Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ
И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра радиотехнических систем (РТС)

Кологривов В.А., Цой О.

БЛОЧНО-ВЕКТОРНЫЕ АЛГЕБРАИЧЕСКИЕ КОДЫ

Учебно-методическое пособие по лабораторной и самостоятельной работе и практическим занятиям

Томск 2019

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)
Кафедра радиотехнических систем (РТС)

Утверждаю:
Зав. кафедрой РТС, профессор, д.т.н.
_____ Мелихов С.В.
«___» _____ 2019 г.

БЛОЧНО-ВЕКТОРНЫЕ АЛГЕБРАИЧЕСКИЕ КОДЫ

Учебно-методическое пособие по лабораторной и самостоятельной работе и практическим занятиям для студентов направления «Инфокоммуникационные технологии и системы связи»

Разработчики:
Доцент каф. РТС Кологривов В.А. _____
Студентка гр. 1В5 Цой О. _____

2019

Кологривов В.А., Цой О.

«Блочно-векторные алгебраические коды»: Учебно-методическое пособие по лабораторной и самостоятельной работе и практическим занятиям для студентов направления «Инфокоммуникационные технологии и системы связи» – Томск: ТУСУР. Образовательный портал, 2019.– 26 с.

Учебно-методическое пособие содержит описание модели векторной реализации кода (7, 4) с синдромным декодированием, выполненной в среде функционального моделирования ***Simulink*** программного обеспечения ***MatLab***.

В пособии приведены краткие теоретические сведения по кодированию и декодированию блочных кодов, краткая характеристика пакета ***Simulink*** системы ***MatLab***, описание лабораторного макета и используемых блоков библиотеки ***Simulink***, а также требования к экспериментальному исследованию и контрольные вопросы, ответы на которые необходимы для успешной защиты лабораторной работы.

АННОТАЦИЯ

Лабораторная работа **«Блочно-векторные алгебраические коды»** посвящена экспериментальному исследованию модели кодека с использованием пакета функционального моделирования ***Simulink*** ПО ***Matlab***.

Работа «Блочно-векторные алгебраические коды» относится к циклу лабораторных работ по разделу «Помехоустойчивое кодирование» входящему в дисциплины по направлению «Инфокоммуникационные технологии и системы связи».

В описании сформулирована цель лабораторной работы, приведены краткие теоретические сведения по блочным кодам, краткая характеристика пакета ***Simulink*** системы ***Matlab***, описание виртуального лабораторного макета и используемых блоков библиотеки ***Simulink***, а также требования к экспериментальному исследованию и контрольные вопросы, ответы на которые необходимы для успешной защиты лабораторной работы.

Оглавление

1 ЦЕЛЬ РАБОТЫ. КРАТКИЕ СВЕДЕНИЯ

Цель работы: изучение структуры и принципа блочно-векторного кодирования и синдромного декодирования алгебраических кодов с использование пакета функционального моделирования Simulink.

Теоретические сведения

В современных системах для более достоверной передачи информации используются различные виды помехоустойчивого кодирования, которые можно разделить на два больших класса: блочные и непрерывные. Сама идея помехоустойчивого кодирования заключается во введении умеренной избыточности в передаваемый цифровой поток. Вводимая избыточность делает возможным обнаружение и исправление ошибок, возникающих из-за влияния помех.

При использовании блочного кодирования в системе происходит разбиение входной информации на блоки, состоящие из k битов. Впоследствии по определенному закону происходит преобразование этих блоков в n-битовые, где n>k. Избыточно введённые биты, количество которых равно (n-k), называются битами чётности.

В данном учебно-методическом пособии предлагается рассмотреть альтернативную - векторную реализацию кода Хэмминга – кода (7, 4). Код Хэмминга является разновидностью блочных кодов: в скалярной реализации кодирование происходит блоками, состоящих из четырёх битов. В процессе кодирования передаваемый 4-х битовый блок превращается в 7-ми битовый путём введения трёх избыточных битов. Избыточность такого кода составляет 3/4.

В рассматриваемом векторном варианте реализации кода бит был заменён символом (вектором), состоящим из 3-х битов (составляющих вектора).

На рисунке 1.1 представлена структурная схема блочно-векторного кодера.

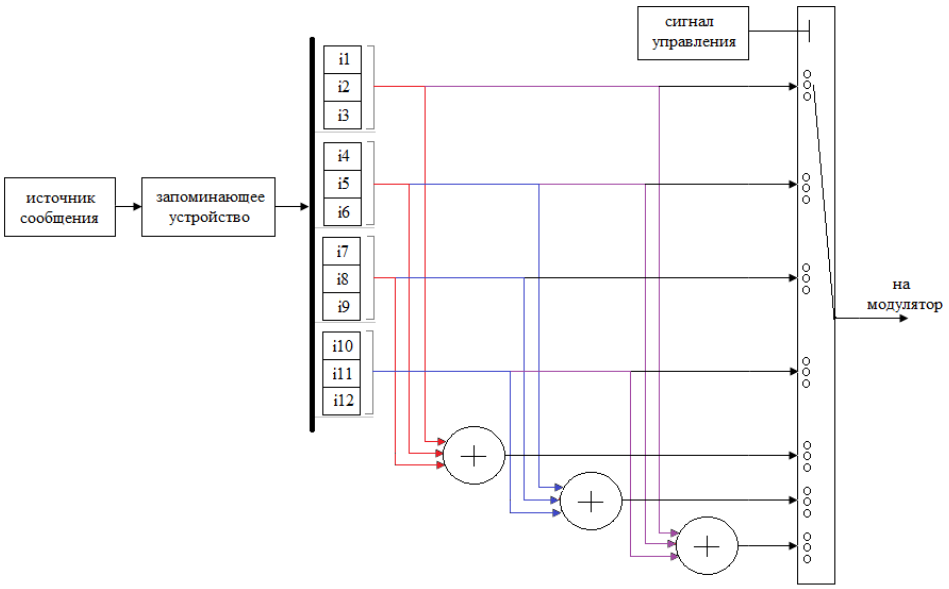


Рисунок 1.1 - Структурная схема блочно-векторного кодера

Данный кодер имеет классическую структуру, только вместо битов используются символы (векторы). В результате этого возрастает число информационных битов, а так же размер кодового символа.

Опишем принцип работы блочно-векторного кодера. Т.к. кодирование является блочным, то весь информационный поток разделяется на блоки. С помощью демультиплексора происходит формирование 3-х битовых символов. Далее с помощью операции сложения по модулю 2 определённых информационных символов (i) происходит образование проверочных символов (r), так же состоящих из 3-х битов. Данной схеме соответствуют следующие проверочные выражения:

$$r_1 = i_1 \oplus i_2 \oplus i_3;$$
$$r_2 = i_2 \oplus i_3 \oplus i_4;$$
$$r_3 = i_1 \oplus i_2 \oplus i_4.$$

Процесс кодирования осуществляется с помощью генерирующей матрицы G, которая имеют структуру:

$$G = [I_{k\times k}|P_{k\times(n-k)}], \qquad G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix},$$

где I_{kxk} – единичный блок;

$P_{kx(n-k)}$ – проверочный блок.

Приведённая выше генерирующая матрица G соответствует систематическому коду. На выходе кодера 4 символа превращаются в 7 символов по 3 бита.

В рассматриваемом кодеке для декодирования принятых символов используется синдромный способ декодирования. Структурная схема декодера представлена на рисунке 1.2.

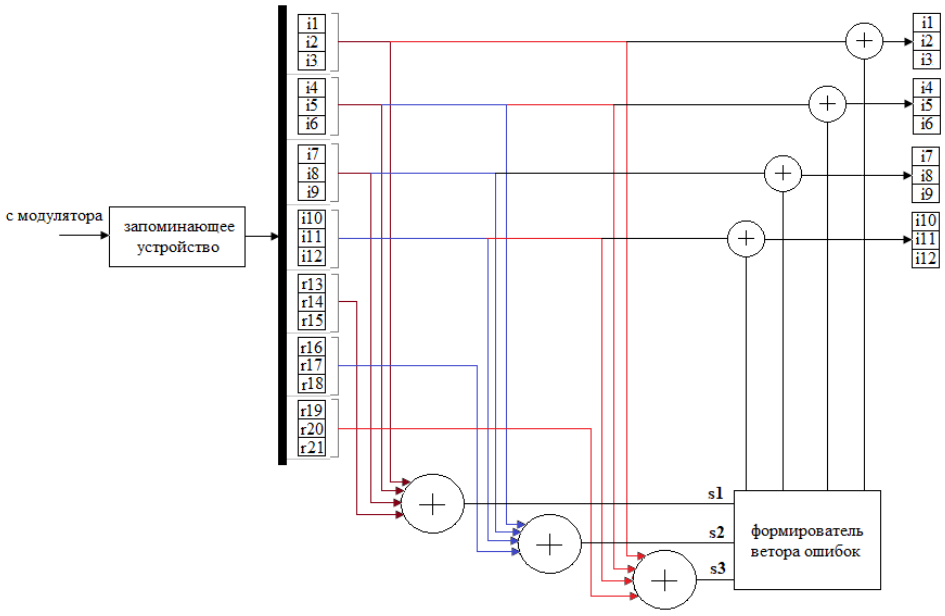


Рисунок 1.2 - Структурная схема декодера

Принятая последовательность на входе декодера представляет собой сумму по модулю 2 переданных кодовых символов и вектора ошибок:

$$\vec{z} = \vec{y} + \vec{e}, \qquad (1.1)$$

где $\vec{z}$ – принятая последовательность;

$\vec{y}$ – переданная последовательность;

$\vec{e}$ – вектор ошибок.

В основе синдромного способа декодирования лежит вычисление контрольного вектора – синдрома. Синдром является нулевым, если кодовая

комбинация была принята без ошибок и, соответственно, синдром отличен от 0, если ошибки произошли. В двоичных кодах вычисленный синдром укажет не только на наличие ошибки в принятой комбинации, но и на место её возникновения, поэтому по вычисленному синдрому можно определить вектор ошибки и произвести их исправление.

В соответствии с рисунком 2 при появлении ошибок в принятой комбинации их исправление происходит с помощью сложения по модулю 2 принятых информационных символов и соответствующего вектора ошибок.

Т.к. используемый код является двоичным, то для исправления ошибки достаточно знать место её возникновения (ведь символ может принимать лишь одно из двух значений: 0 или 1).

Сам процесс декодирования происходит с использованием проверочной матрицы H, структура которой определяется из условия ортогональности порождающей и проверочной матриц:

$$G * H^T = 0. \tag{1.2}$$

Тогда матрица H будет иметь следующую структуру:

$$H = \left[P^T_{k\times(n-k)} | I_{(n-k)\times(n-k)}\right], \qquad H = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

Для нахождения синдрома необходимо принятую кодовую комбинацию умножить на транспонированную проверочную матрицу H^T.

Иначе, когда в запоминающем устройстве накапливаются 7 символов (21 бит), начинается процесс декодирования: с помощью демультиплексора происходит разделение последовательности на символы. Затем, с помощью известных проверочных соотношений происходит декодирование: вычисление синдрома и нахождение соответствующего вектора ошибок.

Пример использования векторного кодека

Допустим, что передавались следующие 4 трёх битовых символа:

$$\vec{x} = \begin{matrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1. \\ 0 & 0 & 1 & 0 \end{matrix}$$

Запишем соответствующие генерирующую и проверочную матрицы:

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}, \qquad H = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

Произведём кодирование, используя генерирующую матрицу:

$$\vec{x} * G = \begin{matrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{matrix} * \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} = \begin{matrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1. \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{matrix}$$

В результате получим 7 трёх битовых символов.

Допустим, что при приёме ошибок не произошло. Произведём декодирование:

$$\vec{y} * H^T = \begin{matrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{matrix} * \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 0. \\ 0 & 0 & 0 \end{matrix}$$

Т.к. ошибок не было, то синдром нулевой. Теперь предположим, что произошла ошибка во всех3-х битах первого символа. Тогда:

$$\vec{z} * H^T = (\vec{y} + \vec{e}) * H^T = \begin{matrix} \overline{\mathbf{0}} & 0 & 1 & 1 & 0 & 0 & 0 \\ \overline{\mathbf{0}} & 1 & 0 & 1 & 0 & 0 & 1 \\ \overline{\mathbf{1}} & 0 & 1 & 0 & 1 & 1 & 0 \end{matrix} * \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{matrix} 1 & 0 & 1 \\ 1 & 0 & 1. \\ 1 & 0 & 1 \end{matrix}$$

Соответственно, вектор ошибки имеет вид:

$$\vec{e} = \begin{matrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0. \\ 1 & 0 & 0 & 0 \end{matrix}$$

2 ЗАПУСК И РАБОТА ПАКЕТА SIMULINK

Для запуска системы ***Simulink*** необходимо предварительно выполнить запуск системы ***MatLab***. После открытия командного окна системы ***MatLab*** нужно запустить систему ***Simulink***. Это можно сделать одним из трех способов:

-нажать кнопку (***Simulink***) на панели инструментов системы ***MatLab***;

-в строке командного окна MatLab напечатать Simulink и нажать клавишу ***Enter***;

-выполнить опцию ***Open*** в меню ***File*** и открыть файл модели (***mdl***-файл).

Последний способ предпочтителен при запуске уже готовой и отлаженной модели, когда требуется лишь провести моделирование и не нужно добавлять новые блоки в модель. При применении двух первых способов открывается окно обозревателя библиотеки блоков (***Simulink Library Browser***).

На рисунке 2.1 выведена библиотека системы ***Simulink*** (в левой части окна) и показаны ее разделы (в правой части окна). Основная библиотека системы содержит следующие разделы:

- ***Continuous*** – блоки аналоговых элементов;

- ***Discontinuous*** – блоки нелинейных элементов;

- ***Discrete*** – блоки дискретных элементов;

- ***Look-Up Tables*** – блоки таблиц;

- ***Math Operations*** – блоки элементов, определяющие математические операции;

- ***Model Verification*** – блоки проверки свойств сигнала;

- ***Model-Wide Utilities*** – раздел дополнительных утилит;

- ***Port&Subsystems*** – порты и подсистемы;

- ***Signal Attributes*** – блоки задания свойств сигналов;

- ***Signal Routing*** – блоки маршрутизации сигналов;

- ***Sinks*** – блоки приема и отображения сигналов;

- ***Sources*** – блоки источников сигнала;

- ***User-Defined Function*** – функции, определяемые пользователем.

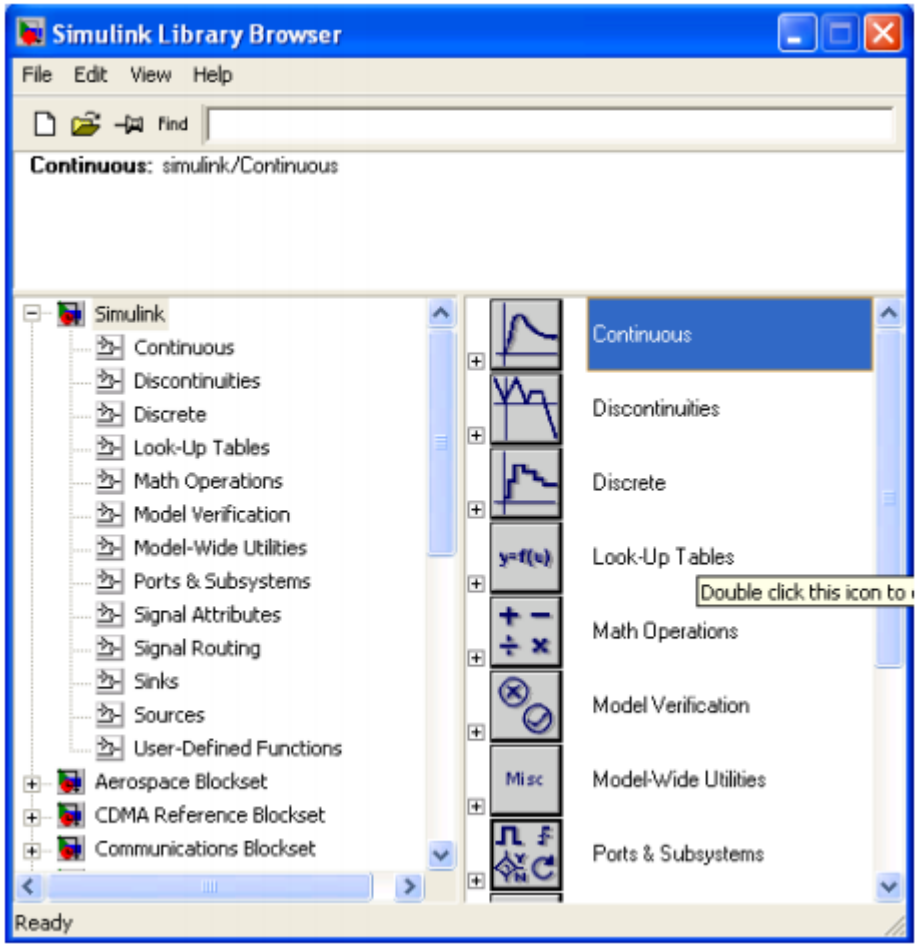


Рисунок 2.1 - Библиотека блоков ***Simulink Library Browser***

Список разделов библиотеки представлен в виде дерева, и правила работы с ним являются общими для списков такого вида: пиктограмма свернутого узла дерева содержит символ «+», а пиктограмма развернутого – символ «-».

Для того чтобы развернуть или свернуть узел дерева, достаточно щелкнуть на его пиктограмме левой клавишей мыши (ЛКМ). При выборе соответствующего раздела библиотеки его содержимое отображается в правой части окна.

При работе элементы разделов библиотек "перетаскивают" в рабочую область удержанием ЛКМ на соответствующих изображениях. Для соединения элементов достаточно указать курсором мыши на начало

соединения и затем при нажатии левой кнопки мыши протянуть соединение в его конец.

При двойном щелчке ЛКМ на выделенном блоке всплывает меню, в котором задаются параметры блоков.

Работа ***Simulink*** происходит на фоне открытого окна системы ***MatLab***, закрытие которого приведёт к выходу из ***Simulink***.

3 ОПИСАНИЕ ЛАБОРАТОРНОГО МАКЕТА

Приведем краткое описание работы блочно-векторного кода с синдромным декодированием, на основе Sim-модели представленной на рисунке 3.1.

На вход кодера приходит поток битов, которые накапливаются в блоке Buffer, затем демультиплексируются блоком DEMUX: происходит образование 3-х битовых символов. С использованием блоков XOR (суммирование по 2) сочетания различных информационных символов образуют проверочные символы.

Далее с помощью блока Multiport Switch происходит изменение скорости передачи символов, так как были добавлены символы четности.

В схеме так же имеется имитатор ошибок (рисунок 3.2), который представлен блоками Pulse generator и XOR (для добавления ошибок в битовый поток). Изменяя период импульсов в блоках Pulse generator можно имитировать, одиночные, двукратные и трёхкратные ошибки в передаваемых символах.

Подробнее разберём работу имитатора ошибок. Как видно на рисунке 3.2 имитатор ошибок содержит 3 генератора прямоугольных импульсов. Используя переключатели, можно задействовать 0, 1, 2 или 3 генератора. В параметрах каждого генератора прописаны сдвиги во времени (Phase delay), поэтому первоначально заданные выражения соответствуют ошибкам в первых 3-х битах:

- для 1-го генератора: $12 + 4/21 * 0$;

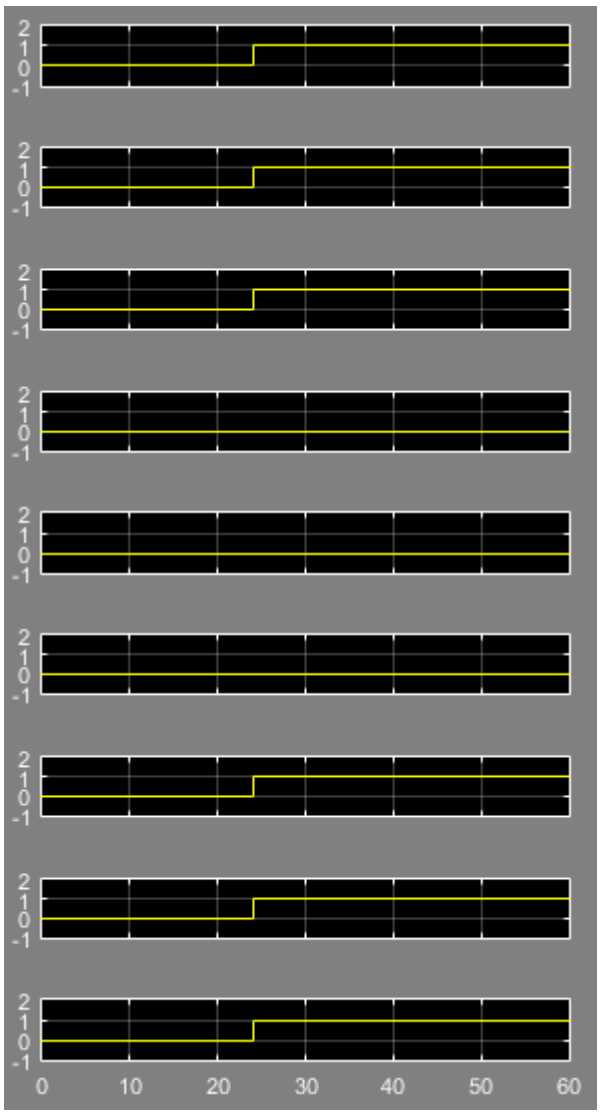
- для 2-го генератора: $12 + 4/21 + 4/21 * 0$;

- для 3-го генератора: $12 + 2 * 4/21 + 4/21 * 0$.

В случае если все 3 генератора включены в схему, вектор ошибок будет иметь следующий вид:

$$\vec{e} = \begin{matrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0. \\ 1 & 0 & 0 & 0 \end{matrix}$$

Смоделируем случай, рассмотренный ранее: ошибка произошла в первых 3-х битах. Тогда синдром, полученный путём моделирования:



Полученный синдром полностью совпадает с вычисленным ранее.

В приемнике с помощью блока Buffer происходит накопление принятых, возможно с ошибками, битов. Далее блок DEMUX исполняет роль формирователя символов: разделяет биты на символы, а затем с помощью блоков XOR происходит вычисление синдрома.

В подсистеме, представленной на рисунке 3.3, находится блок MatLab Fcn, в которой прописана функция соответствия синдромов и векторов ошибок (приложение А). Вектор ошибок, суммируясь по модулю 2 (блок XOR) с информационными символами, исправляет ошибки, которые могли произойти при передаче.

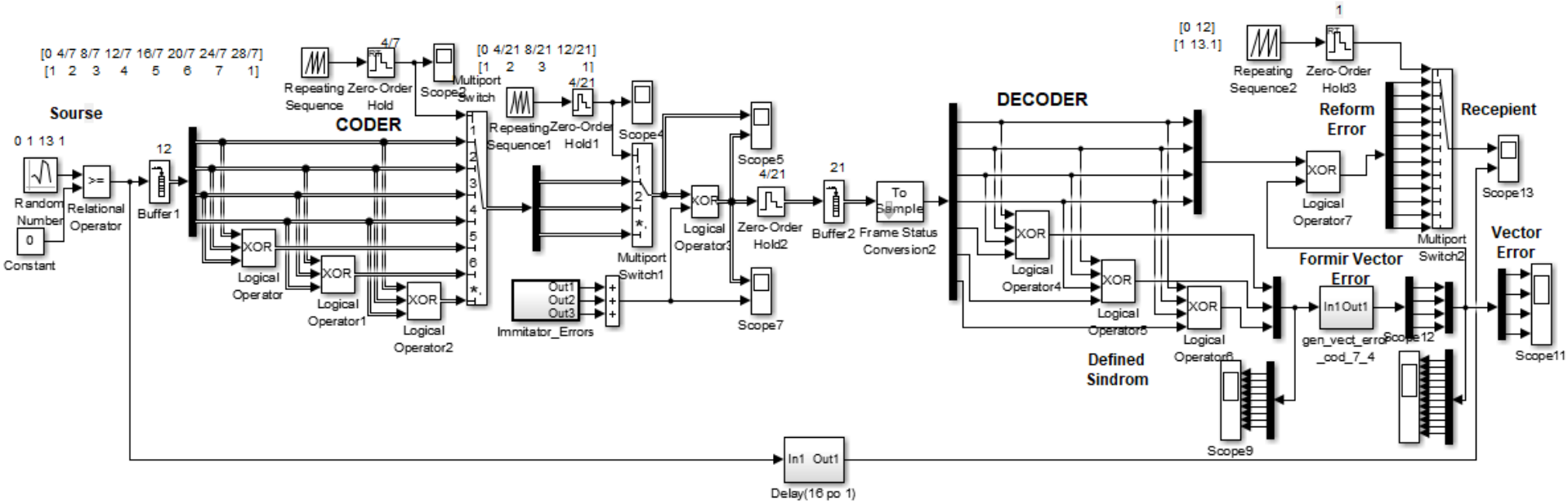


Рисунок 3.1 – Модель блочно-векторного кодека

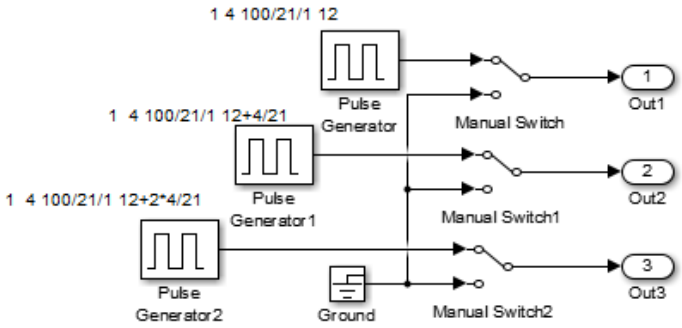


Рисунок 3.2 – Имитатор ошибок

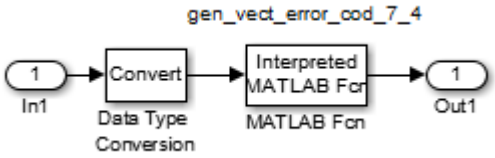


Рисунок 3.3 – Подсистема с функцией соответствия синдромов и векторов ошибок

После исправления ошибок с помощью блока Multiport Switch возвращаемся к прежней скорости передачи символов.

4 ОПИСАНИЕ ИСПОЛЬЗУЕМЫХ БЛОКОВ БИБЛИОТЕКИ SIMULINK

Ниже описаны основные блоки базовых разделов библиотеки Simulink [2], используемые в функциональной схеме.

Random number – источник случайного сигнала с нормальным распределением. Назначение: Формирование случайного сигнала с равномерным распределением уровня сигнала.

Параметры блока: Minimum – минимальный уровень сигнала;

Maximum – максимальный уровень сигнала;

Initial seed – начальное значение генератора случайного сигнала;

Sample time – такт дискретности.

Constant – источник постоянного сигнала. Назначение: задает сигнал постоянного уровня.

Параметры блока: Constant value – постоянная величина, значение которой может быть задано действительным или комплексным числом, вычисляемым выражением, вектором или массивом;

флажок Interpret vector parameters as 1-D – интерпретировать вектор как массив скаляров;

флажок Show additional parameters – показать дополнительные параметры, в нашем случае не используется.

Relational Operator – блок выполнения операций отношения. Назначение: сравнение текущих значений входных сигналов поступающих на входы.

Параметры блока: Relational Operator – тип операции отношения выбираемый из списка:

= =- тождественно равно;

~ = - не равно;

< - меньше;

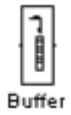
< = - меньше или равно;

> = - больше или равно;

> - больше.

Scope – блок осциллографа. Назначение: построение графиков исследуемых сигналов как функций времени. Открытие окна осциллографа производится двойным щелчком ЛКМ на пиктограмме блока. В случае векторного сигнала каждая компонента вектора отображается отдельным цветом. Настройка окна осциллографа выполняется с помощью панелей

инструментов, позволяющих: осуществить печать содержимого окна осциллографа; установить параметры, в частности: Number of axes - число входов осциллографа, Time range – отображаемый временной интервал и другие; изменить масштабы графиков; установить и сохранить настройки; перевести в плавающий режим и так далее.



Buffer – блок буферизации. Назначение: служит для буферизации сигналов. Его работу можно уподобить получению воды ищ единственного крана с помощью ведер – заполняется одно ведро, затем другое и т.д. Таким оброзом, поток данных сигнала дробится на части (фреймы) заданного размера. Размер буфера, выделяемого под задержанный сигнал, в байтах (число, кратное 8, по умолчанию 1024 байта).

Demux – блок демультиплексора. Назначение: разделение входного векторного сигнала на составляющие (последовательного представления в параллельное).

Параметры блока: Number of output – количество выходов;

Display option – способ отображения выбирается из списка: bar – вертикальный узкий прямоугольник черного цвета; none – прямоугольник с белым фоном без отображения меток входных сигналов.

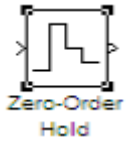
Флажок Bus Selection Mode – режим разделения векторных сигналов в шине, используется для разделения сигналов, объединенных в шину.

Repeating Sequence – источник периодического сигнала. Назначение: формирование заданного пользователем периодического сигнала.

Параметры блока: Time values – вектор значений времени;

Output values – вектор значений сигнала.

Блок выполняет линейную интерполяцию выходного сигнала для моментов времени не совпадающих со значениями, заданными вектором Time values.



Zero-Older Hold – экстраполятор нулевого порядка. Назначение: экстраполяция входного сигнала на интервале дискретизации. Блок фиксирует значение входного сигнала в начале интервала дискретизации и поддерживает на выходе это значение до окончания интервала дискретизации. Затем выходной сигнал изменяется скачком до величины входного сигнала на следующем шаге дискретизации. Параметры блока: Sample time – такт дискретности. Блок экстраполятора нулевого порядка может использоваться также для согласования работы дискретных блоков, имеющих разные такты дискретности.

Logical Operation – блок выполнения логических операций. Назначение: реализует одну из базовых логических операций.

Параметры блока: Operator - вид реализуемой логической операции – выбирается из списка: AND- логическое умножение (операция логическое И); OR- логическое сложение (операция логическое ИЛИ); NAND - операция И-НЕ; NOR - операция ИЛИ-НЕ; XOR- операция сложения по модулю 2 (операция ИСКЛЮЧАЮЩЕЕ ИЛИ); NOT - логическое отрицание (логическое НЕ);

Number of input ports - количество входных портов;

Флажок Show additional parameters – показать дополнительные параметры (в нашем случае не используется);

Флажок Require all inputs to have same data type - установить одинаковый тип входных данных;

Output data type mode- выбор типа выходных данных из списка: Boolean (двоичный), Logical (логический), Specify via dialog

(задаваемый дополнительным списком). В последнем случае появится окно списка Output data type - тип выходных данных.

Входные сигналы могут быть как действительного, так и логического типа (Boolean). Выходным сигналом блока является 1, если результат вычисления логической операции есть ИСТИНА, и 0 - если ЛОЖЬ.

Pulse generator – блок источника импульсного сигнала. Назначение: формирование сигнала в форме прямоугольных импульсов.

Параметры блока: Pulse Type – способ формирования сигнала, может принимать два значения: Time-based – по текущему времени; Sample-based – по величине такта дискретности и количеству шагов моделирования. Вид окна параметров зависит от выбранного способа формирования сигнала;

Amplitude – амплитуда;

Period – период, задается в секундах при способе Time-based или количеством тактов при способе Sample-based; Pulse width – ширина импульса, задается в процентах от периода при способе Time-based или количеством тактов при способе Sample-based;

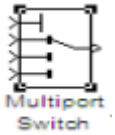
Phase delay – фазовая задержка, задается в секундах при способе Time-based или количеством тактов при способе Sample-based;

Sample time – такт дискретности; флажок Interpret vector parameters as 1 - D – интерпретировать вектор как массив скаляров.

Mux – блок мультиплексора. Назначение: объединяет входные сигналы в вектор.

Параметры блока: Number of Inputs – количество входов;

Display option – способ отображения, выбирается из списка: bar – вертикальный узкий прямоугольник черного цвета; signals – прямоугольник с белым фоном и отображением меток входных сигналов; none – прямоугольник с белым фоном без отображения меток входных сигналов.



Multiport Switch – блок многовходового переключателя. Назначение: выполняет переключение входных сигналов на выход по сигналу управления, задающему номер активного входного порта.

Параметры блока: Number of inputs – количество входов;

флажок Show additional parameters – показать дополнительные параметры, в нашем случае не используется.

Блок Multiport Switch пропускает на выход сигнал с того входного порта, номер которого равен текущему значению управляющего сигнала. Если управляющий сигнал не является сигналом целого типа, то блок Multiport Switch производит округление значения в соответствии со способом, выбранным в графе дополнительного параметра Round integer calculations toward.

5 ЭКСПЕРИМЕНТАЛЬНОЕ ЗАДАНИЕ

1. Собрать Sim-модель для исследования кодека векторного кода (7, 3) в соответствии с рисунком 3.1.
2. Выставить соответствующие параметры блоков.
3. Пронаблюдать и зафиксировать основные осциллограммы, иллюстрирующие работу синдромного декодера.
4. Промоделировать однократную, двукратную и трёхкратную ошибки, определить синдром и внести результаты в функцию соответствия синдромов и векторов ошибок.
5. Написать отчет с кратким описанием принципа работы кодека.
6. Защитить отчет.

6 КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Почему блочное кодирование называется блочным?
2. Для чего в битовый поток вносится избыточность?

3. Из какого условия происходит формирование проверочной матрицы?

4. Какую структуру имеют генерирующая и проверочная матрицы систематического кода?

5. Какова избыточность кода Хэмминга?

6. Чем определяется исправляющая способность помехоустойчивых кодов?

7. В чём заключается основное отличие скалярной и векторной реализаций кодека?

8. Что такое синдром?

9. Какую информацию несёт в себе вычисленный синдром?

10. Как происходит исправление ошибок в двоичном битовом потоке?

Список использованных источников

1. Лузин В.И., Никитин Н.П., Гадзиковский В.И. Основы формирования, передачи и приёма цифровой информации. – М.: СОЛОН-Пресс, 2014. – 316 с.

2. Черных И.В. Simulink: среда создания инженерных приложений. / под общ. ред. В.Г. Потемкина – М.: ДИАЛОГ-МИФИ, 2003.– 496 с.

3. Цой О.В., Васильева Т.В., Куулар Ч.М. Блочно-векторные алгебраические коды // Научное сообщество студентов XXI столетия. ТЕХНИЧЕСКИЕ НАУКИ: сб. ст. по мат. LXXVIII междунар. студ. науч.-практ. конф. № 6(77). URL: https://sibac.info/archive/technic/6(77).pdf (дата обращения: 05.06.2019)

Приложение А

Скрипт-файл MatLab Fcn

```
function mv_err=gen_vect_error_cod_7_4(v);
% function mv_err=gen_vect_error_cod_7_4(v);
% ФункциЯ пребразованиЯ вектора-синдрома
% в соответствующую матрицу векторов ошибок
% v- входной массив вектора-синдрома
% mv_err- матрица векторов ошибок

if all(v==[0 0 0  0 0 0  0 0 0].');
     mv_err=[0 0 0 0;
             0 0 0 0;
             0 0 0 0];
% СООТВЕТСТВИЯ СИНДРОМОВ И ВЕКТОРОВ ОДИНОЧНЫХ ОШИБОК ПИСАТЬ СЮДА

% ошибки в 5 блоке
     elseif all(v==[1 0 0  0 0 0  0 0 0].');
     mv_err=[0 0 0 0;
             0 0 0 0;
             0 0 0 0];
     elseif all(v==[0 1 0  0 0 0  0 0 0].');
     mv_err=[0 0 0 0;
             0 0 0 0;
             0 0 0 0];
     elseif all(v==[0 0 1  0 0 0  0 0 0].');
     mv_err=[0 0 0 0;
             0 0 0 0;
             0 0 0 0];
     elseif all(v==[1 1 0  0 0 0  0 0 0].');
     mv_err=[0 0 0 0;
             0 0 0 0;
             0 0 0 0];
     elseif all(v==[1 0 1  0 0 0  0 0 0].');
     mv_err=[0 0 0 0;
             0 0 0 0;
             0 0 0 0];
     elseif all(v==[0 1 1  0 0 0  0 0 0].');
     mv_err=[0 0 0 0;
             0 0 0 0;
             0 0 0 0];
         %end;
     elseif all(v==[1 1 1  0 0 0  0 0 0].');
     mv_err=[0 0 0 0;
             0 0 0 0;
             0 0 0 0];
% ошибки в 6 блоке
     elseif all(v==[0 0 0  1 0 0  0 0 0].');
     mv_err=[0 0 0 0;
             0 0 0 0;
             0 0 0 0];
     elseif all(v==[0 0 0  0 1 0  0 0 0].');
     mv_err=[0 0 0 0;
             0 0 0 0;
             0 0 0 0];
     elseif all(v==[0 0 0  0 0 1  0 0 0].');
     mv_err=[0 0 0 0;
             0 0 0 0;
             0 0 0 0];
     elseif all(v==[0 0 0  1 1 0  0 0 0].');
     mv_err=[0 0 0 0;
```

```
            0 0 0 0;
            0 0 0 0];
    elseif all(v==[0 0 0  1 0 1  0 0 0].');
    mv_err=[0 0 0 0;
            0 0 0 0;
            0 0 0 0];
    elseif all(v==[0 0 0  0 1 1  0 0 0].');
    mv_err=[0 0 0 0;
            0 0 0 0;
            0 0 0 0];
    elseif all(v==[0 0 0  1 1 1  0 0 0].');
    mv_err=[0 0 0 0;
            0 0 0 0;
            0 0 0 0];
% ошибки в 7 блоке
    elseif all(v==[0 0 0  0 0 0  1 0 0].');
    mv_err=[0 0 0 0;
            0 0 0 0;
            0 0 0 0];
    elseif all(v==[0 0 0  0 0 0  0 1 0].');
    mv_err=[0 0 0 0;
            0 0 0 0;
            0 0 0 0];
    elseif all(v==[0 0 0  0 0 0  0 0 1].');
    mv_err=[0 0 0 0;
            0 0 0 0;
            0 0 0 0];
    elseif all(v==[0 0 0  0 0 0  1 1 0].');
    mv_err=[0 0 0 0;
            0 0 0 0;
            0 0 0 0];
    elseif all(v==[0 0 0  0 0 0  1 0 1].');
    mv_err=[0 0 0 0;
            0 0 0 0;
            0 0 0 0];
    elseif all(v==[0 0 0  0 0 0  0 1 1].');
    mv_err=[0 0 0 0;
            0 0 0 0;
            0 0 0 0];
    elseif all(v==[0 0 0  0 0 0  1 1 1].');
    mv_err=[0 0 0 0;
            0 0 0 0;
            0 0 0 0];
% СООТВЕТСТВИЯ СИНДРОМОВ И ВЕКТОРОВ ДВУКРАТНЫХ ОШИБОК ПИСАТЬ СЮДА

% СООТВЕТСТВИЯ СИНДРОМОВ И ВЕКТОРОВ ТРЁХКРАТНЫХ ОШИБОК ПИСАТЬ СЮДА
    elseif all(v==[1 1 1  0 0 0  1 1 1].');
    mv_err=[1 0 0 0;
            1 0 0 0;
            1 0 0 0];
    else
     mv_err=[0 0 0 0;
            0 0 0 0;
            0 0 0 0];
end;
```