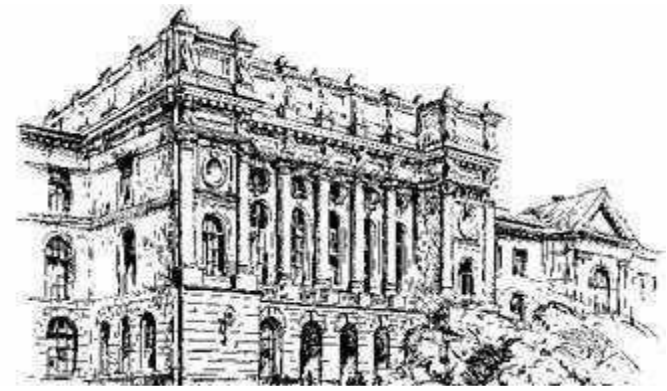
Министерство образования и науки Российской Федерации

Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа интеллектуальных систем и суперкомпьютерных технологий



ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Создание прототипов Soft-процессора для модуля управления

Студент гр. 3530901/70202 Д. А. Курякин

Санкт-Петербург 2021

Министерство образования и науки Российской Федерации

«Санкт-Петербургский политехнический университет Петра Великого»
Институт компьютерных наук и технологий

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

Работа допущена к защите
Директор ВШИСиСТ
_____ В.М. Ицыксон
«___»_____2021 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Создание прототипов Soft-процессора для модуля управления

По направлению 09.03.01 «Информатика и вычислительная техника»
по образовательной программе
09.03.01_02 «Технологии разработки программного обеспечения»

Выполнил студент гр. 3530901/70202 _____ Д. А. Курякин

Руководитель
ст. преподаватель _____А. В. Лупин

Норм контролёр,
ст. преподаватель _____С. А. Нестеров

Санкт-Петербург 2021

САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО

Институт компьютерных наук и технологий

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

УТВЕРЖДАЮ

Директор ВШИСиСТ

_____ В.М. Ицыксон

«___»_____2021 г.

ЗАДАНИЕ

на выполнение выпускной квалификационной работы

студенту Курякину Даниле Александровичу, группа 3530901/70202

1. **Тема работы:** Создание прототипов Soft-процессора для модуля управления

2. Срок сдачи студентом законченной работы: июнь 2021 года.

3. Исходные данные по работе:

3.1. Документация ADSP-218X

3.2. Документация Nios II

3.2. Документация Мультикор 1892ВМ5Я

4. Содержание работы (перечень подлежащих разработке вопросов):

4.1. Возможные варианты создания soft-процессора.

4.2. Реализация прототипа с использованием Nios II.

4.3. Реализация прототипа собственной разработки.

5. **Дата выдачи задания:** «___»_____2021 г.

Руководитель ВКР _____ А. В. Лупин

Задание принял к исполнению «___»___2021 г.

Студент _____ Д. А. Курякин

РЕФЕРАТ

с. 57, рис. 28, табл. 16, прил. 5

КЛЮЧЕВЫЕ СЛОВА: SOFT-ПРОЦЕССОР, ПЛИС, NIOS II, AVR8, MIPS32, МУЛЬТИКОР, Verilog.

В работе рассматривается вопрос модернизации модуля управления МВ-1000М с целью замены устаревшего процессора ADSP-2185N процессором, размещенным в ПЛИС модуля. Характеристики разрабатываемого Soft-процессора соизмеримы с существующим процессором по быстродействию, а ресурсы в ПЛИС EP4CE22 или 5CEFA4 использованы до 30%. Представлены прототипы с использованием NIOS II gen 2 и несколько прототипов с усеченными ядрами процессоров AVR8 и MIPS32. Проекты написаны на языке Verilog. Тестовые программы написаны на языках С, ассемблер и в кодах. Практические результаты работы могут быть интересны фирмам, которые стремятся к уменьшению номенклатуры и типов процессоров и контроллеров в их изделиях.

ABSTRACT

pp. 57, pic. 28, tabl. 16, app. 5

KEYWORDS: SOFT-PROCESSOR, FPGA, NIOS II, AVR8, MIPS32, MULTICORE.

The paper discusses the issue of upgrading the MB-1000M control module in order to replace the outdated ADSP-2185N processor with a processor located in the FPGA of the module. The characteristics of the developed Soft-processor are comparable with the existing processor in terms of speed, and the resources in the EP4CE22 or 5CEFA4 FPGAs are used up to 30%. Presented prototypes using NIOS II gen 2 and several prototypes with reduced cores of AVR8 and MIPS32 processors. Projects are written in Verilog language. Test programs are written in C, assembly and code. The practical results of the work may be of interest to firms that seek to reduce the range and types of processors and controllers in their products.

СОДЕРЖАНИЕ

СПИСОК ОСНОВНЫХ СПЕЦИАЛЬНЫХ ТЕРМИНОВ И СОКРАЩЕНИЙ

Сокращение	Определение
SP	Soft-processor, софт-процессор
АЛУ	Арифметико-логическое устройство
ПЦОС	Процессор цифровой обработки сигналов
ПЛИС	Программируемая логическая интегральная схема
Мультикор	Семейство микропроцессоров
АЦП	Аналого-цифровой преобразователь
Verilog	Язык описания аппаратных средств
JTAG	Joint Test Action Group, отладочный интерфейс
FLASH	Твердотельное устройство хранения
RAM	Random Access Memory, оперативное запоминающее устройство
SoC	System on Cristal, система на кристалле
PC	Personal Computer, персональный компьютер

ВВЕДЕНИЕ

Программируемые логические интегральные схемы (ПЛИС) в последние годы получили широкое распространение для реализации электронных схем и систем. Номенклатура микросхем ПЛИС обновляется, они свободно распространяются и предлагаются типовые модули для обучения. Используя язык описания аппаратных средств высокого уровня такой как Verilog (System Verilog) и программное обеспечение для программирования логических устройств, можно спроектировать и построить сложную функциональную схему, состоящую из десятков тысяч логических элементов.

Наряду с совершенствованием ПЛИС, с середины 90-х годов в электронных устройствах используются процессоры цифровой обработки сигналов (ПЦОС) для решения определенных задач в реальном масштабе времени.

Современные ПЛИС фирмы Intel семейства Cyclone V и даже Cyclone IV имеют 30 тысяч и более логических элементов. Имеется возможность реализовать систему на кристалле, создать Soft-процессор [1] вместо ПЦОС, т.е. объединить два функционала.

В данной работе рассматривается решение частной задачи – модернизация модуля управления МВ-1000М. Цель работы - поиск и оценка возможности замены процессора ADSP-2185N (на кристалле) Soft—процессором для размещения в ПЛИС EP4CE22 (Cyclone IV) или 5CEFA4. В процессе исследования необходимо рассмотреть создание процессора на платформе NIOS II ПЛИС фирмы Intel и спроектировать собственную модель (несколько прототипов) Soft-процессора. В качестве среды проектирования следует использовать Quartus Prime Lite, рекомендуемый Intel (Altera). Для отладки проектов предоставляется плата miniDiLaB-CIV, содержащая ПЛИС EP4CE6E22C8N.

ГЛАВА 1. ВАРИАНТЫ РЕАЛИЗАЦИИ ПРОТОТИПА SOFT-ПРОЦЕССОРА МОДУЛЯ УПРАВЛЕНИЯ

В главе рассматривается объект исследования, формулируется задача на проектирование и приводятся возможные варианты реализации прототипа Soft-процессора модуля управления.

1.1. Описание модуля управления.

Модуль управления MB-1000M выпускается предприятием ООО «Фирма «Нево-Д» (г. Санкт-Петербург) под заказ. Модуль изготовлен в форм-факторе Евромеханика 3U. На рис.1.1 представлена структура модуля. Модуль содержит: контроллер сетевого обмена, блок сбора данных. Оба базируются на ПЛИС CYCLONE IV, процессор цифровой обработки сигналов (ПЦОС) ADSP-2185NKST-320, контроллер PHY GIGABIT ETHERNET и буферные элементы. Модуль имеет до 30 входов для приема данных от АЦП и 10 входов/выходов для приема/выдачи управляющих сигналов. Есть два интерфейса: RS-485 и RS-422.

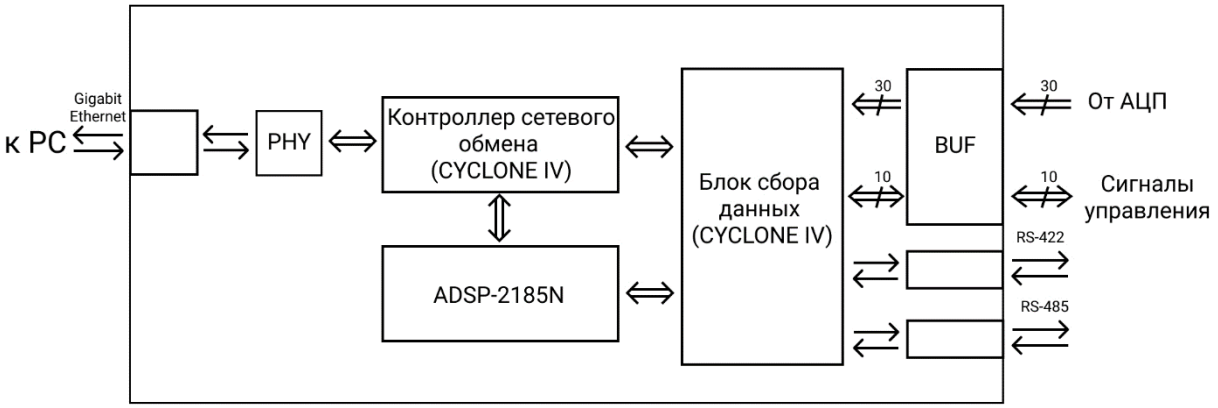


Рис.1.1. Структура модуля MB-1000M

Предполагается после реализации Soft-процессора разместить его в ПЛИС. Процессор ADSP-2185N использоваться не будет. На рис.1.2 показана структура модуля MB-1000M с Soft-процессора.

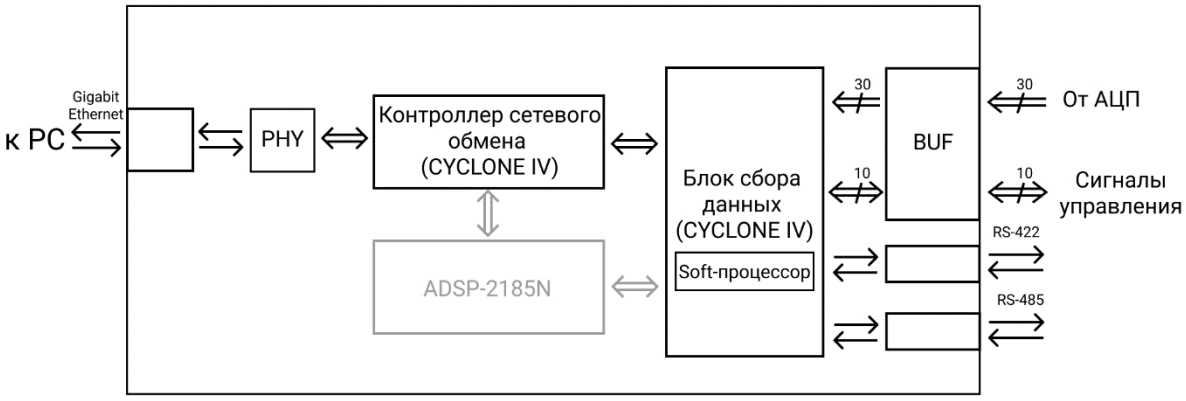


Рис.1.2. Структура модуля MB-1000M с Soft-процессором

1.2. Основные функции и назначение ПЦОС.

В электронных модулях, которые содержат два разных устройства, ПЛИС и ПЦОС, функции между ними распределяются в зависимости от конкретной общей задачи. Ниже перечислены основные функции ПЦОС в модуле управления:

- прием и обработка данных от внешних устройств по прерываниям;
- формирование и выдача данных во внешние устройства;
- коммутация шин в ПЛИС, загрузка параметров:
- реализация алгоритмов интерполяции и экстраполяции;
- реализация функций генератора и осциллографа.

1.3. Требования к Soft – процессору.

ПЦОС ADSP-2185N является одним из последних процессоров семейства ADSP-218x фирмы Analog Devices. Это 16-разрядные процессоры с фиксированной запятой. Хотя процессор «числится» в возможных поставках, как «active», но уже несколько лет фирма не рекомендует использование и не поддерживает программное обеспечение [2].

В связи с такой ситуацией требуется замена данного ПЦОС. Наиболее выгодным вариантом модернизации модуля управления была бы реализация soft-процессора для имплементации его в ПЛИС Cyclone IV и

последующей реализации нового модуля управления на Cyclone V без процессора.

Основные требования к Soft – процессору:

- ограничения по ресурсам 6 тыс. логических ячеек с триггерами в ПЛИС EP4CE22E22C7 (в перспективе – до 10 тыс. в ПЛИС 5CEBA4F23C7);
- память программ - не менее 10 Кбайт;
- память данных – не менее 16 Кбайт;
- быстродействие – 80 млн. команд/с;
- разрядность - 16/32 разряда;
- команды (стандартный набор), умножение и деление;
- среда разработки программ с открытом открытым исходным кодом (Open source).

1.4. Анализ требований и пути реализации SP

Рассмотрим пути реализации решения проблемы замены ПЦОС ADSP-2185N: реализация SoC на базе аппаратного ARM в Cyclone V Sx, эмуляция ADSP на HDL, реализация Soft–процессора на базе NIOS II, использование модели Soft-процессора из OpenCore на HDL, реализация собственного прототипа Soft-процессора на HDL.

1.4.1. Реализация SoC на базе аппаратного ARM в Cyclone V Sx

Системы SoC FPGA объединяют архитектуру процессора и FPGA на одном кристалле. Следовательно, они обеспечивают меньшее энергопотребление, меньший размер платы, сниженную стоимость за счет совмещения двух устройств в одном корпусе и высокую скорость обмена данными между ПЛИС и ЦП. Они также включают в себя богатый набор периферийных устройств, встроенную память, логический массив в стиле ПЛИС [3].

Многие компании, разрабатывающие SoC FPGA, сконцентрировались на широко распространённой архитектуре ARM. Эта архитектура позволяет

использовать ее широкую эко систему совместимых инструментов. Так же использование сложных интеллектуальных блоков (IP-ядер) и программного обеспечения для проектирования на FPGA позволяет сделать процесс разработки очень привлекательным.

Использование SoC особенно полезно, когда требуется высокая производительность для части алгоритма, который может быть реализован аппаратно с использованием параллельных или последовательных методов. Например, этот подход работает лучше всего, когда требуется одновременно выполнить несколько алгоритмов, ориентированных на производительность. Одной из областей применения к которой SoC FPGA добилась значительного успеха, была сложная обработка изображений. Эти алгоритмы могут быть распараллелены или конвейеризированы, что дает хорошую задачу для ПЛИС [4]. Этот путь возможен только при использовании ПЛИС с аппаратным процессором, поэтому далее не рассматривается.

1.4.2. Эмуляция ADSP на HDL

В 2017 году на кафедре КСПТ (ныне ВШИСиСГ) под руководством ст. преподавателя Лупина А.В. в рамках курсовых проектов студентов были созданы эмуляторы трех арифметических блоков ЦПОС ADSP218x на языке System Verilog. Максимальная частота работы блоков ALU – 46 МГц, SHIFTER – 69 МГц, MAC – 34 МГц. На рис.1.3 показаны реализованные блоки.

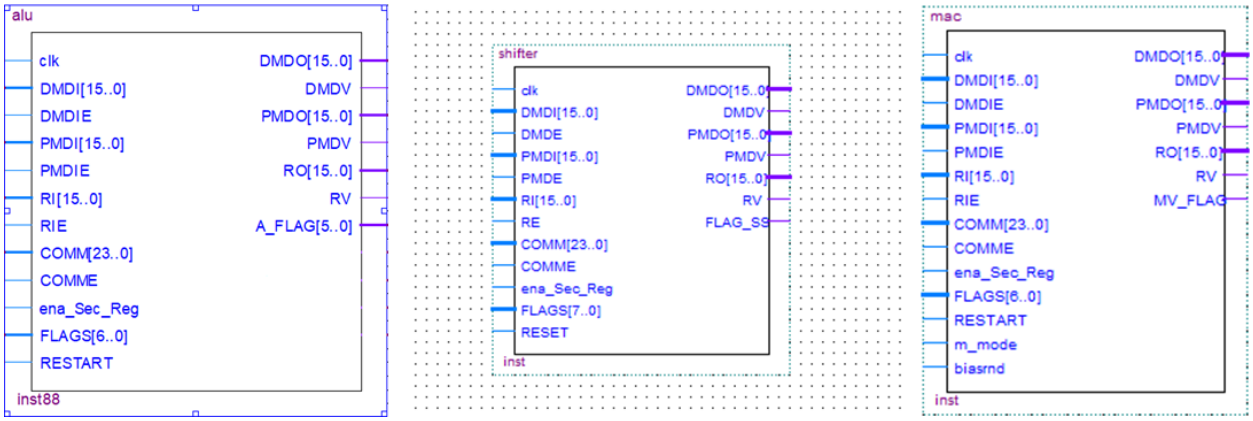


Рис.1.3. Блоки эмулятора ЦПОС ADSP-218x

Позднее было принято решение не реализовывать аналог ПЦОС ADSP-218x, и не использовать программное обеспечение фирмы Analog Devices. Результат проведенной работы показывает недостаточное быстродействие и в дальнейшем следует проектировать процессор с использованием конвейерной структуры. Однако опыт разработки блоков процессора будет использован в настоящей работе.

1.4.3. Реализация Soft-процессора на базе NIOS II

Nios II – это SP с RISC архитектурой реализованный специально для ПЛИС Altera [5]. У него существуют две версии: classic (gen 1) и gen 2. Компания Altera рекомендует использовать более современную, последнюю версию.

Nios II встроен в качестве IP-ядра в среду разработки Intel Quartus Prime. В время настройки можно добавлять и убирать различные функции для достижения желаемых требований. Так же существует большой выбор настраиваемой боков для взаимодействия с периферией. Есть специальная инструменты для написания прошивки и ее отладки. Реализация Soft-процессора на базе NIOS II представлена в главе 2.

1.4.4. Использование модели Soft-процессора из OpenCore на HDL

Сайт OpenCore имеет большое количество различных ядер процессоров, микроконтроллеров и других компонентов [6]. Многие из проектов имеют полную совместимость с существующими архитектурами микропроцессоров, есть компилятор и отладчики. Однако чтение

документации и адаптация ядра под нужную ПЛИС требует значительных усилий. В дальнейшем планируется использовать в своих разработках простейшие варианты проектов этого ресурса.

1.4.5. Реализация собственного прототипа Soft–процессора на HDL

В качестве примера реализации собственного Soft–процессора послужил проект ядра rAVR8 на сайте OpenCore, который создали ПЛИС разработчики компании ООО "ИНПРО ПЛЮС"(г. Таганрог) [7]. В статье рассказывается как можно создать частично совместимый 8-битный AVR микроконтроллер на микросхеме CPLD EPM240T100C5 с 240 логическими элементами. Созданный протопит имеет 16 команд и 7 регистров. Три регистра используются для взаимодействия с перевивными устройствами: чтения информации с кнопок, вывод на светодиоды и управления шаговым двигателем. Для памяти данных используется FLASH - память.

Для создания собственных прототипов возьмем концепцию конвейера от процессора семейства Мультикор, от компании ОАО НПЦ «ЭЛВИС» [8]. Документация по процессору представлена в [9]. Реализация прототипов Soft-процессора представлена в главе 3.

1.5. Выбор направления реализации

Из вышеупомянутых направлений выбираем два направления: реализация Soft – процессора на базе NIOS II gen 2 и реализация собственного прототипа Soft – процессора на HDL.

1.6. Отладка и тестирование

Для тестирования модели прототипов будем использовать среду моделирования ModelSim. Также для отладки прототипов будем использовать отладочную плату miniDiLaB-CIV, содержащую микросхему EP4CE6E22C8N.

ГЛАВА 2. СОЗДАНИЕ ПРОТОТИПА SOFT – ПРОЦЕССОРА НА ОСНОВЕ NIOS II GEN 2

2.1. Основные сведенья

Архитектура Nios II описывает конкретный набор команд. Для выполнения набора команд используются функциональные блоки. Они могут быть реализован аппаратно, эмулироваться программно или полностью исключены. В архитектуре Nios II определены следующие функциональные блоки:

- Зарегистрировать файл
- Арифметико-логический блок (АЛУ)
- Интерфейс для пользовательской логики инструкций
- Контроллер исключений
- Внутренний или внешний контроллер прерываний
- Шина данных
- Блок управления памятью, не используем
- Блок защиты памяти, не используем
- Память инструкций или данных
- Плотно связанные интерфейсы памяти для инструкций и данных
- Модуль отладки JTAG

Реализация Nios II – это набор вариантов дизайна, воплощенных в конкретном ядре процессора Nios II. Все реализации поддерживают одинаковый набор команд, описанный [10].

Каждая реализация SP нужна для определенных целей, таких как меньший размер ядра или более высокая производительность. Эта гибкость позволяет адаптировать архитектуру Nios II к различным задачам.

Существуют две версии ядра Nios: e и f.

- Nios II/f – ядро для высокой производительности. Обладает большими возможностями настройки для более точной конфигурации. Платная версия.

- Nios II/e – ядро для максимальной экономии на размере. Ядро с ограниченными возможностями, многие настройки отсутствуют. Бесплатная версия.

Ядро Nios II/f используется для обеспечения высокой производительности за счет размера ядра. Компания Intel разрабатывала его для специально для максимального повышения эффективности выполнения инструкций за цикл, оптимизации задержки прерывания и максимального увеличения производительности ядра процессора Fmax.

Для увеличения производительности для операций умножения, деления и сдвига используются специальные блоки. Они могут быть реализованы с использованием DSP блоков на плате, встроенных умножителей, созданы на логических элементах, программно эмулироваться, либо отсутствовать.

При использовании специальных блоков сдвига АЛУ выполняет операции сдвига за три тактовых цикла. В противном случае ALU включает в себя специализированную схему сдвига, которая обеспечивает сдвиг на один бит за цикл.

2.2. Настройка ядра

Конфигурация ядра процессора задается с помощью специального программного обеспечения под названием «Platform Designer», в более ранних версиях Quartus оно называлось «Qsys». На рис.2.1. показано окно программы.

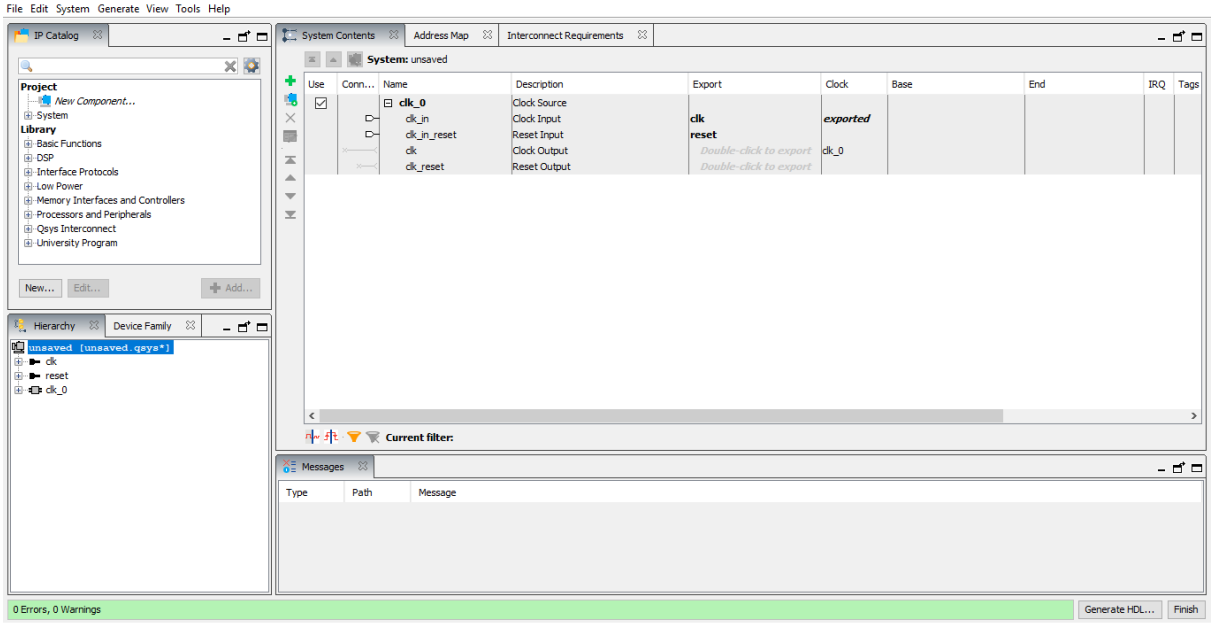


Рис.2.1. Окно конфигуратора Platform Designer

Программа конфигуратор состоит из окна библиотеки IP-ядер IP Catalog, окна конфигурации созданных блоков System Contents, окна консоли Messages, иерархии Hierarchy.

При первой загрузке в окне System Contents находится блок тактового счетчика clk_0. Он принимает тактовый импульс и сигнал сброса. Затем выводит их на другие блоки. Для добавления нового блока в окне поиска IP Catalog введем Nios II Processor. Дважды щёлкнем по нему, откроется окно настройки. На рис.2.2. показано окно настройки ядра SP.

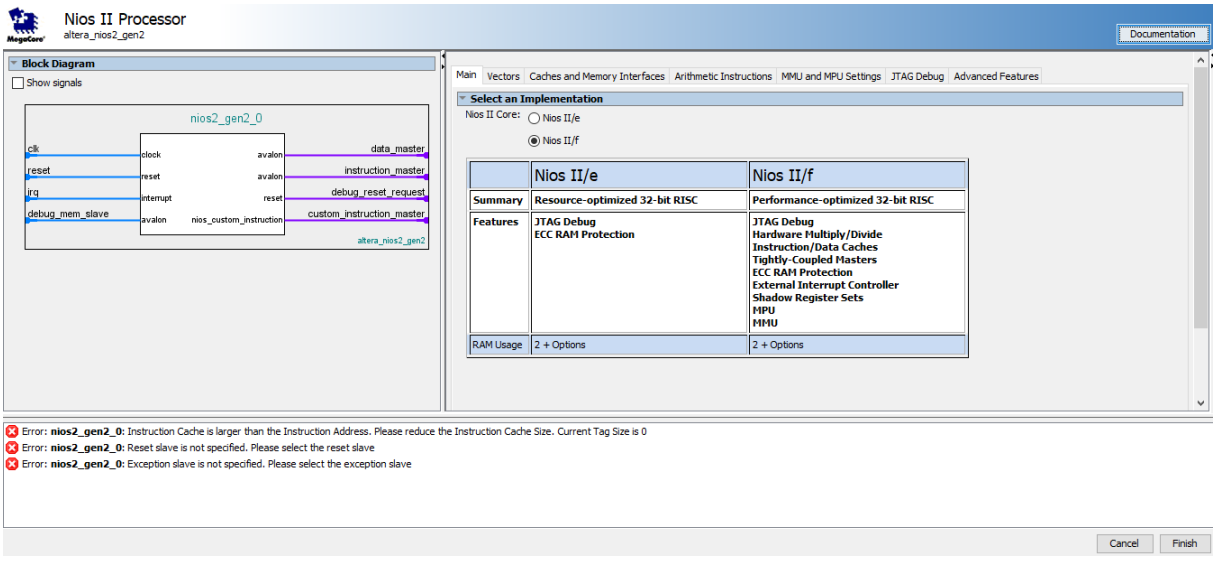


Рис.2.2. Окно настройки ядра SP.

Ядро Nios II содержит множество функций в зависимости от требований можно убирать или добавлять их. Во вкладке Main можно выбрать версию ядра, Vectors можно выбрать участок памяти на который будет осуществляться переход в случае сброса и ошибки, Caches and Memory Interfese можно настроить память программ или данных, Arithmetic Instructions можно настроить АЛУ, MMU and MPU Settings можно настроить MMU и MPU блоки, JTAG Debag настройка JTAG, Advanced Features другие настройки.

Выберем версию ядра Nios II/f остальные настройки оставим теми же. Нажмем кнопку Finish. Также добавим преобразователь частоты AltPLL. Он принимает частотный сигнал и путем умножения и деления на коэффициенты преобразует в необходимую частоту.

Мы создали одну из простейших конфигураций. Система состоит из блоков: счетчика (clk_0), ядра Nios II (nios2_gen2_0) и преобразователя частоты (alltpll_0). На рис.2.3. представлена простейшая конфигурация ядра Nios II бес соединений.

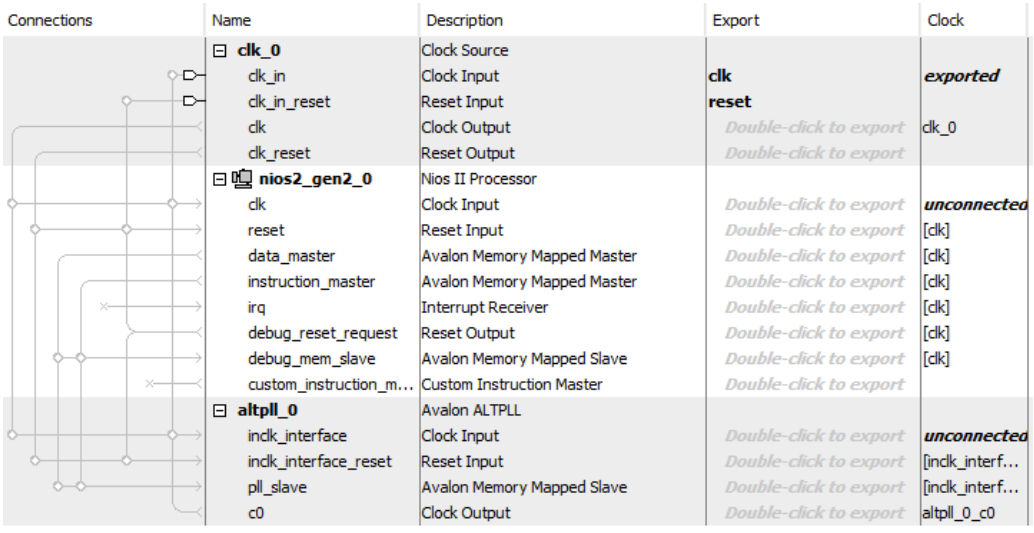


Рис.2.3. Простейшая конфигурация ядра Nios II без соединений

После добавления блоков в систему нужно настроить их соединения. На рис.2.4. наказана конфигурация ядра Nios II с соединениями. Сигнал clk

проведем ко входу преобразователя частоты inck_interface. Преобразованный тактовый импульс c0 с выхода преобразователя частоты проведем к входу clk блока Nios. Объединим сигналы сброса всех блоков и объединим шины обмена данными Avalon между блоком Nios и преобразователем частоты.

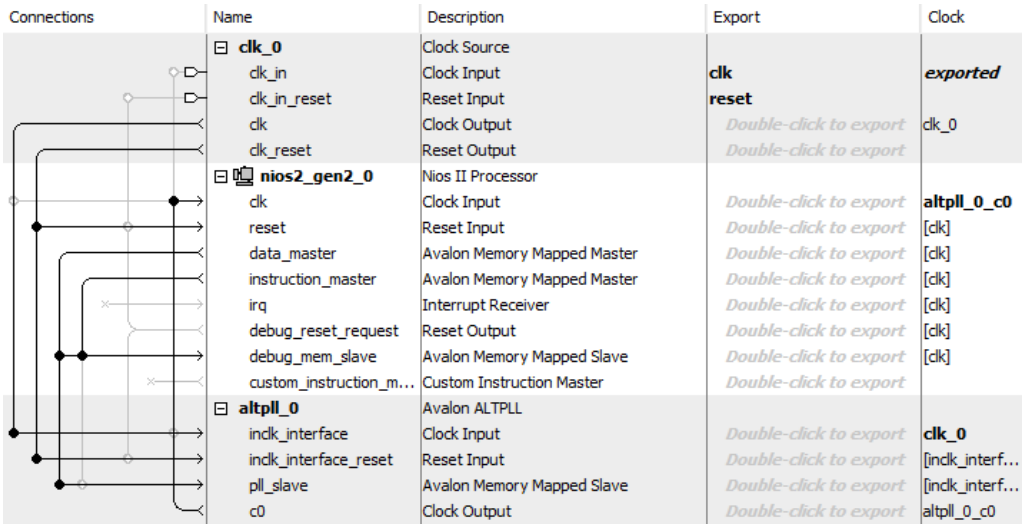


Рис.2.4. Простейшая конфигурация ядра Nios II с соединениями

2.3. Создание схемы

После генерации ядра SP. Была создана диаграмма верхнего уровня. Она представлена рис.2.5. На диаграмме представлено созданое я ядро Nios. Сигнал сброса задается нулем, поэтому к контакту подключен положительный сигнал. На отладочной плате кнопка «a» при нажатии задает сигнал равный 0, а кнопка «b» 1, поэтому для удобства написания программ значение на кнопке «a» инвертируется.

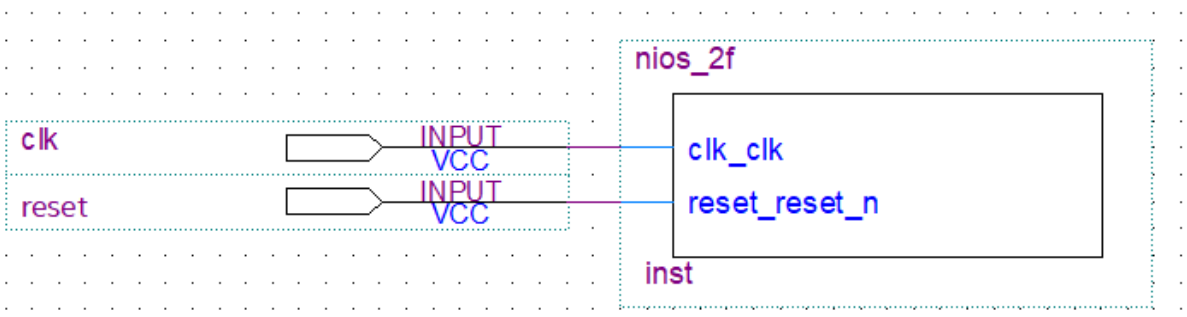


Рис.2.5. Диаграмма верхнего уровня с ядром Nios

2.4. Исследование зависимости характеристик от размера памяти

При настройке блока Nios II можно выбрать версию ядра. При выборе Nios II/f доступны разделенные память данных и память инструкций, при выборе версии Nios II/e таких настроек нет.

Скомпилируем систему с различными настройками памяти. Перед каждой компиляцией будем повышать размер каждой памяти. В таблице 2.1 показаны результаты компиляции. У версии ядра Nios II/e выполняется одна команда за 6 тактов. Поэтому в скобках указана частота, поделенная на 6. Так как ПЛИС в отладочной плате имеет другие показатели быстродействия по сравнению с используемой на модуле MB-1000M, то максимальная частота (Fmax) была измерена для двух ПЛИС EP4CE22E22C7 и EP4CE6E22C8N (отладочная плата).

Таблица 2.1

Результаты компиляции простейшего прототипа

№	Название	Fmax для C7, МГц	Fmax для C8, МГц	Количество логических элементов	Количество используемой памяти, байт
1	2	3	4	5	6
1	Nios II/e	160(26)	146 (24)	1311	10240
2	Nios II/f с 512 б памяти данных и 512 байтами памяти инструкций	131	119	2935	19616
3	Nios II/f с 2 Кбайтами памяти данных и 2 Кбайтами памяти инструкций	128	117	2945	45184

Оконч. табл. 2.1

1	2	3	4	5	6
4	Nios II/f с 4 Кбайтами памяти данных и 4 Кбайтами памяти инструкций	121	110	2928	79104
5	Nios II/f с 8 Кбайтами памяти данных и 8 Кбайтами памяти инструкций	117	107	2963	146688

В ПЛИС EP4CE6E22C8N имеется 30 элементов памяти общем объемом 276480 бит. Наибольшее количество памяти данных и памяти данных удалось достичь 8 Кбайт каждая. При последующем их увеличение при компиляции ведётся ошибка о превышении количества элементов памяти.

Самая высокая частота у прототипа Nios II/e, но на выполнение одной операции требуется 6 тактов счетчика. Поэтому частоту 146 МГц нужно поделить на 6. Реальная частота прототипа равна 24.3 МГц.

Установим размер памяти программ 4 Кбайт и памяти данных 2 Кбайт. Добавим и соединим IP-ядро On chip memory. Ядро используется для создания области памяти типа RAM или ROM. В этом блоке используются ячейки памяти в кристалле ПЛИС, где реализован Nios. На рис.2.6. показана конфигурация ядра с внешней памятью.

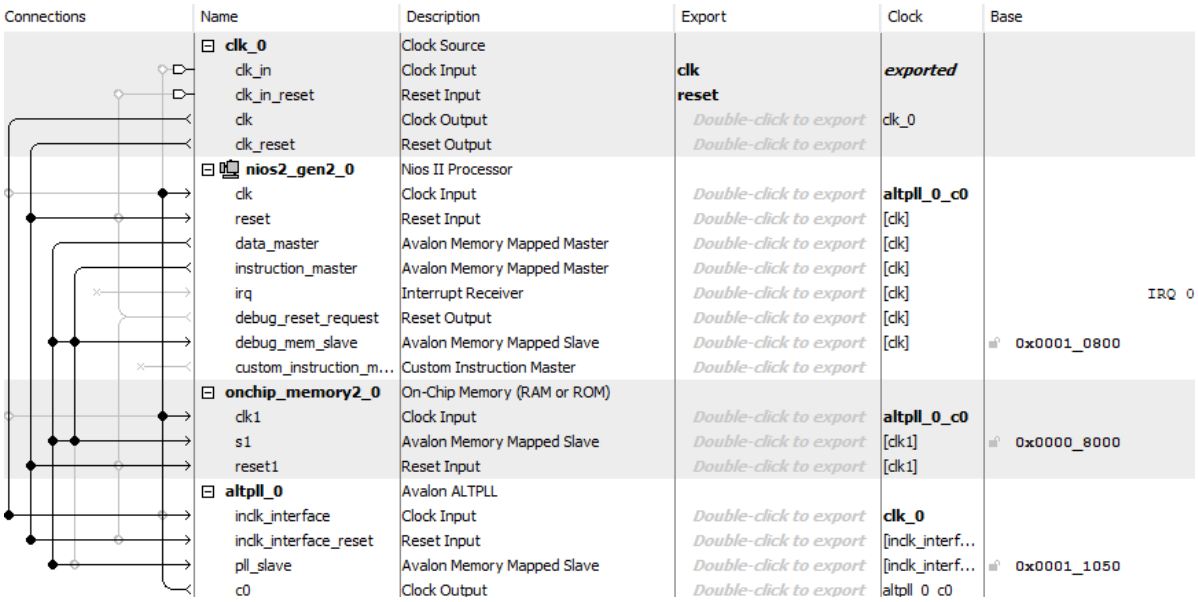


Рис.2.6. Конфигурация ядра Nios II с RAM памятью

В таблице 2.2 показаны результаты компиляции прототипа с RAM памятью.

Таблица 2.2

Результаты компиляции прототипа с RAM памятью

№	Название	Fmax для C7, МГц	Fmax для C8, МГц	Количество логических элементов	Количество используемой памяти, байт
1	2	3	4	5	6
1	Nios II/f с RAM 1 Кбайт	129	118	3001	703336
2	Nios II/e с RAM 1 Кбайт	155(25)	141(23)	1411	18240
3	Nios II/f с RAM 14 Кбайт	127	116	3036	174336
4	Nios II/e с RAM 14 Кбайт	158(26)	144(24)	1415	122240
5	Nios II/e с RAM 26 Кбайт	146(24)	133(22)	1589	218240

В версии ядра Nios II/f максимальное количества памяти удалось достичь 14 Кбайт. Повышая размер памяти с ядром Nios II/e удалось достичь

26 Кбайт. Дальнейшее увеличение памяти невозможно в виду ограничения количества блоков памяти в ПЛИС.

Ядро Nios II не включает в себя взаимодействие с периферией и другую функциональную логику, для этого требуется подключать другие блоки. Эти блоки обменивается данными по специальному интерфейсу Avalon. В зависимости от выбранных блоков и настройках ядра количество используемых логических элементов по окончанию компиляции будет разным.

Добавим в прототип блоки PIO для кнопок переключателей и светодиодов, timer и sysid для идентификации SP. В приложение 1 показана конфигурация ядра созданного прототипа. Ниже представлено описание добавленных блоков и основные настройки будем использовать при дальнейшем исследовании:

- PIO блок параллельного вода/вывода. Он позволяет взаимодействовать с периферией, например, светодиоды, переключатели, кнопки. В конфигурации, представленной в приложение 1, используются три блока PIO это leds, switch и button. В настройках указанно бытность шины, тип сигнала (входной или выходной) и использование прерывания.
- Timer блок интервального таймера. Является простейшим счетчиком. Задавая ему значение, он декрементирует его по достижению нуля выдает сигнал окончания работы. Его можно использовать для прерывания. Значение сигнала можно считывать во время работы, поставить на паузу, либо сбросить. В настройках указаны единицы измерения и период этой единицы.
- Sysid блок идентификации SP во время прошивки. В настройках указан 32-х битный идентификатор

На рис 2.7 показана диаграмма верхнего уровня с ново ядром Nios II.

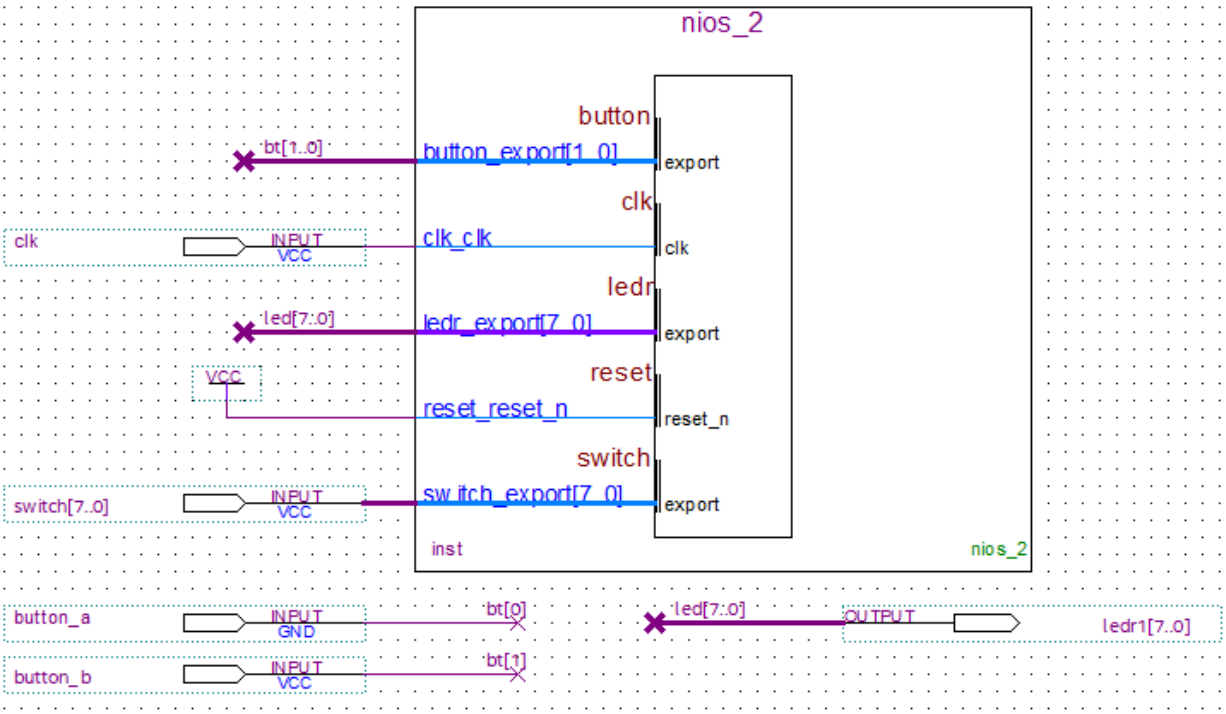


Рис.2.7. Диаграмма верхнего уровня с ядром Nios II

В таблице 2.3 показано результаты компиляции прототипа с добавленными блоками. Перед каждой компиляцией увеличивался размер RAM памяти до выдачи ошибки.

Таблица 2.3

Результаты компиляции прототипа с добавленными блоками

№	Название	Fmax для C7, МГц	Fmax для C8, МГц	Количество логических элементов	Количество используемой памяти, байт
1	Nios II/f с RAM размером 256 байт	120	109	3625	65408
2	Nios II/e с RAM размером 256 байт	148(24)	135(22)	1990	13312
3	Nios II/f с RAM размером 12 Кбайт	121	110	3666	159360
4	Nios II/e с RAM размером 12 Кбайт	142(23)	129(21)	2005	107264
5	Nios II/e с RAM размером 25 Кбайт	134(22)	122(20)	2182	211264

В версии ядра Nios II/f максимальное количества памяти удалось достичь 12 Кбайт, а в версии Nios II/e удалось достичь 25 Кбайт.

В дальнейшем мы будем взаимодействовать только с конфигурацией прототипа представленным в приложение 1. В качестве версии ядра soft–процессора возьмем Nios II/e с RAM размером 25 Кбайт.

2.5. Программирование ядра

Для разработки программного обеспечения для SP Nios существуют инструменты, основанные на компиляторе GNU C/C++:

- Nios II SBT – это интерфейс командной строки.
- Nios II SBT для Eclipse – это графическая среда разработки.

Инструмент Nios II SBT для Eclipse состоит из набора плагинов для IDE Eclipse. Он упрощает работу с Nios использованием графического взаимодействия. На рис.2.8. показан пример графического интерфейса.

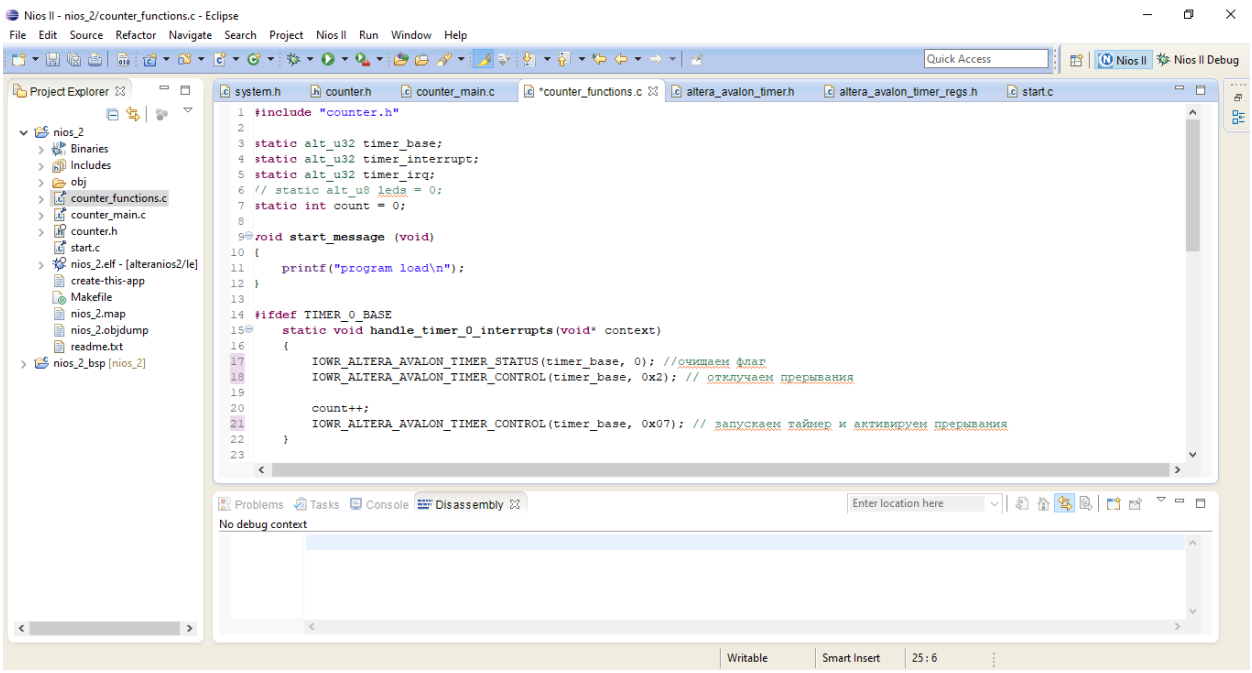


Рис.2.8. Графический интерфейс Nios II SBT

2.6. Моделирование

При отладке с использованием платы часто бывает трудно увидеть глубоко скрытые сигналы в системе. Моделирование RTL решает эту проблему, поскольку позволяет функционально исследовать каждый регистр и сигнал в проекте. Для моделирования использовался симулятор ModelSim.

2.7. Отладка на плате

Для того отладить проект на плате в среде Nios II SBT для Eclipse из написанной программы был скомпилирован hex файл и добавлен для инициализации памяти onchip_memory2_0 в конфигурации ядра SP. После в программе SignalTab были выбраны интересующие сигналы, компиляция проекта, загрузка ядра на плату и захват данных с платы. На рис.2.9 показаны захваченные сигналы.

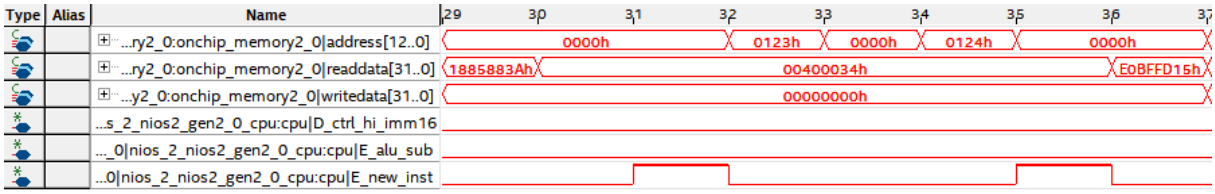


Рис.2.9. Захваченные сигналов с отладочной платы.

2.8. Выполнение программ на отладочной плате

Для выполнения на плате были сознаны программы на ассемблере и на С. На ассемблере написаны программы: чтение из памяти данных чисел и их сложение или умножение, задание состояния светодиодов значениями с кнопок. Код программ представлен в приложение 4. Для измерения быстродействия SP были написаны программы на С: сложения, вычитания, умножения, деления, нахождения, наибольшего и наименьшего числа. Код программ представлен в приложение 3. Все программы были выполнены на одинаковом массиве из 20 случайных чисел с количеством повторений 500 раз. Во время работы было измерено их время выполнения. В таблице 2.4 показаны время, затраченное на выполнение каждой программы.

Таблица 2.4

Время выполнение программ

Название алгоритма	Время, мкс	Среднее время на операцию, нс
Сложение элементов массива	504	50,4
Умножение элементов массива	512	51,2
Деление (одного элемента на остальные)	516	51,6
Поиск наименьшего элемента в массиве	501	50,1

Для выполнения программ был выбран прототип с Nios II/e с RAM размером 25 Кбайт и частотой 134 МГц. Минимальное время выполнения команды 45,5 нс. Из таблицы видно, что во всех алгоритмах время выполнение выше это связанно с издержками во время промежуточных операций. Таким образом было установлено что загруженность SP во время выполнения программ была равна 90%.

ГЛАВА 3. РЕАЛИЗАЦИЯ ПРОТОТИПА SP НА HDL

3.1. Анализ существующих разработок.

Рассмотрим прототип rAVR8 размещенный на сайте OpenCores. Этот SP создали разработчики для ПЛИС компании ООО "ИНПРО ПЛЮС". Он является частично совместимым SP с процессором на архитектуре AVR. Проект выполнен для микросхемы CPLD EPM240T100C5, в ней всего 240 логических элементов.

Блок схема верхнего уровня портов ввода/вывода и состоит из двух блоков: блок ядра SP (rAVR) и блока памяти программ (altufm_none0). На рис.3.1 приведена блок схема.

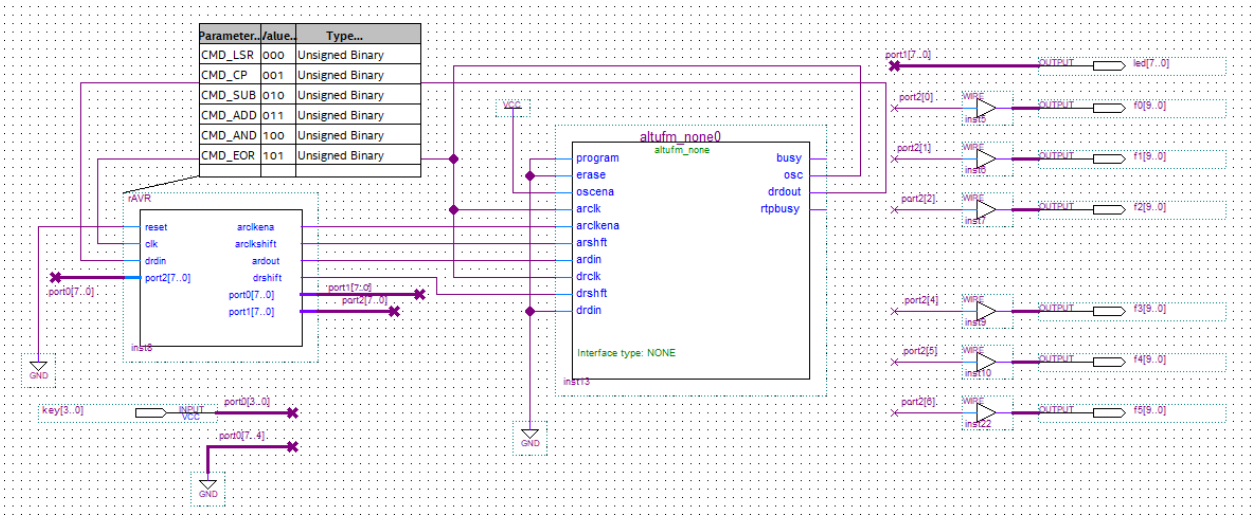


Рис.3.1. Файл верхнего уровня прототипа rAVR8

Ядро SP выполняет 16 команд: 5 команды ветвления, 10 команд Арифметико-логического устройства и 1 команда сдвига. Память данных состоит из 4 регистров общего назначения и 3 регистров выведены на контакты платы для чтения информации с кнопок, вывод информации на светодиоды и управления шаговым двигателем. Максимальная частота Fmax равна 235 МГ, а количество логических ячеек равно 205.

Выполнение команды происходит за два такта. Первым тактом происходит разбор инструкции и чтение из регистров. Вторым выполнение команды на АЛУ и запись в регистры.

Для памяти программ используется флешь память. Чтение одной команды с нее занимает 16 тактов счетчика. В это время мощности процессора простаивают, ожидая команды. Использование RAM в качестве памяти программ позволит уменьшить время простаивания SP. Изменим память программ с флешь на RAM.

3.2. Добавление RAM памяти

В созданном прототипе в качестве памяти программ используется IP ядро RAM:2-port. После компиляции в отчете была получена максимальная частота Fmax 173 МГц и количество логических ячеек 208. На рис.3.2. показана диаграмма верхнего уравня прототип rAVR8 с RAM памятью.

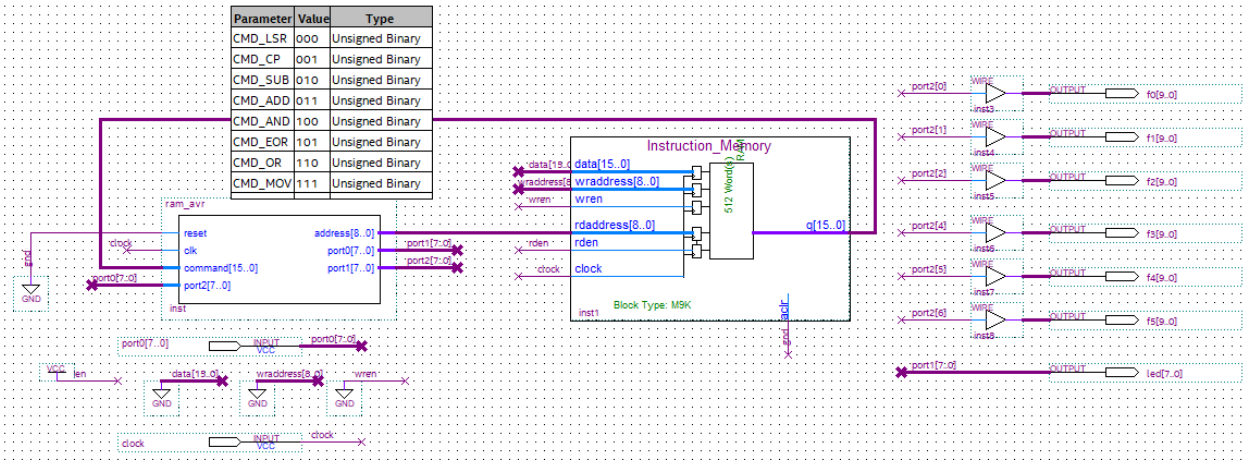


Рис.3.2. Прототип rAVR8 с RAM памятью

После добавления RAM памяти максимальная тактовая частота снизилась, но теперь одна инструкция читается за один такт. Таким образом снизились ожидания SP, что ускорило выполнение инструкций в единицу времени.

Архитектура AVR является устаревшей. Для создания собственной модели SP будем использовать архитектуру MIPS. В качестве отправочного варианта будем использовать документацию на микросхему Мультикор 1892ВМ5Я, разработанную компанией ОАО НПЦ «Элвис». Из нее мы возьмем концепцию 5-ступенчатого конвейера.

Изначально создадим упрощенную версию прототипа, где в архитектуре SP будет описаны 4 команды. Ядро процессора будет состоять из конвейера без обратных связей. Так же добавим отдельную память программ и данных. Для создания регистрового файла будем использовать IP-ядро, так как при реализации памяти на массиве регистров при ее расширении количество логических элементов будет сильно разрастаться. В итоге она будет громоздкой и не удобной.

3.3. Реализация прототипа 8-битного Soft – процессора.

Реализованный прототип имеет конвейерную обработку команд, память программ и память данных. Максимальная частота работы Fmax 107. Количество логических элементов 154.

3.3.1. Описание ядра прототипа

Файл верхнего уровня прототипа SP состоит из 32-х разрядной памяти программ, 8-ми разрядной памяти данных и ядра самого процессора. Обе памяти имеют размер 256 ячеек, они созданы из IP-ядра под названием RAM: 2-port, к каждой памяти привязан mif файл, который задает начальную информацию в ней. На рис.3.3. RTL схема верхнего уровня.

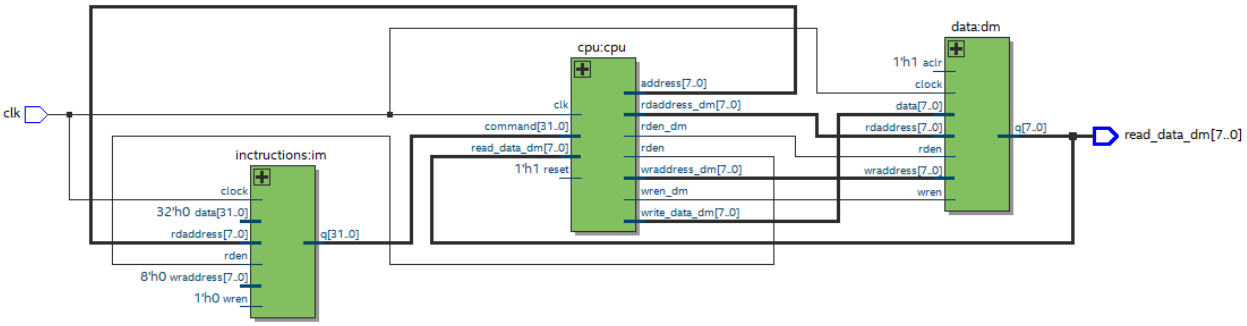


Рис.3.3. RTL схема верхнего уровня 8-битного прототипа

Ядро процессора состоит из счетчика команд, регистрового файла, АЛУ, блока управления конвейером и 5-и ступеней конвейера: I – чтения команды, D – чтения из регистрового файла и проверка условного перехода, E – выполнение операций с АЛУ, M – чтение из памяти данных, W – запись в память данных или регистровый файл. Между ступенями конвейера расположены промежуточные регистры. На рис.3.4. показано устройство ступеней конвейера.

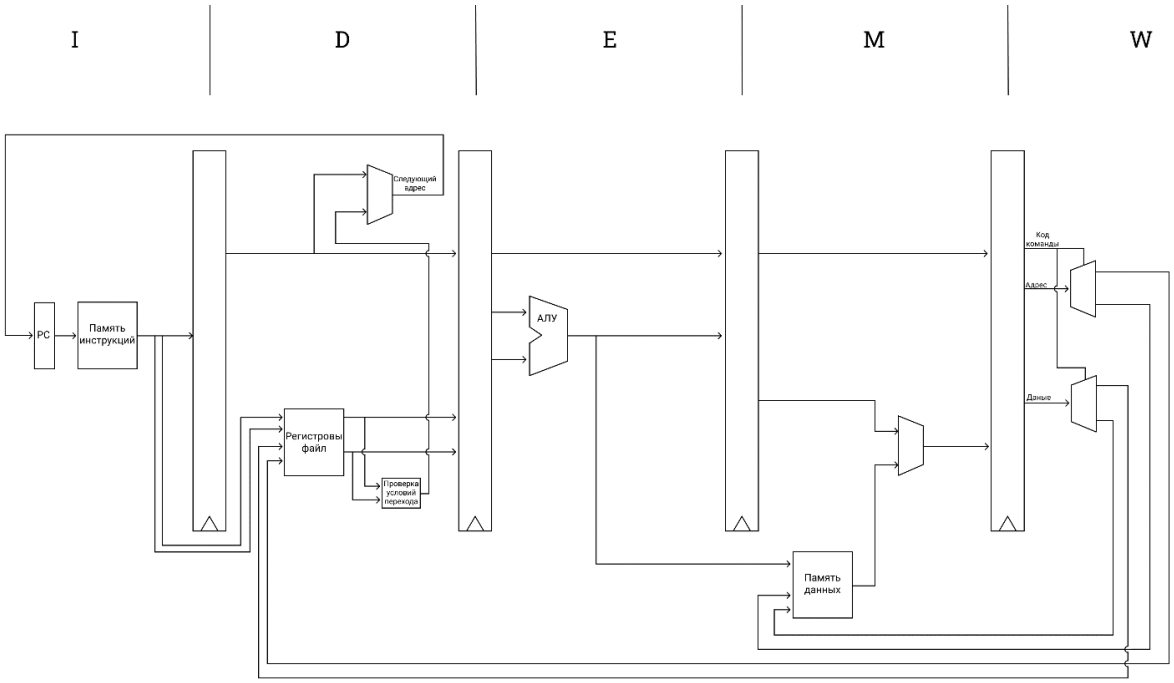


Рис.3.4. Устройство ядра SP с конвейером

Для управления памятью команд в ядре SP организован счетчик команд. По тактовому импульсу clk ему назначается следующий адрес, который подается в память для чтения. В листинги 3.1. показан фрагмент с счетчиком команд.

Листинг 3.1. Счетчик команд

```
always @(posedge clk or posedge reset) begin
    if(reset) begin
        ip <= 0;
    end else begin
      if( need_jump )
      begin
            ip <= jump_addr;
      end else begin
            ip <= ip + 1'b1;
      end
    end
end
endmodule
```

В блоке счетчика команд по тактовому импульсу clk к регистру счётчику ip прибавляется единица, если связь need_jump равняется единицы, то счетчику присваивается адрес перехода. В случае срабатывания сигнала сброса счетчик обнуляется.

После чтения из памяти программ команда выполняется на ступенях конвейера. На первой ступени I происходит чтение из памяти программ инструкции. Из нее извлекается адреса регистров и подаются в регистровый файл для чтения данных. Инструкция предается дальше по конвейеру. В листинге 3.2 приведен фрагмент стадии I.

Листинг 3.2. Фрагмент стадии I

```
num_command_I = opcode_I[31:24];
 if ( num_command_I[2] == 1'b1 ) begin
      rf_rden_1 = 1'b1;
      if( num_command_I[6] == 1'b1 )
           rf_read_addr_1 = opcode_I[23:20];
      else
           rf_read_addr_1 = opcode_I[19:16];
 end else begin
      rf_rden_1 = 1'b0;
      rf_read_addr_1 = 8'b0;
 end
 rf_rden_2 = 1'b1;
 rf_read_addr_2 = opcode_I[15:12];
```

На второй ступени D принимаются данные из регистрового файла, проверяется инструкция на наличие кода условного перехода, если код обнаружен, то регистр jump_addr присваивается адрес следующего перехода и в флаг flag_jump равняется единицы, иначе они оба обнуляются. В листинге 3.3 приведен фрагмент стадии D.

Листинг 3.3. Фрагмент стадии D

```
num_command_D = opcode_D[31:24];
if( num_command_D[7] == 1'b1 & num_command_D[0]
== 1'b1) begin
     jump_addr = opcode_D[7:0];
     flag_jamp = 1'b1;
end else begin
     jump_addr = 8'b0;
     flag_jamp = 1'b0;
end
```

На третьей стадии E из кода команды инструкции вычисляется операция для АЛУ и подается ее в него. Затем рассчитывается результат операции. В зависимости от инструкции результат передается в память данных для чтения или дальше по конвейеру. В листинге 3.4 приведен фрагмент стадии E.

Листинг 3.4. Фрагмент стадии E

```
num_command_E = opcode_E[31:24];
alu_word = opcode_E[7:0];
if( num_command_E[6] == 1'b1 & num_command_E[3]
== 1'b1) begin
     rden_dm = 1'b1;
     rdaddress_dm = alu_result;
end else begin
     rden_dm = 1'b0;
     rdaddress_dm = 4'b0;
end
```

На четвертой стадии M в зависимости от инструкции на следующую стадию конвейера передается данные из регистра результата или из памяти данных. В листинге 3.5 приведен фрагмент стадии M.

Листинг 3.5. Фрагмент стадии M

```
num_command_M = opcode_M[31:24];
if( num_command_M[3] == 1'b1)
     if( num_command_M[6] == 1'b0 )
          reg_write_data_dm_M = reg_rezult_M;
     else
          reg_write_data_dm_M = read_data_dm;
else
     reg_write_data_dm_M = 8'b0;
```

На пятой стадии W происходит запись в регистровый файл или в память данных в зависимости от кода инструкции. В листинге 3.6 приведен фрагмент стадии W.

Листинг 3.6. Фрагмент стадии W

```
num_command_W = opcode_W[31:24];
if( num_command_W[3] == 1'b1) begin
      wren_dm = 1'b0;
      wraddress_dm = 1'b0;
      write_data_dm = 8'b0;
      rf_write = 1'b1;
      rf_write_dest = opcode_W[23:20];
      rf_write_data = reg_write_data_dm_W;
end else begin
      rf_write = 1'b0;
      rf_write_dest = 4'b0;
      rf_write_data = 8'b0;
      if( num_command_W[6] == 1'b1) begin
           wren_dm = 1'b1;
           wraddress_dm = reg_rezult_W;
           write_data_dm = rf_read_data_1_W;
      end else begin
           wren_dm = 1'b0;
           wraddress_dm = 1'b0;
           write_data_dm = 1'b0;
      end
end
```

Между ступенями конвейера расположены промежуточные регистры. При срабатывании сигнала сброса reset конвейер обнуляется. В случае выполнения команды перехода регистры ступени I и D обнуляются тоже. В листинге 3.7 показан фрагмент описания регистров.

Листинг 3.7. Промежуточные регистры

```
always @(posedge clk or posedge reset)
begin
     if(reset)
   begin
          r_opcode_D <= 1'b0;
          r_opcode_E <= 1'b0;
          r_opcode_M <= 1'b0;
          r_opcode_W <= 1'b0;

          rf_read_data_1_E <= 1'b0;
          rf_read_data_2_E <= 1'b0;
          reg_rezult_M <= 1'b0;
          reg_rezult_W <= 1'b0;
   end else begin
          if( need_jump ) begin
               r_opcode_D <= 8'b0;
               r_opcode_E <= 8'b0;
          end else begin
               r_opcode_D <= opcode_I;
               r_opcode_E <= opcode_D;
          end
          r_opcode_M <= opcode_E;
          r_opcode_W <= opcode_M;
          rf_read_data_1_E <= rf_read_data_1;
          rf_read_data_1_M <= rf_read_data_1_E;
          rf_read_data_1_W <= rf_read_data_1_M;
          rf_read_data_2_E <= rf_read_data_2;
          reg_rezult_M <= alu_result;
          reg_rezult_W <= reg_rezult_M;
          reg_write_data_dm_W <= reg_write_data_dm_M;
     end
end
```

Всего Арифметико-логическое устройство выполняет 2 операции сложения 3 чисел и сложение адреса с адресом смещения для памяти данных. В листинге 3.8 показано описание АЛУ.

Листинг 3.8. Устройство АЛУ

```
always @*
begin
     if (num_command_E == 8'h0F) begin
          alu_result = rf_read_data_1_E + rf_read_data_2_E +
alu_word;
```

```
     end else if(num_command_E == 8'h4B | num_command_E ==
8'h47) begin
          alu_result = rf_read_data_2_E + alu_word;
     end else alu_result = 8'b0;
end
```

Всего в прототипе описан один условный переход. На ступени конвейера D после чтения из регистрового файла регистры сравниваются и если они равны и флаг перехода равняется единицы, то переход разрешается (связь need_jump равняется единицы).

Листинг 3.9. Обработка условного перехода

```
assign jump = (rf_read_data_1 == rf_read_data_2) ? 1'b1 :
1'b0;
assign need_jump = ( flag_jamp & jump) ? 1'b1 : 1'b0;
```

3.3.2. Выбор 3-х портовой памяти alt3pram

Для регистрового файла требуется 3-х портовая память. С двумя портами для чтения и одним на запись. В мега функциях Quartus Prime есть 3-х портовая память alt3pram, которая представлена на рис.3.5.

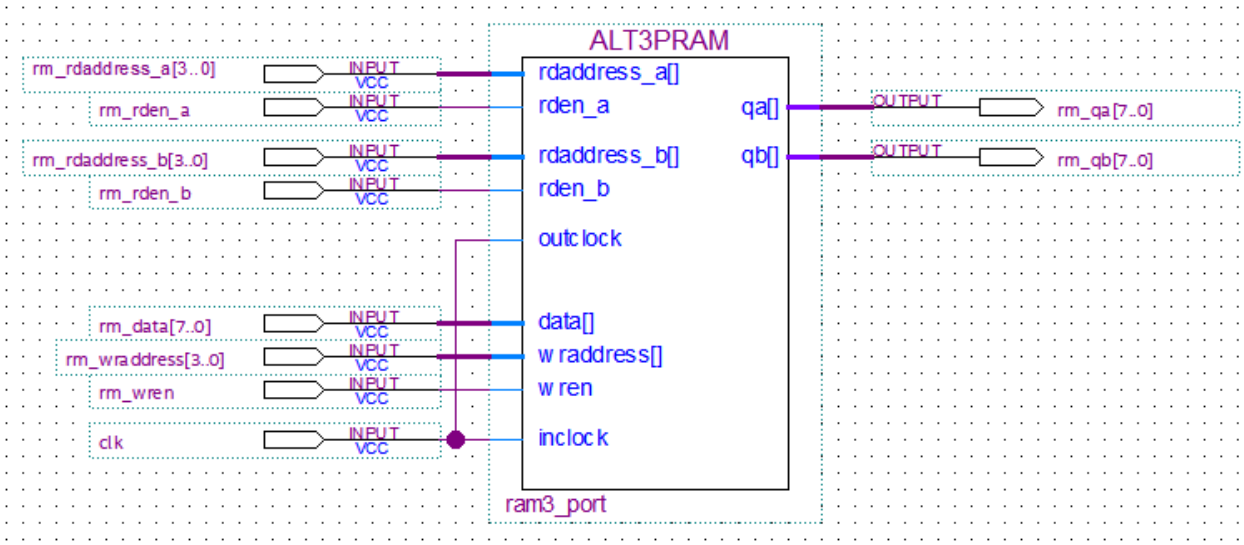


Рис.3.5. Памяти для регистрового файла alt3pram

При компиляции прототипа с этой памятью в отчете была получена максимальная частота Fmax 104 МГц и количество логических ячеек 150. Компания Intel не рекомендует использовать эту память она нужна для поддержки старых проектов [11].

3.3.3. Выбор 3-х портовой памяти на массиве регистров

Рассмотрим 3-х портовую память на массиве регистров. На рис.3.6 представлена RTL диаграмма с памятью на массиве регистров.

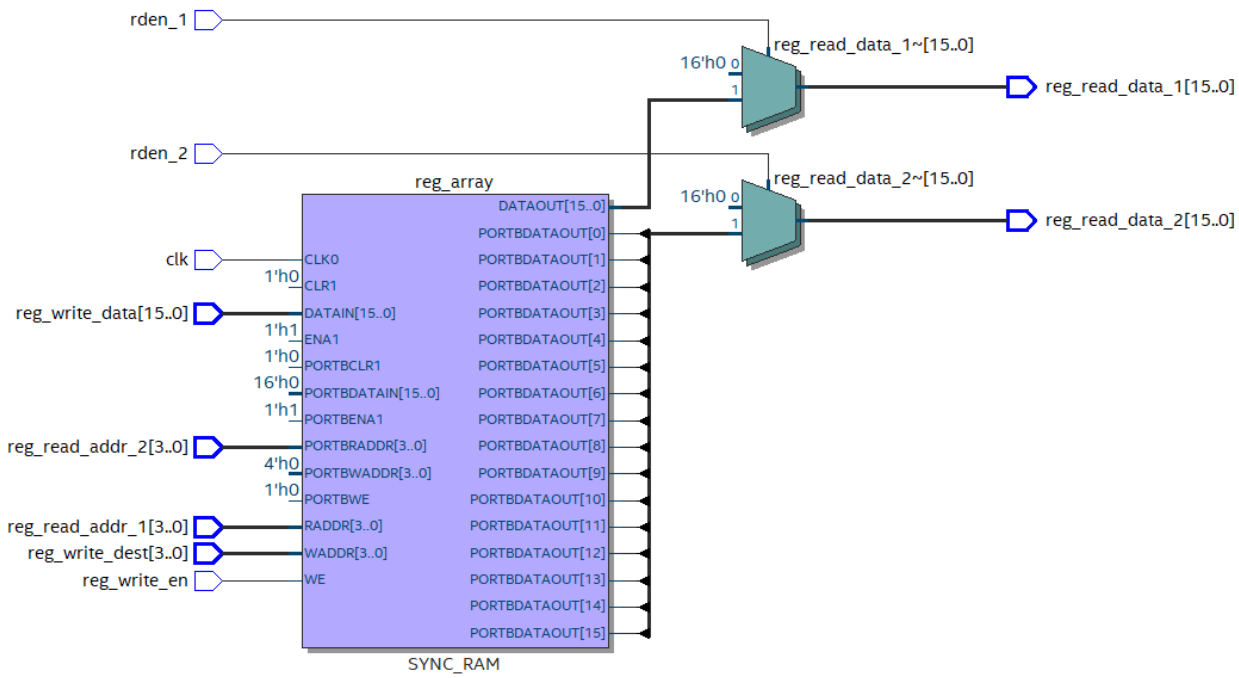


Рис.3.6. RTL диаграмма с памятью на массиве регистров

После компиляции памяти с протопопом максимальная частота Fmax равняется 52, а количество логических элементов равно 342.

3.3.4. Выбор 3-х портовой памяти на двух RAM:2-port

Рассмотрим 3-х портовую память, созданную из двух RAM: 2-port с соединёнными шинами адреса и данных на запись, сигналами на чтение, тактовыми сигналами и сигналами сброса. Количество ячеек в памяти 16, а их длина 8 бит. На картинки 3.7. показана RTL диаграмма этой памяти.

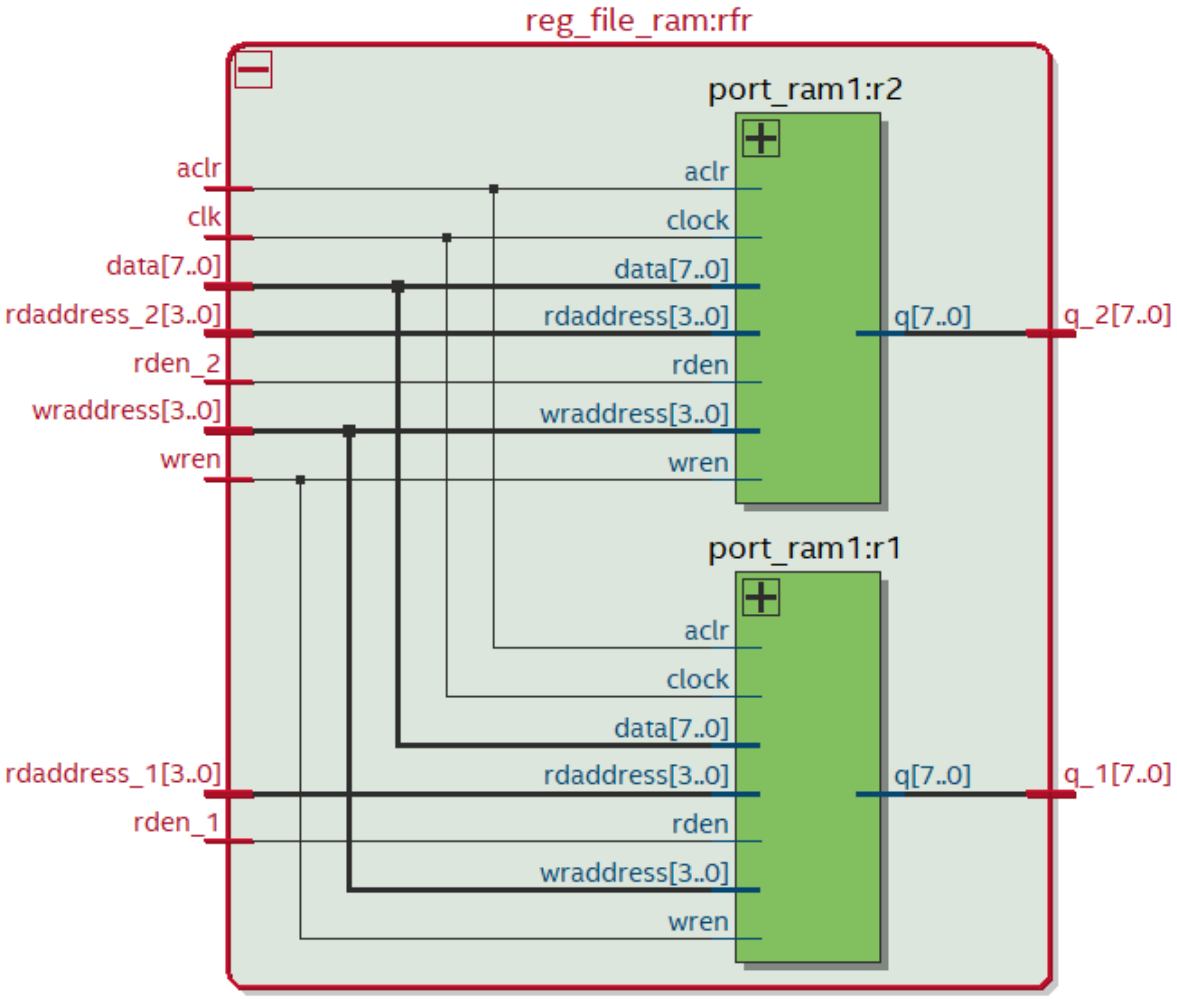


Рис.3.7. RTL диаграмма регистрового файла

При компиляции проекта с этой память в отчете была получена максимальная частота Fmax 107 МГц и количество логических элементов 154.

После компиляции прототипов с различной памятью была создана таблица. В таблице 3.1 характеристики прототипов с разной памятью. Важно отметить что размеры всех вариантов памяти были одинаковы 8 бит длина слова и 16 ячеек.

Таблица 3.1

Характеристики прототипов с разной памятью

Название	Максимальная частота	Количество логических ячеек	Количество ячеек памяти
Массив регистров	52	342	9216
ALT3PRAM	104	150	9472
Две 2-х портовых	107	154	9472

Самый быстры прототип оказался все так же с двумя 2-х портовыми блоками памяти, а самая медленная память оказалась на массивах регистров. Она больше чем в два раза медленнее.

3.3.5. Инструкции

Инструкция в прототипе в общем случае состоит из: 8 бит кода команды, 4 бит адреса регистра назначения, 2-х регистров назначения по 4 бита каждый и константы. На рис.3.8 показан пример инструкции.

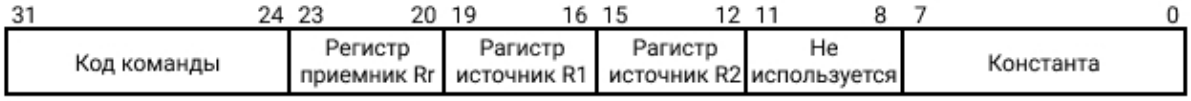


Рис.3.8. Пример инструкции

В коде команды биты с 24 по 26 отвечают за чтение определенного поля в инструкции, например, при наличии единицы в бите 25 на стадии конвейера I в инструкции считывается значение байт с 12 по 15 (регистр источник R2). В таблице 3.2 описаны все действия с битами чтения.

Таблица 3.2

Описание битов чтения

Единца в бите инструкции	Описание
24	Если бит 31 единица, то на стадии конвейера D при выполнение условий перехода считывается значение байт с 0 по 7 (константа), если бит 27 единица, то на стадии конвейера E считывается значение байт с 0 по 7 (константа).
25	На стадии конвейера I в инструкции считывается значение байт с 12 по 15 (регистр источник R2).
26	Если бит 30 единица, то на стадии конвейера I в инструкции считывается значение байт с 20 по 23 (регистр приемник Rr), иначе на стадии конвейера I в инструкции считывается значение байт с 16 по 19 (регистр источник R1).

3.3.6. Реализованные команды

При создании первого прототипа был закодированы 4 команды: сложения, запись в память, чтения из памяти и условный переход. Список закодированных команд представлен в таблице 3.3.

Таблица 3.3

Реализованные команды

Название команды	Код команды	Описание
Добавить	0F	Rr<=R1+R2+const
Записать в память	47	Mem(R2+const)<=Rr
Прочитать из памяти	4B	Rr<=Mem(R2+const)
Условный переход	87	if (R1==R2) переход

Каждая команда обладает своим форматом. Формат команд представлен в таблице 3.4.

Таблица 3.4

Форматы инструкции для данного прототипа

№	Сложение трех чисел и присвоение результата в регистр приемник.					
1	31-24(8)	23-20(4)	19-16(4)	15-12(4)	11-9(4)	8-0(8)
	Код команды	Регистр приёмник Rr	Регистр источник R1	Регистр источник R2	Не используется	Константа
2	Запись в память из регистра назначения					
	Код команды	Не используется	Регистр назначения R1	Регистр источник R2	Не используется	Константа
3	Чтение из памяти и запись результата регистр приемник					
	Код команды	Регистр приемник Rr	Не используется	Регистр источник R2	Не используется	Константа
4	Если значения регистров равны, то переход по адресу					
	Код команды	Не используется	Регистр источник R1	Регистр источник R2	Не используется	Адрес перехода

3.3.7. Тестирование прототипа

Для проверки работоспособности и отладки проекта во время создания схемы использовался симулятор ModelSim. Для симуляции был создан test bench с симуляцией тактового счетчика и файл описания симуляции. На рис.3.9. показано окно ModelSim с симуляцией.

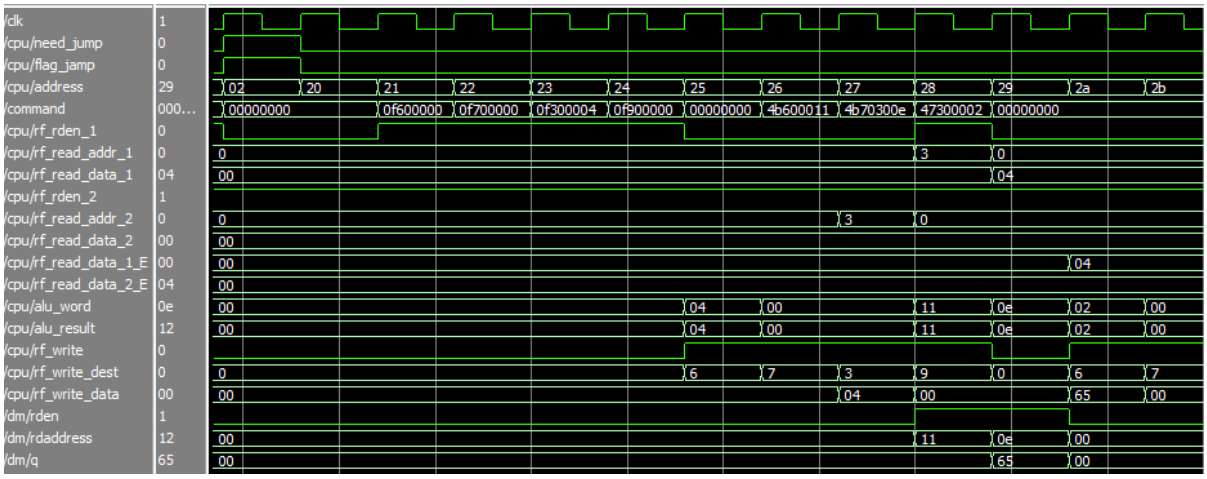


Рис.3.9. Симуляция в ModelSim

Пред симуляцией был заполнен файл начальной инициализации памяти данных в ячейки 11 лежит число 65, а в 12 число 34. На рис.3.7 показан фрагмент файла с инициализацией памяти. На рис.3.10 показан фрагмент файла с инициализацией памяти.

Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
010	58	65	34	00	00	00	00	00	Xe4....

Рис.3.10. Инициализация памяти

Выполненная программа тестирует работоспособность команд процессора. В симуляции на использовалась программа, приведенная в таблице 3.5.

Таблица 3.5

Программа для симуляции прототипа с конвейером

Адрес	Инструкции	Описание	Комментарии
1	2	3	4
20	0F600000	R6<=0	Присвоить R6 значение 0
21	0F700000	R7<=0	Присвоить R7 значение 0
22	0F300004	R3<=4	Присвоить R4 значение 4
23	0F900000	R9<=0	Присвоить R9 значение 0
24	00000000		Пропуск
25	4B600011	R6<=Mem(11)	Чтение из ячейки памяти и сохранение в регистр
26	4B70300E	R7<=Mem(R3+E)	Чтение из ячейки памяти и сохранение в регистр
27	47300002	Mem(2)<=R3	Записать значение регистра в память данных
28	00000000		Пропуск
29	00000000		Пропуск
2A	00000000		Пропуск
2B	0F567000	R5<=R6+R7	Присвоить R5 значение R6+R7
2C	4B900002	R9<=Mem(2)	Чтение из ячейки памяти и сохранение в регистр
2D	87000020	If(R0==R0) переход по адресу 0x20	Переход по адресу 0x20

В программе для симуляции закодированы команды с восьмью нулями. Эти команды нужны для ожидания выполнения прошлых команд, так как в этой версии процессора нет обратных связей. Выполнение команды на запись в память программ или регистровый файл занимает 5 тактов процессора. Поэтому между записью в регистр R7 и чтению из него должно пройти 5 тактов. Конвейер с обратными связями реализован в следующем прототипе.

3.3.8. Выполнение программ на отладочной плате

Для проверки работоспособности программа в таблицы 3.5 была выполнена на отладочной плате в программе SignalTab. На рис.3.11 фрагмент с зачинными сигналами с отладочной платы.

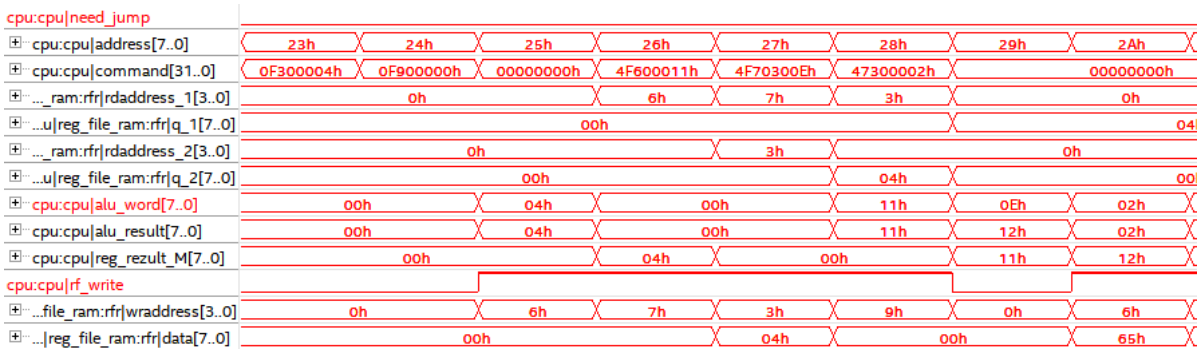


Рис.3.11. Захват сигналов с отладочной платы.

Стоит учитывать, что при добавление нового сигналя для захвата используются логические ячейки.

Размер памяти данных и инструкций равен 8 бит на 256 ячеек, то общий размер каждой памяти 2 Кбайта. Эта память не соответствует заявленным требованиям. Такой размер памяти был выбран для упрощения реализации прототипа.

В реализуемом прототипе без обратных связей предъявляются ограничения к программе. При выполнении команды, где используется регистр результата, результат будет загружен в память данных или регистровый файл через 5 тактов. Нужно учитывать этот фактор для корректного выполнения программы, используя другие адреса.

3.4. Реализация 8-битного прототипа Soft – процессора с обратными связями.

Для создания обратной связи от ступени конвейера E, M и W были протянуты обратные связи к D. В случае если на двух ступенях конвейера окажется один и тот же номер регистра, то в этом случае возникнет конфликт. Поэтому наивысший приоритет присвоения значения от ступени конвейера имеет E, затем M и потом W.

Для выполнения операции записи или чтения с регистрами требуется один такт, то есть чтобы сначала записать, а потом прочитать значение регистра требуется два такта. Поэтому что бы убрать пропуск в такт была создана дополнительная обратная связи для каждого регистра.

При выполнении команд чтения или записи в память данных на ступени E обратной связи не будет, так как адрес и данные, нужные для обратной связи формируются на стадии M. На рис.3.12 показано устройство ядра SP с обратными связями.

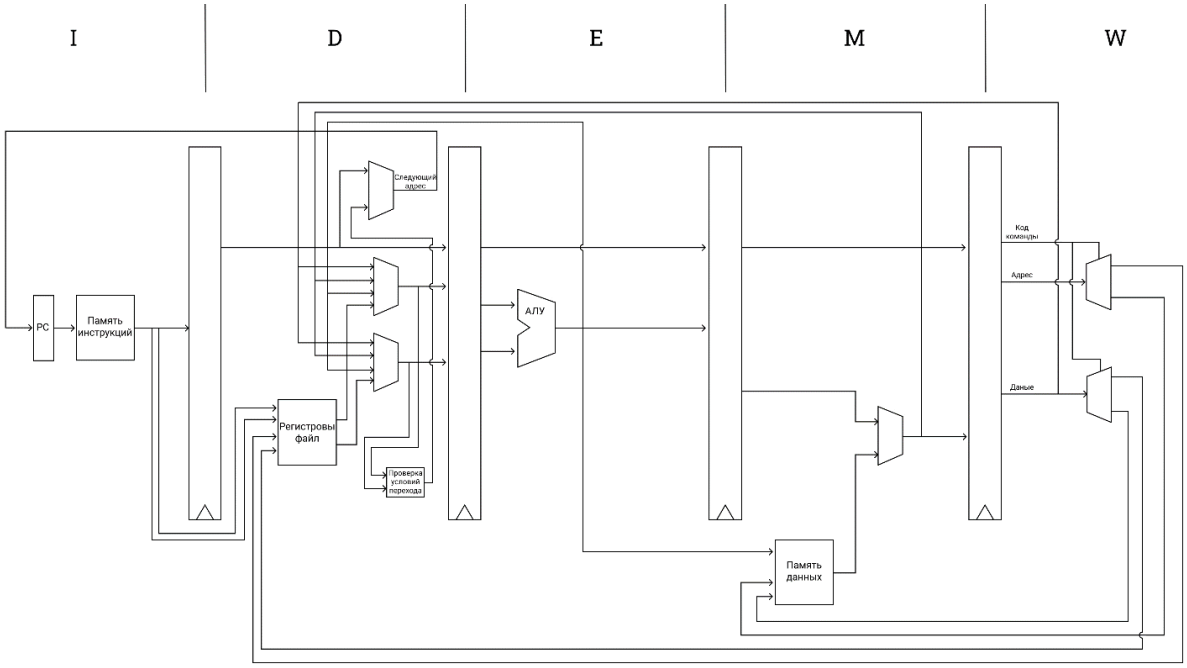


Рис.3.12. Схема ядра SP с обратными связями

Максимальная частота работы Fmax равняется 75 МГц, а количество логических элементов 234. Теперь SP может работать без ожидания записи результата в регистровый файл или память данных. Для примера выполним в ModelSim программу, представленную в таблице 3.6.

Таблица 3.6

Программа записи чисел в память данных

Адрес	Инструкции	Описание	
1	2	3	4
50	0F100000	R1<=0	Присвоить R1 значение 0
51	0F200008	R2<=8	Максимальное число повторений
52	0F300000	R3<=0	Счетчик повторений
53	47103008	Mem(R1)<=R3+8	Записать в память
54	0F110001	R1<=R1+1	Инкрементировать R1
55	0F330001	R3<=R3+1	Инкрементировать R3

Оконч. табл. 3.6

1	2	3	4
56	87032080	If(R3==R2) переход по адресу 0x80	Условный переход. Завершение программы
57	87000053	If(R0==R0) переход по адресу 0x53	Условный переход

Программа записывает в память данных числа от 0x3 до 0xA в ячейки памяти от 0x8 до 0xF. На рис.3.13. состояние памяти данных до и после работы программы. До начала работы программы память данных была инициализирована нулями. Красный цвет. После работы в ячейки с 0x8 по 0x10 были заполнены. Зеленый цвет.

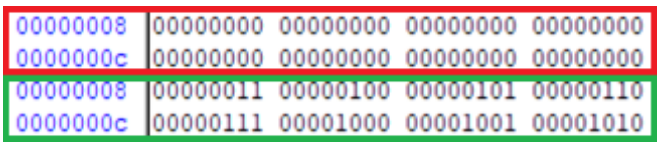


Рис.3.13. Память данных до и после выполнения программы

После реализации обратных связей заметно упала тактовая частота процессора с 107 МГц до 75 МГц. Это связанно с возросшей нагрузкой на ступень D. Прототип может работать в быстродействие 75 млн. команд/с. Такое быстродействие приближено к требованиям, но все равно ниже их.

В этой версии прототипа реализованы четыре команды, которые были реализованы в прошлой версии. Дополним SP новыми командами.

3.5. Реализация 8-битного прототипа с дополнением команд

При дополнении этого прототипа были закодированы 7 команды: сложения, вычитание, инкремент, запись в память, чтения из памяти и два условных перехода. Список закодированных команд представлен в таблице 3.7.

Таблица 3.7

Реализованные команды

Название команды	Код команды	Описание
Добавить	0F	Rr<=R1+R2+const
Вычесть	16	Rr<=R1-R2
Инкремент	1B	Rr<=R1+1
Записать в память	47	Mem(R2+const)<=Rr
Прочитать из памяти	4B	Rr<=Mem(R2+const)
Условный переход	87	if (R1==R2) переход
Условный переход	9F	if (R1==R2) переход

Каждая команда обладает своим форматом. Формат команд представлен в таблице 3.8.

Таблица 3.8

Форматы инструкции для данного прототипа

№	Сложение трех чисел и присвоение результата в регистр приемник.					
1	31-24(8)	23-20(4)	19-16(4)	15-12(4)	11-9(4)	8-0(8)
	1	2	3	4	5	6
	Код команды	Регистр приёмник Rr	Регистр источник R1	Регистр источник R2	Не используется	Константа
2	Вычитание и присвоение результата в регистр приемник.					
	Код команды	Регистр приёмник Rr	Регистр источник R1	Регистр источник R2	Не используется	
3	Инкремент и присвоение результата в регистр приемник.					
	Код команды	Регистр приёмник Rr	Не используется	Регистр источник R2	Не используется	
4	Запись в память из регистра назначения					
	Код команды	Не используется	Регистр назначения R1	Регистр источник R2	Не используется	Константа

Оконч. табл. 3.8

5	Чтение из памяти и запись результата регистр приемник					
	1	2	3	4	5	6
	Код команды	Регистр приемник Rr	Не используется	Регистр источник R2	Не используется	Константа
6	Если значения регистров равны, то переход по адресу					
	Код команды	Не используется	Регистр источник R1	Регистр источник R2	Не используется	Адрес перехода
7	Если регистр 1 больше регистра 2, то переход по адресу					
	Код команды	Не используется	Регистр источник R1	Регистр источник R2	Не используется	Адрес перехода

Максимальная тактовая частота SP равняется 67 МГц, а количество логических элементов равно 282. По сравнению с предыдущим прототипом быстродействие SP снизилось и составляет 67 млн. команд/с.

Созданный прототип при 32 разрядной памяти данных имеет 8 разрядную память данных и регистровый файл. Увеличим разрядность данных до 32, что бы посмотреть частоту SP.

3.6. Реализация 32-разрядного прототипа Soft – процессора

Прототип SP был реализован из прототипа с обратными связями. В этой версии были увеличены разрядность памяти данных, регистрового файла и оптимизированы команды под эту разрядность. В инструкции увеличена разрядность константы с 8 бит до 12 бит. Расположение кода команды и регистров осталось таким же. На рис.3.14. показан общий пример инструкции.

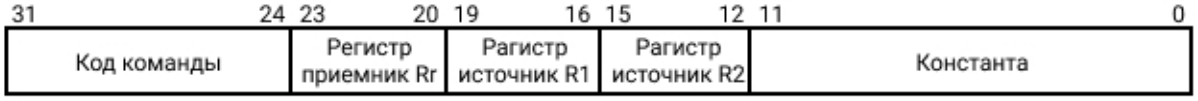


Рис.3.14. Общий пример инструкции 32 разрядного SP

Так как теперь все регистры АЛУ имеют длину 32 бит, а места в инструкции хватает только для константы длиной 12 бит, то при выполнении команды на ступни конвейера E регистр (alu_world) дополняется нулями в старших разрядах. В листинге 3.11. Изменения фазы E.

Листинг 3.11 Изменение фазы E для 32-разрядного конвейера.

```
alu_word = {20'b0, r_opcode_E[11:0]};
```

Прототип имеет максимальную частоту Fmax 57 МГц, быстродействие 57 млн. команд/с и количество логических элементов 602. Для проверки работоспособности SP выполним программу, приведенную в таблице 3.9, которая читает числа с адреса 0x28 до 0x2F, складывает их, после записывает в ячейку 0x20.

Таблица 3.9

Программа чтения с памяти чисел и их сложения

Адрес	Инструкции	Описание	Комментарии
60	0F100000	R1<=0	Обнуления регистра с результатом
61	0F200008	R2<=8	Максимальное число повторений
62	0F300000	R3<=0	Счетчик повторений
63	4B403028	R4<=Mem(R3+28)	Чтение из памяти данных
64	0F330001	R3<=R3+1	Инкремент R3
65	0F110001	R1<=R1+1	Инкремент R1
66	87032070	If(R3==R2) переход по адресу 0x70	Условный переход
67	87000063	If(R0==R0) переход по адресу 0x63	Условный переход
68-6F			Пропуск
70	47010020	Mem(20) <= R1 (Завершение программы)	Условный переход. Завершение программы

Перед выполнением был заполнен файл начальной инициализации памяти данных. На рис.3.15. показан фрагмент файла.

Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
028	00003000	00004000	00005000	00006000	00007000	00008000	00009000	0000A000	.....

Рис 3.15. Инициализация памяти

После работы программы в ячейка памяти 0x20 была заполнена числом 0x34. На рис.3.16 показан фрагмент памяти данных после работы программы.

```
00000020 00034000 00000000 00000000 00000000
00000024 00000000 00000000 00000000 00000000
00000028 00003000 00004000 00005000 00006000
0000002c 00007000 00008000 00009000 0000a000
```

Рис.3.16. Фрагмент памяти данных

В этой версии SP максимальная частота и быстродействие снизилось по сравнения с предыдущим прототипом. Так как длина ячейки памяти теперь равна 32 бита, то общий размер памяти данных или программ равен 8 Кбайт каждая.

3.7. Дополнение командами прототипа 32-разрядного Soft – процессора

В этой версии SP реализованы 14 команд: 7 операций с АЛУ, 4 операции сдвига и 3 условных перехода. В таблице 3.10. представлен весь список команд, а в приложение 2 формат команд.

Таблица 3.10

Реализованные команды

Название	Код команды	Описание
1	2	3
Операции АЛУ		
Сложение	0F	Rr<=R1+R2+const
Вычитание	17	Rr<=R1-R2
Инкремент	1B	Rr<=R1+1
И	26	Rr<=R1 or R2

Оконч. табл. 3.10

1	2	3
Или	2E	Rr<=R1 and R2
Исключающее или	36	Rr<=R1 xor R2
Не	3E	Rr<= not R2
Операции с памятью		
Записать в память	47	Mem(R2+const)<=Rr
Прочитать из памяти	4B	Rr<=Mem(R2+const)
Операции с сдвигами		
Логически сдвиг в лево	66	Rr<=R1<<R2
Логически сдвиг в право	6E	Rr<=R1>>R2
Условные переходы		
Регистры равны	87	If(R1==R2) переход
Не равны	8F	If(R1!=R2) переход
Регистр 1 больше или равен регистра 2	9F	If(R1>=R2) переход

В этой версии процессора максимальная частота равняется Fmax 53 МГц, быстродействие 53 млн. команд/с и количество логических элементов равно 1179. Чтобы дополнить SP командами были изменены АЛУ, обработка команд на ступенях конвейера E, M, W. В них были добавлены обработка команд сдвига и логические операции. Для создания новых условных переходов были добавлены условия перехода.

3.8. Дополнение командами умножения и деления прототипа 32-разрядного Soft – процессора

В этой версии SP реализованы команды умножения и деления. Всего закодировано 18 команд. В таблице 3.11 показаны добавленные команды.

Таблица 3.11.

Добавленные команды

Название	Код команды	Описание
Операции умножения и деления		
Умножить	56	Rr<=R1*R2
Поделить	5E	Rr<=R1/R2

Также для каждой команды существует свой формат инструкций, который показан в таблице 3.12. Полный список команд представлен в приложение 2.

Таблица 3.12.

Форматы инструкции для данного прототипа

1	Умножение				
	31-24(8)	23-20(4)	19-16(4)	15-12(4)	11-0(12)
	Код команды	Регистр приёмник Rr	Регистр источник R1	Регистр источник R2	Не используется
2	Деление				
	Код команды	Регистр приёмник Rr	Регистр источник R1	Регистр источник R2	Не используется

После добавления команд в отчет о компиляции показано что теперь используется блоков умножителей. Этот прототип имеет максимальную частоту Fmax 47 МГц, быстродействие 47 млн команд/с и число логических элементов 2337. Для тестирования работоспособности прототипа была выполнена программа, в которой регистр R1 было присвоено значение 0x3, а в регистр R2 значение 0x1F. После были вызваны все реализованные команды. Так же были выполнены программы чтения из памяти с сложением, умножением и делением 16 чисел, код программы приведен в таблице 3.10 и записи с память 16 чисел после, код программы приведен в таблице 3.7. Результат симуляции показал, что результаты операций ожидаемые.

Упомянутые выше максимальные частоты были измерены для ПЛИС EP4CE6E22C8N, которая используется в отладочной плате. Параллельно были измерены максимальные частоты для ПЛИС EP4CE22E22C7. В таблице 3.13 показаны характеристики всех прототипов. В скобках

показаны частоты, поделенные на 5, так как в прототипах без обратных связей может выполняться одна команда за 5 тактов.

Таблица 3.13

Характеристики всех прототипов собственной реализации

№	Название	Fmax для C7, МГц	Fmax для C8, МГц	Количество логических ячеек	Количество ячеек памяти
1	rAVR8	258(16)	235(14)	224	0 (исп. FLASH CPLD)
2	rAVR8 с RAM	190	173	208	8192
3	8-битный прототип с памятью в виде массива регистров	57	52	342	9216
4	8-битный прототип с ядром памяти alp3pram	114	104	150	9472
5	8-битный прототип с двумя 2-х портовыми RAM	117	107	154	9472
6	8-битный прототип с обратными связями	82	75	234	10496
7	8-битный прототип с добавленными командами	73	67	282	10496
8	32-битный прототип	62	57	602	17408
9	32-битный прототип с добавленными командами	58	53	1179	17408
10	32-битный прототип с добавленными умножением и делением	51	47	2337	17408

Из таблицы 3.13 видно, что с каждым добавление новой логики в прототип наблюдается снижение максимальной частоты. Самый быстрый

прототип из созданный прототип под номером 6 имеет быстродействие для ПЛИС EP4CE22E22C7 82 млн. команд/с.

Прототип под номером 2 и 10 почти идентичный набор команд. Но максимальная частота у прототипа 2 выше. Это связанно с тем, что прототип 10 имеет полностью 32-разрядный SP имеющий 16 регистров, отдельную память инструкций и данных. А прототип 2 имеет 7 регистров размером в 8 бит и 16 битную память инструкций. При сравнении прототипов 1 и 10 SP под номером 10 обладает более высоким быстродействием 51 млн. команд/с.

Заданная частота процессора ADSP-2184N равна 80 МГц по результатам оказывается выше чем по результатам спроектированного процессора. За то операция сложения на ADSP занимает 7 тактов, у собственного прототипа SP 5 тактов. Что на 40% больше.

При реализации прототипов не была реализована ручная трассировка элементов на кристалле в Chip Planer, а также не был проведен временной анализ в Timing Analyzer. Выполнив эти два пункта возможно ускорение прототипов.

ГЛАВА 4. СРАВНЕНИЕ ХАРАКТЕРИСТИК ВАРИАНТОВ ПРОТОТИПОВ SOFT – ПРОЦЕССОРОВ

При сравнении SP собственной реализации с Nios II, то второй вариант имеет лучшие характеристики. Если для ПЛИС EP4CE22E22C7 выбрать быструю версию Nios II/f, представленный в таблице 2.3 под номером 3, то он будет использовать больше половины логических ячеек, а именно 3666 и обладать быстродействием примерно 121 млн. команд/с. Экономичная версия Nios II/e, представленная в таблице 2.3 под номером 4, обладает быстродействием примерно 23 млн. команд/с и количеством логических ячеек 2003, что треть от общего количества. По сравнению с SP представленного в таблице 3.13 под номером 10, который обладает

быстродействием 51 млн команд/с. и количеством логических ячеек 2333. Версия Nios II/e обладает меньшим размером, но уступает быстродействие.

Так как для на собственной модели SP нет совместимого ассемблера, то для написания программ требуется вручную заполнять машинными кодами mif файл и перекомпилировать ядро заново. В Nios II есть специальная графическая среда разработки Nios II SBT для Eclipse, с встроенным компилятором и ассемблером, в котором можно писать программы на ассемблере, C, C++. Стоит упомянуть что задача разработки ассемблера и совместимости с существующим ассемблером не входила задание.

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы изучены: состав модуля управления MB-1000M, характеристики процессора ADSP-2185N, характеристики Soft-процессора, компоненты среды проектирования Quartus Prime 20.1.

Проведен анализ возможных путей построения прототипов Soft-процессора по требованиям к нему взамен ПЦОС ADSP-2185N. В результате анализа было принято решение о разработке прототипа с использованием NIOS II gen 2 и несколько прототипов на языке Verilog процессоров RISC - архитектуры с усеченными ядрами.

В первой части работы было создано несколько, все более сложных, вариантов прототипа на основе NIOS II gen 2, предлагаемый фирмой Intel. Спроектирована структура процессора, написаны тестовые программы на языке С и ассемблере, проведено моделирование и проверка на плате miniDiLaB с ПЛИС Cyclone IV EP4CE6E22C8N с помощью SignalTab. Проект позволил определить сложность, быстродействие и производительность процессора.

Вариант для Nios II/e занимает 2180 ячейки, частота 134 (а по факту 22) МГц на программах тестирования показал 90% загрузку арифметическими операциями в Nios II/e должна рассчитываться как 134/6. Вариант для Nios II/f занимает 3666 ячеек, частота 121 МГц. Удовлетворительным является только вариант Nios II /f (платная версия).

Во второй части работы было создано несколько вариантов прототипов с усеченными ядрами процессоров AVR8 и MIPS32. Проектирование осуществлялось от простого 8-разрядного процессора к процессору 32-разрядному со сложным конвейерным ядром.

Спроектированы структуры прототипов на языке Verilog, определена система команд, написаны тестовые программы в кодах, проведено моделирование и проверка на плате miniDiLaB с ПЛИС Cyclone IV EP4CE6E22C8N с помощью SignalTab. Проект позволил определить сложность при определенном функциональном усечении и быстродействие для каждого варианта.

Для 8-битных протопопов рабочая частота занимает 73 МГц, а для 32-битных прототипов число ячеек от 602 до 2337, частота от 51 до 62 МГц. В частности, для 32-бипного прототипа с шестнадцатью командами, во том числе умножением и деление, 2337 ячеек, частота 51 МГц.

Проектирование показало, что производительность разработанных прототипов Soft-процессора соизмеримы с процессором ADSP-2185N по быстродействию, а ресурсы в ПЛИС EP4CE22 или 5CEFA4 будут использованы до 30%.

Практические результаты работы могут быть интересны фирмам, которые стремятся к уменьшению номенклатуры и типов процессоров и контроллеров в их изделиях.

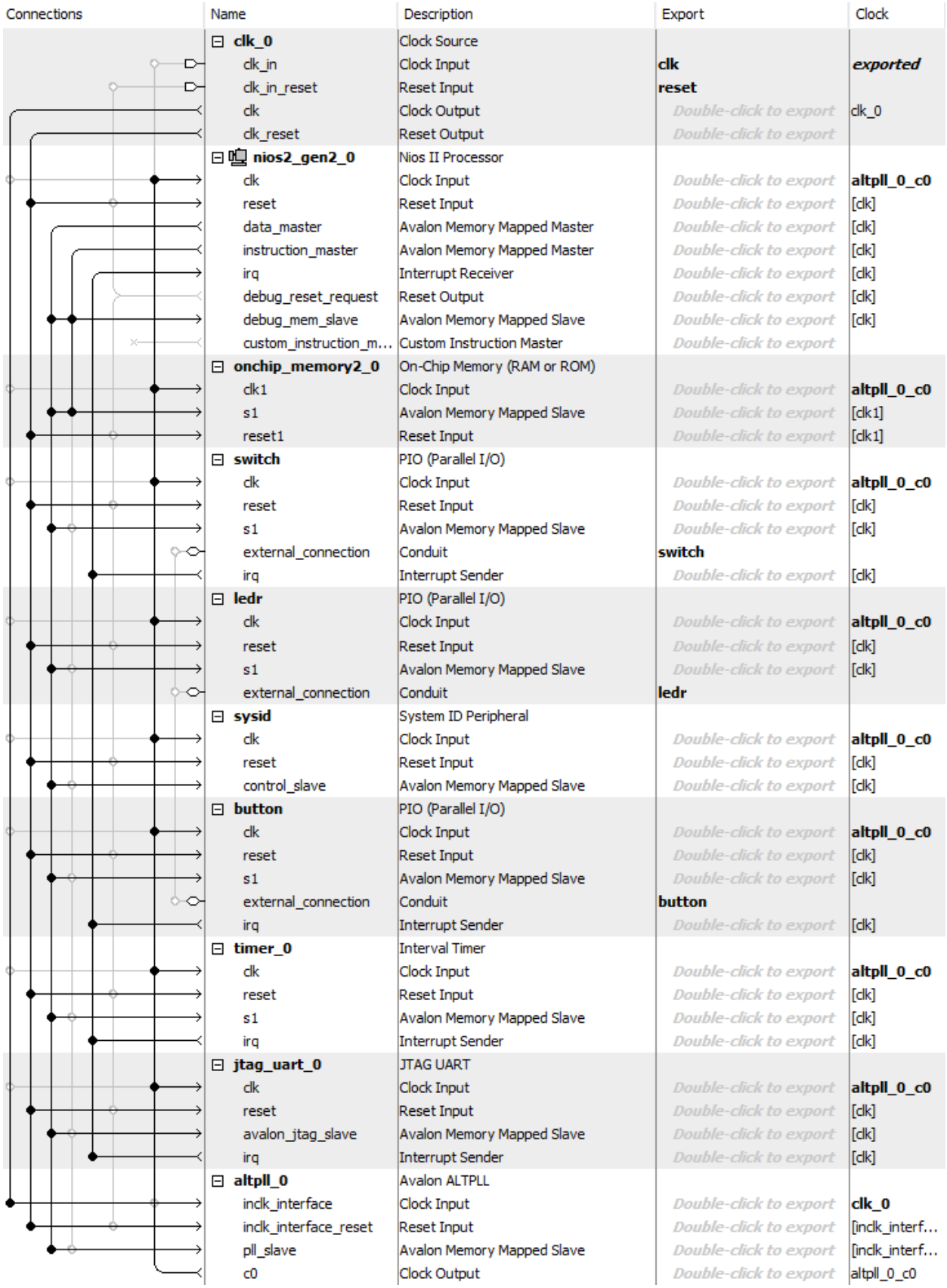
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Дэвид М. Харрис и Сара Л. Харрис, Цифровая схемотехника и архитектура компьютера, Morgan Kaufman, 2013 — 22 с.
2. ADSP-2185N. Electronic resource, Analog Devices. — URL: https://www.analog.com/ru/products/adsp-2185n.html#product-overview
3. Architecture Brief. Electronic resource, Intel. — URL: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ab/ab1_soc_fpga.pdf
4. Donald G. Bailey., Image Processing Using FPGAs, Switzerland, 2019 — 2 с.
5. Nios II Classic Processor Reference Guide. Electronic resource, Intel. — URL: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/nios2/n2cpu_nii5v1.pdf
6. Main page. Electronic resource, OpenCores. — URL: https://opencores.org/
7. Reduced AVR Core for CPLD. Electronic resource, OpenCores. — URL: https://opencores.org/projects/avr8
8. Главная страница. Электронный ресурс, Мультикор. — URL: https://multicore.ru/
9. Руководство пользователя. Микросхема интегральная 892ВМ5Я. Электронный ресурс, Мультикор. — URL: https://multicore.ru/mc/data_sheets/Manual_1892VM5YA_221113.pdf
10. Nios II Processor Reference Guide. Electronic resource, Intel. — URL: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/nios2/n2cpu-nii5v1gen2.pdf
11. Alt3pram Megafunction. Electronic resource, Intel. — URL: https://www.intel.com/content/www/us/en/programmable/quartushelp/17.0/hdl/mega/mega_file_alt3pram.htm

ПРИЛОЖЕНИЕ 1

КОНФИГУРАЦИЯ ЯДРА NIOS II

П.1.1. Конфигурация ядра Nios II.



Connections	Name	Description	Export	Clock
	clk_0	Clock Source		
	clk_in	Clock Input	**clk**	***exported***
	clk_in_reset	Reset Input	**reset**	
	clk	Clock Output	*Double-click to export*	clk_0
	clk_reset	Reset Output	*Double-click to export*	
	nios2_gen2_0	Nios II Processor		
	clk	Clock Input	*Double-click to export*	**altpll_0_c0**
	reset	Reset Input	*Double-click to export*	[clk]
	data_master	Avalon Memory Mapped Master	*Double-click to export*	[clk]
	instruction_master	Avalon Memory Mapped Master	*Double-click to export*	[clk]
	irq	Interrupt Receiver	*Double-click to export*	[clk]
	debug_reset_request	Reset Output	*Double-click to export*	[clk]
	debug_mem_slave	Avalon Memory Mapped Slave	*Double-click to export*	[clk]
	custom_instruction_m...	Custom Instruction Master	*Double-click to export*	
	onchip_memory2_0	On-Chip Memory (RAM or ROM)		
	clk1	Clock Input	*Double-click to export*	**altpll_0_c0**
	s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk1]
	reset1	Reset Input	*Double-click to export*	[clk1]
	switch	PIO (Parallel I/O)		
	clk	Clock Input	*Double-click to export*	**altpll_0_c0**
	reset	Reset Input	*Double-click to export*	[clk]
	s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]
	external_connection	Conduit	**switch**	
	irq	Interrupt Sender	*Double-click to export*	[clk]
	ledr	PIO (Parallel I/O)		
	clk	Clock Input	*Double-click to export*	**altpll_0_c0**
	reset	Reset Input	*Double-click to export*	[clk]
	s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]
	external_connection	Conduit	**ledr**	
	sysid	System ID Peripheral		
	clk	Clock Input	*Double-click to export*	**altpll_0_c0**
	reset	Reset Input	*Double-click to export*	[clk]
	control_slave	Avalon Memory Mapped Slave	*Double-click to export*	[clk]
	button	PIO (Parallel I/O)		
	clk	Clock Input	*Double-click to export*	**altpll_0_c0**
	reset	Reset Input	*Double-click to export*	[clk]
	s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]
	external_connection	Conduit	**button**	
	irq	Interrupt Sender	*Double-click to export*	[clk]
	timer_0	Interval Timer		
	clk	Clock Input	*Double-click to export*	**altpll_0_c0**
	reset	Reset Input	*Double-click to export*	[clk]
	s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]
	irq	Interrupt Sender	*Double-click to export*	[clk]
	jtag_uart_0	JTAG UART		
	clk	Clock Input	*Double-click to export*	**altpll_0_c0**
	reset	Reset Input	*Double-click to export*	[clk]
	avalon_jtag_slave	Avalon Memory Mapped Slave	*Double-click to export*	[clk]
	irq	Interrupt Sender	*Double-click to export*	[clk]
	altpll_0	Avalon ALTPLL		
	inclk_interface	Clock Input	*Double-click to export*	**clk_0**
	inclk_interface_reset	Reset Input	*Double-click to export*	[inclk_interf...
	pll_slave	Avalon Memory Mapped Slave	*Double-click to export*	[inclk_interf...
	c0	Clock Output	*Double-click to export*	altpll_0_c0

ПРИЛОЖЕНИЕ 2

ФОРМАТ ИНСТРУКЦИЙ ДЛЯ 32-РАЗРЯДНОГО ПРОТОТИПА

П.2.1. Форматы инструкции для 32-разрядного прототипа без умножения и деления.

№	Сложение трех чисел и присвоение результата в регистр приемник					
1	31-24(8)	23-20(4)	19-16(4)	15-12(4)	11-0(12)	31-24(8)
	1	2	3	4	5	6
	Код команды	Регистр приёмник Rr	Регистр источник R1	Регистр источник R2	Константа	
2	Вычитание и присвоение результата в регистр приемник					
	Код команды	Регистр приёмник Rr	Регистр источник R1	Регистр источник R2	Не используется	Код команды
3	Инкремент и присвоение результата в регистр приемник					
	Код команды	Регистр приёмник Rr	Не используется	Регистр источник R2	Не используется	
4	Операция «или» и присвоение результата в регистр приемник					
	Код команды	Регистр приёмник Rr	Регистр источник R1	Регистр источник R2	Не используется	
5	Операция «и» и присвоение результата в регистр приемник					
	Код команды	Регистр приёмник Rr	Регистр источник R1	Регистр источник R2	Не используется	
6	Операция исключающее «или» и присвоение результата в регистр приемник					
	Код команды	Регистр приёмник Rr	Регистр источник R1	Регистр источник R2	Не используется	
7	Инверсия и присвоение результата в регистр приемник					
	Код команды	Регистр приёмник Rr	Не используется	Регистр источник R2	Не используется	

8	Запись в память из регистра назначения				
	1	2	3	4	5
	Код команды	Не используется	Регистр назначения R1	Регистр источник R2	Константа
9	Чтение из памяти и запись результата регистр приемник				
	Код команды	Регистр приемник Rr	Не используется	Регистр источник R2	Константа
10	Логический сдвиг в лево				
	Код команды	Регистр приёмник Rr	Регистр источник R1	Регистр источник R2	Не используется
11	Логический сдвиг в право				
	Код команды	Регистр приёмник Rr	Регистр источник R1	Регистр источник R2	Не используется
12	Условный переход по адресу если R1==R2				
	Код команды	Не используется	Регистр источник R1	Регистр источник R2	Адрес перехода
13	Условный переход по адресу если R1==R2				
	Код команды	Не используется	Регистр источник R1	Регистр источник R2	Адрес перехода
14	Условный переход по адресу если R1>=R2				
	Код команды	Не используется	Регистр источник R1	Регистр источник R2	Адрес перехода

П.2.2. Форматы инструкции умножения и деления для 32-разрядного прототипа.

1	Умножение				
	31-24(8)	23-20(4)	19-16(4)	15-12(4)	11-0(12)
	Код команды	Регистр приёмник Rr	Регистр источник R1	Регистр источник R2	Не используется
2	Деление				
	Код команды	Регистр приёмник Rr	Регистр источник R1	Регистр источник R2	Не используется

ПРИЛОЖЕНИЕ 3

КОД ПРОГРАММЫ ДЛЯ NIOS II НА С

П.3.1. Листинг. Обработка таймера. Counter.c

```
#include "counter.h"

static alt_u32 timer_base;
static alt_u32 timer_interrupt;
static alt_u32 timer_irq;
// static alt_u8 leds = 0;
static int count = 0;

void start_message (void)
{
	printf("program load\n");
}

#ifdef TIMER_0_BASE
	static void handle_timer_0_interrupts(void* context)
	{
		IOWR_ALTERA_AVALON_TIMER_STATUS(timer_base, 0); //очищаем
флаг
		IOWR_ALTERA_AVALON_TIMER_CONTROL(timer_base, 0x2); //
отключаем прерывания

		count++;
		IOWR_ALTERA_AVALON_TIMER_CONTROL(timer_base, 0x07); //
запускаем таймер и активируем прерывания
	}

	void timer_init(alt_u32 _base, alt_u32 _interrupt, alt_u32 _irq)
	{
		// присвоить статическое значение переменной к локальной
переменной
		timer_base = _base;
		timer_interrupt = _interrupt;
		timer_irq = _irq;

		alt_ic_irq_disable (timer_interrupt, timer_irq);// отключить
прерывания

		alt_ic_isr_register(timer_interrupt, timer_irq,
handle_timer_0_interrupts, NULL, NULL); //ISR timer_sys

	}

	void timer_set_period(alt_u32 timer_period)
	{
		IOWR_ALTERA_AVALON_TIMER_CONTROL(timer_base, 0x08); //
остановить таймер и отключить прерывания

		IOWR_ALTERA_AVALON_TIMER_PERIODL(timer_base,
timer_period&ALTERA_AVALON_TIMER_PERIODL_MSK);//write timer period l
		IOWR_ALTERA_AVALON_TIMER_PERIODH(timer_base,
(timer_period>>16)&ALTERA_AVALON_TIMER_PERIODH_MSK);//write timer period
h

		IOWR_ALTERA_AVALON_TIMER_CONTROL(timer_base, 0x07); //
запускаем таймер и активируем прерывания
	}
```

```
	int get_count()
	{
		return count;
	}

	void null_count()
	{
		count = 0;
	}

#endif
```

П.3.2. Листинг. Заголовочный файл. Counter.h

```
#ifndef COUNTER_H_
#define COUNTER_H_

#include "alt_types.h"
#include "altera_avalon_pio_regs.h"
#include "altera_avalon_timer_regs.h"
#include "sys/alt_irq.h"
#include "system.h"
#include <stdio.h>

void start_message (void);
void timer_init(alt_u32 timer_base, alt_u32 timer_interrupt, alt_u32
timer_irq);
void timer_set_period(alt_u32 timer_period);

void null_count();
int get_count();

int max(int mas[], int len);
int min(int mas[], int len);
int sum(int mas[], int len);
int mult(int len);
int div(int len);

#endif /* COUNTER_H_ */
```

П.3.3. Листинг. Counter_main.c

```
#include "counter.h"
int main (void)
{
	int time = 0;
	int result = 1;
	timer_init(TIMER_0_BASE, TIMER_0_IRQ_INTERRUPT_CONTROLLER_ID,
TIMER_0_IRQ);
	timer_set_period(TIMER_0_LOAD_VALUE);
	int mas[] = {1,-85,78,15,6,68,2,-15,39,44,69,-95,64,56,72,64,-
6,18,72,-99};
//	int mas[] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20};
//	int mas[] = {-1,-2,-3,-4,-5,-6,-7,-8,-9,-10,-11,-12,-13,-14,-15,-
16,-17,-18,-19,-20};
	int mas_len = (int)(sizeof(mas)/sizeof(int));

	null_count();
```

```
	for(int i = 1; i < 500; i++)
		result = sum(mas, mas_len);
	time = get_count();
	printf("Сложение: время:%dms, result:%d\n", time, result);

	null_count();
	for(int i = 1; i < 500; i++)
		result = mult(mas_len);
	time = get_count();
	printf("Умножение: время:%dms, result:%d\n", time, result);

	null_count();
	for(int i = 1; i < 500; i++)
		result = div(mas_len);
	time = get_count();
	printf("Деление: время:%dms, result:%d\n", time, result);

	null_count();
	for(int i = 1; i < 500; i++)
		result = max(mas, mas_len);
	time = get_count();
	printf("Наибольшее: время:%dms, result:%d\n", time, result);

	null_count();
	for(int i = 1; i < 500; i++)
		result = min(mas, mas_len);
	time = get_count();
	printf("Наименьшее: время:%dms, result:%d\n", time, result);
	return 0;
}

int sum(int mas[], int len){
	int result = 0;
	for (int i = 0; i < len; i++ )
	{
		result = result + mas[i];
	}
	return result;
}

int mult(int len){
	int result = 1;
	for (int i = 0; i < len; i++ )
	{
		result = result * 2;
	}
	return result;
}

int div(int len){
	int result = 1000000;
	for (int i = 0; i < len; i++ )
	{
		result = result / 2;
	}
	return result;
}

int max(int mas[], int len){
	int result = mas[0];
	for(int i = 1; i < len; i++)
	{
		if(mas[i]>result){
```

```
                    result = mas[i];
                }
        }
        return result;
}

int min(int mas[], int len){
        int result = mas[0];
        for(int i = 1; i < len; i++)
        {
                if(mas[i]<result){
                        result = mas[i];
                }
        }
        return result;
}
```

ПРИЛОЖЕНИЕ 4

КОД ПРОГРАММЫ ДЛЯ NIOS II НА АССЕМБЛЕРЕ

П.4.1. Программа на ассемблере. Чтение значений с кнопок.

```
.equ ADR_LEDS, 0x00011040
.equ ADR_BUTTON, 0x00011020

.equ MAX_COUNT_VALUE, 31
.equ MIN_COUNT_VALUE, 1

# r2: для промежуточных вычислений
# r3: последнее прочитанное значение с кнопок
# r6: счетчик "нажатий"
# r8: флаги: 1 бит - 1 светодиод, 2 бит - 2 светодиод
# r9: дл¤ промежуточных вычислений

.global main
main:
movi r6,MIN_COUNT_VALUE # начальное значение счетчика
movi r8,0                    # начальное значение флагов

LOOP:
movia r2, ADR_BUTTON    # присваиваем адрес кнопок
ldwio r3,0(r2)               # читаем значение кнопок и записываем в r3
movi r2,3                    #
and r3, r3, r2               # отсеиваем все что не в 1 и 2 битах
регистра r3
movi r2,1
beq r3,r2, INCREMENT_COUNT # 1 бит единица значит переходим
SKIP_INCREMENT:
#-- проверка отпускания клавиши--
movi r2,1
and r9,r3,r2                 # отсеиваем все что не в 1 бите регистра r3
beq r3,r2, DECREMENT    # 1 бит единица значит переходим
and r9,r8,r2                 # отсеиваем все что не в 1 бите регистра r8
movi r2,0
beq r9,r2, DECREMENT    # если 0 значит переходим
and r8,r8,r2
#--конец проверки--

DECREMENT:
movi r2,2
beq r3,r2, DECREMENT_COUNT
SKIP_DECREMENT:
#--проверка отпускания клавиши--
movi r2,2
and r9,r3,r2
beq r3,r2, LEDS
and r9,r8,r2
movi r2,0
beq r9,r2, LEDS
and r8,r8,r2
#--конец проверки--

LEDS:
movia r2, ADR_LEDS           # загружаем адрес светодиодов
stwio r6,0(r2)               # загружаем значение r6 в r2(светодиоды)
br LOOP                      # безусловный переход к LOOP

#
INCREMENT_COUNT:
```

```
movi r2,1
and r9,r8,r2
beq r9,r2, SKIP_INCREMENT
or r8,r8,r2
#
movi r2,MAX_COUNT_VALUE
beq r6,r2, LOOP
addi r6,r6,1                    # инкрементируем счетчик
br LEDS

DECREMENT_COUNT:
movi r2,2
and r9,r8,r2
beq r9,r2, SKIP_DECREMENT
or r8,r8,r2

movi r2, MIN_COUNT_VALUE
beq r6,r2, LOOP
subi r6,r6,1
br LEDS
```

П.4.2. Программа на ассемблере. Запись в память программ 16 чисел.

```
.equ MAX_COUNT_VALUE, 16
.equ MIN_COUNT_VALUE, 0

# r5: конечное значение счетчика
# r6: начальное значение счетчика


.global main
main:
movi r6,MIN_COUNT_VALUE # начальное значение счетчика
movi r5,MAX_COUNT_VALUE # конечное значение счетчика

LOOP:
stw r6, 8(r6)
addi r6, 1;

bne r6,r5, LOOP
```

П.4.3. Программа на ассемблере. Чтение из памяти данных чисел, их сложение и запись результата в память данных.

```
.equ MAX_COUNT_VALUE, 8
.equ MIN_COUNT_VALUE, 1

# r3: результат
# r5: конечное значение счетчика
# r6: начальное значение счетчика

.global main
main:
movi r6, MIN_COUNT_VALUE # начальное значение счетчика
movi r5, MAX_COUNT_VALUE # конечное значение счетчика

movi r3, 0
```

```
LOOP:
ldw r6, 8(r6)
add r3, r6, r3

bne r6,r5, LOOP
stb r3, 7(0)
```

ПРИЛОЖЕНИЕ 5

КОД 32-РАРЯДНОГО ПРОТОТИПА С ОПРАЦИЯМИ УМНОЖЕНИЯ И ДЕЛЕНИЯ

П.5.1. Листинг. Ядро Soft-процессора

```
module cpu(
input clk, reset,
// для памяти команд
input [31:0] command,
output             rden,
output [7:0] address,

// для память данных
input         [31:0] read_data_dm,
output reg       wren_dm, rden_dm,
output reg [7:0] wraddress_dm, rdaddress_dm,
output reg [31:0] write_data_dm

);
// бит чтения из памяти программы
assign rden = 1'b1;
// биты для перехода
wire need_jump, beq, bneq, bge;
reg flag_jamp;

reg [7:0] ip;

// номер команды для УУ
reg [7:0] num_command_I, num_command_D, num_command_E, num_command_M,
num_command_W;

// адреса значения для регистрового файла
reg               rf_rden_1, rf_rden_2, rf_write;
reg  [3:0]  rf_write_dest, rf_read_addr_1, rf_read_addr_2;
reg  [31:0]       rf_write_data;
wire [31:0]       rf_read_data_1, rf_read_data_2;
reg  [7:0]  rf_read_addr_1_D, rf_read_addr_2_D;

// адрес перхода
reg  [7:0]  jump_addr;

// для ALU
reg [31:0] rf_read_data_1_E, rf_read_data_1_M, rf_read_data_1_W,
rf_read_data_2_E;
reg [31:0] reg_rezult_M, reg_rezult_W;
reg [31:0] alu_word, alu_result;
wire [31:0] rf_data_1, rf_data_2, retunt_data_W_1, retunt_data_W_2,
retunt_data_M_1, retunt_data_M_2, retunt_data_D_1, retunt_data_D_2;
reg [31:0] reg_write_data_dm_M, reg_write_data_dm_W;

reg   [2:0] alu_comand;
reg   [1:0] shift_code;
reg               alu_dm, alu_shift, alu_mdu, alu_active;

// регистры для конвеера
reg [31:0] r_opcode_D, r_opcode_E, r_opcode_M, r_opcode_W;
reg [7:0]  fb_addr_D;
reg [31:0]  fb_data_D;
```

```
wire [31:0] command_I;
reg               flag_stop_com;

initial begin

      fb_addr_D = 8'b0;
      fb_data_D = 32'b0;

      ip = 1'b0;

      alu_comand = 3'b0;
      shift_code = 2'b0;
      alu_dm = 1'b0;
      alu_shift = 1'b0;

      rf_rden_1 = 1'b0;
      rf_read_addr_1 = 4'b0;
      rf_rden_2 = 1'b0;
      rf_read_addr_2 = 4'b0;

      jump_addr = 8'b0;
      flag_jamp = 1'b0;

      rden_dm = 1'b0;
      rdaddress_dm = 4'b0;

      rden_dm = 1'b0;
      rdaddress_dm = 4'b0;
      alu_word = 32'b0;

      reg_write_data_dm_M = 32'b0;

      wren_dm = 1'b0;
      wraddress_dm = 1'b0;
      write_data_dm = 32'b0;
      rf_write = 1'b0;
      rf_write_dest = 4'b0;
      rf_write_data = 32'b0;

      alu_result = 32'b0;

      num_command_I = 32'h0;
      num_command_D = 32'h0;
      num_command_E = 32'h0;
      num_command_M = 32'h0;
      num_command_W = 32'h0;

      r_opcode_D = 32'h0;
      r_opcode_E = 32'h0;
      r_opcode_M = 32'h0;
      r_opcode_W = 32'h0;

      rf_read_data_1_E = 32'b0;
      rf_read_data_1_M = 32'b0;
      rf_read_data_1_W = 32'b0;
      rf_read_data_2_E = 32'b0;

      reg_rezult_M = 32'b0;
      reg_rezult_W = 32'b0;

      reg_write_data_dm_W = 32'b0;
end
```

```
assign command_I = (flag_stop_com) ? 1'b0: command;


// Управляющее устройство для конвеера
always @*
begin

// -------------I BEGINE-------------

      num_command_I = command_I[31:24];
      if(num_command_I[2] == 1'b1) begin
            rf_rden_1 = 1'b1;
            rf_read_addr_1 = command[19:16];
      end else begin
            rf_rden_1 = 1'b0;
            rf_read_addr_1 = 4'b0;
      end


      rf_rden_2 = 1'b1;
      rf_read_addr_2 = command_I[15:12];


// -------------D BEGINE-------------

      num_command_D = r_opcode_D[31:24];
      if( num_command_D[7] == 1'b1 & num_command_D[0] == 1'b1) begin
            jump_addr = r_opcode_D[7:0];
            flag_jamp = 1'b1;
      end else begin
            jump_addr = 8'b0;
            flag_jamp = 1'b0;
      end

// -------------E BEGINE-------------

      num_command_E = r_opcode_E[31:24];
      if( num_command_E[0] == 1'b1 )
            alu_word = {20'b0, r_opcode_E[11:0]};

      if( num_command_E[6] == 2'b0 ) begin // alu_comand
            if( num_command_E[5:3] > 3'b0) begin
                  alu_shift = 1'b0;
                  shift_code = 2'b0;
                  alu_dm = 1'b0;
                  alu_mdu = 1'b0;
                  alu_active = 1'b1;
                  alu_comand = num_command_E[5:3];
            end else begin
                  alu_shift = 1'b0;
                  shift_code = 2'b0;
                  alu_dm = 1'b0;
                  alu_mdu = 1'b0;
                  alu_active = 1'b0;
                  alu_comand = 3'b0;
            end
      end
      else begin
            if( num_command_E[5] == 2'b0 ) begin
                  if( num_command_E[4] == 2'b0 ) begin   //alu_dm
                        alu_shift = 1'b0;
```

```
                        shift_code = 2'b0;
                        alu_dm = 1'b1;
                        alu_mdu = 1'b0;
                        alu_active = 1'b0;
                        alu_comand = 3'b0;
                  end else begin
            //alu_mdu
                        alu_shift = 1'b0;
                        shift_code = 2'b0;
                        alu_dm = 1'b0;
                        alu_mdu = 1'b1;
                        alu_active = 1'b0;
                        alu_comand = 3'b0;
                  end
            end else begin
            //alu_shift
                  alu_shift = 1'b1;
                  shift_code = num_command_E[4:3];
                  alu_dm = 1'b0;
                  alu_mdu = 1'b0;
                  alu_active = 1'b0;
                  alu_comand = 3'b0;
            end
      end

      if( num_command_E[6:3] == 4'b1001) begin
            rden_dm = 1'b1;
            rdaddress_dm = alu_result[7:0];
      end else begin
            rden_dm = 1'b0;
            rdaddress_dm = 4'b0;
      end

// -------------M BEGINE-------------

      num_command_M = r_opcode_M[31:24];
      if( num_command_M[6] == 1'b0 ) begin
            if( num_command_M[5:3] > 3'b0)                        //alu_dm
                  reg_write_data_dm_M = reg_rezult_M;
            else
                  reg_write_data_dm_M = 32'b0;
      end
      else begin
            if( num_command_M[5] == 2'b0 ) begin
                  if( num_command_M[4] == 2'b0 ) begin   //alu_dm
                        reg_write_data_dm_M = read_data_dm;
                  end else begin
            //alu_mdu
                        reg_write_data_dm_M = reg_rezult_M;
                  end
            end else begin
            //alu_shift
                  reg_write_data_dm_M = reg_rezult_M;
            end
      end


// -------------W BEGINE-------------

      num_command_W = r_opcode_W[31:24];
      if( num_command_W[6:3] == 4'b1000) begin
            rf_write = 1'b0;
            rf_write_dest = 4'b0;
```

```
            rf_write_data = 32'b0;
            wren_dm = 1'b1;
            wraddress_dm = reg_rezult_W[8:0];
            write_data_dm = rf_read_data_1_W;
      end else begin
            if( num_command_W[5:3] > 3'b0) begin
                  rf_write = 1'b1;
                  rf_write_dest = r_opcode_W[23:20];
                  rf_write_data = reg_write_data_dm_W;
            end else begin
                  rf_write = 1'b0;
                  rf_write_dest = 4'b0;
                  rf_write_data = 32'b0;
            end
            wren_dm = 1'b0;
            wraddress_dm = 4'b0;
            write_data_dm = 32'b0;
      end


// ---------------------------------
end

// Регистровый файл на RAM
reg_file_ram rfr(
      .clk(clk),
      .aclr(reset),
      .wren(rf_write),
      .wraddress(rf_write_dest),
      .data(rf_write_data),
      .rden_1(rf_rden_1),
      .rdaddress_1(rf_read_addr_1),
      .q_1(rf_read_data_1),
      .rden_2(rf_rden_2),
      .rdaddress_2(rf_read_addr_2),
      .q_2(rf_read_data_2));


// проверка перехода
assign beq  = (rf_data_1 == rf_data_2) & (num_command_D == 8'h87);
assign bneq = (rf_data_1 != rf_data_2) & (num_command_D == 8'h8F);
assign bge  = (rf_data_1 >= rf_data_2) & (num_command_D == 8'h9F);
assign need_jump = ( flag_jamp & (beq | bneq | bge));

//Арифметико-Логическое Устройство
always @* begin
      if( alu_active )
            case( alu_comand )
                  3'b001: alu_result = rf_read_data_1_E +
rf_read_data_2_E + alu_word;
                  3'b010: alu_result = rf_read_data_1_E -
rf_read_data_2_E;
                  3'b011: alu_result = rf_read_data_2_E + 32'b1;
                  3'b100: alu_result = rf_read_data_1_E |
rf_read_data_2_E;
                  3'b101: alu_result = rf_read_data_1_E &
rf_read_data_2_E;
                  3'b110: alu_result = rf_read_data_1_E ^
rf_read_data_2_E;
                  3'b111: alu_result = ~rf_read_data_2_E;
            endcase
```

```
      if( alu_dm ) begin
            alu_result = rf_read_data_2_E + alu_word;
      end

      if( alu_shift )
            case( shift_code )
                  3'b00: alu_result = rf_read_data_1_E <<
rf_read_data_2_E;
                  3'b01: alu_result = rf_read_data_1_E >>
rf_read_data_2_E;

            endcase

      if ( alu_mdu)
            if( num_command_E[3]==1'b0)
                  alu_result = rf_read_data_1_E * rf_read_data_2_E;
            else
                  alu_result = rf_read_data_1_E / rf_read_data_2_E;
end

assign retunt_data_D_1 = ( rf_read_addr_1_D == fb_addr_D) ? fb_data_D :
rf_read_data_1;
assign retunt_data_D_2 = ( rf_read_addr_2_D == fb_addr_D) ? fb_data_D :
rf_read_data_2;

assign retunt_data_W_1 = ( rf_read_addr_1_D == r_opcode_W[23:20] ) ?
reg_write_data_dm_W : retunt_data_D_1;
assign retunt_data_W_2 = ( rf_read_addr_2_D == r_opcode_W[23:20] ) ?
reg_write_data_dm_W : retunt_data_D_2;

assign retunt_data_M_1 = ( rf_read_addr_1_D == r_opcode_M[23:20] ) ?
reg_write_data_dm_M : retunt_data_W_1;
assign retunt_data_M_2 = ( rf_read_addr_2_D == r_opcode_M[23:20] ) ?
reg_write_data_dm_M : retunt_data_W_2;

assign rf_data_1 = ( rf_read_addr_1_D == r_opcode_E[23:20] &
num_command_E != 8'h47 & num_command_E != 8'h4B) ? alu_result :
retunt_data_M_1;
assign rf_data_2 = ( rf_read_addr_2_D == r_opcode_E[23:20] &
num_command_E != 8'h47 & num_command_E != 8'h4B) ? alu_result :
retunt_data_M_2;
// Передача регистров с результатом по конвееру
always @(posedge clk or posedge reset)
begin

      if(reset)
   begin
            r_opcode_D <= 32'b0;
            r_opcode_E <= 32'b0;
            r_opcode_M <= 32'b0;
            r_opcode_W <= 32'b0;

            rf_read_addr_1_D <= 4'b0;
            rf_read_addr_2_D <= 4'b0;

            rf_read_data_1_E <= 32'b0;
            rf_read_data_1_M <= 32'b0;
            rf_read_data_1_W <= 32'b0;
            rf_read_data_2_E <= 32'b0;

            reg_rezult_M <= 32'b0;
            reg_rezult_W <= 32'b0;
```

```
            reg_write_data_dm_W <= 32'b0;

            fb_addr_D = 8'b0;
            fb_data_D = 32'b0;
   end else begin
            if( need_jump ) begin // очищаем конвеер при перходе
                  r_opcode_D <= 32'b0;
                  r_opcode_E <= 32'b0;
                  flag_stop_com <= 1'b1;
            end else begin
                  r_opcode_D <= command_I;
                  r_opcode_E <= r_opcode_D;

                  flag_stop_com <= 1'b0;
            end
            r_opcode_M <= r_opcode_E;
            r_opcode_W <= r_opcode_M;

            rf_read_addr_1_D <= rf_read_addr_1;
            rf_read_addr_2_D <= rf_read_addr_2;

            rf_read_data_1_E <= rf_data_1;
            rf_read_data_1_M <= rf_read_data_1_E;
            rf_read_data_1_W <= rf_read_data_1_M;
            rf_read_data_2_E <= rf_data_2;

            reg_rezult_M <= alu_result;
            reg_rezult_W <= reg_rezult_M;

            reg_write_data_dm_W <= reg_write_data_dm_M;

            fb_addr_D <= rf_write_dest;
            fb_data_D <= rf_write_data;

      end

end

assign address = ip;
// Определение следующего адреса
always @(posedge clk or posedge reset)
begin
    if(reset)
    begin
        ip <= 0;
    end
    else
    begin
            if( need_jump )
            begin
                  ip <= jump_addr;
            end else begin
                  ip <= ip + 1'b1;
            end
    end
end

endmodule
```

П.5.2. Листинг. Конфигурация Soft-процессора

```
module mips_32(
      input clk,
      output [31:0] command
);
wire reset = 1'b0;

wire [31:0] data_instr;
wire [7:0]  address_instr;
reg  [7:0]  wraddress_instr;
wire              rden_instr;
reg               wren_instr;
wire        wren_dm, rden_dm;
wire [7:0]  wraddress_dm, rdaddress_dm;
wire [31:0]  write_data_dm, read_data_dm;
initial begin
wraddress_instr = 8'b0;
wren_instr = 1'b0;
end

inctructions im (
            .clock(clk),

            .wren(wren_instr),
            .wraddress(wraddress_instr),
            .data(data_instr),

            .rden(rden_instr),
            .rdaddress(address_instr),
            .q(command)
);

cpu cpu(
            .clk(clk),
            .reset(reset),
            //для памяти програм
            .rden(rden_instr),
            .address(address_instr),
            .command(command),
            // чтение из памяти данных
            .rden_dm(rden_dm),
            .rdaddress_dm(rdaddress_dm),
            .read_data_dm(read_data_dm),
            // сапись в память данных
            .wren_dm(wren_dm),
            .wraddress_dm(wraddress_dm),
            .write_data_dm(write_data_dm)
);

data dm(
            .clock(clk),
            .aclr(reset),

            .wren(wren_dm),
            .wraddress(wraddress_dm),
            .data(write_data_dm),

            .rden(rden_dm),
            .rdaddress(rdaddress_dm),
            .q(read_data_dm));
endmodule
```