گزارش آزمایش پنجم آزمایشگاه
طراحی سیستم دیجیتال

مزدک تیموریان

۴۰۱۱۰۱۴۹۵

مریم شیران

۴۰۰۱۰۹۴۴۶

مهدی بهرامیان

۴۰۱۱۷۱۵۹۳

دانشکده مهندسی کامپیوتر

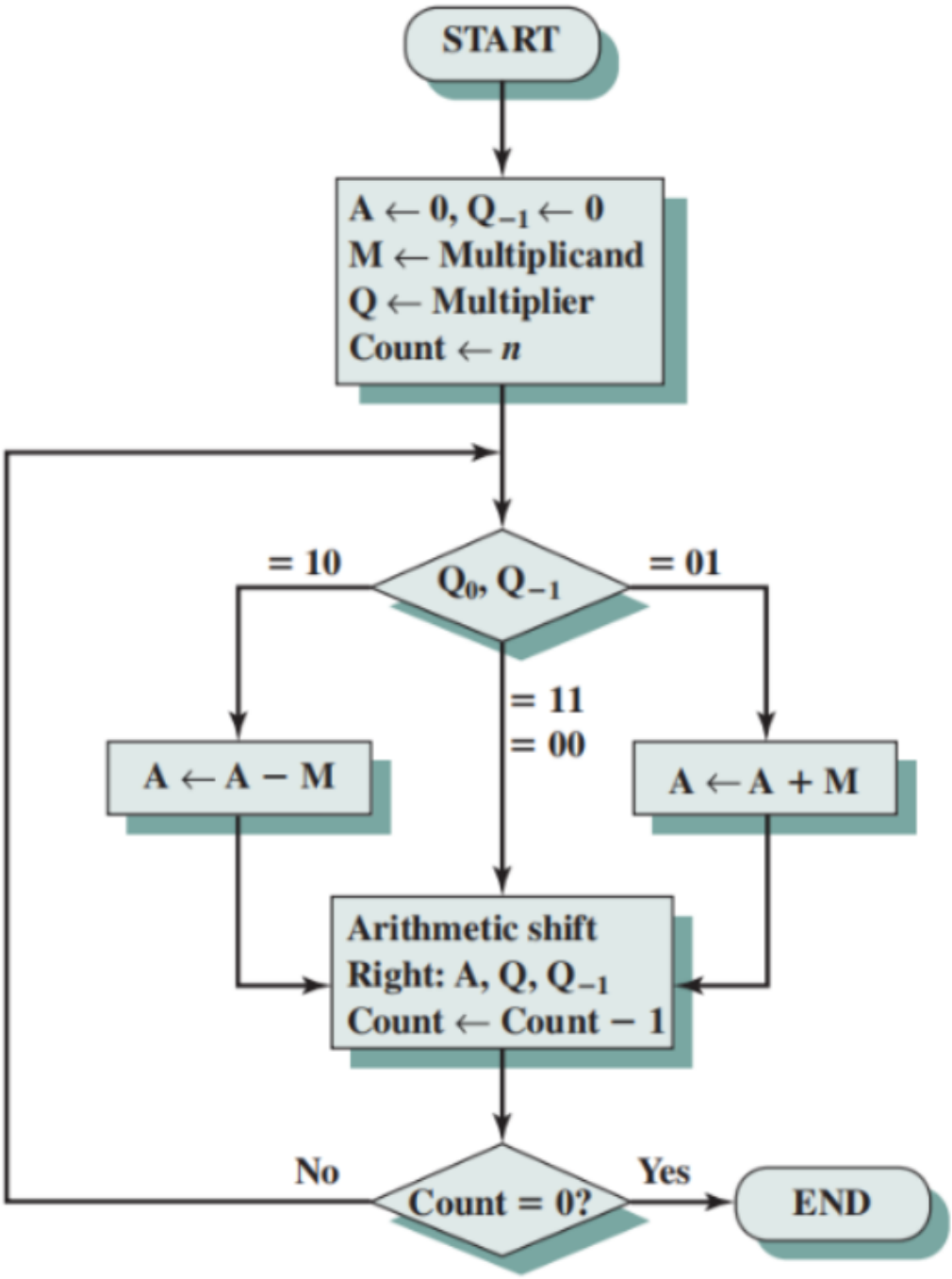
دانشگاه صنعتی شریف

تیر ۱۴۰۳

فهرست مطالب

۱ مقدمه

هدف از این آزمایش طراحی یک واحد ضرب کننده است که برای انجام عمل ضرب از روش ضرب Booth استفاده میکند. برای انجام این آزمایش مسیرداده و واحد کنترل را جداگانه طراحی کرده و سپس آنها را بهم متصل میکنیم.



نمودار الگوریتم Booth را نیز میتوانید در شکل بالا مشاهده کنید که با شیفت دادن یک عدد و چک کردن تغییر بیت از ۰ به ۱ و یا بالعکس مقدار عدد دوم را از حاصل کم کرده و یا به آن اضافه میکند.

۲ شرح آزمایش

۱.۲ First Indices

برای اینکه عملیات شیفت به راست سریع‌تر انجام شود بیت‌های تکراری ۰ و ۱ را شناسایی کرده و در یک کلاک به تعداد آنها شیفت میدهیم زیرا در آن حالت جمع و یا تفریقی انجام نمیشود.
برای انجام این کار ماژولی طراحی میکنیم که اولین بیت ۰ و ۱ را در عدد پیدا کند.

```
module First_indices (
    input [3:0] num,
    output [2:0] index0,
    output [2:0] index1
);
    assign index0 =
    num[0] ? (num[1] ? (num[2] ? (num[3] ? 4 : 3) : 2) : 1) : 0;
    assign index1 =
     ~num[0] ? (~num[1] ? (~num[2] ? (~num[3] ? 4 : 3) : 2) : 1) : 0;

endmodule
```

همچنین برای بررسی درستی آن ماژول تست زیر را طراحی کرده‌ایم.

```
module TB_FI ();
    reg [3:0] num;
    wire [1:0] index0;
    wire [1:0] index1;

    First_indices fi (
        .num(num),
        .index0(index0),
        .index1(index1)
    );


    initial begin
        num = 12;
        #10;
        num = 15;
        #10
        num = 7;
        #10
        $stop();
    end

    initial begin
        $monitor("number: %d", num, " index 0: %d", index0, " index 1: %d", index1);
    end
endmodule
```

Datapath ۲.۲

در ادامه مسیر داده را طراحی میکنیم.

```
module Datapath (
    input clk,
    input rst,
    input done,
    input op,
    input [3:0] multiplier,
    input [3:0] multiplicand,
    input [2:0] shift_a,
    input [2:0] shift_b,
    output reg [7:0] result,
    output reg [3:0] B
);
    reg [7:0] A;

    always @(posedge clk, negedge rst) begin
        if (~rst) begin
            A <= {{4{multiplicand[3]}},multiplicand};
            B <= {{4{multiplier[3]}},multiplier};
            result <= 0;
        end
        else if (~done) begin
            B <= B >> shift_b;
            if (op) begin
                result <= result + (A << shift_a);
            end
            else begin
                result <= result - (A << shift_a);
            end
        end
    end

endmodule
```

همانطور که در شکل بالا مشاهده میکنید در شروع برنامه ابتدا multiplicand و multiplier را Sign Extend کرده و در A و B میریزیم. سپس در هر مرحله B را به راست شیفت داده و بجای شیفت دادن result (نتیجه ضرب) عدد A را به چپ شیفت داده و با توجه به مقدار op که از روی ۰۱ و یا ۱۰ بودن دو بیت سمت چپ معلوم میشود مقدار حاصله را با result جمع کرده و یا از آن کم میکنیم. این پروسه را تاجایی ادامه میدهیم که واحد کنترلی سیگنال done را ۱ کند که به معنای آماده شدن نتیجه است.

Control Unit ۳.۲

حال واحد کنترلی را طراحی میکنیم که وظیفه آن مشخص کردن سیگنال‌های کنترلی از روی وضعیت مدار است.

```
module CU (
    input clk,
    input rst,
    output op,
    output done,
    input [3:0] B,
    output [2:0] shift_a,
    output [2:0] shift_b
);
    wire [2:0] index0, index1;
    reg [2:0] total_shift;
    reg [2:0] start;

    First_indices fi (
        .num(B),
        .index0(index0),
        .index1(index1)
    );

    assign shift_a = total_shift + shift_b;
    assign shift_b = op ? index0 : index1;

    assign op = start & B[0];
    assign done = shift_a >= 4;

    always @(posedge clk, negedge rst) begin
        if (~rst) begin
            total_shift <= 0;
            start <= 0;
        end
        else begin
            start <= 1;
            total_shift <= total_shift + shift_b;
        end
    end

endmodule
```

همانطور که مشاهده میکنید در هر مرحله با استفاده از ماژول First_indices و بیت کم ارزش B عملیات مورد نیاز و مقدار شیفت دادن بدست میاید. همچنین از آنجایی که اعداد ۴ بیتی هستند پس از ۴ بار شیفت دادن A الگوریتم خاتمه میابد و سیگنال done ۱ میشود.

Booth ۴.۲

حال با کنار اتصال مسیرداده و واحد کنترلی ماژول اصلی را طراحی میکنیم.

```
module Booth (
    input [3:0] multiplicand,
    input [3:0] multiplier,
    input clk,
    input rst,
    output done,
    output [7:0] result
);
    wire [3:0] B;
    wire [2:0] shift_a, shift_b;

    CU cu (
        .clk(clk),
        .rst(rst),
        .done(done),
        .op(op),
        .shift_a(shift_a),
        .shift_b(shift_b),
        .B(B)
    );

    Datapath dp (
        .clk(clk),
        .rst(rst),
        .done(done),
        .op(op),
        .multiplicand(multiplicand),
        .multiplier(multiplier),
        .shift_a(shift_a),
        .shift_b(shift_b),
        .B(B),
        .result(result)
    );

endmodule
```

۳ شبیه‌سازی و بررسی عملکرد ضرب‌کننده

برای بررسی عملکرد ماژول اصلی یک ماژول تست برای آن مینویسیم.

```
module TB_Booth ();
    reg clk = 1;
    reg rst = 1;
    reg signed [3:0] multiplicand;
    reg signed [3:0] multiplier;
    wire done;
    wire signed [7:0] result;

    Booth booth (
        .clk(clk),
        .rst(rst),
        .multiplicand(multiplicand),
        .multiplier(multiplier),
        .done(done),
        .result(result)
    );

    always begin
        #5 clk <= ~clk;
    end

    initial begin

        multiplicand=5; multiplier=3;
        rst=0;
        #10 rst=1;
        wait(done);

        $display("time: %d multiplicand: %d multiplier: %d result: %d",
        $time ,multiplicand, multiplier, result);

        multiplicand=3; multiplier=5;
        rst=0;
        #10 rst=1;
        wait(done);

        $display("time: %d multiplicand: %d multiplier: %d result: %d",
```

```
        multiplicand=8; multiplier=7;
        rst=0;
        #10 rst=1;
        wait(done);

        $display("time: %d multiplicand: %d multiplier: %d result: %d",
        $time ,multiplicand, multiplier, result);

        multiplicand=-7; multiplier=-5;
        rst=0;
        #10 rst=1;
        wait(done);

        $display("time: %d multiplicand: %d multiplier: %d result: %d",
        $time ,multiplicand, multiplier, result);

        multiplicand=3; multiplier=-6;
        rst=0;
        #10 rst=1;
        wait(done);

        $display("time: %d multiplicand: %d multiplier: %d result: %d",
        $time ,multiplicand, multiplier, result);

        multiplicand=6; multiplier=-1;
        rst=0;
        #10 rst=1;
        wait(done);

        $display("time: %d multiplicand: %d multiplier: %d result: %d",
        $time ,multiplicand, multiplier, result);

        multiplicand= 7; multiplier=1;
        rst=0;
        #10 rst=1;
```

```
        $display("time: %d multiplicand: %d multiplier: %d result: %d",
        $time ,multiplicand, multiplier, result);

        multiplicand= 7; multiplier=1;
        rst=0;
        #10 rst=1;
        wait(done);

        $display("time: %d multiplicand: %d multiplier: %d result: %d",
        $time ,multiplicand, multiplier, result);

        multiplicand=0; multiplier=6;
        rst=0;
        #10 rst=1;
        wait(done);

        $display("time: %d multiplicand: %d multiplier: %d result: %d",
        $time ,multiplicand, multiplier, result);

        $stop();

    end

endmodule
```

حاصل شبیه‌سازی ماژول بالا را در شکل‌های زیر میتوانید مشاهده کنید.

```
# time:                20 multiplicand:    5 multiplier:    3 result:    15
# time:                60 multiplicand:    3 multiplier:    5 result:    15
# time:                80 multiplicand:   -8 multiplier:    7 result:   -56
# time:               110 multiplicand:   -7 multiplier:   -5 result:    35
# time:               140 multiplicand:    3 multiplier:   -6 result:   -18
# time:               150 multiplicand:    6 multiplier:   -1 result:    -6
# time:               170 multiplicand:    7 multiplier:    1 result:     7
# time:               190 multiplicand:    0 multiplier:    6 result:     0
```