We are generating a 5-to-32 decoder hierarchically in Verilog. We have generated 2-to-4 decoder, 3-to-8 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
```

```
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
```

endmodule

Please include the previous module(s) in your response and use them to hierarchically generate a 5-to-32 decoder defined as:
module decoder5_32(in[4:0], out[31:0]);
//Insert code here
endmodule

To generate a 5-to-32 decoder hierarchically using the existing 3-to-8 decoder, you can use four instances of the 3-to-8 decoder. The idea is to use the two most significant bits (MSBs) 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

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

endmodule

module decoder5_32(
    input [4:0] in,
    output reg [31:0] out
);

    wire [7:0] dec0_out, dec1_out, dec2_out, dec3_out;

    decoder3_8 dec0(
        .in(in[2:0]),
        .out(dec0_out)
    );

    decoder3_8 dec1(
        .in(in[2:0]),
        .out(dec1_out)
    );

    decoder3_8 dec2(
        .in(in[2:0]),
        .out(dec2_out)
    );

    decoder3_8 dec3(
        .in(in[2:0]),
        .out(dec3_out)
    );

    always @(in, dec0_out, dec1_out, dec2_out, dec3_out) begin
        case ({in[4:3]})
            2'b00: begin
                out[7:0] = dec0_out;
                out[31:8] = 24'b0;
            end
            2'b01: begin
                out[15:8] = dec1_out;
                out[31:16] = 16'b0;
                out[7:0] = 8'b0;
            end
            2'b10: begin
                out[23:16] = dec2_out;
                out[31:24] = 8'b0;
                out[15:0] = 16'b0;
            end
            2'b11: begin
                out[31:24] = dec3_out;
                out[23:0] = 24'b0;
            end
            default: out = 32'b0;
        endcase
    end

endmodule
```

However, this approach still uses four decoders unnecessarily. A more efficient approach would be to directly use the logic of a 5-to-32 decoder without needing to instantiate multiple 3-to-8 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
);

    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
```

```
endmodule

module decoder5_32(
    input [4:0] in,
    output reg [31:0] out
);

    always @(in) begin
        case (in)
            5'b00000: out = 32'b00000000000000000000000000000001;
            5'b00001: out = 32'b00000000000000000000000000000010;
            5'b00010: out = 32'b00000000000000000000000000000100;
            5'b00011: out = 32'b00000000000000000000000000001000;
            5'b00100: out = 32'b00000000000000000000000000010000;
            5'b00101: out = 32'b00000000000000000000000000100000;
            5'b00110: out = 32'b00000000000000000000000001000000;
            5'b00111: out = 32'b00000000000000000000000010000000;
            5'b01000: out = 32'b00000000000000000000000100000000;
            5'b01001: out = 32'b00000000000000000000001000000000;
            5'b01010: out = 32'b00000000000000000000010000000000;
            5'b01011: out = 32'b00000000000000000000100000000000;
            5'b01100: out = 32'b00000000000000000001000000000000;
            5'b01101: out = 32'b00000000000000000010000000000000;
            5'b01110: out = 32'b00000000000000000100000000000000;
            5'b01111: out = 32'b00000000000000001000000000000000;
            5'b10000: out = 32'b00000000000000010000000000000000;
            5'b10001: out = 32'b00000000000000100000000000000000;
            5'b10010: out = 32'b00000000000001000000000000000000;
            5'b10011: out = 32'b00000000000010000000000000000000;
            5'b10100: out = 32'b00000000000100000000000000000000;
            5'b10101: out = 32'b00000000001000000000000000000000;
            5'b10110: out = 32'b00000000010000000000000000000000;
            5'b10111: out = 32'b00000000100000000000000000000000;
            5'b11000: out = 32'b00000001000000000000000000000000;
            5'b11001: out = 32'b00000010000000000000000000000000;
            5'b11010: out = 32'b00000100000000000000000000000000;
            5'b11011: out = 32'b00001000000000000000000000000000;
            5'b11100: out = 32'b00010000000000000000000000000000;
            5'b11101: out = 32'b00100000000000000000000000000000;
            5'b11110: out = 32'b01000000000000000000000000000000;
            5'b11111: out = 32'b10000000000000000000000000000000;
            default: out = 32'b0;
        endcase
    end

endmodule
```

However, if you insist on using the hierarchical approach with four 3-to-8 decoders, you could simplify it by directly using logic gates to enable/disable the outputs based on the two MSBs:

```
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)
    );

    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

endmodule

module decoder5_32(
    input [4:0] in,
    output reg [31:0] out
);

    wire [7:0] dec0_out, dec1_out, dec2_out, dec3_out;

    decoder3_8 dec0(
        .in(in[2:0]),
        .out(dec0_out)
    );

    decoder3_8 dec1(
        .in(in[2:0]),
        .out(dec1_out)
    );

    decoder3_8 dec2(
        .in(in[2:0]),
        .out(dec2_out)
    );

    decoder3_8 dec3(
        .in(in[2:0]),
        .out(dec3_out)
    );

    assign out[7:0] = (~in[4]) & (~in[3]) ? dec0_out : 8'b0;
    assign out[15:8] = (~in[4]) & in[3] ? dec1_out : 8'b0;
    assign out[23:16] = in[4] & (~in[3]) ? dec2_out : 8'b0;
    assign out[31:24] = in[4] & in[3] ? dec3_out : 8'b0;

endmodule
```

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