УДК 681.322

ЛЕПЕХА В.Л.,
ЛЕСИК Т.М.,
ФРАНКО Р.А.

РЕАЛИЗАЦИЯ ДЕКОДЕРА РИДА-СОЛОМОНА В ПЛИС

Рассматриваются вопросы реализации декодера Рида-Соломона с учетом архитектурных особенностей программируемых логических интегральных схем (ПЛИС). Разработанный с учетом этих особенностей быстродействующий декодер Рида-Соломона может найти применение в телекоммуникациях и системах хранения данных.

Questions of the Reed-Solomon decoder implementation taking into account the FPGA architecture properties are considered. The designed high-speed Reed-Solomon decoder can be utilized in telecommunications and data storing arrays.

Введение

Современные технологии хранения и передачи данных невозможны без адекватных средств их защиты от потерь. Одним из наиболее эффективных методов помехоустойчивого кодирования данных является метод Рида-Соломона [1]. При этом декодирование кодов Рида-Соломона представляет собой решение сложной неоднородной математической задачи с применением нестандартной арифметики полей Галуа. В результате этого, программная реализация декодера Рида-Соломона, как правило, имеет низкую скорость реализации. Поэтому в подавляющем числе случаев такой декодер реализован аппаратно в виде спецвычислителя.

Применение программируемых логических интегральных схем (ПЛИС) дает возможность в сжатые сроки с минимальными затратами разработать сложную систему на кристалле (СНК). Реализация декодера Рида-Соломона в ПЛИС дает возможность разрабатывать на его основе новые СНК для таких приложений, как сохранение баз данных большого объема, прием и передача спутниковых, мультимедийных и телевизионных сигналов и многие другие.

Такая реализация СНК в ПЛИС дает возможность перенастраивать устройство на различные стандарты кодирования информации, его модернизировать без изменения схемы, изменять длину посылок, количество исправляемых ошибок и т.п.

При этом элементный базис ПЛИС существенно отличается от базиса заказной СБИС. Это приводит к тому, что блоки, разработанные для СБИС, будут реализованы в ПЛИС неэффективно.

В статье исследуются вопросы реализации алгоритма Рида-Соломона с учетом элементного базиса современных ПЛИС, возможностей перенастройки алгоритма. В качестве изменяемых параметров декодера выступает длина кодированного слова, количество исправляемых ошибок, характеристический полином поля Галуа.

Алгоритм декодирования Рида-Соломона

Декодирование кодов Рида-Соломона представляет собой довольно сложную задачу. Типовая схема декодирования, получившая название авторегрессионого спектрального метода декодирования, состоит из следующих шагов [2, 3].

– прием закодированного кадра

– вычисления синдрома ошибки (синдромный декодер);

– построения полинома ошибок, осуществляемое либо с помощью высокоэффективного, но сложного алгоритма Берлекэмпа-Месси, либо посредством простого, но медленного Евклидового алгоритма;

– нахождения корней полинома ошибок, которое обычно выполняется перебором (алгоритм Ченя);

– определения характера ошибки, сводящееся к построению маски, вычисляемой на основе обращения алгоритма Форни или другого алгоритма обращения матрицы;

– исправления ошибочных символов путем наложения маски на информационное

слово и последовательного инвертирования всех искаженных бит операцией XOR.

Структурная схема такого алгоритма представлена на рис. 1.

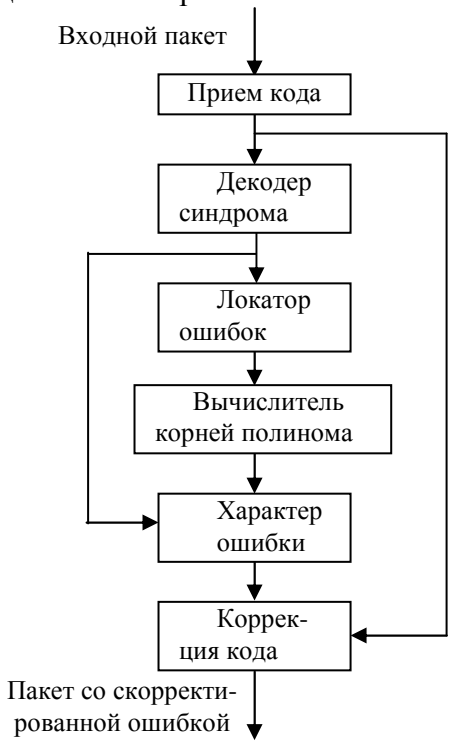

Рис.1. Алгоритм авторегрессионого спектрального декодера Рида-Соломона

Выбор элементной базы

Анализ вышеприведенного алгоритма показывает, что основными операциями являются векторное умножение, деление и возведение в степень в поле Галуа. Это довольно трудоемкие операции, особенно возведение в степень и деление, так как они выполняются в нестандартной системе счисления. Отметим, что в большинстве стандартных декодеров, как и в примерах ниже, вычисления выполняются в поле Галуа GF(2^8).

Ниже будет приведено сравнение реализации декодера на обычном микропроцессоре и ПЛИС. Базовая операция умножения A = B*C может быть выполнена на специализированном комбинационном умножителе чисел Галуа, который имеет сравнительно большие аппаратные затраты. Кроме того, для возведения в m-ю степень необходимо более $\log_2 m$ умножений, а для деления – 16 умножений [1]. Если выполнять вычисления с помощью таблиц, то как умножение, так и другие операции можно выполнять существенно быстрее. Тогда вычисление операции умножения имеет следующий вид при записи на языке VHDL:

```
B0 = tabl0(B) -- выбор операнда из таблицы
C0 = tabl0(C) -- выбор операнда из таблицы
A0 = B0 + C0 -- суммирование в G2^8
A = tabl1(A0) -- выбор операнда из таблицы
```

Здесь tabl0 и tabl1 – таблицы логарифмов и антилогарифмов чисел в поле Галуа размером 256 входов. Как видим, операция умножения будет происходить за 4 макрокоманды, а с учетом организации микропроцессора – за 10 – 15 команд. Учитывая, что, например, для исправления 8 ошибок (часто встречаемый случай) требуется умножение векторов с 16 элементами, такая векторная операция может занять 160 – 240 тактов работы микропроцессора.

При реализации на ПЛИС эти операции можно распараллелить. Современные ПЛИС имеют довольно большое количество независимых блоков памяти, что позволяет сделать такое векторное умножение за 1 – 3 такта. Кроме того, возможны и другие способы ускорения вычислений, которые, в совокупности, дают возможность значительно ускорить реализацию алгоритма.

Исходя из этих рассуждений, сделан вывод о целесообразности реализации декодера Рида-Соломона на ПЛИС с арифметикой, реализованной на таблицах.

Реализация декодера

Как было указано выше, основная операция данного алгоритма – векторное умножение в поле Галуа. Реализация умножителя выполняющего операцию умножение (деления) в G8 за 2 такта показана ниже:

```
d0<=conv_std_logic_vector(rom1(conv_integer(a)),8);
d1<=conv_std_logic_vector(rom1(conv_integer(b)),8
);
sm1 <=ext(d0,9) + ext(d1,9) + 1
     when m_d = '0' else ext(d0,9) - ext(d1,9) -
0;
b1 <= not (m_d xor sm1(8));
sm2 <= sm1 - ext(('0' & b1),9);

process(clk,rst)
begin
if rst = '1' then
sm <= (others => '0');
elsif clk = '1' and clk'event then
          sm <= sm2;
          end if;
end process;

a2 <= sm(7 downto 0);
d2<=conv_std_logic_vector(rom0(conv_integer(a2)),8
);
process(clk,rst)
```

```
begin
if rst = '1' then
z0 <= '0';
elsif clk = '1' and clk'event then
        if a = x"00" or b = "00" then
                z0 <= '1';
        else
                z0 <= '0';
        end if;
        end if;
        end process;
res <=  x"00" when (z0 = '1')  else  d2;
```

Здесь rom1 – двухпортовое ПЗУ таблицы логарифмов, а rom0 – ПЗУ антилогарифмов.

Параметризированный векторный умножитель имеет вид:

```
U_md: for i in 0 to G_range - 1 generate
mul : mul_g8
 port map(
        clk => clk,
rst => rst,
        m_d => m_d,
         a => a(i),
 b => b(i),
         res => c(i)
            );
 end generate;
```

Структурно декодер разбит на три блока, которые выполняют (см. рис.1):

– прием кадра и нахождение синдрома ошибки.

– определение характера ошибки и формирование корректирующей маски;

– коррекцию ошибки.

Прием кадра длиной *N* слов осуществляется в сдвиговый регистр. В дальнейшем эти данные участвуют в формировании синдрома ошибок, который вычисляется последовательно на одном умножителе и накапливающем сумматоре. Результатом является синдром ошибок, на основании которого находится полином ошибки.

Полином ошибки вычисляется с помощью алгоритма Берлекэмпа-Месси. Для этого используется описанный выше векторный умножитель. После этого на том же векторном умножителе реализуются алгоритм Ченя – для поиска корней данного полинома и алгоритм Форни – для формирования корректирующей маски.

В оконечном блоке принятый кадр пословно корректируется путем суммирования с корректирующей маской.

Все модули работают в старт-стопном режиме (последующий модуль начинает работать по сигналу окончания работы предыдущего модуля). Такая реализация позволяет организовать конвейерную работу декодера, что может улучшить временную характеристику системы.

Модуль параметризирован для операций с байтовыми последовательностями (длина слова – 8 бит). В качестве параметров выступает количество исправляемых ошибок *t* (2 – 8 ошибок), длина кадра *N* (до 255 слов), порождающий полином.

Технические характеристики декодера

Основной характеристикой декодера является время выполнения операции декодирования. Поскольку аналитически представление довольно сложное, покажем некоторые данные полученные экспериментальным путем.

В табл.1 указано количество тактов, необходимых для выполнения декодирования, начиная от сигнала начала последовательности, заканчивая, сигналом окончания работы, т.е. время выполнения операции декодирования при различных параметрах. При этом возможность конвейерных вычислений не учитывается.

Табл. 1. Число тактов декодирования

t	*Длина кадра* **N,** *слов*			
	32	**64**	**128**	**255**
2	472	632	1012	1647
3	618	842	1290	2179
4	704	992	1568	2711
5	790	1142	1846	3243
6	876	1292	2124	3775
7	962	1442	2402	4307
8	1048	1592	2680	4839

Существенный рост времени декодирования при увеличении количества исправляемых ошибок связан с тем, что при последовательном нахождении синдрома ошибки здесь используется только один умножитель. Можно ускорить эту операцию, используя векторный умножитель, за счет пропорционального роста аппаратных затрат.

В табл. 2 указаны аппаратные затраты на декодер, параметризованный для различного числа исправляемых слов и длины пакета при его конфигурировании в ПЛИС Xilinx серий Virtex-4 и Spartan-3E. При этом в ячейках указано количество блоков памяти (BRAM) и количество конфигурируемых логических блоков (CLB slices). Данные получены при синтезе проекта декодера в среде ISE 9.2.

Табл. 2. Аппаратные затраты декодера

t	*Длина кадра* **N**, *слов*			
	32	**64**	**128**	**255**
2	8 1324	8 1399	8 1417	8 1567
3	11 1879	11 1882	11 1888	11 1998
4	14 2389	14 2546	14 2638	14 2755
5	17 3052	17 3086	17 3127	17 3196
6	20 3667	20 3714	20 3741	20 3941
7	23 4163	23 4204	23 4252	23 4327
8	26 4425	26 4663	26 4697	26 4712

Максимальная тактовая частота, с которой работает декодер, определяет как его пропускную способность, так и возможности его внедрения в различных проектах СНК. В табл. 3 показана максимальная тактовая частота работы в различных ПЛИС фирмы Virtex.

Табл. 3. Максимальная тактовая частота декодера, МГц

Серия ПЛИС	***N* = 32, *t*=2**	***N* =255, *t* = 8**
Spartan 3 xc3s1500-5	74	66
Spartan 3E xc3s1200e-5	91	72
Spartan 3A xc3sd1800a-4	80	67
Virtex4 xc4vlx40-12	149	126
Virtex5 xc5vlx50-3	184	143

Как по тактовой частоте, так и по аппаратным затратам синтезированный декодер соответствует лучшим мировым образцам декодеров Рида-Соломона, реализованным в ПЛИС.

Вывод

Разработанный декодер Рида-Соломона, благодаря учету структурных свойств современных ПЛИС, реализуется в них с высокой тактовой частотой при умеренных аппаратных затратах, которые сравнимы с параметрами лучших мировых образцов.

Вместе с тем, есть возможности существенного улучшения декодера по увеличению быстродействия. Для этого нужно распараллелить процесс нахождения синдрома ошибки, применяя векторный умножитель в поле Галуа. Также целесообразно увеличить тактовую частоту работы декодера при реализации декодера в сериях ПЛИС Xilinx Spartan 3 и Altera Cyclone, которые .отличаются высоким отношением эффективность-цена.

Список литературы

1. Вернер М. Основы кодирования. –М.: Техносфера. –2004. –288с.
2. Касперски К. Могущество кодов Рида-Соломона или информация, воскресшая из пепла // Системный администратор. –2004. –с.88-94.
3. Блейхут Р. Теория и практика кодов, контролирующих ошибки. –М.:Мир. –1986. –566с.

Поступила в редакцию 17.12.2009