МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА - Российский технологический университет»

РТУ МИРЭА

Институт Информационных Технологий
Кафедра Вычислительной Техники (ВТ)

ОТЧЁТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 3

«Построение процессорного ядра последовательного типа»

по дисциплине

«Схемотехника устройств компьютерных систем»

Выполнил студент группы ИВБО-11-23 — Туктаров Т.А.

Принял преподаватель кафедры ВТ — Дуксин Н.А.

Практическая работа выполнена «__»_____2025 г.

«Зачтено» «__»_____2025 г.

Москва 2025

АННОТАЦИЯ

Данная работа включает в себя 1 рисунок, 4 листингов. Количество страниц в работе — 13

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современных цифровых вычислительных системах ключевым элементом является центральный процессор, внутри которого находится так называемое процессорное ядро — аппаратно-логическая единица, отвечающая за выполнение команд и управление вычислительным процессом. Ядро организовано таким образом, что оно последовательно, шаг за шагом, обрабатывает инструкции: извлекает их из памяти, декодирует, выполняет и сохраняет результат. Такая архитектура называется последовательного типа, поскольку в любой момент времени активна лишь одна команда или поток команд, и выполнение происходит строго одно за другим, без одновременного выполнения нескольких потоков или инструкций.

В схемотехническом плане ядро последовательного типа реализуется как комбинация следующих блоков: модуль выборки инструкций, декодер, устройство управления, арифметико-логическое устройство (АЛУ), регистровый файл и схема взаимодействия с памятью данных и программ. Эти блоки связаны по принципу «конвейерной цепочки» или даже без конвейера: инструкция проходит все фазы последовательно, прежде чем начать следующая. Такое упрощённое ядро имеет меньшую сложность логики и управления по сравнению с многоядерными или многопоточными архитектурами, но его производительность при выполнении многозадачных или параллельных алгоритмов оказывается ограниченной.

Преимущества последовательного ядра: простота схемы, меньшая площадь кристалла, сниженное энергопотребление, облегчённая реализация и проверка. Недостатки: ограниченная пропускная способность, невозможность эффективно использовать параллелизм задач, более низкая производительность при нагрузке, требующей одновременного выполнения множества команд или потоков.

1 ХОД РАБОТЫ

1.1 Постановка задачи

Целью данной работы является разработка и реализация процессорного ядра последовательного типа, предназначенного для выполнения задачи сортировки массива данных.

1.2 Описание логики работы

Для более точного описания проекта создаются схемы и таблицы, которые помогут понять, как именно работает наш процессор.

1.3 Создание эмулятора

Далее опишем эмулятор процессорного ядра для нашей задачи. Код эмулятора доступен по следующей ссылке:

https://github.com/Idontlikeni/Mirea_fifth/blob/main/Python_misc/emulator_ksu.py

Результат можно увидеть в Листинге 1.2.

Листинг 1.1 — Результат работы эмулятора

```
ФИНАЛЬНОЕ СОСТОЯНИЕ:
Текущее состояние процессора:
PC = 31
RF = [0, 1, 3, 3, 0, 0, 1, 5, 3, 0]
mem[0:10] = [3, 5, 7, 3, 0, 0, 0, 0, 0, 0]
Следующая команда: КОНЕЦ ПРОГРАММЫ
Всего выполнено команд: 69
```

1.4 Реализация кода на verilog

Для начала опишем процессорного ядра последовательного типа. Реализация представлена в Листинге 1.3.

Листинг 1.2 — Модуль cpu.v

```
function automatic real real_sin;
`timescale 1ns / 1ps
module cpu(
    input clk, reset,
    output pc
    );
localparam LITERAL_SIZE = 10;
localparam COP_SIZE = 4;

localparam CMD_MEM_SIZE = 32;
localparam CMD_ADDR_SIZE = $clog2(CMD_MEM_SIZE);
localparam CMD_SIZE = 19;

localparam MEM_SIZE = 32;
localparam MEM_ADDR_SIZE = $clog2(MEM_SIZE);
localparam MEM_DATA_SIZE = LITERAL_SIZE;

localparam RF_SIZE = 16;
localparam RF_ADDR_SIZE = $clog2(RF_SIZE);
localparam RF_DATA_SIZE = LITERAL_SIZE;

localparam NOP = 0, LTM = 1, MTR = 2, RTR = 3, JL = 4, SUB = 5, SUM = 6, MTRK =
7, RTM = 8, JMP = 9;

reg [CMD_SIZE       -1:0] cmd_mem [0:CMD_MEM_SIZE-1];
reg [MEM_DATA_SIZE  -1:0] mem [0:MEM_SIZE-1];
reg [RF_DATA_SIZE   -1:0] RF [0:RF_SIZE-1];
reg [CMD_ADDR_SIZE  -1:0] pc;
reg [CMD_SIZE       -1:0] cmd_reg;
reg [LITERAL_SIZE   -1:0] opA, opB;
reg [2*LITERAL_SIZE -1:0] res;
`define hi 2*LITERAL_SIZE-1 -: LITERAL_SIZE
`define lo LITERAL_SIZE - 1: 0
reg [2:0] stage_counter;
integer i;
initial
begin
    for(i = 0; i < MEM_SIZE; i = i + 1)
        mem[i] = 0;
    for(i = 0; i < RF_SIZE; i = i + 1)
        RF[i] = 0;
    RF[1] = 1;

    $readmemb("C:/MyProfile/MienPractik/prac-3/prac-
3.srcs/sources_1/new/cmd_mem.mem", cmd_mem);
end

wire [COP_SIZE-1:0] cop                 = cmd_reg[CMD_SIZE-1 -: COP_SIZE];
wire [RF_ADDR_SIZE-1:0] addr_m_1        = cmd_reg[CMD_SIZE-1 - COP_SIZE -:
MEM_ADDR_SIZE];
```

Продолжение Листинга 1.2

```
wire [RF_ADDR_SIZE-1:0] addr_r_1        = cmd_reg[CMD_SIZE-1 - COP_SIZE -:
RF_ADDR_SIZE];

wire [RF_ADDR_SIZE-1:0] addr_r_1_MTR    = cmd_reg[CMD_SIZE-1 - COP_SIZE -
MEM_ADDR_SIZE -: RF_ADDR_SIZE];
wire [RF_ADDR_SIZE-1:0] addr_r_2        = cmd_reg[CMD_SIZE-1 - COP_SIZE -
RF_ADDR_SIZE -: RF_ADDR_SIZE];
wire [RF_ADDR_SIZE-1:0] addr_r_3        = cmd_reg[CMD_SIZE-1 - COP_SIZE -
2*RF_ADDR_SIZE -: RF_ADDR_SIZE];
wire [LITERAL_SIZE-1:0] literal         = cmd_reg[LITERAL_SIZE-1:0];
wire [CMD_ADDR_SIZE-1:0] addr_to_jmp    = cmd_reg[CMD_ADDR_SIZE-1:0];

always@(posedge clk)
    if(reset || stage_counter == 4)
        stage_counter <= 0;
    else
        stage_counter <= stage_counter + 1;

always@(posedge clk)
    if(reset)
        cmd_reg <= {(CMD_SIZE){1'b0}};
    else
        if(stage_counter == 0)
            cmd_reg <= cmd_mem[pc];

always@(posedge clk)
    if(reset)
        opA <= {(LITERAL_SIZE){1'b0}};
    else
        if(stage_counter == 1)
            case (cop)
                LTM: opA <= addr_m_1;
                MTR: opA <= mem[addr_m_1];
                RTR, RTM: opA <= RF[addr_r_2];
                JL, SUM, SUB, MTRK: opA <= RF[addr_r_1];
            endcase

always@(posedge clk)
    if(reset)
        opB <= {(LITERAL_SIZE){1'b0}};
    else
        if(stage_counter == 2)
            case (cop)
                LTM: opB <= literal;
                JL, SUM, SUB: opB <= RF[addr_r_2];
                MTRK: opB <= mem[opA];
                RTM: opB <= RF[addr_r_1];
            endcase

always@(posedge clk)
    if(reset)
        res <= {(2*LITERAL_SIZE){1'b0}};
    else
        if(stage_counter == 3)
            case (cop)
                LTM, RTM: res <= {opA, opB};
                MTR, RTR: res <= opA;
                MTRK: res <= opB;
                JL: res <= opA < opB;
                SUB: res <= opA - opB;
```

Продолжение Листинга 1.2

```
                SUM: res <= opA + opB;
            endcase

always@(posedge clk)
 if(reset)
        pc <= {(CMD_ADDR_SIZE){1'b0}};
    else
        if(stage_counter == 4)
            case (cop)
                JL: if(res == 1) pc <= pc + 1; else pc <= addr_to_jmp;
                JMP: pc <= addr_to_jmp;
                default: pc <= pc + 1;
            endcase

always@(posedge clk)
    if(stage_counter == 4)
        case (cop)
            MTR: RF[addr_r_1_MTR] <= res;
            RTR: RF[addr_r_1] <= res;
            SUB, SUM: RF[addr_r_3] <= res;
            MTRK: RF[addr_r_2] <= res;
        endcase

always@(posedge clk)
    if(stage_counter == 4)
        case (cop)
            LTM, RTM: mem[res[`hi]] <= res[`lo];
        endcase

endmodule
```

Далее заполним файл памяти команд. Реализация представлена в Листинге 1.4.

Листинг 1.3 — Модуль cmd_mem.mem

```
0001000000000000101
0001000010000000111
0001000100000000011
0001000110000000011
0010000110010000000
0011001100000000000
0100001100100010101
0011010000000000000
0101001000110101000
0101010100010101000
0100010001010010011
0110010000010110000
0111010001110000000
0111011010000000000
0100100001110010001
1000011101100000000
1000100001000000000
0110010000010100000
1001000000000001000
0110001100010011000
1001000000000000110
0000000000000000000
0000000000000000000
0000000000000000000
0000000000000000000
0000000000000000000
0000000000000000000
0000000000000000000
0000000000000000000
0000000000000000000
0000000000000000000
0000000000000000000
```

1.5 Создание тестового модуля и его верификация

Опишем тестовый модуль в модуле testbench. Реализация представлена в Листинге 1.5.

Листинг 1.4 — Модуль testbench.v

```
`timescale 1ns / 1ps
module testbench;
reg reset;
reg clk = 0;
always #5 clk = ~clk;
wire pc;
cpu unit(
.clk(clk),
.reset(reset),
.pc(pc)
);
initial begin
reset = 1;
#100;
reset = 0;
#10000;
end
endmodule
```

Результат тестирования представлены в Рисунке 1.7. Корректность работы можно увидеть по изменениям значений в mem. Как видно, в конце выполнения процессор пришёл к тому же результату, что и эмулятор.

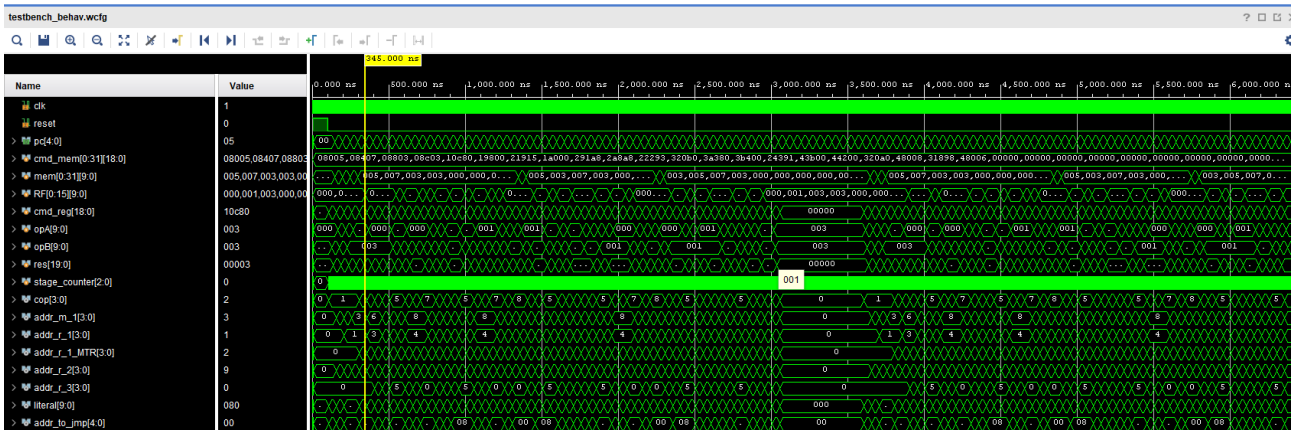

Рисунок 1.7 — Результат работы процессорного ядра

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы было спроектировано и реализовано процессорное ядро последовательного типа, предназначенное для выполнения операции сортировки массива данных. В процессе выполнения были рассмотрены структура и взаимодействие основных блоков процессора: арифметико-логического устройства, регистра команд, счётчика команд, регистрового файла, памяти данных и памяти команд. Разработанное ядро успешно прошло функциональную верификацию. Корректность его работы подтверждается совпадением результатов моделирования на языке Verilog с результатами, полученными при выполнении программы на эмуляторе. Это свидетельствует о правильности реализованной архитектуры и корректной работе всех стадий процессора.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Дуксин, Н. А. Архитектура вычислительных машин и систем. Основы построения вычислительной техники: Практикум : учебное пособие / Н. А. Дуксин, Д. В. Люлява, И. Е. Тарасов. — Москва : РТУ МИРЭА, 2023. — 185 с.

2. Смирнов С.С. Информатика [Электронный ресурс]: Методические указания по выполнению практических и лабораторных работ / С.С. Смирнов — М., МИРЭА — Российский технологический университет, 2018. — 1 электрон. опт. диск (CD-ROM)

3. Тарасов И.Е. ПЛИС Xilinx. Языки описания аппаратуры VHDL и Verilog, САПР, приемы проектирования. — М.: Горячая линия — Телеком, 2021. — 538 с.: ил.

4. Жемчужникова Т.Н. Конспект лекций по дисциплине «Архитектура вычислительных машин и систем»