UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO (UFERSA)
CENTRO MULTIDISCIPLINAR DE PAU DOS FERROS (CMPF)
DEPARTAMENTO DE ENGENHARIAS E TECNOLOGIA (DETEC)

CAPACITA UFERSA

Introdução ao VHDL

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/SystemVerilog;
  - System C;
- Dentre as linguagens de descrição de hardware destaca-se o VHDL;

Linguagem de Descrição de *Hardware*

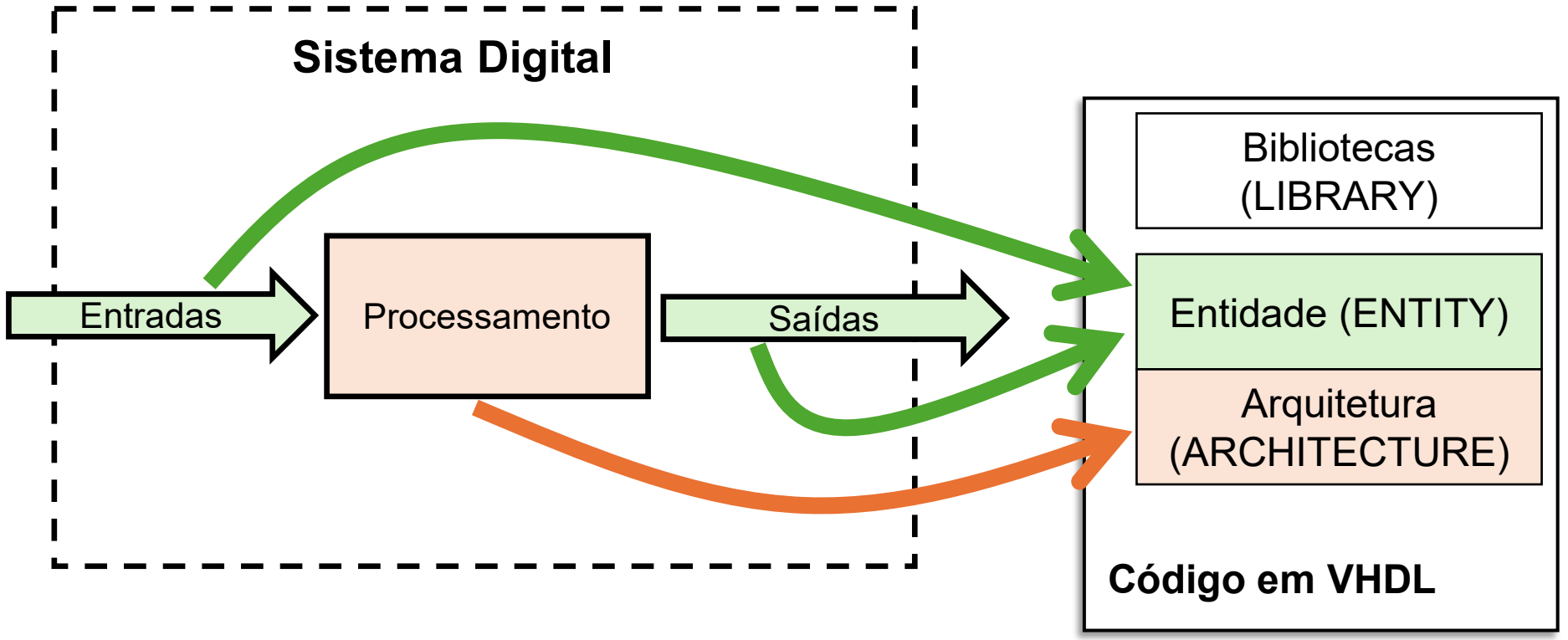
- **Vantagens na utilização do HDL:**
  - Projetos independentes da tecnologia (CPLD/FPGA);
    - Portabilidade;
  - Facilidade de atualização dos projetos;
  - Projeto em um nível mais alto de abstração;
  - Redução do tempo de projeto, de testes e implementação;
  - Simplificação quanto a sua documentação;

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

Estrutura do Código VHDL

- **Não** é *case-sensitive*;
- Espaços em branco não são interpretados;
- --: Comentário em uma linha;
- As regras para formação de identificadores são:
  - Só é permitido letras, números e *underline* (_);
  - O primeiro caractere deve ser uma letra;
  - O último não pode ser *underline*;
  - Não são permitidos 2 *underline* em sequência;
  - Letras maiúscula e minúscula são equivalentes (não é *case-sensitive*);

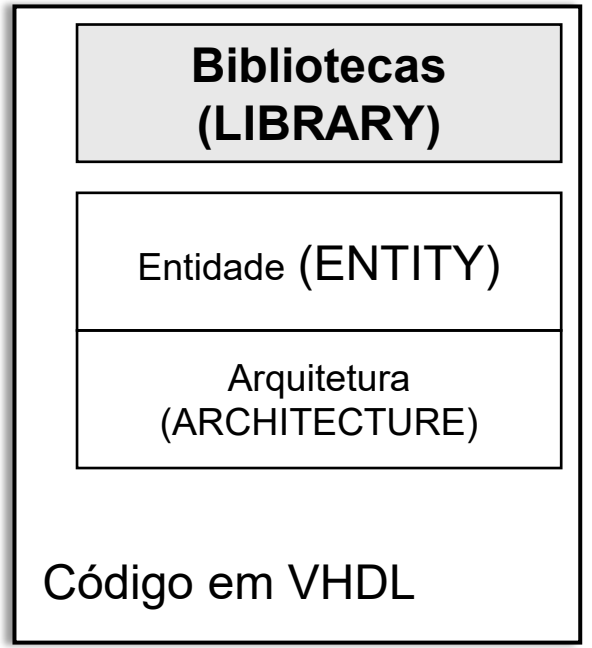
Estrutura do Código VHDL



Estrutura do Código VHDL

- **Bibliotecas (LIBRARY):**
  - Trecho **eletivo** que informa que funções extras serão utilizadas no código VHDL;
  - Declaração:

```
LIBRARY library_name;
USE library_name.package_name.package_parts;
```



Estrutura do Código VHDL

Biblioteca	Pacote	Uso típico
std	*standard*	Define o padrão BIT e BIT_VECTOR, que admite apenas valores ALTO e BAIXO.
	textio	Trabalhar com textos e arquivos
	env	Para comunicação com a simulação

Estrutura do Código VHDL

Biblioteca	Pacote	Uso típico
ieee	*std_logic_1164*	Define os padrões STD_LOGIC, STD_LOGIC_VECTOR, que além dos níveis ALTO e BAIXO, admite também *don't care*, alta impedância, entre outros.
	numeric_bit	Para implementar circuitos aritméticos inteiros sem sinal ou com sinal; tem BIT como tipo base.
	numeric_std	Mesmo que o NUMERIC_BIT, porém tendo o STD_LOGIC como base. Utiliza tipos UNSIGNED e SIGNED.
	numeric_std_unsigned	Permite trabalhar com STD_LOGIC_VECTOR como UNSIGNED.

Estrutura do Código VHDL

- **Bibliotecas (LIBRARY):**
  - Pacotes não padronizadas:

Biblioteca	Pacote	Uso típico
ieee	*std_logic_arith*	Especifica os tipos UNSIGNED e SIGNED
	std_logic_signed	Realiza operações aritméticas sobre dados do tipo SIGNED. Preferir a NUMERIC_STD, se disponível.
	std_logic_unsigned	Realiza operações aritméticas sobre dados do tipo UNSIGNED. Preferir a NUMERIC_STD, se disponível.

  - Exemplos de declaração:

```
LIBRARY ieee;                    -- O ponto-e-virgula (;) indica
USE ieee.std_logic_1164.all;     -- o fim de uma estrutura
```

Estrutura do Código VHDL

- **Tipos no VHDL:**
  - O VHDL é fortemente tipada, portanto é necessário definir os tipos dos identificadores;

Biblioteca	Pacote	Tipo	Comentário
std	*standard*	**BIT**	**0, 1**
		BIT_VECTOR	**Vetor de tipo BIT**
		BOOLEAN	true,false
		BOOLEAN_VECTOR	Vetor de tipo BOOLEAN
		INTEGER	Inteiro 32-bits com sinal
		INTEGER_VECTOR	Vetor de INTEGER
		NATURAL	Inteiro positivos
		POSITIVE	Inteiros > 0
		CHARACTER	Caractere (ISO 8859-1)
		STRING	Vetor de caracteres
ieee	*std_logic_1164*	STD_ULOGIC	1, 0, Z, W, L, H, -, U, X
		STD_ULOGIC_VECTOR	Vetor de STD_ULOGIC
		STD_LOGIC	**STD_ULOGIC resolvido**
		STD_LOGIC_VECTOR	**Vetor de STD_LOGIC**
	numeric_std *numeric_std_unsigned*	**UNSIGNED**	**Inteiro sem sinal**
		SIGNED	**Inteiro com sinal**

Estrutura do Código VHDL

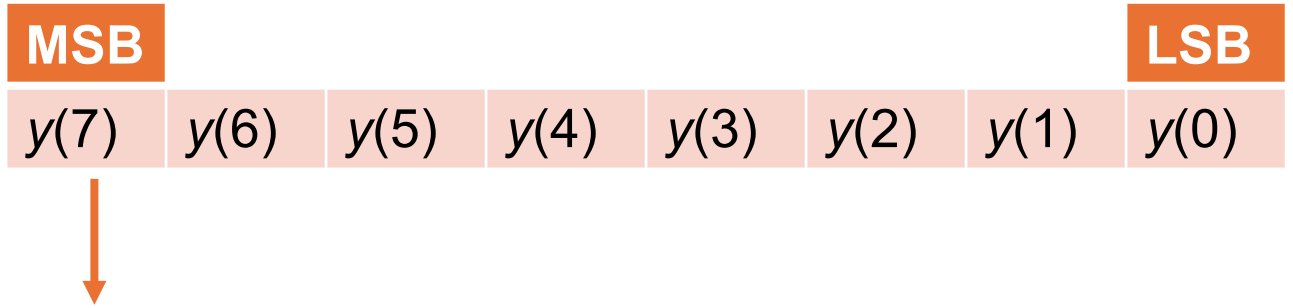
- **Tipos no VHDL:**
  - Conversões entre tipos:

Tipo de dado	Para o tipo de dado	Função de conversão
unsigned, signed	*std_logic_vector*	`std_logic_vector(a)`
signed, std_logic_vector	*unsigned*	`unsigned(a)`
unsigned, std_logic_vector	*signed*	`signed(a)`
unsigned, signed	*integer*	`to_integer(a)`
natural	*unsigned*	`to_unsigned(a, size)`
integer	*signed*	`to_signed(a, size)`

Estrutura do Código VHDL

- **Tipos no VHDL:**
  - Declaração de Barramentos:

```
STD_LOGIC_VECTOR (7 DOWNTO 0);
```

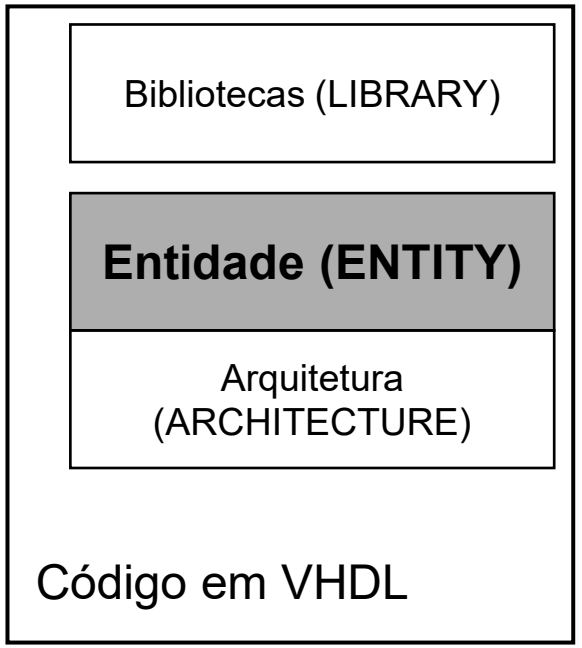


Cada posição corresponde a um tipo STD_LOGIC.

Estrutura do Código VHDL

- **Entidade (ENTITY):**
  - Bloco **obrigatório** que descreve as interfaces de entrada e saída do sistema digital;
  - Como prática, se designa o nome do arquivo VHDL com o mesmo nome da entidade;
  - Utilização da diretiva **PORT**;

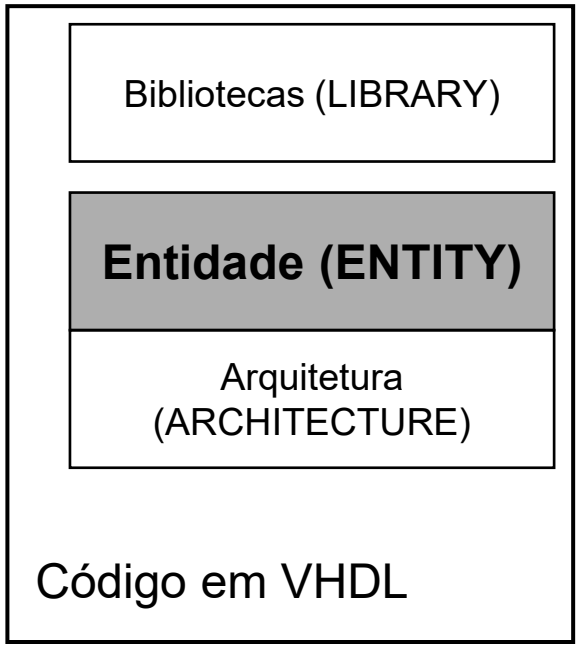
```
ENTITY nome_da_entidade IS
    PORT (
        nome_da_porta : modo_de_operação tipo_da_porta;
        nome_da_porta : modo_de_operação tipo_da_porta;
        ...);
END nome_da_entidade;
```



Estrutura do Código VHDL

- **Entidade (ENTITY):**
  - Nome da entidade: deve seguir as regras dos identificadores.

```
ENTITY nome_da_entidade IS
    PORT (
        nome_da_porta : modo_de_operação tipo_da_porta;
        nome_da_porta : modo_de_operação tipo_da_porta;
        ...);
END nome_da_entidade;
```



Estrutura do Código VHDL

- **Entidade (ENTITY):**
  - Modo de operação:

```
ENTITY nome_da_entidade IS
    PORT (
        nome_da_porta : modo_de_operação tipo_da_porta;
        nome_da_porta : modo_de_operação tipo_da_porta;
        ...);
END nome_da_entidade;
```

Modo	Descrição
IN	Entrada.
OUT	Saída.
INOUT	Bidirecional (Entrada e Saída).

Estrutura do Código VHDL

- **Entidade (ENTITY):**
  - Tipo da porta:

```
ENTITY nome_da_entidade IS
    PORT (
        nome_da_porta : modo_de_operacao tipo_da_porta;
        nome_da_porta : modo_de_operacao tipo_da_porta;
        ...);
END nome_da_entidade;
```

Tipo da Porta	Comentários	Biblioteca
BIT	Valores lógicos 0 ou 1.	
BIT_VECTOR	Coleção do tipo BIT.	
STD_LOGIC	Define outros tipos além do “0” e “1”, como don´t care (X) e alta impedância (Z).	*ieee.std_logic_1164*

Estrutura do Código VHDL

- **Entidade (ENTITY):**
  - Tipo da porta:

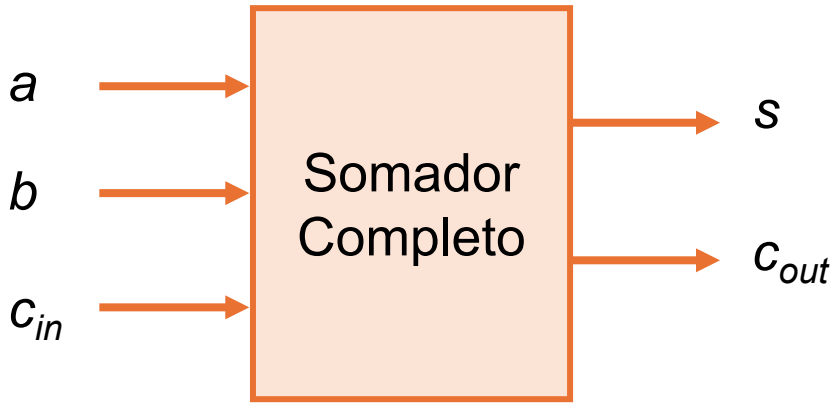
```
ENTITY nome_da_entidade IS
    PORT (
        nome_da_porta : modo_de_operacao tipo_da_porta;
        nome_da_porta : modo_de_operacao tipo_da_porta;
        ...);
END nome_da_entidade;
```

Tipo da Porta	Comentários	Biblioteca
STD_LOGIC_VECTOR	Coleção do tipo STD_LOGIC.	*ieee.std_logic_1164*
UNSIGNED	Número inteiro sem sinal.	*ieee.std_logic_1164* + *ieee.numeric_std*
SIGNED	Número inteiro com sinal.	*ieee.std_logic_1164* + *ieee.numeric_std*

Estrutura do Código VHDL

Exemplo: Entidade para um somador completo.

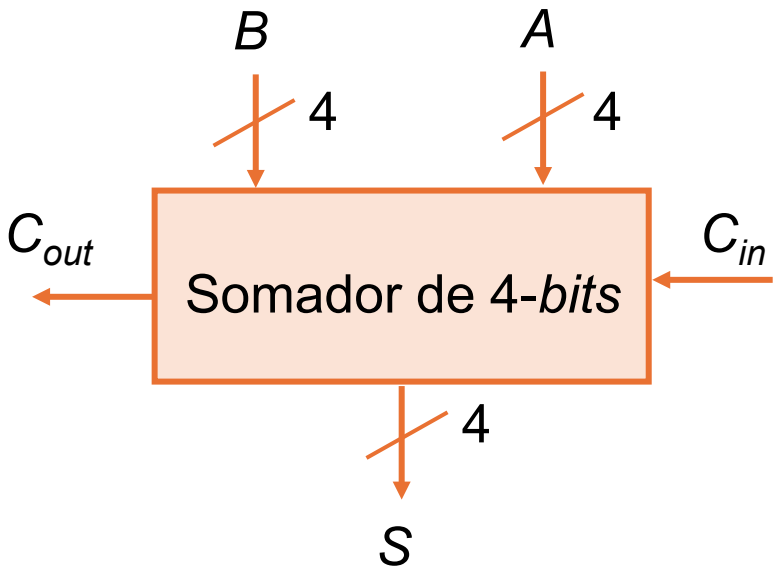
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY somadorcompleto IS
    PORT (
        a, b, cin: IN STD_LOGIC;
        s, cout: OUT STD_LOGIC
        );
END somadorcompleto;
```



Estrutura do Código VHDL

Exemplo: Entidade para um somador de 4 *bits*.

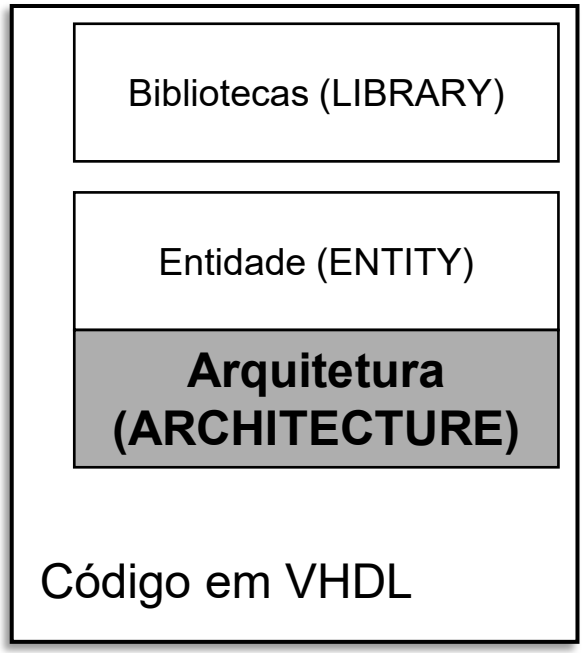
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY somador4b IS
    PORT (
        A, B: IN STD_LOGIC_VECTOR (3 DOWNTO 0);
        S: OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
        Cin: IN STD_LOGIC;
        Cout: OUT STD_LOGIC
        );
END somador4b;
```



Estrutura do Código VHDL

- **Arquitetura (ARCHITECTURE):**
  - Bloco **obrigatório** que descreve o funcionamento do sistema digital propriamente dito;
    - As relações entradas/saídas são especificadas na arquitetura.

```
ARCHITECTURE nome_arquitetura OF nome_entidade IS
    [pre-ambulo]
BEGIN
    (código)
END nome_arquitetura;
```



Estrutura do Código VHDL

- **Arquitetura (ARCHITECTURE):**
  - Abordagens possíveis para o VHDL:
    - Concorrente ↔ Lógica Combinacional;
      - Forma padrão do VHDL;
    - Sequencial ↔ Lógica Sequêncial/Combinacional;
      - Ativo somente com o uso das diretivas **PROCESS**, **FUNCTION**, ou **PROCEDURE**.
    - Estrutural ↔ Constrói sistemas maiores utilizando blocos menores;

Abordagem Concorrente

- Na Abordagem Concorrente (Padrão):
  - Instruções/comandos processados em paralelo;
  - Elementos:
    - Operadores;
      - Lógicos;
      - Aritméticos;
      - Concatenação;
      - Comparação;
      - Deslocamento.
    - Estruturas de seleção;

Operadores do VHDL

- **Operadores Lógicos:**

Operadores Regulares	
and	Operador and regular
nand	Operador nand regular
or	Operador or regular
nor	Operador nor regular
xor	Operador xor regular
xnor	Operador xnor regular

Operadores Unários	
not	Operador not
and	Operador and de redução
nand	Operador nand de redução
or	Operador or de redução
nor	Operador nor de redução
xor	Operador xor de redução
xnor	Operador xnor de redução

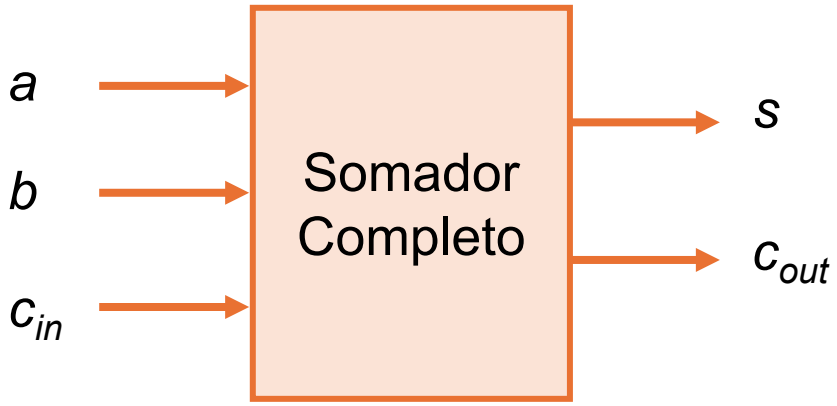
- Exemplos:

```
'1' AND '1' = '1'
'0' XOR '1' = '1'
"0011" XOR "0101" = "0110"
'1' AND "0011" = ('1'AND'0')('1'AND'0')('1'AND'1') ('1'AND'1') = "0011"
AND "1110" = '1' AND '1' AND '1' AND '0' = '0'
OR "1110" = '1' OR '1' OR '1' OR '0' = '1'
```

Operadores do VHDL

Projeto 1 – Somador Completo

Descreva o circuito corresponde ao somador completo em VHDL utilizando a abordagem por *dataflow*.



$$s = a \oplus b \oplus c_{in}$$
$$c_{out} = ab + ac_{in} + bc_{in}$$

Operadores do VHDL

- **Operadores Aritméticos:**

Operadores Regulares	
+	Soma
-	Subtração
*	Multiplicação
/	Divisão
**	Potenciação
rem	Resto da divisão
mod	Operação módulo

Operadores Unários	
-	Negação
abs	Módulo

- Os operandos podem ser:
  - SIGNED, UNSIGNED: inteiros com ou sem sinal. Necessita da ***numeric_std*** ou ***std_logic_arith***.
  - STD_LOGIC_VECTOR: Necessita da biblioteca ***std_logic_unsigned*** ou ***std_logic_signed***.
  - INTEGER, NATURAL, POSITIVE, FLOAT, UFIXED, SFIXED.

Operadores do VHDL

- **Operador de Concatenação:**
  - Servem para juntar os operandos e formar um novo vetor;
  - Exemplo:

```
Z_BUS <= A_BIT & B_BIT & C_BIT & D_BIT;
```

Operadores do VHDL

- **Operadores de Comparação:**

Operadores Regulares	
=	Igualdade
/=	Desigualdade
<	Menor que
<=	Menor ou igual que
>	Maior que
>=	Maior ou igual que
minumum(), maximum()	

Retornam um BOOLEAN. Podem ser utilizados em estruturas de seleção.

Operadores de Casamento	
?=	Igualdade
?/=	Desigualdade
?<	Menor que
?<=	Menor ou igual que
?>	Maior que
?>=	Maior ou igual que

Retornam um STD_LOGIC. Podem ser utilizados como saída. Requer a biblioteca *ieee.numeric_std_unsigned*

Operadores do VHDL

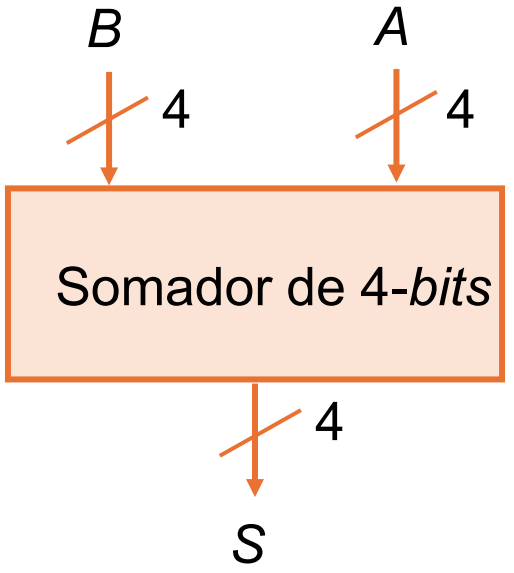
- **Operadores de Deslocamento:**
  - Deslocamentos à esquerda (lógico ou aritméticos): Posições vagas são preenchidas com zero;
  - Deslocamentos à direita:
    - Lógico: Posições vagas são preenchidas com zero;
    - Aritmético: Posições vagas são preenchidas com o MSB.
  - *Bits* deslocados são perdidos.

Operadores Regulares	
SLL	Deslocamento lógico à esquerda
SRL	Deslocamento lógico à direita
SLA	Deslocamento aritmético à esquerda
SRA	Deslocamento aritmético à direita
ROR	Rotacionar à direita
ROL	Rotacionar à esquerda

Operadores do VHDL

Projeto 2 – Somador de 4-bits

Projete um somador de 4-*bits.* Implemente o somador com as operações aritméticas. Considere que os dados não possuem sinal.



Abordagem Concorrente

- Principais declarações concorrentes:
  - WHEN/ELSE;
  - WITH/SELECT/WHEN;

- **Diretiva WHEN/ELSE:**

```
target <= value WHEN condition ELSE
          value WHEN condition ELSE
               (…)
          value;
```

- **Diretiva WITH/SELECT/WHEN:**

```
WITH expression SELECT
    target <= value WHEN condition,
              value WHEN condition,
                 (…)
              value WHEN OTHERS;
```

Abordagem Concorrente

Projeto 3 – Unidade Lógica-Aritmética

Implemente uma unidade lógico-aritmética de 8-*bits* que possua a tabela de operação apresentada ao lado.

Seleção			Operação
x	*y*	*z*	
0	0	0	$S = A + B$
0	0	1	$S = A - B$
0	1	0	$S = A + 1$
0	1	1	$S = A$
1	0	0	$S = A$ **and** B
1	0	1	$S = A$ **or** B
1	1	0	$S = A$ **xor** B
1	1	1	$S =$ **not** A

Abordagem Estrutural

- Pode-se utilizar componentes menores, já definidos, para se construir componentes (projetos) maiores;
- Utilização da diretivas **COMPONENT** e **PORT MAP** e uso de **SIGNAL**.

Abordagem Estrutural

- **SIGNAL:**
  - Variáveis intermediárias que podem assumir valores de outros circuitos;
  - Equivalente a um fio em uma ligação física;
  - Os sinais são utilizados nas seguintes premissas:
    - Quando se deseja levar uma informação de um componente a outro ou;
    - Quando se deseja atribuir valores intermediários resultantes de expressões lógicas.
  - Os **SIGNALS** são declarados no preambulo da arquitetura.
  - Declaração:

```
SIGNAL signal_name: signal_type [range] [:= default_value];
```

Abordagem Estrutural

- **COMPONENT:**
  - Blocos elementares de lógica que podem ser ligados entre si para gerar blocos maiores;
    - Os blocos devem estar definidos no projeto VHDL;

```
COMPONENT nome_do_componente IS
PORT (
        nome_da_porta : modo_de_operação tipo_da_porta;
        nome_da_porta : modo_de_operação tipo_da_porta;
    ...);
END COMPONENT;
```

Abordagem Estrutural

- Após invocar os componentes, é necessário instancia-los no código.
- Forma de instanciar:

```
label: [COMPONENT] component_name PORT MAP (port_list);
```

- Tipos de Associação: forma que as entradas e saídas estão ligadas;
  - Lista;
    - As entradas/saídas são colocadas na ordem no qual foram declaradas na entidade;
    - A ordem importa;
  - Nome;
    - As entradas/saídas são associadas a pinos específicos da entidade usando o operador =>;
    - A ordem não importa;

Abordagem Estrutural

- Exemplo:

```
COMPONENT nand3 IS
    PORT (
        a1, a2, a3: IN STD_LOGIC;
        b: OUT STD_LOGIC
    );
END COMPONENT;
```

```
nand3_1: nand3 PORT MAP (x1, x2, x3, y);
nand3_2: nand3 PORT MAP (a1=>x1, a2=>x2, a3=>x3, b=>y);
nand3_3: nand3 PORT MAP (x1, x2, x3, OPEN);
nand3_4: nand3 PORT MAP (a1=>x1, a2=>x2, a3=>x3, b=>OPEN);
```

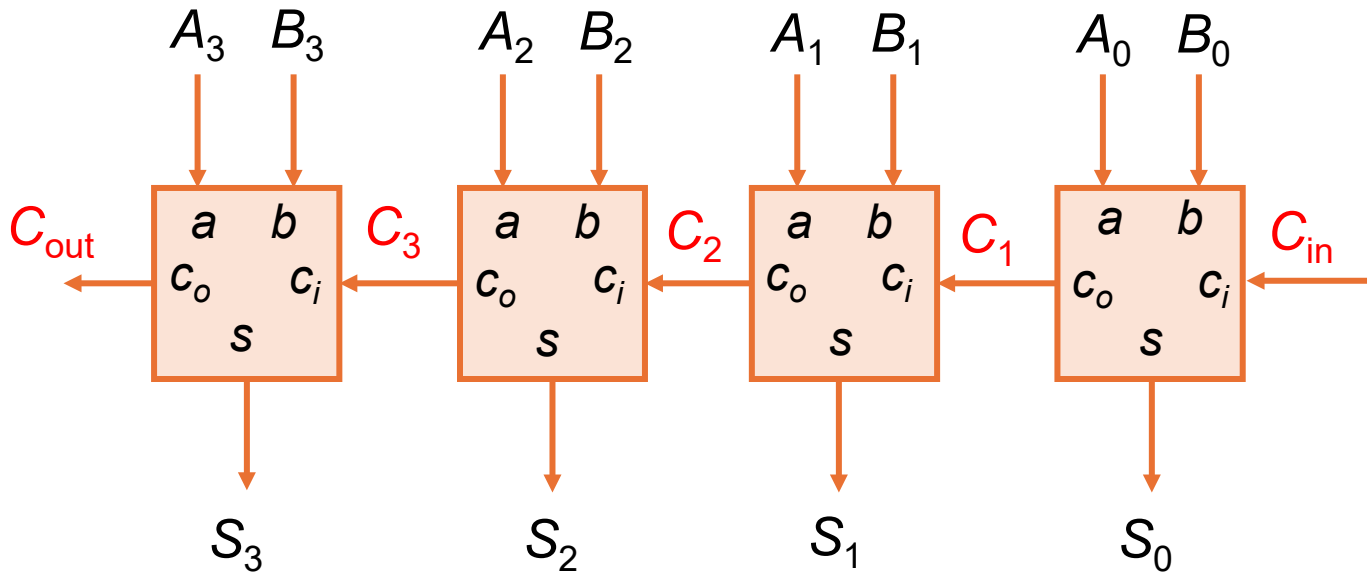
Abordagem Estrutural

Projeto 4 – Somador de 4-bits

Projete um somador de 4-*bits* em VHDL. Para isso:

a) Inicie o seu projeto com um somador completo.

b) Utilizando o somador completo do item (b), implemente o somador de 4-*bits*.



Abordagem Sequencial

- Instruções/comandos são processados sequencialmente;
- Como a abordagem concorrente é a padrão, trechos de código sequenciais precisam ser especificados no código VHDL por meio da diretivas:
  - **PROCESS**;
  - FUNCTION ou
  - PROCEDURE.

Abordagem Sequencial

- Diretiva **PROCESS**:

```
[label:] PROCESS (lista_de_sensibilidade)
    [VARIABLE (nome) (tipo) [range] [:= valor_inicial;]]
BEGIN
    (código)
END PROCESS [label];
```

Abordagem Sequencial

- Diretiva **PROCESS**:

```
[label:] PROCESS (lista_de_sensibilidade)
    [VARIABLE (nome) (tipo) [range] [:= valor_inicial;]]
BEGIN
    (código)
END PROCESS [label];
```

  - Lista de Sensibilidade:
    - Variáveis no qual o PROCESS será executado, caso mudem.

Abordagem Sequencial

- Diretiva **PROCESS**:

```
[label:] PROCESS (lista_de_sensibilidade)
    [VARIABLE (nome) (tipo) [range] [:= valor_inicial;]]
BEGIN
    (código)
END PROCESS [label];
```

  - **VARIABLE:**
    - São objetos de dados usados para armazenar valores intermediários ou temporários, em estruturas sequenciais do VHDL

Abordagem Sequencial

- Diretiva **PROCESS**:
  - Diferença do **SIGNAL** para **VARIABLE**:
    - **VARIABLE** pode ser usado apenas em procedimento. **SIGNAL** podem ser usado tanto em procedimentos, como fora deles.
    - O **VARIABLE** é atualizado imediatamente, já o **SIGNAL** é atualizado a depender se o circuito é combinacional ou sequencial.
    - **VARIABLE** é local para cada procedimento.
    - O **VARIABLE** utiliza o símbolo de atribuição :=

Abordagem Sequencial

- Diretiva **PROCESS**:

```
[label:] PROCESS (lista_de_sensibilidade)
    [VARIABLE (nome) (tipo) [range] [:= valor_inicial;]]
BEGIN
    (código)
END PROCESS [label];
```

  - Código:
    - Operações lógicas, aritméticas, de atribuição;
    - Diretivas de seleção:
      - IF/ELSIF/ELSE;
      - CASE;

Abordagem Sequencial

- Diretiva **CASE**:

```
CASE (identificador) IS
    WHEN (valor) => (ações);
    WHEN (valor) => (ações);
    ...
END CASE;
```

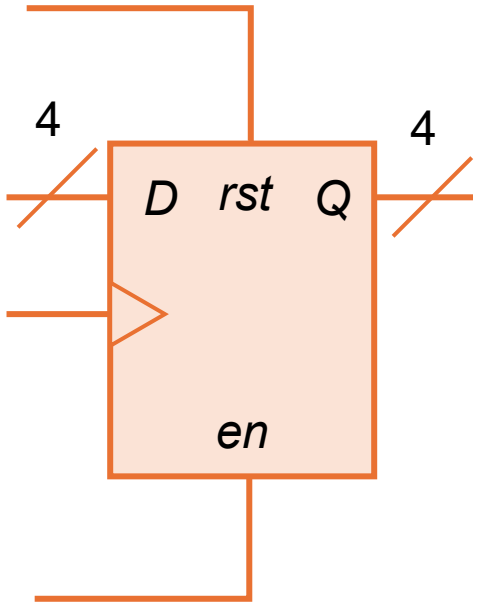
- Diretiva **IF/ELSIF/ELSE**:

```
IF (condição) THEN (ações);
ELSIF (condição) THEN (ações);
...
ELSE (ações);
END IF;
```

Abordagem Sequencial

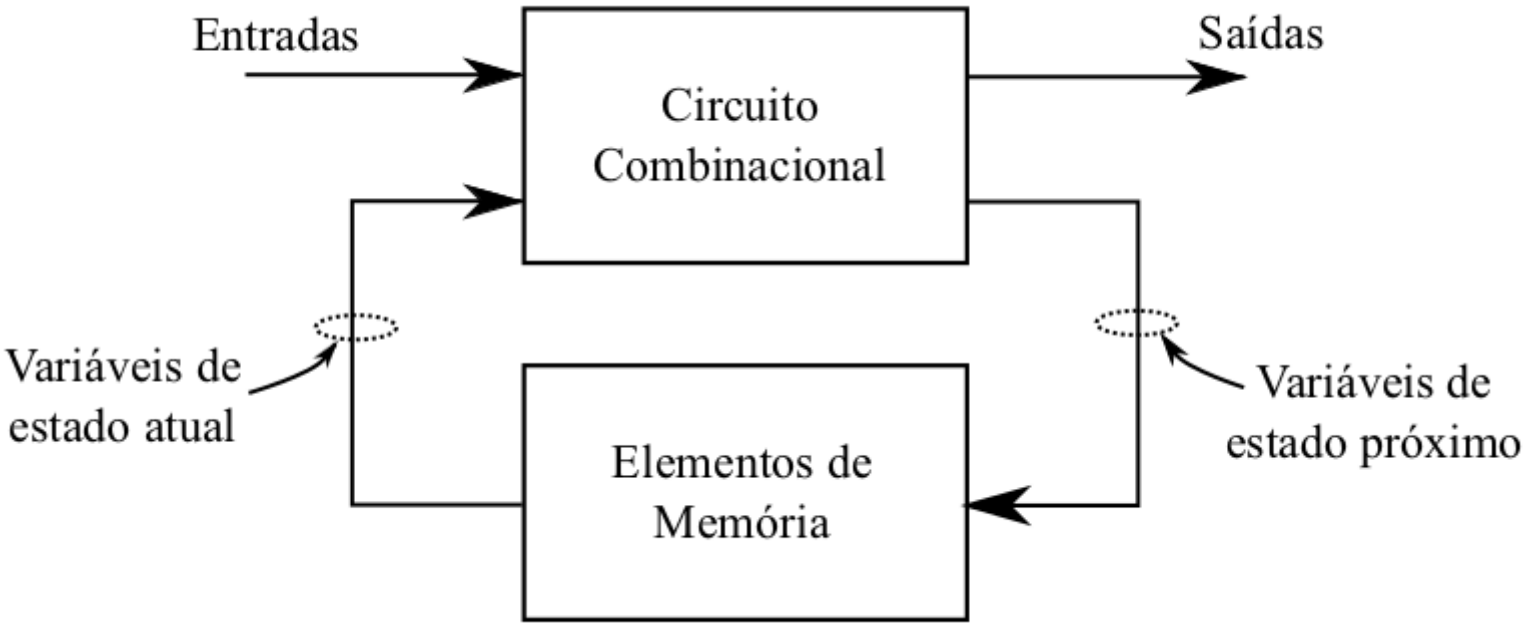
Projeto 5 – Registador com Enable e Reset

Descreva um registrador de 4 *bits* com entrada de *enable* (*en*) e *reset* (*rst*). O registrador possui uma entrada de controle *en*, que quando *en* = 1, deve-se efetuar a carga a cada pulso de *clock*. Caso contrário (*en* = 0), o último valor é mantido. O sinal de *rst* é assíncrono, ou seja, quando *rst* = 1, o registrado é zerado independente do sinal de *clock*. O registrador é sensível à borda de subida do relógio.



Máquinas de Estados

- Estrutura Geral para Máquinas de Estados:
  - Circuito Combinacional;
  - Elementos de Memória (Registrador de Estados);



Máquinas de Estados

- Os estados devem ser codificados e declarados como um novo tipo;
- O estado deve ser armazenado em **SIGNAL**;
- São necessários dois processos;
  - Primeiro processo:
    - Sensibilidade: Sinais de *Clock* e de *Reset*;
    - Registrador de estado;
    - Pode-se utilizar um sinal de *Reset* assíncrono.

Máquinas de Estados

- De forma a facilitar a codificação VHDL, pode-se criar um novo tipo de SIGNAL (enum) para representar o estado;
- Outros tipos podem ser definidos pelo usuário utilizando a diretiva TYPE;
- Exemplo: Tipo estado que pode assumir os valores A, B, C e D.

```
TYPE estado IS (A, B, C, D);
```

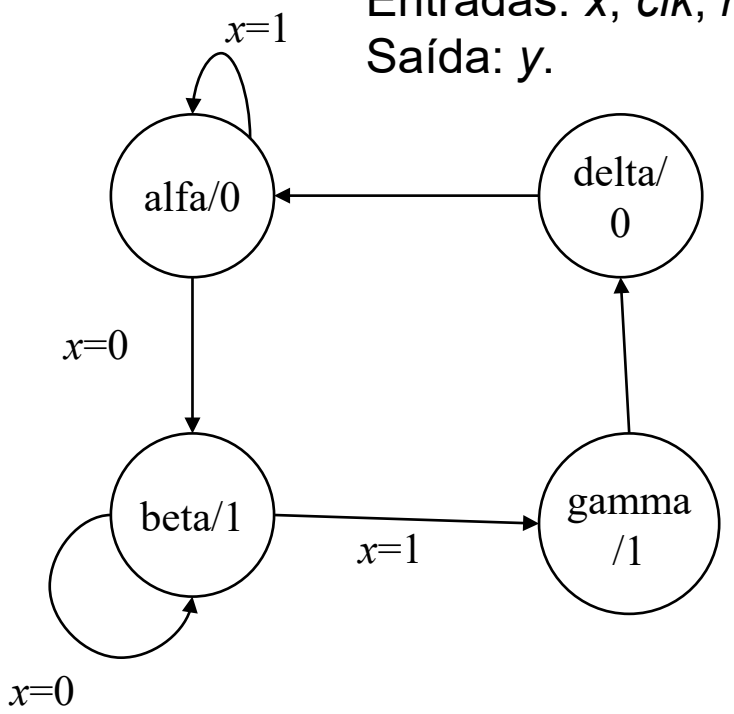
- Segundo processo:
  - Analisa o estado presente e as entradas (caso Mealy) para gerar a *saída*;
  - Sensibilidade: Estado Atual e Entrada;
  - Deve-se utilizar um **CASE**, com identificador de variável de estado, para definir a saída;

Máquinas de Estados

Projeto 6 – Máquina de Estados

Implemente a máquina de estados com o diagrama de transição apresentado na Figura ao lado.

Entradas: *x*, *clk*, *rst*;
Saída: *y*.



Máquinas de Estados Algorítmicas

- São máquinas de estados que permitem realizar operações aritméticas nos estados e tem armazenamento local (registradores internos);
- Formas de projeto:
  - Utilizar a mesma estrutura de uma máquina de estados, porém utilizando **SIGNAL** para armazenamento das informações;
  - Topologia RTL (*datapath* + controlador).

Máquinas de Estados Algorítmicas

Projeto 7 – Máquina de Estados Algoritmicas

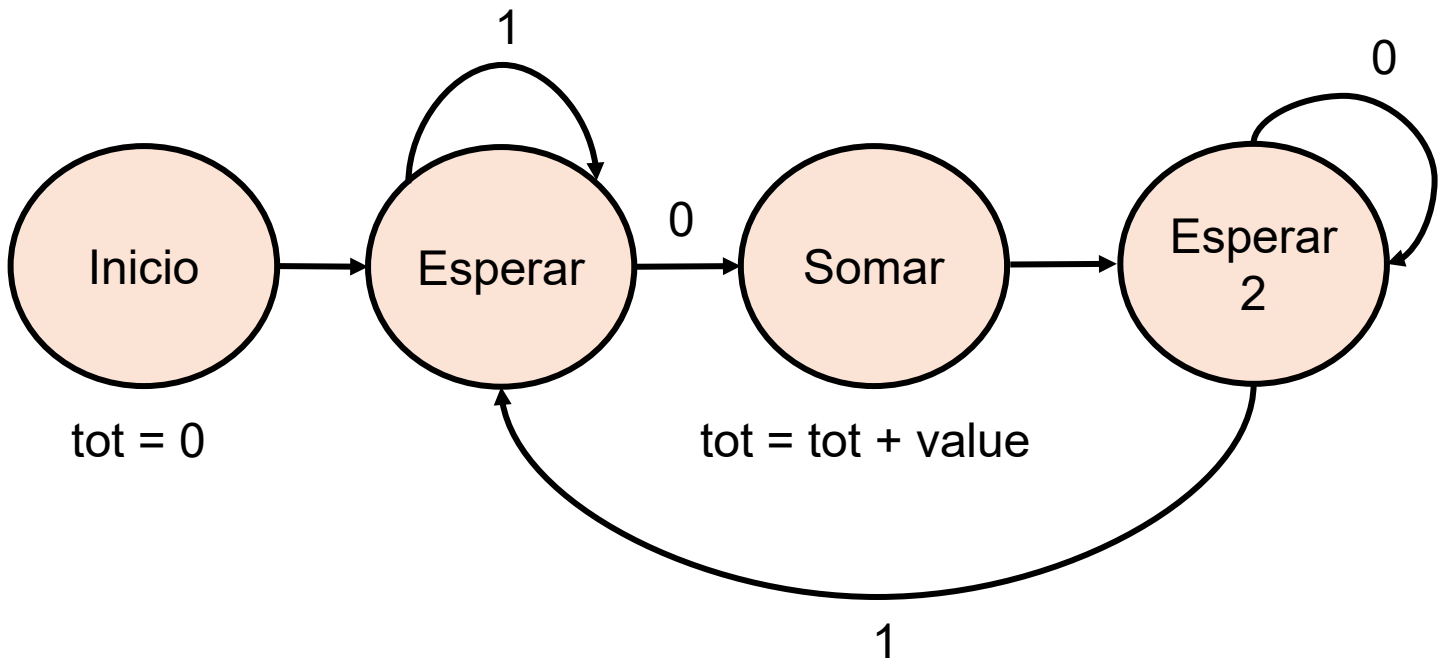
Implemente uma máquina de estados algorítmicas que apresente o seguinte funcionamento:

- A máquina deve possuir um registrador interno, denominado de *tot* (8 *bits*);
- Inicialmente *tot* = 0;
- Quando uma entrada *x* = 0, adiciona-se a entrada *value* (8 *bits*) ao registrador *tot* (*tot* = *tot* + *value*);
- Sincronizar o botão *x*, ou seja, ele pode ser nível lógico baixo por mais de um pulso de relógio, mas deve-se executar apenas uma soma.

Máquinas de Estados Algorítmicas

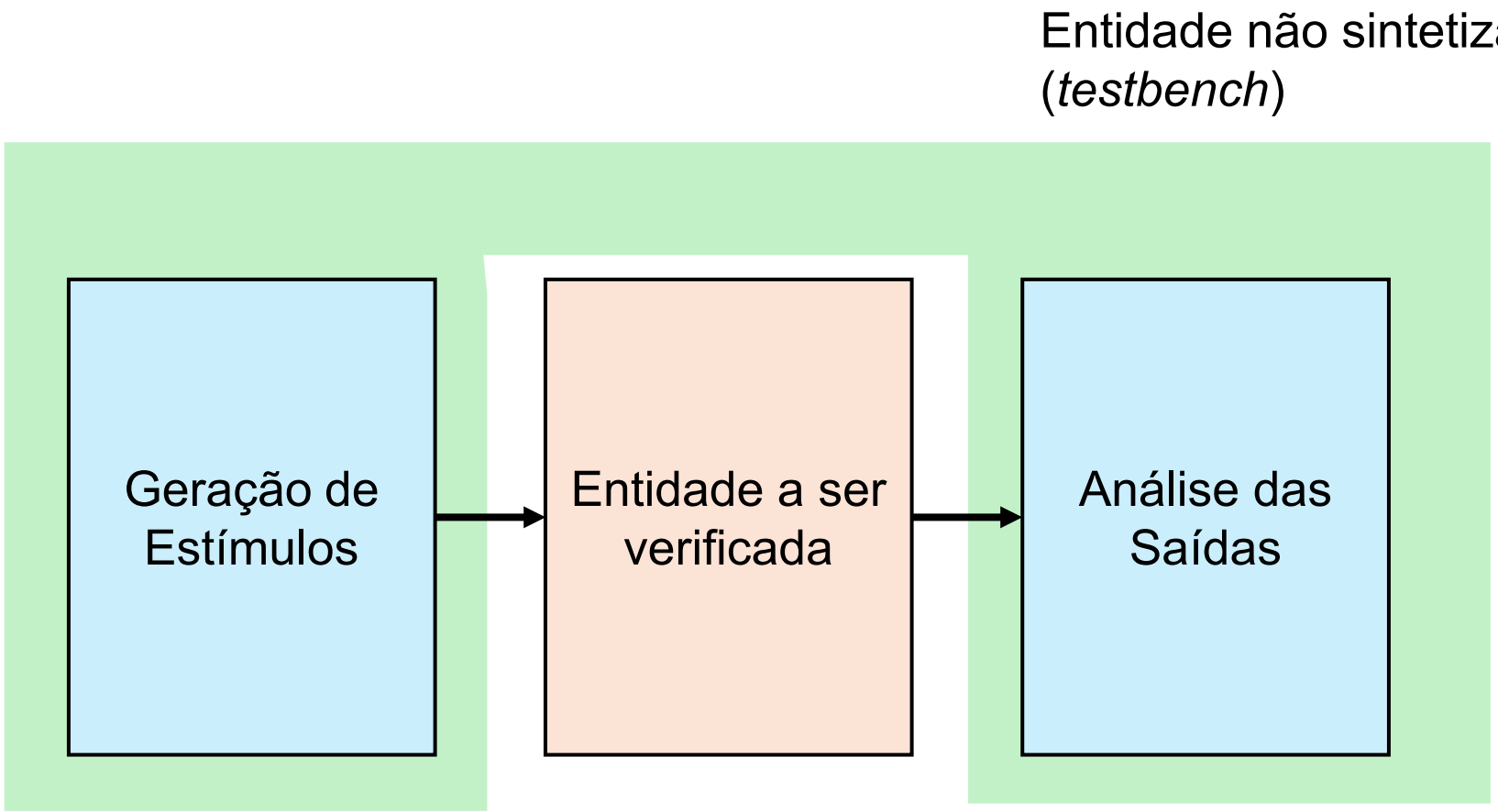
Projeto 7 – Máquina de Estados Algoritmicas

Registradores: tot (8 *bits*)
Entradas: clk, rst, x, value (8 *bits*)
Saídas: tot (8 *bits*)



Bancada de Testes (*Testbench*)

- *Testbech*: *Script* automatizado para simulação (verificação) de módulos descritos em linguagem de descrição de *hardware*;
- Visão geral de um *testbench*:



Bancada de Testes (*Testbench*)

- Os *testbench* não possuem entradas;
- A entidade a ser testada deve ser instanciado no *testbench*;
- As entradas e saídas devem ser declaradas como **`SIGNAL`**;
- Processo de Estimulos:
  - Responsável por gerar todos os sinais de entrada (com excessão do sinal de *clock*);
  - Estimulos devem ser adicionados antes da diretiva **`WAIT FOR`** `xx ns`;
  - Adicionar a biblioteca **`STD.ENV.STOP`** para usar a diretiva **`STOP`** para encerrar a simulação.

Bancada de Testes (*Testbench*)

- Geração de *Clock*:
  - Responsável por gerar o sinal de *clock*;
  - Formato geral:

```
clk <= not clk after clock_period / 2;
```

  - **`clock_period`** deve ser uma constante declarada com o período de *clock*;

```
constant clock_period: time := 10 ns;
```

Referências de Estudo