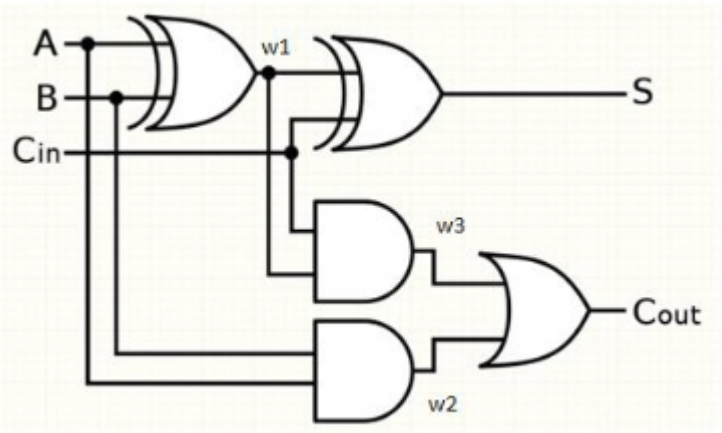
📝 گزارش کار تمرین ۳

طراحی سیستم‌های دیجیتال

> *Alireza Habibzadeh* `99109393`
> *May 7, 2022*

الف

باید مدار زیر را در وریلاگ به صورت ساختاری پیاده‌سازی کنیم.



full adder

برای این کار ماژول `full_adder` را با ورودی و خروجی‌هایش تعریف می‌کنیم. سپس wireهای `w1` و `w2` و `w3` را تعریف کرده و از آن‌ها در گیت‌ها استفاده می‌کنیم. کد نهایی توصیف ساختاری ماژول `full_adder` در زیر آمده است.

```
module full_adder(
    output s,
    output c_out,
    input c_in,
    input a,
    input b
    );

    wire w1, w2, w3;
```

```
    xor(w1, a, b);
    xor(s, w1, c_in);

    and(w3, w1, c_in);
    and(w2, a, b);
    or(c_out, w3, w2);
endmodule
```

حال کافی است در ماژول `carry_ripple_adder` خود از این ماژول ۴ بار instance بگیریم:

```
module carry_ripple_adder(
    output [3:0] S,
    output c_out,
    input c_in,
    input [3:0] A,
    input [3:0] B
    );

    wire [2:0] C;
    full_adder fa0(S[0], C[0], c_in, A[0], B[0]);
    full_adder fa1(S[0], C[1], C[0], A[1], B[1]);
    full_adder fa2(S[0], C[2], C[1], A[2], B[2]);
    full_adder fa3(S[0], c_out, C[2], A[3], B[3]);

endmodule
```

ب

برای نوشتن testbenchای که بتواند همه‌ی حالت‌های ما را امتحان کند باید از حلقه استفاده کنیم.

```
`timescale 1ns / 1ps

module carry_ripple_adder_tb;
    // Inputs
```

```
    reg c_in;
    reg [3:0] A;
    reg [3:0] B;
    // Outputs
    wire [3:0] S;
    wire c_out;
    // Instantiate the Unit Under Test (UUT)
    carry_ripple_adder uut (
        .S(S),
        .c_out(c_out),
        .c_in(c_in),
        .A(A),
        .B(B)
    );

    initial begin
        // Initialize Inputs
        c_in = 0;
        A = 0;
        B = 0;

        // Wait 100 ns for global reset to finish
        #100;

        // Add stimulus here
        for (int i=0; i<16; i=i+1) begin
            for (int j=0; j<16; j=j+1) begin
                c_in = 0;
                A = i;
                B = j;
                if (S == i + j) begin
```

```
                    $display("Test passed, %d + %d = %d", i, j, S);
                end else begin
                    $display("Test failed, %d + %d != %d", i, j, S);
                end

                c_in = 1;
                if (S == i + j + 1) begin
                    $display("Test passed, %d + %d + 1 = %d", i, j, S);
                end else begin
                    $display("Test failed, %d + %d + 1 != %d", i, j, S);
                end
            end
        end
    end
endmodule
```

پ

تاخیر `carry` یک full adder برابر با تاخیر یک لایه xor به علاوه‌ی یک لایه and و یک لایه or است.
پس تاخیر کل یک full adder برابر با ۸ در واحد سوال است.
پس نهایتا برای آن که از `carry` خروجی full adder آخر مطمئن باشیم باید $(4 + 2 + 2) * 4 = 32$ واحد زمانی منتظر بمانیم. پس سیگنال `ready` را طوری تعریف می‌کنیم که پس از این مدت روشن شود.

برای اعمال تاخیرها در گیت‌ها از روش زیر استفاده می‌کنیم:

```
and #(delay_time) a1(out, i1, i2);
```

پس با اعمال تاخیرها کد `full_adder_delay` می‌شود:

```
module carry_ripple_adder_ready(
    output [3:0] S,
```

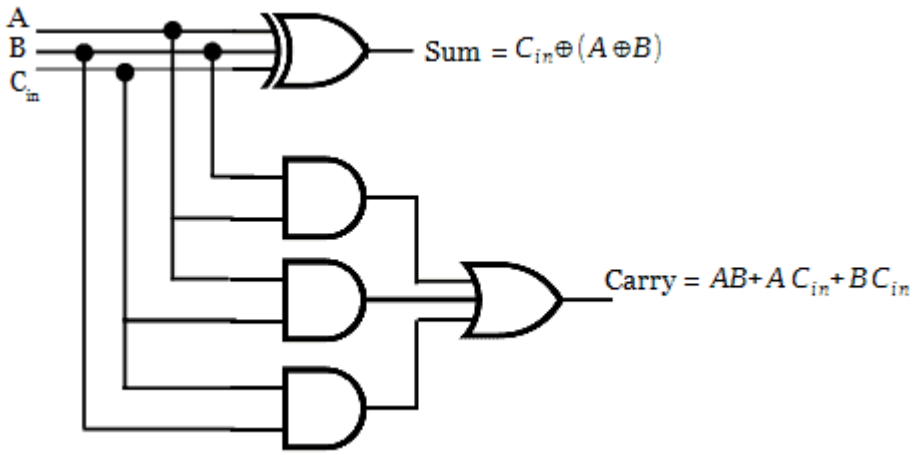
```
    output c_out,
    output ready,
    input c_in,
    input [3:0] A,
    input [3:0] B
    );

    wire [2:0] C;
    full_adder_delay fa0(S[0], C[0], c_in, A[0], B[0]);
    full_adder_delay fa1(S[1], C[1], C[0], A[1], B[1]);
    full_adder_delay fa2(S[2], C[2], C[1], A[2], B[2]);
    full_adder_delay fa3(S[3], c_out, C[2], A[3], B[3]);

    always @(A or B)begin
        ready = 1'b0;
        #32 ready = 1'b1;
    end
endmodule
```

بلاک `always @(A or B` با تغییر هر کدام از سیگنال‌های `A` و `B` اجرا می‌شود. ابتدا سیگنال `ready` را خاموش می‌کند، سپس به مقدار مورد نیاز (۳۲ واحد زمانی) صبر می‌کند و سپس سیگنال را روشن می‌کند. در این مدت زمان حتما سیگنال‌های خروجی به مقدار درست پایای خود رسیده‌اند.

ت



faster full adder

برای سریع‌تر شدن مدار می‌توان از طراحی بالا برای full adder استفاده کرد. تاخیر `carry` در این جمع کننده برابر با ۴ واحد زمانی است. چرا که خروجی تنها دو لایه‌ی and و or (یا یک لایه‌ی xor) با ورودی فاصله دارد.

چون تنها جمع‌کننده را تغییر دادیم، نیازی به تغییر کد `carry_ripple_adder_ready` نیست. تنها باید کد full adder را عوض کنیم: (البته اینجا برای تحویل کدها از اسم متفاوتی برای ماژول‌ها استفاده کردم. در این صورت باید اسم full adderها را در ماژول اصلی عوض کنیم.)

```
module full_adder_delay_fast(
    output s,
    output c_out,
    input c_in,
    input a,
    input b
    );

    wire w1, w2, w3;
    xor #4 (s, a, b, c_in);

    and #2 (w1, a, b);
    and #2 (w2, a, c_in);
         and #2 (w3, b, c_in);
    or #2 (c_out, w1, w2, w3);
endmodule
```

ث

پس از این تغییر ۴ جمع‌کننده داریم که هر کدام نهایتا ۴ واحد زمانی برای تولید carry خود تاخیر دارند. پس carry نهایی (و در این حالت خاص خود بیت نهایی جمع هم) در $(2+2)*4=16$ واحد زمانی تولید می‌شوند. پس تاخیر ۱۶ (نصف قبلی) را باید برای این سیگنال در نظر گرفت.

```
module carry_ripple_adder_ready(
    output [3:0] S,
    output c_out,
    output ready,
    input c_in,
    input [3:0] A,
    input [3:0] B
    );

    wire [2:0] C;
    full_adder_delay_fast fa0(S[0], C[0], c_in, A[0], B[0]);
    full_adder_delay_fast fa1(S[1], C[1], C[0], A[1], B[1]);
    full_adder_delay_fast fa2(S[2], C[2], C[1], A[2], B[2]);
    full_adder_delay_fast fa3(S[3], c_out, C[2], A[3], B[3]);

    always @(A or B)begin
        ready = 1'b0;
        #16 ready = 1'b1;
    end
endmodule
```