№ 5 1994

УДК 681.325.3 : 519.688

И. И. Коршевер, П. А. Полозков

(Новосибирск)

ЭФФЕКТИВНЫЕ АЛГОРИТМЫ И ПРОГРАММЫ КОДИРОВАНИЯ И ДЕКОДИРОВАНИЯ КОДОВ РИДА — СОЛОМОНА ДЛЯ УНИВЕРСАЛЬНОГО МИКРОПРОЦЕССОРА TMS320C30/31

Предложены эффективные способы оптимизации программного кодирования и декодирования перемежающихся кодов Рида — Соломона (Р—С-кодов) над полем Галуа GF(256), в частности, показано, что и при кодировании, и при вычислении синдрома ошибок для быстрого нахождения коэффициентов частотного разложения векторов над GF(256) эффективно используется алгоритм Герцеля, содержащий только операции «Исключающее ИЛИ». Предложен также ускоренный алгоритм разложения полинома локаторов ошибок (ПЛО) и способы более достоверного обнаружения неисправляемого числа ошибок. Приведены и сравнены между собой по быстродействию результаты моделирования разных алгоритмов вычисления синдрома ошибок и разложения ПЛО на процессоре обработки сигналов TMS320C30.

Введение. Широкое развитие систем телекоммуникаций и устройств сверхплотной записи повысило интерес к методам линейного блокового кодирования [1, 2] и средствам реализации этих методов. Наибольшее внимание в литературе уделено кодам Рида — Соломона (далее Р—С-кодам) — линейным блоковым кодам (из множества линейных циклических блоковых кодов БЧХ — Боуза — Чоудхури — Хоквингема), обладающим рядом оптимальных свойств и прозрачной весовой структурой, и архитектурным аспектам реализации на СБИС кодеров/декодеров для них (далее Р—С-кодеков) [3—6]. Гораздо меньшее внимание уделяется микропрограммной реализации этих устройств [7—10]. Между тем использование стандартных микропроцессоров позволило бы перестраивать кодек на меняющиеся от применения к применению параметры кода, и, кроме того, всегда экономически предпочтительнее разработки специализированных БИС с фиксированными параметрами.

В работе [9] Р—С-кодек построен на основе многопроцессорной реализации, базирующейся на 8-разрядном микропроцессоре серии 1818ВМ01 (тактовая частота 4 МГц). Кодек обрабатывал определенные стандартом для магнитооптических дисков (1200, 1040) данные, поступающие со скоростью 375—446 Кбайт/с. Далее эта работа была продолжена в направлении использования однокристального быстродействующего 32-разрядного сигнального процессора TMS320C30/31 [11], работающего на тактовой частоте 16 МГц. Адаптация алгоритмов кодирования/декодирования Рида — Соломона к архитектурным особенностям кристалла TMS320C30/31 потребовала критического анализа всей последовательности процесса кодирования/декодирования и открыла новые возможности для оптимизации программ и сокращения времени обработки.

В работе проанализированы основные этапы кодирования/декодирования Р—С-кодов с точки зрения их микропрограммной реализации и предложен ряд эффективных способов обработки на разных этапах этих преобразований. При

этом кодирование и вычисление синдрома — наиболее трудоемкий этап декодирования — используют одну и ту же процедуру быстрого нахождения коэффициентов частотного разложения в полиномиальном кольце Галуа GF(256), осуществляемую над передаваемым вектором данных (при передаче) и над принятым кодом с ошибками (при приеме). Предложены также ускоренный метод нахождения корней полинома локаторов ошибок (ПЛО) и способы более достоверного обнаружения неисправляемой конфигурации ошибок.

Приведены и сравнены между собой по быстродействию результаты моделирования разных алгоритмов вычисления синдрома и определения последовательности ПЛО на сигнальном процессоре TMS320C30/31. В Приложении сведены некоторые программы из последовательности кодирования/декодирования Р—С-кодов, написанные на Ассемблере этого кристалла.

1. Алгоритм обнаружения и исправления ошибок, используемый в [9]. В данной работе будут широко употребляться используемые в [9] обозначения: (n, k) Р—С-код — блочная последовательность m-битовых символов в поле Галуа GF(2^m). Эта последовательность может быть рассмотрена как набор коэффициентов полинома

$$C(x) = c_0 + c_1 + c_2x^2 + \ldots + c_{n-1}x^{n-1}, \tag{1}$$

где каждый символ — элемент поля Галуа: $c \subset \mathrm{GF}(2^m)$ — конечное поле из 2^m двоичных символов.

Параметры (n, k) Р—С-кода имеют следующее содержание (рис. 1): m — количество бит в символе; $n = 2^m - 1$ — длина кодового слова; k — количество информационных символов в кодовом слове; $d = n - k + 1$ — конструктивное межкодовое расстояние; $t = (d - 1)/2$ — количество исправляемых кодом символов с ошибками.

Для кода, исправляющего ошибки, задается порождающий полином

$$G(x) = \sum_{p=0}^{d-1} g_p x^p = (x - a)(x - a^2) \ldots (x - a^{d-1}) =$$

$$= G(x) = G_1(x)G_2(x) \ldots G_{d-1}(x), \tag{2}$$

где a — примитивный элемент из $G(2^m)$, а $G_p(x) = x - a^p$ — минимальный полином от a^p $(p = 1, 2, \ldots, d - 1)$.

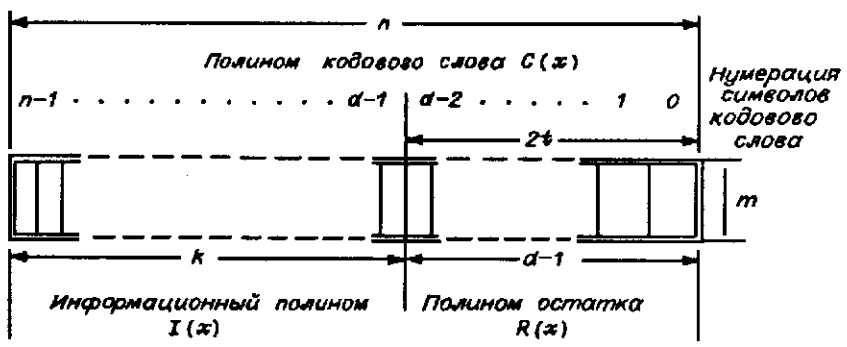

Рис. 1. Кодовое слово Р—С-кода

Пусть

$$I(x) = i_{d-1} + i_d x + \ldots + i_{n-1} x^{n-d} \tag{3}$$

— исходный информационный вектор. Процесс кодирования заключается в том, что кодовое слово $C(x)$ на входе канала передачи данных представляется следующим образом:

$$C(x) = (I(x)x^{d-1}) + R(x) = c_0 + c_1 x + \ldots + c_{n-1} x^{n-1}, \tag{4}$$

где $R(x) = (I(x)x^{d-1}) \bmod G(x)$ есть остаток от деления полинома $(I(x)x^{n-k})$ на $G(x)$.

Пусть далее помехи таковы, что принимаемый на выходе канала вектор

$$V(x) = C(x) + e(x) = v_0 + v_1 x + \ldots + v x^{n-1}. \tag{5}$$

Тогда процесс определения $e(x)$ и вычитания его из принятого вектора и будет декодированием систематического кода.

Основные этапы преобразований, осуществляемых в Р—С-декодере, приведены на рис. 2:

1. Нахождение синдрома ошибок $S(x)$ по рекурсивной схеме Горнера, требующее $n(d-1)$ умножений/сложений.

2. Вычисление полинома локаторов ошибок $A(x)$ по алгоритму Берлекампа — Месси (БМ), требующее в случае программной реализации $1{,}5t^2$ умножений/сложений.

3. Нахождение корней ПЛО $A(x)$ методом Ченя для определения полинома позиций ошибок $X(x)$. Оно требует nt умножений/сложений.

4. Вычисление принятого с кодом вектора ошибок $E(x)$ по алгоритму Форни [1] и восстановление кода $C(x) = V(x) - E(x)$. В общей сложности это требует примерно $2t^2$ умножений/сложений.

Для программной реализации исправляющего ошибки кодека на универсальных устройствах арифметической обработки, не содержащих умножителей в пространстве полиномиальных или числовых колец, существует множество методов, в основе которых лежит алгоритм Питерсона — Горенстейна — Цирлера для кодов БЧХ [1]. Они подробно описаны в [2] и [9], где дается оценка их вычислительной сложности. Так, общепринятый алгоритм кодирования путем полиномиального деления требует $(n-d-1)(d-1)$ умножений/сложений. Декодер требует в общей сложности $(n-1)(d-1) + 7t^2/2 + (n-1)t$ умножений и $n(d-1) + nt + 2t^2$ сложений при количестве ошибок, равном t.

Арифметические операции на всех этапах преобразования производятся в поле полиномиальных колец. В частном случае, когда вычисления сведены в поле GF(2^m), базовая операция суммирования осуществляется по модулю 2 (без арифметических переносов) — однотактная операция, предусмотренная в любом стандартном арифметико-логическом устройстве (в том числе и в процессоре TMS320C30/31). По этой причине такое сведение является глав-

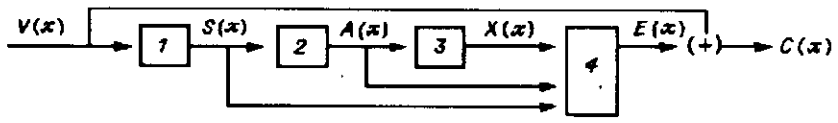

Рис. 2. Основные этапы преобразований, осуществляемых в Р—С-декодере

ной целью большинства методов реализации операций на группах [1]. Более того, 32-разрядное арифметическое устройство и ассоциированные с ним регистры-аккумуляторы позволяют упаковывать по четыре 8-разрядных числа в поле GF в одном 32-разрядном регистре и далее производить операцию над таким 32-разрядным блоком одновременно. Что касается умножения, то оно заменяется сложением логарифмов в полях Галуа с последующим потенцированием результата (логарифмирование и потенцирование на GF осуществляется таблично). Естественно, как и в классической конструктивной математике, сокращение количества умножений здесь остается одной из главных целей общей оптимизации.

Используя все эти приемы, без учета затрат на загрузку/выгрузку, удается выполнить умножение за шесть-семь команд, а сложение — за одну команду. В Приложении А приведена программа, выполняющая операции сложения и умножения чисел в поле GF.

Программы декодирования имеют почти вдвое большую вычислительную сложность, чем программы кодирования, а в них наибольшую сложность представляют операции нахождения синдрома ошибок и корней ПЛО (блоки *1* и *3* на рис. 2). В связи с этим прежде всего приведем и сравним возможные способы их оптимизации.

2. Способы ускорения программного вычисления синдрома ошибок S(x). Наибольшая (по числу умножений) часть реализуемых в декодере вычислительных операций принадлежит определению синдрома ошибок:

$$S(x) = \sum_{i=0}^{d-1} s_i x^i, \tag{6}$$

где $s_i = V(x) = E(x)$ при $x = a^i$, соответствующем корням $G(x)$.

Фрагмент программы рекурсивного вычисления $S(x)$ по схеме Горнера, написанной на Ассемблере TMS320C30/31, приведен в Приложении Б. Для вычисления $S(x)$ в этом случае необходимо $6 \times 254(d-1)$ тактов процессора (по шесть команд на цикл).

Рассмотрим известные способы быстрого вычисления $S(x)$ в полиномиальном кольце и оценим требуемое время на их выполнение применительно к процессору TMS320C30/31.

А. *Вычисление синдромов с помощью фильтра с бесконечной импульсной характеристикой.* Синдромы можно вычислить путем прямого деления вектора считываемых данных на порождающий полином [1]:

$$S(x) = (V(x)G(x)) + R(x), \tag{7}$$

где $R(x) = V(x) \bmod G(x)$ — остаток, степень которого не превышает $d-2$, из которого затем можно вычислить компоненты $S(x)$ на том основании, что

$$E(x) = V(x) = Q(x)G(x) + R(x) = R(x) \quad \text{при} \quad x = a^0, a^1, \ldots, a^{d-2}, \tag{8}$$

где $Q(x)$ — частное от деления $V(x)$ на $G(x)$, а $R(x)$ — остаток. Коэффициенты $r_0 \ldots r_{d-2}$ текущего остатка $R(x)$ на j-м шаге:

$$r_p^{(j)} = r_{p-1}^{(j-1)} + (\mathrm{v}_{n-j} + r_{d-2}^{(j-1)}) g_{p-1}, \tag{9}$$

где $n = 2^m - 2$; g — коэффициенты $G(x)$; v — компоненты $V(x)$; $j = 0, 1, 2, \ldots$ $\ldots, n - 2t$; $p = 0, 1, 2, \ldots, d-2$.

В [1] эту процедуру, аналогичную процедуре кодирования информационного вектора $I(x)$, предложено реализовать с помощью фильтра с бесконечной импульсной характеристикой (БИХ-фильтра).

В случае, когда полином $G(x)$ симметричен и, следовательно, симметричные коэффициенты равны ($g_p = g_{N-p}$), число повторений в программе может быть уменьшено вдвое благодаря использованию один раз вычисленного произведения $g_p x^p$ дважды в одном цикле. При таком выборе порождающего

полинома (как это сделано, например, в [10]) число тактов для вычисления $S(x)$ удается уменьшить до $40(254 - 16) = 9520$ (без учета расходов на непосредственное вычисление $S(x)$ из полученного остатка) или почти вдвое по сравнению со схемой Горнера.

В Приложении В приведены фрагменты программ реализации метода БИХ-фильтра на сигнальном процессоре для случая, когда коэффициенты полинома заданы в общем виде, и для случая, когда для сокращения программы используется их центральная симметрия.

Выражение (6) для вычисления синдрома ошибок может быть представлено в виде

$$S_j = \sum_{i=0}^{n-1} v_i a^{ij}, \qquad (10)$$

где a — корень порождающего полинома. В таком виде выражение (10) может быть интерпретировано и вычислено как фурье-преобразование над вектором входных данных, а его компоненты — как элементы частотного разложения входного вектора $V(x)$ в полиномиальном кольце Галуа GF(256). Далее рассматриваются два метода, использующие спектральное представление для применения к технике вычисления синдромов идей ускоренного преобразования Фурье: модифицированный алгоритм Винограда и «полубыстрый» алгоритм Герцеля.

Б. *Схема Винограда.* Для быстрой реализации преобразования (6) в [7] предлагается использовать китайскую теорему об остатках и модифицированный («гнездовой») алгоритм Винограда (МАВ). Факторизовав $n = 255 = 17 \times 5 \times 3$, построим на базе полученных простых сомножителей числовую систему остаточных классов. Порядки элемента a в (6) исчисляются в этой системе, будучи предварительно переиндексированы по Гуду — Томасу. Далее можно все вычисления $S(x)$ на GF(256) разбить на три итерации, каждая из которых есть последовательность преобразований Фурье (далее ПФ) на поле GF над короткими векторами:

1. 17×5 раз вычислить 3–точечное ПФ, содержащее одно умножение и пять сложений;
2. 17×3 раз вычислить 5–точечное ПФ, содержащее пять умножений и 17 сложений;

} 340 умножений + + 1292 сложения.

Для приближенной оценки временных затрат на первых двух итерациях в Приложении Г приведен алгоритм вычисления 5-точечного ПФ. Общие временные затраты на первых двух итерациях составляют не менее $(340 \times 6{,}5 + 1292) = 3502$ тактов.

3. Далее путем непосредственной подстановки необходимо вычислить 17-точечное ПФ над 15 полиномами; результатом итерации являются группы по $d - 1$ компонент $S(x)$. Для осуществления этого этапа необходимо выполнить $(d - 1)15$ умножений и $(d - 1)16$ сложений. При непосредственном вычислении $S(x)$ каждая операция типа $xy + z$ занимает пять тактов процессора, а вся последняя итерация — по меньшей мере $15(d - 1)5$ тактов. Например, при $d = 17$ она потребует 1200 тактов.

Таким образом, минимально оцененная сумма тактов на вычисление $S(x)$ с применением МАВ без учета затрат на переиндексацию Гуда — Томаса требует $3502 + 1200 = 4700$ тактов процессора при $d = 17$, что вдвое меньше, чем для полиномиального деления $V(x)$ на $G(x)$ со специально подобранными коэффициентами.

В. *Алгоритм Герцеля.* В [1] для вычисления синдромов путем преобразования Фурье в полиномиальном кольце предложен так называемый «полубыстрый» алгоритм Герцеля, аналогичный известному в теории сигналов одноименному алгоритму вычисления фурье-коэффициентов. Алгоритм Герцеля схож с рекурсивной схемой Горнера, отличие лишь в том, что делящие полино-

$$(x - b^p), \quad \text{где} \quad p = 1, 2, ..., N - 1, \tag{11а}$$

если b — элемент из поля $GF(2^N)$ порядка N, т. е. выполняется свойство

$$b^p = b^{p+Nj} \quad \text{для любых } p \text{ и } j. \tag{11б}$$

Ряд элементов b^p, $p = 1, 2, ..., N - 1$, называется классом сопряженных элементов в GF. Поскольку они определены в двоичном поле, то при выполнении над ними арифметических операций умножение отсутствует, а сложение упрощено до поразрядных операций по модулю 2. Более того, по этим причинам упакованные по четыре 8-разрядных символа 32-разрядные слова для выполнения арифметических операций нет необходимости распаковывать, благодаря чему остатки от деления $V(x)$ для четырех Р—С-кодовых слов вычисляются одновременно. При этом 32-разрядный формат процессора TMS320C30/31 используется наиболее эффективно.

Все элементы поля $GF(2^N)$ можно разделить на $2/N$ классов по N сопряженных элементов в каждом. Например, в поле GF(256), порождаемом неприводимым полиномом восьмой степени на GF(2):

$$p(x) = x^0 + x^2 + x^3 + x^5 + x^8,$$

моделированием найдено 34 класса сопряженных элементов. Некоторые из них содержат по два и по четыре элемента. На рис. 3 изображена блок-схема деления $V(x)$ на один из минимальных полиномов:

$$1 + x + x^4, \quad \text{имеющий корни } a^{17},\ a^{34},\ a^{68} \text{ и } a^{136}.$$

Ниже приводится фрагмент программы, выполняющей такое деление.

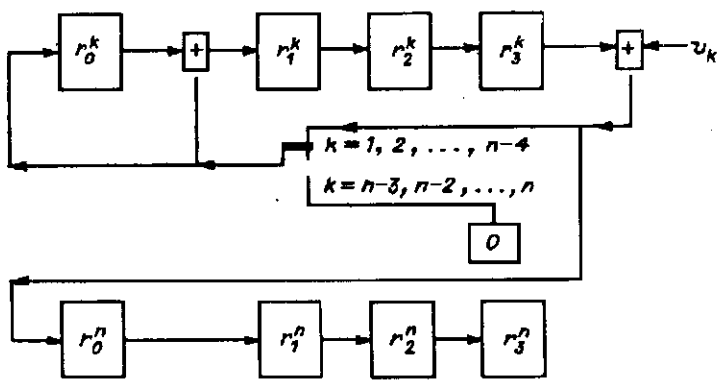

Рис. 3. Блок-схема алгоритма деления $V(x)$ на двоичный полином $1 + x + x^4$

Для итераций $p = 1, 2, ..., n/4$:

```
; ar0 → V(x);
    . . .
      RPTB    L1                ;  повторяется 256/4 раз...

      xor     *ar0 ++ (1),r0    ;  r0 + *ar0 ⇒ r1 ⇒ r0
      xor     r0, r1            ;  r0 + r1 ⇒ r2 ⇒ r1

      xor     *ar0 ++ (1),r3    ;  , r0 + r1  r2  r3 → (+) ← *ar0++
      xor     r3, r0            ;
                                      ↑_____________↓

      xor     *ar0 ++ (1),r2    ;  , r3 + r0  r1  r2 → (+) ← *ar0++
      xor     r2, r3            ;
                                      ↑_____________↓

      xor     *ar0 ++ (1),r1    ;  , r2 + r3  r0  r1 → (+) ← *ar0++
                                      ↑_____________↓
L1    xor     r1, r2            ;
    . . .
```

Выберем, например, 16 подряд лежащих по степеням a элементов GF(256), разделим их на классы и выпишем количество отличных от нуля коэффициентов в двоичных полиномах для каждого класса и соответствующее число команд (тактов) в цикле программы:

a^1, a^2, a^4, a^8, a^{16}	(5 коэффициентов --- 4 команды в цикле);
a^3, a^6, a^{12}	(5 коэффициентов --- 4 команды в цикле);
a^5, a^{10}	(5 коэффициентов --- 4 команды в цикле);
a^7, a^{14}	(5 коэффициентов --- 4 команды в цикле);
a^9	(7 коэффициентов --- 6 команд в цикле);
a^{11}	(5 коэффициентов --- 4 команды в цикле);
a^{13}	(7 коэффициентов --- 6 команд в цикле);
a^{15}	(5 коэффициентов --- 4 команды в цикле);

Поскольку полином для класса a^{17} имеет всего три отличных от нуля коэффициента (две команды в цикле), то, включив его в программу, можно достичь выигрыша в отношении избыточность кода/время вычисления. Избыточность увеличится с 16 до 18, так как a^{18} входит в класс a^9. Для более достоверного обнаружения неисправляемого числа ошибок в процедуру включается также дополнительный нечетный 19-й синдром $s(a^0)$, для вычисления которого достаточно сложить все символы $V(x)$ по модулю 2 (одна команда на цикл). Всего, без учета вычисления синдрома ошибок из полученных остатков, для одновременного вычисления остатков от деления четырех кодовых слов $V(x)$ понадобится $(1 + 4 \times 6 + 6 \times 2 + 2)255 = 9945$ тактов или 2486 тактов для одного вектора $V(x)$ при $d = 20$.

Дополнительное время на вычисление компонент синдрома и распаковку остатков занимает всего около 30 % времени работы всей процедуры.

Таким образом, при $d = 20$ с использованием полной разрядной сетки сигнального процессора «полубыстрый» алгоритм Герцеля работает вдвое быстрее метода МАВ. Блок-схема предлагаемого алгоритма быстрого вычисления частотных компонент векторов над GF(256) с применением алгоритма Герцеля показана на рис. 4. На рис. 5 приведены сравнительные характеристики метода МАВ и предлагаемого алгоритма (по горизонтальной оси — избыточность декодируемого Р—С-кода, по вертикальной — число тактов сигнального процессора, затрачиваемых на вычисление спектральных компонент).

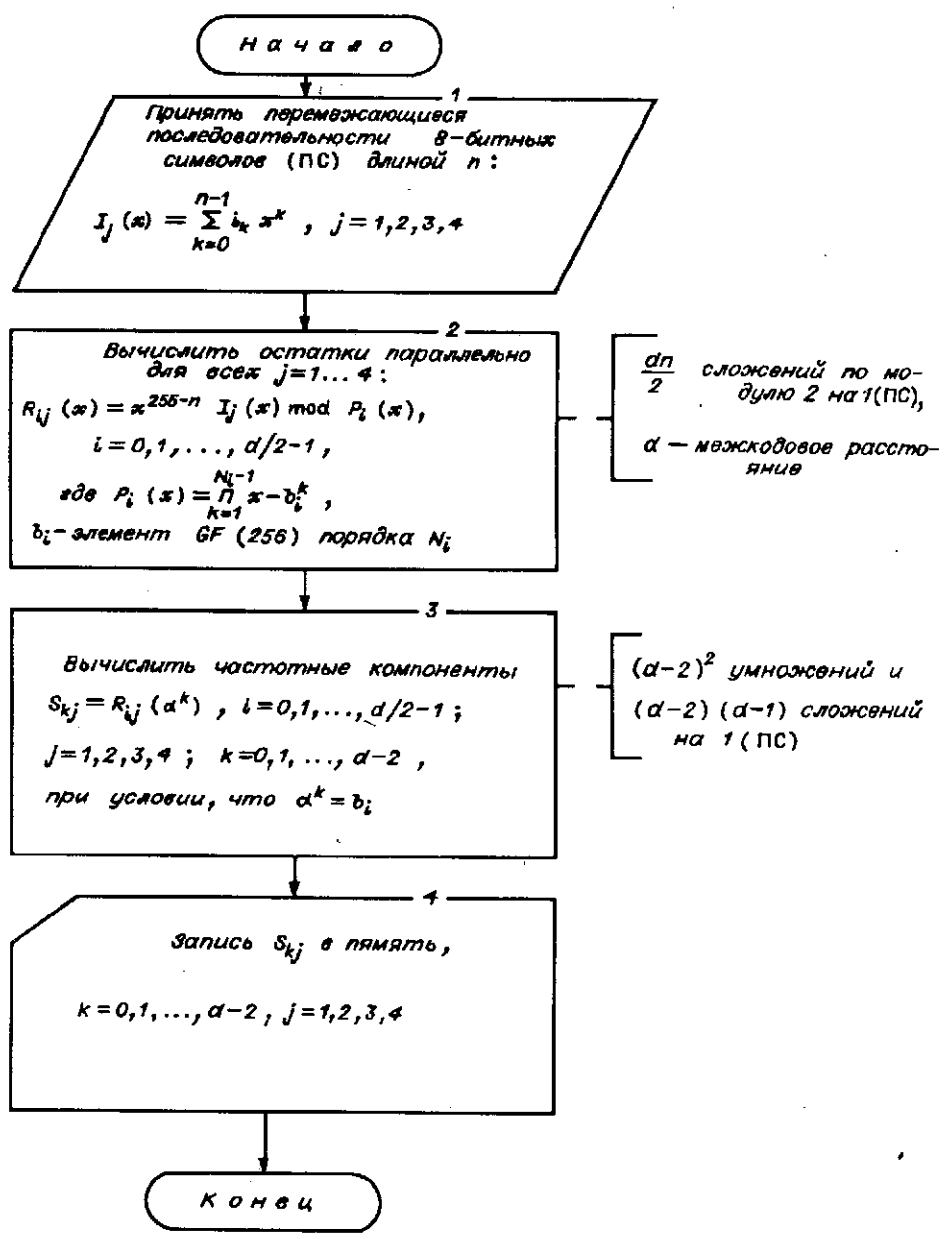

Рис. 4. Блок-схема быстрого вычисления частотных компонент векторов над GF(256) с применением алгоритма Герцеля

3. Оптимизация вычисления полинома локаторов ошибок. При использовании алгоритма Герцеля описанным выше способом вычисление $S(x)$ занимает примерно 1/10 часть общего времени, затрачиваемого на обнаружение и исправление ошибок. Следующей по трудоемкости (при числе ошибок, близком к максимально исправляемому) следует считать процедуру нахождения корней полинома локаторов ошибок [9] (блок *3* на рис. 2 реализует алгоритм Ченя [1]), полученного в результате выполнения процедуры Берлекампа — Месси для нахождения полиномов локатора ошибок и их значений (блок *2* на рис. 2).

В [8] для ускорения процедуры нахождения корней ПЛО при исправлении в $V(x)$ ошибок и отмеченных стираний для (255, 223) Р—С-кода предлагается так же, как и для нахождения $S(x)$ в [7], воспользоваться рассмотренным выше

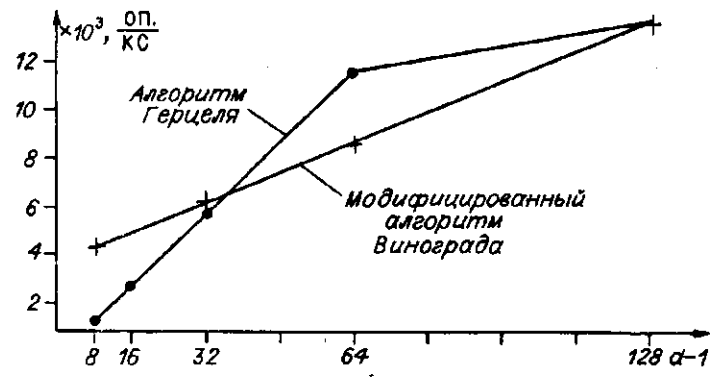

Рис. 5. Сравнительные характеристики метода МАВ и предлагаемого алгоритма

МАВ. По приведенной в [8] оценке число умножений для указанного случая равно 1135, а число сложений — 4465 против $254(s + t)$ умножений и $254(s + t)$ сложений для процедуры Ченя, где s — число обозначенных приемником стираний. Из сопоставления этих величин видно, что этот способ уступает процедуре Ченя даже по числу умножений, если в принятом кодовом слове обнаружено не более четырех ошибок и стираний, в то время как вероятность появления блоков с меньшим числом ошибок гораздо выше, чем с большим.

Авторами предлагается алгоритм нахождения корней ПЛО, в основе которого лежит процедура Ченя, модифицируемая тремя способами:

1. После обнаружения очередного корня b полином ПЛО сокращается на $(x - b)$. Схема сокращения строится на основе БИХ-фильтра, дополненного регистрами для сохранения компонент частного.

2. При уменьшении степени ПЛО до или ниже четырех используется улучшенный в [3] прямой способ для нахождения корней ПЛО степени не выше четырех.

3. Учитывая преобладающий пакетный характер ошибок, введен массив подсказок наиболее вероятных позиций ошибок.

Пусть в принятом блоке данных, состоящем из T перемежающихся кодовых слов:

$$V_1(x), V_2(x), ..., V_T(x),$$

оказалось p пакетов ошибок и пусть для $V_1(x)$ достоверно определены N ошибок в позициях

$$i_1^{(1)}, i_2^{(1)}, i_3^{(1)}, ..., i_N^{(1)}.$$

Тогда в остальных кодовых словах $V_2(x) ... V_T(x)$ позиции ошибок совпадут с найденными для $V_1(x)$, по крайней мере, в $(N - p)$ случаях. При этом в $2p$ случаях могут быть ошибки в других позициях. Для ускоренного разложения ПЛО, имеющего степень выше четырех, нужно сначала подставить и проверить корни, найденные для смежных ПЛО, используя их в качестве подсказок и сокращая таким образом степень ПЛО до $2p$ или ниже, а затем продолжить поиск корней по п. 1 и 2.

Нетрудно убедиться в том, что при равновероятном распределении пакетов ошибок по всей длине кодовых слов каждый из перечисленных приемов дает ускорение в 2 раза, а в целом предложенная модификация работает в 8 раз быстрее обычной процедуры Ченя.

Предлагаемый алгоритм ускоренного нахождения корней ПЛО приведен на рис. 6.

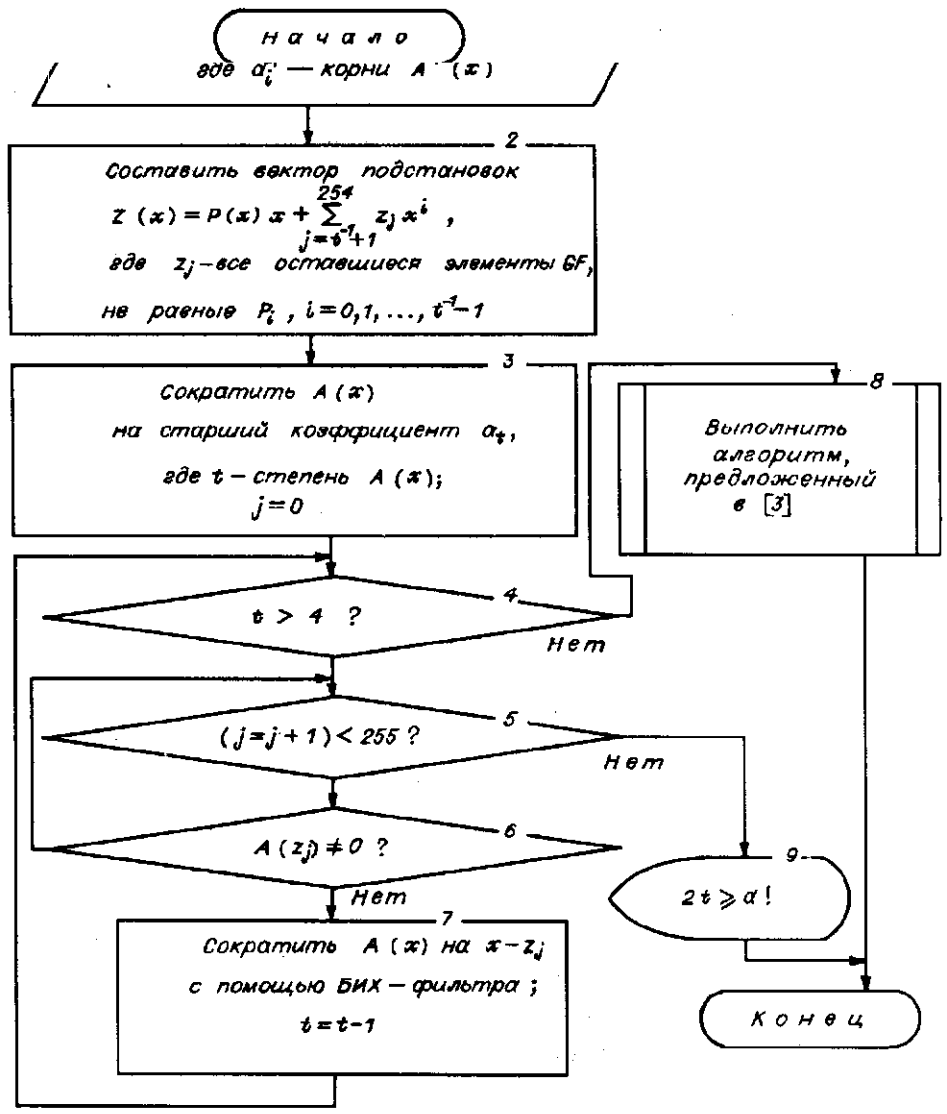

Рис. 6. Блок-схема алгоритма ускоренного нахождения корней ПЛО $A(x)$

Для вычисления самого ПЛО $A(x)$ (блок *2* на рис. 2) в программе декодера был использован описанный в [1] алгоритм Берлекампа — Месси, рекуррентным продолжением которого также вычисляется общий полином ошибок $W(x)$, из которого затем по алгоритму Форни (блок *4* на рис. 2) вычисляется принятый вектор ошибок $E(x)$. Число тактов на выполнение частей *2* и *4* (см. рис. 2) при самой грубой оценке равно $6 \times 2t^2$ (по 6 тактов на операцию $AB + C$) или при максимальном $t = 9$ для $d = 19$ (разд. 2В) требует ориентировочно 972 такта на каждое кодовое слово, что существенно меньше временных затрат на выполнение рассмотренных выше двух этапов.

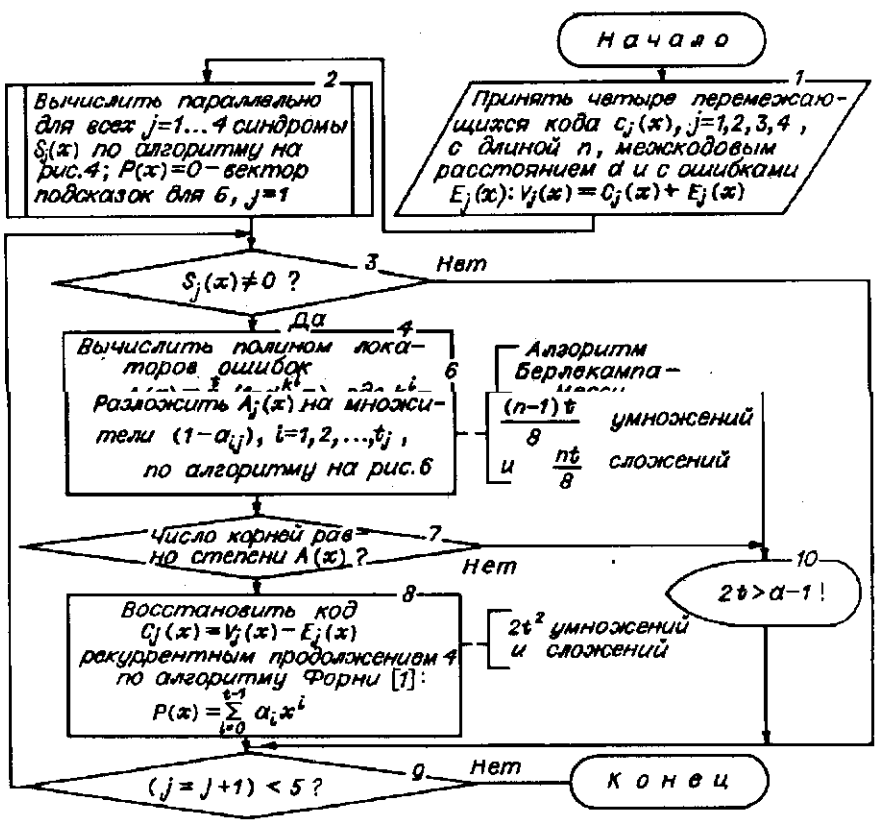

Рис. 7. Блок-схема программного декодирования с использованием предлагаемых алгоритмов

4. Обнаружение неисправляемой конфигурации ошибок. Для обнаружения неисправляемой конфигурации ошибок в коде до начала поиска корней ПЛО в процедуру вычисления синдрома введена дополнительная (нечетная) итерация, использующая его дополнительную 19-ю компоненту:

$$s_{18} = V(a^{18}),$$

и, если для найденных ПЛО и $S(x)$ не выполняется условие

$$\sum_{j=1}^{19} (A_{j-1} s_{19-j}) = 0,$$

где A_{j-1} — коэффициент ПЛО при x^{j-1}, то программа отказывается от дальнейшего декодирования.

В результате моделирования с применением генератора псевдослучайных чисел, используемого для внесения в принятый код $V(x)$ случайного количества ошибок со случайным распределением и случайными величинами, выяснилось, что если количество ошибок больше 19, то описанный метод не всегда обнаруживает неисправляемый вектор данных. В таком случае неис-

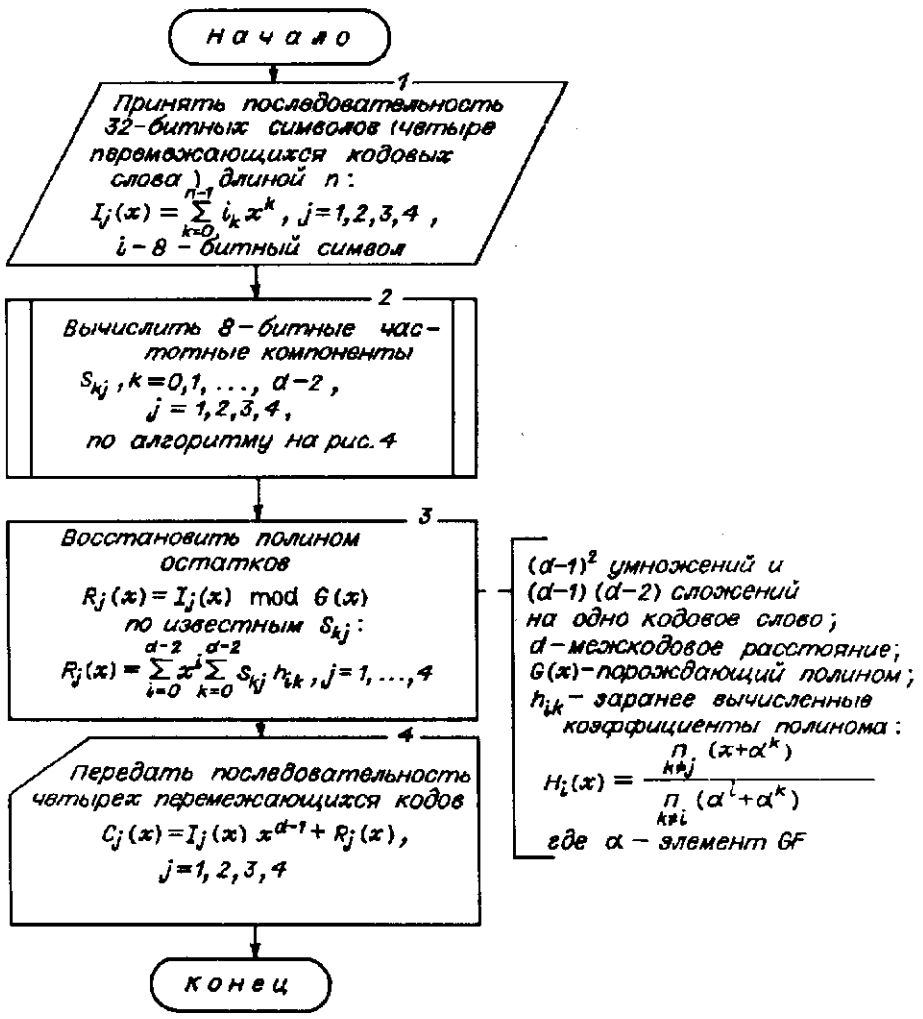

Рис. 8. Блок-схема алгоритма программного кодирования с использованием процедуры быстрого вычисления частотных компонент векторов над GF(256)

правляемые ошибки нетрудно обнаружить после нахождения корней ПЛО. Теорема Берлекампа [1] доказана только для $2t < d$ при отсутствии стираний. И если количество найденных корней не совпадает со степенью исходного ПЛО, то программа также отказывается от неправильного исправления ошибок. Общая блок-схема процесса программного декодирования с использованием предлагаемых алгоритмов изображена на рис. 7.

В основу программного кодера положена та же процедура быстрого вычисления частотных компонент $S(x)$, что и в декодере. Это позволило существенно сократить объем программ кодека и до возможного предела ускорить процесс кодирования. Получив таким образом 19 частотных компонент $s_p = I(a^p)$, $p = 0, 1, ..., 18$, для информационного вектора

$$I(x) = \sum_{j=0}^{254} (i_j x^j),$$

где $i_j = 0$ при $j = 0, 1, 2, ..., 18$, воспользуемся предложенным в [2] и [5] методом систематического кодирования с помощью интерполяционных многочленов Лагранжа, чтобы найти полином остатка

$$R(x) = I(x) \bmod G(x)$$

по его известным значениям s_p в точках a^p.

Полный алгоритм программного кодирования приведен на рис. 8.

5. **Общая оценка быстродействия РС-кодека на TMS320C30/31.** По предложенным выше алгоритмам были написаны программы для Р—С-кодека на языке Ассемблера для TMS320C30/31 и измерены скорости их работы.

Для одновременного кодирования четырех блоковых Р—С-кодов на GF(256) с параметрами (255, 236, 19) затрачивается 1,21 мс, что обеспечивает неразрывность потока данных на выходе кодера 821 Кбайт/с.

Одновременное декодирование четырех кодовых слов без использования п. 2 в разд. 3 выполняется за 0,908 мс при отсутствии ошибок и за 1,46 мс при наличии 36 испорченных байт, сгруппированных в один пакет. Такой декодер справляется с потоком в канале 821 Кбайт/с при исходной вероятности не хуже 1/10 ошибок на кодовое слово длиной 255 байт. Такой кодек может быть, в частности, применен в магнитооптических накопителях: его производительность более чем вдвое превышает производительность кодека, предложенного в [9].

ПРИЛОЖЕНИЕ А

Реализация арифметических операций в поле чисел GF. Ниже приведены программы сложения и умножения чисел в поле GF. В обоих случаях центральный оператор один и тот же — сложение. Если операнды представлены как натуральные элементы из множества GF, то результат — сумма двух операндов ($x + y$ в примере), также принадлежащая множеству GF. Если операнды — логарифмы чисел из GF, то результат — логарифм произведения (xy в примере), также принадлежащего к GF. Если операнды умножения — ранее вычисленные суммы, они должны быть предварительно прологарифмированы (как в примере), если же операнды сложения — произведения ранее выполненных умножений, они должны быть пропотенцированы. Эти преобразования осуществляются в сигнальном процессоре с помощью таблицы логарифмов/антилогарифмов чисел в поле GF.

```
;   ir1 → таблица логарифмирования,
;   ir0 → таблица потенцирования;
;   r1 = x;     r2 = y;
;   x + y = ?
     . . .
    addi3       r1, r2, r3                  ;   r3 = x + y.
    . . .
;   r1 = x;     r2 = y;
;   x * y = ?
     . . .
    ldi         r1, ar1                     ;   чтобы взять логарифм (x);
    ldi         r2, ar2                     ;   чтобы взять логарифм (y);
;   nop                                     ;   конвейерный конфликт...
;   nop                                     ;   —//— ...
    addi3       *+ar1(ir1), *+ar2(ir1), ar3 ;   ..чтобы взять
                                                натуральную величину..
;   nop                                     ;   конвейерный конфликт...
;   nop                                     ;   —//— ...
    ldi         *+ar3(ir0), r3              ;   r3 = x * y !
     . . .
```

ПРИЛОЖЕНИЕ Б

Рекурсивное вычисление синдромов ошибок S(x) по схеме Горнера.

```
;   ir1 → таблица логарифмирования;
;   ir0 → таблица потенцирования;
;   ar1 → вектор V(x);     r0 = a^k;
    . . .
    rptb    L1                      ;  2КОМАНДЫ, j = 1, 2, ..., 254.
    addi3   *+ar6(ir1), r0, ar6     ;  сложить логарифмы s_k^(j-1) и a^k;
L1  xor3    *+ar6(ir0), ar1++, ar6  ;  к произведению прибавить v_n-j...
    . . .
```

В цикле две команды, но из-за возникновения конвейерных конфликтов в процессоре при модификации адресных регистров и последующем косвенном обращении к внешней памяти число тактов в цикле возрастает до шести.

ПРИЛОЖЕНИЕ В

Вычисление синдромов с помощью БИХ-фильтра.

```
;   ar0 → логарифм g_k;   ar1 → r_k;
;
;
;
;   r0 = логарифм (v_n-j + r_15^(j-1));
;
    . . .
    rptb    L1                      ;  4 ТАКТА; k = 15, 14, ...:
    addi3   *--ar0, r0, ar6         ;  умножить g_k-1 на сумму в r0;
;   nop                             ;  конфликт..
;   nop                             ;  конфликт...
L1  xor     *+ar6(ir0), *--ar1, r1  ;  прибавить r_k-1 к произведению,
||  sti     r1, *ar1                ;  сохранив предыдущее r_k.
    . . .
```

Для случая, когда $g_k = g_{d-k}$,

```
    . . .
    rptb    L1                      ;  5 ТАКТОВ * 8 = 40;
    addi3   *--ar0, r0, ar6         ;  умножить g_k-1 на сумму в r0;
;   nop                             ;  конфликт..
;   nop                             ;  конфликт...
    xor     *+ar6(ir0), *--ar1, r1  ;  прибавить r_k-1 к произведению,
||  sti     r1, *ar1                ;  сохранив предыдущее r_k.
L1  xor     *+ar6(ir0), *++ar2, r2  ;  прибавить r_16-k+1 к произведению,
||  sti     r2, *ar2                ;  сохранив предыдущее r.
```

Преобразование Фурье над 5-точечной последовательностью чисел в поле GF. Исходный вектор: $V(x) = v_0 + v_1x + v_2x^2 + v_3x^3 + v_4x^4$, где $x = \alpha^p$ — набор корней порождающего полинома $G(x)$ ($p = 0, 1, 2, 3, 4$). Вектор результата — спектральные составляющие $V_0, V_1, ..., V_4$. Пусть $x^j = \alpha^{jp}$. Тогда в матричном выражении преобразование исходного вектора имеет вид:

$$\begin{vmatrix} V_0 \\ V_1 \\ V_2 \\ V_3 \\ V_4 \end{vmatrix} = \begin{vmatrix} \alpha^0 & \alpha^0 & \alpha^0 & \alpha^0 & \alpha^0 \\ \alpha^0 & \alpha^1 & \alpha^2 & \alpha^3 & \alpha^4 \\ \alpha^0 & \alpha^2 & \alpha^4 & \alpha^6 & \alpha^8 \\ \alpha^0 & \alpha^3 & \alpha^6 & \alpha^9 & \alpha^{12} \\ \alpha^0 & \alpha^4 & \alpha^8 & \alpha^{12} & \alpha^{16} \end{vmatrix} \times \begin{vmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{vmatrix}.$$

Поскольку показатели степени при α существуют в кольце 5, то

$$\begin{vmatrix} V_0 \\ V_1 \\ V_2 \\ V_3 \\ V_4 \end{vmatrix} = \begin{vmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & \alpha^1 & \alpha^2 & \alpha^3 & \alpha^4 \\ 1 & \alpha^2 & \alpha^4 & \alpha^1 & \alpha^3 \\ 1 & \alpha^3 & \alpha^1 & \alpha^4 & \alpha^2 \\ 1 & \alpha^4 & \alpha^3 & \alpha^2 & \alpha^1 \end{vmatrix} \times \begin{vmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{vmatrix}.$$

Вычтя первую строку матрицы из каждой последующей, получим

$$\begin{vmatrix} V_1 - V_0 \\ V_2 - V_0 \\ V_3 - V_0 \\ V_4 - V_0 \end{vmatrix} = \begin{vmatrix} \alpha^1 - 1 & \alpha^2 - 1 & \alpha^3 - 1 & \alpha^4 - 1 \\ \alpha^2 - 1 & \alpha^4 - 1 & \alpha^1 - 1 & \alpha^3 - 1 \\ \alpha^3 - 1 & \alpha^1 - 1 & \alpha^4 - 1 & \alpha^2 - 1 \\ \alpha^4 - 1 & \alpha^3 - 1 & \alpha^2 - 1 & \alpha^1 - 1 \end{vmatrix} \times \begin{vmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{vmatrix}.$$

Для преобразования полученной матрицы в циклическую в соответствии с алгоритмом Рейдера необходимо переупорядочить элементы вектора как группу степеней примитивного элемента поля GF(5) (таковым является элемент $\alpha = 2$):

Входная последовательность:	Выходная последовательность:
$2^0 = 1$	$2^0 = 1$
$2^1 = 2$	$2^{-1} = 3$
$2^2 = 4$	$2^{-2} = 4$
$2^3 = 3$	$2^{-3} = 2$

Осуществив переупорядочение в соответствии с изложенным, получим циклическую матрицу

$$\begin{vmatrix} V_1 - V_0 \\ V_3 - V_0 \\ V_4 - V_0 \\ V_2 - V_0 \end{vmatrix} = \begin{vmatrix} \alpha^1 - 1 & \alpha^2 - 1 & \alpha^4 - 1 & \alpha^3 - 1 \\ \alpha^3 - 1 & \alpha^1 - 1 & \alpha^2 - 1 & \alpha^4 - 1 \\ \alpha^4 - 1 & \alpha^3 - 1 & \alpha^1 - 1 & \alpha^2 - 1 \\ \alpha^2 - 1 & \alpha^4 - 1 & \alpha^3 - 1 & \alpha^1 - 1 \end{vmatrix} \times \begin{vmatrix} v_1 \\ v_2 \\ v_4 \\ v_3 \end{vmatrix},$$

или, после переобозначений,

$$\begin{vmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{vmatrix} = \begin{vmatrix} b_0 & b_3 & b_2 & b_1 \\ b_1 & b_0 & b_3 & b_2 \\ b_2 & b_1 & b_0 & b_3 \\ b_3 & b_2 & b_1 & b_0 \end{vmatrix} \times \begin{vmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{vmatrix},$$

где

$$\begin{vmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{vmatrix} = \begin{vmatrix} v_1 \\ v_2 \\ v_4 \\ v_3 \end{vmatrix}, \qquad \begin{vmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{vmatrix} = \begin{vmatrix} (\alpha^1 - 1) \\ (\alpha^3 - 1) \\ (\alpha^4 - 1) \\ (\alpha^2 - 1) \end{vmatrix}, \qquad \begin{vmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{vmatrix} = \begin{vmatrix} V_1 \\ V_3 \\ V_4 \\ V_2 \end{vmatrix}.$$

Вектор $c(x) = c_0 + c_1x + c_2x^2 + c_3x^3$ есть линейная свертка полиномов $a(x)$ и $b(x)$ такого же порядка по модулю $(x^4 - 1)$, или, что в данном случае одно и то же, циклическая свертка:

$$\begin{vmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{vmatrix} = \begin{vmatrix} b_0 & b_3 & b_2 & b_1 \\ b_1 & b_0 & b_3 & b_2 \\ b_2 & b_1 & b_0 & b_3 \\ b_3 & b_2 & b_1 & b_0 \end{vmatrix} \times \begin{vmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{vmatrix}.$$

Воспользуемся алгоритмом вычисления коротких сверток Винограда для полей характеристики 2; в [1] для нескольких начальных значений длины свертки эти разложения приведены (см. рис.11.1 и 11.2 в [1]):

$$\begin{vmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{vmatrix} = \begin{vmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{vmatrix} \times \left[\begin{vmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \times \begin{vmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{vmatrix} \right] \times \left[\begin{vmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{vmatrix} \times \begin{vmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{vmatrix} \right].$$

Для ортогонального базиса фурье-преобразования $\sum_{i=0}^{4} \alpha^i = 0$ и, следовательно, $\sum_{j=0}^{3} b_j = 1$. Поэтому девять умножений, содержащихся в разложении свертки по Винограду, сокращаются до пяти:

$$\begin{vmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{vmatrix} = \begin{vmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{vmatrix} \times \left[\begin{vmatrix} b_0 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & b_0 + b_2 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & b_0 + b_2 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & b_0 + b_3 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & b_0 + b_1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \end{vmatrix} \times \left[\begin{vmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \times \begin{vmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{vmatrix} \right].$$

СПИСОК ЛИТЕРАТУРЫ

1. **Блейхут Р. Е.** Теория и практика кодов, контролирующих ошибки.—М.: Мир, 1986.
2. **Типикин А. П., Петров В. В., Бабанин А. Г.** Коррекция ошибок в оптических накопителях информации.—Киев: Наук. думка, 1990.
3. **Pat. 4567594 USA.** Reed — Solomon error detecting and correcting system employing pipelined processors /Shirish P. Deodhar.—Publ. 22.01.86.
4. **Pat. 4584686 USA.** Reed — Solomon error correction apparatus /Keith R. Fritze.—Publ. 22.04.86.
5. **Pat. 4633471 USA.** Error detection and correction in an optical storage system /S. Robert Perers, Michael J. O'Keeffe.—Publ. 30.12.86.

6. **Shao H. M., Reed I. S.** On the VLSI design of a pipeline Reed — Solomon decoder using systolic array // IEEE Trans. on Comput.—1988.—37.—P. 1273.
7. **Truong T. K., Miller R. L., Reed I. S.** Fast technique for computing syndromes of B.C.H. and Reed — Solomon codes // Electron. Lett.—1979.—15, N 22.—P. 720.
8. **Miller R. L., Truong T. K., Reed I. S.** Efficient program for decoding the (255, 223) Reed — Solomon code over GF(2^8) with both errors and erasures, using transform decoding // IEE Proc.—1980.—127, N 4.—P. 136.
9. **Козлачков В. А., Коршевер И. И., Полозков П. А. и др.** Обнаружение и исправление ошибок в накопителях на магнитооптических дисках // Автометрия.—1991.—№ 6.
10. **130 mm** Rewritable Optical Disk Cartridges // Document ISO/IEC JTC 1/SC 23, N 283, 1989-09-29 (Second Version of ISO/IEC DP10089).
11. **Texas** Instruments: Third-Generation TMS320 User's Guide, 1988.

Поступила в редакцию 27 апреля 1994 г.