```
`timescale 1ns / 1ps

module uart_top
#(
parameter clk_freq = 1000000,
parameter baud_rate = 9600
)
(
  input clk,rst,
  input rx,
  input [7:0] dintx,
  input send,
  output tx,
  output [7:0] doutrx,
  output donetx,
  output donerx
    );

uarttx
#(clk_freq, baud_rate)
utx
(clk, rst, send, dintx, tx, donetx);

uartrx
#(clk_freq, baud_rate)
rtx
(clk, rst, rx, donerx, doutrx);


endmodule


//////////////////////////////////////////////////////////////////

module uarttx
#(
parameter clk_freq = 1000000,
parameter baud_rate = 9600
)
(
input clk,rst,
input send,
input [7:0] tx_data,
output reg tx,
output reg donetx
);

  localparam clkcount = (clk_freq/baud_rate); ///x

integer count = 0;
integer counts = 0;

reg uclk = 0;

enum bit[1:0] {idle = 2'b00, start = 2'b01, transfer = 2'b10, done = 2'b11} state;

 ////////////uart_clock_gen
  always@(posedge clk)
    begin
      if(count < clkcount/2)
        count <= count + 1;
      else begin
        count <= 0;
        uclk <= ~uclk;
      end
    end


  reg [7:0] din;
```

```
70     ////////////////////Reset decoder
71
72
73     always@(posedge uclk)
74       begin
75         if(rst)
76         begin
77           state <= idle;
78         end
79        else
80        begin
81        case(state)
82          idle:
83            begin
84              counts <= 0;
85              tx <= 1'b1;
86              donetx <= 1'b0;
87
88              if(send)
89              begin
90                state <= transfer;
91                din <= tx_data;
92                tx <= 1'b0;
93              end
94              else
95                state <= idle;
96            end
97
98
99
100        transfer: begin
101          if(counts <= 7) begin
102             counts <= counts + 1;
103             tx <= din[counts];
104             state <= transfer;
105         end
106         else
107         begin
108            counts <= 0;
109            tx <= 1'b1;
110            state <= idle;
111           donetx <= 1'b1;
112         end
113       end
114
115
116
117
118       default : state <= idle;
119     endcase
120   end
121 end
122
123 endmodule
124
125
126
127 ////////////////////////////////////////////////////////////////////
128
129
130
131
132 module uartrx
133 #(
134 parameter clk_freq = 1000000, //MHz
135 parameter baud_rate = 9600
136     )
137  (
138 input clk,
```

```
input rst,
input rx,
output reg done,
output reg [7:0] rxdata
);

localparam clkcount = (clk_freq/baud_rate);

integer count = 0;
integer counts = 0;

reg uclk = 0;


enum bit[1:0] {idle = 2'b00, start = 2'b01} state;

 ////////////uart_clock_gen
  always@(posedge clk)
    begin
      if(count < clkcount/2)
        count <= count + 1;
      else begin
        count <= 0;
        uclk <= ~uclk;
      end
    end



  always@(posedge uclk)
    begin
      if(rst)
      begin
     rxdata <= 8'h00;
     counts <= 0;
     done <= 1'b0;
      end
     else
     begin
     case(state)

     idle :
     begin
     rxdata <= 8'h00;
     counts <= 0;
     done <= 1'b0;

     if(rx == 1'b0)
       state <= start;
     else
       state <= idle;
     end

     start:
     begin
       if(counts <= 7)
      begin
     counts <= counts + 1;
     rxdata <= {rx, rxdata[7:1]};
     end
     else
     begin
     counts <= 0;
     done <= 1'b1;
     state <= idle;
     end
     end


```

```
208      default : state <= idle;
209
210      endcase
211
212    end
213
214    end
215
216    endmodule
217
218
219    //////////////////////////////////////////////////////////////////
220
221    interface uart_if;
222      logic clk;
223      logic uclktx;
224      logic uclkrx;
225      logic rst;
226      logic rx;
227      logic [7:0] dintx;
228      logic send;
229      logic tx;
230      logic [7:0] doutrx;
231      logic donetx;
232      logic donerx;
233
234    endinterface
```