UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
CENTRO MULTIDISCIPLINAR DE PAU DOS FERROS
DEPARTAMENTO DE ENGENHARIAS E TECNOLOGIA

LABORATÓRIO DE CIRCUITOS DIGITAIS

Verilog - Abordagem Estrutural

Prof.: Pedro Thiago Valério de Souza
UFERSA – Campus Pau dos Ferros
pedro.souza@ufersa.edu.br

Linguagem de Descrição de *Hardware*

- Uma linguagem de descrição de *hardware* (HDL - *Hardware Description Language*) é a linguagem textual usada para descrever o *hardware* a ser sintetizado em uma FPGA/CPLD;
- Linguagens comuns:
  - AHDL (*Altera Hardware Description Language*);
  - VHDL (VHSIC - *Very High Speed Integrated Circuits - Hardware Description Language*);
  - Verilog;
  - System C;
- Dentre as linguagens de descrição de *hardware* destaca-se o Verilog;

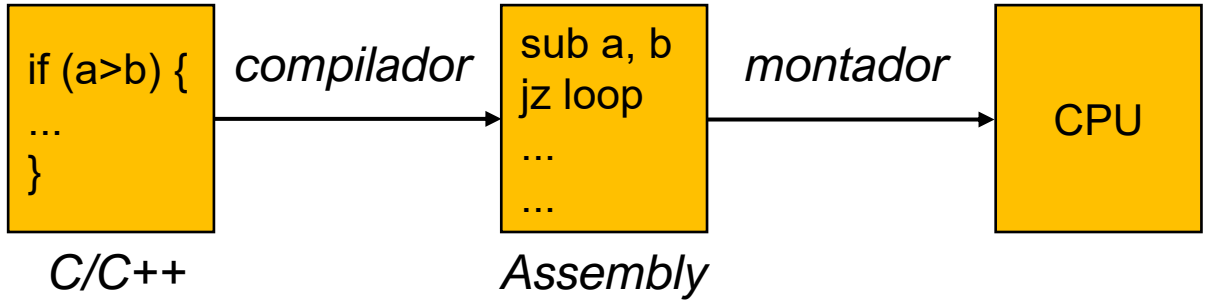
Linguagem de Descrição de *Hardware*

- **Vantagens na utilização do HDL:**
  - Projetos independentes da tecnologia (CPLD/FPGA);
    - Portabilidade;
  - Facilidade de atualização dos projetos;
  - Projeto em um nível mais alto de abstração;
  - Redução do tempo de projeto, de testes e implementação;
  - Simplificação quanto a sua documentação;

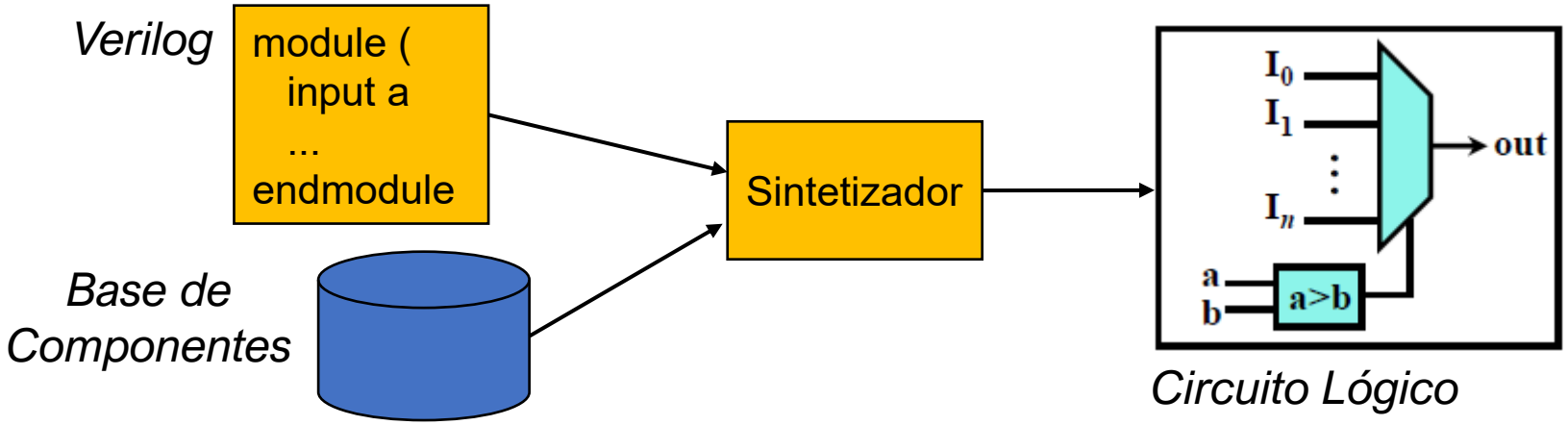
- **Desvantagens na utilização do HDL:**
  - O *hardware* gerado geralmente é menos otimizado.

Linguagem de Descrição de *Hardware*

- Linguagem de Descrição de *Hardware* vs. Linguagem de Programação.
  - Linguagem de Programação:

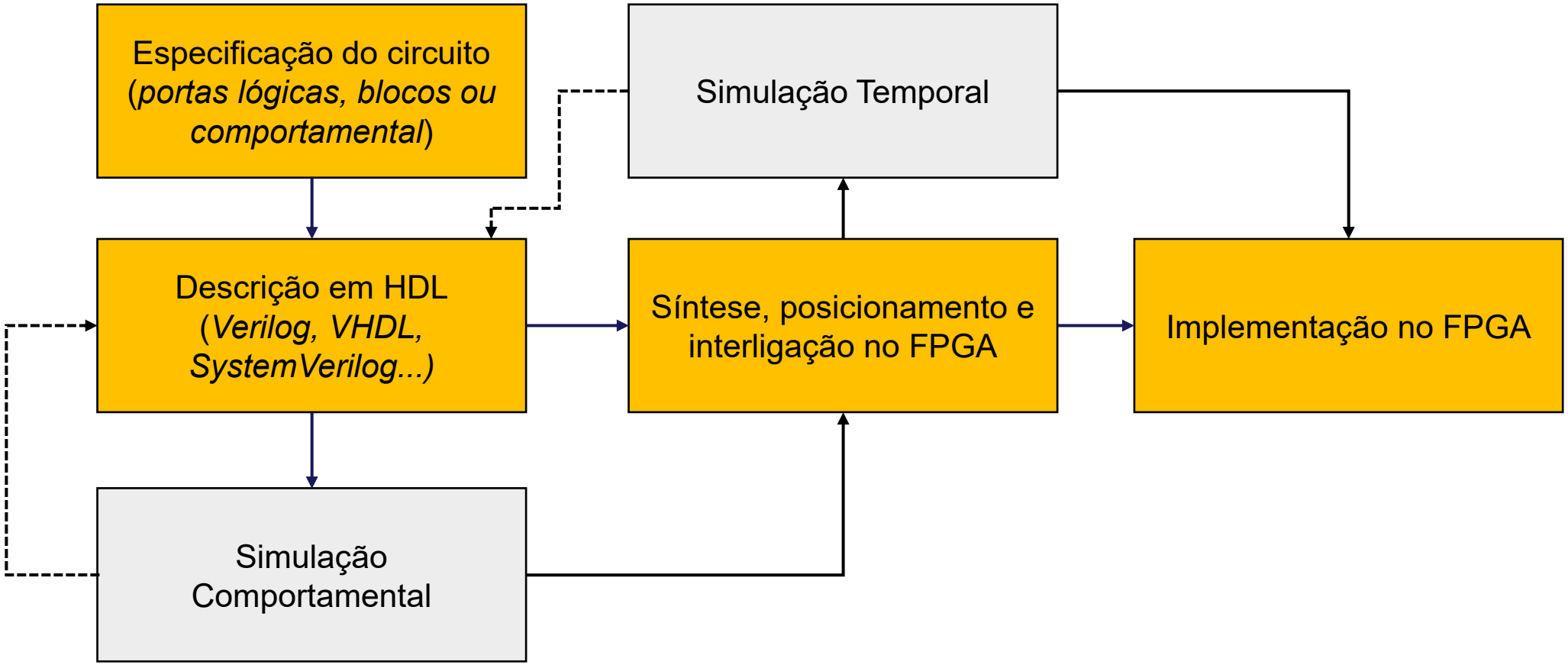


  - Linguagem de Descrição de *Hardware*:



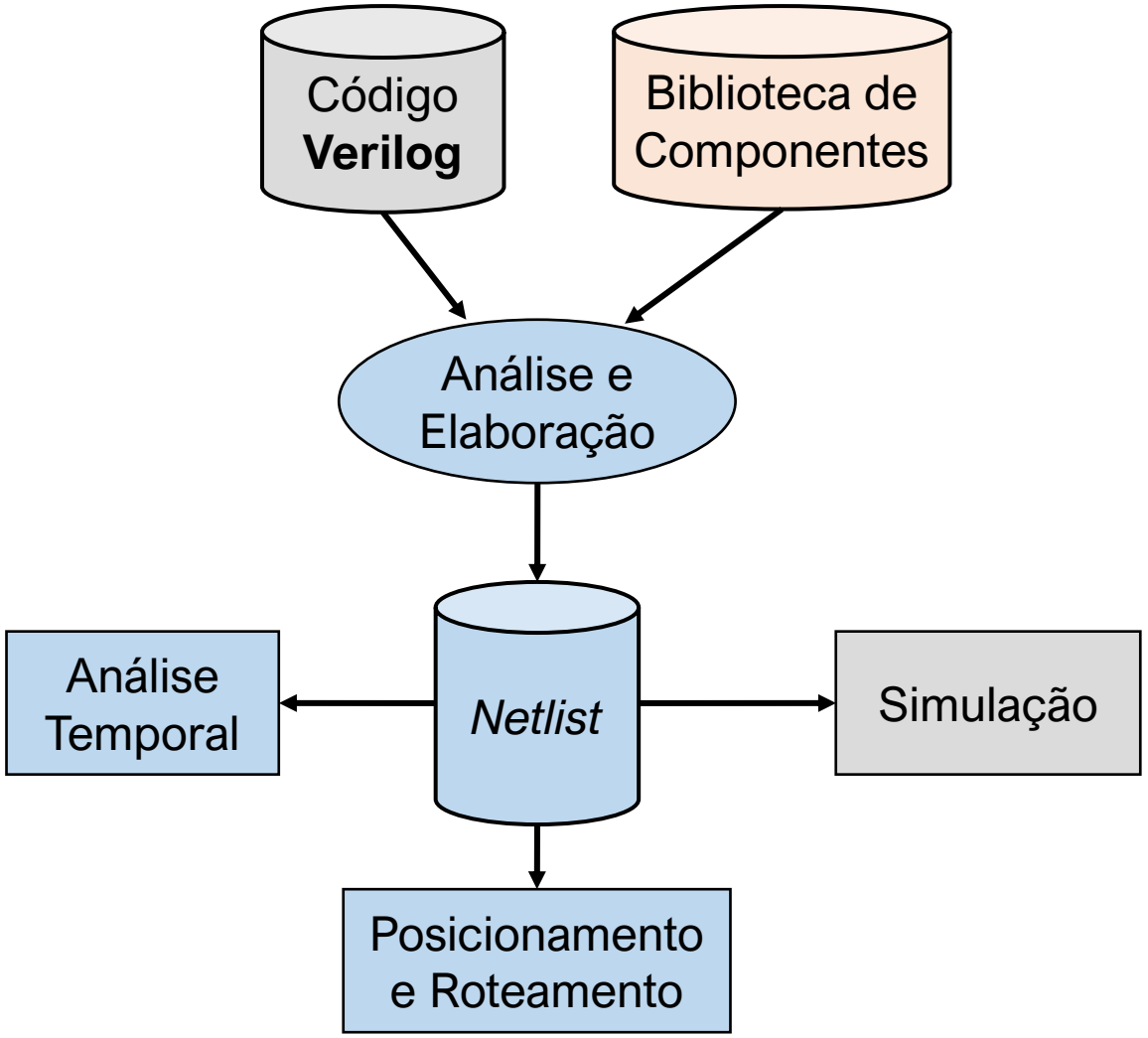
Linguagem de Descrição de *Hardware*

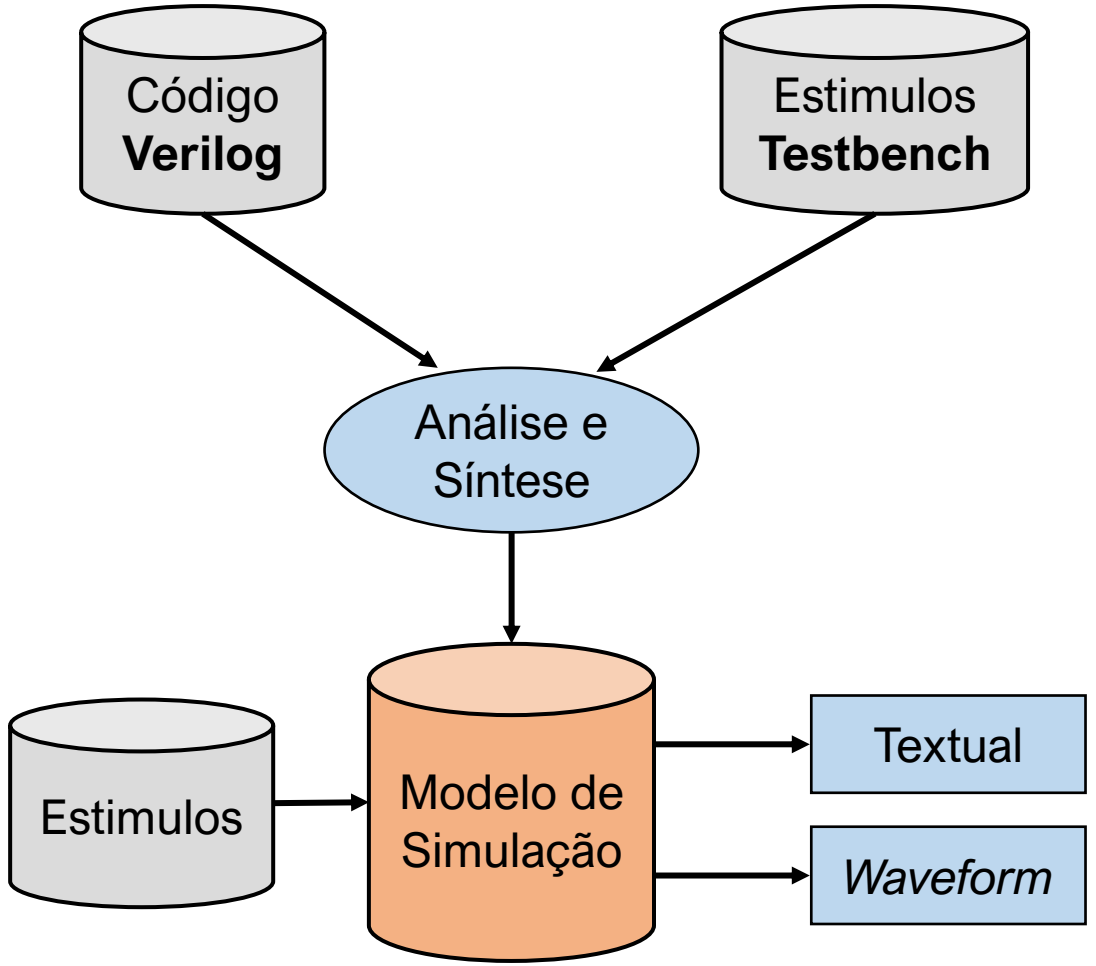
- Fluxo de Projeto:



Linguagem de Descrição de *Hardware*

- Fluxo de Projeto:





Estrutura Básica do Verilog

```
module nome_module(lista_portas);

    Declaração das Portas

    Dados intermediários

    Funcionamento do Circuito

endmodule
```

- Inicia com **`module`** e encerra com **`endmodule`**;
- *Case-sensitive*;
- Todas as palavras chaves são minúsculas;
- Espaços em branco não são interpretados;
- Todos os comandos terminam em ponto-e-vírgula;
- //: Comentário em uma única linha;
- /* */: Comentário em múltiplas linhas;
- Regras para os identificadores: deve sempre iniciar com uma letra ou com _ , jamais deve iniciar com um número.

Declaração do Modulo (*module*)

- Inicia com a palavra **`module`**;
- Em seguida, vem o nome do modulo;
- Após isso, a lista de portas (se necessário).

```
module nome_module(lista_portas);
```

Declaração das Portas

Dados intermediários

Funcionamento do Circuito

```
endmodule
```

Exemplo:

```
module halfadder (a, b, s, cout);
    (...)
endmodule
```

Declaração das Portas

- Comunicação do módulo com o mundo externo;
- Tipos de portas:
  - **input** → entrada;
  - **output** → saída;
  - **inout** → bidirecional.
- **Declaração de portas simples (1-*bit*):**

```
<tipo_da_porta> nome_da_porta;
```

```
module nome_module(lista_portas);
    Declaração das Portas
    Dados intermediários
    Funcionamento do Circuito
endmodule
```

Exemplo:

```
module halfadder (a, b, s, cout);
    input a, b;
    output s, cout;
endmodule
```

Declaração das Portas

- **Declaração de Barramentos (conjunto de *bits*):**

```
<tipo_da_porta> [MSB:LSB] nome_da_porta;
```

```
module nome_module(lista_portas);
    Declaração das Portas
    Dados intermediários
    Funcionamento do Circuito
endmodule
```

Exemplo:

```
module adder (a, b, cin, s, cout);
    input [3:0] a, b;
    input cin;
    output [3:0] s;
    output cout;
endmodule
```

Declaração das Portas

- **Definição de Constantes `(parameter)`:**
  - São valores numéricos que são associados a um nome;
  - Declaração:

```
parameter nome = valor;
```

  - Exemplos:

```
parameter barramento = 8;
```

  - Observação: A declaração de um `parameter` pode ser feita a qualquer momento no código Verilog;

Declaração das Portas

- **Definição de Constantes `(parameter)`:**
  - O uso de constantes pode tornar o código mais legível e mais fácil de atualizações;
  - Exemplo:

```
module adder (a, b, cin, s, cout);
    parameter largura = 4;
    input [largura - 1:0] a, b;
    input cin;
    output [largura - 1:0] s;
    output cout;
endmodule
```

Declaração do Módulo (Verilog-2001)

- Após o Verilog-2001, a declaração do módulo e das portas podem ser combinadas;
- Permite a declaração de forma mais concisa;
- Exemplo:

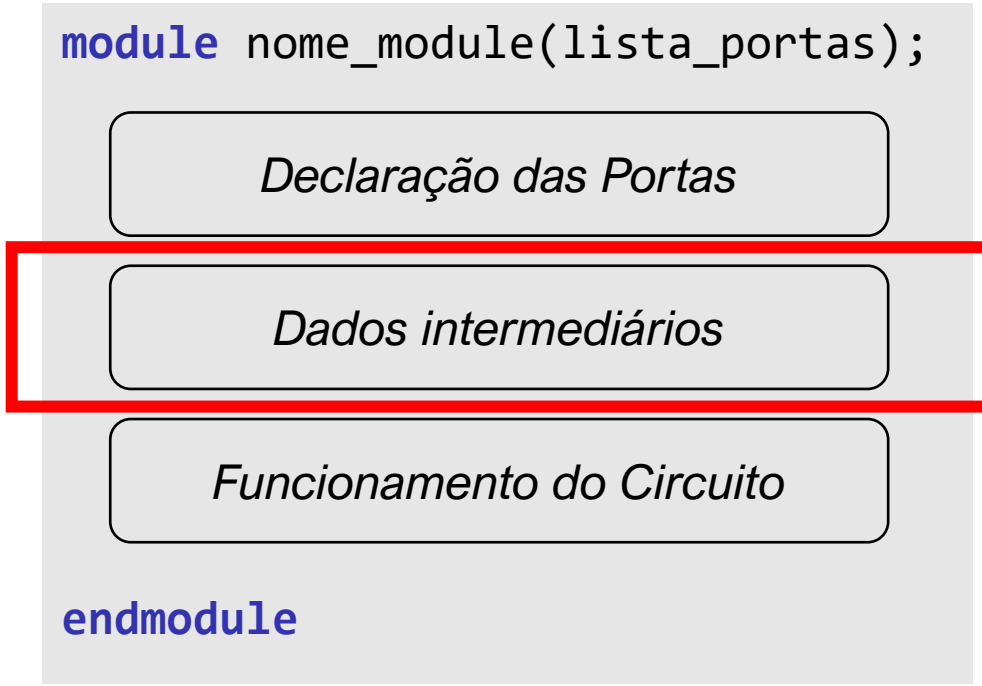
```
module adder
    #(parameter size = 8)
    (
    input [size - 1:0] a, b,
    input cin,
    output [size - 1:0] s,
    output cout
    );
endmodule
```

Separação por vírgulas (como se fosse o argumento de uma função).

A última linha não tem virgula.

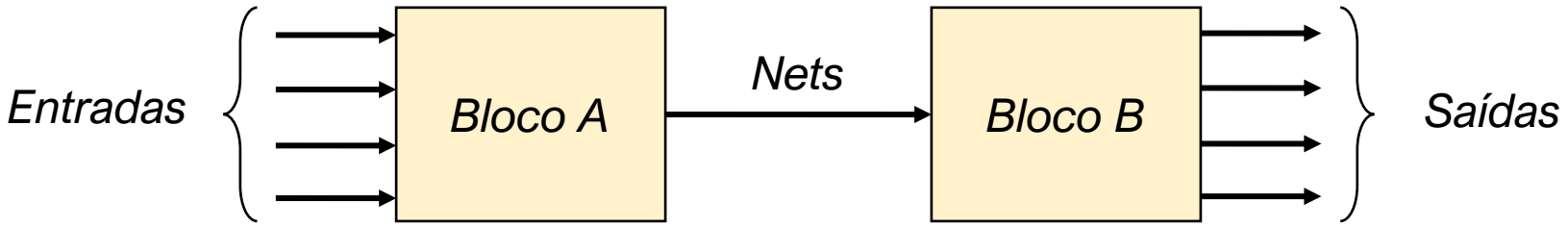
Dados Intermediários (*Data Type*)

- São dados internos ao módulo;
- Podem ser:
  - *Net Data Type*;
  - *Variable Data Type*.

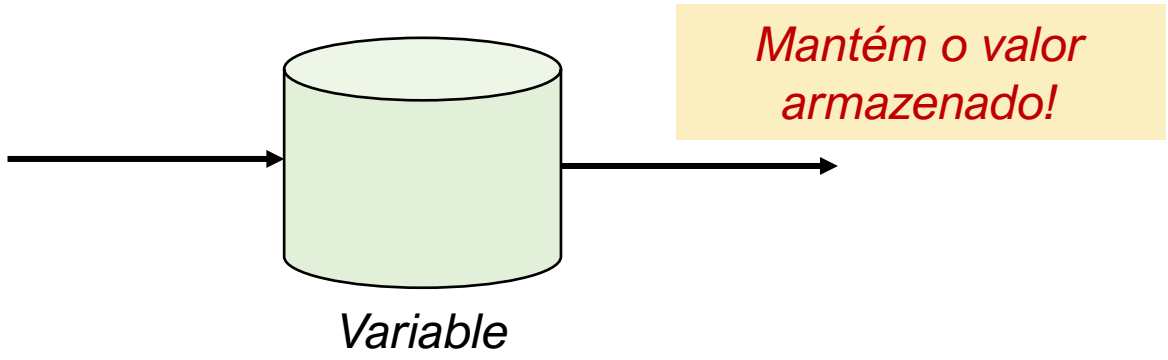


Dados Intermediários (*Data Type*)

- ***Net Data Type*:** Representam uma conexão física (fio) interno ao módulo;



- ***Variable Data Type*:** Representam um armazenamento local (registrador);



Dados Intermediários (*Data Type*)

- ***Net Data Type*:** Não apresentam capacidade de armazenamento;

Tipo	Definição
wire	Representa uma conexão (fio físico)
tri	Representa uma conexão *tri-state*
supply0 *supply1*	Nível lógico BAIXO Nível lógico ALTO

Exemplos:

```
wire ligação;
wire [7:0] soma;
tri barramento;
```

Observação: A declaração de Nets com mais de um bit é feita da mesma forma dos ports!

Dados Intermediários (*Data Type*)

- ***Variable Data Type*:** Apresentam capacidade de armazenamento;

Tipo	Definição
reg	Valor sem sinal
reg signed	Valor com sinal
integer	Inteiro com sinal de 32-*bits*
real, time, realtime	Não sintetizáveis pelo *Hardware*

Exemplos:

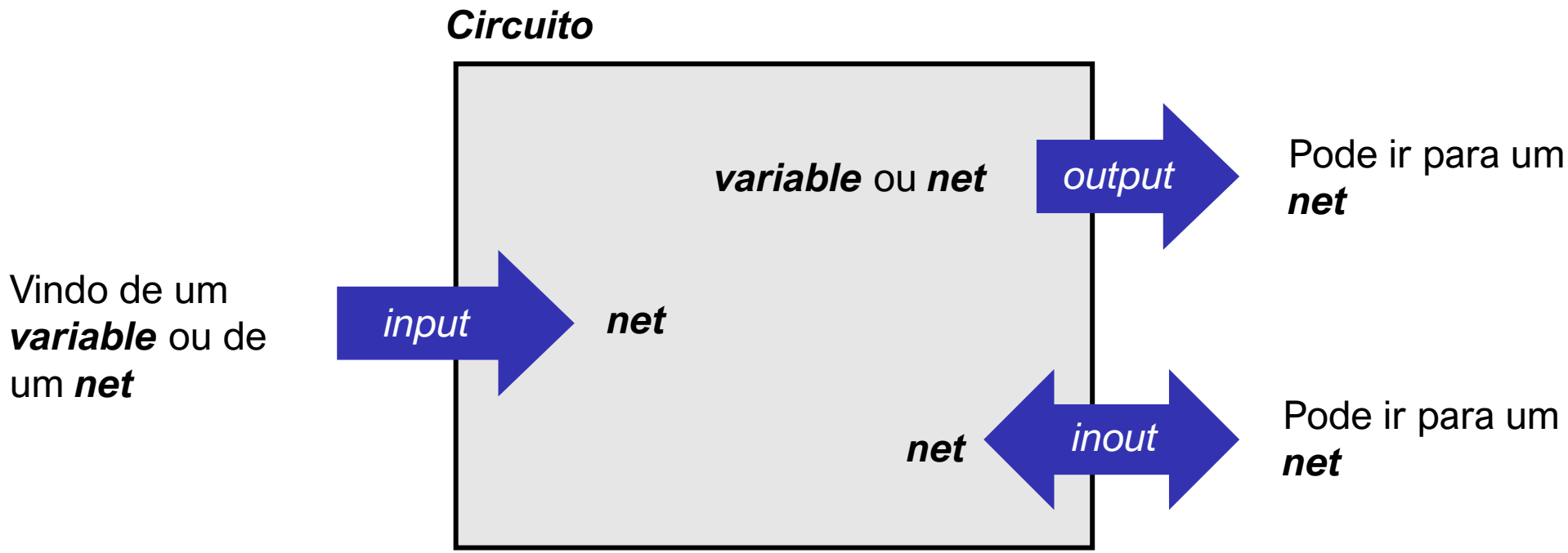
```
reg [7:0] resultado;
integer count;
```

Observação: A declaração de Register com mais de um bit é feita da mesma forma dos ports!

Dados Intermediários (*Data Type*)

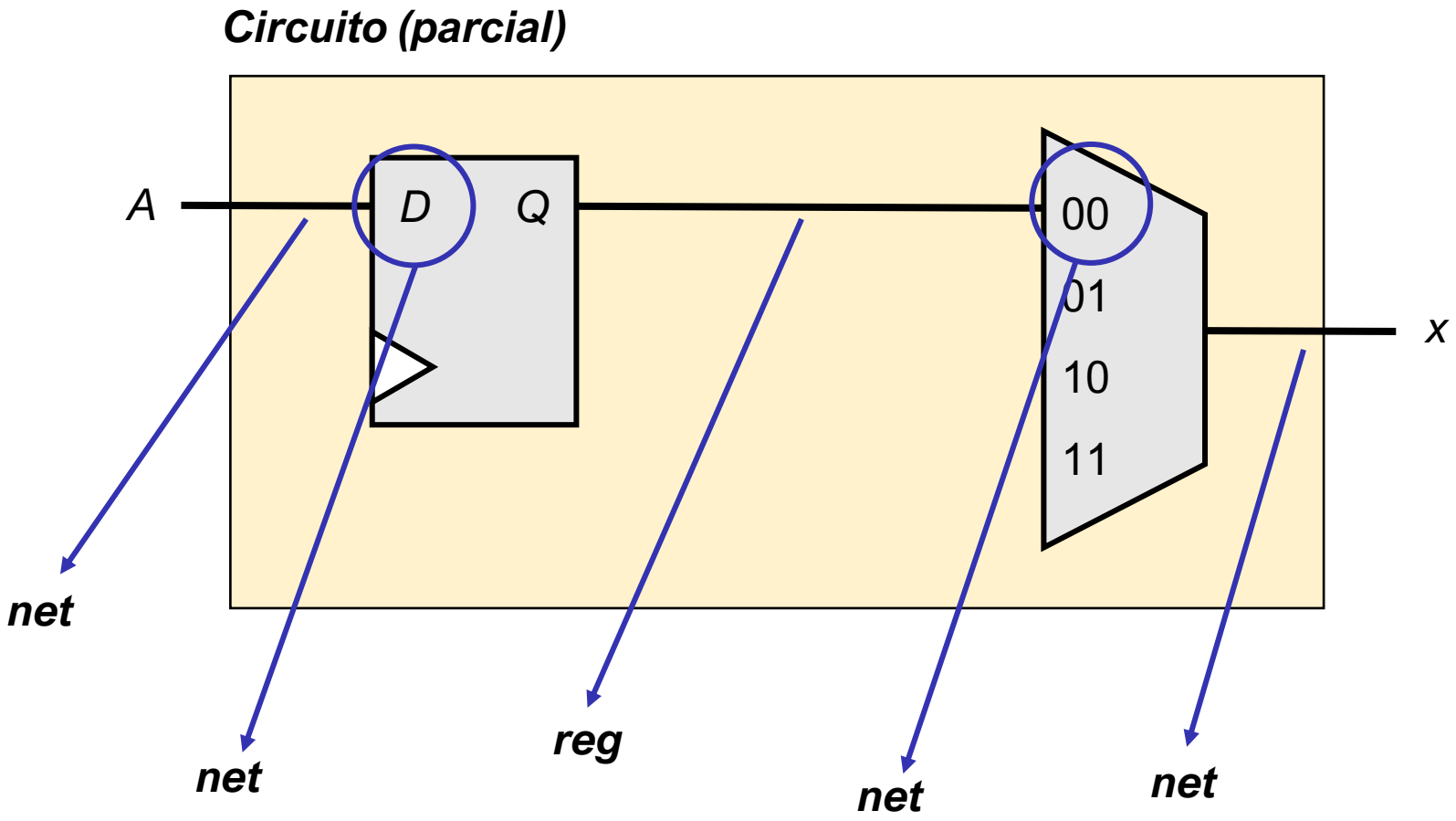
- **Regras para Entrada/Saída:**

Tipo de Variável	Entrada	Saída	IN/OUT
reg	NÃO	SIM	NÃO
wire	SIM	SIM	SIM



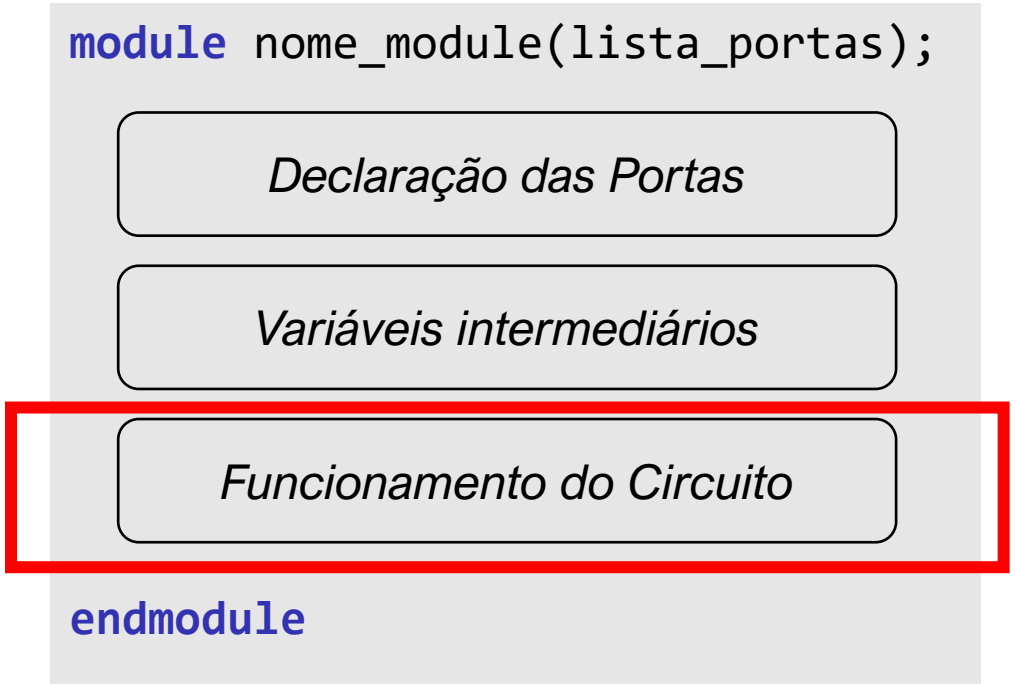
Dados Intermediários (*Data Type*)

- **Regras para Entrada/Saída:**
  - Exemplo:



Funcionamento do Circuito

- Descrição de como o circuito funciona;
- Abordagens:
  - Estrutural;
  - Fluxo de Dados (RTL);
  - Hierárquica;
  - Comportamental;



Descrição Estrutural

- Descreve o circuito em termos das portas lógicas, blocos e suas interconexões
- Utiliza primitivas para construir o comportamento do circuito;
- Blocos elementares (primitivas) → portas lógicas:

Tipo de Primitivas	Primitivas existentes
Portas multientradas	*and*, *or*, *nand*, *nor*, *xor* e *xnor*
Portas com uma única entrada	*not* e *buf*
Portas *tri-state*	*bufif0*, *bufif1*, *notif0* e *notif1*

Descrição Estrutural

- Para fazer um circuito utilizando as primitivas, basta referenciar a porta lógica e indicar as ligações;

```
<primitiva> nome (ligacao);
```

  - **<primitiva>**: *and*, *or*, *nand*, *nor*, *xor, xnor, not, buf, bufif0*, *bufif1*, *notif0* ou *notif1;*
  - **nome**: Recomendável, mas não obrigatório;
  - **ligacao**: como os pinos da primitiva estão ligados, separados entre virgulas;
    - O primeiro item é sempre a saída;
    - Os itens seguintes são as entradas;

Descrição Estrutural

- Exemplos:

```
and u0 (x, a, b);
```

- *Porta lógica and;*
- *Saída: x;*
- *Entradas: a e b;*
- *Nome: u0;*

```
not u1 (x, a);
```

- *Porta lógica not;*
- *Saída: x;*
- *Entradas: a;*
- *Nome: u1;*

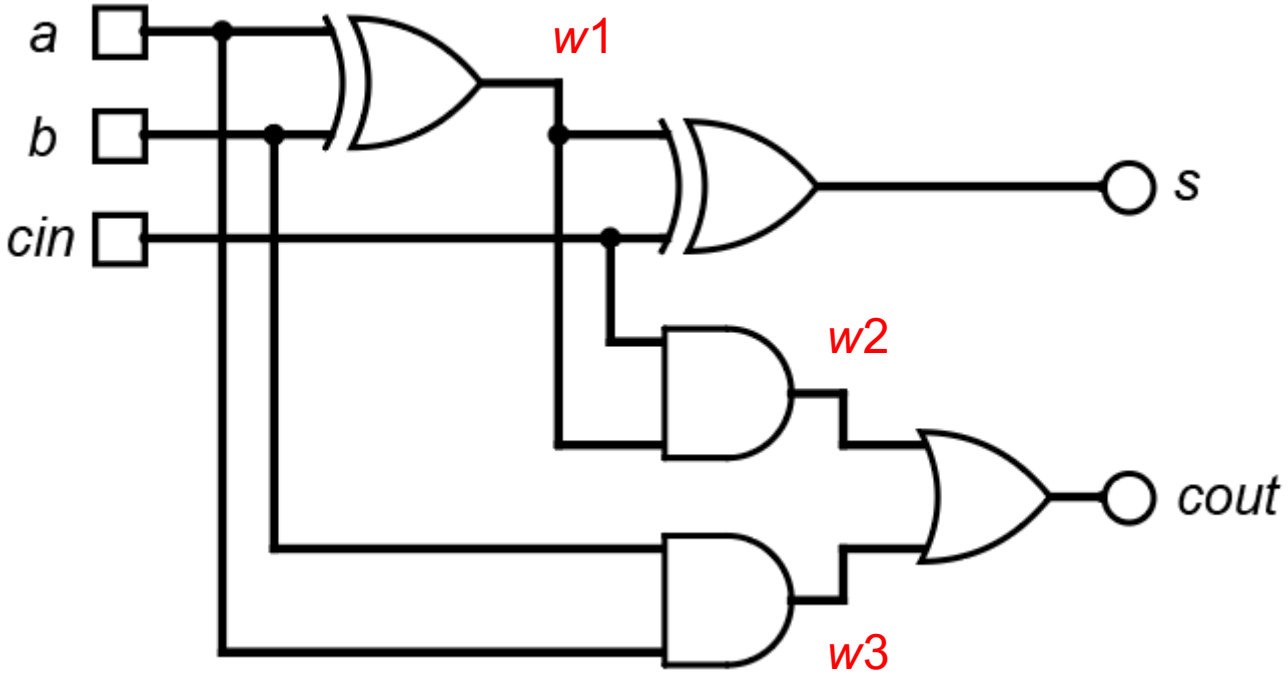
Descrição Estrutural

- Construir o circuito utilizando abordagem estrutural é tão só:
  - Referenciar as primitivas adequadas;
  - Realizar a ligação entre elas.
- Para se ligar a saída de uma primitiva para entrada de outra, deve-se declarar um fio intermediário (`wire`);

Descrição Estrutural

Projeto 01 – O primeiro (de muitos) circuito em Verilog

Descreva o circuito corresponde ao somador completo em Verilog. O diagrama do circuito é apresentado na Figura abaixo.



Referências

- **PIMENTA, TALES CLEBER – Circuitos Digitais – Análise e Síntese Lógica: Aplicações em FPGA. Elsevier, 2017.**
- Altera’s Verilog HDL Basics (https://www.youtube.com/watch?v=PJGvZSIsLKs)
- VAHID, FRANK - Sistemas Digitais - Projeto, Otimização e HDLs. Bookman, 2008.