Controle de Periféricos -Projeto 9 - Entrada PS/2 e teste usando teclado (`ps2_key`)

Controle de Periféricos - Projeto 9 - Entrada PS/2 e teste usando teclado (`ps2_key`)

Projeto em funcionamento

Procedimento para utilizar o programa:

- 1 - Conectar um teclado com conexão PS/2 na porta PS/2 do kit FPGA.
- 2 - Conectar à porta RS232 do kit FPGA um cabo RS232-USB e conectar o cabo no computador.
- 3 - Utilizar um programa para realizar a conexão serial entre o kit FPGA e o PC.
- 4 - No programa com a conexao serial, as teclas que forem pressionadas irão aparecer no console da Serial.

Códigos dos módulos do projeto

Módulo `speed_select`

```
`timescale 1ns / 1ps

module speed_select(input wire clk, input wire rst_n, input wire bps_start, output wire 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 wire clk, input wire rst_n, input wire [7:0] rx_data, input wire rx_int, input w
        // 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 ps2scan

```
`timescale 1ns / 1ps

module ps2scan( input wire clk, input wire rst_n, input wire ps2k_clk, input wire ps2k_data, output wir
        //----------------------------------------
        reg ps2k_clk_r0, ps2k_clk_r1, ps2k_clk_r2;  //registrador de status ps2k_clk
        wire neg_ps2k_clk; //ps2k_clk flag para borda de descida

        always @ ( posedge clk or negedge rst_n )
        begin
                if(!rst_n) begin
                    ps2k_clk_r0 <= 1'b0;
                    ps2k_clk_r1 <= 1'b0;
                    ps2k_clk_r2 <= 1'b0;
                end

                else begin  // Estado de trava, filtro
                    ps2k_clk_r0 <= ps2k_clk;
                    ps2k_clk_r1 <= ps2k_clk_r0;
```

```
            ps2k_clk_r2 <= ps2k_clk_r1;
        end
end

// Atribuicao do estado de 'neg_ps2k_clk'
assign neg_ps2k_clk = ~ps2k_clk_r1 & ps2k_clk_r2;   //Borda de descida

//----------------------------------------
reg [7:0] ps2_byte_r;       // PC recebe um byte de memória de dados do PS2
reg [7:0] temp_data;            // registro de dados de recebimento atual
reg [3:0] num;                 // registro de contagem

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

        // Detectada uma borda de descida em ps2k_clk
        else if( neg_ps2k_clk ) begin
                case( num )
                        4'd0:   num <= num + 1'b1;

                        4'd1:   begin
                                num <= num + 1'b1;
                                temp_data[0] <= ps2k_data;  //bit0
                        end

                        4'd2:   begin
                            num <= num+1'b1;
                            temp_data[1] <= ps2k_data;  //bit1
                        end

                        4'd3:   begin
                            num <= num+1'b1;
                            temp_data[2] <= ps2k_data;  //bit2
                        end

                        4'd4:   begin
                            num <= num+1'b1;
                            temp_data[3] <= ps2k_data;  //bit3
                        end

                        4'd5:   begin
                            num <= num+1'b1;
                            temp_data[4] <= ps2k_data;  //bit4
                        end

                        4'd6:   begin
                            num <= num+1'b1;
                            temp_data[5] <= ps2k_data;  //bit5
                        end
```

```
                        4'd7:   begin
                            num <= num+1'b1;
                            temp_data[6] <= ps2k_data;  //bit6
                        end

                        4'd8:   begin
                            num <= num+1'b1;
                            temp_data[7] <= ps2k_data;  //bit7
                        end

                        4'd9:   begin
                            num <= num+1'b1;    //Bit de paridade, não processa
                        end

                        4'd10: begin
                            num <= 4'd0;    // num é limpo
                        end

                        default: ;
                    endcase
            end
    end

    //---------------
    reg key_f0;     // Solte o bit de sinalizador de chave, definido como 1 para indicar que os dad
    reg ps2_state_r;    // O estado atual do teclado, ps2_state_r=1 significa que uma tecla foi pre

    // Processamento correspondente de dados recebidos, aqui apenas o valor da chave de 1 byte é pr
    always @ ( posedge clk or negedge rst_n )
    begin
            if( !rst_n ) begin
                key_f0 <= 1'b0;
                ps2_state_r <= 1'b0;
            end

            //Acabei de transferir um byte de dados
            else if( num == 4'd10 ) begin
                    if( temp_data == 8'hf0 ) begin
                            key_f0 <= 1'b1;
                    end

                    else begin
                            // Indica que há uma tecla pressionada
                            if( !key_f0 ) begin
                                    ps2_state_r <= 1'b1;
                                    ps2_byte_r <= temp_data; //Travar o valor da chave atual
                            end

                            else begin
                                    ps2_state_r <= 1'b0;
                                    key_f0 <= 1'b0;
                            end
                    end
```

```
                end
        end

        // O código ASCII correspondente dos dados recebidos
        reg [7:0] ps2_asci;

        always @ ( ps2_byte_r ) begin
                // Converta o valor da chave para código ASCII, que é relativamente simples aqui, lidan
                case( ps2_byte_r )
                        8'h15: ps2_asci <= 8'h51;   //Q
                        8'h1d: ps2_asci <= 8'h57;   //W
                        8'h24: ps2_asci <= 8'h45;   //E
                        8'h2d: ps2_asci <= 8'h52;   //R
                        8'h2c: ps2_asci <= 8'h54;   //T
                        8'h35: ps2_asci <= 8'h59;   //Y
                        8'h3c: ps2_asci <= 8'h55;   //U
                        8'h43: ps2_asci <= 8'h49;   //I
                        8'h44: ps2_asci <= 8'h4f;   //O
                        8'h4d: ps2_asci <= 8'h50;   //P
                        8'h1c: ps2_asci <= 8'h41;   //A
                        8'h1b: ps2_asci <= 8'h53;   //S
                        8'h23: ps2_asci <= 8'h44;   //D
                        8'h2b: ps2_asci <= 8'h46;   //F
                        8'h34: ps2_asci <= 8'h47;   //G
                        8'h33: ps2_asci <= 8'h48;   //H
                        8'h3b: ps2_asci <= 8'h4a;   //J
                        8'h42: ps2_asci <= 8'h4b;   //K
                        8'h4b: ps2_asci <= 8'h4c;   //L
                        8'h1a: ps2_asci <= 8'h5a;   //Z
                        8'h22: ps2_asci <= 8'h58;   //X
                        8'h21: ps2_asci <= 8'h43;   //C
                        8'h2a: ps2_asci <= 8'h56;   //V
                        8'h32: ps2_asci <= 8'h42;   //B
                        8'h31: ps2_asci <= 8'h4e;   //N
                        8'h3a: ps2_asci <= 8'h4d;   //M
                        default: ;
                endcase
        end

        // --> Atribuicao dos sinais de 'ps2_byte' e 'ps2_state'
        assign ps2_byte = ps2_asci;
        assign ps2_state = ps2_state_r;
endmodule
```

Módulo ps2_key (módulo principal)

```
module ps2_key(input wire FPGA_CLK, input wire FPGA_RST, input wire PS_CLOCK, input wire PS_DATA, outpu
        // Sinais referentes aos nomes usados no projeto original
        wire clk = FPGA_CLK;
        wire rst_n = FPGA_RST;
        wire ps2k_clk = PS_CLOCK;
        wire ps2k_data = PS_DATA;
```

```
        // Declaracao de sinais e variaveis
        wire [7:0] ps2_byte;  // valor de chave de 1 byte
        wire ps2_state;       // sinalizador de status do botão
        wire bps_start;       // Depois que os dados são recebidos, o sinal de início do clock da taxa d
        wire clk_bps;         // O alto nível de clk_bps é o ponto de amostragem médio de recebimento ou


        // --> Objeto do modulo 'ps2scan'; Módulo de digitalização de chaves
        ps2scan ps2scan( .clk(clk), .rst_n(rst_n), .ps2k_clk(ps2k_clk), .ps2k_data(ps2k_data), .ps2_byte

        // --> Objeto do modulo 'speed_select';
        speed_select speed_select( .clk(clk), .rst_n(rst_n), .bps_start(bps_start), .clk_bps(clk_bps) )

        // --> Objeto do modulo 'my_uart_tx';
        my_uart_tx my_uart_tx( .clk(clk), .rst_n(rst_n), .clk_bps(clk_bps), .rx_data(ps2_byte), .rx_int
endmodule
```