UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA - CT

CIRCUITOS DIGITAIS

SOMADORES

DCA0212.1 - Laboratório 6

Igor Michael Araujo de Macedo
Isaac de Lyra Júnior
Pedro Henrique de Freitas Silva
Vinícius Silva do Carmo

Natal, 11 de fevereiro de 2022

1. INTRODUÇÃO

O presente relatório tem por objetivo desenvolver os conceitos teóricos estudados sobre circuitos digitais de forma prática, mais especificamente sobre circuitos somadores. O desenvolvimento se dará utilizando VHDL, linguagem de descrição de *hardware*, para descrever somadores tanto de forma comportamental quanto utilizando lógica combinacional. Por fim, será utilizado o *software* ModelSim para fazer as simulações necessárias.

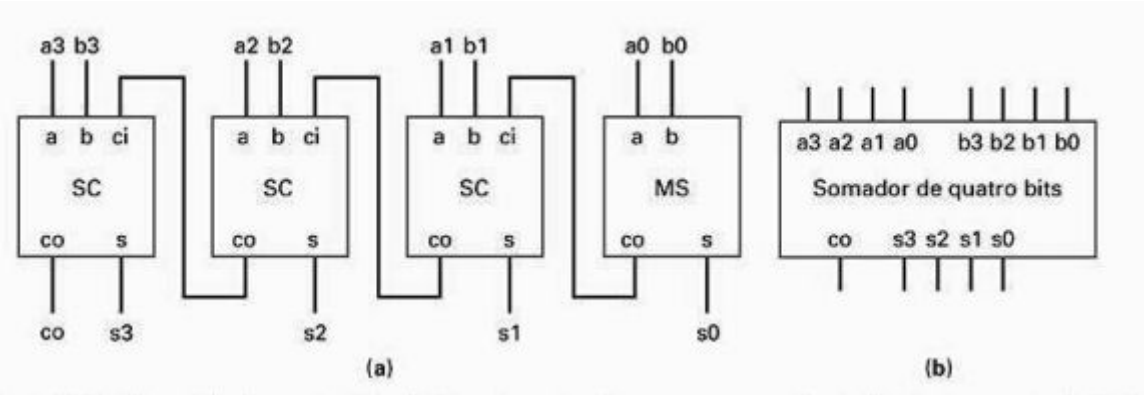
2. DESENVOLVIMENTO

Para dar início às implementações práticas, faz-se necessário, primeiramente, conhecer o funcionamento dos somadores. Eles são componentes combinacionais bastante importantes nos circuitos digitais justamente por realizar operações de adição, operações essas que são básicas e muito utilizadas. Um somador de *N* bits é capaz de somar dois números binários, *A* e *B*, de *N* bits, além de ser possível administrar um *carry C*, de um bit.

Uma das maneiras mais comuns de projetar um somador é utilizando dois componentes combinacionais chamados meio somador e somador completo. Eles serão utilizados em cascata para somar bit a bit cada par de bits das entradas *A* e *B*. O meio somador (MS) é composto por duas entradas e duas saídas de 1 bit cada. As entradas *a* e *b* representam os bits a serem somados, a saída *s* o resultado da adição, e a última saída, *co*, o chamado *carry out*, que funciona como o "vai um" da técnica utilizada, normalmente, para cálculos de adição feitos à mão. Já no o somador completo (SC) há três entradas e duas saídas: ele soma três bits, *a*, *b* e *ci*, sendo o último o *carry in*, resultado do "vai um" da operação anterior; gera uma soma *s*, e um bit de transporte, *carry out*, *co*.

Para realizar uma soma, por exemplo, entre dois números de 4 bits cada, deve-se utilizar um meio somador e três somadores completos, como pode ser visto na Figura 1. Para adicionar números com mais bits, basta utilizar a mesma ideia e adicionar mais SCs.

Figura 1 - Somador de quatro bits: (a) implementação usando três somadores completos e um meio somador e (b) símbolo para diagrama de blocos.



Fonte: (VAHID, 2008)

Tendo todos esses conhecimentos em mente e a fim de colocá-los em prática utilizando VHDL, o desenvolvimento do trabalho se dará em seis partes:

1. Projetar um somador de 2 bits utilizando descrição comportamental;
2. Projetar um somador de 2 bits utilizando lógica combinacional e portas lógicas;
3. Projetar um bloco meio somador;

4. Projetar um bloco somador completo;
5. Projetar um somador para entradas de 3 bits utilizando os blocos dos itens 3 e 4;
6. Projetar um somador para entradas de 6 bits utilizando o bloco do item 5.

2.1 Somador de 2 bits utilizando descrição comportamental

O somador de 2 bits possui três entradas: os números a serem somados *A* e *B*, de 2 bits cada, do tipo *std_logic_vector;* e a entrada *cin*, que indica o *carry in*, de 1 bit, do tipo *std_logic*. O somador também possui duas saídas, sendo *S*, de 2 bits, o resultado da adição, do tipo *std_logic_vector*; e *cout*, que indica o *carry out,* de 1 bit do tipo *std_logic.* A implementação completa utilizando descrição comportamental pode ser vista na Figura 2.

Figura 2 - Código do somador de 2 bits comportamental

```
entity somador2bits_comportamental is
  generic (
    width: integer := 2
  );

  port (
    cin: in std_logic;
    A: in std_logic_vector(width - 1 downto 0);
    B: in std_logic_vector(width - 1 downto 0);
    S: out std_logic_vector(width - 1 downto 0);
    cout: out std_logic
  );

end somador2bits_comportamental;

architecture behavior of somador2bits_comportamental is

  signal sum: std_logic_vector(width downto 0);

begin

  sum <= ('0' & A) + ('0' & B) + cin;
  S <= sum(width-1 downto 0);
  cout <= sum(width);

end behavior ;
```

Fonte: Autores.

2.2 Somador de 2 bits utilizando lógica combinacional

Antes de começarmos a implementar o circuito somador de 2 bits propriamente dito, precisamos primeiramente definir uma tabela verdade que será nosso guia para conseguir uma implementação mais semântica e organizada. Para isso, foram consideradas duas entradas de 2 bits cada e para tratar o resultado da operação foi necessário adotar uma saída de 3 bits, para caso ocorra *overflow*. A tabela verdade do circuito somador de 2 bits utilizando lógica combinacional pode ser vista na figura abaixo:

Figura 3 - Tabela verdade do circuito somador de 2 bits combinacional.

ENTRADAS				SAÍDAS		
A1	A0	B1	B0	S2	S1	S0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	1
1	1	1	1	1	1	0

Fonte: Autores.

Feito isso, podemos extrair equações lógicas para representar cada bit da saída da tabela verdade do circuito proposto. Após extrair as equações, podemos simplificá-las e obter equações mais fáceis de se trabalhar. Estas equações podem ser vistas na figura a seguir:

Figura 4 - Equações lógicas simplificadas da saída do circuito somador de 2 bits combinacional.

EQUAÇÕES
S2 = A1B0B1 + A0B0 + A0A1B1
S1 = A0'A1'B0 + A0'A1B0'B1 + A0'B0B1' + A0A1'B0' + A0B0'B1' + A0A1B0B1
S0 = A1'B1 + A1B1'

Fonte: Autores.

Depois deste passo, agora temos uma maior facilidade e organização para conseguir implementar o circuito proposto. Desta forma, utilizando o *software* ModelSim, definimos a

entidade do circuito somador e descrevemos quais entradas ele recebe e qual é a saída que ele nos retorna. Feito isso, podemos descrever como a saída vai se comportar de acordo com a entrada apenas descrevendo as equações lógicas obtidas no passo anterior. O código em VHDL do circuito somador de 2 bits com lógica combinacional pode ser vista na figura abaixo:

Figura 5 - Código do somador de 2 bits combinacional

```
entity somador2bits_logico is
  port(
    A: in bit_vector(1 downto 0);
    B: in bit_vector(1 downto 0);
    S: out bit_vector(2 downto 0)
  );
end somador2bits_logico;

architecture logic of somador2bits_logico is

  signal  A1, A0, B1, B0, nA1, nA0, nB1, nB0: bit;

  begin
    A1 <= A(1);
    A0 <= A(0);
    B1 <= B(1);
    B0 <= B(0);
    nA1 <= not(A(1));
    nA0 <= not(A(0));
    nB1 <= not(B(1));
    nB0 <= not(B(0));

    S(2) <= (A0 and B1 and B0) or (A1 and B1) or (A1 and A0 and B0);

    S(1) <= (nA1 and nA0 and B1) or (nA1 and A0 and nB1 and B0) or
            (nA1 and B1 and nB0) or (A1 and nA0 and nB1) or
            (A1 and nB1 and nB0) or (A1 and A0 and B1 and B0);

    S(0) <= (nA0 and B0) or (A0 and nB0);

end logic;
```

Fonte: Autores.

2.3 Bloco meio somador

Para construir o bloco meio somador, foi construída a tabela-verdade da Figura 6.

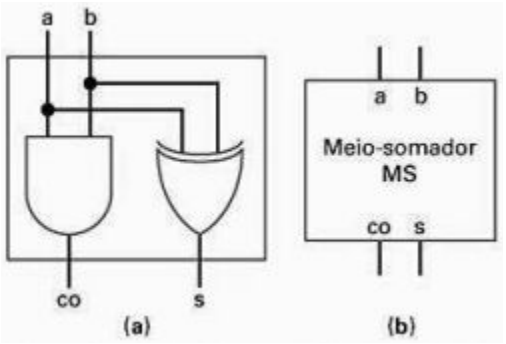
Figura 6 - Tabela-verdade do bloco meio somador.

Entradas		Saídas	
a	b	co	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Fonte: Autores.

Ao analisar a tabela, é possível perceber que a saída *s* tem nível alto quando as entradas são diferentes, característica de uma porta *OR* exclusiva (*XOR*), já a saída *co* tem o comportamento de uma porta *AND*. Sendo assim, um meio somador pode ser representado assim como na Figura 7.

Figura 7 - Meio somador: (a) circuito e (b) símbolo para diagrama de blocos.



Fonte: (VAHID, 2008).

Tendo o comportamento estudado, o próximo passo foi implementá-lo no VHDL, onde o circuito possui duas entradas, *A* e *B*, as quais serão somadas, uma saída *S*, resultado da adição e uma saída *cout* que é o *carry out*. Todas as entradas e saídas são de 1 bit cada e o código pode ser visto na Figura 8.

Figura 8 - Código do bloco meio somador

```
entity half_add is
  port (
    A, B: in bit;
    S, cout: out bit
  );
end half_add;

architecture logic of half_add is

begin
  S <= A xor B;
  cout <= A and B;
end logic;
```

Fonte: Autores.

2.4 Bloco somador completo

Para construir o bloco somador completo, foi construída a tabela-verdade da Figura 9.

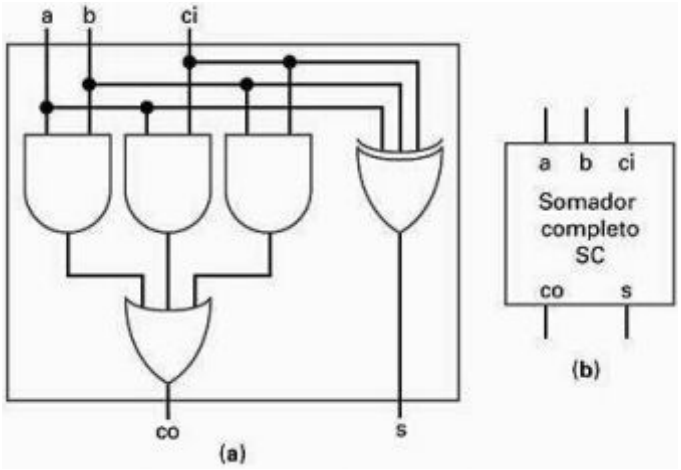
Figura 9 - Tabela-verdade do bloco somador completo.

Entradas			Saída	
a	b	ci	co	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Fonte: Autores.

Ao analisar a tabela, é possível retirar o circuito equivalente para ela. Ele pode ser visto na Figura 10.

Figura 10 - Somador completo: (a) circuito e (b) símbolo para diagrama de blocos.



Fonte: (VAHID, 2008).

Em seguida, o bloco foi implementado no VHDL, onde o circuito possui três entradas, sendo *A* e *B* as entradas somadas e *ci* o bit de *carry in;* uma saída *S*, resultado da adição e uma saída *cout* que é o *carry out*. Todas as entradas e saídas são de 1 bit cada e o código pode ser visto na Figura 11.

Figura 11 - Código do bloco somador completo

```
entity full_add is
  port (
    A, B, cin: in bit;
    S, cout: out bit
  );
end full_add;

architecture logic of full_add is

begin
  S <= A xor B xor cin;
  cout <= (B and cin) or (A and cin) or (A and B);
end logic;
```

Fonte: Autores.

2.5 Somador de 3 bits

Para implementar o somador de 3 bits utilizando os blocos desenvolvidos nas seções 2.4 e 2.5, foi utilizada a mesma ideia da Figura 1, mas como deseja-se somar dois números de 3 bits cada, serão utilizados um MS e dois SC. Para isso, no VHDL será utilizado a composição de blocos, onde o bloco *somador3bits* reutiliza os blocos *half_add* e *full_add*. Todo o desenvolvimento pode ser visto na Figura 12.

Figura 12 - Código do somador de 3 bits

```
entity somador3bits is
  port (
    A, B: in bit_vector(2 downto 0);
    O: out bit_vector(2 downto 0);
    cout: out bit
  );
end somador3bits;

architecture logic of somador3bits is

component half_add is
  port (
    A, B: in bit;
    S, cout: out bit
  );
end component;

component full_add
  port (
    A, B, cin: in bit;
    S, cout: out bit
  );
end component;

signal carry: bit_vector(1 downto 0);

begin
  S0: half_add port map(A(0), B(0), O(0), carry(0));
  S1: full_add port map(A(1), B(1), carry(0), O(1), carry(1));
  S2: full_add port map(A(2), B(2), carry(1), O(2), cout);
end logic;
```

Fonte: Autores.

2.6 Somador de 6 bits

Para desenvolver o somador de 6 bits foi utilizado como componente o somador de 3 bits desenvolvido na seção 2.5. Como o somador de 3 bits desenvolvido não possui entrada de *Carry In*, foi necessário adotar uma estratégia para que a soma dos bits fosse feita da forma correta. Primeiro é somado os 3 LSBs da soma, o resultado dessa soma será somado com os 3 MSBs de um dos números com o *Carry Out* da soma anterior, no nosso caso, escolhemos somar os 3 MSBs do dado A. Por fim, o resultado dessa soma é somado com os outros 3 MSBs do outro dado, que no nosso caso são os 3 números mais significativos do dado B.

Figura 13 - Código do somador de 6 bits

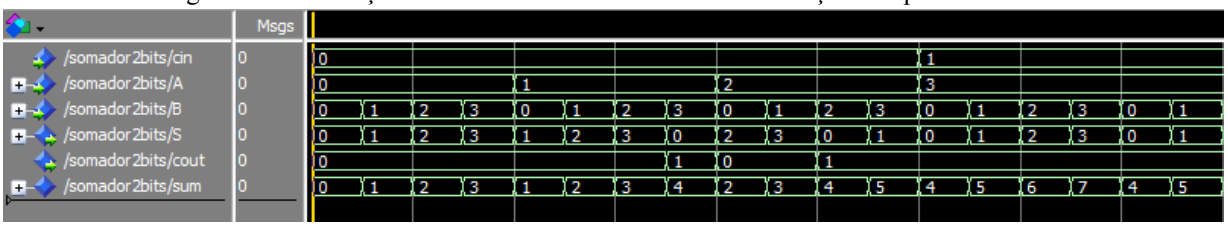
```
entity somador6bits is
  port(
    A,B: in bit_vector (5 downto 0);
    O: out bit_vector (5 downto 0);
    cout: out bit
  );
end somador6bits;

architecture logic of somador6bits is

component somador3bits is
  port(
    A, B: in bit_vector(2 downto 0);
    O: out bit_vector(2 downto 0);
    cout: out bit
  );
end component;

signal RES0,RES1,RES2,RES3: bit_vector(2 downto 0);
signal carry: bit_vector(1 downto 0);

begin
    S0: somador3bits port map(A(2 downto 0), B(2 downto 0), RES1, RES0(0));
    S1: somador3bits port map(A(5 downto 3), RES0, RES2, carry(0));
    S2: somador3bits port map(RES2, B(5 downto 3), RES3, carry(1));

    cout <= RES0(0) or carry(1);

    O(2 downto 0) <= RES1;
    O(5 downto 3) <= RES3;
end logic;
```

Fonte: Autores.

3. RESULTADOS

Para simular o somador de 2 bits utilizando descrição comportamental foram forçados diversos valores para as entradas *A*, *B* e *cin* e analisadas as saídas correspondentes (observação para que, apesar dos valores serem dados em binário, na simulação foi utilizado o modo de visualização decimal para facilitar a análise). Pegando o caso, por exemplo, que *cin = 0, A = 00* e *B=01*, temos que *S = 01* e *cout = 0*; para *cin = 0, A = 01 = 1* e *B = 11 = 3*, temos que *S = 0* e *cout = 1*, é como se a saída fosse *100 = 4;* para *cin = 1, A = 11 = 3* e $B = 11 = 3$, temos que *S = 11 = 3* e *cout = 1*, é como se a saída fosse *111 = 7*. Na Figura 14 é possível ver todas as simulações feitas para este somador.

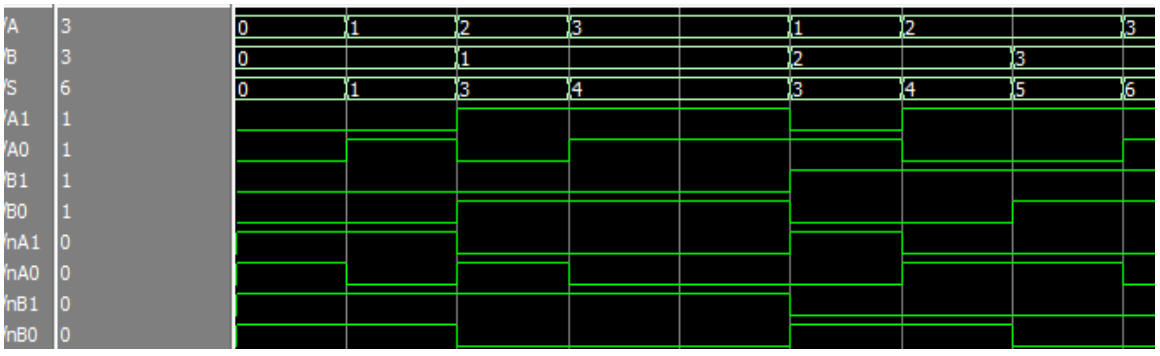
Figura 14 - Simulação do somador de 2 bits utilizando descrição comportamental.



Fonte: Autores.

A simulação do circuito somador de 2 bits utilizando lógica combinacional se deu através de diversos valores para as entradas distintas entre *A* e *B* e analisadas as saídas correspondentes. Tendo como exemplo o caso em que *A = 00 (0)* e *B=00 (0)*, temos que *S = 000 (0)*; para *A = 01 (1)* e *B = 01 (1)*, temos que *S = 10 (2);* para, *A = 10 (2)* e *B=01 (1)*, temos que *S = 11 (3)* e assim por diante. Na Figura 15 é possível ver todas as simulações feitas para este somador.

Figura 15 - Simulação do somador de 2 bits utilizando lógica combinacional.

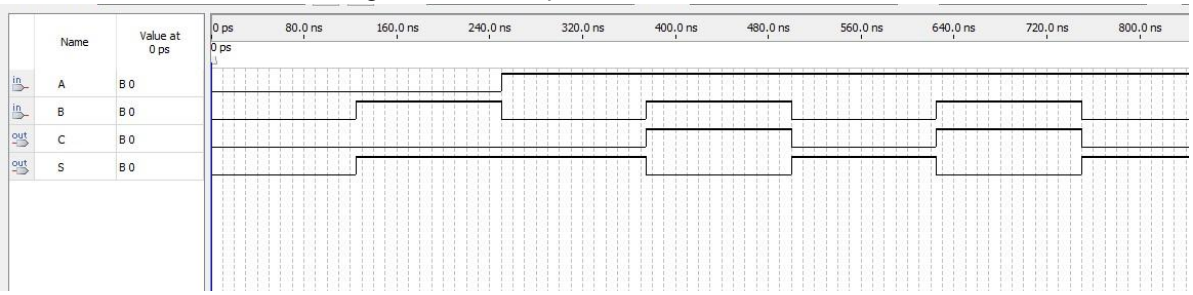


Fonte: Autores.

Na simulação do circuito bloco meio somador foi inserido nas entradas A e B todos os valores possíveis para a comprovação dos valores corretos das saídas C (cout) e S. Como por exemplo *A = 0* e *B=1*, resultou na saída *C = 0* e para a saída *S = 1,* ou seja, para este caso, a

soma *0+1* não possui *cout* (*C=0*) e S está em nível alto (*S=1*), logo *0+1 = 01*. Na Figura 16 é possível as simulações realizadas para este bloco.

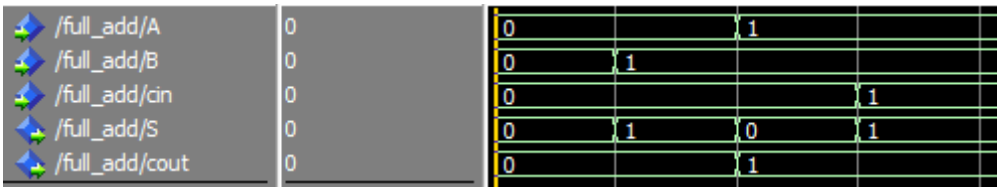
Figura 16 - Simulação do bloco meio somador.



Fonte: Autores.

Na simulação do circuito do somador completo foram inseridas todas as possíveis combinações para as entradas A, B e cin. Primeiro foi introduzido *A = 0*, *B = 0* e *cin=0*, que resultou obviamente em *S = 0* e *cout = 0*. Em seguida foi introduzido *A = 0*, *B = 1* e *cin = 0*, que resultou em *S = 1* e *cout = 0*. Na sequência foi introduzido *A = 1*, *B = 1* e *cin = 0*, resultando em *S = 0* e *cout = 1*. Por fim, foi introduzido *A = 1*, *B = 1* e *cin = 1*, o que resultou em *S = 1* e *cout = 1*. Na Figura 17 é possível observar o resultado das simulações para este componente.

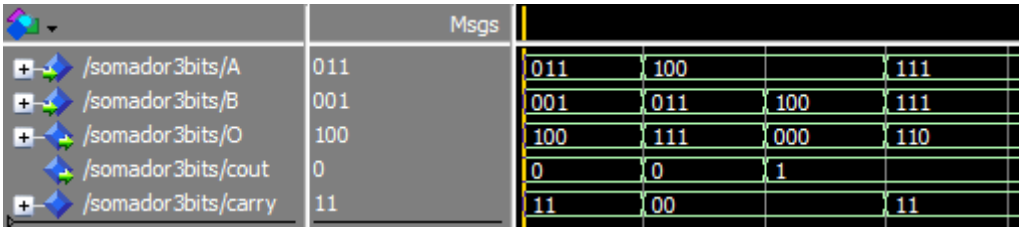
Figura 17 - Simulação do bloco somador completo.



Fonte: Autores.

Na simulação do circuito somador de 3 bits foram introduzidas 4 combinações diferentes de valores para as entradas A e B. Primeiro foi introduzido *A = 011* e *B = 001*, o que resultou em *O = 100* e *cout = 0*. Em sequência foi introduzido *A = 100* e *B = 011*, o que resultou em *O = 111* e *cout = 0*. Em seguida foi introduzido *A = 100* e *B = 100*, o que resultou em *O = 000* e *cout = 1*. Por fim, foi introduzido *A = 111* e *B = 111*, o que resultou em *O = 110* e *cout = 1*. Na Figura 18 é possível observar o resultado das simulações para este componente.

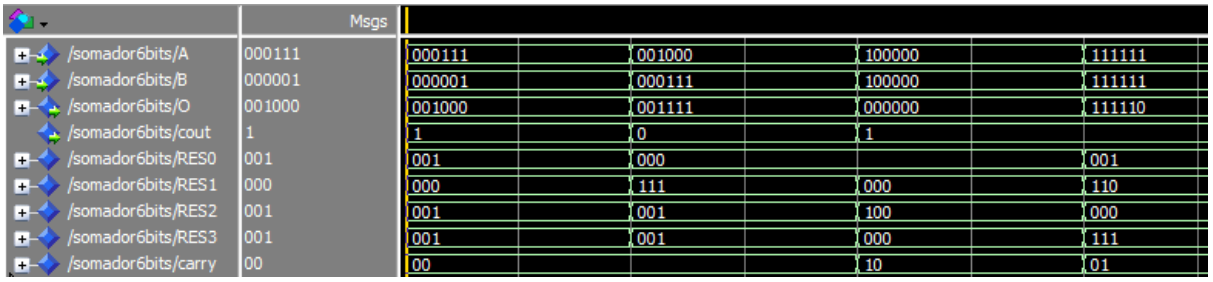
Figura 18 - Simulação do bloco somador de 3 bits.



Fonte: Autores.

Na simulação do somador de 6 bits foi seguida a mesma lógica da simulação anterior, logo, foram introduzidas 4 combinações diferentes de valores para as entradas A e B. Primeiro foi introduzido *A=000111* e *B=000001*, o que resultou em *O=001000* e *cout=0*. Em seguida, foi introduzido *A=001000* e *B=000111*, o que resultou em *O=001111* e *cout=0*. Em sequência, foi introduzido *A=100000* e *B=100000*, o que resultou em *O=000000* e *cout=1*. Por fim, foi introduzido *A=111111* e *B=111111*, o que resultou em *O=111110* e *cout=1*. Na Figura 19 é possível observar o resultado das simulações para este componente.

Figura 19 - Simulação do bloco somador de 6 bits.



Fonte: Autores.

4. CONCLUSÃO

O presente trabalho teve como objetivo pôr em prática os conhecimentos estudados sobre somadores. Para isso, foram implementados e simulados circuitos de meio-somadores, somadores completos, somador de 2 bits de descrição comportamental e com portas lógicas, somador de 3 bits e somador de 6 bits. Para esta solução construímos 6 entidades utilizando a linguagem VHD, os quais o código-fonte completo pode ser visto no Anexo A, sendo uma para meio-somador, uma de somador completo, duas de somador de 2 bits, sendo uma utilizando a descrição comportamental e outra utilizando portas lógicas, uma sendo somador de 3 bits e outra de somador de 6 bits. Todas as entidades foram testadas e simuladas utilizando o software Modelsim. Após todo o processo, a simulação retornou os resultados esperados e, com isso, podemos concluir que foi possível extrair o máximo de conhecimento e proveito do problema.

REFERÊNCIAS

VAHID, Frank. **Sistemas digitais: projeto, otimização e HDLS**. Rio Grande do Sul: Artmed Bookman, 2008. 558 p

ANEXO A - CÓDIGO-FONTE

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

-----------------------
-- SOMADOR DE 2 BITS
--   COMPORTAMENTAL
-----------------------
entity somador2bits_comportamental is
  generic (
    width: integer := 2
  );

  port (
    cin: in std_logic;
    A: in std_logic_vector(width - 1 downto 0);
    B: in std_logic_vector(width - 1 downto 0);
    S: out std_logic_vector(width - 1 downto 0);
    cout: out std_logic
  );

end somador2bits_comportamental;

architecture behavior of somador2bits_comportamental is

  signal sum: std_logic_vector(width downto 0);

begin

  sum <= ('0' & A) + ('0' & B) + cin;
  S <= sum(width-1 downto 0);
  cout <= sum(width);

end behavior ;

-----------------------
-- SOMADOR DE 2 BITS
--   COMBINACIONAL
-----------------------
entity somador2bits_logico is
  port(
    A: in bit_vector(1 downto 0);
    B: in bit_vector(1 downto 0);
    S: out bit_vector(2 downto 0)
  );
end somador2bits_logico;

architecture logic of somador2bits_logico is
```

```
  signal  A1, A0, B1, B0, nA1, nA0, nB1, nB0: bit;

  begin
    A1 <= A(1);
    A0 <= A(0);
    B1 <= B(1);
    B0 <= B(0);
    nA1 <= not(A(1));
    nA0 <= not(A(0));
    nB1 <= not(B(1));
    nB0 <= not(B(0));

     S(2) <= (A0 and B1 and B0) or (A1 and B1) or (A1 and A0
and B0);

     S(1) <= (nA1 and nA0 and B1) or (nA1 and A0 and nB1 and
B0) or
           (nA1 and B1 and nB0) or (A1 and nA0 and nB1) or
           (A1 and nB1 and nB0) or (A1 and A0 and B1 and B0);

    S(0) <= (nA0 and B0) or (A0 and nB0);

end logic;


-----------------------
--   MEIO SOMADOR
-----------------------
entity half_add is
  port (
    A, B: in bit;
    S, cout: out bit
  );
end half_add;

architecture logic of half_add is

begin
  S <= A xor B;
  cout <= A and B;
end logic;


-----------------------
--  SOMADOR COMPLETO
-----------------------
entity full_add is
  port (
    A, B, cin: in bit;
    S, cout: out bit
  );
```

```
end full_add;

architecture logic of full_add is

begin
  S <= A xor B xor cin;
  cout <= (B and cin) or (A and cin) or (A and B);
end logic;


-----------------------
-- SOMADOR DE 3 BITS
-----------------------
entity somador3bits is
  port (
    A, B: in bit_vector(2 downto 0);
    O: out bit_vector(2 downto 0);
    cout: out bit
  );
end somador3bits;

architecture logic of somador3bits is

component half_add is
  port (
    A, B: in bit;
    S, cout: out bit
  );
end component;

component full_add
  port (
    A, B, cin: in bit;
    S, cout: out bit
  );
end component;

signal carry: bit_vector(1 downto 0);

begin
  S0: half_add port map(A(0), B(0), O(0), carry(0));
  S1: full_add port map(A(1), B(1), carry(0), O(1), carry(1));
  S2: full_add port map(A(2), B(2), carry(1), O(2), cout);
end logic;


-----------------------
-- SOMADOR DE 6 BITS
-----------------------
entity somador6bits is
  port(
```

```
    A,B: in bit_vector (5 downto 0);
    O: out bit_vector (5 downto 0);
    cout: out bit
  );
end somador6bits;

architecture logic of somador6bits is

component somador3bits is
  port(
    A, B: in bit_vector(2 downto 0);
    O: out bit_vector(2 downto 0);
    cout: out bit
  );
end component;

signal RES0,RES1,RES2,RES3: bit_vector(2 downto 0);
signal carry: bit_vector(1 downto 0);

begin
     S0: somador3bits port map(A(2 downto 0), B(2 downto 0),
RES1, RES0(0));
      S1: somador3bits port map(A(5 downto 3), RES0, RES2,
carry(0));
      S2: somador3bits port map(RES2, B(5 downto 3), RES3,
carry(1));

    cout <= RES0(0) or carry(1);

    O(2 downto 0) <= RES1;
    O(5 downto 3) <= RES3;
end logic;
```