```
----------------------------------------------------------------------------------
-- Company:
-- Engineer: David Paquette
--
-- Create Date:    11/20/15
-- Design Name:
-- Module Name:
-- Project Name:
-- Target Device:
-- Tool versions:
-- Description:

-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ValuesToSerial is
	port
	(
		-- General
		CLOCK                   :   in      std_logic;
		RESET                   :   in      std_logic;
		RX                      :   in      std_logic;
		TX                      :   out     std_logic;
		temperatureIn : in std_logic_vector(7 downto 0);
		fanSpeedIn : in std_logic_vector(7 downto 0)
		);
end ValuesToSerial;

architecture RTL of ValuesToSerial is
	constant BAUD_RATE              : positive := 115200;
	constant CLOCK_FREQUENCY        : positive := 100000000;

	signal uart_data_in             : std_logic_vector(7 downto 0);
	signal uart_data_out            : std_logic_vector(7 downto 0);
	signal uart_data_in_stb         : std_logic;
	signal uart_data_in_ack         : std_logic;
	signal uart_data_out_stb        : std_logic;
	signal uart_data_out_ack        : std_logic;

      signal s_clk: std_logic;
      type StateType is (start, writeTemperatureData, writeComma,
            writeFanSpeedData, writeNewline, waitForOneSecondTick);
      signal state : StateType := start;


      signal oneSecondCounter :std_logic_vector(31 downto 0):=(others=>'0');
begin
```

```
s_clk<=CLOCK;

 UART_inst1 : entity work.UART
 generic map (
                  BAUD_RATE           => BAUD_RATE,
                  CLOCK_FREQUENCY     => CLOCK_FREQUENCY
 )
 port map    (
                  CLOCK               => CLOCK,
                  RESET               => RESET,
                  DATA_STREAM_IN      => uart_data_in,
                  DATA_STREAM_IN_STB  => uart_data_in_stb,
                  DATA_STREAM_IN_ACK  => uart_data_in_ack,
                  DATA_STREAM_OUT     => uart_data_out,
                  DATA_STREAM_OUT_STB => uart_data_out_stb,
                  DATA_STREAM_OUT_ACK => uart_data_out_ack,
                  TX                  => TX,
                  RX                  => RX
 );

 process(s_clk,RESET)
 begin
      if(RESET='1') then
             uart_data_in_stb        <= '0';
             uart_data_out_ack       <= '0';
             uart_data_in            <= (others => '0');
             state<=start;
             oneSecondCounter<=(others=>'0');
      elsif(s_clk'event and s_clk='1') then
            case state is
                  when start =>
                        uart_data_in_stb <= '0';
                        state<=writeTemperatureData;
                        oneSecondCounter<=(others=>'0');
                  when writeTemperatureData =>
                        if(uart_data_in_ack = '1') then
                              state<=writeComma;
                              uart_data_in_stb <= '0';
                        else
                              uart_data_in_stb <= '1';
                              uart_data_in<= temperatureIn;
                        end if;
                  when writeComma =>
                        if(uart_data_in_ack = '1') then
                              state<=writeFanSpeedData;
                              uart_data_in_stb <= '0';
                        else
                              uart_data_in_stb <= '1';
                              uart_data_in<= x"2c";
                        end if;
                  when writeFanSpeedData =>
                        if(uart_data_in_ack = '1') then
                              state<=writeNewLine;
                              uart_data_in_stb <= '0';
                        else
```

```
                                    uart_data_in_stb <= '1';
                                    uart_data_in<= fanSpeedIn;
                              end if;
                        when writeNewLine =>
                              if(uart_data_in_ack = '1') then
                                    state<=waitForOneSecondTick;
                                    uart_data_in_stb <= '0';
                              else
                                    uart_data_in_stb <= '1';
                                    uart_data_in<= x"0A";
                              end if;
                        when waitForOneSecondTick =>
                              uart_data_in_stb<='0';
                              oneSecondCounter <= oneSecondCounter + 1;
                              --if(oneSecondCounter >= 100000000) then
                              if(oneSecondCounter >= 10000000) then
                                    state<=start;
                              end if;
                  end case;
            end if;
      end process;


end RTL;
```