

# PCS 3115

## Sistemas Digitais I

### VHDL

*Prof. Dr. Marcos A. Simplicio Jr.*

*versão: 4.0 (Jan/2017)*

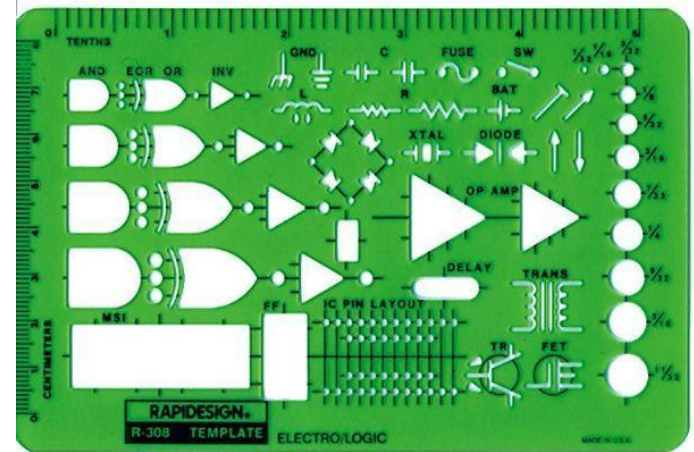
*Adaptado por Glauber (2018)*

# Referências

- Free Range VHDL: [http://freerangefactory.org/books\\_tuts.html](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 40 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

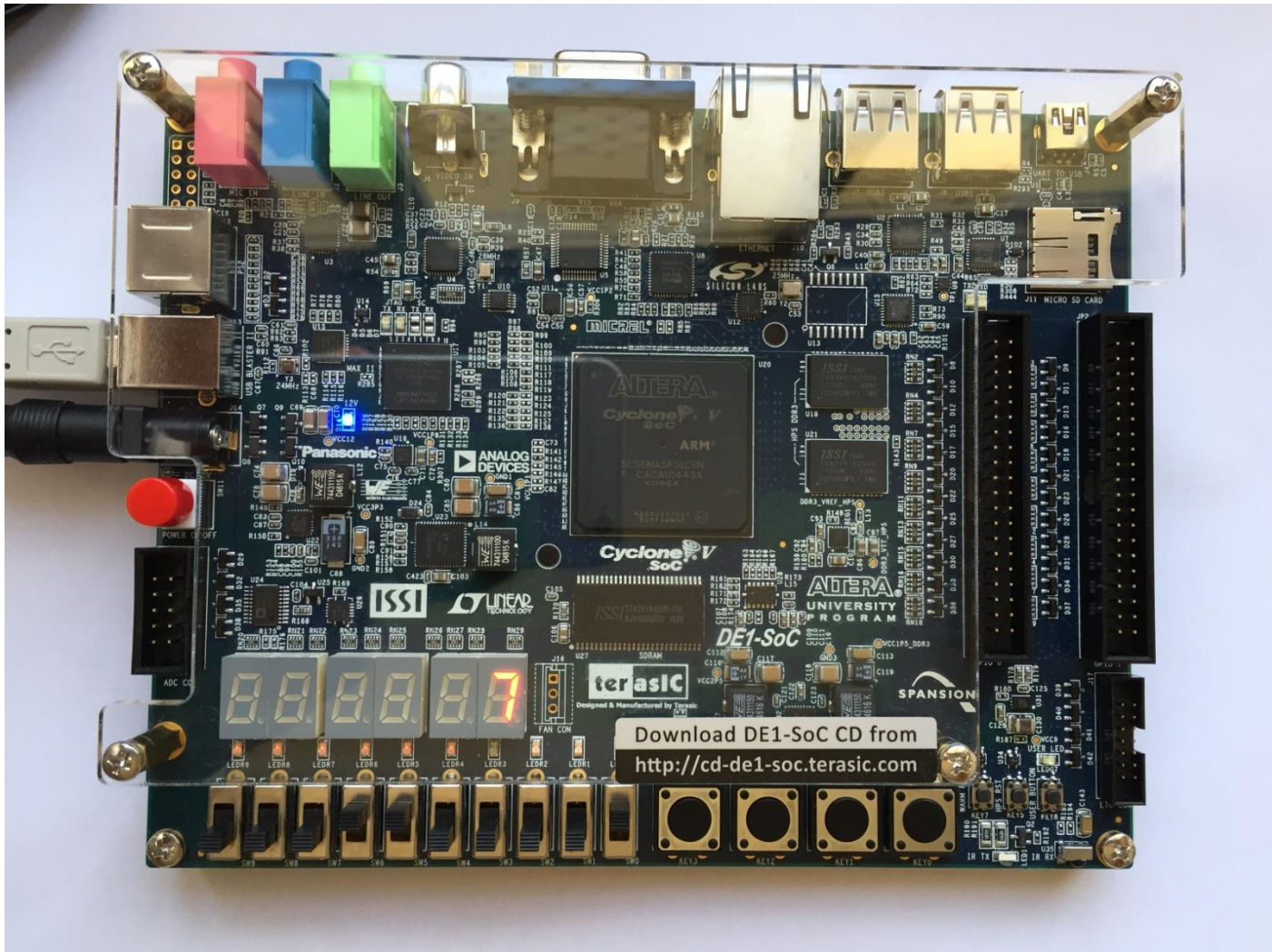
# HDL (*Hardware Description Language*)

- É uma linguagem para
  - Descrever (**modelar**) e **documentar** um sistema digital em qualquer nível de abstração.
  - **Simular** (testar) e **sintetizar** um sistema digital
- **VHDL: Very High Speed Integrated Circuits HDL**
  - 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

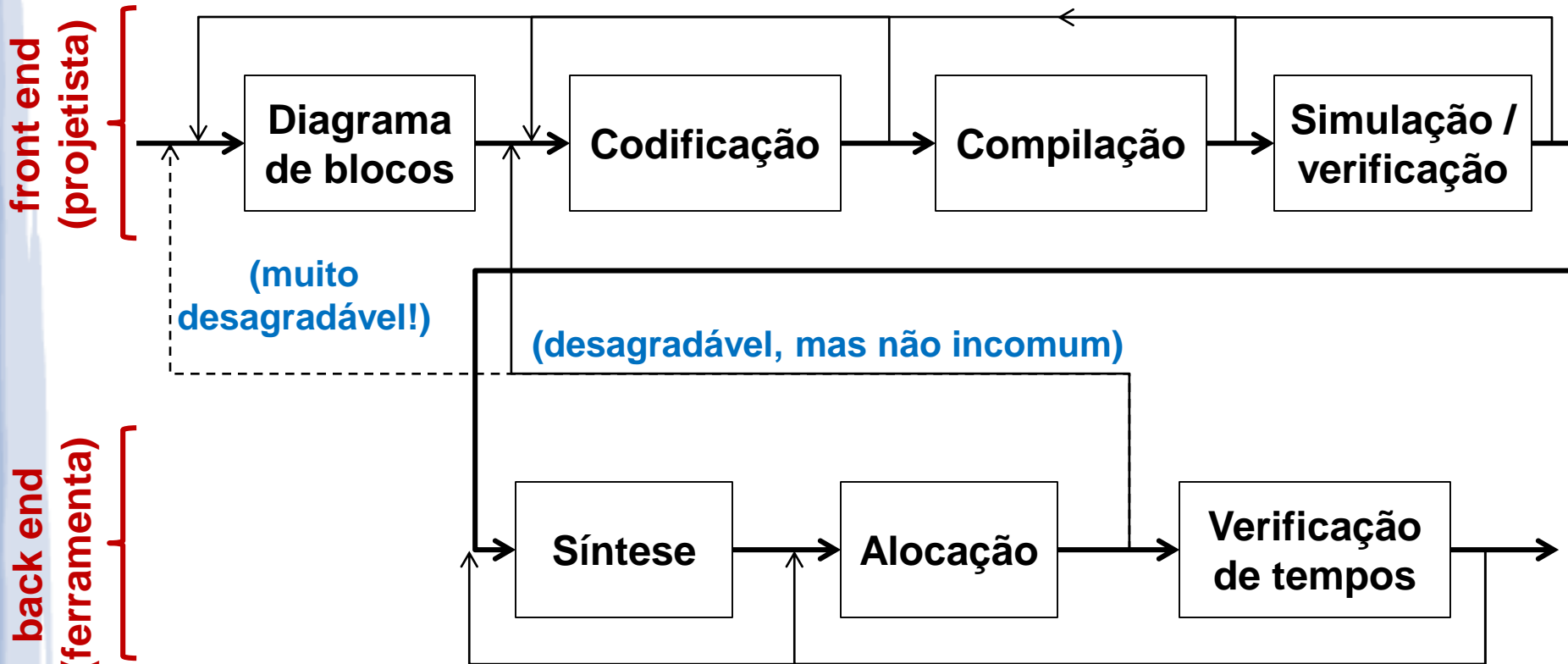
- Diversas **ferramentas** compõem um ambiente HDL
  - **Editor de texto**: para escrever diretamente em HDL
  - **Compilador**: verificação e validação do programa, permitindo identificar erros de sintaxe
  - **Sintetizador**: transforma o projeto compilado no circuito voltado a uma tecnologia de hardware 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

# Kit FPGA Altera





# 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 ";"
```

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

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

Equivalentes

# 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:

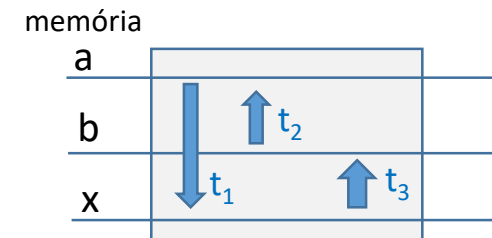
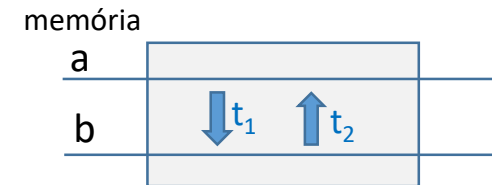
```
1: a = b;  
2: b = a;
```

Em C, isso funciona:

```
1: x = a;  
2: a = b;  
3: b = x
```

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:

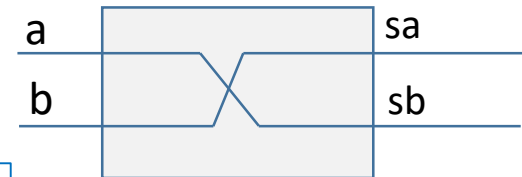
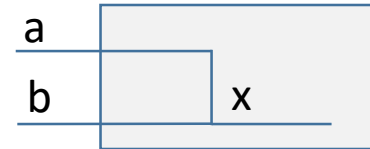
```
1: x <= a;  
2: a <= b;  
3: b <= x;
```

Em VHDL, isso funciona:

```
1: sa <= b;  
2: sb <= a;
```

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

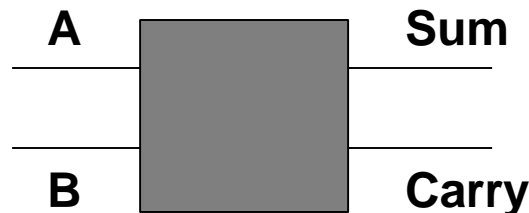
tempo	memória		
	a	b	x
0	5	7	*
1,2,3	??	??	??



# **VHDL: ENTIDADE E ARQUITETURA**

# Estrutura de componentes em VHDL

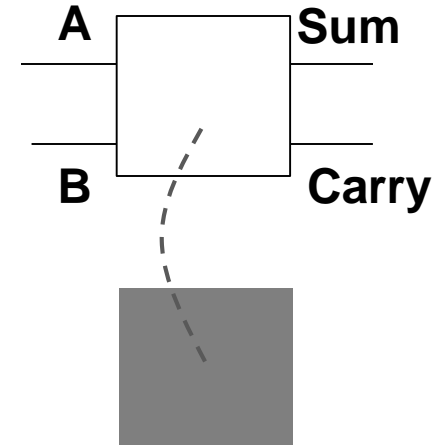
- Estrutura **modularizada**:
  - Definem-se componentes como “**caixas-pretas**”: com **entradas e saídas**
  - Define-se o **conteúdo** da caixa preta
  - Caixas pretas podem ser **reutilizadas**
- ➔ Semelhante a “funções” na linguagem C





# 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

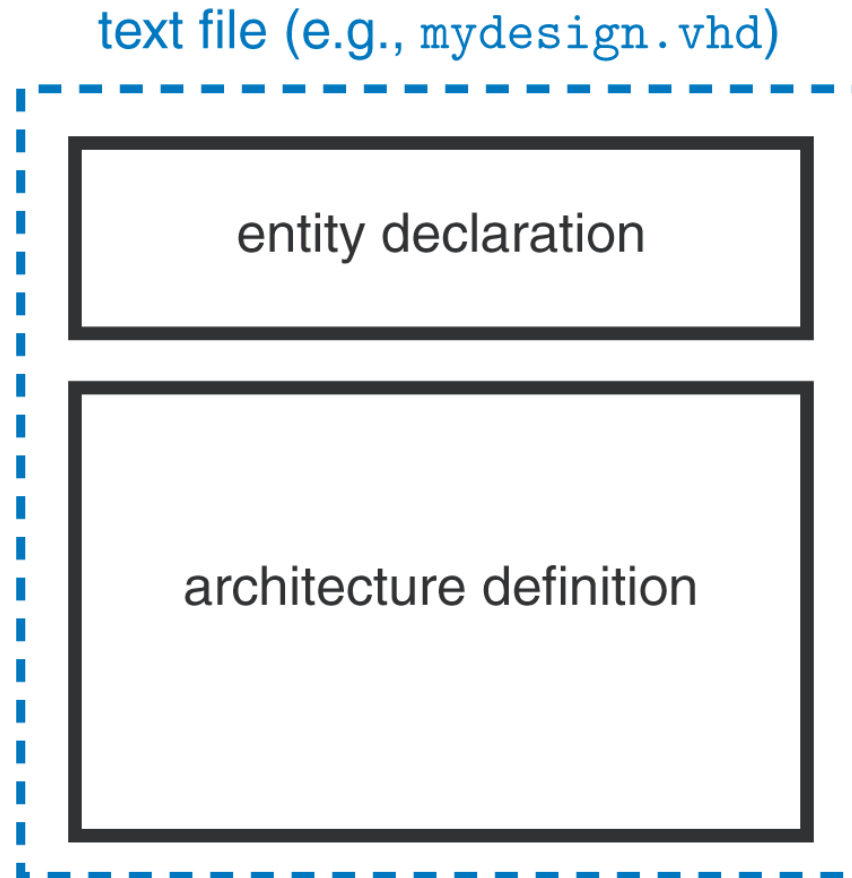


Figure 5-3  
VHDL program file structure.

# Tipos de objetos em VHDL

- Três tipos principais:

- **Sinal** (**signal**) : equivalem a nomes de fios em um circuito

- Mais comumente usados nos exemplos

- Ex.: 

```
signal sig1 : BIT; -- declaração
```

- Ex.: 

```
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.: 

```
variable var1 : BIT; -- declaração
```

- Ex.: 

```
var1 := '1'; -- atribuição de valor
```

- **Constante**: valores fixos

- Ex.: `'1'`

entity declaration

architecture definition

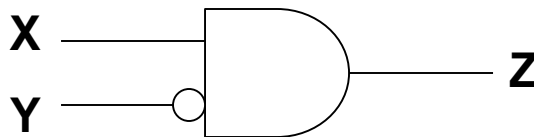
# Entidade: sintaxe

```

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

entity declaration

architecture definition

# 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;

```

entity declaration

architecture definition

# 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;
```

Parênteses para 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
```

opcionais  
("variáveis  
locais")

Deve ser o mesmo  
que na "entity"

```
begin
```

```
  concurrent-statement
  ...
  concurrent-statement
```

operação  
concorrente

```
end architecture-name;
```

## 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`

# Um 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;
```

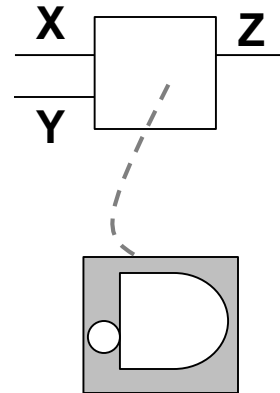


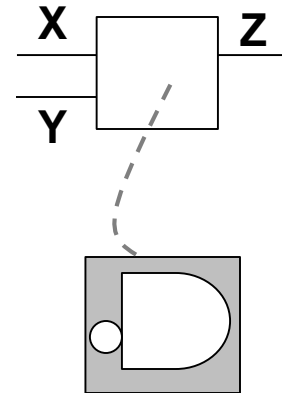
Table 5-13

VHDL program for an “inhibit” gate.

# Um 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
  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...

entity declaration

architecture definition

# Entidade: comparação com C

## VHDL

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

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

## Linguagem C

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



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

```
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)

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

## Linguagem C

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



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

```
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 (128 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.

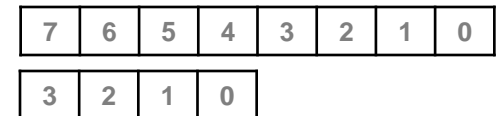
# Tipos de dados

- Exemplos de uso de `std_logic_vector`:

- Declaração:**

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

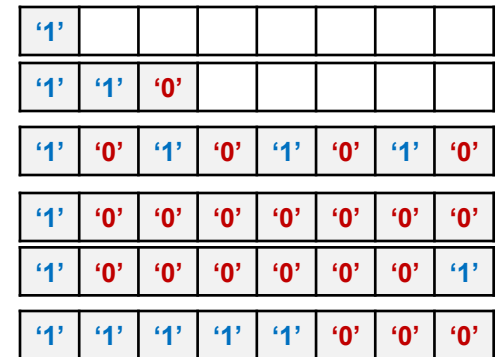
Também possível: `to`



- 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');
```

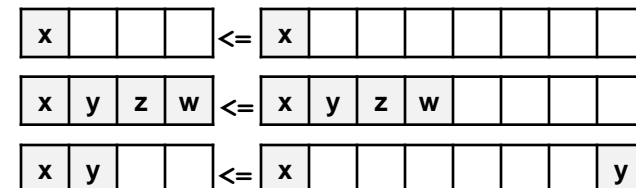
Separação entre vários itens



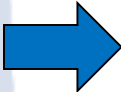
- Acesso:**

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

concatenação



# 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
5                               -- 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.:  $(A \cdot B) \rightarrow A \text{ and } B;$        $A' \rightarrow \text{not } A$
  - Numéricos: **+**, **-**, **\***, **/**, **abs** (*valor absoluto*), **\*\*** (*exponenciação*)
    - Ex.:  $|val| \rightarrow \text{abs } val;$        $X^Y \rightarrow X ** Y$
  - Comparação: **=**, **/=** (*diferente*), **<**, **<=**, **>**, **>=**
    - Ex.:  $A \neq B \rightarrow A \text{ /= } B$
  - Vetores: **&** (*concatenação*), **rol** (*rotação à esquerda*), **ror** (*rotação à direita*)
    - Ex.: A concatenado com B  $\rightarrow A \text{ \& } B$
    - Ex.: A rotacionado à direita de 2 posições  $\rightarrow A \text{ ror } 2$

# Exercício/Exemplo

- Escreva a entity para o circuito a seguir:



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

# Estilos de projeto em VHDL

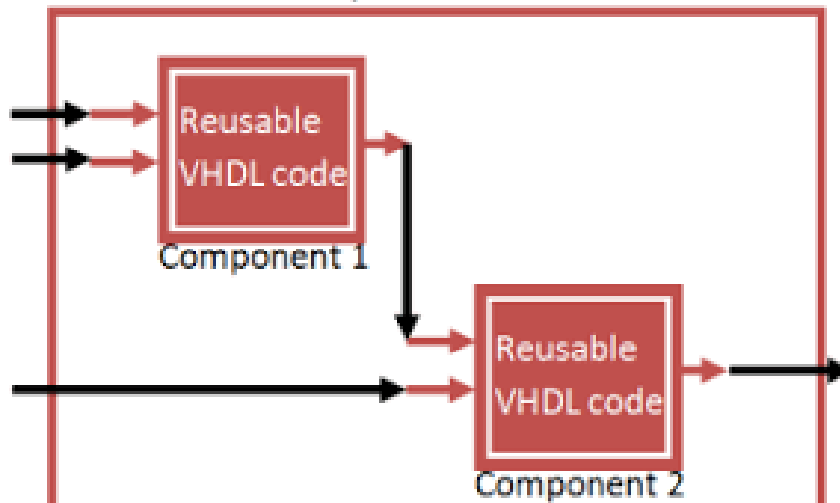
- Diferentes estilos para descrever arquitetura (parte “executável” do componente)
  - **Estrutural:** portas e suas conexões
    - Síntese mais controlada por projetista: útil para combinar componentes prontos (blocos com entradas e saídas)
  - **Fluxo de dados:** relação entre entradas e saídas
  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



# **VHDL: PROJETO USANDO ABORDAGEM ESTRUTURAL**

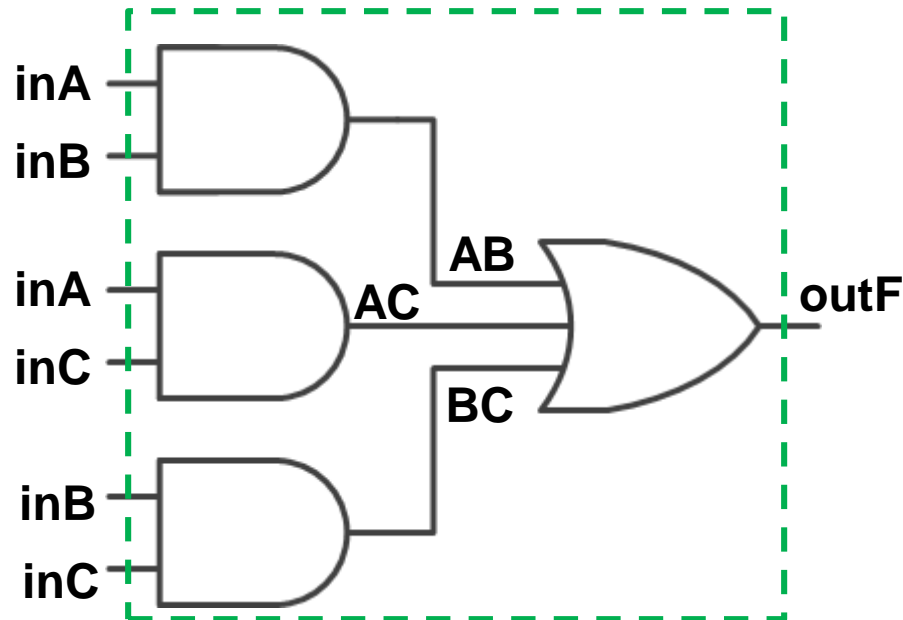
# VHDL Estrutural

- Basicamente conectar caixas pretas
- Elemento principal: **componente**
- Podemos instanciar o mesmo componente diversas vezes



# 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;
```

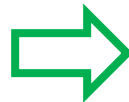
# VHDL Estrutural

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

Passo 2

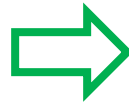
- Passo 2: declarar componentes usados pela entidade
  - Componentes internos : AND2 e OR3

```
-- 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;
```

# VHDL Estrutural

- Vamos mostrar um exemplo: “maioria entre A, B e C”
- Passo 2: declarar componentes usados pela entidade

Passo 2

```
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

- 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;

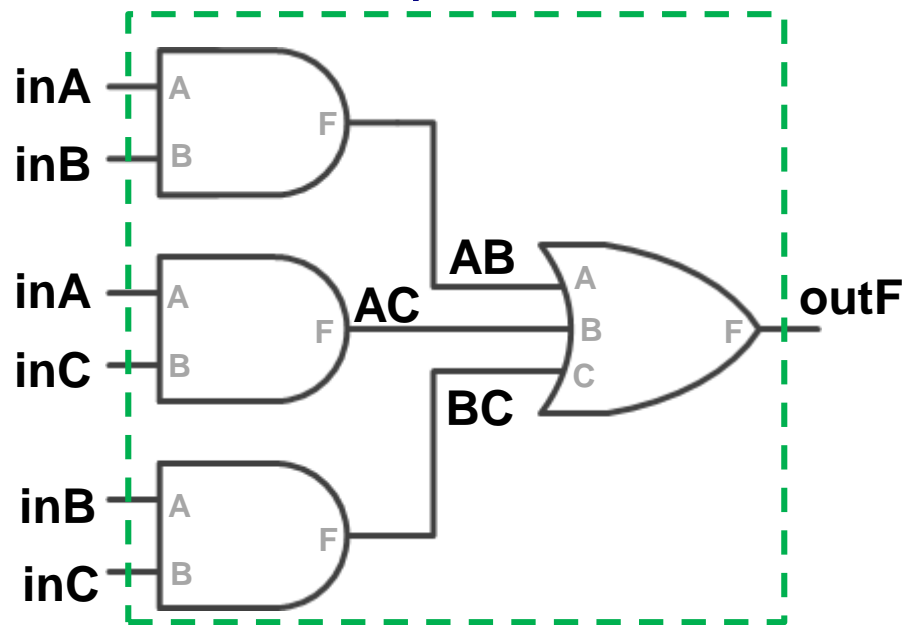
  component and2 is  -- componente: AND2
  port (A,B : in std_logic; F : out std_logic);
  end component;

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

begin                                                -- mapeamentos
  xAB: and2 port map (A => inA, B => inB, F => AB);
  xAC: and2 port map (A => inA, C => inC, F => AC);
  xBC: and2 port map (B => inB, C => 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”



```
begin
```

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

```
  xAC: and2 port map (A => inA, C => inC, F => AC);
```

```
  xBC: and2 port map (B => inB, C => inC, F => BC);
```

```
  xF: or3 port map (A => AB, B => AC, C => BC, F => outF);
```

```
end arch;
```

```
-- mapeamentos
```

# **VHDL FLUXO DE DADOS: 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:  $\leq$
- Ex.: Implemente a função  $F = L' \cdot M' \cdot N + L \cdot M$

```
-- 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;
```

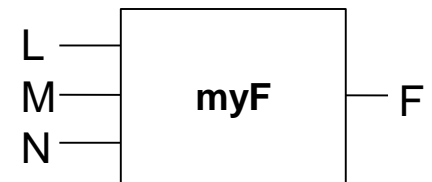


← Com 1 só linha

# Atribuição de Sinais

- Atribuição incondicional:  $\leq$
- Ex.: Implemente a função  $F = L' \cdot M' \cdot N + L \cdot M$

```
-- 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>;
```

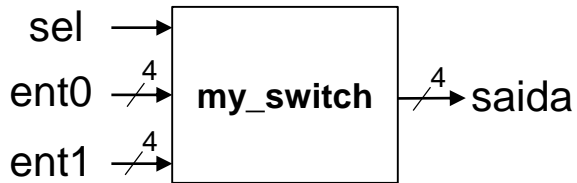
- Condição: variável booleana ou operação booleana (usando =, /=, >, >=, <, <=)
- Ex.: Implemente a arquitetura de  $F = L' \cdot M' \cdot N + L \cdot M$ , usando atribuição condicional

```
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  $F = L' \cdot M' \cdot N + L \cdot M$ , 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  $F = L' \cdot M' \cdot N + L \cdot M$ , usando atribuição condicional com seleção
  - Também podemos usar a soma canônica:  $F = \sum(1,6,7)$

```
architecture myFCanonicalSum 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 myFCanonicalSum;
```

# 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;
```

# **VHDL COMPORTAMENTAL: ATRIBUIÇÕES SEQUENCIAIS (PROCESSOS)**

# 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;
```

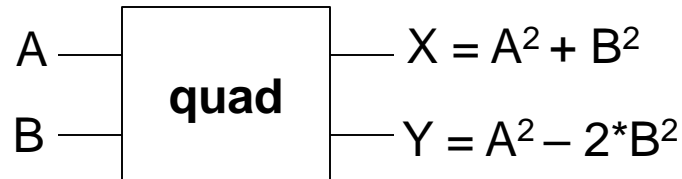
# Ex.: Fluxo de dados vs. Comportamental



```
-- 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;
        b2 := B**2;
        X  <= a2 + b2;
        b2 := 2*b2;
        Y  <= a2 - b2;
    end process squares;
end comp;
```

# Processos: lista de sensibilidade

- Lista de sensibilidade pode conter só 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:

$$F = A \cdot B' \cdot C' + B \cdot C$$

```
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: atribuição condicional

- Atribuição condicional: **if**

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

- Exemplo (uma alternativa):  $F = A \cdot B' \cdot C' + B \cdot C$

```
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 com seleção: **case**

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

- Exemplo:  $F = A \cdot B' \cdot C' + B \cdot C$

```
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:  $F = A \cdot B' \cdot C' + B \cdot C$

```
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;
```

Pode haver mais  
de um comando  
por when



# 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](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

# VHDL Estrutural (exemplo no Wakerly)

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

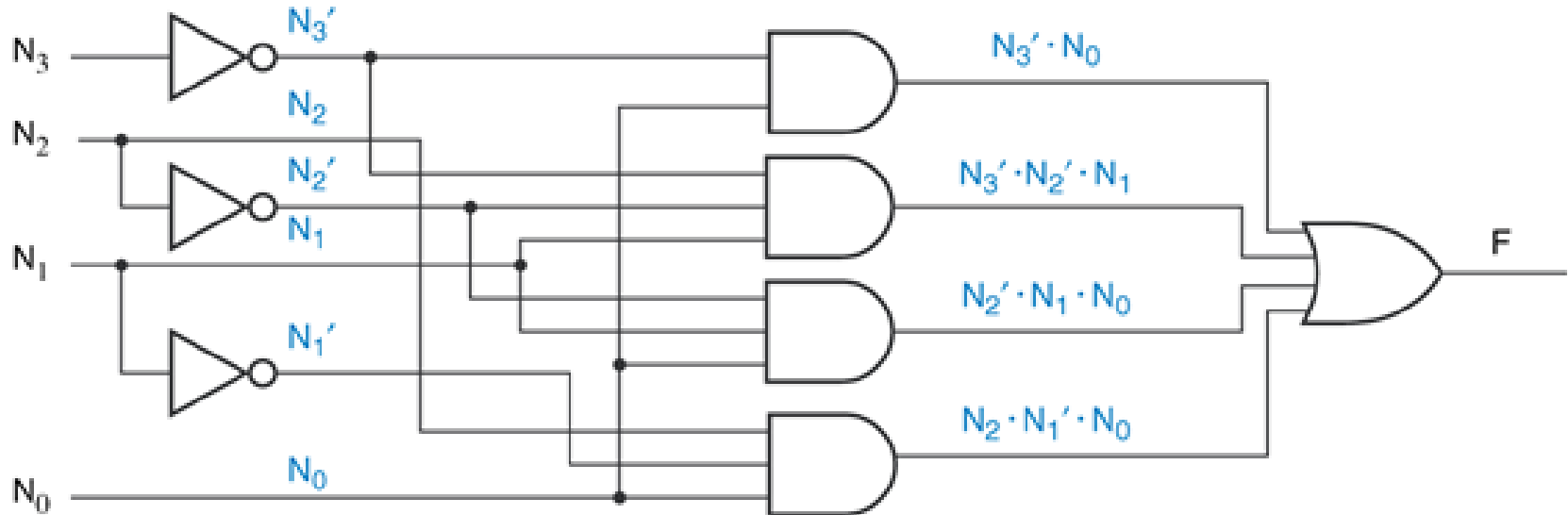
entity prime is
    port ( N: in STD_LOGIC_VECTOR (3 downto 0);
          F: out STD_LOGIC );
end prime;

architecture prime1_arch of prime is
    signal N3_L, N2_L, N1_L: STD_LOGIC;
    signal N3L_NO, N3L_N2L_N1, N2L_N1_NO, N2_N1L_NO: 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_NO);
    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_NO);
    U7: AND3 port map (N(2), N1_L, N(0), N2_N1L_NO);
    U8: OR4 port map (N3L_NO, N3L_N2L_N1, N2L_N1_NO, N2_N1L_NO, 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_NO);  
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_NO);  
U7: AND3 port map (N(2), N1_L, N(0), N2_N1L_NO);  
U8: OR4 port map (N3L_NO, N3L_N2L_N1, N2L_N1_NO, N2_N1L_NO, F);
```

end prime1\_arch;

**Ligação entre  
entradas/saídas e  
componentes e  
sinais**

# Altera Quartus – Editor VHDL

```
File Edit View Project Processing Tools Window Help Search altera.c...
267
268
1 library IEEE;
2 use IEEE.std_logic_1164.all;
3
4 entity prime is
5   port ( N : in std_logic_vector ( 3 downto 0 );
6         f : out std_logic );
7 end entity prime;
8
9 architecture prime_arch of prime is
10
11   signal N3_L, N2_L, N1_L : std_logic;
12   signal N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0 : std_logic;
13
14   component INV
15   port ( I : in std_logic;
16         O : out std_logic );
17 end component INV;
18
19   component GAND2
20   port ( I0, I1 : in std_logic;
21         O : out std_logic );
22 end component GAND2;
23
24   component GAND3
25   port ( I0, I1, I2 : in std_logic;
26         O : out std_logic );
27 end component GAND3;
28
29   component GOR4
30   port ( I0, I1, I2, I3 : in std_logic;
31         O : out std_logic );
32 end component GOR4;
33
34 begin
35   U1 : INV port map ( N(3), N3_L );
36   U2 : INV port map ( N(2), N2_L );
37   U3 : INV port map ( N(1), N1_L );
38   U4 : GAND2 port map ( N3_L, N(0), N3L_N0 );
39   U5 : GAND3 port map ( N3_L, N2_L, N(1), N3L_N2L_N1 );
40   U6 : GAND3 port map ( N2_L, N(1), N(0), N2L_N1_N0 );
41   U7 : GAND3 port map ( N(2), N1_L, N(0), N2_N1L_N0 );
42   U8 : GOR4 port map ( N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0, F );
43 end prime_arch;
44
```

# Quartus – Compilação (Síntese)

The screenshot displays the Quartus IDE interface during a compilation process. The main window shows the 'Flow Summary' for a Cyclone V device (5CSXFC6D6F31C8ES). The flow status is 'Successful - Tue May 9 20:29:50 2017'. The compilation was performed using Quartus Prime Version 16.1.0 Build 196 10/24/2016 SJ Lite Edition. The revision name is 'prime'. The device is Cyclone V, and the timing models are preliminary. The logic utilization is 1 / 41,910 (< 1 %). The total registers are 0, total pins are 5 / 499 (1 %), and total virtual pins are 0. The total block memory bits are 0 / 5,662,720 (0 %). The total DSP blocks are 0 / 112 (0 %). The total HSSI RX PCSs are 0 / 9 (0 %). The total HSSI PMA RX Deserializers are 0 / 9 (0 %). The total HSSI TX PCSs are 0 / 9 (0 %). The total HSSI PMA TX Serializers are 0 / 9 (0 %). The total PLLs are 0 / 15 (0 %). The total DLLs are 0 / 4 (0 %).

The 'Tasks' window shows the following tasks and their completion times:

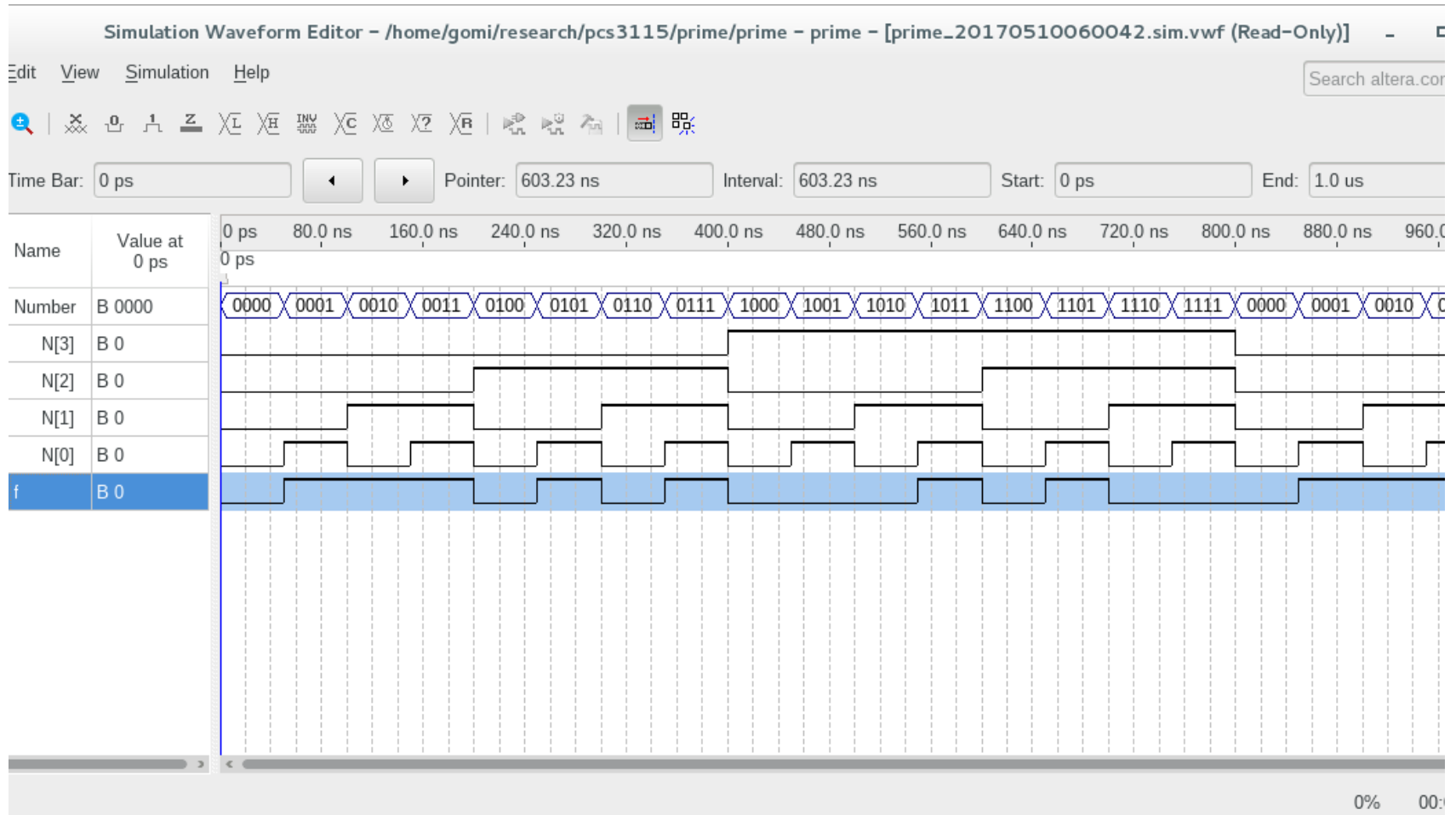
Task	Time
Compile Design	
Analysis & Synthesis	00:00:19
Analysis & Elaboration	
Partition Merge	
Netlist Viewers	
Design Assistant (Post-Mapping)	
I/O Assignment Analysis	
Fitter (Place & Route)	00:01:04
Assembler (Generate programming files)	00:00:09
TimeQuest Timing Analysis	00:00:07

The bottom status bar shows the command used for compilation: `quartus_eda --read_settings_files=off --write_settings_files=off prime -c prime`. The status bar also displays several warnings:

- 18236 Number of processors has not been specified which may cause overloading on shared machines. Set the global assignment NUM\_PARALLEL\_PROCESSORS in your QSF to an appropriate value for best performance.
- 18995 Generated the EDA functional simulation netlist because it is the only supported netlist type for this device.
- 204019 Generated file prime.vho in folder "/home/gpm1/vhdl/pcs3115/prime\_number/simulation/modelsim/" for EDA simulation tool
- 293000 Quartus Prime EDA Netlist Writer was successful. 0 errors, 2 warnings
- 293000 Quartus Prime Full Compilation was successful. 0 errors, 19 warnings



# Quartus – Simulação (ModelSim)



# Quartus – Gravação na FPGA

File Edit View Processing Tools Window Help Search altera.com

Hardware Setup... DE-SoC [3-4.1] Mode: JTAG Progress: **100% (Successful)**

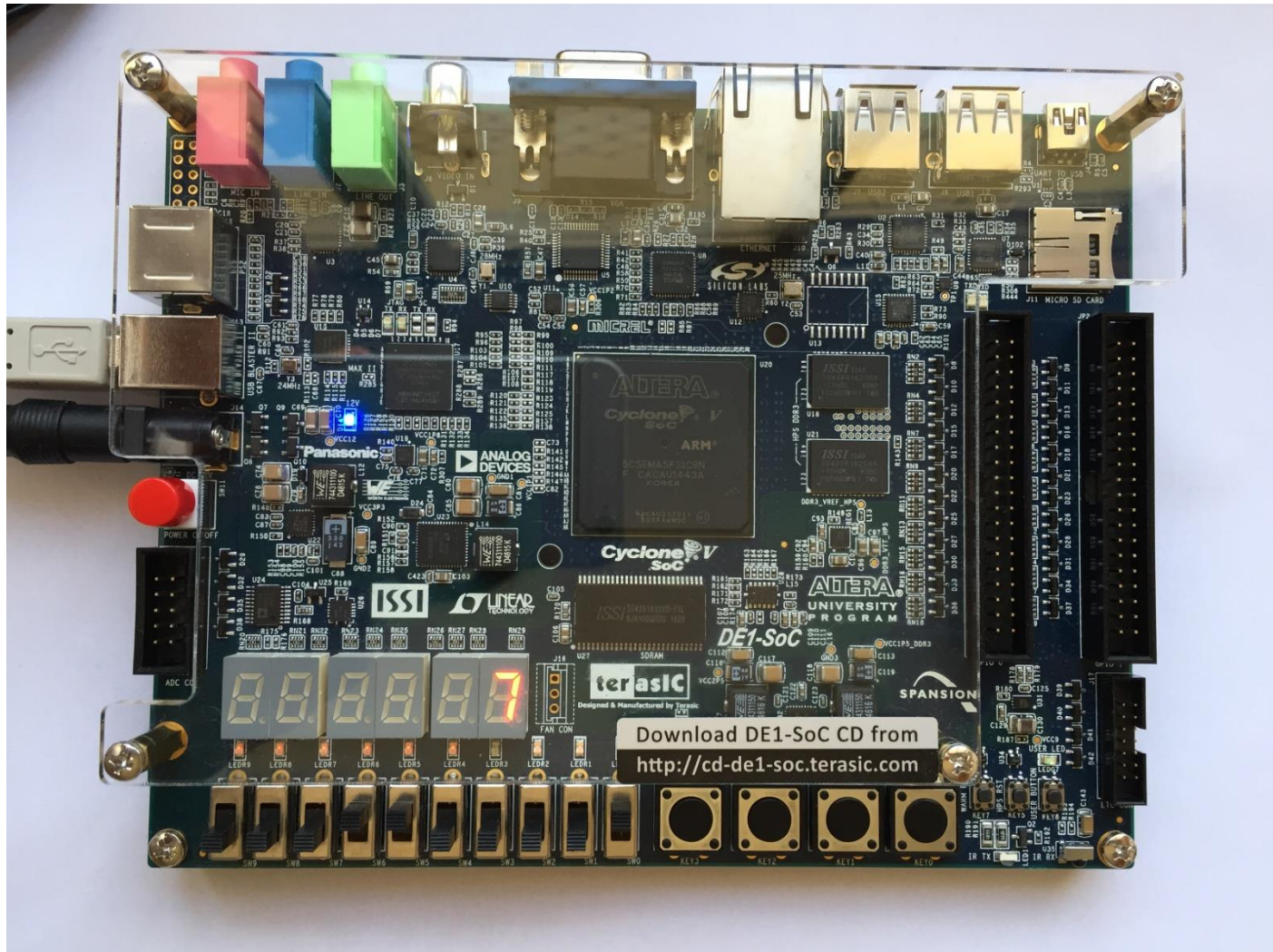
Enable real-time ISP to allow background programming when available

File	Device	Checksum	Usercode	Program/Configure	Verify	Blank-Check	Examine	Security Bit	Erase	ISP CLAMP
<none>	SOCVHPS	00000000	<none>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
output_files/prime.sof	5CSEMA5F31	00AF53D9	00AF53D9	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

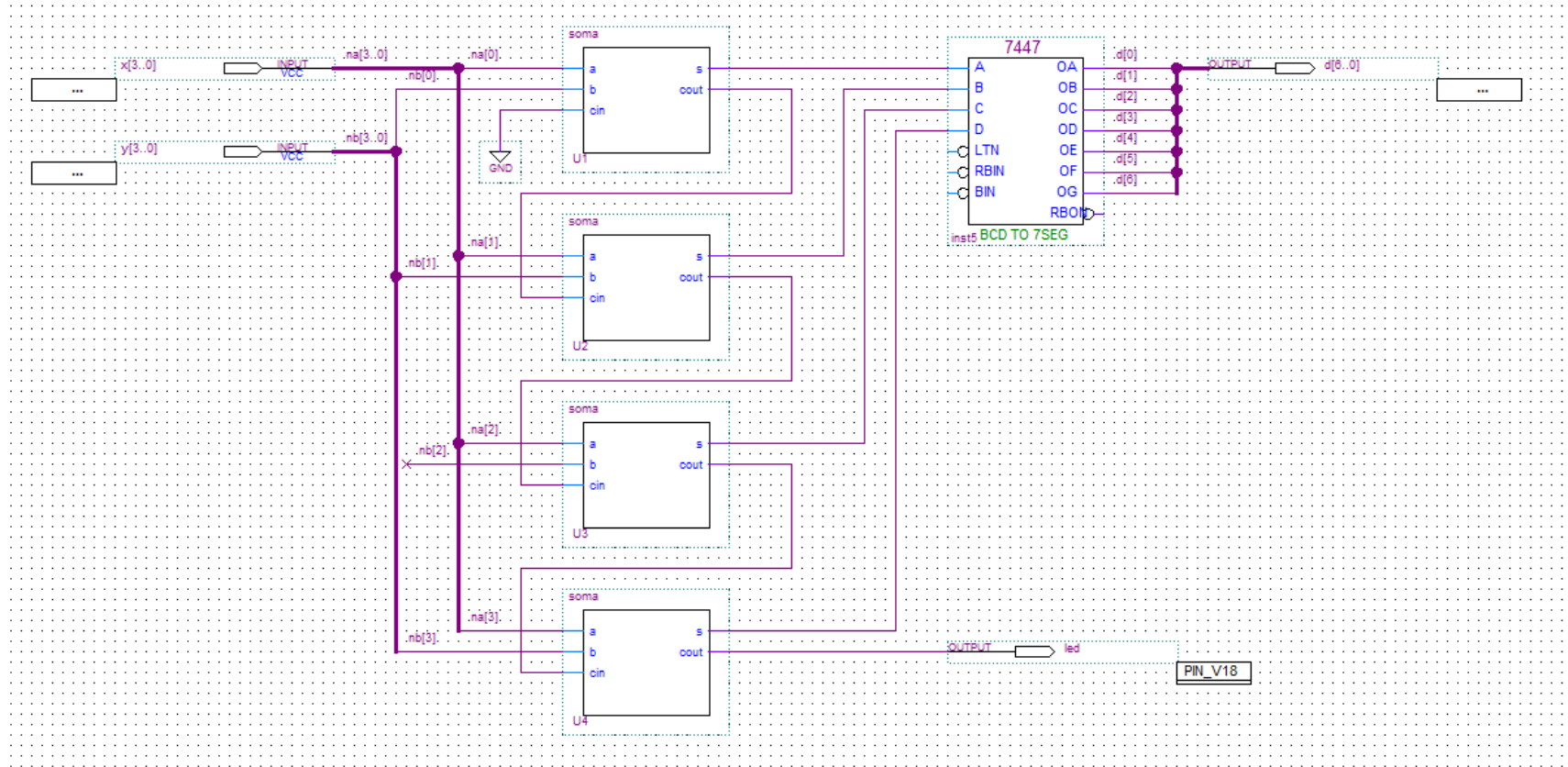
Start Stop Auto Detect Delete Add File... Change File... Save File Add Device... Up Down

```
graph LR; TDI --> SOCVHPS; SOCVHPS --> 5CSEMA5F31; 5CSEMA5F31 --> TDO;
```

# Kit FPGA Altera



# Somador Completo de 4 bits



# Somador Completo de 1 bit

The screenshot displays the Quartus II 64-bit IDE interface. The main window shows the VHDL code for a 1-bit full adder. The code is as follows:

```
1  LIBRARY ieee;  
2  USE ieee.std_logic_1164.ALL;  
3  
4  entity soma is  
5  port ( a:      in    STD_LOGIC;  
6         b:      in    STD_LOGIC;  
7         cin:    in    STD_LOGIC;  
8         s:      out   STD_LOGIC;  
9         cout:   out   STD_LOGIC);  
10 end soma;  
11  
12 architecture soma_arch of soma is  
13 begin  
14     s <= a xor b xor cin;  
15     cout <= (a and b) or (a and cin) or (b and cin);  
16 end soma_arch;  
17  
18  
19
```

The Project Navigator on the left shows the project hierarchy, including the entity 'soma' and its instances 'soma:U1' through 'soma:U4'. The Messages window at the bottom shows a message: '209061 Ended Programmer operation at Mon May 25 08:43:03 2015'. The status bar at the bottom indicates 'System (10) Processing' and '0% 00:00:00'.

# Gravação na FPGA

Programmer - Z:/gomi On My Mac/Dropbox/Pesquisa/Hardware/Altera/somador/somador - somador - [output\_files/somado...

File Edit View Processing Tools Window Help

Hardware Setup... DE-SoC [USB-1] Mode: JTAG Progress: 100% (Successful)

Enable real-time ISP to allow background programming (for MAX II and MAX V devices)

File	Device	Checksum	Usercode	Program/Configure	Verify	Blank-Check	Examine
<none>	SOCVHPS	00000000	<none>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
output_files/somador.sof	5CSEMA5F31	00AFB121	00AFB121	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Start Stop Auto Detect Delete Add File... Change File... Save File Add Device... Up Down

The diagram illustrates the JTAG programming process between two Altera devices. On the left, a device labeled 'SOCVHPS' is shown with an arrow labeled 'TDI' (Test Data In) pointing to it. An arrow labeled 'TDO' (Test Data Out) points from the 'SOCVHPS' device to the right. On the right, a device labeled '5CSEMA5F31' is shown with an arrow pointing to it from the 'SOCVHPS' device. A return arrow labeled 'TDO' points from the '5CSEMA5F31' device back to the 'SOCVHPS' device, completing the JTAG loop.

# 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

---

```
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;
```

---

“variáveis locais”



operações



# 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