МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное   
образовательное учреждение высшего образования

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт компьютерных технологий и информационной безопасности

|  |  |
| --- | --- |
| Руководитель:  Доцент кафедры ИМС  д.п.н., Б.Е. Механцев  \_\_\_\_\_\_\_\_\_\_\_\_  *(подпись)*  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ г. | К защите допустить:  Доцент кафедры ИБТКС  к.т.н., А.П. Плёнкин  \_\_\_\_\_\_\_\_\_\_\_\_\_  *(подпись)*  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ г. |

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К ТВОРЧЕСКОМУ ПРОЕКТУ

ПО ДИСЦИПЛИНЕ «ВвИД»

|  |  |  |
| --- | --- | --- |
| на тему: | Реализация скоростных криптографических систем | |
|  |  | |
| Команда: | CRAFSED | |
|  |  | |
| Выполнили: |  | Нестеренко Пётр Алекссевич, КТбо2-8 |
|  |  | *(подпись, фамилия, имя, отчество, группа)* |
|  |  | Кочубей Даниил Сергеевич, КТбо2-8 |
|  |  | *(подпись, фамилия, имя, отчество, группа)* |
|  |  | Жалнин Дмитрий Игоревич, КТбо2-8 |
|  |  | *(подпись, фамилия, имя, отчество, группа)* |
|  |  | Пучкова Анастасия Денисовна, КТбо2-8 |
|  |  | *(подпись, фамилия, имя, отчество, группа)* |
|  |  | Городилов Никита Сергеевич, КТбо2-10 |
|  |  | *(подпись, фамилия, имя, отчество, группа)* |

Таганрог 2020 г.

**Реферат**

Пояснительная записка содержит 15 страниц, 3 рисунков, 1 таблицы и 5 источников.

Суть проекта заключается в аппаратной реализации криптографического алгоритма “Кузнечик” на программируемой логической интегральной схеме с использованием языка VHDL.

Целью работы является изучение алгоритмов криптографии и основ цифровой схемотехники, а также создание рабочего прототипа устройства способного аппаратно шифровать и расшифровывать сообщения.

**Содержание**

[1 Техническое задание 4](#_Toc53623255)

[2 Введение 5](#_Toc53623256)

[3 Распределение ролей 6](#_Toc53623257)

[4 Реализация криптоалгоритма “Кузнечик” 7](#_Toc53623258)

[4.1 Этапы шифрования сообщений 7](#_Toc53623259)

[4.2 Генерация раундовых ключей 8](#_Toc53623260)

[4.3 Реализация X-функции. Побитовый XOR 9](#_Toc53623261)

[4.4 Реализация S-функции. Нелинейное преобразование 9](#_Toc53623262)

[4.5 Реализация L-функции. Линейное преобразование 10](#_Toc53623263)

[4.6 Генерация раундовых ключей для X-функции. 11](#_Toc53623264)

[5 Результат проделанной работы. 12](#_Toc53623265)

[Список источников 13](#_Toc53623266)

# Техническое задание

Необходимо переложить криптографический алгоритм “Кузнечик” ГОСТ Р 34.12-2015 на ПЛИС (программируемую логическую интегральную схему).

Основным преимуществом такого подхода к решению задачи является возможность распараллеливать задачи, выполнять несколько циклов и итераций алгоритма одновременно, что позволяет достичь в разы большей скорости вычислений.

Описание аппаратуры необходимо провести используя язык VHDL. Так же в процессе проекта необходимо ознакомится с основными принципами работы криптоалгоритмов, а также основам разработки устройство на основе ПЛИС .

Конечный продукт: функционирующий аппаратно-программный комплекс, способный обеспечить скорость шифровки и дешифровки, достаточную для обмена данными в реальном времени.

Потенциальные клиенты: государственные структуры, банки. Плата для разработки Artix-7, Рисунок 1.

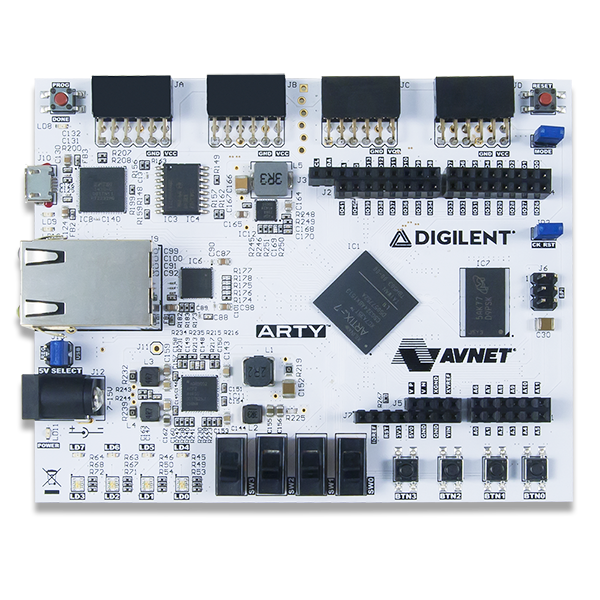


Рис. 1 –Внешний вид платы “Arty A7”

# Введение

Мы каждый день пользуемся интернетом, передаём друг другу сообщения в социальных сетях, посещаем разные сайты. Однако, скрытыми от пользователя остаются многие аспекты этой деятельности. Через чьи сервера проходит трафик? Сколько раз он успевает пересечь границу, побывать в другой стране или даже на другом континенте, прежде чем ваш собеседник получил отправленное вами сообщение? В таких условиях сохранять конфиденциальность помогает шифрование трафика.

Всё это требует вычислительных мощностей. Однако если вашу с собеседником переписку удастся перехватить и расшифровать, вы окажетесь в неприятном положении. Но ситуация окажется катастрофической, если была рассекречена не просто личная переписка, а государственная тайна или иная сверхважная информация. Чтобы такого не происходило для шифрования подобного контента используются более надежные, но и более требовательные к ресурсам алгоритмы. При этом количество информации передаваемой по этим каналам огромно.

В связи с этим программные реализации криптоалгоритмов уже не способны с достаточной скоростью решить эту проблему. Требуется другой подход, переложение криптоалгоритмов на аппаратную основу. Реализованные таким образом специализированные схемы способны в разы быстрее решать подобные задачи. Созданием подобной системы занимается наша команда.

# Распределение ролей

Только слаженные действия всех членов команды могут привести к успеху команды в целом. Выбор ролей для каждого её члена является очень важной и ответственной задачей. Каждый участник проекта должен чётко понимать, какие функции он должен выполнять, какой продукт на выходе получить, на каком этапе сейчас находится проект. Роли должны быть распределены так, чтобы участники команды могли дополнять друг друга, помогать друг другу, таким образом достигая поставленных целей. Если команда разобщена, проект обречён.

Список участников команды с соответствующими им ролями представлен на Таблице 1.

|  |  |
| --- | --- |
| ФИО | Роль в команде |
| Кочубей Даниил Сергеевич | Инженер-программист |
| Нестеренко Пётр Алексеевич | Программист-конструктор (капитан команды) |
| Жалнин Дмитрий Игоревич | Программист |
| Пучкова Анастасия Денисовна | Программистка |
| Городилов Никита Сергеевич | Инженер-программист |

Таблица 1 - распределение ролей

# Реализация криптоалгоритма “Кузнечик”

Реализация, созданная в рамках проекта, отличается от эталонной. Было принято решения, для упрощения отладки и симуляции, сократить размеры блока данных со 128 до 16 бит и блока ключа с 256 до 32 бит. Так же было уменьшено количество раундов, теперь их 4 вместо 10. В остальном алгоритм остался неизменным.

## Этапы шифрования сообщений

На вход алгоритму подаются **128 бит** данных подлежащих шифрованию и ключ длинной **256 бит.** Далее на каждом раунде выполняются**3 основные функции** шифрования:

* X-функция. Побитовое “исключающее или” (далее и везде XOR) блока данных и раундового ключа(его генерацию мы разберём позже).
* S-функция. Нелинейное преобразование. По сути это просто побайтовая замена одних значений на другие в соответствии с таблицей.
* L-функция. Линейное преобразование. Хитрая операция с полиномиальным умножением в полях Галуа. Рассмотрим его подробнее позже. Если кратко, то идея состоит в том, что мы умножаем (полиномиально) байты блока данных на соответствующие им в таблице L-преобразования (каждый на каждый), далее складываем всю пачку и записываем, предварительно выполнив сдвиг влево, в младший байт. Без преувеличения это самая сложная операция всего алгоритма.

Все эти шаги повторяются на протяжении **9-и** раундов раз за разом, кроме **10-ого,** в нём есть только**X-функция**.

## Генерация раундовых ключей

Процесс генерации раундовых ключей основан на **сетях Фейстеля и состоит из 8 раундов.** Первая пара ключей формируется в результате разделения мастер-ключа на две части, каждая последующая пара раундовых ключей получается при помощи 8 итераций сети Фейстеля. На каждой итерации используется итерационная константа, полученная линейным преобразованием номера итерации, Рисунок 2.

* X-функция. Тут так же происходит побитовый XOR. Левая часть ключа складывается с итерационными константами (заранее рассчитанными, приведём их расчёт позже).
* S-функция. Работает так же как и всегда (над левой частью!!!).
* L-функция. Аналогично.
* X-функция с правой половиной + замена местами. Последний этап это побитовый XOR c правой половиной исходного ключа. Далее правая и левая часть меняются местами.

Далее мы повторяем X,S,L и X с правой частью 8 раз для получения новой пары ключей (8 раундов сети Фейстеля на каждую пару ключей).

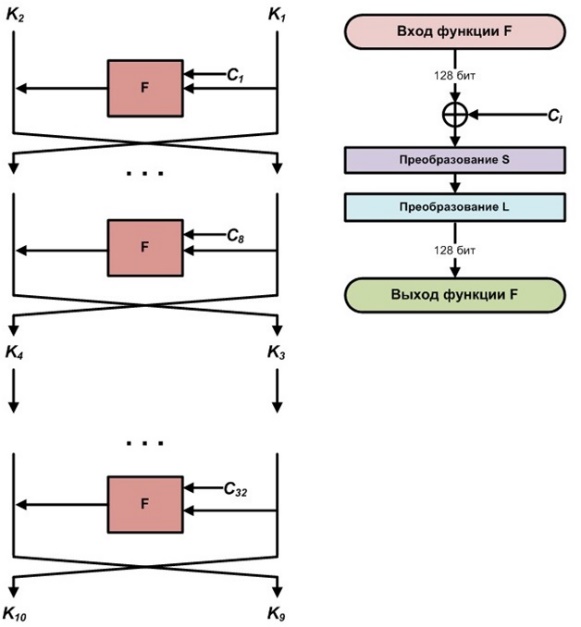


Рис. 2 – Схема создания раундовых ключей

## Реализация X-функции. Побитовый XOR

**Побитовое исключающие или** двух сущностей, раундового ключа и блока данных. Листинг представлен ниже:

procedure xor16bit(signal firBlock, secBlock: in unsigned(0 to 15); signal result: out unsigned(0 to 15)) is

begin

result(0) <= firBlock(0) xor secBlock(0);

result(1) <= firBlock(1) xor secBlock(1);

result(2) <= firBlock(2) xor secBlock(2);

result(3) <= firBlock(3) xor secBlock(3);

result(4) <= firBlock(4) xor secBlock(4);

result(5) <= firBlock(5) xor secBlock(5);

result(6) <= firBlock(6) xor secBlock(6);

result(7) <= firBlock(7) xor secBlock(7);

result(8) <= firBlock(8) xor secBlock(8);

result(9) <= firBlock(9) xor secBlock(9);

result(10) <= firBlock(10) xor secBlock(10);

result(11) <= firBlock(11) xor secBlock(11);

result(12) <= firBlock(12) xor secBlock(12);

result(13) <= firBlock(13) xor secBlock(13);

result(14) <= firBlock(14) xor secBlock(14);

result(15) <= firBlock(15) xor secBlock(15);

end xor16bit;

## Реализация S-функции. Нелинейное преобразование

Так как функция S это по сути замена одних значений на другие в соответствии с таблицей замен. Был написан скрипт на **Си,** где десятичные цифры переводились в бинарный вид, а затем формировались условия для **оператора case**. Проще понять это на примере: первое число у нас **0,** по нулевому индексу в таблице замен находится число **252**. Если перевести его в двоичный вид, получится **11111100**. Из этого следует, что мы ставим в соответствие вектору **00000000** вектор **11111100**. На этом принципе основана работа S-функции в нашем алгоритме.

Часть листинга S-функции представлена ниже:

procedure tableShift(signal inputPart: in unsigned(0 to 7); signal result:out unsigned(0 to 7)) is

begin

C: case inputPart is

when "00000000" =>

result(0) <= '1';

result(1) <= '1';

result(2) <= '1';

result(3) <= '1';

result(4) <= '1';

result(5) <= '1';

result(6) <= '0';

result(7) <= '0';

## Реализация L-функции. Линейное преобразование

Нам нужно переводить числа из двоичной системе счисления в десятичную, далее по таблице степеней выяснить чему соответствует то или иное значение, после чего необходимо его заменить и перевести снова в двоичный вид. После этого можно провести xor все байтов и запись в младший получившейся суммы. Код при этом выглядит так:

numToPower(numba=>outUnsigned(0 to 7), result=>testKey1);

numToPower(numba=>outUnsignedD(0 to 7), result=>testKey2);

numToPower(numba=>testCoef1,result=>testCoef1Tab); numToPower(numba=>testCoef2, result=>testCoef2Tab);

firBlocNoCon(0 to 7) <= to\_unsigned(to\_integer(testKey1) + to\_integer(testCoef1Tab), 8);

SecBlocNoCon(0 to 7) <= to\_unsigned(to\_integer(testKey2) + to\_integer(testCoef2Tab), 8);

powToNumber(powa=>firBlocNoCon, result=>firBlocCon); powToNumber(powa=>SecBlocNoCon,result=>SecBlocCon);

outputBlockPo(0) <= firBlocCon(0) xor SecBlocCon(0); outputBlockPo(1) <= firBlocCon(1) xor SecBlocCon(1); outputBlockPo(2) <= firBlocCon(2) xor SecBlocCon(2); outputBlockPo(3) <= firBlocCon(3) xor SecBlocCon(3); outputBlockPo(4) <= firBlocCon(4) xor SecBlocCon(4); outputBlockPo(5) <= firBlocCon(5) xor SecBlocCon(5); outputBlockPo(6) <= firBlocCon(6) xor SecBlocCon(6); outputBlockPo(7) <= firBlocCon(7) xor SecBlocCon(7);

outputBlockPo(8) <= firBlocCon(0);

outputBlockPo(9) <= firBlocCon(1);

outputBlockPo(10) <= firBlocCon(2);

outputBlockPo(11) <= firBlocCon(3);

outputBlockPo(12) <= firBlocCon(4);

outputBlockPo(13) <= firBlocCon(5);

outputBlockPo(14) <= firBlocCon(6);

outputBlockPo(15) <= firBlocCon(7);

## Генерация раундовых ключей для X-функции.

Как мы выяснили ранее процесс генерации раундовых ключей основан на тех же операциях, что и “основной алгоритм шифрования блока данных”. Единственным отличием будет **X-функция**, в случае с раундовым ключом выполняется **XOR**с итерационной константой, специальной переменной заранее просчитанной для каждого раунда сети Фейстеля.

Вторым важным отличием является то, что после X,S,L преобразования необходимо ещё и сложить(имеется ввиду исключающее или) левую и правую часть ключа, после чего поменять их местами. Листинг функции генерации раундового ключа, c этапами отличными от основного процесса шифрования, представлен ниже:

procedure keyGenRound(signal masterKey2, masterKey1, c1: in unsigned(0 to 15); signal secIterIn:out unsigned(0 to 15);

signal firIterS1, firIterS2, firIterSum1, firIterSum2, firIterPow1, firIterPow2, firIterNum1, firIterNum2: inout unsigned(0 to 7);

signal intCoef1, intCoef2:in integer; signal firIterRes, firIterXor: inout unsigned(0 to 15)) is

begin

xor16bit(firBlock=>masterKey2, secBlock=>c1, result=>firIterXor)

/\* вызовы S и L функций \*/

xor16bit(firBlock=>masterKey1, secBlock=>firIterRes, result=>secIterIn)

end keyGenRound;

# Результат проделанной работы.

В результате выполнения работы были созданы два специализированных ip-ядра, шифрующее и расшифровывающее. Были выполнены симуляции для проверки правильности работы алгоритма в логическом анализаторе среды Vivado, подтвердившие работоспособность описанного на VHDL алгоритма.

Так же был разработан корпус устройства в САПР Autodesk Fusion360, после чего он был распечатан на 3D принтере, Рисунок 3.

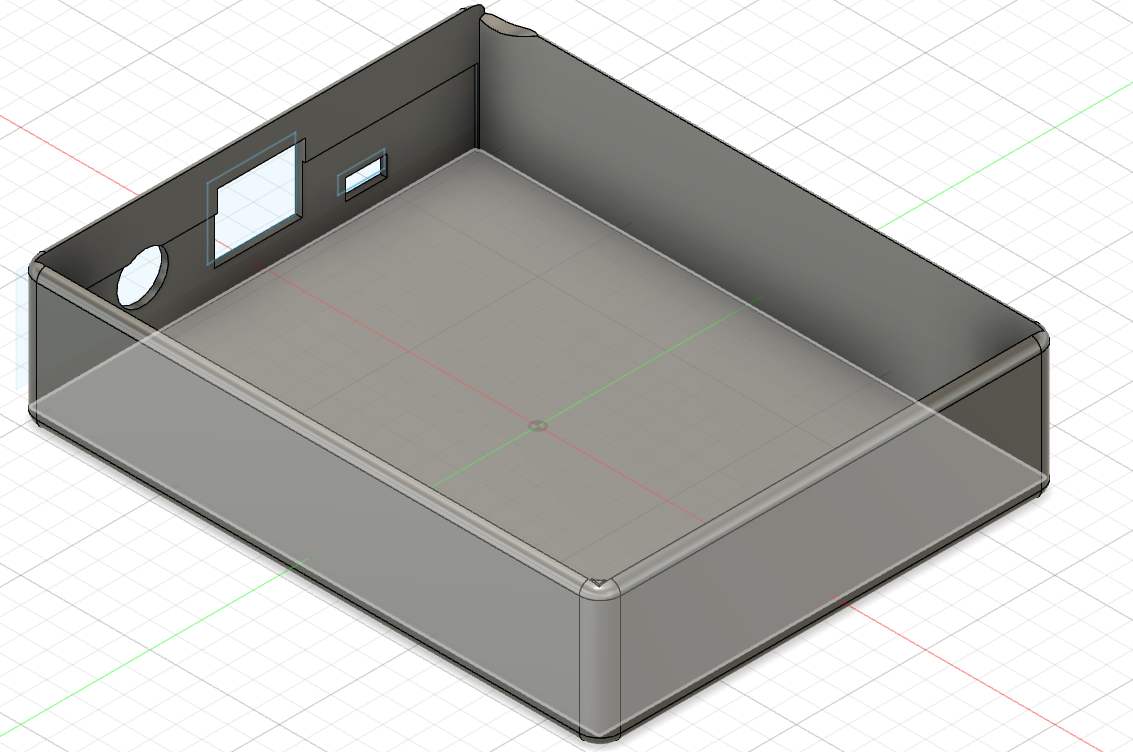


Рис. 3 –Внешний вид платы корпуса устройства

# Список источников

1. Криптографический алгоритм «Кузнечик»: просто о сложном // Habr URL: <https://habr.com/ru/post/459004>
2. Разработка процессорной системы на базе софт-процессора MicroBlaze в среде Xilinx Vivado IDE // fpga-systems URL: <https://fpga-systems.ru/publ/xilinx/microblaze/razrabotka_processornoj_sistemy_na_baze_soft_processora_microblaze_v_srede_xilinx_vivado_ide_hlx_chast_2/10-1-0-7>
3. Creating a custom IP block in Vivado // fpga-developer URL: <http://www.fpgadeveloper.com/2014/08/creating-a-custom-ip-block-in-vivado.html>
4. П. Н. Бибило Основы языка VHDL. Либроком, 2016.
5. И.И. Левин, Б.Е. Механцев Лабораторные работы по дисциплине "ПЛИС-технологии и методы создания эффективных прикладных программ для ПЛИС". Таганрог: Издательство Южного федерального университета, 2017.