Universidade do Minho

Mestrado Integrado em Engenharia de Electrónica Industrial e Computadores

Projeto de Sistemas de Telecomunicações I: Hierarquia Digital Plesiócrona (PDH) – Sistema de Tramas E1

Ano Letivo: 2012/2013

Elementos do Grupo:

João Dias nº 58738

Pedro Pacheco nº 55714

ABSTRACT

Este é um relatório resultante da implementação de um código em VHDL de um sistema de tramas E1.

De forma a que se tornasse possível implementar o sistema, dividimos os dois grandes blocos (Multiplexer e Desmultiplexer) em pequenos simples módulos, cada um com a sua função especifica, e que no seu conjunto realizam a função pretendida: apresentar na saída a informação inserida á entrada.

Os resultados em ambiente de simulação são resultantes do código em anexo, pelo que este se encontra pronto a simular na ferramenta ISim do ISE Design da Xilinx.

Os esquemáticos dos módulos implementados encontram-se junto com o código na secção de anexos

Ao longo do relatório referimo-nos a bloco como sendo, ou o bloco multiplexador ,ou o bloco desmultiplexador e a módulos como sendo os seus constituintes. Considere-se PAT como palavra de alinhamento de trama e PAMT como palavra de alinhamento de multitrama.

INDICE

LISTA DE TABELAS

LISTA DE FIGURAS

BLOCO I
BLOCO MULTIPLEXADOR

Descrição do bloco

O bloco multiplexador trata-se de um bloco com 30 entradas de dados a um ritmo de 2048 Kbps e uma saída de dados que é o resultado da multiplexagem destas entradas e dois grupos de 8 bits (duas tramas) extra que são adicionados para controlo, sinalização e alinhamento de trama.

As tramas são organizadas segundo uma estrutura conhecida, tal como mostrado na *Figura 1.* Podemos ver pela *Figura 1* que para além da estruturação de tramas (32 tramas de 8 bits), temos ainda a estrutura de multitrama. Nesta estrutura tem-se 16 canais de 256 bits (4096 bits).

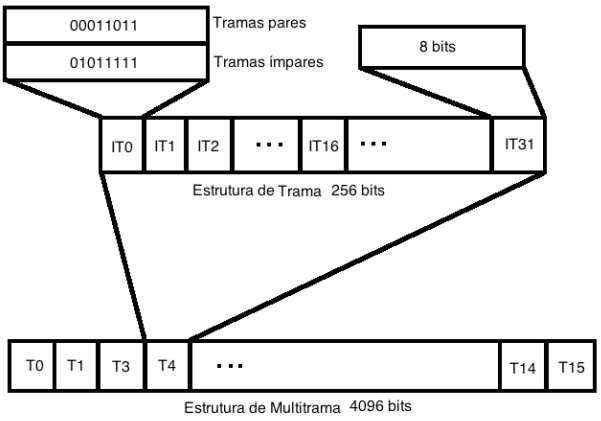


Figura 1 - Estrutura de trama e multitrama usadas

Módulos implementados

No bloco multiplexador foram implementados 6 módulos em VHDL. Cada um deles será descrito com auxilio de imagens da simulação e tabelas de verdade.

Contador de 8 bits

A funcionalidade deste bloco passa por contar o número de bits que entram no multiplexer, sendo que, ao atingir o seu valor máximo (b11111111), o sistema pode então perceber que as 32 tramas foram inseridas na linha.

Este bloco funciona como um pino select de um multiplexer tradicional, habilitando, neste caso, o canal que deve ser inserido na estrutura de dados a enviar.

Para além destas funcionalidades, este bloco é também usado para a construção da multitrama, visto que cada um dos blocos da multitrama (16 blocos) são constituídos por 32 tramas de dados. Com o auxilio de um contador de 4 bits (descrito na Página 7) é possível contar até ao fim da estrutura multitrama.

A implementação deste contador é bastante básica, pelo que ao ritmo de entrada de cada bit (usamos o CLOCK para simular a entrada de cada bit) é aumentada a sua contagem. Caso exista um 0 no pino de CLEAR a contagem volta a 0.

Na simulação foram obtidos os resultados da *Figura 2*, em que basicamente é demonstrada a tabela da verdade do contador.

Número de bits contados	***Saída do contador***
1	00000000
2	00000001
3	00000010
4	00000011
5	00000100
6	00000101
7	00000110
8	00000111
...	...
256	11111111

Tabela 1 – Tabela da verdade do contador de 8 bits

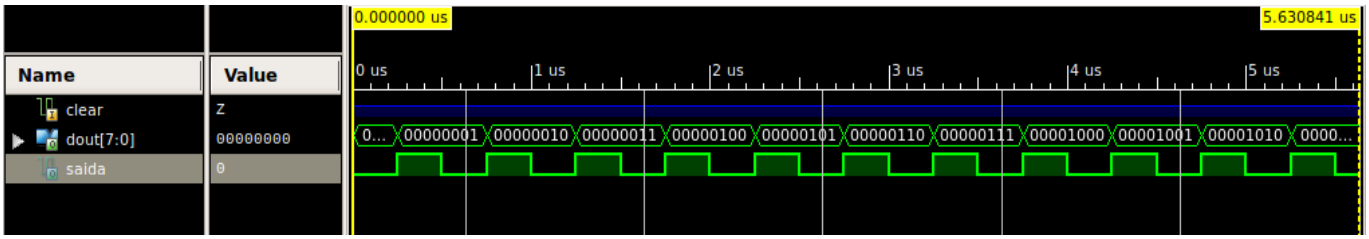


Figura 2 - Resultado da simulação do módulo contador de 8 bits

Legenda:

CLEAR– pino de clear do contador;

Dout[7:0] – saida do contador (8 bits); Saida – (da instancia clk(), clk.saida) CLOCK

Contador de 4 bits

Este contador, também bastante simples de implementar e com um funcionamento muito parecido com o de 8 bits, tem a função de contar até b1111, de forma a que seja possível construir a multitrama. A cada 32 tramas (b11111111 do contador de 8bits) este contador vai incrementar, pelo que será possível inserir dados referentes a multitrama neste instante. Para que seja possível outros blocos detetarem a tempo as alterações neste contador, este contador foi implementado com um CLOCK mais rápido que o CLOCK do contador de 8 bits, sendo assim, o valor da contagem deste contador será atualizada mais rapidamente. Tal será muito útil para o módulo Gerador de palavras de alinhamento.

Na simulação do contador da Fig.4 é possível observar o efeito do CLOCK mais rápido.

Número de bits contados	*Saída do contador*
1	0000
2	0001
3	0010
4	0011
5	0100
6	0101
7	0110
8	0111
...	...
16	1111

Tabela 2 – Tabela da verdade do contador de 4 bits

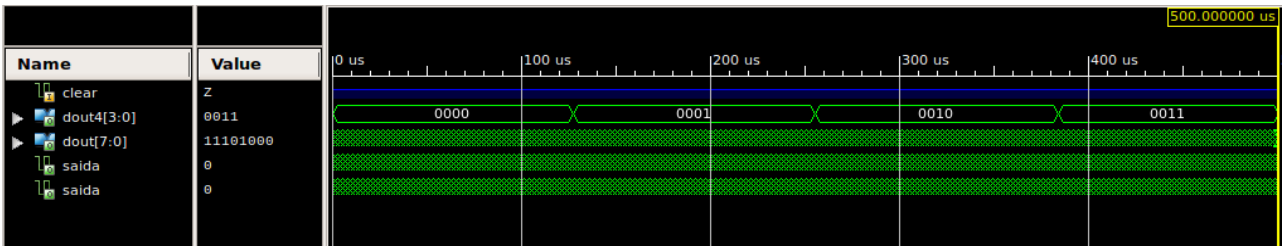


Figura 3 - Simulação do contador de 4bits

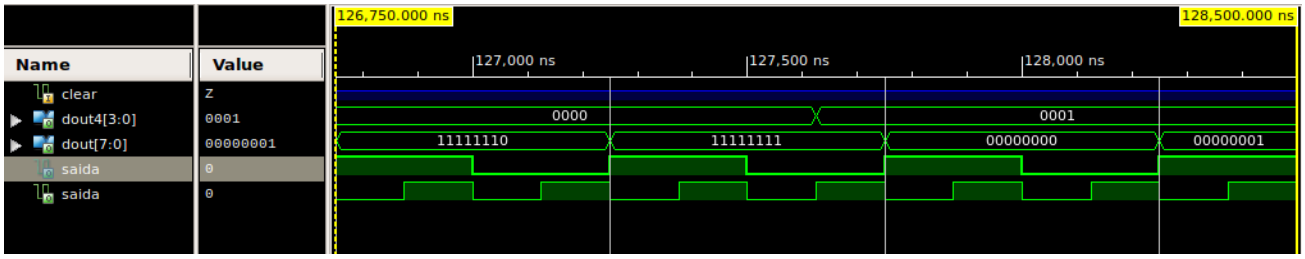


Figura 4 - Efeito da utilização de um CLOCK mais rápido

Legenda:

clear – pino de clear contador

Dout4[3:0] - saida contadorde 4 bits (4bits);

Dout[7:0] - saida contadorde 8 bits (8bits);

Saida – CLOCK (saida de baixo CLOCK de4MHz e saida de cima CLOCK de 2MHz)

Gerador de palavras de alinhamento

Neste bloco estão “armazenadas” as palavras de alinhamento a inserir conforme a contagem do contador de 4 bits. Consiste basicamente em uma condição com vários casos que ditam que a cada contagem diferente, do contador de 4 bits, teremos uma palavra de alinhamento determinada pela respectiva condição.

A saída deste módulo vai servir para o módulo de multiplexagem ter acesso á palavra de alinhamento indicada para o momento definido pela contagem.

As palavras de alinhamento são geradas de forma a serem pouco suscetiveis a erros, pelo que o seu segundo bit mais significativo alterna de trama em trama, ficando mais fácil prevenir erros na desmultiplexagem (Tabela 3).

Na Figura 6 é possível observar a funcionalidade de ter a contagem do contador de 4bits a uma frequência maior (dobro).

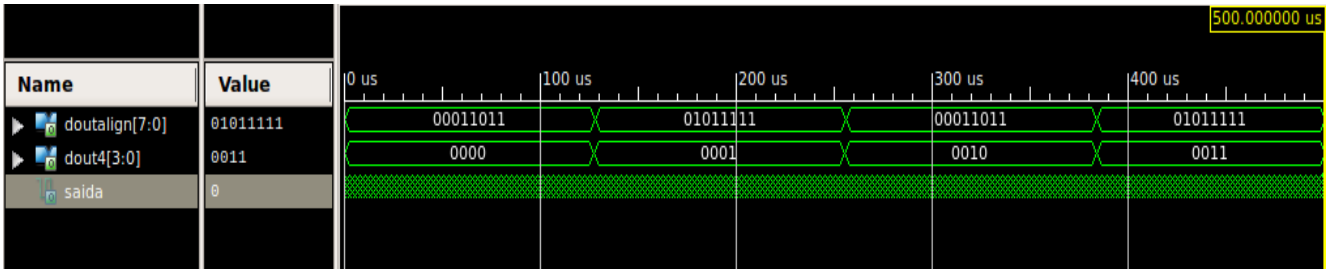


Figura 5 - Simulação do módulo gerador de palavras de alinhamento

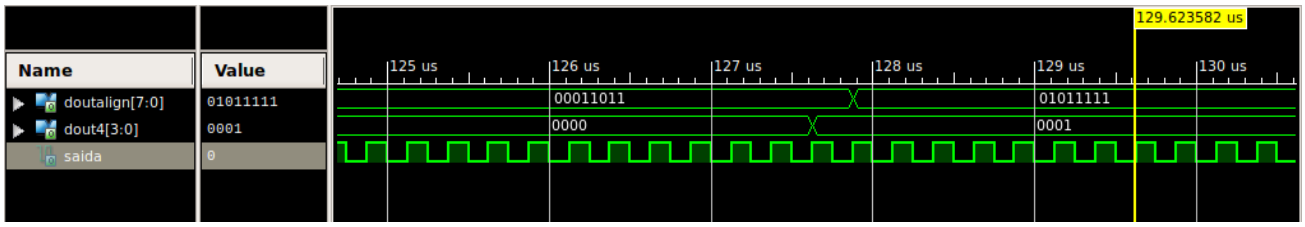


Figura 6 - Efeito do CLOCK mais rápido no módulo gerador de palavras de alinhamento

Legenda:

Doutalign[7:0] – Saída do módulo de alinhamento (8bits);

Dout4[3:0] - saida contadorde 4 bits (4bits);

Saida – (da instancia clk(), clk.saida) CLOCK

Trama da multitrama	*Bits do canal 0 da estrutura de tramas (IT0)*
T0	00011011
T1	01011111
T2	00011011
T3	01011111
T4	00011011
T5	11011111
T6	00011011
T7	01011111
T8	00011011
T9	11011111
T10	00011011
T11	1101111
T12	00011011
T13	01011111
T14	00011011
T15	01011111

Tabela 3 – Palavras de alinhamento para cada canal da multitrama

Gerador de tramas

Este bloco tem o simples propósito de fixar os pinos do multiplexer a um valor fixo para que seja possível obter leituras intuitivas ao longo do processo.

As tramas geradas por este módulo vão ser injetadas na estrutura de trama e multitrama pelo módulo multiplexador e são estes os dados que pretendemos obter na saída do bloco desmultiplexador.

Canal Gerado	*Bits do canal 0 da estrutura de tramas (IT0)*
IT1	11100001
IT2	11100010
IT3	11100011
IT4	11100100
IT5	11100101
IT6	11100110
IT7	11100111
IT8	11101000
IT9	11101001
IT10	11101010
IT11	11101011
IT12	11101100
IT13	11101101
IT14	11101110
IT15	11101111
IT17	11110001
IT18	11110010
IT19	11110011
IT20	11110100
IT21	11110101
IT22	11110110
IT23	11110111
IT24	11111000
IT25	11111001
IT26	11111010
IT27	11111011
IT28	11111100
IT29	11111101
IT30	11111110
IT31	11111111

Tabela 4 – Tramas geradas para os 30 canais de dados

Multiplexagem

Este módulo é responsável por organizar as tramas e as respetivas multitramas, apresentando á sua saída um valor de 8 bits que será depois transformado num valor série pelo registo de deslocamento.

O princípio de funcionamento deste bloco é bastante simples. Conforme a contagem do contador de 8bits este bloco vai selecionar o respetivo dado que deve entrar na sequência de tramas, quer seja uma palavra de alinhamento, quer seja o IT16 (gerado neste modulo), ou dados de qualquer um canal de informação.

Saída do contador de 8 bits	*Bit selecionado para entrar na trama*
00000000	Bit alinhamento
00000001	Bit alinhamento
00000010	Bit alinhamento
00000011	Bit alinhamento
00000100	Bit alinhamento
00000101	Bit alinhamento
00000110	Bit alinhamento
00000111	Bit alinhamento
00001000	Bit do canal IT1 (informação)
00001001	Bit do canal IT1 (informação)
00001010	Bit do canal IT1 (informação)
00001011	Bit do canal IT1 (informação)
00001100	Bit do canal IT1 (informação)
00001101	Bit do canal IT1 (informação)
00001110	Bit do canal IT1 (informação)
00001111	Bit do canal IT1 (informação)
00010000	Bit do canal IT2 (informação)
...	...
11111111	Bit do canal IT31 (informação)

Tabela 5 – Variação da saída do módulo multiplexador com a saída do contador de 8 bits

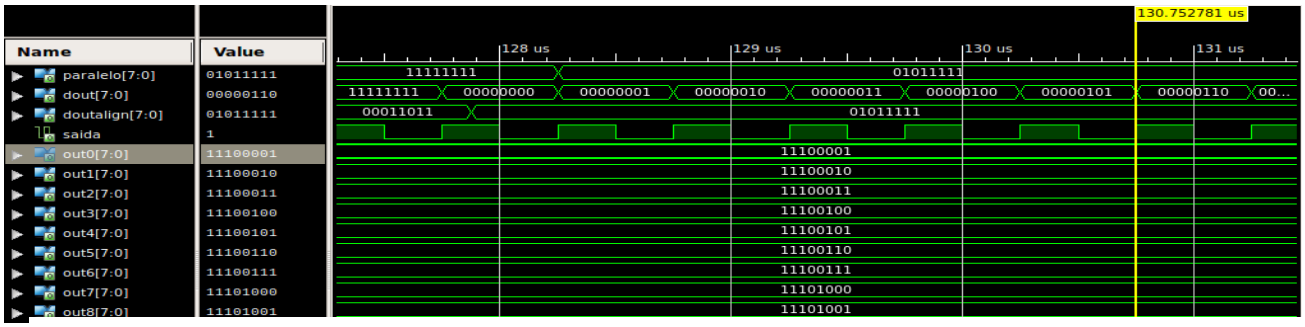


Figura 7 - Inserção da palavra de alinhamento na saída do bloco (doutalign)

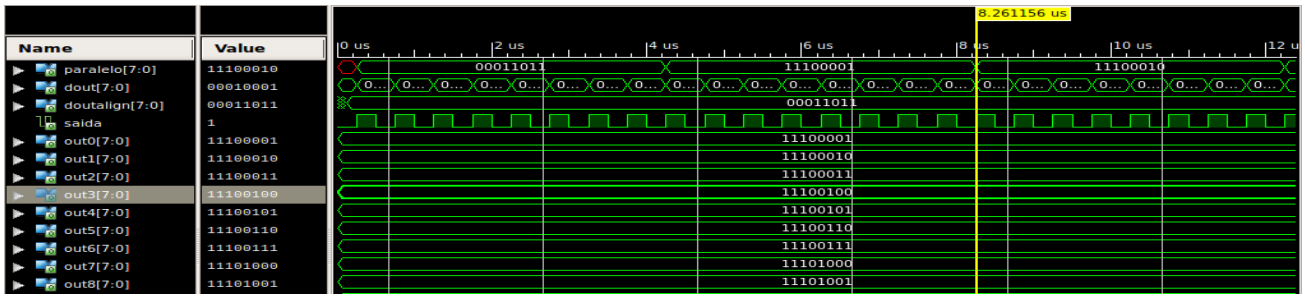


Figura 8 - Inserção de IT0, IT1 e IT2 na estrutura de tramas

Legenda:

Doutalign[7:0] – Saída do módulo de alinhamento (8bits);

Dout[7:0] - saida contadorde 8 bits (8bits);

Saida – (da instancia clk(), clk.saida) clock;

Paralelo[7:0] – Saida em paralelo do mux (8bits);

Outx[7:0] – Canais de info provenientes de “gerador”.

Shift Register (PISO)

Neste módulo implementamos a passagem de paralelo para série, de forma a que se torne possível transmitir dados através de uma linha.

A conversão é feita com a ajuda de uma variável que incrementa a cada variação do CLOCK, que coincide exatamente com o local onde devemos ler o bit da trama. A leitura é feita do bit mais para o menos significativo.

No programa introduzimos um atraso de 1ns a cada ciclo de CLOCK, pois estávamos a experienciar alguns problemas com a leitura, que como é tão exata, no momento de transição não tinha capacidade de saber qual o dado que estava a ler (motivo porque também foi adicionado um CLOCK mais rápido em módulos anteriores). Com a introdução deste nano segundo de atraso ficou possível fazer a conversão.

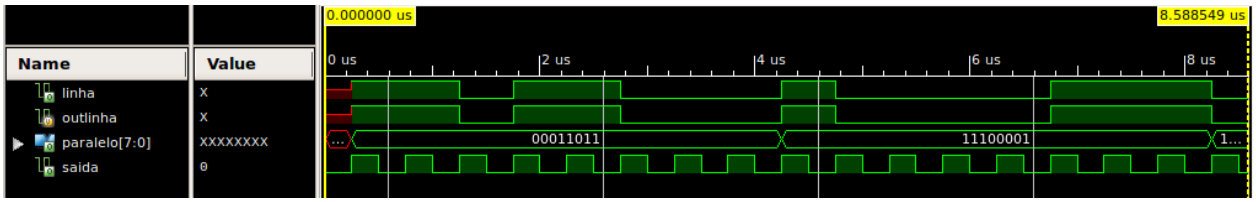


Figura 9 - Conversão paralelo-série dos canais IT0 e IT1

Legenda:

outlinha = linha – Saída do bloco multiplexador (1bit);

Saida – (da instancia clk(), clk.saida) CLOCK;

Paralelo[7:0] – Saida em paralelo do módulo multiplexador (8bits);

BLOCO II
BLOCO DESMULTIPLEXADOR

Descrição do bloco

O bloco desmultiplexador é capaz de receber os dados da linha (série) e depois de verificar, através de uma máquina de estados, se os dados são os esperados, oferece na saída as tramas geradas pelo módulo gerador de tramas do bloco multiplexador.

Módulos implementados

Neste bloco foram implementados 6 módulos em VHDL e são descritos com o auxilio de figuras da simulação e tabelas da verdade.

Contadores (8 bits e 4bits) e gerador de palavras de alinhamento

Estes módulos são bastante semelhantes aos módulos do bloco multiplexador, desempenhando a mesma função, mas neste caso no bloco desmultiplexador.

Shift Register (SIPO)

Este módulo, como é de esperar, é bastante semelhante ao PISO.

De forma a conseguir fazer a conversão série/paralelo usámos uma espécie de um *buffer* que armazena temporariamente os dados e que vai atualizando o seu valor á medida que recebe dados série. O *buffer* é capaz de ao fim dos oito ciclos de CLOCK conter o valor série convertido para paralelo, sendo este posteriormente enviado para a saída do módulo.

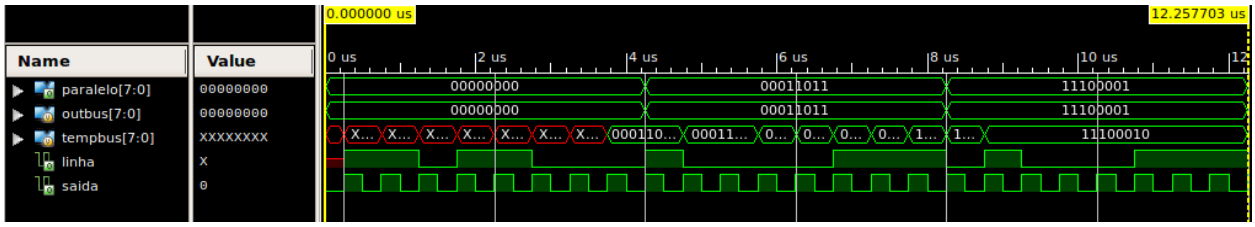


Figura 10 - Conversão Série-Paralelo do IT0, IT1 e IT2

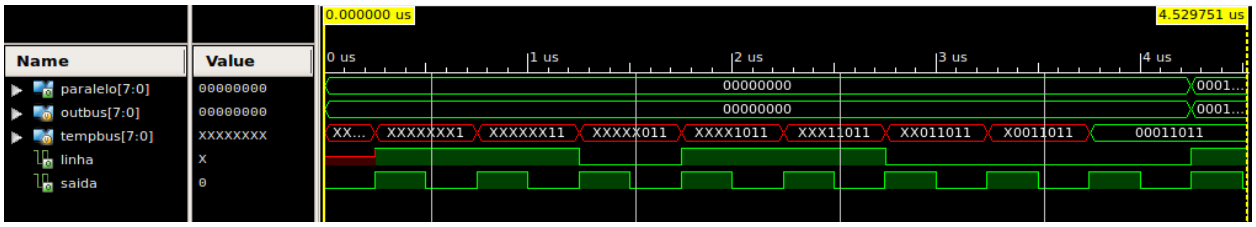


Figura 11 - Atualização do *buffer* temporário

Legenda:

linha – Saída do bloco multiplexador (1bit);

Saida – (da instancia clkd(), clkd.saida) clock;

Paralelo[7:0] = outbus[7:0] – Saida em paralelo do SIPO (8bits);

Tempbus[7:0] – buffer temporário.

Comparador e máquina de estados

É neste módulo que é feita a verificação dos dados recebidos no bloco desmultiplexador (PAT e PAMT).

Para verificar se a PAT é a correta e se encontra na posição esperada usamos a máquina de estados da Figura 12.

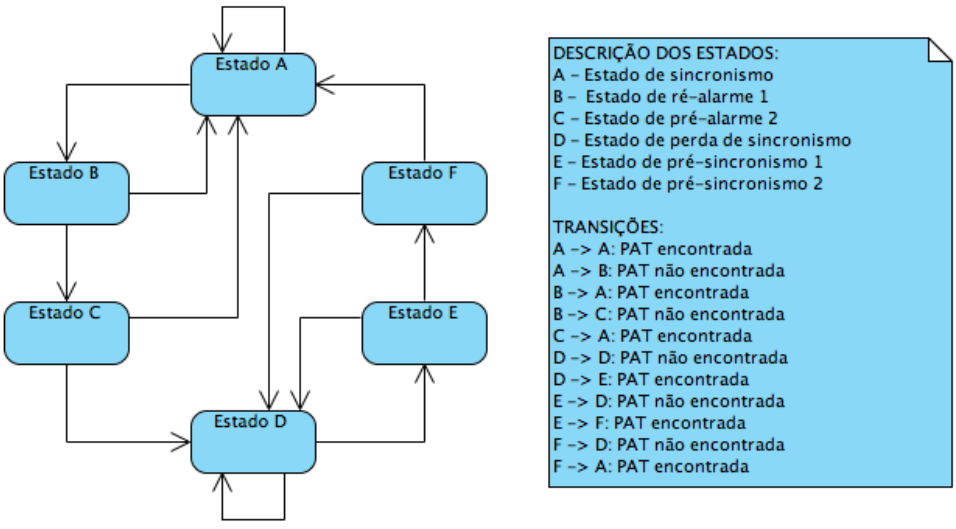


Figura 12 – Máquina de estados para detecção da PAT

Ao iniciar a máquina de estados partimos da suposição que o sistema se encontra em sincronismo e vamos testando esse estado ao longo do tempo.

O sistema só perde o sincronismo ao atingir o estado D.

O alinhamento da PAMT é feito com base neste alinhamento, pois com a PAT alinhada e com o auxilio dos contadores é possível saber onde esta se encontra e fazer a verificação. Nas Figuras 13, 14 e 15 está demonstrado o funcionamento da máquina de estados em ambiente de simulação e ainda alguns testes realizados ao módulo.

Na Figura 13 podemos observar a máquina de estados em pleno sincronismo (estado A), pois as PAT que estão a ser recebidas são as corretas e estão a ser recebidas no tempo esperado.

Na Figura 14 podemos observar uma perda de sincronismo do bloco desmultiplexador (após forçarmos o valor da PAT enviada pelo bloco multiplexador a b00000000) e a respetiva passagem da máquina de estados pelo estado B e C antes de atingir o D (estado de perda de sincronismo).

Na Figura 15 observa-se uma recuperação de sincronismo do sistema, após voltar a ser enviada a PAT esperada. De referir a recuperação da PAMT, representada na mesma figura.

De forma a conseguir simular a posição dos estados no software criamos uma variável do tipo *integer* (variável aux das Figuras 13, 14 e 15). Lendo o seu valor conseguimos saber qual o estado do sistema (A, B, C, D, E ou F) A=000; B=001; C=010; D=011; E=100; F=101.

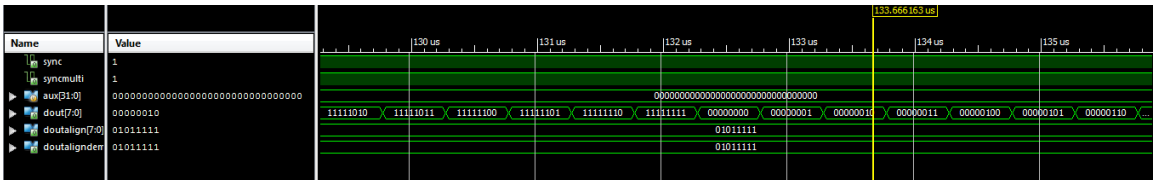


Figura 13 – Máquina de estados em sincronismo (Estado A)

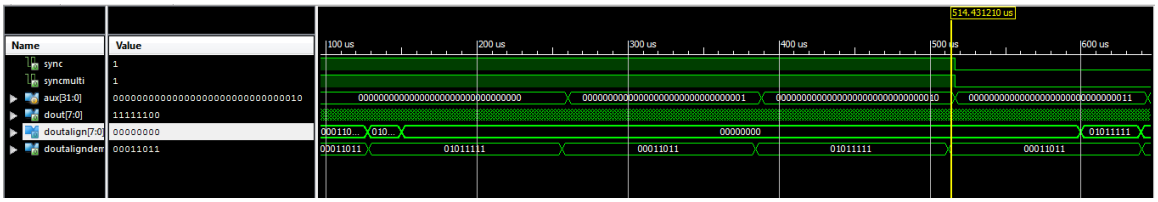


Figura 14 – Máquina de estado perde sincronismo após passar pelo estado B e C

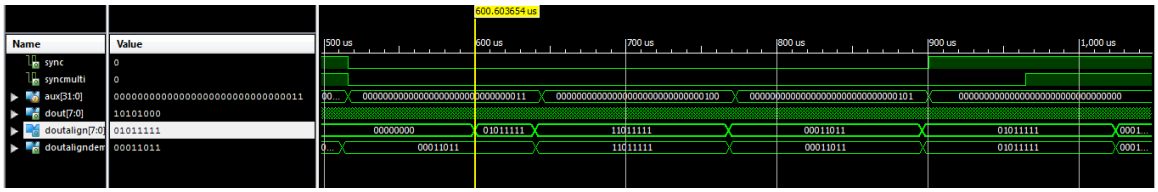


Figura 15 – Máquina de estados volta a ganhar sincronismo (trama e multitrama)

Legenda:

sync – Sincronismo da palavra de alinhamento de trama (1bit);

syncmulti – Sincronismo da palavra de alinhamento de multitrama (1bit);

Saida – (da instancia clkd(), clkd.saida) CLOCK;

doutalign[7:0] – Saida do bloco onde estão guardadas as palavras de alinhamento no bloco multimplexador(8bits);

doutalignde mux[7:0] – Saida do bloco onde estão guardadas as palavras de alinhamento no bloco desmultimplexador(8bits);

dout[7:0] – saida do contador de 8 bits (8bits).

aux[31:0] –variavel que simula os estados.

Modulo Desmultiplexador

Este módulo funciona de maneira a que, conforme o resultado da contagem do contador de 8 bits, os canais que contêm informação útil sejam reencaminhados para a saída do bloco desmultiplexador.

Ao longo do processo são excluídas tramas que não interessam na saída, caso do IT0 e do IT16, que são apenas tramas importantes para controlo e não contêm qualquer informação útil para a saída do sistema.

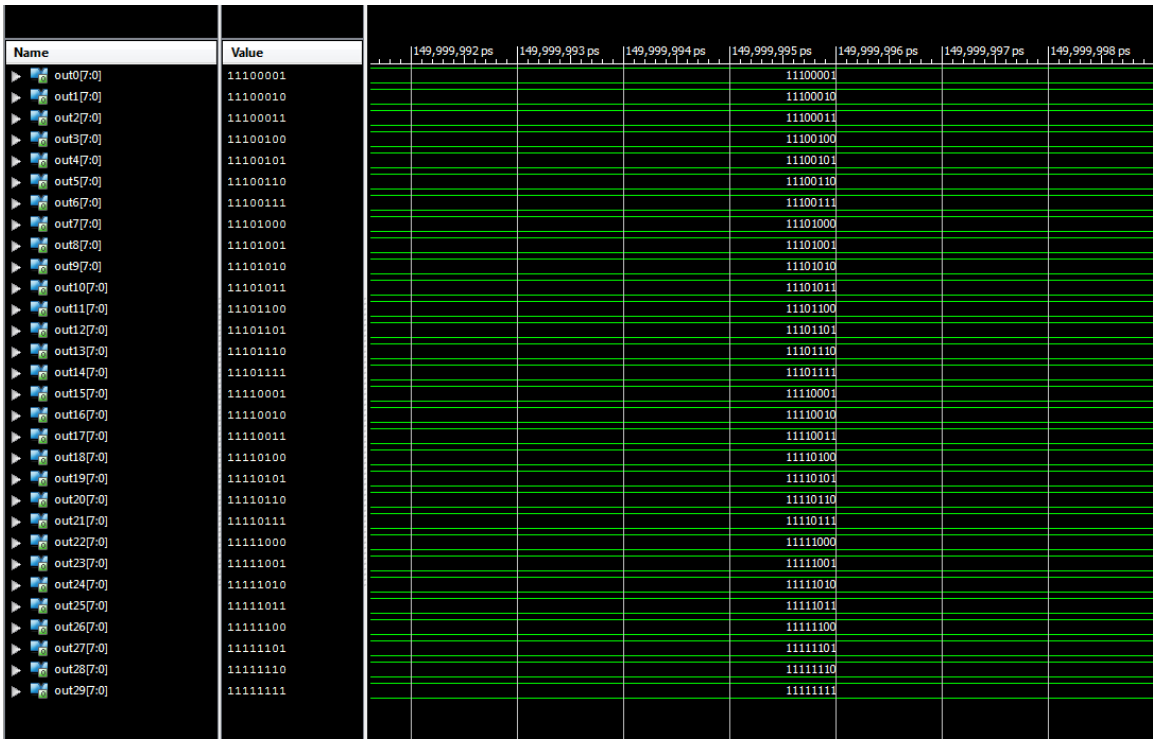


Figura 16 – Saída do bloco desmultiplexador (30 canis de informação útil)

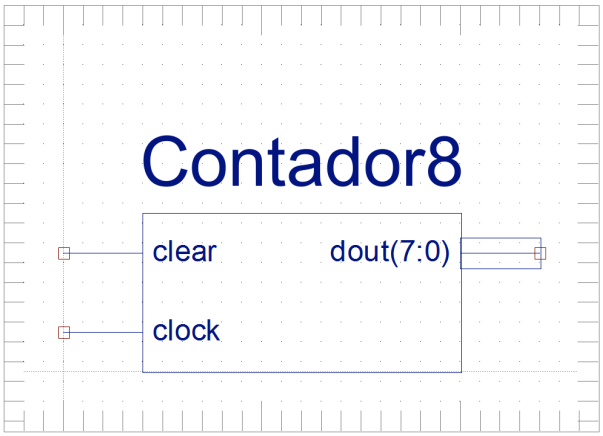
Legenda:

outx[7:0] – Saída do bloco desmultiplexador (1bit);

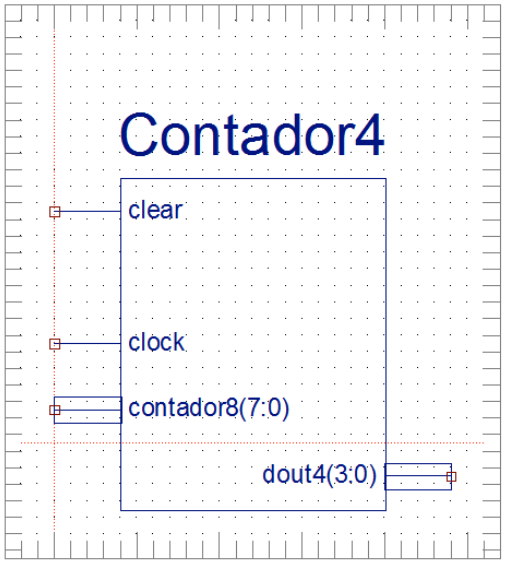
ANEXOS

ESQUEMÁTICOS DOS MODULOS

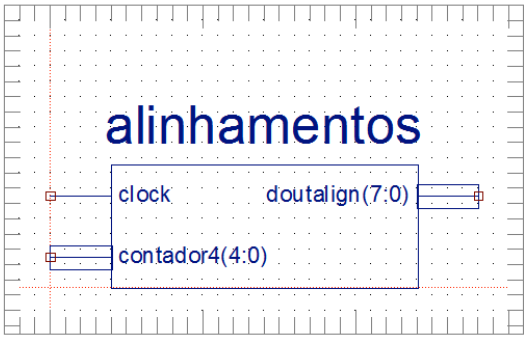
Contador de 8 bits (bloco multiplexador e desmultiplexador)



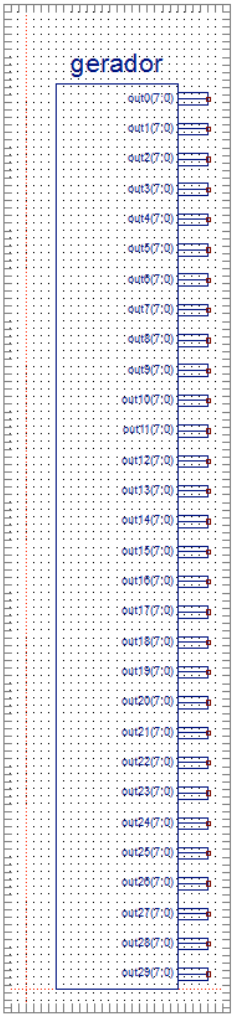
Contador de 4 bits (bloco multiplexador e desmultiplexador)



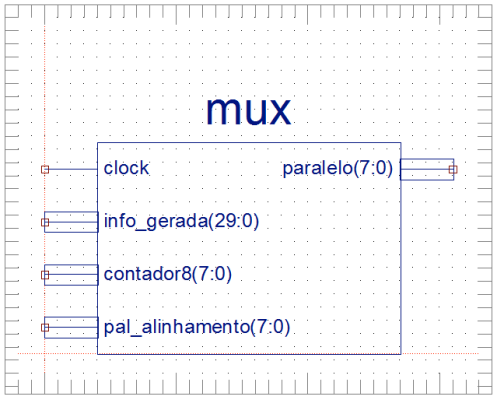
Gerador de palavras de alinhamento



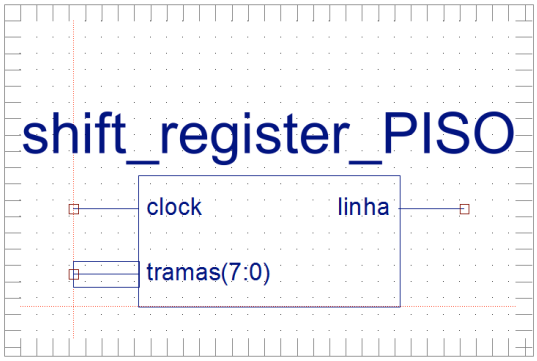
Gerador de canais de informação



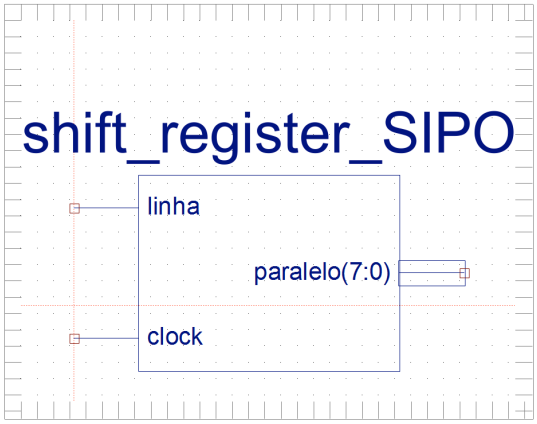
Módulo multiplexador



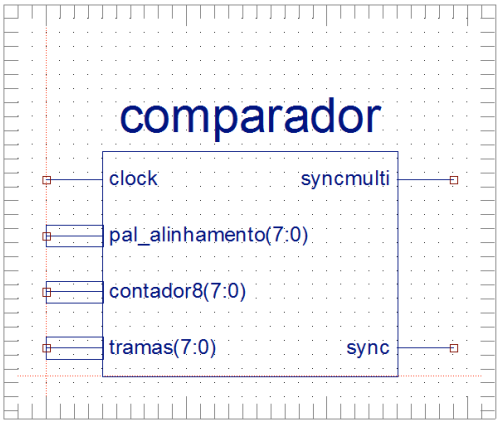
Shift Register (PISO)



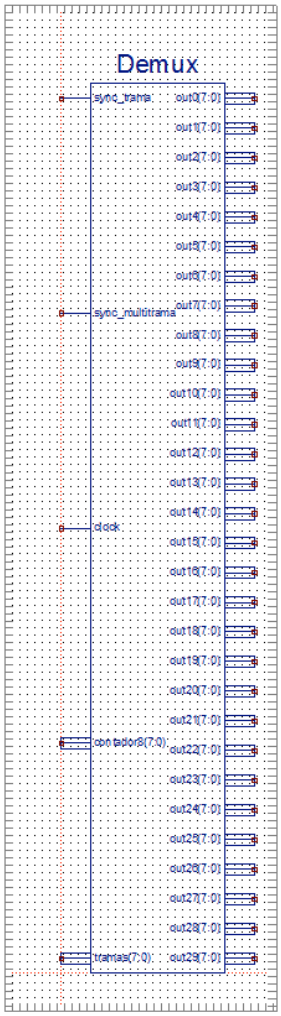
Shift Register (SIPO)



Módulo comparador e máquina de estados



Módulo desmultiplexador



CODIGO VERILOG

Clock 2Mbps

```
module clk_2Mbps (saida, inicio)
                output wire saida;
                        output wire inicio;
                reg out;
                        reg out1;

   assign saida = out;
            assign inicio = out1;
initial
begin
   out = 0;
            out1 = 1;
end
always
begin
  #250 out = ~out;
           out1 = 0;
end
endmodule
```

Clock 4 Mbps

```
module clk_4Mbps (saida, inicio)
                output wire saida;
                        output wire inicio;
                reg out;
                        reg out1;

   assign saida = out;
            assign inicio = out1;
initial
begin
   out = 0;
            out1 = 1;
end
always
begin
  #125 out = ~out;
            out1 = 0;
end
endmodule
```

Contador 8 bits

```
module Contador8(clear, dout);

input   clear;
output wire [7:0] dout;

reg     [7:0] cnt;

clk_2Mbps clk();

initial
begin
       cnt = 8'h00;
end

assign dout = cnt;

always @ (posedge clk.saida)
begin
   if (!clear)
      cnt = 8'h00;
   else
      cnt = cnt + 1;
end
endmodule
```

Contador 4 bits

```
module Contador4(clear, dout4);

input   clear;
output wire [3:0] dout4;

reg     [3:0] cnt;

clk_4Mbps clk();
Contador8 cnt8();

initial
begin
cnt = 4'h00;
end

assign dout4 = cnt;


always @ (posedge clk.saida)
begin

   if (!clear)
      cnt = 4'h00;
   else if(cnt8.dout == 8'hff)
      cnt = cnt + 1;
                        #251;
end
endmodule
```

Gerador de palavras de alinhamento

```
module alinhamentos (doutalign);

  output wire [7:0]doutalign

  reg [7:0] tramas;

  clk_4Mbps clk();
  Contador4 cnt4();

  initial
  begin
    tramas = 8'h00;
  end

  assign doutalign = tramas;

  always @ (posedge clk.saida)
  begin

   case(cnt4.dout4)
    4'h00: tramas = 8'h1b;
    4'h02: tramas = 8'h1b;
    4'h04: tramas = 8'h1b;
    4'h06: tramas = 8'h1b;
    4'h08: tramas = 8'h1b;
    4'h0a: tramas = 8'h1b;
    4'h0c: tramas = 8'h1b;
    4'h0e: tramas = 8'h1b;

    4'h01: tramas = 8'h5f;
    4'h03: tramas = 8'h5f;
    4'h07: tramas = 8'h5f;
    4'h0d: tramas = 8'h5f;
    4'h0f: tramas = 8'h5f;

    4'h05: tramas = 8'hdf;
    4'h09: tramas = 8'hdf;
    4'h0b: tramas = 8'hdf;
    default: tramas = tramas;
   endcase

 end
endmodule
```

Gerador de tramas de informação

```
module gerador (out0, out1, out2, out3, out4, out5, out6, out7, out8, out9, out10, out11,
out12, out13, out14, out15, out16, out17, out18, out19, out20, out21, out22, out23,
out24, out25,  out26, out27, out28, out29);

output wire [7:0] out0;
output wire [7:0] out1;
output wire [7:0] out2;
output wire [7:0] out3;
output wire [7:0] out4;
output wire [7:0] out5;
output wire [7:0] out6;
output wire [7:0] out7;
output wire [7:0] out8;
output wire [7:0] out9;
output wire [7:0] out10;
output wire [7:0] out11;
output wire [7:0] out12;
output wire [7:0] out13;
output wire [7:0] out14;
output wire [7:0] out15;
output wire [7:0] out16;
output wire [7:0] out17;
output wire [7:0] out18;
output wire [7:0] out19;
output wire [7:0] out20;
output wire [7:0] out21;
output wire [7:0] out22;
output wire [7:0] out23;
output wire [7:0] out24;
output wire [7:0] out25;
output wire [7:0] out26;
output wire [7:0] out27;
output wire [7:0] out28;
output wire [7:0] out29;
```

```
reg [7:0] teste0;
reg [7:0] teste1;
reg [7:0] teste2;
reg [7:0] teste3;
reg [7:0] teste4;
reg [7:0] teste5;
reg [7:0] teste6;
reg [7:0] teste7;
reg [7:0] teste8;
reg [7:0] teste9;
reg [7:0] teste10;
reg [7:0] teste11;
reg [7:0] teste12;
reg [7:0] teste13;
reg [7:0] teste14;
reg [7:0] teste15;
reg [7:0] teste16;
reg [7:0] teste17;
reg [7:0] teste18;
reg [7:0] teste19;
reg [7:0] teste20;
reg [7:0] teste21;
reg [7:0] teste22;
reg [7:0] teste23;
reg [7:0] teste24;
reg [7:0] teste25;
reg [7:0] teste26;
reg [7:0] teste27;
reg [7:0] teste28;
reg [7:0] teste29;
```

```
initial begin

   teste0 = 8'he1;
   teste1 = 8'he2;
   teste2 = 8'he3;
   teste3 = 8'he4;
   teste4 = 8'he5;
   teste5 = 8'he6;
   teste6 = 8'he7;
   teste7 = 8'he8;
   teste8 = 8'he9;
   teste9 = 8'hea;
   teste10 = 8'heb;
   teste11 = 8'hec;
   teste12 = 8'hed;
   teste13 = 8'hee;
   teste14 = 8'hef;
   teste15 = 8'hf1;
   teste16 = 8'hf2;
   teste17 = 8'hf3;
   teste18 = 8'hf4;
   teste19 = 8'hf5;
   teste20 = 8'hf6;
   teste21 = 8'hf7;
   teste22 = 8'hf8;
   teste23 = 8'hf9;
   teste24 = 8'hfa;
   teste25 = 8'hfb;
   teste26 = 8'hfc;
   teste27 = 8'hfd;
   teste28 = 8'hfe;
   teste29 = 8'hff;

end
```

```
assign out0 = teste0;
assign out1 = teste1;
assign out2 = teste2;
assign out3 = teste3;
assign out4 = teste4;
assign out5 = teste5;
assign out6 = teste6;
assign out7 = teste7;
assign out8 = teste8;
assign out9 = teste9;
assign out10 = teste10;
assign out11 = teste11;
assign out12 = teste12;
assign out13 = teste13;
assign out14 = teste14;
assign out15 = teste15;
assign out16 = teste16;
assign out17 = teste17;
assign out18 = teste18;
assign out19 = teste19;
assign out20 = teste20;
assign out21 = teste21;
assign out22 = teste22;
assign out23 = teste23;
assign out24 = teste24;
assign out25 = teste25;
assign out26 = teste26;
assign out27 = teste27;
assign out28 = teste28;
assign out29 = teste29;


endmodule
```

Módulo multiplexador

```
module mux (paralelo);

  output wire [7:0]paralelo;
clk_2Mbps clk();
Contador8 cnt8();
alinhamentos algn();
gerador ger();

reg [7:0] trama16;
reg [7:0] out;

initial
begin //inicialização
   //pre definição do it16
   trama16 = 8'h0b;
end

assign paralelo = out;

always @(posedge clk.saida) begin

if(cnt8.dout <= 8'h08) begin
     out = algn.doutalign;
end

if(cnt8.dout <= 8'h10 && cnt8.dout>=8'h08) begin
     out =ger.out0;
end

if (cnt8.dout >= 8'h10 && cnt8.dout < 8'h18) begin //canal2
      out =ger.out1;
end

if (cnt8.dout >= 8'h18 && cnt8.dout < 8'h20) begin //canal3
      out =ger.out2;
end

if (cnt8.dout >= 8'h20 && cnt8.dout < 8'h28) begin  //canal4
      out =ger.out3;
end

if (cnt8.dout >= 8'h28 && cnt8.dout < 8'h30) begin //canal5
      out =ger.out4;
end
```

```
if (cnt8.dout >= 8'h30 && cnt8.dout < 8'h38) begin //canal6
     out =ger.out5;
end

if (cnt8.dout >= 8'h38 && cnt8.dout < 8'h40) begin //canal7
     out =ger.out6;
end

if (cnt8.dout >= 8'h40 && cnt8.dout < 8'h48) begin //canal8
     out =ger.out7;
end

if (cnt8.dout >= 8'h48 && cnt8.dout < 8'h50) begin //canal9
     out =ger.out8;
end

if (cnt8.dout >= 8'h50 && cnt8.dout < 8'h58) begin //canal10
     out =ger.out9;
end

if (cnt8.dout >= 8'h58 && cnt8.dout < 8'h60) begin //canal11
     out = ger.out10;
end

if (cnt8.dout >= 8'h60 && cnt8.dout < 8'h68) begin //canal12
     out = ger.out11;
end

if (cnt8.dout >= 8'h68 && cnt8.dout < 8'h70) begin //canal13
     out = ger.out12;
end

if (cnt8.dout >= 8'h70 && cnt8.dout < 8'h78) begin //canal14
     out = ger.out13;
end

if (cnt8.dout >= 8'h78 && cnt8.dout < 8'h80) begin //canal15
     out = ger.out14;
end

if (cnt8.dout >= 8'h80 && cnt8.dout < 8'h88) begin // it16
     out = trama16;
end
```

```
if (cnt8.dout >= 8'h88 && cnt8.dout < 8'h90) begin // canal17

    out = ger.out15;
end

if (cnt8.dout >= 8'h90 && cnt8.dout < 8'h98) begin // canal18
    out = ger.out16;
end

if (cnt8.dout >= 8'h98 && cnt8.dout < 8'hA0) begin // canal19
    out = ger.out17;
end

if (cnt8.dout >= 8'hA0 && cnt8.dout < 8'hA8) begin // canal20
    out = ger.out18;
end

if (cnt8.dout >= 8'hA8 && cnt8.dout < 8'hB0) begin // canal21
    out = ger.out19;
end

if (cnt8.dout >= 8'hB0 && cnt8.dout < 8'hB8) begin // canal22
    out = ger.out20;
end

if (cnt8.dout >= 8'hB8 && cnt8.dout < 8'hC0) begin // canal23
    out = ger.out21;
end

if (cnt8.dout >= 8'hC0 && cnt8.dout < 8'hC8) begin // canal24
    out = ger.out22;
end

if (cnt8.dout >= 8'hC8 && cnt8.dout < 8'hD0) begin // canal25
    out = ger.out23;
end

if (cnt8.dout >= 8'hD0 && cnt8.dout < 8'hD8) begin // canal26
    out = ger.out24;
end

if (cnt8.dout >= 8'hD8 && cnt8.dout < 8'hE0) begin // canal27
    out = ger.out25;
end
```

```
if (cnt8.dout >= 8'hE0 && cnt8.dout < 8'hE8) begin // canal28
     out = ger.out26;
end

if (cnt8.dout >= 8'hE8 && cnt8.dout < 8'hF0) begin // canal29
     out = ger.out27;
end

if (cnt8.dout >= 8'hF0 && cnt8.dout < 8'hF8) begin // canal30
     out = ger.out28;
end

if (cnt8.dout >= 8'hF8 && cnt8.dout <= 8'hFF) begin // canal31
     out = ger.out29;
end

end
endmodule
```

Shift Register (PISO)

```
module shift_register_PISO (linha);

output wire linha;
integer i;
reg outlinha;

clk_2Mbps clk();
mux mux();
initial
begin
  i=0;
end

  assign linha = outlinha;

  always @(posedge clk.saida)
  begin
    #1;
    if(i ==8) begin
       i = 0;
    end
    outlinha = mux.paralelo[(0+i)];
    i = i+1;
  end
endmodule
```

Shift Resgister (SIPO)

```
module shift_register_SIPO (paralelo);

output wire [7:0]paralelo;
integer i;
reg [7:0]outbus;
reg [7:0]tempbus;

clockdemux clkd();
shift_register_PISO srPISO();
initial
begin
 i=0;
 outbus = 8'h00;
end

 assign paralelo = outbus;

 always @(posedge clkd.saida)
 begin
  #2;
  if(i ==8) begin
     i = 0;
     outbus = tempbus;
  end
  tempbus[(0+i)] = srPISO.linha;
  i = i+1;
 end
endmodule
```

Comparador e máquina de estados

```
module comparador (syncmulti, sync);

output wire sync;
output wire syncmulti;

integer aux;
reg synctrama;
reg syncmultitrama;

clockdemux4M clkd();
shift_register_SIPO srSIPO();
aligndemux algnd();
cnt8demux cnt8();

assign sync = synctrama;
assign syncmulti = syncmultitrama;

initial
begin
   aux = 0;     //aux == 0 estado A; aux==1 estado B; aux=2 estado C; aux=3 estado D;
aux=4 estado E; aux=5 estado F;
   synctrama = 1;
   syncmultitrama = 1;
end
```

```
always @(cnt8.dout==8'h00 || cnt8.dout==8'h80) begin


  if(cnt8.dout==8'h00) begin  //maquina de estados

  case(aux)
    0:
      if(srSIPO.paralelo != algnd.doutalign) begin
        aux=1;
      end
    1:
      if(srSIPO.paralelo != algnd.doutalign) begin
        aux = 2;
      end
      else begin
        aux = 0;
      end
    2:
      if(srSIPO.paralelo != algnd.doutalign) begin
        aux = 3;
        synctrama = 0;
        syncmultitrama = 0;
      end
      else begin
        aux = 0;
      end
    3:
      if(srSIPO.paralelo == algnd.doutalign) begin
        aux=4;
      end
     else begin
       synctrama = 0;
       syncmultitrama = 0;
      end
    4:
      if(srSIPO.paralelo == algnd.doutalign) begin
        aux = 5;
      end
      else begin
       aux = 3;
     end
```

```
5:
        if(srSIPO.paralelo == algnd.doutalign) begin
           aux = 0;
           synctrama = 1;
          syncmultitrama = 0;
        end
        else begin
          aux = 3;
        end
      endcase
   end
   else if(cnt8.dout==8'h80 && aux==0) begin // caso esteja em sincronismo podemos
detectar a posiçºao da PAMT
      if(srSIPO.paralelo == 8'h0b) begin
        syncmultitrama = 1;
      end
      else begin
        syncmultitrama = 0;
      end
          end
          end
endmodule
```

Módulo desmultiplexador

```
module Demux (out0, out1, out2, out3, out4, out5, out6, out7, out8, out9, out10, out11,
out12, out13, out14, out15, out16, out17, out18, out19, out20, out21, out22, out23,
out24, out25,  out26, out27, out28, out29);

output wire [7:0] out0;
output wire [7:0] out1;
output wire [7:0] out2;
output wire [7:0] out3;
output wire [7:0] out4;
output wire [7:0] out5;
output wire [7:0] out6;
output wire [7:0] out7;
output wire [7:0] out8;
output wire [7:0] out9;
output wire [7:0] out10;
output wire [7:0] out11;
output wire [7:0] out12;
output wire [7:0] out13;
output wire [7:0] out14;
output wire [7:0] out15;
output wire [7:0] out16;
output wire [7:0] out17;
output wire [7:0] out18;
output wire [7:0] out19;
output wire [7:0] out20;
output wire [7:0] out21;
output wire [7:0] out22;
output wire [7:0] out23;
output wire [7:0] out24;
output wire [7:0] out25;
output wire [7:0] out26;
output wire [7:0] out27;
output wire [7:0] out28;
output wire [7:0] out29;
```

```
reg [7:0] teste0;
reg [7:0] teste1;
reg [7:0] teste2;
reg [7:0] teste3;
reg [7:0] teste4;
reg [7:0] teste5;
reg [7:0] teste6;
reg [7:0] teste7;
reg [7:0] teste8;
reg [7:0] teste9;
reg [7:0] teste10;
reg [7:0] teste11;
reg [7:0] teste12;
reg [7:0] teste13;
reg [7:0] teste14;
reg [7:0] teste15;
reg [7:0] teste16;
reg [7:0] teste17;
reg [7:0] teste18;
reg [7:0] teste19;
reg [7:0] teste20;
reg [7:0] teste21;
reg [7:0] teste22;
reg [7:0] teste23;
reg [7:0] teste24;
reg [7:0] teste25;
reg [7:0] teste26;
reg [7:0] teste27;
reg [7:0] teste28;
reg [7:0] teste29;


clockdemux clkd();
cnt8demux cnt8d();
comparador cmp();
shift_register_SIPO srSIPO();
```

```
assign out0 = teste0;
assign out1 = teste1;
assign out2 = teste2;
assign out3 = teste3;
assign out4 = teste4;
assign out5 = teste5;
assign out6 = teste6;
assign out7 = teste7;
assign out8 = teste8;
assign out9 = teste9;
assign out10 = teste10;
assign out11 = teste11;
assign out12 = teste12;
assign out13 = teste13;
assign out14 = teste14;
assign out15 = teste15;
assign out16 = teste16;
assign out17 = teste17;
assign out18 = teste18;
assign out19 = teste19;
assign out20 = teste20;
assign out21 = teste21;
assign out22 = teste22;
assign out23 = teste23;
assign out24 = teste24;
assign out25 = teste25;
assign out26 = teste26;
assign out27 = teste27;
assign out28 = teste28;
assign out29 = teste29;
```

```
always @ (srSIPO.paralelo) begin

if (cnt8d.dout >= 8'h00 && cnt8d.dout <= 8'h08 && cmp.sync == 1 && cmp.syncmulti
==1) begin
      teste0 = srSIPO.paralelo;
end

if(cnt8d.dout >= 8'h08 && cnt8d.dout < 8'h10 && cmp.sync == 1 && cmp.syncmulti ==
1) begin
     teste1 =srSIPO.paralelo;
end

if(cnt8d.dout >= 8'h10 && cnt8d.dout < 8'h18 && cmp.sync == 1 && cmp.syncmulti ==
1) begin
     teste2 =srSIPO.paralelo;
end

if (cnt8d.dout >= 8'h18 && cnt8d.dout < 8'h20 && cmp.sync == 1 && cmp.syncmulti
==1) begin
      teste3 =srSIPO.paralelo;
end

if (cnt8d.dout >= 8'h20 && cnt8d.dout < 8'h28 && cmp.sync == 1 && cmp.syncmulti
==1) begin
      teste4 =srSIPO.paralelo;
end

if (cnt8d.dout >= 8'h28 && cnt8d.dout < 8'h30 && cmp.sync == 1 && cmp.syncmulti
==1) begin
      teste5 =srSIPO.paralelo;
end

if (cnt8d.dout >= 8'h30 && cnt8d.dout < 8'h38 && cmp.sync == 1 && cmp.syncmulti
==1) begin
      teste6 =srSIPO.paralelo;
end

if (cnt8d.dout >= 8'h38 && cnt8d.dout < 8'h40 && cmp.sync == 1 && cmp.syncmulti
==1) begin
      teste7 =srSIPO.paralelo;
end
```

```
if (cnt8d.dout >= 8'h40 && cnt8d.dout < 8'h48 && cmp.sync == 1 && cmp.syncmulti
==1) begin
     teste8 =srSIPO.paralelo;
end

if (cnt8d.dout >= 8'h48 && cnt8d.dout < 8'h50 && cmp.sync == 1 && cmp.syncmulti
==1) begin
     teste9 =srSIPO.paralelo;
end

if (cnt8d.dout >= 8'h50 && cnt8d.dout < 8'h58 && cmp.sync == 1 && cmp.syncmulti
==1) begin
     teste10 =srSIPO.paralelo;
end

if (cnt8d.dout >= 8'h58 && cnt8d.dout < 8'h60 && cmp.sync == 1 && cmp.syncmulti
==1)  begin
     teste11 = srSIPO.paralelo;
end

if (cnt8d.dout >= 8'h60 && cnt8d.dout < 8'h68 && cmp.sync == 1 && cmp.syncmulti
==1) begin
     teste12 = srSIPO.paralelo;
end

if (cnt8d.dout >= 8'h68 && cnt8d.dout < 8'h70 && cmp.sync == 1 && cmp.syncmulti
==1) begin
     teste13 = srSIPO.paralelo;
end

if (cnt8d.dout >= 8'h70 && cnt8d.dout < 8'h78 && cmp.sync == 1 && cmp.syncmulti
==1) begin
     teste14 = srSIPO.paralelo;
end

if (cnt8d.dout >= 8'h80 && cnt8d.dout < 8'h88 && cmp.sync == 1 && cmp.syncmulti
==1) begin
     teste15 = srSIPO.paralelo;
end
```

```
if (cnt8d.dout >= 8'h88 && cnt8d.dout < 8'h90 && cmp.sync == 1 && cmp.syncmulti
==1) begin
     teste16 = srSIPO.paralelo;
end

if (cnt8d.dout >= 8'h90 && cnt8d.dout < 8'h98 && cmp.sync == 1 && cmp.syncmulti
==1) begin
     teste17 = srSIPO.paralelo;
end

if (cnt8d.dout >= 8'h98 && cnt8d.dout < 8'hA0 && cmp.sync == 1 && cmp.syncmulti
==1) begin
     teste18 = srSIPO.paralelo;
end

if (cnt8d.dout >= 8'hA0 && cnt8d.dout < 8'hA8 && cmp.sync == 1 && cmp.syncmulti
==1) begin
     teste19 = srSIPO.paralelo;
end

if (cnt8d.dout >= 8'hA8 && cnt8d.dout < 8'hB0 && cmp.sync == 1 && cmp.syncmulti
==1) begin
     teste20 = srSIPO.paralelo;
end

if (cnt8d.dout >= 8'hB0 && cnt8d.dout < 8'hB8 && cmp.sync == 1 && cmp.syncmulti
==1) begin
     teste21 = srSIPO.paralelo;
end

if (cnt8d.dout >= 8'hB8 && cnt8d.dout < 8'hC0 && cmp.sync == 1 && cmp.syncmulti
==1) begin
     teste22 = srSIPO.paralelo;
end

if (cnt8d.dout >= 8'hC0 && cnt8d.dout < 8'hC8 && cmp.sync == 1 && cmp.syncmulti
==1) begin
     teste23 = srSIPO.paralelo;
end
```

```
if (cnt8d.dout >= 8'hC8 && cnt8d.dout < 8'hD0 && cmp.sync == 1 && cmp.syncmulti
==1) begin
    teste24 = srSIPO.paralelo;
end

if (cnt8d.dout >= 8'hD0 && cnt8d.dout < 8'hD8 && cmp.sync == 1 && cmp.syncmulti
==1) begin
    teste25 = srSIPO.paralelo;
end

if (cnt8d.dout >= 8'hD8 && cnt8d.dout < 8'hE0 && cmp.sync == 1 && cmp.syncmulti
==1) begin
    teste26 = srSIPO.paralelo;
end

if (cnt8d.dout >= 8'hE0 && cnt8d.dout < 8'hE8 && cmp.sync == 1 && cmp.syncmulti
==1) begin
    teste27 = srSIPO.paralelo;
end

if (cnt8d.dout >= 8'hE8 && cnt8d.dout < 8'hF0 && cmp.sync == 1 && cmp.syncmulti
==1) begin
    teste28 = srSIPO.paralelo;
end

if (cnt8d.dout >= 8'hF0 && cnt8d.dout < 8'hF8 && cmp.sync == 1 && cmp.syncmulti ==1)
begin
    teste29 = srSIPO.paralelo;
end
```

```
if (cnt8d.dout >= 8'hC8 && cnt8d.dout < 8'hD0 && cmp.sync == 1 && cmp.syncmulti
==1) begin
     teste24 = srSIPO.paralelo;
end

if (cnt8d.dout >= 8'hD0 && cnt8d.dout < 8'hD8 && cmp.sync == 1 && cmp.syncmulti
==1) begin
     teste25 = srSIPO.paralelo;
end

if (cnt8d.dout >= 8'hD8 && cnt8d.dout < 8'hE0 && cmp.sync == 1 && cmp.syncmulti
==1) begin
     teste26 = srSIPO.paralelo;
end

if (cnt8d.dout >= 8'hE0 && cnt8d.dout < 8'hE8 && cmp.sync == 1 && cmp.syncmulti
==1) begin
     teste27 = srSIPO.paralelo;
end

if (cnt8d.dout >= 8'hE8 && cnt8d.dout < 8'hF0 && cmp.sync == 1 && cmp.syncmulti
==1) begin
     teste28 = srSIPO.paralelo;
end

if (cnt8d.dout >= 8'hF0 && cnt8d.dout < 8'hF8 && cmp.sync == 1 && cmp.syncmulti ==1)
begin
     teste29 = srSIPO.paralelo;
end
```

```
if(cmp.sync == 0) begin
    teste0 = 8'h00;
    teste1 = 8'h00;
    teste2 = 8'h00;
    teste3 = 8'h00;
    teste4 = 8'h00;
    teste5 = 8'h00;
    teste6 = 8'h00;
    teste7 = 8'h00;
    teste8 = 8'h00;
    teste9 = 8'h00;
    teste10 = 8'h00;
    teste11 = 8'h00;
    teste12 = 8'h00;
    teste13 = 8'h00;
    teste14 = 8'h00;
    teste15 = 8'h00;
    teste16 = 8'h00;
    teste17 = 8'h00;
    teste18 = 8'h00;
    teste19 = 8'h00;
    teste20 = 8'h00;
    teste21 = 8'h00;
    teste22 = 8'h00;
    teste23 = 8'h00;
    teste24 = 8'h00;
    teste25 = 8'h00;
    teste26 = 8'h00;
    teste27 = 8'h00;
    teste28 = 8'h00;
    teste29 = 8'h00;
end
end

endmodule
```

BIBLIOGRAFIA

http://www.asic-world.com/verilog/vbehave2.html

http://www.asic-world.com/verilog/syntax2.html

http://web.engr.oregonstate.edu/~traylor/ece474/lecture_verilog/beamer/verilog_modules.pdf

http://paginas.fe.up.pt/~mleitao/TSC/Teoricas/TSC_3-RT.pdf