`timescale 1ns / 1ps

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*

\* Module: tx

\*

\* Author: Noah Hanks

\* Class: ECEN 220, Section 3, Fall 2020 - ECEN 220, Section 1, Winter 2020 \* Date: 11 Nov 2020

\*

\* Description: Creates a UART that is able to send ascii code through a transmitter to be diplayed onto a computer.

\*

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ `default\_nettype none

module tx(

input wire logic clk, Reset, Send,

input wire logic[7:0] Din,

output logic Sent, Sout

);

logic clrTimer, timerDone;

logic[12:0] extra1;

mod\_counter #(5208, 13) BaudCoutner (.clk(clk), .reset(clrTimer), .increment(1'b1), .rolling\_over(timerDone), .count(extra1));

typedef enum logic[2:0] {IDLE, START, BITS, PAR, STOP, ACK, ERR='X} StateType; StateType ns, cs;

logic startBit, clrBit, dataBit, bitDone, incBit, parityBit, bitDoneTrash; logic[2:0] bitNum;

mod\_counter #(8, 3) BitCounter (.clk(clk), .reset(clrBit), .increment(incBit), .rolling\_over(bitDoneTrash), .count(bitNum));

assign bitDone = (bitNum == 7)?1:0;

//controls the Sout output to send the start, data, parity and stop bits always\_ff @(posedge clk) begin

if(startBit)

Sout <= 0;

else if(dataBit)

Sout <= Din[bitNum];

else if(parityBit)

Sout <= ~^Din;

else

Sout <= 1;

end

//starts the full state machine for the UART always\_comb begin

startBit = 0;

dataBit = 0;

incBit = 0;

parityBit = 0;

clrBit = 0;

Sent = 0;

ns = ERR;

clrTimer = 0;

if(Reset)

ns = IDLE;

else

case(cs)

IDLE: begin

clrTimer = 1;

if(~Send) begin

ns = IDLE;

end

else

ns = START;

end

START: begin

startBit = 1;

if(~timerDone)

ns = START;

else begin

clrBit = 1;

ns = BITS;

end

end

BITS: begin

dataBit = 1;

if((timerDone & ~bitDone)) begin incBit = 1;

ns = BITS;

end

else if(~timerDone)

ns = BITS;

else if(timerDone & bitDone) ns = PAR;

end

PAR: begin

parityBit = 1;

if(~timerDone)

ns = PAR;

else

ns = STOP;

end

STOP: begin

if(~timerDone)

ns = STOP;

else

ns = ACK;

end

ACK: begin

Sent = 1;

if(Send)

ns = ACK;

else

ns = IDLE;

end

endcase

end

//this block makes sure the current state is always being set to the next state always\_ff @(posedge clk)

cs <= ns;

endmodule