دانشکده مهندسی کامپیوتر

آزمایشگاه طراحی سیستم‌های دیجیتال

آزمایش پنجم - طراحی ضرب کننده

دکتر اجلالی، مهندس اثنی عشری

امیرمهدی کوششی — ۹۸۱۷۱۰۵۳
ایمان محمدی — ۹۹۱۰۲۲۰۷
شایان صالحی — ۹۹۱۰۵۵۶۱

۶ شهریور ۱۴۰۲

فهرست مطالب

۱ مقدمه

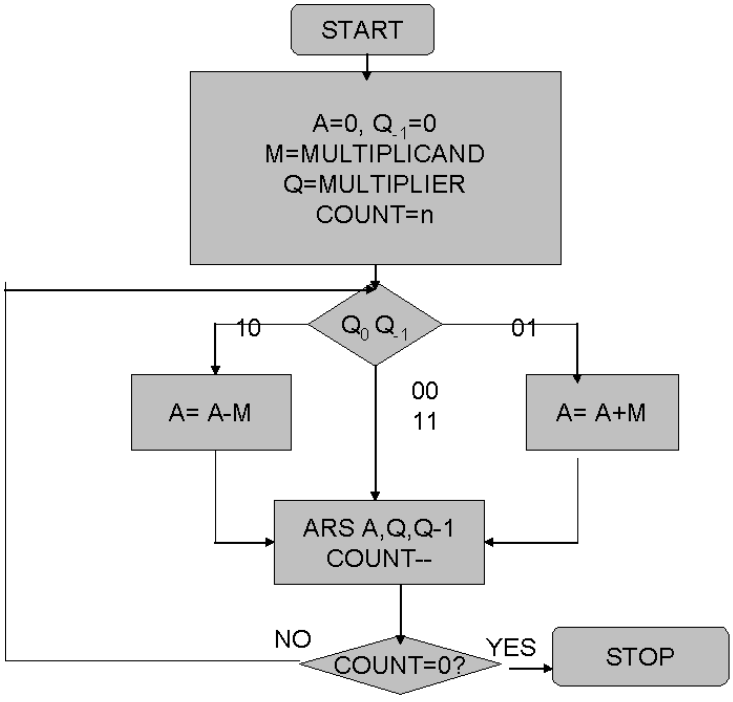
هدف این آزمایش این است که یک ضرب کننده با استفاده از الگوریتم بوث طراحی شود. ما در اینجا یک مدل کلی برای ضرب کننده خواهیم داشت که از دو ماجول کنترل واحد و انتقال داده تشکیل شده است. در قسمت کنترل واحد تمام حالت‌ها و استیت‌های ضرب کننده نگه داشته می‌شود و در ماجول انتقال داده تمامی محاسبات‌ها انجام می‌شود.

۲ شرح آزمایش

در این آزمایش ورودی‌ها به صورت ۴ بیتی دریافت می‌شود که اعداد علامت‌دار هستند، در خروجی نیز یک عدد ۸ بیتی به عنوان نتیجه داده می‌شود. نکته این مدار ضرب کننده این است که در هر مرحله به اندازه‌ای که نیاز دارد عمل شیف را انجام می‌دهد و به نوعی یک ضرب کننده بهینه است.

۱.۲ الگوریتم بوث

با توجه به فلوچارت زیر برای الگوریتم بوث داریم: این الگوریتم برای این اساس کار می‌کند که دارای یک بیت



شکل ۱: نحوه کارکرد الگوریتم بوث

LSB یا همان Least Significant Bit و با توجه به این بیت و کم ارزش‌تر بیت ضرب کننده در حالت‌های 01 و 01 عملیات‌های جمع و تفریق را انجام می‌دهد و مبنای پیاده‌سازی ما نیز همین الگوریتم خواهد بود.

۳ ساختار مدل‌ها

در این قسمت ساختار تمامی مدل‌های زده شده را شرح می‌دهیم.

۱.۳ مدل ضرب کننده

در این قسمت ورودی‌ها به صورت ضرب‌کننده و ضرب‌شونده بوده که در قالب یک عدد علامت‌دار ۴ بیتی وارد می شوند، همچنین سینگنال‌های شروع و کلاک نیز دریافت می‌کنیم و نتیجه و معتبر بودن نتیجه را به عنوان خروجی بیرون می‌دهد. همچنین شامل دو مدل کنترل واحد و مسیر داده بوده و سیم‌های shiftRight ، load و arth برای ارتباط به این دو مدل هستند. در آخر بیت‌های booth-bits نشان‌دهنده تعداد شیف‌های لازم خواهند بود.

```
1   module BoothMultiplier(
2       input wire [3:0] multiplicand, multiplier,
3       input clk, str,
4       output wire [7:0] result,
5       output valid
6   );
7
8       wire shiftRight, load, arth;
9       wire [2:0] booth_bits;
10
11      BoothMultiplierDatapath datapath(
12          .multiplicand(multiplicand), .multiplier(multiplier),
13          .shiftRight(shiftRight), .load(load), .arth(arth),
14          .booth_bits(booth_bits), .clk(clk),
15          .result(result [7:4]), .new_multiplier(result [3:0])
16      );
17
18      BoothMultiplierControlUnit control(
19          .multiplier(result[3:0]), .clk(clk), .str(str),
20          .valid(valid), .load(load), .arth(arth), .shiftRight(shiftRight),
21          .booth_bits(booth_bits)
22      );
23
24  endmodule
```

شکل ۲: مدل ضرب کننده

۲.۳ مدل مسیر داده

در این قسمت شاهد مدل مسیر داده هستیم که در آن محاسبات براساس دو بیت LSB و multiplier[۰] انجام می‌شود. همچنین در هنگام شیف خروجی را به اندازه booth-bits شیفت می‌دهد. در نهایت نیز اگر سیگنال load یک باشد مقدارهای اولیه را مقداردهی می‌کند.

```
module BoothMultiplierDatapath(
    input [3:0] multiplicand, multiplier,
    input shiftRight, load, arth,
    input [2:0] booth_bits,
    input clk,
    output reg [3:0] result, new_multiplier
);

    reg LSB;
    reg [3:0] temp_multiplicand;

    always @(posedge clk) begin
        if (load) begin
            temp_multiplicand <= multiplicand;
            result <= 0;
            new_multiplier <= multiplier;
            LSB <= 0;
        end else if (arth) begin
            if (new_multiplier[0] == 1 && LSB == 0)
                result <= result - temp_multiplicand;
            else if (new_multiplier[0] == 0 && LSB == 1)
                result <= result + temp_multiplicand;
        end else if (shiftRight) begin
            {result, new_multiplier, LSB} <= $signed({result, new_multiplier, LSB}) >>> booth_bits;
        end
    end

endmodule
```

شکل ۳: ساختار مدل مسیر داده

۳.۳ مدل واحد کنترل

در این مدل تمامی استیت‌های و حالت‌های ضرب کننده کنترل می‌شود. در اینجا تعدادی حالات کنونی داریم که براساس آنها حالات بعدی شکل می‌گیرد. همچنین یک counter داریم که در ابتدا با مقدار ۴ مقداردهی می‌شود و تعداد عملیات‌های را برای مدل نگه می‌دارد. همچنین در diffrences تمامی تغییرات بیت‌ها مشخص شده که براساس آن rightMost یا همان راست‌ترین بیتی که تغییر کرده است معین می‌شود و با دانستن این مقدار می توانیم بگوییم که چه مقدار می‌بایست شیف دهیم.

```
module BoothMultiplierControlUnit(
    input [3:0] multiplier,
    input clk, str,
    output valid, load, arth, shiftRight,
    output [2:0] booth_bits
);

    reg [3:0] current_state, next_state;
    reg [2:0] counter;

    assign load = current_state[0];
    assign arth = current_state[1];
    assign shiftRight = current_state[2];
    assign valid = current_state[3];

    always @(current_state, counter) begin
        next_state <= 0;
        if (current_state[0])
            next_state[1] <= 1'b1;
        if (current_state[1])
            next_state[2] <= 1'b1;
        if (current_state[2]) begin
            if (counter > booth_bits)
                next_state[1] <= 1'b1;
            else
                next_state[3] <= 1'b1;
        end
        if (current_state[3])
            next_state[3] <= 1'b1;
    end

```

شکل ۴: قسمت اول ساختار مدل کنترل واحد

```
    always @(posedge clk) begin
        if (str) begin
            current_state <= 1;
            counter <= 4;
        end else begin
            current_state <= next_state;
            if (current_state[2])
                counter <= counter - booth_bits;
            end
    end

    wire [3:0] differences = (multiplier ^ (multiplier >> 1)) | (1'b1 << (4 - 1'b1));
    reg [2:0] rightMost;

    integer i;
    always @(*) begin
        rightMost = 0;
        for (i = 1; i <= 4; i = i + 1) begin
            if (differences[i - 1] && !rightMost)
                rightMost = i;
        end
    end

    assign booth_bits = (counter > rightMost) ? rightMost : counter;

endmodule
```

شکل ۵: قسمت دوم ساختار مدل کنترل واحد

۴ شبیه‌سازی کارایی مدل

در این قسمت به مدل testBench و نتایج آن می‌پردازیم.

۱.۴ مدل تست

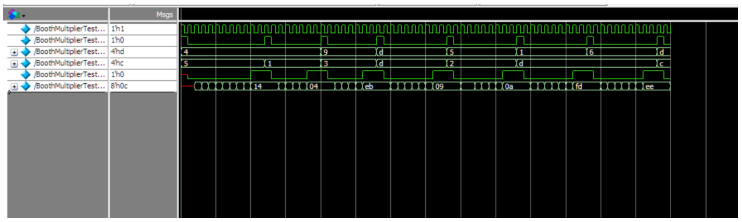
در این قسمت تست بنچی طراحی شده که هربار دو عدد رندوم را به عنوان ورودی گرفته و خروجی را بعد از مدتی می‌دهد. نکته‌ای که در اینجا حائز اهمیت است آن است که زمان خروجی معتبر است که سیگنال valid یک باشد.

```
module BoothMultiplierTestbench();

    reg clk, str;
    reg [3:0] multiplicand, multiplier;

    wire valid;
    wire [7:0] result;

    BoothMultiplier boothMultiplier (
        .multiplicand(multiplicand), .multiplier(multiplier),
        .valid(valid), .clk(clk), .str(str), .result(result)
    );

    initial
    clk = 1;

    always begin
        #5 clk = ~clk;
    end

    initial begin
        str <= 1;
        multiplicand = 4;
        multiplier = 5;
        #10;
        str <= 0;
    end

    always @(posedge valid) begin
        #20;
    str <= 1;
    multiplicand <= {$random};
    multiplier <= {$random};
        #10;
    str <= 0;
        $display("%d * %d = %d", multiplicand, multiplier, result);
    end

endmodule
```

شکل ۶: مدل تست برای ضرب کننده

۲.۴ نتایج شبیه‌سازی

در نهایت برای نتایج شبیه‌سازی با توجه به ورودی‌های رندوم چنین ویو فرمی را خواهیم داشت.



شکل ۷: نتایج مدل تست