

گزارش آزمایش پنجم آزمایشگاه طراحی سیستم دیجیتال



مزدک تیموریان

۴۰۱۱۰۱۴۹۵

مریم شیران

۴۰۰۱۰۹۴۴۶

مهدی بهرامیان

۴۰۱۱۷۱۵۹۳

دانشکده مهندسی کامپیوتر

دانشگاه صنعتی شریف

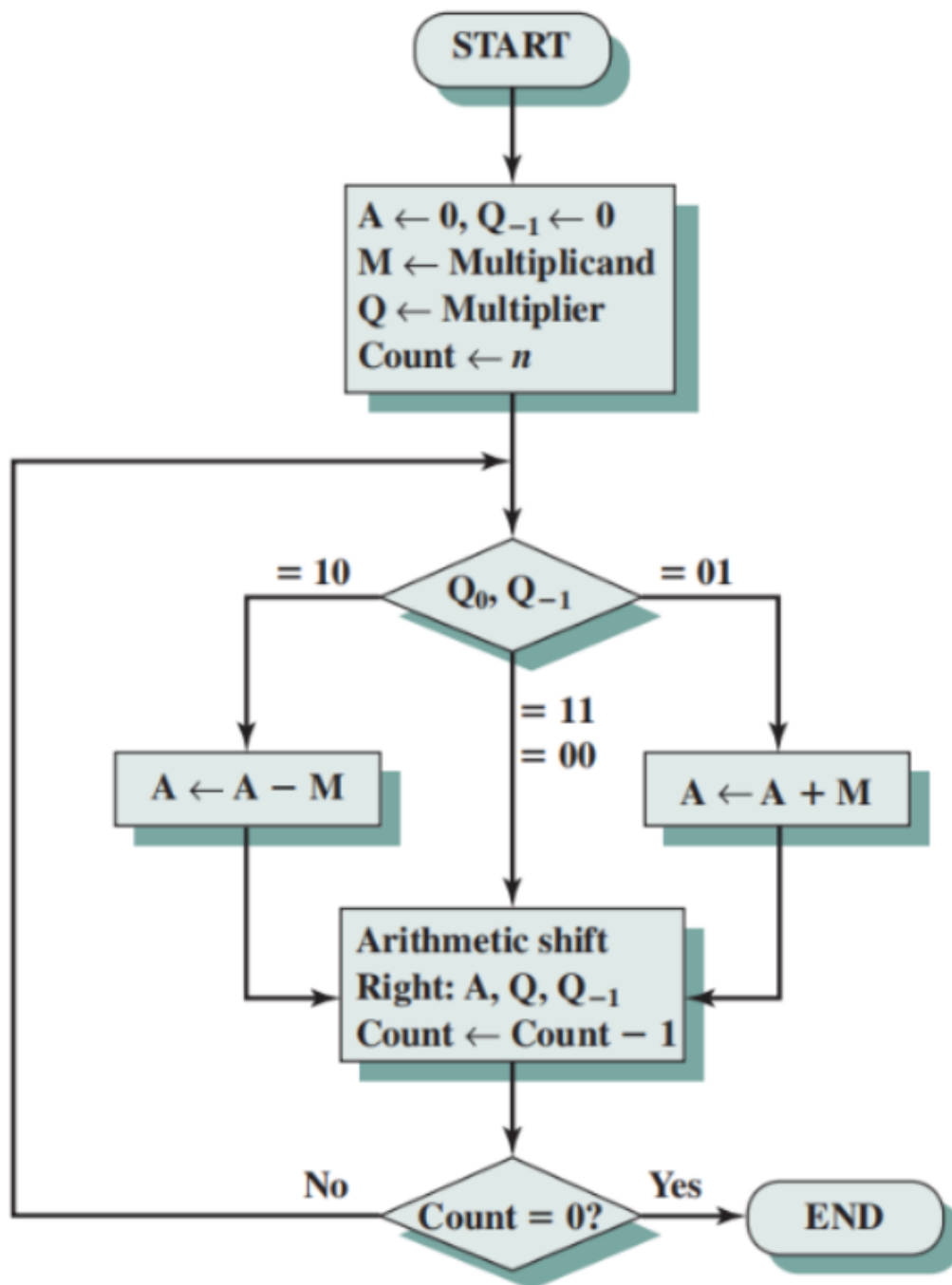
تیر ۱۴۰۳

فهرست مطالب

۲	۱	مقدمه
۳	۲	شرح آزمایش
۳	۱.۲	First Indices
۴	۲.۲	Datapath
۶	۳.۲	Control Unit
۸	۴.۲	Booth
۹	۳	شبیه سازی و بررسی عملکرد ضرب کننده

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 مسیره داده 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.

```
1 module First_indices (  
2     input [3:0] num,  
3     output [2:0] index0,  
4     output [2:0] index1  
5 );  
6     assign index0 =  
7     num[0] ? (num[1] ? (num[2] ? (num[3] ? 4 : 3) : 2) : 1) : 0;  
8     assign index1 =  
9     ~num[0] ? (~num[1] ? (~num[2] ? (~num[3] ? 4 : 3) : 2) : 1) : 0;  
10  
11 endmodule
```

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

```
1 module TB_FI ();  
2     reg [3:0] num;  
3     wire [1:0] index0;  
4     wire [1:0] index1;  
5  
6     First_indices fi (  
7         .num(num),  
8         .index0(index0),  
9         .index1(index1)  
10    );  
11  
12  
13    initial begin  
14        num = 12;  
15        #10;  
16        num = 15;  
17        #10  
18        num = 7;  
19        #10  
20        $stop();  
21    end  
22  
23    initial begin  
24        $monitor("number: %d", num, " index 0: %d", index0, " index 1: %d", index1);  
25    end  
26 endmodule
```

Datapath ۲.۲

در ادامه مسیر داده را طراحی میکنیم.

```
1  module Datapath (  
2      input clk,  
3      input rst,  
4      input done,  
5      input op,  
6      input [3:0] multiplier,  
7      input [3:0] multiplicand,  
8      input [2:0] shift_a,  
9      input [2:0] shift_b,  
10     output reg [7:0] result,  
11     output reg [3:0] B  
12 );  
13     reg [7:0] A;  
14  
15     always @(posedge clk, negedge rst) begin  
16         if (~rst) begin  
17             A <= {{4{multiplicand[3]}},multiplicand};  
18             B <= {{4{multiplier[3]}},multiplier};  
19             result <= 0;  
20         end  
21         else if (~done) begin  
22             B <= B >> shift_b;  
23             if (op) begin  
24                 result <= result + (A << shift_a);  
25             end  
26             else begin  
27                 result <= result - (A << shift_a);  
28             end  
29         end  
30     end  
31  
32 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.

```
1  module CU (  
2      input clk,  
3      input rst,  
4      output op,  
5      output done,  
6      input [3:0] B,  
7      output [2:0] shift_a,  
8      output [2:0] shift_b  
9  );  
10     wire [2:0] index0, index1;  
11     reg [2:0] total_shift;  
12     reg [2:0] start;  
13  
14     First_indices fi (  
15         .num(B),  
16         .index0(index0),  
17         .index1(index1)  
18     );  
19  
20     assign shift_a = total_shift + shift_b;  
21     assign shift_b = op ? index0 : index1;  
22  
23     assign op = start & B[0];  
24     assign done = shift_a >= 4;  
25  
26     always @(posedge clk, negedge rst) begin  
27         if (~rst) begin  
28             total_shift <= 0;  
29             start <= 0;  
30         end  
31         else begin  
32             start <= 1;  
33             total_shift <= total_shift + shift_b;  
34         end  
35     end  
36  
37 endmodule
```

همانطور که مشاهده میکنید در هر مرحله با استفاده از 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 مسیر داده و واحد کنترلی ماژول اصلی را طراحی میکنیم.

```
1  module Booth (  
2      input [3:0] multiplicand,  
3      input [3:0] multiplier,  
4      input clk,  
5      input rst,  
6      output done,  
7      output [7:0] result  
8  );  
9      wire [3:0] B;  
10     wire [2:0] shift_a, shift_b;  
11  
12     CU cu (  
13         .clk(clk),  
14         .rst(rst),  
15         .done(done),  
16         .op(op),  
17         .shift_a(shift_a),  
18         .shift_b(shift_b),  
19         .B(B)  
20     );  
21  
22     Datapath dp (  
23         .clk(clk),  
24         .rst(rst),  
25         .done(done),  
26         .op(op),  
27         .multiplicand(multiplicand),  
28         .multiplier(multiplier),  
29         .shift_a(shift_a),  
30         .shift_b(shift_b),  
31         .B(B),  
32         .result(result)  
33     );  
34  
35 endmodule
```

۳ شبیه‌سازی و بررسی عملکرد ضرب‌کننده

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

```
1  module TB_Booth ();
2      reg clk = 1;
3      reg rst = 1;
4      reg signed [3:0] multiplicand;
5      reg signed [3:0] multiplier;
6      wire done;
7      wire signed [7:0] result;
8
9      Booth booth (
10         .clk(clk),
11         .rst(rst),
12         .multiplicand(multiplicand),
13         .multiplier(multiplier),
14         .done(done),
15         .result(result)
16     );
17
18     always begin
19         #5 clk <= ~clk;
20     end
21
22     initial begin
23
24         multiplicand=5; multiplier=3;
25         rst=0;
26         #10 rst=1;
27         wait(done);
28
29         $display("time: %d multiplicand: %d multiplier: %d result: %d",
30             $time ,multiplicand, multiplier, result);
31
32         multiplicand=3; multiplier=5;
33         rst=0;
34         #10 rst=1;
35         wait(done);
36
37         $display("time: %d multiplicand: %d multiplier: %d result: %d",
```

```

40     multiplicand=8; multiplier=7;
41     rst=0;
42     #10 rst=1;
43     wait(done);
44
45     $display("time: %d multiplicand: %d multiplier: %d result: %d",
46     $time ,multiplicand, multiplier, result);
47
48     multiplicand=-7; multiplier=-5;
49     rst=0;
50     #10 rst=1;
51     wait(done);
52
53     $display("time: %d multiplicand: %d multiplier: %d result: %d",
54     $time ,multiplicand, multiplier, result);
55
56     multiplicand=3; multiplier=-6;
57     rst=0;
58     #10 rst=1;
59     wait(done);
60
61     $display("time: %d multiplicand: %d multiplier: %d result: %d",
62     $time ,multiplicand, multiplier, result);
63
64     multiplicand=6; multiplier=-1;
65     rst=0;
66     #10 rst=1;
67     wait(done);
68
69     $display("time: %d multiplicand: %d multiplier: %d result: %d",
70     $time ,multiplicand, multiplier, result);
71
72     multiplicand= 7; multiplier=1;
73     rst=0;
74     #10 rst=1;

```

```

69     $display("time: %d multiplicand: %d multiplier: %d result: %d",
70     $time ,multiplicand, multiplier, result);
71
72     multiplicand= 7; multiplier=1;
73     rst=0;
74     #10 rst=1;
75     wait(done);
76
77     $display("time: %d multiplicand: %d multiplier: %d result: %d",
78     $time ,multiplicand, multiplier, result);
79
80     multiplicand=0; multiplier=6;
81     rst=0;
82     #10 rst=1;
83     wait(done);
84
85     $display("time: %d multiplicand: %d multiplier: %d result: %d",
86     $time ,multiplicand, multiplier, result);
87
88     $stop();
89
90     end
91
92 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
```

