



DIGITAL SYSTEM DESIGN

PROFESSOR :

BAYAT

آزمایش ۸

HOOMAN KESHVARI

99105667

ALIREZA FOROODNIA

99105645

مقدمه:

در آزمایش یک واحد ALU برای اعداد مختلط طراحی شده است که دارای پایپلاین ۴ مرحله ای (شامل خواندن دستور، لود کردن operand ها، انجام عملیات های مربوط به ضرب و جمع اعداد مختلط، store کردن خروجی واحد arithmetic در مموری) است.

جزئیات پیاده سازی:

ماژول mul :

```
module MUL(
    input clk,rst,
    input [7:0] in1,in2,
    output reg [7:0] out,
    output reg is_done
);

reg signed [3:0] a,b,c,d;
reg [1:0] counter;

always @(posedge clk,negedge rst) begin
    if(rst == 0) begin
        counter = 0;
    end
    else begin
        if(counter < 5) begin
            counter = counter + 1;
            is_done = 0;
        end
        if(counter == 5) begin
            a = in1[7:4];
            b = in1[3:0];
            c = in2[7:4];
            d = in2[3:0];
            out[3:0] = a*d + b*c;
            out[7:4] = a*c - b*d;
            is_done = 1;
            counter = 0;
        end
    end
end

endmodule
```

این ماژول به صورت اسنکرون ریست می پذیرد. سپس هر ورودی ای که به آن وارد شود طی پنج کلاک عملیات ضرب اعداد مختلط را بر روی آن صورت داده و سپس نتایج ضرب را به همراه سیگنال `is_done` به خروجی می دهد.

ماژول `add_sub` :

```
module ADD_SUB(  
    input clk,rst,  
    input [7:0] in1,in2,  
    input [1:0] add_or_sub,  
    output reg [7:0] out,  
    output reg is_done  
);  
  
reg signed [3:0] a,b,c,d;  
reg counter;  
  
always @(posedge clk,negedge rst) begin  
    if(rst == 0) begin  
        counter = 0;  
        is_done = 0;  
    end  
    else begin  
        if(counter < 2) begin  
            counter = counter + 1;  
            is_done = 0;  
        end  
        if(counter == 2) begin  
            a = in1[7:4];  
            b = in1[3:0];  
            c = in2[7:4];  
            d = in2[3:0];  
            if(add_or_sub == 1) begin  
                out[7:4] = a - c;  
                out[3:0] = b - d;  
            end else if(add_or_sub == 0) begin  
                out[7:4] = a + c;  
                out[3:0] = b + d;  
            end  
            is_done = 1;  
            counter = 0;  
        end  
    end  
end  
  
endmodule
```

این ماژول به کمک ورودی add_or_sub ابتدا تشخیص می دهد که کدام یک از اعمال جمع یا تفریق باید صورت بپذیرد و سپس طی دو کلاک نتیجه اعمال تشخیص داده شده را به خروجی ها منتقل کرده و سیگنال is_done را یک می کند.

ماژول های data_mem و instruction_mem :

این ماژول ها به ترتیب مسئولیت حفظ داده های عملیات ها و دستورات را بر عهده دارند. شیوه طراحی آنها به این گونه است که دارای ریست آسنکرون می باشند و در صورت رخ دادن ریست به جای صفر شدن داده ها مقادیری را که برای انجام عملیات ها لازم دارند به درون آنها منتقل می کنیم.

```
module inst_mem(
    input clk,
    input rst,
    input enable,
    input read_writenot,
    input [19 : 0] in_data,
    input [4 : 0] read_address,
    input [4 : 0] write_address,

    output reg [19 : 0] out_data
);

reg [19 : 0] storage [31 : 0];
integer i;

always @(negedge rst) begin
    for (i = 0; i<20 ; i = i + 1 ) begin
        storage[i] = 0;
    end
    storage[0] = 20'b 00_000000_000001_000010;
    storage[1] = 20'b 00_000011_000100_000101;
    storage[2] = 20'b 00_000110_000111_001000;
    storage[3] = 20'b 01_001001_001010_001011;
    storage[4] = 20'b 01_001100_001101_001110;
    storage[5] = 20'b 10_001111_010000_010001;
    storage[6] = 20'b 10_010010_010011_010100;
    storage[7] = 20'b 11_000000_000000_000000;
end

always @(posedge clk) begin

    if(enable) begin

        if(read_writenot) begin
            out_data = storage[read_address];
        end
    end
end
```

```

        end
        else begin
            storage[write_address] = in_data;
        end
    end

end

end

endmodule

```

ماژول های LD, IF, EX :

این ماژول ها از تعدادی بافر که خروجی ها را به ورودی ها منتقل می کنند تشکیل شده اند تا نقش انتقال داده ها را در پایپلاین اجرا کنند. همه آنها از یک سیگنال freeze مشترک نیز استفاده می کنند تا در مواقع لازم (فاصله زمانی اجرای یک دستور ضرب یا جمع یا تفریق که باید سیستم متوقف باشد) بتوان از عملکرد آن جلوگیری کرد.

```

module EX(
    // inputs
    input ld_inst_halt,
    input halted,
    input [7:0] alu_output,
    input [5:0] write_addr,
    input freeze,
    input clk,
    input data_rw,
    input data_mem_write_ex,

    // outputs
    output reg halted_out,
    output reg data_rw_out,
    output reg [7:0] alu_output_out,
    output reg [5:0] write_addr_out,
    output reg data_mem_write_out_ex
);

always @(posedge clk) begin
    if(!freeze && !ld_inst_halt) begin
        halted_out = halted;
        data_rw_out = data_rw;
        alu_output_out = alu_output;
        write_addr_out = write_addr;
        data_mem_write_out_ex = data_mem_write_ex;
    end
end

```

```

    end
end

endmodule

```

ماژول data_path :

این ماژول شامل اتصالات ما بین تمام اجزای سیستم طراحی شده از جمله پایپلین ها، واحد کنترل، واحد جمع و تفریق و ضرب و واحد های مموری داده و دستور است.

نحوه کار کلی سیستم :

با هر کلاک ابتدا دستور از حافظه مرتبط با دستور ها خوانده شده و سپس به اولین واحد پایپلین منتقل می شود.

سپس در ادامه operand ها از داده ها جدا شده و برای گرفته شدن از حافظه داده به آن وارد می شوند. همچنین opcode دستور وارد واحد کنترل و پایپلین می شود تا در ادامه به alu نیز منتقل بشود.

در قسمت مربوط به alu پایپلین operand های گرفته شده از مموری در مرحله قبل را به علاوه opcode گرفته شده را به ALU وروی می دهد. واحد arithmetic نیز پس از انجام عملیات مربوطه با یک کردن سیگنال is_done به بقیه مدار اجازه فعالیت می دهد و خروجی های محاسبات را راهی پایپلین می کند.

در نهایت پایپلین آخر خروجی محاسبات را به علاوه آدرس ذخیره که از مرحله دوم تا اینجا پایپ شده است را به واحد حافظه می دهد تا این واحد آنها را ذخیره کرده و یک عملیات خاتمه بیابد.

فعالیت مدار همواره تحت تاثیر سیگنال is_done خروجی از ALU می باشید زیرا تا زمانی که این واحد نتیجه را آماده نکرده باشد نباید پایپلین به فعالیت ادامه دهد.

واحد کنترل بسته به نوع opcode وظیفه متوقف کردن مدار یا ایجاد سیگنال هایی که برای انجام آن فرمان ضروری هستند را دارد. این سیگنال ها در صورت لزوم در طول مدار پایپ می شود تا به مقصد خود برسند.

تست بنچ :

```

module tb_;
reg clk;
reg rst;
wire halted;

CPU uut(
    .rst (rst),

```

```

        .clk (clk),
        .halted(halted)
    );

localparam CLK_PERIOD = 2;
always #(CLK_PERIOD/2) clk=~clk;

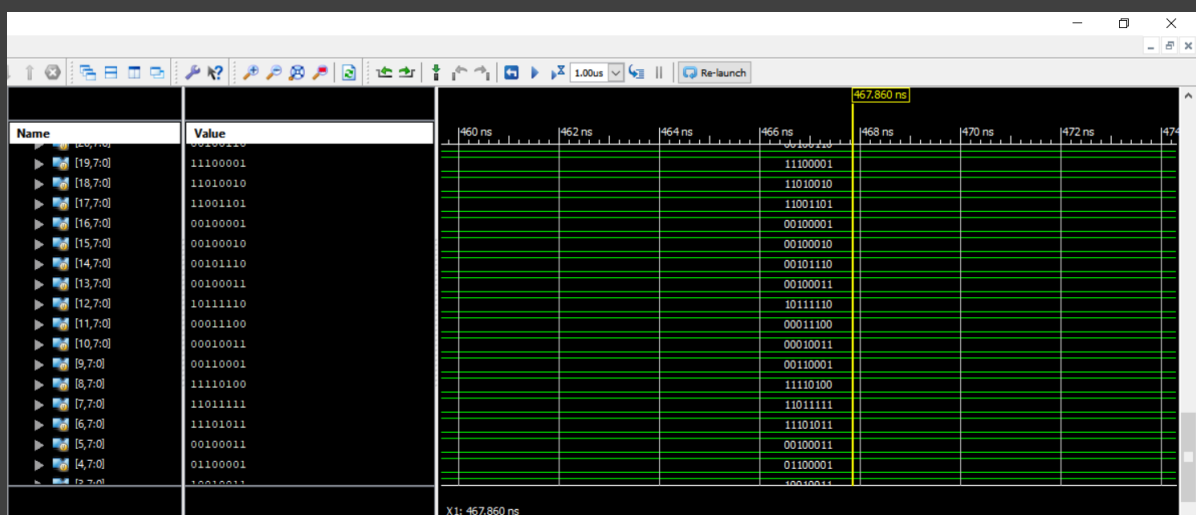
initial begin
    rst = 0;
    clk = 1;
    #1
    rst = 1;
    @(posedge halted)
    $finish();
end

endmodule

```


در تست بنچ ابتدا با ریست شدن مدار مقادیر به واحد های حافظه منتقل می شوند و سپس با ضربان های کلاک مدار فعالیت کرده و خروجی ها را ایجاد می کند.

شکل موج :



d:\lab\8.2\axise - [Design Summary (Implemented)]

WindowLayoutHelp



ADD_SUB Project Status (09/01/2022 - 03:14:19)

| | | | |
|------------------|---|-----------------------|---|
| Project File: | axise | Parser Errors: | No Errors |
| Module Name: | CPU | Implementation State: | Placed and Routed |
| Target Device: | xc6slx9-2tgg144 | • Errors: | No Errors |
| Product Version: | TSE 14.7 | • Warnings: | 722 Warnings (722 new) |
| Design Goal: | Balanced | • Routing Results: | All Signals Completely Routed |
| Design Strategy: | Virtex Default (unlocked) | • Timing Constraints: | |
| Environment: | System Settings | • Final Timing Score: | 0 [Timing Report] |

Device Utilization Summary

| Slice Logic Utilization | Used | Available | Utilization | Note(s) |
|--|------|-----------|-------------|---------|
| Number of Slice Registers | 0 | 11,440 | 0% | |
| Number of Slice LUTs | 0 | 5,720 | 0% | |
| Number of occupied Slices | 0 | 1,430 | 0% | |
| Number of MUXCts used | 0 | 2,860 | 0% | |
| Number of LUT Flip Flop pairs used | 0 | | | |
| Number of bonded IOBs | 1 | 102 | 1% | |
| Number of RAMB16BWRs | 0 | 32 | 0% | |
| Number of RAMB36BWRs | 0 | 64 | 0% | |
| Number of BUFIO2/BUFIO2_2CLKs | 0 | 32 | 0% | |
| Number of BUFIO2FB/BUFIO2FB_2CLKs | 0 | 32 | 0% | |
| Number of BUFG/BUFGMUXs | 0 | 16 | 0% | |
| Number of DCM/DCM_CLKGENs | 0 | 4 | 0% | |
| Number of ILLOGIC2/ISERDES2s | 0 | 200 | 0% | |
| Number of IOB_LAY2IOB_RP2/IOB_RP2_MC0s | 0 | 200 | 0% | |
| Number of OLOGIC2/OSERDES2s | 0 | 200 | 0% | |
| Number of BSCANs | 0 | 4 | 0% | |
| Number of BUFHs | 0 | 128 | 0% | |
| Number of BUFPLLs | 0 | 8 | 0% | |