میان‌ترم DSD

سوال 4

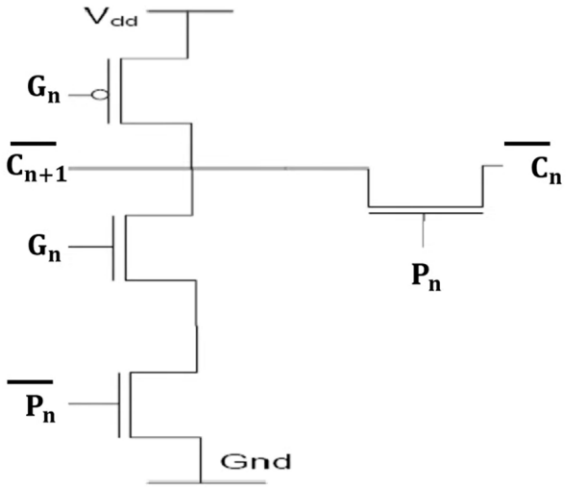
متین باقری **402105727**

جمع کننده ای که برای انتشار رقم نقلی از زنجیره ی Manchester استفاده می‌کند، در یک FA برای تولید رقم نقلی چنین ساختاری دارد:

Static Manchester Carry Chain

A	B	Cin	Cout	P	G
0	0	0	0	0	0
0	0	1	0	0	0
1	0	0	0	1	0
0	1	0	0	1	0
1	0	1	1	1	0
0	1	1	1	1	0
1	1	0	1	0	1
1	1	1	1	0	1

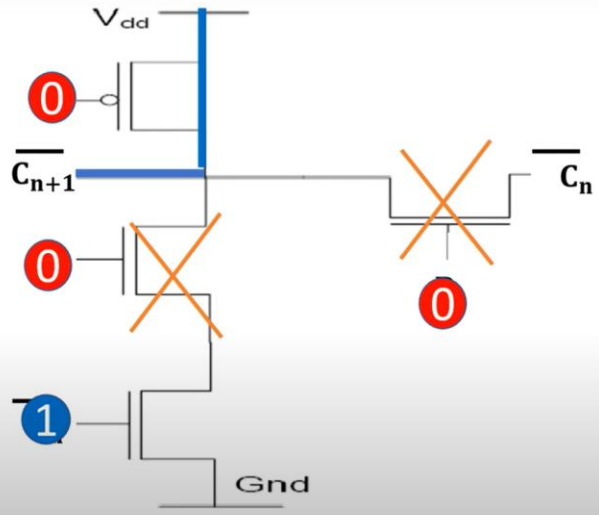
$$C_{out} = A * B + (A \oplus B) * C_{in}$$
$$C_{out} = G + P * C_{in}$$
$$C_{n+1} = G_n + P_n * C_n$$



از ۴ ترانزیستور استفاده می‌کند و حالات مختلف خروجی را به این صورت تولید می‌کند:

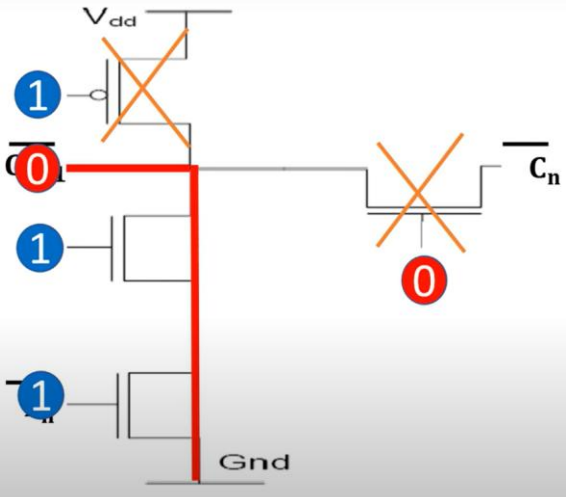
A	B	Cin	Cout	P	G
0	0	0	0	0	0
0	0	1	0	0	0
1	0	0	0	1	0
0	1	0	0	1	0
1	0	1	1	1	0
0	1	1	1	1	0
1	1	0	1	0	1
1	1	1	1	0	1

$$C_{out} = A * B + (A \oplus B) * C_{in}$$
$$C_{out} = G + P * C_{in}$$
$$C_{n+1} = G_n + P_n * C_n$$



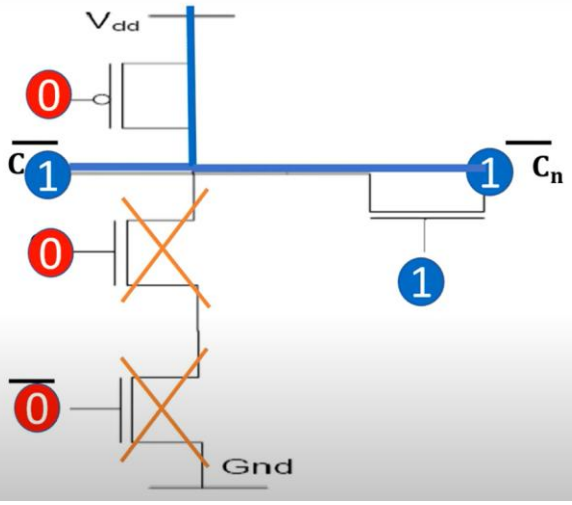
A	B	Cin	Cout	P	G
0	0	0	0	0	0
0	0	1	0	0	0
1	0	0	0	1	0
0	1	0	0	1	0
1	0	1	1	1	0
0	1	1	1	1	0
1	1	0	1	0	1
1	1	1	1	0	1

$$C_{out} = A * B + (A \oplus B) * C_{in}$$
$$C_{out} = G + P * C_{in}$$
$$C_{n+1} = G_n + P_n * C_n$$



A	B	Cin	Cout	P	G
0	0	0	0	0	0
0	0	1	0	0	0
1	0	0	0	1	0
0	1	0	0	1	0
1	0	1	1	1	0
0	1	1	1	1	0
1	1	0	1	0	1
1	1	1	1	0	1

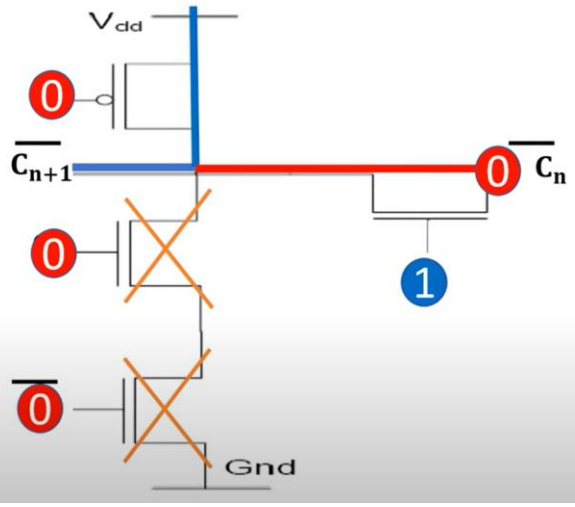
$$C_{out} = A * B + (A \oplus B) * C_{in}$$
$$C_{out} = G + P * C_{in}$$
$$C_{n+1} = G_n + P_n * C_n$$



تا اینجا رقم نقلی بعدی تنها به یک ورودی متصل است و به درستی به دست می‌آید. اما در حالت بعدی، به دو ورودی متفاوت متصل است و در عمل، به کمک تنظیم مقاومت ترانزیستورها، خروجی عددی نزدیک به صفر ولت به دست آمده و صفر در نظر گرفته می‌شود:

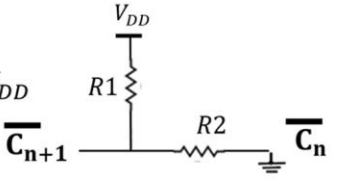
A	B	Cin	Cout	P	G
0	0	0	0	0	0
0	0	1	0	0	0
1	0	0	0	1	0
0	1	0	0	1	0
1	0	1	1	1	0
0	1	1	1	1	0
1	1	0	1	0	1
1	1	1	1	0	1

$$C_{out} = A * B + (A \oplus B) * C_{in}$$
$$C_{out} = G + P * C_{in}$$
$$C_{n+1} = G_n + P_n * C_n$$



A	B	Cin	Cout	P	G
0	0	0	0	0	0
0	0	1	0	0	0
1	0	0	0	1	0
0	1	0	0	1	0
1	0	1	1	1	0
0	1	1	1	1	0
1	1	0	1	0	1
1	1	1	1	0	1

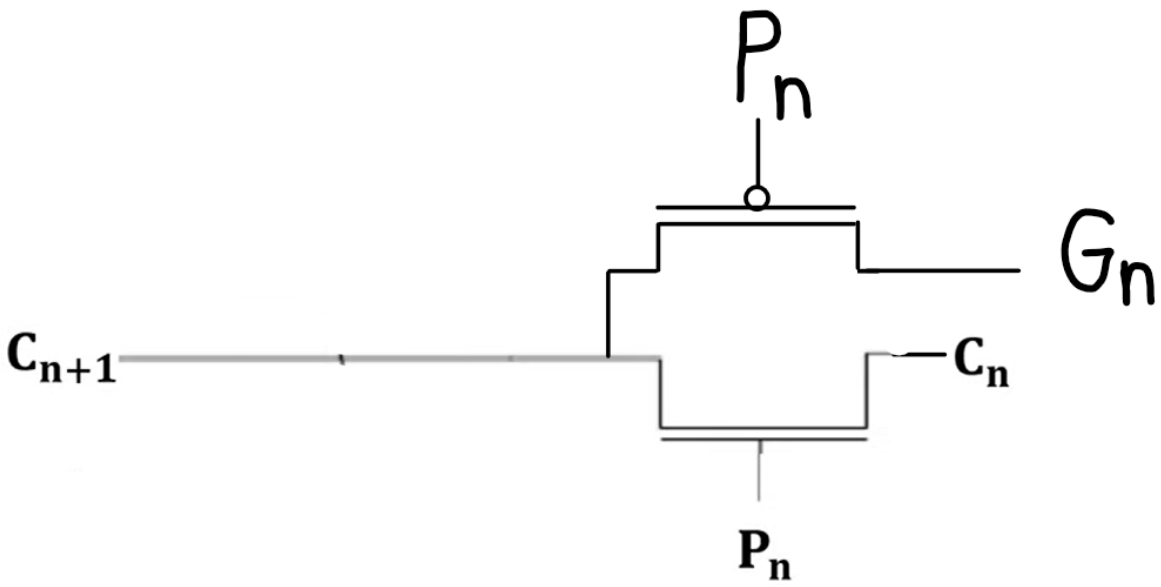
$$V_{n+1} = \frac{R_2}{R_2+R_1} V_{DD}$$



$$R_2 << R_1 \quad V_{n+1} = \frac{R_2}{R_1} V_{DD}$$

اما از آنجایی که در Verilog نمی‌توان برای ترانزیستور های nmos و pmos مقاومت اهمی تعریف کرد، اجرای این پیاده سازی ممکن نبود.

با حفظ ذهنیت Manchester carry chain و بررسی حالت‌های مختلف جدول حالات FA که در تصاویر نیز موجود است، به طراحی زیر رسیدیم:



با استفاده از یک nmos و یک pmos رقم نقلی بعدی را محاسبه کرده و sum را به این صورت به دست آوردیم:

```
xor (Sum[i], p[i], carry[i]); // Sum is P XOR Carry-in
```

نهایتا Carry خروجی را برابر با آخرین رقم نقلی تولید شده در زنجیره قرار دادیم:

```
buf (Cout, carry[16]);
```

در نوشتن کد جمع کننده، تماما از گیت های مختلف استفاده کردیم تا کد ساختاری باشد:

```
module manchester_carry_chain_adder (
    input [15:0] a, b,
    input Cin,
    output [15:0] Sum,
    output Cout
);
    wire [15:0] p, g;
    wire [16:0] carry;

    // Cin is the initial carry
    buf (carry[0], Cin);

    genvar i;
    generate
        for (i = 0; i < 16; i = i + 1) begin: stage

            // Propagate and Generate signals
            xor (p[i], a[i], b[i]);
```

```
            and (g[i], a[i], b[i]);
            // Sum generation
            xor (Sum[i], p[i], carry[i]); // Sum is P XOR Carry-in

            nmos(carry[i+1], carry[i], p[i]);
            pmos(carry[i+1], g[i], p[i]);

        end
    endgenerate

    buf (Cout, carry[16]);
endmodule
```

در تست بنچ ۵۰ ورودی رندوم و ۱۰ ورودی edge case داریم تا صحت عملکرد مدار را بررسی کنیم. توضیحات بیشتر در کامنت‌های کد قابل مشاهده اند:

```
`timescale 1ns/1ps

module testbench;
    reg [15:0] a, b;
    reg Cin;
    wire [15:0] Sum;
    wire Cout;

    // Instantiate
    manchester_carry_chain_adder dut (.a(a), .b(b), .Cin(Cin), .Sum(Sum),
.Cout(Cout));

    reg [16:0] expected_sum; // 17 bits to include carry out
    integer random_correct = 0;
    integer edge_correct = 0;
    integer total_tests = 0;
    integer i;

    // Test task
    task test_case;
        input [15:0] a_val, b_val;
        input cin_val;
        input is_edge_case;
        begin
            a = a_val;
            b = b_val;
            Cin = cin_val;
            expected_sum = a + b + Cin;

            #1; // Allow time for signals to propagate

            total_tests = total_tests + 1;
```

```
            if ({Cout, Sum} === expected_sum) begin
                if (is_edge_case) edge_correct = edge_correct + 1;
                else random_correct = random_correct + 1;

                $display("Test %0d: PASS - %h + %h + %b = %h (Cout=%b) [Expected:
%h]",
                         total_tests, a, b, Cin, Sum, Cout, expected_sum);
            end
            else begin
                $display("Test %0d: FAIL - %h + %h + %b = %h (Cout=%b) [Expected:
%h]",
                         total_tests, a, b, Cin, Sum, Cout, expected_sum);
            end
        end
    endtask

    // Main test
    initial begin
        $display("Starting Manchester Carry Chain Adder Test Bench");
        $display("-------------------------------------------------");

        $display("\nRunning 50 Random Test Cases:");
        // Random test cases
        for (i = 0; i < 50; i = i + 1) begin
            test_case($random, $random, $random % 2, 0);
        end
        $display("\nRandom Test Cases Summary:");
        $display("Correct: %0d/50", random_correct);

        $display("\nRunning 10 Edge Cases:");

        test_case(16'h0000, 16'h0000, 1'b0, 1);// Edge Case 1: All zeros

        test_case(16'hFFFF, 16'hFFFF, 1'b0, 1);// Edge Case 2: All ones

        test_case(16'hFFFF, 16'hFFFF, 1'b1, 1);// Edge Case 3: All ones with
carry in

        test_case(16'hAAAA, 16'h5555, 1'b0, 1);// Edge Case 4: Carry propagation
(alternating bits)

        test_case(16'hAAAA, 16'h5555, 1'b1, 1);// Edge Case 5: Carry propagation
with carry in

        test_case(16'h0001, 16'h0001, 1'b0, 1);// Edge Case 6: Single bit
addition

        test_case(16'h0001, 16'hFFFE, 1'b1, 1);// Edge Case 7: Maximum carry
chain

        test_case(16'h1234, 16'h4321, 1'b0, 1);// Edge Case 8: No carry
generation
```

```
        test_case(16'hA5A5, 16'h5A5A, 1'b1, 1);// Edge Case 9: Random pattern
with carry in

        test_case(16'hF0F0, 16'h0F0F, 1'b0, 1);// Edge Case 10: Mixed pattern

        $display("\nEdge Cases Summary:");
        $display("Correct: %0d/10", edge_correct);

        // Total summary
        $display("\nFinal Summary:");
        $display("Random Tests: %0d/50 correct", random_correct);
        $display("Edge Cases:   %0d/10 correct", edge_correct);
        $display("Total:        %0d/60 correct", random_correct + edge_correct);
        $display("Accuracy:     %.2f%%", ((random_correct + edge_correct) *
100.0) / 60);

        $stop;
    end
endmodule
```

خروجی اجرای تست بنچ به این صورت است:

```
#Starting Manchester Carry Chain Adder Test Bench
# ------------------------------------------------
#
# Running 50 Random Test Cases:
# Test 1: PASS - 3524 + 5e81 + 1 = 93a6 (Cout=0) [Expected: 093a6]
# Test 2: PASS - 5663 + 7b0d + 1 = d171 (Cout=0) [Expected: 0d171]
        …       …       …       …       …
# Test 49: PASS - bf94 + 5d93 + 0 = 1d27 (Cout=1) [Expected: 11d27]
# Test 50: PASS - ca59 + 69db + 1 = 3435 (Cout=1) [Expected: 13435]
#
# Random Test Cases Summary:
# Correct: 50/50
#
# Running 10 Edge Cases:
# Test 51: PASS - 0000 + 0000 + 0 = 0000 (Cout=0) [Expected: 00000]
# Test 52: PASS - ffff + ffff + 0 = fffe (Cout=1) [Expected: 1fffe]
# Test 53: PASS - ffff + ffff + 1 = ffff (Cout=1) [Expected: 1ffff]
# Test 54: PASS - aaaa + 5555 + 0 = ffff (Cout=0) [Expected: 0ffff]
# Test 55: PASS - aaaa + 5555 + 1 = 0000 (Cout=1) [Expected: 10000]
# Test 56: PASS - 0001 + 0001 + 0 = 0002 (Cout=0) [Expected: 00002]
# Test 57: PASS - 0001 + fffe + 1 = 0000 (Cout=1) [Expected: 10000]
# Test 58: PASS - 1234 + 4321 + 0 = 5555 (Cout=0) [Expected: 05555]
# Test 59: PASS - a5a5 + 5a5a + 1 = 0000 (Cout=1) [Expected: 10000]
# Test 60: PASS - f0f0 + 0f0f + 0 = ffff (Cout=0) [Expected: 0ffff]
#
# Edge Cases Summary:
# Correct: 10/10
```

```
#
# Final Summary:
# Random Tests: 50/50 correct
# Edge Cases:   10/10 correct
# Total:        60/60 correct
# Accuracy:     100.00%
```

Manchester carry chain یک مدل dynamic نیز دارد که در آن از خازن و کلاک استفاده می‌شود. اما از آنجا که در Verilog نمی‌توان به طور مستقیم از خازن استفاده کرد، باید برای مدل‌سازی خازن یک module جدید ایجاد کنیم که پیچیدگی زیادی به کد اضافه می‌کند و ما را از هدف اصلی سوال که طراحی جمع کننده است دور می‌کند.