

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ
ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение
высшего профессионального образования

Национальный исследовательский ядерный университет «МИФИ»

Институт Интеллектуальных Кибернетических Систем

Отчет по курсовой работе

Учебная дисциплина : «Схемотехника»

Тема: Разработка модели процессорного ядра RISC-V (блок записи
результата, общая сборка проекта и его тестирование)

Выполнили:

студенты группы С22-501

Павлюк Глеб

Плотников Артем

Смирнов Илья

Москва 2024

Оглавление

Постановка задачи.....	3
Глава 1. Блок записи результата	5
1.1. Спецификация	6
Подключение к соседним модулям.....	7
1.2. Тестирование	8
1.3. Результаты синтеза	12
Глава 2. Общая сборка.....	12
2.1. Спецификация	13
2.1.1. Decoder.....	13
2.1.2. LSU (Load/Store Unit).....	14
2.1.3. ALU (Arithmetic Logic Unit).....	15
2.1.4. CSR (Control and Status Registers).....	16
2.1.5. Writeback.....	18
2.1.6. Формирование топового уровня.....	18
2.2. Тестирование	18
2.2.1. Подготовка программы к запуску	18
2.2.2. Тестирование всего ядра.....	23
2.3. Результаты синтеза	23

Постановка задачи

В рамках курсовой работы требовалось выполнить групповой проект по разработке процессорного ядра с сокращенной системой команд (RISC-V), построенного на базе классической гарвардской архитектуры (раздельная память инструкций и данных). На рисунке 1 представлена упрощенная структура схемы модели IP-ядра RISC-V.

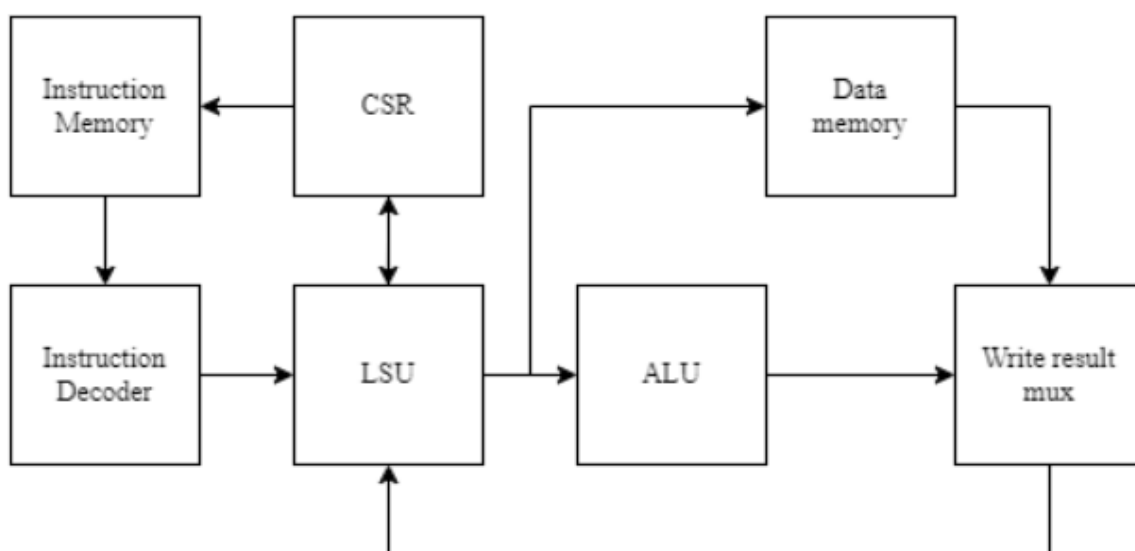


Рисунок 1. Структурная схема модели IP-ядра RISC-V.

В рамках данной курсовой работы были поставлены следующие общие требования:

1. Четырёхступенчатый конвейер со стадиями: декодирования команды, подготовки операндов, выполнения операции, результата выполнения операции.
2. Реализация инструкций RV32I (только load, store, shift, arithmetic, logical, compare) и инструкций умножения из стандартного расширения RV32M.

В рамках данной курсовой работы были определены следующие общие ограничения:

1. Факультативное присутствие (необязательность реализации) команды условного перехода и, как следствие, предсказателя ветвлений (branch predictor);
2. Работа только с целочисленными значениями;
3. Отсутствие аппаратного деления;
4. Отсутствие режима отладки.

В курсовой работе было необходимо разработать блок записи результата и выполнить общую сборку проекта — сформировать топовый уровень процессора, а также подготовить тестовые процедуры для целостного проекта.

Глава 1. Блок записи результата

Постановка задачи

Разработать блок записи результата (*Writeback*), удовлетворяющий функционалу— изменение источника результата для дальнейшей записи в регистровый файл: *АЛУ (ALU)* или *блок памяти данных (Data memory)*, и его передача в *блок загрузки-хранения (LSU)*.

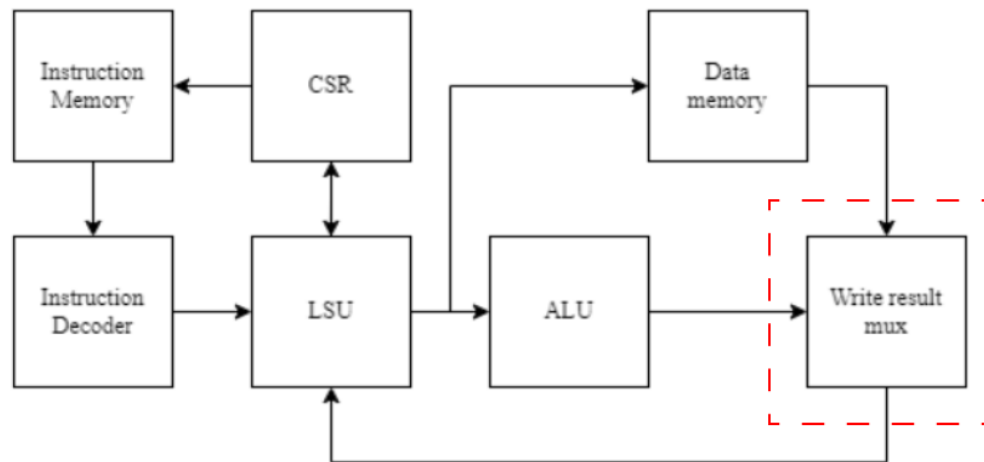


Рисунок 2. Блок записи результата на общей схеме модели процессора
(обведен)

1.1. Спецификация

Условное графическое обозначение блока Writeback представлено на рисунке 3.

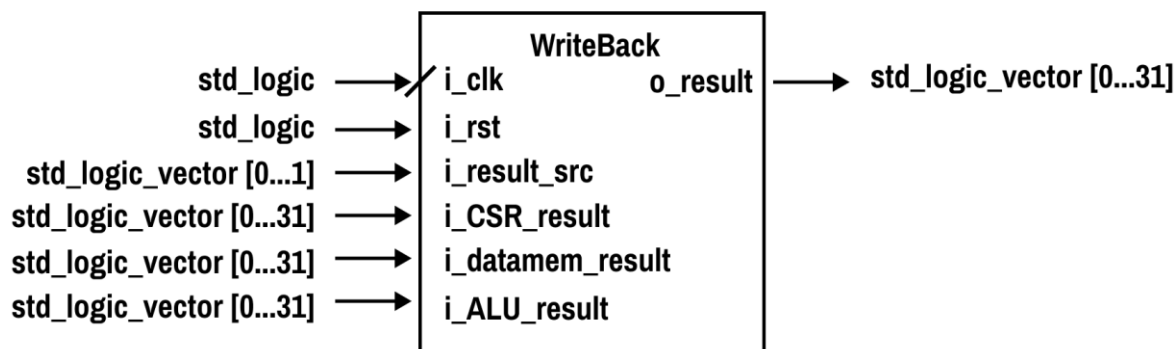


Рисунок 3. УГО для блока записи результата

В таблице 1 представлен список портов ввода/вывода для Writeback модуля.

Таблица 1. Список портов ввода/вывода для блока записи результата

№	Наименование	Тип	Активный уровень	Назначение порта
1.	i_clk	STD_LOGIC	↑	Получение тактового сигнала (от топового уровня)
2.	i_rst	STD_LOGIC	'1'	Получение сигнала сброса (от топового уровня)
3.	i_result_src	STD_LOGIC_VECTOR (1 downto 0);	wb_result_src	Получение сигнала-управляющего слова для выбора источника результата (от декодера)
4.	i_ALU_result	STD_LOGIC_VECTOR (31 downto 0);	alu_result	Получение сигнала-результата при управляющем слове "00" (от АЛУ)

№	Наименование	Тип	Активный уровень	Назначение порта
1.	i_clk	STD_LOGIC	↑	Получение тактового сигнала (от топового уровня)
2.	i_rst	STD_LOGIC	‘1’	Получение сигнала сброса (от топового уровня)
5.	i_datamem_result	STD_LOGIC_VECTOR (31 downto 0);	datamem_result	Получение сигнала-результата при управляющем слове “01” (от Памяти данных)
6.	i_CSR_result	STD_LOGIC_VECTOR (31 downto 0);	CSR_result	Получение сигнала-результата при управляющем слове “10” (от CSR)
7.	o_result	STD_LOGIC_VECTOR (31 downto 0);	wb_result	Выдача сигнала-результата (в LSU)

Подключение к соседним модулям

На рисунке 4 представлена схема подключения между блоком записи результата, АЛУ, блоком памяти данных и декодером.

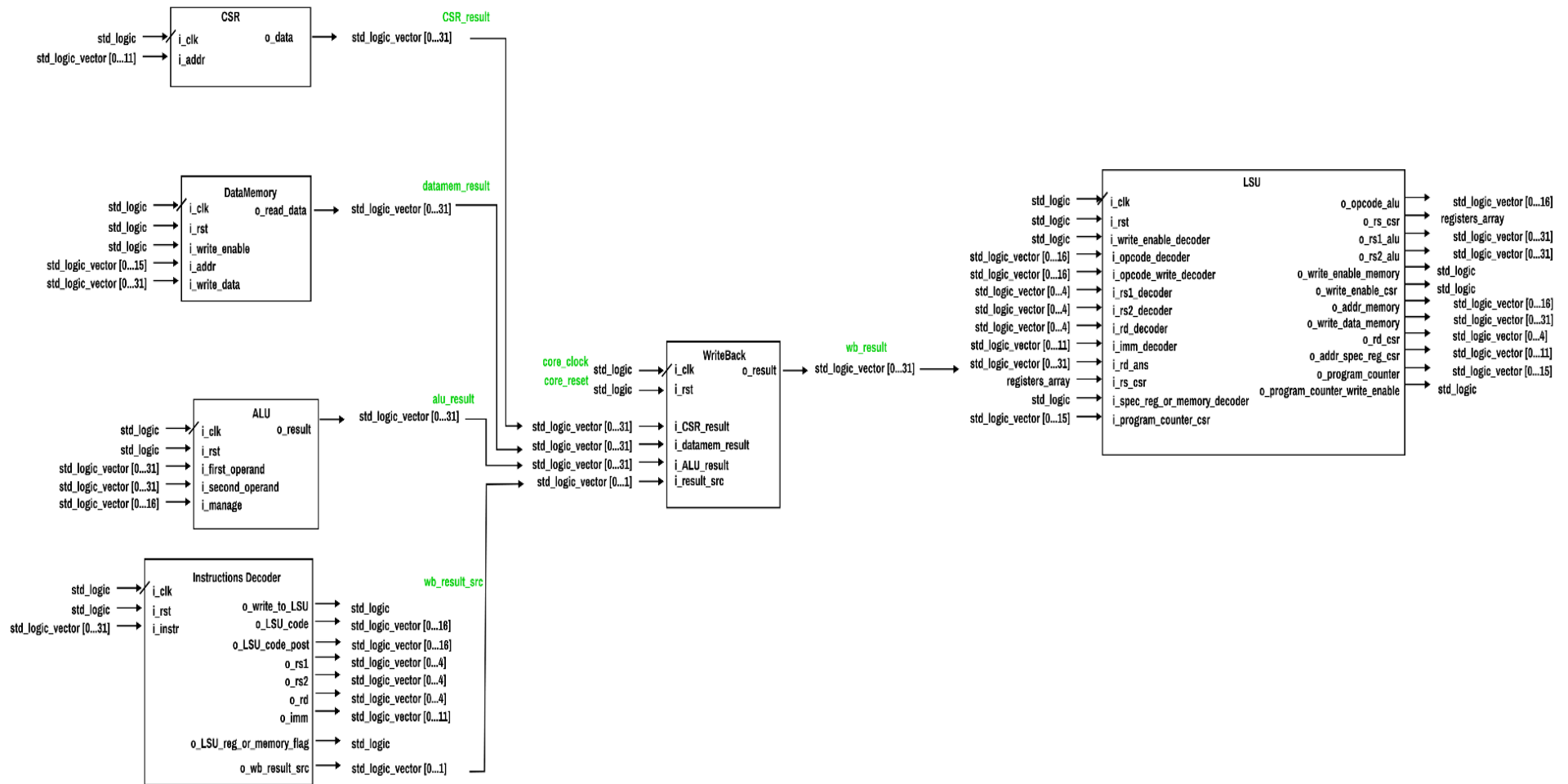
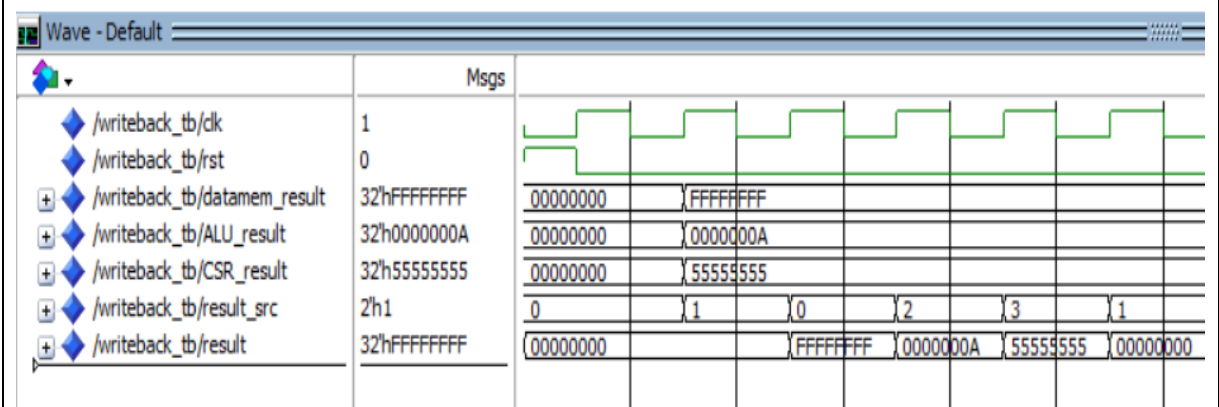


Рисунок 4. Схема подключения блока записи результата

1.2. Тестирование

Было проведено тестирование разработанного блока записи результата. Сценарии тестирования представлены в таблице 2.

Таблица 2. Сценарии проведенных тестов

Сценарий тестирования	Описание
<p>1. Изменение управляющего сигнала</p> <ul style="list-style-type: none">Цель: проверка работы модуля при неизменных значениях на информационных входах и изменяющемся управляющем слове (см. рис. 5)	<p>1. Установка значений на информационных входах</p> <pre>i_ALU_result <= x"0000000A"; i_datamem_result <= x"FFFFFFFF"; i_CSR_result <= x"55555555";</pre> <p>2. Переключение источника информации и ожидание 1 такта</p> <pre>i_result_src <= "01"; wait_clk(1); i_result_src <= "00"; wait_clk(1); i_result_src <= "10"; wait_clk(1); i_result_src <= "11"; wait_clk(1);</pre> <p>Ожидаемый результат: изменение выходного сигнала через такт после поступления соответствующего result_src на "0000000A", "FFFFFFFF", "55555555", "00000000".</p>
 <p>The timing diagram shows a clock signal (clk) and a reset signal (rst) at the top. Below them are several data signals: datamem_result (initially 32'hFFFFFF, then 00000000), ALU_result (initially 32'h0000000A, then 00000000), CSR_result (initially 32'h55555555, then 00000000), result_src (initially 2'h1, then 0, 1, 0, 2, 3, 1), and result (initially 32'hFFFFFF, then 00000000, FFFFFFFF, 0000000A, 55555555, 00000000). The diagram illustrates the sequence of operations described in the test scenario.</p>	
<p>2. Изменение информации на входе от памяти данных при неизменном управляющем слове</p>	<p>1. Установка значения на информационном входе i_datamem_result:</p> <pre>i_datamem_result <= x"FFFFFFFF";</pre>

- Цель: проверка работы модуля при неизменном значении управляющего слова и изменении информации на информационном входе от памяти данных.

(см. рис. 6)

2. Установка управляющего слова на источник - память данных.

```
i_result_src <= "01";
wait_clk(1);
```

3. Установка другого значения на информационном входе

```
i_datamem_result <= x"87654321";
wait_clk(1);
```

Ожидаемый результат: изменение выходного сигнала через такт с "FFFFFFFF" на "87654321".

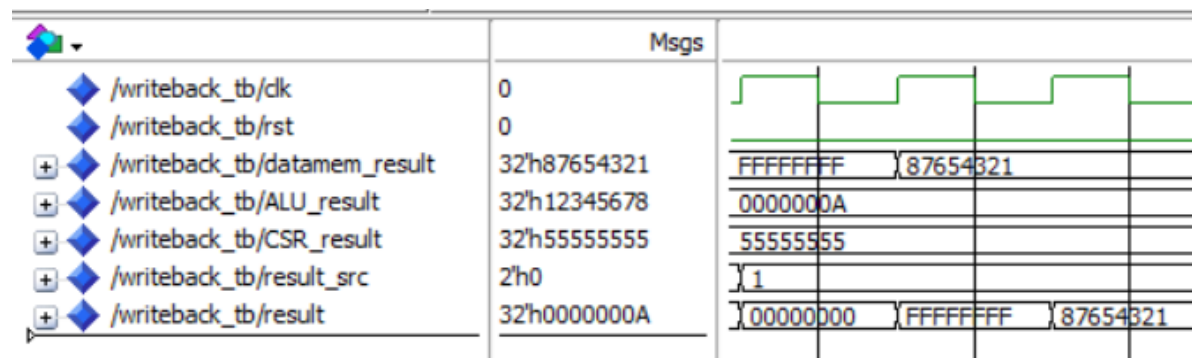


Рисунок 6. Временная диаграмма для сценария №2 “Изменение информации на входе от памяти данных при неизменном управляющем слове”

3. Изменение информации на входе от АЛУ при неизменном управляющем слове

- Цель: проверка работы модуля при неизменном значении управляющего слова и изменении информации на информационном входе от АЛУ.

(см. рис. 7)

1. Установка значения на информационном входе i_ALU_result:

```
i_ALU_result <= x"0000000A";
```

2. Установка управляющего слова на источник - АЛУ.

```
i_result_src <= "00";
wait_clk(1);
```

3. Установка другого значения на информационном входе

```
i_ALU_result <= x"12345678";
wait_clk(1);
```

Ожидаемый результат: изменение выходного сигнала через такт с "0000000A" на "12345678".

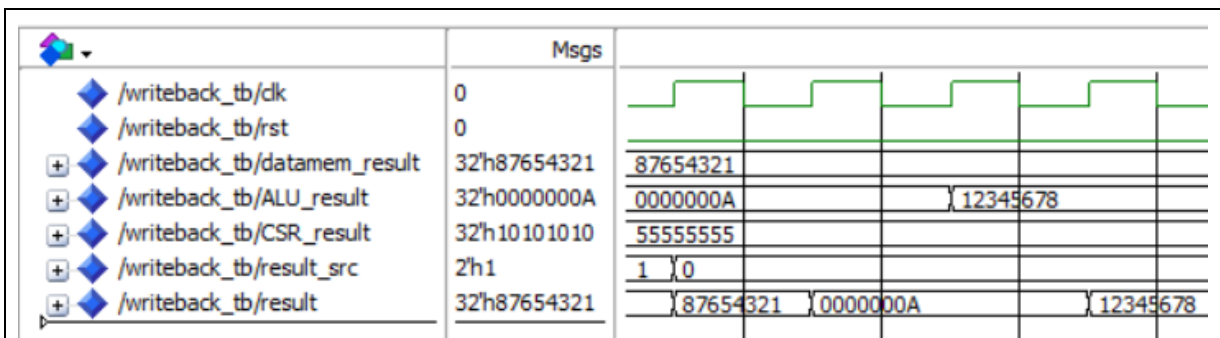


Рисунок 7. Временная диаграмма для сценария №3 “Изменение информации на входе от АЛУ при неизменном управляющем слове”

4. Изменение информации на входе от CSR при неизменном управляющем слове

- Цель: проверка работы модуля при неизменном значении управляющего слова и изменении информации на информационном входе от CSR.

(см. рис. 8)

- Установка значения на информационном входе i_ALU_result:
i_CSR_result <= x"55555555";
- Установка управляющего слова на источник - АЛУ.
i_result_src <= "10";
wait_clk(1);
- Установка другого значения на информационном входе
i_CSR_result <= x"10101010";
wait_clk(1);

Ожидаемый результат: изменение выходного сигнала через такт с "55555555" на "10101010".

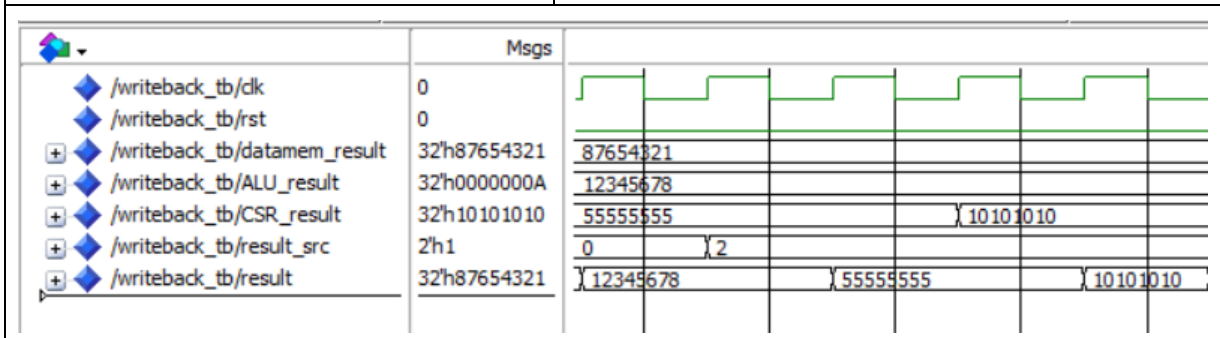


Рисунок 8. Временная диаграмма для сценария №4 “Изменение информации на входе от CSR при неизменном управляющем слове”

5. Одновременное изменение и сигнала управляющего слова, и информации на входах.

(см. рис. 9)

Цель: проверка работы модуля в условиях, приближенных к условиям в работе всего процессора: одновременное изменение двух и более сигналов.

1. Установка исходных значений на входах:
`i_datamem_result <= x"87654321";`
`i_ALU_result <= x"12345678";`
`i_CSR_result <= x"10101010";`
`i_result_src <= "10";`

2. Первая серия одновременных изменений:

`i_result_src <= "00";`
`i_ALU_result <= x"88888888";`
`wait_clk(1);`

Ожидаемый результат: выбранное в тот же такт значение информационного входа (вход от АЛУ) должно через такт стать значением информационного выхода.

3. Вторая серия одновременных изменений:
`i_CSR_result <= x"10101010";`
`i_datamem_result <= x"87654321";`
`i_ALU_result <= x"0000000A";`
`i_result_src <= "01";`
`wait_clk(1);`

Ожидаемый результат: выбранное в тот же такт значение информационного входа (вход от памяти данных) должно через такт стать значением информационного выхода.

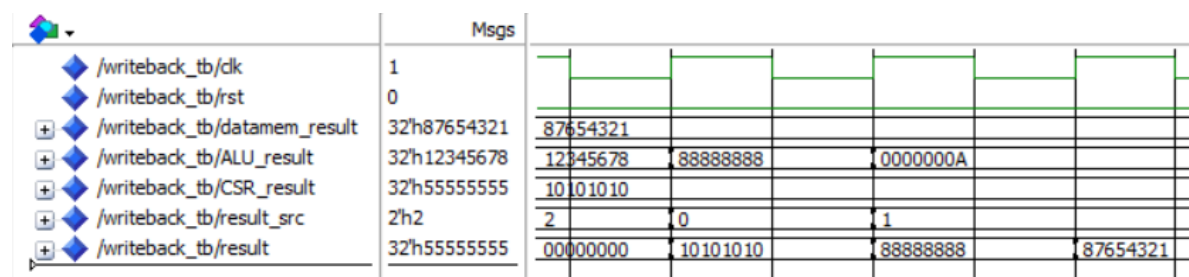


Рисунок 9. Временная диаграмма для сценария №5 “Серия одновременных изменений входных сигналов”

6. Сброс модуля - подача сигнала reset.

(см. рис. 10)

Цель: проверка функционала сброса модуля.

1. Установка исходных значений на входах:
`i_datamem_result <= x"87654321";`
`i_ALU_result <= x"12345678";`
`i_CSR_result <= x"10101010";`
`i_result_src <= "10";`

2. Подача сигнала сброса на вход i_rst
`i_rst <= '1';`
`wait_clk(1);`
`i_rst <= '0';`

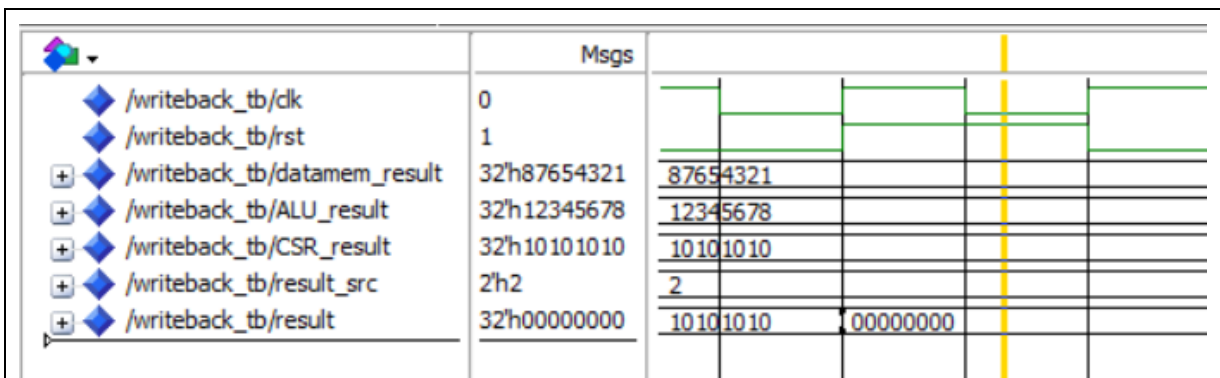


Рисунок 10. Временная диаграмма для сценария №6 “Серия одновременных изменений входных сигналов”

1.3. Результаты синтеза

Результаты синтеза разработанного блока записи результата представлены на рисунках 11, 12.

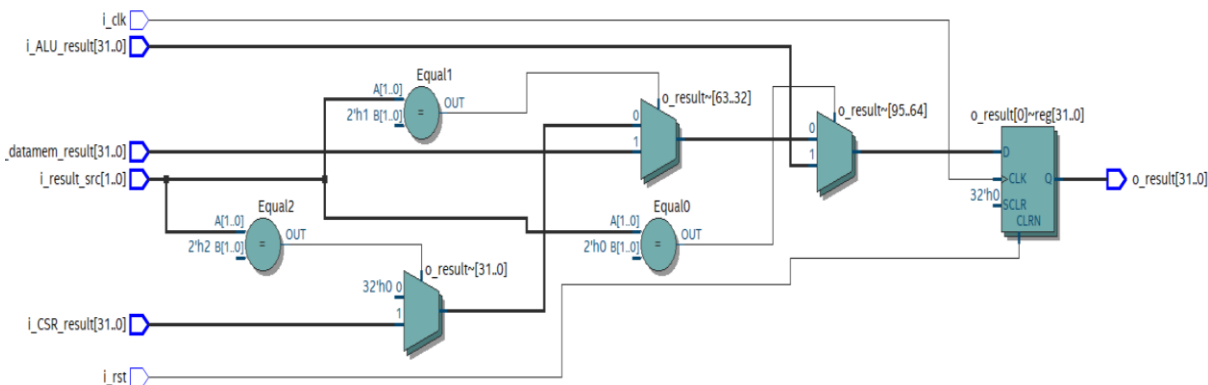


Рисунок 11. Синтез WriteBack

Flow Status	Successful - Fri Dec 27 00:04:42 2024
Quartus Prime Version	23.1std.1 Build 993 05/14/2024 SC Lite Edition
Revision Name	WriteBack
Top-level Entity Name	WriteBack
Family	Cyclone 10 LP
Total logic elements	33
Total registers	33
Total pins	101
Total virtual pins	0
Total memory bits	0
Embedded Multiplier 9-bit elements	0
Total PLLs	0

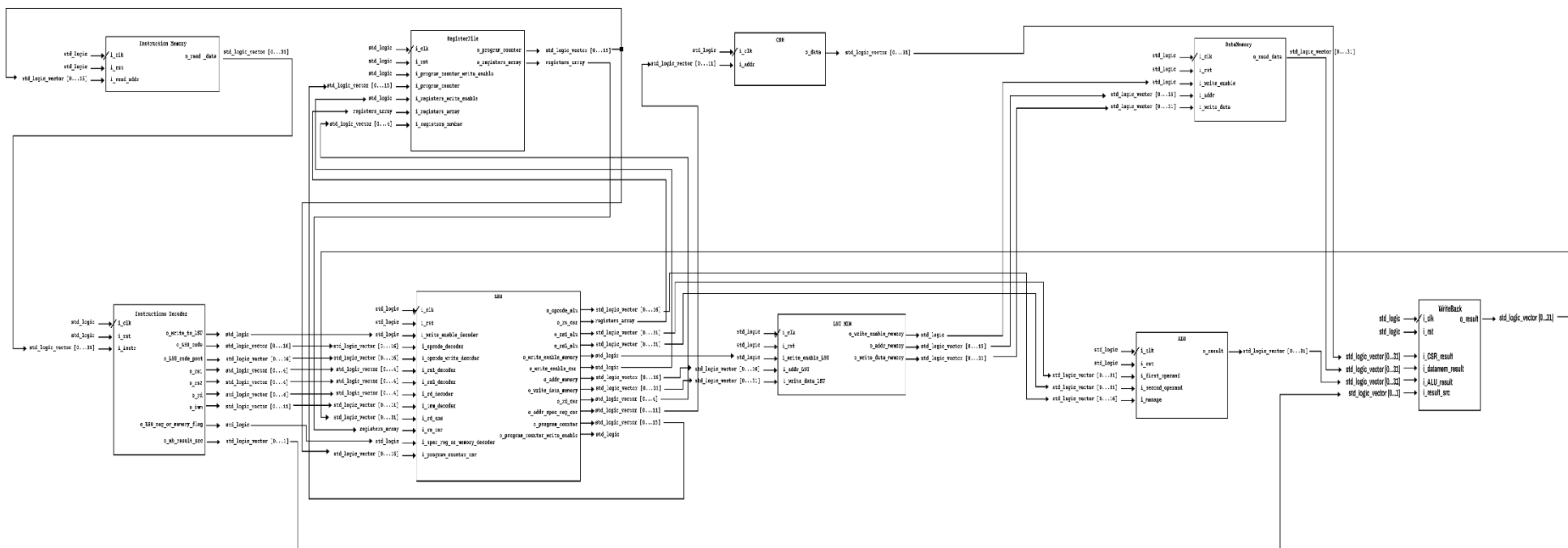
Рисунок 12. Список затраченных ресурсов для WriteBack

Глава 2. Общая сборка

В данной главе представлено описание архитектуры проектируемого ядра RISC-V. Основное внимание уделяется ключевым функциональным блокам, их спецификации, взаимодействию и объединению на уровне топ-модуля. Рассматриваемые модули реализуют основные этапы обработки инструкций: декодирование, выполнение арифметико-логических операций, работа с памятью, обработка регистров состояния и запись результата.

2.1. Спецификация

Функциональные блоки ядра RISC-V обеспечивают выполнение каждой стадии обработки инструкций согласно конвейерной архитектуре. Схема подключения модулей представлена на рисунке ниже.



Ниже представлены описания каждого блока.

2.1.1. Decoder

Блок Decode (декодирование) выполняет разбор машинной инструкции, поступающей из памяти инструкций, и определение ее типа, операндов и операции.

Основные задачи блока:

- Определение формата инструкции (R, I, S, B, U, J).
- Извлечение операндов из полей инструкции.
- Генерация управляющих сигналов для других блоков.

Коды операций представлены в таблице 3.

Таблица 3. Коды операций, с которыми работает декодер

Категория	Инструкция	Код	Funct7 Funct3 opcode	Funct 3
Load	LB	imm[11:0] rs1 000 rd 0000011	00000000000000011	000
	LH	imm[11:0] rs1 001 rd 0000011	00000000010000011	001
	LW	imm[11:0] rs1 010 rd 0000011	00000000100000011	010
	LBU	imm[11:0] rs1 100 rd 0000011	00000001000000011	011
	LHU	imm[11:0] rs1 101 rd 0000011	00000001010000011	100
Store	BS	imm[11:5] rs2 rs1 000 imm[4:0] 0100011	00000000000100011	0
	SH	S	00000000010100011	0
	SW	imm[11:5] rs2 rs1 010 imm[4:0] 0100011	00000000100100011	0
Shift	SLLI	0000000 shamt rs1 001 rd 0010011	00000000010010011	
	SRLI	0000000 shamt rs1 101 rd 0010011	00000001010010011	
	SRAI	0100000 shamt rs1 101 rd 0010011	01000001010010011	

	SLL	0000000 rs2 rs1 001 rd 0110011	00000000010110011	
	SRL	0000000 rs2 rs1 101 rd 0110011	00000001010110011	
	SRA	0100000 rs2 rs1 101 rd 0110011	01000001010110011	
Arithmetic	ADDI	imm[11:0] rs1 000 rd 0010011	00000000000010011	
	ADD	0000000 rs2 rs1 000 rd 0110011	00000000000110011	
	SUB	0100000 rs2 rs1 000 rd 0110011	01000000000110011	
Logical	XORI	imm[11:0] rs1 100 rd 0010011	00000001000010011	
	ORI	imm[11:0] rs1 110 rd 0010011	00000001100010011	
	ANDI	imm[11:0] rs1 111 rd 0010011	00000001110010011	
	XOR	0000000 rs2 rs1 100 rd 0110011	00000001000110011	
	OR	0000000 rs2 rs1 110 rd 0110011	00000001100110011	
	AND	0000000 rs2 rs1 111 rd 0110011	00000001110110011	
Compare	SLTI	imm[11:0] rs1 010 rd 0010011	00000000100010011	
	SLTIU	imm[11:0] rs1 011 rd 0010011	00000000110010011	
	SLT	0000000 rs2 rs1 010 rd 0110011	00000000100110011	
	SLTU	0000000 rs2 rs1 011 rd 0110011	00000000110110011	

Условно-графическое обозначение декодера:

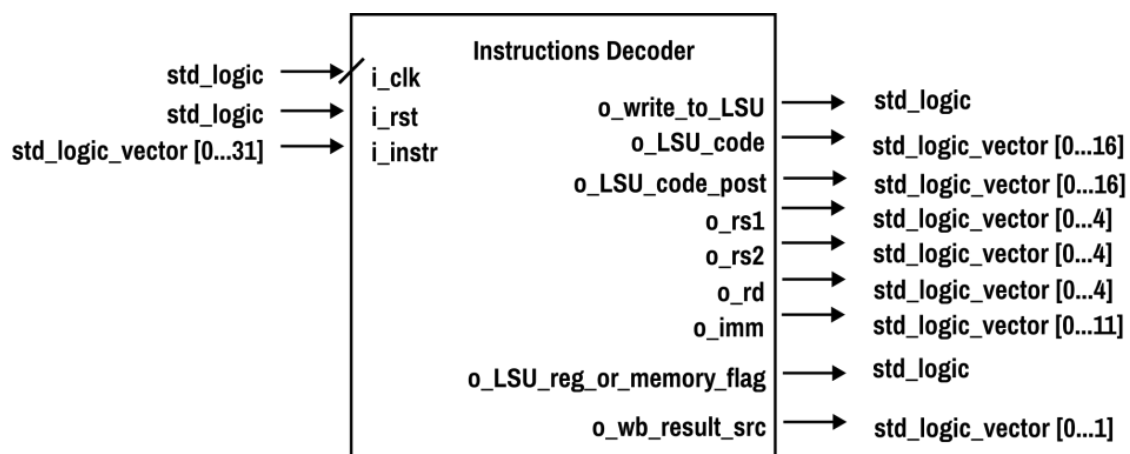


Рисунок 12. УГО для декодера

2.1.2. LSU (Load/Store Unit)

Блок LSU обрабатывает инструкции загрузки и сохранения данных. Он отвечает за взаимодействие с памятью, реализуя следующие функции:

- Вычисление адреса памяти на основе операндов инструкции.
- Чтение данных из памяти при выполнении инструкций загрузки (LOAD).
- Запись данных в память при выполнении инструкций сохранения (STORE).
- Генерация сигналов для корректной работы контроллера памяти.

Блок поддерживает операции с различными размерами данных (байты, полуслова, слова) и корректно работает с учетом порядка байтов (Little Endian).

Условно графическое обозначение:

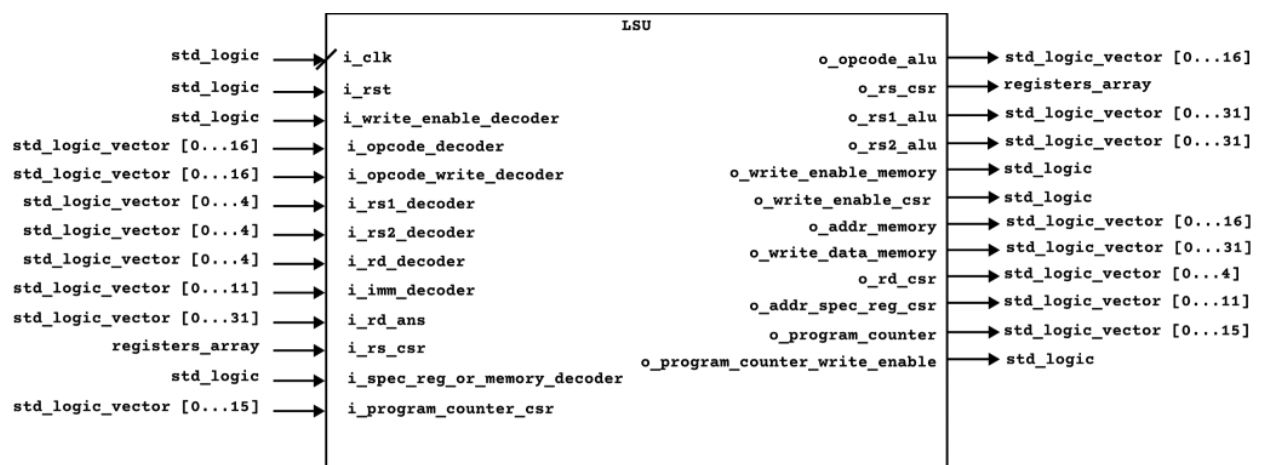


Рисунок 13. УГО для LSU

Список портов ввода/вывода:

2.1.3. ALU (Arithmetic Logic Unit)

Блок ALU выполняет арифметические и логические операции над операндами, указанными в инструкциях. Среди поддерживаемых операций:

- Арифметические: сложение, вычитание, умножение, деление.
- Логические: побитовые операции (AND, OR, XOR).
- Сдвиги: логический и арифметический сдвиг влево/вправо.
- Сравнения: равенство, больше/меньше, знаковые и беззнаковые сравнения.

ALU принимает операнды и управляющие сигналы от блока Decode, а результат отправляет в блок Writeback для записи.

Условно графическое обозначение:

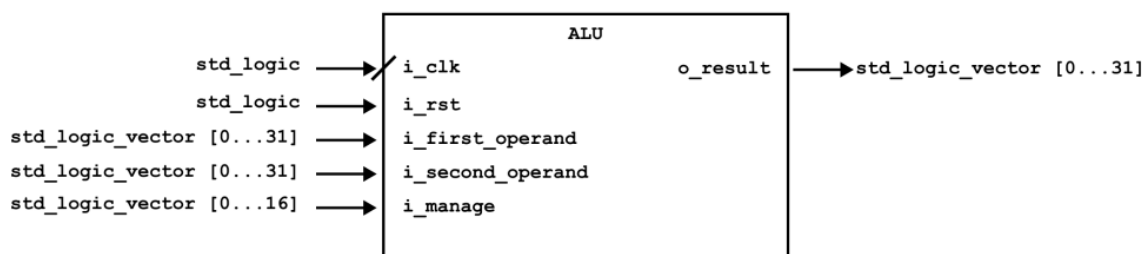


Рисунок 14. УГО для ALU

Список портов ввода/вывода:

№	Наименование	Активный уровень	Описание
1	i_clk	↑	Тактовый сигнал
2	i_rst	‘1’	Сигнал сброса регистра результата

3	i_first_operand [31..0]	data	Первый операнд
4	i_second_operand [31..0]	data	Второй операнд
5	i_manage [16..0]	data	Код операции
6	o_result [31..0]	data	Результат выполнения операции

2.1.4. CSR (Control and Status Registers)

Блок CSR управляет специальными регистрами состояния и контроля, которые используются для управления процессором и выполнения привилегированных операций.

Условно

графическое

обозначение:

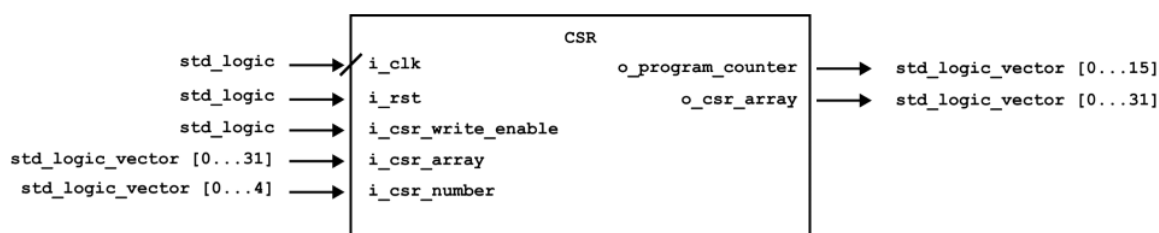


Рисунок 15. УГО для CSR

Список портов ввода/вывода:

№	Наименование	Активный уровень	Описание
1	i_clk	↑	Тактовый сигнал.
2	i_rst	‘1’	Сигнал сброса всех регистров CSR

3	i_csr_write_enable	'1'	Сигнал разрешения на запись в регистр CSR
4	i_csr_array[31..0][31..0]	data	Одновременный вход во все регистры CSR. Представляется композитным типом, который состоит из 32 регистров по 32 бита каждый (см csr_mem_pkg.vhd)
5	i_csr_number[4..0]	addr	Адрес регистра, который необходимо изменить
6	o_program_counter[15..0]	data	Текущее значение счетчика команд
7	o_csr_array[31..0][31..0]	data	Текущие значения регистров CSR. Представляется композитным типом, который состоит из 32 регистров по 32 бита каждый (см csr_mem_pkg.vhd)

2.1.5. Writeback

Блок Writeback (запись результата) обеспечивает передачу результата выполнения из соответствующего места:

- АЛУ, если имело место арифметическая, логическая, сдвиговая или сравнительная команда.
- Память данных;
- CSR.

Условно графическое обозначение и список протов ввода/вывода представлены в пункте 1.1

2.1.6. Формирование топового уровня

На уровне топового модуля все блоки объединяются для формирования общего конвейера процессора. Основные функции топового уровня:

- Организация связи между блоками (передача управляющих сигналов, данных, адресов).
- Обеспечение работы конвейера инструкций.
- Инициализация регистров и памяти.

2.2. Тестирование

2.2.1. Подготовка программы к запуску

Для проверки функциональности разработанного RISC-V ядра необходимо обеспечить запуск тестовых программ, которые имитируют работу инструкций процессора. В разделе описывается процесс подготовки программы к загрузке, начиная с написания исходного текста на языке ассемблера и заканчивая созданием машинного кода в формате, пригодном для загрузки в память. Описаны этапы преобразования программы, особенности форматов данных и настройки для автоматизации процесса.

Общая схема подготовки программы

Для запуска программы на разработанном процессоре применяется следующая последовательность действий:

1. Создание программы на языке ассемблера (файл .s).
2. Компиляция исходного кода в объектный файл (.o).
3. Линковка объектного файла для формирования исполняемого файла ELF-формата (.elf).
4. Преобразование ELF-файла в бинарный файл (.bin), содержащий только машинный код.

5. Преобразование бинарного файла в шестнадцатеричный формат (.hex) для загрузки в память устройства в симуляции.

На каждом этапе выполняются операции, обеспечивающие корректность и согласованность форматов данных.

Этапы подготовки программы

1. Ассемблирование:

На этапе ассемблирования исходный код в формате .s преобразуется в объектный файл .o, содержащий машинный код и метаданные. Для выполнения этой операции используется утилита riscv64-unknown-elf-as.

Пример команды:

```
riscv64-unknown-elf-as program.s -o program.o
```

Листинг 1 — Ассемблирование программы на Bash

2. Линковка:

Линкер разрешает ссылки и формирует ELF-файл, пригодный для анализа и использования на эмуляторах. Этот файл включает заголовки и таблицы секций, описывающие расположение инструкций.

Пример команды:

```
riscv64-unknown-elf-ld program.o -o program.elf
```

Листинг 2 — Линковка программы на Bash

3. Преобразование ELF в бинарный формат:

ELF-файл содержит избыточную информацию, которая не требуется для работы на уровне аппаратного обеспечения. С помощью утилиты `riscv64-unknown-elf-objcopy` производится преобразование в бинарный формат.

Пример команды:

```
riscv64-unknown-elf-objcopy -O binary program.elf program.bin
```

Листинг 3 — Преобразование ELF-файла в бинарный на Bash

4. Преобразование бинарного файла в HEX:

Для загрузки данных в память в симуляции на языке VHDL бинарный файл преобразуется в текстовый шестнадцатеричный формат. В зависимости от порядка байтов используются два режима: Big Endian и Little Endian.

Эндиканс данных

Эндиканс определяет порядок записи байтов в памяти:

- **Big Endian:** старший байт находится в младшем адресе.
- **Little Endian:** младший байт находится в младшем адресе.

Для тестирования разработанного ядра было принято использовать **Little Endian**, так как этот порядок байтов соответствует стандарту архитектуры RISC-V, однако разрабатываемый скрипт разработан с возможностью настраивать эндиканс полученного файла для большей гибкости.

Пример:

Машинная инструкция NOP с кодом 0x13000000:

- **Little Endian:** 00 00 00 13.

- **Big Endian:** 13 00 00 00.

Средства автоматизации подготовки

Для автоматизации подготовки программы был разработан скрипт `build_riscv.sh`. Скрипт объединяет все этапы: ассемблирование, линковку, преобразование в бинарный формат и генерацию HEX-файла. Он поддерживает параметры для настройки формата эндиканс и управления сохранением промежуточных файлов.

Параметры скрипта:

1. `--endian <big|little>` — определяет порядок байтов в HEX-файле.
2. `--keep` — сохраняет промежуточные файлы `.o`, `.elf`, `.bin`.
3. `<filename>` — имя исходного файла без расширения.

Пример работы скрипта:

Запуск:

```
./build_riscv.sh program --endian little --keep
```

Листинг 4 — Пример вызова скрипта на Bash

Результат:

Сгенерированные файлы:

- `program.o` — объектный файл.
- `program.elf` — ELF-файл.
- `program.bin` — бинарный файл.
- `program.hex` — HEX-файл в формате Little Endian.

Пример выполнения

В качестве примера работы программы приведен полный цикл подготовки:

```
.section .text
```

```
.globl _start
_start:
    nop
    nop
    nop
```

Листинг 5 — Тестовая программа program.s на Assembler

```
Assembling program.s...
Linking program.o...
Converting program.elf to binary...
Converting program.bin to hex (Little Endian)...
Build complete! Generated files:
- program.hex (Hex file in little Endian)
- program.o (Object file)
- program.elf (ELF file)
- program.bin (Binary file)
```

Листинг 6 — Результат выполнения скрипта

```
13000000
13000000
13000000
```

Листинг 7 — Содержимое файла program.hex:

В результате выполнения скрипта программа, написанная на языке ассемблера, преобразуется в машинный код, пригодный для загрузки в память ядра. Разработанный процесс подготовки учитывает требования архитектуры RISC-V, минимизирует ручные действия и позволяет гибко управлять параметрами формата данных. Это упрощает тестирование и верификацию функциональности разработанного ядра.

2.2.2. Тестирование всего ядра

Арифметические инструкции:

№	Команда (asm/hex)	Ожидаемый результат	Полученный результат	Примечания
1	ADDI x3, x0, 5 00500193	значение регистра x3=5	значение регистра x3=5	$x3 = 0 + 5 = 5$
2	ADDI x1, x0, 9 00900093	значение регистра x1=9	значение регистра x1=9	$x1 = 0 + 9 = 9$
3	ADDI x2, x0, 8 00800113	Значение регистра x2=8	Значение регистра x2=8	$x2 = 0 + 8 = 8$
4	ADDI x3, x0, 7 00700193	Значение регистра x3=7	Значение регистра x3=7	$x3 = 0 + 7 = 7$
5	ADDI x4, x0, 6 00600213	Значение регистра x4 = 6	Значение регистра x4 = 6	$x4 = 0 + 6 = 6$
6	ADDI x5, x0, 5 00500293	Значение регистра x5 = 5	Значение регистра x5 = 5	$x5 = 0 + 5 = 5$
7	ADDI x6, x0, 4 00400313	Значение регистра x6 = 4	Значение регистра x6 = 4	$x6 = 0 + 4 = 4$
8	ADDI x7, x0, 3 00300393	Значение регистра x7 = 3	Значение регистра x7 = 3	$x7 = 0 + 3 = 3$
9	ADDI x8, x0, 2 00200413	Значение регистра x8 = 2	Значение регистра x8 = 2	$x8 = 0 + 2 = 2$

10	ADDI x9, x0, 1 00100493	Значение регистра x9 = 1	Значение регистра x9 = 1	$x9 = 0 + 1 = 1$
11	ADD x10, x1, x2 00208533	Значение регистра x10 = 17	Значение регистра x10 = 17	$x10 = x1 + x2 = 9 + 8 = 17$
12	ADD x11, x3, x4 004185b3	Значение регистра x11 = 13	Значение регистра x11 = 13	$x11 = x3 + x4 = 7 + 6 = 13$
13	SUB x12, x6, x5 40530633	Значение регистра x12 = -1	Значение регистра x12 = -1	$x12 = x6 - x5 = 4 - 5 = -1$

Логические инструкции

№	Команда (assembler)	Ожидаемый результат	Полученный результат	Примечания
14	XORI x13, x9, 11 00A4C693	Значение регистра x13 = 10	Значение регистра x13 = 10	$x13 = x9 \text{ XOR } 11 = 1 \text{ XOR } 11 = 10$
15	ORI x14, x1, 5 0050E713	Значение регистра x14 = 13	Значение регистра x14 = 13	$x14 = x1 \text{ OR } 5 = 9 \text{ OR } 5 = 13$
16	ANDI x15, x1, 5 0050F793	Значение регистра x15 = 1	Значение регистра x15 = 1	$x15 = x1 \text{ AND } 5 = 9 \text{ AND } 5 = 1$
17	XOR x16, x1, x2 0020C833	Значение регистра x16 = 1	Значение регистра x16 = 1	$x16 = x1 \text{ XOR } x2 = 9 \text{ XOR } 8 = 1$
18	OR x17, x1, x2 0020E8B3	Значение регистра x17 = 9	Значение регистра x17 = 9	$x17 = x1 \text{ OR } x2 = 9 \text{ OR } 8 = 9$
19	AND x18, x1, x2 0020F933	Значение регистра x18 = 8	Значение регистра x18 = 8	$x18 = x1 \text{ AND } x2 = 9 \text{ AND } 8 = 8$

Инструкции сравнения

№	Команда (assembler)	Ожидаемый результат	Полученный результат	Примечания
20	SLTI x19, x1, 5 0050A993	Значение регистра $x19 = 0$	Значение регистра $x19 = 0$	$x19 = x1 < 5 = 9 < 5 = 0$ Установка 1, если $x1 < 5$ (знаковое сравнение), иначе 0: $x19 = (x1 < 5)$
21	SLT x20, x1, x2 0020AA33	Значение регистра $x20 = 0$	Значение регистра $x20 = 0$	$x20 = x1 < x2 = 9 < 8 = 0$ Установка 1, если $x1 < x2$ (знаковое сравнение), иначе 0: $x20 = (x1 < x2)$

Сдвиг

№	Команда (assembler)	Ожидаемый результат	Полученный результат	Примечания
20	SLLI x21, x1, 2 00209A93	Значение регистра $x21 = 36$	Значение регистра $x21 = 36$	$x21 = x1 \text{ SLLI } 2 = 9$ $\text{SLLI } 2 = 36$
21	SRLI x22, x1, 2 0020DB13	Значение регистра $x22 = 2$	Значение регистра $x22 = 2$	$x22 = x1 \text{ SRLI } 2 = 9$ $\text{SRLI } 2 = 2$ Логический сдвиг вправо: $x22 = x1 \gg 2$ (заполнение нулями)
22	SRAI x23, x1, 2 4020DB93	Значение регистра $x23 = 2$	Значение регистра $x23 = 2$	$x23 = x1 \text{ SRAI } 2 = 9$ $\text{SRAI } 2 = 2$ Арифметический сдвиг вправо: $x23 = x1 \gg 2$ (сохраняет знак)
23	SLL x24, x1, x9 00909C33	Значение регистра $x24 = 18$	Значение регистра $x24 = 18$	$x24 = x1 \text{ SLL } x9 = 9$ $\text{SLL } 1 = 18$ Логический сдвиг влево: $x24 = x1 \ll x9$ (количество битов из регистра x9)
24	SRL x25, x1, x9 0090dcb3	Значение регистра $x25 = 4$	Значение регистра $x25 = 4$	$x25 = x1 \text{ SRL } x9 = 9$ $\text{SRL } 1 = 4$
25	SRA x26, x1, x9	Значение	Значение	$x26 = x1 \text{ SRA } x9 = 9$

	4090dd33	регистра x26 = 4	регистра x26 = 4	SRA 1 = 4
--	----------	------------------	------------------	-----------

RV32M Standard Extensions

№	Команда (assembler)	Ожидаемый результат	Полученный результат	Примечания
26	MUL x27, x1, x8 02808db3	Значение регистра x27 = 18	Значение регистра x27 = 18	x27 = x1 MUL x8 = 9 MUL 2 = 18

Load Instructions

№	Команда (assembler)	Ожидаемый результат	Полученный результат	Примечания
26	SW x4, 0(x5) 0042a023	По адресу 5 (значение x5) будет записано значение 6 (значение x4)	По адресу 5 (значение x5) будет записано значение 6 (значение x4)	
27	SW x2, 0(x3) 0021a023	В памяти данных по адресу 7 значение 8	В памяти данных по адресу 7 значение 8	
28	LW x11, 0(x3) 000f2583	Ожидаем в регистре x30 = 8	Ожидаем в регистре x30 = 8	

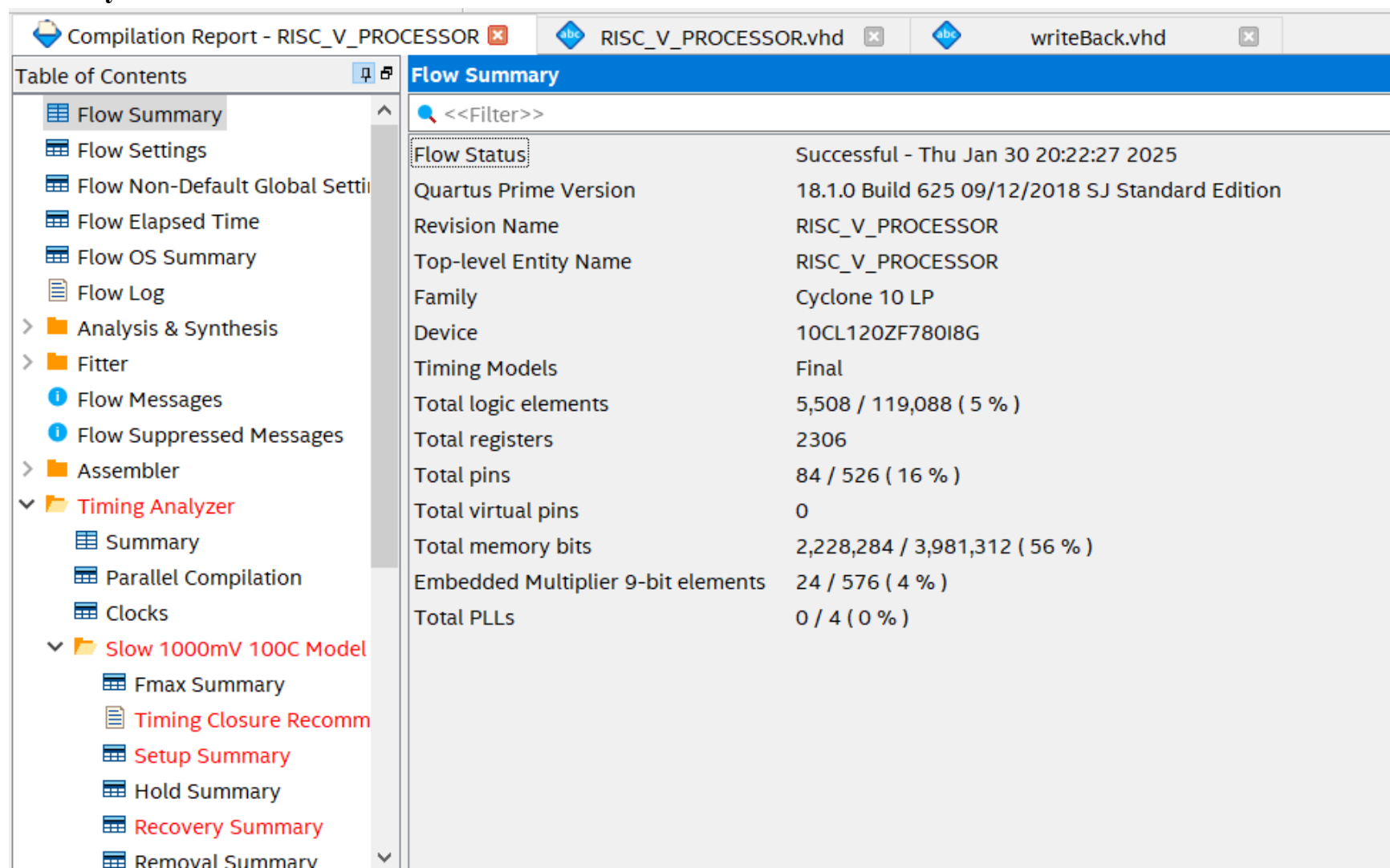
1. Полученный результат (из состояния регистрового файла)

[illegible]

2. Полученный результат (из памяти данных)

[illegible]

2.3. Результаты синтеза



Compilation Report - RISC_V_PROCESSOR

RISC_V_PROCESSOR.vhd

writeBack.vhd

Table of Contents

- Flow Summary
- Flow Settings
- Flow Non-Default Global Settings
- Flow Elapsed Time
- Flow OS Summary
- Flow Log
- Analysis & Synthesis
- Fitter
 - Flow Messages
 - Flow Suppressed Messages
- Assembler
- Timing Analyzer
 - Summary
 - Parallel Compilation
 - Clocks
 - Slow 1000mV 100C Model
 - Fmax Summary
 - Timing Closure Recommendation
 - Setup Summary
 - Hold Summary
 - Recovery Summary
 - Removal Summary

Flow Summary

<<Filter>>

Flow Status	Successful - Thu Jan 30 20:22:27 2025
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Standard Edition
Revision Name	RISC_V_PROCESSOR
Top-level Entity Name	RISC_V_PROCESSOR
Family	Cyclone 10 LP
Device	10CL120ZF780I8G
Timing Models	Final
Total logic elements	5,508 / 119,088 (5 %)
Total registers	2306
Total pins	84 / 526 (16 %)
Total virtual pins	0
Total memory bits	2,228,284 / 3,981,312 (56 %)
Embedded Multiplier 9-bit elements	24 / 576 (4 %)
Total PLLs	0 / 4 (0 %)

Рис. 1. Затраченные ресурсы при синтезе

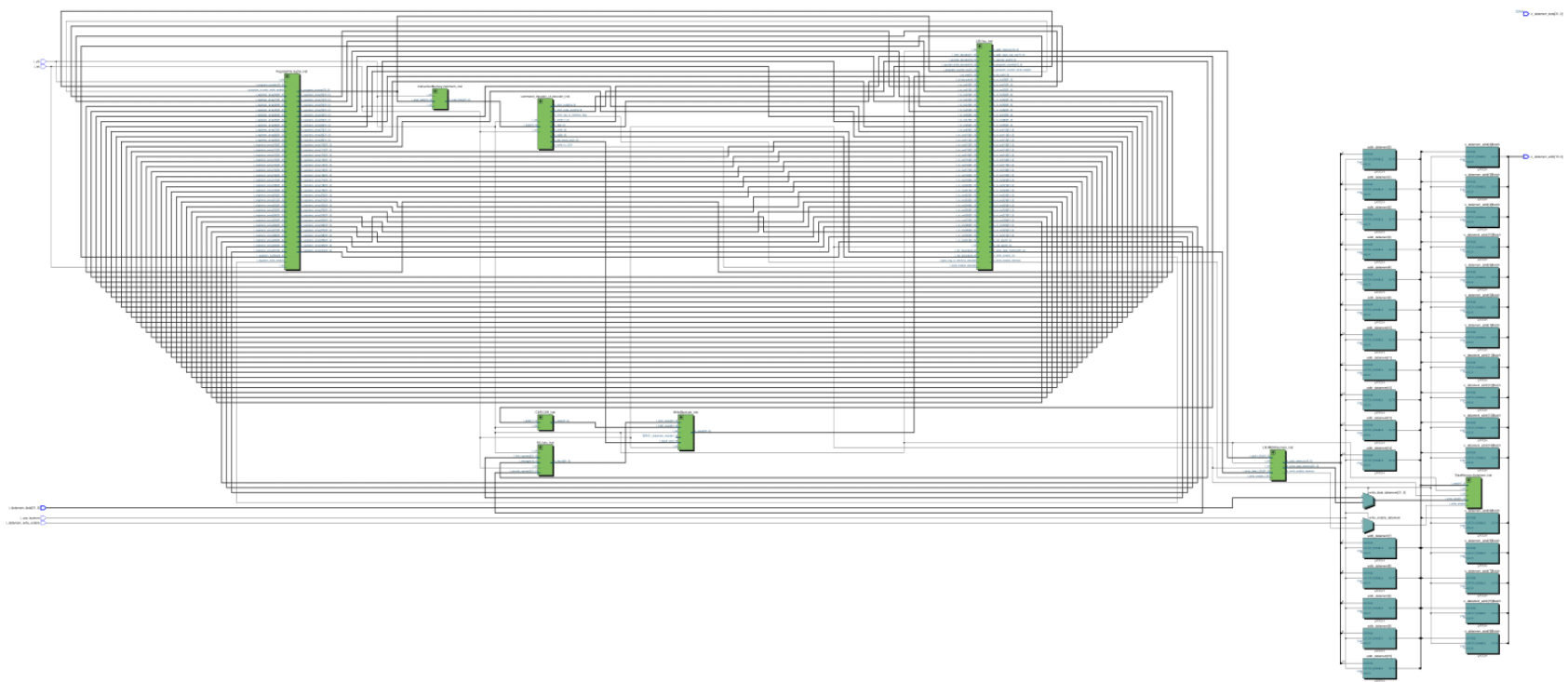


Рис. 2. Результат синтеза

ЗАКЛЮЧЕНИЕ

В рамках курсовой работы была изучена RISC-V архитектура процессора. Был разработан блок записи результата, проведена общая сборка проекта и его тестирование. Получен опыт работы с VHDL, языком описания аппаратуры интегральных схем, с программным обеспечением Quartus Prime 18.1 и Modelsim SE.

