Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт математики и естественных наук

Кафедра высшей математики

Утверждена распоряжением по институту
от_____№_____

Допущена к защите
«25» июня 2020 г.
Зав. кафедрой высшей математики
канд. физ.-мат. наук, доцент,
Бондарь В.В.
(уч. степень, уч. звание, ФИО зав. каф.)

_____
(подпись зав. кафедрой)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Исследование и разработка методов и алгоритмов обнаружения, локализации и исправления ошибок в модулярном коде

Выполнила:
Кучукова Екатерина Андреевна
студентка 4 курса, группы МКН-б-о-16-1
направления подготовки 02.03.01
Математика и компьютерные науки
направленность (профиль)
«Вычислительные, программные, информационные системы и компьютерные технологии»,
очной формы обучения

_____
(Подпись)

Нормоконтролер:
Скорцова Ольга Ивановна
(ФИО)

старший преподаватель кафедры высшей математики
(учена степень, звание, должность)

_____
(Подпись)

Руководитель:
Бондарь Виктория Витальевна
(ФИО)

канд. физ.-мат. наук, доцент, заведующий кафедрой высшей математики
(учёная степень, звание, должность)

_____
(Подпись)

Дата защиты «___» июня 2020 г.

Оценка _____

Ставрополь, 2020 г.

Содержание

Введение 4

1 Аналитический обзор методов и алгоритмов представления и обработки данных в системе остаточных классов 6

1.1 Система остаточных классов и её свойства 6

1.2 Анализ методов перевода чисел из ПСС в СОК 11

1.2.1 Преобразователи ПСС в СОК с использованием ПЗУ 12

1.2.2 Преобразование двоичного кода в СОК с использованием свойств периода 13

1.2.3 Прямое преобразование с использованием метода Паскаля 16

1.2.4 Прямое преобразование для FPGA 18

1.2.5 Модули специального вида $(2^n \pm k)$ 20

1.3 Анализ методов перевода чисел из СОК в ПСС 21

1.3.1 Китайская теорема об остатках 22

1.3.2 Приближенный метод 27

1.3.3 Обобщённо позиционная система счисления (ОПСС) 34

1.3.4 КТО-I, КТО-II, КТО-III 38

1.3.5 Функции ядра 41

1.4 Масштабирование оснований СОК 44

1.5 Выводы по первой главе 49

2 Коды исправления ошибок в модулярном коде 51

2.1 Метод проекции 52

2.2 Метод синдрома 56

2.2.1 Выделение синдромов для идентификации ошибок 56

2.2.2 Обнаружение и исправления ошибок 62
2.2.3 Пример 67
2.2.4 Доказательство корректности метода исправления ошибок на базе синдрома 69
2.3 Выводы по второй главе 76
3 Модификация методов обнаружения локализации и исправления ошибок 77
3.1 Модификация метода синдрома в СОК 77
3.2 Выбор оснований СОК 79
3.3 Выводы по третьей главе 82
Заключение 84
Список использованных источников 85

Введение

Требования к повышению производительности, снижению энергопотребления портативной электроники позволили уменьшить размеры оборудования, но в то же время возникли проблемы надёжности устройств. Обеспечение надёжности хранения и обработки информации, снижения энергопотребления являются актуальными проблемами цифровой экономики. Обеспечение отказоустойчивости цифровых систем представляется наилучшим решением для повышения надёжности. Отказоустойчивость систем может быть обеспечена как аппаратной избыточностью, например, использованием RAID-массивов, так и программной избыточностью – с использованием кодов исправления ошибок, логики самоконтроля, репликации модулей, реконфигурации, однако данные методы имеют следующие недостатки: высокая избыточность, необходимость интенсивных вычислений.

Одним из методов обеспечения надёжности является использование системы остаточный классов (СОК). СОК является непозиционной системой счисления, в которой число может быть представлено в виде вектора чисел меньшей размерности, представляющих собой остатки от деления на набор взаимно простых чисел, называемых модулями. Путём добавления избыточных модулей в существующий набор модулей СОК получающаяся избыточная СОК (ИСОК) может использоваться, чтобы обнаружить и исправить ошибки, не изменяя допустимый диапазон исходной информации. Методы обнаружения и исправления ошибок, основанные на ИСОК, способны исправлять ошибки арифметической обработки [1] из-за шума, производственных дефектов, изменений в процессе, напряжении и температуре или даже преднамеренного ввода неисправности в дополнение к ошибкам передачи или хранения. Вследствие независимости разрядов числа, представленного в СОК, любая ошибка, вносимая в канал модуля, не будет распространяться на другие каналы модуля [2]. Другими словами, ошибочные биты в разряде остатка имеют локализованный эффект. Эта уникальная вычислительная возможность

исправления ошибок особенно полезна в приложениях надёжного хранения, безопасности, биомедицины, которые не допускают ошибок.

Объект исследования: Система остаточных классов.

Предмет исследования: Методы и алгоритмы обнаружения, локализации и исправления ошибок.

Цели исследования:

1. Обзор методов и алгоритмов представления и обработки данных в системе остаточных классов, исследование кодов исправления ошибок в модулярном коде.
2. Модификация методов обнаружения, локализации и исправления ошибок.

Основные задачи выпускной квалификационной работы:

1. Исследовать теоретические основы представления и обработки данных в системе остаточных классов.
2. Предложить эффективные алгоритмы обнаружения и исправления ошибок для аппаратной реализации.
3. Проанализировать модификации методов обнаружения, локализации и исправления ошибок.

1 Аналитический обзор методов и алгоритмов представления и обработки данных в системе остаточных классов

1.1 Система остаточных классов и её свойства

Арифметические свойства той или иной системы счисления прежде всего определяются характером межразрядных связей, появляющихся в ходе выполнения операций над кодовыми словами. Исследования показали, что в рамках обычной позиционной системы счисления (ПСС) значительного ускорения выполнения операций добиться невозможно. Это объясняется тем, что в ПСС значение любого разряда числа, кроме младшего, являющегося результатом двухместной арифметической операции, зависит не только от значения одноименных операндов, но и от переносов всех младших разрядов, т.е. ПСС обладает строго последовательной структурой. Сегодня при совершенствовании технологии производства средств вычислительной техники и внедрении новых, более эффективных способов организации и проведения вычислений предпочтение отдаётся вычислительным структурам, обладающим способностями к параллельной обработке информации. Этими особенностями обладают непозиционные коды с параллельной структурой, которые позволяют реализовать идею распараллеливания операций на уровне выполнения элементарных арифметических действий.

Система остаточных классов (СОК) представляет собой непозиционную систему счисления, позволяющую разбивать числа большой длины на ряд независимых разрядов малой длины, что позволяет ускорить вычисления и организовать их параллелизм. Главным преимуществом СОК является способность очень быстрого выполнения операций сложения и умножения по сравнению со всеми остальными системами счисления, что обуславливает большой интерес к СОК в тех областях, в которых требуются большие объёмы вычислений указанных операций.

Под системой остаточных классов понимают такую непозиционную систему счисления, в которой любое целое неотрицательное число A, $0 \leq A < M$, можно представить в виде набора остатков от деления этого числа на выбранные основания системы $\{m_1, m_2, \dots, m_n\}$, называемые модулями СОК, т.е.

$$A = (\alpha_1, \alpha_2, \dots, \alpha_n), \text{ где } \alpha_i = A - \left[\frac{A}{m_i}\right] \cdot m_i,\ i = \overline{1, n}, \tag{1.1}$$

где $M = \prod_{i=1}^{n} m_i$ и называется диапазоном СОК [3].

Возможность такого представления числа определяется теоремой о делении с остатком в кольце целых чисел [4]. Если $A \in Z$, $m \in Z$, $m \neq 0$, то существуют единственные $p \in Z$ и $\alpha \in Z$ такие, что

$$A = pm + \alpha,\ 0 \leq \alpha < |m|,\ p = \left[\frac{A}{m}\right]. \tag{1.2}$$

Несложно заметить, что каждый остаток α_i получается независимо от других и содержит информацию обо всём числе. Установить взаимно-однозначное соответствие между целыми числами из диапазона $[0, M)$ и их остатками позволяет Китайская теорема об остатках.

Принципиальная возможность применения СОК в вычислительных алгоритмах обуславливается наличием определённого изоморфизма между математическими операциями над целыми числами и соответствующими операциями над системой целых неотрицательных остатков по отдельным модулям. Причём сложение, умножение, возведение в целую положительную степень любых целых положительных чисел совершенно идентичны соответствующим операциям, выполняемым над системой остатков.

Пусть операнды A и B, а также результаты операций сложения и умножения $A + B$ и $A \cdot B$ представлены соответственно остатками α_i, β_i, γ_i, δ_i по основаниям m_i ($i = \overline{1, n}$), причём оба числа и результаты операций находятся в диапазоне $[0, M)$, т. е.

$$\begin{gathered} A = (\alpha_1, \alpha_2, \dots, \alpha_n), \\ B = (\beta_1, \beta_2, \dots, \beta_n), \\ A + B = (\gamma_1, \gamma_2, \dots, \gamma_n), \end{gathered} \tag{1.3}$$

$$A \cdot B = (\delta_1, \delta_2, \dots, \delta_n),$$

и $0 \le A < M$, $0 \le B < M$, $0 \le A + B < M$, $0 \le A \cdot B < M$.

Выражения (1.3) можно переписать в виде

$$\gamma_i = \alpha_i + \beta_i \pmod{m_i},\ \delta_i = \alpha_i \cdot \beta_i \pmod{m_i}. \tag{1.4}$$

При этом в качестве цифры результата берётся соответственно

$$\gamma_i = \alpha_i + \beta_i - \left[\frac{\alpha_i + \beta_i}{m_i}\right] m_i, \quad \delta_i = \alpha_i \beta_i - \left[\frac{\alpha_i \beta_i}{m_i}\right] m_i. \tag{1.5}$$

Справедливость этих правил выполнения арифметических действий в СОК непосредственно вытекает из свойств сравнения.

Действительно, на основании (1.1) можно записать

$$\gamma_i = A + B - \left[\frac{A+B}{m_i}\right] m_i,\ (i = \overline{1,n}).$$

Из представления A и B по теореме о делении с остатком (1.2) следует, что $A = k_i m_i + \alpha_i$, $B = l_i m_i + \beta_i$, $(i = \overline{1,n})$, $k_i \in Z$, $k_i \ge 0$, $l_i \in Z$, $l_i \ge 0$.

Тогда $A + B = (k_i + l_i) m_i + \alpha_i + \beta_i$, $\left[\frac{A+B}{m_i}\right] = k_i + l_i + \left[\frac{\alpha_i + \beta_i}{m_i}\right]$.

Откуда, $\gamma_i = \alpha_i + \beta_i - \left[\frac{\alpha_i + \beta_i}{m_i}\right] m_i$.

В случае умножения

$$\delta_i = AB - \left[\frac{AB}{m_i}\right] m_i.$$

Тогда

$$AB = k_i l_i {m_i}^2 + (\alpha_i l_i + \beta_i k_i) m_i + \alpha_i \beta_i,\ \left[\frac{AB}{m_i}\right] = k_i l_i m_i + \alpha_i l_i + \beta_i k_i + \left[\frac{\alpha_i \beta_i}{m_i}\right].$$

Следовательно, $\delta_i = \alpha_i \beta_i - \left[\frac{\alpha_i \beta_i}{m_i}\right] m_i$, $(i = \overline{1,n})$.

В отличие от позиционной системы счисления (ПСС), в которой число A представляется в виде

$$A = A_n N^n + A_{n-1} N^{n-1} + \dots + A_1 N^1 + A_0 N^0 = \sum_{i=0}^{n} A_i N^i, \tag{1.6}$$

где N – основание ПСС, значение числа в модулярном коде не зависит от местоположения каждого разряда в его представлении, а зависит от значения

основания соответствующего разряда. Поэтому модулярный код является непозиционным.

Таким образом, выполнение арифметических операций в модулярном коде производится независимо по каждому из модулей, что и указывает на параллелизм данной системы. Это обстоятельство определяет возможность независимой обработки, т. е. поразрядного выполнения операций. Это свойство избавляет от необходимости «занимать» или «переносить» единицу старшего разряда, что приводит к появлению кодов с параллельной структурой. Это позволяет распараллелить алгоритмы при выполнении арифметических операций.

Итак, операции сложения и умножения над числами, представленными в СОК, сводятся к соответствующим операциям над цифрами этого представления. Это относится и к любым сложным операциям, составленным из операций сложения и умножения, например, к возведению в степень, к вычислению значений многочлена и т. п. Операция вычитания в СОК заменяется сложением с аддитивной инверсией отрицательного числа. Все эти операции модульные, т. е. не требуют позиционных характеристик обрабатываемых чисел.

Исследования новой системы выявили целый ряд её преимуществ.

Прежде всего, это максимальный параллелизм. Для оценки уровня параллелизма системы счисления вводится специальный показатель $\Pi(v)$, который для СОК принимает максимально возможное значение $\Pi(v) = 1$. Это говорит об отсутствии межразрядных связей в числе, представленном в СОК.

Другим важным достоинством этой системы является малоразрядность остатков. Поэтому, ввиду малого количества возможных кодовых комбинаций появляется возможность построения табличной арифметики. При этом большинство операций превращаются в однотактовые, осуществляемые простой выборкой из таблиц. Это достоинство удалось учесть после появления поколения ЭВМ, работающего на основе БИС и СБИС (середина 70-х годов). По мере совершенствования технологии производства запоминающих устройств с

высокой плотностью записи информации, составляющих техническую основу табличного метода вычислений, интерес к СОК неуклонно возрастает.

Следующее важное качество СОК – реализация принципа конвейерной обработки информации. Это означает, что при выполнении вычислений модульные и следующие за ними операции удаётся совместить во времени даже тогда, когда очередные операции зависят от результатов текущих, ещё не закончившихся операций. Таким образом, алгоритмы модулярной арифметики обладают конвейерной структурой. При работе в конвейерном режиме большинство многотактных операций становятся однотактными, что ведёт к значительному повышению производительности вычислительных устройств.

Ещё одна отличительная особенность этой системы, рассмотренная в данной дипломной работе – её высокая точность, надёжность и способность к самокоррекции. Причём в СОК можно построить непозиционные коды, обнаруживающие и исправляющие ошибки, которые являются полностью арифметическими, т.е. в этих кодах информативная и контрольная части равноправны относительно любой операции. Эта особенность предоставляет возможность варьировать корректирующей способностью кода за счёт изменения точности вычислений. Кроме того, т. к. в этой системе информация представляется параллельно по различным основаниям, то появление ошибки по одному из них не влияет на точность информации в других параллельных каналах. Поэтому можно говорить о поразрядном характере распространения ошибок, что облегчает их локализацию и исправление по сравнению с позиционными системами счисления, которые не обладают такой степенью параллелизма.

Конечно, и эта система не лишена недостатков. К ним относится невозможность визуального сравнения чисел, отсутствие признаков выхода результатов операций за пределы диапазона, ограниченность действия системы сферой целых положительных чисел, получение во всех случаях точного результата операции, а также трудность выполнения немодульных операций. Но эти недостатки не являются непреодолимыми. К настоящему времени

разработаны и продолжают разрабатываться алгоритмы выполнения немодульных операций, причём приоритет отдаётся разработке алгоритмов, отвечающих требованию параллельной обработки информации, идёт поиск наиболее удобных интегральных характеристик модулярного кода, облегчающих выполнение немодульных операций (ранг, ядро, коэффициенты полиадического представления чисел, интервальный индекс и др.). Кроме того, рассматриваются возможности совмещения позиционного и непозиционного процессоров, информация в которых может быть представлена в естественной и нормальной формах, двоичной, десятичной и остаточной арифметике. При этом множественное представление информации позволяет реализовать достоинства разнотипных микропроцессоров в распределённых вычислительных системах, и достоинства позиционных и непозиционных систем счисления, и ослабить или вовсе исключить их недостатки.

Использование модулярной арифметики, с одной стороны, позволяет эффективно реализовать арифметические операции: сложения и умножения чисел, но при этом алгоритмы определения знака числа, сравнения чисел, Евклидово деление, масштабирование и т. д. являются вычислительно сложными. Рассмотрим основные методы и алгоритмы, необходимые для работы в СОК и для обнаружения, локализации и исправления ошибок в модулярном коде:

1. Алгоритмы перевода чисел из ПСС в СОК, рассмотренные в параграфе 1.2.
2. Алгоритмы перевода чисел из СОК в ПСС, анализ которых представлен в параграфе 1.3.
3. Алгоритмы масштабирование чисел, исследование которых приведено в параграфе 1.4.

1.2 Анализ методов перевода чисел из ПСС в СОК

Рассмотрим различные методы перевода чисел в СОК, описанные в литературе. Интуитивно простой метод состоит в делении с остатком, и

отбрасывании неполного частного. Но, как известно, деление является вычислительно сложным процессом [5]. Для решения этой проблемы были исследованы альтернативные методы получения остатка.

1.2.1 Преобразователи ПСС в СОК с использованием ПЗУ

Дженкинс и Леон [6] предложили считывать последовательно остатки по модулю m_i, соответствующие всем битам входного числа из постоянного запоминающего устройства (ПЗУ), и выполнить сложение по модулю m_i. Стуретис [7] предложил считывать остатки, соответствующие различным байтам во входном слове, параллельно из ПЗУ и добавлять их, используя сумматоры по модулю m_i.

Алия и Мартинелли [8] предложили прямое преобразование для данного n-битного входного двоичного слова с использованием $n/2$ обрабатывающих элементов, каждый из которых хранит остатки, соответствующие 2^j и 2^{j+1} (т. е. j-й и $j+1$-й битовые позиции) для $j = 0, \ldots, n/2$ и сложение этих остатков по модулю m_i, если значение соответствующего бита равно «1». Далее результаты $n/2$ обрабатывающих элементов складываются деревом сумматоров по модулю m_i для получения конечного остатка.

Капочелли и Джанкарло [9] предложили использовать t обрабатывающих элементов, где каждый из $t = [n/\log_2 n]$ вычисляет остаток $\log_2 n$-битного слова путём сложения остатков, соответствующих различным битам этого слова, а затем сложение остатков, полученных из различных обрабатывающих элементов, деревом сумматоров по модулю m_i, содержащее h шагов, где $h = \log_2 t$. Отметим, однако, что только остаток, соответствующий позиции младшего бита (LSB, Least Significant Bit) в каждом слове, сохраняется, а остаток, соответствующий каждой следующей позиции бита, вычисляется в режиме реального времени путём удвоения предыдущего остатка и нахождения

остатка по модулю m_i, используя один вычитатель и один мультиплексор. Таким образом, необходимо t запоминающих элементов.

Более поздние разработки избегают использования ПЗУ и в значительной степени используют комбинационную логику. Они обсуждаются в следующих пунктах.

1.2.2 Преобразование двоичного кода в СОК с использованием свойств периода

Рассмотрим сначала пример нахождения остатка от 892 по модулю 19. Выражая 892 в двоичной форме, мы имеем 11 0111 1100. Из арифметических свойств позиционных систем счисления (1.6) имеем:

$$11\ 0111\ 1100_2 = \\ = 1 \cdot 2^9 + 1 \cdot 2^8 + 0 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 \\ + 0 \cdot 2^1 + 0 \cdot 2^0 = 892_{10}.$$

Используя свойства сравнений, можно заменить степени 2 числами, сравнимыми по модулю, т.е. последовательность степеней 2 по модулю 19 будет выглядеть следующим образом: 1, 2, 4, 8, 16, 13, 7, 14, 9, 18. Таким образом, мы можем сложить по модулю те биты, умноженные на соответствующие степени 2 по модулю, входного бита, значения которых равны «1». Это даёт результат $(4 + 8 + 16 + 13 + 7 + 9 + 18) \bmod 19 = 18$. Заметим, что на каждом шаге, когда добавляется новый остаток, соответствующий новой степени 2, можно последовательно найти остаток по модулю 19, чтобы избежать ненужного роста результата: $\Big(\big((4 + 8) \bmod 19 + 16\big) \bmod 19 + 13\Big) \bmod 19$ и т. д.

Заметим, однако, что некоторые упрощения можно сделать, отметив периодическое свойство остатков $2^k \bmod m$ [10-13]. Обозначая $2^T \equiv 1 \bmod m$, мы знаем, что $2^{\alpha T+i} \equiv 2^i \bmod m$, если T – это период модуля m. Таким образом, исходное число может быть разбито на α блоков (α может быть чётным или нечётным), которые состоят из T битов исходного n-битного двоичного числа, где $\alpha = n/T$, и которые могут быть сложены, чтобы сначала получить T-битное

слово, для которого с помощью процедуры, описанной выше, можно получить остаток по модулю m. Обратим внимание, что «T» обозначается как «порядок» и может быть равно $m-1$ или меньше. Например, для $m=89$, $T=11$ и для $m=19$, $T=18$. Рассмотрим нахождение остатка

$0001\ 0100\ 1110\ 1101\ 1011\ 0001\ 0100\ 1110\ 1101\ 1011 \bmod 19 =$

$89887166171 \bmod 19.$

Таким образом, три 18-битных блока (здесь $T=18$) могут быть сложены для получения

				0001
01	0011	1011	0110	1100
01	0100	1110	1101	1011
10	1000	1010	0100	1000

Таким образом, используя 2 сложения, мы получили число 166472, сравнимое по модулю 19 с исходным числом. Остаток этого 18-битного числа можно получить далее, используя процедуру, рассмотренную в примере в начале пункта, сложив остатки различных степеней 2 по модулю 19. Периодическое свойство $2^k \bmod m$ было использовано для упрощения вычислений.

Другое упрощение возможно для модулей, удовлетворяющих свойству $2^{(m-1)/2} \bmod m = 1$. Рассматривая модуль 19, можно заметить, что $2^9 \equiv -1 \bmod 19$, $2^{10} \equiv -2 \bmod 19$, …, $2^{17} \equiv 10 \bmod 19$, $2^{18} \equiv 1 \bmod 19$, $2^{19} \equiv 2 \bmod 19$ и т.д. Таким образом, остатки в верхней половине периода противоположны по знаку остаткам в нижней половине периода. Это свойство можно использовать для уменьшения длины слова сумматора с сохранением переноса до $(m-1)/2$. Обозначая последовательные слова длиной полупериода $W_0, W_1, W_2, \ldots, W_\alpha$, где α – нечётное, мы должны оценить $\left(\sum_{i=0}^{(\alpha-1)/2} W_{2i} - \sum_{i=0}^{(\alpha-1)/2} W_{2i+1}\right) \bmod m$. Рассматривая тот же пример, мы сначала разобьём данное слово на 9-битные блоки, начиная с младших бит, следующим образом:

$$W_4 = 0001$$

$$W_3 = 0\ 1001\ 1101$$

$$W_2 = 1\ 0110\ 1100$$
$$W_1 = 0\ 1010\ 0111$$
$$W_0 = 0\ 1101\ 1011$$

Таким образом, суммируя W_0, W_2 и W_4, мы получаем $S_e = 10\ 0100\ 1000$, а суммируя W_1 и W_3, получаем $S_o = 1\ 0100\ 0100$. Вычитая S_o из S_e, мы имеем $S = 0001\ 0000\ 0100_2 = 260_{10}$. Остаток результирующего слова можно легко найти, используя другой этап, используя периодическое свойство и конечное сокращение по модулю m, как описано ранее.

Отмечается [10-13], что выбор модулей должен быть таким, чтобы период или полупериод были малы по сравнению с динамическим диапазоном в битах полной СОК, чтобы воспользоваться периодическим свойством остатков.

Интересно, что для специальных модулей вида $2^k - 1$ и $2^k + 1$ можно полностью избежать второго этапа преобразования двоичного кода в СОК слова меньшей длины из T или $T/2$ битов [10]. Для модулей вида $2^k - 1$ входное слово может быть разделено на k-битные блоки, все из которых могут быть сложены для получения окончательного остатка. С другой стороны, для модулей вида $2^k + 1$, складываются все чётные k-битовые блоки и все нечётные k-битовые блоки для получения S_e и S_o, соответственно, и один конечный сумматор даёт $(S_e - S_o) \bmod (2^k + 1)$. Например, $892 \bmod 15 = (0011\ 0111\ 1100)_2 \bmod 15 = (3 + 7 + 12) \bmod 15 = 7$ и $892 \bmod 17 = (3 - 7 + 12) = 8$.

Петтенги, Чавес и Соуза [14] предложили для модулей вида $2^n \pm k$ переписать веса (остатки $2^j \bmod m_i$) таким образом, чтобы уменьшить ширину конечного сумматора в двоичном преобразовании в СОК по сравнению с тем, который необходим в конструкциях, использующих период или полупериод. В этом методе отрицательные веса рассматриваются как положительные веса, а биты, соответствующие отрицательным весам, дополняются и добавляется поправочный коэффициент. Остатки u_j допускаются такими, что $-2^n \mp k + 3 \leq u_j \leq 2^n \pm k - 3$.

Например, для модуля 37 показаны остатки, соответствующие 20-битному динамическому диапазону, за полный период для исходного и модифицированного случаев. Поскольку период равен 18, последние два остатка переписываются как -1 и -2. Таким образом, общая взвешенная сумма в наихудшем случае (соответствующая всем входным битам, равным 1) составляет 328 по сравнению с 402 в первом случае. Чтобы избежать отрицательных весов, мы можем рассматривать последние два веса как 1 и 2, но дополнить входные данные и добавить поправочный член 34. Например для 20-битного ввода слова 000…00, 000…01, 000…010, 0000…011, после дополнения последних 2 битов мы имеем 11, 10, 01 и 00, добавив соответствующие «положительные» остатки и добавив поправочный член $\mathrm{COR} = 34$, мы получаем 0, 35, 36 и 34, которые могут быть проверены на правильность.

В альтернативном методе остатки таковы, что $-\frac{2^n \mp k+1}{2} \leq u_j \leq \frac{2^n \pm k-1}{2}$, как показано в четвертом ряду. Таким образом, наихудшая сумма (рассматривающая все отрицательные веса как положительные) составляет всего 177, включая поправочный член $\mathrm{COR} = 3$. Например, для 20-битного входа 0000 0010 0101 1111 0010, инвертирующего биты с отрицательными весами, мы имеем $0000\ 0100\ 1011\ 010\ 0001 = 13 + 3 = 16$, как и ожидалось.

1.2.3 Прямое преобразование с использованием метода Паскаля

Премкумар [15], Премкумар и Лай [16] описали метод прямого преобразования без использования ПЗУ. Они обозначают этот метод как «модульное возведение в степень». В основном, в этом методе различные остатки степеней 2 (т. е. $2^x \bmod m_i$) получаются с использованием логических функций. Для описания метода приведём пример. Рассмотрим поиск $2^{s_3 s_2 s_1 s_0} \bmod 13$, где экспонента является 4-битным двоичным словом. Мы можем записать это выражение как

$$2^{s_3s_2s_1s_0} \bmod 13 = 2^{8s_3+4s_2}4^{s_1}2^{s_0} \bmod 13 = 256^{s_3}16^{s_2}4^{s_1}2^{s_0} \bmod 13 =$$
$$= (255s_3 + 1)(15s_2 + 1)\ 4^{s_1}2^{s_0} \bmod 13 =$$
$$= (3s_3s_2 + 8s_3 + 2s_2 + 1)\ 4^{s_1}2^{s_0} \bmod 13$$

Далее для различных значений s_1, s_0 можно вычислить член в скобках. Например для $s_1 = 0$, $s_0 = 0$, $2^{s_3s_2s_1s_0} \bmod 13 = (3s_3s_2 + 8s_3 + 2s_2 + 1) \bmod 13$. Далее, для четырёх вариантов битов s_3 и s_2, а именно, 11, 10, 01, 00, значение $2^{s_3s_2s_1s_0} \bmod 13$ можно оценить как 1, 9, 3, 1 соответственно. Таким образом, логическая функция g_0 может быть использована для представления $2^{s_3s_2s_1s_0} \bmod 13$ для $s_1 = 0$, $s_0 = 0$, рассматривая битовые значения как

$$g_0 = 8s_3\overline{s_2} + 2\overline{s_3}s_2 + 1$$

Аналогичным образом, другие функции, соответствующие s_1s_0, т. е. 01, 10, 11, могут быть получены как

$$g_1 = 4(s_2 \oplus s_3) + 2(\overline{s_3} + s_2) + s_3\overline{s_2} \quad (1.7а)$$

$$g_2 = 8(s_2 \oplus s_3) + 4(\overline{s_3} + s_2) + 2s_3\overline{s_2} \quad (1.7б)$$

$$g_3 = 8(\overline{s_3} + s_2) + 4s_3\overline{s_2} + 2(s_3 \oplus s_2) + (s_3 \oplus s_2) \quad (1.7в)$$

Обратим внимание, что логические элементы, используемые для генерации и объединения минимальных членов в функциях g_i, могут быть разделены между модулями. Например, $2^{11} \bmod 13$ можно получить из g_3 (так как $s_1 = s_0 = 1$), подставив $s_3 = 1$, $s_2 = 0$ как 7. Архитектура состоит из подачи входной мощности «i», для которой требуется $2^i \bmod 13$. Два младших разряда i, а именно, x_1, x_0 используются для выбора полубайтового вывода с помощью четырёх мультиплексоров 4:1 остатка, соответствующего функции g_j, зависящей от значений s_3 и s_2-бит. Таким образом, для каждой степени 2 остаток будет выбран с помощью набора мультиплексоров, и все эти остатки нужно будет добавить по модулю 13 в дерево сумматоров по модулю, чтобы получить окончательный остаток. Полностью параллельная архитектура или последовательные параллельные архитектуры могут использоваться для достижения компромиссов между используемыми аппаратными ресурсами и временем. Премкумар, Анг и Лэй [16] позже расширили этот метод, чтобы

уменьшить используемое аппаратное обеспечение, используя преимущества периодических свойств модулей, так что на первом этапе будет получено слово длины, равное периоду модуля, а затем может быть использован метод модульного возведения в степень.

1.2.4 Прямое преобразование для FPGA

Прямые преобразователи для набора модулей $\{2^n - 1, 2^n, 2^n + 1\}$ были рассмотрены несколькими авторами. Общая архитектура поиска остатков по модулям $(2^n - 1)$ и $(2^n + 1)$ была впервые предложена Би и Джонсом [17]. Учитывая $3n$-битное двоичное слово $W = A2^{2n} + B2^n + C$, где A, B и C – n-битные слова, $W \bmod (2^n - 1) = (A + B + C) \bmod (2^n - 1)$ и $W \bmod 2^n + 1 = (A - B + C) \bmod (2^n + 1)$. Би и Джонс предлагают сначала найти $S = A + C$, а затем вычислить $(S + B) \bmod (2^n - 1)$ или $(S - B) \bmod (2^n + 1)$ на втором этапе. На третьем этапе выполняет сокращение по модулю m_1 или m_3, используя перенос или заимствование из второго этапа. Таким образом, для каждого из генераторов остатков по модулю $(2^n - 1)$ и $(2^n + 1)$ потребуется три n-разрядных сумматора.

Пурбигараз и Ясин [18] предложили общую архитектуру для вычисления как остатков по модулю $(2^n - 1)$, так и по модулю $(2^n + 1)$ для СОК с большим динамическим диапазоном. Они суммируют k чётных n-битных блока и k нечётных n-битных блока по отдельности, используя сумматор с несколькими операндами для получения суммирующих векторов S_e, S_o, и векторов переноса C_e и C_o из $(n + \beta)$ битов, где $\beta = \log_2 k$. Далее, S_o и C_o могут быть сложены или вычтены из $S_e + C_e$ в двухуровневом сумматоре. Далее, $(m + \beta + 1)$-разрядные слова переноса и суммирования могут быть разделены на m-разрядные слова с младшего бита и $(\beta + 1)$-разрядные слова со старшего бита. Оба они могут быть сложены для получения остатка по модулю $(2^n - 1)$ или старшие биты могут быть вычтены из младших для получения остатка по модулю $(2^n + 1)$. Этот

метод был применён к набору модулей $\{2^n - 1, 2^n, 2^n + 1\}$. Сложность данного метода составляет $O(2n)$.

Концепция совместного аппаратного обеспечения была расширена Пиестраком [19] для нескольких модулей и Скавантзо и Абдалла [20] для сопряжённых модулей (пары модулей вида $2^a - 1$, $2^a + 1$). Пиестрак предположил, что модули, имеющие общий множитель периодов или полупериодов, могут воспользоваться преимуществами совместного использования. Например, рассмотрим два модуля 7 и 73. Периоды этих чисел равны 3 и 9 соответственно. Рассмотрим прямое преобразование 32-разрядного входного слова. На первом этапе можно взять 9-битные блоки данного 32-битного слова и суммировать, чтобы получить 9-битное слово. Затем преобразователь для модуля 7 может сложить три 3-битных блока и получить остаток по модулю 7, тогда как преобразователь для модуля 73 может найти остаток 9-битного слова по модулю 73. В этом случае требуется меньше аппаратных ресурсов по сравнению с использованием двух отдельных преобразователей для модуля 7 и модуля 73.

Метод может быть распространён на случай, когда период и полупериод одинаковы. Например, для модулей 5 и 17 можно привести период $P(5) = HP(17) = 4$, где HP означает полупериод, а P – период. Очевидно, что первый этап берёт 8-битные блоки входного 32-битного слова, так как $P(17) = 8$ и с помощью сумматора получает 8-битную сумму и векторы переноса. Они рассматриваются как два 4-битных блока и подаются рядом с генераторами остатков по модулям 5 и 17.

Можно комбинировать генераторы для модулей с различными полупериодами, в случае, когда НОК является одним из этих полупериодов. Рассмотрим модули 3, 5 и 17, полупериоды которых равны 1, 2 и 4 соответственно. Рассмотрим 32-битное входное двоичное слово, первый этап вычисляет из четырёх 8-битных блоков значение по модулю 255 путём сложения, 8-битную сумму и вектор переноса. Затем эти векторы подаются в

генератор остатков по модулю 17 и генератор остатков по модулю 15. Генератор остатков по модулю 15, в свою очередь, подаётся на генераторы остатков по модулям 3 и 5. С помощью этого метода можно сохранить несколько полных сумматоров. Например, для модулей 5, 7, 9 и 13, для прямого преобразования 32-разрядного входного двоичного слова, используя четыре отдельных генератора остатков, нам нужно 114 полных сумматоров, тогда как в общем оборудовании нам нужно только 66 полных сумматоров.

1.2.5 Модули специального вида $(2^n \pm k)$

Матутино, Петтенги, Чавес и Соуза [21, 22] описали преобразование двоичного кода в СОК для модулей типа $2^n \pm k$ для набора из четырёх модулей $\{2^n - 1, 2^n + 1, 2^n - 3, 2^n + 3\}$ с динамическим диапазоном $4n$ бит. Данное $4n$-разрядное двоичное слово можно рассматривать как четыре n-разрядных блока W_3, W_2, W_1 и W_0, дающие в случае модуля $2^n - k$,

$$W_{2^n} - k = |W_3 k^3 + W_2 k^2 + W_1 k + W_0|_{2^n - k} \tag{1.8}$$

так как $2^n \bmod (2^n - k) = k$.

Следовательно, используя три умножителя ($n \times p$, $n \times 2p$ и $n \times n$) для умножения W_3, W_2 и W_1 соответственно на k^3, k^2 и k, сокращение может осуществляться поэтапно. Суммирование в (1.8) даёт $(2n + 2)$-битное слово, которое снова рассматривается как три n-битных блока и далее сводится к $(n + p + 1)$-битному слову с использованием двух умножителей ($2 \times 2p$, $n \times p$), за которыми следует сумматор, где $p = \lceil \log_2 k \rceil$. Еще два этапа уменьшают длину слова с $(n + p + 1)$ бит до $(n + 2)$ бит и $(n + 2)$ бит до n битов, используя умножитель $(p + 1) \times p$ и умножитель $p \times p$, а также сумматор по модулю.

В случае модуля $(2^n + k)$ вычисление выполняется следующим образом:

$$W_{2^n} + k = |-W_3 k^3 + W_2 k^2 - W_1 k + W_0|_{2^n + k} \tag{1.9a}$$

так как $2^n \bmod (2^n + k) = -k$ и $2^{3n} \bmod (2^n + k) = -k^3$. Обратим внимание, что (1.9a) можно переписать следующим образом:

$$W_{2^n} + k = |\overline{W_3}k^3 + W_2k^2 + \overline{W_1}k + W_0 + c|_{2^n+k} \quad (1.9б)$$

где

$$c = \left(k^3(k+1) + 3k(k+1)\right) \bmod (2^n + k).$$

Обратим внимание, что из-за промежуточных шагов сокращения для уменьшения $(2n+2)$-битного слова до $(n+p+1)$ битов и следующих $(n+p+1)$ битов до n битов поправочный коэффициент равен $c = k^3(k+1) + 3k(k+1)$. Преобразователь для модуля $(2^n + k)$ нуждается в трёх ступенях, тогда как для модуля $(2^n - k)$ требуется четыре ступени. Матутино и др. [22] также предлагают реализацию умножителя путём сложения только сдвинутых версий входных данных вместо использования блока аппаратного умножения.

1.3 Анализ методов перевода чисел из СОК в ПСС

Эффективность реализации алгоритмов обнаружения, локализации и исправления ошибок зависит вычислительной сложности алгоритма перевода чисел из СОК в ПСС. Выбор набора модулей в СОК определяется скоростью преобразования СОК в двоичный код для эффективного выполнения таких операций, как сравнение, масштабирование, обнаружение знаков и исправление ошибок. Рассмотрим различные методы, как основанные на ПЗУ, так и не основанные на ПЗУ. Количество выбираемых модулей определяется желаемым динамическим диапазоном, длиной слова модулей и простотой преобразования СОК в двоичный код. Существует два основных классических подхода к преобразованию числа из СОК в двоичную форму. Они основаны на Китайской теореме об остатках (КТО) и обобщённой позиционной системе счисления (ОПСС) [23]. Несколько новых приёмов были введены в последнее время, такие как Новая КТО-I, Новая КТО-II, КТО для ОПСС, фи-функция Эйлера, функция ядра и диагональная функция. Рассмотрим эти методы подробнее.

1.3.1 Китайская теорема об остатках

Двоичное число X, соответствующее заданным остаткам $(x_1, x_2, x_3, \ldots, x_n)$ в СОК $\{m_1, m_2, m_3, \ldots m_n\}$ может быть получено с помощью КТО в виде

$$X = \left(\sum_{i=1}^{n} x_i \cdot \left| \frac{1}{M_i} \right|_{m_i} \cdot M_i \right) \bmod M \tag{1.10}$$

где $M_i = M/m_i$ для $i = 1,2, \ldots, n$ и $M = M_1 M_2 \ldots M_n$. Заметим, что далее мы обозначим $p \bmod q$ как $|p|_q$. Величины $\left| \frac{1}{M_j} \right|_{m_j}$ известны как обратные числа $M_j \bmod m_j$, определенные таким образом, что $M_j \cdot \left| \frac{1}{M_j} \right|_{m_j} \equiv 1 \bmod m_j$. Сумма в (1.10) может быть намного больше M, а сокращение по модулю M для получения X является громоздким процессом. Преимущество КТО заключается в том, что взвешивание остатков x_i может производиться параллельно и результаты суммируются с последующим восстановлением по модулю M. Архитектура обратного преобразования с использованием КТО представлена на рисунке 1.1.

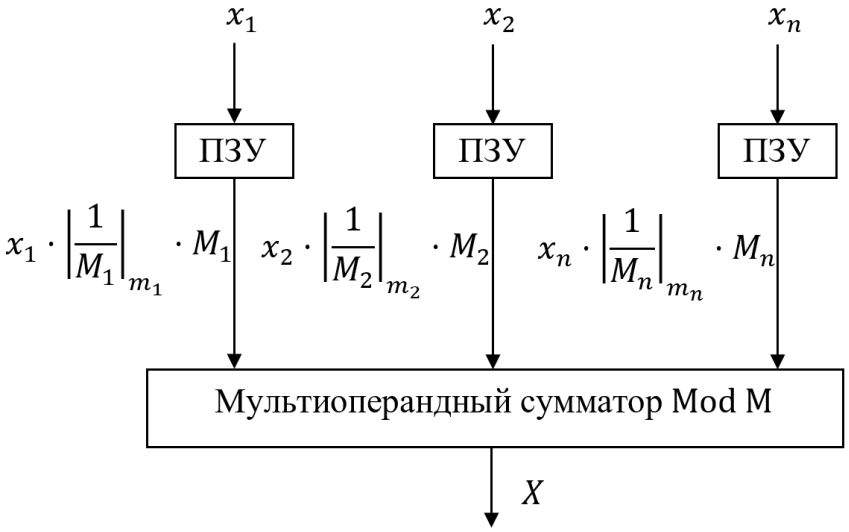

Рисунок 1.1 – Архитектура для реализации КТО

Пример 1.1. Используя КТО, найти двоичное число, соответствующее остаткам (1,2,3) в СОК с модулями {3,5,7}.

Обратим внимание, что $M = 105$, $M_1 = 35$, $M_2 = 21$ и $M_3 = 15$. Таким образом, мы имеем $(1/M_1) \bmod 3 = 2$, $(1/M_2) \bmod 5 = 1$, $(1/M_3) \bmod 7 = 1$.

По формуле (1.10) получаем:

$$X = [1 \cdot 2 \cdot 35 + 2 \cdot 1 \cdot 21 + 3 \cdot 1 \cdot 15] \bmod 105 = (70 + 42 + 45) \bmod 105 = 52.$$

Можно отметить, что в этом примере для получение итогового результата необходимо одно вычитание 105, т.е. ранг числа X равен 1.

КТО может быть эффективно использована в случае трёх- и четырёхмодульных наборов СОК, например $\{2^n - 1, 2^n, 2^n + 1\}$, $\{2^n - 1, 2^n, 2^n + 1, 2^{2n} + 1\}$ и $\{2^{2n} - 1, 2^n, 2^{2n} + 1\}$, $\{2^n - 1, 2^n, 2^{n+1} - 1\}$, $\{2^n - 1, 2^n, 2^{n-1} - 1\}$, поскольку n битов декодированного числа X доступны непосредственно в виде остатка, соответствующего модулю 2^n, и сокращение по модулю, необходимое в конце по отношению к произведению оставшихся модулей, может быть эффективно реализовано в случае первых трёх наборов модулей. Для некоторых из этих наборов модулей модули имеют широкую длину в диапазоне от n до $2n$ бит, что может быть недостатком, поскольку больший модуль определяет время цикла команд процессора СОК. Для общих наборов модулей КТО может потребовать использования сложного аппаратного обеспечения нахождения остатка по модулю и требует реализации на основе ПЗУ для получения значений $x'_i = x_i \cdot \left|\frac{1}{M_i}\right|_{m_i} \cdot M_i$ и множителей для вычисления суммы x'_i.

Далее мы подробно рассмотрим преобразование СОК в двоичный код для набора модулей $\{2^n - 1, 2^n, 2^n + 1\}$ на основе КТО с учётом огромного внимания, уделяемого в литературе [17,24-42]. Для этого набора модулей, обозначающего $m_1 = 2^n - 1$, $m_2 = 2^n$ и $m_3 = 2^n + 1$, имеем $M = 2^n(2^{2n} - 1)$, $M_1 = 2^n(2^n + 1)$, $M_2 = 2^{2n} - 1$, $M_3 = 2^n(2^n - 1)$ и $(1/M_1) \bmod (2^n - 1) = 2^{n-1}$, $(1/M_2) \bmod (2^n) = -1$, $(1/M_3) \bmod (2^n + 1) = (2^{n-1} + 1)$. Таким образом, используя КТО [25], из (1.10) можно получить декодированное число в виде

$$X = [2^n(2^n+1)2^{n-1}x_1 - (2^{2n}-1)x_2 + 2^n(2^n-1)(2^{n-1}+1)x_3] \bmod \left(2^n(2^{2n}-1)\right) = Y2^n + x_2 \quad (1.11)$$

Поскольку мы знаем младшие биты X как x_2, мы можем получить $2n$ старших бит X путём вычисления

$$Y = \left(\frac{X - x_2}{2^n}\right) \bmod (2^{2n}-1) \quad (1.12)$$

Из (1.11) и (1.12) имеем

$$Y = [(2^n+1)2^{n-1}x_1 - 2^n x_2 + (2^n+1)2^{n-1}x_3 - x_3] \bmod (2^{2n}-1) \quad (1.13)$$

Интересно, что вычисление Y включает суммирование четырёх членов, которые могут быть легко найдены с помощью битовых манипуляций с x_1, x_2 и x_3 (циклический сдвиг и обратное кодирование) благодаря задействованным операциям $\bmod (2^{2n}-1)$ [25, 29, 30]. Рассмотрим, например, первое слагаемое $A = [(2^n+1)2^{n-1}x_1] \bmod (2^{2n}-1) = (2^{2n-1}x_1 + 2^{n-1}x_1) \bmod (2^{2n}-1)$. Запишем x_1 как n-разрядное число, $x_{1,n-1}x_{1,n-2}x_{1,n-3} \dots x_{1,2}x_{1,1}x_{1,0}$, имеем

$$A = x_{1,0}x_{1,n-1}x_{1,n-2}x_{1,n-3} \dots x_{1,2}x_{1,1}x_{1,0}x_{1,n-1}x_{1,n-2}x_{1,n-3} \dots x_{1,2}x_{1,1} \quad (1.14)$$

Второе слагаемое $B = (-2^n x_2) \bmod (2^{2n}-1)$ можно записать в виде

$$B = \bar{x}_{2,n-1}\bar{x}_{2,n-2} \dots \bar{x}_{2,2}\bar{x}_{2,1}\bar{x}_{2,0}\underbrace{111 \dots 111}_{n\ \text{бит}} \quad (1.15)$$

где черта указывает на обратный код к битам (инвертированные биты).

Третий член немного задействован, так как x_3 имеет $(n+1)$-битную ширину. Используем тот факт, что когда $x_{3,n}$ равно 1,то неизменно $x_{3,0}$ равно 0. Продолжая аналогичным образом, $\left((2^n+1)2^{n-1}x_3\right) \bmod (2^{2n}-1)$ можно получить в виде

$$C = \left(x_{3,n} + x_{3,0}\right)x_{3,n-1} \dots x_{3,2}x_{3,1}\left(x_{3,n} + x_{3,0}\right)x_{3,n-1} \dots x_{3,1} \quad (1.16а)$$

и

$$D = (-x_3)_{2^{2n}-1} = \underbrace{111 \dots 111}_{n-1\ \text{бит}}\overline{x_{3,n}x_{3,n-1}}\ \overline{x_{3,n-2}} \dots \overline{x_{3,0}} \quad (1.16б)$$

Пестрак [29] предложил складывать четыре слова A, B, C и D, приведённые в (1.14), (1.15), (1.16а) и (1.16б). Дхуркадас [30] предложил переписать три слова B, C и D, чтобы получить два новых слова

$$E = \bar{x}_{2,n-1}\bar{x}_{2,n-2} \dots \bar{x}_{2,1}\bar{x}_{2,0}\overline{x_{3,n} + x_{3,n-1}}\bar{x}_{3,n-2} \dots \bar{x}_{3,0} \quad (1.17а)$$

$$F = \left(x_{3,n} + x_{3,0}\right)x_{3,n-1} \dots x_{3,2}x_{3,1}x_{3,0}x_{3,n-1} \dots x_{3,1} \quad (1.17б)$$

Таким образом, три слова, приведённые в (1.14), (1.17а) и (1.17б), необходимо суммировать. За последние два десятилетия было сделано несколько улучшений путём изучения битовой структуры трёх операндов для сокращения времени сложения [31-40].

Преобразователи СОК в бинарные данные для СОК, использующей модули вида $(2^{2n} - 1)$, могут использовать преимущества основных свойств:

1) $-A \bmod (2^n - 1)$ – это обратный код к A.
2) $(2^x A) \bmod (2^n - 1)$ получается битовым сдвигом влево A на x бит, где A – это n-разрядное целое число.
3) Логику можно упростить, заметив, что полные сумматоры с константой «1» в качестве входных данных могут быть заменены парой двухвходовых элементов: исключающее ИЛИ с инверсией и ИЛИ. Аналогично, полные сумматоры с одним входом «0» могут быть заменены парами элементов: исключающее ИЛИ и И. Заметим также, что полный сумматор с одним входом «0» и одним входом «1» может быть сведён к простому обратному преобразователю.

Ван и др. [33] предложили переписать $(2^n + 1)2^{n-1}x_3 - x_3 - 2^n x_2$ в (1.13) как

$$\left|\overline{x_{3,0}}\hat{x}_{3,n-1} \dots \hat{x}_{3,0}\overline{x_{3,n-1}} \dots \overline{x_{3,1}} + x_{2,n-1}\overline{x_{2,n-2}} \dots \overline{x_{2,0}}\ \overline{x_{2,n-1}}\ \underbrace{x_{2,n-1}x_{2,n-1}x_{2,n-1}}_{n-1\ \text{бит}}\right|_{2^{2n}-1}$$

где $\hat{x}_{3,i} = x_{3,i} \vee x_{3,n}$, $i = 0,1, \dots, n-1$ и $\vee$ указывает на логическое ИЛИ. Конвей и Нельсон [32] предложили преобразователь СОК в двоичные данные на основе расширения КТО (1.11), динамический диапазон которого меньше $2^n(2^{2n} - 1)$ на $2^n(2^n - 2) - 1$. Они переписывают выражение для КТО в (1.13) как $X =$

$D_2 2^{2n} + D_1 2^n + D_0$, так что верхние и средние n бит могут быть вычислены с помощью n-разрядного аппаратного обеспечения. Однако между этими двумя n-разрядными вычислениями существует прямая и обратная зависимость.

Галлахер и др. [39] рассмотрели уравнения для остатков искомого $3n$-разрядного выходного слова $X = D_2 2^{2n} + D_1 2^n + D_0$, соответствующие трём модулям, заданным в виде

$$(D_2 + D_1 + D_0) \bmod (2^k - 1) = x_1, (D_2 - D_1 + D_0) \bmod (2^k + 1) = x_3, \text{ и } D_0 = x_2 \tag{1.18а}$$

Перепишем первые два выражения следующим образом

$$x_1 = (D_2 + D_1 + D_0) - m(2^k - 1), x_3 = (D_2 - D_1 + D_0) - n(2^k + 1) \tag{1.18б}$$

Решая для D_1 и D_2, мы имеем

$$D_1 = X_1 + L_1 = \frac{x_1 - x_3}{2} + \frac{m(2^k - 1)}{2} - \frac{n(2^k + 1)}{2} \tag{1.18в}$$

$$D_2 = X_2 + L_2 = \frac{x_1 + x_3}{2} - r_2 + \frac{m(2^k - 1)}{2} + \frac{n(2^k + 1)}{2} \tag{1.18г}$$

Заметим, что m может быть $(0,1,2)$, а n может быть $(1,0,-1)$. Таким образом, авторы исследуют, какие значения m и n дадут правильный результат. Этот метод был усовершенствован в работе [40].

КТО может быть применена к другим системам СОК, имеющим три или более модулей. Различные операнды, подлежащие суммированию, могут быть легко получены с помощью битовых манипуляций, но окончательное суммирование и сокращение по модулю могут быть очень сложными. Таким образом, описанная выше трёхмодульная система, считается оптимальной. КТО была применена к другим наборам модулей $\{2^{2n}, 2^{2n} - 1, 2^{2n} + 1\}$ [43, 44], $\{2^n, 2^n - 1, 2^{n+1} - 1\}$ [45], $\{2^k, 2^k - 1, 2^{k-1} - 1\}$ [46] и $\{2^n + 1, 2^{n+k}, 2^n - 1\}$ [42].

Чавес и Соуза [42] предложили набор модулей $\{2^n + 1, 2^{n+k}, 2^n - 1\}$ с переменной k такой, что $0 \le k \le n$. КТО может быть использована для

декодирования числа, соответствующего заданным остаткам. В этом случае также обратные числа, необходимые в КТО, очень просты и задаются в виде

$$\left(\frac{1}{M_1}\right)_{m_1} = -2^{n-k-1}, \left(\frac{1}{M_1}\right)_{m_2} = -1, \left(\frac{1}{M_3}\right)_{m_3} = 2^{n-k-1} \tag{1.19}$$

где $m_1 = 2^n + 1$, $m_2 = 2^{n+k}$, $m_3 = 2^n - 1$.

Следовательно, аналогично случаю набора модулей $\{2^n - 1, 2^n, 2^n + 1\}$, используя КТО, обратное преобразование может быть осуществлено путём отображения остатков в $2n$-разрядные слова и сложения с помощью сумматора по модулю $(2^{2n} - 1)$. Хиасат и Свейдан [47] независимо рассмотрели этот случай при $k = n$, т. е. множество модулей $\{2^{2n}, 2^n - 1, 2^n + 1\}$.

1.3.2 Приближенный метод

Содерстранд и др. [48] предложили вычисление взвешенной суммы в (1.10), масштабируемой на $2/M$. Дроби могут быть представлены в виде слов с одним целочисленным битом и несколькими дробными битами и просуммированы. Кратное 2 в целой части результирующей суммы может быть отброшено. Целочисленная часть будет передавать информацию о знаке числа: если целая часть равна 0, число считают положительным, если 1 – отрицательным. Однако Ву [49] указал, что точность этих дробей должна быть достаточной, чтобы дать определенное указание на знак. Учитывая выражение КТО, получим

$$X = \left[\sum_{i=1}^{n} M_i \left|\frac{1}{M_i}\right|_{m_i} x_i\right]_M \tag{1.20а}$$

Умножив обе стороны (1.20а) на $2/M$, получим масштабируемое значение

$$X_s = \left|\sum_{i=1}^{n} \frac{2}{m_i} \left|\frac{1}{M_i}\right|_{m_i} x_i\right|_2 = \left|\sum_{i=1}^{n} u_i\right|_2 \tag{1.20б}$$

Учитывая, что погрешность обусловлена конечным представлением битов в u_i как e_i, так что $0 \le e_i \le 2^{-t}$, где $t + 1$ битов используется для представления

каждого u_i, можно показать, что суммарная погрешность $e < 2/M$ для чётного M и $e < 1/M$ для нечётного M или $n2^{-t} \leq 2/M$ для чётного M и $n2^{-t} \leq 1/M$ для нечётного M.

Пример проиллюстрирует идею обнаружения знака с помощью реализации КТО по Ву.

Пример 1.2. Рассмотрим множество модулей $\{11,13,15,16\}$ и два случая, соответствующие действительным числам $+1$ и -1. Используем технику Ву для обнаружения знаков.

Остатки составляют $(1,1,1,1)$ и $(10,12,14,15)$ соответственно. Выражение (1.20б) становится в первом случае

$$X_s = \frac{16}{11} + \frac{2}{13} + \frac{4}{15} + \frac{2}{16}$$

Сложив первые два члена и вторые два члена и представив их в дробном виде, получим

$$v_1 = u_1 + u_2 = \frac{16}{11} + \frac{2}{13} = \frac{230}{143} \text{ и } v_2 = u_3 + u_4 = \frac{4}{15} + \frac{2}{16} = \frac{94}{240}.$$

Дробное представление для v_1, v_2 и X_s равно

$$\hat{v}_1 = 1.1001101111000000, \hat{v}_2 = 0.0110010001000101,$$
$$\hat{X}_s = 0.000000000000101$$

Для случая, соответствующего -1, имеем, с другой стороны, $\hat{X}_s = 1.111111111100101$. Заметим, что после предложения Содерстранда и других [48] об использовании 11 бит для дробной части, сумма будет равна 0.0000000000, что не даёт правильного знака.

Кардарелли и др. [50] представили систолическую архитектуру для масштабированного преобразования остатка в двоичный код. Заметим, что он основан на масштабировании результата расширения КТО для N-модулей СОК на M для получения

$$\frac{X}{M} = \langle \sum_{i=1}^{N} \frac{x'_i}{m_i} \rangle_1 \tag{1.21}$$

где $x'_i = \left| \left| \frac{1}{M_i} \right|_{m_i} x_i \right|_{m_i}$. Обратим внимание, что x'_i/m_i – это все дроби. Биты, представляющие x'_i/m_i, могут быть получены с помощью итерационного процесса. Полученные N слов могут быть добавлены с помощью дерева сумматоров. Дробь x_i/m_i можно записать в виде

$$\frac{x'_i}{m_i} = \frac{H_1}{2} + \frac{H_2}{2^2} + \frac{H_3}{2^3} + \cdots + \frac{H_L}{2^L} + \frac{\varepsilon_L}{m2^L} \tag{1.22}$$

где ε_L – ошибка, вызванная усечением x'_i/m_i, а (1.22) – разложение по основанию 2 (x'_i/m_i) и $L = \lceil \log_2(NM) \rceil + 1$. Умножив (1.22) на $2^L m_i$ и взяв по $\mathrm{mod}\ 2$ с обеих сторон, мы получим

$$|H_L|_2 = H_L = |-\varepsilon_L m^{-1}|_2 \tag{1.23}$$

где m^{-1} – мультипликативная обратная величина $m \bmod 2$. Чтобы вычислить член H_{L-1}, нам нужно умножить (1.22) на 2^{L-1} и взять $\mathrm{mod}\ 2$ с обеих сторон и использовать уже полученные значения ε_L и H_L. Обратим внимание, что $\varepsilon_L = a_1$ определяется как $x_i \times 2^L = a_1 + a_2 m_i$ и $a_2 = H_1 2^{L-1} + H_2 2^{L-2} + H_3 2^{L-3} + \cdots + H_L$. Аналогичным образом можно получить и другие значения H_i.

Например, для $m = 5$, остатка $x = 1$ и набора модулей $\{3,5,7\}$, $N = 3$ и $M = 105$, $L = \lceil \log_2(NM) \rceil + 1 = 10$. Таким образом, имеем $2^L = 1024$, $a_1 = 4$, $a_2 = 204$ и $1 \times 1024 = 4 + 204 \times 5$. Таким образом, $\varepsilon_L = 4$ и мы можем вычислить итеративно $a_2 = 204$ бит за битом.

Димауро и др. [51] ввели понятие «фактор-функция» (фи-функция Эйлера) для выполнения преобразования СОК в двоичный код. Этот метод, обозначаемый как метод фактор-функции (QFT), похож на метод Андрароса и Ахмада [25]. В этом методе один модуль m_j имеет вид 2^k. Рассмотрим множество модулей $\{m_1, m_2, \dots, m_j\}$. Фактор-функция определяется как

$$QF_j(|X|_M) = \left\lfloor \frac{|X|_M}{m_j} \right\rfloor = \left| \sum_{i=1}^{N} b_i x_i \right|_{M_j} \tag{1.24}$$

Обратим внимание, что $M = \prod_{i=1}^{j} m_i$, $M_j = M/m_j$ и b_i определяется как

$$b_i = -\left|\frac{1}{m_j}\right|_{M_j} \text{ если } i = j \tag{1.25а}$$

и

$$b_i = \left|M_i \left|\frac{1}{SQ_j}\right|_{M_j} \left|\frac{1}{m_j}\right|_{M_j}\right|_{M_j} \text{ для } i = 1, \dots, N,\ i \neq j \tag{1.25б}$$

Отметим, что сумма частных SQ_j определяется как $SQ_j = \sum_{\substack{i=1 \\ i \neq j}}^{N} M_i$. Таким образом, процедура преобразования СОК в двоичный код с использованием (1.24) аналогична вычислению КТО. Отметим, что метод Андрароса и Ахмада [25] реализует частное через КТО. Сложение QF_j с остатком, соответствующим 2^k, а именно r_j, в качестве наименее значащих k бит, приведёт к окончательному декодированному числу $|X|_M$. Рассмотрим пример.

Пример 1.3. Рассмотрим множество модулей $\{5,7,9,11,13,16\}$ и остатки $(2,3,5,0,4,11)$. Получить декодированное число с помощью фактор-функции.

Обратим внимание, что $m_j = 16$ и $M_j = 45{,}045$. Далее имеем, что $\left|\frac{1}{m_j}\right|_{M_j} = 8446$ и $SQ_j = (5 \times 7 \times 9 \times 11 + 5 \times 7 \times 9 \times 13 + 5 \times 9 \times 11 \times 13 + 5 \times 7 \times 11 \times 13 + 7 \times 9 \times 11 \times 13) = 28{,}009$. Отсюда следует, что $\left|\frac{1}{SQ_j}\right|_{M_j} = 16{,}174$. Далее, используя (1.25), b_i можно оценить как $b(1) = 36{,}036$, $b(2) = 12{,}870$, $b(3) = 20{,}020$, $b(4) = 12{,}285$, $b(5) = 17{,}325$ и $b(6) = 36{,}599$. Например, $b(2) = \left|M_2 \left|\frac{1}{SQ_j}\right|_{M_j} \left|\frac{1}{m_j}\right|_{M_j}\right|_{M_j} = |(5 \times 9 \times 11 \times 13) \times 16{,}174 \times 8446|_{45{,}045}$.

Далее, двоичное число, соответствующее заданным остаткам, может быть вычислено как

$$(2 \times 36{,}036 + 3 \times 12{,}870 + 5 \times 20{,}020 + 0 \times 12{,}285 + 4 \times 17{,}325 + 11 \times 36{,}599) \bmod 45{,}045 = 6996,$$

которое в двоичной форме равно $1\,101\,101\,010\,100$. Складывая это с остатком 11 в двоичной форме 1011, соответствующим модулю 16, мы получаем декодированное число, равное $111{,}947$.

Димауро и др. [51] отметили, что поскольку SQ может быть большим, альтернативный метод, когда первый уровень рассматривает исходный набор модулей как два подмножества, для остатков которых можно параллельно выполнить обратное преобразование. Далее можно оценить фактор-функцию. Это уменьшит величину SQ и, следовательно, приведёт к упрощению аппаратного обеспечения. Например, для того же множества модулей, рассмотренного выше, можно рассмотреть подмножества $\{5{,}9{,}16\}$ и $\{7{,}11{,}13\}$, дающие $SQ = 1721$ по сравнению с $28{,}009$ в предыдущем случае.

Ким и др. [52] предложили метод преобразования СОК в двоичный код с округлённой компенсацией ошибок. В этом методе результат КТО умножается на $\frac{2^d}{M}$, где M – произведение модулей, а d – решающий параметр длины выходного слова:

$$X_s = \left[X \times \frac{2^d}{M}\right] = \left[\sum_{i=1}^{L} M_i \left|\frac{1}{M_i} x_i\right|_{m_i} \frac{2^d}{M}\right] - \alpha 2^d \tag{1.26}$$

Очевидно, что операция $\bmod M$ преобразуется в операцию $\bmod 2^d$, в которой мы игнорируем сгенерированный перенос. Квадратные скобки [ ] означают округление до ближайшего целого числа. Ким и др. предлагают рассматривать два условия одновременно, которые будут добавлены с помощью дерева сумматоров. Например, для четырёх модулей СОК,

$$X_s = |f(x_1 + x_2) + f(x_3 + x_4)|_{2^d} \tag{1.27а}$$

где

$$f(x_1 + x_2) = \left|\left[M_1 \left(\frac{1}{M_1} x_1\right)_{m_1} \frac{2^d}{M} + M_2 \left(\frac{1}{M_2} x_2\right)_{m_1} \frac{2^d}{M}\right]\right|_{2^d} \tag{1.27б}$$

После каждого округления $f(x_1, x_2)$ представляет ошибку округления $1/2$ младшего бита, максимальная ошибка при вычислении X_s составляет 1 младший

бит. Ким и др. предлагают вычислить погрешность, обусловленную округлением, используя набор ближайших оценок погрешности, например $\pm 1/3, \pm 2/3, \pm 1/5, \pm 2/5, \dots, \pm 4/5$ и т. д. и вычислить $\hat{\delta} = \sum_{i=1}^{N} \delta_i$, где $\delta_i = x_i - [x_i]$, где $[x_i]$ – округлённое действительное число. Они считываются из ПЗУ для $f(x_1, x_2)$ и для $f(x_3, x_4)$ и суммируются вместе со знаком, а результат складывается с приближенными значениями, полученными ранее. Они показали, что содержание ПЗУ должно быть получено с помощью компьютерного моделирования, чтобы найти максимальную ошибку масштабирования для выбранного d, и таким образом можно избежать отсутствия упорядоченных выходов данных, иначе происходящих без компенсации ошибки округления.

Далее рассмотрим модификацию Китайской теоремы об остатках с применением дробных величин, которую в дальнейшем будем обозначать КТОд. Если обе части формулы $X = \left|\sum_{i=1}^{n} \left|M_i^{-1}\right|_{m_i} M_i x_i\right|_M$ разделить на M, то получим соотношение

$$\bar{X} = \frac{X}{M} = \left|\sum_{i=1}^{n} \frac{\left|M_i^{-1}\right|_{m_i}}{m_i} x_i\right|_1 = \left|\sum_{i=1}^{n} k_i\, x_i\right|_1 \tag{1.28}$$

где операция $|\cdot|_1$ означает отбрасывание целой части числа, а числа

$$k_i = \frac{\left|M_i^{-1}\right|_{m_i}}{m_i}, i = 1,2, \dots, n \tag{1.29}$$

являются константами СОК и могут быть рассчитаны заранее. При этом значение каждой суммы будет находиться в промежутке [0,1), что дает достаточно информации для оценки знака и величины числа, представленного в СОК.

Такой переход позволяет заменить точное число его дробной характеристикой, давая возможность регулирования точности представления в зависимости от имеющихся ресурсов и поставленной задачи. Величина $\bar{X}$ может рассматриваться как позиционная характеристика числа X, при этом число X может быть найдено по формуле

$$X = \bar{X}M. \tag{1.30}$$

Однако в случае машинных вычислений мы можем воспользоваться лишь ограниченной точностью, что требует округления или отбрасывания младших бит дроби. Оценим количество бит, позволяющих однозначно определить дробную характеристику числа X. Пусть $\overline{k_i}$ есть конечная дробь, содержащая N бит, совпадающих с первыми N битами числа k_i для всех $i = 1,2,\dots,n$. Иными словами, $\overline{k_i} = \lfloor k_i \rfloor_{2^{-N}}$, где операция $\lfloor\cdot\rfloor$ означает округление числа вниз. Приближенное значение числа $\bar{X}$ будет не больше точного, так как $k_i \geq \overline{k_i}$, $i = 1,2,\dots,n$. Точное значение числа X может быть восстановлено умножением $\bar{X}$ на M с отбрасыванием дробной части с округлением «вверх». Проведем оценку требуемой точности вычислений N, при которой значение $\bar{X}$, восстановленное с использованием $\overline{k_i}$, не будет приводить к возникновению ошибок при восстановлении точного значения X. Для этого необходимо выполнение соотношения

$$\frac{X-1}{M} < \left|\sum_{i=1}^{n} \overline{k_i} x_i\right|_1 \leq \frac{X}{M} \tag{1.31}$$

показывающего однозначность позиционной характеристики $\bar{X}$ для различных чисел СОК. Преобразование выражения (1.31) с использованием формулы (1.28) приводит к неравенству

$$0 \leq \left|\sum_{i=1}^{n} (k_i - \overline{k_i}) x_i\right|_1 < \frac{1}{M} \tag{1.32}$$

Так как $k_i - \overline{k_i} = \frac{|M_i^{-1}|_{m_i}}{m_i} - \frac{2^N |M_i^{-1}|_{m_i} - \left|2^N |M_i^{-1}|_{m_i}\right|_{m_i}}{2^N \cdot m_i} = \frac{\left|2^N |M_i^{-1}|_{m_i}\right|_{m_i}}{2^N \cdot m_i}$, то

$$\sum_{i=1}^{n} (k_i - \overline{k_i}) x_i = \sum_{i=1}^{n} \frac{\left|2^N \left|M_i^{-1}\right|_{m_i}\right|_{m_i}}{2^N \cdot m_i} x_i. \tag{1.33}$$

Учитывая, что $x_i \leq m_i - 1$, то левая часть равенства (1.33) удовлетворяет неравенству:

$$\sum_{i=1}^{n} \frac{\left|2^N \left|M_i^{-1}\right|_{m_i}\right|_{m_i}}{2^N \cdot m_i} x_i$$
$$\leq \frac{1}{2^N}\left(\sum_{i=1}^{n} \left|2^N \left|M_i^{-1}\right|_{m_i}\right|_{m_i} - \sum_{i=1}^{n} \frac{\left|2^N \left|M_i^{-1}\right|_{m_i}\right|_{m_i}}{m_i}\right) \quad (1.34)$$

Из (1.32) и (1.34) следует, что N удовлетворяет неравенству:

$$2^N \leq M\left(\sum_{i=1}^{n} \left|2^N \left|M_i^{-1}\right|_{m_i}\right|_{m_i} - \sum_{i=1}^{n} \frac{\left|2^N \left|M_i^{-1}\right|_{m_i}\right|_{m_i}}{m_i}\right)$$
$$\leq -SQ + M\sum_{i=1}^{n}(m_i - 1). \quad (1.35)$$

где $SQ = \sum_{i=1}^{n} M_i$.

Обозначив $\rho = \sum_{i=1}^{n}(m_i - 1)$, из формулы (1.35) получим, что при выборе N, равном $N = \lceil \log_2(M \cdot \rho - SQ) \rceil$, полученная оценка является уточнением оценки, полученной в работе [25], где $N = \lceil \log_2(M \cdot \rho) \rceil$.

1.3.3 Обобщённо позиционная система счисления (ОПСС)

Метод ОПСС является последовательным и включает в себя вычитание по модулю и умножение по модулю путём мультипликативных инверсий одного модуля по отношению к остальным модулям. В ОПСС декодированное число выражается как

$$B = x_1 + d_1 m_1 + d_2 m_1 m_2 + d_3 m_1 m_2 m_3 + \cdots$$
$$+ d_{j-1} m_1 m_2 m_3 \ldots m_{j-1} \quad (1.36)$$

где $0 \leq d_i \leq (m_{i+1} - 1)$ для j модулей СОК. Параметры d_i известны как цифры ОПСС.

На каждом шаге определяется одна цифра d_i ОПСС. В конце цифры ОПСС взвешиваются в соответствии с (1.36), чтобы получить окончательное

декодированное число. Нет никакой необходимости в окончательном сокращении по модулю.

Обратим внимание, что на каждом шаге остаток, соответствующий одному модулю, вычитается так, что результат точно делится на этот модуль. Умножение на мультипликативные инверсии завершает это деление. Последний шаг требует умножения больших чисел, например $z_1 m_2 m_3$ в примере с тремя модулями, и сложения результирующих произведений. Но в случае ОПСС окончательное сокращение по модулю не требуется, так как результат всегда меньше $M = m_1 m_2 m_3$. В случае большого числа модулей СОК, например $\{m_1, m_2, m_3, \dots, m_n\}$, различные мультипликативные инверсии должны быть известны априори, а также должны храниться различные произведения модулей $m_{(k-1)} m_k, m_{(k-2)} m_{(k-1)} m_k$ и т. д. Обратим внимание, что алгоритм ОПСС может быть конвейеризован. Следующий пример иллюстрирует этот метод.

Пример 1.4. Рассмотрим метод обобщённой позиционной системы счисления для нахождения десятичного числа, соответствующего остаткам (1,2,3), используя набор модулей {3,5,7}.

Эта процедура проиллюстрирована ниже:

m_3	m_2	m_1	3	5	7
x_3	x_2	x_1	1	2	3
$-x_1$	$-x_1$			-3	-3
$(x_3 - x_1) \bmod m_3$	$(x_2 - x_1) \bmod m_2$		1	4	
$\times (1/m_1) \bmod m_3$	$\times (1/m_1) \bmod m_2$		$\times 1$	$\times 3$	
y_1	d_1		1	2	
$-d_1$			-2		
$(y_1 - d_1) \bmod m_3$			2		
$\times (1/m_2) \bmod m_3$			$\times 2$		
d_2			1		

Результатом является $X = d_2(m_2 m_1) + d_1(m_1) + x_1 = 1 \times (35) + 2 \times (7) + 3 = 52$.

Хуан [53] предложил преобразователь СОК в двоичный код, в котором цифры ОПСС, соответствующие каждому остатку x_i, получаются путём табличного поиска из LUT-таблицы. В конце концов, цифры ОПСС взвешиваются с помощью множителей и суммируются в обычном конечном сумматоре для вычисления (1.36). КТО II Ванга [54] аналогична этому методу [55].

Рассмотрим, например, множество модулей $\{3,5,7\}$ и заданные остатки $(1,2,3)$. Цифры ОПСС, соответствующие трём остаткам $(1,0,0)$, $(0,2,0)$, $(0,0,3)$, равны $[2,0,0]$, $[1,1,0]$, $[1,1,3]$, где $B = d_2(35) + d_1(7) + d_0$ – расширение ОПСС. (Например, $(1,0,0)$ соответствует $70 = 2 \times 35 + 0 \times 7 + 0$ в ОПСС, что даёт цифры ОПСС как $[2,0,0]$.) Таким образом, складывая их по $\bmod m_i$ и добавив перенос в предыдущем столбце, мы получим $[1,2,3]$, что соответствует $1 \times 35 + 2 \times 7 + 3 = 52$.

ОПСС проще в случае некоторых связанных модулей степеней двух, так как различные мультипликативные инверсии, необходимые на последовательных шагах, имеют вид $\pm 2^i$, так что умножение по модулю может быть реализовано с помощью битового сдвига операндов в случае $\bmod (2^n - 1)$, дополнения определенных битов и добавления поправочного члена в случае $\bmod (2^n + 1)$ [37, 38].

Например, для случая ранее рассмотренного множества модулей $\{2^n - 1, 2^n, 2^n + 1\}$ мультипликативные инверсии выглядят следующим образом: $\left|\frac{1}{2^n+1}\right|_{2^n-1} = 2^{n-1}$, $\left|\frac{1}{2^n+1}\right|_{2^n} = -1$, $\left|\frac{1}{2^n}\right|_{2^n-1} = 1$. Таким образом, на каждом шаге ОПСС необходимы только модульные вычитания, а умножение на $2^{n-1} \bmod (2^n - 1)$ может быть реализовано левым циклическим сдвигом. Проиллюстрируем эту процедуру на примере.

Пример 1.5. Рассмотрим множество модулей $\{2^n - 1, 2^n, 2^n + 1\}$ с $n = 3$. Таким образом, СОК – это $\{7,8,9\}$. Мы хотим найти десятичное число, соответствующее остаткам $(1,2,3)$. Процедура ОПСС заключается в следующем:

$$\begin{array}{ccc} 7 & 8 & 9 \\ 1 & 2 & 3 \\ -3 & -3 & \\ 5 & 7 & \\ \times 4 & \times 1 & \\ 6 & 7 & \\ -7 & & \\ 6 & & \\ \times 1 & & \\ 6 & & \end{array}$$

Декодированное число может быть получено в виде $6 \times (8 \times 9) + 7 \times (9) + 3 = 498$. (Ряд операций умножения по модулю здесь не требуют времени вычисления, так как выполняются сдвигом.)

Заметим также, что порядок модулей для ОПСС значительно влияет на сложность вычисления и может быть изменён, чтобы уменьшить сложность. Например, если мы изменим порядок модулей на 7, 9 и 8, то потребуется умножение по модулю $(2^n + 1) =$ модулю 9.

Яссин и Мур [56] предложили выбрать набор модулей, чтобы иметь определенные мультипликативные инверсии как 1, чтобы облегчить простое обратное преобразование, используя только вычитания. Рассматривая набор модулей $\{m_1, m_2, m_3, m_4\}$ в качестве примера, мы выбираем такие модули, что V_i, которые являются постоянными предопределёнными коэффициентами, все равны 1:

$$V_1 = 1, V_2 = \left|\frac{1}{m_1}\right|_{m_2} = 1, V_3 = \left|\frac{1}{m_1 m_2}\right|_{m_3} = 1, V_4 = \left|\frac{1}{m_1 m_2 m_3}\right|_{m_4} = 1 \quad (1.37а)$$

Декодированное число может быть сначала записано в виде смешанного остатка как

$$X = |U_1 V_1|_{m_1} + m_1 |U_2 V_2|_{m_2} + m_1 m_2 |U_3 V_3|_{m_3} + m_1 m_2 m_3 |U_4 V_4|_{m_4} \quad (1.37б)$$

Обратим внимание, что U_i такие, что $|U_i V_i|_{m_i} = \gamma_i$, являются цифрами ОПСС. Это можно доказать следующим образом. Мы можем оценить остатки x_1, x_2, x_3 и x_4 из (1.37б) как

$$x_1 = |U_1V_1|_{m_1} = U_1, x_2 = |U_1 + U_2|_{m_2},$$
$$x_3 = \left|U_1 + m_1|U_2V_2|_{m_2} + U_3\right|_{m_3}, \quad (1.37в)$$
$$x_4 = \left|U_1 + m_1|U_2V_2|_{m_2} + m_1m_2|U_3V_3|_{m_3} + U_4\right|_{m_4}$$

из чего мы можем видеть, что

$$U_1 = x_1, U_2 = |x_2 - x_1|_{m_2}, U_3 = |x_3 - x_1 - m_1U_2|_{m_3},$$
$$U_4 = |x_4 - x_1 - m_1U_2 - m_1m_2U_3|_{m_4} \quad (1.37г)$$

Рассмотрим следующий пример, иллюстрирующий этот метод.

Пример 1.6. Рассмотрим множество модулей $\{m_1, m_2, m_3, m_4\} = \{127,63,50,13\}$, для которых $V_1 = V_2 = V_3 = V_4 = 1$. Найдём число, соответствующее остаткам $\{78,41,47,9\}$.

Цифры ОПСС γ_i вычисляются следующим образом:

$$\gamma_1 = U_1 = 78$$
$$\gamma_2 = U_2 = |41 - 78|_{63} = 26$$
$$\gamma_3 = U_3 = |47 - 78 - 127.26|_{50} = 17$$
$$\gamma_4 = U_4 = |9 - 78 - 127.26 - 127.63.17|_{13} = 9$$

Получаем, что декодированное число составляет $78 + 26 \times (127) + 17 \times (127 \times 63) + 9 \times (127 \times 63 \times 50) = 3{,}739{,}847$. Таким образом, количество арифметических операций уменьшается по сравнению с обычной ОПСС.

1.3.4 КТО-I, КТО-II, КТО-III

В литературе появились вариации КТО, наиболее важной из которых является новая КТО-I [54]. Используя новую КТО-I, учитывая набор модулей $\{m_1, m_2, m_3, \dots, m_n\}$, взвешенное двоичное число, соответствующее остаткам $(x_1, x_2, x_3, \dots, x_n)$, можно найти как

$$X = x_1 + m_1\big(k_1(x_2 - x_1) + k_2 m_2(x_3 - x_2) + \cdots + k_{n-1} m_2 m_3 \dots m_{n-1}(x_n - x_{n-1})\big) \bmod (m_2 m_3 \dots m_{n-1} m_n) \quad (1.38)$$

где $|k_1 m_1|_{m_2 m_3 \dots m_{n-1} m_n} = 1$, $|k_2 m_1 m_2|_{m_3 \dots m_{n-1} m_n} = 1$, …, $|k_{n-1} m_1 m_2 m_3 \dots m_{n-1}|_{m_n} = 1$.

Например, приложение к набору модулей $\{m_1, m_2, m_3\} = \{2^n - 1, 2^n, 2^n + 1\}$ даёт

$$X = x_2 + 2^n Y = \\ = x_2 + 2^n |(x_2 - x_3) + 2^{n-1}(2^n + 1)(x_1 - 2x_2 + x_3)|_{2^{2n}-1} \quad (1.39)$$

Таким образом, сначала можно найти Y и, добавив x_2 в качестве младшего бита, мы можем получить X.

Ван и др. [35] предложили вычисление Y в (1.39) как $(A + 2^n B)_{2^{2n}-1}$, где

$$A = \left\lfloor \frac{x_1 + (x_{10} \oplus x_{30})2^n + (2^n - 1 - x_3) + 2^n - 1}{2} \right\rfloor \quad (1.40а)$$

и

$$B = \left\lfloor \frac{x_1 + (x_{10} \oplus x_{30})2^n + x_3 + 2(2^n - 1 - x_2)}{2} \right\rfloor \quad (1.40б)$$

где x_{10} и x_{30} – это младшие биты x_1 и x_3 соответственно. Значение A может быть вычислено с помощью сумматора с 2 входами для получения суммы и векторов переноса A_1 и A_2. Аналогично, B можно оценить, чтобы получить сумму, векторы переноса B_1 и B_2 и разряд переноса с помощью трёхходового n-разрядного сумматора. Далее, Y может быть получен из A и B с помощью $2n$-разрядного сумматора (Преобразователя I) или n-разрядных сумматоров для уменьшения задержки распространения. Было предложено два решения для n-разрядного случая, обозначенные как Преобразователь II и Преобразователь III.

Би и Гросс [57] описали китайскую теорему об остатках для обобщённой позиционной системы счисления (КТО ОПСС) для преобразования СОК в двоичный код. Результат преобразования СОК в двоичный код может быть

вычислен в этом подходе для СОК, имеющей модули $\{m_1, m_2, \dots, m_n\}$ с остатками $(x_1, x_2, \dots, x_n)$ как

$$X = x_1 + m_1|\gamma_1 x_1 + \gamma_2 x_2|_{m_2} + m_1 m_2 \left|\left|\frac{\gamma_1 x_1 + \gamma_2 x_2 + \gamma_3 x_3}{m_2}\right]\right|_{m_3} + \cdot\cdot$$
$$+ m_1 m_2 \dots m_{n-1} \left|\left|\frac{\gamma_1 x_1 + \gamma_2 x_2 + \gamma_3 x_3 + \dots + \gamma_n x_n}{m_2 m_3 \dots m_{n-1}}\right]\right|_{m_n} \quad (1.41а)$$

где

$$\gamma_1 = \frac{M_1 \left|\frac{1}{M_1}\right|_{m_1} - 1}{m_1} \quad \text{и} \quad \gamma_i = \frac{M}{m_1 m_i} \left|\frac{1}{M_i}\right|_{m_i}. \quad (1.41б)$$

Обратим внимание, что первые два члена используют ОПСС, а другие члены используют КТО, как расширение. Преимуществом этой формулировки является возможность параллельного вычисления различных цифр ОПСС, позволяющая быстро сравнивать два числа за счёт аппаратных средств, так как многие члены в числителях для выражений для нескольких цифр ОПСС, деления на произведение модулей и получения целочисленного значения являются громоздкими, за исключением специальных модулей. Рассмотрим наглядно с помощью примера.

Пример 1.7. Мы хотим найти число, соответствующее остаткам (1,2,3,4) в СОК {3,5,7,11}. Мы можем вычислить, как в КТО, $M_1 = 385$, $M_2 = 231$, $M_3 = 165$, $M_4 = 105$ и различные $\left|\frac{1}{M_i}\right|_{m_i}$ как $\left|\frac{1}{385}\right|_3 = 1$, $\left|\frac{1}{231}\right|_5 = 1$, $\left|\frac{1}{165}\right|_7 = 2$, $\left|\frac{1}{105}\right|_{11} = 2$. Далее мы вычисляем $\gamma_1 = 128$, $\gamma_2 = 77$, $\gamma_3 = 110$, $\gamma_4 = 70$. Таким образом, X можно вычислить как

$$X = 1 + 3 \times (128 + 154)_5 + 15 \times \left|\left|\frac{128 + 154 + 330}{5}\right]\right|_7$$
$$+ 105 \times \left|\left|\frac{128 + 154 + 330 + 280}{35}\right]\right|_{11}$$
$$= 1 + 3 \times 2 + 15 \times 3 + 105 \times 3 = 367$$

Новая КТО III [58, 59] может быть использована для выполнения преобразования СОК в двоичный код, когда модули имеют общие множители.

Рассматривая два модуля m_1 и m_2 с общими множителями d и учитывая, что $m_1 > m_2$, декодированное число, соответствующее остаткам x_1 и x_2, может быть получено в виде

$$X = x_1 + m_1 \left| \left(\frac{1}{m_1/d} \right) \frac{(x_2 - x_1)}{d} \right|_{m_2/d} \tag{1.42}$$

Например, рассмотрим множество модулей $\{15,12\}$ с $d = 3$ в качестве общего множителя и заданными остатками $(5,2)$. Декодированное число может быть получено из (1.42) как

$$X = 5 + 15 \left| \left(\frac{1}{5} \right) \frac{(2-5)}{3} \right|_4 = 50.$$

1.3.5 Функции ядра

Некоторые авторы исследовали использование функции ядра для преобразования СОК в двоичный код, а также масштабирования. Рассмотрим множество модулей $\{m_1, m_2, m_3, \dots, m_k\}$. Нам нужно сначала выбрать константу $C(M)$ в качестве ядра [60-67]. Это может быть самый большой модуль в СОК или произведение двух или более модулей.

Как и в случае КТО, мы определяем различные B_i как $B_i = M_i \left| \frac{1}{M_i} \right|_{m_i}$, где $M_i = M/m_i$ и $\left| \frac{1}{M_i} \right|_{m_i}$ – мультипликативная обратная величина $M_i \bmod m_i$. Далее нам нужно вычислить веса w_i, определенные как

$$w_i = \left(\left| \frac{1}{M_i} \right|_{m_i} C(M) \right) \bmod m_i \tag{1.43а}$$

Веса также должны удовлетворять условию

$$C(M) = \sum_{i=1}^{k} M \frac{w_i}{m_i} \tag{1.43б}$$

таким образом, необходимо, чтобы некоторые веса были отрицательными. Далее веса используются для вычисления функции ядра $C(n)$ заданного числа n в виде

$$C(n) = \left(n\frac{C(M)}{M} - \sum_{i=1}^{k} \frac{w_i}{m_i} r_i \right) \bmod C(M) \tag{1.44}$$

Обратим внимание, что значения ядра $C(B_i)$, соответствующие входному B_i, можно представить из (1.44) как

$$C(B_i) = \frac{B_i\, C(M)}{M} - \frac{w_i}{m_i} \tag{1.45}$$

поскольку остаток r_i, соответствующий $B_i \bmod m_i$, равен 1, а остатки B_i, соответствующие всем другим модулям, равны нулю. Обратим внимание, что различные значения $C(B_i)$ являются константами для выбранного набора модулей. Из (1.43б) и (1.45) видно, что $C(B_i) < C(M)$, так как $B_i < M$.

Преобразование остатка в двоичную форму, соответствующее остаткам $(r_1, r_2, r_3, \dots, r_k)$, осуществляется путём первого определения функции ядра $C(n)$ данного числа n в виде

$$C(n) = \left(\sum_{i=1}^{k} r_i C(B_i) \right) \bmod C(M) = \sum_{i=1}^{k} r_i C(B_i) - \alpha C(M) \tag{1.46а}$$

где α известна как функция ранга, определяемая КТО. Обратим внимание, что (1.45) известна как КТО для функции ядра. Далее n можно вычислить путём переписывания (1.44) как

$$n = \left| \frac{M}{C(M)} \left(C(n) + \sum_{i=1}^{k} \frac{w_i}{m_i} r_i \right) \right|_M \tag{1.46б}$$

Важным свойством $C(n)$ является то, что член $n\frac{C(M)}{M}$ в (1.44) монотонно возрастает с n с некоторой погрешностью из-за второго члена в (1.44), хотя выбор некоторых весовых коэффициентов w_i, определенных в (1.43а) в качестве отрицательных чисел, уменьшает погрешность (см. (1.44)). Следовательно, точное сравнение двух чисел с использованием основной функции ядра или обнаружения знака затруднительно.

Основное преимущество, заявленное для использования функции ядра, состоит в том, что константы $C(B_i)$, участвующие в вычислении функции ядра

после (1.45), малы, поскольку они меньше $C(M)$. Однако, чтобы упростить или избежать громоздкого деления на $C(M)$, необходимого в (1.46а), было предложено выбрать $C(M)$ в качестве степени двух или одного модуля или произведения двух или более модулей.

Следующий пример иллюстрирует процедуру вычисления ядра и обратного преобразования с использованием функции ядра.

Пример 1.8. Рассмотрим множество модулей {3,5,7,11}. Выберем $C(M) = 11$. Тогда $M = 1155$, $M_1 = 385$, $M_2 = 241$, $M_3 = 165$ и $M_4 = 105$. Значениями $\left|\frac{1}{M_i}\right|_{m_i}$ являются 1, 1, 2, 2. Тогда $B_i = M_i \left|\frac{1}{M_i}\right|_{m_i}$ равны 385, 241, 330, 210. Значение w_i можно найти как -1, 1, 1, 0. Далее находим $C(B_1) = 4$, $C(B_2) = 2$, $C(B_3) = 3$ и $C(B_4) = 2$. Рассмотрим остатки (1,2,3,8). Тогда находим $C(n) = (1 \times 4 + 2 \times 2 + 3 \times 3 + 8 \times 2) \bmod 11 = 0$. Далее, n можно найти как $n = 105 \times \left(0 + \frac{1\times -1}{3} + \frac{2\times 1}{5} + \frac{3\times 1}{7}\right) = 52$.

Заметим, что если $0 \le C(X) < C(M)$, то $|C(X)|_{C(M)} = C(X)$. Если $C(X) < 0$, то $|C(X)|_{C(M)} = C(X) + C(M)$. Если $C(X) \ge C(M)$, то $|C(X)|_{C(M)} = C(X) - C(M)$. Таким образом, любое конкретное значение $|C(X)|_{C(M)}$ может ввести два возможных значения для $C(X)$. Существует неопределённость в определении того, какой случай является правильным. Эта неоднозначность обусловлена нелинейной характеристикой функции ядра.

Миллер [61] предложил использовать избыточный модуль m_E, больший, чем $C(M)$, и вычислять

$$C(n) = \left| n\frac{C(M)}{M} - \sum_{i=1}^{k} \frac{w_i}{m_i} r_i \right|_{m_E} \tag{1.47}$$

так что никакой неоднозначности не может возникнуть, потому что $C(n)$ оценивается по модулю m_E, большему, чем диапазон ядра. Однако этот метод требует дополнительного оборудования. Кришнан и др. [67] использовали дополнительный модуль, чтобы найти множитель $C(M)$, который необходимо вычесть, что требует дополнительного оборудования.

1.4 Масштабирование оснований СОК

При реализации кодов исправления ошибок в СОК часто требуется получить масштабируемый остаток, т. е. $\left|\frac{x}{2^{\alpha}}\right|_m$ [68]. Это может быть достигнуто путём последовательного деления на $2 \bmod m$. Например, $\left|\frac{13}{2^3}\right|_{19}$ можно получить, сначала вычислив $\left|\frac{13}{2}\right|_{19} = \left|\frac{13+19}{2}\right|_{19} = 16$. Далее, $\left|\frac{13}{2^2}\right|_{19} = \left|\frac{16}{2}\right|_{19} = 8$, и $\left|\frac{13}{2^3}\right|_{19} = \left|\frac{8}{2}\right|_{19} = 4$.

Процедура масштабирования x на 2 подразумевает добавление модуля m в случае, когда младший бит x равен 1, и деление на два (игнорирование младшего бита или выполнение сдвига вправо на один бит). В случае, если младший бит x равен нулю, достаточно просто разделить его на два (игнорируя младший бит или сдвиг вправо).

Алгоритм Монтгомери [69] позволяет оценить $\left|\frac{x}{2^{\alpha}}\right|_m$, рассматривая α-биты за один раз (также называемый реализацией с более высоким основанием). Здесь мы хотим найти кратное m, которое нужно добавить к x, чтобы сделать его точно делимым на 2^{α}. Сначала нам нужно вычислить $\beta = |-m|_{2^{\alpha}}$. Далее, зная слово Z, соответствующее младшему биту α из x, нам нужно вычислить $Y = x + |Z\beta|_{2^{\alpha}} \times m$, которое будет точно делимо на 2^{α}. Деление осуществляется смещением вправо на α бит.

Рассмотрим $x = |101001101|_2 = 333$, мы хотим найти $\left|\frac{x}{16}\right|_{23}$. Находим $\beta = \left|-\frac{1}{23}\right|_{16} = 9$. Мы знаем, что $\alpha = 13$ (4-битные младшие разряды x). Таким образом, нам нужно вычислить значение $Y = 333 + |13 \times 9|_{16} \times 23 = 333 + 5 \times 23 = 448$, которое точно делится на 16, чтобы получить 28. Принимая по одному биту за раз, нам понадобилось бы четыре шага $(333 + 23)/2 = 178$, $178/2 = 89$, $(89 + 23)/2 = 56$ и $56/2 = 28$. Процедура может быть расширена, чтобы найти масштабирование произвольной степенью $2 \bmod m_i$. Метод

Монтгомери может быть расширен до умножения с той разницей, что каждый раз, когда добавляется частичное произведение, в вычислении используется младшие биты результата.

Расширение системы оснований является одной из основных немодульных операций в СОК. Выполнение этой операции бывает необходимо при выполнении ряда других операций, например, при выполнении операции деления чисел, при вычислении позиционных характеристик, при обнаружении переполнения сложения или умножения чисел. Кроме того, при обнаружении и исправлении ошибок также необходимо добавление одного или нескольких контрольных оснований.

Задачу расширения системы оснований можно сформулировать следующим образом: найти остаточное представление числа по новому основанию (новым основаниям), если известно представление числа по другим основаниям, т. е. найти остаток от деления на число, если известны остатки от деления на другие числа.

Один из путей расширения системы оснований состоит в переводе числа в позиционную систему счисления и вычислении остатка от деления на новый модуль. Надо признать, что этот путь не является рациональным с точки зрения числа операций.

Другой метод расширения системы оснований позволяет определить цифру числа по новому основанию, базируясь на таких позиционных характеристиках числа, как ранг числа, след числа. Пусть задана система оснований $m_1, m_2, \dots, m_n$ с диапазоном M, ортогональными базисами $B_1, B_2, \dots, B_n$, веса которых $w_1, w_2, \dots, w_n$. По определению $B_i = w_i \frac{M}{m_i}$, $i = \overline{1, n}$. Пусть в этой системе задано число $A = (\alpha_1, \alpha_2, \dots, \alpha_n)$. Расширим систему оснований, добавляя основание m_{n+1}, тогда диапазон системы станет $M' = m_{n+1} \cdot M$, ортогональные базисы системы $B'_1, B'_2, \dots, B'_{n+1}$, их веса $w'_1, w'_2, \dots, w'_{n+1}$, причём $B'_i = w_i \frac{m_{n+1} \cdot M}{m_i}$, $i = \overline{1, n+1}$. Задача состоит в определении цифры α_{n+1} числа A по основанию m_{n+1}.

Учитывая, что минимальным следом числа $A = (\alpha_1, \alpha_2, \dots, \alpha_{n+1})$ по основанию m_{n+1} называют цифру S_A^*, при которой число $A^* = (\alpha_1, \alpha_2, \dots, \alpha_n, S_A^*)$ находится в интервале $\left[0; \frac{M'}{m_{n+1}}\right]$, и что число $A \in [0; M)$, то определение цифры по основанию m_{n+1} сводится к определению минимального следа числа A в расширенной системе оснований.

Чтобы получить формулы для цифры α_{n+1} запишем выражение для числа A в основной и расширенных системах:

$$A = \alpha_1 B_1 + \alpha_2 B_2 + \dots + \alpha_n B_n - r_A \cdot M \text{ и } A = \sum_{i=1}^{n} \alpha_i B_i' - r_A' \cdot M',$$

где r_A и r_A' – ранги числа A в основной и расширенной системах.

Приравнивая правые части этих выражений, определяем α_{n+1}:

$$\alpha_{n+1} = \frac{1}{w_{n+1}'}\left(\alpha_1 \frac{w_1 - w_1' m_{n+1}}{m_1} + \alpha_2 \frac{w_2 - w_2' m_{n+1}}{m_2} + \dots + \alpha_n \frac{w_n - w_n' m_{n+1}}{m_n} - r_A + r_A' m_{n+1}\right),$$

или обозначая через λ_i целое число $\lambda_i = \frac{w_i' m_{n+1} - w_i}{m_i}$, $i = \overline{1, n}$, а через δ_A величину $\delta_A = \sum_{i=1}^{n} \alpha_i \lambda_i$, получаем $w_{n+1}' \alpha_{n+1} + r_A = -\delta_A + r_A m_{n+1}$. Наконец, представляя δ_A в виде $\delta_A = k \cdot m_{n+1} - q$, где k, q – целые неотрицательные числа и $q < m_{n+1}$, получаем $w_{n+1}' \alpha_{n+1} + r_A = (\bar{r}_A - k) m_{n+1} + q$, или

$$w_{n+1}' \alpha_{n+1} + r_A \equiv q (\operatorname{mod} m_{n+1}) \tag{1.48}$$

Формула (1.48) и есть формула расширения диапазона чисел.

Для практической реализации этой формулы поступают следующим образом:

1. Вычисляют параметры основной и расширенной систем (ортогональные базисы, их веса, минимальные псевдо-ортогональные числа с их рангами и кратностями).
2. Конструируют число $A = (\alpha_1, \alpha_2, \dots, \alpha_n)$ из минимальных псевдоортогональных чисел $W\alpha_{ij}$, $i = \overline{1, p_j - 1}$, $j = \overline{1, n-1}$, с рангами $r_{\alpha_{ij}}$, которые однозначно определяются выбранной системой оснований $m_1, m_2, \dots, m_n$. В результате получают число $W = (\alpha_1, \alpha_2, \dots, \alpha_n, S_A)$, где S_A – след числа, а его ранг находят по теореме о ранге суммы:

$$r_{W_A} = \sum_{j=1}^{n_i} r_{\alpha_{ij}} - m_n \pi_{W_A}, \tag{1.49}$$

где π_{W_A} – число переходов по основанию m_n.

3. Расширяют число W_A по формуле расширения (1.48). Пользуясь величиной ранга r_{W_A}, вычисленной по формуле (1.49), получают число $A' = (\alpha_1, \alpha_2, \dots, \alpha_n, S_A, \alpha'_{n+1})$, которое отличается от искомого числа A цифрами по двум последним основаниям.
4. Если $S_A = \alpha_n$, то $A' = A$, т. е. A' – искомое расширение числа A.
5. Если $S_A \neq \alpha_n$, то прибавляют к числу A' такое из минимальных псевдоортогональных чисел $\overline{W}_{\beta_n n} = \left(0, \dots, 0, \beta_n, \bar{S}_{\beta_n}\right)$ кратности $k_{\beta_n n}$, где $\beta_n \equiv (\alpha_1 - S_A)(\mathrm{mod}\, m_n)$, которое превратит цифру по основанию m_n в α_n. В результате получают число $A^{(2)} = A' + \overline{W}_{\beta_n n} = (\alpha_1, \alpha_2, \dots, \alpha_n, \alpha_{n+1}^2)$.
6. Если кратность $k_{\beta_n n} \leq m_n - (n-1)$, то число $A^{(2)}$ является искомым расширением числа A, так как к числу A', не превышающему $(n-1)\frac{M}{m_n}$ прибавили число $\overline{W}_{\beta_n n}$, не превышающее $\left(m_n - (n-1)\right) \cdot \frac{M}{m_n}$, т.е. $A^{(2)}$ не превышает M, т.е. величины 1-го интервала.
7. Если $k_{\beta_n n} > m_n - (n-1)$, то число $A^{(2)}$ может располагаться либо в последних $(n-1)\frac{M}{m_n}$ частях 1-го интервала $[0, M)$, либо в младших $(n-2)\frac{M}{m_n}$ частях 2-го интервала $[M, 2M)$, а тогда искомым является число: $A^{(3)} = (\alpha_1, \dots, \alpha_n, (\alpha_{n+1}^2 - 1) \bmod m_{n+1})$.

Проводя анализ рассмотренного метода расширения систем оснований, можно отметить, что он опирается на такие позиционные характеристики, как ранг и след числа, которые требуют расчёта ряда параметров (ортогональных базисов, их весов, минимальных псевдо-ортогональных чисел, их рангов с кратностями), вычисление которых может показаться громоздким. Однако все эти параметры определяются только основаниями системы, поэтому могут быть рассчитаны заранее. Более того, с помощью указанных позиционных

характеристик могут быть организованы и другие немодульные процедуры, что позволяет построить машинную арифметику, опираясь на эти характеристики.

Ещё один путь решения поставленной задачи представляет собой перевод числа из СОК в ОПСС с дополнительным финальным шагом. Рассмотрим этот метод.

Пусть СОК состоит из оснований $m_1, m_2, \dots, m_n$. Объём диапазона этой системы будет $M = \prod_{i=1}^{n} m_i$. Добавим к числу оснований СОК новое основание m_{n+1}. Объём диапазона этой системы $\rho = \prod_{i=1}^{n+1} m_i$. Тогда любое число x из диапазона $[0, \rho)$ в обобщённой позиционной системе счисления представимо в виде $x = a_{n+1} \prod_{i=1}^{n} m_i + a_n \prod_{i=1}^{n-1} m_i + \dots + a_3 m_1 m_2 + a_2 m_1 + a_1$. Если число x будет лежать в первоначальном диапазоне $[0, M)$, то в ОПС цифра $a_{n+1} = 0$. Этот факт и используется для получения остатка от деления числа x на новое основание СОК m_{n+1}.

Пусть число x имело представление $(\alpha_1, \alpha_2, \dots, \alpha_n)$ по основаниям $m_1, m_2, \dots, m_n$. Добавляем новое основание m_{n+1}, тогда число $x = \left(\alpha_1, \alpha_2, \dots, \alpha_n, |x|_{m_{n+1}}\right)$ в системе оснований $m_1, m_2, \dots, m_n, m_{n+1}$, где $|x|_{m_{n+1}}$ – остаток от деления числа x на m_{n+1}, т.е. искомая цифра по новому основанию.

Для определения этой цифры рассматриваем алгоритм перевода числа из СОК в ОПСС, включая неизвестную цифру $|x|_{m_{n+1}}$ в проводимые операции. При этом мы последовательно будем получать цифры ОПСС $a_1, a_2, \dots, a_n$ и выражение для цифры a_{n+1}. Но так как по предположению число $x \in [0, M)$, то цифра $a_{n+1} = 0$. Из полученного соотношения и определяем искомую цифру $|x|_{m_{n+1}}$.

Операцию расширения можно проводить сразу по нескольким основаниям, так как результат образования цифры в СОК по новому основанию m_{n+1} зависит только от первых цифр.

Преимущества метода расширения системы основания с помощью перевода в ОПСС состоит в том, что:

во-первых, все вычисления идут в параллельных каналах по отдельным модулям;

во-вторых, не требуется вычисления большого количества дополнительных величин (необходимо наличие в памяти только констант τ_{ij});

в-третьих, возможно получение расширенного представления числа сразу по нескольким дополнительным основаниям, что не влияет на быстродействие операции.

1.5 Выводы по первой главе

1. Использование модулярной арифметики позволяет, с одной стороны, позволяет эффективно реализовать арифметические операции: сложения и умножения чисел, но при этом алгоритмы определения знака числа, сравнения чисел, Евклидово деление, масштабирование и т. д. являются вычислительно сложными. В основе кодов обнаружения, локализации и исправления ошибок на базе системы остаточных классов лежат три группы алгоритмов:

1. Алгоритмы перевода чисел из позиционную систему счисления (ПСС) в СОК.
2. Алгоритмы перевода чисел из СОК в ПСС.
3. Алгоритмы масштабирование чисел.

2. Эффективность реализации алгоритмов обнаружения, локализации и исправления ошибок зависит от вычислительной сложности алгоритма перевода чисел из СОК в ПСС. Выбор набора модулей в СОК определяется скоростью преобразования СОК в двоичный код для эффективного выполнения таких операций, как сравнение, масштабирование, обнаружение знаков и исправление ошибок.

3. Метод ОПСС является последовательным и включает в себя вычитание по модулю и умножение по модулю путём мультипликативных инверсий одного модуля по отношению к остальным модулям.

4. Возможно использование функции ядра для преобразования СОК в двоичный код, а также масштабирования.

5. Расширение системы оснований является одной из основных немодульных операций в СОК. Выполнение этой операции бывает необходимо при выполнении ряда других операций, например, при выполнении операции деления чисел, при вычислении позиционных характеристик, при обнаружении переполнения сложения или умножения чисел. Кроме того, при обнаружении и исправлении ошибок также необходимо добавление одного или нескольких контрольных оснований.

2 Коды исправления ошибок в модулярном коде

Возможность обнаружения и исправления ошибок ИСОК использовалась в нескольких отказоустойчивых приложениях, описанных в литературе [70-80]. Помимо исправления ошибок, путём добавления двух избыточных модулей может быть обнаружено переполнение диапазона. ИСОК также использовалась для разработки отказоустойчивого алгоритма свёртки, который хорошо подходит для реализации на многопроцессорных системах [73]. Используя метод деления столбиком, алгоритм может обнаруживать и исправлять ошибки обработки. Кроме того, код с исправлением ошибок на основе шестимодульного ИСОК был реализован на гибридных запоминающих устройствах [76]. По сравнению с кодами Рида-Соломона, этот код обеспечивает большее хранилище данных с аналогичной возможностью исправления ошибок.

Как правило, обнаружение и исправление ошибок в ИСОК выполняется в три последовательных этапа: проверка наличия ошибок, идентификация ошибочных цифр остатка и исправление ошибок. Алгоритмы, которые исправляют только одну ошибку разряда остатка, были предложены в [81]-[84]. В случае использования двух избыточных модулей, декодирование величины и местоположения одиночной ошибки может было решено с использованием только одного синдрома и одной таблицы поиска [84]. С другой стороны, обнаружение и исправление нескольких ошибок остаточных разрядов является более сложным и трудоёмким. Это в основном связано с огромным количеством комбинаций различных местоположений и величин остатков для поиска ошибок на втором шаге. Как правило, существующие алгоритмы обнаружения и исправления множественных ошибок используют три различных метода для определения местоположения ошибочных цифр остатка. Это проверка согласованности с использованием синдрома [85], [86], восстановление чисел по Китайской теореме об остатках (КТО) [87], [88] и проекции модуля [86], [89]. Эти алгоритмы требуют итерационных и рекурсивных вычислений и сравнений, которые не поддаются параллельной аппаратной реализации.

2.1 Метод проекции

Рассмотрим подход к обнаружению и исправлению множественных ошибок в ИСОК [90]. Разряды остатков ИСОК разделены на три группы так, чтобы любая комбинация ошибок могла быть однозначно идентифицирована одной из семи категорий местоположения ошибки. Из полученного представления остатка выводятся три синдрома для обнаружения до $2t$ и исправления до t ошибок разрядов остатка, где число избыточных модулей равно $2t$. Независимо от количества модулей, представление остатка ошибок может быть однозначно извлечено этими синдромами из шести таблиц поиска за несколько шагов и вычитается непосредственно из полученных цифр остатка для исправления ошибок. Задержка фиксирована и не зависит от размера набора модулей. Второй критерий выбора модулей, который накладывает большое избыточное пространство, может быть полностью устранён, делая число вычислений синдрома зависимым от числа информационных модулей. По сравнению с существующими алгоритмами, предложенный алгоритм является более простым и имеет очень низкую сложность декодирования ошибок. Для каждого вычисления синдрома требуется только одно вычитание по модулю. Предложенный алгоритм классифицирует различные комбинации ошибок вычетов на семь групп местоположений без ограничения по количеству информации и избыточных модулей, что резко контрастирует с алгоритмом [84], который просто находит одну ошибку разряда вычетов для ИСОК только с двумя избыточными модулями.

Первый алгоритм обнаружения и исправления множественных ошибок разрядов остатков был введён в [85]. Алгоритм декодирования ошибок устанавливает зависимость порядка в разрядах остатка таким образом, что он способен исправлять единичные пакетные ошибки до t соседних разрядов остатков. Метод вычисления синдрома вычисляет разницу между полученными разрядами остатка и разрядами остатков по расширенным основаниям.

Синдромы $|\Delta|_{m_i}$ для $i = 1, 2, \dots, k, k+1, \dots, k+2t$ сгруппированы в следующие наборы:

$$
\begin{gathered}
S_1 \equiv \left\{|\Delta|_{m_{k+1}}, |\Delta|_{m_{k+2}}, \dots, |\Delta|_{m_{k+2t}}\right\} \\
S_2 \equiv \left\{|\Delta|_{m_{k-t}}, |\Delta|_{m_{k-t+1}}, \dots, |\Delta|_{m_{k+t-1}}\right\} \\
S_3 \equiv \left\{|\Delta|_{m_{k-2t-1}}, |\Delta|_{m_{k-2t}}, \dots, |\Delta|_{m_{k-2}}\right\} \\
\vdots \\
S_{[(k+t-1)/(t+1)]+1} \equiv \left\{|\Delta|_{m_1}, |\Delta|_{m_2}, \dots, |\Delta|_{m_{2t}}\right\}
\end{gathered}
\tag{2.1}
$$

где k – количество информационных модулей.

Местоположение ошибки полученных разрядов остатка определяется по S_i для $i = 1,2, \dots, [(k+t-1)/(t+1)] + 1$, при условии, что число ненулевых элементов меньше или равно t. Безошибочное представление остатка затем строится путём замены ошибочных цифр остатка соответствующими базовыми расширенными цифрами остатка. Сложность этого алгоритма зависит от количества проверяемых наборов и количества синдромов, вычисляемых для каждого набора.

Вместо поиска ошибочных цифр остатка в [87] был предложен другой алгоритм, использующий метод непрерывных дробей. Правильная доля ошибки затем используется для вычисления величины представления безошибочного остатка. Вычисления дробей решаются рекурсивным евклидовым алгоритмом. Однако в данном методе требуется большее количество итераций для большего числа модулей и ошибок остаточных разрядов, что делает этот метод неэффективным для аппаратной реализации.

Сан и Кришна [86] представили подход теории кодирования к контролю ошибок в ИСОК и ввели понятия веса Хэмминга, минимального расстояния, распределения веса и возможностей обнаружения и коррекции ошибок в ИСОК. Было предложено четыре алгоритма, а именно: исправление одиночных ошибок и обнаружение множественных ошибок, исправление двойных ошибок и обнаружение множественных ошибок, исправление однократных пакетных ошибок и исправление расширенных двузначных ошибок. Первые три алгоритма

требуют вычисления синдромов, как в [85], за исключением того, что в вычислениях используются только избыточные разряды вычетов. Ошибочные цифры остатков находятся по совпадению всех цифр ошибок, рассчитанных по синдромам. Хотя число вычисляемых синдромов меньше, чем в [85], проверка согласованности требует итерационных вычислений, включающих каждую возможную комбинацию местоположений ошибок. Расширенный алгоритм коррекции двойной ошибки использует подход, отличный от первых трёх алгоритмов. Он использует концепцию проекции модуля, введённую в [91] для коррекции ошибок одиночных разрядов остатков, и является расширением алгоритма коррекции ошибок одиночных разрядов остатков, предложенного в [92]. Положение ошибочных остатков получают, когда величина представления числа после исключения двух ошибочных цифр остатка попадает в разрешённый диапазон. Величина проверяется путём вычисления смешанных разрядов корня представления остатка каждой проекции. Этот процесс аналогичен первым трём алгоритмам в том, что требуется итерационная оценка каждой возможной комбинации местоположений ошибок остаточных разрядов.

В [88] были предложены два алгоритма на основе КТО, однозначного кодирования и списочного декодирования, для обнаружения и исправления малого и большого числа ошибок остаточных разрядов соответственно. Метод однозначного кодирования является расширенным вариантом алгоритма, предложенного в [87]. Он ищет два неизвестных целых числа y и z ограниченного размера, таких, что $\left|y \cdot \tilde{X}\right|_N \equiv z$, где $\tilde{X}$ – величина полученного представления остатка, а N – динамический диапазон ИСОК. Поиск y и z должен выполняться рекурсивно, пока z/y не станет целым числом. Метод списочного декодирования ищет последовательность целых чисел $c_0, c_1, \dots, c_l$, определенных ограниченных размеров из полученных разрядов вычетов, чтобы сформировать полином $C(x) = \sum_{i=0}^{l} c_i x^i$ степени l такой, что $\left|\sum_{i=0}^{l} c_i x^i\right|_N = 0$, где l – функция числа информационных и избыточных модулей, а также значений наибольшего и наименьшего модулей. Безошибочное представление остатка

восстанавливается путём решения полинома для целочисленных корней и нахождения их амплитуд согласования с полученными разрядами вычетов.

Алгоритм исправления ошибок в [89] использует ту же концепцию проекции модуля, что и расширенный алгоритм исправления ошибок с двузначными числами в [86], но он использует КТО вместо обобщённой позиционной системы счисления (ОПСС) для вычисления величины каждой проекции. Алгоритм способен корректировать до t ошибок по размерам остатков, игнорируя t разрядов остатков из полученных остатков на каждой итерации вычисления величины. Если вычисленная величина попадает в допустимый диапазон, игнорируемые цифры остатка ошибочны. Максимальное количество итераций, требуемых в этом алгоритме, составляет C_t^n, где n – общее количество модулей. В [88] также была предложена расширенная схема сокращения числа итераций на основе метода декодирования максимального правдоподобия (MLD). Сначала он вычисляет величину r цифр остатка, где r – количество избыточных цифр остатка. Только когда вычисленная величина находится в допустимом диапазоне, оставшиеся цифры остатка будут вычислены с использованием метода расширения оснований. Если t или менее цифр расширенных оснований остатка отличаются от полученных цифр остатка, то цифры остатка по расширенным основаниям являются безошибочными цифрами остатка. Требуемое количество итераций равно $[C_t^n/C_t^r]$, что значительно меньше, чем в первом методе, особенно когда количество модулей велико.

В общем случае для определения ошибочных цифр остатков для существующих алгоритмов обнаружения и исправления ошибок с несколькими цифрами остатков требуются рекурсивные или итеративные вычисления и сравнения. Поскольку число итераций зависит от местоположения ошибок разрядов, время, затраченное на обнаружение и исправление ошибок разрядов, является недетерминированным. Наихудший случай возрастает с размером выбранного набора модулей, который зависит от количества информационных

модулей и возможностей коррекции ошибок алгоритмов. Следовательно, эти алгоритмы не поддаются аппаратной реализации.

2.2 Метод синдрома

Из выражения вектора ошибок

$$E_l = \left| \sum_{i=1}^{t} \frac{M_N}{m_{l_i}} \left| \left| M_{l_i}^{-1} \right|_{m_{l_i}} e_{l_i} \right|_{m_{l_i}} \right|_{M_N} = \sum_{i=1}^{t} a_{l_i} \frac{M_N}{m_{l_i}} - r_n M_N = a \frac{M_N}{\prod_{i=1}^{t} m_{l_i}} =$$

$$a \prod_{\substack{j=1 \\ l_j \neq l_i}}^{n-t} m_{l_j},$$

E_l кратно произведению безошибочных модулей. Когда $m_{k+2}, \dots, m_{k+r} > m_i \ \forall i \leq k$ и $t \leq [r/2]$, минимальная разница между величинами ошибок любых двух векторов ошибок всегда больше, чем M_K. Начиная с $X < M_K$, ошибочное представление остатка может быть обнаружено путём проверки того, попадает ли X' в недопустимый диапазон. Однако декодирование X' из $(x'_1, x'_2, \dots, x'_n)$ с помощью КТО требует больших вычислительных операций по модулю M_N. После обнаружения ошибок необходимо определить местонахождение ошибочных остатков, прежде чем их можно будет исправить. Обычно для определения местоположения ошибочных остатков требуются трудоёмкие итерационные расчёты, которые возрастают с размером выбранного набора модулей. Для преодоления вышеуказанных недостатков в данном разделе предлагается новый метод расчёта синдрома. Он использует меньшие операции по модулю, и количество вычислений фиксируется независимо от размера набора модулей.

2.2.1 Выделение синдромов для идентификации ошибок

Величина E_l ограничена M_N. Таким образом, каждый вектор ошибки соответствует уникальному E_l. Учитывая величину ошибки E_l, можно

однозначно определить соответствующие цифры ошибки e_{l_i} и местоположение ошибки l_i. К сожалению, E_l не может быть получен путём прямого декодирования представления остатка $(x'_1, x'_2, \dots, x'_n)$, так как X неизвестен.

Чтобы избежать использования больших операций декодирования ошибок по модулю M_N, n цифр остатка $(x'_1, x'_2, \dots, x'_n)$ X разделены на три различные группы, $(x'_1, x'_2, \dots, x'_k)$, $(x'_{k+1}, x'_{k+2}, \dots, x'_{k+t})$ и $(x'_{k+t+1}, x'_{k+t+2}, \dots, x'_{k+r})$, так что операции по модулю, необходимые для однозначного разрешения вектора ошибок, примерно в три раза меньше, чем M_N.

Динамический диапазон M_N ИСОК $\{m_1, m_2, \dots, m_k, m_{k+1}, \dots, m_{k+r}\}$, где $m_{k+1}, m_{k+2}, \dots, m_{k+r} > m_i \ \forall i \leq k$, можно разделить на

$$M_N = \prod_{i=1}^{k} m_i \times \prod_{i=k+1}^{k+r} m_i = \prod_{i=1}^{k} m_i \times \prod_{i=k+1}^{k+t} m_i \times \prod_{i=k+t+1}^{k+r} m_i = \\ = M_K \times M_U \times M_V \tag{2.2}$$

где $M_K = \prod_{i=1}^{k} m_i$ является произведением информационных модулей, $M_U = \prod_{i=k+1}^{k+t} m_i$ – произведение первых t избыточных модулей и $M_V = \prod_{i=k+t+1}^{k+r} m_i$ – произведение остальных t избыточных модулей.

Для удобства определим следующие обозначения:

X обозначает величину безошибочного представления ИСОК $(x_1, x_2, \dots, x_k, x_{k+1}, \dots, x_{k+r})$. X попадает в диапазон $[0, M_K - 1]$.

$\tilde{X}$ обозначает величину полученного представления ИСОК $(\tilde{x}_1, \tilde{x}_2, \dots, \tilde{x}_k, \tilde{x}_{k+1}, \dots, \tilde{x}_{k+r})$. $\tilde{X}$ попадает в диапазон $[0, M_N - 1]$.

$\tilde{X}_K$, $\tilde{X}_U$ и $\tilde{X}_V$ обозначают величины представлений в СОК для информационных модулей, для первых t избыточных модулей, для оставшихся t избыточные модулей соответственно, то есть, $\tilde{X}_K \equiv (\tilde{x}_1, \tilde{x}_2, \dots, \tilde{x}_k)$, $\tilde{X}_U \equiv (\tilde{x}_{k+1}, \tilde{x}_{k+2}, \dots, \tilde{x}_{k+t})$ и $\tilde{X}_V \equiv (\tilde{x}_{k+t+1}, \tilde{x}_{k+t+2}, \dots, \tilde{x}_{k+r})$. $\tilde{X}$, $\tilde{X}_K$, $\tilde{X}_U$ и $\tilde{X}_V$ могут быть ошибочными или безошибочными.

E_K, E_U и E_V обозначают величины цифр ошибки, расположенные в информационных каналах модулей, в первых t избыточных каналах модулей, и

в остальных t избыточных каналах модулей, то есть, $E_K \equiv \left(e_{k_1}, e_{k_2}, \dots, e_{k_i}, 0, \dots, 0\right)$, $E_U \equiv \left(0, \dots, 0, e_{u_1}, e_{u_2}, \dots, e_{u_j}, 0 \dots 0\right)$ и $E_V \equiv \left(0, \dots, 0, e_{v_1}, e_{v_2}, \dots, e_{v_l}\right)$, где $(k_1, k_2, \dots, k_i)$, $\left(u_1, u_2, \dots, u_j\right)$ и $(v_1, v_2, \dots, v_l)$ представляют собой положения цифр ошибки и $i, j, l \leq t$. E_K, E_U и E_V ограничены M_N и могут быть вычислены из их соответствующих цифр ошибки с помощью

$$E_l = \left| \sum_{i=1}^{t} \frac{M_N}{m_{l_i}} \left| \left| M_{l_i}^{-1} \right|_{m_{l_i}} e_{l_i} \right|_{m_{l_i}} \right|_{M_N} = \sum_{i=1}^{t} a_{l_i} \frac{M_N}{m_{l_i}} - r_n M_N = a \frac{M_N}{\prod_{i=1}^{t} m_{l_i}} =$$

$$a \prod_{\substack{j=1 \\ l_j \neq l_i}}^{n-t} m_{l_j}.$$

Исходя из приведённых выше обозначений, $\tilde{X}_K$, $\tilde{X}_U$ и $\tilde{X}_V$ могут быть выражены в членах $\tilde{X}$ через

$$\tilde{X}_K = \left|\tilde{X}\right|_{M_K} \tag{2.3}$$

$$\tilde{X}_U = \left|\tilde{X}\right|_{M_U} \tag{2.4}$$

$$\tilde{X}_V = \left|\tilde{X}\right|_{M_V} \tag{2.5}$$

Если ошибки нет, $\tilde{X} = X$. На основе пунктов (2.3)-(2.5),

$$\tilde{X}_K = |X|_{M_K} = X \tag{2.6}$$

$$\tilde{X}_U = |X|_{M_U} = X \tag{2.7}$$

$$\tilde{X}_V = |X|_{M_V} \tag{2.8}$$

Следует отметить, что (2.7) является действительным тогда и только тогда, когда $M_U > M_K$, а M_U и M_K взаимно просты.

В противном случае, если есть ошибочный разряд(ы) остатка в одной или нескольких группах среди $(\tilde{x}_1, \tilde{x}_2, \dots, \tilde{x}_k)$, $(\tilde{x}_{k+1}, \tilde{x}_{k+2}, \dots, \tilde{x}_{k+t})$ и $(\tilde{x}_{k+t+1}, \tilde{x}_{k+t+2}, \dots, \tilde{x}_{k+r})$, $\tilde{X}$ может быть выражен как $\tilde{X} = X + E$, т. е.,

$$\tilde{X}_K = |X + E_K|_{M_K} = X + |E_K|_{M_K} - \alpha_K M_K \tag{2.9}$$

$$\tilde{X}_U = |X + E_U|_{M_U} = X + |E_U|_{M_U} - \alpha_U M_U \tag{2.10}$$

$$\tilde{X}_V = |X + E_V|_{M_V} = |X|_{M_V} + |E_V|_{M_V} - \alpha_V M_V \tag{2.11}$$

где

$$\alpha_K = \begin{cases} 0, \text{если } X + |E_K|_{M_K} < M_K \\ 1, \text{если } X + |E_K|_{M_K} \geq M_K \end{cases} \tag{2.12}$$

$$\alpha_U = \begin{cases} 0, \text{если } X + |E_U|_{M_U} < M_U \\ 1, \text{если } X + |E_U|_{M_U} \geq M_U \end{cases} \tag{2.13}$$

$$\alpha_V = \begin{cases} 0, \text{если } |X|_{M_V} + |E_V|_{M_V} < M_V \\ 1, \text{если } |X|_{M_V} + |E_V|_{M_V} \geq M_V \end{cases} \tag{2.14}$$

Поскольку полученные цифры остатка сгруппированы в три различные группы, ошибочные цифры остатка могут существовать в одной из следующих категорий в зависимости от их местоположения ошибки.

- *МО1:* Ошибочная цифра(ы) остатка только в $\tilde{X}_K$.
- *МО2:* Ошибочная цифра(ы) остатка только в $\tilde{X}_U$.
- *МО3:* Ошибочная цифра(ы) остатка только в $\tilde{X}_V$.
- *МО4:* Ошибочные цифры остатка в $\tilde{X}_K$ и $\tilde{X}_U$.
- *МО5:* Ошибочные цифры остатка в $\tilde{X}_K$ и $\tilde{X}_V$.
- *МО6:* Ошибочные цифры остатка в $\tilde{X}_U$ и $\tilde{X}_V$.
- *МО7:* Ошибочные цифры остатка в $\tilde{X}_K$, $\tilde{X}_U$ и $\tilde{X}_V$.

В принципе, наличие любых t-остаточных ошибок в числах n остатков может быть обнаружено двумя синдромами, δ_1 и δ_2. δ_1 вычисляется из $(x'_1, x'_2, \dots, x'_k)$ и $(x'_{k+1}, x'_{k+2}, \dots, x'_{k+t})$, δ_2 вычисляется из $(x'_1, x'_2, \dots, x'_k)$ и $(x'_{k+t+1}, x'_{k+t+2}, \dots, x'_{k+r})$. Каждое из этих вычислений синдрома включает цифры остатка $(k + t)$. Поскольку динамические диапазоны δ_1 и δ_2 намного меньше, чем M_N, каждое значение синдрома может отображаться более чем в одном векторе погрешностей не более t остаточных разрядов в разных локациях l_i и величин e_{l_i}. Следовательно, двух синдромов недостаточно для исправления ошибок, введённых в три различные группы цифр остатка. Третий синдром δ_3, вычисленный из $(x'_{k+1}, x'_{k+2}, \dots, x'_{k+t})$ и $(x'_{k+t+1}, x'_{k+t+2}, \dots, x'_{k+r})$, необходим. Найдя общий вектор ошибок, представленный тремя синдромами, можно точно определить погрешности разряда остатка при условии, что $M_U > M_K$.

Три синдрома δ_1, δ_2 и δ_3 можно вычислить следующим образом:

$$\delta_1 = \left|\tilde{X}_U - \tilde{X}_K\right|_{M_U} \tag{2.15}$$

$$\delta_2 = \left|\tilde{X}_V - \tilde{X}_K\right|_{M_V} \tag{2.16}$$

$$\delta_3 = \left|\tilde{X}_V - \tilde{X}_U\right|_{M_V} \tag{2.17}$$

Заменив (2.6)-(2.8) на (2.15)-(2.17), мы получим

$$\delta_1 = |X - X|_{M_U} = 0 \tag{2.18}$$

$$\delta_2 = \left||X|_{M_V} - X\right|_{M_V} = \left|-\left\lfloor\frac{X}{M_V}\right\rfloor M_V\right|_{M_V} = 0 \tag{2.19}$$

$$\delta_3 = \left||X|_{M_V} - X\right|_{M_V} = 0 \tag{2.20}$$

Поэтому, когда нет ошибочной цифры остатка, синдромы $\delta_1 = \delta_2 = \delta_3 = 0$.

Если какая-либо цифра остатка содержит ошибку, то по значениям трёх синдромов можно также определить категорию места ошибки. Заменив (2.9)-(2.11) на (2.15)-(2.17) значения δ_1, δ_2 и δ_3 для каждой возможной категории местоположения ошибки можно вычислить следующим образом:

МО1:

$$\delta_1 = \left|X - \left(X + |E_K|_{M_K} - \alpha_K M_K\right)\right|_{M_U} = \left|\alpha_K M_K - |E_K|_{M_K}\right|_{M_U} \tag{2.21}$$

$$\delta_2 = \left|X - \left(X + |E_K|_{M_K} - \alpha_K M_K\right)\right|_{M_V} = \left|\alpha_K M_K - |E_K|_{M_K}\right|_{M_V} \tag{2.22}$$

$$\delta_3 = \left||X|_{M_V} - X\right|_{M_V} = 0 \tag{2.23}$$

МО2:

$$\delta_1 = \left|\left(X + |E_U|_{M_U} - \alpha_U M_U\right) - X\right|_{M_U} = |E_U|_{M_U} \tag{2.24}$$

$$\delta_2 = \left||X|_{M_V} - X\right|_{M_V} = 0 \tag{2.25}$$

$$\delta_3 = \left||X|_{M_V} - \left(X + |E_U|_{M_U} - \alpha_U M_U\right)\right|_{M_V} = \left|\alpha_U M_U - |E_U|_{M_U}\right|_{M_V} \tag{2.26}$$

МО3:

$$\delta_1 = |X - X|_{M_U} = 0 \tag{2.27}$$

$$\delta_2 = \left||X|_{M_V} - \left(X + |E_V|_{M_V} - \alpha_V M_V\right)\right|_{M_V} = |E_V|_{M_V} \tag{2.28}$$

$$\delta_3 = \left|\left(|X|_{M_V} + |E_V|_{M_V} - \alpha_V M_V\right) - X\right|_{M_V} = |E_V|_{M_V} \tag{2.29}$$

MO4:

$$\delta_1 = \left|\left(X + |E_U|_{M_U} - \alpha_U M_U\right) - \left(X + |E_K|_{M_K} - \alpha_K M_K\right)\right|_{M_U} = \\ = \left||E_U|_{M_U} - |E_K|_{M_K} + \alpha_K M_K\right|_{M_U} \tag{2.30}$$

$$\delta_2 = \left||X|_{M_V} - \left(X + |E_K|_{M_K} - \alpha_K M_K\right)\right|_{M_V} = \left|\alpha_K M_K - |E_K|_{M_K}\right|_{M_V} \tag{2.31}$$

$$\delta_3 = \left||X|_{M_V} - \left(X + |E_U|_{M_U} - \alpha_U M_U\right)\right|_{M_V} = \left|\alpha_U M_U - |E_U|_{M_U}\right|_{M_V} \tag{2.32}$$

MO5:

$$\delta_1 = \left|X - \left(X + |E_K|_{M_K} - \alpha_K M_K\right)\right|_{M_U} = \left|\alpha_K M_K - |E_K|_{M_K}\right|_{M_U} \tag{2.33}$$

$$\delta_2 = \left|\left(|X|_{M_V} + |E_V|_{M_V} - \alpha_V M_V\right) - \left(X + |E_K|_{M_K} - \alpha_K M_K\right)\right|_{M_V} = \\ = \left||E_V|_{M_V} - |E_K|_{M_K} + \alpha_K M_K\right|_{M_V} \tag{2.34}$$

$$\delta_3 = \left|\left(|X|_{M_V} + |E_V|_{M_V} - \alpha_V M_V\right) - X\right|_{M_V} = |E_V|_{M_V} \tag{2.35}$$

MO6:

$$\delta_1 = \left|\left(X + |E_U|_{M_U} - \alpha_U M_U\right) - X\right|_{M_U} = |E_U|_{M_U} \tag{2.36}$$

$$\delta_2 = \left||X|_{M_V} - \left(X + |E_V|_{M_V} - \alpha_V M_V\right)\right|_{M_V} = |E_V|_{M_V} \tag{2.37}$$

$$\delta_3 = \left|\left(|X|_{M_V} + |E_V|_{M_V} - \alpha_V M_V\right) - \left(X + |E_U|_{M_U} - \alpha_U M_U\right)\right|_{M_V} = \\ = \left||E_V|_{M_V} - |E_U|_{M_U} + \alpha_U M_U\right|_{M_V} \tag{2.38}$$

MO7:

$$\delta_1 = \left|\left(X + |E_U|_{M_U} - \alpha_U M_U\right) - \left(X + |E_K|_{M_K} - \alpha_K M_K\right)\right|_{M_U} = \\ = \left||E_U|_{M_U} - |E_K|_{M_K} + \alpha_K M_K\right|_{M_U} \tag{2.39}$$

$$\delta_2 = \left|\left(|X|_{M_V} + |E_V|_{M_V} - \alpha_V M_V\right) - \left(X + |E_K|_{M_K} - \alpha_K M_K\right)\right|_{M_V} = \\ = \left||E_V|_{M_V} - |E_K|_{M_K} + \alpha_K M_K\right|_{M_V} \tag{2.40}$$

$$\delta_3 = \left|\left(|X|_{M_V} + |E_V|_{M_V} - \alpha_V M_V\right) - \left(X + |E_U|_{M_U} - \alpha_U M_U\right)\right|_{M_V} = \\ = \left||E_V|_{M_V} - |E_U|_{M_U} + \alpha_U M_U\right|_{M_V} \tag{2.41}$$

2.2.2 Обнаружение и исправления ошибок

Из (2.21)-(2.41) существует девять различных выражений синдрома, и каждый синдром может принимать три из этих девяти выражений, как показано в таблице 1.

Таблица 1 – Возможные выражения синдрома для категорий местоположения ошибок МО1-МО7

δ	Совпадающие значения	Векторы ошибок			Категории
		E_K	E_U	E_V	
δ_1	$v_1 = \left\lvert \lvert E_U \rvert_{M_U} - \lvert E_K \rvert_{M_K} + \alpha_K M_K \right\rvert_{M_U}$	✓	✓		МО4, МО7
	$v_2 = \left\lvert \alpha_K M_K - \lvert E_K \rvert_{M_K} \right\rvert_{M_U}$	✓			МО1, МО5
	$v_3 = \lvert E_U \rvert_{M_U}$		✓		МО2, МО6
δ_2	$v_4 = \left\lvert \alpha_K M_K - \lvert E_K \rvert_{M_K} \right\rvert_{M_V}$	✓			МО1, МО4
	$v_5 = \left\lvert \lvert E_V \rvert_{M_V} - \lvert E_K \rvert_{M_K} + \alpha_K M_K \right\rvert_{M_V}$	✓		✓	МО5, МО7
	$v_6 = \lvert E_V \rvert_{M_V}$			✓	МО3, МО6
δ_3	$v_7 = \left\lvert \alpha_U M_U - \lvert E_U \rvert_{M_U} \right\rvert_{M_V}$		✓		МО2, МО4
	$v_8 = \lvert E_V \rvert_{M_V}$			✓	МО3, МО5
	$v_9 = \left\lvert \lvert E_V \rvert_{M_V} - \lvert E_U \rvert_{M_U} + \alpha_U M_U \right\rvert_{M_V}$		✓	✓	МО6, МО7

В таблице 2 показано сопоставление категорий местоположения ошибок с выражениями трёх синдромов. Каждое значение синдрома в столбце должно соответствовать вектору ошибки в категории местоположения ошибки столбца, и каждый вектор ошибки в категории местоположения ошибки также должен быть сопоставлен по крайней мере с одним из трёх значений синдрома в столбце.

Таблица 2 – Отображение категорий местоположения ошибок в синдромы

	МО1	МО2	МО3	МО4	МО5	МО6	МО7
δ_1	v_2	v_3	0	v_1	v_2	v_3	v_1
δ_2	v_4	0	v_6	v_4	v_5	v_6	v_5
δ_3	0	v_7	v_8	v_7	v_8	v_9	v_9

Из таблиц 1 и 2 построены шесть справочных таблиц. В каждой таблице поиска значения для векторов ошибок предварительно вычисляются и сохраняются вместе с их векторами ошибок. Если ключ поиска в таблице поиска соответствует значению поиска, будут извлечены векторы ошибок, соответствующие значению поиска. В таблице 3 показаны типы величин (E_K, E_U, E_V) ошибок, соответствующие векторам ошибок, сохранённым в каждой справочной таблице, значениям поиска (вычисленные с v_1 по v_9) и поисковому ключу (δ_1, δ_2 или δ_3), используемому для идентификации категорий местоположения ошибок (МО1-МО7), и поиск векторов ошибок. Например, из первой строки таблицы 3 можно сделать вывод, что T_1 создан для извлечения вектора ошибок E_K для МО1, МО4 и МО5. Значения поиска, v_2 и v_4, для T_1 вычисляются для всех возможных комбинаций $\alpha_K \in \{0,1\}$ и E_K для t или менее ошибок остаточных цифр. Если ошибка идентифицирована как находящаяся в МО1 или МО5 (или МО1 или МО4), её вектор ошибок E_K может быть извлечен из T_1 с помощью синдрома δ_1 (или δ_2). Точный вектор ошибки восстанавливается в два этапа. Категория местоположения ошибки должна быть идентифицирована до того, как соответствующий вектор ошибок, извлечённый из таблицы поиска, будет использован для исправления ошибочных цифр остатков.

Таблица 3 – Характеристики таблиц поиска для поиска векторов ошибок

Таблица	Векторы ошибок			Значение подстановки	Ключ поиска	Группы расположения ошибок
	E_K	E_U	E_V			
T_1	✓			v_2	δ_1	МО1, МО5
				v_4	δ_2	МО1, МО4
T_2		✓		v_3	δ_1	МО2, МО6
				v_7	δ_3	МО2, МО4
T_3			✓	v_6	δ_2	МО3, МО6
				v_8	δ_3	МО3, МО5
T_4	✓	✓		v_1	δ_1	МО4, МО7
T_5	✓		✓	v_5	δ_2	МО5, МО7
T_6		✓	✓	v_9	δ_3	МО6, МО7

Если только один из трёх синдромов равен нулю, как указано в таблице 2, категория местоположения ошибки может быть идентифицирована нулевым синдромом. Это МО1 для $\delta_3 = 0$, МО2 для $\delta_2 = 0$ и МО3 для $\delta_1 = 0$. Векторы ошибок для МО1, МО2 и МО3 могут быть получены из таблиц поиска T_1, T_2 и T_3 соответственно с использованием любого из двух ненулевых синдромов. Если ни один из синдромов не равен нулю, категория местоположения ошибки может быть МО4, МО5, МО6 или МО7, как указано в таблице 2. В этом случае фактическая категория местоположения ошибки может быть определена путём проверки согласованности векторов ошибок, полученных δ_1, δ_2 и δ_3, в соответствии с таблицей 3. Например, поскольку $E_{KV} = E_K \cup E_V$, если ошибки попадают в МО5, вектор ошибок E_{KV}, полученный из T_5 на δ_2, должен включать векторы ошибок E_K и E_V, полученные из T_1 и T_3 на δ_1 и δ_3 соответственно. После определения местоположения фактической категории местоположения ошибки ошибочные цифры остатка исправляются путём вычитания полученного вектора ошибки из полученных цифр остатка.

Пусть $E = \mathrm{find}(T, \delta)$ – функция поиска таблицы, которая возвращает вектор ошибок E для ключа поиска δ в таблицу поиска T или ноль, если соответствующая запись не найдена. Шаги, необходимые для обнаружения и исправления ошибок с несколькими цифрами остатка:

1. Декодировать $\tilde{X}_K$, $\tilde{X}_U$ и $\tilde{X}_V$ из полученных цифр остатка.
2. Вычислить δ_1, δ_2 и δ_3 по (2.18), (2.19) и (2.20) соответственно.
3. Если $\delta_1 = \delta_2 = \delta_3$, ошибки нет, то перейти к шагу 10.
4. Если только один из δ_1, δ_2 и δ_3 не равен нулю, то существует более t ошибок разряда остатка. Перейти к шагу 10.
5. Если $\delta_1 \neq 0$, $\delta_2 \neq 0$ и $\delta_3 \neq 0$, то ошибочная цифра(ы) остатка находятся в МО1. $E_{\mathrm{MO1}} = \mathrm{find}(T_1, \delta_2)$. Вычитание вектора ошибки E_{MO1} из полученной цифры остатка для исправления ошибки. Перейти к шагу 10.
6. Если $\delta_1 \neq 0$, $\delta_2 = 0$ и $\delta_3 \neq 0$, то ошибочная цифра остатка находится в МО2. $E_{\mathrm{MO2}} = \mathrm{find}(T_2, \delta_1)$. Вычитание вектора ошибки E_{MO2} из полученной цифры остатка для исправления ошибки. Перейти к шагу 10.
7. Если $\delta_1 = 0$, $\delta_2 \neq 0$ и $\delta_3 \neq 0$, то ошибочная цифра остатка находится в МО3. $E_{\mathrm{MO3}} = \mathrm{find}(T_3, \delta_2 \text{ или } \delta_3)$. Вычитание вектора ошибки E_{MO3} из полученной цифры остатка для исправления ошибки. Перейти к шагу 10.
8. Если $\delta_1 \neq 0$, $\delta_2 \neq 0$ и $\delta_3 \neq 0$, ошибочные цифры остатка могут быть в МО4, МО5, МО6 или МО7. Точная категория местоположения ошибки определяется следующим образом:
   8.1. Если $E_{\mathrm{MO4}} = \mathrm{find}(T_4, \delta_1) = \mathrm{find}(T_1, \delta_2) \cup \mathrm{find}(T_2, \delta_3) \neq 0$, ошибка находится в МО4. Установить $E = E_{\mathrm{MO4}}$.
   8.2. Если $E_{\mathrm{MO5}} = \mathrm{find}(T_5, \delta_2) = \mathrm{find}(T_1, \delta_1) \cup \mathrm{find}(T_3, \delta_3) \neq 0$, ошибка находится в МО5. Установить $E = E_{\mathrm{MO5}}$.
   8.3. Если $E_{\mathrm{MO6}} = \mathrm{find}(T_6, \delta_3) = \mathrm{find}(T_2, \delta_1) \cup \mathrm{find}(T_3, \delta_2) \neq 0$, ошибка находится в МО6. Установить $E = E_{\mathrm{MO6}}$.

8.4. Если найти $\mathrm{find}(T_4, \delta_1) \cap \mathrm{find}(T_5, \delta_2) \neq 0$ или $\mathrm{find}(T_4, \delta_1) \cap \mathrm{find}(T_6, \delta_3) \neq 0$, ошибка находится в МО7. Множество $E = E_{\mathrm{MO7}} = \mathrm{find}(T_4, \delta_1) \cup \mathrm{find}(T_5, \delta_2)$.

9. Если не найдено ни одной или более одной категории расположения ошибок, то существует более t ошибок цифр остатка. Перейти к шагу 10. В противном случае вычитают вектор ошибки E из полученной цифры(цифр) остатка для исправления ошибочной цифры остатка.
10. Завершить процедуру.

На шаге 8 к каждой таблице поиска обращаются не более трёх различных синдромов. К одному и тому же синдрому могут обращаться одновременно разные таблицы поиска, но разные синдромы не могут одновременно обращаться к одной и той же таблице поиска. Таким образом, поиск вектора ошибок может быть завершён не более чем за три цикла поиска. Поскольку ключами поиска в этих таблицах поиска являются три синдрома размеров M_U и M_V, размеры и скорость доступа к таблицам поиска становятся зависимыми от количества и размеров модулей ИСОК. Эту проблему можно решить, используя многоуровневые таблицы поиска с меньшей шириной входного адреса и менее сложными схемами декодирования адресов.

Блок-схема описанной выше процедуры изображена на рисунке 2.1. Следует отметить, что МО7 возможно только в том случае, если имеются как минимум три ошибки цифры остатка. Следовательно, для обнаружения и исправления двузначных ошибок остатка этап идентификации МО7 может быть опущен.

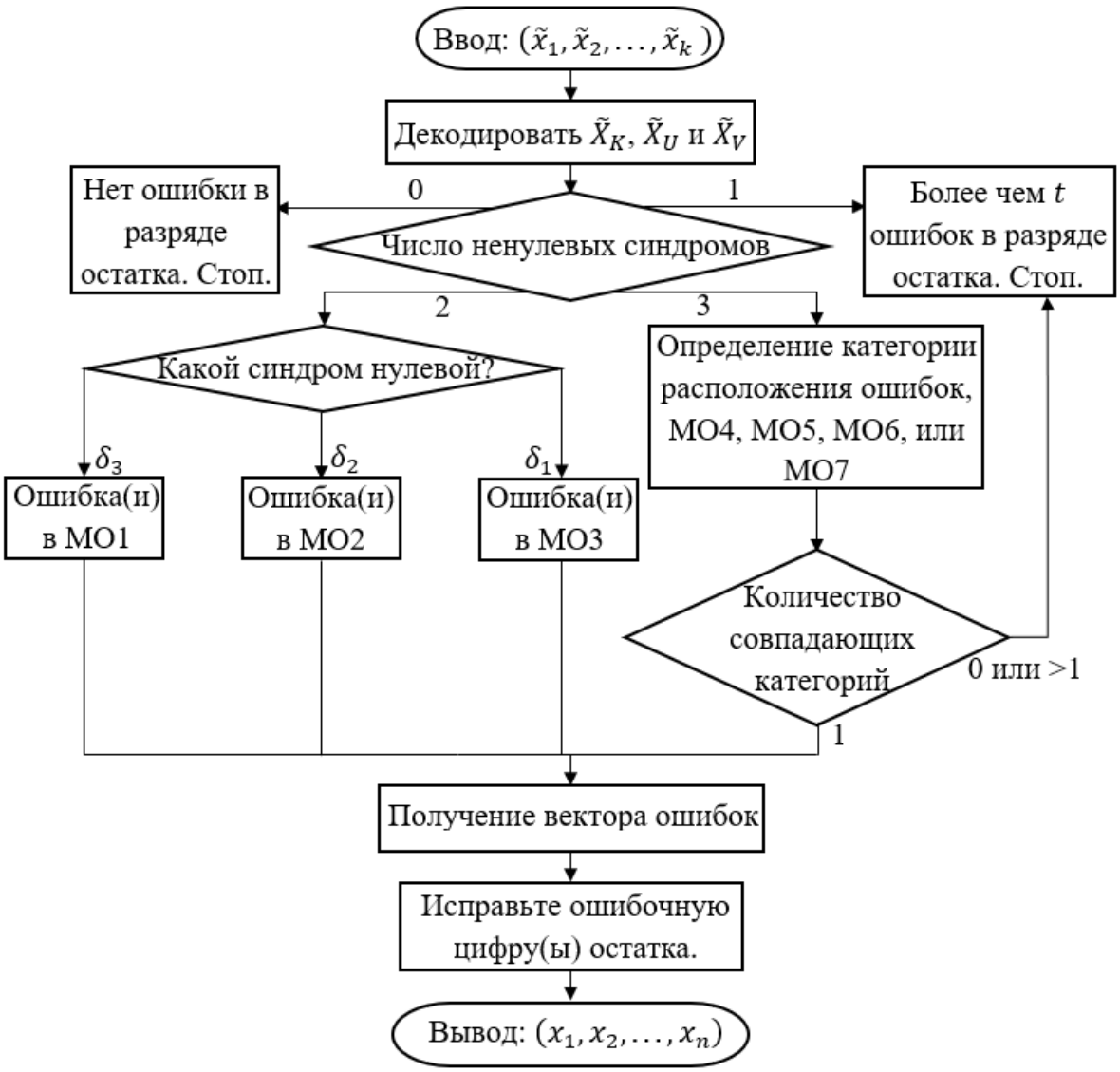

Рисунок 2.1 – Предложенная процедура обнаружения и исправления ошибок нескольких разрядов остатка

2.2.3 Пример

Рассмотрим набор модулей $\{7, 9, 11, 13, 17, 19, 23, 25, 29\}$, где $\{7, 9, 11\}$ – информационные модули, а $\{13, 17, 19, 23, 25, 29\}$ – избыточные модули. Таким образом, $M_K = 693$, $M_U = 4199$ и $M_V = 16675$. Поскольку количество избыточных модулей равно 6, этот ИСОК способен исправлять три ошибки остаточных цифр.

Пусть безошибочное представление числа будет $X = 246 \equiv (1, 3, 4, 12, 8, 18, 16, 21, 14)$. Предположим, что в X введены три ошибки цифр остатка, а вектор ошибок равен $(0, 2, 0, 0, 0, 6, 7, 0, 0)$. $E_l = 2{,}307{,}079{,}775$.

$$\tilde{X} = |X + E_l|_{M_N} = 2{,}307{,}080{,}021 \equiv (1, 5, 4, 12, 8, 5, 0, 21, 14).$$

Следовательно, $\tilde{X}_K \equiv (1, 5, 4) = 554$, $\tilde{X}_U \equiv (12, 8, 5) = 2456$ и $\tilde{X}_V \equiv (0, 21, 14) = 10396$. Используя выражения (2.19)-(2.21), три синдрома вычисляются как

$$\delta_1 = |2456 - 554|_{4199} = 1902,$$
$$\delta_2 = |10396 - 554|_{16675} = 9842,$$
$$\delta_3 = |10396 - 2456|_{16675} = 7940.$$

Поскольку $\delta_1 \neq 0$, $\delta_2 \neq 0$ и $\delta_3 \neq 0$, категория местоположения ошибки – любая из групп МО4, МО5, МО6 или МО7.

Из вектора ошибок E_l,

$$E_K \equiv (0, 2, 0, 0, 0, 0, 0, 0, 0) = 26{,}957{,}055{,}125,$$
$$E_U \equiv (0, 0, 0, 0, 0, 6, 0, 0, 0) = 28{,}092{,}089{,}025,$$
$$E_V \equiv (0, 0, 0, 0, 0, 0, 7, 0, 0) = 44{,}303{,}334{,}075.$$

Следовательно, $|E_K|_{M_K} = 308$, $|E_U|_{M_U} = 2210$ и $|E_V|_{M_V} = 10150$. Со ссылкой на таблицу 1, возможные значения синдрома:

$v_1 = |2210 - 308|_{4199} = 1902$ для $\alpha_K = 0$ и $|1902 + 693|_{4199} = 2595$ для $\alpha_K = 1$;

$v_2 = |-308|_{4199} = 3891$ для $\alpha_K = 0$ и $|693 - 308|_{4199} = 385$ для $\alpha_K = 1$;

$v_3 = 2210$;

$v_4 = |-308|_{16675} = 16367$ для $\alpha_K = 0$ и $|693 - 308|_{16675} = 385$ для $\alpha_K = 1$;

$v_5 = |10150 - 308|_{16675} = 9842$ для $\alpha_K = 0$ и $|9842 + 693|_{16675} = 10535$ для $\alpha_K = 1$;

$v_6 = 10150$;

$v_7 = |-2210|_{16675} = 14465$ для $\alpha_U = 0$ и $|4199 - 2210|_{16675} = 1989$ для $\alpha_U = 1$;

$v_8 = 10150$; $v_9 = |10150 - 2210|_{16675} = 7940$ для $\alpha_U = 0$ и $|10150 - 2210 + 4199|_{16675} = 12139$ для $\alpha_U = 1$.

Эти значения соответствуют МО4-МО7 и приведены в таблице 4.

Таблица 4 – Значения синдрома для МО4-МО7

	МО4	МО5	МО6	МО7
δ_1	1902 или 2595	3891 или 385	2210	1902 или 2595
δ_2	16387 или 385	9842 или 10535	10150	9842 или 10535
δ_3	14465 или 1989	10150	7940 или 12139	7940 или 12139

Из таблицы 4 видно, что величины $\delta_1 = 1902$, $\delta_2 = 9842$ и $\delta_3 = 7940$ вычисленные непосредственно из полученного представления остатка с помощью (2.15)-(2.17), соответствуют значениям, вычисленным с использованием таблицы 1 для МО7 в таблицах поиска. Следовательно, категория местоположения ошибки – МО7.

Поскольку $\mathrm{find}(T_4, \delta_1 = 1902) = (0,2,0,0,0,6,0,0,0)$ и $\mathrm{find}(T_5, \delta_2 = 9842) = (0,2,0,0,0,0,7,0,0)$, то

$$\mathrm{find}(T_4, \delta_1 = 1902) \cap \mathrm{find}(T_5, \delta_2 = 9842) = (0,2,0,0,0,0,0,0,0) \neq 0.$$

Следовательно, вектор ошибки

$$E = E_{\mathrm{MO7}} \equiv (0,2,0,0,0,6,0,0,0) \cup (0,2,0,0,0,0,7,0,0) = (0,2,0,0,0,6,7,0,0)$$

и

$$\tilde{X} - E = (1,5,4,12,8,5,0,21,14) - (0,2,0,0,0,6,7,0,0) =$$
$$(1,3,4,12,8,-1,-7,21,14) = (1,3,4,12,8,18,16,21,14) \equiv X.$$

2.2.4 Доказательство корректности метода исправления ошибок на базе синдрома

Предлагаемая процедура предполагает, что каждое значение синдрома представляет некоторые комбинации ошибок разрядов остатка в

соответствующей группе разрядов остатка, $\tilde{X}_K$, $\tilde{X}_U$ или $\tilde{X}_V$, так что векторы ошибок, извлечённые из таблиц поиска, T_1, T_2 и T_3, одним из двух ненулевых синдромов на шагах с 5 по 7 являются уникальными и однозначными, когда имеется t или меньше ошибок разрядов. Поскольку в расчётах трёх синдромов в (2.15)-(2.17) используются либо операция по модулю M_U, либо операция по модулю M_V, δ_1, δ_2 и δ_3 ограничены $[0, M_U - 1]$, $[0, M_V - 1]$ и $[0, M_V - 1]$, соответственно. Для обеспечения единственности представления синдрома избыточные модули должны быть выбраны таким образом, чтобы M_U и M_V были больше числа возможных векторов ошибок.

Цифра ошибки e_i канала по модулю m_i ограничена $[1, m_i - 1]$. Поэтому число различных комбинаций одиночных разрядов остатка ошибки в канале по модулю m_i равно $(m_i - 1)$. Аналогично, число векторов ошибок из-за множественных ошибок остаточных разрядов может быть вычислено произведением числа векторов ошибок из-за одноразрядной ошибки в каждом из ошибочных каналов по модулю.

Число векторов ошибок для случая ошибки одного разряда остатка в q модулях задаётся:

$$S_1 = \sum_{l_1=1}^{q} \left(m_{l_1} - 1\right) = \sum_{l_1=1}^{q} m_{l_1} - C_1^q \tag{2.42}$$

Число векторов ошибок, связанных с ошибками с двумя остаточными числами в q модулях, равно

$$S_2 = \sum_{l_2=1}^{q-1} \sum_{l_1=l_2+1}^{q} \left(m_{l_2} - 1\right)\left(m_{l_1} - 1\right) =$$

$$= \sum_{l_2=1}^{q-1} \sum_{l_1=l_2+1}^{q} \left(m_{l_2} m_{l_1} - m_{l_2} - m_{l_1} + 1\right) =$$

$$= \sum_{l_2=1}^{q-1} \sum_{l_1=l_2+1}^{q} m_{l_2} m_{l_1} - \sum_{l_2=1}^{q-1} \sum_{l_1=l_2+1}^{q} \left(m_{l_2} + m_{l_1}\right) + C_2^q = \tag{2.43}$$

$$= \sum_{l_2=1}^{q-1} \sum_{l_1=l_2+1}^{q} m_{l_2} m_{l_1} - C_2^{q-1} \sum_{l_1=1}^{q} m_{l_1} + C_2^q$$

Число векторов ошибок, связанных с ошибками в трёхзначном числе остатков в q модулях, равно

$$S_3 = \sum_{l_3=1}^{q-2} \sum_{l_2=l_3+1}^{q-1} \sum_{l_1=l_2+1}^{q} \left(m_{l_3} - 1\right)\left(m_{l_2} - 1\right)\left(m_{l_1} - 1\right) =$$

$$= \sum_{l_3=1}^{q-2} \sum_{l_2=l_3+1}^{q-1} \sum_{l_1=l_2+1}^{q} m_{l_3} m_{l_2} m_{l_1} - \tag{2.44}$$

$$- C_1^{q-2} \sum_{l_2=1}^{q-1} \sum_{l_1=l_2+1}^{q} m_{l_2} m_{l_1} + C_2^{q-1} \sum_{l_1=1}^{q} m_{l_1} - C_3^q$$

В общем случае, число векторов ошибок, связанных с ошибками t остатков в q модулях, равно

$$S_t = \sum_{l_t=1}^{q-t+1} \dots \sum_{l_1=l_2+1}^{q} \left(m_{l_t} m_{l_{t-1}} \dots m_{l_1}\right) + \cdots$$

$$+ (-1)^{t-1} C_{t-1}^{q-1} \sum_{l_1=1}^{q} m_{l_1} + + (-1)^t C_t^q \tag{2.45}$$

Общее количество векторов ошибок из-за t или менее ошибок в остаточных цифрах в q модулях равно

$$S_{q,t} = \sum_{i=1}^{t} S_i. \tag{2.46}$$

Когда категория местоположения ошибки равна МО1, общее число векторов ошибок из-за t или менее ошибок разрядов остатка в k модулях $\tilde{X}_K$ может быть вычислено по (2.46) следующим образом:

$$\begin{aligned} S_{\text{МО1}} = S_{k,t} = \sum_{i=1}^{t} S_i = \\ = \sum_{l_1=1}^{k} \left(m_{l_1} - 1\right) + \sum_{l_2=1}^{k-1} \sum_{l_1=l_2+1}^{k} \left(m_{l_2} - 1\right)\left(m_{l_1} - 1\right) \\ + \cdots + \sum_{l_t=1}^{k-t+1} \ldots \sum_{l_1=l_2+1}^{k} \left(m_{l_t} - 1\right) \ldots \left(m_{l_1} - 1\right) \end{aligned} \tag{2.47}$$

Из (2.22), значения синдрома в таблице подстановки T_1, которые соответствуют всем этим $S_{\text{МО1}}$, число ошибочных векторов вычисляется по $\left|\alpha_K M_K - |E_k|_{M_K}\right|_{M_V}$, где $\alpha_K \in \{0, 1\}$. Из-за смещения M_K, M_V должно быть более чем в два раза больше числа возможных векторов ошибок, чтобы гарантировать, что каждый вектор ошибок из-за ошибок остатка в МО1 может быть однозначно извлечён из T_1 с помощью δ_2, т. е.,

$$M_V > 2 \times S_{\text{МО1}} \tag{2.48}$$

Когда категория местоположения ошибок – МО2, общее число комбинаций из t или менее ошибок остаточных цифр в t модулях $\tilde{X}_U$ может быть вычислено с помощью (2.46) следующим образом:

$$\begin{aligned} S_{\text{МО2}} = S_{t,t} = \sum_{l_1=1}^{t} \left(m_{l_1} - 1\right) + \sum_{l_2=1}^{t-1} \sum_{l_1=l_2+1}^{t} \left(m_{l_2} - 1\right)\left(m_{l_1} - 1\right) + \cdots \\ + \sum_{l_t=1}^{1} \ldots \sum_{l_1=t}^{t} \left(m_{l_t} - 1\right) \ldots \left(m_{l_1} - 1\right) \end{aligned} \tag{2.49}$$

При упрощении (2.49) становится

$$S_{t,t} = \left(m_{l_t} m_{l_{t-1}} \dots m_{l_1}\right) - 1 = \prod_{i=l_1}^{l_t} m_i - 1. \quad (2.50)$$

Поскольку индекс i для m_i находится в диапазоне от $k+1$ до $k+t$ для модулей в $\tilde{X}_U$, (2.50) можно выразить как:

$$S_{\text{MO2}} = S_{t,t} = \prod_{i=k+1}^{k+t} m_i - 1 = M_U - 1 \quad (2.51)$$

Из (2.24) значения синдрома в таблице поиска T_2, которые соответствуют всем этим векторам ошибок S_{MO2}, вычисляются $|E_U|_{M_U}$. Чтобы гарантировать, что каждый возможный вектор ошибок в MO2 может быть однозначно получен δ_1, $M_U > S_{\text{MO2}}$, который всегда удовлетворяется.

Для категории местоположения ошибок MO3 общее число векторов ошибок из-за t или меньших ошибок цифр остатка в t модулях $\tilde{X}_V$ может быть вычислено с использованием (2.50) как:

$$S_{\text{MO3}} = \prod_{i=k+t+1}^{k+2t} m_i - 1 = M_V - 1 \quad (2.52)$$

Эти S_{MO3} числа векторов ошибок из таблицы поиска T_3 могут быть получены с помощью $\delta_2 = |E_V|_{M_V}$ из (2.28) и (2.29). После того же аргумента $M_V > S_{\text{MO3}}$, который также всегда выполняется.

Когда все три синдрома ненулевые, категория местоположения ошибки может быть MO4, MO5, MO6 или MO7. Чтобы избежать двусмысленности, только один уникальный вектор ошибки одной категории местоположения ошибки может быть получен набором из трёх синдромов. Пусть δ_{1_i}, δ_{2_i} и δ_{3_i} – синдромы, использованные для идентификации вектора ошибки с одной ошибкой, категории размещения δ_{1_j}, δ_{2_j} и δ_{3_j} будет другой набор синдромов, которые используются для определения вектора ошибки в другую категорию из MO4, MO5, MO6 и MO7. Из-за операций по модулю неоднозначность в обнаружении ошибок может возникнуть, когда

$$\delta_{1_i} - \delta_{1_j} = 0, \tag{2.53}$$

$$\delta_{2_i} - \delta_{2_j} = 0, \tag{2.54}$$

$$\delta_{3_i} - \delta_{3_j} = 0, \tag{2.55}$$

Из таблицы 2, противоречие происходит между МО4 и МО5 тогда и только тогда, когда $v_1 - v_2 = 0$, $v_4 - v_5 = 0$ и $v_7 - v_6 = 0$. Подставляя (2.30) и (2.33) в (2.53), (2.31) и (2.34) в (2.54), (2.32) и (2.35) в (2.55), мы получаем

$$\left|E_{U_i}\right|_{M_U} - \left|E_{K_i}\right|_{M_K} + \left|E_{K_j}\right|_{M_K} + \left(\alpha_{K_i} - \alpha_{K_j}\right) M_K = \beta_1 M_U, \tag{2.56}$$

$$-\left|E_{K_i}\right|_{M_K} - \left|E_{V_j}\right|_{M_V} + \left|E_{K_j}\right|_{M_K} + \left(\alpha_{K_i} - \alpha_{K_j}\right) M_K = \beta_2 M_V, \tag{2.57}$$

$$\alpha_{U_i} M_U - \left|E_{U_i}\right|_{M_U} - \left|E_{V_j}\right|_{M_V} = \beta_3 M_V, \tag{2.58}$$

где

$$\beta_1 = \left\lfloor \frac{\left|E_{U_i}\right|_{M_U} - \left|E_{K_i}\right|_{M_K} + \alpha_{K_i} M_K}{M_U} \right\rfloor - \left\lfloor \frac{\alpha_{K_j} M_K - \left|E_{K_j}\right|_{M_K}}{M_U} \right\rfloor, \tag{2.59}$$

$$\beta_1 = \left\lfloor \frac{\alpha_{K_i} M_K - \left|E_{K_i}\right|_{M_K}}{M_V} \right\rfloor - \left\lfloor \frac{\left|E_{V_j}\right|_{M_V} - \left|E_{K_j}\right|_{M_K} + \alpha_{K_j} M_K}{M_V} \right\rfloor, \tag{2.60}$$

$$\beta_3 = \left\lfloor \frac{\alpha_{U_i} M_U - \left|E_{U_i}\right|_{M_U}}{M_V} \right\rfloor. \tag{2.61}$$

Вычитая (2.56) и (2.58) из (2.57), имеем,

$$\begin{gathered} -\alpha_{U_i} M_U = \beta_2 M_V - \beta_1 M_U - \beta_3 M_V, \\ \left(\beta_1 - \alpha_{U_i}\right) M_U = (\beta_2 - \beta_3) M_V, \\ \frac{M_U}{M_V} = \frac{\beta_2 - \beta_3}{\beta_1 - \alpha_{U_i}}. \end{gathered} \tag{2.62}$$

Поскольку M_V и M_U относительно простые, (2.62) является правильным тогда и только тогда, когда $M_U = \beta_2 - \beta_3$ и $M_V = \beta_1 - \alpha_{U_i}$. Справедливость (2.7) требует, чтобы $M_U > M_K$. Таким образом, β_1 в (2.59) является целым числом в диапазоне $[0, 1]$. Поскольку $\alpha_{U_i} \in \{0, 1\}$, диапазон $\beta_1 - \alpha_{U_i}$ равен $[-1,1]$. M_V

является произведением t избыточных модулей, и оно всегда будет больше 2. Поскольку $M_V \neq \beta_1 - \alpha_{U_i}$, (2.62) недопустимо, набор синдромов, используемых для получения вектора ошибок в МО4, никогда не столкнётся с этим в МО5. Подобное доказательство может быть получено, чтобы подтвердить, что нет никакого противоречия между двумя наборами синдромов в любых других парах категорий местоположения ошибки.

Таким образом, предлагаемая процедура способна идентифицировать и исправлять до t ошибочных цифр остатка при выполнении следующих двух условий:

Условие 1: $M_V > 2 \times S_{\mathrm{MO1}}$;

Условие 2: $M_U > M_K$.

Для примера, так как $M_U = 4199$ больше, чем $M_K = 693$, выполняется условие 2. Используя (2.47), S_{MO1} получаем равным 692. Условие 1 выполняется также потому, что $M_V > 2 \times S_{\mathrm{MO1}} = 1384$. В таблице 4 также отмечается, что условия столкновения (2.53)-(2.55) не удовлетворяются наборами значений синдрома между любыми двумя столбцами.

Для высоконадёжной вычислительной системы, где $t = k$, одним из простых способов выполнения условий 1 и 2 является выбор модулей из простых чисел с сопоставимыми длинами слов. В противном случае, когда $t < k$, избыточные модули могут быть выбраны из дополнительного пула взаимно простых чисел в формах 2^n и $2^n \pm 1$, которые имеют большую длину слова, чем информационные модули, так что условия 1 и 2 всегда выполняются. Поскольку арифметические операции по модулю 2^n и $2^n \pm 1$ являются более эффективными для аппаратной реализации, различия в задержке между информационными и избыточными каналами модуля не будут значительными.

2.3 Выводы по второй главе

1. Представлен алгоритм обнаружения и исправления многозначных ошибок вычетов с фиксированным числом вычислений синдрома независимо от размера набора модулей.

2. Критерии уникальности могут быть выполнены путём выбора модулей из множества простых чисел, чтобы удовлетворить желаемую возможность исправления ошибок. Расширенная версия этого алгоритма с увеличением числа синдромов в зависимости от количества информационных модулей также была предложена для снятия ограничения, накладываемого на размер избыточных модулей.

3. Идентификация местоположения ошибки и поиск вектора ошибки требуют только таблиц поиска и не требуют арифметических операций. Чтобы минимизировать избыточное пространство, также предлагается расширенный алгоритм, в котором число синдромов и справочных таблиц увеличивается с увеличением количества информационных модулей, но места ошибок по-прежнему можно идентифицировать, не требуя итерационных вычислений.

3 Модификация методов обнаружения локализации и исправления ошибок

3.1 Модификация метода синдрома в СОК

Алгоритм, предложенный в разделе 2.2, может обнаруживать и исправлять множественные ошибки вычетов с фиксированным числом вычислений синдрома независимо от размера набора модулей. Стоимость детерминированной границы во времени вычисления является требованием большего избыточного пространства из-за условия 2. Это условие может быть полностью устранено путём увеличения числа вычислений синдрома и таблиц поиска в соответствии с числом информационных модулей.

Для устранения условия 2, полученные информационные остатки делятся на $\lceil k/t \rceil$ группы из t остатков, каждая из которых, кроме последней группы, может иметь менее t остатков, то есть $\tilde{X}_K^{(1)} \equiv (\tilde{x}_1, \tilde{x}_2, \dots, \tilde{x}_t)$, $\tilde{X}_K^{(2)} \equiv (\tilde{x}_{t+1}, \tilde{x}_{t+2}, \dots, \tilde{x}_{2t}), \dots, \tilde{X}_K^{(\lceil k/t \rceil)} \equiv \left(\tilde{x}_{(\lceil k/t \rceil - 1)t+1}, \tilde{x}_{(\lceil k/t \rceil - 1)t+2}, \dots, \tilde{x}_k\right)$. Включая две избыточные группы вычетов, $\tilde{X}_U$ и $\tilde{X}_V$, всего существует $\lceil k/t \rceil + 2$ группы вычетов. Исключая одну из групп остатков $\lceil k/t \rceil + 2$ в каждом вычислении синдрома, необходимо вычислить синдромы $\lceil k/t \rceil + 2$, и любые ошибки, возникшие в каждой группе цифр остатков, могут быть обнаружены путём проверки, если один из вычисленных синдромов – ноль.

Пусть $\tilde{X}_{UK,(i)} \equiv \left(\tilde{X}_U,\ \tilde{X}_K^{(1)}, \dots, \tilde{X}_K^{(i-1)}, \tilde{X}_K^{(i+1)}, \dots, \tilde{X}_K^{(\lceil k/t \rceil)}\right)$ обозначает величину, обусловленную всеми остатками, кроме $\tilde{X}_K^{(i)}$ в $\tilde{X}_U$ и $\tilde{X}_K$. В этом случае $M_K = \prod_{i=1}^{k} m_i$, $M_K^{(i)} = \prod_{j=1}^{t} m_{(i-1)+j}$ и $M_{UK,(i)} = \frac{M_U \times M_K}{M_K^{(i)}}$. Кроме того, пусть $E_K^{(i)} \equiv \left(e_{k_1}, e_{k_2}, \dots, e_{k_j}, 0, \dots, 0\right)$ обозначает величину вектора ошибок, введенного в $\tilde{X}_K^{(i)}$, где $k_1, k_2, \dots, k_j$ для $j \le t$ представляют позиции цифр ошибок. Аналогично, $E_{UK,(i)} \equiv \left(e_{k_1}, \dots, e_{k_j}, e_{u_1}, \dots, e_{u_l}, 0, \dots, 0\right)$ обозначает величину введённого вектора

ошибки в $\tilde{X}_{UK,(i)}$, где $k_1, k_2, \dots, k_j$ и $u_1, u_2, \dots, u_l$ представляют позиции цифр ошибок, а $j + l \leq t$.

$\tilde{X}_{UK,(i)}$ может быть выражено через $\tilde{X}$ как:

$$\tilde{X}_{UK,(i)} = \left|\tilde{X}\right|_{M_{UK,(i)}} \tag{3.1}$$

Когда ошибки нет, $\tilde{X} = X$ и

$$\tilde{X}_{UK,(i)} = |X|_{M_{UK,(i)}} \tag{3.2}$$

Поскольку $M_{UK,(i)} > M_K$, (3.2) можно переписать как:

$$\tilde{X}_{UK,(i)} = X \tag{3.3}$$

Когда количество ошибочных остатков меньше или равно t,

$$\tilde{X}_{UK,(i)} = \left|X + E_{UK,(i)}\right|_{M_{UK,(i)}} = X + \left|E_{UK,(i)}\right|_{M_{UK,(i)}} - \alpha_{M_{UK,(i)}} M_{UK,(i)} \tag{3.4}$$

где

$$\alpha_{M_{UK,(i)}} = \begin{cases} 0, \text{если } X + \left|E_{UK,(i)}\right|_{M_{UK,(i)}} < M_{UK,(i)} \\ 1, \text{если } X + \left|E_{UK,(i)}\right|_{M_{UK,(i)}} \geq M_{UK,(i)} \end{cases}.$$

Аналогичный подход к обнаружению и идентификации местоположений ошибочных цифр остатков может быть выполнен путём проверки векторов ошибок, извлечённых из таблиц поиска, с использованием следующих $\lceil k/t \rceil + 2$ синдромов в качестве ключей поиска для таблиц поиска.

$$\delta_1 = \left|\tilde{X}_U - \tilde{X}_K\right|_{M_U} \tag{3.5}$$

$$\delta_2 = \left|\tilde{X}_V - \tilde{X}_K\right|_{M_V} \tag{3.6}$$

$$\delta_3 = \left|\tilde{X}_V - \tilde{X}_{UK,(1)}\right|_{M_V} \tag{3.7}$$

$$\vdots$$

$$\delta_{\lceil k/t \rceil + 2} = \left|\tilde{X}_V - \tilde{X}_{UK,(\lceil k/t \rceil)}\right|_{M_V} \tag{3.8}$$

Основываясь на аналогичных выводах в (2.18)-(2.41) для δ_1 и δ_2, от δ_3 до $\delta_{\lceil k/t \rceil + 2}$ извлекаются величины ошибок из-за векторов ошибок, введённых в соответствующие группы вычетов, участвующих в вычислениях.

Процедура обнаружения и исправления ошибок по нескольким основаниям приведена ниже:

1. Декодировать $\tilde{X}_U$, $\tilde{X}_V$ и $\tilde{X}_{UK,(i)}$ для $i = [1, \lceil k/t \rceil]$ из полученных цифр остатка.
2. Вычислить δ_i для $i = [1, \lceil k/t \rceil + 2]$ по (3.5)-(3.8).
3. Если $\delta_1 = \delta_2 = \cdots = \delta_{\lceil k/t \rceil + 2} = 0$, нет ошибки остатка. Перейти к шагу 8.
4. Если только один из синдромов равен нулю, ошибочная цифра остатка находится в одной из групп остатков, $\tilde{X}_U$, $\tilde{X}_V$ или $\tilde{X}_K^{(i)}$ для $i = [1, \lceil k/t \rceil]$. Извлечь вектор ошибки из соответствующей таблицы поиска, используя один из ненулевых синдромов в качестве ключа поиска.
5. Если более чем один или меньше, чем $\lceil k/t \rceil + 2$, синдромы равны нулю, утверждают, что существует более чем t ошибочных цифр остатка. Перейти к шагу 8.
6. Если все синдромы отличны от нуля, ошибочные цифры остатков находятся в более чем одной группе остатков. Извлечь векторы ошибок из справочных таблиц, используя δ_i для $i = [1, \lceil k/t \rceil + 2]$ в качестве ключей поиска. Точные места ошибок определяются путём проверки перехвата найденных векторов ошибок.
7. Вычесть вектор ошибки из полученной цифры остатка, чтобы исправить ошибочную цифру остатка.
8. Завершить процедуру.

3.2 Выбор оснований СОК

Один и тот же рабочий диапазон может быть реализован с помощью различных наборов оснований [93]. Иногда желательно, чтобы $M = \prod_{i=1}^{n} m_i$ было максимально большим, тогда рекомендуют в качестве m_1 выбирать наибольшее простое число, соответствующее машинному слову, в качестве m_2 – наибольшее простое число, меньшее m_1 и т.д. до m_L [4, 94]. В некоторых случаях считается целесообразным один из модулей СОК выбирать чётным, что даёт

возможность осуществить разбиение выбранного диапазона $[0, M]$ на равные поддиапазоны M^+ и M^-, которые будут использоваться для представления положительных и отрицательных чисел $M^+ = \left[0; \frac{M}{2} - 1\right]$ и $M^- = \left[\frac{M}{2}; M - 1\right]$. Однако наличие чётного основания не всегда бывает удобно.

Рост диапазона влечёт за собой увеличение разрядности остатков, что ведёт к аппаратному усложнению арифметического устройства, к увеличению временных затрат на вычислительные операции. Один из возможных путей уменьшения величин оснований реализуется построением иерархической системы остаточных классов [4, 94].

Пусть $M = m_1 \cdot m_2 \cdot \ldots \cdot m_L$ – рабочий диапазон системы и $m_1 < m_2 < \cdots < m_L$. Будем называть эту систему оснований главной. Тогда при умножении остатков максимальное число, которое может быть получено в этой системе, есть $(m_L - 1)^2$. Далее предполагается представить все цифры главной системы в новой системе с основаниями $q_1, q_2, \ldots, q_k$ такими, что $Q = \prod_{i=1}^{k} q_i \geq (m_L - 1)^2$. На втором уровне системы наибольшее число из всех получаемых не сможет превысить $(q_k - 1)^2$, поэтому далее можно ввести систему третьего уровня с основаниями $r_1, r_2, \ldots, r_s$ и диапазоном $R = \prod_{i=1}^{s} r_i \geq (q_k - 1)^2$ и т.д. Такой процесс перехода к меньшим основаниям заметно упрощает реализацию элементарного арифметического устройства и время выполнения операций [4, 94]. Однако в целом избыточность всего арифметического устройства будет расти как за счёт увеличения разрядности операндов, так и за счёт увеличения избыточности с появлением каждого нового уровня [4, 94].

Таким образом, при определении модулей системы возникает очень важная задача оптимального выбора набора модулей. Выбор рационального набора модулей связан с улучшением таких параметров арифметического устройства цифрового процессора обработки сигналов, как количество вентильных ресурсов, надёжность функционирования, сквозная пропускная способность и другие. Существует большое число различных способов, позволяющих выбрать систему оснований СОК, обеспечивающую заданный

вычислительный диапазон $[0, M)$ [4, 94]. В итоге окончательный выбор системы оснований определяется конкретными требованиями к разрабатываемому цифровому процессору для модулярной арифметики. Из соображений повышения надёжности и упрощения структуры процессора наиболее целесообразной системой оснований является система, обеспечивающая минимальное количество разрядов. Под рациональным выбором оснований СОК будем понимать процесс выбора совокупности таких модулей m_i $\left(i = \overline{1{:}L}\right)$, которые позволяют получить максимальный эффект по выбранным критериям [4, 94].

Для решения задач выбора оснований СОК необходимо определить критерий оптимизации набора оснований. Одним из основных критериев является условие минимальности количества двоичных разрядов, которые используются при построении арифметического устройства. Условие минимальности можно представить в виде

$$f(m_1, m_2, \dots, m_L) = \sum_{i=1}^{L} f_i(m_i) = min, \tag{3.9}$$

где величина $f(m_1, m_2, \dots, m_L)$, называемая рангом СОК с модулями $m_1, m_2, \dots, m_L$, определяет количество разрядов, необходимое для отображения чисел из диапазона $[0, M)$. Условие (3.9) указывает на необходимость того, чтобы суммарное количество разрядов $\sum_{i=1}^{L} f_i(m_i)$, необходимое для представления остатков чисел по этим основаниям $m_1, m_2, \dots, m_L$, или оставалось неизменным, или незначительно превышало количество разрядов $f(m_1, m_2, \dots, m_L)$ при обычном позиционном q-ичном представлении чисел.

Если остатки чисел по основаниям m_i $\left(i = \overline{1, L}\right)$ заключаются в пределах $2^{k_i - 1} \leq m_i - 1 < 2^{k_i}$, то для двоичного изображения числа требуется k_i разрядов, следовательно,

$$f_i(m_i) = k_i = [\log_2(m_i - 1)] + 1. \tag{3.10}$$

Известно также, что

$$f(m) = f(m_1, m_2, \dots, m_L) - \left[\sum_{i=1}^{L} a_i\right], \tag{3.11}$$

где величина

$$a_i = \log_2(m_i - 1) - k_i + 1, \tag{3.12}$$

a_i – логарифмический дефект. Величина $\alpha = [\sum_{i=1}^{L} a_i]$ показывает количественное превышение ранга системы, составляемой из группы оснований m_i $\left(i = \overline{1, L}\right)$, над рангом системы по модулю $M = \prod_{i=1}^{L} m_i$, и является оценкой рассматриваемого критерия, суть которого заключается в следующем: оптимальный набор оснований СОК обеспечивает равенство нулю величины α.

При выборе рационального набора модулей необходимо из множества $f(m_1, m_2, \dots, m_L)$ определить такие подмножества, произведения которых не меньше заданного диапазона и имеющие минимальное значение $\sum_{i=1}^{L} f_i(m_i)$.

Рассмотрим алгоритм выбора оснований СОК при заданном числовом диапазоне [4, 94]. Путём перебора по 2, 3, 4, ... выбираем наборы оснований $m_1, m_2, \dots, m_L$ так, чтобы выполнялись следующие условия [4, 94]:

1. $\prod_{i=1}^{L} m_i \geq D$, где D – верхняя граница диапазона чисел.
2. $\alpha = [\sum_{i=1}^{L} a_i] = 0$, т.е. чтобы выполнялось равенство $\log_2((m_1 - 1)(m_2 - 1) \dots (m_L - 1)) = k_1 + k_2 + \dots + k_L - L$, где $k_i = f_i(m_i)$.
3. В случае выполнения условий 1 и 2 сумму $k_1 + k_2 + \dots + k_L - L$ заносим в числовой массив и запоминаем соответствующие наборы m_i.
4. Находим наименьший элемент массива и выводим соответствующий набор оснований $m_1, m_2, \dots, m_L$. Если таких наборов несколько, то выбираем из них тот, произведение которых наименьшее.

3.3 Выводы по третьей главе

1. Можно видеть, что ОПСС, КТО и Новая КТО работают по-разному для разных наборов модулей. Среди трёхмодульные наборов СОК, несмотря на

некоторую сложность операций по модулю $(2^n + 1)$, набор модулей $\{2^n, 2^n - 1, 2^n + 1\}$ является самым эффективным.

2. Четырёхмодульные набора СОК с модулями одинакового размера для эффективной реализации используют двухуровневую ОПСС, а не КТО, вопреки предположению, что конструкции на основе КТО быстрее, чем конструкции на основе ОПСС.

3. Наборы по модулю с модулями длины битов, варьирующимися от n до $2n$ битов, работают хорошо, если модули выбраны правильно, что позволяет реализовать больший динамический диапазон.

Заключение

Система остаточных классов (СОК) за счёт непозиционного представления чисел в ряде случаев является достаточно эффективной заменой позиционным системам счисления, что достигается за счёт эффективного использования параллелизма и снижения разрядности операндов

В выпускной квалификационной работе были предложены эффективные для аппаратной реализации методы для обнаружения и исправления ошибок в модулярном коде, модифицирован метод синдрома для системы остаточных классов.

Для достижения этой цели в выпускной квалификационной работе решены задачи:

1. исследованы теоретические основы представления и обработки данных в системе остаточных классов, рассмотрены достоинства и недостатки данной системы.
2. предложены эффективные алгоритмы обнаружения и исправления ошибок для аппаратной реализации.
3. проанализированы модификации методов обнаружения локализации и исправления ошибок.

Список использованных источников

1. Claudio E. D. D., Orlandi G., Piazza F. A systolic redundant residue arithmetic error correction circuit / IEEE Transactions on Computers. – 1993. – Т. 42. – №. 4. – С. 427-432.
2. Katti R. S. A new residue arithmetic error correction scheme / IEEE transactions on computers. – 1996. – Т. 45. – №. 1. – С. 13-19.
3. Червяков Н. И. Экспериментальный анализ генерации больших простых чисел в системе остаточных классов / Червяков Н. И., Бабенко М. Г., Коняева Д. С., Кучукова Н. Н., Кучукова Е. А., Гудиева Н. Г. – Качество. Инновации. Образование. – 2017. – №. 9. – С. 42-50.
4. Червяков Н. И. Нейрокомпьютеры в остаточных классах. Кн. 11: Учеб. пособие для вузов / Червяков Н. И., Сахнюк П. А., Шапошников А. В., Макоха А. Н. – М.: Радиотехника. – 2003. – 272 с.
5. Hwang K. Computer arithmetic principles, architecture, and design. – 1979.
6. Jenkins W., Leon B. The use of residue number systems in the design of finite impulse response digital filters / IEEE Transactions on Circuits and Systems. – 1977. – Т. 24. – №. 4. – С. 191-201.
7. Stouraitis T. Analogue-and binary-to-residue conversion schemes //IEE Proceedings-Circuits, Devices and Systems. – 1994. – Т. 141. – №. 2. – С. 135-139.
8. Alia G., Martinelli E. A VLSI algorithm for direct and reverse conversion from weighted binary number system to residue number system / IEEE Transactions on circuits and systems. – 1984. – Т. 31. – №. 12. – С. 1033-1039.
9. Capocelli R. M., Giancarlo R. Efficient VLSI networks for converting an integer from binary system to residue number system and vice versa / IEEE transactions on circuits and systems. – 1988. – Т. 35. – №. 11. – С. 1425-1430.
10. Piestrak S.J. Design of residue generators and multi-operand modulo adders using carry save adders / Proceedings of the. 10th Symposium on Computer Arithmetic, Grenoble – 1991 – С. 100–107.

11. Piestrak S. J. Design of residue generators and multioperand modular adders using carry-save adders / IEEE Transactions on Computers. – 1994. – Т. 43. – №. 1. – С. 68-77.

12. Mohan P. V. A. Efficient design of binary to RNS converters / Journal of Circuits, Systems, and Computers. – 1999. – Т. 9. – №. 03n04. – С. 145-154.

13. Mohan P. V. A. Novel design for binary to RNS converters / Proceedings of IEEE International Symposium on Circuits and Systems-ISCAS'94. – IEEE, 1994. – Т. 2. – С. 357-360.

14. Pettenghi H., Chaves R., Sousa L. Method for designing modulo $2^n \pm k$ Binary-to-RNS converters / Proc. of Int. Conference on Design of Circuits and Integrated Systems (DCIS). – 2013.

15. Premkumar A. B. A formal framework for conversion from binary to residue numbers / IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing. – 2002. – Т. 49. – №. 2. – С. 135-144.

16. Premkumar A. B., Ang E. L., Lai E. M. K. Improved memoryless RNS forward converter based on the periodicity of residues / IEEE Transactions on Circuits and Systems II: Express Briefs. – 2006. – Т. 53. – №. 2. – С. 133-137.

17. Bi G., Jones E. V. Fast conversion between binary and residue numbers / Electronics Letters. – 1988. – Т. 24. – №. 19. – С. 1195-1197.

18. Pourbigharaz F., Yassine H. M. Simple binary to residue transformation with respect to 2^m+1 moduli / IEE Proceedings-Circuits, Devices and Systems. – 1994. – Т. 141. – №. 6. – С. 522-526.

19. Piestrak S. J. Design of multi-residue generators using shared logic / 2011 IEEE International Symposium of Circuits and Systems (ISCAS). – IEEE, 2011. – С. 1435-1438.

20. Skavantzos A., Abdallah M. Implementation issues of the two-level residue number system with pairs of conjugate moduli //IEEE Transactions on Signal Processing. – 1999. – Т. 47. – №. 3. – С. 826-838.

21. Matutino P. M. et al. Multiplierbased binary-to-RNS converter modulo $\{2^n \pm k\}$ / Proc. 26th Conf. DCIS. – 2011. – С. 125-130.

22. Matutino P. M., Chaves R., Sousa L. Binary-to-RNS Conversion Units for moduli $\{2^n \pm 3\}$ / 2011 14th Euromicro Conference on Digital System Design. – IEEE, 2011. – С. 460-467.

23. Szabo N. S., Tanaka R. I. Residue arithmetic and its applications to computer technology. – McGraw-Hill, 1967.

24. Bernardson P. Fast memoryless, over 64 bits, residue-to-binary convertor / IEEE Trans. Circuits Syst. – 1985. – Т. 32. – №. 3. – С. 298-300.

25. Andraros S., Ahmad H. A new efficient memoryless residue to binary converter / IEEE Transactions on Circuits and Systems. – 1988. – Т. 35. – №. 11. – С. 1441-1444.

26. Ibrahim K. M., Saloum S. N. An efficient residue to binary converter design / IEEE transactions on circuits and systems. – 1988. – Т. 35. – №. 9. – С. 1156-1158.

27. Dhurkadas A. Comments on «An efficient residue to binary converter design» by KM Ibrahim and SN Saloum / IEEE transactions on circuits and systems. – 1990. – Т. 37. – №. 6. – С. 849-850.

28. Mohan P. V. A., Poornaiah D. V. Novel RNS to binary converters / 1991., IEEE International Sympoisum on Circuits and Systems. – IEEE, 1991. – С. 1541-1544.

29. Piestrak S. J. A high-speed realization of a residue to binary number system converter / IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing. – 1995. – Т. 42. – №. 10. – С. 661-663.

30. Dhurkadas A. Comments on «A high speed realization of a residue to binary number system converter» / IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing. – 1998. – Т. 45. – №. 3. – С. 446-447.

31. Bhardwaj M., Premkumar A. B., Srikanthan T. Breaking the 2n-bit carry propagation barrier in residue to binary conversion for the $[2^n-1, 2^n, 2^n+1]$ modula set / IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications. – 1998. – Т. 45. – №. 9. – С. 998-1002.

32. Conway R., Nelson J. Fast converter for 3 moduli RNS using new property of CRT / IEEE Transactions on Computers. – 1999. – Т. 48. – №. 8. – С. 852-860.

33. Wang Z., Jullien G. A., Miller W. C. An improved residue-to-binary converter / IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications. – 2000. – Т. 47. – №. 9. – С. 1437-1440.

34. Mohan P. V. A., Premkumar A. B., Bhardwaj M. Comments on «Breaking the 2n-bit carry-propagation barrier in residue to binary conversion for the [2^n-1, 2^n, 2^n+1] moduli set» and author's reply / IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications. – 2001. – Т. 48. – №. 8. – С. 1031.

35. Wang Y. et al. Adder based residue to binary number converters for (2^n-1, 2^n, 2^n+1) / IEEE transactions on signal processing. – 2002. – Т. 50. – №. 7. – С. 1772-1779.

36. Wang W. et al. A study of the residue-to-binary converters for the three-moduli sets / IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications. – 2003. – Т. 50. – №. 2. – С. 235-243.

37. Vinnakota B., Rao V. V. B. Fast conversion techniques for binary-residue number systems / IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications. – 1994. – Т. 41. – №. 12. – С. 927-929.

38. Mohan P. V. A. Evaluation of fast conversion techniques for binary-residue number systems / IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications. – 1998. – Т. 45. – №. 10. – С. 1107-1109.

39. Gallaher D. D., Petry F. E., Srinivasan P. The digit parallel method for fast RNS to weighted number system conversion for specific moduli (2^k-1, 2^k, 2^k+1) / IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing. – 1997. – Т. 44. – №. 1. – С. 53-57.

40. Mohan P. V. A. Comments on «The digit parallel method for fast RNS to weighted number system conversion for specific moduli (2^k-1, 2^k, 2^k+1)» / IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing. – 2000. – Т. 47. – №. 9. – С. 972-974.

41. Ashur A. S., Ibrahim M. K., Aggoun A. Novel RNS structures for the moduli set (2^n−1, 2^n, 2^n+1) and their application to digital filter implementation / Signal processing. – 1995. – Т. 46. – №. 3. – С. 331-343.

42. Chaves R., Sousa L. {2^n+1, 2^{n+k}, 2^n-1}: a new RNS moduli set extension / Euromicro Symposium on Digital System Design, 2004. DSD 2004. – IEEE, 2004. – С. 210-217.

43. Mohan P. V. A. Reverse converters for the moduli sets {2^{2N}-1, 2^N, 2^{2N}+1} and {2^N-3, 2^N+1, 2^N-1, 2^N+3} / 2004 International Conference on Signal Processing and Communications, 2004. SPCOM'04. – IEEE, 2004. – С. 188-192.

44. Mohan P. V. A. Reverse Converters for a New Moduli Set {2^{2n}−1, 2^n, 2^{2n}+1} / Circuits, Systems & Signal Processing. – 2007. – Т. 26. – №. 2. – С. 215-227.

45. Mohan P. V. A. RNS-To-Binary Converter for a New Three-Moduli Set {2^{n+1}-1, 2^n, 2^n-1} / IEEE Transactions on Circuits and Systems II: Express Briefs. – 2007. – Т. 54. – №. 9. – С. 775-779.

46. Mohan P. V. A. New residue to binary converters for the moduli set {2^k, 2^k−1, 2^{k-1}−1} / TENCON 2008-2008 IEEE Region 10 Conference. – IEEE, 2008. – С. 1-6.

47. Hiasat A., Sweidan A. Residue-to-binary decoder for an enhanced moduli set / IEE Proceedings-Computers and Digital Techniques. – 2004. – Т. 151. – №. 2. – С. 127-130.

48. Soderstrand M., Vernia C., Chang J. H. An improved residue number system digital-to-analog converter / IEEE transactions on circuits and systems. – 1983. – Т. 30. – №. 12. – С. 903-907.

49. Vu T.V. Efficient implementations of the Chinese remainder theorem for sign detection and residue decoding / IEEE Transactions on Communications. – 1985. Т. 34. – С. 646–651.

50. Cardarilli G. C. et al. A systolic architecture for high-performance scaled residue to binary conversion / IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications. – 2000. – Т. 47. – №. 10. – С. 1523-1526.

51. Dimauro G. et al. Residue-to-binary conversion by the" quotient function" / IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing. – 2003. – Т. 50. – №. 8. – С. 488-493.

52. Kim J. Y., Park K. H., Lee H. S. Efficient residue-to-binary conversion technique with rounding error compensation / IEEE Transactions on circuits and systems. – 1991. – Т. 38. – №. 3. – С. 315-317.

53. Huang C. H. A fully parallel mixed-radix conversion algorithm for residue number applications / IEEE Transactions on computers. – 1983. – №. 4. – С. 398-402.

54. Wang Y. Residue-to-binary converters based on new Chinese remainder theorems / IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing. – 2000. – Т. 47. – №. 3. – С. 197-205.

55. Mohan P. V. A. Comments on «Residue-to-binary converters based on new Chinese remainder theorems» / IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing. – 2000. – Т. 47. – №. 12. – С. 1541.

56. Yassine H. M., Moore W. R. Improved mixed-radix conversion for residue number system architectures / IEE Proceedings G (Circuits, Devices and Systems). – 1991. – Т. 138. – №. 1. – С. 120-124.

57. Bi S., Gross W. J. The mixed-radix Chinese remainder theorem and its applications to residue comparison / IEEE Transactions on Computers. – 2008. – Т. 57. – №. 12. – С. 1624-1632.

58. Skavantzos A., Wang Y. New efficient RNS-to-weighted decoders for conjugate-pair-moduli residue number systems / Conference Record of the Thirty-Third Asilomar Conference on Signals, Systems, and Computers (Cat. No. CH37020). – IEEE, 1999. – Т. 2. – С. 1345-1350.

59. Wang Y. New Chinese remainder theorems / Proceedings of 32nd Asilomar Conference on Signals, Systems and Computers. – 1998. – С. 165–171.

60. Akushskii I. J., Burcev V. M., Pak I. T. A new positional characteristic of non-positional codes and its application / Coding theory and the optimization of complex systems. – SSR, Alm-Ata'Nauka'Kazah, 1977.

61. Miller D. D. et al. Analysis of the residue class core function of Akushskii, Burcev, and Pak / Residue number system arithmetic: modern applications in digital signal processing. – 1986. – С. 390-401.

62. Gonnella J. The application of core functions to residue number systems / IEEE Transactions on Signal Processing. – 1991. – Т. 39. – №. 1. – С. 69-75.

63. Burgess N. Scaled and unscaled residue number system to binary conversion techniques using the core function / Proceedings 13th IEEE Sympsoium on Computer Arithmetic. – IEEE, 1997. – С. 250-257.

64. Burgess N. Scaling an RNS number using the core function / Proceedings 2003 16th IEEE Symposium on Computer Arithmetic. – IEEE, 2003. – С. 262-269.

65. Abtahi M., Siy P. Core function of an RNS number with no ambiguity / Computers & Mathematics with Applications. – 2005. – Т. 50. – №. 3-4. – С. 459-470.

66. Abtahi M., Siy P. The non-linear characteristic of core function of RNS numbers and its effect on RNS to binary conversion and sign detection algorithms / NAFIPS 2005-2005 Annual Meeting of the North American Fuzzy Information Processing Society. – IEEE, 2005. – С. 731-736.

67. Krishnan R., Ehrenberg J., Ray G. A core function based residue to binary decoder for RNS filter architectures / Proceedings of 33rd Midwest Symposium on Circuits and Systems,. – IEEE, 1990. – С. 837-840.

68. Meehan S. J., O'neil S. D., Vaccaro J. J. An universal input and output RNS converter / IEEE Transactions on circuits and systems. – 1990. – Т. 37. – №. 6. – С. 799-803.

69. Montgomery P. L. Modular multiplication without trial division / Mathematics of computation. – 1985. – Т. 44. – №. 170. – С. 519-521.

70. Yang L. L., Hanzo L. A residue number system based parallel communication scheme using orthogonal signaling. I. System outline / IEEE transactions on vehicular technology. – 2002. – Т. 51. – №. 6. – С. 1534-1546.

71. How H. T. et al. A redundant residue number system coded burst-by-burst adaptive joint-detection based CDMA speech transceiver / IEEE transactions on vehicular technology. – 2006. – Т. 55. – №. 1. – С. 387-396.

72. Etzel M., Jenkins W. Redundant residue number systems for error detection and correction in digital filters / IEEE Transactions on Acoustics, Speech, and Signal Processing. – 1980. – Т. 28. – №. 5. – С. 538-545.

73. Beckmann P. E., Musicus B. R. Fast fault-tolerant digital convolution using a polynomial residue number system / IEEE transactions on Signal Processing. – 1993. – Т. 41. – №. 7. – С. 2300-2313.

74. Keller T., Liew T. H., Hanzo L. Adaptive redundant residue number system coded multicarrier modulation / IEEE Journal on Selected Areas in Communications. – 2000. – Т. 18. – №. 11. – С. 2292-2301.

75. Hanzo L., Liew T. H., Yeap B. L. Turbo coding, turbo equalisation, and space-time coding. – J. Wiley, 2002. – С. 257-316.

76. Haron N. Z., Hamdioui S. Redundant residue number system code for fault-tolerant hybrid memories / ACM Journal on Emerging Technologies in Computing Systems (JETC). – 2011. – Т. 7. – №. 1. – С. 1-19.

77. Liew T. H., Yang L. L., Hanzo L. Systematic redundant residue number system codes: Analytical upper bound and iterative decoding performance over AWGN and Rayleigh channels / IEEE transactions on communications. – 2006. – Т. 54. – №. 6. – С. 1006-1016.

78. Zhang S., Yang L. L., Zhang Y. Redundant residue number system assisted multicarrier direct-sequence code-division dynamic multiple access for cognitive radios / IEEE transactions on vehicular technology. – 2012. – Т. 61. – №. 3. – С. 1234-1250.

79. Avik S., N. Balasubramaniam Performance of systematic RRNS based space-time block codes with probability-aware adaptive demapping / IEEE Trans. Wireless Commun. – 2013. – Т. 12. – №. 5. – С. 2458-2469.

80. Yatskiv V. et al. The use of modified correction code based on residue number system in WSN / 2013 IEEE 7th International Conference on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS). – IEEE, 2013. – Т. 1. – С. 513-516.

81. Yin P., Li L. A new algorithm for single error correction in RRNS / 2013 International Conference on Communications, Circuits and Systems (ICCCAS). – IEEE, 2013. – Т. 2. – С. 178-181.

82. Goh V. T., Tinauli M., Siddiqi M. U. A novel error correction scheme based on the Chinese remainder theorem / The Ninth International Conference onCommunications Systems, 2004. ICCS 2004. – IEEE, 2004. – С. 461-465.

83. Tang Y. et al. A new single-error correction scheme based on self-diagnosis residue number arithmetic / 2010 Conference on Design and Architectures for Signal and Image Processing (DASIP). – IEEE, 2010. – С. 27-33.

84. Tay T. F., Chang C. H. A new algorithm for single residue digit error correction in Redundant Residue Number System / 2014 IEEE International Symposium on Circuits and Systems (ISCAS). – IEEE, 2014. – С. 1748-1751.

85. Yau S. S. S., Liu Y. C. Error correction in redundant residue number systems / IEEE Transactions on Computers. – 1973. – Т. 100. – №. 1. – С. 5-11.

86. Sun J. D., Krishna H. A coding theory approach to error control in redundant residue number systems. Part II. multiple error detection and correction / IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing. – 1992. – Т. 39. – №. 1. – С. 18-34.

87. Mandelbaum D. On a class of arithmetic codes and a decoding algorithm / IEEE Transactions on Information Theory. – 1976. – Т. 22. – №. 1. – С. 85-88.

88. Goldreich O., Ron D., Sudan M. Chinese remaindering with errors / IEEE Transactions on Information Theory. – 2000. – Т. 46. – №. 4. – С. 1330-1338.

89. Goh V. T., Siddiqi M. U. Multiple error detection and correction based on redundant residue number systems / IEEE Transactions on Communications. – 2008. – Т. 56. – №. 3. – С. 325-330.

90. Tay T. F., Chang C. H. A non-iterative multiple residue digit error detection and correction algorithm in RRNS //IEEE transactions on computers. – 2015. – Т. 65. – №. 2. – С. 396-408.

91. Barsi F., Maestrini P. Error correcting properties of redundant residue number systems / IEEE Transactions on Computers. – 1973. – Т. 100. – №. 3. – С. 307-315.

92. Jenkins W. K., Altman E. J. Self-checking properties of residue number error checkers based on mixed radix conversion / IEEE Trans. Circuits Syst. – 1988 – С. 159-167.

93. Бабенко М. Г. Анализ подходов к построению арифметических сопроцессоров для задач эллиптической криптографии / Бабенко М. Г., Вершкова Н. Н., Кучеров Н. Н. – Наука. Инновации. Технологии. – 2013. – №. 1.

94. Червяков Н. И. Модулярные параллельные вычислительные структуры нейропроцессорных систем / Червяков Н. И., Сахнюк П. А., Шапошников А. В., Ряднов С. А.; под ред. Червякова Н. И. – М.: ФИЗМАЛИТ, 2003. – 288 с