Arquitetura de Computadores - Somador Vetorial

Carolina Santiago
Gustavo Roxo
Zuilho Segundo

Engenharia de Computação e Informação
Universidade Federal do Rio de Janeiro - UFRJ
Brasil
Julho 2023

Sumário

1 Introdução

O atual relatório visa discorrer sobre o desenvolvimento de um somador vetorial com a capacidade de realizar as instruções vetoriais de para soma e subtração. Dessa forma, o somador recebe entradas de 32 bits e é capaz de realizar a operação em um conjunto de sessões independentes, variando nos tamanhos de 4 a 32 em potências de dois. Assim, somos capazes de realizar operações com vetores contendo até 8 valores independentes de 4 bits cada, passando apenas o tamanho do vetor.

Por fim, caso haja interesse, é possível encontrar os circuitos utilizados no seguinte link do GitHub: `https://github.com/carol-santiago/Somador-Vetorial-EEL580`.

2 Explicação do Código

O circuito apresentado é um somador e subtrator vetorial de n bits em paralelo, onde $n = \{4, 8, 16, 32\}$. Ele foi implementado em VHDL para realizar a operação de soma ou subtração de dois vetores de tamanho n de forma eficiente.

2.1 Entradas e Saídas

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY SomadorVetorial IS
  PORT (
    -- Inputs
    A_i        : IN  std_logic_vector(31 DOWNTO 0);
    B_i        : IN  std_logic_vector(31 DOWNTO 0);
    vecSize_i  : IN  std_logic_vector(1 DOWNTO 0);
    mode_i     : IN  std_logic;

    -- Outputs
    S_o        : OUT std_logic_vector(31 DOWNTO 0)
  );
END SomadorVetorial;
```

Nesta seção, definimos o módulo VHDL chamado `SomadorVetorial` que possui quatro entradas e uma saída. As entradas são:

- `A_i`: Entrada de vetor A com tamanho de 32 bits.
- `B_i`: Entrada de vetor B com tamanho de 32 bits.
- `vecSize_i`: Entrada que representa o tamanho do vetor (2 bits).
- `mode_i`: Entrada que define o modo da operação. 0 para adição e 1 para subtração.

A saída é `S_o`, que é o resultado da operação de soma ou subtração.

2.2 Declaração de Sinais

```
ARCHITECTURE TypeArchitecture OF SomadorVetorial IS
  signal s_C : std_logic_vector(31 downto 0);
  signal s_P : std_logic_vector(31 downto 0);
  signal Carry : std_logic_vector(31 downto 0);
  signal s_Soma  : std_logic_vector(31 downto 0);
  signal B_op : std_logic_vector(31 downto 0);
```

Nesta seção, declaramos os sinais necessários para implementar o circuito. São eles:

- `s_C`: Sinal que representa o resultado da operação AND bit a bit entre os vetores A e B.
- `s_P`: Sinal que representa o resultado da operação XOR bit a bit entre os vetores A e B.
- `Carry`: Sinal que representa o resultado da propagação do carry.
- `s_Soma`: Sinal que representa o resultado da soma ou subtração dos vetores A e B.
- `B_op`: Sinal que representa o complemento de 2 do vetor B, usado na operação de subtração.

A operação de complemento de 2 é realizada na atribuição de `B_op`, onde o sinal `B_i` é negado (`not`) quando `mode_i` é igual a '1' (modo de subtração) e atribuído normalmente quando `mode_i` é igual a '0' (modo de adição).

2.3 Complemento de 2 na Subtração e Geração de C e P

Nesta seção, realizamos o complemento de 2 na subtração e geramos os sinais auxiliares C (carry gerado) e P (carry propagado) necessários para a operação de soma ou subtração.

2.3.1 Complemento de 2

O complemento de 2 é uma técnica utilizada para representar números negativos em sistemas binários. No caso de operação de subtração, precisamos inverter (bit a bit) o vetor B (`B_i`) antes de realizar a soma com o vetor A (`A_i`).

O sinal `B_op` é utilizado para armazenar o complemento de 2 de `B_i`. Usamos a seguinte expressão para realizar o complemento de 2:

$$B_{op}[n] = \begin{cases} B_i[n] & \text{se } \texttt{mode\_i} =' 0' \text{ (modo de adição)} \\ \sim B_i[n] & \text{se } \texttt{mode\_i} =' 1' \text{ (modo de subtração)} \end{cases}$$

Onde n é a posição do bit (0 a 31). Ela foi implementada da seguinte maneira no código:

```
--Complemento de 2 na subtração
B_op <= B_i when (mode_i = '0') else
        not(B_i) when (mode_i = '1');
```

2.3.2 Geração de C e P

Os sinais auxiliares C e P são utilizados para calcular o carry antecipado em cada posição para a operação de soma ou subtração. São calculados por meio das operações lógicas AND e XOR bit a bit entre os vetores `A_i` e `B_op`.

O laço `for` percorre cada bit dos vetores `A_i` e `B_op` e realiza as seguintes operações:

$$s\_C[n] = A_i[n] \wedge B_{op}[n] \quad \text{(carry gerado)}$$

$$s\_P[n] = A_i[n] \oplus B_{op}[n] \quad \text{(carry propagado)}$$

onde n é a posição do bit (0 a 31).

Os sinais `s_C` e `s_P` armazenam os resultados das operações de carry gerado e carry propagado, respectivamente.

```
    GERADORCARRY : for nn in 0 to 31 generate
    s_C(nn)   <= A_i(nn) and B_op(nn);
    s_P(nn)   <= A_i(nn) xor B_op(nn);
  end generate GERADORCARRY;
```

2.3.3 Geração de Carry Antecipado

A partir dos valores de s_C e s_P, podemos calcular o carry bit-a-bit para cada bit de saída (Carry(0) até Carry(31)). Para Carry(0), o cálculo é simples e depende apenas do modo de operação, onde '0' representa uma operação de adição e '1' representa uma operação de subtração:

```
Carry(0) <= '0' when mode_i = '0' else '1';
```

A partir do Carry(1) até Carry(3):

```
Carry(1) <= s_C(0) or (s_P(0) and Carry(0));
Carry(2) <= s_C(1) or (s_P(1) and Carry(1));
Carry(3) <= s_C(2) or (s_P(2) and Carry(2));
```

Depois disso, a lógica de cálculo do carry começa a variar dependendo do valor de vecSize_i e mode_i. Por exemplo, para Carry(4), temos o seguinte cálculo:

```
Carry(4) <= '0' when (vecSize_i = "00" and mode_i ='0') else
            '1' when (vecSize_i = "00" and mode_i ='1') else
            s_G(3) or (s_P(3) and Carry(3));
```

Essa lógica é repetida para cada bloco subsequente de bits múltiplo de 4 (8, 12, 16, 20) dependendo do valor de vecSize_i e mode_i. Essa abordagem garante que o carry seja propagado corretamente entre os blocos de bits para operações com vetores maiores.

```
Carry(8) <= '0' when (vecSize_i = "00" and mode_i ='0')
                  '1' when (vecSize_i = "00" and mode_i ='1')
                  '0' when (vecSize_i = "01" and mode_i ='0')
                  '1' when (vecSize_i = "01" and mode_i ='1')
                  s_C(7) or (s_P(7) and Carry(7));

Carry(9) <= s_C(8) or (s_P(8) and Carry(8));
Carry(10) <= s_C(9) or (s_P(9) and Carry(9));
Carry(11) <= s_C(10) or (s_P(10) and Carry(10));

...

Carry(16) <= '0' when (vecSize_i = "00" and mode_i ='0') else
                  '1' when (vecSize_i = "00" and mode_i ='1') else
                  '0' when (vecSize_i = "01" and mode_i ='0') else
                  '1' when (vecSize_i = "01" and mode_i ='1') else
                  '0' when (vecSize_i = "10" and mode_i ='0') else
                  '1' when (vecSize_i = "10" and mode_i ='1') else
                  s_C(15) or (s_P(15) and Carry(15));

Carry(17) <= s_C(16) or (s_P(16) and Carry(16));
Carry(18) <= s_C(17) or (s_P(17) and Carry(17));
Carry(19) <= s_C(18) or (s_P(18) and Carry(18));
...
```

Após a geração dos bits de carry, realizamos as somas bit-a-bit para obter o resultado final da operação de soma ou subtração:

```
GERADOR_SOMA : for mm in 0 to 31 generate
    s_Soma (mm) <= A_i(mm) xor B_op(mm) xor Carry(mm);
end generate GERADOR_SOMA;
```

O resultado final é armazenado na saída S_o, que representa o resultado da operação de soma ou subtração de N bits em paralelo.

3 Resultados

Relembrando as configurações dos sinais de controle:

vecSize_i	operação
00	8 somas de 4 bits
01	4 somas de 8 bits
10	2 somas de 16 bits
11	1 soma de 32 bits

mode_i	operação
0	soma
1	subtração

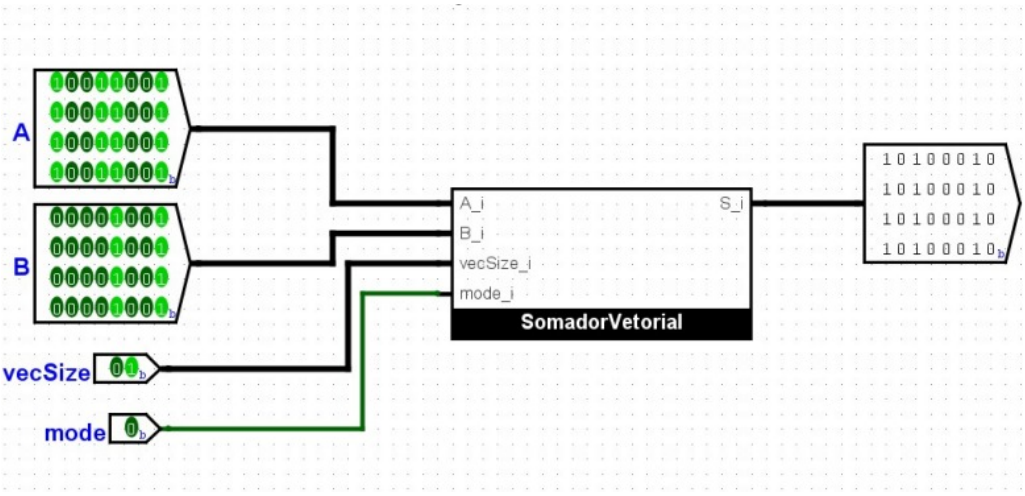

Figura 1: Teste 1

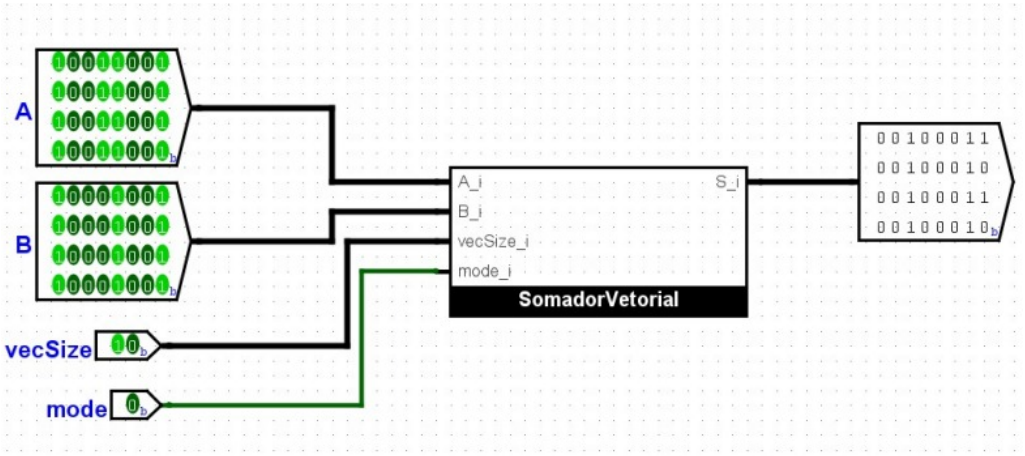

Figura 2: Teste 2

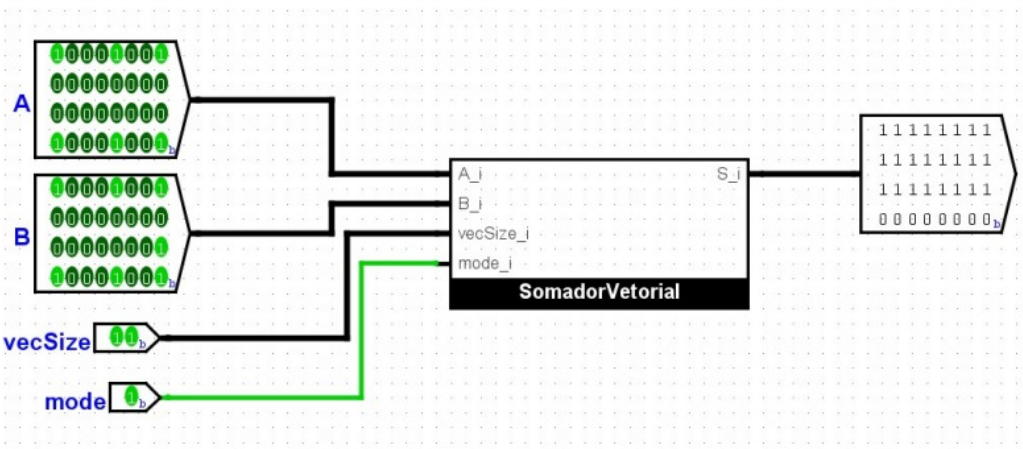

Figura 3: Teste 3

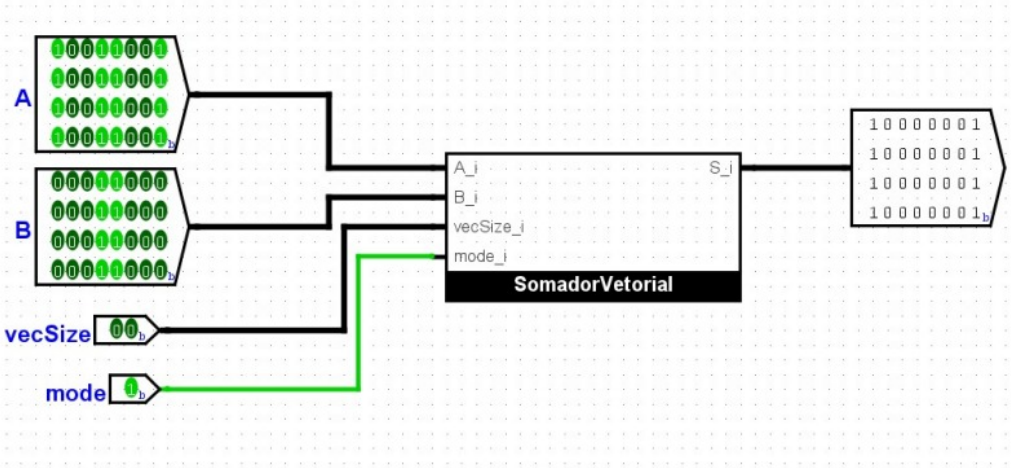

Figura 4: Teste 4