**THỰC HÀNH THIẾT KẾ HỆ THỐNG SỐ VỚI HDL**

**GVHD: TRẦN ĐẠI DƯƠNG**

**Lớp: CE213.L21.MTCL.1**

**Họ và tên sinh viên thực hiện: Phạm Quang Hải**

**Mã số sinh viên: 18520703**

**BÀI THỰC HÀNH SỐ 4:**

**THIẾT KẾ ASIC**

Xác định Input, Output, mô tả chức năng của mạch?

* Input: D3,D2,D1,D0 nhập tuần tự
* Output: Kết quả của phép tính
* Chức năng: Mạch ASIC tính toán biểu thức D3.I3 – D2.I2 + D1.I1 – D0.I0 trong đó I3I2I1I0 tương ứng với bốn số cuối mã số sinh viên (0703).

1. **Sơ đồ khối**
2. Diagram, schematic

   Description automatically generated*Datapath*
3. *Diagram

   Description automatically generatedControl word*
4. *ASIC*

*Diagram

Description automatically generated*

1. **Mô tả chi tiết chức năng, cách mạch hoạt động**

* Biểu thức cần thực hiện: D3.I3 – D2.I2 + D1.I1 – D0.I0
* Nhập tuần tự input D3, D2, D1, D0 từ bàn phím (trong đó input là số 4 bit không dấu), các input này được lưu vào các ô nhớ trong register file. Sau đó đọc các input từ regfile vào ALU để thực hiên các bước tính toán ta sẽ có được kết quả của biểu thức.
* Các bước thực hiện (tương ứng với các trạng thái của máy trạng thái):

S0: Nhập D3, lưu vào ô nhớ thứ 0 (Vì ký số đầu tiên là số 0 nên lưu đè D2 tương đương với \*0)

S1: Nhập D2, lưu vào ô nhớ thứ 0

S2: Nhập D1, lưu vào ô nhớ thứ 1 (Tương tự D3)

S3: Nhập D0, lưu vào ô nhớ thứ 1

S4: Temp1 = (D2 << 3) – D2, lưu temp1 vào ô nhớ thứ 2 (D2 shift trái 3 lần là D2\*8, sau đó trừ 1 lần D2 tương đương với D2\*7 chính là ký số 7 trong mssv)

S5: Temp2 = (D0 << 2) – D0, lưu temp2 vào ô nhớ thứ 3 (Tương tự bước trên)

S6: Out = Temp1 + Temp2 , DONE = 1

* Bảng Control Word:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **sel** | **we** | **waddr1** | **waddr0** | **raddr1\_1** | **raddr1\_0** | **raddr2\_1** | **raddr2\_0** | **op 1** | | **op 0** |
| **s0** | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **s1** | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **s2** | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **s3** | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | |
| **s4** | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | |
| **s5** | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | |
| **s6** | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | |

Ở trạng thái S0, các tín hiệu control trong bảng trên có giá trị lần lượt là: 110000000000, sau khi mã hoá thành dạng hex sẽ có giá trị 0x300

Tương tự với các trạng thái S1: 0x300

S2: 0x340

S3: 0x340

S4: 0x181

S5: 0x1C6

S6: 0x02F

1. **Mô tả Input/Output**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STT** | **Tên port** | **Bit width** | **Loại** | **Mô tả** |
| **1** | Out | 16 | O | Output là kết quả của phép toán |
| **2** | Done | 1 | O | Báo hiệu đã ra kết quả |
| **3** | In | 4 | I |  |
| **4** | RST | 1 | I | Control cần được reset về 0 (tích cực mức thấp) trước khi mạch hoạt động |
| **5** | CLK | 1 | I |  |

1. **Code Verilog**

//Các module sử dụng trong Datapath

module signextend(out,in);

output wire [15:0] out;

input wire [3:0] in;

assign out = (in <4'd7) ? {in[3],in[3],in[3],in[3],in[3],in[3],in[3],in[3],in[3],in[3],in[3],in[3],in[3:0]} : {12'd0, in[3:0]};

endmodule

module mux21\_16bit(out,in1,in2,sel); // được tích hợp cả signextend để mở rộng từ 4->16bit để lưu và thực hiện tính toán

parameter bit\_width = 16;

output wire [bit\_width-1:0] out;

input wire [bit\_width-1:0] in1;

input wire [3:0] in2;

input wire sel;

wire [15:0] outs;

signextend signextend\_inst(

.out(outs),

.in(in2)

);

assign out = sel ? outs : in1;

endmodule

module regfile(out1,out2,we,waddr,raddr1,raddr2,in,clk);//Regfile 16x4

parameter bit\_width = 16;

output wire [bit\_width-1:0] out1, out2;

input wire [bit\_width-1:0] in;

input wire we, clk;

input wire [1:0] waddr, raddr1, raddr2;

reg [bit\_width-1:0] RF [0:3];

assign out1 = RF[raddr1];

assign out2 = RF[raddr2];

always @(posedge clk) begin

if(we)

RF[waddr] <= in;

end

endmodule

module alu\_16bit(out,in1,in2,op);//ALU gồm 3 chức năng: Cộng, <<3, <<2

parameter bit\_width = 16;

parameter nothing = 2'd0;

parameter shiftleft\_3 = 2'd1;

parameter shiftleft\_2 = 2'd2;

parameter add = 2'd3;

output reg [bit\_width-1:0] out;

input wire [bit\_width-1:0] in1,in2;

input wire [1:0] op;

always @(in1,in2,op) begin

case(op)

nothing: begin

out = 16'd0;

end

shiftleft\_3: begin

out = -((in1 << 3) - in1);

end

shiftleft\_2: begin

out = -((in2 << 2) - in2);

end

add: begin

out = in1 + in2;

end

default: out = 16'd0;

endcase

end

endmodule

module register(out,in,clk);

parameter bit\_width = 16;

output reg [bit\_width-1:0] out;

input wire [bit\_width-1:0] in;

input wire clk;

always @(posedge clk) begin

out <= in;

end

endmodule

//Khối Datapath

module datapath(out,in,op,waddr,raddr1,raddr2,sel,we,clk);

output wire [15:0] out;

input wire [3:0] in;

input wire [1:0] op,waddr,raddr1,raddr2;

input wire sel,we,clk;

wire [15:0] outalu,outmux,outA,outB;

mux21\_16bit mux21\_16bit\_inst(

.out(outmux),

.in1(outalu),

.in2(in),

.sel(sel)

);

regfile regfile\_inst(

.out1(outA),

.out2(outB),

.we(we),

.waddr(waddr),

.raddr1(raddr1),

.raddr2(raddr2),

.in(outmux),

.clk(clk)

);

alu\_16bit alu\_16bit\_inst(

.out(outalu),

.in1(outA),

.in2(outB),

.op(op)

);

register register\_inst(

.out(out),

.in(outalu),

.clk(clk)

);

Endmodule

//Các module sử dụng trong khối Control

module countstate(state,clk,rst);

output reg [2:0] state;

input wire clk,rst;

always @(posedge clk or negedge rst) begin

if (!rst) begin

state <= 3'd0;

end

else begin

if (state == 3'b110) begin

state <= 3'b000;

end

else begin

state <= state + 3'd1;

end

end

end

endmodule

module control(done,out,state,clk);

output reg [9:0] out;

output reg done;

input wire clk;

input wire [2:0] state;

parameter s0 = 3'b000;

parameter s1 = 3'b001;

parameter s2 = 3'b010;

parameter s3 = 3'b011;

parameter s4 = 3'b100;

parameter s5 = 3'b101;

parameter s6 = 3'b110;

always @(negedge clk) begin

case (state)

s1: begin

out <= 10'h300;

end

s2: begin

out <= 10'h340;

end

s3: begin

out <= 10'h340;

end

s4: begin

out <= 10'h181;

end

s5: begin

out <= 10'h1C6;

end

s6: begin

out <= 10'h02F;

end

s0: begin

out <= 10'h300;

end

default: out <= 10'h300;

endcase

end

always @(posedge clk) begin

if (state == 3'b110) begin

done <= 1'b1;

end

else begin

done <= 1'b0;

end

end

endmodule

//Khối Control

module control\_finaledition(out,done,rst,clk);

output wire [9:0] out;

output wire done;

input wire rst,clk;

wire [2:0] outcs;

control control\_inst(

.done(done),

.out(out),

.state(outcs),

.clk(clk)

);

countstate countstate(

.state(outcs),

.clk(clk),

.rst(rst)

);

Endmodule

//ASIC

module asic(out,done,in,rst,clk);

output wire [15:0] out;

output wire done;

input wire [3:0] in;

input wire rst,clk;

wire [9:0] control\_signal;// Các tín hiệu tương ứng với opcode ở bảng control word ở phần II.

datapath datapath\_inst(

.out(out),

.in(in),

.op(control\_signal[1:0]),

.waddr(control\_signal[7:6]),

.raddr1(control\_signal[5:4]),

.raddr2(control\_signal[3:2]),

.sel(control\_signal[9]),

.we(control\_signal[8]),

.clk(clk)

);

control\_finaledition control\_inst(

.done(done),

.out(control\_signal),

.rst(rst),

.clk(clk)

);

Endmodule

1. **Testbench và kết quả**

`timescale 1ns/1ps

module testbench();

wire [15:0] out;

wire done;

reg [3:0] in;

reg rst,clk;

reg signed [15:0] expected\_out;

asic asic\_DUT(

.out(out),

.done(done),

.in(in),

.rst(rst),

.clk(clk)

);

initial begin

clk = 1'b0;

rst = 1'b0;

#60 rst=1'b1;

$display("ASIC thuc hien phep toan D3\*I3 - D2\*I2 + D1\*I1 - D0\*I0 voi I3I2I1I0 = 0703");

$display("Nhap in:");

autotest(4,5,6,7,-56);

autotest(11,13,14,15,-136);

autotest(1,2,3,4,-26);

#1000 $finish;

end

task autotest;

input [3:0] D3,D2,D1,D0;

input [15:0] expected\_out\_t;

integer i;

begin

for(i=0;i<=3;i=i+1) begin

@(negedge clk) begin

if (i == 0) begin

in = D3;

end

else if (i == 1) begin

in = D2;

end

else if (i == 2) begin

in = D1;

end

else if (i == 3) begin

in = D0;

expected\_out = expected\_out\_t;

end

end

end

#240;

if (expected\_out == out) begin

$display (" Chuoi so: %0d -- %0d -- %0d -- %0d ,Expected out: %0d => Correct Function!", D3, D2, D1, D0, expected\_out);

end

else begin

$display ("failed!");

end

end

endtask

initial begin

forever #30 clk <= ~clk;

end

endmodule

**Text, letter

Description automatically generated**