PCS 3115 (PCS2215)

Sistemas Digitais I

Módulo 09 – VHDL

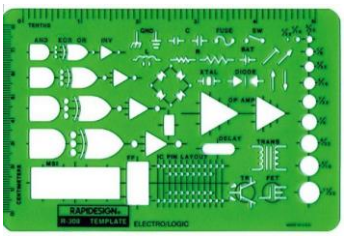
Prof. Dr. Marcos A. Simplicio Jr.

versão: 4.0 (Jan/2017)

Referências

- Free Range VHDL: http://freerangefactory.org/books_tuts.html
  - 1-2: leitura recomendada
  - 3-5: leitura obrigatória
  - 6: até 6.7: leitura obrigatória
  - 8: leitura obrigatória, ignorar parte sequencial
  - 10: leitura recomendada
  - Apendice B: leitura obrigatória
- Capítulo 5 do Wakerly
  - Obrigatório: 5 e 5.1 (todas subseções): introdução
  - Obrigatório: 5.3: VHDL
  - Recomendável: 5.3.3 em diante

Breve Perspectiva Histórica



- Há + de 30 anos:
  - Lápis, papel, gabarito
- Anos 80:
  - Editor de esquemáticos;
  - Introdução de HDL.
- Anos 90: crescimento do uso de HDL
  - Maior disponibilidade e menor custo de dispositivos programáveis (PLD, CPLD's, FPGA's);
  - Aumento da densidade de componentes ASIC's (*Application-Specific Integrated Circuit*): + 1 milhão de componentes.

Breve Perspectiva Histórica

- Editores auxiliam com complexidade, mas...
  - Como **verificar que não há erros** no desenho de um circuito?
    - Ex.: linhas que se cruzam têm conexões ou não?
  - Como expressar o **comportamento** de um circuito?
    - Desenho esquemático nem sempre é muito revelador (análise é necessária)...
- Solução: **linguagem de descrição** simulável e verificável, com alto nível de abstração

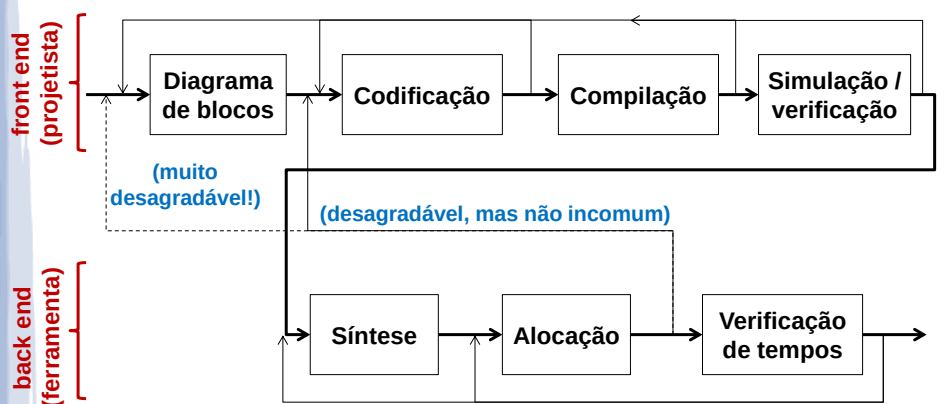
HDL *(Hardware Description Language)*

- É uma linguagem para
  - Descrever (**modelar**) e **documentar** um sistema digital em qualquer nível de abstração.
  - **Simular** (testar) e **sintetizar** um sistema digital
- **Pontos importantes:**
  - VHDL **NÃO** é “mais uma linguagem de programação”
    - **Não é uma “sequência de instruções”**, mas sim uma descrição dos módulos de hardware que compõem o sistema digital
    - Em geral, **instruções consecutivas “executam ao mesmo tempo”**: são partes do hardware processando o sinal de forma **concorrente**!!!
  - VHDL requer uma **ideia de como o hardware deve ficar**
    - Os **módulos** que compõem a solução costumam aparecer em um bom projeto em VHDL: isso requer um pouco de **experiência**...

HDL: ferramentas

- Diversas **ferramentas** compõem um ambiente HDL
  - **Editor de texto**: para escrever diretamente em HDL
  - **Compilador**: verificação e validação do programa, permitindo identificar erros de sintaxe
  - **Sintetizador**: transforma o projeto compilado no circuito voltado a uma tecnologia de hardware específica, (ex.: ASIC, FPGA, …) considerando blocos disponíveis
  - **Simulador**: permite simular comportamento do circuito ao longo do tempo (incluindo atrasos dos circuitos!)
    - ***Testbench***: programa em HDL que fornece entradas e verifica saídas do sistema
    - **Graficamente**: formas de onda
  - **Plataforma**: comumente, uma FPGA

Fluxo de Projeto



Porque [V]HDL?

- **VHDL**: VHSIC Hardware Description Language
  - VHSIC: Very High Speed Integrated Circuits
- Aumenta a **produtividade**
  - Impraticável desenhar um diagrama de um sistema complexo (ex.: processador Intel i7...)
- **Modularização**
  - Reutilizar componentes facilmente
  - Projetar em um time (vários módulos)
- VHDL é **padrão IEEE**
  - Muitas ferramentas disponíveis para todos os passos de um projeto, incluindo síntese

Sintaxe: algumas premissas

- Antes de mais nada, alguns detalhes:
  - VHDL **não diferencia maiúsculas de minúsculas**: a escolha é principalmente uma **questão de estilo**
  - VHDL **ignora espaços e quebras de linha**: servem apenas para melhor visibilidade
  - **Variáveis** definidas por usuários devem **começar com letras** e podem conter **letras, números e "_"**
    - Nota: não podem acabar com "_" ou terem dois "_" em sequência
  - **Comentários** são escritos com **"– –"**
    - Equivalente ao "//" em C e Java
  - **Parênteses**: definem precedência e/ou melhoram visibilidade
  - Variáveis reservadas nos exemplos são mostradas em cores

Sintaxe: algumas premissas

- Dois exemplos simples:

```
-- "<=" usado para atribuição entre sinais
-- comandos devem terminar em ";"
```

Equivalentes

```
Sum <= A xor B;          -- executados
Carry <= a and b;        -- concorrentemente
```

```
cArRy <= A  and   B;      -- executados
sum   <= (A    xor   b); -- concorrentemente
```

Modelo temporal: exemplo ilustrativo

- As declarações em VHDL são executadas **concorrentemente**
  - Estamos descrevendo um hardware, **não uma sequência de instruções** de software…
- Exemplo: operação de **swap(a,b)**: "troca a por b e vice-versa"

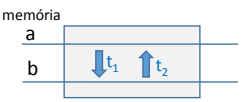
Em C, isso **não** funciona:
1: a = b;
2: b = a;

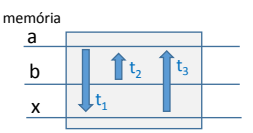
Em C, isso funciona:
1: x = a;
2: a = b;
3: b = x

Em C, também funciona:
(mas desperdiça memória...)
1: sa = b;
2: sb = a;

tempo	memória	
	a	b
0	5	7
1	7	7
2	7	7

tempo	memória		
	a	b	x
0	5	7	*
1	5	7	5
2	7	7	5
3	7	5	5





Modelo temporal: exemplo ilustrativo

- As declarações em VHDL são executadas **concorrentemente**
  - Estamos descrevendo um hardware, **não uma sequência de instruções** de software…
- Exemplo: operação de **swap(a,b)**: "troca a por b e vice-versa"

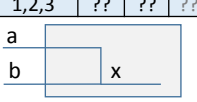
Em C, isso **não** funciona:
1: a = b;
2: b = a;

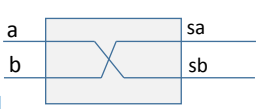
Em C, isso funciona:
1: x = a;
2: a = b;
3: b = x

Em VHDL, isso **não** funciona:
1: x <= a;
2: a <= b;
3: b <= x;

Em VHDL, isso funciona:
1: sa <= b;
2: sb <= a;

tempo	memória		
	a	b	x
0	5	7	*
1,2,3	??	??	??





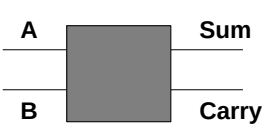

```
entity swap is
  port (a, b: in STD_LOGIC; sa,sb: out STD_LOGIC);
end swap
```

VHDL: ESTRUTURA

Estrutura de componentes em VHDL

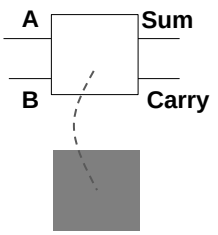
- Estrutura **modularizada**:
  - Definem-se componentes como "**caixas-pretas**": com **entradas e saídas**
  - Define-se o **conteúdo** da caixa preta
  - Caixas pretas podem ser **reutilizadas**

➔ Semelhante a "funções" na linguagem C



Estrutura de componentes em VHDL

- Entidade (`entity`)
  - Declaração bem definida das **entradas e saídas** do componente (i.e., sua **interface**)
- Arquitetura (`architecture`)
  - Descrição da **funcionalidade** da entidade, ou seja, sua estrutura interna
  - A arquitetura pode utilizar outras entidades internamente (ex.: um somador pode ser construído a partir de diversas instâncias de portas lógicas)



- Ambas contidas no mesmo arquivo texto (.vhd)

Estrutura de componentes em VHDL

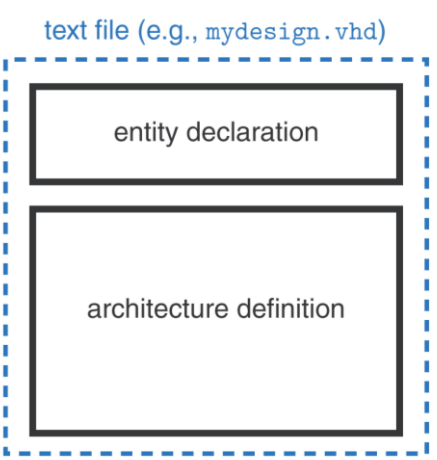


Figure 5-3
VHDL program file structure.

Tipos de objetos em VHDL

- Três tipos principais:
  - **Sinal** (`signal`) : equivalem a nomes de fios em um circuito
    - Mais comumente usados nos exemplos
    - Ex.:
    - Ex.:

```
signal sig1 : BIT;  -- declaração
sig1 <= '1';        -- atribuição de valor
```

  - **Variável** (`variable`): similar a "signal", porém mais abstrato,
    - Não corresponde necessariamente a um ponto físico no circuito sintetizado: podem ser vistos como "variáveis locais" em C
    - Uso ficará mais claro quando discutirmos processos (`process`)
    - Ex.:
    - Ex.:

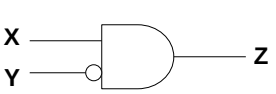
```
variable var1 : BIT;   -- declaração
var1 := '1';           -- atribuição de valor
```

  - **Constante**: valores fixos
    - Ex.: `'1'`

Entidade: sintaxe

```
entity entity-name is
  port (signal-names : mode signal-type;
        signal-names : mode signal-type;
        ...
        signal-names : mode signal-type);
end entity-name;
```

- Exemplo: qual a **entidade** que representa um "Inibidor", que dá saída X quando Y = 0, e saída 0 quando Y = 1?



X	Y	Z
0	0	0
0	1	0
1	0	1
1	1	0

Entidade: sintaxe

```
entity entity-name is
  port (signal-names : mode signal-type;
        signal-names : mode signal-type;
        ...
        signal-names : mode signal-type);
end entity-name;
```

Exemplo

```
entity Inhibit is          -- quando Y é 0, saída é X
   port (X, Y : in BIT;    -- quando Y é 1, saída
         Z    : out BIT); -- é 0 (inibe saída)
end Inhibit;
```

Entidade: sintaxe

- "entity-name": nome da entidade
- "signal-name": nome da porta de entrada/saída
- "mode": tipo e direção das portas. Valores:
  - **in**: entrada; **out**: saída; **inout**: saída E entrada;
- "signal-type": tipo da entrada/saída. Valores: vários...
  - Ex: bit, bit_vector, integer, ... (podem ser criados por usuário)

```
entity entity-name is
  port (signal-names : mode signal-type;
        signal-names : mode signal-type;
        ...
        signal-names : mode signal-type);
end entity-name;
```

Ponto-e-vírgula p/ cada conjunto de sinais

fora do parênteses

Não esquecer o ";" final

Arquitetura: sintaxe

```
architecture architecture-name of entity-name is
   type declarations
   signal declarations
   constant declarations
   function definitions
   procedure definitions
   component declarations
begin
   concurrent-statement
   ...
   concurrent-statement
end architecture-name;
```

Deve ser o mesmo que na "entity"

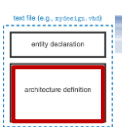
opcionais ("variáveis locais")

operação concorrente

Exemplo

```
architecture Inhibit_arch of Inhibit is
begin
  Z <= '1' when X='1' and Y='0' else '0';
end Inhibit_arch;
```

X	Y	Z
0	0	0
0	1	0
1	0	1
1	1	0



Arquitetura: sintaxe

- Associada a uma entidade pelo nome
  - Pode definir o **comportamento** desta: fluxo de informação (mais parecido com um programa)
  - Pode definir a **estrutura** interna da entidade: ligação entre portas (mais parecido com um desenho esquemático)
- Entidade pode ter várias arquiteturas associadas a ela
  - Comportamento puro (descrição comportamental), estrutura pura (descrição estrutural), ou mista
- Arquitetura tem acesso a portas declaradas na entidade e também a "variáveis locais":
  - `signal signal-names : signal-type; --similar a nomes de fios em um diagrama de circuito`

Componente: exemplo completo

```
entity Inhibit is
  port (X,Y: in BIT;
        Z:   out BIT);
end Inhibit;

architecture Inhibit_arch of Inhibit is
begin
  Z <= '1' when X='1' and Y='0' else '0';
end Inhibit_arch;
```

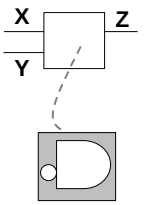
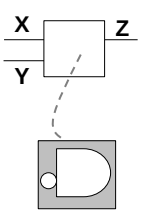


Table 5-13
VHDL program for an "inhibit" gate.

Componente: exemplo completo

```
entity Inhibit is
  port (X, Y : in BIT;
        Z    : out BIT);
end Inhibit;

architecture InhibitArch of Inhibit is
  signal nY : BIT;   -- sinal intermediário
begin                -- descrição alterativa
  Z   <= nY and X;
  nY  <= not Y;
end InhibitArch;
```



Perceba que a ordem das linhas não importa: seria melhor inverter para melhor legibilidade, mas código funciona...

Entidade: comparação com C

VHDL

```
entity Adder is
   port (X, Y : in  BIT;
         Sum  : out BIT);
end Adder;
```

Linguagem C

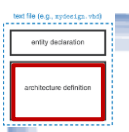
```
int Adder (int X, int Y);
```

Apenas 1 saída, sem nome.
Usando int para armazenar 1 bit.

```
entity Adder2 is
   port (X, Y : in BIT;
         Sum  : out BIT;
         Carry: out BIT);
end Adder2;
```

```
int Adder2 (int X, int Y,
int* Carry);
```

saídas adicionais podem ser implementadas via ponteiros.



Arquitetura: comparação com C

VHDL

```
architecture AdderArch of Adder is
begin
   Sum <= X xor Y;
end AdderArch;
```

Com bibliotecas adequadas, pode-se usar o "+" diretamente sobre bits ou conjuntos de bits (será visto mais adiante)

Linguagem C

```
int Adder (int X, int Y){
    return X ^ Y;
}
```

Soma de dois bits, sem carry (equivale a um XOR)

```
architecture AdderArch of
Adder2 is
begin
   Carry <= X and Y; -- concor-
   Sum   <= X xor Y; -- rente
end AdderArch;
```

```
int Adder2 (int X, int Y,
int* Carry){
   *Carry = X & Y; //sequen-
   return X ^ Y;   //cial
}
```

VHDL: comparação com C

- Nota: comparação para fins didáticos apenas!
  - Código em **C**: **gera conjunto de instruções para um processador**, que as lê e executa sequencialmente
    - Processador costuma ser um hardware de propósito geral
  - Código em **VHDL**: **gera hardware** que processa entradas de forma **concorrente**
    - Pode-se dizer que o código gera um processador de propósito específico
- Problemas práticos de engenharia podem ser resolvidos com software ou com hardware
  - Mas linguagens para desenvolvimento de software e sintetização de hardware criam tipos de **artefatos distintos**

VHDL: TIPOS DE DADOS & OPERAÇÕES

Tipos de dados

- Linguagem **fortemente tipada**
  - Todos os **sinais e variáveis** possuem um tipo
  - Tipos são **verificados na compilação**
  - Cada tipo possui um **conjunto definido de valores** e **operadores** válidos.
- **Tipos podem ser definidos pelos usuários**
  - Tipos mais usados são os definidos pelo IEEE: **std_logic** e **std_logic_vector**
  - Criação de novos tipos: vide apêndice
- **Cuidado com a inicialização**
  - Um sinal não inicializado não possui valor conhecido, embora simulador possa colocar um valor default (ex.: 0)

Tipos de dados

- Tipos padrão no VHDL

`bit`	`character`	`severity_level`
`bit_vector`	`integer`	`string`
`boolean`	`real`	`time`

- Valores para alguns tipos padrão:
  - boolean: TRUE, FALSE
  - Integer (1, 7, 42): pelo menos 32 bits, positivos e negativos
  - char (‘A’, ‘B’, ‘C’, ...): ISO-8 bits (primeiros 8 bits são ASCII)
  - string (“0101”, “ABCD”): vetor de caracteres
- **NOTA**: perceba uso de **aspas** dependendo do tipo

Tipos de dados

- Tipos mais usados em sistemas digitais:
  - **std_logic:** "bit mais flexível" (Padrão IEEE 1164)
  - **std_logic_vector**: vetor de bits do tipo std_logic
- Podem assumir valores diferentes de 0 ou 1
- '0': 0 lógico
- '1': 1 lógico
- 'U': uninitialized. Valor do sinal não inicializado
- 'X': unknown. Impossível determinar o valor/resultado
- 'Z': alta impedância
- 'W': Weak. Sinal fraco: não é possível dizer se deve ser 0 ou 1.
- 'L': Sinal fraco que provavelmente irá valer 0
- 'H': Sinal fraco que provavelmente irá valer 1
- '-': Don't care.

Tipos de dados

- Exemplos de uso de **std_logic_vector**:

•Declaração:

```
signal vec : std_logic_vector (7 downto 0);
signal v4 : std_logic_vector (3 downto 0);
```

Também possível: `to`

7	6	5	4	3	2	1	0

3	2	1	0

•Atribuição de valores:

```
vec(7) <= '1';
vec(7 downto 5) <= "110";
vec <= "10101010";
vec <= (7 => '1', others => '0');
vec <= (7 | 0 => '1', others => '0');
vec <= (7 downto 3 => '1', others => '0');
```

Separação entre vários itens

'1'							
'1'	'1'	'0'					
'1'	'0'	'1'	'0'	'1'	'0'	'1'	'0'
'1'	'0'	'0'	'0'	'0'	'0'	'0'	'0'
'1'	'0'	'0'	'0'	'0'	'0'	'0'	'1'
'1'	'1'	'1'	'1'	'1'	'0'	'0'	'0'

•Acesso:

```
v4(3) <= vec(7);
v4 <= vec(7 downto 4);
v4(3 downto 2) <= vec(7) & vec(0);
```

concatenação

x				<=	x							
x	y	z	w	<=	x	y	z	w				
x	y			<=	x							y

Usando bibliotecas

```
1  -- library declaration
2  library IEEE;
3  use IEEE.std_logic_1164.all;  -- basic IEEE library
4  use IEEE.numeric_std.all;     -- IEEE library for the unsigned type and
                                 -- various arithmetic operators
6
7  -- WARNING: in general try NOT to use the following libraries
8  --          because they are not IEEE standard libraries
9  -- use IEEE.std_logic_arith.all;
10 -- use IEEE.std_logic_unsigned.all;
11 -- use IEEE.std_logic_signed
12
13 -- entity
14 entity my_ent is
15     port ( A,B,C : in  std_logic;
16            F     : out std_logic);
17 end my_ent;
18 -- architecture
19 architecture my_arch of my_ent is
20     signal v1,v2 : std_logic_vector (3 downto 0);
```

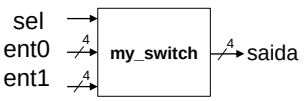
- **`library`**: declara a biblioteca.
- **`use`**: declara quais pacotes serão usados. Ex.: std_logic_1164, todos os componentes contidos neste pacote (all).

Operações padrão

- Dependem do tipo em questão, podendo ser padrão ou definidos pelas bibliotecas usadas. **Exemplos**:
  - Lógicos: **and, or, nand**, **nor, xor, xnor**, **not**
    - Ex.: (A • B) ➔ `A and B;` A' ➔ `not A`
  - Numéricos: **+** , **–** , ***** , **/** , **abs** (*valor absoluto*), ****** (*exponenciação*)
    - Ex.: |val| ➔ `abs val;` X^Y ➔ `X ** Y`
  - Comparação: **=**, **/=** (*diferente*), **<** , **<=**, **>** , **>=**
    - Ex.: A ≠ B ➔ `A /= B`
  - Vetores: **&** (*concatenação*), **rol** (*rotação à esquerda*), **ror** (rotação à direita)
    - Ex.: A concatenado com B ➔ `A & B`
    - Ex.: A rotacionado à direita de 2 posições ➔ `A ror 2`

Exercício/Exemplo

- Escreva a entity para o circuito a seguir:

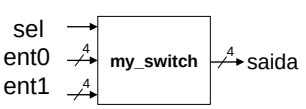


"Cola":

```
entity entity-name is
  port (signal-names : mode signal-type;
        signal-names : mode signal-type;
        ...
        signal-names : mode signal-type);
end entity-name;
```


Exercício/Exemplo

- Escreva a entity para o circuito a seguir:




```
entity my_switch is
port (sel  : in  std_logic;
      ent0 : in  std_logic_vector(3 downto 0);
      ent1 : in  std_logic_vector(3 downto 0);
      saida: out std_logic_vector(3 downto 0));
end my_switch;
```

VHDL: PRINCIPAIS COMANDOS CONCORRENTES

E vários exemplos ilustrativos

Atribuição de Sinais

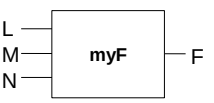
- Atribuição **incondicional**: **<=**
  - Já mostrada anteriormente
- Ex.: Construa um NAND de 3 entradas em VHDL

```
-- library declaration
library IEEE;
use IEEE.std_logic_1164.all;
-- entity
entity my_nand3 is
port (A,B,C : in std_logic;
      F     : out std_logic);
end my_nand3;
-- architecture
architecture impl_nand3 of my_nand3 is
begin
    F <= not (A and B and C);
end impl_nand3;
```

Atribuição de Sinais

- Atribuição **incondicional**: **<=**
- Ex.: Implemente a função F = L'•M'•N + L•M

```
-- library declaration
library IEEE;
use IEEE.std_logic_1164.all;
-- entity
entity myF is
port (L,M,N : in std_logic;
      F     : out std_logic);
end myF;
-- architecture
architecture impl_myF of myF is
begin
    F <= ((not L) and (not M) and N) or (L and M);
end impl_myF;
```



Com 1 só linha

Atribuição de Sinais

- Atribuição **incondicional**: **<=**
- Ex.: Implemente a função F = L'•M'•N + L•M

```
-- library declaration
library IEEE;
use IEEE.std_logic_1164.all;
-- entity
entity myF is
port (L,M,N : in std_logic;
      F     : out std_logic);
end myF;
-- architecture
architecture myFv2 of myF is
    signal A1, A2 : std_logic; -- intermediários
begin
    A1 <= ((not L) and (not M) and N);
    A2 <= (L and M);
    F <= A1 or A2;
end myFv2;
```

Com sinais intermediários

Atribuição de Sinais

- Atribuição **condicional**: **`when`**

```
<alvo> <=    <expressão> when <condição> else
             <expressão> when <condição> else
             <expressão>;
```

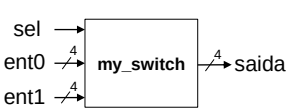
  - Condição: variável booleana ou operação booleana (usando =, /=, >, >=, <, <=)

- Ex.: Implemente a arquitetura de F = L'•M'•N + L•M, usando atribuição condicional

```
architecture myFv3 of myF is
begin
     F <= '1' when (L = '0' and M = '0' and N = '1')    else
          '1' when (L = '1' and M = '1')                else
          '0';
end myFv2;
```


Atribuição de Sinais

- Atribuição **condicional**: **`when`**
- Ex.: Implemente a arquitetura do circuito seletor de 4 bits abaixo




```
entity my_switch is
port (sel         : in  std_logic;
      ent0, ent1 : in  std_logic(3 downto 0);
      saida       : out std_logic(3 downto 0));
end my_switch;

architecture archSel of my_switch is
begin
    saida <= ent0 when (sel = '0') else
             ent1 when (sel = '1') else
             "0000"; -- "catch-all"
end archSel ;
```

Poderia ser omitido, mas assim listam-se todas as possibilidades e cobre-se caso de "valor lógico inválido" ➔ legibilidade + confiabilidade

Atribuição de Sinais

- Atribuição **condicional com seleção**: **`with-select`**
  - Similar a **`when`**, mas usando estilo do "switch-case" em C

```
with <expressão_de_seleção> select
    <alvo>  <=   <expressão> when <valores_seleção>,
                 <expressão> when <valores_seleção>;
```

- Ex.: Implemente a arquitetura de F = L'•M'•N + L•M, usando atribuição condicional com seleção

```
architecture myFvSel of myF is
begin
    with ((L ='0' and M ='0' and N ='1') or (L='1' and M='1')) select
        F <= '1' when '1',
             '0' when '0',
             '0' when others; -- "catch all": útil para legibilidade
end myF;
```

Nota: não é lá uma forma muito interessante de implementar, mas mostra um exemplo funcional do uso de `with-select`…

Atribuição de Sinais

- Atribuição **condicional com seleção**: **`with-select`**

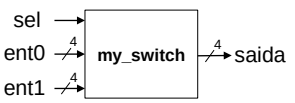
```
with <expressão_de_seleção> select
    <alvo>  <=   <expressão> when <valores_seleção>,
                 <expressão> when <valores_seleção>;
```

- Ex.: Implemente a arquitetura de F = L'•M'•N + L•M, usando atribuição condicional com seleção
  - Também podemos usar um mapa de Karnaugh: F = ∑(1,6,7)

```
architecture myFKarnaugh of myF is
    signal mintermo : std_logic_vector (2 downto 0);
begin
    mintermo <= (L & M & N); -- monta vetor de bits
    with mintermo select
        F <= '1' when "001" | "110" | "111",
             '0' when others; -- "catch all"
end myFKarnaugh;
```

Atribuição de Sinais

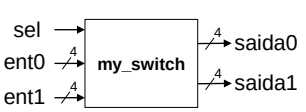
- Atribuição **condicional com seleção**: **`with-select`**
- Ex.: Implemente a arquitetura do circuito seletor de 4 bits abaixo, usando **`with-select`**
  - Envia uma entrada para a saída: ent0 ou ent1




```
architecture archSelv2 of my_switch is
begin
    with sel select
        saida <= ent0    when '0',
                 ent1    when '1',
                "0000"   when others; -- "catch all"
end archSelv2;
```


Atribuição de Sinais

- Atribuição **condicional com seleção**: **`with-select`**
- Ex. (cont): e se "sel" invertesse as saídas?




```
architecture archSelv2 of my_switch is
begin
    with sel select
       saida0 <= ent0    when '0',
                 ent1    when '1',
                "0000"   when others; -- "catch all"
    with sel select
       saida1 <= ent1    when '0',
                 ent0    when '1',
                "0000"   when others; -- "catch all"
end archSelv2;
```

Um só alvo por `with-select` !!!

Atribuição de Sinais

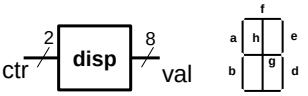
- Atribuição **condicional com seleção**: **`with-select`**
- Ex.: Implemente a arquitetura do verificador de faixas de magnitude de 4 bits abaixo, usando **`with-select`**

entrada	saida
0000 a 0011	100
0100 a 1001	010
1010 a 1111	001
valor desconhecido	000

```
architecture arch of checkMag is
begin
    with entrada select
        saida <= "100"   when "0000"|"0001"|"0010"|"0011",
                 "010"   when "0100"|"0101"|"0110"|"0111"|"1000"|"1001",
                 "001"   when "1010"|"1011"|"1100"|"1101"|"1110"|"1111",
                 "000"   when others; -- "catch all"
end arch;
```


Atribuição de Sinais

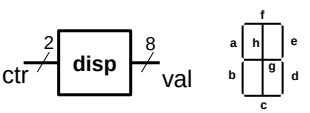
- Atribuição **condicional com seleção**: **`with-select`**
- Ex.: Impressão de “P” “O” “L” “I” em display
  - P = abc’d’efg’h ; O = abcdefg’h’ ;
  - L = abc(defgh)’ ; I = (abcdefg)’h




```
entity disp is
port (ctr  : in  std_logic_vector (1 downto 0);
      val  : out std_logic_vector (7 downto 0));
end disp;

architecture mydisp of disp is
...
```

Atribuição de Sinais

- Atribuição **condicional com seleção**: **`with-select`**
- Ex.: Impressão de “P” “O” “L” “I” em display
  - P = abc’d’efg’h ; O = abcdefg’h’ ;
  - L = abc(defgh)’ ; I = (abcdefg)’h




```
...
architecture selector of disp is
begin
  with (ctr) select
    val <= "11001101" when "00",    -- "P"
    val <= "11111100" when "01",    -- "O"
    val <= "11100000" when "10",    -- "L"
    val <= "00000001" when "11",    -- "I"
    val <= "00000010" when others; -- "catch all": traço "-"
end selector;
```


VHDL: ATRIBUIÇÕES SEQUENCIAIS (PROCESSOS)

E estilos de projeto

Estilos de projeto em VHDL

- Diferentes estilos para descrever arquitetura (parte "executável" do componente)
  - **Estrutural:** portas e suas conexões
    - Síntese mais controlada por projetista: útil para combinar componentes prontos (blocos com entradas e saídas)
  - **Fluxo de dados**: relação entre entradas e saídas
    - Controle de síntese médio: para projetos pequenos a médios
    - Usado em todos os exemplos anteriores...
  - **Comportamental**: abordagem mais alto nível
    - Síntese deixada para o compilador: uso de **`process`**
  - Projetos reais muitas vezes **combinam** os estilos...

VHDL Comportamental: processos

- Bloco contendo instruções **sequenciais**: **`process`**
  - Embora **"sequencial" para projetista**, sintetizador cria **hardware concorrente**...
    - Bloco deve ser **simples**, ou hardware fica complexo e lento...
  - Processos podem ser "**ativados**" em resposta a sinais: permite construção de circuitos sequenciais (foco de PCS3225)

```
label: process (signal1, signal2, ... , signalN)
   <variaveis_locais>
begin   -- exec. sequencial
   <comando_sequencial>
   ...
   <comando_sequencial>
end process label;
```

Ex.: `variable var;`
➔*processos não podem declarar "sinais locais"*

Lista de sensibilidade: processo executa sempre que algum sinal da lista mudar de valor

VHDL Comportamental: processos

- Bloco contendo instruções **sequenciais**: `process`
  - **Diferentes processos** executam **concorrentemente**
  - **Processos** executam **concorrentemente** com **outros comandos**

CONCORRENTES

(1)
```
label1: process (<lista_sensibilidade>)
    <variaveis_locais>
begin   -- exec. sequencial
    <comandos_sequenciais> -- mantenha SIMPLES!!!
end process label1;
```

(2)
```
label2: process (<lista_sensibilidade>)
     <variaveis_locais>
begin   -- exec. sequencial
    <comandos_sequenciais> -- mantenha SIMPLES!!!
end process label2;
```

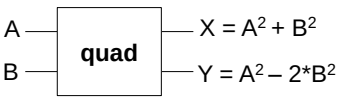
(3)
```
A <= B or C;
```


Ex.: Fluxo de dados vs. Comportamental


```
-- library declaration
library IEEE;
use IEEE.std_logic_1164.all;
-- entity
entity my_xor is
port (A,B   : in std_logic;
      F     : out std_logic);
end my_xor;
-- architecture: fluxo de dados
architecture dflow of my_xor is
begin
    F <= A xor B;
end dflow;
```

```
-- library declaration
library IEEE;
use IEEE.std_logic_1164.all;
-- entity
entity my_xor is
port (A,B   : in std_logic;
      F     : out std_logic);
end my_xor;
-- architecture: comportamental
architecture behav of my_xor is
begin
    xor_proc: process(A,B)
    begin
        F <= A xor B;
    end process xor_proc;
end behav;
```

Ex.: Fluxo de dados vs. Comportamental

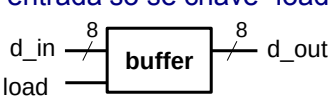



```
-- entity
entity quad is
port (A,B  : in integer;
      X,Y  : out integer);
end quad;
-- architecture: fluxo de dados
architecture fdados of quad is
begin
    X <= A**2 + B**2;
    Y <= A**2 - 2*(B**2);
end fdados;
```

```
-- entity: IDÊNTICA
           ...
-- architecture: comportamental
architecture comp of quad is
begin
    squares: process(A,B)
        variable a2,b2 : integer;
    begin
        a2  := A**2; -- exemplo de
        b2  := B**2; -- "sequência"
        X   <= a2 + b2;
        b2  := 2*b2;
        Y   <= a2 - b2;
    end process squares;
end comp;
```


Processos: lista de sensibilidade

- Lista de sensitividade pode conter algumas entradas
  - Ex.: saída recebe entrada só se chave "load" mudar de valor




```
entity buffer is
port (load : in std_logic;
      d_in  : in std_logic_vector (7 downto 0);
      d_out : out std_logic_vector (7 downto 0));
end buffer;
architecture impl of buffer is
begin
    loader: process(load) -- sensível apenas a "load"
    begin
        d_out <= d_in;     -- se load "0 → 1" ou "1 → 0"
    end process loader;
end impl;
```

Processos: atribuição condicional

- Atribuição condicional: **`if`**
  - Obs: parênteses é opcional

```
if (condição) then
    <expressão>
elsif (condição) then
    <expressão>
else
    <expressão>
end if;
```

- Exemplo:

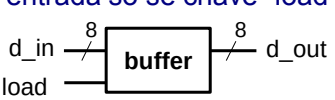
  F = A•B'•C' + B•C

```
entity func is -- entity
port (A,B,C : in std_logic;
      F     : out std_logic);
end func;
```

```
architecture arch1 of func is
begin
  proc: process(A,B,C)
  begin
    if (A='1' and B='0' and C='0') then
        F <= '1';
    elsif (B='1' and C='1') then
        F <= '1';
    else    -- perceba que não tem "then"
        F <= '0';
    end if;
  end process proc;
end arch1;
```


Processos: lista de sensibilidade

- Atribuição condicional: **`if`**
  - Ex.: saída recebe entrada só se chave "load" mudar para 1




```
entity buffer is
port (load : in std_logic;
      d_in  : in std_logic_vector (7 downto 0);
      d_out : out std_logic_vector (7 downto 0));
end buffer;
architecture impl of buffer is
begin
    loader: process(load) -- sensível apenas a "load"
    begin -- se load "0 →1"
        if (load='1') then d_out <= d_in; end if;
    end process loader;
end impl;
```

Processos: atribuição condicional

- Atribuição condicional: **`if`**

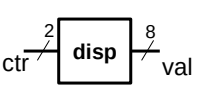
```
if (condição) then
    <expressão>
elsif (condição) then
    <expressão>
else
    <expressão>
end if;
```

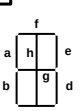
- Exemplo (uma alternativa): F = A•B'•C' + B•C

```
architecture arch2 of func is
begin
  proc: process(A,B,C)
  begin
    if (A='1' and B='0' and C='0') or (B='1' and C='1') then
        F <= '1';
    else            -- perceba que não tem "then"
        F <= '0';
    end if;
  end process proc;
end arch2;
```


Processos: atribuição condicional

- Atribuição condicional: **`if`**
- Ex.: Impressão de "P" "O" "L" "I" em display
  - P = abc'd'efg'h ; O = abcdefg'h' ;
  - L = abc(defgh)' ; I = (abcdefg)'h

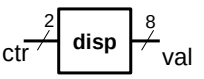


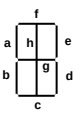



```
entity disp is
port (ctr  : in  std_logic_vector (1 downto 0);
      val  : out std_logic_vector (7 downto 0));
end disp;

architecture mydisp of disp is
...
```

Processos: atribuição condicional

- Atribuição condicional: **`if`**
- Ex.: Impressão de “P” “O” “L” “I” em display
  - P = abc’d’efg’h ; O = abcdefg’h’ ;
  - L = abc(defgh)’ ; I = (abcdefg)’h






```
...
architecture mydisp of disp is
begin
  printer: process(ctr)
  begin
    if      (ctr = "00") then  val <= "11001101"; -- "P"
    elsif   (ctr = "01") then  val <= "11111100"; -- "O"
    elsif   (ctr = "10") then  val <= "11100000"; -- "L"
    elsif   (ctr = "11") then  val <= "00000001"; -- "I"
    else                       val <= "00000010"; -- catch all: "-"
    end if;
  end process printer;
end mydisp;
```


Processos: atribuição condicional

- Atribuição condicional com seleção: **`case`**

```
case <expressão_de_seleção> is
    when escolhas =>
        <expressões_sequenciais>
    when escolhas =>
        <expressões_sequenciais>
    when others =>
         <expressões_sequenciais>
end case;
```

- Exemplo: F = A•B’•C’ + B•C

```
entity func is -- entity
port (A,B,C : in std_logic;
      F     : out std_logic);
end func;
```

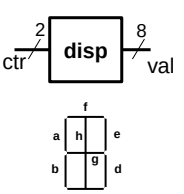
Processos: atribuição condicional

- Atribuição condicional com seleção: **`case`**
- Exemplo: F = A•B'•C' + B•C

```
architecture arch3 of func is
    signal comboABC: std_logic_vector (2 downto 0);
begin
    comboABC <= A & B & C; -- agrupamento de sinais para o case
    func_proc: process(comboABC)
    begin
        case (comboABC) is
            when "100"  => F <= '1';
            when "011"  => F <= '1';
            when "111"  => F <= '1';
            when others => F <= '0';
        end case;
    end process func_proc;
end arch3;
```

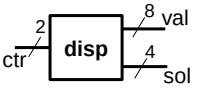

Processos: atribuição condicional

- Atribuição condicional com seleção: **`case`**
- Ex.: Impressão de "P" "O" "L" "I" em display
  - P = abc'd'efg'h ; O = abcdefg'h' ;
  - L = abc(defgh)' ; I = (abcdefg)'h

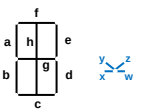



```
...
architecture mydisp2 of disp is
begin
  printer: process(ctr)
  begin
    case (ctr) is
        when "00"   =>  val <= "11001101"; -- "P"
        when "01"   =>  val <= "11111100"; -- "O"
        when "10"   =>  val <= "11100000"; -- "L"
        when "11"   =>  val <= "00000001"; -- "I"
        when others =>  val <= "00000010"; -- catch all: "-"
    end case;
  end process printer;
end mydisp2;
```

Processos: atribuição condicional



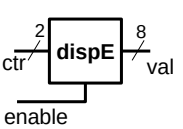
- Atribuição condicional com seleção: **`case`**
- Ex.: Impressão de “P” “O” “L” “I” em display
  - Com “sol piscante” extra



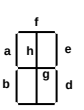

```
...
architecture mydisp3 of disp2 is
begin
  printer: process(ctr)
  begin
    case (ctr) is
      when "00"   => val <= "11001101"; sol <= "1000"; -- "P" , x
      when "01"   => val <= "11111100"; sol <= "0100"; -- "O" , y
      when "10"   => val <= "11100000"; sol <= "0010"; -- "L" , z
      when "11"   => val <= "00000001"; sol <= "0001"; -- "I" , w
      when others => val <= "00000010"; sol <= "0000"; -- catch all: "-"
    end case;
  end process printer;
end mydisp3;
```

Pode haver mais de um comando por `when`

Processos: atribuição condicional



- Atribuição condicional com seleção: **`case`**
- Ex.: Impressão de “P” “O” “L” “I” em display
  - Com sinal de habilitação (enable)




```
...
architecture archE of dispE is
begin
  printer: process(ctr)
  begin
    if (enable = '1') then
        case (ctr) is
            when "00"    =>  val <= "11001101"; -- "P"
            when "01"    =>  val <= "11111100"; -- "O"
            when "10"    =>  val <= "11100000"; -- "L"
            when "11"    =>  val <= "00000001"; -- "I"
            when others  =>  val <= "00000010"; -- catch all: "-"
        end case;
    else val <= "00000000"; -- apagado se enable desativado
    end if;
  end process printer;
end archE;
```

VHDL: PROJETO USANDO ABORDAGEM ESTRUTURAL

Estilos de projeto em VHDL (bis)

- Diferentes estilos para descrever arquitetura
  - **Fluxo de dados**: relação entre entradas e saídas
  - **Comportamental**: abordagem mais alto nível
    - Síntese deixada para o compilador: uso de `process`
  - **Estrutural:** portas e suas conexões
    - Síntese mais controlada por projetista: útil para combinar **componentes prontos**
    - Abordagem **modular:** definição de blocos, ligando suas entradas e saídas
      - Ideia similar a criar funções em C, ou objetos em Java
    - Costuma ser combinado com **bibliotecas gráficas**: componentes são ligados por meio de fios em vez de código

VHDL Estrutural

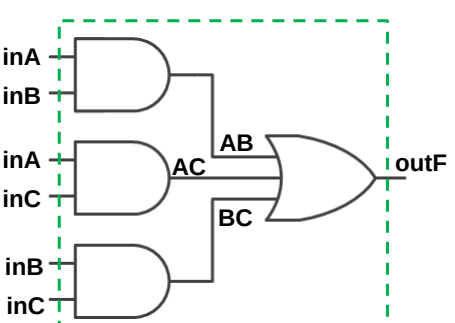
- Elemento principal: **`component`**
- Para facilitar, façamos um paralelo entre VHDL e C

Código em C	VHDL
Descrever interface da função	Definir a **`entity`**
Escrever o código da função	Definir a **`architecture`**
Definir a interface no programa main (diretamente ou via biblioteca.h)	Declarar **`component`** (diretamente ou via biblioteca)
Fazer chamadas à função , passando variáveis de entrada e saída	Mapear **`component`** em sinais de entrada e saída

- Vamos mostrar um exemplo: “maioria entre A, B e C”
  - F = A•B + A•C + B•C

VHDL Estrutural

- Vamos mostrar um exemplo: “maioria entre A, B e C”
  - Sinais de entrada: inA, inB e inC; Sinais de saída: outF
  - Componentes internos: portas lógicas AND2 e OR3
  - Sinais internos: AB, AC, BC



VHDL Estrutural

- Vamos mostrar um exemplo: “maioria entre A, B e C”
- Passo 0
- Componentes internos: portas lógicas AND2 e OR3
  - Em **ferramentas de desenvolvimento**: devem ser **incluídas no projeto** para serem visíveis pelo componente
  - Obs.: **Portas lógicas padrão** costumam estar **disponíveis em bibliotecas de fabricantes**

```
-- entidade: interface
entity and2 is
port (A,B : in std_logic;
        F : out std_logic);
end and2;
-- arquitetura: funcionamento
architecture archA of and2 is
begin
    F <= (A and B);
end archA;
```

```
-- entidade: interface
entity or3 is
port (A,B,C : in std_logic;
          F : out std_logic);
end or3;
-- arquitetura: funcionamento
architecture archO of or3 is
begin
    F <= (A or B or C);
end archO;
```


VHDL Estrutural

- Vamos mostrar um exemplo: “maioria entre A, B e C”
- Passo 1
- Passo 1: definir entidade (interface de alto nível)
  - Sinais de entrada: inA, inB e inC; Sinais de saída: outF

```
-- entidade: interface
entity maioria is
port (inA,inB,inC : in std_logic;
                outF : out std_logic);
end maioria;
```

VHDL Estrutural

- Vamos mostrar um exemplo: “maioria entre A, B e C”

Passo 2

- Passo 2: declarar componentes usados pela entidade
  - Componentes internos : AND2 e OR3

```
-- entidade
entity and2 is
port (A,B : in std_logic;
        F : out std_logic);
end and2;
```

⇨

```
-- componente
component and2 is
port (A,B : in std_logic;
        F : out std_logic);
end component;
```

```
-- entidade
entity or3 is
port (A,B,C : in std_logic;
          F : out std_logic);
end or3;
```

⇨

```
-- componente
component or3 is
port (A,B,C : in std_logic;
          F : out std_logic);
end component;
```

(obs: criados no passo 0)

VHDL Estrutural

- Vamos mostrar um exemplo: “maioria entre A, B e C”

Passo 2

- Passo 2: declarar componentes usados pela entidade

```
entity maioria is                    -- entidade
port (inA,inB,inC : in std_logic;
              outF : out std_logic);
end maioria;

architecture arch of maioria is    -- arquitetura
   component or3 is     -- componente: OR3
   port (A,B,C : in std_logic; F : out std_logic);
   end component;

   component and2 is   -- componente: AND2
   port (A,B : in std_logic; F : out std_logic);
   end component;
    ...
begin
    ...
```

Antes do “begin”: escopo é local

VHDL Estrutural

- Vamos mostrar um exemplo: “maioria entre A, B e C”
- Passo 3: declarar sinais internos usados pela entidade
  - Sinais internos: AB, AC, BC

```
entity maioria is                    -- entidade
port (inA,inB,inC : in std_logic;
              outF : out std_logic);
end maioria;

architecture arch of maioria is   -- arquitetura

    component or3 is     -- componente: OR3
    port (A,B,C : in std_logic; F : out std_logic);
    end component;

    ...
    signal AB, AC, BC : std_logic;
begin
    ...
```

Também antes do “begin”: escopo é local

VHDL Estrutural

- Vamos mostrar um exemplo: “maioria entre A, B e C”
- Passo 4: instanciar e mapear componentes em sinais

label: *component-name* port map(*signal1*, *signal2*, ..., *signaln*);

label: *component-name* port map(*port1*=>*signal1*, *port2*=>*signal2*, ..., *portn*=>*signaln*);

- Exemplo:


```
component and2 is
port (A,B  : in std_logic;
      F    : out std_logic);
end component;
signal AB, ... : std_logic;
```

mapeamento explícito: sinais em qualquer ordem

```
xAB: and2 port map (  A => inA,
                      B => inB,
                      F => AB);
```

OU: mapeamento implícito: ordem específica

```
xAB: and2 port map (inA, inB, AB);
```

VHDL Estrutural

- Vamos mostrar um exemplo: “maioria entre A, B e C”

```
entity maioria is                                        -- entidade
port (inA,inB,inC : in std_logic;  outF : out std_logic);
end maioria;

architecture arch of maioria is                          -- arquitetura
    component or3 is    -- componente: OR3
    port (A,B,C : in std_logic; F : out std_logic);
    end component;

    component and2 is   -- componente: AND2
    port (A,B : in std_logic; F : out std_logic);
    end component;

    signal AB, AC, BC : std_logic;  -- sinais internos

begin                                -- mapeamentos explícitos
    xAB: and2 port map (A => inA, B => inB, F => AB);
    xAC: and2 port map (A => inA, B => inC, F => AC);
    xBC: and2 port map (A => inB, B => inC, F => BC);
    xF: or3 port map (A => AB, B => AC, C => BC, F => outF);
end arch;
```

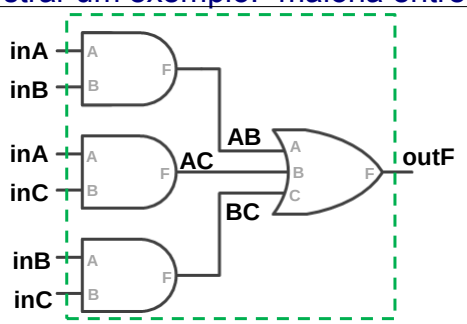

VHDL Estrutural

- Vamos mostrar um exemplo: “maioria entre A, B e C”

```
entity maioria is                                        -- entidade
port (inA,inB,inC : in std_logic;  outF : out std_logic);
end maioria;

architecture arch of maioria is                          -- arquitetura
    component or3 is    -- componente: OR3
    port (A,B,C : in std_logic; F : out std_logic);
    end component;

    component and2 is   -- componente: AND2
    port (A,B : in std_logic; F : out std_logic);
    end component;

    signal AB, AC, BC : std_logic;  -- sinais internos

begin                                -- mapeamentos implícitos
    xAB: and2 port map (inA, inB, AB);
    xAC: and2 port map (inA, inC, AC);
    xBC: and2 port map (inB, inC, BC);
    xF: or3 port map (AB, AC, BC, outF);
end arch;
```

VHDL Estrutural

- Vamos mostrar um exemplo: “maioria entre A, B e C”




```
begin                                          -- mapeamentos explícitos
    xAB: and2 port map (A => inA, B => inB, F => AB);
    xAC: and2 port map (A => inA, B => inC, F => AC);
    xBC: and2 port map (A => inB, B => inC, F => BC);
    xF: or3 port map (A => AB, B => AC, C => BC, F => outF);
end arch;
```

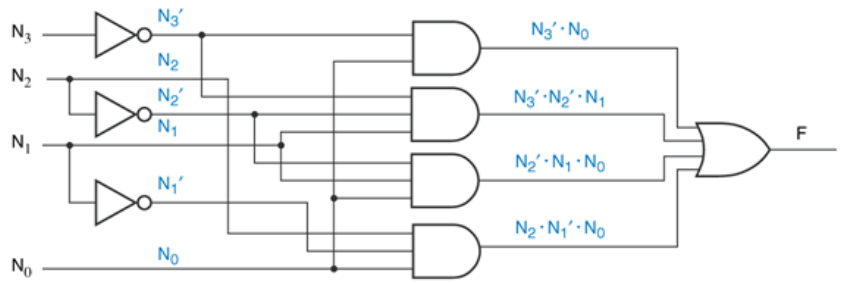

VHDL Estrutural (exemplo no Wakerly)

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

entity prime is
    port ( N: in STD_LOGIC_VECTOR (3 downto 0);
           F: out STD_LOGIC );
end prime;

architecture prime1_arch of prime is
signal N3_L, N2_L, N1_L: STD_LOGIC;
signal N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0: STD_LOGIC;
component INV port (I: in STD_LOGIC; O: out STD_LOGIC); end component;
component AND2 port (I0,I1: in STD_LOGIC; O: out STD_LOGIC); end component;
component AND3 port (I0,I1,I2: in STD_LOGIC; O: out STD_LOGIC); end component;
component OR4 port (I0,I1,I2,I3: in STD_LOGIC; O: out STD_LOGIC); end component;
begin
  U1: INV port map (N(3), N3_L);
  U2: INV port map (N(2), N2_L);
  U3: INV port map (N(1), N1_L);
  U4: AND2 port map (N3_L, N(0), N3L_N0);
  U5: AND3 port map (N3_L, N2_L, N(1), N3L_N2L_N1);
  U6: AND3 port map (N2_L, N(1), N(0), N2L_N1_N0);
  U7: AND3 port map (N(2), N1_L, N(0), N2_N1L_N0);
  U8: OR4 port map (N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0, F);
end prime1_arch;
```

Ligação entre entradas/saídas e componentes e sinais

VHDL Estrutural (exemplo no Wakerly)

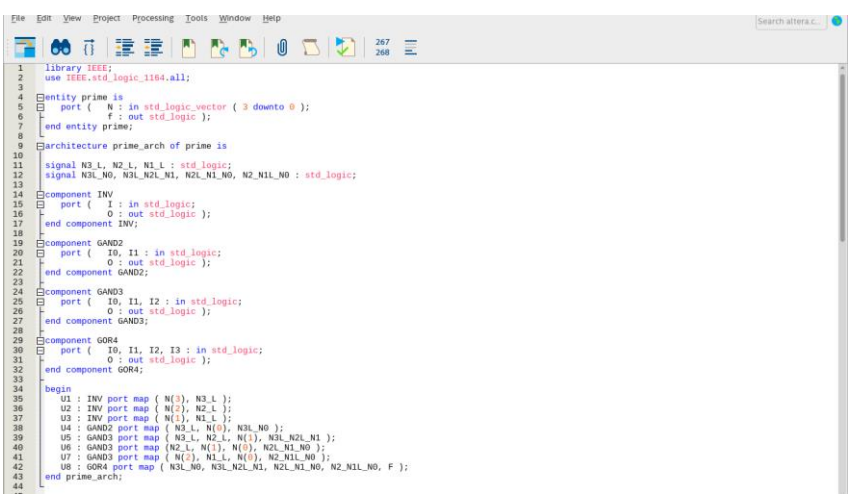
(c)



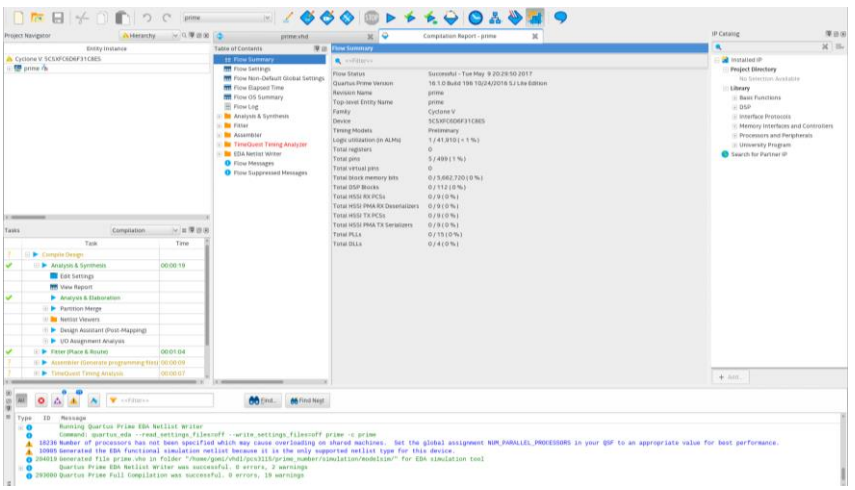

```
begin
  U1: INV port map (N(3), N3_L);
  U2: INV port map (N(2), N2_L);
  U3: INV port map (N(1), N1_L);
  U4: AND2 port map (N3_L, N(0), N3L_N0);
  U5: AND3 port map (N3_L, N2_L, N(1), N3L_N2L_N1);
  U6: AND3 port map (N2_L, N(1), N(0), N2L_N1_N0);
  U7: AND3 port map (N(2), N1_L, N(0), N2_N1L_N0);
  U8: OR4 port map (N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0, F);
end prime1_arch;
```

Ligação entre entradas/saídas e componentes e sinais

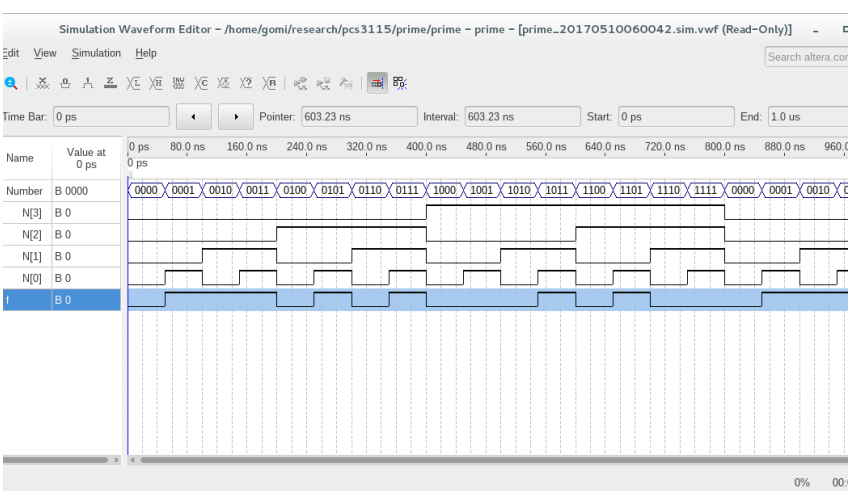
Altera Quartus – Editor VHDL



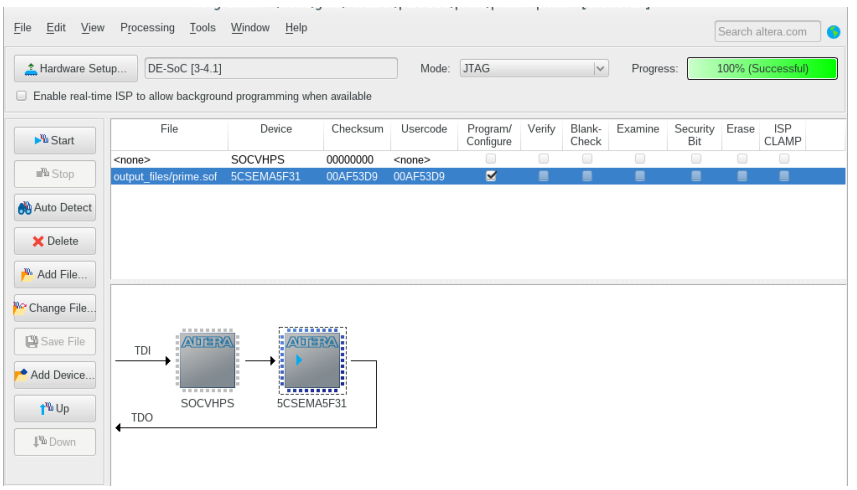
Quartus – “Compilação” (Síntese)



Quartus – Simulação (ModelSim)



Quartus – Gravação na FPGA



CONCLUSÕES

Dicas

- Site LabDigital (http://www.pcs.usp.br/~labdig/)
  - Quartus: software para projetar usando VHDL
    - Nome da entidade deve ser o nome do arquivo
  - Material de apoio
    - Tutorial de como simular circuitos
    - Apostilas de VHDL
- Observações
  - Evite process por enquanto
  - Evite estruturas sequenciais (e.g., if-then-else, case-when)

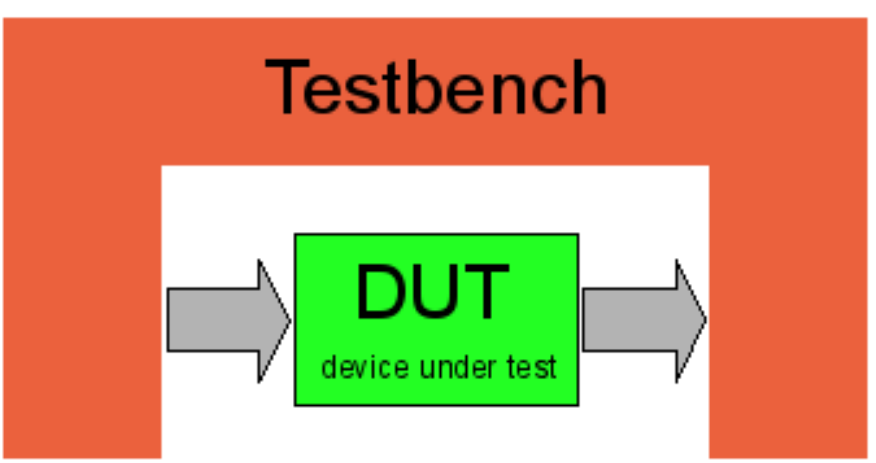
Referências

- Free Range VHDL: http://freerangefactory.org/books_tuts.html
  - 1-2: leitura recomendada
  - 3-5: leitura obrigatória
  - 6: até 6.7: leitura obrigatória
  - 8: leitura obrigatória, ignorar parte sequencial
  - 10: leitura recomendada
  - Apendice B: leitura obrigatória
- Capítulo 5 do Wakerly
  - Obrigatório: 5 e 5.1 (todas subseções): introdução
  - Obrigatório: 5.3: VHDL
  - Recomendável: 5.3.3 em diante
  - **Exercícios**: 5.3 a 5.10, 5.23 a 5.32

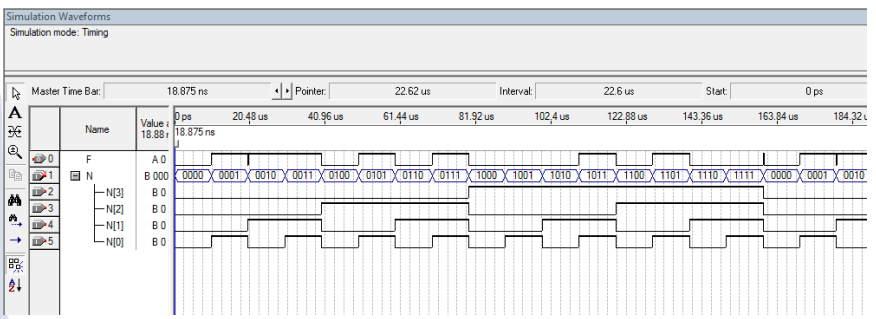
APÊNDICE 1

Um pouco de “feeling” do LabDigital

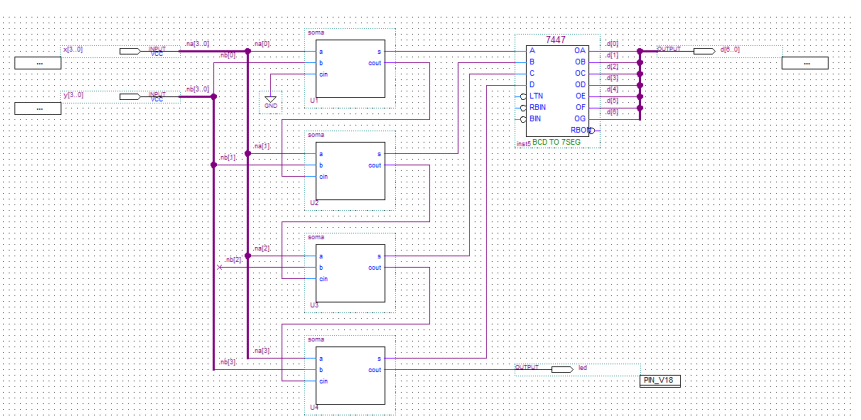
Bancada de Testes



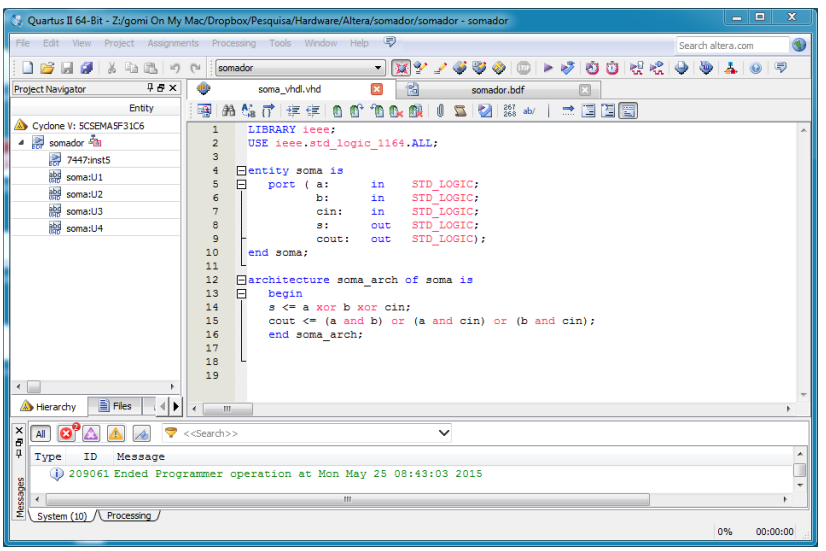
Simulação – Quartus



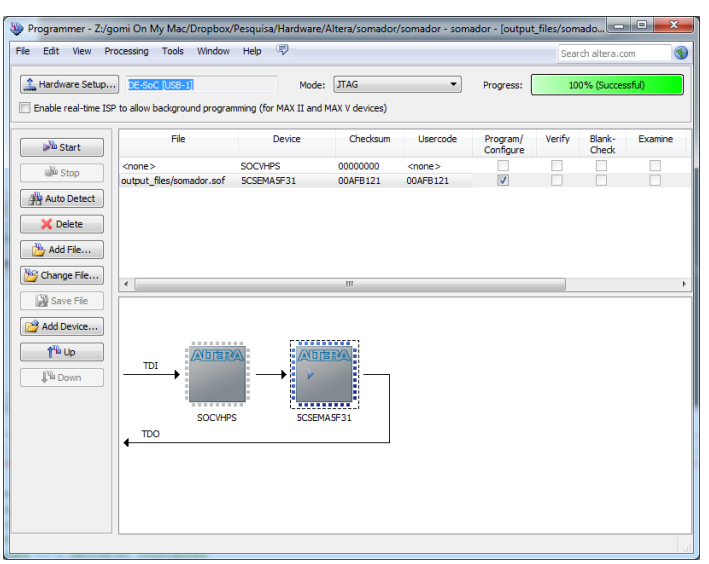
Somador Completo de 4 bits



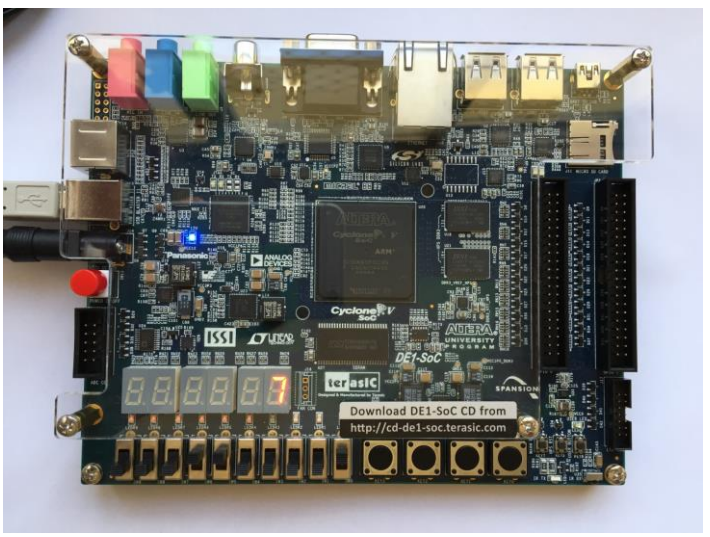
Somador Completo de 1 bit



Gravação na FPGA



Kit FPGA Altera



APÊNDICE 2

VHDL: detalhes adicionais

Tipos de dados

- **Tipos definidos por usuário** podem enumerar valores :

  Sintaxe: `type type-name is (value-list);`

  Exemplo: `type semaforo is (reset, stop, wait, go);`

- Para facilitar, também podem ser usados **arrays**:

```
type type-name is array (start to end) of element-type;
type type-name is array (start downto end) of element-type;
```

Ex.:
```
type databus is array (15 downto 0) of BIT;
type T_2D is array (0 to 1, 0 to 1) of integer;
```

Tipos de dados

- **Sub-tipos e constantes** também podem ser definidos

```
type type-name is (value-list);
subtype subtype-name is type-name range start to end;
subtype subtype-name is type-name range start downto end;
constant constant-name: type-name := value;
```

Ex.:
```
subtype bits8 is integer range 0 to 255;
subtype digit is character range '0' to '9';
constant BUS_SIZE : integer := 32;
constant PERIOD : time := 10 ns;
```

Tipos de dados

- “bit mais flexível” : **std_logic** (Padrão IEEE 1164)
- “resolved”: força barramento tri-state caso mais de um componente gere o mesmo sinal (= saída no mesmo fio)

```
type STD_ULOGIC is ( 'U',  -- Uninitialized
                     'X',  -- Forcing  Unknown
                     '0',  -- Forcing  0
                     '1',  -- Forcing  1
                     'Z',  -- High Impedance
                     'W',  -- Weak     Unknown
                     'L',  -- Weak     0
                     'H',  -- Weak     1
                     '-'   -- Don't care
                   );
subtype STD_LOGIC is resolved STD_ULOGIC;

type STD_LOGIC_VECTOR is array (natural range <>) of STD LOGIC
                     -- array sem tamanho máximo fixado
```


Funções

- Semelhantes a funções em linguagens procedurais
  - Tomam argumentos como entrada, retornam um resultado

“variáveis locais”

operações

```
function function-name (
      signal-names : signal-type;
      signal-names : signal-type;
      ...
      signal-names : signal-type
) return return-type is
   type declarations
   constant declarations
   variable declarations
   function definitions
   procedure definitions
begin
   sequential-statement
   ...
   sequential-statement
end function-name;
```

Funções: exemplo (“but not”)

```
entity Inhibit is     -- also known as 'BUT-NOT'
  port (X,Y: in BIT;     --  as in 'X but not Y'
        Z:   out BIT);   --  (see [Klir, 1972])
end Inhibit;

architecture Inhibit_arch of Inhibit is
begin
  Z <= '1' when X='1' and Y='0' else '0';
end Inhibit_arch;
```

Implementação sem função

```
architecture Inhibit_archf of Inhibit is

function ButNot (A, B: bit) return bit is
begin
  if B = '0' then return A;
  else return '0';
  end if;
end ButNot;
```

Declarando função (comandos sequenciais)

```
begin
  Z <= ButNot(X,Y);
end Inhibit_archf;
```

Implementação com função