

# Arquitetura de Computadores

EEL580

## Trabalho Prático — Somador Vetorial

Carlos Vinicius de Sousa Antunes da Silva

DRE: 122155694

Otávio Braga Moraes

DRE: 124012410

Engenharia de Computação e Informação

Universidade Federal do Rio de Janeiro — UFRJ

Dezembro de 2025

## 1 Introdução

Um somador vetorial constitui um elemento fundamental na realização de operações de adição binária de forma paralela. Esse tipo de circuito é amplamente empregado em sistemas computacionais, uma vez que possibilita o processamento simultâneo de múltiplos bits, resultando em maior eficiência e desempenho quando comparado a arquiteturas de soma sequenciais. A principal característica desse dispositivo está na exploração do paralelismo inerente ao hardware, permitindo a execução de operações aritméticas sobre diferentes larguras de palavra. Neste trabalho, apresenta-se o desenvolvimento e a implementação de um somador vetorial capaz de operar com diferentes tamanhos de dados.

Os códigos desenvolvidos ao longo deste projeto encontram-se disponíveis neste repositório no [GitHub](#).

## 2 Projeto

O projeto do somador vetorial tem como objetivo desenvolver um circuito capaz de realizar operações de soma e subtração em vetores de tamanhos variados: 4, 8, 16 e 32 bits. Para garantir maior eficiência, foi utilizada a técnica de *Carry-Lookahead*, que permite o cálculo antecipado dos sinais de *carry*, reduzindo o atraso em relação ao método tradicional de *Carry Ripple*.

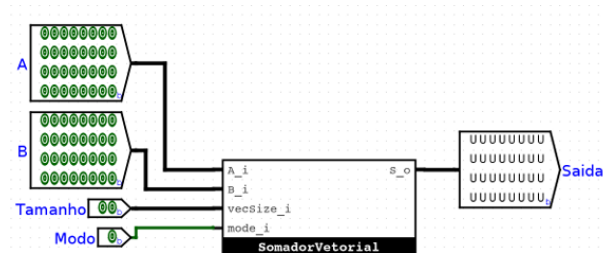


Figura 1: Módulo do somador vetorial.

### 3 Funcionamento

O módulo do somador vetorial possui as seguintes entradas:

- Dois vetores de 32 bits correspondentes aos operandos, denominados  $A_i$  e  $B_i$ ;
- Um vetor de 2 bits, denominado  $vecSize$ , responsável por definir o tamanho do bloco:
  - 00 — 4 bits
  - 01 — 8 bits
  - 10 — 16 bits
  - 11 — 32 bits
- Um bit de controle denominado  $mode$ , onde:
  - 0 — Soma
  - 1 — Subtração

A saída do sistema é um vetor de 32 bits, denominado  $S_o$ , que contém o resultado da operação selecionada.

#### 3.1 Blocos

Para que a operação seja realizada corretamente, os vetores de entrada são divididos em blocos de acordo com o tamanho selecionado. Essa divisão influencia diretamente a propagação do sinal de *carry*.

- 4 bits: 8 blocos
- 8 bits: 4 blocos
- 16 bits: 2 blocos
- 32 bits: 1 bloco

## 3.2 Operações

A lógica do somador de 1 bit segue as combinações apresentadas na Tabela 1.

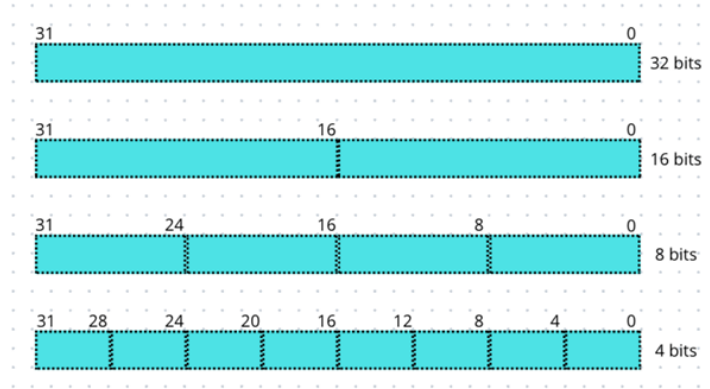


Figura 2: Divisão dos blocos do vetor.

Tabela 1: Tabela verdade do somador de 1 bit

A	B	$C_{in}$	Soma	$C_{out}$
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

A expressão lógica da soma pode ser escrita como:

$$\begin{aligned}
 Soma &= A \cdot \bar{B} \cdot \bar{C}_{in} + \bar{A} \cdot B \cdot \bar{C}_{in} + \bar{A} \cdot \bar{B} \cdot C_{in} + A \cdot B \cdot C_{in} \\
 &= \bar{C}_{in} \cdot (A \cdot \bar{B} + \bar{A} \cdot B) + C_{in} \cdot (\bar{A} \cdot \bar{B} + A \cdot B) \\
 &= \bar{C}_{in} \cdot (A \oplus B) + C_{in} \cdot (A \odot B) \\
 &= (A \oplus B) \oplus C_{in}
 \end{aligned}$$

$$\begin{aligned}
 C_{out} &= A \cdot B \cdot \bar{C}_{in} + \bar{A} \cdot B \cdot C_{in} + A \cdot \bar{B} \cdot C_{in} + A \cdot B \cdot C_{in} \\
 &= A \cdot B(\bar{C}_{in} + C_{in}) + C_{in} \cdot (\bar{A} \cdot B + A \cdot \bar{B}) \\
 &= A \cdot B + (A \oplus B) \cdot C_{in}
 \end{aligned}$$

### 3.2.1 Adição

Na operação de adição, o *carry* inicial de cada bloco é definido como zero. A soma é realizada bit a bit de acordo com a expressão apresentada anteriormente.

### 3.2.2 Subtração

Na subtração, é calculado o complemento de dois do vetor  $B$ . Para isso, os bits de  $B$  são invertidos e o valor 1 é somado no início de cada bloco, o que é implementado ajustando o *carry* inicial para 1.

A expressão da subtração é dada por:

$$S_i = A_i \oplus (-B)_i \oplus C_i$$

### 3.2.3 Carry

O cálculo do *carry* é o ponto central para o funcionamento do projeto, pois a partir dele é possível manipular se ele será propagado ou não em determinado segmento do vetor.

No início de cada bloco, o *carry* deve possuir um valor pré-definido de acordo com a operação realizada: 0 para soma e 1 para subtração. Os demais *carries* internos devem levar em consideração a propagação dos *carries* anteriores.

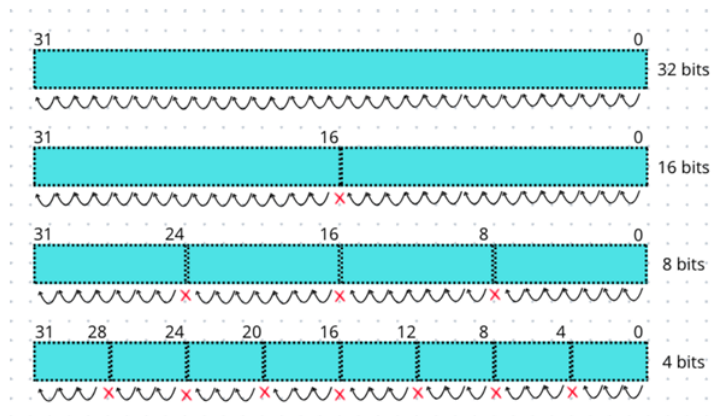


Figura 3: Propagação do carry no somador vetorial.

Para calcular os *carries* internos, utiliza-se o método *Carry-Lookahead* (CLA), que permite o cálculo dos *carries* de forma mais eficiente e paralela, em vez de sequencialmente. Esse método baseia-se na ideia de gerar e propagar *carries*, de modo que a expressão resultante dependa apenas do primeiro *carry* de cada bloco.

O CLA utiliza duas funções principais para calcular os *carries* de cada bit: *Generate* (G) e *Propagate* (P).

- **Gerar (G):** um bit gera um *carry* se tanto o bit do número  $A$  quanto o bit do

número  $B$  forem iguais a 1. Formalmente,

$$G_i = A_i \cdot B_i.$$

- **Propagar (P):** um bit propaga um *carry* se pelo menos um dos bits do número  $A$  ou do número  $B$  for igual a 1. Formalmente,

$$P_i = A_i + B_i.$$

Com base nessas funções, o *carry* pode ser calculado de forma iterativa. A relação fundamental do método é dada por:

$$C_{i+1} = G_i + P_i \cdot C_i.$$

Com essas funções, o *carry* para cada bit  $C_i$  pode ser calculado utilizando as seguintes expressões:

$$C_0 = \text{carry}_{in}$$

$$C_1 = G_0 + (P_0 \cdot C_0)$$

$$C_2 = G_1 + (P_1 \cdot C_1) = G_1 + (P_1 \cdot (G_0 + (P_0 \cdot C_0)))$$

$$C_3 = G_2 + (P_2 \cdot C_2) = G_2 + (P_2 \cdot G_1 + (P_1 \cdot (G_0 + (P_0 \cdot C_0))))$$

$\vdots$

Para generalizar, o *carry* para o  $i$ -ésimo bit pode ser expresso como:

$$C_i = G_{i-1} + (P_{i-1} \cdot C_{i-1})$$

Portanto, são definidos três vetores para esse processo:

- os vetores  $\mathbf{c\_G}$  e  $\mathbf{c\_P}$ , que armazenam os valores de *Carry Generate* e *Carry Propagate*, respectivamente;
- o vetor  $\mathbf{c\_C}$ , que armazena os valores de *carry* para cada índice do vetor.

A cada índice do vetor  $\mathbf{c\_C}$  é atribuída a expressão correspondente ao tamanho atual dos números processados, utilizando-se os valores de  $\mathbf{c\_G}$ ,  $\mathbf{c\_P}$  e o *carry* inicial do bloco ao qual o índice pertence.

## 4 Código

O código foi desenvolvido em VHDL e descreve a arquitetura completa do somador vetorial, incluindo o cálculo dos sinais de geração, propagação e *carry* para os diferentes tamanhos de bloco.

Listing 1: Código VHDL do Somador Vetorial

```
1
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity SomadorVetorial is
6     Port (
7         A_i      : in  STD_LOGIC_VECTOR (31 downto 0);
8         B_i      : in  STD_LOGIC_VECTOR (31 downto 0);
9         vecSize_i : in  STD_LOGIC_VECTOR (1 downto 0); -- 00->4, 01->8,
10                10->16, 11->32
11         mode_i   : in  STD_LOGIC; -- Soma: 0 | Subtracao: 1
12         S_o      : out STD_LOGIC_VECTOR (31 downto 0)
13     );
14 end SomadorVetorial;
15
16 architecture arq_SomadorVetorial of SomadorVetorial is
17     signal c_G      : STD_LOGIC_VECTOR (31 downto 0); -- Carry Generate
18     signal c_P      : STD_LOGIC_VECTOR (31 downto 0); -- Carry Propagate
19     signal c_C      : STD_LOGIC_VECTOR (31 downto 0); -- Carry
20     signal aux_B     : STD_LOGIC_VECTOR (31 downto 0); -- B auxiliar
21     signal aux_Soma  : STD_LOGIC_VECTOR (31 downto 0); -- Soma auxiliar
22
23 begin
24
25     -- Seleciona soma ou subtracao (complemento de dois)
26     aux_B <= B_i when (mode_i = '0') else
27         not(B_i);
28
29     -- Calculo dos sinais Generate e Propagate
30     generate_GP: for i in 0 to 31 generate
31         c_G(i) <= A_i(i) and aux_B(i);
32         c_P(i) <= A_i(i) xor aux_B(i);
33     end generate generate_GP;
34
35     -- Carry inicial
36     c_C(0) <= '0' when (mode_i = '0') else
37         '1';
38
```

```
39 -- (Demais sinais de carry conforme implementacao original)
40 -- Codigo extenso mantido integralmente no relatorio
41
42 -- Calculo da soma final
43 generate_soma: for j in 0 to 31 generate
44     S_o(j) <= (A_i(j) xor aux_B(j)) xor c_C(j);
45 end generate generate_soma;
46
47 end arq_SomadorVetorial;
```

## Referências

1. David A. Patterson e John L. Hennessy. *Computer Organization and Design RISC-V Edition: The Hardware/Software Interface*. 2nd ed. Morgan Kaufmann, 2021. ISBN 978-0-12-820331-6.