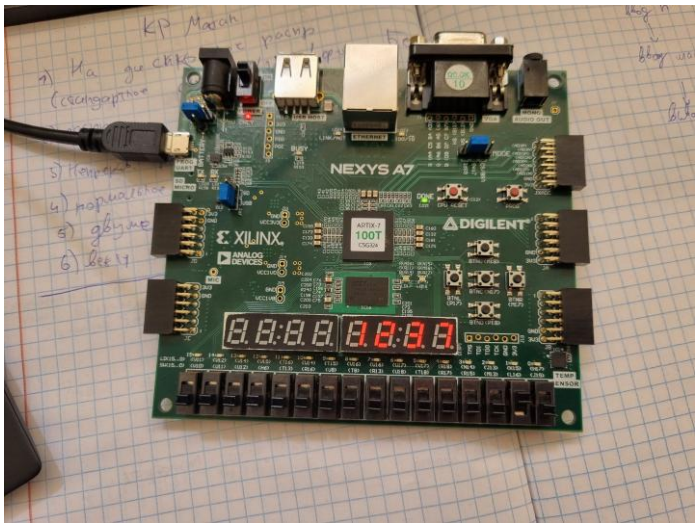

Рисунок 1.13 — Результат теста 1

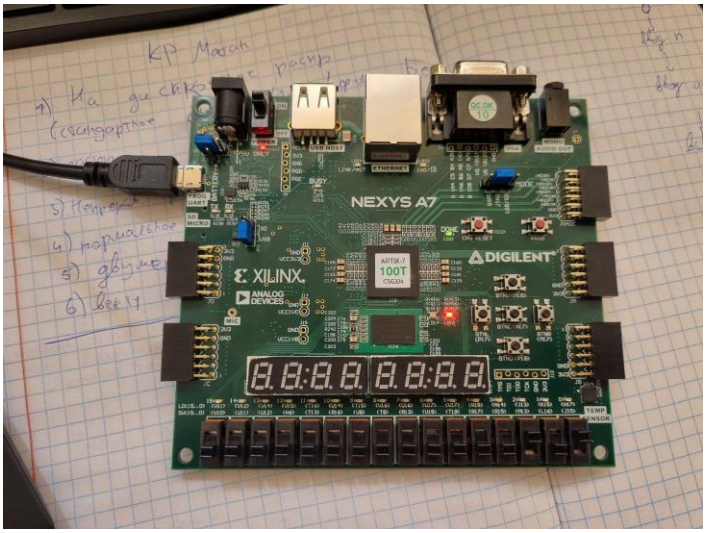

Рисунок 1.13 — Результат теста 3 как вывод ошибка

ЗАКЛЮЧЕНИЕ

В результате выполнения лабораторной работы студентами был освоен маршрут проектирования компонентов аппаратного обеспечения, студенты овладели навыком проектирования и реализации конечных автоматов, а также освоили механизм верификации проекта с использованием ПЛИС.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Дуксин, Н. А. Архитектура вычислительных машин и систем. Основы построения вычислительной техники: Практикум : учебное пособие / Н. А. Дуксин, Д. В. Люлява, И. Е. Тарасов. — Москва : РТУ МИРЭА, 2023. — 185 с.

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. Соловьев В. В. Основы языка проектирования цифровой аппаратуры Verilog. — М.: Горячая линия — Телеком, 2014. — 208 с.

4. Харрис Дэвид М., Харрис Сара Л. Цифровая схемотехника и архитектура компьютера. Издательство: ДМК-Пресс, 2018 г.

5. Максфилд К. Проектирование на ПЛИС. Курс молодого бойца. – М.: Издательский дом «Додэка-XXI», 2007. – 408 с.: илл. (Серия «Программируемые системы»)