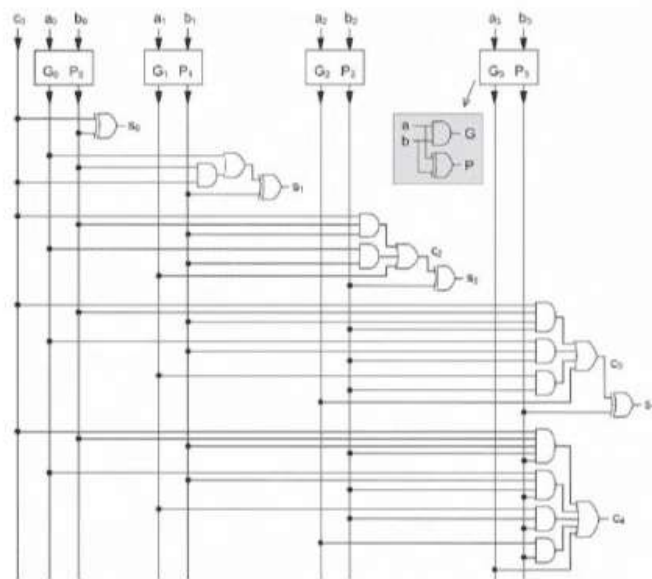


Somador Carry-Lookahead de 4 Bits em VHDL

Por **Gabriel Villanova** - 20/01/2016



ÍNDICE DE CONTEÚDO [MOSTRAR]

Esse somador utiliza uma técnica para aceleração do cálculo do *carry*. Em cada estágio ele calcula seu próprio *bit-carry* de entrada a partir das entradas do primeiro estágio, ou seja, ele não espera a propagação das informações dos estágios anteriores.

Por contra, ele também possui desvantagens, como, por exemplo, o crescimento quase exponencial do número de portas utilizadas a cada estágio, encarecendo e tornando o transporte do *carry* mais lento para um projeto de mais de quatro estágios.

Devido a essas circunstâncias esse somador é geralmente projetado para somar palavras de 4 *bits*. Dessa forma, podemos reutilizar o circuito para somar palavras binárias de 8, 12, 16, 32, etc.

Neste artigo vamos estudar esse tipo de somador rápido, projetá-lo para somar palavras de 4 *bits* e descrevê-lo em VHDL.

Somador Carry-Lookahead

Existem três casos de propagação do *carry*: **Kill**, **Generate** e **Propagate**.

O caso **Kill** significa que quando as entradas A_i e B_i do somador são iguais a zero, independente do seu *carry-in*, a propagação do **carry-out** não existe (“matado”). Isso quer dizer também que o *carry-in* do próximo estágio vale zero.

O caso **Generate** significa que quando as entradas A_i e B_i do somador são iguais a um, independente do seu *carry-in*, a propagação do **carry-out** existe (“gerado”). Isso quer dizer também que o *carry-in* do próximo estágio vale um.

O caso **Propagate** significa que quando as entradas A_i e B_i do somador são iguais a um e zero ou a zero e um respectivamente. Dessa forma o seu **carry-in** será propagado para o **carry-out**.

Com essas informações é possível montar uma tabela verdade e assim encontrar uma expressão booleana para o *carry-out*, ou seja, C_{i+1} .

* i é um número natural.

Tabela 1: Tabela verdade para o *carry-out*.

	A_i	B_i	C_{i+1}
K_i	0	0	0
P_i	0	1	C_i
P_i	1	0	C_i
G_i	1	1	1

$$C_{i+1} = A_i \cdot B_i + C_i \cdot (\bar{A}_i \cdot B_i) + C_i \cdot (A_i \cdot \bar{B}_i)$$

$$C_{i+1} = G_i + (\bar{A}_i \cdot B_i + A_i \cdot \bar{B}_i) \cdot C_i$$

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

Para obter a expressão da saída podemos fazer o mesmo procedimento precedente considerando as entradas A_i , B_i e C_i .

Tabela 2: Tabela verdade para a saída.

C_i	B_i	C_i	S_i
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

$$S_i = \bar{C}_i \cdot \bar{A}_i \cdot B_i + \bar{C}_i \cdot A_i \cdot \bar{B}_i + C_i \cdot \bar{A}_i \cdot \bar{B}_i + C_i \cdot A_i \cdot B_i$$

$$S_i = \bar{C}_i \cdot (\bar{A}_i \cdot B_i + A_i \cdot \bar{B}_i) + C_i \cdot (\bar{A}_i \cdot \bar{B}_i + A_i \cdot B_i)$$

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

Tendo em vista que os sinais **Generate** e **Propagate** são sempre em função das entradas A_i e B_i , e se comportam como Meio-Somadores (*Half-Adders*), modelaremos o projeto utilizando-os.

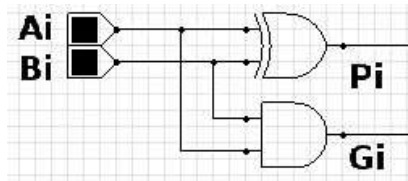


Figura 1: Esquema do Meio-Somador com as saídas Propagate e Generate.

Como nosso objetivo é projetar este somador para palavras de 4 *bits*, acharemos as equações para cada estágio.

Para o estágio zero, nós temos que o *carry-in* é uma **entrada do circuito**, portanto obtemos:

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

Para o estágio um, temos que o *carry* calculado no estágio zero será o *carry-in* desse estágio, isto é:

$$\begin{aligned} C_2 &= G_1 + P_1 \cdot C_1 \\ C_2 &= G_1 + P_1 \cdot (G_0 + P_0 \cdot C_0) \\ C_2 &= G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0 \end{aligned}$$

Seguindo a lógica temos:

$$\begin{aligned} C_3 &= G_2 + P_2 \cdot C_2 \\ C_3 &= G_2 + P_2 \cdot (G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0) \\ C_3 &= G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0 \end{aligned}$$

$$C_4 = G_3 + P_3 \cdot C_3$$

$$C_4 = G_3 + P_3 \cdot (G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0)$$

$$C_4 = G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0$$

Se olharmos para a próxima equação podemos observar que para o próximo estágio (quinto) o número de portas é muito grande, tornando inviável o custo do projeto, além de não se tornar mais rápido em relação a um somador *Ripple-Carry* por exemplo.

$$C_5 = G_4 + P_4 \cdot C_4$$

$$C_5 = G_4 + P_4 \cdot (G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0)$$

$$C_5 = G_4 + P_4 \cdot G_3 + P_4 \cdot P_3 \cdot G_2 + P_4 \cdot P_3 \cdot P_2 \cdot G_1 + P_4 \cdot P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_4 \cdot P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0$$

Desempenho do *Carry-Lookahead* 4 bits em relação ao *Ripple-Carry*

Comparando a propagação dos *carries-out* dos circuitos *Ripple-Carry* e *Carry-Lookahead* podemos observar o percurso do *carry-out* do primeiro somador bem mais longo.

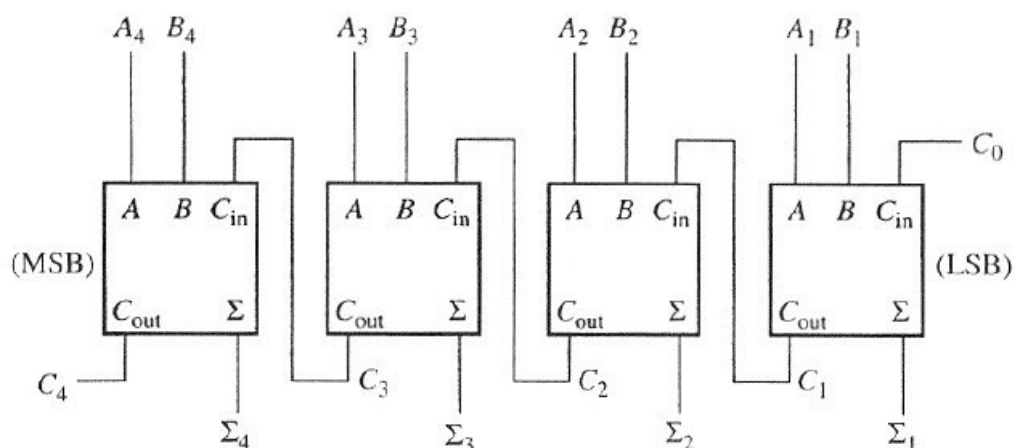


Figura 2: Somador *Ripple-Carry* 4 bits (Fonte: Eletrônica Digital por F.C.C de Castro PUCRS)

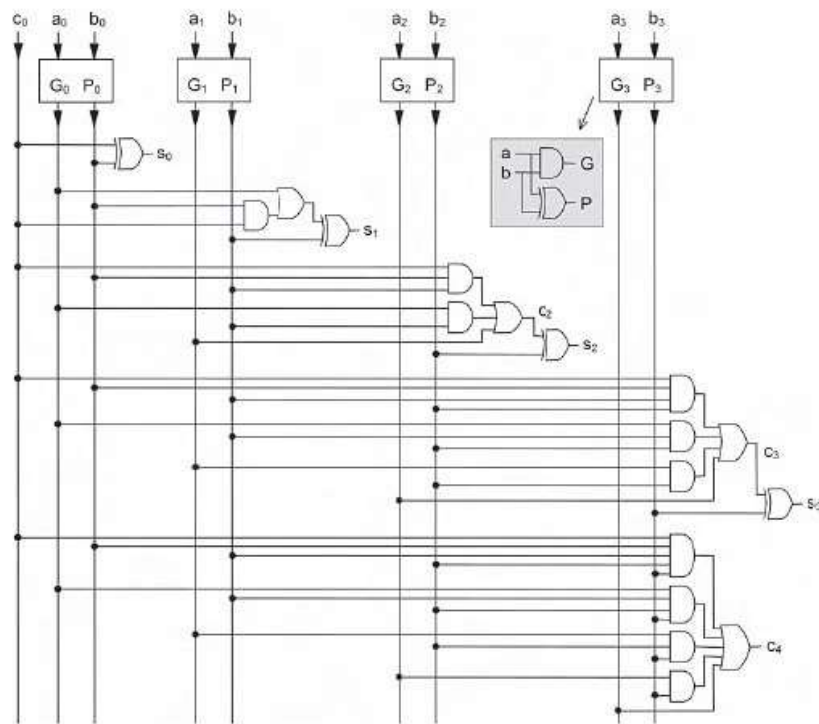


Figura 3: Somador Carry-Lookahead 4 bits (Fonte: Digital Eletronics Pedroni)

Como no *Ripple-Carry* é utilizado blocos de somadores completos, temos que cada estágio tem um tempo crítico agregado a 3 portas lógicas. Ou seja, para o circuito completo 12 portas lógicas até o *carry-out* está bem definido. Diferente do nosso somador estudado que tem tempo crítico agregado a 4 portas lógicas.

Implementação VHDL

Utilizando o software **Altera Quartus II 13.0 Web Edition**, criou-se o componente HALF_ADDER para ser utilizado no módulo-topo do nosso circuito principal. Esse componente foi descrito como segue:

```

1 library IEEE;
2 use IEEE.std_logic_1164 .all;
3
4 entity HALF_ADDER is
5 port(A, B : in std_logic;
6       S : out std_logic;
7       Cout : out std_logic);
8 end entity;
9

```

```

10 architecture RTL of HALF_ADDER is
11 begin
12     process(A,B)
13     begin
14         S <= A xor B;
15         Cout <= A and B;
16     end process;
17 end RTL;

```

Para o modulo-topo utilizamos o componente HALF_ADDER e escrevemos as equações necessárias para a implementação do somador *Carry-Lookahead 4 bits*. Onde a descrição utilizada foi:

```

1  library IEEE;
2
3  use      IEEE.std_logic_1164 .all;
4
5  entity CARRYLOOK is
6
7      port(A, B : in std_logic_vector (3 downto 0);
8            Cin : in std_logic;
9            Cout: out std_logic;
10           S : out std_logic_vector (3 downto 0));
11
12  end entity;
13
14  architecture RTL of CARRYLOOK is
15
16  signal sig_P : std_logic_vector (3 downto 0);
17  signal sig_G : std_logic_vector (3 downto 0);
18  signal sig_S : std_logic_vector (3 downto 0);
19
20  signal C1, C2, C3 : std_logic;
21
22  component HALF_ADDER is
23
24      port(A, B : in std_logic;
25            S : out std_logic;
26            Cout : out std_logic);
27  end component;
28
29
30  begin
31
32
33
34  H0: HALF_ADDER port map (A(0),B(0),sig_P(0),sig_G(0));
35  H1: HALF_ADDER port map (A(1),B(1),sig_P(1),sig_G(1));
36  H2: HALF_ADDER port map (A(2),B(2),sig_P(2),sig_G(2));
37  H3: HALF_ADDER port map (A(3),B(3),sig_P(3),sig_G(3));
38
39
40
41  C1  <= sig_G(0) or (sig_P(0) and Cin);
42
43  C2  <= sig_G(1) or (sig_P(1) and sig_G(0)) or
44      (sig_P(1) and sig_P(0) and Cin);
45
46  C3  <= sig_G(2) or (sig_P(2) and sig_G(1)) or
47      (sig_P(2) and sig_P(1) and sig_G(0)) or
48      (sig_P(2) and sig_P(1) and sig_P(0) and Cin);
49
50  Cout <= sig_G(3) or (sig_P(3) and sig_G(2)) or
51      (sig_P(3) and sig_P(2) and sig_G(1)) or
52      (sig_P(3) and sig_P(2) and sig_P(1) and
53      sig_G(0)) or (sig_P(3) and sig_P(2) and
54      sig_P(1) and sig_P(0) and Cin);
55
56  S(0) <= sig_P(0) xor Cin;
57
58  S(1) <= sig_P(1) xor C1;
59
60  S(2) <= sig_P(2) xor C2;
61
62  S(3) <= sig_P(3) xor C3;
63
64  end RTL;

```

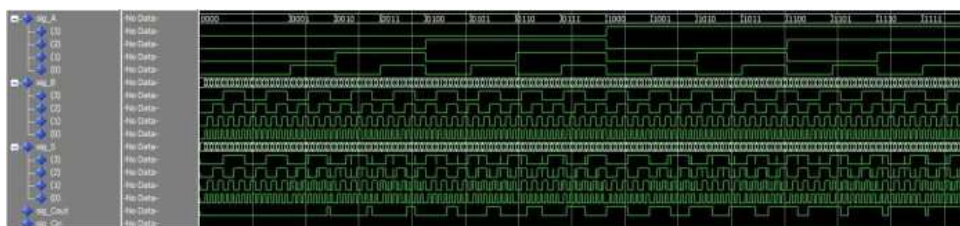
Para verificação do funcionamento foi feito o testbench, abaixo o código e as formas de onda do nosso circuito!

```

1  library IEEE;
2
3  use IEEE.std_logic_1164 .all;
4
5  use IEEE.std_logic_unsigned .all;
6
7
8
9  entity TB_CARRYLOOK is end TB_CARRYLOOK ;
10
11
12
13  architecture COMPORTAMENTAL of TB_CARRYLOOK is
14
15      component CARRYLOOK is
16
17          port(A, B : in std_logic_vector (3 downto 0);
18              Cin : in std_logic;
19              Cout : out std_logic;
20              S : out std_logic_vector (3 downto 0));
21
22      end component;
23
24
25
26
27  signal sig_A : std_logic_vector (3 downto 0) := "0000";
28  signal sig_B : std_logic_vector (3 downto 0) := "0000";
29  signal sig_Cin : std_logic := '0';
30  signal sig_Cout : std_logic;
31  signal sig_S : std_logic_vector (3 downto 0);
32
33  signal sig_i : std_logic_vector (3 downto 0) := "0000";
34  signal sig_j : std_logic_vector (3 downto 0) := "0000";
35
36
37
38  begin
39
40
41
42  U1: CARRYLOOK port map (sig_A,sig_B,sig_Cin,sig_Cout,sig_S);
43
44
45
46  process
47  variable i : integer range 0 to 16;
48  variable j : integer range 0 to 16;
49
50  constant delay : time := 10 ns;
51
52  begin
53
54      for i in 0 to 16 loop
55          sig_A <= sig_i;
56          for j in 0 to 16 loop
57              sig_B <= sig_j;
58              wait for delay;
59
60              assert sig_A+sig_B = sig_S report "SUM CORRECT" severity error;
61              sig_j <= sig_j + '1';
62              if( j = 16 ) then
63                  sig_i <= sig_i + '1';
64              end if;
65          end loop;
66      end loop;
67
68  end process;
69
70  end COMPORTAMENTAL ;

```


Veja a forma de onda resultante dos testes.



Para fazer o download de todos os arquivos do projeto para o Quartus, clique no botão abaixo.



f Facebook 26

Twitter 4

G+ Google+ 0

in LinkedIn 24

NEWSLETTER


Receba os melhores conteúdos sobre sistemas eletrônicos embarcados, dicas, tutoriais e promoções.

E-mail

CADASTRAR E-MAIL

Fique tranquilo, também não gostamos de spam.



Somador Carry-Lookahead de 4 Bits em VHDL por *Gabriel Villanova*. Esta obra está licenciado com uma Licença [Creative Commons Atribuição-CompartilhaIgual 4.0 Internacional](https://creativecommons.org/licenses/by-sa/4.0/) .

Gabriel Villanova

Estudante de Engenharia Elétrica pela Universidade Federal de Campina Grande (UFCG). Participou de intercâmbio (2015/2016) no Instituto Grenoble - INP (Institut National Polytechnique) na filière SEI (Systèmes Électroniques Intégrés) onde realizou alguns projetos de hardware envolvendo FPGA e ASIC. Fez estágio de 3 meses com P&D no laboratório TIMC-IMAG (Techniques de l'Ingénierie Médicale et de la Complexité - Informatique, Mathématiques et Applications, Grenoble) com desenvolvimento de produto médico. Atualmente, trabalha no laboratório Embedded (UFCG) no projeto Idea com desenvolvimento de chips ópticos. Tem grande interesse em Sistemas em Tempo Real, Linux Embarcado, IoT, Microcontroladores e FPGA.

Este site utiliza cookies. Ao usá-lo você concorda com nossos Termos de Uso.
[Saiba mais.](#)

Continuar