

# Introdução ao projeto de sistemas digitais

Prof. Ilan Sousa Correa

Universidade Federal do Pará (UFPA)

Instituto de Tecnologia (ITEC)

Faculdade de Eng. da Computação e Telecomunicações (FCT)

# Recapitulando a aula anterior

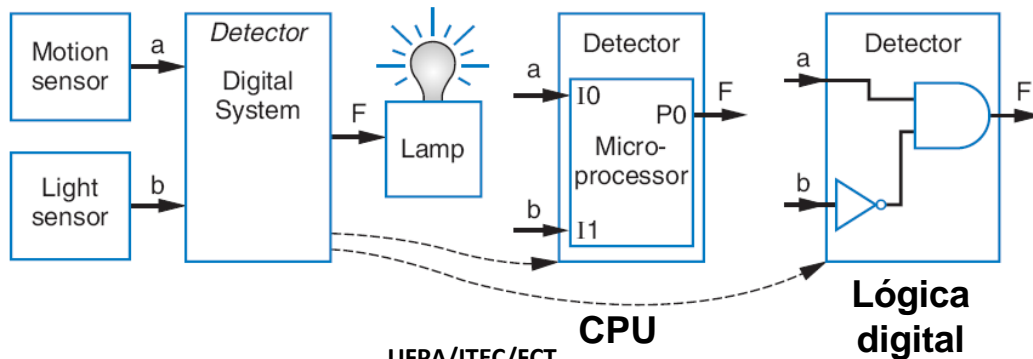
Projeto de sistemas digitais pode utilizar processadores ou lógica digital pura

Especificações  
em alto nível

Escolha da forma  
de implementação

	A	B	C	D	z
(0)	0	0	0	0	0
(1)	0	0	0	1	0
(2)	0	0	1	0	0
(3)	0	0	1	1	0
(4)	0	1	0	0	0
(5)	0	1	0	1	0
(6)	0	1	1	0	0
(7)	0	1	1	1	1 → $\overline{A}BCD$
(8)	1	0	0	0	1 → $A\overline{B}\overline{C}\overline{D}$
(9)	1	0	0	1	1 → $A\overline{B}\overline{C}D$
(10)	1	0	1	0	1 → $A\overline{B}CD$
(11)	1	0	1	1	1 → $A\overline{B}CD$
(12)	1	1	0	0	1 → $AB\overline{C}\overline{D}$
(13)	1	1	0	1	1 → $AB\overline{C}D$
(14)	1	1	1	0	1 → $ABCD$
(15)	1	1	1	1	1 → $ABCD$

$$z = \overline{A}BCD + A = BCD + A$$



```
void main()
{
    while (1) {
        P0 = I0 && !I1;
        // F = a and !b,
    }
}
```

# Recapitulando a aula anterior

Projeto de sistemas digitais pode utilizar processadores ou lógica digital pura

- Componentes normalmente utilizados nas implementações baseadas em lógica digital (a princípio, consideraremos somente circuitos combinacionais)
  - Multiplexadores, Somadores/Subtratores, Decodificadores, Multiplicadores

**Captura da tabela**

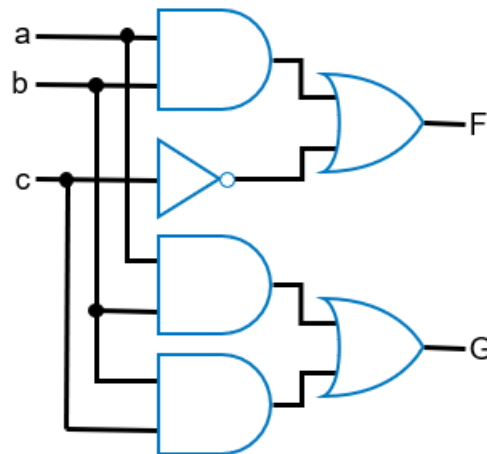
verdade					
	A	B	C	D	z
(0)	0	0	0	0	0
(1)	0	0	0	1	0
(2)	0	0	1	0	0
(3)	0	0	1	1	0
(4)	0	1	0	0	0
(5)	0	1	0	1	0
(6)	0	1	1	0	0
(7)	0	1	1	1	1 → $\bar{A}BCD$
(8)	1	0	0	0	1 → $AB\bar{C}\bar{D}$
(9)	1	0	0	1	1 → $AB\bar{C}D$
(10)	1	0	1	0	1 → $AB\bar{C}\bar{D}$
(11)	1	0	1	1	1 → $AB\bar{C}D$
(12)	1	1	0	0	1 → $ABCD$
(13)	1	1	0	1	1 → $AB\bar{C}D$
(14)	1	1	1	0	1 → $AB\bar{C}\bar{D}$
(15)	1	1	1	1	1 → $ABCD$

**Etapas do projeto destas funções digitais básicas**

**Geração da  
função lógica e  
sua minimização**

$$F = ab + c'$$

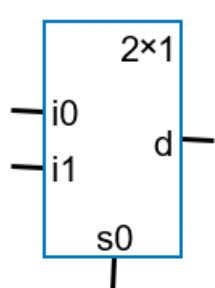
$$G = ab + bc$$



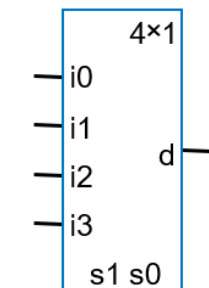
# Recapitulando a aula anterior

Projeto de sistemas digitais pode utilizar processadores ou lógica digital pura

- Componentes normalmente utilizados nas implementações baseadas em lógica digital (a princípio, consideraremos somente circuitos combinacionais)
  - **Multiplexadores**, Somadores/Subtratores, Decodificadores, Multiplicadores



2x1 mux

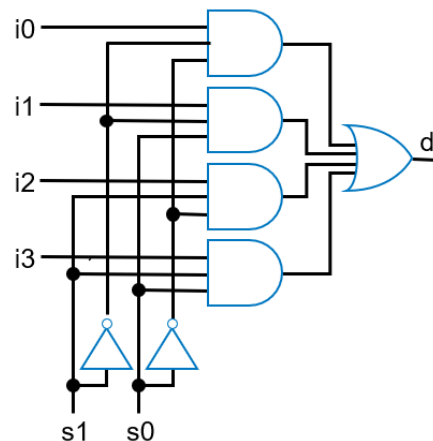


4x1 mux



	A	B	C	D	z
(0)	0	0	0	0	0
(1)	0	0	0	1	0
(2)	0	0	1	0	0
(3)	0	0	1	1	0
(4)	0	1	0	0	0
(5)	0	1	0	1	0
(6)	0	1	1	0	0
(7)	0	1	1	1	1
(8)	1	0	0	0	1
(9)	1	0	0	1	1
(10)	1	0	1	0	1
(11)	1	0	1	1	1
(12)	1	1	0	0	1
(13)	1	1	0	1	1
(14)	1	1	1	0	1
(15)	1	1	1	1	1

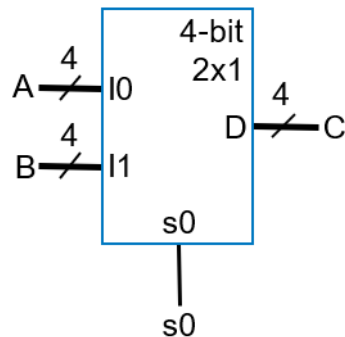
UFPA/ITEC/FCT



# Recapitulando a aula anterior

Projeto de sistemas digitais pode utilizar processadores ou lógica digital pura

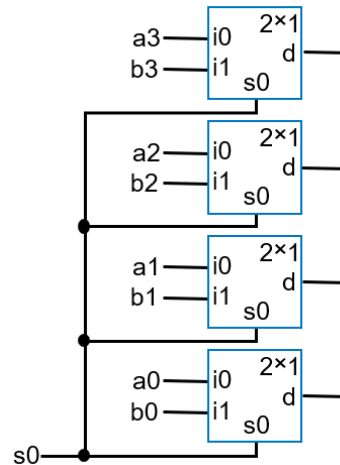
- Componentes normalmente utilizados nas implementações baseadas em lógica digital (a princípio, consideraremos somente circuitos combinacionais)
  - **Multiplexadores**, Somadores/Subtratores, Decodificadores, Multiplicadores



Em geral, trabalha-se com múltiplos bits



Uma instância da função lógica básica por bit



# Recapitulando a aula anterior

Projeto de sistemas digitais pode utilizar processadores ou lógica digital pura

- Componentes normalmente utilizados nas implementações baseadas em lógica digital (a princípio, consideraremos somente circuitos combinacionais)
  - Somadores/Subtratadores**

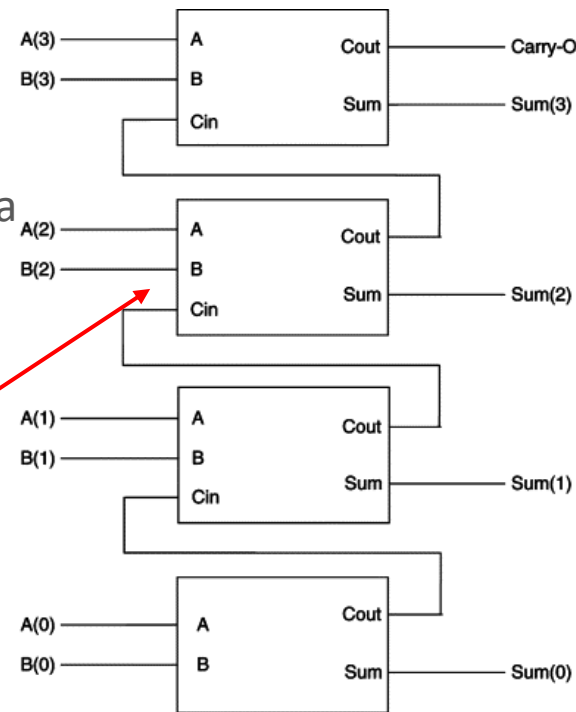
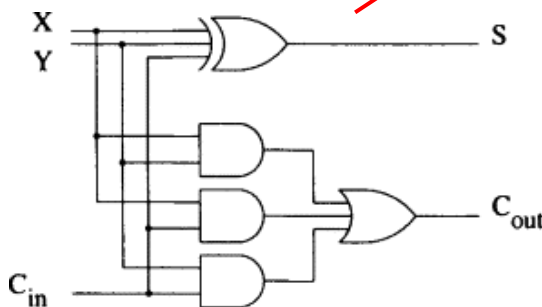
**Somador Completo:**

Bits A e B → Operandos

Bit Cin → Carry-in (“vem um”)

Bit Cout → Carry-out (“vai um”)

Bit S → Soma/Resultado



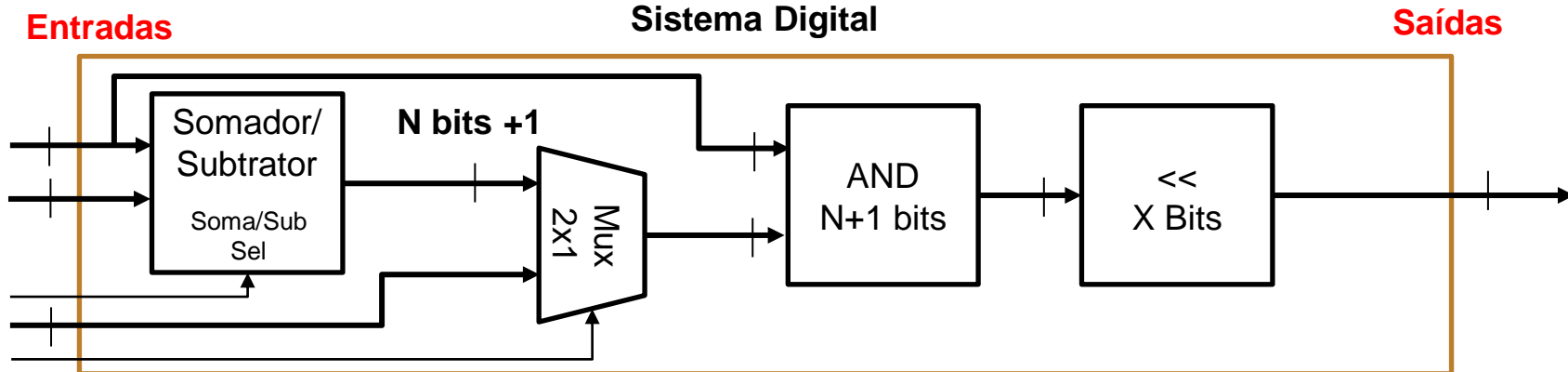
**4-bit binary addition**

	A (3) B (3)	A (2) B (2)	A (1) B (1)	A (0) B (0)	
Carry-Out	Sum (3)	Sum (2)	Sum (1)	Sum (0)	+

# Projeto de sistemas digitais com funções de “alto-nível”

Utilizaremos somente funções lógicas de alto-nível

- Desenvolvimento no chamado “Register transfer level”, ou nível RTL
- Não consideraremos o “register” na aula de hoje.
- Resultado do projeto inicial de um sistema digital: **Funções básicas interconectadas**



# Introdução à linguagem VHDL



# Introdução à linguagem VHDL

VHDL – Very High-Speed Integrated Circuits Hardware Description Language

- VHDL caracteriza um dos tipos de linguagens utilizadas para modelar circuitos digitais.
- A sigla deriva da junção de duas siglas, VHSIC e HDL, que descrevem o real intuito da linguagem.
  - VHSIC – Very High-Speed Integrated Circuits.
  - HDL – Hardware Description Language.
- VHDL, então, é uma linguagem utilizada na descrição do hardware (ou seja, descrição do funcionamento) de componentes digitais. Padronizada pelo IEEE.
- Quais são os dispositivos que utilizam esse tipo de linguagem?
  - Dispositivos Lógicos Programáveis: CPLD – Complex Programmable Logic Device e FPGA – Field Programmable Gate Array

# Aspectos gerais da linguagem VHDL

Permite uma descrição em diversos níveis de abstração:

- Nível Algorítmico;
- Nível de Transferência entre Registradores (RTL);
- Nível de Portas lógicas

Estilo de descrição de hardware

- **Comportamental**: um componente é desenvolvido baseado na análise das suas entradas, associando-as a determinadas respostas esperadas nas suas saídas (nível mais alto de abstração);
- **Estrutural**: um componente é modelado segundo um circuito lógico pré-estabelecido (nível mais baixo de abstração).

# Aspectos gerais da linguagem VHDL

Linguagem de código concorrente: a ordem dos comandos não importa

- Exceto em regiões específicas de código sequencial

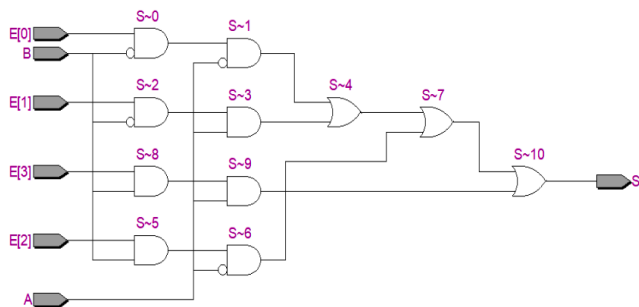
Processamento do código VHDL

- **Síntese:** “tradução” do código em VHDL (ou em outra linguagem HDL) para programar um FPGA ou CPLD  $\Rightarrow$  análoga à compilação
- **Ferramentas de síntese:** Nem todas as estruturas disponíveis na linguagem podem ser sintetizadas pela ferramenta;

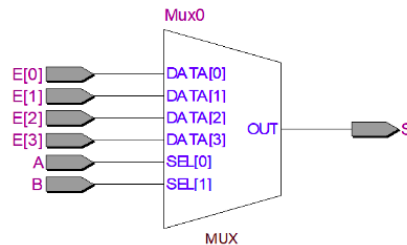
Favorece o desenvolvimento de projetos “Top-Down”,

- Há a fragmentação lógica do projeto. O sistema é visualizado de forma abstrata, com detalhamento posterior de seus subsistemas.

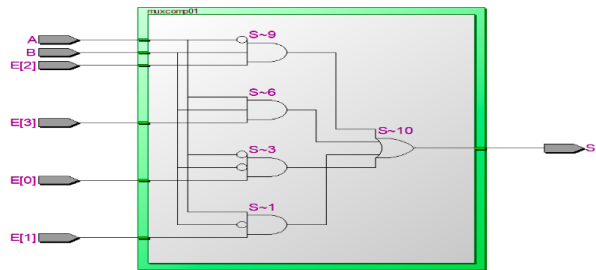
# Níveis de abstração



**Nível de Portas**



**Nível RTL**



**Descrição Estrutural (Resultado)**

```

ARCHITECTURE logic OF mux IS
  SIGNAL sel : BIT_VECTOR(1 DOWNTO 0);
BEGIN
  sel <= B & A;
  WITH (sel) SELECT
    S <= E(0) WHEN "00",
        E(1) WHEN "01",
        E(2) WHEN "10",
        E(3) WHEN "11",
        NULL WHEN OTHERS;
END ARCHITECTURE;

```

**Descrição Comportamental**

**Nível Algorítmico**

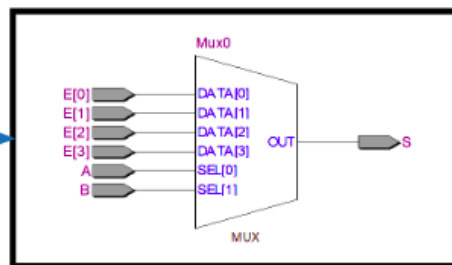
# Níveis de abstração

Descrição VHDL

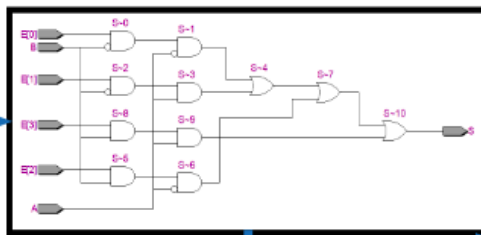
```
ARCHITECTURE logic OF mux IS  
  SIGNAL sel : BIT_VECTOR(1 DOWNT0 0);  
BEGIN  
  sel <= B & A;  
  WITH (sel) SELECT  
    S <= E(0) WHEN "00",  
         E(1) WHEN "01",  
         E(2) WHEN "10",  
         E(3) WHEN "11",  
         NULL WHEN OTHERS;  
END ARCHITECTURE;
```

Inferência

Nível RTL

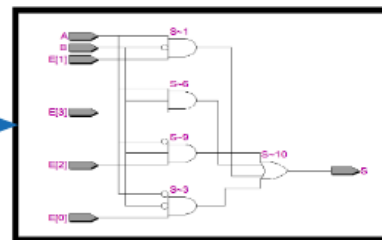


Tradução



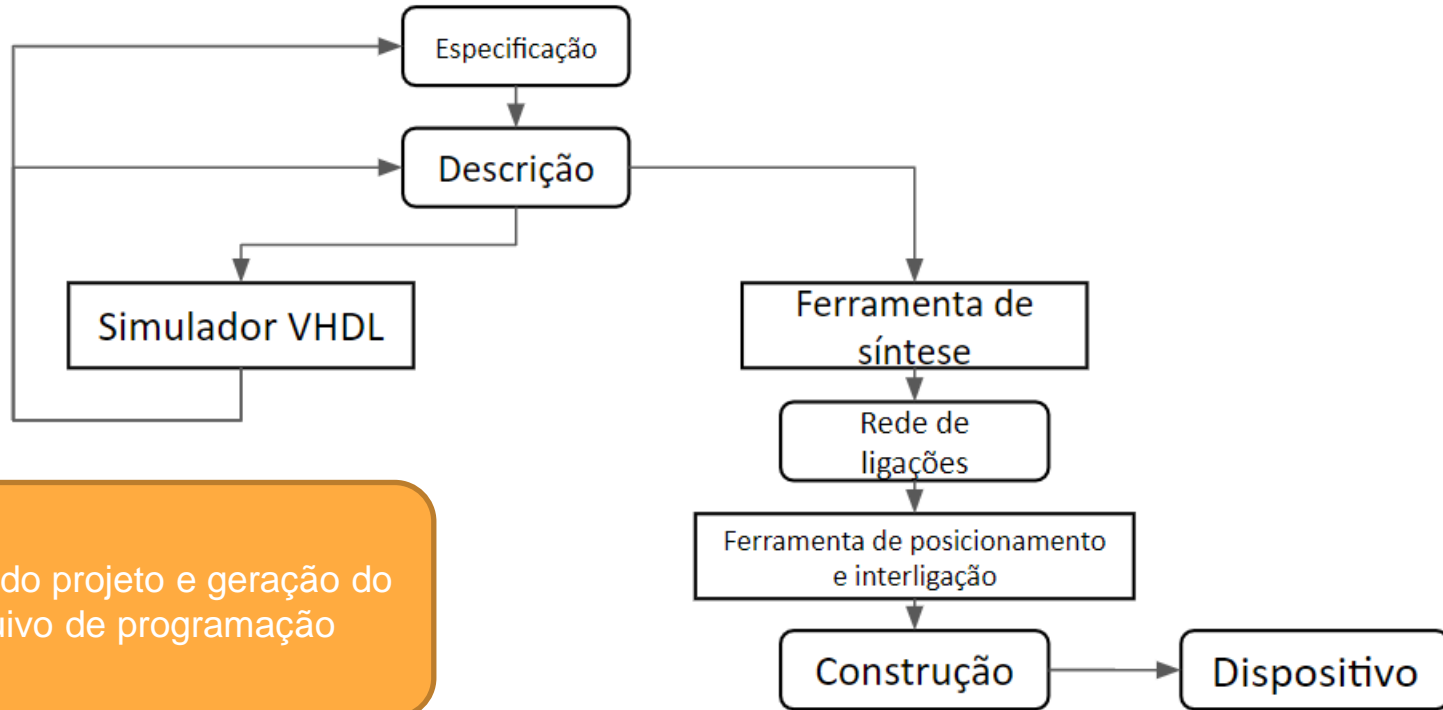
Otimização

Nível de Portas



Uma descrição de hardware em linguagem HDL gerará uma descrição do hardware em um nível mais detalhado

# Níveis de abstração



Etapas do projeto e geração do  
arquivo de programação

# Primeiro contato com a linguagem

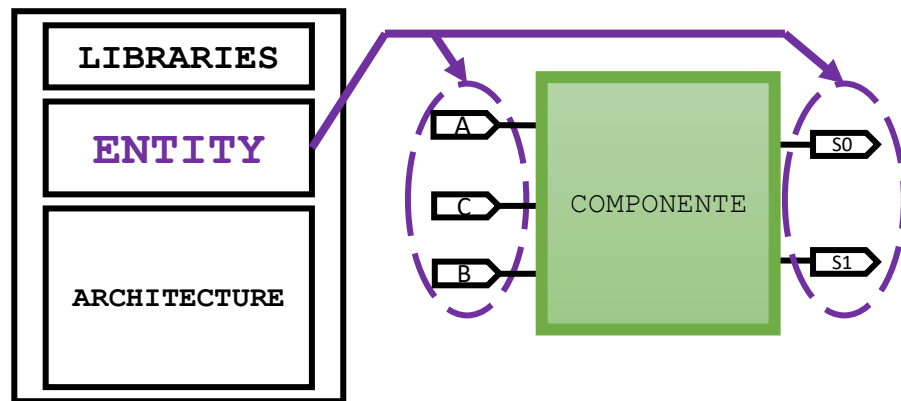
# Componentes básicos de um código VHDL

<b>Entity</b>	Define as portas de entrada e saída da descrição(visão externa do componente).
<b>Architecture</b>	Descreve as relações existentes entre as interfaces de entrada e saída(visão interna do componente).
<b>Package (Library)</b>	Contém a definição de subprogramas, constantes e tipos disponíveis para o projeto.



# Componentes básicos de um código VHDL

<b>Entity</b>	Define as portas de entrada e saída da descrição(visão externa do componente).
<b>Architecture</b>	Descreve as relações existentes entre as interfaces de entrada e saída(visão interna do componente).
<b>Package (Library)</b>	Contém a definição de subprogramas, constantes e tipos disponíveis para o projeto.



```

ENTITY <nome_entidade> IS
  GENERIC (
    -- Declaracao de constantes(somente)
    -- Pode ser omitido na descricao(opcional)
  );
  PORT (
    -- Declaracao das portas de
    -- entrada e saida(input/output)
    i0, i1 : IN      TIPO_A; -- entradas
    s0, s1 : OUT     TIPO_B; -- saidas
    z      : BUFFER  TIPO_C; -- saida
    w      : INOUT   TIPO_D -- entrada/saida
  );
END <nome_entidade>;
    
```

# Componentes básicos de um código VHDL

<b>Entity</b>	Define as portas de entrada e saída da descrição(visão externa do componente).
<b>Architecture</b>	Descreve as relações existentes entre as interfaces de entrada e saída(visão interna do componente).
<b>Package (Library)</b>	Contém a definição de subprogramas, constantes e tipos disponíveis para o projeto.

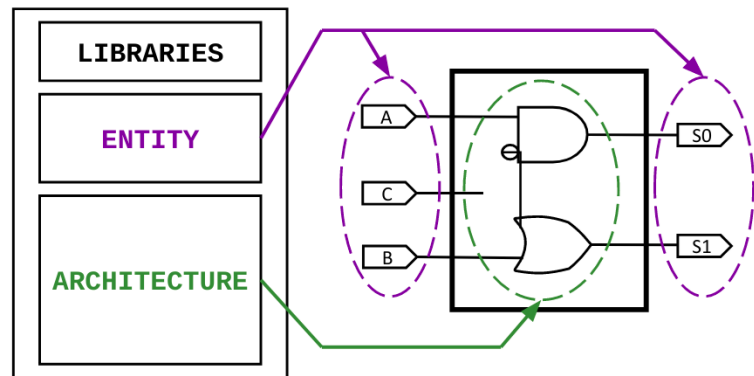
## Modos de operação das interfaces:

- **IN** : opera exclusivamente como entrada;
- **OUT**: opera exclusivamente como saída;
- **BUFFER**: opera como saída e pode ser utilizado para leitura interna no próprio sistema;
- **INOUT**: opera como um pino de entrada ou de saída, bidirecional (selecionável).

```
ENTITY <nome_entidade> IS
  GENERIC (
    -- Declaracao de constantes(somente)
    -- Pode ser omitido na descricao(opcional)
  );
  PORT (
    -- Declaracao das portas de
    -- entrada e saida(input/output)
    i0, i1 : IN      TIPO_A; -- entradas
    s0, s1 : OUT     TIPO_B; -- saidas
    z      : BUFFER  TIPO_C; -- saida
    w      : INOUT   TIPO_D -- entrada/saida
  );
END <nome_entidade>;
```

# Componentes básicos de um código VHDL

<b>Entity</b>	Define as portas de entrada e saída da descrição(visão externa do componente).
<b>Architecture</b>	Descreve as relações existentes entre as interfaces de entrada e saída(visão interna do componente).
<b>Package (Library)</b>	Contém a definição de subprogramas, constantes e tipos disponíveis para o projeto.



```

ARCHITECTURE <nome_identificador> OF <nome_entidade> IS
-- Regiao de declaracoes:
--Declaracao de constantes, sinais e/ou variaveis
--Declaracao de componentes
--Declaracao e corpo de subprogramas
--Definicao de novos tipos
BEGIN
-- Regiao de codigo concorrente
-- Descricao do sistema
-- Construcao de blocos e/ou processos

END <nome_identificador>;
    
```

# Componentes básicos de um código VHDL

<b>Entity</b>	Define as portas de entrada e saída da descrição(visão externa do componente).
<b>Architecture</b>	Descreve as relações existentes entre as interfaces de entrada e saída(visão interna do componente).
<b>Package (Library)</b>	Contém a definição de subprogramas, constantes e tipos disponíveis para o projeto.

```
ENTITY <nome_entidade> IS
  GENERIC (
    -- Declaracao de constantes(somente)
    -- Pode ser omitido na descricao(opcional)
  );
  PORT (
    -- Declaracao das portas de
    -- entrada e saida(input/output)
    i0, i1 : IN      TIPO_A; -- entradas
    s0, s1 : OUT     TIPO_B; -- saidas
    z      : BUFFER  TIPO_C; -- saida
    w      : INOUT   TIPO_D -- entrada/saida
  );
END <nome_entidade>;
```

```
ARCHITECTURE <nome_identificador> OF <nome_entidade> IS
  -- Regiao de declaracoes:
  --Declaracao de constantes, sinais e/ou variaveis
  --Declaracao de componentes
  --Declaracao e corpo de subprogramas
  --Definicao de novos tipos
BEGIN
  -- Regiao de codigo concorrente
  -- Descricao do sistema
  -- Construcão de blocos e/ou processos

END <nome_identificador>;
```

# Especificidades da linguagem VHDL

## Observações

- A linguagem possui palavras reservadas (keywords);
- Não há diferenciação entre maiúsculos e minúsculos (não é case sensitive);
- Cada sentença deve ser terminada com ponto e vírgula ( ; );
- Os nomes das variáveis devem iniciar com caracteres alfanuméricos ou “\_”(esse não deve ser utilizado no final de um nome e não pode ser usado duplicado);
- Os comentários em VHDL ocorrem depois de dois hífen seguidos “--”(não há suporte para comentários em bloco);
- Para melhor entendimento, as palavras reservadas pela linguagem serão escritas com caracteres maiúsculos.

# Especificidades da linguagem VHDL

Classes de Objetos: Constantes, Variáveis, Sinais e Arquivos

- Os objetos são elementos que contêm um valor armazenado;
- **Constante** irá conter um valor estático;
- **Variável** poderá alterar o valor inicialmente atribuído e só pode ser utilizada em regiões de código sequencial;
- **Sinal** uma conexão real de um circuito (interligação física), pode ter seu valor alterado e é utilizado em código sequencial e em regiões de código concorrente.
- O objeto “FILE” está associado à criação/leitura de arquivos (utilizado em simulações).
  - Suportado pelas ferramentas de síntese somente em alguns casos, como inicialização de conteúdo de memórias

# Especificidades da linguagem VHDL

## Classes de Objetos: Constantes, Variáveis, Sinais e Arquivos

```
ARCHITECTURE <nome_identificador> OF <nome_entidade> IS
-- Regiao de declaracoes:
--Declaracao de constantes, sinais e/ou variaveis
--Declaracao de componentes
--Declaracao e corpo de subprogramas
--Definicao de novos tipos
BEGIN
-- Regiao de codigo concorrente
-- Descricao do sistema
-- Construcao de blocos e/ou processos
END <nome_identificador>;
```

```
CONSTANT const_a : TIPO_X;
CONSTANT const_b : TIPO_X := valor_inicial;
CONSTANT z, y    : TIPO_X := valor_inicial;

VARIABLE var_a : TIPO_Y;
VARIABLE var_b : TIPO_Y := valor_inicial;
VARIABLE i, o  : TIPO_Y := valor_inicial;

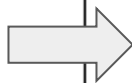
SIGNAL sig_a : TIPO_Z;
SIGNAL sig_b : TIPO_Z := valor_inicial;
SIGNAL s, t  : TIPO_Z := valor_inicial;
```

# Especificidades da linguagem VHDL

## Classes de Objetos: Constantes, Variáveis, Sinais e Arquivos

- Transferência de valores ao longo do código:
  - Observação: Para transferência de valores entre classes diferentes de objetos, devemos atentar para o tipo atribuído aos mesmos, pois devem ser obrigatoriamente iguais.

```
ARCHITECTURE <nome_identificador> OF <nome_entidade> IS
-- Regiao de declaracoes:
--Declaracao de constantes, sinais e/ou variaveis
--Declaracao de componentes
--Declaracao e corpo de subprogramas
--Definicao de novos tipos
BEGIN
-- Regiao de codigo concorrente
-- Descricao do sistema
-- Construcao de blocos e/ou processos
END <nome_identificador>;
```



Sinais:

```
sig_a <= sig_c; -- mesma classe de objetos
sig_b <= var_a; -- classes diferentes de objetos
sig_c <= const_a; -- classes diferentes de objetos
```

Variáveis:

```
var_a := sig_a;    -- classes diferentes de objetos
var_b := const_b;  -- classes diferentes de objetos
var_c := var_a;    -- mesma classe de objetos
```



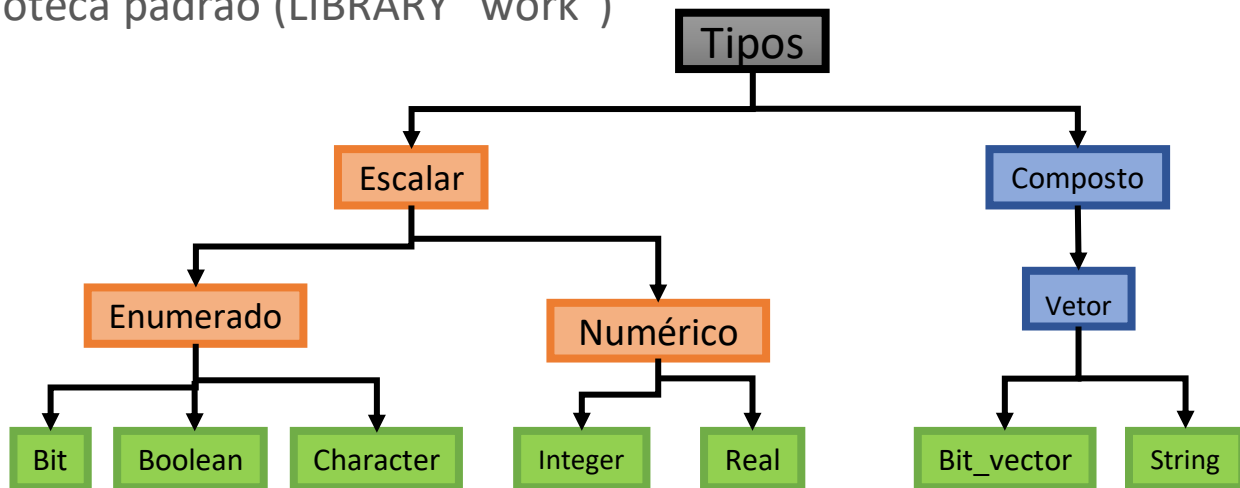
# Especificidades da linguagem VHDL

## Tipos de dados em VHDL

- Todo objeto deve ser declarado segundo uma especificação de tipo.
- Tipos pré-definidos na biblioteca padrão (LIBRARY “work”)

```
CONSTANT const_a : TIPO_X;  
CONSTANT const_b : TIPO_X := valor_inicial;  
CONSTANT z, y    : TIPO_X := valor_inicial;  
  
VARIABLE var_a : TIPO_Y;  
VARIABLE var_b : TIPO_Y := valor_inicial;  
VARIABLE i, o  : TIPO_Y := valor_inicial;  
  
SIGNAL sig_a : TIPO_Z;  
SIGNAL sig_b : TIPO_Z := valor_inicial;  
SIGNAL s, t  : TIPO_Z := valor_inicial;
```

```
ENTITY <nome_entidade> IS  
  GENERIC (  
    -- Declaracao de constantes(somente)  
    -- Pode ser omitido na descricao(opcional)  
  );  
  PORT (  
    -- Declaracao das portas de  
    -- entrada e saida(input/output)  
    i0, i1 : IN    TIPO_A; -- entradas  
    s0, s1 : OUT   TIPO_B; -- saidas  
    z      : BUFFER TIPO_C; -- saida  
    w      : INOUT  TIPO_D; -- entrada/saida  
  );  
END <nome_entidade>;
```



# Especificidades da linguagem VHDL

## Tipos de dados em VHDL

- Tipos escalares
  - Subclasse Enumerados:
    - Bit: Define apenas dois estados lógicos('0', '1');
    - Boolean: Possui dois resultados lógicos(False, True);
    - Character: Caracteres definidos pela tabela ASCII.

# Especificidades da linguagem VHDL

## Tipos de dados em VHDL

- Subclasse Numérico:
  - Integer: Valores positivos e negativos presentes no sistema numérico decimal(excluindo os números fracionários);
  - Real: Todos os formatos de números presentes no sistema numérico decimal(não suportado pela ferramenta de síntese).

# Especificidades da linguagem VHDL

## Tipos de dados em VHDL

<b>Tipo</b>	<b>Valor</b>
<b>BIT</b>	<code>'0'</code> ou <code>'1'</code>
<b>BOOLEAN</b>	<code>TRUE</code> , <code>FALSE</code>
<b>CHARACTER</b>	<code>'a'</code> , <code>'A'</code> , <code>'?'</code> , <code>'('</code>
<b>INTEGER</b>	$-2^{31} \leq x \leq 2^{31}-1$
<b>REAL</b>	$-3.65 \times 10^{47} \leq x \leq 3.65 \times 10^{47}$

# Especificidades da linguagem VHDL

## Tipos de dados em VHDL

- Exemplo de utilização na arquitetura
- Nota: tipos de dados devem também ser definidos na declaração da entidade

```
ARCHITECTURE <nome_identificador> OF <nome_entidade> IS
-- Regiao de declaracoes:
--Declaracao de constantes, sinais e/ou variaveis
--Declaracao de componentes
--Declaracao e corpo de subprogramas
--Definicao de novos tipos
BEGIN
-- Regiao de codigo concorrente
-- Descricao do sistema
-- Construcao de blocos e/ou processos
END <nome_identificador>;
```

```
CONSTANT const_a : BIT;
CONSTANT const_b : BOOLEAN := FALSE;
CONSTANT z, y : INTEGER := 8;

VARIABLE var_a : INTEGER RANGE 0 TO 7;
VARIABLE var_b : CHARACTER := 'F';
VARIABLE i, o : BIT := '0';

SIGNAL sig_a : BOOLEAN;
SIGNAL sig_b : CHARACTER := '?';
SIGNAL s, t : INTEGER RANGE 13 TO 55;
```

# Especificidades da linguagem VHDL

## Tipos de dados em VHDL

- Tipos compostos: São um conjunto de elementos do mesmo tipo acessados por um índice, ordenados em um vetor.
  - Subclasse Vetor:
    - Bit\_vector: contêm elementos do tipo Bit;
    - String: contêm elementos do tipo Character.

# Especificidades da linguagem VHDL

## Tipos de dados em VHDL

- Tipos compostos

```
ARCHITECTURE <nome_identificador> OF <nome_entidade> IS
-- Regiao de declaracoes:
--Declaracao de constantes, sinais e/ou variaveis
--Declaracao de componentes
--Declaracao e corpo de subprogramas
--Definicao de novos tipos
BEGIN
-- Regiao de codigo concorrente
-- Descricao do sistema
-- Construcão de blocos e/ou processos
END <nome_identificador>;
```

```
CONSTANT const_a : BIT_VECTOR(3 DOWNT0 0);
CONSTANT const_b : BIT_VECTOR(0 TO 3);
CONSTANT z, y    : STRING(1 TO 4) := "FPGA";

VARIABLE var_a : BIT_VECTOR(1 TO 6) := "110100";
VARIABLE var_b : BIT_VECTOR(5 DOWTO 0) := B"11_01_00";
VARIABLE i, o  : STRING(3 DOWNT0 0) := ('V', 'H', 'D', 'L');

SIGNAL sig_a : BIT_VECTOR(7 DOWNT0 4) := X"A";
SIGNAL sig_b : BIT_VECTOR(1 TO 4) := O"12";
SIGNAL s, t  : STRING(1 TO 12) := "HELLO WORLD!";
```

# Especificidades da linguagem VHDL

## Operadores

PRECEDÊNCIA	CLASSES	OPERADORES
MENOR	LÓGICOS	and or nand nor xor xnor
.	RELACIONAIS	= /= < <= > >=
.	ADIÇÃO	+ - &
MAIOR	DIVERSOS	not



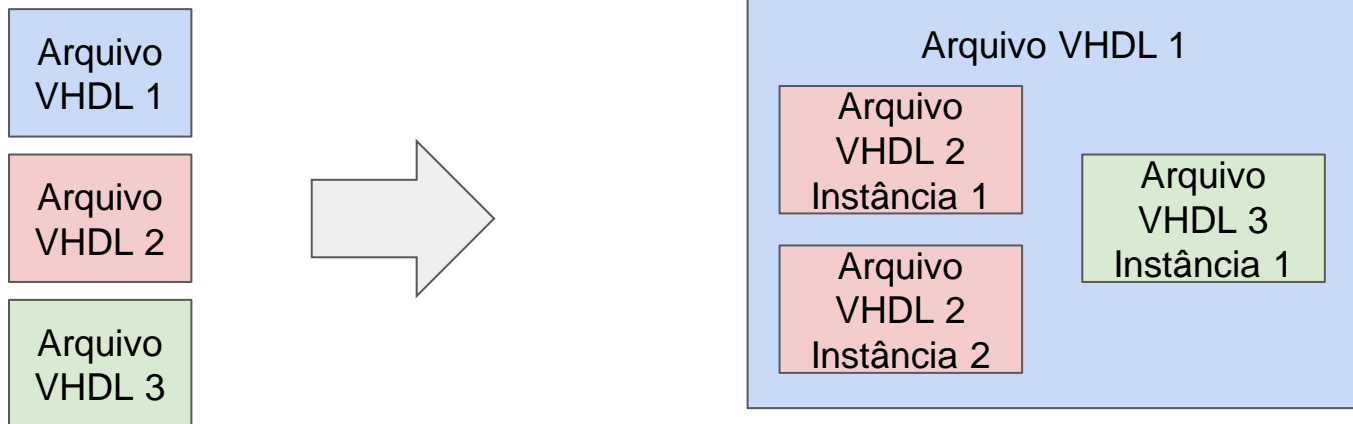
# Especificidades da linguagem VHDL

## Código hierárquico

- Componentes em VHDL: Utilização de uma entidade como componente de outra entidade
  - Código hierárquico: códigos VHDL como “sub”-códigos de outro código VHDL
  - Um código VHDL que é utilizado por outro código VHDL é chamado de componente
  - O código é “replicado” formando instâncias

# Especificidades da linguagem VHDL

Código hierárquico




# Especificidades da linguagem VHDL

## Código hierárquico

- Declaração de um componente de um código VHDL

## Declarar o componente

```
ARCHITECTURE <nome_identificador> OF <nome_entidade> IS
-- Regiao de declaracoes:
--Declaracao de constantes, sinais e/ou variaveis
--Declaracao de componentes
--Declaracao e corpo de subprogramas
--Definicao de novos tipos
BEGIN
-- Regiao de codigo concorrente
-- Descricao do sistema
-- Construcão de blocos e/ou processos
END <nome_identificador>;
```



```
COMPONENT <component_name>
  GENERIC
  ( <name> : <type>;
    <name> : <type> := <default_value>
  );


  PORT
  ( -- Input ports
    <name> : IN <type>;
    <name> : IN <type> := <default_value>;
    -- Inout ports
    <name> : INOUT <type>;
    -- Output ports
    <name> : OUT <type>;
    <name> : OUT <type> := <default_value>
  );
END COMPONENT;
```

# Especificidades da linguagem VHDL

## Código hierárquico

- Instanciação de um componente de um código VHDL

```
ARCHITECTURE <nome_identificador> OF <nome_entidade> IS
-- Regiao de declaracoes:
--Declaracao de constantes, sinais e/ou variaveis
--Declaracao de componentes
--Declaracao e corpo de subprogramas
--Definicao de novos tipos
BEGIN
-- Regiao de codigo concorrente
-- Descricao do sistema
-- Construcao de blocos e/ou processos
END <nome_identificador>;
```



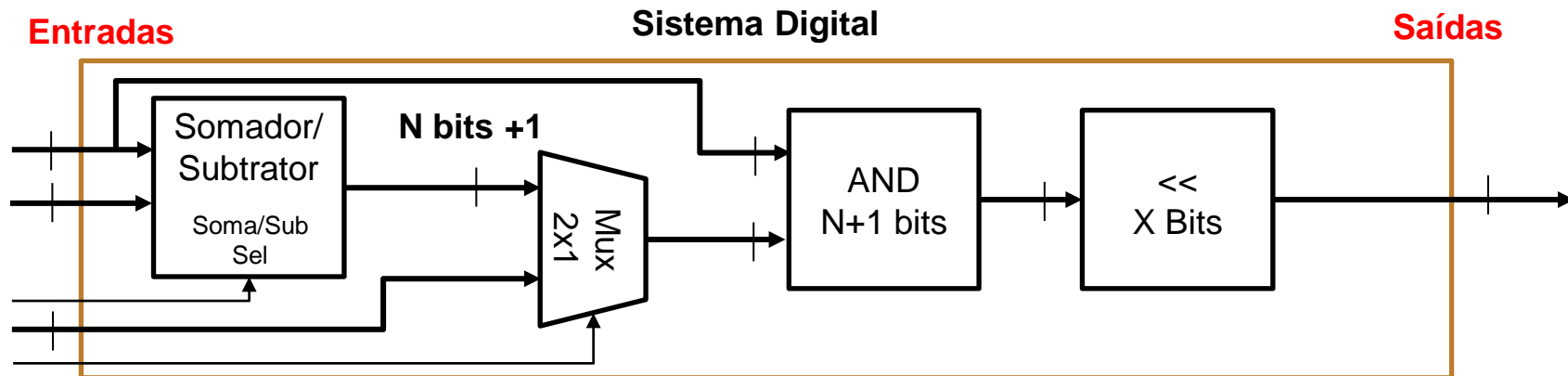
## Instanciar o componente

```
<instance_name> : <component_name>
  GENERIC MAP
  (
    <name> => <value>,
    ...
  )
  PORT MAP
  (
    <formal_input> => <expression>,
    <formal_output> => <signal>,
    <formal_inout> => <signal>,
    ...
  );
```

# Exemplo de códigos em VHDL

# Resgatando o exemplo do nosso sistema digital fictício

Ver detalhes no código de cada um dos componentes



# Bibliografia

- Sistemas digitais: Projeto, Otimização e HDLs, Frank Vahid, Ed. Bookman, 1ª Ed., 2008
- Tocci, R. J., Widmer, N. S. Sistemas digitais. 7. ed. Rio de Janeiro: LTC, 1998.
- Roberto d'Amore, VHDL: Descrição e síntese de circuitos digitais, 2ª Edição, Editora LTC, 2012