УДК 004.021

Аппаратная реализация кодека ранговых кодов

И.Ю. Сысоев[1,2], Э.М. Габидулин[1]

[1]Московский физико-технический институт (государственный университет)

[2]ООО «НПП «Цифровые решения», igor@dsol.ru

***Аннотация* — В работе представлен аппаратный сложнофункциональный блок кодирования и декодирования ранговых кодов. Проведена оценка ресурсоёмкости и скорости разработанного аппаратного блока для платформ на базе ПЛИС Spartan-3AN и Spartan-6. Кодек осуществляет операции в конечном поле GF(2^8) и обеспечивает скорость обработки информации до 77 мегабайт в секунду.**

***Ключевые слова* — кодирование, декодирование, быстрые вычисления, СФ-блок, ранговые коды, алгоритм Евклида.**

I. Введение

Ранговые коды были разработаны в 1985-м году [1]. Они являются аналогом кодов Рида-Соломона. Основное отличие между этими кодами заключается в используемой метрике. В отличие от кодов Рида-Соломона, базирующихся на метрике Хэмминга, ранговые коды используют ранговую метрику.

Новый способ передачи данных под названием «сетевое кодирование» был предложен в 2000-м году [2]. Согласно этой технике, несколько пакетов могут быть комбинированы вместе на узле для дальнейшей передачи. Ошибки в этом случае будут суммироваться по тем же самым правилам. Ранговые коды наилучшим образом подходят для коррекции ошибок в таких системах.

Существует большое число теоретических работ, посвящённых ранговым кодам. Несмотря на это, до сих пор не получены результаты практической реализации этих кодов.

Нами была разработана аппаратная реализация рангового кодека. Параметры кода – (8,4,5). Устройство оперирует с элементами конечного поля GF(2^8). Длина кода равняется 8. Кодовое расстояние равно 5. Декодер может исправлять до 2-х ранговых ошибок. Скорость кода равняется ½. Кодек реализован на коммерческих ПЛИС Spartan-3AN и Spartan-6. Устройство использует приблизительно половину ресурсов программируемой логической схемы. Было выяснено, что этап вычисления ключевого уравнения является наиболее ресурсоёмким. Максимальная рабочая частота на данный момент равняется 77 МГц. Таким образом, устройство может обрабатывать поток данных на скорости 77 Мбайт/сек.

Дальнейшее изложение будет организовано следующим образом. В разделе «Схема кодирования» будет рассказано о параметрах кода, операции кодирования и её сложности. В разделе «Схема декодирования» будут описаны основные этапы декодирования, предложены подробные методы получения необходимых результатов и указана их сложность. В разделе «Ресурсоёмкость» будут изложены результаты реализации на ПЛИС кодека рангового кода с параметрами (8,4,5). В заключении будет сделан вывод относительно результатов и указаны дальнейшие направления исследований.

II. Схема кодирования

Схема кодирования рангового кода реализуется в общем случае через процедуру умножения информационного вектора на порождающую матрицу. Стоит заметить, что схема кодирования совпадает с процедурой вычисления синдрома рангового кода. Элементы матрицы состоят из элементов расширенного поля $GF(q^N)$.

Рассмотрим код с параметрами (n,k,d). Можем записать информационный вектор

$$u=(u_1,u_2,...,u_k),u_i\in GF(q^N) \qquad (1)$$

В дальнейшем, если не указано отдельно, то считаем, что $n=N$ и $n=2k$. Последнее условие задаёт скорость кода. Скорость кода в этом случае равняется 0.5. Код можно задать либо порождающей, либо проверочной матрицей. Порождающая матрица G в случае ранговых кодов имеет вид

$$G_{(k\times n)}=\begin{Vmatrix} g_1 & g_2 & \cdots & g_n \\ g_1^{[1]} & g_2^{[1]} & \cdots & g_n^{[1]} \\ \vdots & \vdots & \ddots & \vdots \\ g_1^{[k-1]} & g_2^{[k-1]} & \cdots & g_n^{[k-1]} \end{Vmatrix}, \qquad (2)$$

где $[i]$ обозначает степень Фробениуса, что эквивалентно q^i.

Схема кодирования выполняется с помощью умножения строки информационного вектора u на проверочную матрицу

$$g = u \cdot G = (g_1, g_2, ..., g_n), g_i \in GF(q^N). \quad (3)$$

В работах [3] и [4] описано, как выполнять подобную операцию с использованием слабого самоортогонального базиса [5]. В этом случае сложность кодирования оказывается асимптотически-максимальной и оценивается как

$$\Xi_{code} = \mathrm{O}\left((\log n)^2 n\right) + \mathrm{O}\left(n^{\log 3}\right). \quad (4)$$

III. Схема декодирования

A. Канал с ошибкой

В процессе передачи данных по каналу отправленный кодовый вектор g искажается, что можно описать как добавление к этому вектору ошибки e

$$y = g + e, \quad (5)$$

где

$$e = (e_1, e_2, ..., e_n), e_i \in GF(q^N). \quad (6)$$

Если декодер не обладает никакой дополнительной информацией, кроме полученного вектора y, то его корректирующая способность будет определяться соотношением

$$rank(e) \le \frac{d-1}{2}. \quad (7)$$

Существует несколько способов декодирования ранговых кодов. В основном рассматривается и лучше изучено синдромное декодирование. Поэтому будем реализовывать декодирование на базе синдрома.

B. Вычисление синдрома

Первым этапом синдромного декодирования является операция вычисления синдрома. Синдром вычисляется как произведение полученного вектора на проверочную матрицу

$$s = y \cdot H. \quad (8)$$

Если полученный синдром (8) равняется нулю, то декодер принимает решение о том, что ошибок в полученном кодовом слове нет и по заданному правилу ставит в соответствие этому значению информационный вектор. Проверочная матрица рангового кода

$$H_{(d-1\times n)} = \begin{Vmatrix} h_1 & h_2 & \cdots & h_n \\ h_1^{[1]} & h_2^{[1]} & \cdots & h_n^{[1]} \\ \vdots & \vdots & \ddots & \vdots \\ h_1^{[d-1]} & h_2^{[d-1]} & \cdots & h_n^{[d-1]} \end{Vmatrix} \quad (9)$$

связана с порождающей матрицей (2) и должна удовлетворять условию

$$GH^T = 0. \quad (10)$$

Сложность операции (8) вычисления синдрома сравнима со сложностью операции кодирования. Поэтому можно воспользоваться оценкой (4).

C. Поиск ключевого многочлена

Важным этапом операции декодирования является поиск ключевого многочлена $\Delta(z)$. Обозначим базис пространства ошибок как b_E, а именно,

$$b_E = (E_1, E_2, ..., E_m), \quad (11)$$

где m - ранг возникших ошибок и $E_i \in GF(q^N)$.

Определение 1: Полином $L(z)$ над $GF(q^N)$ называется линеаризованным, если $L(z) = \sum_i L_i z^{p^i}$.

Пусть $\Delta(z)$ обозначает линеаризованный полином, корни которого являются суперпозицией векторов (11), называется *ключевым многочленом* и имеет вид

$$\Delta(z) = \sum_{p=0}^{m} \Delta_p z^{[p]}. \quad (12)$$

Тогда

$$\Delta(E) = 0, \quad (13)$$

причём E - любая суперпозиция векторов из (11)

$$E = \sum_{i=1}^{m} \beta_i E_i, E_i \in b_E, \forall \beta_i \in GF(q). \quad (14)$$

Можно записать ключевое уравнение для ранговых кодов

$$F(z) = \Delta(z) \cdot S(z) \bmod z^{[d-1]}. \quad (15)$$

В (15) $F(z)$ определяется как

$$F(z) = \sum_{i=0}^{m-1} F_i z^{[i]}, \quad (16)$$

где

$$F_i = \sum_{p=0}^{i} \Delta_p s_{i-p}^{[p]}, i = 0,1,..., m-1 \qquad (17)$$

и $S(z)$ обозначает линеаризованный полином синдрома

$$S(z) = \sum_{j=0}^{d-2} s_j z^{[j]}. \qquad (18)$$

В статье [1] показано, что для того, чтобы вычислить ключевое уравнение, необходимо выполнить деление $z^{[d-1]}$ на многочлен синдрома (18). Одним из способов является метод последовательного деления (алгоритм Евклида). В работах [6] и [7] предложен оптимизированный алгоритм Евклида для линеаризованных многочленов. Он позволяет использовать рекурсивный метод вычисления результата. Рекурсивный метод хорошо подходит для аппаратной реализации. Его асимптотическая сложность

$$\Xi_{Euclid} = \mathrm{O}(N^{3.585}). \qquad (19)$$

D. Поиск базиса ошибок

Пусть мы получили коэффициенты ключевого уравнения $\Delta(z) = 0,$

$$\Delta(z) = \sum_{p=0}^{m} \Delta_p z^{[p]}. \qquad (20)$$

Стоит заметить, что это уравнение должно удовлетворять (14). Для решения можно воспользоваться методом, описанным в [7]. Для этого потребуется следующая теорема:

Теорема 1. Пусть $L(z)$ – линеаризованный многочлен и $z = \sum_k Z_k \alpha^k$, где $Z_k \in GF(q)$, тогда $L(z) = \sum_k Z_k L(\alpha^k)$.

Доказательство см. в [7].

Из теоремы 1 следует, что для нахождения корней (12) необходимо вначале вычислить значения многочлена в точках и выразить их в стандартном базисе

$$b_{std} = (\alpha^0, \alpha^1, ..., \alpha^N), \alpha \in GF(q^N). \qquad (21)$$

То есть

$$\Delta(\alpha^k) = \sum_{i=0}^{m} \lambda_{ki} \cdot \alpha^i, k = 0,1,..., m, \qquad (22)$$

$$\lambda_{ki} \in GF(2^N).$$

Получившимся значениям мы можем сопоставить матрицу $(m+1)\times(m+1)$ с элементами из $GF(2)$:

$$\Omega = \begin{Vmatrix} \lambda_{00} & \lambda_{01} & \cdots & \lambda_{0m} \\ \lambda_{01} & \lambda_{11} & \cdots & \lambda_{1m} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_{m0} & \lambda_{m1} & \cdots & \lambda_{mm} \end{Vmatrix} \qquad (23)$$

Тогда задача поиска корней ключевого уравнения будет эквивалентна задаче поиска решения матричного уравнения:

$$Z \cdot \Omega = \begin{vmatrix} Z_0 \\ Z_1 \\ \vdots \\ Z_m \end{vmatrix}^T \cdot \begin{Vmatrix} \lambda_{00} & \lambda_{01} & \cdots & \lambda_{0m} \\ \lambda_{01} & \lambda_{11} & \cdots & \lambda_{1m} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_{m0} & \lambda_{m1} & \cdots & \lambda_{mm} \end{Vmatrix} = \begin{vmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{vmatrix}^T. \qquad (24)$$

В первой части уравнения (24) стоит нулевой вектор. Это уравнение решается путём приведения начальной матрицы Ω к идемпотентной форме (см. пример 2.57 в [8]). Тогда корни ключевого уравнения расположены в строках преобразованной матрицы. Таким образом, мы получим базис векторов ошибок (11).

E. Поиск решения системы уравнения

После определения базиса ошибок (11) необходимо найти местоположения ошибок. На первом этапе производится поиск решения системы [1]:

$$\sum_{j=1}^{m} E_j x_j^{[p]} = s_p, \; p = 0,1,..., m-1. \qquad (25)$$

На первом шаге алгоритма все уравнения системы преобразуются таким образом, чтобы в них присутствовали неизвестные одинаковой степени, а именно:

$$\sum_{j=1}^{m} E_j^{[m-p]} x_j^{[m-1]} = s_j^{[m-1]} = s_p^{[m-p]}. \qquad (26)$$

Сложность преобразования (25) в (26) оценивается как

$$\Xi_{conv} = \Xi_{mult}\left((m-1)^2 + \frac{(m-1)m}{2}\right), \qquad (27)$$

где Ξ_{mult} – сложность умножения двух элементов расширенного поля $GF(q^N)$ в выбранном базисе. В работах [3] и [4] показана возможность оптимизации набора базовых операций (умножения, возведения в степень, поиска обратного элемента). Система (26) является системой линейных алгебраических уравнений. Её можно записать в виде произведения матрицы на вектор:

$$\begin{Vmatrix} E_1^{[m-1]} & E_2^{[m-1]} & \cdots & E_m^{[m-1]} \\ E_1^{[m-2]} & E_2^{[m-2]} & \cdots & E_2^{[m-2]} \\ \vdots & \vdots & \ddots & \vdots \\ E_1^{[0]} & E_2^{[0]} & \cdots & E_m^{[0]} \end{Vmatrix} \begin{Vmatrix} x_1^{[m-1]} \\ x_2^{[m-1]} \\ \vdots \\ x_{m-1}^{[m-1]} \end{Vmatrix} = \begin{vmatrix} s_0^{[m-1]} \\ s_1^{[m-2]} \\ \vdots \\ s_{m-1}^{[0]} \end{vmatrix}. \quad (28)$$

Для решения системы уравнений (26), представленной в виде матрицы (28), необходимо воспользоваться методом исключения Гаусса. В этом случае получим

$$\begin{Vmatrix} 1 & \tilde{E}_{12} & \cdots & \tilde{E}_{1m} \\ 0 & 1 & \cdots & \tilde{E}_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{Vmatrix} \begin{Vmatrix} x_1^{[m-1]} \\ x_2^{[m-1]} \\ \vdots \\ x_{m-1}^{[m-1]} \end{Vmatrix} = \begin{vmatrix} \tilde{s}_0 \\ \tilde{s}_1 \\ \vdots \\ \tilde{s}_{m-1} \end{vmatrix}. \quad (29)$$

Сложность получения системы (23) с помощью метода Гаусса оценивается как

$$\Xi_G = \Xi_{inv} \cdot m + \Xi_{mult}\left(\sum_{k=1}^{m} k^2\right) + \left(\sum_{l=1}^{m} l(l-1)\right), \quad (30)$$

где Ξ_{Add} и Ξ_{inv} - сложности, соответственно, сложения двух элементов и поиска обратного элемента расширенного поля $GF(q^N)$ в выбранном базисе. Решение системы (преобразование матрицы к единичному виду) определяется сложностью

$$\Xi_{SolveX} = \left(\Xi_{mult} + \Xi_{add}\right)\frac{m(m-1)}{2}. \quad (31)$$

После всех преобразований получим группу независимых уравнений

$$\sum_{i=1}^{m} x_p^{[m-1]} = \hat{s}_p, \; p = 0,1,...,m-1, \quad (32)$$

которые решаются путём возведения в степень $\left[n-(m-1)\right]$ со сложностью

$$\Xi_{xPow} = \Xi_{mult} \cdot m(n-(m-1)). \quad (33)$$

F. Поиск локаторов ошибок

Определив из (32) x_i, составим систему уравнений, решением которой являются локаторы ошибок [1]

$$x_p = \sum_{j=1}^{n} Y_{pj} h_j, \; p = 1,2,...,m. \quad (34)$$

В уравнении (34) h_j – элементы первой строки проверочной матрицы кода H (9), причём $h_j \in GF(q^n)$. Y_{pj} являются элементами матрицы размером $(m \times n)$ локаторов ошибок, $Y_{pj} \in GF(q)$.

Стоит заметить, что все уравнения (34) независимы и имеют единственное решение. Элемент h_j можно представить в виде вектора, элементами которого являются элементы $h_{ji} \in GF(q^N)$. Тогда каждое уравнение (34) можно представить в виде

$$X_p = \hat{H} \cdot Y_P = \begin{vmatrix} x_{p1} \\ x_{p2} \\ \vdots \\ x_{pn} \end{vmatrix}, \quad (35)$$

$$X_p = \begin{Vmatrix} h_{11} & h_{21} & \cdots & h_{n1} \\ h_{12} & h_{22} & \cdots & h_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ h_{1n} & h_{2n} & \cdots & h_{nn} \end{Vmatrix}.$$

Для нахождения Y_p необходимо умножить X_p на матрицу $\hat{H}^{-1}$, которая не зависит от кодовых векторов и может быть вычислена заранее, на этапе разработки устройства. Сложность операции сверху можно оценить как

$$\Xi_Y = \Xi_{add} \cdot (n-1)m. \quad (36)$$

G. Вычисление ошибки и исправление кодового слова

По матрице локаторов ошибок Y_{pj}, известной из (34), и вычисленной матрице базисных векторов пространства ошибок E_i (11) просто получить значения ошибок e_k для каждого кодового вектора

$$e = EY = \begin{vmatrix} E_1 \\ E_2 \\ \vdots \\ E_m \end{vmatrix}^T \begin{Vmatrix} Y_{11} & Y_{21} & \cdots & Y_{n1} \\ Y_{12} & Y_{22} & \cdots & Y_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ Y_{1n} & Y_{2n} & \cdots & Y_{nn} \end{Vmatrix}. \quad (37)$$

Исправление кодового слова выполняется обычным вычитанием

$$g = y - e. \quad (38)$$

Сложность суммарной операции вычисления ошибки и вычисления кодового вектора оценим величиной

$$\Xi_e = \Xi_{add} \cdot (nm). \quad (39)$$

Таблица 1

Результаты моделирования и синтеза блока кодека ранговых кодов

Название этапа	**Задержка, такты**	**Элементарные блоки (slice), шт. (%)**		**Триггеры, шт. (%)**		**4-х портовая память (LUT), шт. (%)**		**Макс. рабочая частота, МГц**	
		S3	S6	S3	S6	S3	S6	S3	S6
Полный кодер	14	850 (14)	209 (9)	921 (8)	630 (3)	921 (8)	630 (3)	163	282
Вычисление синдрома	25	728 (12)	196 (9)	897 (8)	620 (3)	933 (8)	614 (7)	176	313
Алгоритм Евклида	2040	1837 (31)	826 (36)	1980 (17)	1956 (11)	2945 (25)	2045 (22)	87	156
Вычисление базиса ошибок	977	489 (8)	156 (7)	321 (3)	302 (2)	859 (7)	422 (5)	130	174
Вычисление локатора ошибок	401	877 (15)	262 (12)	927 (8)	784 (4)	1214 (10)	545 (6)	143	202
Коррекция ошибки	15	96 (2)	32 (1)	92 (1)	82 (1)	164 (1)	72 (1)	212	344
Полный декодер	3458	3979 (68)	1380 (61)	4420 (38)	4027 (22)	6106 (54)	3564 (40)	86	155
Полный кодек	3472	4829 (82)	1583 (70)	5341 (45)	4657 (26)	7353 (62)	4423 (49)	86	155

H. Вычисление информационного вектора

После исправления кодового вектора и получения информационного вектора необходимо выполнить операцию, обратную декодированию

$$\tilde{u} = g \cdot D = u \cdot G^{-1} \cdot D. \qquad (40)$$

Для того, чтобы $\tilde{u}$ равнялось u , необходимо выполнения условия

$$G_{(k\times n)} \cdot D_{(n\times k)} = I_{(k\times k)}, \qquad (41)$$

где I – единичная матрица, на диагонали которой стоят единичные элементы из $GF(q^N)$. Матрица D определяется однозначно и вычисляется заранее. Сложность умножения кодового вектора g на матрицу D можно оценить как

$$\Xi_u = \Xi_{mult} \cdot mn + \Xi_{add} \cdot (n-1)m. \qquad (42)$$

IV. Ресурсоёмкость

Авторами был реализован описанный алгоритм на языке описания аппаратуры VHDL. Реализован кодек рангового кода с параметрами $(8,4,5)$. В качестве базового поля было взято $GF(2)$, в качестве расширенного – $GF(2^8)$ $(q=2, N=8)$. Перед декодированием устройство не имеет информации о стираниях строк и стираниях столбцов.

Кодек имеет простые интерфейсы для ввода и выводов данных. Пользователь может использовать сигнал «LOAD» и «Y[7:0]» для ввода данных. Для получения данных необходимо использовать исправленный вектор «I[7:0]» с флагами «VALID» и «FAIL». Такой интерфейс не накладывает ограничения на данные в канале. Поэтому кодек может быть использован любым передатчиком, таким как проводной соединитель (RS-485/Ethernet), память (т.е. NAND Flash) или радиоканал (т.е. ZigBee/WiFi).

Для более точных результатов сравнения эта реализация рангового кодека не зависит от специализированных блоков ПЛИС (таких как умножители, блочная память, блоки ввода-вывода). Поэтому СФ-блок может быть использован не только для реализации на ПЛИС, но и для реализации в заказных СБИС.

Для моделирования был использован САПР Active-HDL 7.2 Student Edition от компании Aldec [9]. В качестве языка описания аппаратной части использовался VHDL. Проект создавался для ПЛИС XC3S700AN-4FGG484C и XC6SLX16-3CSG225 [9]. Для получения конфигурационного файла ПЛИС использовался пакет ISE WebPack версии 13.1 фирмы Xilinx. Синтез (synthesize) и алгоритм создания описания для упаковки в память ПЛИС (translation, mapping, placement и routing) выполнялись с настройками по умолчанию. Результаты приведены в таблице 1. Кодек может работать на частоте 155 МГц (в случае ПЛИС Spartan-6) и 86 МГц (в случае ПЛИС Spartan-3). Поэтому, если скорость кода ½, то кодек может обрабатывать данные на скорости 77 Мбайт/сек и 43 Мбайт/сек, соответственно. Ключевое различие между двумя типами ПЛИС заключается в используемой технологии и архитектуры.

Технология, на которой изготовлена ПЛИС Spartan-3 (обозначена как «S3»), – 90 нм. Технология, на которой изготовлена ПЛИС Spartan-6 (обозначена

как «S6») – 45 нм. Подобное различие технологий должно было привести к разнице в достижимой скорости. Действительно, таблица 1 показывает увеличение скорости.

Spartan-3 содержит блоки 4-х входовой таблицы истинности. Каждая такая таблица может реализовывать в себе либо логический элемент, либо ячейку памяти. Каждый конфигурированный блок в Spartan-6 состоит из двух элементарных ячеек. Каждая ячейка содержит 4 блока 6-ти входовой таблицы истинности, восемь триггеров и дополнительные блоки логики. Как видно из таблицы 1, в новой архитектуре количество используемых элементарных ячеек намного меньше, чем в Spartan-6. Полученные результаты необходимо анализировать с учётом особенностей архитектур.

Стоимость ПЛИС данной серии (Spartan) невысока, что делает приемлемым использование рангового кодирования в экспериментальных образцах и мелкосерийных промышленных устройствах.

V. Заключение

Результаты реализации важны для дальнейших научных исследований ранговых кодов. Они позволят более точно оценить сложность систем на их основе. Понимание внутренней структуры позволит сконцентрировать усилия на наиболее важных проблемах. Более того, полученные результаты можно сравнить с точки зрения практического использования с кодеками на базе других кодов (например, кодов Рида-Соломона). Информация об аппаратной реализации полезна не только для теоретических работ, но и при решении прикладных задач.

Дальнейшие исследования необходимо направить на изучение более сложных кодеков (с большими параметрами n и k) и определить зависимость сложности аппаратной реализации от изменения отдельных параметров.

Литература

[1] Габидулин Э.М. Теория кодов с максимальным ранговым расстоянием // Проблемы передачи информации. 1985. Т. 21. Вып. 1. С. 3–14.

[2] R. Ahlswede E., N. Cai S.-Y. R. Li. and R.W. Yeung Network information flow // IEEE Trans. Inform. Theory. 2000. V. 46. P. 1204–1216.

[3] E.M. Gabidulin and I.Y. Sysoev Rank codes using weak self-orthogonal bases // Computational Technologies in Electrical and Electronics Engineering (SIBIRCON). 2010. P. 70–71.

[4] Сысоев И.Ю. Быстрый алгоритм вычисления синдрома рангового кода с использованием слабо самоортогонального базиса в поле Галуа $GF(2^{2^N})$ // Труды 13-й международной конференции «Цифровая обработка сигналов и её применение – DSPA-2011». 2011. Т. 1. С. 78–81.

[5] E.M. Gabidulin and N.I. Pilipchuk Symmetric matrices and codes correcting rank errors beyond the $\lfloor (d-1)/2 \rfloor$ bound // Discrete Appl. Math. 2006. V. 154. №2.

[6] Габидулин Э.М. и Сысоев И.Ю. Модифицированный алгоритм декодирования ранговых кодов // Труды 54-й научной конференции МФТИ. 2011. Т. 2. Ч. 2. С. 55-57.

[7] Сысоев И.Ю. Эффективный алгоритм Евклида для линеаризованных многочленов // Труды 9-й международной конференции «Перспективные технологии в средствах передачи информации – ПТСПИ-2011». 2011. Т. 3. С. 40–45.

[8] Берлекэмп Э., Алгебраическая теория кодирования. М.: Мир, 1971.

[9] URL: http://www.xilinx.com (дата обращения: 03.03.2014).