Somador/Subtrator de 8 Bits em VHDL

Introdução

Este documento descreve o desenvolvimento de um **somador/subtrator** de 8 bits utilizando VHDL. Os operandos A e B são representados por vetores de 8 bits (`std_logic_vector`).

Especificações

- **Operação de subtração:** O subtrator utilizará complemento de dois.
- **Tamanho dos operandos:** A e B terão 8 bits.
- **Descrição estrutural:** O projeto utilizará uma descrição estrutural com módulos reusáveis, como um somador completo de 1 bit.

  Para esse somador, temos:

  A soma é calculada como o XOR das três entradas A, B e C_{in}:

$$S = A \oplus B \oplus C_{\text{in}}$$

  O *carry out* é gerado quando pelo menos duas das três entradas são 1:

$$C_{\text{out}} = (A \cdot B) + (A \cdot C_{\text{in}}) + (B \cdot C_{\text{in}})$$

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity full_adder is
5     Port (
6         A : in STD_LOGIC;
7         B : in STD_LOGIC;
8         Cin : in STD_LOGIC;
```

```
        Sum : out STD_LOGIC;
        Cout : out STD_LOGIC
    );
end full_adder;

architecture Behavioral of full_adder is
begin
    Sum <= A xor B xor Cin;
    Cout <= (A and B) or (B and Cin) or (A and Cin);
end Behavioral;
```

Entidade do Somador/Subtrator

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity add_sub_8bit is
    Port (
        A : in STD_LOGIC_VECTOR(7 downto 0);
        B : in STD_LOGIC_VECTOR(7 downto 0);
        Sub : in STD_LOGIC; -- 0 para soma, 1 para
           subtracao
        Sum : out STD_LOGIC_VECTOR(7 downto 0);
        Cout : out STD_LOGIC -- Carry out
    );
end add_sub_8bit;
```

Arquitetura

```
architecture Structural of add_sub_8bit is

    signal B_inverted : STD_LOGIC_VECTOR(7 downto 0);
    signal carry : STD_LOGIC_VECTOR(7 downto 0);

begin
    -- Inverte B se Sub = '1'
    B_inverted <= B xor (Sub & Sub & Sub & Sub & Sub &
       Sub & Sub & Sub);

```

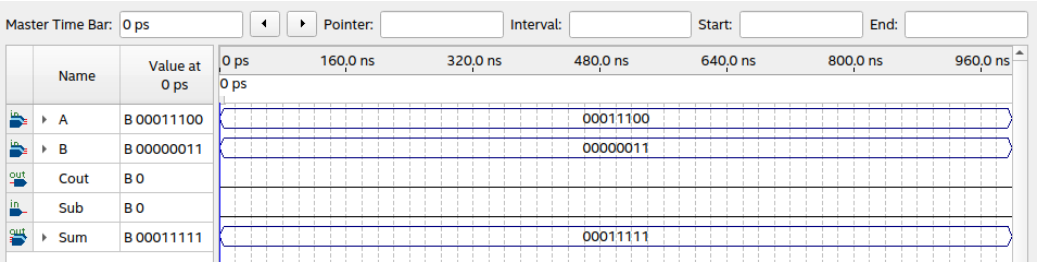
```
10      -- Inst ncias dos somadores completos
11      FA0: entity work.full_adder port map (
12          A => A(0),
13          B => B_inverted(0),
14          Cin => Sub,
15          Sum => Sum(0),
16          Cout => carry(0)
17      );
18
19      FA1: entity work.full_adder port map (
20          A => A(1),
21          B => B_inverted(1),
22          Cin => carry(0),
23          Sum => Sum(1),
24          Cout => carry(1)
25      );
26
27      FA2: entity work.full_adder port map (
28          A => A(2),
29          B => B_inverted(2),
30          Cin => carry(1),
31          Sum => Sum(2),
32          Cout => carry(2)
33      );
34
35      FA3: entity work.full_adder port map (
36          A => A(3),
37          B => B_inverted(3),
38          Cin => carry(2),
39          Sum => Sum(3),
40          Cout => carry(3)
41      );
42
43      FA4: entity work.full_adder port map (
44          A => A(4),
45          B => B_inverted(4),
46          Cin => carry(3),
47          Sum => Sum(4),
48          Cout => carry(4)
49      );
50
51      FA5: entity work.full_adder port map (
52          A => A(5),
```

```
53          B => B_inverted(5),
54          Cin => carry(4),
55          Sum => Sum(5),
56          Cout => carry(5)
57      );
58
59      FA6: entity work.full_adder port map (
60          A => A(6),
61          B => B_inverted(6),
62          Cin => carry(5),
63          Sum => Sum(6),
64          Cout => carry(6)
65      );
66
67      FA7: entity work.full_adder port map (
68          A => A(7),
69          B => B_inverted(7),
70          Cin => carry(6),
71          Sum => Sum(7),
72          Cout => Cout
73      );
74
75  end Structural;
```

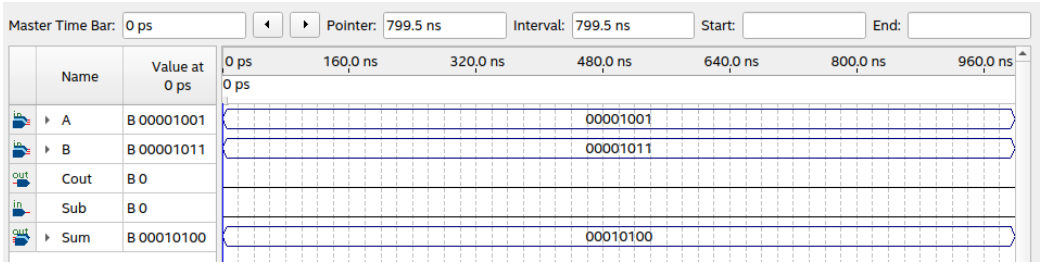
Plano de Simulação e Testes

Para validar o funcionamento do sistema, foram realizados testes considerando as seguintes situações:

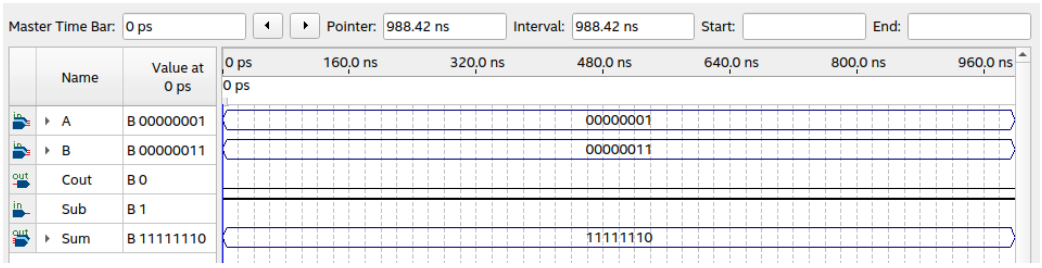
0.1 Caso 1: Soma de dois números positivos



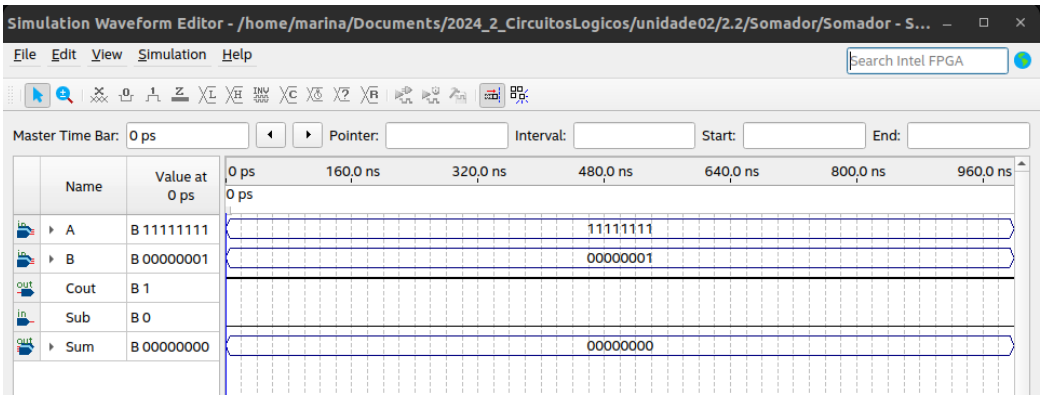
0.2 Caso 2: Soma com *carry*



0.3 Caso 3: Subtração de um número menor por um maior



0.4 Caso 4: Operações com *overflow*



Conclusão

Com os resultados dos testes, podemos ver que o sistema funciona conforme o esperado.