

**میان ترم درس طراحی سیستم های دیجیتال**

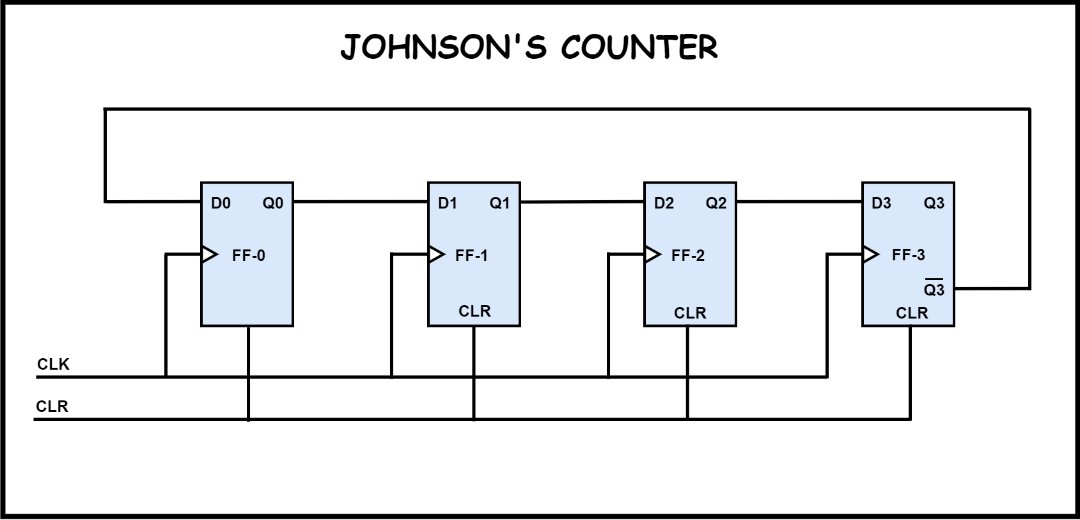
نام و نام خانوادگی: صادق محمدیان

شماره دانشجویی:۴۰۱۱۰۹۴۷۷

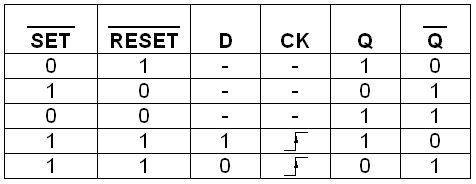
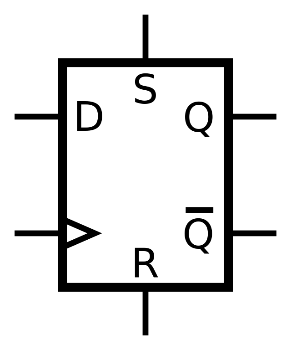
نیم سال دوم تحصیلی ۱۴۰۲-۱۴۰۳

من از بین سوالات داده شده سوال ۶ و ۷ را انتخاب کرده ام.

**سوال ششم:**

در این سوال از ما خواسته شده تا یک شمارنده جانسون N بیتی طراحی کنیم و در طراحی آن از D-flip flop استفاده کنیم. شمای کلی این شمارنده برای N=4 به شکل زیر می باشد:

دنباله ای ک ه این شمارنده تولید می کند به صورت زیر می باشد:

از آنجایی که در این مدار قصد داریم به ازای Nهای مختلف این شمارنده را طراحی کنیم باید از ویژگی پارامتر در کد وریلاگ خود استفاده کنیم. ابتدا اقدام به طراحی یک D-flip flop می کنیم. جدول درستی یک D-flip flop به شکل زیر می باشد:

در این فلیپ فلاپ ورودی های reset و set به صورت آسنکرون هستند.

کد وریلاگ زیر طراحی یک D-flip flop می باشد.

module DFF(input clk , set , reset , d ,

            output reg q);

always @(posedge clk or posedge reset or posedge set)

begin

    if (reset)

    begin

        q <= 1'b0;

    end

    else if(set)begin

        q <= 1'b1;

    end

    else begin

        q <= d;

    end

end

endmodule

در این طراحی سیگنال های set و reset را active high در نظر میگیریم یعنی در صورتی که یک باشند عمل مربوطه انجام خواهد شد و آسنکرون هم هستند. حال به سراغ طراحی شمارنده خود می رویم. آن را در یک ماژول دیگر طراحی می کنیم . در طراحی این شمارنده توجه داریم که ورودی D فلاپ فلاپ اول مانند دیگر فلیپ فلاپ ها نیست و ورودی آن

می باشد.

کد وریلاگ زیر طراحی این شمارنده می باشد.

module jCounter #(parameter N = 4)(input clk , set , reset ,

                                    output [N-1 : 0] out);

        DFF dff (clk , set , reset , ~out[N-1] , out[0]);

        genvar j;

        generate

            for(j = 1; j < N; j=j+1)

            begin

                DFF dffs (clk , set , reset , out[j-1] , out[j] );

            end

        endgenerate

endmodule

در توضیح این ماژول می تولن گفت که در بلاک generate از فلیپ فلاپ ها اینستس گرفته ایم و خروجی های q هرکدام را به ورودی های فلیپ فلاپ های بعدی وصل کرده ایم البته خروجی q` فلیپ فلاپ آخری را به ورودی فلیپ فلاپ اول وصل کرده ایم .

حال باید برای حالت های مختلف N تست بنچ یا ماژول تحریک بنویسیم.

برای شبیه سازی کلاک از تکه کد زیر استفاده میکنیم که کلاک با دوره تناوب ۱۰ واحد زمانی ایجاد می کند .

برای حالت N=4:

کد وریلاگ زیر برای تست در حالت N=4 می باشد.

module N4TB;

    parameter N=4;

    reg clk , set , reset;

    wire [N-1 : 0] out;

    initial

        clk = 0;

    always #5 clk = ~clk;

    jCounter #(.N(N)) johnsonCounter(clk , set , reset , out);

    initial

    begin

        set = 0;

        reset = 1;

        #5

        reset = 0;

        #100

        set = 1;

        #5

        set = 0;

        #100

        $stop();

    end

    initial

    begin

        $display("\t\tTime\tq");

        $monitor($time, "\t%b\t", out);

    end

  initial

  begin

    $dumpfile("TB4\_VCD.vcd");

    $dumpvars;

  end

endmodule

برای ذخیره نتایج در فایل VCD تکه زیر را به آن ماژول تست اضافه شده است:

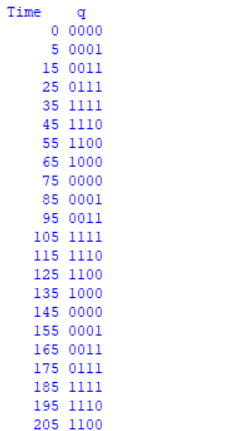
  initial

  begin

    $dumpfile("TB4\_VCD.vcd");

    $dumpvars;

  end

 نتایج زیر بعد از شبیه سازی قابل مشاهده می باشد که درستی طراحی را تایید می کند.

برای حالت N=8 :

کد وریلاگ زیر برای تست در حالت N=8 می باشد:

module N8TB;

    parameter N=8;

    reg clk , set , reset;

    wire [N-1 : 0] out;

    initial

        clk = 0;

    always #5 clk = ~clk;

    jCounter #(.N(N)) johnsonCounter(clk , set , reset , out);

    initial

    begin

        set = 0;

        reset = 1;

        #5

        reset = 0;

        #200

        set = 1;

        #5

        set = 0;

        #200

        $stop();

    end

    initial

    begin

        $display("\t\tTime\tq");

        $monitor($time, "\t%b\t", out);

    end

    initial

    begin

    $dumpfile("TB8\_VCD.vcd");

    $dumpvars;

    end

endmodule

برای ذخیره نتایج در فایل VCD تکه زیر را به آن ماژول تست اضافه شده است:

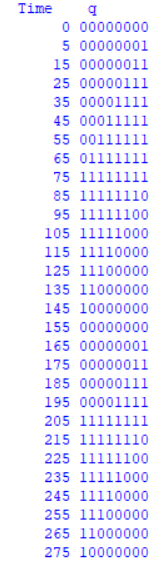
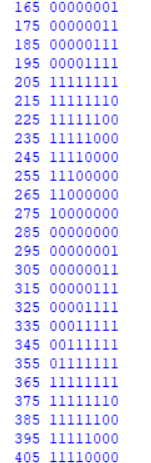
  initial

  begin

    $dumpfile("TB8\_VCD.vcd");

    $dumpvars;

  end

نتایج زیر بعد از شبیه سازی قابل مشاهده می باشد که درستی طراحی را تایید می کند.

برای حالت N=16 :

کد وریلاگ زیر برای تست در حالت N=16 می باشد:

module N16TB;

    parameter N=16;

    reg clk , set , reset;

    wire [N-1 : 0] out;

    initial

        clk = 0;

    always #5 clk = ~clk;

    jCounter #(.N(N)) johnsonCounter(clk , set , reset , out);

    initial

    begin

        set = 0;

        reset = 1;

        #5

        reset = 0;

        #400

        set = 1;

        #5

        set = 0;

        #400

        $stop();

    end

    initial

    begin

        $display("\t\tTime\tq");

        $monitor($time, "\t%b\t", out);

    end

    initial

    begin

    $dumpfile("TB16\_VCD.vcd");

    $dumpvars;

    end

endmodule

برای ذخیره نتایج در فایل VCD تکه زیر را به آن ماژول تست اضافه شده است:

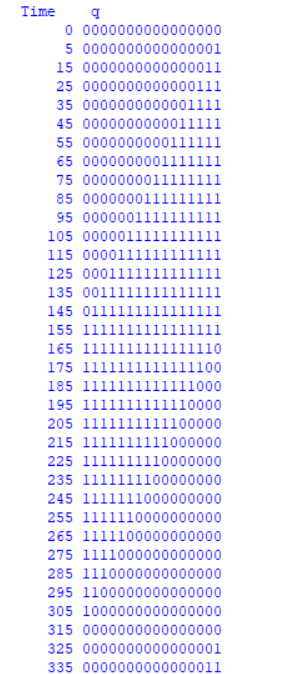
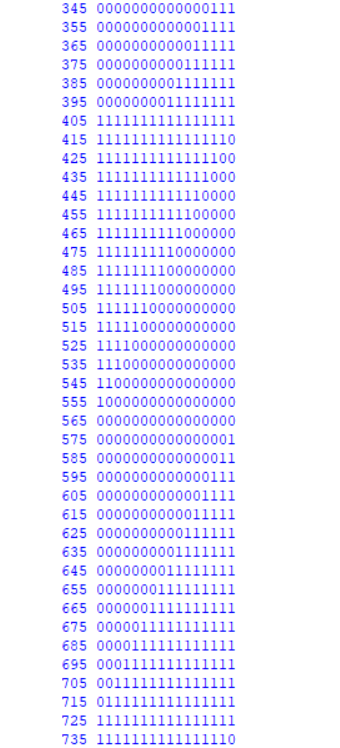
  initial

  begin

    $dumpfile("TB16\_VCD.vcd");

    $dumpvars;

  end

نتایج زیر بعد از شبیه سازی قابل مشاهده می باشد که درستی طراحی را تایید می کند.

برای حالت N=32 :

کد وریلاگ زیر برای تست در حالت N=32 می باشد:

module N32TB;

    parameter N=32;

    reg clk , set , reset;

    wire [N-1 : 0] out;

    initial

        clk = 0;

    always #5 clk = ~clk;

    jCounter #(.N(N)) johnsonCounter(clk , set , reset , out);

    initial

    begin

        set = 0;

        reset = 1;

        #5

        reset = 0;

        #800

        set = 1;

        #5

        set = 0;

        #800

        $stop();

    end

    initial

    begin

        $display("\t\tTime\tq");

        $monitor($time, "\t%b\t", out);

    end

    initial

    begin

    $dumpfile("TB32\_VCD.vcd");

    $dumpvars;

    end

endmodule

برای ذخیره نتایج در فایل VCD تکه زیر را به آن ماژول تست اضافه شده است:

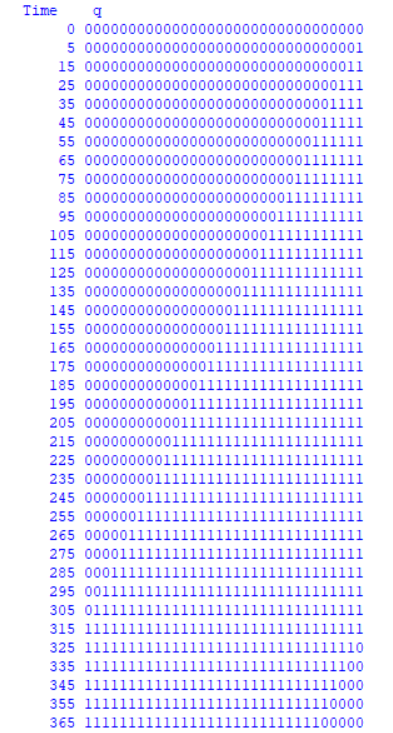
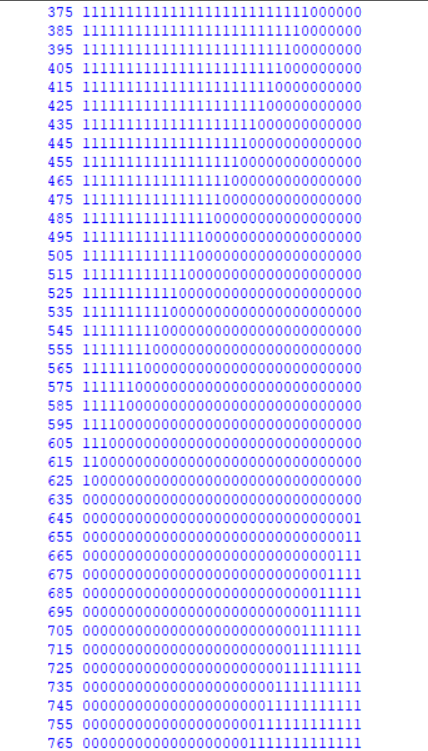
  initial

  begin

    $dumpfile("TB32\_VCD.vcd");

    $dumpvars;

  end

نتایج زیر بعد از شبیه س ازی قابل مشاهده می باشد که درستی طراحی را تایید می کند.

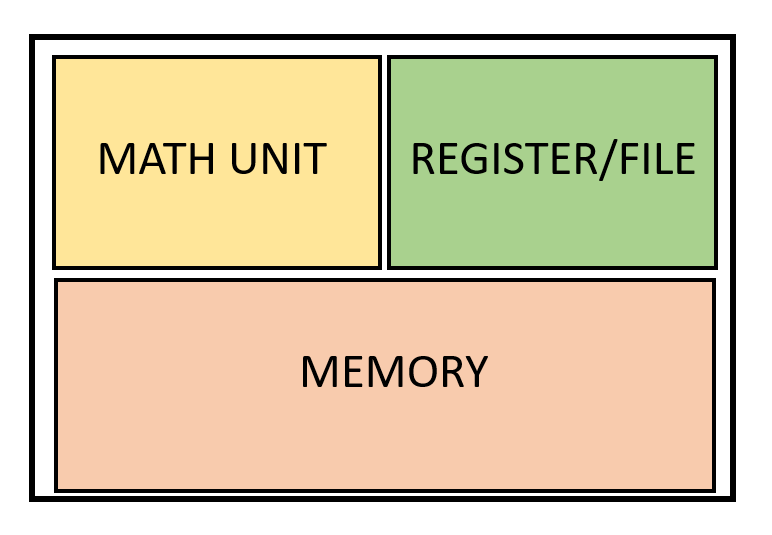
**سوال هفتم:**

در این سوال قصد داریم که پردازنده طراحی کنیم که شامل ۳ بخش زیر می باشد:

۱-رجیستر فایل ۲-واحد محاسبات ۳-حافظه

در ابتدا هر یک از این سه بخش را به طور جداگانه طراحی می کنیم و سپس در یک ماژول نهایی

از هر سه اینستنس میگیریم و کارکرده نهایی پردازنده را پباده سازی می کنیم.



واحد محاسبات:

این واحد برای انجام محاسبات و کارهای منطقی است این واحد باید بتواند دو عملیات ضرب و جمع را انجام دهد .لذا باید نوع عملیاتی که باید انجام شود و دو عدد را به آن به عنوان ورودی بدهیم. و یک خروجی برای آن در نظر بگیریم .البته باید توجه داشته باشیم چون عملیات ضرب این قسمت انجام می دهد و حاضل دو عدد nبیتی یک عدد 2n بیتی خواهد بود و خروجی باید اندازه دوبرابر ورودی ها داشته باشد.

کد وریلاگ زیر طراحی این بخش از پردازنده می باشد. اگر instr که ورودی می دهیم برابر با صفر باشد عملیات جمع و اگر یک باشد عملیات ضرب را انجام میدهد.

اگر instr برابر با صفر باشد در هر مرحله از حلقه، عمل جمع دو عدد 32 بیتی از ورودی (input1 و input2) انجام می‌شود و نتیجه در regout ذخیره می‌شود. عمل جمع اینجا با استفاده از $signed انجام می‌شود که جمع عدد‌های با علامت را انجام می‌دهد. همچنین j \* 64 +: 64 به این معنا است که نتایج جمع در بازه‌های 64 بیتی مختلف (j=0 تا j=15) در regout ذخیره شوند.

اگر instr برابر با یک باشد در هر مرحله از حلقه، عمل ضرب دو عدد 32 بیتی از ورودی (input1 و input2) انجام می‌شود و نتیجه در regout ذخیره می‌شود. عمل ضرب اینجا با استفاده از $signed انجام می‌شود که عمل ضرب عدد‌های با علامت را انجام می‌دهد. همچنین i \* 64 +: 64 به این معنا است که نتایج ضرب در بازه‌های 64 بیتی مختلف (i=0 تا i=15) در regout ذخیره می‌شوند.

module mathUnit(input [511 : 0] input1 , [511 : 0] input2 , instr ,

                output signed [1023 : 0] out

                );

    reg [1023 : 0] regout;

    integer i , j;

    always @(\*)

    begin

        //sum instruction

        if(instr == 1'b0)

        begin

            for (j = 0; j < 16; j = j + 1)

            begin

            regout[j \* 64 +: 64] <= $signed(input1[j \* 32 +: 32]) + $signed(input2[j \* 32 +: 32]);

            end

        end

        //multiply instruction

        else if (instr == 1'b1)

        begin

            for (i = 0; i < 16; i = i + 1)

            begin

            regout[i \* 64 +: 64] <= $signed(input1[i \* 32 +: 32]) \* $signed(input2[i \* 32 +: 32]);

            end

        end

    end

    assign out = regout;

endmodule

واحد register file:

این واحد باید توانایی ذخیره ۴ آرایه ی ۵۱۲ بیتی با نام های A1تاA4 را داشته باشد. برای آدرس دهی هریک از آنها به دو بیت برای آدرس دهی نیاز داریم و آنها را طبق جدول زیر آدرس دهی میکنیم:

|  |  |
| --- | --- |
| 00 | A1 |
| 01 | A2 |
| 10 | A3 |
| 11 | A4 |

این بخش باید توانایی خواندن و نوشتن را داشته باشد پس به آن سیگنال های نوشتن را به عنوان ورودی برای آن ایجاد می کنیم.

همچنین سیگنال ریست را نیز برای آن در نظر می گیریم که در صورت فعال شدن تمامی ثبات ها صفر می شوند و سیگنال ست را نیز برای آن در نظر می گیریم که در صورت فعال شدن تمامی ثبات ها یک می شوند.

طراحی را به گونه ای انجام میدهیم که رجیستر فایل ها همواره در دسترس دیگر بخش ها باشند.

دو درگاه نوشتن داده نیز برای این بخش در نظر میگیریم که نیاز به دو سیگنال فعال سازی می باشند همانطور که قابل مشاهده است با توجه به انتظارات که از طراحی خود داریم تعداد ورودی ها و خروجی های این ماژول زیاد می باشد.

کد وریلاگ زیر طراحی این بخش از پردازنده می باشد:

module register (

        input clk , reset , set ,

        input [511 : 0] input1 , [511 : 0] input2,

        input [1 : 0] wAdd1 , [1 : 0] wAdd2,

        input wEnable1 , wEnable2 ,[1 : 0] rAdd,

        output signed [511 : 0] out,

        output signed [511 : 0] A1 , signed [511 : 0] A2 , signed [511 : 0] A3 , signed [511 : 0] A4

    );

        // 512\*4 registers!

    reg signed [511 : 0] registers [0 : 3];

    integer i,j;

    always @(posedge clk or negedge reset or negedge set)

    begin

        //reset signal is enabled!

        if(!reset)

        begin

            for (j = 0; j < 16; j = j + 1)

            begin

            registers[j] = 512'b0;

            end

        end

        //set signal is enabled!

        else if(!set)

        begin

            for (i = 0; i < 16; i = i + 1)

            begin

            registers[j] = 512'b1;

            end

        end

        //write signal is enabled!

        else

        begin

            if (wEnable1)

            begin

            registers[wAdd1] <= $signed(input1);

            end

            if (wEnable2)

            begin

            registers[wAdd2] <= $signed(input2);

            end

        end

    end

    assign A1 = registers[0];

    assign A2 = registers[1];

    assign A3 = registers[2];

    assign A4 = registers[3];

    assign out = registers[rAdd];

endmodule

**واحد memory:**

چون حافظه دارای ۵۱۲ خانه می باشد پس برای ادرس دهی به ۹ بیت نیاز داریم.

برای حافظه دو سیگنال set و reset را قرار می دهیم که در صورتی که فعال شوند(active low) تمام خانه های حافظه را به ترتیب ۱و۰ می کنند . همچنین برای نوشتن در حافظه یک سیگنال فعالسازی قرار می دهیم و با فعال کردن آن در حافظه می نویسیم.

طراحی که انجام داده ایم به اینگونه می باشد که نوشتن و خواندن در حافظه با گرفتن یک ادرس پایه و دادن آن به عنوان ورودی به memory داده می شود و خروجی memory یک بردار شامل ۱۶ عدد ۳۲ بیتی علامت دار است که ۱۶ خانه ی متوالی از حافظه با شروع از آدرس پایه می باشند. در طراحی خود در اول کار خانه های حافظه را مقدار دهی اولیه میکنیم که در فایل

Initialldata.txtمی باشند و با دستور زیر مقدار دهی اولیه می شود.

    initial

    begin

        $readmemh("Initialldata.txt",data\_memory)

    end

کد وریلاگ زیر طراحی قسمت memory unit می باشد:

module memory (

        input clk , reset , set ,

        input signed [511 : 0] input1 , [8 : 0] address , wEnable,

        output signed [511 : 0] out

    );

    reg signed [31 : 0] mmemory [0 : 511] , signed [511 : 0] MemOut;

    integer i , j , k , l;

    initial

    begin

        $readmemh("Initialldata.txt",mmemory)

    end

    always @(posedge clk or negedge reset or negedge set)

    begin

        if(!reset)

        begin

            for (k = 0 ; k < 32 ; k = k + 1)

            begin

                mmemory[k] <= 32'h0;

            end

        end

        else if(!set)

        begin

            for (l = 0 ; l < 32 ; l = l + 1)

            begin

                mmemory[l] <= 32'h1;

            end

        end

        else

        begin

            if (wEnable)

            begin

                for (i = 0; i < 16; i = i + 1)

                begin

                    mmemory[(i + maddress) % 512] <= $signed(input1[32 \* i +: 32]);

                end

            end

        end

    end

    always @(\*) begin

        for (j = 0; j < 16; j = j + 1)

        begin

            MemOut[32 \* j +: 32] = $signed(mmemory[(j + address) % 512]);

        end

    end

    assign out = MemOut;

endmodule

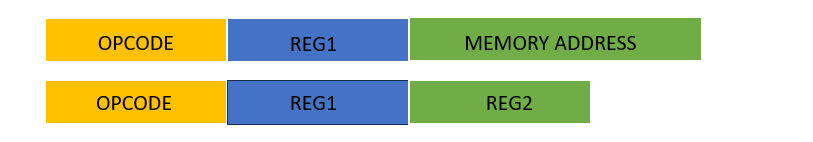
حال ما تمامی بخش های پردازنده موردنظر را طراحی کردیم و باید با اتصال آنها به یکدیگر در ماژول اصلی خود یعنی processor پردازنده را طراحی کنیم.در ایم ماژول از هر ۳ ماژول قبل instance میگیریم.

این پردازنده یک سری دستور را به فرم مشخصی مانند آنچه در ماشین هایی که در درس ساختار و زبان کامپیوتر خواندیم گرفته و آنها را اجرا می کند پس برای آنها یک فرم کلی ارایه میدهیم

باتوجه به اینکه این پردازنده باید چهار عمل گفته شده در صورت سوال را انجام دهد پس برای مقادیر opcode چهار مقدار مختلف داریم و برای مشخص کردن آن دو بیت در نظر می گیریم.

با توجه به اینکه چهار رجیستر داریم برای مشخص کردن رجیستر مورد نیز نیاز به دو بیت داریم.

اندازه حافظه ۵۱۲ خانه می باشد پس با بیت قابل آدرس دهی می باشد.

پس فرم کلی دستورات را میتوان به شکل زیر در نظر گرفت:

برای دستور load کردن ابتدا سیگنال نوشتن رجیستر و شماره ۲حافظه را صفر می کنیم و سیگنال ۱حافظه را فعال میشود سپس بیت ۹و ۱۰دستور در ادرس نوشتن اول قرار میگیرد و بیت های ۰تا۸ دستور در آدرس مموری قرار می گیرد.

برای دستور store کردن سیگنال فعال سازی نوشتن ۱و۲ غیرفعال می شوند سپس سیگنال نوشتن حافظه فعال می شود سپس بیت های ۹و ۱۰ دستور در آدرس خواندن رجیستر قرار می گیرد و در آدرسی که از ۰تا ۸ مشخص می شود ریخته می شود.

برای انجام عمل جمع سیگنال نوشتن مموری صفر می شود و سیگنال های نوشتن رجیستر یک می شوند و خروجی عمل نیز درون ثبات ۳و۴ ریخته می شود همچنین instr مربوط به جمع به sumunitداده می شود تا عملیات جمع را انجام دهد.

برای انجام عمل ضرب سیگنال نوشتن مموری صفر می شود و سیگنال های نوشتن رجیستر یک می شوند و خروجی عمل نیز درون ثبات ۳و۴ ریخته می شود همچنین instr مربوط به ضرب به sumunitداده می شود تا عملیات جمع را انجام دهد.

خروجی های پردازنده هم بصورت آسنکرون آپدیت می شوند.

کد دستور های مختلف در جدول زیر قابل مشاهده می باشد:

|  |  |
| --- | --- |
| 00 | Load |
| 01 | Store |
| 10 | Sum |
| 11 | multiply |

طراحی ماژول اصلی به زبان وریلاگ به شکل زیر می باشد:

module processor(input clk , reset , set , [12 : 0] instruction ,

                output signed [511 : 0] A1 , signed [511 : 0] A2 , signed [511 : 0] A3 , signed [511 : 0] A4);

    //ports for mathUnit

    reg [511 : 0] mathUnitInput1;

    reg [511 : 0] mathUnitInput2;

    reg instr;

    wire signed [1023 : 0] mathUnitOutput;

    //ports register

    reg [511 : 0] registerInput1;

    reg [511 : 0] registerInput2;

    reg [1 : 0] registerWriteAdd1;

    reg [1 : 0] registerWriteAdd2;

    reg registerWriteEnable1;

    reg registerWriteEnable2;

    reg [1 : 0] registerReadAdd;

    wire signed [511 : 0] registerOut;

    wire signed [511 : 0] registerA1;

    wire signed [511 : 0] registerA2;

    wire signed [511 : 0] registerA3;

    wire signed [511 : 0] registerA4;

    //ports for memory

    reg signed [511 : 0] memoryInput;

    reg [8 : 0] memoryAddress;

    reg memoryWriteEnable;

    wire signed [511 : 0] memoryOut;

    //instance from mathUnit

    mathUnit mathUnit (mathUnitInput1 , mathUnitInput2 , instr , mathUnitOutput);

    //instance from register

    register registerr (clk , reset , set , registerInput1 , registerInput2 , registerWriteAdd1 ,

                                    registerWriteAdd2 , registerWriteEnable1 , registerWriteEnable2 ,

                                    registerReadAdd , registerOut , registerA1 , registerA2 , registerA3 , registerA4);

    //instance from memory

    memory memoryy (clk , reset , set , memoryInput, memoryAddress, memoryWriteEnable, memoryOut);

    integer j;

     always @(negedge clk)

     begin

        #5

        if(instruction[12 : 11] == 2'b00)

            begin

                memoryWriteEnable <= 0;

                memoryAddress <= instruction[8 : 0];

                registerWriteEnable1 <= 1;

                registerWriteEnable2 <= 0;

                registerWriteAdd1 <= instruction[10 : 9];

                #5

                registerInput1 <= memoryOut;

            end

        else if(instruction[12 : 11] == 2'b01)

            begin

                memoryWriteEnable <= 1;

                memoryAddress <= instruction[8 : 0];

                registerWriteEnable1 <= 0;

                registerWriteEnable2 <= 0;

                registerWriteAdd1 <= instruction[10 : 9];

                #5

                registerInput1 <= memoryOut;

            end

        else if(instruction[12 : 11] == 2'b10)

            begin

                memoryWriteEnable <= 0;

                registerWriteEnable1 <= 1;

                registerWriteEnable2 <= 1;

                registerWriteAdd1 <= 2'b10;

                registerWriteAdd2 <= 2'b11;

                instr = 1'b0;

                mathUnitInput1 <= registerA1;

                mathUnitInput2 <= registerA2;

                #5

                for(j = 0 ; j < 16 ; j = j + 1)

                begin

                    registerInput1[32 \* j +: 32] <= mathUnitOutput[64 \* j +: 32];

                    registerInput2[32 \* j +: 32] <= mathUnitOutput[64 \* j + 32 +: 32];

                end

            end

        else if(instruction[12 : 11] == 2'b11)

            begin

                memoryWriteEnable <= 0;

                registerWriteEnable1<= 1;

                registerWriteEnable1 <= 1;

                registerWriteAdd1 <= 2'b10;

                registerWriteAdd1  <= 2'b11;

                instr = 1'b1;

                mathUnitInput1 <= registerA1;

                mathUnitInput2 <= registerA2;

                #5

                for(j = 0; j < 16; j = j + 1)

                begin

                    registerInput1[32 \* j +: 32] <= mathUnitOutput[64 \* j +: 32];

                    registerInput2[32 \* j +: 32] <= mathUnitOutput[64 \* j + 32 +: 32];

                end

            end

    end

    assign A1 = registerA1;

    assign A2 = registerA2;

    assign A3 = registerA3;

    assign A4 = registerA4;

endmodule

حال پس از طراحی پردازنده خود باید آن را در حالت های مرزی تست کنیم:

حالت های مرزی می توانند اعداد 1,0,-1بزگترین عدد مثبت و کوچکترین عدد منفی باشند.

تمامی این حالات در مقدار دهی اولیه حافظه در حافظه وجود دارند.

کلاک را برابر ۸۰ واحد زمانی می گیریم تا مطمعن شویم از بیشینه تاخیر های موجود در مدار بیشتر است و این مورد در عملکرد مدار مشکلی ایجاد نمی کند.

ابتدا بدون دادن تست مدار را اجرا می کنیم که خروجی زیر حاصل می شود:

module test1;

    reg clk , reset , set ;

    reg [12 : 0] instruction;

    wire [511 : 0] A1 ;

    wire [511 : 0] A2 ;

    wire [511 : 0] A3 ;

    wire [511 : 0] A4;

    initial

        clk = 0;

    always

        #40 clk = ~clk;

    processor Processor (clk, reset , set , instruction , A1 , A2 , A3 , A4);

    initial

        $monitor($time , ":\nA1 = %h\nA2 = %h\nA3 = %h\nA4 = %h\n " , A1 , A2 , A3 , A4 );

endmodule

حال تست زیر را انجام میدهیم که در صورت تطبیق با داده های اولیه مموری و دستور عمل ها متوجه می شویم که پردازنده به درستی کار می کند.

module test1;

    reg clk , reset , set ;

    reg [12 : 0] instruction;

    wire [511 : 0] A1 ;

    wire [511 : 0] A2 ;

    wire [511 : 0] A3 ;

    wire [511 : 0] A4;

    initial

        clk = 0;

    always

        #40 clk = ~clk;

    processor Processor (clk, reset , set , instruction , A1 , A2 , A3 , A4);

    initial

    begin

        instruction <= {2'b10, 11'b0};

        #80

        instruction <= {2'b11, 11'b0};

        #80

        instruction <= {4'b0000, 9'b0};

        #80

        instruction <= {4'b0001, 9'b10000};

        #80

        instruction <= {2'b10, 11'b0};

        #80

        instruction <= {2'b11, 11'b0};

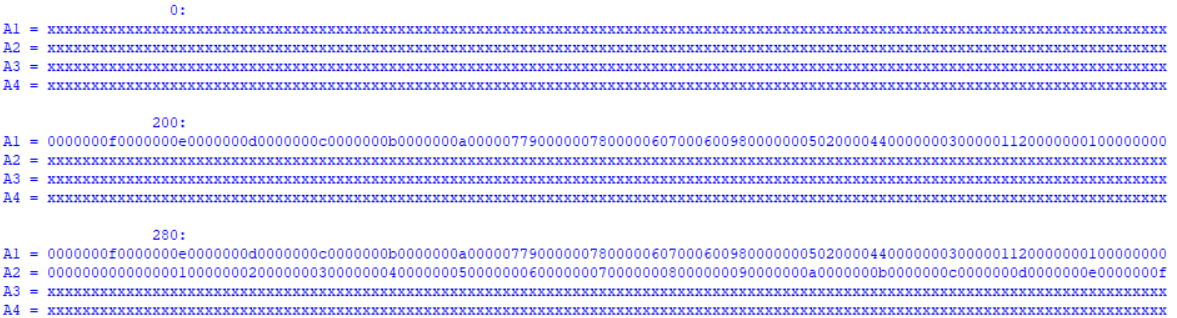
        #80

        $stop();

    end

    initial

        $monitor($time , ":\nA1 = %h\nA2 = %h\nA3 = %h\nA4 = %h\n " , A1 , A2 , A3 , A4 );

endmodule

module test1;

    reg clk , reset , set ;

    reg [12 : 0] instruction;

    wire [511 : 0] A1 ;

    wire [511 : 0] A2 ;

    wire [511 : 0] A3 ;

    wire [511 : 0] A4;

    initial

        clk = 0;

    always

        #40 clk = ~clk;

    processor Processor (clk, reset , set , instruction , A1 , A2 , A3 , A4);

    initial

    begin

        instruction <= {2'b10, 11'b0};

        #80

        instruction <= {2'b11, 11'b0};

        #80

        instruction <= {4'b0000, 9'b0};

        #80

        instruction <= {4'b0001, 9'b10000};

        #80

        instruction <= {2'b10, 11'b0};

        #80

        instruction <= {2'b11, 11'b0};

        #80

        instruction <= {4'b0110, 9'b111111};

        #80

        instruction <= {4'b0111, 9'b1111111};

        #80

        instruction <= {4'b0000, 9'b111111};

        #80

        instruction <= {4'b0001, 9'b1111111};

        #80

        instruction <= {2'b10, 11'b0};

        #80

        instruction <= {2'b11, 11'b0};

        #80

        $stop();

    end

    initial

        $monitor($time , ":\nA1 = %h\nA2 = %h\nA3 = %h\nA4 = %h\n " , A1 , A2 , A3 , A4 );

endmodule



این تست سوم می باشد که نشان می دهد مدار ما به درستی کار می کند.

حال داده های مموری را که در فایل اولیه هستند تغییر می دهیم و دوباره اقدام به تست می کنیم. فایل داده های اولیه قبلی در پوشه سوال ۷ به نام initialldataold هنوز قرار دارد و فایل جدید initialldata جدید را با داده های جدید پر می کنیم و با شبیه سازی دوباره ماژول تست نتایج زیر را می بینیم

