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: Декодер команд

Выполнили:

студенты группы С22-501

Борисова Софья

Васильев Александр

Галимьянов Матвей

Москва 2024

Оглавление

1. Постановка задачи

В рамках данной 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.

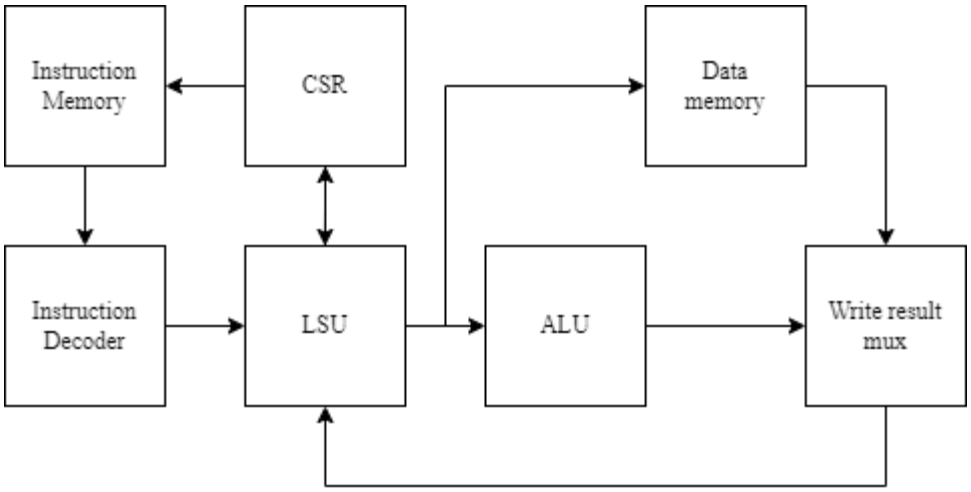

Рисунок 1. Структурная схема модели IP-ядра RISC-V.

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 рамках группового задания разработан декодер инструкций (instruction decoder), предназначенного для:

1. Декодирования типа инструкций (I, R, S).
2. Декодирования типа операций.

3. Декодирования адреса первого операнда, второго операнда/мгновенного значения, регистра назначения, передача сигналов на другие устройства в определенные такты

2. Спецификация

2.1 Условное графическое обозначение и список портов ввода-вывода

Условное графическое обозначение разрабатываемого декодера инструкций представлено на рисунке 1.

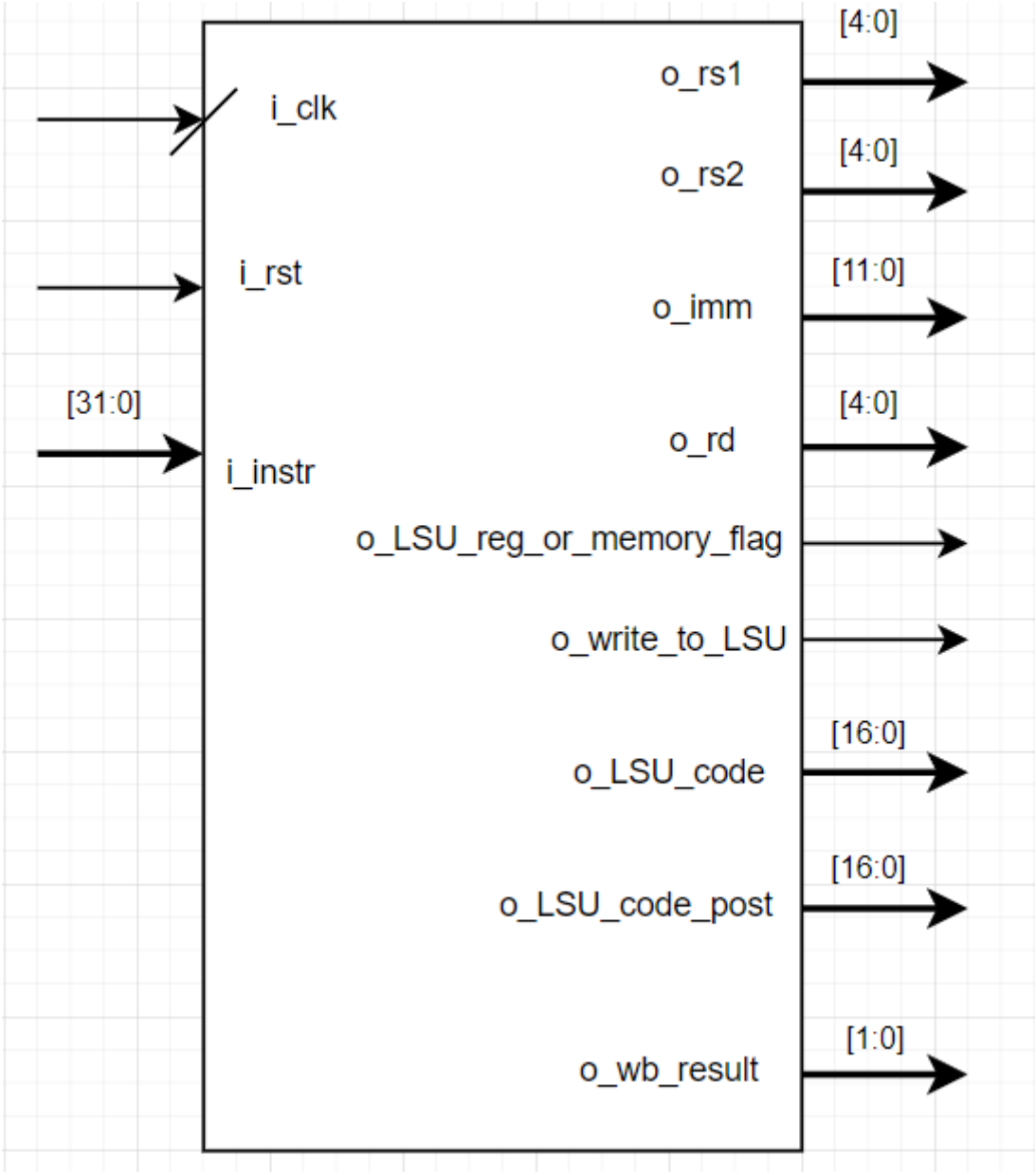

Рисунок 1 – Условное графического обозначение реверсивного счётчика.

Описание типов сигналов разрабатываемого устройства представлена в таблице 1.

Номер	Сигнал	Активный уровень	Тип	Описание
1	i_clk	↑	in	Тактовый сигнал, используется для синхронизации работы устройства.
2	i_rst	1	in	Сигнал сброса, активен при 1. При активации все внутренние регистры и выходные сигналы сбрасываются в нули.
3	i_instr [31:0]	data	in	Входная шина, содержащая инструкцию RISC-V с операционным кодом и полями.
4	o_rs1[4:0]	addr	out	Номер первого исходного регистра (RS1) из инструкции.
5	o_rs2[4:0]	addr	out	Номер второго исходного регистра (RS2) из инструкции.
6	o_imm[11:0]	data	out	Немедленное значение (imm), извлекаемое из инструкций типов I и S.
7	o_rd[4:0]	addr	out	Номер регистра назначения (RD), в который записывается результат.
8.	o_LSU_reg_or_memory_flag	1	out	Если 1, то чтение из спец регистров, если 0, то из памяти
9	o_write_to_LSU	1	out	Сигнал разрешения записи для блока Load-Store Unit (LSU).
10	o_LSU_code[16:0]	data	out	Код операции для блока LSU, формируется на основе полей инструкции (funct3, funct7 и opcode).
11	o_LSU_code_past[16:0]	data	out	Код операции для блока LSU, прошедший через конвейер на

				основе полей инструкции (funct3, funct7 и opcode)
12	o_wb_result	data	out	Сигнал, который указывает блоку WrireBack откуда читать данные: "00" - ALU; "01" - datamem; "11" – ничего не делать

Таблица 1 – Список типов сигналов в разработанном реверсивном счётчике.

Типы операций, которые обрабатывает разработанное устройство указаны в таблице 2.

Типы операций	**Суть**	**opcode**
R - type	Register / Регистровые операции	0110011
I - type	Immediate / Использует регистр и одно немедленное значение (инструкции часто применяются для арифметических операций с константами, а также для загрузки данных и управления потоком)	0000011, 0010011
S - type	Store / Запись данных	0100011

Таблица 2 – Список операций, которые обрабатывает декодер команд.

Инструкции загрузки, предназначенные для загрузки данных из памяти в регистры указаны в таблице 3.

Instruction	**Encoding**	**Binary**
LB	imm[11:0] rs1 000 rd(10000) 0000011	00000000000000011
LH	imm[11:0] rs1 001 rd 0000011	00000000010000011
LW	imm[11:0](0…) rs1(00001) 010 rd(10000) 0000011	00000000100000011
LBU	imm[11:0] rs1 100 rd 0000011	00000001000000011
LHU	imm[11:0] rs1 101 rd 0000011	00000001010000011

Таблица 3 – Load Instructions.

Инструкции сохранения, предназначенные для записи данных из регистров в память указаны в таблице 4.

Instruction	Encoding	Binary
SB	imm[11:5] rs2 rs1 000 imm[4:0] 0100011	00000000000100011
SH	imm[11:5] rs2 rs1 001 imm[4:0] 0100011	00000000010100011
SW	imm[11:5] rs2 rs1 010 imm[4:0] 0100011	00000000100100011

Таблица 4 – Store Instructions.

Инструкции сдвига, предназначенные для перемещения битов внутри регистра на определенное количество позиций влево или вправо указаны в таблице 5.

Instruction	Encoding	Binary
SLLI	0000000 shamt(01100) rs1(10111) 001 rd(00011) 0010011	00000000010010011
SRLI	0000000 shamt rs1 101 rd 0010011	00000001010010010
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

Таблица 5 – Shift Instructions.

Арифметические инструкции, предназначенные для выполнения арифметических операций над числовыми данными указаны в таблице 6.

Instruction	Encoding	Binary
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

Таблица 5 – Arithmetic Instructions.

Логические инструкции, предназначенные для выполнения побитовых логических операций над данными указаны в таблице 6.

Instruction	Encoding	Binary
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

Таблица 6 – Logical Instructions.

Инструкции сравнения, предназначенные для сравнения двух операнд и устанавливают флаги состояния в зависимости от результатов сравнения указаны в таблице 7.

Instruction	Encoding	Binary
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

Таблица 7 – Compare Instructions.

Инструкции RV32M для целочисленного умножения и деления указаны в таблице 8.

Instruction	Encoding	Binary
MUL	0000001 rs2 rs1 000 rd 0110011	00000010000110011
MULH	0000001 rs2 rs1 001 rd 0110011	00000010010110011
MULHSU	0000001 rs2 rs1 010 rd 0110011	00000010100110011
MULHU	0000001 rs2 rs1 011 rd 0110011	00000010110110011

Таблица 8 – RV32M Standart Extensions.

2.2 Описание рабочего режима

Декодер обеспечивает конвейерную обработку команд, обеспечивая синхронизацию тактовым сигналом. Декодер работает на основе тактового сигнала i_clk и сигнала сброса i_rst.

Основные функции декодера**:**

1. Декодирование инструкций:
   - Декодер анализирует 32-битную инструкцию i_instr и выделяет из неё поля:
     - o_rs1 и o_rs2 — номера регистров-источников (source registers).
     - o_rd — номер регистра-назначения (destination register).
     - o_imm — непосредственное значение (immediate value), используемое в инструкциях.
     - o_LSU_code и o_LSU_code_post — коды для управления блоком LSU.
     - o_read_to_LSU и o_write_to_LSU — сигналы чтения и записи для LSU.
     - o_LSU_reg_or_memory_flag — флаг, указывающий, работает ли LSU с регистрами или памятью.
     - o_wb_result_src — сигнал, определяющий источник данных для записи в регистр (WriteBack).

2. Работа с тактовым сигналом:

- Декодер синхронизируется по фронту тактового сигнала i_clk. Все изменения в выходных сигналах происходят только на rising edge (положительном фронте) тактового сигнала.
- При активном сигнале сброса i_rst = '1' все внутренние регистры и выходные сигналы сбрасываются в нулевое состояние.

3. Конвейерная обработка:

- Декодер использует конвейерную обработку для передачи управляющих сигналов через несколько стадий:
  - reg_stage_LSU_1, reg_stage_LSU_2, reg_stage_LSU_3, reg_stage_LSU_4 — регистры для хранения промежуточных данных, связанных с LSU.
  - wb_result_src_1 и wb_result_src_2 — регистры для хранения информации о источнике данных для WriteBack.
- Данные последовательно передаются через стадии конвейера, что позволяет обеспечить задержку в обработке сигналов и синхронизацию с другими блоками процессора.

4. Обработка различных типов инструкций:

- Декодер поддерживает следующие типы инструкций:
  - R-тип (арифметико-логические операции): i_instr(6 downto 0) = "0110011".
  - I-тип (операции с непосредственным значением): i_instr(6 downto 0) = "0010011".
  - Load-инструкции (загрузка из памяти): i_instr(6 downto 0) = "0000011".
  - Store-инструкции (сохранение в память): i_instr(6 downto 0) = "0100011".
- В зависимости от типа инструкции декодер генерирует соответствующие управляющие сигналы:
  - Для R-типа и I-типа инструкций устанавливаются сигналы o_read_to_LSU и o_write_to_LSU, а также формируется код для LSU.
  - Для Load-инструкций дополнительно устанавливается флаг o_LSU_reg_or_memory_flag, указывающий на работу с памятью.
  - Для Store-инструкций формируется immediate значение и сигналы для записи в память.

5. Генерация сигналов WriteBack:

- В зависимости от типа инструкции декодер определяет источник данных для записи в регистр:
  - Для Load-инструкций (i_instr(6 downto 0) = "0000011") источником данных является память (o_wb_result_src = "01").

- Для арифметико-логических операций (R-тип и I-тип) источником данных является ALU (o_wb_result_src = "00").
- Для остальных случаев (например, NOP) используется значение по умолчанию (o_wb_result_src = "11").

6. Обработка immediate значения:

- Для инструкций I-типа и Load/Store immediate значение извлекается из соответствующих битов инструкции:
  - Для I-типа и Load-инструкций immediate значение находится в битах i_instr(31 downto 20).
  - Для Store-инструкций immediate значение формируется из битов i_instr(31 downto 25) и i_instr(11 downto 7).

Особенности работы**:**

- Декодер поддерживает конвейерную обработку, что позволяет ему работать с высокой производительностью.
- Все выходные сигналы обновляются только на тактовом фронте, что обеспечивает синхронность работы с другими блоками процессора.
- При сбросе (i_rst = '1') все внутренние регистры и выходные сигналы сбрасываются в нулевое состояние, что обеспечивает корректное начало работы процессора.

Таким образом, декодер команд command_decoder_v2 обеспечивает корректное декодирование инструкций и генерацию управляющих сигналов для работы процессора.

2.3. Подключение к соседним модулям.

Схема подключения памяти инструкций, декодера инструкций и LSU представлена на рисунке 2.

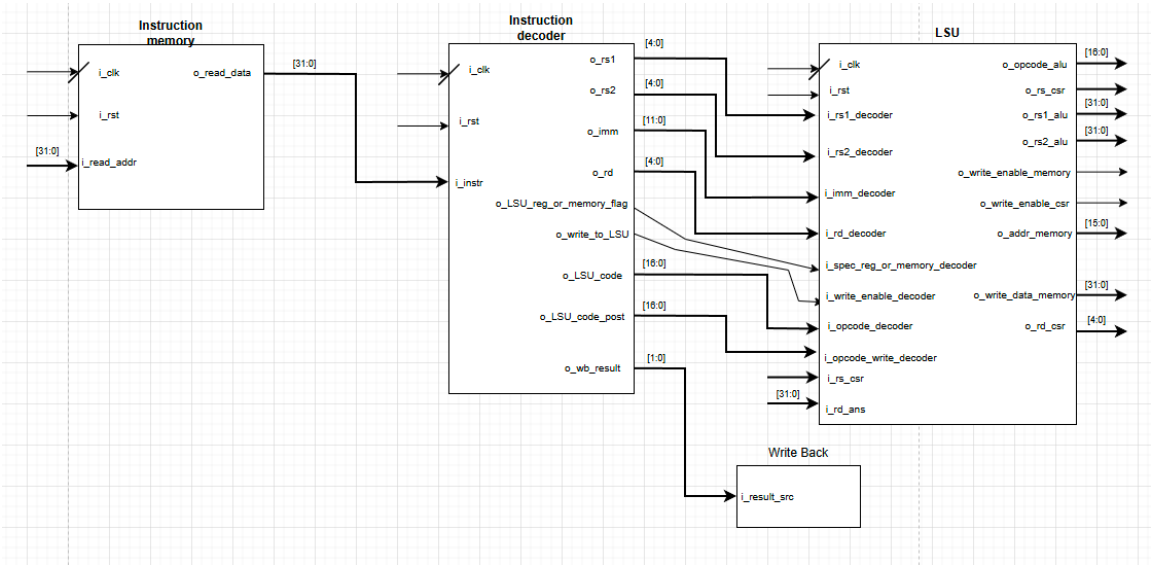

Рисунок 2 – Схема подключения памяти инструкций, декодера инструкций и LSU.

3. Тестирование

3.1. Тестирование разработанного декодера устройства

Такт вызова команды	Значение инструкции i_instr	Инструкция	Описание
1	11000000111001111000010000000011	LB	В 0-ой такт отправляем LSU: o_rs1 = i_instr(19 downto 15) = "01111"; o_imm = i_instr(31 downto 20) = "110000001110"; o_LSU_code = "00000000000000011" На 2-ой такт отправляем на WriteBack сигнал o_wb_result_src = "01"; На 3-ий такт отправляем на LSU o_rd = i_instr(19 downto 15) = "01111"; o_LSU_code_post = "00000000000000011";
2	11111111111111111111111111111111	NOP	В 0-ой такт отправляем LSU: o_rs1 = i_instr(19 downto 15) = "00000"; o_imm = i_instr(31 downto 20) = "000000000000"; o_LSU_code = "00000000000000000" На 2-ой такт отправляем на WriteBack сигнал o_wb_result_src = "11"; На 3-ий такт отправляем на LSU o_rd = i_instr(19 downto 15) = "00000"; o_LSU_code_post = "00000000000000000"; o_write_to_LSU = 0
3	00001010001101000001001000000011	LH	0-й такт: o_rs1 = i_instr(19 downto 15) = "10000"; o_imm = i_instr(31 downto 20) = "000010100011"; o_LSU_code = "00000000000000011"; 2-й такт: o_wb_result_src = "01"; 3-й такт: o_rd = i_instr(11 downto 7) = "01000"; o_LSU_code_post = "00000000000000011"; o_write_to_LSU = '0';
4	10101000010110001010100000000011	LW	0-й такт: o_rs1 = i_instr(19 downto 15) = "10001"; o_imm = i_instr(31 downto 20) = "101010000101"; o_LSU_code = "00000000000000011"; 2-й такт: o_wb_result_src = "01"; 3-й такт: o_rd = i_instr(11 downto 7) = "01000"; o_LSU_code_post = "00000000000000011"; o_write_to_LSU = '0';
5	11100100000110010100000000000011	LBU	0-й такт: o_rs1 = i_instr(19 downto 15) = "10010"; o_imm = i_instr(31 downto 20) = "111001000001"; o_LSU_code = "00000000000000011"; 2-й такт: o_wb_result_src = "01"; 3-й такт: o_rd = i_instr(11 downto 7) = "01000"; o_LSU_code_post = "00000000000000011"; o_write_to_LSU = '0';
6	11000011101010100101000110000011	LHU	0-й такт: o_rs1 = i_instr(19 downto 15) = "10100"; o_imm = i_instr(31 downto 20) = "110000111010"; o_LSU_code = "00000000000000011"; 2-й такт:

			o_wb_result_src = "01"; 3-й такт: o_rd = i_instr(11 downto 7) = "00011"; o_LSU_code_post = "00000000000000011"; o_write_to_LSU = '0';
7	00011001100111011000011000100011	SB	0-й такт: o_rs1 = i_instr(19 downto 15) = "11011"; o_rs2 = i_instr(24 downto 20) = "11001"; o_imm(4 downto 0) = i_instr(11 downto 7) = "00100"; o_imm(11 downto 5) = i_instr(31 downto 25) = "0001100"; o_LSU_code = "00000000000000011"; 2-й такт: o_wb_result_src = "11"; 3-й такт: o_LSU_code_post = "00000000000000011"; o_write_to_LSU = '1';
8	01010000110001110001100000100011	SH	0-й такт: o_rs1 = i_instr(19 downto 15) = "11110"; o_rs2 = i_instr(24 downto 20) = "00001"; o_imm(4 downto 0) = i_instr(11 downto 7) = "00000"; o_imm(11 downto 5) = i_instr(31 downto 25) = "0101000"; o_LSU_code = "00000000000000011"; 2-й такт: o_wb_result_src = "11"; 3-й такт: o_LSU_code_post = "00000000000000011"; o_write_to_LSU = '1';
9	10111000011110001010110110100011	SW	0-й такт: o_rs1 = i_instr(19 downto 15) = "10001"; o_rs2 = i_instr(24 downto 20) = "11011"; o_imm(4 downto 0) = i_instr(11 downto 7) = "01011"; o_imm(11 downto 5) = i_instr(31 downto 25) = "1011100"; o_LSU_code = "00000000000000011"; 2-й такт: o_wb_result_src = "11"; 3-й такт: o_LSU_code_post = "00000000000000011"; o_write_to_LSU = '1';
10	00000000110010111001000110010011	SRLI	0-й такт: o_rs1 = i_instr(19 downto 15) = "10111"; o_imm = i_instr(31 downto 20) = "000000001100"; o_LSU_code = "00000000000000011"; 2-й такт: o_wb_result_src = "00"; 3-й такт: o_rd = i_instr(11 downto 7) = "00011"; o_LSU_code_post = "00000000000000011"; o_write_to_LSU = '0';
11	01000001100000111101011110010011	SRAI	0-й такт: o_rs1 = i_instr(19 downto 15) = "00111"; o_imm = i_instr(31 downto 20) = "010000011000"; o_LSU_code = "00000000000000011"; 2-й такт: o_wb_result_src = "00"; 3-й такт: o_rd = i_instr(11 downto 7) = "11111"; o_LSU_code_post = "00000000000000011"; o_write_to_LSU = '0';

12	00000001000110010001100000110011	SLL	0-й такт: o_rs1 = i_instr(19 downto 15) = "10010"; o_rs2 = i_instr(24 downto 20) = "01000"; o_LSU_code = "0000000000000011"; 2-й такт: o_wb_result_src = "00"; 3-й такт: o_rd = i_instr(11 downto 7) = "00000"; o_LSU_code_post = "0000000000000011"; o_write_to_LSU = '0';
13	00000001000111001101011100110011	SRL	0-й такт: o_rs1 = i_instr(19 downto 15) = "11001"; o_rs2 = i_instr(24 downto 20) = "10101"; o_LSU_code = "0000000000000011"; 2-й такт: o_wb_result_src = "00"; 3-й такт: o_rd = i_instr(11 downto 7) = "11110"; o_LSU_code_post = "0000000000000011"; o_write_to_LSU = '0';
14	01000000101010010101101100110011	SRA	0-й такт: o_rs1 = i_instr(19 downto 15) = "10010"; o_rs2 = i_instr(24 downto 20) = "10110"; o_LSU_code = "0000000000000011"; 2-й такт: o_wb_result_src = "00"; 3-й такт: o_rd = i_instr(11 downto 7) = "11011"; o_LSU_code_post = "0000000000000011"; o_write_to_LSU = '0';
15	00000111011100111000101010010011	ADDI	0-й такт: o_rs1 = i_instr(19 downto 15) = "10010"; o_imm = i_instr(31 downto 20) = "000000100011"; o_LSU_code = "0000000000000011"; 2-й такт: o_wb_result_src = "00"; 3-й такт: o_rd = i_instr(11 downto 7) = "00010"; o_LSU_code_post = "0000000000000011"; o_write_to_LSU = '0';
16	00000001000111100000101110110011	ADD	0-й такт: o_rs1 = i_instr(19 downto 15) = "11110"; o_rs2 = i_instr(24 downto 20) = "01011"; o_LSU_code = "0000000000000011"; 2-й такт: o_wb_result_src = "00"; 3-й такт: o_rd = i_instr(11 downto 7) = "11111"; o_LSU_code_post = "0000000000000011"; o_write_to_LSU = '0';
17	01000000100000001000000110110011	SUB	0-й такт: o_rs1 = i_instr(19 downto 15) = "00001"; o_rs2 = i_instr(24 downto 20) = "00011"; o_LSU_code = "0000000000000011"; 2-й такт: o_wb_result_src = "00"; 3-й такт: o_rd = i_instr(11 downto 7) = "00011"; o_LSU_code_post = "0000000000000011"; o_write_to_LSU = '0';
18	10101101010111111100100100010011	XORI	0-й такт: o_rs1 = i_instr(19 downto 15) = "11111";

			o_imm = i_instr(31 downto 20) = "101011010101"; o_LSU_code = "00000000000000011"; 2-й такт: o_wb_result_src = "00"; 3-й такт: o_rd = i_instr(11 downto 7) = "10010"; o_LSU_code_post = "00000000000000011"; o_write_to_LSU = '0';
19	10111000100110011110101010010011	ORI	0-й такт: o_rs1 = i_instr(19 downto 15) = "10011"; o_imm = i_instr(31 downto 20) = "101110001001"; o_LSU_code = "00000000000000011"; 2-й такт: o_wb_result_src = "00"; 3-й такт: o_rd = i_instr(11 downto 7) = "10101"; o_LSU_code_post = "00000000000000011"; o_write_to_LSU = '0';
20	10100010010001000111011110010011	ANDI	0-й такт: o_rs1 = i_instr(19 downto 15) = "10000"; o_imm = i_instr(31 downto 20) = "101000100100"; o_LSU_code = "00000000000000011"; 2-й такт: o_wb_result_src = "00"; 3-й такт: o_rd = i_instr(11 downto 7) = "11111"; o_LSU_code_post = "00000000000000011"; o_write_to_LSU = '0';
21	00000001111101010100100110110011	XOR	0-й такт: o_rs1 = i_instr(19 downto 15) = "01010"; o_rs2 = i_instr(24 downto 20) = "10011"; o_LSU_code = "00000000000000011"; 2-й такт: o_wb_result_src = "00"; 3-й такт: o_rd = i_instr(11 downto 7) = "10011"; o_LSU_code_post = "00000000000000011"; o_write_to_LSU = '0';
22	00000000110111101110111000110011	OR	0-й такт: o_rs1 = i_instr(19 downto 15) = "11101"; o_rs2 = i_instr(24 downto 20) = "11100"; o_LSU_code = "00000000000000011"; 2-й такт: o_wb_result_src = "00"; 3-й такт: o_rd = i_instr(11 downto 7) = "11100"; o_LSU_code_post = "00000000000000011"; o_write_to_LSU = '0';
23	00000001111111101111010000110011	AND	0-й такт: o_rs1 = i_instr(19 downto 15) = "11101"; o_rs2 = i_instr(24 downto 20) = "10100"; o_LSU_code = "00000000000000011"; 2-й такт: o_wb_result_src = "00"; 3-й такт: o_rd = i_instr(11 downto 7) = "10100"; o_LSU_code_post = "00000000000000011"; o_write_to_LSU = '0';
24	00000010001100010010000100010011	SLTI	0-й такт: o_rs1 = i_instr(19 downto 15) = "10010"; o_imm = i_instr(31 downto 20) = "000000100011";

			o_LSU_code = "0000000000000011"; 2-й такт: o_wb_result_src = "00"; 3-й такт: o_rd = i_instr(11 downto 7) = "00010"; o_LSU_code_post = "0000000000000011"; o_write_to_LSU = '0';
25	01010101111101110011111110010011	SLTIU	0-й такт: o_rs1 = i_instr(19 downto 15) = "11110"; o_imm = i_instr(31 downto 20) = "010101011111"; o_LSU_code = "0000000000000011"; 2-й такт: o_wb_result_src = "00"; 3-й такт: o_rd = i_instr(11 downto 7) = "11111"; o_LSU_code_post = "0000000000000011"; o_write_to_LSU = '0';
26	00000001100101001010000010110011	SLT	0-й такт: o_rs1 = i_instr(19 downto 15) = "01001"; o_rs2 = i_instr(24 downto 20) = "00001"; o_LSU_code = "0000000000000011"; 2-й такт: o_wb_result_src = "00"; 3-й такт: o_rd = i_instr(11 downto 7) = "00001"; o_LSU_code_post = "0000000000000011"; o_write_to_LSU = '0';
27	00000000000110100011100010110011	SLTU	0-й такт: o_rs1 = i_instr(19 downto 15) = "10100"; o_rs2 = i_instr(24 downto 20) = "11000"; o_LSU_code = "0000000000000011"; 2-й такт: o_wb_result_src = "00"; 3-й такт: o_rd = i_instr(11 downto 7) = "10001"; o_LSU_code_post = "0000000000000011"; o_write_to_LSU = '0';

Временные диаграммы полученные при прохождении тестов представлены на рисунках №3-6.

На рисунке №3 поступает сигнал сброса - i_rst (1) и происходит заполнение массива регистров тестовыми значениями. После чего начинают поступать команды, в LSU первая команда на выполнение приходит только в 3 такт (3), хотя инструкция поступает во 2 такт (2), эта задержка происходит из-за того что декодер обрабатывает инструкцию 1 такт.

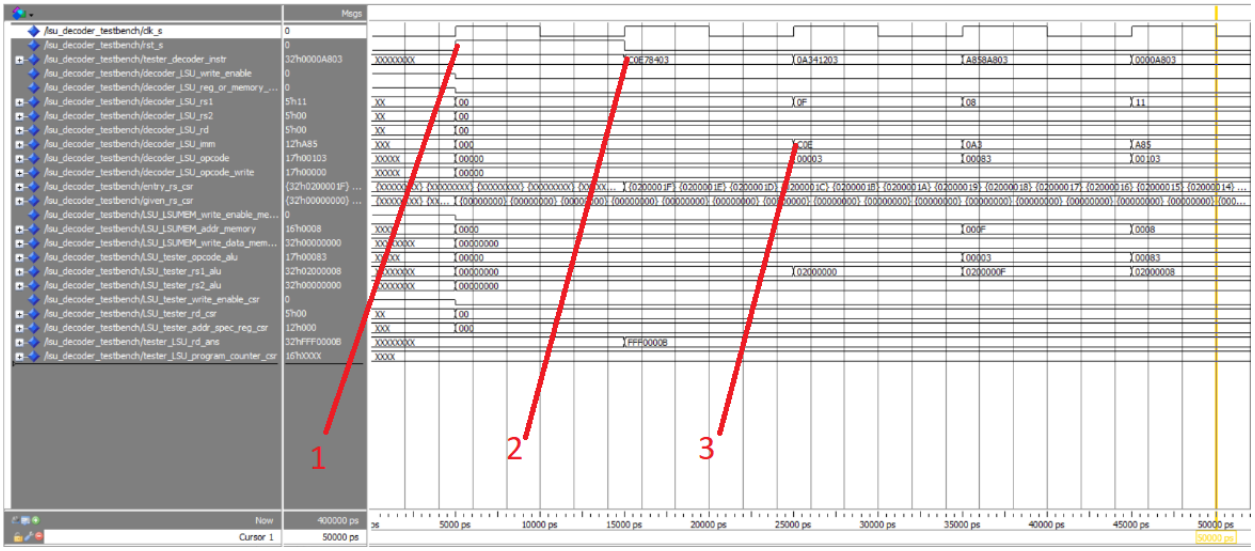

Рисунок 3. Временная диаграмма прохождения теста LSU с декодером.

На рисунке № 4 можно увидеть что декодер с LSU работают в конвейерном режиме. На декодер поступает инструкция (1), декодер обрабатывает инструкцию и направляет opcode на LSU (2), LSU по полученному opcode’у создаёт нужные сигналы (3).

LSU (2), LSU по полученному opcode’у создаёт нужные сигналы (3).

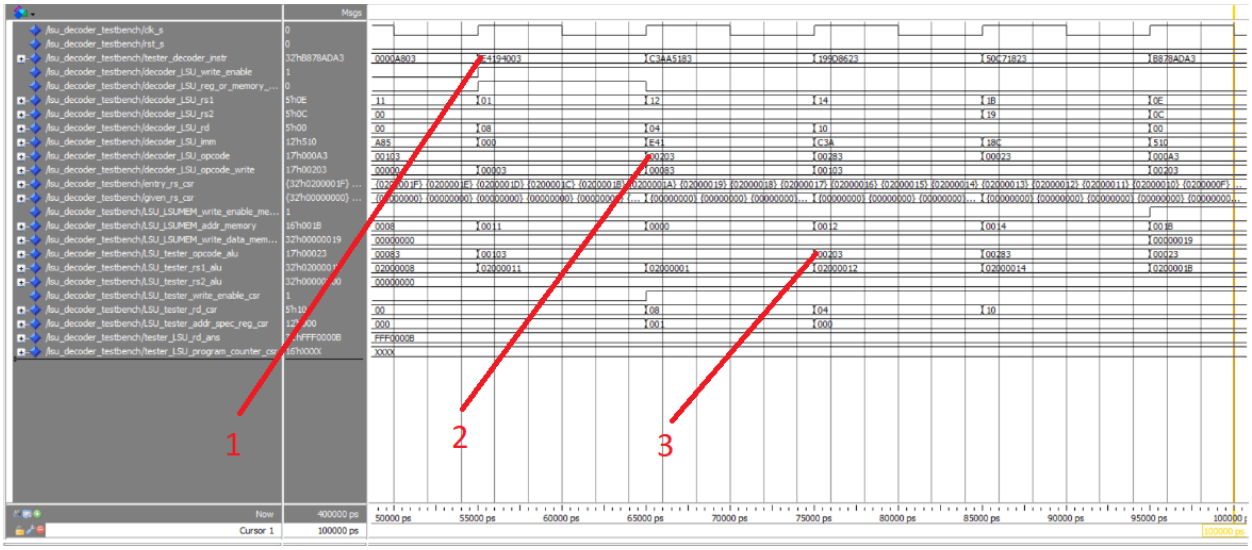

Рисунок 4. Временная диаграмма прохождения теста LSU с декодером.

На рисунке №5 начинают поступать команды, при которых должны происходить вычисления на АЛУ. Поступление инструкции SLLI (1), передача opcode’а на LSU (4), передача номеров регистров из которых нужно взять данные для вычислений (2) и (3) (в данном случае номера регистров не изменились с прошлой операции, по этому номера

регистров не изменились), передача данных с регистров в АЛУ (5) и (6), передача opcode'а в АЛУ (7).

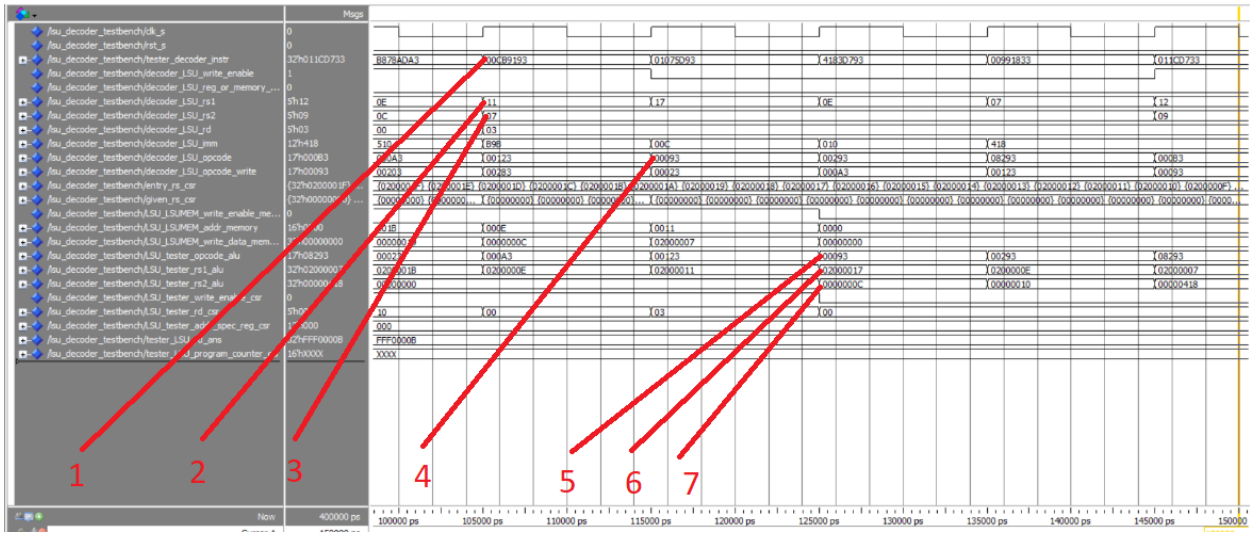

Рисунок 5. Временная диаграмма прохождения теста LSU с декодером.

На рисунке №6 можно заметить, что сигналы на LSU продолжают изменяться до 38 такта (2), хотя последняя команда поступила в 33 такт (1). Это происходит из-за задержки в один такт на декодере, а также из-за того что запись в регистры общего назначения должны происходит через 4 такта (ожидание подачу данных на запись от writeBack'а).

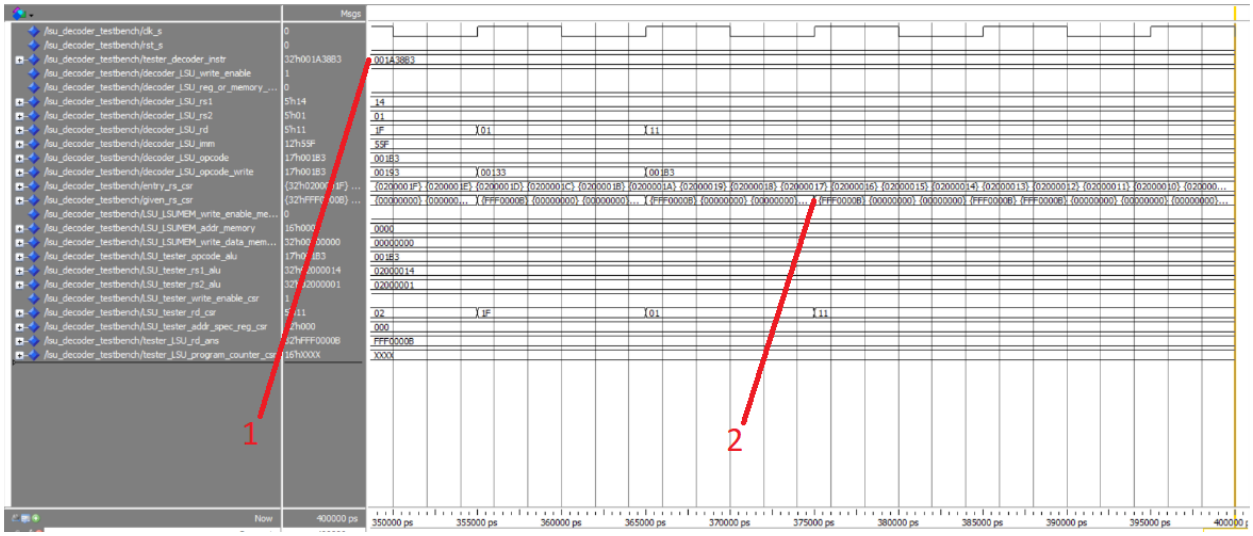

Рисунок 6. Временная диаграмма прохождения теста LSU с декодером.

Покрытие тестов показано на рисунке 7. Исходя из этого тестовые сценарии отработали успешно.

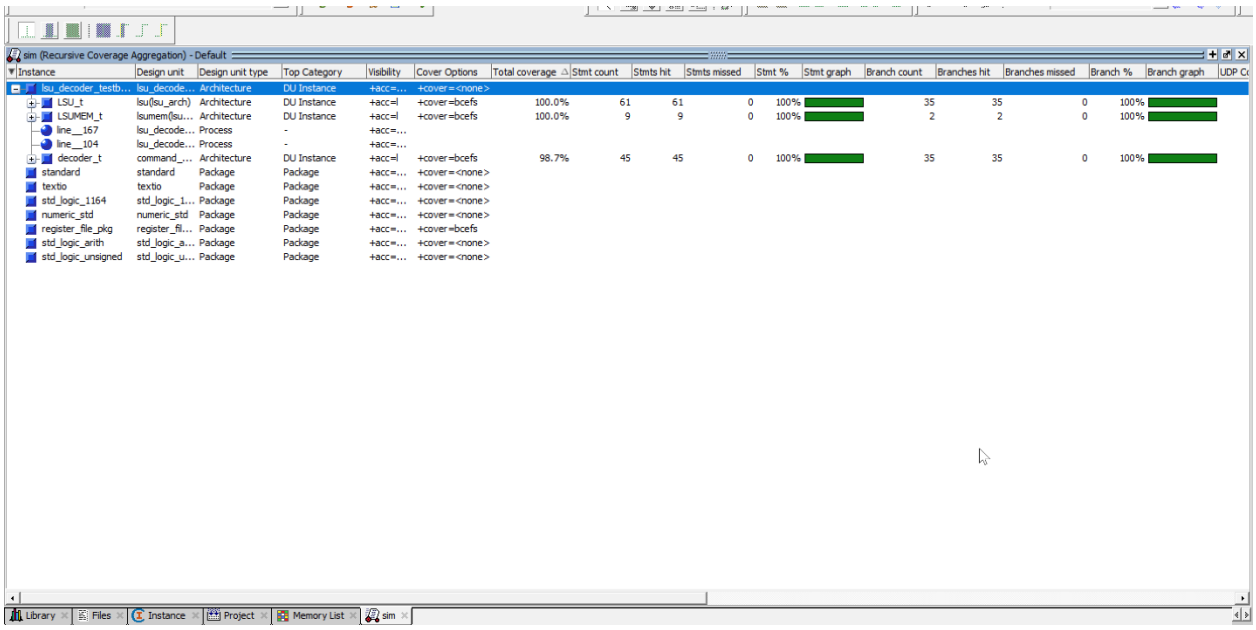

Рисунок 7. Покрытие тестовых сценариев.

3.2. Верификация с соседними блоками.

Согласно рисунку 2 блок LSU соединен с блоком декодера инструкций через следующие шины/сигналы:

1. Адресную шину (o_rs1 [4:0])
2. Адресную шину (o_rs2 [4:0])
3. Шину данных (o_imm [11:0])
4. Адресную шину (o_rd [4:0])
5. Сигнал разрешения (o_write_to_LSU)
6. Шину данных (o_LSU_code [16:0])
7. Шину данных (o_LSU_code_post [16:0])
8. Сигнал разрешение (o_LSU_reg_or_memory_flag)

Согласно рисунку 2 блок Instruction Memory соединен с блоком Instruction Decoder через шину данных (i_instr [31:0]).

Модульное тестирование с блоком LSU прошло успешно (рисунки 3, 4, 5, 6, 7)

4. Результаты синтеза

RTL схема по результатам синтеза проиллюстрирована на рисунке 8.

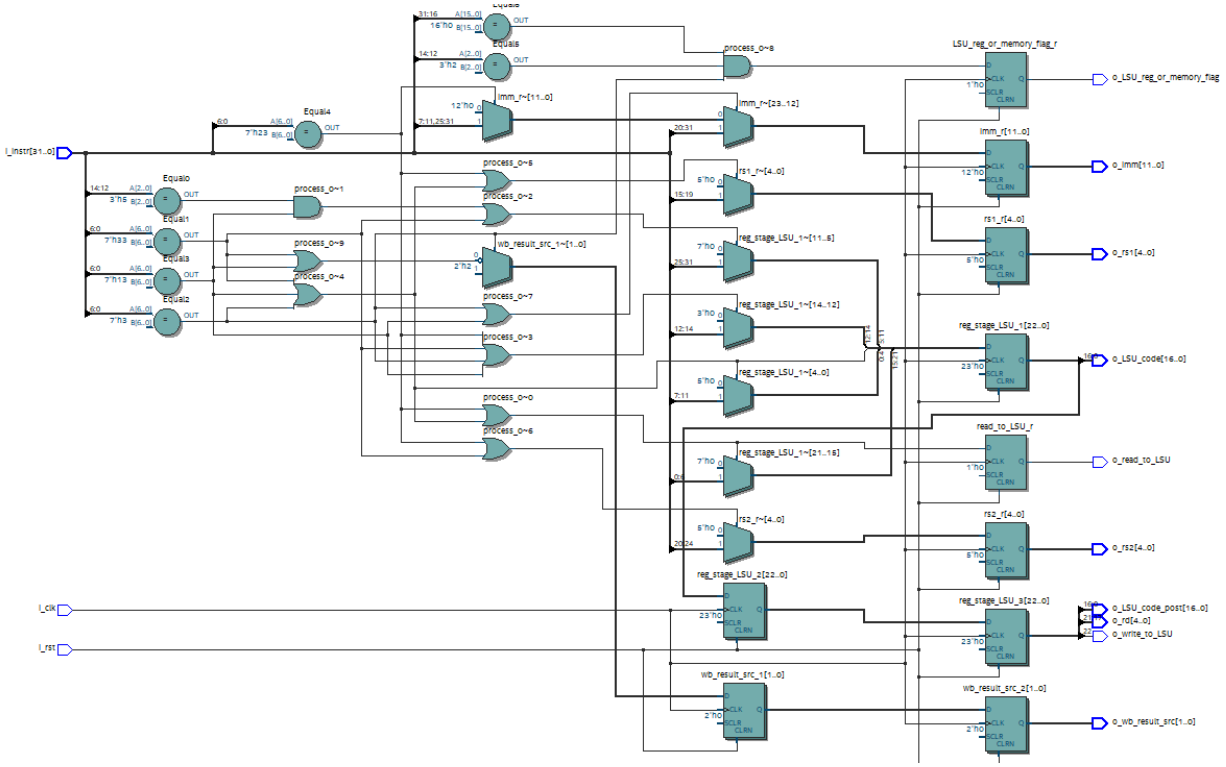

Рисунок 8 – RTL схема разработанного устройства.

Flow Summary

<<Filter>>

Flow Status	Successful - Wed Mar 05 20:57:25 2025
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	scheme
Top-level Entity Name	command_decoder_v1
Family	Cyclone 10 LP
Total logic elements	102 / 6,272 (2 %)
Total registers	88
Total pins	100 / 177 (56 %)
Total virtual pins	0
Total memory bits	0 / 276,480 (0 %)
Embedded Multiplier 9-bit elements	0 / 30 (0 %)
Total PLLs	0 / 2 (0 %)
Device	10CL006YU256C6G
Timing Models	Final

Рисунок 9 – список затраченных ресурсов декодера команд

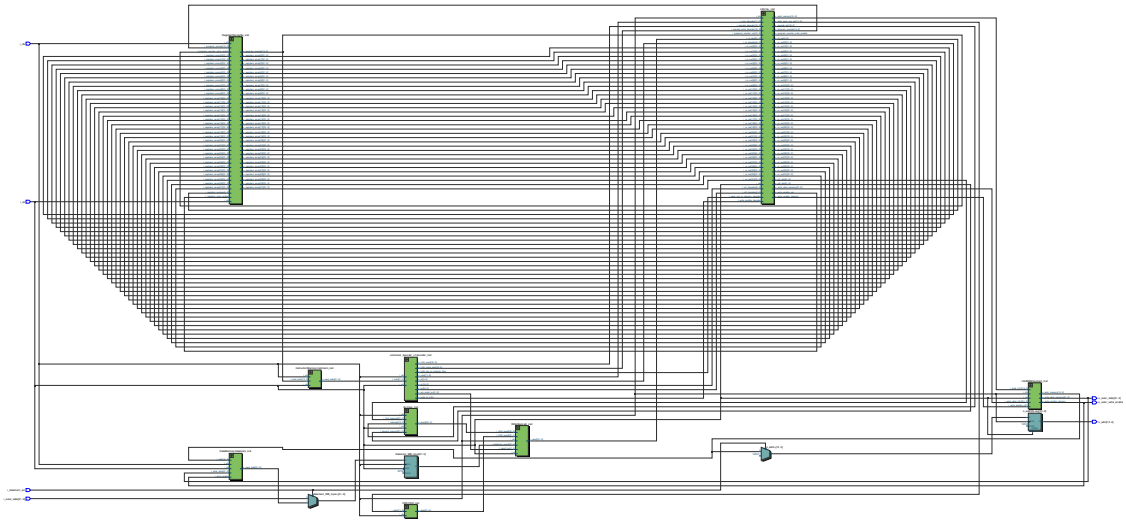

Рисунок 10 – общая схема процессора RISC-V

Flow Summary

<<Filter>>

Flow Status	Successful - Wed Mar 05 20:31:36 2025
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	scheme
Top-level Entity Name	RISC_V_PROCESSOR
Family	Cyclone 10 LP
Total logic elements	7,203 / 15,408 (47 %)
Total registers	2562
Total pins	84 / 163 (52 %)
Total virtual pins	0
Total memory bits	393,216 / 516,096 (76 %)
Embedded Multiplier 9-bit elements	24 / 112 (21 %)
Total PLLs	0 / 4 (0 %)
Device	10CL016YU256C6G
Timing Models	Final

Рисунок 11 – список затраченных ресурсов

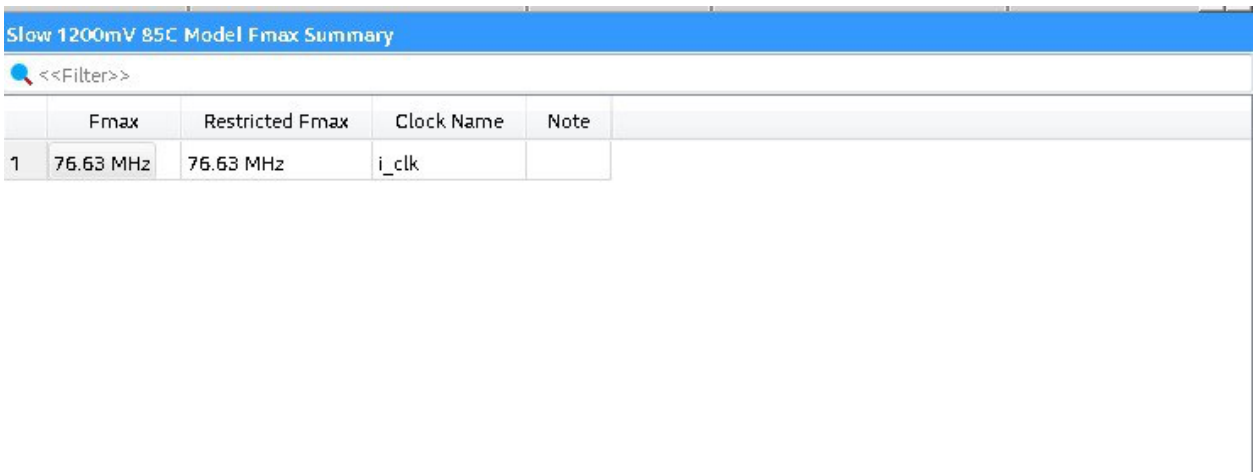

Рисунок 12 – максимальная частота при температуре ядра 85 С и подаче на ядро 1,2V

Рисунок 13 – максимальная частота при температуре 0С и подаче на ядро 1,2V

При 0°C схема работает быстрее из-за увеличенной подвижности носителей и меньших утечек, позволяя достигать 86 МГц**.** При 85°C из-за ухудшения характеристик полупроводников скорость работы снижается, и максимальная частота ограничивается 76 МГц.

5. Заключение

В рамках данной курсовой работы был разработан и смоделирован декодер команд для упрощенной архитектуры RISC-V процессора. Работа охватила этапы проектирования на языке VHDL, моделирования и тестирования разработанного устройства. Декодер успешно идентифицирует три типа инструкций – R-type, I-type и S-type – на основе анализа опкодов, извлекая необходимые операнды и генерируя сигналы управления для последующих этапов обработки данных.

Декодер реализует четырехступенчатый конвейер для передачи информации о регистре-приемнике и необходимости записи в блок LSU (Load/Store Unit). Эта архитектура позволяет оптимизировать обработку инструкций и повысить производительность системы. В процессе проектирования был использован язык VHDL, что позволило описать логику декодера в виде четкого и формализованного кода, пригодного для последующей реализации в виде аппаратной схемы.

Для проверки работоспособности декодера было проведено тестирование. Были разработаны тестовые наборы, охватывающие различные комбинации входных данных и типов инструкций. Результаты моделирования показали полное соответствие выходных сигналов декодера ожидаемым значениям для всех протестированных случаев. Полученные данные подтвердили корректность функционирования декодера и успешную реализацию алгоритма декодирования инструкций.

В ходе выполнения работы были приобретены практические навыки проектирования цифровых устройств на языке VHDL, а также углублены знания в области архитектуры RISC-V процессоров и принципов работы декодера команд.