perplexity

We will be generating a 32-bit Barrel Shifter with Rotation Control hierarchically in Verilog. Please begin by generating a 8-bit Barrel Shifter defined as follows:

module barrel_shift_8(inputData[7:0], shiftVal[2:0], outputData[7:0])

Here's a basic implementation of an 8-bit barrel shifter in Verilog. This module uses a hierarchical approach by breaking down the shift operation into smaller parts, but for simplicity, it directly implements the shift logic using multiplexers.

```
module barrel_shift_8(
    input   [7:0] inputData,
    input   [2:0] shiftVal,
    output  [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
```

However, a more efficient and scalable way to implement a barrel shifter is by using a combination of multiplexers. Here's a more modular version:

```
module barrel_shift_8(
    input   [7:0] inputData,
    input   [2:0] shiftVal,
    output  [7:0] outputData
);

    wire [7:0] shift1, shift2, shift3, shift4;
```

```
    // Shift by 1
    assign shift1 = {inputData[6:0], inputData[7]};

    // Shift by 2
    assign shift2 = {inputData[5:0], inputData[7:6]};

    // Shift by 3
    assign shift3 = {inputData[4:0], inputData[7:5]};

    // Shift by 4
    assign shift4 = {inputData[3:0], inputData[7:4]};

    // Use a case statement to select the output
    always @(inputData, shiftVal) begin
        case (shiftVal)
            3'b000: outputData = inputData; // No shift
            3'b001: outputData = shift1; // Shift left by 1
            3'b010: outputData = shift2; // Shift left by 2
            3'b011: outputData = shift3; // Shift left by 3
            3'b100: outputData = shift4; // Shift left by 4
            3'b101: outputData = {shift4[3:0], shift4[7:4]}; // Shift left by 5
            3'b110: outputData = {shift4[1:0], shift4[7:2]}; // Shift left by 6
            3'b111: outputData = {shift4[0], shift4[7:1]}; // Shift left by 7
            default: outputData = inputData; // Default case, no shift
        endcase
    end

endmodule
```

But for a more hierarchical and scalable approach, you might want to consider using a combination of smaller barrel shifters or a tree of multiplexers. Here's a simplified hierarchical version using a tree of multiplexers:

```
module barrel_shift_8(
    input   [7:0] inputData,
    input   [2:0] shiftVal,
    output  [7:0] outputData
);

    wire [7:0] shift1, shift2, shift3, shift4;

    // Shift by 1
    assign shift1 = {inputData[6:0], inputData[7]};

    // Shift by 2
    assign shift2 = {inputData[5:0], inputData[7:6]};

    // Shift by 3
    assign shift3 = {inputData[4:0], inputData[7:5]};

    // Shift by 4
    assign shift4 = {inputData[3:0], inputData[7:4]};

    // Use a hierarchical approach with multiplexers
```

```
    always @(inputData, shiftVal) begin
        case (shiftVal[2:1])
            2'b00: begin
                case (shiftVal[0])
                    1'b0: outputData = inputData; // No shift
                    1'b1: outputData = shift1; // Shift left by 1
                    default: outputData = inputData; // Default case, no shift
                endcase
            end
            2'b01: begin
                case (shiftVal[0])
                    1'b0: outputData = shift2; // Shift left by 2
                    1'b1: outputData = shift3; // Shift left by 3
                    default: outputData = shift2; // Default case, shift by 2
                endcase
            end
            2'b10: begin
                case (shiftVal[0])
                    1'b0: outputData = shift4; // Shift left by 4
                    1'b1: outputData = {shift4[3:0], shift4[7:4]}; // Shift left by 5
                    default: outputData = shift4; // Default case, shift by 4
                endcase
            end
            2'b11: begin
                case (shiftVal[0])
                    1'b0: outputData = {shift4[1:0], shift4[7:2]}; // Shift left by 6
                    1'b1: outputData = {shift4[0], shift4[7:1]}; // Shift left by 7
                    default: outputData = {shift4[1:0], shift4[7:2]}; // Default case, sh
                endcase
            end
            default: outputData = inputData; // Default case, no shift
        endcase
    end

endmodule
```

Each of these versions has its own advantages depending on the specific requirements and constraints of your design.