



2º Laboratório

Trabalho 1 – (Breitling)

Objectivo: Implementar um micro-cronómetro.

1.1 Implementar um contador

Objectivo: Implementar um contador. O contador deve ser incrementado em cada ciclo de relógio do clock de 50 MHz da placa DE2. A saída do contador deve ser ligada aos leds disponíveis que deverão piscar a frequências diferentes. Pretende-se preparar um módulo para efectuar a divisão do clock de forma a obter um clock com um período de micro-segundo.

- Lançar o Quartus II e abrir o projecto DE2_top;
- Identificar o sinal clk_50 bem como os sinais correspondentes aos LEDs (LEDR- vermelhos; LEDG-verdes);
- Implementar um contador em que a entrada é o clk_50 e ligar as saídas aos leds.

1.2 Descodificador de 7 segmentos

Objectivo: descodificar um número em binário para que este seja mostrado no display de 7 segmentos em base decimal. O número será introduzido através dos interruptores da placa em binário devendo aparecer o número em decimal no display. Pretende-se preparar um módulo que descodifique a contagem binária do relógio para a visualização do tempo decorrido.

- Lançar o Quartus II e abrir o projecto DE2_top;
- Identificar os sinais dos interruptores (SW) bem como os sinais correspondentes a um display de 7 segmentos;
- Implementar o código que permite a descodificação. As entradas serão os sinais dos interruptores e as saídas, os segmentos do display.

(Sugestão: reproveite o código do laboratório anterior)

1.3 Cronómetro

Objectivo: Implementar um cronómetro com a resolução de micro-segundo. O cronómetro deve ter como sinais de controlo um reset (actuado por um botão de pressão) e um sinal start/stop actuado por um interruptor. O valor deve ser apresentado nos displays de 7 segmentos como sm mm μμ (s-segundos, m- milésimas de segundo, μ-micro segundos).

O relógio de 1MHz é construindo a partir do relógio de 50MHz utilizando um contador (“sounds familiar”). Os números são apresentados nos displays de 7 segmentos utilizando um descodificador (“sounds familiar”). Falta implementar um contador para o tempo já decorrido e o controlo do cronómetro.

Reconfigure o cronómetro de forma a que o sinal start/stop seja dado com um input externo (GPIO). Este input externo será uma onda quadrada vinda de um gerador com um certo tempo a “high” para permitir a verificação do funcionamento do cronómetro com a precisão requerida.

Material de Apoio

DISCLAIMER:

OS CONTEÚDOS SEGUINtes NÃO COMPROMETEM O AUTOR POR QUALQUER ERRO QUE POSSAM CONTER. DEVEM SER ENTENDIDOS COMO UM FONTE DE INSPIRAÇÃO...

Contadores assíncronos

Vamos implementar dois contadores assíncronos (o clock não está ligado a todos os estados de saída. O clock do bit seguinte é o bit anterior) de 4bits cada ligados um ao outro de forma a obter um contador de 8 bits. Cada contador terá 4 bits, logo utilizaremos 4 flip-flops T. O flip-flop T é uma primitiva do verilogHDL que tem a inicialização:

```
tff nome_instancia(IN, CLK, CLEAR, PRESET, OUT)
```

O código é:

```
reg [7: 0] Cont;
always@(posedge CLOCK_50)
begin
    Cont <= Cont+1;
end

counter4ass counter40(LEDR[11:8], Cont[4]); //usar modulo counterass
counter4ass counter41(LEDR[15:12], LEDR[11]);

modulo counter4ass(q, clk); //modulo que implementa um contador de 4 bits
assincrono

output [3: 0] q;

input clk;
//nega-se o clock porque se quer que comute nos flancos descendentes, de
forma a dar contagem ascendente
tff tff0(.t(1), .clk(!clk), .clrn(1), .prn(1), .q(q[0]));
tff tff1(.t(1), .clk(!q[0]), .clrn(1), .prn(1), .q(q[1]));
tff tff2(.t(1), .clk(!q[1]), .clrn(1), .prn(1), .q(q[2]));
tff tff3(.t(1), .clk(!q[2]), .clrn(1), .prn(1), .q(q[3]));

endmodule
```

Este código vai implementar um contador e mostrar a contagem nos leds.

Como se pode ver cada clock do flip flop seguinte vai buscar a variação de saída do flip flop anterior.

Contadores síncronos

Vamos implementar dois contadores síncronos (o clock é comum a todos os flip flops do contador) de 4 bits cada ligados um ao outro de forma a obter um contador de 8 bits. Este contador terá 4 bits, logo utilizaremos 4 flip-flops T. O flip-flop T é uma primitiva do verilogHDL que tem a inicialização:

```
tff nome_instancia(IN, CLK, CLEAR, PRESET, OUT)
```

Os contadores têm Enable e rco para permitir ligá-los entre si

O código é:

```
reg [7: 0] Cont;
always@(posedge CLOCK_50)
```

```

begin      Cont    <=    Cont+1;
end

wire rco; //rco para me permitir activar o segundo contador quando o
primeiro chegar a 15
counter4sinc counter42(LEDG[3:0], rco, Cont[4], 1);
counter4sinc counter43(LEDG[7:4], , Cont[4], rco);

module counter4sinc(q, rco, clk, enb); //modulo que implementa um contador
de 4 bits sincrono com enable e rco

output [3:0] q, rco=(q[0] && q[1] && q[2] && q[3]) ;
input clk, enb;
wire a[4:0];

tff tff0(. t(enb), . clk(clk), . clrn(1), . prn(1), . q(q[0]));
and and0(a[0], q[0], enb);
tff tff1(. t(a[0]), . clk(clk), . clrn(1), . prn(1), . q(q[1]));
and and1(a[1], q[1], enb);
and and2(a[2], a[1], a[0]);
tff tff2(. t(a[2]), . clk(clk), . clrn(1), . prn(1), . q(q[2]));
and and3(a[3], q[2], enb);
and and4(a[4], a[3], a[2]);
tff tff3(. t(a[1]), . clk(clk), . clrn(1), . prn(1), . q(q[3]));

endmodule

```

Este código vai implementar um contador e mostrar a contagem nos leds.

Como se pode ver cada clock do flip flop seguinte vai buscar a variação de saída do flip flop anterior.

Contador em Verilog – A maneira simples

```

reg [7:0] Cont;
always@(posedge CLOCK_50)
begin
    Cont <= Cont+1;
end

assign LEDR[7:4] = Cont[7:4];

```

Descodificadores usando lógica combinatória (mapas de Karnaugh)

Começamos por projectar um circuito que, usando os switchs para representar um número em binário de 2 bits, apresente esse número no painel de 7 segmentos. Vamos implementar este circuito usando tabelas de Karnaugh.

Depois de fazermos e simplificarmos as tabelas de Karnaugh obtemos o seguinte:

```
A= !M1. M2
B= 0
C= !M2 . M1
D= !M1 . M2
E= M2
F= M2 + M1
G= !M1
```

Logo o código é:

```
//usar os sw 5 e 6 para introduzir um numero em binario e aparece no
//display de sete segmentos (fazer logica combinatoria)
assign HEXO[0] = !SW[6] && SW[5];
assign HEXO[1] = 1'b0;
assign HEXO[2] = !SW[5] && SW[6];
assign HEXO[3] = !SW[6] && SW[5];
assign HEXO[4] = SW[5];
assign HEXO[5] = SW[5] || SW[6];
assign HEXO[6] = !SW[6];
```

Descodificadores usando “case statement”

Agora vamos apresentar, no ecran de sete segmentos:

- 1 se os switchs estiverem a 0 0
- 3 se os switchs estiverem a 0 1
- 5 se os switchs estiverem a 1 0
- 7 se os switchs estiverem a 1 1

O código é:

```
reg [1:0] sel;
sel[1]=SW[5];
sel[0]=SW[6];
case (sel)
  2'b00 : begin
    assign HEXO[0] = 1'b1;
    assign HEXO[1] = 1'b0;
    assign HEXO[2] = 1'b0;
    assign HEXO[3] = 1'b1;
    assign HEXO[4] = 1'b1;
    assign HEXO[5] = 1'b1;
    assign HEXO[6] = 1'b1;
  end
  2'b01 : begin
    assign HEXO[0] = 1'b0;
```

```

assign      HEX0[ 1]      =      1' b0;
assign      HEX0[ 2]      =      1' b0;
assign      HEX0[ 3]      =      1' b0;
assign      HEX0[ 4]      =      1' b1;
assign      HEX0[ 5]      =      1' b1;
assign      HEX0[ 6]      =      1' b0;
end
2' b10 : begin
      assign      HEX0[ 0]      =      1' b0
assign      HEX0[ 1]      =      1' b1;
assign      HEX0[ 2]      =      1' b0;
assign      HEX0[ 3]      =      1' b0;
assign      HEX0[ 4]      =      1' b1;
assign      HEX0[ 5]      =      1' b0;
assign      HEX0[ 6]      =      1' b0;
end
2' b11 : begin
      assign      HEX0[ 0]      =      1' b0
assign      HEX0[ 1]      =      1' b0;
assign      HEX0[ 2]      =      1' b0;
assign      HEX0[ 3]      =      1' b1;
assign      HEX0[ 4]      =      1' b1;
assign      HEX0[ 5]      =      1' b1;
assign      HEX0[ 6]      =      1' b1;
end
endcase

```