perplexity

We are generating a 8-bit UART hierarchically in Verilog. We have generated BaudRateGenerator, Uart8Receiver, Uart8Transmitter defined as follows:

module BaudRateGenerator #(
parameter CLOCK_RATE = 100000000,
parameter BAUD_RATE = 9600,
parameter RX_OVERSAMPLE_RATE = 16
)(
input wire clk,
output reg rxClk,
output reg txClk
);

```
// Calculate clock dividers
localparam RX_DIVIDER = CLOCK_RATE / (BAUD_RATE * RX_OVERSAMPLE_RATE);
localparam TX_DIVIDER = CLOCK_RATE / BAUD_RATE;

// Counters for clock generation
reg [$clog2(RX_DIVIDER)-1:0] rxCounter = 0;
reg [$clog2(TX_DIVIDER)-1:0] txCounter = 0;

always @(posedge clk) begin
    // RX Clock generation
    if (rxCounter == RX_DIVIDER - 1) begin
        rxCounter <= 0;
        rxClk <= ~rxClk;
    end else begin
        rxCounter <= rxCounter + 1;
    end

    // TX Clock generation
    if (txCounter == TX_DIVIDER - 1) begin
        txCounter <= 0;
        txClk <= ~txClk;
    end else begin
        txCounter <= txCounter + 1;
    end
end
```

endmodule

module Uart8Receiver #(
parameter CLOCK_RATE = 100000000,
parameter BAUD_RATE = 9600,
parameter RX_OVERSAMPLE_RATE = 16
)(
input wire clk,
input wire en,
input wire in,
output reg busy,
output reg done,
output reg err,
output reg [7:0] out
);

```
// Instantiate the BaudRateGenerator for generating the RX clock
wire rxClk;
BaudRateGenerator #(
    .CLOCK_RATE(CLOCK_RATE),
    .BAUD_RATE(BAUD_RATE),
    .RX_OVERSAMPLE_RATE(RX_OVERSAMPLE_RATE)
) baudRateGen (
    .clk(clk),
    .rxClk(rxClk),
    .txClk() // TX clock is unused in the receiver module
);

// State machine states
localparam IDLE       = 2'b00;
localparam START_BIT  = 2'b01;
localparam DATA_BITS  = 2'b10;
localparam STOP_BIT   = 2'b11;

reg [1:0] state = IDLE;
reg [3:0] bitIndex = 0; // Track received bits (8 data bits)
reg [7:0] shiftReg = 8'b0; // Shift register for data reception

always @(posedge rxClk) begin
    if (!en) begin
        // Reset state when disabled
        state <= IDLE;
        busy <= 0;
        done <= 0;
        err <= 0;
        bitIndex <= 0;
        shiftReg <= 8'b0;
    end else begin
        case (state)
            IDLE: begin
                done <= 0; // Clear done flag in IDLE state
                err <= 0; // Clear error flag in IDLE state
                if (!in) begin // Detect start bit (low signal)
                    state <= START_BIT;
                    busy <= 1; // Indicate busy during reception process
```

```
                end
            end

            START_BIT: begin
                if (in) begin // Error if start bit is not held low long enough
                    state <= IDLE;
                    busy <= 0;
                    err <= 1; // Set error flag for invalid start bit
                end else begin
                    state <= DATA_BITS; // Move to data reception phase after valid start
                    bitIndex <= 0; // Reset bit index for data reception
                end
            end

            DATA_BITS: begin
                shiftReg[bitIndex] <= in; // Shift received bit into shift register
                bitIndex <= bitIndex + 1;

                if (bitIndex == 7) begin // All data bits received?
                    state <= STOP_BIT; // Move to stop bit phase after receiving all data
                end
            end

            STOP_BIT: begin
                if (!in) begin // Error if stop bit is not high signal
                    err <= 1; // Set error flag for invalid stop bit
                end else begin
                    out <= shiftReg; // Output received byte on successful reception
                    done <= 1;       // Indicate successful reception completion via done
                end

                state <= IDLE; // Return to idle state after stop bit phase completes
                busy <= 0;     // Clear busy flag as reception process ends here.
            end

            default: state <= IDLE; // Default case to reset to idle state on unexpected
        endcase
    end
end
```

endmodule

module Uart8Transmitter #(
parameter TURBO_FRAMES = 0,
CLOCK_RATE = 100000000,
BAUD_RATE = 9600
)(
input wire clk,
input wire en,
input wire start,
input wire [7:0] in,
output reg busy,
output reg done,

output reg out
);

```
// Instantiate the BaudRateGenerator for generating the TX clock.
wire txClk;

BaudRateGenerator #(
    .CLOCK_RATE(CLOCK_RATE),
    .BAUD_RATE(BAUD_RATE),
    .RX_OVERSAMPLE_RATE(16) // Not used for TX, but required by the module definition.
) baudRateGen (
    .clk(clk),
    .rxClk(),   // RX clock is unused in the transmitter module.
    .txClk(txClk)
);

// State machine states.
localparam IDLE       = 2'b00;
localparam START_BIT  = 2'b01;
localparam DATA_BITS  = 2'b10;
localparam STOP_BIT   = 2'b11;

reg [1:0] state = IDLE;
reg [3:0] bitIndex = 0;   // Track transmitted bits (8 data bits).

always @(posedge txClk) begin
    if (!en) begin
        state   <= IDLE;
        busy    <= 0;
        done    <= 0;
        out     <= 1; // Default idle line state is high.
        bitIndex <= 0;
    end else begin
        case (state)
            IDLE: begin
                done <= 0; // Clear done flag in IDLE state.
                if (start) begin
                    state <= START_BIT;
                    busy <= 1; // Indicate busy during transmission process.
                end
            end

            START_BIT: begin
                out <= 0; // Transmit start bit (low signal).
                state <= DATA_BITS; // Move to data transmission phase.
                bitIndex <= 0; // Reset bit index for data transmission.
            end

            DATA_BITS: begin
                out <= in[bitIndex]; // Transmit data bit.
                bitIndex <= bitIndex + 1;

                if (bitIndex == 7) begin // All data bits transmitted?
                    state <= STOP_BIT; // Move to stop bit phase after transmitting all d
                end
```

```
            end

            STOP_BIT: begin
                out <= 1; // Transmit stop bit (high signal).
                state <= IDLE; // Return to idle state after stop bit phase completes.
                busy <= 0;     // Clear busy flag as transmission process ends here.
                done <= 1;     // Indicate successful transmission completion via done fl
            end

            default: state <= IDLE; // Default case to reset to idle state on unexpected
        endcase
    end
end
```

endmodule

Please use the previous module(s) without printing out the previous results in the response to hierarchically generate a State definitions defined as:
module UartStates (
)
//Insert code here
endmodule

```
module UartStates;

    // UART State Definitions
    localparam IDLE       = 2'b00;
    localparam START_BIT  = 2'b01;
    localparam DATA_BITS  = 2'b10;
    localparam STOP_BIT   = 2'b11;

endmodule
```