

Санкт-Петербургский политехнический университет Петра Великого
Институт информационных технологий и управления
Кафедра информационных и управляющих систем

Отчет по НИР

***Исследование возможностей технологии
ПЛИС и СБИС SoC FPGA Cyclone V для
создания специализированных сетевых
процессоров***

Выполнил студент гр. 63501/2 _____ Мурашко Д. С.

Научный руководитель _____ Антонов А. П.

Санкт-Петербург
2017

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	6
1. ЦЕЛЬ И ЗАДАЧИ РАБОТЫ	8
1.1 Решаемые задачи.....	8
1.2 Требования к устройству фильтрации.....	8
2. АНАЛИЗ И ФОРМАЛИЗАЦИЯ ПОСТАВЛЕННЫХ ЗАДАЧ...10	
2.1 Анализ подходов к построению межсетевых экранов	10
2.1.1 Анализ подходов к построению межсетевых экранов.....	10
2.1.2 Типы межсетевых экранов	11
2.1.2.1 Фильтрация на сетевом уровне	12
2.1.2.2 Фильтрация на сеансовом уровне	13
2.1.2.3 Фильтрация на прикладном уровне	14
2.1.3 Сравнение аппаратных и программных реализаций межсетевых экранов.....	15
2.2 Анализ элементной базы для аппаратной реализации межсетевого экрана	17
2.3 Анализ средств проектирования	21
2.4 Выводы.....	21
3. РАЗРАБОТКА АРХИТЕКТУРЫ СИСТЕМЫ ФИЛЬТРАЦИИ ПАКЕТОВ И ВЫБОР ПЛАТЫ-ПРОТОТИПА	22
3.1 Требования к системе фильтрации пакетов и плате-прототипу.....	22
3.2 Архитектура системы фильтрации пакетов	22
3.3 Выбранная плата-прототип.....	23
3.4 Выводы.....	26
4 РАЗРАБОТКА СТРУКТУРЫ АППАРАТНО РЕАЛИЗУЕМОГО МЕЖСЕТЕВОГО ЭКРАНА.....	27
4.1 Общая структура межсетевого экрана	27
4.2 Разработка структуры блока проверки правил и выработки решения	28
4.2.1 Разработка структуры модуля решения DL (Decision Line).....	29
4.3 Разработка формата хранения правил фильтрации	30
4.3.1 Разработка формата хранения правила	30
4.3.2 Разработка формата записи поля в правиле.....	32
4.4 Анализ вариантов аппаратной реализации межсетевого экрана на СБИС ПЛ	33
4.5 Выводы.....	35

5	РАЗРАБОТКА ОПИСАНИЯ МОДУЛЕЙ И УСТРОЙСТВА ФИЛЬТРАЦИИ ПАКЕТОВ НА ЯЗЫКЕ VERILOG.....	36
5.1	Разработка блока взаимодействия с внешней Ethernet (приемная часть).....	36
5.1.1	Временная диаграмма приема пакетов.....	36
5.1.2	Описание блока на языке Verilog	36
5.2	Разработка блока взаимодействия с внешней Ethernet (передающая часть).....	39
5.2.1	Временная диаграмма передачи пакетов	40
5.2.2	Описание блока на языке Verilog	40
5.3	Разработка блока анализа Ethernet пакетов	41
5.3.1	Разработка алгоритма поиска параметров пакета	41
5.3.2	Описание блока на языке Verilog	43
5.4	Разработка блока хранения правил – SoDR (Set of Defined Rules)	45
5.4.1	Расчет параметров блока	45
5.4.2	Реализация блока.....	46
5.5	Разработка блока проверки правил и выработки решения RCE (Rule Checking Engine).....	47
5.5.1	Разработка блока решения DL (Decision Line)	48
5.5.2	Разработка модуля DM (Decision Make), вырабатывающего общее решение по всем правилам	49
5.6	Разработка блока буферизации сетевых пакетов.....	50
5.7	Разработка процедуры и программы задания правил.....	51
5.7.1	Алгоритм работы программы	52
5.7.2	Формат задания правил и описание программы	53
5.7.3	Формат задания правил и описание программы	55
5.8	Выводы.....	56
6	РАЗРАБОТКА ТЕСТОВ И МОДЕЛИРОВАНИЕ	57
6.1	Моделирование блока взаимодействия с внешней Ethernet (приемная часть).....	57
6.1.1	Описание теста	57
6.1.2	Результаты моделирования	57
6.2	Моделирование блока взаимодействия с внешней Ethernet (передающая часть).....	58
6.2.1	Описание теста	58
6.2.2	Результаты моделирования	59
6.3	Моделирование блока анализа Ethernet пакетов	59
6.3.1.	Описание теста	59
6.3.2.	Результаты моделирования	60
6.4	Моделирование блока решения DL (Decision Line).....	61

6.4.1	Описание теста	61
6.4.2	Результаты моделирования	61
6.5	Моделирование модуля DM (Decision Make), вырабатывающего общее решение по всем правилам	63
6.5.1.	Описание теста	63
6.5.2.	Результаты моделирования	63
6.6	Выводы.....	64
7	СИНТЕЗ И РЕАЛИЗАЦИЯ НА СБИС ПЛ.....	65
7.1	Настройки системы синтеза и трассировки	65
7.2	Анализ результатов синтеза и трассировки	65
7.3	Выводы.....	66
8	ТЕСТИРОВАНИЕ НА ПЛАТЕ-ПРОТОТИПЕ.....	67
8.1	Разработка процедуры тестирования	67
8.2	Настройка системного окружения и контроль трафика	68
8.2.1	Настройка отладочного комплекса.....	68
8.2.2	Контроль передачи сгенерированного сетевого пакета.....	68
8.3	Тестирование.....	70
8.4	Выводы.....	72
9	АНАЛИЗ БЫСТРОДЕЙСТВИЯ И ПРОИЗВОДИТЕЛЬНОСТИ.....	73
	ЗАКЛЮЧЕНИЕ	78
	СПИСОК ИСПОЛЪЗУЕМЫХ ИСТОЧНИКОВ.....	79
	ПРИЛОЖЕНИЕ А ИЛЛЮСТРАЦИИ	80
	ПА1. Структура компонентов проекта и аппаратные затраты на каждый блок	80
	ПА2. RTL Viewer проекта	81
	ПРИЛОЖЕНИЕ Б ТЕКСТЫ ПРОГРАММ.....	82
	ПБ.1 Текст программы блока взаимодействия с внешней Ethernet (приемная часть).....	82
	ПБ.2 Текст программы теста блока взаимодействия с внешней Ethernet (приемная часть).....	85
	ПБ.3 Текст программы модуля PPar	87
	ПБ.4 Текст программы теста модуля PPar	90
	ПБ.5 Текст программы модуля DL.....	93
	ПБ.6 Текст программы теста модуля DL	102
	ПБ.7 Текст программы модуля DM	104
	ПБ.8 Текст программы задания правил.....	105

СПИСОК ОБОЗНАЧЕНИЙ И СОКРАЩЕНИЙ

СБИС	Сверхбольшая интегральная схема
FPGA	Field-Programmable Gate Array, программируемая пользователем вентильная матрица
FTP	File Transfer Protocol, протокол передачи файлов
GMII	Gigabit Media Independent Interface, независимый от среды передачи гигабитный интерфейс
HTTP	HyperText Transfer Protocol, протокол передачи гипертекста
IP	Internet Protocol, интернет протокол
NAT	Network Address Translation, преобразование сетевых адресов
OSI	Open systems interconnection basic reference model, базовая эталонная модель взаимодействия открытых систем
RAM	Random Access Memory, память с произвольной выборкой
ROM	Read-only memory, постоянное запоминающее устройство
TCP	Transmission Control Protocol, протокол управления передачей

ВВЕДЕНИЕ

Актуальность проблемы защиты информации сегодня не вызывает сомнений. Успех современной компании и ее развитие в условиях острой конкуренции в значительной степени зависят от применения информационных технологий, а, следовательно, от степени обеспечения информационной безопасности.

Интенсивное развитие глобальных компьютерных сетей, появление новых технологий поиска информации привлекают все большее внимание к сети Интернет со стороны частных лиц и различных организаций. Многие организации принимают решения по интеграции своих локальных и корпоративных сетей в Интернет. Использование Интернета в коммерческих целях, а также при передаче информации, содержащей сведения конфиденциального характера, влечет за собой необходимость построения эффективной системы защиты данных. Использование глобальной сети Интернет обладает неоспоримыми достоинствами, но, как и многие другие новые технологии, имеет и свои недостатки. Развитие глобальных сетей привело к многократному увеличению количества не только пользователей, но и атак на компьютеры, подключенные к Интернету. Ежегодные потери из-за недостаточного уровня защищенности компьютеров оцениваются десятками миллионов долларов. Поэтому при подключении к Интернету локальной или корпоративной сети необходимо позаботиться об обеспечении ее информационной безопасности.

Глобальная сеть Интернет создавалась как открытая система, предназначенная для свободного обмена информацией. В силу открытости своей идеологии Интернет предоставляет злоумышленникам значительно большие возможности по проникновению в информационные системы. Через Интернет нарушитель может:

- вторгнуться во внутреннюю сеть предприятия и получить несанкционированный доступ к конфиденциальной информации;
- незаконно скопировать важную и ценную для предприятия информацию;
- получить пароли, адреса серверов, а подчас и их содержимое;
- войти в информационную систему предприятия под именем зарегистрированного пользователя и т.д.

Посредством получения злоумышленником информации может быть серьезно подорвана конкурентоспособность предприятия и доверие его клиентов [1].

Ряд задач по отражению наиболее вероятных угроз для внутренних сетей способны решать межсетевые экраны. Межсетевой экран (другие названия файрволл (от англ. Firewall), брандмауэр (от нем. Brandmauer)) — программный или программно-аппаратный элемент компьютерной сети, осуществляющий контроль и фильтрацию проходящего через него сетевого трафика в соответствии с заданными правилами [2].

Пропускная способность сетей все увеличивается. Программные межсетевые экраны не справляются с такими скоростями. Для решения данной проблемы необходимо высокопроизводительное аппаратное обеспечение. Реализация на СБИС программируемой логики — новый перспективный подход, который общего решения не имеет.

В ходе данной работы будут проанализированы подходы к задаче фильтрации сетевых пакетов, варианты реализации и создание собственного аппаратно-реализуемого устройства фильтрации, а также его моделирование и тестирование.

1. ЦЕЛЬ И ЗАДАЧИ РАБОТЫ

Цель работы – исследование возможностей СБИС программируемой логики для аппаратной реализации межсетевого экрана, разработка архитектуры и структуры аппаратной реализации, создание собственного аппаратно-реализуемого устройства фильтрации, а также его моделирование и тестирование.

1.1 Решаемые задачи

Для достижения цели требуется решить следующие задачи:

- Провести анализ подходов к решению задачи фильтрации сетевых пакетов, элементной базы для аппаратной реализации межсетевых фильтров, средств проектирования.
- Разработать архитектуру системы фильтрации пакетов и выбрать плату-прототип для аппаратного тестирования создаваемого межсетевого экрана.
- Разработать структуру аппаратно-реализуемого межсетевого экрана.
- Разработать формата хранения правил фильтрации.
- Анализ вариантов аппаратной реализации межсетевого экрана на СБИС программируемой логики при различных требованиях к быстродействию и ограничениях к аппаратным затратам.
- Разработать описание модулей и устройства фильтрации пакетов на языке Verilog.
- Осуществить разработку тестов для модулей проекта и провести моделирование и отладку проекта.
- Разработать формат задания правил фильтрации, процедуру и программные средства их подготовки для записи в память СБИС ПЛ.
- Осуществить синтез и реализацию проекта системы фильтрации сетевых пакетов на базе СБИС ПЛ, провести анализ быстродействия.
- Разработать процедуру проведения тестирования и осуществить отладку проекта на плате-прототипе.

1.2 Требования к устройству фильтрации

Требования к изделию, описанные в техническом задании ССПЛ ТЗ 18.11.16[3]:

- Изделие должно обеспечивать защиту внутренней сети (проводная Ethernet 10/100/1000, стек протоколов TCP/IPv4) от

несанкционированного доступа (НСД) в соответствии с правилами, устанавливаемыми администратором.

- Изделие должно обеспечивать возможность хранения правил во внутренней памяти. Максимальное число правил – 256.
- Изделие должно проверять каждый сетевой пакет, поступающий на любой фильтрующий интерфейс изделия, на соответствие правилам фильтрации.
- Требования к параметрам пакета, использующих при фильтрации, требования к формату правил и т.д.

2. АНАЛИЗ И ФОРМАЛИЗАЦИЯ ПОСТАВЛЕННЫХ ЗАДАЧ

Для решения поставленных задач необходимо провести анализ:

- подходов к решению задачи фильтрации сетевых пакетов,
- элементной базы для аппаратной реализации межсетевых фильтров,
- средств проектирования.

2.1 Анализ подходов к построению межсетевых экранов

В данном разделе рассматриваются принцип работы, типы межсетевых экранов, сравнение аппаратных и программных подходов к реализации межсетевых экранов, анализ элементной базы – СБИС Программируемой Логике (СБИС ПЛ), для их аппаратной реализации.

2.1.1 Анализ подходов к построению межсетевых экранов

Для того чтобы эффективно обеспечивать безопасность сети, межсетевой экран отслеживает и управляет всем потоком данных, проходящим через него. Для принятия управляющих решений для ТСП/ІР-сервисов (то есть передавать, блокировать или отмечать в журнале попытки установления соединений) межсетевой экран должен получать, запоминать, выбирать и обрабатывать информацию, полученную от всех коммуникационных уровней и от других приложений.

Межсетевой экран пропускает через себя весь трафик, принимая относительно каждого проходящего пакета решение: дать ему возможность пройти или нет. Для того чтобы межсетевой экран мог осуществить эту операцию, ему необходимо определить набор правил фильтрации. Решение о том, фильтровать ли с помощью межсетевого экрана пакеты данных, связанные с конкретными протоколами и адресами, зависит от принятой в защищаемой сети политики безопасности. По сути, межсетевой экран представляет собой набор компонентов, настраиваемых для реализации выбранной политики безопасности. Политика сетевой безопасности каждой организации должна включать (кроме всего прочего) две составляющие: политика доступа к сетевым сервисам и политика реализации межсетевых экранов.

Однако недостаточно просто проверять пакеты по отдельности. Информация о состоянии соединения, полученная из инспекции соединений в прошлом и других приложений – главный фактор в принятии управляющего решения при попытке установления нового

соединения. Для принятия решения могут учитываться как состояние соединения (полученное из прошлого потока данных), так и состояние приложения (полученное из других приложений).

Таким образом, управляющие решения требуют, чтобы межсетевой экран имел доступ, возможность анализа и использования следующих факторов:

- информации о соединениях – информация от всех семи уровней (модели OSI) в пакете;
- истории соединений – информация, полученная от предыдущих соединений;
- состоянии уровня приложения – информация о состоянии соединения, полученная из других приложений;
- манипулировании информацией – вычисление разнообразных выражений, основанных на всех вышеперечисленных факторах [4].

2.1.2 Типы межсетевых экранов

Различают несколько типов межсетевых экранов в зависимости от следующих характеристик:

- обеспечивает ли экран соединение между одним узлом и сетью или между двумя, или более различными сетями;
- происходит ли контроль потока данных на сетевом уровне или более высоких уровнях модели OSI;
- отслеживаются ли состояния активных соединений или нет.

В зависимости от охвата контролируемых потоков данных межсетевые экраны подразделяются на:

- традиционный сетевой (или межсетевой) экран – программа (или неотъемлемая часть операционной системы) на шлюзе (устройстве, передающем трафик между сетями) или аппаратное решение, контролирующее входящие и исходящие потоки данных между подключенными сетями (объектами распределённой сети);
- персональный межсетевой экран – программа, установленная на пользовательском компьютере и предназначенная для защиты от несанкционированного доступа только этого компьютера.

В зависимости от уровня OSI, на котором происходит контроль доступа, сетевые экраны могут работать на:

- сетевом уровне, когда фильтрация происходит на основе адресов отправителя и получателя пакетов, номеров портов транспортного уровня модели OSI и статических правил, заданных администратором;

- сеансовом уровне (также известные, как stateful), когда отслеживаются сеансы между приложениями и не пропускаются пакеты, нарушающие спецификации TCP/IP, часто используемые в злонамеренных операциях – сканирование ресурсов, взломы через неправильные реализации TCP/IP, обрыв/замедление соединений, инъекция данных;

- прикладном уровне (или уровне приложений), когда фильтрация производится на основании анализа данных приложения, передаваемых внутри пакета. Такие типы экранов позволяют блокировать передачу нежелательной и потенциально опасной информации на основании политик и настроек [4].

2.1.2.1 Фильтрация на сетевом уровне

Фильтрация входящих и исходящих пакетов осуществляется на основе информации, содержащейся в следующих полях TCP- и IP-заголовков пакетов: IP-адрес отправителя; IP-адрес получателя; порт отправителя; порт получателя.

Фильтрация может быть реализована различными способами для блокирования соединений с определенными компьютерами или портами. Например, можно блокировать соединения, идущие от конкретных адресов тех компьютеров и сетей, которые считаются ненадежными.

К преимуществам такой фильтрации относятся:

- сравнительно невысокая стоимость;
- гибкость в определении правил фильтрации;
- небольшая задержка при прохождении пакетов.

Недостатки:

- не собирает фрагментированные пакеты;
- нет возможности отслеживать взаимосвязи (соединения) между пакетами [4].

2.1.2.2 Фильтрация на сеансовом уровне

В зависимости от отслеживания активных соединений межсетевые экраны могут быть:

- stateless (простая фильтрация), которые не отслеживают текущие соединения (например, TCP), а фильтруют поток данных исключительно на основе статических правил;
- stateful, stateful packet inspection (SPI) (фильтрация с учётом контекста), с отслеживанием текущих соединений и пропуском только таких пакетов, которые удовлетворяют логике и алгоритмам работы соответствующих протоколов и приложений.

Межсетевые экраны с SPI позволяют эффективнее бороться с различными видами DoS-атак и уязвимостями некоторых сетевых протоколов. Кроме того, они обеспечивают функционирование таких протоколов, как SIP, FTP и т. п., которые используют сложные схемы передачи данных между адресатами, плохо поддающиеся описанию статическими правилами, и зачастую несовместимых со стандартными, stateless сетевыми экранами.

К преимуществам такой фильтрации относится:

- анализ содержимого пакетов;
- не требуется информации о работе протоколов прикладного уровня.

Недостатки:

- сложно анализировать данные уровня приложений (возможно с использованием ALG – Application level gateway).

Application level gateway, ALG (шлюз прикладного уровня) – компонент NAT-маршрутизатора, который понимает какой-либо прикладной протокол, и при прохождении через него пакетов этого протокола модифицирует их таким образом, что находящиеся за NAT'ом пользователи могут пользоваться протоколом.

Служба ALG обеспечивает поддержку протоколов на уровне приложений (таких как SIP, H.323, FTP и др.), для которых подмена адресов/портов (Network Address Translation) недопустима. Данная служба определяет тип приложения в пакетах, приходящих со стороны интерфейса внутренней сети и соответствующим образом выполняя для них трансляцию адресов/портов через внешний интерфейс.

Технология SPI (Stateful Packet Inspection) или технология инспекции пакетов с учетом состояния протокола на сегодня является передовым методом контроля трафика. Эта технология позволяет контролировать данные вплоть до уровня приложения, не требуя при этом отдельного приложения посредника или проху для каждого защищаемого протокола или сетевой службы.

Архитектура stateful inspection уникальна потому, что она позволяет оперировать всей возможной информацией, проходящей через машину-шлюз: данными из пакета, данными о состоянии соединения, данными, необходимыми для приложения [4].

2.1.2.3 Фильтрация на прикладном уровне

С целью защиты ряда уязвимых мест, присущих фильтрации пакетов, межсетевые экраны должны использовать прикладные программы для фильтрации соединений с такими сервисами, как, например, Telnet, HTTP, FTP. Подобное приложение называется проху-службой, а хост, на котором работает проху-служба – шлюзом уровня приложений. Такой шлюз исключает прямое взаимодействие между авторизованным клиентом и внешним хостом. Шлюз фильтрует все входящие и исходящие пакеты на прикладном уровне (уровне приложений – верхний уровень сетевой модели) и может анализировать содержимое данных, например, адрес URL, содержащийся в HTTP-сообщении, или команду, содержащуюся в FTP-сообщении. Иногда эффективнее бывает фильтрация пакетов, основанная на информации, содержащейся в самих данных. Фильтры пакетов и фильтры уровня канала не используют содержимое информационного потока при принятии решений о фильтрации, но это можно сделать с помощью фильтрации уровня приложений. Фильтры уровня приложений могут использовать информацию из заголовка пакета, а также содержимого данных и информации о пользователе. Администраторы могут использовать фильтрацию уровня приложений для контроля доступа на основе идентичности пользователя и/или на основе конкретной задачи, которую пытается осуществить пользователь. В фильтрах уровня приложений можно установить правила на основе отдаваемых приложением команд. Например, администратор может запретить конкретному пользователю скачивать файлы на конкретный компьютер с помощью FTP или разрешить пользователю размещать файлы через FTP на том же самом компьютере.

К преимуществам такой фильтрации относятся:

- простые правила фильтрации;

- возможность организации большого числа проверок. Защита на уровне приложений позволяет осуществлять большое количество дополнительных проверок, что снижает вероятность взлома с использованием "дыр" в программном обеспечении;

- способность анализировать данные приложений.

Недостатки:

- относительно низкая производительность по сравнению с фильтрацией пакетов;

- ргоху должен понимать свой протокол (невозможность использования с неизвестными протоколами);

- как правило, работает под управлением сложных ОС [4].

2.1.3 Сравнение аппаратных и программных реализаций межсетевых экранов

Существует два варианта реализации межсетевых экранов - программный и аппаратный.

Первый вариант - наиболее часто используемый в настоящее время и на первый взгляд более привлекательный. Это связано с тем, что, по мнению многих, для его применения достаточно только приобрести программное обеспечение межсетевого экрана и установить на любой компьютер, имеющийся в организации. Однако на практике далеко не всегда в организации находится свободный компьютер, да еще и удовлетворяющий достаточно высоким требованиям по системным ресурсам. Поэтому одновременно с приобретением программного обеспечения приобретается и компьютер для его установки. Потом следует процесс установки на компьютер операционной системы и ее настройка, что также требует времени и оплаты работы установщиков. И только после этого устанавливается и настраивается программное обеспечение системы обнаружения атак.

Второй вариант может быть реализован двумя способами - в виде специализированного устройства и в виде модуля в маршрутизаторе или коммутаторе. Интерес к программно-аппаратным решениям за последнее время возрос. Такие решения, специализированные программно-аппаратные решения, называемые security appliance, постепенно вытесняют "чисто" программные системы. Они поставляются, как специальные программно-аппаратные комплексы, использующие специализированные или обычные операционные системы (как правило, Unix), "урезанные" для выполнения только заданных функций. К достоинству таких решений можно отнести:

- Простота внедрения в технологию обработки информации. Поскольку такие устройства поставляются уже с предустановленной и настроенной операционной системой и защитными технологиями, необходимо только подключить его к сети, что выполняется в течение нескольких минут. И хотя некоторая настройка все же требуется, время, затрачиваемое на нее, существенно меньше, чем в случае установки и настройки межсетевого экрана "с нуля".

- Простота управления. Данные устройства могут управляться с любой рабочей станции Windows или Unix. Взаимодействие консоли управления с устройством осуществляется либо по стандартным протоколам, например, Telnet или SNMP, либо при помощи специализированных или защищенных протоколов, например, Ssh или SSL.

- Производительность. За счет того, что из операционной системы исключаются все "ненужные" сервисы и подсистемы, устройство работает более эффективно с точки зрения производительности и надежности.

- Отказоустойчивость и высокая доступность. Реализация межсетевого экрана в специальном устройстве позволяет реализовать механизмы обеспечения не только программной, но и аппаратной отказоустойчивости и высокой доступности. Такие устройства относительно легко объединяются в кластеры.

- Сосредоточение на защите. Решение только задач обеспечения сетевой безопасности не приводит к трате ресурсов на выполнение других функций, например, маршрутизации и т.п. Обычно, попытка создать универсальное устройство, решающее сразу много задач, ни к чему хорошему не приводит.

Основные направления, присущие и первому, и второму типам:

- обеспечение безопасности входящего и исходящего трафика;
- значительное увеличение безопасности сети и уменьшение риска для хостов подсети при фильтрации заведомо незащищенных служб;
- возможность контроля доступа к системам сети;
- уведомление о событиях с помощью соответствующих сигналов тревоги, которые срабатывают при возникновении какой-либо подозрительной деятельности (попытки зондирования или атаки);
- обеспечение недорогого, простого в реализации и управлении решения безопасности.

Аппаратные и программно-аппаратные межсетевые экраны дополнительно поддерживают функционал, который позволяет:

- препятствовать получению из защищенной подсети или внедрению в защищенную подсеть информации с помощью любых уязвимых служб;
- регистрировать попытки доступа и предоставлять необходимую статистику об использовании Интернет;
- предоставлять средства регламентирования порядка доступа к сети;
- обеспечивать централизованное управление трафиком.

Программные межсетевые экраны, кроме основных направлений, позволяют:

- контролировать запуск приложений на том хосте, где установлены;
- защищать объект от проникновения через "люки" (back doors);
- обеспечивать защиту от внутренних угроз [5].

2.2 Анализ элементной базы для аппаратной реализации межсетевого экрана

Два основных производителя FPGA – Xilinx и Altera, в совокупности занимают более 80% рынка программируемой логики. Линейки микросхем обеих компаний имеют сходные параметры и, зачастую, выбор между ними обуславливается не техническими характеристиками, а предпочтениями заказчика.

Каждая серия FPGA включает различные характеристики, такие как количество логических элементов, объема встроенной памяти, количество умножителей, блоков PLL, блоков цифровой обработки сигналов (DSP), портов ввода-вывода и др.

Рассмотрим основные серии СБИС компании Altera и их характеристики:

1. Серия Stratix - Высокопроизводительные СБИС ПЛ большой логической емкости (до 1 млн. эквивалентных логических элементов). Выпускаются по технологии статического ОЗУ. На рисунке 2.1 приведены характеристики и особенности архитектуры семейства Stratix 10 GX.

Stratix 10 Product Line	GX 400 SX 400	GX 660 SX 660	GX 860 SX 860	GX 1100 SX 1100	GX 1660 SX 1660	GX 2100 SX 2100	GX 2600 SX 2600	GX 2800 SX 2800	GX 4600 SX 4600	GX 6600 SX 6600
Equivalent LUT ¹	378,000	612,000	841,000	1,092,000	1,624,000	2,008,000	2,422,000	2,793,000	4,469,000	6,810,000
Adaptive Logic Modules (ALMs)	128,160	207,360	284,960	370,080	560,640	679,680	821,160	933,120	1,512,820	1,867,680
ALM Registers	612,640	829,440	1,139,840	1,480,320	2,202,160	2,718,720	3,284,600	3,732,480	6,061,280	7,470,720
Hyper-Registers from HyperFlex FPGA Architecture	Millions of Hyper-Registers distributed throughout the monolithic FPGA fabric									
Programmable Clock Trees Synthesizable	Hundreds of synthesizable clock trees									
Maximum Transceiver Count	24	48	48	48	96	96	96	96	24	24
GXT Full Duplex Transceiver Count (30 Gbps)	16	32	32	32	64	64	64	64	16	16
GX Full Duplex Transceiver Count (17.4 Gbps)	8	16	16	16	32	32	32	32	8	8
M20K Memory Blocks	1,837	2,489	3,477	4,401	6,851	6,501	9,963	11,721	7,033	7,033
M20K Memory (Mb)	30	49	68	68	114	127	196	229	137	137
MLAB Memory (Mb)	2	3	4	6	8	11	13	16	23	29
Variable-Precision DSP Blocks	648	1,162	2,016	2,820	3,148	3,744	5,011	5,760	1,980	1,980
18 x 18 Multipliers	1,296	2,304	4,032	5,040	6,290	7,488	10,022	11,520	3,960	3,960
Fixed Point Performance (TMACS) ²	2.6	4.6	8.1	10.1	12.6	16.0	20.0	23.0	7.9	7.9
Single Precision Floating Point (TFLOPS) ³	1.0	1.8	3.2	4.0	5.0	6.0	8.0	9.2	3.2	3.2
Maximum User I/O Pins	392	400	736	736	704	704	1,160	1,160	1,640	1,640

Рисунок 2.1. Характеристики и особенности архитектуры семейства Stratix 10 GX

- Серия Arria - СБИС ПЛ среднего диапазона, оптимизированные для решения телекоммуникационных задач. Все микросхемы этой серии содержат встроенные высокоскоростные приемопередатчики. Выпускаются по технологии статического ОЗУ. На рисунке 2.2 приведены характеристики и особенности архитектуры семейства Arria 10.

Arria 10 SoC Product Line	SX 160	SX 220	SX 270	SX 320	SX 480	SX 570	SX 660
Logic elements (LEs) (K)	160	220	270	320	480	570	660
Adaptive logic modules (ALMs)	61,510	83,730	101,620	118,730	181,720	217,080	250,450
Registers	246,040	334,920	406,480	474,920	727,160	868,320	1,002,160
M20K memory blocks	440	588	750	891	1,438	1,800	2,133
M20K memory (Mb)	9	11	15	17	28	35	42
MLAB counts	1,680	2,227	3,968	4,673	7,137	8,241	9,345
MLAB memory (Mb)	1.0	1.8	2.4	2.8	4.3	5.0	5.7
Variable-precision DSP blocks	156	191	830	985	1,368	1,523	1,688
18 X 19 multipliers	312	382	1,660	1,970	2,736	3,046	3,376
Peak GMACS	343	420	1,826	2,167	3,010	3,351	3,714
Single-precision floating-point multipliers	156	191	830	985	1,368	1,523	1,688
Single-precision floating-point adders	156	191	830	985	1,368	1,523	1,688
Peak giga floating-point operation per second (GFLOPs)	140	172	747	887	1,231	1,371	1,519
Maximum GPIOs	288	288	384	384	492	696	696
Maximum transceiver count (17.4 G)	12	12	24	24	36	48	48
Fractional PLLs	6	6	8	8	12	16	16
I/O PLLs	6	6	8	8	12	16	16

Рисунок 2.2. Характеристики и особенности архитектуры семейства Arria 10

- Серия Cyclone - недорогие СБИС ПЛ для решения широкого круга задач. Предназначены для применения в массовых проектах, где требуются низкая себестоимость и низкое энергопотребление. Выпускаются по технологии статического ОЗУ. На рисунке 2.3 приведены характеристики и особенности архитектуры семейства Cyclone 10 GX.

Product Line	10CX085	10CX105	10CX150	10CX220
KLEs	85	104	150	220
Memory Blocks (20K)	291	382	475	587
Memory Block (Kb)	5,820	7,640	9,500	11,740
Distributed memory (Kb)	653	799	1,152	1,690
Hardened single-precision floating-point multipliers/adders	84	125	156	192
Global clock networks	32	32	32	32
Regional clocks	8	8	8	8
18 x 19 multipliers	168	250	312	384
Hard Memory Controllers (DDR3/L, LPDDR3)	1	2	2	2
Maximum LVDS channels (1.434 Gbps)	72	118	118	118
Maximum user I/O pins	192	284	284	284
Maximum 3 V I/O	48	48	48	48
Transceiver count (12.5 Gbps)	6	12	12	12
PCI Express* (PCIe*) hardened IP blocks (up to Gen2 x4)	1	1	1	1

Рисунок 2.3. Характеристики и особенности архитектуры семейства Cyclone 10 GX

4. Серия MAX 10 - недорогие СБИС ПЛ с энергонезависимой конфигурационной Flash-памятью для решения широкого круга задач. Предназначены для применения в различных отраслях, в том числе в промышленной и автомобильной. На рисунке 2.4 приведены характеристики ресурсов семейства MAX 10 [6].

	Количество логических элементов	Объем встроенного ОЗУ, Кбит	Объем пользовательской FLASH-памяти, Кбит	Количество умножителей 18x18	Количество блоков PLL
10M02	2,000	108	96	16	1, 2
10M04	4,000	189	128	19	1, 2
10M08	8,000	378	256	24	1, 2
10M16	16,000	549	256	45	1, 4
10M25	25,000	756	256	61	1, 4
10M40	40,000	1,260	512	125	1, 4
10M50	50,000	1,638	512	144	1, 4

Рисунок 2.4. Характеристики ресурсов семейства MAX 10

2.3 Анализ средств проектирования

Полноценный маршрут проектирования на СБИС ПЛ содержит множество этапов, ниже приведены ключевые этапы:

- текстовый (языки VHDL, Verilog) или графический ввод проекта;
- синтез описания проекта на архитектурно-независимом уровне регистровых передач (RTL - Register Transfer Level);
- функциональное моделирование проекта – на уровне RTL;
- синтез описания проекта в базе выбранной СБИС ПЛ, разводка в кристалле с учетом ограничений (constraints) заданных пользователем и временного анализа;
- временное моделирование на уровне логических элементов СБИС ПЛ с учетом пользовательских ограничений;
- генерация конфигурационного файла СБИС ПЛ;
- программирование конфигурационного файла в СБИС ПЛ, внутрисхемная отладка проекта.

САПР Quartus II поддерживает весь маршрут проектирования цифровых устройств на базе СБИС ПЛ Altera, начиная с ввода проекта пользователем и заканчивая прошивкой микросхемы программируемой логики и отладкой как самой микросхемы, так и системы, построенной на базе этой СБИС ПЛ, в целом.

В дополнение к Quartus II, использование среды ModelSim-Altera позволяет моделировать и отлаживать проекты, описанные на языках VHDL и Verilog HDL, с использованием графического интерфейса, скриптов и тестбенчей.

Используемый язык описания аппаратуры - Verilog HDL (Verilog Hardware Description Language), используемый для описания и моделирования электронных систем. Принят международным стандартом IEEE Standard 1364-2001. Преимущество – разделение разработки системы и способы ее реализации на различные уровни абстракции (вентильный уровень, уровень регистровых передач, уровень функциональных блоков и т.д.).

2.4 Выводы

В разделе рассмотрены принципы работы, типы межсетевых экранов, проведено сравнение аппаратных и программных подходов к реализации межсетевых экранов, дан обзор современной элементной базы для аппаратной реализации СБИС ПЛ и средств проектирования.

3. РАЗРАБОТКА АРХИТЕКТУРЫ СИСТЕМЫ ФИЛЬТРАЦИИ ПАКЕТОВ И ВЫБОР ПЛАТЫ-ПРОТОТИПА

3.1 Требования к системе фильтрации пакетов и плате-прототипу

Требования к разрабатываемой системе фильтрации пакетов:

- Аппаратная реконфигурируемость – возможность реализации различных алгоритмов фильтрации и архитектур устройства фильтрации на одной платформе.
- Масштабируемость – возможность наращивания числа внешних и внутренних сетей.
- Возможность реализации дополнительных сервисов на базе встроенных процессоров.

Для прототипирования разрабатываемой системы фильтрации пакетов необходимо выбрать плату-прототип, удовлетворяющую приведенным выше требованиям к системе фильтрации пакетов.

3.2 Архитектура системы фильтрации пакетов

Разработанная архитектура системы фильтрации пакетов, удовлетворяющая заданным требованиям, приведена на рисунке 3.1.

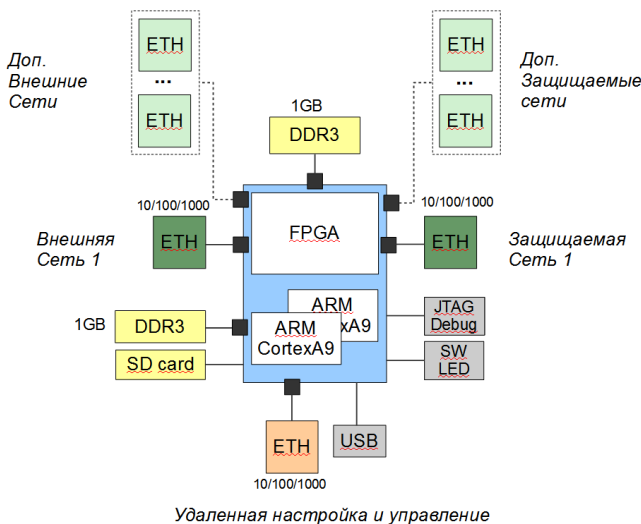


Рисунок 3.1. Разработанная архитектура системы фильтрации пакетов

Разработанная архитектура включает СБИС со встраиваемой системой (SoC – System-on-Chip), включающей:

- процессорную часть (HPS – Hardware Processor System – часть SoC), в которой двух-ядерный процессор ARM Cortex A9);
- логическую часть (FPGA – часть SoC);
- 10/100/1000 Ethernet;
- DDR3 SDRAM;
- USB;
- Micro-SD Card;
- On-Board USB Blaster II;
- Switches, Buttons and LEDs.

Разработанная архитектура позволяет:

- обеспечить подключение нескольких 10/100/1000 Mbps Ethernet каналов к SoC. Таким образом организовать подключение защищаемой и внешней сетей к устройству фильтрации пакетов;
- обеспечить подключение PC, реализующего пользовательский интерфейс управления и анализа состояния устройства фильтрации пакетов, через Ethernet или через USB интерфейсы;
- обеспечить автономную, без управления, работу устройства фильтрации пакетов с хранением правил фильтрации и статистики использования устройства на SD карте;
- обеспечить возможность реализации разветвленный алгоритмов управления и тестирования на базе встроенных процессоров ARM (в том числе с возможностью реализации на базе платформы Linux или RTOS операционных систем).

3.3 Выбранная плата-прототип

В качестве платы-прототипа для реализации и тестирования устройства выбрана плата-прототип SoCKIT, интерфейсы которой приведены на рисунке 3.2, а внешний вид – на рисунке 3.3.



Рисунок 3.2. Интерфейсы платы-прототипа SoCKit



Рисунок 3.3. Внешний вид платы SoCKit

Для проведения тестирования разрабатываемого устройства фильтрации пакетов необходимо иметь возможность подключения как минимум одной внешней и одной внутренней сети. Поэтому, для платы-прототипа был выбран модуль расширения Ethernet-HSMC, структура

которого приведена на рисунке 3.4, внешний вид – на рисунке 3.5[7].

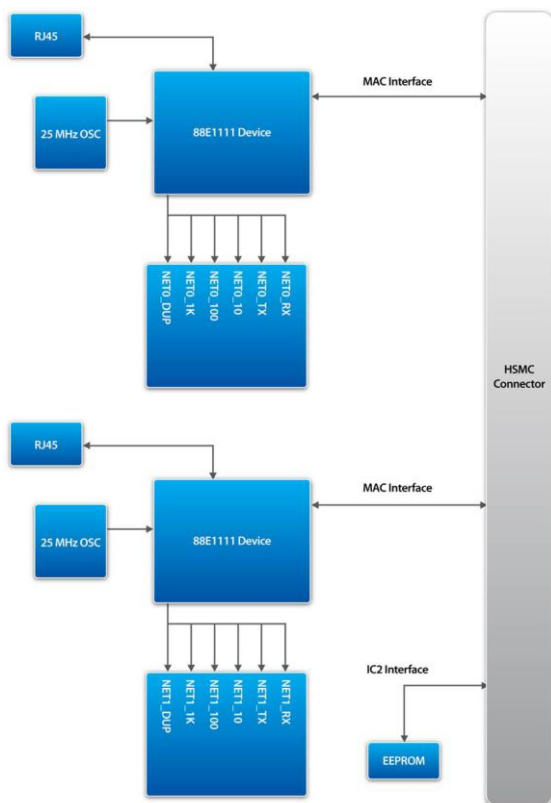


Рисунок 3.4. Структура модуля расширения Ethernet-HSMC



Рисунок 3.5. Внешний вид модуля расширения Ethernet-HSMC

Подключение модуля Ethernet-HSMC к плате SoCKIT осуществляется через HSMC разъем, подключенный к FPGA части SoC. На модуле Ethernet-HSMC используется СБИС Marvell 88E1111, реализующая физический уровень стека протоколов.

Передача данных между модулем Ethernet-HSMC и платой SoCKIT реализована с использованием протокола GMII.

Выбранная плата-прототип с модулем Ethernet-HSMC позволяет:

- реализовать различные алгоритмы фильтрации и архитектур устройства фильтрации на одной платформе;
- возможность наращивания числа внешних и внутренних сетей;
- обеспечить подключение PC, реализующего пользовательский интерфейс управления и анализа состояния устройства фильтрации пакетов, через Ethernet или через USB интерфейсы;
- обеспечить автономную, без управления, работу устройства фильтрации пакетов с хранением правил фильтрации и статистики использования устройства на SD карте;
- возможность реализации дополнительных сервисов на базе встроенных процессоров.

3.4 Выводы

Разработанная архитектура устройства фильтрации обеспечивает возможность:

- аппаратной реконфигурируемости – реализации различных алгоритмов фильтрации и архитектур устройства фильтрации на одной платформе;
- масштабируемости – наращивания числа внешних и внутренних сетей;
- реализации дополнительных сервисов на базе встроенных процессоров.

4 РАЗРАБОТКА СТРУКТУРЫ АППАРАТНО РЕАЛИЗУЕМОГО МЕЖСЕТЕВОГО ЭКРАНА

Разрабатываемая структура устройства фильтрации пакетов должна обеспечивать:

- возможность взаимодействия с внешней (внешними) и защищаемой (защищаемыми) сетями;
- анализ пакетов, поступающих от внешней (внешних) сети;
- задержку пакетов на время анализа пакетов и выработки решения на основе заданных правил;
- выработку решения на основе заданных правил и параметров поступающего пакета;
- хранение правил;
- возможность изменения правил.

4.1 Общая структура межсетевого экрана

Разработанная структура устройства фильтрации пакетов, приведена на рисунке 4.1:

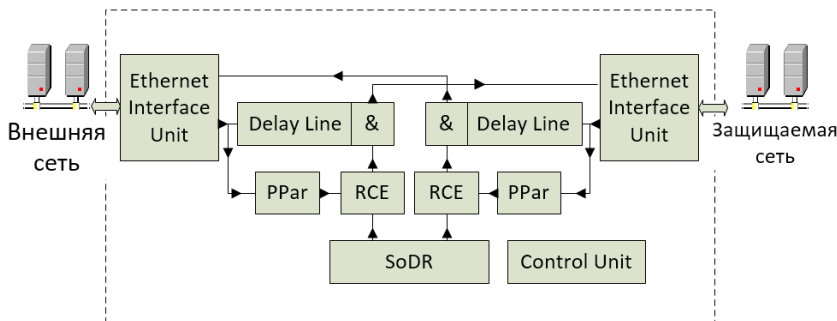


Рисунок 4.1. Структура устройства фильтрации пакетов

В ее состав входят следующие блоки:

- Ethernet Interface Unit – блок взаимодействия с внешней Ethernet РНУ микросхемой (Marvell 88E1111) по интерфейсу GMII;
- PPar ((**P**acket **P**arser) – блок анализа Ethernet пакетов;
- Delay Line – блок буферизации сетевых пакетов;
- RCE (**R**ule **C**hecking **E**ngine) – блок проверки правил и выработки решения;
- SoDR (**S**et of **D**efined **R**ules) – блок хранения правил.

4.2 Разработка структуры блока проверки правил и выработки решения

Задачи блока проверки правил и выработки решений:

- считывание данных (содержимого полей анализируемого пакета) по сигналу готовности с блока PPar;
- считывание текущих правил для всех полей с блока SoDR;
- выработка решения:
 - “пропуск” (accept) - передать пакет;
 - “удаление” (drop) – запретить дальнейшее прохождение пакета.

Разработанная структура блока отображена на рисунке 4.2.

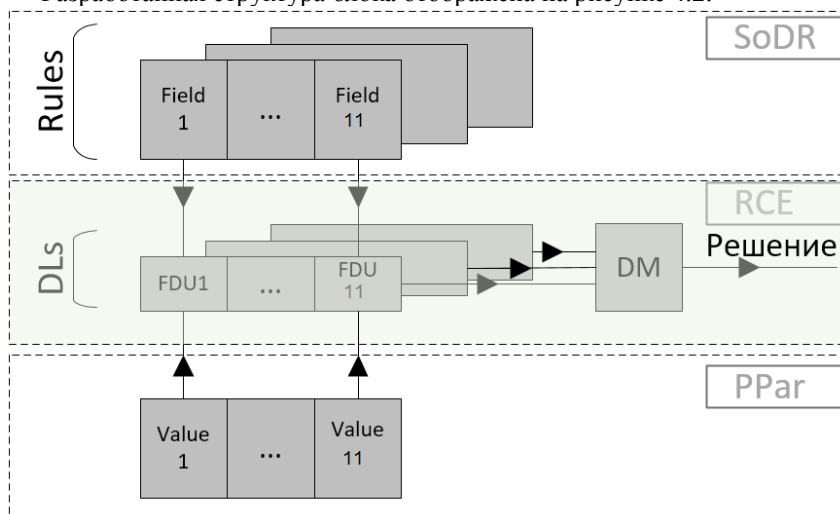


Рисунок 4.2. Структура блока проверки правил и выработки решения

На приведенной выше структуре блока проверки правил (RCE) отображены:

- DL (Decision Line) – модуль вырабатывающий решение для одного правила.
- Каждый модуль DL содержит 11 компонентов FDU (Field Decision Unit) – компонент вырабатывает решение для одного поля в правиле.
- Один модуль DM (Decision Make) – модуль, вырабатывающий общее решение по всем правилам.

4.2.1 Разработка структуры модуля решения DL (Decision Line)

Разработанная структура модуля решений (DL), представлена на рисунке 4.3.

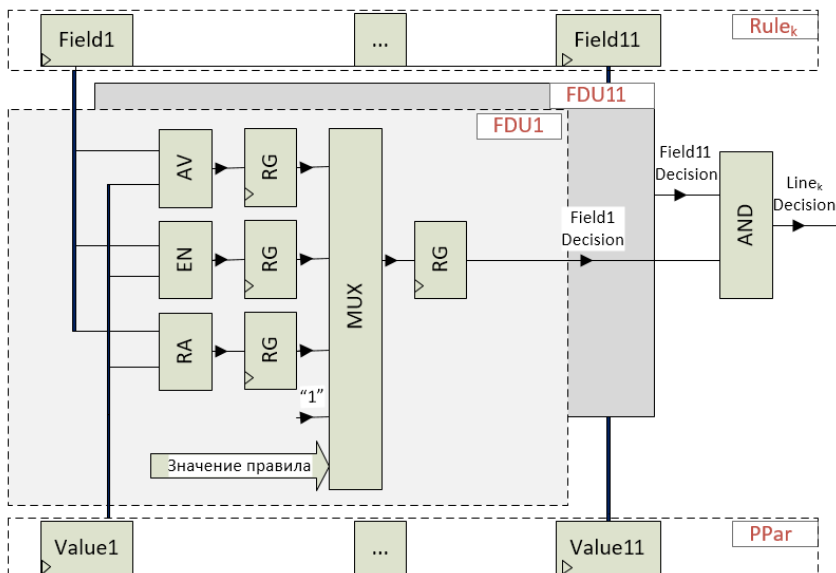


Рисунок 4.3. Структура модуля решений

На структуре изображены:

- AV – Число (сравнение поля и числа, заданного в правиле)
- EN – Перечисление (сравнение поля с числами, заданными в правиле (количество чисел в перечислении – 4))
- RA – Диапазон (сравнение поля с диапазоном, заданным двумя числами (одно определяет нижнюю границу, другое – верхнюю) в правиле)
- MUX – выбирает результат в зависимости от значения правила
- RG – синхронный триггер.
- AND – формирование общего результата для правила.

4.3 Разработка формата хранения правил фильтрации

Для ограничения типов пакетов, которые будет пропускать или удалять межсетевой экран, необходимо создать правила с определенной структурой.

В обобщенном виде любое правило фильтрации представляет собой логическую конструкцию “IF (параметры правила) – THEN (действие правила)”, означающую, что если заголовок поступившего пакета соответствует параметрам правила, то к пакету следует применить действие, указанное в правиле. Допускаются следующие возможные действия над пакетом:

- “пропуск” (accept) - передать пакет (внутри изделия) на выходной фильтрующий интерфейс (интерфейсы);
- “удаление” (drop) – запретить дальнейшее прохождение пакета.

4.3.1 Разработка формата хранения правила

Разработанный формат для хранения правил фильтрации, приведенный на рисунке 4.4, включает 2 группы параметров:

- 1) атрибуты правила;
- 2) параметры пакета.

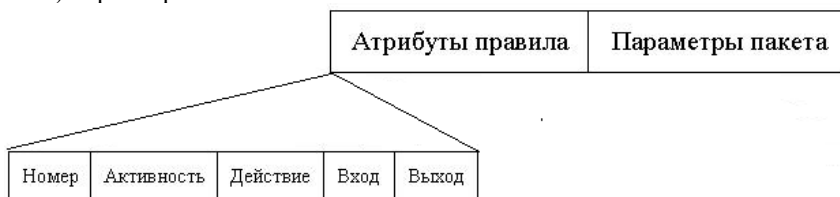


Рисунок 4.4. Формат для хранения правил фильтрации

Атрибуты правила включают следующие поля:

- Номер. Каждое правило имеет номер. Номер правила – целое число в диапазоне от 0 до 255.
- Активность. Каждое правило фильтрации имеет статус активного или не активного. Неактивные правила не учитываются при фильтрации пакетов. Допустимые значения - активно или не активно.
- Действие. Действие над пакетом по данному правилу. Допустимые значения – пропуск или удаление пакета.
- Вход, Выход. Входные (Вход) и выходные (Выход) интерфейсы правила, указывающие направление передачи пакетов. Допустимые значения: 0, 1, 2.

Параметры пакета, указываемые в правиле, включают поля, указанные в таблице 4.1.

Таблица 4.1. Параметры пакета, указываемые в правиле

Параметр	Описание, возможные значения
Тип кадра	Формат кадра Ethernet. Возможные значения: 0 - EthII, 1 - 802.3/Novell, 2 - Eth SNAP, 3 - 802.3/LLC, любой.
Eth-протокол	Код протокола, вложенного в Ethernet-кадр. Возможные значения: - число в шестнадцатеричном формате; - любой.
MAC-адрес источника	- число в шестнадцатеричном коде; - любой
MAC-адрес приемника	- число в шестнадцатеричном коде; - любой
IP-протокол	Протокол, инкапсулированный в IP-пакет. - десятичный номер протоколов (по RFC 1700); - любой
Источник IP-адрес	IP-адрес источника. Записываются в соответствии с правилами записи IP-адресов и разбиения IP-сетей на подсети.
Источник порт	Номер порта TCP/UDP источника пакета. Используется, если в параметре Протокол указаны протоколы TCP или UDP. Допустимые значения: Номер - целое число от 0 до 65535; Диапазон номеров (через дефис): Например, 0-1024; Любой
Приемник IP-адрес	IP-адрес приемника. Записываются в соответствии с правилами записи IP-адресов и разбиения IP-сетей на подсети.
Приемник Порт	Номер порта TCP/UDP источника пакета. Используется, если в параметре Протокол указаны протоколы TCP или UDP.
Тип/код сообщения ICMP	Возможные значения: - число - список - любой
Фрагментация IP-пакетов	Указывает, к каким пакетам применяется данное правило: Да – только к фрагментированным, Нет- только к не фрагментированным, Любой – к фрагментированным и не фрагментированным пакетам.

4.3.2 Разработка формата записи поля в правиле

В Техническом задании определены следующие значения правил для каждого поля:

- Число – поле соответствует числу, заданному в правиле;
- Перечисление – поле соответствует одному из чисел, заданных в правиле (количество чисел в перечислении – 4);
- Диапазон – значение поля попадает в диапазон из двух чисел (одно определяет нижнюю границу, другое – верхнюю), заданных в правиле;
- Любое – поле может принимать любое значение.

В соответствии с заданными значениями правил для каждого поля в правиле определена структура записи, представленная в таблице 4.2.

Таблица 4.2. Структура записи для каждого поля в правиле

Значение правила	Код значения	Структура записи для каждого поля в правиле			
		Раздел 4	Раздел 3	Раздел 2	Раздел 1
Любое	00	-	-	-	-
Число	01	-	-	-	Число
Перечисление	10	Число 4	Число 3	Число 2	Число 1
Диапазон	11			Верхняя граница	Нижняя граница

Для того, чтобы исключить поле из рассмотрения в *i*-ом правиле для него следует установить код значения правила – любое (00). Для того чтобы исключить все *i*-ое правило из рассмотрения следует для всех полей этого правила поставить код – любое (00).

Для того чтобы в режиме Перечисление задать 1, 2 или 3 числа, пустые поля следует заполнить одним из задаваемых чисел.

4.4 Анализ вариантов аппаратной реализации межсетевого экрана на СБИС ПЛ

Для оценки производительности разрабатываемой системы и ее отдельных блоков необходимо знать для 10/100/1000 Mbps Ethernet минимальное и максимальное значения размера пакета, частоты поступления пакетов, размера заголовка пакета. В таблице 4.4, сведены указанные параметры [8].

Таблица 4.3. Параметры 10/100/1000 Mbps Ethernet

Скорость Ethernet	10 Мбит/сек		100 Мбит/сек		1000 Мбит/сек	
Размер кадра, байт	мин	макс	мин	макс	мин	макс
	64	1518	64	1518	64 (дополняется до 512)	1518
Межкадровый интервал, бит	47	96	47	96	47	96
Скорость передачи, кадры/сек	14.8K	812	148K	8,1K	1,48M	81K
Скорость передачи данных, Мбит/сек	5,5	9,8	55	98	550	980
Интервал между кадрами, мкс	67	1200	6,7	120	0,7	12

Для совместимости с 10\100Mbps Ethernet, в 1000Mbps Ethernet минимальный размер кадра был оставлен прежним — 64 байта, а к кадру добавилось дополнительное поле carrier extension (расширение носителя), которое дополняет кадр до 512 байт. Естественно, поле не добавляется в случае, если размер кадра больше 512 байт. Таким образом, результирующий минимальный размер кадра в 1000Mbps Ethernet получился равным 512 байтам. Межкадровый интервал IPG - InterPacket Gap – применяется чтобы один из узлов не смог захватить всю полосу пропускания в сети и дав всем равные возможности а также дать время сетевому адаптеру на обработку полученного кадра. В технологиях передачи Ethernet, FastEthernet & Gigabit Ethernet IPG совпадает, отличается лишь продолжительность данного интервала.

В Ethernet предусмотрена возможность снижения межкадрового

интервала активным оборудованием при обнаружении переполнения буфера и его быстрой очистки. Межкадровый интервал не должен понижаться ниже, чем на 49 битовых последовательностей ($\text{Min IPG} = 96 - 49 = 47\text{bit}$). Иначе кадр, сохраненный в буфер, не будет обработан перед поступлением нового.

В таблице 4.5 приведена оценка требуемого количества тактов частоты работы блока RCE для обработки последовательно идущих самых коротких кадров (что соответствует максимальной загрузке блока).

Таблица 4.4. Оценка требуемого количества тактов частоты работы блока RCE

Скорость Ethernet	10Mbps	100Mbps	1000Mbps
Максимальная частота поступления кадров (теоретическая, 64 – минимальный размер кадра, 8 – бит в 1 байте), кадров/сек	10МГц/(64*8)	100МГц/(64*8)	1000МГц/(512*8)
Число тактов на обработку кадра (Тактовая частота работы блока RCE 125 МГц)	125МГц /10МГц*(64*8) 6400 тактов	125МГц /100МГц*(64*8) 640 тактов	125МГц /1000МГц*(512*8) 512 тактов
Число тактов на обработку кадра (Тактовая частота работы блока RCE 250 МГц)	250МГц /10МГц*(64*8) 12800 тактов	250МГц /100МГц*(64*8) 1280 тактов	250МГц /1000МГц*(512*8) 1024 такта

Анализ таблицы показывает, что при числе правил 256 (как задано в ТЗ) и числе полей 11, целесообразно использовать смешанную реализацию.

4.5 Выводы

Разработанная структура является универсальной и допускает:

- Параллельную реализацию – один модуль DL на каждое правило; один компонент FDU на каждое анализируемое поле;
- Последовательную реализацию – один компонент FDU на все поля и все правила;
- Смешанную реализацию – количество модулей DL и компонентов FDU варьируется в зависимости от поставленных ограничений.

В результате оценки требуемого количества тактов частоты работы блока RCE целесообразно использовать смешанную реализацию.

5 РАЗРАБОТКА ОПИСАНИЯ МОДУЛЕЙ И УСТРОЙСТВА ФИЛЬТРАЦИИ ПАКЕТОВ НА ЯЗЫКЕ VERILOG

Для реализации разработанной структуры устройства фильтрации пакетов на базе СБИС ПЛ необходимо разработать синтезируемые описания модулей устройства и всего устройства в целом на языке Verilog.

5.1 Разработка блока взаимодействия с внешней Ethernet (приемная часть)

Данный блок обрабатывает сетевые пакеты, приходящие от Ethernet РНУ, по интерфейсу GMII, и на выходе выдает данные пакета, исключая стартовые биты и межкадровый интервал.

5.1.1 Временная диаграмма приема пакетов

Временная диаграмма корректного приема пакета приведена на рисунке 5.1[9]:

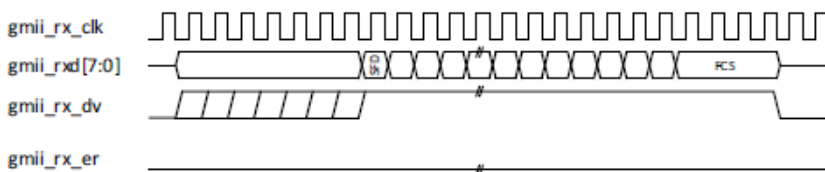


Рисунок 5.1. Временная диаграмма корректного приема пакета

Как видно из диаграммы, информация о пакете и данные пакета идут после преамбулы (до 7 байт) и бита SFD (Start of Frame Delimiter) и заканчиваются спадом сигнала gmii_rx_dv - сигналом достоверности принятых данных.

5.1.2 Описание блока на языке Verilog

Символ блока gmii_gx приведен на рисунке 5.2:

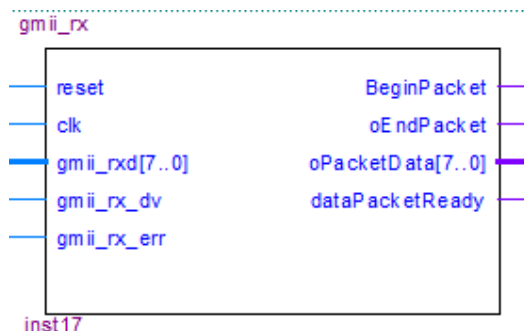


Рисунок 5.2. Символ компонента gmii_rx

Порты ввода-вывода блока описаны в таблице 5.1.

Таблица 5.1. Порты ввода-вывода блока gmii_rx

Входы	Выходы
reset – сигнал сброса	BeginPacket – сигнал, информирующий о начале приема пакета
clk – частота для тактирования приема данных	oEndPacket – сигнал, информирующий об окончании приема пакета
gmii_rxd[7..0] – группа параллельных сигналов данных, выдаются из трансивера в MAC-контроллер	oPacketData – данные сетевого пакета
gmii_rx_dv – сигнал достоверности принятых данных. Трансивер устанавливает этот сигнал, когда он получает достоверный пакет данных и, соответственно, выдает достоверные данные на rxd	dataPacketReady – сигнал, информирующий о получении пакета
gmii_rx_err – сигнал ошибки приема	

Текстовое описание блока на языке Verilog приведено в файле gmii_rx.v. Блок реализован в виде конечного автомата. Состояния автомата приведены в таблице 5.2.

Таблица 5.2. Состояния конечного автомата

Состояние автомата	Описание
State_idle	Состояние простоя. Ожидание принимаемых данных
State_preamble	Преамбула
State_data	Состояние приема данных пакета
State_drop	Прекращение анализа текущего кадра
State_ErrEnd	Ошибка при приеме данных
State_IFG	Межкадровый интервал

Граф переходов конечного автомата приведен на рисунке 5.3, таблица переходов на рисунке 5.4.

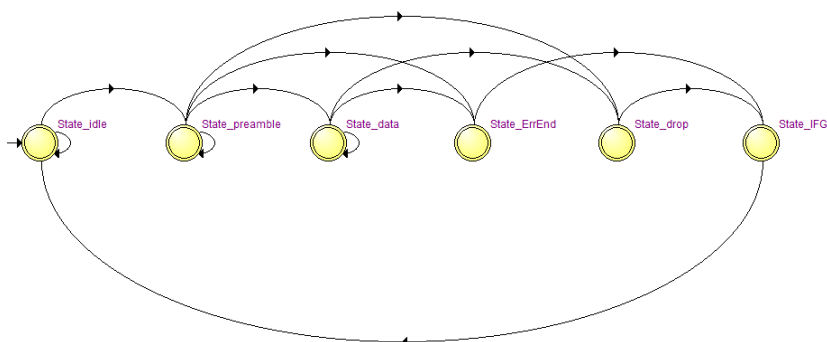


Рисунок 5.3. Граф переходов конечного автомата

State Table			
	Source State	Destination State	Condition
1	State_data	State_ErrEnd	(!gmii_rx_dv)
2	State_data	State_drop	(gmii_rx_err).(gmii_rx_dv)
3	State_data	State_data	(!gmii_rx_err).(gmii_rx_dv)
4	State_drop	State_IFG	
5	State_ErrEnd	State_IFG	
6	State_idle	State_idle	(!always1)
7	State_idle	State_preamble	(always1)
8	State_IFG	State_idle	
9	State_pream...	State_ErrEnd	(!gmii_rx_dv)
10	State_pream...	State_drop	(!gmii_rx_err). (gmii_rx_dv).(lrx_d[0]) + (!gmii_rx_err). (gmii_rx_dv).(rx_d[0]). (lrx_d[1]).(lrx_d[2]) + (!gmii_rx_err). (gmii_rx_dv).(rx_d[0]). (lrx_d[1]).(rx_d[2]). (lrx_d[3]).(lrx_d[4]) + (!gmii_rx_err). (gmii_rx_dv).(rx_d[0]). (lrx_d[1]).(rx_d[2]). (lrx_d[3]).(rx_d[4]). (lrx_d[5]).(lrx_d[6]) + (!gmii_rx_err). (gmii_rx_dv).(rx_d[0]). (lrx_d[1]).(rx_d[2]). (lrx_d[3]).(rx_d[4]).(rx_d[5]) + (!gmii_rx_err). (gmii_rx_dv).(rx_d[0]). (lrx_d[1]).(rx_d[2]).(rx_d[3]) + (!gmii_rx_err). (gmii_rx_dv).(rx_d[0]). (rx_d[1]) + (gmii_rx_err). (gmii_rx_dv)
11	State_pream...	State_preamble	(rx_d[0]).(lrx_d[1]).(rx_d[2]). (lrx_d[3]).(rx_d[4]). (lrx_d[5]).(rx_d[6]). (lrx_d[7]).(!gmii_rx_err). (gmii_rx_dv)
12	State_pream...	State_data	(!gmii_rx_err). (gmii_rx_dv).(rx_d[0]). (lrx_d[1]).(rx_d[2]). (lrx_d[3]).(rx_d[4]). (lrx_d[5]).(rx_d[6]).(rx_d[7])

Рисунок 5.4. Таблица переходов конечного автомата

5.2 Разработка блока взаимодействия с внешней Ethernet (передающая часть)

Данный блок передает сетевые пакеты, хранящиеся в блоке буферизации сетевых пакетов, в Ethernet PHY по интерфейсу GMII, на основе выработанного решения для пакета в блоке RCE.

5.2.1 Временная диаграмма передачи пакетов

Временная диаграмма корректной передачи пакета приведена на рисунке 5.5[9]:

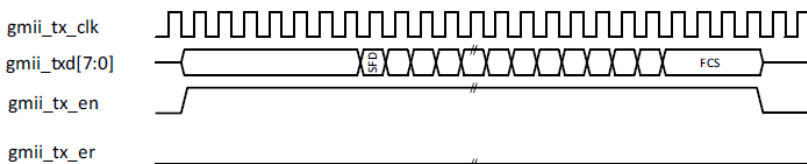


Рисунок 5.5. Временная диаграмма корректной передачи пакета

Как видно из диаграммы, чтобы осуществить передачу принятого пакета во внутреннюю сеть требуется установить сигнал tx_en в «1» и передавать побайтно буферизированный принятый пакет из блока буферизации сетевых пакетов.

5.2.2 Описание блока на языке Verilog

Символ компонента gmii_tx приведен на рисунке 5.6:

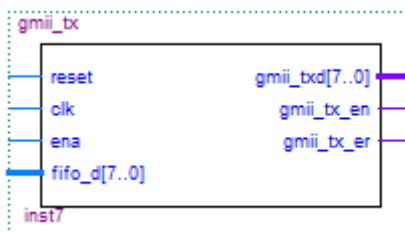


Рисунок 5.6. Символ компонента gmii_tx

Порты ввода-вывода компонента представлены в таблице 5.3. Компонент расположен в файле gmii_tx.v.

Таблица 5.3. Порты ввода-вывода блока gmii_tx

Входы	Выходы
reset – сигнал сброса	gmii_txd[7..0] – группа параллельных сигналов данных, передающиеся в трансивер
clk – тактовый сигнал для передачи	gmii_tx_en – сигнал разрешения передачи
fifo_d[7..0] – данные, поступающие с блока буферизации сетевых пакетов	gmii_rx_err – сигнал ошибки передачи
ena – сигнал, разрешающий работу блока	

5.3 Разработка блока анализа Ethernet пакетов

Задача блока обеспечить формирование параметров анализируемого пакета, сопровождая выходные данные признаком готовности.

5.3.1 Разработка алгоритма поиска параметров пакета

Для анализа сетевого пакета вначале требуется определить тип Ethernet-кадра каждого входящего сетевого пакета, схема алгоритма приведена на рисунке 5.7.

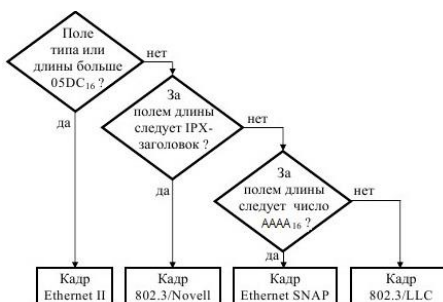


Рисунок 5.7. Схема определения Ethernet кадра

Для кадров типа Ethernet II требуется определять наличие тега VLAN в принятом кадре. Результат определения влияет на позицию (номера) байтов значимых полей в принятом кадре. Для определения наличия тега VLAN анализируются байты B12-B13.

Определение тега VLAN реализуется следующим образом:

- Если значение байтов B12-B13 не равно 0x8100, то кадр не содержит тег VLAN.
- Если значение байтов B12-B13 = 0x8100, то кадр содержит тег VLAN, при этом позиции байтов поля Ethproto, а также всех последующих заголовков (ipproto, srcip, dstip) входного пакета сдвигаются на 4 байта.

Вложенные IP пакеты анализируются следующим образом:

- Для кадров, в которые вложен пакет IP, следует определять значение длины заголовка IP-пакета. Длина заголовка (ipheader_length) влияет на позицию (номера) байтов полей вложенных протоколов (TCP, UDP, ICMP) в принятом кадре. Для определения длины IP-заголовка анализируются биты Bit 4 - Bit7 первого байта заголовка IP (в кадре Ethernet II, при отсутствии VLAN – это байт B14 принятого кадра). Параметр ipheader_length может принимать целочисленные значения от 5 до 15.
- Если значение поля ipheader_length = 5 (стандартный заголовок), то поля srcport, dstport, icmp соответствуют байтам пакета, указанным в табл. 5.4
- При значении поля ipheader_length > 5, поля srcport, dstport и icmp в пакете сдвигаются на величину $[(ipheader_length - 5) \times 4]$ байт.

Требуется также определять факт фрагментации дейтаграммы. Для этого анализируются биты Bit1 и Bit2 байта B6 (флаги DF и MF, соответственно), биты 3-7 байта B6 и биты 0-7 байта B7 заголовка IP (смещение фрагмента). Для кадра Ethernet II без VLAN это биты 1-7 байта B20 и биты 0-7 байта B21 принятого кадра.

Параметры пакета и их расположение в пакете представлены в таблице 5.4.

Таблица 5.4. Параметры/поля пакета и их расположение в пакете

N	Параметр/Поле	Размер поля (бит)	Примечания
1	Frame =<Тип Ethernet-кадра>	2	смотри алгоритм на рисунке 5.7
2	Dstmac=<MAC_адрес_назначения>	48	Байты B0-B5
3	Srcmac=<MAC_адрес_источника>	48	Байты B6-B11
4	Ethproto=<протокол>	16	Байты B12-B13 (при наличии VLAN - байты B16-B17)
5	Ipproto=<протокол>	8	Байт B23 (при VLAN - B27)
6	srcip4=<IPv4_адрес_источника>	32	Байты B26-B29 (при VLAN B30-B33)
7	dstip4=<IPv4_адрес_назначения>	32	Байты B30-B33 (B34-B37)
8	isFragment=<фрагментация>	1	алгоритм приведен выше
8	Srcport=<порт_источника>	16	Байты B34-B35 и учесть ipheader_length. (Байты B38-B39 и учесть ipheader_length).
9	Dstport=<порт_назначения>	16	Байты B36-B37 и учесть ipheader_length. (Байты B40-B41 и учесть ipheader_length).
10	icmp=<тип_код_icmp> - тип и код ICMP-сообщения	8+8	Байты B34-B35 и учесть ipheader_length. (Байты B38-B39 и учесть ipheader_length).

5.3.2 Описание блока на языке Verilog

Блок осуществляет синхронную обработку входных данных и обеспечивает формирование признаков и полей анализируемого пакета, сопровождая выходные данные признаком готовности.

Компонент расположен в файле PPar.v.

Символ компонента PPar приведен на рисунке 5.8:

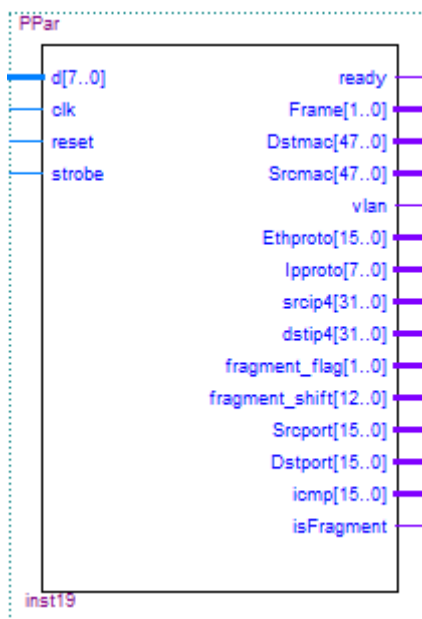


Рисунок 5.8. Символ компонента PPar

Интерфейс блока:

- Входы
 - Тактовый сигнал – clk;
 - Сигнал асинхронного сброса – reset;
 - Сигнал начала пакета – strobe;
 - Входные 8-разрядные данные – d[7:0], синхронизированные сигналом clk.
- Выходы
 - Сигнал готовности выходных данных – ready;
 - Параметры и значения полей анализируемого пакета:
 1. [1:0] Frame,
 2. [47:0] Dstmac,
 3. [47:0] Srcmac,
 4. vlan,
 5. [15:0] Ethproto,
 6. [7:0] Ipproto,
 7. [31:0] srcip4,
 8. [31:0] dstip4,
 9. [1:0] fragment_flag,

10. [12:0] fragment_shift,
11. [15:0] Srcport,
12. [15:0] Dstport,
13. [15:0] icmp_type_code,
14. isFragment.

5.4 Разработка блока хранения правил – SoDR (Set of Defined Rules)

Задача данного блока хранить и предоставлять доступ к правилам фильтрации. Блок реализован на модулях статической памяти, встроенной в FPGA часть СБИС SoC.

5.4.1 Расчет параметров блока

В таблице 5.5 приведен расчет количества разрядов в записи одного правила. Максимальное число правил – 256.

Таблица 5.5. Расчет количества разрядов в записи одного правила

N	Параметр/Поле	Размер	Количество разрядов в
		поля	
1	Порядковый номер	8	8
2	Активность	1	1
3	Действие	1	1
4	Направление Вход/Выход	2	2
5	Frame =<Тип Ethernet-кадра>	2	10
6	Dstmac=<MAC_адрес_назначения>	48	194
7	Srcmac=<MAC_адрес_источника>	48	194
8	Ethproto=<протокол>	16	6
9	Ipproto=<протокол>	8	34
10	srcip4=<IPv4_адрес_источника>	32	130
11	dstip4=<IPv4_адрес_назначения>	32	130

Окончание таблицы 5.5

12	fragment=<фрагментация>	2	2
13	Srcport=<порт_источника>	16	66
14	Dstport=<порт_назначения>	16	66
15	icmp=<тип_код_icmp> - тип и код	8+8	66
Общее количество разрядов в записи правила			971

5.4.2 Реализация блока

Для реализации данного блока целесообразно воспользоваться готовыми решениями, реализованными в IP Components Quartus II, список которых приведен на рисунке 5.9.

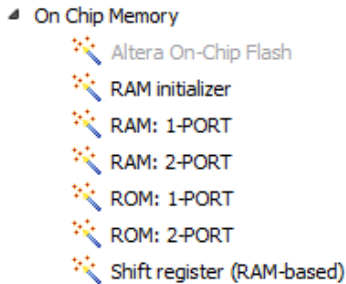


Рисунок 5.9. IP Components Quartus II

Наиболее подходящий компонент для реализации – двухпортовая ROM, компонент RAM избыточен, т.к. запись правил не производим, только считывание.

Двухпортовая память для хранения правил представлена на рисунке 5.10. Для реализации одного блока памяти требуется 26 модулей памяти M10K.

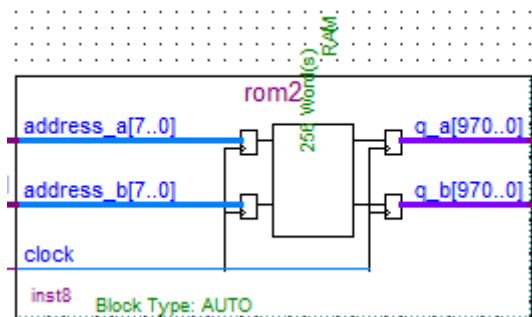


Рисунок 5.10. Двухпортовая память для хранения правил

Интерфейс блока:

- Входы
 - address_a[7..0] – адреса 1 правила;
 - address_b[7..0] – адрес 2 правила;
 - clock – тактовый сигнал.
- Выходы
 - q_a[970..0] – правило 1;
 - q_b[970..0] – правило 2.

Память инициализируется hex файлом, в котором хранятся правила. Процесс формирования hex файла описан в разделе 5.7.

5.5 Разработка блока проверки правил и выработки решения RCE (Rule Checking Engine)

Задачи блока проверки правил и выработки решений:

- считывание данных (содержимого полей анализируемого пакета) по сигналу готовности с блока PPar;
- считывание текущих правил для всех полей с блока SoDR;
- выработка решения:
 - “пропуск” (accept) - передать пакет;
 - “удаление” (drop) – запретить дальнейшее прохождение пакета.

Структура блока отображена на рисунке 4.2.

5.5.1 Разработка блока решения DL (Decision Line)

Блок вырабатывает решение (пропуск/удаление) для двух правил (первоначально предполагалась выработка решения для одного правила, но поскольку используется двухпортовая память, то целесообразно вырабатывать решения параллельно сразу для двух правил), сравнивая параметры пакета и учитывая атрибуты правила.

Компонент расположен в файле DL.v.

Символ компонента DL приведен на рисунке 5.11:

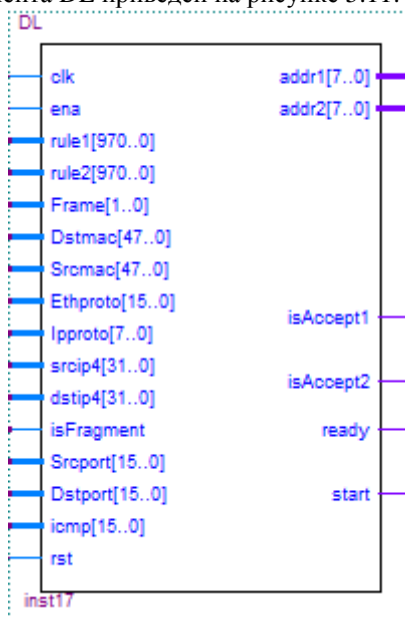


Рисунок 5.11. Символ компонента DL

Интерфейс блока:

- Входы
 - Тактовый сигнал – clk;
 - Сигнал разрешения работы – ena;
 - Правила, считываемые из памяти – rule1[970:0], rule2[970:0];
 - Параметры и значения полей анализируемого пакета:
 1. [1:0] Frame,
 2. [47:0] Dstmac,
 3. [47:0] Srcmac,
 4. [15:0] Ethproto,

5. [7:0] Ipproto,
 6. [31:0] srcip4,
 7. [31:0] dstip4,
 8. [15:0] Srcport,
 9. [15:0] Dstport,
 10. [15:0] icmp_type_code,
 11. isFragment.
- Выходы
 - Сигнал готовности выходных данных – ready;
 - Решения для считываемых правил – isAccept1, isAccept2;
 - Сигнал выработки решений для первого и второго правила (используется для синхронизации блоков) - start;
 - Адреса правил, считываемых из памяти, которые на следующем такте поступят на вход блока – addr1[7:0], addr2[7:0].

Структура модуля DL, анализирующего параметры пакеты, представлена на рисунке 4.3 и реализована на языке Verilog HDL.

Модуль DL содержит 11 компонентов FDU (Field Decision Unit), вырабатывающих решение для одного поля в правиле и реализованных параллельно.

5.5.2 Разработка модуля DM (Decision Make), вырабатывающего общее решение по всем правилам

Модуль, на основании решений для каждого правила, принимает решение для пакета:

- “пропуск” (accept) - передать пакет;
- “удаление” (drop) – запретить дальнейшее прохождение пакета.

Компонент расположен в файле DM.v.

Символ компонента DM приведен на рисунке 5.12:

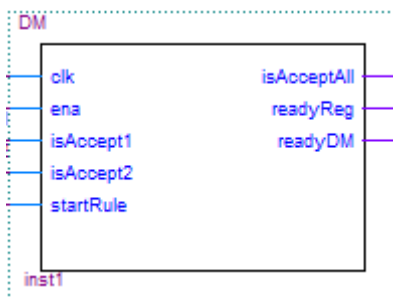


Рисунок 5.12. Символ компонента DM

Интерфейс блока:

- Входы
 - Тактовый сигнал – clk;
 - Сигнал разрешения работы – ena;
 - Решения для правил – isAccept1, isAccept2;
 - Сигнал, что пришло решение для первого правила – startRule;
- Выходы
 - Сигнал готовности выходных данных – readyDM;
 - Сигнал, информирующий о том, что все решения пришли – readyReg (используется для отладки);
 - Решение для всех правил – isAcceptAll.

Алгоритм работы блока прост – решения для каждого правила храним в сдвигающем регистре, убеждаемся, что решения по всем правилам пришли, и собираем их оператором &.

5.6 Разработка блока буферизации сетевых пакетов

Задачей данного блока является буферизация сетевого пакета и передача его в блок взаимодействия с внешней Ethernet PHY микросхемой (на передачу).

Для реализации временного буфера целесообразно воспользоваться параметризованным модулем LPM_FIFO из состава библиотеки Megafunctions/LPM пакета Quartus II. Очередь (First In First Out, FIFO) – разновидность запоминающего устройства с последовательным доступом, из которого слова считываются в том порядке, в котором были записаны. Запись в FIFO и считывание из него – независимые операции и могут выполняться одновременно. Для управления записью и считыванием FIFO имеет управляющие сигналы.

Временный буфер FIFO представлен на рисунке 5.13.

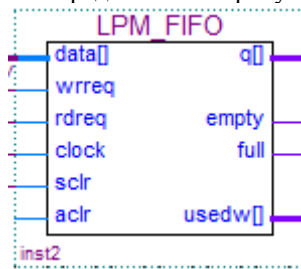


Рисунок 5.13. Временный буфер FIFO

Интерфейс блока:

- Входы
 - data[] – входные данные;
 - wrreq – сигнал разрешения записи;
 - rdreq – сигнал разрешения чтения;
 - clock – тактовый сигнал;
 - sclr – синхронная сброс очереди;
 - aclr – асинхронный сброс очереди.
- Выходы
 - q[] – считываемые данные;
 - empty – сигнал, информирующий что очередь пуста;
 - full – сигнал, информирующий что очередь заполнена;
 - usedw[] – сигнал, показывающий текущее количество слов в очереди.

5.7 Разработка процедуры и программы задания правил

Как видно из раздела 5.4 (блок хранения правил), для задания одного правила требуется 971 бит. Полностью вручную задавать правила крайне неудобно по следующим причинам:

1. Ненаглядность – по правилу нельзя сказать, какую фильтрацию оно выполняет;
2. Трудозатратность – задавать 971 бит вручную занимает много времени;
3. Большая вероятность ошибки – следствие первых двух причин.

Вследствие выше приведенных причин было решено написать программу для задания правил в hex формате.

Правила было решено задавать в формате json – легко читаемый

текстовый формат, удобный для задания и редактирования правил (в отличие от hex), также имеются библиотеки для работы с ним. Считается независимым от языка и может использоваться практически с любым языком программирования [10].

Выбран язык программирования Python, т.к. является скриптовым языком с достаточно простым синтаксисом и большим количеством библиотек.

5.7.1 Алгоритм работы программы

Описание работы программы представлено на рисунке 5.14.

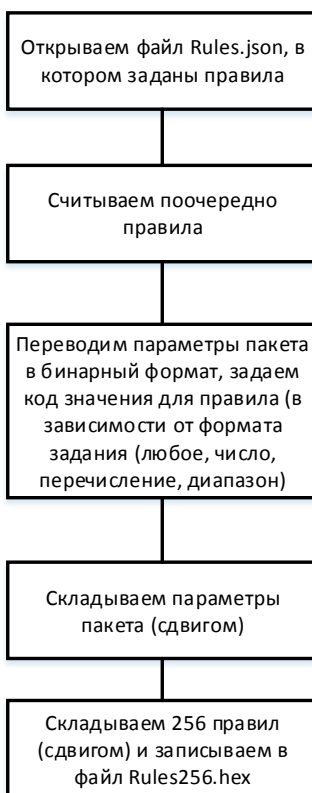


Рисунок 5.14. Описание работы программы

Правила задаются и редактируются в файле Rules.json. Считываются правила с помощью функции load библиотеки pythpn. Далее для каждого правила переводим параметры/поля пакета из шестнадцатеричного и десятичного в двоичный формат с помощью функций hex_to_bin(), dec_to_bin(), dec_with_point_to_bin(). Потом складываем (сдвигом) все, переведенные в двоичный формат, параметры/поля пакета – получаем одно правило в двоичном формате. Далее также складываем все 256 правил. Запись правил в hex файл происходит с помощью функции tofile(“имя файла”, “формат”) библиотеки IntelHex.

5.7.2 Формат задания правил и описание программы

Формат задания параметров пакета, указываемых в правиле, приведен в таблице 5.6.

Таблица 5.6. Параметры пакета, указываемые в правиле

Параметр	Формат задания
Тип кадра	число в десятичном формате от 0 до 3 0 - EthII, 1 - 802.3/Novell, 2 - Eth SNAP, 3 - 802.3/LLC
Eth-протокол	число в шестнадцатеричном формате
MAC-адрес источника	число в шестнадцатеричном коде
MAC-адрес приемника	число в шестнадцатеричном коде
IP-протокол	десятичный номер протоколов (по RFC 1700).
Источник IP-адрес	в десятичном формате, подсети через точку (пример 192.168.0.1)
Источник порт	десятичный номер от 0 до 65535
Приемник IP-адрес	в десятичном формате, подсети через точку (пример 192.168.0.1)
Приемник Порт	десятичный номер от 0 до 65535
Тип/код сообщения ICMP	число в шестнадцатеричном коде
Фрагментация IP-пакетов	1 – только к фрагментированным, 0 - только к не фрагментированным

Пример задания правила в формате json приведен на рисунке 5.15:

```
{
  "number": "0",
  "activity": "1",
  "action": "1",
  "direction": "00",
  "registration": "0",
  "Frame": "1",
  "Dstmac": "AABBCCDDEEFF",
  "Srcmac": "AAAAAAAAAAAA",
  "Ethproto": "0800",
  "Ipproto": "06,11,01",
  "srcip4": "192.168.0.1",
  "dstip4": "32.32.32.32",
  "isFragment": "1",
  "Srcport": "2048-3048",
  "Dstport": "1024,1025,1026",
  "icmp": ""
},
```

Рисунок 5.15. Пример задания правила в формате json

На рисунке 5.16 приведена функция перевода параметра пакета из шестнадцатеричного формата в двоичный, которая вначале определяет форму записи поля (число, перечисление, диапазон, любое), а потом переводит в bin формат с помощью библиотеки bitstring, функции pack(), которая упаковывает значения и параметры и возвращает тип BitStream (непостоянный контейнер битов с методами и свойствами, которые позволяют ему быть проанализированным как поток битов).

```

def hex_to_bin(field, size):
    # all
    if (field == ""):
        return bitstring.pack("0b00, uint:size_bit=0, uint:size_bit=0, uint:size_bit=0, uint:size_bit=0", size_bit=size)
    # диапазон
    if ("-" in field):
        return bitstring.pack("0b11, uint:size_bit=0, uint:size_bit=0, hex:size_bit, hex:size_bit", *field.split("-"),
                                size_bit=size)
    s = len(field.split(",")) # size field
    # 1 число
    if (s == 1):
        return bitstring.pack("0b01, uint:size_bit=0, uint:size_bit=0, uint:size_bit=0, hex:size_bit",
                                field.split(",")[0], size_bit=size)
    # перечисление
    if ("," in field):
        if (s == 4):
            return bitstring.pack("0b10, hex:size_bit, hex:size_bit, hex:size_bit, hex:size_bit", *field.split(","),
                                    size_bit=size)
        if (s == 3):
            return bitstring.pack("0b10, uint:size_bit=0, hex:size_bit, hex:size_bit, hex:size_bit", *field.split(","),
                                    size_bit=size)
        if (s == 2):
            return bitstring.pack("0b10, uint:size_bit=0, uint:size_bit=0, hex:size_bit, hex:size_bit",
                                    *field.split(","), size_bit=size)

```

Рисунок 5.16. Функция перевода из шестнадцатеричного в двоичный формат

Ниже приведен фрагмент программы, собирающий все правила в переменную `rules_bin` и записывающий правила в файл `rules256.hex`.

```

rules_bin = BitArray()
for x in rules:
    rule_bin = to_hex(x)
    rules_bin += rule_bin
ih = IntelHex()
ih.puts(0, rules_bin.tobytes())
ih.tofile("rules256.hex", "hex")

```

Текст программы полностью приведен в приложении ПБ.8.

5.7.3 Формат задания правил и описание программы

Тестирование было осуществлено с помощью логирования – производилась проверка, что в hex файле записаны заведомо корректные данные.

Также производилось тестирование с помощью `asserts`. `Asserts` - это специальная конструкция, позволяющая проверять предположения о значениях произвольных данных в произвольном месте программы. Эта конструкция автоматически сигнализирует при обнаружении

некорректных данных с указанием места обнаружения некорректных данных.

5.8 Выводы

В результате проделанной работы разработаны синтезируемые описания модулей устройства и всего устройства в целом на языке Verilog, разработана программа задания правил на языке Python.

6 РАЗРАБОТКА ТЕСТОВ И МОДЕЛИРОВАНИЕ

Для проверки разработанных синтезируемых описаний блоков необходимо разработать набор тестов и осуществить моделирование.

6.1 Моделирование блока взаимодействия с внешней Ethernet (приемная часть)

Описание блока приведено в разделе 5.1.

6.1.1 *Описание теста*

Моделирование производилось в рамках пакета Modelsim, позволяющего использовать текстовое описание теста и осуществлять анализ правильности работы тестируемого модуля.

Для реализации тестирования разработан тест на языке Verilog (исходный код приведен в приложении ПБ.2). В разработанном тесте входные данные (имитируемый Ethernet пакет, приходящий от Ethernet РНУ) считываются из файла packet.txt. Данные в файле packet.txt - сетевые пакеты, полученные с помощью приложения WireShark, к которым добавлены преамбула, стартовый бит и межкадровый интервал имитируя физический уровень.

Тест позволяет осуществлять проверку правильности работы разработанного блока, убедившись, что данные на выходе совпадают с полученными в WireShark.

6.1.2 *Результаты моделирования*

Временные диаграммы работы блока приведены на рисунках 6.1 и 6.2.

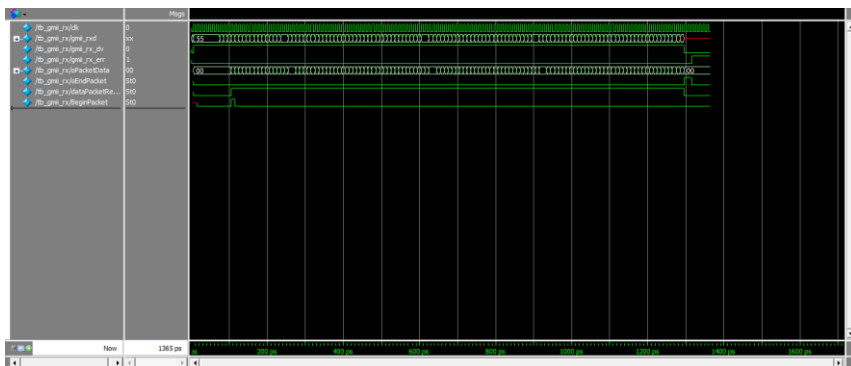


Рисунок 6.1. Временная диаграмма работы блока

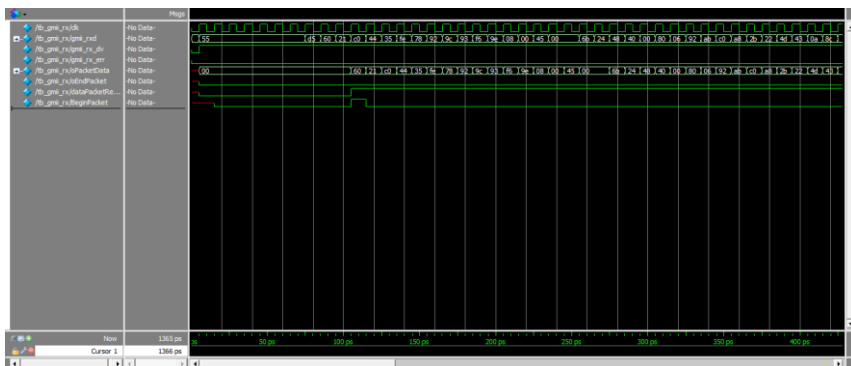


Рисунок 6.2. Временная диаграмма работы блока, укрупненный вариант

Как видно из временных диаграмм, блок правильно обрабатывает сетевые пакеты и на выходе выдает данные пакета, исключая преамбулу, стартовый бит и межкадровый интервал.

6.2 Моделирование блока взаимодействия с внешней Ethernet (передающая часть)

Описание блока приведено в разделе 5.2.

6.2.1 Описание теста

Проверка правильности функционирования блока осуществлена в рамках пакета Quartus II с использованием встроенной в пакет графической системы моделирования, предполагающей графический ввод

тестов и визуальный анализ результатов.

На вход блока подаются произвольные данные, проверяется соответствие передачи стандарту GMII, временная диаграмма корректной передачи приведена на рисунке 5.5.

6.2.2 Результаты моделирования

Временная диаграмма работы блока приведена на рисунке 6.3.

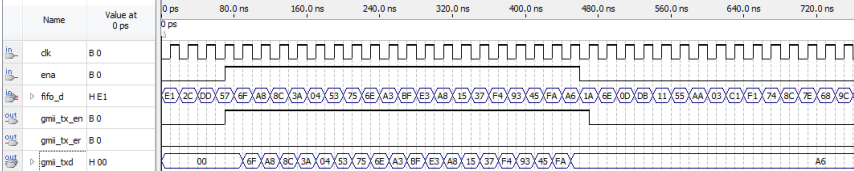


Рисунок 6.3. Временная диаграмма работы блока

С момента появления фронта сигнала ena, на выход блока последовательно подаются данные, сопровождаясь фронтом сигнала tx_en.

Полученная временная диаграмма соответствует диаграмме корректной передачи пакета, приведенной на рисунке 5.5.

6.3 Моделирование блока анализа Ethernet пакетов

Описание блока приведено в разделе 5.3.

6.3.1. Описание теста

Проверка правильности работы блока осуществлена с помощью пакета ModelSim с использованием данных реальных пакетов, полученных с помощью приложения WireShark (поля и параметры пакетов заранее известны). Пример сетевого пакета и полей этого пакета приведен на рисунке 6.4.

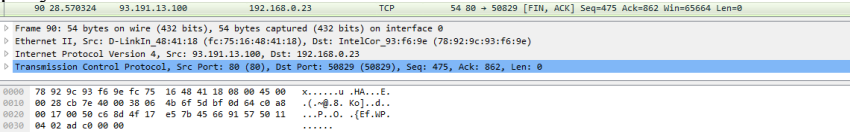
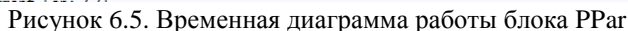


Рисунок 6.4. Пример сетевого пакета в среде WireShark

Для реализации тестирования разработан тест на языке Verilog (исходный код приведен в приложении ПБ.4). В разработанном тесте входные данные (анализируемый Ethernet пакет) считываются из файла packet.txt. Данные в файле packet.txt - сетевые пакеты, полученные с

Тест имитирует интерфейс на входе блока, подавая сетевые пакеты из файла packet.txt и позволяет осуществлять проверку правильности работы разработанного блока, сверив полученные результаты с параметрами пакета, полученными в WireShark.

Временная диаграмма работы блока PPar, на вход которого поданы данные сетевого пакета, и пакет в среде Wireshark, приведены на рисунке 6.5.



Проведенное моделирование показало правильность работы разработанного блока.

6.4 Моделирование блока решения DL (Decision Line)

Описание блока приведено в разделе 5.5.1.

6.4.1 *Описание теста*

Проверка правильности работы блока осуществлена с помощью пакета ModelSim с использованием параметров пакета, полученных в результате моделирования блока анализа Ethernet пакетов и сохраненных в файл.

Тест имитирует интерфейс на входе блока, подавая на вход параметры пакета и правила фильтрации, и позволяет осуществлять проверку правильности выработки решения (заранее известного для заданных правил).

6.4.2 *Результаты моделирования*

Временная диаграмма работы блока приведена на рисунке 6.7. Решения принимались на основе двух правил фильтрации, приведенных на рисунке 6.6. Обработываемый сетевой пакет (приведенный на рисунке 6.5) удовлетворяет только первому правилу (второму правилу не удовлетворяет параметр IP-протокол (Ipproto) – т.к. на вход поступает TCP пакет (номер протокола - 06).

```
{
  "number": "0",
  "activity": "1",
  "action": "1",
  "direction": "00",
  "registration": "0",
  "Frame": "",
  "Dstmac": "6021c04435fe",
  "Srcmac": "78929c93f69e",
  "Ethproto": "0800",
  "Ipproto": "06,11,01",
  "srcip4": "",
  "dstip4": "",
  "isFragment": "0",
  "Srcport": "",
  "Dstport": "",
  "icmp": ""
},
{
  "number": "1",
  "activity": "1",
  "action": "1",
  "direction": "00",
  "registration": "0",
  "Frame": "",
  "Dstmac": "6021c04435fe",
  "Srcmac": "78929c93f69e",
  "Ethproto": "0800",
  "Ipproto": "11,01",
  "srcip4": "",
  "dstip4": "",
  "isFragment": "0",
  "Srcport": "",
  "Dstport": "",
  "icmp": ""
},
```

Рисунок 6.6. Правила фильтрации, подаваемые на вход блока DL

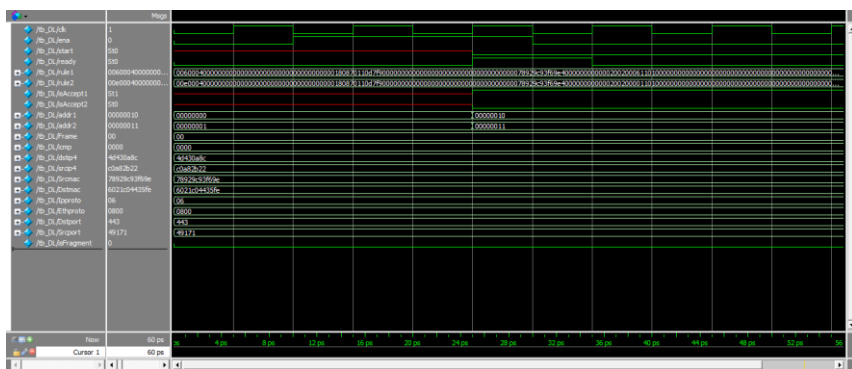


Рисунок 6.7. Временная диаграмма работы блока DL

Как видно из временной диаграммы, блок правильно выработал решение для заданных правил.

Дальнейшее моделирование (меняя правила и сетевые пакеты на входе) показало правильность работы разработанного блока.

**6.5 Моделирование модуля DM (Decision Make),
вырабатывающего общее решение по всем правилам**

Описание блока приведено в разделе 5.5.2.

6.5.1. Описание теста

Проверка правильности функционирования блока осуществлена в рамках пакета Quartus II с использованием встроенной в пакет графической системы моделирования.

На вход блока подаются последовательно решения (пропуск/удаление) для каждого правила, проверяется правильность решения общего решения для всего пакета.

6.5.2. Результаты моделирования

Временные диаграммы работы блока приведены на рисунках 6.8 и 6.9.

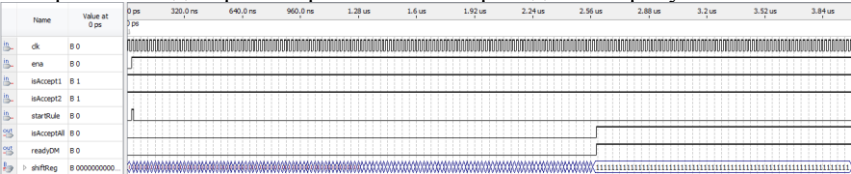


Рисунок 6.8. Временная диаграмма работы блока DM

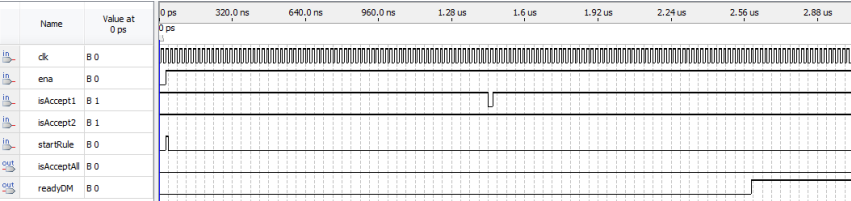


Рисунок 6.9. Временная диаграмма работы блока DM

На первой диаграмме для каждого правила принято решение о пропуске пакета. Принято верное общее решение – пропуск пакета.

На второй диаграмме для одного из правил принято решение об удалении пакета. Принято верное общее решение – удаление пакета.

Проведенное моделирование показало правильность работы разработанного блока.

6.6 Выводы

В результате проведенной работы были созданы тесты для проверки работоспособности каждого из блоков системы фильтрации. Результаты моделирования показывают, что все блоки работают в соответствии с заданным алгоритмом.

7 СИНТЕЗ И РЕАЛИЗАЦИЯ НА СБИС ПЛ

Синтез и реализация устройства фильтрации системы на базе СБИС ПЛ осуществлялась с помощью среды Quartus II.

7.1 Настройки системы синтеза и трассировки

Настройки среды Quartus II использовались базовые по умолчанию. Для проверки проекта использовано средство диагностики Design Assistant. Для проверки проекта нужно во вкладке Design Assistant установить галочку напротив Run Design Assistant during compilation. Настройки ввода-вывода приведены на рисунке 7.1.

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard
in KEY[0]	Input	PIN_AE9	3A	B3A_N0	PIN_AE9	2.5 V
in KEY3	Input	PIN_AD11	3A	B3A_N0	PIN_AD11	2.5 V
out LED[3]	Output	PIN_AD7	3A	B3A_N0	PIN_AD7	3.3-V LVTTL
out myNET0_RESETN	Output	PIN_G13	8A	B8A_N0	PIN_G13	2.5 V
in myNET0_RX_CLK	Input	PIN_D10	8A	B8A_N0	PIN_D10	2.5 V
in myNET0_RX_D[7]	Input	PIN_B6	8A	B8A_N0	PIN_B6	2.5 V
in myNET0_RX_D[6]	Input	PIN_D9	8A	B8A_N0	PIN_D9	2.5 V
in myNET0_RX_D[5]	Input	PIN_E12	8A	B8A_N0	PIN_E12	2.5 V
in myNET0_RX_D[4]	Input	PIN_B5	8A	B8A_N0	PIN_B5	2.5 V
in myNET0_RX_D[3]	Input	PIN_D12	8A	B8A_N0	PIN_D12	2.5 V
in myNET0_RX_D[2]	Input	PIN_E1	8A	B8A_N0	PIN_E1	2.5 V
in myNET0_RX_D[1]	Input	PIN_D11	8A	B8A_N0	PIN_D11	2.5 V
in myNET0_RX_D[0]	Input	PIN_D1	8A	B8A_N0	PIN_D1	2.5 V
in myNET0_RX_DV	Input	PIN_D2	8A	B8A_N0	PIN_D2	2.5 V
in myNET0_RX_ER	Input	PIN_B1	8A	B8A_N0	PIN_B1	2.5 V
out myNET1_RESETN	Output	PIN_H8	8A	B8A_N0	PIN_H8	2.5 V
out myNET1_RX_ER	Output	PIN_K8	8A	B8A_N0	PIN_K8	2.5 V
out myNET1_TX_CLK	Output	PIN_K7	8A	B8A_N0	PIN_K7	2.5 V
out myNET1_TX_D[7]	Output	PIN_H7	8A	B8A_N0	PIN_H7	2.5 V
out myNET1_TX_D[6]	Output	PIN_J7	8A	B8A_N0	PIN_J7	2.5 V
out myNET1_TX_D[5]	Output	PIN_C4	8A	B8A_N0	PIN_C4	2.5 V
out myNET1_TX_D[4]	Output	PIN_D4	8A	B8A_N0	PIN_D4	2.5 V
out myNET1_TX_D[3]	Output	PIN_F8	8A	B8A_N0	PIN_F8	2.5 V
out myNET1_TX_D[2]	Output	PIN_F9	8A	B8A_N0	PIN_F9	2.5 V
out myNET1_TX_D[1]	Output	PIN_D5	8A	B8A_N0	PIN_D5	2.5 V
out myNET1_TX_D[0]	Output	PIN_E4	8A	B8A_N0	PIN_E4	2.5 V
out myNET1_TX_EN	Output	PIN_E2	8A	B8A_N0	PIN_E2	2.5 V
in OSC_50_B5B	Input	PIN_Y26	5B	B5B_N0	PIN_Y26	2.5 V

Рисунок 7.1. Настройки ввода-вывода

7.2 Анализ результатов синтеза и трассировки

Структура компонентов проекта и аппаратные затраты на каждый блок приведены в ПА.1.

Аппаратурные затраты проекта приведены на рисунке 7.2:

Top-level Entity Name	firewall
Family	Cyclone V
Device	5CSXFC6D6F31C8ES
Timing Models	Preliminary
Logic utilization (in ALMs)	2,528 / 41,910 (6 %)
Total registers	1929
Total pins	25 / 499 (5 %)
Total virtual pins	0
Total block memory bits	253,952 / 5,662,720 (4 %)

Рисунок 7.2. Аппаратурные затраты

Как видно из результатов синтеза, использовано 1929 регистров, 6% логических элементов и 4% блоков памяти.

RTL Viewer – утилита, которая дает возможность увидеть логическую реализацию проекта в графическом виде. Это очень полезный инструмент для анализа результатов синтеза HDL проектов [11]. RTL Viewer устройства фильтрации приведен в приложении ПА.2.

7.3 Выводы

Синтез устройство фильтрации на базе платы-прототипа SoCKIT CV с помощью среды Quartus II успешно выполнен. Проанализировав аппаратные затраты проекта, видно, что ресурсов платы-прототипа недостаточно для полностью параллельной реализации (для этого требуется увеличить количество блоков решений DL и блоков хранения правил до 128 и незначительно подкорректировать блок выработки решений для всех правил) – не хватает логических элементов и блоков памяти.

8 ТЕСТИРОВАНИЕ НА ПЛАТЕ-ПРОТОТИПЕ

Для комплексного тестирования разработанного устройства необходимо разработать процедуру проведения тестирования на базе платы-прототипа, и реализовать тестирование.

8.1 Разработка процедуры тестирования

На рисунке 8.1 приведена разработанная структура отладочного комплекса.

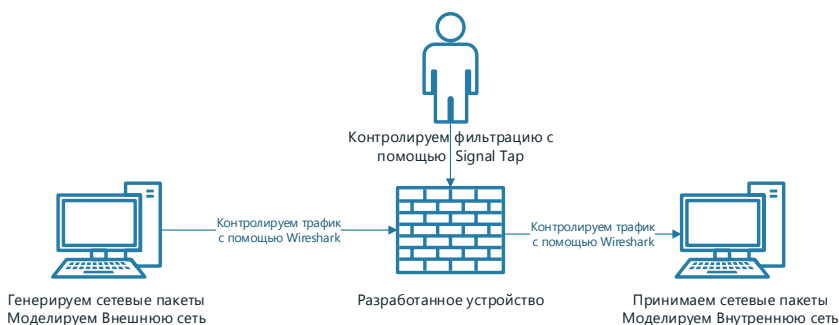


Рисунок 8.1 - Структура отладочного комплекса

Разработанная процедура тестирования включает в себя следующие этапы:

- Формирование набора правил в формате HEX;
- Настройка средства анализа внутренних сигналов СБИС ПЛ – SignalTapII;
- Порождение файла конфигурации СБИС ПЛ с заданным набором правил;
- Настройка внешнего системного окружения (внешнюю и внутреннюю сеть);
- Конфигурирование СБИС ПЛ на плате-прототипе;
- Проведение тестирования с контролем трафика внутренней и внешней сети и сигналов СБИС ПЛ.

Для проведения отладки и испытания разработанного устройства необходимо собрать комплекс, включающий: плату SoCKIT-CV с установленным модулем расширения Ethernet-HSMC, ПК с установленным пакетом Quartus II, средство программирования СБИС ПЛ

- USB-Blaster; два ethernet кабеля, подключаемые к модулю расширения с помощью RJ-45.

8.2 Настройка системного окружения и контроль трафика

8.2.1 Настройка отладочного комплекса

Для генерирования сетевых пакетов использовалась библиотека scapy. Scapy – интерактивная оболочка и программная библиотека для манипулирования сетевыми пакетами на языке программирования Python.

Настройки виртуальной машины версии Linux version 3.2.0-4-486 (в ней создавались сетевые пакеты с помощью библиотеки scapy), приведены на рисунке 8.2.

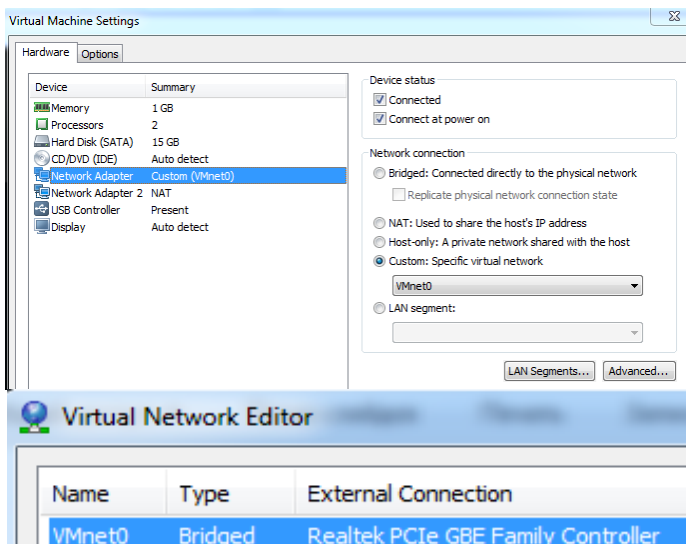


Рисунок 8.2 - Настройки виртуальной машины

8.2.2 Контроль передачи сгенерированного сетевого пакета

Пример создания UDP пакета с помощью библиотеки scapy приведен на рисунке 8.3:

```
>>> b
<IP frag=0 proto=udp dst=192.168.0.23 |<UDP dport=45 |<Raw load='TEST2' |>>>
>>> sendp(Ether()/b,iface="eth0")
```

Рисунок 8.3. Пример создания UDP пакета

Контроль трафика созданного пакета с помощью Wireshark приведен на рисунке 8.4. На рисунке также видны параметры созданного пакета (подчеркнуты).

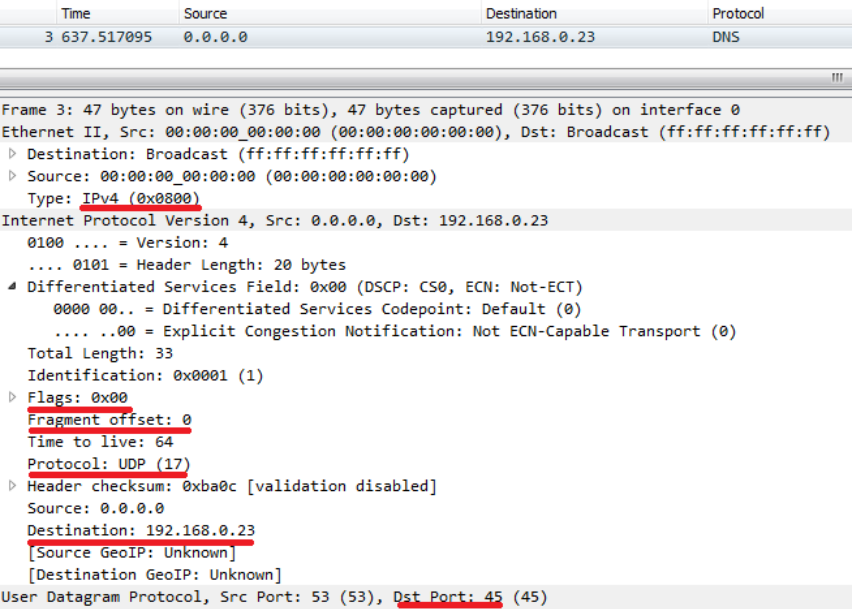


Рисунок 8.4 Контроль трафика с помощью Wireshark

Контролируем созданный пакет в Signal Tap, временная диаграмма приведена на рисунке 8.5:

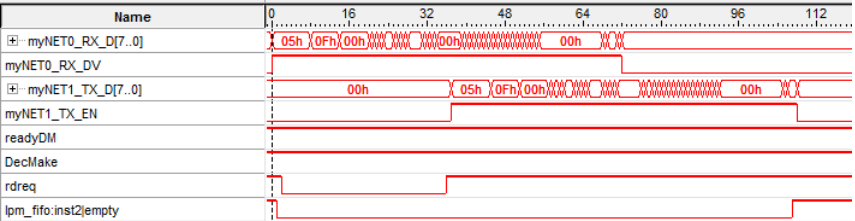


Рисунок 8.5. Созданный пакет в Signal Tap

Таким способом, как представлено на рисунках выше, контролируется сетевой трафик.

8.3 Тестирование

Настройка средства анализа внутренних сигналов СБИС ПЛ SignalTap II Logic Analyzer приведена на рисунке 8.6.

Node			Data Enable	Trigger Enable	Trigger Conditions
Type	Alias	Name	22	22	1 Basic OR
in		myNET0_RX_D[7..0]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	EEEEEEEEb (OR)
*		myNET0_RX_DV	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	X
out		myNET1_TX_D[7..0]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	xxh (OR)
*		myNET1_TX_EN	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
*		readyDM	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	X
*		DecMake	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	X
*		rdreq	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	X
*		lpm_fifo:inst2 empty	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	X

Рисунок 8.6. Настройка средства анализа внутренних сигналов

Для тестирования был создан набор сетевых пакетов, имеющих различные параметры/поля, чтобы проверить работу разных правил фильтрации. Было задано 10 различных правил, отвечающие за проверку своих параметров/полей пакета, остальные 246 правил пропускали сетевые пакеты.

На рисунке 8.7 представлена временная диаграмма, на которой сетевой пакет удовлетворяет правилам фильтрации.

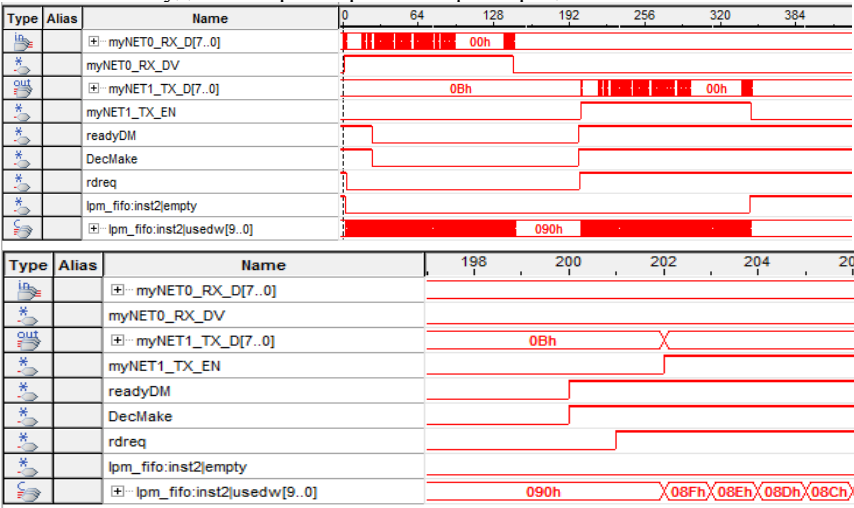


Рисунок 8.7. Временная диаграмма при пропуске пакета (обычный и укрупненный вариант)

Как видно из диаграммы, устройство при приеме сетевого кадра (после преамбулы и стартового бита), сбрасывает решение по предыдущему кадру, блок буферизации (FIFO) принимает поступающий пакет (увеличивается usedw), далее принимается решение по каждому правилу. Когда все решения готовы, принимается общее решение по всем правилам (DecMake) и появляется фронт сигнала готовности (readyDM). После этого, из блока буферизации (по rdreq, usedw уменьшается) принятый пакет передается во внутреннюю сеть.

На рисунке 8.8 представлена временная диаграмма, на которой сетевой пакет не удовлетворяет правилам фильтрации.

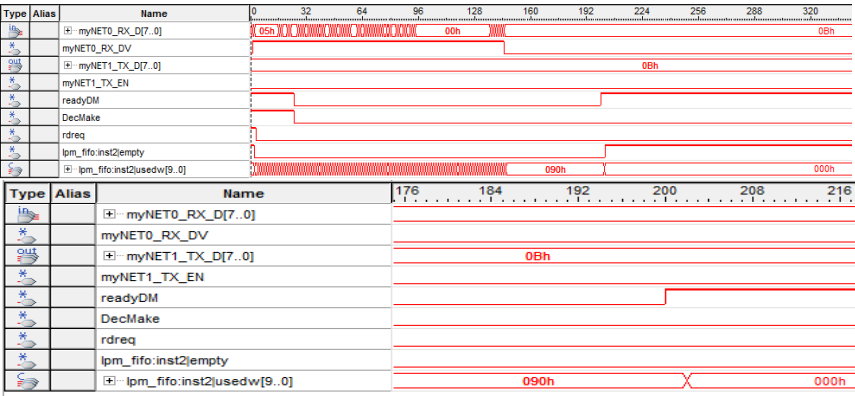


Рисунок 8.8. Временная диаграмма при удалении пакета (обычный и укрупненный вариант)

Как видно из диаграммы, устройство при приеме кадра (после преамбулы и стартового бита), сбрасывает решение по предыдущему пакету, блок буферизации (FIFO) принимает поступающий пакет (увеличивается usedw), далее принимается решение по каждому правилу. Когда все решения готовы, принимается общее решение по всем правилам (DecMake) и появляется фронт сигнала готовности (readyDM). После этого, блок буферизации сбрасывается (usedw равно нулю), принятый пакет удаляется.

Проведенное тестирование показало правильность работы разработанного межсетевого экрана.

8.4 Выводы

Для комплексного тестирования разработанного устройства разработана процедура проведения тестирования на базе платы-прототипа, и проведено тестирование. Разработанный межсетевой экран успешно прошел тестирование.

9 АНАЛИЗ БЫСТРОДЕЙСТВИЯ И ПРОИЗВОДИТЕЛЬНОСТИ

Для оценки производительности разрабатываемой системы и ее отдельных блоков необходимо знать для 10/100/1000 Mbps Ethernet минимальное и максимальное значения размера пакета, частоты поступления пакетов, размеры заголовка пакета. В таблице 9.1, сведены указанные параметры [8].

Таблица 9.1. Параметры 10/100/1000 Mbps Ethernet

Скорость Ethernet	10 Мбит/сек		100 Мбит/сек		1000 Мбит/сек	
Размер кадра, байт	мин	макс	мин	макс	мин	макс
	64	1518	64	1518	64 (дополняется до 512)	1518
Скорость передачи, кадры/сек	14.8K	812	148K	8,1K	1,48M	81K

Для подсчета скорости обработки кадров в секунду необходимо подсчитать количество тактов работы разработанного устройства. Время работы складывается из времени работы блока анализа пакетов и блока выработки решения, так как эти блоки работают последовательно (блок выработки решения работает после работы блока анализа пакетов).

Время работы блока анализа пакетов = 90/50 (мин. возм. пакет) тактов. 90 тактов – максимальное количество тактов работы блока анализа пакетов (соответствует максимальному сдвигу ip заголовка, наличию тега vlan). Такт работы блока – соответствует частоте приема данных (для 1000 Мбит/с каждый байт данных принимается с частотой 125МГц). 50 тактов – максимальное количество тактов работы блока анализа пакетов при обработке минимального размера кадра (соответствует максимальной загрузке устройства).

Время работы блока проверки правил и выработки решения зависит от структуры (которая является универсальной), допускающей

- параллельную реализацию;
- последовательную реализацию;
- смешанную.

Для последовательной реализации время работы блока выработки

решения составляет – 5634 тактов (умножается время работы компонента FDU (2 такта), количество полей правил (11) и количество правил (256)).

Для параллельной реализации – 4 такта (поля и правила проверяются параллельно).

Для смешанной реализации – время зависит от степени распараллеливания и оценивается в каждом отдельном случае.

Таким образом, время работы устройства при последовательной реализации составляет - 5724/5684 (мин. размер кадра) тактов, для параллельной – 94/54 (мин. размер кадра) тактов.

Для расчета скорости обработки кадров в секунду тактовую частоту работы устройства (соответствует частоте подачи кадров (125 МГц для 1 Гбит/с) требуется поделить на время работы устройства (на количество тактов работы). Расчеты скорости обработки кадров/с приведены в таблице 9.2

Таблица 9.2 Расчет производительности (скорость обработки кадров/с)

Скорость\Тактовая частота	125 МГц	250 МГц
Последовательная, кадры/с	$125 \cdot 10^6 / 5724 = 0.021\text{M}$	$250 \cdot 10^6 / (5724 + 90) = 0.043\text{M}$
Последовательная м.п., кадры/с	$125 \cdot 10^6 / 5684 = 0.022\text{M}$	$250 \cdot 10^6 / (5684 + 50) = 0.043\text{M}$
Параллельная, кадры/с	$125 \cdot 10^6 / 94 = 1.32\text{M}$	$250 \cdot 10^6 / (94 + 90) = 1.35\text{M}$
Параллельная м.п., кадры/с	$125 \cdot 10^6 / 54 = 2.31\text{M}$	$250 \cdot 10^6 / (54 + 50) = 2.4\text{M}$

Также для увеличения производительности можно поднять частоту работы устройства (например, рассмотрен вариант в 2 раза – 250 МГц), но это позволяет существенно увеличить производительность только для последовательной и смешанной реализации. Для параллельной реализации производительность прежде всего зависит от времени работы блока анализа пакетов, время работы которого зависит от частоты приема данных (частота фиксирована и ее нельзя увеличить для 1000 Мбит/с) и не зависит от частоты работы устройства. Увеличение частоты работы

устройства (по сравнению с частотой приема данных) дает выигрыш по времени (а следовательно и по производительности) для блока выработки решения, но поскольку для параллельной реализации блок работает быстро, то и выигрыша практически не получается.

В таблице 9.3 приведен анализ производительности для каждой реализации при максимальной нагрузке устройства (обработке минимального размера кадра):

Таблица 9.3

Реализация	Тактовая частота, МГц	10 Мбит/с	100 Мбит/с	1000 Мбит/с
Параллельная	125	2.31M	2.31M	2.31M
	250	2,4M	2,4M	2,4M
Последовательная	125	22K	22K	22K
	250	43K	43K	43K
Смешанная	125		>148K	>1.48M
	250			>1.48M
Максимальная скорость передачи, кадры/с		14.8K	148K	1.48M

В таблице зеленым цветом выделены рекомендуемые реализации, желтым – неоптимальные реализации, красным – реализации, несоответствующие по производительности (скорость обработки пакетов меньше, чем скорость их поступления). Для скорости 1000 Мбит/с

рекомендуется использовать параллельную или смешанную реализацию (степень распараллеливания должна быть достаточна, чтобы обработать 1.48М кадров/с).

Производительность параллельной реализации (худший случай, соответствующий максимально долгой работы блока анализа пакетов) равна 1.32М кадров/с (и более).

Оптимальной по критериям аппаратные затраты/производительность является смешанная реализация (которая реализована и тестирование проводилось на ней).

В реализованной смешанной реализации компоненты FDU выработки решения для одного поля в правиле реализованы параллельно, блоки проверки правил – не полностью распараллелены.

На рисунке 9.1 приведен график зависимости количества тактов работы блока выработки решения (по которым вычисляется производительность) от количества блоков проверки правил (степень распараллеливания). Блоки проверки правил реализованы парами (за 1 такт проверяют 2 правила), для параллельной реализации нужно 128 блока.

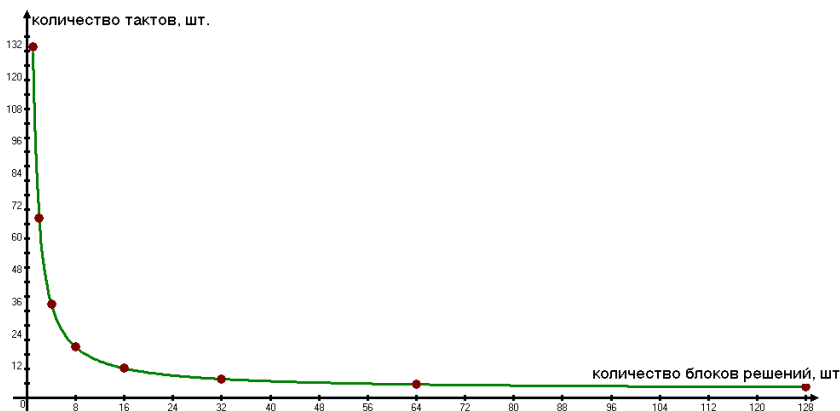


Рисунок 9.1 Зависимость количества тактов блока выработки решения от количества блоков проверки правил

Как видно из графика, небольшое увеличение блоков решений ведет к большому выигрышу в количестве тактов (линейная зависимость). Для того, чтобы производительности смешанной реализации хватало для скорости 1000Мбит/с требуется 8 блоков решений для правил (рассчитано по процедуре расчета производительности).

Аппаратные ресурсы платы-прототипа: общее число логических элементов – 41910, блоков памяти – 5662720.

В таблице 9.4 приведены аппаратные затраты на каждый разработанный блок и временные затраты (количество тактов их работы).

Таблица 9.4

Блок	Аппаратные затраты		Временные затраты, такты
	логич. элем	блоки памяти	
Анализа пакета	1637 (3,9%)	-	15-90
Выработки решения для двух правил	1542 (3,9%)	-	2
Выработка общего решения	94 (0,2%)	-	2
Интерфейсные	20 (0,04%)	-	1
Хранения правил	-	245760 (4%)	1
Буферизации сетевых пакетов	39 (0,09%)	8192 (0,1%)	1

ЗАКЛЮЧЕНИЕ

В результате проделанной работы все задачи исследовательской работы решены:

- Проведен анализ подходов к решению задачи фильтрации сетевых пакетов, элементной базы для аппаратной реализации межсетевых фильтров, средств проектирования.
- Разработана архитектура системы фильтрации пакетов и выбрана плата-прототип для аппаратного тестирования создаваемого межсетевого экрана.
- Разработана структура аппаратно-реализуемого межсетевого экрана.
- Разработан формата хранения правил фильтрации.
- Проведен анализ вариантов аппаратной реализации межсетевого экрана на СБИС программируемой логики при различных требованиях к быстродействию и ограничениях к аппаратным затратам.
- Разработано описание модулей и устройства фильтрации пакетов на языке Verilog.
- Разработаны тесты для модулей проекта и проведено моделирование и отладка проекта.
- Разработан формат задания правил фильтрации, процедура и программные средства их подготовки для записи в память СБИС ПЛ.
- Осуществлен синтез и реализация устройства фильтрации сетевых пакетов на базе СБИС ПЛ, проведен анализ быстродействия.
- Разработана процедура проведения тестирования и осуществлена отладка проекта на плате-прототипе.

Цель работы – исследование возможностей СБИС программируемой логики для аппаратной реализации межсетевого экрана, работающем на сетевом и транспортном уровнях, разработка его аппаратной реализации, исследование возможности создания собственного аппаратно-реализуемого устройства фильтрации, а также его моделирование и тестирование – достигнута.

Созданное устройство фильтрации может служить основой для разработки более сложных межсетевых экранов или использоваться в дополнение к другим межсетевым экранам.











СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Влад Максимов «Межсетевые экраны. Способы организации защиты», 2003 г. [электронный ресурс] URL: <http://compress.ru/article.aspx?id=10145> (дата обращения 10.10.2016).
2. «Межсетевой экран» [электронный ресурс] URL: https://ru.wikipedia.org/wiki/Межсетевой_экран (дата обращения 10.10.2016).
3. Техническое задание ССПЛ ТЗ 18.11.16.
4. НОУ «ИНТУИТ» Лекция 2 «Механизмы защиты информации» [электронный ресурс] URL: <http://www.intuit.ru/studies/courses/16655/1300/lecture/25505?page=5> (дата обращения 06.11.2016).
5. «Современные методы и средства сетевой защиты» [электронный ресурс] URL: http://www.lghost.ru/lib/security/kurs6/theme03_chapter01.htm (дата обращения 06.11.2016).
6. Официальный сайт компании Altera, продукты компании и их характеристики. [электронный ресурс] URL: <https://www.altera.com/products.html> (дата обращения 02.02.2017).
7. Официальный сайт компании Terasic, описание SoKIT. [электронный ресурс] URL: <http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=167&No=816&PartNo=2> (дата обращения 02.02.2017).
8. «Сети на витой паре от Ethernet до 100 Gigabit Ethernet» [электронный ресурс] URL: http://ru.gecid.com/netlan/seti_na_vitoyi_pare_ot_ethernet_do_100_gigabit_ethernet/?s=all (дата обращения 14.02.2017).
9. LogiCore IP Product Guide «GMII to RGMII v3.0», 2014.
10. «Формат JSON». [электронный ресурс] URL: <https://ru.wikipedia.org/wiki/JSON> (дата обращения 16.03.2017).
11. « Назначение RTL Viewer в Quartus II» [электронный ресурс] URL: <http://fpga.in.ua/fpga/cad-pld/faq-naznachenie-rtl-viewer-v-quartus-ii.html> (дата обращения 11.03.2017).

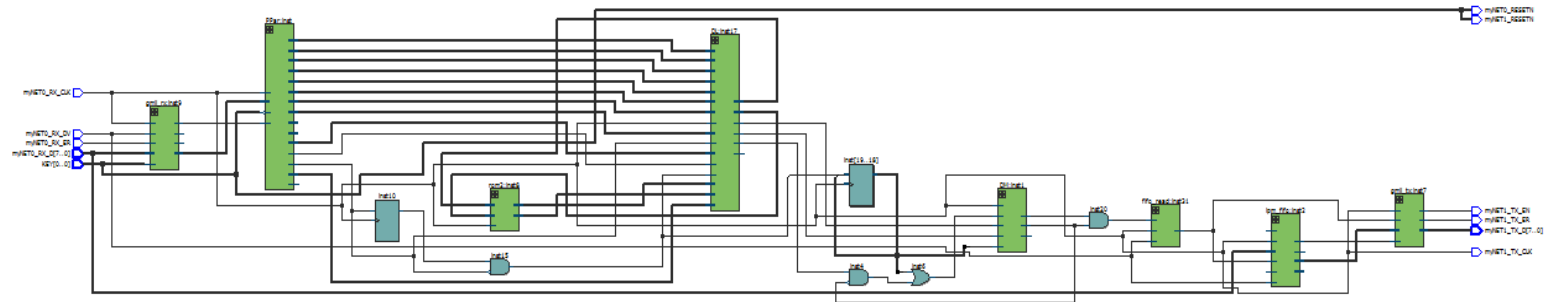
ПРИЛОЖЕНИЕ А

ИЛЛЮСТРАЦИИ

ПА1. Структура компонентов проекта и аппаратные затраты на каждый блок

Entity	Combinational ALUTs	Dedicated Logic Registers	I/O Registers	Block Memory Bits
Cyclone V: 5CSXFC6D6F31C8ES				
 firewall 	3376 (3)	1929 (3)	0 (0)	253952
 PPar:inst	1637 (1637)	1490 (1490)	0 (0)	0
 DM:inst1	94 (94)	311 (311)	0 (0)	0
▷  lpm_fifo:inst2	39 (0)	32 (0)	0 (0)	8192
 gmii_tx:inst7	1 (1)	1 (1)	0 (0)	0
▷  rom2:inst8	0 (0)	0 (0)	0 (0)	245760
 gmii_rx:inst9	19 (19)	24 (24)	0 (0)	0
 DL:inst17	1542 (1542)	34 (34)	0 (0)	0
 fifo_read:inst31	41 (41)	34 (34)	0 (0)	0

ПА2. RTL Viewer проекта



ПРИЛОЖЕНИЕ Б ТЕКСТЫ ПРОГРАММ

ПБ.1 Текст программы блока взаимодействия с внешней Ethernet (приемная часть)

```
module gmii_rx(
    input    reset,
    input    clk,
    input [7:0] gmii_rxd,
    input    gmii_rx_dv,
    input    gmii_rx_err,

    // output module
    output reg BeginPacket,
    output reg oEndPacket,
    output reg [7:0] oPacketData,
    output reg dataPacketReady
);

parameter State_idle=4'h00;//Состояние простоя. Ожидание принимаемых
данных.
parameter State_preamble=4'h01;//Синхронизация по преамбуле.
parameter State_SFD=4'h02;//Принят октет — разделитель данных.
parameter State_data=4'h03;//Состояние приема данных пакета.
parameter State_drop=4'h04;//Прекращение анализа текущего кадра.
parameter State_ErrEnd=4'h05;//Ошибка при приеме данных.
parameter State_IFG=4'h06;//Межкадровый интервал

reg [7:0] state;
reg [7:0] rxd;
reg oBeginPacket;
reg rDataValid;

//delay 1 clk
always @(posedge clk)
begin
    BeginPacket<=oBeginPacket;
    rxd<=gmii_rxd;
end
```

```

always @(posedge clk or negedge reset)
begin
if (!reset) begin
    state<=State_idle;
    oEndPacket<=1'b0;
    oEndPacket<=1'b0;
end
else begin
    case (state)
        State_idle:
            begin
                if ((gmii_rx_dv)&&(rxd == 8'h55)) begin
                    oPacketData<=8'h0;
                    state <= State_preamble;
                end
            end
        else begin
            oPacketData<=8'h0;
            state <= State_idle;
        end
    end
    State_preamble:
        begin
            oPacketData<=8'h0;
            if (!gmii_rx_dv)
                state <= State_ErrEnd;
            else if (gmii_rx_err)
                state <= State_drop;
            else if (rxd == 8'hd5) begin
                oBeginPacket <= 1'b1;
                state <= State_data;
            end
            else if (rxd == 8'h55)
                state <= State_preamble;
            else
                state <= State_drop;
        end
    State_data:
        if (!gmii_rx_dv) begin
            oEndPacket<=1'b1;
            oPacketData<=8'h0;
            state <= State_ErrEnd;
        end
    end
end

```

```

        dataPacketReady<=1'b0;
    end
    else if (gmii_rx_err) begin
        oEndPacket<=1'b1;
        oPacketData<=8'h0;
        state <= State_drop;
        dataPacketReady<=1'b0;
    end
    else begin
        oBeginPacket <= 1'b0;
        dataPacketReady <= 1'b1;
        oPacketData <= rxd;
        state <= State_data;
    end

State_drop: begin
    state = State_IFG;
    oPacketData<=8'h0;
    dataPacketReady<=1'b0;
end

State_ErrEnd: begin
    state = State_IFG;
    oPacketData<=8'h0;
    dataPacketReady<=1'b0;
end

State_IFG: begin
    state <= State_idle;
    oBeginPacket <= 1'b0;
    oEndPacket<=1'b0;
    dataPacketReady<=1'b0;
end

default: begin
    state <= State_idle;
    oBeginPacket<= 1'b0;
    oEndPacket<=1'b0;
    oPacketData<=8'h0;
    dataPacketReady<=1'b0;
end
end
endcase
end

```

```
end  
endmodule
```

ПБ.2 Текст программы теста блока взаимодействия с внешней Ethernet (приемная часть)

```
`timescale 1 ps/ 1 ps  
module tb_gmii_rx();  
parameter N=200;//size packet  
reg clk;  
reg gmii_rx_dv;  
reg gmii_rx_err;  
reg [7:0] gmii_rxd;  
reg reset;  
// wires  
wire BeginPacket;  
wire dataPacketReady;  
wire oEndPacket;  
wire [7:0] oPacketData;  
  
// assign statements (if any)  
gmii_rx i1 (  
// port map - connection between master ports and signals/registers  
    .BeginPacket(BeginPacket),  
    .clk(clk),  
    .dataPacketReady(dataPacketReady),  
    .gmii_rx_dv(gmii_rx_dv),  
    .gmii_rx_err(gmii_rx_err),  
    .gmii_rxd(gmii_rxd),  
    .oEndPacket(oEndPacket),  
    .oPacketData(oPacketData),  
    .reset(reset)  
);  
reg [7:0] array[0:N];  
reg [7:0] array_4 [0:N];  
integer data_f, data_f4;  
integer i,cnt,cnt4;  
  
initial begin  
    clk=1'b0;  
    cnt=1;
```

```

cnt4=0;
forever #5 clk=!clk;
end

initial begin
data_f = $fopen("packet.txt", "r");
$readmemh ("packet.txt", array);

for (i=0; i<N; i=i+1) begin
    $display("array_f=%0h",array[i]);
end
end

always @(posedge clk)
begin
cnt <= cnt + 1;
if (cnt<122) begin
    gmii_rxd <= array[cnt];
end
else begin
    cnt <= cnt + 1;
    gmii_rxd <= array[cnt];
    $display("d=%0h",gmii_rxd);
end
end

initial begin
$display("Running testbench");
gmii_rx_dv <= 1'b0;
gmii_rx_err <= 1'b0;
gmii_rxd <= array[0];

#5; gmii_rx_dv = 1'b1;
#1290; gmii_rx_dv = 1'b0;
#20; gmii_rx_err = 1'b1;

#50;
$fclose(data_f);
$display("Simulation ended succesfully");
$stop;
end
endmodule

```

ПБ.3 Текст программы модуля PPar

```
module PPar
#(parameter N=200)//size packet
(
input [7:0] d,
input clk,
input reset,
input strobe,
output reg ready,//additional output
output reg [1:0] Frame,
output reg [47:0] Dstmac,
output reg [47:0] Srcmac,
output reg vlan, //for test
output reg [15:0] Ethproto,
output reg [7:0] Ipproto,
output reg [31:0] srcip4,
output reg [31:0] dstip4,
output reg [1:0] fragment_flag,// 2
output reg [12:0] fragment_shift,// + 13
output reg [15:0] Srcport,
output reg [15:0] Dstport,
output reg [15:0] icmp,
output reg isFragment
);
//necessary
reg [7:0] packet [N:0];
reg [3:0] ip_header_length;
reg [5:0] ip_shift;
reg ready0,ready1,ready2;
integer i, index, vlan_shift;
//counter
//register
always @(posedge clk or posedge reset)
begin
if (reset) begin
ready0 <= 1'b0;
index=0;
for (i=0;i<=N; i=i+1)
packet[i]<=0;
end
else if (strobe) begin
```

```

ready0 <= 1'b0;
for (i=0;i<=N; i=i+1) begin
    packet[i]<=0;
end
packet[0] <= d;
index = 0;
end
else begin
    index = index+1;
    packet[index] <= d;
    /*if (index==N-1)
        index=index;
    for (i=1;i<=N;i=i+1)
        packet[i] <= packet[i-1];
    packet[0] <= d;*/
end
end
//Frame Dstmac Srcmac
always @(posedge clk)
begin
    if (strobe) begin
        ready1<=1'b0;
    end
    if (index==12) begin
        Dstmac<={packet[0],packet[1],packet[2],packet[3],packet[4],packet[5]};
        Srcmac<={packet[6],packet[7],packet[8],packet[9],packet[10],packet[11]};
    end
    else if (index == 15) begin
        //Type Frame
        if ({packet[12],packet[13]} > 16'h05DC) begin
            //EthII
            Frame <= 2'b00;
            //VLAN
            if ({packet[12],packet[13]}==16'h8100)begin
                vlan <= 1'b1;
                vlan_shift = 4;//4 bytes shift
            end
            else begin
                vlan <= 1'b0;
                vlan_shift = 0;
            end
        end
    end
end
end

```



```

//Type Frame
else if ({packet[14],packet[15]}==16'hFFFF) begin
    //802.3/Novell
    Frame <= 2'b01;
    ready1 <= 1'b1;
end
else if ({packet[14],packet[15]}==16'hAAAA) begin
    //SNAP
    Frame <= 2'b10;
    ready1 <= 1'b1;
end
else begin
    //802.3/LLC
    Frame <= 2'b11;
    ready1 <= 1'b1;
end
end

end

//Ethproto Ipproto fragment_flag fragment_shift srcip4 dstip4
always @(posedge clk)
begin
if (strobe) begin
    ready2 <= 1'b0;
end
ready <= ready0 | ready1 | ready2;//net

if (index == 25+vlan_shift)begin //22+vlan_shft
    Ethproto <= {packet[12+vlan_shift],packet[13+vlan_shift]};
    Ipproto <= packet[23+vlan_shift];
end
//no necessary fields => ready
if ((index == 26+vlan_shift) & (Ethproto !== 16'h0800) & ((Ipproto !== 8'h06) |
(Ipproto !== 8'h11) | (Ipproto !== 8'h01))) begin
    ready2<=1'b1;
end
//if ip
if (Ethproto == 16'h0800) begin
    ip_header_length <= packet[14+vlan_shift][3:0];
    ip_shift <= (ip_header_length - 4'b0101);
    fragment_flag<= packet[20+vlan_shift][6:5];
end
end

```

```

if ((Ethproto == 16'h0800) & (fragment_flag==2'b00)|(fragment_flag==2'b01))
begin
    fragment_shift<={packet[20+vlan_shift][4:0],packet[21+vlan_shift]};
    end
//isFragment
if (index == 27 + vlan_shift)begin
    if (((fragment_flag==2'b00)&(fragment_shift != 13'h0)) |
(fragment_flag==2'b01)) begin
        isFragment <= 1'b1;
        end
    else isFragment <= 1'b0;
end

//if ip - TCP or UDP or ICMP
if ((index==34+vlan_shift)&(Ethproto==16'h0800)) begin

    srcip4<={packet[26+vlan_shift],packet[27+vlan_shift],packet[28+vlan_shift]
,packet[29+vlan_shift]};

    dstip4<={packet[30+vlan_shift],packet[31+vlan_shift],packet[32+vlan_shift]
,packet[33+vlan_shift]};
    end
if ((index==38+vlan_shift+ip_shift) & ((Ipproto==8'h06) | (Ipproto==8'h11)))
begin
    Srcport<={packet[34+vlan_shift+ip_shift],packet[35+vlan_shift+ip_shift]};
    Dstport<={packet[36+vlan_shift+ip_shift],packet[37+vlan_shift+ip_shift]};
    ready2<=1'b1;
    end
if ((index==35+vlan_shift+ip_shift)&(Ipproto==8'h01)) begin
    icmp<={packet[34+vlan_shift+ip_shift],packet[35+vlan_shift+ip_shift]};
    ready2<=1'b1;
    end
end
endmodule

```

ПБ.4 Текст программы теста модуля PPar

```

`timescale 1 ns/ 100ps
module tb8_firewall;
parameter N=200;//size packet
// test vector input registers

```

```

reg clk;
reg [7:0] d;
reg reset;
reg strobe;
// wires
wire [47:0] Dstmac;
wire [15:0] Dstport;
wire [15:0] Ethproto;
wire [1:0] Frame;
wire [7:0] Ipproto;
wire [47:0] Srcmac;
wire [15:0] Srcport;
wire [31:0] dstip4;
wire [1:0] fragment_flag;
wire [12:0] fragment_shift;
wire [15:0] icmp;
wire isFragment;
wire ready;
wire [31:0] srcip4;
wire vlan;

// assign statements (if any)
PPar i1 (
// port map - connection between master ports and signals/registers
.Dstmac(Dstmac),
.Dstport(Dstport),
.Ethproto(Ethproto),
.Frame(Frame),
.Ipproto(Ipproto),
.Srcmac(Srcmac),
.Srcport(Srcport),
.clk(clk),
.d(d),
.dstip4(dstip4),
.fragment_flag(fragment_flag),
.fragment_shift(fragment_shift),
.icmp(icmp),
.ready(ready),
.reset(reset),
.srcip4(srcip4),
.strobe(strobe),
.isFragment(isFragment),

```

```

        .vlan(vlan)
    );
    reg [7:0] array_3 [0:N];
    reg [7:0] array_4 [0:N];
    integer data_f, data_f4;
    integer i,cnt,cnt4;

    initial begin
        clk=1'b0;
        cnt=1;
        cnt4=0;
        forever #5 clk=!clk;
    end

    initial begin
        data_f = $fopen("packet.txt", "r");
        data_f4 = $fopen("packet4.txt", "r");
        //if (!data_f) $stop;

        $readmemh ("packet3.txt", array_3);
        $readmemh ("packet4.txt", array_4);

        for (i=0; i<N; i=i+1) begin
            $display("array_f=%0h",array_4[i]);
        end
    end

    always @(posedge clk)
    begin
        cnt <= cnt + 1;
        if (cnt<122) begin
            d <= array_3[cnt];
        end
        else begin
            cnt4 <= cnt4 + 1;
            d <= array_4[cnt4];
            $display("d=%0h",d);
        end
    end

    initial begin
        $display("Running testbench");
    end

```

```

reset = 1'b0;
d <= array_3[0];
strobe = 1'b1;
//#5; strobe = 1'b1;
#10; strobe = 1'b0;

#1210; strobe = 1'b1;
#10; strobe = 1'b0;
//#20; strobe = 1;
//#30; strobe = 0;

#3000;
$fclose(data_f);
$display("Simulation ended succesfully");
$stop;
end
endmodule

```

ПБ.5 Текст программы модуля DL

```

module DL
(
input clk,
input ena,
//from rules
input [970:0] rule1,
input [970:0] rule2,
//fields packet from PPar
input [1:0] Frame,
input [47:0] Dstmac,
input [47:0] Srcmac,
input [15:0] Ethproto,
input [7:0] Ipproto,
input [31:0] srcip4,
input [31:0] dstip4,
input isFragment,
input [15:0] Srcport,
input [15:0] Dstport,
input [15:0] icmp,
output reg [7:0] addr1,

```

```

output reg [7:0] addr2,
output reg isAccept1,
output reg isAccept2,
output reg ready,
//output reg nextAdressExist,
output reg start
//output reg [7:0] number1
);

```

```

//1st rule
reg Frame1_accept;
reg Dstmac1_accept;
reg Srcmac1_accept;
reg Ethproto1_accept;
reg Ipproto1_accept;
reg srcip41_accept;
reg dstip41_accept;
reg isFragment1_accept;
reg Srcport1_accept;
reg Dstport1_accept;
reg icmp1_accept;

```

```

//2nd rule
reg Frame2_accept;
reg Dstmac2_accept;
reg Srcmac2_accept;
reg Ethproto2_accept;
reg Ipproto2_accept;
reg srcip42_accept;
reg dstip42_accept;
reg isFragment2_accept;
reg Srcport2_accept;
reg Dstport2_accept;
reg icmp2_accept;

```

```

reg ready_accept; //to sync logic

```

```

//attributes packet
//wire [7:0] number1 = rule1[970:963];
wire activity1 = rule1[962];
wire action1 = rule1[961];
wire [1:0] direction1 = rule1[960:959];
wire registration1 = rule1[958];

```

```

wire [1:0] Frame1_code = rule1[957:956];
wire [1:0] Dstmac1_code = rule1[947:946];
wire [1:0] Srcmac1_code = rule1[753:752];
wire [1:0] Ethproto1_code = rule1[559:558];
wire [1:0] Ipproto1_code = rule1[493:492];
wire [1:0] srcip41_code = rule1[459:458];
wire [1:0] dstip41_code = rule1[329:328];
wire [1:0] isFragment1_rule = rule1[199:198]; // "10" bit-any, 198: "1" -
fragment, "0" - dont fragment, "11" - impossible
wire [1:0] Srcport1_code = rule1[197:196];
wire [1:0] Dstport1_code = rule1[131:130];
wire [1:0] icmp1_code = rule1[65:64];

```

//2nd rule

```

//wire [7:0] number2 = rule1[970:963];
wire activity2 = rule2[962];
wire action2 = rule2[961];
wire [1:0] direction2 = rule2[960:959];
wire registration2 = rule2[958];
wire [1:0] Frame2_code = rule2[957:956];
wire [1:0] Dstmac2_code = rule2[947:946];
wire [1:0] Srcmac2_code = rule2[753:752];
wire [1:0] Ethproto2_code = rule2[559:558];
wire [1:0] Ipproto2_code = rule2[493:492];
wire [1:0] srcip42_code = rule2[459:458];
wire [1:0] dstip42_code = rule2[329:328];
wire [1:0] isFragment2_rule = rule2[199:198]; //199 bit-any, 198: "1" -
fragment, "0" - dont fragment, "11" - impossible
wire [1:0] Srcport2_code = rule2[197:196];
wire [1:0] Dstport2_code = rule2[131:130];
wire [1:0] icmp2_code = rule2[65:64];

```

initial begin

addr1 = 8'h0;

addr2 = 8'h1;

ready = 1'b0;

end

always@(posedge clk)

begin

if (ena) begin

ready_accept <= 1'b1;

```

case (Frame1_code)
2'b00: Frame1_accept <= 1'b1;
2'b01: Frame1_accept <= (Frame==rule1[949:948]) ? 1'b1 :1'b0;
2'b10: Frame1_accept <=
((Frame==rule1[949:948])|(Frame==rule1[951:950])|(Frame==rule1[953:952])|(
Frame==rule1[955:954])) ? 1'b1 :1'b0;
2'b11: Frame1_accept <=
((Frame>=rule1[949:948])|(Frame<=rule1[951:950])) ? 1'b1 :1'b0;
endcase

```

```

case (Dstmac1_code)
2'b00: Dstmac1_accept <= 1'b1;
2'b01: Dstmac1_accept <= (Dstmac==rule1[801:754]) ? 1'b1 :1'b0;
2'b10: Dstmac1_accept <=
((Dstmac==rule1[801:754])|(Dstmac==rule1[849:802])|(Dstmac==rule1[897:85
0])|(Dstmac==rule1[945:898])) ? 1'b1 :1'b0;
2'b11: Dstmac1_accept <=
((Dstmac>=rule1[801:754])|(Dstmac<=rule1[849:802])) ? 1'b1 :1'b0;
endcase

```

```

case (Srcmac1_code)
2'b00: Srcmac1_accept <= 1'b1;
2'b01: Srcmac1_accept <= (Srcmac==rule1[607:560]) ? 1'b1 :1'b0;
2'b10: Srcmac1_accept <=
((Srcmac==rule1[607:560])|(Srcmac==rule1[655:608])|(Srcmac==rule1[703:656
])|(Srcmac==rule1[751:704])) ? 1'b1 :1'b0;
2'b11: Srcmac1_accept <=
((Srcmac>=rule1[607:560])|(Srcmac<=rule1[655:608])) ? 1'b1 :1'b0;
endcase

```

```

case (Ethproto1_code)
2'b00: Ethproto1_accept <= 1'b1;
2'b01: Ethproto1_accept <= (Ethproto==rule1[509:494]) ? 1'b1 :1'b0;
2'b10: Ethproto1_accept <=
((Ethproto==rule1[509:494])|(Ethproto==rule1[525:510])|(Ethproto==rule1[541:
526])|(Ethproto==rule1[557:542])) ? 1'b1 :1'b0;
2'b11: Ethproto1_accept <=
((Ethproto>=rule1[509:494])|(Ethproto<=rule1[525:510])) ? 1'b1 :1'b0;
endcase

```

```

case (Ipproto1_code)
2'b00: Ipproto1_accept <= 1'b1;

```



```

2'b01: Ipproto1_accept <= (Ipproto==rule1[467:460]) ? 1'b1 :1'b0;
2'b10: Ipproto1_accept <=
((Ipproto==rule1[467:460])|(Ipproto==rule1[475:468])|(Ipproto==rule1[483:476
])|(Ipproto==rule1[491:484])) ? 1'b1 :1'b0;
2'b11: Ipproto1_accept <=
((Ipproto>=rule1[467:460])|(Ipproto<=rule1[475:468])) ? 1'b1 :1'b0;
endcase

```

```

case (srcip41_code)
2'b00: srcip41_accept <= 1'b1;
2'b01: srcip41_accept <= (srcip4==rule1[361:330]) ? 1'b1 :1'b0;
2'b10: srcip41_accept <=
((srcip4==rule1[361:330])|(srcip4==rule1[393:362])|(srcip4==rule1[425:394])|(s
rcip4==rule1[457:426])) ? 1'b1 :1'b0;
2'b11: srcip41_accept <=
((srcip4>=rule1[361:330])|(srcip4<=rule1[393:362])) ? 1'b1 :1'b0;
endcase

```

```

case (dstip41_code)
2'b00: dstip41_accept <= 1'b1;
2'b01: dstip41_accept <= (dstip4==rule1[231:200]) ? 1'b1 :1'b0;
2'b10: dstip41_accept <=
((dstip4==rule1[231:200])|(dstip4==rule1[263:232])|(dstip4==rule1[295:264])|(d
stip4==rule1[327:296])) ? 1'b1 :1'b0;
2'b11: dstip41_accept <=
((dstip4>=rule1[231:200])|(dstip4<=rule1[263:232])) ? 1'b1 :1'b0;
endcase

```

```

if (isFragment1_rule == 2'b00) begin //"11" - impossible
    isFragment1_accept <= 1'b1;
end
else isFragment1_accept <= (rule1[198] == isFragment) ? 1'b1 : 1'b0; //[198]
- 2 bit isFragment1_rule

```

```

case (Srcport1_code)
2'b00: Srcport1_accept <= 1'b1;
2'b01: Srcport1_accept <= (Srcport==rule1[147:132]) ? 1'b1 :1'b0;
2'b10: Srcport1_accept <=
((Srcport==rule1[147:132])|(Srcport==rule1[163:148])|(Srcport==rule1[179:164
])|(Srcport==rule1[195:180])) ? 1'b1 :1'b0;
2'b11: Srcport1_accept <=
((Srcport>=rule1[147:132])|(Srcport<=rule1[163:148])) ? 1'b1 :1'b0;

```

```

endcase

case (Dstport1_code)
2'b00: Dstport1_accept <= 1'b1;
2'b01: Dstport1_accept <= (Dstport==rule1[81:66]) ? 1'b1 :1'b0;
2'b10: Dstport1_accept <=
((Dstport==rule1[81:66])|(Dstport==rule1[97:82])|(Dstport==rule1[113:98])|(Dst
port==rule1[129:114])) ? 1'b1 :1'b0;
2'b11: Dstport1_accept <=
((Dstport>=rule1[81:66])|(Dstport<=rule1[97:82])) ? 1'b1 :1'b0;
endcase

case (icmp1_code)
2'b00: icmp1_accept <= 1'b1;
2'b01: icmp1_accept <= (Dstport==rule1[15:0]) ? 1'b1 :1'b0;
2'b10: icmp1_accept <=
((Dstport==rule1[15:0])|(Dstport==rule1[31:16])|(Dstport==rule1[47:32])|(Dstpo
rt==rule1[63:48])) ? 1'b1 :1'b0;
2'b11: icmp1_accept <= ((Dstport>=rule1[15:0])|(Dstport<=rule1[31:16])) ?
1'b1 :1'b0;
endcase
//end 1st rule

//2nd rule
case (Frame2_code)
2'b00: Frame2_accept <= 1'b1;
2'b01: Frame2_accept <= (Frame==rule2[949:948]) ? 1'b1 :1'b0;
2'b10: Frame2_accept <=
((Frame==rule2[949:948])|(Frame==rule2[951:950])|(Frame==rule2[953:952])|(
Frame==rule2[955:954])) ? 1'b1 :1'b0;
2'b11: Frame2_accept <=
((Frame>=rule2[949:948])|(Frame<=rule2[951:950])) ? 1'b1 :1'b0;
endcase

case (Dstmac2_code)
2'b00: Dstmac2_accept <= 1'b1;
2'b01: Dstmac2_accept <= (Dstmac==rule2[801:754]) ? 1'b1 :1'b0;
2'b10: Dstmac2_accept <=
((Dstmac==rule2[801:754])|(Dstmac==rule2[849:802])|(Dstmac==rule2[897:85
0])|(Dstmac==rule2[945:898])) ? 1'b1 :1'b0;
2'b11: Dstmac2_accept <=
((Dstmac>=rule2[801:754])|(Dstmac<=rule2[849:802])) ? 1'b1 :1'b0;

```

endcase

```
case (Srcmac2_code)
2'b00: Srcmac2_accept <= 1'b1;
2'b01: Srcmac2_accept <= (Srcmac==rule2[607:560]) ? 1'b1 :1'b0;
2'b10: Srcmac2_accept <=
((Srcmac==rule2[607:560])|(Srcmac==rule2[655:608])|(Srcmac==rule2[703:656
])|(Srcmac==rule2[751:704])) ? 1'b1 :1'b0;
2'b11: Srcmac2_accept <=
((Srcmac>=rule2[607:560])|(Srcmac<=rule2[655:608])) ? 1'b1 :1'b0;
endcase
```

```
case (Ethproto2_code)
2'b00: Ethproto2_accept <= 1'b1;
2'b01: Ethproto2_accept <= (Ethproto==rule2[509:494]) ? 1'b1 :1'b0;
2'b10: Ethproto2_accept <=
((Ethproto==rule2[509:494])|(Ethproto==rule2[525:510])|(Ethproto==rule2[541:
526])|(Ethproto==rule2[557:542])) ? 1'b1 :1'b0;
2'b11: Ethproto2_accept <=
((Ethproto>=rule2[509:494])|(Ethproto<=rule2[525:510])) ? 1'b1 :1'b0;
endcase
```

```
case (Ipproto2_code)
2'b00: Ipproto2_accept <= 1'b1;
2'b01: Ipproto2_accept <= (Ipproto==rule2[467:460]) ? 1'b1 :1'b0;
2'b10: Ipproto2_accept <=
((Ipproto==rule2[467:460])|(Ipproto==rule2[475:468])|(Ipproto==rule2[483:476
])|(Ipproto==rule2[491:484])) ? 1'b1 :1'b0;
2'b11: Ipproto2_accept <=
((Ipproto>=rule2[467:460])|(Ipproto<=rule2[475:468])) ? 1'b1 :1'b0;
endcase
```

```
case (srcip42_code)
2'b00: srcip42_accept <= 1'b1;
2'b01: srcip42_accept <= (srcip4==rule2[361:330]) ? 1'b1 :1'b0;
2'b10: srcip42_accept <=
((srcip4==rule2[361:330])|(srcip4==rule2[393:362])|(srcip4==rule2[425:394])|(s
rcip4==rule2[457:426])) ? 1'b1 :1'b0;
2'b11: srcip42_accept <=
((srcip4>=rule2[361:330])|(srcip4<=rule2[393:362])) ? 1'b1 :1'b0;
endcase
```

```

case (dstip42_code)
2'b00: dstip42_accept <= 1'b1;
2'b01: dstip42_accept <= (dstip4==rule2[231:200]) ? 1'b1 :1'b0;
2'b10: dstip42_accept <=
((dstip4==rule2[231:200])|(dstip4==rule2[263:232])|(dstip4==rule2[295:264])|(d
stip4==rule2[327:296])) ? 1'b1 :1'b0;
2'b11: dstip42_accept <=
((dstip4>=rule2[231:200])|(dstip4<=rule2[263:232])) ? 1'b1 :1'b0;
endcase

if (isFragment2_rule == 2'b00) begin //"11" - impossible
    isFragment2_accept <= 1'b1;
end
else isFragment2_accept <= (rule2[198] == isFragment) ? 1'b1 : 1'b0; //[198]
- 2 bit isFragment1_rule

case (Srcport2_code)
2'b00: Srcport2_accept <= 1'b1;
2'b01: Srcport2_accept <= (Srcport==rule1[147:132]) ? 1'b1 :1'b0;
2'b10: Srcport2_accept <=
((Srcport==rule1[147:132])|(Srcport==rule1[163:148])|(Srcport==rule1[179:164
])|(Srcport==rule1[195:180])) ? 1'b1 :1'b0;
2'b11: Srcport2_accept <=
((Srcport>=rule1[147:132])|(Srcport<=rule1[163:148])) ? 1'b1 :1'b0;
endcase

case (Dstport2_code)
2'b00: Dstport2_accept <= 1'b1;
2'b01: Dstport2_accept <= (Dstport==rule1[81:66]) ? 1'b1 :1'b0;
2'b10: Dstport2_accept <=
((Dstport==rule1[81:66])|(Dstport==rule1[97:82])|(Dstport==rule1[113:98])|(Dst
port==rule1[129:114])) ? 1'b1 :1'b0;
2'b11: Dstport2_accept <=
((Dstport>=rule1[81:66])|(Dstport<=rule1[97:82])) ? 1'b1 :1'b0;
endcase

case (icmp2_code)
2'b00: icmp2_accept <= 1'b1;
2'b01: icmp2_accept <= (Dstport==rule2[15:0]) ? 1'b1 :1'b0;
2'b10: icmp2_accept <=
((Dstport==rule2[15:0])|(Dstport==rule2[31:16])|(Dstport==rule2[47:32])|(Dstpo
rt==rule2[63:48])) ? 1'b1 :1'b0;

```

```

    2'b11: icmp2_accept <= ((Dstport>=rule2[15:0])|(Dstport<=rule2[31:16])) ?
1'b1 :1'b0;
    endcase
    //end 2nd rule
end
else ready_accept <= 1'b0;
end

always@(posedge clk)
begin
if (ena)
    if (ready_accept) begin
        //inputs TODO
        //if action=1 - if accept - acceptAll
        //if action=0 - if accept - dont acceptAll
        isAccept1 <= !activity1 | ((action1==1'b1) & (Frame1_accept &
Dstmac1_accept & Srcmac1_accept & Ethproto1_accept &
Ipproto1_accept & srcip41_accept & dstip41_accept &
isFragment1_accept & Srcport1_accept & Dstport1_accept & icmp1_accept)) |
        ((action1==1'b0) & !(Frame1_accept & Dstmac1_accept &
Srcmac1_accept & Ethproto1_accept &
Ipproto1_accept & srcip41_accept & dstip41_accept &
isFragment1_accept & Srcport1_accept & Dstport1_accept & icmp1_accept));

        isAccept2 <= !activity2 | ((action2==1'b1) & (Frame2_accept &
Dstmac2_accept & Srcmac2_accept & Ethproto2_accept &
Ipproto2_accept & srcip42_accept & dstip42_accept &
isFragment2_accept & Srcport2_accept & Dstport2_accept & icmp2_accept))|
        ((action2==1'b0) & !(Frame2_accept & Dstmac2_accept &
Srcmac2_accept & Ethproto2_accept &
Ipproto2_accept & srcip42_accept & dstip42_accept &
isFragment2_accept & Srcport2_accept & Dstport2_accept & icmp2_accept));

        //number1 <= rule1[970:963];
        ready <= 1'b1;
        if (addr1 < 253) begin
            addr1 <= addr1 + 2'b10;
            addr2 <= addr2 + 2'b10;
            //nextAddressExist <= 1'b1;
            start <= 1'b0;
        end
    else begin

```

```

        //nextAddressExist <= 1'b0;
        start <= 1'b1;
        addr1 <= 8'b0;
        addr2 <= 8'b1;
    end
end
else ready <= 1'b0;
else ready <= 1'b0;
end
endmodule

```

ПБ.6 Текст программы теста модуля DL

```

`timescale 1 ps/ 1 ps
module tb_DL();
// test vector input registers
reg [47:0] Dstmac;
reg [15:0] Dstport;
reg [15:0] Ethproto;
reg [1:0] Frame;
reg [7:0] Ipproto;
reg [47:0] Srcmac;
reg [15:0] Srcport;
reg clk;
reg [31:0] dstip4;
reg ena;
reg [15:0] icmp;
reg isFragment;
reg [970:0] rule1;
reg [970:0] rule2;
reg [31:0] srcip4;
// wires
wire [7:0] addr1;
wire [7:0] addr2;
wire isAccept1;
wire isAccept2;
wire ready;
wire start;

```



```

Frame=2'h00;
Dstmact=48'h6021c04435fe;
Srcmac=48'h78929c93f69e;
Ethproto=16'h0800;
Ipproto=8'h06;
srcip4=32'hc0a82b22;
dstip4=32'h4d430a8c;
isFragment=1'b0;
Srcport=16'd49171;
Dstport=16'd4443;
icmp=16'h0;
$display("Running testbench");
#10; ena=1'b1;
#20; ena=1'b0;
#30;
$display("Simulation ended succesfully");
$stop;
end
endmodule

```

```

module DM
(
    input clk,
    input ena,
    input isAccept1,
    input isAccept2,
    input startRule,
    output reg isAcceptAll,
    output reg readyReg,
    output reg readyDM
);

```



```

reg [255:0] shiftReg ;
integer i;
integer size=0;

always @(posedge clk)
begin
if (ena)
    if (startRule) begin
        shiftReg <= {254'b0,isAccept1,isAccept2};
        readyReg <= 1'b0;
        size = 2;
        end
    else begin
        shiftReg <= {shiftReg[253:0],isAccept1,isAccept2};
        if (size==254) begin
            readyReg <= 1'b1;
            end
        else begin
            readyReg <= 1'b0;
            size = size + 2;
            end
        end
    end
end

always @(posedge clk)
begin
    if (readyReg) begin
        isAcceptAll <= &shiftReg[255:0];
        readyDM <= 1'b1;
        end
    else readyDM <= 1'b0;
end
endmodule

```

ПБ.8 Текст программы задания правил

```

import bitstring
from bitstring import BitArray
from intelhex import IntelHex
import json

```

```

rules = json.load(open("rules.json", "r"))

# for example 192.168.0.1
def dec_point(field):
    return bitstring.pack("uint:8, uint:8, uint:8, uint:8", *field.split("."))

def hex_to_bin(field, size):
    # любое
    if (field == ""):
        return bitstring.pack("0b00, uint:size_bit=0, uint:size_bit=0,
uint:size_bit=0, uint:size_bit=0", size_bit=size)
    # диапазон
    if ("-" in field):
        return bitstring.pack("0b11, uint:size_bit=0, uint:size_bit=0,
hex:size_bit, hex:size_bit", *field.split("-"),
size_bit=size)
    s = len(field.split(",")) # size field
    # 1 число
    if (s == 1):
        return bitstring.pack("0b01,
uint:size_bit=0, uint:size_bit=0, uint:size_bit=0, hex:size_bit",
field.split(",")[0], size_bit=size)
    # перечисление
    if ("," in field):
        if (s == 4):
            return bitstring.pack("0b10, hex:size_bit, hex:size_bit, hex:size_bit,
hex:size_bit", *field.split(","),
size_bit=size)
        if (s == 3):
            return bitstring.pack("0b10, uint:size_bit=0, hex:size_bit,
hex:size_bit, hex:size_bit", *field.split(","),
size_bit=size)
        if (s == 2):
            return bitstring.pack("0b10, uint:size_bit=0, uint:size_bit=0,
hex:size_bit, hex:size_bit",
*field.split(","), size_bit=size)

def dec_to_bin(field, size):
    # all
    if (field == ""):
        return bitstring.pack("0b00, uint:size_bit=0, uint:size_bit=0,

```

```

uint:size_bit=0, uint:size_bit=0", size_bit=size)
    # диапазон
    if ("-" in field):
        return bitstring.pack("0b11, uint:size_bit=0, uint:size_bit=0,
uint:size_bit, uint:size_bit", *field.split("-"), size_bit=size)
        s = len(field.split(",")) # size field
        # 1 число
        if (s == 1):
            return bitstring.pack("0b01, uint:size_bit=0, uint:size_bit=0,
uint:size_bit=0, uint:size_bit",
                                field.split(",")[0], size_bit=size)
        # перечисление
        if ("," in field):
            if (s == 4):
                return bitstring.pack("0b10, uint:size_bit, uint:size_bit,
uint:size_bit, uint:size_bit", *field.split(","),
                                size_bit=size)
            if (s == 3):
                return bitstring.pack("0b10, uint:size_bit=0, uint:size_bit,
uint:size_bit, uint:size_bit",
                                *field.split(","), size_bit=size)
            if (s == 2):
                return bitstring.pack("0b10, uint:size_bit=0, uint:size_bit=0,
uint:size_bit, uint:size_bit",
                                *field.split(","), size_bit=size)
            assert "DEC_TO_BIN"

def dec_with_point_to_bin(field, size):
    # all
    if (field == ""):
        return bitstring.pack("0b00, uint:size_bit=0, uint:size_bit=0,
uint:size_bit=0, uint:size_bit=0", size_bit=size)
    # диапазон
    if ("-" in field):
        return bitstring.pack("0b11, uint:size_bit=0, uint:size_bit=0",
size_bit=size) + dec_point(
        field.split("-")[1]) + dec_point(field.split("-")[0])
        s = len(field.split(",")) # size field
        # 1 число
        if (s == 1):
            return bitstring.pack("0b01, uint:size_bit=0, uint:size_bit=0,

```

```

uint:size_bit=0", size_bit=size) + dec_point(
    field.split(",")[0])
# перечисление
if ("," in field):
    if (s == 4):
        return bitstring.pack("0b10") + dec_point(field.split(",")[3]) +
dec_point(field.split(",")[2]) + dec_point(
    field.split(",")[1]) + dec_point(field.split(",")[0])
    if (s == 3):
        return bitstring.pack("0b01, uint:size_bit=0", size_bit=size) +
dec_point(field.split(",")[2]) + dec_point(
    field.split(",")[1]) + dec_point(field.split(",")[0])
    if (s == 2):
        return bitstring.pack("0b01, uint:size_bit=0, uint:size_bit=0",
size_bit=size) + dec_point(field.split(",")[1]) + dec_point(field.split(",")[0])

```

```

def to_hex(rule):
    bs = BitArray()
    number = bitstring.pack("uint:8", rule["number"])
    #print "number\n", number
    activity = bitstring.pack("uint:1", rule["activity"])
    #print "activity\n", activity
    action = bitstring.pack("uint:1", rule["action"])
    #print "action\n", action
    direction = bitstring.pack("uint:2", rule["direction"])
    #print "direction\n", direction
    registration = bitstring.pack("uint:1", rule["registration"])
    #print "registration\n", registration
    Frame = dec_to_bin(rule["Frame"], 2)
    #print "Frame\n", Frame
    Dstmac = hex_to_bin(rule["Dstmac"], 48)
    #print "Dstmac\n", Dstmac
    Srcmac = hex_to_bin(rule["Srcmac"], 48)
    #print "Srcmac\n", Srcmac
    Ethproto = hex_to_bin(rule["Ethproto"], 16)
    #print "Ethproto\n", Ethproto
    Ipproto = hex_to_bin(rule["Ipproto"], 8)
    #print "Ipproto\n", Ipproto
    srcip4 = dec_with_point_to_bin(rule["srcip4"], 32)
    #print "srcip4\n", srcip4
    dstip4 = dec_with_point_to_bin(rule["dstip4"], 32)

```

```

##print "dstip4\n", dstip4
Srcport = dec_to_bin(rule["Srcport"], 16)
#print "Srcport\n", Srcport
Dstport = dec_to_bin(rule["Dstport"], 16)
#print "Dstport\n", Dstport
isFragment = bitstring.pack("uint:2", rule["isFragment"])
#print "isFragment\n", isFragment
icmp = hex_to_bin(rule["icmp"], 16)
#print "icmp\n", icmp
bs += number + activity + action + direction + registration + Frame + Dstmac
+ Srcmac + Ethproto + Ipproto + srcip4 + dstip4 + isFragment + Srcport +
Dstport + icmp
#print "bs\n", bs
assert len(bs) == 971
return bs

```

```

rules_bin = BitArray()

```

```

for x in rules:

```

```

    rule_bin = to_hex(x)
    rules_bin += rule_bin

```

```

print "len=", len(rules_bin)
ih = IntelHex()
ih.puts(0, rules_bin.tobytes())
ih.tofile("test.hex", "hex")

```