Introdução às Árvores de Soma - CSA

Autores	
Odilon de Oliveira Dutra	Unifei

Histórico de Revisões		
5 de fevereiro de 2025	1.0	Primeira versão do documento.

Tópicos

Introdução

Somadores

Antes de entendermos as Árvores de Soma e o CSA, é importante compreender como os somadores binários operam:

- **Ripple Carry Adder**: Somadores simples que propagam o carry sequencialmente.
- **Carry Look-Ahead Adder**: Usa lógica para reduzir a propagação de carry.
- **Carry-Save Adder (CSA)**: Focado em somas paralelas, eliminando a necessidade de propagação imediata de carry.

Árvore de Soma

Árvores de soma são estruturas eficientes para a soma de números binários, utilizadas principalmente em adição paralela. Elas são frequentemente usadas em circuitos como somadores de alta performance, como o **Carry-Save Adder** (CSA).

- Reduzem o número de propagação de carrys.
- Acelera a operação de soma.
- Cruciais em operações de multiplicação e outros algoritmos aritméticos.

Carry-Save Adder (CSA)

O **Carry-Save Adder** é um circuito usado para somar três números binários. Ele tem como principal vantagem evitar a propagação de carry durante o processo de soma.

- A principal vantagem do CSA é evitar a propagação de carry, o que acelera o processo de soma.
- Ele é usado principalmente em operações de multiplicação e outros algoritmos aritméticos.

Exemplo de código Verilog para CSA:

```verilog
// csa.v
    module csa(input [3:0] A, B, Cin, output [3:0] Sum, Cout);
        assign Sum = A ^ B ^ Cin;          // Soma
        assign Cout = (A & B) | (B & Cin) | (Cin & A);  // Carry
    endmodule
```

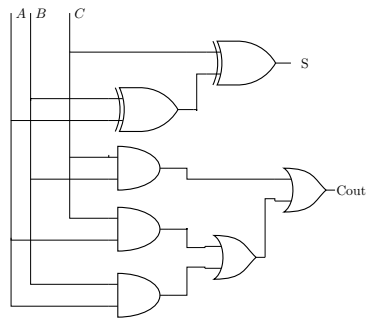


Figura: 1-bit CSA Circuit

Idéia do CSA

Não propague o Carry até o último estágio possível!

Funcionamento do CSA

Como funciona o Carry-Save Adder?

- O CSA recebe três números binários como entrada.
- Produz duas saídas:
  - **Sum**: Soma parcial.
  - **Cout**: Bit de carry a ser propagado.
- A soma final é realizada com um somador Ripple Carry ou outro circuito eficiente (este irá definir a ordem do algoritmo de soma final já que o CSA faz apenas a soma parcial).

Comparação com Outros Somadores

Ripple Carry Adder vs. **Carry-Save Adder**:

- Ripple Carry Adder:
  - Soma sequencial (lento).
  - Bom para somas pequenas.
  - Entrega o resultado final da soma.
- Carry-Save Adder:
  - Soma em paralelo (rápido).
  - Ideal para somas de múltiplos números.
  - Entrega apenas a soma parcial.

Diferença entre o CSA e um Full Adder comum

Full Adder Comum:
Recebe três bits de entrada: A, B, Cin
Produz duas saídas:

- Sum: A soma dos bits na mesma posição.
- Carry-out: O bit de carry que será propagado para o próximo bit mais significativo. Em um somador sequencial (como o Ripple-Carry Adder), o Carry-out propaga imediatamente para o próximo Full Adder.
- Já entrega o resultado final.

Carry-Save Adder (CSA): Também recebe três bits de entrada: A, B, Cin (onde Cin pode ser outra soma parcial ou carry). Produz duas saídas, mas com uma interpretação diferente:

- Sum: Soma parcial sem considerar o carry gerado na mesma posição.
- Carry: O carry gerado é "salvo" (ou armazenado) na mesma posição, não propagado imediatamente. O Carry e o Sum permanecem como duas palavras intermediárias e não são combinadas imediatamente. Isso significa que: O carry não propaga através de todas os bits, evitando o atraso de propagação típico de um Ripple-Carry Adder.
- Não entrega o resultado final já que o Carry ainda não foi propagado. O processo de soma demandará um Full Adder para a soma final.

Exemplo

Exemplo de funcionamento do CSA

Vamos somar 3 números binários: $A = 1011$, $B = 1101$, $C = 0110$.

1. **CSA - Soma parcial:**
   - Sum parcial (Sum): 0000
   - Cout ($Cout$): 1111
2. **Soma final:**

$$\text{Soma Final} = \text{Sum} + (\text{Cout} << 1)$$
$$\text{Soma Final} = 0000 + (1111 << 1)$$
$$\text{Soma Final} = 1110$$
$$\text{Carry} = 1$$

O CSA não propaga imediatamente o carry. Assim, não realiza a soma completa. A soma final é realizada apenas no último estágio. Esta pode se dar por Ripple Carry (maior atraso), ou somadores por árvores binárias a serem vistos nas próximas aulas (menor atraso).

Outro exemplo de funcionamento do CSA

Vamos somar 3 números binários: $A = 1111$, $B = 1111$, $C = 0111$.

1. **CSA - Soma parcial:**
   - Sum parcial (Sum): 0111
   - Cout ($Cout$): 1111
2. **Soma final:**

$$\text{Soma Final} = \text{Sum} + (\text{Cout} << 1)$$
$$\text{Soma Final} = 0111 + (1111 << 1)$$
$$\text{Soma Final} = 00101$$
$$\text{Carry} = 1$$

O CSA não propaga imediatamente o carry. Assim, não realiza a soma completa. A soma final é realizada apenas no último estágio. Esta pode se dar por Ripple Carry (maior atraso), ou somadores por árvores binárias a serem vistos nas próximas aulas (menor atraso).

Ordem da Propagação do Carry

Ordem do Carry-Save Adder (CSA)

- O CSA realiza somas parciais em paralelo, reduzindo o tempo necessário para acumular múltiplos valores binários $\rightarrow O(1)$.
- A soma, pode, então, ser dividida em duas etapas principais:
  1. **Etapa 1**: Soma parcial bit a bit $\rightarrow O(1)$ com CSA.
  2. **Etapa 2**: Soma final utilizando somadores paralelos $\rightarrow O(\log n)$ ou Ripple Carry $\rightarrow O(n)$.

Etapa 1: Somas Parciais com CSAs(Tempo Constante O(

- A soma parcial é realizada bit a bit, calculando:

$$\text{Sum} = A \oplus B \oplus C$$

$$\text{Cout} = (A \wedge B) \vee (A \wedge C) \vee (B \wedge C)$$

- Essa etapa ocorre de forma paralela para todos os bits.
- Por isso, o tempo desta etapa é O(1) (constante).

Etapa 2: Soma Final (Tempo Logarítmico O(log n))

- As saídas intermediárias (**Sum** e **Cout**) dos CSAs são somadas usando somadores paralelos como:
  - Brent-Kung Adder
  - Kogge-Stone Adder
- A propagação do carry é feita de forma paralela, resultando em:

$$\text{Ordem da Propagação do Carry na Soma Final} = O(\log n)$$

- Isso torna a etapa final muito mais rápida do que a soma linear em somadores tradicionais.

Complexidade Total da Soma

- A complexidade total da soma é:

$$O(1) + O(\log n) = O(\log n)$$

- O uso de somadores paralelos permite uma soma eficiente na segunda etapa.
- Essa abordagem é muito mais rápida que a soma sequencial no Ripple Carry Adder.

Comparação com o Ripple Carry Adder (RCA)

- No RCA:
  - A propagação de carry é sequencial.
  - A complexidade é O(n).
- No CSA com somador paralelo:
  - A soma parcial ocorre em tempo $O(1)$ nos CSAs.
  - A soma final ocorre em tempo $O(\log n)$ nos somadores de árvores binárias.
- Portanto, o CSA com somadores paralelos é muito mais eficiente para somas grandes.

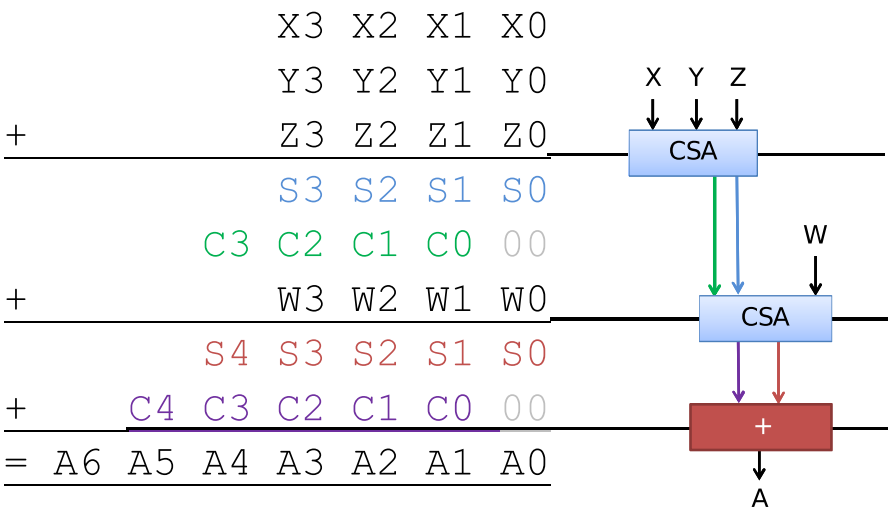
Aplicações do CSA

O CSA é amplamente utilizado em:

- **Multiplicadores**:
  - Reduzir o número de somas em matrizes parciais.
- **Processadores de Alto Desempenho**:
  - Operações aritméticas intensivas.
- **Unidades Aritméticas de GPUs e FPGAs**.

Aplicações do CSA - Multiplicador de 4-bits

Necessário somar 4 resultados parciais (X, Y, Z e W) oriundas da multiplicação.



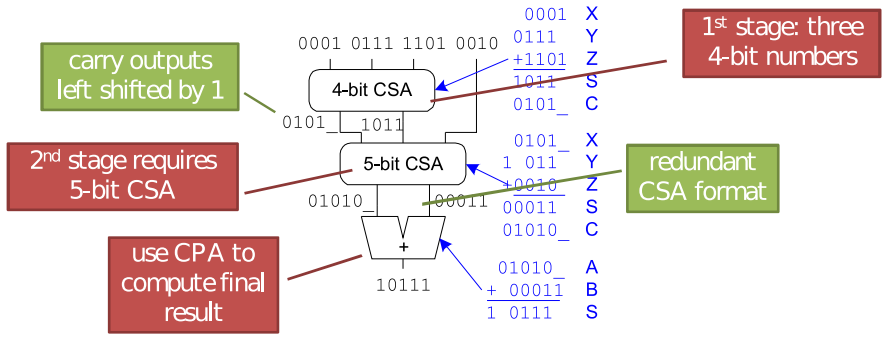


Figura: Soma de 4-operandos com CSAs

Figura: Soma de 4-operandos com CSAs

Cosiderando que X, Y, Z e W são os resultados parciais da multiplicação de cada um dos bits do multiplicador pelo multiplicando.

Hands-On: Projeto e Verificação do CSA

Hands-On

Atividades:

1. Projetar um **Carry-Save Adder** (CSA) em Verilog, simular seu funcionamento e verificar a corretude da operação para diferentes entradas binárias.
2. Projetar um CSA parametrizado e verificar seu funcionamento através de testbench.
3. Verificar a utilização de um CSAs na realização de somas dos produtos intermediários em um multiplicador de 4-bits

Ferramentas: Xcelium (Cadence) e SimVision.

Atividade 1: Descrição

Crie um módulo Verilog para o CSA com:

- **Entradas:** Três números binários de 4 bits (`A`, `B`, `Cin`).
- **Saídas:** Soma parcial (`Sum`) e Carry parcial (`Cout`).

Exemplo de código Verilog:

```
1 // csa.v
2     module csa(input [3:0] A, B, Cin, output [3:0] Sum, Cout);
3         assign Sum = A ^ B ^ Cin;          // Soma
4         assign Cout = (A & B) | (B & Cin) | (Cin & A);  // Carry
5     endmodule
```

Atividade 1-a: Testbench para Verificação

Implementar um testbench para fornecer combinações de `A`, `B`, `Cin` e validar `Sum` e `Carry`.

Verifique os resultados (utilize como referência os exercícios resolvidos em 13 e 14.)

Exemplo de Testbench:

```
1  module csa_tb;
2      reg [3:0] A, B, Cin;
3      wire [3:0] Sum, Cout;
4
5      csa uut (
6          .A(A),
7          .B(B),
8          .Cin(Cin),
9          .Sum(Sum),
10         .Cout(Cout)
11     );
12
13     initial begin
14         $display("A      B      Cin    | Sum    Carry");
15         $monitor("%b %b %b | %b %b", A, B, Cin, Sum, Cout);
16
17         // Test cases
18         A = 4'b1011; B = 4'b1101; Cin = 4'b0110; #10;
19         A = 4'b1111; B = 4'b1111; Cin = 4'b0111; #10;
20         A = 4'b0001; B = 4'b0010; Cin = 4'b0001; #10;
21         A = 4'b0101; B = 4'b1010; Cin = 4'b0110; #10;
22         A = 4'b1111; B = 4'b1111; Cin = 4'b1111; #10;
23
24         $finish;
25     end
26 endmodule
```

Atividade 2: Verilog CSA Parametrizado

```
1  // Modulo CSA parametrizavel
2  module parameterized_csa #(parameter WIDTH = 4) (
3      input  [WIDTH-1:0] A,
4      input  [WIDTH-1:0] B,
5      input  [WIDTH-1:0] Cin,
6      output [WIDTH-1:0] Sum,
7      output [WIDTH-1:0] Cout
8  );
9      genvar i;
10     generate
11         for (i = 0; i < WIDTH; i = i + 1) begin : CSA_BITS
12             assign Sum[i] = A[i] ^ B[i] ^ Cin[i];
13             assign Cout[i] = (A[i] & B[i]) | (B[i] & Cin[i]) | (A[i] & Cin[i]);
14         end
15     endgenerate
16 endmodule
```

- Este exemplo parametriza o CSA para o tamanho do operando.

Atividade 2-a: Testbench para CSA Parametrizado

```
1  module parameterized_csa_tb;
2      reg [3:0] A, B, Cin;
3      wire [3:0] Sum, Cout;
4
5      parameterized_csa uut (
6          .A(A),
7          .B(B),
8          .Cin(Cin),
9          .Sum(Sum),
10         .Cout(Cout)
11     );
12
13     initial begin
14         $display("A      B      Cin    | Sum    Carry");
15         $monitor("%b %b %b | %b %b", A, B, Cin, Sum, Cout);
16
17         // Test cases
18         A = 4'b1011; B = 4'b1101; Cin = 4'b0110; #10;
19         A = 4'b1111; B = 4'b1111; Cin = 4'b0111; #10;
20         A = 4'b0001; B = 4'b0010; Cin = 4'b0001; #10;
21         A = 4'b0101; B = 4'b1010; Cin = 4'b0110; #10;
22         A = 4'b1111; B = 4'b1111; Cin = 4'b1111; #10;
23
24         $finish;
25     end
26 endmodule
```

Atividade 3: Utilização de CSAs em Multiplicadores

- Integrar o CSA a um multiplicador.

Multiplicador com CSA - Parte 1

```
1  // Modulo Multiplicador usando CSA simplificado com apenas dois CSAs
2  module multiplier_csa #(parameter WIDTH = 4) (
3      input  [WIDTH-1:0] multiplicand,
4      input  [WIDTH-1:0] multiplier,
5      output [2*WIDTH-1:0] product
6  );
7      wire [2*WIDTH-1:0] partial_products [WIDTH-1:0]; // Produtos parciais estendidos
8      wire [2*WIDTH-1:0] sum_stage1, sum_stage2;      // Somadores parciais
9      wire [2*WIDTH-1:0] carry_stage1, carry_stage2;  // Carrys parciais
10
11     // Geracao dos produtos parciais
12     genvar i;
13     generate
14         for (i = 0; i < WIDTH; i = i + 1) begin : gen_partial_products
15             assign partial_products[i] = multiplier[i] ? (multiplicand << i) : 0;
16         end
17     endgenerate
18
19     // Primeira etapa: somar os primeiros dois produtos parciais
20     parameterized_csa #(2*WIDTH) csa_stage1 (
```

Observação: A soma final das multiplicações parciais estão sendo realizadas por Ripple-Carry Adder.

Multiplicador com CSA - Parte 2

```
        .A(partial_products[0]),
        .B(partial_products[1]),
        .Cin(partial_products[2]),
        .Sum(sum_stage1),
        .Cout(carry_stage1)
    );

    // Segunda etapa: somar os resultados intermediarios
    parameterized_csa #(2*WIDTH) csa_stage2 (
        .A(partial_products[3]),
        .B(sum_stage1),
        .Cin(carry_stage1 << 1),
        .Sum(sum_stage2),
        .Cout(carry_stage2)
    );

    // Produto final
    assign product = sum_stage2 + (carry_stage2 << 1); // Concatena soma e carry diretamente
endmodule
```

Observação: A soma final das multiplicações parciais estão sendo realizadas por Ripple-Carry Adder.

Multiplicador com CSA - TestBench Parte 1

```
1  // Test bench
2  module multiplier_csa_tb;
3  reg  [3:0] multiplicand;
4  reg  [3:0] multiplier;
5  wire [7:0] product;
6
7  // Instancia do multiplicador
8  multiplier_csa #(4) uut (
9      .multiplicand(multiplicand),
10     .multiplier(multiplier),
11     .product(product)
12 );
13
14 initial begin
15     // Testar combinacoes de entradas
16     multiplicand = 4'b0011; // 3
17     multiplier = 4'b0101;   // 5
18     #10;
19     $display("Multiplicand: %d, Multiplier: %d, Product: %d", multiplicand, multiplier,
           product);
```

Multiplicador com CSA - TestBench Parte 2

```
1     multiplicand = 4'b1111; // 15
2     multiplier = 4'b0001;   // 1
3     #10;
4     $display("Multiplicand: %d, Multiplier: %d, Product: %d", multiplicand, multiplier,
            product);
5
6     multiplicand = 4'b1011; // 11
7     multiplier = 4'b1111;   // 15
8     #10;
9     $display("Multiplicand: %d, Multiplier: %d, Product: %d", multiplicand, multiplier,
            product);
10
11    multiplicand = 4'b1111; // 15
12    multiplier = 4'b1111;   // 15
13    #10;
14    $display("Multiplicand: %d, Multiplier: %d, Product: %d", multiplicand, multiplier,
            product);
15
16    $finish;
17 end
18 endmodule
```

Multiplicador com CSA - TestBench Completo - Parte 1

Um circuito aritmético deve ter todas as possibilidades testadas. A seguir, é apresentado um testbench que faz isso para o multiplicador anterior.

```
1  // Test bench
2  module multiplier_testAll_tb;
3  reg  [3:0] multiplicand;
4  reg  [3:0] multiplier;
5  wire [7:0] product;
6  reg  [7:0] expected_product;
7  integer i, j;
8  reg success;
9
10 // Instancia do multiplicador
11 multiplier_csa #(4) uut (
12     .multiplicand(multiplicand),
13     .multiplier(multiplier),
14     .product(product)
15 );
16
17 initial begin
18     success = 1;
19     for (i = 0; i < 16; i = i + 1) begin
20         for (j = 0; j < 16; j = j + 1) begin
```

Verifique que, na janela console do Xview, será impresso Teste bem-sucedido para o caso da verificação de todos os produtos estiverem corretas.

Multiplicador com CSA - TestBench Completo - Parte 2

Um circuito aritmético deve ter todas as possibilidades testadas. A seguir, é apresentado um testbench que faz isso para o multiplicador anterior.

```
            multiplicand = i;
            multiplier = j;
            expected_product = i * j;
            #10;
            if (product !== expected_product) begin
                $display("Erro: multiplicand = %d, multiplier = %d, product = %d, expected =
                      %d", multiplicand, multiplier, product, expected_product);
                success = 0;
            end
        end
    end

    if (success) begin
        $display("Teste bem-sucedido");
    end else begin
        $display("Teste falhou");
    end

    $finish;
end
endmodule
```

Verifique que, na janela console do Xview, será impresso Teste bem-sucedido para o caso da verificação de todos os produtos estiverem corretas.

Conclusão

Conclusão do Algoritmo Carry Save Adder (CSA)

O **Carry Save Adder (CSA)** é uma técnica de adição eficiente, comumente utilizada em somadores de alta velocidade e sistemas de multiplicação. O principal benefício do CSA é a sua capacidade de realizar a adição de múltiplos números em paralelo sem propagar imediatamente o carry entre as somas, o que reduz significativamente o tempo de execução. No entanto, vale ressaltar que o **CSA não realiza a soma completa**, ele apenas gera dois resultados intermediários: o **Sum** e o **Carry**.
Esses dois resultados (Sum e Carry) precisam ser somados posteriormente, em um estágio adicional, usando outro somador, como o **Adder de propagação de Carry** (Carry Lookahead Adder ou outro circuito de soma). Isso significa que, embora o CSA seja eficiente para cálculos parciais, ele não resolve a soma final de forma completa.
Considerações Finais:

- O **CSA** é eficaz para somas parciais rápidas, mas não realiza a soma final diretamente.
- Ideal para implementações em múltiplos estágios, como na multiplicação de números grandes.
- A soma final requer um segundo estágio de adição para gerar o resultado completo.