Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Магистратура

Кафедра электронных вычислительных машин

Специальность: 1-40 81 02 «Технологии виртуализации и облачных вычислений»

ОТЧЕТ

ПРАКТИЧЕСКОЕ ЗАДАНИЕ

по курсу «Прикладная криптография»

Выполнил: Е.В. Пичкур

магистрант группы 758442

Проверил: А.В. Станкевич

доцент кафедры ЭВС

МИНСК 2018

СОДЕРЖАНИЕ

[**1 ЗАДАНИЕ** 3](#_Toc531325605)

[**2 АЛГОРИТМЫ ЗАШИФРОВАНИЯ И РАСШИФРОВАНИЯ БЛОКА** 4](#_Toc531325606)

[**3 РЕАЛИЗАЦИЯ VHDL** 6](#_Toc531325607)

[**4 РЕЗУЛЬТАТЫ МОДЕЛИРОВАНИЯ** 9](#_Toc531325608)

[**5 ВЫВОД** 10](#_Toc531325609)

# **1 ЗАДАНИЕ**

Практическое задание состояло в аппаратной реализации и моделировании алгоритмов зашифрования и расшифрования блока данных по СТБ 34.101.31-2011.

* 1. Теоретический материал

Стандарт СТБ 34.101.31-2011 определяет семейство криптографических алгоритмов шифрования и контроля целостности, которые используются для защиты информации при ее хранении, передаче и обработке. Криптографические алгоритмы стандарта построены на основе базовых алгоритмов шифрования блока данных, которые и были реализованы и промоделированы в данной работе.

Входными данными для базовых алгоритмов зашифрования и расшифрования являются блок данных *Х ϵ* {0,1}128длиной 128 бит и секретный ключ *θ ϵ* {0,1}256 длиной 256 бит. Выходными данными является блок *Y ϵ* {0,1}128 – результат зашифрования или расшифрования слова *Х* на ключе *θ*.

Входные данные подготавливаются следующим образом:

1 Слово *X* записывается в виде *X = X1 || X2 || X3 || X4*, где *Xi ϵ* {0,1}32.

2 Ключ *θ* записывается в виде *θ = θ1 || θ2 || θ3 || θ4 || θ5 || θ6 || θ7 || θ8*, где *θi ϵ* {0,1}32, и определяются тактовые ключи *K1 = θ1*, *K2 = θ2*, …, *K8 = θ8*, *K9 = θ1*, *K10 = θ2*, …, *K56 = θ8*.

Также определены 2 вспомогательных преобразования:

1 Подстановка *H :* {0,1}8 → {0,1}8. Задается таблицей, приведенной в стандарте, однако можно разработать и использовать собственную таблицу подстановки.

1. Преобразования *Gr (r=5,13,21)*. Преобразование *Gr :* {0,1}32→{0,1}32 ставит в соответствие слову *u = u1 || u2 || u3 || u4, ui ϵ* {0,1}8, слово

*Gr(u) = RotHir(H(u1)||H(u2)||H(u3)||H(u4))*

Также используется 5 вспомогательных переменных *a, b, c, d, e* со значениями из {0,1}32.

Используются операции побитового исключающего «ИЛИ» над 32-битными словами и операции сложения и вычитания по модулю 232 над 32-битными словами.

**2 АЛГОРИТМЫ ЗАШИФРОВАНИЯ И РАСШИФРОВАНИЯ БЛОКА**

* 1. Алгоритм зашифрования

Для зашифрования блока *X* на ключе *θ* выполняются следующие шаги:

1 Установить *a ← X1*, *b ← X2*, *c ← X3*, *d ← X4*.

2 Для *i =* 1,2,…,8 выполнить:

1. *b ← b xor G5(a + K7i-6)*
2. *с ← с xor G21(d + K7i-5)*
3. *a ← a – G13(b + K7i-4)*
4. *e ← G21(b + c + K7i-3) xor <i>32*
5. *b ← b + e*
6. *c ← c - e*
7. *d ← d + G13(c + K7i-2)*
8. *b ← b xor G21(a + K7i-1)*
9. *c ← c xor G5(d + K7i)*
10. *a ↔ b*
11. *c ↔ d*
12. *b ↔ c*

3 Установить *Y ← b || d || a || c*

4 Вернуть *Y*

* 1. Алгоритм расшифрования

Для расшифрования блока *X* на ключе *θ* выполняются следующие шаги:

1 Установить *a ← X1*, *b ← X2*, *c ← X3*, *d ← X4*.

2 Для *i =* 8,7,…,1 выполнить:

1. *b ← b xor G5(a + K7i)*
2. *с ← с xor G21(d + K7i-1)*
3. *a ← a – G13(b + K7i-2)*
4. *e ← G21(b + c + K7i-3) xor <i>32*
5. *b ← b + e*
6. *c ← c - e*
7. *d ← d + G13(c + K7i-4)*
8. *b ← b xor G21(a + K7i-5)*
9. *c ← c xor G5(d + K7i-6)*
10. *a ↔ b*
11. *c ↔ d*
12. *a ↔ d*

3 Установить *Y ← c || a || d || b*

4 Вернуть *Y*

Примечание: операция *<>32,* совершаемая над десятичным числом *i* представляет собой перевод десятичного числа *i* в его двоичное представление с прямым порядком байт и дополнение двоичного представления нулевыми битами слева до длины 32.

1. **РЕАЛИЗАЦИЯ VHDL**

---- модуль шифрования

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

library xil\_defaultlib;

use xil\_defaultlib.belt\_functions.all;

use IEEE.NUMERIC\_STD.ALL;

entity encrypter is

Port ( plaintext : in STD\_LOGIC\_VECTOR (0 to 127);

key : in STD\_LOGIC\_VECTOR (0 to 255);

ciphertext : out STD\_LOGIC\_VECTOR (0 to 127));

end encrypter;

architecture Behavioral of encrypter is

-- определяем тип данных для хранения раундовых ключей

type vector\_array\_56by32 is array (1 to 56) of std\_logic\_vector(0 to 31);

begin

belt: process(plaintext, key)

variable a, b, c, d, e, temp: std\_logic\_vector(0 to 31);

variable round\_keys: vector\_array\_56by32;

begin

-- инициализируем раундовые ключи

init: for i in 0 to 6 loop

round\_keys(8\*i + 1) := swap\_endian(key(0 to 31));

round\_keys(8\*i + 2) := swap\_endian(key(32 to 63));

round\_keys(8\*i + 3) := swap\_endian(key(64 to 95));

round\_keys(8\*i + 4) := swap\_endian(key(96 to 127));

round\_keys(8\*i + 5) := swap\_endian(key(128 to 159));

round\_keys(8\*i + 6) := swap\_endian(key(160 to 191));

round\_keys(8\*i + 7) := swap\_endian(key(192 to 223));

round\_keys(8\*i + 8) := swap\_endian(key(224 to 255));

end loop init;

-- инициализируем вспомогательные переменные

a := swap\_endian(plaintext(0 to 31));

b := swap\_endian(plaintext(32 to 63));

c := swap\_endian(plaintext(64 to 95));

d := swap\_endian(plaintext(96 to 127));

-- главный цикл

main: for i in 1 to 8 loop

b := b xor G\_transformation(modadd(a, round\_keys(7\*i-6)), 5);

c := c xor G\_transformation(modadd(d, round\_keys(7\*i-5)), 21);

a := modsub(a, G\_transformation(modadd(b, round\_keys(7\*i-4)), 13));

e := G\_transformation(modadd(b, c, round\_keys(7\*i-3)), 21) xor std\_logic\_vector(to\_unsigned(i, 32));

b := modadd(b, e);

c := modsub(c, e);

d := modadd(d, G\_transformation(modadd(c, round\_keys(7\*i-2)), 13));

b := b xor G\_transformation(modadd(a, round\_keys(7\*i-1)), 21);

c := c xor G\_transformation(modadd(d, round\_keys(7\*i)), 5);

-- swap a, b

temp := a;

a := b;

b := temp;

-- swap c, d

temp := c;

c := d;

d := temp;

-- swap b, c

temp := b;

b := c;

c := temp;

end loop main;

-- выдаем результат на выходной порт

ciphertext(0 to 31) <= swap\_endian(b);

ciphertext(32 to 63) <= swap\_endian(d);

ciphertext(64 to 95) <= swap\_endian(a);

ciphertext(96 to 127) <= swap\_endian(c);

end process;

end Behavioral;

--- модуль расшифрования

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

library xil\_defaultlib;

use xil\_defaultlib.belt\_functions.all;

use IEEE.NUMERIC\_STD.ALL;

entity decrypter is

Port ( ciphertext : in STD\_LOGIC\_VECTOR (0 to 127);

key : in STD\_LOGIC\_VECTOR (0 to 255);

plaintext : out STD\_LOGIC\_VECTOR (0 to 127));

end decrypter;

architecture Behavioral of decrypter is

-- определяем тип данных для хранения раундовых ключей

type vector\_array\_56by32 is array (1 to 56) of std\_logic\_vector(0 to 31);

begin

belt: process(ciphertext, key)

variable a, b, c, d, e, temp: std\_logic\_vector(0 to 31);

variable round\_keys: vector\_array\_56by32;

begin

-- инициализируем раундовые ключи

init: for i in 0 to 6 loop

round\_keys(8\*i + 1) := swap\_endian(key(0 to 31));

round\_keys(8\*i + 2) := swap\_endian(key(32 to 63));

round\_keys(8\*i + 3) := swap\_endian(key(64 to 95));

round\_keys(8\*i + 4) := swap\_endian(key(96 to 127));

round\_keys(8\*i + 5) := swap\_endian(key(128 to 159));

round\_keys(8\*i + 6) := swap\_endian(key(160 to 191));

round\_keys(8\*i + 7) := swap\_endian(key(192 to 223));

round\_keys(8\*i + 8) := swap\_endian(key(224 to 255));

end loop init;

-- инициализируем вспомогательные переменные

a := swap\_endian(ciphertext(0 to 31));

b := swap\_endian(ciphertext(32 to 63));

c := swap\_endian(ciphertext(64 to 95));

d := swap\_endian(ciphertext(96 to 127));

-- главный цикл

main: for i in 8 downto 1 loop

b := b xor G\_transformation(modadd(a, round\_keys(7\*i)), 5);

c := c xor G\_transformation(modadd(d, round\_keys(7\*i-1)), 21);

a := modsub(a, G\_transformation(modadd(b, round\_keys(7\*i-2)), 13));

e := G\_transformation(modadd(b, c, round\_keys(7\*i-3)), 21) xor std\_logic\_vector(to\_unsigned(i, 32));

b := modadd(b, e);

c := modsub(c, e);

d := modadd(d, G\_transformation(modadd(c, round\_keys(7\*i-4)), 13));

b := b xor G\_transformation(modadd(a, round\_keys(7\*i-5)), 21);

c := c xor G\_transformation(modadd(d, round\_keys(7\*i-6)), 5);

-- swap a, b

temp := a;

a := b;

b := temp;

-- swap c, d

temp := c;

c := d;

d := temp;

-- swap a, d

temp := a;

a := d;

d := temp;

end loop main;

-- выдаем результат на выходной порт

plaintext(0 to 31) <= swap\_endian(c);

plaintext(32 to 63) <= swap\_endian(a);

plaintext(64 to 95) <= swap\_endian(d);

plaintext(96 to 127) <= swap\_endian(b);

end process;

end Behavioral;

Для выполнения описанных в пункте 1.1 операций был создан пакет belt\_functions, в котором были определены следующие функции:

* function byte\_change(input : in std\_logic\_vector(0 to 7)) return std\_logic\_vector; – функция подстановки *H*
* function G\_transformation(input : in std\_logic\_vector(0 to 31); shift\_size : in integer) return std\_logic\_vector; – преобразования *Gr*
* function modadd(op1 : in std\_logic\_vector(0 to 31); op2 : in std\_logic\_vector(0 to 31)) return std\_logic\_vector; – функция сложения двух аргументов по модулю 232. Также была разработана функция для сложения трех аргументов
* function modsub(op1 : in std\_logic\_vector(0 to 31); op2 : in std\_logic\_vector(0 to 31)) return std\_logic\_vector; – функция вычитания двух аргументов по модулю 232.
* function swap\_endian(vec : in std\_logic\_vector(0 to 31)) return std\_logic\_vector; – функция выполняет смену порядка байт в 32-битном слове. Была реализована, поскольку тестовые примеры стандарта используют запись в обратном порядке байт, а все расчеты должны проводиться в прямом порядке байт.

1. **РЕЗУЛЬТАТЫ МОДЕЛИРОВАНИЯ**

В приложении А стандарта приведены тестовые примеры. При зашифровании блока *X = B194BAC8 0A08F53B 366D008E 584A5DE416* на ключе *θ = E9DEE72C* *8F0C0FA6 2DDB49F4 6F739647 06075316 ED247A37 39CBA383 03A98BF6*16 должен получиться результат *Y = 69CCA1C9 3557C9E3 D66BC3E0 FA88FA6E*16.

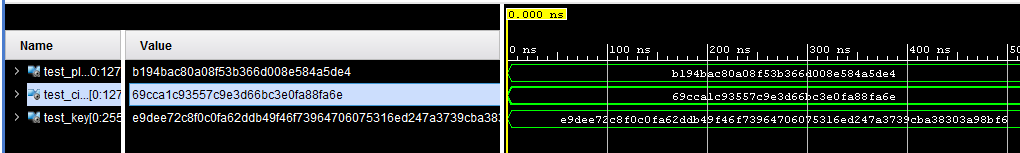


Рис 4.1 – Результат симуляции алгоритма шифрования в Xilinx Vivado

При расшифровании блока *X = E12BDC1A E28257EC 703FCCF0 95EE8DF116* на ключе *θ = 92BD9B1C E5D14101 5445FBC9 5E4D0EF2 682080AA 227D642F 2687F934 9040551116*  должен получиться результат *Y = 0DC53006 00CAB840 B38448E5 E993F42116.*

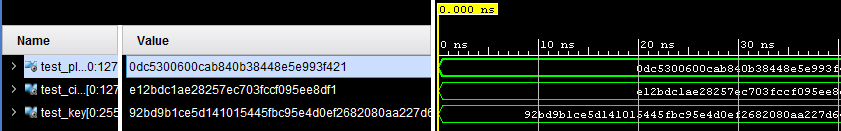
**

Рис 4.2 – Результат симуляции алгоритма расширования в Xilinx Vivado

1. **ВЫВОД**

В результате работы были изучены алгоритмы зашифрования и расшифрования блока данных из СТБ 34.101.31-2011. Данный шифр можно отнести к блочным симметричным шифрам. Также выполнена реализация алгоритмов на языке описания аппаратуры VHDL. Результаты моделирования показывают, что алгоритмы реализованы верно. Также был выполнен автоматический синтез схем, завершившийся успешно, что показывает возможность реализовать данный алгоритм аппаратно.