Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Пермский национальный исследовательский политехнический университет»
Кафедра «Автоматика и телемеханика»

Курсовой проект

по дисциплине

«Цифровые системы передачи»

по теме

# «Проектирование узлов первичного мультиплексора»

Вариант 5: Выделитель метки цикла потока E1

Выполнил: студент гр. ТК-18-1б

Казаков А.А.

Проверил: ст. пр. кафедры АТ
Гаврилов А.В.

Пермь, 2021

**Содержание:**

Задание на проектирование…… 3
Введение…… 4
1. Теоретическая часть…… 5
1.1. Структура потока Е1……5
1.2. Приемник синхросигнала со скользящим поиском……..7
2. Разработка программного алгоритма…….. 12
2.1. Разработка алгоритма формирования потока Е1…….. 12
2.2. Разработка алгоритма распознавателя метки цикла потока Е1….. 15
Заключение…… 18
Список использованной литературы……... 19
Приложение……... 20

## Задание на проектирование

Целью курсового проекта является разработка отдельных узлов первичного мультиплексора ИКМ-30.

Целью данного варианта (вариант 5) курсового проекта является разработка выделителя метки цикла потока Е1.

Задачи, которые нужно решить в процессе выполнения курсового проекта:

- Изучить структуру потока Е1.
- Изучить теорию по разрабатываемому блоку первичного мультиплексора, в данном варианте выделителю метки цикла потока Е1;
- Разработать программный алгоритм, реализующий формирование потока Е1.
- Разработать программный алгоритм, реализующий функционал блока выделителя метки цикла потока Е1.

## Введение

Цикловая синхронизация обеспечивает правильное разделение и декодирование кодовых групп циклового сигнала и распределение декодированных отсчетов по соответствующим каналам в приемной части аппаратуры. [1]

Важной составляющей цикловой синхронизации является метка циклового синхросигнала.

Номером варианта курсового проекта мы разработаем выделитель метки циклового синхросигнала, который соответствует моменту прохождения соответствующей метки.

## 1. Теоретическая часть

### 1.1. Структура потока E1

Рассмотрим структуру кадра передачи ЦСП ИКМ-30 (рис. 1). Структура потока E1 определена в рекомендации ITU-T G.704. Данный поток называется первичным цифровым потоком и организуется объединением 30-ти информационных ОЦК [1].

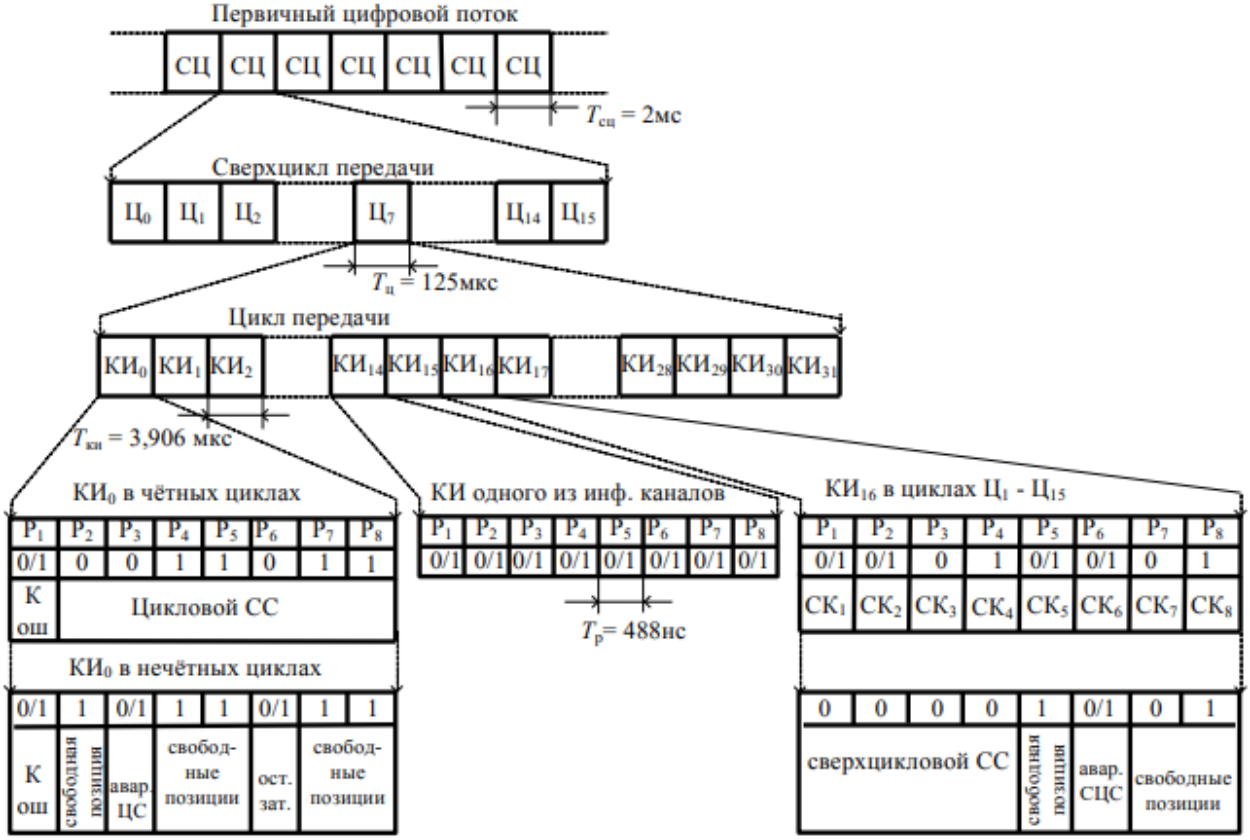


Рис. 1. Структура потока E1

Линейный сигнал системы построен на основе сверхциклов, циклов, канальных и тактовых интервалов, как это показано на рис. 2.15 (обозначение 0/1 соответствует передаче в данном тактовом интервале случайного сигнала). Сверхцикл передачи (СЦ) соответствует минимальному интервалу времени, за который передаётся один отсчёт каждого из 30 сигнальных каналов (СК) и каналов передачи аварийной сигнализации (потери сверхцикловой или цикловой синхронизации). Длительность СЦ Тсц=2мс. Сверхцикл состоит из 16 циклов передачи (с Ц0 по Ц15). Длительность цикла Тц=125мкс и соответствует интервалу дискретизации канала ТЧ с частотой 8

кГц. Каждый цикл подразделяется 62 на 32 канальных интервала длительностью Тки=3,906 мкс. Канальные интервалы КИ1–КИ15, КИ17–КИ31 отведены под передачу информационных сигналов. КИ0 и КИ16 – под передачу служебной информации. Каждый канальный интервал состоит из восьми интервалов разрядов (Р1–Р8) длительностью по Тр=488нс. Половина разрядного интервала может быть занята прямоугольным импульсом длительностью Ти=244нс при передаче в данном разряде единицы (при передаче нуля импульс в разрядном интервале отсутствует). Интервалы КИ0 в четных циклах предназначаются для передачи циклового синхросигнала (ЦСС), имеющего вид 0011011 и занимающего интервалы Р2–Р8. В интервале Р1 всех циклов передается информация контроля ошибок передачи (К. ош.). В нечетных циклах интервалы Р3 и Р6 КИ0 используются для передачи информации о потере цикловой синхронизации (Авар. ЦС – LOF) и снижении остаточного затухания каналов до значения, при котором в них может возникнуть самовозбуждение (Ост. зат.). Интервалы Р4, Р5, Р7 и Р8 являются свободными, их занимают единичными сигналами для улучшения работы выделителей тактовой частоты. В интервале КИ16 нулевого цикла (Ц0) передается сверхцикловой синхросигнал вида 0000 (Р1–Р4), а также сигнал о потере сверхцикловой синхронизации (Р6 – Авар. СЦС – LOM). Остальные три разрядных интервала свободны. В канальном интервале КИ16 остальных циклов (Ц1–Ц15) передаются сигналы служебных каналов СК1 и СК2, причем в Ц1 передаются СК для 1-го и 16-го каналов ТЧ, в Ц2 – для 2-го и 17-го и т.д. Интервалы Р3, Р4, Р6 и Р7 свободны.

С точки зрения передачи телефонный канал является 8-битным отсчётом. Полезная нагрузка – разговор двух абонентов. Кроме того, передаётся служебная информация (набор номера, отбой и т.п.) – сигналы управления и взаимодействия (СУВ). Для передачи таких сигналов достаточно повторения их 1 раз в 15 циклов, при этом каждый СУВ будет занимать 4 бита (СУВ для какого-то конкретного канала). Для этих целей был выбран канальный интервал №16 (КИ16). В один канал помещаются

СУВ для двух телефонных каналов. Т.к. всего 30 каналов, за один разговор используется два канала, то цикл нужно повторить 15 раз, следовательно, с Ц1 по Ц15 передаём всю информацию о СУВ. Таким образом, необходимо определить номер цикла. Для этих целей нулевой цикл содержит сверхцикловой СС («0000» в 1-х четырёх байтах –MFAS). В 6-м бите передаётся потеря сверхцикла (LOM) [1].

### 1.1. Приемник синхросигнала со скользящим поиском

Если рассмотреть принцип работы приемника синхросигнала со скользящим поиском (рис. 2.), который выполняет следующие основные функции: установление синхронизма после включения системы в работу; контроль за синхронным состоянием системы в процессе работы; обнаружение сбоя синхронизма; восстановление состояния синхронизма после каждого сбоя; то в нем можно выделить три основных функциональных узла. Ими являются опознаватель, анализатор и решающее устройство.

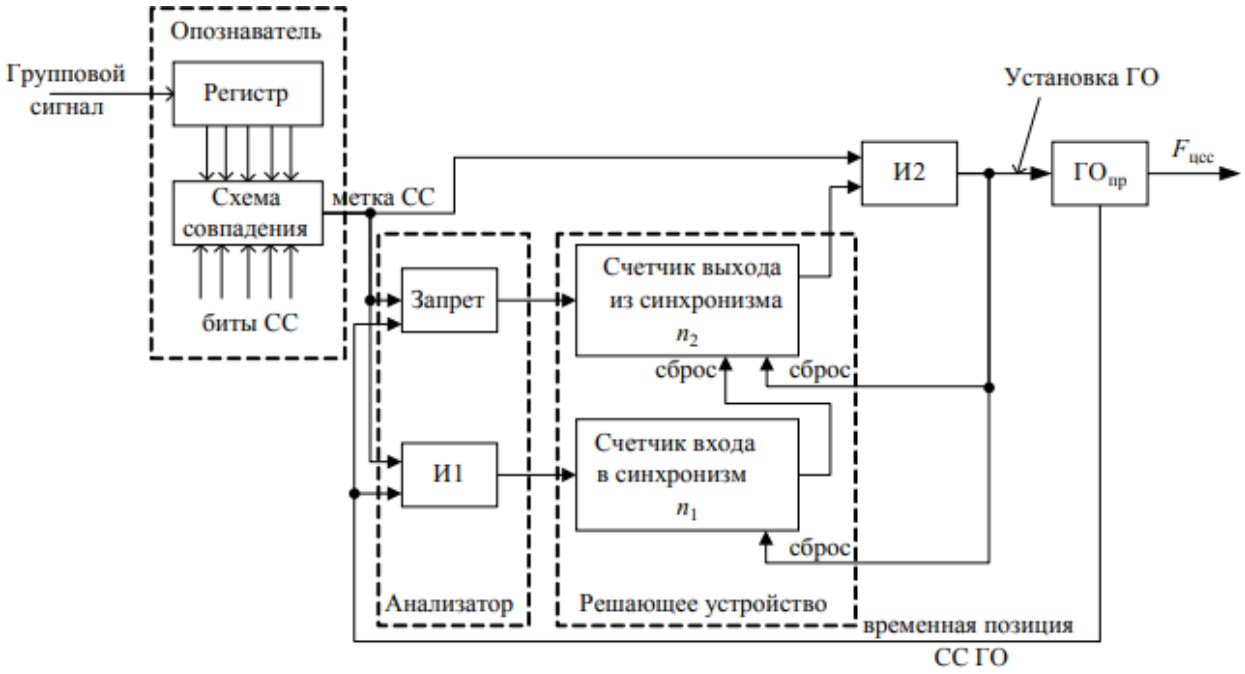


Рис. 2. Приемник синхросигнала со скользящим поиском

Опознаватель содержит регистр сдвига, число разрядов в котором совпадает с числом символов в синхросигнале, а именно 7, и дешифратор (Дш), настроенный на дешифрацию синхросигнала заданной структуры.

Как только в регистре сдвига, на вход которого поступает групповой цифровой сигнал, оказывается записанной кодовая комбинация, совпадающая по структуре с принятой структурой синхросигнала, который для первичного мультиплексора ИКМ-30, выделитель метки циклового синхросигнала для которого мы разрабатываем, равен следующей бинарной комбинации: 0011011, на выходе опознавателя (выделителя метки цикла) появляется импульс.

Анализатор с помощью контрольного сигнала (временная позиция СС ГО), поступающего от $ГО_{пр}$, проверяет соответствие момента появления импульса на выходе опознавателя ожидаемому моменту появления синхросигнала, т.е. осуществляется проверка по периоду следования и времени появления синхросигнала.

Появление импульса на выходе схемы запрета означает отсутствие синхросигнала (сигнала с выхода схемы совпадения) в момент поступления контрольного импульса от $ГО_{пр}$, а появление импульса на выходе схемы И1 означает совпадение по времени синхросигнала и контрольного сигнала от $ГО_{пр}$.

Решающее устройство оценивает выходные сигналы анализатора по определенному критерию, принимает решение о наличии или отсутствии синхронизма и управляет работой $ГО_{пр}$ в процессе вхождения в синхронизм. Решающее устройство содержит накопитель по выходу из синхронизма и накопитель по входу в синхронизм, представляющие собой двоичные счетчики со сбросом (счетчик по входу в синхронизм $n_1$ и счетчик выхода из синхронизма $n_2$).

Накопитель по входу в синхронизм, вход которого соединен с выходом схемы И1, обеспечивает защиту ПрСС от ложного вхождения в синхронизм в режиме поиска синхросигнала, когда на вход опознавателя поступают случайные комбинации цифрового группового сигнала, совпадающие по структуре с синхросигналом. Обычно емкость накопителя по входу в синхронизм $n_1$ составляет 2–3 разряда.

Накопитель по выходу из синхронизма, вход которого соединен с выходом схемы запрета анализатора, обеспечивает защиту от ложного выхода из состояния синхронизма, когда из-за ошибок в линейном тракте или по другим причинам происходит кратковременное изменение структуры синхросигнала. Обычно емкость накопителя по выходу из синхронизма $n_2$ составляет 4–6 разрядов.

Рассмотрим работу приемника синхросигнала. Если система находится в режиме синхронизма, то накопитель по входу в синхронизм будет заполнен, поскольку на выходе схемы И1 регулярно появляются импульсы, подтверждающие совпадение моментов поступления импульсов с выхода опознавателя и контрольных импульсов от $ГО_{пр}$. Накопитель по выходу из синхронизма опустошается. Импульсы на выходе опознавателя, соответствующие случайным комбинациям со структурой, аналогичной структуре синхросигнала, не влияют на работу ПрСС, т.к. не совпадают по времени с контрольными импульсами от $ГО_{пр}$.

Если, например, в результате ошибок в одном из циклов будет искажен синхросигнал, на выходе опознавателя в нужный момент импульс не появится, в результате чего с выхода схемы запрета в накопитель по выходу из синхронизма поступит импульс. Однако схема остается в прежнем состоянии, поддерживая ранее установленное состояние синхронизма. Только в том случае, если будут искажены $n_2$ синхросигналов подряд, т.е. когда полностью заполнится накопитель по выходу из синхронизма, будет принято решение о выходе системы из состояния синхронизма. При этом если накопитель по входу в синхронизм будет заполнен раньше накопителя по выходу из синхронизма, то последний будет сбрасываться в исходное нулевое положение. Таким образом, обеспечивается защита от ложного выхода из синхронизма при кратковременных искажениях синхросигнала.

При длительном нарушении синхронизма накопитель по выходу из синхронизма оказывается заполненным и принимается решение о действительном выходе системы из состояния синхронизма. Начинается

поиск нового состояния синхронизма. В этом случае первый же импульс от опознавателя через открытый элемент И2 переводит $ГО_{пр}$ и накопитель по входу в синхронизм в исходное нулевое состояние, а накопитель по выходу из синхронизма – в состояние, соответствующее $(n_2 – 1)$-му импульсу, т.е. уменьшает его содержимое на 1. Если в следующем цикле моменты появления импульса на выходе опознавателя и импульса от $ГО_{пр}$ не совпадают (это означает, что синхрогруппа оказалась ложной), то вновь заполняется накопитель по выходу из синхронизма, открывается схема И2 и очередной импульс от опознавателя вновь устанавливает $ГО_{пр}$ и накопители в указанное ранее состояние. Таким образом, обеспечивается защита от ложного установления синхронизма. Этот процесс продолжается до тех пор, пока на выходе опознавателя не появляется импульс, соответствующий истинному синхросигналу. В этом случае через $n_1$ циклов заполняется накопитель по входу в синхронизм, сбрасывается в нулевое состояние накопитель по входу в синхронизм, сбрасывается в нулевое состояние накопитель по выходу из синхронизма, схема И2 закрывается, т.е. устанавливается новое состояние синхронизма.

Из анализа работы ПрСС следует, что процесс восстановления синхронизма содержит три последовательно выполняемых этапа: обнаружение выхода из синхронизма, поиск синхросигнала и подтверждение нового состояния синхронизма. Соответственно, время восстановления синхронизма $t_в = t_{н\ вых} + t_п + t_{н\ вх}$, где $t_{н\ вых}$ – время заполнения накопителя по выходу из синхронизма; $t_п$ – время поиска синхросигнала; $t_{н\ вх}$ – время заполнения накопителя по входу в синхронизм.

Недостатки рассмотренного способа построения ПрСС заключаются в следующем.

Во-первых, поиск синхросигнала начинается только после окончания процесса заполнения накопителя по выходу из синхронизма, т.е. через $t_{н\ вых}$, что приводит к увеличению времени восстановления синхронизма $t_в$.

Во-вторых, емкости накопителей по входу в синхронизм и выводу из синхронизма ($n_1$ и $n_2$) фиксированы, что не позволяет добиваться оптимальных соотношений между временем восстановления синхронизма и помехоустойчивостью. Если вероятность ошибок в линейном тракте увеличивается (по сравнению с расчетной величиной), то время удержания состояния синхронизма оказывается меньше требуемого. Однако при уменьшении вероятности ошибки возникает запас по времени удержания синхронизма, что свидетельствует о необоснованном увеличении времени восстановления синхронизма.

Первый недостаток может быть устранен, если процессы накопления по выходу из синхронизма и поиска синхросигнала осуществлять параллельно. Для этого схему ПрСС, приведенную на рис. 1, необходимо дополнить схемой поиска синхросигнала, содержащей собственные анализатор и решающее устройство. Эта схема начинает работать при появлении первого же импульса на входе накопителя по выходу из синхронизма, т.е. не дожидаясь его заполнения, и осуществляет поиск нового состояния синхронизма. Генераторное оборудование будет сохранять предыдущее состояние до тех пор, пока не будет зафиксировано новое состояние синхронизма.

Второй недостаток может быть устранен, если емкости накопителей ($n_1$ и $n_2$) сделать величинами переменными, зависящими от вероятности ошибок в линейном тракте. При понижении вероятности ошибок уменьшается емкость накопителя по выходу из синхронизма, а при увеличении вероятности ошибок уменьшается емкость накопителя по входу в синхронизм. Такие приемники синхросигнала называются адаптивными и широко применяются в высокоскоростных отечественных ЦСП. [1]

## 2. Разработка программного алгоритма

### 2.1. Разработка алгоритма формирования потока E1

Основной функцией формирующей поток, является функция *generateSequence()*. Внутри мы формируем последовательность из нулей, имитирующую работу устройства, которая вставляется в начало и конец формируемого функцией потока E1. При вызове функции она возвращает поток E1 сформированный в цикле внутри функции. Количество итераций цикла показывает количество сверхциклов в конечном потоке.

```
function generateSequence() {
    let stubBits = [0, 0, 0, 0, 0, 0, 0, 0]
    let outputCode = [...stubBits]

    for (let index = 0; index < 12; index++) {
        ////цикл формирования сверхциклов ( возьмем для примера 12)
        outputCode = [...outputCode, ...generateSuperCycle(index)]
    }

    return outputCode.concat(stubBits)
}
```

Внутри каждой итерации цикла мы вызываем функцию формирования сверхцикла *generateSuperCycle(index).* Аргументом данной функции является индекс текущего сверхцикла, он нам понадобится позже для имитации потери цикловой СС.

```
function generateSuperCycle(iSuperCycle) {
    let outputCode = []
    for (let index = 0; index <= 15; index++) {
        // сверхцикл состоит из 16 циклов
        outputCode = [...outputCode, ...generateCycle(index, iSuperCycle)]
    }
    return outputCode
}
```

В данной функции так же, как и в предыдущем случае, на каждой итерации цикла мы вызываем функцию, но в данном случае функция

*generateCycle(index, iSuperCycle)* генерирует цикл. Количество итераций равно количеству циклов в сверхцикле.

```
function generateCycle(cycleIndex, iSuperCycle) {
let outputCode = []
for (let index = 0; index <= 31; index++) {
// цикл состоит из 32 канальных интервалов
outputCode = [
...outputCode,
...generateTimeslot(index, cycleIndex, iSuperCycle),
]
}
```

Аргументами данной функции являются индекс текущего цикла и индекс текущего сверхцикла. В данной функции, как и в предыдущем случае, мы итеративно вызываем такую функцию как *generateTimeslot(index, cycleIndex, iSuperCycle)*, генерирующую канальный интервал.

Внутри функции *generateTimeslot(index, cycleIndex, iSuperCycle)* сначала мы находим значение ошибочного бита, определенного псевдослучайным образом, при условии что мы находимся не в нулевом цикле:

```
let errorBit = cycleIndex ? Math.floor(Math.random() * 2) : 1
```

Это необходимо для имитации ошибок в линии. Так же для имитации потери цикловой сс мы обнуляем ошибочный при условии нахождения в втором третьем или четвертом сверхцикле.

```
errorBit = iSuperCycle >= 2 && iSuperCycle <= 4 ? 0 : errorBit
```

Далее мы встраиваем этот ошибочный бит в последовательность цикловой сс:

```
let frameSync = [0, 0, errorBit, 1, 0, 1, 1]
```

Потом мы определяем условие нахождения в нулевом КИ. Внутри мы определяем четность цикла.

```
if (timeslotIndex === 0) {
```

```
        if (cycleIndex % 2 === 0) {
            return [0, ...frameSync] /// в четных циклах передается первым битом конт
роль ошибок, а остальное цикловая сс (frameSync)
        } else {
            return [0, 1, 0, 1, 1, 0, 1, 1] /// в нечетных различная служебная информ
ация
        }
    }
```

Если итерация происходит в четном цикле функция возвращает биты цикловой сс, в остальных случаях различную служебную информацию.

Далее мы проверяем, что индекс КИ равен 16, и внутри проверяем четность:

```
if (timeslotIndex === 16) {
        //определяем 16 канальный интервал
        if (cycleIndex === 0) {
            return [0, 0, 0, 0, 1, 0, 0, 1] // в нулевом цикле первые четыре бита све
рхцикловой сс
        } else {
            return [1, 1, 0, 1, 1, 1, 0, 1] // в остальных циклах сигналы управления
и взаимодейтсвия
        }
    }
```

Если ни одна из проверок не сработала, значит мы находимся в КИ с нагрузкой, тогда функция возвращает вызов функции *getLoad()*. Эта функция возвращает 8 бит определенных псевдослучайным образом.

```
function getLoad() {
    let outputCode = []
    for (let index = 0; index < 8; index++) {
        outputCode.push(Math.floor(Math.random() * 2))
    }

    return outputCode
}
```

### 2.2. Разработка алгоритма распознавателя метки цикла потока E1

Перед тем как описать алгоритм распознавателя, нам нужно описать класс *Counters*, для хранения состояний счетчиков и методов работы с ними (описание приведено в приложениях).

Алгоритм распознавателя определён внутри функции *receiver(sequence)* на вход которой поступает сформированный ранее поток E1.

Начинается алгоритм с объявления экземпляра *counters* класса *Counters*.

```
let counters = new Counters()
```

Далее мы объявляем цикл, количество итераций которого, равно длине приходящей в функцию последовательности иначе говоря, количеству поступающих бит. Все дальнейшие операции описаны внутри цикла, и будут выполняться при каждой итерации.

В первых строках цикла проверяем что текущий рассчитанный бит ненулевой, а текущий индекс итерации не равен рассчитанному биту. При выполнении условий мы пропускаем эту итерацию, так как проверять наличие цикловой сс мы можем только через определенное количество битов.

```
if (counters.expectedSignal && index !== counters.expectedSignal) {
            continue
        }
```

Далее мы вызываем функцию *identifier(sequence, counters.predeterminationExpSignal(index))*, которая возвращает булевое значение синхронизационной метки.

```
function identifier(sequence, start) {
    let seq = [...sequence]
    let frameSync = [0, 0, 1, 1, 0, 1, 1].join('')
    const register = seq.splice(start, 7).join('')
    return register === frameSync
}
```

Функция *identifier* выбирает 7 бит из последовательности и сравнивает с битами цикловой сс, если они совпадают метка синхронизации найдена и переменной *syncMark* присваивается значение true.

После того как синхронизационная метка определена значением true или false, мы проверяем условие, что метка синхронизации найдена (значение true), при выполнении которого мы увеличиваем счетчик входа в синхронизм и рассчитываем следующее положение бита, на котором должны встретить биты цикловой сс. Если условие не выполняется, мы проверяем что текущий индекс итерации равен рассчитанному сигналу и не равен нулю. Если условие выполняется. Мы инкрементируем счетчик выхода из синхронизма и рассчитываем индекс новой цикловой сс.

```
if (syncMark) {
        counters.n1Increment()
        counters.setExpectedSignal(index)
} else if (index === counters.expectedSignal && !syncMark && index) {
        counters.n2Increment()
        counters.setExpectedSignal(index)
}
```

Далее проверяем счетчики на переполненность. При переполнении счетчика входа в синхронизм в консоль будет выведено сообщение «синхронизм достигнут» и вызываем метод сброса счетчиков.

```
if (counters.countN1 === 8) {
            console.log('синхронизм достигнут!')
            counters.resetCounters()
         }
```

При переполнении счетчика выхода из синхронизма в консоль выводится сообщение «произошел выход из синхронизма», сбрасывается счетчик входа в синхронизм, сбрасывается ожидаемый номер бита и декрементируется счетчик выхода из синхронизма.

```
if (counters.countN2 === 16) {
```

```
            console.log('произошел выход из синхронизма!')
            counters.resetN1()
            counters.n2Decrement()
            counters.resetExpSignal()
}
```

Полный код алгоритма приведен в приложении.

## Заключение

В ходе данной работы был разработан программный алгоритм одного из узлов первичного мультиплексора ИКМ-30, выделитель метки циклового синхросигнала. Были изучены все функции данного блока, а также теория построения потока Е1.

## Список использованной литературы:

1. [1]Цифровые и аналоговые системы передачи : учебник для вузов / В. И. Иванов [и др.]; Под ред. В. И. Иванова. – Москва: Горячая линия-Телеком, 2003. – 232 с.: ил.

## Код программы

```
/////=================Генератор последовательности==================
function generateSequence() {
    let stubBits = [0, 0, 0, 0, 0, 0, 0, 0]
    let outputCode = [...stubBits]

    for (let index = 0; index < 12; index++) {
        ////цикл формирования сверхциклов ( возьмем для примера 12)
        outputCode = [...outputCode, ...generateSuperCycle(index)]
    }

    return outputCode.concat(stubBits)
}

function generateSuperCycle(iSuperCycle) {
    let outputCode = []
    for (let index = 0; index <= 15; index++) {
        // сверхцикл состоит из 16 циклов
        outputCode = [...outputCode, ...generateCycle(index, iSuperCycle)]
    }
    return outputCode
}

function generateCycle(cycleIndex, iSuperCycle) {
    let outputCode = []
    for (let index = 0; index <= 31; index++) {
        // цикл состоит из 32 канальных интервалов
        outputCode = [
            ...outputCode,
            ...generateTimeslot(index, cycleIndex, iSuperCycle),
        ]
    }
    return outputCode
}

function generateTimeslot(timeslotIndex, cycleIndex, iSuperCycle) {
    let errorBit = cycleIndex ? Math.floor(Math.random() * 2) : 1 // имитация помех в
 линии
    errorBit = iSuperCycle >= 2 && iSuperCycle <= 4 ? 0 : errorBit /// имитация потер
и цикловой сс
    let frameSync = [0, 0, errorBit, 1, 0, 1, 1]
```

```
    if (timeslotIndex === 0) {
        ///определяем первый канальный интервал
        if (cycleIndex % 2 === 0) {
            return [0, ...frameSync] /// в четных циклах передается первым битом конт
роль ошибок, а остальное цикловая сс (frameSync)
        } else {
            return [0, 1, 0, 1, 1, 0, 1, 1] /// в нечетных различная служебная информ
ация
        }
    }

    if (timeslotIndex === 16) {
        //определяем 16 канальный интервал
        if (cycleIndex === 0) {
            return [0, 0, 0, 0, 1, 0, 0, 1] // в нулевом цикле первые четыре бита све
рхцикловой сс
        } else {
            return [1, 1, 0, 1, 1, 1, 0, 1] // в остальных циклах сигналы управления
и взаимодейтсвия
        }
    }

    return getLoad() // в остальных канальных интервалах передается нагрузкка
}

function getLoad() {
    let outputCode = []

    for (let index = 0; index < 8; index++) {
        outputCode.push(Math.floor(Math.random() * 2))
    }

    return outputCode
}
/////=================Генератор последовательности==================
///////////////=====================================================

class Counters {
    constructor() {}

    expectedSignal = 0
    countN1 = 0 // счетчик входа в синхронизм
```

```
    countN2 = 0 // счетчик выхода из синхронизма

    setExpectedSignal(index) {
        this.expectedSignal = generator(index)
    }

    n1Increment() {
        this.countN1++
    }

    n2Increment() {
        this.countN2++
    }

    n2Decrement() {
        this.countN2--
    }

    predeterminationExpSignal(index) {
        return this.expectedSignal || index
    }

    resetCounters() {
        this.countN1 = 0
        this.countN2 = 0
    }

    resetExpSignal() {
        this.expectedSignal = 0
    }
    resetN2() {
        this.countN2 = 0
    }
    resetN1() {
        this.countN1 = 0
    }
}

///////////////=====================================================
/////=================Выделитель метки цикловой сс==================
function receiver(sequence) {
    let counters = new Counters()
```

```
    for (let index = 0; index < sequence.length; index++) {
        if (counters.expectedSignal && index !== counters.expectedSignal) {
            continue
        }

        let syncMark = identifier(
            sequence,
            counters.predeterminationExpSignal(index)
        ) /// метка цикла

        if (syncMark) {
            counters.n1Increment()
            counters.setExpectedSignal(index)
        } else if (index === counters.expectedSignal && !syncMark && index) {
            counters.n2Increment()
            counters.setExpectedSignal(index)
        }

        if (counters.countN1 === 8) {
            console.log('синхронизм достигнут!')
            counters.resetCounters()
        }

        if (counters.countN2 === 16) {
            console.log('произошел выход из синхронизма!')
            //-1
            counters.resetN1()
            counters.n2Decrement()
            counters.resetExpSignal()
        }
    }
}

function identifier(sequence, start) {
    let seq = [...sequence]
    let frameSync = [0, 0, 1, 1, 0, 1, 1].join('')
    const register = seq.splice(start, 7).join('')
    return register === frameSync
}

function generator(index) {
    let genCycle = 512
    return genCycle + index
```

```
}

/////==================Выделитель метки цикловой сс==================

receiver(generateSequence())
```