

## We are generating a 5-to-32 decoder hierarchically in Verilog. We have generated 2-to-4 decoder defined as follows:

```
module decoder2_4(  
input [1:0] in,  
output reg [3:0] out  
);
```

```
    always @(in) begin  
        case (in)  
            2'b00: out = 4'b0001;  
            2'b01: out = 4'b0010;  
            2'b10: out = 4'b0100;  
            2'b11: out = 4'b1000;  
            default: out = 4'b0000;  
        endcase  
    end
```

```
endmodule
```

Please include the previous module(s) in your response and use them to hierarchically generate a 3-to-8 decoder defined as:

```
module decoder3_8(in[2:0], out[7:0]);  
//Insert code here  
endmodule
```

To generate a 3-to-8 decoder hierarchically using the existing 2-to-4 decoder, you can use two instances of the 2-to-4 decoder. The idea is to use the most significant bit (MSB) of the input to select which decoder's output should be passed through. Here's how you can implement it:

```
module decoder2_4(  
    input [1:0] in,  
    output reg [3:0] out  
);  
  
    always @(in) begin  
        case (in)  
            2'b00: out = 4'b0001;  
            2'b01: out = 4'b0010;  
            2'b10: out = 4'b0100;  
            2'b11: out = 4'b1000;  
            default: out = 4'b0000;  
        endcase  
    end
```

```

        endcase
    end

endmodule

module decoder3_8(
    input [2:0] in,
    output reg [7:0] out
);

    wire [3:0] dec0_out, dec1_out;

    decoder2_4 dec0(
        .in(in[1:0]),
        .out(dec0_out)
    );

    decoder2_4 dec1(
        .in(in[1:0]),
        .out(dec1_out)
    );

    always @(in, dec0_out, dec1_out) begin
        if (~in[2]) begin
            out[3:0] = dec0_out;
            out[7:4] = 4'b0000;
        end else begin
            out[7:4] = dec1_out;
            out[3:0] = 4'b0000;
        end
    end

end

endmodule

```

However, this approach requires additional logic to handle the enable/disable of each decoder's output based on the MSB of the input. A more straightforward way to implement a 3-to-8 decoder using two 2-to-4 decoders would involve using the MSB to select which half of the output should be active. Here's a simpler version:

```

module decoder2_4(
    input [1:0] in,
    output reg [3:0] out
);

    always @(in) begin
        case (in)
            2'b00: out = 4'b0001;
            2'b01: out = 4'b0010;
            2'b10: out = 4'b0100;
            2'b11: out = 4'b1000;
            default: out = 4'b0000;
        endcase
    end

end

```

```

endmodule

module decoder3_8(
    input [2:0] in,
    output reg [7:0] out
);

    wire [3:0] dec0_out, dec1_out;

    decoder2_4 dec0(
        .in(in[1:0]),
        .out(dec0_out)
    );

    decoder2_4 dec1(
        .in(in[1:0]),
        .out(dec1_out)
    );

    always @(in, dec0_out, dec1_out) begin
        if (~in[2]) begin
            out[3:0] = dec0_out;
            out[7:4] = 4'b0000;
        end else begin
            out[7:4] = dec1_out;
            out[3:0] = 4'b0000;
        end
    end
end

endmodule

```

However, the above implementation still uses two decoders unnecessarily. A more efficient approach would be to directly use the logic of a 3-to-8 decoder without needing to instantiate two 2-to-4 decoders. Here's how you can do it directly:

```

module decoder2_4(
    input [1:0] in,
    output reg [3:0] out
);

    always @(in) begin
        case (in)
            2'b00: out = 4'b0001;
            2'b01: out = 4'b0010;
            2'b10: out = 4'b0100;
            2'b11: out = 4'b1000;
            default: out = 4'b0000;
        endcase
    end
endmodule

module decoder3_8(
    input [2:0] in,
    output reg [7:0] out

```

```

);

always @(in) begin
    case (in)
        3'b000: out = 8'b00000001;
        3'b001: out = 8'b00000010;
        3'b010: out = 8'b00000100;
        3'b011: out = 8'b00001000;
        3'b100: out = 8'b00010000;
        3'b101: out = 8'b00100000;
        3'b110: out = 8'b01000000;
        3'b111: out = 8'b10000000;
        default: out = 8'b00000000;
    endcase
end

endmodule

```

But if you insist on using the hierarchical approach with two 2-to-4 decoders, you could simplify it by directly using logic gates to enable/disable the outputs based on the MSB:

```

module decoder2_4(
    input [1:0] in,
    output reg [3:0] out
);

always @(in) begin
    case (in)
        2'b00: out = 4'b0001;
        2'b01: out = 4'b0010;
        2'b10: out = 4'b0100;
        2'b11: out = 4'b1000;
        default: out = 4'b0000;
    endcase
end

endmodule

module decoder3_8(
    input [2:0] in,
    output reg [7:0] out
);

wire [3:0] dec0_out, dec1_out;

decoder2_4 dec0(
    .in(in[1:0]),
    .out(dec0_out)
);

decoder2_4 dec1(
    .in(in[1:0]),
    .out(dec1_out)
);

```

```
    assign out[3:0] = (~in[2]) ? dec0_out : 4'b0000;  
    assign out[7:4] = (in[2]) ? dec1_out : 4'b0000;  
  
endmodule
```

This version uses continuous assignment to directly enable/disable the outputs based on the MSB of the input.