электронное научно-техническое издание
НАУКА и ОБРАЗОВАНИЕ
Эл № ФС 77 - 30569. Государственная регистрация №0421100025. ISSN 1994-0408

Аппаратная реализация решателя ключевых уравнений Берлекэмпа-Месси для кодов Рида-Соломона на ПЛИС

 # 07, июль 2011

автор: Федоров С. В.

МГТУ им. Н.Э. Баумана
svf@iu3.bmstu.ru

Задача декодирования кодов Рида-Соломона может быть решена различными способами. В [1] приведен обзор различных алгоритмов декодирования и сравнение эффективности их реализации. Наиболее ресурсоемкой частью декодера является схема решения ключевого уравнения. В [2, 3] приведено сравнение наиболее широко распространенных алгоритмов решения ключевого уравнения Берлекэмпа-Месси и Евклида с точки зрения аппаратной эффективности и быстродействия. В данной работе рассматривается реализация декодера на основе алгоритмов Берлекэмпа-Месси, Ченя и Форни [1, 2].

Рассмотрим процедуру определения позиций ошибок и вычисления их значений. Пусть в приемник поступило сообщение

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

где $c(x)$ – переданное сообщение, в которой в процессе передачи по каналу связи произошло ν ошибок, отображаемых многочленом $e(x)$.

Каждый ненулевой компонент $e(x)$ описывается парой элементов: Y_i – величина ошибки и X_i – локатор ошибки в позиции i. Y_i, X_i – элементы $GF(n)$, и элемент $X_i = \alpha^i$, $\alpha^i \in GF(n)$.

Для описания ошибок используются:

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

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

имеющий корни $x_i = X_i^{-1}$, $i = 1, \ldots, \nu$, обратные к локаторам ошибок, то есть $X_i^{-1} \cdot X_i = 1$.

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

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

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 поступившей комбинации кода Рида-Соломона.

Здесь

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

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

Значение $S_j = e(\alpha^j)$ вычисляется как:

$$S_j = r(\alpha^j) = c(\alpha^j) + e(\alpha^j) = e(\alpha^j),$$

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. Это становится понятным, если учесть, что степень $\Omega(x)$ не превышает $\nu - 1$ и поэтому справедливо:

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

где

$$[a(x)]_m^n = a_m x^m + a_{m+1} x^{m+1} + \ldots + a_n x^n,$$

Из этого уравнения необходимо получить ν уравнений для ν неизвестных коэффициентов Λ_k. Эти уравнения являются линейными. После нахождения многочлена $\Lambda(x)$ ключевое уравнение позволяет найти неизвестные компоненты вектора $e(x)$ и по ним выходной вектор декодера

$$c(x) = r(x) + e(x).$$

Решение ключевого уравнения является основной задачей декодирования кода Рида-Соломона. Прямолинейная реализация решения влечет за собой решение системы из 2t линейных уравнений. Для сокращения вычислительной сложности используются различные алгоритмы. Реализация декодеров на жесткой логике и ПЛИС рассмотрена в большом количестве работ, например, в [3, 4, 5]. Наиболее распространены алгоритмы, основанные на подходе Берлекэмпа-Месси [3] и алгоритмы, использующие алгоритм Евклида [4]. Алгоритмы с мягким решением, например алгоритм Судана, в аппаратных реализациях используются редко ввиду нерегулярности получаемой архитектуры.

Решение ключевого уравнения в алгоритме Берлекэмпа-Месси сводится к задаче итеративного построения минимального сдвигового регистра с линейной обратной связью, порождающего на выходе последовательность синдромов, соответствующую коэффициентам *S(x)* (см. рис. 1).

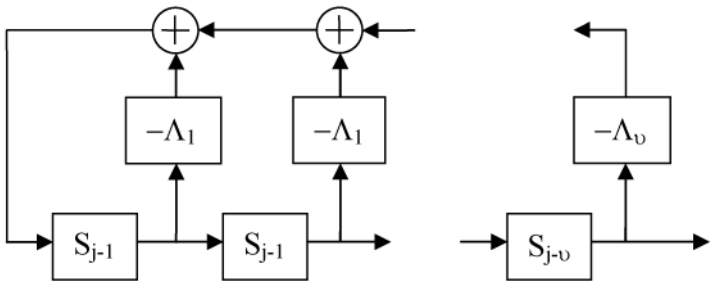

Рисунок 1. Сдвиговый регистр, определяющий многочлен локаторов

В алгоритме Берлекэмпа-Месси используется операция нахождения обратного в используемом поле, что значительно повышает сложность и снижает быстродействие аппаратуры, реализующей алгоритм. Имеются модификации алгоритма, не использующие делений (iBM). В работе [3] реализуется несколько архитектур на основе iBM и проводится их сравнение с реализациями алгоритма Евклида (см. таблицу 1). В таблицу также включена предлагаемая в данной работе архитектура CSiBM. Также в таблице 1 приведена оценка критического пути в комбинационной логике в виде задержки на используемых функциональных блоках.

Т А Б Л И Ц А 1

Архитектура	add	mult	reg	mux	Тактов	крит. путь
iBM (Блэйхут)	2t+1	3t+3	4t+2	t+1	3t	$>2(T_{mult}+T_{add})$
iBM (Берлекэмп)	3t+1	5t+3	6t+2	2t+1	2t	$>2(T_{mult}+T_{add})$
riBM	3t+1	6t+2	6t+2	3t+1	2t	$T_{mult}+T_{add}$
RiBM	3t+1	6t+2	6t+2	3t+1	2t	$T_{mult}+T_{add}$
Евклид	4t+2	8t+8	4t+4	8t+8	2t	$T_{mult}+T_{add}+T_{mux}$
Евклид итерат.	2t+1	2t+1	10t+5	14t+7	12t	$T_{mult}+T_{add}+T_{mux}$
CSiBM	2t+1	2t+2	5t+3	5t+1	6t	$T_{mult}+T_{add}+2T_{mux}$

Здесь

add – число сумматоров в $GF(2^8)$;

mult – число умножителей в $GF(2^8)$;

reg – число регистров разрядности 8;

mux – число мультиплексоров 2 в 1 разрядности 8;

В базисе ПЛИС высокой и сверхвысокой степени интеграции комбинационная логика реализуется в базисе таблиц перекодировки. Приведем объем требуемых ресурсов для операций над 8-ми разрядными символами для четырехвходовых таблиц перекодировки (ТП), которые используются в ряде семейств Stratix и Cyclone фирмы Altera, Virtex и Spartan фирмы Xilinx:

- add – 8 ТП (возможно сложение от 2 до 4 символов)
- mult – 50 ТП
- mux – 8 ТП

В таблице 1 для ряда архитектур дана неточная оценка задержки критического пути, связанная с различной эффективностью реализации дерева сумматоров в разных аппаратных архитектурах и дополнительной задержки в схеме управления.

Задержку критического пути в комбинационной схеме можно оценить по его длине - максимальному количеству ТП, через которые проходит сигнал.

- add – 1 ТП
- mult – 4 ТП
- mux – 1 ТП

В таблице 2 приведена оценка количества используемых ТП, регистров и длина критического пути в ТП.

Т А Б Л И Ц А 2

Архитектура	ТП	reg	тактов	крит. путь, ТП
iBM (Блэйхут)	166t+166	32t+16	3t	>10
iBM (Берлекэмп)	292t+166	48t+16	2t	>10
riBM	348t+116	48t+16	2t	5
RiBM	348t+116	48t+16	2t	5
Евклид	496t+480	32t+32	2t	6
Евклид итерат.	220t+114	80t+40	12t	6
CSiBM	156t+116	40t+24	6t	7

Логический элемент ПЛИС с четырехвходовой ТП содержит одну ТП и один регистр, которые могут использоваться одновременно. Таким образом, общее количество требуемых для реализации логических элементов можно примерно оценить как максимальное значение от количества регистров и ТП. Исходя из этого, для всех рассматриваемых архитектур решателей ключевого уравнения оценка ресурсоемкости может проводиться по количеству используемых ТП.

При синтезе комбинационных схем функции минимизируются синтезатором совместно, что может приводить к сокращению конечного объема проекта и количества слоев логики. Это существенно зависит от

применяемого синтезатора и архитектуры ПЛИС, однако соотношения для различных архитектур решателей, как правило, остаются неизменными.

В ряде задач, например, в системах цифрового телевидения [6,7] требуется обрабатывать пакеты укороченного кода длиной от 69 символов с t=8. При реализации потоковой обработки в темпе поступления информации с аппаратной точки зрения это накладывает ограничение по числу тактов работы алгоритма не более 69. Это сразу исключает из рассмотрения компактную итеративную реализацию алгоритма Евклида, требующую 96 тактов для t=8. Обычная (параллельная) реализация алгоритма Евклида требует намного большего, чем вариации алгоритма Берлекэмпа-Месси, количества умножителей, что значительно повышает объем проекта, так как умножитель является наиболее ресурсоемким функциональным блоком, используемым при декодировании. В дополнение к известным алгоритмам iBM в трактовке Блэйхута и Берлекэмпа, авторами работы [3] также предложены аппаратно избыточные алгоритмы riBM и RiBM, позволяющие достичь более высокой тактовой частоты. Однако, в пространстве решений нет промежуточного по быстродействию и объему решения между итерационным алгоритмом Евклида и различными реализациями алгоритма Берлекэмпа-Месси.

Для создания промежуточного решения CSiBM (cycle-shared iBM – iBM с разделением по тактам) используем в качестве основы алгоритм iBM в реализации [5]. Алгоритм iBM требует наименьшее количество умножителей и выдает результат (коэффициенты многочлена локаторов ошибок и многочлена значений ошибок) за 3t тактов. Для повышения тактовой частоты и уменьшения объема используемых ресурсов была реализована дополнительная конвейеризация с разделением использования умножителей между тактами. По этой причине алгоритм CSiBM требует 6t тактов.

Рассмотрим алгоритм Берлекэмпа-Месси без инверсий более подробно. Алгоритм требует 3t шагов. На первых 2t шагах осуществляется определение коэффициентов полинома $\beta^*\Lambda(x)$, на следующих t шагах определяются

коэффициенты полинома $\beta*\Omega(x)$, где β – некоторая скалярная величина. Так как умножение на β не влияет на корни полинома $\Lambda(x)$, а отношение $\Lambda(x)$ и $\Omega(x)$ не изменяется при умножении их на β, это не изменяет определяемые в дальнейшем алгоритмами Ченя и Форни положение и величины ошибок. Ниже приведен псевдокод алгоритма.

Входы:

S_i, i=0,1,2,…,2t-1;

Переменные:

B(x) – полином 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 полинома локаторов:

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

Шаг 3. Обновление корректора и длины:

если (($\Delta(r)!=0$) и ($K(r)*2<=r$))

{

$B_i(r+1)=\Lambda_i(r)$, i=0,1,…,t

$\Gamma(r+1)=\Delta(r)$

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

}

иначе

{

$B_i(r+1)= B_{i-1}(r)$, i=0,1,…,t

$Г(r+1)= Г(r)$

$K(r+1)=K(r)$

}

}

Шаг 4. Расчет полинома ошибок:

от i=2t до 3t-1 с шагом 1

$\Omega_i(2t)=S_i * Л_0(2t)+ S_{i-1} * Л_1(2t)+…+ S_0 * Л_i(2t)$

Выходы:

$Л_i(2t)$, i=0,1,…,t

$\Omega_i(2t)$, i=0,1,…, t-1

Аппаратная реализация данного алгоритма в предлагаемой архитектуре CSiBM использует два типа функциональных блоков. Шаг 1 реализуется модулем расчета невязки (рис. 2). Перед началом работы алгоритма осуществляется загрузка сдвигового регистра значениями синдромов S_i. Следует заметить, что на первых t циклах работы алгоритма согласно описанию шага 1 должны использоваться значения S_{-1}, S_{-2} и т.д., а не загруженные значения. Однако, так как начальные значения коэффициентов локаторов Λ, на которые они умножаются, равны нулю, это не влияет на работу алгоритма. Загруженные значения синдромов S, будут использоваться на следующих циклах при их перемещении по сдвиговому регистру. Вычисленное значение невязки сохраняется в регистре невязки.

В данном модуле критический путь от регистра, содержащего S, до регистра невязки состоит из умножителя и дерева сумматоров 8 в 1. Однако, так как одна таблица перекодировки реализует сложение до 4 операндов, данное дерево реализуется схемой с длиной критического пути $T_{mult}+2T_{add}$,

что меньше задержки $T_{mult}+T_{add}+3T_{mux}$, приведенной в табл. 1, которая ограничивает тактовую частоту в рассматриваемом ниже модуле коррекции.

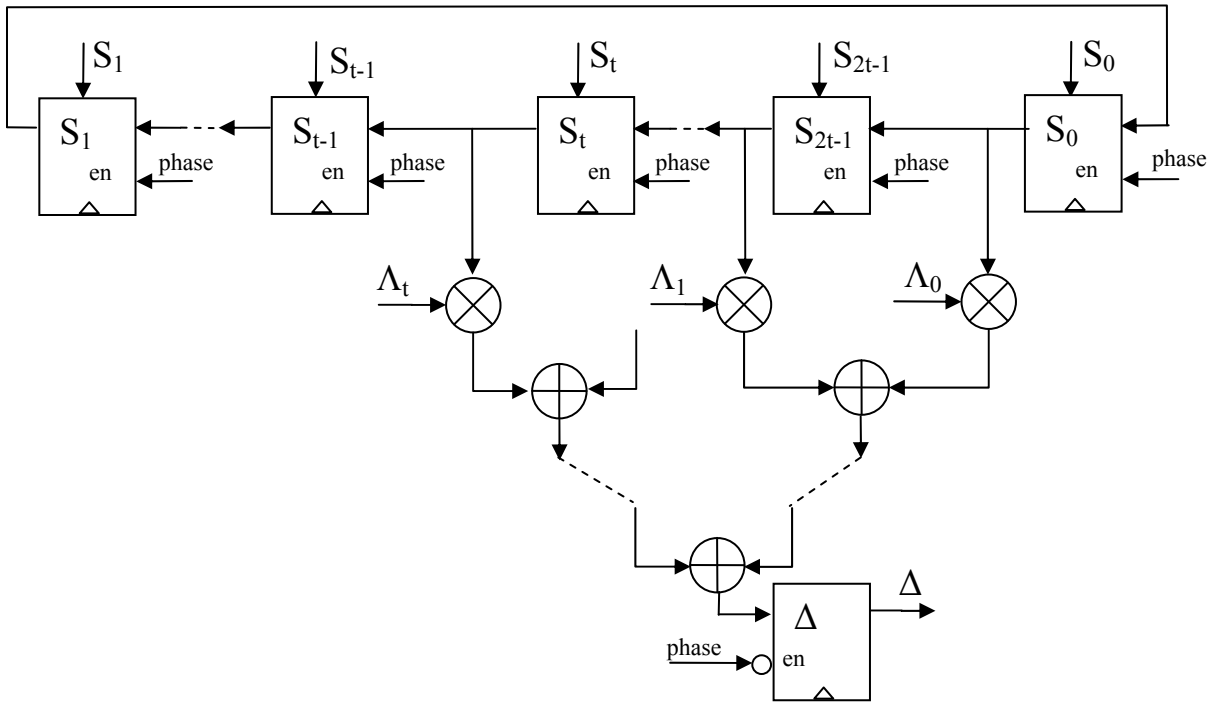

Рисунок 2. Модуль расчета невязки

Шаг 2 реализуется модулями коррекции, идентичными для всех коэффициентов Λ_i (рис. 3).

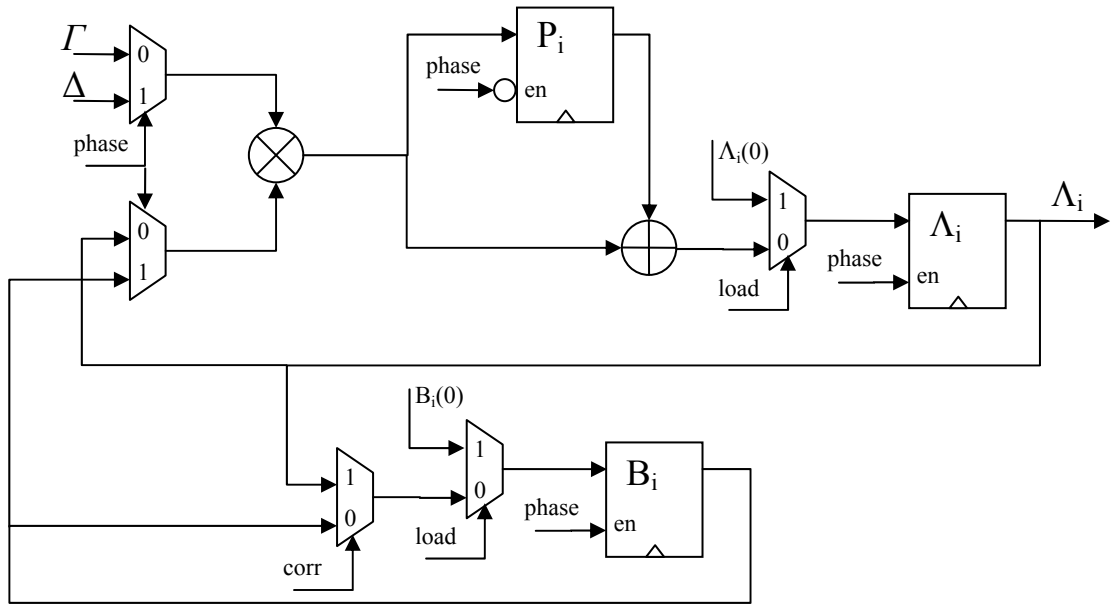

Рисунок 3. Модуль коррекции

Одна итерация циклов алгоритма выполняется за два такта. Выполнение шага 1 и умножения $\Gamma(r)^* \Lambda_i(r)$ на шаге 2 осуществляется на первом такте в модуле коррекции (phase=0), результат вычисления невязки сохраняется в регистре невязки (рис. 2). Выполнение второго умножения $\Delta(r)^* B_{i-1}(r)$ на шаге 2 и шага 3 осуществляется на втором такте (phase=1). Это разбивает критический путь в схеме и позволяет реализовать умножения, выполняемые на шаге 2 на одном умножителе. Критический путь в данной схеме проходит от выхода до входа регистра Λ_i через два мультиплексора, умножитель и сумматор.

Результатом работы первого цикла, состоящего из шагов 1, 2 и 3 являются коэффициенты многочлена локаторов ошибок Λ_i. На шаге 4 реализуется расчет коэффициентов многочлена ошибок Ω_i, при этом вычисленные коэффициенты Λ_i фиксируются и модуль расчета невязки используется для расчета Ω_i. Вычисленные значения Ω_i сохраняются в регистрах, использовавшихся для хранения B_i. Для этого и для начальной загрузки коэффициентов используется сигнал load.

Заключение

Разработан модифицированный вариант архитектуры решателя ключевых уравнений для декодера Рида-Соломона на основе алгоритма Берлекэмпа-Месси без инверсии. По сравнению с существующими реализациями, оптимизированными по объему или по быстродействию, предложенная схема предоставляет новый вариант баланса между объемом и быстродействием. Объем требуемых ресурсов меньше всех рассмотренных реализаций алгоритма iBM, но при этом достижимая частота работы схемы примерно в два раза выше по сравнению с базовым алгоритмом iBM. Время решения в тактах составляет 6t против 3t для алгоритма iBM, однако для большинства используемых кодов это меньше длины кода, что позволяет декодировать информацию в темпе поступления данных.

Литература

1. Р. Морелос-Сарагоса. Искусство помехоустойчивого кодирования. Методы, алгоритмы, применение. М.:Техносфера, 2005 – 320с.
2. Р.Блэйхут. Теория и практика кодов, контролирующих ошибки. Пер. с англ. – М.: Мир, 1986. – 576 с., ил.
3. D.V.Sarwate, N.R.Shanbhag. High-Speed Architectures for Reed-Solomon Decoders. IEEE Transcations on VLSI Systems, Vol. 9, No. 5, October 2001, pp. 641-655.
4. H. Lee. High-Speed VLSI Architecture for Parallel Reed-Solomon Decoder. IEEE Transactions on VLSI Systems, Vol. 11, No. 2, April 2003, pp. 288-294.
5. I. S. Reed, M. T. Shih, T. K. Truong, “VLSI design of inverse-free Berlekamp–Massey algorithm,” Proc. Inst. Elect. Eng., pt. E, vol. 138, pp. 295–298, Sept. 1991.
6. ETSI EN 300 421 Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for 11/12 GHz satellite services.
7. ETSI EN 301 790. Digital Video Broadcasting (DVB); Interaction channel for satellite distribution systems.