UNIVERSIDADE DE SÃO PAULO

Escola de Engenharia de São Carlos

SEL0621 - Projetos de Circuitos Integrados Digitais I

Prof. Dr. João Pereira do Carmo

Projeto 10

Davi Diório Mendes	7546989
Nivaldo Henrique Bondança	7143909

29 de outubro de 2014

Lista de Figuras

Lista de Tabelas

Códigos Fontes

Introdução

Neste relatório são exploradas a ferramenta *LeonardoSpectrum*, o roteamento de circuitos com um alto número de componentes e a criação, manipulação e análise de circuitos gerados a partir de um código `VHDL`.

Resumo

O objetivo deste projeto é gerar a partir de uma descrição de alto nível o *layout* de um circuito.

Questões

1. Vamos utilizar agora o programa *LeonardoSpectrum (Windows)*. Para poder empregar as bibliotecas da $AMS_0.35\mu m$ devemos ter alguns arquivos (`c35_CORELIB.syn`; `c35_CORELIB_3B.syn`; etc.) no diretório `C:\MGC\LeoSpec\LS2005a_82\lib`. Verifique se eles já estão lá; caso não estejam, copie os arquivos do diretório `Mentor/Lib` (pen drive) para aquele diretório. Ao abrir o Leonardo configure o *working diretory* para sua área de trabalho. Neste diretório serão colocados os resultados do que fizer.

2. Vamos sintetizar inicialmente o circuito descrito no arquivo uart.vhd do diretório Demo (`C:\MGC\LeoSpec\LS2005a_82\Demo`) que descreve um universal asynchronous receiver/-transmitter. Abra este arquivo no Leonardo (`INPUT - Open files`) e então execute o comando `INPUT - Read`. Observe que ao ser feita a leitura já é realizada uma primeira síntese.

3. Abra o arquivo texto dentro do *Leonardo* e dê uma olhada no conteúdo do `uart.vhd` (click duas vezes em cima do nome do arquivo). Faça alguma modificação no arquivo de forma a causar erro (por exemplo, troque `"ENTITY uart IS"` por `"ENTITY art IS"`). Salve, feche o texto e execute novamente o `INPUT - Read`. Veja as mensagens de erro. Caso abra novamente o arquivo texto aparecerão indicações dos erro.

4. Selecione a tecnologia para `AMS-C35_CORELIB` (*Technology*) e otimize o circuito (`Optimize-Optimize`).

5. Veja os esquemáticos que foram gerados (observe o esquemático com a opção *multipages* ou não). Qual é a diferença entre o esquemático associado a `EXEMPLAR_XTR` e o esquemático associado a `EXEMPLAR`.

O esquemático `EXEMPLAR` é o esquemático otimizado utilizando apenas componentes da tecnologia especificada, enquanto o esquemático `EXEMPLAR_XTR` é o esquemático originalmente gerado pelo programa, sem, necessariamente utilizar os componentes escolhidos.

6. Verifique o *Critical Path* do esquemático sintetizado e mapeado na tecnologia da AMS. O que significa este *critical path* e como é calculado?

O *Critical Path* mostra o caminho do crítico do circuito, isto é, o caminho que utiliza o maior tempo total de propagação, o tempo que limitará a freqüência de operação do circuito. O caminha crítico, *Critical Path*, é calculado somando os atrasos de propagação gerados por cada componente em um caminho entre dois *Flip-Flops*.

7. Refaça a otimização alterando as opções de objetivo, área, velocidade, etc. e verifique os resultados. Variando as opções, minimize o *Critical Path*. Qual o valor final obtido e qual a freqüência máxima de operação que o circuito sintetizado pode atingir? Apresente no relatório a figura do caminho crítico encontrado.

Após algumas otimizações, foi encontrado um caminho crítico, como mostra a **Figura 1** com tempo de atraso de propagação de $2,48ns$, o que implica uma freqüência máxima de operação de $403,2MHz$.

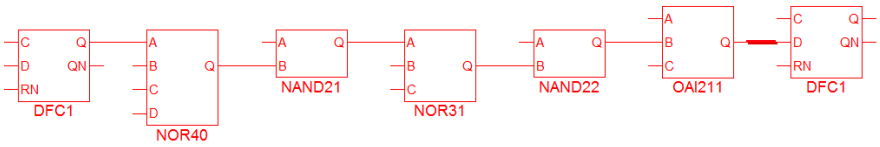

Figura 1: Caminho crítico, *Critical Path*, do sistema de acordo com o *LeonardoSpectrum*

8. Vamos gerar agora um arquivo de saída no formato *Verilog*. O *Verilog* é uma linguagem de descrição de hardware usada com C.I.s (mais usada que o VHDL na indústria). Este formato servirá de interface para passarmos os resultados para o *Design Architecture*. No *Leonardo*, depois de fazer a síntese, vá ao menu *Output*, e configure para gerar *Verilog*. De o nome que deseja e lembre-se que o resultado será colocado no diretório de trabalho que escolheu.

9. Verifique o arquivo de saída com um editor de texto e tente compreender a descrição que está feita (ao menos ter uma idéia).

10. Gere arquivos *Verilog* para os circuitos `uart.vhd` e `priority_encoder.vhd`. Transfira os arquivos *Verilog* para o Linux (no sistema Linux, os arquivos do Windows podem ser

vistos no diretório /windows). Agora devemos convertê-los para gerar um *layout*. Abra o *IcStudio* (em algum projeto que já usou ou em um novo). Nele dê o comando `import verilog`. Configure:

```
Output library: onde quer colocar
Verilog netlist: o arquivo gerado
Name map file: local/tools/dkit/ams_3.70_mgc/mentor/c35/verilogin_cellmapfiles/
    c35b4_digital.cellmap
```

Execute. Deve ser criado tanto o esquemático como o símbolo do circuito `uart` e do circuito `priority.encoder`. Verifique ambos. No esquemático da `uart` há mais de uma página e para o *Check Schematic* passar corretamente as duas páginas do esquemático devem estar abertas.

11. A partir do esquemático (utilize o *viewpoint*) faça a geração do *layout* da `uart`. No roteamento utilize para *VDD* e *VSS* apenas linhas mais largas do que $1,8\mu m$. Para conseguir isso utilize, dentro do menu `ARoute`, a edição dos `net Classes`.

Procure nesse circuito fazer o roteamento mas observe que devido ao tamanho, completa-lo é bastante trabalhoso. Após o roteamento passe o *LVS* e o *DRC*.

Obs.1: Para se posicionar todas as células da `uart` é necessário se ter as duas páginas do esquemático abertas no ICStation. Para isso utilize o comando `$open_sheet()`. Posteriormente selecione todas as células de cada esquemático e faça o seu *placement*.

Obs.2: Para terminar *layout* a sugestão é seguir os passos:

- Faça inicialmente o roteamento das linhas de alimentação (*VDD* e *VSS*);
- Execute, sem colocar os ports, o roteamento automático do restante dos sinais. Deixe os metais configurados para serem utilizados em apenas uma direção;
- Quando o número de ligações não feitas estiver em torno de 30, altere a configuração para permitir que os metais sejam utilizados nas duas direções;
- Agora selecione uma linha de cada vez e mande executar o roteamento automático. Caso a ferramenta não consiga executar um roteamento, apague as ligações que estão atrapalhando, sempre há, ou tente o RIP. Não use indiscriminadamente o comando RIP, pois algumas vezes ele piora o roteamento;
- Quando conseguiu realizar todas as ligações execute o *LVS* (coloque nele a opção "ignorar os ports");
- Após o *LVS* dar resultado correto acrescente os *ports* e termine o roteamento;

- Passe o *DRC* e corrija os erros (muitos);
- Termine o *layout* passando o *LVS* (agora considerando os *ports*).

Obs.3: o item acima é bastante trabalhoso, mas será um excelente treino para roteamento e *DRC*.

O *layout* final está representada na **Figura 2** .

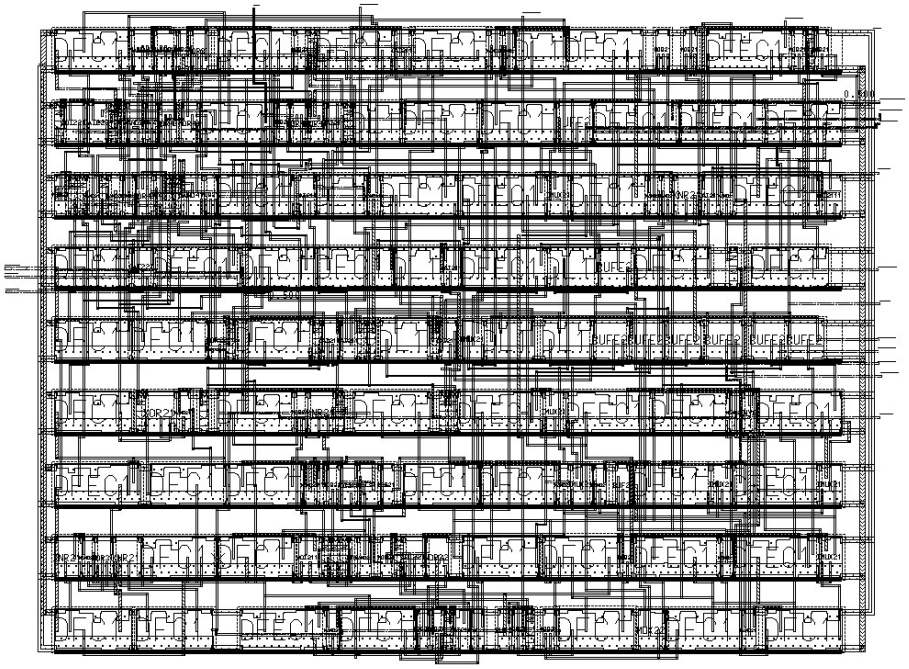

Figura 2: *Layout* do circuito *uart*

12. Considere o circuito da **Figura 3** (circuito prescaler).
O circuito composto pelos blocos hachurados (três D-*flip-flops* e duas portas lógicas) compões uma máquina de estados. Considere que:

- os sinais A, B e C definem o estado da máquina (ex.: o estado 000 é quando A="0", B="0"e C="0");
- esta máquina tem como entrada o sinal MC que define se o circuito divide o *clock* por 4, MC = "1", ou por 5, MC = "0";
- a saída é o sinal A.

A máquina de estados implementada está representada na **Figura 4** .

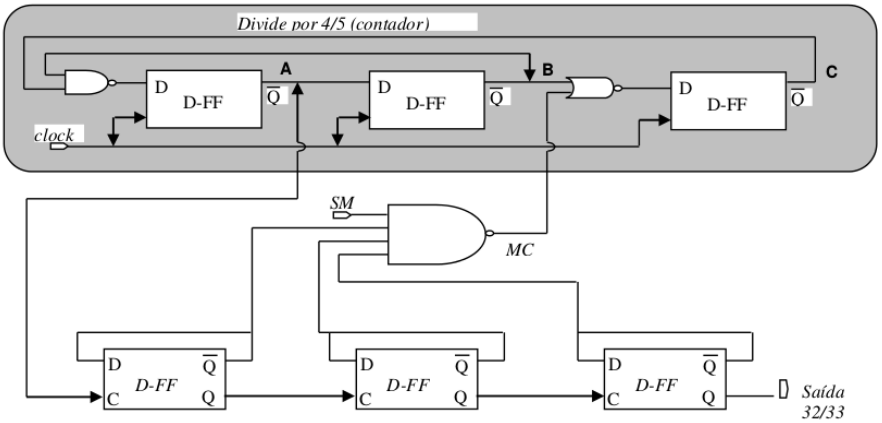

Figura 3: *Prescaler* 32/33

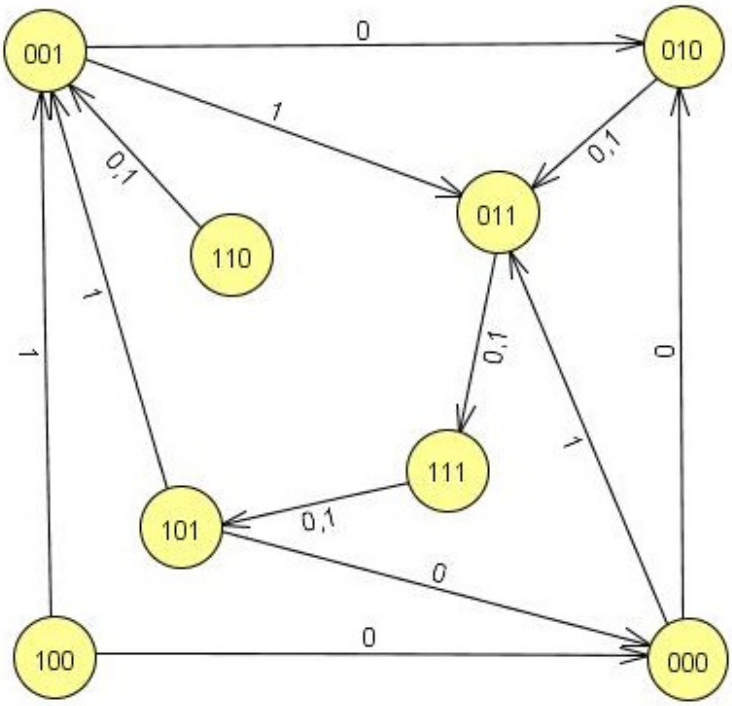

Figura 4: Máquina de estados de um contador 4/5

13. Verifique que a máquina de estados representada pelo diagrama que traçou funciona dividindo o sinal de *clock* por 4 ou 5, de acordo com o valor de *MC*.

14. Descreva a máquina de estados em *VHDL*. Utilize o manual de *VHDL* do software Leonardo (*LeonardoSpectrum HDL Synthesis Manual:* `D:\MGC\LeoSpec\LS2005a_82\doc\leospec_hdl.pdf`) para ver modelos de descrição para máquinas de estado. Apresente o *VHDL* no relatório.

A descrição da máquina de estados está descrita no **Código fonte 1**

Código Fonte 1: Código fonte da máquina de estados

```
entity divisor_4_5 is

        port (  mc, clk : in  bit;
                A       : out bit );

end divisor_4_5;

architecture div_4_5_imp of divisor_4_5 is
        type   state_type is (s000, s001, s010, s011, s100, s101, s110, s111);
        signal act_state, next_state : state_type;
begin

        -- Atualização de estados.
        registers : process (clk)
        begin
                if(clk'event and clk='1') then
                        act_state <= next_state;
                end if;
        end process;

        --Transição de estados.
        transitions : process (mc)
        begin
                case act_state is
                        when s000 =>
                                A <= '0';
                                if (mc='0') then
                                        next_state <= s010;
                                else
                                        next_state <= s011;
                                end if;
                        when s001 =>
                                A <= '0';
                                if (mc='0') then
                                        next_state <= s010;
                                else
                                    next_state <= s011;
                                end if ;
                        when s010 =>
                                A <= '0';
                                next_state <= s011;
                        when s011 =>
                                A <= '0';
                                next_state <= s111;
                        when s100 =>
                                A <= '1';
                                if (mc='0') then
                                        next_state <= s000;
                                else
                                    next_state <= s001;
                                end if ;
                        when s101 =>
                                A <= '1';
```

```
54                                  if (mc='0') then
55                                          next_state <= s000;
56                                  else
57                                      next_state <= s001;
58                                  end if;
59                              when s110 =>
60                                  A <= '1';
61                                  next_state <= s001;
62                              when s111 =>
63                                  A <= '1';
64                                  next_state <= s101;
65                  end case;
66          end process;
67  end div_4_5_imp;
```

15. Abra e leia o circuito *VHDL*, corrija os possíveis erros e otimize. Verifique o circuito gerado e compare com o esquemático da Figura 1.

16. Feche o Leonardo e abra novamente. Leia o *VHDL*, mas tome cuidado para utilizar como `Encoding Style` a opção `Binary`. Otimize e veja se o resultado melhorou. Otimize o circuito para obter o menor *Critical Path*.
Qual é o valor encontrado? Apresente o esquemático obtido.

Após as otimizações, foi encontrado, para o caminho crítico, um valor de atraso de $0,96ns$. O esquemático da máquina de estados está representado na **Figura 5**

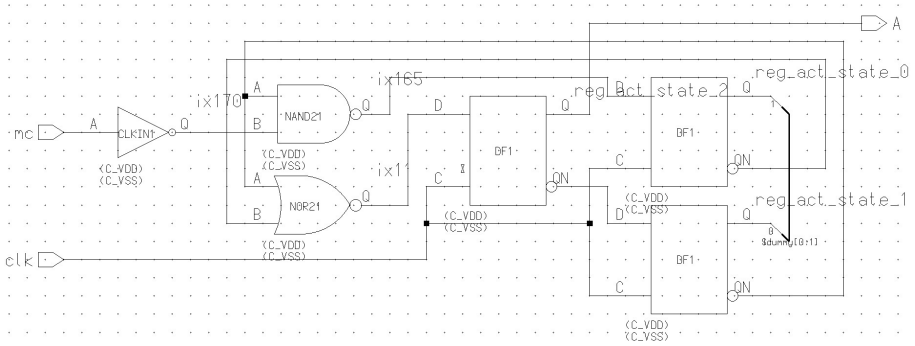

Figura 5: Esquemático da máquina de estados

17. Sintetize o circuito utilizando a biblioteca da *AMS*.

18. Faça as verificações com *DRC* e *LVS* e apresente o *layout* final.

O *layout* final da máquina de estados está representado na **Figura 6** .

19. Faça a extração do circuito com `R+C+CC` e determine a máxima velocidade que o circuito atinge (teste para `MC = "1"` e para `MC= "0"`). Compare o resultado com o encontrado no item

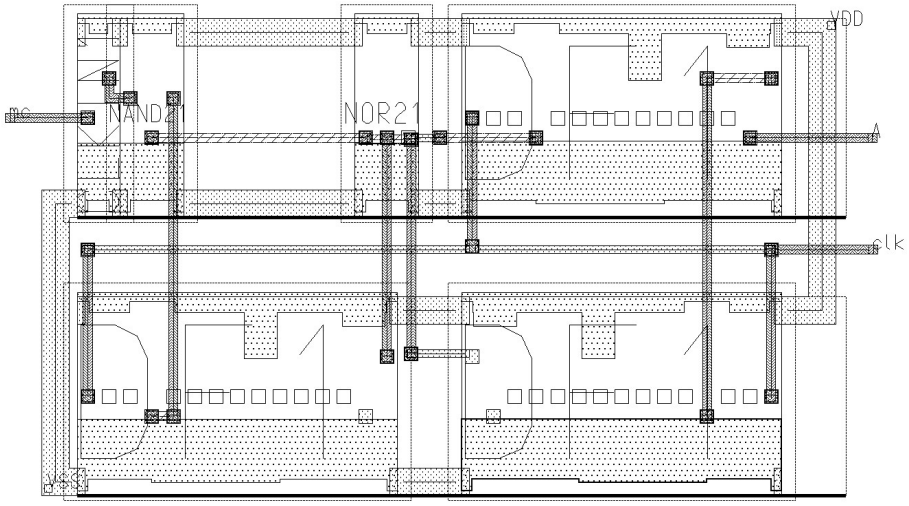

Figura 6: *Layout* da máquina de estados

17.

Os resultados para maior frequência de operação estão representados na **Tabela 1** . As frequências de operação máxima da simulação foram obtidas através da **Figura 7** e **Figura 8** .

Após uma análise dos dados, percebe-se que há uma grande diferença entre o valor calculado no *LeonardoSpectrum* e os calculados na extração `R+C+CC`. Isso ocorre, pois, na simulação, são considerados os elementos parasitas do circuito, que são ignorados no cálculo do *LeonardoSpectrum*.

Tabela 1: Máximas frequência de operação.

Modelo	**Frequência (GHz)**
LeonardoSpectrum – Critical Path	1,04
Simulação (MC = 1)	0,92
Simulação (MC = 0)	0,91

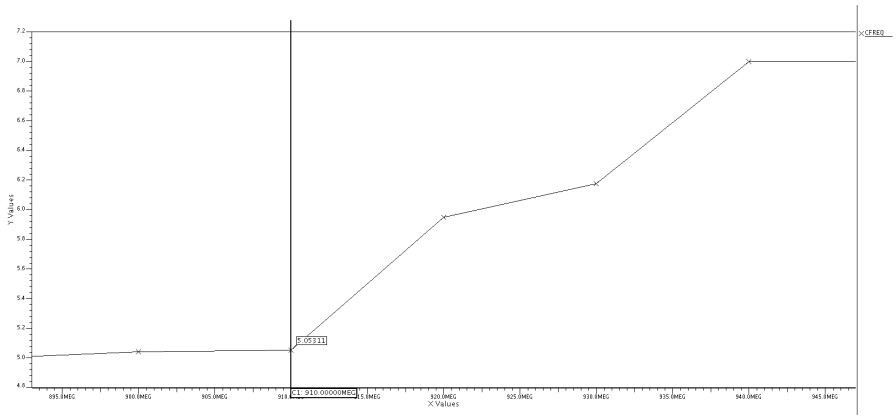

Figura 7: Máxima frequência de operação para MC = 0

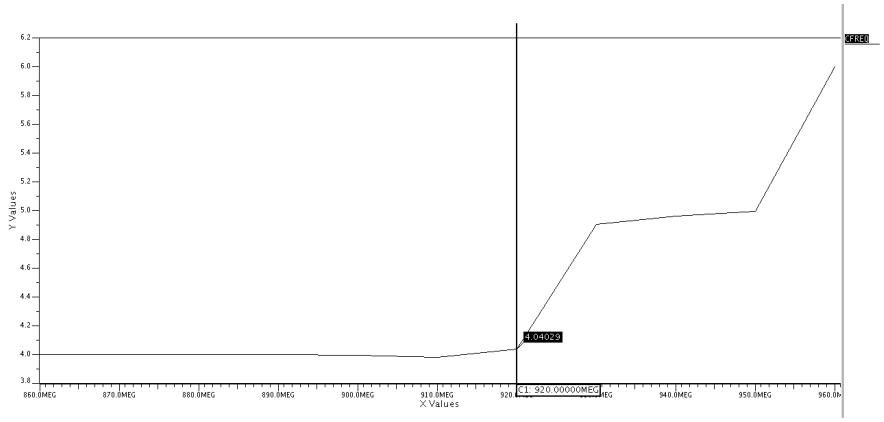

Figura 8: Máxima frequência de operação para MC = 1