پروژه امتیازی DSD(سوال 7)

نام و نام خانوادگی : یاشار پیمایی

شماره دانشجویی : 401100325

در این سوال باید به طراحی یک Vector Processor با register های 512 بیتی به کمک verilog بپردازیم.

پردازنده از 3 بخش زیر تشکیل شده است :

- ALU با قابلیت ضرب و جمع دو vector 512 بیتی .
- Memory با عمق 512 و عرض 32 بیت .
- Register file حاوی 4 رجیستر 512 بیتی .

حالا طراحی و کد Verilog هر module را بررسی میکنیم .

Register file :

این بخش شامل ورودی های :

- clk
- write
- write_address :

دو بیت اول برای آدرس دهی رجیستر در حالت write = 1 و بیت سوم برای حالتی است که عملیات محاسباتی انجام شده و نتیجه در A4,A3 نوشته می شود .

- write_data :

اگر عملیات محاسباتی انجام نشده باشد دیتا در 512 بیت کم ارزش write_data قرار دارد.

و خروجی های :

- A1_out , A2_out , A3_out , A4_out

میباشد .

: Register file کد

```
module RegisterFile(
    input clk , write ,
    input [2:0] write_address ,
    input [1023:0] write_data ,
    output [511:0] A1_out , A2_out , A3_out , A4_out
);
    reg [511 : 0] registers[3:0] ;

    always @(posedge clk) begin
        if (write_address[2])
            {registers[3] , registers[2]} <= write_data ;
        else if (write)
            registers[write_address] <= write_data[511:0] ;
    end

    assign A1_out = registers[0] ;
    assign A2_out = registers[1] ;
    assign A3_out = registers[2] ;
    assign A4_out = registers[3] ;
endmodule
```

ALU :

این بخش شامل ورودی های :

- mul_add :

  0 به معنای ضرب و 1 به معنای جمع .

- in1 , in2

و خروجی 1024 بیتی out که 512 بیت کم ارزش آن L و 512 بیت پر ارزش آن H است .
که هر کدام از H و L را نیز به شکل 16 تکه 32 بیتی میتوان نگاه کرد که به این صورت مقدار دهی شده اند :

$$\{H[i], L[i]\} = op(in1[i], in2[i])$$

که منظور از op ضرب یا جمع است .

کد Alu :

```
module Alu(
    input mul_add, //0 for mul and 1 for add
    input [511:0] in1 , in2 ,
    output [1023:0] out
);

    wire [511:0] H  , L ;
    genvar i;
    generate
        for (i = 0 ; i < 16 ; i = i + 1) begin
            assign {H[32*(i+1)-1:32*i] , L[32*(i+1)-1:32*i]} = mul_add ?
            $signed(in1[32*(i+1)-1:32*i]) + $signed(in2[32*(i+1)-1:32*i]) :
            $signed(in1[32*(i+1)-1:32*i]) * $signed(in2[32*(i+1)-1:32*i]) ;
        end
    endgenerate

    assign out = {H , L} ;

endmodule
```

Memory :

این بخش شامل ورودی های :

- clk
- write
- read
- in_data

و خروجی out_data میباشد .

کد Memory :

```
module memory(
    input clk , write , read ,
    input [8:0] address ,
    input [511:0] in_data ,
    output [511:0] out_data
);

    reg [31:0] MEM[511:0] ;

    assign out_data = read ? {
        MEM[address] ,MEM[address+1] , MEM[address+2] ,MEM[address+3] ,
        MEM[address+4] ,MEM[address+5] , MEM[address+6] ,MEM[address+7] ,
        MEM[address+8] ,MEM[address+9] , MEM[address+10] ,MEM[address+11] ,
        MEM[address+12] ,MEM[address+13] , MEM[address+14] ,MEM[address+15]
    } : 512'bz ;

    always @(posedge clk) begin
        if (write) begin
            {MEM[address] ,MEM[address+1] , MEM[address+2] ,MEM[address+3] ,
            MEM[address+4] ,MEM[address+5] , MEM[address+6] ,MEM[address+7] ,
            MEM[address+8] ,MEM[address+9] , MEM[address+10] ,MEM[address+11] ,
            MEM[address+12] ,MEM[address+13] , MEM[address+14] ,MEM[address+15] }
<= in_data ;
        end
    end

endmodule
```

Cpu :

این بخش شامل ورودی های :

- clk
- instruction :
    1. 000 : load
    2. 001 : store
    3. 010 : add
    4. 011 : multiply
    5. 100 : register initialize

Opcode 100 برای این است که در هنگام تست بتوانیم رجیستر ها را از خارج از محیط memory و Alu مقدار دهی کنیم.

- reg_addr :

حاوی آدرس رجیستری که عملیات load,store,initialize بر روی آن قرار است انجام شود.

- mem_address :

حاوی آدرس ممووری که عملیات load,store بر روی آن قرار است انجام بگیرد.

- Initialize_value:

برای دستور initialize .

و خروجی های آن A1_out , A2_out , A3_out , A4_out اند که برای بررسی در testbench قرار داده شده اند .

حالا ارتباطات اجزا را توضیح میدهیم .

```
RegisterFile regFile (
    .clk(clk) ,
    .write(rf_write) ,
    .write_address(write_address) ,
    .write_data(rf_write_data) ,
    .A1_out(A1_out) ,
    .A2_out(A2_out) ,
    .A3_out(A3_out) ,
    .A4_out(A4_out)
);
```

write در registerFile را به rf_write متصل میکنیم که اگر دستور load باشد rf_write = 1 است .

write_address در registerFile را برابر $\{instruction[1], reg\_addr\}$ قرار میدهیم زیرا فقط در دو opcode مربوط به دستورات add و mul این اندیس instruction 1 میباشد .

rf_write_data در registerFile را اینگونه مقدار دهی میکنیم که اگر دستور add یا mul بود alu_out اساین بشود ، اگر دستور initialize بود initialize_value به آن اساین بشود و اگر load یا store بود mem_out_data .

که این تقسیم بندی را به این شکل انجام میدهیم :

```
assign rf_write_data[511:0] = instruction[2] ? initialize_value :
                              (instruction[1] ? alu_out[511:0] : mem_out_data) ;
```

```
Alu alu (
    .mul_add(alu_mull_add) ,
    .in1(A1_out) ,
    .in2(A2_out) ,
    .out(alu_out)
);
```

alu_mull_add را بر اگر دستور add بود 1 میکنیم و گرنه 0.

```
    memory mem (
        .clk(clk) ,
        .write(mem_write) ,
        .read(mem_read) ,
        .address(mem_address) ,
        .in_data(mem_in_data) ,
        .out_data(mem_out_data)
    );
```

mem_write و mem_read را برحسب load و یا store بودن مقدار دهی میکنیم .

mem_in_data یکی از اعضای registerFile است که بر اساس reg_addr آن را به این شکل مقدار دهی میکنیم :

```
assign mem_in_data = reg_addr[1] ? (reg_addr[0] ? A4_out : A3_out) :
                                   (reg_addr[0] ? A2_out : A1_out) ;
```

کد Cpu :

```
module Cpu(
    input clk ,
    input [2:0] instruction , //add(10) , mul(11) , load(00) , store(01) ,
initialize registers(100)
    input [1:0] reg_addr ,
    input [8:0] mem_address ,
    input [511:0] initialize_value ,
    output [511:0] A1_out , A2_out , A3_out , A4_out
);
    wire [2:0] write_address ;
    wire [1023:0] rf_write_data ;
    wire rf_write ;
    assign write_address = {instruction[1] , reg_addr} ;
    assign rf_write = (instruction[1:0] == 2'b00) ;


    RegisterFile regFile (
        .clk(clk) ,
        .write(rf_write) ,
        .write_address(write_address) ,
        .write_data(rf_write_data) ,
        .A1_out(A1_out) ,
        .A2_out(A2_out) ,
        .A3_out(A3_out) ,
        .A4_out(A4_out)
    );

    wire alu_mull_add ;
    wire [1023:0] alu_out ;
    assign alu_mull_add = (instruction == 3'b010) ;
    Alu alu (
        .mul_add(alu_mull_add) ,
        .in1(A1_out) ,
        .in2(A2_out) ,
        .out(alu_out)
    );

    wire mem_write , mem_read ;
    wire [511:0] mem_in_data , mem_out_data ;
    assign mem_write = (instruction == 3'b001) ; //store
    assign mem_read = (instruction == 3'b000) ; //load
```

```
    assign mem_in_data = reg_addr[1] ? (reg_addr[0] ? A4_out : A3_out) :
                                       (reg_addr[0] ? A2_out : A1_out) ;

    memory mem (
        .clk(clk) ,
        .write(mem_write) ,
        .read(mem_read) ,
        .address(mem_address) ,
        .in_data(mem_in_data) ,
        .out_data(mem_out_data)
    );


    assign rf_write_data[511:0] = instruction[2] ? initialize_value :
                                    (instruction[1] ? alu_out[511:0] :
mem_out_data) ;
    assign rf_write_data[1023:512] = instruction[1] ? alu_out[1023:512] : 512'b0
;



endmodule
```

حالا باید test_bench طراحی کنیم .

به دلیل طولانی بودن کد test_bench کامل در اینجا آورده نشده است و در فایل test_bench.v قرار دارد .

روند کلی test_bench به این صورت است که هر سری ورودی های کنترلی Cpu را تغییر داده و سپس نتیجه را به این شکل چاپ میکنیم :

```
        $display("explanation for current instruction") ;
        for (i = 0 ; i < 16 ; i = i + 1)begin
            $display("A1[%d] = %d ," ,i ,  A1[32*(i+1)-1-:32]) ;
        end
        for (i = 0 ; i < 16 ; i = i + 1)begin
            $display("A2[%d] = %d ," ,i ,  A2[32*(i+1)-1-:32]) ;
        end
        for (i = 0 ; i < 16 ; i = i + 1)begin
            $display("A4A3[%d] = %d ," ,i ,  $signed({A4[32*(i+1)-1-:32] ,
A3[32*(i+1)-1-:32]})) ;
```

دستورات انجام شده در test_bench به این صورت است :

1. $initialize\ A1$
2. $initialize\ A2$
3. $mem[10] = A1$
4. $A2 = mem[10]$
5. $multiply$
6. add
7. $mem[26] = A3$
8. $A1 = mem[26]$
9. $A3 = mem[18]$
10. $initialize\ A2\ with\ negative\ numbers$
11. $multiply$
12. add

کد بخشی از test_bench :

```
module TB();

    reg clk ;
    reg [2:0] opcode ;
    reg [1:0] reg_addr ;
    reg [8:0] mem_address ;
    reg [511:0] initialize_value ;

    wire [511:0] A1 , A2 , A3 , A4 ;


    Cpu cpu(
        .clk(clk),
        .instruction(opcode),
        .reg_addr(reg_addr),
        .mem_address(mem_address),
        .initialize_value(initialize_value),
        .A1_out(A1),
        .A2_out(A2),
        .A3_out(A3),
        .A4_out(A4)
    );


    always
        #5 clk = ~clk ;

    integer i;
    initial begin
        clk = 0 ;

        //initializing A1 with initialize_value
        initialize_value = {32'd1048576 , 32'd15 ,32'd14 , 32'd13 ,32'd12 ,
32'd11 ,32'd10 , 32'd9 ,
                            32'd8 , 32'd7 ,32'd6 , 32'd5 ,32'd4 , 32'd3 ,32'd2 ,
32'd1 } ;
        opcode = 3'b100 ; // initialize register
        reg_addr = 2'b00 ;
        mem_address = 9'd10 ;

        #10
        $display("initializing A1 with initialize_value") ;
        for (i = 0 ; i < 16 ; i = i + 1)begin
```

```
            $display("A1[%d] = %d ," ,i ,  A1[32*(i+1)-1-:32]) ;
        end
        for (i = 0 ; i < 16 ; i = i + 1)begin
            $display("A2[%d] = %d ," ,i ,  A2[32*(i+1)-1-:32]) ;
        end
        for (i = 0 ; i < 16 ; i = i + 1)begin
            $display("A4A3[%d] = %d ," ,i ,  $signed({A4[32*(i+1)-1-:32] ,
A3[32*(i+1)-1-:32]})) ;
        end

             .
             .
             .
             .
             .


        //adding (A4A3 = A1 + A2)
        opcode = 3'b010 ;

        #10
        $display("adding (A4A3 = A1 + A2)") ;
        for (i = 0 ; i < 16 ; i = i + 1)begin
            $display("A1[%d] = %d ," ,i ,  $signed(A1[32*(i+1)-1-:32])) ;
        end
        for (i = 0 ; i < 16 ; i = i + 1)begin
            $display("A2[%d] = %d ," ,i ,  $signed(A2[32*(i+1)-1-:32])) ;
        end
        for (i = 0 ; i < 16 ; i = i + 1)begin
            $display("A4A3[%d] = %d ," ,i ,  $signed({A4[32*(i+1)-1-:32] ,
A3[32*(i+1)-1-:32]})) ;
        end



        $finish ;
    end

endmodule
```

و نتیجه اجرای آن در فایل testBenchResult.txt قرار دارد که به دلیل طولانی نشدن document در این جا قرار داده نشده است .