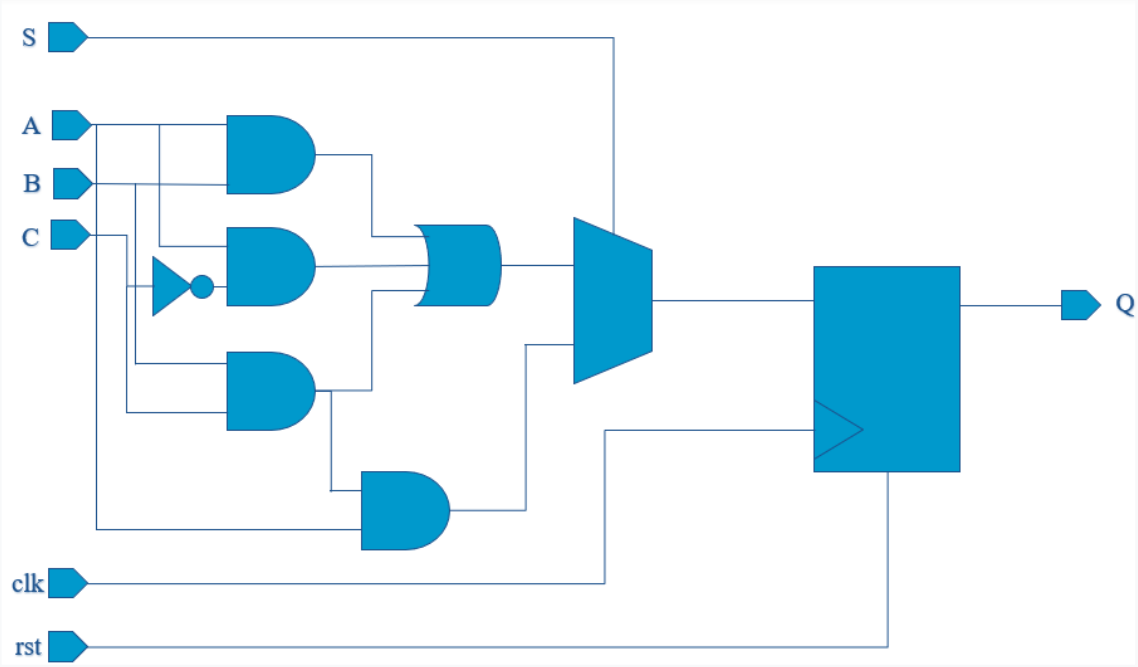
Họ tên: Dương Anh Khôi

MSSV: 22520696

Câu 1: Hãy thiết kế mạch sau bằng ngôn ngữ Verilog HDL:



- Hãy thiết kế bằng mô tả hành vi

- Hãy thiết kế sử dụng các cổng cơ bản được hỗ trợ bằng Verilog

Câu 2: Hãy thiết kế mạch Carry Look Ahead Adder 4 bit bằng ngôn ngữ Verilog HDL

- Hãy thiết kế sử dụng các cổng cơ bản được hỗ trợ bằng Verilog

- Hãy thiết kế bằng cấu trúc structure và các lệnh gán, mô tả hành vi trong thiết kế

Bài làm

Câu 1:

- Thiết kế bằng mô tả hành vi

```
module circuit(
    input wire S, A, B, C, clk, rst,
    output reg Q
);
    // Sử dụng biến reg để lưu kết quả tạm thời
    reg mux_out;
```

```
    // Mô tả hành vi tổ hợp cho phần logic
    always @(*) begin
        // Tính toán đầu ra của bộ MUX dựa trên S
        if (S) begin
            // Khi S=1
            mux_out = (A&B)|(A&~C)|(B&C);
        end else begin
            // Khi S=0:
            mux_out = (B&C)&A;
        end
    end

    // Mô tả hành vi tuần tự cho flip-flop
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            Q <= 1'b0; // Reset đồng bộ, Q = 0 khi rst = 1
        end else begin
            Q <= mux_out; // Cập nhật Q tại mỗi xung clock dương
        end
    end

endmodule
```

- Thiết kế bằng các cổng cơ bản được hỗ trợ bởi Verilog

```
module circuit(
    input wire S, A, B, C, clk, rst,
    output wire Q
```

```
);
    // Khai báo các dây nối trung gian
    wire not_C;
    wire and1_out, and2_out, and3_out, and4_out;
    wire or1_out, or2_out;
    wire mux_out;

    // Thực hiện thiết kế bằng các cổng cơ bản
    // Cổng NOT
    not not1(not_C, C);

    // Các cổng AND
    and and1(and1_out, A, B);       // A & B
    and and2(and2_out, A, not_C);   // A & ~C
    and and3(and3_out, C, B);       // C & B
    and and4(and4_out, and3_out, A);      //A&B&C
    or or1(or1_out, and3_out, and2_out, and1_out);  // (C & B) | (A & ~C) | (A & B)

    // Multiplexer 2-to-1 sử dụng các cổng cơ bản
    wire not_S, and_S_0, and_S_1;
    not not2(not_S, S);
    and and_mux1(and_S_0, not_S, or1_out);    // S=0 path
    and and_mux2(and_S_1, S, and4_out);        // S=1 path
    or or_mux(mux_out, and_S_0, and_S_1);     // Kết hợp 2 đường dẫn

    // DFF với reset
    dff dff_inst(
        .d(mux_out),   // Đầu vào D là đầu ra của MUX
```

```
        .clk(clk),      // Tín hiệu clock
        .rst(rst),      // Truyền tín hiệu reset vào flip-flop
        .q(Q)           // Đầu ra Q
    );
endmodule

// Module D Flip-Flop với reset không đồng bộ
module dff(
    input wire d, clk, rst,
    output reg q
);
    always @(posedge clk or posedge rst) begin
        if (rst)
            q <= 1'b0;  // Reset không đồng bộ
        else
            q <= d;     // Cập nhật giá trị
    end
endmodule
```

Câu 2:

- Thiết kế sử dụng các cổng cơ bản được hỗ trợ bằng Verilog

```
module CLA_4bit(
    input [3:0] A,
    input [3:0] B,
    input Cin,
    output [3:0] Sum,
    output Cout
);
```

```
    // Khai báo các dây dẫn nội bộ
    wire [3:0] P; // Propagate
    wire [3:0] G; // Generate
    wire [4:0] C; // Carry

    // Gán giá trị ban đầu cho C[0]
    assign C[0] = Cin;

    // Tính các tín hiệu Propagate và Generate
    assign P[0] = A[0] ^ B[0];
    assign P[1] = A[1] ^ B[1];
    assign P[2] = A[2] ^ B[2];
    assign P[3] = A[3] ^ B[3];

    assign G[0] = A[0] & B[0];
    assign G[1] = A[1] & B[1];
    assign G[2] = A[2] & B[2];
    assign G[3] = A[3] & B[3];

    // Tính các tín hiệu Carry sử dụng logic CLA
    assign C[1] = G[0] | (P[0] & C[0]);
    assign C[2] = G[1] | (P[1] & G[0]) | (P[1] & P[0] & C[0]);
    assign C[3] = G[2] | (P[2] & G[1]) | (P[2] & P[1] & G[0]) | (P[2] & P[1] & P[0] & C[0]);
    assign C[4] = G[3] | (P[3] & G[2]) | (P[3] & P[2] & G[1]) | (P[3] & P[2] & P[1] & G[0]) |
(P[3] & P[2] & P[1] & P[0] & C[0]);

    // Tính Sum
```

```
    assign Sum[0] = P[0] ^ C[0];
    assign Sum[1] = P[1] ^ C[1];
    assign Sum[2] = P[2] ^ C[2];
    assign Sum[3] = P[3] ^ C[3];

    // Gán Cout
    assign Cout = C[4];

endmodule
```

- Thiết kế bằng cấu trúc structure và các lệnh gán, mô tả hành vi trong thiết kế

```
// Module tạo tín hiệu Generate và Propagate
module GP_Generator(
    input a, b,
    output g, p
);
    // Mô tả hành vi
    assign g = a & b;    // Generate
    assign p = a ^ b;    // Propagate
endmodule

// Module sum generator
module Sum_Generator(
    input p, c,
    output s
);
    // Mô tả hành vi
    assign s = p ^ c;    // Sum = P XOR C
```

```
endmodule


// Module Carry Look Ahead Generator
module CLA_Generator(
    input [3:0] G, P,
    input Cin,
    output [4:0] C
);
    // Mô tả hành vi với các lệnh gán
    assign C[0] = Cin;
    assign C[1] = G[0] | (P[0] & C[0]);
    assign C[2] = G[1] | (P[1] & G[0]) | (P[1] & P[0] & C[0]);
    assign C[3] = G[2] | (P[2] & G[1]) | (P[2] & P[1] & G[0]) | (P[2] & P[1] & P[0] & C[0]);
    assign C[4] = G[3] | (P[3] & G[2]) | (P[3] & P[2] & G[1]) | (P[3] & P[2] & P[1] & G[0]) |
(P[3] & P[2] & P[1] & P[0] & C[0]);
endmodule


// Module top level - CLA 4 bit
module CLA_4bit_Structural(
    input [3:0] A, B,
    input Cin,
    output [3:0] Sum,
    output Cout
);
    // Khai báo dây dẫn nội bộ
    wire [3:0] G, P;
    wire [4:0] C;
```

```
    // Khởi tạo các module con sử dụng thiết kế cấu trúc

    // Generate G và P cho từng bit
    GP_Generator gp0(.a(A[0]), .b(B[0]), .g(G[0]), .p(P[0]));
    GP_Generator gp1(.a(A[1]), .b(B[1]), .g(G[1]), .p(P[1]));
    GP_Generator gp2(.a(A[2]), .b(B[2]), .g(G[2]), .p(P[2]));
    GP_Generator gp3(.a(A[3]), .b(B[3]), .g(G[3]), .p(P[3]));

    // Module tạo các tín hiệu carry
    CLA_Generator cla_gen(
        .G(G),
        .P(P),
        .Cin(Cin),
        .C(C)
    );
    // Tạo các tín hiệu Sum
    Sum_Generator sum0(.p(P[0]), .c(C[0]), .s(Sum[0]));
    Sum_Generator sum1(.p(P[1]), .c(C[1]), .s(Sum[1]));
    Sum_Generator sum2(.p(P[2]), .c(C[2]), .s(Sum[2]));
    Sum_Generator sum3(.p(P[3]), .c(C[3]), .s(Sum[3]));
    // Gán Cout
    assign Cout = C[4];
endmodule
```