Controle de Periféricos -Projeto 8 - UART RS232 Serial (`my_uart_top`)

Controle de Periféricos - Projeto 8 - UART RS232 Serial (`my_uart_top`)

Projeto em funcionamento

Procedimento para utilizar o programa:

- 1 - Conectar à porta RS232 do kit FPGA um cabo RS232-USB e conectar o cabo no computador.
- 2 - Buscar no "Gerenciador de Dispositivos" o número da porta Serial.
- 3 - Usar um programa de comunicação Serial (RealTerm, Arduino etc.) e usar as seguintes configurações:
  - *Baud rate*: 9600.
  - 8 bits de dados.
  - 1 bit de parada (*stop bit*).
- 4 - No programa que realiza a comunicação serial:
  - O usuário deverá enviar, por meio do programa, caracteres para o kit FPGA conectado ao computador.
  - O kit FPGA, quando receber os dados, irá enviar os mesmo dados de volta para o PC.

Códigos dos módulos do projeto

Módulo `speed_select`

```
`timescale 1ns / 1ps

module speed_select(input clk, input rst_n, input bps_start, output clk_bps);
/*
parameter       bps9600     = 5207, //Taxa de transmissao de 9600bps
                   bps19200     = 2603, //Taxa de transmissao de 19200bps
                   bps38400     = 1301, //Taxa de transmissao de 38400bps
                   bps57600     = 867,  //Taxa de transmissao de 57600bps
                   bps115200    = 433;  //Taxa de transmissao de 115200bps

parameter       bps9600_2   = 2603,
                   bps19200_2   = 1301,
                   bps38400_2   = 650,
                   bps57600_2   = 433,
```

```
                    bps115200_2 = 216;
*/

// Valores de cntagem de divisao de frequencia de taxa de transmissao
//          podem ser alterados de acordo com os valores acima
//
`define BPS_PARA        5207
`define BPS_PARA_2  2603

        // Declaracao de variaveis e sinais
        reg[12:0] cnt;
        reg clk_bps_r;

        // Registro de selecao da taxa de transmissao
        reg[2:0] uart_ctrl;

        // Bloco always executado nas bordas de subida de clk ou bordas de descida de rst
        always @ ( posedge clk or negedge rst_n )
        begin
                if( !rst_n ) begin
                        cnt <= 13'd0;
                end

                else if( (cnt == `BPS_PARA) || !bps_start ) begin
                        cnt <= 13'd0;
                end

                else begin
                        cnt <= cnt + 1'b1;
                end
        end

        // Bloco always executado nas bordas de subida de clk ou bordas de descida de rst
        always @ ( posedge clk or negedge rst_n )
        begin
            if( !rst_n ) begin
                    clk_bps_r <= 1'b0;
            end

            else if( (cnt == `BPS_PARA_2) && bps_start ) begin
                    clk_bps_r <= 1'b1;
            end

            else begin
                    clk_bps_r <= 1'b0;
            end
        end

        // --> Atribuir a 'clk_bps' o valor de 'clk_bps_r'
        assign clk_bps = clk_bps_r;
endmodule
```

Módulo `my_uart_tx`

```
`timescale 1ns / 1ps

module my_uart_tx(input clk, input rst_n, input [7:0] rx_data, input rx_int, input clk_bps, output rs23
        // Declaracao dos sinais e variaveis
        reg rx_int0, rx_int1, rx_int2;
        wire neg_rx_int;

        always @ ( posedge clk or negedge rst_n )
        begin
                if( !rst_n ) begin
                        rx_int0 <= 1'b0;
                        rx_int1 <= 1'b0;
                        rx_int2 <= 1'b0;
                end

                else begin
                        rx_int0 <= rx_int;
                        rx_int1 <= rx_int0;
                        rx_int2 <= rx_int1;
                end
        end

        // Estado do sinal 'neg_rx_int'
        assign neg_rx_int =  ~rx_int1 & rx_int2;

        //-------------------------------------------------------
        reg[7:0] tx_data;   // Cadastrar para enviar dados
        //-------------------------------------------------------
        reg bps_start_r;
        reg tx_en;  //Enviar sinal de habilitacao de dados, ativo em HIGH
        reg[3:0] num;

        always @ ( posedge clk or negedge rst_n )
        begin
                if( !rst_n ) begin
                        bps_start_r <= 1'bz;
                        tx_en <= 1'b0;
                        tx_data <= 8'd0;
                end

                else if( neg_rx_int ) begin //Depois de receber os dados, preparar para enviar os dados
                        bps_start_r <= 1'b1;
                        tx_data <= rx_data; // Armazenar os dados recebidos no registrador de dados de
                        tx_en <= 1'b1;      // Entrando no estado de envio de dados
                end

                else if( num==4'd11 ) begin //Transmissao de dados concluida
                        bps_start_r <= 1'b0;
                        tx_en <= 1'b0;
                end
        end
```

```
        // Estado de 'bps_start'
        assign bps_start = bps_start_r;

        //------------------------------------------------------------
        reg rs232_tx_r;

        always @ ( posedge clk or negedge rst_n )
        begin
                if(!rst_n) begin
                        num <= 4'd0;
                        rs232_tx_r <= 1'b1;
                end

                else if( tx_en ) begin
                        if(clk_bps) begin
                                num <= num+1'b1;

                                case (num)
                                        4'd0: rs232_tx_r <= 1'b0;   //Enviar bit inicial
                                        4'd1: rs232_tx_r <= tx_data[0]; //Enviar bit0
                                        4'd2: rs232_tx_r <= tx_data[1]; //Enviar bit1
                                        4'd3: rs232_tx_r <= tx_data[2]; //Enviar bit2
                                        4'd4: rs232_tx_r <= tx_data[3]; //Enviar bit3
                                        4'd5: rs232_tx_r <= tx_data[4]; //Enviar bit4
                                        4'd6: rs232_tx_r <= tx_data[5]; //Enviar bit5
                                        4'd7: rs232_tx_r <= tx_data[6]; //Enviar bit6
                                        4'd8: rs232_tx_r <= tx_data[7]; //Enviar bit7
                                        4'd9: rs232_tx_r <= 1'b1;   // Enviar bit final
                                        default: rs232_tx_r <= 1'b1;
                                endcase
                        end
                end

                else if( num == 4'd11) begin
                        num <= 4'd0;  // Redefinir
                end
        end

        // Atribuir o estado de 'rs232_tx'
        assign rs232_tx = rs232_tx_r;
endmodule
```

Módulo `my_uart_rx`

```
`timescale 1ns / 1ps

module my_uart_rx( input clk, input rst_n, input rs232_rx, input clk_bps, output [7:0] rx_data, output r
        //-----------------------------------------------------------------
        reg rs232_rx0, rs232_rx1, rs232_rx2, rs232_rx3; // Receber registro de dados para filtragem
        wire neg_rs232_rx;  // Indica que a linha de dados recebe uma borda de descida

        always @ ( posedge clk or negedge rst_n )
        begin
```

```
        if(!rst_n) begin
                rs232_rx0 <= 1'b0;
                rs232_rx1 <= 1'b0;
                rs232_rx2 <= 1'b0;
                rs232_rx3 <= 1'b0;
        end

        else begin
                rs232_rx0 <= rs232_rx;
                rs232_rx1 <= rs232_rx0;
                rs232_rx2 <= rs232_rx1;
                rs232_rx3 <= rs232_rx2;
        end
end

// A detecção de borda descendente abaixo pode filtrar falhas <20ns-40ns (falhas de pulso alto
// Aqui é para trocar recursos por estabilidade (desde que não sejamos tão rigorosos com os req
//          porque o sinal de entrada foi batido algumas vezes)
//(Eh claro que nosso sinal de pulso baixo efetivo deve ser muito maior que 40ns)

// Neg_rs232_rx é definido como alto por um ciclo de clock após receber a borda descendente
assign neg_rs232_rx = rs232_rx3 & rs232_rx2 & ~rs232_rx1 & ~rs232_rx0;

//--------------------------------------------------------------
reg bps_start_r;
reg[3:0] num;   // Número de turnos
//reg rx_int;       //Recebe sinal de interrupção de dados, sempre em alto nível durante a rece


always @ ( posedge clk or negedge rst_n )
begin
        if( !rst_n ) begin
                bps_start_r <= 1'bz;
                rx_int <= 1'b0;
        end

        else if( neg_rs232_rx ) begin // Recebeu o sinal de bandeira de borda descendente da li
                bps_start_r <= 1'b1;    // Inicie a porta serial para preparar a recepção de da
                rx_int <= 1'b1;  // Ativar sinal de interrupção de dados
        end

        else if( num == 4'd11 ) begin //Informações de dados úteis recebidas
                bps_start_r <= 1'b0;  // Depois que os dados forem recebidos, solte o sinal de
                rx_int <= 1'b0;       // Receber sinal de interrupção de dados desligado
        end
end

// Atribuicao do estado de 'bps_start'
assign bps_start = bps_start_r;

//--------------------------------------------------------------
reg[7:0] rx_data_r;     //A porta serial recebe o registro de dados e o salva até que os próxim
//--------------------------------------------------------------
```

```
        reg[7:0] rx_temp_data;  //registro de dados de recebimento atual

        always @ ( posedge clk or negedge rst_n )
        begin
                if( !rst_n ) begin
                        rx_temp_data <= 8'd0;
                        num <= 4'd0;
                        rx_data_r <= 8'd0;
                end

                else if(rx_int) begin   //receber processamento de dados
                        if( clk_bps ) begin //Leia e salve dados, receba dados como um bit inicial, dad
                                num <= num + 1'b1;

                                case (num)
                                        4'd1: rx_temp_data[0] <= rs232_rx;  //Trave o bit 0
                                        4'd2: rx_temp_data[1] <= rs232_rx;  //Trave o bit 1
                                        4'd3: rx_temp_data[2] <= rs232_rx;  //Trave o bit 2
                                        4'd4: rx_temp_data[3] <= rs232_rx;  //Trave o bit 3
                                        4'd5: rx_temp_data[4] <= rs232_rx;  //Trave o bit 4
                                        4'd6: rx_temp_data[5] <= rs232_rx;  //Trave o bit 5
                                        4'd7: rx_temp_data[6] <= rs232_rx;  //Trave o bit 6
                                        4'd8: rx_temp_data[7] <= rs232_rx;  //Trave o bit 7
                                        default: ;
                                endcase
                        end

                        // Nosso modo de recebimento padrão tem apenas 1+8+1(2)=11 bits de dados válido
                        // Altere o número após num neste lugar para 11. Acabou sendo 12! !
                        else if(num == 4'd11) begin
                                num <= 4'd0; // Terminar após receber o bit STOP, num é apagado
                                rx_data_r <= rx_temp_data;  //Trave os dados no registro de dados rx_da
                        end
                end
        end

        // Atribuir o estado de 'rx_data'
        assign rx_data = rx_data_r;
endmodule
```

Módulo `my_uart_top` (módulo principal)

```
module my_uart_top( input wire FPGA_CLK, input wire FPGA_RST, input wire UART_RXD, output wire UART_TXD
        // Sinais referentes aos nomes usados no projeto original
        wire clk = FPGA_CLK;
        wire rst_n = FPGA_RST;
        wire rs232_rx = UART_RXD;

        // Declaracao de sinais
        wire bps_start1, bps_start2;
        wire clk_bps1,clk_bps2;
        wire[7:0] rx_data;
        wire rx_int;
```

```
/*
** --> Entre os quatro módulos a seguir, speed_rx e speed_tx são dois módulos de hardware completamente
**              ser chamados de replicação lógica.
** --> (Não é compartilhamento de recursos e não pode ser confundido com a mesma chamada de subrotina n
*/

    // Objeto do modulo 'speed_select'; Módulo de seleção de taxa de transmissão
    speed_select speed_rx( .clk(clk), .rst_n(rst_n), .bps_start(bps_start1), .clk_bps(clk_bps1) );

    // Objeto do modulo 'my_uart_rx'; receber módulo de dados
    my_uart_rx my_uart_rx( .clk(clk), .rst_n(rst_n), .rs232_rx(rs232_rx), .rx_data(rx_data), .rx_int(rx_

    ////////////////////////////////////////
    // Objeto do modulo 'speed_select'; Módulo de seleção de taxa de transmissão
    speed_select speed_tx( .clk(clk), .rst_n(rst_n), .bps_start(bps_start2), .clk_bps(clk_bps2) );

    // Objeto do modulo 'my_uart_tx'; receber módulo de dados
    my_uart_tx my_uart_tx(.clk(clk), .rst_n(rst_n), .rx_data(rx_data), .rx_int(rx_int), .rs232_tx(UART_T
endmodule
```