

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Политехнический институт
Факультет приборостроения,
информационных технологий и
электроники

Кафедра
«Информационная безопасность
систем и технологий»

Специальность 10.05.02 –

Специализация –

Информационная безопасность
телекоммуникационных систем
Разработка защищенных
телекоммуникационных систем

ДИПЛОМНЫЙ ПРОЕКТ

на тему

ПРОГРАММНО-АППАРАТНЫЙ МОДУЛЬ ФИЛЬТРАЦИИ,
МАРШРУТИЗАЦИИ И ПОДГОТОВКИ К ШИФРОВАНИЮ СЕТЕВОГО
ТРАФИКА УСТРОЙСТВА IP-ШИФРОВАНИЯ

Студент	_____	<u>Цыганова Валентина Алексеевна</u>
Руководитель	_____	<u>Огурцов С.В.</u>
Нормоконтролёр	_____	<u>Фатеев А.Г.</u>
Рецензент <u>начальник НИС АО «ПНИЭИ»</u>	_____	<u>Чернов П.С.</u>

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой _____ С.Л. Зефилов

Работа защищена с отметкой _____ (протокол заседания ГЭК
от _____ № _____)

Секретарь ГЭК _____ А.Г. Фатеев

Пенза 2016

РЕФЕРАТ

Отчет 148 с., 25 рис., 5 табл., 6 источников, 7 прил.

ФИЛЬТРАЦИЯ, МАРШРУТИЗАЦИЯ, СЕТЕВОЙ ТРАФИК, АЛГОРИТМ, СТРУКТУРА ДАННЫХ, ПРОТОКОЛ IP, ПРОТОКОЛ IPSec, ОБРАБОТКА ПАКЕТОВ, ПРОГРАММНО-АППАРАТНЫЙ МОДУЛЬ, ПРОГРАММА, УСТРОЙСТВО IP-ШИФРОВАНИЯ

Объектом исследования является программно-аппаратный модуль фильтрации, маршрутизации и подготовки к шифрованию сетевого трафика устройства шифрования.

Целью дипломного проекта является разработка программно-аппаратного модуля фильтрации, маршрутизации и подготовки к шифрованию сетевого трафика устройства IP-шифрования.

В процессе работы были рассмотрены протоколы IP и IPSec, был проведен анализ принципов работы и средств приема и обработки трафика в устройстве IP-шифрования. Были разработаны структура и алгоритм работы аппаратного модуля обработки пакетов. Аппаратная часть модуля была реализована на языке VHDL. Были разработаны структуры данных для программной обработки пакетов и алгоритмы программных частей модуля. Полученные алгоритмы программных частей модуля были реализованы на языке ассемблер процессорного ядра. В завершение работы разработанный модуль в составе устройства IP-шифрования был успешно протестирован.

В результате дипломного проектирования был разработан и успешно протестирован программно-аппаратный модуль фильтрации, маршрутизации и подготовки к шифрованию сетевого трафика устройства IP-шифрования.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

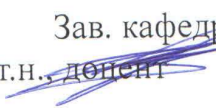
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

УДК 004.056

Политехнический институт

УТВЕРЖДАЮ

Факультет приборостроения,
информационных технологий и
электроники
Выпускающая кафедра:
Информационная безопасность систем
и технологий
Учебная группа 11ПТ1

Зав. кафедрой ИБСТ
к.т.н., доцент  С.Л. Зефирова
« 25 » 10 2016 г.

**ТЕХНИЧЕСКОЕ ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ
РАБОТУ**

студента группы 11ПТ1 Цыгановой Валентины Алексеевны

Специальность 100502 – Информационная безопасность
телекоммуникационных систем

Тема дипломного проекта:

Программно-аппаратный модуль фильтрации, маршрутизации и
подготовки к шифрованию сетевого трафика устройства IP-шифрования

Руководитель ВКР

Начальник НИС-45 АО «ПНИЭИ»



С.В. Огурцов

Нормоконтролер

к.т.н., доцент кафедры ИБСТ



А.Г. Фатеев

1 Цели и задачи ВКР

Цель работы – разработка программно-аппаратного модуля фильтрации, маршрутизации и подготовки к шифрованию сетевого трафика устройства IP-шифрования.

Модуль предназначен для анализа, фильтрации, обработки и маршрутизации входящего и исходящего сетевого трафика устройства IP-шифрования.

2 Тактико – технические требования к выполнению ВКР

Технические требования:

- должны быть определены правила фильтрации для протоколов Ethernet, IP, ESP;
- должен быть определен формат структур данных, используемых для обработки сетевого трафика защищенных каналов передачи данных;
- должны быть определены правила маршрутизации обработанных пакетов;
- должна быть разработана аппаратная часть модуля, встраиваемая в архитектуру устройства IP-шифрования;
- аппаратная часть должна быть реализована для программируемых логических интегральных схем Xilinx семейства Spartan 6;
- аппаратная часть должна быть реализована на языке описания аппаратуры VHDL;
- аппаратная часть должна включать процессорное ядро PicoBlaze, реализованное на языке описания аппаратуры VHDL;
- должна быть разработана программа, обеспечивающая фильтрацию и подготовку к шифрованию входящего трафика со стороны абонентской сети;
- должна быть разработана программа, обеспечивающая фильтрацию и маршрутизацию исходящего трафика со стороны абонентской сети;
- должна быть разработана программа, обеспечивающая фильтрацию и подготовку к расшифрованию входящего трафика со стороны сети



зашифрованной информации;

- должна быть разработана программа, обеспечивающая фильтрацию и маршрутизацию исходящего трафика со стороны сети зашифрованной информации;

- реализация программ должна быть выполнена на языке ассемблер процессорного ядра PicoBlaze.

В ВКР должны быть рассмотрены следующие вопросы:

- анализ архитектуры и места в системе связи устройства IP-шифрования;

- анализ аппаратных средств приема и передачи сетевого трафика, реализованных в канальных интерфейсах устройства IP-шифрования;

- обзор протоколов IP и IPSec;

- определение форматов структур данных, используемых для обработки сетевого трафика защищенных каналов передачи данных;

- разработка алгоритма и структуры аппаратной части модуля;

- реализация аппаратной части модуля на языке VHDL;

- разработка алгоритмов программных частей модуля;

- реализация полученных алгоритмов программных частей модуля на языке ассемблер процессорного ядра;

- тестирование разработанного модуля в составе устройства.



3 Требования к разрабатываемой документации

Отчет по ВКР выполняется в соответствии с требованиями ГОСТ 7.32.

Перечень и содержание графической части ВКР:

- архитектура и место в системе связи устройства IP-шифрования – 1л., ф. А1 (плакат);
- формат структур данных, используемых для обработки сетевого трафика защищенных каналов передачи данных – 2л., ф. А1 (плакат);
- структура аппаратной части модуля фильтрации, маршрутизации и подготовки к шифрованию сетевого трафика устройства IP-шифрования – 1л., ф. А1 (плакат);
- алгоритм обработки входящего трафика коммутатора сообщений абонентской сети – 1л., ф. А1 (плакат);
- алгоритм обработки исходящего трафика коммутатора сообщений абонентской сети – 1л., ф. А1 (плакат);
- алгоритм обработки входящего трафика коммутатора сообщений сети зашифрованной информации – 1л., ф. А1 (плакат);
- алгоритм обработки исходящего трафика коммутатора сообщений сети зашифрованной информации – 1л., ф. А1 (плакат).

Электронная копия отчета и графической части ВКР должна быть представлена на магнитном или оптическом носителе.

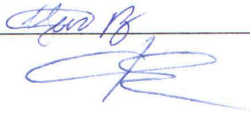
4 Сроки выполнения ВКР

Тема утверждена приказом ректора ПГУ № ____ от «__» ____ 2016 г.

Дата выдачи задания «22» августа 2016 г.

Время дипломного проектирования с 10 октября 2016 г. по 15 января 2017 г.

Задание к исполнению принял «22» августа 2016 г.

Исполнитель ВКР  Цыганова В.А.

ОПРЕДЕЛЕНИЯ

В настоящем дипломном проекте применяют следующие термины с соответствующими определениями:

- VHDL – стандартизированный язык высокого уровня для описания аппаратуры [5];
- ESP (Encapsulating Security Payload) – протокол, обеспечивающий безопасность сетевого трафика при помощи шифрования [1];
- IP – протокол, предназначенный для использования в соединенных между собой компьютерных сетях обмена данными на основе коммутации пакетов [1];
- IP Security – комплект протоколов, касающихся вопросов шифрования, аутентификации и обеспечения защиты при транспортировке IP-пакетов [1].

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

АН (Authentication Header) – заголовок аутентификации;
CRC (Cyclic redundancy check) – циклический избыточный код;
DMA (Direct memory access) – прямой доступ к памяти;
IP (Internet Protocol) – межсетевой протокол;
TTL (Time to live) – время жизни пакета;
АКС – коммутатор сообщений абонентской сети;
БФП – блок функциональных преобразований;
МКС – коммутатор сообщений зашифрованной сети;
ОЗУ – оперативное запоминающее устройство;
ПЛИС – программируемая логическая интегральная схема;
ПСП – псевдослучайная последовательность;
ЦП – центральный процессор;
ЦПУ – центральное процессорное устройство.

СОДЕРЖАНИЕ

Введение.....	11
1 Анализ архитектуры и места в системе связи устройства IP-шифрования .	12
2 Анализ аппаратных средств приема и передачи сетевого трафика, реализованных в канальных интерфейсах устройства IP-шифрования	15
3 Обзор протоколов IP и IPSec.....	18
3.1 Обзор протокола IP	18
3.2 Обзор протокола IPSec	25
4 Определение форматов структур данных, используемых для обработки сетевого трафика защищенных каналов передачи данных.....	31
4.1 Структуры данных для обработки входящего трафика коммутатора сообщений абонентской сети.....	31
4.2 Структуры данных для обработки исходящего трафика коммутатора сообщений абонентской сети.....	33
4.3 Структуры данных для обработки входящего трафика коммутатора сообщений сети зашифрованной информации	36
4.4 Структуры данных для обработки исходящего трафика коммутатора сообщений сети зашифрованной информации	38
5 Разработка алгоритма и структуры аппаратной части модуля	40
6 Реализация аппаратной части модуля на языке VHDL.....	44
7 Разработка алгоритмов программных частей модуля.....	48
7.1 Алгоритм обработки входящего трафика коммутатора сообщений абонентской сети.....	48
7.2 Алгоритм обработки исходящего трафика коммутатора сообщений абонентской сети.....	49
7.3 Алгоритм обработки входящего трафика коммутатора сообщений сети зашифрованной информации	51
7.4 Алгоритм обработки исходящего трафика коммутатора сообщений сети зашифрованной информации	53

8 Реализация полученных алгоритмов программных частей модуля на языке ассемблер процессорного ядра	55
9 Тестирование разработанного модуля в составе устройства	65
Заключение	68
Список использованных источников	69
Приложение А. Исходные коды аппаратной части модуля на языке VHDL..	70
Приложение Б. Алгоритм обработки входящего трафика коммутатора сообщений абонентской сети.....	102
Приложение В. Алгоритм обработки исходящего трафика коммутатора сообщений абонентской сети.....	105
Приложение Г. Алгоритм обработки входящего трафика коммутатора сообщений сети зашифрованной информации	108
Приложение Д. Алгоритм обработки исходящего трафика коммутатора сообщений сети зашифрованной информации	111
Приложение Е. Исходные коды программных частей модуля на языке ассемблер процессорного ядра	113
Приложение Ж. Результаты проверки ВКР на оригинальность	147

ВВЕДЕНИЕ

Шифрование решает задачу обеспечения конфиденциальности информации. При этом алгоритм шифрования должен быть реализован в виде программного или аппаратного шифратора. При реализации встает задача обеспечения взаимодействия устройства шифрования с системами, являющимися источниками и получателями защищаемой информации. Такое взаимодействие должно исключать возможность создания каналов утечки. С этой целью для устройств шифрования реализуются процедуры фильтрации входных и контроля выходных данных.

Данный дипломный проект посвящен разработке программно-аппаратного модуля фильтрации, маршрутизации и подготовки к шифрованию сетевого трафика устройства IP-шифрования.

Актуальность дипломного проекта состоит в том, что с переходом на новую аппаратную платформу и в связи с новыми требованиями к устройству шифрования необходима разработка новых средств обработки информации с повышенными характеристиками в части пропускной способности и безопасности процесса обработки информации.

1 Анализ архитектуры и места в системе связи устройства IP-шифрования

Место устройства в системе связи представлено на рисунке 1.

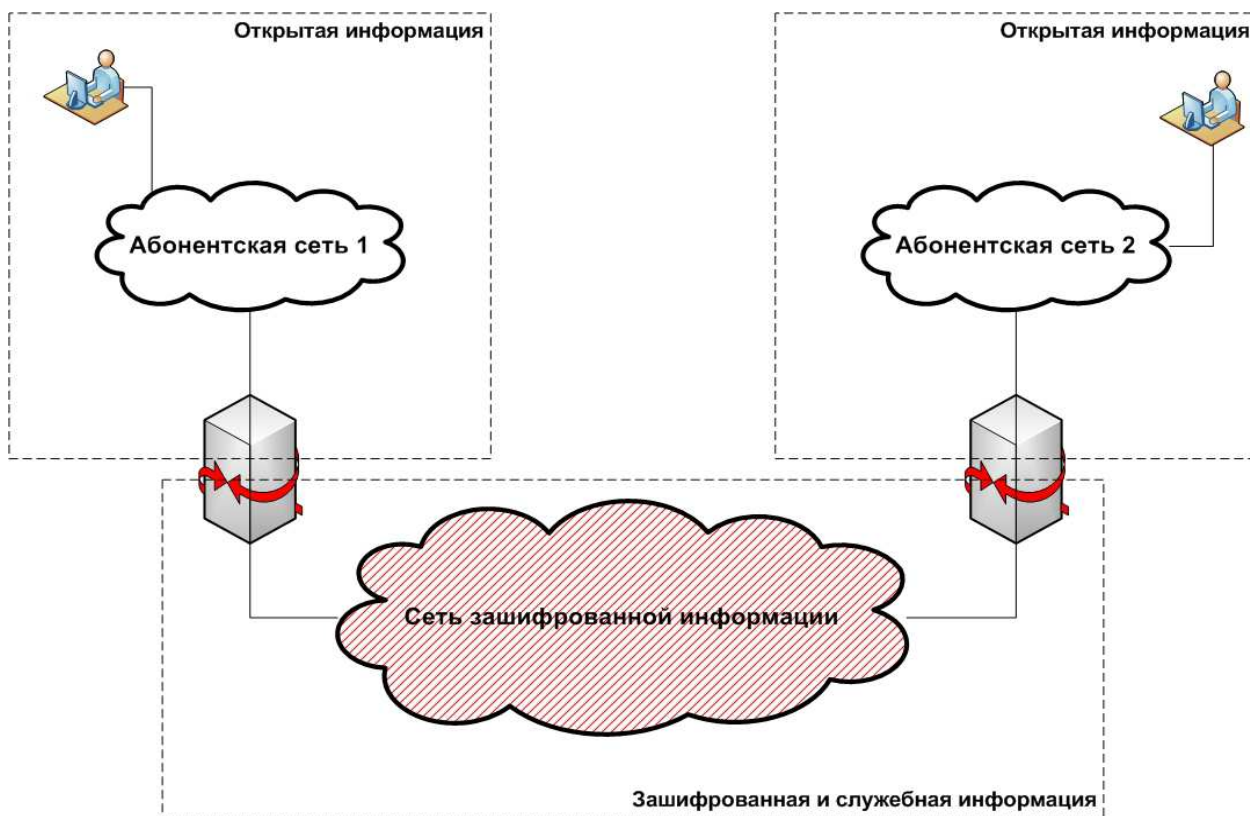


Рисунок 1 – Место устройства в системе связи

Передача информации между устройствами шифрования осуществляется по защищенным каналам, построенным на основе протокола ESP в режиме туннелирования.

Параметры каналов связи описываются конфигурацией, состоящей из параметров туннелей: адресов и масок абонентских сетей.

На основе конфигурации формируются структуры, используемые при обработке трафика:

- таблицы сеансов – содержат параметры для обработки входящего трафика;
- таблицы маршрутизации – содержат параметры для обработки исходящего трафика.

Внутренняя архитектура устройства представлена на рисунке 2.

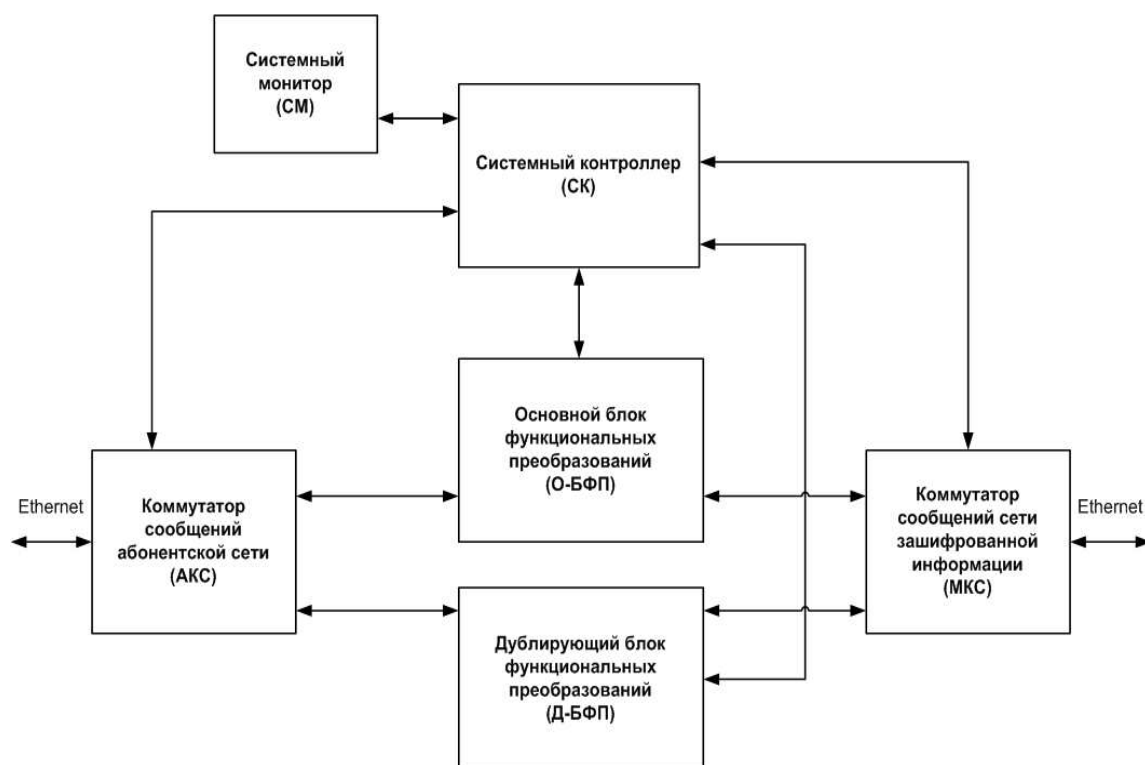


Рисунок 2 – Архитектура устройства

Функциональные блоки, составляющие архитектуру устройства, реализованы на отдельных ПЛИС и выполняют следующие задачи:

- коммутатор сообщений абонентской сети (АКС) – реализует взаимодействие с абонентской сетью: прием, проверку и обработку пакетов поступающих из абонентской сети, передачу корректных пакетов на шифрование в основной и дублирующий БФП, прием и сравнение расшифрованных пакетов, поступающих из БФП, маршрутизацию корректных пакетов из БФП в абонентскую сеть;
- коммутатор сообщений зашифрованной сети (МКС) – реализует взаимодействие с сетью зашифрованной информации: прием, проверку и обработку пакетов поступающих из сети зашифрованной информации, передачу корректных пакетов на расшифрование в основной и дублирующий БФП, прием и сравнение зашифрованных пакетов, поступающих из БФП, маршрутизацию корректных пакетов из БФП в сеть зашифрованной информации;

- основной блок функциональных преобразований (О-БФП), дублирующий блок функциональных преобразований (Д-БФП) – реализуют шифрование пакетов, поступающих от АКС, и расшифрование пакетов, поступающих от МКС;

- системный контроллер (СК) – реализует функции по контролю функционирования блоков устройства, взаимодействие блоков, конфигурирование устройства;

- системный монитор (СМ) – реализует контроль за средой функционирования устройства: питанием, температурой и т.д.

2 Анализ аппаратных средств приема и передачи сетевого трафика, реализованных в канальных интерфейсах устройства IP-шифрования

Функциональная схема модулей, участвующих в обработке трафика в коммутаторе сообщений представлена на рисунке 3.

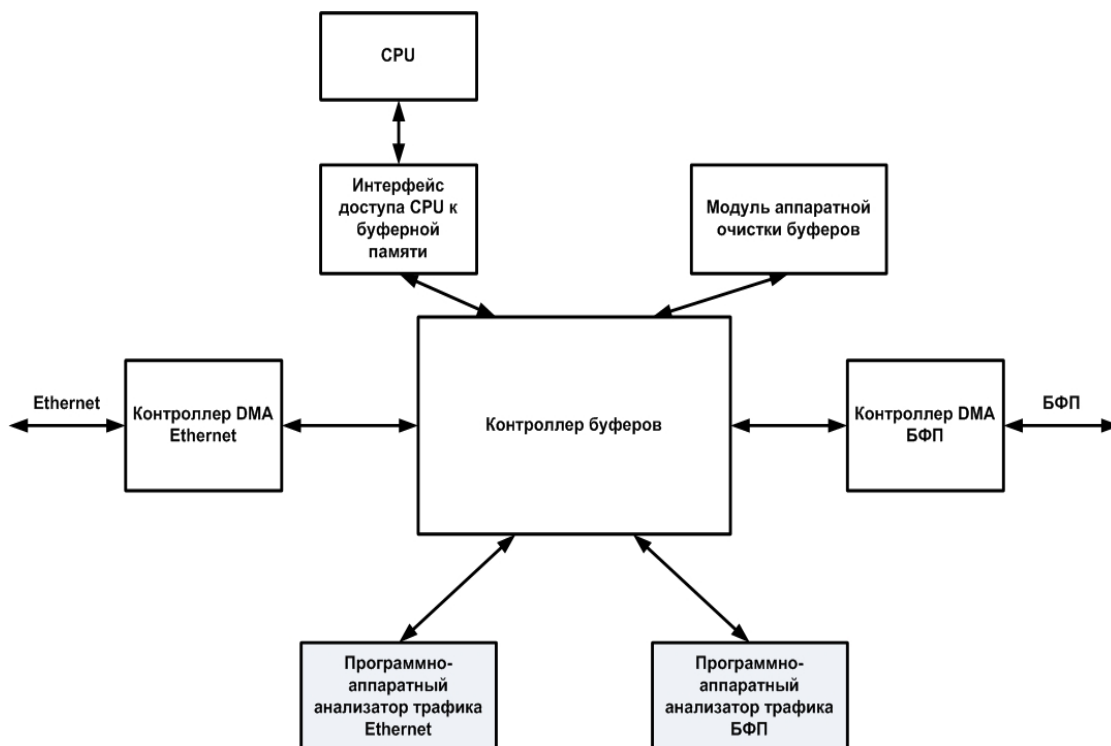


Рисунок 3 – Структура системы приема и обработки сетевого трафика коммутатора сообщений

В коммутаторах сообщений реализуется двухуровневая модель обработки трафика:

- верхний уровень – обработка пакетов выполняется центральным процессором – не основной режим работы, реализуется для пакетов, которые не могут быть обработаны нижним уровнем:
 - пакеты, для которых не сформированы записи таблиц сеансов и конфигурации;
 - пакеты, требующие фрагментации;
 - пакеты служебных протоколов;

– нижний уровень – обработка пакетов выполняется анализаторами пакетов – основной режим работы, реализуется высокоскоростная обработка пакетов с данными.

После снятия сигнала сброса для контроллера буферов, запускается процедура начальной инициализации очереди свободных буферов путем заполнения ее последовательными номерами буферов, выделенных в буферной области. После завершения инициализации начинается распределение пакетов из очереди свободных буферов в очереди приема пакетов. Инициализация очередей завершается при полном заполнении очередей приема. Из очередей приема пакетов контроллерам DMA выделяются по одному буферу на прием пакета.

Пакет от модуля MAC по каналу LocalLink поступает в контроллер DMA канала Ethernet. Контроллер DMA начинает сохранение получаемого пакета в выделенный буфер со смещением относительно начала буфера, заданным для текущего модуля. Запись в буфер выполняется транзакциями, не превышающими 256 байт (максимальный размер блока данных для одной операции MCB). При заполнении приемного буфера MCB блоком данных, для канала LocalLink выставляется сигнал неготовности приемника и запрашивается операция записи MCB, выполняющая запись сформированного блока данных в DDR. По завершении записи текущего блока, сигнал неготовности приемника снимается и начинается прием следующего блока. Если получен сигнал завершения передачи пакета, выполняется запись текущего блока, для канала LocalLink устанавливается сигнал неготовности. По завершении записи текущего блока, выполняется запись заголовка буфера. Указатель на буфер передается в очередь пакетов на обработку анализатором трафика Ethernet. Буфер из очереди поступает на обработку в анализатор трафика Ethernet, где по полученному указателю вычитывается начало буфера, содержащее заголовок буфера DMA и область пакета, занимаемую всеми необходимыми для анализа заголовками (Ethernet, IP, ESP). Заголовок буфера читается из памяти DDR во внутреннюю память

BRAM анализатора пакетов. По завершении чтения, процессор анализатора пакетов начинает разбор пакета из внутренней памяти. Анализатор пакета выполняет проверку пакета, по результатам которой:

- для некорректных пакетов – указатель на буфер передается в очередь буферов на очистку;
- для корректных пакетов, которые не могут быть обработаны анализатором – указатель на буфер передается в очередь буферов на обработку ЦПУ;
- для корректных пакетов, которые могут быть обработаны анализатором – выполняется модификация заголовков в памяти DDR и передача указателя на буфер в очередь пакетов на передачу в БФП.

Указатель на буфер из очереди на передачу в БФП поступает в контроллер DMA канала БФП. Контроллер передачи выполняет чтение заголовка буфера и проверку его корректности (размер передаваемых данных). Если параметры заголовка корректны, начинается передача пакета блоками по 256 байт, кроме последнего который может быть менее 256 байт. По завершении передачи, буфер, содержащий в себе данные переданного пакета, отправляется в очередь буферов на очистку.

Буфер из очереди буферов на очистку поступает в модуль очистки, где выполняется перезапись области от начала буфера до конца данных. Размер очищаемой области определяется по содержимому заголовка буфера. Если размер определяется некорректно, очистке подвергается буфер целиком. По завершении очистки, указатель на буфер передается в очередь свободных буферов.

Аналогичным образом обрабатываются пакеты, поступающие от БФП.

3 Обзор протоколов IP и IPSec

3.1 Обзор протокола IP

Протокол IP предназначен для использования в соединенных между собой компьютерных сетях обмена данными на основе коммутации пакетов. Протокол обеспечивает передачу блоков данных, называемых дейтаграммами между отправителем и получателем, хосты которых идентифицируются адресами фиксированной длины. Протокол также обеспечивает фрагментацию и сборку для дейтаграмм большого размера, если сеть не позволяет передать дейтаграмму целиком [1].

Протокол IP выполняет две основных функции – адресацию и фрагментацию/сборку дейтаграмм.

Модули IP используют адреса из заголовков IP для передачи дейтаграмм в направлении получателя. Процесс выбора пути к адресату называется маршрутизацией.

Модули IP используют поля заголовков IP для фрагментации и сборки дейтаграмм IP при необходимости передачи через сети с малым размером пакетов.

Модули IP используются на каждом хосте, участвующем в сети и на каждом маршрутизаторе, соединяющем сети. Эти модули используют общие правила интерпретации полей адреса и фрагментации/сборки дейтаграмм IP. Кроме того, эти модули (особенно в маршрутизаторах) выполняют процедуры принятия решения о пересылке дейтаграмм и выполняют еще ряд функций. Протокол IP трактует каждую дейтаграмму как независимый элемент, не связанный с другими дейтаграммами IP. Протокол не использует соединения или логических устройств (виртуальных или иных).

Для обеспечения сервиса протокол IP использует 4 ключевых механизма – ToS (тип обслуживания), TTL (время жизни), Options (опции) и Header Checksum (контрольная сумма заголовка).

Тип обслуживания (ToS) используется для индикации желаемого качества сервиса. ToS представляет собой абстрактный или обобщенный набор параметров, характеризующих выбранный сервис, который обеспечивается в сетях, образующих Internet. Индикация ToS используется маршрутизаторами для выбора реальных параметров передачи применительно к конкретной сети, следующего интервала или следующего маршрутизатора при доставке дейтаграмм IP.

Время жизни TTL определяет максимальный срок существования дейтаграмм IP. Это значение устанавливается отправителем и уменьшается в каждой точке на пути доставки, где дейтаграмма подвергается обработке. Если значение TTL становится нулевым до того, как дейтаграмма будет доставлена адресату, такая дейтаграмма просто уничтожается. Можно рассматривать TTL как время саморазрушения дейтаграмм.

Опции обеспечивают функции контроля требуемые или полезные в некоторых ситуациях, но не используемые для большинства рутинных задач. Опции включают временные метки, параметры безопасности и специальные средства маршрутизации. Контрольная сумма заголовка обеспечивает возможность проверки корректности передачи дейтаграмм IP. Если при передаче дейтаграмма была повреждена вычисленная заново при обработке контрольная сумма заголовка не совпадет с содержащимся в дейтаграмме значением контрольной суммы и такая дейтаграмма отбрасывается как ошибочная.

При обнаружении ошибок информация о них может передаваться с помощью протокола ICMP (Internet Control Message Protocol), реализуемого в модуле IP.

Модель передачи дейтаграмм от одной прикладной программы к другой можно проиллюстрировать описанным ниже сценарием. Предполагается, что передача включает лишь один промежуточный шлюз.

Передающая программа готовит свои данные и вызывает локальный модуль IP для передачи этих данных как дейтаграммы, указывая адрес получателя и другие параметры в качестве аргументов.

Модуль IP готовит заголовок дейтаграммы и присоединяет к нему данные. После этого модуль IP определяет локальный сетевой адрес для указанного получателя (в данном случае это адрес шлюза). Модуль передает дейтаграмму и локальный адрес локальному сетевому интерфейсу.

Интерфейс канального уровня создает заголовок и присоединяет к нему дейтаграмму IP, после чего пакет передается в локальную. Дейтаграмма приходит на хост-шлюз в пакете канального уровня. Интерфейс канального уровня удаляет заголовок канального уровня и передает дейтаграмму модулю IP. Модуль IP определяет на основе IP-адреса, что дейтаграмму следует переслать хосту другой сети. Тогда модуль IP определяет адрес канального уровня для пересылки дейтаграммы получателю и вызывает интерфейс канального уровня той сети, куда будет передаваться дейтаграмма. Интерфейс канального уровня создает заголовок и, присоединив к нему дейтаграмму, передает пакет хосту-адресату. На хосте получателя дейтаграмма выделяется из пакета интерфейсом канального уровня и передается модулю IP. Модуль IP определяет по заголовку, что дейтаграмма адресована приложению на данном хосте и передает прикладной программе данные из дейтаграммы вместе с адресом отправителя и другими параметрами в ответ на системный вызов.

Задачей протокола IP является перемещение дейтаграмм через множество соединенных между собою сетей. Эта задача решается путем передачи дейтаграмм от одного модуля IP к другому, пока дейтаграмма не будет доставлена адресату. Модули IP размещаются на хостах и шлюзах (маршрутизаторах) Internet. Дейтаграммы маршрутизируются от одного модуля IP к другому через промежуточные сети на основе интерпретации адресов IP. Таким образом, одним из важнейших механизмов IP является адресация. При маршрутизации сообщений от одного модуля IP к другому

может потребоваться передача дейтаграмм через сети, для которых максимальный размер пакета меньше размера дейтаграммы. Для решения этой проблемы протокол IP обеспечивает механизмы фрагментации и сборки дейтаграмм.

Протокол IP имеет дело преимущественно с адресами. Отображение адресов на имена и обратно (преобразование) является задачей протоколов более высоких уровней (т.е. транспортного и сеансового). Модуль IP преобразует адреса IP в адреса локальной сети. Отображение адресов локальной сети в маршруты является задачей процедур нижележащего уровня (т.е. локальной сети или шлюзов).

Адреса имеют IP фиксированную длину – 4 октета (32 бита). Адрес начинается с номера сети, за которым следует локальный адрес(его называют полем *rest* – остаток). Существует три класса адресов IP – класс A, в котором старший бит имеет значение 0, остальные 7 битов старшего октета задают номер сети, а 24 младших бита – номер хоста; класс B, в котором два старших бита имеют значения 10, следующие 14 битов определяют номер сети, а последние 16 битов – номер хоста; класс C, в котором три старших бита имеют значения 110, следующие 21 образуют номер сети, а последние 8 битов определяют номер хоста.

Фрагментация дейтаграмм IP требуется в тех случаях, когда дейтаграмма происходит из сети, которая поддерживает больший размер пакетов, нежели промежуточные сети на пути к адресату.

Дейтаграмма IP может быть помечена как «don't fragment» (не фрагментировать). Такие дейтаграммы не будут фрагментироваться ни при каких обстоятельствах. Если нефрагментируемая дейтаграмма IP не может быть доставлена адресату без фрагментации, она просто отбрасывается.

Допускается использование невидимой для модуля IP фрагментации, передачи и сборки дейтаграмм в локальной сети. Процедуры фрагментации и сборки дейтаграмм должны обеспечивать возможность разбиения

дейтаграмм на почти произвольное число частей, которые впоследствии могут быть собраны воедино. Получатель фрагментов использует поле идентификации для того, чтобы фрагменты разных дейтаграмм не перемешивались. Поле смещения дейтаграммы говорит получателю о положении фрагмента в исходной дейтаграмме. Поля смещения и размера фрагмента определяют часть исходной дейтаграммы, содержащуюся в отдельном фрагменте. Флаг наличия последующих фрагментов (если он сброшен) говорит о том, что фрагмент является последним в дейтаграмме.

Поле идентификации позволяет различать фрагменты разных дейтаграмм. Отправляющий дейтаграмму модуль протокола устанавливает значение поля идентификации в каждой дейтаграмме так, чтобы оно было уникальным для данной пары отправитель-получатель и протокола в течение времени присутствия дейтаграммы в сети Internet. Этот модуль также устанавливает нулевые значения смещения фрагмента и флага наличия других фрагментов (more-fragments flag).

Для фрагментирования длинной дейтаграммы IP модуль IP (например, на маршрутизаторе) создает две новых дейтаграммы IP и копирует содержимое полей заголовка из длинной дейтаграммы в заголовки обеих новых дейтаграмм. Данные исходной дейтаграммы делятся на две части по 64 битовой (8 октетов) границе. Вторая часть дейтаграммы может иметь размер, не кратный 8 октетам (64 битам), но первая часть должна содержать целое число 8-октетных блоков. Первая часть дейтаграммы помещается в первую из новых дейтаграмм IP и поле длины устанавливается в соответствии с длиной первой дейтаграммы. Для первой дейтаграммы устанавливается флаг наличия дополнительных фрагментов. Вторая часть данных помещается во вторую из созданных заново дейтаграмм и поле размера устанавливается в соответствии с длиной новой дейтаграммы. Значение поля смещения увеличивается на величину NFB. Значение флага наличия дополнительных фрагментов сохраняется в соответствии с флагом исходной нефрагментированной дейтаграммы.

Для сборки фрагментов дейтаграммы IP, модуль IP объединяет дейтаграммы IP с совпадающими значениями полей идентификации, адресов отправителя и получателя, а также протокола. Объединение осуществляется путем размещения данных из каждой дейтаграммы в позицию буфера, указанную полем смещения фрагмента в заголовке IP. Первый фрагмент будет иметь нулевое смещение, а для последнего фрагмента флаг «more-fragments» будет иметь нулевое значение.

Формат заголовка IP-пакета представлен на рисунке 4.

0	4	8	16	31
Версия (Version)	Длина (IHL)	Тип сервиса (ToS)	Общая длина пакета (Total Length)	
Идентификатор			Флаги	Смещение фрагмента
Время жизни (TTL)	Протокол (см. след. слайд)		Контрольная сумма	
Адрес отправителя (Source IP address)				
Адрес получателя (Destination IP address)				
Опции (поле переменной длины)			Выравнивание до 32 бит (padding)	

Рисунок 4 – Формат заголовка IP-пакета

Заголовок IP-пакета содержит следующие поля:

- Version (4 бита) указывает номер версии протокола и определяет формат заголовка;
- IHL (4 бита) содержит размер заголовка IP в 32-битовых словах и указывает на начало данных;
- ToS (8 бит) обеспечивает индикацию абстрактных параметров желаемого качества обслуживания. Это значение используется при выборе реальных параметров обслуживания в процессе передачи дейтаграммы через отдельную сеть. Некоторые сети предлагают приоритетный сервис, который тем или иным способом трактует трафик с большим уровнем предпочтения как более важный, нежели трафик другого типа (обычно при высокой загрузке просто воспринимается только трафик с уровнем предпочтения

выше некоторого порога). Основной выбор осуществляется между тремя вариантами – малая задержка, высокая надежность, высокая пропускная способность;

- Total Length (16 бит) указывает общий размер (в октетах) дейтаграммы с учетом заголовка и данных;

- Identification (16 бит) присваивается отправителем для обеспечения корректной сборки фрагментов дейтаграммы;

- Flags (3 бита) – набор флагов управления:

- бит 0: зарезервирован (должен иметь значение 0);

- бит 1: (DF) 0 = фрагментация возможна, 1 = фрагментация недопустима;

- бит 2: (MF) 0 = последний фрагмент, 1 = фрагмент не является последним;

- Fragment Offset (13 бит) показывает положение данного фрагмента в исходной дейтаграмме. Смещение измеряется в единицах, кратных 8 октетам (64 бита). Смещение первого фрагмента равно нулю;

- TTL (8 бит) определяет максимальный срок существования дейтаграммы. Дейтаграммы с нулевым значением времени жизни должны уничтожаться. Значение этого поля изменяется при обработке заголовков IP;

- Protocol (8 бит) указывает протокол следующего уровня, содержащийся в поле данных дейтаграммы IP;

- Header Checksum (16 бит) – контрольная сумма полей заголовка. Поскольку некоторые поля заголовка (например, TTL) изменяются в процессе доставки, значение контрольной суммы проверяется и вычисляется заново в каждой точке обработки заголовков IP;

- Source Address (32 бита) – адрес отправителя;

- Destination Address (32 бита) – адрес получателя;

- Options – переменная длина. Поле опций является необязательным [1].

3.2 Обзор протокола IPSec

IP Security – это комплект протоколов, касающихся вопросов шифрования, аутентификации и обеспечения защиты при транспортировке IP-пакетов.

Гарантии целостности и конфиденциальности данных в спецификации IPsec обеспечиваются за счет использования механизмов аутентификации и шифрования соответственно. Последние, в свою очередь, основаны на предварительном согласовании сторонами информационного обмена т.н. «контекста безопасности» – применяемых криптографических алгоритмов, алгоритмов управления ключевой информацией и их параметров. Спецификация IPsec предусматривает возможность поддержки сторонами информационного обмена различных протоколов и параметров аутентификации и шифрования пакетов данных, а также различных схем распределения ключей. При этом результатом согласования контекста безопасности является установление индекса параметров безопасности (SPI), представляющего собой указатель на определенный элемент внутренней структуры стороны информационного обмена, описывающей возможные наборы параметров безопасности.

IPSec работает на сетевом уровне. В результате передаваемые IP-пакеты будут защищены прозрачным для сетевых приложений и инфраструктуры образом. В отличие от SSL (Secure Socket Layer), который работает на транспортном уровне и теснее связан с более высокими уровнями модели OSI, IPSec призван обеспечить низкоуровневую защиту.

К IP-данным, готовым к передаче по виртуальной частной сети, IPSec добавляет заголовок для идентификации защищенных пакетов. Перед передачей по Internet эти пакеты инкапсулируются в другие IP-пакеты. IPSec поддерживает несколько типов шифрования, в том числе Data Encryption Standard (DES) и Message Digest 5 (MD5).

Чтобы установить защищенное соединение, оба участника сеанса должны иметь возможность быстро согласовать параметры защиты, такие как алгоритмы аутентификации и ключи. IPSec поддерживает два типа схем управления ключами, с помощью которых участники могут согласовать параметры сеанса.

IPSec – протокол для обеспечения безопасности сетевого трафика при помощи шифрования. Эта функция осуществляется при помощи протокола, называемого ESP (Encapsulating Security Payload). В случае использования инкапсуляции зашифрованных данных заголовок ESP является последним в ряду опциональных заголовков, видимых в пакете. Формат заголовка ESP представлен на рисунке 5. Поскольку основной целью ESP является обеспечение конфиденциальности данных, разные виды информации могут требовать применения существенно различных алгоритмов шифрования. Следовательно, формат ESP может претерпевать значительные изменения в зависимости от используемых криптографических алгоритмов. Тем не менее, можно выделить следующие обязательные поля: SPI, указывающее на контекст безопасности и Sequence Number Field, содержащее порядковый номер пакета. Поле «ESP Authentication Data» (контрольная сумма), не является обязательным в заголовке ESP. Получатель пакета ESP расшифровывает ESP заголовок и использует параметры и данные применяемого алгоритма шифрования для декодирования информации транспортного уровня.



Рисунок 5 – Формат заголовка ESP

Другой функцией IPSec является способность проверки подлинности и целостности пакетов при помощи протокола АН (Authentication Header). Аутентифицирующий заголовок (АН) является обычным опциональным заголовком и, как правило, располагается между основным заголовком пакета IP и полем данных. Наличие АН никак не влияет на процесс передачи информации транспортного и более высокого уровней. Основным и единственным назначением АН является обеспечение защиты от атак, связанных с несанкционированным изменением содержимого пакета, и в том числе от подмены исходного адреса сетевого уровня. Протоколы более высокого уровня должны быть модифицированы в целях осуществления проверки аутентичности полученных данных. Формат заголовка АН представлен на рисунке 6. Формат АН достаточно прост и состоит из 96-битового заголовка и данных переменной длины, состоящих из 32-битовых слов. Названия полей достаточно ясно отражают их содержимое: Next Header (следующий заголовок) указывает на следующий заголовок, Payload Len представляет длину пакета, SPI является указателем на контекст безопасности и Sequence Number Field (порядковый номер) содержит последовательный номер пакета.

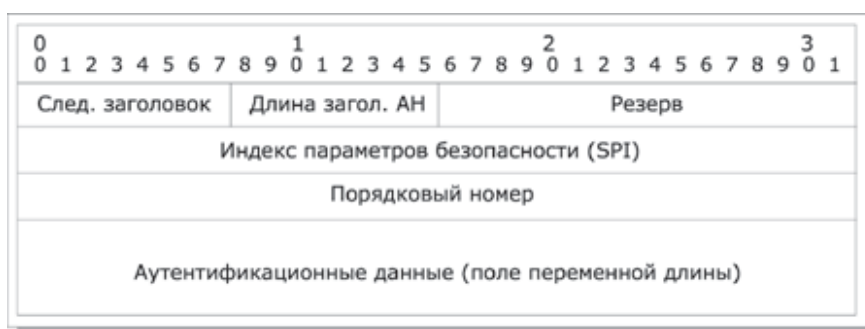


Рисунок 6 – Формат заголовка АН

Различают два режима применения ESP и АН – транспортный и туннельный.

Транспортный режим используется для шифрования поля данных IP пакета, содержащего протоколы транспортного уровня (TCP, UDP, ICMP), которое, в свою очередь, содержит информацию прикладных служб.

Примером применения транспортного режима является передача электронной почты. Все промежуточные узлы на маршруте пакета от отправителя к получателю используют только открытую информацию сетевого уровня и, возможно, некоторые опциональные заголовки пакета (в IPv6). Недостатком транспортного режима является отсутствие механизмов скрытия конкретных отправителя и получателя пакета, а также возможность проведения анализа трафика. Результатом такого анализа может стать информация об объемах и направлениях передачи информации, области интересов абонентов, расположение руководителей.

Туннельный режим предполагает шифрование всего пакета, включая заголовки сетевого уровня. Туннельный режим применяется в случае необходимости скрытия информационного обмена организации с внешним миром. При этом, адресные поля заголовка сетевого уровня пакета, использующего туннельный режим, заполняются межсетевым экраном организации и не содержат информации о конкретном отправителе пакета. При передаче информации из внешнего мира в локальную сеть организации в качестве адреса назначения используется сетевой адрес межсетевого экрана. После расшифровки межсетевым экраном начального заголовка сетевого уровня пакет направляется получателю.

Формирование контекстов безопасности в IPSec разделено на две фазы. Сначала создается управляющий контекст, назначение которого – предоставить доверенный канал, т. е. аутентифицированный, защищенный канал для выработки (в рамках второй фазы) протокольных контекстов и, в частности, для формирования криптографических ключей, используемых протоколами AH и ESP.

Для функционирования механизмов IPSec необходимы только протокольные контексты; управляющий играет вспомогательную роль. Более того, явное выделение двух фаз утяжеляет и усложняет формирование ключей, если рассматривать последнее как однократное действие. Тем не

менее, из архитектурных соображений управляющие контексты не только могут, но и должны существовать, поскольку обслуживают все протокольные уровни стека TCP/IP, концентрируя в одном месте необходимую функциональность. Первая фаза начинается в ситуации, когда взаимодействующие стороны не имеют общих секретов (общих ключей) и не уверены в аутентичности друг друга. Если с самого начала не создать доверенный канал, то для выполнения каждого управляющего действия с ключами (их модификация, выдача диагностических сообщений и т.п.) в каждом протоколе (AH, ESP и т.д.) этот канал придется формировать заново.

Существует несколько способов формирования управляющего контекста. Они различаются двумя показателями:

- используемым механизмом выработки общего секретного ключа;
- степенью защиты идентификаторов общающихся сторон.

В простейшем случае секретные ключи задаются заранее (ручной метод распределения ключей). Для небольших сетей такой подход вполне работоспособен, но он не является масштабируемым. Последнее свойство может быть обеспечено при автоматической выработке и распределении секретных ключей в рамках протоколов, основанных на алгоритме Диффи-Хелмана.

При формировании управляющего контекста идентификаторы общающихся сторон (например, IP-адреса) могут передаваться в открытом виде или шифроваться. Поскольку ISAKMP предусматривает функционирование в режиме клиент/сервер (т.е. ISAKMP-сервер может формировать контекст для клиента), сокрытие идентификаторов в определенной степени повышает защищенность от пассивного прослушивания сети.

Системы, реализующие IPSec, должны поддерживать две базы данных:

- базу данных политики безопасности (Security Policy Database, SPD);

- базу данных протокольных контекстов безопасности (Security Association Database, SAD).

Все IP-пакеты (входящие и исходящие) сопоставляются с упорядоченным набором правил политики безопасности. При сопоставлении используется фигурирующий в каждом правиле селектор - совокупность анализируемых полей сетевого уровня и более высоких протокольных уровней. Первое подходящее правило определяет дальнейшую судьбу пакета:

- пакет может быть ликвидирован;
- пакет может быть обработан без участия средств IPSec;
- пакет должен быть обработан средствами IPSec с учетом набора протокольных контекстов, ассоциированных с правилом.

Таким образом, системы, реализующие IPSec, функционируют как межсетевые экраны, фильтруя и преобразуя потоки данных на основе предварительно заданной политики безопасности [1].

4 Определение форматов структур данных, используемых для обработки сетевого трафика защищенных каналов передачи данных

4.1 Структуры данных для обработки входящего трафика коммутатора сообщений абонентской сети

Пакет, принимаемый из канала Ethernet, помещается в буфер DMA по смещению 42 байта. Структура полей буфера с полученным пакетом представлена на рисунке 7. Используется адресация процессорного ядра.

	+0x00			+0x01	
0x00	ID источника		Смещение данных	Размер данных	
0x02	Порядковый номер пакета				
0x04					
0x06					
0x08					
0x0A					
0x0C					
0x0E					
0x10					
0x12					
0x14					
0x16	MAC-адрес получателя				
0x18	MAC-адрес отправителя				
0x1A				Тип протокола	
0x1C	Версия IP	Длина заголовка	Тип сервиса	Длина IP пакета	
0x1E	Идентификатор пакета IP			Флаги (3 бита)	Смещение фрагмента
0x20	Время жизни пакета		Тип протокола	Контрольная сумма заголовка	
0x22	IP-адрес источника				
0x24	IP-адрес получателя				

Рисунок 7 – Структура полей буфера коммутатора сообщений абонентской сети с принятым из Ethernet пакетом

Размер считываемой из буфера структуры составляет 74 байта. Структура включает:

- заголовок буфера DMA;
- заголовок Ethernet пакета;

- заголовок IP пакета.

Для обработки пакета используется запись таблицы сеансов. Структура записи представлена на рисунке 8.

	+0x00	+0x01
0x00	Флаг активности записи таблицы	Идентификатор ключа шифрования
0x02	IP-адрес получателя	
0x04	Идентификатор параметров безопасности ESP	
0x06	Параметр MTU канала зашифрованных данных	

Рисунок 8 – Структура полей записи таблицы сеансов коммутатора сообщений абонентской сети

Размер записи таблицы составляет 64 байта, включая служебные поля.

С использованием таблицы сеансов изменяется структура данных в буфере пакета:

- удаляется заголовок Ethernet;
- добавляется заголовок ESP;
- добавляется заголовок БФП;
- корректируется заголовок буфера.

В заголовок ESP помещается синхропосылка, формируемая генератором псевдослучайной последовательности.

Заголовок БФП содержит служебную информацию для шифратора:

- порядковый номер пакета – используется схемой сравнения для синхронизации пакетов основного и дублирующего БФП;
- маска пакета – используется для маскирования данных при передаче в БФП;
- идентификатор ключа шифрования – номер ключа в ОЗУ ключей БФП;
- поле контроля – используется для задания статуса пакета (для пакетов с данными 0).

Структура буфера, передаваемого в БФП, представлена на рисунке 9.

	+0x00		+0x01		
0x00	ID источника	Смещение данных	Размер данных		
0x02	Порядковый номер пакета				
0x04					
0x06					
0x08	Идентификатор пакета				
0x0A	Маска пакета				
0x0C	Идентификатор ключа шифрования		Поле контроля шифрования		
0x0E	Идентификатор параметров безопасности ESP				
0x10	Порядковый номер пакета ESP				
0x12	Синхропосылка				
0x14					
0x16					
0x18					
0x1A	Имитовставка				
0x1C	Версия IP	Длина заголовка	Тип сервиса	Длина IP пакета	
0x1E	Идентификатор пакета IP			Флаги (3 бита)	Смещение фрагмента
0x20	Время жизни пакета		Тип протокола	Контрольная сумма заголовка	
0x22	IP-адрес источника				
0x24	IP-адрес получателя				

Рисунок 9 – Структура полей буфера коммутатора сообщений абонентской сети с пакетом, передаваемого в БФП

4.2 Структуры данных для обработки исходящего трафика коммутатора сообщений абонентской сети

Структура полей буфера с пакетом, принятым из БФП, представлена на рисунке 10.

Пакет помещается в буфер по смещению 48 байт. Размер считываемой из буфера структуры составляет 104 байта. Структура включает:

- заголовок буфера DMA;
- заголовок БФПпакета;
- заголовок ESP пакета;
- заголовок IP пакета.

	+0x00		+0x01
0x00	ID источника	Смещение данных	Размер данных
0x02	Порядковый номер пакета		
0x04			
0x06			
0x08			
0x0A			
0x0C			
0x0E			
0x10			
0x12			
0x14			
0x16			
0x18	Идентификатор пакета		
0x1A	Маска пакета		
0x1C	Идентификатор ключа шифрования		Поле контроля шифрования
0x1E	Идентификатор параметров безопасности ESP		
0x20	Порядковый номер пакета ESP		
0x22	Синхропосылка		
0x24			
0x26			
0x28			
0x2A	Имитовставка		
0x2C	Версия IP	Длина заголовка	Тип сервиса
0x2E	Идентификатор пакета IP		Длина IP пакета
0x30	Время жизни пакета		Флаги (3 бита)
0x32	Тип протокола		Смещение фрагмента
0x34	Контрольная сумма заголовка		
0x36	IP-адрес источника		
0x38	IP-адрес получателя		

Рисунок 10 – Структура полей буфера коммутатора сообщений абонентской сети с пакетом, принятым из БФП

Для обработки пакета используется запись таблицы маршрутов. Структура записи представлена на рисунке 12.

Структура буфера, передаваемого в Ethernet, представлена на рисунке 11.

	+0x00	+0x01
0x00	Флаг активности записи таблицы	Идентификатор ключа шифрования
0x02	MAC-адрес получателя	
0x04		
0x06	MAC-адрес отправителя	
0x08	IP-адрес отправителя	
0x0A	IP-адрес получателя	

Рисунок 11 – Структура записи таблицы маршрутов коммутатора сообщений абонентской сети

	+0x00		+0x01
0x00	ID источника	Смещение данных	Размер данных
0x02	Порядковый номер пакета		
0x04			
0x06			
0x08			
0x0A			
0x0C			
0x0E			
0x10			
0x12			
0x14			
0x16			
0x18			
0x1A			
0x1C			
0x1E			
0x20			
0x22			
0x24			
0x26	MAC-адрес получателя		
0x28	MAC-адрес отправителя		
0x2A			Тип протокола
0x2C	Версия IP	Длина заголовка	Тип сервиса
0x2E	Идентификатор пакета IP		Длина IP пакета
0x30	Время жизни пакета		Флаги (3 бита)
0x32	IP-адрес источника		Смещение фрагмента
0x34	IP-адрес получателя		Контрольная сумма заголовка

Рисунок 12 – Структура полей буфера коммутатора сообщений абонентской сети с пакетом, передаваемого в Ethernet

4.3 Структуры данных для обработки входящего трафика коммутатора сообщений сети зашифрованной информации

Структура полей буфера с пакетом, принятым из Ethernet, представлена на рисунке 13.

	+0x00			+0x01	
0x00	ID источника		Смещение данных		Размер данных
0x02	Порядковый номер пакета				
0x04					
0x06					
0x08					
0x0A					
0x0C					
0x0E	MAC-адрес получателя				
0x10	MAC-адрес отправителя				
0x12				Тип протокола	
0x14	Версия IP	Длина заголовка	Тип сервиса	Длина IP пакета	
0x16	Идентификатор пакета IP			Флаги (3 бита)	Смещение фрагмента
0x18	Время жизни пакета		Тип протокола	Контрольная сумма заголовка	
0x1A	IP-адрес источника				
0x1C	IP-адрес получателя				
0x1E	Идентификатор параметров безопасности ESP				
0x20	Порядковый номер пакета ESP				
0x22	Синхропосылка				
0x24					
0x26					
0x28					
0x2A	Имитовставка				

Рисунок 13 – Структура полей буфера коммутатора сообщений сети зашифрованной информации с принятым из Ethernet пакетом

Пакет помещается в буфер по смещению 26 байт. Размер считываемой из буфера структуры составляет 84 байта.

Структура записи таблицы сеансов представлена на рисунке 14.

	+0x00	+0x01
0x00	Флаг активности записи таблицы	Идентификатор ключа шифрования
0x02	IP-адрес отправителя	
0x04	IP-адрес получателя	
0x06	Идентификатор параметров безопасности ESP	
0x08	Порядковый номер пакета ESP	
0x0A	Параметр MTU канала зашифрованных данных	

Рисунок 14 – Структура полей записи таблицы сеансов коммутатора сообщений сети зашифрованной информации

Структура буфера, передаваемого в БФП, представлена на рисунке 15.

	+0x00		+0x01
0x00	ID источника	Смещение данных	Размер данных
0x02	Порядковый номер пакета		
0x04			
0x06			
0x08			
0x0A			
0x0C			
0x0E			
0x10			
0x12			
0x14			
0x16			
0x18	Идентификатор пакета		
0x1A	Маска пакета		
0x1C	Идентификатор ключа шифрования	Поле контроля шифрования	
0x1E	Идентификатор параметров безопасности ESP		
0x20	Порядковый номер пакета ESP		
0x22	Синхропосылка		
0x24			
0x26			
0x28			
0x2A	Имитовставка		

Рисунок 15 – Структура полей буфера коммутатора сообщений сети зашифрованной информации с пакетом, передаваемого в БФП

4.4 Структуры данных для обработки исходящего трафика коммутатора сообщений сети зашифрованной информации

Структура буфера с пакетом из БФП аналогична рисунку 15.

Пакет помещается в буфер по смещению 48 байт. Размер считываемой из буфера структуры составляет 84 байта. Структура включает:

- заголовок буфера DMA;
- заголовок БФП пакета;
- заголовок ESP пакета.

Для обработки пакета используется запись таблицы маршрутов. Структура записи таблицы сеансов представлена на рисунке 16.

	+0x00	+0x01
0x00	Флаг активности записи таблицы	Идентификатор ключа шифрования
0x02	Идентификатор параметров безопасности ESP	
0x04	Порядковый номер пакета ESP	
0x06	MAC-адрес получателя	
0x08		
0x0A	MAC-адрес отправителя	
0x0C	IP-адрес отправителя	
0x0E	IP-адрес получателя	

Рисунок 16 – Структура полей записи таблицы маршрутов коммутатора сообщений сети зашифрованной информации

С использованием таблицы изменяется структура данных в буфере пакета:

- удаляется заголовок БФП;
- добавляется заголовок IP;
- добавляется заголовок Ethernet;
- корректируется заголовок буфера.

В заголовок ESP помещается значение порядкового номера пакета в текущем сеансе, взятое из записи таблицы маршрутов.

Структура буфера, передаваемого в канал Ethernet, представлена на рисунке 17.

	+0x00			+0x01	
0x00	ID источника		Смещение данных	Размер данных	
0x02	Порядковый номер пакета				
0x04					
0x06					
0x08					
0x0A					
0x0C					
0x0E	MAC-адрес получателя				
0x10	MAC-адрес отправителя				
0x12				Тип протокола	
0x14	Версия IP	Длина заголовка	Тип сервиса	Длина IP пакета	
0x16	Идентификатор пакета IP			Флаги (3 бита)	Смещение фрагмента
0x18	Время жизни пакета		Тип протокола	Контрольная сумма заголовка	
0x1A	IP-адрес источника				
0x1C	IP-адрес получателя				
0x1E	Идентификатор параметров безопасности ESP				
0x20	Порядковый номер пакета ESP				
0x22	Синхропосылка				
0x24					
0x26					
0x28					
0x2A	Имитовставка				

Рисунок 17 – Структура полей буфера коммутатора сообщений сети зашифрованной информации с пакетом, передаваемого в Ethernet

5 Разработка алгоритма и структуры аппаратной части модуля

В соответствии с механизмом обработки пакетов в коммутаторах сообщений, разрабатываемый модуль должен обеспечивать:

- взаимодействие с контроллером буферов;
- взаимодействие с памятью DDR;
- проверку корректности принятого пакета;
- обработку заголовков пакетов при передаче в БФП и канал

Ethernet.

Предлагаемая структура модуля представлена на рисунке 18.

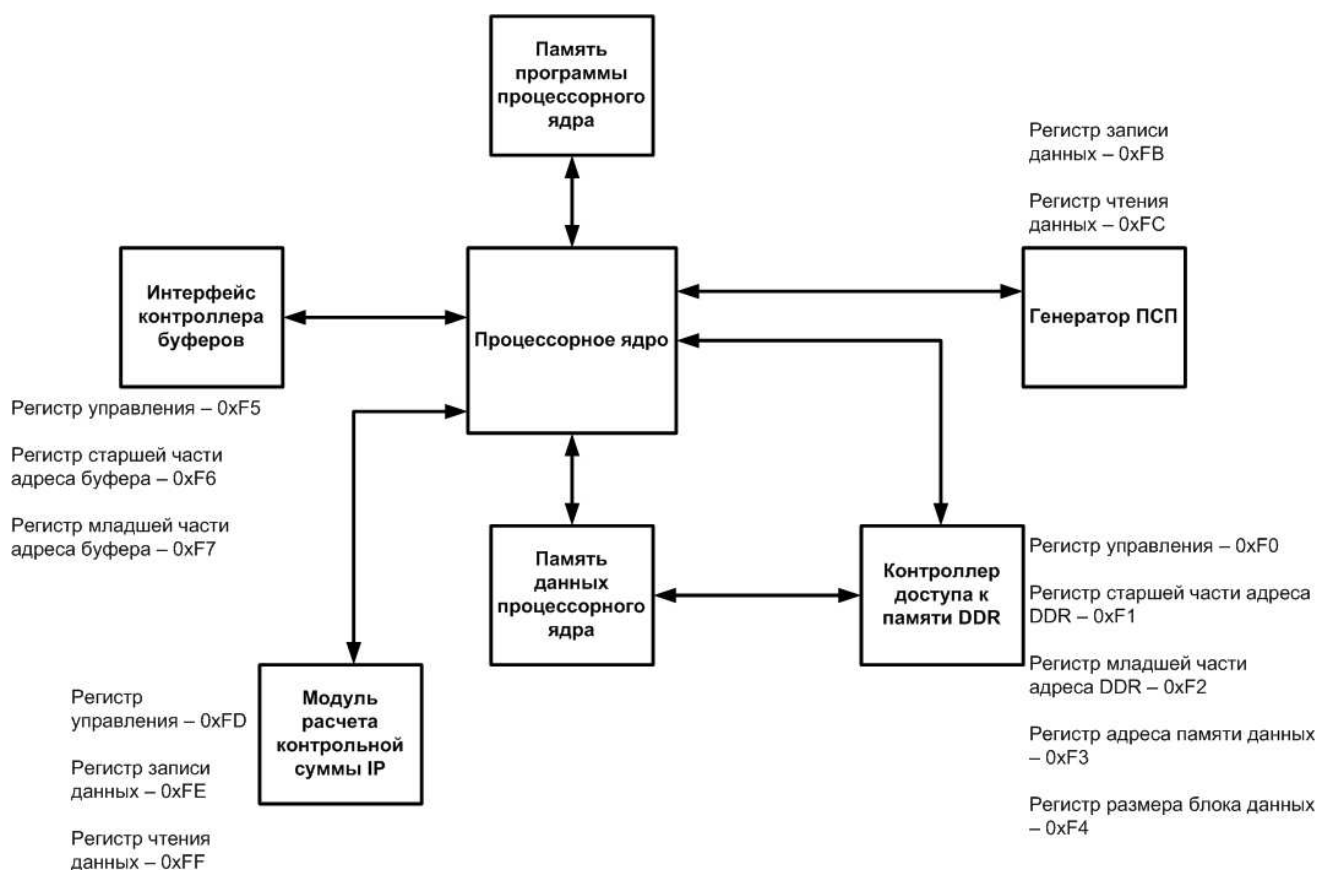


Рисунок 18 – Структура разрабатываемого модуля

В таблице 1 представлена структура разрабатываемого модуля с указанием адресов регистров, входящих в его состав.

Таблица 1 – Структура разрабатываемого модуля с указанием адресов регистров, входящих в его состав

Структура разрабатываемого модуля	Регистры, входящие в состав модуля	Адреса регистров
Интерфейс контроллера буферов	Регистр управления	0xF5
	Регистр старшей части адреса буфера	0xF6
	Регистр младшей части адреса буфера	0xF7
Модуль расчета контрольной суммы IP	Регистр управления	0xFD
	Регистр записи данных	0xFE
	Регистр чтения данных	0xFF
Генератор ПСП	Регистр записи данных	0xFB
	Регистр чтения данных	0xFC
Контроллер доступа к памяти DDR	Регистр управления	0xF0
	Регистр старшей части адреса	0xF1
	Регистр младшей части адреса	0xF2
	Регистр адреса памяти данных	0xF3
	Регистр размера блока данных	0xF4

Взаимодействие с контроллером буферов обеспечивается соответствующим интерфейсом, который реализует:

- прием указателя на буфер с пакетом;
- передачу указателя на буфер с пакетом в очередь ЦПУ;
- передачу указателя на буфер с пакетом в очередь буферов на очистку;
- передачу указателя на буфер с пакетом в очередь передачи канального DMA.

Интерфейс включает 3 16-разрядных регистра:

- регистр управление – содержит флаги статуса и управления;
- регистр старшей части адреса – содержит старшую часть адреса буфера в памяти DDR;
- регистр младшей части адреса – содержит младшую часть адреса буфера в памяти DDR.

Флаги статуса указывают наличие активного буфера на обработку (адрес буфера помещается в регистры адреса). Флаги управления задают направление передачи активного буфера.

Взаимодействие с памятью DDR обеспечивает контроллер доступа, который реализует:

- чтение данных из выделенной области памяти DDR и запись во внутреннюю память BRAM модуля;
- чтение из внутренней памяти BRAM и запись данных в выделенную область памяти DDR.

Контроллер доступа к памяти DDR включает 5 регистров:

- регистр управления – содержит флаги статуса и управления;
- регистр старшей части адреса DDR – содержит старшую часть начального адреса в памяти DDR для выполняемой операции;
- регистр младшей части адреса DDR – содержит младшую часть начального адреса в памяти DDR для выполняемой операции;
- регистр адреса во внутренней памяти – содержит начальный адрес во внутренней памяти для выполняемой операции;
- регистр размера блока данных – содержит размер блока данных, пересылаемого в текущей операции, но не превышающего 256 байт.

Обработка пакета выполняется процессорным ядром. Исполняемая программа помещается в память программ процессорного ядра, реализованную на блоке памяти BRAM, доступном только процессору.

Данные, считываемые из буфера, и информация, необходимая в процессе обработки, помещаются в память данных. Память данных реализуется на блоке памяти BRAM, организованном по двухпортовой схеме, к которой подключаются контроллер доступа к памяти DDR и шина данных процессорного ядра.

Для реализации функций по обработке пакетов в состав модуля вводятся два специализированных блока:

- генератор псевдослучайной последовательности;
- модуль расчета контрольной суммы IP.

Выделение этих функций в аппаратные блоки позволяет ускорить процесс обработки пакетов и сократить размер программного кода.

Генератор псевдослучайной последовательности включает рекуррентный регистр, который реализует многочлен:

$$x^{32} + x^7 + x^5 + x^3 + x^2 + x + 1.$$

Генератор включает 2 регистра:

- регистр записи – используемый для записи значений шума в генератор;
- регистр чтения – используемый для чтения сформированной псевдослучайной последовательности.

Модуль расчета контрольной суммы IP, реализует сложение 16-битных поразрядных дополнений для заданного блока данных. Начало блока данных задается процессорным ядром.

Включает 3 регистра:

- регистр управления – используется для задания начала расчета контрольной суммы;
- регистр записи – используется для записи данных, для которых рассчитывается контрольная сумма;
- регистр чтения – используется для чтения значения рассчитанной контрольной суммы.

При необходимости структура модуля может быть дополнена другими специализированными модулями, например, расчета контрольной суммы CRC.

6 Реализация аппаратной части модуля на языке VHDL

Язык VHDL является фактически международным стандартом в области автоматизации проектирования цифровых систем, это входной язык многих современных систем автоматизированного проектирования (САПР) как заказных, так и ПЛИС [4].

Язык VHDL предназначен для решения комплекса задач в ходе проектирования и применения цифровых систем, их аппаратных средств, в том числе:

- описания структуры системы, декомпозиции системы на подсистемы, спецификации связей и взаимодействия подсистем;
- спецификации функционирования системы, узлов, блоков, реализуемых функций. Спецификация дается в алгоритмической форме, с использованием привычных современному специалисту программных конструкций алгоритмического языка, включающих в себя спецификацию временного поведения сигналов и блоков;
- моделирования системы и ее работы на основе четкой спецификации структуры системы, а также функционирования ее компонентов;
- синтеза схемотехнической реализации системы, автоматической генерации детальной структуры на основе строгой спецификации системы на языке VHDL – спецификации на более абстрактном уровне [5].

Архитектура модуля реализована на языке описания аппаратуры VHDL.

Верхний уровень модуля описан в файле `link_processor.vhd`. Он включает компоненты модуля:

- `multi_mem_processor` (файл `multi_mem_processor.vhd`) – процессорпрограммной памятью;
- `rnd` (файл `rec_reg.vhd`) – генератор псевдослучайной последовательности;

- ipcsun_calc (файл ipcsun_calc.vhd) – компонент расчета контрольной суммы IP-заголовка;
- lp_buffers_controller (файл lp_buffers_controller.vhd) – компонент взаимодействия с контроллером буферов;
- lp_dma_controller (файл lp_dma_controller.vhd) – контроллер DMA модуля.

Файл описывает систему адресации модуля, а также взаимодействия между компонентами.

Файл multi_mem_processor.vhd описывает подключение программной памяти к процессору. Программная память поддерживает многостраничную организацию на 4 программы. Выбор страницы с программой реализуется через управляющий регистр.

Файл res_reg.vhd содержит структуру генератора псевдослучайной последовательности. Включает рекуррентный регистр разрядностью 32 бита. Регистр поддерживает загрузку начального заполнения по сигналу сброса и подгрузку шума для изменения заполнения регистра:

```
SHIFT_REG_PROC : process (CLK, RST)
begin
    if (RST = '1') then
        shift_reg <= C_INIT;
    elsif (rising_edge(CLK)) then
        if (WE = '1') then
            shift_reg(C_DWIDTH - 1 downto 0) <= DIN;
        else
            Shift_loop : for j in 0 to C_SR_SIZE - 2 loop
                shift_reg(j) <= shift_reg(j+1);
            end loop Shift_loop;
            shift_reg(C_SR_SIZE - 1) <= rnd_bit;
        end if;
    end if;
end process SHIFT_REG_PROC;

rnd_bit <= '1' xor shift_reg(0) xor shift_reg(1) xor shift_reg(2)
        xor shift_reg(4) xor shift_reg(6) xor shift_reg(31);
```

Сформированное число, разрядностью 128 бит, хранится в регистре **ВЫХОДНЫХ ДАННЫХ**:

```
RND_REG_PROC : process (CLK, RST)
begin
    if (RST = '1') then
```

```

        rnd_reg <= (others => '0');
    elsif (rising_edge(CLK)) then
        Shift_loop : for j in 1 to C_RND_SIZE - 1 loop
            rnd_reg(j) <= rnd_reg(j-1);
        end loop Shift_loop;
        rnd_reg(0) <= rnd_bit;
    end if;
end process RND_REG_PROC;

```

Файл `ipcsun_calc.vhd` описывает регистр-накопитель, в котором реализуется сложение 16-битных слов данных согласно алгоритму расчета контрольной суммы заголовка IP:

```

CSUM_CALCULATE_PROC : process (CLK, RST)
    variable csum_acc : std_logic_vector(16 downto 0);
begin
    if (RST = '1') then
        csum_value <= (others => '0');
    elsif (rising_edge(CLK)) then
        if (WE = '1') then
            csum_acc := ('0' & csum_value) + ('0' & DIN);
            csum_acc := ('0' & csum_acc(15 downto 0)) + (x"0000" &
csum_acc(16));
            csum_acc := ('0' & csum_acc(15 downto 0)) + (x"0000" &
csum_acc(16));
            csum_value <= csum_acc(15 downto 0);
        elsif (NEXT_CSUM = '1') then
            csum_value <= (others => '0');
        end if;
    end if;
end process CSUM_CALCULATE_PROC;

DOUT<= notcsum_value;

```

Файл `lp_buffers_controller.vhd` содержит регистры статуса и управления, а так же формирование сигналов управления для контроллера буферов.

Файл `lp_dma_controller.vhd` содержит ядро контроллера (файл `lp_dma_core.vhd`) и описывает подключение регистров к управляющим сигналам ядра.

Ядро включает две машины состояний, задающих состояние ядра и состояние интерфейса с памятью DDR:

```

type DMA_STATEMACHINE is (
    IDLE_ST,
    PORT_REQ_ST,
    SDRAM_READ_ST,
    BRAM_WRITE_ST,
    BRAM_READ_ST,
    SDRAM_WRITE_ST,
    COMPLETE_ST

```

```

);

type MCB_STATEMACHINE is (
    IDLE_ST,
    READ_REQ_ST,
    READ_WAIT_ST,
    READ_COMP_ST,
    WRITE_REQ_ST,
    WRITE_WAIT_ST,
    WRITE_COMP_ST
);

```

Процессы описывают изменение состояний и выполнение операций.

Исходные коды аппаратной части модуля на языке VHDL приведены в приложении А.

7 Разработка алгоритмов программных частей модуля

7.1 Алгоритм обработки входящего трафика коммутатора сообщений абонентской сети

Обработка пакета, приходящего со стороны канала Ethernet коммутатор сообщений абонентской сети, включает следующие действия:

- а) ожидание получения буфера DMA на обработку;
- б) чтение адреса буфера;
- в) чтение структуры данных с обрабатываемыми заголовками из памяти DDR во внутреннюю память;
- г) проверка типа протокола в заголовке Ethernet, если тип не соответствует IP, выполняется переход к пункту аа);
- д) проверка версии протокола IP, если версия отлична от 4, выполняется переход к пункту аб);
- е) проверка длины заголовка IP, если длина не равна 20 байт (заголовок без опций), выполняется переход к пункту аб);
- ж) проверка контрольной суммы заголовка IP, если контрольная сумма неверна, выполняется переход к пункту аб);
- и) проверка длины пакета IP, если длина больше размера реально принятых данных, выполняется переход к пункту аб);
- к) декремент поля «Время жизни пакета»;
- л) проверка поля «Время жизни пакета» на 0, если поле равно 0 выполняется, переход к пункту аб);
- м) корректировка контрольной суммы заголовка для нового значения поля «Время жизни пакета»;
- н) расчет указателя на запись таблицы сеансов;
- п) чтение записи таблицы сеансов;
- р) проверка флага активности таблицы, если флаг не установлен, выполняется переход к пункту аа);
- с) проверка IP-адреса получателя пакета по таблице, если адреса не совпадают, выполняется переход к пункту аа);

- т) проверка размера IP-пакета на mtu сеанса, если пакет больше значения mtu, выполняется переход к пункту aa);
- у) дополнение пакета до кратности 16 байт;
- ф) удаление заголовка Ethernet;
- х) формирование заголовка ESP;
- ц) запись значения синхропосылки с генератора ПСП в заголовок ESP;
- ш) формирование заголовка БФП;
- щ) корректировка размера данных в заголовке буфера;
- э) корректировка смещения данных в заголовке буфера;
- ю) запись структуры данных с обрабатываемыми заголовками в память DDR из внутренней памяти;
- я) передача указателя на буфер в очередь буферов на отправку в БФП, переход к пункту а);
- aa) передача указателя на буфер в очередь буферов ЦП, переход к пункту а);
- аб) передача указателя на буфер в очередь буферов на очистку, переход к пункту а).

Алгоритм обработки входящего трафика коммутатора сообщений абонентской сети в виде блок-схемы представлен в приложении Б.

7.2 Алгоритм обработки исходящего трафика коммутатора сообщений абонентской сети

Обработка пакета, приходящего со стороны канала БФП коммутатор сообщений абонентской сети, включает следующие действия:

- а) ожидание получения буфера DMA на обработку;
- б) чтение адреса буфера;
- в) чтение структуры данных с обрабатываемыми заголовками из памяти DDR во внутреннюю память;
- г) проверка статуса расшифрования пакета, если статус не 0, выполняется переход к пункту ю);

- д) проверка версии протокола IP, если версия отлична от 4, выполняется переход к пункту я);
- е) проверка длины заголовка IP, если длина не равна 20 байт (заголовок без опций), выполняется переход к пункту я);
- ж) проверка контрольной суммы заголовка IP, если контрольная сумма неверна, выполняется переход к пункту я);
- и) проверка длины пакета IP, если длина больше размера реально принятых данных, выполняется переход к пункту я);
- к) декремент поля «Время жизни пакета»;
- л) проверка поля «Время жизни пакета» на 0, если поле равно 0, выполняется переход к пункту я);
- м) корректировка контрольной суммы заголовка для нового значения поля «Время жизни пакета»;
- н) расчет указателя на запись таблицы маршрутов;
- п) чтение записи таблицы маршрутов;
- р) проверка флага активности таблицы, если флаг не установлен, выполняется переход к пункту ю);
- с) проверка идентификатора ключа расшифрования пакета, если значение из заголовка БФП не совпадает со значением из таблицы, выполняется переход к пункту ю);
- т) проверка IP-адреса получателя пакета по таблице, если адреса не совпадают, выполняется переход к пункту ю);
- у) удаление заголовка БФП;
- ф) удаление заголовка ESP;
- х) формирование заголовка Ethernet;
- ц) корректировка размера данных в заголовке буфера;
- ш) корректировка смещения данных в заголовке буфера;
- щ) запись структуры данных с обрабатываемыми заголовками в память DDR из внутренней памяти;

э) передача указателя на буфер в очередь буферов на отправку в БФП, переход к пункту а);

ю) передача указателя на буфер в очередь буферов ЦП, переход к пункту а);

я) передача указателя на буфер в очередь буферов на очистку, переход к пункту а).

Алгоритм обработки исходящего трафика коммутатора сообщений абонентской сети в виде блок-схемы представлен в приложении В.

7.3 Алгоритм обработки входящего трафика коммутатора сообщений сети зашифрованной информации

Обработка пакета, приходящего со стороны канала Ethernet коммутатор сообщений сети зашифрованной информации, включает следующие действия:

а) ожидание получения буфера DMA на обработку;

б) чтение адреса буфера;

в) чтение структуры данных с обрабатываемыми заголовками из памяти DDR во внутреннюю память;

г) проверка типа протокола в заголовке Ethernet, если тип не соответствует IP, выполняется переход к пункту я);

д) проверка версии протокола IP, если версия отлична от 4, выполняется переход к пункту аа);

е) проверка длины заголовка IP, если длина не равна 20 байт (заголовок без опций), выполняется переход к пункту аа);

ж) проверка контрольной суммы заголовка IP, если контрольная сумма неверна, выполняется переход к пункту аа);

и) проверка длины пакета IP, если длина больше размера реально принятых данных, выполняется переход к пункту аа);

к) проверка кратности содержимого ESP пакета 16 байтам, если пакета не кратен, выполняется переход к пункту аа);

л) декремент поля «Время жизни пакета»;

- м) проверка поля «Время жизни пакета» на 0, если поле равно 0 выполняется, переход к пункту аа);
- н) расчет указателя на запись таблицы сеансов;
- п) чтение записи таблицы сеансов;
- р) проверка флага активности таблицы, если флаг не установлен, выполняется переход к пункту я);
- с) проверка IP-адресов пакета по таблице, если адреса не совпадают, выполняется переход к пункту я);
- т) проверка индекса заголовка ESP, если индекс пакета не совпадает с индексом таблицы, выполняется переход к пункту я);
- у) проверка размера IP-пакета на mtu сеанса, если пакет больше значения mtu, выполняется переход к пункту я);
- ф) удаление заголовка Ethernet;
- х) удаление заголовка IP;
- ц) формирование заголовка БФП;
- ш) корректировка размера данных в заголовке буфера;
- щ) корректировка смещения данных в заголовке буфера;
- э) запись структуры данных с обрабатываемыми заголовками в память DDR из внутренней памяти;
- ю) передача указателя на буфер в очередь буферов на отправку в БФП, переход к пункту а);
- я) передача указателя на буфер в очередь буферов ЦП, переход к пункту а);
- аа) передача указателя на буфер в очередь буферов на очистку, переход к пункту а).

Алгоритм обработки входящего трафика коммутатора сообщений сети зашифрованной информации в виде блок-схемы представлен в приложении Г.

7.4 Алгоритм обработки исходящего трафика коммутатора сообщений сети зашифрованной информации

Обработка пакета, приходящего со стороны канала БФП коммутатор сообщений сети зашифрованной информации, включает следующие действия:

- а) ожидание получения буфера DMA на обработку;
- б) чтение адреса буфера;
- в) чтение структуры данных с обрабатываемыми заголовками из памяти DDR во внутреннюю память;
- г) проверка статуса расшифрования пакета, если статус не 0, выполняется переход к пункту у);
- д) расчет указателя на запись таблицы маршрутов;
- е) чтение записи таблицы маршрутов;
- ж) проверка флага активности таблицы, если флаг не установлен, выполняется переход к пункту у);
- и) проверка идентификатора ключа расшифрования пакета, если значение из заголовка БФП не совпадает со значением из таблицы, выполняется переход к пункту у);
- к) проверка индекса заголовка ESP, если индекс пакета не совпадает с индексом таблицы, выполняется переход к пункту у);
- л) удаление заголовка БФП;
- м) формирование заголовка IP;
- н) формирование заголовка Ethernet;
- п) корректировка размера данных в заголовке буфера;
- р) корректировка смещения данных в заголовке буфера;
- с) запись структуры данных с обрабатываемыми заголовками в память DDR из внутренней памяти;
- т) передача указателя на буфер в очередь буферов на отправку в БФП, переход к пункту а);

у) передача указателя на буфер в очередь буферов ЦП, переход к пункту а).

Алгоритм обработки исходящего трафика коммутатора сообщений сети зашифрованной информации в виде блок-схемы представлен в приложении Д.

8 Реализация полученных алгоритмов программных частей модуля на языке ассемблер процессорного ядра

Совокупность команд, поддерживаемых микропроцессорным ядром PicoBlaze, можно разбить по функциональному признаку на шесть групп. К первой группе относятся команды, управляющие последовательностью выполнения операций в программе, и команды обработки подпрограмм. В эту группу входят команды переходов JUMP, вызова подпрограмм CALL и возврата из подпрограмм RETURN. Вторую группу образуют логические команды. Эти команды выполняют поразрядные операции «Логическое И» (поразрядное умножение) AND, «Логическое ИЛИ» (поразрядное сложение) OR, «Исключающее ИЛИ» XOR. К этой же группе относится команда загрузки значения в регистр общего назначения LOAD. В третью группу входят арифметические команды. Четвертую группу составляют команды сдвига. Они позволяют реализовать операции логического (арифметического) или циклического сдвига данных, записанных в регистр общего назначения, номер которого указан в тексте команды. Пятая группа включает в себя команды ввода-вывода. Эти команды предназначены для организации обмена данными между регистрами, входящими в блок РОН, и портами ввода-вывода. Последняя группа объединяет команды, используемые для обслуживания прерываний [6].

Исходный текст программы в кодах ассемблера записывается в файл с расширением `psm`, название которого должно содержать не более восьми символов. Из этого файла ассемблер формирует VHDL-описание начального содержимого блочного ОЗУ, исполняющего роль программной памяти. Результирующий файл с расширением `.vhd` включается в состав разрабатываемого проекта и используется далее в процессе синтеза, моделирования, размещения и трассировки в кристалле [6].

Инициализация программной памяти микропроцессорного ядра с помощью набора данных, которые соответствуют машинным кодам

разработанной программы, выполняется автоматически в процессе загрузки конфигурационной последовательности проекта кристалла.

Алгоритмы обработки пакетов, описанные в разделах с 7.1 по 7.4, были реализованы на языке ассемблера.

Регистры контроллеров, участвующих в обработке пакета, поля заголовков и записей таблиц заданы в секции констант.

Для ускорения выполнения процедур программы часть констант размещается в зарезервированных регистрах:

```
Set_masks: call    c, FF      ;
            load    sF, FF      ;
            call    c, 08      ;
            load    sE, 00      ;
            call    c, 01      ;
            load    sD, 00      ;
            call    c, ff      ;
            load    sC, 00      ;
            call    c, 00      ;
            load    sB, ff      ;
            call    c, 45      ;
            load    sA, 00      ;
            call    c, 12      ;
            load    s9, 34      ;
            call    c, 03      ;
            load    s8, FF      ;
```

Основным циклом работы программы является ожидание входящего буфера на обработку:

```
Wait_buf_for_proc : input    s2, BUF_CNTRL;
                   and       s2, BUF_PROC_REQ      ;
                   jump      z, Wait_buf_for_proc;
```

При получении буфера запускаются процедуры анализа пакета, в зависимости от алгоритма программной части модуля.

При получении пакета от контроллера Ethernet выполняется проверка типа протокола:

```
Check_eth_protocol : input    s3, ETH_PROTOCOL      ;
                   xor       s3, sE                  ;
                   jump      nz, Push_buf_to_cpu     ;
```

При получении пакета из шифратора проверяется поле статуса шифрования из заголовка шифратора:

```
Check_zas_status :  input    s3, ZAS_CONTROL      ;
                   or        s3, s0              ;
                   jump     nz, Push_buf_to_cpu  ;
```

Для IP пакета выполняется проверка версии протокола и размера заголовка:

```
Check_ip_version:  input    s3, IP_VERSION      ;
                   and      s3, sC              ;
                   xor      s3, sA              ;
                   jump     nz, Push_buf_to_del  ;
```

Проверяется время жизни пакета. Для этого выполняется декремент поля и проверяется на 0. Полученное значение поля сохраняется для последующей обработки заголовка:

```
Check_ttl :       inputs3, IP_TTL              ;
sub              s3, sD                          ;
                load    s7, s3                  ;
                and     s3, sC                  ;
                jump    z, Push_buf_to_del      ;
```

Проверка размера IP пакета включает сравнение поля из заголовка с размером принятых DMA данных, получаемым из заголовка буфера:

```
Check_ip_length:  input    s3, IP_TOTAL_LEN    ;
                  input    s4, DMA_FRAME_SIZE  ;
                  sub      s4, ETH_HDR_SIZE    ;
; Если размер IP больше принятого пакета, пакет является поврежденным
subs4, s3          ;
jump              c, Push_buf_to_del          ;
```

Проверка контрольной суммы заголовка выполняется с использованием модуля расчета контрольной суммы:

```
Check_checksum :  call     Compute_IP_csum     ;
                  input    s3, CSUM_DOUT       ;
                  or        s3, s0              ;
                  jump     nz, Push_buf_to_del  ;

Compute_IP_csum :  output   s0, CSUM_NEXT      ;
                  load     s2, IP_VERSION      ;
                  load     s5, 0A              ;
```

```

Next_csum_data :   input    s3, (s2)           ;
                   output   s3, CSUM_DIN       ;
                   add      s2, 01             ;
                   sub      s5, 01             ;
                   jump     nz, Next_csum_data  ;

return            ;

```

Для ESP пакетов проверяется тип протокола в заголовке IP:

```

Check_ip_protocol : input    s3, IP_PROTOCOL   ;
                   call     c, 00              ;
                   and      s3, ff             ;
                   xor      s3, 32             ;
                   jump     nz, Push_buf_to_cpu ;

```

Для содержимого ESP пакета проверяется кратность содержимого 16 байтам:

```

Check_rate :      input    s3, IP_TOTAL_LEN   ;
                   sub      s3, IP_HDR_SIZE   ;
                   sub      s3, ESP_HDR_SIZE   ;
                   and      s3, 0F             ;
                   jump     nz, Push_buf_to_del ;

```

Для обращения к таблицам маршрутизации и сеансов необходимо определить номер записи. Номер записи в абонентском коммутаторе сообщений формируется с использованием свертки из IP-адреса получателя пакета:

```

Compute_hash :    inputs3, IP_DST_p2         ;
input    s4, IP_DST_p1           ;
                   add      s4, s9             ;
                   addcy    s3, 00             ;
                   xor      s4, s3             ;
                   xor      s4, s9             ;
                   load     s3, s4             ;
                   sla      s3                ;
                   and      s3, FF             ;
                   add      s3, s4             ;

ands3, s8          ;

```

В коммутаторе сообщений сети зашифрованной информации номер записи определяется младшими 10 битами индекса SPI пакета:

```

Convert_spi_to_ptr: input    s3, ESP_SPI_p2   ;
                   ands3, s8                  ;

```

Полученный номер записи преобразуется в указатель умножением на 64 и сложением с базовым адресом таблицы.

После прочтения записи таблицы, выполняется проверка активности записи таблицы:

```
Check_Active :      inputs3, T_ACTIVE          ;
call      c , 00          ;
xor       s3, 01          ;
jump      nz, Push_buf_to_cpu;
```

Если запись таблицы активна, запускаются процедуры анализа пакета по таблице.

Для пакета ESP, формируемого в абонентском коммутаторе сообщений, выполняется процедура дополнения пакета до кратности 16 байтам с последним байтом, содержащим тип передаваемого пакета. Для пакетов с данными указывается тип «Пакет с данными» равный 4. Остальные байты дополнения пакета устанавливаются равными 0. Дополнение пакета записывается в конец и размер данных в заголовке буфера корректируется на размер дополнения:

```
; Выравнивание пакета до кратности 16 байт
Compute_padding_size :      loads4, PADDING_BASE          ;
; Проверка кратностиразмера IP пакета 16 байтам
inputs3, IP_TOTAL_LEN          ;
and       s3, 0F          ;

                                load      s5, 0F          ;
                                sub       s5, s3          ;
                                load      s3, s5          ;
                                sr0      s5          ;
                                sr0      s5          ;
                                jump      z, Write_type    ;

; Цикл записи дополнения в конец пакета
Padding_loop   :      output  s0, (s4)          ;
                                add       s4, 01          ;
                                output  s0, (s4)          ;
                                add       s4, 01          ;
                                sub      s5, 01          ;
                                jump     nz, Padding_loop   ;

; Запись типа пакета в последний байт дополнения
Write_type   :      output  s0, (s4)          ;
```

```

                                add      s4, 01                ;
                                call      c, 00                ;
                                load      s2, 04                ;
                                outputs2, (s4)                  ;
; Вычисление адреса конца данных IP пакета
Compute_packet_end :          input     s2, DMA_FRAME_SIZE      ;
                                input     s4, DMA_DATA_OFFSET    ;
                                and       s4, sB                  ;
                                adds4, s2                        ;
; Изменение размера данных в заголовке буфера
Frame_resize :              add      s3, 01                ;
                                input     s2, DMA_FRAME_SIZE      ;
                                add       s2, s3                  ;
                                output    s2, DMA_FRAME_SIZE      ;
; Запись конца пакета в память
Write_packet_end :          input     s2, BUF_HADDR            ;
                                output    s2, DMA_HADDR          ;
                                input     s2, BUF_LADDR          ;
                                add       s2, s4                  ;
                                output    s2, DMA_LADDR          ;
                                load      s2, PADDING_BASE        ;
                                output    s2, DMA_BADDR          ;
                                output    s3, DMA_SIZE            ;
                                call      Write_DDR                ;

```

По полученной записи таблицы выполняется формирование заголовков пакета.

При обработке входящего трафика абонентского коммутатора сообщений добавляется ESP заголовок, в который помещается значение синхропосылки, получаемое с генератора псевдослучайной последовательности, и резервируются поля под имитовставку:

```

Insert_ESP_header :          inputs3, T_SPI_p1                ;
                                output    s3, ESP_SPI_p1        ;
                                input     s3, T_SPI_p2          ;
                                output    s3, ESP_SPI_p2        ;
; Резервирования поля под порядковый номер
                                outputs0, ESP_SN_p1              ;
                                output    s0, ESP_SN_p2          ;
; Запись псевдослучайной последовательности в поле синхропосылки
Write_sync :                  load      s4, ESP_SYNC_p1          ;
                                load      s5, 08                  ;

```

```

Next_sync_word :      input    s3, RND_DOUT          ;
                      output    s3, (s4)             ;
                      add       s4, 01               ;
                      sub       s5, 01               ;
                      jump      nz, Next_sync_word    ;

```

; Резервирование поля под битовуюставку

```

output s0, ESP_IM_p1      ;
output s0, ESP_IM_p2      ;

```

Начальное заполнение генератора обновляется шумом, через каждые 128 пакетов:

```

Rnd_refresh_check :    ands1, 7F                    ;
jump      nz, Push_buf_to_tx      ;
; Чтение шума из памяти
Read_noise :          load     s3, 40                ;
                      output    s4, DMA_HADDR        ;
                      load     s3, s0                ;
                      output    s3, DMA_LADDR        ;
                      load     s3, NOISE_BASE        ;
                      output    s3, DMA_BADDR        ;
                      load     s3, NOISE_SIZE        ;
                      output    s3, DMA_SIZE        ;
                      call Read_DDR                  ;
; Запись шума в генератор псевдослучайной последовательности
Rnd_write :           inputs3, NOISE_BASE            ;
output s3, RND_DIN      ;

```

При отправке в шифратор, к пакету добавляется заголовок шифратора:

```

Insert_ZAS_header :   output    s0, ZAS_MASK_p1      ;
                      output    s0, ZAS_MASK_p2      ;
; Копирование порядкового номера пакета из заголовка буфера
inputs3, DMA_SN_p2      ;
output s3, ZAS_PKT_ID_p2      ;
                      input     s3, DMA_SN_p1        ;
                      output    s3, ZAS_PKT_ID_p1      ;
; Копирование номера ключа из таблицы
inputs3, T_ID           ;
output s3, ZAS_KEY_ID      ;
; Установка флагов управления в 0
output s0, ZAS_CONTROL    ;

```

Заголовок DMA модифицируется в соответствии с внесенными изменениями:

```

; Изменениеразмераданных
Frame_size_update      :   input    s3, DMA_FRAME_SIZE      ;
                           add       s3, FR_SIZE_CHANGE      ;
                           output    s3, DMA_FRAME_SIZE      ;
; Изменениесмещенияданных
Frame_offset_update     :   input    s3, DMA_DATA_OFFSET     ;
                           sub       s3, FR_SIZE_CHANGE      ;
                           output    s3, DMA_DATA_OFFSET     ;

```

При отправке в канал Ethernetтак же выполняется проверка на минимально допустимый размер пакета (60 байт):

```

Frame_size_update      :   input    s3, IP_TOTAL_LEN        ;
                           add       s3, ETH_HDR_SIZE        ;
; Проверка размера пакета на минимальный размер (60 байт)
Check_little_frame     :   load      s4, s3                  ;
                           sub       s4, 3c                   ;
                           jump      nc, Write_size           ;
; Установка размера данных для коротких пакетов
callc, 00               ;
loads3, 3c              ;

```

По результатам обработки буфер передается в соответствующий канал и программа возвращается к основному циклу:

```

; Передача буфера в очередь на отправку
Push_buf_to_tx :         load      s2, BUF_TO_TX             ;
                           output   s2, BUF_CNTRL            ;
                           jump     Wait_buf_for_proc        ;
; Передача буфера в очередь на обработку ЦПУ
Push_buf_to_cpu :         load      s2, BUF_TO_CPU           ;
                           output   s2, BUF_CNTRL            ;
                           jump     Wait_buf_for_proc        ;
; Передача буфера в очередь на очистку
Push_buf_to_cpu :         load      s2, BUF_TO_DEL           ;
                           output   s2, BUF_CNTRL            ;
                           jump     Wait_buf_for_proc        ;

```

Исходные коды программприведены в приложении Е.

Для отладки были разработаны модификации программ, не выполняющие удаления буфера. Шаг анализа пакета сохраняется в поле буфера и записывается в память при передаче пакета ЦПУ.

В таблице 2 приведены коды ошибок для программы обработки входящего трафика абонентского коммутатора сообщений.

Таблица 2 – Коды ошибок программы обработки входящего трафика абонентского коммутатора сообщений

Код ошибки	Описание
0x01	Тип протокола Ethernet не соответствует IP
0x02	Версия IP не соответствует 4 либо длина заголовка IP не равна 20 байт
0x03	Время жизни пакета истекло
0x04	Длина IP пакета превышает размер принятых из канала данных
0x05	Неправильная контрольная сумма заголовка IP
0x06	Запись таблицы сеансов неактивна
0x07	Несовпадение адреса получателя из пакета с адресом из таблицы сеансов
0x08	Несоответствие размера пакета с mtu канала к получателю

В таблице 3 приведены коды ошибок для программы обработки исходящего трафика абонентского коммутатора сообщений.

Таблица 3 – Коды ошибок программы обработки исходящего трафика абонентского коммутатора сообщений

Код ошибки	Описание
0x01	Ошибка статуса расшифрования
0x02	Версия IP не соответствует 4 либо длина заголовка IP не равна 20 байт
0x03	Время жизни пакета истекло
0x04	Длина IP пакета превышает размер принятых из канала данных
0x05	Содержимое ESP пакета не кратно 16
0x06	Неправильная контрольная сумма заголовка IP
0x07	Запись таблицы маршрутов неактивна
0x08	Несовпадение номеров ключа из заголовка и таблицы
0x09	Несовпадение адреса получателя из пакета с адресом из таблицы
0x0A	Несоответствие типа пакета «IP-пакет с данными»

В таблице 4 приведены коды ошибок для программы обработки входящего трафика коммутатора сообщений сети зашифрованной информации.

Таблица 4 – Коды ошибок программы обработки входящего трафика коммутатора сообщений сети зашифрованной информации

Код ошибки	Описание
0x01	Тип протокола Ethernet не соответствует IP
0x02	Версия IP не соответствует 4 либо длина заголовка IP не равна 20 байт
0x03	Время жизни пакета истекло
0x04	Длина IP пакета превышает размер данных принятых из канала
0x05	Содержимое ESP пакета не кратно 16
0x06	Неправильная контрольная сумма заголовка IP
0x07	Тип протокола IP не соответствует ESP
0x08	Запись таблицы сеансов неактивна
0x09	Несовпадение адреса получателя из пакета с адресом из таблицы сеансов
0x0A	Несоответствие размера пакета с mtu канала к получателю

В таблице 5 приведены коды ошибок для программы обработки исходящего трафика коммутатора сообщений сети зашифрованной информации.

Таблица 5 – Коды ошибок программы обработки исходящего трафика коммутатора сообщений сети зашифрованной информации

Код ошибки	Описание
0x01	Ошибка статуса шифрования
0x02	Содержимое ESP пакета не кратно 16
0x03	Запись таблицы маршрутов неактивна
0x04	Несовпадение номеров ключа из заголовка и таблицы
0x05	Несовпадение SPI пакета и записи таблицы

9 Тестирование разработанного модуля в составе устройства

Было проведено тестирование разработанного модуля в составе устройства.

В процессе тестирования были исследована реакция разработанных программ на пакеты с некорректными параметрами.

Для такого исследования были использованы модификации программы с включенными функциями отладки. Реакция на некорректные пакеты при этом описывается в таблицах с 2 по 5 из раздела 8.

В качестве примеров приведена реакция программы обработки входящего трафика абонентского коммутатора сообщений на следующие пакеты:

- пакет с истекшим временем жизни, результат показан на рисунке 19;
- пакет, длина IP которого превышает размер принятых из канала данных, результат показан на рисунке 20;
- пакет с неправильной контрольной суммой заголовка IP, результат показан на рисунке 21.

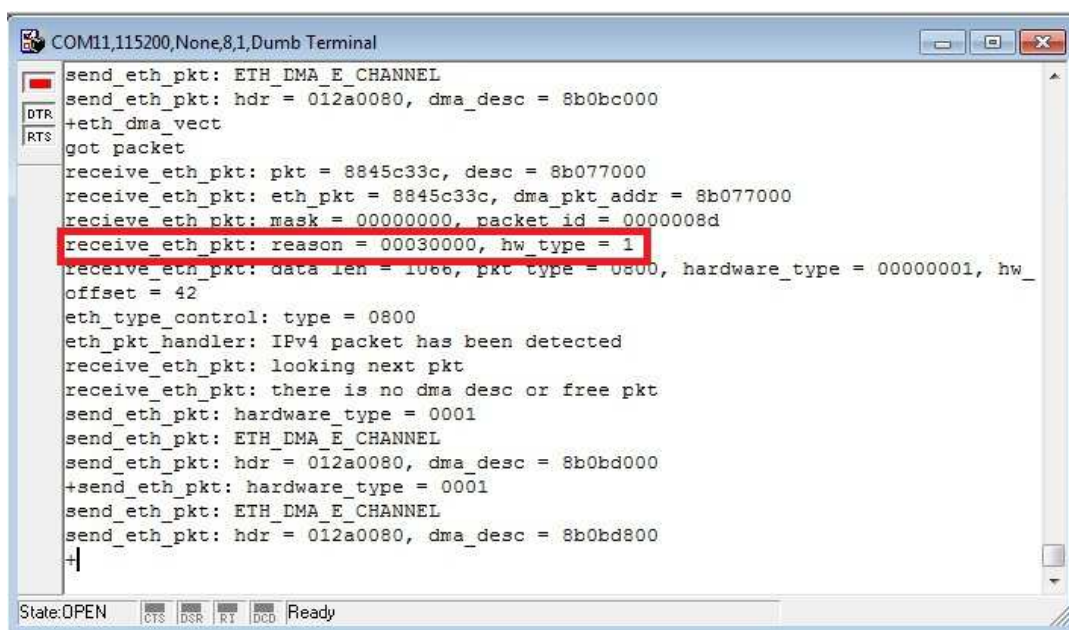


Рисунок 19 – Признак ошибки при обработке пакета с истекшим временем жизни

```
COM11,115200,None,8,1,Dumb Terminal
send_eth_pkt: ETH_DMA_E_CHANNEL
send_eth_pkt: hdr = 012a0080, dma_desc = 8b0a3800
+send_eth_pkt: hardware_type = 0001
send_eth_pkt: ETH_DMA_E_CHANNEL
send_eth_pkt: hdr = 012a0080, dma_desc = 8b0a4000
+send_eth_pkt: hardware_type = 0001
send_eth_pkt: ETH_DMA_E_CHANNEL
send_eth_pkt: hdr = 012a0080, dma_desc = 8b0a4800
+eth_dma_vect
got packet
receive_eth_pkt: pkt = 884700f8, desc = 8b06e000
receive_eth_pkt: eth_pkt = 884700f8, dma_pkt_addr = 8b06e000
receive_eth_pkt: mask = 00000000, packet_id = 00000086
receive_eth_pkt: reason = 00040001, hw_type = 1
receive_eth_pkt: data len = 1066, pkt type = 0800, hardware_type = 00000001, hw_offset = 42
eth_type_control: type = 0800
eth_pkt_handler: IPv4 packet has been detected
receive_eth_pkt: looking next pkt
receive_eth_pkt: there is no dma desc or free pkt

State:OPEN CTS DSR RI DCD Ready
```

Рисунок 20 – Признак ошибки при обработке пакета, длина IP которого превышает размер принятых из канала данных

```
COM11,115200,None,8,1,Dumb Terminal
send_eth_pkt: ETH_DMA_E_CHANNEL
send_eth_pkt: hdr = 012a004a, dma_desc = 8b0b4000
+receive_eth_pkt: looking next pkt
receive_eth_pkt: there is no dma desc or free pkt
eth_dma_vect
got packet
receive_eth_pkt: pkt = 88462afc, desc = 8b075000
receive_eth_pkt: eth_pkt = 88462afc, dma_pkt_addr = 8b075000
receive_eth_pkt: mask = 00000000, packet_id = 0000008c
receive_eth_pkt: reason = 00050001, hw_type = 1
receive_eth_pkt: data len = 74, pkt type = 0800, hardware_type = 00000001, hw_offset = 42
eth_type_control: type = 0800
eth_pkt_handler: IPv4 packet has been detected
my_ip_pkt_handler: ICMP type
send_eth_pkt: hardware_type = 0001
send_eth_pkt: ETH_DMA_E_CHANNEL
send_eth_pkt: hdr = 012a004a, dma_desc = 8b0b5000
+receive_eth_pkt: looking next pkt
receive_eth_pkt: there is no dma desc or free pkt

State:OPEN CTS DSR RI DCD Ready
```

Рисунок 21 – Признак ошибки при обработке пакета с с неправильной контрольной суммой заголовка IP

Полученные результаты подтверждают правильность реакции разработанного модуля на некорректные параметры.

Для трафика с корректными параметрами был зафиксировано успешное прохождение пакетов через устройство шифрования, заключающиеся в наличии ESP-трафика с зашифрованными данными на

выходе МКС и расшифрованного трафика на выходе АКС встречного устройства. Пропускная способность устройства при обработке информации разработанным модулем составляет 250 Мбит/с. При обработке с использованием только ЦПУ пропускная способность составляет 5 Мбит/с, что показывает значительный прирост пропускной способности.

Таким образом, тестирование разработанного модуля в составе устройства IP-шифрования было проведено успешно.

ЗАКЛЮЧЕНИЕ

В процессе дипломного проектирования были рассмотрены протоколы IP и IPSec, был проведен анализ принципов работы и средств приема и обработки трафика в устройстве IP-шифрования. Были разработаны структура и алгоритм работы аппаратного модуля обработки пакетов. Аппаратная часть модуля была реализована на языке VHDL. Были разработаны структуры данных для программной обработки пакетов и алгоритмы программных частей модуля. Полученные алгоритмы программных частей модуля были реализованы на языке ассемблера процессорного ядра. В завершение работы разработанный модуль в составе устройства IP-шифрования был успешно протестирован и введен в эксплуатацию, чему свидетельствует акт внедрения результатов дипломного проекта.

Результатом дипломного проектирования является разработанный программно-аппаратный модуль фильтрации, маршрутизации и подготовки к шифрованию сетевого трафика устройства IP-шифрования.

Техническое задание на выпускную квалификационную работу выполнено в полном объеме.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 RFC 791 – Протокол IP (Internet Protocol)
- 2 Дуглас Э. Камер – Сети TCP-IP. Принципы, протоколы и структура.
Том 1. Четвёртое Издание. – 2003
- 3 Крейг Хант – TCP-IP. Сетевое администрирование, 3-издание – 2008
- 4 П.Н. Бибило – Основы языка VHDL. Издание второе, исправленное и
дополненное. – 2002
- 5 Е. А. Суворова, Ю. Е. Шейнин – Проектирование цифровых систем
на VHDL. – 2003
- 6 Зотов Валерий – PICOBLAZE – семейство восьмиразрядных
микропроцессорных ядер, реализуемых на основе ПЛИС фирмы XILINX. –
Журнал Компоненты и Технологии. Выпуск № 30. – 2003

ПРИЛОЖЕНИЕ А
(обязательное)
ИСХОДНЫЕ КОДЫ АППАРАТНОЙ ЧАСТИ МОДУЛЯ НА
ЯЗЫКЕ VHDL

Файл ipcsum_calc.vhd

```
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_arith.conv_integer;
use ieee.std_logic_arith.conv_std_logic_vector;
use ieee.numeric_std.all;
use ieee.std_logic_misc.all;

entity ipcsum_calc is
  port
  (
    CLK          : in    std_logic;
    RST          : in    std_logic;
    NEXT_CSUM    : in    std_logic;
    DIN          : in    std_logic_vector(15 downto 0);
    WE          : in    std_logic;
    DOUT         : out   std_logic_vector(15 downto 0)
  );
end entity ipcsum_calc;

architecture Behavioral of ipcsum_calc is

  signal  csum_value : std_logic_vector(15 downto 0);

begin

  CSUM_CALCULATE_PROC : process (CLK, RST)
    variable csum_acc : std_logic_vector(16 downto 0);
  begin
    if (RST = '1') then
      csum_value <= (others => '0');
    elsif (rising_edge(CLK)) then
      if (WE = '1') then
        csum_acc := ('0' & csum_value) + ('0' & DIN);
        csum_acc := ('0' & csum_acc(15 downto 0)) + (x"0000" &
csum_acc(16));
        csum_acc := ('0' & csum_acc(15 downto 0)) + (x"0000" &
csum_acc(16));
        csum_value <= csum_acc(15 downto 0);
      elsif (NEXT_CSUM = '1') then
        csum_value <= (others => '0');
      end if;
    end if;
  end process CSUM_CALCULATE_PROC;

  DOUT <= not csum_value;

end architecture Behavioral;
```

Файл rec_reg.vhd

```
library ieee;
```

```

use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_arith.conv_integer;
use ieee.std_logic_arith.conv_std_logic_vector;
use ieee.numeric_std.all;
use ieee.std_logic_misc.all;

library unisim;
use unisim.vcomponents.all;

entity rnd is
  generic
  (
    C_INIT      :    std_logic_vector    := x"1234_5678";
    C_SR_SIZE   :    integer              := 32;
    C_DWIDTH    :    integer              := 16;
    C_RND_SIZE  :    integer              := 128
  );
  port
  (
    CLK         :    in    std_logic;
    RST         :    in    std_logic;
    DIN         :    in    std_logic_vector(C_DWIDTH - 1 downto 0);
    WE         :    in    std_logic;
    DOUT        :    out   std_logic_vector(C_DWIDTH - 1 downto 0);
    RE         :    in    std_logic
  );
end entity rnd;

architecture Behavioral of rnd is

  constant C_CNTRS_SIZE : integer := C_RND_SIZE / C_DWIDTH;

  signal  gen_cntr      : integer range 0 to C_SR_SIZE - 1;
  signal  out_cntr      : integer range 0 to C_CNTRS_SIZE - 1;

  signal  shift_reg     : std_logic_vector(C_SR_SIZE - 1 downto 0);
  signal  rnd_reg       : std_logic_vector(C_RND_SIZE - 1 downto 0);
  signal  out_reg       : std_logic_vector(C_RND_SIZE - 1 downto 0);

  signal  rnd_bit       : std_logic;

begin

  SHIFT_REG_PROC : process (CLK, RST)
  begin
    if (RST = '1') then
      shift_reg <= C_INIT;
    elsif (rising_edge(CLK)) then
      if (WE = '1') then
        shift_reg(C_DWIDTH - 1 downto 0) <= DIN;

```

```

        else
            Shift_loop : for j in 0 to C_SR_SIZE - 2 loop
                shift_reg(j) <= shift_reg(j+1);
            end loop Shift_loop;
            shift_reg(C_SR_SIZE - 1) <= rnd_bit;
        end if;
    end if;
end process SHIFT_REG_PROC;

rnd_bit <= '1' xor shift_reg(0) xor shift_reg(1) xor shift_reg(2)
        xor shift_reg(4) xor shift_reg(6) xor shift_reg(31);

RND_REG_PROC : process (CLK, RST)
begin
    if (RST = '1') then
        rnd_reg <= (others => '0');
    elsif (rising_edge(CLK)) then
        Shift_loop : for j in 1 to C_RND_SIZE - 1 loop
            rnd_reg(j) <= rnd_reg(j-1);
        end loop Shift_loop;
        rnd_reg(0) <= rnd_bit;
    end if;
end process RND_REG_PROC;

OUT_REG_PROC : process (CLK, RST)
begin
    if (RST = '1') then
        out_reg <= (others => '0');
    elsif (rising_edge(CLK)) then
        if (RE = '1' and out_cntr = 0) then
            out_reg <= rnd_reg;
        end if;
    end if;
end process OUT_REG_PROC;

OUT_CNTR_PROC : process (CLK, RST)
begin
    if (RST = '1') then
        out_cntr <= 0;
    elsif (rising_edge(CLK)) then
        if (RE = '1') then
            out_cntr <= (out_cntr + 1) mod C_CNTRS_SIZE;
        end if;
    end if;
end process OUT_CNTR_PROC;

DOUT <= rnd_reg(C_DWIDTH - 1 downto 0) when out_cntr = 0
        else out_reg(C_DWIDTH * (out_cntr + 1) - 1 downto C_DWIDTH * out_cntr);

end architecture Behavioral;

```

Файл lp_dma_core.vhd

```

library ieee;
use ieee.std_logic_1164.all;

```

```

use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_arith.conv_integer;
use ieee.std_logic_arith.conv_std_logic_vector;
use ieee.numeric_std.all;
use ieee.std_logic_misc.all;

library unisim;
use unisim.vcomponents.all;

entity lp_dma_core is
  generic
  (
    C_DATA_PORT_SIZE      :      integer          := 32;
-- разрядность шины данных MCB
    C_MASK_SIZE           :      integer          := 4
-- разрядность маски данных MCB
  );
  port
  (
    CLK                   :  in    std_logic;
    RST                   :  in    std_logic;
    SDRAM_PTR             :  in    std_logic_vector(31 downto 0);
    SDRAM_READ            :  in    std_logic;
    SDRAM_WRITE           :  in    std_logic;
    BRAM_PTR              :  in    std_logic_vector(8 downto 0);
    SIZE                  :  in    std_logic_vector(7 downto 0);
    BUSY                  :  out   std_logic;
    COMPLETE              :  out   std_logic;
    -- BRAM INTERFACE
    BRAM_ADDR             :  out   std_logic_vector(8 downto 0);
    BRAM_WE               :  out   std_logic;
    BRAM_DIN              :  out   std_logic_vector(31 downto 0);
    BRAM_RE               :  out   std_logic;
    BRAM_DOUT             :  in    std_logic_vector(31 downto 0);
    -- MCB ARBITER INTERFACE
    PORT_REQUEST          :  out   std_logic;
    PORT_GRANT            :  in    std_logic;
    PORT_RETURN           :  out   std_logic;
    -- MCB INTERFACE
    CMD_CLK               :  out   std_logic;
    CMD_EN                :  out   std_logic;
    CMD_INSTR             :  out   std_logic_vector(2 downto 0);
    CMD_BL                :  out   std_logic_vector(5 downto 0);
    CMD_BYTE_ADDR         :  out   std_logic_vector(29 downto 0);
    CMD_EMPTY             :  in    std_logic;
    CMD_FULL              :  in    std_logic;
    WR_CLK                :  out   std_logic;
    WR_EN                 :  out   std_logic;
    WR_MASK               :  out   std_logic_vector(C_MASK_SIZE - 1 downto
0);
    WR_DATA               :  out   std_logic_vector(C_DATA_PORT_SIZE - 1
downto 0);
    WR_FULL              :  in    std_logic;
    WR_EMPTY              :  in    std_logic;

```

```

        WR_COUNT          : in    std_logic_vector(6 downto 0);
        WR_UNDEERRUN      : in    std_logic;
        WR_ERROR          : in    std_logic;
        RD_CLK            : out   std_logic;
        RD_EN             : out   std_logic;
        RD_DATA           : in    std_logic_vector(C_DATA_PORT_SIZE - 1
downto 0);
        RD_FULL          : in    std_logic;
        RD_EMPTY         : in    std_logic;
        RD_COUNT          : in    std_logic_vector(6 downto 0);
        RD_OVERFLOW       : in    std_logic;
        RD_ERROR         : in    std_logic
    );
end entity lp_dma_core;

```

architecture Behavioral of lp_dma_core is

```

constant C_WRITE_CMD      : std_logic_vector := "000";
constant C_READ_CMD       : std_logic_vector := "001";

```

```

type DMA_STATEMACHINE is (
    IDLE_ST,
    PORT_REQ_ST,
    SDRAM_READ_ST,
    BRAM_WRITE_ST,
    BRAM_READ_ST,
    SDRAM_WRITE_ST,
    COMPLETE_ST
);

```

```

type MCB_STATEMACHINE is (
    IDLE_ST,
    READ_REQ_ST,
    READ_WAIT_ST,
    READ_COMP_ST,
    WRITE_REQ_ST,
    WRITE_WAIT_ST,
    WRITE_COMP_ST
);

```

```

signal dma_state      : DMA_STATEMACHINE;
signal dma_next_state : DMA_STATEMACHINE;
signal mcb_state      : MCB_STATEMACHINE;

signal sdram_addr_i   : std_logic_vector(31 downto 0);
signal bram_addr_i    : std_logic_vector(8 downto 0);

signal sdram_rd_req   : std_logic;
signal sdram_rd_cntr  : integer;
signal bram_wr_addr   : std_logic_vector(8 downto 0);
signal bram_wr_req    : std_logic;

signal bram_rd_req    : std_logic;
signal bram_rd_addr   : std_logic_vector(8 downto 0);
signal bram_rd_cntr   : integer;

```

```

signal    sdram_wr_req    : std_logic;

signal    req_size        : std_logic_vector(5 downto 0);
signal    read_size       : std_logic_vector(6 downto 0);

signal    first_mask      : std_logic_vector(3 downto 0);
signal    first_word      : std_logic;

begin

    CMD_CLK <= CLK;
    WR_CLK  <= CLK;
    RD_CLK  <= CLK;

DMA_STATE_PROC : process (CLK, RST)
    variable byte_size : integer := 0;
begin
    if (RST = '1') then
        dma_state      <= IDLE_ST;
        dma_next_state <= IDLE_ST;
        sdram_addr_i   <= (others => '0');
        bram_addr_i    <= (others => '0');
    elsif (rising_edge(CLK)) then
        if (dma_state = IDLE_ST) then
            if (SDRAM_WRITE = '1') then
                dma_state      <= PORT_REQ_ST;
                dma_next_state <= BRAM_READ_ST;
                sdram_addr_i   <= SDRAM_PTR;
                bram_addr_i    <= BRAM_PTR;
                byte_size      := conv_integer(SIZE);
                req_size        <= conv_std_logic_vector(((byte_size + 3) / 4),
6);
                read_size      <= conv_std_logic_vector(((byte_size + 3) / 4 +
1), 7);
            elsif (SDRAM_READ = '1') then
                dma_state      <= PORT_REQ_ST;
                dma_next_state <= SDRAM_READ_ST;
                sdram_addr_i   <= SDRAM_PTR;
                bram_addr_i    <= BRAM_PTR;
                byte_size      := conv_integer(SIZE);
                req_size        <= conv_std_logic_vector(((byte_size + 3) / 4),
6);
                read_size      <= conv_std_logic_vector(((byte_size + 3) / 4 +
1), 7);
            else
                dma_state      <= IDLE_ST;
                sdram_addr_i   <= (others => '0');
            end if;
        elsif (dma_state = PORT_REQ_ST) then
            if (PORT_GRANT = '1') then
                dma_state      <= dma_next_state;
            end if;
        elsif (dma_state = SDRAM_READ_ST) then
            if (mcb_state = READ_COMP_ST) then
                dma_state      <= BRAM_WRITE_ST;

```

```

        else
            dma_state    <= SDRAM_READ_ST;
        end if;
    elsif (dma_state = BRAM_WRITE_ST) then
        if (sdram_rd_cntr < read_size) then
            dma_state    <= BRAM_WRITE_ST;
        else
            if (bram_wr_req = '0') then
                dma_state    <= COMPLETE_ST;
            else
                dma_state    <= BRAM_WRITE_ST;
            end if;
        end if;
    elsif (dma_state = BRAM_READ_ST) then
        if (bram_rd_cntr < read_size) then
            dma_state    <= BRAM_READ_ST;
        else
            if (sdram_wr_req = '0') then
                dma_state    <= SDRAM_WRITE_ST;
            else
                dma_state    <= BRAM_READ_ST;
            end if;
        end if;
    elsif (dma_state = SDRAM_WRITE_ST) then
        if (mcb_state = WRITE_COMP_ST) then
            dma_state    <= COMPLETE_ST;
        else
            dma_state    <= SDRAM_WRITE_ST;
        end if;
    elsif (dma_state = COMPLETE_ST) then
        dma_state    <= IDLE_ST;
    end if;
end if;
end process DMA_STATE_PROC;

PORT_REQUEST    <=    '1'    when    dma_state = PORT_REQ_ST
                  else    '0';

PORT_RETURN     <=    '1'    when    dma_state = COMPLETE_ST
                  else    '0';

DMA_STATUS_PROC    :    process (CLK, RST)
begin
    if (RST = '1') then
        COMPLETE    <= '0';
        BUSY        <= '0';
    elsif (rising_edge(CLK)) then
        if (dma_state = IDLE_ST) then
            if (SDRAM_WRITE = '1' or SDRAM_READ = '1') then
                COMPLETE    <= '0';
                BUSY        <= '1';
            end if;
        elsif (dma_state = COMPLETE_ST) then
            COMPLETE    <= '1';
            BUSY        <= '0';
        end if;
    end if;
end process;

```

```

        end if;
    end if;
end process DMA_STATUS_PROC;

BRAM_READ_PROC : process (CLK, RST)
begin
    if (RST = '1') then
        bram_rd_req    <= '0';
        bram_rd_addr    <= (others => '0');
        bram_rd_cntr    <= 0;
        sdram_wr_req    <= '0';
        first_word      <= '0';
    elsif (rising_edge(CLK)) then
        if (dma_state = BRAM_READ_ST) then
            if (bram_rd_cntr < read_size) then
                bram_rd_req    <= '1';
            else
                bram_rd_req    <= '0';
            end if;
            if (bram_rd_cntr > 0) then
                bram_rd_addr    <= bram_rd_addr + 1;
            else
                bram_rd_addr    <= bram_addr_i;
            end if;
            sdram_wr_req    <= bram_rd_req;
            first_word      <= not sdram_wr_req;
            bram_rd_cntr    <= bram_rd_cntr + 1;
        else
            bram_rd_req    <= '0';
            bram_rd_addr    <= bram_addr_i;
            bram_rd_cntr    <= 0;
            sdram_wr_req    <= '0';
            first_word      <= '0';
        end if;
    end if;
end process BRAM_READ_PROC;

first_mask    <=    "1000"    when sdram_addr_i(1 downto 0) = "01"
                  else    "1100"    when sdram_addr_i(1 downto 0) = "10"
                  else    "1110"    when sdram_addr_i(1 downto 0) = "11"
                  else    (others => '0');

BRAM_RE        <=    bram_rd_req;
WR_EN          <=    sdram_wr_req;
WR_MASK        <=    first_mask when first_word = '1'
                  else    (others => '0');
WR_DATA        <=    BRAM_DOUT;

SDRAM_READ_PROC : process (CLK, RST)
begin
    if (RST = '1') then
        sdram_rd_req    <= '0';
        sdram_rd_cntr    <= 0;
        bram_wr_addr    <= (others => '0');
        bram_wr_req      <= '0';
    end if;
end process SDRAM_READ_PROC;

```

```

    elsif (rising_edge(CLK)) then
        if (dma_state = BRAM_WRITE_ST) then
            if (sdram_rd_cntr < read_size) then
                sdram_rd_req    <= '1';
                bram_wr_req     <= '1';
            else
                sdram_rd_req    <= '0';
                bram_wr_req     <= '0';
            end if;
            if (bram_wr_req = '1') then
                bram_wr_addr    <= bram_wr_addr + 1;
            end if;
            sdram_rd_cntr    <= sdram_rd_cntr + 1;
        else
            sdram_rd_req    <= '0';
            sdram_rd_cntr    <= 0;
            bram_wr_addr    <= bram_addr_i;
            bram_wr_req     <= '0';
        end if;
    end if;
end process SDRAM_READ_PROC;

RD_EN        <= sdram_rd_req;
BRAM_WE      <= bram_wr_req;
BRAM_DIN     <= RD_DATA;

BRAM_ADDR    <=  bram_wr_addr  when dma_state = BRAM_WRITE_ST
                else bram_rd_addr when dma_state = BRAM_READ_ST
                else (others => '0');

MCB_STATEMACHINE_PROC : process (CLK, RST)
begin
    if (RST = '1') then
        mcb_state    <= IDLE_ST;
    elsif (rising_edge(CLK)) then
        if (mcb_state = IDLE_ST) then
            if (dma_state = SDRAM_READ_ST) then
                mcb_state    <= READ_REQ_ST;
            elsif (dma_state = SDRAM_WRITE_ST) then
                mcb_state    <= WRITE_REQ_ST;
            else
                mcb_state    <= IDLE_ST;
            end if;
        elsif (mcb_state = READ_REQ_ST) then
            if (CMD_EMPTY = '1') then
                mcb_state    <= READ_WAIT_ST;
            else
                mcb_state    <= READ_REQ_ST;
            end if;
        elsif (mcb_state = READ_WAIT_ST) then
            if (RD_COUNT = read_size) then
                mcb_state    <= READ_COMP_ST;
            else
                mcb_state    <= READ_WAIT_ST;
            end if;
        end if;
    end if;
end process;

```

```

        elsif (mcb_state = READ_COMP_ST) then
            mcb_state <= IDLE_ST;
        elsif (mcb_state = WRITE_REQ_ST) then
            if (CMD_EMPTY = '1') then
                mcb_state <= WRITE_WAIT_ST;
            else
                mcb_state <= WRITE_REQ_ST;
            end if;
        elsif (mcb_state = WRITE_WAIT_ST) then
            if (WR_EMPTY = '1' and CMD_EMPTY = '1') then
                mcb_state <= WRITE_COMP_ST;
            else
                mcb_state <= WRITE_WAIT_ST;
            end if;
        elsif (mcb_state = WRITE_COMP_ST) then
            mcb_state <= IDLE_ST;
        end if;
    end if;
end process MCB_STATEMACHINE_PROC;

    CMD_EN <= '1' when (mcb_state =
    READ_REQ_ST or mcb_state = WRITE_REQ_ST)
    and CMD_EMPTY = '1'
    else '0';

    CMD_INSTR <= C_READ_CMD when mcb_state =
    READ_REQ_ST and CMD_EMPTY = '1'
    else C_WRITE_CMD when mcb_state =
    WRITE_REQ_ST and CMD_EMPTY = '1'
    else (others => '0');

    CMD_BL <= req_size when (mcb_state =
    READ_REQ_ST or mcb_state = WRITE_REQ_ST)
    and CMD_EMPTY = '1'
    else (others => '0');

    CMD_BYTE_ADDR <= sdram_addr_i(29 downto 0) when (mcb_state =
    READ_REQ_ST or mcb_state = WRITE_REQ_ST)
    and CMD_EMPTY = '1'
    else (others => '0');

end architecture Behavioral;

```

Файл lp_dma_controller.vhd

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_arith.conv_integer;
use ieee.std_logic_arith.conv_std_logic_vector;
use ieee.numeric_std.all;
use ieee.std_logic_misc.all;

library unisim;

```

```

use unisim.vcomponents.all;

entity lp_dma_controller is
    generic
    (
        C_SDRAM_BASE_ADDRESS :      std_logic_vector := x"8800_0000";
        C_LDMA_BASE_ADDRESS  :      std_logic_vector := x"8A00_0000";
        C_DATA_PORT_SIZE     :      integer           := 32;
        C_MASK_SIZE          :      integer           := 4
    );
    port
    (
        CLK                : in    std_logic;
        RST                : in    std_logic;

        REG_ID             : in    std_logic_vector(3 downto 0);
        REG_DIN            : in    std_logic_vector(15 downto 0);
        REG_WE             : in    std_logic;

        STATUS             : out   std_logic_vector(1 downto 0);

        -- BRAM INTERFACE
        BRAM_ADDR          : out   std_logic_vector(8 downto 0);
        BRAM_WE            : out   std_logic;
        BRAM_DIN           : out   std_logic_vector(31 downto 0);
        BRAM_RE            : out   std_logic;
        BRAM_DOUT          : in    std_logic_vector(31 downto 0);
        -- MCB ARBITER INTERFACE
        PORT_REQUEST       : out   std_logic;
        PORT_GRANT         : in    std_logic;
        PORT_RETURN        : out   std_logic;
        -- MCB INTERFACE
        CMD_CLK            : out   std_logic;
        CMD_EN             : out   std_logic;
        CMD_INSTR          : out   std_logic_vector(2 downto 0);
        CMD_BL             : out   std_logic_vector(5 downto 0);
        CMD_BYTE_ADDR      : out   std_logic_vector(29 downto 0);
        CMD_EMPTY          : in    std_logic;
        CMD_FULL           : in    std_logic;
        WR_CLK             : out   std_logic;
        WR_EN              : out   std_logic;
        WR_MASK            : out   std_logic_vector(C_MASK_SIZE - 1 downto
0);
        WR_DATA            : out   std_logic_vector(C_DATA_PORT_SIZE - 1
downto 0);
        WR_FULL           : in    std_logic;
        WR_EMPTY          : in    std_logic;
        WR_COUNT           : in    std_logic_vector(6 downto 0);
        WR_UNDERRUN        : in    std_logic;
        WR_ERROR           : in    std_logic;
        RD_CLK            : out   std_logic;
        RD_EN              : out   std_logic;
        RD_DATA            : in    std_logic_vector(C_DATA_PORT_SIZE - 1
downto 0);
        RD_FULL           : in    std_logic;

```

```

        RD_EMPTY          : in      std_logic;
        RD_COUNT          : in      std_logic_vector(6 downto 0);
        RD_OVERFLOW       : in      std_logic;
        RD_ERROR          : in      std_logic
    );
end entity lp_dma_controller;

architecture Behavioral of lp_dma_controller is

    component lp_dma_core is
        generic
        (
            C_DATA_PORT_SIZE      : integer      := 32;
            C_MASK_SIZE           : integer      := 4
        );
        port
        (
            CLK                   : in      std_logic;
            RST                   : in      std_logic;
            SDRAM_PTR             : in      std_logic_vector(31 downto 0);
            SDRAM_READ            : in      std_logic;
            SDRAM_WRITE           : in      std_logic;
            BRAM_PTR              : in      std_logic_vector(8 downto 0);
            SIZE                  : in      std_logic_vector(7 downto 0);
            BUSY                  : out     std_logic;
            COMPLETE              : out     std_logic;
            -- BRAM INTERFACE
            BRAM_ADDR             : out     std_logic_vector(8 downto 0);
            BRAM_WE               : out     std_logic;
            BRAM_DIN              : out     std_logic_vector(31 downto 0);
            BRAM_RE               : out     std_logic;
            BRAM_DOUT             : in      std_logic_vector(31 downto 0);
            -- MCB ARBITER INTERFACE
            PORT_REQUEST          : out     std_logic;
            PORT_GRANT            : in      std_logic;
            PORT_RETURN           : out     std_logic;
            -- MCB INTERFACE
            CMD_CLK               : out     std_logic;
            CMD_EN                : out     std_logic;
            CMD_INSTR             : out     std_logic_vector(2 downto 0);
            CMD_BL                : out     std_logic_vector(5 downto 0);
            CMD_BYTE_ADDR        : out     std_logic_vector(29 downto 0);
            CMD_EMPTY             : in      std_logic;
            CMD_FULL              : in      std_logic;
            WR_CLK                : out     std_logic;
            WR_EN                 : out     std_logic;
            WR_MASK               : out     std_logic_vector(C_MASK_SIZE - 1
downto 0);
            WR_DATA               : out     std_logic_vector(C_DATA_PORT_SIZE - 1
downto 0);
            WR_FULL               : in      std_logic;
            WR_EMPTY             : in      std_logic;
            WR_COUNT              : in      std_logic_vector(6 downto 0);
            WR_UNDERRUN           : in      std_logic;
            WR_ERROR              : in      std_logic;

```

```

        RD_CLK                : out    std_logic;
        RD_EN                  : out    std_logic;
        RD_DATA                : in     std_logic_vector(C_DATA_PORT_SIZE - 1
downto 0);
        RD_FULL                : in     std_logic;
        RD_EMPTY               : in     std_logic;
        RD_COUNT               : in     std_logic_vector(6 downto 0);
        RD_OVERFLOW            : in     std_logic;
        RD_ERROR               : in     std_logic
    );
end component lp_dma_core;

    signal    sdram_addr_reg : std_logic_vector(31 downto 0) := (others =>
'0');
    signal    sdram_addr     : std_logic_vector(31 downto 0) := (others =>
'0');
    signal    bram_addr_reg  : std_logic_vector(8 downto 0)  := (others =>
'0');
    signal    size_reg       : std_logic_vector(7 downto 0)  := (others =>
'0');
    signal    read_req       : std_logic                      := '0';
    signal    write_req      : std_logic                      := '0';
    signal    dma_busy       : std_logic;
    signal    dma_complete   : std_logic;
    signal    status_reg     : std_logic_vector(1 downto 0);

begin

SDRAM_ADDR_PROC    : process (CLK, RST)
begin
    if (RST = '1') then
        sdram_addr_reg <= (others => '0');
    elsif (rising_edge(CLK)) then
        if (REG_WE = '1') then
            if (REG_ID = "0001") then
                sdram_addr_reg(31 downto 16) <= REG_DIN;
            elsif (REG_ID = "0010") then
                sdram_addr_reg(15 downto 0) <= REG_DIN;
            end if;
        end if;
    end if;
end process SDRAM_ADDR_PROC;

BRAM_ADDR_PROC : process (CLK, RST)
begin
    if (RST = '1') then
        bram_addr_reg <= (others => '0');
    elsif (rising_edge(CLK)) then
        if (REG_WE = '1') then
            if (REG_ID = "0011") then
                bram_addr_reg <= REG_DIN(9 downto 1);
            end if;
        end if;
    end if;
end process BRAM_ADDR_PROC;

```

```

end process BRAM_ADDR_PROC;

SIZE_PROC : process (CLK, RST)
begin
    if (RST = '1') then
        size_reg <= (others => '0');
    elsif (rising_edge(CLK)) then
        if (REG_WE = '1') then
            if (REG_ID = "0100") then
                size_reg <= REG_DIN(7 downto 0);
            end if;
        end if;
    end if;
end process SIZE_PROC;

CONTROL_PROC : process (CLK, RST)
begin
    if (RST = '1') then
        read_req    <= '0';
        write_req   <= '0';
        status_reg  <= "00";
    elsif (rising_edge(CLK)) then
        if (REG_WE = '1') then
            if (REG_ID = "0000") then
                if (REG_DIN(2) = '1') then
                    read_req    <= '1';
                    write_req   <= '0';
                    status_reg  <= "01";
                elsif (REG_DIN(3) = '1') then
                    read_req    <= '0';
                    write_req   <= '1';
                    status_reg  <= "01";
                end if;
            end if;
        elsif (dma_busy = '1') then
            read_req    <= '0';
            write_req   <= '0';
            status_reg  <= "01";
        elsif (dma_complete = '1') then
            read_req    <= '0';
            write_req   <= '0';
            status_reg  <= "10";
        end if;
    end if;
end process CONTROL_PROC;

sdram_addr <= sdram_addr_reg + C_LDMA_BASE_ADDRESS - C_SDRAM_BASE_ADDRESS;

STATUS    <= status_reg;

CORE_I : lp_dma_core
generic map
(
    C_DATA_PORT_SIZE    => C_DATA_PORT_SIZE,
    C_MASK_SIZE         => C_MASK_SIZE

```

```

)
port map
(
    CLK                => CLK,
    RST                => RST,
    SDRAM_PTR          => sdram_addr,
    SDRAM_WRITE        => write_req,
    SDRAM_READ         => read_req,
    BRAM_PTR           => bram_addr_reg,
    SIZE               => size_reg,
    COMPLETE           => dma_complete,
    BUSY               => dma_busy,
    -- BRAM INTERFACE
    BRAM_ADDR          => BRAM_ADDR,
    BRAM_WE            => BRAM_WE,
    BRAM_DIN           => BRAM_DIN,
    BRAM_RE            => BRAM_RE,
    BRAM_DOUT          => BRAM_DOUT,
    -- MCB ARBITER INTERFACE
    PORT_REQUEST       => PORT_REQUEST,
    PORT_GRANT         => PORT_GRANT,
    PORT_RETURN        => PORT_RETURN,
    -- MCB INTERFACE
    CMD_CLK            => CMD_CLK,
    CMD_EN             => CMD_EN,
    CMD_INSTR          => CMD_INSTR,
    CMD_BL             => CMD_BL,
    CMD_BYTE_ADDR      => CMD_BYTE_ADDR,
    CMD_EMPTY          => CMD_EMPTY,
    CMD_FULL           => CMD_FULL,
    WR_CLK             => WR_CLK,
    WR_EN              => WR_EN,
    WR_MASK            => WR_MASK,
    WR_DATA            => WR_DATA,
    WR_FULL            => WR_FULL,
    WR_EMPTY           => WR_EMPTY,
    WR_COUNT           => WR_COUNT,
    WR_UNDERERRUN      => WR_UNDERERRUN,
    WR_ERROR           => WR_ERROR,
    RD_CLK             => RD_CLK,
    RD_EN              => RD_EN,
    RD_DATA            => RD_DATA,
    RD_FULL            => RD_FULL,
    RD_EMPTY           => RD_EMPTY,
    RD_COUNT           => RD_COUNT,
    RD_OVERFLOW        => RD_OVERFLOW,
    RD_ERROR           => RD_ERROR
);
end architecture Behavioral;

```

Файл lp_buffers_controller.vhd

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

```

```

use ieee.std_logic_arith.all;
use ieee.std_logic_arith.conv_integer;
use ieee.std_logic_arith.conv_std_logic_vector;
use ieee.numeric_std.all;
use ieee.std_logic_misc.all;

library unisim;
use unisim.vcomponents.all;

entity lp_buffers_controller is
  generic
  (
    C_BACK_CH_ENABLE      :      integer range 0 to 1 := 1
  );
  port
  (
    CLK          :  in   std_logic;
    RST          :  in   std_logic;

    PROC_PTR     :  in   std_logic_vector(15 downto 0);
    PROC_EN      :  in   std_logic;
    PROC_COMP    :  out  std_logic;

    CPU_PTR      :  out  std_logic_vector(15 downto 0);
    CPU_WE       :  out  std_logic;
    CPU_ACK      :  in   std_logic;

    TX_PTR       :  out  std_logic_vector(15 downto 0);
    TX_WE        :  out  std_logic;
    TX_ACK       :  in   std_logic;

    BACK_PTR     :  out  std_logic_vector(15 downto 0);
    BACK_WE      :  out  std_logic;
    BACK_ACK     :  in   std_logic;

    CLEAN_PTR    :  out  std_logic_vector (15 downto 0);
    CLEAN_WE     :  out  std_logic;
    CLEAN_ACK    :  in   std_logic;

    BUF_PTR      :  out  std_logic_vector(15 downto 0);
    BUF_TO_CPU   :  in   std_logic;
    BUF_TO_BACK  :  in   std_logic;
    BUF_TO_TX    :  in   std_logic;
    BUF_TO_CLR   :  in   std_logic;
    BUF_STATUS   :  out  std_logic_vector(1 downto 0)
  );
end entity lp_buffers_controller;

architecture Behavioral of lp_buffers_controller is

  signal  ptr_reg      :  std_logic_vector(15 downto 0);
  signal  status       :  std_logic_vector(1 downto 0);
  signal  ptr_to_tx    :  std_logic;
  signal  ptr_to_back  :  std_logic;
  signal  ptr_to_cpu   :  std_logic;

```

```

    signal    ptr_to_clr    :    std_logic;

    signal    ack_op        :    std_logic;
    signal    do_op         :    std_logic;

begin

    ack_op    <=    '1'    when    (ptr_to_cpu = '1' and CPU_ACK = '1') or
                                (ptr_to_tx = '1' and TX_ACK = '1') or
                                (ptr_to_back = '1' and BACK_ACK = '1' and
C_BACK_CH_ENABLE = 1) or
                                (ptr_to_clr = '1' and CLEAN_ACK = '1')
                                else    '0';

    do_op    <=    '1'    when    (BUF_TO_CPU = '1' or BUF_TO_TX = '1' or BUF_TO_CLR
= '1'
                                or (BUF_TO_BACK = '1' and C_BACK_CH_ENABLE = 1))
                                else    '0';

STATUS_PROC :    process (CLK, RST)
begin
    if (RST = '1') then
        ptr_reg    <= (others => '0');
        status    <= (others => '0');
    elsif (rising_edge(CLK) ) then
        if (PROC_EN = '1' and status = "00") then
            ptr_reg    <= PROC_PTR;
            status    <= "01";
        elsif (ack_op = '1' and status = "10") then
            ptr_reg    <= (others => '0');
            status    <= "00";
        elsif (status = "01" and do_op = '1') then
            ptr_reg    <= ptr_reg;
            status    <= "10";
        end if;
    end if;
end process STATUS_PROC;

BUF_STATUS    <= status;
BUF_PTR    <= ptr_reg;

CONTROL_PROC    :    process (CLK, RST)
begin
    if (RST = '1') then
        ptr_to_tx    <= '0';
        ptr_to_back    <= '0';
        ptr_to_cpu    <= '0';
        ptr_to_clr    <= '0';
    elsif (rising_edge(CLK)) then
        if (ack_op = '1') then
            ptr_to_tx    <= '0';
            ptr_to_back    <= '0';
            ptr_to_cpu    <= '0';
            ptr_to_clr    <= '0';
        elsif (BUF_TO_CPU = '1' and status = "01") then

```

```

        ptr_to_tx    <= '0';
        ptr_to_back <= '0';
        ptr_to_cpu   <= '1';
        ptr_to_clr   <= '0';
    elsif (BUF_TO_TX = '1' and status = "01") then
        ptr_to_tx    <= '1';
        ptr_to_back <= '0';
        ptr_to_cpu   <= '0';
        ptr_to_clr   <= '0';
    elsif (BUF_TO_BACK = '1' and C_BACK_CH_ENABLE = 1 and status = "01")
then
        ptr_to_tx    <= '0';
        ptr_to_back <= '1';
        ptr_to_cpu   <= '0';
        ptr_to_clr   <= '0';
    elsif (BUF_TO_CLR = '1' and status = "01") then
        ptr_to_back <= '0';
        ptr_to_tx    <= '0';
        ptr_to_cpu   <= '0';
        ptr_to_clr   <= '1';
    elsif (status = "10") then
        ptr_to_back <= ptr_to_back;
        ptr_to_tx    <= ptr_to_tx;
        ptr_to_cpu   <= ptr_to_cpu;
        ptr_to_clr   <= ptr_to_clr;
    else
        ptr_to_back <= '0';
        ptr_to_tx    <= '0';
        ptr_to_cpu   <= '0';
        ptr_to_clr   <= '0';
    end if;
end if;
end process CONTROL_PROC;

CPU_PTR    <= ptr_reg;
CPU_WE     <= ptr_to_cpu and (not CPU_ACK);

TX_PTR     <= ptr_reg;
TX_WE      <= ptr_to_tx and (not TX_ACK);

BACK_PTR   <= ptr_reg;
BACK_WE    <= ptr_to_back and (not BACK_ACK);

CLEAN_PTR  <= ptr_reg;
CLEAN_WE   <= ptr_to_clr and (not CLEAN_ACK);

PROC_COMP  <= ack_op;

end architecture Behavioral;

```

Файл multi_mem_processor.vhd

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

```

```

use ieee.std_logic_arith.all;
use ieee.std_logic_arith.conv_integer;
use ieee.std_logic_arith.conv_std_logic_vector;
use ieee.numeric_std.all;
use ieee.std_logic_misc.all;

library unisim;
use unisim.vcomponents.all;

library picoblaze_v1_00_a;
use picoblaze_v1_00_a.all;

library lp_progs_v1_00_a;
use lp_progs_v1_00_a.all;

entity multi_mem_processor is
    generic
    (
        C_TYPE          : integer range 0 to 1 := 0
    );
    port
    (
        CLK              : in    std_logic;
        RST              : in    std_logic;
        MODE             : in    std_logic_vector(1 downto 0);
        PORT_ID          : out   std_logic_vector(7 downto 0);
        WRITE_STROBE     : out   std_logic;
        READ_STROBE      : out   std_logic;
        OUT_PORT         : out   std_logic_vector(15 downto 0);
        IN_PORT          : in    std_logic_vector(15 downto 0)
    );
end entity multi_mem_processor;

architecture Behavioral of multi_mem_processor is

    constant DATA_WIDTH : integer := 16;
    constant LOAD_DELAY  : integer := 0;

    signal  reset        : std_logic;
    signal  mode_config  : std_logic_vector(1 downto 0);
    signal  mode_change  : std_logic;

    signal  address      : std_logic_vector(7 downto 0);
    signal  instruction  : std_logic_vector(15 downto 0);
    signal  clk_enable   : std_logic;

begin

    Mode_config_proc : process (CLK, RST)
    begin
        if (RST = '1') then
            mode_config <= (others => '0');
            mode_change <= '0';
        elsif (rising_edge(CLK)) then
            if (mode_config /= MODE) then

```

```

        mode_change <= '1';
    else
        mode_change <= '0';
    end if;
    mode_config <= MODE;
end if;
end process Mode_config_proc;

reset <= RST or mode_change;

PROCESSOR_INST : entity picoblaze_v1_00_a.picoblaze
    generic map
    (
        DATA_WIDTH      => DATA_WIDTH,
        LOAD_DELAY        => LOAD_DELAY
    )
    port map
    (
        clk               => CLK,
        reset             => reset,
        address           => address,
        instruction        => instruction,
        clk_enable         => clk_enable,
        port_id           => PORT_ID,
        write_strobe       => WRITE_STROBE,
        out_port           => OUT_PORT,
        read_strobe        => READ_STROBE,
        in_port            => IN_PORT,
        interrupt          => '0'
    );

ETH_PROG_GEN : if (C_TYPE = 0) generate
    program_inst : entity lp_progs_v1_00_a.ETH_PROG
        port map
        (
            clk           => clk,
            clk_enable    => clk_enable,
            page          => MODE,
            address        => address,
            instruction    => instruction
        );
end generate ETH_PROG_GEN;

AUR_PROG_GEN : if (C_TYPE = 1) generate
    program_inst : entity lp_progs_v1_00_a.AUR_PROG
        port map
        (
            clk           => clk,
            clk_enable    => clk_enable,
            page          => MODE,
            address        => address,
            instruction    => instruction
        );
end generate AUR_PROG_GEN;

```

```
end architecture Behavioral;
```

Файл link_processor.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_arith.conv_integer;
use ieee.std_logic_arith.conv_std_logic_vector;
use ieee.numeric_std.all;
use ieee.std_logic_misc.all;

library unisim;
use unisim.vcomponents.all;

entity link_processor is
  generic
  (
    C_TYPE                : integer           := 1;
    C_BACK_CH_ENABLE      : integer range 0 to 1 := 1;
    C_SDRAM_BASE_ADDRESS  : std_logic_vector   := x"8800_0000";
    C_LDMA_BASE_ADDRESS   : std_logic_vector   := x"8A00_0000";
    C_DATA_PORT_SIZE      : integer           := 32;
    C_MASK_SIZE           : integer           := 4
  );
  port
  (
    CLK                : in    std_logic;
    CLKx2              : in    std_logic;
    RST                : in    std_logic;
    DMA_RELOAD         : in    std_logic;

    MODE               : in    std_logic_vector(1 downto 0);

    PROC_PTR           : in    std_logic_vector(15 downto 0);
    PROC_EN            : in    std_logic;
    PROC_COMP          : out   std_logic;

    CPU_PTR            : out   std_logic_vector(15 downto 0);
    CPU_WE             : out   std_logic;
    CPU_ACK            : in    std_logic;

    TX_PTR             : out   std_logic_vector(15 downto 0);
    TX_WE              : out   std_logic;
    TX_ACK             : in    std_logic;

    BACK_PTR           : out   std_logic_vector(15 downto 0);
    BACK_WE            : out   std_logic;
    BACK_ACK           : in    std_logic;

    CLEAN_PTR          : out   std_logic_vector (15 downto 0);
    CLEAN_WE           : out   std_logic;
    CLEAN_ACK          : in    std_logic;
```

```

PORT_REQUEST      : out    std_logic;
PORT_GRANT        : in     std_logic;
PORT_RETURN       : out    std_logic;

CMD_CLK           : out    std_logic;
CMD_EN            : out    std_logic;
CMD_INSTR         : out    std_logic_vector(2 downto 0);
CMD_BL            : out    std_logic_vector(5 downto 0);
CMD_BYTE_ADDR     : out    std_logic_vector(29 downto 0);
CMD_EMPTY         : in     std_logic;
CMD_FULLL        : in     std_logic;
WR_CLK            : out    std_logic;
WR_EN             : out    std_logic;
WR_MASK           : out    std_logic_vector(C_MASK_SIZE - 1 downto
0);
WR_DATA           : out    std_logic_vector(C_DATA_PORT_SIZE - 1
downto 0);
WR_FULLL         : in     std_logic;
WR_EMPTY         : in     std_logic;
WR_COUNT         : in     std_logic_vector(6 downto 0);
WR_UNDEERRUN     : in     std_logic;
WR_ERROR         : in     std_logic;
RD_CLK           : out    std_logic;
RD_EN            : out    std_logic;
RD_DATA          : in     std_logic_vector(C_DATA_PORT_SIZE - 1
downto 0);
RD_FULLL         : in     std_logic;
RD_EMPTY         : in     std_logic;
RD_COUNT         : in     std_logic_vector(6 downto 0);
RD_OVERFLOW      : in     std_logic;
RD_ERROR         : in     std_logic
);
end entity link_processor;

```

architecture Behavioral of link_processor is

```

constant C_REGS_BASE_ADDR : std_logic_vector := x"F0";
constant C_DMA_CNTRL      : std_logic_vector := x"F0";
constant C_DMA_DDR_HADDR  : std_logic_vector := x"F1";
constant C_DMA_DDR_LADDR  : std_logic_vector := x"F2";
constant C_DMA_BRAM_ADDR  : std_logic_vector := x"F3";
constant C_DMA_SIZE       : std_logic_vector := x"F4";
constant C_BUF_CNTRL      : std_logic_vector := x"F5";
constant C_BUF_HADDR      : std_logic_vector := x"F6";
constant C_BUF_LADDR      : std_logic_vector := x"F7";
constant C_RND_DIN        : std_logic_vector := x"FB";
constant C_RND_DOUT       : std_logic_vector := x"FC";
constant C_CSUM_NEXT      : std_logic_vector := x"FD";
constant C_CSUM_DIN       : std_logic_vector := x"FE";
constant C_CSUM_DOUT      : std_logic_vector := x"FF";

```

```

component multi_mem_processor is
  generic
  (

```

```

        C_TYPE          :          integer range 0 to 1 := 0
    );
port
(
    CLK          :  in    std_logic;
    RST          :  in    std_logic;
    MODE         :  in    std_logic_vector(1 downto 0);
    PORT_ID      :  out   std_logic_vector(7 downto 0);
    WRITE_STROBE :  out   std_logic;
    READ_STROBE  :  out   std_logic;
    OUT_PORT     :  out   std_logic_vector(15 downto 0);
    IN_PORT      :  in    std_logic_vector(15 downto 0)
);
end component multi_mem_processor;

component rnd is
generic
(
    C_INIT       :    std_logic_vector  := x"1234_5678";
    C_SR_SIZE    :    integer           := 32;
    C_DWIDTH     :    integer           := 16;
    C_RND_SIZE   :    integer           := 128
);
port
(
    CLK          :  in    std_logic;
    RST          :  in    std_logic;
    DIN          :  in    std_logic_vector(C_DWIDTH - 1 downto 0);
    WE           :  in    std_logic;
    DOUT         :  out   std_logic_vector(C_DWIDTH - 1 downto 0);
    RE           :  in    std_logic
);
end component rnd;

component ipcsum_calc is
port
(
    CLK          :  in    std_logic;
    RST          :  in    std_logic;
    NEXT_CSUM    :  in    std_logic;
    DIN          :  in    std_logic_vector(15 downto 0);
    WE           :  in    std_logic;
    DOUT         :  out   std_logic_vector(15 downto 0)
);
end component ipcsum_calc;

component lp_buffers_controller is
port
(
    CLK          :  in    std_logic;
    RST          :  in    std_logic;

    PROC_PTR     :  in    std_logic_vector(15 downto 0);
    PROC_EN      :  in    std_logic;
    PROC_COMP    :  out   std_logic;

```

```

CPU_PTR      : out    std_logic_vector(15 downto 0);
CPU_WE       : out    std_logic;
CPU_ACK      : in     std_logic;

TX_PTR       : out    std_logic_vector(15 downto 0);
TX_WE        : out    std_logic;
TX_ACK       : in     std_logic;

BACK_PTR     : out    std_logic_vector(15 downto 0);
BACK_WE      : out    std_logic;
BACK_ACK     : in     std_logic;

CLEAN_PTR    : out    std_logic_vector (15 downto 0);
CLEAN_WE     : out    std_logic;
CLEAN_ACK    : in     std_logic;

BUF_PTR      : out    std_logic_vector(15 downto 0);
BUF_TO_CPU   : in     std_logic;
BUF_TO_BACK  : in     std_logic;
BUF_TO_TX    : in     std_logic;
BUF_TO_CLR   : in     std_logic;
BUF_STATUS   : out    std_logic_vector(1 downto 0)
);
end component lp_buffers_controller;

component lp_dma_controller is
generic
(
    C_SDRAM_BASE_ADDRESS : std_logic_vector := x"8800_0000";
    C_LDMA_BASE_ADDRESS  : std_logic_vector := x"8A00_0000";
    C_DATA_PORT_SIZE     : integer          := 32;
    C_MASK_SIZE          : integer          := 4
);
port
(
    CLK           : in     std_logic;
    RST           : in     std_logic;
    REG_ID        : in     std_logic_vector(3 downto 0);
    REG_WE        : in     std_logic;
    REG_DIN       : in     std_logic_vector(15 downto 0);
    STATUS        : out    std_logic_vector(1 downto 0);
    BRAM_ADDR     : out    std_logic_vector(8 downto 0);
    BRAM_WE       : out    std_logic;
    BRAM_DIN      : out    std_logic_vector(31 downto 0);
    BRAM_RE       : out    std_logic;
    BRAM_DOUT     : in     std_logic_vector(31 downto 0);
    PORT_REQUEST  : out    std_logic;
    PORT_GRANT    : in     std_logic;
    PORT_RETURN   : out    std_logic;
    CMD_CLK       : out    std_logic;
    CMD_EN        : out    std_logic;
    CMD_INSTR     : out    std_logic_vector(2 downto 0);
    CMD_BL        : out    std_logic_vector(5 downto 0);
    CMD_BYTE_ADDR : out    std_logic_vector(29 downto 0);

```

```

        CMD_EMPTY      : in    std_logic;
        CMD_FULL       : in    std_logic;
        WR_CLK         : out   std_logic;
        WR_EN          : out   std_logic;
        WR_MASK        : out   std_logic_vector(C_MASK_SIZE - 1
downto 0);
        WR_DATA        : out   std_logic_vector(C_DATA_PORT_SIZE - 1
downto 0);
        WR_FULL       : in    std_logic;
        WR_EMPTY      : in    std_logic;
        WR_COUNT       : in    std_logic_vector(6 downto 0);
        WR_UNDEERRUN   : in    std_logic;
        WR_ERROR       : in    std_logic;
        RD_CLK         : out   std_logic;
        RD_EN          : out   std_logic;
        RD_DATA        : in    std_logic_vector(C_DATA_PORT_SIZE - 1
downto 0);
        RD_FULL       : in    std_logic;
        RD_EMPTY      : in    std_logic;
        RD_COUNT       : in    std_logic_vector(6 downto 0);
        RD_OVERFLOW    : in    std_logic;
        RD_ERROR       : in    std_logic
    );
end component lp_dma_controller;

component RAMB16_S18_S36
port
(
    DOA : out std_logic_vector(15 downto 0);
    DOB : out std_logic_vector(31 downto 0);
    DOPA : out std_logic_vector(1 downto 0);
    DOPB : out std_logic_vector(3 downto 0);

    ADDRA : in std_logic_vector(9 downto 0);
    ADDRb : in std_logic_vector(8 downto 0);
    CLKA : in std_ulogic;
    CLKB : in std_ulogic;
    DIA : in std_logic_vector(15 downto 0);
    DIB : in std_logic_vector(31 downto 0);
    DIPA : in std_logic_vector(1 downto 0);
    DIPB : in std_logic_vector(3 downto 0);
    ENA : in std_ulogic;
    ENB : in std_ulogic;
    SSRA : in std_ulogic;
    SSRB : in std_ulogic;
    WEA : in std_ulogic;
    WEB : in std_ulogic
);
end component;

signal pico_rst      : std_logic;
signal port_id       : std_logic_vector(7 downto 0);
signal write_strobe   : std_logic;
signal read_strobe    : std_logic;
signal out_port       : std_logic_vector(15 downto 0);

```

```

signal    in_port          :    std_logic_vector(15 downto 0);

signal    bram_cs          :    std_logic;
signal    pico_mem_we      :    std_logic;
signal    pico_mem_en      :    std_logic;
signal    pico_mem_addr    :    std_logic_vector(9 downto 0);
signal    pico_mem_din     :    std_logic_vector(15 downto 0);
signal    pico_mem_dout    :    std_logic_vector(15 downto 0);

signal    dma_reg_id       :    std_logic_vector(3 downto 0);
signal    dma_reg_din      :    std_logic_vector(15 downto 0);
signal    dma_reg_we       :    std_logic;
signal    dma_status       :    std_logic_vector(1 downto 0);

signal    dma_mem_re       :    std_logic;
signal    dma_mem_we       :    std_logic;
signal    dma_mem_addr     :    std_logic_vector(8 downto 0);
signal    dma_mem_din      :    std_logic_vector(31 downto 0);
signal    dma_mem_dout     :    std_logic_vector(31 downto 0);

signal    csum_dout        :    std_logic_vector(15 downto 0);
signal    csum_din         :    std_logic_vector(15 downto 0);
signal    csum_we          :    std_logic;
signal    csum_next        :    std_logic;

signal    buf_addr         :    std_logic_vector(31 downto 0);
signal    buf_ptr          :    std_logic_vector(15 downto 0);
signal    buf_status       :    std_logic_vector(1 downto 0);
signal    buf_to_tx        :    std_logic;
signal    buf_to_back      :    std_logic;
signal    buf_to_cpu       :    std_logic;
signal    buf_to_clr       :    std_logic;

signal    mem_we           :    std_logic;
signal    mem_en           :    std_logic;
signal    mem_addr         :    std_logic_vector(8 downto 0);
signal    mem_din          :    std_logic_vector(31 downto 0);
signal    mem_dout         :    std_logic_vector(31 downto 0);

signal    rnd_dout         :    std_logic_vector(15 downto 0);
signal    rnd_din          :    std_logic_vector(15 downto 0);
signal    rnd_re           :    std_logic;
signal    rnd_we           :    std_logic;

begin

RST_PROC : process (CLK, RST, DMA_RELOAD)
begin
    if (RST = '1') then
        pico_rst <= '1';
    elsif (rising_edge(CLK)) then
        if (DMA_RELOAD = '1') then
            pico_rst <= '1';
        else
            pico_rst <= '0';
        end if;
    end if;
end process;

```

```

        end if;
    end if;
end process RST_PROC;

PROCESSOR_I : multi_mem_processor
generic map
(
    C_TYPE          => C_TYPE
)
port map
(
    CLK              => CLK,
    RST              => pico_rst,
    MODE            => MODE,
    PORT_ID          => port_id,
    WRITE_STROBE     => write_strobe,
    READ_STROBE      => read_strobe,
    OUT_PORT         => out_port,
    IN_PORT          => in_port

);

in_port      <=    pico_mem_dout                      when bram_cs = '1'
and read_strobe = '1'
              else x"000" & "00"
& dma_status      when port_id = C_DMA_CNTRL   and read_strobe =
'1'
              else x"000" & "00"
& buf_status      when port_id = C_BUF_CNTRL   and read_strobe =
'1'
              else buf_addr(31 downto 16)      when port_id = C_BUF_HADDR
and read_strobe = '1'
              else buf_addr(15 downto 0)      when port_id = C_BUF_LADDR
and read_strobe = '1'
              else csum_dout                  when port_id = C_CSUM_DOUT
and read_strobe = '1'
              else rnd_dout                  when port_id = C_RND_DOUT
and read_strobe = '1'
              else (others => '0');

bram_cs        <=    '1'          when port_id < C_REGS_BASE_ADDR
                  else '0';

pico_mem_addr  <=    "00" & port_id(7 downto 1) & (not port_id(0));
pico_mem_en    <=    bram_cs and (write_strobe or read_strobe);
pico_mem_we    <=    bram_cs and write_strobe;
pico_mem_din   <=    out_port;

rnd_re         <=    '1'          when port_id = C_RND_DOUT and read_strobe =
'1'
                  else '0';

rnd_we         <=    '1'          when port_id = C_RND_DIN  and write_strobe
= '1'
                  else '0';

```

```

rnd_din      <=    out_port    when  port_id = C_RND_DIN   and write_strobe
= '1'
                else  (others => '0');

RND_INST : rnd
generic map
(
    C_INIT      => x"1234_5678",
    C_SR_SIZE   => 32,
    C_DWIDTH    => 16,
    C_RND_SIZE  => 128
)
port map
(
    CLK         => CLK,
    RST         => pico_rst,
    DIN         => rnd_din,
    WE          => rnd_we,
    DOUT        => rnd_dout,
    RE          => rnd_re
);

csum_din     <=    out_port      when  port_id = C_CSUM_DIN   and
write_strobe = '1'
                else  (others => '0');

csum_we      <=    '1'          when  port_id = C_CSUM_DIN   and
write_strobe = '1'
                else  '0';

csum_next    <=    '1'          when  port_id = C_CSUM_NEXT and
write_strobe = '1'
                else  '0';

CSUM_CALC_INST : ipcsum_calc
port map
(
    CLK        => CLK,
    RST        => pico_rst,
    NEXT_CSUM  => csum_next,
    DIN        => csum_din,
    WE         => csum_we,
    DOUT       => csum_dout
);

buf_addr     <=    "00000" & BUF_PTR & "000000000000";

buf_to_clr   <=    out_port(1) when  port_id = C_BUF_CNTRL and write_strobe
= '1'
                else  '0';

buf_to_cpu   <=    out_port(2) when  port_id = C_BUF_CNTRL and write_strobe
= '1'
                else  '0';

```

```

buf_to_tx      <=    out_port(3) when  port_id = C_BUF_CNTRL and write_strobe
= '1'

                        else  '0';
buf_to_back    <=    out_port(4) when  port_id = C_BUF_CNTRL and write_strobe
= '1' and C_BACK_CH_ENABLE = 1
                        else  '0';

BUF_CONTR_I    :  lp_buffers_controller
    port map
    (
        CLK                      => CLK,
        RST                      => pico_rst,

        PROC_PTR                 => PROC_PTR,
        PROC_EN                  => PROC_EN,
        PROC_COMP                => PROC_COMP,

        CPU_PTR                  => CPU_PTR,
        CPU_WE                   => CPU_WE,
        CPU_ACK                  => CPU_ACK,

        TX_PTR                   => TX_PTR,
        TX_WE                    => TX_WE,
        TX_ACK                   => TX_ACK,

        BACK_PTR                 => BACK_PTR,
        BACK_WE                  => BACK_WE,
        BACK_ACK                 => BACK_ACK,

        CLEAN_PTR                => CLEAN_PTR,
        CLEAN_WE                 => CLEAN_WE,
        CLEAN_ACK                => CLEAN_ACK,

        BUF_PTR                  => buf_ptr,
        BUF_TO_CPU               => buf_to_cpu,
        BUF_TO_TX                => buf_to_tx,
        BUF_TO_BACK              => buf_to_back,
        BUF_TO_CLR               => buf_to_clr,
        BUF_STATUS               => buf_status
    );

    dma_reg_id  <=    x"0"      when  port_id = C_DMA_CNTRL      and
write_strobe = '1'
                        else  x"1"      when  port_id = C_DMA_DDR_HADDR  and
write_strobe = '1'
                        else  x"2"      when  port_id = C_DMA_DDR_LADDR  and
write_strobe = '1'
                        else  x"3"      when  port_id = C_DMA_BRAM_ADDR  and
write_strobe = '1'
                        else  x"4"      when  port_id = C_DMA_SIZE      and
write_strobe = '1'
                        else  (others => '0');
    dma_reg_we  <=    '1'      when  write_strobe = '1' and (port_id =
C_DMA_CNTRL or port_id = C_DMA_DDR_HADDR

```

```

C_DMA_DDR_LADDR or port_id = C_DMA_BRAM_ADDR

C_DMA_SIZE)

        else '0';
        dma_reg_din <= out_port;

DMA_I : lp_dma_controller
generic map
(
    C_SDRAM_BASE_ADDRESS => C_SDRAM_BASE_ADDRESS,
    C_LDMA_BASE_ADDRESS  => C_LDMA_BASE_ADDRESS,
    C_DATA_PORT_SIZE     => C_DATA_PORT_SIZE,
    C_MASK_SIZE          => C_MASK_SIZE
)
port map
(
    CLK                  => CLK,
    RST                  => RST,

    REG_ID               => dma_reg_id,
    REG_WE               => dma_reg_we,
    REG_DIN              => dma_reg_din,

    STATUS               => dma_status,

    -- BRAM INTERFACE
    BRAM_ADDR            => dma_mem_addr,
    BRAM_WE              => dma_mem_we,
    BRAM_DIN             => dma_mem_din,
    BRAM_RE              => dma_mem_re,
    BRAM_DOUT            => dma_mem_dout,
    -- MCB ARBITER INTERFACE
    PORT_REQUEST         => PORT_REQUEST,
    PORT_GRANT           => PORT_GRANT,
    PORT_RETURN          => PORT_RETURN,
    -- MCB INTERFACE
    CMD_CLK              => CMD_CLK,
    CMD_EN               => CMD_EN,
    CMD_INSTR            => CMD_INSTR,
    CMD_BL               => CMD_BL,
    CMD_BYTE_ADDR        => CMD_BYTE_ADDR,
    CMD_EMPTY            => CMD_EMPTY,
    CMD_FULL             => CMD_FULL,
    WR_CLK               => WR_CLK,
    WR_EN               => WR_EN,
    WR_MASK              => WR_MASK,
    WR_DATA              => WR_DATA,
    WR_FULL              => WR_FULL,
    WR_EMPTY             => WR_EMPTY,
    WR_COUNT             => WR_COUNT,
    WR_UNDERRUN          => WR_UNDERRUN,
    WR_ERROR             => WR_ERROR,
    RD_CLK              => RD_CLK,
    RD_EN               => RD_EN,

```

or port_id =

or port_id =

```

        RD_DATA          => RD_DATA,
        RD_FULL          => RD_FULL,
        RD_EMPTY         => RD_EMPTY,
        RD_COUNT         => RD_COUNT,
        RD_OVERFLOW      => RD_OVERFLOW,
        RD_ERROR         => RD_ERROR
    );

    mem_we      <=    dma_mem_we;
    mem_en      <=    dma_mem_re or dma_mem_we;
    mem_addr    <=    dma_mem_addr;
    mem_din     <=    dma_mem_din;
    dma_mem_dout <=    mem_dout;

INTERNAL_BRAM_I    : RAMB16_S18_S36
    port map
    (
        -- Pico
        CLKA  => CLKx2,
        ENA   => pico_mem_en,
        WEA   => pico_mem_we,
        ADDRA => pico_mem_addr,
        DIA   => pico_mem_din,
        DIP   => (others => '0'),
        DOA   => pico_mem_dout,
        DOPA  => open,
        SSRA  => '0',
        -- DMA
        CLKB  => CLK,
        ENB   => mem_en,
        WEB   => mem_we,
        ADDR  => mem_addr,
        DIB   => mem_din,
        DIPB  => (others => '0'),
        DOB   => mem_dout,
        DOPB  => open,
        SSRB  => '0'
    );

end architecture Behavioral;

```

ПРИЛОЖЕНИЕ Б

(обязательное)

АЛГОРИТМ ОБРАБОТКИ ВХОДЯЩЕГО ТРАФИКА КОММУТАТОРА СООБЩЕНИЙ АБОНЕНТСКОЙ СЕТИ

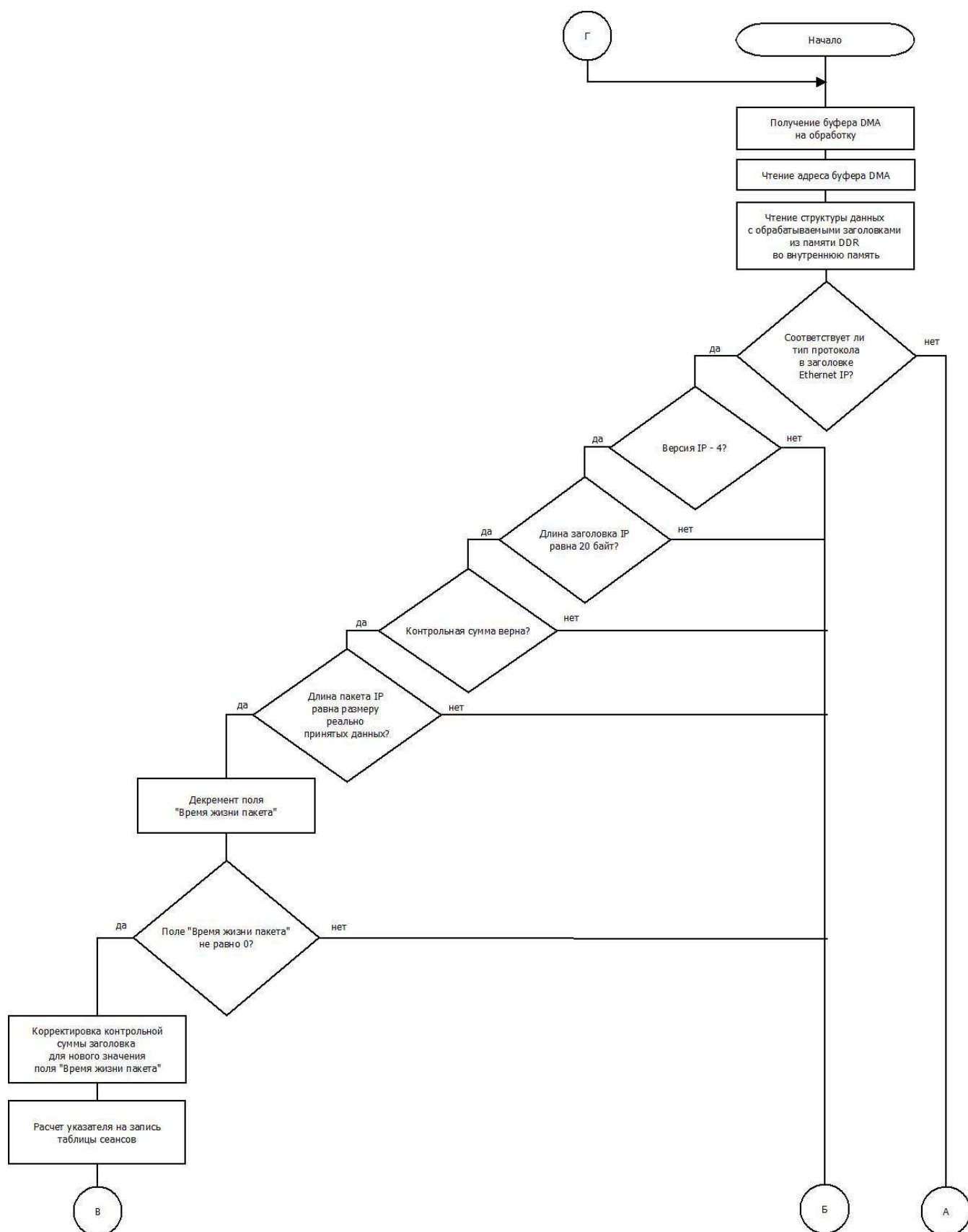


Рисунок Б.1 – Алгоритм обработки входящего трафика коммутатора сообщений абонентской сети, лист 1

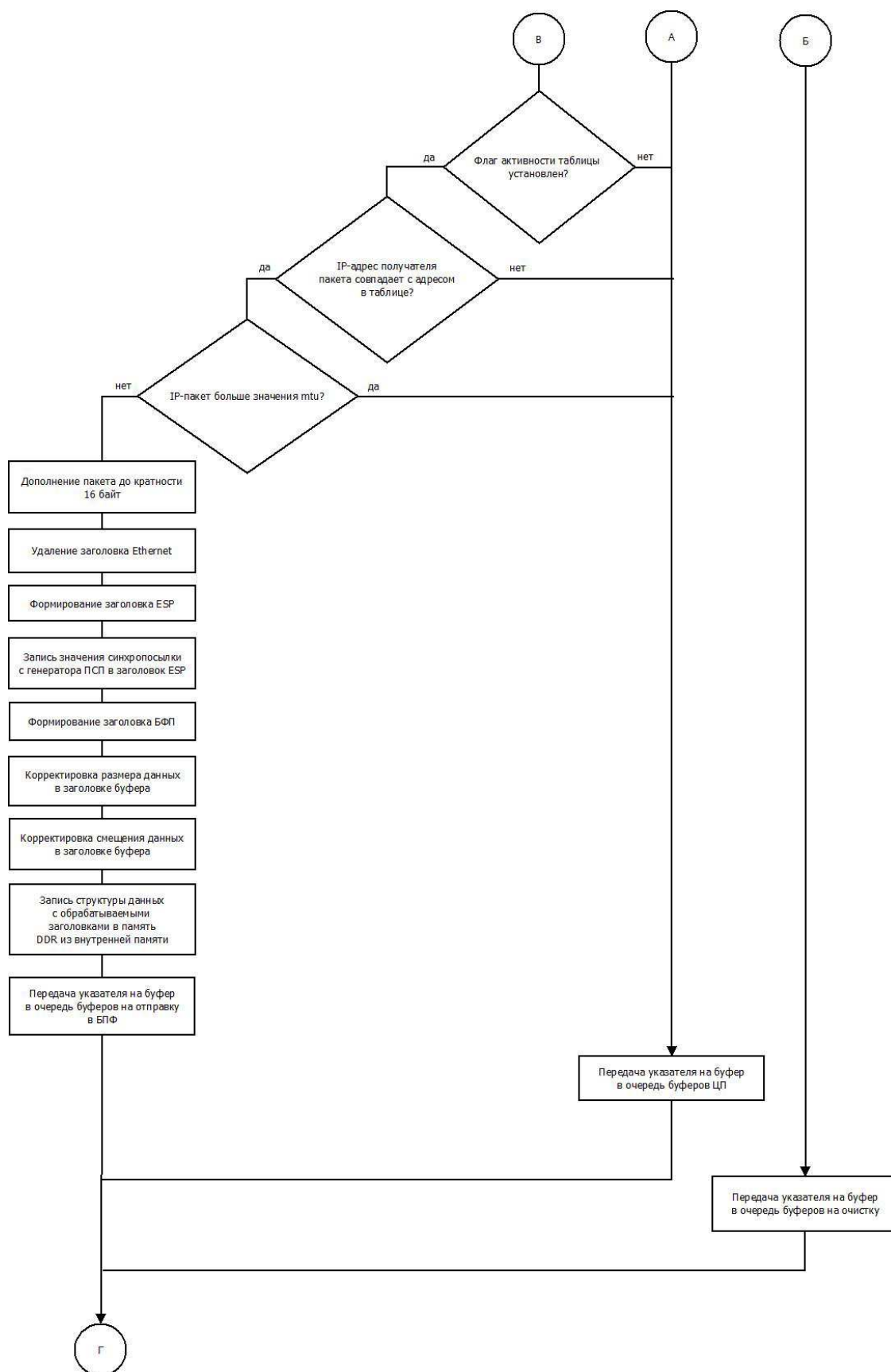


Рисунок Б.1 – Алгоритм обработки входящего трафика коммутатора сообщений абонентской сети, лист 2

ПРИЛОЖЕНИЕ В
(обязательное)
АЛГОРИТМ ОБРАБОТКИ ИСХОДЯЩЕГО ТРАФИКА
КОММУТАТОРА СООБЩЕНИЙ АБОНЕНТСКОЙ СЕТИ

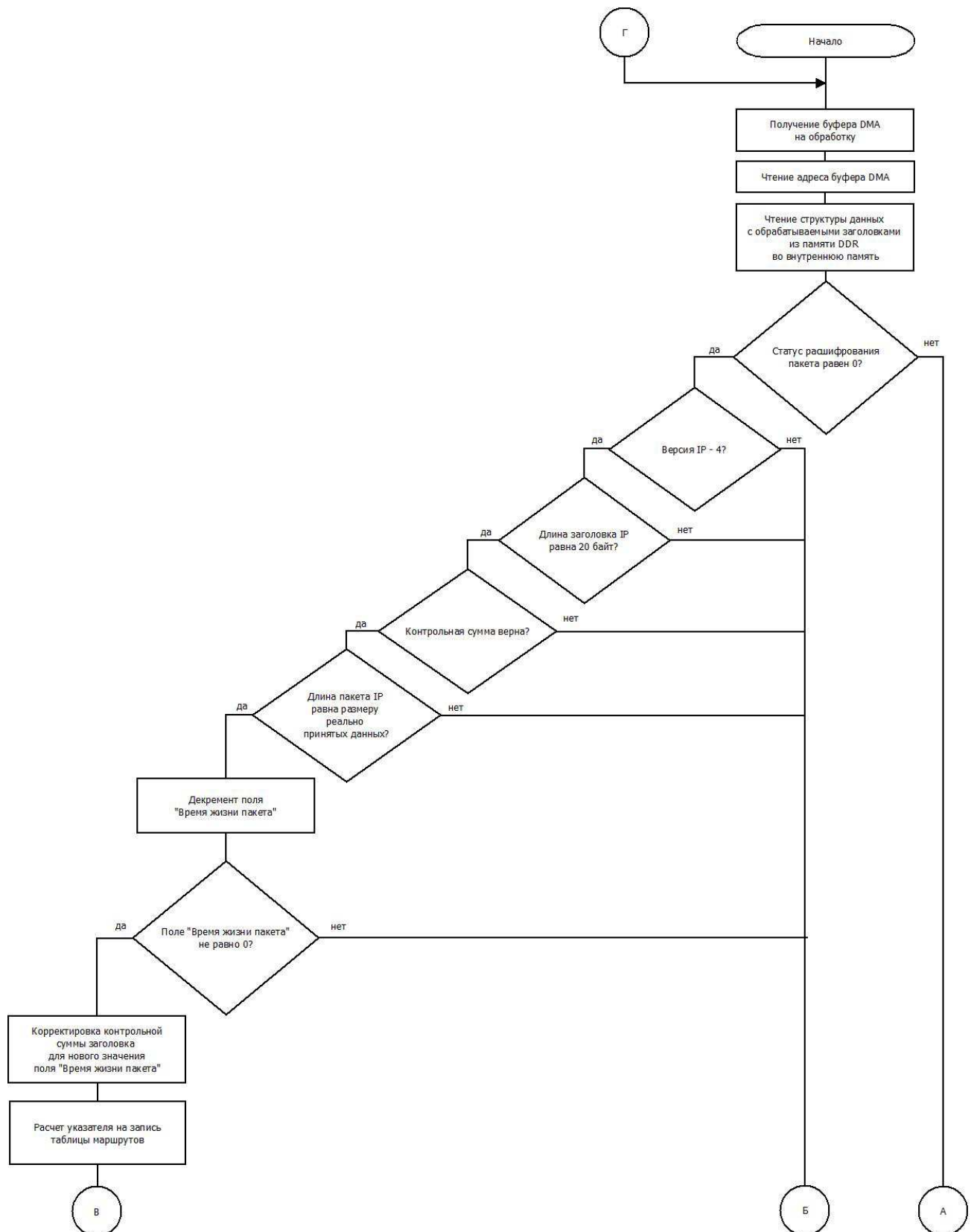


Рисунок В.1 – Алгоритм обработки исходящего трафика коммутатора сообщений абонентской сети, лист 1

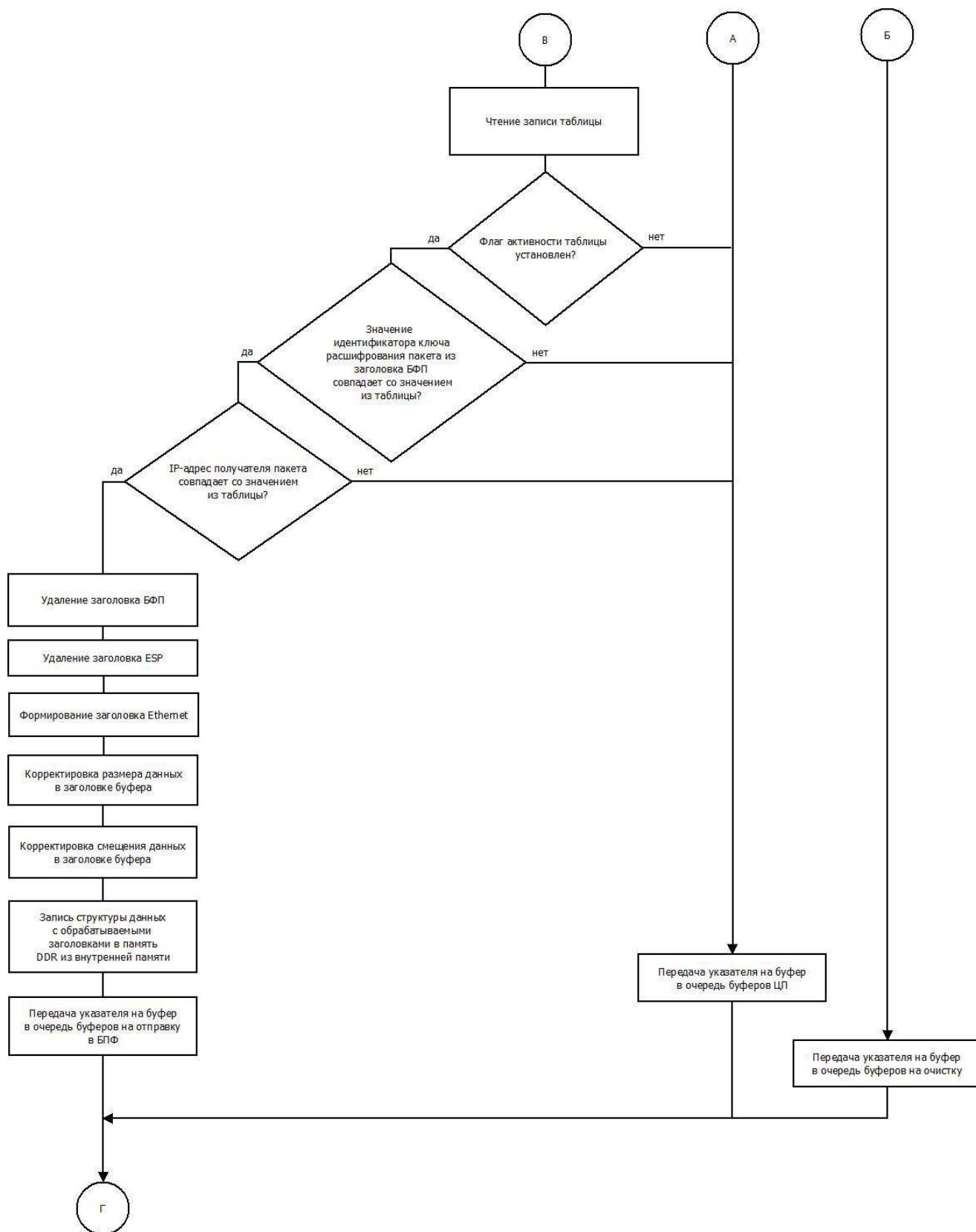
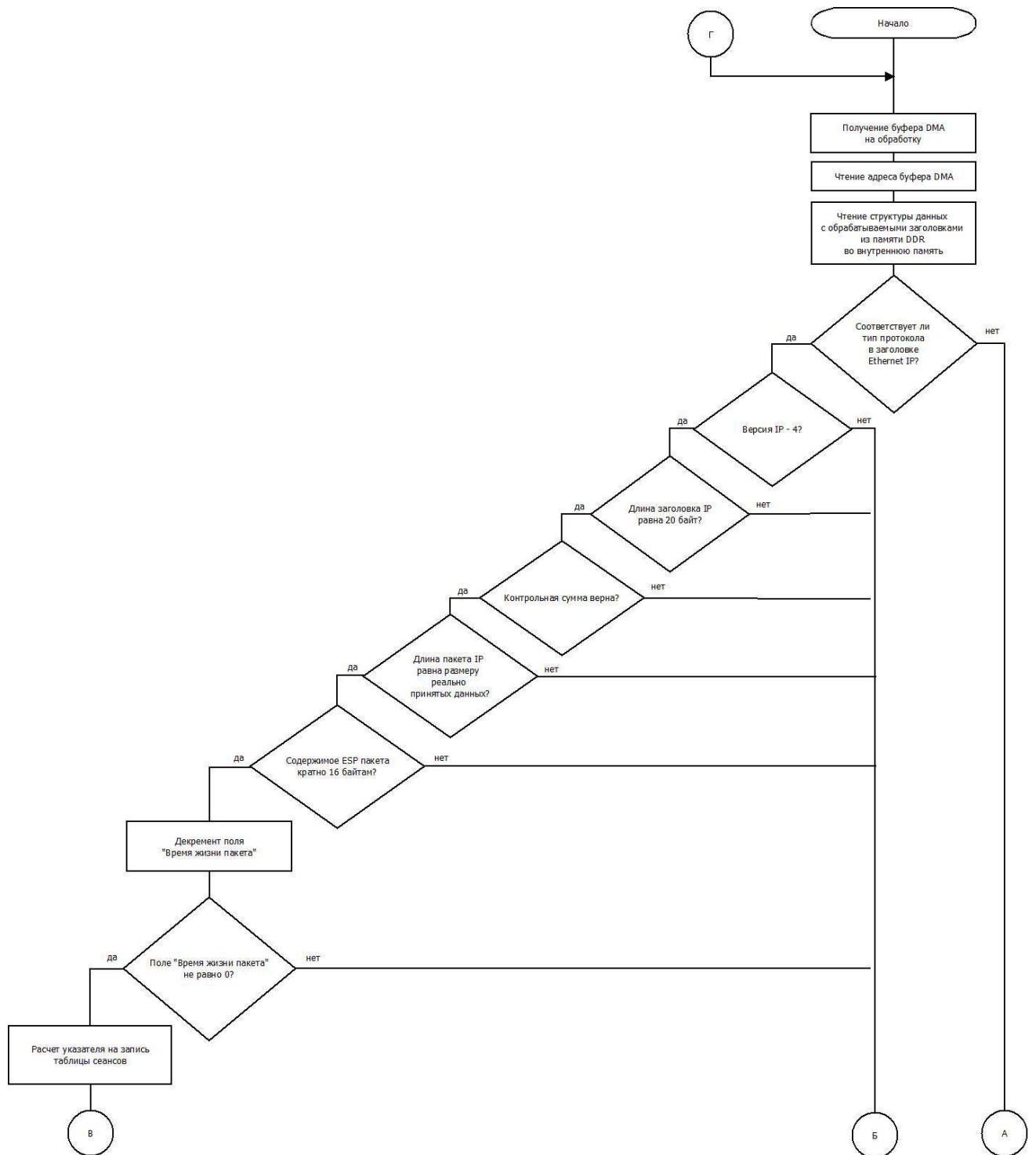
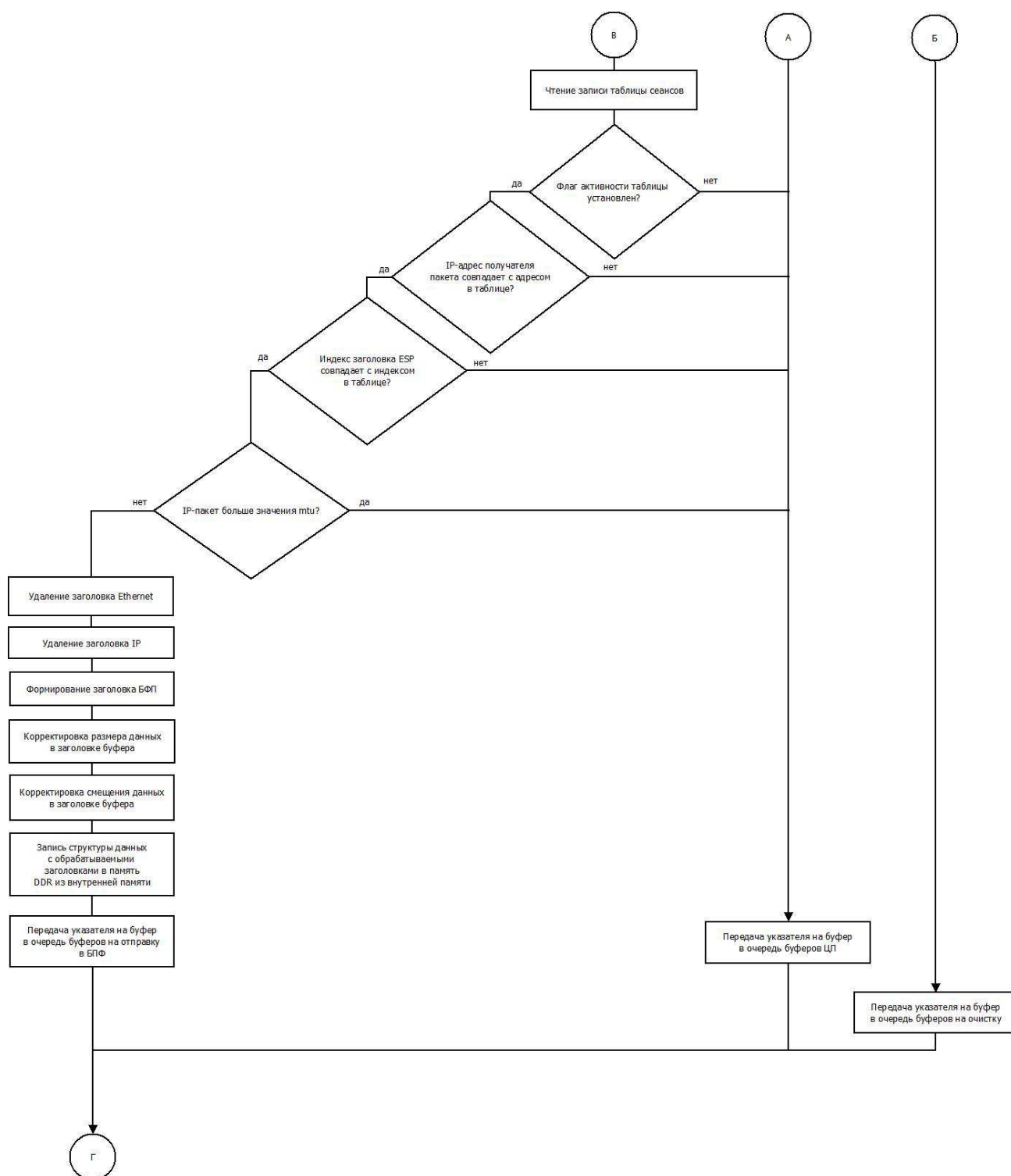


Рисунок В.1 – Алгоритм обработки исходящего трафика коммутатора сообщений абонентской сети, лист 2

ПРИЛОЖЕНИЕ Г
(обязательное)
АЛГОРИТМ ОБРАБОТКИ ВХОДЯЩЕГО ТРАФИКА
КОММУТАТОРА СООБЩЕНИЙ СЕТИ ЗАШИФРОВАННОЙ
ИНФОРМАЦИИ



Риснок Г.1 – Алгоритм обработки входящего трафика коммутатора сообщений сети зашифрованной информации, лист 1



Риснок Г.1 – Алгоритм обработки входящего трафика коммутатора сообщений сети зашифрованной информации, лист 2

ПРИЛОЖЕНИЕ Д
(обязательное)
АЛГОРИТМ ОБРАБОТКИ ИСХОДЯЩЕГО ТРАФИКА
КОММУТАТОРА СООБЩЕНИЙ СЕТИ ЗАШИФРОВАННОЙ
ИНФОРМАЦИИ

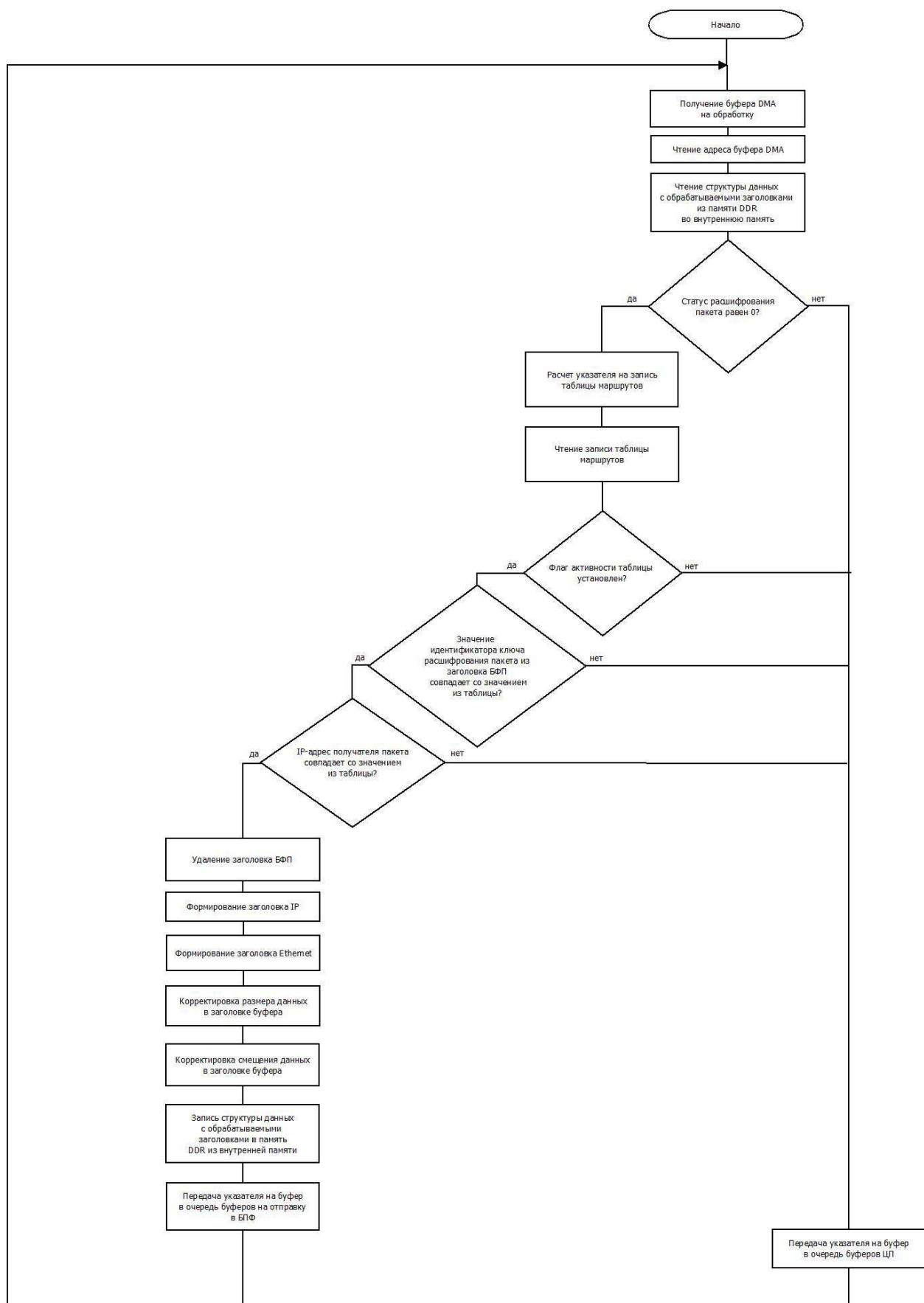


Рисунок Д.1 – Алгоритм обработки исходящего трафика коммутатора сообщений сети зашифрованной информации

ПРИЛОЖЕНИЕ Е
(обязательное)
ИСХОДНЫЕ КОДЫ ПРОГРАММНЫХ ЧАСТЕЙ МОДУЛЯ НА
ЯЗЫКЕ АССЕМБЛЕР ПРОЦЕССОРНОГО ЯДРА

Программа обработки входящего трафика со стороны абонентской сети

```
; Регистры контроллера DMA
constantDMA_CNTRL , F0 ; Адрес регистра управления DMA
constantDMA_HADDR , F1 ; Адрес регистра старшей части адреса памяти DDR
constantDMA_LADDR , F2 ; Адрес регистра младшей части адреса памяти DDR
constantDMA_BADDR , F3 ; Адрес регистра адреса памяти BRAM
constantDMA_SIZE , F4 ; Адрес регистра размера данных
; Регистры контроллера буферов
constantBUF_CNTRL , F5 ; Адрес регистра управления контроллера буферов
constantBUF_HADDR , F6 ; Адрес регистра старшей части адреса буфера
constantBUF_LADDR , F7 ; Адрес регистра младшей части адреса буфера
; Регистры генератора псевдослучайной последовательности
constantRND_DIN , FB ; Адрес регистра записи данных в генератор
constantRND_DOUT , FC ; Адрес регистра чтения данных из генератора
; Регистры модуля расчета контрольной суммы IP
constantCSUM_NEXT , FD ; Адрес регистра управления модулем
constantCSUM_DIN , FE ; Адрес регистра записи данных в модуль
constantCSUM_DOUT , FF ; Адрес регистра чтения данных из модуля
;-----
; Флаги статуса регистра управления контроллера буферов
constantBUF_PROC_REQ , 01 ; Флаг получения буфера на обработку
constantBUF_PROC_COMP , 02 ; Флаг завершения обработки буфера
; Флаги контроля регистра управления контроллера буферов
constantBUF_TO_DEL , 02 ; Флаг отправки буфера в очередь на очистку
constantBUF_TO_CPU , 04 ; Флаг отправки буфера в очередь обработки ЦПУ
constantBUF_TO_TX , 08 ; Флаг отправки буфера в очередь на передачу
; Флаги регистра управления контроллера DMA
constantDMA_BUSY , 01 ; Флаг выполнения операции
constantDMA_COMPLETE , 02 ; Флаг завершения операции
constantDMA_READ , 04 ; Флаг операции чтения из памяти DDR
constantDMA_WRITE , 08 ; Флаг операции записи в память DDR
;-----
; Поля заголовка DMA
constantDMA_SRC , 00 ; Поле идентификатора контроллера DMA
constantDMA_DATA_OFFSET , 00 ; Поле смещения данных в буфере
constantDMA_FRAME_SIZE , 01 ; Поле размера данных в буфере
constantDMA_SN_p1 , 02 ; Поле порядкового номера полученного пакета
constant DMA_SN_p2 , 03 ;
; Полязаголовок Ethernet
constantMAC_DST_p1 , 15 ; Поле адреса получателя
constant MAC_DST_p2 , 16 ;
```

```

constant MAC_DST_p3      , 17 ;
constant MAC_SRC_p1      , 18 ; Поле адреса отправителя
constant MAC_SRC_p2      , 19 ;
constant MAC_SRC_p3      , 1A ;
constant ETH_PROTOCOL    , 1B ; Поле типа протокола
; Поля заголовка IP
constant IP_VERSION      , 1C ; Поле версии IP
constant IP_IHL          , 1C ; Поле размера заголовка IP
constant IP_TOS          , 1C ; Поле «Тип сервиса»
constant IP_TOTAL_LEN    , 1D ; Поле размера пакета IP
constant IP_ID           , 1E ; Поле идентификатора IP пакета
constant IP_FRAG         , 1F ; Поле фрагментации IP пакета
constant IP_PROTOCOL     , 20 ; Поле протокола
constant IP_TTL          , 20 ; Поле время жизни пакета
constant IP_CHECKSUM     , 21 ; Поле контрольной суммы заголовка
constant IP_SRC_p1       , 22 ; Поле адреса отправителя
constant IP_SRC_p2       , 23 ;
constant IP_DST_p1       , 24 ; Поле адреса получателя
constant IP_DST_p2       , 25 ;
; Поля заголовка шифратора
constant ZAS_PKT_ID_p1   , 08 ; Поле идентификатора пакета
constant ZAS_PKT_ID_p2   , 09 ;
constant ZAS_MASK_p1     , 0A ; Поле маски пакета
constant ZAS_MASK_p2     , 0B ;
constant ZAS_KEY_ID      , 0C ; Поле номера ключа
constant ZAS_CONTROL     , 0D ; Поле флагов управления
; Поля заголовка ESP
constant ESP_SPI_p1      , 0E ; Поле SPI
constant ESP_SPI_p2      , 0F ;
constant ESP_SN_p1       , 10 ; Поле порядкового номера
constant ESP_SN_p2       , 11 ;
constant ESP_SYNC_p1     , 12 ; Поле синхропосылки пакета
constant ESP_SYNC_p2     , 13 ;
constant ESP_SYNC_p3     , 14 ;
constant ESP_SYNC_p4     , 15 ;
constant ESP_SYNC_p5     , 16 ;
constant ESP_SYNC_p6     , 17 ;
constant ESP_SYNC_p7     , 18 ;
constant ESP_SYNC_p8     , 19 ;
constant ESP_IM_p1       , 1A ; Поле имитовставки пакета
constant ESP_IM_p2       , 1B ;
; -----

```

```

; Размеры структур данных
constant ETH_HDR_SIZE      , 0E ; Размер заголовка Ethernet
constant IP_HDR_SIZE       , 14 ; Размер заголовка IP
constant ESP_HDR_SIZE      , 1C ; Размер заголовка IP
constant FR_SIZE_CHANGE    , 1A ; Изменение размера данных при обработке
constant BUF_SIZE          , 4C ; Размер данных считываемых из буфера
constant TABLE_SIZE       , 40 ; Размер данных таблицы сеансов
constant NOISE_SIZE        , 04 ; Размер шума
;-----
; Адреса внутренней памяти
constant BUF_BASE          , 00 ; Адрес для данных буфера
constant TABLE_BASE      , 80 ; Адрес для данных таблицы
constant PADDING_BASE      , A0 ; Адрес для данных, используемых для
                                ; выравнивания пакета
constant NOISE_BASE        , C0 ; Адрес для шума
;-----
; Поля таблицы сеансов
constant T_ACTIVE          , 80 ; Флаг активности записи таблицы
constant T_ID              , 81 ; Номер ключа шифрования сеанса
constant T_IP_DST_p1       , 82 ; IP-адрес получателя пакета
constant T_IP_DST_p2       , 83 ;
constant T_SPI_p1          , 84 ; Индекс SPI сеанса
constant T_SPI_p2          , 85 ;
constant T_MTU             , 86 ; Значение MTU канала
;-----
; Точка входа в программу
start : loads0, 00          ;
        load      s1, 00    ;
; Установка констант
Set_masks : callc, FF      ;
load      sF, FF          ;
        call      c, 08     ;
        load      sE, 00    ;
        call      c, 01     ;
        load      sD, 00    ;
        call      c, ff     ;
        load      sC, 00    ;
        call      c, 00     ;
        load      sB, ff     ;
        call      c, 45     ;
        load      sA, 00    ;
        call      c, 12     ;

```

```

        load      s9, 34      ;
        call      c, 03      ;
        load      s8, FF      ;
; Циклоожиданиябуфераанаобработку
Wait_buf_for_proc : input    s2, BUF_CNTRL      ;
                   and      s2, BUF_PROC_REQ    ;
                   jump     z, Wait_buf_for_proc ;
; Чтение данных из буфера
callSet_buffer      ;
callRead_DDR        ;
; Проверка поля тип протокола в заголовке Ethernet
Check_eth_protocol : input    s3, ETH_PROTOCOL  ;
                   xor      s3, sE              ;
                   jump     nz, Push_buf_to_cpu  ;
; Проверкаверсии IP
Check_ip_version :   input    s3, IP_VERSION    ;
                   and      s3, sC              ;
                   xor      s3, sA              ;
                   jump     nz, Push_buf_to_del  ;
; Проверка времени жизни пакета
Check_ttl :          inputs3, IP_TTL            ;
sub      s3, sD      ;
                   load     s7, s3              ;
                   and      s3, sC              ;
                   jump     z, Push_buf_to_del  ;
; Проверкаразмера IP пакета
Check_ip_length :   input    s3, IP_TOTAL_LEN   ;
                   input    s4, DMA_FRAME_SIZE  ;
                   sub      s4, ETH_HDR_SIZE    ;
; Если размер IP больше принятого пакета, пакет является поврежденным
sub      s4, s3      ;
                   jump     c, Push_buf_to_del  ;
; Корректировка размера данных в заголовке DMA
Data_resize :       input    s3, IP_TOTAL_LEN   ;
                   add      s3, ETH_HDR_SIZE    ;
                   output   s3, DMA_FRAME_SIZE  ;
; Проверкаконтрольнойсуммызаголовка IP
Check_checksum :    call     Compute_IP_csum    ;
                   input    s3, CSUM_DOUT       ;
                   or      s3, s0               ;
                   jump     nz, Push_buf_to_del  ;
; Расчет свертки от адреса получателя

```

```

Compute_hash :      inputs3, IP_DST_p2          ;
input      s4, IP_DST_p1          ;
                add      s4, s9              ;
                addcy     s3, 00              ;
                xor       s4, s3              ;
                xor       s4, s9              ;
                load      s3, s4              ;
                sla       s3                  ;
                and       s3, FF              ;
                add       s3, s4              ;
                ands3, s8                  ;

; Преобразование свертки в указатель на запись таблицы
Convert_hash_to_ptr :  load      s5, 06          ;
                    call      Left_cycle_roll    ;
                    load      s4, s3          ;
                    and       s4, 3F          ;
                    add       s4, 20          ;
                    output    s4, DMA_HADDR      ;
                    call      c, FF          ;
                    and       s3, C0          ;
                    outputs3, DMA_LADDR        ;

; Установка параметров операции DMA
loads3, 80          ;
output    s3, DMA_BADDR      ;
                load      s3, TABLE_SIZE      ;
                output    s3, DMA_SIZE          ;

; Чтение записи таблицы
callRead_DDR          ;

; Проверка активности записи таблицы
Check_Active :      inputs3, T_ACTIVE          ;
call      c , 00          ;
                xor       s3, 01              ;
                jump      nz, Push_buf_to_cpu    ;

; Проверка адреса получателя по таблице
Check_table_record :  inputs3, T_IP_DST_p1      ;
input      s4, IP_DST_p1          ;
                xor       s3, s4              ;
                jump      nz, Push_buf_to_cpu    ;
                input    s3, T_IP_DST_p2        ;
                input    s4, IP_DST_p2          ;
                xor       s3, s4              ;
                jump      nz, Push_buf_to_cpu    ;

```

```

; Проверка размера пакета на MTU
Check_mtu :          inputs3, IP_TOTAL_LEN          ;
input    s4, T_MTU          ;
                                sub    s4, s3          ;
                                jump    c, Push_buf_to_cpu    ;

; Выравнивание пакета до кратности 16 байт
Compute_padding_size :    loads4, PADDING_BASE          ;

; Проверка кратности размера IP пакета 16 байтам
input    s3, IP_TOTAL_LEN          ;
                                and    s3, 0F          ;
                                load    s5, 0F          ;
                                sub    s5, s3          ;
                                load    s3, s5          ;
                                sr0    s5          ;
                                sr0    s5          ;
                                jump    z, Write_type      ;

; Цикл записи дополнения в конец пакета
Padding_loop :          output    s0, (s4)          ;
                                add    s4, 01          ;
                                output    s0, (s4)          ;
                                add    s4, 01          ;
                                sub    s5, 01          ;
                                jump    nz, Padding_loop    ;

; Запись типа пакета в последний байт дополнения
Write_type :          output    s0, (s4)          ;
                                add    s4, 01          ;
                                call    c, 00          ;
                                load    s2, 04          ;
                                outputs2, (s4)          ;

; Вычисление адреса конца данных IP пакета
Compute_packet_end :    input    s2, DMA_FRAME_SIZE          ;
                                input    s4, DMA_DATA_OFFSET          ;
                                and    s4, sB          ;
                                add    s4, s2          ;

; Изменение размера данных в заголовке буфера
Frame_resize :          add    s3, 01          ;
                                input    s2, DMA_FRAME_SIZE          ;
                                add    s2, s3          ;
                                output    s2, DMA_FRAME_SIZE          ;

; Запись конца пакета в память
Write_packet_end :          input    s2, BUF_HADDR          ;
                                output    s2, DMA_HADDR          ;

```

```

        input      s2, BUF_LADDR          ;
        add        s2, s4                  ;
        output     s2, DMA_LADDR          ;
        load       s2, PADDING_BASE       ;
        output     s2, DMA_BADDR          ;
        output     s3, DMA_SIZE           ;
        call       Write_DDR              ;

; Обработка полей заголовка IP
; Уменьшение поля TTL на 1
Update_ip_header :      output     s7, IP_TTL          ;
; Пересчет контрольной суммы заголовка
outputs0, IP_CHECKSUM      ;
call      Compute_IP_csum      ;
        inputs3, CSUM_DOUT          ;

; Сохранение новой контрольной суммы в заголовок
outputs3, IP_CHECKSUM      ;
; Формирование заголовка ESP
; Копирование поля индекса из таблицы
Insert_ESP_header :      inputs3, T_SPI_p1          ;
output     s3, ESP_SPI_p1          ;
        input     s3, T_SPI_p2          ;
        output     s3, ESP_SPI_p2          ;

; Резервирования поля под порядковый номер
outputs0, ESP_SN_p1          ;
output     s0, ESP_SN_p2          ;

; Запись псевдослучайной последовательности в поле синхропосылки
Write_sync :      load      s4, ESP_SYNC_p1          ;
        load      s5, 08                  ;
Next_sync_word :      input     s3, RND_DOUT          ;
        output     s3, (s4)              ;
        add       s4, 01                  ;
        sub       s5, 01                  ;
        jump      nz, Next_sync_word      ;

; Резервирование поля под имитовставку
outputs0, ESP_IM_p1          ;
output     s0, ESP_IM_p2          ;
; Формирование заголовка шифратора
; Резервирование поля под маску
Insert_ZAS_header :      output     s0, ZAS_MASK_p1          ;
        output     s0, ZAS_MASK_p2          ;

; Копирование порядкового номера пакета из заголовка буфера
input      s3, DMA_SN_p2          ;

```

```

                                output    s3, ZAS_PKT_ID_p2        ;
                                input     s3, DMA_SN_p1           ;
                                output    s3, ZAS_PKT_ID_p1        ;
; Копирование номера ключа из таблицы
inputs3, T_ID                ;
output    s3, ZAS_KEY_ID      ;
; Установка флагов управления в 0
outputs0, ZAS_CONTROL        ;
; Обработка полей заголовка буфера
; Изменение размера данных
Frame_size_update    :    input    s3, DMA_FRAME_SIZE        ;
                        add        s3, FR_SIZE_CHANGE          ;
                        output    s3, DMA_FRAME_SIZE          ;
; Изменение смещения данных
Frame_offset_update  :    input    s3, DMA_DATA_OFFSET        ;
                        sub        s3, FR_SIZE_CHANGE          ;
                        output    s3, DMA_DATA_OFFSET          ;
; Запись данных буфера в память
callSet_buffer        ;
callWrite_DDR         ;
; Проверка необходимости обновления заполнения генератора
Rnd_refresh_check    :    and        s1, 7F                    ;
                        jump       nz, Push_buf_to_tx          ;
; Чтение шума из памяти
Read_noise    :        load        s3, 40                      ;
                        output     s4, DMA_HADDR               ;
                        load        s3, s0                     ;
                        output     s3, DMA_LADDR               ;
                        load        s3, NOISE_BASE              ;
                        output     s3, DMA_BADDR               ;
                        load        s3, NOISE_SIZE              ;
                        output     s3, DMA_SIZE                 ;
                        callRead_DDR                            ;
; Запись шума в генератор псевдослучайной последовательности
Rnd_write    :        input     s3, NOISE_BASE                ;
                        outputs3, RND_DIN                      ;
; Передача буфера в очередь на отправку
Push_buf_to_tx :        add        s1, 01                      ;
                        load        s2, BUF_TO_TX              ;
                        output     s2, BUF_CNTRL               ;
                        jump       Wait_buf_for_proc            ;
; Передача буфера в очередь на обработку ЦПУ

```

```

Push_buf_to_cpu :      load      s2, BUF_TO_CPU      ;
                       output    s2, BUF_CNTRL      ;
                       jump      Wait_buf_for_proc   ;

; Передача буфера в очередь на очистку
Push_buf_to_cpu :      load      s2, BUF_TO_DEL      ;
                       output    s2, BUF_CNTRL      ;
                       jump      Wait_buf_for_proc   ;

; -----
; Функции
; Установка параметров для операции с буфером
Set_buffer :           input     s3, BUF_HADDR      ;
                       output    s3, DMA_HADDR      ;
                       input     s3, BUF_LADDR      ;
                       output    s3, DMA_LADDR      ;
                       load      s3, s0             ;
                       output    s3, DMA_BADDR      ;
                       load      s3, BUF_SIZE       ;
                       output    s3, DMA_SIZE       ;
                       return    ;

; Чтениеизпамяти DDR
Read_DDR :             load      s3, DMA_READ      ;
                       output    s3, DMA_CNTRL      ;
                       jump      Wait_DMA_complete  ;

; Записьвпамять DDR
Write_DDR :            load      s3, DMA_WRITE     ;
                       output    s3, DMA_CNTRL      ;

Wait_DMA_complete :    input     s3, DMA_CNTRL      ;
                       and       s3, DMA_COMPLETE   ;
                       jump      z, Wait_DMA_complete ;
                       return    ;

; Циклическийсдвигвлево
Left_cycle_roll :      rl       s3                ;
                       sub       s5, 01            ;
                       jump      nz, Left_cycle_roll ;
                       return    ;

; Расчет контрольной суммы заголовка IP
Compute_IP_csum :      output    s0, CSUM_NEXT      ;
                       load      s2, IP_VERSION     ;
                       load      s5, 0A            ;

Next_csum_data :       input     s3, (s2)          ;
                       output    s3, CSUM_DIN       ;

```

```
add      s2, 01      ;  
sub      s5, 01      ;  
jump     nz, Next_csum_data ;  
return                               ;
```

Программа обработки исходящего трафика для абонентской сети

```
; Регистры контроллера DMA
constantDMA_CNTRL , F0 ; Адрес регистра управления DMA
constantDMA_HADDR , F1 ; Адрес регистра старшей части адреса памяти DDR
constantDMA_LADDR , F2 ; Адрес регистра младшей части адреса памяти DDR
constantDMA_BADDR , F3 ; Адрес регистра адреса памяти BRAM
constantDMA_SIZE , F4 ; Адрес регистра размера данных
; Регистры контроллера буферов
constantBUF_CNTRL , F5 ; Адрес регистра управления контроллера буферов
constantBUF_HADDR , F6 ; Адрес регистра старшей части адреса буфера
constantBUF_LADDR , F7 ; Адрес регистра младшей части адреса буфера
; Регистры генератора псевдослучайной последовательности
constantRND_DIN , FB ; Адрес регистра записи данных в генератор
constantRND_DOUT , FC ; Адрес регистра чтения данных из генератора
; Регистры модуля расчета контрольной суммы IP
constantCSUM_NEXT , FD ; Адрес регистра управления модулем
constantCSUM_DIN , FE ; Адрес регистра записи данных в модуль
constantCSUM_DOUT , FF ; Адрес регистра чтения данных из модуля
;-----
; Флаги статуса регистра управления контроллера буферов
constantBUF_PROC_REQ , 01 ; Флаг получения буфера на обработку
constantBUF_PROC_COMP , 02 ; Флаг завершения обработки буфера
; Флаги контроля регистра управления контроллера буферов
constantBUF_TO_DEL , 02 ; Флаг отправки буфера в очередь на очистку
constantBUF_TO_CPU , 04 ; Флаг отправки буфера в очередь обработки ЦПУ
constantBUF_TO_TX , 08 ; Флаг отправки буфера в очередь на передачу
; Флаги регистра управления контроллера DMA
constantDMA_BUSY , 01 ; Флаг выполнения операции
constantDMA_COMPLETE , 02 ; Флаг завершения операции
constantDMA_READ , 04 ; Флаг операции чтения из памяти DDR
constantDMA_WRITE , 08 ; Флаг операции записи в память DDR
;-----
; Поля заголовка DMA
constantDMA_SRC , 00 ; Поле идентификатора контроллера DMA
constantDMA_DATA_OFFSET , 00 ; Поле смещения данных в буфере
constantDMA_FRAME_SIZE , 01 ; Поле размера данных в буфере
constantDMA_SN_p1 , 02 ; Поле порядкового номера полученного пакета
constant DMA_SN_p2 , 03 ;
; Полязаголовок Ethernet
constantMAC_DST_p1 , 25 ; Поле адреса получателя
constant MAC_DST_p2 , 26 ;
constant MAC_DST_p3 , 27 ;
```

```

constant MAC_SRC_p1      , 28 ; Поле адреса отправителя
constant MAC_SRC_p2      , 29 ;
constant MAC_SRC_p3      , 2A ;
constant ETH_PROTOCOL     , 2B ; Поле типа протокола
; Поля заголовка IP
constant IP_VERSION       , 2C ; Поле версии IP
constant IP_IHL           , 2C ; Поле размера заголовка IP
constant IP_TOS           , 2C ; Поле «Тип сервиса»
constant IP_TOTAL_LEN     , 2D ; Поле размера пакета IP
constant IP_ID            , 2E ; Поле идентификатора IP пакета
constant IP_FRAG          , 2F ; Поле фрагментации IP пакета
constant IP_PROTOCOL      , 30 ; Поле протокола
constant IP_TTL           , 30 ; Поле время жизни пакета
constant IP_CHECKSUM      , 31 ; Поле контрольной суммы заголовка
constant IP_SRC_p1        , 32 ; Поле адреса отправителя
constant IP_SRC_p2        , 33 ;
constant IP_DST_p1        , 34 ; Поле адреса получателя
constant IP_DST_p2        , 35 ;
; Поля заголовка шифратора
constant ZAS_PKT_ID_p1    , 18 ; Поле идентификатора пакета
constant ZAS_PKT_ID_p2    , 19 ;
constant ZAS_MASK_p1      , 1A ; Поле маски пакета
constant ZAS_MASK_p2      , 1B ;
constant ZAS_KEY_ID       , 1C ; Поле номера ключа
constant ZAS_CONTROL      , 1D ; Поле флагов управления
; Поля заголовка ESP
constant ESP_SPI_p1       , 1E ; Поле SPI
constant ESP_SPI_p2       , 1F ;
constant ESP_SN_p1        , 20 ; Поле порядкового номера
constant ESP_SN_p2        , 21 ;
; -----
; Размеры структур данных
constant ZAS_HDR_SIZE     , 0C ; Размер заголовка шифратора
constant ETH_HDR_SIZE     , 0E ; Размер заголовка Ethernet
constant IP_HDR_SIZE      , 14 ; Размер заголовка IP
constant ESP_HDR_SIZE     , 1C ; Размер заголовка ESP
constant FR_SIZE_CHANGE   , 1A ; Изменение размера данных при обработке
; -----
; Поля записи таблицы маршрутов
constant T_ACTIVE         , 80 ; Флаг активности записи
constant T_ID             , 81 ; Номер ключа расшифрования
constant T_MAC_DST_p1     , 82 ; MAC-адрес получателя

```

```

constant T_MAC_DST_p2      , 83 ;
constant T_MAC_DST_p3      , 84 ;
constant T_MAC_SRC_p1      , 85 ; MAC-адресотправителя
constant T_MAC_SRC_p2      , 86 ;
constant T_MAC_SRC_p3      , 87 ;
constant T_IP_SRC_p1       , 88 ; IP-адрес отправителя
constant T_IP_SRC_p2       , 89 ;
constant T_IP_DST_p1       , 8A ; IP-адресполучателя
constant T_IP_DST_p2       , 8B ;
;-----
; Точка входа в программу
start :          loads0, 00 ;
; Установка констант
Set_masks :      callc, FF ;
load      sF, FF ;

                call      c, 08 ;
                load      sE, 00 ;
                call      c, 01 ;
                load      sD, 00 ;
                call      c, ff ;
                load      sC, 00 ;
                call      c, 00 ;
                load      sB, ff ;
                call      c, 45 ;
                load      sA, 00 ;
                call      c, 12 ;
                load      s9, 34 ;
                call      c, 03 ;
                loads8, FF ;

; Цикложданиябуфераобработку
Wait_buf_for_proc : input    s2, BUF_CNTRL ;
                    and      s2, BUF_PROC_REQ ;
                    jump     z, Wait_buf_for_proc ;

; Чтение данных из буфера
callSet_buffer ;
callRead_DDR ;

; Проверкастатусасшифрования
Check_zas_status : input    s3, ZAS_CONTROL ;
                    or       s3, s0 ;
                    jump     nz, Push_buf_to_cpu ;

; Проверкаверсии IP
Check_ip_version : input    s3, IP_VERSION ;

```

```

        and      s3, sC          ;
        xor      s3, sA          ;
        jump     nz, Push_buf_to_del ;

; Проверка времени жизни пакета
Check_ttl :      inputs3, IP_TTL          ;
sub      s3, sD          ;
        load     s7, s3          ;
        and      s3, sC          ;
        jump     z, Push_buf_to_del ;

; Проверка длины IP пакета
Check_length :   input     s3, IP_TOTAL_LEN      ;
                input     s4, DMA_FRAME_SIZE    ;
                sub      s4, ZAS_HDR_SIZE        ;
                sub      s4, ESP_HDR_SIZE        ;
                load     s2, s4          ;
                sub      s4, s3          ;
                jump     c, Push_buf_to_del ;

; Проверка кратности содержимого ESP пакета 16 байтам
Check_rate :     and      s2, 0F          ;
                jump     nz, Push_buf_to_del ;

; Проверка контрольной суммы заголовка IP
Check_checksum : call     Compute_IP_csum      ;
                input     s3, CSUM_DOUT        ;
                or      s3, s0          ;
                jump     nz, Push_buf_to_del ;

; Расчет свертки от адреса получателя
Compute_hash :   inputs3, IP_DST_p2          ;
input     s4, IP_DST_p1          ;
        add      s4, s9          ;
        addc     s3, 00          ;
        xor      s4, s3          ;
        xor      s4, s9          ;
        load     s3, s4          ;
        sla      s3          ;
        and      s3, FF          ;
        add      s3, s4          ;
        ands     s3, s8          ;

; Преобразование свертки в указатель на запись таблицы
Convert_hash_to_ptr : load     s5, 06          ;
                    call     Left_cycle_roll    ;
                    load     s4, s3          ;
                    and      s4, 3F          ;

```

```

                                add      s4, 20                ;
                                output   s4, DMA_HADDR        ;
                                call     c, FF                ;
                                and      s3, C0                ;
                                outputs3, DMA_LADDR           ;
; Установка параметров операции DMA
loads3, 80                    ;
output   s3, DMA_BADDR        ;
                                load     s3, 40                ;
                                output   s3, DMA_SIZE          ;
; Чтение записи таблицы
callRead_DDR                  ;
; Проверка активности записи таблицы
Check_Active :                inputs3, T_ACTIVE              ;
call     c , 00                ;
                                xor      s3, 01                ;
                                jump     nz, Push_buf_to_cpu    ;
; Проверка корректности использованного ключа
Check_ID :                    inputs3, T_ID                  ;
input    s4, ZAS_KEY_ID        ;
                                xor      s3, s4                ;
                                jump     nz, Push_buf_to_cpu    ;
; Проверка адреса получателя по таблице
Check_table_record :         inputs3, T_IP_DST_p1            ;
input    s4, IP_DST_p1         ;
                                xor      s3, s4                ;
                                jump     nz, Push_buf_to_cpu    ;
                                input    s3, T_IP_DST_p2        ;
                                input    s4, IP_DST_p2          ;
                                xor      s3, s4                ;
                                jump     nz, Push_buf_to_cpu    ;
; Расчет указателя на конец пакета
Compute_type_ptr :           input    s2, DMA_FRAME_SIZE     ;
                                input    s4, DMA_DATA_OFFSET   ;
                                and      s4, sB                ;
                                add      s4, s2                ;
                                sub      s4, 04                ;
; Чтение поля типа
Read_packet_end :            input    s2, BUF_HADDR          ;
                                output   s2, DMA_HADDR        ;
                                input    s2, BUF_LADDR          ;
                                add      s2, s4                ;

```

```

        output    s2, DMA_LADDR          ;
        load      s2, A0                  ;
        output    s2, DMA_BADDR          ;
        load      s2, 04                  ;
        output    s2, DMA_SIZE            ;
        callRead_DDR                        ;
; Проверка типа пакета на соответствие пакету с данными
Check_type :      input    s4, A1          ;
                  xor      s4, 04          ;
                  jump     nz, Push_buf_to_cpu ;
; Обработка полей заголовка IP
; Уменьшение поля TTL на 1
Update_ip_header :      output    s7, IP_TTL          ;
; Пересчет контрольной суммы заголовка
outputs0, IP_CHECKSUM      ;
call      Compute_IP_csum      ;
                  input    s3, CSUM_DOUT          ;
; Сохранение новой контрольной суммы в заголовок
outputs3, IP_CHECKSUM      ;
; Формирование заголовка Ethernet по таблице
Insert_eth_header :      input    s3, T_MAC_DST_p1    ;
                          output   s3, MAC_DST_p1     ;
                          input    s3, T_MAC_DST_p2    ;
                          output   s3, MAC_DST_p2     ;
                          input    s3, T_MAC_DST_p3    ;
                          output   s3, MAC_DST_p3     ;
                          input    s3, T_MAC_SRC_p1    ;
                          output   s3, MAC_SRC_p1     ;
                          input    s3, T_MAC_SRC_p2    ;
                          output   s3, MAC_SRC_p2     ;
                          input    s3, T_MAC_SRC_p3    ;
                          output   s3, MAC_SRC_p3     ;
                          outputsE, ETH_PROTOCOL      ;
; Обработка полей заголовка буфера
; Изменениеразмераданныхбуфера
Frame_size_update :      input    s3, IP_TOTAL_LEN    ;
                          add      s3, ETH_HDR_SIZE    ;
; Проверка размера пакета на минимальный размер (60 байт)
Check_little_frame :      load     s4, s3              ;
                          sub      s4, 3c              ;
                          jump     nc, Write_size      ;
; Установка размера данных для коротких пакетов

```

```

callc, 00 ;
loads3, 3c ;
; Запись размера данных в заголовок
Write_size : outputs3, DMA_FRAME_SIZE ;
; Изменение смещения данных в буфере
Frame_offset_update : inputs3, DMA_DATA_OFFSET ;
add s3, FR_SIZE_CHANGE ;
output s3, DMA_DATA_OFFSET ;
; Запись данных в буфер
callSet_buffer ;
callWrite_DDR ;
; Передача буфера в очередь на отправку
Push_buf_to_tx : load s2, BUF_TO_TX ;
output s2, BUF_CNTRL ;
jump Wait_buf_for_proc ;
; Передача буфера в очередь на обработку ЦПУ
Push_buf_to_cpu : load s2, BUF_TO_CPU ;
output s2, BUF_CNTRL ;
jump Wait_buf_for_proc ;
; Передача буфера в очередь на очистку
Push_buf_to_cpu : load s2, BUF_TO_DEL ;
output s2, BUF_CNTRL ;
jump Wait_buf_for_proc ;
; -----
; Функции
; Установка параметров для операции с буфером
Set_buffer : input s3, BUF_HADDR ;
output s3, DMA_HADDR ;
input s3, BUF_LADDR ;
output s3, DMA_LADDR ;
load s3, s0 ;
output s3, DMA_BADDR ;
load s3, BUF_SIZE ;
output s3, DMA_SIZE ;
return ;
; Чтение из памяти DDR
Read_DDR : load s3, DMA_READ ;
output s3, DMA_CNTRL ;
jump Wait_DMA_complete ;
; Запись в память DDR
Write_DDR : load s3, DMA_WRITE ;
output s3, DMA_CNTRL ;

```

```

Wait_DMA_complete :   input    s3, DMA_CNTRL           ;
                      and      s3, DMA_COMPLETE        ;
                      jump     z, Wait_DMA_complete    ;
                      return                                ;

; Циклический сдвиг влево
Left_cycle_roll    :   rl      s3                      ;
                      sub      s5, 01                  ;
                      jump     nz, Left_cycle_roll      ;
                      return                                ;

; Расчет контрольной суммы заголовка IP
Compute_IP_csum    :   output   s0, CSUM_NEXT          ;
                      load     s2, IP_VERSION          ;
                      load     s5, 0A                  ;
Next_csum_data     :   input    s3, (s2)              ;
                      output   s3, CSUM_DIN            ;
                      add      s2, 01                  ;
                      sub      s5, 01                  ;
                      jump     nz, Next_csum_data       ;
                      return                                ;

```

Программа обработки входящего трафика со стороны сети зашифрованной информации

```
; Регистры контроллера DMA
constantDMA_CNTRL , F0 ; Адрес регистра управления DMA
constantDMA_HADDR , F1 ; Адрес регистра старшей части адреса памяти DDR
constantDMA_LADDR , F2 ; Адрес регистра младшей части адреса памяти DDR
constantDMA_BADDR , F3 ; Адрес регистра адреса памяти BRAM
constantDMA_SIZE , F4 ; Адрес регистра размера данных
; Регистры контроллера буферов
constantBUF_CNTRL , F5 ; Адрес регистра управления контроллера буферов
constantBUF_HADDR , F6 ; Адрес регистра старшей части адреса буфера
constantBUF_LADDR , F7 ; Адрес регистра младшей части адреса буфера
; Регистры генератора псевдослучайной последовательности
constantRND_DIN , FB ; Адрес регистра записи данных в генератор
constantRND_DOUT , FC ; Адрес регистра чтения данных из генератора
; Регистры модуля расчета контрольной суммы IP
constantCSUM_NEXT , FD ; Адрес регистра управления модулем
constantCSUM_DIN , FE ; Адрес регистра записи данных в модуль
constantCSUM_DOUT , FF ; Адрес регистра чтения данных из модуля
;-----
; Флаги статуса регистра управления контроллера буферов
constantBUF_PROC_REQ , 01 ; Флаг получения буфера на обработку
constantBUF_PROC_COMP , 02 ; Флаг завершения обработки буфера
; Флаги контроля регистра управления контроллера буферов
constantBUF_TO_DEL , 02 ; Флаг отправки буфера в очередь на очистку
constantBUF_TO_CPU , 04 ; Флаг отправки буфера в очередь обработки ЦПУ
constantBUF_TO_TX , 08 ; Флаг отправки буфера в очередь на передачу
; Флаги регистра управления контроллера DMA
constantDMA_BUSY , 01 ; Флаг выполнения операции
constantDMA_COMPLETE , 02 ; Флаг завершения операции
constantDMA_READ , 04 ; Флаг операции чтения из памяти DDR
constantDMA_WRITE , 08 ; Флаг операции записи в память DDR
;-----
; Поля заголовка DMA
constantDMA_SRC , 00 ; Поле идентификатора контроллера DMA
constantDMA_DATA_OFFSET , 00 ; Поле смещения данных в буфере
constantDMA_FRAME_SIZE , 01 ; Поле размера данных в буфере
constantDMA_SN_p1 , 02 ; Поле порядкового номера полученного пакета
constant DMA_SN_p2 , 03 ;
; Поля заголовка Ethernet
constantMAC_DST_p1 , 0D ; Поле адреса получателя
```

```

constant MAC_DST_p2      , 0E ;
constant MAC_DST_p3      , 0F ;
constant MAC_SRC_p1      , 10 ; Поле адреса отправителя
constant MAC_SRC_p2      , 11 ;
constant MAC_SRC_p3      , 12 ;
constant ETH_PROTOCOL     , 13 ; Полетиппротокола
; Полязаголовок IP
constant IP_VERSION       , 14 ; Полеверсии IP
constant IP_IHL           , 14 ; ПолеразмеразаголовокIP
constant IP_TOS           , 14 ; Поле «Типсервиса»
constant IP_TOTAL_LEN     , 15 ; Полеразмерapakета IP
constant IP_ID            , 16 ; Поле идентификатора IP пакета
constant IP_FRAG          , 17 ; Поле фрагментации IP пакета
constant IP_PROTOCOL      , 18 ; Поле протокола
constant IP_TTL           , 18 ; Поле время жизни пакета
constant IP_CHECKSUM       , 19 ; Поле контрольной суммы заголовка
constant IP_SRC_p1        , 1A ; Поле адреса отправителя
constant IP_SRC_p2        , 1B ;
constant IP_DST_p1        , 1C ; Поле адреса получателя
constant IP_DST_p2        , 1D ;
; Полязаголовок ESP
constant ESP_SPI_p1       , 1E ; Поле SPI
constant ESP_SPI_p2       , 1F ;
constant ESP_SN_p1        , 20 ; Поле порядкового номера
constant ESP_SN_p2        , 21 ;
; Поля заголовка шифратора
constant ZAS_PKT_ID_p1    , 18 ; Поле идентификатора пакета
constant ZAS_PKT_ID_p2    , 19 ;
constant ZAS_MASK_p1      , 1A ; Поле маскпакета
constant ZAS_MASK_p2      , 1B ;
constant ZAS_KEY_ID       , 1C ; Поле номераключа
constant ZAS_CONTROL      , 1D ; Поле флаговуправления
;-----
; Размеры структур данных
constant ZAS_HDR_SIZE     , 0C ; Размер заголовка шифратора
constant ETH_HDR_SIZE     , 0E ; Размерзаголовок Ethernet
constant IP_HDR_SIZE      , 14 ; Размерзаголовок IP
constant ESP_HDR_SIZE     , 1C ; Размерзаголовок ESP
constant FR_SIZE_CHANGE   , 16 ; Изменение размера данных при обработке
; Поля записи таблицы сеансов
constant T_ACTIVE         , 80 ; Флаг активности записи таблицы
constant T_ID             , 81 ; Номер ключа расшифрования

```

```

constant T_IP_SRC_p1      , 82 ; IP-адрес отправителя
constant T_IP_SRC_p2      , 83 ;
constant T_IP_DST_p1      , 84 ; IP-адресполучателя
constant T_IP_DST_p2      , 85 ;
constant T_ESP_SPI_p1     , 86 ; Индекс SPI сеанса
constant T_ESP_SPI_p2     , 87 ;
constant T_ESP_SN_p1      , 88 ;
constant T_ESP_SN_p2      , 89 ;
constant T_MTU             , 8A ; MTU канала
;-----
; Точка входа в программу
start :                loads0, 00                ;
; Установка констант
Set_masks :           callc, FF                ;
load      sF, FF                ;
                                call      c, 08                ;
                                load      sE, 00                ;
                                call      c, 01                ;
                                load      sD, 00                ;
                                call      c, ff                ;
                                load      sC, 00                ;
                                call      c, 00                ;
                                load      sB, ff                ;
                                call      c, 45                ;
                                load      sA, 00                ;
                                call      c, 12                ;
                                load      s9, 34                ;
                                call      c, 03                ;
                                loads8, FF                ;
; Цикл ожидания буфера на обработку
Wait_buf_for_proc : input  s2, BUF_CNTRL                ;
                                and      s2, BUF_PROC_REQ        ;
                                jump      z, Wait_buf_for_proc    ;
; Чтение данных из буфера
callSet_buffer          ;
callRead_DDR            ;
; Проверка поля тип протокола в заголовке Ethernet
Check_eth_protocol : input  s3, ETH_PROTOCOL                ;
                                xor      s3, sE                ;
                                jump      nz, Push_buf_to_cpu    ;
; Проверка версии IP
Check_ip_version :      input  s3, IP_VERSION                ;

```

```

        and      s3, sC          ;
        xor      s3, sA          ;
        jump     nz, Push_buf_to_del ;
; Проверка времени жизни пакета
Check_ttl :      inputs3, IP_TTL          ;
sub      s3, sD          ;
        load     s7, s3          ;
        and      s3, sC          ;
        jump     z, Push_buf_to_del ;
; Проверка размера IP пакета
Check_ip_length : input  s3, IP_TOTAL_LEN          ;
        input    s4, DMA_FRAME_SIZE          ;
        sub      s4, ETH_HDR_SIZE          ;
; Если размер IP больше принятого пакета, пакет является поврежденным
sub      s4, s3          ;
        jump     c, Push_buf_to_del ;
; Корректировка размера данных в заголовке DMA
Data_resize :      input  s3, IP_TOTAL_LEN          ;
        add      s3, ETH_HDR_SIZE          ;
        output   s3, DMA_FRAME_SIZE          ;
; Проверка контрольной суммы заголовка IP
Check_checksum :   call    Compute_IP_csum          ;
        input    s3, CSUM_DOUT          ;
        or       s3, s0          ;
        jump     nz, Push_buf_to_del ;
; Проверка поля IP заголовка тип протокола на соответствие ESP
Check_ip_protocol : input  s3, IP_PROTOCOL          ;
        call     c, 00          ;
        and      s3, ff          ;
        xor      s3, 32          ;
        jump     nz, Push_buf_to_cpu ;
; Проверка кратности содержимого ESP 16 байтам
Check_rate :       input  s3, IP_TOTAL_LEN          ;
        sub      s3, IP_HDR_SIZE          ;
        sub      s3, ESP_HDR_SIZE          ;
        and      s3, 0F          ;
        jump     nz, Push_buf_to_del ;
; Преобразование SPI в указатель на запись таблицы
Convert_spi_to_ptr : input  s3, ESP_SPI_p2          ;
        and      s3, s8          ;
        load     s5, 06          ;
Left_cycle_roll :  rl      s3          ;

```

```

        sub      s5, 01                ;
        jump     nz, Left_cycle_roll  ;
        load     s4, s3                ;
        ands4, 3F                      ;
adds4, 20                               ;
; Подготовка операции чтения из памяти
output  s4, DMA_HADDR                ;
        call     c, FF                ;
        and      s3, C0                ;
        output   s3, DMA_LADDR         ;
        load     s3, 80                ;
        output   s3, DMA_BADDR         ;
        load     s3, 40                ;
        output   s3, DMA_SIZE         ;
        callRead_DDR                  ;
; Проверка активности записи таблицы
Check_Active :      input   s3, T_ACTIVE        ;
                  call     c , 00            ;
                  xor      s3, 01            ;
                  jump     nz, Push_buf_to_cpu ;
; Проверка полей пакета по таблице
Check_table_record : load     s1, T_IP_SRC_p1    ;
                  load     s2, IP_SRC_p1        ;
                  load     s5, 06              ;
fields_cmp  :      input   s3, (s1)            ;
                  input   s4, (s2)            ;
                  xor      s3, s4              ;
                  jump     nz, Push_buf_to_del  ;
                  add      s1, 01              ;
                  add      s2, 01              ;
                  sub      s5, 01              ;
                  jumpnzs, fields_cmp          ;
; Проверка размера пакета по MTU канала
Check_mtu   :      input   s3, IP_TOTAL_LEN     ;
                  input   s4, T_MTU            ;
                  sub      s4, s3              ;
                  jump     c, Push_buf_to_cpu   ;

; Формирование заголовка шифратора
; Резервирование поля под маску
Insert_ZAS_header : output   s0, ZAS_MASK_p1    ;
                  output   s0, ZAS_MASK_p2      ;

```

```

; Копирование порядкового номера пакета из заголовка буфера
input      s3, DMA_SN_p2          ;
                                output s3, ZAS_PKT_ID_p2      ;
                                input  s3, DMA_SN_p1          ;
                                output s3, ZAS_PKT_ID_p1      ;

; Копирование номера ключа из таблицы
inputs3, T_ID                    ;
output    s3, ZAS_KEY_ID          ;

; Установка флагов управления в 0
outputs0, ZAS_CONTROL            ;

; Обработка полей заголовка буфера
; Изменение размера данных
Frame_size_update : input  s3, DMA_FRAME_SIZE      ;
                    add     s3, FR_SIZE_CHANGE      ;
                    output  s3, DMA_FRAME_SIZE      ;

; Изменение смещения данных
Frame_offset_update : input  s3, DMA_DATA_OFFSET    ;
                    sub     s3, FR_SIZE_CHANGE      ;
                    output  s3, DMA_DATA_OFFSET      ;

; Запись данных буфера в память
callSet_buffer          ;
callWrite_DDR           ;

; Передача буфера в очередь на отправку
Push_buf_to_tx :      load    s2, BUF_TO_TX          ;
                    output   s2, BUF_CNTRL          ;
                    jump     Wait_buf_for_proc      ;

; Передача буфера в очередь на обработку ЦПУ
Push_buf_to_cpu :      load    s2, BUF_TO_CPU        ;
                    output   s2, BUF_CNTRL          ;
                    jump     Wait_buf_for_proc      ;

; Передача буфера в очередь на очистку
Push_buf_to_cpu :      load    s2, BUF_TO_DEL        ;
                    output   s2, BUF_CNTRL          ;
                    jump     Wait_buf_for_proc      ;

; -----
; Функции
; Установка параметров для операции с буфером
Set_buffer :          input    s3, BUF_HADDR        ;
                    output    s3, DMA_HADDR         ;
                    input     s3, BUF_LADDR         ;
                    output    s3, DMA_LADDR         ;
                    load      s3, s0                ;

```

```

                                output    s3, DMA_BADDR          ;
                                load      s3, BUF_SIZE          ;
                                output    s3, DMA_SIZE          ;
                                return                                          ;

; Чтение из памяти DDR
Read_DDR :                     load      s3, DMA_READ          ;
                                output    s3, DMA_CNTRL         ;
                                jump      Wait_DMA_complete     ;

; Запись в память DDR
Write_DDR :                     load      s3, DMA_WRITE         ;
                                output    s3, DMA_CNTRL         ;
Wait_DMA_complete :            input     s3, DMA_CNTRL          ;
                                and        s3, DMA_COMPLETE      ;
                                jump      z, Wait_DMA_complete   ;
                                return                                          ;

; Расчет контрольной суммы заголовка IP
Compute_IP_csum :              output    s0, CSUM_NEXT          ;
                                load      s2, IP_VERSION         ;
                                load      s5, 0A                 ;
Next_csum_data :               input     s3, (s2)               ;
                                output    s3, CSUM_DIN           ;
                                add        s2, 01                 ;
                                sub        s5, 01                 ;
                                jump      nz, Next_csum_data      ;
                                return                                          ;

```

Программа обработки исходящего трафика для сети зашифрованной информации

```
; Регистры контроллера DMA
constantDMA_CNTRL , F0 ; Адрес регистра управления DMA
constantDMA_HADDR , F1 ; Адрес регистра старшей части адреса памяти DDR
constantDMA_LADDR , F2 ; Адрес регистра младшей части адреса памяти DDR
constantDMA_BADDR , F3 ; Адрес регистра адреса памяти BRAM
constantDMA_SIZE , F4 ; Адрес регистра размера данных
; Регистры контроллера буферов
constantBUF_CNTRL , F5 ; Адрес регистра управления контроллера буферов
constantBUF_HADDR , F6 ; Адрес регистра старшей части адреса буфера
constantBUF_LADDR , F7 ; Адрес регистра младшей части адреса буфера
; Регистры генератора псевдослучайной последовательности
constantRND_DIN , FB ; Адрес регистра записи данных в генератор
constantRND_DOUT , FC ; Адрес регистра чтения данных из генератора
; Регистры модуля расчета контрольной суммы IP
constantCSUM_NEXT , FD ; Адрес регистра управления модулем
constantCSUM_DIN , FE ; Адрес регистра записи данных в модуль
constantCSUM_DOUT , FF ; Адрес регистра чтения данных из модуля
;-----
; Флаги статуса регистра управления контроллера буферов
constantBUF_PROC_REQ , 01 ; Флаг получения буфера на обработку
constantBUF_PROC_COMP , 02 ; Флаг завершения обработки буфера
; Флаги контроля регистра управления контроллера буферов
constantBUF_TO_DEL , 02 ; Флаг отправки буфера в очередь на очистку
constantBUF_TO_CPU , 04 ; Флаг отправки буфера в очередь обработки ЦПУ
constantBUF_TO_TX , 08 ; Флаг отправки буфера в очередь на передачу
; Флаги регистра управления контроллера DMA
constantDMA_BUSY , 01 ; Флаг выполнения операции
constantDMA_COMPLETE , 02 ; Флаг завершения операции
constantDMA_READ , 04 ; Флаг операции чтения из памяти DDR
constantDMA_WRITE , 08 ; Флаг операции записи в память DDR
;-----
; Поля заголовка DMA
constantDMA_SRC , 00 ; Поле идентификатора контроллера DMA
constantDMA_DATA_OFFSET , 00 ; Поле смещения данных в буфере
constantDMA_FRAME_SIZE , 01 ; Поле размера данных в буфере
constantDMA_SN_p1 , 02 ; Поле порядкового номера полученного пакета
constant DMA_SN_p2 , 03 ;
; Поля заголовка Ethernet
constantMAC_DST_p1 , 0D ; Поле адреса получателя
constant MAC_DST_p2 , 0E ;
```

```

constant MAC_DST_p3      , 0F ;
constant MAC_SRC_p1      , 10 ; Поле адреса отправителя
constant MAC_SRC_p2      , 11 ;
constant MAC_SRC_p3      , 12 ;
constant ETH_PROTOCOL     , 13 ; Полетиппротокола
; Полязаголовок IP
constant IP_VERSION       , 14 ; Полеверсии IP
constant IP_IHL           , 14 ; ПолеразмеразаголовокIP
constant IP_TOS           , 14 ; Поле «Типсервиса»
constant IP_TOTAL_LEN     , 15 ; Полеразмерапакета IP
constant IP_ID            , 16 ; Поле идентификатора IP пакета
constant IP_FRAG          , 17 ; Поле фрагментации IP пакета
constant IP_PROTOCOL      , 18 ; Поле протокола
constant IP_TTL           , 18 ; Поле время жизни пакета
constant IP_CHECKSUM      , 19 ; Поле контрольной суммы заголовка
constant IP_SRC_p1        , 1A ; Поле адреса отправителя
constant IP_SRC_p2        , 1B ;
constant IP_DST_p1        , 1C ; Поле адреса получателя
constant IP_DST_p2        , 1D ;
; Полязаголовок ESP
constant ESP_SPI_p1       , 1E ; Поле SPI
constant ESP_SPI_p2       , 1F ;
constant ESP_SN_p1        , 20 ; Поле порядкового номера
constant ESP_SN_p2        , 21 ;
; Поля заголовка шифратора
constant ZAS_PKT_ID_p1    , 18 ; Поле идентификатора пакета
constant ZAS_PKT_ID_p2    , 19 ;
constant ZAS_MASK_p1      , 1A ; Поле маски пакета
constant ZAS_MASK_p2      , 1B ;
constant ZAS_KEY_ID       , 1C ; Поле номера ключа
constant ZAS_CONTROL      , 1D ; Поле флагов управления
; -----
; Размеры структур данных
constant ZAS_HDR_SIZE     , 0C ; Размер заголовка шифратора
constant ETH_HDR_SIZE     , 0E ; Размер заголовка Ethernet
constant IP_HDR_SIZE      , 14 ; Размер заголовка IP
constant ESP_HDR_SIZE     , 1C ; Размер заголовка ESP
constant FR_SIZE_CHANGE   , 16 ; Изменение размера данных при обработке
; Поля записи таблицы маршрутов
constant T_ACTIVE         , 80 ; Флаг активности записи таблицы
constant T_ID             , 81 ; Номер ключа шифрования
constant T_ESP_SPI_p1     , 82 ; Индекс SPI сеанса
constant T_ESP_SPI_p2     , 83 ;

```

```

constant T_ESP_SN_p1      , 84 ; Порядковый номер пакета ESP
constant T_ESP_SN_p2      , 85 ;
constant T_MAC_DST_p1     , 86 ; MAC-адрес получателя
constant T_MAC_DST_p2     , 87 ;
constant T_MAC_DST_p3     , 88 ;
constant T_MAC_SRC_p1     , 89 ; MAC-адрес отправителя
constant T_MAC_SRC_p2     , 8A ;
constant T_MAC_SRC_p3     , 8B ;
constant T_IP_SRC_p1      , 8C ; IP-адрес отправителя
constant T_IP_SRC_p2      , 8D ;
constant T_IP_DST_p1      , 8E ; IP-адрес получателя
constant T_IP_DST_p2      , 8F ;
constant T_BLOCKED        , 91 ; Флаг блокировки записи таблицы ЦПУ
;-----
; Точка входа в программу
start :                loads0, 00 ;
; Установка констант
Set_masks :           callc, FF ;
load      sF, FF ;

                        call      c, 08 ;
                        load      sE, 00 ;
                        call      c, 01 ;
                        load      sD, 00 ;
                        call      c, ff ;
                        load      sC, 00 ;
                        call      c, 00 ;
                        load      sB, ff ;
                        call      c, 45 ;
                        load      sA, 00 ;
                        call      c, 12 ;
                        load      s9, 34 ;
                        call      c, 03 ;
                        load      s8, FF ;

; Цикл ожидания буфера на обработку
Wait_buf_for_proc : input s2, BUF_CNTRL ;
                        and      s2, BUF_PROC_REQ ;
                        jump      z, Wait_buf_for_proc ;

; Чтение данных из буфера
callSet_buffer ;
callRead_DDR ;

; Проверка статуса шифрования
Check_status : input s3, ZAS_CONTROL ;
               or      s3, 00 ;

```

```

                                jump      nz, Push_buf_to_cpu      ;
; Проверка кратности содержимого ESP 16 байтам
Check_rate :      input      s3, DMA_FRAME_SIZE      ;
                                sub       s3, ZAS_HDR_SIZE      ;
                                sub       s3, ESP_HDR_SIZE      ;
                                and       s3, 0F      ;
                                jump      nz, Push_buf_to_cpu      ;
; Преобразование SPI в указатель на запись таблицы
Convert_spi_to_ptr : input      s3, ESP_SPI_p2      ;
                                and       s3, s8      ;
                                load      s5, 06      ;
Left_cycle_roll  :  rl       s3      ;
                                sub       s5, 01      ;
                                jump      nz, Left_cycle_roll      ;
                                load      s4, s3      ;
                                ands4, 3F      ;
adds4, 20      ;
; Подготовка операции чтения из памяти
output      s4, DMA_HADDR      ;
                                call      c, FF      ;
                                and       s3, C0      ;
                                output     s3, DMA_LADDR      ;
                                load      s3, 80      ;
                                output     s3, DMA_BADDR      ;
                                load      s3, 40      ;
                                output     s3, DMA_SIZE      ;
                                callRead_DDR      ;
; Проверка активности записи таблицы
Check_Active :      input      s3, T_ACTIVE      ;
                                call      c , 00      ;
                                xor       s3, 01      ;
                                jump      nz, Push_buf_to_cpu      ;
; Проверка номера ключа шифрования по таблице
Check_ID :      input      s3, T_ID      ;
                                input     s4, ZAS_KEY_ID      ;
                                xor       s4, s3      ;
                                jump      nz, Push_buf_to_cpu      ;
; Проверка индекса SPI по таблице
Check_SPI :      input      s3, T_ESP_SPI_p1      ;
                                input     s4, ESP_SPI_p1      ;
                                xor       s4, s3      ;
                                jump      nz, Push_buf_to_cpu      ;
                                input     s3, T_ESP_SPI_p2      ;

```

```

        input    s4, ESP_SPI_p2        ;
        xor      s4, s3                ;
        jump     nz, Push_buf_to_cpu    ;
; Цикл проверки флага блокировки (ожидание освобождения записи ЦПУ)
Check_blocked :   input    s3, T_BLOCKED        ;
                  or       s3, 00                ;
                  jump     z, Update_ESP_SN      ;
                  sub      s5, 01                ;
                  jump     z, Push_buf_to_cpu    ;
                  call     Read_DDR              ;
                  jump     Check_blocked         ;
; Запись порядкового номера пакета ESP
Update_ESP_SN :   input    s3, T_ESP_SN_p1      ;
                  output   s3, ESP_SN_p1        ;
                  input    s3, T_ESP_SN_p2      ;
                  output   s3, ESP_SN_p2        ;
                  add      s3, 01                ;
                  output   s3, T_ESP_SN_p2      ;
                  input    s3, T_ESP_SN_p1      ;
                  addcy    s3, 00                ;
                  output   s3, T_ESP_SN_p1      ;
                  call     Write_DDR             ;
; Формирование заголовка IP
; Запись версии IP
Insert_IP_header : load     s3, sA                ;
                  outputs3, IP_VERSION          ;
; Запись идентификатора пакета
add      s1, 01                ;
                  call     c , 80                ;
                  or      s1, 00                ;
                  outputs1, IP_ID               ;
; Запись фрагментации
outputs0, IP_FRAG            ;
; Запись времени жизни пакета и тип протокола
call     c, 80                ;
                  load     s3, 32                ;
                  output   s3, IP_TTL           ;
; Копирование адресов из таблицы
inputs3, T_IP_SRC_p1        ;
output   s3, IP_SRC_p1      ;
                  input    s3, T_IP_SRC_p2      ;
                  output   s3, IP_SRC_p2        ;
                  input    s3, T_IP_DST_p1      ;

```

```

        output    s3, IP_DST_p1          ;
        input     s3, T_IP_DST_p2        ;
        output    s3, IP_DST_p2          ;
; Формирование размера пакета
Compute_IP_total_len : input    s3, DMA_FRAME_SIZE      ;
                        sub      s3, ZAS_HDR_SIZE        ;
                        add      s3, IP_HDR_SIZE         ;
                        output    s3, IP_TOTAL_LEN       ;
Compute_IP_checksum : output    s0, IP_CHECKSUM         ;
                        call      Compute_IP_csum        ;
                        input     s3, CSUM_DOUT          ;
                        outputs3, IP_CHECKSUM           ;
; Формирование заголовка Ethernet по таблице
Insert_eth_header  : input     s3, T_MAC_DST_p1         ;
                        output    s3, MAC_DST_p1        ;
                        input     s3, T_MAC_DST_p2      ;
                        output    s3, MAC_DST_p2        ;
                        input     s3, T_MAC_DST_p3      ;
                        output    s3, MAC_DST_p3        ;
                        input     s3, T_MAC_SRC_p1      ;
                        output    s3, MAC_SRC_p1        ;
                        input     s3, T_MAC_SRC_p2      ;
                        output    s3, MAC_SRC_p2        ;
                        input     s3, T_MAC_SRC_p3      ;
                        output    s3, MAC_SRC_p3        ;
                        outputsE, ETH_PROTOCOL          ;
; Обработка полей заголовка буфера
; Изменение размера данных буфера
Frame_size_update  : input     s3, IP_TOTAL_LEN        ;
                        add      s3, ETH_HDR_SIZE       ;
; Проверка размера пакета на минимальный размер (60 байт)
Check_little_frame : load      s4, s3                  ;
                        sub      s4, 3c                 ;
                        jump      nc, Write_size         ;
; Установка размера данных для коротких пакетов
callc, 00          ;
loads3, 3c         ;
; Запись размера данных в заголовок
Write_size : outputs3, DMA_FRAME_SIZE      ;
; Изменение смещения данных в буфере
Frame_offset_update : inputs3, DMA_DATA_OFFSET      ;
add      s3, FR_SIZE_CHANGE              ;
                        output    s3, DMA_DATA_OFFSET  ;

```

```

; Запись данных в буфер
callSet_buffer          ;
callWrite_DDR           ;
; Передача буфера в очередь на отправку
Push_buf_to_tx :        load      s2, BUF_TO_TX          ;
                        output    s2, BUF_CNTRL          ;
                        jump      Wait_buf_for_proc      ;
; Передача буфера в очередь на обработку ЦПУ
Push_buf_to_cpu :        load      s2, BUF_TO_CPU        ;
                        output    s2, BUF_CNTRL          ;
                        jump      Wait_buf_for_proc      ;
; Передача буфера в очередь на очистку
Push_buf_to_cpu :        load      s2, BUF_TO_DEL        ;
                        output    s2, BUF_CNTRL          ;
                        jump      Wait_buf_for_proc      ;

;-----
; Функции
; Установка параметров для операции с буфером
Set_buffer :             input     s3, BUF_HADDR         ;
                        output    s3, DMA_HADDR         ;
                        input     s3, BUF_LADDR         ;
                        output    s3, DMA_LADDR         ;
                        load      s3, s0                 ;
                        output    s3, DMA_BADDR         ;
                        load      s3, BUF_SIZE          ;
                        output    s3, DMA_SIZE          ;
                        return                                          ;
; Чтениеизпамяти DDR
Read_DDR :               load      s3, DMA_READ         ;
                        output    s3, DMA_CNTRL         ;
                        jump      Wait_DMA_complete     ;
; Записыввпамять DDR
Write_DDR :               load      s3, DMA_WRITE        ;
                        output    s3, DMA_CNTRL         ;
Wait_DMA_complete :      input     s3, DMA_CNTRL         ;
                        and       s3, DMA_COMPLETE      ;
                        jump      z, Wait_DMA_complete  ;
                        return                                          ;
; Расчет контрольной суммы заголовка IP
Compute_IP_csum :        output    s0, CSUM_NEXT        ;
                        load      s2, IP_VERSION        ;
                        load      s5, 0A                ;
Next_csum_data :         input     s3, (s2)             ;

```

```
output    s3, CSUM_DIN      ;
add       s2, 01            ;
sub       s5, 01            ;
jump     nz, Next_csum_data ;
return                                         ;
```

ПРИЛОЖЕНИЕ Ж
(обязательное)
РЕЗУЛЬТАТЫ ПРОВЕРКИ ВКР НА ОРИГИНАЛЬНОСТЬ

Протокол

проверки на оригинальность в системе «Антиплагиат.ВУЗ» выпускной квалификационной работы на тему «Программно-аппаратный модуль фильтрации, маршрутизации и подготовки к шифрованию сетевого трафика устройства IP-шифрования» студента 6 курса В.А. Цыгановой специальности 10.05.02 «Информационная безопасность телекоммуникационных систем».

Руководитель ВКР Сергей Владимирович Огурцов

Информация о документе:

Имя исходного файла: Диплом_Цыганова.doc
Имя компании: Пензенский гос. Университет
Тип документа: Прочее
Имя документа: Группа 11ПТ1 ВКР Цыганова ВА
Дата проверки: 30.12.2016 21:50
Модули поиска: Интернет (Антиплагиат), Модуль поиска ЭБС "Лань", Диссертации и авторефераты РГБ, Модуль поиска ЭБС БиблиоРоссики, Университетская библиотека онлайн, Кольцо вузов, Пензенский гос. Университет

Текстовые статистики:

Индекс читаемости: сложный
Неизвестные слова: в пределах нормы
Макс. длина слова: в пределах нормы
Большие слова: выше нормы!

Оригинальные блоки: 76,97%

Заимствованные блоки: 23,03%

Заимствование из "белых" источников: 0%

Итоговая оценка оригинальности: **76,97%**

30.12.2016 _____ С.В. Огурцов