ФГБОУВПО «Владимирский государственный университет имени Александра Григорьевича и Николая Григорьевича Столетовых» - ВлГУ

На правах рукописи

Сидоренко Александр Анатольевич

РАЗРАБОТКА И ИССЛЕДОВАНИЕ АДАПТИВНОГО ПОМЕХОУСТОЙЧИВОГО КОДЕРА-ДЕКОДЕРА ДЛЯ ЛОКАЛЬНЫХ СИСТЕМ ТЕЛЕМЕТРИИ

Специальность: 05.12.13 – Системы, сети и устройства телекоммуникаций

Диссертация
на соискание ученой степени кандидата технических наук

Научный руководитель: д.т.н., профессор А.Г. Самойлов

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Актуальность работы. Сегодня в мире постоянно растут объемы информации во всех отраслях деятельности. Системы телеметрии обеспечивают получение, преобразование, передачу по каналу связи, прием, обработку и регистрацию измерительной информации и информации о событиях с целью контроля на расстоянии состояния и функционирования контролируемых объектов.

Далеко не всегда есть возможность связать диспетчерский центр (ДЦ) с контролируемым объектом (КО) кабельным каналом, кроме того, обычно это требует значительных финансовых затрат. Актуальной задачей является организация надёжного цифрового радиоканала передачи данных.

Радиоканал связи ДЦ и КО является слабым звеном систем телеметрии, поскольку именно в нём передаваемые сигналы подвержены искажениям и затуханию из-за негативного воздействия многочисленных факторов. Помехи и замирания снижают достоверность передачи информации. Повышение достоверности, передаваемой по каналу связи информации можно организовать различными способами, например увеличением мощности передатчика, улучшением чувствительности приемника, увеличением усиления антенн. Реализация приведенных способов обычно требует значительных материальных затрат, а главное, не обеспечивает повышение достоверности передаваемой информации при частотно-селективных замираниях.

В настоящее время задача обеспечения достоверности передачи информации всё чаще решается применением помехоустойчивого кодирования, которое представляет собой класс преобразований сигнала, выполняемых для повышения качества связи. Работы таких ученых как Фано Р.М., Форни Г.Д., Омура Д.К., Витерби А.Д., Берлекэмп Э.Р., Хемминг Р.У., Боуз Р.Ч., Рей-Чоудхури Д.К., Хоквингем А.М., Голей М.Д., Рид И.С., Соломон Г.М., Нордстром К.А., Робинсон Д.М., Зубарев Ю.Б., Золотарев В.В., Овечкин Г.В. и многих других сформировали и развили теорию помехоустойчивого

кодирования. Однако проблема выбора вида кодирования для конкретного канала передачи информации пока не решена.

В процессе кодирования происходит преобразование последовательности данных в новую, «улучшенную последовательность», имеющую избыточные символы. Избыточные разряды служат для определения и исправления ошибок. Применение каскадного кодирования, при котором кодирование осуществляется в два уровня (внешним и внутренним кодом) ещё больше улучшает достоверность передачи информации. Количество избыточных бит (степень кодирования) логично варьировать в зависимости от числа ошибок в канале, то есть использовать адаптивное кодирование [59].

Объектом исследований являются локальные системы телеметрии (ЛСТ) в радиоканалах связи которых возникают ошибки.

Предметом исследований является повышение достоверности передачи данных в радиоканалах ЛСТ.

Целью работы является разработка адаптивного каскадного кодера-декодера с исправлением ошибок для повышения достоверности передачи сообщений нерегулярной длины по цифровым радиоканалам связи ЛСТ.

Поставленная цель достигается решением следующих задач:

1). Анализ методов помехоустойчивого кодирования-декодирования с последующим выбором вида кодирования.

2). Синтез алгоритмов кодирования-декодирования и разработка соответствующего программного обеспечения.

3). Проведение теоретических и экспериментальных исследований эффективности кодера-декодера при наличии в канале связи независимых ошибок и пакетов ошибок.

4). Разработка универсального программно-аппаратного комплекса, осуществляющего передачу данных между ЭВМ контролируемых объектов и ЭВМ диспетчерского центра с использованием радиомодемов.

5). Практические исследования эффективности работы программного кодера-декодера в разработанном программно-аппаратном комплексе.

6). Разработка рекомендаций по применению разработанного кодера-декодера в системах передачи данных различных ЛСТ.

Методы исследования. В работе использованы положения теории информации и теории помехоустойчивого кодирования сигналов, методы теории вероятности и математической статистики.

Научная новизна диссертационной работы заключается в разработке новых методов, алгоритмов и устройств, повышающих помехоустойчивость ЛСТ. В диссертационной работе получены следующие основные результаты, содержащие элементы научной новизны:

1). Предложен метод адаптивного каскадного кодирования-декодирования нерегулярных по длине информационных сообщений. Разработаны соответствующие алгоритмы и программное обеспечение.

2). Получены аналитические выражения и проведены практические исследования разработанного кодера-декодера.

3). Даны рекомендации по применению разработанного кодера-декодера для применения в различных системах телеметрии.

4). Разработан универсальный программно-аппаратный комплекс для ЛСТ, осуществляющий передачу данных между ЭВМ контролируемых объектов и ЭВМ диспетчерского центра с применением радиомодемов, использующий в своей работе адаптивный программный кодер-декодер.

Практическая значимость работы заключается в следующем:

1). Разработанный кодек с исправлением ошибок (в зависимости от выбранной конфигурации) обеспечивает исправление пакетов ошибок до 94 бит.

2). Применение разработанного программного кодера-декодера снижает вероятность появления ошибочного бита с $1 \cdot 10^{-2}$ до $7{,}2 \cdot 10^{-9}$, что соответствует выигрышу в величине Eb/No на 2,5 дБ.

3). Разработанный программный кодер-декодер может применяться в разнообразных системах телеметрии и ретрансляции сообщений.

Достоверность и обоснованность результатов обусловлена корректным применением теории информации, теории помехоустойчивого кодирования сигналов, теории вероятности и математической статистики, а также подтверждением теоретических результатов экспериментально полученными данными.

Основные положения, выносимые на защиту

1). Алгоритмы и программы преобразования и кодирования цифровых данных.

2). Адаптивный каскадный кодер-декодер с исправлением ошибок для повышения достоверности передачи нерегулярных по длине сообщений по цифровым радиоканалам связи, содержащим одиночные ошибки и пакеты ошибок.

3). Аналитические выражения для оценки вероятности ошибок на выходе разработанного декодера.

4). Результаты исследования разработанного кодера-декодера.

5). Разработанный универсальный программно-аппаратный комплекс для ЛСТ, осуществляющий передачу данных между ЭВМ контролируемых объектов и ЭВМ диспетчерского центра с применением радиомодемов, использующий в своей работе адаптивный программный кодер-декодер.

Апробация работы, публикации

Основные результаты диссертационного исследования докладывались на 7 научно-технических конференциях различного уровня и опубликованы в 13 публикациях, в том числе в 3-х статьях, включенных ВАК в перечень журналов для диссертационных работ.

Реализация и внедрение результатов работы

Результаты диссертационной работы внедрены и нашли практическое применение в организациях: ООО «Миробэк», г. Москва; ФГБОУ ВПО «Владимирский государственный университет имени Александра Григорьевича и Николая Григорьевича Столетовых». Все результаты внедрения подтверждены соответствующими актами.

Структура и объем работы. Диссертация состоит из четырех глав, введения, заключения, списка сокращений, списка литературы и семи приложений.

ГЛАВА 1. ОБЗОР И АНАЛИЗ СИСТЕМ ТЕЛЕМЕТРИИ

1.1. Задачи систем телеметрии

1.1.1. Объекты и задачи телеметрии

Как правило, системы телеметрии создаются с целью решения двух следующих задач: централизованный оперативный контроль с ДЦ над процессами происходящими на КО, и управление этими процессами [47, 66, 73].

Под КО понимаются производственные, исследовательские, лабораторные, биологические и другие объекты, для которых необходимо производить удалённые измерения и сбор информации для предоставления оператору или пользователю.

По числу КО системы телеметрии разделяются на глобальные (сотни и тысячи объектов) и локальные (единицы и десятки объектов). Также следует отметить, что глобальные системы часто бывают многоуровневыми.

Существуют три основных направления систем телеметрии: телеметрия опасных объектов, технологических объектов, исследовательских объектов.

К опасным объектам относятся: газифицированные объекты; объекты и склады атомной, химической, военной промышленности и т. п. Оперативный контроль параметров опасных объектов помогает предотвратить крупные аварии на производстве.

К технологическим объектам относятся объекты, на которых, для оптимизации и улучшения качества работы или продукции, необходимо знать параметры протекающих процессов. К технологическим объектам можно отнести разнообразные производственные объекты сельского хозяйства, пищевой промышленности, химической, энергетической и и т. п. Так же к технологическим объектам можно отнести объекты водоканала и теплосетей, такие как, насосные станции, тепловые узлы, тепловые пункты. Контроль

параметров объектов в этом случае способствует энергосбережению и улучшению качества производства и услуг.

К исследовательским объектам относятся объекты, параметры которых необходимо исследовать. К таким объектам относятся объекты научно-технических разработок, испытательные аппараты, технологические процессы и тому подобное. Контроль параметров таких объектов является экспериментальной частью исследований, необходим для выявления "слабых мест" и "сильных мест", оптимизации работы, как отдельных частей объекта, так и всего объекта в целом.

1.1.2. Эффекты от внедрения систем телеметрии

Внедрение системы телеметрии требует определенных финансовых и временных затрат, также потребуются затраты на подготовку персонала диспетчерского центра. Как правило эти затраты оправдывают положительные эффекты от внедрения системы телеметрии:

- экономия ресурсов при достижении необходимого качества работы КО;
- повышение качества и эффективности оперативного управления за счет обеспечения диспетчерского и управленческого персонала оперативной информацией о текущем состоянии КО;
- увеличение достоверности и повышение оперативности учета и контроля работы персонала и оборудования КО;
- повышение производительности труда;
- повышение трудовой и технологической дисциплины персонала;
- наглядная оценка деятельности служб и участков КО;
- своевременное техническое обслуживание и ремонт оборудования;
- улучшение условий труда сотрудников.

Особо следует отметить, что в некоторых случаях ЛСТ полностью окупает себя, если с её помощью была предотвращена всего одна авария.

1.2. Проблема повышения достоверности передачи данных

1.2.1. Структурная схема передачи данных от контролируемого объекта на диспетчерский центр

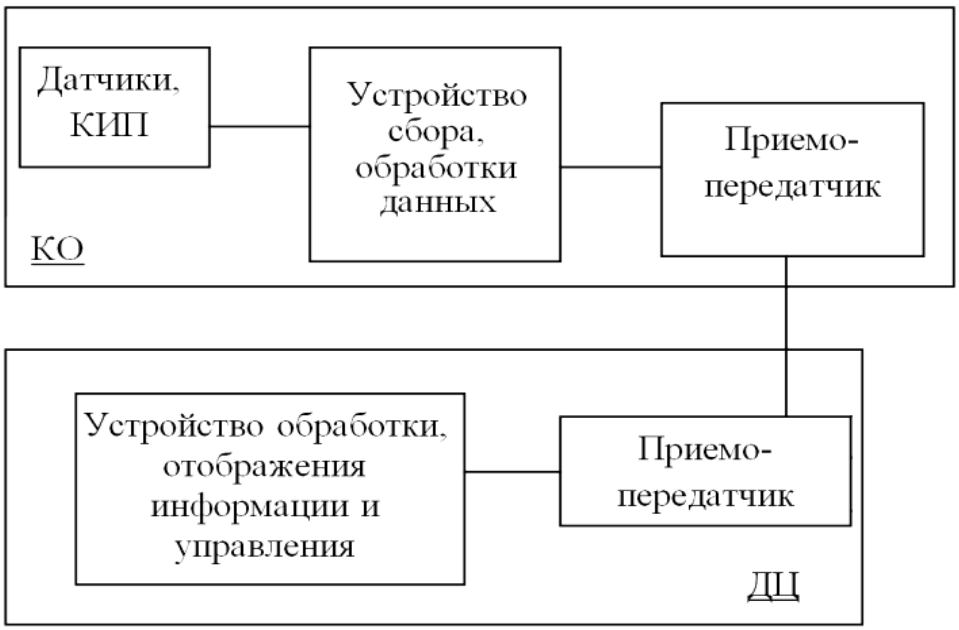


Рисунок 1.1. Структурная схема системы телеметрии

На рисунке 1.1. изображен один из вариантов структурной схема системы телеметрии. На КО установлены контрольно-измерительные приборы (КИП) и датчики. Сигналы от КИП и датчиков поступают на устройство сбора, обработки данных, а затем на передатчик. Далее по каналу передачи данных сигнал попадает на приемник ДЦ. После приемника сигнал попадает на устройство обработки, отображения информации и управления. Оператор воспринимает отображаемую информацию и при необходимости инициирует сигналы управления, проходящие обратным маршрутом

В качестве среды передачи данных используются как беспроводные (радио, GSM/GPRS, ZigBee, WiFi, WiMax, LTE), так и проводные (телефонные, ISDN, xDSL, компьютерные) сети (электрические или оптические) [1, 15, 77, 78, 81].

Недостатком применения GSM-модемов для передачи данных между ДЦ и КО является оплата использования канала, и безусловная зависимость от

оператора сотовой связи. Система перестает работать в случае выхода из строя ближайшей базовой станции и отсутствии достаточного уровня сигнала от других станций. Использование «обычных» (не GSM) радиомодемов не имеет указанных недостатков.

1.2.2. Причины ухудшения качества передачи сигнала

Источником помех в идеальном канале является тепловой шум, генерируемый в приемнике [44, 69, 87]. Тепловой шум имеет, как правило, постоянную спектральную плотность мощности по всей полосе сигнала и гауссову функцию плотности вероятности напряжения с нулевым средним. Сигнал в идеальном канале затухает с расстоянием точно так же, как при распространении в идеальном свободном пространстве. Мощность сигнала убывает пропорционально квадрату расстояния. При таком идеальном распространении, мощность сигнала весьма предсказуема.

Дополнительными источниками потерь в реальном радиоканале являются естественные и искусственные источники шума и помех, негативное влияние которых часто оказываются более значительными, чем тепловой шум приемника [14, 33, 36].

При радиосвязи между КО и ДЦ распространение сигнала происходит в атмосфере и вблизи земной поверхности. Радиосигнал может передаваться от передатчика к приемнику по множеству отражательных путей. Это явление, называемое многолучевым распространением, может вызывать флуктуации амплитуды, фазы и угла прибытия полученного сигнала, что называется замиранием вследствие многолучевого распространения сигнала. Замирание вызывает случайные флуктуации сигнала.

Для типичного радиоканала полученный сигнал состоит из нескольких дискретных многолучевых компонентов, приводящих к расширению сигнала во времени (или дисперсия сигнала).

При дисперсии сигнала типы ухудшений характеристик, возникающих вследствие замирания, разделены на частотно-селективные и частотно-неселективные (амплитудные). При нестационарном поведении канала типы ухудшения характеристик, возникающих вследствие замирания, разделены на быстрые и медленные.

В канале с замираниями взаимосвязь между максимальной избыточной задержкой распространения Тм и временем передачи символа Тс можно рассматривать с позиции двух различных категорий ухудшения качества передачи: частотно-селективного и частотно-неселективного, или амплитудного замирания. Канал обнаруживает частотно-селективное замирание, если Тм больше Тс. Это условие реализуется, когда принятый многолучевой компонент символа выходит за пределы длительности передачи символа. Другим названием этой категории ухудшения передачи вследствие замирания является вводимая каналом межсимвольная интерференция.

Канал является частотно-неселективным, или проявляет амплитудное замирание, если Тм меньше Тс. В этом случае все полученные многолучевые компоненты символа поступают в течение времени передачи символа. В данном случае отсутствуют искажения за счет вводимой каналом межсимвольной интерференции, так как расширение сигнала во времени не приводит к существенному наложению соседних полученных символов. Однако, ухудшение характеристик все же имеет место, поскольку полученные за время длительности символа многолучевые компоненты сигнала могут деструктивно суммироваться, что приводит к значительному ухудшению отношения сигнал/шум.

На рисунке 1.2. [1] в логарифмическом масштабе отражены три основные категории характеристик, выраженных через вероятность битовой ошибки в зависимости от E_b/N_0, где E_b это энергия бита, а N_0 спектральная плотность мощности шума.

Крайняя левая кривая, имеющая экспоненциальную форму, соответствует ожидаемому поведению данной зависимости при использовании любых

номинальных схем модуляции в канале с белым аддитивным гауссовым шумом. Видно, что даже при относительно небольшом уровне E_b/N_0 можно ожидать хорошей достоверности передачи.

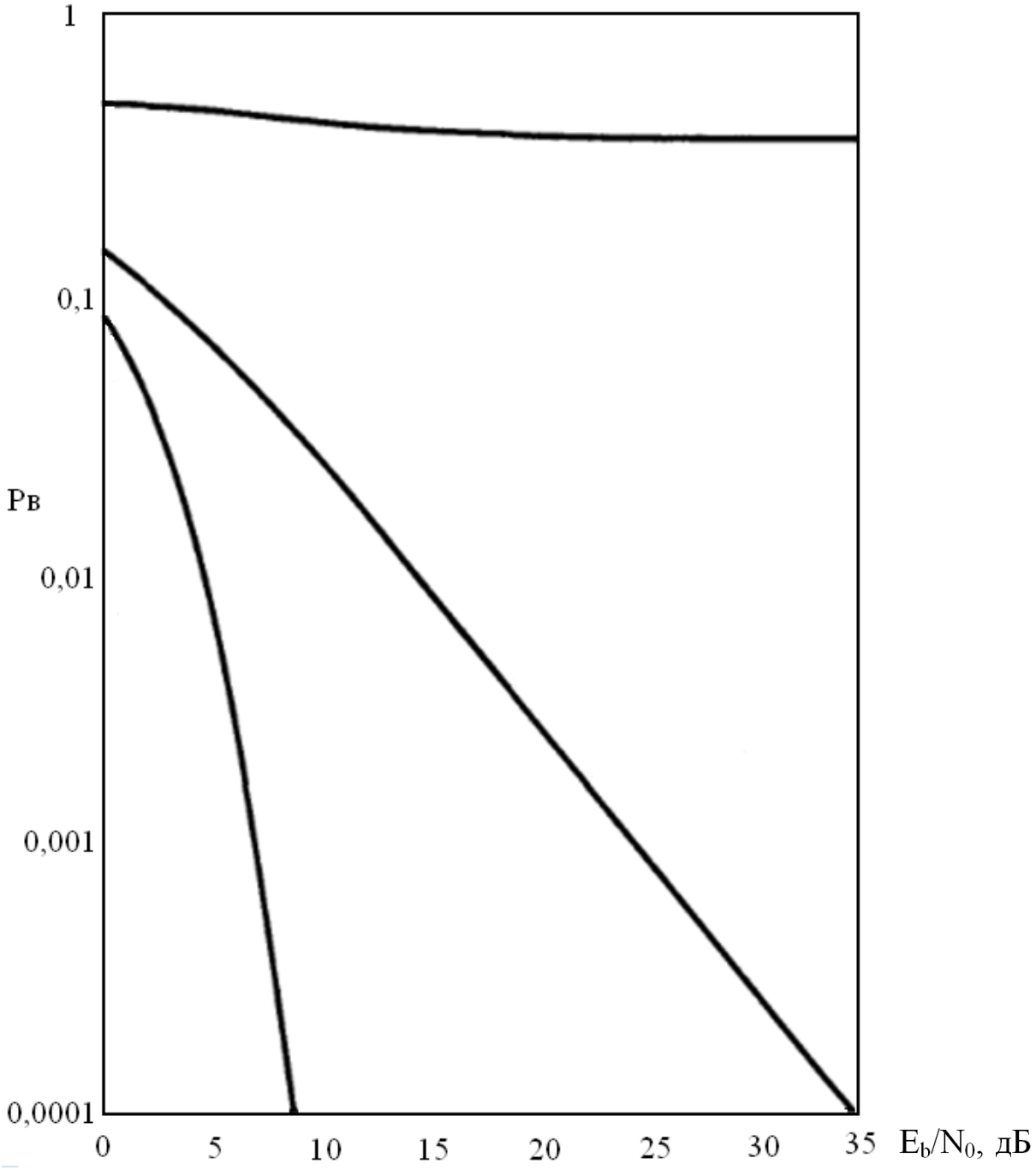


Рисунок 1.2. Достоверность передачи сигналов.

Средняя кривая демонстрирует ухудшение достоверности передачи, вытекающее из уменьшения E_b/N_0, что характерно для амплитудного или медленного замирания. Кривая является функцией, обратно пропорциональной

E_b/N_0 так что для значений E_b/N_0, представляющих практический интерес, характеристики будут плохими.

Верхняя кривая, достигающая непоправимого уровня ошибок, соответствует эффекту сильного ухудшения характеристик, который может проявиться при частотно-селективном или быстром замирании.

1.3. Построение локальных систем телеметрии

1.3.1. Топологические структуры локальных систем телеметрии

От структуры построения ЛСТ зависит время реакции системы, надежность работы системы, стоимость системы, необходимая пропускная способность каналов [63, 74, 83].

Время реакции системы – это время, проходящее между появлением события на контролируемом объекте (изменением контролируемого параметра свыше порогового уровня) и отображением сообщения о нем на ДЦ.

На время реакции системы влияют: топологическая структура комплекса, пропускная способность каналов связи, используемые протоколы обмена и дисциплина обслуживания.

При определении требуемого времени реакции системы мониторинга необходимо руководствоваться соображениями целесообразности, так как снижение времени реакции требует увеличения затрат на создание надежных высокоскоростных каналов связи и изменения аппаратной части комплекса в сторону усложнения.

Имеется три основных варианта топологической структуры ЛСТ: радиальная, магистральная, цепочечная [55, 62, 75].

1). Комплексы с радиальной структурой.

Радиальной структурой считается двухуровневая структура, при которой каждый из КО связан с общим для всех ДЦ отдельным независимым каналом связи (смотри рисунок 1.3.). Комплексы с радиальной структурой имеют самое

низкое время реакции системы, однако требуют большого количества каналов связи и аппаратного усложнения устройства ДЦ для отдельного обслуживания каждого из каналов.

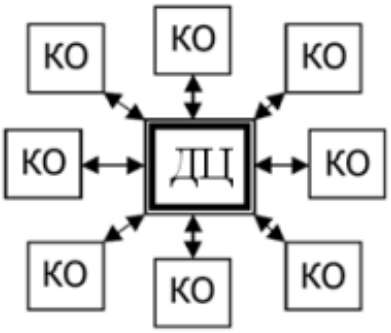


Рисунок 1.3. Комплекс с радиальной структурой

Для каждого направления КО - ДЦ выделен отдельный канал связи или интерфейс. ДЦ производит одновременный прием данных по всем направлениям, поэтому задержка распространения сигнала определяется только пропускной способностью каналов связи. Каналы связи с КО могут иметь любую физическую реализацию, любые скорости обмена, иметь или не иметь модуляцию, обмен может производиться с использованием любого из поддерживаемых протоколов.

Комплексы такого типа рекомендуется использовать в системах, имеющих различные каналы связи с разной пропускной способностью, а также содержащих в своем составе устройства КО других изготовителей.

2). Комплексы с магистральной структурой.

Магистральной структурой считается двухуровневая структура, при которой каждый из КО связан с общим для всех ДЦ общим каналом связи (смотри рисунок 1.4.). Обслуживание КО производится поочередно с использованием запросов со стороны ДЦ, содержащих номер запрашиваемого КП.

При поочередном обслуживании ДЦ посылает запросы на обмен данными в канал связи, последовательно обращаясь к каждому устройству КО по его индивидуальному номеру (адресу). В ответ на запрос ДЦ КО с указанным

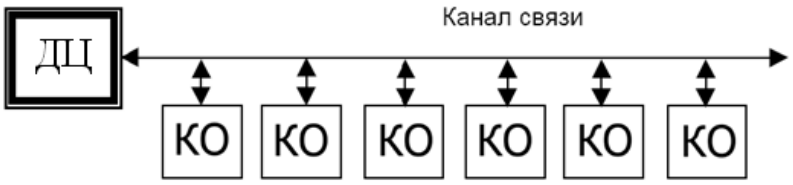


Рисунок 1.4. Комплекс с магистральной структурой

номером, имеющим данные для передачи, посылает их в ДЦ, при этом остальные КО в обмене не участвуют. Время реакции системы складывается из периода опроса КО (определяемого в зависимости от количества КО и объема принимаемой от них информации) и времени передачи информации (определяемой пропускной способностью канала связи). Устройство ДЦ комплекса такой структуры имеет только один коммуникационный адаптер, поэтому его стоимость значительно ниже стоимости ДЦ комплексов радиальной структуры.

3). Комплексы с цепочечной структурой.

Цепочечной структурой считается многоуровневая структура, при которой каждый из КО более высокого уровня является промежуточным пунктом управления для контролируемых пунктов нижнего уровня. На самом верхнем уровне находится ДЦ (смотри рисунок 1.5.). Каждое устройство КО более высокого уровня собирает информацию о состоянии устройств КО, находящихся ниже уровнем, добавляет к ней свою информацию и передает на КО верхнего уровня. КО самого верхнего уровня передает информацию на ДЦ.

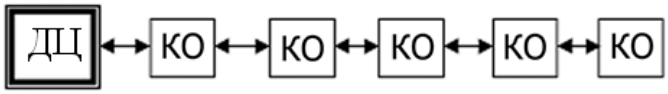


Рисунок 1.5. Комплекс с цепочечной структурой

Комплексы с такой структурой имеют время реакции системы (для всех сигналов и команд), увеличивающееся по мере удаления КО от ДЦ за счет многократной ретрансляции данных. Для эффективной работы комплекса цепочечной структуры пропускная способность каналов связи должна увеличиваться с увеличением уровня иерархии. Устройство ДЦ комплекса такой структуры требует поддержки только одного канала связи или

интерфейса для связи с устройствами нижнего уровня, поэтому его стоимость значительно ниже стоимости ДЦ комплексов радиальной структуры, однако каждое устройство КО должно поддерживать работу по двум каналам (за исключением последнего КО). Важной особенностью комплексов данной структуры является возможность использования различных типов каналов связи, скоростей и протоколов обмена на различных участках цепочечной структуры.

Комплексы с цепочечной структурой целесообразно применять только в случаях, когда использование другой топологической структуры экономически неоправданно из-за отсутствия требуемых каналов связи или высокой стоимости их организации. Использование данной структуры возможно как части комплекса более сложной (комбинированной) структуры. Ввиду роста задержек при обмене информации с ростом иерархических уровней, в системе не рекомендуется использовать более 10 КО.

1.3.2. Варианты организации множественного доступа

Множественный доступ – способ распределения ресурса связи. В нашем случае необходимо организовать связь ДЦ с одним или несколькими КО.

Основные способы организации множественного доступа [84].

1). Частотное разделение.

Каждому КО выделяется определенный поддиапазон используемой полосы частот. Первый частотный диапазон f_0-f_1, второй частотный диапазон f_2-f_3, третий f_4-f_5 и так далее. Области спектра, находящиеся между используемыми диапазонами, называют защитными полосами частот f_1-f_2, f_3-f_4 и так далее. Защитные полосы выполняют роль буфера, что позволяет снизить интерференцию между соседними каналами.

2). Временное разделение.

Каждому КО предоставляется весь частотный диапазон, но в строго определенные интервалы времени. Промежутки времени, разделяющие

используемые интервалы, называются защитными интервалами. Защитные интервалы выступают в роли буфера, снижая тем самым интерференцию.

В простейшем случае применения временного разделения время разбито на интервалы, называемые кадрами, а каждый кадр, в свою очередь, сам делится на временные интервалы, которые могут быть распределены между КО.

3). Кодовое разделение.

Для каждого КО выделяются определенные элементы набора ортогонально (или почти ортогонально) распределенных спектральных кодов, каждый из которых использует весь диапазон частот.

4). Пространственное разделение.

С помощью точечных лучевых антенн на ДЦ радиосигналы разделяются и направляются в разные стороны. Данный метод допускает одновременное использование всего частотного диапазона.

5). Поляризационное разделение.

Для разделения сигналов применяется ортогональная поляризация, что позволяет использовать один частотный диапазон.

1.3.3. Варианты информационного обмена.

Основные варианты информационного обмена ДЦ с КО:

- ДЦ производит бесконечный поочередный опрос КО с фиксацией изменений контролируемых параметров;
- КО сами без запросов поочередно докладывают на ДЦ изменения контролируемых параметров;
- КО производят доклад одновременно;
- КО без запросов докладывают на ДЦ изменения контролируемых параметров, только если эти изменения превысили некоторый пороговый уровень.

1.3.4. Выбор варианта построения универсального программно-аппаратного комплекса

Ряд современных серийно выпускаемых радиомодемов («Невод-5», «Спектр-433» и др.) имеет режим «прямого доступа», который даёт пользователям возможность реализации собственных методов кодирования-декодирования и протоколов обмена данных. Преимуществом указанных радиомодемов является также и их низкая стоимость (20 тысяч рублей в ценах 2014 года в комплекте с антенной).

Адаптивный программный кодер-декодер для нерегулярной по длине последовательности данных совместно с программными модулями осуществляющими обмен данных с модемом и прием информационных данных будут программной основой универсального программно-аппаратного комплекса (далее - УПАК) для ЛСТ, осуществляющего передачу данных между ЭВМ КО и ЭВМ ДЦ с использованием радиомодемов. На рисунке 1.6. изображена структурная схема КО и ДЦ, осуществляющих обмен данными таким образом. Описанный низкобюджетный УПАК позволяет строить систему передачи данных с учетом особенностей конкретной ЛСТ.

Использование программного кодера-декодера из-за широкого распространения ЭВМ и своей гибкости имеет во многих случаях преимущества перед аппаратными средствами кодирования-декодирования. Кроме того, построение аппаратных средства декодирования требует значительных финансовых затрат.

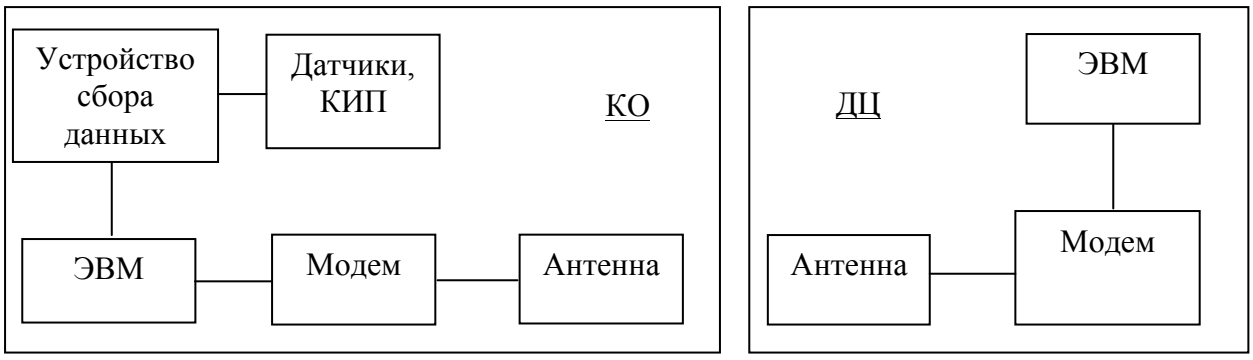


Рисунок 1.6. Структурная схема КО и ДЦ

Выводы по главе 1

Системы телеметрии создаются с целью решения двух следующих задач: централизованный оперативный контроль с диспетчерского центра над процессами, происходящими на контролируемых объектах, и управление этими процессами.

Внедрение системы телеметрии требует определенных финансовых и временных затрат. Как правило, это оправдывают положительные эффекты от внедрения системы телеметрии. Особо следует отметить, что в некоторых случаях ЛСТ полностью окупает себя, если с её помощью была предотвращена всего одна авария.

Построение ЛСТ с использованием радиомодемов имеет ряд преимуществ, например: отсутствие проводного соединения, независимость от оператора сотовой связи.

Источником помех в идеальном канале является тепловой шум, генерируемый в приемнике. Дополнительными источниками потерь в реальном радиоканале являются замирания, естественные и искусственные источники шума и помех, негативное влияние которых часто оказывается более значительным, чем тепловой шум приемника.

Имеется три основных варианта топологической структуры ЛСТ (радиальная, магистральная, цепочечная) и пять основных способов организации множественного доступа (с частотным, временным, кодовым, пространственным и поляризационным разделением каналов).

Для демонстрации возможностей разрабатываемого программного кодера-декодера наиболее доступным является вариант построения системы передачи данных на базе персональной ЭВМ с подключенным радиомодемом.

ГЛАВА 2. РАССМОТРЕНИЕ И ВЫБОР ВАРИАНТОВ ПОМЕХОУСТОЙЧИВОГО КОДИРОВАНИЯ

2.1. Варианты повышения помехозащищенности ЛСТ

Как уже говорилось в п. 1.2, основными факторами снижающими достоверность передачи информации являются помехи и замирания в канале передачи информации [64, 68, 79]. Рассмотрим теперь более подробно методы борьбы с этими отрицательными факторами.

2.1.1. Борьба с частотно-селективными замираниями

Для борьбы с вызванной каналом межсимвольной интерференцией, которая возникает вследствие частотно-селективного замирания, может использоваться выравнивание [33, 35, 52, 89]. Процесс выравнивания для уменьшения воздействия межсимвольной интерференции заключается в использовании методов, собирающих рассеянную энергию символа в ее исходный временной интервал. По сути, эквалайзер (устройство выравнивания) является обратным фильтром канала. Если канал является частотно-селективным, эквалайзер усиливает частотные компоненты с малыми амплитудами и ослабляет с большими. Целью комбинации канала и выравнивающего фильтра является получение плоской частотной характеристики и линейного изменения фазы. Поскольку характеристика канала может со временем меняется, выравнивающий фильтр должен изменяться или приспосабливаться к нестационарным характеристикам канала. Следовательно, такие фильтры являются адаптивными устройствами. Поскольку ослабление искажений выполняется путем сбора рассеянной энергии символа в исходный временной интервал символа, так чтобы это не мешало детектированию других символов, эквалайзер попутно предоставляет приемнику энергию символа, которая в противном случае была бы утрачена.

2.1.2. Борьба с быстрыми замираниями

Для борьбы с быстрыми замираниями можно увеличить скорость передачи символов $W=1/T_c$, чтобы она превышала скорость замирания $f=1/T_o$, путем введения избыточности сигнала [34, 35, 90]. Кодирование с коррекцией ошибок может также вносить улучшения. Взамен повышения энергии сигнала код снижает E_b/N_0, требуемое для получения заданной достоверности передачи информации. При кодировании мы можем допустить более высокий уровень ошибок в сигналах, поступающих от демодулятора, сохранив заданную достоверность передачи данных.

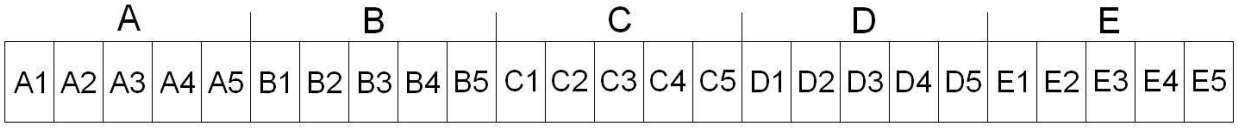
Чтобы воспользоваться преимуществами кодирования, ошибки вне демодулятора не должны коррелировать, что обычно бывает в среде с быстрым замиранием, либо в систему должно внедряться устройство чередования.

2.1.3. Чередование

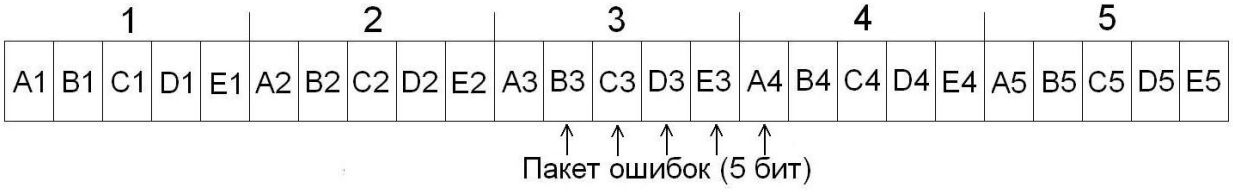
Распространение сигнала в среде с быстрыми замираниями приводит к появлению коррелированных ошибок имеющих вид пакетов. Пакеты ошибок также возникают из-за наличия в каналах импульсных помех. Большинство блочных и сверточных кодов, о которых будет рассказано в п. 2.2., разрабатывается для борьбы с независимыми случайными ошибками. Чередование бит [1, 46, 57] кодированного сообщения перед передачей и обратная операция после приема приводят к рассеиванию пакета ошибок во времени: таким образом, они становятся для декодера случайно распределенными. Идея, лежащая в основе метода чередования бит, заключается в разнесении символов кодовых слов во времени. Получаемые промежутки времени точно так же заполняются символами других кодовых слов. Разнесение символов во времени эффективно превращает канал с памятью в канал без памяти и, следовательно, позволяет использовать коды с

коррекцией случайных ошибок в системах с замираниями и импульсными помехами.

Устройство чередования смешивает кодовые символы в промежутке нескольких длин блоков (для блочных кодов) или нескольких длин кодового ограничения (для сверточных кодов). Требуемый промежуток определяется длительностью пакета. Подробности структуры перераспределения бит должны быть известны приемнику, чтобы иметь возможность выполнить восстановление порядка бит перед декодированием. На рисунке 2.1. показан простейший пример чередования. На рисунке 2.1.а. показаны исходные кодовые слова от A до E без чередования. Пусть каждое кодовое слово состоит из пяти кодовых символов. Допустим, что наш код может исправлять однобитовые ошибки в любой 5-символьной последовательности. Если промежуток памяти канала равен длительности одного кодового слова или мы просто имеем дело с импульсной ошибкой длинной в 5 символов, то такой 5-символьной пакет может уничтожить информацию не более чем в двух кодовых словах. На рисунке 2.1.б. показано, что после получения кодированных данных кодовые символы затем перемешиваются. То есть каждый кодовый символ каждого кодового слова отделяется от своего соседа на расстояние пяти символов. Полученный поток затем преобразуется в модулированный сигнал и передается по каналу.



а. исходное слово



б. слово после чередования

Рисунок 2.1. Пример чередования бит

Как можно видеть на рисунке 2.1.б. последовательные канальные пакеты шума попадают на пять символьных промежутков, влияя на один кодовый символ каждого из пяти исходных кодовых слов. Во время приема в потоке вначале восстанавливается исходный порядок бит, так что он становится похож на исходную кодированную последовательность, изображенную на рисунке 2.1.а. Далее поток декодируется. Поскольку в каждом кодовом слове возможно исправление одиночной ошибки, импульсная помеха не оказывает никакого влияния на конечную последовательность.

2.1.4. Кодирование с исправлением ошибок

На рисунке 2.2. [1] приведен характерный вид зависимости вероятности появления битовой ошибки от нормированного значения энергии бита, для случаев передачи не кодированной и кодированной последовательности бит. Применение кодирования с исправлением ошибок позволяет при тех же значениях энергии бита (излучаемой мощности) получить более высокую достоверность передачи данных [23, 37, 39, 40, 48].

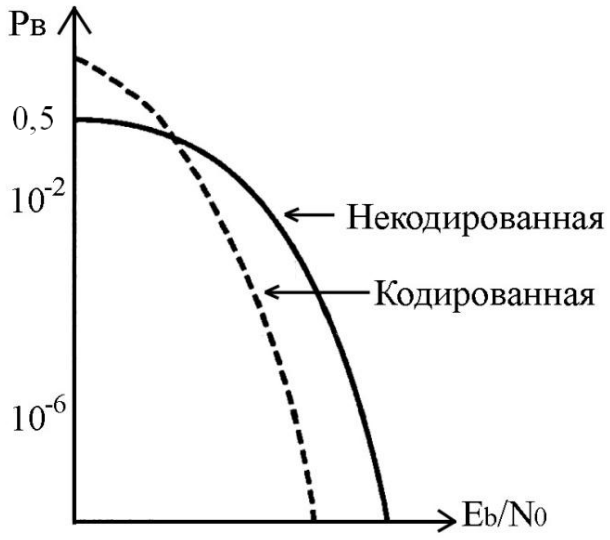


Рисунок 2.2. Достоверность передачи с кодированием и без него

2.2. Свойства различных методов помехоустойчивого кодирования

2.2.1. Блочные коды

Важное семейство кодов образуют линейные двоичные блочные коды [2, 3, 12, 21, 67]. Эти коды замечательны тем, что представляя информационные и кодовые слова в форме двоичных векторов, мы можем описать процессы кодирования и декодирования с помощью аппарата линейной алгебры, при этом компонентами вводимых векторов и матриц являются символы 0 и 1. Операции над двоичными компонентами производятся по правилам двоичной арифметики.

Кодер двоичного блочного (n, k) – кода (смотри рисунок 2.3.) отображает множество 2^k возможных двоичных информационных слов в множество 2^k n – мерных кодовых слов. Где k – число информационных символов (символов в информационном слове), а n – число кодовых символов (символов в кодовом слове). В теории кодирования между этими множествами всегда существует однозначное соответствие.

Рисунок 2.3. Кодер блокового кода

Вместо k бит информационного вектора в канал передается n бит кодового вектора. К каждому блоку данных кодирующее устройство прибавляет (n - k) избыточных бит, которые также называют контрольными битами или битами чётности. Отношение числа избыточных бит к числу информационных бит, (n - k)/k называется избыточностью кода. Отношение числа бит данных к общему числу бит, k/n, называется степенью кодирования. Степень кодирования показывает долю кода, которая приходится на полезную информацию.

Кодирование линейного блокового (n, k)-кода задается порождающей матрицей G размером (k x n). Таким образом, кодовое слово v и информационное слово u связаны соотношением v = u*G.

Правильно декодировать можно далеко не все модели ошибки. Возможности кода для исправления ошибок в первую очередь определяются его структурой. Весовой коэффициент Хэмминга w(U) кодового слова U определяется как число ненулевых элементов в U. Для двоичного вектора это эквивалентно числу единиц в векторе. Например, если U=100101101, то w(U)=5. Расстояние Хэмминга между двумя кодовыми словами U и V, обозначаемое как d(U,V), определяется количеством элементов, в которых они отличаются. Согласно свойствам сложения по модулю 2, можно отметить, что сумма двух двоичных векторов является другим двоичным вектором, двоичные единицы которого расположены на тех позициях, которыми эти векторы отличаются. Таким образом, можно видеть, что расстояние Хэмминга между двумя векторами равно весовому коэффициенту их суммы, то есть d(U,V)= w(U+V). Также видно, что весовой коэффициент кодового слова равен его расстоянию до нулевого вектора. Наименьший элемент из множества расстояний между парами кодовых слов называется минимальным расстоянием кода и обозначается dmin. Минимальное расстояние дает нам меру минимальных возможностей кода и, следовательно, характеризует его помехоустойчивость. Сумма двух произвольных кодовых слов дает другой элемент кодовых слов, что является свойством линейных кодов. Если U и V кодовые слова, то и W=U+V тоже должно быть кодовым словом. Следовательно, расстояние между двумя кодовыми словами равно весовому коэффициенту третьего кодового слова. Таким образом, минимальное расстояние линейного кода соответствует минимальному весу кодового слова. Иными словами, минимальное расстояние соответствует наименьшему из множества расстояний между нулевым кодовым словом и всеми остальными кодовыми словами.

Линейный код с минимальным расстоянием Хэмминга $d_{min} \geq 2t+1$ может обнаружить dmin-1 ошибок и исправить t ошибок. Код корректирующий ошибки в t битах называется совершенным.

Систематический линейный блочный код это код, в котором n символов кодового слова содержат k символов информационного сообщения. Таким образом, кодовое слово систематического кода имеет n-k бит четности и k информационных символов.

Порождающую матрицу любого систематического кода всегда можно путем перестановки столбцов к виду:

$$G_{kxn}=(P_{kx(n-k)}\ I_k), \qquad (2.1.)$$

где нижние индексы обозначают размерность матрицы, а I_k единичная матрица размером (k x k).

Цикличные коды являются подмножеством линейных кодов. Линейный (n, k)-код является циклическим, если циклический сдвиг любого кодового слова также является также кодовым словом этого кода. Циклический сдвиг соответствует сдвигу всех компонент на один разряд вправо, причем, освободившееся место слева занимает крайняя правая компонента.

2.2.2. Свёрточные коды

Свёрточные коды [1, 4, 24, 45] это коды, использующие непрерывную обработку потока данных короткими блоками. Свёрточный кодер имеет память и символы на его выходе зависят не только от очередного блока символов на входе, но и от предыдущих символов. Свёрточное кодирование является отображением информационной последовательности символов в кодовую последовательность с помощью линейной схемы с параметрами, не меняющимися во времени.

Выходная последовательность n-кодовых символов, получаемая при свёрточном кодировании, является функцией не только одной входной последовательности k-информационных символов, но и предыдущих К-1 входных последовательностей. Целое число К является параметром, называемым длинной кодового ограничения, - оно указывает число разрядов в

кодирующем регистре сдвига в которые помещаются информационные символы.

Теоретически входная последовательность бит бесконечна, но на практике свёрточным кодам обычно придают блоковую структуру, устанавливая состояние сверточного кодера в некоторое заранее известное состояние (например нулевое).

Декодирование свёрточных кодов, обычно, производится по довольно сложному алгоритму Витерби. Алгоритм Витерби проводит декодирование согласно критерию максимального правдоподобия.

Свёрточные коды эффективно борются с одиночными ошибками, но плохо справляются с пакетами ошибок. Более того, ошибка декодера приводит к образованию на его выходе пакета ошибок. Свёрточные коды нашли широкое применение в системах спутниковой связи [85].

2.2.3. Каскадные коды и турбокоды

В каскадных кодах кодирование осуществляется в два уровня [8, 22, 56]. Входные данные сначала кодируются внешним кодом, а затем внутренним, после чего осуществляется модуляция сигнала. Декодирование происходит в обратном порядке. Искаженные каналом данные с демодулятора поступают сначала на декодер внутреннего кода, а затем на декодер внешнего кода. Структурная схема системы каскадного кодирования изображена на рисунке 2.4.

Достоинством каскадных кодов является относительно низкая сложность кодирующих и декодирующих устройств, так как каскадные коды позволяют выполнить процедуры кодирования и декодирования по этапам, применяя на каждом этапе достаточно короткие по сравнению с результирующим коды.

Каскадные коды позволяют реализовать достаточно большое кодовое расстояние, поэтому их применение в каналах с высоким уровнем помех эффективно.

Турбокод [26, 27, 29-30, 88] является развитием системы каскадного кодирования путем применения итеративного декодирования. Основная идея турбо-кодирования заключается в подаче мягкого решения с выхода одного декодера на вход другого и повторение этой процедуры до тех пор, пока не будет достигнута необходимая точность принятия решения.

Методы декодирования турбокодов имеют большую вычислительную сложность. Этого недостатка лишены методы многопорогового декодирования [91, 92]. Итеративные алгоритмы при каждом изменении корректируемых ими символов всегда находят строго более правдоподобные решения.

В системах передачи данных использующей радиомодемы, выдающие жесткое решение, применение турбокодов и многопорогового кодирования затруднительно.

Рисунок 2.4. Структурная схема системы каскадного кодирования

2.3. Обзор часто применяемых на практике кодов

2.3.1. Коды Хемминга

Коды Хемминга [1-3, 82] образуют важное семейство простейших линейных блоковых кодов. Для каждого натурального числа $m \geq 3$ существует двоичный код Хэмминга со следующими параметрами:

- длина кодовых слов $n = 2^m - 1$;
- число информационных разрядов $k = 2^m - 1 - m$;
- число проверочных разрядов $m = n - k$;

- корректирующая способность t = 1, dmin = 3.

Код Хэмминга требует минимальной избыточности при заданной длине блока для исправления одной ошибки. Код Хемминга является совершенным кодом.

Из всего вышесказанного видно, что код Хемминга или только обнаруживает все ошибки кратностью не выше 2, или только исправляет все однократные ошибки. Преимуществом данного кода является его простота, и как следствие высокие скорости кодирования и декодирования. Недостатком является его способность исправлять лишь одиночные ошибки.

2.3.2. Код Голея

Код Голея [1-3, 76] это совершенный код с параметрами n=23, k=12 и минимальным расстоянием Хэмминга равным 7. Этот код гарантирует исправление всех трехбитовых ошибок. Преимуществом данного кода является относительно простой алгоритм декодирования и способность противостоять даже трехбитовым ошибкам. Недостатком является высокая избыточность кода.

2.3.3. Коды Боуза-Чоудхури-Хоквенгема

Коды Боуза-Чоудхури-Хоквенгема (БЧХ) [7, 25, 80] представляют собой развитие кодов Хэмминга. Коды БЧХ являются циклическими и позволяют исправлять множественные ошибки. Данный вид кодов предоставляет большую свободу выбора длины блока, степени кодирования, размеров алфавита и возможностей коррекции ошибок.

В случае когда кодовые слова состоят из нескольких сотен символов коды БЧХ дают значительный выигрыш по сравнению с другими блочными кодами имеющими ту же длину и степень кодирования. Наиболее часто в кодах БЧХ применяются кодовые слова длинной $n = 2^m - 1$, где m = 3, 4, 5… Для кодов

БЧХ максимальная эффективность кодирования достигается при степенях кодирования между 1/3 и 3/4. Одним из подклассов кодов БЧХ с недвоичными символами являются коды Рида Соломона.

Преимуществом двоичных кодов БЧХ является их разнообразие и хорошие возможности по борьбе с одиночными ошибками. Недостатками являются довольно сложные алгоритмы декодирования (особенно для кодов большой длины) и невозможность противостоять пакетам ошибок.

2.3.4. Коды Рида-Соломона

Код Рида-Соломона (РС) [31, 32, 38, 41] это недвоичный случай кода БЧХ. Недвоичные это означает, что символы этих кодов представляют собой многобитовые
(m-битовые) последовательности. Коды РС имеют минимальное расстояние $d_{min} = n – k + 1$ и способны исправлять $t=](n – k)/2[$ ошибок.

Преимуществом кодов РС является их возможность противостоять пакетам ошибок [51]. Недостатками являются сложные алгоритмы декодирования.

2.4. Выбор метода кодирования

2.4.1. Теоретические исследования эффективности кодов

Проведем исследование эффективности кода Хэмминга с параметрами: число кодовых бит n=7, число информационных бит k=4, число гарантированно исправляемых ошибок t=1.

Ошибка в декодировании кодового слова P_X возникает в случае наличия в слове 2 ошибочных бит, при условии, что хотя бы один из них находится в информационной части слова.

В соответствии с положениями теории вероятности [10, 19, 20, 41], вероятность данного события P_X равна сумме вероятностей двух событий:

1) в проверочной части один «ошибочный» бит, в информационной – один;

2) в проверочной части нет «ошибочных» бит, в информационной – два.

$$P = P_3(1)P_4(1)+P_3(0)P_4(2) = C_3^1 p(1-p)^{3-1} \cdot C_4^1 p(1-p)^{4-1}+C_3^0(1-p)^{3-0} \cdot C_4^2 p^2(1-p)^{4-2} = \frac{3!}{1!2!}p(1-p)^2 \cdot \frac{4!}{1!3!}p(1-p)^3 + \frac{3!}{0!3!}(1-p)^3 \cdot \frac{4!}{2!2!}p^2(1-p)^2 = \frac{3!}{1!2!} \cdot \frac{4!}{1!3!}p^2(1-p)^5 + \frac{4!}{2!2!}p^2(1-p)^5 = 12p^2(1-p)^5 + 6p^2(1-p)^5 = 18p^2(1-p)^5 \quad (2.6.)$$

Где p – вероятность появления ошибочного бита в канале передачи данных. Тогда получаем следующие результаты:

При p=10^{-2}: $P_{\mathbf{X}} = 1{,}7 \cdot 10^{-3}$ и $P_{\mathbf{X1000}} = 4{,}3 \cdot 10^{-1}$.

При p=10^{-3}: $P_{\mathbf{X}} = 1{,}8 \cdot 10^{-5}$ и $P_{\mathbf{X1000}} = 4{,}5 \cdot 10^{-3}$.

При p=10^{-4}: $P_{\mathbf{X}} = 1{,}8 \cdot 10^{-7}$ и $P_{\mathbf{X1000}} = 4{,}5 \cdot 10^{-5}$.

При p=10^{-5}: $P_{\mathbf{X}} = 1{,}8 \cdot 10^{-9}$ и $P_{\mathbf{X1000}} = 4{,}5 \cdot 10^{-7}$.

Для проведения сравнения эффективности кодов найдена вероятность ошибки декодирования информационного сообщения длинной 1000 бит закодированного кодом Хэмминга (P_{X1000}).

Проведем исследование эффективности кода Голея.

Ошибка в декодировании кодового слова $P_Г$ возникает в случае наличия в слове 4 ошибочных бит, при условии, что хотя бы один из них находится в информационной части слова.

Вероятность данного события равна сумме вероятностей четырех событий:

1) в проверочной части два «ошибочных» бита, в информационной – два;

2) в проверочной части три «ошибочный» бит, в информационной – один;

3) в проверочной части один «ошибочный» бит, в информационной – три;

4) в проверочной части нет «ошибочных» бит, в информационной – четыре.

$$P = P_{11}(2)P_{12}(2) + P_{11}(3)P_{12}(1) + P_{11}(1)P_{12}(3) + P_{11}(0)P_{12}(4) = C_{11}^2 p^2(1-p)^{11-2} \cdot C_{12}^2 p^2(1-p)^{12-2} + C_{11}^3 p^3(1-p)^{11-3} \cdot C_{12}^1 p(1-p)^{12-1} + C_{11}^1 p(1-p)^{11-1} \cdot C_{12}^3 p^3(1-p)^{12-3} + C_{11}^0 p^0(1-p)^{11-0} \cdot C_{12}^4 p^4(1-p)^{12-4} =$$

$$\frac{11!}{2!9!}p^2(1-p)^9 \cdot \frac{12!}{2!10!}p^2(1-p)^{10} + \frac{11!}{3!8!}p^3(1-p)^8 \cdot \frac{12!}{1!11!}p(1-p)^{11} + \frac{11!}{1!10!}p(1-p)^{10} \cdot \frac{12!}{3!9!}p^3(1-p)^9 + \frac{11!}{0!11!}p^0(1-p)^{11} \cdot \frac{12!}{4!8!}p^4(1-p)^8 = \frac{11!}{2!9!} \cdot \frac{12!}{2!10!}p^4(1-p)^{19} + \frac{11!}{3!8!} \cdot \frac{12!}{1!11!}p^4(1-p)^{19} + \frac{11!}{1!10!} \cdot \frac{12!}{3!9!}p^4(1-p)^{19} + \frac{12!}{4!8!}p^4(1-p)^{19} = \frac{10 \cdot 11}{2} \cdot \frac{11 \cdot 12}{2}p^4(1-p)^{19} + \frac{9 \cdot 10 \cdot 11}{6} \cdot \frac{12}{1}p^4(1-p)^{19} + \frac{11}{1} \cdot \frac{10 \cdot 11 \cdot 12}{6}p^4(1-p)^{19} + \frac{9 \cdot 10 \cdot 11 \cdot 12}{24}p^4(1-p)^{19} = 3630p^4(1-p)^{19} + 1980p^4(1-p)^{19} + 2420p^4(1-p)^{19} + 495p^4(1-p)^{19} = 8525p^4(1-p)^{19} \quad (2.8)$$

Где p – вероятность появления ошибочного бита в канале передачи данных, тогда:

При p=10^{-2}: $P_Г = 7{,}0 \cdot 10^{-5}$ и $P_{Г1000} = 5{,}8 \cdot 10^{-3}$.

При p=10^{-3}: $P_Г = 8{,}4 \cdot 10^{-9}$ и $P_{Г1000} = 6{,}9 \cdot 10^{-7}$.

При p=10^{-4}: $P_Г = 8{,}5 \cdot 10^{-13}$ и $P_{Г1000} = 7{,}1 \cdot 10^{-11}$.

При p=10^{-5}: $P_Г = 8{,}5 \cdot 10^{-17}$ и $P_{Г1000} = 7{,}1 \cdot 10^{-15}$.

Проведем исследование эффективности кода БЧХ с параметрами: n=15, k=7, t=2.

Ошибка в декодировании кодового слова $P_{БЧХ}$ возникает в случае наличия в слове 3 ошибочных бит, при условии, что хотя бы один из них находится в информационной части слова.

Вероятность данного события $P_{БЧХ}$ равна сумме вероятностей трех событий:

1) в проверочной части два «ошибочных» бита, в информационной – один;

2) в проверочной части один «ошибочный» бит, в информационной – два;

3) в проверочной части нет «ошибочных» бит, в информационной – три.

$$P = P_8(2)P_7(1) + P_8(1)P_7(2) + P_8(0)P_7(3) = C_8^2p^2(1-p)^{8-2} \cdot C_7^1p(1-p)^{7-1} + C_8^1p(1-p)^{8-1} \cdot C_7^2p^2(1-p)^{7-2} + C_8^0p^0(1-p)^{8-0} \cdot C_7^3p^3(1-p)^{7-3} = \frac{8!}{2!6!}p^2(1-p)^6 \cdot \frac{7!}{1!6!}p(1-p)^6 + \frac{8!}{1!7!}p(1-p)^7 \cdot \frac{7!}{2!5!}p^2(1-p)^5 + \frac{8!}{0!8!}(1-p)^8 \cdot \frac{7!}{3!4!}p^3(1-p)^4 = \frac{8!}{2!6!} \cdot \frac{7!}{1!6!}p^3(1-p)^{12} + \frac{8!}{1!7!} \cdot \frac{7!}{2!5!}p^3(1-p)^{12} + \frac{7!}{3!4!}p^3(1-p)^{12} =$$

$$\frac{7\cdot8}{2}\cdot\frac{7}{1}p^3(1-p)^{12}+\frac{8}{1}\cdot\frac{6\cdot7}{2}p^3(1-p)^{12}+\frac{5\cdot6\cdot7}{6}p^3(1-p)^{12}=196p^3(1-p)^{12}+$$
$$168p^3(1-p)^{12}+42p^3(1-p)^{12}=406p^3(1-p)^{12} \quad (2.7)$$

Где p – вероятность появления ошибочного бита в канале передачи данных, тогда:

При $p=10^{-2}$: $P_{БЧХ}=3{,}6\cdot10^{-4}$ и $P_{БЧХ1000}=5{,}2\cdot10^{-2}$.

При $p=10^{-3}$: $P_{БЧХ}=4\cdot10^{-7}$ и $P_{БЧХ1000}=5{,}7\cdot10^{-5}$.

При $p=10^{-4}$: $P_{БЧХ}=4\cdot10^{-10}$ и $P_{БЧХ1000}=5{,}7\cdot10^{-8}$.

При $p=10^{-5}$: $P_{БЧХ}=4\cdot10^{-13}$ и $P_{БЧХ1000}=5{,}7\cdot10^{-11}$.

Проведем исследование эффективности кода РС при разной вероятности появления ошибочного бита в канале.

Для исследований возьмем код РС с параметрами n=9, k=5, t=2. Выберем размер символа кода РС равным 1 байту, что удобно для ЭВМ. В результате кодирования каждое кодовое слово содержит 5 информационных символов и 4 проверочных символа. Структура такого кодового слова содержащего девять 8-битных символов (72 бит) изображена на рисунке 2.5.

1 байт	2 байт	3 байт	4 байт	5 байт	6 байт	7 байт	8 байт	9 байт
Проверочные символы				Информационные символы				

Рисунок 2.5.

Такой код способен исправить два любых ошибочных символа. Наличие трех ошибочных символов приводит к ошибке декодирования слова, за исключением случая, когда все три ошибочных бита находятся среди в проверочной части слова.

Вероятность ошибки декодирования слова приблизительно равна вероятности появления трех ошибочных бит в сообщении, при условии, что все три ошибочных бита расположены в разных байтах сообщения и хотя бы один из них находиться в информационной части слова. Разберем этот случай подробнее с точки зрения теории вероятности исходя из заданной вероятности появления ошибочного бита в канале.

Вероятность данного события P_{PC} равна сумме вероятностей следующих трех событий:

1) в проверочной части два «ошибочных» байта, в информационной – один;

2) в проверочной части один «ошибочный» байт, в информационной – два;

3) в проверочной части нет «ошибочных» байт, в информационной – три.

$$\begin{aligned} P &= P_4(2)P_5(1) + P_4(1)P_5(2) + P_4(0)P_5(3) = C_4^2 q^2 (1-q)^{4-2} C_5^1 q (1-q)^{5-1} + \\ &+ C_4^1 q (1-q)^{4-1} C_5^2 q^2 (1-q)^{5-2} + C_4^0 q^0 (1-q)^{4-0} C_5^3 q^3 (1-q)^{5-3} = \\ &= \frac{4!}{2!2!}\frac{5!}{1!4!} q^3 (1-q)^6 + \frac{4!}{1!3!}\frac{5!}{2!3!} q^3 (1-q)^6 + \frac{5!}{2!3!} q^3 (1-q)^6 = 30q^3(1-q)^6 + 40q^3(1-q)^6 + 10q^3(1-q)^6 = \\ &= 80q^3(1-q)^6 \end{aligned} \quad (2.2.)$$

Где q – вероятность наличия в 1 байте 1 ошибочного бита. Найдем эту вероятность по формуле Бернулли

$$q = P_8(1) = C_8^1 p(1-p)^{8-1} = \frac{8!}{1!7!} p(1-p)^7 = 8p(1-p)^7 \quad (2.3.)$$

Где p – вероятность появления ошибочного бита в канале передачи данных, тогда:

При $p=10^{-2}$: $q=7{,}4\cdot10^{-2}$; $P_{PC} = 2{,}0\cdot10^{-2}$; $P_{PC1000} = 5{,}0\cdot10^{-1}$.

При $p=10^{-3}$: $q=7{,}9\cdot10^{-3}$; $P_{PC} = 3{,}8\cdot10^{-5}$; $P_{PC1000} = 9{,}5\cdot10^{-4}$.

При $p=10^{-4}$: $q=8{,}0\cdot10^{-4}$; $P_{PC} = 4{,}1\cdot10^{-8}$; $P_{PC1000} = 1{,}0\cdot10^{-6}$.

При $p=10^{-5}$: $q=8{,}0\cdot10^{-5}$; $P_{PC} = 4{,}1\cdot10^{-11}$; $P_{PC1000} = 1{,}0\cdot10^{-9}$.

Проведем исследование эффективности работы кодера-декодера РС при наличии пакетов ошибок в принятом сообщении. Рассмотрим вероятность правильного декодирования сообщения при наличии пакетов ошибок разной длинны.

Рассматриваемый код в состоянии исправить два любых искаженных 8-битных символа в кодовом слове. Пакеты ошибок длиной менее 10 бит всегда затрагивают не более 2 байт и успешно исправляются. При определенном расположении, пакет ошибок длиной 10 бит уже может исказить более двух символов (смотри рисунок 2.6.) и если это затронет хотя бы один информационный бит, то декодер будет не в состоянии исправить ошибку.

Вероятность этого события при сообщении длинной в 1 кодовое слово равна 0,11. При дальнейшем увеличении длины пакета, вероятность ошибки на выходе декодера будет возрастать. В случае, когда сообщение состоит из одного кодового слова, максимальная длинна пакета, при которой возможно правильное декодирование, составит 32 бита. Этот случай, когда пакет ошибок полностью занимает всю проверочную часть кодового слова (4 байта), - смотри рисунок 2.7. Вероятность ошибки на выходе декодера в этом случае равна 0,975.

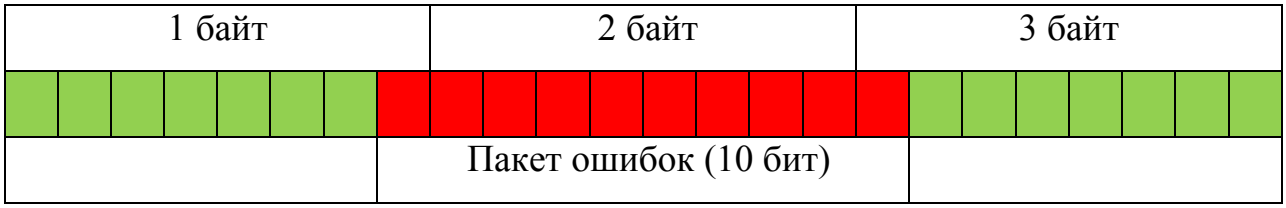


Рисунок 2.6. Воздействие пакета ошибок на 3 байта.

В случае сообщения длиной в 2 и более слов, максимальная длинна пакета, при которой возможно правильное декодирование, составит 48 бит. Этот случай когда пакет ошибок полностью занимает всю проверочную часть кодового слова и ещё 2 байта предыдущего слова (смотри рисунок 2.8.). Вероятность ошибки на выходе декодера при этом равна 0,988, для случая когда сообщение состоит из 3 кодовых слов.

Рассчитаем вероятность появления пакета ошибок длинной 10 бит в случае наличия в канале передачи данных лишь независимых ошибок с вероятностью появления ошибочного бита p. Расчеты проведем для случая кодового сообщения максимальной длинны (смотри п. 2.4.2.) - четырёх кодовых слов суммарной длинной 288 бит.

Вероятность того, что 10 бит из 10 будут ошибочными равна:

$$w = P_{10}(10) = C_{10}^{10} p^{10} (1-p)^{10-10} = p^{10} \qquad (2.4.)$$

Кортеж из 10 ошибочных бит представим как 1 условный «ошибочный бит». Тогда, получаем что в нашем сообщении 278 правильных бит и 1 «ошибочный». При этом, тогда необходимо найти вероятность того, что среди 279 бит будет 1 «ошибочный». Данная вероятность будет равна

$$P_{279}(1) = C_{279}^{1}\, w(1-w)^{279-1} = \frac{279!}{1!278!} w(1-w)^{278} = 279w(1-w)^{278} \quad (2.5.)$$

При $p=10^{-2}$: $w=10^{-20}$ и $P_{279}=2{,}79\cdot10^{-18}$.

При $p=10^{-3}$: $w=10^{-30}$ и $P_{279}=2{,}79\cdot10^{-28}$.

Полученные вероятности крайне малы и ими можно пренебречь.

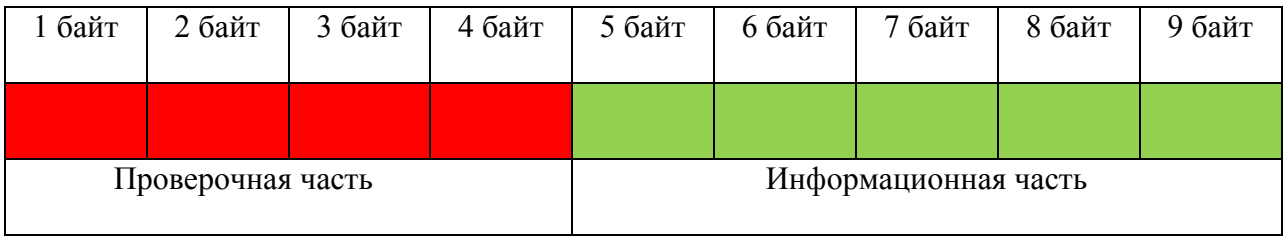


Рисунок 2.7. Пакет ошибок полностью занимает всю проверочную часть кодового слова.

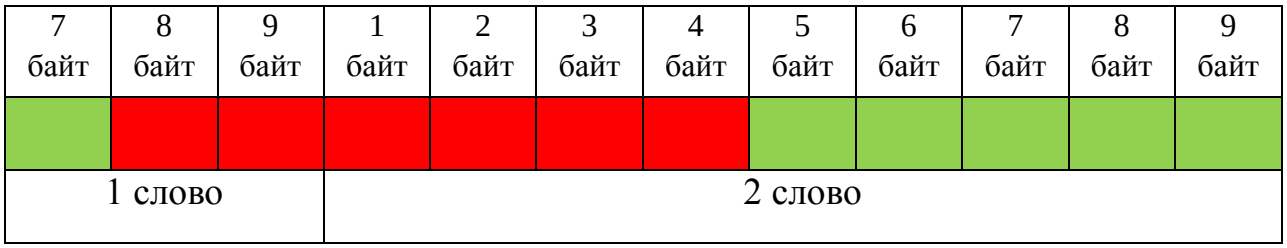


Рисунок 2.8. Пакет ошибок полностью занимает всю проверочную часть кодового слова и два байта предыдущего кодового слова.

Способные противостоять пакетам ошибок коды РС показали низкую эффективны в борьбе с независимым ошибкам, особенно при высокой вероятности появления ошибочного бита. При вероятности появления ошибочного бита равной $p=10^{-2}$ коды Хэмминга оказались эффективнее.

На рисунке 2.9. изображена зависимость вероятности ошибочного декодирования информационного сообщения длиной 1000 бит, закодированного с использованием рассмотренных выше кодов, от вероятности появления ошибочного бита в канале связи.

Проанализировав полученные зависимости можно сделать следующие выводы. Простейший код Хемминга при вероятности появления ошибочного бита в канале равной 10^{-2} сравним по эффективности с кодом РС. Код БЧХ способен значительно эффективней противостоять независимым ошибкам, чем

код Хэмминга, и несколько эффективней, чем код РС. Код Голея значительно эффективней борется с независимым ошибкам, чем все рассмотренные коды.

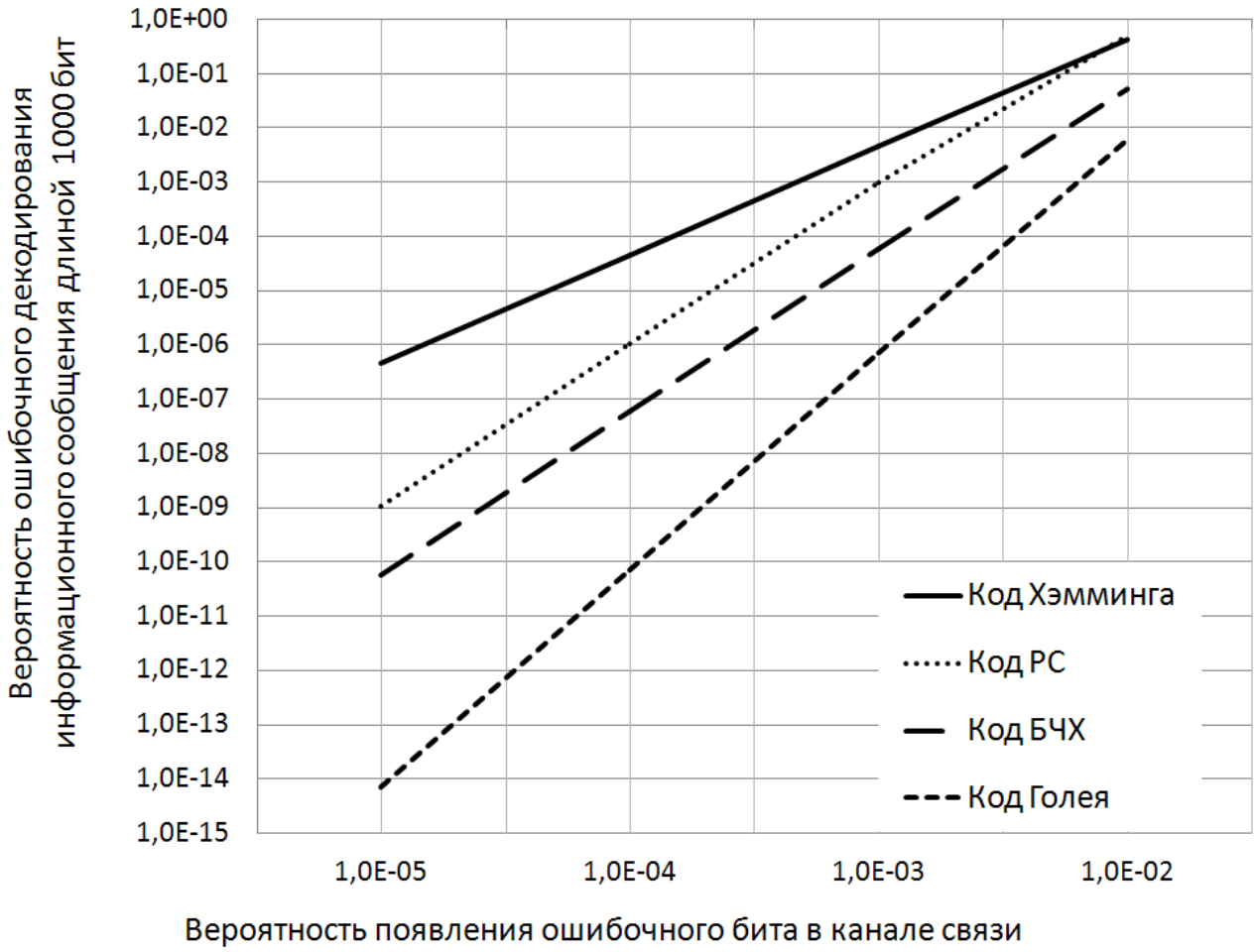


Рисунок 2.9. – Сравнение эффективности кодов при наличии независимых ошибок в канале связи

2.4.2. Выбор метода кодирования

В настоящее время качество проводных и оптоволоконных каналов связи довольно высоко. В компьютерных сетях связи, использующих такие каналы для передачи данных, могут успешно применяться простейшие двоичные коды, которые справляясь с редко возникающими независимыми ошибками обеспечивают необходимую достоверность передачи данных.

В случае ЛСТ использующей радиоканал для передачи данных имеется высокая вероятность появления как независимых ошибок, так и пакетов

ошибок. Поэтому представляется нецелесообразным применение сверточных кодов, способных порождать дополнительные пакеты ошибок. Применение простейших двоичных кодов типа кодов Хэмминга здесь не имеет смысла даже при переспросе, ведь повторная посылка с высокой долей вероятности будет также содержать ошибки. Такие коды вместо исправления ошибок будут вносить новые. Двоичные коды БЧХ более успешно решат проблему возросшей вероятности появления независимых ошибок, однако и они с пакетами ошибок эффективно не справляются.

Решение задач борьбы с длинными пакетами ошибок под силу недвоичным кодам РС. Код РС не искажается ошибками внутри m-битового символа. Коду РС все равно сколько бит повреждено в символе, - 1 или m. Для большей эффективности будем использовать код РС в качестве внешнего кода в каскаде с внутренним двоичным кодом [49, 54, 61, 65]. Разновидность двоичного кода, применяемого в каскаде целесообразно изменять адекватно помеховой обстановке (числу ошибок, переспросов).

2.4.3. Оценки эффективности каскадного кодирования

Рассмотренный нами в п. 2.4.1. код РС одно кодовое слово которого состоит из 72 бит удобно соединять в каскад как с рассмотренным кодом Хемминга, так и с кодом Голея (23, 12). В первом случае минимальный кодовый элемент каскадного кода (МКЭ) образуется следующим образом: 40 информационных бит кодируются в одно кодовое слово РС (72 бита) из которого в свою очередь образуется 18 информационных слов для кода Хемминга (смотри рисунок 2.10.), что после кодирования образует 126 кодовых бит. Во втором случае МКЭ образуется следующим образом: 40 информационных бит кодируются в одно кодовое слово РС (72 бита) из которого в свою очередь образуется 6 информационных слов для кода Голея (смотри рисунок 2.11), что после кодирования образует 138 кодовых бит.

Как было рассмотрено в п. 2.4.1., вероятность ошибки декодирования слова РС приблизительно равна:

$$P = 80q^3(1-q)^6 \qquad (2.9.)$$

9 символов (байт) кодового слова РС																	
1 байт		2 байт		3 байт		4 байт		5 байт		6 байт		7 байт		8 байт		9 байт	
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
18 информационных слов кода Хемминга																	

Рисунок 2.10. Одно кодовое слово РС соответствует 18 информационным словам Хемминга.

9 символов (байт) кодового слова РС								
1 байт	2 байт	3 байт	4 байт	5 байт	6 байт	7 байт	8 байт	9 байт
1	2	3	4	5	6			
6 информационных слов кода Голея								

Рисунок 2.11. Одно кодовое слово РС соответствует 6 информационным словам Голея.

Где q – вероятность наличия ошибки в 1 символе (байте). Найдем вероятность ошибки декодирования слова РС при каскадном кодировании $P_{РСХ}$ исходя из того, что 1 символ кода РС состоит из 2 информационных слов кода Хемминга. Вероятность ошибки декодирования слова Хемминга определена в п. 2.4.1.

Вероятность наличия ошибки в кодовом символе РС равна сумме вероятностей двух событий:

1) из двух слов Хемминга одно содержит ошибки;

2) из двух слов Хемминга оба содержат ошибки.

$$q = P_2(1)+P_2(2) = C_2^1 P_{\text{Х}}(1-P_{\text{Х}})^1 + P_{\text{Х}}^2 = 2P_{\text{Х}}(1-P_{\text{Х}}) + P_{\text{Х}}^2 =$$
$$2P_{\text{Х}} - 2P_{\text{Х}}^2 + P_{\text{Х}}^2 = 2P_{\text{Х}} - P_{\text{Х}}^2 \qquad (2.10.)$$

Пусть р – вероятность появления ошибочного бита в канале передачи данных, тогда вероятность появления ошибки при декодировании МКЭ такого каскадного кода:

При $p=10^{-2}$: $q=3{,}4\cdot10^{-3}$; $P_{\mathbf{РСХ}}=3{,}1\cdot10^{-6}$; $P_{\mathbf{РСХ1000}} = 7{,}8\cdot10^{-5}$.

При $p=10^{-3}$: $q=3{,}6\cdot10^{-5}$; $P_{\mathbf{РСХ}}=3{,}7\cdot10^{-12}$; $P_{\mathbf{РСХ1000}} = 9{,}3\cdot10^{-11}$.

При p=10^{-4}: q=3,6·10^{-7}; $P_{РСХ}$ =3,7·10^{-18}; $P_{РСХ1000}$ = 9,3·10^{-17}.

При p=10^{-5}: q=3,6·10^{-9}; $P_{РСХ}$ =3,7·10^{-24}; $P_{РСХ1000}$ = 9,3·10^{-23}.

Найдем вероятность ошибки декодирования слова РС при каскадном кодировании $P_{РСХ}$ исходя из того, что 3 символа кода РС состоят из 2 информационных слов кода Голея, а одно кодовое слово РС образуется 6 информационных слов для кода Голея. Как видно из рисунка 2.10. ошибка в 1 слове кода Голея приводит к искажению 1 или 2 символов кода РС (в зависимости от количества и расположения ошибочных бит), поэтому найти точное значение $P_{РСГ}$ довольно сложно. Приняв ряд допущений и найдем оценочное значение $P_{РСГ}$.

Ошибочное декодирование двух слов кода Голея приводит к искажению от 2 до 4 символов кода РС и в большинстве случаев ведет к ошибке в декодировании слова РС. Вероятность появления ошибки в 2 словах кода Голея из 6:

$$P = C_6^2 P_Г^2 (1 - P_Г)^{6-2} = \frac{6!}{2!4!} P_Г^2 (1 - P_Г)^4 = 15 P_Г^2 (1 - P_Г)^4 \approx P_{РСГ} \qquad (2.11.)$$

При p=10^{-2}: $P_{Г}$ = 7,0·10^{-5}; $P_{РСГ}$ ≈ 7,4·10^{-8}; $P_{РСГ1000}$ = 1,8·10^{-6}.

При p=10^{-3}: $P_{Г}$ = 8,4·10^{-9}; $P_{РСГ}$ ≈ 1,05·10^{-15}; $P_{РСГ1000}$ = 2,6·10^{-14}.

При p=10^{-4}: $P_{Г}$ = 8,5·10^{-13}; $P_{РСГ}$ ≈ 1,1·10^{-22}; $P_{РСГ1000}$ = 2,7·10^{-21}.

При p=10^{-5}: $P_{Г}$ = 8,5·10^{-17}; $P_{РСГ}$ ≈ 1,1·10^{-29}; $P_{РСГ1000}$ = 2,7·10^{-28}.

Проведём оценку эффективности каскадного кода при наличии в канале пакетов ошибок.

Найдем для кодовой последовательности закодированной каскадным кодом РС+Х максимально возможную для исправления длину пакета ошибок.

В случае если пакет ошибок воздействует на 12 кодовых слов Хемминга, которые после декодирования преобразуются в 2 информационных и 4 проверочных символы РС (смотри рисунок 2.12.), то сообщение декодируется правильно. Если пакет ошибок увеличить справа и слева на один символ (исправимый кодом Хемминга) то сообщение также декодируется правильно. В итоге мы имеем максимальную длину исправимого пакета ошибок – 86 бит.

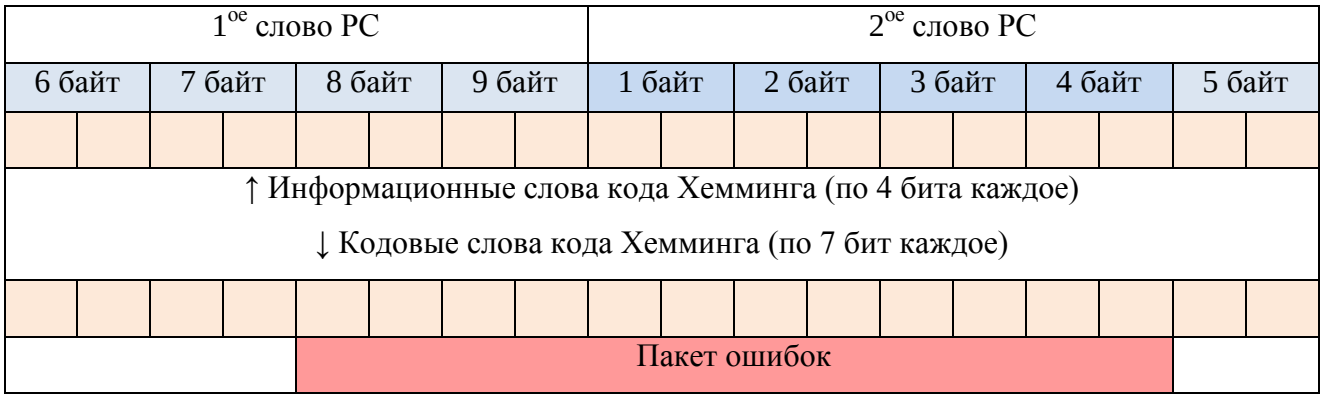


Рисунок 2.12. Воздействие пакета ошибок на кодовую последовательность при каскадном кодировании кодом РС+Х

Найдем для кодовой последовательности закодированной каскадным кодом РС+Г максимально возможную для исправления длину пакета ошибок.

В случае если пакет ошибок воздействует на кодовые слова Голея, которые после декодирования преобразуются в 2 информационных и 4 проверочных символы РС (смотри рисунок 2.13.), то сообщение декодируется правильно. Шесть символов кода РС образуются тремя словами кода Голея (69 бит) с примыкающими к ним справа и слева частями еще двух слов. Слева примыкают 4 информационных бита слова. Справа – 8 информационных бит слова, которые предваряют 11 отбрасываемых при кодировании проверочных бит. В итоге мы имеем максимальную длину исправимого пакета ошибок – 92 бит.

Рисунок 2.13. Воздействие пакета ошибок на кодовую последовательность при каскадном кодировании кодом РС+Г

Выводы по главе 2

В случае ЛСТ использующей радиоканал для передачи данных имеется высокая вероятность появления как независимых ошибок, так и пакетов ошибок. Применение простейших двоичных кодов типа кодов Хэмминга здесь не имеет смысла даже при переспросе, ведь повторная посылка с высокой долей вероятности будет также содержать ошибки. Такие коды вместо исправления ошибок будут вносить новые. Двоичные коды БЧХ более успешно решат проблему возросшей вероятности появления независимых ошибок, однако и они с пакетами ошибок эффективно не справляются.

Решение задач борьбы с пакетами ошибок под силу недвоичным кодам РС. Код РС не искажается ошибками внутри m-битового символа. Коду РС все равно сколько бит повреждено в символе, - 1 или m. Для большей эффективности будем использовать код РС в качестве внешнего кода в каскаде с внутренним двоичным кодом. Разновидность двоичного кода, применяемого в каскаде целесообразно изменять адекватно помеховой обстановке (числу ошибок, переспросов).

Рассмотренный код РС (n=9; k=5) одно кодовое слово которого состоит из 72 бит удобно соединять в каскад как с кодом Хемминга (n=7; k=4), так и с кодом Голея. В первом случае образуется 18 информационных слов для кода Хемминга. Во втором случае образуется 6 информационных слов для кода Голея.

Расчеты показали, что каскадный код способен эффективно противостоять как одиночным ошибкам, так и пакетам ошибок.

ГЛАВА 3. РАЗРАБОТКА ПРОГРАММ ПОМЕХОУСТОЙЧИВОГО КОДИРОВАНИЯ И ДЕКОДИРОВАНИЯ

3.1. Разработка программ кодирования и декодирования кода Хемминга

В этой главе будет проведена разработка программ кодирования и декодирования для кодов Хемминга, Голея, двоичных кодов БЧХ, кодов РС и каскадного кода.

Рассмотрим распространенный систематический код Хемминга (n=7, k=4). Этот код, с учетом формулы 2.1., задается порождающей матрицей G_{7x4}:

$$G_{4\times 7} = (P_{4\times 3}, I_4) = \begin{pmatrix} 1101000 \\ 0110100 \\ 1110010 \\ 1010001 \end{pmatrix} \quad (3.1.)$$

Задача декодирования заключается в том, чтобы по принятому слову r восстановить информационное слово u. При декодировании кодов Хемминга, БЧХ, как и многих других кодов, важное значение имеет понятие синдрома. Синдром S это результат проверки на четность принятого сообщения. Если синдром принятого сообщения равен нулю, то данное сообщение принадлежит набору кодовых слов, если нет, то принятое слово содержит ошибку.

Для вычисления синдрома определим вначале для нашего систематического кода проверочную матрицу H:

$$H_{(n-k)\times n} = (I_{n-k}, P^T_{k\times(n-k)}) \quad (3.2.)$$

$$H_{3\times 7} = (I_3, P^T_{4\times 3}) = \begin{pmatrix} 1001011 \\ 0101110 \\ 0010111 \end{pmatrix} \quad (3.3.)$$

Тогда система проверочных уравнений, определяющих синдром будет выглядеть следующим образом:

$$s = r \times H^T = r \times \begin{pmatrix} 100 \\ 010 \\ 001 \\ 110 \\ 011 \\ 111 \\ 101 \end{pmatrix} \quad (3.4.)$$

После вычисления синдрома существуют два основных метода дальнейшего декодирования кодов Хемминга: табличный метод и метод декодирования по проверочной матрице H.

Табличный метод основан на получении таблицы соответствия позиции одиночной ошибки получающемуся при этом синдрому.

Метод декодирования по проверочной матрице основан на том, что синдром найденный при наличии ошибки в $i^{ом}$ символе равен $i^{ому}$ столбцу проверочной матрицы.

В приложении №1 приведена программа осуществляющая:

- кодирование кодом Хемминга вводимой пользователем последовательности информационных бит;

- искажение кодового слова в соответствии с задаваемым пользователем количеством и номерами ошибочных бит;

- декодирование искаженного слова.

Программные реализации разработаны на языке программирования C++, который в настоящее время является одним из наиболее распространенных международный языков программирования [18, 43, 71, 72].

3.2. Разработка программы кодирования кодом БЧХ

Символы кода БЧХ берут из конечного поля Галуа [4, 11, 53, 58]. Прежде чем приступить к разработке алгоритмов и программ кодирования и декодирования кодов БЧХ, необходимо научиться формировать элементы полей, проводить арифметические операции с ними и разработать соответствующее программное обеспечение.

3.2.1. Введение в поля Галуа

Полем называют множество элементов, если для любых элементов этого множества определены операции сложения и умножения а также выполняется ряд аксиом (замкнутости, ассоциативности, коммутативности, дистрибутивности …).

В каналах связи множество передаваемых сигналов всегда конечно. Поля с конечным числом элементов q называют полями Галуа по имени их первого исследователя Эвариста Галуа и обозначают GF(q). Число элементов поля q называют порядком поля. Конечные поля используются для построения большинства известных кодов и их декодирования.

Двоичное поле GF(2) является простейшим полем Галуа, операции сложения и умножения в котором проводятся по правилам арифметики по модулю 2. Двоичное поле применяется при построении двоичных кодов БЧХ.

В высшей математике доказано, что число элементов конечного поля q всегда удовлетворяет условию

$$q=p^m, \qquad (3.5.)$$

где p – простое число называемое характеристикой поля, а m – положительное целое.

Если m=1, то такое поле называется простым, если m>1, такое поле называется расширенным.

В простом поле операции сложения и умножения выполняются по модулю p, а сами элементы образуют последовательность состоящую из чисел {0, 1, 2, …, p-1}.

В простом поле существует, по крайней мере, один примитивный элемент α, такой, что каждый не нулевой элемент GF(p) может быть представлен как некоторая степень α по модулю p.

Если простое поле образуется из чисел, то расширенное поле образуется из многочленов. Таким образом, можно провести и аналогию формирования поля.

Элементы простого поля формируются из степеней примитивного элемента по модулю простого числа p, а элементы расширенного поля формируются из степеней примитивного элемента по модулю примитивного полинома. В качестве примитивного элемента обычно берётся число 2.

Примитивный полином – это нередуцируемый полином f(X) порядка m, если наименьшим положительным целым числом n, для которого X^n+1 делится на f(X) без остатка, будет $n=2^m-1$. Нередуцируемый полином – это полином, который нельзя представить в виде произведения полиномов меньшего порядка.

Поле, образованное многочленами над полем GF(p) по модулю примитивного многочлена f(X) степени m, называется расширением поля степени m над GF(p) или расширенным полем. Оно содержит p^m элементов и обозначается $GF(p^m)$. Поле $GF(p^m)$ содержит в качестве подмножества все элементы поля GF(p).

Для построения кодов БЧХ используются символы из поля расширения $GF(2^m)$. Каждый элемент поля $GF(2^m)$ можно представить в виде слова длинны m над полем GF(2) или многочлена с двоичными коэффициентами.

$$a = a(X) = a_{m-1}X^{m-1} \oplus \ldots \oplus a_2X^2 \oplus a_1X \oplus a_0, \text{ где } a_{m-1},\ldots, a_2, a_1, a_0 \in \{0, 1\} \quad (3.6.)$$

Где $\oplus$ обозначает операцию сложения по модулю 2. Учитывая, что коэффициенты $a_{m-1},\ldots,a_1,a_2,a_0$ являются либо 0 либо 1, каждый элемент конечного поля $GF(2^m)$ можно представить двоичным вектором или просто двоичным числом. В некоторых случаях удобно представлять элементы поля десятичными, восьмеричными или шестнадцатеричными числами.

Бесконечное множество элементов образуется из стартового множества $\{0, 1, \alpha^1\}$ добавлением дополнительных элементов путем умножения последнего элемента на α. Тогда бесконечное множество элементов поля будет представлено как $\{0, \alpha^0, \alpha^1, \alpha^2, \alpha^3\ldots\}$.

Рассмотрим образование конечного множества содержащего 2^m элементов $\{0, \alpha^0, \alpha^1, \alpha^2, \dots, \alpha^{(2^m-2)}\}$. Из формулы 3.6. мы знаем, что максимальный порядок полинома m-1, но очевидно, что через m-1 проведенных умножений на α (которые эквивалентны сдвигу полинома), этот порядок будет превышен. Если у нас на очередном шаге формирования элементов поля получилось, что порядок полинома элемента равен m-1 (то есть, очевидно, что при сдвиге полученного полинома произойдет выход за ограничение порядка полинома), то при формировании следующего элемента после сдвига полинома мы вычитаем из результата примитивный полином.

Сложение и вычитание полиномов осуществляется как суммирование по модулю 2 всех коэффициентов при элементах одинаковых степеней.

3.2.2. Построение поля $GF(2^8)$

Во второй главе был выбран используемый нами метод кодирования и обоснованы параметры каскадного кода. Применяемый в качестве внутреннего кода двоичный код БЧХ, как было сказано выше, строится с использованием простейшего двоичного поля Галуа GF(2). Применяемый в качестве внешнего кода код РС с размером символа равным 1 байту строится с использованием поля Галуа $GF(2^8)$.

Для построения поля $GF(2^8)$ [16, 58] выберем примитивный полином $f(X) = X^8 \oplus X^4 \oplus X^3 \oplus X^2 \oplus 1$ (что эквивалентно двоичному вектору 100011101), который определяет конечное поле $GF(2^m)$, где степень полинома m=8. Поле, определяемое выбранным нами полиномом, имеет $2^m=2^8=256$ элементов от $0^{го}$ до $255^{го}$. Для удобства рассмотрения процесса формирования поля, равным нулю обозначим не $0^{ой}$ элемент, а $255^{ый}$.

Основываясь на рассуждениях п. 3.1.1., представим математическое описание процесса формирования элементов поля.

A[0]=1, A[255]=0;

При A[i-1]<{10000000} A[i]=A[i-1]<<1;

При A[i-1]=>{10000000} A[i]=(A[i-1]<<1)⊕100011101.

Где символом << обозначен сдвиг влево двоичного представления числа.

Таблица 3.1. Разные виды представления элементов поля GF(2^8)

степень α (номер элемента поля)	Двоичное число	Десятичное число	степень α (номер элемента поля)	Двоичное число	Десятичное число
0	00000001	1	12	11001101	205
1	00000010	2	13	10000111	135
2	00000100	4	14	00010011	19
3	00001000	8	15	00100110	38
4	00010000	16	16	01001100	76
5	00100000	32	17	10011000	152
6	01000000	64	18	00101101	45
7	10000000	128	19	01011010	90
8	00011101	29	20	10110100	180
9	00111010	58	21	01110101	117
10	01110100	116	22	11101010	234
11	11101000	232	23	11001001	201

В таблице 3.1. показаны разные виды представления элементов поля GF(2^8) для первых 24 элементов.

Более привычно и компактно представлять элементы поля в виде десятичных чисел. Представим теперь математическое описание процесса формирования элементов поля, представив их десятичными числами.

A[0]=1, A[255]=0;

При A[i-1]<128 A[i]=A[i-1]*2;

При A[i-1]=>128 A[i]=(A[i-1]*2)⊕285.

Заметим, однако, что суммирование элементов поля GF(2^8), представленных в виде десятичных чисел, по модулю 2 необходимо проводить после их перевода в двоичный вид. Листинг программы осуществляющий сложение чисел A1 и B1 в виде функции C++ приведён ниже.

```
float show_summ (float A1, float B1)
{int i, A2, B2, C1;
bool A[9], B[9], C[9];
for (i=0; i<=8; i++)
{A2=A1/2;
A[i]=A1-A2*2;
A1=A2;}
for (i=0; i<=8; i++)
{B2=B1/2;
B[i]=B1-B2*2;
B1=B2;}
for (i=0; i<=8; i++)
{C[i]=A[i]^B[i];}
C1=C[0]*1+C[1]*2+C[2]*4+C[3]*8+C[4]*16+C[5]*32+C[6]*64+C[7]*128+C
[8]*256;
return (C1);}
```

Программа переводит числа A1 и B1 в двоичный вид, представляя их в виде массивов и осуществляет сложение по модулю 2 элементов с одинаковым индексом. Результирующий массив преобразуется в десятичное число C1, являющееся суммой. Программа выполнена в виде функции на языке C++.

Операции вычитания и сложения элементов поля сводятся к побитовому сложению по модулю 2 их двоичных представлений. Проводить операции умножения и деления с элементами поля $GF(2^8)$ удобно, сформировав обратное поле. Основное поле, сформированное на основании приведенного выше математического описания, позволяет по значению степени примитивного элемента (первый и четвертый столбец таблицы 3.1.) найти значение элемента поля. Все элементы поля $GF(2^8)$ в десятичном виде представлены в таблице 3.2. с заполнением таблицы построчно. Обратное поле позволяет по заданному значению элемента поля найти степень примитивного элемента (числа 2).

Обратное поле - это поле логарифмов по основанию 2. Элементы обратного поля вычисляются следующим образом:

$$L[A[i]]=i \qquad (3.7.)$$

Приведём далее листинг программы, формирующей все элементы поля $GF(2^8)$ и обратного поля в десятичном представлении.

```
// Формирование элементов основного поля
A[0]=1;
A[255]=0;
for(i=1; i<=254; i++)
{if (A[i-1]<128)
A[i]=A[i-1]*2;
else
A[i]=show_summ(2*A[i-1], 299);}
// Формирование элементов обратного поля
for(i=0; i<=255; i++)
{n=A[i];
L[n]=i;}
```

Где «show_summ» - обращение к функции сложения элементов поля Галуа.

Все элементы обратного поля в десятичном виде представлены в таблице 3.3. с заполнением таблицы построчно.

Таблица 3.2. Элементы поля $GF(2^8)$ в десятичном виде.

1	2	4	8	16	32	64	128	29	58	116	232	205	135	19	38
76	152	45	90	180	117	234	201	143	3	6	12	24	48	96	192
157	39	78	156	37	74	148	53	106	212	181	119	238	193	159	35
70	140	5	10	20	40	80	160	93	186	105	210	185	111	222	161
95	190	97	194	153	47	94	188	101	202	137	15	30	60	120	240
253	231	211	187	107	214	177	127	254	225	223	163	91	182	113	226
217	175	67	134	17	34	68	136	13	26	52	104	208	189	103	206

129	31	62	124	248	237	199	147	59	118	236	197	151	51	102	204
133	23	46	92	184	109	218	169	79	158	33	66	132	21	42	84
168	77	154	41	82	164	85	170	73	146	57	114	228	213	183	115
230	209	191	99	198	145	63	126	252	229	215	179	123	246	241	255
227	219	171	75	150	49	98	196	149	55	110	220	165	87	174	65
130	25	50	100	200	141	7	14	28	56	112	224	221	167	83	166
81	162	89	178	121	242	249	239	195	155	43	86	172	69	138	9
18	36	72	144	61	122	244	245	247	243	251	235	203	139	11	22
44	88	176	125	250	233	207	131	27	54	108	216	173	71	142	0

Таблица 3.3. Элементы обратного поля в десятичном виде.

255	0	1	25	2	50	26	198	3	223	51	238	27	104	199	75
4	100	224	14	52	141	239	129	28	193	105	248	200	8	76	113
5	138	101	47	225	36	15	33	53	147	142	218	240	18	130	69
29	181	194	125	106	39	249	185	201	154	9	120	77	228	114	166
6	191	139	98	102	221	48	253	226	152	37	179	16	145	34	136
54	208	148	206	143	150	219	189	241	210	19	92	131	56	70	64
30	66	182	163	195	72	126	110	107	58	40	84	250	133	186	61
202	94	155	159	10	21	121	43	78	212	229	172	115	243	167	87
7	112	192	247	140	128	99	13	103	74	222	237	49	197	254	24
227	165	153	119	38	184	180	124	17	68	146	217	35	32	137	46
55	63	209	91	149	188	207	205	144	135	151	178	220	252	190	97
242	86	211	171	20	42	93	158	132	60	57	83	71	109	65	162
31	45	67	216	183	123	164	118	196	23	73	236	127	12	111	246
108	161	59	82	41	157	85	170	251	96	134	177	187	204	62	90
203	89	95	176	156	169	160	81	11	245	22	235	122	117	44	215
79	174	213	233	230	231	173	232	116	214	244	234	168	80	88	175

Имея сформированное основное и обратное поле, определим проведение операции умножения чисел Ma и Mb.

Если Ma=0 или Mb=0, то Ma*Mb=0;

Если (L[Ma]+L[Mb])<255, то Ma*Mb=A[(L[Ma]+L[Mb])];

Если (L[Ma]+L[Mb])=>255, то Ma*Mb=A[(L[Ma]+L[Mb]-255)];

Листинг программы осуществляющий умножение в виде функции C++ приведён ниже.

```
int show_proizv (int Ma, int Mb)
{i=L[Ma]+L[Mb];
{if (i>255)
i=i-255;}
P=A[i];
return (P);}
```

Определим проведение операции деления чисел Dm и Dl.

Если Dm =0, то Dm/Dl =0;

Если Dl =0 будет ошибка деления на 0;

Если (L[Dm]+L[Dl])>0, то Dm/Dl =A[(L[Dm]-L[Dl])];

Если (L[Dm]+L[Dl])<=0, то Dm/Dl =A[(L[Dm]-L[Dl]+255)];

Программная реализация функции деления аналогична программной реализации функции умножения.

3.2.3. Кодирование в систематической форме

Рассмотрим процедуру полиномиального кодирования в систематической форме кодов БЧХ.

Пусть $M=(m_0, m_1, m_2,\ldots,m_{k-1})$ - вектор информационного сообщения. Тогда вектор кодового слова при систематическом кодировании будет выглядеть следующим образом:

$$F=(m_0, m_1, m_2,\ldots, m_{k-1}, p_0, p_1, p_2,\ldots, p_{n-k-1}) . \qquad (3.8)$$

Где $p_0, p_1, p_2,\ldots, p_{n-k-1}$ – проверочные символы.

Вектор сообщения можно записать в полиномиальной форме следующим образом:

$$M(X)= m_{k-1}X^{k-1}\oplus\ldots\oplus m_2X^2\oplus m_1X\oplus m_0. \quad (3.9.)$$

Чтобы получить полином кодового сообщения необходимо осуществить сдвиг полинома сообщения в k крайне правых разряда кодового слова, а затем прибавить проверочные биты в n-k разрядов слева. Получить сдвинутый вправо полином сообщения мы можем путем умножения его на X^{n-k}:

$$X^{n-k}\,M(X)= m_{k-1}X^{n-1}\oplus\ldots\oplus m_1X^{n-k+1}\oplus m_0\,X^{n-k}. \quad (3.10.)$$

Полином кодового сообщения можно представить как:

$$F(X)= X^{n-k}\,M(X)\oplus P(X)\,. \quad (3.11.)$$

Проверочные символы мы получаем, используя генератор кода g(X).

$$P(X)= X^{n-k}\,M(X) \text{ по модулю } g(X). \quad (3.12.)$$

То есть P(X) получается как остаток от деления $X^{n-k}M(X)$ на g(X). Полиномиальный генератор имеет вид:

$$g(X)= g_{n-k}X^{n-k}\oplus\ldots\oplus g_2X^2\oplus g_1X\oplus g_0. \quad (3.13.)$$

и P(X) можно записать следующим образом:

$$P(X) = p_{n-k-1}X^{n-k-1}\oplus\ldots\oplus p_2X^2\oplus p_1X\oplus p_0. \quad (3.14.)$$

Запись кодового слова через все члены полинома имеет вид:

$$F(X)=X^{n-k}M(X)\oplus P(X)=m_{k-1}X^{n-1}\oplus\ldots\oplus m_1X^{n-k+1}\oplus m_0X^{n-k}\oplus p_{n-k-1}X^{n-k-1}\oplus\ldots\oplus p_2X^2\oplus p_1X\oplus p_0 \quad (3.15.)$$

Необходимо также отметить, что корни полиномиального генератора являются также корнями кодового слова им сгенерированного.

3.2.4. Вычисления остатка от деления

Как было рассмотрено в предыдущем пункте, с математической точки зрения в основе построения кодов БЧХ лежит операция полиномиального деления. А точнее вычисление остатка от деления одного полинома на другой.

Мы имеем полином-делимое $F(X)= X^{n-k}M(X)$ степени n-1 и полином-делитель g(X) степени r = n - k. Деление полиномов осуществляется с некоторыми отличиями по отношению к традиционной математике. На каждом шаге деления между соответствующими коэффициентами полиномов выполняется не вычитание, а операция сложения по модулю 2. На каждом шаге деления получается остаток, состоящий из r коэффициентов. Всего шагов деления s=1… n-r. Старший коэффициент полинома остатка всегда равен нулю, так что степень полинома остатка будет r - 1. На каждом шаге проводить вычислении старшего коэффициента нет необходимости. Очередной коэффициент частного удобно брать из остатка, вычисленного на предыдущем шаге. С учетом всего вышесказанного получаем математическую модель процедуры вычисления остатка:

$R_j^0=F_{n-r+j}$, j=0…r-1

$R_j^S=F_{n-r-s}\oplus g_0 R_{r-1}^{S-1}$, j=0

$R_j^S= R_{j-1}^{S-1}\oplus g_j R_{r-1}^{S-1}$, j=1…r-1

R_j^0 здесь - начальный остаток, а R_j^S остаток после шага s.

3.2.5. Программная реализация кодера

Рассмотрим программную реализацию кодера, кодирующего 5 информационных байт в кодовое слово из 9 символов.

Представив каждый информационный байт в виде десятичного числа от 0 до 255, получаем массив входных данных M[5]. Обозначим коэффициенты полиномиального генератора как массив g[5]. Обозначим массив остатков на каждом шаге деления как R[6][4]. Формируемый массив кодовых символов обозначим как F[9].

Вначале программы проводится формирование кодовых символов, сдвигая информационные на 4 позиции. Далее осуществляя полиномиальное деление как описано в п. 3.2.2. находим проверочные символы.

```
«int _tmain(int argc, _TCHAR* argv[])
{int M[5], F[9],  R[6][4], s, j, RA, RB, RC, RD;
int g[5]={116,231,216,30,1};
for(j=0; j<4; j++)
F[j+4]=M[j];
for (j=0; j<=r-1; j++)
{R[0][j]=F[n-r+j];}
for (s=1; s<=n-r; s++)
{R[s][0]=show_summ(F[n-r-s], show_proizv(g[0],R[s-1][r-1]));
for (j=1; j<=r-1; j++)
{R[s][j]=show_summ(R[s-1][j-1], show_proizv(g[j],R[s-1][r-1]));}}
for(j=0; j<4; j++)
F[j]=R[5][j];»
```

В программе «show_summ» и «show_proizv» - обращение к функциям сложения и умножения элементов поля Галуа приведенных в п. 3.1.2.

В случае двоичных кодов БЧХ строки программы в которых происходят вычисления R[s][0] и R[s][j] записываются следующим образом:

```
«R[s][0]= F[n-r-s]^(g[0]*R[s-1][r-1]);
R[s][j]=R[s-1][j-1] ^( g[j]*R[s-1][r-1]);»
```

Кодирование двоичных кодов БЧХ также может осуществляться рассмотренным в п. 3.1. методом с использованием порождающей матрицы.

3.3. Разработка программы декодирования кодов БЧХ

3.3.1. Реализационные основы декодера

После кодирования мы имеем кодовое слово F(x) длины n состоящее из k информационных и r проверочных символов. Если при передачи кодового слова возникла ошибка, то принятый полином кодового слова C(x) можно представить как сумму полиномов кодового слова F(x) и полинома модели ошибки E(x).

$$C(x) = F(x) + E(x), \qquad (3.16.)$$

Чтобы исправить ошибку при двоичном декодировании, необходимо знать лишь расположение ошибочного бита и затем инвертировать его значение. При недвоичном декодировании для исправления ошибки кроме расположения, необходимо знать и значение ошибки.

Синдром ошибки состоит из совокупности коэффициентов S_j , j =1…r, вычисляемых подстановкой в полином C(x) корней 2^1, 2^2,…, 2^r порождающего полинома g(x). Иными словами коэффициенты синдрома это значения принятого полинома в нулях кода.

Программная реализация процедуры вычисления коэффициентов S_j для случая кодирования кодом РС рассмотренного в п. 3.2. имеет следующий вид:

```
int j, i, r, SN, n, C[9], S[4];
n=9;
r=4;
for (j=0;j<r;j++)
{SN= C[0];
for (i=1;i<=n;i++)
{S[j]=show_summ(SN, show_proizv(C[i],A[(j+1)*i]);
SN=S[j];} }
```

В случае двоичных кодов БЧХ строка программы с вычислением S[j] записывается следующим образом:

```
S[j]= SN^(C[i]*A[(j+1)*i]);
```

Зная, что полином кодового слова *F(x)* обращается в ноль при подстановке в него корней 2^1, 2^2,…, 2^r, получаем:

$$S_j = C(2^j) = F(2^j) + E(2^j) = 0 + E(2^j) = E(2^j). \qquad (3.17.)$$

Как видно из (3.17.), синдром не зависит от кодового сообщения и определяется только значением ошибки.

Пусть τ - предполагаемое количество искаженных символов сообщения. Для описания ошибок введём понятие полинома локатора ошибок:

$$\Lambda(x) = 1 \oplus \sum_{i=1}^{\tau} \Lambda_i \cdot x^i = \prod_{i=1}^{\tau} \left(1 \oplus x^i \cdot 2^{u_i}\right) \qquad (3.18.)$$

Корнями полинома локатора являются элементы $2^{-u1},\ldots, 2^{-u\tau}$, тогда сами степени элементов - $u_1,\ldots, u_\tau$ являются локаторами местоположения ошибок в принятом сообщении.

Коэффициенты синдрома и полинома локаторов связаны ключевой системой уравнений декодирования:

$$S_j = \sum_{\substack{i=1 \\ j=\tau+1\ldots 2t}}^{\tau} \Lambda_i S_{j-i} \qquad (3.19.)$$

В развернутом виде эта система выглядит следующим образом:

$$\begin{cases} \Lambda_1 S_\tau \oplus \ldots \oplus \Lambda_\tau S_1 = S_{\tau+1} \\ \Lambda_1 S_{\tau+1} \oplus \ldots \oplus \Lambda_\tau S_2 = S_{\tau+2} \\ \vdots \\ \Lambda_1 S_{2t-1} \oplus \ldots \oplus \Lambda_\tau S_{2t-\tau} = S_{2t} \end{cases} \qquad (3.20.)$$

Известно несколько вариантов решения системы ключевых уравнений:

- Алгоритм Берлекэмпа – Месси [6, 7, 13]. Данный метод имеет наименьшее количество арифметический операций с элементами полей Галуа, и наиболее эффективен с точки зрения количества вычислений. Данный алгоритм использует программные инструкции «если» - «то» и не очень удобен для аппаратной реализации.

- Алгоритм Евклида. Этот метод решения ключевых уравнений в полиномиальной форме. Данный алгоритм имеет высокую регулярность структуры, в связи с чем широко используется для аппаратной реализации декодеров.

- Алгоритм Питерсона. Это метод находит коэффициенты многочлена локаторов ошибок прямым решением системы (3.20.) как системы линейных

уравнений. Данный метод используется довольно редко и чрезвычайно сложен при больших значений корректирующей способности кода.

Алгоритм декодирования кодов БЧХ в целом приведен на рисунке 3.2.

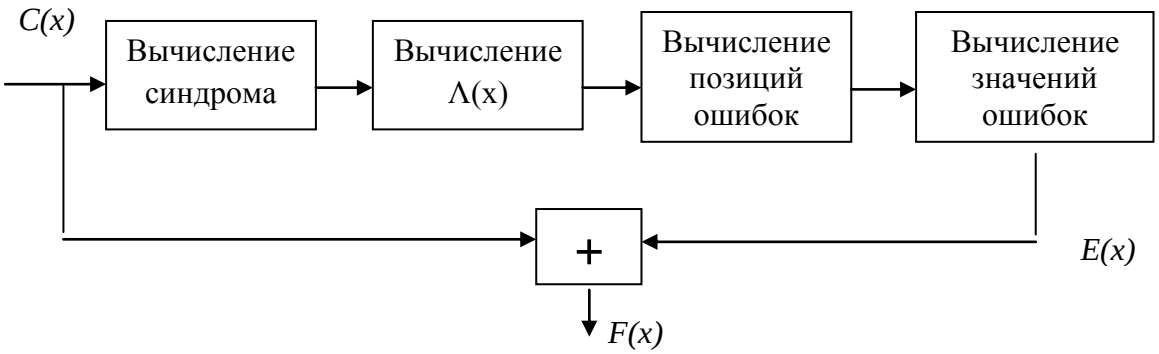


Рисунок 3.2. Алгоритм декодирования кодов БЧХ

В рассмотренном алгоритме этап вычисления значений ошибок в случае применения двоичных кодов является лишним. Следует также отметить, что декодирование двоичных кодов с небольшой исправляющей способностью может производиться, аналогично кодам Хемминга, с использованием проверочной матрицы.

В приложении № 2 приведена программа осуществляющая:

- кодирование двоичным кодом БЧХ (15,7), исправляющим 2 ошибочных бита, вводимой пользователем последовательности информационных бит;

- искажение кодового слова в соответствии с задаваемым пользователем количеством и номерами ошибочных бит;

- вычисление синдрома с использованием проверочной матрицы и проведение дальнейшего декодирования искаженного слова табличным методом.

3.3.2. Алгоритм Берлекэмпа–Месси

Теперь разберем подробно каждый этап алгоритма декодирования кодов БЧХ, за исключением рассмотренной выше процедуры вычисления синдрома.

Как было сказано выше, для разработки программного обеспечения целесообразно применять алгоритм Берлекэмпа-Месси [60] (смотри рисунок 3.3.).

Данный алгоритм позволяет найти решение не более чем за 2t шагов.

Алгоритм Берлекэмпа–Месси удобно рассматривать как итеративный процесс построения минимального линейного регистра сдвига с обратными связями, генерирующего последовательность коэффициентов синдрома (смотри рисунок 3.4.). Это аналогично построению многочлена Λ(x) наименьшей степени который генерирует коэффициенты полинома синдрома.

Вначале находят самый короткий многочлен Λ(x), генерирующий S_1. Далее проверяют, порождает ли этот регистр также S_2. Если порождает, то данный многочлен по-прежнему остается наилучшим решением, и нужно проверить, порождает ли он следующие символы синдромного многочлена. На каком-то шаге очередной символ уже не будет генерироваться. В этот момент нужно изменить многочлен Λ(x) таким образом, чтобы он:

– правильно предсказывал следующий символ,

– не менял предсказание предыдущих символов,

– увеличивал длину регистра на минимально возможную величину.

Вначале находят самый короткий многочлен Λ(x), генерирующий S_1. Далее проверяют, порождает ли этот регистр также S_2. Если порождает, то данный многочлен по-прежнему остается наилучшим решением, и нужно проверить, порождает ли он следующие символы синдромного многочлена. На

каком-то шаге очередной символ уже не будет генерироваться. В этот момент нужно изменить многочлен Λ(x) таким образом, чтобы он:

– правильно предсказывал следующий символ,

– не менял предсказание предыдущих символов,

– увеличивал длину регистра на минимально возможную величину.

Процесс вычисления продолжается до тех пор, пока не будут порождены первые 2t символов синдрома.

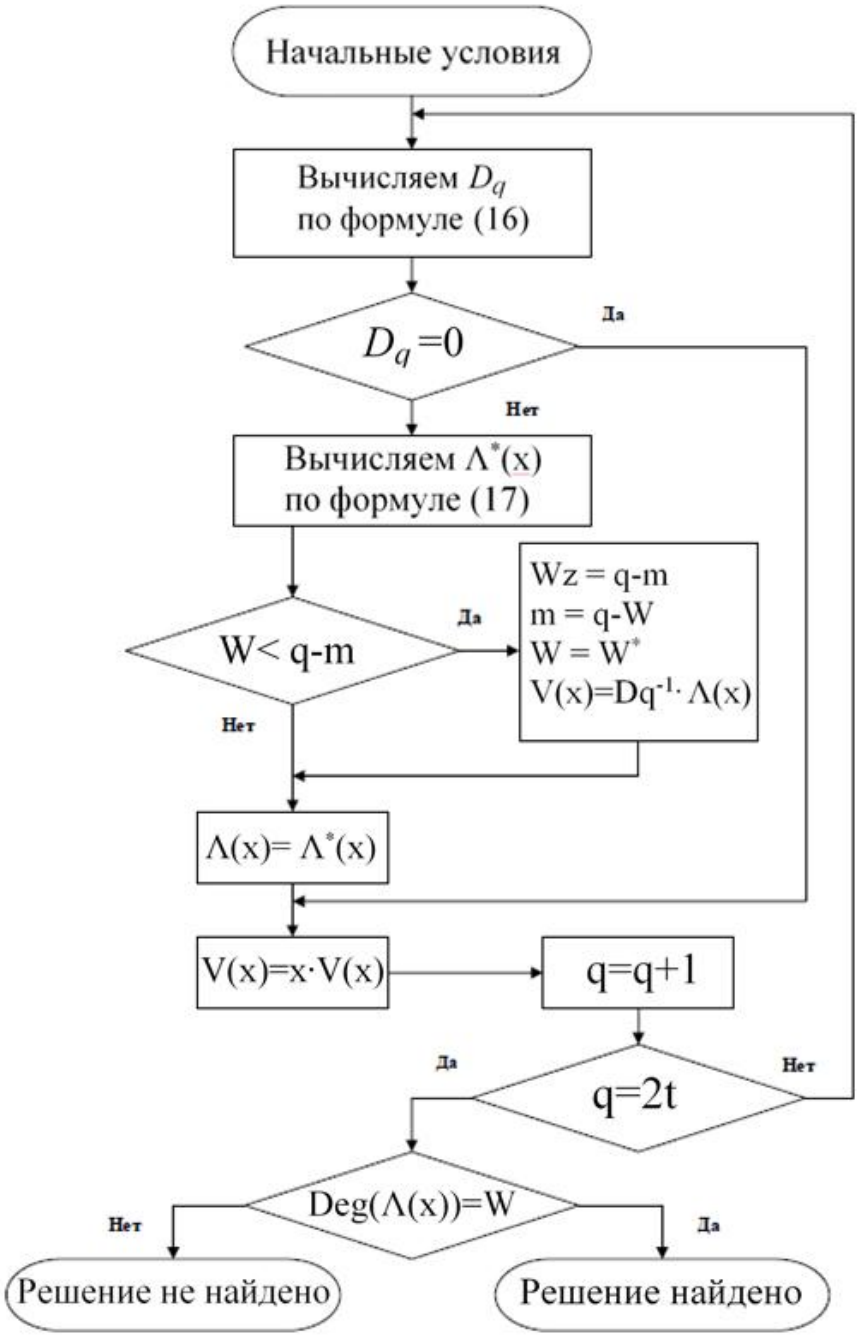


Рисунок 3.3. Алгоритм Берлекэмпа – Месси

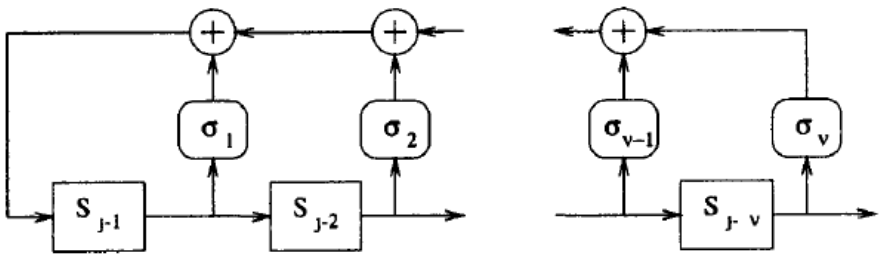


Рисунок 3.4. Линейный регистр сдвига

Алгоритм строится на основе итеративной процедуры. При каждой итерации должны сохраняться как многочлен связей Λ(x), так и добавка (полином модификаций) V(x). Для каждого нового члена S(x) предусматривается проверка правильности предсказания этого символа текущим многочленом связи. Если предсказание правильно и вычисляемая по формуле (3.21.) невязка D_q равна нулю, то многочлен связей не меняется, а добавка умножается на *x*.

$$D_q = \sum_{i=0}^{W} \Lambda_i \cdot S_{q-i+1}\ , \qquad (3.21.)$$

где W - количество членов в левой части уравнений (3.20.) (фактически этот параметр отражает количество искаженных байтов, предполагаемое вовремя итераций алгоритма).

Если предсказание неправильно (невязка $D_q \neq 0$), то изменяют текущий многочлен связей:

$$\Lambda^*(x) = \Lambda(x) \oplus D_q \cdot V(x). \qquad (3.22.)$$

После этого проверяют, увеличилась ли длина регистра. Если она не увеличилась, то текущую добавку оставляют. Если длина регистра возрастает, то лучшей добавкой считают предыдущий многочлен связей. Для недвоичных полей добавку нормируют, чтобы невязка стала равной 1. Далее при каждом исправлении эту нормализованную добавку умножают на значение текущей невязки.

Алгоритм стартует со следующими начальными условиями:

– номер итерации алгоритма q=0;

– рассчитываемый полином локаторов Λ(x)=1;

– полином модификации V(x)= x;

– номер шага, на котором полином локатора последний раз модифицировался m=-1;

– W=0.

На конечном этапе работы алгоритма необходимо проверить совпадает ли степень полинома локатора Deg(Λ(x)) с предполагаемым количеством

искаженных символов. Если возникло несовпадение, то делается вывод о невозможности исправления ошибок. В случае совпадения можно преступать к процедуре нахождения местоположения ошибок.

Далее рассмотрим программную реализацию алгоритма Берлекэмпа - Месси для случая кодирования кодом РС рассмотренного в п. 3.2.

В программной реализации введем следующие обозначения. Коэффициенты Λ(x) определим в качестве массива L. Коэффициенты $\Lambda^*(x)$ определим в качестве массива Lz. Полином модификаций $V(x)$ зададим как массив коэффициентов данного полинома Vx. Степень полинома локатора обозначим как deg.

Программная реализация алгоритма Берлекэмпа - Месси:

```
int _tmain(int argc, _TCHAR* argv[])
{int W, Wz, q, m, deg, i, Dg, Lz[3];
int Vx[3]={0,1,0};
int L[4]={1,0,0,0};
q=0;
W=0;
m=-1;
while (q!=4)
{Dq=0;
for(i=0; i<=W; i++)
Dq=show_summ(Dq,show_proizv(L[i],S[q-i]),0,0);
if (Dq!=0)
{for(i=0; i<=2; i++)
{Lz[i]=show_summ(L[i],show_proizv(Dq,Vx[i]),0,0);
if(W<q-m)
{Wz=q-m;
m=q-W;
W=Wz;
for(i=0; i<=2; i++)
```

```
{Vx[i]=show_chastn(L[i],Dq);
for(i=0; i<=2; i++)
L[i]=Lz[i];
for(i=2; i>=1; i--)
Vx[i]=Vx[i-1];
Vx[0]=0;}
for(i=0; i<=3; i++)
{if (L[i]>0)
deg =i;}
if (deg ==W)
printf("Полином найден");
else printf("Полином не нейден");}
```

3.3.3. Нахождение местоположения ошибок

Для нахождения местоположения ошибок необходимо приравнять нулю полином локатора $\Lambda(x)$ и найти корни. Простейшим путем нахождения корней является метод проб и ошибок, названный процедурой Ченя. Эта процедура состоит в последовательном вычислении всех корней $2^{-u_1},\ldots,2^{-u_\tau}$ уравнения. Делается это полным перебором элементов поля Галуа и подстановкой каждого из них в полином локаторов. По найденным корням, выделяются их степени $u_1,\ldots, u_\tau$, которые и являются номерами ошибочных символов. При этом учитывается, что *u=255-(- u).*

Далее необходимо учесть, что найденные номера не должны быть больше чем n-1. (В случае кода РС, описанного в п. 3.2. n-1=7.) Если же это условие не выполняется, то делается вывод о невозможности исправления ошибок.

Далее рассмотрим программную реализацию процедуры нахождения местоположения ошибок для случая кодирования кодом РС рассмотренного в п. 3.2, имеющего возможность исправлять не более 2 ошибок.

Обозначим через u[] массив номеров позиций ошибок, содержащий не более 2 элементов. Массив L[] содержит найденные по алгоритму Б-М в п. 3.3.2. значения коэффициентов полинома локатора. A[i], как и ранее, массив содержащий элементы основного поля GF(2^8). qs – значение полинома локатора при подстановке очередного элемента поля из массива A[i]. k- счетчик корней полинома локатора, значение которого в конечном счете должно быть равно числу ошибок.

Нахождения местоположения ошибок:

```
k=0;
for(i=0;i<=255;i++)
{qs=show_summ(show_proizv(L[2],show_proizv(A[i],A[i])),show_proizv(L[1],
A[i]),L[0]);
if (qs=0)
{k=k+1;
u[k-1]=255-i;
if (u[k-1]>7)
printf("Исправление ошибок невозможно")
printf(" \n Позиция ошибки\n%4.1d", u[k-1]);}}
printf(" \n Количество ошибок=%4.1d", k);
if (k= st-1)
else printf("Исправление ошибок невозможно")
```

3.3.4. Нахождение значений ошибок

Если номера ошибочных символов успешно вычислены, можно приступать к нахождению значений ошибок. Для вычисления значений ошибок будем использовать алгоритм Форни. Введём полином величин ошибок, связанный с полиномом синдрома и полиномом локаторов ошибок следующим образом:

$$\Omega(x) = (S(x)\cdot\Lambda(x)) mod\ (x^{\tau}). \qquad (3.23.)$$

Вычисление остатка по модулю x^r в данном случае можно заменить обнулением всех коэффициентов с индексами $i \geq r$.

Далее введём производную полинома локаторов – Λ’(x), которая вычисляется следующим образом:

$$\Lambda'(x) = \frac{d}{dx}\Lambda(x) = \Lambda_1 \oplus \Lambda_3 x^2 \oplus \ldots \oplus \begin{cases} \Lambda_\tau x^{\tau-1}, \tau \bmod 2 = 1 \\ \Lambda_{\tau-1} x^{\tau-2}, \tau \bmod 2 = 0 \end{cases} \quad (3.24.)$$

Величины ошибок $v_1, \ldots, v_\tau$,, вычисляются, как отношение значения полинома Ω(x) к значению производной Λ’(x):

$$v_l = \Omega\left(2^{-u_l}\right) \Big/ \Lambda'\left(2^{-u_l}\right) \quad (3.25.)$$

, где l=1…τ.

Тогда полином ошибок выглядит следующим образом:

$$E(x) = \sum_{l=1}^{\tau} v_l \cdot x^{u_l} \quad (3.26.)$$

Восстанавливаем полином неискаженного слова:

$$F(x) = C(x) + E(x). \quad (3.27.)$$

Декодирование завершено.

Далее рассмотрим программную реализацию процедуры нахождения местоположения ошибок для случая кодирования кодом РС рассмотренного в п. 3.2.

В этом случае r=4 и полином величин ошибок содержит коэффициенты с индексами от 0 до 3. Обозначим PO[4] – массив коэффициентов полинома величин ошибок.

Вычисление массива коэффициентов полинома величин ошибок:

PO[3]=show_summ(show_proizv(S[1],L[2]),show_proizv(S[2],L[1]),show_proizv(S[3],L[0]));

PO[2]=show_summ(show_proizv(S[0],L[2]),show_proizv(S[1],L[1]),show_proizv(S[2],L[0]));

PO[1]=show_summ(show_proizv(S[0],L[1]),show_proizv(S[1],L[0]),0);

PO[0]=show_proizv(S[0],L[0]);

Учитывая, что максимальная степень полинома локатора равна двум, из формулы (19) находим, что производная полинома локаторов $\Lambda'(x)= \Lambda_1$. В программе обозначим XP – значение аргумента, XI[4] – массив степеней аргументов полинома $\Omega(x)$, Ve[2] – массив значений ошибок.

```
int Ve[2], XI[4], XP;
for(j=0;j<=k-1;j++)
{XI[0]=1;
XP=A[255-u[j]];
for(i=1;i<=3;i++)
XI[i]=show_proizv(XI[i-1],XP);
Ve[j]=show_chastn(show_summ(show_proizv(PO[3],XI[3]),show_proizv(PO[2
],XI[2]),show_proizv(PO[1],XI[1]),PO[0]),L[1]);
//Исправляем ошибки:
for(j=0;j<=k-1;j++)
C[u[j]]=show_summ(C[u[j]],Ve[j],0,0);
```

В приложении № 3 приведена программа осуществляющая:

- кодирование недвоичным кодом РС (9,5) рассмотренным в п. 3.2. методом вводимой пользователем последовательности информационных символов (байт);

- искажение кодового слова в соответствии с задаваемым пользователем количеством, номерами и значением ошибочных символов;

- декодирование искаженного слова рассмотренным в п. 3.3. методом.

3.4. Коды Голея

Код Голея (23,12) имеет генерирующий полином $g(x)=x^{11}+x^{10}+x^6+x^5+x^4+x^2+x^1$. Кодирование кода Голея происходит аналогично кодированию кодов БЧХ, - рассмотренным выше методом полиномиального деления. Программная

реализация кодирования полностью соответствует программной реализации кодирования двоичного кода БЧХ.

Декодирование кода Голея как правило осуществляется с помощью декодера Меггита. Алгоритм работы декодера Меггита выглядит следующим образом.

Шаг 1. Делим принятое слово на генерирующий полином. Остатком от деления будет синдром. Если синдром нулевой, - ошибки отсутсвуют. Если вес синдрома не превышает 3, то все ошибки находятся в младших 11 битах принятого слова (с 0 по 10 биты). Позиции ошибок совпадают с позициями единичных бит в синдроме. Если вес синдрома превышает 3, то переходим к шагу 2.

22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Рисунок 3.5. Биты принятого сообщения

Шаг 2. Делаем предположение, что одна из ошибок находится в 17 бите принятого слова, а остальные располагаются в 11 младших битах. Находим новый синдром путем побитовому сложению по модулю 2 синдрома найденного на шаге 1 с остатком от деления x^{17} на генерирующий полином (который равен 11011001100). Если вес синдрома равен нулю, то мы имеем одну ошибку в 17 бите. Если вес синдрома 1 или 2, то в 17 бите находится ошибка, а все остальные ошибки располагаются в младших 11 битах принятого слова. Если вес синдрома превышает 2, то переходим к шагу 3.

Шаг 3. Делаем предположение, что одна из ошибок находится в 16 бите принятого слова, а остальные располагаются в 11 младших битах. Находим новый синдром путем побитовому сложению по модулю 2 синдрома найденного на шаге 1 с остатком от деления x^{16} на генерирующий полином (который равен 01101100110). Если вес синдрома равен нулю, то мы имеем одну ошибку в 16 бите. Если вес синдрома 1 или 2, то в 16 бите находится ошибка, а все остальные ошибки располагаются в младших 11 битах принятого слова. Если вес синдрома превышает 2, то переходим к шагу 4.

Шаг 4. Циклически сдвигаем кодовое слово в сторону младших разрядов и возвращаемся к шагу 1. Если на очередной итерации ошибки будут обнаружены, то кодовое слово необходимо будет циклически сдвинуть обратно на количество бит, соответствующее номеру повторной итерации.

В приложении № 4 приведена программа осуществляющая:

- кодирование кодом Голея полиномиальным методом вводимой пользователем последовательности информационных бит;

- искажение кодового слова в соответствии с задаваемым пользователем количеством и номерами ошибочных бит;

- декодирование искаженного слова декодером Меггита.

3.5. Описание программы каскадного кодирования и декодирования

Опишем программу адаптивного кодирования-декодирования для нерегулярных по длительности сообщений приведенную в приложении № 5. Программа состоит из трех блоков: программный кодер, блок введения ошибок, программный декодер. Алгоритм процедуры кодирования – декодирования показан на рисунке 3.6.

Программное обеспечение осуществляет:

- кодирование нерегулярных по длительности сообщений кодом РС, каскадом из внешнего кода РС и внутреннего кода Хэмминга, каскадом из внешнего кода РС и внутреннего кода Голея;

- включение и отключение перемежения символов кода РС;

- искажение кодовой введением случайных ошибок (будет подробно рассмотрено в п. 4.1.);

- декодирование искаженной кодовой последовательности.

В начале программы задаются:

- число информационных байт (переменная kolinf);

- метод кодирования (переменная mkod);

- флаг перемежения (переменная PER).

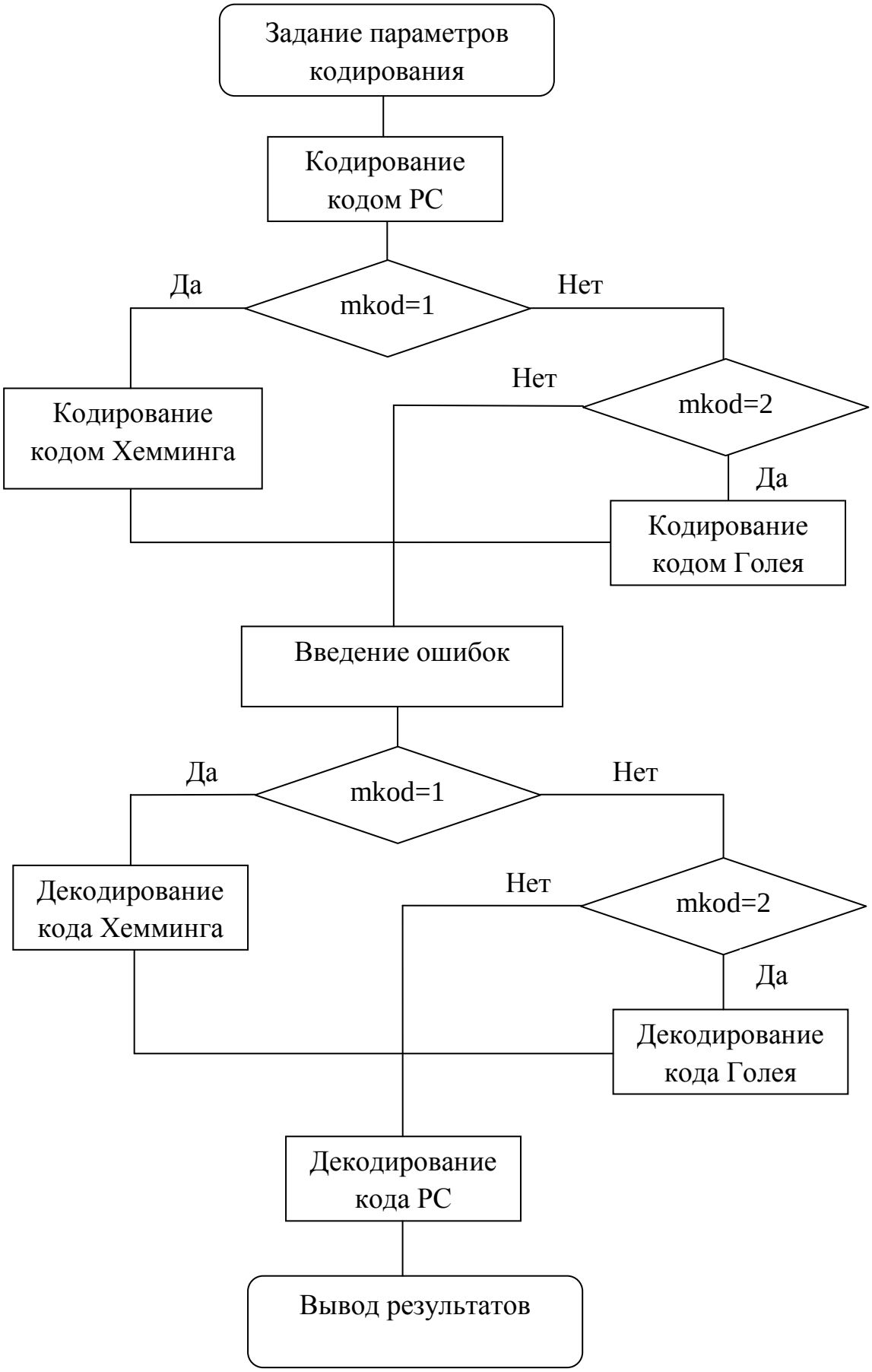


Рисунок 3.6. Алгоритм программы каскадного кодирования-декодирования

Далее проводится кодирование информации в соответствии с заданными параметрами и создается массив кодовых бит BV[].

В массив кодовых бит вносятся ошибки в соответствии с задаваемым пользователем количеством (параметр kerg) и номерами ошибочных бит (параметра nerg). Внесение ошибочных бит с заданной вероятностью будет рассмотрено в п. 4.

По окончании декодирования на экран выводится декодированная информационная последовательность.

Если в приведенном в приложении № 5 программном продукте убрать блок задания ошибок и переставить местами кодер и декодер то получится программа-ретранслятор принятого кодового сообщения.

Дальность действия современных радиомодемов как правило не превышает 10-12 км, что делает программный ретранслятор востребованным в ЛСТ.

Выводы по главе 3

Разработано ПО кодирования кодом Хемминга с использованием порождающей матрицы для генерации кодовых слов. Разработано ПО декодирования последовательности кодовых слов, закодированных кодом Хемминга табличным методом.

Разработано ПО кодирования последовательности информационных бит кодом Голея с использованием для генерации кодовых слов метода полиномиального деления.

Разработано ПО декодирования последовательности кодовых слов закодированных кодом Голея при помощи реализации декодера Меггита.

Разработано ПО кодирования последовательности информационных бит двоичным кодом БЧХ с использованием для генерации кодовых слов метода полиномиального деления.

Разработано ПО декодирования последовательности кодовых слов закодированных кодом БЧХ с использованием проверочной матрицы для нахождения синдрома и проведением дальнейшего декодирования искаженного слова табличным методом.

Разработано ПО, осуществляющее операции сложения, умножения и деления элементов поля Галуа $GF(2^8)$.

Разработано ПО, осуществляющее методом полиномиального деления кодирование кодом РС.

Разработано ПО осуществляющее декодирование последовательности кодовых слов закодированных кодом РС. В разработанном ПО для вычисления полинома локатора ошибок применяется алгоритм Берлекэмпа-Месси, для вычисления позиции ошибки – метод Ченя, для вычисления значения ошибки – алгоритм Форни.

Разработано ПО каскадного кодирования и декодирования.

ГЛАВА 4. ЭКСПЕРИМЕНТАЛЬНЫЕ ИССЛЕДОВАНИЯ КОДЕРА-ДЕКОДЕРА

Целью экспериментов является исследование эффективности работы программного кодера-декодера при наличии в канале независимых ошибок и пакетов ошибок, а также проверка теоретически полученных в главе 2 результатов.

В п. 4.1. исследования будут проводиться при программной эмуляции появления независимых ошибок и пакетов ошибок.

В п. 4.3. будет описано проведение экспериментальных исследований эффективности работы программного кодера-декодера при его работе в составе ЛСТ осуществляющей передачу данных между ДЦ и КО с использованием радиомодемов.

4.1. Исследования кодера-декодера программными методами

4.1.1. Разработка инструментария для исследования работы программных кодеров-декодеров при наличии независимых ошибок в принятом сообщении

Для экспериментальных исследований программных кодеров-декодеров разработано ПО осуществляющее генерацию информационных данных и массива ошибок.

Ниже приведено ПО осуществляющее генерацию массива anf[] чисел значением от 0 до 255, которые удобно преобразовать в байты двоичных данных и использовать в качестве информационных данных.

```
void Random0to255(int *mas,int size)
{for (int i=0;i<size;i++)
mas[i] = rand()%256;}
int _tmain(int argc, _TCHAR* argv[])
```

```
{int anf[100];
Random0to255(anf,100);
printf("\nInformation: ");
for(int j=0;j<100;j++) printf("%d ", anf [j]);}»
```

Далее приведено ПО осуществляющее генерацию массива двоичных чисел err[] с заданной вероятностью появления единичного бита.

```
«int RandomBinary(int *mas,int size,int vd)
{int count=0;
for (int i=0;i<size;i++)
{mas[i] = 0;
int st = pow(10.0,vd);
if (rand()%(st+1) == st)
{count++;
mas[i] = 1;}}
return count;}
int _tmain(int argc, _TCHAR* argv[])
{printf("Input vd: ");
int vd;
scanf("%d",&vd);
int err[1000];
int aer=0;
int ker = RandomBinary(err,1000,vd);
aer += ker;
printf("\nQuantity Errors: %d",ker);
printf("\nNumbers: ");
for (int j=0;j<1000;j++)
if (err[j]==1) printf("%d ",j);}
```

Вероятность появления единичного бита задается параметром vd, таким что вероятность появления единичного бита равна 10^{-vd}.

Поэлементное суммирование по модулю 2 сгенерированного массива err[] с массивом информационных бит эквивалентно появлению в последнем независимых ошибок с заданной вероятностью.

Проведя слияние программ генерации информационных данных и ошибочные бит с программами кодирования-декодирования, получаем инструмент для проведения экспериментальных исследований эффективности кодов при наличии независимых ошибок в принятом сообщении.

4.1.2. Исследования помехоустойчивости разработанного каскадного кодера-декодера при наличии в канале независимых ошибок

Результаты исследований эффективности работы кодов: РС, РС+Х, РС+Г при наличии в канале независимых ошибок отражены в таблице 4.1. и на рисунке 4.1. На рисунке специальными символами обозначены практически полученные точки.

В таблице 4.1. использованы следующие обозначения: P_B – вероятность появления ошибочного бита при передаче закодированного сообщения, P_{PC} – вероятность появления ошибки при декодировании 1 слова РС рассчитанная ранее (в том числе и при каскадном кодировании), $P_{PCП}$ – практически полученная вероятность появления ошибки в 1 слове РС, P_{BD} – вероятность появления ошибочного бита в декодированном информационном сообщении, Vk – увеличение отношения Eb/No в дБ связанное с добавлением избыточных проверочных бит при кодировании, Vo – увеличение отношения Eb/No (где Eb - энергии бита, а No - спектральной плотности мощности шума) необходимое для снижения P_B до уровня P_{BD} без применения помехоустойчивого кодирования, VW – от применения кодирования (разница Vo и Vk). Значения Vk и Vo найдены по графику зависимости Eb/No от вероятности появления ошибочного бита в канале связи для модуляции типа 16QAM изображенным на рисунке 4.2. Для всех видов модуляции изображенных на графике значения Vk и Vo отличаются незначительно.

Таблица 4.1. – Сравнение эффективности кодов при наличии независимых ошибок в канале связи

№	P_B	Метод кодир.	$P_{РС}$	$P_{РСП}$	P_{BD}	Vk, дБ	Vo, дБ	VW, дБ
1.	10^{-2}	РС	0,02	0,022	$1{,}4\cdot10^{-3}$	2,5	2	-0,5
2.	10^{-3}	РС	$3{,}8\cdot10^{-5}$	$4{,}5\cdot10^{-5}$	$2{,}9\cdot10^{-6}$	2,5	3,2	0,7
3.	10^{-2}	РС+Х	$3{,}1\cdot10^{-6}$	$4{,}5\cdot10^{-6}$	$4{,}4\cdot10^{-7}$	5	6,7	1,7
4.	10^{-2}	РС+Г	$7{,}4\cdot10^{-8}$	$7{,}5\cdot10^{-8}$	$7{,}2\cdot10^{-9}$	5,4	7,9	2,5

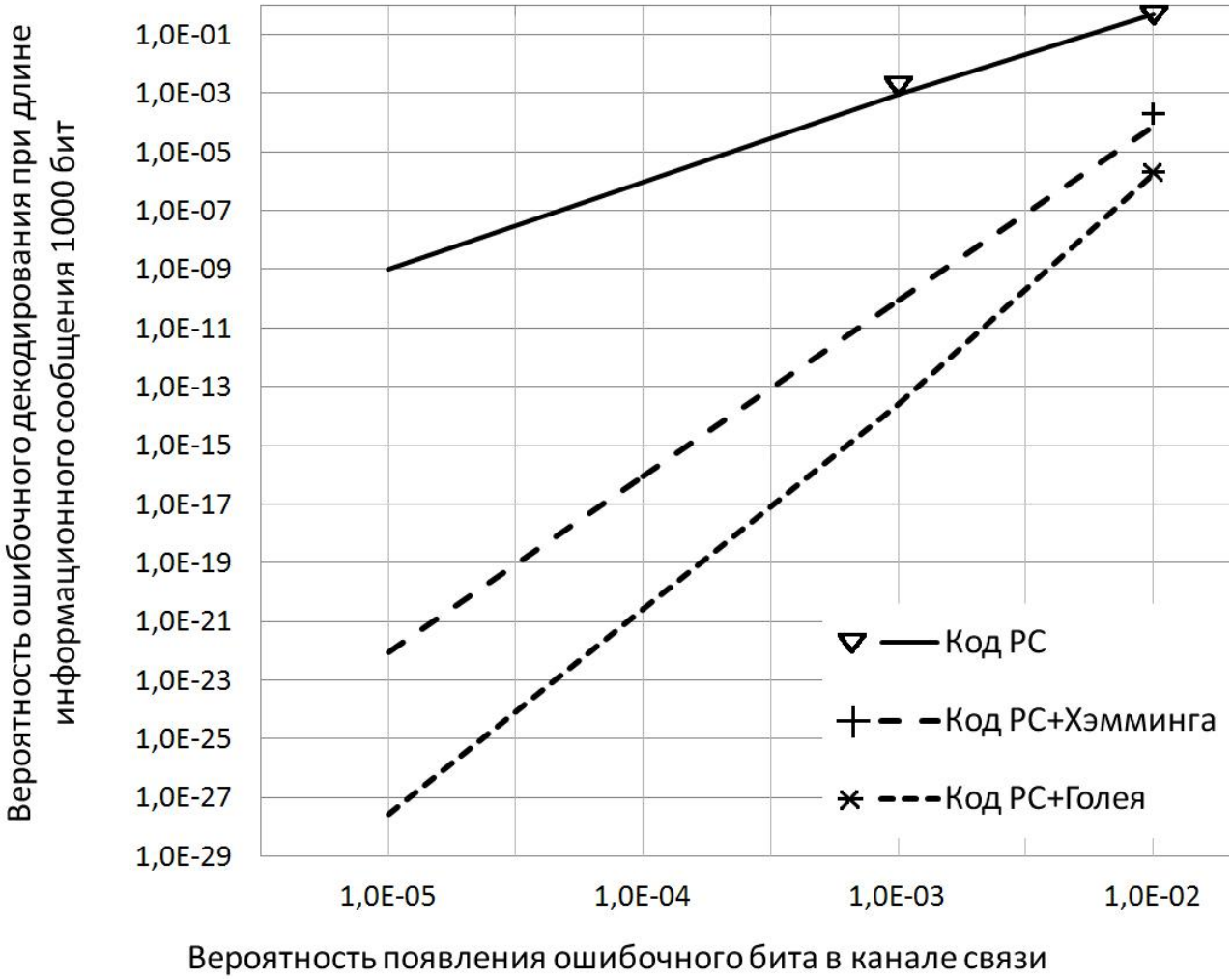


Рисунок 4.1. – Зависимость вероятности ошибочного декодирования сообщения от P_B при длине информационного сообщения 1000 бит

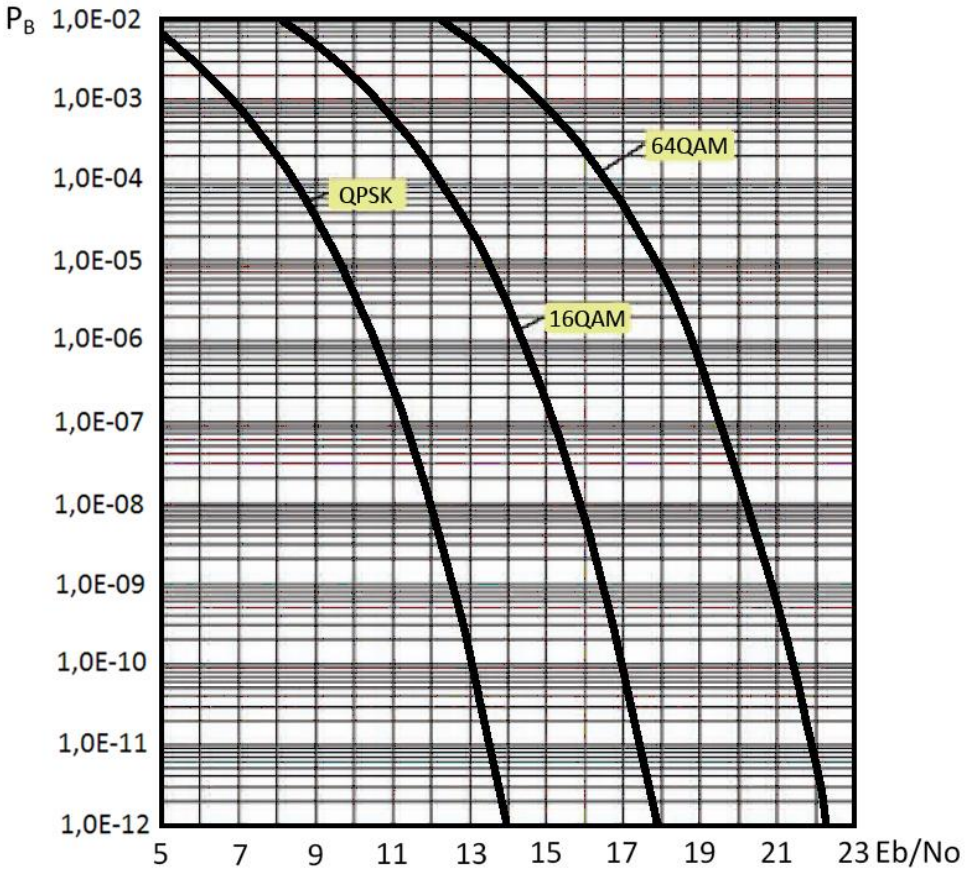


Рисунок 4.2. – Зависимость Eb/N_0 от вероятности появления ошибочного бита в канале связи для разных видов модуляции

Проанализировав полученные результаты можно сделать следующие выводы. Применение кодирования с использованием только кодов РС целесообразно если P_B превышает 10^{-3}. Каскадное соединение кода РС с кодом Хемминга при вероятности появления ошибочного бита в канале равной 10^{-2} даёт энергетический выигрыш на 1,7 дБ по сравнению с использованием одного кода РС. Наибольший энергетический выигрыш получен при использовании каскадного соединения кода РС с кодом Голея.

4.1.3. Разработка инструментария для исследования работы программного кодера-декодера при наличии пакетов ошибок

Для проведения исследования эффективности кодов при наличии в канале пакетов ошибок, разработано ПО, осуществляющее генерацию пакетов ошибок

разной длинны. Положение пакета ошибок сдвигается на 1 бит каждый такт исследований от начала до конца кодовой последовательности.

В начале программы пользователь задает длину пакета ошибок lpak. Обозначим число бит в сообщении – lmass. Тогда количество возможных вариантов расположения пакета ошибок внутри сообщения равно:

$$kitp = lmass - lpak + 1 \qquad (4.1.)$$

Далее начинаем цикл по переменной kitp. В цикле мы инвертируем lpak бинарных элементов массива кодового сообщения, начиная с элемента под номером kitp, что эмулирует появление пакета ошибок в канале передачи данных во всех возможных позициях (смотри рисунок 4.3.).

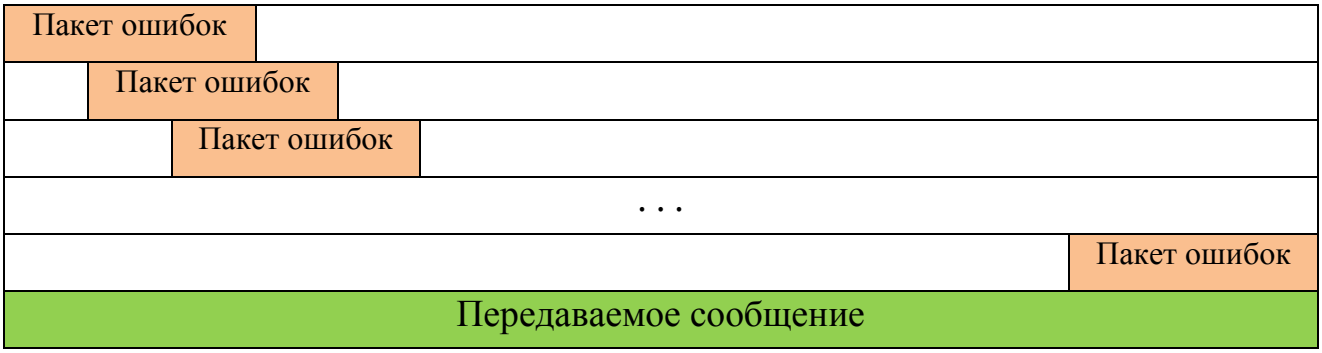


Рисунок 4.3. Ввод пакетов ошибок в передаваемое сообщение

Ниже приведен текст программы:

```
int lpak, lmass, kitp, i, j;
// ввод lpak
printf("Input lpak: ");
scanf("%d",& lpak);
kitp= lmass–lpak+1;
for (i=0; i<kitp; i++)
{ for (j=0; j<lpak; j++)
BV[i+j]=BV[i+j]^1;}
```

Проведя слияние программ генерации информационных данных и генерации пакетов ошибок с программами кодирования-декодирования, получаем инструмент для проведения экспериментальных исследований эффективности кодов при наличии пакетов ошибок в принятом сообщении.

4.1.4. Исследования помехоустойчивости разработанного каскадного кодера-декодера при наличии в канале пакетов ошибок

Результаты исследований эффективности кода РС при наличии в канале передачи данных пакетов ошибок приведены в таблице 4.2. Исследования проводились при длине информационного сообщения – 320 бит, преобразованных в процессе кодирования в 576 кодовых бит.

В указанной таблице использованы следующие обозначения: M – число кодовых бит, L – длинна пакета ошибок, N – число циклов, P_M – вероятность появления ошибочного бита в декодированном информационном сообщении, вычисляемая по формуле $P_M = K_M/N$.

Таблица 4.2. Исследование эффективности кода РС при наличии пакетов ошибок

№	M	L	N	K_M	P_M
1.	576	Менее 10	-		0
2.	576	10	567	40	0,071
3.	576	11	566	92	0,163
4.	576	12	565	125	0,221
5.	576	13	564	160	0,284
6.	576	14	563	200	0,355
7.	576	15	562	256	0,456
8.	576	16	561	292	0,520
9.	576	17	560	327	0,584
10.	576	18	559	327	0,585
11.	576	19	558	334	0,599
12.	576	20	557	350	0,628
13.	576	21	556	364	0,655
14.	576	22	555	358	0,645
15.	576	23	554	370	0,668
16.	576	24	553	376	0,680
17.	576	25	552	383	0,694
18.	576	26	551	394	0,715

19.	576	27	550	396	0,720
20.	576	28	549	406	0,740
21.	576	29	548	412	0,752
22.	576	30	547	419	0,766
23.	576	31	546	426	0,780
24.	576	32	545	433	0,794
25.	576	33	544	440	0,809
26.	576	34	543	446	0,821
27.	576	35	542	452	0,834
28.	576	36	541	458	0,847
29.	576	37	540	464	0,859
30.	576	38	539	470	0,872
31.	576	39	538	476	0,885
32.	576	40	537	482	0,898
33.	576	41	536	488	0,910
34.	576	42	535	491	0,918
35.	576	43	534	497	0,931
36.	576	44	533	500	0,938
37.	576	45	532	506	0,951
38.	576	46	531	510	0,960
39.	576	47	530	516	0,974
40.	576	48	529	522	0,987
41.	576	Более 48	-	-	1

В таблице 4.3. приведены результаты исследования работы каскадного кода РС+Х при наличии пакетов ошибок разной длины. В результате каскадного кодирования из 320 информационных бит были получены 1008 кодовых бита.

Таблица 4.3. Исследование эффективности каскадного кода РС+Х при наличии пакетов ошибок

№	M	L	N	$К_М$	$Р_М$
1.	1008	Менее 18	-	0	0
2.	1008	18	991	42	0,042

3.	1008	19	990	92	0,093
4.	1008	20	989	126	0,127
5.	1008	21	988	160	0,162
6.	1008	22	987	204	0,207
7.	1008	23	986	250	0,254
8.	1008	24	985	298	0,303
9.	1008	25	984	354	0,360
10.	1008	26	983	386	0,393
11.	1008	27	982	416	0,424
12.	1008	28	981	468	0,477
13.	1008	29	980	480	0,490
14.	1008	30	979	542	0,554
15.	1008	31	978	563	0,576
16.	1008	32	977	570	0,583
17.	1008	33	976	577	0,591
18.	1008	34	975	586	0,601
19.	1008	35	974	599	0,615
20.	1008	36	973	611	0,628
21.	1008	37	972	610	0,628
22.	1008	38	971	621	0,640
23.	1008	39	970	625	0,644
24.	1008	40	969	625	0,645
25.	1008	41	968	637	0,658
26.	1008	42	967	643	0,665
27.	1008	43	966	647	0,670
28.	1008	44	965	654	0,678
29.	1008	45	964	664	0,689
30.	1008	46	963	672	0,698
31.	1008	47	962	682	0,709
32.	1008	48	961	686	0,714
33.	1008	49	960	705	0,734
34.	1008	50	959	700	0,730
35.	1008	51	958	710	0,741

36.	1008	52	957	716	0,748
37.	1008	53	956	723	0,756
38.	1008	54	955	730	0,764
39.	1008	55	954	737	0,773
40.	1008	56	953	744	0,781
41.	1008	57	952	751	0,789
42.	1008	58	951	758	0,797
43.	1008	59	950	764	0,804
44.	1008	60	949	770	0,811
45.	1008	61	948	776	0,819
46.	1008	62	947	782	0,826
47.	1008	63	946	788	0,833
48.	1008	64	945	794	0,840
49.	1008	65	944	800	0,847
50.	1008	66	943	806	0,855
51.	1008	67	942	812	0,862
52.	1008	68	941	818	0,869
53.	1008	69	940	824	0,877
54.	1008	70	939	830	0,884
55.	1008	71	938	836	0,891
56.	1008	72	937	842	0,899
57.	1008	73	936	848	0,906
58.	1008	74	935	851	0,910
59.	1008	75	934	857	0,918
60.	1008	76	933	860	0,922
61.	1008	77	932	866	0,929
62.	1008	78	931	872	0,937
63.	1008	79	930	878	0,944
64.	1008	80	929	884	0,952
65.	1008	81	928	888	0,957
66.	1008	82	927	894	0,964
67.	1008	83	926	898	0,970
68.	1008	84	925	904	0,977

69.	1008	85	924	910	0,985
70.	1008	86	923	916	0,992
71.	1008	Более 86	-	-	1

В таблице 4.4. приведены результаты исследования работы каскадного кода РС+Г при наличии пакетов ошибок разной длины. В результате каскадного кодирования из 320 информационных бит были получены 1104 кодовых бита.

Таблица 4.4. Исследование эффективности каскадного кода РС+Г при наличии пакетов ошибок

№	M	L	N	K_M	$P_{ош\ сообщ}$
1.	1104	Менее 8	-	0	0
2.	1104	8	1097	24	0,022
3.	1104	9	1096	31	0,028
4.	1104	10	1095	48	0,044
5.	1104	11	1094	80	0,073
6.	1104	12	1093	120	0,110
7.	1104	13	1092	158	0,145
8.	1104	14	1091	169	0,155
9.	1104	15	1090	206	0,189
10.	1104	16	1089	239	0,219
11.	1104	17	1088	288	0,265
12.	1104	18	1087	315	0,290
13.	1104	19	1086	307	0,283
14.	1104	20	1085	331	0,305
15.	1104	21	1084	336	0,310
16.	1104	22	1083	397	0,367
17.	1104	23	1082	410	0,379
18.	1104	24	1081	440	0,407
19.	1104	25	1080	480	0,444
20.	1104	26	1079	514	0,476
21.	1104	27	1078	549	0,509
22.	1104	28	1077	562	0,522
23.	1104	29	1076	592	0,550

24.	1104	30	1075	636	0,592
25.	1104	31	1074	646	0,601
26.	1104	32	1073	668	0,623
27.	1104	33	1072	680	0,634
28.	1104	34	1071	684	0,639
29.	1104	35	1070	691	0,646
30.	1104	36	1069	701	0,656
31.	1104	37	1068	704	0,659
32.	1104	38	1067	714	0,669
33.	1104	39	1066	728	0,683
34.	1104	40	1065	727	0,683
35.	1104	41	1064	736	0,692
36.	1104	42	1063	756	0,711
37.	1104	43	1062	772	0,727
38.	1104	44	1061	766	0,722
39.	1104	45	1060	787	0,742
40.	1104	46	1059	775	0,732
41.	1104	47	1058	804	0,760
42.	1104	48	1057	795	0,752
43.	1104	49	1056	808	0,765
44.	1104	50	1055	803	0,761
45.	1104	51	1054	833	0,790
46.	1104	52	1053	817	0,776
47.	1104	53	1052	838	0,797
48.	1104	54	1051	830	0,790
49.	1104	55	1050	836	0,796
50.	1104	56	1049	848	0,808
51.	1104	57	1048	849	0,810
52.	1104	58	1047	856	0,818
53.	1104	59	1046	870	0,832
54.	1104	60	1045	868	0,831
55.	1104	61	1044	870	0,833
56.	1104	62	1043	877	0,841

57.	1104	63	1042	883	0,847
58.	1104	64	1041	887	0,852
59.	1104	65	1040	894	0,860
60.	1104	66	1039	899	0,865
61.	1104	67	1038	905	0,872
62.	1104	68	1037	911	0,878
63.	1104	69	1036	917	0,885
64.	1104	70	1035	923	0,892
65.	1104	71	1034	929	0,898
66.	1104	72	1033	935	0,905
67.	1104	73	1032	941	0,912
68.	1104	74	1031	947	0,919
69.	1104	75	1030	953	0,925
70.	1104	76	1029	959	0,932
71.	1104	77	1028	965	0,939
72.	1104	78	1027	971	0,945
73.	1104	79	1026	977	0,952
74.	1104	80	1025	976	0,952
75.	1104	81	1024	975	0,952
76.	1104	82	1023	981	0,959
77.	1104	83	1022	980	0,959
78.	1104	84	1021	986	0,966
79.	1104	85	1020	985	0,966
80.	1104	86	1019	984	0,966
81.	1104	87	1018	990	0,972
82.	1104	88	1017	996	0,979
83.	1104	89	1016	1002	0,986
84.	1104	90	1015	1001	0,986
85.	1104	91	1014	1007	0,993
86.	1104	92	1013	1012	0,999
87.	1104	Более 92			1

Исследована эффективность работы кодера-декодера РС при наличии в канале пакетов ошибок. При размере пакета ошибок менее 10 бит, декодер

всегда проводит правильное декодирование. Правильное декодирование возможно с определенной вероятностью при размере пакета ошибок от 10 до 48 бит.

Исследована эффективность работы кодера-декодера РС+Х при наличии в канале пакетов ошибок. При размере пакета ошибок менее 18 бит, декодер всегда проводит правильное декодирование. Правильное декодирование возможно с определенной вероятностью при размере пакета ошибок от 18 до 86 бит.

Исследована эффективность работы кодера-декодера РС+Г при наличии в канале пакетов ошибок. При размере пакета ошибок менее 8 бит, декодер всегда проводит правильное декодирование. Правильное декодирование возможно с определенной вероятностью при размере пакета ошибок от 8 до 92 бит.

На рисунке 4.4. показаны сравнительные характеристики для разных методов кодирования-декодирования, возможных в разработанном ПО каскадного кодирования-декодирования.

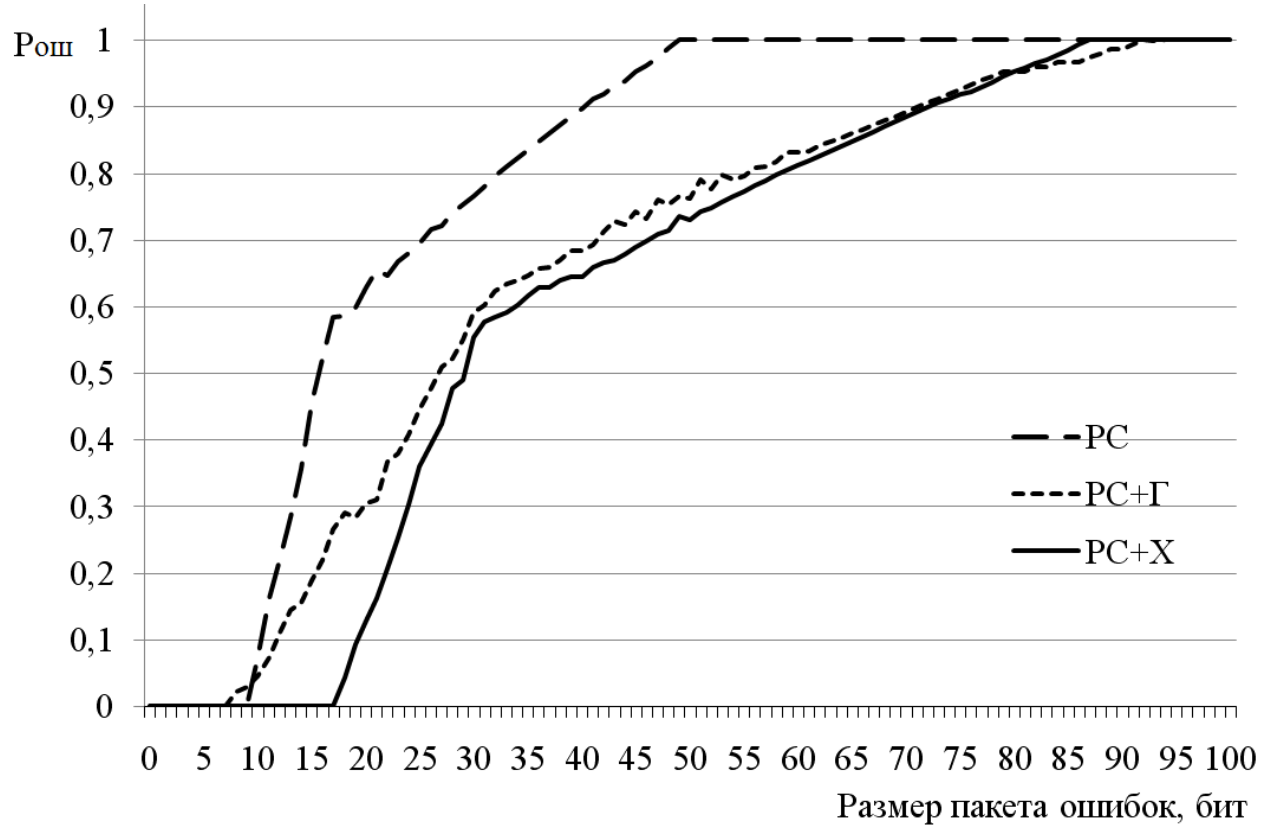


Рисунок 4.4. – Зависимость вероятности приема ошибочного сообщения от величины пакета ошибок

4.2. Построение универсального программно-аппаратного комплекса для ЛСТ на базе разработанного каскадного кодера-декодера

4.2.1. Рекомендации по применению кодера-декодера в различных ЛСТ

Разработанный каскадный кодер-декодер применим для всех вариантов построения ЛСТ рассмотренных в п. 1.3.

В любой ЛСТ программный кодер взаимодействует с одной стороны с ПО осуществляющим сбор данных, - с другой стороны с ПО обеспечивающим взаимодействие с устройством передачи данных. Программный декодер взаимодействует с одной стороны с ПО обеспечивающим взаимодействие с устройством приема данных, - с другой стороны с ПО обеспечивающим отображение и/или хранение декодированных данных.

В настоящее время существует большое количества совместимых с ЭВМ устройств ввода данных и микроконтроллеров. Эти устройства имеют разнообразные интерфейсы связи с ЭВМ и конструктивное исполнение. Также имеются готовые программно-аппаратные комплексы обеспечивающие сбор, хранение и отображение данных. При помощи данных устройств информация от КИП и датчиков может быть принята и обработана ЭВМ. Из этих, поступающих в режиме реального времени или записанных в файл данных, ПО необходимо создать информационный массив который будет закодирован программным кодером. Специальный программный модуль формирует из закодированных данных информационные пакеты и взаимодействует с ПО модема осуществляющего передачу данных на преобразуется в пакеты интерфейсом устройства передачи данных (например, радиомодемом)

После принятия модемом пакетов данных, специальный программный модуль взаимодействует с ПО модема и осуществляет извлечение данных из пакетов и преобразование их в массив который будет обрабатываться программным декодером. После декодирования информационные данные

поступают в программный модуль осуществляющий отображение и/или хранение информационных данных.

В зависимости от вида ЛСТ необходимо разработать разнообразные программные модули подключаемые к программам кодирования и декодирования, а также применять различные варианта алгоритмов обмена данными между ДЦ и КО.

4.2.2. Алгоритмы работы универсального программно-аппаратного комплекса

Проведена разработка и построение УПАК для ЛСТ, осуществляющего передачу данных между ЭВМ КО и ЭВМ ДЦ с использованием радиомодемов осуществляющих обмен данными с ЭВМ через интерфейс RS-232. Комплекс построен для случая магистральной топологической структуры ЛСТ, и использует временное разделение каналов. Информационные биты берутся путем преобразования файла с данными в массив данных. В комплексе применяется разработанный адаптивный каскадный кодер-декодер.

Упрощенный алгоритм обмена данными КО с ДЦ приведен на рисунке 4.5.

Работа УПАК происходит следующим образом.

ПО, установленное на ЭВМ ДЦ формирует запрос передачи данных. В случае первичного запроса на передачу данных, в запросе не содержаться рекомендации по применению вида кодирования информационных бит. Повторные запросы (переспросы) содержат рекомендации по применению кодирования, - вначале каскадный код РС+Х, а затем каскадный код РС+Г (адаптивное кодирование). Биты запроса передачи данных передаются на ЭВМ КО.

ПО, установленное на ЭВМ КО декодировав биты запроса передачи данных, преобразует файл с информацией в массив данных, сохраняя в массиве информацию об имени и расширении файла (для его последующего восстановления). Полученный массив информационных бит кодируется. Вид

кодирования определяется количеством ошибок исправленных при декодировании бит служебной информации, и рекомендациями, содержащимися в запросе передачи данных (из них выбирается наиболее сложный вариант кодирования). Закодированные биты служебной информации, предваряя массив закодированных данных, передаются модемом.

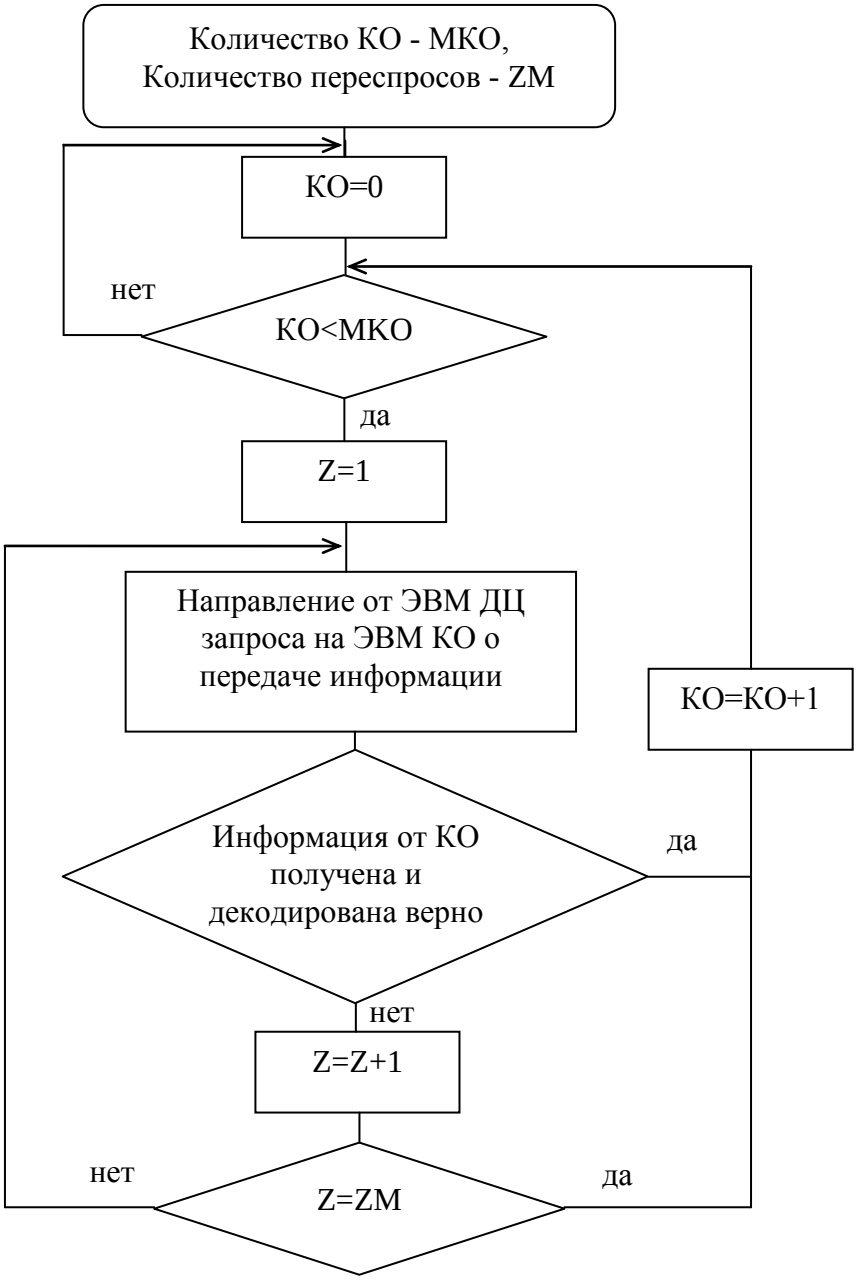


Рисунок 4.5. – Алгоритм обмена данными ДЦ и КО

ПО, установленное на ЭВМ ДЦ, получает от модема закодированную служебную информацию и декодирует её. Далее считываются остальные биты принятой искаженной закодированной информации. Затем происходит формирование массива кодовых бит, его декодирование и вычисление

проверочного символа (смотри п. 4.2.3.). Найденный проверочный символ сравнивается с проверочным символом, содержащимся в служебной информации. В случае равенства символов декодированный массив преобразуется в файл. Иначе происходит повторный запрос передачи данных. Повторные запросы (переспросы) содержат рекомендации по применению кодирования, - вначале каскадный код РС+Х, а затем каскадный код РС+Г.

Для реализации обмена данных, в дополнение к ПО каскадного кодирования - декодирования, разработаны четыре программных модуля:

1) программный модуль осуществляющий преобразование файла любого типа в массив данных;

2) программный модуль осуществляющий обратное преобразование массива данных в файл;

3) программный модуль осуществляющий формирование пакетов закодированных данных и взаимодействие с программным обеспечением модема «Спектр-43З» для осуществления передачи данных на другую ЭВМ;

4) программный модуль осуществляющий взаимодействие с программным обеспечением модема «Спектр-43З» при приёме пакетов данных, проверку целостности пакетов данных и формирование массива пригодного для проведения процедуры декодирования.

Два программных модуля, осуществляющих взаимодействие с модемом, объединены в один приведенный в приложении № 7.

Два программных модуля, осуществляющих преобразование файла в массив данных и обратно, объединены в один приведенный в приложении № 8.

4.2.3. Структура запроса передачи данных и служебной информации

Инициированию передачи данных от КО на ДЦ может служить специальный запрос от ДЦ. Далее будем называть такой запрос направляемый от ДЦ на КО (смотри п. 4.2.2.) - запросом передачи данных. Биты запроса передачи даннных можно распределить следующим образом:

- код ДЦ – 5 бит;

- номер КО – 4 бит;

- разрешающий бит варианта кодирования – 1 бит;

- вариант кодирования – 2 бита;

- разрешающий бит флага перемежения – 1 бит;

- флаг перемежения – 1 бит;

- разрешающий бит ретрансляции – 1 бит;

- флаг ретрансляции – 1 бит;

- информация о местоположении файла данных – 5 бита.

Разрешающие биты разрешают использовать информацию (1-2 бита), идущую следом за ними.

Для успешного декодирования принятого сообщения программе декодирования должен быть известен ряд параметров информационного сообщения. Назовем эти данные служебной информацией (смотри п. 4.2.2.). Служебная информация передается, предшествуя информационному сообщению. После декодирования запроса передачи данных, ПО КО формирует массив служебной информации. Служебные биты содержат всю информацию, необходимую ЭВМ ДЦ для проведения успешного декодирования. Например их можно распределить следующим образом:

- номер КО – 4 бит;

- вариант кодирования – 2 бита;

- флаг перемежения – 1 бит;

- флаг ретрансляции (устанавливается если сообщение предназначено для приема ретранслятором, который произведя успешное декодирование, убирает флаг и кодирует сообщение повторно) – 1 бит;

- объем информационных данных – 12 бит;

- проверочный символ – 8 бит.

Вычисление проверочного символа проводится суммированием по модулю 2 побитно всех информационных байт, например, с помощью следующего короткого программного модуля:

```
int SGF[8]={0,0,0,0,0,0,0,0};
for(i=0; i< kolinf; i++)
{for(j=0; j<8; j++)
SGF[j]= SGF[j]^ uinf [i*8+j];}
```

В программном модуле применены следующие обозначения: uinf[] – массив информационных бит, kolinf – количество информационных байт, SGF[8] – проверочный символ.

Служебные биты и биты запроса данных и биты служебной информации кодируются кодом БЧХ (n=15, k=7, t=2) образуя четыре и три кодовых слова соответственно. Далее, для лучшей устойчивости к пакетам ошибок, проводится их перемежение. После кодирования получается соответственно 60 и 45 кодовых бит. Проведение перемежения кодовой последовательности из трех слов, делает её устойчивой к появлению пакетов ошибок. На основе кодера-декодера БЧХ разработан программный кодер-декодер бит запроса данных.

В приложении № 6 приведен программный кодер-декодер бит запроса данных осуществляющий:

- кодирование информации тремя словами кода БЧХ и перемежение слов;
- искажение кодовой последовательности в соответствии с задаваемым пользователем количеством и номерами ошибочных бит;
- расперемежение и декодирование кодовых слов.

4.3. Эксперименты по передачи закодированных данных с использованием радиомодемов

Для экспериментов использованы два ноутбука с подключенными радиомодемами «Спектр-433». В качестве антенн для радиомодемов использованы всенаправленные антенны АШ-433, имеющие усиление 3 дБ, и представляющие собой вертикальный полуволновой диполь с круговой диаграммой направленности в горизонтальной плоскости с вертикальной

поляризацией. Высота подъема антенн – 1 м. Тип модуляции модема – FSK, частота работы 433 МГц. Радиомодем «Спектр-433» и антенна АШ-433 изображены на рисунке 4.6.

Рисунок 4.6. – Радиомодем «Спектр-433» и антенна АШ-433

На радиомодеме установлен режим «Прямой доступ», предоставляющий пользователю возможность применять собственные протоколы и методы кодирования [86]. Место проведения эксперимента – сельская местность с одноэтажной застройкой и расстоянием между домами около 20 м.

Исследования проводились при длине информационного сообщения – 800 бит. Количество переданных сообщений для каждого эксперимента – 200. Мощность передатчика – 120 мВт. Результаты экспериментов приведены в таблице 4.5. В указанной таблице использованы следующие обозначения: M – число кодовых бит, l – расстояние между приёмной и передающей антеннами.

Таблица 4.5. Количество ошибок в зависимости от дальности

№	Метод кодирования	M	Количество успешно декодированных информационных сообщений.			
			l=1000 м	l=2000 м	l=2500 м	l=3200 м
1.	-	800	98	18	6	2
2.	РС	1440	181	48	21	1
3.	РС+Х	2520	189	72	43	1
4.	РС+Г	2760	197	118	65	0

Антенны с большим коэффициентом усиления и направленными свойствами обеспечат значительное увеличение дальности.

Результаты экспериментов подтверждают эффективность каскадного кодирования, выявленную ранее. Из полученных результатов также видно, что при большом количестве ошибок (эксперимент на дальности 3200 м) помехоустойчивое кодирование не эффективно. Избыточные биты значительно увеличивают вероятность появления ошибочного бита при передаче кодового сообщения, а декодер уже не в состоянии противостоять большому числу ошибок [1, 28, 50].

Выводы по главе 4

Для экспериментальных исследований программных кодеров-декодеров разработано ПО осуществляющее генерацию массива информационных данных, массива независимых ошибок и пакетов ошибок. Проведены исследования эффективности работы каскадного кодера-декодера при программной эмуляции наличия в канале независимых ошибок и пакетов ошибок. Исследования подтвердили правильность расчетов, проведенных в главе 2.

Даны рекомендаций по применению разработанного кодера-декодера в системах передачи данных различных ЛСТ. В зависимости от вида ЛСТ необходимо разработать разнообразные программные модули подключаемые к программам кодирования и декодирования, а также применять различные варианта алгоритмов обмена данными между ДЦ и КО.

Проведена разработка и построение универсального программно-аппаратного комплекса для ЛСТ, осуществляющего передачу данных между ЭВМ КО и ЭВМ ДЦ с использованием радиомодемов осуществляющих обмен данными с ЭВМ через интерфейс RS-232.

Разработан программный модуль, осуществляющий взаимодействие с радиомодемом.

Разработан программный модуль, осуществляющий преобразование файла в массив данных и обратно.

Разработан программный кодер-декодер бит запроса данных и служебной информации.

Проведены эксперименты по передаче закодированных данных с использованием радиомодемов «Спектр-433» и антенна АШ-433. Результаты экспериментов подтверждают эффективность каскадного кодирования, выявленную ранее.

ЗАКЛЮЧЕНИЕ

1. Рассмотрены задачи современных систем телеметрии. Определены факторы снижающие достоверность передачи радиосигналов связи ДЦ и КО. Рассмотрены различные варианты повышения помехозащищенности.

2. Рассмотрены различные методы помехозащищающего кодирования. В случае ЛСТ использующей радиоканал для передачи данных имеется высокая вероятность появления как независимых ошибок, так и пакетов ошибок. Применение простейших двоичных кодов типа кодов Хэмминга здесь не имеет смысла даже при переспросе, ведь повторная посылка с высокой долей вероятности будет также содержать ошибки. Такие коды вместо исправления ошибок будут вносить новые. Двоичные коды БЧХ более успешно решат проблему возросшей вероятности появления независимых ошибок, однако и они с пакетами ошибок эффективно не справляются. Пакетами ошибок наиболее эффективно противостоит код РС. Для большей эффективности будем использовать код РС в качестве внешнего кода в каскаде с внутренним двоичным кодом. Разновидность двоичного кода, применяемого в каскаде целесообразно изменять адекватно помеховой обстановке (числу ошибок, переспросов).

3. На основании анализа методов помехоустойчивого кодирования предложен метод адаптивного каскадного кодирования-декодирования нерегулярных по длине информационных сообщений, для которого разработаны соответствующие алгоритмы и программное обеспечение.

4. Разработан программный адаптивный каскадный кодер-декодер для нерегулярных по длительности сообщений. Разработан инструментарий и проведены экспериментальные исследования помехоустойчивости разработанного каскадного кодера-декодера. Проведен теоретический анализ и экспериментальные испытания разработанного программного кодера-декодера, показавшие эффективность его борьбы как с независимыми ошибками, так и с

пакетами ошибок, возникающими в канале связи. Получены следующие результаты исследования кодера декодера:

- при наличии в канале независимых ошибок с вероятностью появления ошибочного бита $1 \cdot 10^{-2}$, вероятность появления ошибочного бита в декодированном сообщении снижается до $7{,}2 \cdot 10^{-9}$, что соответствует выигрышу в величине Eb/No на 2,5 дБ;

- при наличии в канале пакетов ошибок, декодер способен исправить пакет максимальным размером: при кодировании кодом РС - 48 бит; при кодировании кодом РС в каскаде с кодом Хемминга - 86 бит; при кодировании кодом РС в каскаде с кодом Голея – 92 бит.

5. Даны рекомендации по применению разработанного каскадного кодера-декодера в различных ЛСТ. Разработан универсальный программно-аппаратный комплекс для ЛСТ, осуществляющий передачу данных между ЭВМ контролируемых объектов и ЭВМ диспетчерского центра с применением радиомодемов, использующий в своей работе адаптивный программный кодер-декодер. Разработана программа кодирования-декодирования бит запроса данных и служебной информации.

6. Разработаны программные модули, осуществляющие преобразование файла в массив данных и обратно, а также реализующие взаимодействие с ПО радиомодема. Проведены эксперименты по передаче закодированных данных с использованием радиомодемов «Спектр-433» и антенна АШ-433. Результаты экспериментов подтверждают эффективность каскадного кодирования, выявленную ранее.

Список сокращений

ДЦ – диспетчерский центр

КО – контролируемый объект

ЛСТ – локальны системы телеметрии

КИП – контрольно-измерительные приборы

БЧХ – Боуз-Чоудхури-Хоквенгем

РС – Рид-Соломон

МКЭ – минимальный кодовый элемент каскадного кода

GF – поле Галуа

РС+Х – Рида-Соломона и Хемминга

РС+Г – Рида-Соломона и Голея

ПО – программное обеспечение

УПАК – универсальный программно-аппаратный комплекс

Список литературы

1. Скляр Б. Цифровая связь. Теоретические основы и практическое применение. – М.: Издательский дом «Вильямс», 2007 – 1104 с.

2. Вернер М. Основы кодирования. – М.: Техносфера, 2006 – 286 с.

3. Морелос-Сарагоса Р. Искусство помехоустойчивого кодирования. Методы, алгоритмы, применение. – М.: Техносфера, 2006 – 319 с.

4. I.N. Herstein, Topics in Algebra, 2-ое издание, John Wiley and Sons, 1975.

5. Питерсон У., Уэлдон Э. Коды, исправляющие ошибки. – М.: Мир, 1976.

6. E.R. Berlekamp, Algebraic Coding Theory, rev. ed., Aegean Park Press, 1984.

7. J.L. Massey, «Shift Register and BCH Decoding», IEEE Trans. Info. Theory, vol. IT-15, no. 1, pp. 122-127, Jan. 1969.

8. Форни Д. Каскадные коды. – М.: Мир, 1970.

9. Касперский К. Техника защиты компакт-дисков от копирования. – СПб.: БХВ-Петербург, 2004.

10. Боровков А. А. Теория вероятностей. – М.: – Наука, 1986.

11. Чеботарев Н.Г. Основы теории Галуа. Часть1. – М.: Едиториал УРСС, 2004.

12. Блейхут Р. Теория и практика кодов, контролирующих ошибки. – М.: Мир, 1986.

13. Берлекэмп Э. Алгебраическая теория кодирования. – М. Мир, 1971.

14. Финк Л.М. Сигналы, помехи, ошибки: – М.: Радио и связь, 1984.

15. Д. Сэломон. Практическое руководство по методам сжатия данных. – М.: Техносфера, 2004.

16. Василенко О.Н. Теоретико-числовые алгоритмы в криптографии. – М.: МЦНМО, 2003.

17. Булинский А. В. Ширяев. А. Н. Теория случайных процессов. – М.: Физматлит, 2003.

18. Кнут Д. Искусство программирования. Том 2. – М.: Вильямс, 2005.

19. Гнеденко Б.В. Курс теории вероятностей. – М.: Едиториал УРСС, 2005.

20. Баврин И.И. Высшая математика. – М.: Наука, 2000.

21. Мак-Вильямс Ф. Дж., Слоэн Н.Дж.А. Теория кодов, исправляющих ошибки. – М.: Связь, 1979.

22. Блох Э.Л., Зяблов В.В. Обобщенные каскадные коды. – М.: Связь,1976.

23. Коржик В.И., Финк Л.М. Помехоустойчивое кодирование дискретных сообщений в каналах со случайной структурой. – М.: Связь, 1976.

24. Кларк Дж. мл., Кейн Дж. Кодирование с исправлением ошибок в системах цифровой связи. – М.: Радио и связь, 1987.

25. Боуз Р.Ч., Рой-Чоудхури Д.К. Об одном классе двоичных групповых кодов с исправлением ошибок / Кибернетический сборник, вып. 2. – М.: ИЛ, 1961. – С.83 - 94.

26. Berrou C. Glavieux A., Thitimajshima P. Near Shannon Limit Error – Correcting Coding and Decoding: Turbo-Codes, Proc. Intern. Conf. On Communication – ICC-93, Geneva, Svitzerland, 1993. – Pp. 1064 – 1070.

27. Паньшо С.П., Югай В.В. Турбокодирование / Успехи современной радиоэлектроники, № 2, 2004. – С. 3 – 16.

28. Самсонов Б.Б., Плохов Е.М., Филоненков А.И., Кречет Т.В. Теория информации и кодирования. – Ростов на Дону: Феникс, 2002. - 288 с.

29. Григорьев А.С., Дронов А.Е. Турбокодирование в системах однонаправленной передачи цифровой информации // Материалы 12 межрегиональной конференции «Обработка сигналов в системах телефонной связи и вещания», - Пушкинские горы – Москва, МТУСИ, 2003. – С. 103.

30. Красносельский И.Н. Турбокоды: принципы и перспективы // Электросвязь, № 1, 2001. – С. 17 – 20.

31. Осмоловский С.А. Сравнительный анализ некоторых свойств стохастических кодов и кодов Рида – Соломона // Электросвязь, № 1, 1991.

32. Рид, Соломон. Полиномиальные коды над некоторыми конечными полями / Кибернетический сборник, вып. 7. – М.: ИЛ, 1963.

33. Кловский Д.Д. Передача дискретных сообщений по радиоканалам. – М.: Связь, 1982.

34. Галкин А.П., Лапин А.Н., Самойлов А.Г. Моделирование каналов систем связи. – М.: Связь, 1979.

35. Феер К. Беспроводная цифровая связь. Методы модуляции и расширения спектра. Пер. с англ. / Под ред. В.И. Журавлева. – М.: Радио и связь, 2000.

36. Справочник по радиорелейной связи / Под. Ред. С.В. Бородича. – М.: Радио и связь, 1981.

37. Прокис Д. Цифровая связь / Под. Ред. Д.Д. Кловского. – М.: Радио и связь, 2000.

38. Каганцов С.М., Полушин П.А., Самойлов А.Г., Самойлов С.А. Кодек для цифровых радиорелейных станций // Пятая Российская конференция по атмосфеному электричеству. Том 2, изд. «Транзит ИКС», Владимир. 2003. – С. 148 – 149.

39. Коржик В.И., Финк Л.М., Щелкунов К.Н. Расчет помехоустойчивости систем передачи дискретных сообщений. – М.: Радио и связь, 1981. – 232 с.

40. Злотник Б.М. Помехоустойчивые коды в системах связи. – М.: Радио и связь, 1989.

41. Андре Анго Математика для электро- и радиоинженеров. – М.: Наука, 1964.

42. Альшрайдех А.М., Гомес Ж.Л., Самойлов С.А., Сидоренко А.А. Исследование «мягкого» декодирования кода Рида-Соломона / Проектирование и технология электронных средств. – 2014, №1. – С. 8-11.

43. Айвор Хортон. Visual C++ 2010: полный курс. – М.: Диалектика, 2010.

44. Возенкрафт Дж., Джекобс И. Теоретические основы техники связи. – М.: Мир, 1969.

45. Виттерби А.Д., Омура Дж.К. Принципы цифровой связи и кодирования. – М.: Радио и связь, 1982.

46. Волков Л.Н., Немировский М.С., Шинаков Ю.С. Системы цифровой радиосвязи: базовые методы и характеристики: Учебное пособие. – М.: Эко-Трендз, 2005.

47. Сорока Н.И., Кривинченко Г.А. Телемеханика: Конспект лекций для студентов специальности "Автоматическое управление в технических системах". Ч.I: Сообщения и сигналы. – Мн.: БГУИР, 2000.

48. Бородин Л.Ф. Введение в теорию помехоустойчивого кодирования. – М.: Сов. Радио, 1968.

49. Советов Б.Я. Сравнительная оценка надежности передачи информации избыточными кодами // Автоматика и телемеханика, 1970, № 2. – С. 41-44.

50. Советов Б.Я. Эффективность введения избыточности в системы передачи телемеханической информации. – Л.: Наука, 1970.

51. Комаров В.Н. Об исправлении многократных пакетов ошибок / Техника средств связи, 1980. Вып. 3.

52. Котов П.А. Повышение достоверности передачи цифровой информации. – М.: Связь, 1966. – 192 с.

53. Сидоренко А.А. Использование арифметики полей Галуа при построении кодов Рида-Соломона / Проектирование и технология электронных средств. – 2011, №3. – С. 13-17.

54. Самойлов А.Г., Сидоренко А.А. Применения кодов РС в каскаде с двоичными кодами с целью повышения эффективности борьбы с независимыми ошибками / Проектирование и технология электронных средств. – 2014, №3. – С. 2-7.

55. Сидоренко А.А. Методы построения систем мониторинга объектов // Материалы IX международной научно-технической конференции «Перспективные технологии в средствах передачи информации». – Владимир-Суздаль, 2011. – Т. 2. – С. 209-210.

56. Сидоренко А.А. Применение помехоустойчивого кодирования с исправлением ошибок при передаче цифровых сигналов // Материалы вторых Всероссийских Армандовских чтений. Сб. тезисов докладов научно-практического семинара. – Муром, 2012. – С. 36-37.

57. Сидоренко А.А. Кодек для систем телеметрии, работающих в условиях сложной помеховой обстановки // Материалы XV Всероссийской научной конференции студентов-радиофизиков. – Санкт-Петербург, 2011. – С. 108-112.

58. Сидоренко А.А. Построение кодов, исправляющих ошибки с использованием арифметики полей Галуа // II международная заочная научно-техническая конференция «Информационные технологии. Радиоэлектроника. Телекоммуникации (ITRT-2012)». – Тольятти, 2012. – Ч. 3. – С. 230-236.

59. Сидоренко А.А. Адаптивное помехоустойчивое кодирование // Материалы X международной научно-технической конференции «Перспективные технологии в средствах передачи информации». – Владимир-Суздаль, 2013. – Т. 1. – С. 152-154.

60. Сидоренко А.А. Построение универсального алгоритма Берлекемпа-Месси для кодов Боуза-Чоудхури-Хоквенгема // Материалы X международной научно-технической конференции «Перспективные технологии в средствах передачи информации». – Владимир-Суздаль, 2013. – Т. 1. – С. 155-157.

61. Сидоренко А.А. Анализ эффективности кодов Рида-Соломона в борьбе с независимыми ошибками и пакетами ошибок // 21 Всероссийская межвузовская научно-техническая конференция студентов и аспирантов «Микроэлектроника и информатика-2014». – Москва, 2014. – С. 196.

62. Самойлов А.Г., Сидоренко А.А. Топология системы мониторинга городского водоснабжения // Международная НТК «Физика и радиоэлектроника в медицине и экологии». – Владимир, 2012. – Кн.2. – С. 210-212.

63. Самойлов А.Г., Сидоренко А.А. Алгоритм информационного обмена для системы городского водоснабжения // Международная НТК «Физика и радиоэлектроника в медицине и экологии». – Владимир, 2012. – Кн.2. – С. 312-313.

64. Советов Б.Я. Помехоустойчивость передачи команд телеуправления в системе с запросом // Автоматика и телемеханика, 1966, № 12. – с. 43 – 48.

65. S. Samoilov. Reed – Solomon Codes Communication System // 5 – th Workshop Digital Broadcasting, Erlangen, Germany, September 23 – 24, 2004. pp. 85 – 86.

66. Mayo-Wells The Origins of Space Telemetry, – Technology and Culture, 1963.

67. Ekroot L., Dolinar S. A Decoding of Block Codes // IEEE Transaction Inform Theory, September 1993, pp. 1052 – 1056.

68. Rappaport T.S. Wireless Communication (Principles and Practice). – N.Y. Prentice Hall, 1996.

69. Робинсон Ф.Н. Шумы и флуктуации в электронных схемах и цепях. – М.: Атомиздат, 1980. – 256 с.

70. Зюко А.Г., Кловский Д.Д., Назаров М.В., Финк Л.М. Теория передачи сигналов. – М.: Радио и связь, 1998.

71. Керниган Б.У., Ритчи Д.М. Язык программирования Си – М.: Вильямс. – 2013.

72. Страуструп Б. Язык программирования С++. – М: Бином. – 2011 г.

73. Пятибратов А.П. Вычислительные системы, сети и телекоммуникации. – М. : Радио и связь, 1998.

74. Ратынский М.В. Основы сотовой связи. – М.: Радио и связь, 1998. – 392 с.

75. Шмалько А.В. Цифровые сети связи: Основы планирования и построения. – М.: Эко-Трендз, 2001. – 282 с.

76. Коржик В.И., Финк Л.М. Помехоустойчивое кодирование дискретных сообщений в каналах со случайной структурой. – М.: Связь, 1979. – 272 с.

77. Андрианов В.И., Соколов А.В. Сотовые, пейджинговые и спутниковые средства связи. – СПб.: БХВ-Петербург, 2001. – 400 с.

78. Телекоммуникационные системы и сети, т. 2. – Радиосвязь, радиовещание и телевидение. / Под ред. В.П. Шувалова. – Горячая линия – Телеком, 2004. – 672 с.

79. Цифровая обработка сигналов. / Под ред. А.Б. Сергиенко – СПб.: Питер, 2003. – 604 с.

80. Некоторые вопросы теории кодирования. Сборник переводов. / Под ред. Э.Л. Блоха и М.С. Пинскера. – М.: Мир, 1970 – 275 с.

81. Ziemer R. and Peterson R. Introduction to Digital Communication, 2 ded., Prentice Hall, 2001.

82. Clark G.C. and Cain J.B. Error Correction Coding for Digital Communications. – Plenum Press, New York, 1981.

83. Громаков Ю.А. Стандарты и системы подвижной радиосвязи. – М.: Эко-Трендз, 1998. – 239с.

84. Бородич С.В. Искажения и помехи в многоканальных системах радиосвязи с частотной модуляцией – М.: Связь, 1976. – 256 с.

85. Банкет В.Л., Дорофеев В.М. Цифровые методы в спутниковой связи. – М.: Радио и связь, 1988. – 240 с.

86. Белоцерковский И.Л. Протоколы передачи файлов для модемов // Сети, 1995, № 3 – с. 53 – 59.

87. Голд Б., Рейдер Ч. Цифровая обработка сигналов. – М.: Сов. Радио, 1973. – 367 с.

88. Berrou C., Glavieux A, Thitimajshima P. Near Shannon Limit Error-Correcting Coding and Decoding: Turbo-Codes", Proceedings of International Conference on Communications, Geneva, Switzerland. – 1993, May. – P. 1064-1070.

89. Шлома А.М., Бакулин М.Г., Крейнделин В.Б., Шумов А.П. Новые алгоритмы формирования и обработки сигналов в системах подвижной связи / Под редакцией профессора А.М.Шломы. – М.: Горячая линия–Телеком. – 2008. – 344 с.

90. Волков А.А., Карпова Г.В., Журавлев О.Е. Повышение помехоустойчивости радиосвязи / А.А. Волков, Г.В. Карпова, О.Е. Журавлев// Мир транспорта. – 2012. - №3. – с.31-33.

91. Золотарёв В.В., Овечкин Г.В. Помехоустойчивое кодирование. Методы и алгоритмы. – М.: Горячая линия - Телеком, 2004. – 126 с.

92. Золотарёв В.В., Овечкин Г.В., Зубарев Ю.Б., Левин В.К. Многопороговые декодеры и оптимизационная теория кодирования. – М.: Горячая линия - Телеком, 2012. – 239 с.

Приложение № 1

Кодер-декодер Хемминга (7,4)

```
int _tmain(int argc, _TCHAR* argv[])
{int i, j, kerh, nerh;
bool AGH[7]={1,1,0,1,0,0,0};//Задаем по строкам порождающую матрицу
bool BGH[7]={0,1,1,0,1,0,0};
bool CGH[7]={1,1,1,0,0,1,0};
bool DGH[7]={1,0,1,0,0,0,1};
int uhm[4];//Информационное слово
int vhm[7];//Кодовое слово
printf("\n Ввод данных: \n");
for(j=0; j<4; j++)
scanf("%d", &uhm[j]);
printf("\n Кодовые символы:          ");
vhm[0]=uhm[0]*AGH[0]^uhm[1]*BGH[0]^uhm[2]*CGH[0]^uhm[3]*DGH[0];
//printf("\n S0: %4.1d \n", SINDR[0]);
printf("%4.1d", vhm[0]);
vhm[1]=uhm[0]*AGH[1]^uhm[1]*BGH[1]^uhm[2]*CGH[1]^uhm[3]*DGH[1];
printf("%4.1d", vhm[1]);
vhm[2]=uhm[0]*AGH[2]^uhm[1]*BGH[2]^uhm[2]*CGH[2]^uhm[3]*DGH[2];
printf("%4.1d", vhm[2]);
vhm[3]=uhm[0]*AGH[3]^uhm[1]*BGH[3]^uhm[2]*CGH[3]^uhm[3]*DGH[3];
printf("%4.1d", vhm[3]);
vhm[4]=uhm[0]*AGH[4]^uhm[1]*BGH[4]^uhm[2]*CGH[4]^uhm[3]*DGH[4];
printf("%4.1d", vhm[4]);
vhm[5]=uhm[0]*AGH[5]^uhm[1]*BGH[5]^uhm[2]*CGH[5]^uhm[3]*DGH[5];
printf("%4.1d", vhm[5]);
vhm[6]=uhm[0]*AGH[6]^uhm[1]*BGH[6]^uhm[2]*CGH[6]^uhm[3]*DGH[6];
printf("%4.1d", vhm[6]);
printf("\n Vvedi chislo oshibok: \n");//Зададим количество ошибочных бит
```

```
scanf("%d", &kerh);
printf("\n Vvedi nomer oshibki: \n");//Зададим номера ошибочных бит
for(i=0; i<kerh; i++)
{scanf("%d", &nerh);
vhm[nerh]=vhm[nerh]^1;}
//Задаем по строкам транспонированную проверчную матрицу:
bool APH[3]={1,0,0};
bool BPH[3]={0,1,0};
bool CPH[3]={0,0,1};
bool DPH[3]={1,1,0};
bool EPH[3]={0,1,1};
bool FPH[3]={1,1,1};
bool GPH[3]={1,0,1};
bool sihm[3]; //значение синдрома в виде двоичных коэффициентов
int SSHM; //значение синдрома в виде десятичного числа
printf("\n      Синдром:          ");
sihm[0]=vhm[0]*APH[0]^vhm[1]*BPH[0]^vhm[2]*CPH[0]^vhm[3]*DPH[0]^vhm[
4]*EPH[0]^vhm[5]*FPH[0]^vhm[6]*GPH[0];
sihm[1]=vhm[0]*APH[1]^vhm[1]*BPH[1]^vhm[2]*CPH[1]^vhm[3]*DPH[1]^vhm[
4]*EPH[1]^vhm[5]*FPH[1]^vhm[6]*GPH[1];
sihm[2]=vhm[0]*APH[2]^vhm[1]*BPH[2]^vhm[2]*CPH[2]^vhm[3]*DPH[2]^vhm[
4]*EPH[2]^vhm[5]*FPH[2]^vhm[6]*GPH[2];
SSHM=sihm[0]+2*sihm[1]+4*sihm[2];
printf("%4.1d", SSHM);
if (SSHM==0)
printf("\n Ошибок не обнаружено \n");
//Исправляем только ошибки в информационных битах:
if (SSHM==3)
vhm[3]=vhm[3]^1;
```

```
if (SSHM==6)
vhm[4]=vhm[4]^1;
if (SSHM==7)
vhm[5]=vhm[5]^1;
if (SSHM==5)
vhm[6]=vhm[6]^1;
//Проводим проверку успешности декодирования
int ERO=0;//включаем счетчик ошибок
printf(" \n \n Исходные данные - Декодированные \n \n");
for(i=0;i<4;i++)
{printf("%4.1d", uhm[i]);
printf("%4.1d \n", vhm[i+3]);
if (uhm[i]!=vhm[i+3])
ERO=ERO+1;}
printf(" \n Количество ошибок \n");
printf("%4.1d", ERO);}
```

Приложение № 2

Кодер-декодер БЧХ (15,7)

```
int _tmain(int argc, _TCHAR* argv[])
{int s, i, j, n, rg, kk,nerb,kerb;
int g[9]={1,0,0,0,1,0,1,1,1};//-генерирующий полином
int UBC[7];//Информационное слово
int PRN[15]={0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0};//Кодовое слово
printf("\n Ввод данных: \n");
for(j=0; j<7; j++)
{scanf("%d", &UBC[j]);
PRN[j+8]=UBC[j];}
kk=15;
rg=8;
//Размерность массива остатков R[kk-rg+1][rg]
int R[8][8], RA, RB, RC, RD;
for(j=0; j<=rg-1; j++)
R[0][j]=PRN[kk-rg+j];
for(s=1; s<=kk-rg; s++)
{RA=g[0]*R[s-1][rg-1];
R[s][0]=RA^PRN[kk-rg-s];
for(j=1; j<=rg-1; j++)
{RB=g[j]*R[s-1][rg-1];
R[s][j]=RB^R[s-1][j-1];}}
printf("\n Проверочные символы:");
for(j=0; j<=rg-1; j++)
{PRN[j]=R[kk-rg][j];
printf("%4.1d ", PRN[j]);}
bool SINDR[8];//Двоичный массив синдромов
//Задаем 8 столбцов проверочной матрицы:
bool APR[15]={0,0,0,0,0,0,0,1,1,0,1,0,0,0,1};
```

```
bool BPR[15]={0,0,0,0,0,0,1,1,0,1,0,0,0,1,0};
bool CPR[15]={0,0,0,0,0,1,1,0,1,0,0,0,1,0,0};
bool DPR[15]={0,0,0,0,1,1,0,1,0,0,0,1,0,0,0};
bool EPR[15]={0,0,0,1,1,0,1,0,0,0,1,0,0,0,0};
bool FPR[15]={0,0,1,1,0,1,0,0,0,1,0,0,0,0,0};
bool GPR[15]={0,1,1,0,1,0,0,0,1,0,0,0,0,0,0};
bool HPR[15]={1,1,0,1,0,0,0,1,0,0,0,0,0,0,0};
//Две Таблицы проверки синдромов содержащие номера ошибочных бит:
int RPR[256]={30,14,13,13,12,12,12,6,11,11,11,0,11,30,5,30,10,10,6,10,6,10,6,6,
30,10,30,30,30,4,6,30,9,30,9,9,5,2,9,30,5,30,9,30,5,5,5,30,0,30,30,9,6,30, 30,30,
2,30,5,3,30,5,30,30,0,8,2,30,8,8,30,8,30,4,30,1,2,8,4,30,4,30,30,6,8,30,30,30,4,4,
4,30,4,30,30,30,2,2,2,30,2,30,30,8,2,5,30,30,30,30,30,30,1,2,30,30,4,30,2,30,30,30,4,
30,30,30,30,30,0,0,0,7,0,1,30,30,0,7,7,7,30,30,30,7,30,0,3,30,30,30,0,30,1,30,7,30,3,
6,30,30,1,3,0,30,30,30,3,5,30,7,30,30,0,30,30,30,3,3,3,30,3,30,30,30,3,30,30,30,30,3
0,30,30,1,1,1,30,1,0,30,30,1,30,30,30,30,30,7,30,2,1,30,4,30,30,1,30,0,30,30,30,30,3
0,30,30,0,30,1,30,30,30,30,30,3,30,30,2,1,30,30,30,30,30,30,30,30,3,30,30,30,30,30,
30,30,30,30,30};
intVTPR[256]={30,30,30,14,30,14,13,10,30,14,13,7,12,30,9,30,14,30,12,13,13,12,30
,14, 30,11,30,30,30,8,11,30,13,30,30,14,11,8,12,30,12,30,11,30,30,14,13,30,3,
30,30,10,9,30,30,30,4,30,6,7,30,10,30,30,1,12,9,30,14,30,30,13,30,10,30,7,5,11,6,30,
11,30,30,8,10,30,30,30,30,14,13,30,12,30,30,30,30,14,13,30,12,30,30,9,11,8,30,30,3
0,30,30,30,3,10,30,30,5,30,6,30,30,30,9,30,30,30,30,30,30,14,13,11,12,8,30,30,11,13
,14,30,30,30,30,12,30,10,9,30,30,30,6,30,4,30,10,30,5,7,30,30,2,10,9,30,30,30,6,7,30
,9,30,30,5,30,30,30,30, 14,13,30,12,30,30,30,11,30,30,30,30,30,30,30,
30,14,13,30,12,8,30,30,11,30,30,30,30,30,8,30,3,10,30,7,30,30,6,30,4,30,30,30,30,30
,30,30,2,30,9,30,30,30,30,30,4,30,30,7,5,30,30,30,30,30,30,30,30,8,30,30,30,30,30,3
0,30,30,30,30};
int prv, vte, SSINDR;//SSINDR - синдром в десятичном виде
prv=0;
vte=0;
```

```
printf("\n Кодовое слово: ");
for(i=0; i<=14; i++)
printf("%4.1d", PRN[i]);
//ВВодим ошибки:
printf("\n Введите число ошибок: \n");//Зададим количество ошибочных бит
scanf("%d", &kerb);
printf("\n Введите номера ошибок: \n");//Зададим номера ошибочных бит
for(i=0; i<kerb; i++)
{scanf("%d", &nerb);
PRN[nerb]=PRN[nerb]^1;}
printf("\n Искаженное кодовое слово: ");
for(i=0; i<=14; i++)
printf("%4.1d", PRN[i]);
printf("\n Синдром: ");
//Находим синдром осуществляя умножение вектора принятого слова на
//проверочную матрицу:
SINDR[0]=PRN[0]*APR[0]^PRN[1]*APR[1]^PRN[2]*APR[2]^PRN[3]*APR[3]^P
RN[4]*APR[4]^PRN[5]*APR[5]^PRN[6]*APR[6]^PRN[7]*APR[7]^PRN[8]*APR[
8]^PRN[9]*APR[9]^PRN[10]*APR[10]^PRN[11]*APR[11]^PRN[12]*APR[12]^PR
N[13]*APR[13]^PRN[14]*APR[14];
printf("%4.1d", SINDR[0]);
SINDR[1]=PRN[0]*BPR[0]^PRN[1]*BPR[1]^PRN[2]*BPR[2]^PRN[3]*BPR[3]^P
RN[4]*BPR[4]^PRN[5]*BPR[5]^PRN[6]*BPR[6]^PRN[7]*BPR[7]^PRN[8]*BPR[8
]^PRN[9]*BPR[9]^PRN[10]*BPR[10]^PRN[11]*BPR[11]^PRN[12]*BPR[12]^PRN
[13]*BPR[13]^PRN[14]*BPR[14];
printf("%4.1d", SINDR[1]);
SINDR[2]=PRN[0]*CPR[0]^PRN[1]*CPR[1]^PRN[2]*CPR[2]^PRN[3]*CPR[3]^P
RN[4]*CPR[4]^PRN[5]*CPR[5]^PRN[6]*CPR[6]^PRN[7]*CPR[7]^PRN[8]*CPR[8
]^PRN[9]*CPR[9]^PRN[10]*CPR[10]^PRN[11]*CPR[11]^PRN[12]*CPR[12]^PRN
[13]*CPR[13]^PRN[14]*CPR[14];
```

```
printf("%4.1d", SINDR[2]);
SINDR[3]=PRN[0]*DPR[0]^PRN[1]*DPR[1]^PRN[2]*DPR[2]^PRN[3]*DPR[3]^P
RN[4]*DPR[4]^PRN[5]*DPR[5]^PRN[6]*DPR[6]^PRN[7]*DPR[7]^PRN[8]*DPR[
8]^PRN[9]*DPR[9]^PRN[10]*DPR[10]^PRN[11]*DPR[11]^PRN[12]*DPR[12]^PR
N[13]*DPR[13]^PRN[14]*DPR[14];
printf("%4.1d", SINDR[3]);
SINDR[4]=PRN[0]*EPR[0]^PRN[1]*EPR[1]^PRN[2]*EPR[2]^PRN[3]*EPR[3]^PR
N[4]*EPR[4]^PRN[5]*EPR[5]^PRN[6]*EPR[6]^PRN[7]*EPR[7]^PRN[8]*EPR[8]^
PRN[9]*EPR[9]^PRN[10]*EPR[10]^PRN[11]*EPR[11]^PRN[12]*EPR[12]^PRN[1
3]*EPR[13]^PRN[14]*EPR[14];
printf("%4.1d", SINDR[4]);
SINDR[5]=PRN[0]*FPR[0]^PRN[1]*FPR[1]^PRN[2]*FPR[2]^PRN[3]*FPR[3]^PR
N[4]*FPR[4]^PRN[5]*FPR[5]^PRN[6]*FPR[6]^PRN[7]*FPR[7]^PRN[8]*FPR[8]^
PRN[9]*FPR[9]^PRN[10]*FPR[10]^PRN[11]*FPR[11]^PRN[12]*FPR[12]^PRN[13
]*FPR[13]^PRN[14]*FPR[14];
printf("%4.1d", SINDR[5]);
SINDR[6]=PRN[0]*GPR[0]^PRN[1]*GPR[1]^PRN[2]*GPR[2]^PRN[3]*GPR[3]^P
RN[4]*GPR[4]^PRN[5]*GPR[5]^PRN[6]*GPR[6]^PRN[7]*GPR[7]^PRN[8]*GPR[
8]^PRN[9]*GPR[9]^PRN[10]*GPR[10]^PRN[11]*GPR[11]^PRN[12]*GPR[12]^PR
N[13]*GPR[13]^PRN[14]*GPR[14];
printf("%4.1d", SINDR[6]);
SINDR[7]=PRN[0]*HPR[0]^PRN[1]*HPR[1]^PRN[2]*HPR[2]^PRN[3]*HPR[3]^P
RN[4]*HPR[4]^PRN[5]*HPR[5]^PRN[6]*HPR[6]^PRN[7]*HPR[7]^PRN[8]*HPR[
8]^PRN[9]*HPR[9]^PRN[10]*HPR[10]^PRN[11]*HPR[11]^PRN[12]*HPR[12]^PR
N[13]*HPR[13]^PRN[14]*HPR[14];
printf("%4.1d", SINDR[7]);
SSINDR=128*SINDR[7]+64*SINDR[6]+32*SINDR[5]+16*SINDR[4]+8*SINDR[3
]+4*SINDR[2]+2*SINDR[1]+SINDR[0];
printf("\n Суммарный синдром: %4.1d", SSINDR);
if (SSINDR<=0)
```

```
printf("\n Ошибок нет ");
else
{//PRPR[SSINDR] - номер первой ошибки, VTPR[SSINDR] - номер второй ошибки
int SPOZ=PRPR[SSINDR]+VTPR[SSINDR];
printf("\n Номер первой ошибки: %4.1d", PRPR[SSINDR]);
printf("\n Номер второй ошибки: %4.1d", VTPR[SSINDR]);
if (SPOZ>=60)
printf("\n Ошибки не исправимы ");
else
{if (SPOZ>=30)
{printf("\n Одна ошибка ");
PRN[PRPR[SSINDR]]=PRN[PRPR[SSINDR]]^1;}
else
{printf("\n Две ошибки ");
PRN[PRPR[SSINDR]]=PRN[PRPR[SSINDR]]^1;
PRN[VTPR[SSINDR]]=PRN[VTPR[SSINDR]]^1;}
}
printf("\n Декодированное слово: ");
for(i=0; i<=14; i++)
printf("%4.1d", PRN[i]);
}}
```

Приложение № 3

Кодер-декодер РС (9,5)

```
int _tmain(int argc, _TCHAR* argv[])
{int s, i, j, R[6][4], RA, RB, RC, RD, A[256],ners,kers,zers;
int URS[5];//Информационное слово
int C[9]={0,0,0,0,0,0,0,0,0};//Кодовое слово
printf("\n Ввод инф символов значением от 0 до 255: \n");
for(j=0; j<5; j++)
{scanf("%d", &URS[j]);
//ВВодим информационные символы сразу на нужные позиции со сдвигом:
C[j+4]=URS[j];}
int g[5]={116,231,216,30,1};//-генерирующий полином
for(j=0; j<4; j++)
R[0][j]=C[5+j];
for(s=1; s<6; s++)
{RA=show_proizv(g[0], R[s-1][3]);
R[s][0]=show_summ(RA,C[5-s],0,0);
for(j=1; j<4; j++)
{RB=show_proizv(g[j], R[s-1][3]);
R[s][j]=show_summ(RB,R[s-1][j-1],0,0);}}
printf("Проверочные символы:");
for(j=0; j<4; j++)
{C[j]=R[5][j];
printf("%4.1d ", C[j]);}
printf("\n Кодовая последовательность:");
for(j=0; j<9; j++)
printf("%4.1d", C[j]);
//ВВодим ошибки:
printf("\n Введи число ошибок: \n");//Зададим количество ошибочных байт
scanf("%d", &kers);
```

```
//Зададим номера ошибочных байт
printf("\n Введи номер ошибки 0-8 и значение ошибки 0-255: \n");
for(i=0; i<kers; i++)
{scanf("%d", &ners);
scanf("%d", &zers);
C[ners]=show_summ(C[ners],zers,0,0);}
// Декодирование
int k, q1, q2, qs, u[2], Ke;
// Формирование элементов основного поля
A[0]=1;
A[255]=0;
for(i=1;i<=254;i++)
{if (A[i-1]<128)
A[i]=A[i-1]*2;
else
A[i]=show_summ(2*A[i-1],285,0,0);}
//Конец формирования
//Вычисление синдромов
int SN, SA, SS;
int S[4];
printf("\n  Синдромы:  \n");
for(j=0;j<=3;j++)
{SN=C[0];
for(i=1;i<=8;i++)
{SA=show_proizv(C[i],A[(j+1)*i]);
S[j]=show_summ(SN,SA,0,0);
SN=S[j];}
printf("%4.1d", S[j]);}
//Конец вычисления синдрома
SS=S[0]+S[1]+S[2]+S[3];
```

```
if (SS==0)
printf("\n  Нет ошибок  \n");
else
{//Начало алгоритма Б-М
int W, Wz, q, m, Dq, Lz[3], deg;
int Vx[3]={0,1,0};
int L[4]={1,0,0,0};
q=0;
W=0;
m=-1;
while (q!=4)
{Dq=0;
for(i=0; i<=W; i++)
Dq=show_summ(Dq,show_proizv(L[i],S[q-i]),0,0);
//Цикл В
if (Dq!=0)
{for(i=0; i<=2; i++)
{Lz[i]=show_summ(L[i],show_proizv(Dq,Vx[i]),0,0);
//Цикл А
if(W<q-m)
{Wz=q-m;
m=q-W;
W=Wz;
for(i=0; i<=2; i++)
{Vx[i]=show_chastn(L[i],Dq);}}
//Конец цикла А
for(i=0; i<=2; i++)
{L[i]=Lz[i];}}
//Конец цикла В
for(i=2; i>=1; i--)
```

```
Vx[i]=Vx[i-1];
Vx[0]=0;
q=q+1;}
//Ищем максимальную степень полинома L[i]:
for(i=0; i<=3; i++)
{if (L[i]>0)
deg=i;}
//Конец алогоритма Б-М
//Ищем локаторы ошибок
k=0;
//Подставляем в полином L[] все элементы поля Галуа и ищем его корни:
for(i=0;i<=255;i++)
{qs=show_summ(show_proizv(L[2],show_proizv(A[i],A[i])),show_proizv(L[1],A[i])
,L[0],0);
//Если полином локатора обращается в ноль, то корень найден:
//k- счетчик корней полинома L[3] и количество ошибок
if (qs==0)
{k=k+1;
u[k-1]=255-i;}}
int PO[4];
//Вычисляем полином величин ошибок PO состоящий из 4 коэффицентов:
PO[3]=show_summ(show_proizv(S[1],L[2]),show_proizv(S[2],L[1]),show_proizv(S
[3],L[0]),0);
PO[2]=show_summ(show_proizv(S[0],L[2]),show_proizv(S[1],L[1]),show_proizv(S
[2],L[0]),0);
PO[1]=show_summ(show_proizv(S[0],L[1]),show_proizv(S[1],L[0]),0,0);
PO[0]=show_proizv(S[0],L[0]);
//Вычисляем значение полинома величин ошибок при подставке
//в него значения x, что в дальнейшем позволит найти величины ошибок.
int Ve[2], XI[4], XP;
```

```
//XP - значение аргумента, XI[i]- массив степеней аргументов полинома x в квадрате, в кубе...
//Ve[2] - это сами величины ошибок (массив ошибок)
for(j=0;j<=k-1;j++)
{//Вычисляем разные степени x
XI[0]=1;
XP=A[255-u[j]];
for(i=1;i<=3;i++)
XI[i]=show_proizv(XI[i-1],XP);
Ve[j]=show_chastn(show_summ(show_proizv(PO[3],XI[3]),show_proizv(PO[2],XI[2]),show_proizv(PO[1],XI[1]),PO[0]),L[1]);}
//Исправляем ошибки:
for(j=0;j<=k-1;j++)
C[u[j]]=show_summ(C[u[j]],Ve[j],0,0);}
int ERO=0;//включаем счетчик ошибок
printf(" \n \n Исходные данные - Декодированные \n \n");
for(i=0;i<5;i++)
{printf("%4.1d", URS[i]);
printf("%4.1d \n", C[i+4]);
if (URS[i]!=C[i+4])
ERO=ERO+1;}
printf(" \n Количество ошибок \n");
printf("%4.1d", ERO);}
```

Приложение № 4

Кодирование-декодирование кодом Голея

```
int main(int argc, char* argv[])
{int R[13][11], SINDR[11],SSR, RA, RB, RC, RD,s,tcik,i,j,bufs,kerg,nerg;
int g[12]={1,0,1,0,1,1,1,0,0,0,1,1};//-генерирующий полином
int ERRM[11]={0,0,1,1,0,0,1,1,0,1,1};
int ERRS[11]={0,1,1,0,0,1,1,0,1,1,0};
int UGL[12];//-Информационное слово
//Кодовое слово:
int C[23]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
printf("\n Ввод данных: \n");
for(j=0; j<12; j++)
{scanf("%d", &UGL[j]);
C[j+11]=UGL[j];}
for(j=0; j<=10; j++)
R[0][j]=C[12+j];
for(s=1; s<=12; s++)
{RA=g[0]*R[s-1][10];
R[s][0]=RA^C[12-s];
for(j=1; j<=10; j++)
{RB=g[j]*R[s-1][10];
R[s][j]=RB^R[s-1][j-1];}}
for(j=0; j<=10; j++)
C[j]=R[12][j];
printf("\n\n Кодовое сообщение:");
for(j=0; j<=22; j++)
printf("%4.1d", C[j]);
//Зададим количество ошибочных бит
printf("\n Введите число ошибок: \n");
scanf("%d", &kerg);
```

```
//Зададим номера ошибочных бит
printf("\n Введите номера ошибок: \n");
for(i=0; i<kerg; i++)
{scanf("%d", &nerg);
C[nerg]=C[nerg]^1;}
tcik=0;
while (tcik<=22)
{printf("\n\n Новая итерация:");
//Делим принятое слово на генератор кода и находим синдром
for(j=0; j<=10; j++)
R[0][j]=C[12+j];
for(s=1; s<=12; s++)
{RA=g[0]*R[s-1][10];
R[s][0]=RA^C[12-s];
for(j=1; j<=10; j++)
{RB=g[j]*R[s-1][10];
R[s][j]=RB^R[s-1][j-1];}}
printf("\n Синдром:");
for(j=0; j<=10; j++)
{SINDR[j]=R[12][j];
printf("%4.1d", SINDR[j]);}
SSR=0; //Найдем вес вектора синдрома, для начала обнулив его
for(j=0; j<=10; j++)
SSR=SSR+SINDR[j];
printf("\n Вес синдрома: %4.1d ", SSR);
if (SSR<4)
{printf("\n Ошибки в младших битах ");
for(j=0; j<=10; j++)
C[j]=C[j]^SINDR[j];
for(j=0; j<tcik; j++)//сдвигаем исправленое слово на число тактов
```

```
{printf("\n Сдвигаем слово \n");
bufs=C[22];
for(i=22; i>=1; i--)
C[i]=C[i-1];
C[0]=bufs;}
tcik=23;}//объявляем об окончании цикла
else
{//предполагаем ошибку в 17 бите
printf("\n Модифицируем синдром:");
int SINDRM[11]={0,0,0,0,0,0,0,0,0,0,0};
SSR=0;
for(j=0; j<=10; j++)
{SINDRM[j]=ERRM[j]^SINDR[j];
printf("%4.1d", SINDRM[j]);
SSR=SSR+SINDRM[j];}
printf("\n Вес синдрома: %4.1d ", SSR);
if (SSR<=2)
{if (SSR==0)
{printf("\n Ошибка только в 17 бите ");
C[17]=C[17]^1;
for(j=0; j<tcik; j++)//сдвигаем исправленое слово на число тактов
{printf("\n Сдвигаем слово \n");
bufs=C[22];
for(i=22; i>=1; i--)
C[i]=C[i-1];
C[0]=bufs;}
tcik=23;}//объявляем об окончании цикла
if (SSR==1||SSR==2)
{printf("\n Ошибка в 17 и первых битах ");
C[17]=C[17]^1;
```

```
for(j=0; j<=10; j++)
C[j]=C[j]^SINDRM[j];
for(j=0; j<tcik; j++)//сдвигаем исправленое слово на число тактов
{printf("\n Сдвигаем слово \n");
bufs=C[22];
for(i=22; i>=1; i--)
C[i]=C[i-1];
C[0]=bufs;}
tcik=23;}//объявляем об окончании цикла
}
if (tcik<23)
{//предполагаем ошибку в в 16 бите
printf("\n Модифицируем синдром:");
int SINDRS[11]={0,0,0,0,0,0,0,0,0,0,0};
SSR=0;
for(j=0; j<=10; j++)
{SINDRS[j]=ERRS[j]^SINDR[j];
printf("%4.1d", SINDRS[j]);
SSR=SSR+SINDRS[j];}
printf("\n Вес синдрома: %4.1d ", SSR);
if (SSR<=2)
{if (SSR==0)
{printf("\n Ошибка только в 16 бите ");
C[16]=C[16]^1;
for(j=0; j<tcik; j++)//сдвигаем исправленое слово на число тактов
{printf("\n Сдвигаем слово \n");
bufs=C[22];
for(i=22; i>=1; i--)
C[i]=C[i-1];
C[0]=bufs;}
```

```
tcik=23;}//объявляем об окончании цикла
if (SSR==1||SSR==2)
{printf("\n Ошибка в 17 и первых битах ");
C[16]=C[16]^1;
for(j=0; j<=10; j++)
C[j]=C[j]^SINDRS[j];
for(j=0; j<tcik; j++)//сдвигаем исправленое слово на число тактов
{printf("\n Сдвигаем слово \n");
bufs=C[22];
for(i=22; i>=1; i--)
C[i]=C[i-1];
C[0]=bufs;}
tcik=23;}//объявляем об окончании цикла
}}}
if (tcik<23)
{printf("\n\n Сдвигаем слово ");
printf("\n Kodovoe soobschenie:\n");
for(j=0; j<=22; j++)
printf("%4.1d", C[j]);
int buf;
buf=C[0];
for(j=0; j<=21; j++)
C[j]=C[j+1];
C[22]=buf;
printf("\n Сдвинутое сообщение:\n");
for(j=0; j<=22; j++)
printf("%4.1d", C[j]);
tcik=tcik+1;}
printf("\n\n Итерация номер:\n");
printf("%4.1d", tcik);}
```

```
int ERO=0;//включаем счетчик ошибок
printf(" \n \n Исходные данные - Декодированные \n \n");
for(i=0;i<12;i++)
{printf("%4.1d", UGL[i]);
printf("%4.1d \n", C[i+11]);
if (UGL[i]!=C[i+11])
ERO=ERO+1;}
printf(" \n Количество ошибок: \n");
printf("%4.1d", ERO);}
```

Приложение № 5

Каскадный кодер-декодер

```
int _tmain(int argc, _TCHAR* argv[])
{int x, i, j, R[6][4], RA, RB, RC, RD, A[256], ARS[120][9], kis, mkod, PER, H[100], kerg,nerg;
int BV[17000];//-массив бинарных чисел, получаемый после всех процедур кодирования
mkod=0;//mkod - метод кодирования: 0 - только РС, 1 - РС и Х, 2 - РС и Голей
PER=0;//- флаг перемежения символов РС 0-без перемежения, 1-перемежение
//uinf[]- массив информационных байт int uinf[100]={51,52,53,54,55, 201,202,203,204,205, 1,2,3,4,5, 11,12,13,14,15, 21,22,23,24,25, 31,32,33,34,35, 41,42,43,44,45, 101,102,103,104,105, 241,242,243,244,245, 121,122,123,124,125, 191,192,193,194,195, 61,62,63,64,65, 111,112,113,114,115, 51,52,53,54,55, 201,202,203,204,205, 21,22,23,24,25, 31,32,33,34,35, 41,42,43,44,45, 101,102,103,104,105, 241,242,243,244,245};
int kolinf;//- Количество байт которые надо передать
kolinf=28;
kis=kolinf/5+1;//- количество слов РС
printf("Количество слов РС: %4.1d \n", kis);
for(i=0; i<100; i++)//Обнуляем элементы массива H[100]
H[i]=0;// H[] - вспомогательный массив информационных байт.
for(i=0; i<kolinf; i++)//заполняем H[i] из массива uinf[]
H[i]=uinf[i];
for(i=0; i<kis; i++)
{for(j=0; j<9; j++)
ARS[i][j]=0;}
for(i=0; i<=(kis*5-1); i++)
printf(" %4.1d", H[i]);
//Вводим информационные символы в кодовые слова в позиции с 5 по 8 (считая с 0)
```

```
for(i=0; i<=(kis-1); i++)
{for(j=0; j<=4; j++)
ARS[i][j+4]=H[j+5*i];}//ARS[x][y] Это массив кодовых слов РС, где x - номер слова, а y - номер символа в слове
// Формирование элементов основного поля
A[0]=1;
A[255]=0;
for(i=1;i<=254;i++)
{if (A[i-1]<128)
A[i]=A[i-1]*2;
else
A[i]=show_summ(2*A[i-1],285,0,0);}
//Конец формирования
//Кодируем кодом РС
for(x=0; x<=(kis-1); x++)
{ARS[x][0]=0;
ARS[x][1]=0;
ARS[x][2]=0;
ARS[x][3]=0;
int g[5]={116,231,216,30,1};
for(j=0; j<4; j++)
R[0][j]=ARS[x][5+j];
for(i=1; i<6; i++)
{RA=show_proizv(g[0], R[i-1][3]);
R[i][0]=show_summ(RA,ARS[x][5-i],0,0);
for(j=1; j<4; j++)
{RB=show_proizv(g[j], R[i-1][3]);
R[i][j]=show_summ(RB,R[i-1][j-1],0,0);}}
for(j=0; j<4; j++)
ARS[x][j]=R[5][j];}
```

```
int ODN[1100];
if (PER==0)
{//По строкам создадим одномерный массив ODN[]
i=0; //считывая элементы по строкам
while (i<kis)
{j=0;
while (j<9)
{ODN[j+9*i]=ARS[i][j];
j=j+1;}
i=i+1;}
printf("\n Одномерный массив: \n");
j=0;
while (j<kis*9)
{printf(" %4.1d", ODN[j]);
j=j+1;}}
else//П Е Р Е М Е Ж Е Н И Е
{//Выполним перемежение символов, создав снова одномерный массив
//считывая элементы по столбцам
i=0;
while (i<=8)
{j=0;
while (j<kis)
{ODN[j+kis*i]=ARS[j][i];
j=j+1;}
i=i+1;}
printf("\n После перемежения: \n");
j=0;
while (j<kis*9)
{printf(" %4.1d", ODN[j]);
j=j+1;}}// Перемежение закончено
```

```
//Из массива символов ODN[] создаем массив двоичных чисел BV[]
int A2;
float A1;
j=0;
while(j<kis*9)
{A1=ODN[j];
i=0;
while(i<8)
{A2=A1/2;
BV[i+j*8]=A1-A2*2;
A1=A2;
i=i+1;}
j=j+1;}
printf("\n Массив двоичных чисел: \n");
j=0;
while (j<kis*72)//так как в 1 строке 72 бита
{printf(" %4.1d", BV[j]);
j=j+1;}
int HEMB[17000];//Вспомогательный массив для временного хранения закодир слов, потом их все в BV
if (mkod==1)
{//Начинаем кодирование кодом Хемминга
int uhm[4];//Информационное слово
int vhmk[7];//Кодовое слово
for(i=0; i<kis*18; i++)//одно слово РС это сейчас 18 инф слов Хемминга
{for(j=0; j<4; j++)//берем по 4 бита из массива BV, образуем информационное слово и кодируем
uhm[j]=BV[j+i*4];
//Начинаем кодирование
bool AGH[7]={1,1,0,1,0,0,0};//Задаем по строкам порождающую матрицу
```

```
bool BGH[7]={0,1,1,0,1,0,0};
bool CGH[7]={1,1,1,0,0,1,0};
bool DGH[7]={1,0,1,0,0,0,1};
vhmk[0]=uhm[0]*AGH[0]^uhm[1]*BGH[0]^uhm[2]*CGH[0]^uhm[3]*DGH[0];
vhmk[1]=uhm[0]*AGH[1]^uhm[1]*BGH[1]^uhm[2]*CGH[1]^uhm[3]*DGH[1];
vhmk[2]=uhm[0]*AGH[2]^uhm[1]*BGH[2]^uhm[2]*CGH[2]^uhm[3]*DGH[2];
vhmk[3]=uhm[0]*AGH[3]^uhm[1]*BGH[3]^uhm[2]*CGH[3]^uhm[3]*DGH[3];
vhmk[4]=uhm[0]*AGH[4]^uhm[1]*BGH[4]^uhm[2]*CGH[4]^uhm[3]*DGH[4];
vhmk[5]=uhm[0]*AGH[5]^uhm[1]*BGH[5]^uhm[2]*CGH[5]^uhm[3]*DGH[5];
vhmk[6]=uhm[0]*AGH[6]^uhm[1]*BGH[6]^uhm[2]*CGH[6]^uhm[3]*DGH[6];
for(j=0; j<7; j++)//из кодовых слов составляем одномерный массив
HEMB[j+i*7]=vhmk[j];}
printf("\n Каскадно закодировано с кодом Хемминга: \n");
for(i=0; i<kis*18*7; i++)//kis*18*7 это всего бит кодового массива
{BV[i]=HEMB[i];
printf(" %4.1d", BV[i]);}}
//Кодирование Хемминга завершено
if (mkod==2)
{//Начинаем кодирование кодом Г О Л Е Я
int Rgl[13][11], fgl, RAgl, RBgl, RCgl, RDgl,sgl,tcikgl,kkgl,rggl,igl,jgl,bufsgl,hbgl;
int ggl[12]={1,0,1,0,1,1,1,0,0,0,1,1};//-генерирующий полином
for(fgl=0; fgl<6*kis; fgl++)// fgl - количество циклов для кодирования массива по
12 бит,
{int GHgl[23]={0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0, 0,0,0};
for(hbgl=0; hbgl<=11; hbgl++)
GHgl[hbgl+11]=BV[hbgl+fgl*12];
kkgl=23;
rggl=11;
for(jgl=0; jgl<=rggl-1; jgl++)
Rgl[0][jgl]=GHgl[kkgl-rggl+jgl];
```

```
for(sgl=1; sgl<=kkgl-rggl; sgl++)
{RAgl=ggl[0]*Rgl[sgl-1][rggl-1];
Rgl[sgl][0]=RAgl^GHgl[kkgl-rggl-sgl];
for(jgl=1; jgl<=rggl-1; jgl++)
{RBgl=ggl[jgl]*Rgl[sgl-1][rggl-1];
Rgl[sgl][jgl]=RBgl^Rgl[sgl-1][jgl-1];}}
for(jgl=0; jgl<=rggl-1; jgl++)
GHgl[jgl]=Rgl[kkgl-rggl][jgl];
for(hbgl=0; hbgl<=22; hbgl++)//Формируем массив из кодовых слов
HEMB[hbgl+fgl*23]=GHgl[hbgl];}
printf("\n Каскадно закодировано с кодом Голея: \n");
for(igl=0; igl<138*kis; igl++)
{BV[igl]=HEMB[igl];
printf("%4.1d", BV[igl]);}}
//Кодирование Г О Л Е Я  завершено
// В В О Д     О Ш И Б О К
//Зададим количество ошибочных бит
printf("\n Введи число ошибок: \n");
scanf("%d", &kerg);
//Зададим номера ошибочных бит
printf("\n Введи номер ошибки: \n");
for(i=0; i<kerg; i++)
{scanf("%d", &nerg);
BV[nerg]=BV[nerg]^1;}
//Завершен  В В О Д     О Ш И Б О К
//Д Е К О Д И Р О В А Н И Е
int udek[100];//-финальный декодированный массив байт
for(i=0; i<100; i++)//Обнуляем элементы массива udek[100]
udek[i]=0;
```

```
int HEMD[10000];//Вспомогательный массив для временного хранения декодир
//слов
if (mkod==1)
{//Начинаем декодирование кода Хемминга
int vhm[7];//Кодовое слово
for(i=0; i<kis*18; i++)//одно слово РС это сейчас 18 инф слов Хемминга
{for(j=0; j<7; j++)//берем по 7 бит из массива BV, образуем кодовое слово и
//декодируем
vhm[j]=BV[j+i*7];
//Начинаем декодирование
bool APH[3]={1,0,0};//Задаем по строкам транспонированную проверчную
матрицу
bool BPH[3]={0,1,0};
bool CPH[3]={0,0,1};
bool DPH[3]={1,1,0};
bool EPH[3]={0,1,1};
bool FPH[3]={1,1,1};
bool GPH[3]={1,0,1};
bool sihm[3];//коэффициенты синдрома
int SSHM;//значение синдрома в виде десятичного числа
sihm[0]=vhm[0]*APH[0]^vhm[1]*BPH[0]^vhm[2]*CPH[0]^vhm[3]*DPH[0]^vhm[
4]*EPH[0]^vhm[5]*FPH[0]^vhm[6]*GPH[0];
sihm[1]=vhm[0]*APH[1]^vhm[1]*BPH[1]^vhm[2]*CPH[1]^vhm[3]*DPH[1]^vhm[
4]*EPH[1]^vhm[5]*FPH[1]^vhm[6]*GPH[1];
sihm[2]=vhm[0]*APH[2]^vhm[1]*BPH[2]^vhm[2]*CPH[2]^vhm[3]*DPH[2]^vhm[
4]*EPH[2]^vhm[5]*FPH[2]^vhm[6]*GPH[2];
SSHM=sihm[0]+2*sihm[1]+4*sihm[2];//значение синдрома в виде десятичного
числа
//Исправляем только ошибки в информационных битах
if (SSHM==3)
```

```
vhm[3]=vhm[3]^1;
if (SSHM==6)
vhm[4]=vhm[4]^1;
if (SSHM==7)
vhm[5]=vhm[5]^1;
if (SSHM==5)
vhm[6]=vhm[6]^1;
for(j=0; j<4; j++)//из кодовых слов составляем одномерный массив
HEMD[j+i*4]=vhm[j+3];}
printf("\n После декодера Хемминга: \n");
for(i=0; i<kis*72; i++)//kis*72 это всего бит кодового массива PC
{BV[i]=HEMD[i];
printf(" %4.1d", BV[i]);}
}//Конец декодирования кода Хемминга
if (mkod==2)
{//Начинаем декодирование кодом Г О Л Е Я
int Rgl[13][11], SINDRgl[11],SSRgl, RAgl, RBgl, RCgl, RDgl,sgl,tcikgl,kkgl,rggl,igl,jgl,bufsgl,fgl,hbgl,ssdggl;
int ggl[12]={1,0,1,0,1,1,1,0,0,0,1,1};  //-генерирующий полином
int ERRMgl[11]={0,0,1,1,0,0,1,1,0,1,1};//вспомогат массив при ошибке в 17 элементе (считая с нуля) GH[23]
int ERRSgl[11]={0,1,1,0,0,1,1,0,1,1,0};//вспомогат массив при ошибке в 16 элементе (считая с нуля) GH[23]
int GHgl[23]={0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0, 0,0,0};//одно кодовое слово
kkgl=23;
rggl=11;
for(fgl=0; fgl<6*kis; fgl++)
{for(igl=0; igl<23; igl++)//Из массива выделяем кодированные слова и начинаем декодирование
GHgl[igl]=BV[igl+23*fgl];
```

```
tcikgl=0;
while (tcikgl<=22)//Новая итерация
{for(jgl=0; jgl<=rggl-1; jgl++)//Делим принятое слово на генератор кода и находим синдром
Rgl[0][jgl]=GHgl[kkgl-rggl+jgl];
for(sgl=1; sgl<=kkgl-rggl; sgl++)
{RAgl=ggl[0]*Rgl[sgl-1][rggl-1];
Rgl[sgl][0]=RAgl^GHgl[kkgl-rggl-sgl];
for(jgl=1; jgl<=rggl-1; jgl++)
{RBgl=ggl[jgl]*Rgl[sgl-1][rggl-1];
Rgl[sgl][jgl]=RBgl^Rgl[sgl-1][jgl-1];}}
for(jgl=0; jgl<=rggl-1; jgl++)
SINDRgl[jgl]=Rgl[kkgl-rggl][jgl];
SSRgl=0; //Найдем вес вектора синдрома, для начала обнулив его
for(jgl=0; jgl<=10; jgl++)
SSRgl=SSRgl+SINDRgl[jgl];
if (SSRgl<4)//Если ошибки в младших битах
{for(jgl=0; jgl<=10; jgl++)
GHgl[jgl]=GHgl[jgl]^SINDRgl[jgl];
for(jgl=0; jgl<tcikgl; jgl++)//сдвигаем исправленое слово на число тактов
{bufsgl=GHgl[22];
for(igl=22; igl>=1; igl--)
GHgl[igl]=GHgl[igl-1];
GHgl[0]=bufsgl;}
tcikgl=23;}//объявляем об окончании цикла
else //предполагаем ошибку в 17 элементе
{int SINDRMgl[11]={0,0,0,0,0,0,0,0,0,0,0};
SSRgl=0;
for(jgl=0; jgl<=10; jgl++)
```

```
{SINDRMgl[jgl]=ERRMgl[jgl]^SINDRgl[jgl];
SSRgl=SSRgl+SINDRMgl[jgl];}
if (SSRgl<=2)
{if (SSRgl==0)//Ошибка только в 17 бите!
{GHgl[17]=GHgl[17]^1;
for(jgl=0; jgl<tcikgl; jgl++)//сдвигаем исправленое слово на число тактов
{bufsgl=GHgl[22];
for(igl=22; igl>=1; igl--)
GHgl[igl]=GHgl[igl-1];
GHgl[0]=bufsgl;}
tcikgl=23;}//объявляем об окончании цикла
if (SSRgl==1||SSRgl==2)//ошибка в 17 и первых 11 битах
{GHgl[17]=GHgl[17]^1;
for(jgl=0; jgl<=10; jgl++)
GHgl[jgl]=GHgl[jgl]^SINDRMgl[jgl];
for(jgl=0; jgl<tcikgl; jgl++)//сдвигаем исправленое слово на число тактов
{bufsgl=GHgl[22];
for(igl=22; igl>=1; igl--)
GHgl[igl]=GHgl[igl-1];
GHgl[0]=bufsgl;}
tcikgl=23;}//объявляем об окончании цикла
}
if (tcikgl<23)
{int SINDRSgl[11]={0,0,0,0,0,0,0,0,0,0,0};//предполагаем ошибку в в 16 бите,
модифицируем синдром
SSRgl=0;
for(jgl=0; jgl<=10; jgl++)
{SINDRSgl[jgl]=ERRSgl[jgl]^SINDRgl[jgl];
SSRgl=SSRgl+SINDRSgl[jgl];}
if (SSRgl<=2)
```

```
{if (SSRgl==0)//ошибка только в 16 бите
{GHgl[16]=GHgl[16]^1;
for(jgl=0; jgl<tcikgl; jgl++)//сдвигаем исправленое слово на число тактов
{bufsgl=GHgl[22];
for(igl=22; igl>=1; igl--)
GHgl[igl]=GHgl[igl-1];
GHgl[0]=bufsgl;}
tcikgl=23;}//объявляем об окончании цикла
if (SSRgl==1||SSRgl==2)//ошибка в 16 и первых 11 битах
{GHgl[16]=GHgl[16]^1;
for(jgl=0; jgl<=10; jgl++)
GHgl[jgl]=GHgl[jgl]^SINDRSgl[jgl];
for(jgl=0; jgl<tcikgl; jgl++)//сдвигаем исправленое слово на число тактов
{bufsgl=GHgl[22];
for(igl=22; igl>=1; igl--)
GHgl[igl]=GHgl[igl-1];
GHgl[0]=bufsgl;}
tcikgl=23;}//объявляем об окончании цикла
}}}
if (tcikgl<23)//Делаем циклический сдвиг слова в сторону младших бит
{int bufgl;
bufgl=GHgl[0];
for(jgl=0; jgl<=21; jgl++)
GHgl[jgl]=GHgl[jgl+1];
GHgl[22]=bufgl;
tcikgl=tcikgl+1;}
}//завершение цикла tcik
for(jgl=11; jgl<=22; jgl++)//Формируем одномерный декодир массив
HEMD[jgl-11+fgl*12]=GHgl[jgl];}//завершен цикл декодирования
printf("\n После декодирования кода Голея: \n");
```

```
for(i=0; i<kis*72; i++)//kis*72 это всего бит кодового массива РС
{BV[i]=HEMD[i];
printf(" %4.1d", BV[i]);}}
//Декодирование Г О Л Е Я  завершено
// Превращаем массив двоичных чисел BV[] возможно содержащий ошибки
//в массив символов VHD[](каждый символ это 8 бит в виде десятичного числа)
int VHP[1080],VHD[1080];
printf("\n Массив с ошибками: \n");
i=0;
while (i<kis*9)
{VHD[i]=BV[0+8*i]*1+BV[1+8*i]*2+BV[2+8*i]*4+BV[3+8*i]*8+BV[4+8*i]*16
+BV[5+8*i]*32+BV[6+8*i]*64+BV[7+8*i]*128;
printf("%4.1d", VHD[i]);
i=i+1;}
//Р А С П Е Р Е М Е Ж Е Н И Е
//А теперь делаем расперемежение если нужно
if (PER>0)
{for(i=0; i<9; i++)
{for(j=0; j<kis; j++)
VHP[i+j*9]=VHD[j+i*kis];}
printf("\n Массив с ошибками без перемежения: \n");
for(i=0; i<kis*9; i++)
{VHD[i]=VHP[i];
printf(" %4.1d", VHD[i]);}}
//Д Е К О Д И Р О В А Н И Е   Р С
int k, q1, q2, qs, u[2], Ke;
int in, C[9];
for(in=0;in<kis;in++)
{for(i=0;i<=8;i++)//заполняем массив С элементами массива VHD КАЖДЫЙ
ЦИКЛ in=0;in<=3 ПО 9
```

```
C[i]=VHD[i+in*9];
//Вычисление синдромов
int SN, SA, SS;//SS - это сумма синдромов
int S[4];
for(j=0;j<=3;j++)
{SN=C[0];
for(i=1;i<=8;i++)
{SA=show_proizv(C[i],A[(j+1)*i]);
S[j]=show_summ(SN,SA,0,0);
SN=S[j];}}
//Конец вычисления синдрома
SS=S[0]+S[1]+S[2]+S[3];
if (SS>0)
{//Начало алгоритма Б-М
int W, Wz, q, m, Dq, Lz[3], deg;
int Vx[3]={0,1,0};
int L[4]={1,0,0,0};
q=0;
W=0;
m=-1;
while (q!=4)
{Dq=0;
for(i=0; i<=W; i++)
Dq=show_summ(Dq,show_proizv(L[i],S[q-i]),0,0);
//Цикл B
if (Dq!=0)
{for(i=0; i<=2; i++)
Lz[i]=show_summ(L[i],show_proizv(Dq,Vx[i]),0,0);
//Цикл A
if(W<q-m)
```

```
{Wz=q-m;
m=q-W;
W=Wz;
for(i=0; i<=2; i++)
Vx[i]=show_chastn(L[i],Dq);}
//Конец цикла A
for(i=0; i<=2; i++)
L[i]=Lz[i];     }
//Конец цикла B
for(i=2; i>=1; i--)
Vx[i]=Vx[i-1];
Vx[0]=0;
q=q+1;  }
//Ищем максимальную степень полинома L[i]:
for(i=0; i<=3; i++)
{if (L[i]>0)
deg=i;}
if (deg==W)
printf("Полином найден");
else printf("Полином не найден ");
//Конец алогоритма Б-М
//Ищем локаторы ошибок Рахман стр. 24 (0)
k=0;
//Подставляем в заданный полином L[] все элементы поля Галуа и ищем его
//корни:
for(i=0;i<=255;i++)
{qs=show_summ(show_proizv(L[2],show_proizv(A[i],A[i])),show_proizv(L[1],A[i])
,L[0],0);
//Если полином локатора обращается в ноль, то корень найден:
//k- счетчик корней полинома L[3]
```

```
if (qs==0)
{k=k+1;
u[k-1]=255-i;}}
int PO[4];
//Вычисляем полином величин ошибок PO состоящий из 4 коэффицентов
PO[3]=show_summ(show_proizv(S[1],L[2]),show_proizv(S[2],L[1]),show_proizv(S
[3],L[0]),0);
PO[2]=show_summ(show_proizv(S[0],L[2]),show_proizv(S[1],L[1]),show_proizv(S
[2],L[0]),0);
PO[1]=show_summ(show_proizv(S[0],L[1]),show_proizv(S[1],L[0]),0,0);
PO[0]=show_proizv(S[0],L[0]);
int Ve[2], XI[4], XP;
//XP - значение аргумента, XI[i]- массив степеней аргументов полинома x в
//квадрате, в кубе...
//Ve[2] - это сами величины ошибок (массив ошибок)
for(j=0;j<=k-1;j++)
{//Вычисляем разные степени x
XI[0]=1;
XP=A[255-u[j]];
for(i=1;i<=3;i++)
XI[i]=show_proizv(XI[i-1],XP);
Ve[j]=show_chastn(show_summ(show_proizv(PO[3],XI[3]),show_proizv(PO[2],XI[
2]),show_proizv(PO[1],XI[1]),PO[0]),L[1]);}
//Исправляем ошибки:
for(j=0;j<=k-1;j++)
{if(j<2)
{C[u[j]]=show_summ(C[u[j]],Ve[j],0,0);}}
for(i=0;i<=8;i++)//заполняем массив VHD исправленными элементами массива
//C
VHD[i+in*9]=C[i];}}
```

```
printf(" \n \n Исправленный массив \n \n");
for(i=0;i<kis*9;i++)
printf("%4.1d", VHD[i]);
for(i=0;i<kis;i++)//заполняем массив udek только информационными символами массива VHD
{for(j=0;j<5;j++)
udek[i*5+j]=VHD[i*9+j+4];}
printf(" \n \n Декодированная информация \n \n");
for(i=0;i<kolinf;i++)
printf("%4.1d", udek[i]);}
```

Приложение № 6

Кодер-декодер служебной информации

```
int _tmain(int argc, _TCHAR* argv[])
{int sbh,i,j,nerb,kerb,ibh;
int gbh[9]={1,0,0,0,1,0,1,1,1};//-генерирующий полином
int UBC[21];//Информационное сообщение
//Служебная информация содержит 5 бит содержащих номер КО (от 0 до 31),
//2 бита содержащих один из трех (РС, РС+Х, РС+Г) вариантов кодирования
//информации,
//1 бит – флаг перемежения, 1 бит – флаг ретрансляции и 12 бит содержащих
//величину объема информационных
//данных (в байтах) (до 4095 байт). Всего 21 бит.
int NKO=30;//- Номер контролируемого объекта
int mkod=2;//mkod - метод кодирования: 0 - только РС, 1 - РС и Х, 2 - РС и
//Голей
int PER=1;//- флаг перемежения символов РС 0-без перемежения, 1-
//перемежение
int FRET=1;//- флаг необходимости ретрансляции
int kolinf=2000;//- Количество байт которые надо передать
float AVSP1;
int AVSP2;
AVSP1=NKO;
for (i=0; i<5; i++)
{AVSP2=AVSP1/2;
UBC[i]=AVSP1-AVSP2*2;
AVSP1=AVSP2;}
if (mkod==0)
{UBC[5]=0;
UBC[6]=0;}
if (mkod==1)
```

```
{UBC[5]=1;
UBC[6]=0;}
if (mkod==2)
{UBC[5]=0;
UBC[6]=1;}
UBC[7]=PER;
UBC[8]=FRET;
AVSP1=kolinf;
for (i=9; i<21; i++)
{AVSP2=AVSP1/2;
UBC[i]=AVSP1-AVSP2*2;
AVSP1=AVSP2;}
//Значения полученные после декодирования
int NKOD;//с 0 по 4 бит
int mkodD;//5,6 биты
int PERD;//7 бит
int FRETD;//8 бит
int kolinfD;//с 9 по 20 бит
int DBC[45];//Кодовое сообщение
//начинаем 3 цикла кодирования
for(ibh=0; ibh<3; ibh++)
{int PRP[15]={0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0};//Кодовое слово
for(j=0; j<7; j++)
PRP[j+8]=UBC[j+7*ibh];
int Rbh[8][8], RAbh, RBbh, RCbh, RDbh;
for(j=0; j<=7; j++)
Rbh[0][j]=PRP[7+j];
for(sbh=1; sbh<=7; sbh++)
{RAbh=gbh[0]*Rbh[sbh-1][7];
Rbh[sbh][0]=RAbh^PRP[7-sbh];
```

```
for(j=1; j<=7; j++)
{RBbh=gbh[j]*Rbh[sbh-1][7];
Rbh[sbh][j]=RBbh^Rbh[sbh-1][j-1];}}
printf("\n Проверочные символы:");
for(j=0; j<8; j++)
{PRP[j]=Rbh[7][j];
printf("%4.1d ", PRP[j]);}
for(j=0; j<15; j++)//заполняем закодированными словами массив кодовых слов
DBC[j+15*ibh]=PRP[j];}
printf("\n Массив из 3 кодовых слов:");
for(j=0; j<45; j++)
printf("%4.1d ", DBC[j]);
//Кодирование БЧХ завершено
//Осуществляем перемежение
int DBCP[45];
for(i=0; i<15; i++)
{for(j=0; j<3; j++)
DBCP[i*3+j]=DBC[i+j*15];}
//Перемежение завершено
//  ВВодим ошибки:
//Зададим количество ошибочных бит
printf("\n Введи число ошибок: \n");
scanf("%d", &kerb);
//Зададим номера ошибочных бит
printf("\n Введи номер ошибки 0-44: \n");
for(i=0; i<kerb; i++)
{scanf("%d", &nerb);
DBCP[nerb]=DBCP[nerb]^1;}
//Готовимся декодировать
int UBD[21];//Декодироавнное Информационное сообщение
```

```
bool SINDR[8];//Двоичный массив синдромов
bool APR[15]={0,0,0,0,0,0,0,1,1,0,1,0,0,0,1};//Задаем 8 столбцов проверочной
//матрицы.
bool BPR[15]={0,0,0,0,0,0,1,1,0,1,0,0,0,1,0};
bool CPR[15]={0,0,0,0,0,1,1,0,1,0,0,0,1,0,0};
bool DPR[15]={0,0,0,0,1,1,0,1,0,0,0,1,0,0,0};
bool EPR[15]={0,0,0,1,1,0,1,0,0,0,1,0,0,0,0};
bool FPR[15]={0,0,1,1,0,1,0,0,0,1,0,0,0,0,0};
bool GPR[15]={0,1,1,0,1,0,0,0,1,0,0,0,0,0,0};
bool HPR[15]={1,1,0,1,0,0,0,1,0,0,0,0,0,0,0};
//Две Таблицы проверки синдромов содержащие номера ошибочных бит
int PRPR[256]={30,14,13,13,12,12,12,6,11,11,11,0,11,30,5,30,10,10,6,10,6,10,6,6,
30,10,30,30,30,4,6,30,9,30,9,9,5,2,9,30,5,30,9,30,5,5,5,30,0,
30,30,9,6,30,30,30,2,30,5,3,30,5,30,30,
0,8,2,30,8,8,30,8,30,4,30,1,2,8,4,30,4,30,30,6,8,30,30,30,
4,4,4,30,4,30,30,30,2,2,2,30,2,30,30,8,2,5,30,30,30,30,30,30,1,
2,30,30,4,30,2,30,30,30,4,30,30,30,30,30,
0,0,0,7,0,1,30,30,0,7,7,7,30,30,30,7,30,0,3,30,30,30,0,30,
1,30,7,30,3,6,30,30,1,3,0,30,30,30,3,5,30,7,30,30,0,30,30,30,3,
3,3,30,3,30,30,30,3,30,30,30,30,30,30,30,
1,1,1,30,1,0,30,30,1,30,30,30,30,30,7,30,2,1,30,4,30,30,1,30,
0,30,30,30,30,30,30,30,0,30,1,30,30,30,30,30,3,30,30,2,1,30,30,30,30,
30,30,30,30,3,30,30,30,30,30,30,30,30,30,30};
int
VTPR[256]={30,30,30,14,30,14,13,10,30,14,13,7,12,30,9,30,14,30,12,13,13,12,30,1
4,  30,11,30,30,30,8,11,30,13,30,30,14,11,8,12,30,12,30,11,30,30,14,13,30,3,
30,30,10,9,30,30,30,4,30,6,7,30,10,30,30,
1,12,9,30,14,30,30,13,30,10,30,7,5,11,6,30,11,30,30,8,10,30,30,30,
30,14,13,30,12,30,30,30,30,14,13,30,12,30,30,9,11,8,30,30,30,30,30,30,3,
10,30,30,5,30,6,30,30,30,9,30,30,30,30,30,
```

```
30,14,13,11,12,8,30,30,11,13,14,30,30,30,30,12,30,10,9,30,30,30,6,30,
4,30,10,30,5,7,30,30,2,10,9,30,30,30,6,7,30,9,30,30,5,30,30,30,30,
14,13,30,12,30,30,30,11,30,30,30,30,30,30,30,
30,14,13,30,12,8,30,30,11,30,30,30,30,30,8,30,3,10,30,7,30,30,6,30,
4,30,30,30,30,30,30,30,2,30,9,30,30,30,30,30,4,30,30,7,5,30,30,30,30,
30,30,30,30,8,30,30,30,30,30,30,30,30,30,30,30};
int SSINDR;//SSINDR - синдром в десятичном виде младший бит - SINDR[0],
старший бит - SINDR[7]
int PRN[15]={0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0};//Кодовое слово
//Осуществляем РАСПЕРЕМЕЖЕНИЕ
for(i=0; i<15; i++)
{for(j=0; j<3; j++)
DBC[i+j*15]=DBCP[i*3+j];}
//РАСПЕРЕМЕЖЕНИЕ завершено
// Декодируем
for(ibh=0; ibh<3; ibh++)
{for(j=0; j<15; j++)
PRN[j]=DBC[j+15*ibh];
//Находим синдром осуществляя умножение вектора принятого //слова на
проверочную матрицу:
SINDR[0]=PRN[0]*APR[0]^PRN[1]*APR[1]^PRN[2]*APR[2]^PRN[3]*APR[3]^P
RN[4]*APR[4]^PRN[5]*APR[5]^PRN[6]*APR[6]^PRN[7]*APR[7]^PRN[8]*APR[
8]^PRN[9]*APR[9]^PRN[10]*APR[10]^PRN[11]*APR[11]^PRN[12]*APR[12]^PR
N[13]*APR[13]^PRN[14]*APR[14];
SINDR[1]=PRN[0]*BPR[0]^PRN[1]*BPR[1]^PRN[2]*BPR[2]^PRN[3]*BPR[3]^P
RN[4]*BPR[4]^PRN[5]*BPR[5]^PRN[6]*BPR[6]^PRN[7]*BPR[7]^PRN[8]*BPR[8
]^PRN[9]*BPR[9]^PRN[10]*BPR[10]^PRN[11]*BPR[11]^PRN[12]*BPR[12]^PRN
[13]*BPR[13]^PRN[14]*BPR[14];
SINDR[2]=PRN[0]*CPR[0]^PRN[1]*CPR[1]^PRN[2]*CPR[2]^PRN[3]*CPR[3]^P
RN[4]*CPR[4]^PRN[5]*CPR[5]^PRN[6]*CPR[6]^PRN[7]*CPR[7]^PRN[8]*CPR[8
```

```
]^PRN[9]*CPR[9]^PRN[10]*CPR[10]^PRN[11]*CPR[11]^PRN[12]*CPR[12]^PRN
[13]*CPR[13]^PRN[14]*CPR[14];
SINDR[3]=PRN[0]*DPR[0]^PRN[1]*DPR[1]^PRN[2]*DPR[2]^PRN[3]*DPR[3]^P
RN[4]*DPR[4]^PRN[5]*DPR[5]^PRN[6]*DPR[6]^PRN[7]*DPR[7]^PRN[8]*DPR[
8]^PRN[9]*DPR[9]^PRN[10]*DPR[10]^PRN[11]*DPR[11]^PRN[12]*DPR[12]^PR
N[13]*DPR[13]^PRN[14]*DPR[14];
SINDR[4]=PRN[0]*EPR[0]^PRN[1]*EPR[1]^PRN[2]*EPR[2]^PRN[3]*EPR[3]^PR
N[4]*EPR[4]^PRN[5]*EPR[5]^PRN[6]*EPR[6]^PRN[7]*EPR[7]^PRN[8]*EPR[8]^
PRN[9]*EPR[9]^PRN[10]*EPR[10]^PRN[11]*EPR[11]^PRN[12]*EPR[12]^PRN[1
3]*EPR[13]^PRN[14]*EPR[14];
SINDR[5]=PRN[0]*FPR[0]^PRN[1]*FPR[1]^PRN[2]*FPR[2]^PRN[3]*FPR[3]^PR
N[4]*FPR[4]^PRN[5]*FPR[5]^PRN[6]*FPR[6]^PRN[7]*FPR[7]^PRN[8]*FPR[8]^
PRN[9]*FPR[9]^PRN[10]*FPR[10]^PRN[11]*FPR[11]^PRN[12]*FPR[12]^PRN[13
]*FPR[13]^PRN[14]*FPR[14];
SINDR[6]=PRN[0]*GPR[0]^PRN[1]*GPR[1]^PRN[2]*GPR[2]^PRN[3]*GPR[3]^P
RN[4]*GPR[4]^PRN[5]*GPR[5]^PRN[6]*GPR[6]^PRN[7]*GPR[7]^PRN[8]*GPR[
8]^PRN[9]*GPR[9]^PRN[10]*GPR[10]^PRN[11]*GPR[11]^PRN[12]*GPR[12]^PR
N[13]*GPR[13]^PRN[14]*GPR[14];
SINDR[7]=PRN[0]*HPR[0]^PRN[1]*HPR[1]^PRN[2]*HPR[2]^PRN[3]*HPR[3]^P
RN[4]*HPR[4]^PRN[5]*HPR[5]^PRN[6]*HPR[6]^PRN[7]*HPR[7]^PRN[8]*HPR[
8]^PRN[9]*HPR[9]^PRN[10]*HPR[10]^PRN[11]*HPR[11]^PRN[12]*HPR[12]^PR
N[13]*HPR[13]^PRN[14]*HPR[14];
SSINDR=128*SINDR[7]+64*SINDR[6]+32*SINDR[5]+16*SINDR[4]+8*SINDR[3
]+4*SINDR[2]+2*SINDR[1]+SINDR[0];
if (SSINDR<=0)
printf("\Ошибок нет ");
else
{//PRPR[SSINDR] - номер первой ошибки, VTPR[SSINDR] - номер второй
//ошибки
int SPOZ=PRPR[SSINDR]+VTPR[SSINDR];
```

```
printf("\n номер первой ошибки: %4.1d", PRPR[SSINDR]);
printf("\n номер второй ошибки: %4.1d", VTPR[SSINDR]);
if (SPOZ>=60)
printf("\n Ошибки не исправимы ");
else
{if (SPOZ>=30)
{printf("\n Одна ошибка ");
PRN[PRPR[SSINDR]]=PRN[PRPR[SSINDR]]^1;}
else
{printf("\n Две ошибки ");
PRN[PRPR[SSINDR]]=PRN[PRPR[SSINDR]]^1;
PRN[VTPR[SSINDR]]=PRN[VTPR[SSINDR]]^1;}}
printf("\n Исправленое слово: ");
for(i=0; i<=14; i++)
printf("%4.1d", PRN[i]);}
for(i=0; i<7; i++)
UBD[i+7*ibh]=PRN[i+8];}
printf("\n Декодировал: ");
for(i=0; i<21; i++)
printf("%4.1d", UBD[i]);
NKOD=UBD[0]+UBD[1]*2+UBD[2]*4+UBD[3]*8+UBD[4]*16;
if (UBD[5]==0||UBD[6]==0)
mkodD=0;
if (UBD[5]==1||UBD[6]==0)
mkodD=1;
if (UBD[5]==0||UBD[6]==1)
mkodD=2;
PERD=UBD[7];
FRETD=UBD[8];
```

```
kolinfD=UBD[9]+UBD[10]*2+UBD[11]*4+UBD[12]*8+UBD[13]*16+UBD[14]*3
2+UBD[15]*64+UBD[16]*128+UBD[17]*256+UBD[18]*512+UBD[19]*1024+UB
D[20]*2048;
printf("\n Номер КО: %4.1d \n", NKOD);
printf("\n Вид кодирования: %4.1d \n", mkodD);
printf("\n Флаг перемежения: %4.1d \n", PERD);
printf("\n Флаг ретрансляции: %4.1d \n", FRETD);
printf("\n Количество информации: %4.1d \n", kolinfD);}
```

Приложение № 7

Программа обмена данными с модемом

```
const char* packet_mark = "SPECTR PACKET";
const int packet_mark_length = 13;
vector<unsigned char> ConvertIntBitsToRealBits(const vector<int>& intBits)
{int resultLength = (intBits.size() / 8) + ( (intBits.size() % 8) > 0 ? 1 : 0 );
vector<unsigned char> result(resultLength);
for (unsigned int i = 0; i < intBits.size(); ++i)
{if (intBits[i] != 0)
{int realBitsByteIndex = i / 8;
int realBitsBitIndex = i % 8;
unsigned char mask = 1 << realBitsBitIndex;
result[realBitsByteIndex] = result[realBitsByteIndex] | mask;} }
return result;}
Modem::Modem(const ModemId modemId, const TCHAR* portName)
: m_modemId(modemId), m_portName(portName),
m_comPort(INVALID_HANDLE_VALUE), m_sendMode(false)
{m_comPort = ::CreateFile(
portName,
GENERIC_READ | GENERIC_WRITE,
0,
NULL,
OPEN_EXISTING,
0/*FILE_FLAG_WRITE_THROUGH*/,
0);
SerialComm::CheckWinCall(m_comPort);
SerialComm::CheckWinCall(::EscapeCommFunction(m_comPort, RESETDEV));
DCB dcb = {};
dcb.DCBlength = sizeof(DCB);
SerialComm::CheckWinCall(::GetCommState(m_comPort, &dcb));
```

```
dcb.BaudRate = CBR_4800;
dcb.fBinary = true;
dcb.fParity = false;
dcb.fOutxCtsFlow = true;
dcb.fOutxDsrFlow = false;
dcb.fNull = false;
dcb.fRtsControl = RTS_CONTROL_ENABLE;
dcb.fAbortOnError = false;
dcb.ByteSize = 8;
dcb.Parity = NOPARITY;
dcb.StopBits = ONESTOPBIT;
SerialComm::CheckWinCall(::SetCommState(m_comPort, &dcb));
COMMTIMEOUTS timeouts;
timeouts.ReadIntervalTimeout = 1000;
timeouts.ReadTotalTimeoutMultiplier = 1000;
timeouts.ReadTotalTimeoutConstant = 1000;
timeouts.WriteTotalTimeoutMultiplier = 1000;
timeouts.WriteTotalTimeoutConstant = 1000;
SerialComm::CheckWinCall(::SetCommTimeouts(m_comPort, &timeouts));
SerialComm::CheckWinCall(::SetCommMask(m_comPort, EV_RXCHAR |
EV_RXFLAG | EV_TXEMPTY | EV_CTS | EV_DSR | EV_RLSD | EV_BREAK |
EV_ERR | EV_RING | EV_PERR | EV_RX80FULL | EV_EVENT1 |
EV_EVENT2));
SerialComm::CheckWinCall(::EscapeCommFunction(m_comPort, CLRRTS));
SerialComm::CheckWinCall(::PurgeComm(m_comPort, PURGE_TXABORT |
PURGE_RXABORT | PURGE_TXCLEAR | PURGE_RXCLEAR));}
Modem::~Modem(void)
{if (m_comPort != INVALID_HANDLE_VALUE)
{HANDLE comPort = m_comPort;
m_comPort = INVALID_HANDLE_VALUE;
```

```
SerialComm::CheckWinCall(::CloseHandle(comPort));  }}
void Modem::Send(const vector<bool>& packet)
{setSendMode(true);
// Пишем «тишину».
char zero = 0;
char* pzero = &zero;
for (unsigned int i = 0; i < 64; i++)
{DWORD dwNumberOfBytesWritten;
SerialComm::CheckWinCall(::WriteFile(m_comPort, (LPCVOID)pzero,
sizeof(char), &dwNumberOfBytesWritten, NULL));}
// Пишем преамбулу для модема.
unsigned char preamble = 170; // 10101010
unsigned char* ppreamble = &preamble;
for (unsigned int i = 0; i < 8; i++)
{DWORD dwNumberOfBytesWritten;
SerialComm::CheckWinCall(::WriteFile(m_comPort, (LPCVOID)ppreamble,
sizeof(unsigned char), &dwNumberOfBytesWritten, NULL));}
// Пишем признак начала пакета.
char* mark = (char*)packet_mark;
for (unsigned int i = 0; i < 13; i++)
{DWORD dwNumberOfBytesWritten;
SerialComm::CheckWinCall(::WriteFile(m_comPort, (LPCVOID)mark, 1,
&dwNumberOfBytesWritten, NULL));
++mark;}
// Пишем два экземпляра длины пакета.
for (unsigned int j = 0; j < 2; j++)
{unsigned long length = packet.size();
unsigned char* length_byte = (unsigned char*)&length;
for (unsigned int i = 0; i < sizeof(unsigned long); i++)
{DWORD dwNumberOfBytesWritten;
```

```
SerialComm::CheckWinCall(::WriteFile(m_comPort, (LPCVOID)length_byte, 1,
&dwNumberOfBytesWritten, NULL));
++length_byte;}}
// Пишем сами данные.
for (unsigned long i = 0; i < packet.size(); i++)
{// 128 = 10000000b, для синхронизации модему нужен хотя бы 1 переход 0->1
или наоборот на 8 бит.
char value = 128 | (packet[i] ? 1 : 0);
char* pvalue = &value;
DWORD dwNumberOfBytesWritten;
SerialComm::CheckWinCall(::WriteFile(m_comPort, (LPCVOID)pvalue, 1,
&dwNumberOfBytesWritten, NULL));}}
vector<bool> Modem::Receive()
{setSendMode(false);
char* packet_mark_next_char = (char*)packet_mark;
ofstream fs("out.bin", ios::out | ios::binary | ios::trunc);
for(;;)
{char buffer[] = "               ";
LPVOID lpBuffer = &buffer;
DWORD dwNumberOfBytesRead;
SerialComm::CheckWinCall(::ReadFile(m_comPort, lpBuffer, 1,
&dwNumberOfBytesRead, NULL));
if (dwNumberOfBytesRead > 0)
{fs.write(&buffer[0], dwNumberOfBytesRead);
if (*packet_mark_next_char == buffer[0])
{// Нашли следующий ожидаемый символ признака пакета.
++packet_mark_next_char;
if (packet_mark_next_char == packet_mark + packet_mark_length)
{// Найден последний символ признака пакета.
// Читаем два экземпдяра длины пакета.
```

```
unsigned long length = readPacketLength(),
length2 = readPacketLength();
if (length != length2)
{// Не смогли корректно прочитать длину пакета. Считаем, что пакета не было.
continue;}
cout << endl << "-----------------------------------------------" << endl;
cout << "Receiving packet. Length: " << length << endl << endl;
// Читаем данные пакета.
vector<bool> result(length);
for (unsigned long i = 0; i < length; i++)
{do
{SerialComm::CheckWinCall(::ReadFile(m_comPort, lpBuffer, 1,
&dwNumberOfBytesRead, NULL));
} while (dwNumberOfBytesRead <= 0);
result[i] = (buffer[0] & 1) == 1;}
return result;}}
else
{// Не нашли следующий символ признака пакета. Считаем, что это был не
пакет.
packet_mark_next_char = (char*)packet_mark;}}}}
void Modem::setSendMode(bool sendMode)
{if (m_sendMode == sendMode)
return;
if (sendMode)
{SerialComm::CheckWinCall(::EscapeCommFunction(m_comPort, SETRTS));
DWORD evtMask;
do
{SerialComm::CheckWinCall(::WaitCommEvent(m_comPort, &evtMask, NULL));
        } while ((evtMask & EV_CTS) == 0);}
else
```

```
{SerialComm::CheckWinCall(::EscapeCommFunction(m_comPort, CLRRTS));}}
unsigned long Modem::readPacketLength()
{unsigned long length = 0;
unsigned char* length_byte = (unsigned char*)&length;
char buffer[] = "    ";
LPVOID lpBuffer = &buffer;
DWORD dwNumberOfBytesRead;
for (int i = 0; i < sizeof(unsigned long); i++)
{do
{SerialComm::CheckWinCall(::ReadFile(m_comPort, lpBuffer, 1,
&dwNumberOfBytesRead, NULL));
} while (dwNumberOfBytesRead <= 0);
*length_byte = buffer[0];
++length_byte;}
return length;}
```

Приложение № 8

Программный модуль преобразующий файл в массив данных, и массив данных в файл

```
#include "StdAfx.h"
#include <sys/types.h>
#include <sys/stat.h>
using namespace std;
vector<unsigned char> ReadFileData(const String& file_name)
{basic_ifstream<unsigned char> fs(file_name.c_str(), ios::in | ios::binary | ios::ate);
if(!fs.is_open())
throw runtime_error("Couldn't open file.");
unsigned long file_size = fs.tellg();
fs.seekg(0, ios::beg);
vector<unsigned char> result;
// Длина строки названия файла.
result.push_back((file_name.size() >> 24) & 0xFF);
result.push_back((file_name.size() >> 16) & 0xFF);
result.push_back((file_name.size() >> 8) & 0xFF);
result.push_back(file_name.size() & 0xFF);
// Само название файла.
for (unsigned int i = 0; i < file_name.size(); i++)
result.push_back(file_name.at(i));
vector<unsigned char>::size_type data_start_index = result.size();
result.resize(data_start_index + file_size, 0);
if(!fs.read(&result[data_start_index], file_size))
throw runtime_error("failed to read from htdocs/image.png");
return result;}
void WriteFileData(const vector<unsigned char>& data)
{// Длина строки названия файла.
```

```
unsigned int file_name_length = (data[0] << 24) + (data[1] << 16) + (data[2] << 8) +
data[3];
// Само название файла.
string file_name;
for (unsigned int i = 0; i < file_name_length; i++)
file_name.push_back(data[4 + i]);
basic_ofstream<unsigned char> fs(file_name.c_str(), ios::out | ios::binary |
ios::trunc);
if(!fs.is_open())
{cout << "Error opening file for save.";
return;}
fs.write(&data[4 + file_name_length], data.size() - 4 - file_name_length);}
```

Приложение № 9

УТВЕРЖДАЮ
Проректор по УМР
_____ А.А. Панфилов
30 декабря 2014 г.

АКТ

внедрения результатов диссертационного исследования Сидоренко А.А. на тему «Разработка адаптивного помехоустойчивого кодера-декодера для локальных систем телеметрии».

Результаты диссертационной работы Сидоренко А.А., представленной на соискание ученой степени кандидата технических наук по специальности 05.12.13 – Системы, сети и устройства телекоммуникаций, были использованы при выполнении курсовых работ по курсу «Методы и устройства передачи сигналов» студентами направления 210700.62 - «Инфокоммуникационные технологии и системы связи».

При выполнении курсовых проектов студентами использовались предложенные в диссертационной работе методы и алгоритмы адаптивного каскадного кодирования-декодирования нерегулярных по длине информационных сообщений и рекомендации по практическому применению устройств помехоустойчивого кодирования в различных системах телеметрии.

Заведующий кафедрой
радиотехники и радиосистем
д.т.н., профессор О.Р. Никитин

Приложение № 10

УТВЕРЖДАЮ

Заместитель директора
ООО «Миробэк»
К.Б.Подлепецкий

«19» ноября 2014 г.

АКТ

Об использовании результатов кандидатской диссертационной работы Сидоренко А.А. на тему «Разработка адаптивного помехоустойчивого кодера-декодера для локальных систем телеметрии»

Настоящим актом комиссия в составе: специалист технического отдела Львов Л.Н., начальник технического отдела Гудков П.В. подтверждает, что результаты кандидатской диссертационной работы Сидоренко А.А. на тему «Разработка адаптивного помехоустойчивого кодера-декодера для локальных систем телеметрии» были использованы ООО «Миробэк» при проектировании и строительстве сетей связи компании.

Специалист технического отдела Л.Н.Львов

Начальник технического отдела П.В.Гудков

Общество с ограниченной ответственностью "Миробэк" ОГРН 1047796224068 * Москва *