13.3 Somadores

Objetivos: são três os objetivos deste laboratório: (i) compreender e apreender as bases do processo de depuração de modelos dos três somadores; (ii) medir e comparar seus tempos de propagação; e (iii) construir um modelo para um somador do tipo "seleção do vai-um" (*carry-select adder*), verificar sua corretude, e medir seu tempo de propagação.

O trabalho pode ser efetuado em duplas.

Preparação: leia as Seções 6.6, 6.7 e 6.9 de `aritm.pdf`.

13.3.1 Material Disponibilizado Para Sua Tarefa

Etapa 18 Copie para sua área de trabalho o arquivo com o código VHDL:
(a) `wget http://www.inf.ufpr.br/roberto/ci210/vhdl/l_somadores.tgz`
(b) expanda-o com: `tar xzf l_somadores.tgz`
(c) mude para o diretório recém criado: `cd somadores`

O *script* `run.sh` compila o código VHDL e produz um simulador. Se executado sem nenhum argumento de linha de comando, `run.sh` somente (re)compila o simulador e o executa; com um argumento o *script* também dispara a execução de `gtkwave`: `./run.sh 1 &` .

13.3.2 Somadores com Adiantamento de Vai-um

O arquivo `packageWires.vhd` contém definições de abreviaturas para nomes de sinais, e constantes para os tempos de propagação das portas lógicas. Neste laboratório todos os valores são do tipo bit, com valores em $\mathbb{B}$.

O arquivo `aux.vhd` contém os modelos das portas lógicas *inv, and, or, xor*, que são os componentes básicos para este laboratório. Este arquivo não deve ser editado.

O arquivo `somador.vhd` contém três modelos de somadores, com complexidade e desempenho crescentes, a saber *adderCadeia*, *adderAdiant4* e *adderAdianta16*, descritos em `aritm.pdf`.

O modelo *adderCadeia* é um modelo estrutural do somador composto por 16 instâncias do somador completo (*full-adder*), sem circuito de adiantamento de vai-um.

O modelo *adderAdianta4* estende o modelo *adderCadeia* com cadeias de adiantamento de vai-um, para grupos de 4 bits.

O modelo *adderAdianta16* estende o modelo *adderAdianta4* com uma cadeia de adiantamento de vai-um de 16 bits.

O arquivo `tb_somador.vhd` contém o programa de testes (*testbench*, ou TB) para verificar a corretude dos modelos. A arquitetura do TB declara os componentes que serão testados e um **record** que é usado para excitar os modelos. O registro test_record, mostrado no Programa 13.53, possui cinco campos e os valores destes campos devem ser atribuídos de forma a gerar todas (*todas?*) as combinações de entradas para garantir a corretude do modelo. O vetor de testes test_array contém alguns elementos para ilustrar as possibilidades.

Programa 13.53: Vetor de valores de entrada para testar os modelos.

```
type test_record is
   record
     a : reg16;       -- entrada
     b : reg16;       -- entrada
     f : bit;         -- operação: 0=ADD, 1=SUB
     c : reg16;       -- saída esperada
     v : bit;         -- vai-um/empresta-um esperado
   end record;

 type test_array is array(positive range <>) of test_record;

 constant test_vectors : test_array := (
   -- a,     b,      f, c,        vai-um
   -- testes para soma
   (x"0000",x"0000",'0',x"0000",'0'),
   -- acrescente novos vetores aqui, testes para soma f=0
   (x"0001",x"0001",'0',x"0002",'0'),
   (x"0010",x"0010",'0',x"0020",'0'),
   ...
   -- acrescente novos vetores aqui, testes para subtração f=1
   (x"0000",x"0000",'1',x"0000",'1')
   ...
   );

 -- troque a constante para FALSE para testar seus modelos
 constant TST_CADEIA   : boolean := false;
 constant TST_ADIANT4  : boolean := true;
 constant TST_ADIANT16 : boolean := true;
```

No test_record, os campos a,b,c são vetores de bits codificados em hexadecimal (x"0FA4") e contém as parcelas da soma e o valor esperado para o resultado. O campo f determina a função do circuito: se f='0', o circuito efetua somas; se f='1', o circuito efetua subtrações. O campo v contém o vai-um esperado. Nas subtrações o valor de v deve ser complementado porque este bit se comporta como empresta-um, ao invés de vai-um.

Para testar cada modelo de somador, altere a respectiva constante para false e então verifique os resultados. As constantes são TST_CADEIA, TST_ADIANT4 ou TST_ADIANT16.

A sequência de testes é implementada no processo U_testValues, com um laço **for** ... **loop**. A variável de iteração itera no espaço definido pelo número de elementos do vetor de testes (test_vectors '**range**) – o atributo '**range** representa a faixa de valores do índice do vetor. Se mais elementos forem acrescentados ao vetor, o laço executará mais iterações. O elemento do vetor é atribuído à variável v e os vários campos do vetor são então atribuídos aos sinais que excitam os modelos. O processo U_testValues executa concorrentemente com os modelos dos somadores e, quando os sinais de teste são atribuídos no laço, estes provocam alterações nos sinais das entradas dos modelos.

O comando **assert** verifica se a saída observada do somador é igual à saída esperada. Se os valores são iguais, o comportamento é o esperado, e portanto *correto, do ponto de vista dos vetores de teste que você escreveu*. Note que se você escolher valores de teste inadequados, ou errados, o diagnóstico de eventuais problemas no modelo pode ser difícil.

Ao final do laço, a simulação termina no comando **wait**, que faz com que a execução do simulador se encerre.

O arquivo `s.vcd` contém definições para o `gtkwave` tais como a escala de tempo e sinais a serem exibidos na tela para a verificação dos modelos dos 3 somadores: os sinais no topo são do somador sem adiantamento de vai-um, ao centro são os sinais do somador com adiantamento de 4 em 4 bits, e embaixo os sinais do somador com adiantamento de 4 em 4, e de 16 em 16 bits.

Etapa 19 Os modelos no arquivo `somador.vhd` contém erros que você deve encontrar e corrigir. Os erros estão nos modelos dos somadores. Você deve ampliar o vetor de testes para explicitar os erros.

Acrescente ao arquivo `tb_somadores.vhd` os vetores de teste para verificar a corretude dos três somadores. Acrescente tantas tuplas quantas forem necessárias à test_vectors para que se possa ter um mínimo de confiança na corretude do projeto.

Cada um dos modelos contêm um erro e sua tarefa é gerar vetores de teste que evidenciem os erros para então corrigir os modelos. Os erros estão (somente) nos modelos dos somadores.

Que valores usar nos testes?

Revise a Seção 6.9. Os valores escolhidos para os testes ajudam a evidenciar os locais onde, provavelmente, há um erro no modelo. É imprescindível ler com atenção as saídas de texto do simulador para descobrir a eventual inconsistência em cada um dos testes. A posição do bit em erro indica a posição no modelo que pode estar errada.

13.3.3 Temporização

Os modelos dos somadores contém informação de temporização.

No topo do arquivo `packageWires.vhd` estão as declarações das constantes com o tempo de propagação das portas lógicas definidas em `aux.vhd`. Edite `packageWires.vhd` e altere a definição da constante simulate_time, no topo do arquivo, de 0 para 1. Note que as constantes que definem os tempos de propagação estão multiplicadas por um número que pode ser zero ou um. Recompile os modelos e verifique seu funcionamento com `gtkwave`.

Espaço em branco proposital.

A Figura 13.17 mostra a tela do `gtkwave` com uma simulação dos três somadores. As barras de cor ciano indicam o tempo de propagação dos sinais, desde a mudança nas entradas, até que a saída estabiliza. Por "estabiliza" entenda-se que *ambos, a soma e o vai-um* estabilizaram.

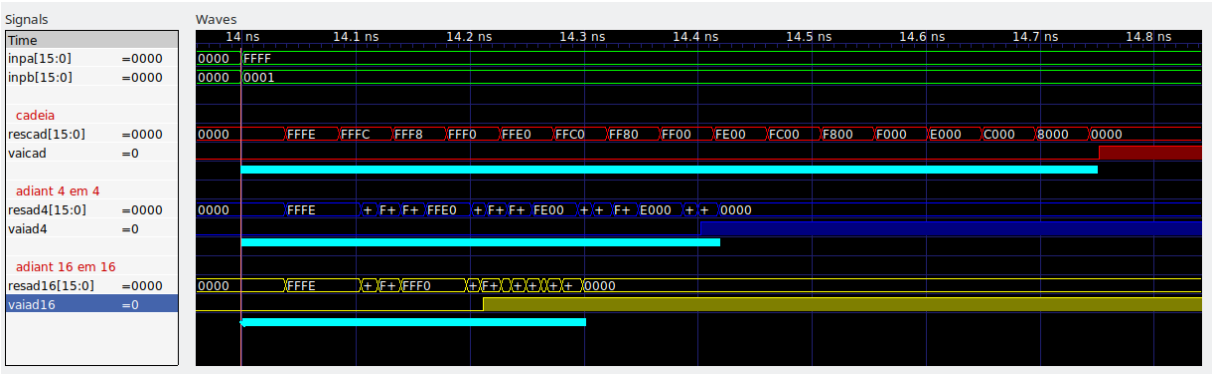

Figura 13.17: Tempo de propagação dos três somadores.

A saída do somador sem adiantamento (cadeia) passa por 15 estágios, e em cada estágio, um bit dos menos significativos assume o seu valor definitivo. No somador com adiantamento de 4 bits, também são 15 estágios, e em quatro deles se percebe que um quarteto inteiro assumiu seu valor definitivo. A saída do somador com adiantamento de 16 bits, por causa dos vários caminhos de adiantamento, embaralha a evolução do resultado ao longo do tempo. Ajustanto-se a escala de tempo horizontal (*zoom*) é possível que todos os estágios fiquem aparentes.

Lembre que o *tempo de propagação* é aquele do caminho mais longo entre entradas e saídas. No caso de somadores, o caminho mais longo inicia no *vem-um* e termina, ou no *vai-um*, ou no bit mais significativo.

Achtung: o tempo de propagação pode mudar em função dos valores de teste; escolha valores das entradas que atravessam o caminho crítico provocando alteração em todos os bits da saída. O par `(x0001,xffff)` não é o que produz o pior tempo de propagação.

Etapa 20 Meça e compare os tempos de propagação dos três somadores.

Não esqueça de entregar os resultados das medições ao professor. Use os cursores do `gtkwave` para medir os tempos de propagação.

(1) Quais são os valores para as entradas que explicitam os piores casos do tempo de propagação? Justifique sua resposta.

(2) Quais os ganhos de desempenho dos modelos com adiantamento, com relação ao modelo sem adiantamento? Mostre como efetuou a comparação.

Espaço em branco proposital.

13.3.4 Somador de 32 bits – *carry select adder*

Reveja a Seção 6.7 de `aritm.pdf`.

Nos diagramas, os números inteiros são denotados por cadeias de bits $a_{n-1}a_{n-2}\cdots a_0$ que representam o número $a_{n-1}\cdot 2^{n-1} + a_{n-2}\cdot 2^{n-2} + \cdots + a_0\cdot 2^0$.

Uma alternativa de projeto para a implementação de somadores é mostrada na Figura 13.18, que contém um somador de 8 bits construído com três somadores de 4 bits. A parte menos-significativa do resultado é obtida pela soma dos dois operandos:

$$\langle V_3 S_{3..0}\rangle = A_{3..0} + B_{3..0} + vem_0\,.$$

A parte mais-significativa é obtida em paralelo com a parte menos-significativa, computando-se simultaneamente as duas alternativas, uma com $vem_4 = 0$, e a outra com $vem_4 = 1$.

Quando os sinais se propagam através do somador da metade menos significativa e o valor de $V_3 = vai_3$ fica estável, este é usado para escolher uma das duas somas para a metade mais significativa, com o multiplexador de 4 bits de largura. Este circuito é chamado de "somador com seleção de vai-um" (*carry select adder*).

Verifique no modelo se a função lógica que determina o valor de vai_8 está correta.

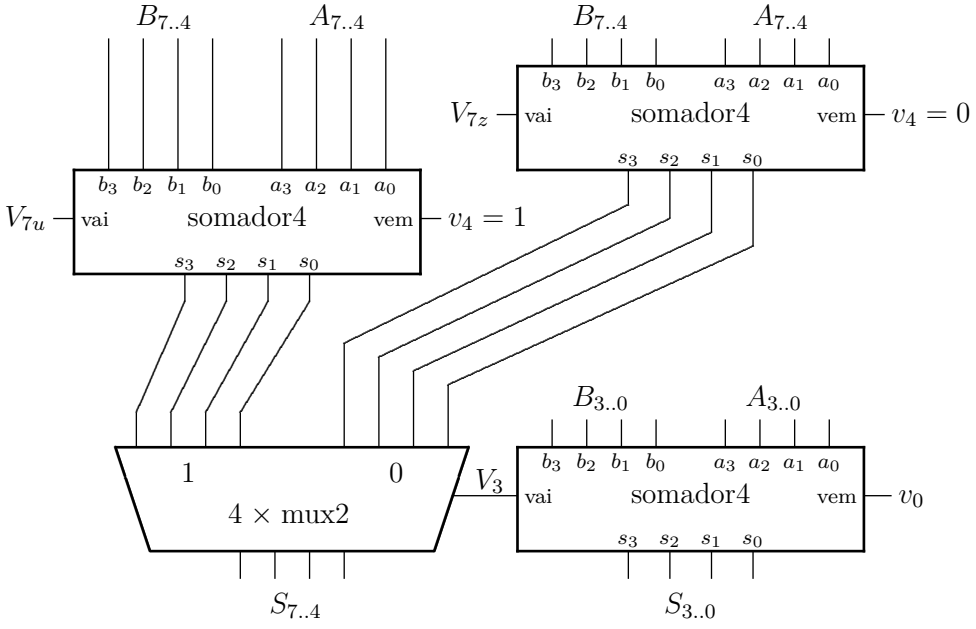

Figura 13.18: Somador com seleção de vai-um.

Etapa 21 Sua tarefa é escrever um modelo de somador com seleção de vai-um com 32 bits de largura. Acrescente seu código VHDL ao final do arquivo `somadores.vhd`. O *testbench* a ser usado nesta etapa é `tb_csa.vhd`, que já contém alguns dos vetores de teste para seu modelo. Para executar os testes, invoque o script `./runCSA.sh`, que compila e executa a simulação.

O vetor de testes test_array foi ajustado para os operandos de 32 bits e é mostrado no Programa 13.54. Esse modelo efetua somente somas.

Programa 13.54: Vetor de testes para o somador de 32 bits.

```
type test_record is
   record
      a : reg32;       -- entrada
      b : reg32;       -- entrada
      c : reg32;       -- saída
      v : bit;         -- vai-um
   end record;

 type test_array is array(positive range <>) of test_record;

 constant test_vectors : test_array := (
   -- a,        b,           c,          vai-um
   -- testes para soma
   (x"00000000",x"00000000",x"00000000",'0'),
   -- acrescente novos valores aqui
   (x"00000001",x"00000001",x"00000002",'0'),
   (x"00000fff",x"00000001",x"00001000",'0'),
   ...
   );
```

Etapa 22 O tempo de propagação do somador com seleção de vai-um é mais curto do que um circuito com dois somadores de 16 bits ligados em série? Nesta etapa, para efetuar a comparação, suponha que dois somadores de 16 bits rápidos são ligados em série e o vai-um do primeiro (V_{15}) é ligado diretamente à entrada vem-um do segundo.

Entregar até as 07:30 do dia 24fev (quarta-feira).

Histórico das Revisões:
22fev2021: adição do diagrama de tempos;
10fev2021: revisão no texto;
02set2019: revisão no texto; remoção do multiplicador;
23ago2018: pequenos ajustes c.r.aos testes;
30ago2017: revisão no texto;
31ago2016: adição de `tb_csa.vhd` e `runCSA.sh`;
03set2014: melhora no texto sobre testes; mult para dever de casa;
05mar2014: adição de texto sobre vetores de teste;
19ago2013: revisão do texto, adição do carry-select adder;
20mar2013: segunda versão;
01nov2012: primeira versão.

EOF