```
`timescale 1ns / 1ps
/********************************************************************** *
* Module: rx
*
* Author: Noah Hanks
* Class: ECEN 220, Section 3, Fall 2020 - ECEN 220, Section 1, Winter 2020 * Date: 2
December 2020
*
* Description: takes an ascii input from a computer and is able to receive it on the
fpga board
*
*
**********************************************************************/
`default_nettype none

module rx(
    input wire logic clk, Reset, Sin, Received,
    output logic Receive, parityErr,
    output logic[7:0] Dout
    );
    logic BaudHalfDone, BaudFullDone, clrTimer, incBit, bitDone, clrBit;
    logic[3:0] bitCounter;
    logic[12:0] extra1, extra2;
    typedef enum logic[2:0] {IDLE, START, BITS, ACK, ERR='X} StateType;
    StateType ns, cs;
    logic[8:0] register;

    mod_counter #(2604, 13) BaudCoutnerHalf (.clk(clk), .reset(clrTimer),
.increment(1'b1), .rolling_over(BaudHalfDone), .count(extra1));
    mod_counter #(5208, 13) BaudCoutnerFull (.clk(clk), .reset(clrTimer),
.increment(1'b1), .rolling_over(BaudFullDone), .count(extra2));

    //the state machine for the the uart receiver
    always_comb begin
        incBit = 0;
        Receive = 0;
        clrTimer = 0;
        clrBit = 0;
        ns = ERR;
        if(Reset) begin
            ns = IDLE;
            clrTimer = 1;
            end
        else
            case(cs)
        IDLE: begin
```

```
            clrTimer = 1;
            if(Sin)
                ns = IDLE;
            else if(~Sin)
                ns = START;
            end
        START: begin
            clrBit = 1;
            if(~BaudHalfDone)
                ns = START;
            else if(BaudHalfDone) begin
                clrTimer = 1;
                ns = BITS;
                end
        end
        BITS: begin
            if(~BaudFullDone) begin
                ns = BITS;
                end
            else if(BaudFullDone & ~bitDone) begin
                incBit = 1;
                ns = BITS;
                end
            else if(BaudFullDone & bitDone)
                ns = ACK;
        end
        ACK: begin
            Receive = 1;
            if(~Received)
                ns = ACK;
            else if(Received)
                ns = IDLE;
        end
        endcase
    end

    assign bitDone = (bitCounter == 9);

    //sets the current state to the next state, resets and increments the bit
counter, and shifts the register
    always_ff @(posedge clk) begin
    cs <= ns;
    if((bitCounter == 9 && incBit) || Reset || clrBit)
        bitCounter <= 0;
    if(incBit) begin
        bitCounter <= bitCounter + 1;
```

```
        register <= {Sin, register[8:1]};
        end
    end
    assign Dout = register[7:0];
    assign parityErr = ~^register;


endmodule
```