Introdução ao VHDL

Circuitos Lógicos

DCC-IM/UFRJ

Prof. Gabriel P. Silva

Original por
Ayman Wahba

VHDL

- É uma linguaguem de descrição de “hardware”, ou seja, uma forma estruturada para a descrição de circuitos digitais.
- Essa linguagem permite que o circuito eletrônico seja descrito com sentenças, tais como em uma linguagem de programação, possibilitando que seja simulado e sintetizado, isto é, transformado em portas lógicas.
- Very High Speed ASIC Description Language

História do VHDL

- 1981: Iniciada pelo Departamento de Defesa dos EUA para resolver a crise do ciclo de vida dos projetos eletrônicos.
- 1983-85: Desenvolvimento da linguagem básica pela empresa Intermetrics, IBM e Texas Instruments.
- 1986: Todos os direitos transferidos para o IEEE.
- 1987: Publicação do padrão IEEE – VHDL 87.
- 1994: Padrão revisado (VHDL 93)

Porquê VHDL?

Aumenta dramaticamente a sua produtividade.

É uma forma muito mais rápida para projetar circuitos digitais.

Porquê VHDL?

Permite que o mesmo código seja usado com diversas tecnologias.

Isso garante portabilidade e longevidade para seu projeto.

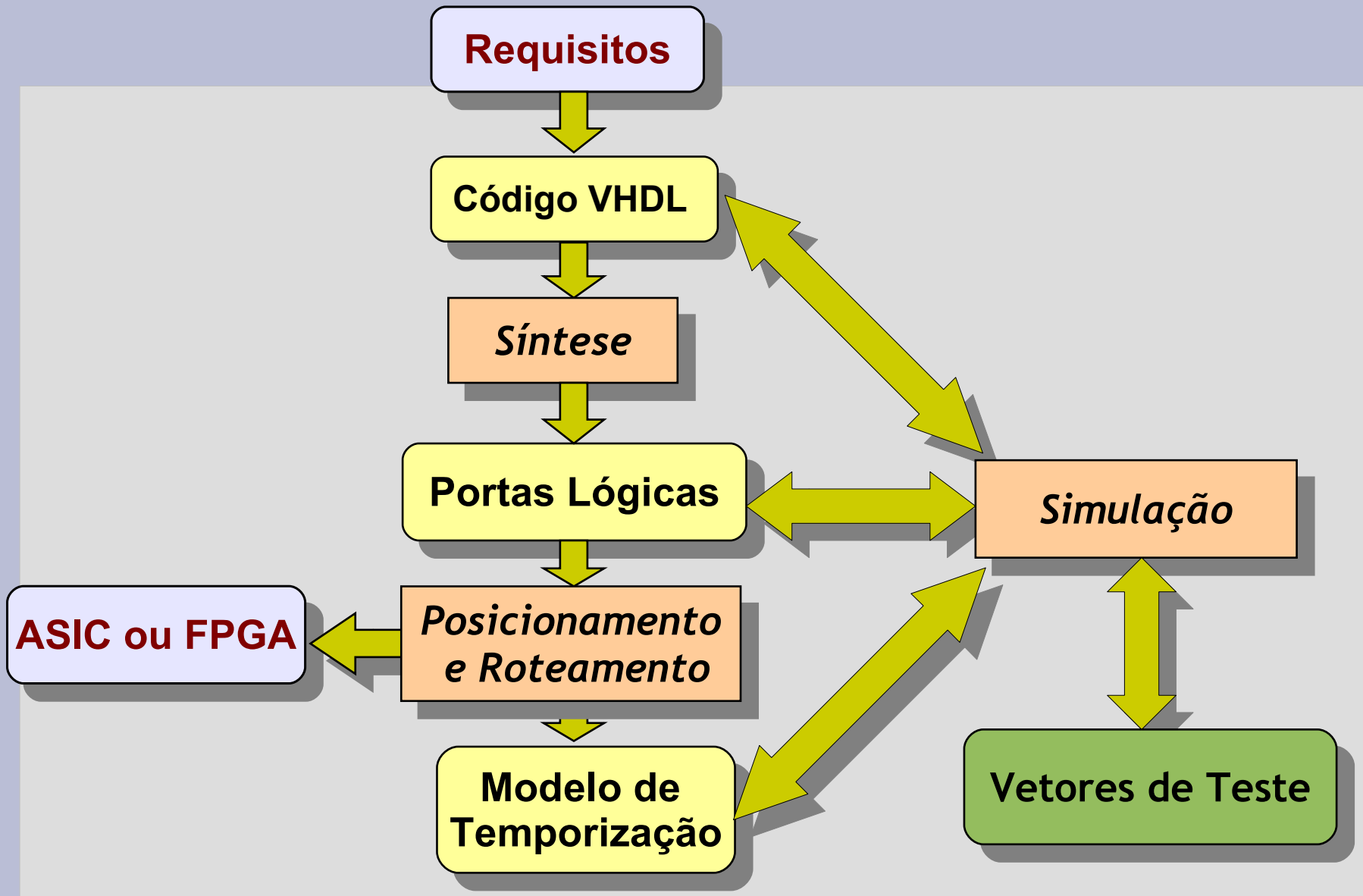
Porquê VHDL?

É possível testar o seu código em diversos níveis, garantindo maior confiabilidade nos resultados.

Como o VHDL é Utilizado?

- Para a especificação do projeto.
- Para a captura do projeto.
- Para a simulação do projeto.
- Para documentação do projeto.
- Como uma alternativa ao esquemático.
- Como uma alternativa às linguagens proprietárias.

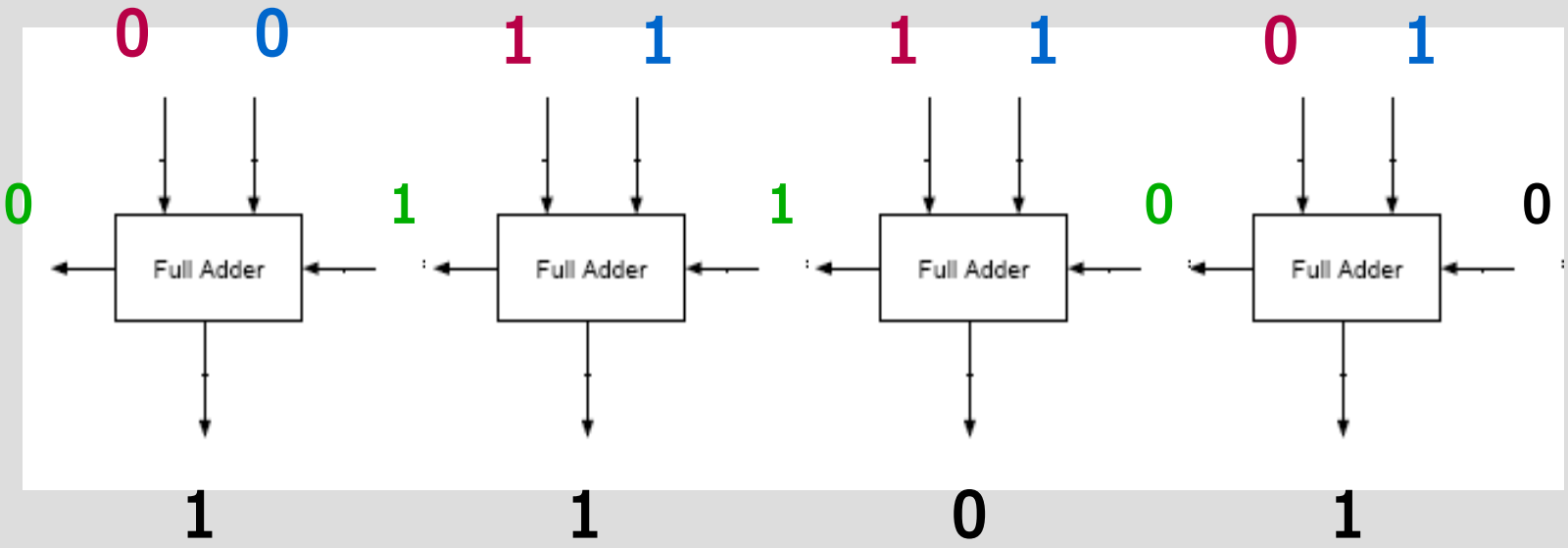
Metodologia Básica de Projeto



Somador

```
  1 1 0
  0 1 1 0
  0 1 1 1
-----------
  1 1 0 1
```

Somador

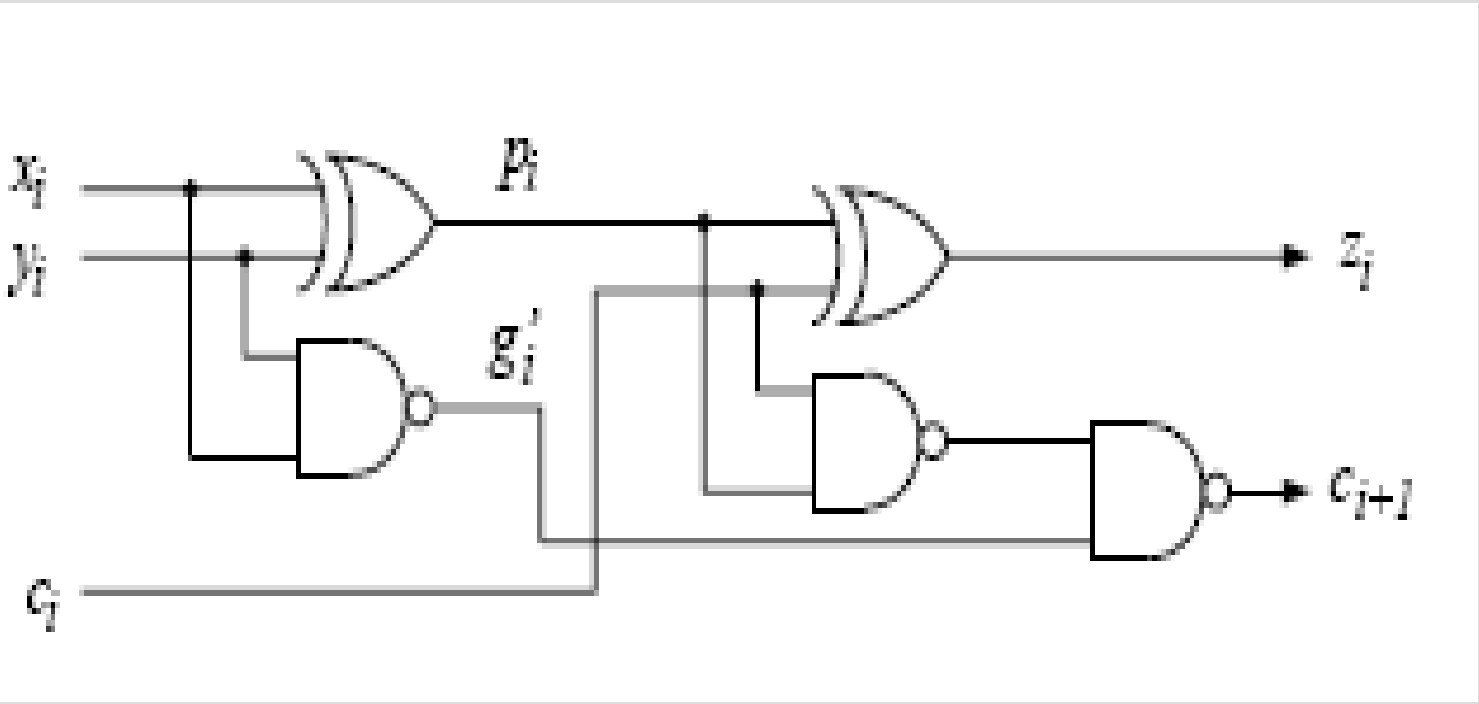

0
0
1
1
1
1
0
1
0
1
1
0
Full Adder
Full Adder
Full Adder
Full Adder
1
1
0
1

Somador

- Tabela da verdade

C_i	X_i	Y_i	S_i	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

Somador



Somador em VHDL

```
ENTITY somador IS
PORT ( Ci, Xi, Yi: IN BIT;
        Cout, Si; OUT BIT);
END somador;


ARCHITECTURE inicial OF somador IS
BEGIN
        Si <= Ci XOR Xi XOR Yi;
        Cout <= (Xi NAND Yi) NAND ( Ci NAND ( Xi XOR Yi));
END inicial;
```

Exemplo de Processo de Projeto

- Problema:

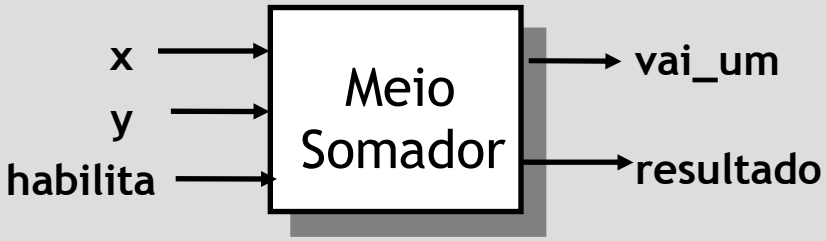
  Projetar um meio somador de um bit com vai_um e habilita.

- Especificações:
  - Passa o resultado apenas se habilita for igual a '1'.
  - Resultado é zero se habilita for igual a '0'.
  - Resultado recebe x + y
  - Vai_um recebe o vai_um, se houver, de x + y

Projeto Comportamental

- Iniciando com um algoritmo, uma descrição de alto nível do somador é criada:

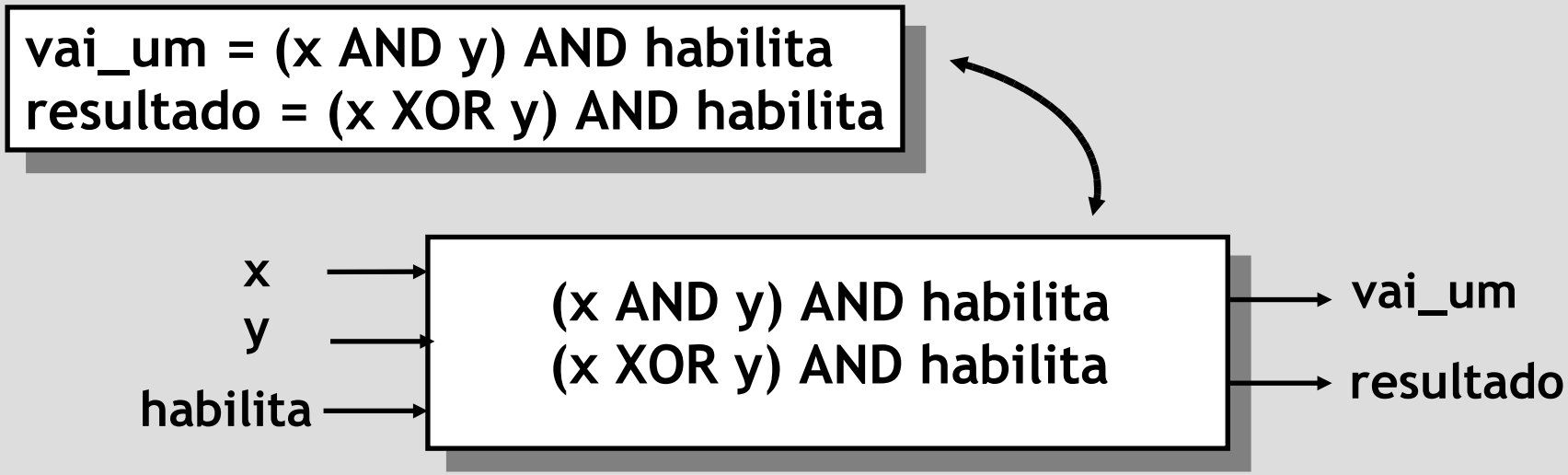
```
IF habilita = 1 THEN
        resultado = x XOR y
        vai_um = x AND y
ELSE
        vai_um = 0
        resultado = 0
```



- O modelo pode ser agora simulado nesse nível de descrição para verificar o correto entendimento do problema.

Projeto Fluxo de Dados

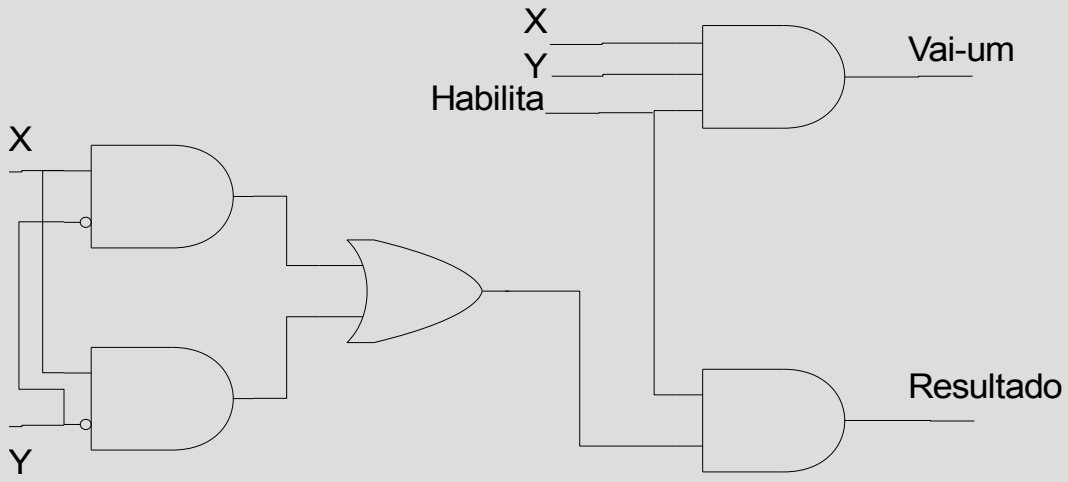
- Com a descrição de alto nível confirmada, equações lógicas descrevendo o fluxo de dados são então criadas.



- Novamente, o modelo pode ser simulado neste nível para confirmar as equações lógicas.

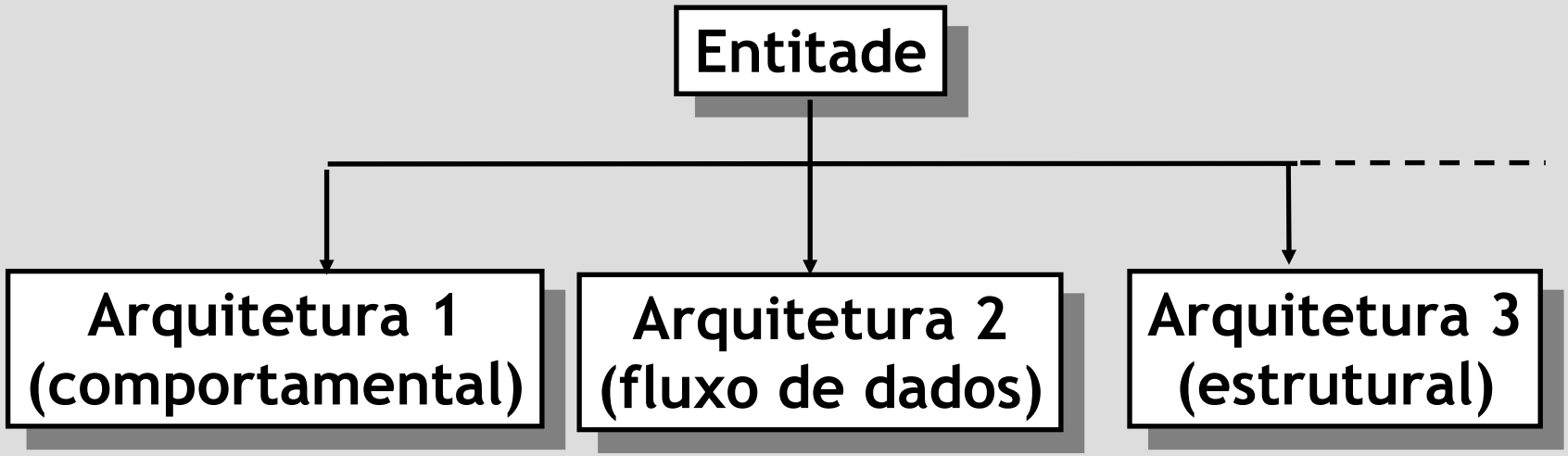
Projeto Lógico

- Finalmente, uma descrição estruturada é criada no nível de portas.



- Essas portas podem ser obtidas de uma biblioteca de componentes.

Processo de Projeto VHDL



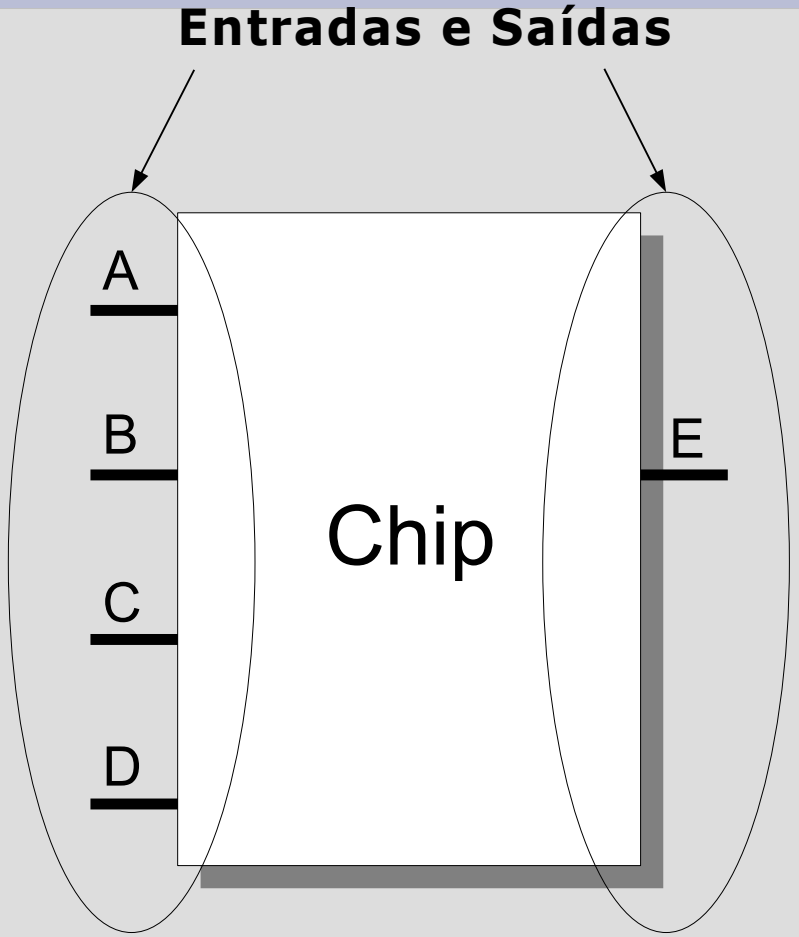
Declaração de Entidade

- Uma declaração de entidade (ENTITY) descreve a interface do componente.
- Uma cláusula PORT indica as portas de entrada e saída.
- Uma entidade pode ser pensada com um símbolo para um componente.

Entidade

- Define entradas e saídas
- Exemplo:

```
ENTITY teste IS
PORT ( A,B,C,D: IN STD_LOGIC;
       E: OUT STD_LOGIC);
End teste;
```



Declaração de Entidade

```
ENTITY meio_somador IS
     PORT (x, y, habilita: IN BIT;
                vai_um, resultado: OUT BIT);
END meio_somador;
```

Declaração de Porta

- Uma declaração de porta (PORT) estabelece a interface entre o componente e o mundo externo

- Há três partes na declaração PORT
  - Nome
  - Modo
  - Tipos de Dados

```
ENTITY teste IS
        PORT (<nome> : <modo>  <tipos_dados>);
END teste;
```

Nome

Qualquer identificador legal em VHDL

- Apenas letras, dígitos e sublinhados podem ser usados;
- O primeiro caracetere deve ser uma letra;
- O último caractere não pode ser um sublinhado;
- Não são permitidos dois sublinhados consecutivos.

Nomes Legais	Nomes Ilegais
rs_clk	_rs_clk
ab08B	sinal#1
A_1023	A__1023
	rs_clk_

Nome

- Não é sensível à “Caixa Alta ou Baixa”
  - inputa, INPUTA e InputA se referem à mesma variável.
- Comentários
  - ‘--’ marca um comentário até o final da linha atual
  - Se você deseja comentar múltiplas linha, um ‘--’ precisa ser colocado no início de cada linha.
- As sentenças são terminadas por ‘;’
- Atribuição de valores aos sinais: ‘<=’
- Atribuição de valores às variáveis: ‘:=’

Modo de Porta

- O modo da porta de interface descreve o sentido do fluxo de dados tomando com referência o componente.
- Os cinco tipos de fluxo de dados são:
  - IN: os dados entram nesta porta e podem apenas ser lidos (é o padrão).
  - OUT: os dados saem por essa porta e podem apenas serem escritos.
  - BUFFER: similar a Out, mas permite realimentação interna.
  - INOUT: o fluxo de dados pode ser em qualquer sentido, com qualquer número de fontes permitido (barramento)
  - LINKAGE: o sentido do fluxo de dados é deconhecido

Tipos de Dados

- Os tipos de dados que passam através de uma porta devem ser especificados para completar a interface.
- Os dados podem ser de diferentes tipos, dependendo do pacote e bibliotecas utilizados.
- Alguns tipos de dados definidos no padrão IEEE são:
  - BIT, BIT_VECTOR
  - BOOLEAN
  - INTEGER
  - STD_LOGIC, STD_ULOGIC

Tipos de Dados

- **bit values:** '0', '1'
- **boolean** values: TRUE, FALSE
- **integer** values: -(231) to +(231 – 1)
- **std_logic** values: 'U','X','1','0','Z','W','H','L','-'

  U' = uninitialized
  'X' = unknown
  'W' = weak 'X‘
  'Z' = floating
  'H'/'L' = weak '1'/'0‘
  '-' = don't care

- **std_logic_vector** (n downto 0);
- **std_logic_vector** (0 upto n);

Arquitetura

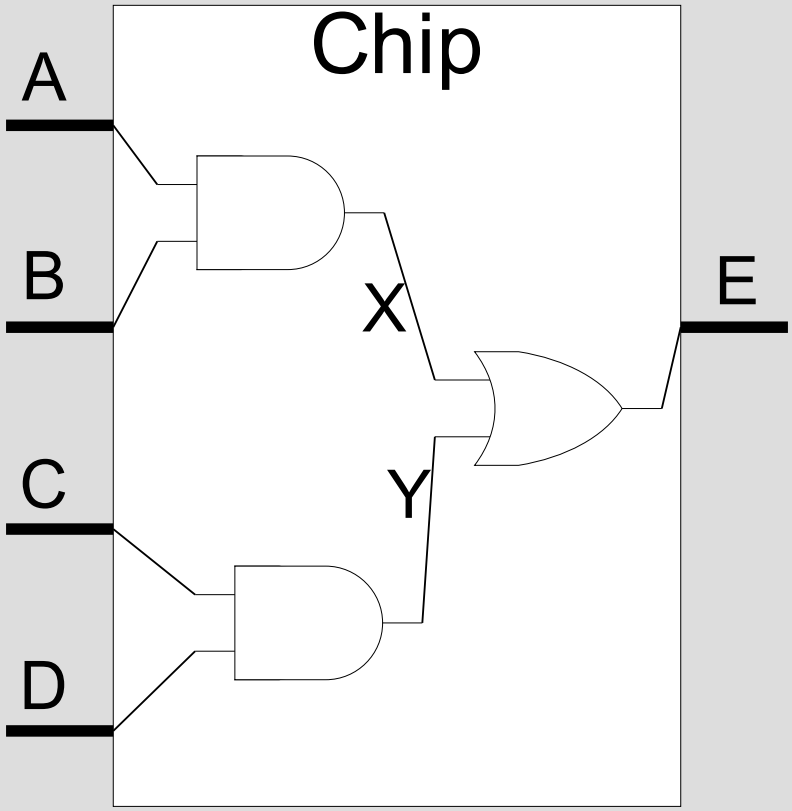
- Declarações do tipo Architecture descrevem o funcionamento do componente.

- Muitas arquiteturas podem existir para uma mesma entidade, mas apenas pode haver uma delas ativa por vez.

Arquitetura

- Define a funcionalidade do circuito

```
X <= A AND B;
Y <= C AND D;
E <=  X OR Y;
```



Arquitetura # 1

```
ARCHITECTURE comportamental OF meio_somador IS
BEGIN
        PROCESS (habilita, x, y)
        BEGIN
            IF (habilita = '1') THEN
                resultado <= x XOR y;
                vai_um <= x AND y;
            ELSE
                vai_um <= '0';
                resultado <= '0';
        END PROCESS;
END comportamental;
```

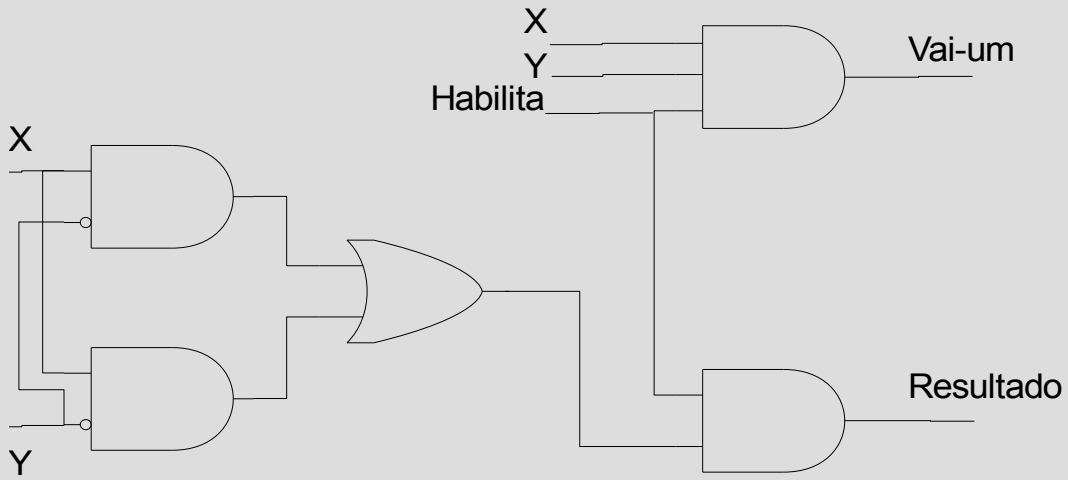
Arquitetura # 2

```
ARCHITECTURE fluxo_dados OF meio_somador IS
BEGIN
        vai_um <= (x AND y) AND habilita;
        resultado <= (x XOR y) AND habilita;
END fluxo_dados;
```

Arquitetura # 3

- Para fazer a arquitetura estrutural, nós precisamos primeiro definir as portas a serem utilizadas.

- No exemplo a seguir, nós precisamos definir as portas NOT, AND, e OR.



Arquitetura # 3

```
ENTITY not_1 IS
        PORT (a: IN BIT; output: OUT BIT);
END not_1;

ARCHITECTURE data_flow OF not_1 IS
BEGIN
        output <= NOT(a);
END data_flow;
```

```
ENTITY and_2 IS
        PORT (a,b: IN BIT; output: OUT BIT);
END not_1;

ARCHITECTURE data_flow OF and_2 IS
BEGIN
        output <= a AND b;
END data_flow;
```

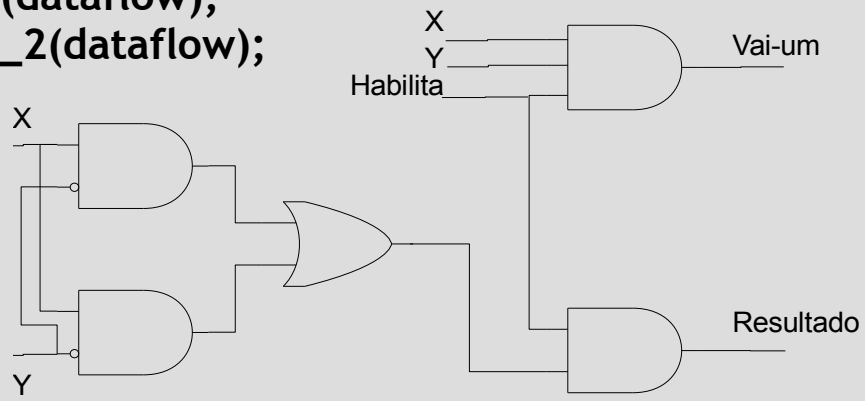
Arquitetura # 3

```
ENTITY or_2 IS
        PORT (a,b: IN bit; output: OUT bit);
END or_2;

ARCHITECTURE data_flow OF or_2 IS
BEGIN
        output <= a OR b;
END data_flow;
```

```
ENTITY and_3 IS
        PORT (a,b,c: IN BIT; output: OUT BIT);
END and_3;

ARCHITECTURE data_flow OF  and_3 IS
BEGIN
        output <= a AND b AND c;
END data_flow;
```

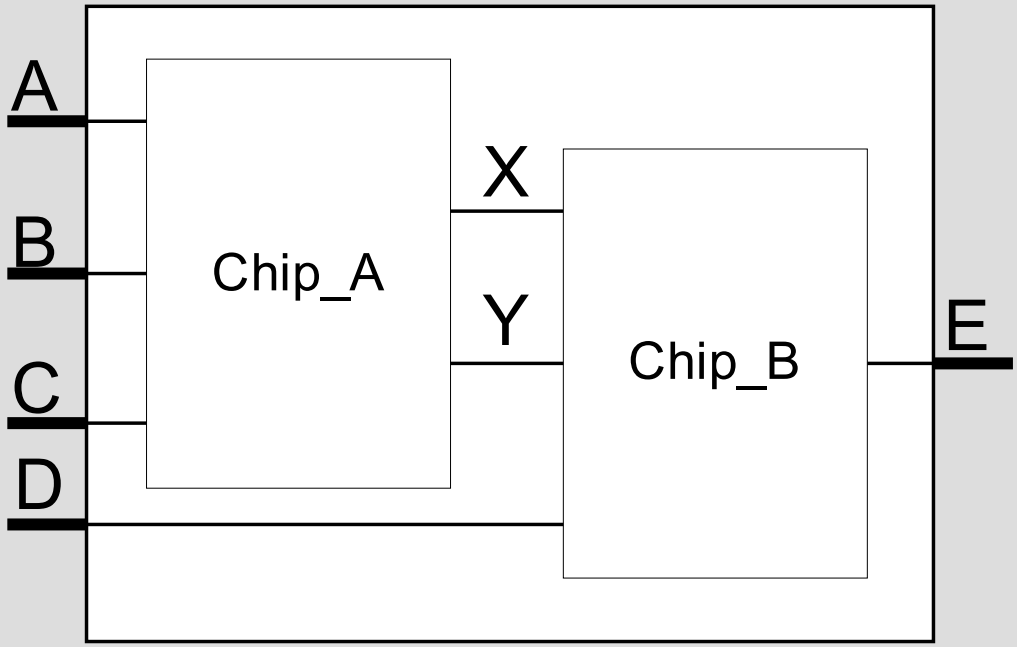
Arquitetura # 3

```
ARCHITECTURE structural OF meio_somador IS
   COMPONENT and2 PORT(a,b: IN bit; output: OUT bit); END COMPONENT;
   COMPONENT and3 PORT(a,b,c: IN bit; output: OUT bit); END COMPONENT;
   COMPONENT or2 PORT(a,b: IN bit; output: OUT bit); END COMPONENT;
   COMPONENT not1 PORT(a: IN bit; output: OUT bit); END COMPONENT;


   FOR ALL: and2 USE ENTITY work.and_2(dataflow);
   FOR ALL: and3 USE ENTITY work.and_3(dataflow);
   FOR ALL: or2 USE ENTITY work.or_2(dataflow);
   FOR ALL: not1 USE ENTITY work.not_2(dataflow);

   SIGNAL v,w,z,nx,nz: BIT;

BEGIN
   c1: not1 PORT MAP (x,nx);
   c2: not1 PORT MAP (y,ny);
   c3: and2 PORT MAP (nx,y,v);
   c4: and2 PORT MAP (x,ny,w);
   c5: or2 PORT MAP (v,w,z);
   c6: and2 PORT MAP (habilita,z,result);
   c7: and3 PORT MAP (x,y,habilita,vai_um);
END structural;
```



Port Map

```
Chip1 : Chip_A
PORT MAP (A,B,C,X,Y);

Chip2 : Chip_B
PORT MAP (X,Y,D,E);
```



Exemplo Port Map

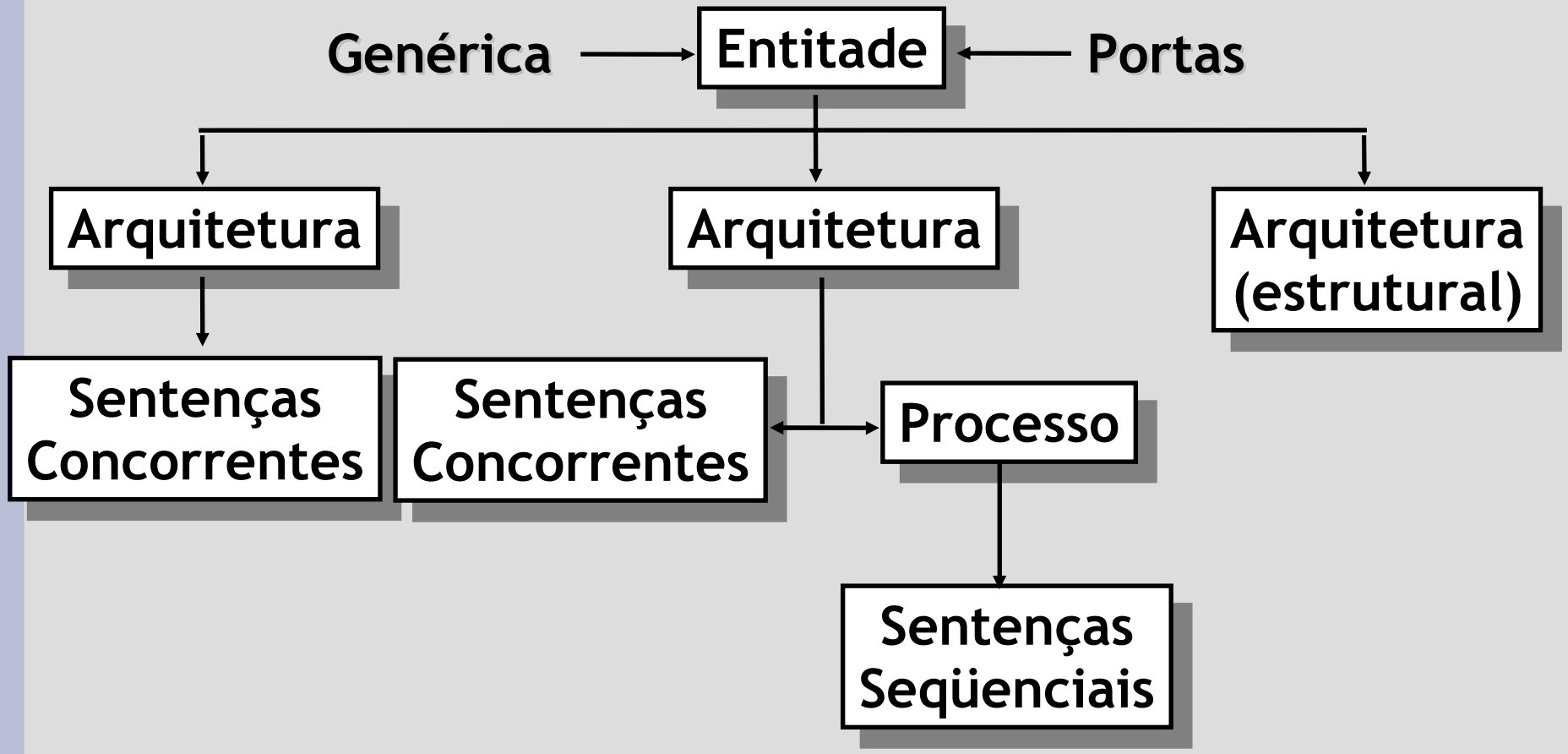
```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY TEST IS
PORT (A,B,C,D  :  IN STD_LOGIC;
          E          : OUT STD_LOGIC);
END TEST;

ARCHITECTURE BEHAVIOR OF TEST IS

SIGNAL X,Y : STD_LOGIC;

COMPONENT Chip_A
PORT (L,M,N   :  IN STD_LOGIC;
          O,P    : OUT STD_LOGIC);
END COMPONENT;
```

```
COMPONENT Chip_B
PORT (Q,R,S :  IN STD_LOGIC;
             T  : OUT STD_LOGIC);
END COMPONENT;

BEGIN

Chip1 : Chip_A
PORT MAP (A,B,C,X,Y);

Chip2 : Chip_B
PORT MAP (X,Y,D,E);

END BEHAVIOR;
```

Exemplo AND

```
ENTITY and2 IS
    PORT ( a, b : IN BIT;  y : OUT BIT);
END ENTITY and2;

ARCHITECTURE basic OF and2 IS
BEGIN
    and2_behavior : PROCESS IS
    BEGIN
        y <= a AND b AFTER 2 ns;
        WAIT ON a, b;
    END PROCESS and2_behavior;
END ARCHITECTURE basic;
```

Resumo


Genérica
Entitade
Portas
Arquitetura
Arquitetura
Arquitetura (estrutural)
Sentenças Concorrentes
Sentenças Concorrentes
Processo
Sentenças Seqüenciais

VHDL

* Objetos de Dados
* Tipos de Dados
* Tipos e Subtipos
* Atributos
* Sentenças Concorrentes e Sequenciais
* Procedimetos e Funções
* Pacotes e Bibliotecas
* Generics
* Tipos de Atraso