Наука и Образование
МГТУ им. Н.Э. Баумана
Сетевое научное издание
ISSN 1994-0408

Наука и Образование. МГТУ им. Н.Э. Баумана. Электрон. журн. 2016. № 06. С. 184–199.

DOI: **10.7463/0616.0842221**

Представлена в редакцию: 13.05.2016
Исправлена: 27.05.2016

УДК 519.725.2; 004.312.4

Реализация потокового декодера укороченных кодов Рида-Соломона на ПЛИС

Федоров С. В.[1,*], **Ромашкин В. И.**[2], **Вялых К. М.**[1]

*svf@bmstu.ru

[1]МГТУ им. Н.Э. Баумана, Москва, Россия
[2]НИЦ «Курчатовский институт» , Москва, Россия

Предложена новая методика декодирования укороченных кодов Рида-Соломона. Аппаратная реализация по данной методике позволяет осуществлять потоковую обработку, сократить количество тактов задержки и уменьшить количество используемых логических элементов относительно реализации декодера компанией-производителем программируемых логических интегральных схем. Отличительной особенностью методики является то, что декодер обрабатывает поток кодовых слов разной длины без изменения их структуры и введения дополнительных задержек, связанных с расширением кодовых слов нулями. Для этого введено понятие корректоров локаторов ошибок для укороченного кода и предложен метод их расчета в процессе приема слова укороченного кода.

Ключевые слова: коды, исправляющие ошибки, блочные коды, код Рида-Соломона, коды БЧХ, ПЛИС

1. Введение

В ряде систем используются укороченные коды Рида-Соломона. Традиционный метод декодирования таких кодов заключается в дополнении кодового слова нулями до полного кода и последующем декодировании. Это делает невозможной обработку данных в темпе поступления информации, что снижает производительность системы при использовании пакетов, содержащих кодовые слова различной длины. Несмотря на распространение в настоящее время более эффективных турбокодов и кодов с малой плотностью проверок на четность (LDPC кодов), коды Рида-Соломона по-прежнему широко используют в системах, где имеются ограничения на вычислительную сложность или требуется высокая пропускная способность, например, в сотовых сетях пятого поколения. В данной работе рассматривается реализация потокового декодера для сокращенного кода Рида-Соломона в системах цифрового телевидения, соответствующего стандартам ETSI EN 300 421 [1] и ETSI EN 301 790 [2]. В данных стандартах используются укороченные коды Рида-Соломона для пакетов различной длины, производные от стандартного кода (255,239,8).

Рассмотрим алгоритм определения позиций ошибок для полного кода и вычисления их значений. Пусть в приемник поступило сообщение

$$c(x) = f(x) + e(x),$$

где $f(x)$ – переданное сообщение (кодовое слово), в котором в процессе передачи по каналу связи произошло υ ошибок, отображаемых многочленом $e(x)$.

Каждый ненулевой компонент $e(x)$ в процессе декодирования описывается парой значений: Y_i – величина ошибки и X_i – номер позиции ошибки (локатор ошибки). При этом Y_i, X_i являются элементами $GF(n)$, и значение $X_i = \alpha^i$, $\alpha^i \in GF(n)$.

Для описания ошибок используются:

1. Многочлен локаторов ошибок $\Lambda(x)$:

$$\Lambda(x) = \prod_{i=1}^{\upsilon}(1 - xX_i) = \Lambda_\upsilon x^\upsilon + \Lambda_{\upsilon-1}x^{\upsilon-1} + \ldots + \Lambda_1 x^1 + \Lambda_0,$$

имеющий корни $x_i = X_i^{-1}$. Данные корни имеют значения, обратные к локаторам ошибок, т. е.

$$X_i^{-1} \cdot X_i = 1.$$

2. Многочлен значений ошибок $\Omega(x)$:

$$\Omega(x) = S(x)\Lambda(x)\left(\mathrm{mod}\, x^{2t}\right),$$

где

$$S(x) = \sum_{j=1}^{2t} S_j x^j = \sum_{j=1}^{2t}\sum_{i=1}^{\upsilon} Y_i X_i^j x^j$$

синдромный многочлен бесконечной степени, для которого известны только $2t$ коэффициентов для поступившего сообщения.

Здесь

$$S_j = \sum_{i=1}^{\upsilon} Y_i X_i^j = e\left(\alpha^j\right)$$

элемент синдрома, α^j – корень порождающего многочлена кода.

Значение $S_j = e\left(\alpha^j\right)$ вычисляется в модуле вычисления синдромов как:

$$S_j = c\left(\alpha^j\right) = f\left(\alpha^j\right) + e\left(\alpha^j\right) = e\left(\alpha^j\right),$$

где $m_0 \le j \le m_0 + 2t - 1$ при $m_0 = 1$.

Уравнение для многочлена значений ошибок определяет множество из $(2t–\upsilon)$ уравнений и называется ключевым уравнением, так как оно представляется “ключом” решения задачи декодирования. Так как степень $\Omega(x)$ не превышает $\upsilon–1$, то поэтому справедливо:

$$\left[\Lambda(x)S(x)\right]_\upsilon^{2t-1} = 0,$$

где $\left[a(x)\right]_m^n = a_m X^m + a_{m+1}X^{m+1} + \ldots + a_n X^n$.

Из этого уравнения необходимо получить υ уравнений для υ неизвестных коэффициентов Λ_k. Эти уравнения являются линейными. Они могут быть решены обычными ме-

тодами либо с помощью итерационных процедур. После нахождения многочлена $\Lambda(x)$ ключевое уравнение позволяет найти неизвестные компоненты вектора $e(x)$ и по ним выходной вектор декодера $f(x)= c(x) + e(x)$.

Обобщенно аппаратную реализацию декодера Рида-Соломона можно представить в виде следующих этапов:

1. вычисление синдромов;
2. решение ключевого уравнения;
3. поиск позиций ошибок;
4. расчет значений ошибок;
5. коррекция ошибок.

Решение ключевого уравнения является основной задачей декодирования кодов Рида-Соломона, для которой разработан целый ряд алгоритмов, со своими достоинствами и недостатками.

Прямолинейная реализация решения (см. [3] и [4]) влечет за собой решение системы из $2t$ уравнений, где t – корректирующая способность кода. Из-за сложности обращения матриц больших размерностей, применяется он только для малых значений t. Алгоритмы с мягким решением, например алгоритм Судана [5], в аппаратных реализациях используются редко ввиду нерегулярности получаемой архитектуры.

Для сокращения вычислительной сложности наиболее часто применяют алгоритмы, основанные на подходе Берлекэмпа-Месси [6] и Евклида [7].

Алгоритм Евклида обладает высокой регулярностью структуры, что приводит к эффективной аппаратной реализации. Но классический алгоритм Евклида требует много ресурсов, а итеративный обладает большой вычислительной задержкой. В [7] рассмотрена аппаратная реализация решателя на основе алгоритма Евклида на жесткой логике и программируемых логических интегральных схемах (ПЛИС).

Алгоритм Берлекэмпа-Месси обладает высокой эффективностью по числу операций в конечном поле, из-за чего он наиболее часто применяется в программных декодерах и при моделировании. Этот факт также благоприятно сказывается на необходимых для аппаратной реализации ресурсах и времени работы алгоритма. К сожалению, в исходном алгоритме Берлекэмпа-Месси используется операция нахождения обратного, что значительно повышает сложность и снижает быстродействие аппаратуры, реализующей алгоритм. Имеются модификации алгоритма, не использующие делений (iBM) [6]. В работах [6,7] реализуется несколько алгоритмов на основе iBM и проводится их сравнение с алгоритмом Евклида. В работе [8] предложен алгоритм на основе iBM, требующий меньшее количество умножителей.

На рис. 1 показана общая структура декодера полного кода Рида-Соломона, включающего рассмотренные выше этапы, причем для поиска позиций и значений ошибок, а также для их исправления используются процедура Ченя и алгоритм Форни [9].

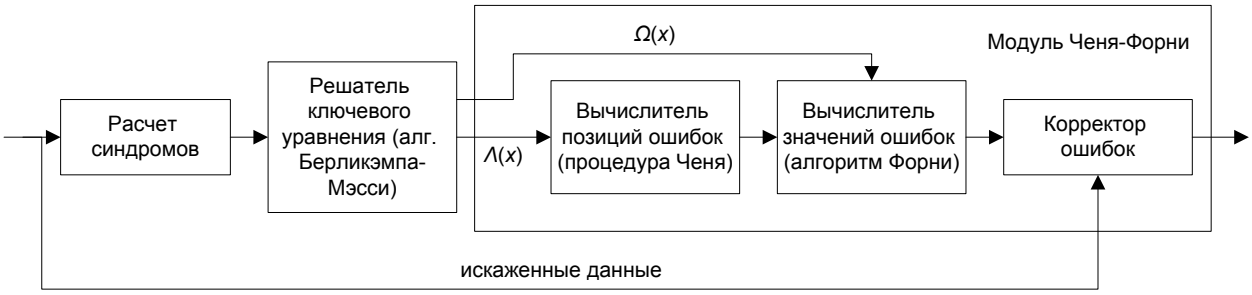

Рис. 1. Структура декодера полного кода.

В рассмотренном выше алгоритме использование укороченного кода не требует изменений в блоке расчета синдромов, но при определении позиций и значений ошибок процедурой Ченя и алгоритмом Форни на дальнейших шагах декодирования необходимо вносить коррекции для декодирования укороченного кода.

Один из возможных подходов декодирования укороченного кода – расширение полученного слова нулями до полного кода и его последующая обработка. Это делает невозможной непрерывную обработку поступающих пакетов данных. Два варианта коррекции без дополнения слова нулями предложены в [9]. Один метод предполагает заполнение сдвигового регистра для расчета синдрома с учетом укорочения кода, что соответствует дополнению слова нулями, однако, требует знания длины кодового слова, а второй метод корректирует позицию найденной ошибки при обработке синдромов. Методы ускоренного декодирования укороченного кода, схожие с подходом, используемым в данной работе, были рассмотрены в [10,11]. Однако, в отличие от этих методов, предлагаемый метод упрощает расчет коррекций и позволяет не модифицировать модули расчета локаторов, значений ошибок и синдромов для полного кода, внося минимальные изменения в модули. Предложенный метод позволяет использовать особенности структуры декодера на основе алгоритма Берлекэмпа-Месси и эффективно вычислить коррекции, требуемые для слова укороченного кода непосредственно в процессе его приема.

2. Декодер укороченного кода

Далее будет рассматриваться декодер со структурой, представленной на рис. 2. Дополнительный модуль расчета корректора локаторов позволяет осуществлять расчет синдромов для укороченного кода в процессе его приема без дополнения его нулями и последующую коррекцию результатов решения ключевого уравнения для корректной работы процедуры Ченя и Алгоритма Форни.

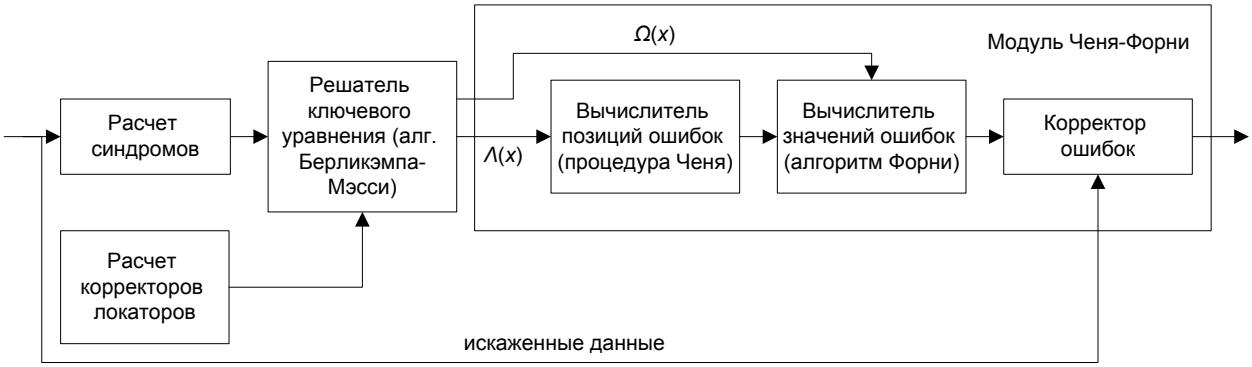

Рис. 2. Структура декодера укороченного кода.

2.1. Расчет синдромов

Модуль расчета синдромов осуществляет первый этап декодирования. В данном модуле расчет всех синдромов ведется параллельно с приемом слова и все *2t* синдромов доступны через такт после приема сообщения.

Входами модуля являются байт принимаемой последовательности *r*, сигнал *sop*, оповещающий о начале слова, входы тактирования, разрешения и сбора. На выходе – 16 значений синдромов, т.к. рассматриваемый код может исправлять до 8 ошибок. Ниже представлена формула расчета синдромов.

$$S_i = \sum_{j=0}^{n-1} c_j \alpha^{j \cdot i},\ i = \overline{0:(2t-1)}.$$

Можно разложить это выражение по схеме Горнера, тогда получим выражение, использованное в нашей аппаратной реализации.

$$S_i = \left(\ldots\left(\left(c_{n-1}\cdot\alpha^i + c_{n-2}\right)\cdot\alpha^i + c_{n-3}\right)\cdot\alpha^i + c_{n-4}\right)\cdot\alpha^i + \ldots + c_0\Big).$$

Данное выражение справедливо и при работе с укороченными на *k* символов кодами. При привидении укороченного кода к полному, старшие степени полиномиального представления каждого слова заполнятся нулями. Этот факт изменит выражение расчета синдрома следующим образом:

$$S_i = \left(\ldots\left(\left(c_{n-k-1}\cdot\alpha^i + c_{n-k-2}\right)\cdot\alpha^i + c_{n-k-3}\right)\cdot\alpha^i + c_{n-k-4}\right)\cdot\alpha^i + \ldots + c_0\Big).$$

Легко заметить, что это изменение никак не меняет алгоритм вычисления.

Данная схема реализована в модуле, схема которого представлена на рис. 3.

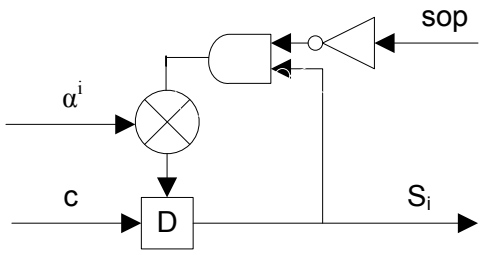

Рис. 3. Схема блока расчета *i*-го синдрома.

Из схемы видно, что при приходе сигнала начала слова *sop* в регистре просто сохраняется значение пришедшего информационного байта (данный сигнал держится 1 такт). После чего начинается нормальная работа по схеме Горнера.

Модуль содержит 15 блоков расчетов синдромов, т.к. расчет S_0 сводится к суммированию по модулю 2 всех приходящих символов, что реализовано на регистре с минимумом комбинационной логики (рис. 4).

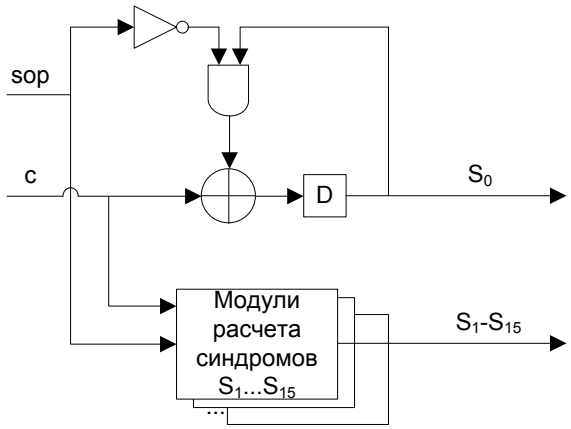

Рис. 4. Общая структура блока вычисления синдромов.

2.2. Расчет корректоров локаторов и решение ключевого уравнения

Рассмотрим коррекции, которые требуется внести для обеспечения возможности непрерывной обработки. Многочлен локаторов ошибок

$$\Lambda(x) = \prod_{i=1}^{\upsilon}(1 - xX_i) = \Lambda_\upsilon x^\upsilon + \Lambda_{\upsilon-1}x^{\upsilon-1} + ... + \Lambda_o$$

имеет корни $x_i = X_i^{-1}$, обратные к локаторам ошибок, т.е. $X_i^{-1} \cdot X_i = 1$.

В случае, если получено сообщение укороченного кода Рида-Соломона длины *n-b*, многочлен сообщения записывается как

$$c(x) = \sum_{i=0}^{n-b-1} c_i \cdot x^i$$

и степень старшего члена равна *n-b-1*, чему соответствует локатор $X = \alpha^{n-b-1}$ и корень многочлена локаторов $x = \alpha^{-(n-b-1)} = \alpha^{-n+b+1} = \left\{\alpha^{-n} = \alpha^{0}\right\} = \alpha^{b+1}$. Перебор в процедуре Ченя должен начинаться с α^{b+1} и далее до α^{255}. Например, если *b=1*, то имеет место укорочение на 1 байт, и поиск должен начинаться с α^{2}.

Заметим, что при решении ключевого уравнения может быть сформирован многочлен локаторов, имеющий корни $x = \alpha^{i}$, $i < b+1$. Такой многочлен соответствует нахождению ошибок в позициях, которые не передавались по каналу связи и были приняты при расчете синдромов равными нулю, что называется неисправимой конфигурацией ошибок. Подобная ситуация может возникнуть при нахождении ошибок в позициях $i \geq n$.

Процедуру Ченя для полного кода можно описать следующей последовательностью вычисления значений многочлена локаторов ошибок, начиная с α^{1}:

$$\Lambda(\alpha^{1}) = \Lambda_{\upsilon}(\alpha^{1})^{\upsilon} + \Lambda_{\upsilon-1}(\alpha^{1})^{\upsilon-1} + ... + \Lambda_{o}$$

$$\Lambda(\alpha^{2}) = \Lambda_{\upsilon}(\alpha^{2})^{\upsilon} + \Lambda_{\upsilon-1}(\alpha^{2})^{\upsilon-1} + ... + \Lambda_{o}$$

$$\dots$$

$$\Lambda(\alpha^{255}) = \Lambda_{\upsilon}(\alpha^{255})^{\upsilon} + \Lambda_{\upsilon-1}(\alpha^{255})^{\upsilon-1} + ... + \Lambda_{o}$$

Для сокращенного кода последовательность вычисления значений многочлена локаторов ошибок, начиная с α^{b+1}:

$$\Lambda(\alpha^{b+1}) = \Lambda_{\upsilon}(\alpha^{b+1})^{\upsilon} + \Lambda_{\upsilon-1}(\alpha^{b+1})^{\upsilon-1} + ... + \Lambda_{o} = \Lambda_{\upsilon}(\alpha^{b})^{\upsilon}(\alpha^{1})^{\upsilon} + \Lambda_{\upsilon-1}(\alpha^{b})^{\upsilon-1}(\alpha^{1})^{\upsilon-1} + ... + \Lambda_{o}$$

$$\Lambda(\alpha^{b+2}) = \Lambda_{\upsilon}(\alpha^{b+2})^{\upsilon} + \Lambda_{\upsilon-1}(\alpha^{b+2})^{\upsilon-1} + ... + \Lambda_{o} = \Lambda_{\upsilon}(\alpha^{b})^{\upsilon}(\alpha^{2})^{\upsilon} + \Lambda_{\upsilon-1}(\alpha^{b})^{\upsilon-1}(\alpha^{2})^{\upsilon-1} + ... + \Lambda_{o}$$

$$\dots$$

$$\Lambda(\alpha^{255}) = \Lambda_{\upsilon}(\alpha^{255})^{\upsilon} + \Lambda_{\upsilon-1}(\alpha^{255})^{\upsilon-1} + ... + \Lambda_{o} = \Lambda_{\upsilon}(\alpha^{b})^{\upsilon}(\alpha^{255-b})^{\upsilon} + \Lambda_{\upsilon-1}(\alpha^{b})^{\upsilon-1}(\alpha^{255-b})^{\upsilon-1} + ... + \Lambda_{o}$$

Аналогичные рассуждения верны и для вычисления значения многочлена значений ошибок $\Omega(x)$.

Пусть *k=n-b* – длина слова укороченного кода Рида-Соломона. Тогда $\alpha^{b} = \alpha^{n-k} = \alpha^{-k}$ и итерации процедуры Ченя можно записать как

$$\Lambda(\alpha^{b+i}) = \Lambda_{\upsilon}(\alpha^{\upsilon})^{-k}(\alpha^{\upsilon})^{i} + \Lambda_{\upsilon-1}(\alpha^{\upsilon-1})^{-k}(\alpha^{\upsilon-1})^{i} + ... + \Lambda_{o}.$$

Назовем υ чисел $\left(\alpha^{-j}\right)^{k}$, $j = 1:\upsilon$, корректорами локаторов ошибок lc_j. При домножении коэффициентов многочлена локаторов ошибок и многочлена значений ошибок на корректоры локаторов возможно использование реализации алгоритмов Ченя и Форни для полного кода, начинающих поиск корней с α^{1}.

Модуль, который реализует расчет корректоров локаторов при приеме слова, состоит из однотипных ячеек, структура которых приведена на рис. 5. По сигналу *sop*, обозначающему начало слова, в регистр *j*-й ячейки загружается начальное значение α^{-j}. Далее начинаются итерации последовательного домножения значения в регистре на α^{-j} при приеме каждого символа. Таким образом, если сообщение содержит *k* символов, на *k*-й

итерации в регистре будет храниться значение $\left(\alpha^{-j}\right)^k$, которое будет защелкнуто модулем вычисления локаторов и значений ошибок по сигналу завершения слова.

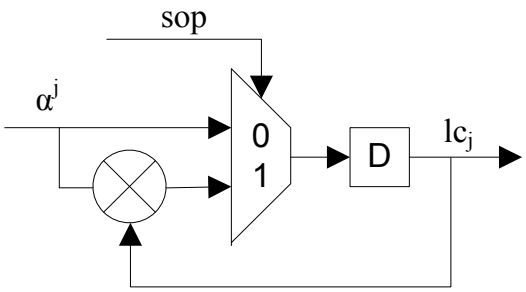

Рис. 5. Элемент вычисления корректора локатора

Модуль также содержит счетчик, определяющий число принятых символов в слове для последующего ограничения числа итераций алгоритмов Ченя и Форни.

Рассчитанные корректоры локаторов передаются в декодер. В данной работе реализация решателя подробно не рассматривается, так как в рамках предлагаемого подхода декодирования укороченных кодов могут использоваться любые перечисленные выше алгоритмы решения ключевого уравнения. Рассмотрим модификацию реализации алгоритма Берлекемпа-Месси, приведенной в [8], в которой осуществление домножения на корректоры локаторов осуществляется в модуле решения ключевого уравнения на имеющихся в нем умножителях после решения ключевого уравнения (шаг 5), что позволит уменьшить количество требуемых ресурсов.

Входы:

S_i, i=0,1,2,…,2t-1;

lc_j, j=1,2,, υ

Переменные:

B(x) – многочлен корректора;

Δ – невязка;

Γ – масштабирующий коэффициент, вводится для устранения деления;

K – текущая длина регистра сдвига;

Инициализация.

$\Lambda_0(0)=B_0(0)=1$, $\Lambda_i(0)=B_i(0)=0$ для i=1,2,…,t; $K(0)=0$; $\Gamma(0)=1$;

Итерации:

от r=0 до 2t-1 с шагом 1

{

Шаг 1. Вычисление невязки:

$$\Delta(r)=S_r*\Lambda_0(r)+S_{r-1}*\Lambda_1(r)+\ldots+S_{r-t}*\Lambda_t(r)$$

Шаг 2. Коррекция многочлена локаторов:

$$\Lambda_i(r+1)=\Gamma(r)*\Lambda_i(r)-\Delta(r)*B_{i-1}(r),\ i=0,1,\ldots,t$$

Шаг 3. Обновление корректора и длины:

если ((Δ(r)!=0) и (K(r)*2<=r))

{

B_i(r+1)= Λ_i(r), i=0,1,…,t

Γ(r+1)= Δ(r)

K(r+1)=r-K(r)

}

иначе

{

B_i(r+1)= B_{i-1}(r), i=0,1,…,t

Γ(r+1)= Γ(r)

K(r+1)=K(r)

}

}

Шаг 4. Расчет многочлена ошибок:

от i=2t до 3t-1 с шагом 1

$\Omega_i(2t)=S_i * \Lambda_0(2t)+ S_{i-1} * \Lambda_1(2t)+\ldots+ S_0 * \Lambda_i(2t)$

Шаг 5. Домножение на корректоры:

$\Lambda_i(2t)= \Lambda_i(2t) * lc_i;$

$\Omega_i(2t)= \Omega_i(2t) * lc_i;$

Выходы:

$\Lambda_i(2t)$, i=0,1,…,t

$\Omega_i(2t)$, i=0,1,…, t-1

На шаге 5 осуществляется домножение вычисленных коэффициентов многочленов локаторов ошибок и значений ошибок на корректоры локаторов ошибок. Для этого используется умножитель, используемый на шаге 2. Это увеличивает количество мультиплексоров, подающих данные на умножители. так как в зависимости от состояния схемы уиножитель должен умножать 4 различных пары операндов (2 пары на шаге 2 и 2 при домножении на корректоры), однако, для рассматриваемых семейств ПЛИС эти мультиплексоры объединяются синтезатором с логикой умножителей и практически не занимают дополнительных элементов. Похожий подход для другой реализации алгоритма Берлекемпа-Месси, использован в [11], однако, в рамках подхода не выделяется отдельный модуль для вычисления корректоров локаторов, общих для $\Lambda(x)$ и $\Omega(x)$, и не предлагается эффективный метод их вычисления.

2.3. Коррекция ошибок

Простейшим путем нахождения корней многочлена $\Lambda(x)$ является метод проб и ошибок, известный как процедура Ченя. Эта процедура состоит в последовательном вычислении $\Lambda(\alpha^j)$ для каждого j и проверки полученных значений на нуль. Если величина

$\Lambda(\alpha^j)$ равна нулю, то $x_j = \alpha^j$ является обратным к корню многочлена локаторов ошибок. Так как в $GF(n+1)$ примитивный элемент образует циклическую группу по умножению, порождающую все элементы поля, то $\alpha^n = \alpha^0 = 1$, и $x_j \cdot X_j = \alpha^j \cdot \alpha^{-j} = \alpha^j \cdot \alpha^{n-j} = 1$.

Тогда многочлен сообщения

$$c(x) = \sum_{i=0}^{n-1} c_i \cdot x^i$$

для полного кода длины *n* имеет степень старшего члена *n-1*, чему соответствует локатор $X = \alpha^{n-1}$ и корень многочлена локаторов $x = \alpha^1$. Так как сообщение принимается начиная с символов c_i, соответствующих старшему члену $c(x)$, то перебор в процедуре Ченя должен начинаться с α^1 и далее до α^{255}.

В данной работе были использованы подход и структура, близкие к рассмотренным в [7]. На рис. 6 представлен элемент, реализующий перебор по алгоритму Ченя. В начальный момент времени в регистр по сигналу *load,* загружается значение *i*-го коэффициента многочлена локаторов $\Lambda(x)$ ошибок или значений ошибок $\omega(x)$. Загруженные значения на каждом такте домножаются на α^i, где *i* соответствует степени коэффициента в многочлене локаторов.

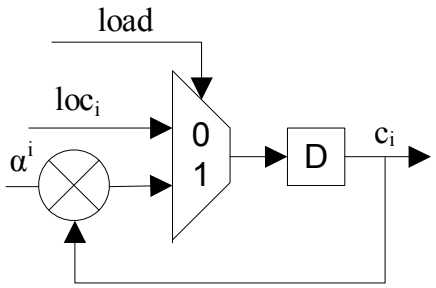

Рис. 6. Элемент перебора.

Структура модуля, реализующего процедуру Ченя, приведена на рис. 7. Модуль, приведенный на рис 6, обозначен как Ci.

На выходе данной части модуля имеем значение $\Lambda(\alpha^i)$ и его формальную производную $\Lambda'(\alpha^i)$. Законность вычисления формальной производной таким образом покажет следующая математическая выкладка.

$$\Lambda(x) = \sum_{i=0}^{8} \Lambda_i \cdot x^i;$$

$$\Lambda'(x) = 8 \cdot \Lambda_8 \cdot x^7 + 7 \cdot \Lambda_7 \cdot x^6 + 6 \cdot \Lambda_6 \cdot x^5 + 5 \cdot \Lambda_5 \cdot x^4 + 4 \cdot \Lambda_4 \cdot x^3 + 3 \cdot \Lambda_3 \cdot x^2 + 2 \cdot \Lambda_2 \cdot x^1 + \Lambda_1 =$$
$$= 7 \cdot \Lambda_7 \cdot x^6 + 5 \cdot \Lambda_5 \cdot x^4 + 3 \cdot \Lambda_3 \cdot x^2 + \Lambda_1 = \Lambda_7 \cdot x^6 + \Lambda_5 \cdot x^4 + \Lambda_3 \cdot x^2 + \Lambda_1 = \sum_{i-нечетное} \Lambda_i \cdot x^i \Big/ x.$$

Но в нашем случае данное значение использовать неудобно, т.к. в алгоритме Форни придется использовать деление на x. Поэтому в модуле, реализующем процедуру Ченя, вместо формальной производной $\Lambda'(\alpha^i)$ сразу вычисляется значение $\Lambda'(\alpha^i)_{new}$:

$$\Lambda'(x)_{new} = \Lambda'(x) \cdot x = \sum_{i-нечетное} \Lambda_i \cdot x^i.$$

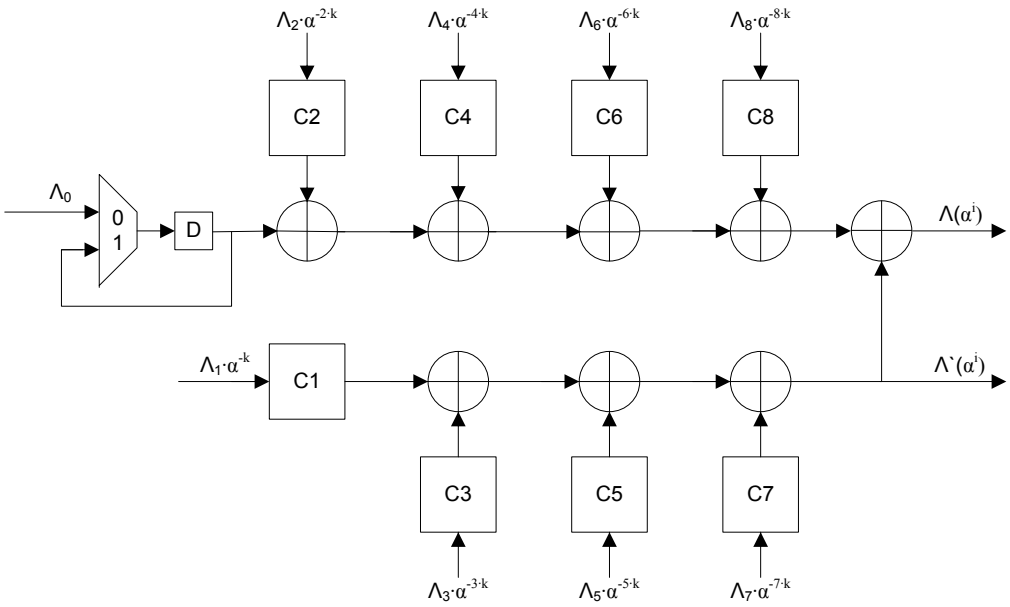

Рис. 7. Модуль, реализующий процедуру Ченя.

На рис. 8 представлена схема, представляющая вторую половину модуля – алгоритм Форни.

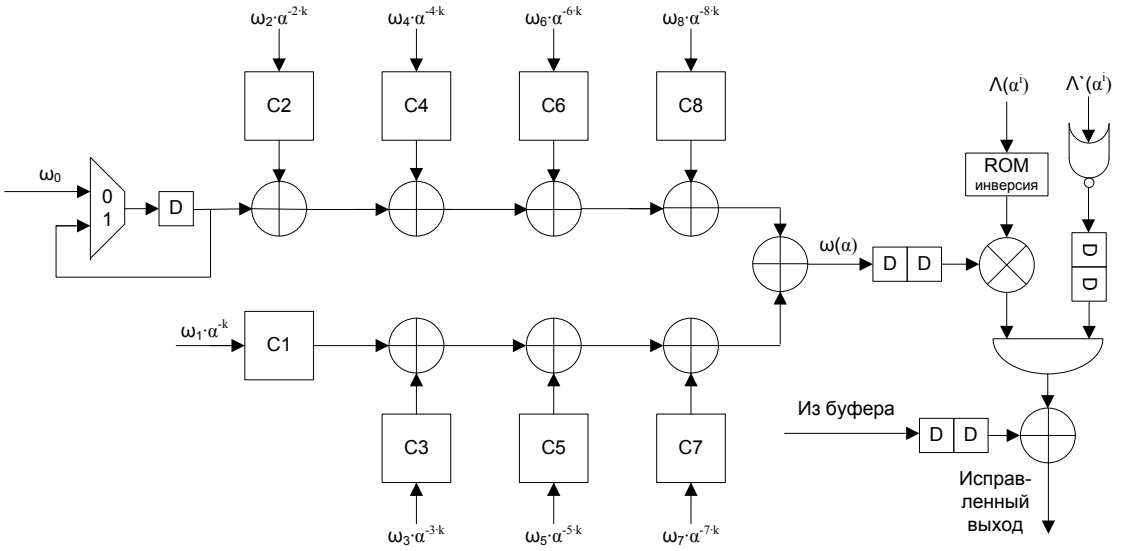

Рис. 8. Модуль, реализующий алгоритм Форни.

Левая часть схемы структурно повторяет схему реализации процедуры Ченя, в результате чего получаем значение $\omega(\alpha^i)$. Модуль памяти “ROM инверсия” позволяет заменить деление на формальную производную умножением на ее обратное. В модуле памяти хранится предвычисленная таблица обратных значений. Причиной табличной реализации вычисления обратного является сложность комбинационной реализации функции вычисления обратного в поле Галуа, что может приводить к возникновению критического пути в комбинационной логике. Модуль памяти вносит задержку в 2 такта, поэтому в остальные пути в схеме внесены элементы задержки на один такт, обозначенные символом D.

В случае, если $\Lambda(\alpha^i)$ имеет нулевое значение, текущая позиция соответствует найденной ошибке и происходит коррекция значения с выходного буфера путем сложения с выходом алгоритма Форни. Таким образом, выходное значение модуля можно описать следующим выражением:

$$co_{n-i} = c_{n-i} + \frac{\omega(\alpha^i)}{\Lambda'(\alpha^i)_{new}} = c_{n-i} + \frac{\omega(\alpha^i)}{x \cdot \Lambda'(\alpha^i)},$$

где n – длина слова, i – номер текущей (обрабатываемой) позиции, c – входной символ.

3. Сравнение результатов

Используем для сравнения функцию декодера Рида-Соломона Reed-Solomon II IP-Core 14.0 и САПР Quartus II 14.0 фирмы Altera. Для синтеза использовалась ПЛИС Altera семейства Cyclone IV EP4CE115F29C7. К сожалению, в современной практике проектирования цифровых схем исходные коды и описание используемых алгоритмов в коммерческих продуктах, как правило, недоступны, поэтому сравнение возможно лишь по объему и быстродействию реализаций. Декодеры были сконфигурированы в соответствии с требованиями стандарта DVB [1,2] для декодирования укороченных кодов, производных от полного кода (255,239,8). Приведем сравнительные таблицы по затрачиваемым ресурсам и быстродействию декодеров (таблица 1).

Таблица 1. Сравнение декодеров

Декодер	Спроектированный	Altera
Частота, МГц	146	153
Число логических элементов	2666	2612
Память ОЗУ, биты	6144	5120
Задержка, такты	54	112

Несмотря на то, что декодеры имеют схожий объем и быстродействие, реализованный в данной работе декодер имеет вдвое меньшую задержку от момента подачи последнего символа слова до появления первого символа на выходе. Кроме того, реализованный декодер имеет важное функциональное преимущество перед декодером фирмы Altera - он позволяет обрабатывать слова различной длины в непрерывном потоке, чего не могут делать декодеры фирмы Altera. Декодеру фирмы Altera требуется перерыв между словами

для дополнения слова укороченного кода до полной длины (в такой ситуации декодер выставляет флаг занятости и не принимает входные данные), что значительно снижает пропускную способность системы при обработке слов укороченного кода.

В реализованном декодере осуществляется буферизация результатов работы алгоритма Берлекэмпа-Месси для нескольких поступивших слов для обеспечения непрерывности обработки данных при прохождении через декодер слов различной длины. Для выполнения требований стандарта EN 301 790 [2], определяющих минимальную и максимальную длину слов, для этого потребовался буфер FIFO глубиной 4 и разрядностью 144 бита, который был реализован на триггерах в логических элементах. Также осуществляется расчет корректоров локаторов, занимающий еще 89 логических элементов.

Таким образом, если отказаться от возможности обрабатывать непрерывный поток данных, это позволит сократить объем примерно на 650 логических элементов, что даст объем реализации примерно в 2000 логических элементов. Экспериментально также установлено, что при этом за счет упрощения модуля решателя ключевых уравнений увеличится его максимальная частота тактирования. Такой вариант декодера будет иметь функциональность и объем, примерно соответствующие декодеру фирмы Altera, при меньшем объеме, большем быстродействии и значительно меньшей задержке выхода в тактах.

Заключение

Несмотря на появление новых, более перспективных кодов, в ряде задач, а также для поддержки совместимости в существующих системах оправдано использование кодов Рида-Соломона и требуется эффективная реализация их декодеров, поэтому создание такого проектного решения представляется актуальным. Предложенная методика декодирования укороченных кодов с расчетом корректоров локаторов может быть применена для других кодов Боуза-Чоудхури-Хоквингема, в том числе для широко применяемых в настоящее время двоичных кодов БЧХ с большой длиной слова.

Список литературы

1. ETSI EN 300 421 Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for 11/12 GHz satellite services. 1997. 24 p.
2. ETSI EN 301 790 Digital Video Broadcasting (DVB); Interaction channel for satellite distribution systems. 2005. 176 p.
3. D. Gorenstein, N. Zierler. A Class of Error Correcting Codes in p^m Symbols // J SIAM. Vol. 9. 1961. Pp. 207-214.
4. Р. Морелос-Сарагоса. Искусство помехоустойчивого кодирования. Методы, алгоритмы, применение. М.: Техносфера, 2006. 320 с.
5. M. Sudan. Decoding of Reed-Solomon Codes Beyond the Error-Correction Bound // Journal of Complexity. 1997. Vol. 13. No.1. Pp. 180-193. DOI: 10.1006/jcom.1997.0439

6. Dilip V. Sarwate, Naresh R. Shanbhag. High-Speed Architectures for Reed–Solomon Decoder // IEEE Transactions on VLSI Systems. 2001. Vol. 9. No. 5. Pp. 641-655. DOI: 10.1109/92.953498
7. Hanho Lee. High-Speed VLSI Architecture for Parallel Reed–Solomon Decoder // IEEE Transactions on VLSI Systems. 2003. Vol. 11. No. 2. Pp. 288-294. DOI: 10.1109/TVLSI.2003.810782
8. Федоров С.В. Аппаратная реализация решателя ключевых уравнений Берлекэмпа-Месси для кодов Рида-Соломона на ПЛИС // Наука и Образование. МГТУ им. Н.Э.Баумана. Электрон. журн. 2011. №7. С. 1-11. Режим доступа: http://technomag.bmstu.ru/doc/198028.html (дата обращения: 23.05.2016)
9. Todd K. Moon. Error Correction Coding: Mathematical Methods and Algorithms. Wiley-Interscience, 2005. 759 p.
10. Shin-Lin Shieh, Shuenn-Gi Lee. Wern-Ho Sheen. A low-latency decoder for punctured/shortened Reed-Solomon codes // IEEE 16th International Symposium on Personal, Indoor and Mobile Radio Communications. 2005. Vol. 4. Pp. 2547-2551. DOI: 10.1109/PIMRC.2005.1651903
11. Hoyoung Yoo, Youngjoo Lee. Low-latency area-efficient decoding architecture for shortened Reed-Solomon codes // SoC Design Conference (ISOCC). 2012. Pp. 223-226. DOI: 10.1109/ISOCC.2012.6407080

Science & Education of the Bauman MSTU
Electronic journal
ISSN 1994-0408

Science and Education of the Bauman MSTU, 2016, no. 06, pp. 184–199.

DOI: **10.7463/0616.0842221**

Received: 13.05.2016
Revised: 27.05.2016

FPGA-based Implementation of a Streaming Decoder for Shortened Reed-Solomon Codes

S.V. Fedorov[1,*], V.I. Romashkin[2], K.M. Vyalyh[1]

*svf@bmstu.ru

[1]Bauman Moscow State Technical University, Moscow, Russia
[2]National Research Centre "Kurchatov Institute", Moscow, Russia

Keywords: codes, error correcting codes, block codes, Reed-Solomon code, BCH codes, FPGA

The traditional approaches to decoding shortened block codes are the padding of received codeword with zeros or correction of the syndrome polynomial values. Such approaches make stream-oriented decoding of continuous input data stream impossible. The paper offers a new technique for decoding the shortened Reed-Solomon codes. Its hardware implementation allows us to provide stream processing, reduce latency in clock cycles, and decrease a required quantity of hardware resources as compared to decoder implementation with padding of received packet with zeros. A distinctive feature of the proposed technique is that the decoder processes a stream of code words of different lengths without changing their structure and insertion of additional delays and that it is possible to use the modules of existing Reed-Solomon decoders for full codeword length. To enable this, a notion of error locator corrector for shortened code is introduced and a technique of their calculation is offered. Error locator correctors are calculated during the reception of a codeword in parallel to syndrome polynomial calculation. Their values allow us to modify the polynomial values of locators and errors at the output of the key equation solver. The paper considers a shortened code decoder that is implemented on the full code decoder modules based on Berlekamp-Massey algorithm, describes architecture of additional modules and required modifications of the algorithm. It shows the way to save hardware resources by using the multipliers in Berlekemp-Massey key equation solver to correct values. The Altera FPGA-based decoder has been tested in and compared to the Reed-Solomon II decoder IP from Altera.

References

1. *ETSI EN 300 421 Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for 11/12 GHz satellite services*. 1997. 24 p.
2. *ETSI EN 301 790 Digital Video Broadcasting (DVB); Interaction channel for satellite distribution systems*. 2005. 176 p.

3. D. Gorenstein, N. Zierler. A Class of Error Correcting Codes in p^m Symbols. *J SIAM*, vol. 9, 1961, pp. 207-214.
4. R. Morelos-Zaragosa. *Iskusstvo pomekhoustoychivogo kodirovaniya. Metody, algoritmy, primenenie* [The Art of Error-Correcting Coding. Methods, Algorithms, Applications]. Moscow, Technosfera Publ., 2006, 320 p. (in Russian).
5. M. Sudan. Decoding of Reed-Solomon Codes Beyond the Error-Correction Bound. *Journal of Complexity*, 1997, vol. 13, no.1, pp. 180-193. DOI: 10.1006/jcom.1997.0439
6. Dilip V. Sarwate, Naresh R. Shanbhag. High-Speed Architectures for Reed–Solomon Decoder. *IEEE Transactions on VLSI Systems*, 2001, vol. 9, no. 5, pp. 641-655. DOI: 10.1109/92.953498
7. Hanho Lee. High-Speed VLSI Architecture for Parallel Reed–Solomon Decoder. *IEEE Transactions on VLSI Systems*, 2003, vol. 11, no. 2. pp. 288-294. DOI: 10.1109/TVLSI.2003.810782
8. Fedorov S.V. Hardware implementation of Berlekamp-Massey key equation solver for Reed-Solomon codes in FPGA. *Nauka i obrazovanie MGTU im. N.E. Baumana = Science and Education of the Bauman MSTU*, 2011, no. 7, pp. 1-11. Available at: http://technomag.bmstu.ru/doc/198028.html (accessed 23.05.2016) (in Russian).
9. Todd K. Moon. *Error Correction Coding: Mathematical Methods and Algorithms. Wiley-Interscience*, 2005. 759 p.
10. Shin-Lin Shieh, Shuenn-Gi Lee. Wern-Ho Sheen. A low-latency decoder for punctured/shortened Reed-Solomon codes. *IEEE 16th International Symposium on Personal, Indoor and Mobile Radio Communications*, 2005, Vol.4, pp. 2547-2551. DOI: 10.1109/PIMRC.2005.1651903
11. Hoyoung Yoo, Youngjoo Lee. Low-latency area-efficient decoding architecture for shortened Reed-Solomon codes. *SoC Design Conference (ISOCC)*, 2012, pp. 223-226. DOI: 10.1109/ISOCC.2012.6407080