Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
«Сыктывкарский государственный университет имени Питирима Сорокина»
Институт точных наук и информационных технологий
Кафедра информационной безопасности

Допустить к защите
Зав. кафедрой информационной безопасности,
к. ф. -м. н., доцент
_____ Л. С. Носов
«____»_____ 2016 года

Курсовая работа
по дисциплине «Программно-аппаратные средства защиты информации»
Разработка аппаратного средства ЗИ на основе ПЛИС.

Научный руководитель:
преподаватель _____ А. Н. Некрасов
«____»_____ 2016 г.

Исполнитель:
Студент 143 группы _____ Е. Ю. Пипуныров
«____»_____ 2016 г.

Сыктывкар 2016

Содержание

Определения, обозначения и сокращения

В настоящей работе применяются следующие сокращения с соответсвующими расшифровками:

1. **ЛЭ** — логические элементы.
2. **ЗИ** — защита информации.
3. **ПЛИС** — программируемые логические интегральные схемы.
4. **ЦПУ** — центральное процессорное устройство.
5. **HDL** — hardware describe language.
6. **ПО** — программное обеспечение.
7. **PLA** — programmable logic array.
8. **RISC** — reduced instruction set computer.
9. **CISC** — complex instruction set computer.
10. **ВКР** — Выпускная квалификационная работа.

В настоящей работе применяются следующие термины с соответствующими определениями:

1. **Архитектура ЦПУ**— набор типов данных, операций и регистров, доступных для программирования.
2. **Микроархитектура ЦПУ** — способ, которым данная архитектура реализована в процессоре.

ВВЕДЕНИЕ

Опутывание мира сетью коммуникаций, стремительное расширение применения мобильных и облачных сервисов, а так же соц. сетей приводит к тому, что список угроз безопасности передачи данных продолжает расти. В него постоянно добавляются новые способы взлома, мошенничества, новые вредоносные программы, новые типы вирусов, новые способы перехвата данных. Такое бурное развитие угроз, в свою очередь, стимулирует массовое применение средств защиты. Самым распространённым средством защиты передаи данных является криптографический метод.

Криптографические Методы Защиты Информации (КМЗИ) серьёзно повышают защищённость данных, однако, требуют большого количества дополнительных вычислений, нагружающих аппаратуру. Помимо того, стандартные микропроцессорные устройства имеют общий, неспециализированный набор команд, использование которых снижает скорость обработки данных.

В данных условиях было решено разработать свой вариант аппаратного средства защиты информации (ЗИ) на основе программируемых логических интегральных схем(ПЛИС). Данная задача слишком велика для выполнения её в рамках курсовой работы, поэтому было принято решение разбить её на подзадачи, часть из которых выполнить в рамках данной курсовой работы, а оставшиеси — в рамках выпускной квалификационной работы (ВКР).

Актуальность данной работы определяется увеличением потоков информации в условиях постепенного снижения скорости развития аппаратных мощностей современных систем. Перемещние криптографической обработки данных в отдельные аппаратные блоки является одним из вариантов решения сложившейся ситуации. Ускоренная аппаратная криптографическая обработка вместо программного выполнения этих же алгоритмов позволяет разгрузить центральный процессор для выполнения других операций. Некоторые производители уже имеют наработки в данной области. Реализация данных технологий, однако, носит закрый пропреитарный характер. Данные технологии не имеют широкого распространения. В рамках работы будет произведена попытка расширить применяемость данной технологии посредством создания открытого аналога данной технологии и описания её на языке Verilog.

Объектом исследования курсовой работы является процесс создания процессорного ядра на основе ПЛИС и создание на его основе аппаратного средства ЗИ.

Предметами исследования являютс применимость ПЛИС для решения задач ЗИ и способы построения аппаратных средств ЗИ на основе процессоров.

Целью курсовой работы является создание процессорного ядра на основе ПЛИС для дальнейшего использования его в качестве платформы для аппаратного средства ЗИ.

Постановка задачи

Для достижения поставленной цели необходимо решить следующие задачи:

- Рассмотреть применимость ПЛИС и Hardware Description Language (HDL) для задач ЗИ.
- Изучить процессорную архитектуру MIPS.
- Рассмотреть способы построения аппаратных средств ЗИ на основе процессоров.
- Создать ядро ЦПУ, на основе которого и будет выполняться средство ЗИ.

Выбор метода реализации задач:

Основной задачей данной работы является создание ядра ЦПУ, на основе которого и будет выполняться средство ЗИ. В качестве базовой архитектуры процессора выбрана архитектура MIPS. Инструментами для решения основной задачи были выбраны язык описания аппаратуры Verilog и тестовая ПЛИС Xilinx Spartan-3AN FPGA Starter Kit Board.

1. ПЛИС И HDL. ПРИМЕНИМОСТЬ ДЛЯ РЕШЕНИЯ ЗАДАЧ ЗИ

1.1. Метод абстракций

Проектирование сложных систем невозможно без использования различного рода абстракций. Абстракция подразумевает исключение из рассмотрения тех элементов, которые в данном случае несущественны для понимания работы системы.

Современная электронная система состоит из полупроводниковых устройств, таких как транзисторы. Любое электронное устройство может быть представлено абстрактной математической моделью, описывающей изменяющуюся во времени взаимозависимость тока и напряжения.

Следующий уровень абстракции — это аналоговые схемы, в которых полупроводниковые устройства соединены таким образом, чтобы они образовывали функциональные компоненты, такие как усилители, например. Напряжение на входе и на выходе аналоговой цепи изменяется в непрерывном диапазоне.

В отличие от аналоговых цепей, цифровые схемы, такие как логические вентили, используют два строго ограниченных дискретных уровня напряжения. Один из этих дискретных уровней — это логический нуль, другой — логическая единица. Связывая вместе логические элементы мы создаём микроархитектуру.

Микроархитектурный уровень абстракции, или просто микроархитектура, связывает логический и архитектурный уровни абстракции. Архитектурный уровень абстракции, или архитектура, описывает компьютер с точки зрения программиста.

Наконец, сегодняшние пользователи общаются с уровнем программного обеспечения (ПО). Операционная система (ОС) управляет операциями нижнего уровня, такими как доступ к жесткому диску или управление памятью. И, наконец, программное обеспечение использует ресурсы операционной системы для решения конкретных задач пользователя.

Даннаая работа подразумевает общение с тремя уровнями абстракции:

- уровнем цифровых схем,
- микроархитектурным уровнем,
- архитектурным уровнем,

На данных уровнях абстракции самыми удобными инструментами работы являются языки описания аппаратуры и ПЛИС[1].

1.2. ПЛИС

На сегодняшний день при проектировании цифровых устройств никто не использует дискретные элементы. Данный подход трудозатратен, дорогостоящ и неэффективен. Вместо этого

применяют матрицы логических элементов. Такие матрицы состоят из наборов логических элементов, соединения которых можно сконфигурировать для реализации произвольной логической функции, при этом не надо будет изменять соединения между микросхемами на плате. Регулярная структура упрощает проектирование. Матрицы логических элементов производятся в больших количествах, что обеспечивает их малую стоимость. Существует программное обеспечение, позволяющее перенести проекты цифровых устройств в такие матрицы. Большинство матриц логических элементов реконфигурируемо, что позволяет изменить проект без замены аппаратного обеспечения. Реконфигурируемость очень ценна при разработке и полезна при эксплуатации изделия, поскольку оно может быть обновлено путём простой загрузки новой конфигурации.

В основном, сегодня используются два типа матриц логических элементов:

- PLA (Programmable Logic Array) или ПЛМ (программируемая логическая матрица) — программируемый массив (матрица) логических элементов.
- FPGA (field-programmable gate array) или ПЛИС (программируемая логическая интегральная схема) — программируемая **пользователем** матрица логических **реконфигурируемых** элементов.

PLA остаются более дешёвым решением, однако, ввиду ограниченности функциональности, а так же невозможности реконфигурирования, PLA постепенно вытесняются FPGA[2].

Программируемые логические интегральные Схемы

ПЛИС представляет собой матрицу конфигурируемых логических элементов, которые также называются конфигурируемыми логическими блоками. Каждый ЛЭ можно сконфигурировать для выполнения функций некоторой комбинационной или последовательной схемы. На Рисунке1 приведена обобщённая структура ПЛИС. ЛЭ окружены элементами ввода/вывода, которые предназначены для организации обмена информацией между FPGA и прочими компонентами системы. Элементы ввода/вывода соединяют входы и выходы логических элементов с контактами корпуса микросхемы. Логические элементы могут быть соединены между собой и с элементами ввода/вывода с помощью программируемых каналов трассировки[2].

1.3. HDL

В 1990-е годы разработчики обнаружили, что их производительность труда резко возрастала, если они работали на более высоком уровне абстракции, определяя только логическую функцию и предоставляя создание оптимизированных логических элементов системе автоматического проектирования (САПР). Для этой цели стали использовать языки описания аппаратуры или Hardware Description Language (HDL). Два основных языка описания аппаратуры — Verilog и VHDL. Verilog и VHDL построены на похожих принципах, но их синтаксис весьма различается. На обоих

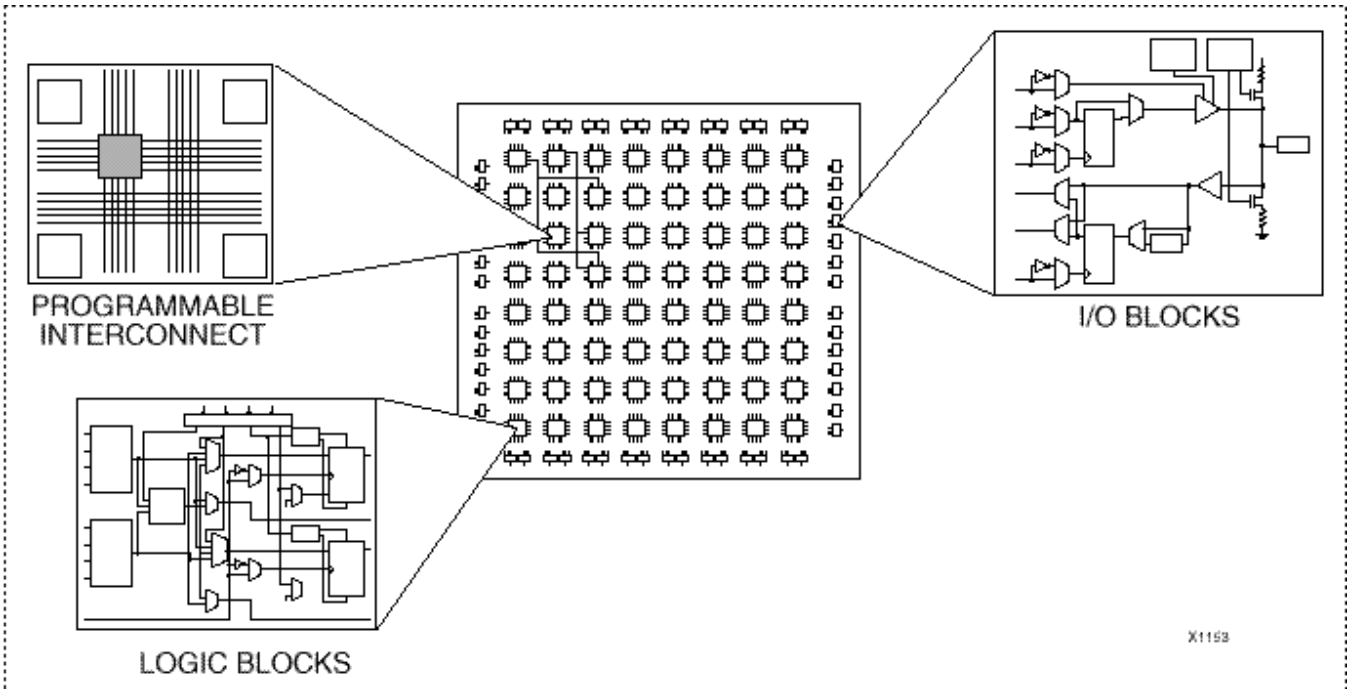

Рисунок 1— Пример структуры ПЛИС

языках можно полностью описать любую электронную систему. Две основные цели HDL — логическая симуляция и синтез[1].

Симуляция предназанчена для снижения затрат на тестирование системы. Тестирование системы в лаборатории весьма трудоёмко. Исследовать причины ошибок в лаборатории может быть очень сложно, так как наблюдать можно только сигналы, подключенные к контактам чипа, а то, что происходит внутри чипа, напрямую наблюдать невозможно. Исправление ошибок уже после того, как система была выпущена, может быть очень дорого. Например, исправление одной ошибки в новейших интегральных микросхемах стоит больше миллиона долларов и занимает несколько месяцев.

Во время симуляции на входы модуля подаются некоторые воздействия и проверяются выходы, чтобы убедиться, что модуль функционирует корректно.

Логический синтез преобразует код на HDL в нетлист, описывающий цифровую аппаратуру (т.е. логические элементы и соединяющие их проводники). Логический синтезатор может выполнять оптимизацию для сокращения количества необходимых элементов. Нетлист может быть текстовым файлом или нарисован в виде схемы, чтобы было легче визуализировать систему.

В конце концов, нетлист можно преобразовать в бинарную прошивку и загрузить её на ПЛИС.

1.4. Применимость для решения задач ЗИ

Можно выделить следующие важные для ЗИ особенности ПЛИС и HDL:

1. ПЛИС очень гибки в настройке, что позволяет использовать их для решения многих задач, в том числе, необычных.
2. При исчезновении угрозы или её изменении, ПЛИС можно реконфигурировать, чтобы испоьзовать для других возможных нужд.
3. Реконфигурацию ПЛИС можно произвести втечение очень малого времени, не приостанавливая деятельность надолго, так как используется достаточно высокий уровень абстракции в виде HDL.
4. Так как VHDL и Verilog являются стандартизованными языками, проекты, описанные на этих языках являются кроссплатформенными.
5. Современные ПЛИС оснащаются встроенными оптимизированными модулями, выполняющими различные функции, что делает их ещё более простыми и привлекательными для построения систем сложных вычислений.
6. На данный момент ПЛИС являются доступным, изящным и относительно недорогим решением широкого круга проблем.

На основании вышеизложенных преимуществ, можно сделать вывод о том, что ПЛИС и HDL являются мощными инструментами для решения широкого круга задач и в частности, задач ЗИ.

2. АРХИТЕКТУРА MIPS

В качестве основы в данной работе используется архитектура MIPS. Архитектура MIPS является простой reduced instruction set computer (RISC) архитектурой. В последующих главах будут рассмотрены основные особенности архитектуры MIPS, такие как:

- Форматы инструкций;
- Режимы адресации;
- Карта памяти;

2.1. Инструкции

В архитектуре MIPS в качестве компромисса между простотой и универсальностью используются три формата инструкций: типа R, типа I и типа J. Небольшое количество форматов обеспечивает определенное единообразие между всеми тремя типами и, как следствие, более простую аппаратную реализацию. При этом разные форматы позволяют учитывать различные потребности инструкций, как, например, необходимость хранить большие константы внутри инструкций.

2.1.1. Инструкции типа R

Название типа R является сокращением от "Register-type" — регистрового типа Инструкции типа R используют три регистра в качестве операндов: два регистра-источника и один регистр-назначение. На Рисунке 2 показан машинный формат команды типа R. 32-битная команда состоит из шести полей: `op, rs, rt, rd, shamt` и `funct`. Каждое поле состоит из пяти или шести бит[1].

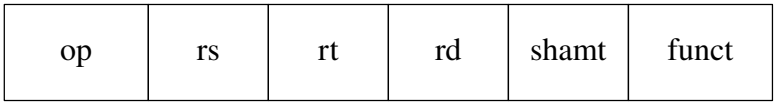

Рисунок 2 — Rtype

Операция, выполняемая командой, закодирована двумя полями: полем `op` (также называемым `opcode` или кодом операции) и полем `func` (также называемым функцией). У всех команд типа R поле `opcode` равно нулю. Операция, выполняемая этими командами, определяется исключительно полем `funct`. Например, поля `opcode` и `funct` у инструкции `add` равны 0 и 32 соответственно. Аналогично, у команды `fsub` поля `opcode` и `funct` равны 0 и 34.

Операнды закодированы тремя полями: `rs, rt` и `rd` . Поля содержат номера регистров. Пятое поле, `shamt`, является сокращением от shift amount используется только для операций сдвига. В таких командах двоичное значение, хранимое в 5-битном поле `shamt`, задаёт величину сдвига. У всех остальных команд типа R поле `shamt` равно 0.

2.1.2. Инструкции типа I

Название типа I является сокращением от immediate-type или непосредственного типа. Инструкции типа I используют в качестве операндов два регистра и один непосредственный операнд (константу).

На Рисунке 3 показан формат машинной команды типа I. 32-битная команда состоит из четырёх полей: `op`, `rs`, `rt` и `imm`. Первые три поля (`op`, `rs` и `rt`) аналогичны таким же полям в командах типа R. Поле `imm` (сокр. от immediate) содержит 16-битную константу. 16-битные константы перед использованием в операциях будут расширены до 32 бит посредством знакового расширения либо дополнения нулями[1].

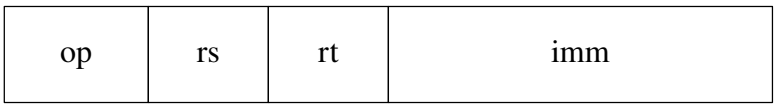

Рисунок 3 — Itype

Операция определяется исключительно полем `opcode`. Операнды заданы в трёх полях: `rs`, `rt` и `imm`. Поля `rs` и imm всегда используются как операнды-источники. Поле `rt` в некоторых командах (например, `addi` и `lw`) содержит номер регистра-назначения, в других (например, `sw`) — номер регистра-источника.

2.1.3. Инструкции типа J

Название типа J является сокращением от английского слова jump — прыжок. Этот формат используется только для инструкций безусловного перехода и ветвления. Как и другие команды, команды типа J начинаются с 6-битного поля кода операции `opcode`. Также, этот формат инструкций содержит 26-битный операнд `addr`. Константное значение `addr` используется для указания адреса перехода[1].

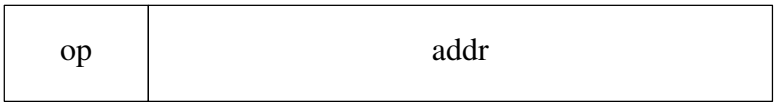

Рисунок 4 — Jtype

2.2. Режимы адресации

В архитектуре MIPS используются пять режимов адресации: регистровый, непосредственный, базовый, относительно счетчика команд и псевдопрямой. Первые три режима (регистровый, непосредственный и базовый) определяют способы чтения и записи операндов. Последние два (режим адресации относительно счетчика команд и псевдопрямой режим) определяют способы записи счётчика команд[1].

2.2.1. Регистровая адресация

При регистровой адресации регистры используются для всех операндов-источников и операндов-назначений (иными словами — для всех операндов и результата). Все инструкции типа R используют именно такой режим адресации[1].

2.2.2. Непосредственная адресация

При непосредственной адресации в качестве операндов наряду с регистрами используют 16-битные константы (непосредственные операнды). Этот режим адресации используют некоторые инструкции типа I, такие как сложение с константой `(addi)` и загрузка константы в старшие 16 бит регистра `(lui)`[1].

2.2.3. Базовая адресация

Инструкции для доступа в память, такие как загрузка слова `(lw)` и сохранение слова `(sw)`, используют базовую адресацию. Эффективный адрес операнда в памяти вычисляется путём сложения базового адреса в регистре `rs` и 16-битного смещения с расширенным знаком, являющегося непосредственным операндом[1].

2.2.4. Адресация относительно счётчика команд

Инструкции условного перехода, или ветвления, используют адресацию относительно счётчика команд для определения нового значения счётчика команд в том случае, если нужно осуществить переход. Смещение со знаком прибавляется к счётчику команд (PC) для определения нового значения PC, поэтому тот адрес, куда будет осуществлен переход, называют адресом относительно счётчика команд[1].

2.2.5. Псевдопрямая адресация

При прямой адресации адрес перехода задаётся внутри инструкции. Инструкции безусловного перехода `j` и `jal` в идеале могли бы использовать прямую адресацию для определения 32-битного целевого адреса перехода, указывающего адрес инструкции, которая будет выполнена следующей. К сожалению, в формате инструкций типа J нет достаточного количества бит для того, чтобы задать полный 32-битный адрес перехода. Шесть старших бит инструкции занимает код операции (поле `opcode`), поэтому для адреса перехода остаётся только 26 бит. К счастью, два младших бита адреса перехода всегда должны быть равны нулю, потому что все инструкции выровнены по словам. Следующие 26 бит адреса перехода берутся из поля `addr` инструкции. Четыре старших бита адреса перехода берутся из четырёх старших бит значения `PC + 4`. Такой способ адресации называется псевдопрямым[1].

2.3. Карта памяти

Так как архитектура MIPS использует 32-битные адреса, то размер адресного пространства составляет 4 гигабайта. Адреса слов кратны 4 и располагаются в промежутке от 0 до 0xFFFFFFFC. На Рисунке 5 изображена карта памяти MIPS. Адресное пространство разделено на четыре части, или сегмента: сегмент кода, сегмент глобальных данных, сегмент динамических данных и зарезервированный сегмент. Эти сегменты рассматриваются в следующих разделах.

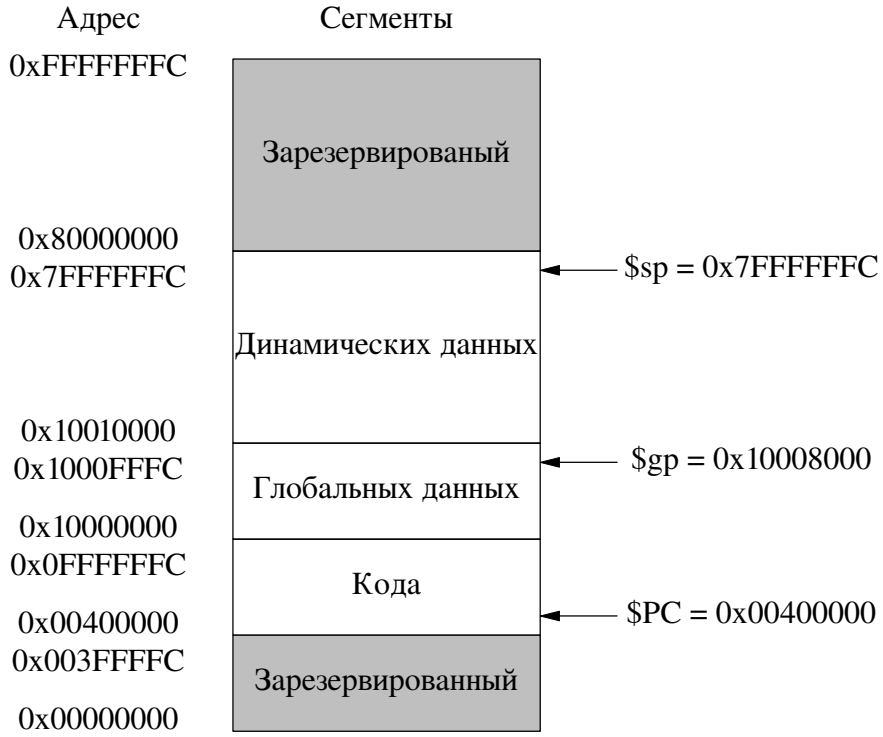

Рисунок 5 — Карта памяти MIPS

2.3.1. Сегмент кода

Сегмент кода (text segment) содержит машинные команды исполняемой программы. Его размер достаточен для размещения почти 256 Мбайт кода. Четыре старших бита адреса в сегменте кода всегда равны нулю, что позволяет использовать инструкцию `j` для перехода по любому адресу в программе[1].

2.3.2. Сегмент глобальных данных

Сегмент глобальных данных (global data segment) содержит глобальные переменные, которые, в отличие от локальных переменных, находятся в области видимости всех функций программы. Глобальные переменные инициализируются при загрузке программы, но до начала её выполнения[1].

Доступ к глобальным переменным осуществляется при помощи глобального указателя `(gp)`, который инициализируется значением 0x100080000. Любая глобальная переменная доступна при помощи 16-битного положительного или отрицательного смещения относительно `gp`.

2.3.3. Сегмент динамических данных

Сегмент динамических данных содержит стек и кучу. В момент запуска программы этот сегмент не содержит данных — они динамически выделяются и освобождаются в нём в процессе выполнения программы. Сегмент динамических данных — это самый большой сегмент памяти, в карте памяти архитектуры MIPS — его размер составляет почти 2 Гбайт.

Стек используется для сохранения и восстановления регистров, используемых функциями, а также хранения локальных переменных, таких как массивы. Стек растёт вниз от верхней границы сегмента динамических данных (0x7FFFFFFC), а доступ к кадрам стека осуществляется в режиме очереди LIFO.

Куча используетя для хранения блоков памяти, динамически выделяемых программе во время работы.

2.3.4. Зарезервированный сегмент

Зарезервированный сегмент выделяется для нужд операционной системой. В частности, зарезервированная память используется для программных прерываний и для отображения устройств ввода-вывода в адресное пространство.

2.4. Итог

Подводя итоги вышеорисанных особенностей, можно сказать, что архитектура MIPS имеет понятный, единообразный и простой, но достаточный для выполнения поставленной задачи формат инструкций. Результатом описанных особенностей является простая аппаратная реализация. Простота реализации стала определяющим фактором при выборе архитектуры.

3. МЕТОДЫ РЕАЛИЗАЦИИ И ПРИМЕНЕНИЯ СРЕДСТВА ЗИ

3.1. Методы реализации

Конечной целью является создание аппаратного средства ЗИ криптографическими методами на основе микропроцессора с MIPS-подобной архитектурой с помощью ПЛИС и HDL.

Данная цель может быть реализована двумя способами:

1. Посредством расширения встроенного тракта данных ядра;
2. Посредством создания специализированного сопроцессора для выполнения криптографических функций.

3.1.1. Расширение встроенного тракта данных ядра

Стандартный миркопроцессор содержит общий, неспециализированный набор инструкций. Написание криптографических алгоритмов с помощью стандартного набора инструкций может стать очень трудоёмкой задачей. Объём программного кода при использовании только стандартного набора команд становится очень большим при написании сложных алгоритмов, время выполнения таких алгоритмов тоже становится большим.

Стандартный набор инструкций можно расширить. Такие системы называются системами с расширенным набором команд или Complex Instruction Ser Computer(CISC). Расширение набора команд можно совершить посредством добавления дополнительных модулей в тракт данных ядра процессора. Эти модули могут быть специализированы для выполнения строго определённых функций.

В представленном случае это могут быть функции, часто используемые в криптографических алгоритмах, такие как:

- Логические операции, как XOR, XNOR, NOR и т.д;
- Операции над конечными полями;
- Перестановки бит или байт в блоке;
- Расширение блока
- и др.

Главная особенность данного метода состоит в том, что модули расширения не выходят за границы ядра, то есть не являются отдельными устройствами.

3.1.2. Создание сопроцессора

Сопроцессор — специализированный процессор, расширяющий возможности центрального процессора компьютерной системы, но оформленный как отдельный функциональный модуль. Физически сопроцессор может быть отдельной микросхемой или может быть встроен в

центральный процессор.

Сопроцессор расширяет базовый набор инструкций и регистров ЦПУ. ЦПУ и его сопроцессор соединяются общей системной шиной данных, сигнальной шиной, шиной флагов,и т.д. в набор инструкций ЦПУ могут быть встроены специальные инструкции обмена данными с сопроцессором.

Декодирование и выполнение инструкций может проходить по одному из следующих сценариев:

1. ЦПУ декодирует инструкцию и отправит соответствующую команду сопроцессору на выполнение вместе с сопутствующими данными. Сопроцессор декодирует команду и начнёт выполнение. Данные могут как передаваться в регистры сопроцессора, так и оставаться в регистрах ЦПУ, в таком случае сопроцессору будут переданы лишь номера регистров, в которых находятся данные.
2. При возникновении новой инструкции на системной шине, она может быть декодирована и ЦПУ и сопроцессором одновременно. Если инструкция является инструкцией сопроцессора, ЦПУ сразу приступает к передаче соответствующих операндов, а сопроцессор — к выполнению инструкции.

Во время выполнения инструкций сопроцессор и ЦПУ обмениваются управляющими сигналами посредством соответствующих флагов, так, например, на время выполнения может быть выставлен флаг `busy`, а на случай прерываний может быть выставлен флаг `interrupt`[2].

Сопроцессор может выполнять как отдельные операции, присущие криптоалгоритмам, так и полностью выполнять криптоалгоритмы. Важно так же отметить, что у одного ЦПУ может быть и несколько сопроцессоров, каждый из которых может быть специализирован на выполнение отдельного набора инструкций, или же их наборы инструкций могут пересекаться. В таком случае вводится ещё один вид управляющего сигнала `select`. Общая картина соединения ЦПУ и сопроцессоров показана на Рисунке 6.

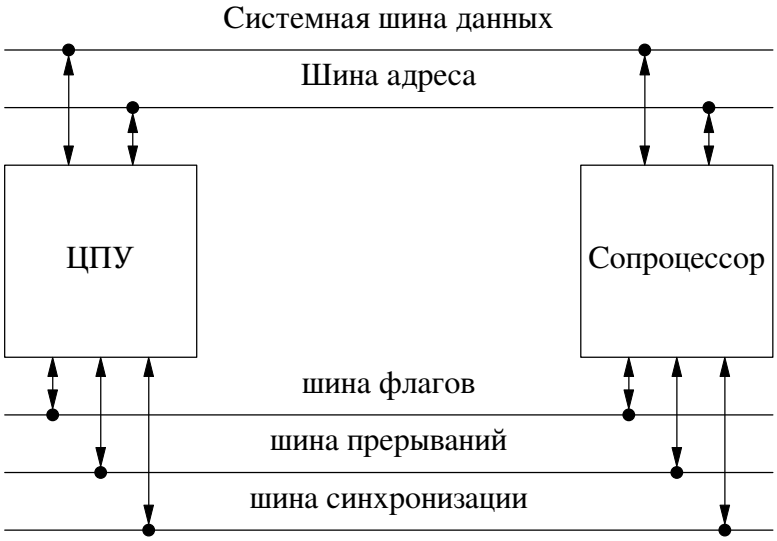

Рисунок 6 — Общая схема соединения сопроцессора

3.2. Возможности применения

Данное устройство, при снабжении его соответствующими интерфейсами, может быть использовано для решения широкого круга задач.

Так, например, на его основе может быть построен потоковый шифратор для шифрования тракта данных, выходящего за границу некой организации. Для выполнения данной задачи, необходимо разработать два устройства, одно — для шифрования исходящего трафика, другое — для дешифрования принимаемого трафика, как показано на Рисунке 7

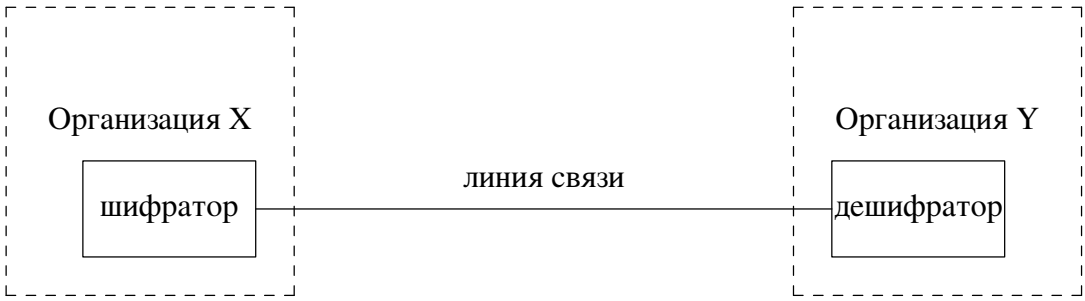

Рисунок 7 — Схема применения потокового шифратора

Другой вариант применения — в составе устройства для шифрования данных носителей информации. В случае применения такого устройства, данные носителя будут храниться в зашифрованном виде, нормальная работа с ними будет возможна лишь при наличии устройства. Подобное решение может использоваться для защиты конфиденциальности информации, хранимой на носителе.

Рисунок 8 показывает схему применения такого устройства.

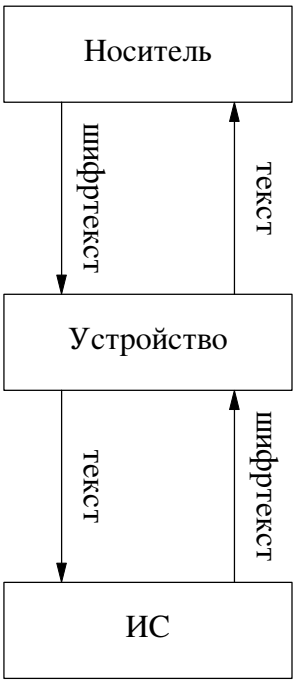

Рисунок 8 — Схема применения устройства для шифрования данных носителей информации

Стоит отметить, что в обоих случаях нагрузка по реализации ЗИ ложится на отдельное утройство, а не на систему обработки данных.

Как было отмечено ранее, данные варианты испоьзования показаны для примера, в реальности их может быть намного больше, причём для реализации нового решения, не придётся закупать полностью новое оборудование, можно лишь изменить программу или, при необходимости, изменить микроархитектуру устройства, ведь оно построено на базе ПЛИС.

4. РАЗРАБОТКА ЯДРА MIPS-ПОДОБНОГО ПРОЦЕССОРА НА ПЛИС

Глваной задачей данной работы была создать ядро ЦПУ, на основе которого и будет выполняться средство ЗИ, основной архитектурой которого была выбрана архитектура MIPS.

Компьютерная архитектура определяется набором команд и архитектурным состоянием. Архитектурное состояние процессора MIPS определяется содержимым счётчика команд (program counter) и 32 видимых программисту регистров, поэтому любой процессор, реализующий архитектуру MIPS, вне зависимости от его микроархитектуры обязан иметь счётчик команд и ровно 32 регистра. Зная текущее архитектурное состояние, процессор точно знает, какую операцию и над какими данными надо выполнить для получения нового архитектурного состояния.

Рассмотрим структуру, показанную на Рисунке 9 и модуль верхнего уровня для разработанного процессорного ядра.

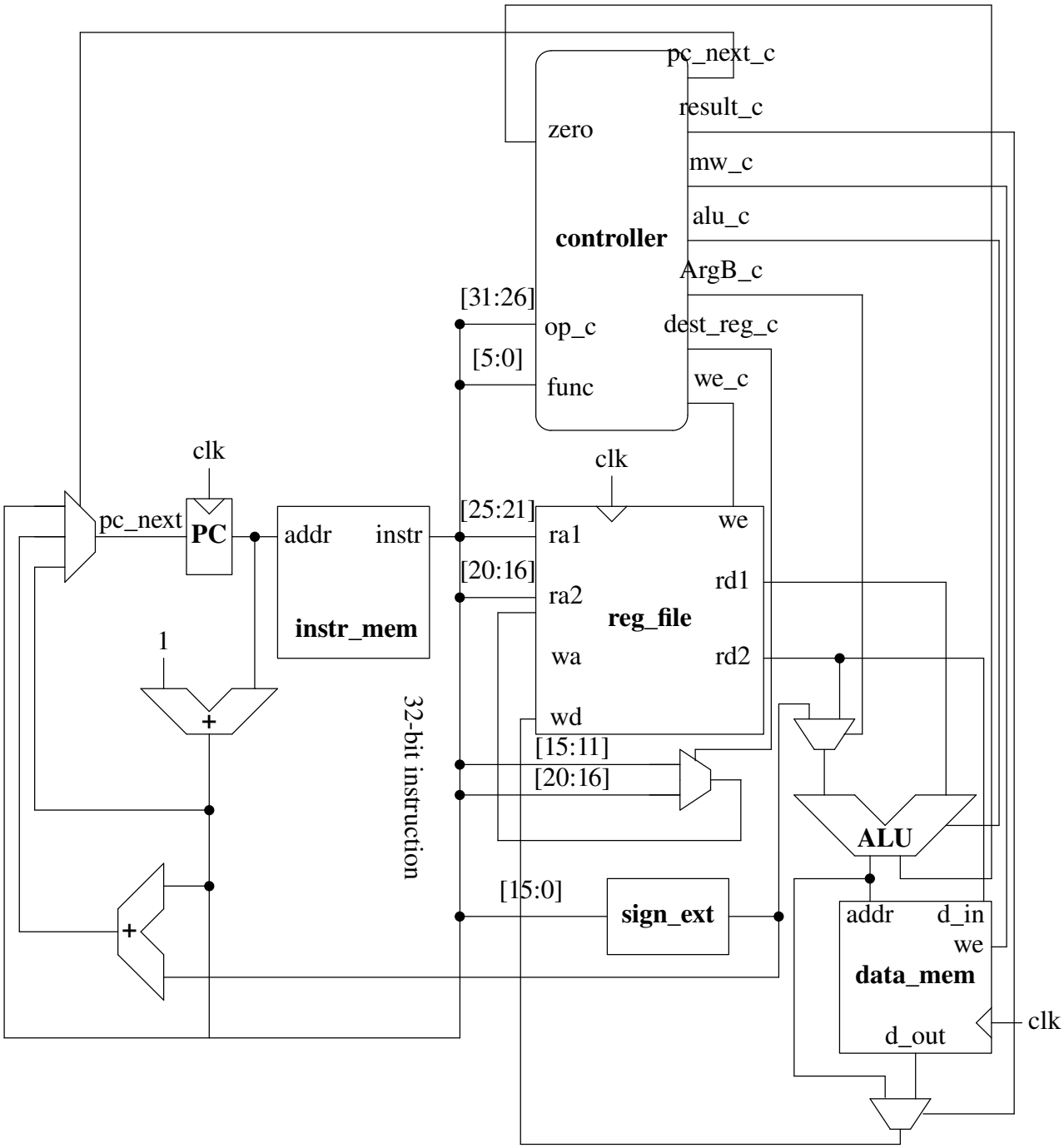

Рисунок 9 — CPU

В файле CPU.v описана общая структура ядра.

```
module cpu (
    input wire clk,
    input wire reset,
    );

    wire [31:0] instr;
    wire [31:0]  A, B, rd2, C, s_imm, read_data, result;
    wire [31:0] pc_next, pc_val, pc_inc, pc_br;
```

```
wire [4:0] dest_reg;
wire argB_c, dest_reg_c, we_c, result_c, mw_c, branch_c, zero;
wire [3:0] alu_c;

PC     pc(.ctrl(clk), .reset(reset), .in(pc_next), .out(pc_val));

rom2       rom_uut(.clk(clk), .addr(pc_val), .data(instr));

adder      pc_incr(.A(32'h00000001), .B(pc_val), .C(pc_inc));

adder      branch_add(.A(s_imm), .B(pc_inc), .C(pc_br));

pc_val_mux pc_mux(.ctrl(branch_c), .in0(pc_inc),
.in1(pc_br), .out(pc_next));

sign_ext   s_e(.in(instr[15:0]), .out(s_imm));

reg_file   r_f(.clk(clk),.we(we_c), .ra1(instr[25:21]),
.ra2(instr[20:16]), .wa(dest_reg), .rd1(A), .rd2(rd2),
.wd(result) );

mux2to1    #(.SIZE(5))mux21_dest(.in0(instr[15:11]),
.in1(instr[20:16]), .ctrl(dest_reg_c), .out(dest_reg));

mux2to1    mux21_argB(.in0(rd2), .in1(s_imm), .ctrl(argB_c),
.out(B));

alu        alu_uut(.A(A), .B(B), .C(C), .mode(alu_c), .zero(zero));

ram    data_mem(.clk(clk), .we(mw_c), .addr(C),
.d_in(rd2), .d_out(read_data));

mux2to1    mux21_result(.in0(C), .in1(read_data),
.ctrl(result_c), .out(result));


contr      c_uut(.op_c(instr[31:26]), .funct(instr[5:0]),
.zero(zero), .argB_c(argB_c), .dest_reg_c(dest_reg_c), .we_c(we_c),
.result_c(result_c), .mw_c(mw_c), .branch_c(branch_c),
.alu_c(alu_c));
```

```
endmodule
```

Файл CPU.v

В структуре итогового процессорного ядра можно выделить 3 основных модуля:

1. Регистровый файл;
2. Арифметико-логическое устройство;
3. Контроллер.

В рамках данной работы будут рассмотрены эти три модуля. Файлы описания остальной части ядра приведены в приложении.

Файл reg_file.v содержит описание регистрового файла, хранящего все 32 видимых регистра.

```
module reg_file(
    input wire clk,
    input wire we,//for write enable

    input wire [4:0] ra1,//
    input wire [4:0] ra2,//for read  addresses
    input wire [4:0] wa,//for write address

    output wire [31:0] rd1,// for read data
    output wire [31:0] rd2,//
    input wire [31:0] wd//for write data
);

localparam
//registers
    zero = 5'b00000,

    at = 5'b00001,

    v0 = 5'b00010,
    v1 = 5'b00011,

    a0 = 5'b00100,
    a1 = 5'b00101,
    a2 = 5'b00110,
    a3 = 5'b00111,

    t0 = 5'b01000,
```

```
    t1 = 5'b01001,
    t2 = 5'b01010,
    t3 = 5'b01011,
    t4 = 5'b01100,
    t5 = 5'b01101,
    t6 = 5'b01110,
    t7 = 5'b01111,

    s0 = 5'b10000,
    s1 = 5'b10001,
    s2 = 5'b10010,
    s3 = 5'b10011,
    s4 = 5'b10100,
    s5 = 5'b10101,
    s6 = 5'b10110,
    s7 = 5'b10111,

    t8 = 5'b11000,
    t9 = 5'b11001,

    k0 = 5'b11010,
    k1 = 5'b11011,

    gp = 5'b11100,
    sp = 5'b11101,
    fp = 5'b11110,
    ra = 5'b11111;

    reg [31:0] rf [31:0];

    always @(posedge clk)
    if (we)
        rf[wa] <= wd;

    assign rd1 = (ra1 != 0) ? rf[ra1] : 0;
    assign rd2 = (ra2 != 0) ? rf[ra2] : 0;
endmodule
```

Файл reg_file.v

Регистровый файл выполнен в виде блока RAM. На вход поступают 3 адреса — два адреса регистров-источников операндов (`rd1, rd2`), значения которых подаются на выходы `rd1, rd2` и один адрес регистра-назначения результата (`wa`), которому присваивается значение, поданое на вход `wd`, при условии, что был послан управляющий сигнал `we`.

Файл alu.v содержит описание АЛУ.

```
module alu(
    input wire [3:0] mode,
    input wire [31:0] A,
    input wire [31:0] B,
    output reg [31:0] C,
    output reg zero
    );

localparam
    //functions
    //logical
    and_f = 4'b0000,
    or_f = 4'b0001,
    xor_f = 4'b0010,
    nor_f = 4'b0011,
    slt_f = 4'b0100,
    nand_f = 4'b0101,

    //arithmetic
    add_f = 4'b1000,
    subtr_f = 4'b1001;

    always @ *
    begin
    zero = 0;
       case(mode)
       and_f:
           C = A & B;
       nand_f:
           C = ~(A & B);
       or_f:
           C = A | B;
       xor_f:
           C = A ^ B;
```

```
        nor_f:
            C = ~(A | B);
        compl_f:
            C = -A;
        subtr_f:
            C = A - B;
        add_f:
            C = A + B;
        default:
            C = A;
        endcase

        if (C == 0)
                zero = 1'b1;
     end

endmodule
```

Файл alu.v

На входы A и B подаются значения операндов, на вход mode — код выполняемой инструкции, результат подаётся на выход C. Стоит так же отметить наличие флага zero, выставляемого при равенстве результата нулю.

Файл contr.v содержит описание контроллера.

```
module contr(
    input wire [5:0] op_c,
    input wire [5:0] funct,
    input wire zero,
    output wire argB_c,
    output wire dest_reg_c,
    output wire we_c,
    output wire result_c,
    output wire mw_c,
    output wire branch_c,
    output wire [3:0] alu_c
    );
    wire [1:0] aluop;
    maindec    main_dec(.op_c(op_c), .argB_c(argB_c),
    .dest_reg_c(dest_reg_c), .we_c(we_c), .result_c(result_c),
    .mw_c(mw_c), .branch(branch), .aluop(aluop));
```

```
    aludec      alu_dec(.funct(funct), .aluop(aluop),
    .alu_c(alu_c) );
    assign      branch_c = zero & branch;
endmodule
```

Файл contr.v

Контроллер разделён на две части: основной декодер и декодер АЛУ. Основной декодер преобразует код операции в управляющие сигналы. Декодер АЛУ преобразует поле функции и сигнал `aluop` в управляющие сигналы АЛУ.

4.1. Итог

В рамках курсовой было создано ядро процессора с MIPS-подобной архитектурой. Данное ядро реализует лишь основные инструкции архитектуры MIPS, такие как арифметические и логические, функции перехода, общения с памятью. И всё же устройство является полноценным однотактным процессором с 32-битным набором команд.

ЗАКЛЮЧЕНИЕ

В данной работе были выполнены все поставленные задачи.

Была рассмотрена применимость ПЛИС и Hardware Description Language (HDL) для задач ЗИ. Резульатом было решение о том, что ПЛИС имеют ряд особенностей, делающих их мощным инструментом для решения задач ЗИ, в том числе и необычных.

Была изучена архитектура MIPS и рассмотрены способы построения и пременения аппаратных средств КМЗИ на основе процессоров.

Была изучена архитектура MIPS;

Было разработано и описано процессорное ядро с MIPS-подобной архитектурой на языке Verilog.

Таким образом,на основании выполнения всех поставленных задач, цель данной работы можно считать полностью выполненной. В дальнейшем планируется дополнение, расширение ядра и создание на его основе аппаратного устройства КМЗИ одним из вышеописанных способов, а так же снабжение его соответствующими интерфейсами. Данные задачи будут выполнятся в рамках ВКР.

СПИСОК ИСПОЛЬЗОВАННЫХ ЛИТЕРАТУРЫ И ИСТОЧНИКОВ

1) David M. Harris and Sarah L. Harris. Digital Design and Computer Architecture. —1-е изд. — Boston:Morgan Kaufman, 2007 .— 570 с.
2) P. Pal Chaudhuri. Computer organisation and design. — 3-е изд. — Delhi:PHI Learning, 2014. — 897 с.
3) Самоделов А. Криптография в отдельном блоке: криптографический сопроцессор семейства STM32F4xx: [Эдектронный ресурс] // Официальный сайт компании "Компэл": URL: http://www.compel.ru/lib/ne/2012/6/4-kriptografiya-v-otdelnom-bloke-kriptograficheskiy-soprotsessor-semeystva-stm32f4xx. (дата обращения 03.12.2016).

ПРИЛОЖЕНИЯ

Приложение А

СПИСОК ФАЙЛОВ НА CD-ДИСКЕ

А. Spartan-3A/3AN FPGA Starter Kit Board User Guide.

Б. Файл adder.v

В. Файл aludec.v

Г. Файл alu.v

Д. Файл contr.v

Е. Файл cpu.v

З. Файл maindec.v

И. Файл mux2to1.v

К. Файл PC.v

Л. Файл pc_val_mux.v

Н. Файл ram.v

П. Файл reg_file.v

Р. Файл instr_mem.v

С. Файл sign_ext.v