
# گزارش آزمایش پنجم طراحی سیستم‌های دیجیتال

## گروه شش

اعضا:

احمد سلیمی

همیلا میلی

درنا دهقانی

# شرح آزمایش

هدف این آزمایش طراحی ضرب‌کننده‌ایست که به کمک روش Booth، حاصل‌ضرب دو ورودی را محاسبه کند.

ورودی‌ها:

- A_in: ورودی اول ۴ بیتی یا مضروب.
- B_in: ورودی دوم ۴ بیتی یا مضروب‌فیه.
- rstN
- clk

خروجی‌ها:

- res: حاصل‌ضرب خروجی ۸ بیتی.
- Done: با پایان محاسبه، برابر با ۱ می‌شود.

# Booth Algorithm

به طور کلی در این الگوریتم، با رسیدن به ۱ در مضروب‌فیه، مقدار مضروب را از حاصل کم می‌کنیم و با رسیدن به ۰ در مضروب‌فیه، مقدار مضروب را به حاصل اضافه می‌کنیم و سپس حاصل را به راست شیفت می‌دهیم و در صورتی‌که رقم فعلی و پیشین مضروب‌فیه یکسان بود، حاصل را فقط شیفت می‌دهیم. به عنوان مثال، حاصلضرب ۶ و 3- که توسط این وب‌سایت محاسبه شده را بررسی می‌کنیم:

| | | |
|---|---|---|
| A | 0 1 1 0 | 6 |
| X | x 1 1 0 1 | -3 |
| Y | 0 -1 1 -1 | recoded multiplier |
| Add -A | + 1 0 1 0 | |
| Shift | 1 1 0 1 0 | |
| Add A | + 0 1 1 0 | |
| | 0 0 1 1 0 | |
| Shift | 0 0 0 1 1 0 | |
| Add -A | + 1 0 1 0 | |
| | 1 0 1 1 1 0 | |
| Shift | 1 1 0 1 1 1 0 | |
| Shift Only | 1 1 1 0 1 1 1 0 | -18 |

اما ما در این پیاده‌سازی، اندکی متفاوت عمل می‌کنیم. با برخورد به هر رقم، محاسبات آن (جمع یا تفریق) در مرحله قبلی انجام شده است. پس فقط شیفت می‌دهیم و محاسبات مربوط به رقم بعدی را انجام می‌دهیم.

برای انجام چندین شیفت در یک پالس ساعت، تنها کافیست در هر مرحله‌ای که باید محاسبات انجام شود (یعنی در مضروب‌فیه بیت قبلی ۰ و بیت فعلی ۱ باشد، یا برعکس)، اندیس اولین بیت ۰ یا ۱ پس از آن را بیابیم.

## ماژول‌ها

first_one: در این ماژول اندیس اولین (کم‌ارزش‌ترین) بیت از عدد که برابر با ۱ است را می‌یابیم. محاسبه آن به کمک جدول کارنو انجام می‌گیرد. کد وریلاگ آن به شرح زیر 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 اولین (کم‌ارزش‌ترین) بیت از عدد که برابر با 0 است را می‌یابیم. محاسبه آن به کمک جدول کارنو انجام می‌گیرد. کد وریلاگ آن به شرح زیر است:

```
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 مقادیر لازم برای شیفت دادن دو عدد A و B و هم چنین پایان یافتن ضرب مشخص می‌شود. در op مقدار کم‌ارزش‌ترین بیت B قرار می‌گیرد و با توجه به ۰ یا ۱ بودن مقدار آن، به ترتیب اندیس کم‌ارزش‌ترین بیت ۱ یا ۰ پس از آن در B_shft_amt قرار می‌گیرد. مقدار shifted که برابر با مقدار شیفت خوردن B تا کلاک قبلیست نیز با B_shft_amt جمع زده می‌شود تا A_shft_amt مشخص شود، یعنی مقداری 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 کم‌ارزش‌ترین بیت B برابر با ۰ می‌شود و در حاصل نهایی منها رخ می‌دهد.

اگر مقدار B_shft_amt + shifted برابر با ۴ یا بیشتر شود، یعنی با شیفت به راست دادن B به این مقدار، عملاً تمامی بیت‌های B بررسی شده است. پس محاسبات به اتمام رسیده است.

برای بخش ترتیبی مدار، در صورت ریست شدن، مقدار shifted برابر با صفر شده و در کلاک اول پس از شروع محاسبات قرار می‌گیریم. در غیر اینصورت و با بالا رفته لبه کلاک، مقدار shifted با B_shft_amt جمع شده و دیگر در اولین کلاک قرار نداریم.

کد وریلاگ آن به شرح زیر است:

```
module control_unit (
    input   [3:0]   B,
    input           rstN,
    input           clk,
    output  [2:0]   A_shft_amt,
    output  [2:0]   B_shft_amt,
    output          op,     // 0: subtract, 1: add
    output          done
);

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

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

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

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

endmodule
```

Datapath: چون مدار ترتیبی‌ست، از always block استفاده می‌کنیم که به لبه‌ی مثبت کلاک یا لبه‌ی منفی rstN حساس است. در صورت ریست شدن، مقدار ورودی‌های ۴ بیتی A_in و B_in به ترتیب در دو رجیستر ۸ بیتی A و B که از سمت بیت پرارزش extend شده‌اند، منتقل می‌شوند، چون تمامی شیفت به راست‌ها arithmetic انجام می‌گیرند. مقدار خروجی نیز برابر با ۰ می‌شود.

در غیر این صورت و با بالا رفتن لبه کلاک و پایان نپذیرفتن محاسبات، مقدار B به اندازه‌ی B_shft_amt به راست شیفت می‌خورد و مقدار خروجی متناسب با ۰ یا ۱ بودن op با مدار A که به اندازه A_shtf_amt به چپ شیفت خورده، منها یا جمع می‌شود.

کد وریلاگ آن به شرح زیر است:

```
module datapath (
    input       [3:0]   A_in,
    input       [3:0]   B_in,
    input               rstN,
    input               clk,
    input       [2:0]   A_shft_amt,
    input       [2:0]   B_shft_amt,
    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_shft_amt;
        ACC <= ACC + (op ? 1 : -1) * (A << A_shft_amt);
    end
end

endmodule
```

Booth: این ماژول، ماژول اصلی آزمایش است. یک instance از ماژول control_unit در آن ساخته می‌شود که مقادیر A_shft_amt و B_shft_amt و done و op خروجی‌های آن هستند. یک instance هم از ماژول datapath ساخته می‌شود که مقادیر res و B را خروجی می‌دهد. این دو ماژول به هم وابسته‌اند. کد وریلاگ آن به شرح زیر است:

```
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_shft_amt;
wire [2:0]  B_shft_amt;
wire [3:0]  B;

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

endmodule
```

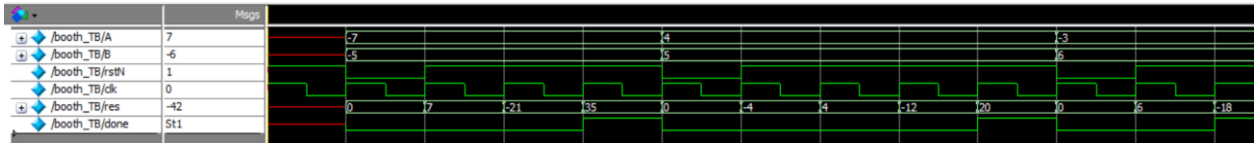
Booth_tb: کارکرد صحیح ماژول booth توسط این ماژول بررسی می‌شود و کد وریلاگ آن به شرح زیر است:

```
module booth_TB ();

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 = 4;
    B = 5;
    rstN = 0;
    #20 rstN = 1;
    wait (done);

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

    #20;
    A = -3;
    B = 6;
    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;

    $stop;
end

endmodule
```

# Waveform

حاصل شبیه‌سازی test bench این آزمایش به شکل زیر است:



```
# res: xxxxxxxx
# res: 00000000
# res: 00000111
# res: 11101011
#  -7 *  -5 =   35
# res: 00100011
# res: 00000000
# res: 11111100
# res: 00000100
# res: 11110100
#   4 *   5 =   20
# res: 00010100
# res: 00000000
# res: 00000110
#  -3 *   6 =  -18
# res: 11101110
# res: 00000000
# res: 11110010
# res: 00001110
#   7 *  -6 =  -42
# res: 11010110
```