به نام خدا

پیش گزارش آزمایش پنجم

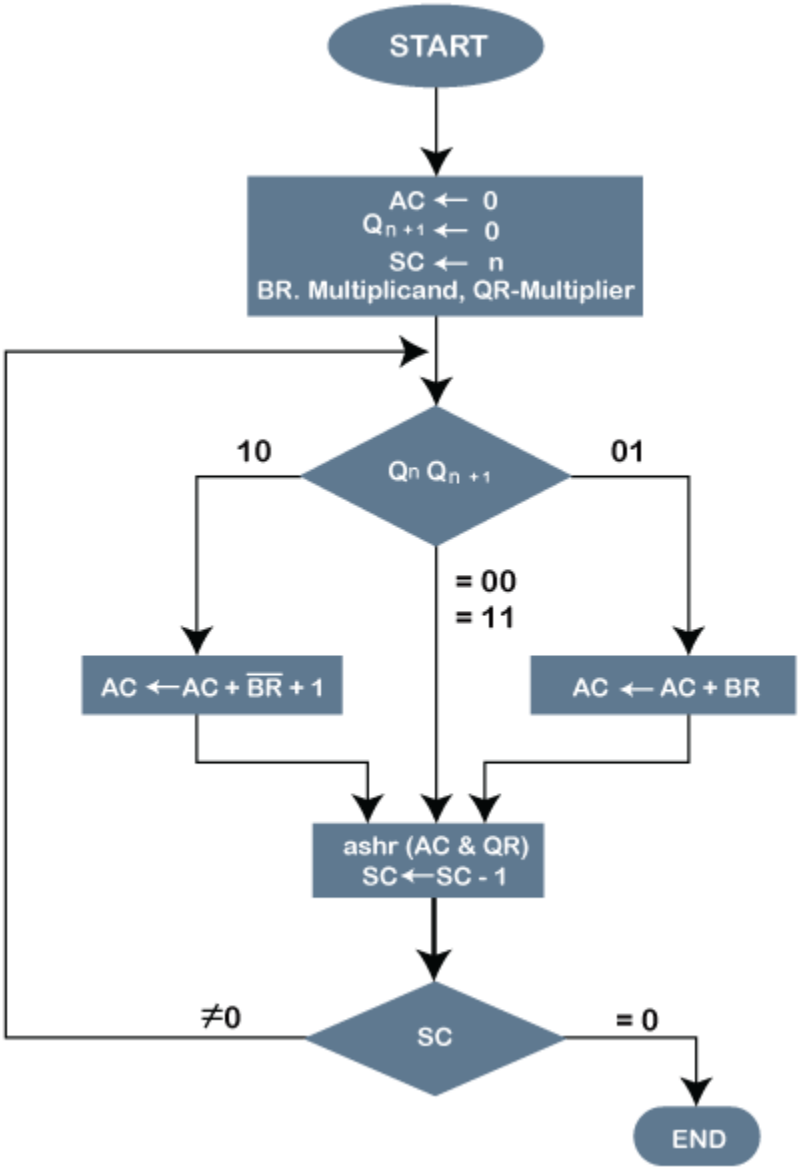
علیرضا سلیمیان: 400105036

امیر حسین علمدار:400105144

پیام تائبی:400104867

شرح آزمایش:

در این آزمایش هدف ما طراحی مدار ضرب کننده توسط الگوریتم booth است. در شکل زیر میتوانید asm chart این الگوریتم را مشاهده کنید.



در این آزمایش ما به این گونه عمل میکنیم که در ابتدا اولین یا صفر را پیدا میکنیم و بعد فرض میکنیم محاسبات از آنجا انجام میشود.

ورودی ها:

- A_in: ورودی اول با ۴ بیت یا مضروب.
- B_in : ورودی دوم با ۴ بیت یا مضروب فیه.
- resN
- clk

خروجی ها:

- res: حاصل ضرب خروجی ۸ بیتی.
- Done: با پایان یافتن محاسبات برابر یک میشود.

ماژول هایی که در این آزمایش مورد استفاده قرار گرفته است:

:First_one

در این ماژول ما اندیس کم ارزش ترین بیت از عدد که برابر یک است را پیدا میکنیم. برای محاسبه آن چون فقط ۴ بیت عدد داریم میتوانیم از جدول کارنو استفاده کنیم و نیاز به مدار پیچیده ای ندارد. در ادامه کد این ماژول آورده شده است.

```
module first_one(
    input [3:0] in,
    output [1:0] index
);

assign index = {~(in[1] | in[0]), ~in[0] & (in[1] | ~in[2])};

endmodule
```

:First_zero

در این ماژول اندیس کم ارزش ترین بیت از عدد که برابر صفر است را پیدا میکنیم. برای پیدا کردن این نیز میتوان از جدول کارنو کمک گرفت.

```
module first_zero(
    input [3:0] in,
    output [2:0] index
);

assign index[2] = in[3] & in[2] & in[1] & in[0];
assign index[1] = in[1] & in[0] & (~(in[3] & in[2]));
assign index[0] = in[0] & (~in[1] | in[2]);

endmodule
```

:Control_unit

در این ماژول که طبق خواسته سوال از ماژول datapath جدا شده است باید مقادیر کنترلی تولید شوند، مقدار مانند این که چه مقدار هر عدد شیفت داده بشود یا اینکه جمع شود یا تفریق یا اینکه آیا الگوریتم به پایان رسیده است یا خیر.

در op مقدار کم ارزش ترین بیت B قرار میگیرد و با توجه به ۰ یا ۱ بودن آن، به ترتیب دنبال اندیس کم ارزش ترین ۱ یا ۰ پس از آن میگردیم و آنرا در B_shift_amount قرار میدهیم. مقدار shifted یعنی تا اینجای مدار چه مقدار شیفت داده شده است.

برای اینکه مقدار A_shift_amount مشخص شود باید مقدار shifted را به اضافه B_shift_amount کرد و این برابر مقداری است که A برای جمع یا منها شدن با حاصل نهایی باید به چپ شیفت بخورد. البته اگر در کلاک اول باشیم مقدار op بدون توجه به کم ارزش ترین بیت B، برابر با صفر میشود.

اگر مقدار B_shift_amount+shifted بزگتر مساوی ۴ شود، یعنی با شیفت به راست دادن B به این مقدار، عملا تمامی بیت های B بررسی خواهند شد. پس محاسبات به اتمام رسیده است.

همچنین امکان ریست شدن نیز در مدار فراهم شده است و با فعال شدن آن مقدار shifted برابر صفر میشود و در کلاک اول پس از شروع محاسبات قرار میگیریم. در غیر این صورت و با بالا رفتن لبه کلاک، مقدارshifted با مقدار B_shift_amount جمع شده و دیگر در اولین کلاک قرار نداریم.

```
module control_unit(
    input [3:0] B,
    input rstN,
    input clk,
    output [2:0] A_shift_amount,
    output [2:0] B_shift_amount,
    output op,
    output done
);

 reg [2:0] shifted;
 reg first_clk;
 wire [1:0] one_index;
 wire [1:0] zero_index;

 first_one FO (B, one_index);
 first_zero FZ (B, zero_index);

 assign op = B[0] & (~first_clk);
 assign B_shift_amount = op ? zero_index : {1'b0, one_index};
 assign A_shift_amount = shifted + B_shift_amount;
 assign done = shifted + B_shift_amount > 3;

always @(posedge clk or negedge rstN) begin
    if(~rstN) begin
        shifted <= 0;
        first_clk <= 1;
    end else begin
        first_clk <= 0;
        shifted <= shifted + B_shift_amount;
    end
end

endmodule
```

:Datapath

در صورت فعال شدن rstN، مقدار ورودی های ۴ بیتی A_in و B_in به ترتیب در دو رجیستر ۸ بیتی Aو B قرار میگیرند که البته تفاوت آنها این است که در Aو B مقدار آنها sign extend شده است چراکه تمامی شیفت ها به صورت arithmetic انجام میشوند (البته میشود که در کد از <<< یا >>> استفاده شود، میدانیم که این عملگر ها مقدار شیفت arithmetic میدهند) و مقدار خروجی نیز برابر صفر میشود.

در غیر این صورت و با بالا رفتن لبه کلاک و تمام نشدن محاسبات، مقدار B به اندازه B_shift_amount به راست شیفت میخورد و مقدار خروجی متناسب با صفر یا یک بودن op، با مقدار A که به اندازه A_shift_amount به چپ شیفت خورده، منها یا جمع میشود.

```
module datapath(
    input [3:0] A_in,
    input [3:0] B_in,
    input       rstN,
    input       clk,
    input [2:0] A_shift_amount,
    input [2:0] B_shift_amount,
    input       op,
    input       done,
    output reg [7:0] acc,
    output reg [3:0] B
);

 reg [7:0] A;

always @(posedge clk or negedge rstN) begin
    if(~rstN) begin
        A <= {{4{A_in[3]}}, A_in};
        B <= {{4{B_in[3]}}, B_in};
        acc <= 0;
    end else if (~done) begin
        B <= B >> B_shift_amount;
        acc <= acc + (op ? 1 : -1) * (A << A_shift_amount);
    end
end

 endmodule
```

:Booth

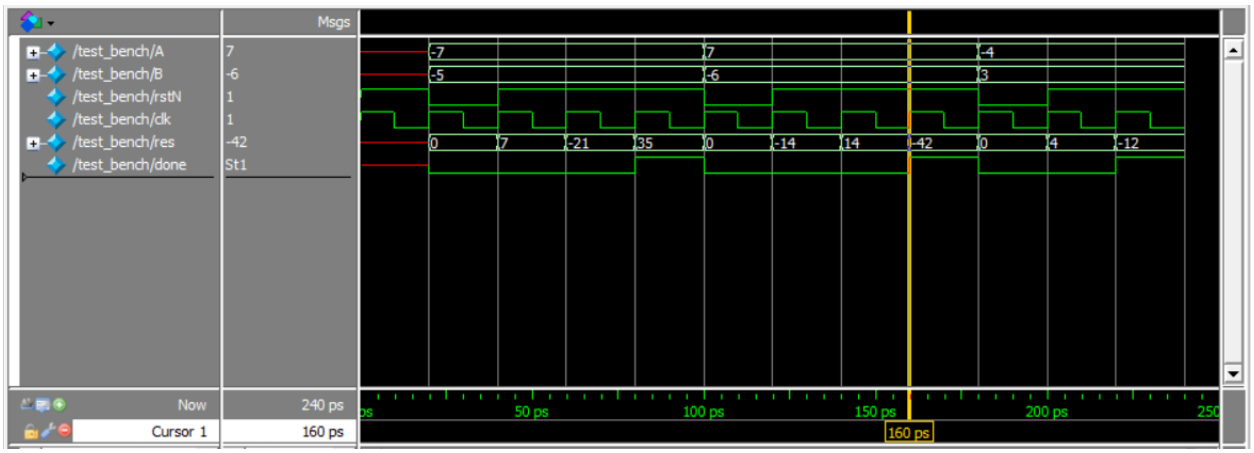
این ماژول در حقیقت بخش اصلی ماجرا است و جایی است که دو ماژول بالا را به هم مرتبط میکند. از کدام از ماژول های بالا یک instance ساخته میشود. از instance ماژول control_unit مقادیر A_shift_amount و B_shift_amount خروجی گرفته میشود و از instance ماژول datapath نیز مقادیر res و B خروجی گرفته میشود که این دو ماژول به یکدیگر وابسته هستند.

```
module booth(
    input [3:0] A_in,
    input [3:0] B_in,
    input rstN,
    input clk,
    output [7:0] res,
    output done
);

wire [2:0] A_shift_amount;
wire [2:0] B_shift_amount;
wire [3:0] B;

control_unit CU(B, rstN, clk, A_shift_amount, B_shift_amount, op, done);
datapath DP (A_in, B_in, rstN, clk, A_shift_amount, B_shift_amount, op, done, res, B);

endmodule
```

در ادامه نیز یک تست بنچ برای تست مدار آورده شده است:

```
module test_bench();


reg signed [3:0] A;
reg signed [3:0] B;
reg rstN = 1, clk = 1;
wire signed [7:0] res;
wire done;

booth mul(A, B, rstN, clk, res, done);

always #10 clk = ~clk;

initial begin
    $monitor("res: %b", res);

    #20
    A = -7;
    B = -5;
    rstN = 0;
    #20 rstN = 1;
    wait (done);

    $display("$d * %d = %d", A, B, res);

    #20
    A = 7;
    B = -6;
    rstN = 0;
    #20 rstN = 1;
    wait (done);

    $display("%d * %d = %d", A, B, res);

    #20
    A = -4;
    B = 3;
    rstN = 0;
    #20 rstN = 1;
    wait (done);

    $display("%d * %d = %d", A, B, res);

    #20;

    $stop;
end

endmodule
```

نتیجه مدار را میتوانید در شکل زیر مشاهده کنید:




```
# res: xxxxxxxx
# res: 00000000
# res: 00000111
# res: 11101011
# $d *  -7 =  -5  35
# res: 00100011
# res: 00000000
# res: 11110010
# res: 00001110
#   7 *  -6 =  -42
# res: 11010110
# res: 00000000
# res: 00000100
#  -4 *   3 =  -12
# res: 11110100
```