1 ОСНОВНОЙ РАЗДЕЛ

1.1 Модуль ps2_keyboard

Данный модуль ps2_keyboard является обработчиком данных, получаемых из PS/2 клавиатуры. Он позволяет декодировать скан-коды, отправляемые клавиатурой, в байты данных. Модуль использует внешний тактовый сигнал clk для синхронизации своей работы, что позволяет обрабатывать асинхронные сигналы ps2_clk и ps2_data от клавиатуры.

Модуль отслеживает изменения на линии ps2_clk с помощью двух процедур, синхронизированных с внешним тактовым сигналом clk. Первая процедура копирует состояние ps2_clk в регистр ps2_clk_reg, что позволяет детектировать передний и задний фронт сигнала ps2_clk.

Если обнаружен задний фронт ps2_clk (сигнал перешёл из высокого в низкое состояние), модуль считывает бит из ps2_data и записывает его в shift_reg, начиная с младших битов. Это продолжается до тех пор, пока не будут считаны все 11 бит скан-кода (1 стартовый бит, 8 бит данных, 1 бит чётности, 1 стоповый бит).

Как только в регистре shift_reg окажется полный скан-код (10 битов данных и стоп-бит), модуль извлекает 8 бит данных (пропуская стартовый и биты чётности и стопа), помещает их в выходной регистр data и устанавливает флаг ready в 1. Это сигнализирует о том, что данные готовы к чтению. После того как данные прочитаны, флаг ready сбрасывается в 0, и процесс может начаться заново (Листинг 1.1).

Листинг 1.1 – Модуль ps2_keyboard Verilog

```
`timescale 1ns / 1ps


//0x1C ??? ??????? 'A' (?????????? ??????????)
//0x1B ??? ??????? 'S' (?????????? ??????????)
//0x15 ??? ??????? 'W' (?????????? ???????)
//0x1D ??? ??????? 'D' (?????????? ???????)
```

Продолжение листинга 1.1

```
module ps2_keyboard (
    input clk,            // ??????? ???????? ??????
    input reset,          // ?????? ??????
    input ps2_clk,        // ???????? ?????? PS/2
    input ps2_data,       // ?????? PS/2
    output reg [7:0] data,  // ????????? ???? ??????
    output reg ready        // ?????? ?????????? ??????
);

reg [3:0] bit_count = 0;    // ??????? ?????
reg [10:0] shift_reg = 0;  // ??????? ?????? ??? ?????? ??????
reg ps2_clk_reg;            // ??????????? ????????? ??????? PS/2

// ????????????? ???????? ??????? PS/2 Clock
always @(posedge clk) begin
    if (reset) begin
        ps2_clk_reg <= 1'b1;
    end else begin
        ps2_clk_reg <= ps2_clk;
    end
end


// ????????? ??????????? ?????? ??????? ps2_clk
always @(posedge clk) begin
    if (reset) begin
        bit_count <= 0;
        shift_reg <= 0;
        data <= 0;
        ready <= 0;
    end else begin
        if (ps2_clk_reg && !ps2_clk) begin // ?????????? ????? ps2_clk
            if (bit_count < 11) begin
                shift_reg <= {ps2_data, shift_reg[10:1]};
                bit_count <= bit_count + 1'b1;
            end

            if (bit_count == 10) begin
                // ???? ???? ?????? ??????? (1 ????? ???, 8 ??????, 1
??????, 1 ???? ???)
                data <= shift_reg[8:1]; // ????????? 8 ??? ??????
                ready <= 1'b1;
                bit_count <= 0;
                shift_reg <= 0;
            end
        end else if (ready) begin
            ready <= 0;  // ??????? ????? ??????????
        end
    end
end


endmodule
```

1.2 Модуль dynamic_pwm_controller

Далее опишем модуль dynamic_pwm_controller. Модуль dynamic_pwm_controller предназначен для генерации сигнала широтно-импульсной модуляции (ШИМ, или PWM) с динамически изменяемыми 2 параметрами скважностью (duty_cycle_input) и частотой (frequency_input)

duty_cycle_input – 8-битное значение, определяющее скважность ШИМ сигнала. Значение 0 соответствует 0%, а 255 соответствует 100%. frequency_input – 16-битное значение, определяющее частоту ШИМ сигнала в герцах .

MAX_PWM_FREQ – максимальная частота ШИМ сигнала, заданная как 10 кГц. MIN_PWM_FREQ – минимальная частота ШИМ сигнала, заданная как 1Гц. CLOCK_FREQ – частота входного тактового сигнала, заданная как 50 МГц.

На каждом такте часов проверяется значение frequency_input. Если оно больше нуля, max_count вычисляется как отношение частоты тактового сигнала CLOCK_FREQ к заданной частоте ШИМ frequency_input. Это определяет, сколько тактов основного сигнала составляет один период ШИМ. Если frequency_input равно нулю (что может быть ошибочной ситуацией), используется max_count, соответствующий минимальной частоте ШИМ, чтобы избежать деления на ноль.

Чтобы определить, должен ли выход pwm_out быть высоким или низким, сравнивается текущее значение counter с произведением max_count и duty_cycle_input. Это выражение определяет количество тактов в текущем периоде ШИМ, в течение которых выход pwm_out должен быть установлен в логическую «1» (высокий уровень). Использование деления на 256 связано с тем, что duty_cycle_input — 8-битное значение, максимальное значение которого равно 255, а не 256, что соответствует 100% скважности.

Если текущее значение counter меньше вычисленного порога threshold, то pwm_out устанавливается в «1». Если же counter больше или равен threshold, то pwm_out устанавливается в «0». Таким образом формируется выходной ШИМ

сигнал с заданными параметрами частоты и скважности (Листинг 1.2).

Листинг 1.2 – Модуль dynamic_pwm_controller Verilog

```
`timescale 1ns / 1ps

module dynamic_pwm_controller (
    input clk,
    input reset,
    input [31:0] angle,
    input [15:0] frequency_input,
    output reg pwm_out
);

    // Выходы CORDIC
    wire signed [16:0] sin_out, cos_out;

    // Входы для CORDIC
    reg [15:0] x_in = 16'd32767;
    reg [15:0] y_in = 16'd0;

    CORDIC cordic_inst (
        .clk(clk),
        .angle(angle),
        .x_in(x_in),
        .y_in(y_in),
        .sin_out(sin_out),
        .cos_out(cos_out)
    );

    parameter CLOCK_FREQ = 50_000_000;

    reg [31:0] max_count;
    reg [31:0] counter;
    reg [7:0] duty_cycle;

    always @(posedge clk or posedge reset) begin
        if (reset) begin
            counter <= 0;
            pwm_out <= 0;
            max_count <= CLOCK_FREQ / frequency_input;
        end else begin
            max_count <= CLOCK_FREQ / frequency_input;
            duty_cycle <= (sin_out[16:9] + 8'd128);

            if (counter < max_count - 1)
                counter <= counter + 1;
            else
                counter <= 0;
```

Продолжение Листинга 1.2

```
            if (counter < (max_count * duty_cycle) >> 8)
                pwm_out <= 1;
            else
                pwm_out <= 0;
        end
    end
endmodule
```

1.3 Модуль верхнего уровня

Модуль top_module представляет собой управление RGB-светодиодом с использованием сигналов ШИМ, параметры которого можно динамически изменять, реагируя на ввод с PS/2 клавиатуры. Модуль управляет цветом светодиода, плавно переливая его между различными цветами: от красного к жёлтому, зелёному, голубому, синему, фиолетовому и обратно к красному.

duty_cycle_R/G/B это 8-битные регистры, определяющие скважность ШИМ для красного, зелёного и синего каналов соответственно. color_timer счётчик для контроля скорости перехода между цветами. color_transition_rate скорость перехода между цветами. brightness_change_rate – скорость изменения яркости при переливании цветов (Листинг 1.3).

Листинг 1.3 – Модуль верхнего уровня Verilog

```
`timescale 1ns / 1ps

module top_module (
    input clk,
    input reset,
    input ps2_clk,
    input ps2_data,
    output pwm_out_R,
    output pwm_out_G,
    output pwm_out_B
);

    reg [31:0] angle_R = 0;
    reg [31:0] angle_G = 32'd715827883; // 2^32 / 3
    reg [31:0] angle_B = 32'd1431655765; // 2 * (2^32 / 3)
    reg [15:0] pwm_freq = 1000;


    always @(posedge clk or posedge reset) begin
        if (reset) begin
            angle_R <= 0; angle_G <= ANGLE_120; angle_B <= ANGLE_240;
        end else begin
            angle_R <= angle_R + 32'd50000;
            angle_G <= angle_G + 32'd50000;
            angle_B <= angle_B + 32'd50000; end
```

Продолжение Листинга 1.3

```
    end

    dynamic_pwm_controller pwm_ctrl_R (
        .clk(clk),
        .reset(reset),
        .angle(angle_R),
        .frequency_input(pwm_freq),
        .pwm_out(pwm_out_R)
    );

    dynamic_pwm_controller pwm_ctrl_G (
        .clk(clk),
        .reset(reset),
        .angle(angle_G),
        .frequency_input(pwm_freq),
        .pwm_out(pwm_out_G)
    );

    dynamic_pwm_controller pwm_ctrl_B (
        .clk(clk),
        .reset(reset),
        .angle(angle_B),
        .frequency_input(pwm_freq),
        .pwm_out(pwm_out_B)
    );

    wire [7:0] kb_data;
    wire kb_ready;

    ps2_keyboard kb (
        .clk(clk),
        .reset(reset),
        .ps2_clk(ps2_clk),
        .ps2_data(ps2_data),
        .data(kb_data),
        .ready(kb_ready)
    );

    reg [23:0] color_timer = 0;
    reg [3:0] state = 0;
    reg [15:0] color_transition_rate = 50000;
    reg [7:0] brightness_change_rate = 5;

    always @(posedge clk) begin
        if (reset) begin
            angle_R <= ANGLE_0; angle_G <= ANGLE_120; angle_B <= ANGLE_240;
            state <= 0;
            color_timer <= 0;
            color_transition_rate <= 50000;
            brightness_change_rate <= 5; end else begin
            if (kb_ready) begin
                case (kb_data)
                    8'h1D: begin // a
                        if (color_transition_rate < 1000000)
                            color_transition_rate <= color_transition_rate +
10000;
                    end
```

Продолжение Листинга 1.3

```
                    8'h23: begin // d
                        if (color_transition_rate > 10000)
                            color_transition_rate <= color_transition_rate -
10000;
                    end
                    8'h1A: begin // w
                        if (brightness_change_rate < 50)
                            brightness_change_rate <= brightness_change_rate
+ 1;
                    end
                    8'h1B: begin // s
                        if (brightness_change_rate > 1)
                            brightness_change_rate <= brightness_change_rate
- 1;
                    end
                    default: begin
                    end
                endcase
            end

            // auto change color
            color_timer <= color_timer + 1;
            if (color_timer >= color_transition_rate) begin
                color_timer <= 0;
                case (state)
                    0: begin
                        // Red -> Yellow
                        if (angle_G < 16'd65535)
                            angle_G <= angle_G + brightness_change_rate;
                        else
                            state <= 1;
                    end
                    1: begin
                        // Yellow -> Green
                        if (angle_R > 0)
                            angle_R <= angle_R - brightness_change_rate;
                        else
                            state <= 2;
                    end
                    2: begin
                        // Green -> Cyan
                        if (angle_B < 16'd65535)
                            angle_B <= angle_B + brightness_change_rate;
                        else
                            state <= 3;
                    end
                    3: begin
                        // Cyan -> Blue
                        if (angle_G > 0)
                            angle_G <= angle_G - brightness_change_rate;
                        else
                            state <= 4;
                    end
                    4: begin
                        // Blue -> Magenta
                        if (angle_R < 16'd65535)
                            angle_R <= angle_R + brightness_change_rate;
                        else
```

Продолжение Листинга 1.3

```
                                state <= 5;
                        end
                        5: begin
                            // Magenta -> Red
                            if (angle_B > 0)
                                angle_B <= angle_B - brightness_change_rate;
                            else
                                state <= 0;
                        end
                        default: state <= 0;
                    endcase
                end
            end
        end

endmodule
```

1.4 Файл проектных ограничений

Добавим в проект файл проектных ограничений (Листинг 1.4).

Листинг 1.4 – Файл проектных ограничений

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

# reset
set_property -dict { PACKAGE_PIN M17 IOSTANDARD LVCMOS33 } [get_ports {
reset }];

set_property -dict { PACKAGE_PIN R12 IOSTANDARD LVCMOS33 } [get_ports {
pwm_out_B }];
set_property -dict { PACKAGE_PIN M16 IOSTANDARD LVCMOS33 } [get_ports {
pwm_out_G }];
set_property -dict { PACKAGE_PIN N15 IOSTANDARD LVCMOS33 } [get_ports {
pwm_out_R }];

set_property -dict { PACKAGE_PIN F4 IOSTANDARD LVCMOS33 } [get_ports {
ps2_clk }];
set_property -dict { PACKAGE_PIN B2 IOSTANDARD LVCMOS33 } [get_ports {
ps2_data }];
```

1.5 Тестирование на плате

Протестируем работу проекта на плате (Рисунок 1.1).

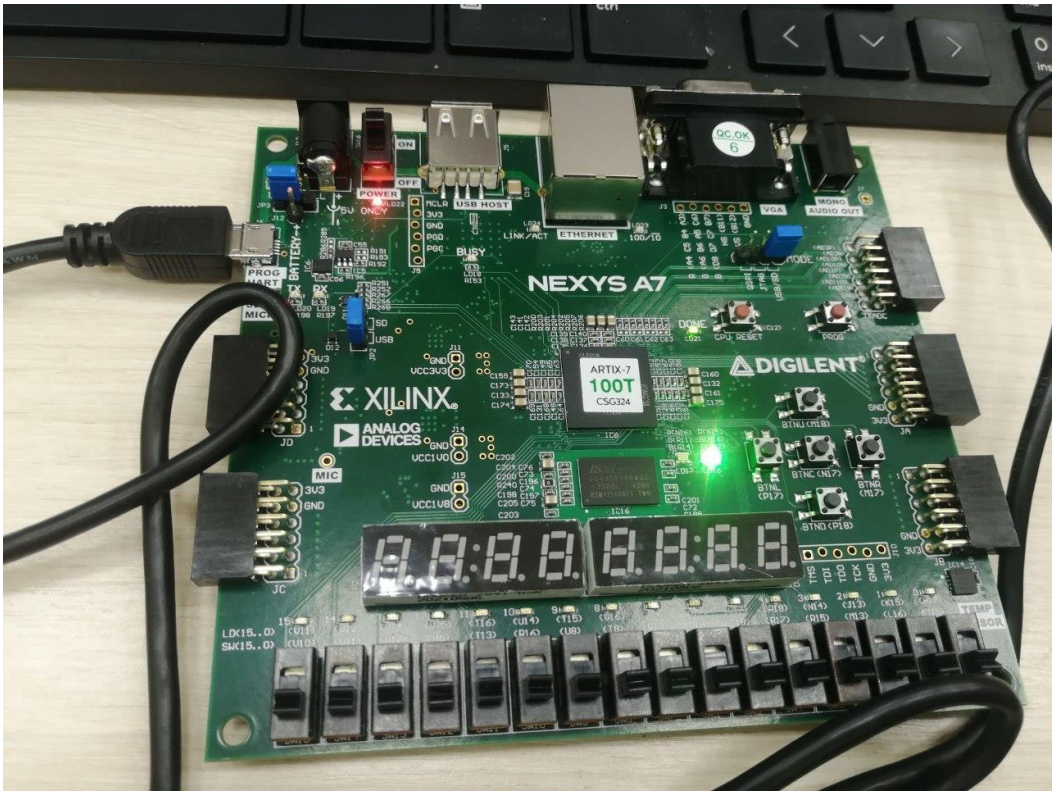

Рисунок 1.1 – тестирование на плате

Далее проверим как уменьшается яркость светодиода на плате (Рисунки 1.2, 1.3, 1.4).

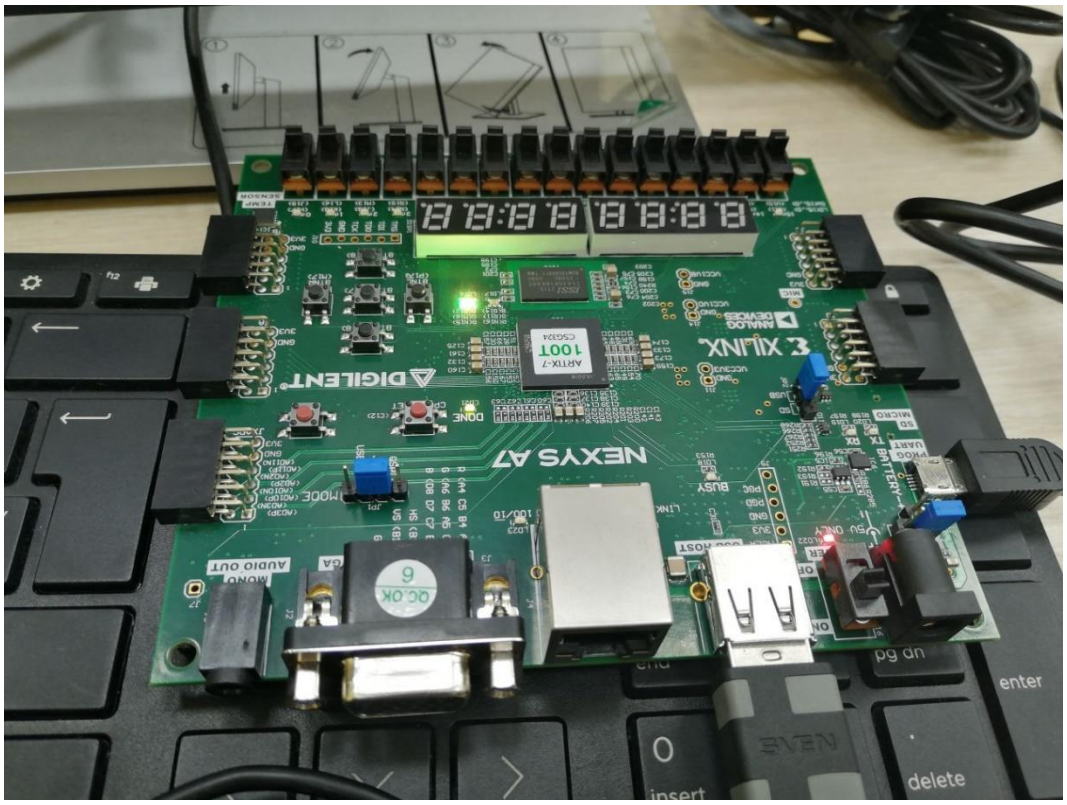

Рисунок 1.2 – уменьшение яркости

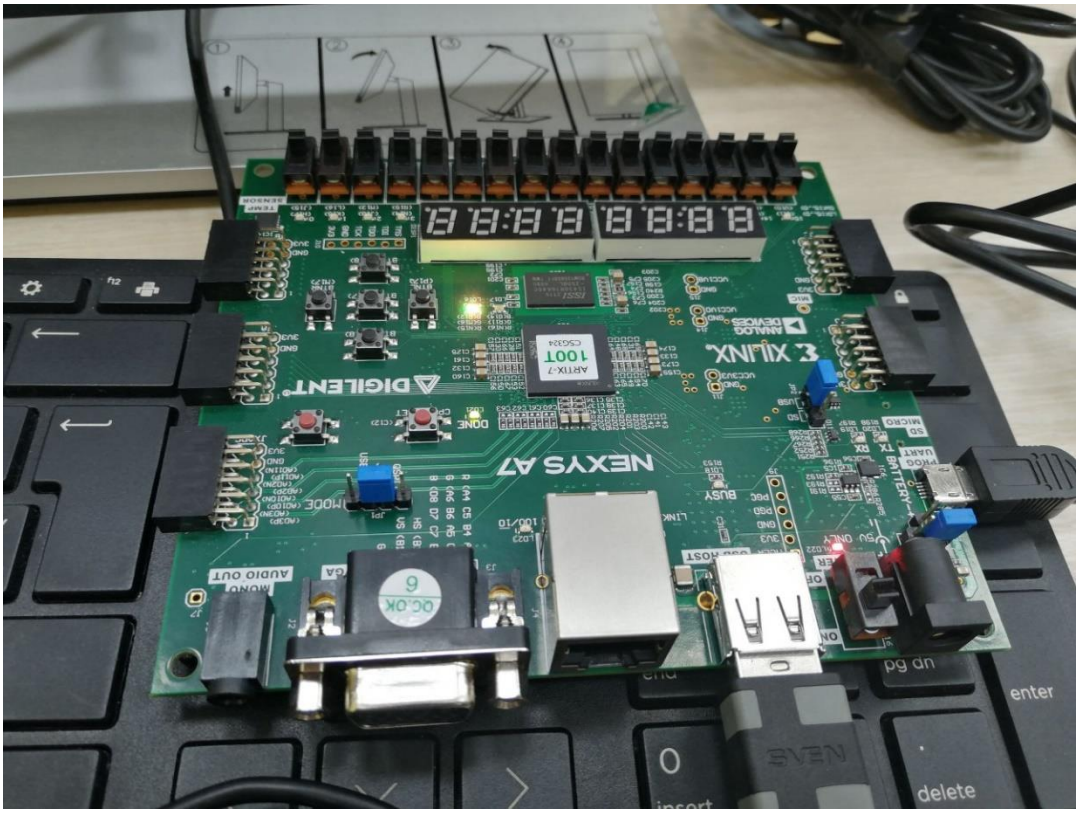

Рисунок 1.3 – уменьшение яркости

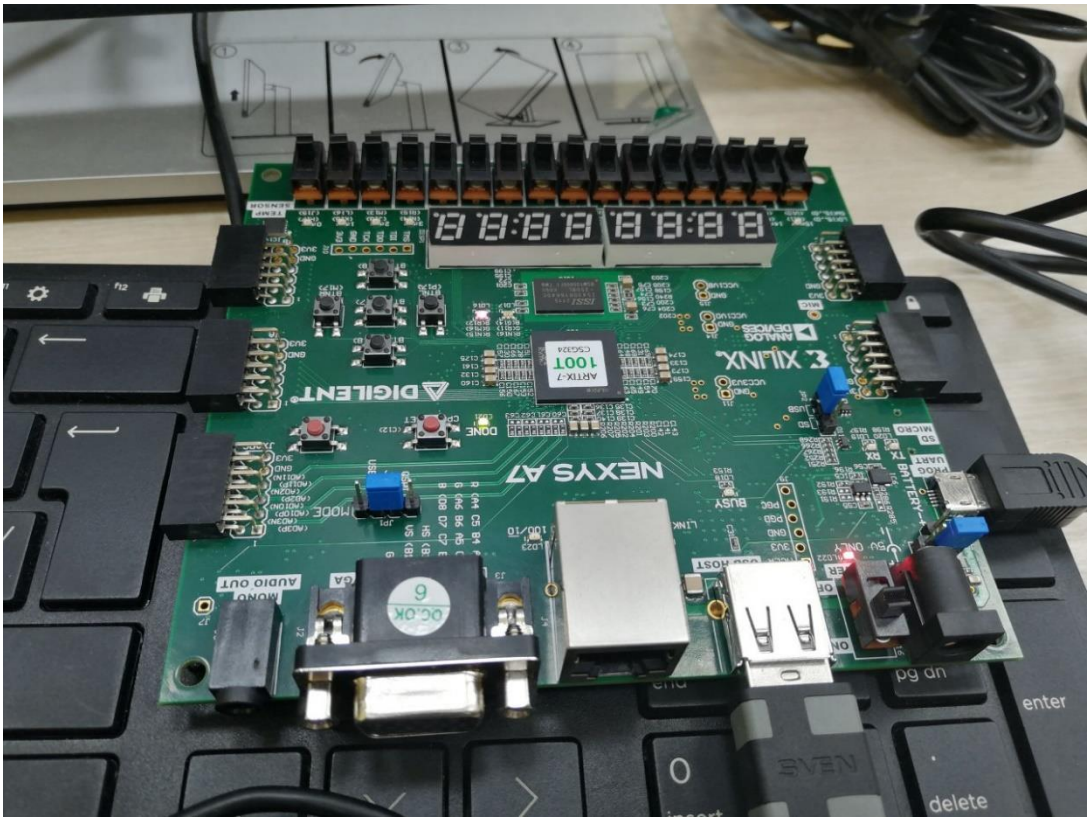

Рисунок 1.4 – уменьшение яркости

ЗАКЛЮЧЕНИЕ

В рамках данной лабораторной работы изучили принципы работы широтно-импульсного регулирования и ширитно-импульсной модуляции. Спроектировали устройство управления светодиодом по принципу широтно-импульсного модуляции с использованием алгоритма CORDIC.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Методические указания по ПР № 1 — URL: https://online-edu.mirea.ru/mod/resource/view.php?id=405132 (Дата обращения: 23.09.2022).

2. Методические указания по ПР № 2 — URL: https://online-edu.mirea.ru/mod/resource/view.php?id=409130 (Дата обращения: 23.09.2022).

3. 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).

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

5. Антик М.И. Дискретная математика [Электронный ресурс]: Учебное пособие / Антик М.И., Казанцева Л.В. — М.: МИРЭА — Российский технологический университет, 2018 — 1 электрон. опт. диск (CD-ROM).

6. Антик М.И. Математическая логика и программирование в логике [Электронный ресурс]: Учебное пособие / Антик М.И., Бражникова Е.В.— М.: МИРЭА – Российский технологический университет, 2018. — 1 электрон. опт. диск (CD-ROM).

7. Жемчужникова Т.Н. Конспект лекций по дисциплине «Архитектура вычислительных машин и систем» — URL: https://drive.google.com/file/d/12OAi2_axJ6mRr4hCbXs-mYs8Kfp4YEfj/view?usp=sharing (Дата обращения: 23.09.2022).

8. Антик М.И. Теория автоматов в проектировании цифровых схем [Электронный ресурс]: Учебное пособие / Антик М.И., Казанцева Л.В. — М.: МИРЭА – Российский технологический университет, 2020. — 1 электрон. опт. диск (CD-ROM).