|  |
| --- |
|  |
| 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 № 1**

«Изучение принципов широтно-импульсной модуляции» по дисциплине

«Схемотехника устройств компьютерных систем»

|  |  |
| --- | --- |
| Выполнил студент группы ИВБО-01-22 | Воробьев Д.М. |
| Принял ассистент кафедры ВТ | Люлява Д.В. |
| Практическая работа выполнена | « » 2024 г. |
| «Зачтено» | « » 2024 г. |

Москва 2024

**АННОТАЦИЯ**

2

Данная работа включает в себя 1 рисунок, 3 листинга. Количество страниц в работе — 13.

**СОДЕРЖАНИЕ**

1. [ОСНОВНОЙ РАЗДЕЛ 5](#_bookmark0)

1.1 [Модуль ps2\_keyboard 5](#_bookmark1)

* 1. [Модуль dynamic\_pwm\_controller 8](#_bookmark2)
  2. [Модуль верхнего уровня 9](#_bookmark2)

1.4 [Файл проектных ограничений 12](#_bookmark3)

1.5 [Тестирование на плате 13](#_bookmark4)

# ВВЕДЕНИЕ

# ОСНОВНОЙ РАЗДЕЛ

* 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. **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 1 Гц. 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 }];

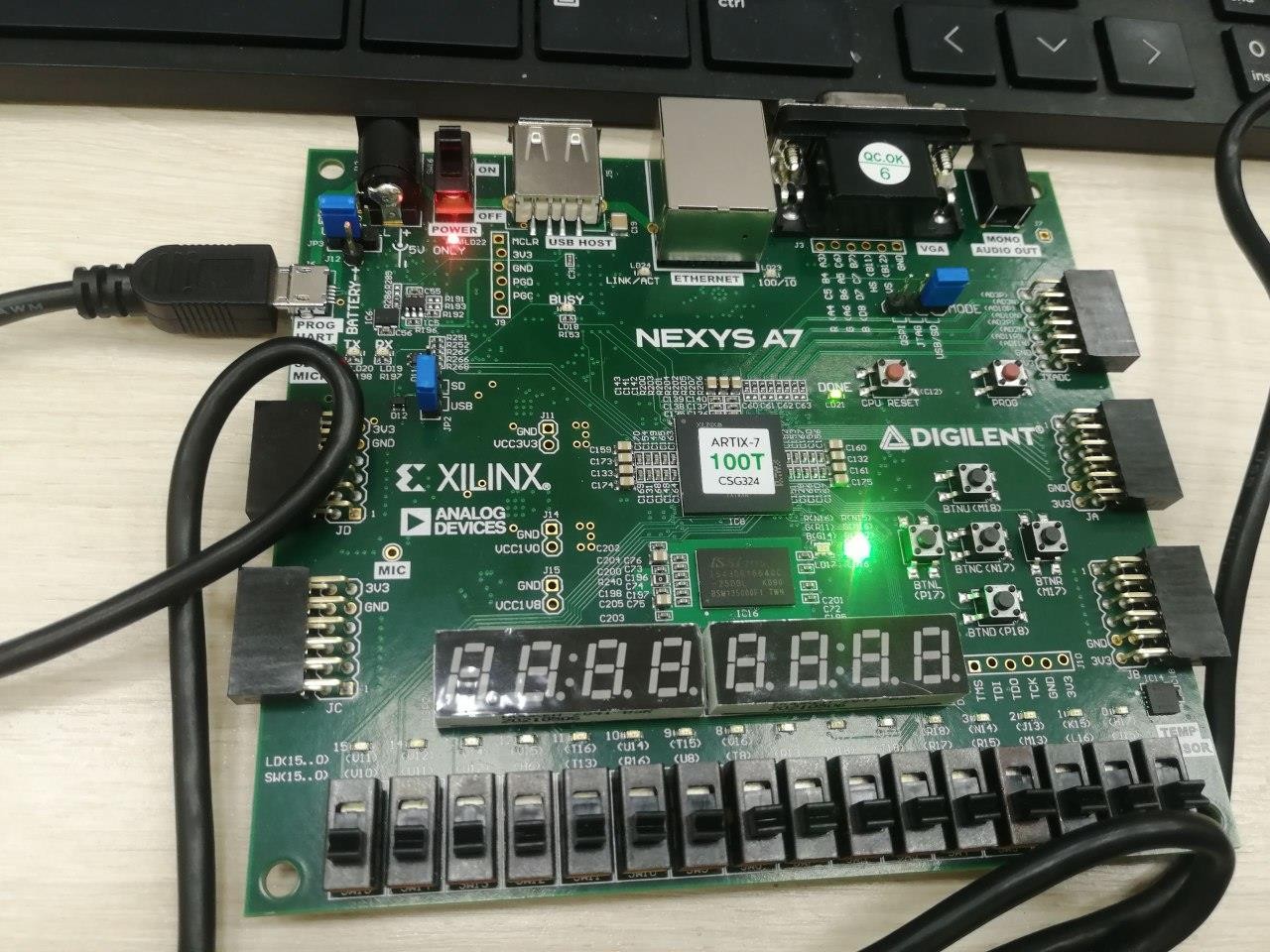
set\_property -dict { PACKAGE\_PIN M16 IOSTANDARD LVCMOS33 } [get\_ports { pwm\_out\_G }];

set\_property -dict { PACKAGE\_PIN N15 IOSTANDARD LVCMOS33 } [get\_ports { pwm\_out\_R }];

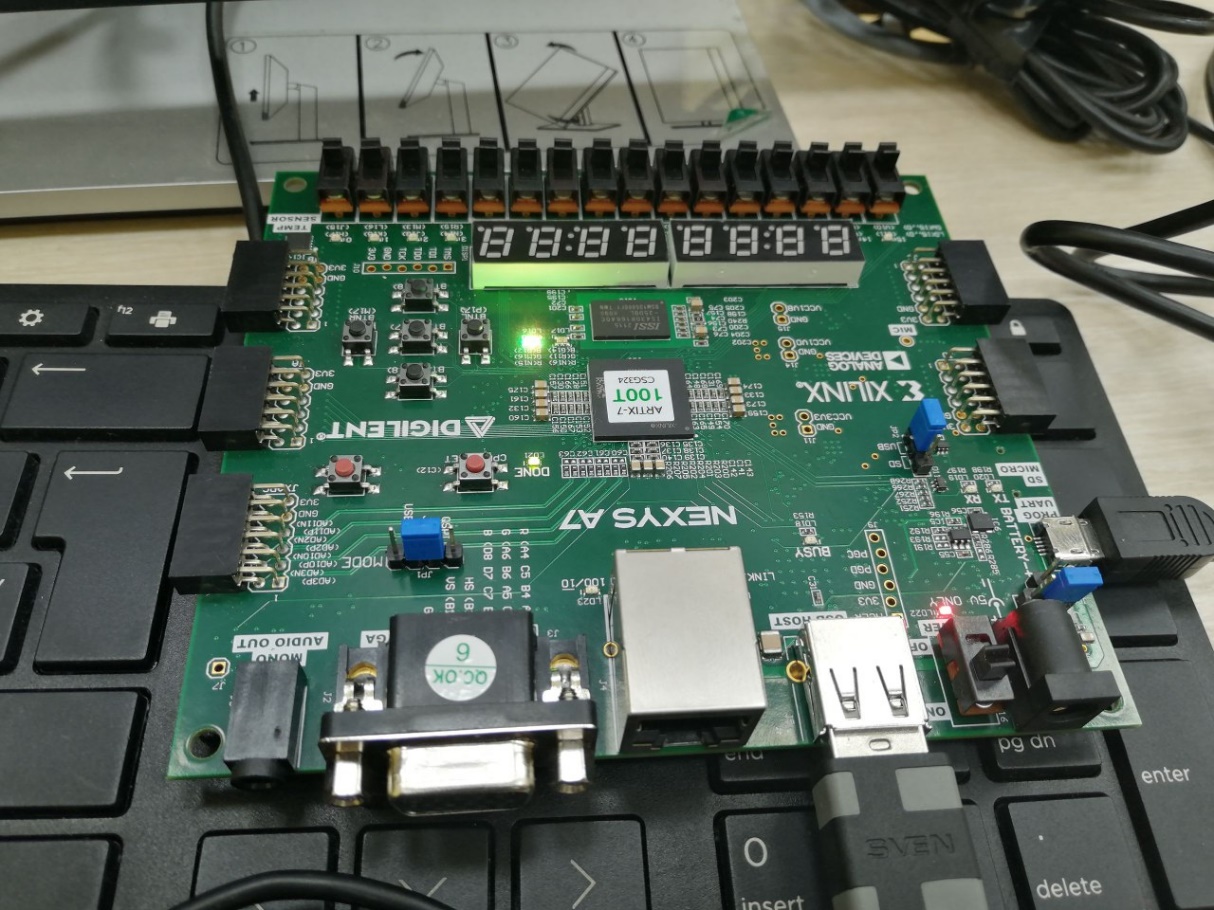
set\_property -dict { PACKAGE\_PIN F4 IOSTANDARD LVCMOS33 } [get\_ports { ps2\_clk }];

set\_property -dict { PACKAGE\_PIN B2 IOSTANDARD LVCMOS33 } [get\_ports { ps2\_data }];

**1.5 Тестирование на плате**

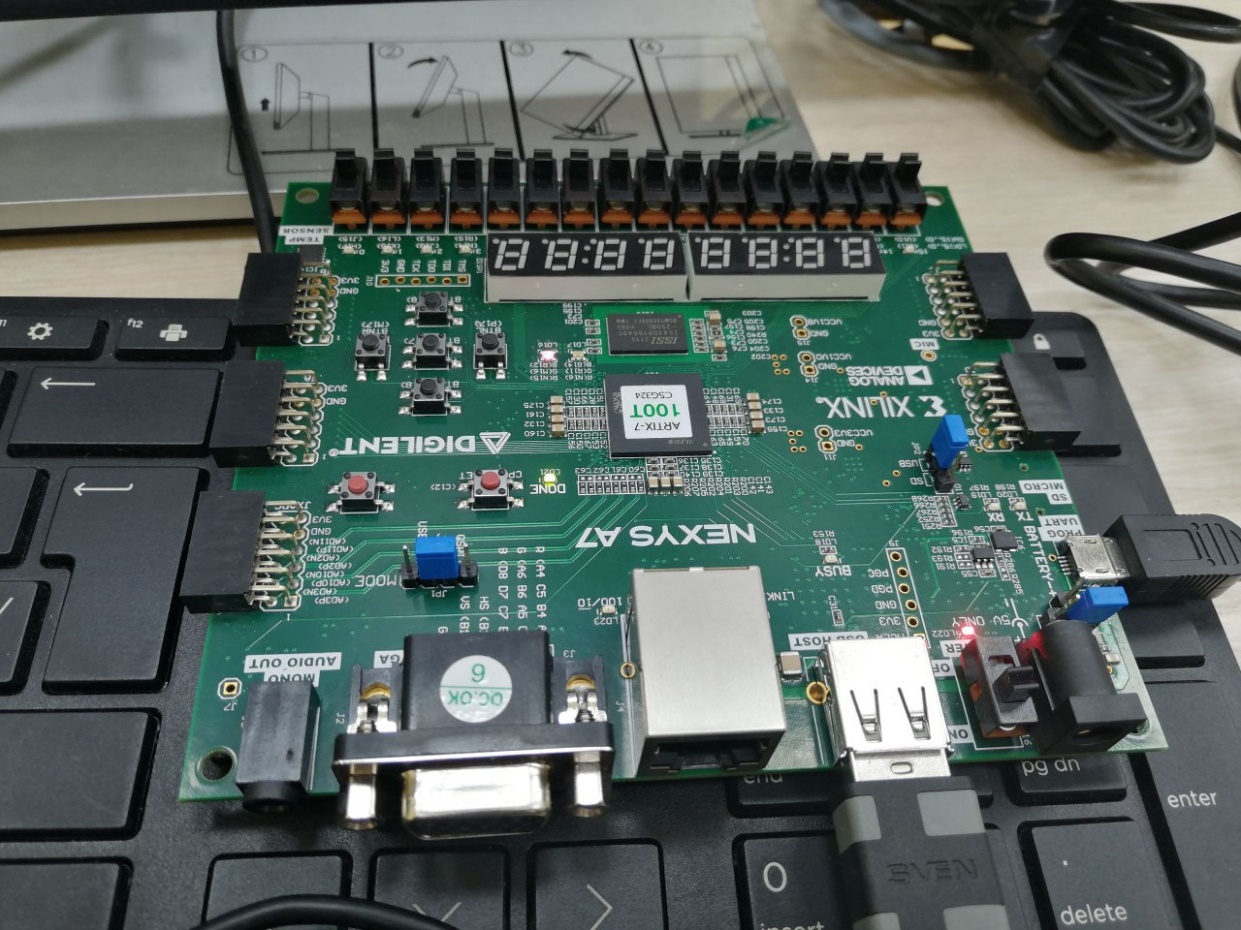
Протестируем работу проекта на плате (Рисунок 1.1).

**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).

1. Тарасов И.Е. ПЛИС Xilinx. Языки описания аппаратуры VHDL и Verilog, САПР, приемы проектирования. — М.: Горячая линия — Телеком, 2021. — 538 с.: ил.
2. 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).
3. 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).
4. Жемчужникова Т.Н. Конспект лекций по дисциплине «Архитектура вычислительных машин и систем» — URL: https://drive.google.com/file/d/12OAi2\_axJ6mRr4hCbXs-mYs8Kfp4YEfj/view?us p=sharing (Дата обращения: 23.09.2022).
5. 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).