АННОТАЦИЯ

Данная курсовая работа включает в себя 9 рисунков, 9 листингов, 7 приложений и 7 использованных источников.

Количество страниц в работе — 39.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире существует два основных способа отображения графической информации на экране: векторный и растровый. Растровый способ, в котором изображение представлено прямоугольной матрицей пикселей, стал наиболее распространенным на персональных компьютерах и других устройствах. Это связано с возможностью отображения сложных графиков и функций в доступной для восприятия форме.

Актуальность данной работы заключается в том, что генерация графиков трансцендентных функций, таких как синус, косинус и квадратный корень, позволяет лучше понять их поведение при изменении параметров. Это знание служит хорошей основой для изучения более сложных функций и методов их отображения.

Таким образом, можно определить тему курсовой работы: разработка генераторов трансцендентных функций и вывод графиков на экран монитора с использованием растрового способа формирования изображений.

Целью курсовой работы является изучение и разработка модулей для генерации трансцендентных функций и их визуализация на экране монитора.

В связи с поставленной целью, необходимо решить следующие задачи:

1. Провести анализ предметной области.
2. Разработать набор модулей, описывающих несколько генераторов трансцендентных функций. Варианты функций: sin, cos, квадратный корень. Варианты генерации: CORDIC, табличный, на основе любого из рядов.
3. Разработать набор модулей, описывающих драйвер для вывода изображения на монитор по протоколу VGA.
4. Разработать модуль верхнего уровня, реализующий управление для совместной работы всех модулей.
5. Провести верификацию.
6. Составить отчетную документацию по проделанной работе.

Объектом исследования данной курсовой работы является генерация и визуализация трансцендентных функций. Предметом исследования будет реализация различных методов генерации синуса, косинуса и квадратного корня, а также вывод полученных графиков на монитор с использованием протокола VGA.

Теоретическая часть курсовой работы основана на литературе, касающейся принципов генерации трансцендентных функций и особенностей растрового изображения.

Практическая часть написана с опорой на материалы по проектированию цифровой аппаратуры и разработке модулей для FPGA.

1 СИСТЕМНАЯ МОДЕЛЬ

Существует два способа реализации построения изображений на экране дисплея – векторный и растровый. Векторный способ основывается на отображении совокупности заготовленных фигур в определённом положении на экране. Объекты векторной графики описываются при помощи координат, параметров и атрибутов. Однако нас больше интересует растровый способ формирования изображения, так как именно его мы будем использовать. Растровое изображение представляет собой сетку пикселей, которые являются цветными точками (обычно прямоугольными) на мониторе и других отображающих устройствах. Важными характеристиками растрового изображения являются:

1. Размер: разрешение в нашей работе составляет 800(h_av) × 600(v_av) пикселей, что означает количество пикселей по ширине и высоте.
2. Количество используемых цветов: в нашей работе используется 8(3-битное значение) цветов.
3. Цветовое пространство (цветовая модель) — RGB.
4. Разрешение изображения — величина, определяющая количество точек (элементов растрового изображения) на единицу площади (или единицу длины).

Для вывода изображения мы используем интерфейс VGA. Чтобы понять, как с его помощью выводить необходимые графики и другие элементы, необходимо углубиться в устройство его работы. VGA (Video Graphics Array) — это стандарт передачи видео для аналоговых сигналов.VGA – это интерфейс, используемый для передачи сигналов.

Интерфейс VGA представляет собой 15-контактный разъём D-типа, разделённый на три строки, по пять контактов в каждой. Наиболее важными из них являются три сигнала цветового компонента RGB и два сигнала синхронизации сканирования — HSYNC и VSYNC.

Режим сканирования монитора VGA начинается с точки в верхнем левом углу экрана и сканируется слева направо от точки к точке. После сканирования линии электронный луч возвращается в исходное положение следующей строки на левой стороне экрана. В течение этого периода ЭЛТ перекрывает электронный луч. В конце каждой строки сигнал синхронизации строки используется для синхронизации: когда все строки сканируются, формируется кадр, сигнал синхронизации поля используется для синхронизации поля, и сканирование возвращается в верхний левый угол экрана, пока выполняется вертикальное гашение, и начинается следующий кадр. Наглядный пример работы представлен на Рисунке 1.1.

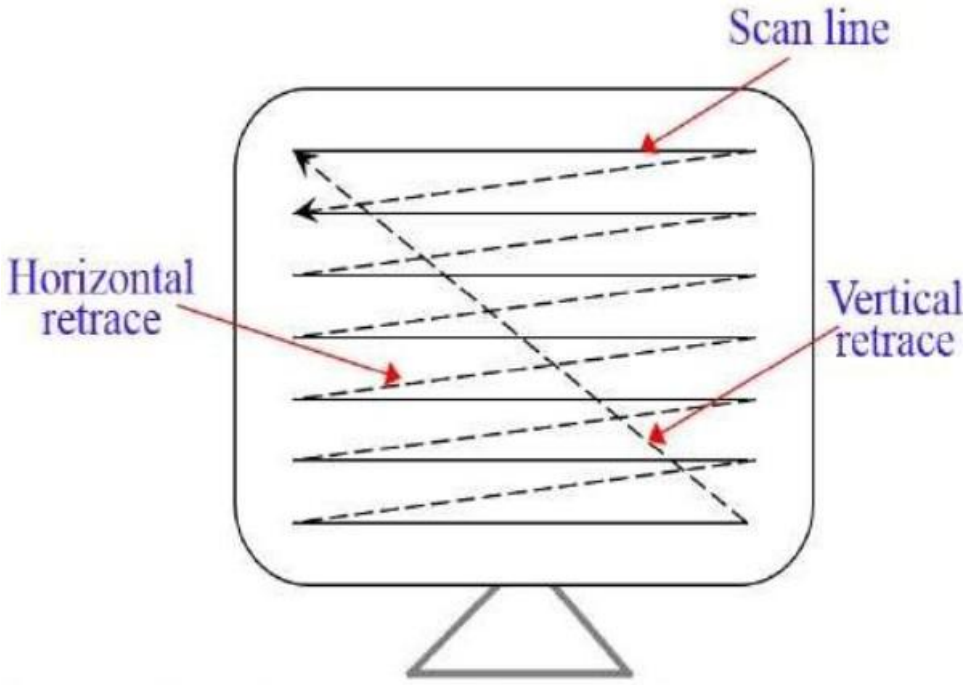

Рисунок 1.1 — Визуализация внутренней работы VGA

Для отображения данных на экране требуется последовательное подсвечивание пикселей, двигаясь слева направо и сверху вниз, либо оставлять их неактивными. Я выделил три подхода к реализации этой задачи.

Первый подход заключается в статическом формировании изображения. В этом случае можно заранее вычислить все пиксели, которые необходимо подсветить, и сохранить их координаты. Когда вертикальные и горизонтальные счётчики совпадают с этими координатами, соответствующий пиксель будет принимать заданный цвет. Однако данный метод требует

значительных затрат памяти, так как необходимо хранить большое количество координат. Кроме того, формирование массива координат требует времени, что не всегда оправдано.

Второй подход, который я нашёл в процессе изучения литературы, заключается в динамическом формировании и определении пикселей, которые должны быть подсвечены. В этом случае при каждом обновлении состояния системы пиксели вычисляются на лету, что позволяет избежать хранения большого объёма данных и сэкономить память.

1.1 Описание предметной области

Проект посвящен созданию устройства для визуализации трансцендентных функций. Трансцендентные функции, такие как синус, косинус и квадратный корень, имеют ключевое значение в математике и физике. Проект включает интерфейс, позволяющий пользователю наблюдать за изменениями графиков этих функций в зависимости от угла, с интерактивным управлением. Основная цель — создать интуитивно понятный способ изучения трансцендентных функций, демонстрируя их динамику и взаимосвязь.

1.2 Реализация системной модели

Для реализации системы визуализации трансцендентных функций был выбран язык программирования Python. Системная модель была разработана с учетом математических свойств трансцендентных функций, таких как синус, косинус и квадратный корень, а также интерактивного управления с помощью клавиатуры и отображения графиков на экране.

На вход система принимает сигналы от клавиш управления, которые позволяют пользователю изменять угол отображения функций. Результатом выполнения является визуализация трансцендентных функций, где

пользователь может наблюдать за изменениями графиков в реальном времени. На экране отображаются графики синуса, косинуса и квадратного корня, что позволяет лучше понять их поведение и взаимосвязь.

На рисунке 1.2 представлен вывод программы.

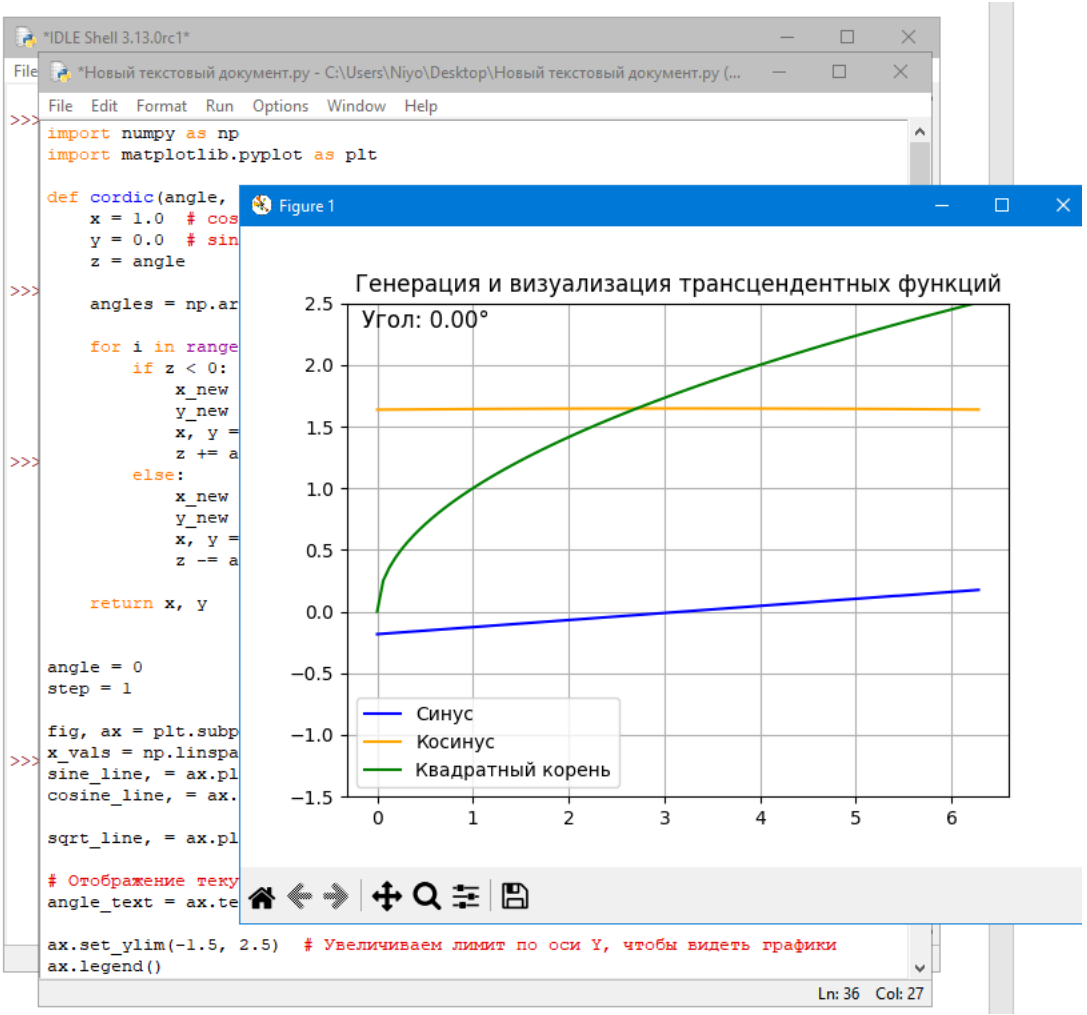

Рисунок 1.2 — Вывод системной модели

Программа разделена на несколько этапов:

1. Инициализация модулей и игровых объектов (птичка, трубы, фон).
2. Управление событиями игры (прыжок птички и сброс игры при завершении).
3. Генерация значений функций с использованием численных методов (например, метод Кордик для синуса и косинуса).
4. Вывод графиков на экран и отрисовка осей координат.
5. Обновление значений графиков в реальном времени в зависимости от угла.

6. Обработка пользовательского ввода для изменения параметров отображения функций.
7. Обновление экрана и пересчет значений графиков на каждом кадре.
8. Тестирование функциональности программы через визуализацию и взаимодействие с пользователем.

Эта модель корректно обрабатывает ввод и выводит графики на экран с соблюдением логики трансцендентных функций.

1.3 Верификация системной модели

В ходе тестирования программы были проведены проверки: корректность отображения графиков синуса, косинуса и квадратного корня, обновление значений функций при изменении угла, а также реакция программы на пользовательский ввод. Результат отображения графиков представлены на рисунке 1.3.

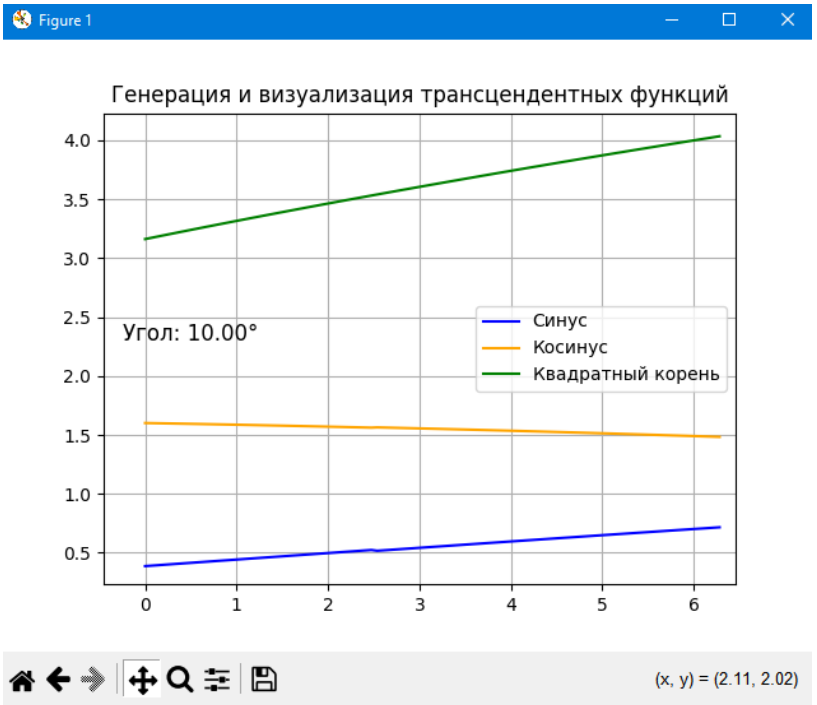

Рисунок 1.3 — Значения при изменении угла

2 RTL-МОДЕЛЬ УСТРОЙСТВА

На данном этапе устройство описывается на уровне соединений между регистрами. Основными вопросами, подлежащими разработке на данном этапе являются: структурная схема RTL-модели [11], описание модулей RTL-модели, верификация RTL-модели.

2.1 RTL – уровень проектирования

Структурная RTL-схема представлена на Рисунке 2.1.

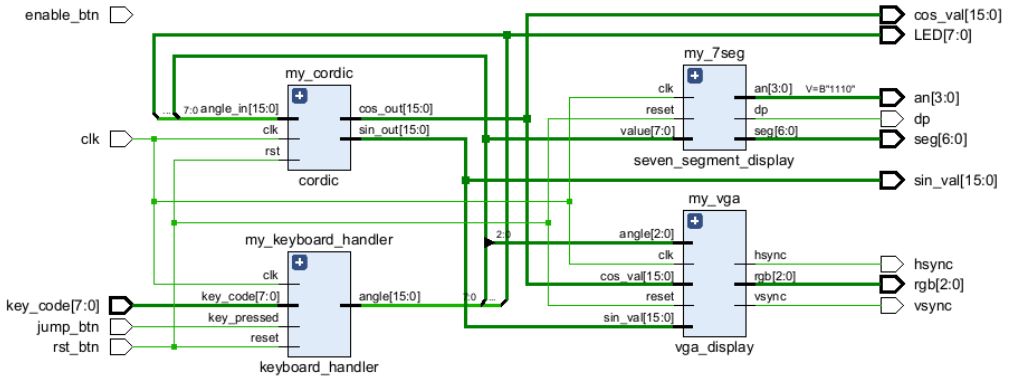

Рисунок 2.1 — RTL-схема

2.2 Описание модулей RTL-модели

Для реализации устройства для отображения графиков трансцендентных функций состоит из следующих модулей:

1. top_module — модуль верхнего уровня;
2. keyboard_handler — модуль обработки клавиатурного ввода;
3. cordic — модуль вычисления трансцендентных функций (sin, cos);
4. vga_display — модуль вывода графической информации на VGA.
5. Seven_segment_display – модуль управления семисегментным дисплеем.

Top_module - Модуль верхнего уровня, управляющий всеми остальными компонентами (как PS2_Manager, FB, VGA). Отвечает за обработку сигнала

клавиатуры, вычисление угловых функций (синус и косинус) и вывод графического результата на VGA-монитор. На вход принимает значения clk, reset, key_pressed, key_code и на выход передает значения hsync, vsync, rgb, sin_val, cos_val. Включает в себя истанцированные модули: keyboard_handler, cordic, vga_display.

Keyboard_handler- Этот модуль отвечает за обработку нажатий клавиш. Когда клавиша нажата, он преобразует её код в значение угла, которое используется для вычислений в модуле CORDIC. Модуль обрабатывает ввод с помощью конечного автомата, как в случае с обработкой PS2, только с меньшими функциями. Работает на каждом такте Posedge тактового сигнала clk. При поступлении сигнала reset значения угла становится равно 0. Использовал конструкцию case, которая определяет, какая клавиша именно была нажата. В зависимости от нажатой клавиши модуль изменяет значение угла. Например, символ 1 на клавиатуре увеличивает угол на 10, а символ 2 уменьшает.

Cordic - Модуль вычисляет синус и косинус угла, используя алгоритм CORDIC. Это ключевой модуль для генерации трансцендентных функций. Были созданы основные элементы модуля, такие как: iterations - определяет количество итераций, которые нужно выполнить для достижения нужной точности. В вашем случае используется 16 итераций, параметр к - масштабный коэффициент (CORDIC Gain). В моем коде это значение установлено как 16'd19898 (в фиксированной точке). Этот коэффициент учитывает общую длину вектора после нескольких итераций алгоритма. Значение близко к реальному коэффициенту ~0.607252935. Таблица углов angle_table - это массив фиксированных углов в радианах, представленных в фиксированной точке. Эти углы используются на каждой итерации для корректировки текущего угла (приближения угла к нулю). Таблица содержит арктангенсы степеней. Алгоритм кордик работает путем последовательного уменьшения остатка угла z(получает значение входного угла angle_in) с помощью шагов, которые уменьшают угол на известные значения (арктангенсы степеней

двойки). При этом векторы x и y(начальное приближение для синуса) модифицируются таким образом, чтобы они приближались к точному значению косинуса и синуса для входного угла.

Vga_display -управляет отображением цветных пикселей на экране VGA на основе входных значений синуса и косинуса, полученных от модуля CORDIC, а также угла, определяющего цвет. Он принимает входные сигналы clk, reset, sin_val, cos_val и angle. Внутри модуля генерируется тактовый сигнал VGA и управляется синхронизация по горизонтали и вертикали. На основании значений синуса и косинуса формируются цветовые каналы RGB. Красный цвет устанавливается, если старший бит синуса равен 1 и первый бит угла установлен, зелёный цвет - при аналогичных условиях для косинуса и второго бита угла, синий цвет - если синус превышает половину своего максимума и третий бит угла равен 1. Вне активной области синхронизации цвета пикселей устанавливаются в черный.

Модуль seven_segment_display отвечает за отображение значения угла на четырехразрядном семисегментном дисплее, что позволяет пользователю наблюдать за изменениями текущего угла, используемого для расчетов в модуле cordic. Входной сигнал value представляет собой младшие 8 бит значения угла, которые требуется отобразить на дисплее. Модуль принимает на вход сигналы clk, reset и value, а на выходе формирует сигналы seg, управляющие семью сегментами дисплея (CA-G), сигнал точки dp, а также сигналы an, активирующие один из четырех анодов дисплея. С помощью счетчика digit_select и конструкций case поочередно активируется каждый разряд дисплея, а значение угла преобразуется в соответствующие символы для сегментов CA-G. При поступлении сигнала reset дисплей очищается, и значения сегментов устанавливаются в неактивное состояние. Модуль предоставляет пользователю визуальную обратную связь для отслеживания текущего угла, используемого в расчетах.

2.3 Верификация RTL-модели

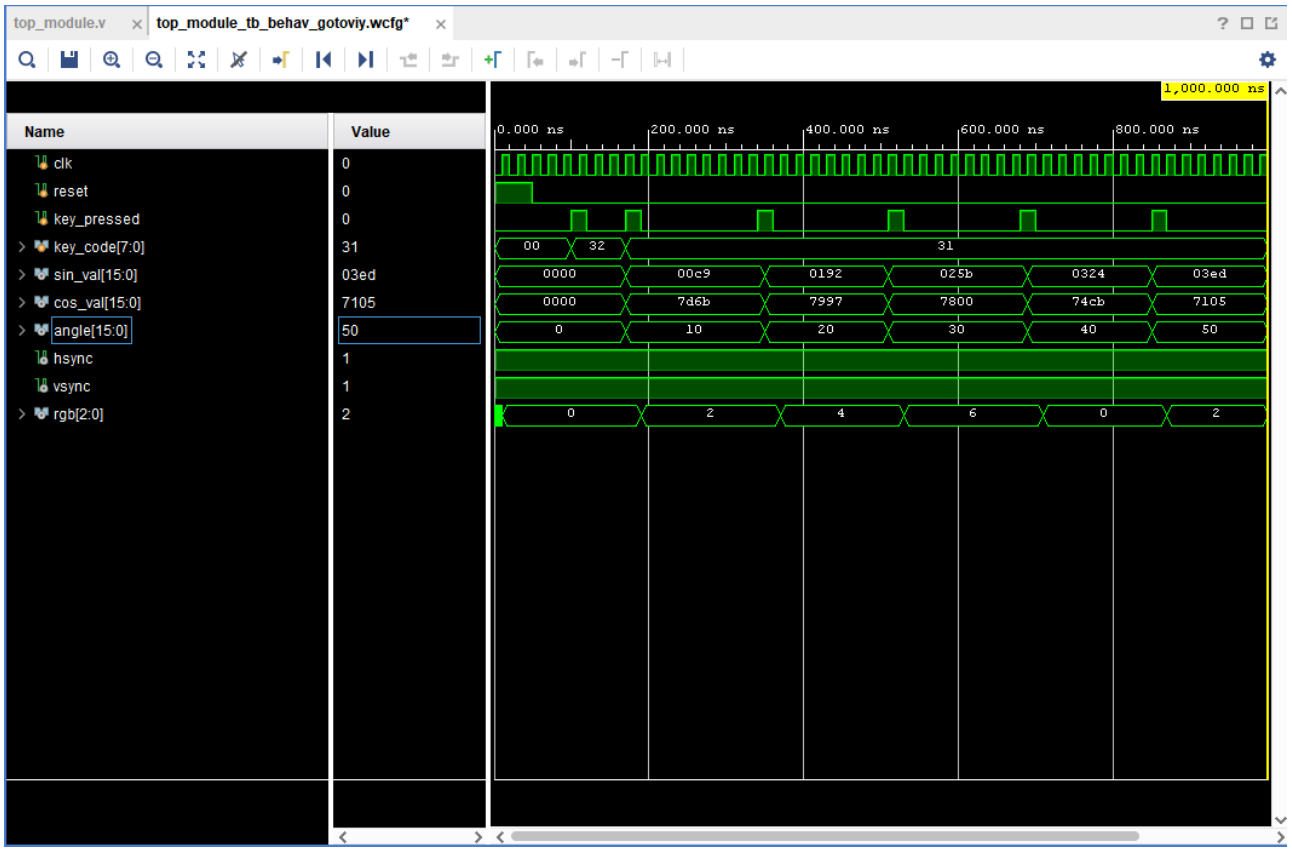

Рисунок 2.2 — Временная диаграмма

В Листинге 2.1 предоставлен код данного тестового модуля.

Листинг 2.1 — Код тестового модуля

```
module top_module_tb;

    reg clk;
    reg reset;
    reg key_pressed;
    reg [7:0] key_code;
    wire hsync;
    wire vsync;
    wire [2:0] rgb;
    wire signed [15:0] sin_val;
    wire signed [15:0] cos_val;
    wire [15:0] angle;

    // Генерация тактового сигнала: 50 МГц
    initial begin
        clk = 0;
        forever #10 clk = ~clk; // Инверсия каждые 10 нс
    end

    // Сценарий теста с различными изменениями угла
    initial begin
        reset = 1;
        key_pressed = 0;
        key_code = 8'h00;

        #50;
```

Продолжение листинга 2.1

```
        reset = 0;
        // Установка угла в 0
        #50;
        key_pressed = 1;
        key_code = 8'h32; // Уменьшение угла, чтобы проверить начало с 0
        #20 key_pressed = 0;

        // Увеличение угла на 10 градусов пять раз
        repeat (5) begin
            #50;
            key_pressed = 1;
            key_code = 8'h31; // Увеличение угла на 10
            #20 key_pressed = 0;

            #100; // Ожидание для обновления значений
        end

        // Дополнительные случайные нажатия для проверки вариативности
        repeat (10) begin
            #50;
            key_pressed = 1;

            // Случайный выбор между увеличением и уменьшением угла
            if ($random % 2 == 0) begin
                key_code = 8'h31; // Увеличение угла на 10
            end else begin
                key_code = 8'h32; // Уменьшение угла на 10
            end

            #20;
            key_pressed = 0;
            #200;
        end

        #50000;
        $finish;
    end

    // Подключение верхнего модуля (DUT)
    top_module uut (
        .clk(clk),
        .rst_btn(reset),
        .jump_btn(key_pressed),
        .enable_btn(1'b1),
        .key_code(key_code),
        .hsync(hsync),
        .vsync(vsync),
        .rgb(rgb),
        .sin_val(sin_val),
        .cos_val(cos_val),
        .angle(angle)
    );

    // Отслеживание значений sin, cos и угла
    initial begin
        $monitor("Time: %0t | clk: %b | reset: %b | key_pressed: %b | key_code:
%h | angle: %d | sin_val: %d | cos_val: %d | hsync: %b | vsync: %b | rgb: %b",
                 $time, clk, reset, key_pressed, key_code, angle, sin_val,
cos_val, hsync, vsync, rgb);
    end
```

Продолжение листинга 2.1

```
endmodule
```

Для начала распишем каждый сигнал и что он обозначает:

1. Clk(тактовый сигнал) - Это основной тактовый сигнал, который синхронизирует все операции в коде. Все остальные сигналы и процессы в системе будут реагировать на фронты этого сигнала.
2. Reset - Сигнал сброса, который используется для инициализации системы в известное состояние. Когда reset установлен в 1, все значения регистров, включая состояние вывода, сбрасываются, и система начинает работу с начальных условий.
3. key_pressed - Бит, указывающий, нажата ли клавиша на клавиатуре. Если key_pressed установлен в 1, это означает, что клавиша была нажата, и система может выполнять определённые действия, такие как обновление key_code
4. key_code(7:0) - Код, соответствующий нажатой клавише.
5. Sin_val - Значение синусоидальной функции, используемое для вычислений цветовых значений. Это значение используется в формуле для вычисления цвета, отображаемого на экране. Изменения в этом значении напрямую влияют на выходной цвет rgb.
6. Cos_val - Значение косинусоидальной функции. значение также участвует в вычислениях цвета, влияя на то, как именно цвет будет выглядеть на экране.
7. Angle - Угол, который используется для определения, как должны изменяться sin_val и cos_val. Изменение угла может вызвать изменение значений sin_val и cos_val, что, в свою очередь, изменит отображаемый цвет.
8. Hsync - Горизонтальная синхронизация, сигнал, который сообщает дисплею, что линия отрисовки завершена.
9. Vsync - Вертикальная синхронизация, сигнал, который указывает на завершение одного кадра.

10. rgb (2:0) - Значение цвета, которое будет отображаться на экране. сигнал управляет цветом, который отображается на экране. Значения rgb зависят от sin_val, cos_val и, возможно, angle, что позволяет динамически изменять цвет в зависимости от вводимых данных.

Сигнал clk управляет циклом обработки в системе, синхронизируя изменение всех других сигналов на своем фронте. Когда reset активен, все регистры сбрасываются, и система инициализируется, устанавливая rgb в стандартное значение (например, 000). При нажатии клавиши (key_pressed = 1) считывается key_code, что может изменить угол (angle), который затем используется для расчета sin_val и cos_val. Эти значения влияют на вывод цвета через rgb. Сигналы hsync и vsync управляют синхронизацией отображения, при этом hsync запускает отрисовку новой строки, а vsync сигнализирует о начале нового кадра. Цвет rgb обновляется в соответствии с последними вычисленными значениями sin_val и cos_val, которые зависят от нажатых клавиш и текущего угла.

3 РАЗМЕЩЕНИЕ НА ПЛИС

3.1 Основные подходы к размещению

Размещение на ПЛИС (программируемых логических интегральных схемах) включает в себя несколько ключевых этапов:

1. Синтез аппаратной схемы: Этот этап включает в себя создание логической схемы, которая определяет функциональность и поведение ПЛИС. В отличие от традиционных микросхем, которые имеют фиксированную функциональность, ПЛИС можно программировать для выполнения различных задач.
2. Размещение и маршрутизация: после синтеза аппаратной схемы необходимо разместить логические элементы на ПЛИС и настроить соединения между ними. Этот процесс называется размещением и маршрутизацией. Размещение определяет физическое расположение логических элементов на ПЛИС, а маршрутизация устанавливает соединения между этими элементами
3. Генерация конфигурационного файла: после размещения и маршрутизации необходимо сгенерировать конфигурационный файл, который содержит информацию о том, как настроить ПЛИС для работы с созданной аппаратной схемой. Конфигурационный файл может быть сгенерирован с использованием специальных инструментов, предоставляемых производителем ПЛИС.
4. Загрузка конфигурационного файла на ПЛИС: последний этап программирования ПЛИС – загрузка конфигурационного файла на ПЛИС. Это может быть выполнено с помощью специального программатора, который подключается к ПЛИС и передает конфигурационный файл. После загрузки конфигурационного файла ПЛИС будет настроена для работы с созданной аппаратной схемой.

Результаты размещения согласно основным метрикам представлены на Рисунке 3.1

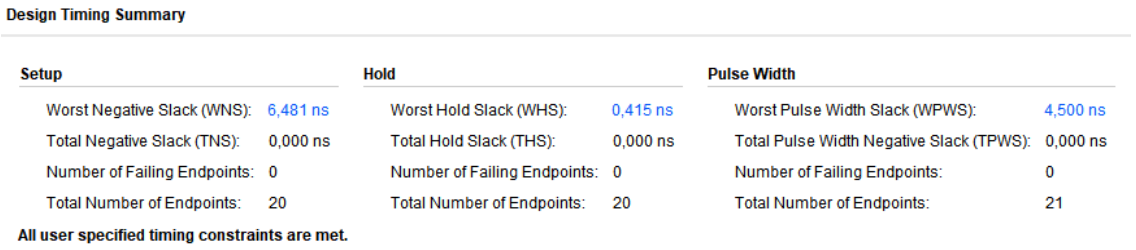

Рисунок 3.1 — показатели Slack

3.2 Тестирование работы устройства на ПЛИС

В Листинге 3.1 представлен набор проектных ограничений для данного проекта.

Листинг 3.1 — Набор ограничений

```
# Тактовый сигнал
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];
# Кнопка сброса
set_property -dict {PACKAGE_PIN P17 IOSTANDARD LVCMOS33} [get_ports rst_btn];
# Кнопка jump_btn
set_property -dict {PACKAGE_PIN N17 IOSTANDARD LVCMOS33} [get_ports jump_btn];
# Кнопка enable_btn
set_property  -dict  {PACKAGE_PIN  M18  IOSTANDARD  LVCMOS33}  [get_ports
enable_btn];
# key_code
set_property -dict { PACKAGE_PIN D14    IOSTANDARD LVCMOS33 } [get_ports {
key_code[0] }];
set_property  -dict  {PACKAGE_PIN  F16  IOSTANDARD  LVCMOS33}  [get_ports
{key_code[1]}];
set_property  -dict  {PACKAGE_PIN  B2  IOSTANDARD  LVCMOS33}  [get_ports
{key_code[2]}];
set_property  -dict  {PACKAGE_PIN  B1  IOSTANDARD  LVCMOS33}  [get_ports
{key_code[3]}];
set_property  -dict  {PACKAGE_PIN  C1  IOSTANDARD  LVCMOS33}  [get_ports
{key_code[4]}];
set_property  -dict  {PACKAGE_PIN  C2  IOSTANDARD  LVCMOS33}  [get_ports
{key_code[5]}];
set_property -dict { PACKAGE_PIN G13    IOSTANDARD LVCMOS33 } [get_ports {
key_code[6] }];
set_property -dict { PACKAGE_PIN D2    IOSTANDARD LVCMOS33 } [get_ports {
key_code[7] }];

# VGA сигналы
set_property -dict {PACKAGE_PIN A3 IOSTANDARD LVCMOS33} [get_ports {rgb[2]}];
# Красный канал VGA
set_property -dict {PACKAGE_PIN B4 IOSTANDARD LVCMOS33} [get_ports {rgb[1]}];
```

Продолжение листинга 3.1

```
  # Зеленый канал VGA
set_property -dict {PACKAGE_PIN C5 IOSTANDARD LVCMOS33} [get_ports {rgb[0]}];
# Синий канал VGA
set_property -dict {PACKAGE_PIN B11 IOSTANDARD LVCMOS33} [get_ports hsync];
set_property -dict {PACKAGE_PIN B12 IOSTANDARD LVCMOS33} [get_ports vsync];

# Сегменты семисегментного дисплея
set_property -dict {PACKAGE_PIN T10 IOSTANDARD LVCMOS33} [get_ports {seg[0]}];
# Сегмент A
set_property -dict {PACKAGE_PIN R10 IOSTANDARD LVCMOS33} [get_ports {seg[1]}];
# Сегмент B
set_property -dict {PACKAGE_PIN K16 IOSTANDARD LVCMOS33} [get_ports {seg[2]}];
# Сегмент C
set_property -dict {PACKAGE_PIN K13 IOSTANDARD LVCMOS33} [get_ports {seg[3]}];
# Сегмент D
set_property -dict {PACKAGE_PIN P15 IOSTANDARD LVCMOS33} [get_ports {seg[4]}];
# Сегмент E
set_property -dict {PACKAGE_PIN T11 IOSTANDARD LVCMOS33} [get_ports {seg[5]}];
# Сегмент F
set_property -dict {PACKAGE_PIN L18 IOSTANDARD LVCMOS33} [get_ports {seg[6]}];
# Сегмент G
set_property -dict {PACKAGE_PIN H15 IOSTANDARD LVCMOS33} [get_ports dp];

# Выводы для 7-сегментного дисплея
set_property -dict {PACKAGE_PIN J17 IOSTANDARD LVCMOS33} [get_ports {an[0]}];
set_property -dict {PACKAGE_PIN J18 IOSTANDARD LVCMOS33} [get_ports {an[1]}];
set_property -dict {PACKAGE_PIN T9 IOSTANDARD LVCMOS33} [get_ports {an[2]}];
set_property -dict {PACKAGE_PIN J14 IOSTANDARD LVCMOS33} [get_ports {an[3]}];
# Светодиоды
set_property -dict {PACKAGE_PIN H17 IOSTANDARD LVCMOS33} [get_ports {LED[0]}];
set_property -dict {PACKAGE_PIN K15 IOSTANDARD LVCMOS33} [get_ports {LED[1]}];
set_property -dict {PACKAGE_PIN J13 IOSTANDARD LVCMOS33} [get_ports {LED[2]}];
set_property -dict {PACKAGE_PIN N14 IOSTANDARD LVCMOS33} [get_ports {LED[3]}];
set_property -dict {PACKAGE_PIN R18 IOSTANDARD LVCMOS33} [get_ports {LED[4]}];
set_property -dict {PACKAGE_PIN V17 IOSTANDARD LVCMOS33} [get_ports {LED[5]}];
set_property -dict {PACKAGE_PIN U17 IOSTANDARD LVCMOS33} [get_ports {LED[6]}];
set_property -dict {PACKAGE_PIN U16 IOSTANDARD LVCMOS33} [get_ports {LED[7]}];
```

Были подключены тактовый сигнал платы, тактовый сигнал клавиатуры, вход передаваемых данных с клавиатуры и VGA

3.3 Вариант размещения

На Рисунке 3.2 представлен вариант размещения на ПЛИС, сгенерированный Vivado.

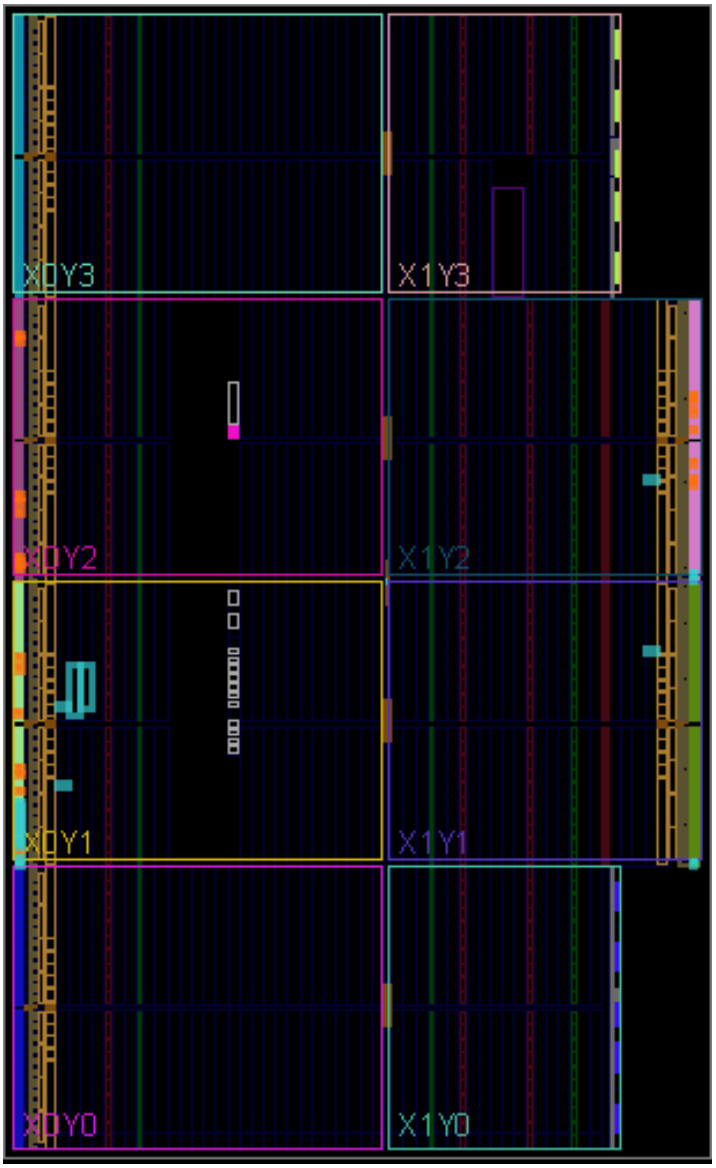

Рисунок 3.2 — Вариант размещения на ПЛИС

Потребляемая мощность представлена на Рисунке 3.3.

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.116 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**25,5°C**
Thermal Margin:	59,5°C (12,9 W)
Effective ϑJA:	4,6°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

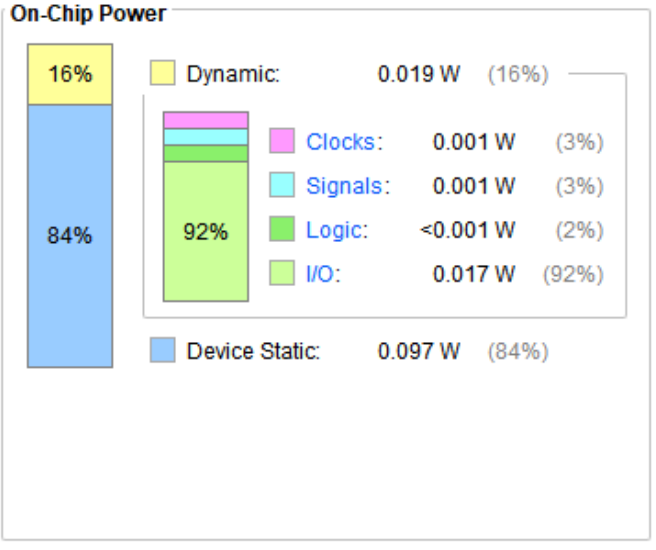

Рисунок 3.3 — Показатели потребляемой мощности

4 ТЕСТИРОВАНИЕ

Для тестирования всего проекта был создан файл на языке SystemVerilog — test_Top. Код представлен в Приложении Е.

В данном модуле генерируется код различных клавиш и отправляется в основной модуль. На рисунке 4.1 отображено изменение сигнала во времени. Данная временная диаграмма демонстрирует работу тестового модуля.

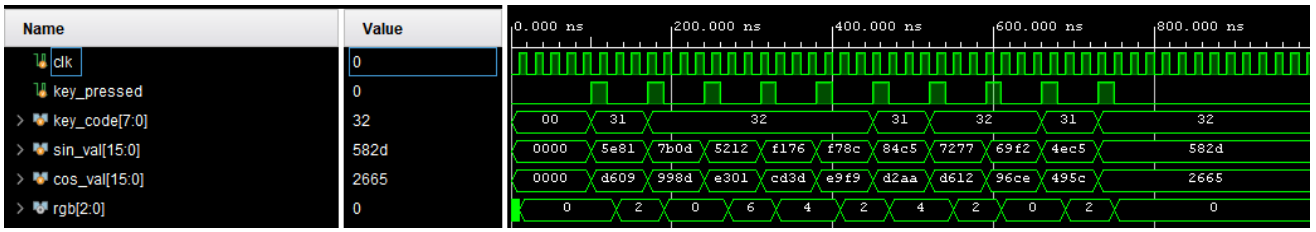

Рисунок 4.1 — Временная диаграмма

ЗАКЛЮЧЕНИЕ

В результате выполнения данной курсовой работы была исследована предметная область проекта, подготовлена и изучена теоретическая база, необходимая при создании устройства.

Было спроектировано и описано на языке описания аппаратуры Verilog устройство, которое способно принимать десятичные числа, а также команды для работы с ними.

Работа данного устройства была протестирована с помощью программных средств САПР. Также произведено тестирование прототипа с помощью ПЛИС. Оба тестирования прошли успешно.

В процессе выполнения курсовой работы были получены и закреплены знания в области проектирования цифровых устройств. Кроме того, были приобретены практические навыки по запуску и подключению компонентов устройства к ПЛИС. Важной частью работы стало исследование взаимодействия с изображением при использовании протокола передачи данных VGA.

Руководство пользователя прикреплено в Приложении Ж.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

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. 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).

4. Авдеев, В.А. Периферийные устройства: интерфейсы, схемотехника, программирование / В.А. Авдеев. - М.: ДМК, 2016. - 848 с.

5. Волонович, Г.И. Схемотехника аналоговых и аналогово-цифровых электронных устройств / Г.И. Волонович. - М.: ДМК, 2015. - 528 с.

6. Вельтмандер, П.В. Основные алгоритмы компьютерной графики.: ДМК, 2017. - 926 с.

7. Соловьев, В.В. Основы языка проектирования цифровой аппаратуры Verilog. - М.: Горячая линия - Телеком, 2015. — 206 с.: ил.

ПРИЛОЖЕНИЯ

Приложение А — Модуль верхнего уровня

Приложение Б — Модуль cordic

Приложение В — Модуль VGA

Приложение Г — Модуль Обработки клавиатурного ввода

Приложение Д — Семисегментный индикатор

Приложение Е — Тестовый модуль

Приложение Ж — Руководство пользователя

Приложение А

Модуль верхнего уровня

Листинг А.1 – Модуль верхнего уровня

```
module top_module(
    input wire clk,                 // Тактовый сигнал
    input wire reset,               // Сигнал сброса
    input wire jump_btn,         // Сигнал нажатия клавиши(key_pressed)
    input wire btn_enable,
    input wire [7:0] key_code,      // Код нажатой клавиши
    output wire hsync,              // Горизонтальная синхронизация для VGA
    output wire vsync,              // Вертикальная синхронизация для VGA
    output wire [2:0] rgb           // Цвет пикселя для VGA
);

    // Внутренние сигналы
    wire [15:0] angle;              // Угол для вычисления CORDIC
    wire signed [15:0] sin_val;     // Результат синуса от CORDIC
    wire signed [15:0] cos_val;     // Результат косинуса от CORDIC
    // Выходы для 7-сегментного дисплея
    output wire [6:0] seg,          // Сегменты CA-G
    output wire dp,                 // Точка дисплея
    output wire [3:0] an,           // Аноды

    // Выходы для светодиодов
    output wire [7:0] LED,          // Светодиоды

    // Экспорт значений sin, cos и угла
    output wire signed [15:0] sin_val,  // Результат синуса
    output wire signed [15:0] cos_val,  // Результат косинуса
    output wire [15:0] angle

    cordic my_cordic (
        .clk(clk),
        .rst(reset),
        .angle_in(angle), // Передаем угол от клавиатуры
        .sin_out(sin_val),
        .cos_out(cos_val)
    );
    keyboard_handler my_keyboard_handler (
        .clk(clk),
        .reset(reset),
        .key_pressed(key_pressed),  // Подключаем сигнал нажатия клавиши
        .key_code(key_code),        // Подключаем код нажатой клавиши
        .angle(angle)               // Подключаем выход угла
    );

    // Инстанцирование модуля VGA
    vga_display my_vga (
    .clk(clk),
    .reset(reset),
    .sin_val(sin_val),
    .cos_val(cos_val),
    .angle(angle[2:0]), // Передаем только младшие 3 бита
    .rgb(rgb),
    .hsync(hsync),
    .vsync(vsync)
    );

Endmodule
```

Приложение Б

Модуль cordic

Листинг Б.1 – Модуль cordic

```
module cordic (
    input wire clk,                     // Тактовый сигнал
    input wire rst,                     // Сигнал сброса
    input wire signed [15:0] angle_in, // Входной угол в фиксированной точке
    output reg signed [15:0] sin_out,  // Выход синуса
    output reg signed [15:0] cos_out   // Выход косинуса
);

    // Параметры
    parameter ITERATIONS = 16;

    // Масштабный коэффициент (K = 0.607252935 в фиксированной точке)
    parameter signed [15:0] K = 16'd19898;

    // Таблица углов (в фиксированной точке)
    reg signed [15:0] angle_table [0:15];

    // Внутренние регистры для хранения текущих значений x, y, z
    reg signed [15:0] x, y, z;

    integer i;

    // Инициализация таблицы углов
    initial begin
        angle_table[0]  = 16'd11520;  // arctan(2^0) в фиксированной точке
        angle_table[1]  = 16'd6803;   // arctan(2^-1)
        angle_table[2]  = 16'd3596;   // arctan(2^-2)
        angle_table[3]  = 16'd1824;   // arctan(2^-3)
        angle_table[4]  = 16'd916;    // arctan(2^-4)
        angle_table[5]  = 16'd458;    // arctan(2^-5)
        angle_table[6]  = 16'd229;    // arctan(2^-6)
        angle_table[7]  = 16'd114;    // arctan(2^-7)
        angle_table[8]  = 16'd57;     // arctan(2^-8)
        angle_table[9]  = 16'd28;     // arctan(2^-9)
        angle_table[10] = 16'd14;     // arctan(2^-10)
        angle_table[11] = 16'd7;      // arctan(2^-11)
        angle_table[12] = 16'd3;      // arctan(2^-12)
        angle_table[13] = 16'd1;      // arctan(2^-13)
        angle_table[14] = 16'd1;      // arctan(2^-14)
        angle_table[15] = 16'd0;      // arctan(2^-15)
    end

    // Основной CORDIC алгоритм
    always @(posedge clk) begin
        if (rst) begin
            // Инициализация начальных значений при сбросе
            x <= K;         // Начальное значение для косинуса (масштабированный
коэффициент)
            y <= 16'd0;      // Начальное значение для синуса
            z <= angle_in;   // Входной угол
        end else begin
            // Основной CORDIC алгоритм
            for (i = 0; i < ITERATIONS; i = i + 1) begin
                if (z >= 0) begin
                    x = x - (y >>> i);
                    y = y + (x >>> i);
```

Продолжение листинга Б.1

```
                    z = z - angle_table[i];
                end else begin
                    x = x + (y >>> i);
                    y = y - (x >>> i);
                    z = z + angle_table[i];
                end
            end

            // Присваиваем выходам синуса и косинуса
            sin_out <= y;
            cos_out <= x;
        end
    end
endmodule
```

Приложение В

Модуль VGA

Листинг В.1 – Модуль VGA

```
`timescale 1ns / 1ps
module vga_display(
    input clk,
    input reset,
    input [15:0] sin_val, // Значение синуса от CORDIC
    input [15:0] cos_val, // Значение косинуса от CORDIC
    input [2:0] angle,    // Угол для изменения цвета
    output reg [2:0] rgb,
    output hsync,
    output vsync
);

parameter h_av = 800, h_fp = 56, h_sp = 120, h_bp = 64;
parameter v_av = 600, v_fp = 37, v_sp = 6, v_bp = 23;

reg [10:0] hc, vc;
reg vgaClk;

// Генерация тактового сигнала VGA
always @(posedge clk) begin
    vgaClk <= ~vgaClk;
end

// Синхронизация по горизонтали
assign hsync = (hc >= (h_av + h_fp) && hc < (h_av + h_fp + h_sp)) ? 0 : 1;

// Счётчик горизонтальной позиции
always @(posedge vgaClk or posedge reset) begin
    if (reset)
        hc <= 0;
    else if (hc < (h_av + h_fp + h_sp + h_bp - 1))
        hc <= hc + 1;
    else
        hc <= 0;
end

// Синхронизация по вертикали
assign vsync = (vc >= (v_av + v_fp) && vc < (v_av + v_fp + v_sp)) ? 0 : 1;

// Счётчик вертикальной позиции
always @(posedge vgaClk or posedge reset) begin
    if (reset)
        vc <= 0;
    else if (hc == (h_av + h_fp + h_sp + h_bp - 1)) begin
        if (vc < (v_av + v_fp + v_sp + v_bp - 1))
            vc <= vc + 1;
        else
            vc <= 0;
end
end

// Управление цветом в зависимости от синуса, косинуса и угла
always @(posedge clk) begin
    if (hc < h_av && vc < v_av) begin
        // Используем синус и косинус для RGB
        rgb[0] <= sin_val[15] ? 1 : 0; // Красный канал
```

Продолжение Листинга В.1

```
        rgb[1] <= cos_val[15] ? 1 : 0; // Зеленый канал
        rgb[2] <= (sin_val[15] & cos_val[15]) ? 1 : 0; // Синий канал

        // Меняем цвет RGB в зависимости от значения угла
        rgb[0] <= angle[0]; // Красный канал зависит от младшего бита угла
        rgb[1] <= angle[1]; // Зеленый канал зависит от второго бита угла
        rgb[2] <= angle[2]; // Синий канал зависит от третьего бита угла
    end else begin
        rgb <= 3'b000; // Черный цвет вне активной области
    end
end

endmodule
```

Приложение Г

Модуль обработки клавиатурного ввода

Листинг Г.1 – Модуль keyboard_handler

```
`timescale 1ns / 1ps
module keyboard_handler(
    input wire clk,
    input wire reset,
    input wire key_pressed,
    input wire [7:0] key_code,
    output reg [15:0] angle
);

    always @(posedge clk or posedge reset) begin
        if (reset) begin
            angle <= 16'd0; // сброс угла
        end else if (key_pressed) begin
            case (key_code)
                8'h31: if (angle < 16'd360) angle <= angle + 16'd10; //
Увеличение угла на 10, если не превышает 360
                8'h32: if (angle > 16'd0) angle <= angle - 16'd10;    //
Уменьшение угла на 10, если больше 0
                default: angle <= angle; // ничего не делать
            endcase
        end
    end
endmodule
```

Приложение Д

Семисегментный индикатор

Листинг Д.1 – Семисегментный индикатор

```
`timescale 1ns / 1ps
module seven_segment_display(
    input wire clk,
    input wire reset,
    input wire [7:0] value,     // Входное значение для отображения
    output reg [6:0] seg,       // Сегменты CA-G
    output reg dp,              // Точка (DP)
    output reg [3:0] an         // Аноды
);

    reg [3:0] current_digit;
    reg [1:0] digit_select;

    always @(posedge clk or posedge reset) begin
        if (reset) begin
            digit_select <= 0;
        end else begin
            digit_select <= digit_select + 1;
        end
    end

    always @(*) begin
        case (digit_select)
            2'd0: current_digit = value[3:0];
            2'd1: current_digit = value[7:4];
            default: current_digit = 4'b0000;
        endcase
    end

    always @(*) begin
        case (current_digit)
            4'd0: seg = 7'b1000000; // 0
            4'd1: seg = 7'b1111001; // 1
            4'd2: seg = 7'b0100100; // 2
            4'd3: seg = 7'b0110000; // 3
            4'd4: seg = 7'b0011001; // 4
            4'd5: seg = 7'b0010010; // 5
            4'd6: seg = 7'b0000010; // 6
            4'd7: seg = 7'b1111000; // 7
            4'd8: seg = 7'b0000000; // 8
            4'd9: seg = 7'b0010000; // 9
            default: seg = 7'b1111111; // Пусто
        endcase
        dp = 1'b1; // Десятичная точка выключена
        an = 4'b1110; // Активен первый анод
    end
endmodule
```

Приложение Е

Тестовый модуль

Листинг Е.1 – Тестовый модуль

```
`timescale 1ns / 1ps

module test_Top;

    // Param
    reg clk;
    reg reset;
    reg key_pressed;
    reg [7:0] key_code;
    reg [15:0] sin_val;
    reg [15:0] cos_val;
    reg [2:0] angle;
    wire hsync;
    wire vsync;
    wire [2:0] rgb;

    // Clock generation: 50 MHz
    initial begin
        clk = 0;
        forever #10 clk = ~clk; // every 10 ns
    end

    // Test scenario
    initial begin
        // Initialize and reset system
        reset = 1;
        key_pressed = 0; // Initially, no key pressed
        key_code = 0;    // Initially, key code is 0
        sin_val = 16'h0000; // Initialize sine value
        cos_val = 16'h0000; // Initialize cosine value
        angle = 3'b000;     // Initialize angle

        #50; // Delay before releasing reset
        reset = 0;

        // Simulate key presses and value changes
        repeat (10) begin
            #50; // Delay between changes
            key_pressed = 1;  //

            // Change key code to increase or decrease angle
            if ($random % 2 == 0) begin
                key_code = 8'h31;
            end else begin
                key_code = 8'h32;
            end

            // sin, cos angle values
            sin_val = $random; // Generate random sine value
            cos_val = $random; // Generate random cosine value
            angle = angle + 1; // Increment angle

            #20;
            key_pressed = 0;  // Reset key pressed signal

end
```

Продолжение Листинга Е.1

```
        // End sim
        #50000;
        $finish;
    end

    // top module
    top_module uut (
        .clk(clk),
        .reset(reset),
        .key_pressed(key_pressed),
        .key_code(key_code),
        .hsync(hsync),
        .vsync(vsync),
        .rgb(rgb)
    );
endmodule
```

Приложение Ж

Листинг кода реализации игры на Python

Листинг Ж.1 – Код реализации на Python

```
import numpy as np
import matplotlib.pyplot as plt

def cordic(angle, n_iter=100):
    x = 1.0  # cos(0)
    y = 0.0  # sin(0)
    z = angle

    angles = np.arctan(1 / (2 ** np.arange(1, n_iter + 1)))
    for i in range(n_iter):
        if z < 0:
            x_new = x + (y * (2 ** -i))
            y_new = y - (x * (2 ** -i))
            x, y = x_new, y_new
            z += angles[i]
        else:
            x_new = x - (y * (2 ** -i))
            y_new = y + (x * (2 ** -i))
            x, y = x_new, y_new
            z -= angles[i]

    return x, y


angle = 0
step = 1

fig, ax = plt.subplots()
x_vals = np.linspace(0, 2 * np.pi, 100)
sine_line, = ax.plot(x_vals, np.sin(x_vals), label='Синус', color='blue')
cosine_line, = ax.plot(x_vals, np.cos(x_vals), label='Косинус', color='orange')

sqrt_line, = ax.plot(x_vals, np.sqrt(np.clip(x_vals + angle, 0, None)),
label='Квадратный корень', color='green')

# Отображение текущего угла
angle_text = ax.text(0.5, 2.3, f'Угол: {angle:.2f}°', fontsize=12, ha='center')

ax.set_ylim(-1.5, 2.5)  # Увеличиваем лимит по оси Y, чтобы видеть графики
ax.legend()
plt.title('Генерация и визуализация трансцендентных функций')
plt.grid()

def update_plot():
    # Обновление графиков синуса и косинуса с использованием метода Кордик
    sine_values = []
    cosine_values = []

    for val in x_vals:
        # Преобразуем угол в радианы для вычисления
        radian_angle = np.radians(val + angle)
        cos_val, sin_val = cordic(radian_angle)  # Обновляем углы
        sine_values.append(sin_val)
        cosine_values.append(cos_val)
```

Продолжение Листинга Ж.1

```
    sine_line.set_ydata(sine_values)
    cosine_line.set_ydata(cosine_values)

    # Обновление графика квадратного корня
    sqrt_line.set_ydata(np.sqrt(np.clip(x_vals + angle, 0, None)))     #
Квадратный корень изменяется в зависимости от угла

    # Обновление текста с текущим углом
    angle_text.set_text(f'Угол: {angle:.2f}°')

    plt.draw()

def on_key(event):
    global angle
    if event.key == '1':  # Увеличить угол
        angle += step
        update_plot()
    elif event.key == '2':  # Уменьшить угол
        angle -= step
        update_plot()

# Подключаем обработчик событий клавиатуры
fig.canvas.mpl_connect('key_press_event', on_key)

plt.show()
```

Приложение 3

Руководство пользователя

Данное руководство предоставляет инструкции по подключению, запуску и использованию устройства, разработанного на языке описания аппаратуры Verilog. Устройство принимает десятичные числа и команды для работы с ними, использует алгоритм CORDIC для вычисления синуса и косинуса, а также выводит результаты на VGA-дисплей.

Перед началом убедитесь, что у вас есть следующее оборудование: Плата с ПЛИС (программируемая логическая интегральная схема), Клавиатура с USB-подключением, Монитор с VGA-входом, Кабели для подключения

Шаги подключения:

Подключение клавиатуры: Вставьте USB-кабель клавиатуры в соответствующий разъем на плате.

Подключение монитора: Подключите VGA-кабель монитора к плате и к источнику питания. При необходимости включите монитор.

Включение платы: Подключите питание к плате и загрузите на неё битовый поток (Bitstream).

Когда на плате загорится зеленым индикатор Done, это будет обозначать, что она запрограммирована, а значит, можно приступать к работе с ней. Также на экране должна появится сетка игровой области.

Ввод данных: Для ввода угла нажмите соответствующую клавишу на клавиатуре. Код нажатой клавиши будет передан в устройство. Для начала расчета угла нажмите клавишу Enter.

Управление: Для сброса текущих данных используйте клавишу Сброса (пробел). Убедитесь, что при работе с клавиатурой не нажимаете две клавиши одновременно, так как это будет воспринято как ошибка.

Вывод данных на экран: После ввода угла устройство будет вычислять значения синуса и косинуса с использованием алгоритма CORDIC. Цвет

пикселей на VGA-дисплее будет изменяться в зависимости от значения угла и вычисленных синуса и косинуса. Если угол изменится, цвет будет обновлен в соответствии с новыми значениями.