Санкт-Петербургский Национальный Исследовательский Университет

Информационных Технологий, Механики и Оптики

Факультет ПИиКТ

Лабораторная работа 4

по дисциплине

«Функциональная схемотехника»

Выполнил: студент группы Р33111

Киселёв Сергей Владимирович

г. Санкт-Петербург

2021 г.

**Задание**

1. Разработайте и опишите на Verlog HDL комбинационную схему, вычисляющую значение функции **.**
2. Написать релевантное тестовое покрытие для модуля вычисления корня.
3. Интегрировать модуль как команду в процессор MIPS (<https://github.com/MIPSfpga/schoolMIPS/tree/03_pipeline>).
4. Составьте отчет по результатам выполнения работы.

**Git**

<https://github.com/kamikadze328/Circuit-Design/lab4>

**Решение**

**sqrt.v**

`timescale 1ns / 1ps

module sqrt(

input [31:0] x,

output [31:0] y

);

wire [31:0] num\_1, res\_1, check\_1;

assign num\_1 = x;

assign res\_1 = 0;

assign check\_1 = {32{ num\_1 >= (1 << 30 + res\_1) }};

wire [31:0] num\_2, res\_2, check\_2;

assign num\_2 = num\_1 - (check\_1 & ((1 << 30) + res\_1));

assign res\_2 = (res\_1 >> 1) + (check\_1 & (1 << 30));

assign check\_2 = {32{ num\_2 >= ((1 << 28) + res\_2) }};

wire [31:0] num\_3, res\_3, check\_3;

assign num\_3 = num\_2 - (check\_2 & ((1 << 28) + res\_2));

assign res\_3 = (res\_2 >> 1) + (check\_2 & (1 << 28));

assign check\_3 = {32{ num\_3 >= ((1 << 26) + res\_3) }};

wire [31:0] num\_4, res\_4, check\_4;

assign num\_4 = num\_3 - (check\_3 & ((1 << 26) + res\_3));

assign res\_4 = (res\_3 >> 1) + (check\_3 & (1 << 26));

assign check\_4 = {32{ num\_4 >= ((1 << 24) + res\_4) }};

wire [31:0] num\_5, res\_5, check\_5;

assign num\_5 = num\_4 - (check\_4 & ((1 << 24) + res\_4));

assign res\_5 = (res\_4 >> 1) + (check\_4 & (1 << 24));

assign check\_5 = {32{ num\_5 >= ((1 << 22) + res\_5) }};

wire [31:0] num\_6, res\_6, check\_6;

assign num\_6 = num\_5 - (check\_5 & ((1 << 22) + res\_5));

assign res\_6 = (res\_5 >> 1) + (check\_5 & (1 << 22));

assign check\_6 = {32{ num\_6 >= ((1 << 20) + res\_6) }};

wire [31:0] num\_7, res\_7, check\_7;

assign num\_7 = num\_6 - (check\_6 & ((1 << 20) + res\_6));

assign res\_7 = (res\_6 >> 1) + (check\_6 & (1 << 20));

assign check\_7 = {32{ num\_7 >= ((1 << 18) + res\_7) }};

wire [31:0] num\_8, res\_8, check\_8;

assign num\_8 = num\_7 - (check\_7 & ((1 << 18) + res\_7));

assign res\_8 = (res\_7 >> 1) + (check\_7 & (1 << 18));

assign check\_8 = {32{ num\_8 >= ((1 << 16) + res\_8) }};

wire [31:0] num\_9, res\_9, check\_9;

assign num\_9 = num\_8 - (check\_8 & ((1 << 16) + res\_8));

assign res\_9 = (res\_8 >> 1) + (check\_8 & (1 << 16));

assign check\_9 = {32{ num\_9 >= ((1 << 14) + res\_9) }};

wire [31:0] num\_10, res\_10, check\_10;

assign num\_10 = num\_9 - (check\_9 & ((1 << 14) + res\_9));

assign res\_10 = (res\_9 >> 1) + (check\_9 & (1 << 14));

assign check\_10 = {32{ num\_10 >= ((1 << 12) + res\_10) }};

wire [31:0] num\_11, res\_11, check\_11;

assign num\_11 = num\_10 - (check\_10 & ((1 << 12) + res\_10));

assign res\_11 = (res\_10 >> 1) + (check\_10 & (1 << 12));

assign check\_11 = {32{ num\_11 >= ((1 << 10) + res\_11) }};

wire [31:0] num\_12, res\_12, check\_12;

assign num\_12 = num\_11 - (check\_11 & ((1 << 10) + res\_11));

assign res\_12 = (res\_11 >> 1) + (check\_11 & (1 << 10));

assign check\_12 = {32{ num\_12 >= ((1 << 8) + res\_12) }};

wire [31:0] num\_13, res\_13, check\_13;

assign num\_13 = num\_12 - (check\_12 & ((1 << 8) + res\_12));

assign res\_13 = (res\_12 >> 1) + (check\_12 & (1 << 8));

assign check\_13 = {32{ num\_13 >= ((1 << 6) + res\_13) }};

wire [31:0] num\_14, res\_14, check\_14;

assign num\_14 = num\_13 - (check\_13 & ((1 << 6) + res\_13));

assign res\_14 = (res\_13 >> 1) + (check\_13 & (1 << 6));

assign check\_14 = {32{ num\_14 >= ((1 << 4) + res\_14) }};

wire [31:0] num\_15, res\_15, check\_15;

assign num\_15 = num\_14 - (check\_14 & ((1 << 4) + res\_14));

assign res\_15 = (res\_14 >> 1) + (check\_14 & (1 << 4));

assign check\_15 = {32{ num\_15 >= ((1 << 2) + res\_15) }};

wire [31:0] num\_16, res\_16, check\_16;

assign num\_16 = num\_15 - (check\_15 & ((1 << 2) + res\_15));

assign res\_16 = (res\_15 >> 1) + (check\_15 & (1 << 2));

assign check\_16 = {32{ num\_16 >= res\_16 }};

wire [31:0] num\_final, res\_final;

assign num\_final = num\_16 - (check\_16 & (res\_16 + 1));

assign res\_final = (res\_16 >> 1) + (check\_16 & 1);

assign y = res\_final;

endmodule

**test\_sqrt.v**

`timescale 1ns / 1ps

module test\_sqrt();

reg [31:0] x;

wire [31:0] result;

sqrt sqrt\_module (

.x(x),

.y(result)

);

initial begin

x=32'hffffffff; #3 //sqrt(4 294 967 295)

$display("%h should be ffff", result);

x=32'hfffe0001; #3 //sqrt(4 294 836 225)

$display("%h should be ffff", result);

x=32'hfffe0000; #3 //sqrt(4 294 836 224)

$display("%h should be fffe", result);

x=32'hfffeffff; #3 //sqrt(4 294 836 224)

$display("%h should be fffe", result);

x=32'hfffa0009; #3 //sqrt(4 294 574 089)

$display("%h should be fffd", result);

x='b0; #3 //sqrt(0)

$display("%h should be 0", result);

x=32'h00000001; #3 //sqrt(1)

$display("%h should be 1", result);

x=32'h00000004; #3 //sqrt(4)

$display("%h should be 2", result);

end

endmodule

**sm\_cpu.vh**

`define ALU\_ADD 3'b000

`define ALU\_OR 3'b001

`define ALU\_LUI 3'b010

`define ALU\_SRL 3'b011

`define ALU\_SLTU 3'b100

`define ALU\_SUBU 3'b101

**`define ALU\_MYSQRT 3'b110**

//instruction operation code

`define C\_SPEC 6'b000000 // Special instructions (depends on function field)

`define C\_ADDIU 6'b001001 // I-type, Integer Add Immediate Unsigned

// Rd = Rs + Immed

`define C\_BEQ 6'b000100 // I-type, Branch On Equal

// if (Rs == Rt) PC += (int)offset

`define C\_LUI 6'b001111 // I-type, Load Upper Immediate

// Rt = Immed << 16

`define C\_BNE 6'b000101 // I-type, Branch on Not Equal

// if (Rs != Rt) PC += (int)offset

`define C\_LW 6'b100011 // I-type, Load Word

// Rt = memory[Rs + Immed]

`define C\_SW 6'b101011 //I-type, Store Word

// memory[Rs + Immed] = Rt

**`define C\_MYSQRT 6'b000001 //I-type, sqrt(Rs) = Rt**

//instruction function field

`define F\_ADDU 6'b100001 // R-type, Integer Add Unsigned

// Rd = Rs + Rt

`define F\_OR 6'b100101 // R-type, Logical OR

// Rd = Rs | Rt

`define F\_SRL 6'b000010 // R-type, Shift Right Logical

// Rd = Rs∅ >> shift

`define F\_SLTU 6'b101011 // R-type, Set on Less Than Unsigned

// Rd = (Rs∅ < Rt∅) ? 1 : 0

`define F\_SUBU 6'b100011 // R-type, Unsigned Subtract

// Rd = Rs – Rt

`define F\_ANY 6'b??????

**sm\_cpu.v**

Вставленны только изменённые участки кода

module sm\_control

(

input [5:0] cmdOper,

input [5:0] cmdFunk,

input aluZero,

output pcSrc,

output reg regDst,

output reg regWrite,

output reg aluSrc,

output reg [2:0] aluControl,

output reg memWrite,

output reg memToReg,

output reg branch

);

reg condZero;

assign pcSrc = branch & (aluZero == condZero);

always @ (\*) begin

branch = 1'b0;

condZero = 1'b0;

regDst = 1'b0;

regWrite = 1'b0;

aluSrc = 1'b0;

aluControl = `ALU\_ADD;

memWrite = 1'b0;

memToReg = 1'b0;

casez( {cmdOper,cmdFunk} )

default : ;

{ `C\_SPEC, `F\_ADDU } : begin regDst = 1'b1; regWrite = 1'b1; aluControl = `ALU\_ADD; end

{ `C\_SPEC, `F\_OR } : begin regDst = 1'b1; regWrite = 1'b1; aluControl = `ALU\_OR; end

{ `C\_SPEC, `F\_SRL } : begin regDst = 1'b1; regWrite = 1'b1; aluControl = `ALU\_SRL; end

{ `C\_SPEC, `F\_SLTU } : begin regDst = 1'b1; regWrite = 1'b1; aluControl = `ALU\_SLTU; end

{ `C\_SPEC, `F\_SUBU } : begin regDst = 1'b1; regWrite = 1'b1; aluControl = `ALU\_SUBU; end

**{ `C\_MYSQRT,`F\_ANY } : begin regWrite = 1'b1; aluSrc = 1'b1; aluControl = `ALU\_MYSQRT; end**

{ `C\_ADDIU, `F\_ANY } : begin regWrite = 1'b1; aluSrc = 1'b1; aluControl = `ALU\_ADD; end

{ `C\_LUI, `F\_ANY } : begin regWrite = 1'b1; aluSrc = 1'b1; aluControl = `ALU\_LUI; end

{ `C\_LW, `F\_ANY } : begin regWrite = 1'b1; aluSrc = 1'b1; aluControl = `ALU\_ADD; memToReg = 1'b1; end

{ `C\_SW, `F\_ANY } : begin memWrite = 1'b1; aluSrc = 1'b1; aluControl = `ALU\_ADD; end

{ `C\_BEQ, `F\_ANY } : begin branch = 1'b1; condZero = 1'b1; aluControl = `ALU\_SUBU; end

{ `C\_BNE, `F\_ANY } : begin branch = 1'b1; aluControl = `ALU\_SUBU; end

endcase

end

endmodule

…..

module sm\_alu

(

input [31:0] srcA,

input [31:0] srcB,

input [ 2:0] oper,

input [ 4:0] shift,

output zero,

output reg [31:0] result

);

**wire [31:0] output\_sqrt;**

**sqrt sqrt(srcA, output\_sqrt);**

always @ (\*) begin

case (oper)

default : result = srcA + srcB;

`ALU\_ADD : result = srcA + srcB;

`ALU\_OR : result = srcA | srcB;

`ALU\_LUI : result = (srcB << 16);

`ALU\_SRL : result = srcB >> shift;

`ALU\_SLTU : result = (srcA < srcB) ? 1 : 0;

`ALU\_SUBU : result = srcA - srcB;

**`ALU\_MYSQRT : result = output\_sqrt;**

endcase

end

assign zero = (result == 0);

endmodule

**Выводы**

Я получил опыт работы с реальным микропроцессором и узнал строение и принципы работы конвеерного процессора.