Somador Binário de 12 Bits

1.0 - PROBLEMA:

Criar um circuito combinacional capaz de somar ou subtrair uma palavra A de 12 bits {a11, a10, a9, a8, a7, a6, a5, a4, a3, a2, a1, a0} com outra palavra B {b11, b10, b9, b8, b7, b6, b5, b4, b3, b2, b1, b0} de 12 bits no formato complemento de 2. Para somas ou subtrações não suportadas, será ativado o sinal de overflow, indicando uma operação inválida.

Componentes:

- Negador;
- Mux 2x12;
- Somador 12 bits;
- Saída overflow;
- No VHDL, haverá diversos sinais para conectar os diversos elementos do circuito.

2.0 - CIRCUITO COMBINACIONAL:

2.1 - NEGADOR

Como no formato c2, para efetuar a subtração de A com B é necessário negar B e somar 1 ao mesmo, e depois somar A, ((A – B) = (A+(-B+1)), esse componente irá fazer o processo de inverter os valores do número.

1) Comportamento:

Entradas: Uma palavra de 12 bits {b11, b10, b9, b8, b7, b6, b5, b4, b3, b2, b1, b0}, escrita em Complemento de 2. O bit mais significativo (b11) equivale ao sinal do número (1 = - e 0 = +).

Saídas: a negação dessa palavra de 12 bits {~b11, ~b10, ~b9, ~b8, ~b7, ~b6, ~b5, ~b4, ~b3, ~b2, ~b1, ~b0}, onde todos os bits de número 1 saem como 0 e todos os bits de número de 1.

Exemplo: Entrada: {010110110110}.
Saída: {101001001001}.

2) Tabela-Verdade:

Nota: cada negação será feita em um bit de cada vez, portanto a tabela verdade foi simplificada para cada bit.

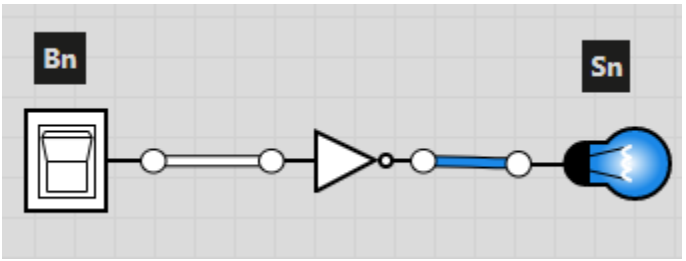
situação	entrada	saida
0	0	1
1	1	0

3) Diagrama Veitch-Karnaugh

B	~B
0	1

4) Expressão Booleana e circuito lógico

- Sn <= ~Bn



2.3 - Multiplexador (MUX 2x12):

O somador terá a opção para somar ou subtrair, para o usuário fazer essa escolha, será utilizado um multiplexador.

1) Comportamento

Após passar pelo negador, o sinal da palavra ~B (12 bits) e B (12 bits, será transmitido para as entradas do mux (cada bit encaminhado para uma entrada), o sinal seletor OP (1 bit) será usado para definir qual sinal de cada bit será encaminhado a saída.

Entradas:

- B {bn};
- ~B {~bn};
- OP.

Saída:

- caso OP, S <= ~B
- caso ~OP, S <= B

2) Tabela verdade

Nota: como cada bit será encaminhado para um mux 2x1 que será replicado 12 vezes, a tabela verdade foi simplificada e contém apenas 2 bits

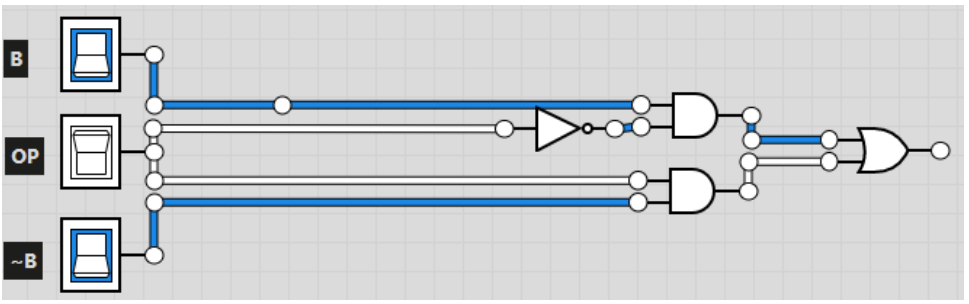
B	~B	OP	S
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0

3) Diagrama Veitch-Karnaugh

	~OP	OP
~B	0	1
B	1	0

4) Expressão Booleana e Circuito lógico

S <= (B . ~OP) + (~B.OP)



2.4 - Somador:

Para efetuar a soma ou subtração, o somador irá receber as duas palavras e realizará a operação, caso seja selecionada a subtração, o sinal OP, que é utilizado no multiplexador como seletor, também será utilizado no somador como carry-in para a soma do bit ~b11, assim completando a inversão da palavra (-B+1), e economizando recursos.

1) Comportamento

O somador efetuará a operação (soma ou subtração) de um bit da palavra A com outro bit da palavra B e o bit de carry in (bit OP no caso do somador dos bits a11 e b11 (ou ~b11))

2) Tabela verdade

A	B	Cin	Soma	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

3) Diagrama Veitch Karnaugh

- Diagrama para a saída soma:

	~Cin	Cin
~A.~B	0	1
~A.B	1	0
A.B	0	1
A.~B	1	0

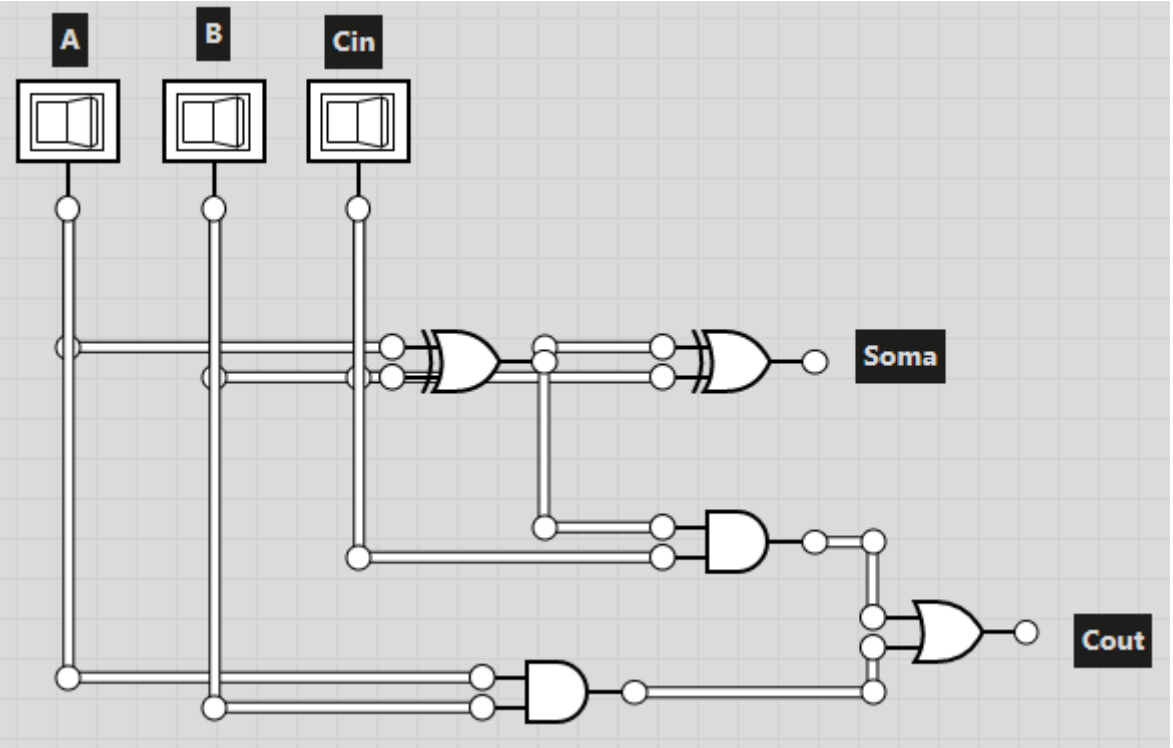
- Diagrama para a saída cout:

	~Cin	Cin
~A.~B	0	0
~A.B	0	1
A.B	1	1
A.~B	0	1

4) expressões booleanas e circuito lógico

Soma <= (~A.~B.Cin) + (~A.B.~Cin) + (A.B.Cin) + (A.~B.~Cin)
~A.(~B.Cin) + (B.~Cin) + A.(~B.~Cin + B.Cin)
~A.(B ⊕ Cin) + A.(~(B ⊕ Cin))
A ⊕ B ⊕ C

Cout <= (A.B) + (A.Cin) + (B.Cin)
Cout <= A.(B+Cin) + (B.Cin)



2.4 - Saída Overflow:

Caso durante as operações ocorra um overflow, isso irá distorcer o resultado da operação e o usuário será notificado disso.

1) comportamento

Levamos em conta a regra do overflow, onde um overflow só ocorre quando uma operação entre dois números de sinais iguais resulta em um número de sinal diferente.

- Exemplo:

A,B e C $\in \mathbb{R} > 0$
-A,-B e -C $\in \mathbb{R} < 0$

(A + B = C) - não há overflow
(A + B = -C) - há overflow
(-A + (-B) = -C) - não há overflow
(-A + (-B) = C) - há overflow

Como no formato c2, o bit mais significativo é o que determina o sinal de um número, podemos usar esses sinais e o sinal da soma para determinar a existência ou não de um overflow

2) tabela verdade

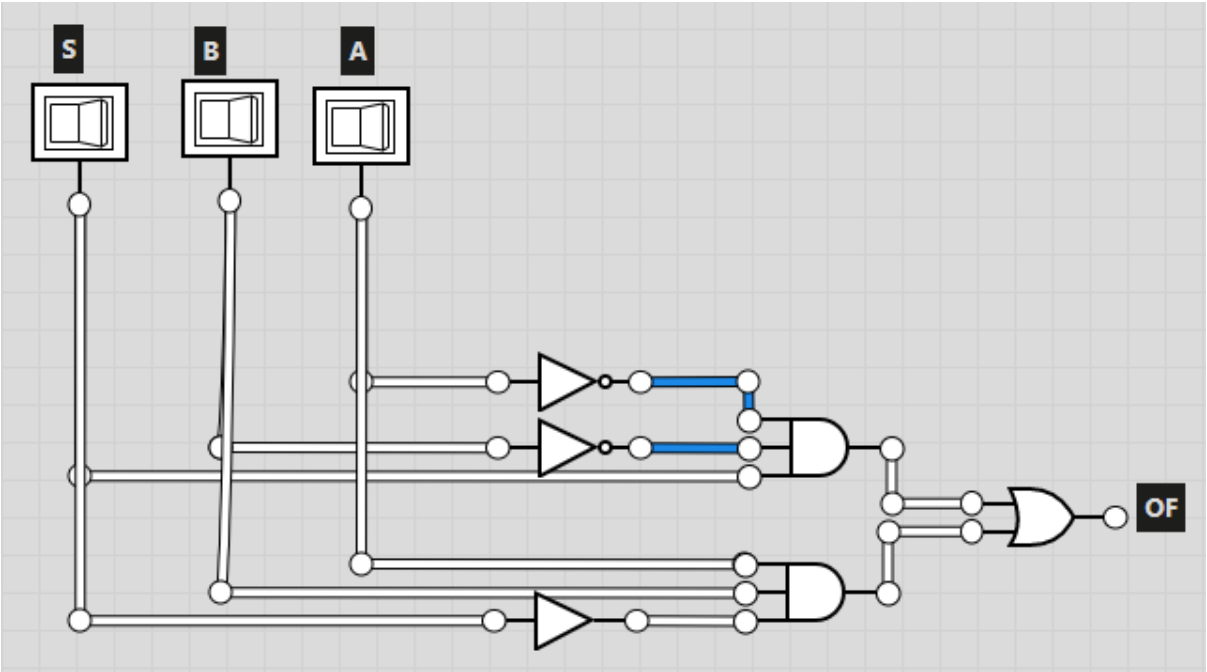
A	B	S	OF
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

3) Diagrama Veitch-Karnaugh

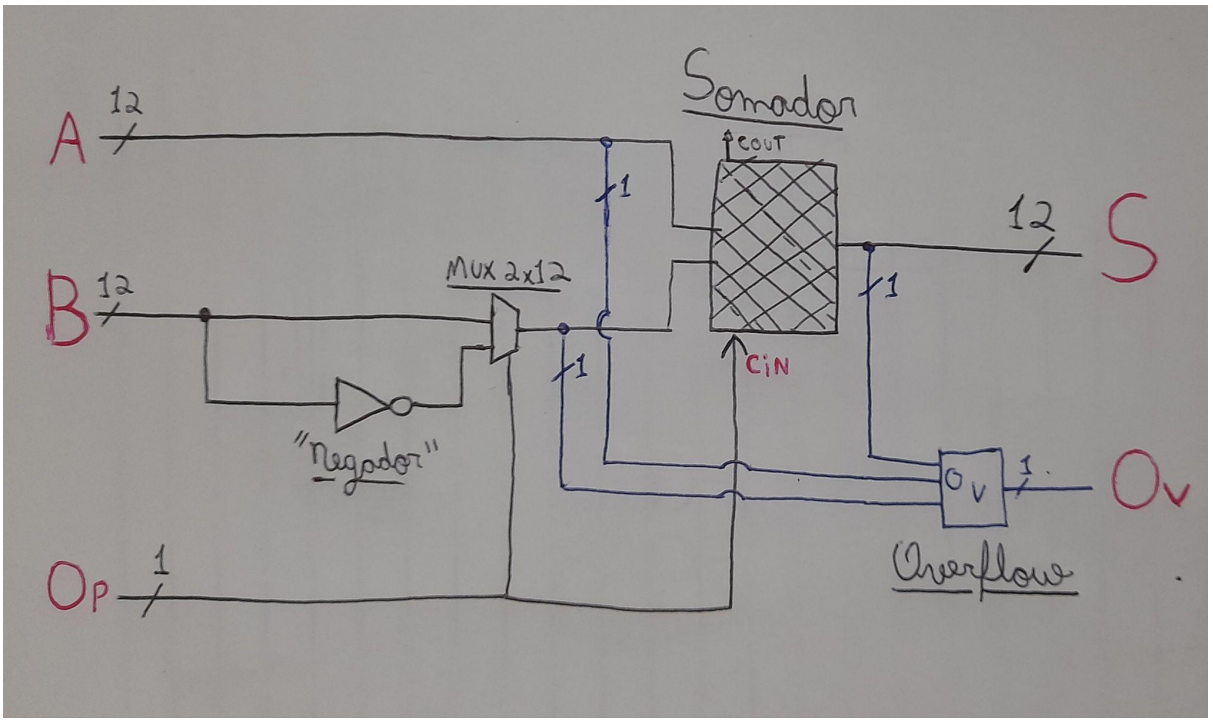
	~S	S
~A.~B	0	1
~A.B	0	0
A.B	1	0
A.~B	0	0

4) Expressão booleana e circuito lógico

- OF <= (~A.~B.S) +(A.B.~S)



2.5 - Circuito Final:



3.0 - VHDL:

O código em VHDL foi feito pelo sistema operacional Windows 10, escrito utilizando o software Visual Studio Code, com as extensões “TerosHDL” e “VHDL”, e usando a biblioteca “*ieee*”. Os códigos em foram convertidos num arquivo .ghw com os mesmos comandos que se usariam no terminal do Linux.

1) Mux2x12:

O multiplexador foi feito de acordo com o apresentado nas aulas práticas sobre, com a exceção de que as entradas foram substituídas por variáveis do tipo ‘*std_logic_vector(11 downto 0)*’, pertencentes à biblioteca “*ieee*”. A seguir, uma print do código do multiplexador:

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity mux2x12 is
5          port(
6              Bsim : in  std_logic_vector (11 downto 0);
7              Bnao : in  std_logic_vector (11 downto 0);
8              Op   : in  std_logic;
9              Bout : out std_logic_vector (11 downto 0)
10         );
11 end entity;
12
13 architecture muxacao of mux2x12 is
14 begin
15
16         Bout <= Bsim when Op = '0' else Bnao;
17
18 end architecture;
```

2) “Negador”

O “Negador” foi feito de maneira simples, com apenas uma saída, que é o inverso da entrada.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity negador is
5      port(
6          Bnegara : in  std_logic_vector (11 downto 0);
7          Bnegado : out std_logic_vector (11 downto 0)
8  );
9  end entity;
10
11 architecture negacao of negador is
12 begin
13     Bnegado <= not Bnegara;
14
15 end architecture;
```

3) Overflow

Dentro da testbench, o overflow irá duplicar o bit mais significativo das entradas e da saída do somador.

```
library ieee;
use ieee.std_logic_1164.all;


entity overflow is
    port(
        Aov : in std_logic;
        Bov : in std_logic;
        Sov : in std_logic;
        Ov : out std_logic
        );
end entity;

architecture overflacao of overflow is
    begin

    Ov <= ((not Aov) and (not Bov) and Sov) or (Aov and Bov and (not Sov));
end architecture;
```

4) Somador de 12 Bits

O somador de 12 bits foi feito em duas etapas. Primeiramente, criou-se um componente contendo um somador completo de 1 bit:

```
library ieee;
use ieee.std_logic_1164.all;

entity somador is
    port(
        a    : in std_logic;
        b    : in std_logic;
        cin  : in std_logic;
        s    : out std_logic;
        cout : out std_logic
        );
end entity;


architecture comuta of somador is
    begin
        cout <= (a and (b or cin)) or (b and cin);
        s <= (a xor b) xor cin;
end architecture;
```

Logo após, o somador de 1 bit foi usado como componente dentro da arquitetura de uma entidade chamada “somador12bits”.

```
library ieee;
use ieee.std_logic_1164.all;

entity somador12bits is
    port(
        Asum   : in  std_logic_vector (11 downto 0);
        Bsum   : in  std_logic_vector (11 downto 0);
        Cin    : in  std_logic;
        Sumado : out std_logic_vector (11 downto 0);
        Cout   : out std_logic
    );
end entity;

architecture somagem of somador12bits is
    component somador is
        port(
            a    : in std_logic;
            b    : in std_logic;
            cin  : in std_logic;
            s    : out std_logic;
            cout : out std_logic
            );
    end component;
```

O somador de 1 bit foi repetido em 12 unidades diferentes, conectadas por meio de um único signal do tipo “std_logic_vector (10 downto 0)”, para unir as unidades por meio dos carry in’s e carry out’s:

```
signal carry : std_logic_vector (10 downto 0);

begin

    u_soma0 : somador port map (
        Asum(0), Bsum(0), Cin, Sumado(0), carry(0)
    );

    u_soma1 : somador port map (
        Asum(1), Bsum(1), carry(0), Sumado(1), carry(1)
    );

    u_soma2 : somador port map (
        Asum(2), Bsum(2), carry(1), Sumado(2), carry(2)
    );

    u_soma3 : somador port map (
        Asum(3), Bsum(3), carry(2), Sumado(3), carry(3)
    );

    u_soma4 : somador port map (
        Asum(4), Bsum(4), carry(3), Sumado(4), carry(4)
    );

    u_soma5 : somador port map (
```

E assim obteve-se a entidade do somador de 12 bits.

5) Testbench

Por fim, o *testbench* foi feito utilizando 8 sinais dos tipos “std_logic” e “std_logic_vector(11 downto 0)”.

Os sinais feitos e o portmap dos componentes estão representado abaixo:

```
48      signal s_Bim,  s_Bao      : std_logic_vector (11 downto 0);
49      signal s_Bmux, s_A, s_Som : std_logic_vector (11 downto 0);
50      signal s_Op, s_Of, s_Cout : std_logic;
51
52  begin
53      u_negador  : negador  port map (s_Bim, s_Bao);
54      u_mux2x12  : mux2x12  port map (s_Bim, s_Bao, s_Op, s_Bmux);
55      u_overflow : overflow port map (s_A(11), s_Bmux(11), s_Som(11), s_Of);
56      u_somador  : somador12bits port map (s_A, s_Bmux, s_Op, s_Som, s_Cout);
```

Por fim, o testbench foi finalizado com a inserção de diversos valores para serem somados e subtraídos. Aqui vão alguns exemplos:

```
--0xFFE 0xFFE
        s_A   <= x"FFE";
        s_Bim <= x"FFE";
        s_Op  <= '0';
        wait for 20 ns;

--0x0FF 0x0FF
        s_A   <= x"0FF";
        s_Bim <= x"0FF";
        s_Op  <= '0';
        wait for 20 ns;

--Algum caso intermediário
--0xABC 0xABC
        s_A   <= x"ABC";
        s_Bim <= x"ABC";
        s_Op  <= '0';
        wait for 20 ns;
```

6) Versão Com Delay

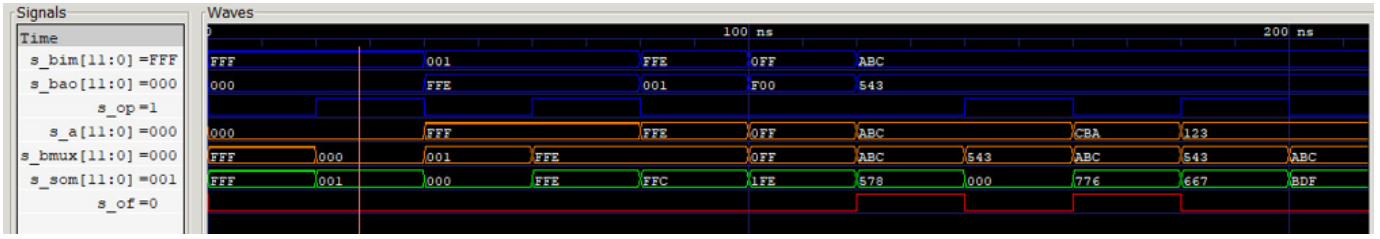
A versão dos arquivos com latência foi criada aplicando o comando “after **X** ns” após as expressões booleanas representadas dentro dos componentes. Muitos valores foram testados para a latência do somador, mas o que obteve resultados mais satisfatórios foi:

```
architecture comuta of somador is
    begin
        cout <= ((a and (b or cin)) or (b and cin)) after 4*2 ns;
        s <= ((a xor b) xor cin) after 4*2 ns;
end architecture;
```

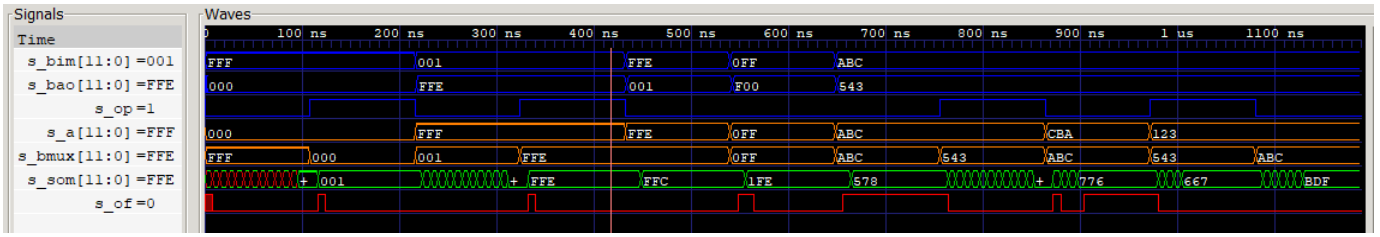
3.0 - GTKwave:

Dentro das pastas “[SD P04] VitorMayorca RafaelPascoali\Com_Latencia” e “[SD P04] VitorMayorca RafaelPascoali\Sem_Latencia” serão encontrados arquivos do tipo “.GTKW”. Eles possuem o seu diagrama de ondas já organizado, e com diferentes sinais sendo representados com cores diferentes, para a fácil análise dos resultados.

Em seguida, vemos o diagrama de ondas do testbench do somador sem latência:

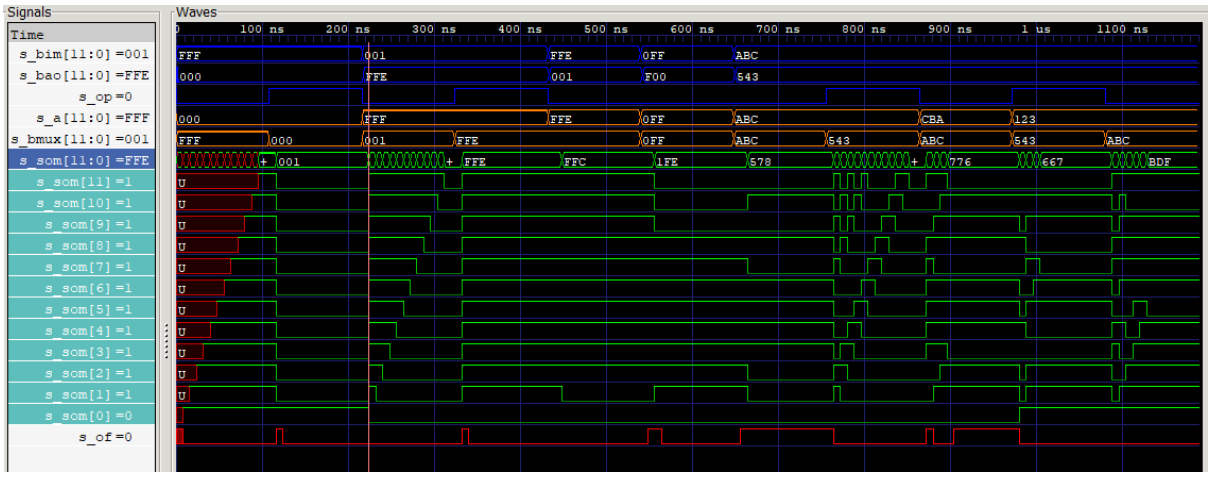


Seguido do diagrama de ondas do testbench com latência:



Percebe-se que, apesar de a latência garantir a qualidade do resultado final da operação de soma ou subtração, ambos os testbenches funcionaram perfeitamente, e sem nenhum problema nas saídas. Isso pode se dar pelo fato de que ambos os testbenches receberam um tempo considerável para se estabilizarem antes de iniciarem a operação seguinte.

Além disso, a nível de curiosidade, no testbench com latência é possível visualizar o somador de 12 bits agindo bit por bit:



Perceba que o sinal “s_som” forma diversas “escadas”. Como cada uma delas possui 12 “degraus”, pode-se concluir que cada degrau representa um bit sendo somado e liberando o resultado da soma, do bit menos significativo (mais abaixo) para o bit mais significativo (mais acima).