

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Программное обеспечение средств вычислительной техники и автоматизированных систем»»

БАКАЛАВРСКАЯ РАБОТА

на тему

**Устройство реализации тригонометрических
функций на ПЛИС**

Студент

31.05.2022

Бровкин
(подпись, дата)

Бровкин Андрей Алексеевич
(ФИО полностью)

Руководитель

31.05.2022

Е
(подпись, дата)

Е.И. Гурин
(фамилия, инициалы)

Нормоконтролёр

31.05.2022

А.С.Бычков
(подпись, дата)

А.С.Бычков
(фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 02.06.2022г. № 13)

Заведующий кафедрой

М.А. Митрохин
(подпись)

М.А. Митрохин
(фамилия, инициалы)

Работа защищена с отметкой Отлично протокол заседания ГЭК от

21 июня 2022 № 16)

Секретарь ГЭК

Е
(подпись)

Е.И. Гурин
(фамилия, инициалы)

Пенза 2022

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
 М.А. Митрохин
« 26 » апреля 2022г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студен Бровкин Андрей Алексеевич гр. 18BB2
Факультета вычислительной техники

2. Тема работы «Устройство реализации тригонометрических функций на ПЛИС

Тема утверждена приказом ПГУ №357/О от " 26 " апреля 2022г.

3. Руководитель бакалаврской работы д.т.н., профессор Гурин Евгений Иванович

4. Задание на работу (назначение разработки, исходные данные и т. п.)

Разработать устройства реализации тригонометрических функций на программируемых логических интегральных схемах (ПЛИС).

Реализуемые тригонометрические функции – синус, косинус.

Методы реализации тригонометрических функций – табличный, DDS, CORDIC.

Элементная база - ПЛИС фирмы Xilinx семейства Artix-7.

Разрядность данных – 24 -32.

Средства проектирования аппаратной части на языке Verilog: системы проектирования на ПЛИС фирмы Xilinx ISE 14.7, Vivado.

Средства отладки – система моделирования ISim фирмы Xilinx

Средства проектирования программ на языке C: система проектирования Visual Studio

5. Перечень подлежащих разработке вопросов

Обзор известных методов формирования тригонометрических функций.

Разработка схем на языке Verilog, реализующих используемые методы генерации тригонометрических функций – табличный, DDS, CORDIC

Разработка вспомогательных программ формирования тригонометрических функций на языке С.

Проверка работоспособности разработанных схем методом моделирования.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
	Анализ методов реализации тригонометрических функций	15%	26.04.2022 – 30.04.2022	
	Разработка устройств с применением табличного метода	10%	3.05.2022 – 5.05.2022	
	Моделирование устройств с применением табличного метода	10%	5.05.2022 – 7.05.2022	
	Разработка устройств с применением метода DDS	10%	10.05.2022 – 12.05.2022	
	Моделирование устройств с применением метода DDS	10%	12.05.2022 – 14.05.2022	
	Разработка устройств с применением метода CORDIC	15%	16.05.2022 – 19.05.2022	
	Моделирование устройств с применением метода CORDIC	15%	19.05.2022 – 21.05.2022	
	Оформление пояснительной записки	15%	21.05.2022 - 31.05.2022	

Дата выдачи задания " 26 " апреля 2022г.

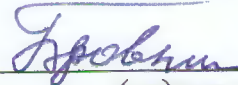
Руководитель бакалаврской работы



(подпись, дата)

26.04.2022

Задание к исполнению принял студент

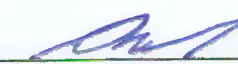


(подпись, дата)

26.04.2022

Работу к защите допустить

Декан факультета



(подпись,

дата)

Реферат

Бакалаврская работа содержит 78 листов, 29 рисунков, 10 источников и 4 приложения.

ТРИГОНОМЕТРИЯ, ПЛИС, ТАБЛИЧНЫЙ МЕТОД, DDS, CORDIC.

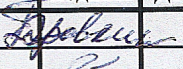
Цель работы – разработать устройства реализации тригонометрических функций на программируемых логических интегральных схемах.

Объектами разработки являются:

– схемы устройств реализации тригонометрических функций на ПЛИС.

В результате проделанной работы разработаны устройства, полностью соответствующие поставленной задаче, а именно: реализованы тригонометрические функции – синус, косинус тремя методами: табличным методом, DDS и CORDIC.

ПГУ 1.090301.05.001 ПЗ

					ПГУ 1.090301.05.001 ПЗ		
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.		Бровкин А. А.			Устройство реализации тригонометрических функций на ПЛИС Пояснительная записка		
Провер.		Гурин Е. И.					
И. Контр		Бычков А. С.					
Утверд.		Митрохин М.А.					
					Лит.	Лист	Листов
						2	
					ФВТ, каф. ВТ		
					гр. 18ВВ2		

Содержание

Содержание	3
Введение.....	5
1 Описание известных методов	7
1.1 Описание работы табличного метода	7
1.2 Описание метода <i>DDS</i>	8
1.3 Описание метода <i>CORDIC</i>	10
2 Реализация табличного метода	13
2.1 Разработка программы для табличного метода	13
2.2 Разработка схемы, реализующей табличный метод	16
2.3 Отладка схемы, реализующей табличный метод.....	19
2.4 Реализация схемы на ресурсах ПЛИС	22
3 Реализация метода <i>DDS</i>	24
3.1 Разработка программы для метода <i>DDS</i>	24
3.2 Разработка схемы генерации синусоиды методом <i>DDS</i>	26
3.3 Отладка схемы генерации синусоиды методом <i>DDS</i>	30
3.4 Реализация схемы <i>DDS</i> на ресурсах ПЛИС.....	32
4 Разработка устройства по методу <i>CORDIC</i>	36
4.1 Разработка схемы, реализующей алгоритм <i>CORDIC</i>	36
4.2 Отладка схемы <i>CORDIC</i>	39
5 Безопасность жизнедеятельности.....	45
5.1 Характеристика потенциальных опасностей и вредностей, которые могут возникнуть при работе с ЭВМ	45
5.2 Эргономические требования к рабочему месту пользователя	45

5.3 Электробезопасность при работе со средствами вычислительной

техники	46
Заключение	48
Список использованных источников	49
Приложение А – Листинги программ	50
Приложение Б – Аппаратная реализация	64
Приложение В – Временные диаграммы.....	67
Приложение Г – Презентация	70

Введение

Современные системы на кристалле, построенные на программируемых логических интегральных схемах (ПЛИС), широко применяются во многих областях науки и техники [1–3]. Микроэлектронные технологии позволяют реализовывать подобные системы на базе ПЛИС типа *FPGA*. Реализация систем в виде специализированных *ASIC* схем требует значительных временных и финансовых затрат, поэтому для построения таких систем сейчас часто применяются ПЛИС.

Современные ПЛИС могут иметь большую емкость, что позволяет включать в состав разрабатываемых систем процессорные блоки. Стандартная система на кристалле состоит из трех основных блоков: процессора, памяти и логики. Процессор выполняет функции управления, а также алгоритмически сложные и относительно медленные вычислительные операции. Память используется для хранения программы и данных. Логика предназначена для реализации конкретных задач, требующих высокой скорости обработки данных. Для выполнения операций в логике создаются блоки для вычисления функций, в том числе, тригонометрических.

В данной работе рассматриваются способы построения специализированных блоков на основе логики ПЛИС, позволяющих вычислять синусоидальные последовательности разными методами без использования процессорных блоков.

Для решения данной задачи использовались следующие инструменты:

— *Vivado* 2017.4 – для разработки и моделирования схем, реализующих тригонометрические функции синуса и косинуса;

— *ISE Design Suite* 14.7 – для разработки и оценки размещения в ПЛИС схем, реализующих тригонометрические функции синуса и косинуса;

— *Microsoft Visual Studio* 2019 – для написания программ, предназначенных для предварительной обработки данных, необходимой при реализации некоторых из рассматриваемых методов.

При разработке схем использовался язык текстового описания аппаратуры *Verilog HDL*.

При разработке вспомогательных программ, формирующих последовательности данных для разрабатываемых устройств использовался язык C++.

					ПГУ 1.090301.05.001 ПЗ	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

1 Описание известных методов

В настоящее время существует множество различных методов генерации синусоидальных сигналов и последовательностей. Однако, большинство из них либо требуют значительных аппаратных и временных затрат, либо не обладают высокой точностью [4-9].

В настоящей работе рассматриваются наиболее известные: табличные методы на основе блоков памяти, и метод «цифра за цифрой», в зарубежной литературе называемый – *CORDIC*.

1.1 Описание работы табличного метода

Табличные методы одни из наиболее простых в реализации методов генерации синусоидальных сигналов и последовательностей. Простейший из табличных методов просто выдает заранее рассчитанные значения функции, из памяти. Но не смотря на его простоту, или благодаря ей, данный метод обладает достаточно высоким быстродействием. Обратной же стороной метода, является высокая требовательность к аппаратным ресурсам и низкая точность [10].

Для работы табличного метода необходимо заранее вычислить значения тригонометрических функций и записать полученные данные в запоминающее устройство. В некоторых модификациях метода, записывается не весь период тригонометрической функции, а лишь часть. Таким образом, пытаются найти компромисс между быстродействием и точностью.

На рисунке 1.1 демонстрируется пример работы табличного метода. Здесь, в памяти устройства по адресам x_1 , x_2 , x_3 записываются значения y_1 , y_2 , y_3 . В разделе 2 данной работы, представлена реализация табличного метода.

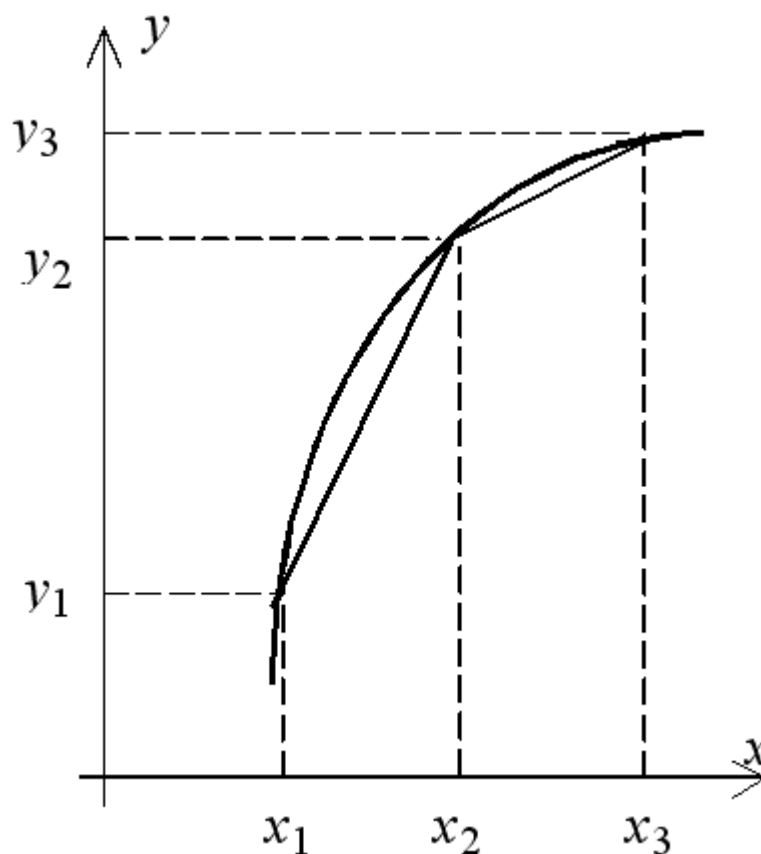


Рисунок 1.1 – Табличный метод

1.2 Описание метода *DDS*

Метод *DDS* – это модификация табличного метода для вычисления синуса, в котором таблица строится лишь для четверти периода. Так же, как и в обычном табличном методе, значения функции вычисляются заранее и записываются в память [10].

На рисунке 1.2 представлен тригонометрический круг для функции синус. Здесь римскими цифрами отмечены квадранты, четверти круга. Четвертью периода функции синус, для которой вычисляются значения при реализации метода *DDS*, является один квадрант. Чтобы быть точным, вычисляются значения первой четверти периода и, соответственно, первого квадранта.

Из тригонометрического круга можно заметить некоторые закономерности.

Так, второй квадрант зеркально равен первому квадранту, следовательно, чтобы получить значения второго квадранта, зная значения первого, достаточно прочитать первый квадрант в обратном порядке.

Третий квадрант равен первому по модулю, следовательно, чтобы получить значения третьего квадранта, зная значения первого, достаточно изменить знак у значений первого квадранта.

Четвертый квадрант равен второму по модулю, следовательно, чтобы получить значения четвертого квадранта, зная значения первого, необходимо прочитать первый квадрант в обратном порядке и изменять знак у его значений.

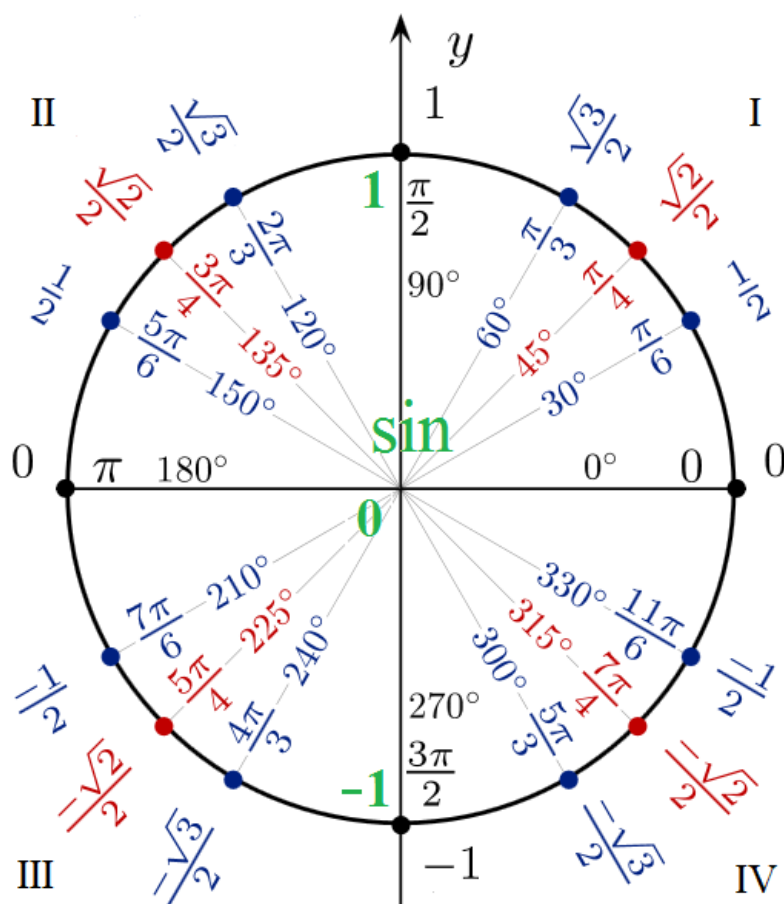


Рисунок 1.2 – Тригонометрический круг для функции синус

За счет вышеописанных преобразований, метод *DDS* немного теряет в быстродействии, но менее требователен к аппаратным ресурсам и выигрывает у обычного табличного метода в точности.

1.3 Описание метода *CORDIC*

Изначально алгоритм *CORDIC*, был придуман для реализации поворота вектора аппаратно при помощи цифровой схемотехники [4-5].

Чтобы повернуть вектор нужно вычислить его новые координаты.

Для этого текущие координаты вектора умножаются на матрицу поворота,

$$M(\varphi) = \begin{pmatrix} \cos \varphi & -\sin \varphi \\ \sin \varphi & \cos \varphi \end{pmatrix}, \quad (1.1)$$

где φ – угол поворота.

Умножение квадратной матрицы на вектор-столбец осуществляется по формуле:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \times \begin{bmatrix} b_{11} \\ b_{21} \end{bmatrix} = \begin{bmatrix} c_{11} = a_{11} \cdot b_{11} + a_{12} \cdot b_{21} \\ c_{21} = a_{21} \cdot b_{11} + a_{22} \cdot b_{21} \end{bmatrix}. \quad (1.2)$$

Тогда координаты вычисляются по формулам (1.3) и (1.4),

$$x_i = x_{i-1} \cdot \cos(\varphi)_i - y_{i-1} \cdot \sin(\varphi)_i, \quad (1.3)$$

$$y_i = x_{i-1} \cdot \sin(\varphi)_i + y_{i-1} \cdot \cos(\varphi)_i. \quad (1.4)$$

Так как тангенс – это синус, деленный на косинус, то вынеся косинус за скобку, эти формулы можно переписать в виде:

$$x_i = \cos(\varphi)_i \cdot (x_{i-1} - y_{i-1} \cdot \tan(\varphi)_i), \quad (1.5)$$

$$y_i = \cos(\varphi)_i \cdot (y_{i-1} + x_{i-1} \cdot \tan(\varphi)_i). \quad (1.6)$$

Если выразить $\cos(\varphi)_i$ через тангенс по основному тригонометрическому тождеству (1.7), и выбирать такой угол поворота, при котором $\tan(\varphi)_i$ принимает значения $\pm 2^{-i}$, где i – целое число. Тогда умножение значений x_{i-1} и y_{i-1} на $\tan(\varphi)_i$ превращается в простую операцию сдвига их значений на i разрядов вправо. В виде формул это будет выглядеть так:

$$\cos(\varphi)_i = \frac{1}{\sqrt{1+\tan^2(\varphi)_i}}, \quad (1.7)$$

$$x_i = K_i \cdot (x_{i-1} - \sigma \cdot 2^{-i} \cdot y_{i-1}), \quad (1.8)$$

$$y_i = K_i \cdot (y_{i-1} + \sigma \cdot 2^{-i} \cdot x_{i-1}), \quad (1.9)$$

где $K_i = \frac{1}{\sqrt{1+2^{-2i}}}$.

Умножение на K_i можно исключить, и в конце умножать на K_n ,

$$x_i = x_{i-1} - \sigma \cdot 2^{-i} \cdot y_{i-1}, \quad (1.10)$$

$$y_i = y_{i-1} + \sigma \cdot 2^{-i} \cdot x_{i-1}, \quad (1.11)$$

$$K_n = \prod_{i=0}^{n-1} K_i = \prod_{i=0}^{n-1} \frac{1}{\sqrt{1+2^{-2i}}}. \quad (1.12)$$

Алгоритм работает следующим образом. Если представить угол в виде суммы углов $\varphi_i = \tan^{-1}(2^{-i})$, где $i = 0, 1, 2$ и т. д., то операция поворота вектора будет состоять из последовательных элементарных поворотов. На первой итерации, поворот осуществляется на угол 45 градусов. Каждый следующий угол поворота меньше предыдущего. Для каждой итерации проверяется, больше ли найденный угол исходного угла, если нет, то σ равна единице, иначе – минус единице. Чем больше проведено итераций, тем точнее будет результат. Работа алгоритма представлена на рисунке 1.3, зеленым отмечен искомый вектор, черным итерации алгоритма.

2 Реализация табличного метода

В этом разделе описывается построение блока, реализующего табличный метод, рассматривается вариант табличного метода, в котором таблица строится для интервала 2π . При таком варианте построения достигается максимальное быстродействие схемы, однако, его точность хуже по сравнению с другими способами. Другим минусом этого варианта являются увеличенные аппаратные затраты. Плюсы и минусы данного варианта табличного метода обуславливаются тем, что генерация синусоидального сигнала не требует дополнительных преобразований, так как вся синусоида уже рассчитана. Здесь используется свойство периодичности синуса, чей период как раз равен 2π .

Для работы данного блока требуется массив данных, который в настоящей работе был сформирован программой *sinApp*.

2.1 Разработка программы для табличного метода

При работе с табличным методом необходимо построить таблицу значений. Так, для генерации синусоидального сигнала с заданным периодом, была написана программа *sinApp* на языке C++, вычисляющая значения функции,

$$y = f(x): \quad (2.1)$$

$$y_0 = f(x_0), y_1 = f(x_1), \dots, y_n = f(x_n), \quad (2.2)$$

$$x_{i+1} = x_i + \Delta x, \quad (2.3)$$

где $\Delta x = \text{const}$.

Всего вычисляется 256 значений. Вычисления каждого значения y_i в программе *sinApp* производится отдельно по формуле:

$$y_i = M \cdot \sin(x_i), \quad (2.4)$$

где $x_i = \frac{i \cdot 2\pi}{256}$, M – масштаб.

Масштаб используется для того, чтобы избежать использования в схеме иррациональных значений. В данном случае он равен $7FFF$ в шестнадцатеричной системе счисления, или 32767 в десятичной. Чем больше масштаб, тем выше точность, так как при умножении значение y_i округляется до целых. Интервал, для которого строится таблица, определяется с помощью количества рассчитываемых значений и знаменателя функции для вычисления x_i . Чтобы построить таблицу для интервала 2π , они должны быть равны. При этом, чем больше делитель, тем меньше шаг, за счёт чего повышается точность.

Результатом работы программы является текстовый файл *sinApp.txt*, содержащий требуемые значения функции. Этот файл использовался при работе схемы. Полностью программа *sinApp* представлена в приложении А.

Фрагмент программы приведен ниже. Как можно увидеть, вычисления проводятся в цикле, каждый такт цикла, по описанным выше формулам, вычисляется дробное значение x_i , которое затем используется при вычислении целого значения y_i . Результат записывается в файл *sinApp.txt* в шестнадцатеричной системе счисления. За исключением файла *sinApp.txt*, полученные данные нигде не хранятся, каждый такт цикла, старое значение перезаписывается.

```
// Вычисление и запись 256 значений y(x)
for (int i = 0; i < 256; i++)
{
    // Вычисление x-итого по формуле  $x_i = 2\pi * i / 256$ 
    x = (2 * PI * i) / 256;

    // Вычисление y(x) по формуле  $y_i = M * \sin(x_i)$ 
    y = round(M * sin(x));

    // Вывод в файл форматированного значения y
    out << hex                // шестнадцатеричная система
                                // счисления
}
```

```

    << uppercase      // верхний регистр
    << setw(4)         // четыре символа
    << setfill(' ')    // заполнение пробелом
    << y
    << endl;
}

```

Правильность работы программы можно определить, найдя пики функции. Как можно заметить по графику функции, на рисунке 2.1, на заданном промежутке, значения функции должны начинаться с нуля, возрастать до максимального положительного значения, затем убывать до максимального по абсолютной величине отрицательного значения и, наконец, опять возрастать до достижения нуля. Под “значением”, подразумевается значение функции синус умноженное на масштаб. Из свойства периодичности помним, последующие значения просто повторяются, поэтому и не требуется рассчитывать и запоминать значения для большего промежутка.

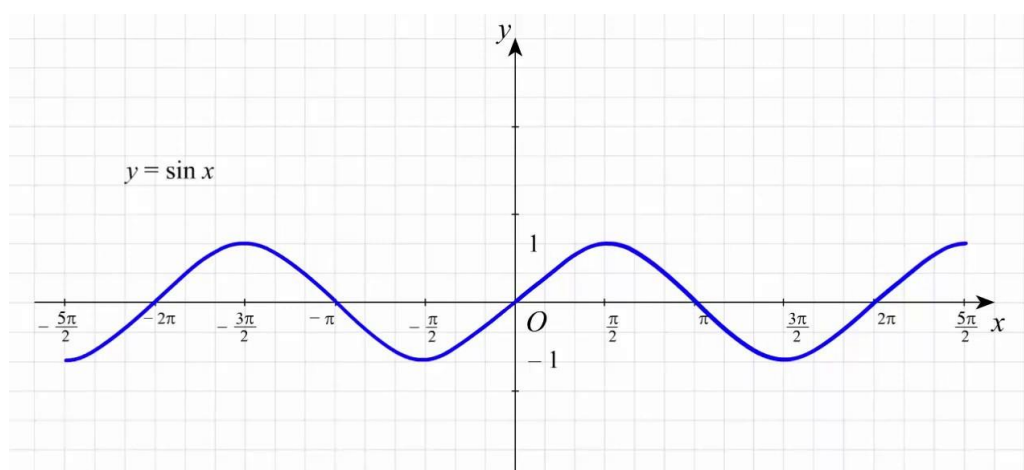


Рисунок 2.1 - График синуса

Фрагмент файла *sinApp.txt*, в котором представлены результаты работы программы, приведен ниже.

```

0
324  — Минимальное положительное число

```

648
... — Возрастание числа
7FF5
7FFF — Максимальное положительное число
7FF5
... — Убывание числа
324
0
FCDC — Минимальное по абсолютной величине отрицательное число
... — (Возрастание по абсолютной величине / Убывание) числа
800B
8001 — Максимальное по абсолютной величине отрицательное число
800B
... — (Убывание по абсолютной величине / Возрастание) числа
FCDC

Из выше перечисленного можно сделать вывод о правильности работы программы.

2.2 Разработка схемы, реализующей табличный метод

Следующим шагом является построение на языке *Verilog* специализированного блока на основе ПЛИС, позволяющего генерировать синусоидальный сигнал.

Данный блок берет значения из файла *sinApp.txt*, рассчитанного в первой части второй главы, и на их основе строит синусоиду.

Блок состоит из двух модулей. Главного модуля *topm1*, схема которого описана в файле *topm1.v*, предназначенного для управления работой модуля *fsin*, который описан в файле *fsin.v*. Модуль *fsin* собственно и генерирует синусоиду.

Главный модуль, *topm1*, содержит восьмиразрядный счетчик *ct8*, который каждый такт увеличивается на константу, с помощью которой, в свою очередь, задается частота синусоиды. Связь между константой и частотой — прямая, чем

больше константа, тем выше частота. Это будет продемонстрировано в третьей части раздела.

```
// Переменная счетчика
reg [7:0] ct8 = {8{1'b0}};

// Цикл увеличения счетчика
always @(posedge clk)
    ct8 <= ct8 + 8'h01; // Чем больше константа, тем выше
                        // частота
```

Далее вызывается модуль *fsin*, который используется как компонент. В него передаются значения счетчика, они сохранены в отдельную переменную *x*, и сигналы: синхронизации – *clk*, и сброса – *reset*. Обратно получают результат работы модуля *fsin*, сигнал *qsin*. Полный код модуля *topm1* представлен в приложении А.

```
// Подключение модуля генерации синуса
fsin u1 ( .x(x), .clk(clk), .reset(reset), .qsin(qsin) );
// Соединение сигналов
assign qs=qsin[15:0];
assign x=ct8;
```

Для использования внешних данных в схеме на языке *Verilog*, необходимо проинициализировать блок памяти.

Для инициализации используется команда *initial \$readmemh* с аргументами определяющими: имя файла, из которого берутся данные, имя регистра, в который данные сохраняются, и необязательные адреса начала и конца считывания данных.

```
initial $readmemh("<data_file_name>", <ram_name>,
                  <begin_address>, <end_address>);
```

Во внутреннем модуле *fsin* инициализируется блок памяти, в который записываются сгенерированная таблица.

```
// Инициализация блока памяти
reg signed [31:0] roms [255:0];
initial $readmemh("sinApp.txt", roms);
```

Для навигации по массиву памяти используется переменная *addr*, хранящая адрес.

```
// Инициализация переменной адреса
reg unsigned [7:0] addr;
initial addr=0;
```

Далее проверяется, не установлен ли сигнал сброса, и если да, то адрес устанавливается в нуль. В противном случае адрес формируется от входной переменной *x*, в которой хранится полученное значение счётчика.

```
if (reset == 1)
// Обнуление адреса, при сигнале сброса
    addr <= 0;
else
// Присвоение адресу значения, полученного из управляющего
// модуля
    addr <= x;
```

Значение синуса берется из массива памяти по полученному адресу. Таким образом, формируется синусоида.

```
// Получение значения из таблицы по адресу
qsin <= (roms[addr]);
```

Полный код модуля *fsin* на языке *Verilog* представлен в приложении А.

2.3 Отладка схемы, реализующей табличный метод

Для проектирования была выбрана система *Vivado* от фирмы *Xilinx*, в которой представлена удобная система моделирования работы схем.

На рисунках в этой главе представлены графики зависимости частоты синусоиды от константы, под константой, здесь и далее подразумевается величина значения, на которое увеличивается счетчик *ct8* в управляющей схеме *topm1*.

Наименьшим шагом счётчика является единица, частоту синусоиды для данного шага можно увидеть на рисунке 2.2. На всех графиках в этой главе представлены сигнал синхронизации *clk*, сигнал сброса *reset*, сигнал адреса *addr*, и выходной сигнал *qs*, последний представлен в двух вариантах, цифровой и аналоговой формах.

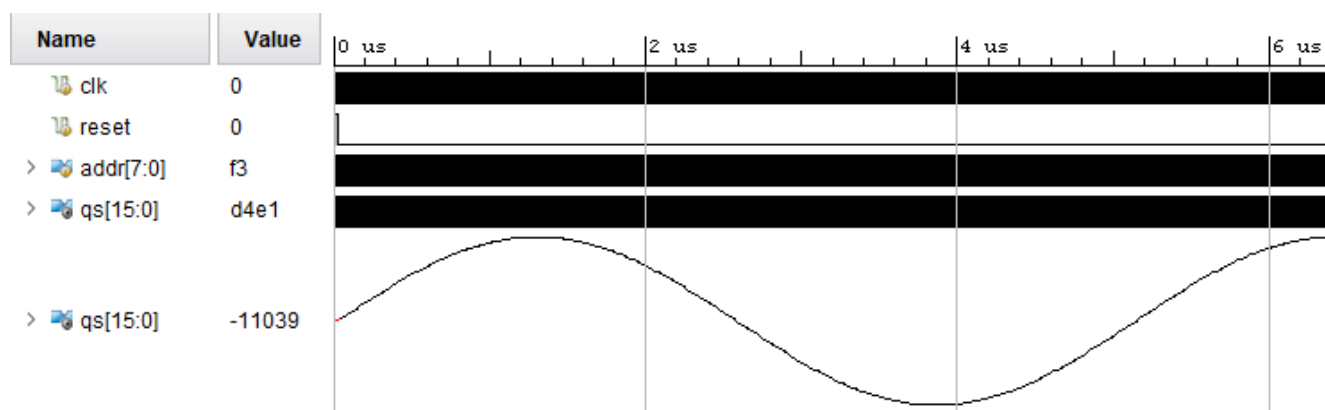


Рисунок 2.2 - Частота синусоиды при *const* = 1

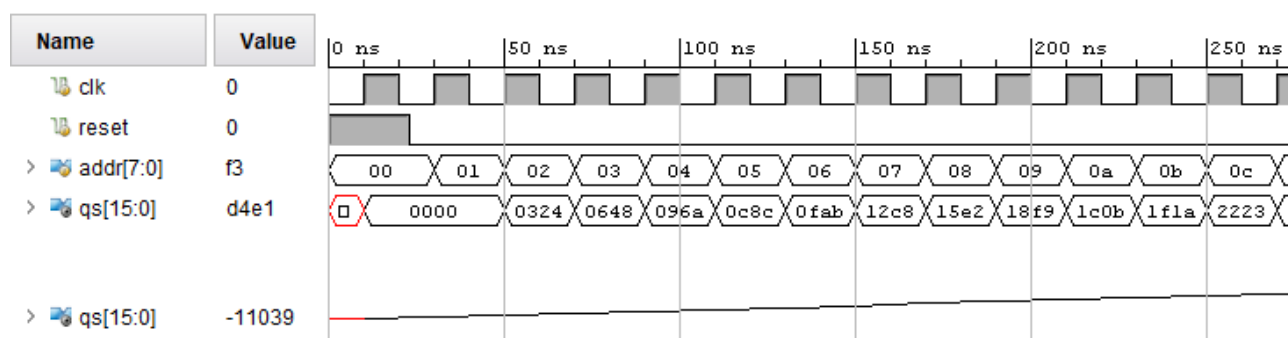


Рисунок 2.3 - Значения синусоиды при *const* = 1

Из рисунка 2.2 мы видим, что при данном значении константы, частота сигнала достаточно низкая, один период длится приблизительно пять микросекунд. Рисунок 2.3 показывает тот же график, но в более крупном масштабе, он подтверждает, что адрес увеличивается на значение константы. Для построения графика при этом из таблицы берутся все значения.

На рисунках 2.4 и 2.5 в разном масштабе, представлен график для константы равной трем. Если сравнить его с предыдущим, можно заметить, что при увеличении константы, частота сигнала повысилась. В сравнении с предыдущим графиком, константа была увеличена в три раза, частота также повысилась в три раза. За пять микросекунд, теперь генерируются приблизительно три периода функции.

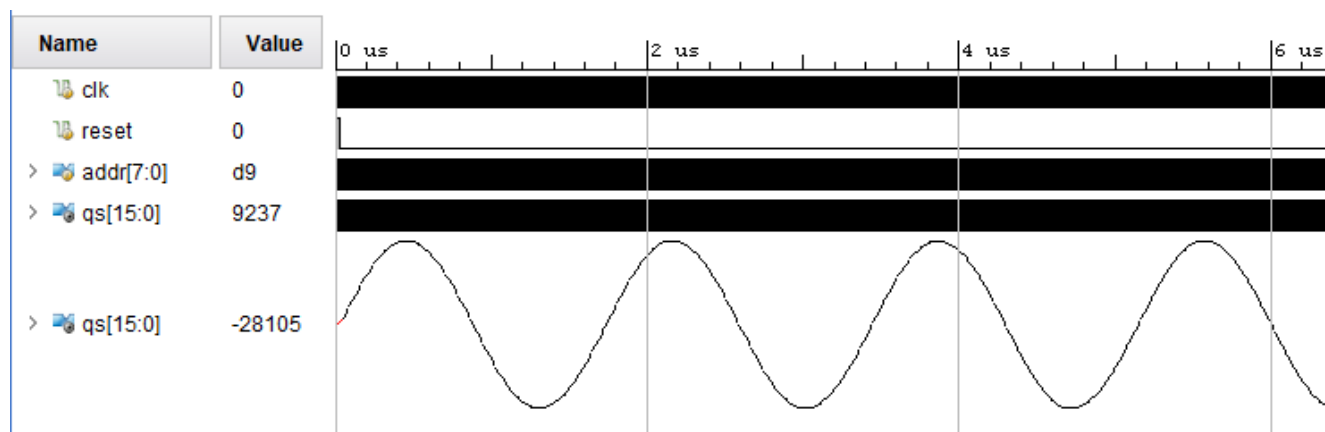


Рисунок 2.4 - Частота синусоиды при $const = 3$

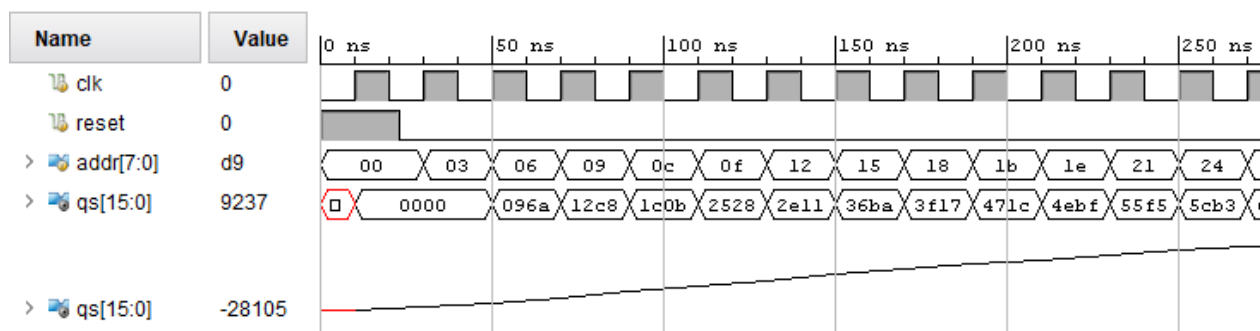


Рисунок 2.5 - Значения синусоиды при $const = 3$

Для построения графика при этом из таблицы берется лишь каждое третье значение.

График для константы равной семи, представленный на рисунках 2.6 и 2.7, подтверждает зависимость частоты сигнала от величины константы. Как видно из графика, за пять микросекунд теперь генерируются приблизительно семь периодов функции

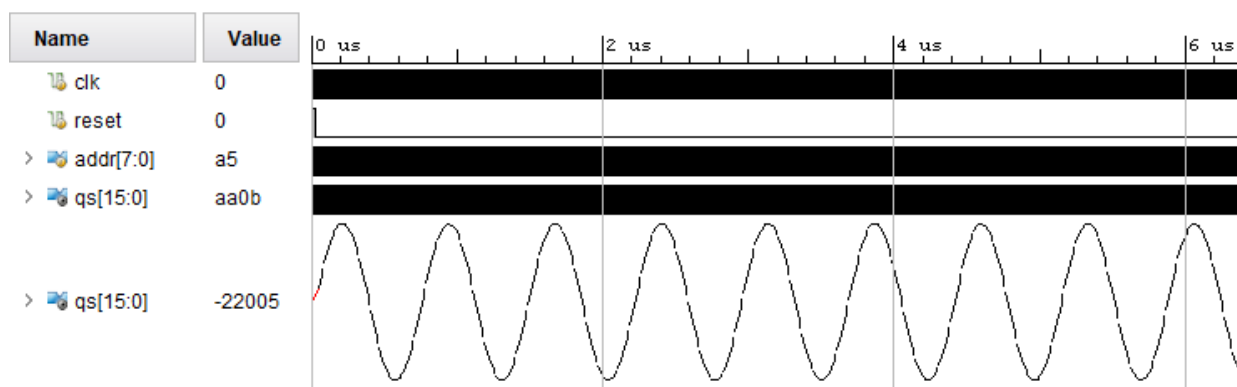


Рисунок 2.6 - Частота синусоиды при $const = 7$

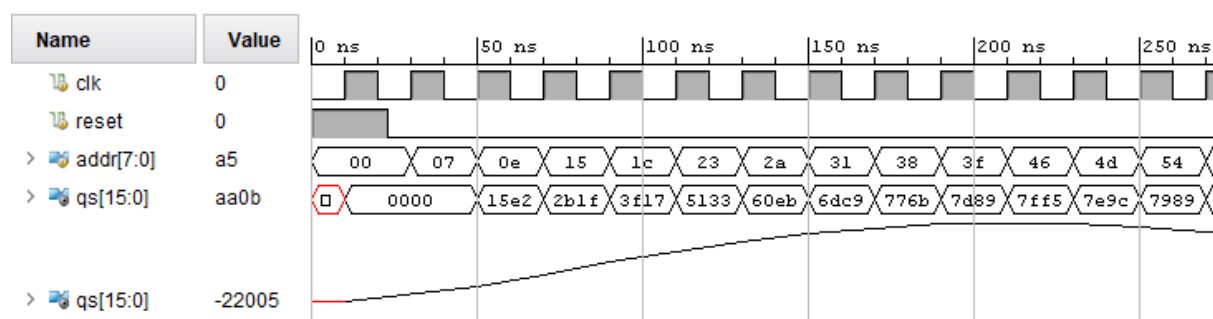


Рисунок 2.7 - Значения синусоиды при $const = 7$

Для построения графика при константе равной 7, из таблицы берется каждое седьмое значение.

На рисунке 2.8, приведен фрагмент файла *sinApp.txt*, содержащего рассчитанные значения функции. С его помощью определим быстродействие схемы. Для этого рассмотрим рисунки 2.3, 2.5 и 2.7. Из них видим, что 30-ой наносекунде переменная *addr*, хранящая адрес, равна $01h$, $03h$ и $07h$, соответственно. Как можно увидеть из этих рисунков, на 50-ой наносекунде, т. е. на следующем такте синхросигнала, выходной сигнал получает соответственные значения из памяти по этому адресу $324h$, $96Ah$ и $15E2h$. Из этого делаем вывод,

что данная схема имеет достаточно высокое быстродействие не зависимо от значений константы.

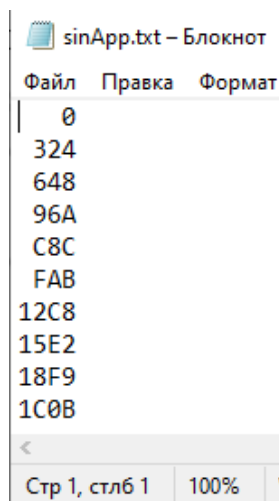


Рисунок 2.8 - Фрагмент файла со значениями функции

2.4 Реализация схемы на ресурсах ПЛИС

Для оценки размещения схемы в ПЛИС была выбрана система *ISE Design Suite* 14.7 от фирмы *Xilinx*. Она позволяет удобно проверить, какие и сколько ресурсов ПЛИС использует схема.

Была проведена оценка размещения описанной схемы в ПЛИС семейства *Artix-7*. На рисунке 2.9, приведена таблица с результатами размещения, показывающая, какие ресурсы плис были использованы, и сколько осталось свободных.

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slice Registers	16	126800	0%
Number of Slice LUTs	8	63400	0%
Number of fully used LUT-FF pairs	8	16	50%
Number of bonded IOBs	18	210	8%
Number of Block RAM/FIFO	1	135	0%
Number of BUFG/BUFGCTRLs	1	32	3%

Рисунок 2.9 - Используемые ресурсы ПЛИС

Строка *Number of Slice Registers*, показывает количество использованных секций, 16 из 126800.

Строка *Number of LUTs*, показывает количество использованных таблиц, реализующих функции алгебры логики, 8 из 63400.

Строка *Number of bonded IOBs*, показывает количество использованных входов и выходов ПЛИС, 18 из 210.

Строка *Number of Block RAM/FIFO*, показывает количество использованных блоков памяти, 1 из 135.

Результат синтеза схемы по описанию на языке Verilog и временная диаграмма работы приведены в приложениях Б и В, соответственно.

					ПГУ 1.090301.05.001 ПЗ	Лист
						23
Изм.	Лист	№ докум.	Подпись	Дата		

3 Реализация метода *DDS*

В этом разделе, описывается модификация табличного метода, в которой таблица строится для интервала $\frac{\pi}{2}$. Данный вариант построения имеет свои плюсы, в частности, поскольку в массив памяти сохраняется лишь первый квадрант, равный четверти периода функции, он позволяет сэкономить значительный объем памяти, или же, при тех же затратах памяти, повысить точность. Но из этого же вытекает и минус, получение всего периода функции потребует больше времени, из-за выполнения дополнительных преобразований.

3.1 Разработка программы для метода *DDS*

Для построения таблицы значений, была написана программа *sinAppDDS* на языке C++, вычисляющая значения функции,

$$y = f(x): \quad (3.1)$$

$$y_0 = f(x_0), y_1 = f(x_1), \dots, y_n = f(x_n), \quad (3.2)$$

$$x_{i+1} = x_i + \Delta x, \quad (3.3)$$

где $\Delta x = \text{const}$.

Программа *sinAppDDS* рассчитывает 256 значений. Вычисления каждого значения y_i производится отдельно по формуле

$$y_i = M \cdot \sin(x_i), \quad (3.4)$$

где $x_i = \frac{i \cdot 2\pi}{1024}$, M – масштаб.

Масштаб используется для того, чтобы избежать использования в схеме иррациональных значений. В данном случае он равен *FFFFFF* в шестнадцатеричной системе счисления, или 16777215 в десятичной. Чем больше масштаб, тем выше точность, так как при умножении значение y_i округляется до

целых. Интервал, для которого строится таблица, определяется соотношением количества рассчитываемых значений и знаменателя функции для вычисления x_i . Чтобы построить таблицу для интервала $\frac{\pi}{2}$ необходимо, чтобы знаменатель был в четыре раза больше, чем количество значений. При этом, чем он больше, тем меньше шаг, следовательно, тем выше точность.

Результатом работы программы является текстовый файл, содержащий требуемые значения функции. Значения начинаются с 0 и возрастают до $FFFFFFh$. Полный код программы *sinAppDDS* приведен в приложении А.

Фрагмент программы приведен ниже. Данная программа работает схожим образом с программой для генерации периода 2π , так как они реализуют те же формулы. Вычисления проводятся в цикле, каждый такт цикла по описанным выше формулам вычисляется x_i , который затем используется при вычислении y_i . Полученное значение записывается в файл *sinApp2.txt* в шестнадцатеричной системе счисления. За исключением файла *sinApp2.txt*, результаты вычислений нигде не хранятся, каждый такт цикла, старое значение перезаписывается.

Основным отличием является изменение делителя для вычисления x_i , это позволяет уменьшить шаг, благодаря чему повышается точность, и регулировать длину интервала, изменение которого до $\frac{\pi}{2}$ и является особенностью данной модификации табличного метода.

```
// Вычисление и запись 256 значений y(x)
for (int i = 0; i < 256 ; i++)
{
    // Вычисление x-итого по формуле xi = 2π * i / 1024
    x = (2 * PI * i) / 1024;

    // Вычисление y(x) по формуле yi= M * sin(xi)
    y = round(M * sin(x));

    // Вывод в файл форматированного значения y
    out << hex                      // шестнадцатеричная система
```

```

// счисления
<< uppercase // верхний регистр
<< setw(6) // шесть символов
<< setfill(' ') // заполнение пробелом
<< y // выводимая переменная
<< endl;
}

```

3.2 Разработка схемы генерации синусоиды методом *DDS*

Следующим шагом является построение самой схемы, предназначенной для генерации синусоидального сигнала, на языке *Verilog*.

Схема *dds* берёт значения из файла *sinApp2.txt*, сгенерированного в первой части третьей главы и на их основе строит синусоиду. Так как в файле находятся значения только для первой четверти, значения для трех оставшихся четвертей вычисляются в самой схеме по известным методам.

Вначале, в файле *fdds.v*, описывающем модуль *dds*, прописываются параметры, задающие разрядность для выходных данных и разрядность адреса.

```

// Разрядность вых.данных
parameter RAM_WIDTH = 25;
// Разрядность адреса
parameter ADDR_BITS = 16;

```

Частота синусоиды регулируется входным сигналом *n1*.

```

// Код частоты
input [19:0] n1;

```

На выход схемы подается сигнал *doa* заданной выше разрядности.

```

// Выход синус
output reg [RAM_WIDTH-1:0] doa;

```

Переменная счетчик *r2* используется для изменения адреса.

```
// Выход накапливающего сумматора  
reg [ADDR_BITS+1+DR:0] r2=0;
```

Для хранения адреса используется переменная *addrA*. Данные первой четверти периода, хранящиеся в памяти, дополнительно сохраняются в переменной *dataA*, где при необходимости преобразуются в данные других квадрантов.

```
// Адрес на входе ПЗУ [15:0]  
reg [ADDR_BITS-1:0] addrA;  
  
// Данные на выходе ПЗУ [24:0]  
reg [RAM_WIDTH-1:0] dataA;
```

Также используются переменные *zx* и *zx0*. Они отвечают за определение текущего квадранта.

```
// Переменные квадрантов (00, 01, 10, 11)  
reg [1:0] zx=0, zx0=0;
```

Для преобразований значений первого квадранта в последующие, используются константы, все биты которых равны единицам.

```
// Константа адреса (все единицы)  
reg [ADDR_BITS-1:0] c111={ADDR_BITS{1'b1}};  
  
// Константа данных (все единицы)  
reg [RAM_WIDTH-1:0] r111={RAM_WIDTH{1'b1}};
```

В массив памяти записываются значения из файла *sinApp2.txt*, сгенерированного в первой части третьей главы, содержащего значения первого квадранта.

```
// Массив памяти 25 x 2**8
reg [RAM_WIDTH-1:0] ram1 [(2**ADDR_BITS)-1:0];
...
// Инициализация памяти
initial
    $readmemh("sinApp2.txt", ram1, 0, (2**ADDR_BITS)-1);
```

Теперь переходим к генерации синусоиды, первым делом определяем адрес значения в памяти. Для этого используется счетчик *r2*.

```
// Сохранение значения счетчика
assign cta = r2[ADDR_BITS+1+DR:DR];
```

Проверяем, не установлен ли сигнал сброса, если да, обнуляем переменные адреса, счетчика и данных.

```
// Если сигнал сброса, обнулить переменные адреса, счетчика
// и данных
if(reset==1)
    begin
        addrA<=0; r2<=0; dataA <= 0;
    end
```

Счетчик *r2* увеличивается на значение кода частоты *n1*, чем больше значение кода частоты, тем выше частота синусоиды. Номер квадранта, также определяется с помощью счётчика, для этого проверяются его два старших бита.

```
// увеличение счетчика
r2 <= r2 + n1;
```

```

// Определение квадранта, старшие два бита
zx0 <= cta[ADDR_BITS+1:ADDR_BITS];
// Сохранение квадранта
zx <= zx0;

```

В квадрантах 1 и 3 абсолютные значения функции синуса меняются от *0h*, до *FFFFFFh*, так как в памяти находится значение первой четверти периода, адреса для значений этих квадрантов берутся без изменений. В квадрантах 2 и 4 абсолютные значения функции синуса, наоборот, меняются от *FFFFFFh*, до *0h*. То есть, для генерации этих четвертей периода нужно прочесть память в обратном порядке. Для этого используется инверсия адреса, путём его вычитания из константы *c111*, все биты которой равны единице.

```

// Если первый квадрант, берём значение счетчика без
// изменений, исключая два старших бита
if(cta[ADDR_BITS+1:ADDR_BITS]==2'b00)
    addrA<=cta[ADDR_BITS-1:0];
// Если второй квадрант, берём инверсию значения счетчика
// исключая два старших бита
else if(cta[ADDR_BITS+1:ADDR_BITS]==2'b01)
    addrA<=c111-cta[ADDR_BITS-1:0];
// Если третий квадрант, берём значение счетчика без
// изменений, исключая два старших бита
else if(cta[ADDR_BITS+1:ADDR_BITS]==2'b10)
    addrA<=cta[ADDR_BITS-1:0];
// Если четвертый квадрант, берём инверсию значение счетчика
// исключая два старших бита
else
    addrA<=c111-cta[ADDR_BITS-1:0];

```

Значения квадрантов 1 и 2 положительные, так как убывание второго квадранта уже реализовано, то значения этих квадрантов берутся из памяти без изменений. Значения квадрантов 3 и 4 отрицательные. Для изменения знака

используется инверсия данных, путём их вычитания из константы *r111*, все биты которой равны единице.

```
// Сохраняем значение из памяти, по определенному выше адресу
dataA <= ram1[addrA];
// Если квадранты 1 или 2, используем значение без изменений
if (zx==0)
    doa <= dataA;
else if (zx==1)
    do a<= dataA;
// Если квадранты 3 или 4, используем инверсию значения
else if (zx==2)
    doa <= r111 - dataA;
else
    doa <= r111 - dataA;
```

Полный код модуля *dds* на языке *Verilog* представлен в приложении А.

3.3 Отладка схемы генерации синусоиды методом *DDS*

Для проектирования была выбрана система *Vivado* от фирмы *Xilinx*, в которой представлена удобная система моделирования работы схем.

В отличие варианта реализации рассмотренного во втором разделе, частота синусоиды генерируется не константой внутри схемы, а внешним сигналом *n1*. Он задаётся в файле сигналов *tb3.v*. Другим отличием является величина сигнала, из-за большего, в четыре раза, количества значений, для генерации сигнала той же частоты, величина сигнала *n1* также должна быть в четыре раза больше.

На графиках этой главы представлены сигналы: синхронизации – *clk*, сброса – *reset*, разрешения работы – *en*, кода частоты – *n1*, сигнал *cta*, старшие два бита которого определяют квадрант, а младшие – адрес, сигнал *zx*, показывающий текущий квадрант и выходной сигнал *doa*, последний представлен в двух вариантах, цифровой и аналоговой формах.

На рисунке 3.1 представлен график изменения кода чистоты. Как можно заметить при $n1$, равном $00005h$, за две микросекунды генерируется приблизительно один период функции. При удвоении сигнала, при $0000Ah$, за те же две микросекунды генерируется около двух периодов функции. Аналогично, снова удвоив сигнал и получив $00014h$, за то же время генерируется четыре периода. Затем опять передается сигнал $00005h$.

Также стоит обратить внимание на квадранты, как видно из графика на рисунке 3.1, преобразования, проводимые над первой четвертью периода, проходят успешно. Синусоида не имеет разрывов в местах переключения квадрантов, при изменениях сигнала zx .

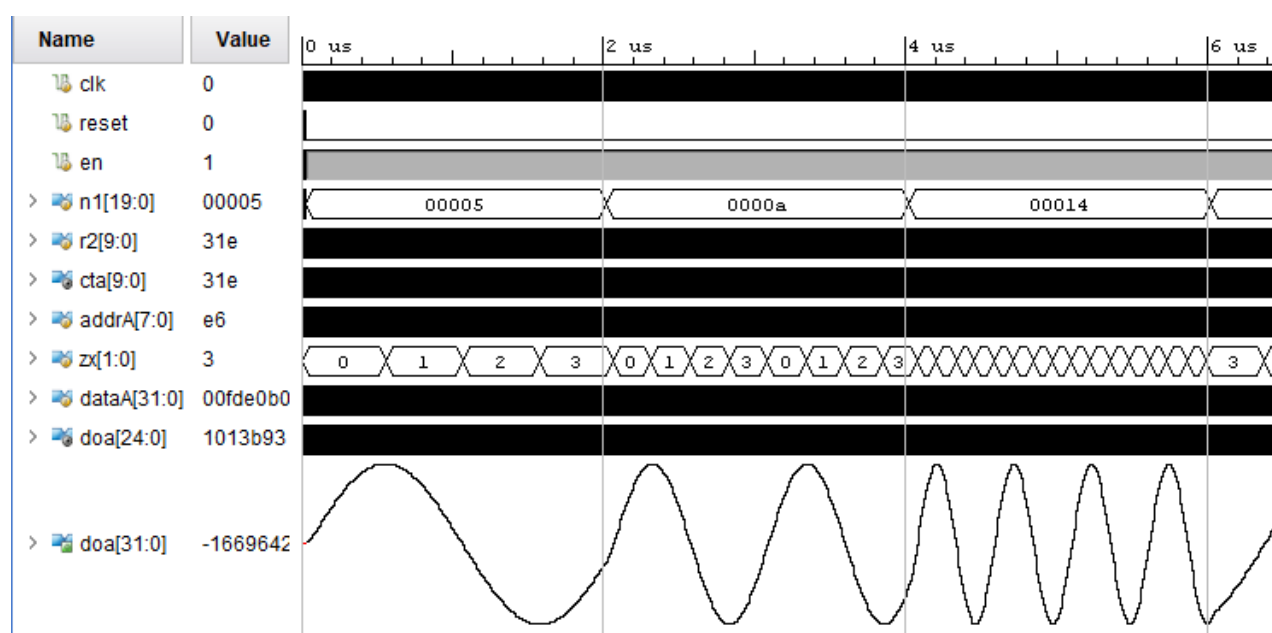


Рисунок 3.1 - График изменения кода чистоты

На рисунке 3.2 приведен график, показывающий быстродействие схемы. Так на 25-ой наносекунде переменная $r2$, хранящая адрес, равна $001h$. На рисунке 3.3, приведен фрагмент файла *sinApp2.txt*, содержащего рассчитанные значения функции. Как можно увидеть из этих рисунков, только на 55-ой наносекунде, т. е. на третьем такте синхросигнала, выходной сигнал получает значение $1921Fh$ из памяти по этому адресу. Из этого можно сделать вывод, что данная схема имеет меньшее быстродействие, чем при реализации интервала 2π .

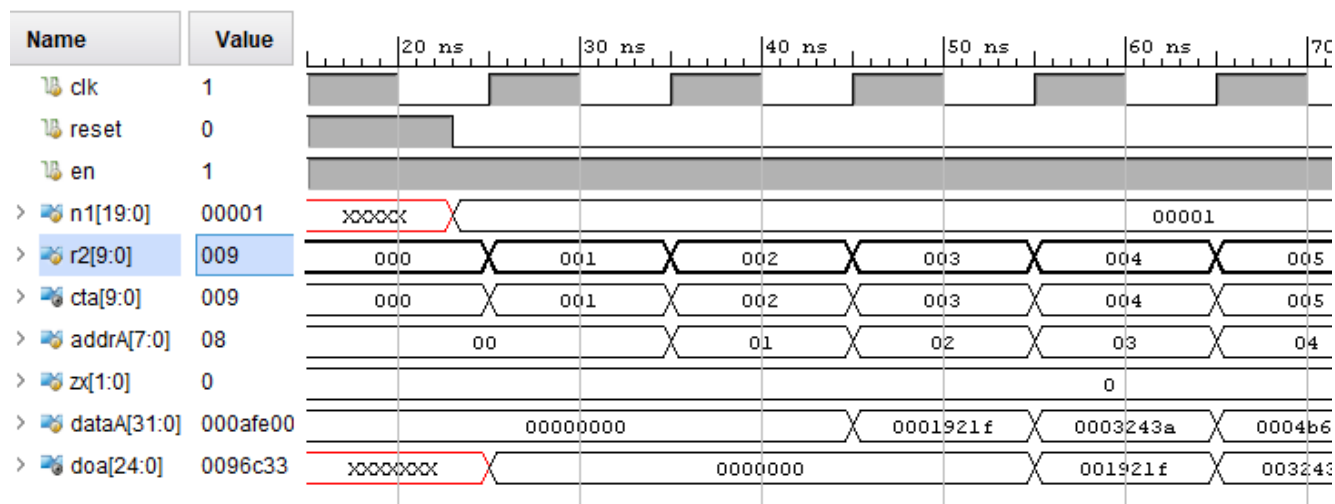


Рисунок 3.2 - График задержки вычисления значения синусоиды

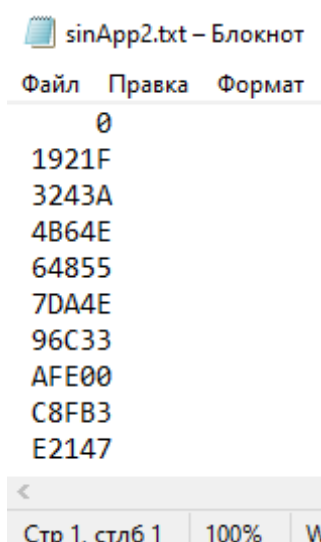


Рисунок 3.3 - Фрагмент файла со значениями функции

3.4 Реализация схемы *DDS* на ресурсах ПЛИС

Для оценки размещения схемы в ПЛИС, была выбрана система *ISE Design Suite 14.7* от фирмы *Xilinx*. Она позволяет удобно проверить, какие и сколько ресурсов ПЛИС использует схема.

Оценка размещения была проведена для разной разрядности адреса от 8 до 18 бит. На основе полученных данных были построены диаграммы для разрядностей от 8 до 16 бит. При разрядности адреса в 18 бит, в плате *Artix-7*, используемой для размещения, не хватило блоков памяти.

На рисунке 3.4 приведена диаграмма, показывающая изменение количества используемых секций платы, при увеличении разрядности адреса. Как видно из рисунка зависимость линейная.



Рисунок 3.4 - Зависимость количества секций платы от разрядности адреса

На рисунке 3.5 приведена диаграмма, показывающая изменение количества используемых таблиц алгебры логики, при увеличении разрядности адреса. Как видно из рисунка зависимость линейная.

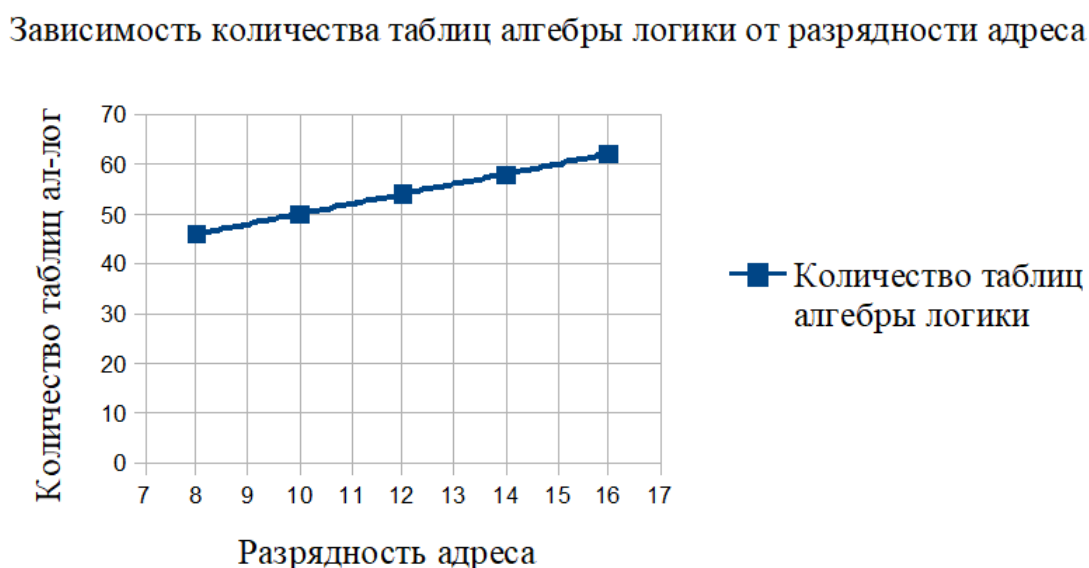


Рисунок 3.5 - Зависимость количества таблиц алгебры логики от разрядности адреса

На рисунке 3.6 приведена диаграмма, показывающая изменение количества используемых входов/выходов, при увеличении разрядности адреса. Как видно из рисунка зависимость также линейная.

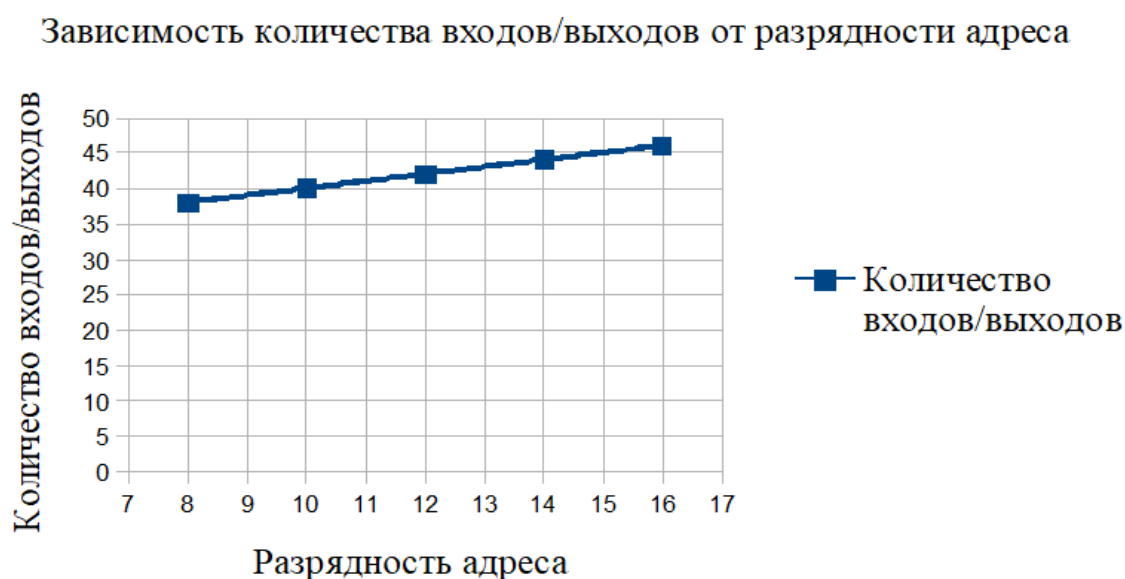


Рисунок 3.6 - Зависимость количества входов/выходов от разрядности адреса

На рисунке 3.7 приведена диаграмма, показывающая изменение количества используемых блоков памяти, при увеличении разрядности адреса. Как видно из рисунка зависимость экспоненциальная.



Рисунок 3.7 - Зависимость количества блоков памяти от разрядности адреса

Результат синтеза схемы по описанию на языке Verilog и временная диаграмма работы приведены в приложениях Б и В, соответственно.

					ПГУ 1.090301.05.001 ПЗ	Лист
						35
Изм.	Лист	№ докум.	Подпись	Дата		

4 Разработка устройства по методу *CORDIC*

4.1 Разработка схемы, реализующей алгоритм *CORDIC*

В системе *Vivado* от фирмы *Xilinx*, был разработан специализированный блок на основе ПЛИС, предназначенный для генерации синусоидального сигнала.

Блок состоит из двух модулей. Главного модуля *maincrd*, который управляет работой модуля *crd*. И модуля *crd*, который реализует алгоритм *CORDIC*.

В главном модуле, задаются управляющий сигнал *start1*, по сигналу которого начинается вычисление, и значение угла поворота *btarg*, для которого рассчитываются синус и косинус.

```
// Сигнал начала работы
reg start1=0;
// Угол поворота
reg [17:0] btarg=17'h010d0*1;
```

Далее в качестве компонента вызывается модуль *crd*. В него передаются сигнал синхронизации – *clk*, сигнал начала работы – *start1* и угол поворота *btarg*. Назад получают рассчитанные значения косинуса и синуса.

```
// Подключение модуля CORDIC
crd u1 (.clk(clk), .start1(start1), .btarg(btarg),
        .cosRz(cosRz), .sinRz(sinRz));
```

Затем описывается счетчик *cten*. С его помощью изменяют аргумент *btarg* и переключают сигнал *start1*. Тем самым, запуская вычисление новых значений синуса и косинуса.

```
// Работа управляющего счетчика
if(cten==6'h18)
```

```

        cten <= 0;
else
        cten <= cten+6'h01;

// Переключение сигнала начала работы
if(cten[5:0]==6'h03)
        start1 <= 1'b1;
else
        start1 <= 0;// #10;

// Изменение угла поворота
if(cten[5:0]==6'h01)
        btarg <= btarg + 17'h010e5;

```

Полный код модуля *maincrd* приведен в приложении А.

В модуле *crd* задается массив углов для итераций. Данный массив используется для определения угла поворота вектора.

```

// Массив углов
reg [BT_WIDTH-1 : 0] bti [0:N+1];
initial begin
bti[0] = 18'd32767; // bti - угол поворота на i-м шаге
bti[1] = 18'd19344; bti[2] = 18'd10220; bti[3] = 18'd5188;
bti[4] = 18'd2604; bti[5] = 18'd1303; bti[6] = 18'd651;
bti[7] = 18'd325; bti[8] = 18'd162; bti[9] = 18'd81;
bti[10] = 18'd40; bti[11] = 18'd20; bti[12] = 18'd10;
bti[13] = 18'd5; bti[14] = 18'd2; bti[15] = 18'd1;
bti[16] = 18'd0; bti[17] = 18'd0;
end

```

Далее начинается цикл вычислений функций синуса и косинуса. По сигналу *start1* в первом шаге цикла задаются начальные значения переменных. Значению *y_i* присваивается константа, значение *x_i* обнуляется. Аргумент заданного угла

поворота сохраняется без знака и квадранта. Из аргумента заданного угла получают его квадрант.

```
// Начало подготовка цикла
Begin
    // косинус = константа
    yi <= CORDIC_GAIN;
    // синус = 0
    xi <= 0;
    i <= ct8[5:0];
    // Аргумент без знака
    argBt <= {2'b0, btarg[BT_WIDTH-3:0]};
    // Аргумент текущего угла
    btiRez <= 0;
    // Определение, в какой четверти заданный угол
    quadr <= btarg[BT_WIDTH-1 : BT_WIDTH-2];
end
```

Затем начинаются вычисления, текущий угол сравнивается с заданным. Если он больше, то его уменьшают на угол поворота текущего шага, иначе – увеличивают. Одновременно с этим, высчитываются x_i и y_i текущего шага. Чем больше итераций, тем точнее будет результат.

```
// Вычисление на i-м шаге
if(btiRez > argBt)
    begin
        yi <= yi + (xi >>> i);
        xi <= xi - (yi >>> i);
        btiRez <= btiRez - bti[i];
    end
else
    begin
        yi <= yi - (xi >>> i);
        xi <= xi + (yi >>> i);
    end
```

```

        btiRez <= btiRez + bti[i];
    end

```

После завершения всех итераций, по формулам приведения тригонометрических функций, получают результат для каждого квадранта.

```

// Получение косинуса каждого квадранта
rgcos <= quadr == 2'b00 ? yi :
        quadr == 2'b01 ? -xi :
        quadr == 2'b10 ? -yi :    xi;
// Получение синуса каждого квадранта
rgsin <= quadr == 2'b00 ? xi :
        quadr == 2'b01 ? yi :
        quadr == 2'b10 ? -xi :    -yi;

```

Полный код модуля *crd* приведен в приложении А.

4.2 Отладка схемы **CORDIC**

Для проектирования была выбрана система *Vivado* от фирмы *Xilinx*, в которой представлена удобная система моделирования работы схем.

Алгоритм *CORDIC* достаточно сложный в реализации. Поэтому работоспособность схемы была проверена на большом количестве различных комбинаций.

Ожидаемое поведение схемы заключается в том, что если задать небольшой угол поворота, то синус этого угла будет также небольшим, а косинус, наоборот, примет почти максимально возможное значение.

Чтобы проверить схему на соответствие ожиданиям, была проведена симуляция работы схемы. График симуляции приведен на рисунке 4.1. Аргументу угла *btarg*, для которого вычисляются значения синуса и косинуса, было присвоено значение *00007h*. По сигналу *start1* начались вычисления. Сигнал *btiRez* будет меняться каждый такт, таким образом, чтобы в конце приблизиться к значению аргумента угла *btarg*.

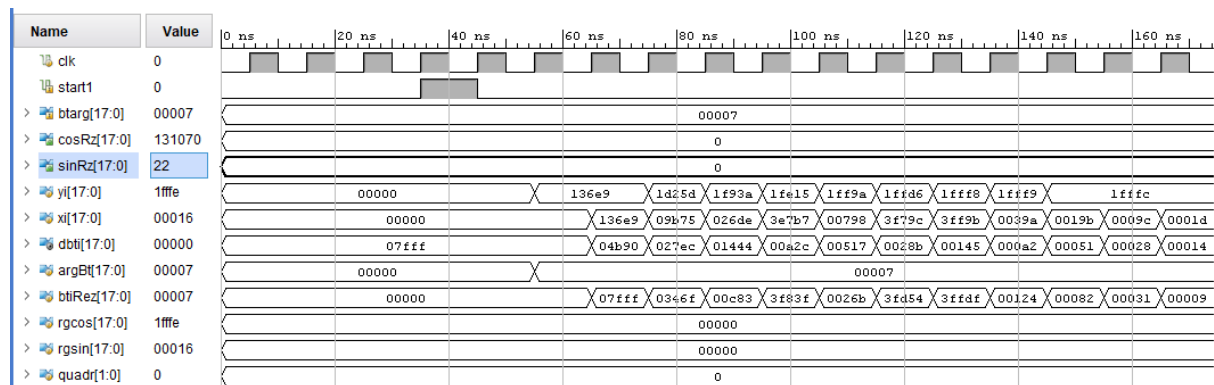


Рисунок 4.1 - График вычисления синуса и косинуса угла 00007h

На рисунке 4.2 приведено продолжение графика для угла 00007h. Сигналы *cosRz* и *rgcos*, показывают значение косинуса заданного угла, в десятичной и шестнадцатеричной системах счисления соответственно. Аналогично, сигналы *sinRz* и *rgsin* показывают значение синуса заданного угла. Как можно увидеть результаты соответствуют ожиданиям.

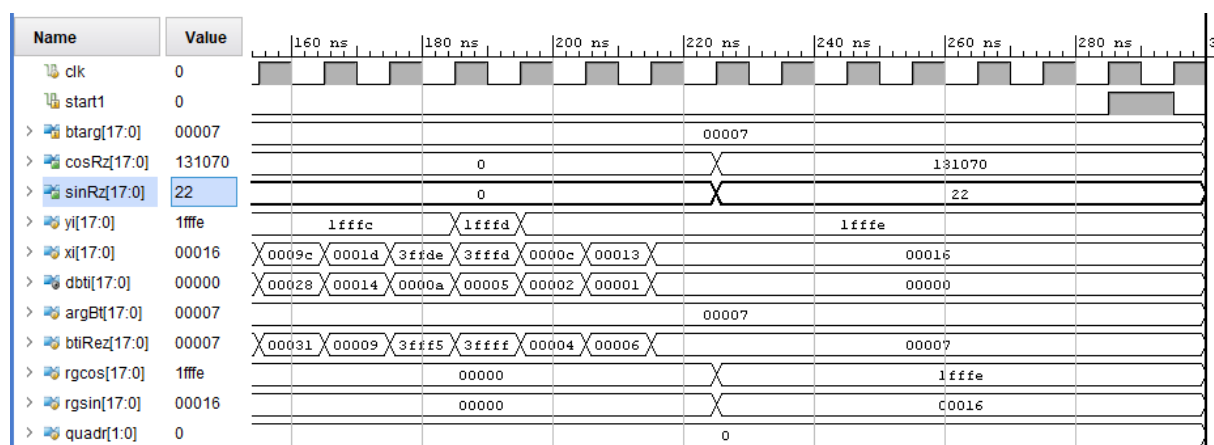


Рисунок 4.2 - Продолжение графика вычисления синуса и косинуса угла 00007h

Кроме этого ожидается, что при увеличении аргумента угла, будет увеличиваться синус и уменьшаться косинус.

На рисунке 4.3 приведен график для большего значение *btarg*, 000F7h. Рассматривается только конец графика. При завершении вычислений сигналы *cosRz* и *rgcos*, показывающие значение косинуса, немного уменьшились.

Аналогично, сигналы *sinRz* и *rgsin*, показывающие значение синуса, несколько увеличились.

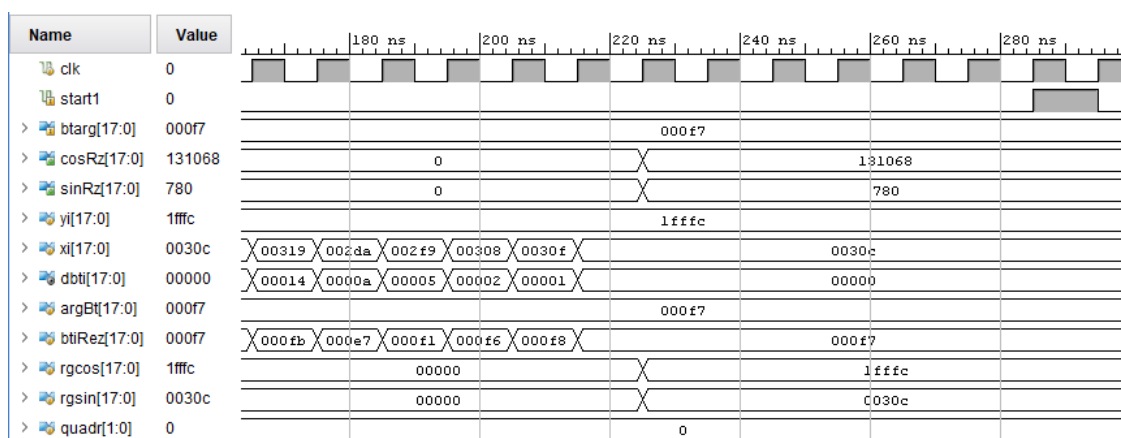


Рисунок 4.3 - График вычисления синуса и косинуса угла 000F7h

Как можно увидеть результаты соответствуют ожиданиям. Чтобы убедиться были проверены ещё несколько значений.

На рисунке 4.4 приведен график для значения 009F7h. При завершении вычислений сигналы *cosRz* и *rgcos* ещё немного уменьшились. А сигналы *sinRz* и *rgsin* увеличились.

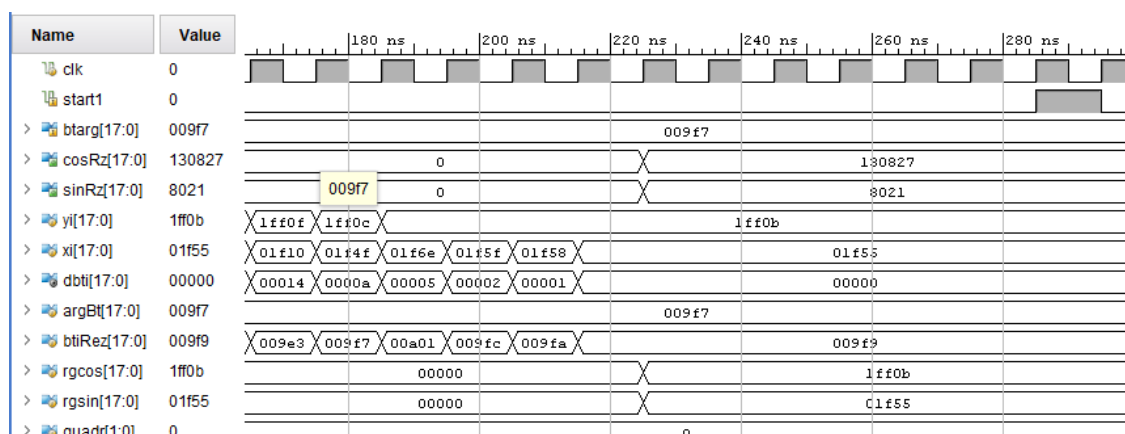


Рисунок 4.4 - График вычисления синуса и косинуса угла 009F7h

Пока, изменение синуса значительно заметнее изменений косинуса, но это связано с тем, что изначально значение синуса было маленьким, и увеличение с двух десятков до восьми тысяч заметнее, чем уменьшение с 131 тысячи до 130

тысяч для значений косинуса. Далее изменения косинуса должны протекать заметнее.

На рисунке 4.5 приведен график для значения *btarg* равном *079F7h*. При завершении вычислений сигналы *cosRz* и *rgcos*, показывают заметное уменьшение своих значений, а сигналы *sinRz* и *rgsin*, наоборот, заметно увеличились.

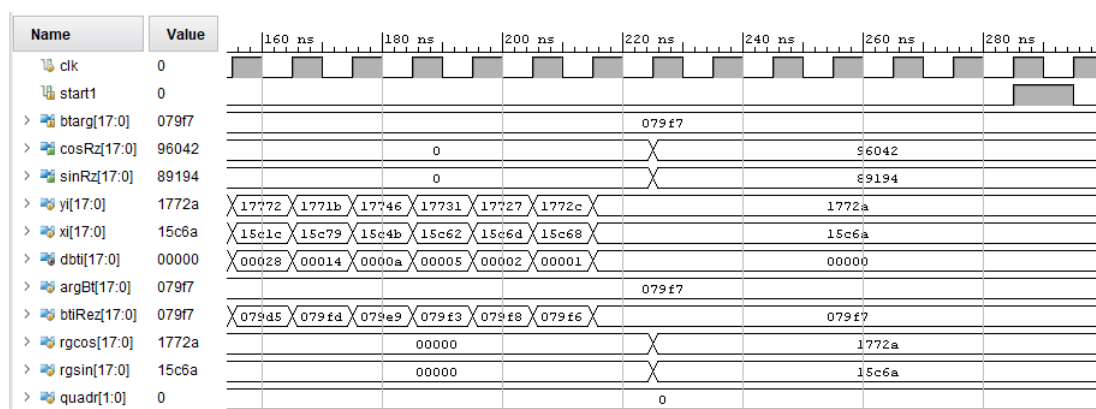


Рисунок 4.5 - График вычисления синуса и косинуса угла *079F7h*

При таком значении угла *btarg*, синус и косинус почти сравнялись, при дальнейшем его увеличении, уже синус будет больше косинуса. И так продолжится до достижения косинуса нуля или близкого к нему значения. Значение синуса при этом, приблизится к максимальному.

На рисунке 4.6 приведен график для значения *btarg*, *0C9F7h*. По завершению вычислений, как и предполагалось, значение сигналов *sinRz* и *rgsin*, увеличились достаточно, чтобы превзойти значения сигналов *cosRz* и *rgcos*.

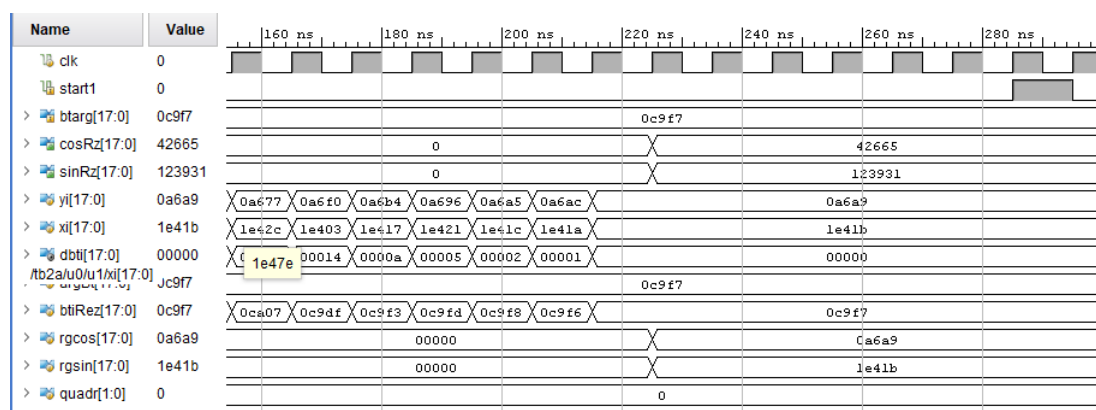


Рисунок 4.6 - График вычисления синуса и косинуса угла *0C9F7h*

На рисунке 4.7 приведен график для значения *btarg*, 0FFFFh. При завершении вычислений значение сигналов *cosRz* и *rgcos* приблизилось к нулю, значения сигналов *sinRz* и *rgsin* увеличились до максимально возможных.

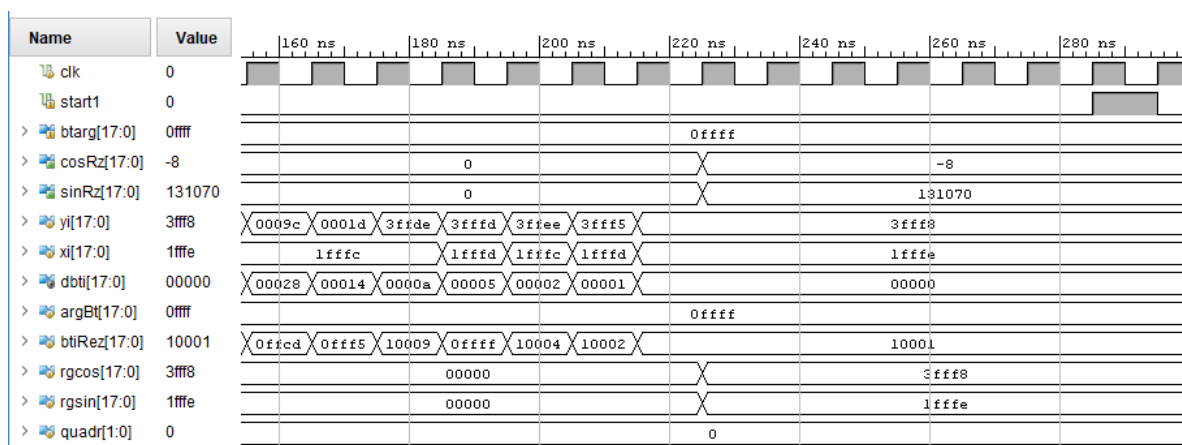


Рисунок 4.7 - График вычисления синуса и косинуса угла 0FFFFh

После проверки работоспособности схемы, была сгенерирована синусоида. Ее график представлен на рисунке 4.8. Как можно увидеть из графика, синусоида генерируется верно, сигнал *quadr* показывает четверть периода, при *quadr* равном нулю генерируется первая четверть, от 0 до $\frac{\pi}{2}$. Аналогично генерируются следующие четверти.

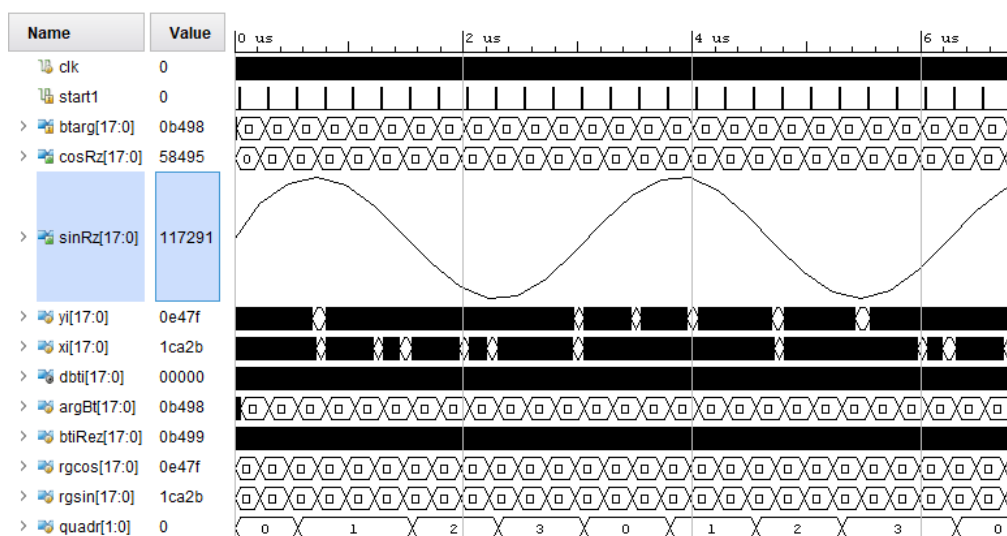


Рисунок 4.8 - График построения синусоиды

На рисунке 4.9 приведен тот же график, но в большем масштабе. Он показывает, как именно строится синусоида с помощью алгоритма *CORDIC*. По сигналу *start1* вычисляется значение синуса текущего угла поворота *btarg*. После вычислений заданный угол увеличивается на значение константы. Чем больше константа, тем выше частота синусоиды, но ниже точность. В случае переполнения *btarg*, старший бит отбрасывается. На графике видно, что на третьей микросекунде произошло переполнение, и следующее значение *btarg* стало меньше предыдущего, как видно из графика, это не повлияло на синусоиду.

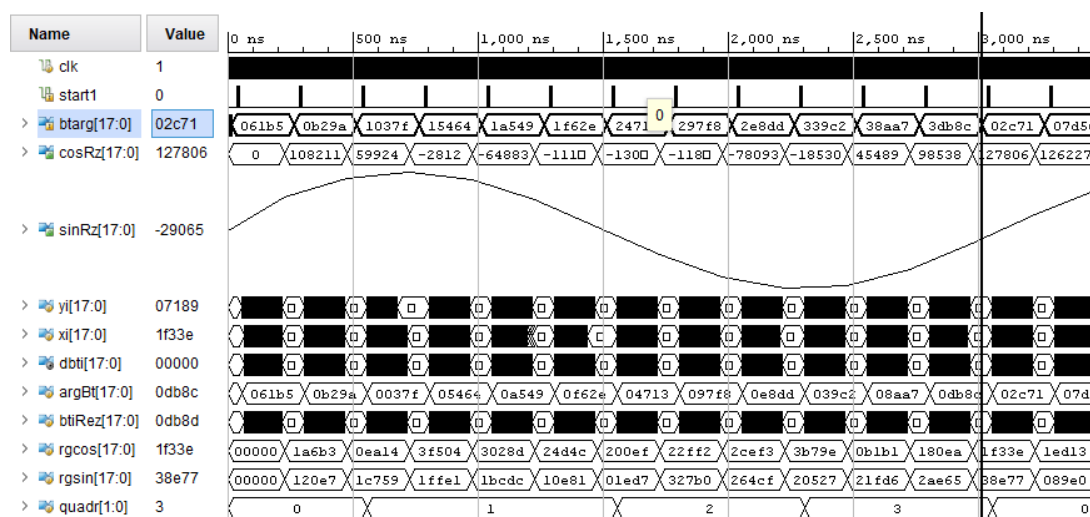


Рисунок 4.9 - График построения синусоиды с большим масштабом

5 Безопасность жизнедеятельности

В данном разделе рассматриваются травмирующие и вредные факторы, возникновение которых потенциально возможно при работе с ЭВМ, а также способы их минимизации.

5.1 Характеристика потенциальных опасностей и вредностей, которые могут возникнуть при работе с ЭВМ

Персональные электронно-вычислительные машины незаменимы для решения многих разнообразных задач, но также они могут причинить вред пользователю и находящимся вблизи людям. Для минимизации травмирующих последствий пользователям необходимо строго придерживаться рекомендуемых правил безопасности, таких как: нормирование времени нахождения на рабочем месте, правильное расположение рабочего места, соблюдать нормы уровней освещенности и шума внутри помещения.

5.2 Эргономические требования к рабочему месту пользователя

В задачи эргономики входит предупреждение профессиональных заболеваний, и улучшение общего состояния здоровья людей. Исследования в этой области показывают, что возникновение многих профессиональных заболеваний часто связано с неудачной конструкцией орудий труда, машин и механизмов, неправильной или недостаточно продуманной организацией рабочего места. Таким образом, задача эргономики предотвратить ошибки проектирования, создавая такие орудия и механизмы, которые в максимальной степени отвечали бы потребностям производства и соответствовали возможностям самого человека.

При конструировании рабочего места должны соблюдаться следующие основные условия:

— по отношению к световым проемам, рабочее место с ПЭВМ расположено так, чтобы естественный свет падал сбоку, преимущественно слева;

					ПГУ 1.090301.05.001 ПЗ	Лист
						45
Изм.	Лист	№ докум.	Подпись	Дата		

- оконные проемы оборудованы жалюзи;
- при организации рабочего места обеспечено соответствие эргономическим требованиям каждого элемента рабочего места и их взаимного расположения с учетом характера выполняемой деятельности, комплексности технических средств, форм организации труда и основного рабочего положения;
- конструкция рабочего стола обеспечивает оптимальное размещение на рабочей поверхности используемого оборудования.

При конструировании и размещении рабочего места должны быть предусмотрены меры, снижающие преждевременное утомление, предотвращающие возникновения психофизиологического стресса. Конструкция рабочего места должна обеспечивать быстроту, безопасность, простоту и экономичность технического обслуживания в нормальных и аварийных условиях.

5.3 Электробезопасность при работе со средствами вычислительной техники

Электрические устройства, к которым относится оборудование ЭВМ, представляют для человека большую потенциальную опасность, так как в процессе эксплуатации или проведения профилактических работ человек может коснуться частей, находящихся под напряжением. Важно отметить специфичность опасности электрооборудования, оказавшиеся под напряжением в результате повреждения изоляции, части устройства, не подают каких-либо сигналов, которые предупреждали бы человека об опасности. Реакция человека на электрический ток возникает лишь при протекании последнего через тело человека.

При прохождении через тело человека, электрический ток оказывает на него сложное воздействие, вызывая термические, электролитические, механические и биологические повреждения. Термические повреждения током проявляется в ожогах на отдельных участках тела и нагреве тканей, что вызывает в них функциональные расстройства. Электролитические повреждения током

выражаются в разложении органической жидкости и крови, и в изменении их физико-химического состава. Механические повреждения током приводят к разрывам мышечных тканей. Биологические – заключаются в способности тока раздражать и возбуждать живые ткани организма

Перечисленные выше повреждения принято называть электрической травмой. Электрическая травма – это повреждение организма, вызванное воздействием электрического тока или электрической дуги.

Электробезопасность напрямую зависит от параметров окружающей среды в помещениях. Опасность поражения электрическим током повышается в таких условиях, как:

- повышенная влажность, в том числе, относительная влажность воздуха выше 75 процентов. А также наличие в помещении паров или конденсата;
- превышение температуры длительно 35 градусов Цельсия, или кратковременно 40 градусов Цельсия;
- наличие токопроводящей пыли;
- токопроводящий материал пола;
- и другие.

Тяжесть поражения электрическим током зависит от плотности и площади контакта человека с частями, находящимися под напряжением.

Для минимизации рисков поражения электрическим током рекомендуется работать в помещениях, соответствующих следующим критериям:

- влажность воздуха не превышает 60 процентов;
- температура воздуха не выше 30 градусов Цельсия, рекомендованный диапазон от 18 до 20 градусов Цельсия;
- помещение должно проветриваться;
- полы выполнены из диэлектрического материала.

Заключение

В ходе реализации данной работы были изучены различные методы генерации синусоидальных сигналов и последовательностей, в частности, табличные методы и метод «цифра за цифрой», *CORDIC*.

Были получены навыки разработки цифровых устройств на базе программируемых логических интегральных схем, ПЛИС. Также были усовершенствованы навыки разработки консольных программ.

Во время выполнения данной работы были разработаны: программы построения таблиц значений тригонометрических функций, устройства генерации синусоидальных сигналов и последовательностей на базе ПЛИС фирмы *Xilinx* семейства *Artix-7*.

Разработанные устройства могут быть применены в специализированных системах цифровой обработки сигналов и встраиваемых вычислительных системах, где необходимо использование синусоидальных сигналов.

В дальнейшем, возможно усовершенствование алгоритмов разработанных схем.

					ПГУ 1.090301.05.001 ПЗ	Лист
						48
Изм.	Лист	№ докум.	Подпись	Дата		

Список использованных источников

1. Немудров В. Системы на кристалле / В. Немудров, Г. Мартин. – М. : Техно-сфера, 2004. – 216 с.
2. Максфилд, К. Проектирование на ПЛИС. Курс молодого бойца / К. Максфилд. – М. : ДОДЭКА, 2007. – 408 с.
3. Шагурин, И. И. Системы на кристалле. Особенности реализации и перспективы применения / И. И. Шагурин // Электронные компоненты. – 2009. – № 1. – С. 37–39.
4. Байков, В. Д. Аппаратурная реализация элементарных функций в ЦВМ / В. Д. Байков, В. Б. Смолов. – Л. : Изд-во Ленингр. ун-та, 1975. – 96 с.
5. Функционально ориентированные процессоры / А. И. Водяхо, В. Б. Смолов, Д. В. Плюснин, Д. В. Пузанков; под ред. В. Б. Смолова. – Л. : Машиностроение, Ленингр. отд-е, 1988. – 224 с.
6. Stork, M. Sine Approximation for Direct Digital Frequency Synthesizers and Function Generators / M. Stork // Proceedings of the 3rd International conference on Circuits, Systems, Control, Signals (CSCS'12). – Barcelona, Spain, 2012. – P. 127–132.
7. Nouman, Z. Design and Implementation A Digital Sine-Cosine Generator Based FPGA / Z. Nouman, B. Klima // Knobloch International Journal of Engineering and Advanced Technology (IJEAT). – 2013. – Vol. 2, June. – P. 304–307.
8. Jassim, M. Abdul-Jabbar and Noor N. Qaqos, Performance Analysis of Single-Multiplier Digital Sine-Cosine Generators / M. Jassim, N. Qaqos // Al-Rafidain Engineering. – 2012. – Vol. 20, № 4. – P. 32–46.
9. United States Patent 4860238, MKI G06F 3/00. Digital Sine Generator / A. Kraker, appl. 28.01.1987, publ. 22.08.1989.
10. Лайонс, Р. Цифровая обработка сигналов / Р. Лайонс. – М. : Бином-Пресс, 2006. – 656 с.

Приложение А

(Обязательное)

Устройство реализации тригонометрических функций на ПЛИС

Листинги программ

					ПГУ 1.090301.05.001 ПЗ	Лист
						50
Изм.	Лист	№ докум.	Подпись	Дата		

Табличный метод

Программа sinApp

```
// Подключение библиотек
#include <iostream>
#include <fstream>
#include <iomanip>

// Использование пространства имён std
using namespace std;

int main()
{
    // Создание потока для записи в файл
    ofstream out("E:\\sinApp.txt");

    // Вычисление числа пи
    const float PI = acos(-1.0);

    // Инициализация необходимых переменных
    float x = 0;
    short y = 0;
    int M = 32767;

    // Вычисление и запись 256 значений y(x)
    for (int i = 0; i < 256; i++)
    {
        // Вычисление x-итого по формуле  $x_i = 2\pi * i / 256$ 
        x = (2 * PI * i) / 256;

        // Вычисление y(x) по формуле  $y_i = M * \sin(x_i)$ 
        y = round(M * sin(x));

        // Вывод в файл форматированного значения y
        out << hex                // шестнадцатеричная с. сч.
```

```

        << uppercase          // верхний регистр
        << setw(4)            // четыре символа
        << setfill(' ')      // заполнение пробелом
        << y
        << endl;
    }

    // Сообщение о завершении работы программы.
    cout << "End of program" << std::endl;

```

Схема управляющего модуля topm1

```

`timescale 1ns / 1ps

// Начало модуля
// Входы и выходы схемы
module topm1( input clk,
              input reset,
              output signed [15:0] qs );

// Переменная счетчика
reg [7:0] ct8 = {8{1'b0}};

// Цикл увеличения счетчика
always @(posedge clk)
    ct8 <= ct8 + 8'h01; // Чем больше константа, тем выше частота

// Шины данных
wire signed [31:0] qsin;
wire [7:0] x;

// Подключение модуля генерации синуса
fsin u1 ( .x(x), .clk(clk), .reset(reset), .qsin(qsin) );

// Соединение сигналов

```

```
assign qs=qsin[15:0];
assign x=ct8;
```

```
// Конец модуля
endmodule
```

Схема модуля fsin

```
`timescale 1ns / 1ps
```

```
// Начало модуля
// Входы и выходы схемы
module fsin( input [7:0] x,
             input clk,reset,
             output reg signed [31:0] qsin );
```

```
// Инициализация блока памяти
reg signed [31:0] roms [255:0];
initial $readmemh("sinApp.txt", roms);
```

```
// Инициализация переменной адреса
reg unsigned [7:0] addr;
initial addr=0;
```

```
// Цикл получения значений из памяти
always @ (posedge clk)
begin
    if (reset == 1)
        // Обнуление адреса, при сигнале сброса
        addr <= 0;
    else
        // Присвоение адресу значения,
        // полученного из управляющего модуля
        addr <= x;
    // Получение значения из таблицы по адресу
```

					ПГУ 1.090301.05.001 ПЗ	Лист
						53
Изм.	Лист	№ докум.	Подпись	Дата		

```

        qsin <= (roms[addr]);
end

// Конец модуля
Endmodule

```

Метод DDS

Программа sinAppDDS

```

// Подключение библиотек
#include <iostream>
#include <fstream>
#include <iomanip>

// Использование пространства имён std
using namespace std;

int main()
{
    // Создание потока для записи в файл
    ofstream out("E:\\sinApp2.txt");

    // Вычисление числа пи
    const float PI = acos(-1.0);

    // Инициализация необходимых переменных
    float x = 0;
    int y = 0;
    int M = 16777215;

    // Вычисление и запись 256 значений y(x)
    for (int i = 0; i < 256; i++)
    {
        // Вычисление x-итого по формуле  $x_i = 2\pi * i / 1024$ 

```

```

x = (2 * PI * i) / 1024;

// Вычисление y(x) по формуле yi= M * sin(xi)
y = round(M * sin(x));

// Вывод в файл форматированного значения y
out << hex                // шестнадцатеричная с. сч.
    << uppercase          // верхний регистр
    << setw(6)             // шесть символов
    << setfill(' ')        // заполнение пробелом
    << y                   // выводимая переменная
    << endl;
}

// Сообщение о завершении работы программы.
cout << "End of program" << std::endl;
}

```

Схема реализации метода DSS

```

`timescale 1ns / 1ps

// Начало модуля
module p03v( clk, reset, en, n1, doa);

// Разрядность вых.данных
parameter RAM_WIDTH = 25;
// Разрядность адреса
parameter ADDR_BITS = 8;
// Вес разрядности кода частоты
parameter DR = 0;

// Входы схемы
input clk,reset,en;

```

```

// Код частоты
input [19:0] n1;

// Выход синус
output reg [RAM_WIDTH-1:0] doa;

// Массив памяти 25 x 2**8
reg [RAM_WIDTH-1:0] ram1 [(2**ADDR_BITS)-1:0];

// Выход накапливающего сумматора
reg [ADDR_BITS+1+DR:0] r2=0;

// Разряды r2
wire [ADDR_BITS+1:0] cta;

// Адрес на входе ПЗУ [15:0]
reg [ADDR_BITS-1:0] addrA;

// Данные на выходе ПЗУ [24:0]
reg [RAM_WIDTH-1:0] dataA;

// Переменные квадрантов (00, 01, 10, 11)
reg [1:0] zx=0, zx0=0;

// Константа адреса (все единицы)
reg [ADDR_BITS-1:0] c111={ADDR_BITS{1'b1}};

// Константа данных (все единицы)
reg [RAM_WIDTH-1:0] r111={RAM_WIDTH{1'b1}};

// Инициализация памяти
initial
    $readmemh("sinApp2.txt", ram1, 0, (2**ADDR_BITS)-1);

// Сохранение значения счетчика

```

```

assign cta = r2[ADDR_BITS+1+DR:DR];

// Цикл
always @(posedge clk) begin
    // Если сигнал сброса,
    // обнулить переменные адреса, счетчика и данных
    if(reset==1) begin
        addrA<=0; r2<=0; dataA <= 0;
    end
    // Иначе, если сигнал разрешения, генерация синуса
    else if (en==1) begin
        // увеличение счетчика
        r2 <= r2 + n1;
        // Определение квадранта, старшие два бита
        zx0 <= cta[ADDR_BITS+1:ADDR_BITS];
        // Сохранение предыдущего квадранта
        zx<=zx0;
        // Если первый квадрант,
        // берём значение счетчика без изменений,
        // исключая два старших бита
        if(cta[ADDR_BITS+1:ADDR_BITS]==2'b00)
            addrA<=cta[ADDR_BITS-1:0];
        // Если второй квадрант, берём инверсию значение счетчика
        // исключая два старших бита
        else if(cta[ADDR_BITS+1:ADDR_BITS]==2'b01)
            addrA<=~cta[ADDR_BITS-1:0];
        // Если третий квадрант,
        // берём значение счетчика без изменений,
        // исключая два старших бита
        else if(cta[ADDR_BITS+1:ADDR_BITS]==2'b10)
            addrA<=cta[ADDR_BITS-1:0];
        // Если четвертый квадрант,
        // берём инверсию значение счетчика
        // исключая два старших бита
        else

```

```

        addrA<=c111-cta[ADDR_BITS-1:0];

// Сохраняем значение из памяти,
// по определенному выше адресу
    dataA <= ram1[addrA];
// Если квадранты 1 или 2,
// используем значение без изменений
    if (zx==0)
        doa <= dataA;
    else if (zx==1)
        doa <= dataA;
// Если квадранты 3 или 4, используем инверсию значения
    else if (zx==2)
        doa <= r111-dataA;
    else
        doa <= r111-dataA;
    end
end

// Конец модуля
endmodule

```

Метод CORDIC

Схема управляющего модуля maincrd

```

`timescale 1ns / 1ps

// Начало модуля
module maincrd(input clk );

// Сигнал начала работы
reg start1=0;
// Угол поворота
reg [17:0] btarg=17'h010d0*1;

```

					ПГУ 1.090301.05.001 ПЗ	Лист
						58
Изм.	Лист	№ докум.	Подпись	Дата		

```

// Результаты работы
wire [17:0]cosRz,sinRz;
// Счетчики
reg [31:0] ct=0;
reg [5:0] cten=0;

// Подключение модуля CORDIC
crd u1 (.clk(clk), .start1(start1), .btarg(btarg), .cosRz(cosRz),
.sinRz(sinRz));

// Цикл счетчиков
always@(posedge clk)
begin
    // Счетчик тактов
    if(ct==32'h00001000)
        ct <= 0;
    else
        ct <= ct+32'h00000001;

    // Работа управляющего счетчика
    if(cten==6'h18)
        cten <= 0;
    else
        cten <= cten+6'h01;

    // Переключение сигнала начала работы
    if(cten[5:0]==6'h03)
        start1 <= 1'b1;
    else
        start1 <= 0; // #10;

    // Изменение угла поворота
    if(cten[5:0]==6'h01)
        btarg <= btarg + 17'h010e5;
end

```

```
// Конец модуля
endmodule
```

Схема модуля crd, алгоритм CORDIC

```
// Начало модуля
module crd(input wire clk, start1, // cosRz, sinRz - results
          input wire [17:0] btarg,
          output wire [17 : 0] cosRz, sinRz);

// BT, bt - угол бета, btarg - аргумент;
// ищем sinRz=sin(btarg) и cosRz

// Разрядности
localparam N = 16;
localparam DAT_WIDTH = 18;
localparam BT_WIDTH = 18;

// Массив углов
reg [BT_WIDTH-1 : 0] bti [0:N+1];
initial begin
bti[ 0] = 18'd32767; // bti - угол поворота на i-м шаге,
bti[ 1] = 18'd19344; bti[ 2] = 18'd10220; bti[ 3] = 18'd5188;
bti[ 4] = 18'd2604; bti[ 5] = 18'd1303; bti[ 6] = 18'd651;
bti[ 7] = 18'd325; bti[ 8] = 18'd162; bti[ 9] = 18'd81;
bti[10] = 18'd40; bti[11] = 18'd20; bti[12] = 18'd10;
bti[13] = 18'd5; bti[14] = 18'd2; bti[15] = 18'd1;
bti[16] = 18'd0;
end

// Константа
reg signed [DAT_WIDTH-1: 0] CORDIC_GAIN = 18'd79593; // 136e9h
reg signed [DAT_WIDTH-1: 0] yi=0, xi=0;
wire [17:0] dbti;
// argBt-аргумент, btiRez - i-й результат
```

```

reg signed [BT_WIDTH-1: 0] argBt=0, btiRez=0;
// Переменные результата
reg signed [DAT_WIDTH-1: 0] rgcos = 0, rgsin = 0;
// Переменная квадрантов
reg [1:0] quadr=0;
reg [7:0] ct8=0;
reg [5:0] i=0;
reg job=0, jobz=0;

always@(posedge clk)
begin
    jobz <= job;
    // Если сигнал старт, запуск счетчика
    if(start1==1)
        ct8 <= 0;
    else if (ct8 < 8'h11)
        ct8 <= ct8 +1;
    else
        ct8 <= ct8;
    // Если старт, начало работы
    if(start1==1)
        job <= 1;
    // Если счётчик 16, завершение работы
    else if (ct8 == 8'h10)
        job <= 0;
    else
        job <= job;

    if(job==1 && jobz==0)
        // Начало подготовка цикла
        Begin
            // косинус = константа
            yi <= CORDIC_GAIN;
            // синус = 0
            xi <= 0;

```

```

        i <= ct8[5:0];
// Аргумент без знака
        argBt <= {2'b0, btarg[BT_WIDTH-3:0]};
// Аргумент текущего угла
        btiRez <= 0;
// Определение, в какой четверти заданный угол
        quadr <= btarg[BT_WIDTH-1 : BT_WIDTH-2];
    end
else if(job==1 && jobz==1)
    begin
        i <= ct8[5:0];
        if(btiRez > argBt)
            begin
                // Если текущий угол больше заданного, уменьшить
                yi <= yi + (xi >>> i);
                xi <= xi - (yi >>> i);
                btiRez <= btiRez - bti[i];
            end
        else
            begin
                // Иначе, увеличить
                yi <= yi - (xi >>> i);
                xi <= xi + (yi >>> i);
                btiRez <= btiRez + bti[i];
            end
        end
    end
else if(job==0 && jobz==1)
    begin
        // Получение косинуса каждого квадранта
        rgcos <= quadr == 2'b00 ? yi :
                quadr == 2'b01 ? -xi :
                quadr == 2'b10 ? -yi : xi;

        // Получение синуса каждого квадранта
        rgsin <= quadr == 2'b00 ? xi :

```

Изм.	Лист	№ докум.	Подпись	Дата

```

                                quadr == 2'b01 ? yi :
                                quadr == 2'b10 ? -xi :    -yi;

                                end

// Конец always
end

// Соединение сигналов
assign cosRz = rgcos;
assign sinRz = rgsin;
assign dbti = bti[i];
// Конец модуля
Endmodule

```

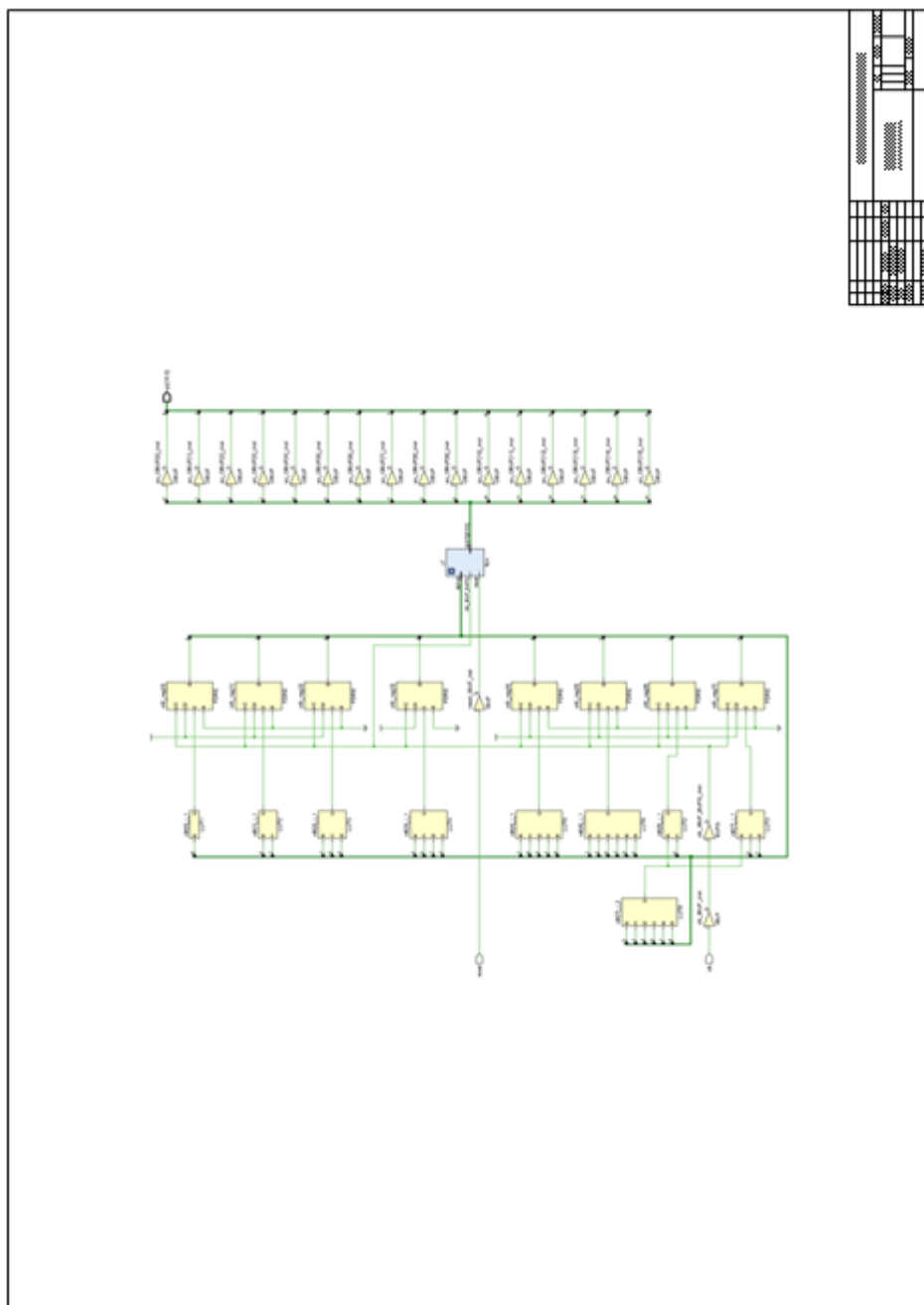
Приложение Б

(Обязательное)

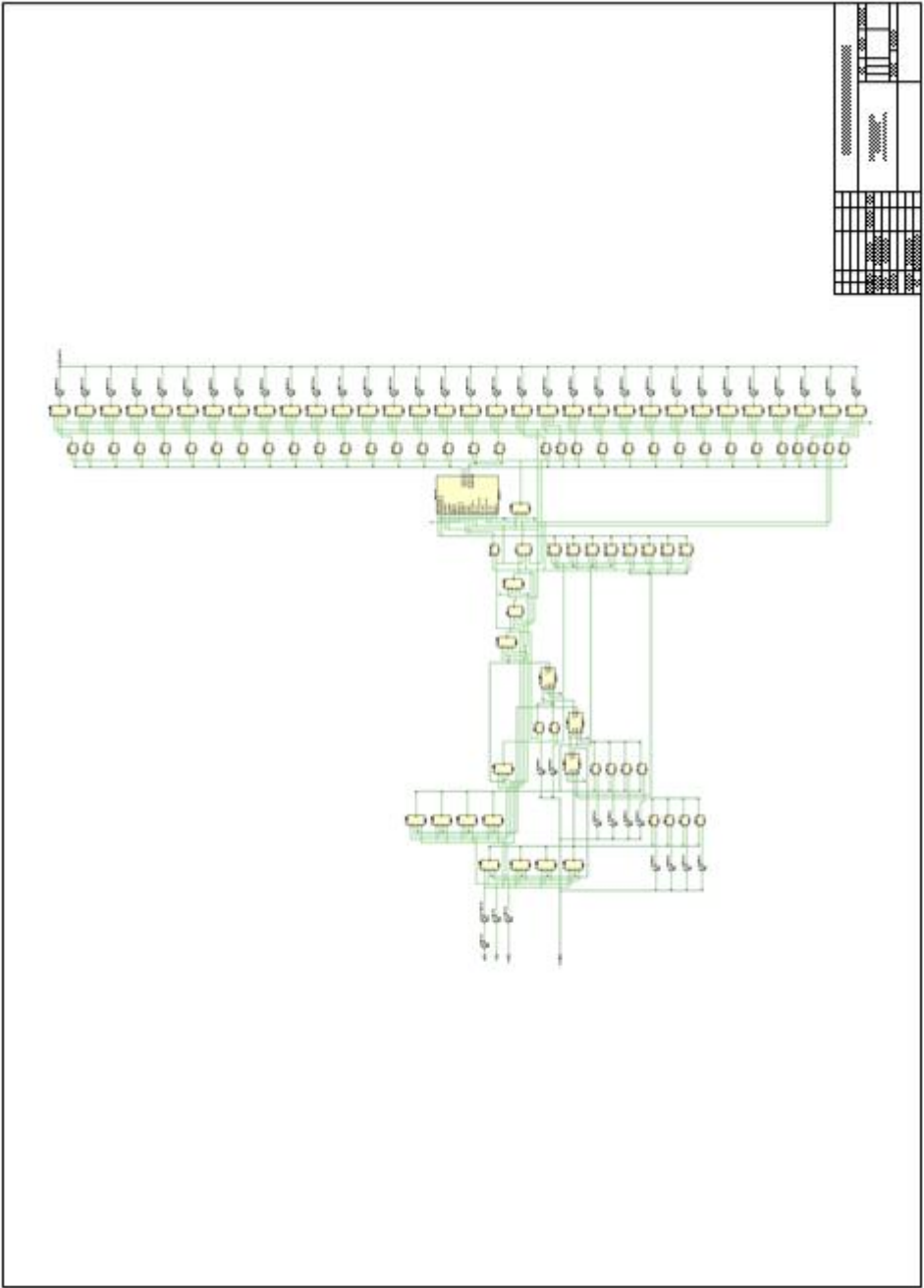
Устройство реализации тригонометрических функций на ПЛИС

Аппаратная реализация

Реализация табличного метода



Реализация метода DDS



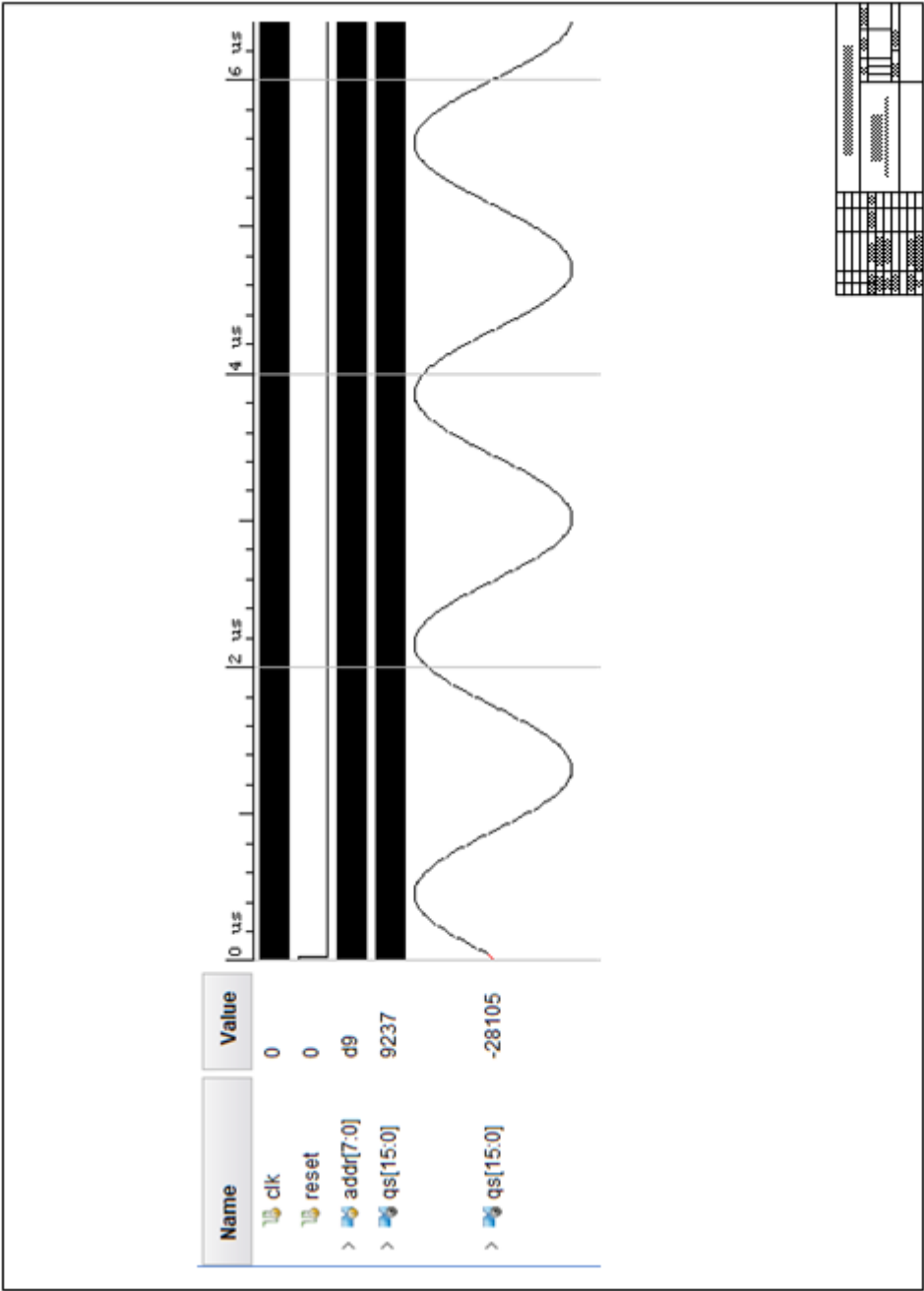
Приложение В

(Обязательное)

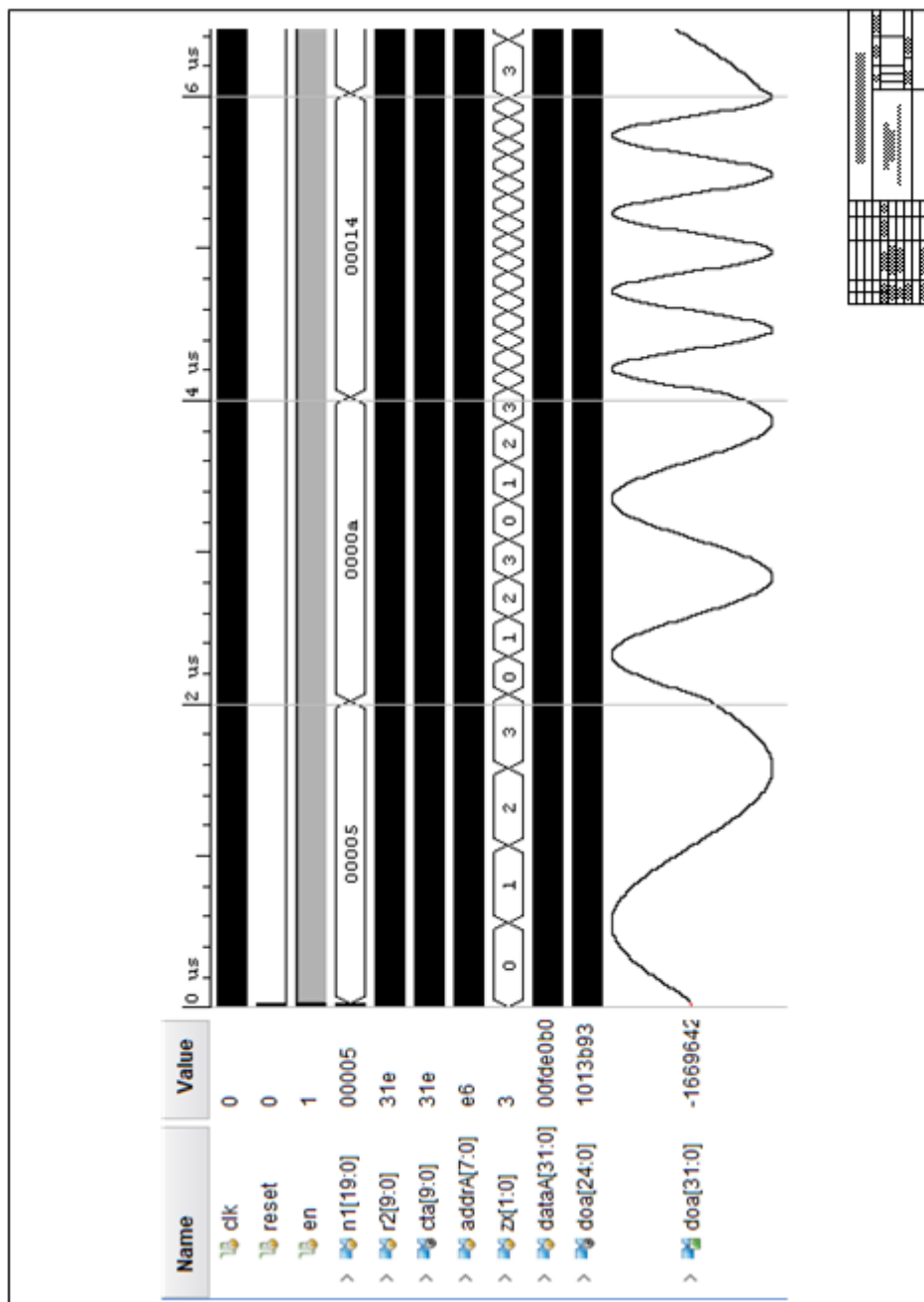
Устройство реализации тригонометрических функций на ПЛИС

Временные диаграммы работы

Временная диаграмма работы табличного метода



Временная диаграмма работы метода DDS



Приложение Г

(Обязательное)

Устройство реализации тригонометрических функций на ПЛИС

Презентация

УСТРОЙСТВО РЕАЛИЗАЦИИ ТРИГОНОМЕТРИЧЕСКИХ ФУНКЦИЙ НА ПЛИС

Автор работы:

Студент группы 18ВВ2 Бровкин Андрей Алексеевич

Научный руководитель:

д.т.н., профессор Гурин Евгений Иванович

Пенза, 2022

Цель работы и объекты

Цель работы – разработать устройства реализации тригонометрических функций на программируемых логических интегральных схемах.

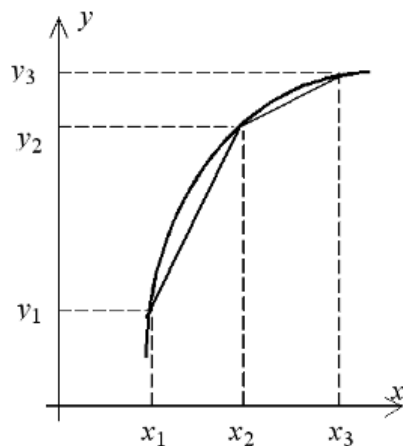
Объекты работы – схемы устройств реализации тригонометрических функций на ПЛИС.

Схема реализации функции синус с помощью табличного метода.

Схема реализации функции синус с помощью модификации табличного метода *DDS*.

Схема реализации функций синус и косинус с помощью метода «цифра за цифрой» или же *CORDIC*

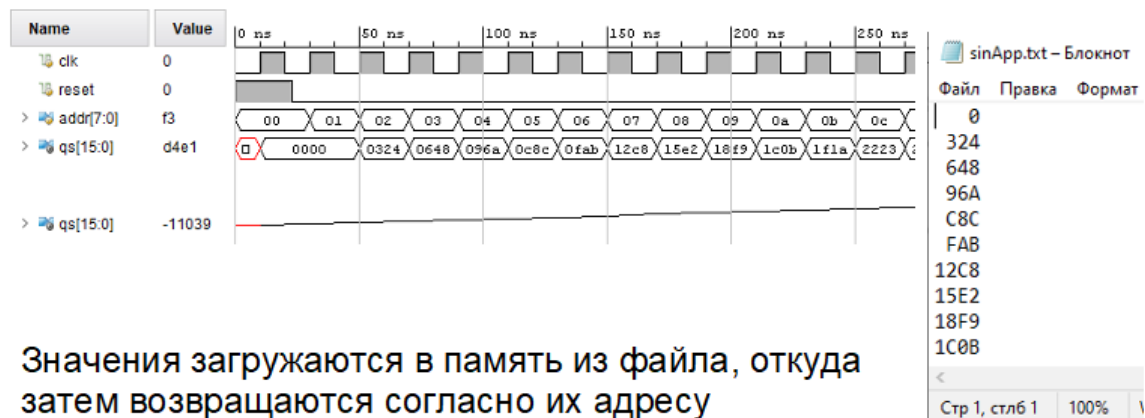
Табличный метод



Суть табличного метода заключается в использовании таблицы заранее рассчитанных значений.

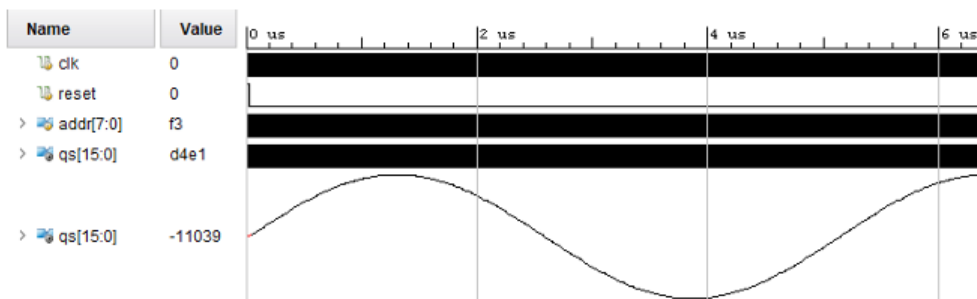
В нашей реализации таблица строится для интервала 2π функции синус

Работа табличного метода

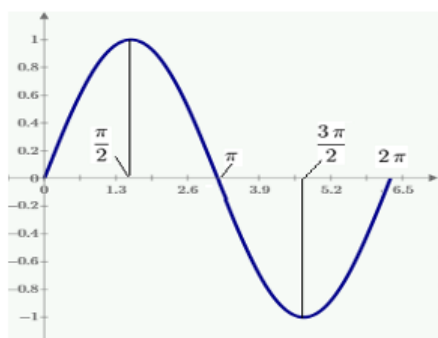


Значения загружаются в память из файла, откуда затем возвращаются согласно их адресу

Временная диаграмма работы табличного метода



Метод *DDS*



Метод *DDS* является модификацией табличного метода он также использует таблицы заранее рассчитанных значений.

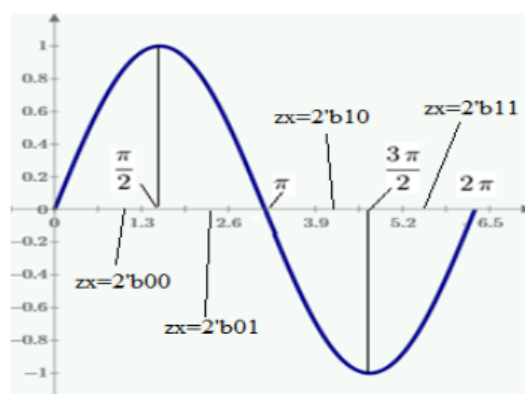
В нашей модификации таблица строится для интервала $\pi/2$ функции синус

Формирование адреса

В квадрантах 2 и 4 абсолютные значения функции синуса убывают.

Для получения адресов их значений из значений первого квадранта используется инверсия адреса.

Адреса значений третьего квадранта берутся без изменений

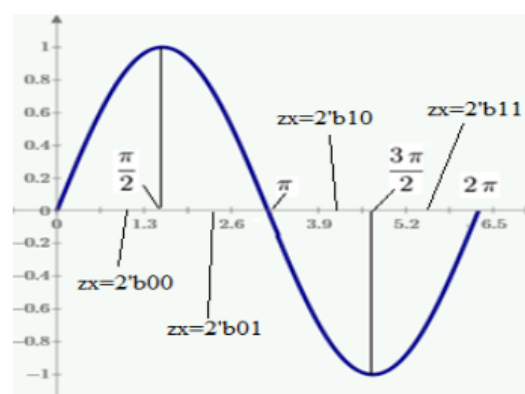


Формирование данных

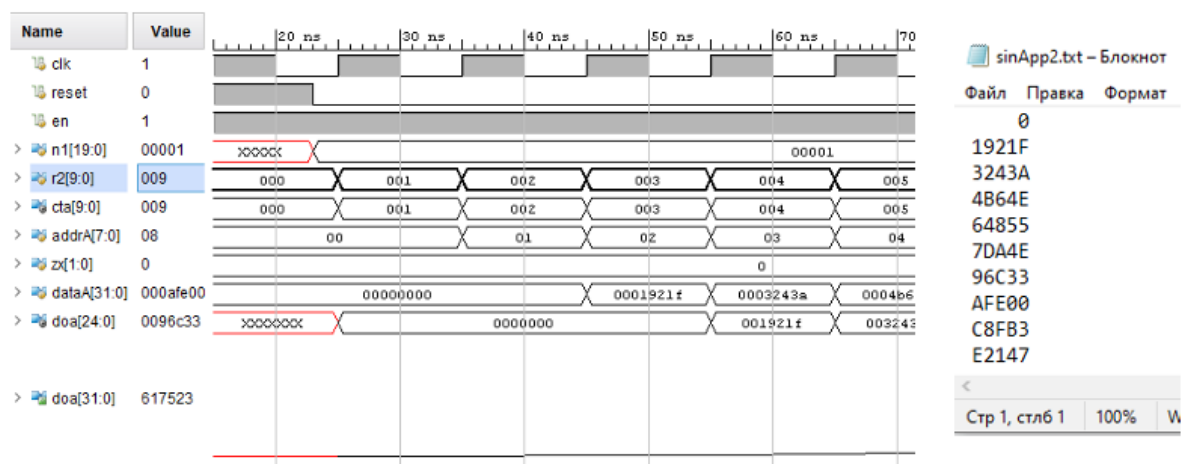
В квадрантах 3 и 4 значения функции синус принимают отрицательные значения.

Для получения их значений из значений первого квадранта используется инверсия данных.

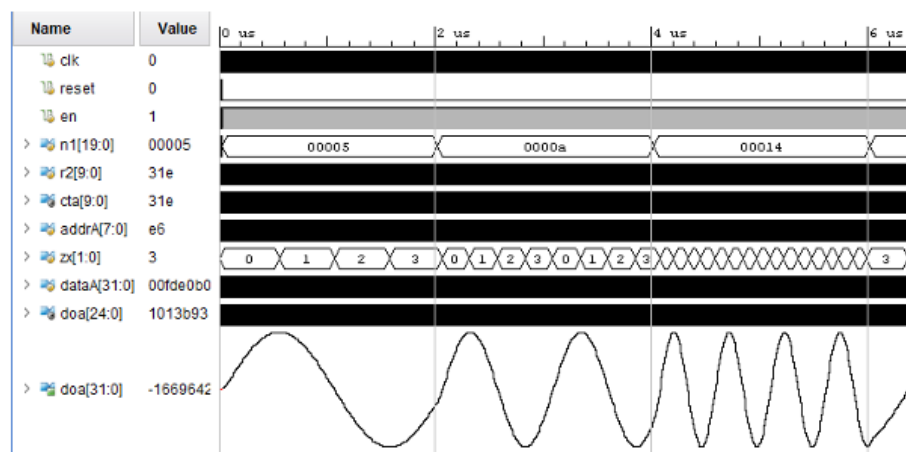
Знак значений 2 квадранта не изменяется.



Работа метода *DDS*

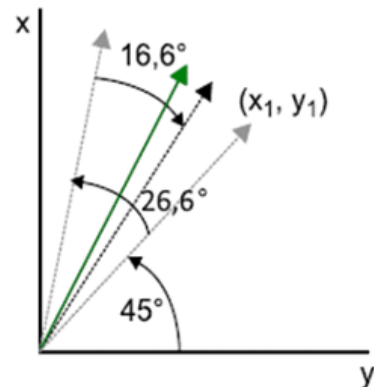


Временная диаграмма работы метода *DDS*



Алгоритм *CORDIC*

CORDIC (*COordinate Rotation Digital Computer* — цифровой вычислитель поворота системы координат; метод «цифра за цифрой») — итерационный метод вычисления сложных функций путем выполнения простых операций (сложения, сдвига и др.).

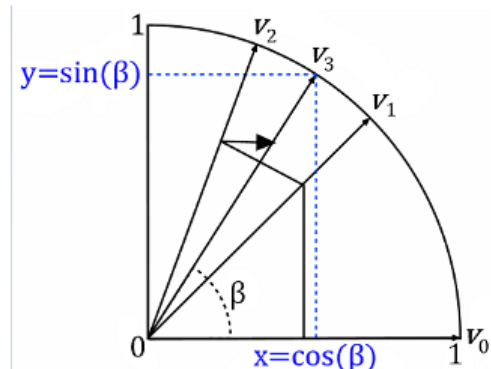


Применение *CORDIC* для вычисления тригонометрических функций

Для определения синуса и косинуса угла β надо найти координаты точки на единичной окружности.

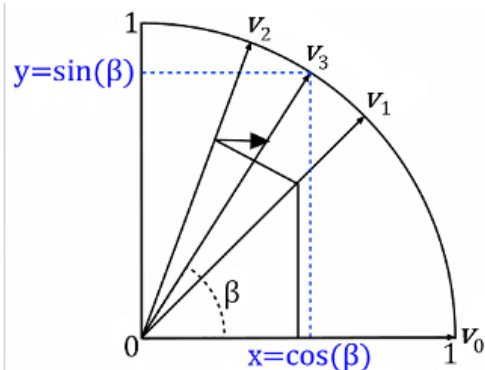
Расчет начинается с вектора v_0

$$v_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$



Применение CORDIC для вычисления тригонометрических функций

Следующий вектор v_1 получается поворотом против часовой стрелки вектора v_0 на угол $\gamma_1 = \pi/4 = 45^\circ$. Вектор v_2 получается поворотом вектора v_1 на угол $\gamma_2 < 45^\circ$ и т. д.



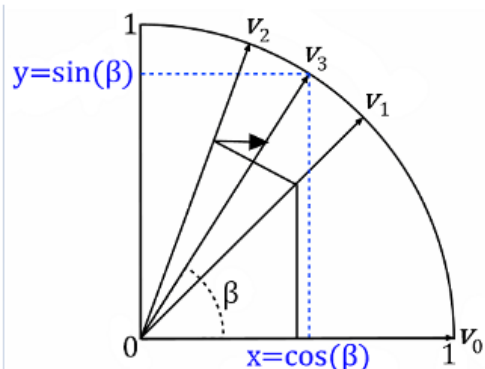
Применение CORDIC для вычисления тригонометрических функций

Координаты каждой итерации высчитываются по следующим формулам,
Где K_n - коэффициент

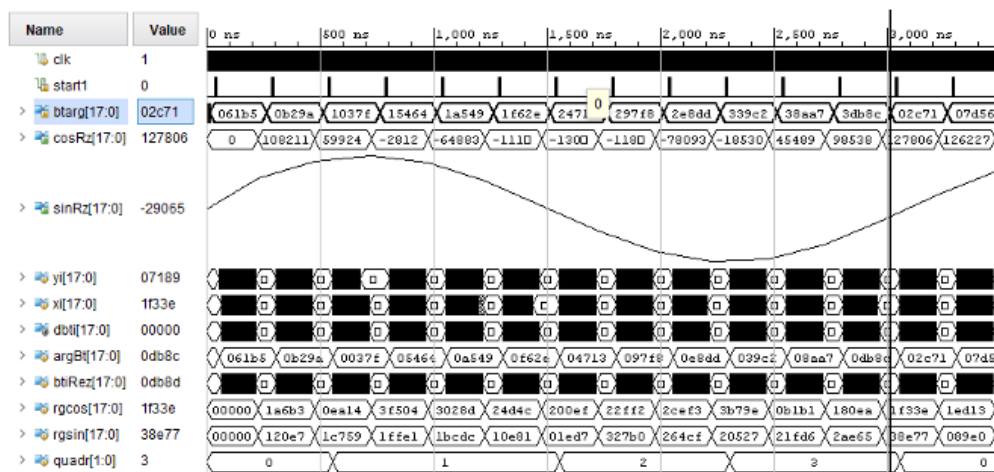
$$x_i = x_{i-1} - \sigma \cdot 2^{-i} \cdot y_{i-1},$$

$$y_i = y_{i-1} + \sigma \cdot 2^{-i} \cdot x_{i-1},$$

$$K_n = \prod_{i=0}^{n-1} K_i = \prod_{i=0}^{n-1} \frac{1}{\sqrt{1+2^{-2i}}}.$$



Временная диаграмма работы алгоритма *CORDIC*



Спасибо за внимание!