perplexity

We are generating a 32-bit Barrel Shifter with Rotation Control hierarchically in Verilog. We have generated 8-bit Barrel Shifter defined as follows:

module barrel_shift_8(
input [7:0] inputData,
input [2:0] shiftVal,
output reg [7:0] outputData // Declare outputData as reg
);

```
// Use a case statement to simplify the logic
always @(inputData, shiftVal) begin
    case (shiftVal)
        3'b000: outputData = inputData; // No shift
        3'b001: outputData = {inputData[6:0], inputData[7]}; // Shift left by 1
        3'b010: outputData = {inputData[5:0], inputData[7:6]}; // Shift left by 2
        3'b011: outputData = {inputData[4:0], inputData[7:5]}; // Shift left by 3
        3'b100: outputData = {inputData[3:0], inputData[7:4]}; // Shift left by 4
        3'b101: outputData = {inputData[2:0], inputData[7:3]}; // Shift left by 5
        3'b110: outputData = {inputData[1:0], inputData[7:2]}; // Shift left by 6
        3'b111: outputData = {inputData[0], inputData[7:1]}; // Shift left by 7
        default: outputData = inputData; // Default case, no shift
    endcase
end
```

endmodule

Please include the previous module(s) in your response and use them to hierarchically generate a 32-bit Barrel Shifter defined as:
module barrel_shift_32(inputData[31:0], shiftVal[4:0], outputData[31:0]);
//Insert code here
endmodule

To generate a 32-bit barrel shifter hierarchically using the 8-bit barrel shifter module, you can divide the 32-bit input into four 8-bit segments and apply the shift operation to each segment. However, since the shift value can be up to 32 bits, you need to handle shifts that cross segment boundaries. Here's how you can implement it:

```
// 8-bit Barrel Shifter Module
module barrel_shift_8(
    input   [7:0] inputData,
    input   [2:0] shiftVal,
    output reg [7:0] outputData
```

```
);

    // Use a case statement to simplify the logic
    always @(inputData, shiftVal) begin
        case (shiftVal)
            3'b000: outputData = inputData; // No shift
            3'b001: outputData = {inputData[6:0], inputData[7]}; // Shift left by 1
            3'b010: outputData = {inputData[5:0], inputData[7:6]}; // Shift left by 2
            3'b011: outputData = {inputData[4:0], inputData[7:5]}; // Shift left by 3
            3'b100: outputData = {inputData[3:0], inputData[7:4]}; // Shift left by 4
            3'b101: outputData = {inputData[2:0], inputData[7:3]}; // Shift left by 5
            3'b110: outputData = {inputData[1:0], inputData[7:2]}; // Shift left by 6
            3'b111: outputData = {inputData[0], inputData[7:1]}; // Shift left by 7
            default: outputData = inputData; // Default case, no shift
        endcase
    end

endmodule

// 32-bit Barrel Shifter Module
module barrel_shift_32(
    input   [31:0] inputData,
    input   [4:0] shiftVal,
    output reg [31:0] outputData
);

    // Divide the 32-bit input into four 8-bit segments
    wire [7:0] segment1, segment2, segment3, segment4;
    assign segment1 = inputData[31:24];
    assign segment2 = inputData[23:16];
    assign segment3 = inputData[15:8];
    assign segment4 = inputData[7:0];

    // Outputs from each 8-bit shifter
    wire [7:0] shifted1, shifted2, shifted3, shifted4;

    // Instantiate 8-bit shifters for each segment
    barrel_shift_8 shifter1(segment1, shiftVal[2:0], shifted1);
    barrel_shift_8 shifter2(segment2, shiftVal[2:0], shifted2);
    barrel_shift_8 shifter3(segment3, shiftVal[2:0], shifted3);
    barrel_shift_8 shifter4(segment4, shiftVal[2:0], shifted4);

    // Handle shifts that cross segment boundaries
    always @(inputData, shiftVal) begin
        if (shiftVal < 8) begin
            // Shift within the same segment
            outputData = {shifted1, shifted2, shifted3, shifted4};
        end else if (shiftVal < 16) begin
            // Shift across segment boundaries
            outputData = {shifted2[7 - (shiftVal - 8):0], shifted1[7:8 - (shiftVal - 8)],
        end else if (shiftVal < 24) begin
            outputData = {shifted3[7 - (shiftVal - 16):0], shifted2[7:8 - (shiftVal - 16)
        end else begin
            outputData = {shifted4[7 - (shiftVal - 24):0], shifted3[7:8 - (shiftVal - 24)
        end
    end
```

```
endmodule
```

However, the above implementation does not correctly handle shifts that cross segment boundaries for all cases. A more accurate approach involves using a combination of multiplexers to handle the shifts across segments. Here's a revised version:

```
// 8-bit Barrel Shifter Module
module barrel_shift_8(
    input   [7:0] inputData,
    input   [2:0] shiftVal,
    output reg [7:0] outputData
);

    // Use a case statement to simplify the logic
    always @(inputData, shiftVal) begin
        case (shiftVal)
            3'b000: outputData = inputData; // No shift
            3'b001: outputData = {inputData[6:0], inputData[7]}; // Shift left by 1
            3'b010: outputData = {inputData[5:0], inputData[7:6]}; // Shift left by 2
            3'b011: outputData = {inputData[4:0], inputData[7:5]}; // Shift left by 3
            3'b100: outputData = {inputData[3:0], inputData[7:4]}; // Shift left by 4
            3'b101: outputData = {inputData[2:0], inputData[7:3]}; // Shift left by 5
            3'b110: outputData = {inputData[1:0], inputData[7:2]}; // Shift left by 6
            3'b111: outputData = {inputData[0], inputData[7:1]}; // Shift left by 7
            default: outputData = inputData; // Default case, no shift
        endcase
    end

endmodule

// 32-bit Barrel Shifter Module
module barrel_shift_32(
    input   [31:0] inputData,
    input   [4:0] shiftVal,
    output reg [31:0] outputData
);

    // Divide the 32-bit input into four 8-bit segments
    wire [7:0] segment1, segment2, segment3, segment4;
    assign segment1 = inputData[31:24];
    assign segment2 = inputData[23:16];
    assign segment3 = inputData[15:8];
    assign segment4 = inputData[7:0];

    // Outputs from each 8-bit shifter
    wire [7:0] shifted1, shifted2, shifted3, shifted4;

    // Instantiate 8-bit shifters for each segment
    barrel_shift_8 shifter1(segment1, shiftVal[2:0], shifted1);
    barrel_shift_8 shifter2(segment2, shiftVal[2:0], shifted2);
    barrel_shift_8 shifter3(segment3, shiftVal[2:0], shifted3);
    barrel_shift_8 shifter4(segment4, shiftVal[2:0], shifted4);

    // Handle shifts that cross segment boundaries
    always @(inputData, shiftVal) begin
        if (shiftVal < 8) begin
            // Shift within the same segment
            outputData = {shifted1, shifted2, shifted3, shifted4};
        end else if (shiftVal < 16) begin
            // Shift across segment boundaries
            outputData = {shifted2, shifted3, shifted4, shifted1[7 - (shiftVal - 8):0]};
        end else if (shiftVal < 24) begin
            outputData = {shifted3, shifted4, shifted1, shifted2[7 - (shiftVal - 16):0]};
        end else begin
            outputData = {shifted4, shifted1, shifted2, shifted3[7 - (shiftVal - 24):0]};
        end
    end

endmodule
```

However, the above implementation still doesn't handle the rotation correctly for all cases. To correctly implement a 32-bit barrel shifter with rotation control, you should ensure that bits shifted out from one end are correctly rotated to the other end. Here's a corrected version:

```
// 8-bit Barrel Shifter Module
module barrel_shift_8(
    input   [7:0] inputData,
    input   [2:0] shiftVal,
    output reg [7:0] outputData
);

    // Use a case statement to simplify the logic
    always @(inputData, shiftVal) begin
        case (shiftVal)
            3'b000: outputData = inputData; // No shift
            3'b001: outputData = {inputData[6:0], inputData[7]}; // Shift left by 1
            3'b010: outputData = {inputData[5:0], inputData[7:6]}; // Shift left by 2
            3'b011: outputData = {inputData[4:0], inputData[7:5]}; // Shift left by 3
            3'b100: outputData = {inputData[3:0], inputData[7:4]}; // Shift left by 4
            3'b101: outputData = {inputData[2:0], inputData[7:3]}; // Shift left by 5
            3'b110: outputData = {inputData[1:0], inputData[7:2]}; // Shift left by 6
            3'b111: outputData = {inputData[0], inputData[7:1]}; // Shift left by 7
            default: outputData = inputData; // Default case, no shift
        endcase
    end

endmodule

// 32-bit Barrel Shifter Module
module barrel_shift_32(
    input   [31:0] inputData,
    input   [4:0] shiftVal,
    output reg [31:0] outputData
);

    // Divide the 32-bit input into four 8-bit segments
    wire [7:0] segment1, segment2, segment3, segment4;
    assign segment1 = inputData[31:24];
    assign segment2 = inputData[23:16];
    assign segment3 = inputData[15:8];
    assign segment4 = inputData[7:0];

    // Outputs from each 8-bit shifter
    wire [7:0] shifted1, shifted2, shifted3, shifted4;

    // Instantiate 8-bit shifters for each segment
    barrel_shift_8 shifter1(segment1, shiftVal[2:0], shifted1);
    barrel_shift_8 shifter2(segment2, shiftVal[2:0], shifted2);
    barrel_shift_8 shifter3(segment3, shiftVal[2:0], shifted3);
    barrel_shift_8 shifter4(segment4, shiftVal[2:0], shifted4);

    // Handle shifts that cross segment boundaries
    always @(inputData, shiftVal) begin
        if (shiftVal < 8) begin
            // Shift within the same segment
            outputData = {shifted1, shifted2, shifted3, shifted4};
        end else if (shiftVal < 16) begin
            // Shift across segment boundaries
            outputData = {shifted2, shifted3, shifted4, shifted1};
        end else if (shiftVal < 24) begin
            outputData = {shifted3, shifted
```