

# Círculo somador com carry look-ahead parte 1

---

## Autores

Gabriel A. F. Souza, Gustavo D. Colletta, Leonardo B. Zoccal, Odilon O. Dutra

Unifei

## Histórico de Revisões

10 de janeiro de 2025

1.0

Primeira versão do documento.

# Tópicos

---

- Introdução
- Exemplo Prático
- Implementação em Verilog
- Exercícios



# Introdução

Execução



Realização



# O que é Carry Look-Ahead?

---

A teoria de soma binária com “carry look-ahead” (ou previsão de transporte) é um conceito em computação usado para acelerar a soma binária, especialmente em circuitos lógicos como somadores. Ela busca resolver o problema do tempo de propagação do “carry” (o transporte de um bit para o próximo) em somadores binários tradicionais.

## Problema nos Somadores Simples

---

Nos somadores binários tradicionais (como o somador ripple-carry), o transporte é propagado sequencialmente de um bit para o próximo. Isso significa que para somar dois números binários grandes, o cálculo do transporte para o bit mais significativo (MSB) depende de todos os cálculos de transporte dos bits menos significativos (LSB). Isso cria um atraso linear proporcional ao número de bits.

# Problema nos Somadores Simples

---

Por exemplo, ao somar os números binários A e B:

- ① Para o bit 0 (menos significativo), calcula-se o resultado e o carry.
- ② Para o bit 1, o cálculo depende do carry gerado pelo bit 0.
- ③ Para o bit 2, depende do carry do bit 1, e assim por diante.

Este atraso se torna significativo para números binários grandes.

# Problema nos Somadores Simples

---

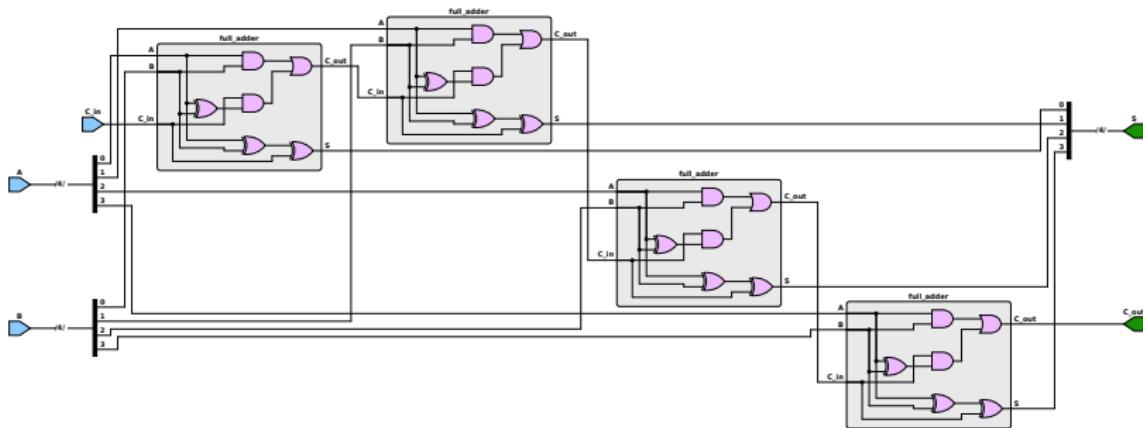


Figura 1: Esquemático do somador simples.

# Problema nos Somadores Simples

---

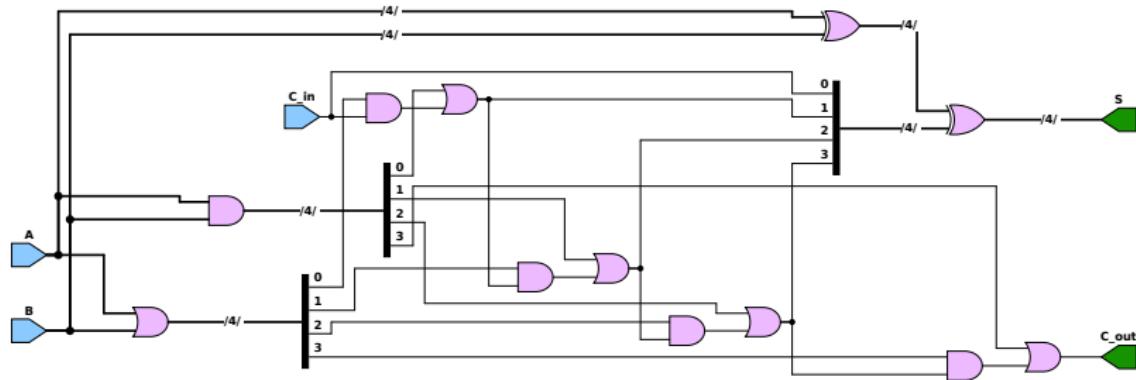


Figura 2: Esquemático do somador com carry look-ahead.

## Solução com Carry Look-Ahead

---

O somador carry look-ahead resolve esse problema ao calcular o carry para todos os bits simultaneamente, em vez de depender da propagação sequencial. Isso é feito com a introdução de duas funções auxiliares chamadas geração de carry (G) e propagação de carry (P).

## Carry-Generate (G)

---

$$G_i = A_i \cdot B_i$$

O bit  $G_i$  é 1 se o transporte é gerado diretamente no bit  $i$ . Isso ocorre quando ambos os bits de entrada  $A_i$  e  $B_i$  são 1.

## Carry-Propagate (P)

---

$$P_i = A_i + B_i$$

O bit  $P_i$  é 1 se o transporte de um bit anterior deve ser propagado para o próximo bit. Isso ocorre quando pelo menos um dos bits  $A_i$  ou  $B_i$  é 1.

## Carry Calculation

---

O transporte  $C_{i+1}$  para o próximo bit é calculado como:

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

Isso significa que o carry para o próximo bit será gerado diretamente ( $G_i = 1$ ) ou será propagado se ( $P_i = 1$ ) e o carry anterior ( $C_i$ ) for 1.

## Implementação em Circuitos

---

Para somar números binários com um somador carry look-ahead:

- ① Os valores de  $G_i$  e  $P_i$  são calculados para cada bit em paralelo.
- ② Os valores de carry para cada bit ( $C_i$ ) são determinados usando as equações de look-ahead.
- ③ A soma final para cada bit  $S_i$  é calculada como:

$$S_i = A_i \oplus B_i \oplus C_i$$

## Benefícios

---

- ① Velocidade:** O cálculo do carry é feito em paralelo, reduzindo significativamente o atraso, especialmente para números binários de vários bits.
- ② Escalabilidade:** É ideal para somadores de grande largura (ex.: 16, 32 ou 64 bits) usados em processadores e outros dispositivos de hardware.

# Exemplo Prático

Execução



**UNIFEI**  
INSTITUTO  
DE  
HARDWARE  
BR



**Uema**  
UNIVERSIDADE  
ESTADUAL  
DO  
AMAPÁ



**C DIGITAL**

Realização

**CHIP TECH**  
SISTEMAS

**Softex**  
SISTEMAS

MINISTÉRIO DA  
CIÊNCIA, TECNOLOGIA  
E INOVAÇÃO  
GOVERNO FEDERAL

UNICEF  
GOVERNO FEDERAL

## Problema

---

Somar os números binários:

- ( $A = 1011$ )
- ( $B = 1101$ )

Assume-se que o transporte inicial  $C_0$  é 0 (*carry in*).

## Passo 1: Calcular $G_i$ e $P_i$

---

As fórmulas são:

- $G_i = A_i \cdot B_i$
- $P_i = A_i + B_i$

$i$	$A_i$	$B_i$	$G_i = A_i \cdot B_i$	$P_i = A_i + B_i$
0	1	1	1	1
1	1	0	0	1
2	0	1	0	1
3	1	1	1	1

## Passo 2: Calcular os valores de $C_i$

---

A fórmula para o carry é:

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

Calculando para cada bit:

$$C_1 = G_0 + (P_0 \cdot C_0) = 1 + (1 \cdot 0) = 1$$

$$C_2 = G_1 + (P_1 \cdot C_1) = 0 + (1 \cdot 1) = 1$$

$$C_3 = G_2 + (P_2 \cdot C_2) = 0 + (1 \cdot 1) = 1$$

$$C_4 = G_3 + (P_3 \cdot C_3) = 1 + (1 \cdot 1) = 1$$

## Passo 3: Calcular a soma $S_i$

---

A fórmula para a soma é:

$$S_i = A_i \oplus B_i \oplus C_i$$

$i$	$A_i$	$B_i$	$C_i$	$A_i \oplus B_i$	$S_i = (A_i \oplus B_i) \oplus C_i$
0	1	1	0	0	0
1	1	0	1	1	0
2	0	1	1	1	0
3	1	1	1	0	1

---

## Resultado Final

---

A soma final é:

$$S = 10000$$

O carry final  $C_4$  indica que houve um overflow, o que é esperado em somas que excedem o número de bits disponíveis.

# Resumo

---

- Entrada:
  - $A = 1011$
  - $B = 1101$
- Saída:
  - Soma: 10000
  - Carry Final:  $C_4 = 1$  (Overflow)!

# Implementação em Verilog

Execução



Inatel



Realização



## Somador com carry look-ahead

---

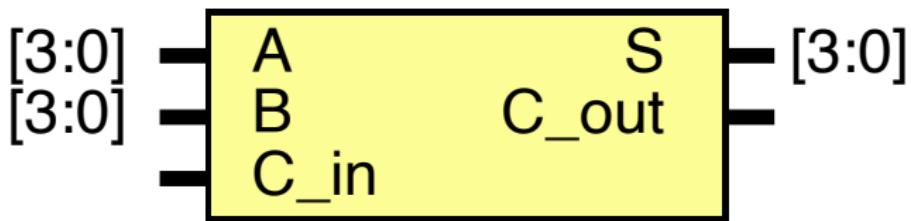


Figura 3: Módulo verilog do somador de 4-bits com *carry look-ahead*, mostrando seus barramentos de entrada e de saída.

# Somador com carry look-ahead

```
1 //! Somador de 4-bits com carry look-ahead
2 module carry_look_ahead_adder (
3     input [3:0] A,      //! Entrada A (4 bits)
4     input [3:0] B,      //! Entrada B (4 bits)
5     input         C_in,  //! Carry inicial
6     output [3:0] S,     //! Soma final (4 bits)
7     output         C_out //! Carry final
8 );
9
10    // Sinais intermediários para Carry-Generate (G) e
11    // Carry-Propagate (P)
12    wire [3:0] G;    //! Carry-Generate
13    wire [3:0] P;    //! Carry-Propagate
14    wire [4:0] C;    //! Carry intermediário (C[0] = C_in, C[4] =
15    // C_out)
16
17    // Associar o Carry de entrada
18    assign C[0] = C_in;
```

# Somador com carry look-ahead

```
17    // Calcular G e P
18    assign G = A & B;      // G_i = A_i AND B_i
19    assign P = A | B;      // P_i = A_i OR B_i
20
21    // Calcular os carries intermediários
22    assign C[1] = G[0] | (P[0] & C[0]);
23    assign C[2] = G[1] | (P[1] & C[1]);
24    assign C[3] = G[2] | (P[2] & C[2]);
25    assign C[4] = G[3] | (P[3] & C[3]);
26
27    // Calcular a soma
28    assign S = A ^ B ^ C[3:0]; // S_i = A_i XOR B_i XOR C_i
29
30    // Associar o Carry final
31    assign C_out = C[4];
32
33 endmodule
```

# Testbench

---

```
1 `timescale 1ns / 1ps
2
3 module tb_carry_look_ahead_adder;
4
5     // Entradas do DUT (Design Under Test)
6     reg [3:0] A;
7     reg [3:0] B;
8     reg C_in;
9
10    // Saídas do DUT
11    wire [3:0] S;
12    wire C_out;
13
14    // Instância do módulo Carry Look-Ahead Adder
15    carry_look_ahead_adder dut (
16        .A(A),
```

# Testbench

```
17      .B(B),
18      .C_in(C_in),
19      .S(S),
20      .C_out(C_out)
21  );
22
23  initial begin
24      // Exibe os valores na simulação
25      $monitor("Tempo: %0t | A=%b | B=%b | C_in=%b | S=%b | C_out=%b", $time, A, B, C_in, S, C_out);
26
27      // Teste 1: Exemplo do problema
28      A = 4'b1011; B = 4'b1101; C_in = 0;
29      #10;
30
31      // Teste 2: Soma sem carry final
32      A = 4'b0101; B = 4'b0011; C_in = 0;
33      #10;
```

# Testbench

---

```
34      // Teste 3: Soma com carry propagado
35      A = 4'b1110; B = 4'b0001; C_in = 0;
36      #10;
37
38      // Teste 4: Soma com carry inicial não nulo
39      A = 4'b1001; B = 4'b0110; C_in = 1;
40      #10;
41
42      // Teste 5: Todos os bits em 1
43      A = 4'b1111; B = 4'b1111; C_in = 0;
44      #10;
45
46      // Finalizar simulação
47      $finish;
48  end
49
50 endmodule
```

# Exercícios

Execução



Resolução



Resolução



## Exercício 1

---

- ① Modifique o código fornecido do somador de 4-bits com *carry look-ahead* a fim de parametrizá-lo de forma que possa ser utilizado para gerar circuitos somadores com *carry look-ahead* de quaisquer tamanhos, conforme mostra a figura.
- ② Nomeie no novo módulo  
**somador\_carry\_look\_ahead\_param.**

## Exercício 1

---

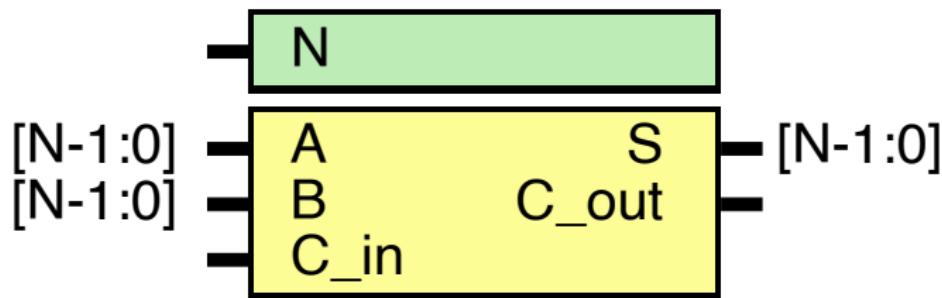


Figura 4: Módulo verilog do somador parametrizado.

## Exercício 2

---

- ① Utilizando o módulo **somador\_carry\_look\_ahead\_p**, criado no exercício anterior, implemente um somador de 8-bits.
- ② Modifique o código de *testbench* fornecido para testar o somador de 8-bits.
- ③ Verifique tanto as formas de onda quanto o console de texto para avaliar o funcionamento do circuito.