UFRGS

Circuitos Programáveis

PLD

FPGA

Aula 17

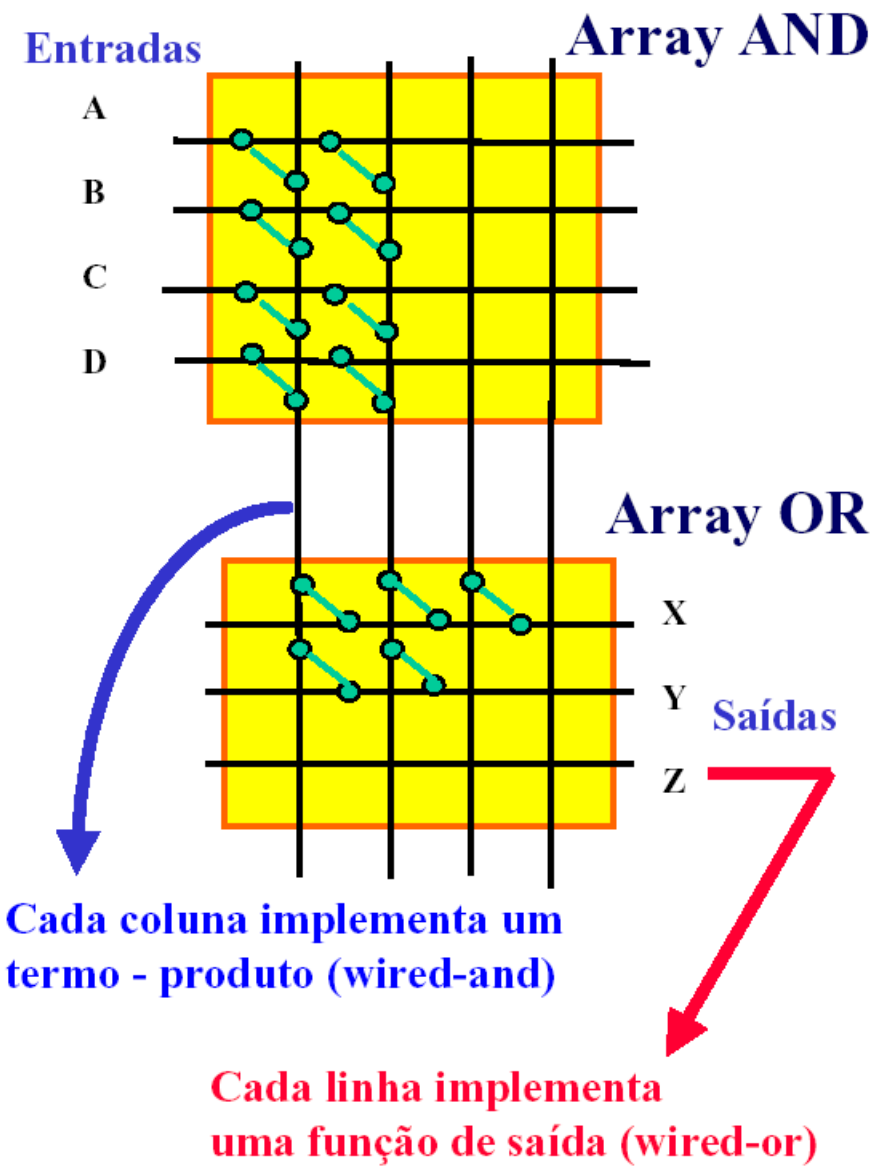
Introdução

Especificação de uma função como SDP (Soma de Produtos)

Implementação trivial por um circuito AND-OR

PLD's

- dispositivos pré-difundidos
- arrays AND-OR
- "fusíveis " inicialmente intactos
  (fusíveis são na verdade implementados por transistores/diodos)
- programação corresponde à queima dos "fusíveis"



Tipos de de Programmable Logic Devices (PLDs)

- **PROM (Programmable Read-only memory)**
- **PLA (Programmable Logic Array)**
- **FPGA**

Diferem Diferem:

- Na organização dos arrays AND e OR

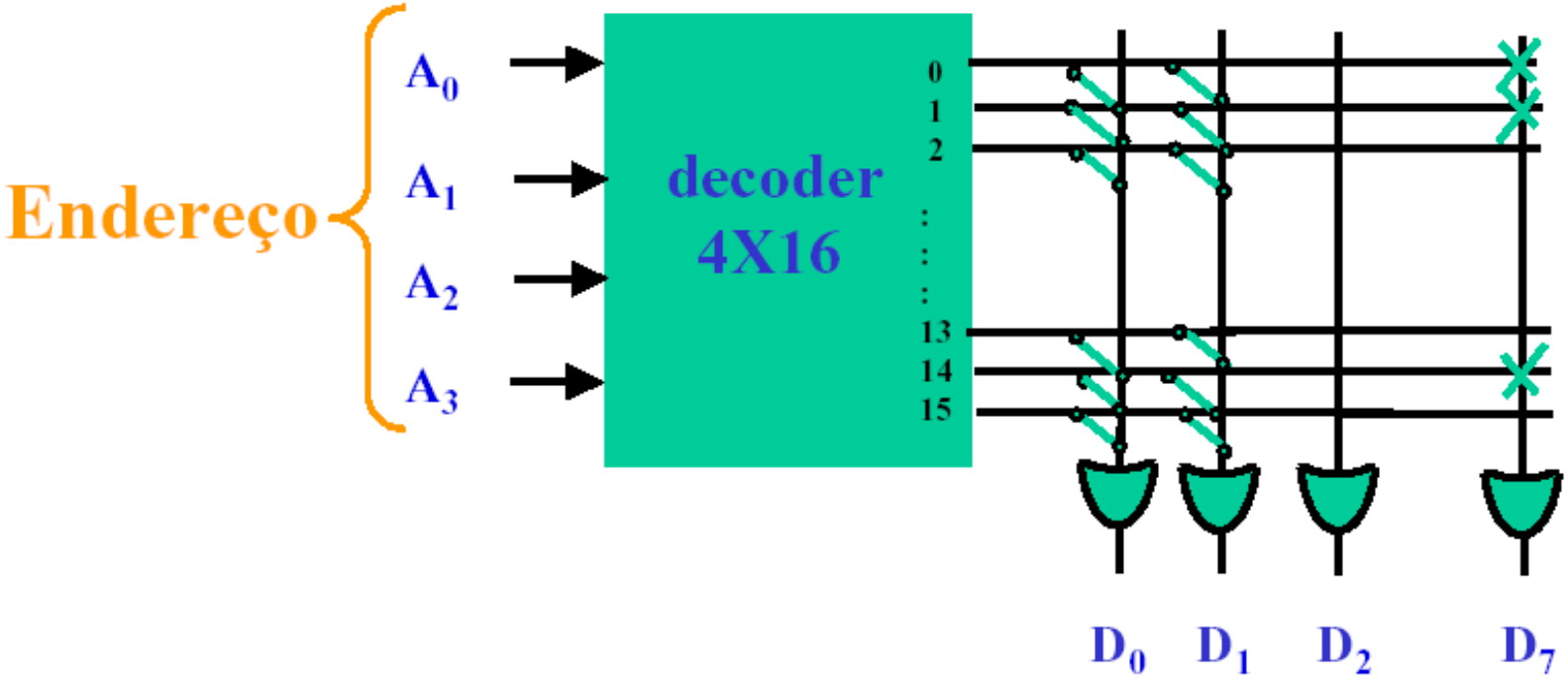
- Na programabilidade dos arrays (colocação dos fusíveis ou transistores)

- Programáveis pelo fabricante ou pelo usuário

Memória PROM usada como PLD

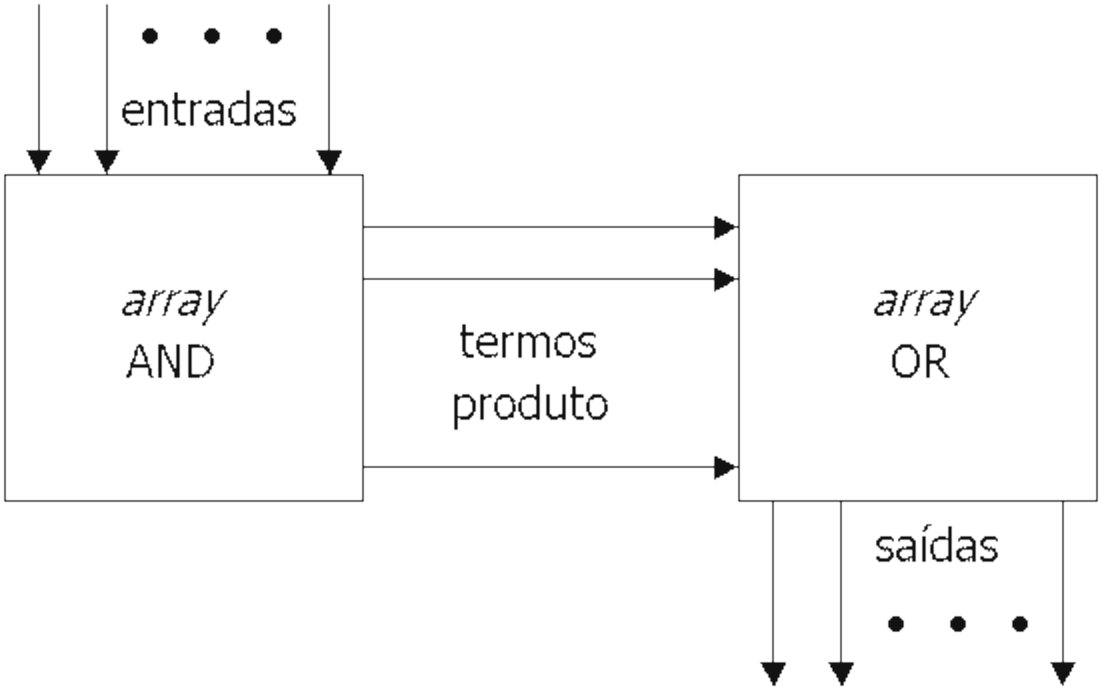
PROM - Programmable Read-Only-Memory

Representação simbólica

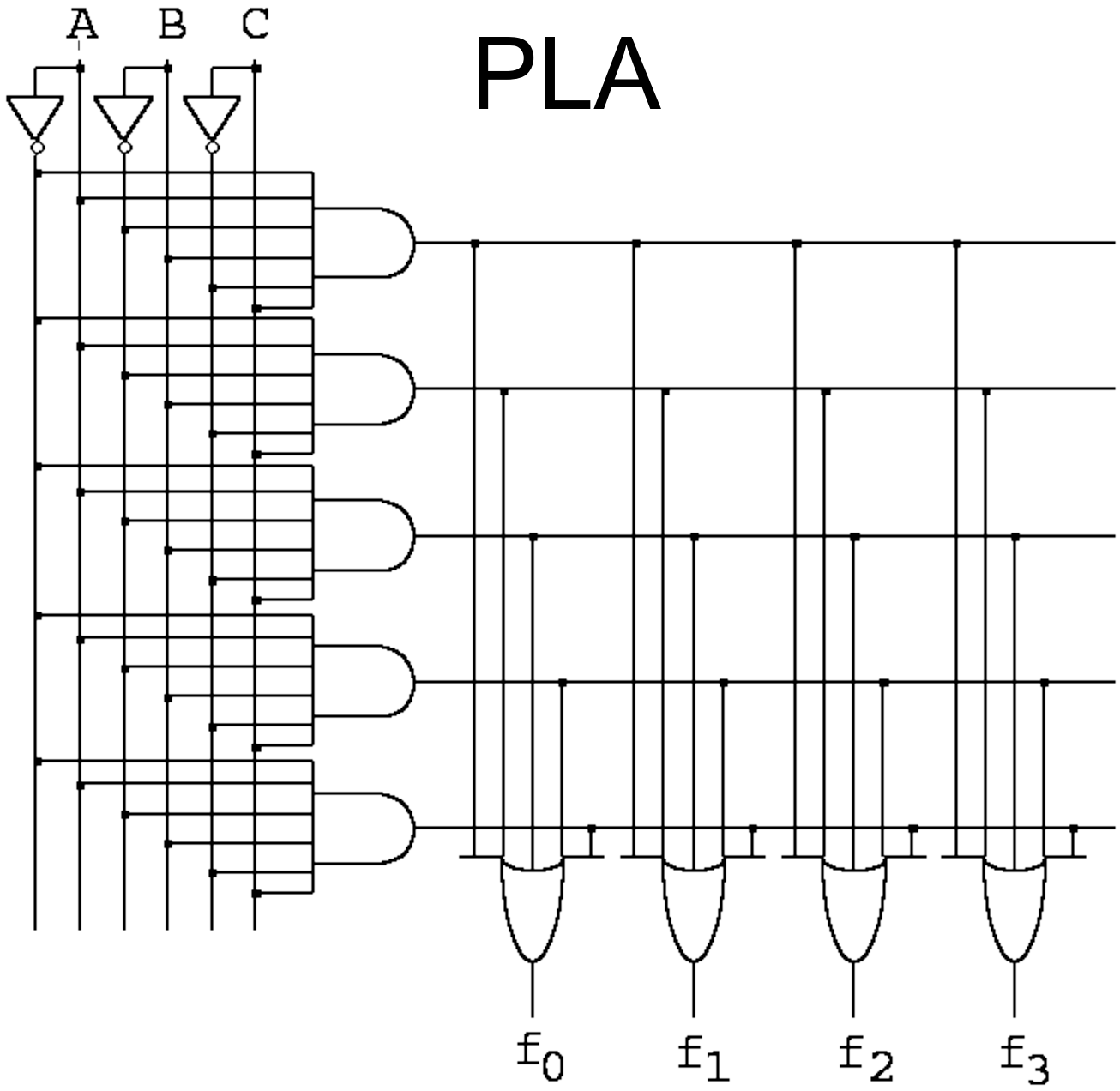


- Notar que se trata de uma memória, onde cada célula tem originalmente todos os bits iguais a 1.
- Programação corresponde à queima dos "fusíveis"
- Encarando a PROM como um PLD
  - Bits de endereço Ai ⇨ variáveis de entrada da função
  - Decodificador: efetua a decodificação de todos os mintermos
    cada palavra corresponde então a um mintermo
  - Bits de dados de saída ⇨ saída de múltiplas funções
  - Por exemplo: $F = \Sigma\, m\,(0,1,14) = m_0 + m_1 + m_{14}$
    Implementada por D_7 na figura
    Precisa memória com 16 mintermos ⇨ 4 variáveis de entrada
    4 bits de endereço
- Comparando com a forma geral de um PLD
  - PROM - decodifica todos os mintermos
  - ARRAY AND é fixo (decodificador) - saída dele são mintermos
  - ARRAY OR é programável - soma dos mintermos
  - Número de funções = número de bits das palavras
  - Número de variáveis de entrada = número de bits de endereço

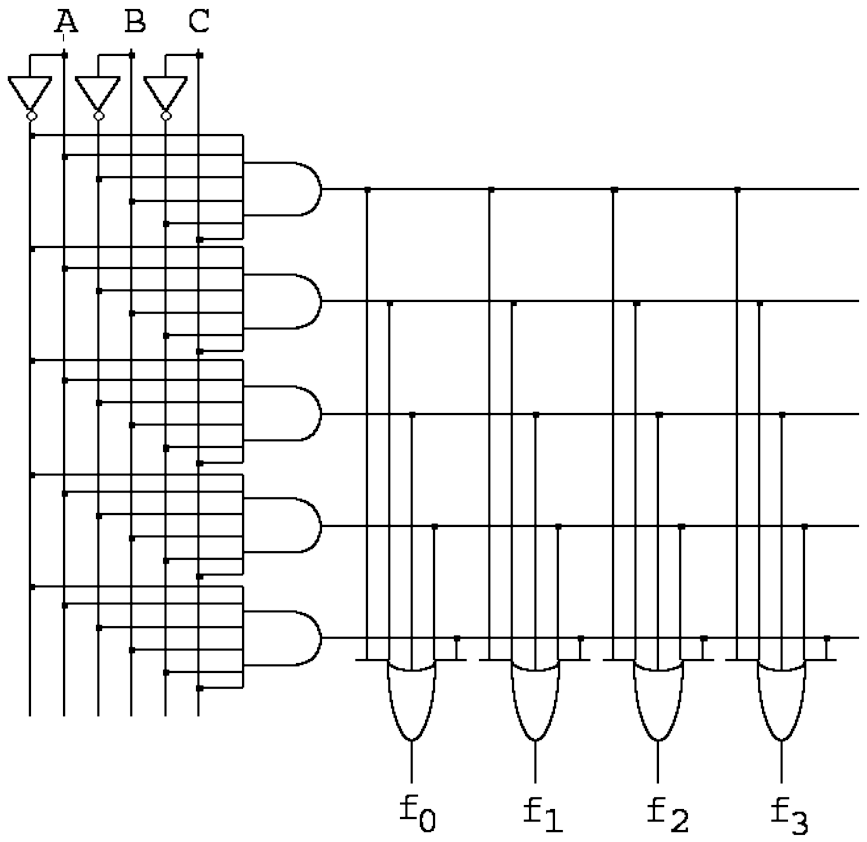
PLA



PLA

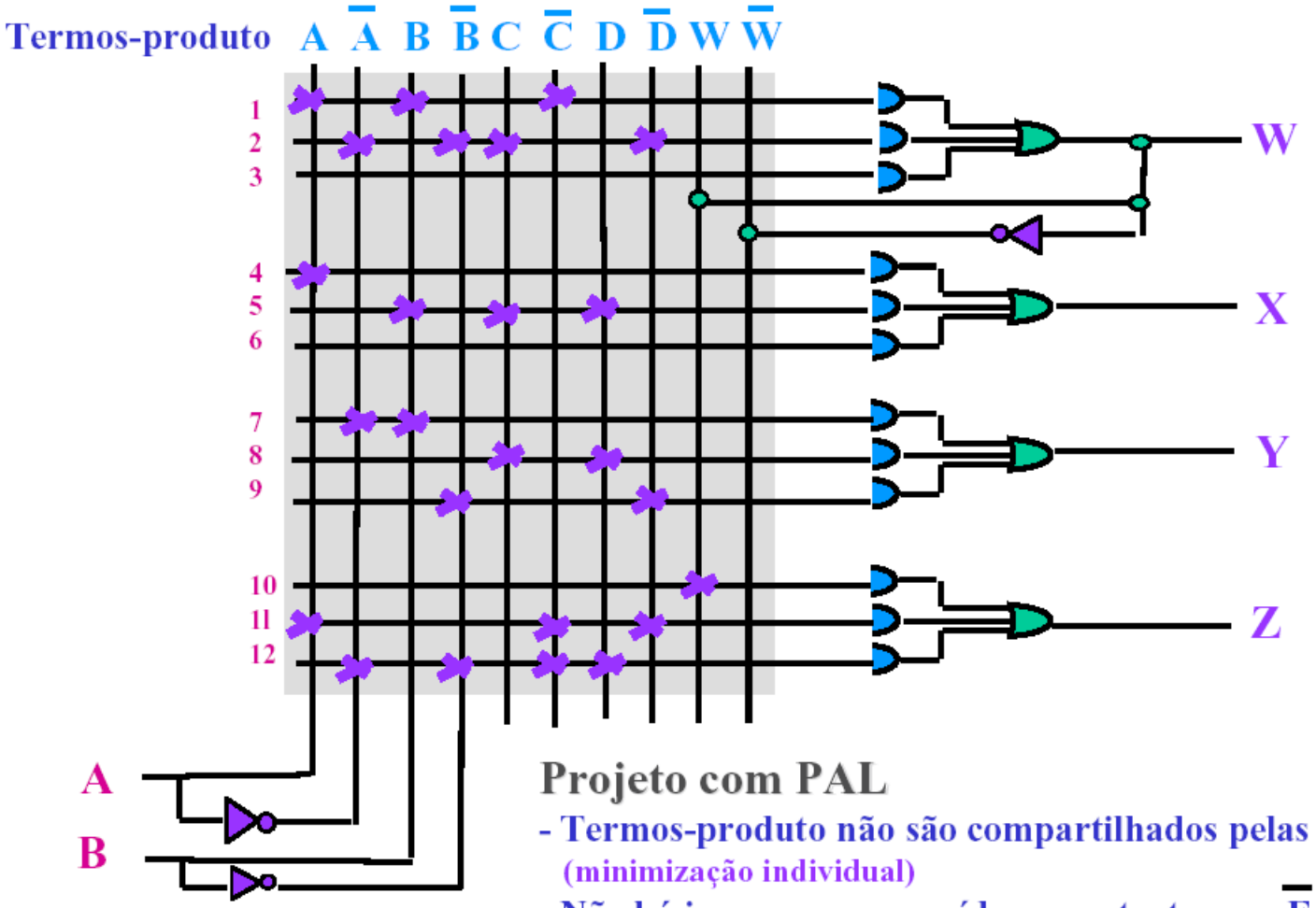


- F0 = A + B' C'
- F1 = A C' + A B
- F2 = B' C' + A B
- F3 = B' C + A



PAL - Programmable Array Logic

Características:
- Array AND é programável
- Array OR é fixo (cada OR é limitado a n termos-produto)
- Feedbacks das saídas para o array and permitem funções com mais de n termos



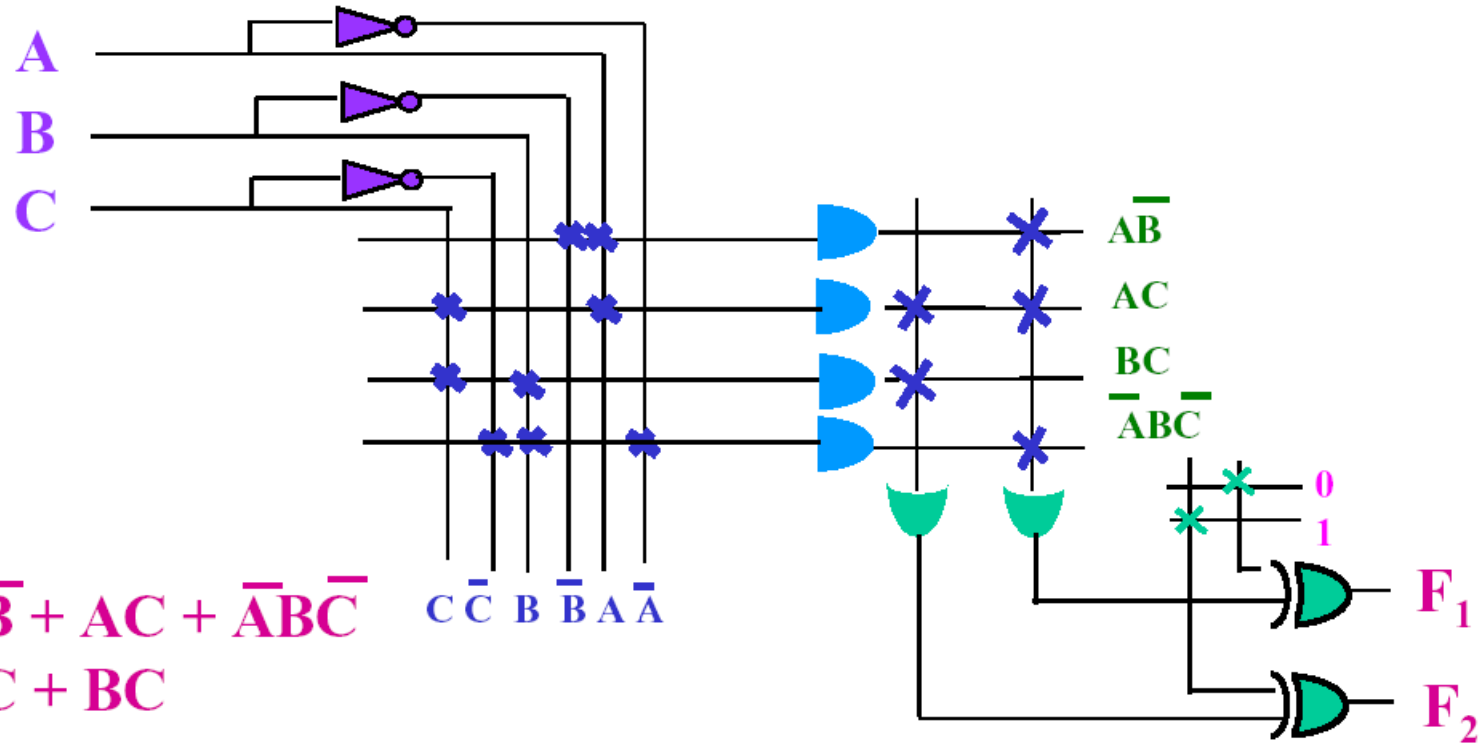
Projeto com PAL

- Termos-produto não são compartilhados pelas funções de saída (minimização individual)
- Não há inversores nas saídas para tentar um $\overline{F}$

PLA - Programmable Logic Array

Características

- Arrays AND e OR são programáveis
- Para n variáveis de entrada, o array AND tem 2n linhas (variável e variável complementada)
- São decodificados apenas os termos-produto necessários



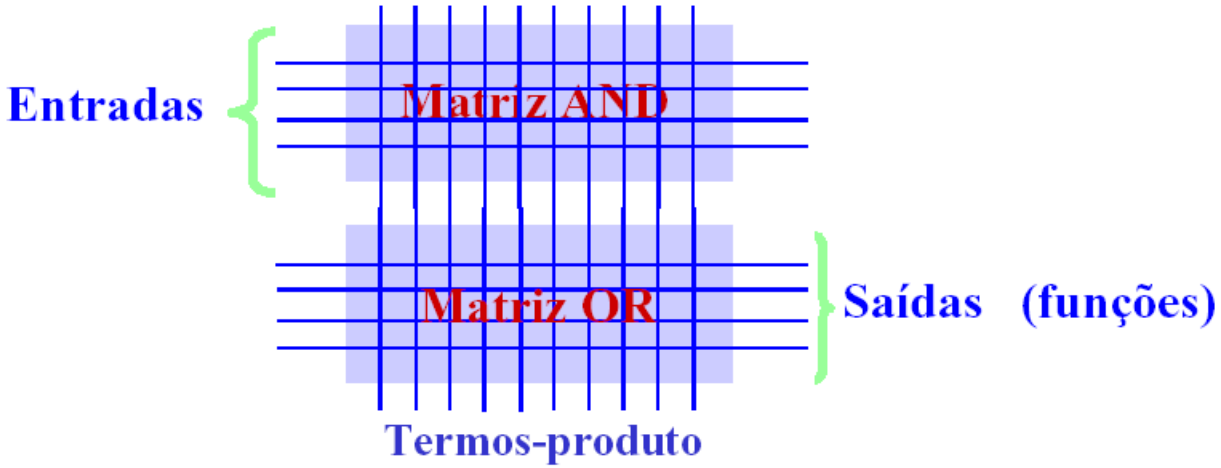
$F_1 = A\overline{B} + AC + \overline{A}\overline{B}\overline{C}$

$\overline{F}_2 = AC + BC$

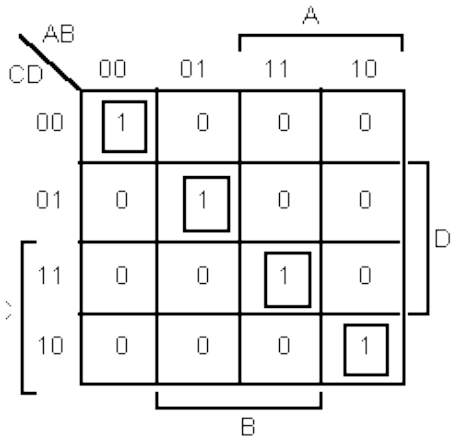
Projeto com PLA

- Deve-se minimizar todas as funções de saída simultaneamente, procurando-se o menor número de termos-produto comuns a todas as funções
- Deve-se tentar solução ótima tanto com F como com $\overline{F}$, procurando:
  - menor número de termos
  - termos comuns a outras funções

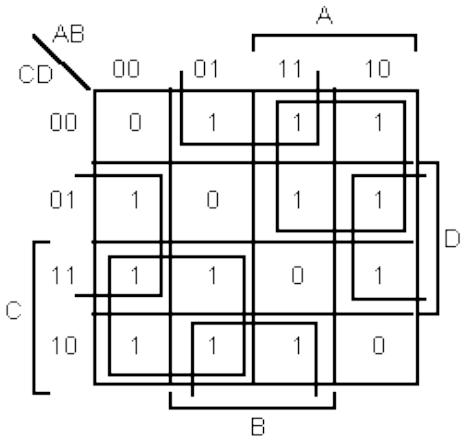
  No exemplo: AC é comum a F1 e $\overline{F2}$
- Tamanho do PLA
  - Nro. entradas = número de variáveis da função
  - Nro. linhas do array AND = nro. máximo de termos-produto no conjunto das funções
  - Nro. saídas = número de funções simultâneas



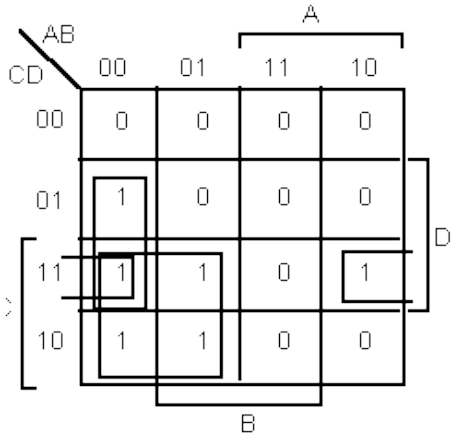
- Tipos de PLA's:
  - Programável por "máscaras", na fábrica (silicon foundry)
  - FPLA - field programmable (programável pelo usuário)



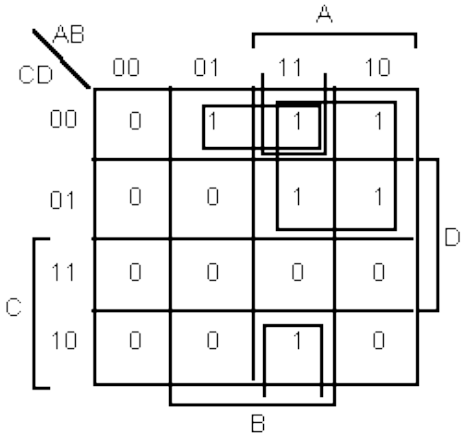
K-map for EQ



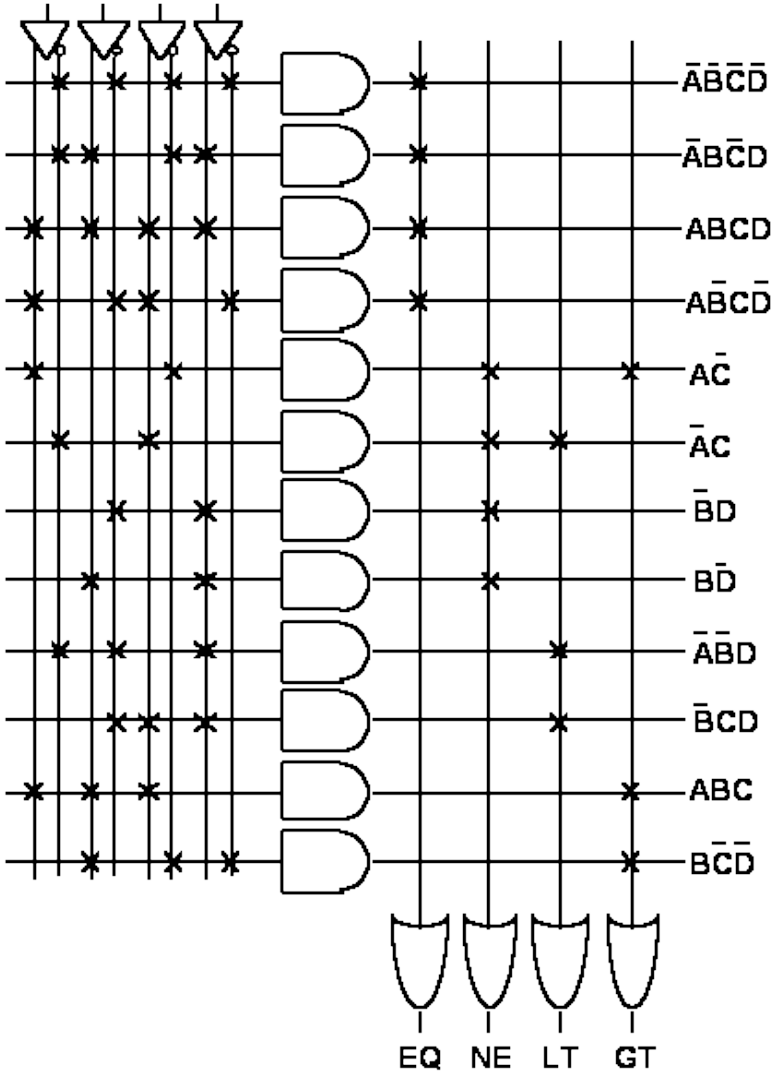
K-map for NE



K-map for LT



K-map for GT



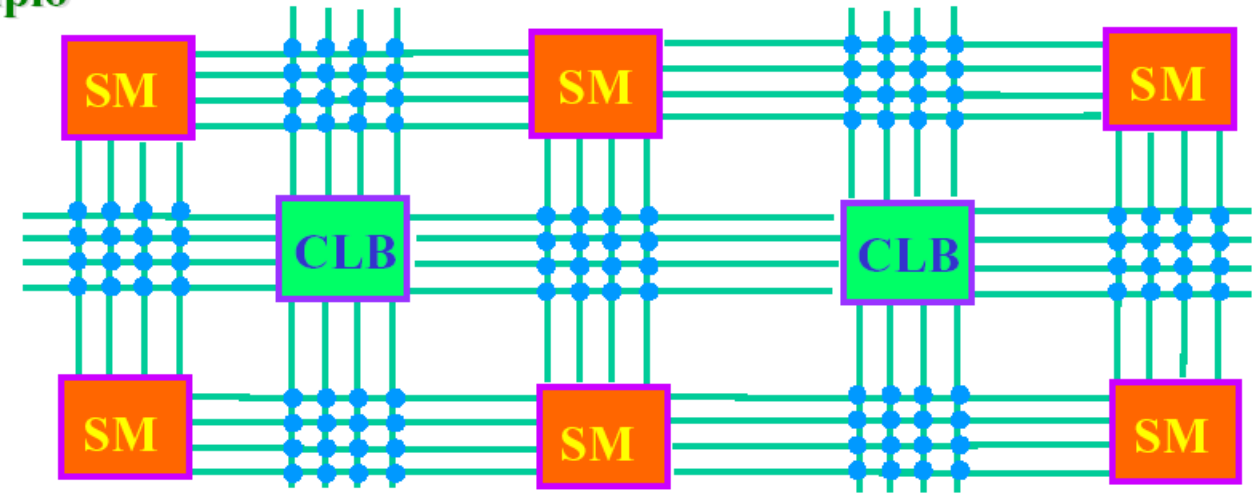
Field Programmable Gate Array

- FPGA
- Programável por
  - SRAM cells
  - Fúsivel
  - EEPROM cell

FPGA - Field Programmable Gate Array

Características
- Reprogramável n vezes
- Array de blocos lógicos (função definida por seleção)
- Blocos lógicos e conexões são programáveis

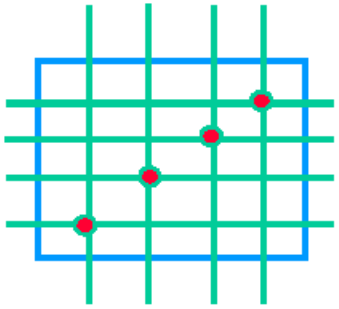
Exemplo



PLB — Blocos Lógicos Programáveis (Programmable Logic Blocks)
CLB — Configurable Logic Blocks

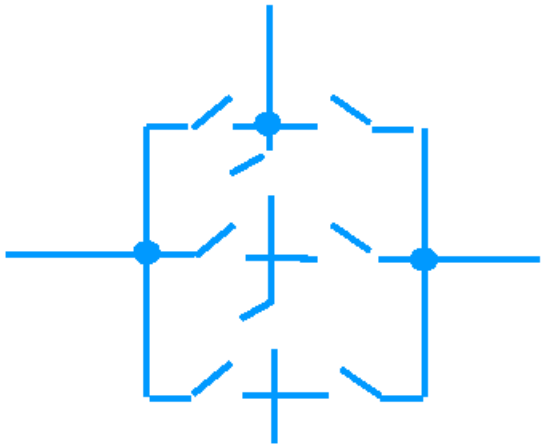
CLB

SM



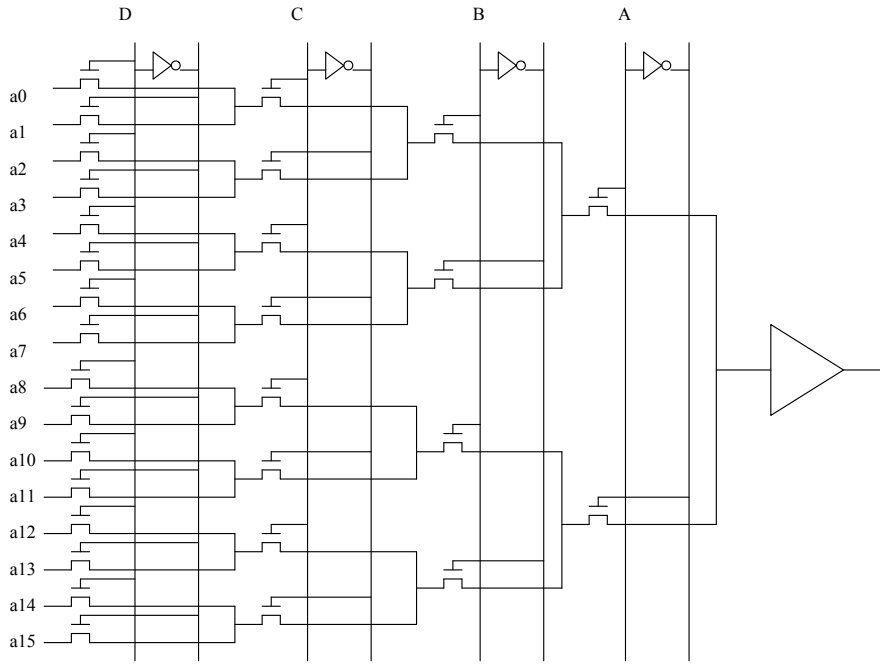
SM - Switching Matrix

Interconnection Point



Exercicio: A figura a seguir mostra o esquemático de um elemento lógico presente na maioria dos CLBs de arquiteturas programáveis (FPGAs). Esse elemento chamado de Lookup Table (LUT) é responsável pela implementação da lógica combinacional nos CLBs. Implemente a função H nesta LUT indicando os valores nas entradas A, B, C e D e a0-15.

$$H = \overline{X}.Y.Z.W + X.\overline{Y} + \overline{W}$$



Xilinx : Technology Solutions : Programmable Logic

- Virtex-4 FPGAs
- Virtex-II Pro Platform FPGAs
- Virtex-II Platform FPGAs
- Virtex Series
- Spartan-3E FPGAs
- Spartan-3 FPGAs
- Spartan-IIE FPGAs
- Spartan-II FPGAs
- Spartan-XL FPGAs
- Spartan FPGAs
- CoolRunner-II CPLDs
- CoolRunner XPLA3 CPLDs
- XC9500 Series CPLDs
- CPLD Tutorial
- Configuration Storage Devices
- Aerospace and Defense
- RocketPHY
- XA Automotive

Programmable Logic

The Programmable Logic area provides a complete list of all Xilinx devices and easy access to all the information you need to design and develop with Xilinx silicon devices

FPGA

- EasyPath FPGAs

Virtex™ Series

- Virtex-4 FPGAs
- Virtex-II Pro / ProX Platform FPGAs
- Virtex-II Platform FPGAs
- Virtex / E / EM FPGAs

Spartan™ Series

- Spartan-3E FPGAs
- Spartan-3 FPGAs
- Spartan-IIE FPGAs
- Spartan-II FPGAs
- Spartan-XL FPGAs
- Spartan FPGAs

CPLD — Learn More

- CoolRunner™-II CPLDs
- XC9500 Series
- CoolRunner XPLA3 CPLDs
- CPLD Tutorial

Configuration Storage Devices — Learn More

Xilinx offers a range of configuration storage devices to configure all Xilinx FPGAs.

- Platform FLASH
- System ACE
- Legacy PROM
- Cable Support
- Programmer Solutions

Aerospace and Defense — Learn More

Choose Your Device
FPGA vs ASIC Calculator
Device Family Overview
Performance Advantages

FPGAs
- Stratix II
- Stratix
- Cyclone II
- Cyclone
- Stratix II GX
- Stratix GX
- APEX II
- APEX 20K
- Mercury
- FLEX 10K
- ACEX 1K
- FLEX 6000

CPLDs
- MAX II
- MAX 3000A
- MAX 7000

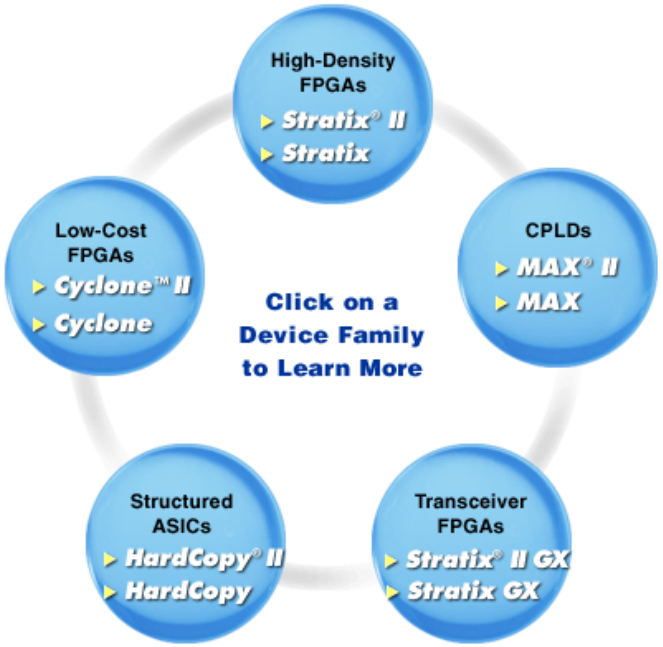
Structured ASICs
- About HardCopy
- HardCopy II
- HardCopy Stratix
- HardCopy APEX 20K

Configuration Devices
- Enhanced Configuration
- Serial Configuration

Embedded Processors
About Excalibur
- Excalibur Devices

Market-Specific Offerings

Devices



Device Family Overview

Discussão crítica do projeto de blocos combinacionais

- Alternativas de projeto de circuito digitais
  - Componentes discretos (de prateleira)
  - Circuito integrado dedicado

- Estilos de projeto de blocos combinacionais
  - Lógica programável
  - Lógica hardwired
    - Decodificadores/multiplexadores
    - Lógica em 2 níveis
    - Lógica aleatória multi-nível

- Fatores a considerar na escolha
  - Eficiência da solução: área, timing, consumo
  - Custo de produção
  - Custo de projeto
  - Volume de produção
  - Número de entradas/saídas (encapsulamento)

Somente em função do problema a ser resolvido é que podemos definir a melhor solução.

VHDL

- **VHDL = VHSIC Hardware Description Language**
- **VHSIC = Very High Speed Integrated Circuit**

Descrição de hardware usada para modelagem, simulação e sintese de sistemas digitais.

Conceito de entidade

Primeiro passo escrever a declaração de entidade que identifica o módulo como um bloco lógico distinto, e então, definir portas de entradas e saídas.

```
entity porta_simples_nor3 is
    Port   (a, b, c       : in bit;
            s             : ou bit);
end porta_simples_nor3;
```

Conceito de Arquitetura

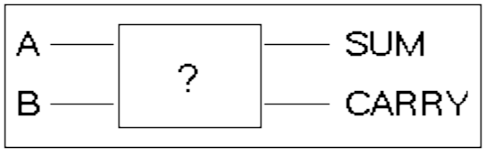
Uma vez definida a entidade, faz-se necessário definir a arquitetura, ou seja, o comportamento do bloco lógico.

```
architecture exemplo1 of porta_simples_nor3 is
begin
        s <= not(a or b or c);
end exemplo1;
```

```
architecture exemplo2 of porta_simples_nor3 is
signal z : bit;
begin
        z <= a or b or c;
    s <= not z;
end exemplo2;
```

Entity

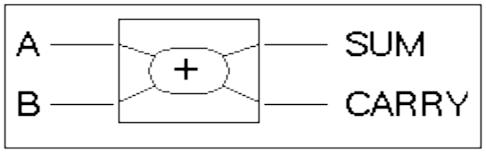
```
entity HALFADDER is
   port(
      A, B:                in   bit;
      SUM, CARRY: out bit);
end HALFADDER;
-- VHDL'93: end entity HALFADDER
```




```
entity ADDER is
    port(
      A, B:              in     integer range 0 to 3;
      SUM:               out integer range 0 to 3;
      CARRY:             out bit );
end ADDER;
```

Architecture

```
entity HALFADDER is
   port(
      A, B:              in   bit;
      SUM, CARRY: out bit);
end HALFADDER;

architecture RTL of HALFADDER is

begin

   SUM      <= A xor B;
   CARRY <= A and B;

end RTL;
end architecture RTL ;
```



Conceito de funções concorrentes

As declarações e comandos no VHDL são executados de maneira concorrente. Porem, é possível inserir atraso.

```
architecture exemplo3 of porta_simples_nor3 is
signal z : bit;
begin
      z <= a or b or c after 4 ns;
   s <= not z after 2 ns;
end exemplo3;
```

Uso de Componentes

```
entity funcao1 is
      Port   (x1, x2, x3, x4              : in bit;
                  y                      : ou bit);
end funcao1;
architecture exemplo1 of funcao1 is
component porta_simples_nor3
port( a,b,c : in bit;
      s     : out bit);
end component;
component porta_simples_nand2
port( a,b   : in bit;
      s     : out bit);
end component;
signal z1, z2 : bit;
BEGIN
G1: porta_simples_nor3 port map(x1, x2, x3, z1);
G2: porta_simples_nand2 port map(z1, x4, z2);
Y <= z2;
end exemplo1;
```

Blocos combinacionais

```
-- detector de igualdade
entity igualdade is
     Port (a,b: in bit;
           igual: out bit);
end igualdade;
architecture ex1 of igualdade is
begin
   igual <= ‘1’ when a=b else
            ‘0’;
end ex1;
```

Blocos Combinacionais

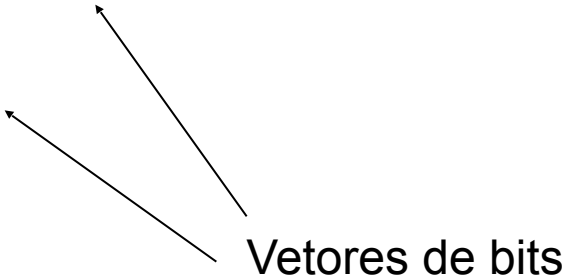
```
-- tabela verdade qualquer
entity tabelaverdade1 is
    Port (a,b,c : in bit;
          f     : out bit);
end tabelaverdade1;
architecture ex1 of tabelaverdade1 is
begin
    f <= ‘1’ when (a=’0’ and b=’0’ and c=’1’) else
         ‘1’ when (a=’1’ and b=’0’ and c=’1’) else
          ‘1’ when (a=’1’ and b=’1’ and c=’0’) else
          ‘1’ when (a=’1’ and b=’1’ and c=’1’) else
         ‘0’;
end ex1;
```

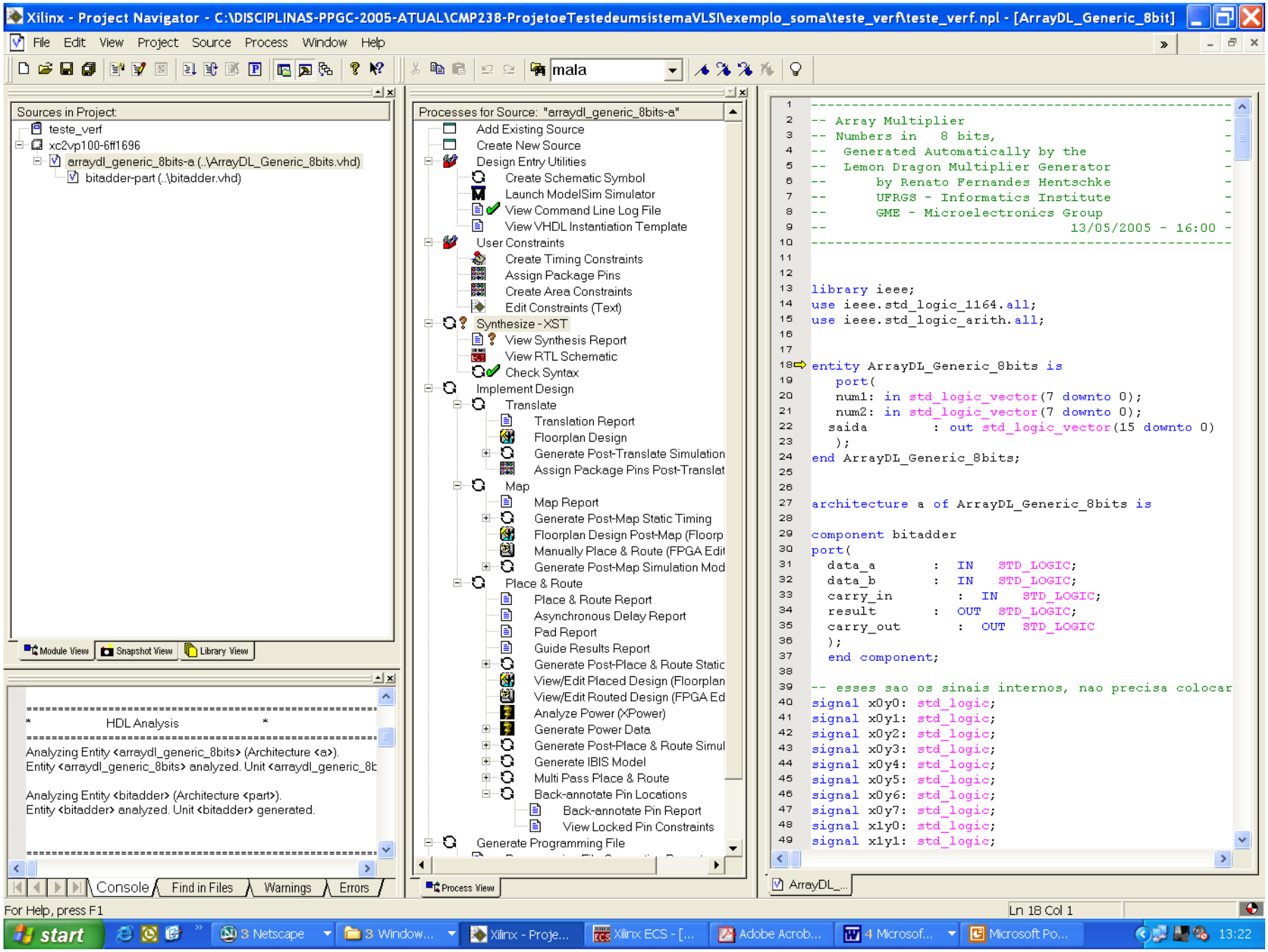
Bloco Combinacional

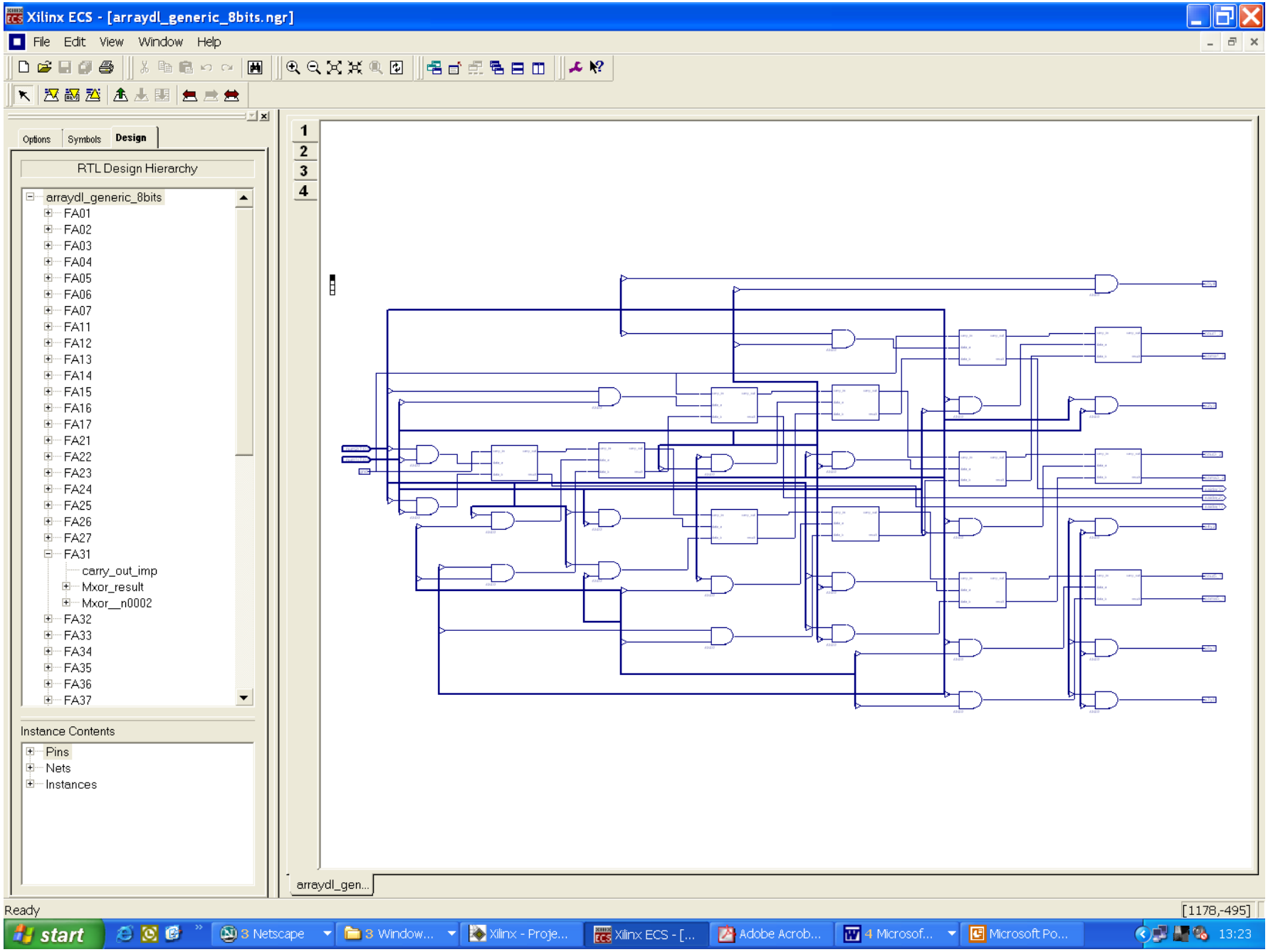
```
-- multiplexador
entity mux41 is
    Port (a0,a1,a2,a3: in bit;
          s1, s0  : in bit;
          f       : out bit);
end tabelaverdade1;

architecture ex1 of mux41 is
begin
    f <= a0  when (s1='0' and s0='0') else
         a1 when (s1='0' and s0='1') else
          a2 when (s1='1' and s0='0') else
          a3
end ex1;
```

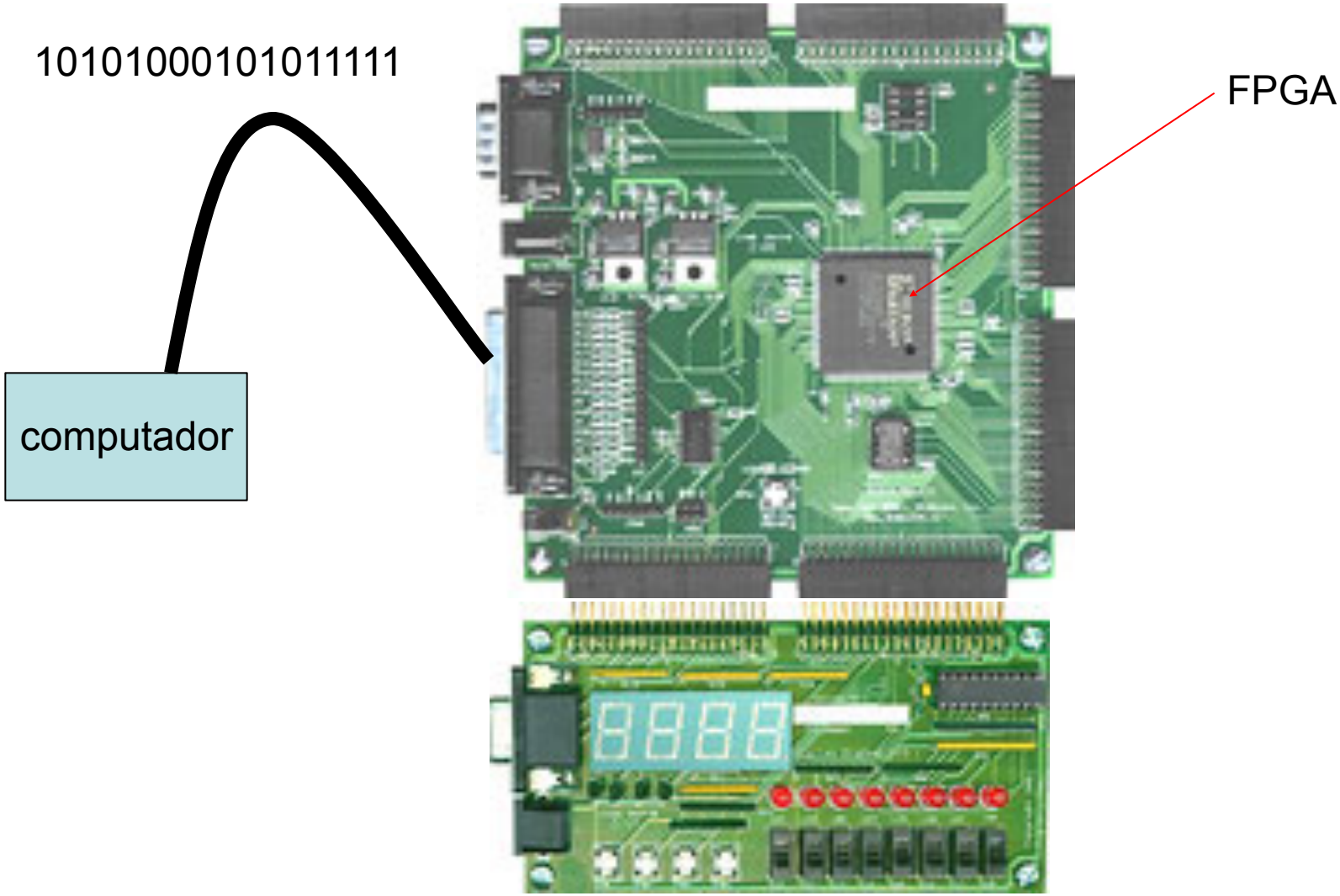
Bloco Combinacional

```
-- multiplexador
entity mux41 is
    Port (a0,a1,a2,a3: in bit_vector(3 downto 0);
          s1, s0  : in bit;
          f       : out bit_vector(3 downto 0));
end tabelaverdade1;

architecture ex1 of mux41 is
begin
    f <= a0  when (s1='0' and s0='0') else
         a1 when (s1='0' and s0='1') else
          a2 when (s1='1' and s0='0') else
          a3
end ex1;
```




Xilinx - Project Navigator - C:\DISCIPLINAS-PPGC-2005-ATUAL\CMP238-ProjetoeTestedeumsistemaVLSI\exemplo_soma\teste_verf\teste_verf.npl - [ArrayDL_Generic_8bit]
File Edit View Project Source Process Window Help
mala
Sources in Project:
teste_verf
xc2vp100-6ff1696
arraydl_generic_8bits-a (..\ArrayDL_Generic_8bits.vhd)
bitadder-part (..\bitadder.vhd)
Module View
Snapshot View
Library View
Processes for Source: "arraydl_generic_8bits-a"
Add Existing Source
Create New Source
Design Entry Utilities
Create Schematic Symbol
Launch ModelSim Simulator
View Command Line Log File
View VHDL Instantiation Template
User Constraints
Create Timing Constraints
Assign Package Pins
Create Area Constraints
Edit Constraints (Text)
Synthesize - XST
View Synthesis Report
View RTL Schematic
Check Syntax
Implement Design
Translate
Translation Report
Floorplan Design
Generate Post-Translate Simulation
Assign Package Pins Post-Translat
Map
Map Report
Generate Post-Map Static Timing
Floorplan Design Post-Map (Floorp
Manually Place & Route (FPGA Edit
Generate Post-Map Simulation Mod
Place & Route
Place & Route Report
Asynchronous Delay Report
Pad Report
Guide Results Report
Generate Post-Place & Route Static
View/Edit Placed Design (Floorplan
View/Edit Routed Design (FPGA Ed
Analyze Power (XPower)
Generate Power Data
Generate Post-Place & Route Simul
Generate IBIS Model
Multi Pass Place & Route
Back-annotate Pin Locations
Back-annotate Pin Report
View Locked Pin Constraints
Generate Programming File
Process View
* HDL Analysis *
Analyzing Entity <arraydl_generic_8bits> (Architecture <a>).
Entity <arraydl_generic_8bits> analyzed. Unit <arraydl_generic_8b
Analyzing Entity <bitadder> (Architecture <part>).
Entity <bitadder> analyzed. Unit <bitadder> generated.
Console
Find in Files
Warnings
Errors
-- Array Multiplier
-- Numbers in 8 bits,
-- Generated Automatically by the
-- Lemon Dragon Multiplier Generator
-- by Renato Fernandes Hentschke
-- UFRGS - Informatics Institute
-- GME - Microelectronics Group
-- 13/05/2005 - 16:00
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
entity ArrayDL_Generic_8bits is
port(
num1: in std_logic_vector(7 downto 0);
num2: in std_logic_vector(7 downto 0);
saida : out std_logic_vector(15 downto 0)
);
end ArrayDL_Generic_8bits;
architecture a of ArrayDL_Generic_8bits is
component bitadder
port(
data_a : IN STD_LOGIC;
data_b : IN STD_LOGIC;
carry_in : IN STD_LOGIC;
result : OUT STD_LOGIC;
carry_out : OUT STD_LOGIC
);
end component;
-- esses sao os sinais internos, nao precisa colocar
signal x0y0: std_logic;
signal x0y1: std_logic;
signal x0y2: std_logic;
signal x0y3: std_logic;
signal x0y4: std_logic;
signal x0y5: std_logic;
signal x0y6: std_logic;
signal x0y7: std_logic;
signal x1y0: std_logic;
signal x1y1: std_logic;
ArrayDL_...
For Help, press F1
Ln 18 Col 1
start
3 Netscape
3 Window...
Xilinx - Proje...
Xilinx ECS - [...
Adobe Acrob...
4 Microsof...
Microsoft Po...
13:22


Xilinx ECS - [arraydl_generic_8bits.ngr]
File Edit View Window Help
Options Symbols Design
RTL Design Hierarchy
arraydl_generic_8bits
FA01
FA02
FA03
FA04
FA05
FA06
FA07
FA11
FA12
FA13
FA14
FA15
FA16
FA17
FA21
FA22
FA23
FA24
FA25
FA26
FA27
FA31
carry_out_imp
Mxor_result
Mxor__n0002
FA32
FA33
FA34
FA35
FA36
FA37
Instance Contents
Pins
Nets
Instances
1
2
3
4
arraydl_gen...
Ready
[1178,-495]
start
3 Netscape
3 Window...
Xilinx - Proje...
Xilinx ECS - [...
Adobe Acrob...
4 Microsof...
Microsoft Po...
13:23


1010100010101111
computador
FPGA