Оглавление

[Анализ методов передачи данных. 2](#_Toc417246523)

[Исследование существующих методов исправления ошибок 2](#_Toc417246524)

[Коды Соломона Рида 2](#_Toc417246525)

[Коды Боуза—Чоудхури—Хоквингема (БЧХ-коды) 4](#_Toc417246526)

[Разработка архитектуры и реализация протокола передачи данных с исправлением данных. 10](#_Toc417246527)

[Модель p a 10](#_Toc417246528)

[Модель ОПП. 10](#_Toc417246529)

[Выбор алгоритма декодирования кодов БЧХ. 11](#_Toc417246530)

[Выбор алгоритма кодирования БЧХ кодов 11](#_Toc417246531)

[Анализ результатов. 12](#_Toc417246532)

# Анализ методов передачи данных.

## Исследование существующих методов исправления ошибок

### Коды Соломона Рида

**Коды Рида — Соломона**  — недвоичные циклические коды, позволяющие исправлять ошибки в блоках данных. Элементами кодового вектора являются не биты, а группы битов (блоки).

Коды Рида — Соломона являются важным частным случаем БЧХ-кода, корни порождающего полинома которого лежат в том же поле, над которым строится код (m=1). Пусть \alpha — элемент поля \textstyle GF(q) порядка \textstyle n. Если \alpha —*примитивный* элемент, то его порядок равен q-1, то есть \alpha^{q-1}=1,\quad \alpha^i \neq 1, 0<i<q-1. Тогда нормированный полином g(x) минимальной степени над полем \textstyle GF(q), корнями которого являются d-1 подряд идущих степеней \alpha^{l_0}, \alpha^{l_0+1},...,\alpha^{l_0+d-2} элемента \alpha, является порождающим полиномом кода Рида — Соломона над полем \textstyle GF(q):

g(x) = (x - \alpha^{l_0})(x - \alpha^{l_0+1})\dots(x - \alpha^{l_0+d-2})

где l_0 — некоторое целое число (в том числе 0 и 1), с помощью которого иногда удается упростить кодер. Обычно полагается l_0 = 1. Степень многочлена \textstyle g(x) равна d-1.

Длина полученного кода n, минимальное расстояние d (минимальное расстояние d линейного кода является минимальным из всех расстояний Хемминга всех пар кодовых слов, см. Линейный код). Код содержит r=d-1=\deg (g(x)) проверочных символов, где \deg() обозначает степень полинома; число информационных символов k = n - r = n - d + 1. Таким образом \textstyle d = n - k + 1 и код Рида — Соломона является *разделимым кодом с максимальным расстоянием* (является оптимальным в смысле границы Синглтона).

Кодовый полином c(x) может быть получен из информационного полинома m(x), \deg m(x) \leqslant k-1, путем перемножения m(x) и g(x):

c(x) = m(x)g(x)

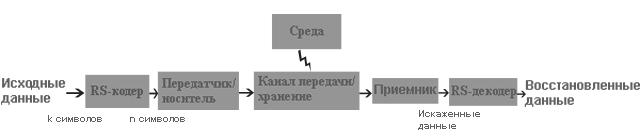
Код Рида — Соломона является одним из наиболее мощных кодов, исправляющих многократные пакеты ошибок. Применяется в каналах, где пакеты ошибок могут образовываться столь часто, что их уже нельзя исправлять с помощью кодов, исправляющих одиночные ошибки.

(q^m - 1,  q^m -2 - 2t)-код Рида — Соломона над полем \textstyle GF(q^m) с кодовым расстоянием d = 2t + 1 можно рассматривать как ((q^m - 1)m,(q^m -1 - 2t)m)-код над полем \textstyle GF(q), который может исправлять любую комбинацию ошибок, сосредоточенную в t или меньшем числе блоков из m символов. Наибольшее число блоков длины m, которые может затронуть пакет длины l_i, где l_i \leqslant mt_i - (m-1), не превосходит t_i, поэтому код, который может исправить t блоков ошибок, всегда может исправить и любую комбинацию из p пакетов общей длины l, если l+(m-1) \leqslant mt.

Кодирование с помощью кода Рида — Соломона может быть реализовано двумя способами: систематическим и несистематическим.

При несистематическом кодировании информационное слово умножается на некий неприводимый полином в поле Галуа. Полученное закодированное слово полностью отличается от исходного и для извлечения информационного слова нужно выполнить операцию декодирования и уже потом можно проверить данные на содержание ошибок. Такое кодирование требует большие затраты ресурсов только на извлечение информационных данных, при этом они могут быть без ошибок.

При систематическом кодировании к информационному блоку из k символов приписываются 2t проверочных символов, при вычислении каждого проверочного символа используются все k символов исходного блока. В этом случае нет затрат ресурсов при извлечении исходного блока, если информационное слово не содержит ошибок, но кодировщик/декодировщик должен выполнить k(n - k) операций сложения и умножения для генерации проверочных символов. Кроме того, так как все операции проводятся в поле Галуа, то сами операции кодирования/декодирования требуют много ресурсов и времени. Быстрый алгоритм декодирования, основанный на быстром преобразовании Фурье, выполняется за время порядка  O({ln( {n}) }^2).

[](https://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:%D0%A1%D1%85%D0%B5%D0%BC%D0%B0_%D0%BF%D1%80%D0%B8%D0%BC%D0%B5%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F_%D0%BA%D0%BE%D0%B4%D0%B0_%D0%A0%D0%B8%D0%B4%D0%B0-%D0%A1%D0%BE%D0%BB%D0%BE%D0%BC%D0%BE%D0%BD%D0%B0.gif)

При операции кодирования информационный полином умножается на порождающий многочлен. Умножение исходного слова S длины k на неприводимый полином при систематическом кодировании можно выполнить следующим образом:

* К исходному слову приписываются 2t нулей, получается полином \textstyle T = S x^{2t}.
* Этот полином делится на порождающий полином G, находится остаток R, \textstyle S x^{2t} = Q G + R, где Q — частное.
* Этот остаток и будет корректирующим кодом Рида — Соломона, он приписывается к исходному блоку символов. Полученное кодовое слово \textstyle C = S x^{2t} + R.

Кодировщик строится из сдвиговых регистров, сумматоров и умножителей. Сдвиговый регистр состоит из ячеек памяти, в каждой из которых находится один элемент поля Галуа.

Существует и другая **процедура** кодирования (более практичная и простая). Положим  a_{i} \in GF(q) , (i = 1,2,\ldots,k-1) , \alpha \in GF(q) - примитивный элемент поля GF(q), и пусть 
a = (a_0,a_1,\ldots,a_{k-1})
 - вектор информационных символов , а значит a(x) = a_0 + a_{1}x + \ldots + a_{k-1}x^{k-1} - информационный многочлен. Тогда вектор u = (a(1),a(\alpha),\ldots,a(\alpha^{q-2})) есть вектор кода Рида - Соломона , соответствующий информационному вектору a. Этот способ кодирования показывает , что для кода РС вообще не нужно знать порождающего многочлена и порождающей матрицы коды, достаточно знать разложение поля GF(q)по примитивному элементу \alpha и размерность кода k (длина кода в этом случае определяется как n = q - 1). Все дело в том , что за разностью n-k полностью скрывается порождающий многочлен g(x) и кодовое расстояние.

Декодировщик, работающий по авторегрессивному спектральному методу декодирования, последовательно выполняет следующие действия:

* Вычисляет синдром ошибки
* Строит полином ошибки
* Находит корни данного полинома
* Определяет характер ошибки
* Исправляет ошибки

### Коды Боуза—Чоудхури—Хоквингема (БЧХ-коды)

  БЧХ-код является циклическим кодом, который задается порождающим полиномом. Для его нахождения в случае БЧХ-кода необходимо заранее определить длину кода n  и требуемое минимальное расстояние d \leqslant n. Найти порождающий полином можно следующим образом.

Пусть ~\alpha — примитивный элемент поля ~GF(q^m) (то есть \alpha^{q^m-1}=1, \alpha^i \neq 1, i< q^m-1), пусть ~\beta=\alpha^s , — элемент поля ~GF(q^m) порядка ~n, \quad s = (q^m-1) / n . Тогда нормированный полином g(x) минимальной степени над полем GF(q), корнями которого являются ~d-1 подряд идущих степеней ~\beta^{l_0}, \beta^{l_0+1},\ldots,\beta^{l_0+d-2}элемента ~\beta для некоторого целого ~l_0 (в том числе 0 и 1), является порождающим полиномом БЧХ-кода над полем ~GF(q) с длиной n и минимальный расстоянием ~d_0 \geqslant d. Поясним почему у получившегося кода будут именно такие характеристики (длина кода ~n , минимальное расстояние ~d_0). Действительно, как показано в [1] , длина БЧХ кода равна порядку элемента ~\beta, если ~d>2 и равна порядку элемента ~\beta^{l_0}, если ~d=2, тогда, так как случай ~d=2нам не интересен (такой код не может исправлять ошибки, только обнаруживать), то длина кода будет равна порядку элемента ~\beta ,то есть равна ~n. Минимальное расстояние ~d_0 может быть больше ~d, когда корнями минимальных функций(стр.83[2]) от элементов ~\beta^{l_0}, \beta^{l_0+1},\ldots,\beta^{l_0+d-2} будут элементы расширяющие последовательность, то есть элементы ~\beta^{l_0+d-1},\beta^{l_0+d},\ldots,\beta^{l_0+d_0 - 2}.

Число проверочных символов r равно степени g(x), число информационных символов k=n-r, величина dназывается *конструктивным расстоянием* БЧХ-кода. Если n=q^m-1, то код называется *примитивным*, иначе *не примитивным*.

Так же, как и для циклического кода, кодовый полином c(x) может быть получен из информационного полинома m(x), степени не больше k-1, путём перемножения m(x) и g(x):

c(x)=m(x)g(x).

Для нахождения порождающего полинома необходимо выполнить несколько этапов:

* выбрать q, то есть поле GF(q), над которым будет построен код;
* выбрать длину n кода из условия n=(q^m-1)/s, где m,s — целые положительные числа;
* задать величину d конструктивного расстояния;

1) построить циклотомические классы элемента \beta=\alpha^s поля GF(q^m) над полем GF(q), где \alpha — примитивный элемент GF(q^m);

2) поскольку каждому такому циклотомическому классу соответствует неприводимый полином над GF(q), корнями которого являются элементы этого и только этого класса, со степенью равной количеству элементов в классе, то выбрать \beta^{l_0}, \beta^{l_0+1},\ldots, \beta^{l_0+d-2} таким образом, чтобы суммарная длина циклотомических классов была минимальна; это делается для того, чтобы при заданных характеристиках кода ~n и ~d минимизировать количество проверочных символов ~k;

3) вычислить порождающий полином g(x)=f_1(x)f_2(x)\ldots f_h(x), где f_i(x) — полином, соответствующий i-ому циклотомическому классу; или вычислить g(x), как НОК минимальных функций от элементов \beta^{l_0}, \beta^{l_0+1},\ldots,\beta^{l_0+d-2} .

При несистематическом кодировании кодовое слово получается в виде произведения информационного полинома на порождающий

c(x)=m(x)g(x).

Оно может быть реализовано при помощи перемножения полиномов.

При систематическом кодировании кодовое слово формируется в виде информационного подблока и проверочного c(x) = [s(x)\;m(x)]

Пусть информационное слово образует старшие степени кодового слова, тогда

c(x)=x^rm(x) + s(x),r=n-k

Тогда из условия c(x)=x^rm(x) + s(x)=0\mod g(x), следует

s(x)=-x^r m(x)\mod g(x)

Главной идеей в декодировании БЧХ кодов является использование элементов конечного поля для нумерации позиций кодового слова (или, эквивалентно, в порядке коэффициентов ассоциированного многочлена). **Алгоритм Берлекемпа-Мэсси** (BMA). По числу операций в конечном поле этот алгоритм обладает высокой эффективностью. BMA обычно используется для программной реализации или моделирования кодов БЧХ и кодов Рида-Соломона.

Основные методы декодирования кодов БЧХ:

* **Евклидов алгоритм** (ЕА). Из-за высокой регулярности структуры этого алгоритма его широко используют для аппаратной реализации декодеров БЧХ и кодов Рида-Соломона.

В основе этого метода лежит широко известный алгоритм Евклида по нахождению наибольшего общего делителя двух чисел (НОД), только в данном случае ищем НОК не двух чисел, а двух полиномов. Обозначим *полином значений ошибок* как \Lambda = \sigma(x) S(x), где синдромный полином равен S(x) = 1 + S_1 x + \ldots + S_{2*t_d} x^{2t_d}\quad (1). Из системы уравнений (\*) следует что  \Lambda(x) = \sigma(x) S(x) \mod x^{2t_d + 1} \quad (2). Задача по сути сводится к тому чтобы определить \Lambda(x)удовлетворяющего (2) и при этом степени не выше t_{d}. По сути такое решение и будет давать расширенный алгоритм Евклида, примененный к многочленам r_0(x) = x^d  и r_1(x)=S(x) , где d = 2t_d + 1. Если на j-ом шаге расширенный алгоритм Евклида выдает решение r_{j} = a_{j}(x) x^{2t_d + 1} + b_{j}(x) S(x), такое что  \deg[r_{j}(x)] \le t_d , то \Lambda(x) = r_j(x)  и \sigma_i(x) = b_j(x). При этом найденный полином a_j(x) дальше не принимает участия в декодировании(он ищется только как вспомогательный). Таким образом будет найден полином локаторов ошибок \sigma(x).

* **Прямое решение** (алгоритм Питерсона — Горенстейна — Цирлера, ПГЦ). Исторически это первый метод декодирования, найденный Питерсоном для двоичного случая (q=2), затем Горенстейном и Цирлером для общего случая. Этот алгоритм находит коэффициенты *многочлена локаторов ошибок* прямым решением соответствующей системы линейных уравнений. В действительности, так как сложность этого алгоритма растет как куб минимального расстояния ~d, прямой алгоритм может быть использован только для малых значений ~d.

Пусть БЧХ код над полем GF(q) длины n и с конструктивным расстоянием d задается порождающим полиномом g(x), который имеет среди своих корней элементы \beta^{l_0},\beta^{l_0+1},\ldots,\beta^{l_0+d-2} \quad \beta \in GF(q^m), \quad \beta^n=1, \quad l_0 — целое число (например 0 или 1). Тогда каждое кодовое слово обладает тем свойством, что c(\beta^{l_0-1+j}) = 0, \quad j=1,2,\ldots,d-1. Принятое слово r(x) можно записать как r(x)=c(x)+e(x), где e(x) — полином ошибок. Пусть произошло u \leqslant t = (d-1)/2 ошибок на позициях i_1,i_2,\ldots,i_u (tмаксимальное число исправляемых ошибок), значит e(x) = e_{i_1}x^{i_1}+e_{i_2}x^{i_2}+\ldots+e_{i_u}x^{i_u}, а e_{i_1}, e_{i_2},\ldots, e_{i_u} — величины ошибок.

Можно составить j-ый *синдром* S_j принятого слова r(x):

* S_j = r(\beta^{l_0-1+j}) = e(\beta^{l_0-1+j}), \quad j=1,\ldots,d-1\quad\quad (1).

Задача состоит в нахождений числа ошибок u, их позиций i_1,i_2,\ldots,i_u и их значений e_{i_1}, e_{i_2},\ldots, e_{i_u} при известных синдромах S_j.

Предположим, для начала, что u в точности равно t. Запишем (1) в виде системы нелинейных(!) уравнений в явном виде:

* 
  { \begin{cases}
  S_1 = e_{i_1} \beta^{l_0 i_1} + e_{i_2} \beta^{l_0 i_2} + \dots + e_{i_t} \beta^{l_0 i_t} \\
  S_2 = e_{i_1} \beta^{(l_0+1) i_1} + e_{i_2} \beta^{(l_0+1) i_2} + \dots + e_{i_t} \beta^{(l_0+1) i_t} \\
  \cdots \cdots \cdots \cdots \cdots \cdots \cdots \\
  S_{d-1} = e_{i_1} \beta^{(l_0+d-2) i_1} + e_{i_2} \beta^{(l_0+d-2) i_2} + \dots + e_{i_t} \beta^{(l_0+d-2) i_t} \\
  \end{cases} }
  

Обозначим через X_k = \beta^{i_k} *локатор* k-ой ошибки, а через Y_k = e_{i_k} *величину* ошибки, k=1,\ldots,t. При этом всеX_k различны, так как порядок элемента \beta равен n, и поэтому при известном X_k можно определить i_k как i_k = \log_{\beta} X_k .

* 
  { \begin{cases}
  S_1 = Y_1 X_1^{l_0} + Y_2 X_2^{l_0} + \dots + + Y_t X_t^{l_0} \\
  S_2 = Y_1 X_1^{l_0+1} + Y_2 X_2^{l_0+1} + \dots + + Y_t X_t^{l_0+1} \quad \quad \quad \quad \quad\quad(2) \\
  \cdots \cdots \cdots \cdots \cdots \cdots \cdots \\
  S_{d-1} = Y_1 X_1^{l_0+d-2} + Y_2 X_2^{l_0+d-2} + \dots + + Y_t X_t^{l_0+d-2} 
  \end{cases} }
  

Составим *полином локаторов ошибок*:

\Lambda (x) = (1-xX_1)(1-xX_2)\dots (1-xX_t) = \Lambda_t x^t + \Lambda_{t-1} x^{t-1} + \dots + \Lambda_1 x + 1

Корнями этого полинома являются элементы, обратные локаторам ошибок. Помножим обе части этого полинома наY_l X_{l}^{\vartheta+t}. Полученное равенство будет справедливо для \vartheta = l_0,l_0+1,\dots,l_0+d-1,\quad l=1,\ldots,t:

\Lambda (x) Y_l X_{l}^{\vartheta+t} = \Lambda_t x^t Y_l X_{l}^{\vartheta+t} + \Lambda_{t-1} x^{t-1} Y_l X_{l}^{\vartheta+t} + \dots + \Lambda_1 x Y_l X_{l}^{\vartheta+t} + Y_l X_{l}^{\vartheta+t}  \quad (3)

Положим x=X_l^{-1} и подставим в (3). Получится равенство, справедливое для каждого l \in {1,2,...,t} и при всех \vartheta \in { l_0,l_0+1,\dots,l_0+d-1 }:

* 0 = \Lambda_t Y_l X_{l}^{\vartheta} + \Lambda_{t-1} Y_l X_{l}^{\vartheta+1} + \dots + \Lambda_{1} Y_l X_{l}^{\vartheta+t-1} + Y_l X_{l}^{\vartheta+t}

Таким образом для каждого l можно записать свое равенство. Если их просуммировать по l, то получится равенство, справедливое для каждого \vartheta \in { l_0,l_0+1,\dots,l_0+d-1 }:

0 = \Lambda_t \sum_{l=1}^t Y_l X_{l}^{\vartheta} + \Lambda_{t-1} \sum_{l=1}^t Y_l X_{l}^{\vartheta+1} + \dots + \Lambda_{1} \sum_{l=1}^t Y_l X_{l}^{\vartheta+t-1} + \sum_{l=1}^t Y_l X_{l}^{\vartheta+t}.

.

Учитывая (2) и то, что S_{j+p} = \sum_{l=1}^t Y_l X_{l}^{l_0+j+p-1} = \sum_{l=1}^t Y_l X_{l}^{\vartheta+p}, \quad j=1,\ldots,d-1, \quad \vartheta = l_0+j-1, (то есть \vartheta меняется в тех же пределах, что и ранее) получаем систему линейных уравнений:


{ \begin{cases}
S_1 \Lambda_t + S_2 \Lambda_{t-1} + \dots + S_t \Lambda_1 = -S_{t+1} \\
S_2 \Lambda_t + S_3 \Lambda_{t-1} + \dots + S_{t+1} \Lambda_1 = -S_{t+2}   \quad \quad \quad \quad \quad\quad(4) \\
\cdots \cdots \cdots \cdots \cdots \cdots \cdots \\
S_t \Lambda_t + S_{t+1} \Lambda_{t-1} + \dots + S_{2t-1} \Lambda_1 = -S_{2t}
\end{cases} }


.

Или в матричной форме

* 
  S^{(t)} \bar\Lambda^{(t)} = -\bar s^{(t)}
  

,

где


S^{(t)}={ \left[ \begin{matrix}
S_1 & S_2 & \dots & S_t \\
S_2 & S_3 & \dots & S_{t+1} \\
\cdots & \cdots & \cdots &  \\
S_t & S_{t+1} & \dots & S_{2t-1} 
\end{matrix} \right] },  \quad \quad \quad \quad \quad\quad(5)



\bar\Lambda^{(t)} = 
{ \left[ \begin{matrix}
\Lambda_t  \\
\Lambda_{t-1}  \\
\cdots  \\
\Lambda_1
\end{matrix} \right] },  

\quad \quad 
\bar s^{(t)}
{ \left[ \begin{matrix}
S_{t+1}  \\
S_{t+2} \\
\cdots  \\
S_{2t}
\end{matrix} \right] }



Если число ошибок и в самом деле равно t, то система (4) разрешима, и можно найти значения коэффициентов \Lambda_{1},\ldots,\Lambda_{t}. Если же число u < t, то определитель матрицы S^{(t)} системы (4) будет равен 0. Это есть признак того, что количество ошибок меньше t. Поэтому необходимо составить систему (4), предполагая число ошибок равным t-1. Высчитать определитель новой матрицы S^{(t-1)} и т. д., до тех пор, пока не установим истинное число ошибок.

* **Алгоритм Берлекемпа-Мэсси** (BMA). Задается требуемая последовательность битов ~s_0, s_1, ..., s_{n-1}.

Создать массивы ~b, ~t, ~c длины ~n, задать начальные значения b_0 \leftarrow 1, c_0 \leftarrow 1, N \leftarrow 0, L \leftarrow 0, m \leftarrow -1.

Пока ~N < n:

* 1. Вычислить d \leftarrow s_N \oplus c_1s_{N-1} \oplus c_2s_{N-2} \oplus ... \oplus c_Ls_{N-L}.
  2. Если ~d=0, то текущая функция генерирует выбранный участок ~s_{N-L}, s_{N-L+1}, ..., s_Nпоследовательности; оставить функцию прежней.
  3. Если ~d \not = 0:
     + Сохранить копию массива ~c в ~t.
     + Вычислить новые значения ~c_{N-m} \leftarrow c_{N-m} \oplus b_0, c_{N-m+1} \leftarrow c_{N-m+1} \oplus b_1, ..., c_{n-1} \leftarrow c_{n-1} \oplus b_{n-N+m-1}.
     + Если 2L \leqslant N, установить значения L \leftarrow N+1-L, m \leftarrow N и скопировать ~t в ~b.
  4. N \leftarrow N+1.
* В результате массив ~c — функция обратной связи, то есть c_Ls_i \oplus c_{L-1}s_{i+1} \oplus c_{L-2}s_{i+2} \oplus ... \oplus c_0s_{i+L} = 0 для любых ~i.

# Разработка архитектуры и реализация протокола передачи данных с исправлением данных.

## Модель p a

Основным понятием, положенным в основу данной модели является плотность ошибка порядка t. Это неслучайная функция от n и t , где - среднее число ошибок на блоке длинной n с t или большим количеством ошибок. Значение плотности порядка t подчиняется следующему условию . Значения функции v(t,n) не убывают с ростом t, то есть

, если выполняются условия больше p хотя бы в несколько раз. Параметр носит название показателя группирования ошибок, подчиняется условию . При получаем канал с независимыми ошибками, а при получаем канал с “жестким” пакетированием ошибок.

На практике применяют данное соотношение для задания модели (p, a). Параметр p – характеризует вероятность ошибки символа и рассчитывается по формуле ФОРМУЛА. Параметр а – находят из уравнения .

Достоинством применение данной модели для исследования каналов связи является то, что учитывается пакетирования ошибок, возможность единообразно описать разные типы каналов передачи данных. Так для кабельных каналов значение максимально (>0.5), а для радиоканалов применяют значение в промежутке от 0.3 до 0.45. У данного способа моделирования каналов связи есть недостаток, заключающийся в неполноте и вопрос модели на уровне блоков.

## Модель ОПП.

Наблюдаемое пакетирование ошибок в каналах связи при предположении о пуассоновском характере потока можно объяснить, если считать параметр л не константой, а случайной величиной или процессом. Получающийся путем рандомизации л новый случайный процесс называют обобщенным пуассоновским . Будем считать л случайной величиной, закон распределения которой известен F(л). Тогда канал задается как поток ошибок первым способом:

Т.е формула для P(k,n) сохраняется, но осуществляется усреднение по параметру.

Поскольку вид и параметры закона распределения для реальных каналов обычно неизвестны, указанной выше формулой воспользоваться не удается.

Используя производящую функцию вероятностей:

и

Таким образом, для ОПП, зная функцию распределения интервалов между ошибками или P0(t) вычисляются вероятности P(k,t), таким образом получаем первый способ задание потока.

На рисунках представлен алгоритм генерации потока по данной модели.

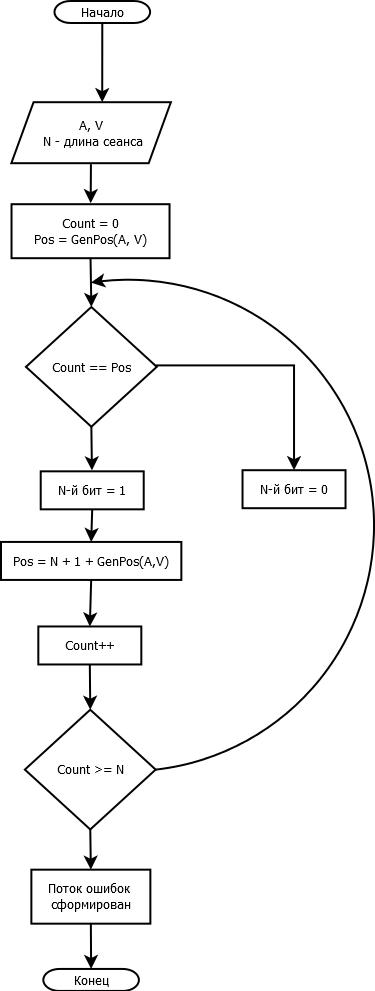


Рисунок . Алгоритм генерации потока ошибок

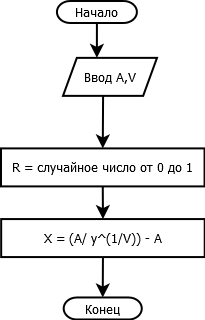


Рисунок . Алгоритм функции генерации позиции ошибки в потоке битов.

## Выбор алгоритма декодирования кодов БЧХ.

Из существующих методов декодирования кодов БЧХ был выбран ВМА. По числу операций в конечном поле этот алгоритм обладает высокой эффективностью. Подробное описание данного алгоритма приведено в методах декодирования кодов БЧХ.

## Выбор алгоритма кодирования БЧХ кодов

Для реализации кодирования было выбрано несистематическое кодирование. Данный метод кодирования позволяет реализовать модель наиболее простым способом.

# Анализ результатов.