Федеральное государственное автономное образовательное учреждение высшего образования
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

На правах рукописи

Мыцко Евгений Алексеевич

АЛГОРИТМЫ И АППАРАТНАЯ РЕАЛИЗАЦИЯ НА ПЛИС УСТРОЙСТВ ОБНАРУЖЕНИЯ И ИСПРАВЛЕНИЯ ПАКЕТНЫХ ИЛИ НЕЗАВИСИМЫХ ОШИБОК ДЛЯ СООБЩЕНИЙ КОРОТКОЙ ДЛИНЫ

05.13.05 – «Элементы и устройства вычислительной техники и систем управления»

ДИССЕРТАЦИЯ
На соискание ученой степени
кандидата технических наук

Научный руководитель
д. т. н., доцент
Ким Валерий Львович

Томск – 2019

ОГЛАВЛЕНИЕ

ПЕРЕЧЕНЬ ИСПОЛЬЗУЕМЫХ СОКРАЩЕНИЙ

ПЛИС – программируемая логическая интегральная схема.

МК – микроконтроллер.

CRC – cyclic redundancy code.

КС – кодовое слово.

ПК – помехоустойчивый код.

ЦПК – циклический помехоустойчивый код.

коды БЧХ – коды Боуза-Чоудхури-Хоквингема.

коды РС – коды Рида-Соломона.

БМА – алгоритм Берлекэмпа-Месси.

ПО – программное обеспечение.

ЭВМ – электронная вычислительная машина.

САПР – система автоматизированного проектирования.

ЛМС – локальный модуль синхронизации.

ЦБС – центральный блок синхронизации.

ВУ – вычислительные устройства.

код БЧХ (15, 7, 5) – код Боуза-Чоудхури-Хоквингема с длиной кодового слова 15 бит, длиной информационного блока 7 бит и расстоянием Хэмминга равным 5.

код РС (7, 3) – код Рида-Соломона с длиной кодового слова 7 символов и длиной информационного блока 3 символа.

ВВЕДЕНИЕ

Актуальность работы

В настоящее время в эпоху цифровой передачи данных актуальным является обеспечение целостности передаваемых данных от одного вычислительного устройства (ВУ) к другому. Каналы передачи данных могут быть ненадежными в виду воздействия различных шумов и наводок. В свою очередь, в вычислительных устройствах при передаче информации от одного блока к другому любая одиночная ошибка может существенно нарушить процесс вычислений или привести к снижению производительности ВУ. Искажения в данных или командах, возникающие в процессе передачи между элементами вычислительной техники, также обуславливаются старением элементов, ухудшением качества электрических соединений и нестабильностью питания. Из-за отказа отдельных интегральных схем ошибки возникают и при хранении данных, поэтому большая часть ошибок, как правило, приходится на память. Таким образом, при проектировании устройств передачи (приёма), а также хранения информации требуется реализация способов и алгоритмов обнаружения и исправления ошибок, возникающих в результате воздействия помех или сбоев оборудования.

Одним из известных подходов к обнаружению и исправлению ошибок является применение помехоустойчивого кодирования с избыточностью кода [1–7].

Первые классы кодов, исправляющие одиночные ошибки, были введены Р. Хэммингом [1] в 1956 году. В 60-х годах XX века Р. Боуз [2], Д. Рой-Чоудхури [2], А. Хоквингем [3] (коды Боуза-Чоудхури Хоквингема, БЧХ), И. Рид [4] и Г. Соломон [5] (коды Рида-Соломона, РС) предложили более сильные коды, исправляющие многократные ошибки. Такие коды используются в системах управления, связи и устройствах вычислительной техники по сей день. Из российских ученых большой вклад в исследование помехоустойчивых кодов внесли Харкевич А.А., Колесников В.Д., Золотарёв В.В., Зяблов В.В. В

процессе развития вычислительной техники предложенные алгоритмы исправления ошибок нашли применение в различных устройствах передачи и хранения данных.

После открытия мощных кодов, исправляющих многократные ошибки, актуальной стала задача разработки эффективных способов и алгоритмов их построения и декодирования. Первые алгоритмы для построения и декодирования кодов БЧХ и Рида-Соломона с практическим применением были предложены У. Питерсоном [6] и Э. Берлекэмпом [7]. В современных научных трудах можно встретить исследования способов декодирования различных помехоустойчивых кодов таких российских ученых как Золотарев В.В. [8–11] (многопороговые декодеры), Гладких А.А. [12–15] (декодирование блоковых и избыточных кодов), Егоров С.И. [16–19] (декодеры низкоплотностных кодов и кодов Рида-Соломона), Башкиров А.В. [20–22] (реализация многофункциональных декодеров на программируемых логических интегральных схемах (ПЛИС)).

В 1961 году У. Питерсоном был предложен отдельный вид помехоустойчивых кодов, позволяющих обнаруживать ошибки без прямого исправления – циклический избыточный код CRC (Cyclic redundancy code) [23]. Основной целью кодов CRC было эффективное обнаружение ошибок с применением наименьшей и фиксированной избыточности. Исправление ошибок на основе таких кодов осуществляется за счёт повторной передачи пакетов данных.

На данный момент CRC-коды длиной от 4 до 64 бит (CRC4, CRC8, CRC16, CRC32, CRC64) применяются в протоколах передачи данных ETHERNET, ZigBee, ModBus, DLMS/SLIP, а также в архиваторах WinRaR, WinZIP. В сетевых платах ЭВМ проверка целостности данных с применением CRC-кодов аппаратно реализуются в составе модуля FCS (Frame check sequence) [24, 25]. Помехоустойчивые коды, исправляющие ошибки, широко применяются в стандартах цифрового телевизионного вещания DVB-T2 [26],

системах спутниковой [27, 28], сотовой связи [29], системах автоматизации производства и телекоммуникациях [30], системах дистанционного зондирования Земли [31]. Применение помехоустойчивых кодов, обнаруживающих и исправляющих ошибки, является одним из способов повышения надежности цифровых и запоминающих устройств ЭВМ [32–37]. Современные модули памяти SDRAM DDR реализуются с аппаратной поддержкой алгоритмов исправления ошибок на основе кодов Хэмминга и дополнительных модулей хранения кодов ECC (Error-correcting code) [38–40].

Существующие алгоритмы обнаружения ошибок, применяемые в устройствах с контролем целостности данных и основанные на применении циклических избыточных кодов, позволяют эффективно обнаруживать ошибки в больших пакетах данных, однако либо обладают низким быстродействием (классический побитовый алгоритм), либо при модификации требуется достаточно большой объем памяти (табличный алгоритм). Таким образом, актуальной является задача разработки алгоритма вычисления циклических избыточных кодов, который будет компромиссным вариантом по быстродействию и аппаратным затратам при реализации на микроконтроллерах и ПЛИС.

Циклические помехоустойчивые коды (ЦПК) БЧХ, исправляющие независимые ошибки, обычно записываются в формате (n, m, d), где n – длина кодового слова, m – длина информационного блока, d – расстояние Хэмминга [1]. Например, ЦПК БЧХ (15, 7, 5) имеет длину кодового слова 15 бит, длину информационного блока 7 бит и расстояние Хэмминга равное 5. Длины кодовых слов таких кодов ограничены выражением $n = 2^h - 1$, где h – натуральное число. Данный факт не позволяет построить помехоустойчивый код для любых n и m без применения операции укорачивания, что приводит к снижению эффективности кода, а также к низкой эффективности использования аппаратных ресурсов при разработке устройств исправления ошибок. Коды Рида-Соломона, являющиеся частным случаем кодов БЧХ,

имеют меньшую избыточность, но при этом ориентированы на исправление только пакетных ошибок. Для реализации существующих методов и алгоритмов декодирования помехоустойчивых кодов БЧХ и Рида-Соломона требуется знание арифметики полей Галуа и решение ключевых уравнений, что увеличивает сложность разработки устройств исправления ошибок. Таким образом, имеет место проблема построения помехоустойчивых кодов, аналогичных кодам БЧХ и Рида-Соломона, но более эффективных по избыточности и для любых длин информационного сообщения без применения операции укорачивания.

Во многих системах с защитой передаваемых данных от ошибок модули обнаружения и исправления ошибок являются служебными (вспомогательными), а основные ресурсы памяти и вычислительные мощности устройств используются для решения задач согласно прямому назначению системы. В связи с этим, сформулируем следующие требования к способам, алгоритмам и устройствам, обеспечивающим целостность информации:

- высокая эффективность – наилучшее отношение полезной информации к избыточной;
- простота реализации – низкая сложность алгоритмов, а также наименьшая область знаний для программных и аппаратных реализаций;
- быстродействие – минимальное время (количество тактов устройства) для обнаружения и исправления ошибок;
- минимальные аппаратные затраты – наименьшее количество логических элементов модуля исправления ошибок и небольшой требуемый объем памяти запоминающего устройства для реализации в системах с дефицитом ресурсов.

Таким образом, актуальными задачами являются разработка программ и аппаратная реализации устройств обнаружения и исправления ошибок, удовлетворяющих сформулированным ранее требованиям с применением

новых и модификацией существующих алгоритмов. С учетом описанных выше проблем и задач сформулируем цель настоящей работы.

Целью работы является разработка эффективных алгоритмов, программ и устройств обнаружения и исправления пакетных или независимых ошибок при передаче и хранении данных с информационным сообщением короткой длины.

Для выполнения поставленной цели необходимо выполнить следующие задачи:

1. Провести обзор и анализ существующих способов и алгоритмов обнаружения и исправления ошибок, применяемых в устройствах передачи и хранения данных.
2. Разработать алгоритм быстрого вычисления циклических избыточных кодов CRC для реализации на микроконтроллерах и ПЛИС. Реализовать разработанный алгоритм на микроконтроллере и ПЛИС для сравнения с существующими алгоритмами.
3. Разработать алгоритм поиска образующих полиномов, позволяющий находить полиномы, применяемые для построения более эффективных циклических помехоустойчивых кодов, чем коды БЧХ.
4. Разработать устройства исправления независимых ошибок на ПЛИС с использованием известных циклических помехоустойчивых кодов короткой длины и предложенного помехоустойчивого кода, построенного на основе результатов поиска образующего полинома, для сравнения быстродействия и аппаратных затрат.
5. Осуществить поиск образующих полиномов для построения циклических кодов, исправляющих пакетные ошибки. Разработать алгоритм и устройства исправления пакетных ошибок на ПЛИС на основе циклического помехоустойчивого кода, исправляющего пакетные ошибки.

Объектом исследования являются программы и устройства на ПЛИС для обнаружения и исправления пакетных или независимых ошибок.

Предметом исследования являются алгоритмы обнаружения и исправления пакетных или независимых ошибок для реализации на ПЛИС.

Методы исследований. Для достижения поставленной цели в работе применены методы компьютерного моделирования, разработки программ на языках программирования низкого и высокого уровней; основы цифровой схемотехники и разработки цифровых устройств с применением систем автоматизированного проектирования.

Достоверность и обоснованность предложенных алгоритмов, программ и аппаратных реализаций подтверждается результатами экспериментальных исследований, компьютерным моделированием в САПР Quartus II и публикациями основных результатов работы в ведущих российских изданиях и трудах зарубежных конференций.

Научную новизну полученных в работе результатов определяют:

1. Матричный алгоритм вычисления контрольной суммы CRC, отличающийся от известных лучшим быстродействием при реализации на ПЛИС и меньшим требуемым объёмом памяти при программной реализации.
2. Алгоритм поиска образующих полиномов, адаптированный для параллельных вычислений и отличающийся от известных тем, что позволяет получать полиномы более короткой длины для построения циклических помехоустойчивых кодов, более эффективных с точки зрения отношения полезной информации к избыточной, чем коды БЧХ.
3. Предложенные быстродействующие декодирующие устройства на основе циклического помехоустойчивого кода (17, 9, 5) с меньшими аппаратными затратами, чем устройства на основе укороченного кода БЧХ (19, 9, 5).
4. Предложенная модификация циклического алгоритма декодирования с применением образующих полиномов, полученных с помощью программы поиска, позволяющая исправлять пакетные ошибки для циклических помехоустойчивых кодов без ограничения длины кодового слова.

Практическая значимость работы

Практически значимыми являются программы вычисления контрольной суммы CRC с применением предложенного алгоритма, аппаратные реализации устройств вычисления CRC8 и CRC32 на ПЛИС, программы вычисления CRC8 на языке Assembler для микроконтроллера ATtiny44, алгоритмы и программы поиска образующих полиномов на языке C++ с применением технологии OpenMP. Разработанные устройства исправления независимых и пакетных ошибок на ПЛИС с применением табличного и циклического алгоритмов декодирования обеспечивают целостность информации в устройствах вычислительной техники. По результатам исследований получено 4 свидетельства о регистрации программы для ЭВМ.

Работа выполнена в рамках ФЦП «Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2014—2020 годы», соглашение № 05.578.21.0272 от 20.12.2018 г. (уникальный идентификатор проекта RFMEFI57818X0272) по теме: «Разработка технических решений и аппаратно-программного комплекса управления цифровыми электрическими подстанциями для построения интеллектуальной энергосистемы» и хозяйственного договора № 4-673/16У от 26.09.2016 «Разработка средств технического обеспечения устройств сбора данных, контроля и защиты электрофизической установки токамак» (заказчик ООО «ТомИУС-проект», г. Томск).

Реализация и внедрение результатов работы

Полученные алгоритмы и аппаратные модули на ПЛИС применены в системе управления электрофизической установки Токамак КТМ (Национальный ядерный центр, Республика Казахстан, г. Курчатов), в том числе разработанное устройство на ПЛИС для исправления пакетных ошибок длиной до 3 бит при длине сообщения 8 бит с модулем вычисления CRC8 в системе передачи команд от центрального блока подсистемы синхронизации к локальным модулям синхронизации.

Отдельные результаты диссертационного исследования использованы в учебном процессе НИ ТПУ в дисциплинах «Микропроцессоры и микроконтроллеры», «Программирование на языках описания аппаратуры».

Основные положения, выносимые на защиту:

1. Матричный алгоритм вычисления циклических избыточных кодов CRC позволяет при программной реализации задействовать до 32 раз меньший объем памяти для хранения предвычисленных значений, а при аппаратной реализации на ПЛИС увеличить быстродействие устройств вычисления CRC до 8,5 раз по сравнению с табличным алгоритмом.
2. Алгоритм поиска образующих полиномов, адаптированный для параллельных вычислений, позволяет находить полиномы для построения циклических помехоустойчивых кодов, имеющих избыточность до 25 % меньше, чем у циклических кодов БЧХ с сообщением длиной от 2 до 32 бит и количеством исправляемых ошибок от 2 до 4.
3. Разработанное устройство исправления независимых ошибок на ПЛИС с применением циклического помехоустойчивого кода (17, 9, 5) на основе циклического алгоритма декодирования обладает в 2 раза лучшим быстродействием и позволяет повысить эффективность использования логических элементов в 2,6 раза по сравнению с устройством на основе укороченного кода БЧХ (19, 9, 5) с аналогичной длиной информационного блока.
4. Разработанное устройство исправления пакетных ошибок на ПЛИС с применением модифицированного циклического алгоритма декодирования и образующих полиномов, полученных с помощью программы поиска, позволяет исправлять пакетные ошибки для циклических помехоустойчивых кодов без ограничения длины кодового слова.

Соответствие результатов паспорту научной специальности

В работе приведены результаты создания принципиально новых схем и устройств с применением языка описания аппаратуры для решения задачи

контроля целостности информации, передаваемой между элементами вычислительной техники, что соответствует п.1 паспорта научной специальности 05.13.05 Элементы и устройства вычислительной техники и систем управления. В соответствие с п.4. паспорта научной специальности разработаны алгоритмы и программы контроля целостности данных в устройствах вычислительной техники.

Апробация работы. Основные результаты работы обсуждались на следующих конференциях и семинарах:

Международная научно-практическая конференция «Молодежь и современные информационные технологии» (г. Томск, 2012 – 2017 гг.), Международная научно-практическая конференция студентов, аспирантов и молодых учёных «Современные техника и технологии» (г. Томск, 2012, 2014 гг.), 9-й Международный Форум по Стратегическим Технологиям 2014, IFOST-2014 (г. Дакка, Бангладеш, 2014 г.), XII Международная IEEE Сибирская конференция по управлению и связи, SIBCON-2016 (г. Москва, 2016 г.), III Международная научная конференция «Информационные технологии в науке, управлении, социальной сфере и медицине» (г. Томск, 2016 г.), 55-ой Международная научная студенческая конференция МНСК-2017, (г. Новосибирск, 2017 г), Международная научно-техническая конференция студентов, аспирантов и молодых ученых, посвященная 55-летию ТУСУРа, «Научная сессия ТУСУР-2017» (г. Томск, 2017 г.), III Всероссийский молодежный научный форум «Наука будущего – наука молодых», (г. Нижний новгород, 2017 г), Всероссийская научная конференция молодых ученых «Наука. Технологии. Инновации», (г. Новосибирск, 2017 г), VIII Международная научно-практическая конференция «Высокопроизводительные вычислительные системы и технологии в научных исследованиях, автоматизации управления и производства», ВВСТ– 2018 (г. Барнаул, 2018 г.).

Результаты диссертационной работы были отмечены медалью Российской академии наук в области информатики, вычислительной техники и

автоматизации за научно-исследовательскую работу «Исследования матричного алгоритма вычисления контрольной суммы CRC и его аппаратная реализация» по итогам конкурса 2014 г (постановление Президиума РАН № 24 от 17.02.2015 г.).

Публикации

Результаты диссертационного исследования опубликованы в 6 статьях рецензируемых изданий, рекомендованных ВАК РФ. Четыре публикации индексированы в Международной базе данных SCOPUS (Conference paper) и 6 публикаций – в Web of Science (Conference paper). Получено 4 свидетельства о регистрации программы для ЭВМ.

Личный вклад

Основные научные результаты, выносимые на защиту, получены автором самостоятельно. В работах, опубликованных в соавторстве, личный вклад состоит в следующем: в публикациях [58 – 61, 64, 65, 70, 71, 73] автором реализован на ПЛИС, микроконтроллере и под ОС Linux матричный алгоритм вычисления контрольной суммы CRC. В работах [86–91, 95, 96] автором предложены и реализованы алгоритмы поиска образующих полиномов для построения циклических помехоустойчивых кодов, исправляющих независимые и пакетные ошибки. Поставлены компьютерные эксперименты по поиску образующих полиномов с применением технологий параллельных вычислений. В публикациях [103–107] автором проведены исследования аппаратных реализаций на ПЛИС циклического алгоритма декодирования на примере кодов БЧХ короткой длины (до 15 бит). В работах [97, 117, 118] автором предложены и реализованы устройства исправления ошибок на основе циклического помехоустойчивого кода (17, 9, 5), обладающего лучшими характеристиками по сравнению с кодом БЧХ (15, 7, 5) и укороченным кодом БЧХ (19, 9, 5). Предложенный автором циклический алгоритм декодирования и устройство на ПЛИС для исправления пакетных ошибок представлены в работах [69, 99 – 101].

Структура и объём работы. Диссертация состоит из введения, четырех глав, заключения, списка литературы из 121 наименования и пяти приложений. Объём диссертации составляет 217 страниц, включая 102 рисунка и 40 таблиц.

Первая глава посвящена анализу современных способов и алгоритмов обнаружения и исправления пакетных и независимых ошибок при передаче и хранении данных.

Рассмотрены известные алгоритмы вычисления циклических избыточных кодов CRC. Анализ показал, что существуют два основных алгоритма вычисления CRC: классический с применением последовательного сдвига бит данных через регистр и операции сложения по модулю 2 (Исключающее ИЛИ), а также табличный, ускоряющий процесс вычисления CRC за счет применения таблицы предвычисленных значений. Недостатком первого алгоритма является низкая скорость вычисления из-за побитового сдвига. Недостатком табличного алгоритма является то, что требуемый объем памяти для хранения таблицы предвычисленных значений (256 байт для CRC8, 1 Кб для CRC32) превышает объем памяти программ для исходного кода вычисления CRC.

Проанализированы существующие способы и алгоритмы исправления пакетных и независимых ошибок при передаче данных. Обозначена проблема выбора образующего полинома, позволяющего строить эффективные помехоустойчивые коды, исправляющие как независимые, так и пакетные ошибки, а также алгоритма декодирования данных кодов, требующего минимальных аппаратных и вычислительных затрат для его реализации.

Во **второй** главе предложен матричный алгоритм вычисления контрольной суммы CRC с модификациями, позволяющими увеличить размер блока данных, обрабатываемого за итерацию. Поставлен компьютерный эксперимент по вычислению контрольной суммы CRC32 для различных данных объёмом от 10 до 1010 Мб. Установлено, что реализация матричного алгоритма со сдвигом и буфером, равным 4 байта, является более

быстродействующей, чем реализация табличного алгоритма. Установлено, что для данной матричной реализации требуется в 8 раз меньший объем памяти.

Поставлен эксперимент на микроконтроллере Attiny44 по вычислению контрольной суммы CRC8 для системы измерения температуры с применением датчика DS18B20. По результатам эксперимента выработаны рекомендации к применению алгоритмов расчёта CRC для систем с дефицитом ресурсов – небольшим объёмом памяти программ и данных и низкой производительностью.

Поставлен эксперимент по вычислению CRC8 и CRC32 на ПЛИС Cyclone III от Altera. Результаты экспериментов показали, что увеличение размера блока данных, обрабатываемого за итерацию матричным алгоритмом, позволяет увеличить быстродействие устройства вычисления CRC, однако при этом увеличивается количество логических элементов ПЛИС. Установлено, что в отличие от микроконтроллерной реализации, где вычисления производятся строго последовательно по тактовому сигналу, архитектура ПЛИС позволяет параллельно обрабатывать несколько байт данных, что хорошо подходит для реализации матричного алгоритма.

В **третьей** главе предложен алгоритм поиска образующих полиномов, отличающийся от известных тем, что позволяет находить полиномы для построения циклических помехоустойчивых кодов более эффективных по скорости кода, чем коды БЧХ.

Поставлен компьютерный эксперимент по поиску образующих полиномов с применением технологий параллельных вычислений для построения кодов, исправляющих независимые и пакетные ошибки. Разработанный алгоритм поиска позволяет находить полиномы для построения кодов, длина которых на 1–5 бит короче, чем коды БЧХ при длине информационного блока до 32 бит и кратности исправляемых ошибок до 4..

Поставлен эксперимент по исправлению независимых и пакетных ошибок на микроконтроллере Atmega8515 циклическим и табличным алгоритмами

декодирования. По результатам эксперимента сформулированы рекомендации по применению исследуемых алгоритмов для исправления ошибок в микропроцессорных системах сбора данных и управления.

Проведено исследование аппаратных реализаций на ПЛИС декодеров помехоустойчивого кода БЧХ (15, 7, 5) с различными алгоритмами декодирования. Циклический алгоритм декодирования при реализации на ПЛИС обладает лучшим быстродействием по сравнению с известным алгоритмом Берлекэмпа-Месси, за счёт простых операций проверки веса остатка и сложений по модулю 2. При параллельной реализации на ПЛИС это требует наименьшего количества тактов.

Предложены устройства исправления независимых ошибок на ПЛИС на основе циклического помехоустойчивый кода (17, 9, 5), обладающего большей скоростью и позволяющего кодировать более длинный информационный блок при сохранении избыточности, чем код БЧХ (15, 7, 5). Проведено сравнение характеристик разработанных устройств исправление ошибок на ПЛИС с применением предложенного помехоустойчивого кода (17, 9, 5) и укороченного кода БЧХ (19, 9, 5). Сделаны выводы о том, что декодирующие устройства с применением помехоустойчивого кода (17, 9, 5) являются более быстродействующими и менее аппаратно-затратными, чем устройства с применением укороченного кода БЧХ (19, 9, 5).

Предложена модификация циклического алгоритма декодирования, позволяющая исправлять пакетные ошибки с применением образующих полиномов, найденных с помощью алгоритма поиска для пакетных ошибок. Приведено описание и исследование разработанных декодеров циклического помехоустойчивого кода (15, 8, 3) на ПЛИС, исправляющих пакетные ошибки длиной до 3 бит с применением табличного и циклического алгоритмов декодирования. Установлено, что декодеры обладают лучшим быстродействием и меньшими аппаратными затратами, чем декодеры БЧХ кода (15, 7, 5) с аналогичной длиной кодового слова.

На примере устройства декодирования аналога кода РС (7, 3) с длиной сообщения 9 бит и длиной кодового слова 21 бит продемонстрирована возможность циклического алгоритма исправлять пакетные ошибки длиной до 6 бит.

В **четвёртой** главе приведено описание практического применения устройств обнаружения и исправления пакетных ошибок с помехоустойчивым кодом (15, 8, 3) и циклическим избыточным кодом CRC8 для обеспечения целостности передаваемых команд подсистемы синхронизации системы управления электрофизической установки Токмак КТМ. Представлены описание и результаты эксперимента по исправлению пакетных и независимых ошибок на специальном стенде с применением макетов SDK-6.1 и осциллографа DS01012A.

В **заключении** приведены выводы и кратко описаны основные результаты диссертационной работы.

В **приложении А** приведены блок-схемы алгоритмов вычисления CRC8 для программной реализации на микроконтроллере, описанной в главе 2.

В **приложении Б** приведены таблицы шаблонов ошибок для помехоустойчивого кода БЧХ (15, 7, 5) и предложенного помехоустойчивого кода (17, 9, 5), описанных в главе 3.

В **приложении В** приведены таблицы, показывающие количество тактов, необходимое для исправления независимых ошибок для циклических помехоустойчивых кодов (17, 9, 5) и укороченного кода БЧХ (19, 9, 5) при классической реализации циклического алгоритма декодирования.

В **приложении Г** представлены копии свидетельств о регистрации программ для ЭВМ.

В **приложении Д** представлены копии актов о внедрении результатов диссертационной работы.

ГЛАВА 1. АНАЛИЗ СПОСОБОВ И АЛГОРИТМОВ ОБНАРУЖЕНИЯ И ИСПРАВЛЕНИЯ ПАКЕТНЫХ ИЛИ НЕЗАВИСИМЫХ ОШИБОК

Способы и алгоритмы обнаружения и исправления ошибок являются важными элементами для обеспечения надежности цифровых устройств [33] вычислительной техники. При этом наиболее чувствительными к искажению информации являются запоминающие устройства, поэтому в вычислительной технике наибольшее внимание уделяется обнаружению и исправлению ошибок в различных микросхемах памяти [32–37].

Существует множество методов обнаружения или исправления ошибок разного типа в различных системах передачи или хранения данных [1–15]. В основе таких методов лежит помехоустойчивое кодирование, которое подразумевает изменение или дополнение исходных данных путём внесения дополнительной (избыточной) информации по определенному алгоритму. Теория помехоустойчивого кодирования базируется на результатах исследований, проведенных Клодом Шенноном [41, 42], который сформулировал теорему для дискретного канала с шумом. В данной главе рассмотрены классы существующих помехоустойчивых кодов, а также способы реализации операций декодирования этих кодов.

1.1. Классификация помехоустойчивых кодов

Помехоустойчивые коды можно разбить на две большие группы: блоковые и свёрточные [43]. В блоковых кодах кодирование и декодирование осуществляется в пределах кодовой комбинации, и каждый блок информационных символов обрабатывается отдельно. В свёрточных кодах обработка символов производится непрерывно, без разделения на блоки, и кодовая последовательность зависит от предыдущих, уже прошедших кодирование символов. На рисунке 1.1 приведена классификация помехоустойчивых кодов.

Блоковые коды бывают разделимыми и неразделимыми. При кодировании разделимыми кодами в выходных последовательностях символов

можно разграничить информационные символы и проверочные. В неразделимых кодах такое разграничение невозможно. В свою очередь, разделимые коды делятся на систематические и несистематические.

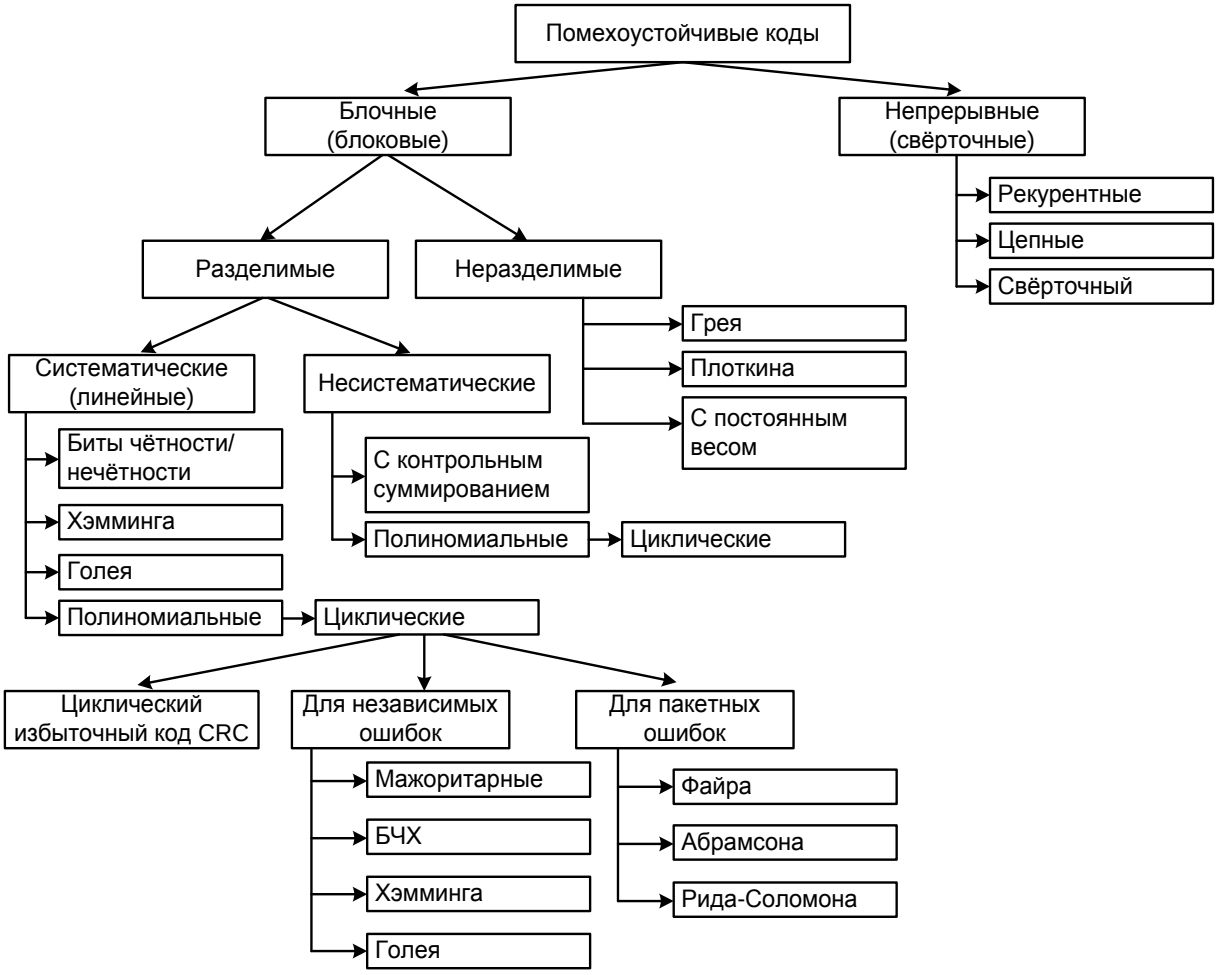


Рисунок 1.1 – Типы помехоустойчивых кодов

Систематические коды также называют линейными, так как в них контрольные (проверочные) разряды представляют собой различные линейные комбинации информационных разрядов. Линейные коды являются наиболее применяемыми на практике, поэтому существует большое количество видов кодов данного типа.

1.2. Коды для обнаружения ошибок

В сфере информационных технологий большую роль играют различные протоколы передачи и алгоритмы архивации данных. Одной из важных задач является контроль целостности данных при передаче по различным каналам

связи. Существует несколько способов обнаружения ошибок в процессе передачи (хранения) данных, отличающиеся количеством детектируемых ошибок, быстродействием, требуемыми программными и аппаратными ресурсами для реализации.

1.2.1. Контрольная сумма

Самый простой способ решения задачи обнаружение ошибок – вычисление некоторого опознавателя сообщения в виде контрольной суммы. Простейшим опознавателем достоверности переданного сообщения является бит контроля четности. Данный бит является суммой по модулю 2 всех бит сообщения, добавляется в конец сообщения и указывает на четность или нечетность единичных разрядов сообщения. Данный способ позволяет обнаружить ошибку нечётной кратности (1,3,5, и. т. д), требует минимальных аппаратных затрат, но снижает производительность системы передачи данных из-за побитового сложения информационных бит сообщения.

Существуют модификации метода контроля чётности, в которых вычисляется сумма по модулю 2 всех байтов пакета [44]. Однократные ошибки в таком методе обнаруживаются с вероятностью 100 %, двукратные – с вероятностью 88% [44]. Однако пакетные ошибки (искажения нескольких смежных битов) данным методом выявляются довольно плохо. Также существует метод, основанный на арифметической сумме всех байтов (или слов) пакета [45, 46]. В данном методе при ее вычислении отбрасываются старшие разряды для сохранения заданной разрядности контрольной суммы (обычно 8 или 16). Однократные ошибки обнаруживаются с вероятностью 100 %. Вероятность не обнаружения двукратных ошибок в наихудшем случае составляет приблизительно 3 % [44]. Это наблюдается, когда в каждом из 8 разрядов всех байт пакета или в каждом из 16 разрядов всех слов пакета соотношение нулей и единиц составляет 50/50. При этом двукратные ошибки не выявляются, когда в одном разряде вследствие ошибки один из битов из «0»

переходит в «1», а другой бит в этом же разряде из «1» переходит в «0», что не изменяет общей суммы [45].

1.2.2. Циклический избыточный код CRC

Более совершенным является способ обнаружения, основанный на вычислении некоторого значения фиксированной длины (4,8,16,32,64 бит), являющейся функцией сообщения. Передающее устройство вычисляет данное значение и добавляет его к передаваемому сообщению [46]. Структура последнего, как правило, имеет следующий вид:

<исходное неизмененное сообщение> <контрольная сумма>.

Для проверки целостности данных на приёмной стороне, используя тот же самый алгоритм, применяют одну из следующих методик:

1) осуществляется расчет контрольной суммы CRC принятого сообщения и сравнение результата с полученным значением CRC;

2) осуществляется расчет контрольной суммы CRC для всего сообщения вместе CRC и сравнение результата с нулевым значением.

В случаях несоответствия результатов полученному значению CRC (подход 1) либо нулевому значению (подход 2) осуществляется повторная пересылка данных.

CRC является практическим приложением теории помехоустойчивого кодирования и применяется в различных протоколах передачи данных (ETHERNET, ZigBee, USB, ModBus, DLMS/SLIP), стандартах MPEG-2, PNG и алгоритмах архивации данных (WinRaR, WinZIP). Аппаратно реализуется на микросхемах, входящих в состав сетевых плат [24, 25]. Под избыточностью подразумевается наличие дополнительного (контрольного) блока данных в передаваемом сообщении. Данный код, построение которого основано на математических свойствах циклического кода [47], позволяет гарантированно обнаруживать ошибки кратности $d - 1$, где d – минимальное кодовое расстояние. Ошибки кратностью d и больше CRC обнаруживает с достаточно высокой вероятностью $P_{об} = 1 - \frac{1}{2^r}$, где r – разрядность CRC в битах. Методика

вычисления CRC базируется на свойствах деления многочленов по модулю два. Значение CRC является остатком от деления по модулю два многочлена, соответствующего входным данным, на образующий многочлен [48], который выбирается в зависимости от протокола или стандарта передачи (архивации) данных.

Существуют различные виды CRC, отличающиеся разрядностью контрольного блока, такие как CRC4, CRC8, CRC16, CRC32, CRC64. При увеличении разрядности CRC уменьшаются вероятности «коллизий» (случаев, при которых для разных сообщений можно получить одинаковое значение CRC) и увеличивается кратность обнаруживаемых ошибок. Так, контрольная сумма CRC4 применяется в стандарте цифровых сетей ITU-T G.704, CRC8 – в стандарте телевещания DVB-S2, стандарте передачи данных Bluetooth, в цифровых датчиках температуры Dallas Semiconductor (шина 1-Wire), CRC16 – в промышленных протоколах ModBus и DLMS/SLIP для сетей АСУТП, CRC32 – в протоколах передачи данных Ethernet, ZigBee, CRC64 – в стандартах ECMA-182. Виды наиболее применяемых на практике контрольных сумм CRC приведены в таблице 1.1.

Таблица 1.1 – Контрольные суммы CRC с образующими полиномами

Вид CRC	Применение	Образующий полином (hex)
CRC4	ITU-T G.704	0x3
CRC8	DVB-S2	0xD5
CRC8-Bluetooth	Bluetooth	0xA7
CRC8-CCITT	ITU-T I.432.1, (02/99), ATM HEC, ISDN HEC	0x07
CRC8-Dallas/Maxim	1-Wire bus	0x31
CRC8-DARC	Data Radio Channel	0x39

Окончание таблицы 1.1

CRC8-WCDMA	mobile networks	0x9B
CRC16-CCITT	X.25, XMODEM, Bluetooth, SLIP	0x1021
CRC16-IBM	Modbus, USB, ANSI X3.28	0x8005
CRC16-Profibus	fieldbus networks	0x1DCF
CRC32	Ethernet, ZigBee, WinRaR, PKZIP, PNG, MPEG-2, SATA	0x04C11DB7
CRC64-ECMA	ECMA-182	0x42F0E1EBA9EA3693
CRC64-ISO	ISO 3309 (HDLC)	0x000000000000001B

1.2.3. Параметрическая модель CRC-алгоритма

Параметрическая модель CRC-алгоритма предложена австралийским ученым в области информатики Ross Williams [46] и подразумевает краткое описание основных параметров, применяемых в реализации алгоритмов вычисления CRC. Модель должна параметризировать алгоритм так, чтобы отражать поведение различных реальных алгоритмов и включает следующие параметры:

- Name – имя, присвоенное данному алгоритму. Строковое значение.
- Width – степень алгоритма (разрядность CRC), выраженная в битах.
- Poly – образующий полином. Записывается как битовая последовательность, при этом старший бит опускается – он всегда равен единице. Например, многочлен $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$ будет записан как 100110000010001110110110111. Для удобства полученное двоичное представление записывают в шестнадцатеричной форме.
- Init – исходное содержимое регистра CRC на момент запуска вычислений. Указывается в шестнадцатеричной системе счисления. По умолчанию 0x0.
- RefIn – (reflect in) логический параметр. Если он имеет значение «False» («Ложь»), то байты сообщения обрабатываются, начиная с 7-го бита, который считается наиболее значащим, а наименее значащим считается

бит 0. Если параметр имеет значение «True» («Истина»), то каждый байт перед обработкой зеркально отражается.

- RefOut – логический параметр. Если он имеет значение «False» («Ложь»), то конечное содержимое регистра сразу передается на стадию XorOut, в противном случае, когда параметр имеет значение «True» («Истина»), содержимое регистра зеркально отражается перед передачей на следующую стадию вычислений.
- XorOut – значение, обозначаемое шестнадцатеричным числом, с разрядностью, равной Width. Комбинируется с конечным содержимым регистра (после стадии RefOut) прежде, чем будет получено окончательное значение контрольной суммы.
- Check – данное поле служит контрольным значением, которое может быть использовано для проверки правильности реализации алгоритма. Поле содержит контрольную сумму, рассчитанную для ASCII строки «123456789».

В таблице 1.2 приведен пример модели CRC-алгоритма.

Таблица 1.2 – Параметрическая модель CRC-алгоритма для Ethernet

Name	CRC32
Width	32
Poly	0x04C11DB7
Init	0xFFFFFFFF
RefIn	True
RefOut	True
XorOut	0xFFFFFFFF
Check	0xCBF43926

Для одного и того же набора данных и образующего полинома можно получить разные значения контрольной суммы CRC, поэтому при программной или аппаратной реализациях алгоритмов необходимо указывать полную модель вычисления.

В настоящее время на практике применяются два основных алгоритма вычисления контрольной суммы CRC: классический (побитовый) и табличный.

Данные алгоритмы принципиально отличаются быстродействием, сложностью программной и аппаратной реализаций и требуемыми для реализации ресурсами (памяти программ и данных, логические элементы). Основная проблема в данном случае заключается в поиске наиболее компромиссного решения между медленным, но наименее аппаратно затратным классическим алгоритмом, и быстрым, но требовательным к ресурсам табличным алгоритмом.

Циклические избыточные коды, несмотря на малую избыточность, не позволяют непосредственно исправлять ошибки без запроса повторной передачи и применяются в системах передачи данных, в которых повторная либо многократная посылка одного и того же пакета данных допустима (Ethernet, MobBus, SLIP). В случаях, когда повторная передача данных не допустима или должна быть сведена к минимуму необходимо применять прямую коррекцию ошибок (FEC–Forward Error Correction) на основе помехоустойчивых кодов, исправляющих ошибки без запроса (либо с минимальной вероятностью запроса) на повторную передачу данных.

1.3. Коды для исправления пакетных или независимых ошибок

Для обеспечения целостности данных в процессе передачи (хранения) без повторной передачи одного и того же блока данных применяют прямую коррекцию ошибок на основе помехоустойчивых кодов. Известно большое количество помехоустойчивых кодов, отличающихся структурой, функциональным назначением, энергетической эффективностью, алгоритмами кодирования и декодирования.

1.3.1. Систематические помехоустойчивые коды

Коды Хэмминга

Код Хэмминга позволяет исправлять однократные и обнаруживать двукратные ошибки [1]. Идея кодов Хемминга заключается в разбиении данных на блоки фиксированной длины и вводе в эти блоки контрольных (избыточных) бит, дополняющих до четности несколько пересекающихся групп,

охватывающих все биты блока. Количество контрольных разрядов k для исправления всех однократных ошибок кодового слова длины $n = m + k$. рассчитывается по формуле

$$k=2^k - m - 1, \quad (1.1)$$

где m – длина информационного блока, k –длина контрольного блока.

Контрольные (проверочные) разряды размещаются среди информационных бит по позициям, равным степеням двойки. Каждый контрольный бит проверяет определенную группу информационных бит, при этом значения контрольного бита равно сумме по модулю два всех битов группы, которую он проверяет. Для определения контролируемых групп информационных бит необходимо разложить порядковые номера информационных бит по степени двойки. Так для сообщения длины m = 8 бит код Хэмминга будет иметь вид, представленный на рисунке 1.2, где Ki – контрольный бит, Иi – информационный бит. При этом, согласно формуле (1.1), k = 4. Сверху над разрядами обозначены позиции разрядов в кодовом слове.

1	2	3	4	5	6	7	8	9	10	11	12
К1	К2	И1	К3	И2	И3	И4	К4	И5	И6	И7	И8

Рисунок 1.2 – Формат кодового слова Хэмминга

Проверочные биты, которыми контролируются информационные, определяются следующим образом:

$И_1$: поз. 3 => $3 = 2^0 + 2^1 = 1 + 2$ => контролируется битами К1 и К2

$И_2$: поз. 5 => $5 = 2^0 + 2^2 = 1 + 4$ => контролируется битами К1 и К3

$И_3$: поз. 6 => $6 = 2^1 + 2^4 = 2 + 4$ => контролируется битами К2 и К3

$И_4$: поз. 7 => $7 = 2^0 + 2^1 + 2^2 = 1 + 2 + 4$ => контролируется битами К1, К2 и К3

$И_5$: поз. 9 => $9 = 2^0 + 2^3 = 1 + 8$ => контролируется битами К1 и К4

$И_6$: поз. 10 => $7 = 2^1 + 2^3 = 2 + 8$ => контролируется битами К2 и К4

$И_7$: поз. 11 => $7 = 2^0 + 2^1 + 2^3 = 1 + 2 + 8$ => контролируется битами К1, К2 и К4

$И_8$: поз. 12 => $12 = 2^2 + 2^3 = 4 + 8$ => контролируется битами К3 и К4

По полученным расчетам можно определить группы информационных бит для каждого контрольного бита:

$К_1 = И_1 + И_2 + И_4 + И_5 + И_7$

$К_2 = И_1 + И_3 + И_4 + И_6 + И_7 + И_8$

$К_3 = И_2 + И_3 + И_4 + И_8$

$К_4 = И_5 + И_6 + И_7 + И_8$

Таким образом, зная значения контрольных бит, можно получить кодовое слово (КС), передаваемое по каналу связи. Для исправления однократной ошибки необходимо на приёмной стороне осуществить контроль четности каждой информационной группы:

$К_1$: $К_1 + И_1 + И_2 + И_4 + И_5 + И_7$

$К_2$: $К_2 + И_1 + И_3 + И_4 + И_6 + И_7$

$К_3$: $К_3 + И_2 + И_3 + И_4 + И_8$

$К_4$: $К_4 + И_5 + И_6 + И_7 + И_8$

В тех группах, где сумма информационных и контрольных бит не равна нулю, содержится ошибка. Если ошибка только одна, то она должна быть в одном из битов, принадлежащих обеим группам. Для того, чтобы уточнить, в каком именно бите произошла ошибка, нужно обратиться к группам, в которых проверка на четность прошла успешно, а, следовательно, в этих группах все биты корректны. Таким образом, исправление однократной ошибки осуществляется систематически с применением линейных комбинаций информационных и контрольных бит.

Преимуществами кодов Хэмминга являются простота аппаратной реализации, небольшая избыточность (для кодирования 7 бит данных избыточность составляет чуть больше 57%, для кодирования 256 бит избыточность равна 3,5%, а для 1024 – 1%). Существенным недостатком является ограниченное количество исправляемых ошибок. Это ограничивает область применения данного типа кодов.

Существуют также циклические коды Хэмминга, в которых контрольные разряды размещаются в конце кодового слова. Построение таких кодов основано на операциях с образующими многочленами (полиномами), а их декодирование осуществляется способами, применяемыми для циклических кодов.

Код Голея

Данный вид кодов, позволяющий обнаруживать и исправлять ошибки кратности до трех включительно, был открыт швейцарским математиком Голеем. Применялся в программном обеспечении космических аппаратов Вояджер для передачи изображений. В настоящее время применяется в радиостанциях, работающих на низких радиочастотах и передающих информацию друг другу за несколько тысяч километров.

В основе идеи кода Голея лежит формула [43, 49] для определения количества проверочных разрядов *k,* которая имеет следующий вид:

$C^0_n + C^1_n + \ldots + C^{t_i}_n = 2^k$, где t_i – количество исправляемых ошибок, а C^i_n – число сочетаний из n по i. Занимаясь поиском совершенных кодов, Голей заметил, что $C^0_{23} + C^1_{23} + C^3_{23} = 2^{11}$. Это говорит о возможности существования совершенного двоичного кода (23,12), где n = 23, m = 12, k =11, исправляющего все ошибки кратности до трёх включительно. В данном случае под совершенным кодом подразумевается такой код, в котором число избыточных (контрольных) разрядов ровно столько, сколько нужно для исправления заданного количества ошибок. На данный момент известно только 2 вида совершенных кодов – это код Хэмминга (для однократных ошибок) и код Голея (для трёхкратных ошибок) [43].

Для декодирования кодов Голея применяется циклический алгоритм декодирования, описанный в разделе 3.4.2. Недостатком кода является его ограниченная длина: код можно построить только для длины информационного блока, равной 12. Это сужает область применения данного типа кодов.

1.3.2. Полиномиальные циклические коды

Полиномиальные коды подразумевают представление кодового слова $C = (c_0,c_1,\ldots,c_{n-1})$ в виде многочлена $C(x) = c_0 + c_1*x + \ldots + c_{n-1}*x^{n-1}$. Применяя данное обозначение, можно определить полиномиальный код как множество всех многочленов степени, не большей $n – 1$, содержащих в качестве множителя некоторый фиксированный многочлен $G(x)$, называемым *образующим многочленом* (*полиномом*) [43].

В полиномиальных циклических кодах любой циклический сдвиг любого кодового слова является другим кодовым словом [43]. В полиномиальном представлении циклический сдвиг на одну позицию $C^1(x)$ соответствует умножению на x по модулю $(x^n – 1)$:

$$C^1(x)= xC(x)mod(x^n – 1).$$

Операция циклического сдвига реализуется на регистрах сдвига (рисунок 1.3).

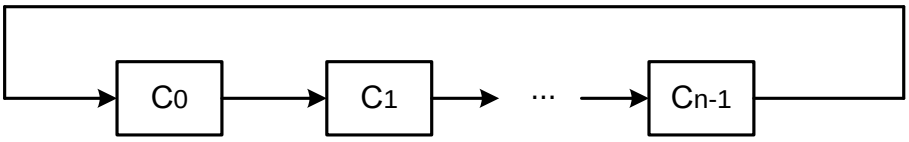


Рисунок 1.3 – Сдвиг кодового слова вправо

В теории циклических кодов для получения образующих многочленов используют многочлены, делящиеся без остатка на себя и на единицу, которые также называют *неприводимым минимальным многочленом (полиномом)*. Образующий многочлен может быть равен неприводимому минимальному многочлену $\Phi(x)$ или являться произведением нескольких неприводимых многочленов:

$$G(x) = \Phi_1(x)\Phi_2(x)\ldots\Phi_i(x).$$

Построение полиномиальных циклических кодов

Одним из способов построения циклического кода (кодирования) является применение образующей матрицы $G(x)$ [50]. Несмотря на то, что на практике применяются в основном циклические коды, принадлежащие к классу систематических кодов, существуют также несистематические циклические

коды. Для несистематических циклических кодов образующая матрица получается в результате *m*-кратного циклического сдвига первой строки образующей матрицы, которая в свою очередь получается дополнением *G(x)* слева нулями до длины кодовой комбинации, равной *n* [50]. В полиномиальном виде образующая матрица для несистематического циклического кода будет иметь вид:

$$G = \begin{vmatrix} G(x) \\ xG(x) \\ x^2G(x) \\ \dots\dots\dots \\ x^{m-1}G(x) \end{vmatrix}$$

Для того чтобы получить кодовое слово необходимо информационное сообщение $M = (M0,M1,M2,\dots..,Mm\text{-}1)$ умножить по модулю 2 на матрицу *G*:

$$C(x) = M(x) \cdot G = M_0 \cdot G(x) \text{ xor } M_1 \cdot xG(x) \text{ xor } M_2 \cdot x^2G(x) \text{ xor} \dots \text{xor } M_{m-1} \cdot x^{m-1}G(x))$$

Для построения несистематического циклического кода без образующей матрицы необходимо полином, представляющий информационный блок, умножить по модулю 2 на образующий полином [50]:

$$C(x) = M(x) \cdot G(x).$$

Для построения образующей матрицы систематического циклического кода необходимо составить диагональную транспонированную матрицу E^T, в которой число строк и столбов равно длине информационного блока, а также дополнительную матрицу G', в которой размещаются остатки от деления полиномов $x^{n-k} \dots x^{n-1}$ на образующий полином по следующему принципу:

$$G' = \begin{vmatrix} x^{n-1} mod G(x) \\ x^{n-k+1} mod G(x) \\ \dots\dots\dots \\ x^{n-k} mod G(x) \end{vmatrix}$$

Например, для кода (15,7), где $n = 15$, $m = 7$, $G(x) = x^8+x^7+x^6+x^4+1$ диагональная матрица будет иметь вид:

$$E^T = \begin{vmatrix} 0\,0\,0\;\;0\,0\,0\,1 \\ 0\,0\,0\,0\,0\,1\,0 \\ 0\,0\,0\,0\,1\,0\,0 \\ 0\,0\,0\,1\,0\,0\,0 \\ 0\,0\,1\,0\,0\,0\,0 \\ 0\,1\,0\,0\,0\,0\,0 \\ 1\,0\,0\,0\,0\,0\,0 \end{vmatrix}$$

Дополнительная матрица будет иметь вид:

$$G^{'} = \begin{vmatrix} 1\,1\,1\,0\,1\,0\,0\,0 \\ 0\,1\,1\,1\,0\,1\,0\,0 \\ 0\,0\,1\,1\,1\,0\,1\,0 \\ 0\,0\,0\,1\,1\,1\,0\,1 \\ 1\,1\,1\,0\,0\,1\,1\,0 \\ 0\,1\,1\,1\,0\,0\,1\,1 \\ 1\,1\,0\,1\,0\,0\,0\,1 \end{vmatrix}$$

Образующая матрица получается путём совмещения диагональной и дополнительной матриц:

$$G = \begin{vmatrix} 0\,0\,0\,0\,0\,0\,1\,1\,1\,1\,0\,1\,0\,0\,0 \\ 0\,0\,0\,0\,0\,1\,0\,0\,1\,1\,1\,0\,1\,0\,0 \\ 0\,0\,0\,0\,1\,0\,0\,0\,0\,1\,1\,1\,0\,1\,0 \\ 0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,1\,1\,1\,0\,1 \\ 0\,0\,1\,0\,0\,0\,0\,1\,1\,1\,0\,0\,1\,1\,0 \\ 0\,1\,0\,0\,0\,0\,0\,0\,1\,1\,1\,0\,0\,1\,1 \\ 1\,0\,0\,0\,0\,0\,0\,0\,1\,0\,1\,0\,0\,0\,1 \end{vmatrix}$$

Систематическое кодирование с помощью образующей матрицы осуществляется следующим образом:

$$C(x) = M(x) \cdot x^{n-k} \; xor \; M(x) \cdot x^{n-k} \cdot G. \qquad (1.2)$$

Из выражения 1.2 видно, что в систематическом коде информационные (выражение $M(x) \cdot x^{n-k}$) и контрольные разряды (выражение $M(x) \cdot x^{n-k} \cdot G$) кодового слова разделены, при этом контрольные символы являются линейной комбинацией информационных символов.

Без применения образующей матрицы систематический циклический код строится следующим образом:

$$C(x) = M(x) \cdot x^{n-k} \; xor \; M(x) \cdot x^{n-k} \; mod \; G(x).$$

Таким образом, в циклических кодах вычисление остатка от деления эквивалентно умножению по модулю 2 на предварительно вычисленную образующую матрицу.

Декодирование полиномиальных циклических кодов

Задача декодирования полиномиальных циклических кодов сводится к поиску полинома ошибок $E(x)$ в уравнении $C_{err}(x) = C(x) + E(x)$.

Для определения полинома ошибок $E(x)$ используется синдром ошибки $S(x)$, который определяется выражением:

$$S(x) = E(x) \; mod \; G(x) = C_{err}(x) \; mod \; G(x). \qquad (1.3)$$

На рисунке 1.4 приведена обобщенная схема декодирования циклических кодов.

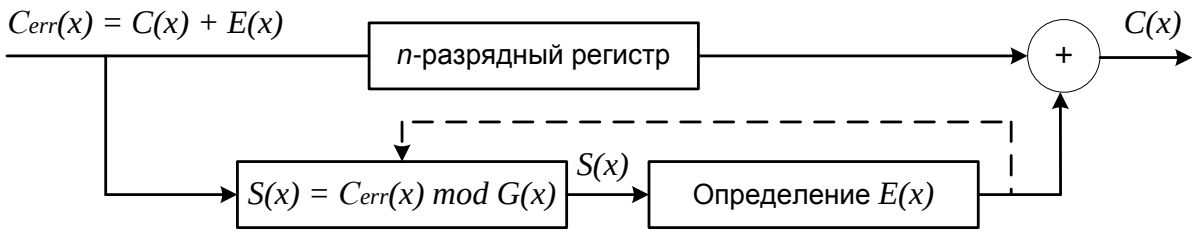


Рисунок 1.4 – Структурная схема поиска полинома ошибок

Уравнение (1.3) является основой для построения табличного (синдромного) декодера циклического кода, в котором синдром *S(0,1)* является адресом соответствующего шаблона ошибки *E(0,1)* [50]. При аппаратной реализации такого декодера требуется $n \cdot 2^k$ (где n – длина кодового слова, k – длина контрольного блока) бит памяти для хранения таблицы шаблонов ошибок.

Другой вариант декодера основан на циклическом алгоритме декодирования [50], который заключается в последовательном выполнении нескольких этапов:

1) Кодовое слово с ошибкой $C_{err}(0,1)$ делится на образующий полином *G(0,1)*;

2) Вычисляется вес (количество единиц кода) *w* синдрома *S(0,1)*. Если $w \leq t$ (t – количество исправляемых ошибок), то $C(0,1) = C_{err}(0,1)$ *xor* $S(0,1)$ и исправленное кодовое слово подаётся на выход декодера. *Если* $w > t$, то осуществляется переход на этап 3.

3) Кодовое слово *C(0,1)* циклически сдвигается на 1 разряд влево (или вправо). Полученное кодовое слово $C'(0,1)$ делится на образующий полином *G(0,1)*. Если $w \leq t$, то $C(0,1) = C'(0,1)$ *xor* $S(0,1)$, результат циклически сдвигается обратно (относительно направления сдвига до деления на *G(0,1)*) на количество разрядов, равное количеству предшествующих сдвигов. Исправленное кодовое слово подаётся на выход декодера. *Если* $w > t$, то этап 3 повторяется. Если количество сдвигов достигает $n - 1$ и ошибка не обнаружена (условие $w \leq t$ не выполнилось ни на одной из итераций), то ошибка считается неисправимой для данного декодера.

Для реализации циклического алгоритма декодирования не требуется память для хранения шаблонов ошибок. Однако данный алгоритм значительно уступает в быстродействии табличному алгоритму в виду многократного применения операций циклического сдвига и деления полиномов. Также

недостатком описанного циклического алгоритма является невозможность исправлять ошибки для кодов, в которых $m > k$, т.е. когда длина информационного блока больше длины контрольного блока. Данный недостаток связан с понятием *скользящее окно* – область кода, в которой ошибки заданной кратности будут гарантировано исправляться циклическим алгоритмом декодирования. Длина скользящего окна равна длине контрольного блока кода, поэтому в случаях, когда длина сообщения $m > k$ циклический алгоритм декодирования будет работать не для всех возможных ошибок кратности t. Это обусловлено тем, что некоторые ошибки кратности t выйдут за диапазон скользящего окна. Табличный алгоритм декодирования данным недостатком не обладает.

Таким образом, для декодирования циклических кодов является актуальным разработка быстродействующего и эффективного алгоритма декодирования, совмещающего достоинства табличного и циклического способов декодирования.

Циклические коды Боуза-Чоудхури-Хоквингема

Коды Боуза-Чоудхури-Хоквингема (БЧХ) представляют собой обширный класс кодов с минимальным кодовым расстоянием $d_{min} \geq 2 \cdot t+1$, исправляющих несколько ошибок. Конструкция кодов БЧХ определяется корнями образующего многочлена [43]. Образующий многочлен циклического кода БЧХ является наименьшим общим кратным (НОК) нечетных минимальных многочленов:

$$G(x) = \text{НОК}[\Phi_1(x), \Phi_3(x), \ldots, \Phi_{2t-1}(x)]$$

где t – количество исправляемых ошибок, $\Phi i(x)$ – минимальный неприводимый многочлен.

Число минимальных многочленов, необходимых для построения кода БЧХ, равно t. Максимальный порядок минимальных многочленов вычисляется по формуле: $p = 2 \cdot t - 1$.

Длина кода БЧХ должна соответствовать выражению $n = 2^h - 1$, где h – любое целое число [50]. В таблице 1.3. приведены минимальные неприводимые многочлены для $h \in [2;7]$ степени многочлена $l = 1,3,5,\ldots 15$.

Таблица 1.3 – Двоичное представление неприводимых многочленов

l	Минимальные неприводимые многочлены при значении степени h						
	2	3	4	5	6	7	8
1	111	1011	10011	100101	1000011	10001001	100011101
3	–	1101	11111	111101	1010111	10001111	101110111
5	–	111	110111	1100111	10011101	111110011	1100110001
7	–	–	11001	101111	1001001	11110111	101101001
9	–	–	–	1101111	1101	10111111	110111101
11	–	–	–	111011	1101101	11010101	111100111
13	–	–	–	–	–	10000011	100101011
15	–	–	–	–	–	–	111010111
17	–	–	–	–	–	–	010011
19	–	–	–	–	–	10000011	101100101
21	–	–	–	–	–	11001011	110001011

Например, для построения образующего полинома кода длиной $n = 15$, исправляющего $t = 2$ ошибки, необходимо выполнить следующие операции:

1) вычислить степень $h = \log_2(n+1) = 4$;
2) из таблицы 1.3 выбрать $t = 2$ минимальных неприводимых многочленов, степень старшего из которых равна $p = 2 \cdot t - 1 = 3$;
3) Вычислить произведение $G(0,1) = \Phi_1(0,1) \cdot \Phi_2(0,1) = 10011 \cdot 11111 = 111010001 \Rightarrow G(x) = x^8+x^7+x^6+x^4+1$.

Построение циклического кода БЧХ осуществляется одним из систематических способов, описанных ранее.

Особенностью и недостатком кодов БЧХ является то, что для исправления числа ошибок $t \geq 2$ еще недостаточно условия, чтобы между комбинациями кода минимальное кодовое расстояние было равно $d_{min}=2 \cdot t+1$. Необходимо также, чтобы длина кода n удовлетворяла условию $n = 2^h - 1$, где h-любое целое число [43].

В таблице 1.4. приведены примеры БЧХ кодов (*m*-длина информационного блока, *k* – длина контрольного блока) для различных длин кодовых слов *n* и кратности исправляемых ошибок *t* [41].

Таблица 1.4 – Параметры кодов БЧХ (*n*, *m*, *k*)

n	*m*	*k*	*t*	*d*	*m/n*
7	4	3	1	3	0,57
15	11	4	1	3	0,73
15	7	8	2	5	0,47
15	5	10	3	7	0,33
31	26	5	1	3	0,84
31	21	10	2	5	0,68
31	16	15	3	7	0,52
31	11	20	5	11	0,35
31	6	25	7	15	0,19
63	57	6	1	3	0,9
63	51	12	2	5	0,81
63	45	18	3	7	0,72
63	39	24	4	9	0,62
63	36	27	5	11	0,57
63	30	33	6	13	0,48
63	24	39	7	15	0,37
63	18	45	10	21	0,29
63	16	47	11	23	0,25
63	10	53	13	27	0,16
63	7	56	15	31	0,11

Первая строка таблицы 1.4 соответствует коду Хэмминга. При увеличении кратности ошибок *t* повышается избыточность кода, что видно из соотношения *m/n*, которое называется *скоростью* кода. Значение длины кода *n* всегда будет нечетным числом: $n = 1,3,5,\ldots(2^h-1)$, т.е. не все длины информационных сообщений *m* могут быть заданы, при этом значение *n* определяет число контрольных символов *k*: $k<=h \cdot t<=[\log_2(n+1)] \cdot t$. Данный недостаток может быть существенным при разработке кодеков помехоустойчивых кодов, в которых исходными параметрами являются длина информационного (неизбыточного) сообщения *m* и количество исправляемых

ошибок *t*. Одним из способов решения проблемы является укорачивание БЧХ кода путём уменьшения разрядности информационного блока с сохранением разрядности контрольного блока. Однако это приводит к повышению избыточности, потере цикличности и эффективности кода. Заметим, что длина и избыточность кода зависит от длины образующего полинома, поэтому одним из способов увеличения длин информационного блока *m* может быть применение такого алгоритма построения образующего полинома, который позволит получить эффективные образующие полиномы для любых значений *m*. Такие полиномы следует искать с помощью программных и вычислительных средств, исходя их двух параметров: *m* – длина информационного блока и *t* – кратность исправляемых ошибок.

Декодирование циклических кодов БЧХ

При реализации декодеров БЧХ кодов обычно используют способ, основанный на применении элементов поля Галуа или конечного поля для нумерации позиций кодового слова [43, 49, 51].

Процесс декодирования (рисунок 1.5) заключается в последовательном выполнении этапов:

1) Вычисление синдромов ошибок;
2) Поиск коэффициентов локаторов ошибок;
3) Поиск позиций ошибок в кодовом слове;
4) Поиск значений ошибок (для недвоичных кодов);
5) Исправление ошибок на вычисленных позициях.

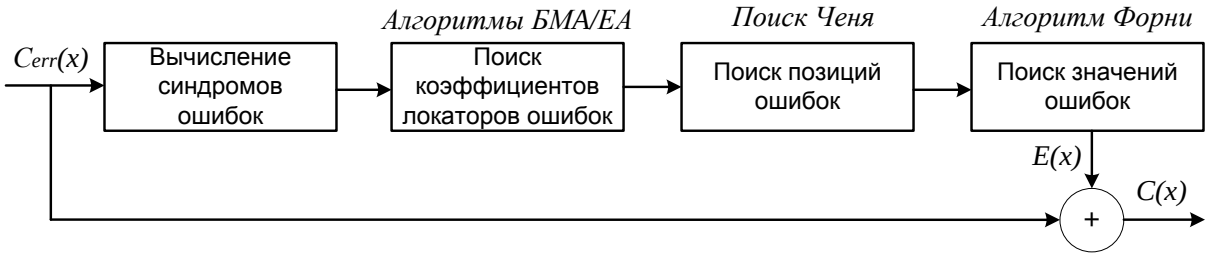


Рисунок 1.5 – Общая схема декодирования БЧХ кодов

Позиции ошибок находятся из решения системы уравнений в поле Галуа $GF(2^k)$, где k – старшая степень образующего многочлена (таблица 1.5). Систему уравнений можно получить, введя многочлен ошибок $E(x)$ и учитывая нули кода α^j для $b \le j \le b + 2t_d - 1$, где t_d – количество ошибок в принятом кодовом слове, при этом количество исправляемых ошибок декодером $t \le t_d$.

Позиции ошибок находятся из решения системы уравнений в поле Галуа $GF(2^k)$, где k – старшая степень образующего многочлена. Систему уравнений можно получить, введя многочлен ошибок $E(x)$ и учитывая нули кода α^j для $b \le j \le b + 2t_d - 1$, где t_d – количество ошибок в принятом кодовом слове, при этом количество исправляемых ошибок декодером $t \le t_d$.

Если кодовое слово с ошибкой задается уравнением $C_{err}(x) = C(x) + E(x)$, то многочлен ошибок определяется как:

$$E(x) = e_{j1}x^{j1} + e_{j2}x^{j2} + \ldots + e_{jt}. \quad (1.4)$$

Множества $\{ e_{j1}, e_{j2}, \ldots, e_{jt} \}$, где $e_{j1} \in \{0,1\}$ (для двоичных кодов), и $\{ \alpha^{j1}, \alpha^{j2}, \ldots, \alpha^{jt}\}$, где $\alpha^j \in GF(2^k)$ являются локаторами ошибок.

Таблица 1.5 – Нумераций позиций КС в полях Галуа

Символы КС	C_0	C_1	C_2	…	C_{n-1}
Локаторы позиций	1	α	α^2	…	α^{n-1}

Исходя из выражения (1.4) и определения полей Галуа, вычисляются синдромы ошибок как значения принятого кодового слова в нулях кода и тем самым выполняется первый этап декодирования (рисунок 1.5):

$$S_1 = C_{err}(\alpha^b) = e_{j1}\,\alpha^{bj1} + \ldots + e_{jt}\,\alpha^{bjt},$$

$$S_2 = C_{err}(\alpha^{b+1}) = e_{j1}\,\alpha^{(b+1)j1} + \ldots + e_{jt}\,\alpha^{(b+1)jt},$$

$$\cdot S_{2td} = C_{err}(\alpha^{b+2td-1}) = e_{j1}\,\alpha^{(b+2td-1)j1} + \ldots + e_{jt}\,\alpha^{(b+2td-1)jt}.$$

Поиск коэффициентов локаторов ошибок (этап 2 процесса декодирования) заключается в решении ключевого уравнения:

$$\begin{pmatrix} S_1 \\ S_2 \\ \vdots \\ S_{2t} \end{pmatrix} = \begin{pmatrix} S_1 & S_2 & \cdots & S_t \\ S_2 & S_3 & \cdots & S_{t+1} \\ \vdots & \vdots & \ddots & \vdots \\ S_t & S_{t+1} & \cdots & S_{2t-1} \end{pmatrix} \begin{pmatrix} \sigma_t \\ \sigma_{t-1} \\ \vdots \\ \sigma_1 \end{pmatrix}. \qquad (1.5)$$

При этом многочлен локаторов ошибок имеет вид:

$$\sigma_t(x) = \prod_{l=1}^{t}(1 + \alpha^{j_l}x) = 1 + \sigma_1 x + \sigma_2 x^2 + \ldots + \sigma_t.$$

Решение уравнения (1.5) является трудоёмкой задачей и требует интенсивных вычислений. Известны различные алгоритмы решения ключевого уравнения, ориентированные как на программные (алгоритм Берлекемпа-Месси, алгоритм Питерсона-Горленштейна-Цирлера), так и на аппаратные реализации (Евклидов алгоритм) [43, 53–56].

Алгоритм Берлекемпа-Месси

Алгоритм Берлекемпа-Месси [52, 53] представляет собой итеративный процесс построения многочлена обратной связи $\sigma^{(i+1)}(x)$ наименьшей степени, удовлетворяющего следующему уравнению, полученному из (1.5):

$$\sum_{j=0}^{l_i+1} S_{k-j}\sigma_j^{(i+1)} = 0, \text{ где } l_i < k < i + 1. \quad (1.6)$$

Решение этой задачи эквивалентно условию, что многочлен

$$\sigma^{(i+1)}(x) = 1 + \sigma_1^{(i+1)}x + \cdots + \sigma_{l_{i+1}}^{(i+1)}x^{l_{i+1}}$$

является многочленом обратной связи линейного регистра сдвига с обратной связью (ЛРОС), который генерирует ограниченную последовательность синдромов.

При решении данной задачи вводится понятие *несовместность* – расхождение на i-ой итерации, которое определяется как:

$$d_i = S_{i+1} + S_i\sigma_1^{(i)} + \cdots + S_{i-l_i+1}\sigma_{l_i}^{(i)}. \qquad (1.7)$$

Выражение (1.7) является мерой соответствия синдромной последовательности и генерируемой ЛРОС, а также содержит корректирующий множитель для вычислений на последующих итерациях.

Если $d_i = 0$, то уравнение (1.6) будет выполняться при:

$$\sigma^{i+1}(x) = \sigma^{i}(x),\ l_{i+1} = l_i.$$

Если $d_i \neq 0$, то решение на следующей итерации примет вид:

$$\sigma^{i+1}(x) = \sigma^{i}(x) + d_i d_m^{-1} x^{i-m},$$

$$l_{i+1} = \max\{l_i, l_m + i - m\}.$$

При этом должно выполняться условие: *-1* $\leq m \leq i$.

Итеративное выполнение алгоритма продолжается до тех пор, пока выполняются одно или оба из условий проверки:

1) либо $i \geq l_{i+1} + t_d - 1$;
2) либо $i = 2t_d - 1$

При этом начальными условиями алгоритма являются:

$\sigma^{(-1)}(x) = 1,\ l_{-1} = 0,\ d_{-1} = 1, \sigma^{(0)}(x) = 1,\ l_0 = 0,\ d_0 = S_1.$

По числу операций в конечном поле *GF(2^k)* БМА считается эффективным, однако алгоритм не является хорошо адаптированным для аппаратных реализаций ввиду наличия последовательных (зависимых) итераций и условных переходов. Вследствие этого используется в основном в программных реализациях [54] для моделирования работы кодеков БЧХ кода. Для аппаратного решения (на ПЛИС) ключевого уравнения наиболее приемлемым является Евклидов алгоритм [43], идея которого заключается в поиске наибольшего общего делителя (НОД) двух полиномов.

Алгоритм Евклида

В алгоритме Евклида [43] для нахождения локатора ошибок определяется *полином значений ошибок* через *синдромный полином* в виде $E(x) = \sigma(x)S(x)$, где $S(x) = 1 + S_1 x + \cdots + S_{2td} x^{2td}$.

Из системы уравнений (1.5) следует, что $E(x) = \sigma(x)S(x) \mathrm{mod}(x^{2td+1})$. Для поиска многочлена *E(x)* применяется *расширенный алгоритм Евклида* к многочленам $r_0(x) = x^d$, где $d = 2t_d + 1$ и $r_1(x) = S(x)$. Если на *j*-ом шаге алгоритма получено решение $r_j(x) = a_j(x)x^{2td+1} + b_j(x)S(x)$, причем $\deg[r_j(x)] \leq t_d$, то $E(x) = r_j(x)$ и $\sigma_0(x) = b_j(x)$.

Непосредственно процесс декодирования является итеративным и заключается в решении задачи вычисления НОД $(r_0(x), r_1(x))$:

1) Входные значения: $r_0(x), r_1(x), \deg[r_0(x)] \geq \deg[r_1(x)]$;
2) Начальные условия: $a_0(x) = 1, b_0(x) = 0, a_1(x) = 0, b_1(x) = 1$;
3) На j-ом шаге, при условии, что j≥2, выполняется длинное деление для многочленов $r_{j-2}(x)$ *и* $r_{j-1}(x)$.

   В результате будет получено

   $r_{j-2}(x) = q_j(x)r_{j-1}(x) + r_j(x), \quad 0 \leq \deg[r_j(x)] \leq \deg[r_{j-1}(x)]$;
4) Вычисляется

   $a_j(x) = a_{j-2}(x) - q_j(x)a_{j-1}(x), \; b_j(x) = b_{j-2}(x) - q_j(x)b_{j-1}(x)$;
5) Останавливаются вычисления на итерации *last*=j_{last} в случае, если $\deg[r_{last}(x)] = 0$;
6) Выходные значения: НОД $(r_0(x), r_1(x)) = r_k(x)$, где *k*-такое наибольшее ненулевое целое, при котором будут выполняться неравенства:

   $r_k(x) \neq 0$ и $k \leq j_{last}$.

Регулярность структуры алгоритма Евклида способствует широкому применению его в аппаратных реализациях декодеров кодов БЧХ и Рида-Соломона.

Алгоритм Питерсона-Горленштейна-Цирлера

Существует также алгоритм Питерсона-Горленштейна-Цирлера [55], являющийся прямым решением системы уравнений (1.5), однако его сложность растёт как t^3, где t – корректирующая способность кода, поэтому он применим только для небольших значений *t*.

Основной проблемой прямого решения системы (1.5) является то, что неизвестно действительное число ошибок в кодовом слове. Необходимо проверять гипотезу о том, что число ошибок равно *t*.

Работа алгоритма начинается с предположения о том, что кодовое слово содержит максимальное количество ошибок $t_{max} = t_d$, а также вычисления определителя Δ_i для $i = t_{max} = t_d$:

$$\Delta_i = det\begin{pmatrix} S_1 & S_2 & \cdots & S_i \\ S_2 & S_3 & \cdots & S_{i+1} \\ \vdots & \vdots & \ddots & \vdots \\ S_i & S_{i+1} & \cdots & S_{2i-1} \end{pmatrix}.$$

Если $\Delta_i = 0$, то действительное число ошибок меньше, чем t_d. Значение i уменьшается на единицу и снова проверяется условие $\Delta_i = 0$. Процесс повторяется пока $i > 1$.

Если $\Delta_i \neq 0$, то вычисляется обратная матрица синдромов для вычисления значения $\sigma_1, \sigma_2, \ldots, \sigma_t$ из уравнения (1.5).

Если $\Delta_i \neq 0$ и $i = 1$, то регистрируется ошибка, неисправимая для декодера.

Все изложенные алгоритмы позволяют с разной степенью сложности реализовать второй и самый трудоёмкий этап декодирования БЧХ-кодов (рисунок 1.5), заключающий в поиске коэффициентов многочлена локаторов ошибок. Для поиска конкретных позиций ошибок в кодовом слове с помощью найденных локаторов применяется алгоритм поиска Ченя (3-й этап декодирования).

Алгоритм поиска Ченя

Алгоритм Ченя [43] заключается в поиске корней многочлена локаторов ошибок $\sigma(x)$ методом проб и ошибок, который подразумевает полный перебор всех ненулевых элементов поля $\beta \in GF(2^k)$, которые берутся по порядку $1, \alpha$ и т.д. с проверкой условия $\sigma(\beta^{-1}) = 0$. Значения β, для которых $\sigma(\beta^{-1}) = 0$, будут указывать на номера позиций кодового слова, в которых содержатся ошибочные символы.

Коды Рида-Соломона

Пакетные ошибки, возникающие в процессе передачи (хранения) данных, являются частным случаем независимых ошибок. В помехоустойчивом

кодировании для исправления такого типа ошибок применяются недвоичные коды Рида-Соломона [4, 5, 43, 56], которые можно интерпретировать как частный случай *недвоичных* кодов БЧХ. Данные коды обладают меньшей избыточностью, но при этом их длины также ограничены согласно выражению $2^h - 1$, где h – натуральное число. Важной особенностью является то, что единица измерения длины кодового слова – это один из элементов поля Галуа $GF(2^k)$, где k – старшая степень образующего многочлена. Таким образом, в качестве параметров (n,m) кодов Рида-Соломона указывается количество символов (n–длина кодового слова, m – длина информационного блока), разрядность каждого из которых равна k бит. В частности, нулями РС кода, исправляющего t ошибок, являются $2t$ последовательных степеней примитивного элемента поля Галуа. Образующий многочлен для кода РС задаётся выражением

$$G(x) = \prod_{j=b}^{b+2td-1}(x - \alpha^i),$$

где b – целое число, обычно равное 0 или 1.

Принцип декодирования кодов РС аналогичен с декодированием двоичных кодов БЧХ (рисунок 1.5), за исключением нескольких отличий:

- образующий многочлен отличен от кодов БЧХ;
- коды ориентированы на исправление *пакетов ошибок*;
- значения ошибок после выполнения этапа поиска позиций вычисляются с помощью алгоритма Форни [43], так как код является недвоичным.

1.4. Основные результаты и выводы по главе

На основании результатов анализа способов и алгоритмов обнаружения и исправления пакетных и независимых ошибок можно сделать следующие выводы.

- Задача разработки алгоритмов кодирования циклическими избыточными кодами (CRC), обнаруживающими ошибки, является актуальной с точки

зрения поиска компромисса по быстродействию и объёму памяти между классической (побитной) и табличной реализациями.

- Задача разработки алгоритма поиска образующих полиномов для построения эффективных помехоустойчивых кодов, исправляющих независимые или пакетные ошибки, является актуальной как альтернатива существующим способам построения образующих полиномов для кодов БЧХ и Рида-Соломона без ограничения на длину информационного блока и с меньшей избыточностью, чем у укороченных БЧХ кодов. Ввиду трудоёмкости такой задачи необходимо применение технологий параллельных вычислений.
- Задача разработки устройств исправления пакетных и независимых ошибок на ПЛИС с применением циклического алгоритма декодирования и его модификаций для кодов малой длины является актуальной с точки зрения практического применения алгоритмов исправления ошибок с лучшим быстродействием, чем у существующих устройств и меньшими аппаратными затратами, чем в случае применения известных помехоустойчивых кодов БЧХ.

Таким образом, **целью диссертационного исследования** является разработка эффективных алгоритмов, программ и устройств обнаружения и исправления пакетных или независимых ошибок при передаче и хранении данных с информационным сообщением короткой длины.

Для достижения поставленной цели необходимо решение следующих задач.

1. Разработать алгоритм быстрого вычисления циклических избыточных кодов (CRC).
2. Поставить компьютерные эксперименты и эксперименты на микроконтроллерах и ПЛИС по вычислению контрольной суммы CRC различными алгоритмами.

3. Разработать алгоритм и программу поиска образующего полинома для построения ЦПК, исправляющих независимые ошибки. Провести компьютерный эксперимент по поиску образующих полиномов с применением технологий параллельных вычислений.
4. Осуществить аппаратную реализацию алгоритмов декодирования ЦПК, исправляющих независимые ошибки на ПЛИС, для оценки быстродействия и аппаратных затрат.
5. Разработать алгоритм и программу поиска образующего полинома для построения ЦПК, исправляющих пакетные ошибки. Провести компьютерный эксперимент по поиску образующих полиномов с применением технологий параллельных вычислений.
6. Осуществить аппаратную реализацию алгоритмов декодирования ЦПК, исправляющих пакетные ошибки на ПЛИС, для оценки быстродействия и аппаратных затрат.

ГЛАВА 2. АЛГОРИТМЫ, ПРОГРАММЫ И УСТРОЙСТВА ОБНАРУЖЕНИЯ ОШИБОК

Циклический избыточный код CRC позволяет осуществлять контроль целостности данных в процессе передачи или хранения путём обнаружения ошибок без прямого исправления. Для реализации исправления ошибок (за счёт многократного переспроса) необходимо наличие в системе передачи данных стабильного канала обратной связи.

2.1. Алгоритмы вычисления CRC

Существуют разные алгоритмы вычисления контрольной суммы CRC [46, 57], отличающиеся сложностью программной и аппаратной реализации, быстродействием и требуемым для реализации ресурсами (память программ и данных, логические элементы).

2.1.1. Классический алгоритм

Классический алгоритм [57] вычисления контрольной суммы CRC заключается в побитовом делении полинома, представляющего данные (файл), на образующий полином. Реализация алгоритма основа на последовательном сдвиге данных через линейный регистр с обратной связью. После выполнения всех сдвигов в регистре содержится контрольная сумма CRC. На рисунке 2.1 приведена схема вычисления CRC32 по классическому алгоритму с образующим полиномом $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$ (старший бит опускается).

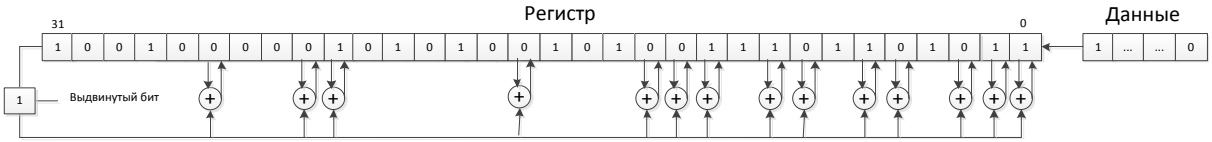


Рисунок 2.1 – Схема вычисления CRC32 по классическому алгоритму

Далее представлено пошаговое описание классического алгоритма вычисления CRC.

Начало. Регистр разрядностью 32 бита содержит значение *Init* согласно параметрической модели алгоритма вычисления CRC32 [46].

1) Осуществляется сдвиг данных в регистре на один бит вправо (влево).

2) Очередной бит из потока данных заносится в регистр.

3) Бит переноса, полученный в шаге 1, складывается по модулю два с разрядами регистра, которые соответствуют степеням образующего полинома $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$.

4) Если не все биты данных поступили в регистр, то осуществляется переход на шаг 1, иначе алгоритм завершает работу.

Конец.

Недостатком классического алгоритма является низкое быстродействие вычисления CRC, поэтому на практике, наиболее часто в протоколах передачи данных применяется табличный алгоритм [46], позволяющий вычислять CRC побайтно. Существуют разные варианты табличного алгоритма: прямой и обратный. Каждый из вариантов алгоритмов для одного и того же набора данных вычисляет разную контрольную сумму CRC.

2.1.2. Прямой табличный алгоритм

Идея табличного алгоритма заключается в предварительном вычислении специальной таблицы остатков от деления по модулю 2 определенных наборов данных на образующий полином. В качестве наборов данных выступают всевозможные значения байта данных (от 0 до 255, дополненные справа нулями до длины образующего полинома). Адресация элементов таблицы осуществляется на основе старшего байта регистра. Далее приведены схема (рисунок 2.2) и пошаговое описание алгоритма.

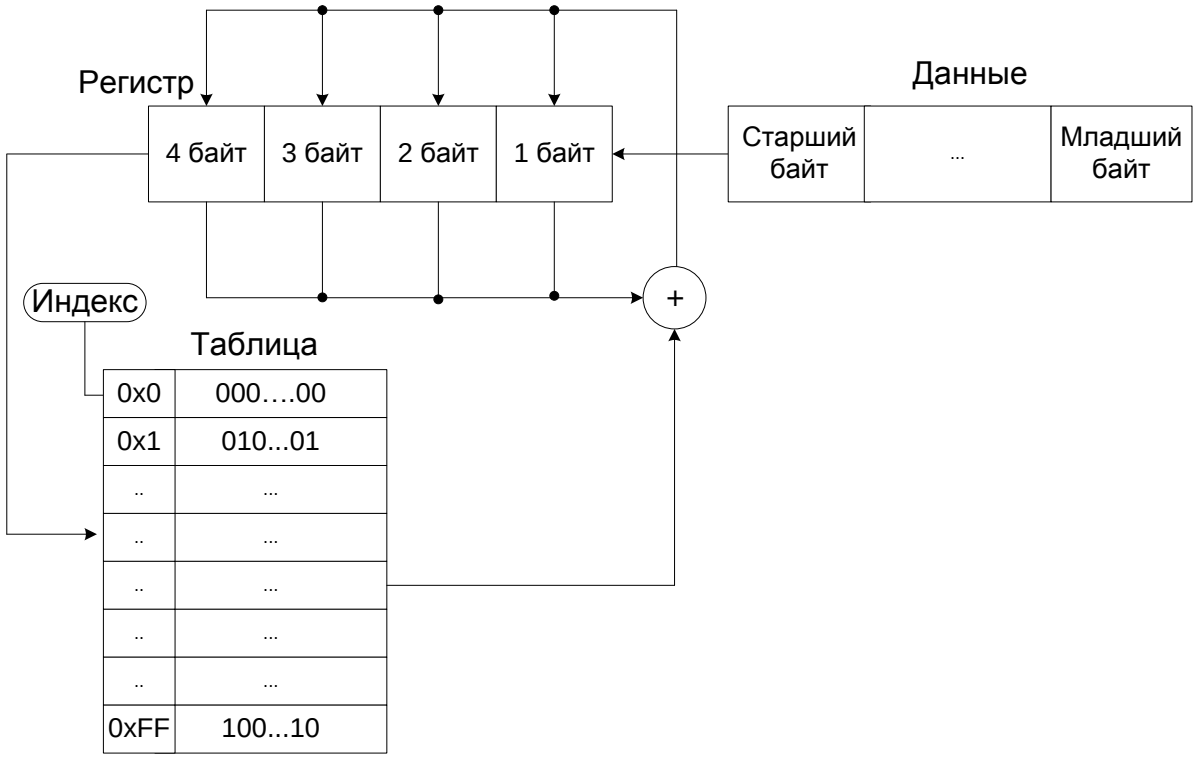


Рисунок 2.2 – Схема вычисления CRC по табличному алгоритму

Начало. Регистр CRC разрядностью 32 бита содержит значение *Init* согласно параметризированной модели алгоритма.

Шаг 1. В младший байт регистра CRC загружается старший байт из потока данных.

Шаг 2. Данные в регистре сдвигаются на один байт влево. Старший байт регистра CRC является адресом элемента таблицы.

Шаг 3. Выбранное из таблицы значение по адресу, полученному на шаге 2, складывается по модулю два с регистром CRC. Результат записывается в регистр CRC.

Шаг 4. Если ещё не все байты из потока данных прошли через регистр, то переходим на шаг 1, иначе алгоритм завершает работу.

Конец. В регистре содержится контрольная сумма CRC32.

2.1.3. Обратный табличный алгоритм

На практике в протоколах Ethernet, ZigBee и архиваторах WinRaR используется обратный табличный алгоритм, который является некоторой

модификацией прямого. Основные отличия данного алгоритма от прямого заключаются в следующем (рисунок 2.3):

1) данные считываются с младших байтов файла;
2) значения предвычисленной таблицы и их положение зеркально отражаются.

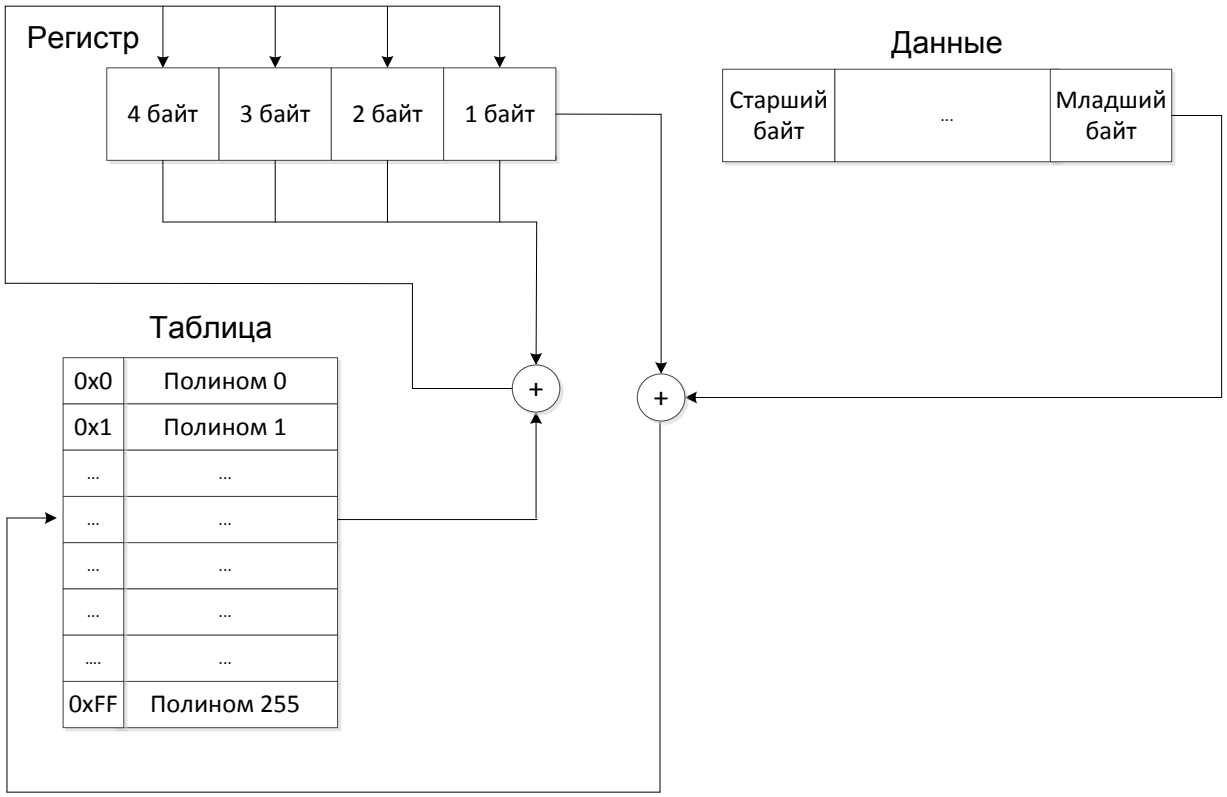


Рисунок 2.3 – Схема вычисления CRC по обратному табличному алгоритму

Результаты вычислений контрольной суммы CRC по прямому и табличному алгоритмам отличаются, поэтому для приёмника и передатчика помимо модели алгоритма CRC необходимо также указывать тип алгоритма.

Ввиду того, что табличный алгоритм требует некоторый объём памяти для хранения таблицы предвычисленных значений (1Кб для CRC32, 512 байт для CRC16, 256 байт для CRC8), а классический алгоритм обладает низким быстродействием из-за применения побитовых вычислений, предложен матричный алгоритм [58–62], который сочетает преимущества обоих алгоритмов и является масштабируемым для ускорения вычислений.

2.1.4. Матричный алгоритм

Из [43] известно, что при систематическом кодировании остаток от деления кодового слова *C(x)* на образующий полином *G(x*) является также произведением по модулю 2 кодового слова (вектора) и предвычисленной образующей матрицы *G*, т.е.

$$R(x) = C(x) \bmod G(x) \Leftrightarrow C(x) * G.$$

Контрольная сумма CRC относится к классу систематических полиномиальных циклических кодов, поэтому для вычисления CRC предложен *матричный алгоритм*, в котором можно применить операцию умножения вектора на матрицу как альтернативу табличному алгоритму. Таким образом, основная идея матричного алгоритма заключается в применении операции умножения по модулю два вектора (байт данных) на матрицу предвычисленных значений. При этом матрица для CRC вычисляется следующим образом:

$$G = \begin{vmatrix} x^k \bmod G(x) \\ x^{k+1} \bmod G(x) \\ \ldots\ldots\ldots\ldots \\ x^{k+m-1} \bmod G(x) \end{vmatrix}$$

где *m* – длина блока данных, обрабатываемого за итерацию алгоритма, *k* – разрядность контрольной суммы CRC. В полиномиальном виде формула матричного алгоритма для блока данных имеет следующий вид:

$$CRC(x) = (CRC'(x) \text{ xor } Data(x)) \cdot G$$

где *Data(x)* – полином, представляющий пакет данных, *CRC'(x)* – исходное значение CRC (*Init*).

В символьном виде:

$$CRC(0,1) = (CRC'(0,1) \text{ xor } Data(0,1)) \cdot G(0,1) \qquad (2.1)$$

где *Data(0,1)* – пакет данных в двоичном представлении, CRC*'(0,1)* – исходное значение CRC в двоичном представлении (*Init*).

Из выражения (2.1) следует, что теоретически матричный алгоритм позволяет рассчитать CRC для всего пакета данных за одну итерацию в отличие от табличного алгоритма, который вычисляет CRC побайтно. С практической точки зрения ограничениями для реализации матричного алгоритма являются доступные аппаратные ресурсы вычислительного устройства. Предложенная схема матричного алгоритма вычисления CRC приведена рисунке 2.4.

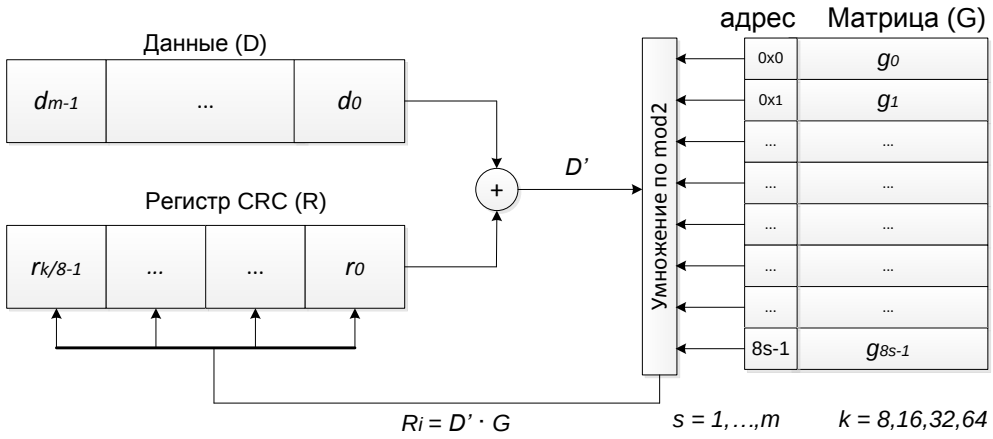


Рисунок 2.4 – Структурная схема матричного алгоритма CRC

Параметр k указывает разрядность контрольной суммы CRC. Параметр s показывает количество байт, обрабатываемых за итерацию. При $s = 1$ CRC вычисляется побайтно по аналогии с табличным алгоритмом.

Контрольная сумма CRC на i-ой итерации получается путём умножения вектора $D' = (d_0', d_1', \ldots, d_{m-1}') = R$ xor D на матрицу $G = (g_0, g_1, \ldots, g_{8s-1})$. Таким образом, CRC на i-ой итерации можно получить следующим образом:

$$R_i = D' \cdot G = d_0' \cdot g_0 \; xor \; d_1' \cdot g_1 \; xor \ldots . xor \; d_{8s-1}' \cdot g_{8s-1}.$$

На рисунке 2.5 приведена схема матричного алгоритма вычисления CRC8 для $s = 1$. Данные считываются побайтно, контрольная сумма на каждой итерации хранится в регистре CRC.

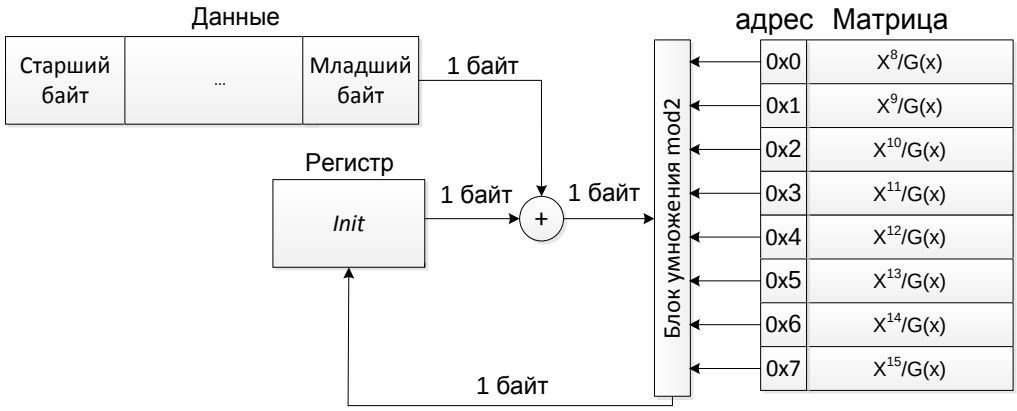


Рисунок 2.5 – Структурная схема матричного алгоритма вычисления CRC8

Для CRC32 матричный алгоритм будет отличаться матрицей и разрядностью регистра (рисунок 2.6).

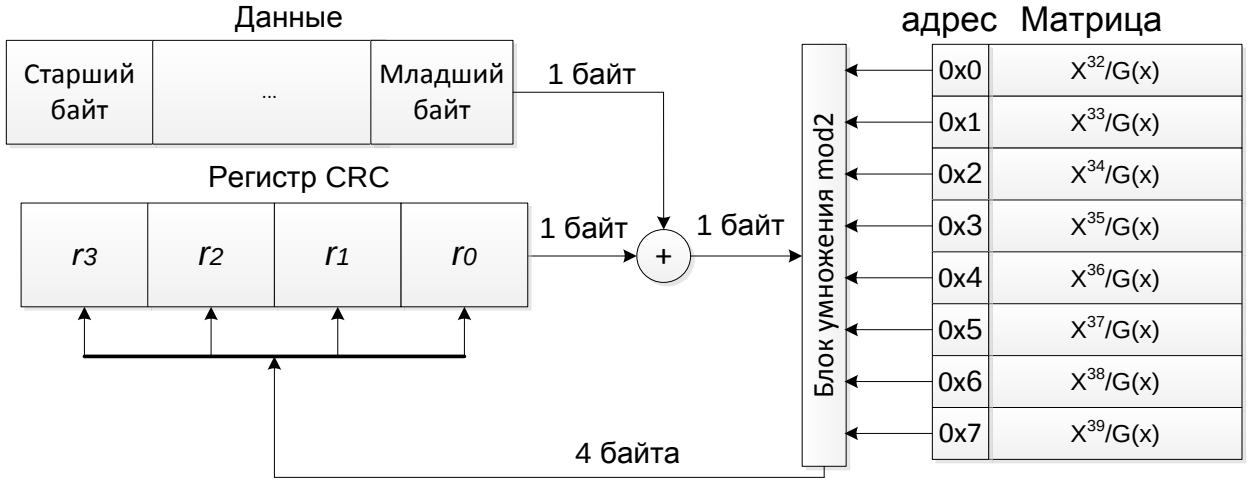


Рисунок 2.6 – Структурная схема матричного алгоритма вычисления CRC32 при $s = 1$

При s > 1 можно получить модификации матричного алгоритма с расширенной матрицей предвычисленных значений. Увеличение количества байт, обрабатываемых за итерацию, позволяет значительно повысить быстродействие операции вычисления CRC.

На рисунке 2.7 приведена схема алгоритма вычисления CRC32 с двухбайтовым сдвигом данных. Размер предвычисленной матрицы увеличивается с 8 до 16 элементов. Два байта данных складываются по модулю 2 с двумя байтами из регистра CRC и результат сложения умножается на матрицу по аналогии с однобайтовым сдвигом.

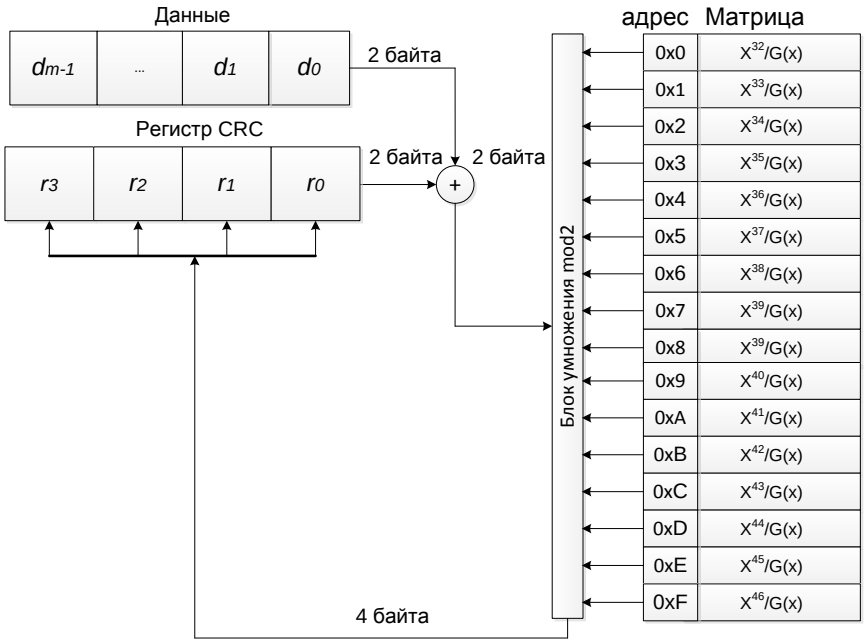


Рисунок 2.7 – Структурная схема матричного алгоритма CRC32 при $s = 2$

На рисунке 2.8. приведена схема матричного алгоритма CRC32 при $s = 4$. Блок данных размером 4 байта складывается по модулю 2 с регистром CRC32 и результат умножается на матрицу, расширенную до 32-х элементов.

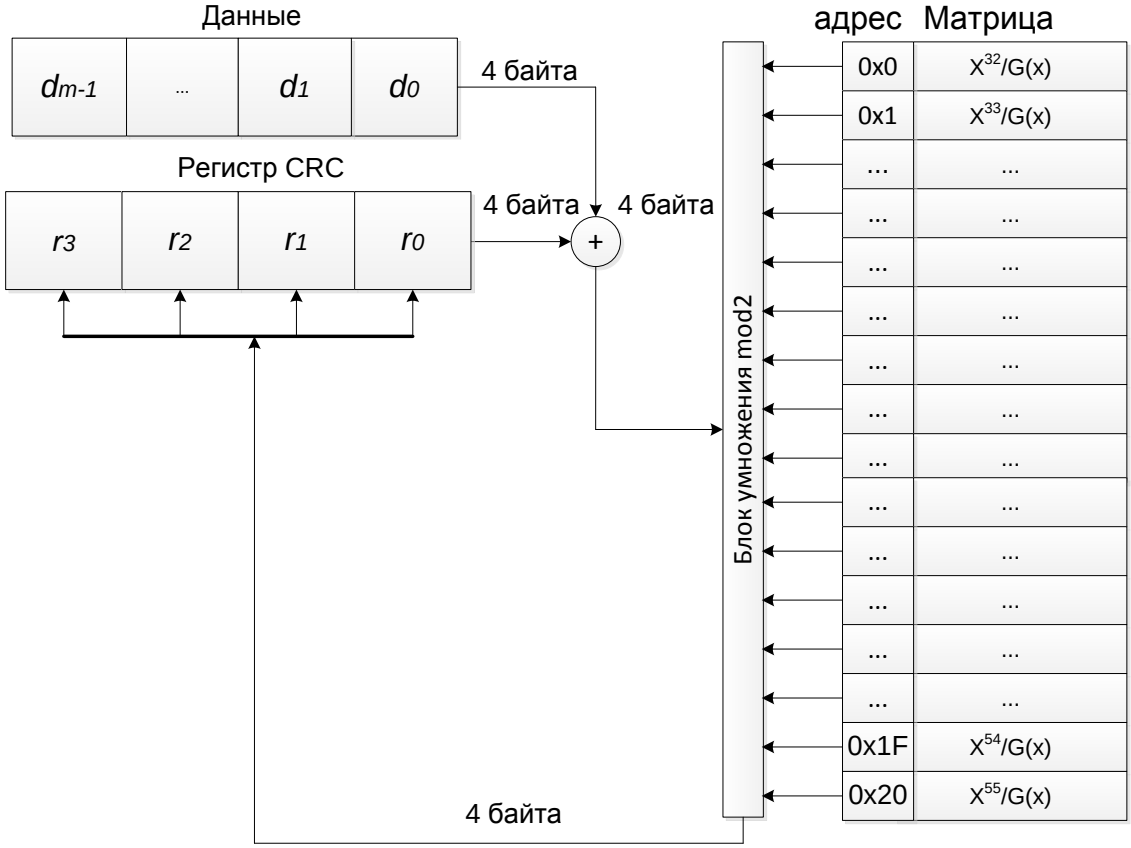


Рисунок 2.8 – Структурная схема матричного алгоритма CRC32 при $s = 4$

В случае, если размер данных не кратен длине обрабатываемого блока, оставшийся на последней итерации блок дополняется старшими нулями.

При реализации матричного алгоритма объём памяти для хранения матрицы будет равен: $V = 8 \cdot k\ s$ бит, где k – разрядность контрольной суммы CRC, s – количество байт, обрабатываемых за одну итерацию (сдвиг). Очевидно, что можно увеличивать размер блока, обрабатываемого за итерацию, до тех пор, пока аппаратные ресурсы вычислительного устройства это позволяют. В таблице 2.1 приведены размеры матрицы в байтах при различных модификациях (сдвигах) матричного алгоритма CRC.

Таблица 2.1 – Сравнение размеров матрицы и таблицы (в байтах)

Контрольная сумма	Сдвиг (s)					Табличный
	1 байт	2 байта	4 байта	32 байта	1024 байта	
CRC8	8	16	32	256	8192	**256**
CRC16	16	32	64	512	16384	**512**
CRC32	32	64	128	1024	32768	**1024**
CRC64	64	128	256	2048	65536	**2048**

Из таблицы 2.1. видно, что при количестве сдвигов $s = 32$ объём памяти для хранения значений матрицы равен объёму памяти для хранения значений таблицы табличного алгоритма. По аналогии с матричным алгоритмом для табличного алгоритма можно увеличить количество байт, обрабатываемых за итерацию. Однако объём таблицы (в битах) будет расти согласно выражению: $V_T = 2^{8s} \cdot k$, что при реализации потребует достаточно большой объём памяти (таблица 2.2).

Таблица 2.2 – Размер таблицы (в байтах) для табличного алгоритма

Контрольная сумма	Сдвиг (s)		
	1 байт	2 байта	4 байта
CRC8	2^8	2^{16}	2^{32}
CRC16	2^9	2^{17}	2^{33}
CRC32	2^{10}	2^{18}	2^{34}
CRC64	2^{11}	2^{19}	2^{35}

Таким образом, матричный алгоритм обладает преимуществом по сравнению с табличным алгоритмом по требуемому объёму памяти для хранения предвычисленных значений, при этом является простым в реализации и не требует интенсивных вычислений.

Для исследования быстродействия различных вариантов матричного алгоритма необходимо провести компьютерный эксперимент по вычислению контрольной суммы CRC для различных наборов данных. На разработанную программу вычисления контрольной суммы CRC32 матричным алгоритмом получено свидетельство о регистрации ПО для ЭВМ [58], копия которого представлена в приложении Г.

2.2. Исследование быстродействия алгоритмов вычисления CRC

Эксперимент по вычислению проводился с контрольной суммой CRC32, как наиболее распространённой при контроле целостности данных в известных протоколах и стандартах Ethernet, WinRaR, ZigBee, PNG, MPEG-2.

2.2.1. Постановка задачи исследования

Для постановки компьютерного эксперимента по определению быстродействия различных алгоритмов вычисления контрольной суммы CRC32 использовался один из узлов суперкомпьютерного кластера «СКИФ-политех» с процессором Intel XEON 5150, 2.66 ГГц и оперативной памятью объёмом 8 Гб. Оценка времени вычисления контрольной суммы производилась путем выполнения 50 запусков для файлов объёмом от 10 до 1010 мегабайт с шагом 200 мегабайт. На основе данных, полученных по запускам каждого алгоритма, составлены таблицы средних значений времени расчёта контрольной суммы с доверительными интервалами и размерами исполняемых файлов.

Заметим, что для больших файлов необходимо вычислять CRC по частям, помещая блоки файла в буфер заданного размера. В связи с этим, исследование проводилось в несколько этапов: с буферами в 1 Мб, в 4 байта и в 1 байт – для моделирования вычисления контрольной суммы в системах с малым доступным объёмом памяти.

2.2.2. Компьютерный эксперимент по вычислению CRC32

*Первый эксперимент для буфера 1Мб***.**

В исходном коде программной реализации CRC32 был установлен буфер размером 1 Мб.

В таблице 2.3 приведены времена расчёта CRC32 для матричного и табличного алгоритмов.

Таблица 2.3 – Время расчёта CRC32 и доверительные интервалы для буфера 1 Мб

Алгоритм	Время расчёта CRC32, с (Доверительные интервалы, %)					
	Объём использованных файлов, Мб					
	10	210	410	610	810	1010
Табличный	0,057 (±1,4)	1,194 (±0,217)	2,299 (±0,203)	3,227 (±0,162)	4,334 (±0,211)	6,184 (±0,155)
МС 1 байт	0,125 (±0,732)	2,81 (±0,112)	5,399 (±0,067)	7,602 (±0,166)	10,23 (±0,161)	14,557 (±0,048)
МС 2 байта	0,092 (±3,882)	2,084 (±0,133)	4,02 (±0,09)	5,645 (±0,105)	7,566 (±0,097)	10,85 (±0,072)
МС 4 байта	0,08 (±0,827)	1,729 (±0,172)	3,32 (±0,011)	4,694 (±0,314)	6,272 (±0,188)	8,965 (±0,064)

Размеры исполняемых файлов табличной и матричных программных реализаций приведены в таблице 2.4.

Таблица 2.4 – Размеры исполняемых файлов, байт

Алгоритм	Табличный	МС 1 байт	МС 2 байта	МС 4 байта
Размер исполняемого файла, байт	15 490	11 729	12 247	12 994

Для оценки прироста быстродействия при увеличении количества байт, обрабатываемых за итерацию, вычислена разница данных из табл. 2.3 для матричных алгоритмов с 2-х и 4-х байтными сдвигами относительно однобайтового (таблица 2.5).

Таблица 2.5 – Ускорение относительно однобайтового сдвига для буфера 1 Мб

Алгоритм	Ускорение относительно матричного алгоритма со сдвигом 1 байт, %						Среднее ускорение, %
	Объём использованных файлов, Мб						
	10	210	410	610	810	1010	
МС 2 байта	26,4	25,8	25,5	25,7	26	25,5	25,8
МС 4 байта	36	37,9	38,5	38,2	38,6	38,4	37,9

В среднем матричный 4-х байтовый алгоритм вычисляет CRC32 быстрее базового (однобайтового) приблизительно на 38 %. Однако из таблицы 2.3. следует, что быстродействие матричного алгоритма со сдвигом 4 байта меньше, чем быстродействие табличного. Для более точного анализа быстродействия составлена таблица 2.6, показывающая отставание матричных алгоритмов от табличного.

Таблица 2.6 – Отставание матричного алгоритма относительно табличного для буфера 1 Мб

Алгоритм	Отставание относительно табличного алгоритма, %						Среднее отставание, %
	Объём использованных файлов, Мб						
	10	210	410	610	810	1010	
МС 1 байт	119,2	135,3	134,8	135,5	136,5	135,3	132,8
МС 2 байта	61,4	74,5	74,8	74,9	74,5	75,2	72,5
МС 4 байта	40,3	44,8	44,4	45,4	44,7	44,9	44,1

В среднем матричный однобайтовый алгоритм отстает по быстродействию от табличного приблизительно на 133%. Увеличение количества байт, обрабатываемых за итерацию (до 2-х), позволяет сократить отставание приблизительно до 73%, а при увеличении до 4-х отставание сокращается приблизительно до 44%.

Второй эксперимент для буфера 4 байта

Для буфера размером 4 байта результаты исследования приведены в таблицах 2.7 – 2.9.

Таблица 2.7 – Время расчёта CRC32 и доверительные интервалы для буфера 4 байта

Алгоритм	Время расчёта CRC32, с (Доверительные интервалы, %)					
	Объём использованных файлов, Мб					
	10	210	410	610	810	1010
Табличный	0,118 (±2,069)	2,341 (±1,655)	4,477 (±1,177)	6,592 (±0,436)	8,898 (±1,781)	10,885 (±0,608)
МС 1 байт	0,191 (±0,91)	3,797 (±0,622)	7,319 (±0,490)	10,775 (±0,519)	14,282 (±0,369)	17,828 (±0,439)
МС 2 байта	0,142 (±1,147)	2,793 (±0,335)	5,457 (±0,533)	8,83 (±0,296)	10,709 (±0,262)	13,396 (±0,430)
МС 4 байта	0,103 (±1,524)	2,038 (±0,473)	3,931 (±0,677)	5,813 (±0,406)	7,786 (±0,528)	9,671 (±0,440)

Таблица 2.8 – Отставание относительно однобайтового сдвига для буфера 4 байта

Алгоритм	Отставание относительно матричного алгоритма со сдвигом 1 байт, %						Среднее отставание, %
	Объём использованных файлов, Мб						
	10	210	410	610	810	1010	
МС 2 байта	25,65	26,45	25,45	24,98	25,01	24,86	25,4
МС 4 байта	46,07	46,33	46,29	46,05	45,48	45,75	45,99

Таблица 2.9 – Отставание матричного алгоритма относительно табличного для буфера 4 байта

Алгоритм	Отставание относительно табличного алгоритма, %						Среднее отставание, %
	Объём использованных файлов, Мб						
	10	210	410	610	810	1010	
МС 1 байт	60,99	62,15	63,47	63,44	60,5	63,78	62,38
МС 2 байта	19,68	19,28	21,88	22,61	20,34	23,06	21,14
МС 4 байта	-13,18	-12,96	-12,19	-11,82	-12,5	-11,15	-12,3

Таким образом, в данном случае ускорение двухбайтового и четырехбайтового матричных алгоритмов при уменьшении размера буфера до 4 байт достигает в среднем 25,40% и 45,99% соответственно. При исследовании скорости расчета матричных алгоритмов относительно табличного выяснилось, что для однобайтового алгоритма отставание сократилось в среднем до 62,38 % в то время, как отставание по скорости двухбайтового матричного алгоритма от табличного уменьшилось до 21,14%. Матричный четырехбайтовый алгоритм стал более быстродействующим относительно табличного в среднем на 12,3%.

Третий эксперимент для буфера 1 байт

Для буфера размером 1 байт исследованы только табличный и матричный однобайтовый алгоритмы, так как обработка по 2 и 4 байта за итерацию в данном случае невозможна. Результаты сравнения приведены в таблицах 2.10 – 2.11.

Таблица 2.10 – Время расчёта CRC32 и доверительные интервалы для буфера 1 б

Алгоритм	Время расчёта CRC32, с (Доверительные интервалы, %)					
	Объём использованных файлов, Мб					
	10	210	410	610	810	1010
Табличный	0,315 (±2,526)	6,044 (±0,598)	11,797 (±1,36)	17,452 (±1,082)	23,152 (±0,985)	28,767 (±0,896)
МС 1 байт	0,361 (±0,637)	7,152 (±0,630)	13,958 (±0,621)	20,610 (±0,6483)	27,558 (±0,5331)	34,129 (±0,523)

Таблица 2.11 – Отставание матричного алгоритма относительно табличного для буфера 1 б

Алгоритм	Отставание относительно табличного алгоритма, %						Среднее отставание, %
	Объём использованных файлов, Мб						
	10	210	410	610	810	1010	
МС 1 байт	14,60	18,33	18,31	18,09	19,03	18,63	18

При исследовании реализаций с буфером 1 байт, что можно приравнять к вычислениям без буфера при побайтовой передаче данных, было установлено, что отставание матричного однобайтового алгоритма относительно таблично сократилось до 18%. Это связано с тем, что при минимальном размере буфера

(1 байт) табличный алгоритм значительно теряет в скорости относительно варианта с буфером 4 байт.

2.2.3. Анализ результатов компьютерного эксперимента

Компьютерный эксперимент по определению быстродействия алгоритмов вычисления контрольной суммы CRC показал, что изменения размера буфера, установленного в программной реализации, существенно влияет на скорость расчёта CRC [59, 61]. При уменьшении размера буфера с 1 Мб до 1 байта быстродействие табличного и матричного алгоритмов падает, однако потери скорости вычислений для табличного алгоритма намного значительней, чем для матричных алгоритмов (рисунок 2.9). Поэтому для буфера размером 4 байта 4-х байтовый матричный алгоритм является более быстродействующим, чем табличный.

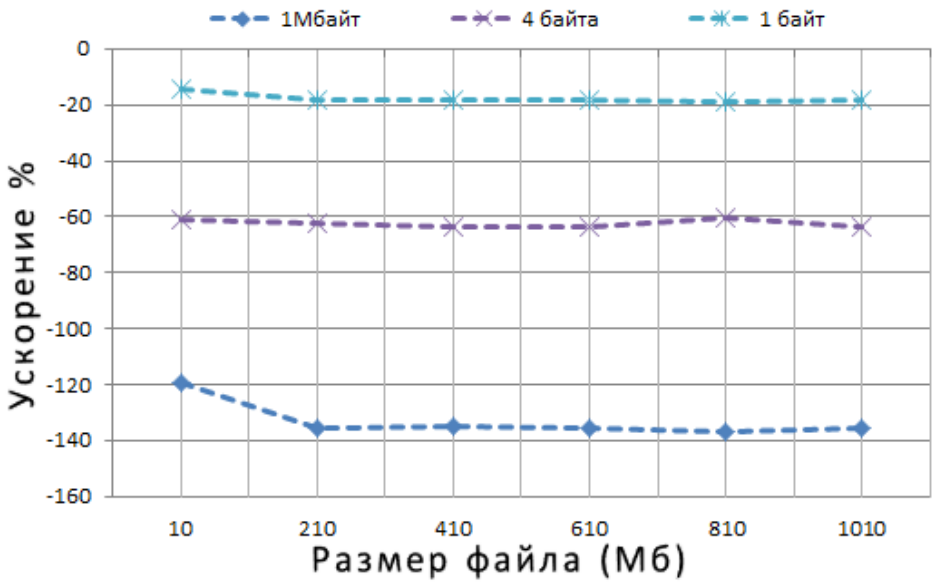


Рисунок 2.9 – Ускорение (отставание) матричного алгоритма со сдвигом 1 байт

На рисунках 2.10, 2.11 приведены графики ускорений (значения со знаком минус интерпретируются как отставание) матричных алгоритмов с различными сдвигами относительно табличного алгоритма с разными размерами буфера.

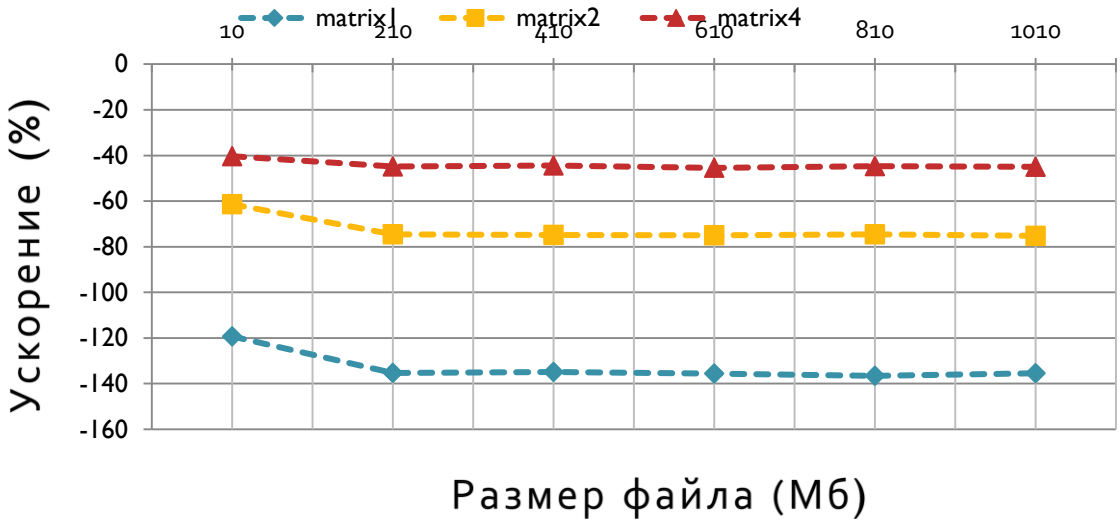


Рисунок 2.10 – Ускорение (отставание) матричных алгоритмов с буфером 1 Мб

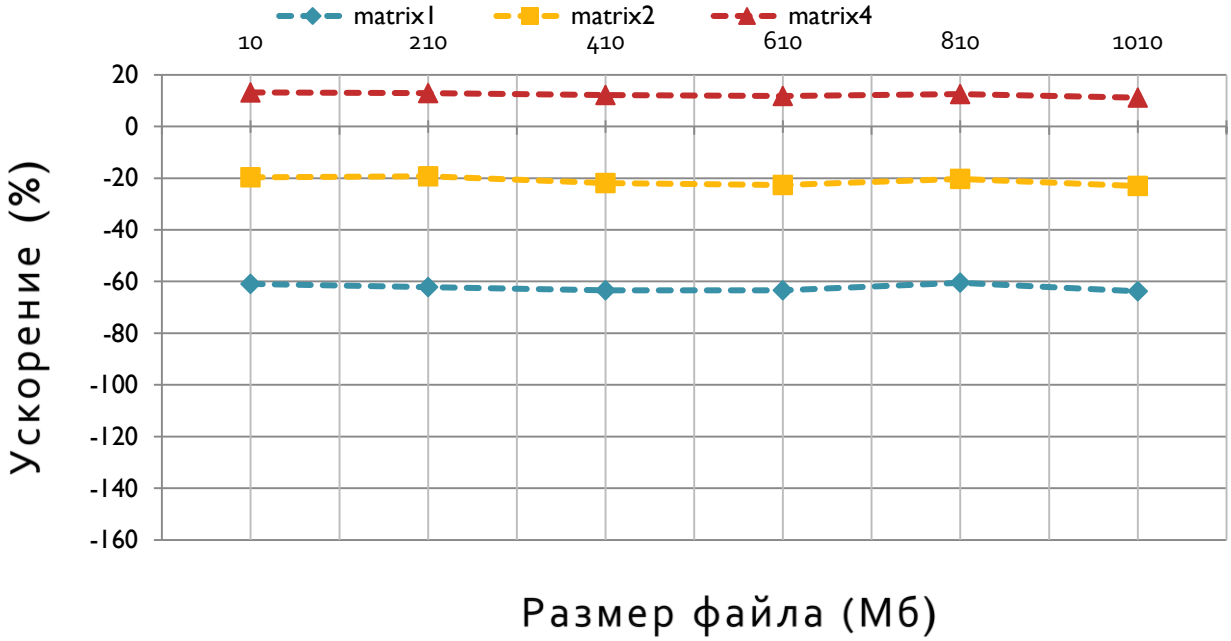


Рисунок 2.11 – Ускорение (отставание) матричных алгоритмов с буфером 4 байта

Таким образом, однобайтовый матричный алгоритм наиболее эффективен (относительно табличного алгоритма) при установленном буфере 1 байт. Его отставание составляет приблизительно 18%. Матричный 4-х байтовый алгоритм эффективнее по быстродействию табличного приблизительно на 12% при размере буфера равного 4 байта и при этом требует для хранения матрицы в 8 раз меньше объема памяти.

Проведённые эксперименты моделируют вычисления контрольной суммы CRC для 32-разрядных микроконтроллеров, в которых доступный объём памяти для реализации алгоритмов контроля целостности данных значительно ограничен. Однако моделирование не всегда может отражать действительную ситуацию и в данном случае даёт только оценку алгоритмам по быстродействию и требуемым ресурсам для реализации. Для исследования быстродействия алгоритмов вычисления CRC на реальных микроконтроллерах (МК) поставлен эксперимент по реализации функции расчёта CRC для микропроцессорной системы измерения температуры на основе цифрового датчика DS18B20 и 8-разрядного микроконтроллера ATtiny44.

2.3. Исследование программных реализаций алгоритмов вычисления CRC для микропроцессорного устройства

2.3.1. Постановка задачи исследования

В системах промышленной автоматизации, автоматизированных системах управления широко применяются микропроцессорные системы для сбора и первичной обработки данных с датчиков, выработки управляющих сигналов на исполнительные устройства (нижний уровень системы), коммутации сигналов и различных интерфейсов между собой, а также выдачи информации на верхний уровень системы в едином формате (средний уровень системы). Микропроцессорные системы нижнего уровня, как правило, решают задачи контроля параметров технологических процессов и непосредственного управления оборудованием, а также реализуют такие возможности современных автоматизированных систем управления, как автоматический пуск и остановка оборудования с целью предотвращения аварийных ситуаций [63]. Непосредственный контроль производственных процессов и их параметров осуществляется системой датчиков. Сигналы от датчиков поступают на соответствующие микропроцессоры (микроконтроллеры), в которых происходит сравнение параметров сигнала датчика с пороговыми значениями. Нижний уровень системы может включать множество недорогих

микропроцессорных (микроконтроллерных) решений с дефицитом ресурсов – небольшой объём памяти программ и данных, низкая производительность. Как правило, возлагаемые на такие решения задачи требуют для реализации весь объём памяти (EEPROM –энергонезависимая перепрограммируемая память, PROM – энергонезависимая FLASH память, RAM – энергозависимая оперативная память, регистры общего назначения (РОН), память программ) и процессорное время. Помимо решения основных задач необходимо также обеспечивать приём, подготовку и отправку данных в различные модули микропроцессорной системы, при этом необходимо обеспечивать защиту передаваемых и обрабатываемых данных от ошибок с помощью кодирования информации.

Одним из примеров нижнего уровня системы на основе микроконтроллеров с дефицитом ресурсов является специализированный инкубатор с многоточечным контролем и регулированием температуры по заданному алгоритму. В данной системе основные ресурсы МК задействованы для реализации алгоритмов регулирования температуры и важных для системы вычислений, при этом ресурсы на решение дополнительных задач, таких как приём, отправка и реализация алгоритмов кодирования данных с применением контрольных сумм CRC значительно ограничены [64].

2.3.2. Описание системы измерения температуры

Микропроцессорная система измерения температуры [65] построена на основе цифрового однопроводного (1-wire) датчика DS18B20 фирмы DALLAS [66] и 8-разрядного микроконтроллера ATtiny44 от Atmel [67] с 32-мя РОН (рисунок 2.12).

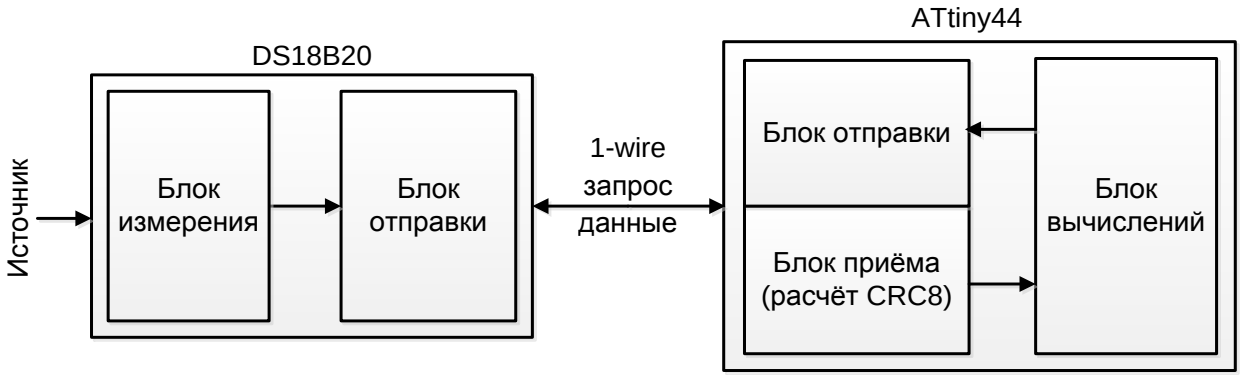


Рисунок 2.12 – Структурная схема обмена данными между микроконтроллером и датчиком

Ведущее устройство – МК инициирует запрос на передачу данных с датчика путём отправки на него 64-х разрядного кода устройства, включающего контрольную сумму CRC8 (рисунок 2.13). Датчик в ответ на запрос отправляет пакет данных, содержащий значение температуры, граничные контрольные значения, регистр конфигурации и вычисленную контрольную сумму CRC8.

Рисунок 2.13 – Формат пакета данных DS18B20

Вычисление контрольной суммы CRC8 осуществляется согласно параметрической модели [65], представленной в таблице 2.12.

Таблица 2.12 – Параметрическая модель CRC8

Параметр	**Значение**
Name	CRC 8
Width	8
Poly	0x131
Init	0x00
RefIn	False
RefOut	False
XorOut	0x00

В протоколе передачи данных датчика DS18B20 применяется контрольная сумма CRC8 на основе образующего полинома (Poly) $X^8+ X^5+ X^4+1$ с нулевым инициализирующим значением (Init), нулевой константой (XorOut), без отражения байт входных и выходных данных (RefIn, RefOut) [65]. На рисунке 2.14 представлена схема вычисления CRC кода, применяемая в датчике DS18B20.

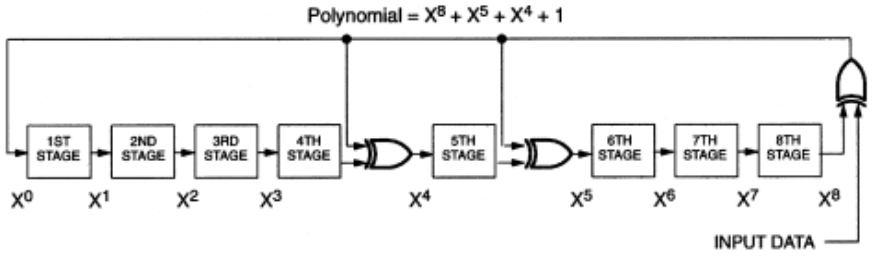


Рисунок 2.14 – Вычисление CRC8 для датчика DS18B20

Контрольная сумма по схеме на рисунке 2.14 вычисляется побитно. Логические элементы «Исключающее ИЛИ» расставлены между регистрами согласно степеням образующего полинома.

2.3.3. Программная реализация алгоритмов вычисления CRC8

Для выбора рекомендации по применению алгоритмов вычисления CRC8 в микропроцессорных (микроконтроллерных) системах исследованы различные варианты программной реализации для микроконтроллера ATtiny44 на языке Assembler. Всего в ходе эксперимента реализовано 2 варианта классического, один вариант табличного и 5 вариантов матричного алгоритмов. Для табличного алгоритма таблица предвычисленных значений взята из технической документации к датчику температуры DS18B20 [68]. Для

матричного алгоритма матрица рассчитана на основе образующего полинома $X^8+X^5+X^4+1$ и представлена в 16-ричной системе счисления (hex):

$$G = \left| \begin{array}{l} X^8 mod\ X^8+X^5+X^4+1 \\ X^9 mod\ X^8+X^5+X^4+1 \\ X^{10} mod\ X^8+X^5+X^4+1 \\ X^{11} mod\ X^8+X^5+X^4+1 \\ X^{12} mod\ X^8+X^5+X^4+1 \\ X^{13} mod\ X^8+X^5+X^4+1 \\ X^{14} mod\ X^8+X^5+X^4+1 \\ X^{15} mod\ X^8+X^5+X^4+1 \end{array} \right| => \left| \begin{array}{l} 31 \\ 62 \\ C4 \\ B9 \\ 43 \\ 86 \\ 3D \\ 7A \end{array} \right|$$

На рисунке А.1 Приложения А приведена схема программной реализации классического алгоритма для микроконтроллера ATtiny44. Реализация заключается в последовательном сдвиге бит данных и выполнение операций XOR в зависимости от значения выдвинутого бита во флаге переноса, а также младшего бита промежуточного результата вычисления CRC. Для программной реализации данного варианта алгоритма размер кода составляет 35 слов при времени выполнения 5516 тактов с использованием пяти регистров.

На рисунке А.2 Приложения А приведена схема реализации табличного алгоритма вычисления CRC. При реализации табличного алгоритма размер кода составляет 16 слов, при этом требуется объём памяти PROM 128 слов (256 байт). Время выполнения составляет 116 тактов МК, при этом используются 8 РОН.

На рисунке А.3 Приложения А приведена схема реализации модифицированного (второй вариант) классического алгоритма. Размер кода составляет 24 слова, время выполнения 858 тактов МК, используются 6 РОН.

Для достижения компромисса между табличной и классической реализациями алгоритмов вычисления CRC по требованиям к объёму памяти и количеству тактов МК реализованы различные модификации матричного алгоритма вычисления CRC8. На рисунке А.4 Приложения А приведена схема

реализации матричного алгоритма вычисления CRC с применением памяти EEPROM. В этой реализации байт данных суммируется по модулю 2 с регистром CRC (начальное значение – 0x00). Полученный результат логически сдвигается влево на 1 бит и в зависимости от значения выдвинутого бита осуществляется операция XOR элементов матрицы из EEPROM с регистром CRC (результат сохраняется в регистр CRC). Операции повторяются, пока все байты данных не будут обработаны. Размер кода для данной реализации составляет 23 слова + 8 байт EEPROM, время выполнения 602–634 такта МК (зависит от количества единиц в результате сложения байта входных данных и регистра CRC), при этом используются 6 РОН.

На рисунке А.5 Приложения А представлена похожая реализация матричного алгоритма вычисления CRC с применением оперативной памяти (RAM). Размер кода составляет 22 слова, а также требуется 8 байт RAM. Время выполнения 545 тактов МК, используются 10 РОН. Для аналогичной реализации с применением FLASH памяти PROM время выполнения составит 603 такта МК, а размер кода – 24 слова при использовании 10 РОН.

На рисунке А.6 Приложения А представлена схема реализации матричного алгоритма вычисления CRC с использованием РОН без применения запоминающих устройств. Идея реализации матричного алгоритма вычисления CRC на РОН заключается в выполнении операции XOR между РОН и регистром CRC. Эта операция производится в зависимости от результата сложения байта данных с контрольной суммой, полученной на предыдущей итерации (начальное значение 0x00). Размер кода в данном случае увеличивается до 50 слов, однако время выполнения уменьшается до 244 тактов МК при использовании 13 РОН.

На рисунке А.7 Приложения А представлена аналогичная реализация алгоритма с матрицей в коде программы. Размер кода для данной реализации составляет 42 слова, время выполнения 250–254 такта МК при использовании шести РОН.

2.3.4. Анализ результатов эксперимента

Результаты вычисления контрольной суммы CRC8 для реализаций, полученных для микроконтроллера ATtiny44 в среде AVR Studio 4.18, для блока данных размером 64 бита представлены в таблице 2.13.

Таблица 2.13 – Сравнение реализаций алгоритмов вычисления CRC8 на ATtiny44

Алгоритм	Время, такты	Память программ, слова	Внеш. память, слова	РОН, кол-во
Классический	5516	35	-	5
Классический (модифицированный)	858	24	-	6
Табличный	116	16	128	7
Матричный (EEPROM)	602–634	23	4	6
Матричный (PROM)	603	24	4	10
Матричный (RAM)	545	22	4	10
Матричный (РОН)	244	50	-	13
Матричный (память программ)	254	42	-	6

Представленные в таблице 2.13 результаты позволяют выбрать наиболее подходящую реализацию алгоритма, исходя из доступных ресурсов и требований к быстродействию. Преимуществом табличного алгоритма является его быстродействие (116 тактов МК) и простота реализации, однако существенным недостатком является применение таблицы предвычисленных значений, для хранения которой требуется 128 слов внешней памяти при реализации алгоритма CRC8 (256 слов при реализации алгоритма CRC16 и 512 слов при реализации алгоритма CRC32). Наименьшие затраты внешней памяти и памяти программ требуются для реализации классического модифицированного алгоритма (24 слова), но при времени выполнения 858 тактов МК, что примерно в 7 раз медленнее табличного алгоритма.

Модификации матричного алгоритма являются компромиссом между быстродействием вычисления CRC и требуемым объёмом памяти. При доступном объёме памяти программ и данных до 42–50 слов наиболее

подходящей может быть реализация матричного алгоритма с матрицей в РОН со временем выполнения 244 такта МК или с матрицей в памяти программ, где время выполнения – 254 такта МК. Это примерно в 2 раза медленнее табличного алгоритма и в 3,5 раза быстрее классического модифицированного.

При значительном ограничении памяти программ для модуля расчёта CRC можно применить реализацию с матрицей в RAM. Тогда потребуется объём памяти программ на два слова меньше, чем для классического оптимизированного варианта (22 против 24), но при этом требуется объём внешней памяти 8 байт. Количество тактов МК уменьшится на 36% (с 858 до 545).

Таким образом, в результате проведения эксперимента на микроконтроллере установлено, что в случае наличия свободного объёма памяти под модуль расчёта CRC от 144 слов целесообразно применять табличную реализацию. Однако в случаях, когда для систем измерения на основе микроконтроллеров основные ресурсы задействованы на выполнения основных задач системы и ресурсы для реализации дополнительных модулей контроля целостности значительно ограничены, целесообразно применение реализаций с матрицей в RAM или в РОН.

Приведённые исследования, ориентированные на программную реализацию алгоритмов вычисления контрольной суммы CRC, позволяют оценить быстродействие и требуемые ресурсы различных алгоритмов обнаружения ошибок в процессе передачи или хранения данных. Во многих системах передачи данных помимо микроконтроллеров также применяются ПЛИС для приёма/отправки пакетов данных. Для оценки быстродействия и аппаратных затрат (логических элементов, памяти) алгоритмов расчёта CRC в таких системах необходима разработка устройств вычисления CRC на ПЛИС.

2.4. Разработка устройств вычисления CRC на ПЛИС

2.4.1. Постановка задачи

В системах автоматизации производства для управления технологическими объектами, синхронизации и противоаварийной защиты помимо микроконтроллеров также применяют ПЛИС [63, 69]. Для обеспечения целостности данных и команд в таких системах необходима аппаратная реализация на ПЛИС [70–82] модулей обнаружения и исправления ошибок с применением языков описания аппаратуры. При этом быстродействие и аппаратные затраты могут отличаться от реализации аналогичных алгоритмов на микроконтроллерах.

Для исследования быстродействия и аппаратных затрат устройств вычисления CRC на ПЛИС рассмотрена реализация алгоритмов вычисления CRC8 и CRC32 на языке описания аппаратуры Verilog [83] с применением САПР Quartus II и ПЛИС Cyclone III EP3C5E144C8 от Altera. При исследовании реализаций CRC8 размер пакета задан 8 байт, что соответствует размеру пакета данных для системы измерения температуры (п. 2.3). Для исследования алгоритмов CRC32 размер пакета данных задан равным 64 байта, что соответствует минимальной длине пакета протокола Ethernet (без учета поля CRC32). Табличный алгоритм реализован согласно параметрической модели для протокола Ethernet (обратный вариант). Матричный алгоритм реализован в 4 вариантах комбинационных схем с обработкой по 1,2,4,8 и 64 байта (для CRC32) за итерацию.

2.4.2. Разработка устройств вычисления CRC8 на ПЛИС

Применение контрольной суммы CRC8 может являться менее затратной и более быстродействующей альтернативой декодерам, исправляющим ошибки в устройствах приёма и отправки пакетов данных в случаях, когда система и канал связи позволяют осуществлять многократную повторную передачу пакета, т.е. в системах с обратной связью от приёмника.

Табличный алгоритм позволяет быстро вычислять контрольную сумму CRC на устройствах на основе ПЛИС или ASIC [81, 82], однако требует для хранения предвычисленных констант $V = 256 \cdot n$ бит памяти, где n – разрядность контрольной суммы. Устройство вычисления CRC8 с предвычисленными константами в постоянной памяти состоит из буфера приёма данных, комбинационной схемы вычисления адреса и ROM (рисунок 2.15).

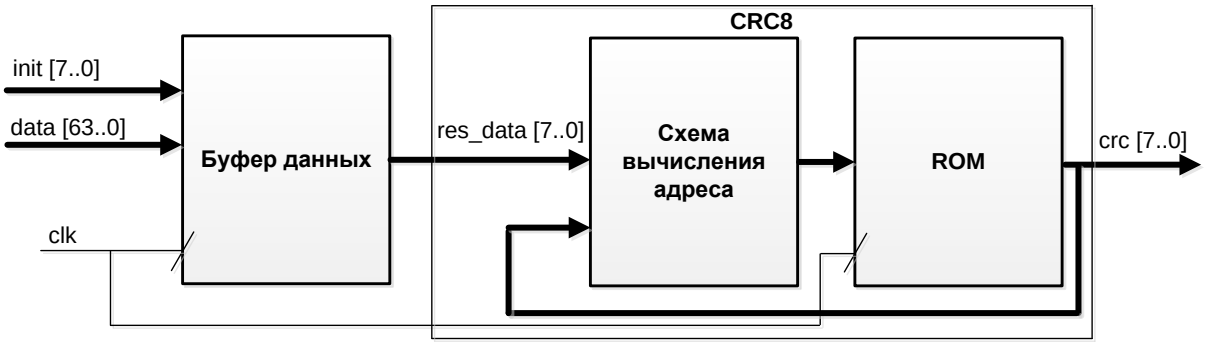


Рисунок 2.15 – Структурная схема устройства вычисления CRC8 по табличному алгоритму с памятью ROM

По фронту тактового сигнала *clk* из памяти по адресу, сформированному в схеме вычисления адреса с применением байта данных *res_data* и значения CRC8 на предыдущем шаге, выбирается значение CRC из таблицы. Процесс повторяется, пока все байты пакета данных не будут обработаны. На k-м такте, где k – длина пакета в байтах, на выходе устройства будет сформирована CRC8. Недостатком такой реализации является сложность масштабирования, т.к. увеличение длины пакета данных, обрабатываемого за такт, приводит к значительному увеличению требуемого объёма памяти для хранения констант. Для устранения такого недостатка следует применять реализацию на основе матричного алгоритма, который позволяет строить асинхронную комбинационную схему вычисления CRC без применения памяти ROM.

На рисунке 2.16 приведена структурная схема устройства вычисления CRC8 на ПЛИС с асинхронным блоком вычисления в виде комбинационной схемы. Значение m соответствует количеству бит, обрабатываемых за одну итерацию матричного алгоритма. Вычисленная контрольная сумма CRC8 записывается в выходной регистр CRC по тактовому сигналу генератора.

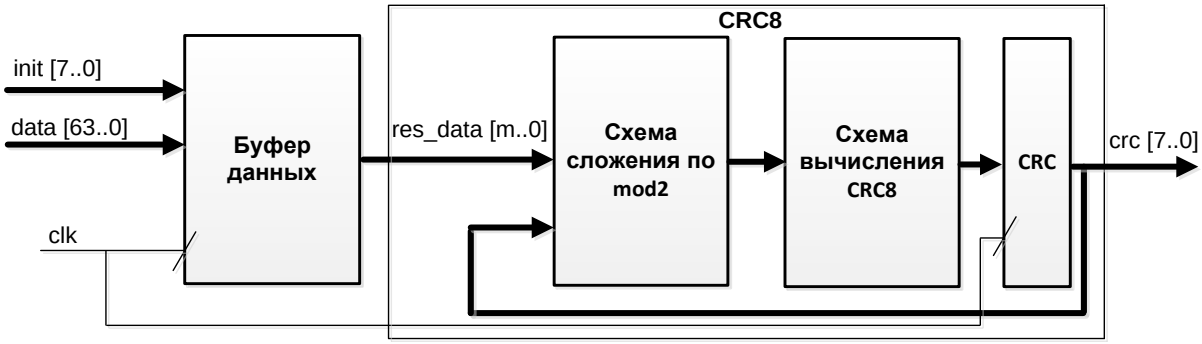


Рисунок 2.16 – Общая структурная схема устройства вычисления CRC8 по матричному алгоритму

Процесс вычисления является итеративным и значение CRC каждого последующего байта определяется значением CRC предыдущего, что приводит к построению устройства с обратной связью. Для каждого байта данных вычисление CRC осуществляется за один такт генератора импульсов, максимальная частота сигнала которого зависит от быстродействия блока приёма (буфера) данных и вычисления CRC. Отличительной особенностью матричного алгоритма является возможность вычислять CRC за одну итерацию (такт) для блока данных, больше чем один байт. Таким образом, можно сформировать матрицу, позволяющую обработать весь пакет данных за один такт и построить устройство без обратной связи (рисунок 2.17).

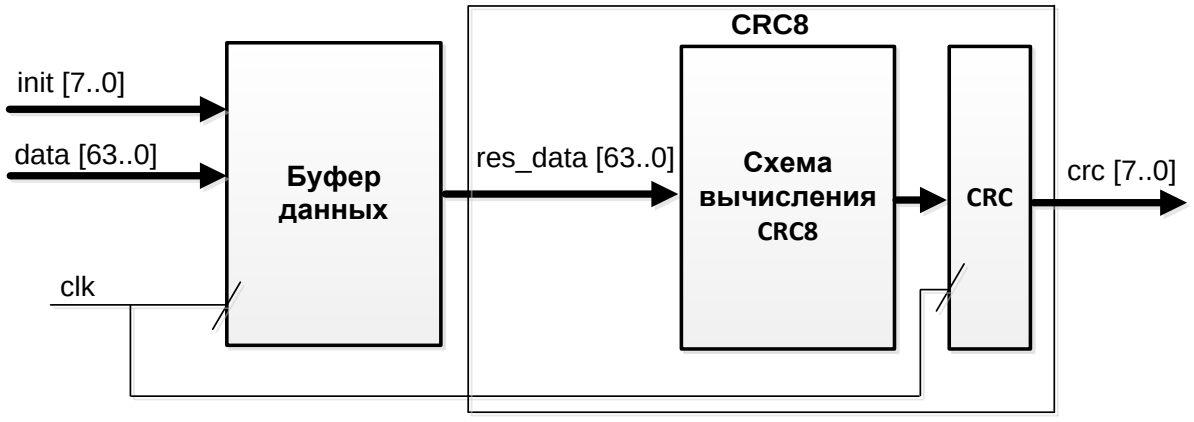


Рисунок 2.17 – Структурная схема устройства вычисления CRC8 по матричному алгоритму с обработкой по 64 бита

Быстродействие такого устройства будет значительно выше, чем при побайтовой обработке пакета данных, однако для реализации требуется больше логических элементов ПЛИС.

На рисунке 2.18 представлена схема устройства вычисления CRC8 в RTL-view (Register transfer level).

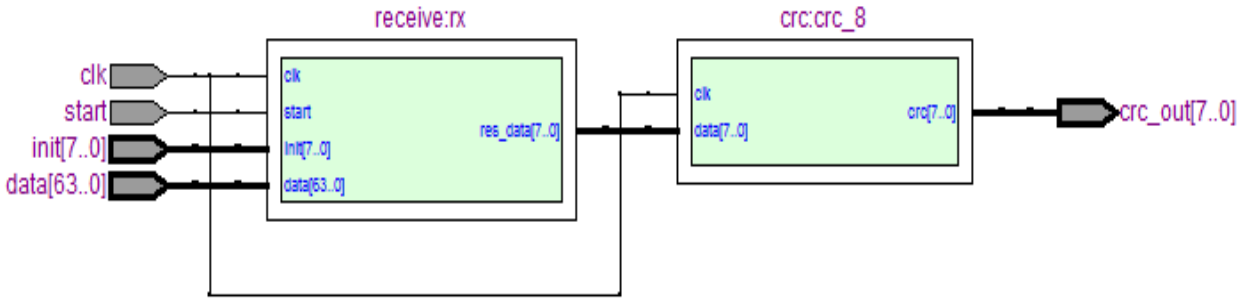


Рисунок 2.18 – RTL-view устройства вычисления CRC8

Вычисление по матричному алгоритму осуществляется согласно схеме на рисунке 2.19.

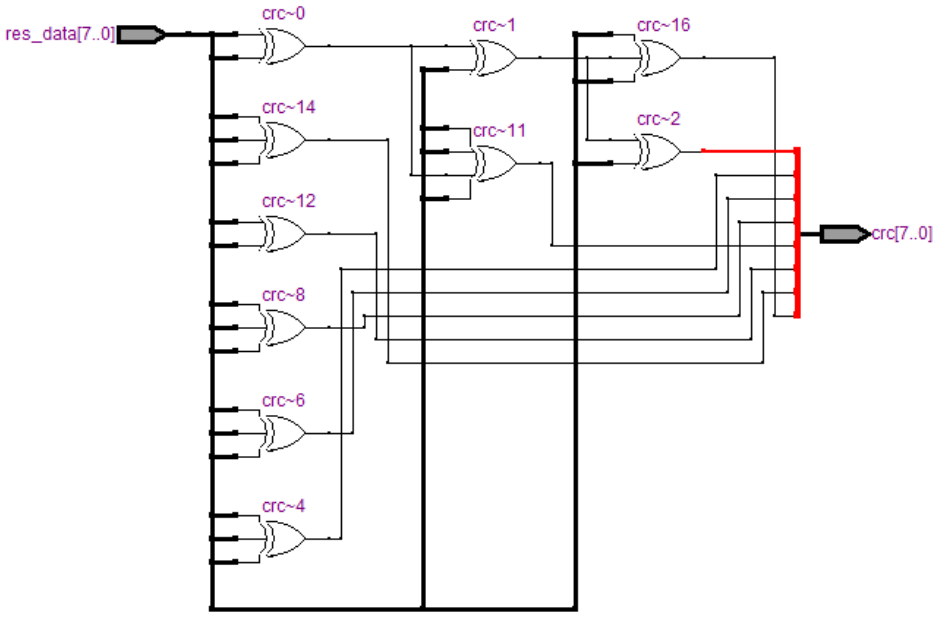


Рисунок 2.19 – Комбинационная схема блока вычисления CRC8 по матричному алгоритму

2.4.3. Обсуждение результатов по CRC8

На рисунке 2.20 представлены аппаратные затраты и быстродействие устройства вычисления CRC8 по табличному алгоритму при реализации на ПЛИС Cyclone III EP3C5E144C8.

Slow 1200mV 85C Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	182.35 MHz	182.35 MHz	clk	

Entity	Logic Cells	Dedicated Logic Re...	Memory Bits
Cyclone III: EP3C5E144C8			
crc8	72 (0)	72 (0)	2048
crc:crc_8	8 (8)	0 (0)	2048
receive:rx	72 (72)	72 (72)	0

Рисунок 2.20 – Характеристики устройства вычисления CRC8 по табличному алгоритму

Временные диаграммы работы устройства вычисления CRC8 по табличному алгоритму приведены на рисунке 2.21. Для заданной строки шестнадцатеричных значений 0102030405060708 код CRC8 (86h) вычисляется за 8 тактов.

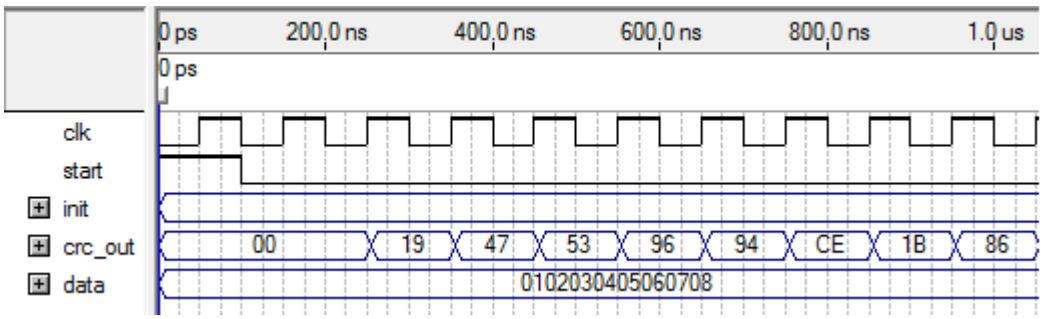


Рисунок 2.21 – Диаграмма работы устройства вычисления CRC8 по табличному алгоритму

На рисунке 2.22 приведены аппаратные затраты и быстродействие устройств вычисления CRC8 по матричному алгоритму для разных блоков, обрабатываемых за итерацию.

Slow 1200mV 85C Model Fmax Summary — 1 байт

	Fmax	Restricted Fmax	Clock Name	Note
1	386.1 MHz	386.1 MHz	clk	

Entity	Logic Cells	Dedicated Logic Re...	Memory Bits
Cyclone III: EP3C5E144C8			
crc8	80 (0)	80 (0)	0
crc:crc_8	13 (13)	8 (8)	0
receive:rx	72 (72)	72 (72)	0

Slow 1200mV 85C Model Fmax Summary — 2 байта

	Fmax	Restricted Fmax	Clock Name	Note
1	295.42 MHz	295.42 MHz	clk	

Entity	Logic Cells	Dedicated Logic Re...	Memory Bits
Cyclone III: EP3C5E144C8			
crc8	90 (0)	88 (0)	0
crc:crc_8	21 (21)	8 (8)	0
receive:rx	80 (80)	80 (80)	0

Slow 1200mV 85C Model Fmax Summary — 4 байта

	Fmax	Restricted Fmax	Clock Name	Note
1	266.88 MHz	250.0 MHz	clk	limit due to minimum period restriction (max I/O toggle rate)

Entity	Logic Cells	Dedicated Logic Re...	Memory Bits
Cyclone III: EP3C5E144C8			
crc8	110 (0)	104 (0)	0
crc:crc_8	38 (38)	8 (8)	0
receive:rx	96 (96)	96 (96)	0

Slow 1200mV 85C Model Fmax Summary — 8 байт

	Fmax	Restricted Fmax	Clock Name	Note
1	233.1 MHz	233.1 MHz	clk	

Entity	Logic Cells	Dedicated Logic Re...	Memory Bits
Cyclone III: EP3C5E144C8			
crc8	85 (0)	72 (0)	0
crc:crc_8	54 (54)	8 (8)	0
receive:rx	65 (65)	64 (64)	0

Рисунок 2.22 – Характеристики устройств вычисления CRC8 по матричному алгоритму

Диаграммы работы устройств вычисления CRC8 по матричному алгоритму приведены на рисунке 2.23.

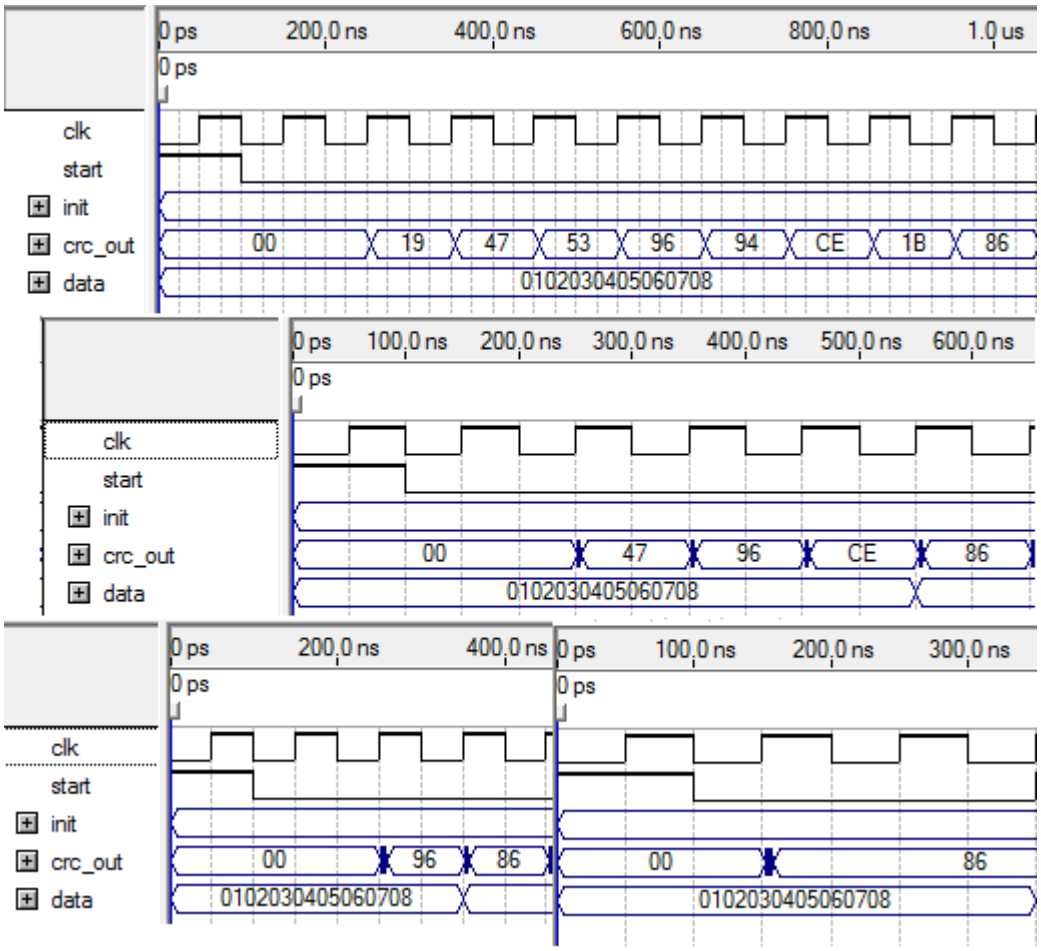


Рисунок 2.23 – Диаграммы работы устройств вычисления CRC8 по матричному алгоритму

Для анализа аппаратных реализаций алгоритмов вычисления CRC8 характеристики разработанных устройств на ПЛИС Cyclone III EP3C5E144C8 сведены в таблицу 2.14.

Таблица 2.14 –Характеристики устройств вычисления CRC8

Алгоритм	**Макс. частота, МГц**	**Такты**	**Мин. время, нс**	**Ячейки**		**Память, бит**
				модуль приёма	**модуль CRC8**	
Табличный	182	8	43,9	72	8	2048
Матричный 1 байт	386	8	20,7	72	13	0
Матричный 2 байта	295	4	13,5	80	21	0
Матричный 4 байта	266	2	7,5	96	38	0
Матричный 8 байт	233	1	4,3	65	54	0

Табличный и матричный однобайтовый алгоритмы при одинаковом количестве тактов показывают разное быстродействие и аппаратные затраты. Максимальная частота работы устройств составляет 182 МГц и 386 МГц соответственно.

Оценка эффективности устройств вычисления CRC8 приведена в таблице 2.15, в которой показано во сколько раз устройства на основе матричного алгоритма с модификациями быстрее устройств на основе табличного алгоритма. Также приведён прирост количества логических элементов для модуля вычисления CRC8 при увеличении размера матричного сдвига. В строке «эффективность» приведены значения вычисленных коэффициентов, которые являются отношением прироста быстродействия к приросту количества логических элементов. Наибольшее значение коэффициента показывает наилучшую эффективность.

Таблица 2.15 – Показатели эффективности устройств вычисления CRC8

Характеристика	**Табличный**	**М 1 байт**	**М 2 байта**	**М 4 байта**	**М 8 байт**
Быстродействие	1	2,12	3,25	5,85	10,2
Логические элементы	1	1,62	2,62	4,75	6,75
Эффективность	1	1,3	1,24	1,23	1,51

Аппаратная реализация однобайтового матричного алгоритма является примерно в 2 раза более быстродействующей, чем реализация табличного. Однако для реализации матричного алгоритма требуется на 38 % больше логических элементов, при этом для реализации таблицы предвычисленных значений требуется 2048 бит памяти ROM.

Увеличение матричного сдвига позволяет увеличить быстродействие устройства вычисления CRC за счет уменьшения количества тактов устройства. При этом максимальная частота работы снижается из-за увеличения критического пути прохождения сигнала. При обработке данных по 8 байт контрольная сумма CRC вычисляется за 1 такт при максимальной частоте 233 МГц, что позволяет достичь прироста производительности до 10 раз, однако количество логических элементов при этом увеличится в 6,75 раз.

Таким образом, наиболее эффективной, по предложенной оценке, является матричная реализация алгоритма вычисления CRC8 с обработкой по 8 байт. Наименее эффективной, по данной оценке, является реализация табличного алгоритма.

Из сравнения результатов, приведенных в таблицах 2.13 и 2.14 можно сделать вывод о значительном приросте быстродействия вычисления CRC за счёт параллельной архитектуры ПЛИС и хорошей адаптации алгоритмов под параллельные вычисления. В отличие от микроконтроллерной реализации, где вычисления производятся строго последовательно по тактовому сигналу, архитектура ПЛИС позволяет параллельно обрабатывать несколько байт данных, что хорошо подходит для реализации матричного алгоритма.

2.4.4. Разработка устройств вычисления CRC32 на ПЛИС

В протоколе Ethernet для контроля целостности данных применяется контрольная сумма CRC32, которая вычисляется аппаратно на специальной микросхеме [24, 25] сетевой платы. При реализации собственных протоколов, аналогичных Ethernet с применением ПЛИС необходимо также разрабатывать модули расчёта CRC32 (рисунок 2.24) с учетом требуемого быстродействия и аппаратных ресурсов. На каждой итерации алгоритма предыдущее значение CRC складывается по модулю 2 с текущим, и результат записывается в результирующий регистр CRC. После обработки всех байт данных в регистре CRC содержится итоговая контрольная сумма CRC32.

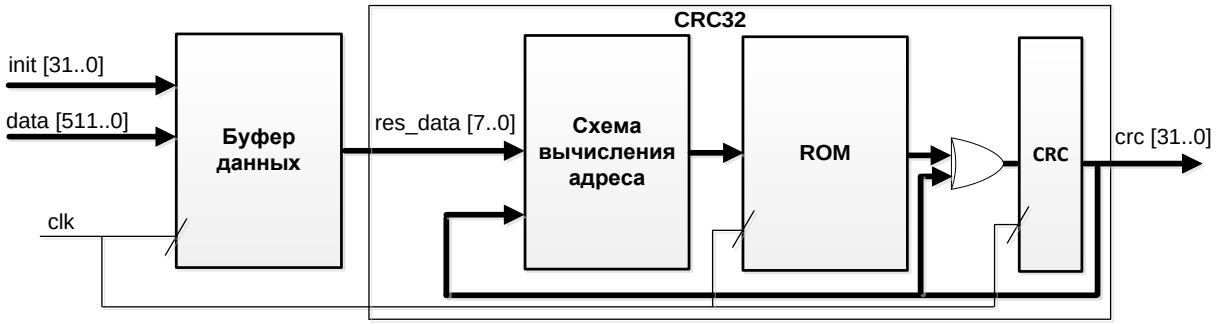


Рисунок 2.24 – Структурная схема устройства вычисления CRC32 по табличному алгоритму с памятью ROM

По аналогии с устройством вычисления CRC8 блок вычисления можно сделать асинхронным (рисунок 2.25) для устранения дополнительных задержек синхросигнала.

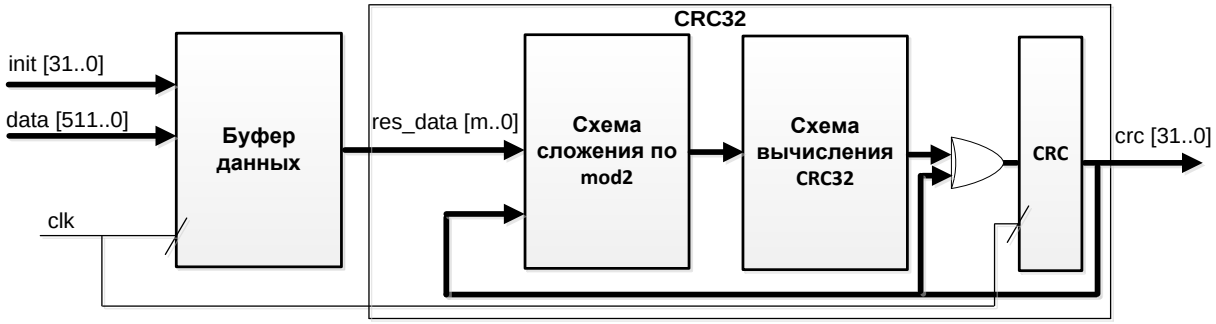


Рисунок 2.25 – Общая структурная схема устройства вычисления CRC32 по матричному алгоритму

Применяя матричную обработку по 64 байта за итерацию, можно построить устройство без обратной связи и обработать весь пакет данных за один такт (рисунок 2.26).

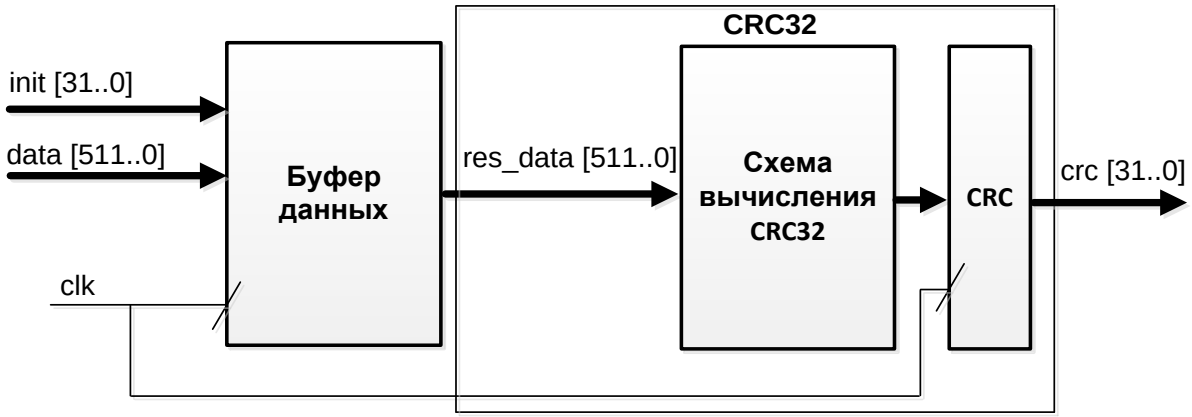


Рисунок 2.26 – Структурная схема устройства вычисления CRC32 по матричному алгоритму с обработкой по 64 байта за такт

2.4.5. Обсуждение результатов по CRC32

На рисунке 2.27 представлены аппаратные затраты и быстродействие устройства вычисления CRC32 по табличному алгоритму при реализации на ПЛИС Cyclone III EP3C5E144C8.

Slow 1200mV 85C Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	181.79 MHz	181.79 MHz	clk	

Entity	Logic Cells	Dedicated Logic Re...	Memory Bits
Cyclone III: EP3C120F780C8			
crc32	552 (0)	544 (0)	8192
crc:crc_32	40 (40)	24 (24)	8192
data:rx	520 (520)	520 (520)	0

Рисунок 2.27 – Характеристики устройства вычисления CRC32 по табличному алгоритму

Диаграммы работы устройства на ПЛИС для вычисления CRC32 по табличному алгоритму приведены на рисунке 2.28.

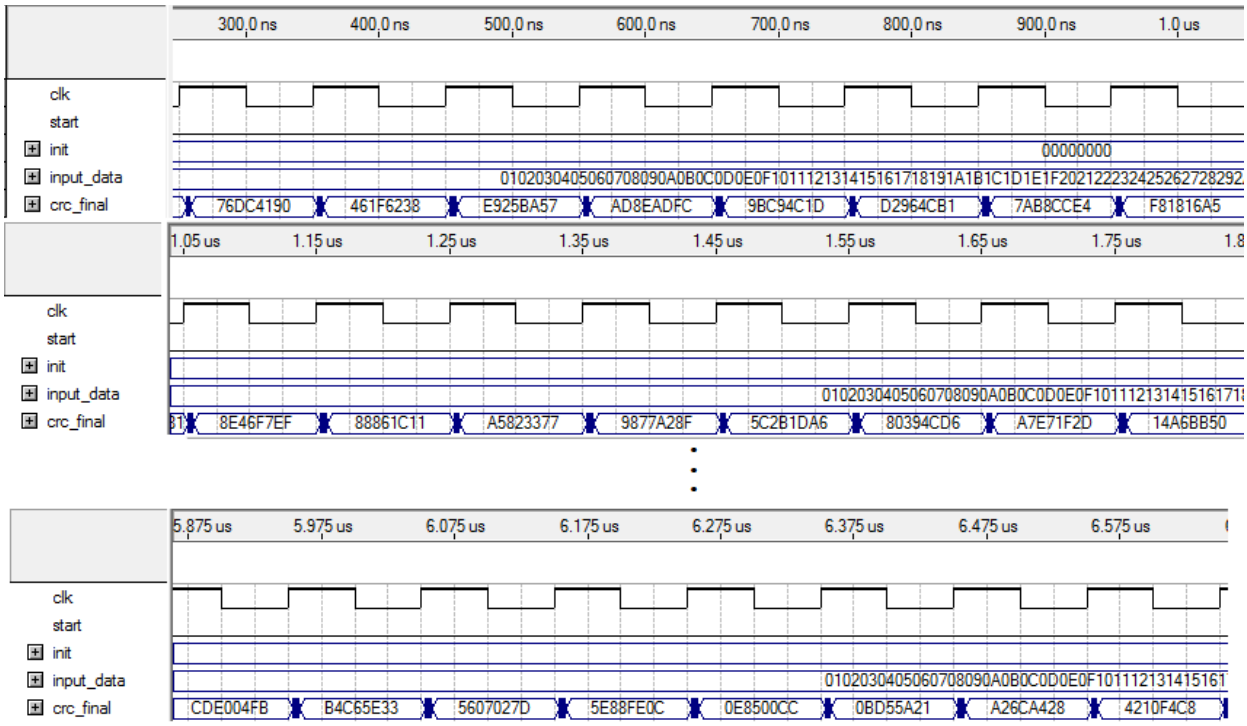


Рисунок 2.28 – Диаграмма работы устройства вычисления CRC32 по табличному алгоритму

Для заданной строки шестнадцатеричных значений 0102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f40 код CRC32 (4210F4C8h) вычисляется за 64 такта.

На рисунке 2.29 (а,б) приведены аппаратные затраты и быстродействие устройств вычисления CRC32 по матричному алгоритму для разных длин блоков, обрабатываемых за итерацию.

а)

Slow 1200mV 85C Model Fmax Summary — 1 байт

	Fmax	Restricted Fmax	Clock Name	Note
1	388.35 MHz	388.35 MHz	clk	

Entity	Logic Cells	Dedicated Logic Re...	Memory Bits
Cyclone III: EP3C120F780C8			
crc32	568 (0)	552 (0)	0
crc:crc_32	48 (48)	32 (32)	0
data:rx	520 (520)	520 (520)	0

Slow 1200mV 85C Model Fmax Summary — 2 байта

	Fmax	Restricted Fmax	Clock Name	Note
1	289.44 MHz	289.44 MHz	clk	

Entity	Logic Cells	Dedicated Logic Re...	Memory Bits
Cyclone III: EP3C5E144C8			
crc32	615 (0)	560 (0)	0
crc:crc_32	89 (89)	32 (32)	0
data:rx	528 (528)	528 (528)	0

Slow 1200mV 85C Model Fmax Summary — 4 байта

	Fmax	Restricted Fmax	Clock Name	Note
1	228.62 MHz	228.62 MHz	clk	

Entity	Logic Cells	Dedicated Logic Re...	Memory Bits
Cyclone III: EP3C5E144C8			
crc32	698 (0)	576 (0)	0
crc:crc_32	155 (155)	32 (32)	0
data:rx	544 (544)	544 (544)	0

Slow 1200mV 85C Model Fmax Summary — 8 байт

	Fmax	Restricted Fmax	Clock Name	Note
1	194.44 MHz	194.44 MHz	clk	

Entity	Logic Cells	Dedicated Logic Re...	Memory Bits
Cyclone III: EP3C5E144C8			
crc32	816 (0)	640 (0)	0
crc:crc_32	251 (251)	64 (64)	0
data:rx	577 (577)	576 (576)	0

б)

Slow 1200mV 85C Model Fmax Summary

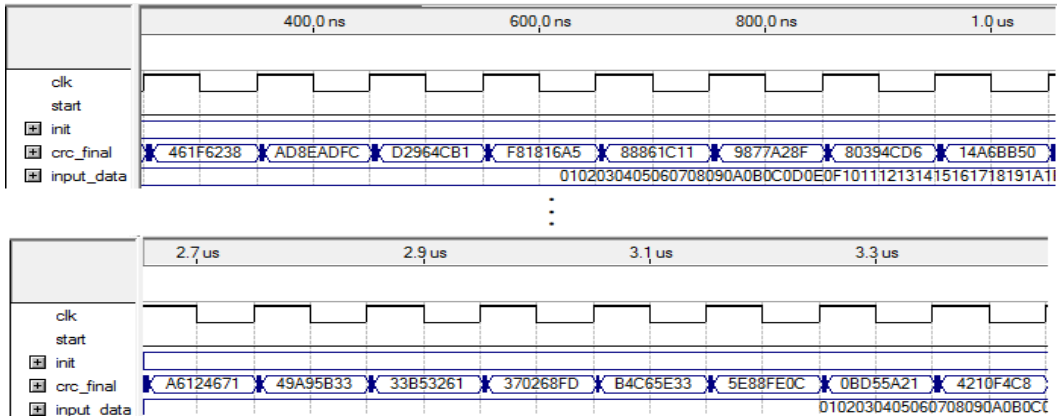
	Fmax	Restricted Fmax	Clock Name	Note	64 байта
1	102.46 MHz	102.46 MHz	clk		

Entity	Logic Cells	Dedicated Logic Re...	Memory Bits
Cyclone III: EP3C5E144C8			
crc32	1542 (0)	544 (0)	0
crc:crc_32	1203 (1203)	32 (32)	0
data:rx	519 (519)	512 (512)	0

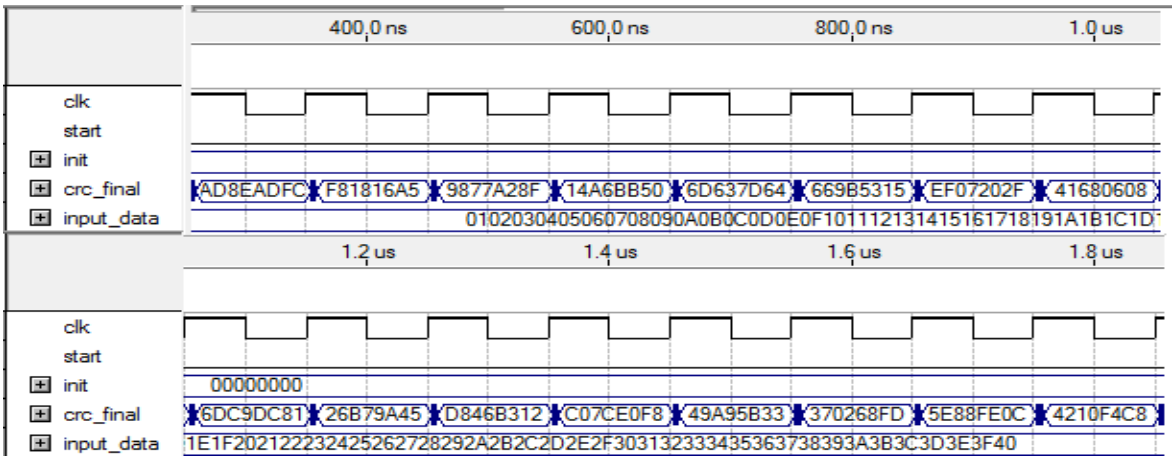
Рисунок 2.29 – Характеристики устройства вычисления CRC32 по табличному алгоритму: а) блок 1,2,4,8 байт; б) блок 64 байта

Диаграммы работы устройств вычисления CRC32 по матричному алгоритму приведены на рисунке 2.30 (а–в).

а)



б)



в)

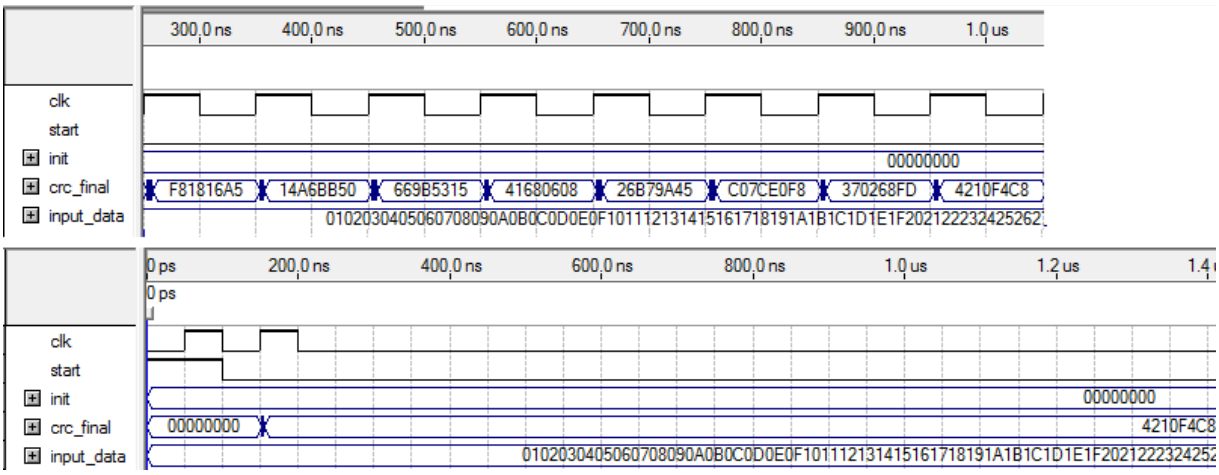


Рисунок 2.30 – Диаграммы работы устройств вычисления CRC32 по матричному алгоритму: а) блок 2 байта; б) блок 4 байта; в) блоки 8 и 64 байта

Проведем сравнение характеристик разработанных устройств вычисления CRC32 (таблица 2.16).

Таблица 2.16 – Характеристики устройств вычисления CRC32

Алгоритм	**Макс. частота, МГц**	**Такты, шт**	**Мин. время, нс**	**Ячейки, шт**		**Память, бит**
				модуль приём	**модуль CRC32**	
Табличный (с ROM)	181	64	353,1	520	40	8192
Матричный 1 байт	388	64	164,9	520	48	0
Матричный 2 байта	289	32	110,7	528	89	0
Матричный 4 байта	228	16	70,1	544	155	0
Матричный 8 байт	194	8	41,2	577	251	0
Матричный 64 байта	102	1	9,8	519	1203	0

При однобайтовой обработке матричный алгоритм вычисляет CRC32 примерно в 2 раза быстрее, чем табличный. Увеличение длины блока, обрабатываемого за итерацию, несмотря на падение максимальной частоты из-за увеличения критического пути прохождения сигнала, приводит к росту быстродействия за счёт уменьшения количества тактов, необходимых для обработки всего пакета данных.

В таблице 2.17 приведены значения вычисленных коэффициентов, которые являются отношением прироста минимального времени вычисления к приросту количества логических элементов (по аналогии с CRC8).

Таблица 2.17 – Показатели эффективности устройств вычисления CRC32

Хар-ка	**Табл.**	**М 1 байт**	**М 2 байта**	**М 4 байта**	**М 8 байт**	**М 64 байта**
Быстродействие	1	2,14	3,18	5,03	8,57	36
Лог. элементы	1	1,2	2,22	3,87	6,27	30
Эффективность	1	1,78	1,43	1,29	1,36	1,2

Таким образом, наиболее эффективной, по предложенной оценке, является матричная реализация алгоритма вычисления CRC32 с побайтовой обработкой. Наименее эффективной, по данной оценке, является реализация табличного алгоритма. Увеличение размера блока при матричной обработке до 8 байт позволяет повысить быстродействие до 8,5 раз относительно табличного алгоритма при росте количества логических элементов модуля вычисления CRC32 до 6,2 раз. Увеличение размера блока при матричной обработке до 64 байт позволяет повысить быстродействие до 36 раз при росте количества логических элементов до 30 раз.

2.5. Результаты и выводы по главе

1. Рассмотрен матричный алгоритм вычисления контрольной суммы CRC с модификациями, позволяющими увеличить размер блока данных, обрабатываемого за итерацию. Установлено, что матричный алгоритм для программной реализации требует в 8–32 раза меньший объем памяти для хранения предвычисленных значений (в зависимости от длины блока, обрабатываемого за итерацию), чем при реализации табличного алгоритма.
2. Поставлен компьютерный эксперимент по вычислению контрольной суммы CRC32 для различных данных объёмом от 10 до 1010 Мб с различными буферами, установленными в программной реализации. Результаты эксперимента показали, что реализация матричного

алгоритма с обработкой по 4 байта и буфером, равным 4 байта, на 12% быстрее, чем реализация табличного алгоритма, при этом требует в 8 раз меньший объем памяти.

3. Поставлен эксперимент на микроконтроллере ATtiny44 по вычислению контрольной суммы CRC8 для системы измерения температуры с применением датчика DS18B20. По результатам эксперимента выработаны рекомендации к применению алгоритмов расчёта CRC для систем с дефицитом ресурсов – небольшим объёмом памяти программ и данных и низкой производительностью.
4. Разработаны устройства вычисления CRC 8 и CRC32 на ПЛИС Cyclone III от Altera. Результаты исследования реализаций показали, что при использовании матричного алгоритма быстродействие устройств увеличивается до 2 раза, а увеличение длины блока, обрабатываемого за итерацию, позволяет увеличить быстродействие устройства вычисления CRC до 8–10 раз, однако при этом увеличивается количество логических элементов ПЛИС. Установлено, что в отличие от микроконтроллерной реализации, где вычисления производятся строго последовательно по тактовому сигналу, архитектура ПЛИС позволяет параллельно обрабатывать несколько байт данных, что хорошо подходит для реализации матричного алгоритма.

ГЛАВА 3. АЛГОРИТМЫ, ПРОГРАММЫ И УСТРОЙСТВА ИСПРАВЛЕНИЯ ОШИБОК

Для исправления ошибок в процессе передачи или хранения данных без повторного запроса применяются корректирующие помехоустойчивые коды. Циклические помехоустойчивые коды являются широко применяемыми на практике [26, 27, 29, 37] для исправления различных видов ошибок в системах передачи и хранения данных. Коды БЧХ обладают хорошими корректирующими свойствами и позволяют исправлять независимые ошибки большой кратности, однако, длина информационного блока таких кодов ограничена, а при укорачивании кода увеличивается избыточность и теряется эффективность кода. При этом длина кода и корректирующая способность задаётся образующим многочленом (далее *образующий полином*), который является произведением неприводимых многочленов.

Применяя программные методы поиска образующего полинома, ориентированные на исправление независимых ошибок, можно подобрать образующий полином по заданным параметрам m – длина информационного блока и t – корректирующая способность кода, что удобно для проектирования устройств исправления ошибок.

Применяя методы поиска образующего полинома, ориентированные на исправление пакетных ошибок, можно подобрать образующий полином, позволяющий строить эффективные помехоустойчивые коды, аналогичные по назначению кодам Рида-Соломона.

3.1. Алгоритм поиска образующих полиномов для кодов, исправляющих независимые ошибки

Известно, что образующий полином циклического кода БЧХ в полиномиальном представлении является наименьшим общим кратным (НОК) нечетных минимальных многочленов [43]:

$$G(x) = \text{НОК}[\Phi_1(x), \Phi_3(x), \ldots, \Phi_{2t-1}(x)].$$

Это позволяет строить образующий полиномы, применяя математический аппарат, но накладывает ограничения на длину кодов БЧХ, которая должна соответствовать выражению $n = 2^h - 1$, где h -любое целое число. Чтобы обойти данное ограничение, необходимо применить операцию *укорачивания* кода, т.е. сократить длину информационного блока с сохранением длины избыточного блока. Укороченные помехоустойчивые коды являются менее эффективными, чем неукороченные и при реализации устройств декодирования требуют больших аппаратных затрат.

Известен алгоритм поиска образующих полиномов [84], позволяющий находить полиномы меньшей длины, чем полиномы для построения некоторых БЧХ кодов, при этом входными параметрами являются длина информационного блока (m) и количество исправляемых ошибок (t). Данный алгоритм заключается в полном переборе всех полиномов – претендентов заданной длины $k + 1$, где $k = \log_2(\sum_{i=1}^{t} C_n^i)$ – длина контрольного блока, и проверке их на выполнение необходимых условий:

1) строящийся на основе этого образующего полинома код должен иметь расстояние Хэмминга не меньшее, чем $d=2t+1$, где t – количество исправляемых ошибок;

2) остатки от деления всех комбинаций ошибок на образующий полином в рамках заданной t должны быть уникальными.

В таблице 3.1 приведены образующие полиномы для $m \in [1,32]$ и $t \in [1,4]$, полученные с помощью алгоритма [84].

Таблица 3.1 – Образующие полиномы в двоичном виде

m \ t	1	2	3	4
1	111	11111	1111111	111111111
2	1101	1101101	1110110101	1110101110101
3	1101	11001101	11101100101	111010011101001
4	1101	11100101	11101100101	1111000101100101
5	11001	110101001	11101100101	1111000100101101
6	11001	111010001	101101110001	10111000110110001
7	11001	111010001	110001110101	111000101001011001

Окончание таблицы 3.1

8	11001	100111001	110001110101	110000111000110101
9	11001	100111001	110001110101	111000110010010101
10	11001	1100001101	110001110101	100110000101010110 1
11	11001	1101000011	110001110101	100011111001010000 1
12	110001	1101000011	110001110101	111000010101100000 11
13	110001	1101000011	110010101000101	101100011001010010 01
14	110001	11000011001	1100001010010101	111001000000100110101
15	110001	11000011001	1100001010010101	111001110001000000101
16	110001	11000011001	1100001010010101	111101100000100000010 1
17	101001	11000011001	11000011010000011	111101000010001100000 1
18	101001	11000011001	11001000001100101	111000010001110001000 1
19	101001	11000011001	11010000100001001 1	101110010000000111000 1
20	101001	11000011001	10011001000110000 1	111100010000101000010 01
21	101001	11000011001	10011001000110000 1	110010110001000000101 0
22	101001	101100001001	10011001000110000 1	111100100000010001001 001
23	101001	101100001001	11001010000110000 1	111100100000010001001 001
24	101001	101100001001	11001010000110000 1	111010001000100001000 011
25	101001	1100010010001	11001010000110000 1	111010001000100001000 011
26	101001	1101010000001	11001010000110000 1	111010001000100001000 011
27	1100001	1100010010001	11001001000100010 01	111100100001001000000 0011
28	1100001	1010110000001	11010001010000000 11	111100100001001000000 0011
29	1100001	1010110000001	11010001010000000 11	110011000000001000101 0011
30	1100001	1001110010101	11110011010000011 11	111011101110010011011 0111
31	1100001	1001110010101	11110011010000011 11	111011101110010011011 0111
32	1100001	1001110010101	11110011010000011 11	111011101110010011011 0111

Приведенный в [84] алгоритм обладает рядом недостатков, которые не позволяют осуществлять быстрый поиск полиномов и строить помехоустойчивые коды, обладающие свойством цикличности:

1) Алгоритм не учитывает проверку полинома на цикличность. В алгоритме отсутствует проверка следующего условия: остаток от деления полинома $x^n + 1$ на образующий полином должен быть нулевым, где n – длина кодового слова. Отсутствие данной проверки говорит о том, что не гарантируется цикличность кодов, построенных с помощью найденных образующих полиномов, и такие коды не могут быть декодированы циклическим алгоритмом. Для декодирования таких кодов можно применять табличный алгоритм, который требует объём памяти $V = n \cdot 2^k$ бит, где k – длина контрольного блока кодового слова.

2) В алгоритме введено ограничение на вес полинома–кандидата $w = t$, что приводит к уменьшению количества полиномов-кандидатов длины $k + 1$ и соответственно к увеличению длины полинома, и, следовательно, к увеличению избыточности построенного кода.

3) Алгоритм требует трудоёмких вычислений и при этом не адаптирован под реализацию с применением технологий параллельных или распределённых вычислений. Адаптация алгоритма под параллельные вычисления позволит существенно уменьшить время поиска образующего полинома [85–91].

Учитывая приведенные недостатки данный алгоритм необходимо модифицировать следующим образом:

- добавить проверку на цикличность;
- убрать ограничение полинома по весу;
- провести анализ алгоритма на возможность применения технологий параллельных или распределённых вычислений для поиска образующего полинома.

В общем виде, модифицированный алгоритм поиска будет включать 3 этапа:

1) проверка условия деления без остатка хотя бы одного из полиномов $x^n + 1$ на образующий полином, где n изменяется от $m + k$ до $2^{\lfloor \log_2(m+k) \rfloor + 1} - 1$;
2) проверка всех кодовых слов длины m, строящихся для искомого полинома, на соответствие расстоянию Хэмминга не меньшему, чем $d=2t+1$, где t – количество исправляемых ошибок;
3) проверка на уникальность остатков от деления (синдромов) всех шаблонов ошибок кратности t на образующий полином.

На рисунке 3.1 приведена схема модифицированного алгоритма поиска образующего полинома с учетом устранения недостатков, описанных ранее.

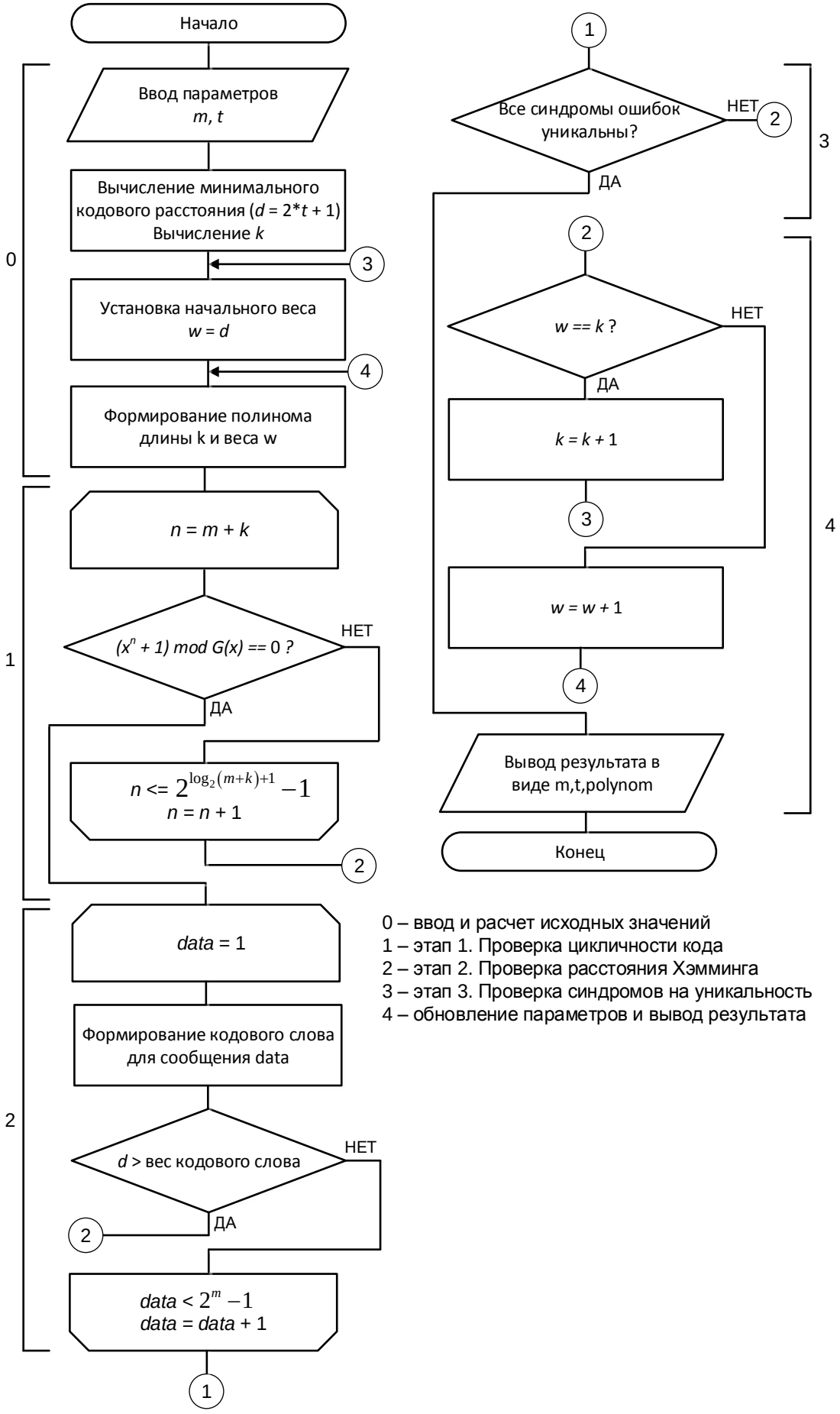


Рисунок 3.1 – Алгоритм поиска образующего полинома для циклических кодов

Для того чтобы найти подходящий полином необходимо проверить все полиномы на выполнения этапов 1 – 3, и если среди них не найдётся полином, который будет удовлетворять необходимым условиям, то поиск продолжится среди полиномов, длина которых на единицу больше, чем предыдущие.

В связи с проверкой большого множества полиномов на соответствие необходимым условиям задача по поиску образующего полинома является достаточно трудоёмкой, что требует применения технологии параллельных или распределённых вычислений. Для составления таблицы полиномов с учетом предложенных модификаций необходимо провести компьютерный эксперимент с применением технологии параллельных вычислений и многоядерной системы.

3.2. Компьютерный эксперимент по поиску образующих полиномов с применением технологии OpenMP

Анализ алгоритма поиска образующего полинома (рисунок 3.1) показал, что наиболее трудоёмким является этап проверки расстояния Хэмминга для кодовых слов (этап 2), построенных на основе полинома-кандидата. При этом данный этап подлежит распараллеливанию путём распределения кодовых слов по вычислительным ядрам (потокам). Каждое вычислительное ядро осуществляет проверку условия $d=2t+1$ для своего подмножества сформированных кодовых слов (рисунок 3.2). Если расстояние Хэмминга (*distance*) хотя бы для одного из кодовых слов меньше d, то по завершению потока формируется значение 1. При завершении работы параллельной секции осуществляется проверка всех выходных значений потоков и если хотя бы у одного из потоков значение равно 1, то этап 1 для полинома-кандидата считается не пройденным. Эффективность алгоритма поиска на основе такого способа зависит от того, насколько «далеко» от начала подмножества кодовых слов выполняется условие $distance < d$.

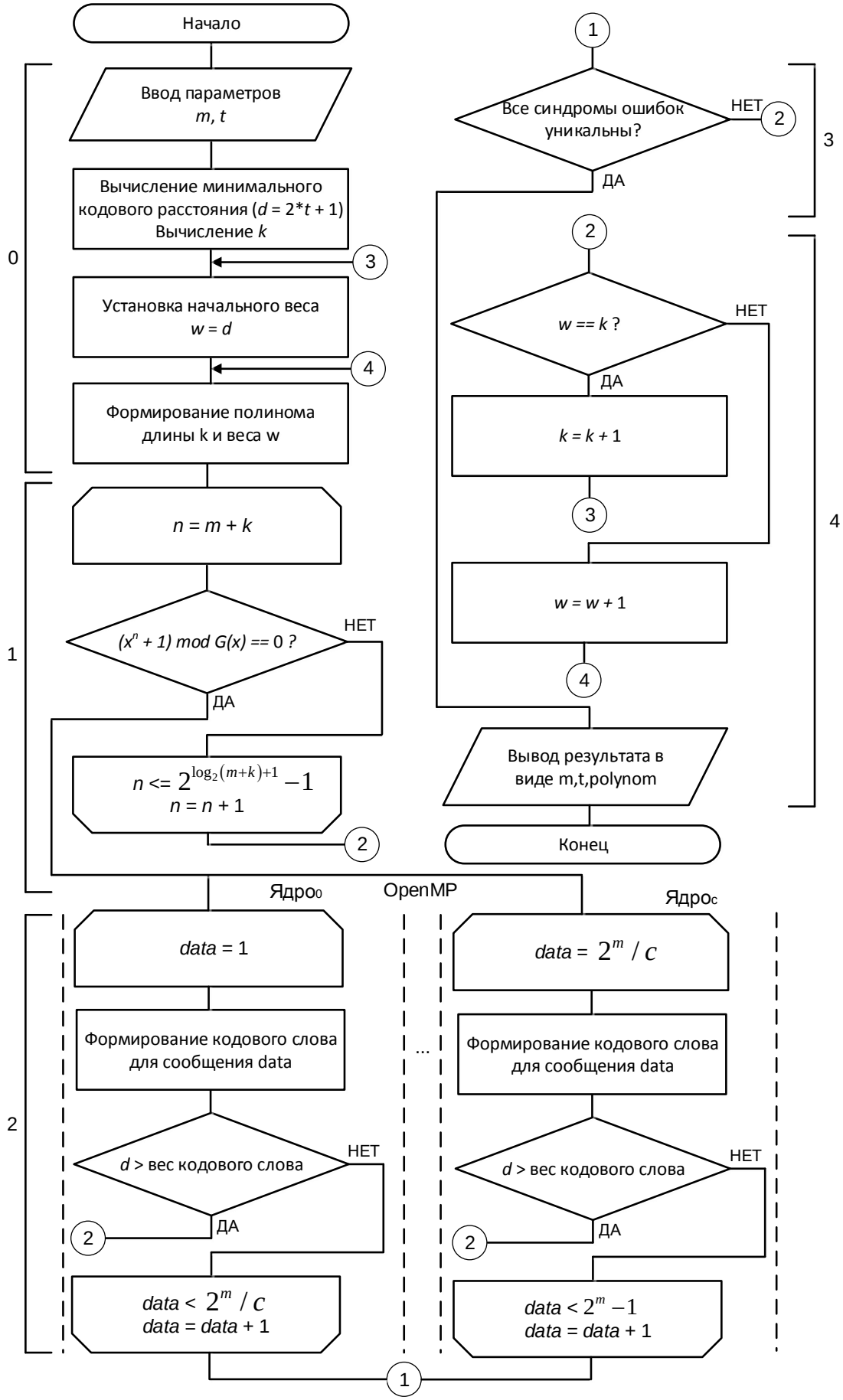


Рисунок 3.2 – Алгоритм с распараллеленным этапом 2

Для демонстрации эффективности предложенного способа распараллеливания проведен компьютерный эксперимент по поиску образующих полиномов на многоядерной вычислительной системе (раздел 3.2.1).

На разработанную программу поиска образующих полиномов с применением технологий параллельных вычислений получено свидетельство о регистрации ПО для ЭВМ [91], копия которого представлена в приложении Г.

3.2.1. Постановка компьютерного эксперимента

Эксперимент по поиску образующих полиномов проведён для разных значений длин информационного блока $m \in [1; 32]$ и количества исправляемых ошибок $t \in [2; 4]$ на многоядерной системе с двумя 4-х ядерными процессорами Intel Xeon X5450 с частотой 3 ГГц (всего 8 ядер) и оперативной памятью DDR2 FB-DIMM 667 МГц объёмом 16Гб. Для оценки эффективности программных реализаций осуществлено по 10 запусков параллельного алгоритма, реализованного с применением технологии OpenMP [92] и исходного алгоритма с вычислением среднего времени выполнения. В таблице 3.2. представлены полученные образующие полиномы по предложенному модифицированному алгоритму.

Таблица 3.2 – Значения экспериментально полученных образующих полиномов

m	t		
	2	3	4
1	10111101	110111011	11011011011011
2	1011101	10100110111	11011011011011
3	100010111	10100110111	1001101110010101
4	100010111	10100110111	1001101110010101
5	100010111	10100110111	10001100101011111
6	100010111	101011100011	100001100101100111011
7	100010111	101011100011	100001100101100111011
8	100111001	101011100011	100001100101100111011
9	100111001	101011100011	100001100101100111011
10	1100110111	101011100011	100001100101100111011

Окончание таблицы 3.2

11	1100110111	101011100011	100001100101100111011
12	1100110111	101011100011	100001111010111100001
13	10001110001	1000111110101111	100001111010111100001
14	10001110001	1000111110101111	101111100111001111101
15	10001110001	1000111110101111	101111100111001111101
16	10001110001	1000111110101111	101111100111001111101
17	10001110001	10010101000100011	101111100111001111101
18	10001110001	10010101000100011	101111100111001111101
19	10001110001	100110010011001101	101111100111001111101
20	10001110001	101110101000110111	101111100111001111101
21	10001110001	101110101000110111	101111100111001111101
22	101001100101	101110101000110111	100011000111011011101111
23	1001010011011	101110101000110111	100011000111011011101111
24	1001010011011	101110101000110111	100011000111011011101111
25	1001010101011	101110101000110111	100010101101111001101011
26	1001010101011	101110101000110111	100001011011011101101011
27	1001010101011	101110101000110111	100001011011011101101011
28	1001010101011	101110101000110111	100001011011011101101011
29	1001010101011	101110101000110111	100001011011011101101011
30	1001010101011	101110101000110111	100001011011011101101011
31	1001010101011	101110101000110111	100001011011011101101011
32	1001010101011	101110101000110111	100001011011011101101011

В таблице 3.3. представлено среднее время поиска образующих полиномов (*T*) из таблицы 3.2. для двух вариантов реализаций, где S – последовательный вариант, P – распараллеленный вариант. Для параллельных реализаций задействовано 8 ядер вычислительной системы.

Таблица 3.3 – Время поиска образующих полиномов *T*, с

m	t					
	2		3		4	
	S	P	S	P	S	P
1	0,0002	0,0039	0,0002	0,0029	0,0039	0,0075
2	0,0001	0,0049	0,0007	0,0040	0,0033	0,0066
3	0,0004	0,0048	0,0012	0,0056	0,0885	0,0919
4	0,0005	0,0036	0,0012	0,00813	0,0948	0,1210
5	0,0003	0,0035	0,0014	0,0061	0,194	0,2252

Окончание таблицы 3.3

6	0,0004	0,0089	0,0080	0,0104	3,18	3,25
7	0,0004	0,0035	0,0090	0,0107	2,90	3,01
8	0,0008	0,0049	0,0091	0,0103	2,60	2,74
9	0,0009	0,0032	0,0071	0,0104	2,26	2,29
10	0,0050	0,0070	0,0091	0,0119	1,92	1,94
11	0,0062	0,0074	0,0115	0,0173	1,55	1,58
12	0,0079	0,0078	0,0162	0,0200	2,51	2,54
13	0,0151	0,008	0,1233	0,1259	19,30	19,63
14	0,0236	0,012	0,1301	0,1196	34,95	35,41
15	0,0431	0,013	0,1478	0,1226	38,26	38,80
16	0,0827	0,017	0,1911	0,1256	39,44	40,07
17	0,1682	0,0355	2,69	1,95	45,19	45,15
18	0,3439	0,060	3,43	2,80	44,58	44,34
19	0,7083	0,115	10,03	7,07	44,19	44,18
20	1,4605	0,230	15,45	8,56	42,59	42,08
21	3,0179	0,46	17,24	9,00	43,62	41,28
22	60,96	19,22	20,87	9,84	253,16	245,03
23	84,45	54,53	28,57	11,69	263,25	240,79
24	98,57	81,21	44,34	14,89	273,55	239,39
25	158,41	86,93	77,57	22,25	517,30	434,51
26	219,82	97,10	146,04	36,14	575,93	434,99
27	345,8	117,52	287,29	64,07	720,64	463,09
28	606,2	160,15	578,09	121,27	1036,5	537,94
29	1142,1	247,69	1176,1	234,73	1706,5	699,20
30	2247,6	426,21	2409,0	466,79	3103,0	1069,3
31	4522,5	788,46	4945,9	943,95	6262,9	1787,1
32	9195,8	1441,0	10172,1	1895,21	12503,4	3297,3

На рисунках 3.3 (а,б) – 3.5 (а,б) приведены графики зависимости среднего времени поиска образующего полинома (T) от длины информационного блока (m). Для каждого из исследуемых значений $t \in [2; 4]$ (количество исправляемых ошибок) графики разбиты на 4 поддиапазона длин m: 1–8, 9 – 16, 17 – 24, 25 – 32. Также для каждого из значений t приведены значения коэффициентов ускорения распараллеленной реализации по сравнению с последовательной.

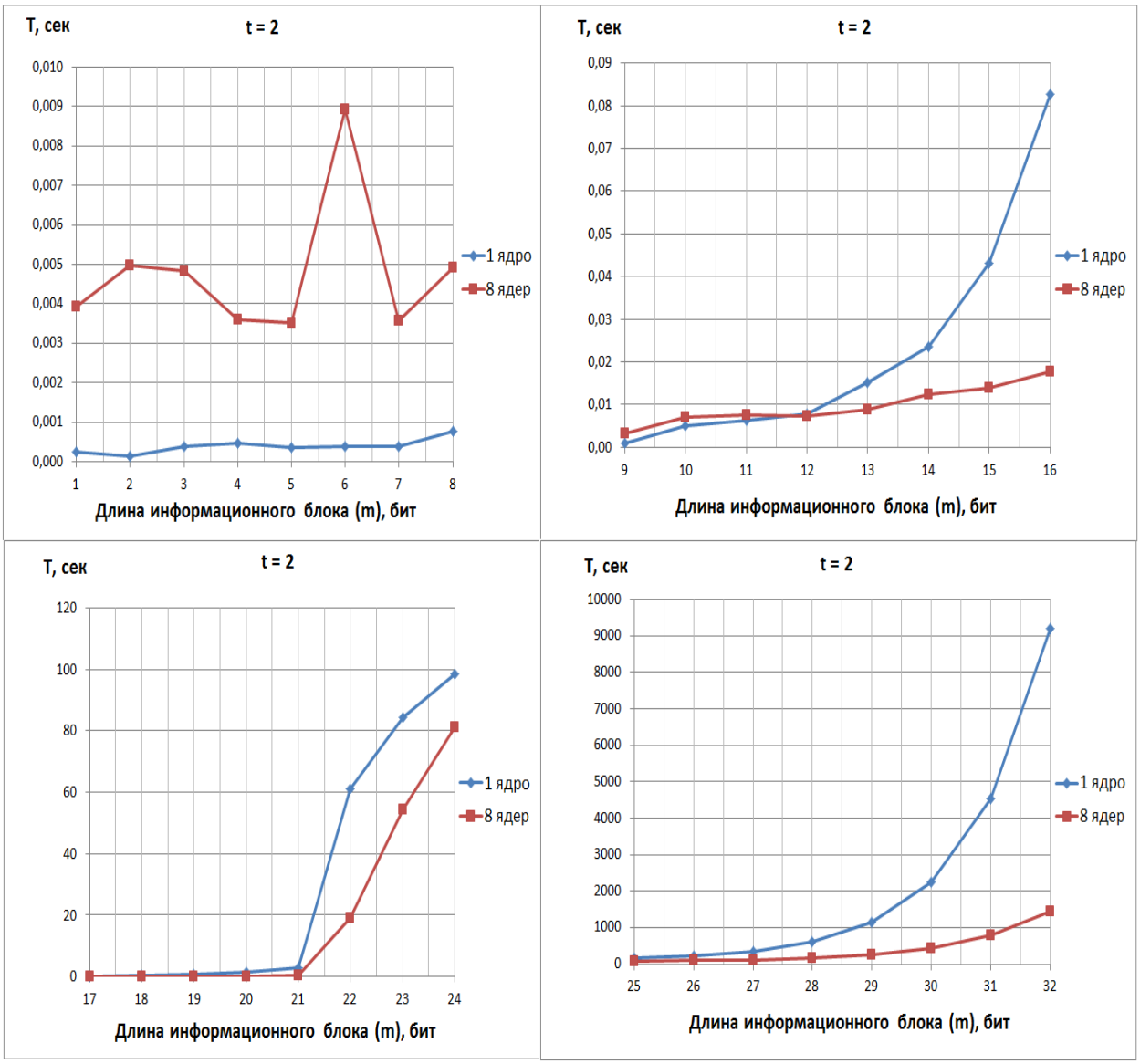


Рисунок 3.3 – Среднее время поиска образующих полиномов для $t = 2$

Из рисунка 3.3 следует, что для двукратных независимых ошибок при длине информационного блока $m \in [1, 12]$ применение технологии OpenMP не приводит к увеличению быстродействия. Рост быстродействия поиска при использовании технологии OpenMP наблюдается для $m \in [12, 21]$ с 1,7 до 6,5 раз и для $m \in [24, 32]$ с 1,2 до 6,3 раз. При этом для значений $m \in [23, 24]$ наблюдается резкий спад быстродействия ввиду особенности расположения искомого образующего полинома во множестве полиномов-кандидатов.

На рисунке 3.4 приведены результаты поиска образующего полинома для количества исправляемых ошибок, равного 3.

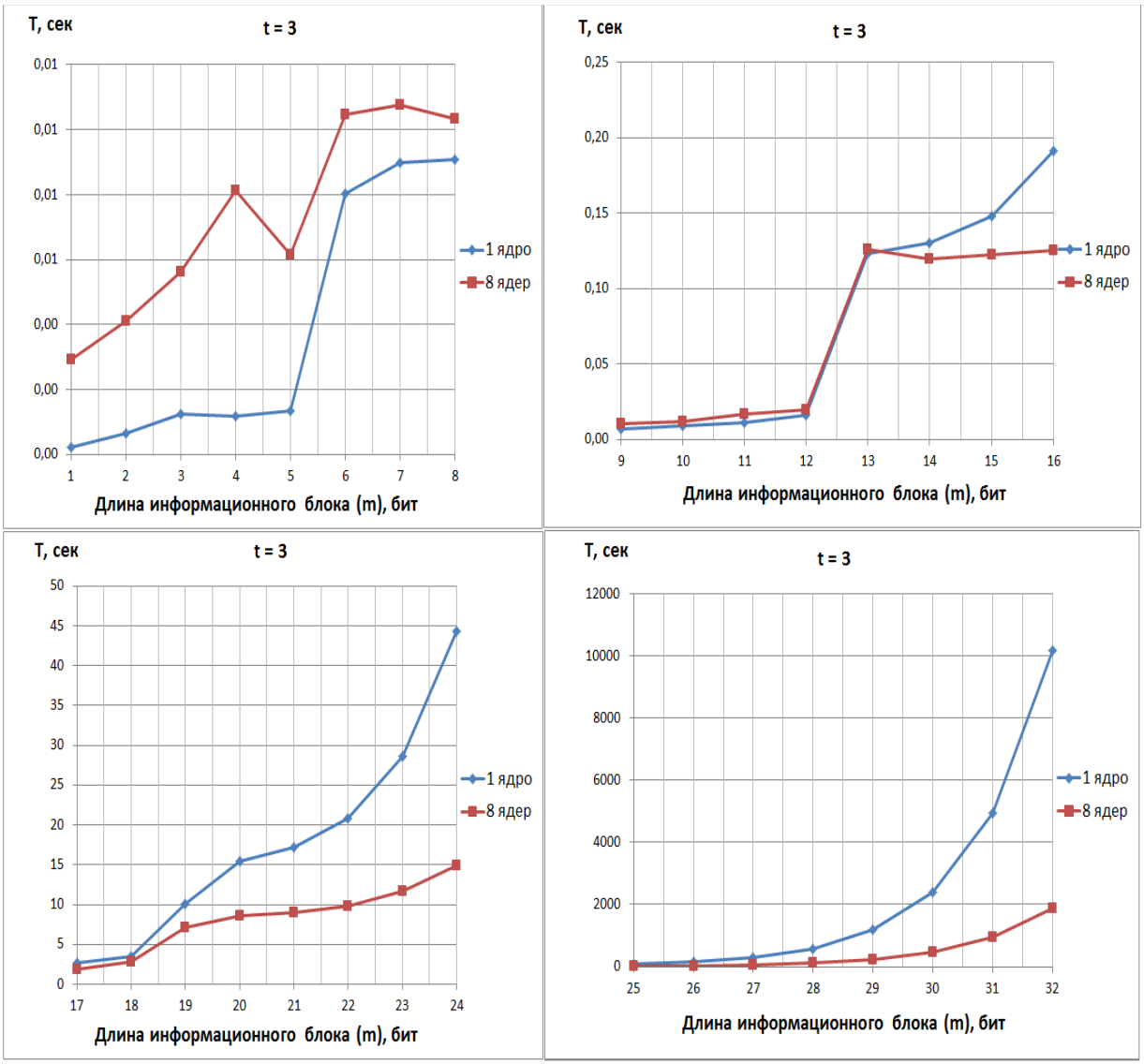


Рисунок 3.4 –. Среднее время поиска образующих полиномов для t = 3

Из графиков этого рисунка видно, что для трёхкратных независимых ошибок при длине информационного блока $m \in [1, 13]$ применение технологии OpenMP не приводит к увеличению быстродействия. Рост быстродействия поиска при использовании этой технологии параллельных вычислений наблюдается для $m \in [13, 32]$ с 1 до 5,3 раз.

На рисунке 3.5 приведены результаты поиска образующего полинома для $t = 4$. Для четырёхкратных независимых ошибок при длине информационного блока $m \in [1, 22]$ применение технологии OpenMP не приводит к увеличению быстродействия. Рост быстродействия поиска при использовании технологии OpenMP наблюдается для $m \in [22, 32]$ с 1 до 3,8 раз.

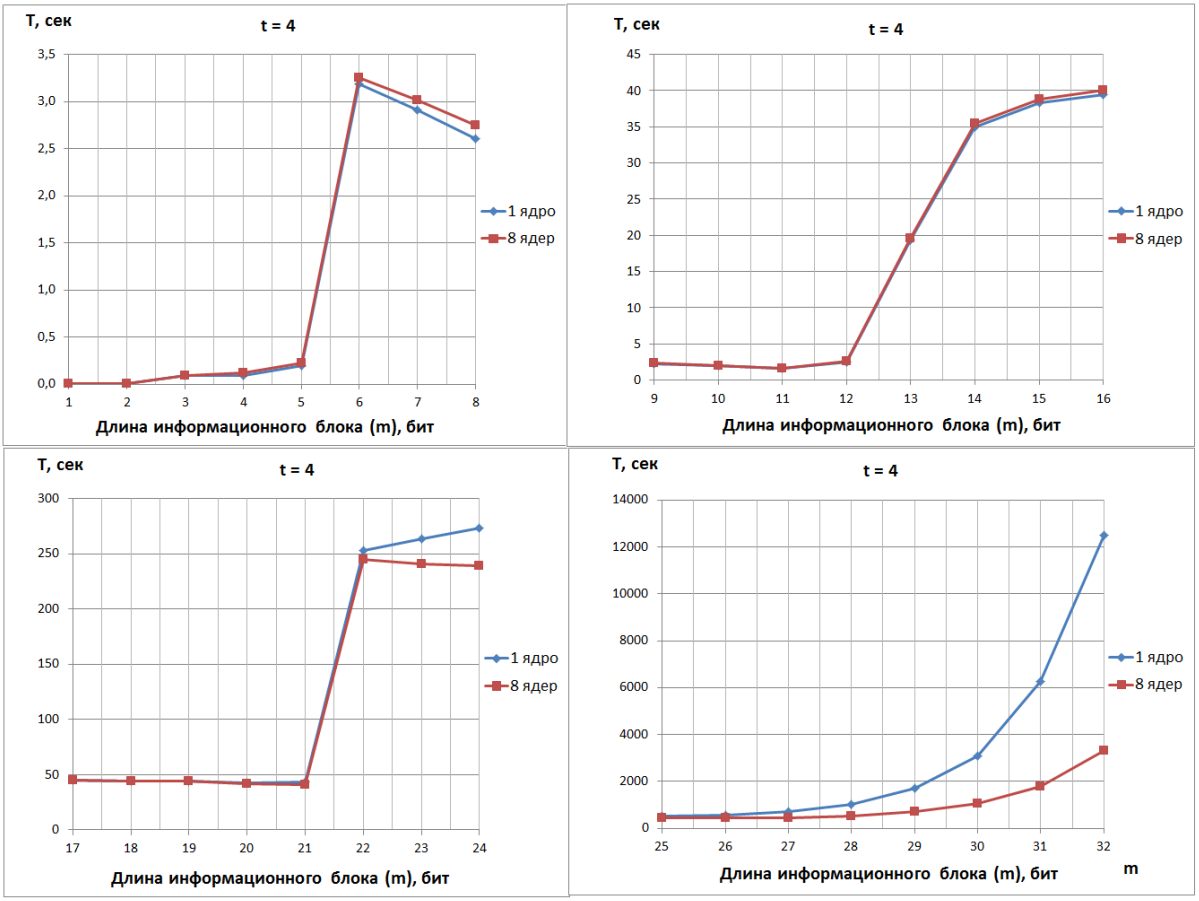


Рисунок 3.5– Среднее время поиска образующих полиномов для t = 4

В целом, по результатам компьютерного эксперимента, можно сделать вывод о том, что предложенный алгоритм поиска образующих полиномов для построения циклических помехоустойчивых кодов, исправляющих независимые ошибки длиной от 2 до 4 бит, адаптирован для применения технологии параллельных вычислений OpenMP. При этом увеличивается быстродействие поиска образующих полиномов в 3,8 – 6,3 раз при использовании 8 вычислительных ядер с частотой 3 ГГц.

3.2.2. Обсуждение результатов

Основной задачей алгоритма поиска образующих полиномов является получение полиномов для построения циклических кодов более эффективных, чем известные БЧХ-коды. В таблице 3.4 приведены длины кодовых слов для полученных образующих полиномов в сравнении с длинами кодов БЧХ,

образующие полиномы которых найдены с помощью ПО, описанного в работе [55].

Таблица 3.4 – Длины кодовых слов для образующих полиномов:
А–коды, построенные по полученному полиному, Б–коды БЧХ,
В–разность длин А – Б

Код	**А**			**Б**			**В**		
t / m	**2**	**3**	**4**	**2**	**3**	**4**	**2**	**3**	**4**
2	**8**	-	-	10	-	-	**-2**	-	-
3	11	13	-	11	13	-	0	0	-
4	12	14	**19**	12	14	24	0	0	**-5**
5	13	15	**21**	13	15	25	0	0	**-4**
6	14	**17**	26	14	21	26	0	**-4**	0
7	15	**18**	27	15	22	27	0	**-4**	0
8	**16**	**19**	28	18	23	28	**-2**	**-4**	0
9	**17**	**20**	29	19	24	29	**-2**	**-4**	0
10	**19**	**21**	30	20	25	30	**-1**	**-4**	0
11	**20**	**22**	31	21	26	31	**-1**	**-4**	0
12	**21**	**23**	**32**	22	27	36	**-1**	**-4**	**-4**
13	23	28	**33**	23	28	37	0	0	**-4**
14	24	29	**34**	24	29	38	0	0	**-4**
15	25	30	**35**	25	30	39	0	0	**-4**
16	26	31	**36**	26	31	40	0	0	**-4**
17	27	**33**	**37**	27	35	41	0	**-2**	**-4**
18	28	**34**	**38**	28	36	42	0	**-2**	**-4**
19	29	**36**	**39**	29	37	43	0	**-1**	**-4**
20	30	**37**	**40**	30	38	44	0	**-1**	**-4**
21	31	**38**	**41**	31	39	45	0	**-1**	**-4**
22	**33**	**39**	**45**	34	40	46	**-1**	**-1**	**-1**
23	35	**40**	**46**	35	41	47	0	**-1**	**-1**
24	36	**41**	**47**	36	42	48	0	**-1**	**-1**
25	37	**42**	49	37	43	49	0	**-1**	0
26	38	**43**	50	38	44	50	0	**-1**	0
27	39	**44**	51	39	45	51	0	**-1**	0
28	40	**45**	52	40	46	52	0	**-1**	0
29	41	**46**	53	41	47	53	0	**-1**	0
30	42	**47**	54	42	48	54	0	**-1**	0
31	43	**48**	55	43	49	55	0	**-1**	0
32	44	**49**	56	44	50	56	0	**-1**	0

Из этой таблицы можно сделать вывод о том, что найденные с помощью предложенного алгоритма полиномы позволяют для некоторых параметров *m* и *t* строить укороченные циклические помехоустойчивые коды с длиной кодового слова на 1–5 бит меньше укороченных кодов БЧХ.

Предложенный алгоритм поиска находит образующие полиномы как для укороченных, так и для неукороченных помехоустойчивых кодов. Такие результаты достигаются путем выполнения первого этапа поиска, в котором степень полинома $x^n + 1$ изменятся от $m + k$ до $2^{\lfloor \log_2(m+k) \rfloor + 1} - 1$. Укороченные коды получают путем уменьшения длины информационного блока при сохранении длины контрольного. Для декодирования укороченных кодов циклическим методом необходимо кодовое слово дополнить нулевыми старшими разрядам до полной длины кодового слова. В таблице 3.5 приведены полные длины кодовых слов для укороченных кодов с учетом добавления необходимого количества нулей.

Таблица 3.5 – Длины кодовых слов укороченных кодов для образующих полиномов: А – коды, построенные по найденному полиному, Б – коды БЧХ, В – разность длин А – Б

Код	**А**			**Б**			**В**		
t / m	**2**	**3**	**4**	**2**	**3**	**4**	**2**	**3**	**4**
2	15	-	-	15	-	-	0	-	-
3	15	15	-	15	15	-	0	0	-
4	15	15	31	15	15	31	0	0	0
5	15	15	**21**	15	15	31	0	0	**-10**
6	15	**23**	31	15	31	31	0	**-8**	0
7	15	**23**	31	15	31	31	0	**-8**	0
8	**17**	**23**	31	31	31	31	**-14**	**-8**	0
9	**17**	**23**	31	31	31	31	**-14**	**-8**	0
10	**21**	**23**	31	31	31	31	**-10**	**-8**	0
11	**21**	**23**	31	31	31	31	**-10**	**-8**	0
12	**21**	**23**	**33**	31	31	63	**-10**	**-8**	**-30**
13	31	31	**33**	31	31	63	0	0	**-30**
14	31	31	**41**	31	31	63	0	0	**-22**
15	31	31	**41**	31	31	63	0	0	**-22**
16	31	31	**41**	31	31	63	0	0	**-22**

Окончание таблицы 3.5

17	31	**51**	**41**	31	63	63	0	**-12**	**-22**
18	31	**51**	**41**	31	63	63	0	**-12**	**-22**
19	31	63	**41**	31	63	63	0	0	**-22**
20	31	63	**41**	31	63	63	0	0	**-22**
21	31	63	**41**	31	63	63	0	0	**-22**
22	**33**	63	**47**	63	63	63	**-30**	0	**-16**
23	63	63	**47**	63	63	63	0	0	**-16**
24	63	63	**47**	63	63	63	0	0	**-16**
25	63	63	63	63	63	63	0	0	0
26	63	63	63	63	63	63	0	0	0
27	63	63	63	63	63	63	0	0	0
28	63	63	63	63	63	63	0	0	0
29	63	63	63	63	63	63	0	0	0
30	63	63	63	63	63	63	0	0	0
31	63	63	63	63	63	63	0	0	0
32	63	63	63	63	63	63	0	0	0

Наиболее эффективными образующими полиномами являются те, которые позволяют строить неукороченные циклические помехоустойчивые коды. Для таких кодов нет необходимости добавлять нулевые разряды при выполнении операции декодирования. Из таблиц 3.2, 3.4 и 3.5 можно выделить несколько образующих полиномов, найденных с помощью предложенного алгоритма поиска, которые позволяют строить неукороченные циклические помехоустойчивые коды:

- для $t = 2$: $m = 7,9,12,21,22$;
- для $t = 3$: $m = 5,12,16$;
- для $t = 4$: $m = 5,11,13,21,24$;

Заметим, что образующие полиномы для параметров $t = 2$ и $m = 7,21$; $t = 3$ и $m = 5,16$; $t = 4$ и $m = 11$ являются полиномами неукороченных кодов БЧХ, что говорит о корректности работы алгоритма поиска.

В таблице 3.6 приведены скорости циклических помехоустойчивых кодов, построенные на основе образующих полиномов из таблицы 3.2, и их сравнение со скоростями кодов БЧХ при таких же значений m и t.

Таблица 3.6 – Скорости циклических помехоустойчивых кодов: А – для найденных полиномов, Б – для полиномов БЧХ, В – прирост скорости, %

Код	**А**			**Б**			**В**		
t / m	**2**	**3**	**4**	**2**	**3**	**4**	**2**	**3**	**4**
2	**0,25**	-	-	0,20	-	-	**25,00**	-	-
3	0,27	0,23	-	0,27	0,23	-	0	0	0
4	0,33	0,29	**0,21**	0,33	0,29	0,17	0	0	**26,32**
5	0,38	0,33	**0,24**	0,38	0,33	0,20	0	0	**19,05**
6	0,43	**0,35**	0,23	0,43	0,29	0,23	0	**23,53**	0
7	0,47	**0,39**	0,26	0,47	0,32	0,26	0	**22,22**	0
8	**0,50**	**0,42**	0,29	0,44	0,35	0,29	**12,50**	**21,05**	0
9	**0,53**	**0,45**	0,31	0,47	0,38	0,31	**11,76**	**20,00**	0
10	**0,53**	**0,48**	0,33	0,50	0,40	0,33	**5,26**	**19,05**	0
11	**0,55**	**0,50**	0,35	0,52	0,42	0,35	**5,00**	**18,18**	0
12	**0,57**	**0,52**	**0,38**	0,55	0,44	0,33	**4,76**	**17,39**	**12,50**
13	0,57	0,46	**0,39**	0,57	0,46	0,35	0	0	**12,12**
14	0,58	0,48	**0,41**	0,58	0,48	0,37	0	0	**11,76**
15	0,60	0,50	**0,43**	0,60	0,50	0,38	0	0	**11,43**
16	0,62	0,52	**0,44**	0,62	0,52	0,40	0	0	**11,11**
17	0,63	**0,52**	**0,46**	0,63	0,49	0,41	0	**6,06**	**10,81**
18	0,64	**0,53**	**0,47**	0,64	0,50	0,43	0	**5,88**	**10,53**
19	0,66	**0,53**	**0,49**	0,66	0,51	0,44	0	**2,78**	**10,26**
20	0,67	**0,54**	**0,50**	0,67	0,53	0,45	0	**2,70**	**10,00**
21	0,68	**0,55**	**0,51**	0,68	0,54	0,47	0	**2,63**	**9,76**
22	**0,67**	**0,56**	**0,49**	0,65	0,55	0,48	3,03	**2,56**	**2,22**
23	0,66	**0,58**	**0,50**	0,66	0,56	0,49	0	**2,50**	**2,17**
24	0,67	**0,59**	**0,51**	0,67	0,57	0,50	0	**2,44**	**2,13**
25	0,68	**0,60**	0,51	0,68	0,58	0,51	0	**2,38**	0
26	0,68	**0,60**	0,52	0,68	0,59	0,52	0	**2,33**	0
27	0,69	**0,61**	0,53	0,69	0,60	0,53	0	**2,27**	0
28	0,70	**0,62**	0,54	0,70	0,61	0,54	0	**2,22**	0
29	0,71	**0,63**	0,55	0,71	0,62	0,55	0	**2,17**	0
30	0,71	**0,64**	0,56	0,71	0,63	0,56	0	**2,13**	0
31	0,72	**0,65**	0,56	0,72	0,63	0,56	0	**2,08**	0
32	0,73	**0,65**	0,57	0,73	0,64	0,57	0	**2,04**	0

Таким образом, можно выделить циклические помехоустойчивые коды, обладающие лучшей скоростью и эффективностью по сравнению с кодами БЧХ для $t = 2$ при $m = 2$, $m \in [8,12]$, $m = 22$; для $t = 3$ при $m \in [6,12]$, m = 17, m = 18; для $t = 4$ при $m = 4$, $m = 5$, $m \in [12,24]$. Данные результаты говорят о том, что

образующие полиномы из таблицы 3.2 можно использовать при проектировании устройств декодирования [93, 94] с более короткими и эффективными кодами.

Одним из найденных образующих полиномов для построения неукороченных циклических помехоустойчивых кодов является полином $G(0,1) = 100111001$ для параметров $m = 9$, $t = 2$. Особенность данного полинома заключается в том, что его длина равна длине образующего полинома для кода БЧХ (15, 7, 5). Это позволяет построить циклический помехоустойчивый код (17, 9, 5) с длиной кодового слова $n = 17$ и длиной информационного блока $m = 9$, т.е. кодировать на 2 бита информации больше при сохранении избыточности кода. Данный факт является важным при проектировании устройств исправления ошибок, так как позволяет передать больше информации с той же избыточностью по каналам связи, а также уменьшить аппаратные затраты декодирующего устройства. Далее в работе (раздел 3.6) будут рассмотрены реализованные устройства на ПЛИС с применением различных алгоритмов декодирования циклического помехоустойчивого кода (17, 9, 5) как более эффективные альтернативы устройствам с кодом БЧХ (15, 7, 5) и укороченным кодом БЧХ (19, 9, 5).

3.3. Алгоритм поиска образующих полиномов для кодов, исправляющих пакетные ошибки

Пакетные ошибки в кодовых словах представляют собой группу искаженных смежных бит длины p и являются частными случаем независимых ошибок. Для построения циклических помехоустойчивых кодов, исправляющих только пакетные ошибки, требуется меньшая избыточность, чем для исправления независимых ошибок. Существуют специальные коды Рида-Соломона [43, 50], ориентированные на исправление только пакетных ошибок, при этом избыточность данных кодов меньше, чем БЧХ кодов, ориентированных на исправление независимых ошибок.

Разработанный алгоритм поиска образующих полиномов для пакетных ошибок позволяет находить полиномы для построения кодов, аналогичных кодам Рида-Соломона малой длины (до 32 одноразрядных информационных символов). В целом алгоритм поиска полинома для пакетных ошибок аналогичен поиску для независимых ошибок, за исключением некоторых изменений [95, 96]. Далее приведено пошаговое описание алгоритма.

Начало.

Шаг 1. Задаются m – разрядность информационного блока и p – длина исправляемого пакета в битах.

Шаг 2. Вычисляется длина контрольного блока по формуле

$$k = \begin{cases} n + 1, p = 1 \\ 2*n + 1, p = 2 \\ \log_2(2*n + 1 + \sum_{i=1}^{p-2} n * 2^i), \ p > 2, \end{cases}$$

где n – разрядность кодового слова.

Шаг 3. Устанавливается начальное значение веса полинома $w = p$. Минимальное расстояние кода $d_{min} = p$.

Шаг 4. Генерируются полиномы со старшей степенью, равной k и весом, равным w.

Шаг 5. Если хотя бы один из полиномов $x^n + 1$ делится на образующий полином без остатка, где n изменяется от $m + k$ до $2^{\lfloor \log_2(m+k) \rfloor + 1} - 1$, то переходим на шаг 6, иначе переходим на шаг 12.

Шаг 6. По полиному, удовлетворяющему условиям в шаге 5, строятся все возможные кодовые слова с длиной информационного блока от 1 до m.

Шаг 7. Вычисляется минимальное кодовое расстояние каждого построенного кодового слова.

Шаг 8. Если вычисленное кодовое расстояние для всех слов $d \geq d_{min}$, то переходим на шаг 9, иначе переходим на шаг 12.

Шаг 9. Вычисляются синдромы ошибок путем деления всех комбинаций пакетных ошибок на образующий полином.

Шаг 10. Если все синдромы ошибок уникальны, то алгоритм завершает работу, иначе переходим на шаг 12.

Шаг 12. Если вес *w* равен *k*, увеличиваем *k* на единицу и переходим на шаг 4, иначе увеличиваем вес *w* и переходим на шаг 4.

Конец.

Для составления таблицы образующих полиномов проведен компьютерный эксперимент, аналогичный поиску полиномов для независимых ошибок.

Постановка компьютерного эксперимента

Эксперимент по поиску образующих полиномов проведён для разных значений длин информационного блока $m \in [1; 32]$ и длины пакета, исправляемых ошибок $p \in [2; 6]$. Для ускорения процедуры поиска применена технология параллельных вычислений OpenMP по аналогии с алгоритмом поиска полиномов для независимых ошибок.

В таблице 3.7 представлены найденные образующие полиномы для кодов, исправляющих пакетные ошибки длиной от 2 до 6 бит.

Таблица 3.7 – Образующие полиномы (пакетные ошибки)

m / t	2				
	2	3	4	5	6
2	10101	-	-	-	-
3	11101	1001001	-	-	-
4	110101	11010001	10100110111	-	-
5	1101011	101010001	10100110111	10100110111	-
6	1110111	11001111	101010001	1111100100101	100111011100011
7	1111001	11110011	100010111	1101010001111	100111011100011
8	1000101	11010001	10001000101	1100101001101	10101111010111
9	1111001	1001111	10011000011	1100101001101	1100101001101
10	110101	1111011101	10011010101	111110010001001	1110111001000011

Окончание таблицы 3.7

11	1101111	1110011101	11011000011	1111100100101	1100001001110111
12	1000111	1001000001	1111011101	101011100011	1101000011011101
13	1100101	101001011	11101101001	111101110100011	1110111001000011
14	1101111	10011111	101000101011	111110010001001	1110111001000011
15	1101111	1011100111	101101001001	110001011101111	1110111001000011
16	1001101	11011000111	101010011111	111110010001001	1010100101000011
17	111101101	11010000011	101101001001	1001110100001001	1000110110101101
18	1101111	1110011101	100100101101	110100010001011	110110111001001
19	1111011	1011100111	100100101101	11111101001011	110110111001001
20	1001101	1111000011	10100010001	100001100111	1111010101110011
21	11011101	1100001111	11000011001	11100010010001	110110111001001
22	101000001	11000100011	111010001011	11111101001011	110110111001001
23	1100000101	1101011111	111010001011	11111101001011	1100100010011001
24	100100111	1101011111	111010001011	11111101001011	1100100010011001
25	1101111	1001011011	10101110101	11100010010001	1011110001110111
26	10100011	1111101011	110100010111	11100010010001	11100010010001
27	10110111	110101101	111001101011	11010010111111	10000100101110011
28	10010101	1001011011	110100010111	110000110100011	11000100101000111
29	11101001	1001011011	111011110101	11010010111111	10110100001101111
30	10100011	110110001	1100001000101	110000110100011	10110100001101111
31	10100011	1110010001	101110110011	110000110100011	10100001110111011
32	10100011	1101001101	100100100011	101101111101101	10100101011111001

С помощью образующих полиномов, представленных в таблице 3.7, можно строить циклические помехоустойчивые коды, ориентированные на исправление пакетных ошибок. Избыточность таких кодов будет заметно ниже, чем у кодов, исправляющих независимые ошибки. В разделе 3.7 и главе 4 приведены реализации на ПЛИС устройства исправления пакетных ошибок с применением циклического помехоустойчивого кода (15, 8, 3), построенного на основе образующего полинома из таблицы 3.7 (11010001).

3.4. Алгоритмы исправления пакетных и независимых ошибок

Существуют различные алгоритмы исправления ошибок, применяемые на практике, такие как Берлекэмпа-Месси [43, 53–56], Евклида [43], Питерсона-Горленштейна-Цирлера [55].

Также существуют алгоритмы, описание которых можно встретить в литературе [50], но практических реализаций и применения они не нашли. Однако преимущество этих алгоритмов заключается в простоте реализации и

отсутствии необходимости решать уравнения, что позволяет значительно упростить разработку устройств исправления ошибок.

3.4.1. Табличный алгоритм

Описание табличного алгоритма декодирования можно встретить в литературе [50]. Алгоритм заключается в формировании таблицы предвычисленных шаблонов ошибок, при этом адресами шаблонов являются синдромы (остатки от деления шаблона на образующий полином) ошибок.

Пошагово данный алгоритм можно описать следующим образом:

Начало.

Шаг 1. Вычисляется остаток от деления кодового слова с ошибкой на образующий полином.

Шаг 2. По полученному остатку выбирается значение шаблона ошибки из таблицы.

Шаг 3. Выбранный шаблон ошибки складывается по модулю 2 с кодовым словом. На выходе декодера получается исправленное кодовое слово (при необходимости контрольные разряды можно отсечь).

Конец.

Реализация такого алгоритма декодирования обладает высоким быстродействием, а также не требует цикличности кода. Возможность исправлять ошибки в кодах, не обладающих свойством цикличности, позволяет применять для декодера более короткие образующие полиномы, представленные в таблице 3.1. Недостатком такого алгоритма является значительный рост требуемого объёма памяти для хранения таблицы при увеличении длины кодового слова ($V = n \cdot 2^k$ бит, где k – длина контрольного блока, n – длина кодового слова). При этом хранить можно не всё кодовое слово, а только информационный блок, что позволит сократить требуемый объем памяти до $m \cdot 2^k$ бит.

3.4.2. Циклический алгоритм для исправления независимых ошибок

Описание циклического алгоритма декодирования можно встретить в литературе [50]. Основная идея заключается в итерационном процессе деления кодового слова с ошибкой на образующий полином и определения веса остатка. Циклический алгоритм декодирования можно описать следующим образом.

Начало.

Шаг 1. Вычисляется остаток от деления по модулю 2 кодового слова на образующий полином. Если остаток нулевой, то кодовое слово принято без ошибок и алгоритм завершает работу. Иначе переход на шаг 2.

Шаг 2. Вычисляется вес остатка. Если вес остатка $w \geq t$, где t – количество исправляемых кодом ошибок, то кодовое слово циклически сдвигается на 1 разряд вправо и осуществляется переход на шаг 4. Иначе перехода на шаг 3.

Шаг 3. Кодовое слово складывается по модулю 2 с остатком от деления. Результат сдвигается циклически влево на количество разрядов, равное количеству предшествующих сдвигов вправо. Алгоритм завершает работу.

Шаг 4. Если достигли максимального количества сдвигов кодового слова $n - 1$, где n – длина кодового слова, и не нашли подходящего остатка (условия описаны в шаге 2), то алгоритм завершает работу с флагом обнаружения неисправимой ошибки. Иначе переход на шаг 1.

Конец.

Недостатком циклического алгоритма декодирования является невозможность исправлять ошибки для кодов со скоростью > 0,5, т.е. в которых длина информационного блока больше длины контрольного блока, например для кода (17, 9, 5). Это связано с понятием *скользящее окно* – область кодового слова, в которой ошибки заданной кратности будут гарантировано исправляться. Также при программной реализации на микроконтроллерах быстродействие алгоритма зависит от позиций в кодовом слове: чем ближе

ошибки к старшим разрядам (при циклическом сдвиге вправо), тем больше сдвигов необходимо осуществить для исправления.

Циклический алгоритм является достаточно простым в реализации и в отличие от известных популярных методов и алгоритмов [43, 52–56] не требует вычислений в полях Галуа, решения системы ключевых уравнений (алгоритм БМА [52, 53]) и полного перебора корней многочлена локаторов ошибок (поиск Ченя [43]). Данный алгоритм при соответствующих модификациях позволяет не менее эффективно исправлять ошибки, чем другие методы и алгоритмы.

3.4.3. Модифицированный циклический алгоритм для исправления независимых ошибок

Для устранения недостатка циклического алгоритма декодирования, связанного с зависимостью от скорости кода, необходимо подробнее рассмотреть понятие *скользящее окно*, введенное в разделе 3.4.2.

Для *циклического алгоритма декодирования* длина *скользящего окна* равна длине контрольного блока (k). Соответственно, если длина сообщения $m > k$, то циклический метод декодирования будет работать не для всех возможных ошибок при кратности $t > 1$, так как при такой кратности существуют шаблоны ошибок, которые не покрывают диапазон скользящего окна. Заметим, что табличный алгоритм декодирования данным недостатком не обладает, так как он не использует *скользящее окно* и содержит всевозможные синдромы ошибок кратности t. Используя данную особенность, дополним циклический алгоритм декодирования частью табличного таким образом, чтобы те шаблоны ошибок, которые циклический алгоритм не покрывает, исправлялись табличным алгоритмом. Для этого необходимо из полной таблицы выбрать подмножество шаблонов, выходящих из поля зрения «скользящего окна». В таком случае получим комбинацию циклического и табличного алгоритмов декодирования, где большая часть ошибок будет исправляться циклическим алгоритмом, а для случаев выхода ошибки за окно, применяться обращение к «сокращенной» таблице. Таблица 3.8 является

примером сокращенной таблицы для модифицированного циклического алгоритма декодирования кода (17,9,5).

Таблица 3.8 – Шаблоны двукратной независимой ошибки для (17,9,5) вне скользящего окна

Синдром	**Значение (шаблон ошибки)**
00111000	**000000001**00000001
01110000	**000000010**00000010
11100000	**000000100**00000100
11111001	**000001000**00001000
11001011	**000010000**00010000
10101111	**000100000**00100000
01100111	**001000000**01000000
11001110	**010000000**10000000
10100101	**100000001**00000000
01110011	**000000010**00000001
11100110	**000000100**00000010
11110101	**000001000**00000100
11010011	**000010000**00001000
10011111	**000100000**00010000
00000111	**001000000**00100000
00001110	**010000000**01000000
00011100	**100000000**10000000

Пошагово модифицированный циклический алгоритм представим в следующем виде [97, 98].

Начало.

Шаг 1. Вычисляется остаток от деления по модулю 2 кодового слова на образующий полином. Если остаток нулевой, то кодовое слово принято без ошибок и алгоритм завершает работу. Иначе переход на шаг 2.

Шаг 2. Если вес остатка $w \leq t$, то кодовое слово суммируется по модулю 2 с остатком от деления. Результат сдвигается циклически влево на количество разрядов, равное количеству предшествующих сдвигов вправо и алгоритм завершает работу. Иначе переход на шаг 3.

Шаг 3.

a. Проверить остаток на совпадение с *первым* синдромом двукратной ошибки, выходящей за диапазон скользящего окна (первая строка сокращённой таблицы).

b. Ели совпадение есть, то кодовое слово суммируется по модулю 2 с остатком от деления. Результат сдвигается циклически влево на количество разрядов, равное количеству предшествующих сдвигов вправо и алгоритм завершает работу. Иначе кодовое слово циклически сдвигается на 1 разряд вправо и осуществляется переход на шаг 4.

Шаг 4. Если достигли максимального количества сдвигов кодового слова $n - 1$, где n – длина кодового слова, и не нашли подходящего остатка (условия описаны в шаге 2), то алгоритм завершает работу с флагом обнаружения неисправимой ошибки. Иначе переход на шаг 1.

Конец.

В описанном алгоритме проверка остатка на совпадение с сокращённой таблицей выполняется на каждой итерации, что наиболее удобно для реализации на ПЛИС, так как все циклические сдвиги кодового слова можно проверять параллельно.

3.4.4. Циклический алгоритм для исправления пакетных ошибок

Разработанный циклический алгоритм декодирования для исправления пакетных ошибок аналогичен алгоритму для независимых, за исключением одного этапа – проверки весов остатков от деления кодового слова на образующий полином. Циклический алгоритм декодирования для пакетных ошибок можно описать следующим образом [99 – 101].

Начало.

Шаг 1. Вычисляется остаток от деления по модулю 2 кодового слова на образующий полином. Если остаток нулевой, то кодовое слово принято без ошибок и алгоритм завершает работу. Иначе переход на шаг 2.

Шаг 2. Остаток от деления кодового слова на образующий полином проверяется на совпадение с одним из шаблонов ошибки длины p. Если

совпадений нет, то кодовое слово циклически сдвигается на 1 разряд вправо и осуществляется переход на шаг 4. Иначе переход на шаг 3.

Шаг 3. Кодовое слово суммируется по модулю 2 с остатком от деления. Результат сдвигается циклически влево на количество разрядов, равное количеству предшествующих сдвигов вправо. Алгоритм завершает работу.

Шаг 4. Если достигли максимального количества сдвигов кодового слова $n - 1$, где n – длина кодового слова, и не нашли подходящего остатка (условия описаны в шаге 2), то алгоритм завершает работу с флагом обнаружения неисправимой ошибки. Иначе переход на шаг 1.

Конец.

3.5. Программная реализация алгоритмов для микропроцессорного устройства

В автоматизированных системах управления на основе микропроцессорных устройств важным фактором является обеспечение целостности и достоверность передаваемых команд и данных. Искажение хотя бы одного бита информации может значительно изменить ход эксперимента, что приведёт к непредсказуемым последствиям. Обычно, в подобных системах для контроля целостности применяются контрольные суммы CRC, обнаруживающие ошибки (см. глава 2). Однако коды CRC не позволяют исправлять ошибки и для получения достоверных данных (команд) необходима их повторная передача до тех пор, пока не будет получен пакет без ошибок. Для обеспечения надёжной передачи команд и данных без повторного запроса требуется применение помехоустойчивых кодов, а также соответствующих программ и устройств кодирования и декодирования таких кодов. Известно множество реализаций декодеров помехоустойчивых кодов на ПЛИС, однако на нижнем уровне систем сбора данных и управления технологическими процессами применяются недорогие микропроцессоры с дефицитом ресурсов. Это требует реализации простых программ декодирования с наименьшими аппаратными затратами.

Для оценки эффективности циклического и табличного алгоритмов декодирования осуществлены их программные реализации на микроконтроллере ATmega8515 [102].

3.5.1. Табличный алгоритм

При программной реализации на микропроцессорных вычислительных устройствах табличный алгоритм является наиболее быстродействующим, однако требует для реализации объём памяти данных для хранения предвычисленных констант в 2 раза превышающий объём памяти программ для хранения исходного кода. На рисунке 3.6 приведена разработанная блок-схема программной реализации на микроконтроллере ATmega8515 табличного алгоритма исправления двукратных независимых ошибок.

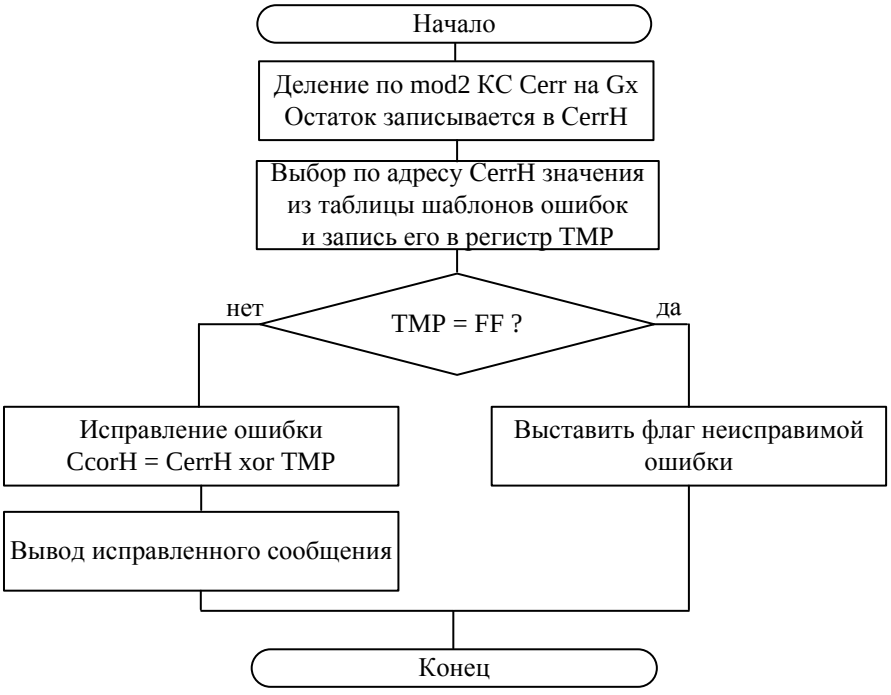


Рисунок 3.6 – Схема реализации табличного алгоритма исправления независимых ошибок

Кодовое слово с ошибкой *Cerr* длиной *n* бит поступает на входы 8-разрядного МК и записывается в регистры *CerrH* и *CerrL*. В результате деления по mod2 *Cerr* на образующий полином *Gx* остаток записывается в *CerrH*. По адресу *CerrH* из таблицы шаблонов ошибок выбирается значение (запись в *TMP*) и, если это значение не сигнализирует о неисправимой ошибке,

осуществляется сложение по mod2 шаблона *TMP* с *CerrH*. В регистре *CcorH* содержится исправленное сообщение. Деление по mod2 осуществляется по классическому алгоритму.

В таблице 3.9 приведено быстродействие (в тактах) программной реализации табличного алгоритма для кода БЧХ (15, 7, 5), циклического помехоустойчивого кода (17, 9, 5) и (15, 8, 3) при частоте 16 МГц.

Таблица 3.9 – Быстродействие программной реализации табличного алгоритма, такты

Код БЧХ (15, 7, 5)	Циклический помехоустойчивый код (17, 9, 5)	Циклический помехоустойчивый код (15, 8, 3)
84–87	90	81

3.5.2. Циклический алгоритм

При программной реализации циклический алгоритм исправления независимых ошибок позволяет не использовать таблицу для хранения предвычисленных значений, однако требует больше тактов для исправления ошибок. На рисунке 3.7 приведена разработанная блок-схема программной реализации циклического алгоритма декодирования для микроконтроллера ATmega8515 [102] на примере кода БЧХ (15, 7, 5).

В таблице 3.10 приведены аппаратные средства, необходимые для реализации табличного и циклического алгоритмов декодирования на микроконтроллере.

Таблица 3.10 – Аппаратные затраты алгоритмов

Алгоритм	**Размер кода (байт)**	**Число регистров**	**Объём ПЗУ (байт)**
код БЧХ (15,7,5)			
Табличный	126	14	256
Циклический	164	13	0
ЦПК (17,9,5)			
Табличный	164	16	512
Циклический	236	16	0
ЦПК (15,8,3)			

Окончание таблицы 3.10

Табличный	120	13	128
Циклический	214	11	0

Как видно из таблицы 3.10 программная реализация табличного алгоритма требует наименьший объем памяти программ среди всех рассмотренных помехоустойчивых кодов, однако объёмы ПЗУ для хранения констант в 2-3 раза больше объёма памяти программ. Наименее эффективной по аппаратным затратам является реализация циклического алгоритма декодирования для помехоустойчивого кода (15, 8, 3), исправляющего пакетные ошибки.

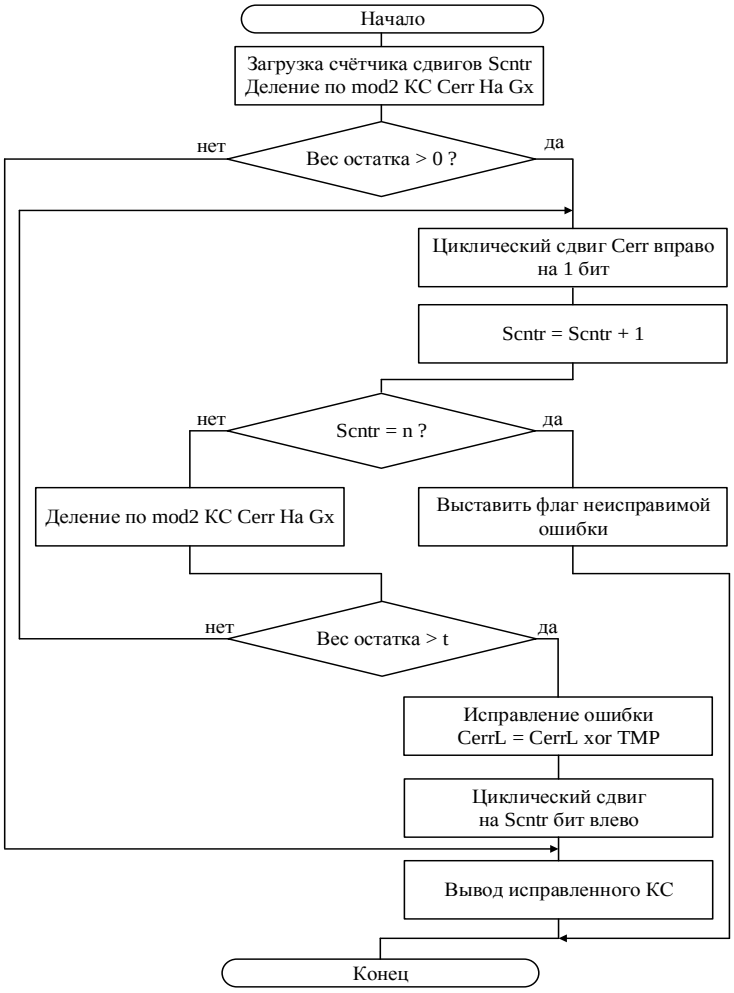


Рисунок 3.7 – Схема программной реализации циклического алгоритма исправления независимых ошибок

Объем памяти программ для циклического алгоритма превышает в 1,5 раза объем памяти данных для табличного алгоритма, при этом общие затраты

памяти для циклического алгоритма (214 байт) на 14 % меньше, чем для табличного. Реализация циклического алгоритма для кода БЧХ (15, 7, 5) и ЦПК (17, 9, 5) является достаточно эффективной по аппаратным затратам (164 байта против 382 и 236 байт против 676 байт соответственно).

Для исправления пакетных ошибок схема циклического алгоритма будет аналогична схеме на рисунке 3.7, за исключением проверки веса остатка. В циклическом алгоритме для пакетных ошибок остаток от деления проверяется на соответствие одному из шаблонов пакетной ошибки (рисунок 3.8).

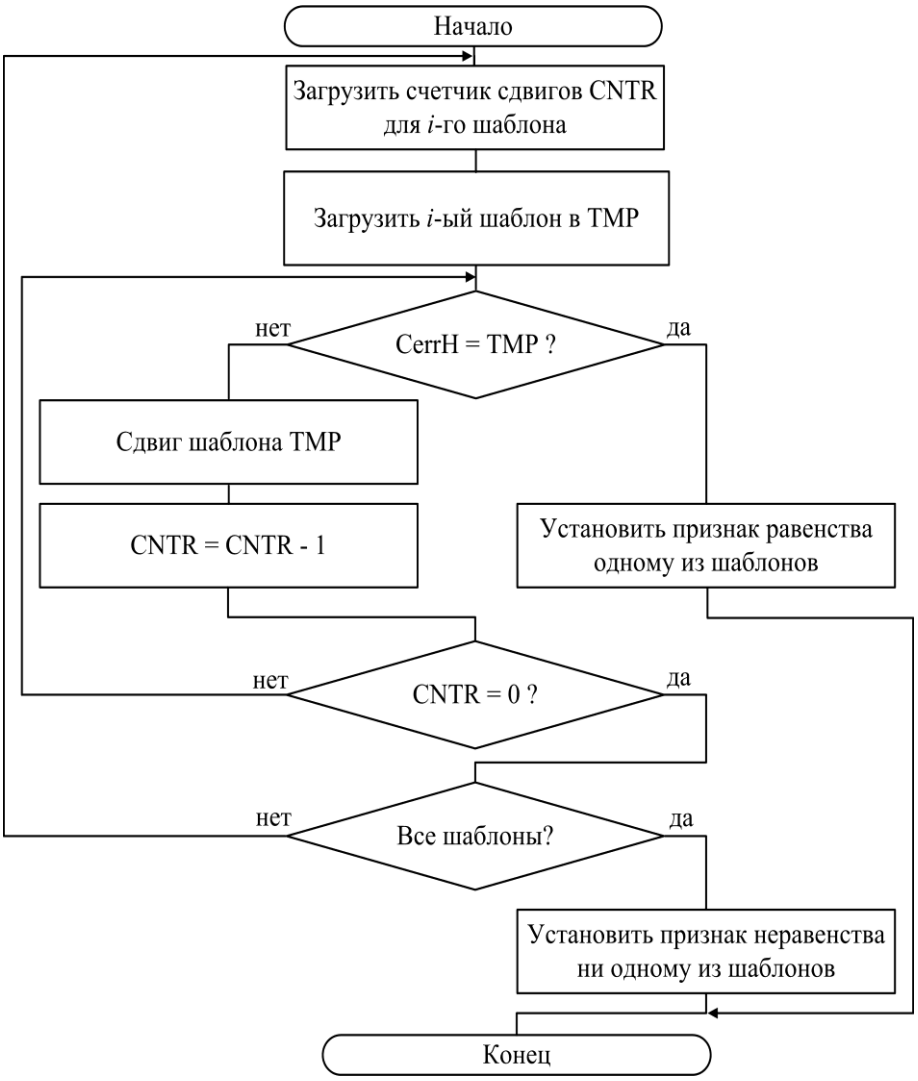


Рисунок 3.8 – Схема алгоритма проверки остатка на совпадение с шаблоном ошибки

В таблице 3.11 представлено быстродействие программных реализаций циклического и табличного алгоритмов декодирования на МК. Для ЦПК,

исправляющих двукратные независимые ошибки, результаты эксперимента приведены для всех позиций двукратной ошибки. Для ЦПК, исправляющего трехкратные пакетные ошибки, результаты эксперимента приведены для всех позиций пакета ошибки длиной 3 бита.

Таблица 3.11 – Быстродействие алгоритмов декодирования на микроконтроллере

Алгоритм	**Наилучшее быстродействие, такты МК**	**Наихудшее быстродействие, такты МК**	**Среднее кол-во тактов МК**
код БЧХ (15, 7, 5)			
Табличный	83	87	86
Циклический	181	1836	720
ЦПК (17, 9, 5)			
Табличный	90	90	90
Циклический	176	2079	898
ЦПК (15, 8, 3)			
Табличный	81	81	81
Циклический	167	2608	1067

При программной реализации на микроконтроллере циклического алгоритма декодирования его быстродействие зависит от позиций ошибок в кодовом слове. Наилучшее быстродействие для кода БЧХ (15, 7, 5) наблюдается при шаблоне 0x0180h (110000000b, 181 такт МК с частотой 16 МГц), а наихудшее при шаблоне 0x0081h (10000001b, 1836 тактов МК с частотой 16 МГц). В среднем быстродействие циклического алгоритма декодирования для кода БЧХ (15, 7, 5) составляет 720 тактов МК с частотой 16 МГц. При реализации табличного алгоритма среднее быстродействие составляет 86 тактов МК.

Для ЦПК (17, 9, 5) среднее быстродействие (898 тактов) ниже, чем для кода БЧХ (15, 7, 5), что связано с увеличенной длиной кодового слова. Для исправления пакетной ошибки длиной 3 бита потребуется в среднем 1067 тактов МК с частотой 16 МГц. Это связано с применением более медленной процедуры проверки остатка от деления на соответствие шаблону пакетной ошибки по сравнению с процедурой оценки веса остатка.

В целом реализации циклического алгоритма декодирования отстают по быстродействию от реализации табличных в среднем в 8-13 раз, что обусловлено более простыми операциями выбора предвычисленных значений по сравнению с многократными циклическими сдвигами. При наличии соответствующих объемов доступной памяти (128 байт для хранения пакетных ошибок, 256 байт для кода БЧХ (15, 7, 5), 512 байт для ЦПК (17, 9, 5)) табличный алгоритм декодирования является наиболее приемлемым при реализации на микроконтроллерах. Циклический алгоритм декодирования можно применять как альтернативу табличному алгоритму при отсутствии возможности хранить предвычисленные значения в памяти типа ROM, PROM, RAM.

3.6. Разработка устройств исправления независимых ошибок на ПЛИС

В разделе представлено исследование реализаций устройств исправления независимых ошибок на ПЛИС с применением известных [43, 103 – 107] и предложенных [97, 98, 117, 118] помехоустойчивых кодов на основе табличного и циклического алгоритмов декодирования [50]. Результаты исследования позволят сравнить быстродействие и аппаратные затраты (количество логических элементов и требуемый объем памяти) устройств с различными помехоустойчивыми кодами.

3.6.1. Разработка устройств на ПЛИС для исправления ошибок с применением кода БЧХ (15,7,5)

Табличный алгоритм декодирования

Помехоустойчивый циклический БЧХ код (15, 7, 5) имеет длину n = 15 бит, из которых m = 7 бит – информационное сообщение и k = 8 бит – контрольный блок. Данный код с расстоянием Хэмминга d = 5 позволяет исправлять 2 независимые ошибки. Скорость кода составляет 0,466. Образующий полином данного кода $X^8 + X^7 + X^6 + X^4 + 1$ (111010001).

На рисунке 3.9 приведена общая структурная схема декодера помехоустойчивого кода с применением табличного алгоритма декодирования [50].

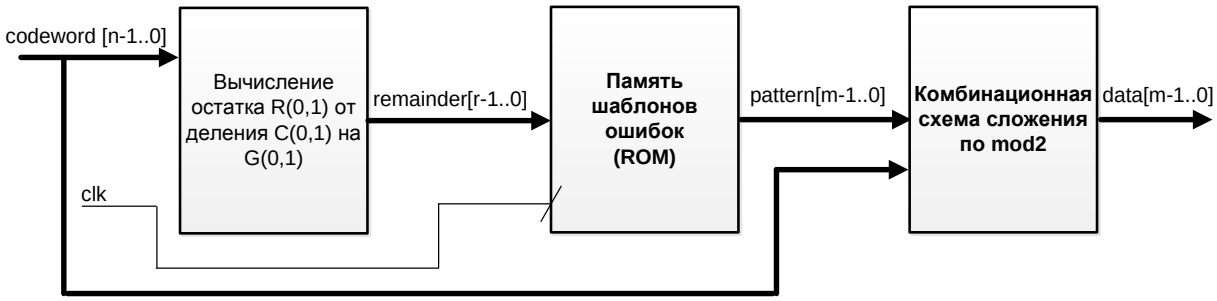


Рисунок 3.9 – Структурная схема декодера с табличным алгоритмом

На вход декодера поступает кодовое слово *codeword*. В блоке (модуле) remainder вычисляется остаток от деления кодового слова на образующий полином. Выбранный по адресу *remainder* шаблон *pattern* поступает на схему сложения по модулю 2 шаблона со старшими разрядами кодового слова (исправляются только информационные разряды).

Для реализации декодера БЧХ кода (15, 7, 5) с применением табличного метода необходимо составить таблицу шаблонов двукратной независимой ошибки, где в качестве адреса задаётся синдром ошибки (приложение Б, таблица Б.1).

На рисунке 3.10 приведена схема модулей декодера БЧХ кода (15, 7, 5). В модуле reminder осуществляется вычисление адреса памяти, реализованной в модуле decoder_final.

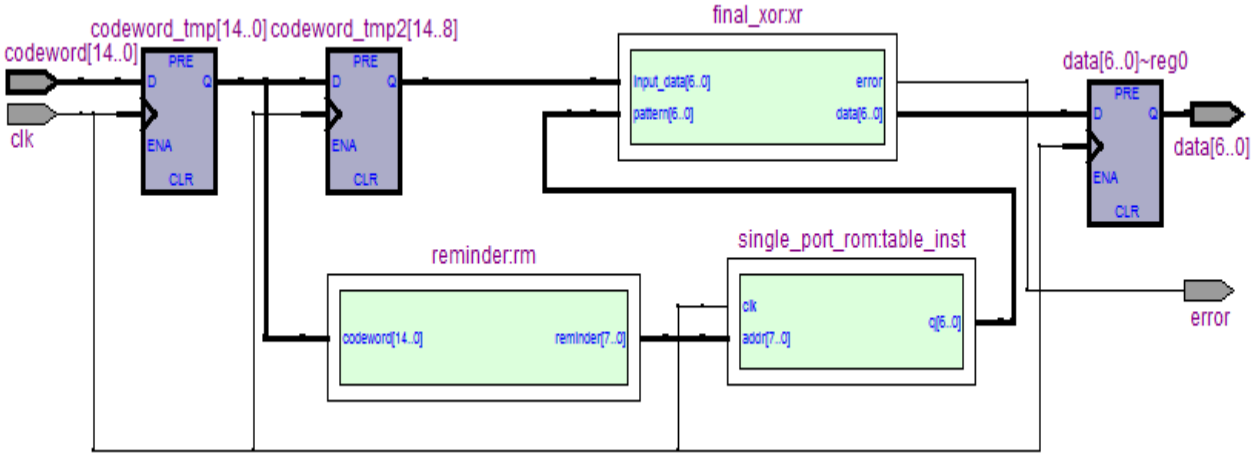


Рисунок 3.10 – RTL-view декодера БЧХ кода (15, 7, 5)

На рисунке 3.11 приведена комбинационная схема вычисления остатка от деления по матричному алгоритму [108].

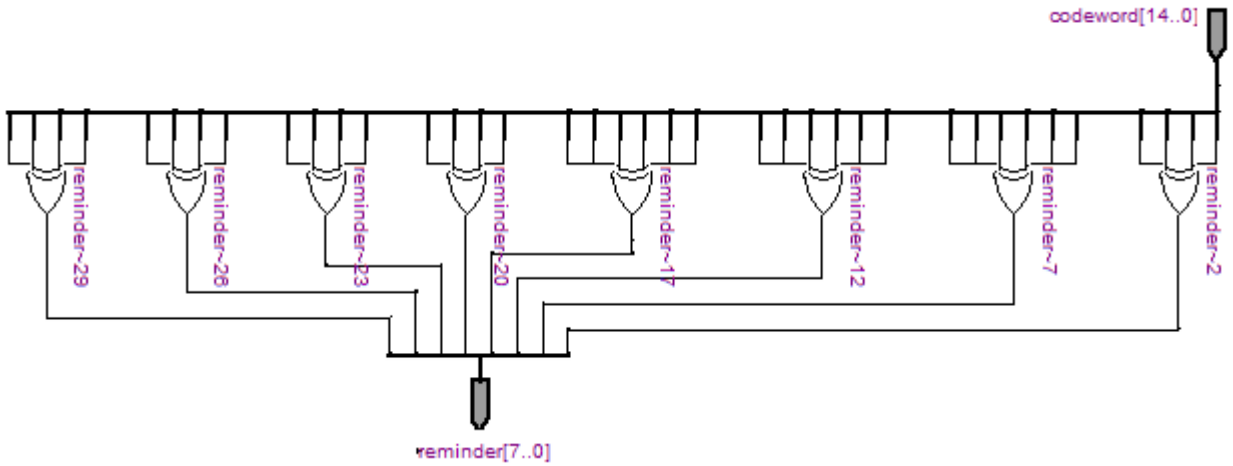


Рисунок 3.11 – Комбинационная схема деления декодера БЧХ кода (15, 7, 5)

На рисунке 3.12 приведен RTL-view схемы исправления ошибки декодера БЧХ (15, 7, 5).

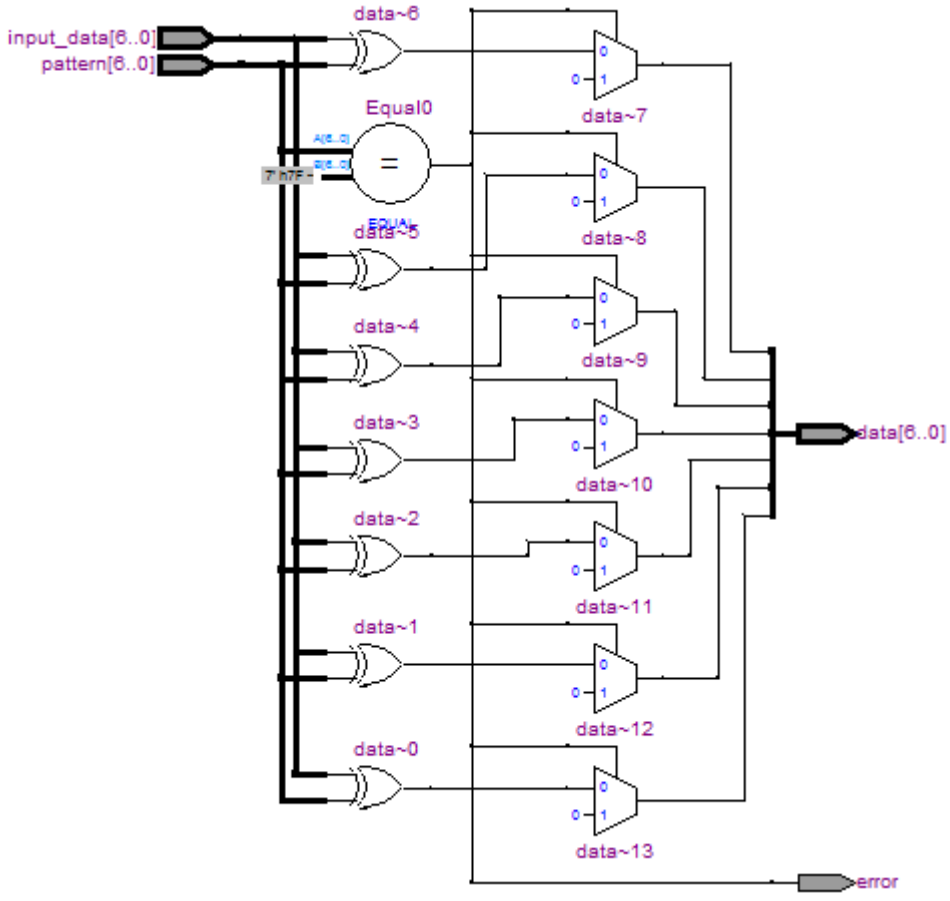


Рисунок 3.12 – Схема исправления ошибки кода БЧХ (15, 7, 5)

Архитектура устройства с табличным алгоритмом декодирования представляет собой трехэтапный конвейер (рисунок 3.13). Время декодирования одного кодового слова, поступающего на вход устройства, составляет 3 такта синхросигнала ПЛИС (*T1–T3*). После декодирования первого кодового слова (заполнения конвейера) каждое последующее декодируется за один такт *T4* и *T5*. Таким образом, максимальное время декодирования *n* кодовых слов составляет 4 такта.

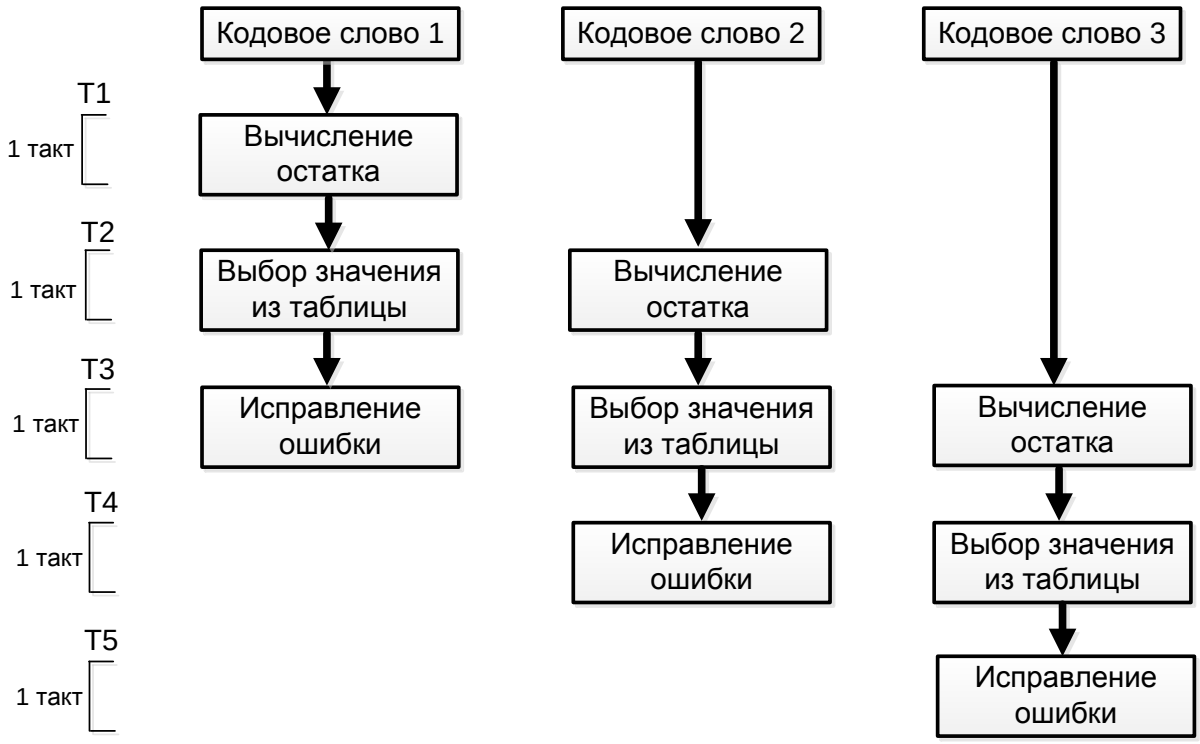


Рисунок 3.13 – Конвейер, реализующий табличный алгоритм декодирования

На рисунок 3.14 представлена диаграмма моделирования работы декодера с применением табличного алгоритма декодирования и конвейерной архитектурой.

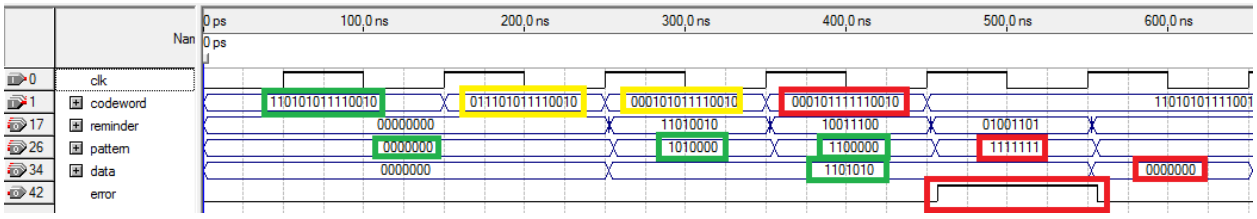


Рисунок 3.14 – Моделирование работы декодера БЧХ кода (15, 7, 5). Табличный алгоритм

На временной диаграмме приведены кодовое слово без ошибки, 2 варианта кодовых слов с двукратной ошибкой (шаблоны 101000000000000 и 110000000000000) и кодовое слово с трёхкратной ошибкой (шаблон 110000100000000). Из диаграммы видно, что остатки от деления (3 строка) для двукратных ошибок соответствуют искаженным битам кодового слова. При неисправимой ошибке формируется единичное значение сигнала *error*.

На рисунке 3.15 приведены характеристики декодера из САПР Quartus II для ПЛИС Stratix III EP3SL70F780C2.

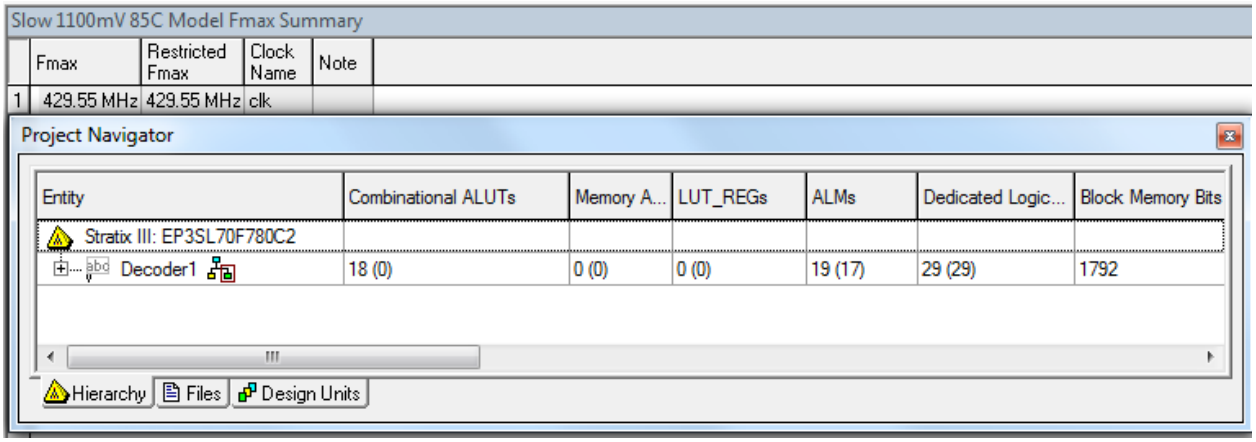


Рисунок 3.15 – Характеристики декодера БЧХ кода (15, 7, 5). Табличный алгоритм

Для реализации декодера БЧХ (15,7,5) табличным алгоритмом требуется 19 ячеек логических (ALMs) и 1792 бита запоминающего устройства. Максимальная частота декодера составляет 429 MHz, что позволяет декодировать первое кодовое слово за 6,9 нс и каждое последующее – за 2,3 нс.

Циклический алгоритм декодирования

Циклический алгоритм декодирования на ПЛИС может быть реализован в двух вариантах: классический (последовательный) и параллельный [103–107].

Особенностью циклического алгоритма декодирования является возможность параллельного декодирования при реализации на ПЛИС. Процедуры деления и проверки весов остатков можно осуществлять для всех сдвигов кодового слова одновременно, что позволяет осуществлять декодирование за 1–2 такта. Классический вариант подразумевает побитовый

сдвиг кодового слова на каждом такте устройства, что наиболее актуально при реализации устройства декодирования на микроконтроллерах. При реализации на ПЛИС классический вариант циклического алгоритма позволяет уменьшить количество логических ячеек и увеличить максимальную частоту устройства. Однако, при этом увеличится количество тактов, необходимых для исправления заданного количества ошибок.

Классическая реализация алгоритма на ПЛИС

Структурная схема [105, 106] классической реализации циклического алгоритма декодирования приведена на рисунке 3.16.

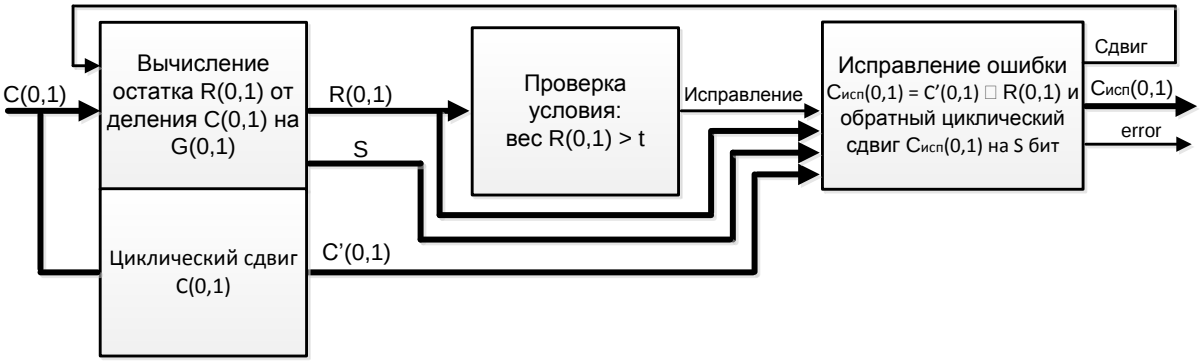


Рисунок 3.16 – Структурная схема декодера БЧХ кода (15,7,5). Классическая реализация циклического алгоритма

Кодовое слово *C(0,1)* подаётся на блок вычисления остатка от деления *R(0,1)* кодового слова на образующий полином *G(0,1)*, при этом остаток вычисляется матричным алгоритмом деления полиномов [108]. Полученный остаток подаётся на блок проверки веса остатка от деления. Если вес остатка от деления больше кратности исправляемых ошибок *t*, то формируется сигнал для сдвига кодового слова и повторения операций деления с вычислением веса остатка, иначе формируется управляющий сигнал для блока исправления ошибки с обратным циклическим сдвигом результата на число разрядов *s*, равное количеству сдвинутых позиций. На выход декодера поступает исправленное кодовое слово *C'(0,1)*, либо сигнал неисправимой ошибки *error*.

На рисунке 3.17 приведен RTL-view декодера БЧХ-кода (15, 7, 5), разработанного по структурной схеме рисунка 3.14.

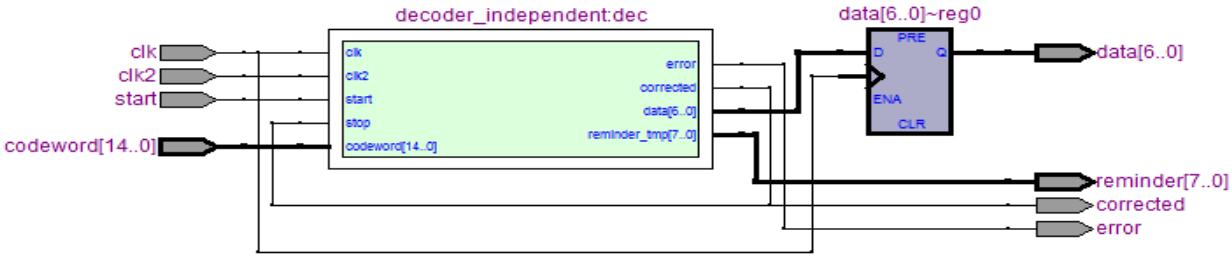


Рисунок 3.17 – RTL-view декодера БЧХ кода (15, 7, 5). Классическая реализация циклического алгоритма

На рисунке 3.18 приведен декодер БЧХ кода (15, 7, 5), состоящий из модулей *receiver*, *reminder*, *check_weight* и *decoder_final*.

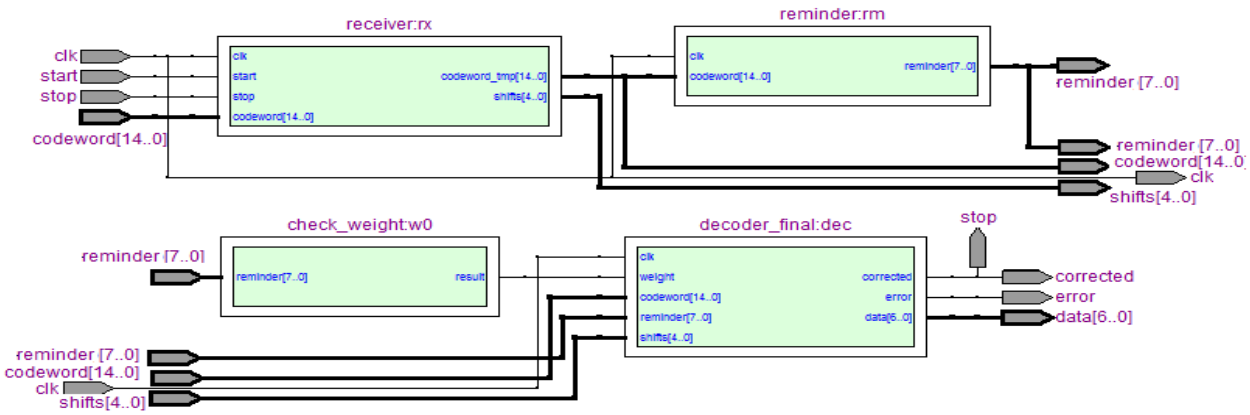


Рисунок 3.18 – RTL-view модулей приёма, вычисления остатка, вычисления веса и исправления ошибки

Кодовое слово принимается в модуле приёма и сдвига *receiver* (рисунок 3.19) и поступает в модуль вычисления остатка *reminder*.

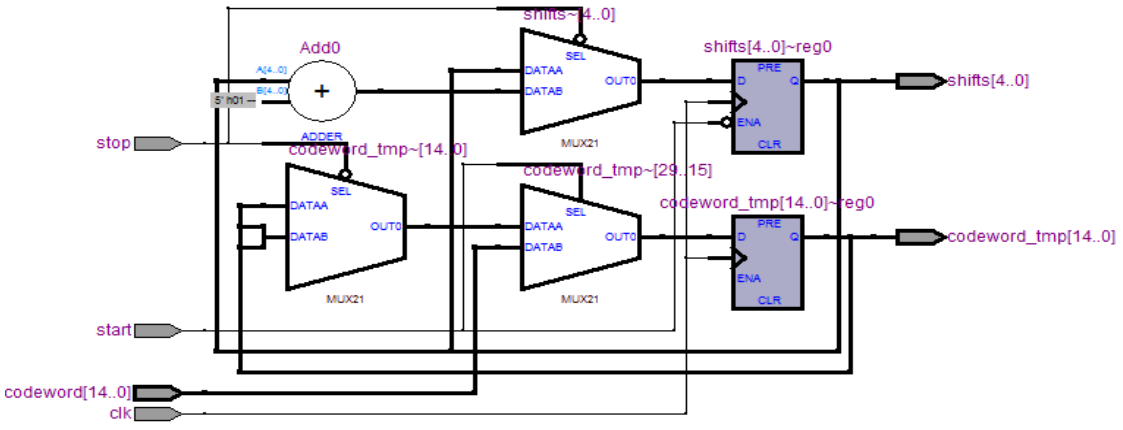


Рисунок 3.19 – Схема приёма и сдвига кодового слова

Вычисленный остаток поступает в модуль вычисления веса *check_weight* (рисунок 3.20). Результат проверки веса поступает на модуль исправления

ошибки *decoder_final*, в котором либо кодовое слово складывается с остатком (*result* = 1), либо формируется сигнал для сдвига кодового слова (*corrected* = 0) и повторения этапов декодирования.

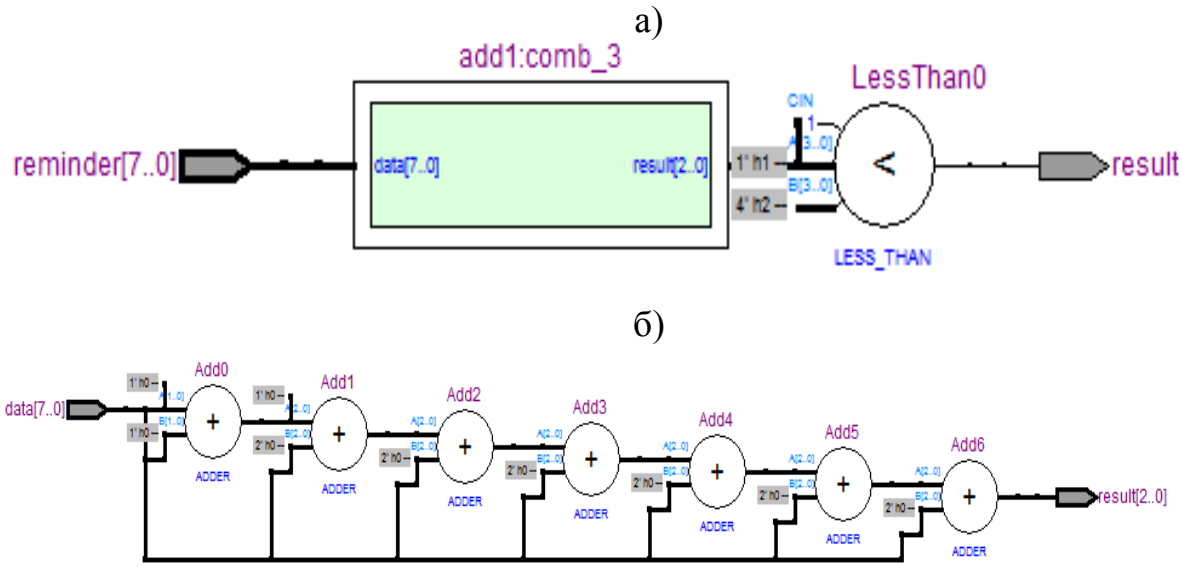


Рисунок 3.20 – Модуль *check_weight*: а) схема проверки условия $w \leq t$; б) схема вычисления веса w

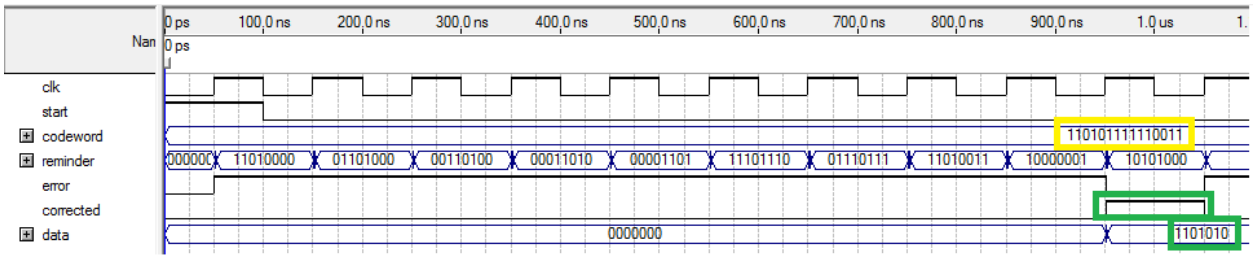
На выходе декодера формируются сигналы ошибки исправления (*error*), флага успешного исправления ошибки (*corrected*) и информационные разряды исправленного кодового слова (*data*).

На рисунке 3.21 приведены диаграммы работы декодера БЧХ кода (15, 7, 5) для разных ошибок. Информационное сообщение задано *M(0,1)* = 1101010, кодовое слово без ошибки *C(0,1)* = 110101011110010.

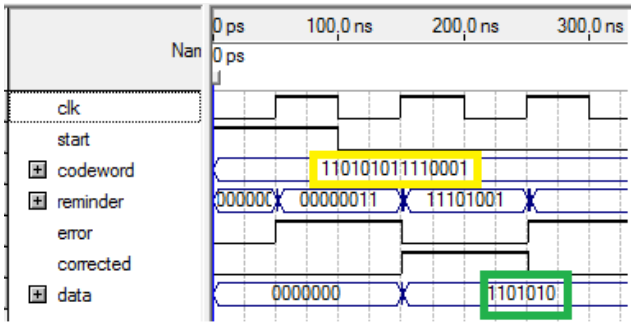
Из диаграмм на рисунке 3.21 видно, что быстродействие декодера зависит от характера ошибки. При искаженных младших битах кодового слова ошибка исправляется за 2 такта, так как первый же остаток от деления является подходящим шаблоном ошибки ($w \leq t$). При искаженных двух битах в крайних позициях кодового слова ошибка исправляется за 10 тактов *clk*, так как требуется 7 циклических сдвигов по 1 разряду и 8 операций деления кодового слова на образующий полином для определения шаблона ошибки. При искаженных 3-х битах, что теоретически является неисправимой ошибкой для

декодера, после 14 циклических сдвигов ошибка не исправляется, так как ни один из остатков не удовлетворяет условию $w \leq t$ [93].

а)



б)



в)

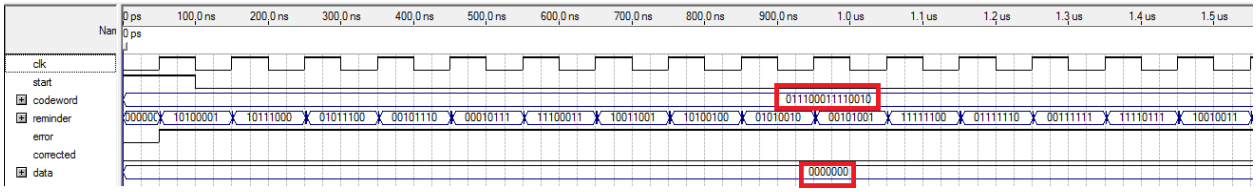


Рисунок 3.21 – Диаграммы работы декодера БЧХ кода (15, 7, 5). Классическая реализация циклического алгоритма; а) двукратная ошибка 000000100000001; б) двукратная ошибка 000000000000011; в) трёхкратная ошибка 101001000000000

На рисунке 3.22 приведены характеристики декодера из САПР Quartus II для ПЛИС Stratix III EP3SL70F780C2.

Slow 1100mV 85C Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	333.78 MHz	333.78 MHz	clk	

Entity	Combinational ALUTs	Memory ALUTs	LUT_REGs	ALMs
Stratix III: EP3SL70F780C2				
Decoder1	80 (0)	0 (0)	0 (0)	70 (0)

Рисунок 3.22 – Характеристики декодера БЧХ кода (15, 7, 5)

Для классической реализации декодера БЧХ кода (15, 7, 5) циклическим алгоритмом требуется 70 логических ячеек. Максимальная частота декодера составляет 333 MHz.

Параллельная реализация алгоритма на ПЛИС

На рисунке 3.23 приведена структурная схема [107–109] декодера БЧХ кода (15,7,5) с учетом особенностей циклического алгоритма декодирования.

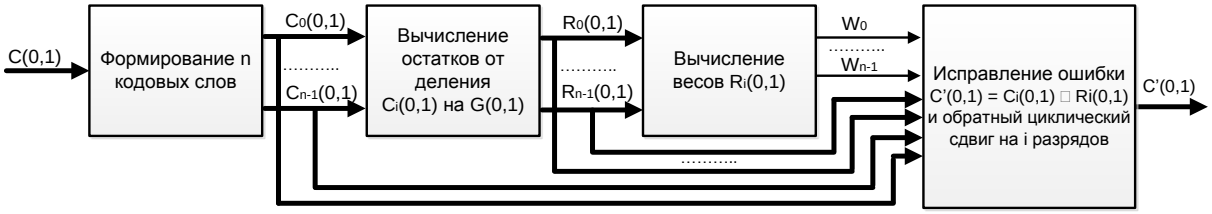


Рисунок 3.23 – Структурная схема декодера БЧХ кода (15, 7, 5). Параллельная реализация циклического алгоритма

Кодовое слово *C(0,1)* подаётся на блок генерации всех вариантов сдвигов исходного кодового слова, после чего сформированные кодовые слова (*C0(0,1)* – без сдвига, *Cn-1(0,1)* – соответствует кодовому слову, сдвинутому на *n*-1 раз) параллельно подаются на блок вычисления остатков от деления. Рассчитанные для каждого из *n* вариантов кодового слова остатки *R0(0,1)* – *Rn-1(0,1)* подаются на блок вычисления весов для остатков и формирования соответствующих сигналов *W0* – *Wn-1* для блока исправления ошибки. Если вес остатка больше заданной кратности ошибок *t*, то на выходе («*Wi*») формируется логический «0», иначе формируется логическая «1». Если сигнал «*Wi*» равен «1», то кодовое слово (*Ci(0,1)*) суммируется по модулю 2 с остатком *Ri(0,1)*. Затем, выбирается результат, соответствующий первой единице сигналов *Ri(0,1)* (поиск производится от 0 до *n*-1). Выбранный результат сдвигается циклически обратно на *i* разрядов и подаётся на выход декодера [107].

На рисунке 3.24 приведён RTL-view декодера БЧХ кода (15, 7 ,5), реализованного по структурной схеме рис. 3.23. В данной реализации кодовое слово не сдвигается циклически на каждом такте, а все кодовые слова формируются путём перестановки бит из регистра кодового слова.

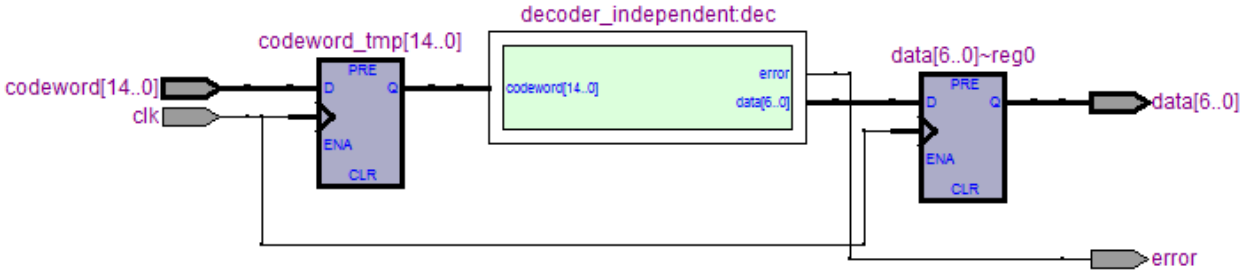


Рисунок 3.24 – RTL-view декодера БЧХ кода (15, 7, 5). Параллельная реализация циклического алгоритма

Схемы вычисления остатков от деления и весов аналогичны схемам для классической реализации циклического метода. В параллельной реализации добавляется модуль вычисления номера кодового слова (1 из n), которое будет суммироваться по модулю 2 с остатком от деления (рисунок 3.25).

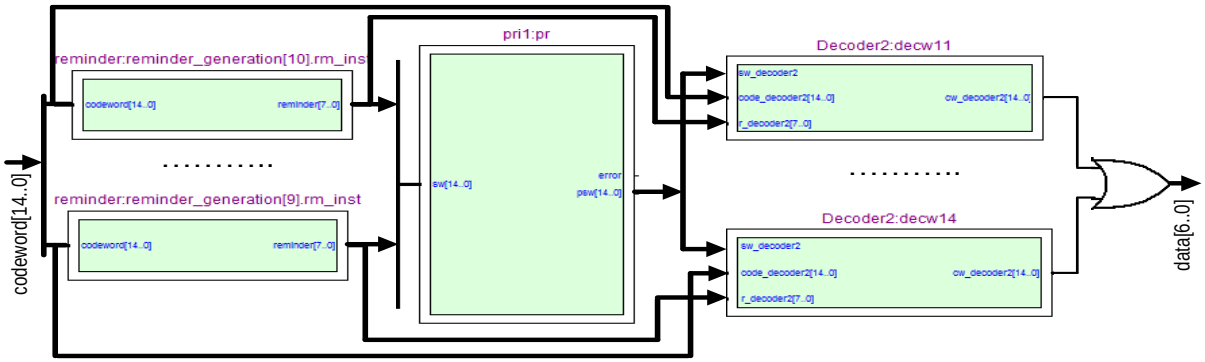


Рисунок 3.25 – RTL-view модулей декодера БЧХ кода (15, 7, 5). Параллельная реализация циклического алгоритма

Архитектура параллельного декодера с циклическим методом декодирования представляет собой двухэтапный конвейер (рисунок 3.26). Время декодирования кодового слова, поступающего в декодер, составляет 2 такта синхросигнала ПЛИС (*T1–T2*). В отличие от табличного алгоритма конвейер содержит на один этап меньше. Это достигается за счёт применения асинхронной схемы вычисления веса остатка и проверки условия $w \leq t$ вместо тактируемой памяти, что в свою очередь приводит к увеличению критического пути и снижению максимальной частоты работы устройства.

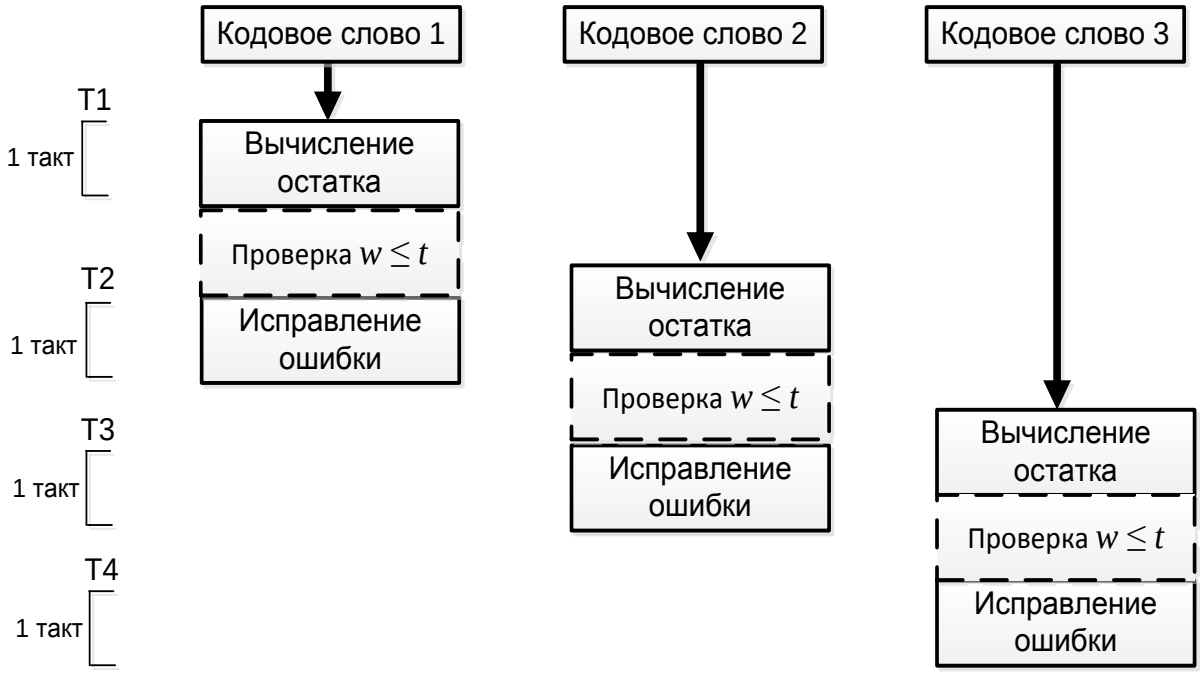


Рисунок 3.26 – Двухэтапный конвейер, реализующий циклический алгоритм декодирования

На рисунке 3.27 приведены результаты моделирования работы декодера с параллельной реализацией с двукратной ошибкой 100000000000001, одиночной ошибкой 000000000000001 и трёхкратной ошибкой 101001000000000. Исправление ошибки осуществляется за 1–2 такта.

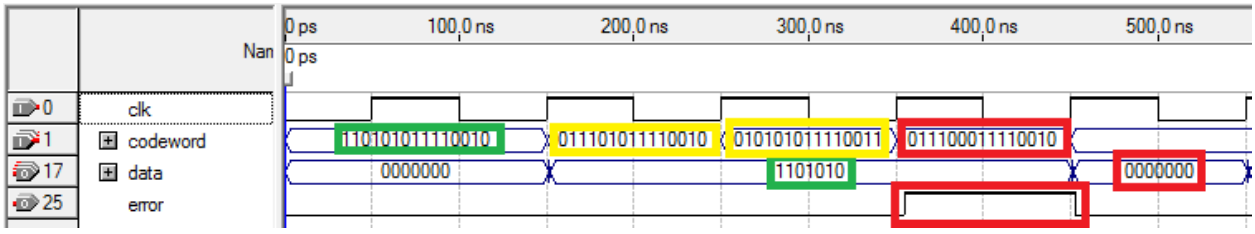


Рисунок 3.27 – Диаграммы работы декодера БЧХ кода (15, 7, 5). Параллельная реализация циклического алгоритма

На рисунке 3.28 приведены характеристики декодера из САПР Quartus II для ПЛИС Stratix III EP3SL70F780C2.

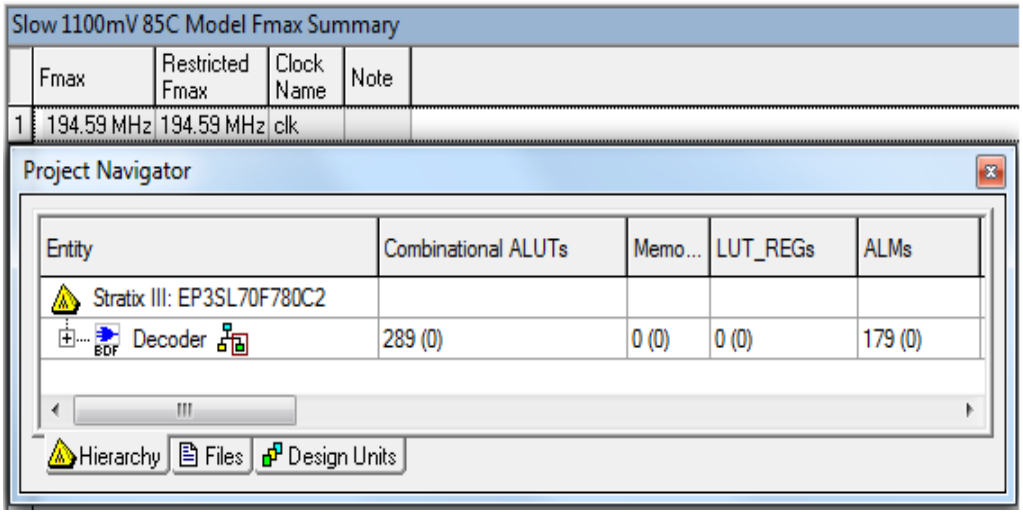


Рисунок 3.28 – Характеристика декодера БЧХ кода (15, 7, 5). Параллельная реализация циклического алгоритма

Для параллельной реализации декодера БЧХ кода (15, 7, 5) с циклическим алгоритмом требуется 179 логических ячеек. Максимальная частота декодера составляет 194 MHz.

На разработанный декодер циклического помехоустойчивого кода БЧХ (15,7,5) получено свидетельство о регистрации ПО для ЭВМ [103], копия которого представлена в приложении Г.

3.6.2. Разработка устройств на ПЛИС для исправления ошибок с применением кода (17, 9, 5)

Помехоустойчивый циклический код (17, 9, 5) является более эффективным относительно БЧХ кода (15, 7, 5), так как длина информационного блока на 2 бита больше, при сохранении длины контрольного блока. Данный код удается построить по найденному с помощью алгоритма в разделе 3.2 образующему полиному $X^8+ X^5+ X^4+ X^3+1$ (100111001). Данный код с расстоянием Хэмминга $d = 5$ позволяет исправлять 2 независимые ошибки. Скорость кода составляет 0,529.

Табличный алгоритм декодирования

Таблица шаблонов для данного кода (приложение Б, таблица Б.2) строится по аналогии с таблицей для кода БЧХ (15, 7, 5), только длина кодового слова составляет 17 бит. Реализация декодера (17, 9, 5) с табличным

алгоритмом аналогична реализации декодера БЧХ (15, 7, 5) за исключение нескольких изменений: комбинационная схема для вычисления остатка строится в соответствие с образующим полиномом 100111001; значения таблицы шаблонов ошибок имеют на 2 бита большую разрядность; схема исправления ошибки включает на 2 логических элемента «Исключающее ИЛИ» больше, ввиду увеличения длины информационного блока на 2 бита. На рисунке 3.29 приведены результаты моделирования работы декодера помехоустойчивого кода (17, 9, 5).

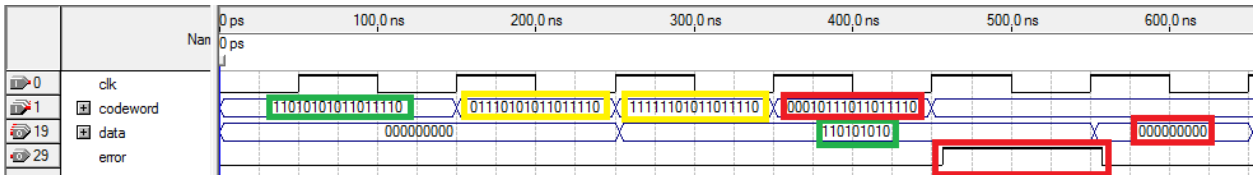


Рисунок 3.29 – Диаграмма работы декодера помехоустойчивого кода (17, 9, 5).

Табличный алгоритм

На рисунке 3.30 приведены характеристики декодера из САПР Quartus II для ПЛИС Stratix III EP3SL70F780C2.

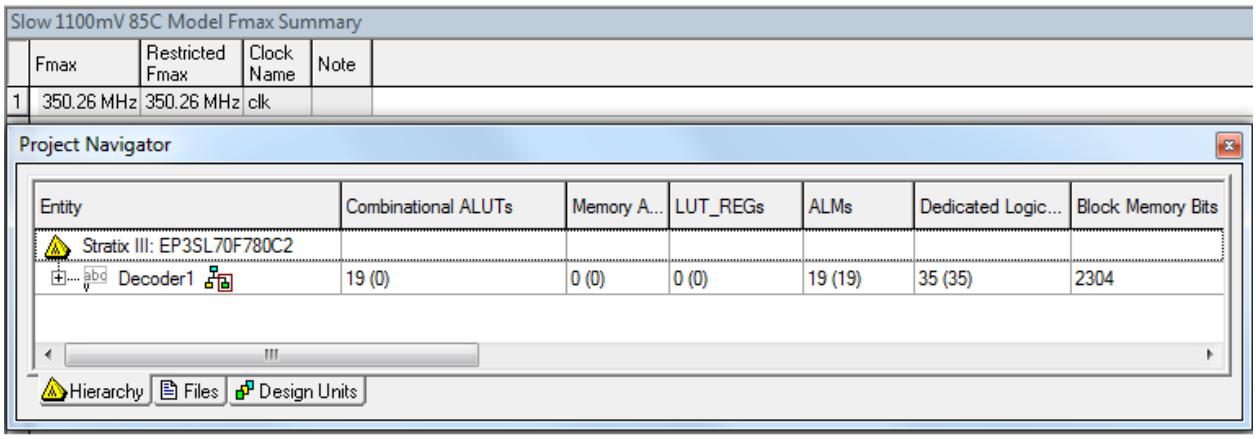


Рисунок 3.30 – Характеристики декодера помехоустойчивого кода (17, 9, 5).

Табличный алгоритм

Для реализации декодера помехоустойчивого кода (17, 9, 5) табличным алгоритмом требуется 19 логических ячеек и 2304 бита запоминающего устройства. Максимальная частота декодера составляет 350 MHz.

Модифицированный циклический алгоритм декодирования

В отличие от БЧХ кода (15, 7, 5) помехоустойчивый код (17, 9, 5) имеет скорость больше 0,5. Это не позволяет применить циклический алгоритм декодирования [50] без модификаций, описанных в разделе 3.4.3. На рисунке 3.31 приведена структурная схема устройства декодирования циклического помехоустойчивого кода (17, 9, 5) с модифицированным циклическим алгоритмом и проверкой остатка на совпадение с синдромом ошибки, выходящей за диапазон скользящего окна, при каждом циклическом сдвиге. В данной реализации сигнал «Исправление 2» устанавливается в значение «1» в случае, когда есть совпадение вычисленного синдрома ошибки с первым синдромом из таблицы шаблонов ошибок, находящихся вне скользящего окна. Финальный этап исправления ошибки реализуется сложением по модулю 2 кодового слова либо с подходящим под условие остатком (сигнал Исправление 1 установлен в «1»), либо с шаблоном ошибки из таблицы (сигнал Исправление 2 установлен в «1»).

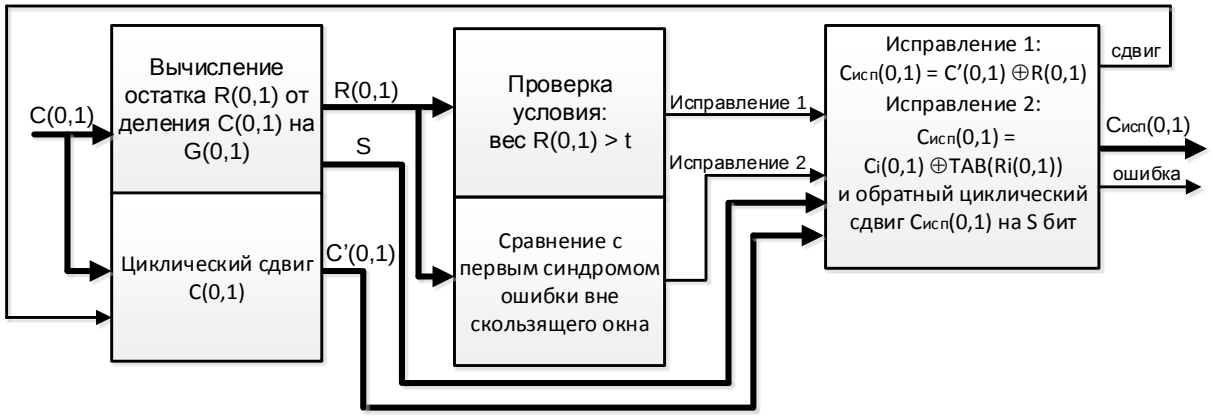
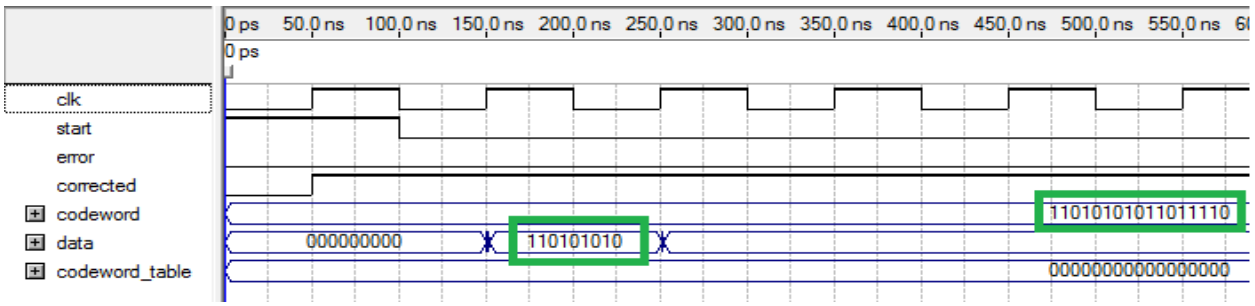


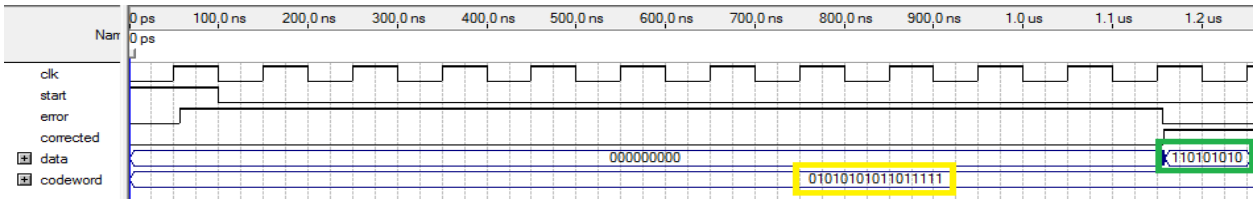
Рисунок 3.31 – Структурная схема устройства декодирования циклического помехоустойчивого кода (17, 9, 5). Классическая реализация модифицированного циклического алгоритма

На рисунке 3.32 (а–г) приведены диаграммы работы декодера кода (17, 9, 5), разработанного по структурной схеме рисунка 3.31, с различными ошибками в кодовом слове. Максимальное количество тактов для исправления двукратной ошибки составляет 12. При исправлении ошибки, выходящей за диапазон скользящего окна, требуется 2 такта.

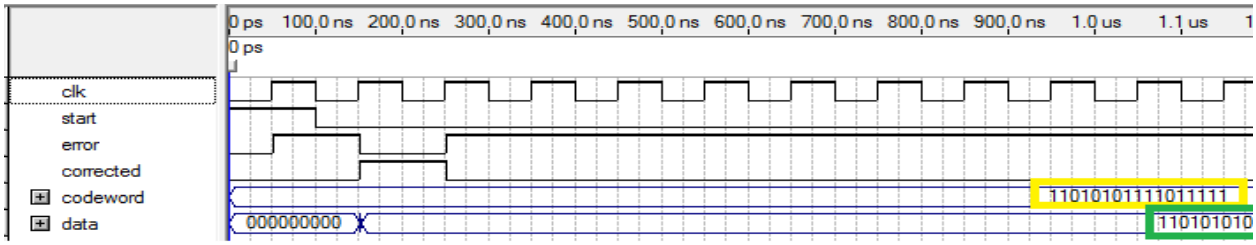
а)



б)



в)



г)

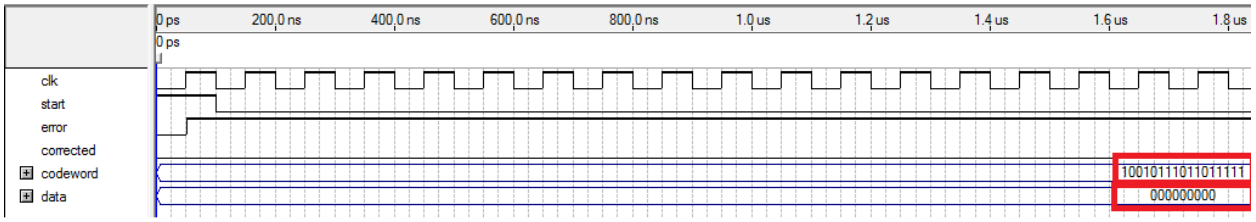


Рисунок 3.32 – Диаграмма работы декодера помехоустойчивого кода (17, 9, 5). Классическая реализация модифицированного циклического алгоритма. а) без ошибки; б) ошибка 10000000000000001; в) ошибка, выходящая за скользящее окно 00000000100000001; г) неисправимая ошибка 01000010000000001

Структурная схема параллельной реализации отражена на рисунке 3.33 Если сигнал «*Wi*» равен «1», то кодовое слово (*Ci(0,1)*) суммируется по модулю 2 с остатком *Ri(0,1)*, иначе, если сигнал «*Ti*» равен «1», то кодовое слово (*Ci(0,1)*) суммируется по модулю 2 со значением из таблицы по адресу *Ri(0,1)*.

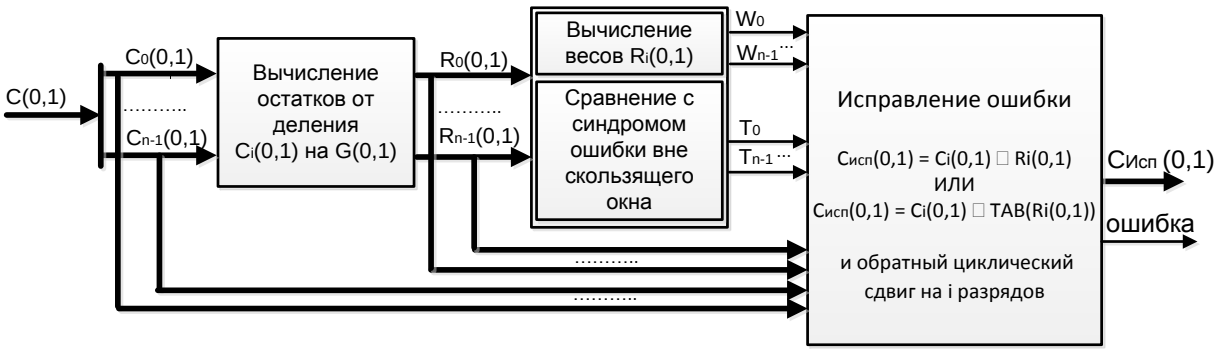


Рисунок 3.33 – Структурная схема декодера помехоустойчивого кода (17, 9, 5) Параллельная реализация модифицированного циклического алгоритма

На рисунке 3.34 приведена диаграмма работы декодера кода (17, 9, 5) с модифицированным циклическим алгоритмом с двукратной ошибкой 100000000000001, трёхкратной ошибкой 101001000000000 и двукратной ошибкой, выходящей за диапазон скользящего окна 000000100000001. Исправление ошибки осуществляется за 2 такта (с учетом записи результата в регистр).

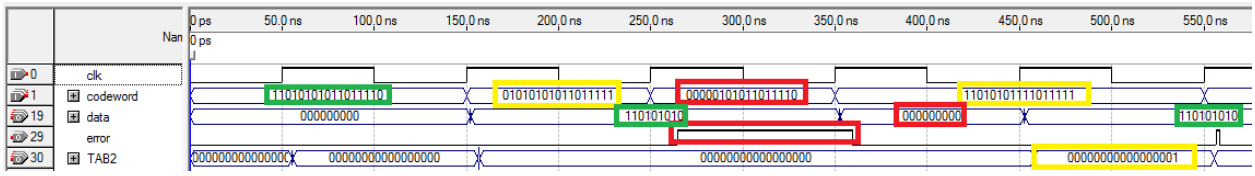


Рисунок 3.34 – Диаграмма работы декодера помехоустойчивого кода (17, 9, 5). Параллельная реализация модифицированного циклического алгоритма

Регистр *TAB2* содержит код 1 из *n*, единичный разряд которого показывает номер кодового слова, остаток которого совпал с синдромом первой строки таблицы 3.8 (00111000).

На рисунке 3.35 (а,б) приведены характеристики устройств декодирования на ПЛИС Stratix III EP3SL70F780C2 для классической и параллельной реализаций.

а)

Slow 1100mV 85C Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	258.8 MHz	258.8 MHz	clk	

Entity	Combinational ALUTs	Block Memory Bits	Dedicated Lo...	ALMs
Stratix III: EP3SL70F780C2				
Decoder1	140 (0)	0	50 (0)	103 (0)

б)

Slow 1100mV 85C Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	177.68 MHz	177.68 MHz	clk	

Entity	Combinational ALUTs	Memory A...	LUT_REGs	ALMs	Dedicated Logic.
Stratix III: EP3SL70F780C2					
Decoder	461 (0)	0 (0)	0 (0)	287 (0)	26 (0)

Рисунок 3.35 – Характеристики декодеров помехоустойчивого кода (17, 9, 5) с модифицированным циклическим алгоритмом: а) классическая реализация; б) параллельная реализация

Для классической реализации декодера помехоустойчивого кода (17, 9, 5) циклическим алгоритмом требуется 103 логических ячеек. Максимальная частота устройства составляет 258 МГц. При параллельной реализации, требуемые вычислительные ресурсы увеличиваются в 2,8 раз при уменьшении частоты в 1,5 раза.

3.6.3. Разработка устройств на ПЛИС для исправления ошибок с применением укороченного кода БЧХ (19, 9, 5)

Циклический помехоустойчивый код БЧХ (19, 9, 5) является укороченным кодом от БЧХ (31, 21, 5). В данном коде длина информационного блока сокращена на 12 бит. Помехоустойчивый код БЧХ (19, 9, 5) строится на основе образующего полинома $G(0,1)$ = 11101101001.

Недостатками кода БЧХ (19, 9, 5) относительно предложенного кода (17,9,5) являются: меньшая скорость кода (0,47), при одинаковой длине информационного блока длина избыточного блока на 2 бита больше;

необходимость добавления 12-ти нулевых бит при декодировании циклическим алгоритмом, что увеличивает аппаратные затраты декодирующего устройства.

Для сравнения быстродействия и аппаратных затрат устройств декодирования кода БЧХ (19, 9, 5) и кода (17, 9, 5) осуществлена реализация декодеров на ПЛИС с применением табличного и модифицированного циклического алгоритмов. Применение модифицированного циклического алгоритма обусловлено тем, что при добавлении 12-ти нулевых бит (необходимое условие для укороченного кода БЧХ (19, 9, 5)), некоторые ошибки могут выходить за диапазон скользящего окна.

На рисунке 3.36 приведена диаграмма работы устройства декодирования на ПЛИС для укороченного кода БЧХ (19, 9, 5).

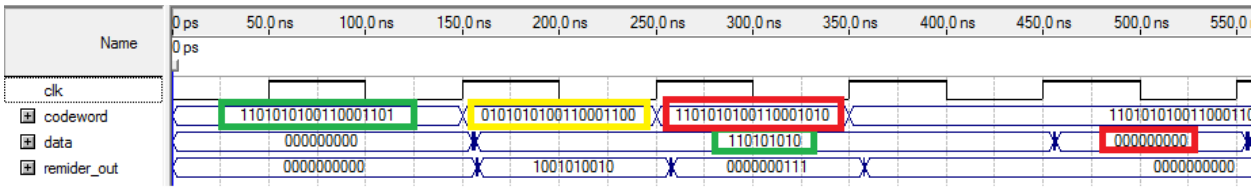


Рисунок 3.36 – Диаграмма работы декодера укороченного кода БЧХ (19, 9, 5). Табличный алгоритм

На рисунке 3.37 приведены характеристики устройства декодирования кода БЧХ (19, 9, 5) на ПЛИС Stratix III EP3SL70F780C2 с применением табличного алгоритма.

Slow 1100mV 85C Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	348.07 MHz	348.07 MHz	clk	

Entity	Combinational ALUTs	Block Memory Bits	Dedicated Logic...	ALMs
Stratix III: EP3SL50F780C2				
Decoder	23 (0)	9216	27 (0)	21 (0)

Рисунок 3.37 – Характеристики декодера укороченного кода БЧХ (19, 9, 5). Табличный алгоритм

Для реализации декодера кода БЧХ (19, 9, 5) табличным алгоритмом требуется 21 логическая ячейка и 9216 бит запоминающего устройства. Максимальная частота декодера составляет 348 MHz. Таким образом, быстродействие данного устройства по сравнению с декодером кода (17, 9, 5)

остается практическим неизменным, однако объем памяти для хранения шаблонов ошибок увеличивается в 4 раза.

На рисунке 3.38 приведена диаграмма работы устройства декодирования на ПЛИС для укороченного кода БЧХ (19, 9, 5) с применением параллельной реализации модифицированного циклического алгоритма декодирования.

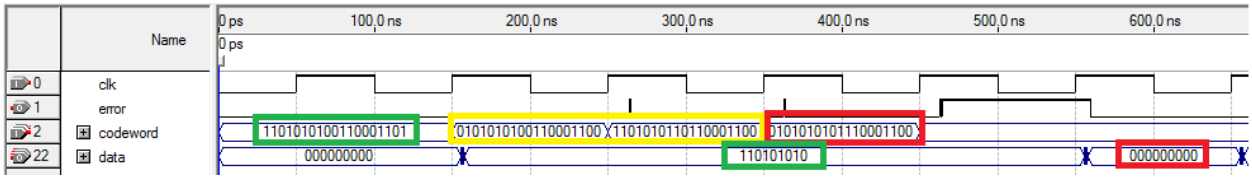


Рисунок 3.38 – Диаграмма работы декодера укороченного кода БЧХ (19, 9, 5).

Параллельная реализация модифицированного циклического алгоритма

На рисунке 3.39 (а,б) приведены характеристики устройств декодирования на ПЛИС Stratix III EP3SL70F780C2 для классической и параллельной реализаций.

а)

Slow 1100mV 85C Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	181.09 MHz	181.09 MHz	clk	

Entity	Combinational ALUTs	Block Memory Bits	Dedicated Lo...	ALMs
Stratix III: EP3SL70F780C2				
Decoder1	210 (0)	0	74 (0)	159 (0)

б)

Slow 1100mV 85C Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	86.99 MHz	86.99 MHz	Decoder1:inst|codeword_tmp[0]	

Entity	Combinational ALUTs	Memory ALUTs	Dedicated Logic...	ALMs
Stratix III: EP3SL70F780C2				
Decoder	1306 (0)	0 (0)	28 (0)	767 (0)

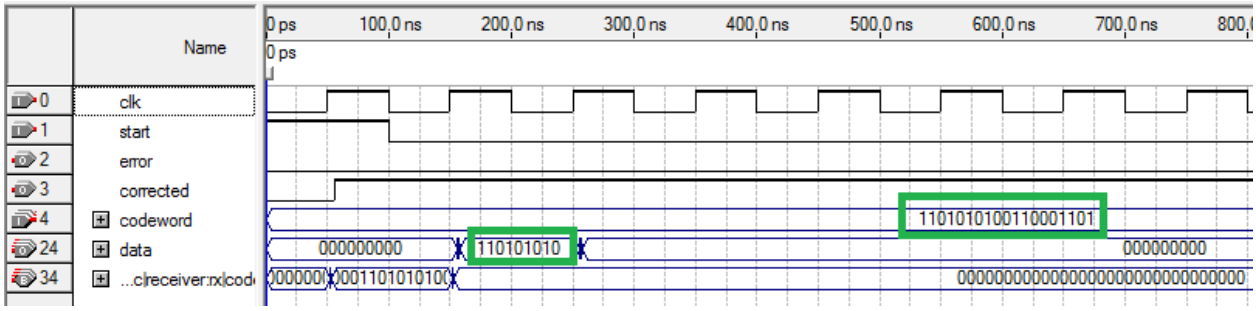
Рисунок 3.39 – Характеристики декодеров укороченного кода БЧХ (19, 9, 5). Модифицированный циклический алгоритм: а) классическая реализация; б) параллельная реализация

Для классической реализации декодера помехоустойчивого кода БЧХ (19, 9, 5) циклическим алгоритмом требуется 159 логических ячеек. Максимальная частота устройства составляет 181 MHz. При параллельной реализации,

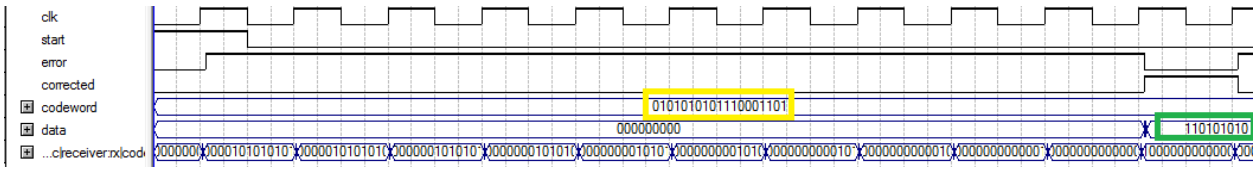
требуемые вычислительные ресурсы увеличиваются в 4,8 раз при уменьшении частоты в 2 раза.

На рисунке 3.40 (а–г) приведены диаграммы работы устройств декодирования на ПЛИС для укороченного кода БЧХ (19, 9, 5) с применением классической реализации циклического модифицированного алгоритма декодирования.

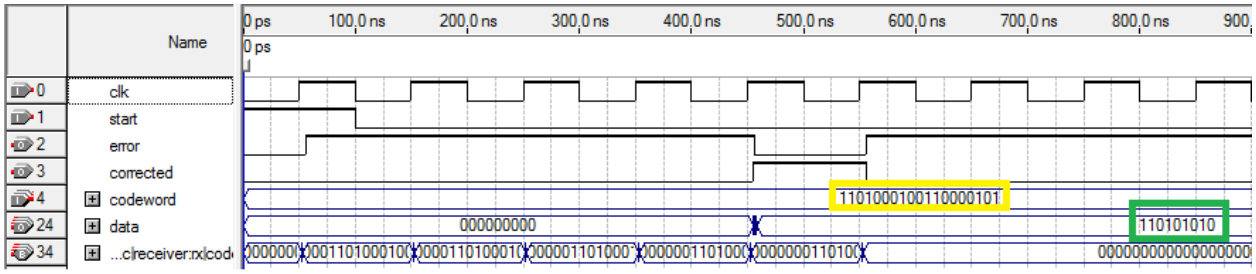
а)



б)



в)



г)

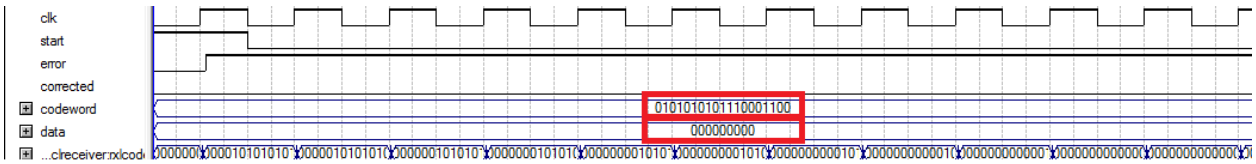


Рисунок 3.40 – Диаграмма работы декодера укороченного кода БЧХ (19, 9, 5). Классическая реализация модифицированного циклического алгоритма: а) без ошибки; б) ошибка 1000000000000000000; в) ошибка, выходящая за скользящее окно 00000010000000001; г) неисправимая ошибка 1000000001000000001

3.6.4. Обсуждение результатов и сравнение

Устройство декодирования кода БЧХ (15,7,5)

Существует множество реализаций декодера БЧХ кода (15, 7, 5) на ПЛИС, описанных в основном в зарубежных публикациях [109–116]. Однако, во многих из них приводится только описание реализаций методов и алгоритмов декодирования без указания характеристик, по которым можно оценить быстродействие и аппаратные затраты устройства. Описание наиболее близкого по проведенному исследованию аналога декодера БЧХ кода (15, 7, 5) с конвейерной архитектурой (рисунок 3.41) приведено в работе [113].

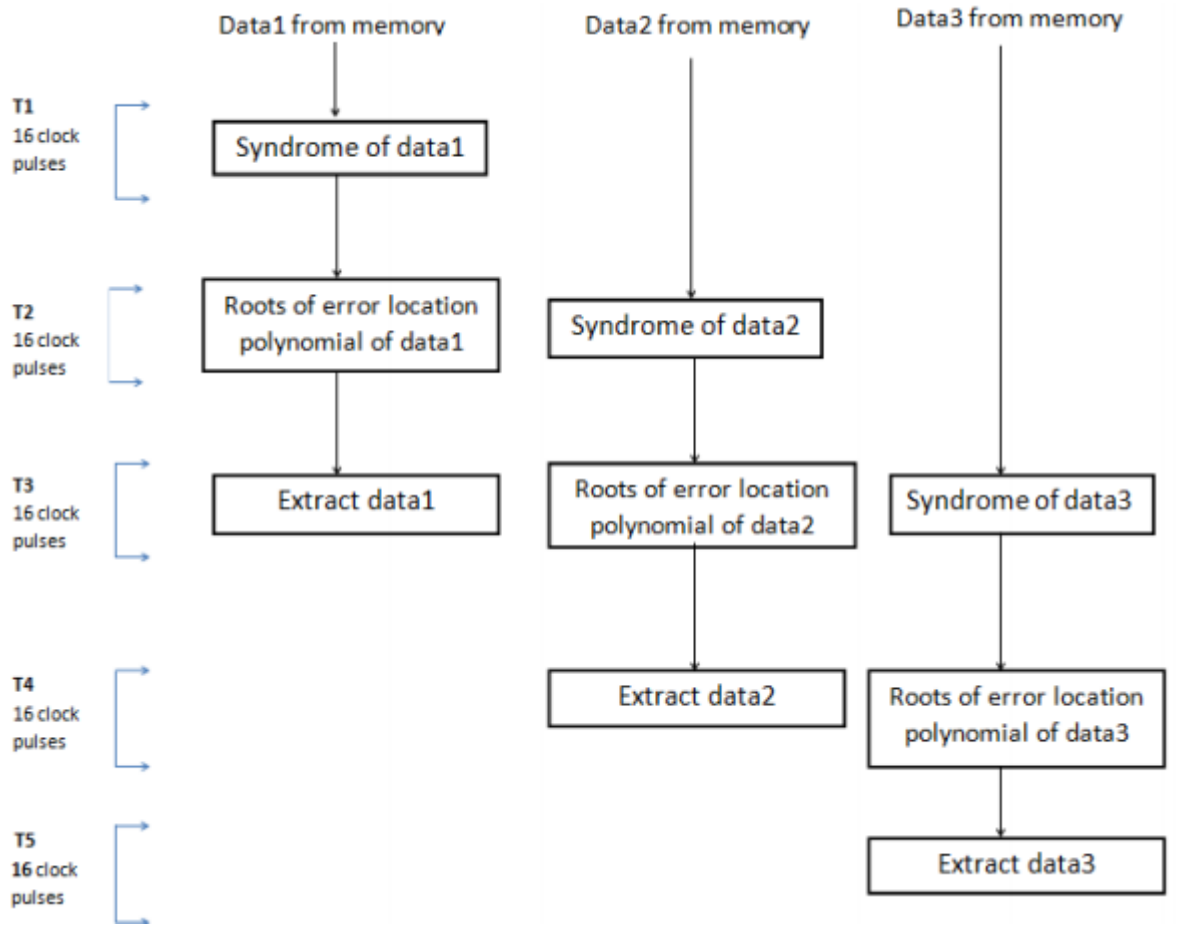


Рисунок 3.41 – Конвейерная архитектура декодера БЧХ кода (15,7,5)

В отличие от предложенной реализации, исправление ошибок в [113] осуществляется путём решения ключевого уравнения с помощью алгоритма Берлекэмпа-Месси, описанного в главе 1, на что требуется 16 тактов устройства (*T2*, рисунок 3.41). Таким образом, если опустить этап извлечения данных, применяемый только при несистематическом кодировании [113], время

загрузки конвейера составит 32 такта, а каждое следующее кодовое слово будет декодироваться за 16 тактов.

В таблице 3.12 приведены характеристики реализованных декодеров БЧХ кода (15, 7, 5) и декодеров-аналогов (на основе БМА). Для оценки быстродействия и аппаратных затрат была выбрана ПЛИС Stratix III EP3SL70F780C2 как наиболее производительное устройство из имеющихся в библиотеке САПР Quartus II 9.1.

Таблица 3.12 – Характеристики декодеров БЧХ кода (15, 7, 5)

Алгоритм	**Макс. частота, МГц**	**Такты**	**Мин. время, нс**	**Кол-во ячеек**	**Память, бит**	**ПЛИС**
Табличный	429	3	6,9	19	1792	Stratix III
Циклический (классический)	331	16	48	67	0	Stratix III
Циклический (параллельный)	194	2	10,2	179	0	Stratix III
БМА [113]	1000	32	32	-	0	Xilinx
БМА [116]	204	15	73,5	120	0	Xilinx xc4013

Максимальной частотой декодирования (1 ГГц) (согласно сведениям в работе [113]) обладает устройство на основе метода с применением алгоритма БМА, однако исправление ошибок осуществляется за 32 такта (для первого кодового слова в потоковой передаче или при одиночной передаче) и 16 тактов для каждого последующего кодового слова. Время декодирования составляет 32 нс. Данные по используемым логическим ячейкам в работе [113] не приводятся.

Табличный алгоритм декодирования при реализации по конвейерной архитектуре позволяет осуществлять загрузку конвейера за 3 такта при частоте 429 МГц, что достигается за счёт простых операции выбора значения из памяти и сложения по модулю 2. Минимальное время декодирования составляет 6,9 нс, что в 4,5 раза быстрее, чем реализация с БМА. Однако для реализации данного

алгоритма требуется 1792 бита памяти для хранения шаблонов ошибок кода БЧХ (15, 7, 5).

При классической реализации циклического алгоритма декодирования количество итераций (тактов устройства) зависит от позиций ошибок в кодовом слове и варьируется от 2 до 16. В таблице 3.13 приведены шаблоны однократных и двукратных ошибок с указанием количества тактов для их исправления. Так, для исправления двух ошибок по шаблону 000000000000011 потребуется 2 такта, а для шаблона 100000001000000 – 16 тактов. Таким образом, согласно таблице 3.13 для декодирования кодового слова БЧХ (15, 7, 5) потребуется до 16 тактов с частотой 331 МГц.

Таблица 3.13 – Количество тактов в зависимости от позиций ошибки

Шаблон	**Кол-во тактов**	**Шаблон**	**Кол-во тактов**	**Шаблон ошибки**	**Кол-во тактов**
100000000000000	9	10000000000010	11	1000001000	4
10000000000000	8	10000000000001	10	1000000100	4
1000000000000	7	1100000000000	7	1000000010	11
100000000000	6	1010000000000	7	1000000001	10
10000000000	5	1001000000000	7	110000000	3
1000000000	4	1000100000000	7	101000000	3
100000000	3	1000010000000	7	100100000	3
10000000	2	1000001000000	7	100010000	3
1000000	2	1000000100000	7	100001000	3
100000	2	1000000010000	14	100000100	3
10000	2	1000000001000	13	100000010	3
1000	2	1000000000100	12	100000001	10
100	2	1000000000010	11	11000000	2
10	2	1000000000001	10	10100000	2
1	2	110000000000	6	10010000	2
110000000000000	9	101000000000	6	10001000	2
101000000000000	9	100100000000	6	10000100	2
100100000000000	9	100010000000	6	10000010	2
100010000000000	9	100001000000	6	10000001	2
100001000000000	9	100000100000	6	1100000	2
100000100000000	9	100000010000	6	1010000	2
100000010000000	9	100000001000	13	1001000	2
100000001000000	16	100000000100	12	1000100	2
100000000100000	15	100000000010	11	1000010	2
100000000010000	14	100000000001	10	1000001	2
100000000001000	13	11000000000	5	110000	2
100000000000100	12	10100000000	5	101000	2

Окончание таблицы 3.13

100000000000010	11	10010000000	5	100100	2
100000000000001	10	10001000000	5	100010	2
11000000000000	8	10000100000	5	100001	2
10100000000000	8	10000010000	5	11000	2
10010000000000	8	10000001000	5	10100	2
10001000000000	8	10000000100	12	10010	2
10000100000000	8	10000000010	11	10001	2
10000010000000	8	10000000001	10	1100	2
10000001000000	8	1100000000	4	1010	2
10000000100000	15	1010000000	4	1001	2
10000000010000	14	1001000000	4	110	2
10000000001000	13	1000100000	4	101	2
10000000000100	12	1000010000	4	11	2
Среднее количество тактов					6,2

Циклический алгоритм декодирования с параллельной реализацией позволяет декодировать кодовое слово с ошибкой за 2 такта с частотой 194 МГц, при этом время декодирования не зависит от позиций ошибок. Минимальное время декодирования для параллельной реализации декодирования составляет 10,2 нс, что в 3 раза быстрее, чем декодер с реализацией БМА [113] и в 7 быстрее, чем декодер с БМА [116], работающим приблизительно на такой же частоте (204 МГц).

Таким образом, при сравнении реализаций декодера БЧХ кода (15, 7, 5) установлено, что параллельная реализация циклического алгоритма декодирования обладает лучшим быстродействием по сравнению с рассмотренными реализациями на основе алгоритма БМА [113, 116]. Однако, в данном случае нет возможности сравнить аппаратные затраты. Заметим, что классическая реализация циклического алгоритма декодирования для некоторых шаблонов ошибок (с количеством тактов для исправления ≤ 5) обладает более высоким быстродействием, чем реализации с БМА [113], при аппаратных затратах (количество логических ячеек) в 2,5 раза меньше, чем при параллельной реализации. При этом, классическая реализация ЦМК исправляет ошибки в 1,5 раза быстрее реализации с БМА [116] при аппаратных затратах в 1,7 раза меньше.

Устройства декодирования кода БЧХ (19, 9, 5) и кода (17, 9, 5)

В таблице 3.14 приведены характеристики различных декодеров кода БЧХ (15, 7, 5), укороченного кода БЧХ (19, 9, 5) и предложенного кода (17, 9, 5) при реализации на ПЛИС Stratix III EP3SL70F780C2.

Таблица 3.14 – Характеристики исследуемых устройств декодирования

Алгоритм	Макс. частота, МГц	Такты	Мин. время, нс	Кол-во ячеек	Память ROM, бит
Декодер кода БЧХ (15, 7, 5)					
Табличный	429	3	6,9	19	1792
Циклический (классический)	333	16	48	70	0
Циклический (параллельный)	194	2	10,2	179	0
Декодер кода (17, 9, 5)					
Табличный	350	3	8,4	19	2304
Циклический (классический)	258	18	69,3	103	0
Циклический (параллельный)	177	2	11,2	287	0
Декодер укороченного кода БЧХ (19, 9, 5)					
Табличный	348	3	8,6	21	9216
Циклический (классический)	181	11	60,7	159	0
Циклический (параллельный)	87	2	23	767	0

При классической реализации циклического алгоритма декодирования с применением помехоустойчивого кода (17, 9, 5) требуется до 18 тактов (наихудший случай) с частотой 258 МГц для исправления ошибок. Применение укороченного кода БЧХ (19, 9, 5) позволяет исправлять ошибки за 11 тактов (наихудший случай) с частотой 181 МГц, при этом в среднем количество тактов составляет 5 (таблица В.2 приложения В), что соответствует времени работы 27,6 нс. Декодирующее устройство на основе кода БЧХ (19, 9, 5) работает с меньшей частотой, однако исправляет ошибки за меньшее количество тактов, что обусловлено наличием большего числа ошибок, выходящих за диапазон

скользящего окна, чем у кода (17, 9, 5). Как известно из раздела 3.4.3 ошибки, выходящие за диапазон скользящего окна, исправляются с помощью отдельной схемы, требующей наименьшего количества тактов.

При параллельной реализации циклического алгоритма декодирования устройство исправления ошибок на основе ЦПК (17, 9, 5) работает в 2 раза быстрее устройства на основе БЧХ (19, 9, 5), при этом обладает в 2,6 раза меньшими аппаратными затратами.

Таким образом, при сравнении устройств декодирования на ПЛИС с применением укороченного кода БЧХ (19, 9, 5) и кода (17, 9, 5) можно сделать следующие выводы:

- при реализации табличного алгоритма декодирования предложенного кода (17, 9, 5) требуется в 4 раза меньший объем памяти для хранения шаблонов ошибок, чем при реализации укороченного кода БЧХ (19, 9, 5);
- при классической реализации циклических алгоритмов максимальная частота работы декодирующего устройства для предложенного кода (17, 9, 5) на 42% выше, чем для укороченного кода БЧХ (19, 9, 5), однако время исправления ошибки для помехоустойчивого кода (17, 9, 5) в наихудшем случае на 8,6 нс больше (69,3 нс против 60,7 нс);
- при параллельной реализации циклических алгоритмов максимальная частота работы декодирующего устройства для предложенного кода (17, 9, 5) в 2 раза выше, чем для укороченного кода БЧХ (19, 9, 5) при одинаковом количестве тактов;
- количество логических элементов для предложенного кода (17, 9,5) в 1,54 меньше, чем для укороченного кода БЧХ (19, 9, 5) при классической реализации циклического алгоритма и в 2,67 раза меньше при параллельной реализации.

Таким образом, предложенный помехоустойчивый код (17, 9, 5) по сравнению с кодами БЧХ (15, 7, 5) и (19, 9, 5) обладает следующими преимуществами [117, 118]:

- является более эффективным помехоустойчивым кодом, чем код БЧХ (15, 7, 5) с точки зрения передачи информации и скорости кода. Предложенный код (17, 9, 5) при сохранении длины избыточного блока, позволяет контролировать на 2 бита информационного сообщения больше;
- является менее аппаратно-затратным и более быстродействующим (при параллельной реализации), чем укороченный код БЧХ (19, 9, 5) при реализации устройства декодирования на ПЛИС. Предложенный код (17, 9, 5) при одинаковой длине информационного сообщения имеет на 2 бита меньшую длину кодового слова и не требует добавления нулевых бит при выполнении операции декодирования циклическим алгоритмом.

3.7. Разработка устройств исправления пакетных ошибок на ПЛИС

Пакетные ошибки являются частным случаем независимых ошибок, поэтому для их исправления можно применять декодеры, исправляющие независимые ошибки. Однако для исправления только пакетных ошибок требуется меньшая избыточность кодового слова, что позволяет разработать более эффективное с точки зрения требования к аппаратным ресурсам устройство декодирования. Далее рассмотрены реализации декодера циклического помехоустойчивого кода (15, 8, 3) для исправления пакетных ошибок [99 – 101], длиной до 3-х бит при передаче информационных символов размером 1 байт.

3.7.1. Разработка устройств на ПЛИС для исправления пакетных ошибок с применением кода (15, 8, 3)

Табличный алгоритм декодирования

Табличный алгоритм декодирования для исправления пакетных ошибок реализуется аналогично табличному алгоритму для независимых, за исключением того, что таблица содержит шаблоны пакетных ошибок, которых значительно меньше, чем независимых. На рисунке 3.41 показана работа декодера кода (15, 8, 3), исправляющего пакетные ошибки до 3-х бит.

Для моделирования заданы следующие шаблоны ошибок: двукратная – **11**0000000000000, пакетная 3 бита – **111**000000000000, пакетная 4 бита – **1111**00000000000.

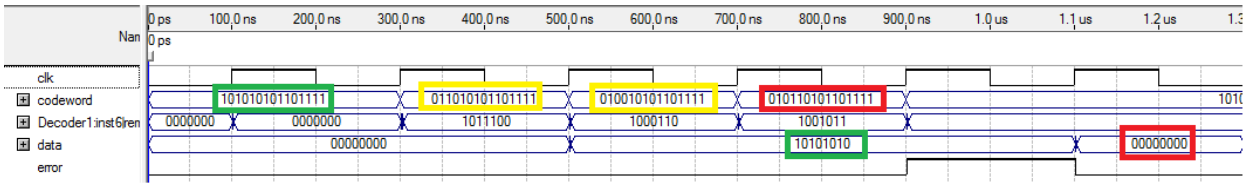


Рисунок 3.41 – Диаграмма работы декодера помехоустойчивого кода (15, 8, 3). Табличный алгоритм

На рисунке 3.42 приведены характеристики декодера (15, 8, 3) реализации на ПЛИС Stratix III EP3SL70F780C2.

Slow 1100mV 85C Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	565.61 MHz	565.61 MHz	clk	

Entity	Combinati...	Memory ALUTs	LUT_REGs	ALMs	Dedicated Lo...	Block Memory Bits
Stratix III: EP3SL70F780C2						
Decoder1	17 (0)	0 (0)	0 (0)	17 (17)	31 (31)	1024

Рисунок 3.42 – Характеристики декодера помехоустойчивого кода (15, 8, 3). Табличный алгоритм

Частота работы декодера немного выше, чем для декодера БЧХ кода (15, 7, 5) (429 МГц), при этом объём памяти для хранения таблицы составляет 1024 бита.

Циклический алгоритм декодирования с классической реализацией на ПЛИС

На рисунке 3.43 приведена схема декодера циклического помехоустойчивого кода, исправляющего пакетные ошибки.

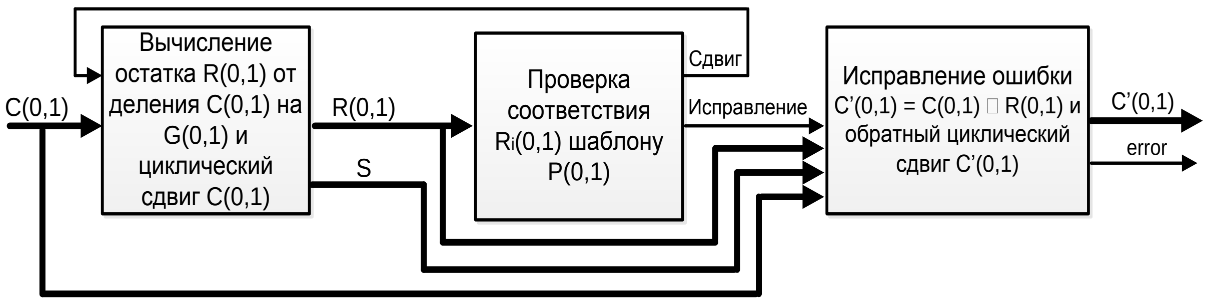


Рисунок 3.43 – Структурная схема декодера ПК, исправляющего пакетные ошибки

Остаток от деления кодового слова на образующий полином проверяется на соответствие одному из шаблонов пакетной ошибки длины *p*. При реализации на ПЛИС сформированные шаблоны ошибок можно реализовать в виде формулы (рисунок 3.44).

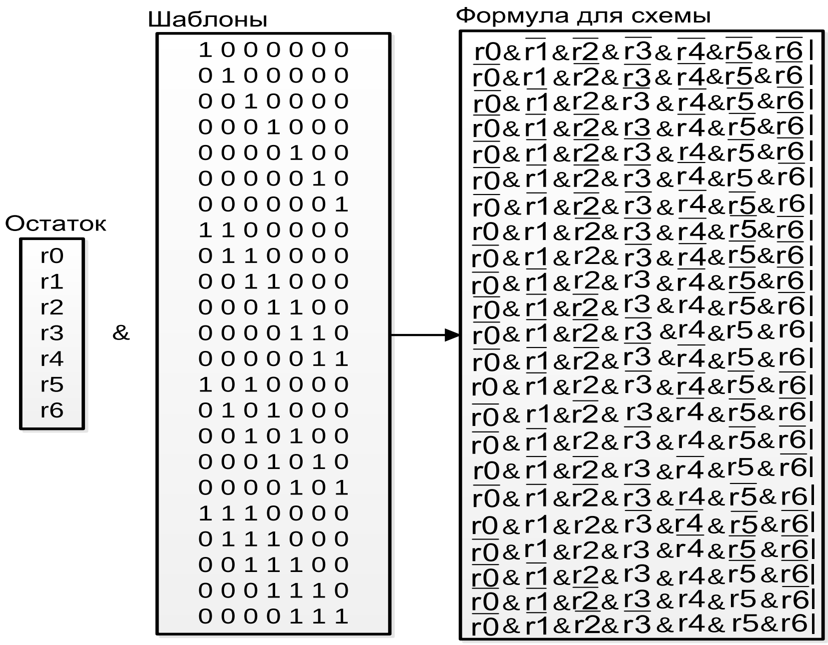


Рисунок 3.44 – Формула проверки остатка на соответствие шаблону ошибки для кода (15, 8, 3)

Комбинационная схема, реализующая формулу по рисунку 3.44, представлена на рисунке 3.45. Остаток от деления побитового умножается (операция поразрядного «И») на матрицу шаблонов трёхкратной пакетной ошибки. По полученной формуле строится комбинационная схема на элементах «И», «ИЛИ». «НЕ».

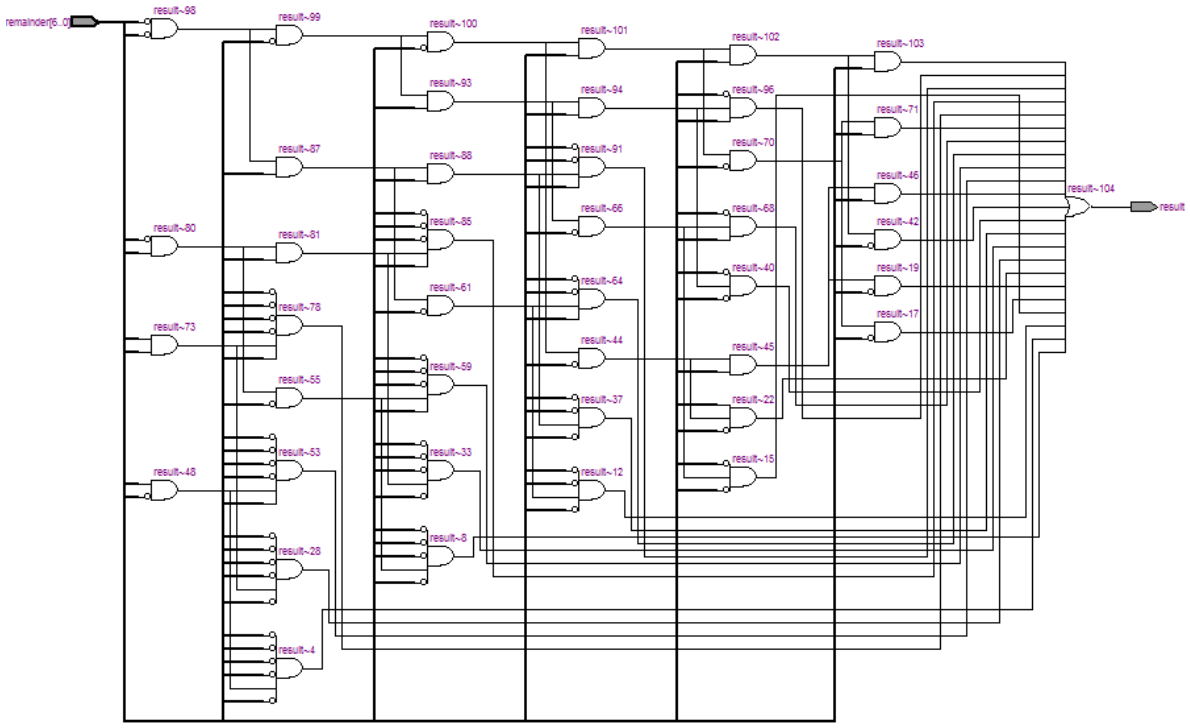


Рисунок 3.45 – Комбинационная схема проверки остатка на соответствие шаблону ошибки для кода (15, 8, 3)

Исправление ошибки осуществляется с помощью комбинационной схемы, представленной на рисунке 3.46.

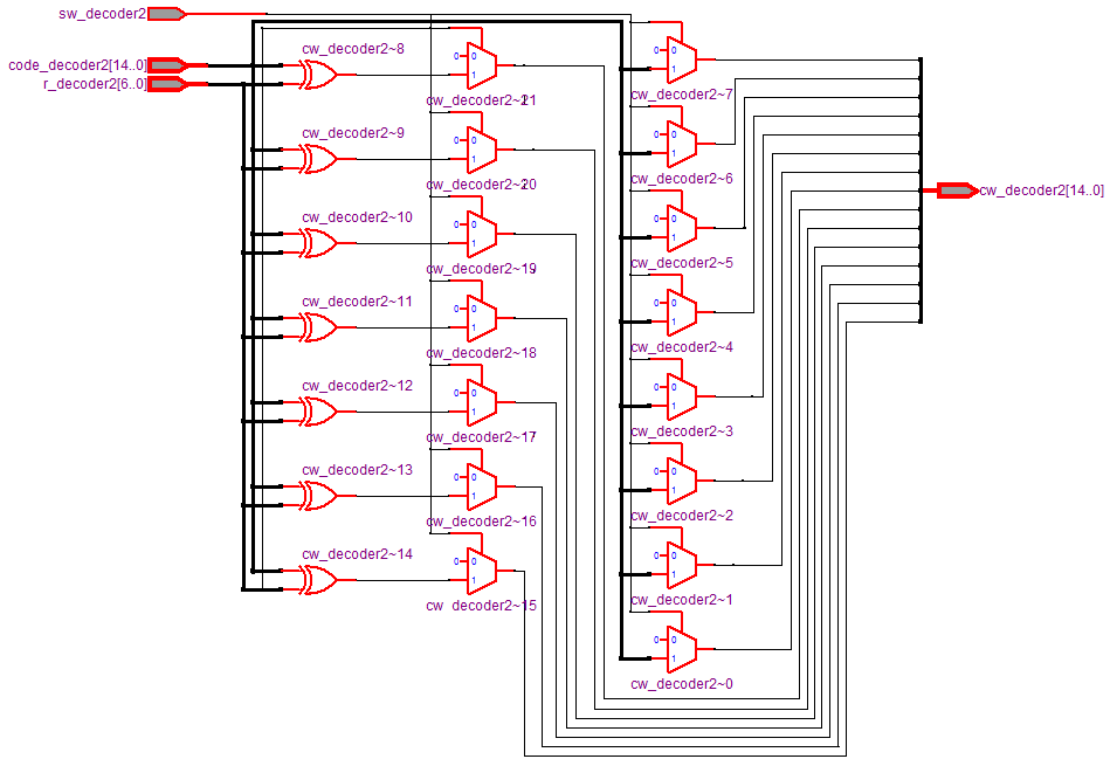
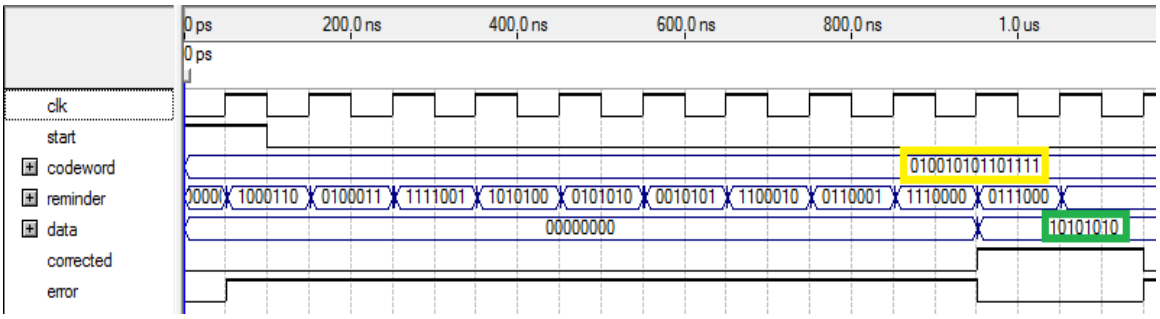


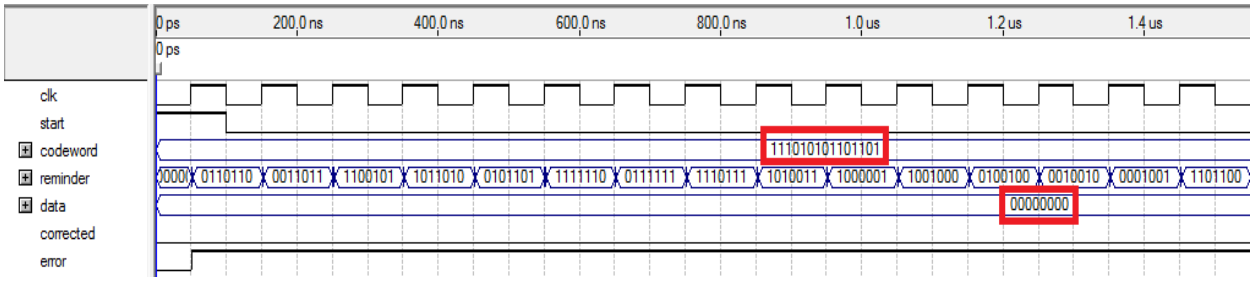
Рисунок 3.46 – Комбинационная схема исправления пакетной ошибки для кода (15, 8, 3)

На рисунке 3.47 приведены диаграммы работы декодера ПК (15, 8, 3), исправляющего пакетные ошибки длиной до 3 бит. Для моделирования заданы разные шаблоны ошибок: а) пакетная ошибка **111**000000000000; б) независимая ошибка 0**1**0000000000000**1**0; в) пакетная ошибка **1111**00000000000.

а)



б)



в)

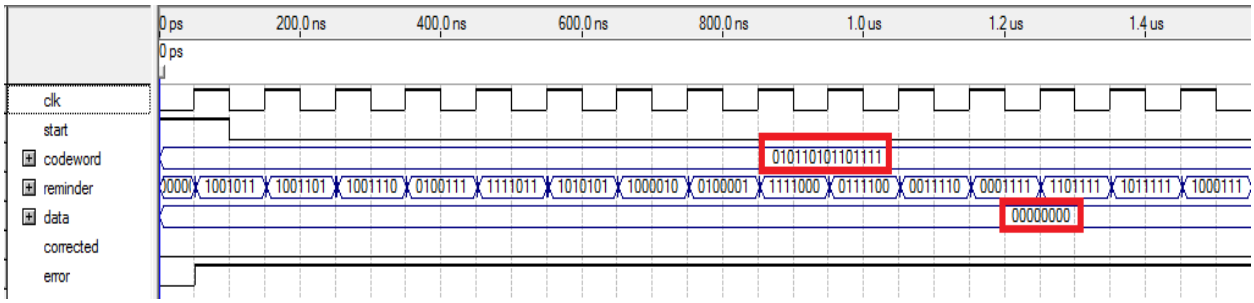


Рисунок 3.47 – Диаграмма работы декодера ПК (15, 8, 3). Классическая реализация циклического алгоритма: а) пакетная ошибка 111000000000000; б) независимая ошибка 010000000000010; в) пакетная ошибка 111100000000000

По диаграммам на рисунке 3.47 можно установить, что устройство на основе ЦПК (15, 8, 3) с классической реализацией циклического алгоритма декодирования позволяет исправлять пакет ошибок длиной до 3-х бит. При попытке исправить пакет большей длины или независимые ошибки на выходе устройства формируется сигнал ошибки.

На рисунке 3.48 приведены характеристики декодера ПК (15, 8, 3) из САПР Quartus II для ПЛИС Stratix III EP3SL70F780C2.

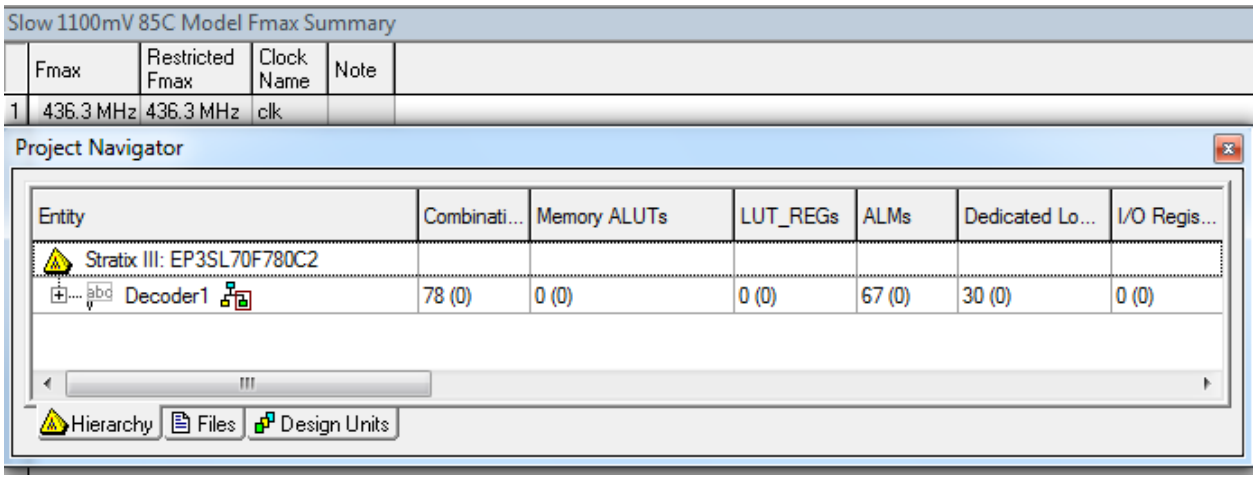


Рисунок 3.48 – Характеристики декодера ПК (15, 8, 3). Классическая реализация циклического алгоритма

Частота работы декодера составляет 436 МГц, что превышает частоту работы декодера БЧХ кода (15, 7, 5) (336 МГц).

При классической реализации циклического алгоритма декодирования для исправления пакетных ошибок время работы (также как и для независимых ошибок) зависит от позиций ошибок. В таблице 3.15 приведено количество тактов и время при частоте 436 МГц (рисунок 3.48), необходимое для исправления пакетной ошибки длиной 3 в различных позициях кодового слова.

Таблица 3.15 – Быстродействие исправления пакетной ошибки в зависимости от позиций ошибки в трех битах

Шаблон ошибки	**Такты**	**Время, нс**
000000000000111	2	4,6
000000000001110	2	4,6
000000000011100	2	4,6
000000000111000	2	4,6
000000001110000	2	4,6
000000011100000	3	6,9
000000111000000	4	9,2
000001110000000	5	11,5
000011100000000	6	13,8
000111000000000	7	16,1
001110000000000	8	18,4
011100000000000	9	20,7
111000000000000	10	23

Таким образом, максимальное время исправления пакетной ошибки при частоте 436 МГц составляет 23 нс.

Параллельная реализация алгоритма на ПЛИС

Циклический алгоритм декодирования, применяемый для пакетных ошибок, можно распараллелить по аналогии с декодером, исправляющим независимые ошибки. На рисунке 3.49 приведена структурная схема декодера с параллельной реализацией.

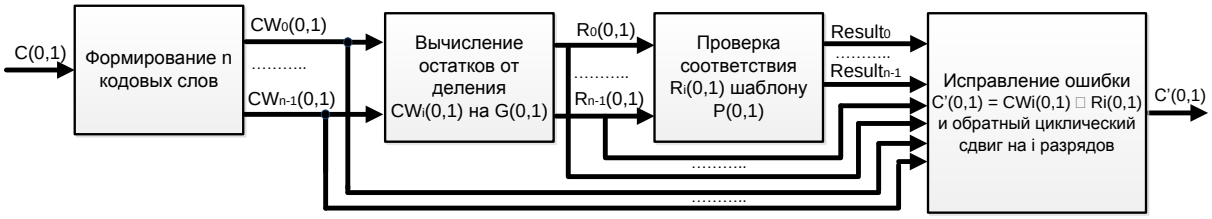


Рисунок 3.49 – Структурная схема параллельного декодера ПК (15, 8, 3)

Кодовое слово *C(0,1)* подаётся на блок генерации всех вариантов сдвигов исходного кодового слова, после чего сформированные кодовые слова (*C_0(0,1)* – без сдвига, *C_{n-1}(0,1)* – соответствует кодовому слову, сдвинутому на *n*-1 раз) параллельно подаются на блок вычисления остатков от деления. Рассчитанные для каждого из *n* вариантов кодового слова остатки *R_0(0,1) – R_{n-1}(0,1)* подаются на блок проверки на совпадение с одним из шаблонов ошибок. Если остаток *R_i* совпадает с одним из шаблонов ошибок, то на выходе («*Resulti*») формируется логический «1», иначе формируется логическая «0». Если сигнал «*Resulti*» равен «1», то кодовое слово *Ci(0,1)* суммируется по модулю 2 с остатком *Ri(0,1)*. Затем выбирается результат, соответствующий первой единице сигналов *Ri(0,1)* (поиск производится от 0 до *n*-1). Выбранный результат сдвигается циклически обратно на *i* разрядов и подаётся на выход декодера [105–107].

На рисунке 3.50 приведена диаграмма работы декодера ПК (15, 8, 3) с параллельной реализацией циклического алгоритма декодирования. Для моделирования заданы следующие шаблоны ошибок: пакетная 3 бита

111000000000000, однократная 010000000000000, пакетная 4 бита: **111**100000000000.

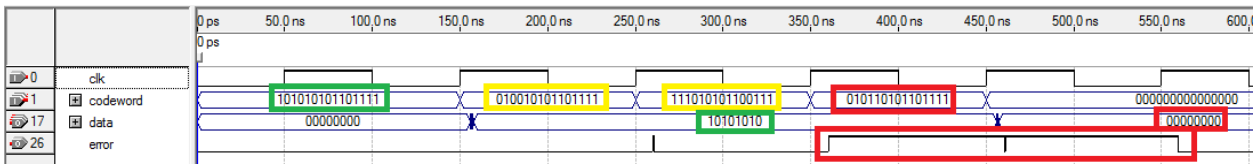


Рисунок 3.50 – Диаграмма работы декодера ПК (15, 8, 3). Параллельная реализация циклического алгоритма

На рисунке 3.51 приведены характеристики декодера ПК (15, 8, 3) из САПР Quartus II для ПЛИС Stratix III EP3SL70F780C2.

Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	241.78 MHz	241.78 MHz	clk	

Entity	Combinational ALUTs	Block Memory Bits	Dedicated Lo...	ALMs
Stratix III: EP3SL70F780C2				
decoder_packet	178 (0)	0	23 (0)	111 (0)

Рисунок 3.51 – Характеристика декодера ПК (15, 8, 3). Параллельная реализация циклического алгоритма

Частота работы декодера составляет 258 МГц, что превышает частоту работы декодера БЧХ кода (15, 7, 5) (194 МГц), при этом количество логических ячеек меньше на 33% (119).

В главе 4 рассмотрено применение декодера ПК (15, 8, 3) в устройстве для обнаружения и исправления пакетных ошибок при передаче команд в подсистеме синхронизации системы управления электрофизической установки Токамак КТМ.

3.7.2. Разработка устройств на ПЛИС для исправления пакетных ошибок с применением аналога кода Рида-Соломона (7, 3)

Код Рида-Соломона являются наиболее эффективными помехоустойчивыми кодами для обнаружения и исправления пакетных ошибок [43, 50]. В данном разделе рассмотрена реализация на ПЛИС алгоритма декодирования циклического кода (21, 9, 6), в котором длина кода составляет

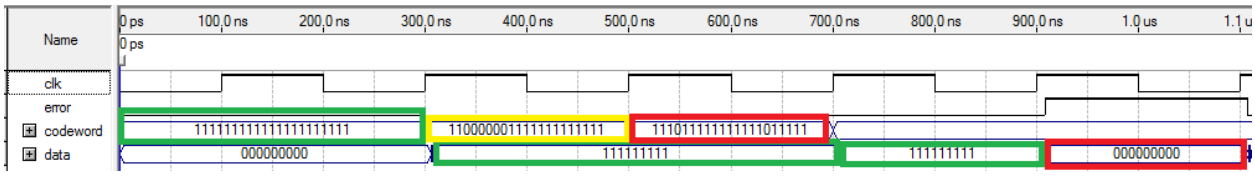
21 бит, длина информационного сообщения – 9 бит, длина исправляемого пакета – 6 бит. Данный код является двоичным аналогом кода Рида-Соломона (7, 3), в котором длина кода составляет 7 символов конечного поля $GF(2^3)$, длина информационного блока 3 символа, а длина исправляемого пакета 2 символа. Образующий полином для кода (21, 9, 6) выбран из таблицы 3.7 по параметрам $m = 9$, $p = 3$ (строка 9, столбец 6), составленной из найденных полиномов с применением предложенного в разделе 3.3 алгоритма поиска.

Табличный алгоритм декодирования

Для циклического кода (21, 9, 6) таблица шаблонов ошибок значительно увеличивается относительно таблицы для кода (15, 8, 3). Длина контрольного блока составляет 12 бит, тогда количество элементов таблицы составит 2^{12}, а объём таблицы – $9 \cdot 2^{12} = 36864$ бит.

На рисунке 3.52 (а,б) приведены диаграммы работы устройства декодирования кода (21, 9, 6), исправляющего пакеты ошибок длиной до 6 бит.

а)



б)

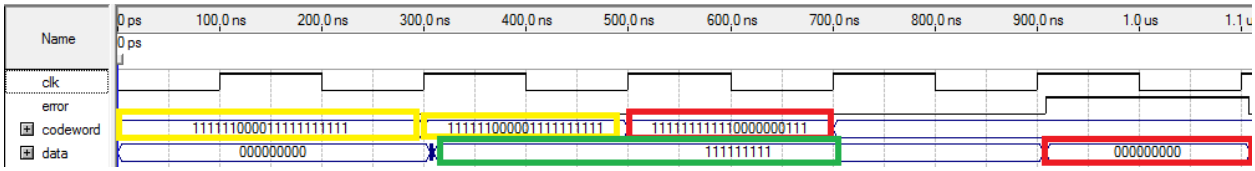


Рисунок 3.52 – Диаграммы работы декодера ПК (21, 9, 6). Табличный алгоритм: а) КС без ошибки 111111111111111111111, КС с ошибкой 001111110000000000000, КС с ошибкой 000100000000000100000; б) КС с ошибкой 000000111100000000000, КС с ошибкой 000000111110000000000, КС с ошибкой 111111111110000000111

На рисунке 3.53 приведены характеристики декодера ПК (21, 9, 6) с применением табличного алгоритма из САПР Quartus II для ПЛИС Stratix III EP3SL70F780C2.

	Fmax	Restricted Fmax	Clock Name	Note
1	580.05 MHz	580.05 MHz	clk	limit due to minimum period restriction (tmin)

Entity	Combinational ALUTs	Block Memory Bits	Dedicated Lo...	ALMs
Stratix III: EP3SL70F780C2				
decoder_packet_table	26 (0)	36864	30 (0)	18 (0)

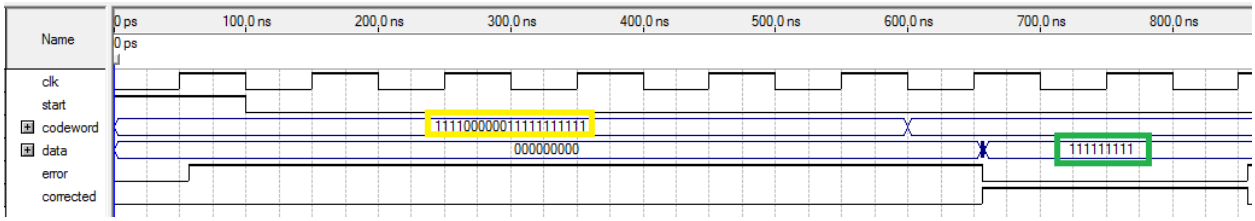
Рисунок 3.53 – Характеристика декодера ПК (21, 9, 6). Табличный алгоритм

Частота работы декодера составляет 580 МГц, что немного больше частоты работы декодера кода (15, 8, 3) (565 МГц), при этом объём памяти для хранения таблицы составляет 36864 бит.

Циклический алгоритм декодирования

Для кода (21, 9, 6) циклический алгоритм декодирования позволяет не хранить большой объем шаблонов ошибок в памяти ROM. Реализация устройства декодирования с циклическим алгоритмом для кода (21, 9, 6) аналогична процедуре для кода (15, 8, 3). На рисунке 3.54 (а,б) приведены диаграммы работы устройства декодирования.

а)



б)

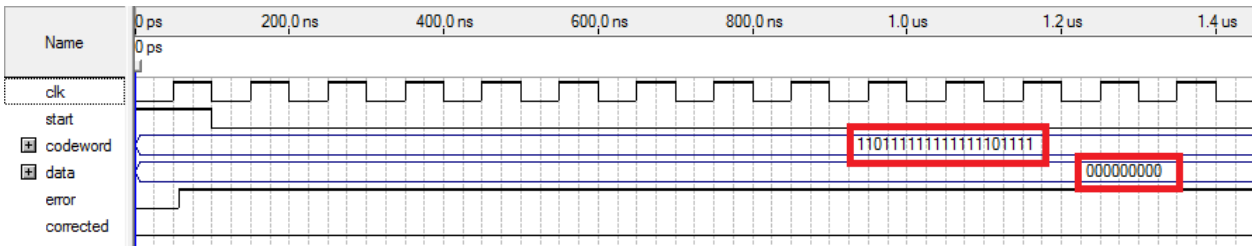


Рисунок 3.54 – Диаграммы работы декодера ПК (21, 9, 6). Классическая реализация циклического алгоритма: а) КС с ошибкой 000011111100000000000; б) КС с ошибкой 001000000000000001000

На рисунке 3.55 приведены характеристики декодера ПК (21, 9, 6) из САПР Quartus II для ПЛИС Stratix III EP3SL70F780C2 с применением классической реализации циклического алгоритма декодирования.

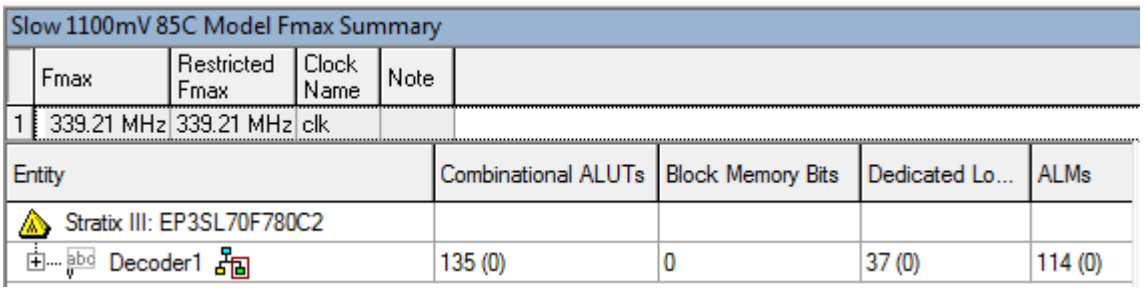


Рисунок 3.55 – Характеристика декодера ПК (21 ,9, 6). Классическая реализация циклического алгоритма

Частота работы декодера составляет 327 МГц, что примерно соответствует частоте работы декодера кода (15, 8, 3) (331 МГц), при этом количество логических ячеек составляет 110, что примерно в 2 раза больше, чем для декодера кода (15, 8, 3).

В таблице 3.16 приведено количество тактов, необходимых для исправления ошибки. Максимальное время исправления пакетной ошибки длиной 6 бит при частоте 327 МГц составляет 33,6 нс.

Таблица 3.16 – Быстродействие исправления пакетной ошибки длиной 6 бит в зависимости от позиций ошибок

Шаблон ошибки	**Такты**	**Время, нс**
000000000000000111111	2	6,1
000000000000001111110	2	6,1
000000000000011111100	2	6,1
000000000000111111000	2	6,1
000000000001111110000	2	6,1
000000000011111100000	2	6,1
000000000111111000000	2	6,1
000000001111110000000	3	9,15
000000011111100000000	4	12,2
000000111111000000000	5	15,25
000001111110000000000	6	18,3
000011111100000000000	7	21,35
000111111000000000000	8	24,4
001111110000000000000	9	27,45
011111100000000000000	10	30,5
111111000000000000000	11	33,55

При параллельной реализации устройства декодирования можно увеличить быстродействие процесса декодирования за счёт увеличения количества применяемых логических ячеек. На рисунке 3.56 приведена диаграмма работы устройства декодирования для кода (21, 9, 6).

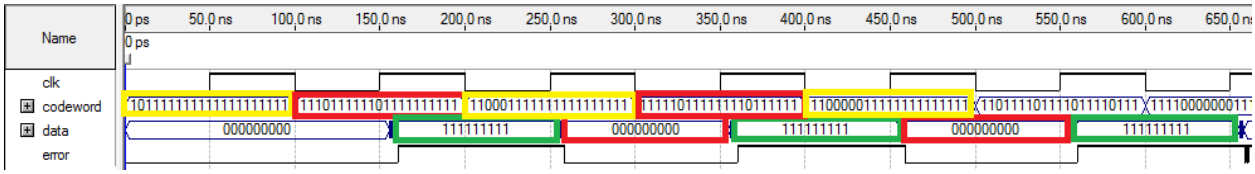


Рисунок 3.56 – Диаграммы работы декодера ПК (21, 9, 6). Циклический алгоритм. Параллельная реализация: КС с ошибкой 010000000000000000000; КС с ошибкой 000100000100000000000, КС с ошибкой 001110000000000000000, КС с ошибкой 000001000000001000000, КС с ошибкой 001111100000000000000

На рисунке 3.57 приведены характеристики декодера ПК (21, 9, 6) из САПР Quartus II для ПЛИС Stratix III EP3SL70F780C2 с применением параллельной реализации циклического алгоритма декодирования.

	Fmax	Restricted Fmax	Clock Name	Note
1	187.41 MHz	187.41 MHz	clk	

Entity	Combinational ALUTs	Block Memory Bits	Dedicated Lo...	ALMs
Stratix III: EP3SL70F780C2				
decoder_packet	715 (0)	0	30 (0)	475 (0)

Рисунок 3.57 – Диаграммы работы декодера ПК (21, 9, 6). Циклический алгоритм. Параллельная реализация

Частота работы декодера составляет 184 МГц, что на 30% ниже частоты работы декодера кода (15, 8, 3) (258 МГц), при этом количество логических ячеек равно 485, что примерно в 4 раза больше, чем для декодера кода (15, 8, 3).

В таблице 3.17 приведены характеристики декодеров ПК (15, 8, 3) и ПК (21,9,6) при реализации на ПЛИС Stratix III EP3SL70F780C2.

Таблица 3.17 – Характеристики устройств декодирования

Алгоритм	**Макс. частота, МГц**	**Такты**	**Мин. время, нс**	**Кол-во ячеек**	**Память ROM, бит**
Декодер кода БЧХ (15, 7, 5)					
Табличный	429	3 (1)	6,9 (2,3)	19	1792
Циклический (классический)	333	2–16	6–48	70	0
Циклический (параллельный)	194	2 (1)	10,2 (5,1)	179	0
Декодер кода (15, 8, 3)					
Табличный	565	3 (1)	5,4 (1,8)	17	1024
Циклический (классический)	436	2–10	4,6–22,9	67	0
Циклический (параллельный)	241	2 (1)	8,2 (4,1)	111	0
Декодер кода (21, 9, 6)					
Табличный	580	3 (1)	5,1 (1,7)	18	36864
Циклический (классический)	339	2 – 11	5,9 – 31,2	114	0
Циклический (параллельный)	187	2 (1)	10,6 (5,3)	475	0

Из таблицы 3.17 видно, что при одинаковой длине кодового слова n = 15, декодер, исправляющий пакетные ошибки длины до 3, работает на большей частоте и при параллельной реализации циклического алгоритма декодирования требует меньше аппаратных ресурсов.

Устройства декодирования кода (21, 9, 6) работают с более низкой частотой (за исключением табличного алгоритма) и требуют для реализации больше аппаратных ресурсов, но при этом позволяют исправлять пакет ошибок длиной до 6 бит в сообщении длиной 9 бит.

В главе 4 рассмотрено применением декодера ПК (15, 8, 3) для обнаружения и исправления пакетных ошибок при передаче команд в подсистеме синхронизации системы управления электрофизической установки Токамак КТМ [69].

3.8. Основные результаты и выводы по главе

1. Разработан алгоритм поиска образующих полиномов, отличающийся от известных тем, что позволяет находить полиномы для построения циклических помехоустойчивых кодов более эффективных по скорости кода, чем коды БЧХ.
2. Поставлен компьютерный эксперимент по поиску образующих полиномов с применением технологий параллельных вычислений для построения кодов, исправляющих независимые и пакетные ошибки. Установлено, что разработанный алгоритм поиска позволяет находить полиномы для построения кодов длиной до 5 бит короче, чем коды БЧХ, при длине информационного блока до 32 бит и кратности исправляемых ошибок до 4. Установлено, что применение технологий параллельных вычислений целесообразно для большинства из исследуемых длин информационного блока.
3. Проведено исследование аппаратных реализаций на ПЛИС декодеров помехоустойчивого кода БЧХ (15, 7, 5) с различными алгоритмами декодирования. Установлено, что циклический алгоритм декодирования при реализации на ПЛИС обладает лучшим быстродействием по сравнению с известными алгоритмами на основе БМА за счёт простых операций проверки веса остатка и сложений по модулю 2, что при параллельной реализации на ПЛИС требует наименьшего количества тактов.
4. Проведено исследование аппаратных реализаций на ПЛИС модифицированного циклического алгоритма декодирования для предложенного кода (17, 9, 5), который является более эффективным кодом по скорости, чем код БЧХ (15, 7, 5), и позволяет реализовать более эффективные по быстродействию и аппаратным затратам устройства исправления ошибок, чем с применением укороченного кода БЧХ (19, 9, 5).

5. Разработаны декодирующие устройства циклического помехоустойчивого кода (15, 8, 3) на ПЛИС, исправляющие пакетные ошибки длиной до 3 бит с применением табличного и циклического алгоритмов декодирования. Установлено, что декодеры обладают лучшим быстродействием и меньшими аппаратными затратами, чем декодеры БЧХ кода (15, 7, 5) с аналогичной длиной кодового слова.
6. Разработаны декодирующие устройства циклического помехоустойчивого кода (21, 9, 6), являющегося аналогом кода РС (7, 3), на ПЛИС, исправляющие пакетные ошибки длиной до 6 бит.

ГЛАВА 4. ПРАКТИЧЕСКОЕ ПРИМЕНЕНИЕ УСТРОЙСТВ ОБНАРУЖЕНИЯ И ИСПРАВЛЕНИЯ ОШИБОК НА ПЛИС

Описываются результаты применения разработанных устройств на ПЛИС исправления пакетных ошибок и вычисления контрольной суммы CRC8 для решения прикладных задач обеспечения надёжности передаваемых команд в системе управления установкой Токамак КТМ, а также результаты эксперимента по исправлению пакетных и независимых ошибок с применением стенда на основе макетов SDK-6.1.

4.1. Разработка устройства исправления ошибок на ПЛИС для подсистемы синхронизации системы управления электрофизической установки Токамак КТМ

Подсистемы синхронизации и противоаварийной защиты, входящие в состав системы управления электрофизической установки Токамак КТМ [119, 120], предназначены для контроля технологического оборудования и предупреждения аварийных ситуаций. В последнем случае системы должны обеспечивать перевод установки в безопасное состояние и снижение последствий аварий. Процесс функционирования установки токамак КТМ характеризуется сложной последовательностью технологических операций. Выполнение этих операций должно быть синхронизировано по времени. В качестве основных структурных элементов подсистемы синхронизации выделяют: центральный блок синхронизации (ЦБС), локальные модули синхронизации (ЛМС) и цифровая сеть передачи данных и синхросигналов (рисунок 4.1). В ЦБС есть модуль «Формирователь кодов команд», который формирует и передает команды к локальным модулям синхронизации. Для обеспечения надёжности передаваемых команд необходима реализация способов и алгоритмов обнаружения ошибок в соответствующих модулях отправки (кодер) и приёма команд (декодер).

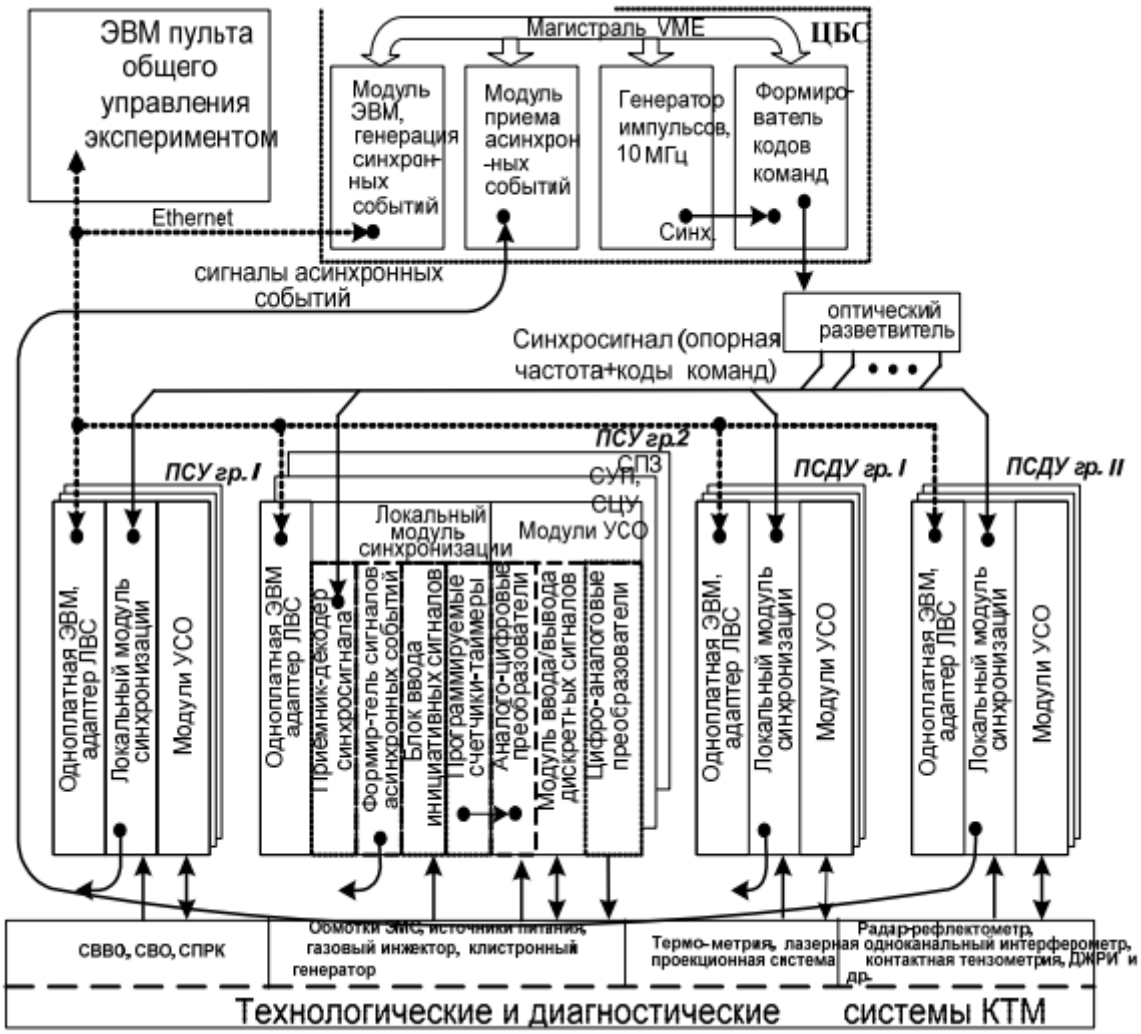


Рисунок 4.1 – Структурная схема подсистемы синхронизации КТМ

Команды передаются в виде пакета, формат которого представлен на рисунке 4.2.

START	TYPE	COP	CS
55h	01h	Код Операции	Контрольная сумма

Рисунок 4.2 – Структура команды от ЦБС к ЛМС

Пакет имеет следующие поля [101, 119]: START (1 байт) – начало кадра; TYPE (1 байт) – тип операции; COP (1 байт) – код операции; CS (1 байт) – контрольная сумма.

Формат команды содержит поле контрольной суммы CRC8, которая позволяет обнаруживать ошибки. Однако, для подсистем синхронизации важно

обеспечение передачи команды без повторного запроса, поэтому помимо контрольной суммы необходимо кодировать пакет помехоустойчивым кодом. В соответствии с техническим заданием по обеспечению целостности данных для данной системы предложен помехоустойчивый код, исправляющий пакетные ошибки длиной до 3 бит, что соответствует коду (15, 8, 3) [69, 101].

4.1.1. Разработка устройства исправления ошибок

На рисунке 4.3 приведена структурная схема устройства исправления ошибок с применением ПК (15, 8, 3) для пакета данных по рисунку 4.2.

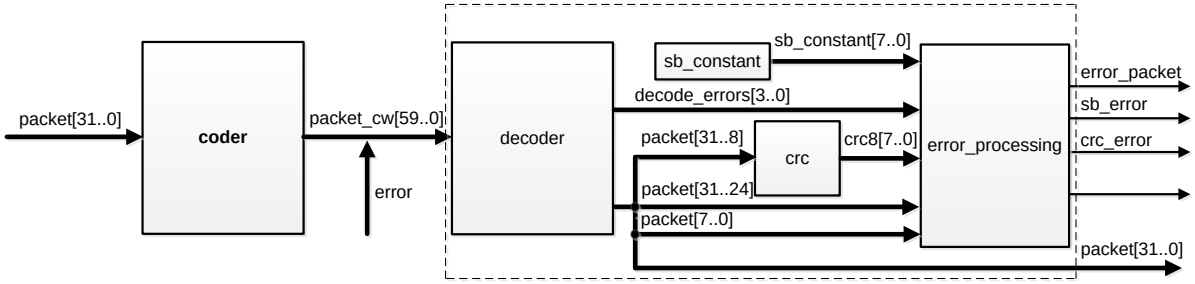


Рисунок 4.3 – Структурная схема кодирующего устройства кода (15, 8, 3)

Устройство состоит из кодера (рис. 4.4) и декодера, соединённых каналом связи. Декодер включает в состав модули обнаружения ошибок (crc8), исправления ошибок (decoder1) и компаратор для формирования ошибки.

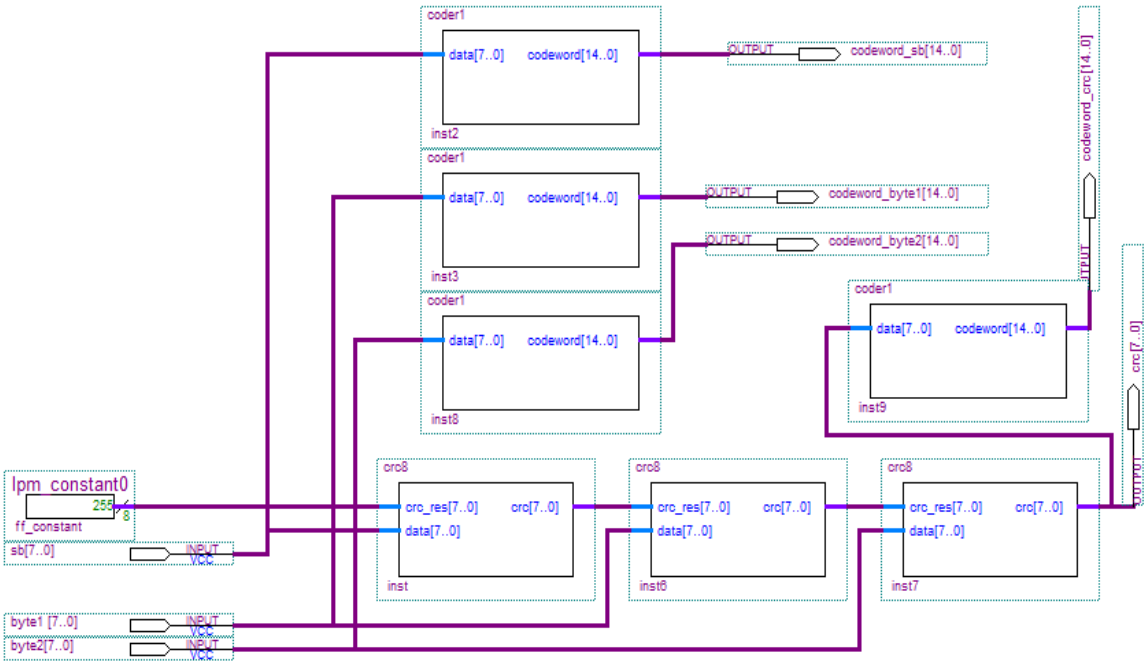


Рисунок 4.4 – Функциональная схема кодирующего устройства кода (15, 8, 3)

На вход кодирующего устройства поступает пакет данных (packet [31..0]) для которого рассчитывается контрольная сумма CRC8 (всего 4 байта). Данный пакет кодируется по байтам и по каналу связи передается избыточный пакет (packet_cw [59..0]) длины 60 бит (рисунок 4.5).

8 бит	7 бит	8 бит	7 бит	8 бит	7 бит	8 бит	7 бит
START	START_K	TYPE	TYPE_K	COP	COP_K	CS	CS_K
55h	Контрольный блок	01h	Контрольный блок	Код операции	Контрольный блок	Контрольная сумма	Контрольный блок

Рисунок 4.5 – Закодированный пакет команды

Кодирование данных (модуль ЦБС) осуществляется согласно выражению: $CW(0,1) = M(0,1)* 2^k \mid mod((M(0,1) * 2^k)/G(0,1)))$, где:

M(0,1) – байт данных;

G(0,1) – образующий полином 11010001;

k = n – m = 7 – длина контрольного (избыточного) блока.

На приемной стороне декодирующее устройство получает пакет с ошибками (либо без ошибок) и осуществляет процесс исправления (декодирования). На выходе декодера образуются массив ошибок (decode_errors [3..0]) и исправленный пакет данных (packet [31..0]). При этом исправленный пакет разбивается на части: packet [7..0] – контрольная сумма CRC, packet [31..8] – старт-байт, packet [31..24] – данные и старт-байт. При этом, packet [31..24] поступает на блок расчета CRC8.

Сформированные старт-байт (sb_constant [7..0]), массив ошибок, рассчитанная контрольная сумма CRC8, packet [31..24] и декодированная контрольная сумма CRC8 поступают на блок обработки ошибок (error_processing). Последний на основе полученных данных формирует 4 типа ошибки: ошибка пакета (error_packet) – обнаружена не пакетная ошибка, ошибка старт-байта (sb_error), ошибка контрольной суммы (crc_error) – декодированная и рассчитанная контрольные суммы не совпадают, ошибка декодирования (decoder_error) – кратность ошибки больше 3. Функциональная схема устройства декодирования приведена на рисунке 4.6.

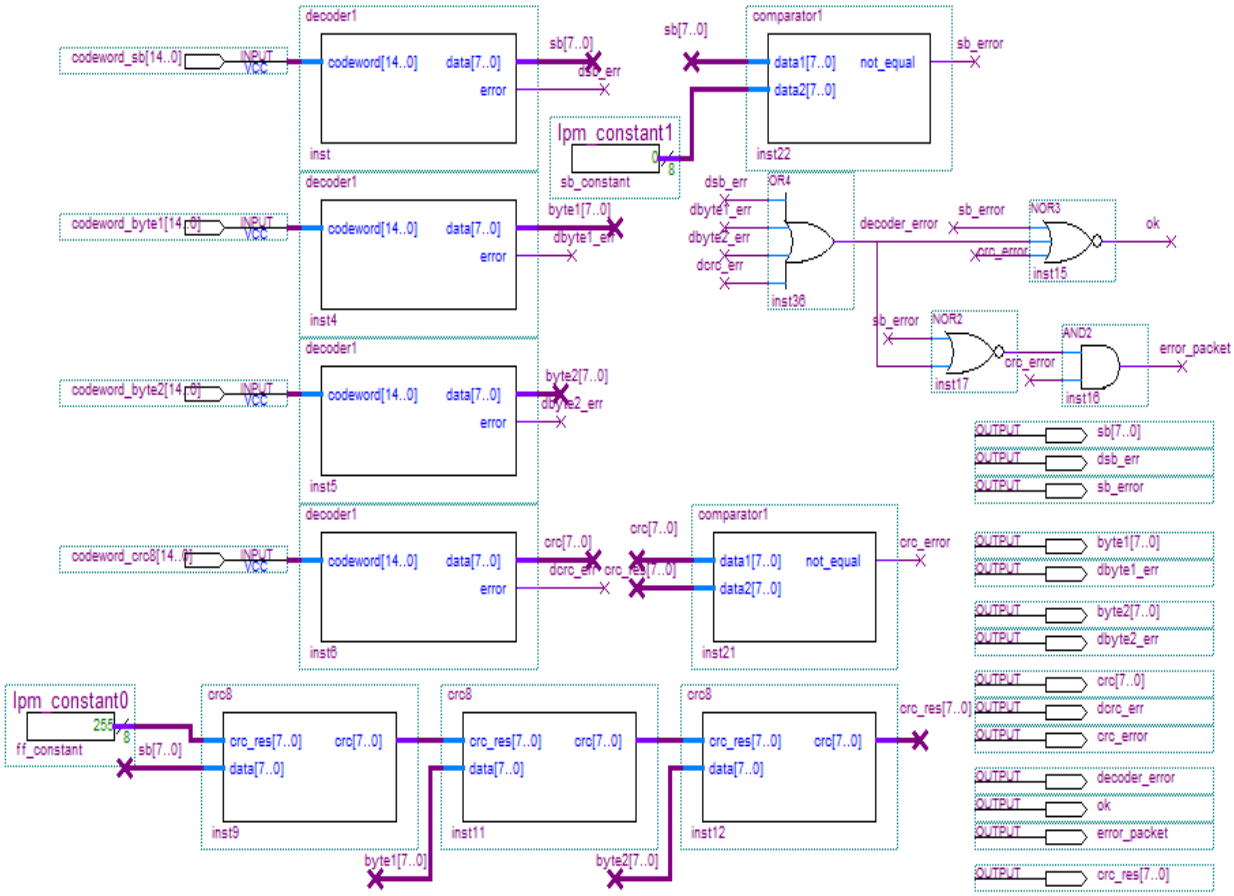


Рисунок 4.6 – Функциональная схема декодирующего устройства кода (15, 8, 3)

4.1.2. Работа устройства исправления ошибок

Для проверки работоспособности кодирующего устройства на вход были поданы следующие данные: старт-байт: 00h, 1-байт: AAh, 2-ой байт: 0Fh. На выходе кодера сформированы кодовые слова длиной 15 бит. На рисунке 4.7 представлены результаты работы кодера.

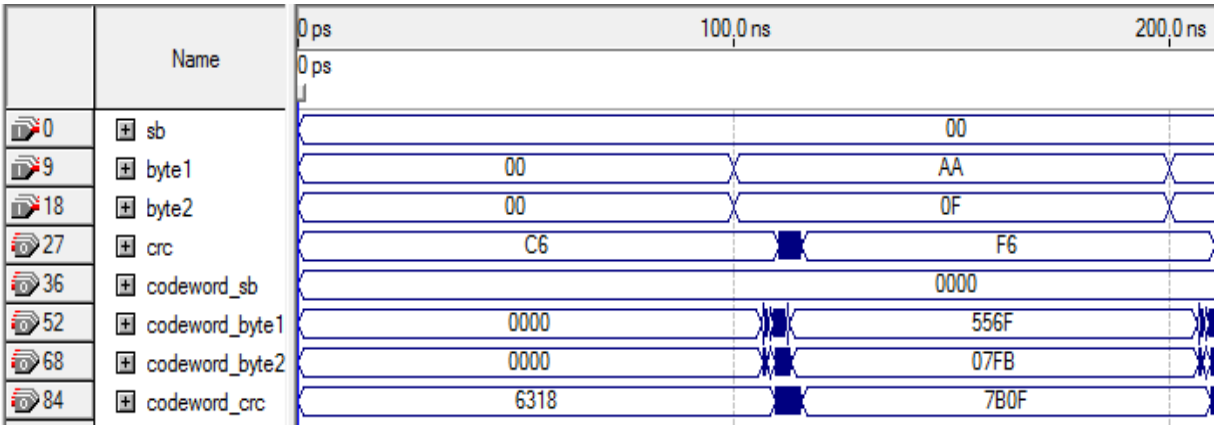


Рисунок 4.7 – Диаграмма работы кодирующего устройства кода (15, 8, 3)

Для проверки работы декодера сформированы кодовые слова с различными типами ошибок (см. codeword_byte1, рисунок 4.8): кодовое слово без ошибок (556F) (101010101101111), с трёхкратной пакетной ошибкой (5568) (101010101101**000**), двукратной независимой (546E) (101010**0**01101111**0**), четырёхкратной пакетной (5560) (101010101110**0000**). Все кодовые слова представлены в шестнадцатеричной системе счисления.

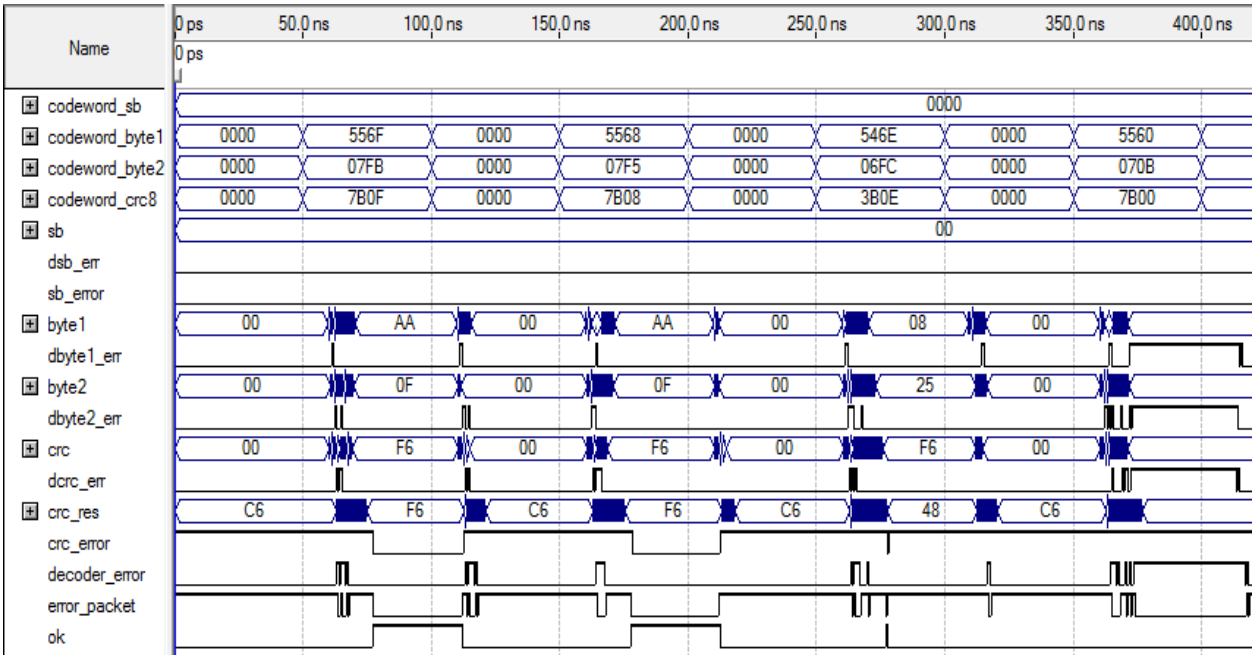


Рисунок 4.8 – Диаграмма работы декодирующего устройства кода (15, 8, 3)

По результатам работы декодера можно сказать, что для кодового слова без ошибки и с трёхкратной пакетной ошибкой на выходе устройства получены исходные байты данных, то есть ошибка исправляется. Для четырёхкратной пакетной ошибки сформирован сигнал *error*, сигнализирующий о неисправимой ошибке из-за превышения заданной кратности.

На рисунке 4.9 (а,б) приведены характеристики устройств кодирования и декодирования команд. Устройства реализованы на базе ПЛИС Cyclone EP1C12Q240C6.

а)

Timing Analyzer Summary

	Type	Slack	Required Time	Actual Time	From	To	From Clock	To Clock	F. P
1	Worst-case tpd	N/A	None	15.939 ns	sb[4]	codeword[10]	--	--	0
2	Total number of failed paths								0

Entity	Logic Cells	LC Registers	Memory Bits	M4Ks
Cyclone: EP1C12Q240C6				
coder	70 (0)	0	0	0

б)

Timing Analyzer Summary

	Type	Slack	Required Time	Actual Time	From	To	From Clock	To Clo
1	Worst-case tpd	N/A	None	34.983 ns	codeword_byte2[10]	crc_error	--	--
2	Total number of failed paths							

Entity	Logic Cells	LC Registers	Memory Bits	M4Ks
Cyclone: EP1C12Q240C6				
Decoder	947 (3)	0	0	0

Рисунок 4.9 – Характеристики устройств на ПЛИС а) кодирующее устройство; б) декодирующее устройство

Для устройства кодирования требуется 70 логических ячеек ПЛИС (Logic Cells). Время кодирования составляет 15,939 нс (асинхронный режим). Для устройства декодирования требуется 947 логических ячеек ПЛИС (Logic Cells). Время декодирования составляет 34,983 нс (асинхронный режим). При частоте синхронизации локальной шины ЛМС 10 МГц (рисунок 4.1) быстродействие декодирования составит не более 1 такта синхросигнала.

4.2. Проверка работы устройств декодирования на экспериментальном стенде

4.2.1. Описание эксперимента

Для проверки работы устройств был собран экспериментальный стенд на основе макетов SDK-6.1 [121] с реальными каналами передачи данных. Макет SDK-6.1 включает в состав ПЛИС Cyclone EP1C3T144C8, жидкокристаллический индикатор (ЖКИ), блок движковых переключателей, блок светодиодов, последовательный порт RS-232 и порт дискретных сигналов (DIO) (рис. 4.10).

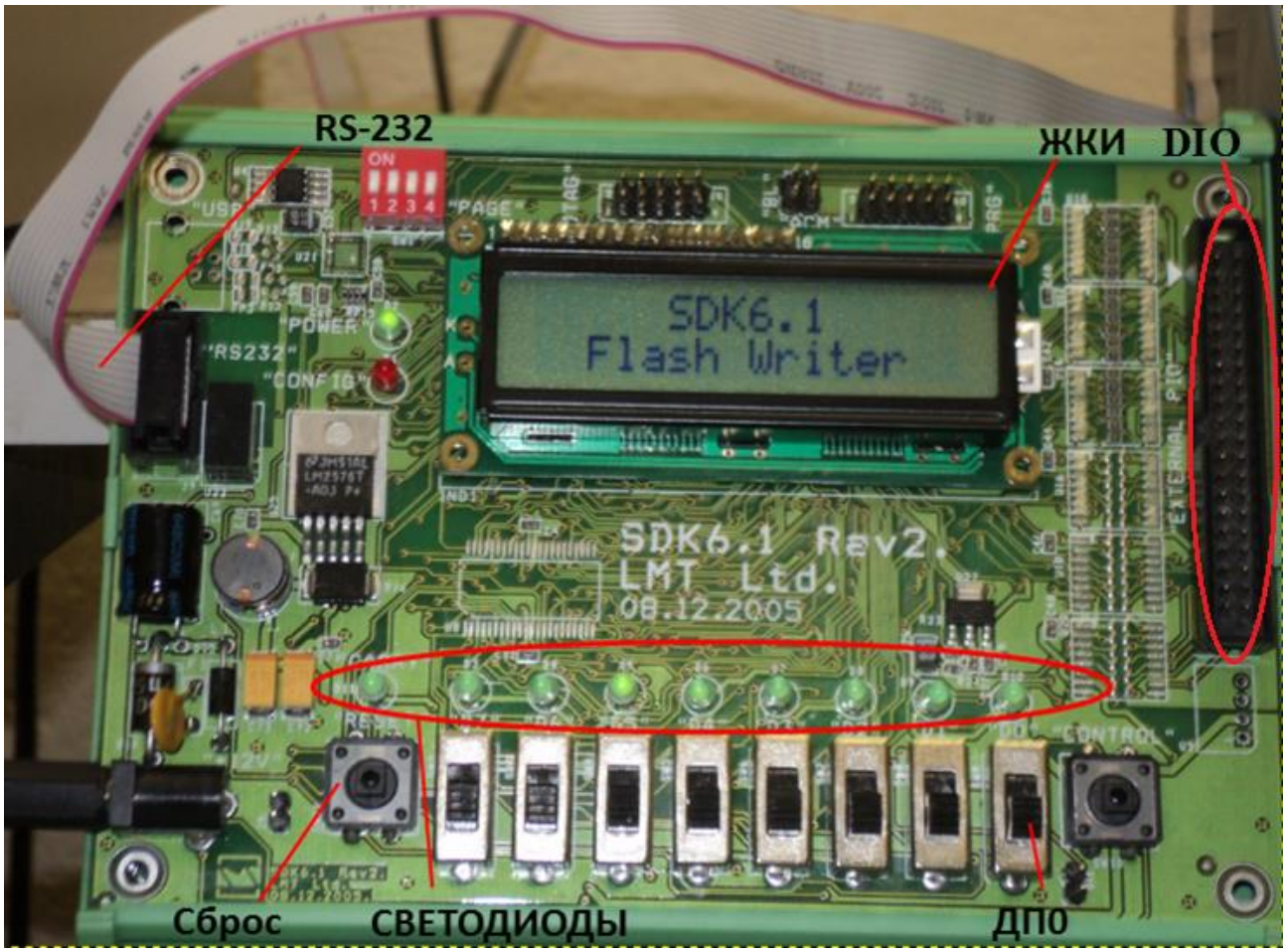


Рисунок 4.10 – Макет SDK-6.1

Структурная схема экспериментального стенда приведена на рис. 4.11. Экспериментальный стенд состоит из 3 макетов: устройство кодирования данных «Кодер», устройство, выполняющее роль канала связи с помехами «Ошибка» и устройство декодирования данных «Декодер».

Все устройства подключены 40-контактным кабелем через интерфейс дискретного ввода-вывода. Для передачи данных используются LVDS-сигналы.

Устройство «Кодер» осуществляет кодирование сообщения, задаваемого в двоичной системе счисления с помощью движковых переключателей. По нажатию кнопки «Control» закодированное сообщение передается на устройство «Ошибки», которое создает псевдослучайный шаблон ошибки. Полученный шаблон складывается по модулю 2 с принятым кодовым словом, формируя, таким образом, кодовое слово с ошибкой.

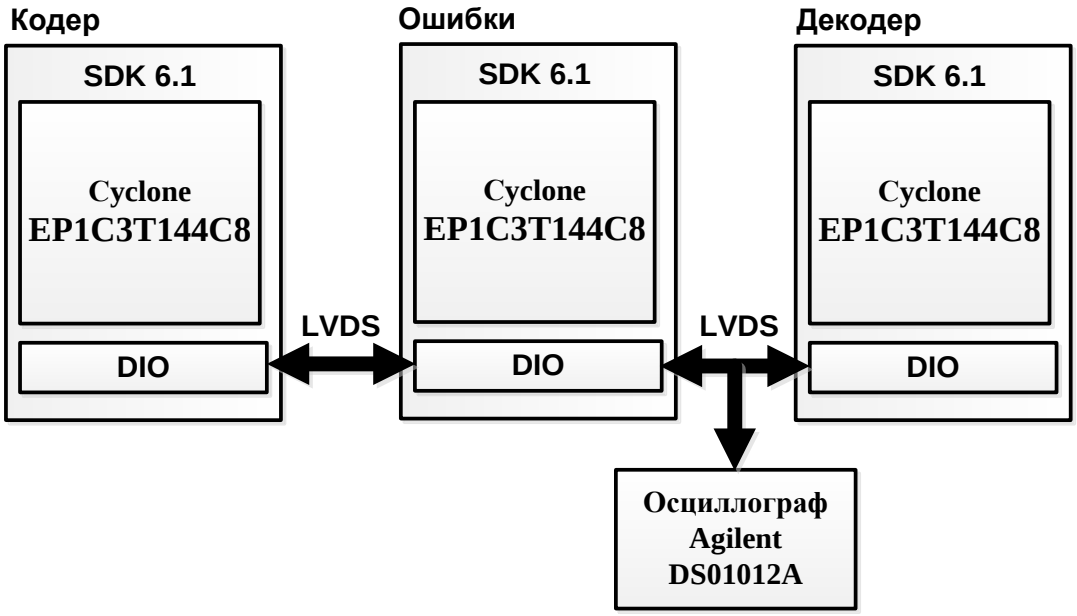


Рисунок 4.11 – Структурная схема стенда

На рис. 4.12. приведена схема формирования псевдослучайного шаблона ошибки для кодового слова. Устройство «Ошибки» после внесения искажений в принятое кодовое слово осуществляет непрерывную отправку сообщения на устройство «Декодер», которое принимает кодовое слово с ошибкой и осуществляет процесс исправления.

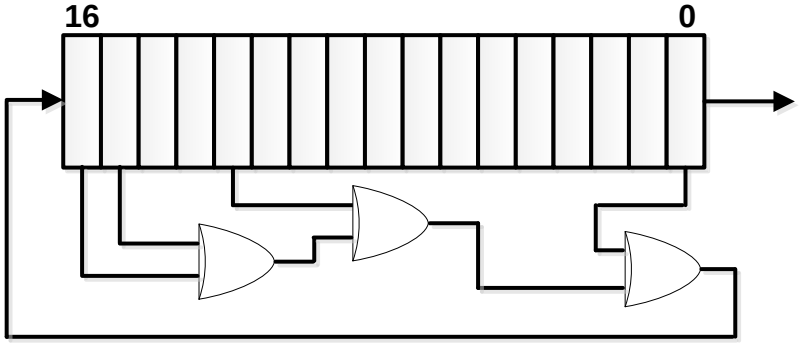


Рисунок 4.12 – Схема формирования шаблона псевдослучайной ошибки в кодовом слове длиной 17 бит

Для проведения эксперимента реализованы по 2 устройства кодирования, генератора «помех» и декодирования на основе ЦПК (17, 9, 5), исправляющего двукратные независимые ошибки и ЦПК (15, 8. 3), исправляющего пакет ошибок длиной до 3 бит. После завершения операции декодирования,

устройство «Декодер» передает исправленное сообщение обратно, что позволяет отобразить искаженное и исправленное сообщение с помощью осциллографа.

К каналу связи от устройства «Ошибки» к устройству «Декодер» подключен двухканальный осциллограф DS01012A с полосой пропускания 100 МГц. Один канал осциллографа подключен к проводу, передающему информацию от устройства «Ошибки» к Декодеру. Второй канал к проводу, передающему информацию обратно. На рис. 4.13 приведены разъемы 40-pin DIO с обозначением передающихся сигналов разных устройств.

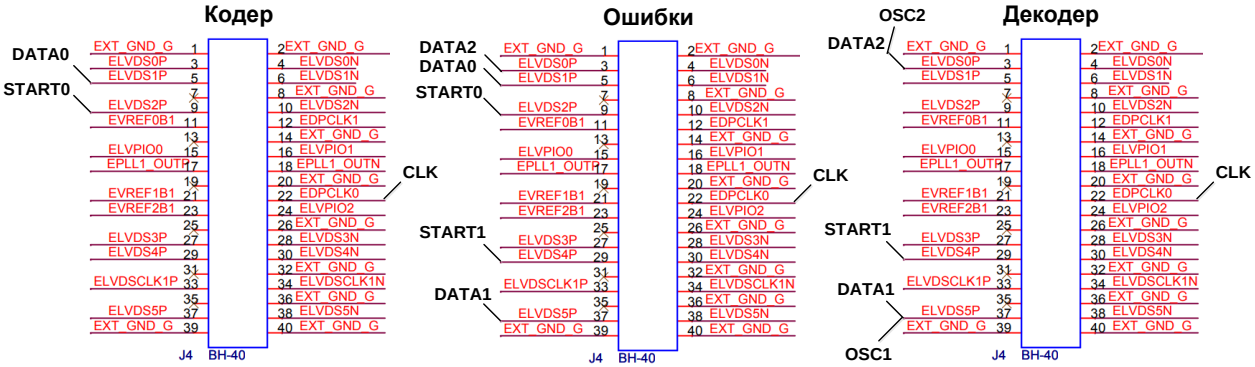


Рисунок 4.13 – Передача сигналов через разъем 40-pin DIO

Обозначение сигналов представлено в таблице 4.1.

Таблица 4.1. Описание передаваемых сигналов

Краткое название	PIN	Описание
DATA0	ELVDS1P (5)	Данные от «Кодера» к «Ошибкам»
START0	ELVDS2P (9)	Сигналы начала работы от «Кодера» к «Ошибками»
DATA1	ELVDS5P (37)	Данные от «Ошибок» к «Декодеру»
START1	ELVDS4P (29)	Сигналы начала работы от «Ошибок» к «Декодеру»
DATA2	ELVDS0P (3)	Данные от «Декодера» к «Ошибками»
OSC1	ELVDS5P (37)	Канал 1 осциллографа
OSC2	ELVDS0P (3)	Канал 2 осциллографа

4.2.2. Исправление двукратной независимой ошибки с применением ЦПК (17,9,5)

Собранный стенд для проведения эксперимента с подключенным осциллографом представлен на рис. 4.14. ЦПК (17, 9, 5) должен гарантированно исправлять двукратные независимые ошибки в кодовом слове длиной 17 бит, при длине информационного блока 9 бит.

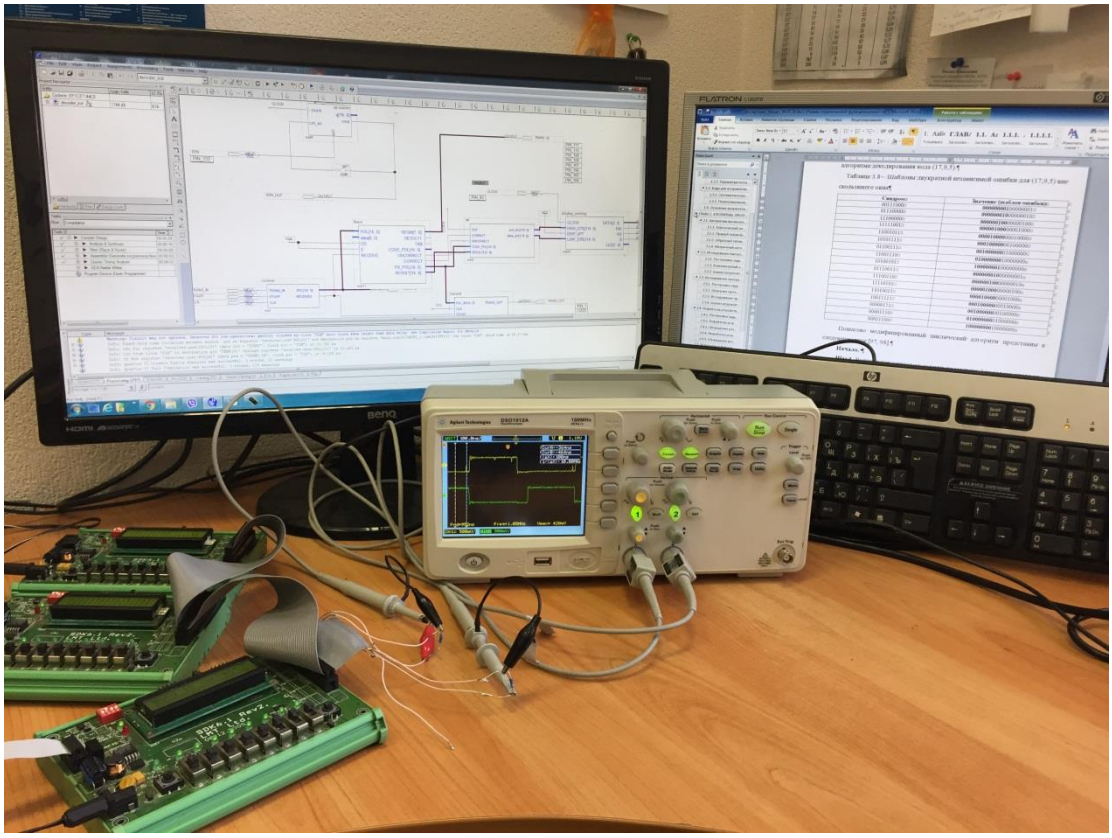


Рисунок 4.14 – Экспериментальный стенд

С персонального компьютера осуществляется загрузка прошивки для ПЛИС в каждый из макетов SDK-6.1. Затем на устройстве «Кодер» задаётся 9-ти битовое сообщение, состоящее из всех единиц (111111111). Выбор такого сообщения обусловлен наилучшей наглядностью при демонстрации работы декодера с применением осциллографа. На рис. 4.15 приведен общий план стенда с отображением сообщений на макетах и осциллографе.

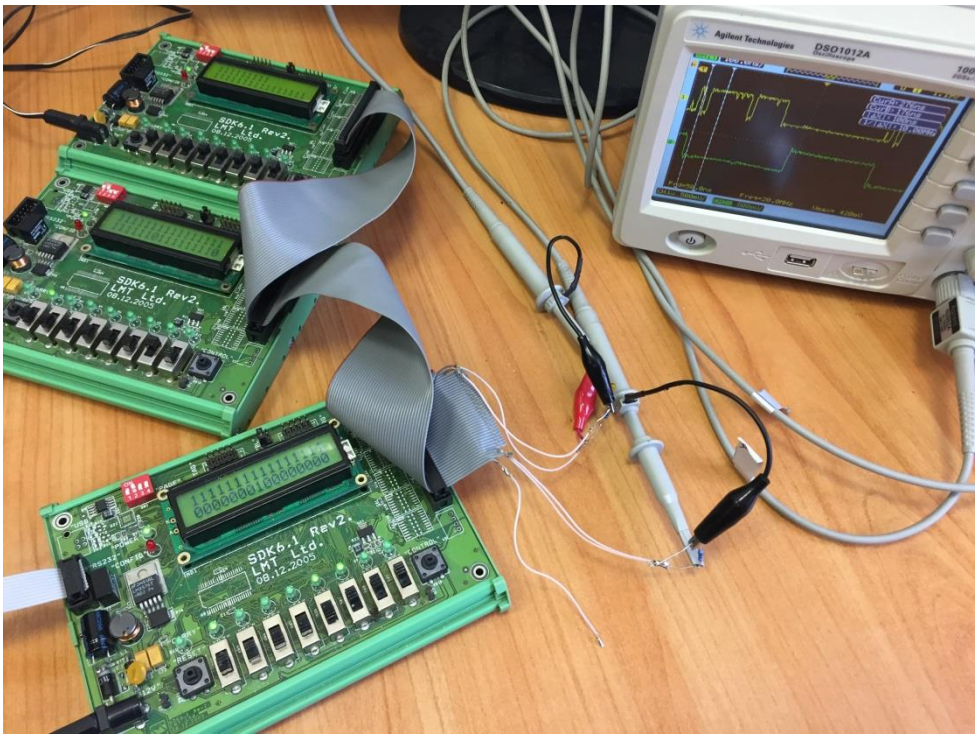


Рисунок 4.15 – Общий план эксперимента с ЦПК (17,9,5)

На рис. 4.16 показаны два варианта передачи сообщений: без ошибок (слева) и с двукратной ошибкой (справа).

Рисунок 4.16 – Передача сообщений без ошибок (слева) и с двукратной ошибкой (справа)

Осциллограммы результатов исправления ошибок представлены на рис. 4.17. Слева показана диаграмма исходного сообщения на 1-м и 2-м канале, справа – исходное (1 канал) и искаженное (2 канал) сообщение.

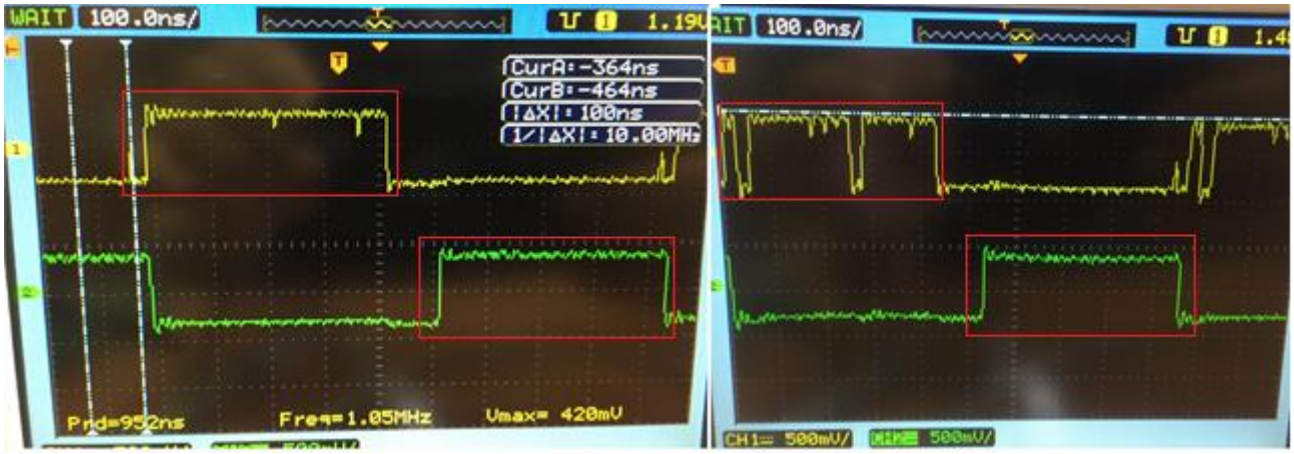


Рисунок 4.17 – Осциллограммы сообщений без искажений (слева) и с двукратной ошибкой (справа)

Устройства работают на частоте 40 МГц, таким образом, 1 бит сообщения передается за 25 нс, что соответствует 1/4 шага (100 нс) сетки диаграммы на рис. 4.17. Все сообщение занимает чуть больше 4 шагов сетки. На диаграмме (рис. 4.17) справа видно, как искажены 2 бита сообщения, что соответствует низкому уровню сигнала (канал 1). После приёма сообщения декодером искаженные биты восстановлены (канал 2).

4.2.3. Исправление пакетной ошибки с применением ЦПК (15,8,3)

По аналогии с экспериментом по передаче закодированных сообщений с ЦПК (17, 9, 5) проведен эксперимент по исправлению пакетных ошибок на основе ЦПК (15, 8, 3), который должен гарантированно исправлять пакет ошибок длиной до 3-х бит в кодовом слове длиной 15 бит, при длине информационного блока 8 бит. На рис. 4.18 показаны варианты передачи сообщения (111111110110000) без ошибок и с пакетной ошибкой длиной 3 бита.

Рисунок 4.18 – Передача сообщений без ошибок (слева) и с пакетной ошибкой (справа)

Осциллограммы результатов исправления ошибок представлены на рис. 4.19. Слева показана диаграмма исходного сообщения на 1-м и 2-м канале, справа – исходное (1 канал) и искаженное (2 канал) сообщение.

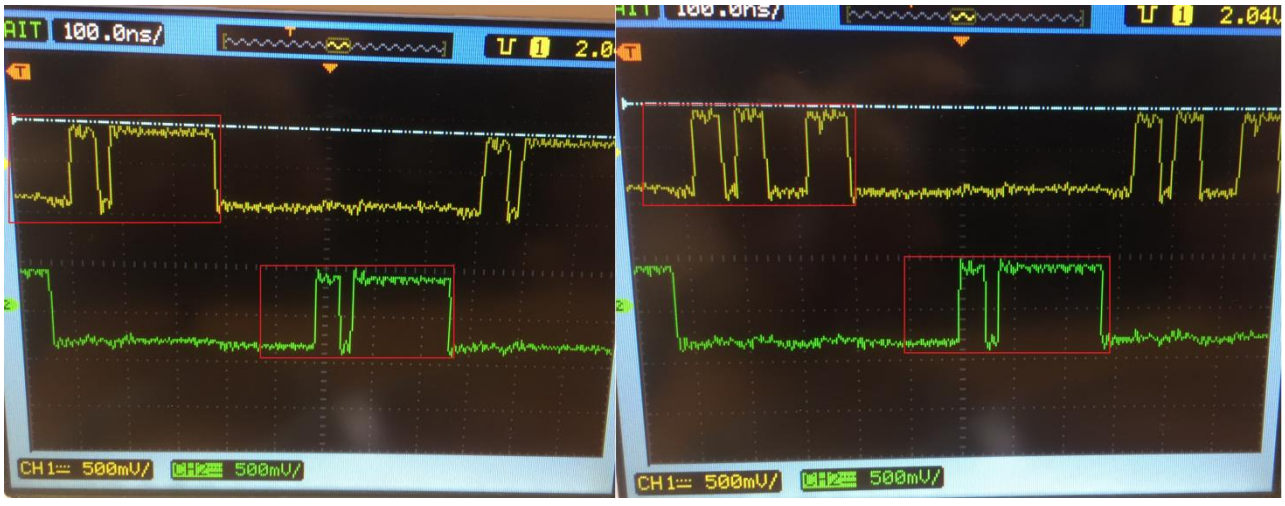


Рисунок 4.19 – Осциллограммы сообщений без ошибок (слева) и с пакетной ошибкой (справа)

На рис. 4.19 видно, как пакет ошибок длиной 3 бита (с 4-й по 6-ю позиции справа) восстанавливается декодером.

4.3. Основные результаты и выводы по главе

В главе рассмотрено применение циклического помехоустойчивого кода (15, 8, 3) совместно с контрольной суммой CRC8 для обнаружения и исправления пакетных ошибок в подсистеме синхронизации системы управления установки Токамак КТМ. Разработанные устройства на ПЛИС осуществляют кодирование команды при отправке и декодирование с исправлением пакетных ошибок длиной до 3 бит на стороне приёма.

Результаты практического применения устройства исправления ошибок с применением циклического помехоустойчивого кода (15, 8, 3) подтверждаются актом о внедрении, представленном в приложении Д.

Для проверки работы устройств исправления ошибок на основе ЦПК (17,9,5) и ЦПК (15,8,3) был собран экспериментальный стенд с использованием макетов SDK-6.1. Результаты эксперимента показали возможность предложенных устройств на ПЛИС исправлять двукратные независимые и трёхкратные пакетные ошибки при передаче информации между устройствами. По результатам экспериментов оформлен акт о внедрении в учебный процесс НИ ТПУ, который представлен в приложении Д.

ЗАКЛЮЧЕНИЕ

В ходе выполнения диссертационной работы получены следующие научные и практические результаты.

1. Проведён анализ существующих алгоритмов обнаружения ошибок без исправления. Обоснована актуальность разработки алгоритмов вычисления контрольной суммы CRC для систем с малым объёмом памяти и низкими вычислительными мощностями.
2. Разработаны устройства на ПЛИС вычисления циклических избыточных кодов CRC8 и CRC32 на основе матричного алгоритма с модификациями, позволяющими увеличить размер блока данных, обрабатываемого за итерацию. Установлено, что матричный алгоритм при программной реализации требует до 32 раз меньший объем памяти для хранения предвычисленных значений (в зависимости от размера блока, обрабатываемого за итерацию), чем при реализации табличного алгоритма. Результаты исследования аппаратных реализаций показали, что при использовании матричного алгоритма быстродействие устройств увеличивается в 2 раза, а изменение длины блока, обрабатываемого за итерацию, позволяет увеличить быстродействие до 8–10 раз по сравнению с устройствами на основе табличного алгоритма.
3. Проведён анализ существующих кодов БЧХ, применяемых для исправления независимых ошибок. Обоснована актуальность разработки алгоритма поиска образующих полиномов для построения циклических помехоустойчивых кодов более эффективных, чем коды БЧХ.
4. Разработан и реализован алгоритм поиска образующих полиномов, адаптированный для параллельных вычислений и отличающийся от известных тем, что позволяет находить полиномы для построения циклических помехоустойчивых кодов более эффективных по скорости кода, чем коды БЧХ.

5. Поставлен компьютерный эксперимент по поиску образующих полиномов с применением технологий параллельных вычислений для построения кодов, исправляющих независимые и пакетные ошибки. Установлено, что разработанный алгоритм поиска позволяет находить полиномы для построения кодов длиной до 5 бит короче, чем коды БЧХ, при длине информационного блока до 32 бит и кратности исправляемых ошибок до 4. Установлено, что применение технологий параллельных вычислений целесообразно для большинства из исследуемых длин информационного блока.
6. Проведено исследование аппаратных реализаций на ПЛИС модифицированного циклического алгоритма декодирования для предложенного кода (17, 9, 5), являющегося более эффективным кодом по скорости, чем код БЧХ (15, 7, 5) и позволяющего реализовать более эффективные по быстродействию и аппаратным затратам устройства исправления ошибок, чем с применением укороченного кода БЧХ (19, 9, 5).
7. Разработаны декодирующие устройства циклического помехоустойчивого кода (15, 8, 3) на ПЛИС, исправляющие пакетные ошибки длиной до 3 бит с применением табличного и циклического алгоритмов декодирования. Установлено, что декодеры обладают лучшим быстродействием и меньшими аппаратными затратами, чем декодеры БЧХ кода (15, 7, 5) с аналогичной длиной кодового слова.
8. Разработано устройство исправления пакетных ошибок длиной до 3 бит с применением модифицированного циклического алгоритма декодирования для решения задачи по обеспечению надежности передаваемых команд подсистемы синхронизации системы управления электрофизической установки Токамак КТМ.

СПИСОК ЛИТЕРАТУРЫ

1. Хэмминг Р.В. Коды с обнаружением и исправлением ошибок. / Р.В. Хэмминг. – М. : ИЛ, 1956. – 23 с.
2. Bose R.C. On a class of error correcting binary group codes / R.C. Bose, D.K. Ray-Chaudhuri // Information and Control. – 1960. – Vol. 3. – Iss. 1. – P.: 68-79.
3. Hocquenghem A. Codes correcteurs d'erreurs // Chiffres, Paris, French, 1959. – Vol. 2. – P. 147–156.
4. Reed I.S. A class of multiple-error-correcting codes and the decoding scheme / I.S. Reed // Trans. of the IRE Professional Group on Information Theory (TIT).– 1954. – Vol. 4:– Iss. 4. – P. 38–49.
5. Solomon G. A Note on a New Class of Codes / G. Solomon // Information and Control. – 1961. – Vol. 4. – Iss. 4:– P. 364–370.
6. Питерсон У. Коды, исправляющие ошибки / У. Питерсон, Э.М. Уэлдон. – М. :Мир, 1976. – 593 с.
7. Берлекэмп Э. Алгебраическая теория кодирования = Algebraic Coding Theory / Э. Берлекэмп. — М.: Мир, 1971. — С. 478.
8. Золотарев В.В. Эффективные многопороговые методы декодирования самоортогональных кодов / В.В. Золотерев, Г.В. Овечкин, П.В. Овечкин // Вестник Рязанского государственного радиотехнического университета. – 2017. – № 60. – С. 113–122.
9. Золотарёв В.В. О новом этапе развития оптимизационной теории кодирования / В.В. Золотарев // Цифровая обработка сигналов. – 2017. – № 1. – С. 33–41.
10. Золотарев В.В. Программная реализация многопороговых декодеров с использованием GPU / В.В. Золотарев, Г.В. Овечкин, П.В. Овечкин // Радиотехника. – 2016. – № 11. – С. 90–96.

11. Золотарев В.В. Новые средства коррекции ошибок для высокоскоростной передачи и хранения данных / В.В. Золотарев, Г.В. Овечкин // Радиотехника. – 2016. – № 8. – С. 104–109.
12. Гладких А.А. Перестановочное декодирование как инструмент повышения энергетической эффективности систем обмена данными / А.А. Гладких // Электросвязь. – 2017. – № 8. – С. 52–56.
13. Гладких А.А. Эффективное перестановочное декодирование двоичных блоковых избыточных кодов / А.А. Гладких, С.М. Наместников, Н.А. Пчелин // Автоматизация процессов управления. – 2017. – № 1 (47). – С. 67–74.
14. Гладких А.А. Обобщенный метод декодирования по списку на базе кластеризации пространства кодовых векторов / А.А. Гладких // Радиотехника. – 2015. – № 6. – С. 37–41.
15. Гладких А.А. Моделирование алгоритмов совместной обработки полярных кодов в системе произведения кодов / А.А. Гладких, Н.Ю. Чилихин // Радиотехника. – 2014. – № 7. – С. 111–115.
16. Егоров С.И. Декодирование произведений кодов Рида-Соломона в каналах с группированием ошибок / С.И. Егоров, А.В. Кривонос, В.С. Титов // Телекоммуникации. – 2018. – № 11. – С. 15-22.
17. Егоров С.И. Повышение эффективности декодирования кодов Рида-Соломона по обобщенному минимальному расстоянию / С.И. Егоров, Д.Б. Борзов, С.В. Дегтярев, В.Э. Дрейзин, И.Б. Михайлов // Известия Юго-Западного государственного университета. – 2018. – № 3 (78). – С. 51-58.
18. Егоров С.И. Процедуры коррекции ошибок для оптической памяти / С.И. Егоров, А.В. Кривонос, А.О. Сазонов, Д.В. Цвелик // Известия высших учебных заведений. Приборостроение. – 2015. – Т. 58. № 2. – С. 109-114.
19. Егоров С.И., Кривонос А.В. Устройство коррекции ошибок для оптической памяти массового применения / С.И. Егоров, А.В. Кривонос // Известия

Юго-Западного государственного университета. – 2017. – № 6 (75). – С. 22-31.

20. Башкиров А.В. Проектирование на основе ПЛИС и реализация многофункционального LDPC-декодера / А.В. Башкиров, М.В. Хорошайлова // Радиотехника. — 2018. — № 7. — С. 46-51.
21. Башкиров А.В., Свиридова И.В., Андреева Д.С. Эффективная архитектура на основе ПЛИС для полностью параллельного стохастического LDPC-декодера / А.В. Башкиров, И.В. Свиридова, Д.С. Андреева // Вестник Воронежского государственного технического университета. — 2018. — Т. 14, № 3. — С. 101-107.
22. Башкиров А.В. Архитектура и реализация на ПЛИС регулярных (2, DC) NB-LDPC-декодеров / А.В. Башкиров, А.В. Муратов, М.В. Хорошайлова // Радиотехника. — 2017. — № 6. С. — 179-183.
23. Peterson W.W., Brown, D.T. Cyclic Codes for Error Detection / W.W. Peterson., D.T. Brown // Proceedings of the IRE. – 1961. – Vol: 49. – Iss: 1. – P. 228 – 235.
24. Chen M-C. Design and implementation of a video-oriented network-interface-card system / M-C. Chen, S-F. Hsiao, C.-H. Yang // *Proceedings of the ASP-DAC Asia and South Pacific Design Automation Conference*, Kitakyushu, Japan: IEEE Press, 2003. – P. 559-560.
25. Implementation of a PCI based gigabit Ethernet network adapter on an FPGA together with a Linux device driver [Электронный ресурс]. – URL: https://www.diva-portal.org/smash/get/diva2:22746/FULLTEXT01.pdf (посл. обращ.: 09.01.2019).
26. Помехоустойчивое кодирование для передачи цифрового телевидения по каналам связи [Электронный ресурс]. // Сайт «Телевидение и радиовещание». – URL: http://www.broadcasting.ru/articles2/codobor/pomehoystoichivoekodirovanie-dlya-peredachi (посл. обращ.: 09.01.2019).

27. Безуглов Д.А. Исследование параметров помехоустойчивого кодирования при анализе космических систем связи / Д.А. Безуглов, А.П. Лыков, Д.Г. Киреев, П.М. Поморцев, С.А. Швидченко [Электронный ресурс] // Современные проблемы науки и образования. – 2012. – № 6. URL: http://www.science-education.ru/ru/article/view?id=7663 (посл. обращ.: 09.01.2019).
28. Тимофеев Г.С. Выбор алгоритма помехоустойчивого кодирования для систем беспроводной цифровой связи / Г.С. Тимофеев // Актуальные проблемы авиации и космонавтики. – 2016. – №12. – С. 655–657.
29. Кодер канала в стандарте GSM [Электронный ресурс]. // Сайт «Искусственный разум» URL: https://intellect.ml/4-4-koder-kanala-v-standarte-gsm-7634 (посл. обращ: 09.01.2019).
30. Сорокин И.А. Принципы помехоустойчивого кодирования в современных системах телекоммуникации / И.А. Сорокин // Вестник НГИЭИ. – 2015. – №8 (51). – С. 70–75.
31. Бабкин В.Ф. Современные методы помехоустойчивого кодирования для систем дистанционного зондирования Земли / В.Ф. Бабкин, В.В. Золотарёв // Современные проблемы дистанционного зондирования Земли из космоса. – 2005. – Т.2. – № 1. – С. 199-202.
32. Краснюк А.А. Особенности применения методов помехоустойчивого кодирования в суб-100-нм микросхемах памяти для космических систем / А.А. Краснюк, К.А. Петров // Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС). – 2012. – №1. – С. 638-641.
33. Хетагуров Я.А. Повышение надежности цифровых устройств методами избыточного кодирования. / Я.А. Хетагуров, Ю.П. Руднев. – М.: Энергия, 1974. – 272 с.
34. Сагалович Ю.Л. Выбор системы кодирования для защиты запоминающих устройств от ошибок / Ю.Л. Сагалович, Н.С. Щербаков // Проблемы передачи информации. – 1984. – Т. 20. – № 1. – С. 19-27.

35. Волошина В.Н. Обеспечение достоверности хранения информации в АСУ с применением помехоустойчивого кодирования./ Диссертация на соискание ученой степени к.т.н. 61:90-5/1048-5, М. :1989, 153 с.
36. Гарбузов Н.И. Модификация корректирующих кодов для запоминающих устройств с паралельно-последовательной передачей информации / Н.И. Гарбузов, А.Л. Абашкин // Тез. докл. 9 Всесоюзной конференции. по теории кодирования и передачи информации, Одесса, 1988. – С. 242–245.
37. Саголович Ю.Л. Кодовая защита оперативной памяти ЭВМ от ошибок / Ю.Л. Саголович // Автоматика и телемеханика. – 1991. – № 5. – С. 4–40.
38. Mittal S. A survey of techniques for improving error-resilience of DRAM / S. Mittal, M.S. Inukonda // Journal of Systems Architecture. – 2018. – Vol. 91. – P. 11-40.
39. Basak A. Reconfigurable ECC for adaptive protection of memory / A. Basak, S. Paul, Ja. Park, Jo Park, S. Bhunia // Proceedings of *56th IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, Columbus (OH, USA): IEEE Press, 2013. – P. 1085-1088.
40. Sridharan V. Memory Errors in Modern Systems: The Good, The Bad, and The Ugly / V. Sridharan, N. DeBardeleben, S. Blanchard, K. Ferreira, J. Stearley, J. Shalf, S. Gurumurthi // ACM SIGPLAN Notices. – 2015. – № 50. – P. 297–310.
41. Шеннон К. Работы по теории информации и кибернетики / К. Шеннон. – М.: Иностр. лит., 1963. – 832 с.
42. Shannon C.E. A Mathematical Theory of Communication / C.E. Shannon // Bell System Technical Journal. – 1948. – Vol. 27. – P. 379-423.
43. Морелос-Сарагоса Р. Искусство помехоустойчивого кодирования: методы, алгоритмы, применение: Пер. с англ. / Р. Морелос-Сарагоса. – М.: Техносфера, 2006. – 320 с.
44. Еленев Д.В. Компьютерные сети: учебное пособие / Д.В. Еленев. –Самара: Изд-во СГАУ, 2010. – 80 с.

45. Контроль правильности передачи [Электронный ресурс]. – URL: http://icmicro.narod.ru/info_ru/crc/crc.htm (посл. обращ.: 09.01.2019).
46. Ross N.W. A Painless Guide to CRC Error Detection Algorithms [Электронный ресурс]. – URL: http://www.ross.net/crc/download/crc_v3.txt (посл. обращ.: 09.01.2019).
47. Когновицкий О.С. Основы циклических кодов. / О.С. Когновицкий. – СПб.: ЛЭИС, 1972. – 64 с.
48. Koopman P. 32-Bit Cyclic Redundancy Codes for Internet Applications / P. Koopman // Proceedings of International Conference on Dependable Systems and Networks (DSN). – Washington (USA): IEEE, 2002. – P. 459–468.
49. Мак-Вильямс Ф. Дж. Теория кодов, исправляющих ошибки. / Ф.Дж. Мак-Вильямс, Н.Дж. Слоэн: Пер. с англ. Грушко И.И., Зиновьева В.А. – М: Связь, 1979. –744 с.
50. Березюк Н.Т. Кодирование информации: Двоичные коды: справочник / Под ред. Н.Т. Березюка. – Харьков : Вища школа, 1978. – С. 241-249.
51. Акритас А. Основы компьютерной алгебры с приложениями: Пер. с англ. / А. Акритас. — М., Мир, 1994. — 544 с.
52. Massey J.L. Shift-register synthesis and BCH decoding / J.L. Massey // IEEE Transactions on Information Theory. – 1969. – Vol. IT-15. – № 1. – P. 122–127.
53. Sugiyama Y. A method for solving key equation for Goppa codes / Y. Sugiyama, Y. Kasahara, S. Hirasawa, T. Namekawa // Info and Control. – 1975. – Vol. 27. – P. 87–99.
54. Morelos-Zaragoza. BCH codes [Электронный ресурс]. – URL: http://www.eccpage.com/bch3.c (посл. обращ.: 09.01.2019).
55. Peterson W.W. Encoding and error-correction procedures for the Bode-ChaudhuriCodes / W.W. Peterson // IRE Trans Info Theory. – 1960. – Vol. IT-6. – P. 459-470.

56. Золотарёв В.В. Помехоустойчивое кодирование. Методы и алгоритмы : справочник / В.В. Золотарёв, Г.В. Овечкин. – М.: Горячая линия-Телеком. – 2004. – 126 с.
57. Темников Ф.Е. Теоретические основы информационной техники. / Ф.Е. Темников, В.А. Афонин, В.И. Дмитриев. – 2-е изд., испр. и доп. – М.: Энергия, 1979. – 512 с.
58. Мыцко Е.А., Мальчуков А.Н., Рыжова С.Е., Зоев И.В. Вычисление контрольной суммы CRC32 матричным алгоритмом // Свидетельство о государственной регистрации программы для ЭВМ №2016617787.
59. Мыцко Е.А. Исследование программных реализаций алгоритмов вычисления CRC совместных с PKZIP, WINRAR, ETHERNET / Е.А. Мыцко, А.Н. Мальчуков // Известия Томского политехнического университета. – 2013 – Т. 322, №. 5. – С. 170–175.
60. Мыцко Е.А. Примеры программных реализаций алгоритмов вычисления контрольной суммы CRC32 / Е.А. Мыцко // Молодежь и современные информационные технологии: сборник трудов XII Всероссийской научно-практической конференции студентов, аспирантов и молодых ученых, Томск, 12-14 ноября 2014 г. / Томский политехнический университет (ТПУ). – 2014 – Т. 2 – С. 78–79.
61. Mytsko E.A. Software Implementation Research of CRC Computation Algorithms Compatible with PKZIP, WINRAR, ETHERNET / E.A. Mytsko, A.N. Malchukov, V.L. Kim , A.N. Osokin, I.V. Zoev, S.E. Ryzhova // Advances in Comptuer Science Research. – 2016. – Vol. 51. – P. 134–138.
62. Мыцко Е. А. Сравнение быстродействия матричного и табличного алгоритмов вычисления CRC / А. Н. Мальчуков, И. В. Зоев, С. Е. Рыжова // Информационные технологии в науке, управлении, социальной сфере и медицине: сборник научных трудов III Международной конференции, Томск, 23-26 Мая 2016 г. / Томский политехнический университет (ТПУ). – 2016. – Т. 1. – С. 182–184.

63. Павлов В.М. Модернизированная система управления комплексом импульсных модуляторов «Виктория» / В.М. Павлов, К.И. Байструков, С.В. Меркулов и др. // Приборы и техника эксперимента. – 2016. – №. 2. – С. 61–66.
64. Мыцко Е.А. Исследование алгоритмов вычисления контрольной суммы CRC8 в микропроцессорных системах при дефиците ресурсов / Е.А. Мыцко, А.Н. Мальчуков, С.Д. Иванов // Приборы и системы. Управление, контроль, диагностика. – 2018. – №. 6. – С. 22–29.
65. Мыцко Е.А. Исследование алгоритмов вычисления контрольной суммы CRC8 для микроконтроллерной системы измерения температуры / Е.А. Мыцко // Наука. Технологии. Инновации: сборник научных трудов: в 10 т., Новосибирск, 4-8 Декабря 2017 г. / Новосибирский государственный технический университет (НГТУ). – 2017. – Т. 1. – С. 90-94.
66. DS18B20 – Датчик температуры с интерфейсом 1-Wire [Электронный ресурс]. – URL: http://mypractic.ru/ ds18b20-datchik-temperatury-s-interfejsom-1-wire opisanie-na-russkom-yazyke.html (посл. обращ.: 09.01.2019).
67. ATtiny44 [Электронный ресурс]. – URL: https://www.microchip.com/wwwproducts/en/ATtiny44 (посл. обращ.: 09.01.2019).
68. Understanding and Using Cyclic Redundancy Checks with Dallas Semiconductor iButtonTM Products [Электронный ресурс]. – URL: http://www.sal.wisc.edu/PFIS/docs/rssnir/archive/public/Product%20Manuals/maxim-ic/AN27.pdf (посл. обращ.: 09.01.2019).
69. Мыцко Е.А. Реализация устройства контроля правильности передачи данных в системе синхронизации и противоаварийной защиты установки токамак КТМ / Е.А. Мыцко, А.Н. Мальчуков // Молодежь и современные информационные технологии: сборник трудов XIII Международной научно-практической конференции студентов, аспирантов и молодых

ученых, Томск, 9-13 Ноября 2015 г. / Томский политехнический университет (ТПУ). – 2016. – Т. 1. – С. 320–321.

70. Мыцко Е.А. Исследование аппаратных реализаций табличного и матричного алгоритмов вычисления CRC32 / Е.А. Мыцко, А.Н. Мальчуков // Известия Томского политехнического университета. – 2013. – Т. 322, №. 5. – С. 182–186.
71. Мыцко Е.А. Примеры аппаратных реализаций алгоритмов вычисления контрольной суммы CRC32 / Е.А. Мыцко // Молодежь и современные информационные технологии: сборник трудов XII Всероссийской научно-практической конференции студентов, аспирантов и молодых ученых: в 2 т., Томск, 12-14 Ноября 2014 г. / Томский политехнический университет (ТПУ). – 2014. – Т. 2. – С. 76–77.
72. Мыцко Е.А. Исследование программных и аппаратных реализаций алгоритмов вычисления контрольной суммы CRC / Е.А. Мыцко // Наука будущего - наука молодых: сборник тезисов участников III Всероссийского научного форума, Нижний Новгород, 12-14 Сентября 2017. / Москва: Инконсалт К. – 2017. – С. 125–126.
73. Mytsko E.A. A study of hardware implementations of the CRC computation algorithms / E.A. Mytsko, A.N. Malchukov, S.E. Ryzhova, V.L. Kim // MATEC Web of Conferences . – 2016. – Vol. 48, Article number 04001. – P. 1–7.
74. Ахметов Н.Р. Алгоритмы помехоустойчивого кодирования и их аппаратная реализация на основе ПЛИС / Н.Р. Ахметов, А.А. Макаров // Молодой ученый. – 2015. – №13. – С. 92–96.
75. Pramod S. FPGA Implementation of Single Bit Error Correction using CRC / Pramod S., A. Rajagopal, S. Akshay // International Journal of Computer Applications. – 2012. – Vol. 52, №.10. – P. 15–19.
76. Selokar S.D. Design and Implementation of CRC Code Generator based on Parallel Execution Method for High Speed Wireless LAN / S.D. Selokar, P.H.

Rangaree // International Journal of Engineering Research and Applications (IJERA). – 2011. – Vol. 1, Iss. 3. – P. 704–709.

77. Stavinov E.A. Practical Parallel CRC Generation Method / E.A. Stavinov // Circuit Cellar. – 2010. – Iss. 234. – P. 38–45.
78. Campobello G. Parallel CRC realization / G. Campobell, G. Patane, M. Russo // Transactions on Computers. – 2003. – Vol. 52, №. 10. – P. 1312–1319.
79. Shieh M.D. A Systematic Approach for Parallel CRC Computations / M.D. Shieh // Journal of Information Science and Engineering. – 2001. – Vol. 17. – P. 445–461.
80. Youngju D. High-Speed Parallel Architecture for Software-Based CRC / D. Youngju, Y. Sung-Rok, K. Taekyu, E.P. Kwang, P. Sin-Chong // Proceedings of Consumer Communications and Networking Conference (CCNC), Las Vegas, NV, USA: IEEE Press, 2008. – P. 74–78.
81. Yan S. High Performance Table-Based Algorithm for Pipelined CRC Calculation / S. Yan, S.K. Min // Journal of Communications. – 2013. – Vol. 8, №. 2. – P. 1–8.
82. Yuanhong H. High performance table-based architecture for parallel CRC calculation / H. Yuanhong, L. Xiaoyang, W. Wang, L. Dake // Proceedings of The 21st IEEE International Workshop on Local and Metropolitan Area Networks (LANMAN), Beijing, China: IEEE Press, 2015. – P. 1–6.
83. Поляков А.К. Языки VHDL и VERILOG в проектировании цифровой аппаратуры / А.К. Поляков. – М.: Изд-во «СОЛОН-Пресс», 2003. – 320 с.
84. Мальчуков А.Н. Алгоритм поиска образующих полиномов для системы проектирования кодеков помехоустойчивых кодов / А.Н. Мальчуков // Современные техника и технологии: сборник трудов XIV Международной научно-практической конференции студентов, аспирантов и молодых ученых. – Томск, 24-28 марта 2008 г. / Томский политехнический университет (ТПУ). – 2008. – С. 340–341.

85. Мыцко Е.А. Программная реализация алгоритма поиска образующих полиномов с применением технологий OpenMP и MPI [Электронный ресурс] / Е.А. Мыцко, А.Н. Мальчуков // Современные проблемы науки и образования. – 2014 – №. 6. – С. 1–8. URL: http://www.science-education.ru/120-r15685 (посл. обращ.: 09.01.2019).
86. Мыцко Е.А. Применение технологий параллельных и распределённых вычислений для поиска образующих полиномов / Е.А. Мыцко, А.Н. Мальчуков // Современные техника и технологии: сборник трудов XX международной научно-практической конференции студентов, аспирантов и молодых ученых: в 3 т., Томск, 14-18 Апреля 2014 г. / Томский политехнический университет (ТПУ), 2014. – Т. 2. – С. 215–216.
87. Мыцко Е.А. Применение технологий параллельных вычислений в задаче поиска образующих полиномов для построения эффективных помехоустойчивых кодов / Е.А. Мыцко // 55-я Международная научная студенческая конференция (МНСК-2017): Информационные технологии: материалы конференции, Новосибирск, 17-20 Апреля 2017 г. / Новосибирский государственный университет (НГУ), 2017. – с. 135.
88. Мыцко Е.А. Применение технологий параллельных вычислений для поиска образующих полиномов, используемых при построении эффективных помехоустойчивых кодов / Е.А. Мыцко // Научная сессия ТУСУР-2017: материалы Международной научно-технической конференции студентов, аспирантов и молодых ученых, посвященной 55-летию ТУСУРа в 8 частях, Томск, 10-12 Мая 2017 г. / В-Спектр, 2017. – Т. 5. – С. 141–143.
89. Mytsko E.A. Adaptation of technology MPI and OpenMP to search for the generators polynomials / E.A. Mytsko, A.N. Malchukov // Proceedings of 9th International Forum on Strategic Technology (IFOST-2014), Chittagong, Bangladesh: IEEE Press, 2014. – P. 5–8.

90. Mytsko E.A. Application of parallel computing technology openmp to search for the generator polynomials / E.A. Mytsko, A.N. Malchukov // Mechanical Engineering, Automation and Control Systems: Proceedings of International Conference, Tomsk, October 16-18, 2014. – Tomsk: TPU Publishing House, 2014. – P. 1–5.
91. Мыцко Е.А. Поиск образующих полиномов для построения циклических помехоустойчивых кодов с применением технологий параллельных вычислений // Свидетельство о государственной регистрации программы для ЭВМ № 2018665738.
92. Антонов А.С. Параллельное программирование с использованием технологии OpenMP: учебное пособие / А.С. Антонов. – М.: Изд-во МГУ, 2009. – 77 с.
93. Мальчуков А.Н. Реализация кодека помехоустойчивого двоичного циклического кода в потоке параллельных данных на ПЛИС / А.Н. Мальчуков, Ю.Б. Буркатовская, А.Н. Осокин // Системы и средства автоматизации: материалы 5-ой Всероссийской конференции, Томск. – : ТУСУР, 2004. – С. 102–105.
94. Мальчуков А.Н. Система проектирования кодеков помехоустойчивых кодов / А.Н. Мальчуков, А.Н. Осокин // Молодежь и современные информационные технологии: сборник трудов VI Всероссийской научно-практической конференции студентов, аспирантов и молодых ученых, Томск, 26-28 февраля 2008 г. / Томский политехнический университет (ТПУ), 2008. – С. 45–46.
95. Мыцко Е.А. Реализация алгоритма поиска образующих полиномов для циклических кодов, исправляющих пакетные ошибки / Е.А. Мыцко // Молодежь и современные информационные технологии : сборник трудов XIII Международной научно-практической конференции студентов, аспирантов и молодых ученых, Томск, 7-11 ноября 2016 г. / Томский политехнический университет (ТПУ), 2016. – Т. 1. – С. 310–311.

96. Мыцко Е.А. Поиск образующих полиномов для построения декодеров, исправляющих пакетные ошибки с применением технологий параллельных вычислений / Е.А. Мыцко // Высокопроизводительные вычислительные системы и технологии. – 2018. – № 1 (8). – С. 97-101.
97. Мыцко Е.А. Разработка декодера БЧХ-кода (17,9,5) на ПЛИС с применением комбинированного метода декодирования / Е.А. Мыцко, С.Е. Рыжова // 56-я Международная научная студенческая конференция (МНСК-2018): Информационные технологии: материалы конференции, Новосибирск, 22-27 Апреля 2018 г. / Новосибирский государственный университет (НГУ), 2018. – с. 111.
98. Рыжова С.Е. Реализация комбинированного метода декодирования на примере декодера полиномиального кода (17, 9) / С.Е. Рыжова, Е.А. Мыцко // Сборник избранных статей научной сессии ТУСУР: по материалам Международной научно-технической конференции студентов, аспирантов и молодых ученых «Научная сессия ТУСУР–2018», Томск, 16-18 Мая 2018 г. / Томский университет систем управления и радиоэлектроники (ТУСУР), 2018. – Ч. 3.– С. 180-183.
99. Mytsko E.A. FPGA design of the fast decoder for burst errors correction / E.A. Mytsko, A.N. Malchukov, I.V. Zoev, S.E. Ryzhova, V.L. Kim // Journal of Physics: Conference Series. – 2016. –Vol. 803, Article Number: 012105. – P. 1–6.
100. Рыжова С.Е. Разработка и аппаратная реализация декодера помехоустойчивого полиномиального кода (15, 8, 3), исправляющего пакетные ошибки, на ПЛИС / С.Е. Рыжова, А.Н. Мальчуков, Е.А. Мыцко // Информационные технологии в науке, управлении, социальной сфере и медицине: сборник научных трудов III Международной конференции, Томск, 23-26 мая, 2016 г: / Томский политехнический университет (ТПУ). – 2016. – Т. 1. – С. 736–738.

101. Мыцко Е.А. Исследование программных реализаций декодеров циклических помехоустойчивых кодов, исправляющих пакетные ошибки при дефиците ресурсов / Е.А. Мыцко, А.Н. Мальчуков, С.Д. Иванов // Приборы и системы. Управление, контроль, диагностика. – 2018. – №. 9. – С. 27-36.
102. ATmega8515 [Электронный ресурс]. – URL: http://ww1.microchip.com/downloads/en/devicedoc/doc2512.pdf (посл. обращ.: 09.01.2019).
103. Рыжова С.Е., Мальчуков А.Н., Мыцко Е.А., Зоев И.В. Быстродействующее декодирование кода Боуза-Чоудхури-Хоквингема (15,7,5) // Свидетельство о государственной регистрации программы для ЭВМ №2016662525.
104. Рыжова С.Е. , Мыцко Е.А. , Мальчуков А.Н. Быстродействующий декодер кода Боуза-Чоудхури-Хоквингема (15,5,7), исправляющий до 3-х независимых ошибок // Свидетельство о государственной регистрации программы для ЭВМ №2018663370.
105. Mytsko E.A. Structure Development of the BCH Code High-speed Decoder Based on the Cyclic Decoding Method / E.A. Mytsko, A.N. Malchukov, S.E. Ryzhova, V.L. Kim // Proceeding of International Conference on Applied Mechanics and Mechatronics Engineering (AMME 2015), Bangkok, Thailand: Lancaster: DEStech Publications Inc., 2015. – P. 175-178.
106. Mytsko E.A. Fast Decoder of BCH Code with Cyclic Decoding Method / E.A. Mytsko, A.N. Malchukov, I.V. Novozhilov, V.L. Kim // Proceedings of International Siberian Conference on Control and Communications (SIBCON - 2016): Москва: ВШЭ, 2016. – P. 1–4.
107. Мыцко Е.А. Разработка структуры быстродействующего декодера БЧХ-кода (15,7,5) на основе метода циклического декодирования / Е.А. Мыцко, А.Н. Мальчуков, С.Е. Рыжова, И.В. Зоев, В.Л. Ким // Прикладная информатика. – 2017. – Т. 12, №. 2 (68). – С. 72–78.

108. Мальчуков А.Н. Матричный метод деления полиномов в арифметике по модулю два / А.Н. Мальчуков // Технологии Microsoft в информатике и программировании: Конференция - конкурс работ студентов, аспирантов и молодых ученых, Новосибирск, 21-23 февраля 2004 г. / Новосибриский государственный университет (НГУ), 2004. – С. 115–116.
109. Prakash G. FPGA Implementation of Bose Chaudhuri Hocquenghem Code (Bch) Encoder and Decoder for Multiple Error Correction Control / G. Prakash, I. Muthamizhse // International Journal of Innovative Research in Science, Engineering and Technology. – 2016. – Vol. 5, Iss. 3. – P. 4467–4473.
110. Panda A. FPGA Implementation of Encoder for (15, k) Binary BCH Code Using VHDL and Performance Comparison for Multiple Error Correction Control / A. Panda, S. Sarik, A. Awasthi // Proceedings of International Conference on Communication Systems and Network Technologies (CSNT), Rajkot, India: IEEE Press, 2012. – P. 780 – 784.
111. Rohith S. FPGA Implementation of (15,7) BCH encoder and decoder for text message / S. Rohith, S. Pavithra // International Journal of Research in Engineering and Technology. – 2013. – Vol. 02. – Iss: 09. – P. 209–214.
112. Hiremath M. A Novel Method Implementation of a FPGA using (n, k) Binary BCH Code / M. Hiremath, M.A. Devi // International Journal of Research in Engineering Technology and Management. – 2014. – Spec. Iss. – P. 1–8.
113. Sunita M.S. Pipeline architecture for fast decoding of bch codes For nor flash memory / M.S. Sunita, V. Chiranth, H.C. Akash, V.S. Kanchana Bhaaskaran // ARPN Journal of Engineering and Applied Sciences. – 2015. – Vol. 10, №. 8. – P. 3397–3404.
114. Yathiraj H. Implementation of BCH Code (n, k) Encoder and Decoder for Multiple Error Correction Control / H. Yathiraj, R. Hiremath. // International Journal of Computer Science and Mobile Applications. – 2014. – Vol. 2, Iss. 5. – P. 45–54.

115. Mohammed S. Design and Implementation of 2 bits BCH Error Correcting Codes using FPGA / S. Mohammed, H. Abdulsada // Journal of telecommunications. – 2013. – Vol. 19. – Iss. 2. – P. 11–17.
116. Lakhendra K. FPGA Implementation of (15, 7) BCH Encoder and Decoder for Audio Message / K. Lakhendra // International Journal of Engineering Sciences & Research Technology (IJESRT). – 2014. – №. 3 (8). – P. 407–413.
117. Рыжова С.Е. Сравнение аппаратных реализаций комбинированного метода декодирования на примере кода (17, 9) / С.Е. Рыжова, Е.А. Мыцко // Высокопроизводительные вычислительные системы и технологии. – 2018 .– № 1 (8). – С. 24–28.
118. Мыцко Е.А. Проектирование и реализация устройств на ПЛИС для исправления независимых ошибок с применением циклических помехоустойчивых кодов (17, 9, 5) и (19, 9, 5) / Е.А. Мыцко // Высокопроизводительные вычислительные системы и технологии. – 2018. – № 2 (9). – С. 29-34.
119. Павлов В.М. Система синхронизации и противоаварийной защиты: учебное пособие / В.М. Павлов, К.И. Байструков, С.В. Меркулов. – Томск: Изд-во Томского политехнического университета, 2008. – 143 с.
120. Дериглазов А.А. Система синхронизации и противоаварийной защиты токамак КТМ / А.А. Дериглазов // VIII Школа-конференция молодых атомщиков Сибири : сборник тезисов докладов, 17-19 мая 2017 г. / Томский университет систем управления и радиоэлектроники (ТУСУР), 2017. – 1 с.
121. Учебный лабораторный стенд SDK-6.1 [Электронный ресурс]. – URL: http://embedded.ifmo.ru/index.php/support/sdk-61 (посл. обращ.: 01.05.2019).

Приложение А. Блок-схемы алгоритмов вычисления контрольной суммы crc8 для микроконтроллера ATtiny 44.

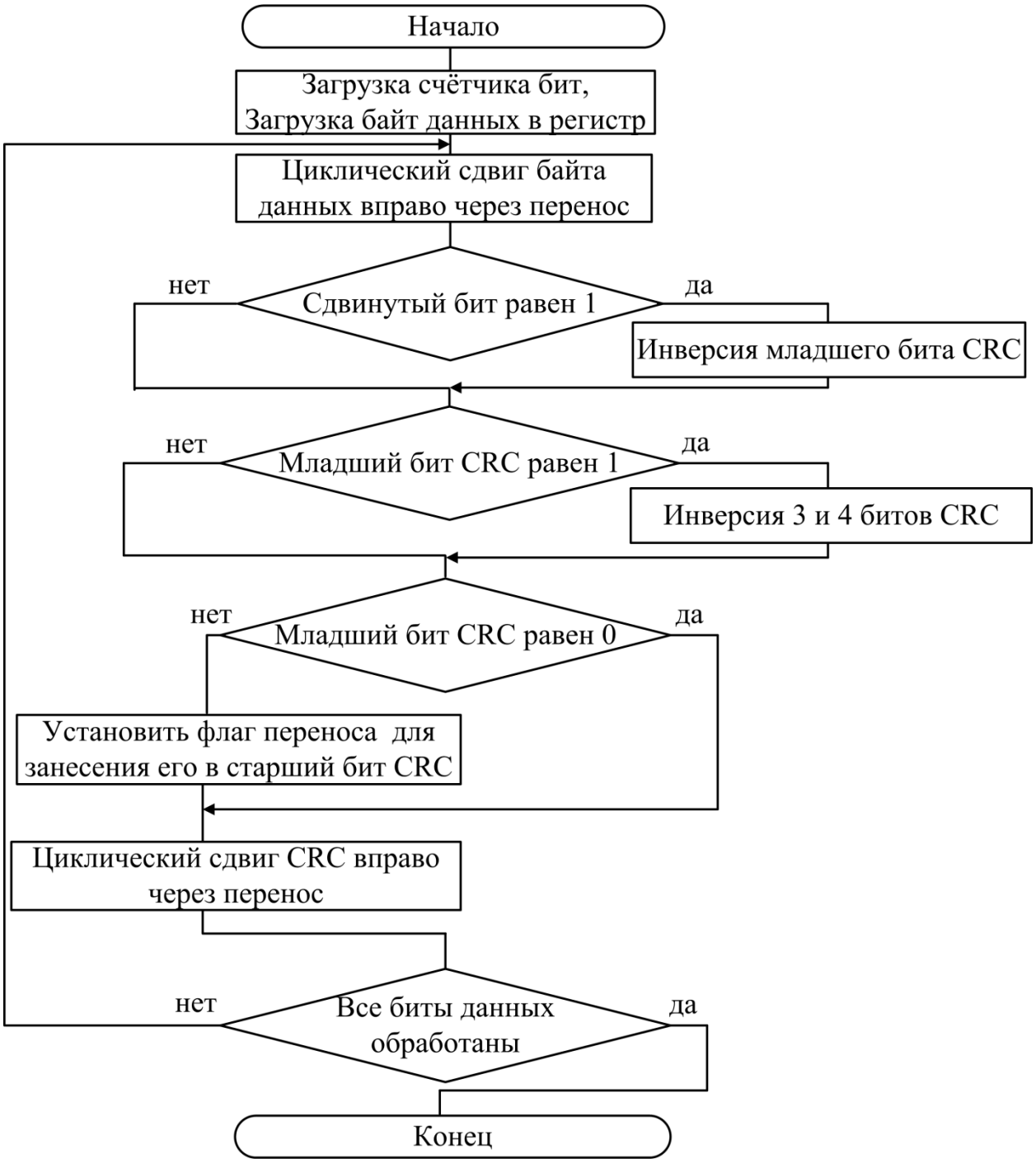


Рисунок А.1 – Схема классического алгоритма CRC8 для ATtiny44

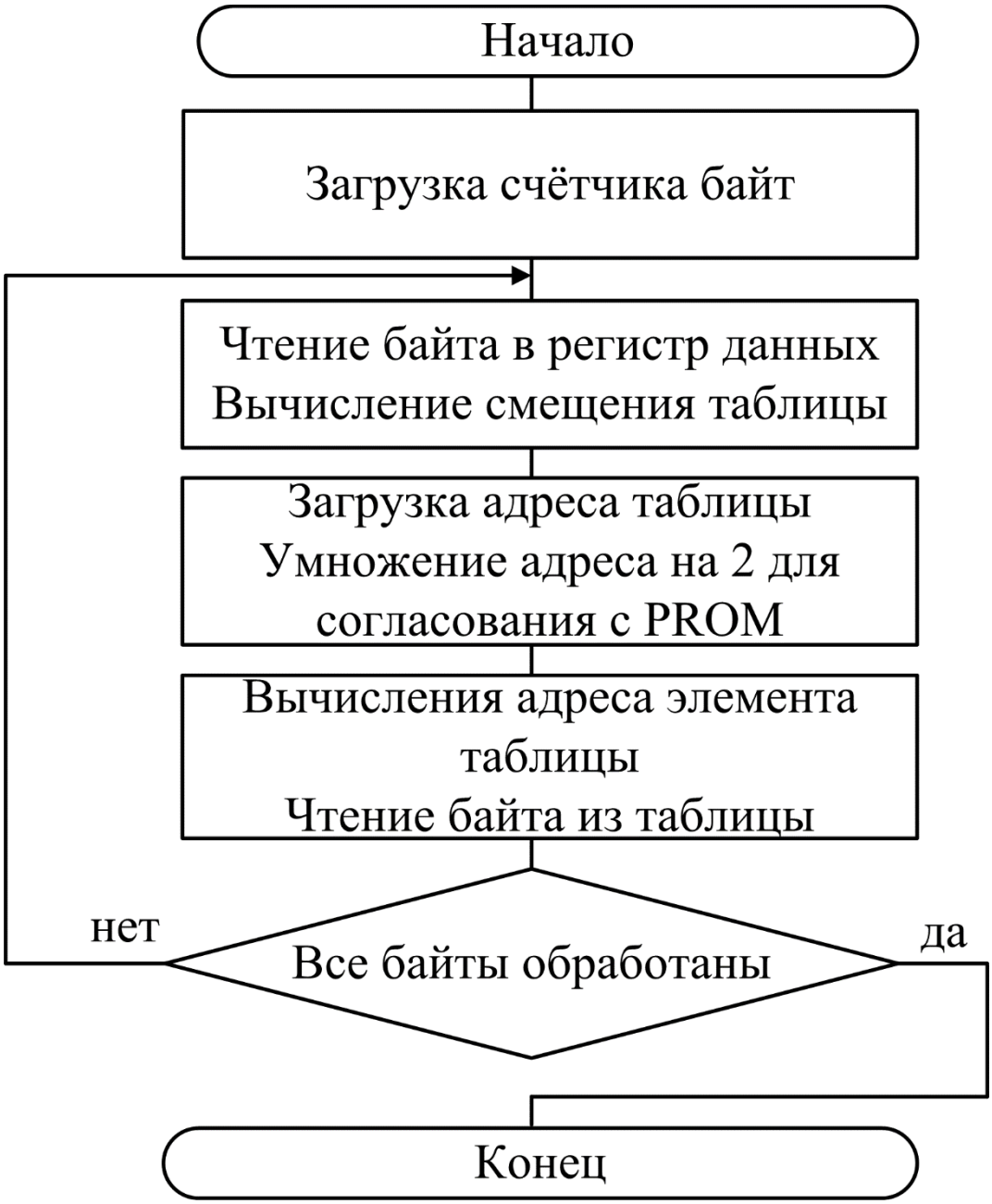


Рисунок А.2 – Схема табличного алгоритма CRC8 для ATtiny44

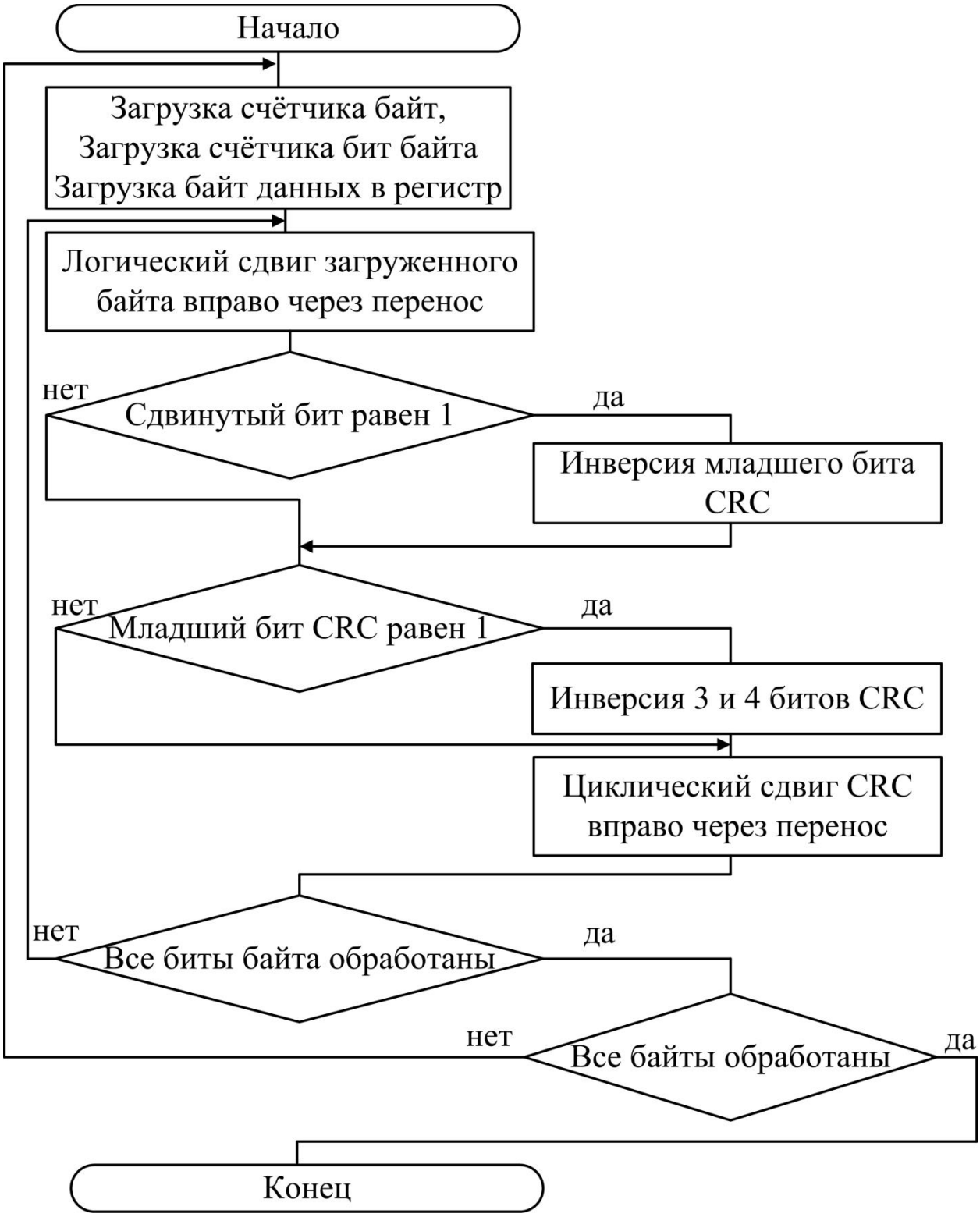


Рисунок А.3 – Схема модифицированного классического алгоритма CRC8 для ATtiny44

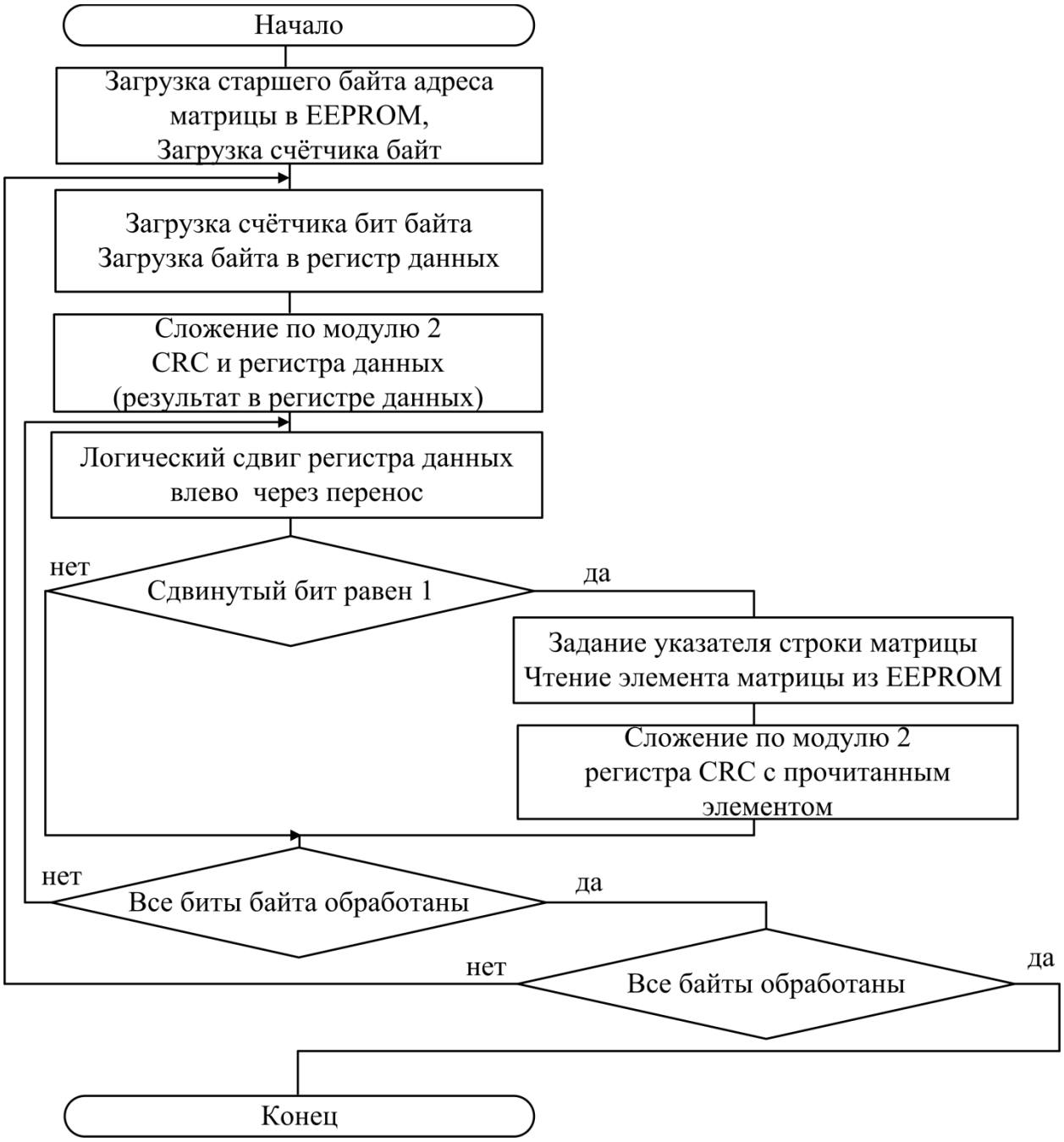


Рисунок А.4 – Схема матричного алгоритма CRC8 с EEPROM для ATtiny44

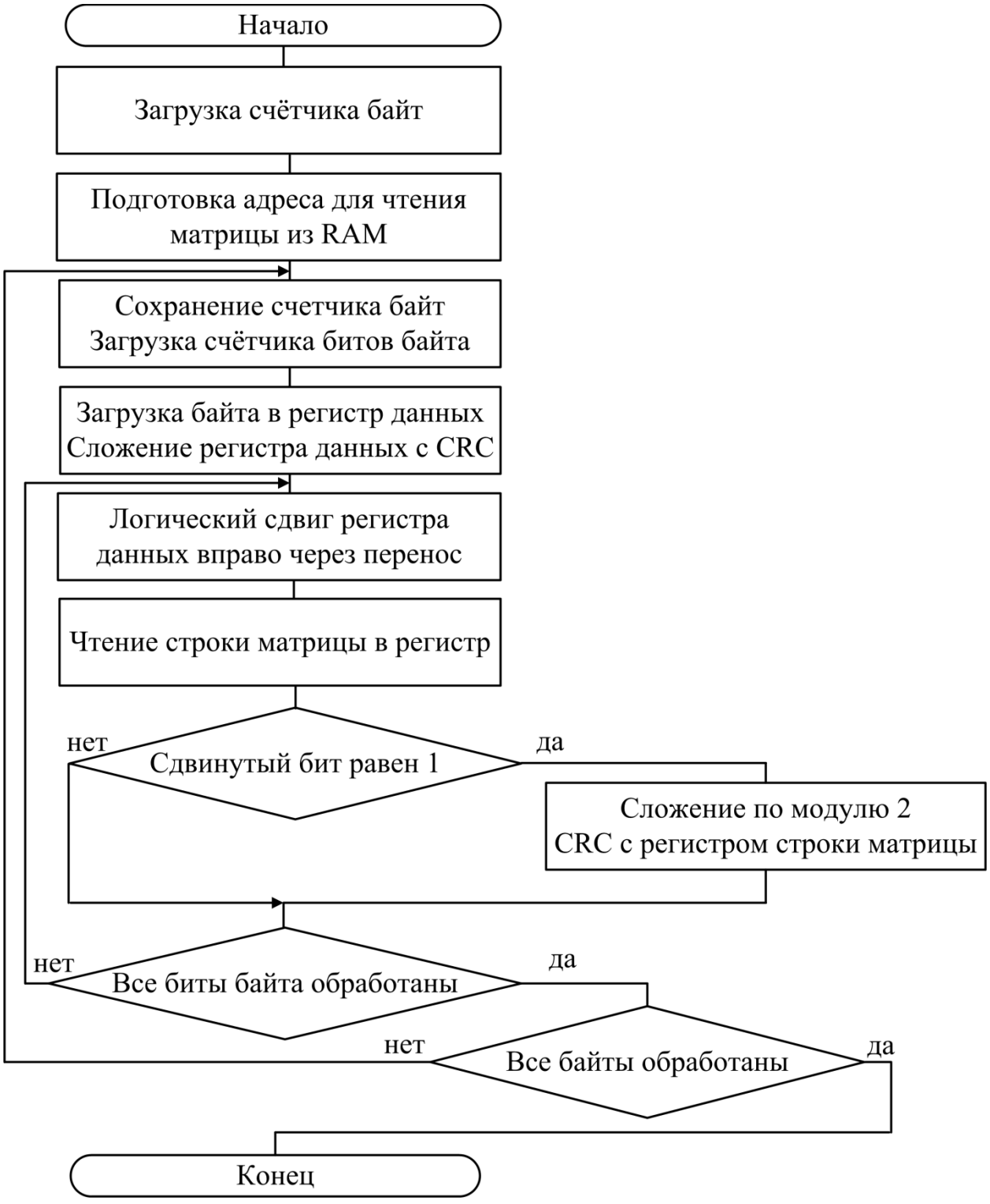


Рисунок А.5 – Схема матричного алгоритма CRC8 с RAM для ATtiny44

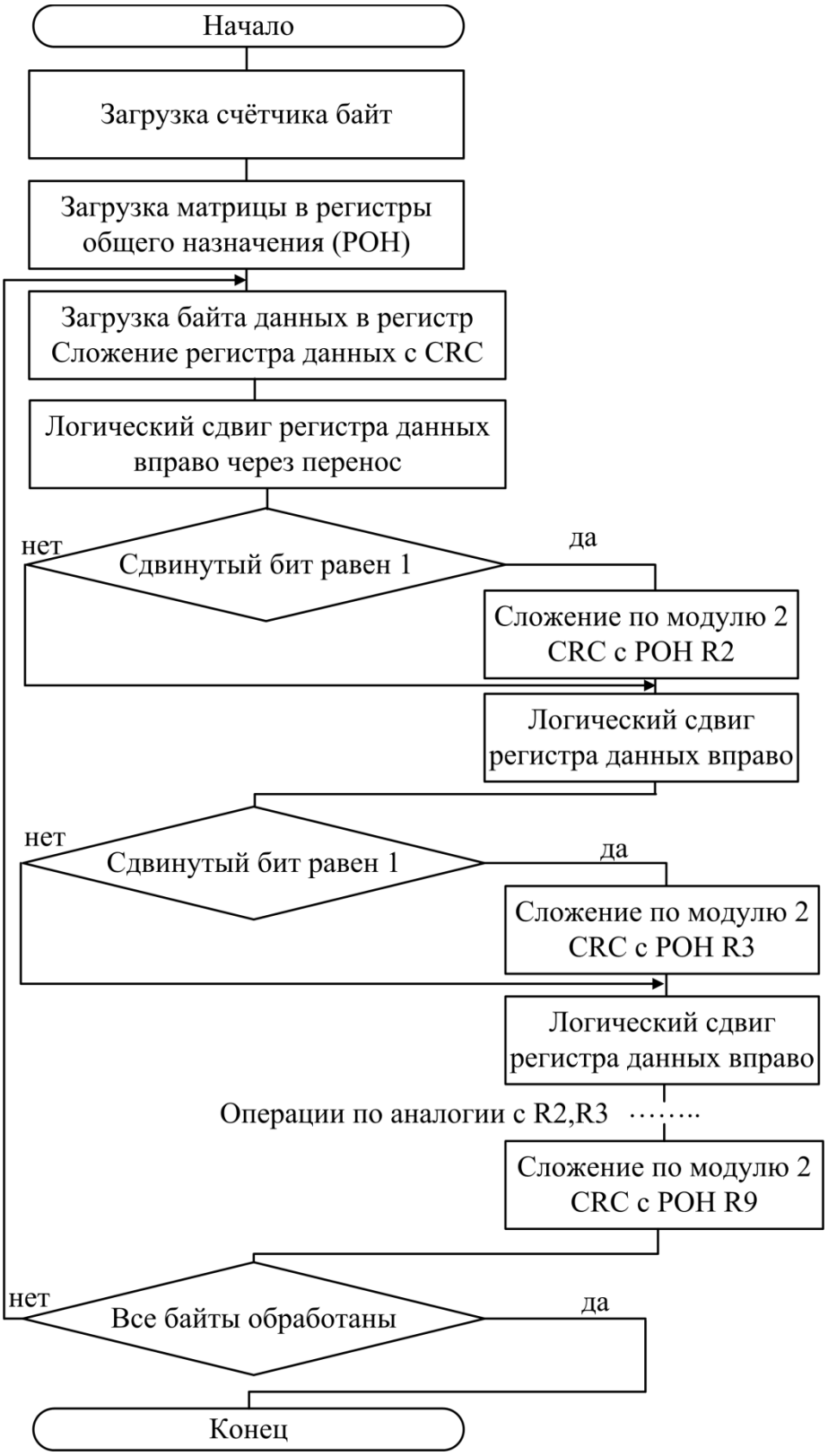


Рисунок А.6 – Схема матричного алгоритма CRC8 с регистрами общего назначения для ATtiny44

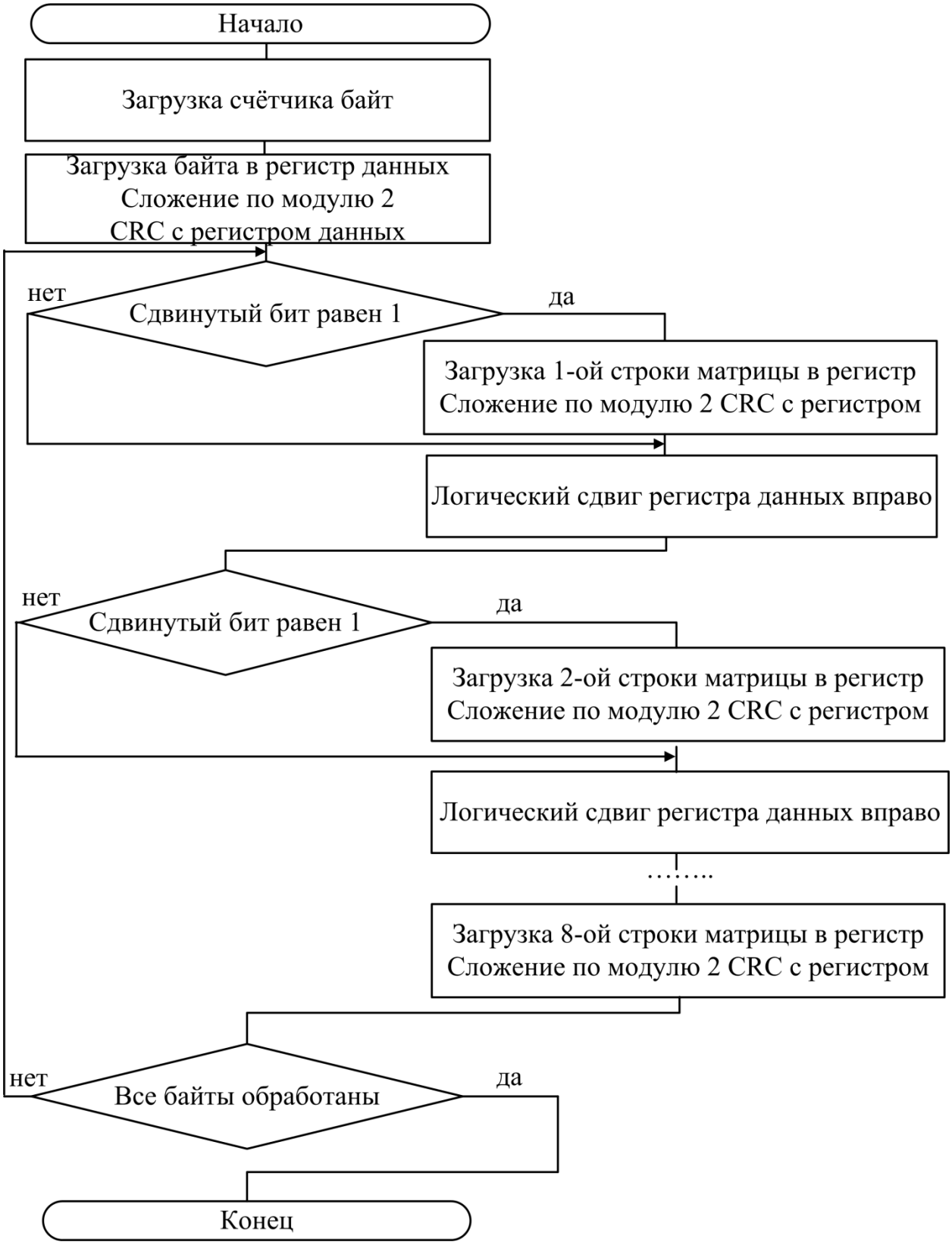


Рисунок А.7 – Схема матричного алгоритма CRC8 с матрицей в коде программы для ATtiny44

Приложение Б. Шаблоны ошибок для табличного алгоритма декодирования

Таблица Б.1 – Шаблоны двукратной независимой ошибки для (15, 7, 5)

Адрес	Значение (шаблон ошибки)
00000000	**0000000**00000000
00000001	**0000000**00000001
00000010	**0000000**00000010
00000100	**0000000**00000100
00001000	**0000000**00001000
00010000	**0000000**00010000
00100000	**0000000**00100000
01000000	**0000000**01000000
10000000	**0000000**10000000
11010001	**0000001**00000000
01110011	**0000010**00000000
11100110	**0000100**00000000
00011101	**0001000**00000000
00111010	**0010000**00000000
01110100	**0100000**00000000
11101000	**1000000**00000000
00000011	**0000000**00000011
00000101	**0000000**00000101
00001001	**0000000**00001001
00100001	**0000000**00100001
01000001	**0000000**01000001
10000001	**0000000**10000001
11010000	**0000001**00000001
01110010	**0000010**00000001
11100111	**0000100**00000001
00011100	**0001000**00000001
00111011	**0010000**00000001
01110101	**0100000**00000001
11101001	**1000000**00000001
00000110	**0000000**00000110
00001010	**0000000**00001010
00010010	**0000000**00010010
00100010	**0000000**00100010
01000010	**0000000**01000010
10000010	**0000000**10000010
11010011	**0000001**00000010
01110001	**0000010**00000010
11100100	**0000100**00000010
00011111	**0001000**00000010
00111000	**0010000**00000010
01110110	**0100000**00000010

Продолжение таблицы Б.1

11101010	**1000000**00000010
00001100	**0000000**00001100
00010100	**0000000**00010100
00100100	**0000000**00100100
01000100	**0000000**01000100
10000100	**0000000**10000100
11010101	**0000001**00000100
01110111	**0000010**00000100
11100010	**0000100**00000100
00011001	**0001000**00000100
00111110	**0010000**00000100
01110000	**0100000**00000100
11101100	**1000000**00000100
00011000	**0000000**00011000
00101000	**0000000**00101000
01001000	**0000000**01001000
10001000	**0000000**10001000
11011001	**0000001**00001000
01111011	**0000010**00001000
11101110	**0000100**00001000
00010101	**0001000**00001000
00110010	**0010000**00001000
01111100	**0100000**00001000
11100000	**1000000**00001000
00110000	**0000000**00110000
01010000	**0000000**01010000
10010000	**0000000**10010000
11000001	**0000001**00010000
01100011	**0000010**00010000
11110110	**0000100**00010000
00001101	**0001000**00010000
00101010	**0010000**00010000
01100100	**0100000**00010000
11111000	**1000000**00010000
01100000	**0000000**01100000
10100000	**0000000**10100000
11110001	**0000001**00100000
01010011	**0000010**00100000
11000110	**0000100**00100000
00111101	**0001000**00100000
00011010	**0010000**00100000
01010100	**0100000**00100000
11001000	**1000000**00100000
11000000	**0000000**01100000
10100000	**0000000**10100000
11110001	**0000001**00100000
01010011	**0000010**00100000

Окончание таблицы Б.1

11000110	**0000100**00100000
00111101	**0001000**00100000
00011010	**0010000**00100000
01010100	**0100000**00100000
11001000	**1000000**00100000
11000000	**0000000**11000000
10010001	**0000001**01000000
00110011	**0000010**01000000
10100110	**0000100**01000000
01011101	**0001000**01000000
01111010	**0010000**01000000
00110100	**0100000**01000000
10101000	**1000000**01000000
01010001	**0000001**10000000
11110011	**0000010**10000000
01100110	**0000100**10000000
10011101	**0001000**10000000
10111010	**0010000**10000000
11110100	**0100000**10000000
01101000	**1000000**10000000
10100010	**0000011**00000000
00110111	**0000101**00000000
11001100	**0001001**00000000
11101011	**0010001**00000000
10100101	**0100001**00000000
00111001	**1000001**00000000
10010101	**0000110**00000000
01101110	**0001010**00000000
01001001	**0010010**00000000
00000111	**0100010**00000000
10011011	**1000010**00000000
11111011	**0001100**00000000
11011100	**0010100**00000000
10010010	**0100100**00000000
00001110	**1000100**00000000
00100111	**0011000**00000000
01101001	**0101000**00000000
11110101	**1001000**00000000
01001110	**0110000**00000000
11010010	**1010000**00000000
10011100	**1100000**00000000

Таблица Б.2 – Шаблоны двукратной независимой ошибки для циклического кода (17, 9, 5)

Адрес	**Значение (шаблон ошибки)**
00000000	00000000000000000

Продолжение таблицы Б.2

00000001	00000000000000001
00000010	00000000000000010
00000011	00000000000000011
00000100	00000000000000100
00000101	00000000000000101
00000110	00000000000000110
00000111	00100000000100000
00001000	00000000000001000
00001001	00000000000001001
00001010	00000000000001010
00001100	00000000000001100
00001110	01000000001000000
00001111	00010000010000000
00010000	00000000000010000
00010001	00000000000010001
00010010	00000000000010010
00010011	10010000000000000
00010100	00000000000010100
00010101	00000110000000000
00011000	00000000000011000
00011001	00000000100100000
00011100	10000000010000000
00011110	00100000100000000
00100000	00000000000100000
00100001	00000000000100001
00100010	00000000000100010
00100011	00100000000000100
00100100	00000000000100100
00100101	00100000000000010
00100110	00100000000000001
00100111	00100000000000000
00101000	00000000000101000
00101001	00000000100010000
00101010	00001100000000000
00101111	00100000000001000
00110000	00000000000110000
00110001	00000000100001000
00110010	00000001001000000
00110111	00100000000010000
00111000	00000000100000001
00111001	00000000100000000
00111011	00000000100000010

Продолжение таблицы Б.2

00111100	01000001000000000
00111101	00000000100000100
00111111	00001010000000000
01000000	00000000001000000
01000001	00000000001000001
01000010	00000000001000010
01000100	00000000001000100
01000110	01000000000001000
01000111	10001000000000000
01001000	00000000001001000
01001010	01000000000000100
01001011	00000001100000000
01001100	01000000000000010
01001110	01000000000000000
01001111	01000000000000001
01010000	00000000001010000
01010010	00000001000100000
01010100	00011000000000000
01010101	00100001000000000
01011011	00001000010000000
01011110	01000000000010000
01100000	00000000001100000
01100010	00000001000010000
01100100	00000010010000000
01100111	00100000001000000
01101001	01100000000000000
01101011	00010010000000000
01101101	10000100000000000
01101110	01000000000100000
01110000	00000001000000010
01110001	00000100010000000
01110010	00000001000000000
01110011	00000001000000001
01110110	00000001000000100
01110111	01000000100000000
01111000	10000010000000000
01111001	00000000101000000
01111010	00000001000001000
01111110	00010100000000000
10000000	00000000010000000
10000001	00000000010000001
10000010	00000000010000010
10000011	00000101000000000
10000100	00000000010000100
10000111	00010000000001000
10001000	00000000010001000
10001011	00010000000000100

Продолжение таблицы Б.2

10001100	10000000000010000
10001101	00010000000000010
10001110	00010000000000001
10001111	00010000000000000
10010000	00000000010010000
10010100	10000000000001000
10010101	01001000000000000
10010110	00000011000000000
10011000	10000000000000100
10011011	00001000001000000
10011100	10000000000000000
10011101	10000000000000001
10011110	10000000000000010
10011111	00010000000010000
10100000	00000000010100000
10100100	00000010001000000
10100101	10000000100000000
10100111	00100000010000000
10101000	00110000000000000
10101001	00001001000000000
10101010	01000010000000000
10101111	00010000000100000
10110001	00000100001000000
10110110	00010000100000000
10111001	00000000110000000
10111011	10100000000000000
10111100	10000000000100000
10111111	01000100000000000
11000000	00000000011000000
11000001	01010000000000000
11000011	00100010000000000
11000100	00000010000100000
11001000	00000100100000000
11001011	00001000000010000
11001110	01000000010000000
11001111	00010000001000000
11010001	00000100000100000
11010010	11000000000000000
11010011	00001000000001000
11010100	00001000000001000
11010110	00100100000000000
11011001	00001000000000010
11011010	00001000000000001
11011011	00001000000000000
11011100	10000000001000000
11011101	00000010100000000
11011111	00001000000000100

Окончание таблицы Б.2

11100000	00000010000000100
11100001	00000100000010000
11100010	00001000100000000
11100100	00000010000000000
11100101	00000010000000001
11100110	00000010000000010
11101100	00000010000001000
11101110	10000001000000000
11110000	00000100000000001
11110001	00000100000000000
11110010	00000001010000000
11110011	00000100000000010
11110100	00000010000010000
11110101	00000100000000100
11111001	00000100000001000
11111011	00001000000100000
11111100	00101000000000000
11111101	00010001000000000

Приложение В . Количество тактов, необходимых для исправления шаблонов ошибок для помехоустойчивых кодов (17,9,5) и БЧХ (19,9,5)

Таблица В.1 – Количество тактов для помехоустойчивого кода (17, 9, 5)

Шаблон ошибки	Кол-во тактов	Шаблон ошибки	Кол-во тактов	Шаблон ошибки	Кол-во тактов
10000000000000000	11	100010000000000	9	10000100000	5
1000000000000000	10	100001000000000	9	10000010000	5
100000000000000	9	100000100000000	9	10000001000	5
10000000000000	8	100000010000000	9	10000000100	3
1000000000000	7	100000001000000	3	10000000010	3
100000000000	6	100000000100000	3	10000000001	12
10000000000	5	100000000010000	16	1100000000	4
1000000000	4	100000000001000	15	1010000000	4
100000000	3	100000000000100	14	1001000000	4
10000000	2	100000000000010	13	1000100000	4
1000000	2	100000000000001	12	1000010000	4
100000	2	11000000000000	8	1000001000	4
10000	2	10100000000000	8	1000000100	4
1000	2	10010000000000	8	1000000010	3
100	2	10001000000000	8	1000000001	3
10	2	10000100000000	8	110000000	3
1	2	10000010000000	8	101000000	3
11000000000000000	11	10000001000000	8	100100000	3
10100000000000000	11	10000000100000	3	100010000	3
10010000000000000	11	10000000010000	3	100001000	3
10001000000000000	11	10000000001000	15	100000100	3
10000100000000000	11	10000000000100	14	100000010	3
10000010000000000	11	10000000000010	13	100000001	3
10000001000000000	11	10000000000001	12	11000000	2
10000000100000000	3	1100000000000	7	10100000	2
10000000010000000	3	1010000000000	7	10010000	2
10000000001000000	18	1001000000000	7	10001000	2
10000000000100000	17	1000100000000	7	10000100	2
10000000000010000	16	1000010000000	7	10000010	2
10000000000001000	15	1000001000000	7	10000001	2
10000000000000100	14	1000000100000	7	1100000	2
10000000000000010	13	1000000010000	3	1010000	2
10000000000000001	12	1000000001000	3	1001000	2
1100000000000000	10	1000000000100	14	1000100	2
1010000000000000	10	1000000000010	13	1000010	2
1001000000000000	10	1000000000001	12	1000001	2
1000100000000000	10	110000000000	6	110000	2
1000010000000000	10	101000000000	6	101000	2
1000001000000000	10	100100000000	6	100100	2
1000000100000000	10	100010000000	6	100010	2

Окончание таблицы В.1

1000000010000000	3	100001000000	6	100001	2
1000000001000000	3	100000100000	6	11000	2
1000000000100000	17	100000010000	6	10100	2
1000000000010000	16	100000001000	3	10010	2
1000000000001000	15	100000000100	3	10001	2
1000000000000100	14	100000000010	13	1100	2
1000000000000010	13	100000000001	12	1010	2
1000000000000001	12	11000000000	5	1001	2
1100000000000000	9	10100000000	5	110	2
1010000000000000	9	10010000000	5	101	2
1001000000000000	9	10001000000	5	11	2
Среднее количество тактов					6,66

Таблица В.2 – Количество тактов для укороченного кода БЧХ (19, 9, 5)

Шаблон ошибки	Кол-во тактов	Шаблон ошибки	Кол-во тактов	Шаблон ошибки	Кол-во тактов
1000000000000000000	11	10000000000100000	7	100001000000	4
1000000000000000000	10	10000000000010000	6	100000100000	4
10000000000000000	9	10000000000001000	5	100000010000	4
1000000000000000	8	10000000000000100	4	100000001000	4
100000000000000	7	10000000000000010	3	100000000100	4
10000000000000	6	10000000000000001	2	100000000010	3
1000000000000	5	1100000000000000	8	100000000001	2
100000000000	4	1010000000000000	8	11000000000	3
10000000000	3	1001000000000000	8	10100000000	3
1000000000	2	1000100000000000	8	10010000000	3
100000000	2	1000010000000000	8	10001000000	3
10000000	2	1000001000000000	8	10000100000	3
1000000	2	1000000100000000	8	10000010000	3
100000	2	1000000010000000	8	10000001000	3
10000	2	1000000001000000	8	10000000100	3
1000	2	1000000000100000	7	10000000010	3
100	2	1000000000010000	6	10000000001	2
10	2	1000000000001000	5	1100000000	2
1	2	1000000000000100	4	1010000000	2
1100000000000000000	11	1000000000000010	3	1001000000	2
1010000000000000000	11	1000000000000001	2	1000100000	2
1001000000000000000	11	1100000000000000	7	1000010000	2
1000100000000000000	11	1010000000000000	7	1000001000	2
1000010000000000000	11	1001000000000000	7	1000000100	2
1000001000000000000	11	1000100000000000	7	1000000010	2
1000000100000000000	11	1000010000000000	7	1000000001	2
1000000010000000000	11	1000001000000000	7	110000000	2
1000000001000000000	11	1000000100000000	7	101000000	2
1000000000100000000	10	1000000010000000	7	100100000	2
1000000000010000000	9	1000000001000000	7	100010000	2

Окончание таблицы В.2

1000000000001000000	8	100000000010000	6	100001000	2
1000000000000100000	7	100000000001000	5	100000100	2
1000000000000010000	6	100000000000100	4	100000010	2
1000000000000001000	5	100000000000010	3	100000001	2
1000000000000000100	4	100000000000001	2	11000000	2
1000000000000000010	3	11000000000000	6	10100000	2
1000000000000000001	2	10100000000000	6	10010000	2
1100000000000000000	10	10010000000000	6	10001000	2
1010000000000000000	10	10001000000000	6	10000100	2
1001000000000000000	10	10000100000000	6	10000010	2
1000100000000000000	10	10000010000000	6	10000001	2
1000010000000000000	10	10000001000000	6	1100000	2
1000001000000000000	10	10000000100000	6	1010000	2
1000000100000000000	10	10000000010000	6	1001000	2
1000000010000000000	10	10000000001000	5	1000100	2
1000000001000000000	10	10000000000100	4	1000010	2
1000000000100000000	9	10000000000010	3	1000001	2
1000000000010000000	8	10000000000001	2	110000	2
1000000000001000000	7	1100000000000	5	101000	2
1000000000000100000	6	1010000000000	5	100100	2
1000000000000010000	5	1001000000000	5	100010	2
1000000000000001000	4	1000100000000	5	100001	2
1000000000000000100	3	1000010000000	5	11000	2
1000000000000000010	2	1000001000000	5	10100	2
1000000000000000001	9	1000000100000	5	10010	2
1100000000000000000	9	1000000010000	5	10001	2
1010000000000000000	9	1000000001000	5	1100	2
1001000000000000000	9	1000000000100	4	1010	2
1000100000000000000	9	1000000000010	3	1001	2
1000010000000000000	9	1000000000001	2	110	2
1000001000000000000	9	110000000000	4	101	2
1000000100000000000	9	101000000000	4	11	2
1000000010000000000	9	100100000000	4		
1000000001000000000	8	100010000000	4		
Среднее количество тактов					5

Приложение Г. Копии свидетельств о регистрации программ для ЭВМ

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2016617787

Вычисление контрольной суммы CRC32 матричным алгоритмом

Правообладатель: ***федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский Томский политехнический университет» (RU)***

Авторы: ***Мыцко Евгений Алексеевич (RU), Мальчуков Андрей Николаевич (RU), Рыжова Светлана Евгеньевна (RU), Зоев Иван Владимирович (RU)***

Заявка № **2016615013**
Дата поступления **17 мая 2016 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***14 июля 2016 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Г.П. Ивлиев

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2016662525

Быстродействующее декодирование кода Боуза-Чоудхури-Хоквингема (15, 7, 5)

Правообладатель: ***федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский Томский политехнический университет» (RU)***

Авторы: ***Рыжова Светлана Евгеньевна (RU), Мальчуков Андрей Николаевич (RU), Мыцко Евгений Алексеевич (RU), Зоев Иван Владимирович (RU)***

Заявка № **2016619897**
Дата поступления **22 сентября 2016 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***14 ноября 2016 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Г.П. Ивлиев

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2018663370

Быстродействующий декодер кода Боуза-Чоудхури-Хоквингема (15,5,7), исправляющий до 3-х независимых ошибок

Правообладатели: ***Рыжова Светлана Евгеньевна (RU), Мыцко Евгений Алексеевич (RU), Мальчуков Андрей Николаевич (RU)***

Авторы: ***Рыжова Светлана Евгеньевна (RU), Мыцко Евгений Алексеевич (RU), Мальчуков Андрей Николаевич (RU)***

Заявка № **2018661121**
Дата поступления **02 октября 2018 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***26 октября 2018 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Г.П. Ивлиев

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2018665738

Поиск образующих полиномов для построения циклических помехоустойчивых кодов с применением технологий параллельных вычислений

Правообладатель: ***Мыцко Евгений Алексеевич (RU)***

Автор: ***Мыцко Евгений Алексеевич (RU)***

Заявка № **2018663768**
Дата поступления **03 декабря 2018 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***10 декабря 2018 г.***

Руководитель Федеральной службы
по интеллектуальной собственности

Г.П. Ивлиев

Приложение Д. Копии актов внедрения результатов диссертационного исследования

Общество с ограниченной ответственностью «ТомИУС-ПРОЕКТ»
(ООО «ТомИУС-ПРОЕКТ»)
Советская ул., дом 114, Томск, Россия, 634034
Тел./факс (3822) 42-64-63, 606-340 E-mail: info@tomics.tomsk.ru http://tomics.tomsk.ru

«УТВЕРЖДАЮ»
Генеральный директор
ООО «ТомИУС-проект»,
_____К.И.Байструков
«03» 09 2018 г.

АКТ О ВНЕДРЕНИИ

результатов диссертационной работы Мыцко Евгения Алексеевича «Алгоритмы и аппаратная реализация на ПЛИС устройств обнаружения и исправления пакетных или независимых ошибок для сообщений короткой длины» на соискание ученой степени кандидата технических наук по специальности 05.13.05 – Элементы и устройства вычислительной техники и систем управления.

Настоящим актом подтверждается, что результаты диссертационной работы Мыцко Е.А. «Алгоритмы и аппаратная реализация на ПЛИС устройств обнаружения и исправления пакетных или независимых ошибок для сообщений короткой длины», а именно устройство обнаружения и исправления пакетных ошибок на основе циклического помехоустойчивого кода (15, 8, 3) с модулем расчета контрольной суммы CRC8 на ПЛИС принят в опытную эксплуатацию для подсистемы синхронизации системы управления установки Токамак КТМ (Национальный ядерный центр, Республика Казахстан, г. Курчатов). Работы по внедрению выполнены в рамках хоз. договора № 4-673/16У от 26.09.2016 «Разработка средств технического обеспечения устройств сбора данных, контроля и защиты электрофизической установки токамак», заключенного между ООО «ТомИУС-проект» и Томским политехническим университетом.

Разработанное устройство на ПЛИС с применением циклического помехоустойчивого кода (15, 8, 3) позволяет повысить надежность передаваемых команд и данных от центрального блока подсистемы синхронизации к локальным модулям синхронизации (ЛМС) и исправлять пакет ошибок длиной до 3-х бит согласно требованиям к надежности и достоверности передаваемых данных в системе. Для контроля ошибок, выходящих за возможности помехоустойчивого кода (15, 8, 3), реализован дополнительный модуль вычисления циклического избыточного кода CRC8 на основе матричного алгоритма.

Общее время работы устройства составляет ~35 нс, что позволило достичь проектной частоты синхронизации локальной шины ЛМС 10 МГц.

Ведущий инженер
ООО «ТомИУС-проект» С. В. Меркулов

TOMSK POLYTECHNIC UNIVERSITY
ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Ministry of Science and Higher Education of the Russian Federation
Federal State Autonomous Educational Institution of Higher Education
«National Research Tomsk Polytechnic University» (TPU)
30, Lenin ave., Tomsk, 634050, Russia
Tel. +7-3822-606333, +7-3822-701779,
Fax +7-3822-606444, e-mail: tpu@tpu.ru, tpu.ru
OKPO (National Classification of Enterprises and Organizations): 02069303,
Company Number: 027000890168,
VAT/KPP (Code of Reason for Registration) 7018007264/701701001, BIC 046902001

Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение высшего образования
«Национальный исследовательский Томский политехнический университет» (ТПУ)
Ленина, пр., д. 30, г. Томск, 634050, Россия
тел.:+7-3822-606333, +7-3822-701779,
факс +7-3822-606444, e-mail: tpu@tpu.ru, tpu.ru
ОКПО 02069303, ОГРН 1027000890168,
ИНН/КПП 7018007264/701701001, БИК 046902001

«УТВЕРЖДАЮ»
Проректор по образовательной деятельности,
_____ А.Р. Вагнер
«30» 04 2019 г.

АКТ О ВНЕДРЕНИИ

результатов диссертационной работы Мыцко Евгения Алексеевича «Алгоритмы и аппаратная реализация на ПЛИС устройств обнаружения и исправления пакетных или независимых ошибок для сообщений короткой длины» в учебный процесс Инженерной школы информационных технологий и робототехники

Настоящим актом подтверждается, что результаты диссертационной работы Мыцко Е.А. используются в учебном процессе в отделении Информационных технологий Инженерной школы информационных технологий и робототехники в виде программ на языках Си и Assembler для проведения занятий по дисциплине «Микропроцессоры и микроконтроллеры» и в виде аппаратных модулей на языке Verilog по дисциплине «Программирование на языках описания аппаратуры» для бакалавров, обучающихся по направлению 09.03.01 «Информатика и вычислительная техника» профиля «Вычислительные машины, комплексы, системы и сети». Использование программ и аппаратных модулей позволяет студентам исследовать быстродействие и аппаратные затраты алгоритмов обнаружения и исправления пакетных или независимых ошибок.

Директор ИШИТР
«30» апреля 2019 г. _____ Д.М. Сонькин

И.о. Руководителя отделения ИТ
«30» апреля 2019 г. _____ В.С. Шерстнёв