

2 Somadores e Multiplicadores

Objetivos São três os objetivos deste laboratório: (i) avaliar a corretude e o desempenho de três circuitos somadores, medir e comparar seus tempos de propagação; (ii) construir um modelo para um somador do tipo “seleção do vai-um” (*carry-select adder*); e (iii) escrever um modelo para o multiplicador combinacional, verificar sua corretude, e medir seu tempo de propagação.

Preparação Ler seções 1.6.1, 8.1 e 8.3 de [RH12] sobre somadores.

2.1 Material Disponibilizado Para Sua Tarefa

Da tarefa:

1. O trabalho pode ser efetuado em duplas;
2. copie para sua área de trabalho o arquivo com o código VHDL:

```
wget http://www.inf.ufpr.br/roberto/ci210/vhdl/l_somadores.tgz
```


expand-o com `tar xzf l_somadores.tgz`
mude para o diretório somadores 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; com um argumento o *script* dispara a execução de `gtkwave`: `./run.sh 1 & .`

2.2 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 $\{0, 1\}$.

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 nas seções 1.6.1, 8.1 e 8.3 de [RH12]. Os números inteiros são denotados por cadeias de bits $a_{n-1}a_{n-2}\cdots a_0$ que representam o número $a_{n-1} \cdot 2^{n-1} + a_{n-2} \cdot 2^{n-2} + \cdots + a_0 \cdot 2^0$. Em geral, o dígito menos significativo é representado no lado direito dos diagramas.

É possível que os modelos no arquivo `somador.vhd` contenham erros de projeto ou de modelagem. A verificação da corretude é parte de sua tarefa.

O modelo *adderCadeia* é um modelo estrutural do somador composto por 16 instâncias do somador completo e sem cadeia de adiantamento de vai-um. Este modelo contém informação aproximada da temporização do circuito. O modelo

adderCadeia,
adderAdiantaN

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 entidade `tb_somador` é vazia porque o programa de testes é autocontido e não tem interfaces com nenhum outro circuito. 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 em Prog. 1, 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 nove elementos para ilustrar as possibilidades.

testbench

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.

Prog. 1: Vetor de valores de entrada para testar os modelos.

```

type test_record is
  record
    a : reg16;      — entrada
    b : reg16;      — entrada
    f : bit;        — operacao: 0=ADD, 1=SUB
    c : reg16;      — saida esperada
    v : bit;        — vai-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'),
  — crescente novos vetores aqui, testes de soma f=0
  (x"0001",x"0001", '0',x"0002", '0'),
  — testes para subtracao f=1
  (x"0000",x"0000", '1',x"0000", '1')
);

```

O formato de um vetor de valores pode ser denotado, respectivamente como decimal, binário (prefixo `b`), octal (prefixo `o`), ou hexadecimal (prefixo `x`) ou texto/caracteres. Escalares com um único componente são denotados entre aspas simples (`'0'`), valores com mais de um componente (vetores) são denotados com aspas duplas – vetor vazio: `""`; vetor com 4 elementos: `"1101"`. Binários podem ser representados com ou sem o prefixo `b`. Inteiros não são vetores e portanto são representados sem aspas. O trecho de código no Prog. 2 contém alguns exemplos.

vetores de
inteiros, hexa,
octal, binário,
caracteres

Prog. 2: Atribuições a vetores de inteiros, bits, octais, hexa e *string*.

```

signal dec: integer;
signal bin: reg4;
signal oct: reg12;
signal hex: reg16;
signal msg: string;
...
dec <= 2012;           — inteiro
bin <= b"1001";      — binario, vetor de 4 bits
oct <= o"274";       — octal, vetor de 12 bits
hex <= x"4AF0";      — hexadecimal, vetor de 16 bits
msg <= "isso eh uma mensagem"; — string

```

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. Lembre que 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 dos modelos.

atributo **'range'**

O comando **assert**, em Prog. 3, 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.

assert

o projetista é o responsável por escrever os testes

Prog. 3: Verificação da corretude da saída.

```

assert resCad = esp_res
  report "cadeia: saida errada A=" & BV2STR(inpA) & " B=" & BV2STR(inpB)
  & " soma=" & BV2STR(resCAD) & " esperada=" & BV2STR(esp_res)
  & " som0/1sub=" & B2STR(v.f)
  severity error;

```

Ao final do laço a simulação termina no comando **assert FALSE**, 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.

Da tarefa: (continuação)

1. 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 no projeto. Os três modelos contém erros – sua tarefa é gerar vetores de teste que evidenciem os erros para então corrigir os modelos.

os modelos contém erros

2.3 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`, na linha 11, 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`.

Da tarefa: (continuação)

1. meça e compare os tempos de propagação de pior caso dos três somadores. Quais são os piores casos? Quais os ganhos de desempenho dos modelos com adiantamento, com relação ao modelo sem adiantamento? Use os cursores do `gtkwave` para medir os tempos de propagação;

tempo de propagação de pior caso

2.4 Multiplicador Combinacional

O arquivo `multiplicador.vhd` contém uma arquitetura vazia para o multiplicador combinacional baseado no circuito `mult-por-1`, definido no Prog. 4.

Prog. 4: Entidade e Arquitetura do `mult-por-1`.

```
entity m_p_1 is
    port (A,B : in reg16;
          S : in bit;
          R : out reg17)
end m_p_1;

architecture funcional of m_p_1 is
    component adderAdianta16 ...
    signal somaAB : reg17;
begin

    U_soma: adderAdianta16
        port map(A,B,somaAB(15 downto 0), '0', somaAB(16));

    R <= somaAB when S = '1' else ('0' & A);

end funcional;
```

Em pseudo-C, a função do circuito pode ser descrita por

```
#define mult-por-1(A,B,S,R) (R <= ( S ? A+B : 0&A ))
```

que é um *circuito combinacional*, sendo A e B inteiros representados em 16 bits, R um inteiro representado em 17 bits, S um bit, $X&Y$ a concatenação de X com Y , e $(x ? y : z)$ a expressão de seleção da linguagem C. Usando várias instâncias de `mult-por-1`, mostre como implementar um multiplicador combinacional de 16x16 bits.

parcela da multiplicação

O comando concorrente `when condicao else` pode ser usado na área concorrente de uma arquitetura para selecionar uma de duas possibilidades, como se fora um multiplexador. Na arquitetura do modelo `m_p_1`, o comando `when else` escolhe o valor que será atribuído a R em função do valor de S . Para simular o atraso através do multiplexador, é necessário acrescentar o atraso estimado para o multiplexador de duas entradas `t_mux2`: quando $S=1$, $R <= somaAB$;

`when else`

quando $S/=1$, $R <= '0' \& A$. Qual deve ser o valor da constante t_mux2 , na linha 26 de `packageWires.vhd` ?

```
R <= somaAB when S = '1' else ('0' & A) after t_mux2;
```

O arquivo `tb_multiplicador.vhd` contém o TB do multiplicador e seu conteúdo é similar ao *testbench* dos somadores.

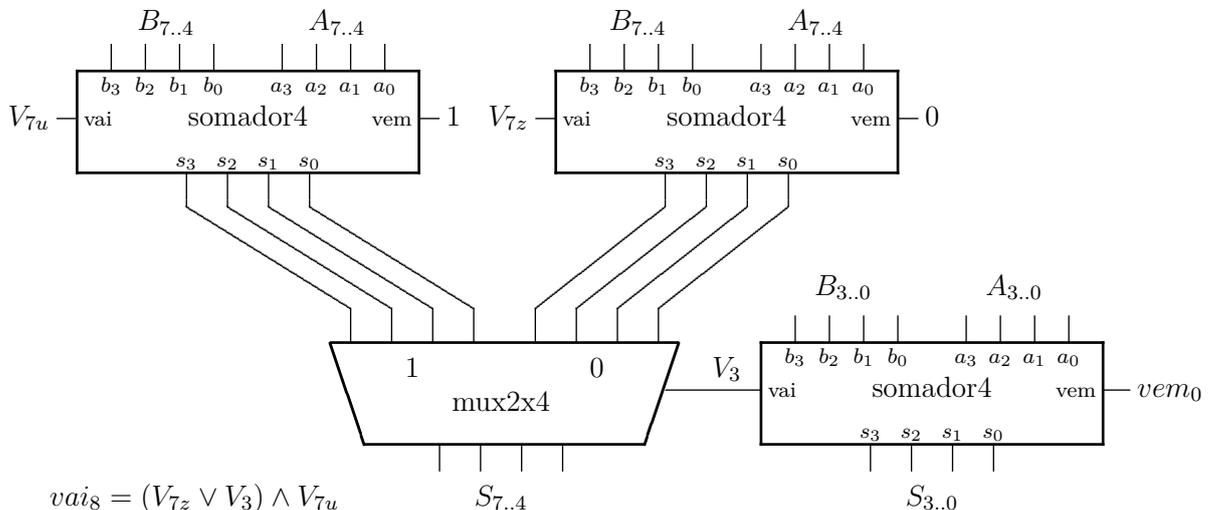
O arquivo `runMult.sh` é similar ao `run.sh`, compila e simula o multiplicador.

Da tarefa: (continuação)

1. complete o modelo do multiplicador combinacional;
2. acrescente os vetores de teste que achar conveniente ao TB do multiplicador;
3. edite `packageWires.vhd` e altere a definição da constante `simulate_time`, na linha 11, de 1 para 0. Compile o modelo e verifique sua funcionalidade com `gtkwave`;
4. retorne os tempos de propagação para os valores não-zero – mude `simulate_time` na linha 11 de `packageWires.vhd` de 0 para 1 – recompile e meça os tempos de propagação do multiplicador. Note que as variações podem ser grandes em função dos valores de entrada por causa da propagação de vai-um em cadeias que podem ser longas – no pior caso, propagação de vai-um do bit 0 até o bit 31 do produto.

2.5 Somador de 32 bits – *carry select adder*

Uma alternativa de projeto para a implementação de somadores é mostrada na figura abaixo, 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 e o valor de $V_3 = vai_3$ fica estável, este é usado para escolher uma das duas somas, através do multiplexador de 4 bits de largura. Este circuito é chamado de “somador com seleção de vai-um” (*carry select adder*). Verifique se a função lógica que determina o valor de vai_8 está correta.



Da tarefa: (continuação)

1. sua tarefa é escrever um modelo de somador com seleção de vai-um com 32 bits de largura, e para tanto use um dos modelos de somadores da Seção 2.2;
2. modifique o `test_record` no programa de testes para verificar a corretude do seu modelo: os operandos e a soma devem ter 32 bits. Note que, além de modificar o `test_record`, você deverá alterar o `test_array` para que os novos vetores de teste reflitam as modificações no `test_record` – veja o Prog. 5.
3. 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? Neste caso são usados somente dois somadores de 16 bits e o vai-um do primeiro é ligado diretamente à entrada vem-um do segundo.

Prog. 5: Vetor de valores de entrada para somador de 32 bits.

```

type test_record is
  record
    a : reg32;      — entrada
    b : reg32;      — entrada
    f : bit;        — operacao: 0=ADD, 1=SUB
    c : reg32;      — saida esperada
    v : bit;        — vai-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
  (x"00000000",x"00000000", '0',x"00000000", '0'),
  — crescente novos vetores aqui, testes de soma f=0
  (x"00000001",x"00000001", '0',x"00000002", '0'),
  — testes para subtracao f=1
  (x"00000000",x"00000000", '1',x"00000000", '1')
);

```

Referências

[RH12] *Sistemas Digitais e Microprocessadores*, R.A.Hexsel, 2012, Editora da UFPR.

[PJA04] *VHDL Tutorial*, <http://www.inf.ufpr.br/roberto/ci210/vhdl/vhdl-tutorial.pdf>