федеральное государственное автономное образовательное учреждение
высшего образования

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

ОТЧЕТ

по лабораторной работе №4

по дисциплине «**Функциональная схемотехника**»

Вариант BUFFER_LRU

Авторы: Кулаков Н. В.

Факультет: ПИиКТ

Группа: P33312

Преподаватель: Васильев С.Е.

Санкт-Петербург 2023

Оглавление

1. Задание

В лабораторной работе вам предлагается разобраться во внутреннем устройстве простейшего процессорного ядра архитектуры RISC-V. Результатом изучения микроархитектуры процессорного ядра и системы команд RISC-V станут ваши функциональные и нефункциональные модификации ядра.

1.1. Основное

1. Модифицировать процессорное ядро, в соответствии с вашим вариантом;
2. Подготовить тестовое окружение системного уровня и убедиться в корректности вашей реализации путём запуска симуляционных тестов.

1.2. Дополнительное

1. Выполнить основное задание;
2. Разработать интерфейсы для UART-приёмника и подключить его к проекту;
3. Реализовать возможность заполнения памяти команд данными с UART-приёмника;
4. Реализовать возможность вывода значений регистров на семисегментные инди-каторы и новую команду для такого вывода;
5. Добиться корректной работы вашего проекта на FPGA Nexys4DDR.

2. Выполнение основного задания

2.1. Микроархитектурная схема модифицированного процессорного ядра

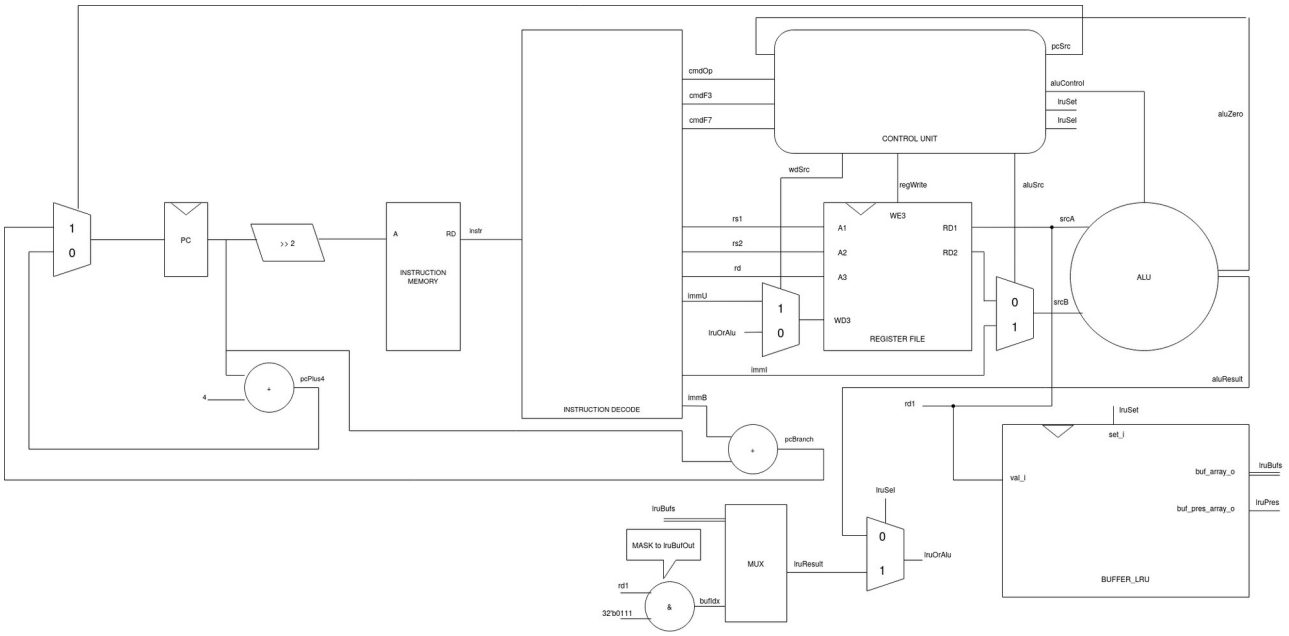


Рисунок 1: Схема процессорного ядра с LRU

BUFFER_LRU — добавленный модуль

Связанные входные порты:

- rd1 — значение регистра, требуемое записать в буфер
- lruSet — на вход установлено значение, которое необходимо записать

Связанные выходные порты:

- lruBuffers — внутренние состояния всех буферов
- lruPres (не использован) — биты присутствия элементов

Прочие добавленные сигналы:

- bufIdx — использован, чтобы выбрать тот элемент, который хотим прочитать из BUFFER_LRU. Сформирован на основании обрезанного значения младших бит rd1
- lruResult — выбранный буфер BUFFER_LRU. Получен посредством выбора с мультиплексора соотвественной ячейки из lruBufs и bufIdx.
- lruOrAlu — выбранное значение между lruResult и aluResult. Затем данный результат поступает на вход мультиплексора, который выбирает между чтением Imm и lruOrAlu, таким образом уже использованная логика не переписана, а дополнена.
- lruSel — выбор мультиплексора между lruResult и aluResult. Когда 0 то выбирается aluResult.

2.2. Описание добавленных форматов и инструкций

2.2.1. Формат PO

- [31-19] — any
- [18-15] — rs1
- [14-12] — funct3
- [11-7] — any
- [6-0] — opcode

2.2.2. Формат PI

- [31-19] — any
- [18-15] — rs1
- [14-12] — funct3
- [11-7] — rd

- [6-0] — opcode

2.2.3. push

- Формат PO
- rs1 — номер регистра, с которого считываются данные
- lru.push(rs1[15:0])

2.2.4. pop

- Формат PI
- rs1 — номер регистра, с которой получаем номер считываемой ячейки буфера lru
- rd — номер регистра, куда пишем новое значение (в частном случае выполняет условие задачи на то, чтобы операции чтения и записи производились с одним регистром)
- rd = lru.pop(rs1[2:0]) ; так как всего ячеек 8

2.3. Описание алгоритма функционирования новой части микропроцессорного ядра

2.3.1. push

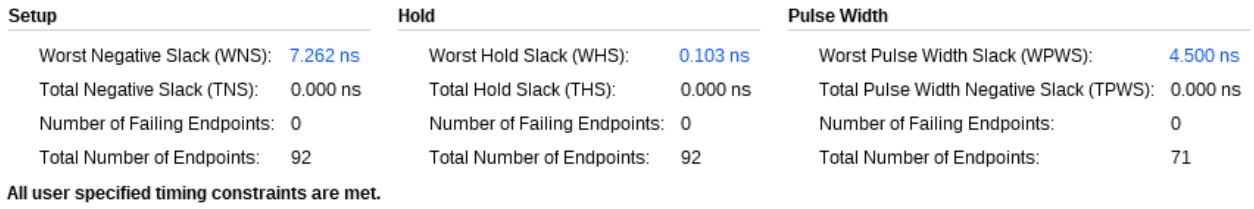
В CONTROL_UNIT на выходе устанавливается высокий уровень (lruSet = 1) и осуществляется чтение значения регистра из регистрового файла (regWrite = 0). На вход BUFFER_LRU подается (lruSet = 1, rs1 = RF[rs1]) и осуществляется запись в BUFFER_LRU.

2.3.2. pop

В CONTROL_UNIT на выходе устанавливается низкий уровень (lruSet = 0), в целом для всех команд кроме push он в низкий, значит запись не происходит. На

выходе BUFFER_LRU всегда находятся значения буферов lruBufs. Из регистра rs1 считывается число, затем это число обрезается по младшим байтам там, чтобы получить индекс буфера lruBufs, результат — сигнал bufIdx. Затем на мультиплексоре с помощью bufIdx и lruBufs выбирается соответствующая ячейка буфера lruResult. aluSel равняется 1, значит выбирается lruResult, теперь это значение на линии aluOrLru, в противном случае при aluSel = 0 там значение с ALU. wdSrc = 1, осуществляется запись в регистровый файл в регистр rd, в частном случае равный rs1.

2.4. Результаты по временным параметрам



Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	7.262 ns	Worst Hold Slack (WHS):	0.103 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	92	Total Number of Endpoints:	92	Total Number of Endpoints:	71

All user specified timing constraints are met.

Рисунок 2: Отчет по временным параметрам

Основной причиной увеличения таймингов является последовательносное включение buffer_lru, мультиплексора для выбора линии буфера на вывод и пары мультиплексоров для выбора значений.

3. Выполнение дополнительного задания

TODO

4. Выводы по работе

При выполнении основной части лабораторной работы для push и pop были созданы свои типы команд. На самом деле данная процедура является излишней и может налагать дополнительные сложности для независимого расширения, поскольку opcode, funct3, funct7 выбирались только так, чтобы они не пересекались с командами riscv32i.

На основании полученных данных для данной частоты нарушений таймингов не было.

Добавление BUFFER_LRU является довольно прямолинейным, поэтому даже нечего сказать по поводу других реализаций, максимум можно определить способы подключения компонентов друг за другом, так например подключить BUFFER_LRU после ALU, а не параллельно ему, однако в таком случае было бы больше последовательностной логики для одного такта инструкции, и предоставило возможность выполнения параллельно получения результата через ALU и запись уже измененного результата. Правда это бы потребовало изменить формат команды.

Также можно было не выделять отдельные сигналы для управления BUFFER_LRU, а использовать вариации уже существующих. Например, дополнить AluControl управлением также для BUFFER_LRU.

Чтобы увеличить тактовую частоту процессора, можно было бы сделать процессор многотактовым и, например, разбить его работу на несколько простейших этапой: Fetching, Execution, Writing. Так, например, это позволило нам сделать память большей емкости за счет увеличения максимально допустимого времени доступа или увеличить тактовую частоту процессора, однако это бы потребовало полностью переписать реализацию.

В ходе выполнения данной ЛР возникли трудности с настройкой окружения, в частности, с установкой компилятора под riscv32 для ОС Gentoo, а также правильной установкой флагов, поскольку его gcc собирает код динамически линкуемым, что в результате дает неработоспособный код в связи со вставкой дополнительныъ инструкций для динамической линковки. Чтобы это исправить, потребовалось догадаться установить флаг -static.

Пока это осознавалось, то простейшим образом разобрался с тем, что такое .ld файлы, их форматом и тем, как его можно создавать и изменять под свои нужды.

На момент написания отчета автор не справился с выполнением доп-задания из-за недостатка времени.