Introdução à Linguagem VHDL

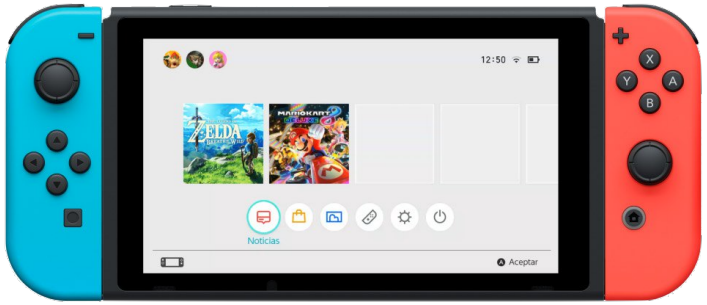
Prof. Fernando G. Moraes

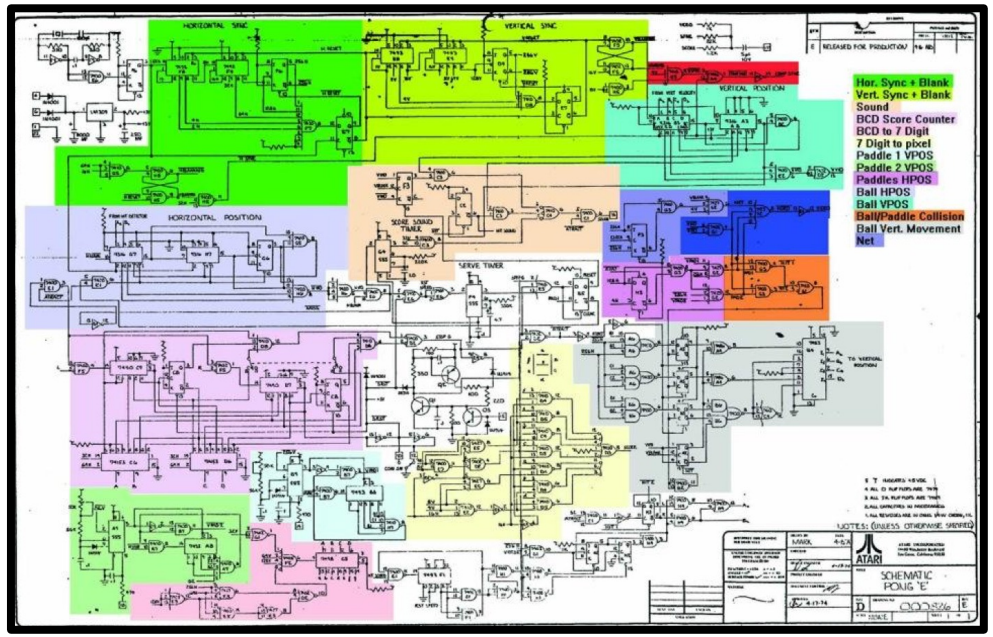
Motivação

Qual é o problema do projetarmos sistemas digitais a partir de *esquemáticos* ?

✓ Há 50 anos atrás já não era muito trivial...

✓ Hoje em dia...





Introdução

VHDL – **V**HSIC **H**ardware **D**escription **L**anguage

- ✓ Uma linguagem para *descrever* sistemas digitais
- ✓ Feita para *especificar* hardware. Atualmente serve para simulação e síntese
- ✓ Linguagem utilizada mundialmente por empresas de CAD, onde tem grande adoção na Europa. Já *Verilog* é muito usado nos EUA

Benefícios:

- ✓ Projetos independentes da tecnologia (implementação física é postergada)
- ✓ Ferramentas de CAD compatíveis entre si
- ✓ Permite, através de simulação, verificar o comportamento do sistema digital
- ✓ Reduz tempo e custo de projeto. Consequência: reduz "*time-to-market*" (tempo de chegada de um produto ao mercado)

O que este documento aborda?

Estrutura dos módulos: par entidade-arquitetura

Comandos simples

- Atribuição
- Operadores lógicos

Hierarquia: instanciar módulos dentro de módulos

Validação por simulação: test bench

VHDL é uma linguagem de programação?

- **Resposta educada ☺**
  - **Não!** É uma linguagem de descrição de hardware!

- **Código é executado em um simulador**
  - Não se enxerga o “compilador” de VHDL, não há um “código executável” visível

- **Testbench:**
  - Uma descrição em VHDL ou outra linguagem do procedimento de teste de um circuito
  - Especificação comportamental do ambiente externo ao projeto (estímulos externos)
  - Interage com o projeto
  - Não precisa ser descrito em VHDL (o projeto VHDL pode ser validado em ambiente C/C++)

Estrutura de um programa VHDL

- Cada módulo tem sua própria **entity** e **architecture**
- Toda a comunicação ocorre através das portas declaradas em cada entity, observando-se o tipo, tamanho, se se trata de sinal ou barramento e a direção
- Várias funções e tipos básicos são armazenados em bibliotecas (*library*). A biblioteca "IEEE" sempre é incluída
- Biblioteca do usuário (default): **work**. Todos os arquivos contidos no diretório de trabalho fazem parte da biblioteca do usuário

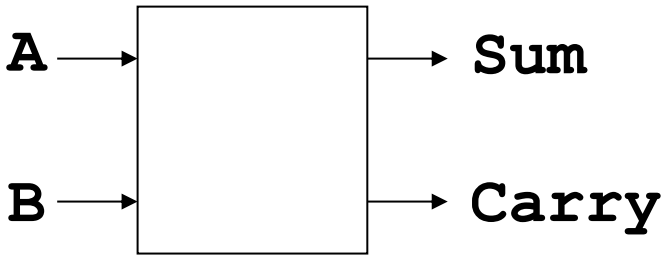
Entidade

Entity – Interface Externa

- ✓ Especifica somente a **interface** entre o hardware e o ambiente
- ✓ Não contém definição do comportamento ou da estrutura interna

Exemplo – Meio somador:

✓ Esquemático:



✓ Código VHDL da entidade:

```
entity half_adder is
port (A, B: in std_logic;
      Sum, Carry: out std_logic);
end entity;
```

Arquitetura

- A função de uma "*entity*" é determinada pela sua "*architecture*"
- Organização:

Architecture
Declarações ***signal*** - sinais de comunicação entre processos concorrentes sinais que comunicação entre processos concorrentes e os pinos de E/S ***type*** - novos tipos ***constant*** - constantes ***component*** - componentes (para descrever estrutura) ***function*** - subprogramas (*apenas a declaração destes)*
Begin (declarações concorrentes)
Atribuição a sinais Chamadas a "functions" e a "procedures" Instanciação de Componentes Processos: descrição de algoritmo
End

Arquitetura

Architecture – Comportamento

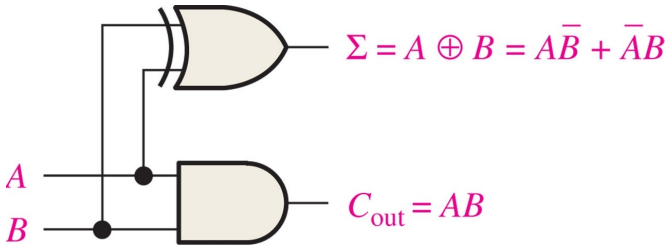
- ✓ Especifica o comportamento e/ou a estrutura interna da *entity*
- ✓ Deve ser associada a uma *entity* específica
- ✓ Uma *entity* pode ter associada a si várias ***architecture*** (representando diferentes formas de implementar um mesmo módulo)

Exemplo – Meio somador:

✓ Tabela Verdade:

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

✓ Circuito Combinacional:



✓ Código VHDL da arquitetura:

```
architecture half_adder of half_adder is
begin
  Sum   <= A xor B;
  Carry <= A and B;
end architecture;
```

DESCRIÇÃO COMPLETA DO MEIO SOMADOR

```
----------------------------------------
-- Biblioteca
----------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;

----------------------------------------
-- Entidade
----------------------------------------
entity half_adder is
    port (A, B: in std_logic;
          Sum, Carry: out std_logic);
end entity;

----------------------------------------
-- Arquitetura
----------------------------------------
architecture half_adder of half_adder is
begin
    Sum   <= A xor B;
    Carry <= A and B;
end architecture;
```

Esta biblioteca e pacote definem o tipo ***std_logic***, o qual não faz parte da linguagem VHDL!

- ✓ Assume valores {'0', '1', 'X', 'L', 'l', H', 'h', 'Z', 'U'}
- ✓ ***std_logic_vector***: tipo que indica um conjunto de bits *std_logic*, isto é um barramento de N bits
  - Exemplo: ***std_logic_vector(7 downto 0)***

Expressões Lógicas

Operadores nativos de VHDL:

- ✓ Operações lógicas: and, or, nand, nor, xor, not
- ✓ Operações relacionais: =, /=, <, <=, >, >=
- ✓ Operações aritméticas: - (unária), abs
- ✓ Operações aritméticas: +, -
- ✓ Operações aritméticas: *, /
- ✓ Operações aritméticas: mod, rem, **
- ✓ Concatenação: &
  - Exemplo: "1001" & "0011" resulta em "10010011"

Observação:

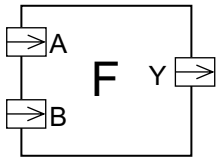
- ✓ Incluindo-se o pacote "use ieee.std_logic_unsigned.all" pode-se somar vetores de bits (*std_logic_vector*) apenas com operador +

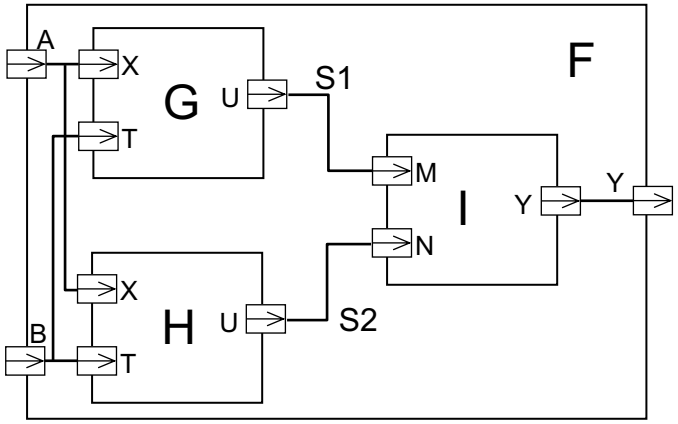
  – **Questão**: o que a seguinte linha de VHDL realiza? **X <= A <= B**

Uso de hierarquia: instanciação (1/2)

Módulos compostos por sub-módulos (**instâncias**) conectados por sinais:

- ✓ Na Figura ao lado, tem-se o módulo **F** (***entity*** desenhada à esquerda), com entradas A e B e saída Y.
- ✓ A ***architecture*** de **F** é composta pelos módulos G, H, I e pelos sinais internos s1 e s2. Note-se que G e H possuem a mesma *entity* (entradas X, T, e saída U), representando (G e H) duas instâncias distintas de um mesmo módulo de hardware.





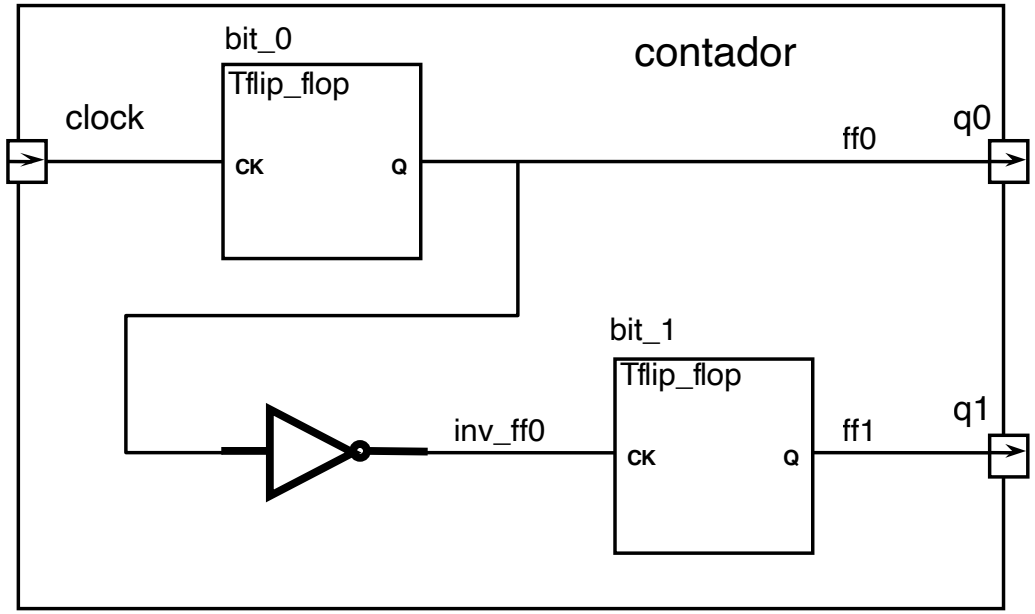
Em VHDL:

- ✓ Instâncias são denominadas ***component***
- ✓ Fios de interconexão: ***signal*** (sinais de um determinado tipo)
- ✓ Para criar relação entre sinais e conectores das instâncias: comando ***port map***

Uso de hierarquia: instanciação (2/2)

```
library IEEE;
use IEEE.std_logic_1164.all;

entity contador is
      port( clock : in std_logic;  q1, q0 : out std_logic);
end contador;

architecture estrutural of contador is
   signal ff0, ff1, inv_ff0 : std_logic;
begin
    bit_0:    entity work.Tflip_flop
              port map( ck=> clock, q => ff0);
    inv_ff0 <= not ff0;
    bit_1:    entity work.Tflip_flop
              port map( ck=> inv_ff0, q => ff1);
   q0 <= ff0;
   q1 <= ff1;
end estrutural;
```

Declaração da Interface Externa



Instanciações de módulos (lê-se ”pino” em ”sinal”)

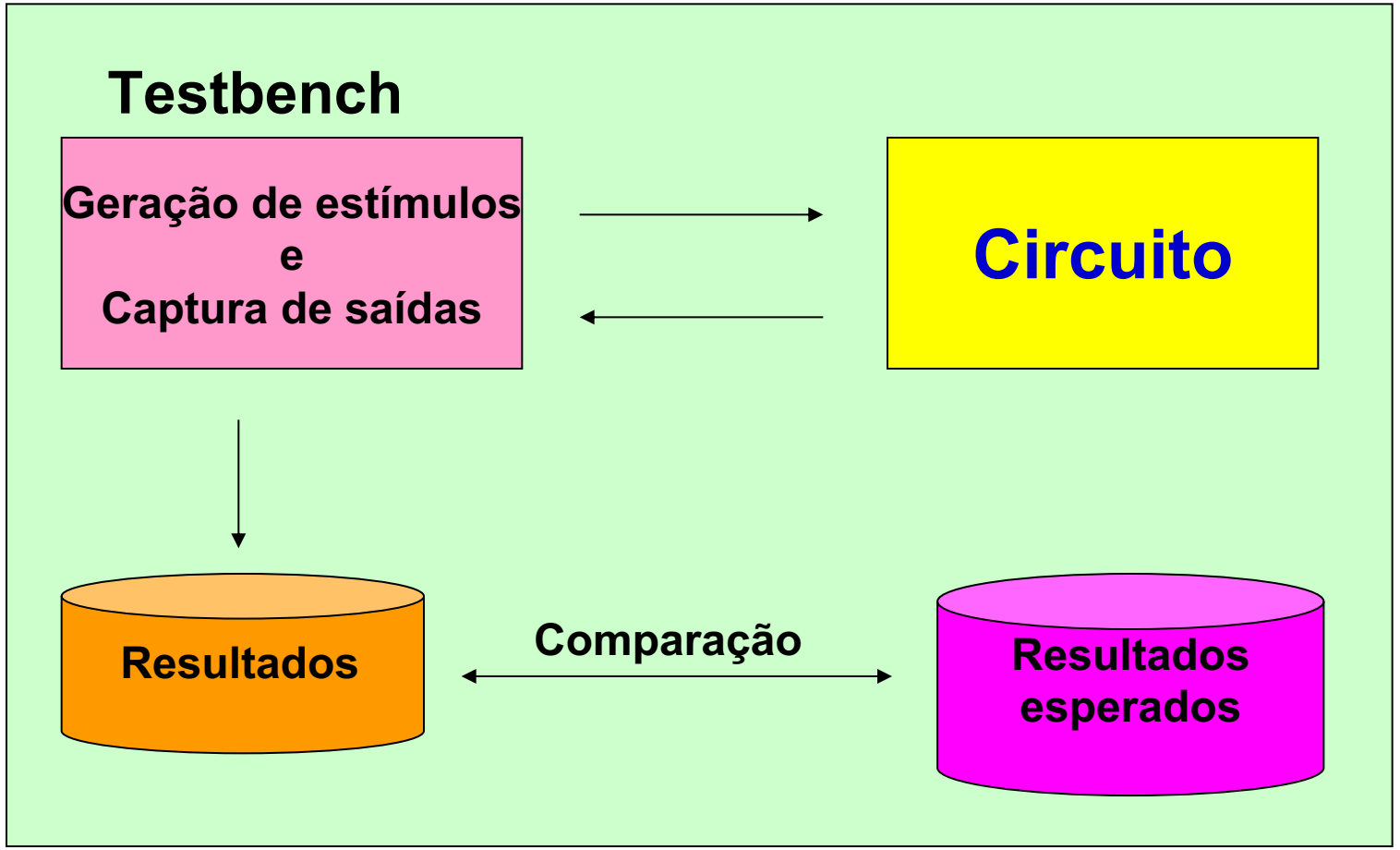
ATENÇÃO AOS SINAIS INTERNOS!

Ambiente de Verificação

O que este PPT aborda em VHDL?

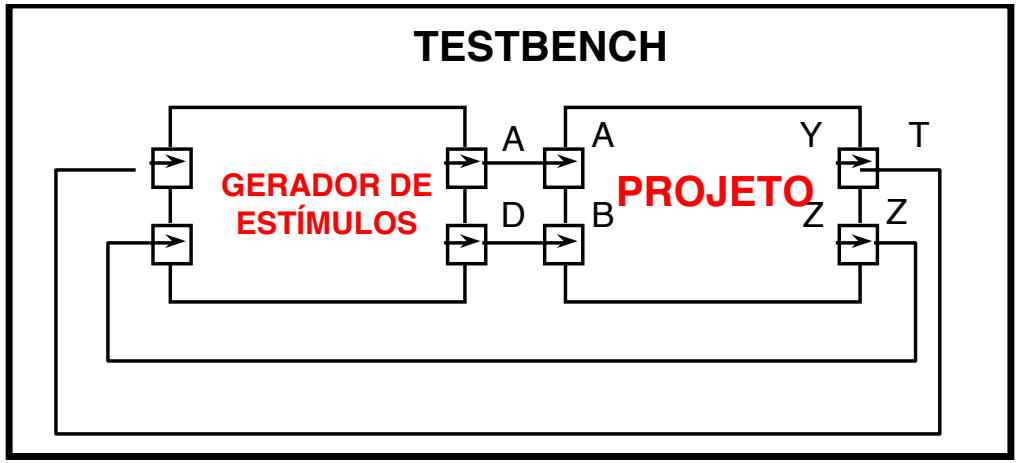
Estrutura dos módulos: par entidade-arquitetura
Comandos simples
- Atribuição
- Operadores lógicos

Hierarquia: instanciar módulos dentro de módulos
Validação por simulação: test bench



Simulação

- **Uma forma simples de testar o projeto: *testbenches***
  - Na sua forma mais simples, um *testbench* consiste em um ou mais "processos geradores de estímulos" e uma **instância** do projeto que se quer testar
  - O *testbench* é construído como um módulo VHDL que **não** contém portas de entrada/saída. Ou seja, trata-se de um sistema fechado, ou autônomo



Exemplo de testbench para o meio somador

```
-----------------------------------------
-- Biblioteca
-----------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;

-----------------------------------------
-- Entidade
-----------------------------------------
entity tb is
end entity;

-----------------------------------------
-- Arquitetura
-----------------------------------------
architecture tb of tb is
    signal A_tb, B_tb : std_logic;
    signal Sum_tb, Carry_tb : std_logic;
begin

    A_tb <= '0', '1' after 10 ns, '0' after 20 ns, '1' after 30 ns;

    B_tb <= '0', '1' after 20 ns;

    DUT: entity work.half_adder
         port map(A => A_tb, B => B_tb, Sum => Sum_tb, Carry => Carry_tb);

end architecture;
```

Testbench não tem pinos externos (ports in ou out)

Nome do projeto (entidade)

Geração dos estímulos, dizendo como os fios se comportam

Instanciação do projeto, conectando pinos do projeto aos fios do *testbench*

✓ *Atenção a ordem de **pinos** e **sinais***

Material de apoio para simulação

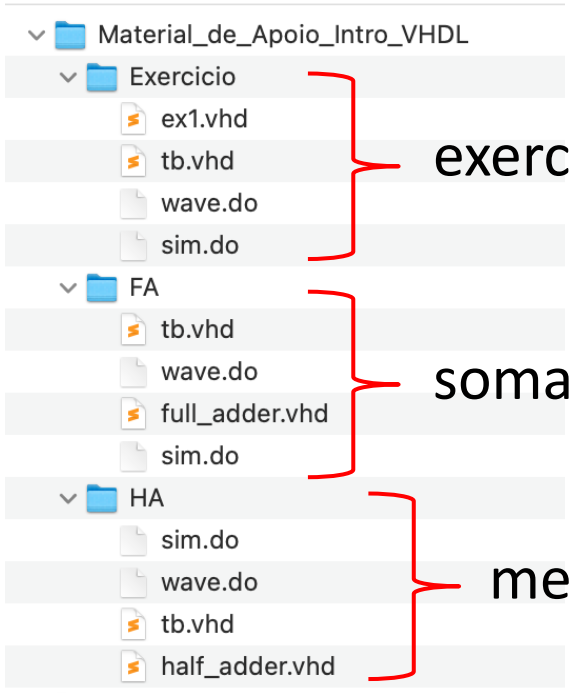
1. Baixe o Material de Apoio do Moodle

Introdução à VHDL e Simulação com ModelSim

Introdução a VHDL

Material de Apoio para Introdução à VHDL

2. Descompacte o Material de Apoio



exercício

somador completo

meio somador

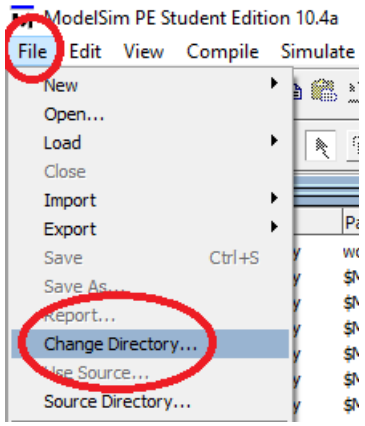
Cada pasta contém:

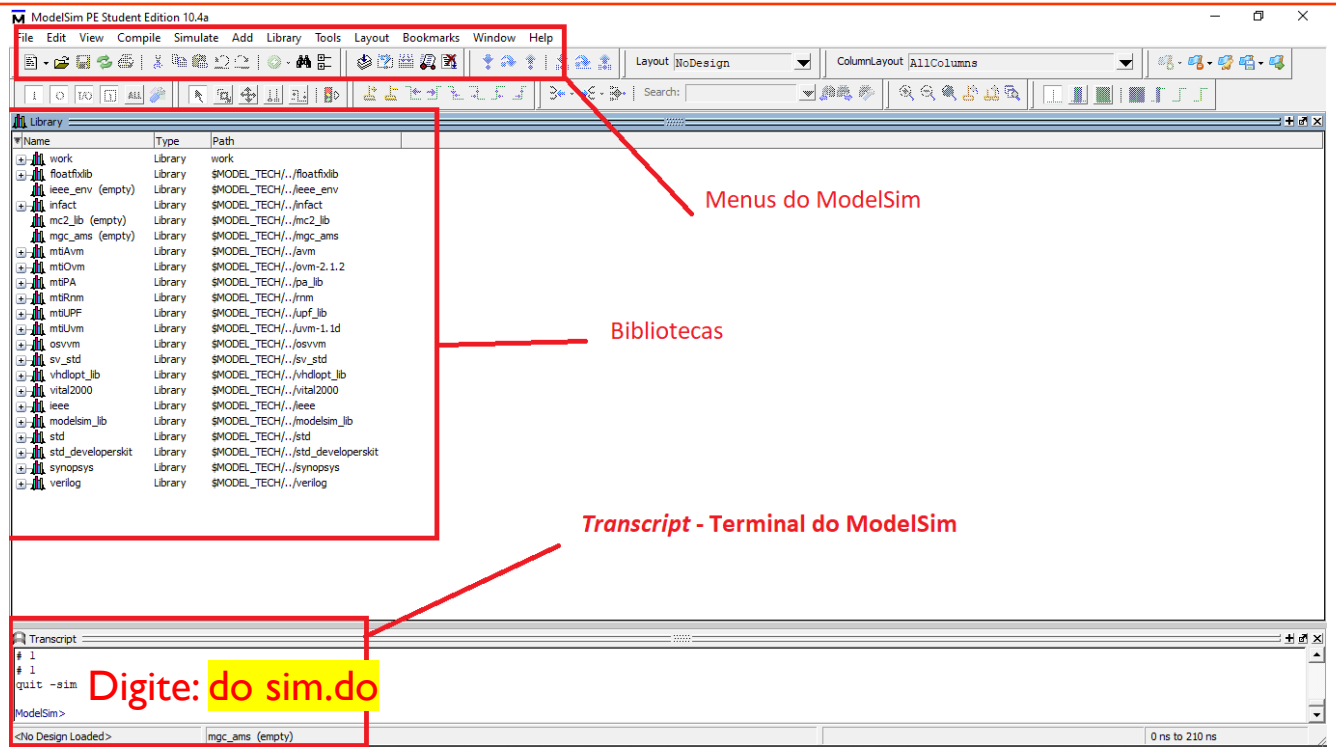
- VHDL do circuito
- test bench
- forma de onda – wave.do
- script de simulação

Exemplo de simulação para o meio somador

Abra o **Modelsim**

Altere o diretório escolhendo a pasta HA (meio somador) do Material de Apoio descompactado

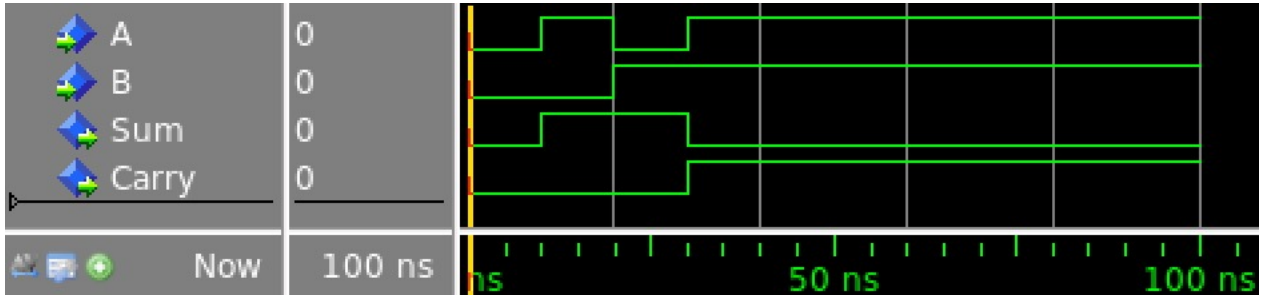




Nos laboratórios da PUC:

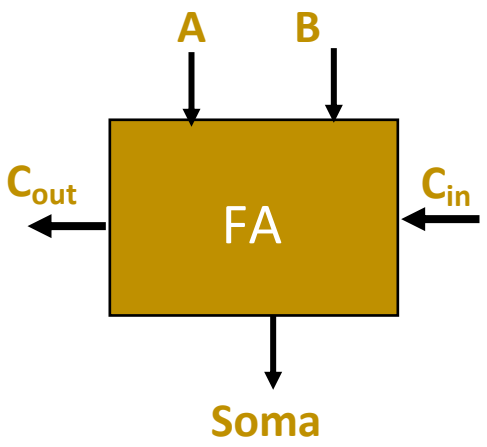
1. Login no LINUX
2. Digitar: source /soft64/source_gaph
3. Digitar: module load modelsim
4. Ir para o diretório de trabalho e digitar: vsim

Na aba *Wave* deve aparecer a seguinte forma de onda:



Somador Completo

✓ Esquemático:



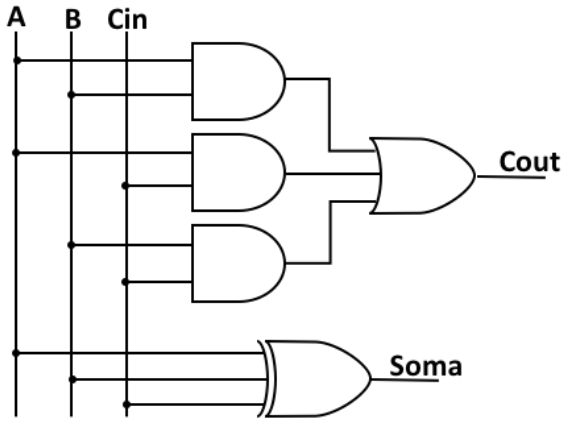
✓ Tabela Verdade:

A	B	C_{in}	Soma	C_{out}
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

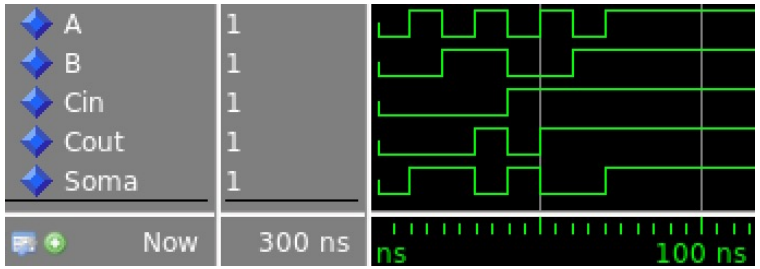
✓ Descrição em VHDL:

```
-------------------------------------------
-- Biblioteca
-------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

-------------------------------------------
-- Entidade
-------------------------------------------
entity full_adder is
    port (A, B, Cin: in std_logic;
          Cout, Soma: out std_logic);
end entity;

-------------------------------------------
-- Arquitetura
-------------------------------------------
architecture full_adder of full_adder is
begin

    Cout <= (A and B) or (A and Cin) or (B and Cin);

    Soma <= A xor B xor Cin;

end architecture;
```

✓ Circuito Combinacional:

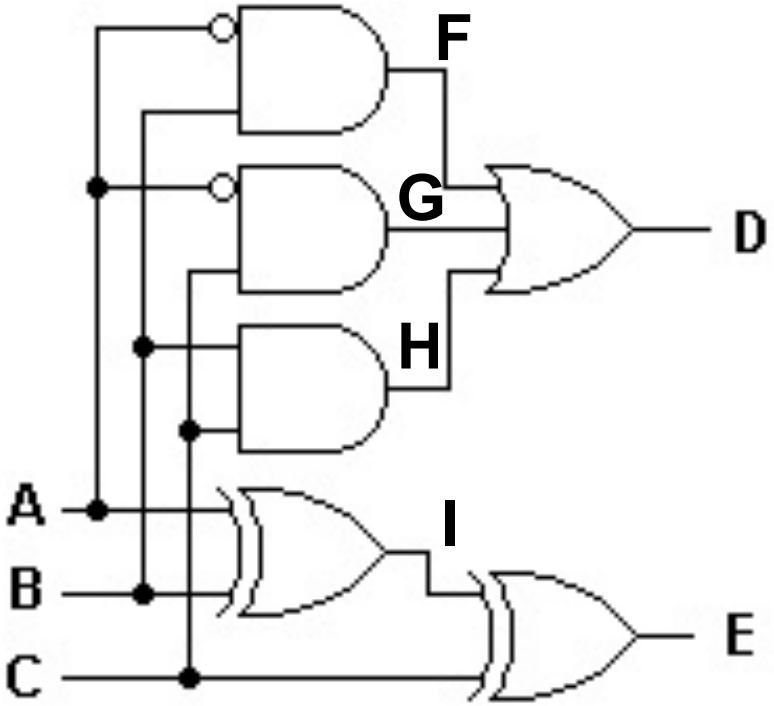


✓ Forma de Onda:



Exemplo de como descrever um circuito combinacional

Como você descreveria o circuito abaixo utilizando a linguagem VHDL?

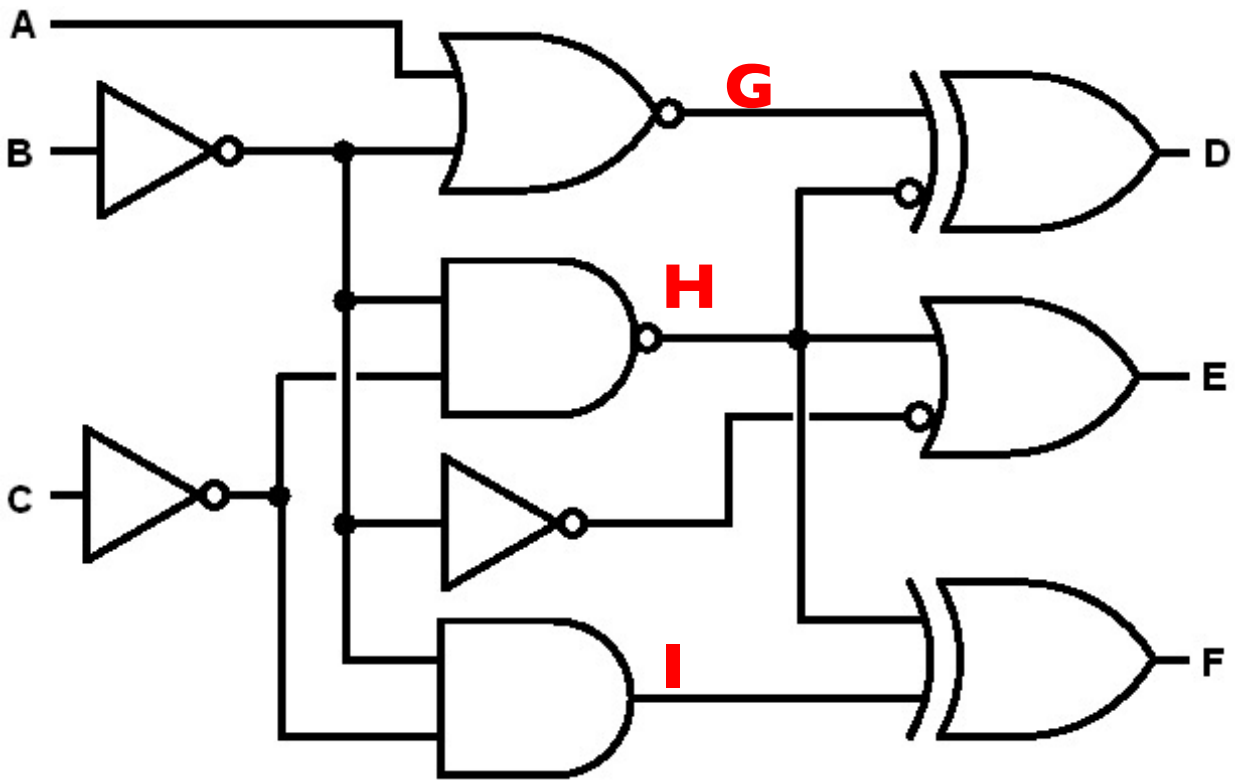


1. Desenhe os limites do circuito digital e descreva os pinos de entrada e saída escrevendo a *entidade* do circuito
2. Escreva letras indicando os sinais internos e comece a escrever a *arquitetura* declarando esses sinais
3. Descreva as portas e as ligações internais que modele a *arquitetura*

```
-- Biblioteca

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

-- Entidade

entity exemplo is
    port (A, B, C: in std_logic;
          D, E: out std_logic);
end entity;

-- Arquitetura

architecture exemplo of exemplo is
    signal F, G, H, I: std_logic;
begin
    F <= (not a) and B;
    G <= (not a) and C;
    H <= B and C;
    I <= A xor B;
    D <= F or G or H;
    E <= I xor C;
end architecture;
```

Exercício

1. Dado o esquemático abaixo, descreva utilizando a linguagem VHDL.

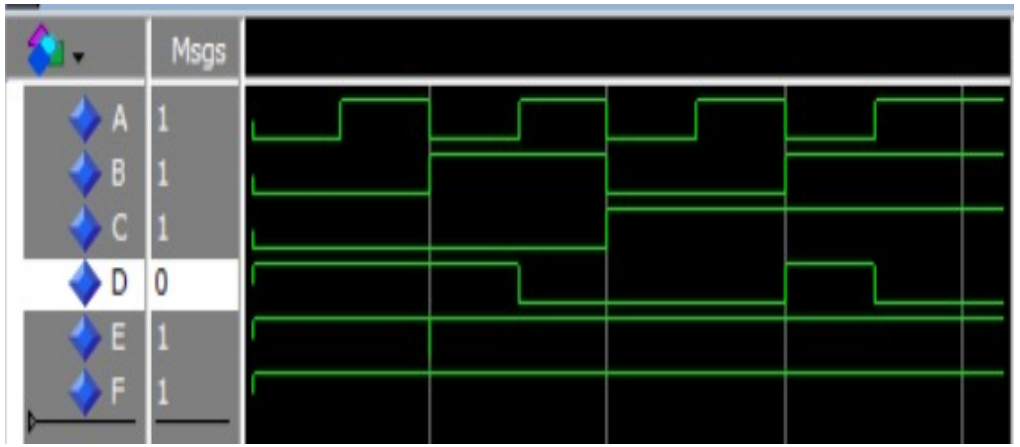


✓ Forma de onda da solução:

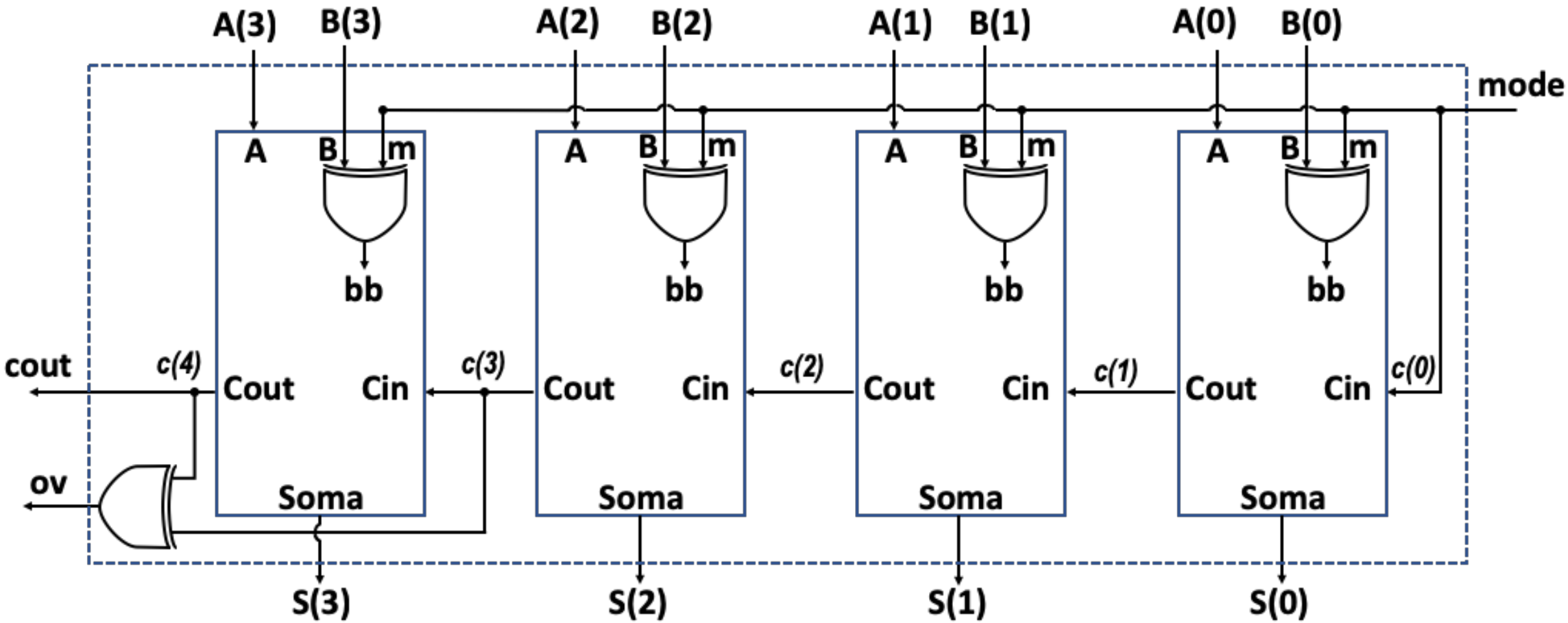

```
G <= A nor (not B);
H <= (not B) nand (not C);
I <= (not B) and (not C);
D <= G xor (not H);
E <= H or (not B));
F <= H xor I;
```

Solução

```
-------------------------------------
-- Biblioteca
-------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

-------------------------------------
-- Entidade
-------------------------------------
entity ex1 is
    port (A, B, C: in std_logic;
          D, E, F: out std_logic);
end entity;

-------------------------------------
-- Arquitetura
-------------------------------------
architecture ex1 of ex1 is
  signal G, H, I: std_logic;
begin
    G <= A nor (not B);
    H <= (not B) nand (not C);
    I <= (not B) and (not C);
    D <= G xor (not H);
    E <= H or (not B);
    F <= H xor I;
end architecture;
```



Soma e subtração

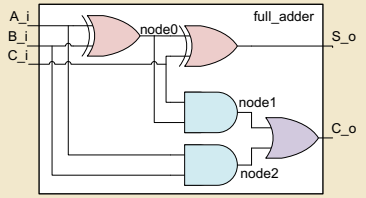


Soma e subtração

1. O primeiro passo é criar um par entidade-arquitetura baseado no somador completo.

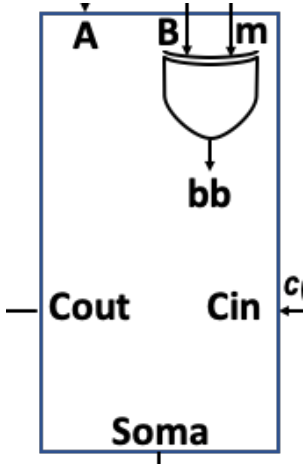
Há 3 modificações em relação ao FA original:

1. Uma entrada a mais na entidade – m (modo)
2. Um sinal interno – bb
3. Uma porta lógica xor adicional para inverter B quando m=1



Assim, as equações lógicas na arquitetura do *soma_sub* correspondem a:

bb <= B xor **m**;
Cout <= (A and **bb**) or (A and Cin) or (**bb** and Cin);
Soma <= A xor **bb** xor Cin;

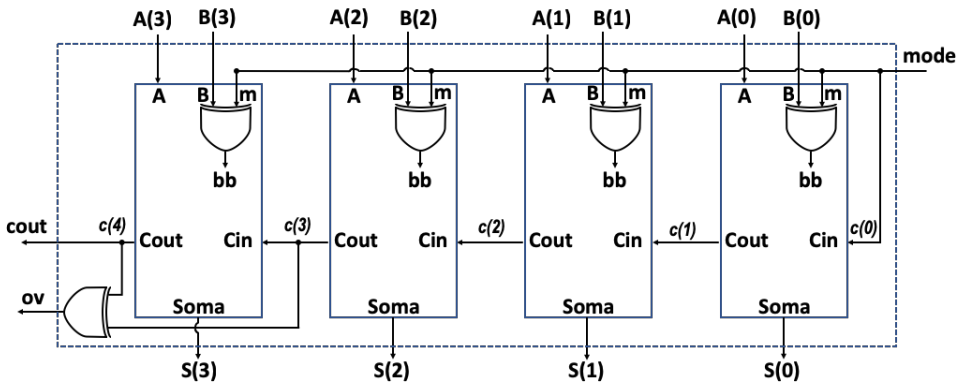



```
library IEEE;
use IEEE.std_logic_1164.all;

entity soma_sub is
    port (a, b, Cin, m: in std_logic;
          s, co: out std_logic);
end entity;

architecture a1 of soma_sub is
    signal bb: std_logic;
begin

    bb <= b xor m;
    co <= (a and bb) or (a and Cin) or (bb and Cin);
    s <= a xor bb xor Cin;

end architecture;
```

Soma e subtração

2. O segundo passo é criar um par entidade-arquitetura, *soma_sub4*. As modificações necessárias em relação ao somador de 4 bits são uma entrada a mais na entidade – *m* (modo), e a conexão deste sinal no *Cin* do primeiro *soma_sub*.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity soma_sub4 is
      port( a, b : in  std_logic_vector(3 downto 0);
            s    : out std_logic_vector(3 downto 0);
            M    : in  std_logic;
            OV   : out std_logic;
            CO   : out std_logic
     );
end soma_sub4;
```



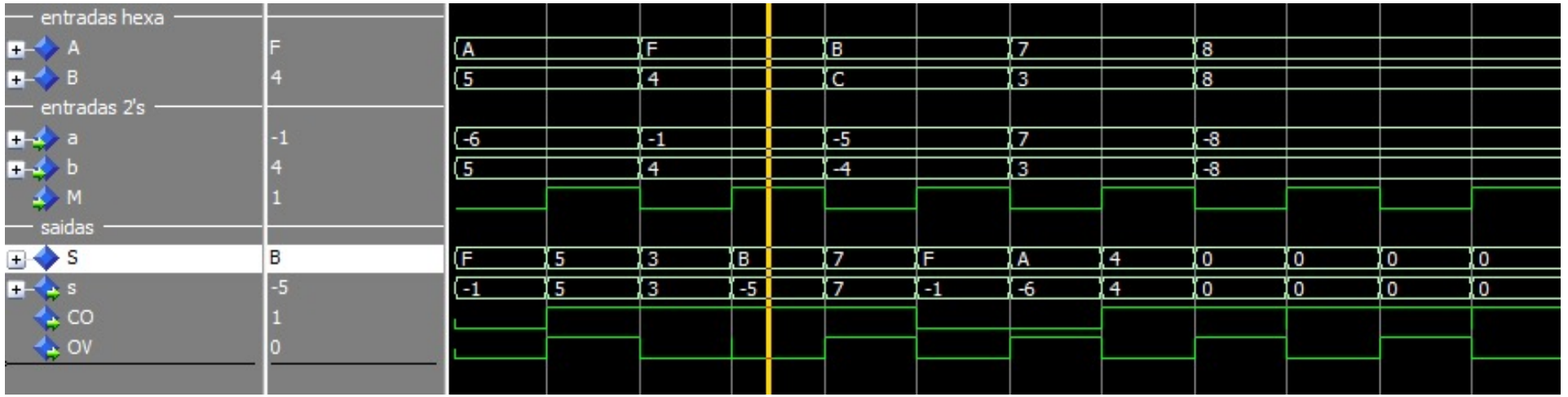

```
architecture  a1 of  soma_sub4 is
  signal carry : std_logic_vector(4 downto 0);
begin
  carry(0) <= M;
  CO       <= carry(4);
  OV       <= carry(4) xor carry(3);

  FA1: entity work.soma_sub
     port map( a => a(0),  b => b(0),  Cin  => carry(0),  m=>M, s => s(0), co => carry(1) );

  FA2: entity work.soma_sub
     port map( a => a(1),  b => b(1),  Cin  => carry(1),  m=>M, s => s(1), co => carry(2) );

  FA3: entity work.soma_sub
     port map( a => a(2),  b => b(2),  Cin  => carry(2),  m=>M, s => s(2), co => carry(3) );

  FA4: entity work.soma_sub
     port map( a => a(3),  b => b(3),  Cin  => carry(3),  m=>M, s => s(3), co => carry(4) );

end architecture a1;
```

Soma e subtração

3. O terceiro passo é criar o *test bench* para a geração das entradas para realizar somas e subtrações.

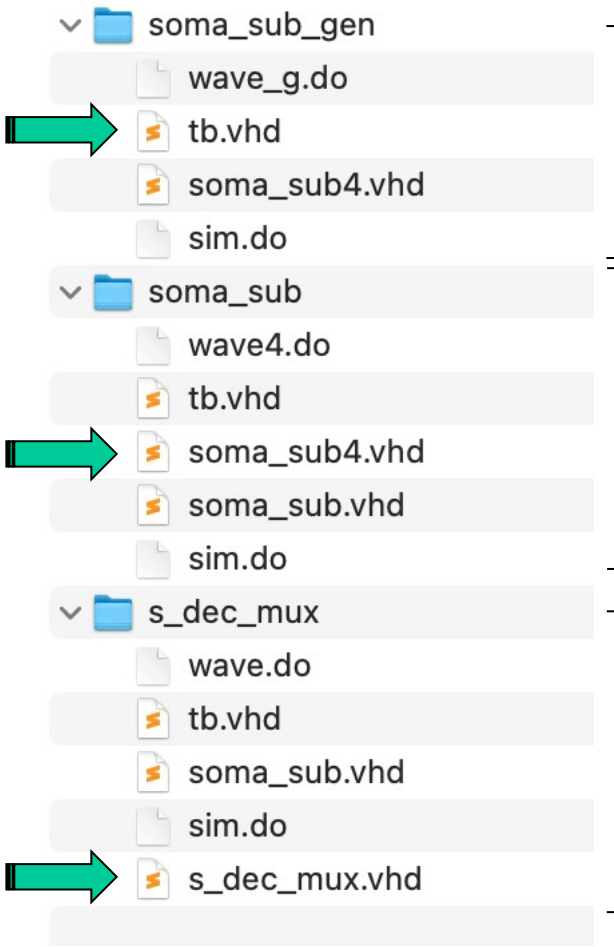
```
library IEEE;
use IEEE.std_logic_1164.all;

entity tb is
end entity;

architecture tb of tb is
    signal A, B : std_logic_vector(3 downto 0);
    signal S: std_logic_vector(3 downto 0);
    signal mode, cout, ov: std_logic := '0';
begin

    A <= x"A", x"F" after 20 ns, x"B" after 40 ns, x"7" after 60 ns, x"8" after 80 ns;
    B <= x"5", x"4" after 20 ns, x"C" after 40 ns, x"3" after 60 ns, x"8" after 80 ns;
    mode <= not mode after 10 ns;

    DUT: entity work.soma_sub4
         port map(a=>A, b=>B, M=>mode, s=>S, CO=>cout, OV=>ov);

end architecture;
```

Simulação – solução



Exercício: realizar esta simulação completando o código no arquivo soma_sub4.vhd

Material de apoio “soma sub”



Simulação 2 – somador genérico

Simulação 1 – somador 4 bits

Simulação 3 – exercício