УДК 004.315.7

# Устройство для вычисления скалярного произведения векторов с коррекцией ошибок на базе системы остаточных классов

Р.А. Соловьев, Д.В. Тельпухов, Е.С. Балака

Институт проблем проектирования в микроэлектронике Российской академии наук (ИППМ РАН), turbo@ippm.ru

***Аннотация* — В статье рассмотрен подход к построению аппаратной реализации для операции вычисления скалярного произведения векторов. Операция выполняется в системе остаточных классов (СОК), что дает возможность обнаружения и коррекции ошибок за счет дополнительных модулярных каналов. Приведено описание устройства и всех его ключевых блоков на базе эффективных реализаций прямых и обратных преобразователей модулярной арифметики, а также индексных умножителей, построенных на базе таблиц. Таблицы, в свою очередь, могут быть дополнительно защищены от сбоев существующими кодовыми методами.**

***Ключевые слова* — система остаточных классов, модулярная арифметика, полиадический код, коррекция ошибок, резервирование.**

## I. Введение

В некоторых критических к сбоям областях промышленности, как например, в космических исследованиях, всё больше внимания уделяется вычислительным блокам, умеющим обнаруживать и исправлять ошибки в процессе вычислений. Достигается это за счет резервирования [1], применения специальных технологий для повышения надежности элементной базы (например, библиотека радиационно-стойких стандартных ячеек) [2] и применения кодов, контролирующих ошибки, таких как коды Рида-Соломона [3] и других БЧХ-кодов [4]. Однако большинство известных алгоритмов для исправления ошибок подходят лишь для хранения и передачи информации. Система остаточных классов (также известная как модулярная арифметика) позволяет контролировать, в том числе и арифметические операции.

Одной из базовых операций задач цифровой обработки сигналов (ЦОС) является скалярное произведение векторов (СПВ) [5], которое используется почти во всех операциях над матрицами. Появление даже единичной ошибки в расчете для больших векторов может полностью испортить результат и потребовать длительного пересчета. В связи с частым использованием этой операции её защита от ошибок является весьма актуальной.

В статье предлагается использовать аппарат модулярной арифметики для проектирования аппаратной реализации скалярного произведения векторов с коррекцией ошибок. Используется реализация с фиксированной запятой с заданным динамическим диапазоном (то есть, известно максимальное возможно значение на выходе устройства). Также в работе делается упор на сохранение быстродействия устройства на приемлемом уровне. Разработанные средства имеют синхронную конвейерную структуру, поэтому результат вычисления появляется на выходе с задержкой в несколько тактов.

Пусть заданы векторы: $X = (x_1, x_2, \dots, x_l)$ и $Y = (y_1, y_2, \dots, y_l)$ одинаковой длины $l$. Размерность каждого элемента вектора равна $k$ бит. Скалярное произведение векторов задается следующей формулой:

$$\langle X, Y \rangle = x_1 y_1 + x_2 y_2 + \cdots + x_l y_l = S.$$

Аппаратная реализация такой операции в вычислительных блоках, не требующих высокого быстродействия или где длина векторов мала или неизвестна, обычно базируется на MAC-Unit (Multiply–accumulate, умножение с накоплением) (см. рис. 1). Здесь умножение и сложение выполняются за один общий такт. Всего требуется $l$ тактов для получения окончательного результата. Результат вычисления $S$ доступен сразу же после поступления на вход последних элементов вектора. Такой подход является универсальным и легким для разработки, однако для устройств ЦОС, где требуется высокое быстродействие и операция СПВ выполняется часто, разрабатывают специализированные вычислительные блоки.

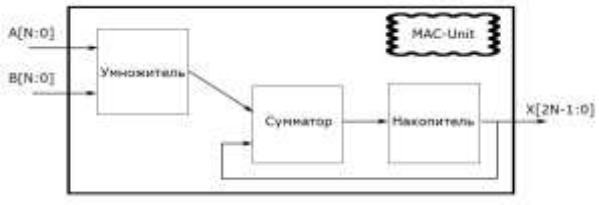


**Рис. 1. MAC-Unit**

Существуют различные подходы для ускорения расчетов. В первом подходе можно подавать больше элементов вектора за один такт, увеличивая число сумматоров и умножителей, что уменьшит общее время расчета конечного результата.

Во втором подходе арифметические блоки разбиваются на несколько подблоков с включением элементов памяти между ними. Производится так называемая конвейеризация. Это позволяет повысить тактовую частоту конечного устройства. Однако такой подход приводит к тому, что конечный результат на выходе устройства получается не сразу, а спустя T тактов, где T – число ступеней конвейера. Впрочем, для задач, где длина векторов большая и число элементов вектора известно, т.е. не требуется оперативно получать промежуточные результаты умножения с накоплением, этот недостаток не играет роли.

На практике применяют некоторое сочетание из методик, в зависимости от тактовой частоты устройства, ограничений на площадь, размерности решаемых задач и т.д.

В данной работе для сравнения полученных результатов рассмотрен позиционный вычислитель СПВ, основанный на втором подходе, так как первый подход почти без изменений может быть применен и к модулярному случаю.

Рассмотрим структуру конвейеризированного вычислителя СПВ. Пусть каждый элемент вектора $x_i = A$ имеет размерность $k$ бит, тогда его можно записать как:

$$A = a_0 2^0 + a_1 2^1 + \cdots + a_{k-1} 2^{k-1}.$$

Тогда произведение векторов можно записать так:

$$\begin{aligned} AB = (a_0 2^0 + a_1 2^1 + \cdots + a_{k-1} 2^{k-1})(b_0 2^0 + b_1 2^1 + \cdots \\ + b_{k-1} 2^{k-1}) \\ = P_0 2^0 + P_1 2^1 + \cdots + P_k 2^k + \cdots \\ + P_{2k-2} 2^{2k-2}. \end{aligned}$$

Здесь $P_i$ частичная сумма при соответствующей степени двойки $2^i$:

$$P_0 = a_0 b_0,$$

$$P_1 = a_1 b_0 + a_0 b_1,$$

$$P_2 = a_2 b_0 + a_1 b_1 + a_0 b_2,$$

...

Известно также, что умножение на степень двойки реализуется путем сдвига. Соответственно частичные суммы после расчета можно сложить на пирамидальном сумматоре, сдвинув соответствующим образом их значение.

Рассмотрим пример для $k = 3$:

$$\begin{aligned} S = AB = (a_0 2^0 + a_1 2^1 + a_2 2^2)(b_0 2^0 + b_1 2^1 + b_2 2^2) \\ = P_0 2^0 + P_1 2^1 + P_2 2^2 + P_3 2^3 + P_4 2^4 \\ = (P_0 2^0 + P_1 2^1) + (P_2 2^0 + P_3 2^1) 2^2 \\ + P_4 2^4 = (S_{10} + S_{11} 2^2) + S_{12} 2^4, \end{aligned}$$

$$S_{10} = P_0 + P_1 2^1 = P_0 + (P_1 \ll 1),$$

$$S_{11} = P_2 + P_3 2^1 = P_2 + (P_3 \ll 1),$$

...

Схема конвейерного вычислителя соответственно может быть представлена следующим образом (рис. 2):

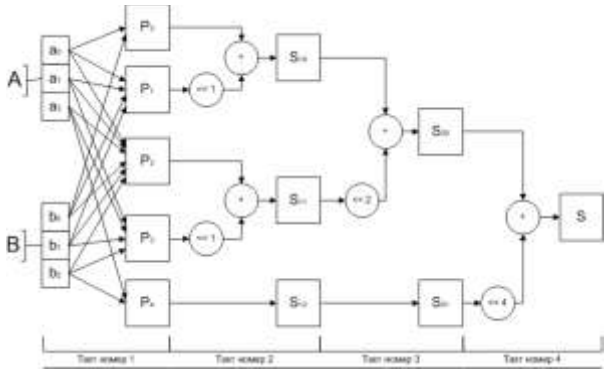


**Рис. 2. Схема конвейеризации умножителя**

На последней ступени вычислений в схеме выполняется сложение с аккумулированным значением (на рисунке 2 оно не приводится). Заметим, что конвейеризация может быть выполнена ещё более глубоко (уже на уровне сложения) для того, чтобы достичь ещё большей частоты, на которой работает схема.

## II. Система остаточных классов и выбор базисных оснований

Представление числа в системе остаточных классов основано на понятии вычета [6] и китайской теореме об остатках [7]. СОК определяется набором попарно взаимно простых модулей $(p_1, p_2, \dots, p_n)$ с произведением $M = p_1 \cdot p_2 \cdot \ldots \cdot p_n$ так, что каждому целому числу $x$ из отрезка $[0; M-1]$ ставится в соответствие набор наименьших неотрицательных вычетов $(x_1, x_2, \dots, x_n)$, где

$$x_1 \equiv x \bmod p_1, x_2 \equiv x \bmod p_2, \dots, x_n \equiv x \bmod p_n.$$

Далее вычет будем обозначать следующим выражением: $|x|_{p_1}$ .

Китайская теорема об остатках гарантирует однозначность представления для чисел из отрезка $[0; M-1]$.

Главным преимуществом такой системы является возможность выполнять основные арифметические операции покомпонентно, если про результат известно, что он не превосходит значение $M$, т.е., если $Z = (z_1, z_2, \dots, z_n) = (x_1, x_2, \dots, x_n) + (y_1, y_2, \dots, y_n)$ , то значения $Z$ можно рассчитать следующим образом:

$$z_1 = |x_1 + y_1|_{p_1},$$

$$z_2 = |x_2 + y_2|_{p_2},$$

...

$$z_n = |x_n + y_n|_{p_n}.$$

Однако операции преобразования в позиционный код, операции сравнения чисел и операции округления в СОК достаточно затратные, поэтому желательно избегать их в процессе проектирования устройств. Для реализации скалярного произведения необходимо заранее знать максимальное значение на выходе устройства и выбрать значение $M$ больше него:

$$M > l(2^k - 1)^2.$$

В итоге общая производительность устройства будет определяться худшим по быстродействию модулярным каналом. Поэтому, чтобы не было дисбаланса между операциями, желательно выбирать модули одного порядка, не забывая при этом про условие попарной простоты.

## III. Обнаружение и коррекция ошибок в системе остаточных классов

Для обнаружения и коррекции ошибок в системе остаточных классов для представления чисел используется лишь часть динамического диапазона. Этот динамический диапазон $M_I = p_1 \cdot p_2 \cdot \ldots \cdot p_u$, где $u < n$, называется информационным, и составляющие его модули $(p_1, p_2, \ldots, p_u)$ называются информационными. Оставшаяся часть модулей $(p_{u+1}, p_{u+2}, \ldots, p_n)$ из полного набора используется для контроля, и эти модули называются контрольными с динамическим диапазоном $M_K = p_{u+1} \cdot p_{u+2} \cdot \ldots \cdot p_n$. В этом случае числа, попадающие в диапазон $[0; M_I)$, называются легитимными, а те, которые попадают в интервал $[M_I; M_I \cdot M_K)$ – нелегитимными.

Известно также, что если каждый из контрольных модулей больше, чем любой из информационных, то такой код позволяет обнаруживать $(n-u)$ ошибок или исправлять $\left\lfloor \frac{(n-u)}{2} \right\rfloor$ ошибок. Для обнаружения достаточно проверить, что результат после преобразования в позиционный вид попадает в легитимный диапазон. Для исправления ошибки требуется последовательно исключать модули из полного набора и тот набор, который даст результат, попадающий в легитимный диапазон, и есть верный.

Пример: для исправления одиночной ошибки требуется два дополнительных модуля. Возьмем три информационных (3,5,7) и два контрольных (8,11) модуля. Легитимный динамический диапазон $M_I = 3 \cdot 5 \cdot 7 = 105$.

Таблица 1

*Коррекция ошибок в СОК*

| Набор модулей | Значения в СОК | Значение в позиционной записи |
|---|---|---|
| (5,7,8,11) | (3,0,0,1) | 1288 |
| (3,7,8,11) | (2,0,0,1) | 56 |
| (3,5,8,11) | (2,3,0,1) | 848 |
| (3,5,7,11) | (2,3,0,1) | 518 |
| (3,5,7,8) | (2,3,0,0) | 728 |

Пусть задано число 56, представим его в виде остатков по всем модулям: $(2,1,0,0,1)$. Внесем ошибку в модулярный канал по модулю 5. Было (2,1,0,0,1), стало (2,3,0,0,1). Начинаем последовательно исключать по одному модулю и выполняем обратное преобразование. Как видно из таблицы 1 только одно значение попадает в легитимный диапазон, оно и является верным.

## IV. Прямой преобразователь из позиционной системы в СОК

Так как данные на вход устройства обычно приходят в стандартном позиционном двоичном виде, то необходимо преобразовать их в модулярный код. Операция взятия остатка от деления для каждого модуля выполняется по одинаковой схеме. Рассмотрим произвольное число $p$ из набора $(p_1, p_2, \ldots, p_n)$ размерности $b$ бит. Требуется найти остаток от деления числа $X$ размерности $k$ бит.

В общем случае количество бит $k$ у входных данных значительно превосходит количество бит $t$ у выходных данных. Представим входные данные $X$ в следующем виде:

$$X = 2^0 \cdot X[0] + 2^1 \cdot X[1] + 2^2 \cdot X[2] + \ldots + 2^{k-1} \cdot X[k-1].$$

Здесь $X[i]$ означает бит в позиции $i$ в двоичной записи числа $X$. Затем воспользуемся следующими свойствами вычетов:

$$|A + B|_p = \left||A|_p + |B|_p\right|_p \text{ и } |A \cdot B|_p = \left||A|_p \cdot |B|_p\right|_p.$$

Тогда вычет $X$ по модулю $p$ можно записать следующим образом:

$$|X|_p = |2^0 \cdot X[0] + 2^1 \cdot X[1] + 2^2 \cdot X[2] + \ldots + 2^{k-1} \cdot X[k-1]|_p =$$

$$= \left||2^0|_p \cdot X[0] + |2^1|_p \cdot X[1] + |2^2|_p \cdot X[2] + \ldots + |2^{k-1}|_p \cdot X[k-1]\right|_p =$$

$$= |A_0 \cdot X[0] + A_1 \cdot X[1] + A_2 \cdot X[2] + \ldots + A_{k-1} \cdot X[k-1]|_p.$$

Константы вида $A_i = |2^i|_p$ могут быть рассчитаны на этапе проектирования устройства, и каждая константа не превосходит значение $p$. Общее значение выражения

$$S = (A_0 \cdot X[0] + A_1 \cdot X[1] + A_2 \cdot X[2] + \ldots + A_{k-1} \cdot X[k-1]) < (p-1) \cdot k. \quad (1)$$

Так как коэффициенты $A_i$ известны, то оценку $S$ сверху можно уменьшить.

Таким образом, чтобы посчитать значение вычета $|X|_p$ потребуется выполнить следующие операции:

1) найти сумму $S = (A_0 \cdot X[0] + A_1 \cdot X[1] + \cdots + A_{k-1} \cdot X[k-1])$;
2) оценить максимальное значение $S$ сверху - $S_{max} < (p-1) \cdot k$;

3) определить, в какой интервал вида $[0:p), [p, 2p), \dots, [(m-1)\cdot p, S_{max}]$ попадает значение $S$, и вычесть из него соответствующее значение 0 для $[0:p)$, $p$ для $[p, 2p)$ и т.д.

Так как в преобразователе есть два четко выраженных этапа, то в синхронных устройствах можно увеличить частоту работы преобразователя увеличив количество тактов вычисления до 2-х. На первом такте вычисляется сумма $S$, на втором ищется интервал, куда она попадает, и затем корректируется с помощью вычитания.

## V. Умножение и сложение по модулю в системе остаточных классов

Пусть на вход умножителю (сумматору) приходят значения $a$ и $b$ размерности $t$ бит, а на выходе умножителя (сумматора) получаем значение $x$. Так как используется умножитель (сумматор) по модулю $p$, то значения на входе и выходе имеют одинаковую размерность.

Сумматор по модулю

Сумматор по модулю может быть реализован через обычное сложение с последующей коррекцией результата на значение $p$, если требуется:

$$|A+B|_p = \begin{cases} A+B, \text{если } (A+B) < p \\ A+B-p, \text{если } (A+B \geq p) \end{cases}$$

Умножитель по модулю

Существует достаточно много подходов к реализации этой операции в микроэлектронных устройствах, зависящих от размерности и характеристики модуля. Рассмотрим основные:

1) *обычное умножение с последующим извлечением остатка* – этот метод редко применяется на практике, так как содержит много лишних операций. Промежуточный результат в этом методе содержит $2t$ бит и затем применяется затратная операция взятия вычета;

2) *полностью табличная реализация*. Используется двумерная таблица, в которой содержатся значения для операции умножения. Подходит для очень маленьких значений $p$ ($p \leq 7$);

3) *индексный метод* [8,9]. Работает только для тех $p$, которые являются простыми числами. В этом методе используются свойства дискретного логарифма. Входные данные по одномерным таблицам размерности $p$ преобразуются в дискретные логарифмы. Затем производится операция сложения по модулю $(p-1)$ и по обратной таблице находится искомое значение. Случай, когда на один из входов приходит значение 0, обрабатывается отдельно. Схема работы представлена на рисунках 3 и 4;

4) *метод разности квадратов* [10, 11]. Метод работает только для нечетных $p$ и базируется на следующей формуле: $A \cdot B = \frac{(A+B)^2}{4} - \frac{(A-B)^2}{4}$.

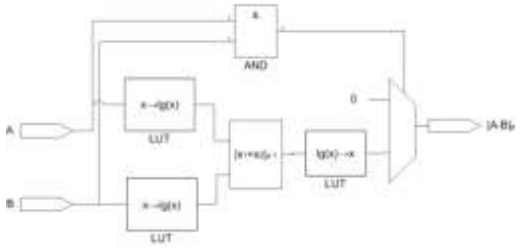


**Рис. 3. Схема работы индексного умножителя**

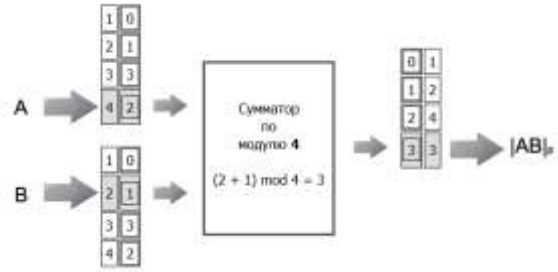


**Рис. 4. Пример работы арифметической части индексного умножителя по модулю для $p = 5$**

$$|A \cdot B|_p = \left| \left| \frac{(A+B)^2}{4} \right|_p - \left| \frac{(A-B)^2}{4} \right|_p \right|_p .$$

На первом этапе рассчитываются значения суммы ( $A+B$ ) и разности ( $A-B$ ). Далее по таблицам размерности $2p$ рассчитываются значения $\left|\frac{(A+B)^2}{4}\right|_p$ и $\left|\frac{(A-B)^2}{4}\right|_p$. На последнем этапе находится разность по модулю $p$.

Два последних подхода являются весьма эффективными и предоставляют дополнительные преимущества: из-за наличия четко выраженных этапов эти схемы хорошо конвейеризируются (три такта); они имеют в своей структуре таблицы, которые можно дополнительно защищать от ошибок с помощью кодов Хемминга [12].

## VI. Обратный преобразователь

Обратный преобразователь используется для восстановления числа $X$ из набора остатков ( $x_1, x_2, \dots, x_n$ ). Для конвейерных структур лучше подходят преобразователи на базе полиадического кода (система счисления со смешанным основанием) [13]. Обратное преобразование на базе полиадического кода базируется на идее, что любое число $X$ может быть представлено в системе взаимно простых чисел $p_1, p_2, \dots, p_n$ как:

$$X = a_1 + a_2 \cdot p_1 + a_3 \cdot p_1 \cdot p_2 + \dots + a_i \cdot p_1 \cdot \dots \cdot p_{i-1} + \dots a_n \cdot p_1 \cdot p_2 \cdot \dots \cdot p_{n-1},$$

где $0 \leq a_i < p_i$

$$|X|_{p_1} = x_1 = a_1,$$

$$|X - a_1|_{p_2} = |x_2 - a_1|_{p_2} = |a_2 \cdot p_1| \to a_2 = \left||p_1^{-1}|_{p_2}(x_2 - a_1)\right|_{p_2},$$

$$|X - a_1 - a_2 \cdot p_1|_{p_3} = |a_3 \cdot p_1 \cdot p_2|_{p_3} \to a_3 = \left||p_2^{-1}|_{p_3} \cdot \left(|p_1^{-1}|_{p_3} \cdot (x_3 - a_1) - a_2\right)\right|_{p_3}$$

… … …

$$a_n = \left||p_{n-1}^{-1}|_{p_n} \cdot (|p_{n-2}^{-1}|_{p_n} \cdot \left(\dots |p_2^{-1}|_{p_n} \cdot \left(|p_1^{-1}|_{p_n} \cdot (x_n - a_1) - a_2\right) - \cdots\right) - a_{n-1}\right|_{p_n}.$$

Для применения этого метода требуются константы вида $|p_i^{-1}|_{p_j}$, которые легко рассчитать на этапе проектирования, так как $p_i$ и $p_j$ уже известны. Рассмотрим сначала схему обратного преобразователя без коррекции ошибок. Схема его работы для 4-х модулей представлена на рисунке 5.

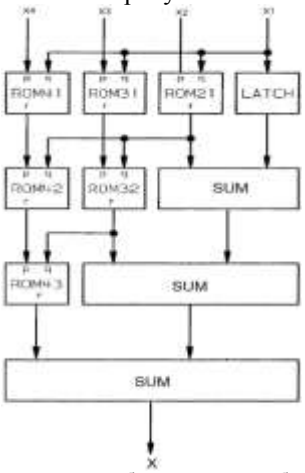


**Рис. 5. Архитектура обратного преобразователя**

Здесь «Вычислитель ROMij» – это устройство, выполняющее следующую операцию: $z = \left||p_i^{-1}|_{p_j} \cdot (a - b)\right|_{p_i} \cdot (p_1 \cdot p_2 \cdot \dots \cdot p_{i-1})$. Это устройство может быть реализовано с помощью двух последовательных этапов. На первом этапе производится вычитание операндов, каждый из которых не превосходит $p_i$. На втором этапе на основе полученной разности выбирается нужное значение из таблицы, содержащей $(p_i - 1)$ элементов, что при малых $p$ является быстрой операцией, не требующей существенных аппаратных затрат. Размерность сумматоров плавно растет от входов к выходу преобразователя.

Преобразователь с обнаружением и коррекцией ошибок можно строить на базе канонических обратных преобразователей, размещая их для параллельной работы и проверяя результат на выходах преобразователей (см. раздел III). Значение, попадающее в легитимный диапазон, и является верным. Назовем каждый из таких обратных преобразователей «исключающим».

Существуют также методы сокращения аппаратных затрат на обратные преобразователи с коррекцией ошибок за счет объединения повторяющихся частей [14].

## VII. Схема работы скалярного умножения векторов в системе остаточных классов

На рисунке 6 представлена схема, вычисляющая скалярное произведение векторов на базе системы остаточных классов. Приведен пример схемы, которая может исправить одиночную ошибку в любом из модулярных каналов. Всего каналов 4, два информационных и два контрольных.

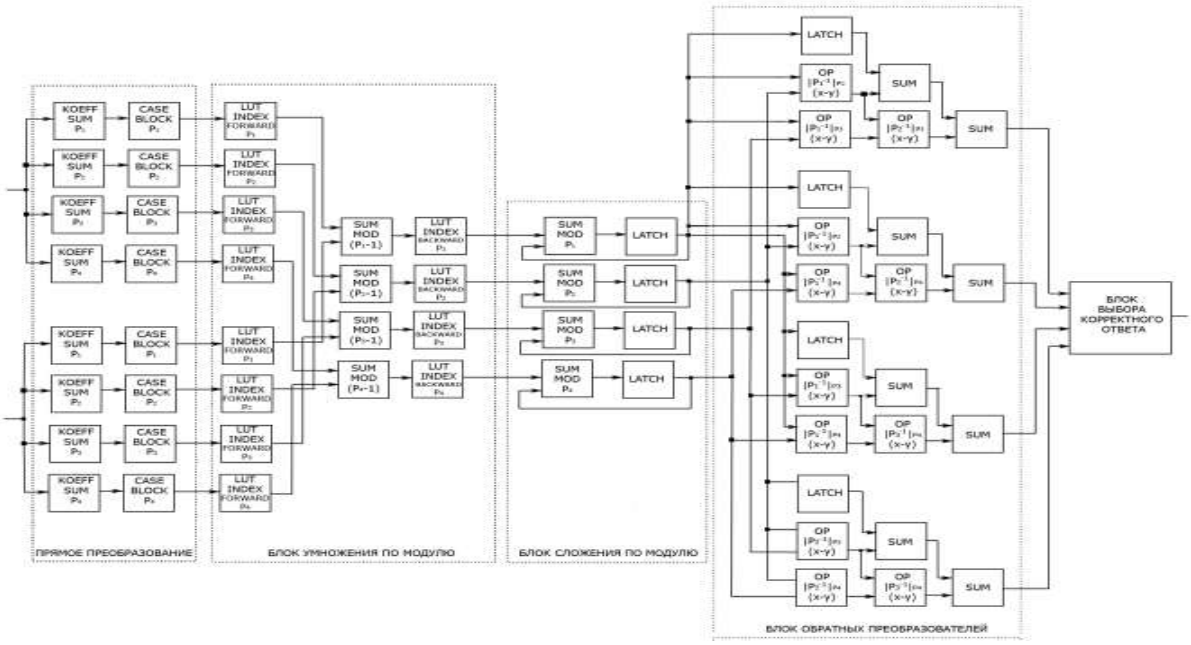


**Рис. 6. Схема работы скалярного умножения векторов с коррекцией ошибок на базе системы остаточных классов**

На схеме представлено 5 этапов вычислений: прямое преобразование, блок умножения по модулю, блок сложения по модулю, обратное преобразование и схема выбора корректного ответа. Каждый вертикальный ряд – это одна ступень конвейера. В этой схеме данные проходят от начала до конца конвейера ровно за 11 тактов.

Рассмотрим каждый этап вычислений.

Прямое преобразование. Первый ряд вычислителей вида «KOEFF SUM Pi» выполняет вычисление суммы S из формулы (1). Вычисления производятся отдельно по каждому модулярному каналу (так как коэффициенты разные) и отдельно по каждому входу.

Второй ряд вычислителей вида «CASE BLOCK Pi» определяет, в какой интервал попало вычисленное значение S, и выдает соответствующий ответ за счет коррекции результата на соответствующее значение из таблицы.

Блок умножения по модулю. Блок умножения по модулю в данном случае выполнен по индексному методу и состоит из трех этапов. На первом такте (LUT INDEX FORWARD Pi) значение преобразуется в индекс, на втором (SUM MOD (Pi-1)) выполняется сложение по модулю (Pi-1), на последнем этапе (LUT INDEX BACKWARD Pi) выполняется обратное преобразование из индексного в модулярный вид.

Данный преобразователь может быть замен на схожий по структуре, выполненный по методу разности квадратов, для тех случаев, когда нечетный модуль не является простым числом.

Блок сложения по модулю. В блоке сложения по модулю данные берутся из накопителя (ячейки памяти), в которых хранится текущий результат вычисления скалярного произведения векторов в модулярном виде, и складываются с данными пришедшими с соответствующих входов умножителя. В первоначальном состоянии накопитель должен быть обнулен.

Блок обратных преобразователей. Блок обратных преобразователей состоит из 4 структурно похожих частей, каждая из которых получена исключением одного из модулей. Весь обратный преобразователь может быть структурно сокращен за счет объединения общих частей, как, например, предлагается в [14].

Обратный преобразователь в скалярном умножении векторов может вызываться не каждый раз, а только в тот момент, когда получен финальный результат в модулярном виде, или когда требуется проверить текущий результат на корректность. В качестве использования можно включать его в работу постоянно в случае повышенной вероятности ошибок (вспышка на солнце) или когда выполняются критически важные расчеты.

Блок выбора корректного ответа. В этом блоке каждый выход из набора исключающих обратных преобразователей сравнивается со значением легитимного динамического диапазона. На выход выдается значение с того входа, где оно попадает в заданный легитимный динамический диапазон.

## VIII. Заключение

В статье приведена подробная схема реализации одной из базовых операций ЦОС – вычисления скалярного произведения векторов. В отличие от мажорирования базовых элементов или всего устройства использование модулярной арифметики позволяет защищать устройство на архитектурно-алгоритмическом уровне и гибко контролировать требуемую избыточность. Удаление или добавление модулярных каналов позволяет как уменьшить избыточность по сравнению с мажорированием, так и увеличивать её в зависимости от требуемой надежности и числового значения вероятности сбоя.

## Литература


[1] Козлов Б.А., Ушаков И.А. Краткий справочник по расчету надежности радиоэлектронной аппаратуры. М., 1966.

[2] Ness, D.C.D., Hescott, C.C.J., and Lilja, D.D.J. Exploring subsets of standard cell libraries to exploit natural fault masking capabilities for reliable logic // In Proceedings of the 17th ACM Great Lakes symposium on VLSI. Stresa-Lago Maggiore, Italy. ACM. 2007. P. 208–211.

[3] Морелос-Сарагоса Р. Искусство помехоустойчивого кодирования. Методы, алгоритмы, применение / пер. с англ. В.Б. Афанасьева. М.: Техносфера. 2006. 320 с.

[4] Сагалович Ю. Л. Введение в алгебраические коды: Учебное пособие. М.: МФТИ, 2007. С. 175-176.

[5] http://en.wikipedia.org/wiki/Multiply%E2%80%93accumulate_operation (дата обращения: 01.04.2014)

[6] Вейль А. Основы теории чисел. М.: Мир, 1972.

[7] http://ru.wikipedia.org/wiki/Китайская_теорема_об_остатках

[8] Jullien G.A. Implementation of multiplication modulo a prime number, with applications to number theoretic transforms // IEEE Transactions on Computers. 1980.

[9] Он-лайн генератор аппаратных описаний для умножителей по модулю: http://vscripts.ru/2012/index-modulo-multiplication.php (дата обращения: 01.04.2014)

[10] Soderstrand M.A. A new hardware implementation of modulo adders for residue number systems // In: Proceedings, 26th Midwest Symposium on Circuits and Systems. 1983. P. 412-415.

[11] Он-лайн генератор аппаратных описаний для умножителей по модулю: http://vscripts.ru/2012/index-modulo-multiplication-sqr.php (дата обращения: 01.04.2014)

[12] Блейхут Р. Теория и практика кодов, контролирующих ошибки. Пер. с англ. М.: Мир, 1986. 576 с.

[13] Тельпухов Д.В. Построение обратных преобразователей модулярной логарифметики для устройств цифровой обработки сигналов // Информационные технологии. 2011. №4.

[14] Патент US4752904. Efficient structure for computing mixed-radix projections from residue number systems. 26.06.1988.