

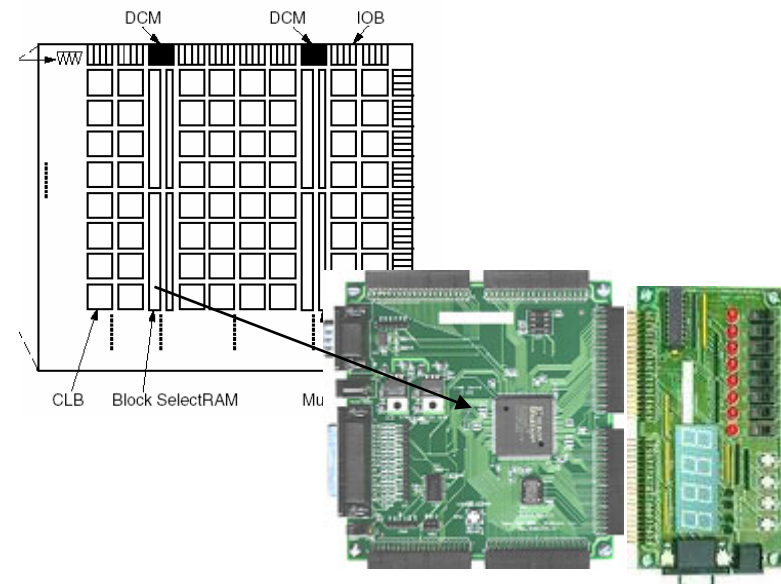
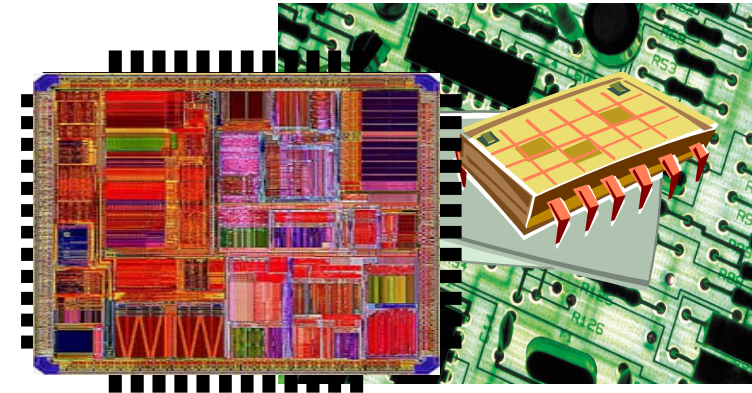
Circuitos Programáveis

- CPLD

- FPGA

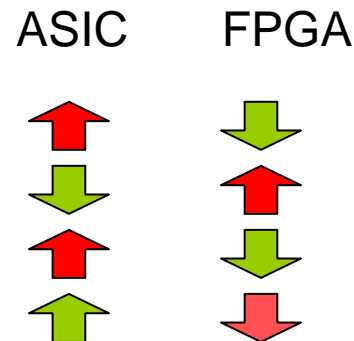
Tipos de componentes

- **Circuito de aplicação específica (ASIC):** circuito integrado projetado especialmente para uma determinada função. Fabricado em uma *foundry* com todos os conjuntos de mascaras de metal, polissilício, dopagem no silício, etc.
 - Full-custom
 - Standard cell (baseado em biblioteca de células já caracterizadas)
- **Lógica programável:** circuito que pode ser customizado e re-programado para realizar diversas funções. Pode ser customizado nas *foundries* (gate array) ou no campo pelo usuário.

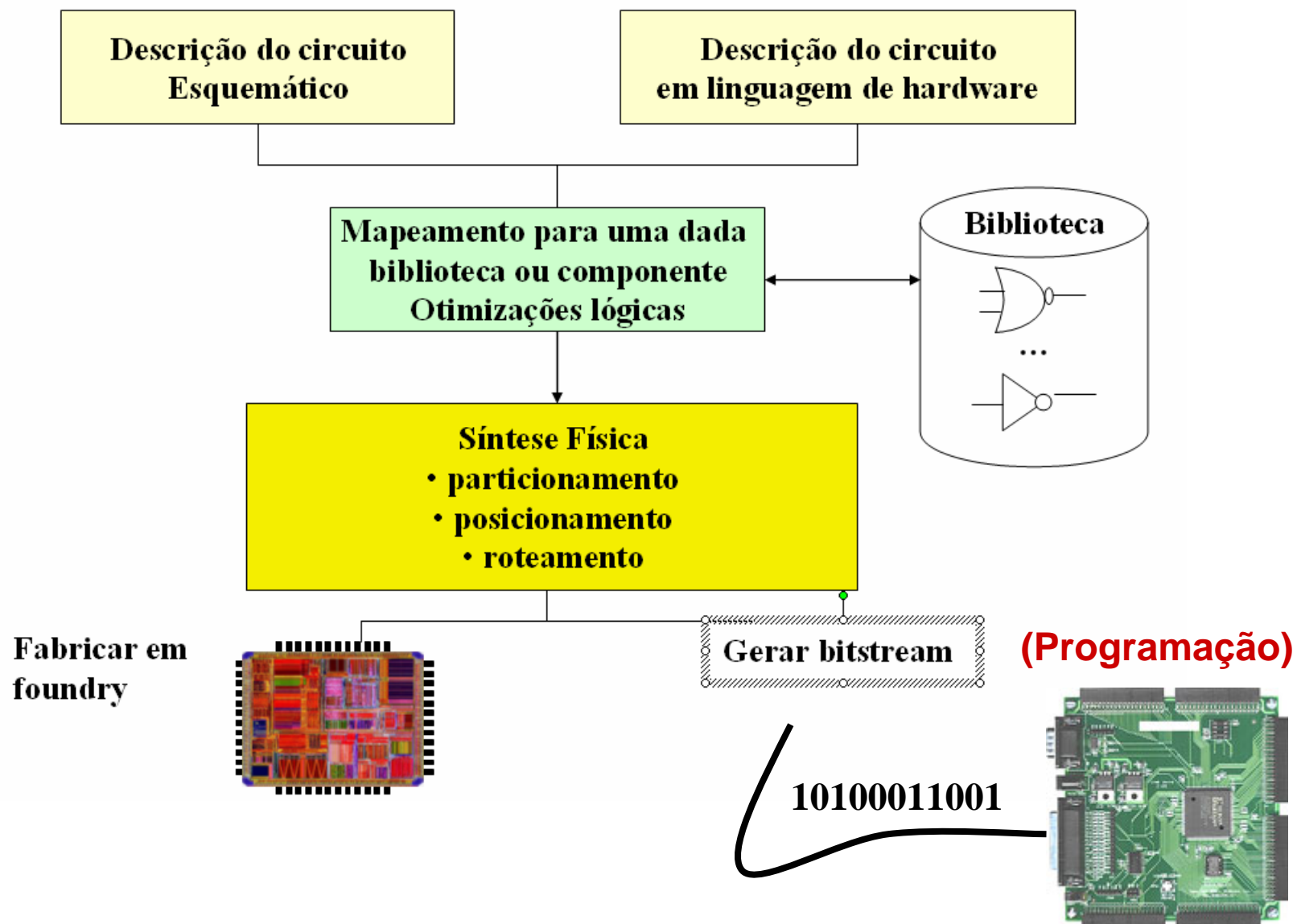


Compromisso:

Custo X
Custo por unidade X
tempo de projeto X
desempenho



Fluxo de Projeto (simplificado)



Circuitos Programáveis

- As PALs estão disponíveis somente em tamanhos pequenos, equivalentes a algumas centenas de portas lógicas.
- Para circuitos lógicos maiores pode-se usar **Complex PLD** ou CPLDs.
- Este contém o equivalente a muitas PALs ligadas por interconexão programadas, tudo num circuito integrado.
- Os CPLDs podem substituir milhares ou até centenas de milhares portas lógicas.
- Algumas CPLDs são programadas usando o programador PAL, mas este método torna-se inconveniente para dispositivos com centenas de pinos.
- O segundo método de programação é soldar o componente à sua placa de circuito impresso, e depois ligar um cabo de série de dados ao PC.
- O CPLD contém um circuito que decodifica os dados e configura-o para realizar a função lógica específica.
- Cada fabricante tem um nome proprietário para este sistema de programação, mas normalmente é dado pela interface **Joint Test Action Group** (JTAG).

Estruturas do Circuito Programável

Aula
18

- Tecnologia de programação (anti-fusível, EPROM/EEPROM/FLASH, memória SRAM)
- Blocos lógicos básicos customizáveis (multiplexadores também conhecido como Lookup Table-LUT, arranjos OR e AND, etc)
- Conexões programáveis
- Ferramenta para projeto e síntese no componente programável.

Principais fabricantes:

- Actel
- Altera
- Xilinx

Conforme o bloco lógico básico e a estrutura das conexões programáveis denominamos de:

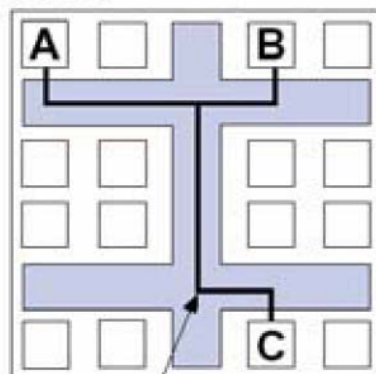
- **CPLD** (Complex Programmable Logic Devices)
- **FPGA** (Field Programmable Gate Array)

Diferenças entre CPLD x FPGA

1) **CPLDs** tem um conjunto grande de **interconexões contínuas** e pre-definidas, enquanto o **FPGA** tem **conexões segmentadas** em tamanhos pequenos.

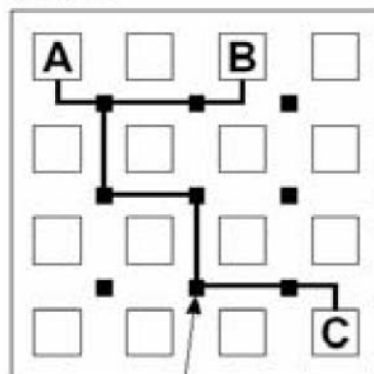
Figure 3. CPLD vs. FPGA Routing Scheme

CPLD Continuous Interconnect Structure



Fixed/Predictable Delay

FPGA Segmented Interconnect Structure



Variable/Unpredictable Delay

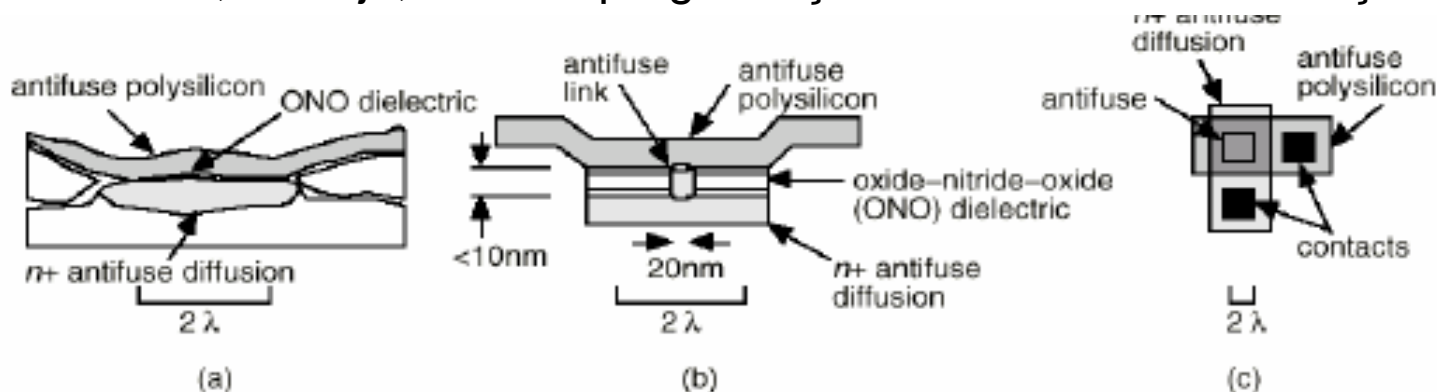
Interconnect structures affect the following device characteristics:

- ☐ Performance predictability
- ☐ In-system performance
- ☐ Logic utilization

2) Os **CPLDs** normalmente tem um **conjunto de arrays AND e OR** lembrando muito os antigos PLAs, enquanto os **FPGAs** são **arrays regulares de multiplexadores** separados em colunas e linhas, o que aumenta a **densidade lógica**.

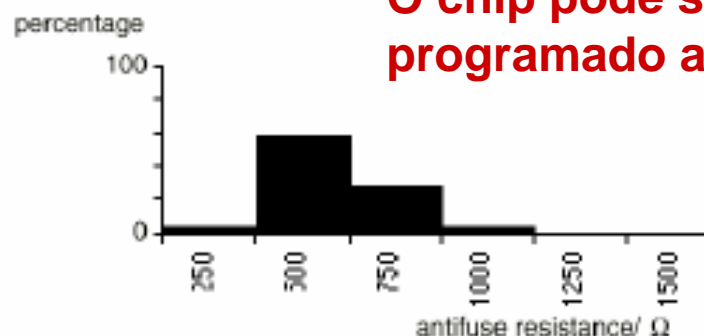
Tecnologia Anti-fusível

- É composto por um elemento que faz ou não a conexão entre duas camadas de metal conforme uma tensão de programação.
- Não é volátil, ou seja, retém a programação mesmo sem alimentação.



Number of antifuses on Actel FPGAs

Device	Antifuses
A1010	112,000
A1020	186,000
A1225	250,000
A1240	400,000
A1280	750,000



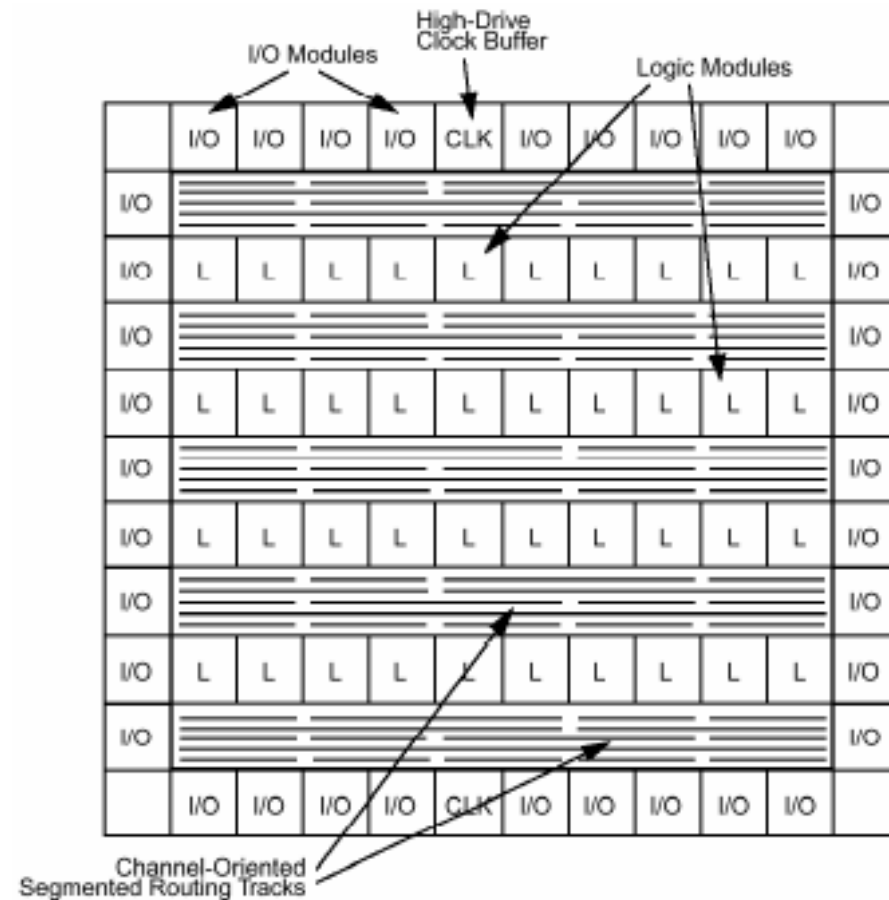
O chip pode ser programado apenas 1 vez.

The resistance of blown Actel antifuses

FPGA da Actel

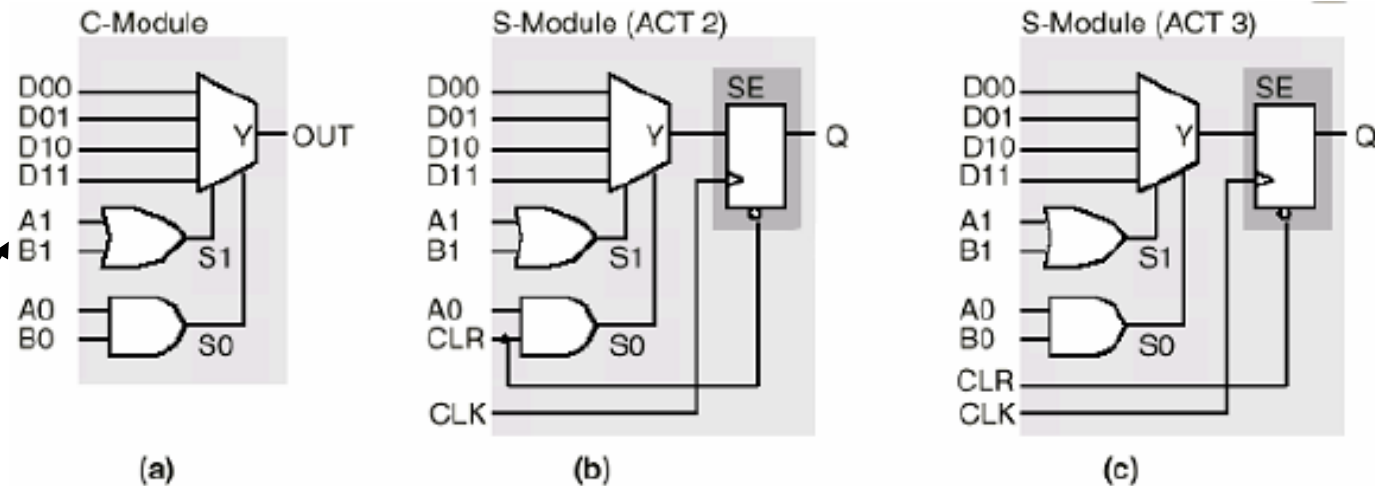
Aula
18

- Usa tecnologia anti-fusível
- Baseado em um array de portas lógicas com canal de roteamento
- Cada elemento lógico programável é baseado em multiplexadores.

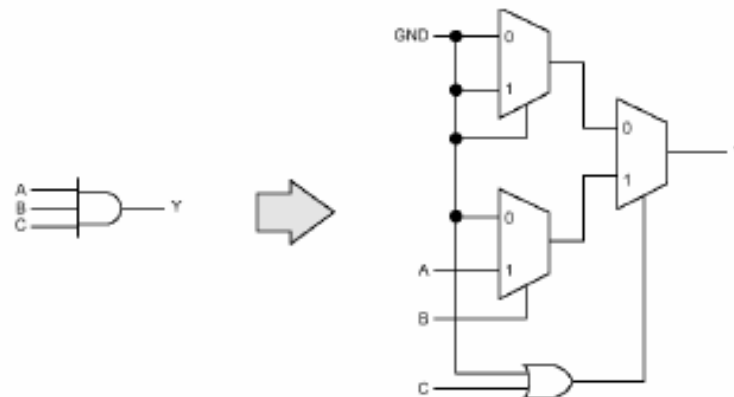


Logic Elements (LE) da Actel

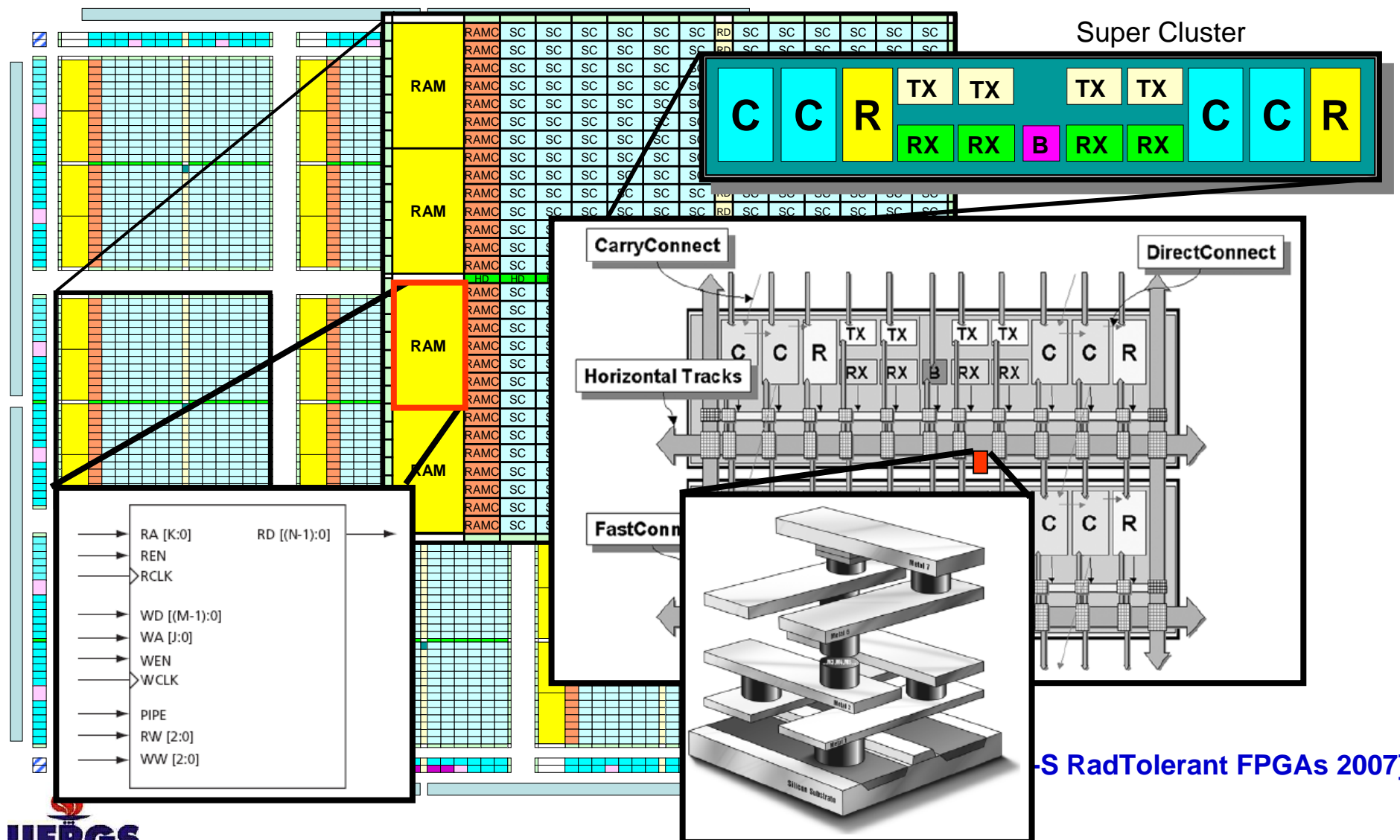
Aula
18



Programa para
implementar
tabelas verdades
de funções
Booleanas

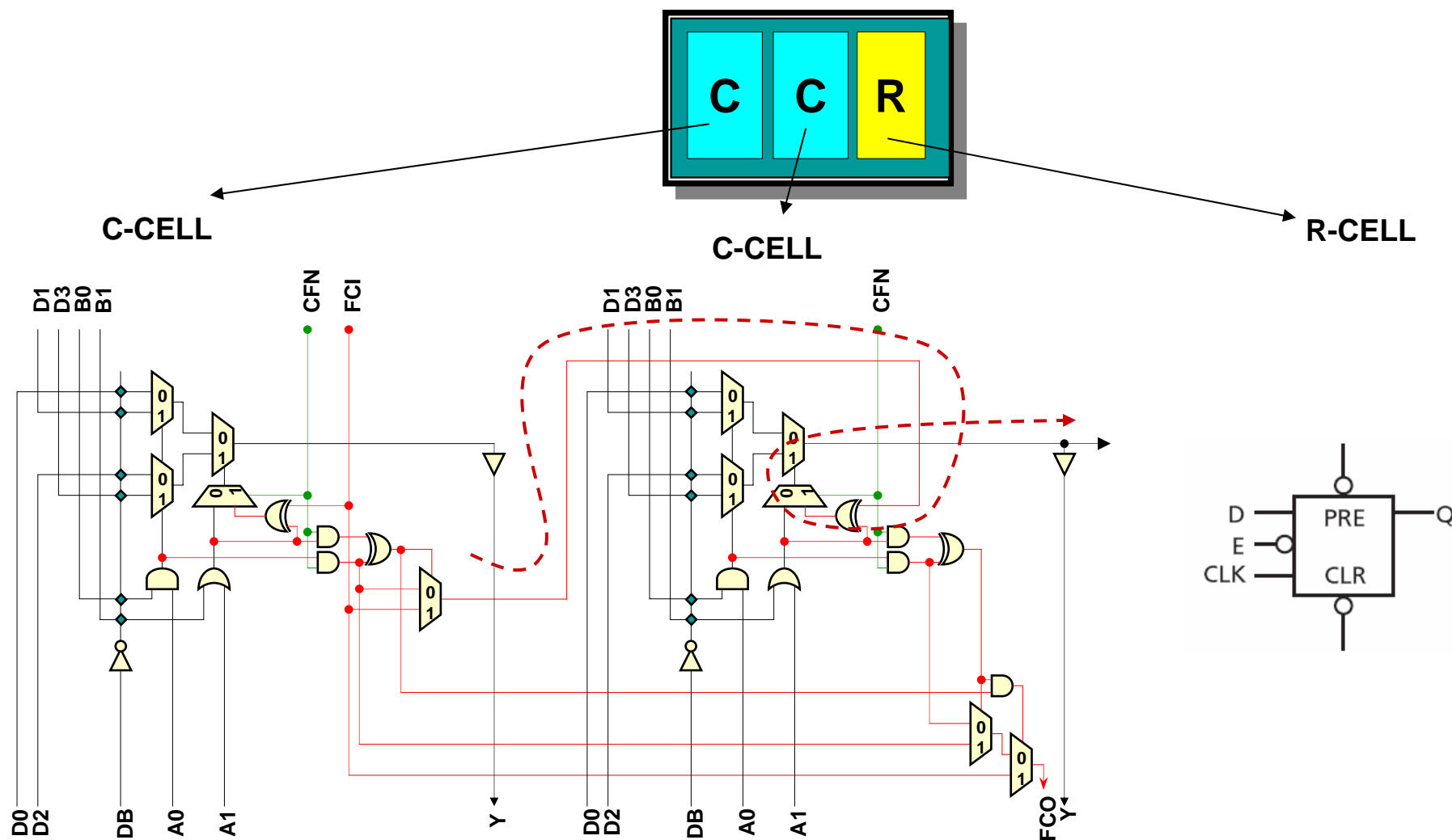


ACTEL: RTAX-S device



-S RadTolerant FPGAs 2007]

ACTEL: RTAX-S device





PRODUCTS & SERVICES

products & services

Search

[Advanced Search](#) [Site Map](#)

Software Tools: Libero Integrated Design Environment (IDE)



Actel's Libero IDE offers the latest and best-in-class tools from leading EDA vendors such as Magma Design Automation, Mentor Graphics, SynaptiCAD, and Synplicity. These tools, combined with custom developed tools from Actel, are integrated into a single FPGA development package. Actel truly offers a one-stop shopping solution to completely orchestrate the [Libero design flow](#) including a powerful design manager that guides you through the design process, keeps track of your design files, and seamlessly manages file exchanges between the various tools.



Libero IDE supports all currently released Actel devices including the new [Fusion](#) AFS600, [ARM-enabled Fusion M7AFS600](#), and [ARM-enabled M7A3P/E](#) devices. Libero IDE is available in several editions to meet budget and tool needs: [Libero Gold](#), [Libero Platinum](#), and a [Libero Platinum Evaluation](#) version. Libero IDE includes Actel's [Designer](#) software, which offers premier back-end tools for physical implementation, including a comprehensive floorplanning capability via Actel's ChipPlanner feature. Timing constraint setting and analysis is performed with SmartTime, a new and highly advanced environment for quickly setting and meeting timing objectives. [Designer](#) is available as a standalone product for those who want to use their own design and verification tools.

Libero IDE

- [Libero Editions](#)
- [Product Brochure](#)
- [User Manuals & Guides](#)
- [Licensing & Registration](#)
- [What's New in Libero](#)
- [Libero Release Notes](#)
- [Request Software](#)

Designer Software

Device Support

Power Calculators

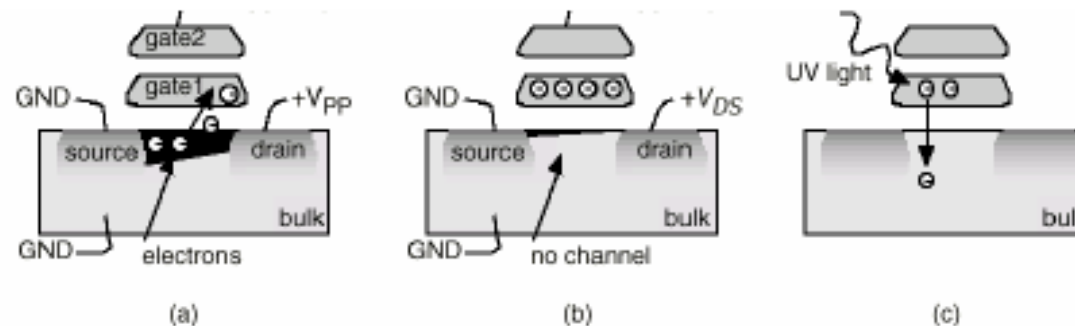
Tools Overview

License Types

Partners

Tecnologia EPROM e EEPROM

- EPROM e EEPROM é uma tecnologia não volátil, ou seja, mantem a programação mesmo na falta de alimentação e é re-programável.
- O chip pode ser programado inúmeras vezes.



- O transistor fabricado em tecnologia EPROM pode ser programado quando uma tensão alta ($\sim 12V$) é aplicada no seu gate. E a programação pode ser apagada com raios ultra-violetas.
- O transistor fabricado em tecnologia EEPROM também é programado por uma tensão alta ($\sim 12V$) aplicada no seu gate e pode ser apagado eletricamente.

FPGA e CPLDs da Altera

Aula
18

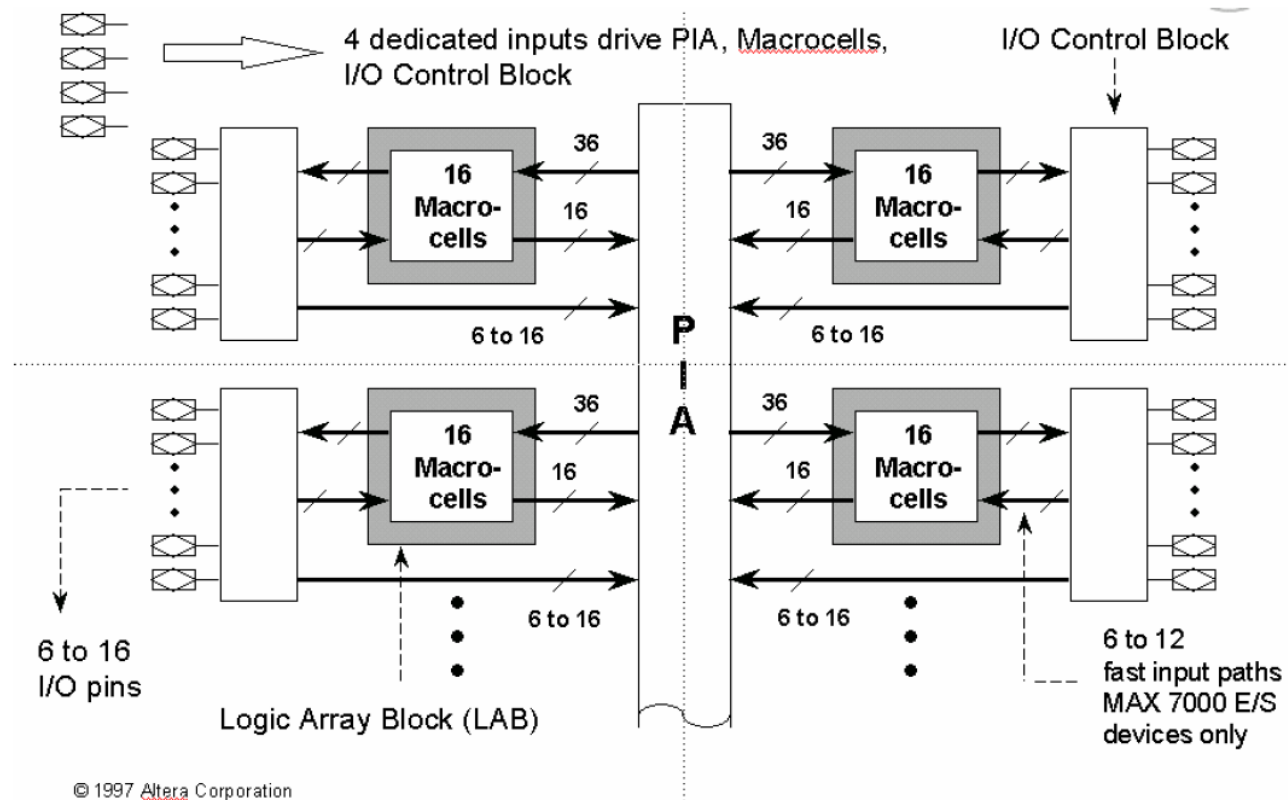
- ◆ CPLD = Complex Programmable Logic Devices
- ◆ FPGA = Field Programming Gate Arrays
- ◆ Altera has four different PLD families:
 - ❖ MAX family – product-term based macrocells CPLDs
 - ❖ FLEX family – SRAM based lookup tables (LUTs)
 - ❖ APEX family – mixture of product-term and LUT based devices
 - ❖ Stratix family – Advanced FPGAs with embedded blocks (Stratix-2 is currently the most advanced FPGA devices)

Família	Número de Gates	Programação
MAX5000	600 a 3,7K	EEPROM
MAX7000	600 a 5K	EEPROM
MAX9000	6K a 12K	EEPROM
FLEX6000	5K a 24K	SRAM
FLEX8000	2,5K a 16K	SRAM
FLEX10K	10K a 250K	SRAM
FLEX20K	53K a 1000K	SRAM
Mercury	120k a 350k	SRAM
Apex	700k a 2M	SRAM
ApexII	1.9M a 5.2M	SRAM
Ciclone		SRAM
Stratix	10k a 40k LE	SRAM

Bloco Lógico do PLD MAX7000

Aula
18

- ◆ Consists of Logic Array Blocks (LABs), each with 16 macro-cells
- ◆ PIA = Programmable Interconnect Array



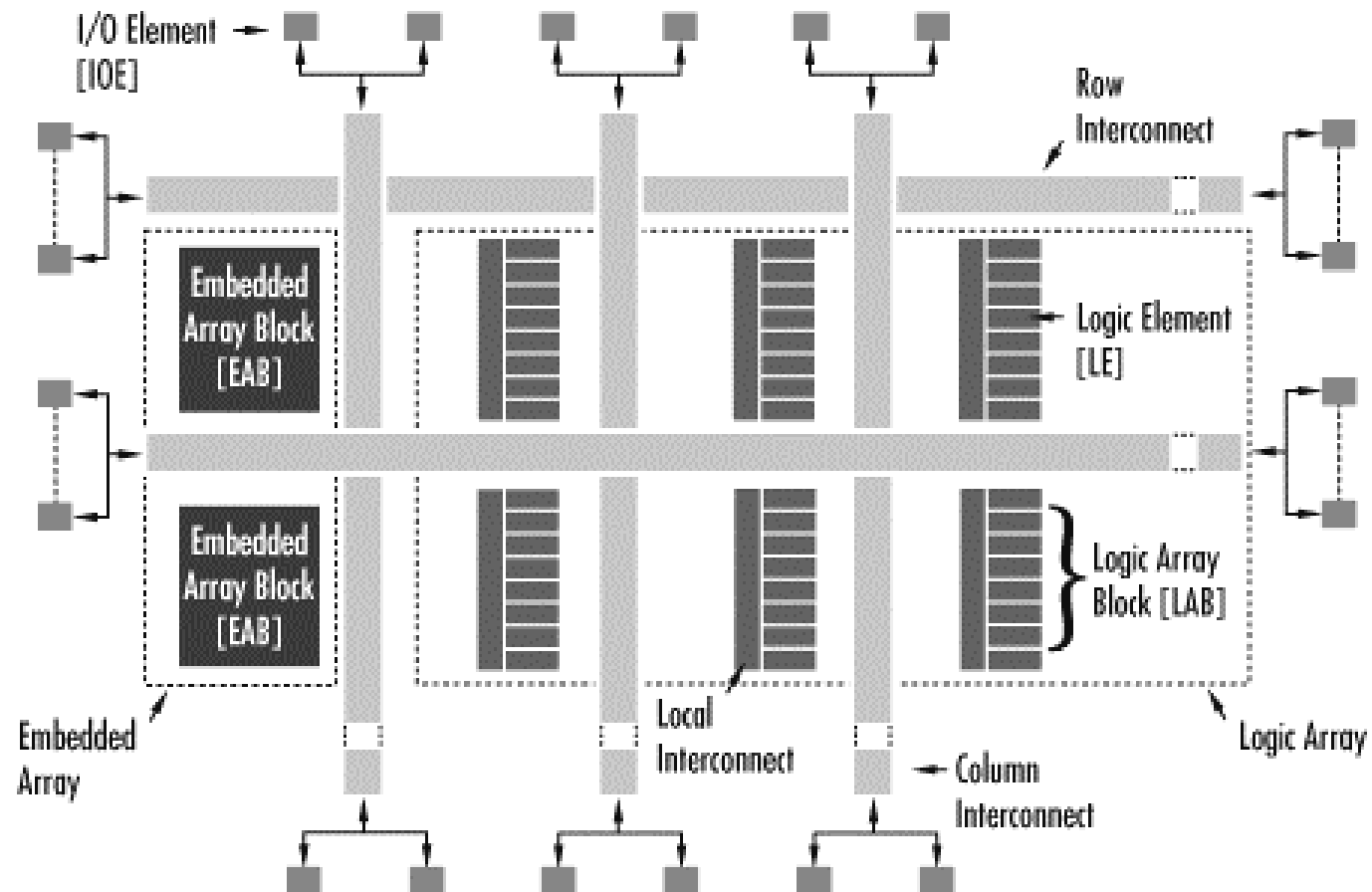
Field Programmable Gate Array

Aula
18

FPGAs comerciais

Altera

FLEX 10K



Field Programmable Gate Array

FPGAs comerciais

Altera

FLEX 10K

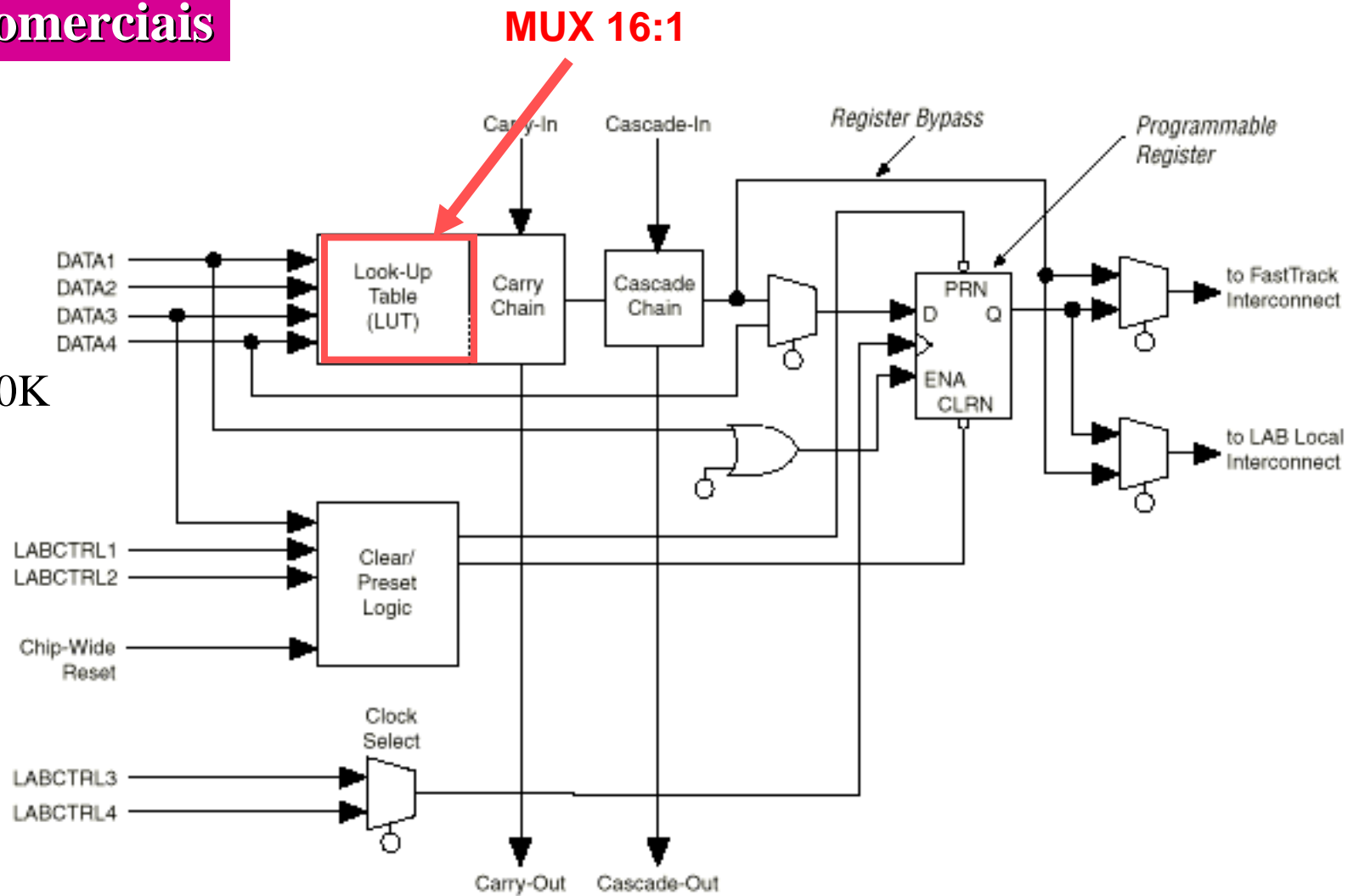


Figure 1. Mercury Architecture Block Diagram Note (1)

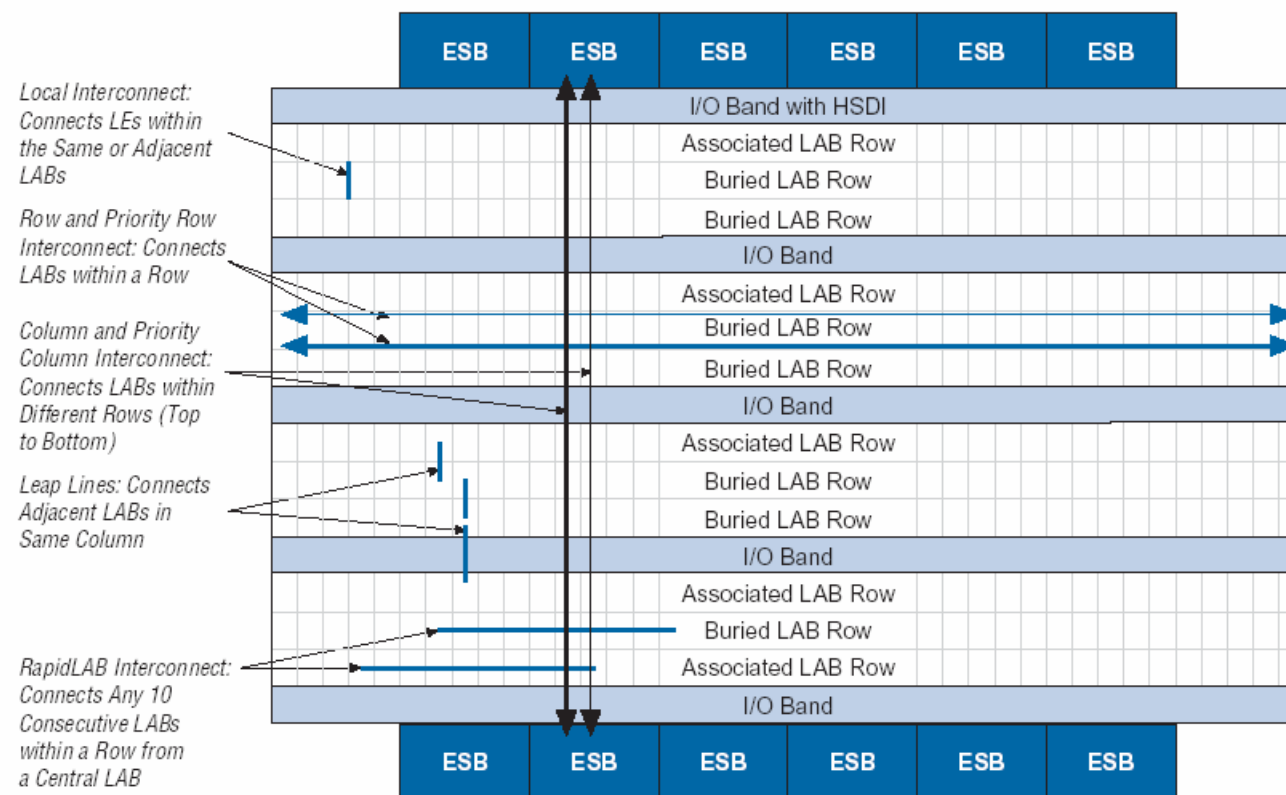


Figure 7. Mercury LE

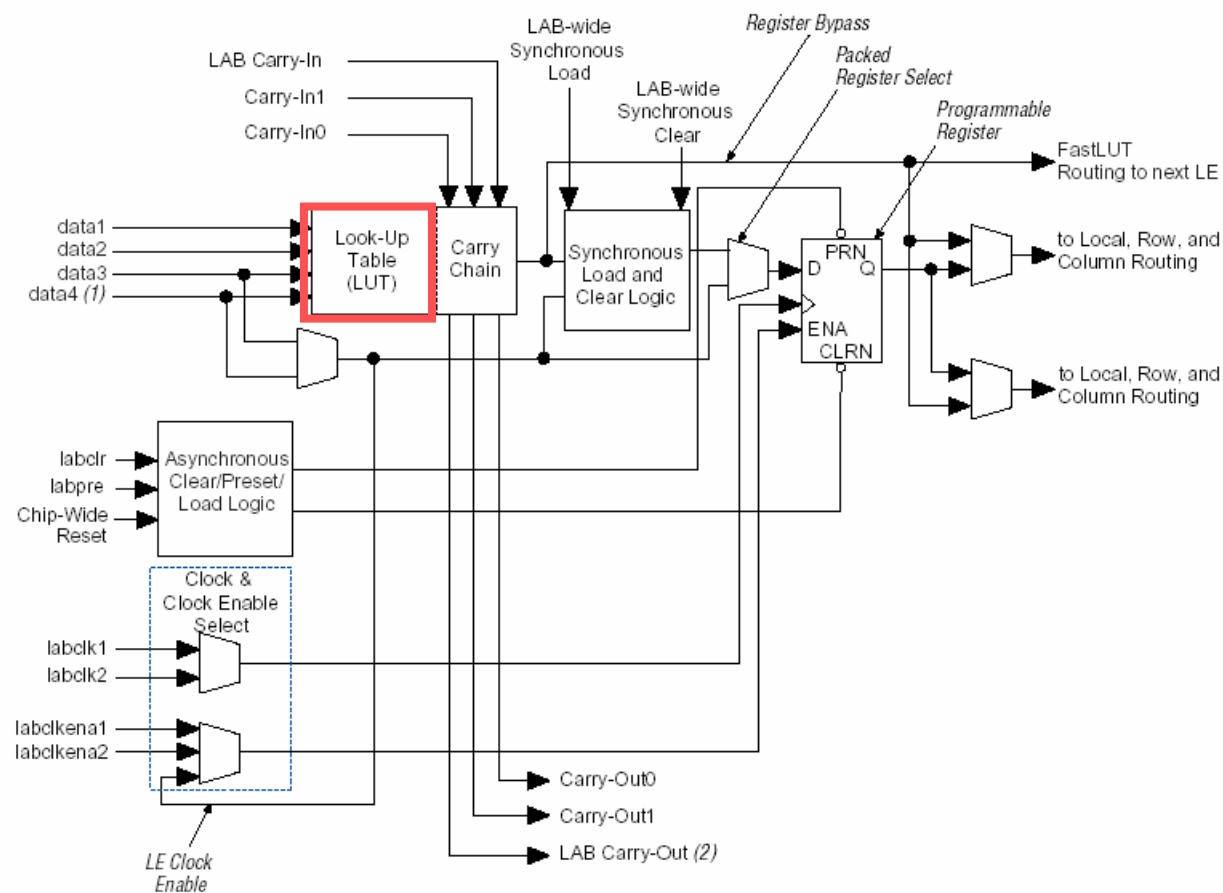


Figure 1. APEX 20K Device Block Diagram

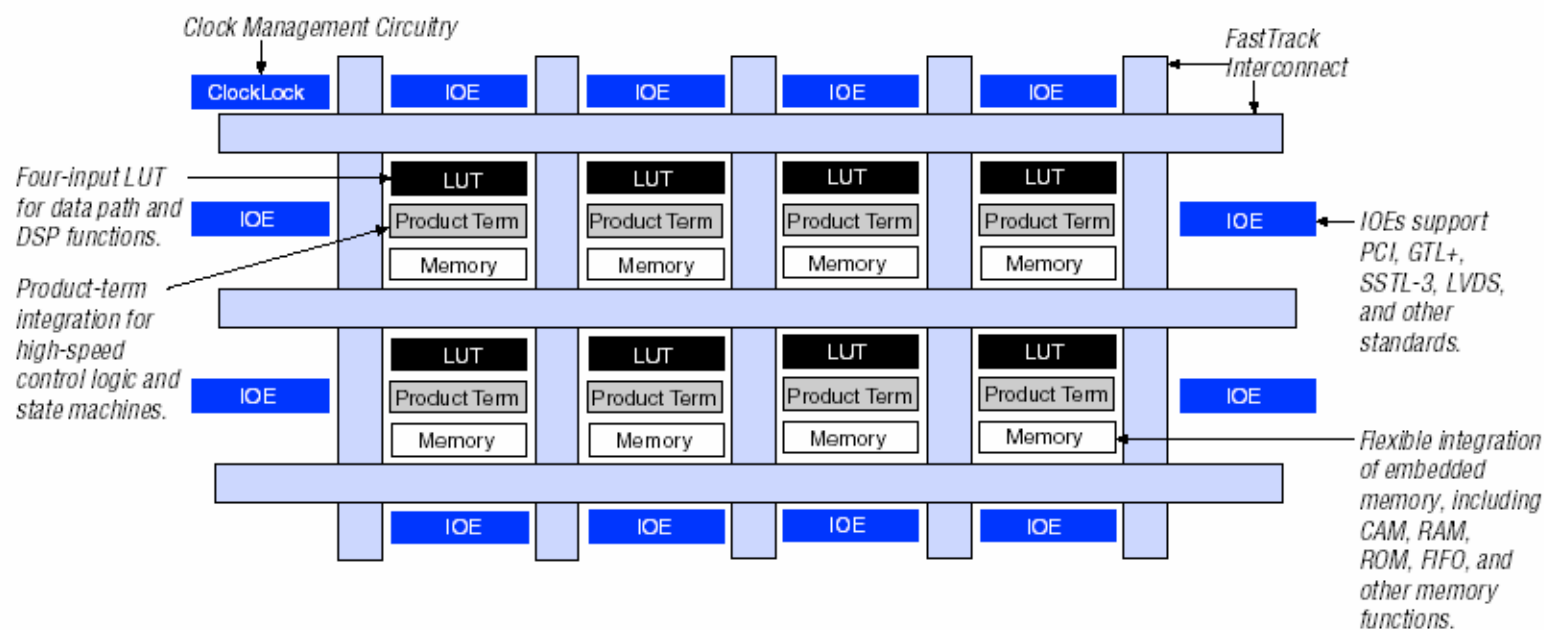
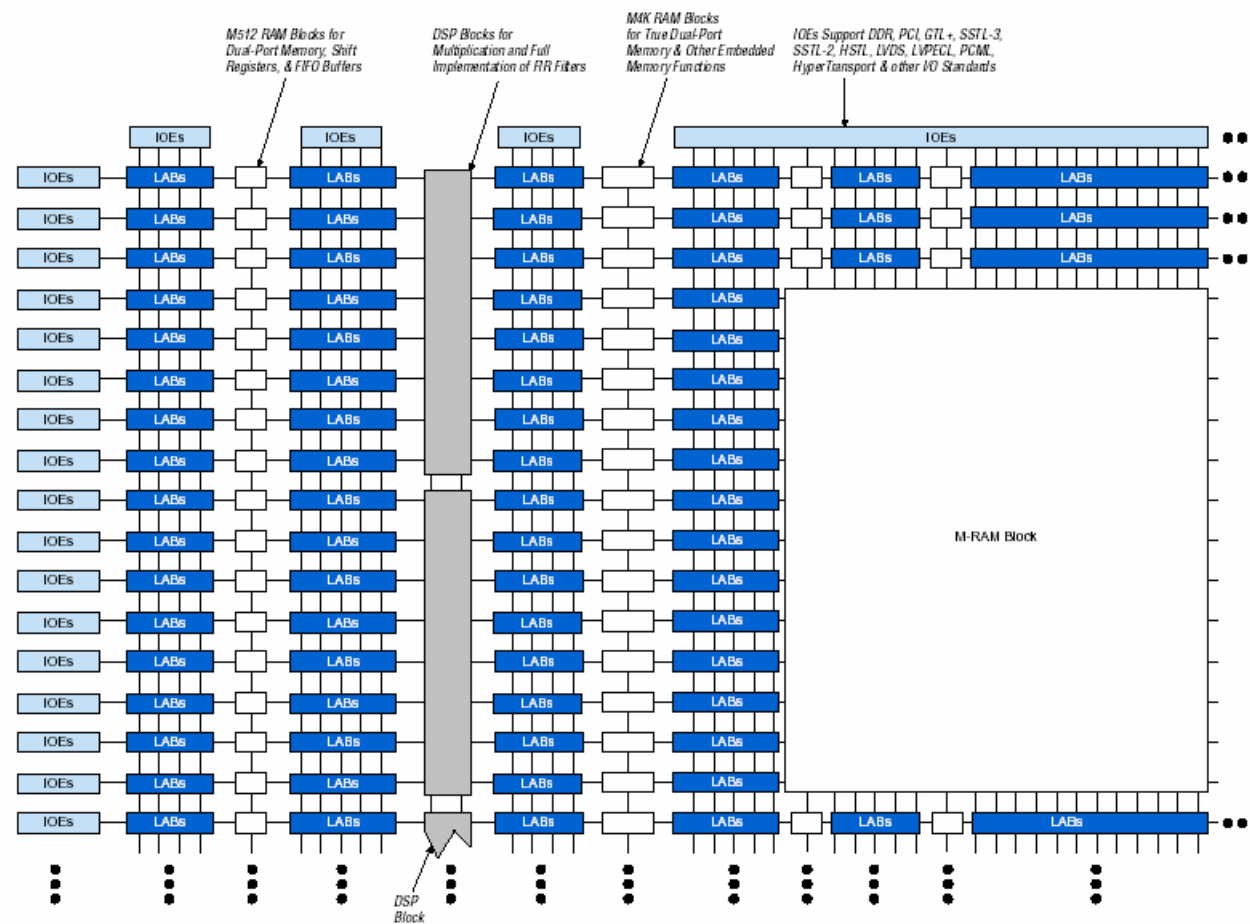


Figure 1–2. Stratix GX Block Diagram



Aula
18



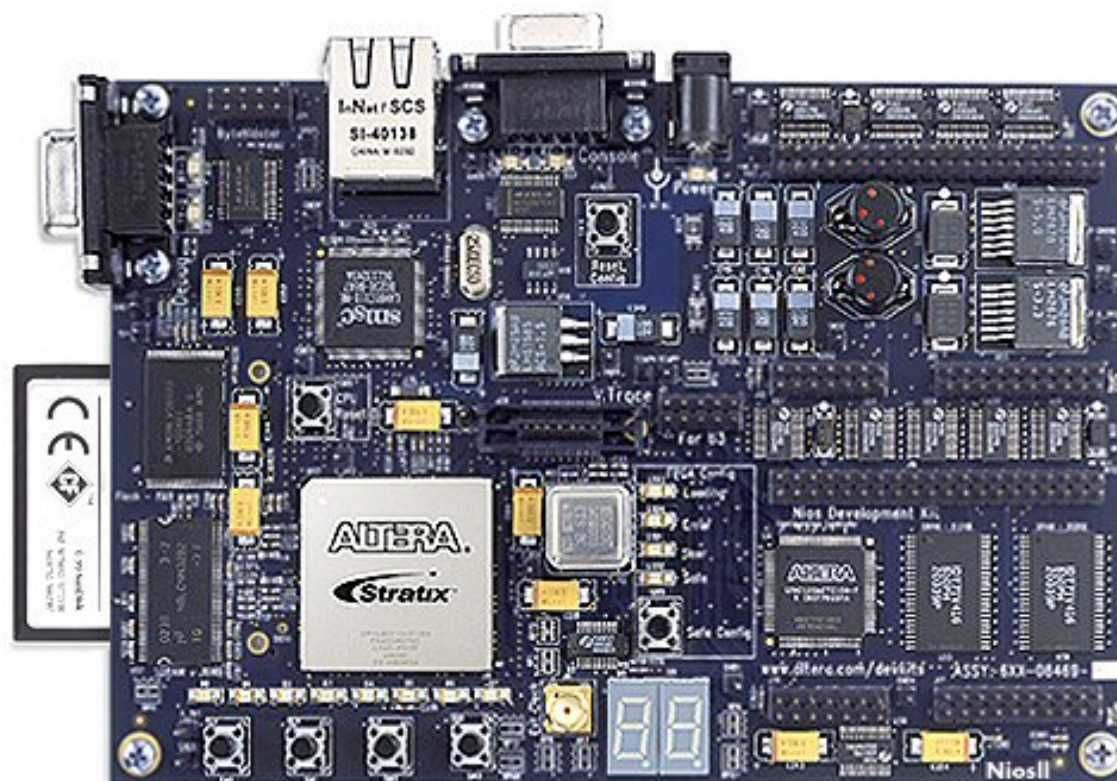
Exemplos de Design Kits




MAX (CPLD)

FLEX (FPGA)

Outros kits...



Com processador embarcado NIOS (processador soft core), ou seja, descrito em linguagem de hardware (VHDL/VERILOG) que pode ser implementado na matriz pelo usuário.



[Literature](#)
[Licensing](#)
[Buy On-Line](#)
[Download](#)

[Entire Site](#)
[Search](#)

[Home](#) | [Products](#) | [Support](#) | [End Markets](#) | [Technology Center](#) | [Education & Events](#) | [Corporate](#) | [Buy On-Line](#)

[Devices](#) | [Design Software](#) | [Intellectual Property](#) | [Design Services](#) | [Dev. Kits/Cables](#) | [Literature](#)

CPLDs

- MAX II
- MAX 3000A
- MAX 7000

FPGAs

- Cyclone III
- Cyclone II
- Cyclone
- Stratix III
- Stratix II
- Stratix
- Stratix II GX
- Stratix GX
- Arria GX

HardCopy ASICs

- About HardCopy Series
- HardCopy III
- HardCopy II
- HardCopy Stratix

Device-Specific Offerings

- Lead-Free
- Extended Temperature
- Industrial Temperature
- Military Temperature
- Automotive Temperature

Configuration Devices

- Enhanced Configuration
- Serial Configuration

Mature Products

- Product Listing

[Please Give Us Feedback](#)

[Home](#) > [Products](#) > [Devices](#)

Altera Devices

Low-Cost FPGAs



- Built from the ground up for low cost
- 60% faster than competing FPGAs
- Low power consumption

[Cyclone® III](#) [Cyclone II](#) [Cyclone](#)

High-End FPGAs



- High-density, high-end FPGAs
- Integrated GX transceivers variant
- Design entire systems-on-a-chip

[Stratix® III](#) [Stratix II GX](#) [Stratix II](#)

Low-Cost Transceiver-Based FPGAs



- Low-cost, risk-free FPGAs with transceivers
- Optimized for PCIe, GbE, and SRIO
- Simple solution for bridging and endpoint applications

[Arria™ GX](#)

HardCopy ASICs



- Lowest-risk, lowest total cost ASIC
- Seamless prototyping using Stratix series FPGAs
- Ultimate system development methodology

[HardCopy® III](#) [HardCopy II](#)

Low-Cost CPLDs



- Lowest cost CPLD ever
- Lowest power for portable apps
- Instant-on single chip solution

[MAX® II](#) [MAX](#)

[Devices Overview](#)
[Product Catalog \(PDF\)](#)

[Print This Page](#)
[Subscribe to Email](#)

Connect with Arria™ GX FPGAs
[> Learn More](#)

View the Arria™ GX QuickCast Now


View the Cyclone® III Flash Movie


☒ [Rate This Page](#)

Aula
18



Disciplina: Técnicas Digitais – Profa. Dra. Fernanda Gusmão de Lima Kastensmidt

25

Address https://www.altera.com/support/software/download/altera_design/quartus_we/dnl-quartus_we.jsp

Google quartus altera Search 53 blocked Check AutoLink AutoFill Options quartus altera

ALTERA

Literature Licensing Buy On-Line Download

Entire Site Search

Home Products Support End Markets Technology Center Education & Events Corporate Buy On-Line
mySupport Devices Design Software Intellectual Property Design Examples

Products

- Quartus II
 - SOPC Builder
 - ModelSim-Altera
 - MAX+PLUS II

Software Resources

- OS Support
- Driver Installation
- Technical Documentation

Download & Licensing

Download

Licensing

Quartus II EDA Support

- Quartus II Interface
- Synthesis Tools
- Simulation Tools
- Verification Tools
- Timing Analysis Tools
- Physical Synthesis Tools

Legacy Sw. EDA Support

- View by Vendor
- View by Tool
- View by Function

Home > Support > Design Software > Download > **Quartus II Web Edition Software**

Print This Page
E-mail This Page

Quartus II Web Edition Software v6.0 Service Pack 1

Now Includes Nios® II Embedded Design Suite, Evaluation Edition Download

1 Choose File → 2 Sign In → 3 Download

Quartus® II Web Edition Software v6.0 Service Pack 1	Download	File Size
Windows Windows XP and Windows 2000	Download ▶ Requires a License	269 MB
Nios II Embedded Design Suite, Evaluation Edition v6.0		
Windows Windows XP and Windows 2000	Download ▶	493 MB

System Requirements

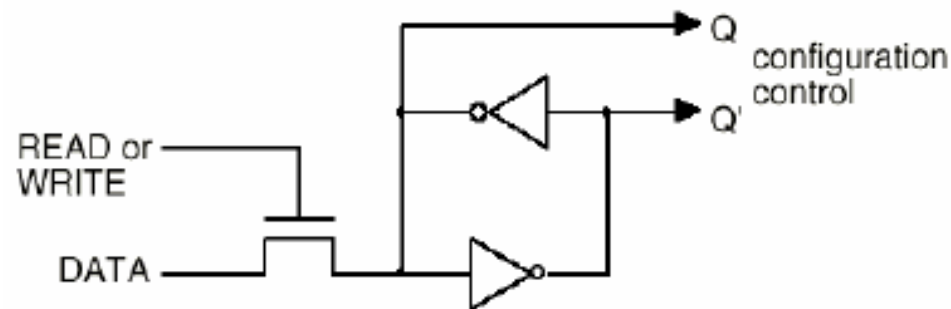
Requirement	Type
Operating System (1)	Windows XP Windows 2000 For Solaris or Linux support, purchase an Altera® software subscription

Note:

- Beginning with version 5.1, the Quartus II Web Edition software no longer supports Windows NT.

Tecnologia SRAM

- A maioria dos FPGAs são programados por SRAM.
- A programação fica armazenada em uma célula de memória SRAM que controla transistores de passagem e multiplexadores para configurar a lógica e roteamento.
- É volátil, ou seja, perde a programação na falta de alimentação.
- Pode ser programada inúmeras vezes pelo usuário através da leitura do bistream (array de bits de programação) para dentro do FPGA.



Field Programmable Gate Arrays

FPGAs comerciais

Xilinx

- A empresa Xilinx foi fundada em 1984 em San José (Califórnia, USA) e foi ela que introduziu o FPGA. Hoje em dia, esta empresa domina cerca de 50 % do mercado em FPGAs.

Família	Número de Portas	Característica
XC2000	1,2K a 1,8K	-
XC3000	2K a 9K	Low-power
XC4000E	2K a 20K	Low-power
XC4000XL/XLA	10K a 200K	High-density
XC4000XV	75K a 500K	High-density
XC5200	3 K a 23K	Low-power
SPARTAN/XL	2K a 40K	Low-power
VIRTEX	50K a 1M	High-density
SPARTAN-2	1k a 15k	LOW COST
SPARTAN-3	2k a 33k	LOW COST
VIRTEXII	40k a 8 M	High density
VIRTEXII-PRO	Power-PC inside	
VIRTEX4	13K a 200K	Low-power
VIRTEX4-FX	Power-PC inside	
VIRTEX5		

Tecnologia CMOS

220nm

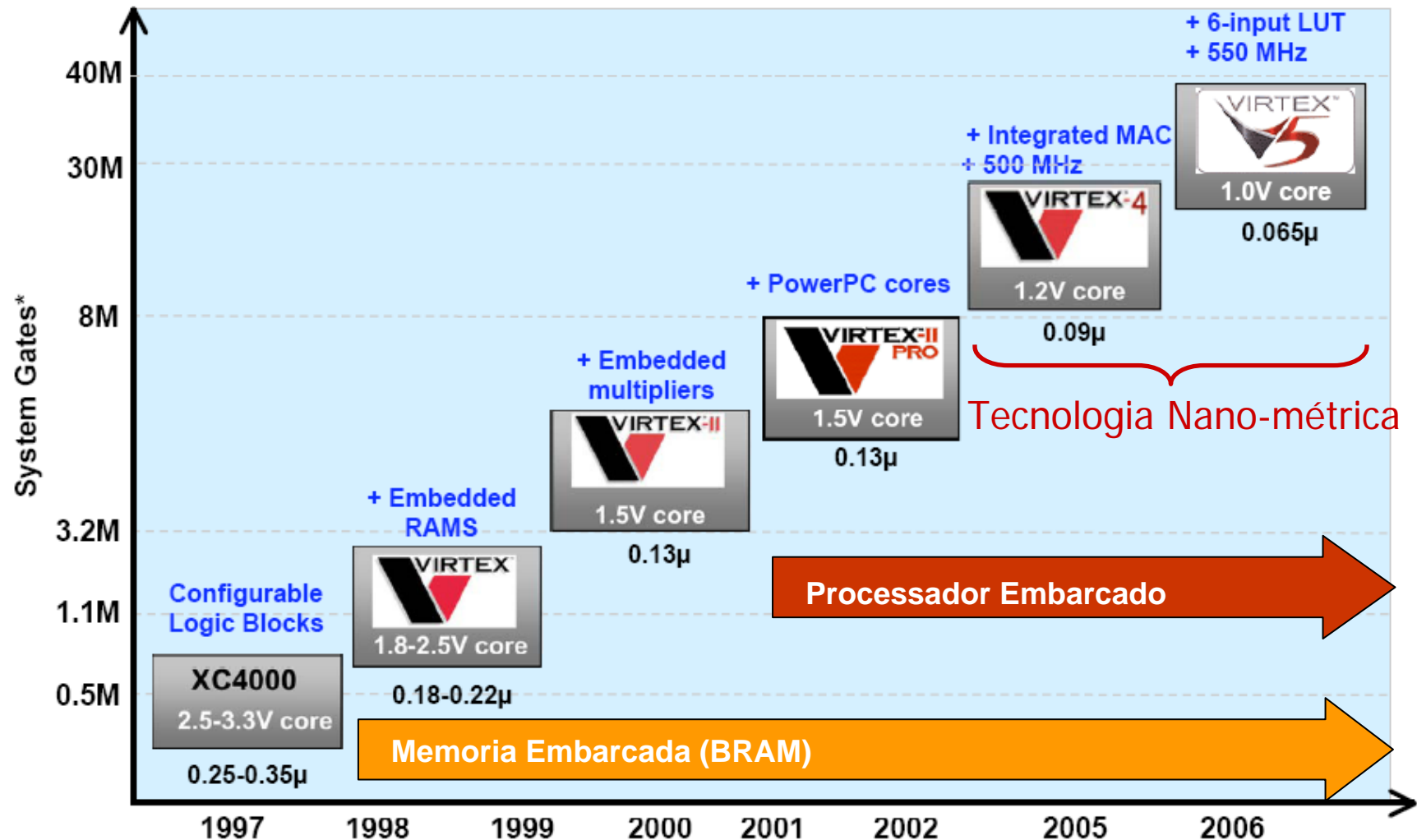
130nm

90nm

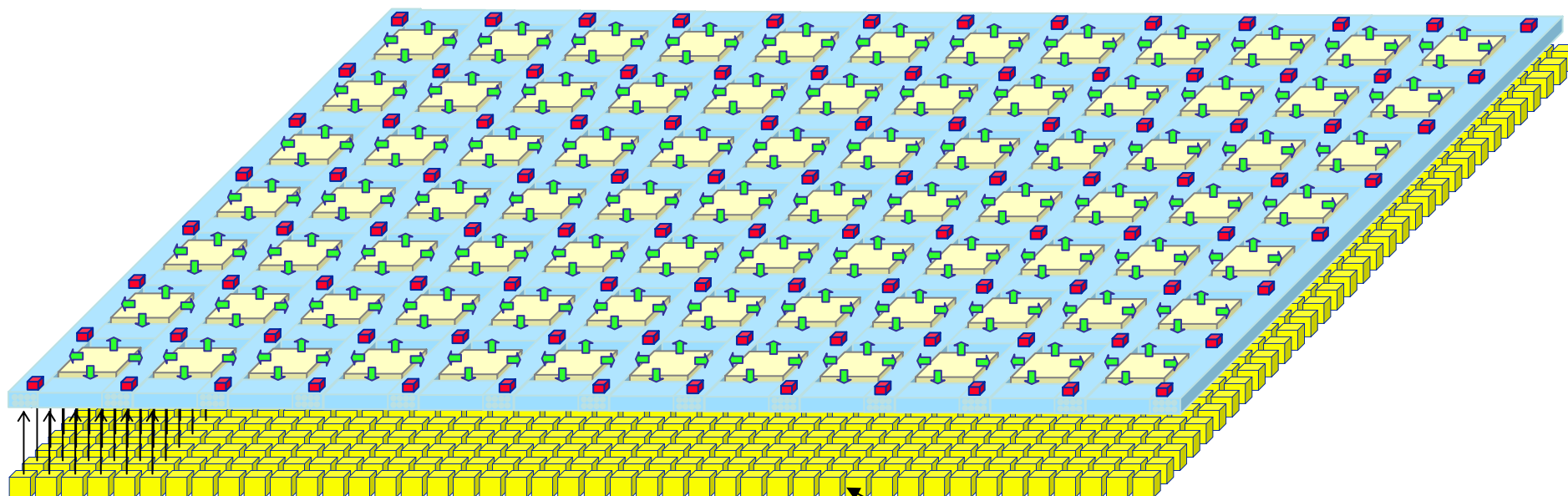
65 nm

Technology Scaling in Xilinx FPGAs

Aula
18



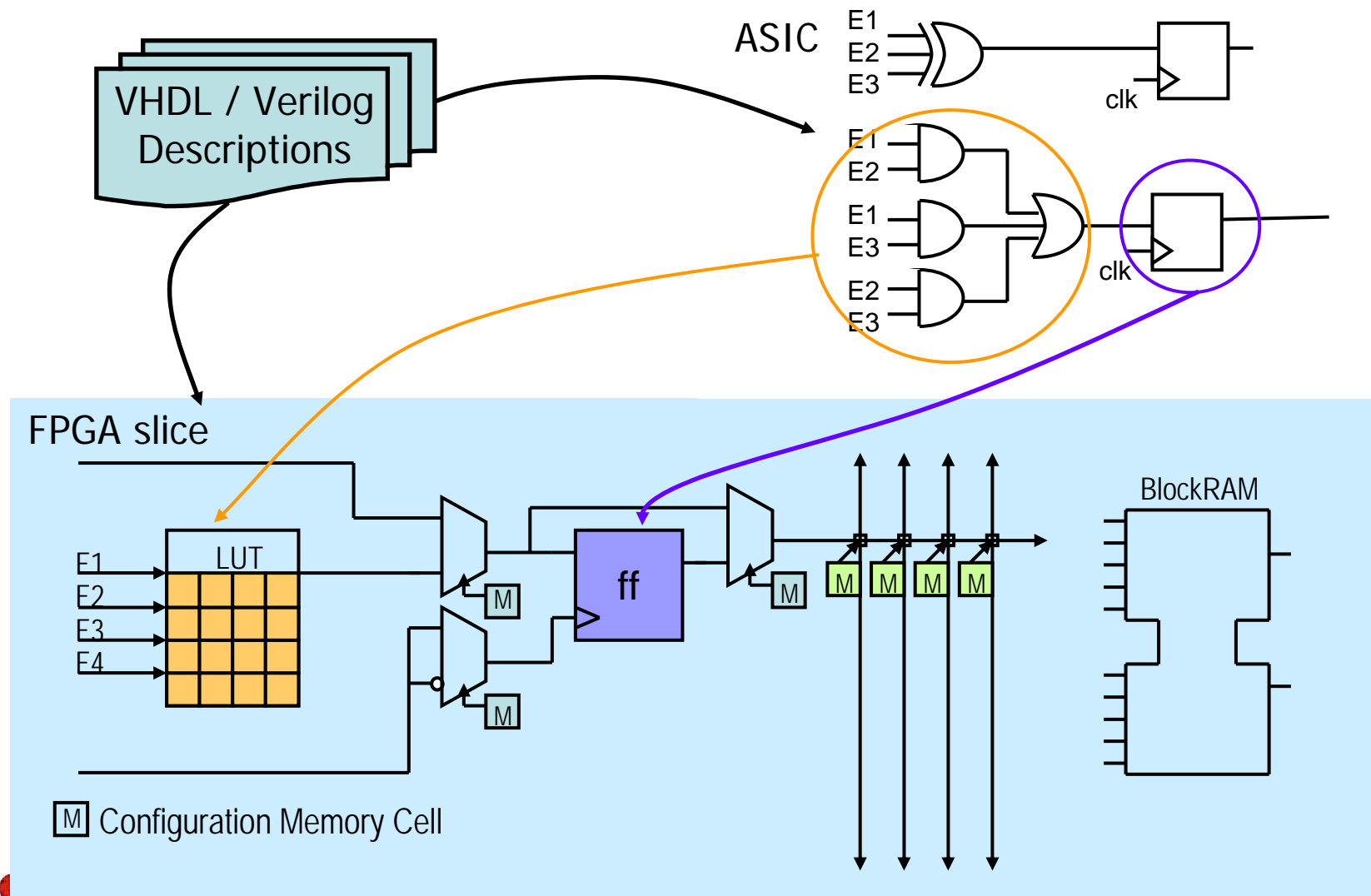
A informação é customizada por um vetor de bits chamado de **BITSTREAM** (set of SRAM bits)



Essas são as células
SRAM que guardam a
programação

Mapeamento lógico SRAM-based FPGAs

Aula
18

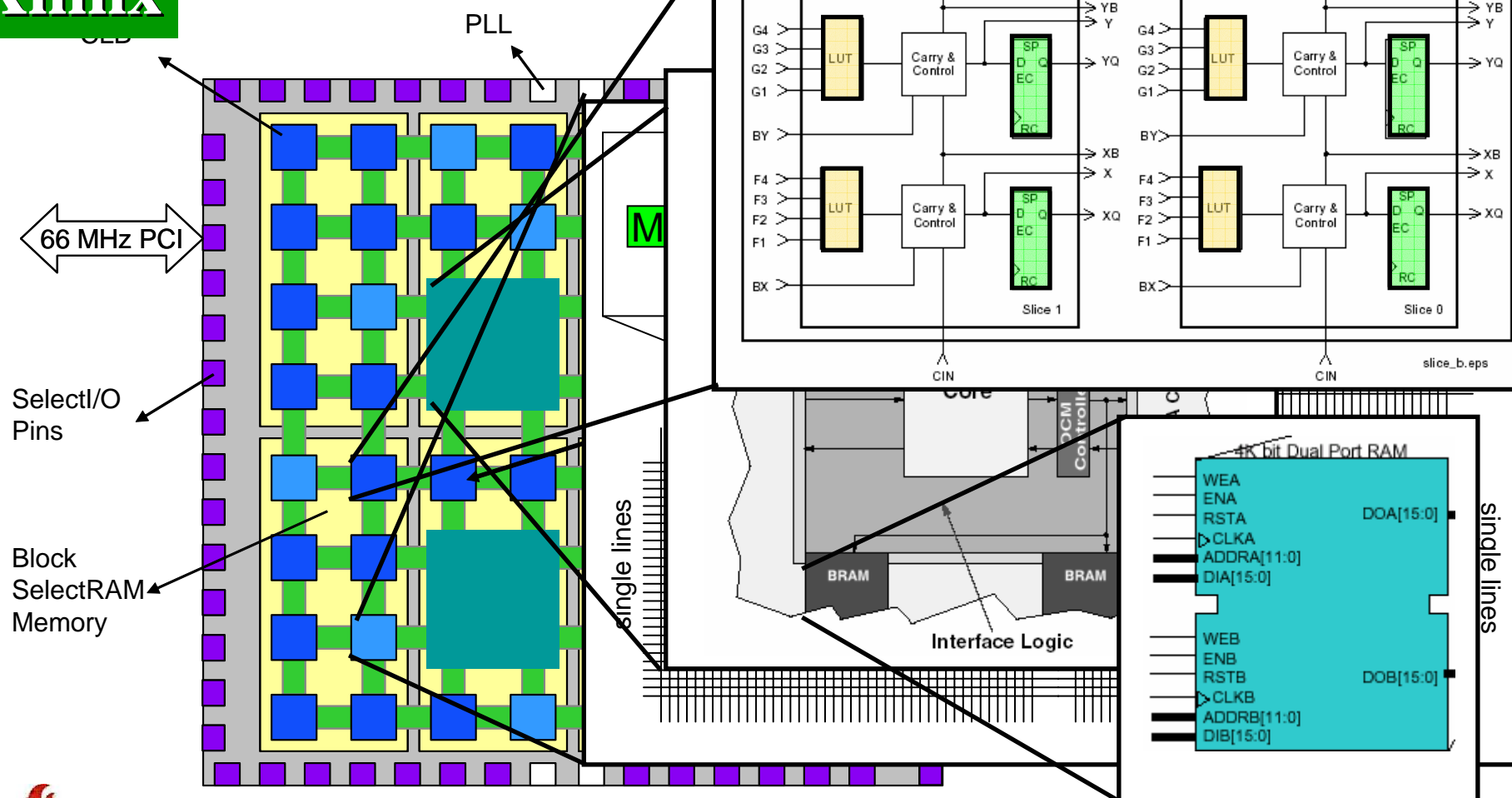


Field Programmable Gate Arrays

FPGAs comerciais

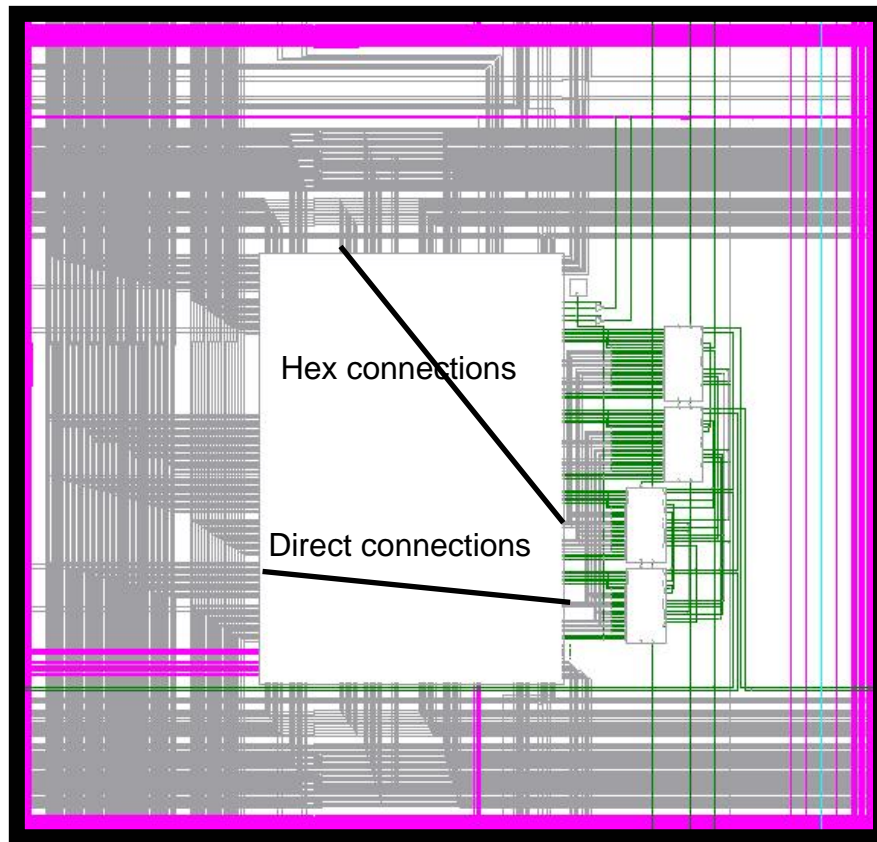
Xilinx

Virtex Family from Xilinx, Inc.

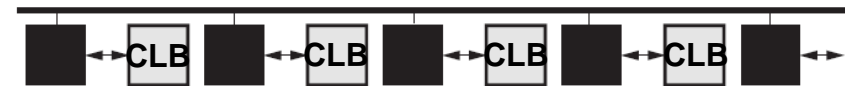


General Routing Matrix (GRM) - VirtexII

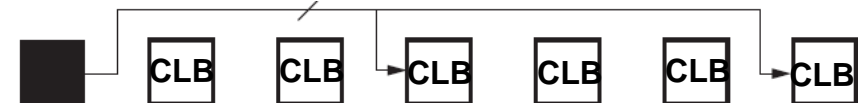
Aula
18



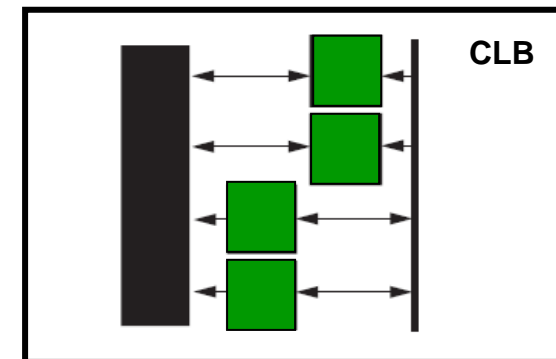
Long lines



Hex lines

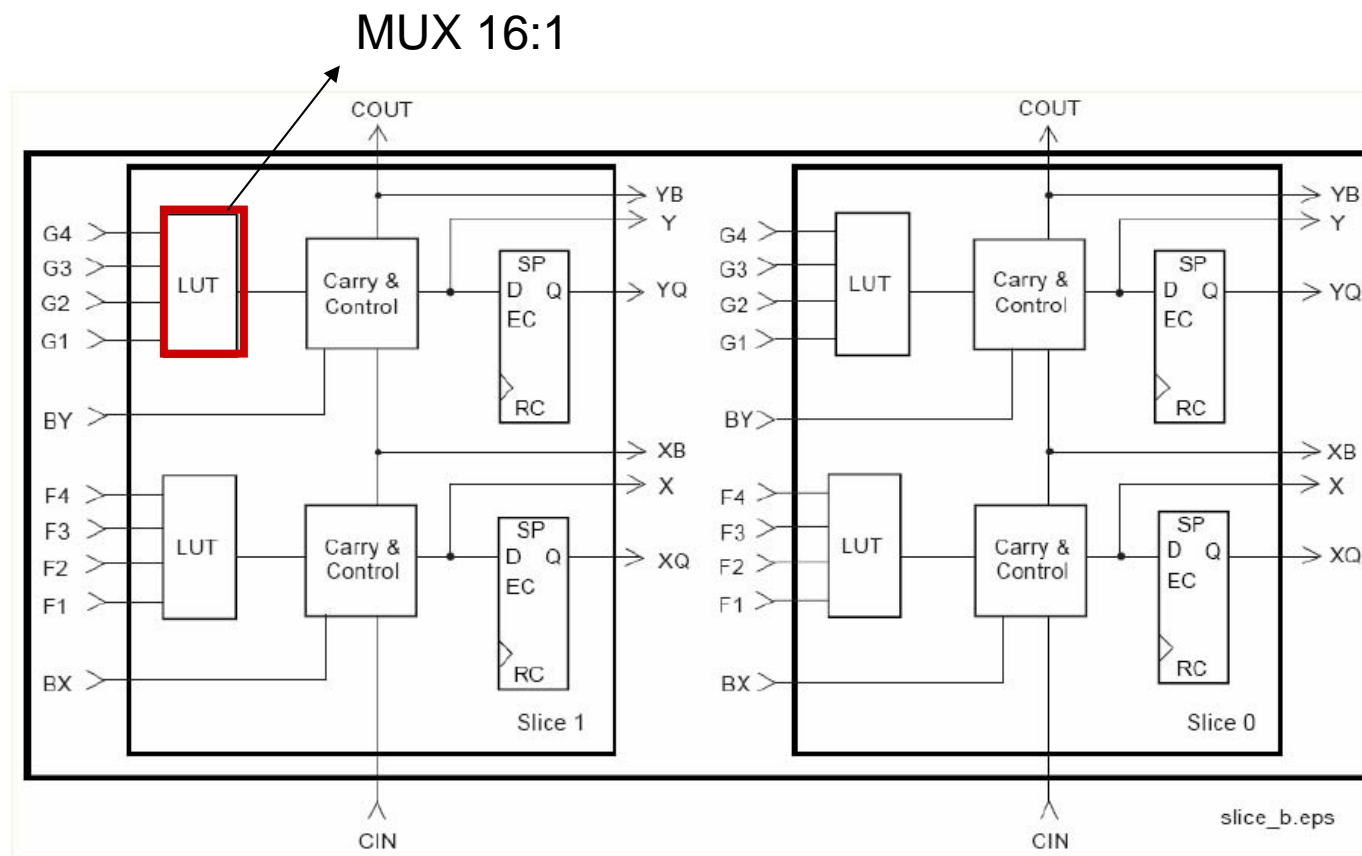


Fast connect



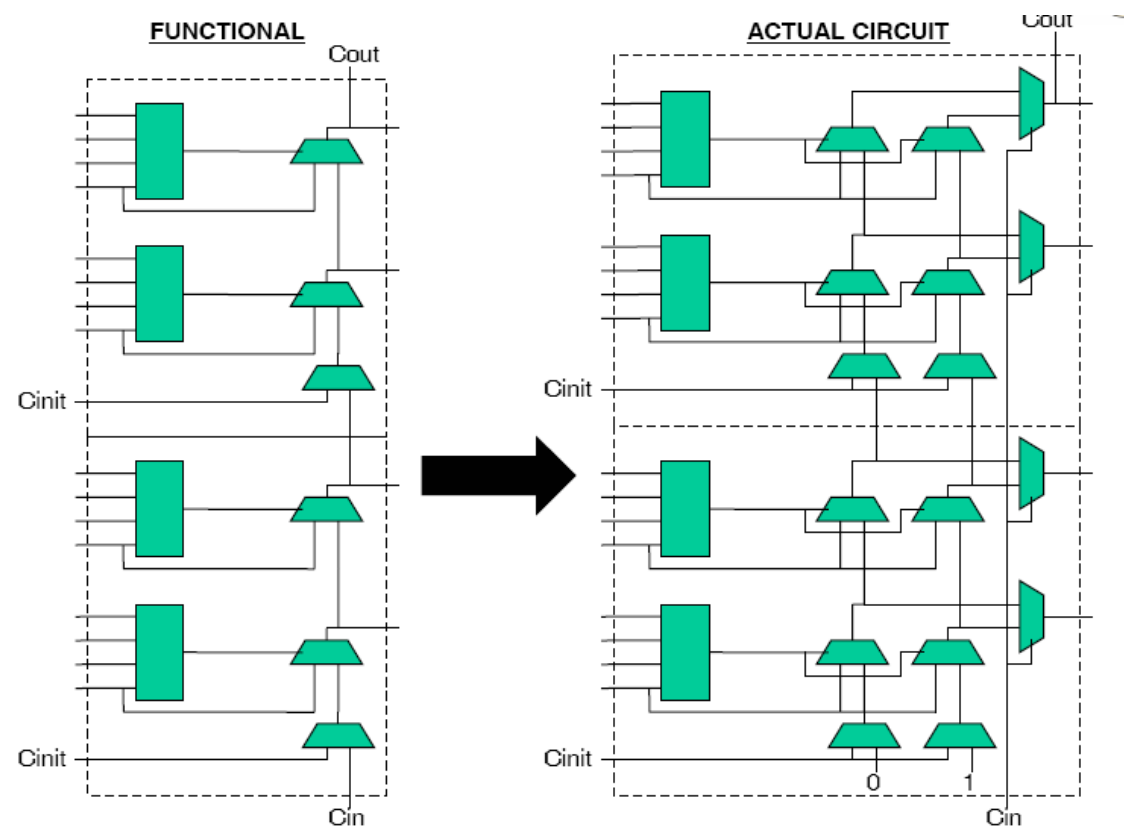
Xilinx VirtexII - CLB

Aula
18

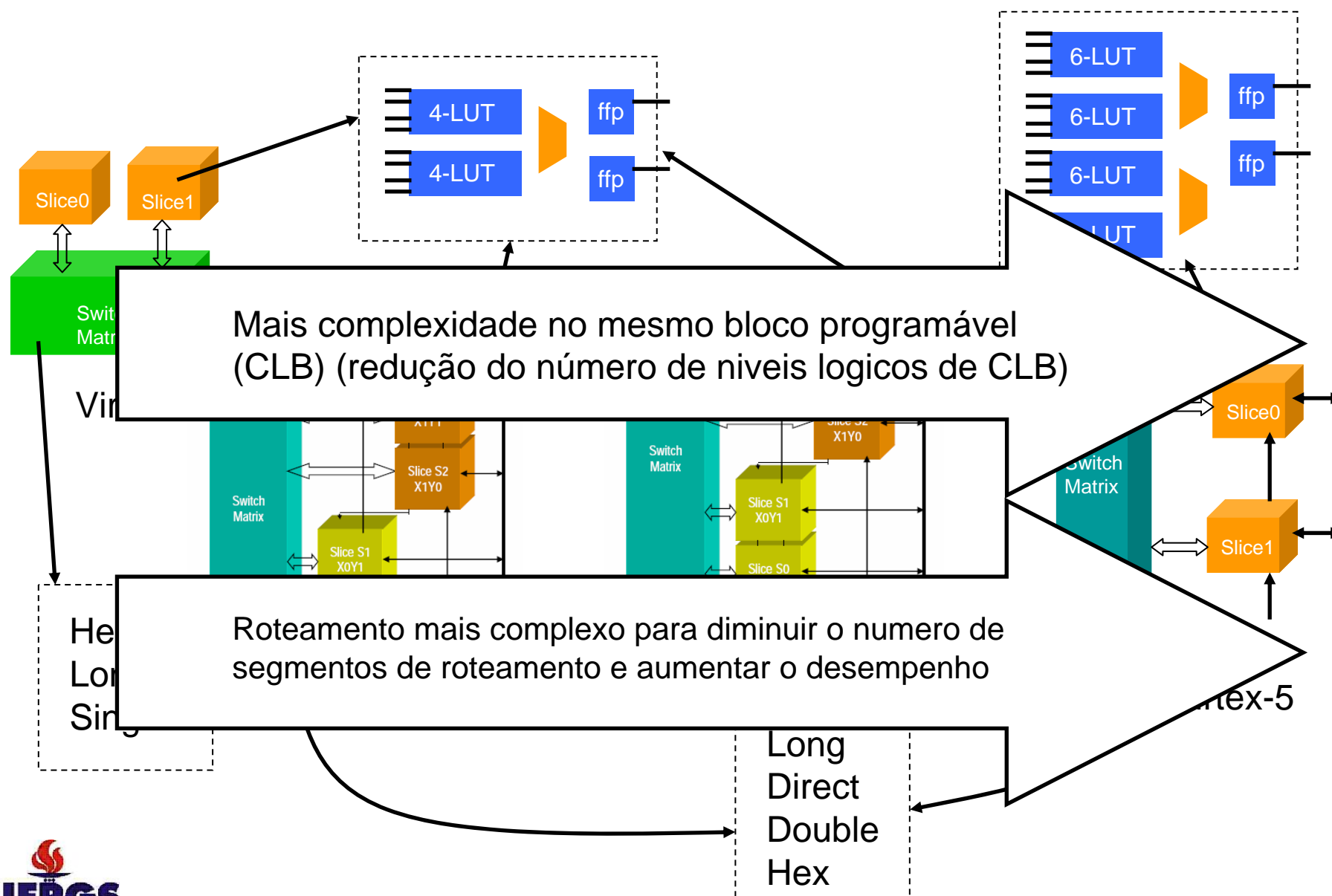


Virtex Carry Select

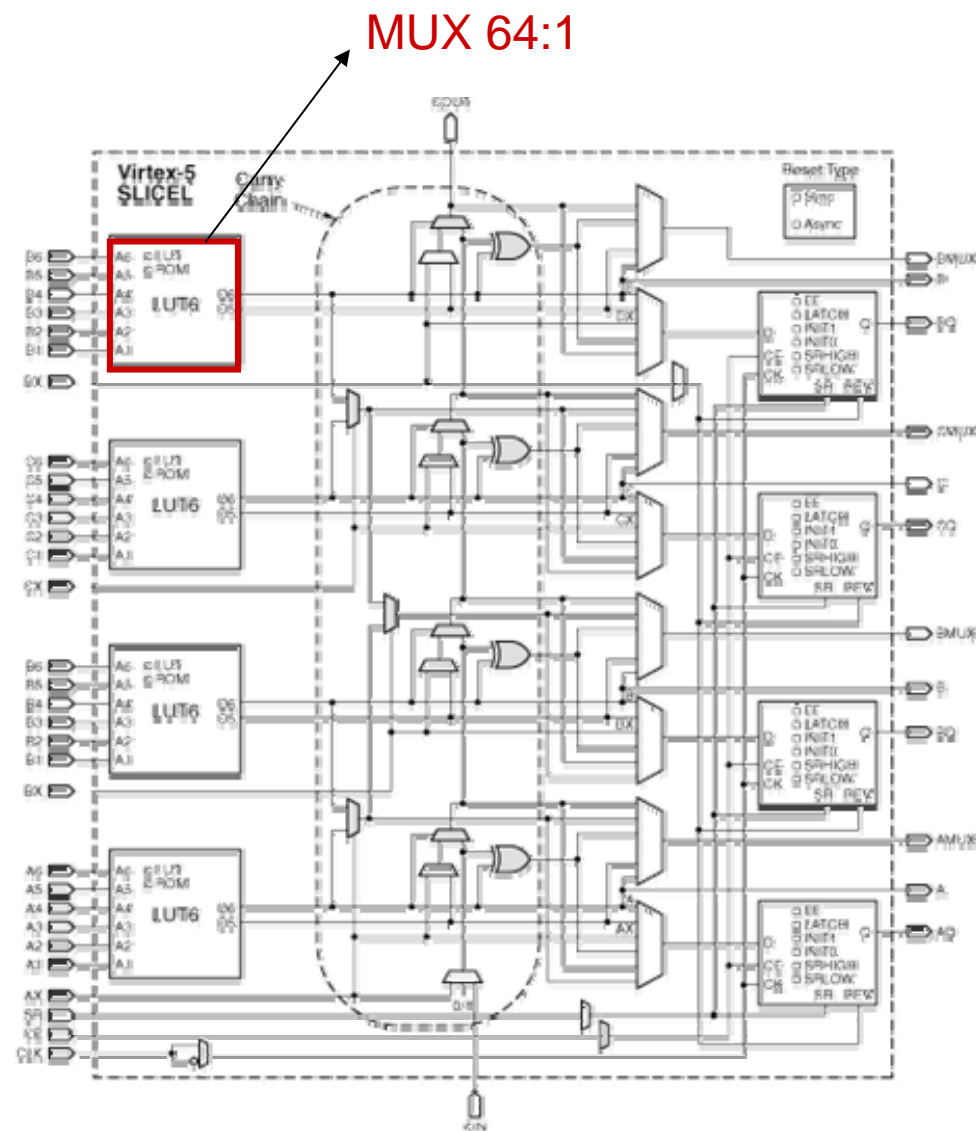
Lógica específica pre-fabricada na matriz para propagação de Carry para acelerar o desempenho de somadores e multiplicadores implementados em FPGA



CLB Evolution

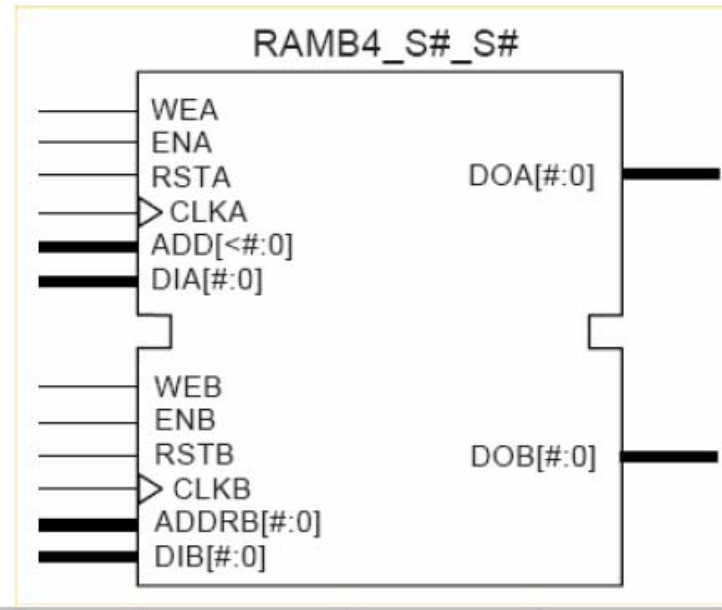


Virtex 5: CLB



BRAM

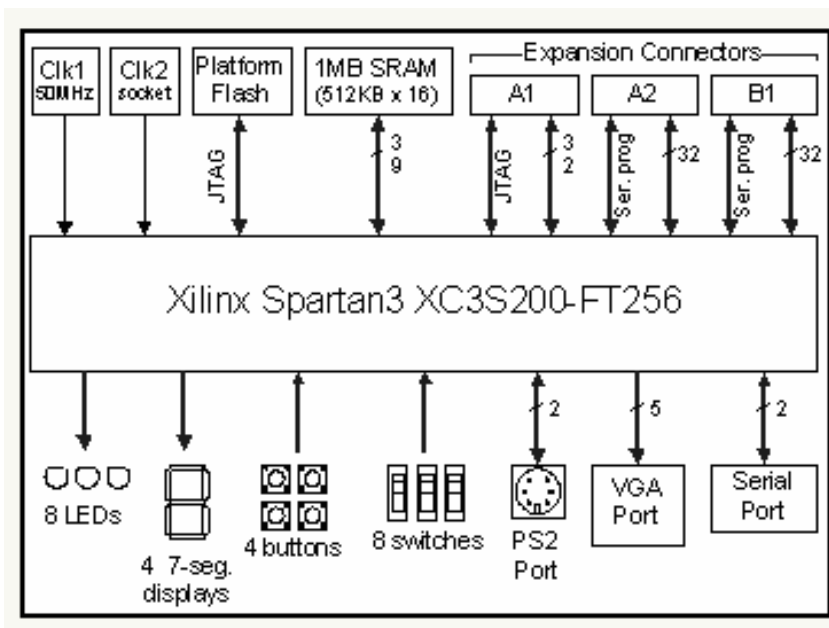
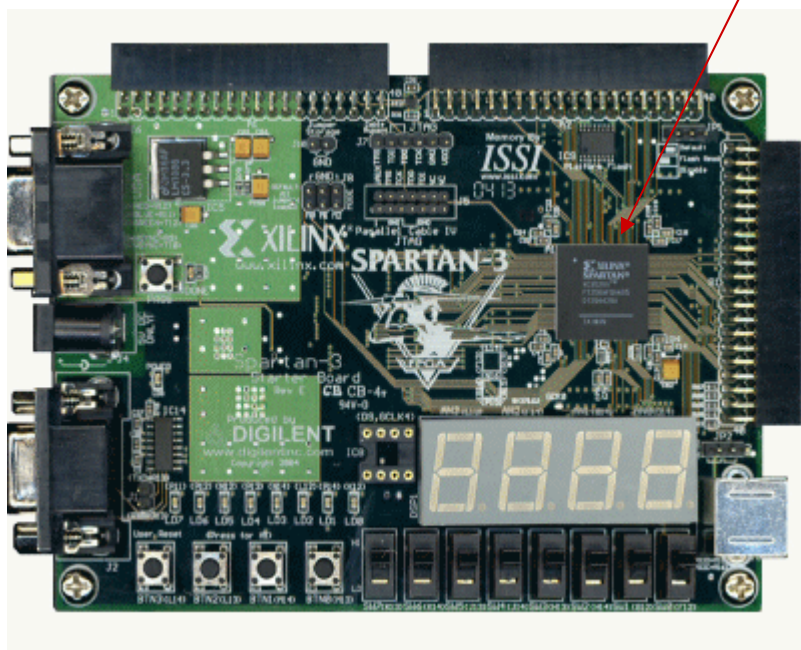
- ◆ 2 Columns of Blocks on left and right
- ◆ 1 Block per 4 CLB rows.
- ◆ 4K bits of data
- ◆ Full Synchronous operation
 - ❖ No Asynchronous Read
- ◆ Ports can be configured to different widths
- ◆ Synchronous reset for Finite State Machine



ADDR	DATA	#/Width	Depth
(11:0)	(0:0)	1	4096
(10:0)	(1:0)	2	2048
(9:0)	(3:0)	4	1024
(8:0)	(7:0)	8	512
(7:0)	(15:0)	16	256

Placa de Prototipação (Exemplo)

FPGA Spartan3



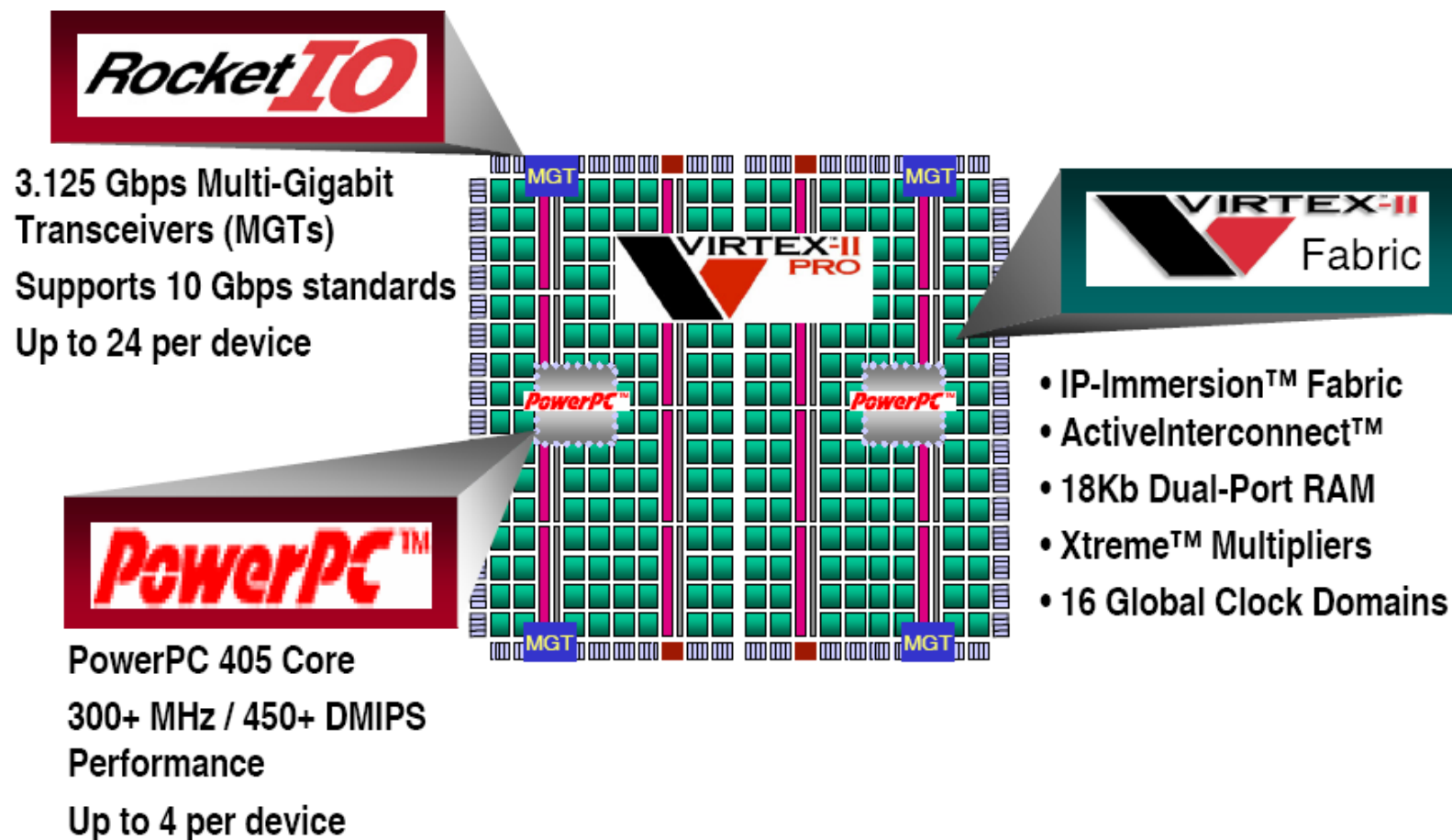
Manual no site: <http://www.digilentinc.com/Products/Detail.cfm?Prod=S3BOARD&Nav1=Products&Nav2=Programmable>

Processadores embarcados em FPGA

- Hard Core:
 - Processador tipo ASIC dentro do FPGA.
- Soft Core:
 - Processador em VHDL/VERILOG a ser sintetizado no FPGA.

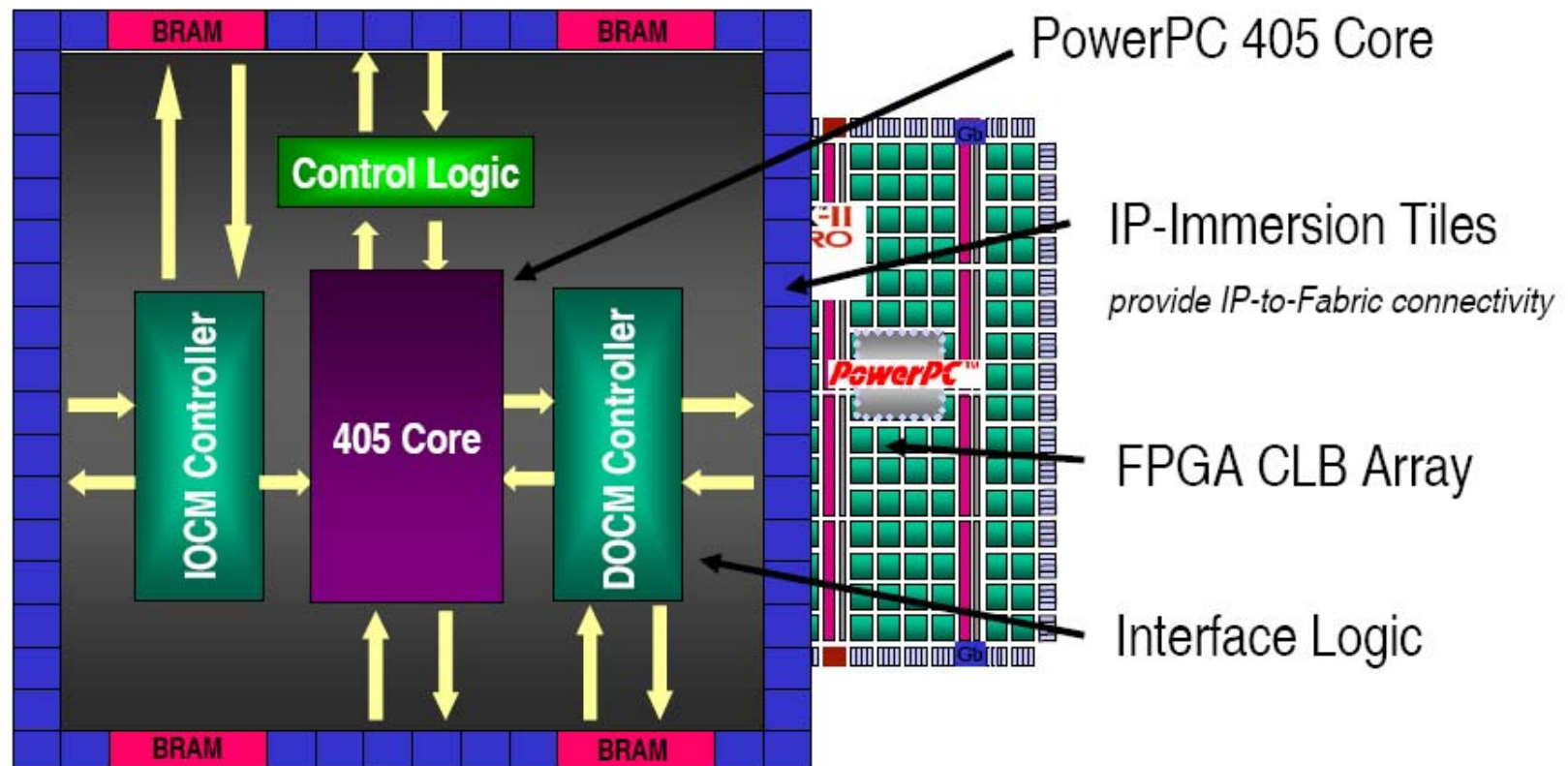
VirtexII-Pro Platform

A família que contém o processador PowerPC embarcado no FPGA



Embedded Processor

Aula
18

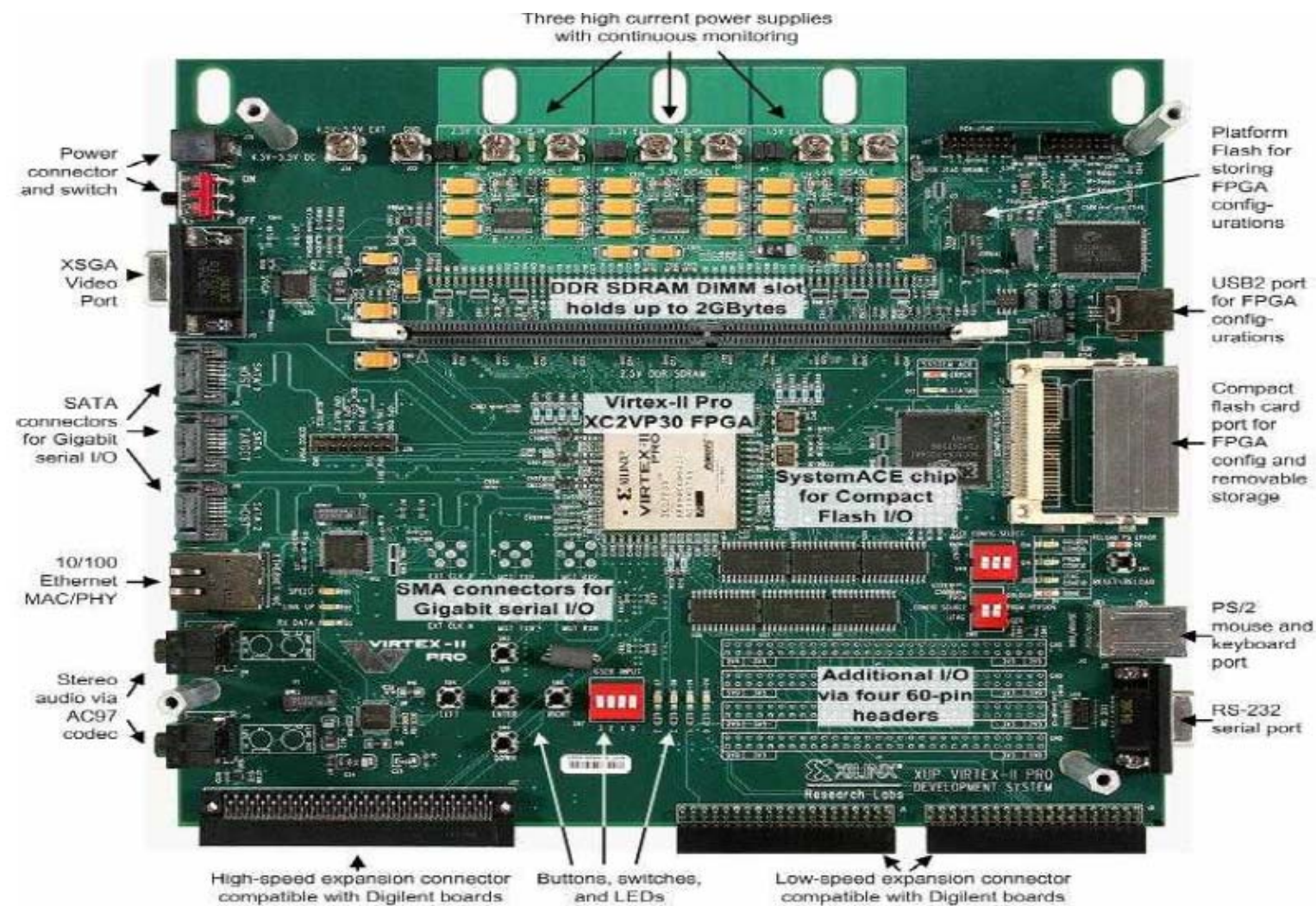


Microblaze (soft core)

Processador descrito em VHDL/Verilog que pode ser implementado no FPGA através da ferramenta de programação.

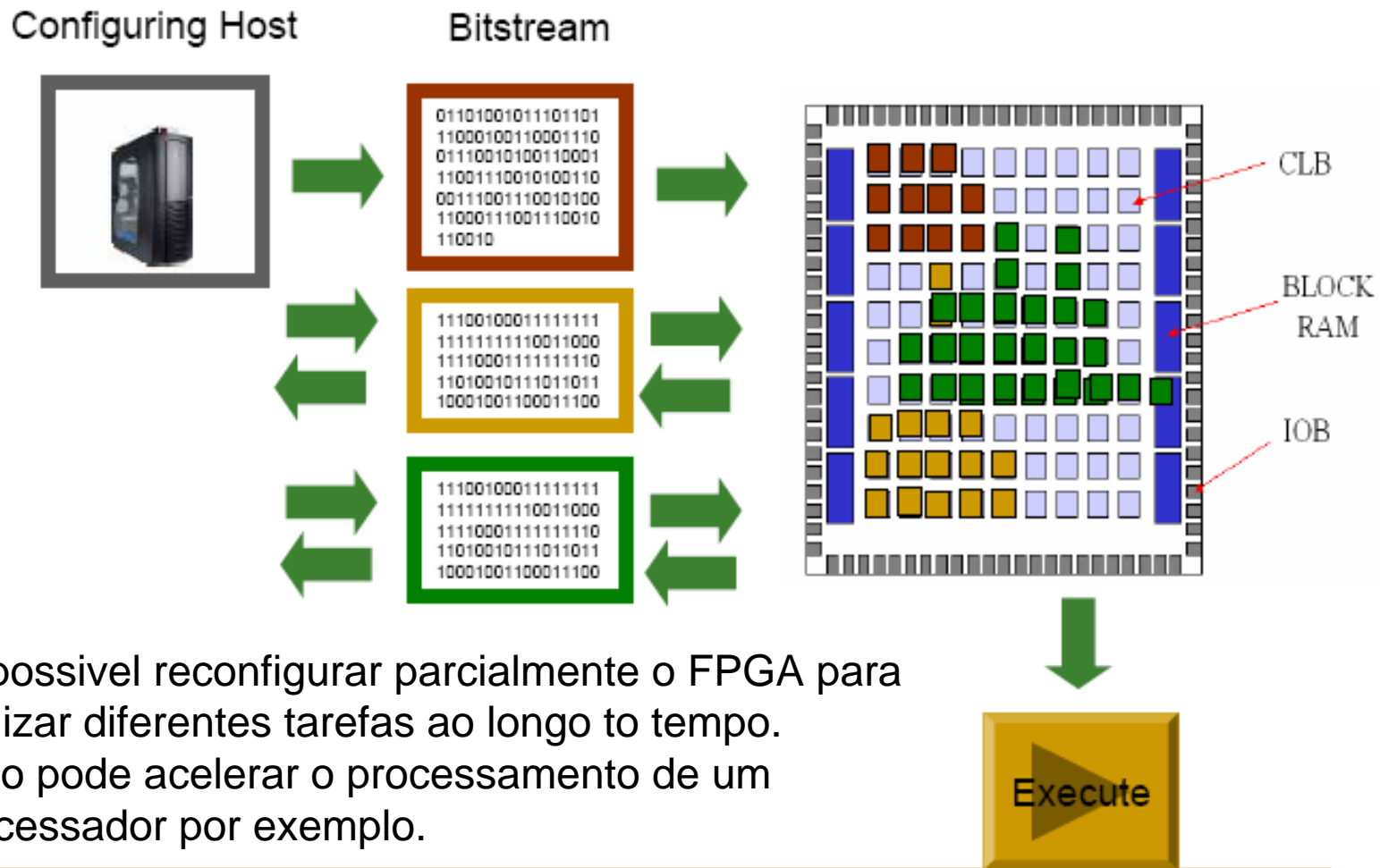
- ◆ RISC
 - ❖ 32-bit ALU, 32-bit data bus, 32-bit instruction word, 32 x 32 General Purpose Register file
- ◆ Harvard architecture (i.e. separate program and data memory space)
- ◆ 3 stage pipeline (IF, OF, EX)
- ◆ Proprietary instruction set has been created for MicroBlaze.

VirtexII-Pro Board (Exemplo)



Reconfigurable Computing

Aula
18

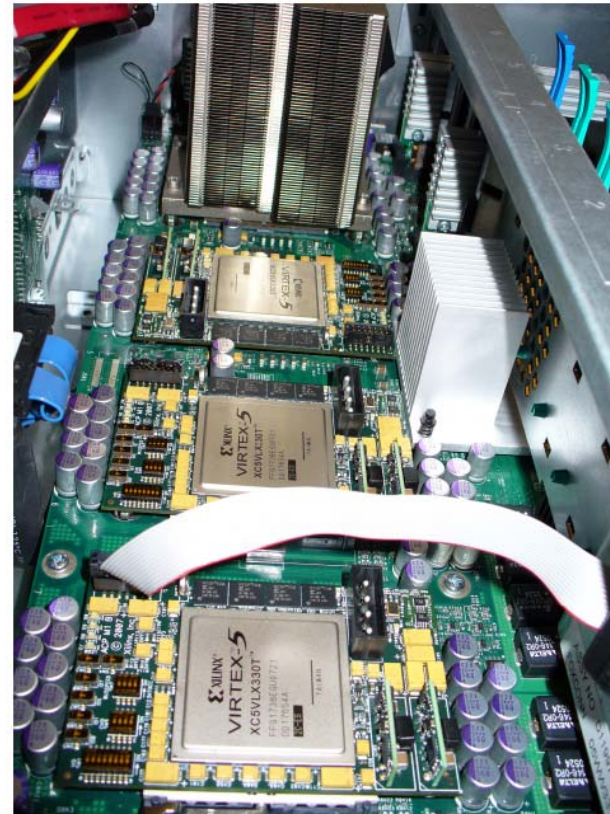


Presented by Trimberger (FPL, 2007)

Exemplo de FPGA (Virtex5) acoplado a uma placa de video para acelerar o processamento de imagem.

FPGA Computing

- Xilinx Virtex-5 FPGA in standard Intel Xeon server platform socket
 - Intel Front-Side Bus (FSB) connection
 - Increased performance in the available power budget
- FSB: An excellent interface for accelerated computing
 - Move to more BW (PCIe x8: 2.0 GB/s, FSB1066: 8.5 GB/s)
 - Much lower latency
 - Coherent system protocol




Technology Solutions	Product & Services	Market Solutions	Support	Online Store	About Xilinx
Silicon Devices	Design Tools	Intellectual Property	Boards & Kits	Training	Services
Third Party					

[Home](#) : [Products & Services](#) : Silicon Devices


Silicon Devices

Featured Products




[Virtex™-5](#) FPGA Family
The Ultimate System Integration Platform

- > Virtex-5 LX Platform - Optimized for high-performance logic
- > Virtex-5 LXT Platform - Optimized for high-performance logic with low-power serial connectivity
- > Virtex-5 SXT Platform - Optimized for DSP and memory-intensive applications with low-power serial connectivity




[Virtex-4](#) FPGA Family
Breakthrough Performance at the Lowest Cost

- > Virtex-4 LX Platform - Optimized for high-performance logic
- > Virtex-4 SX Platform - Optimized for DSP and memory-intensive applications
- > Virtex-4 FX Platform - Optimized for embedded processing and serial connectivity



[Spartan™-3](#) Generation FPGA Families
World's Lowest Cost FPGAs

- > Spartan-3A DSP Platform - DSP optimized
- > Spartan-3AN Platform - Non volatile
- > Spartan-3A Platform - I/O optimized
- > Spartan-3E Platform - Logic optimized
- > Spartan-3 Platform - For highest density and pin-count applications



[CoolRunner™-II](#) CPLD Family
World's Lowest Cost, Lowest Power CPLDs

- > CoolRunner-II - Up to 512 macrocells
- > CoolRunner XPLA3 - Low power, higher voltage applications

FPGA	CPLD
<p>Virtex Series</p> <ul style="list-style-type: none"> > Virtex-5 > Virtex-4 > Virtex-II Pro > Virtex-II > Virtex / E / EM <p>Learn more about the Virtex Series</p> <p>Spartan Series</p>	<p>CoolRunner Series</p> <ul style="list-style-type: none"> > CoolRunner-II > CoolRunner XPLA3 <p>Learn more about the CoolRunner Series</p> <p>XC9500 Series</p> <ul style="list-style-type: none"> > XC9500XL > XC9500xV > XC9500

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

Sources in Project:

- teste_verf
 - xc2vp100-6ff1696
 - arraydl_generic_8bits-a (.ArrayDL_Generic_8bits.vhd)
 - bitadder-part (.bitadder.vhd)

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

HDL Analysis

Analyzing Entity <arraydl_generic_8bits> (Architecture <a>).
Entity <arraydl_generic_8bits> analyzed. Unit <arraydl_generic_8b<

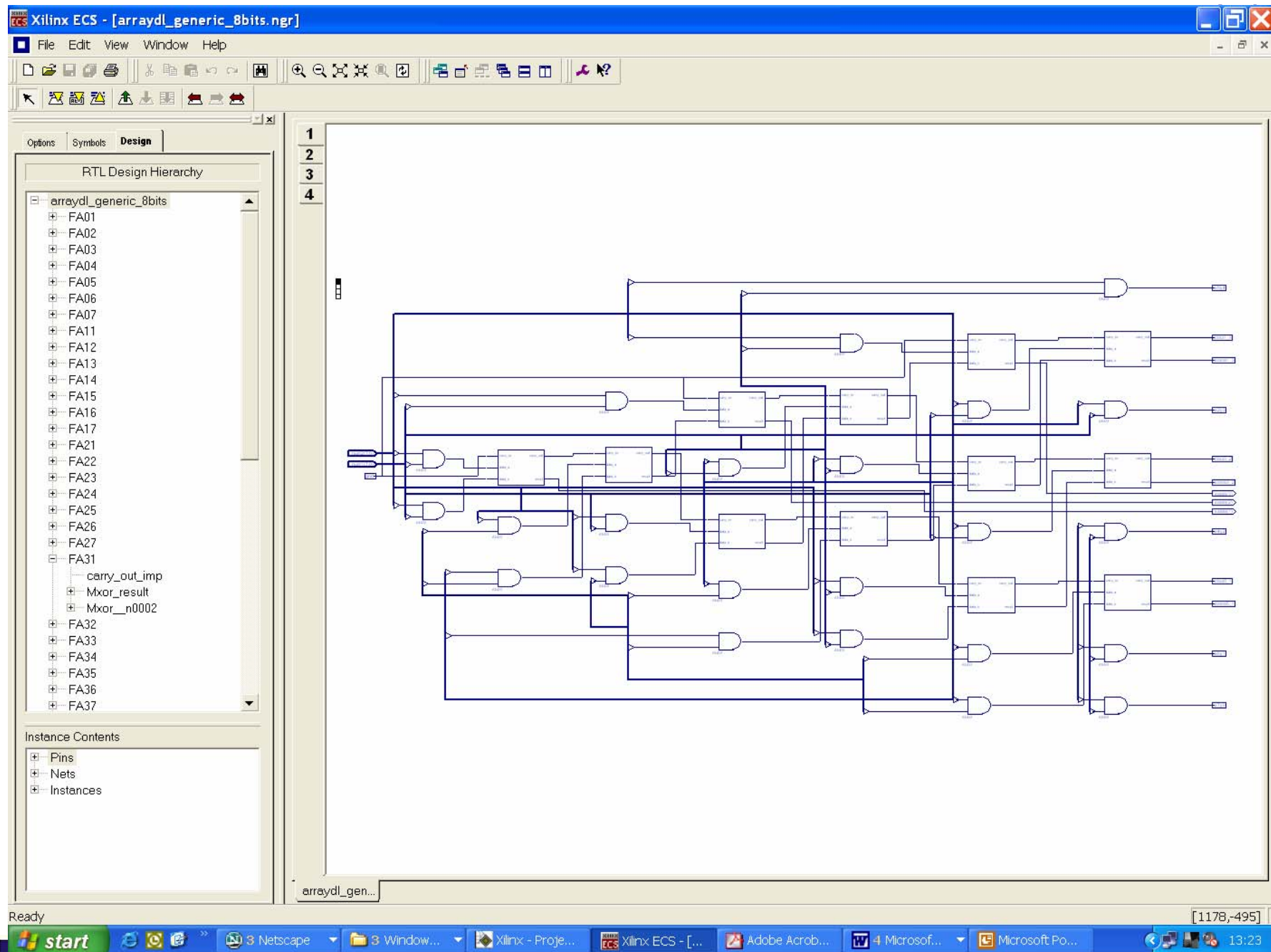
Analyzing Entity <bitadder> (Architecture <part>).
Entity <bitadder> analyzed. Unit <bitadder> generated.

```

1  -----
2  -- Array Multiplier
3  -- Numbers in 8 bits,
4  -- Generated Automatically by the
5  -- Lemon Dragon Multiplier Generator
6  -- by Renato Fernandes Hentschke
7  -- UFRGS - Informatics Institute
8  -- GME - Microelectronics Group
9  -- 13/05/2005 - 16:00
10 -----
11
12
13 library ieee;
14 use ieee.std_logic_1164.all;
15 use ieee.std_logic_arith.all;
16
17
18 entity ArrayDL_Generic_8bits is
19     port(
20         num1: in std_logic_vector(7 downto 0);
21         num2: in std_logic_vector(7 downto 0);
22         saida : out std_logic_vector(15 downto 0)
23     );
24 end ArrayDL_Generic_8bits;
25
26
27 architecture a of ArrayDL_Generic_8bits is
28
29     component bitadder
30     port(
31         data_a      : IN  STD_LOGIC;
32         data_b      : IN  STD_LOGIC;
33         carry_in    : IN  STD_LOGIC;
34         result      : OUT STD_LOGIC;
35         carry_out   : OUT STD_LOGIC
36     );
37     end component;
38
39 -- esses sao os sinais internos, nao precisa colocar
40 signal x0y0: std_logic;
41 signal x0y1: std_logic;
42 signal x0y2: std_logic;
43 signal x0y3: std_logic;
44 signal x0y4: std_logic;
45 signal x0y5: std_logic;
46 signal x0y6: std_logic;
47 signal x0y7: std_logic;
48 signal x1y0: std_logic;
49 signal x1y1: std_logic;

```

Ln 18 Col 1



Noções de VHDL

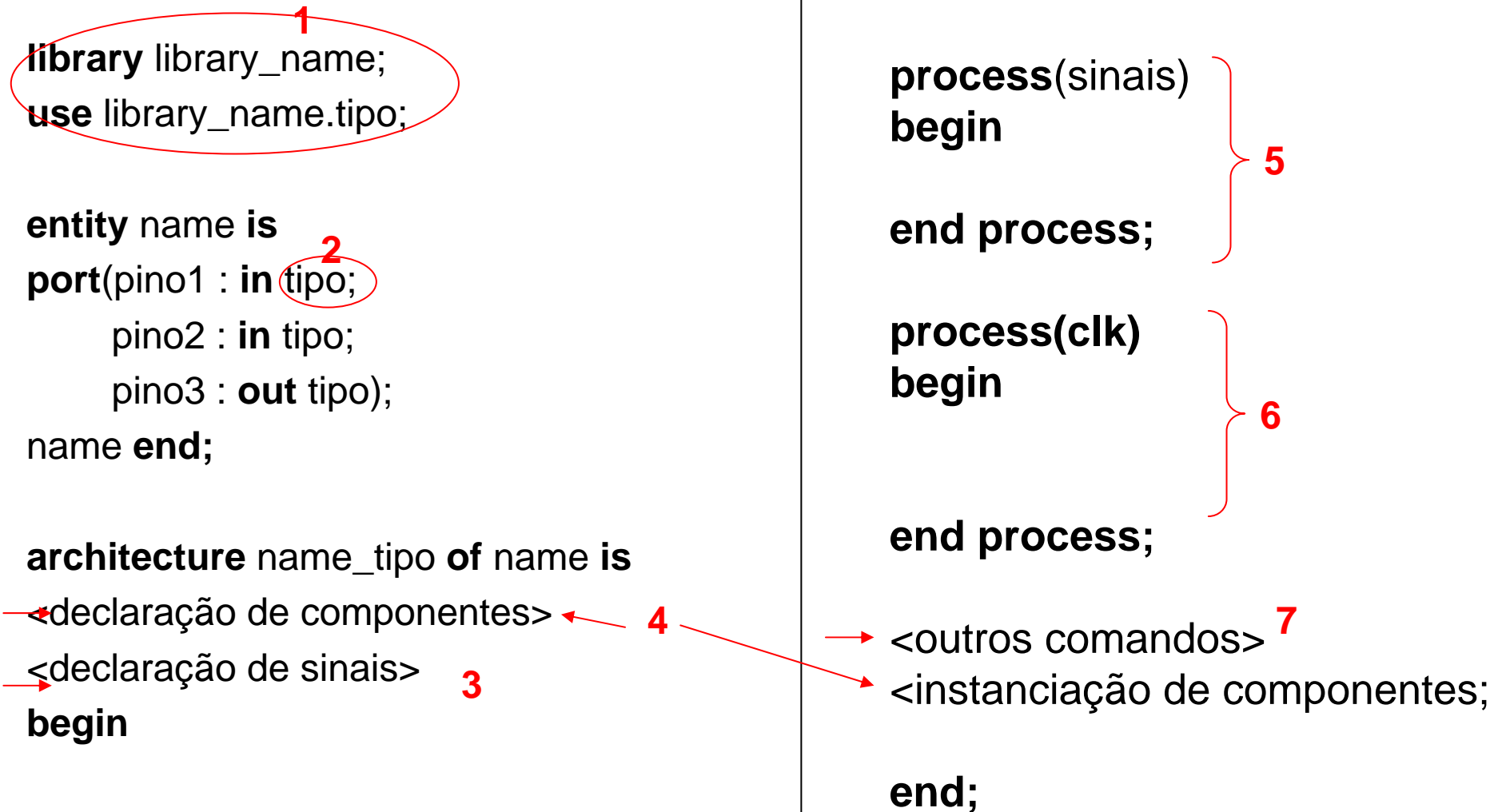
- VHDL foi desenvolvido primeiramente pelo Departamento de Defesa Americano: the American Department of Defense (DoD).
- Em 1987, VHDL foi padronizado pelo Instituto Americano de Engenharia Eletro e Eletrônica: American Institute of Electrical and Electronics Engineers (IEEE) em 1993.

International Standards

- **IEEE Std 1076-1987**
- **IEEE Std 1076-1993**

- Hardware Description Language (HDL) = "Programming"-language para modelar hardware digital
- **VHDL = VHSIC Hardware Description Language**
- **VHSIC = Very High Speed Integrated Circuit Hardware description, simulation, and synthesis**

Estrutura do VHDL



1 Bibliotecas

```
LIBRARY ieee;  
USE ieee.std_logic_1164.ALL;  
USE ieee.std_logic_unsigned.all;  
USE ieee.numeric_std.ALL;
```

- São necessárias para usar operadores de soma e subtração e usar o tipo std_logic e std_logic_vector para definir pinos e sinais internos.

Estrutura do VHDL

library library_name;
use library_name.tipo;

entity name **is**
port(pino1 : **in** tipo;
pino2 : **in** tipo;
pino3 : **out** tipo);
name **end**;

architecture name_tipo **of** name **is**
— <declaração de componentes>
— <declaração de sinais>
begin

process(sinais)
begin

end process;

process(clk)
begin

end process;

<outros comandos>
<instanciação de componentes>

end;

Tipo Binário com múltiplos valores: STD_LOGIC e STD_LOGIC_VECTOR

Aula

18

- IEEE-standard
- 9 valores padronizados pela IEEE
- standard IEEE 1164 (STD_LOGIC_1164)

IEEE Standard Logic Type

```
type STD_ULOGIC is (  
    `U`, -- uninitialized  
    `X`, -- strong 0 or 1 (= unknown)  
    `0`, -- strong 0  
    `1`, -- strong 1  
    `Z`, -- high impedance  
    `W`, -- weak 0 or 1 (= unknown)  
    `L`, -- weak 0  
    `H`, -- weak 1  
    `-`, -- don't care);
```

Exemplo até agora...

```
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;  
use IEEE.STD_LOGIC_ARITH.ALL;  
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

Declara as bibliotecas

```
entity memoria_teste is  
    Port ( clk : in  STD_LOGIC;  
          reset : in  STD_LOGIC;  
          pronto : out STD_LOGIC;  
          dado : out  STD_LOGIC_VECTOR (7 downto 0));  
end memoria_teste;
```

Declara a entidade que é a interface do teu circuito com os pinos de entrada (in) e saída (out) e tipo std_logic (1 bit) ou std_logic_vector (vetor de n bits)

Estrutura do VHDL

library library_name;
use library_name.tipo;

entity name **is**
port(pino1 : **in** tipo;
pino2 : **in** tipo;
pino3 : **out** tipo);
name **end**;

architecture name_tipo **of** name **is**
— <declaração de componentes>
— <declaração de sinais>
begin

process(sinais)
begin

end process;

process(clk)
begin

end process;

<outros comandos>
<instanciação de componentes>

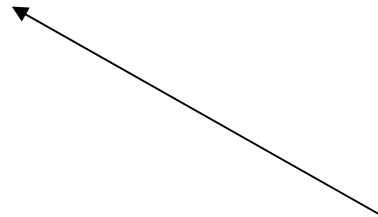
end;

3 Declaração de sinais

architecture Behavioral of memoria_teste is

signal cont, dadomem_in: STD_LOGIC_VECTOR (7 downto 0);
signal leitura, escrita, WR: std_logic;

begin



Entre a declaração da arquitetura e o begin (início do circuito), podes declarar os sinais internos do circuito e o tipo deles.

Estrutura do VHDL

library library_name;
use library_name.tipo;

entity name **is**
port(pino1 : **in** tipo;
pino2 : **in** tipo;
pino3 : **out** tipo);
name **end**;

architecture name_tipo **of** name **is**
— <declaração de componentes>
— <declaração de sinais>
begin

process(sinais)
begin

end process;

process(clk)
begin

end process;

<outros comandos>
<instanciação de componentes>

end;

Usando componentes prontos

- Temos que declarar o componente e depois instancia-lo.
- O componente é uma outra entidade e arquitetura definido em outro arquivo que ja foi compilado e está no mesmo diretório de trabalho.

Declaração de componente

```
entity FULLADDER is
  port (A,B, CARRY_IN: in  std_logic;
        SUM, CARRY:  out std_logic);
end FULLADDER;

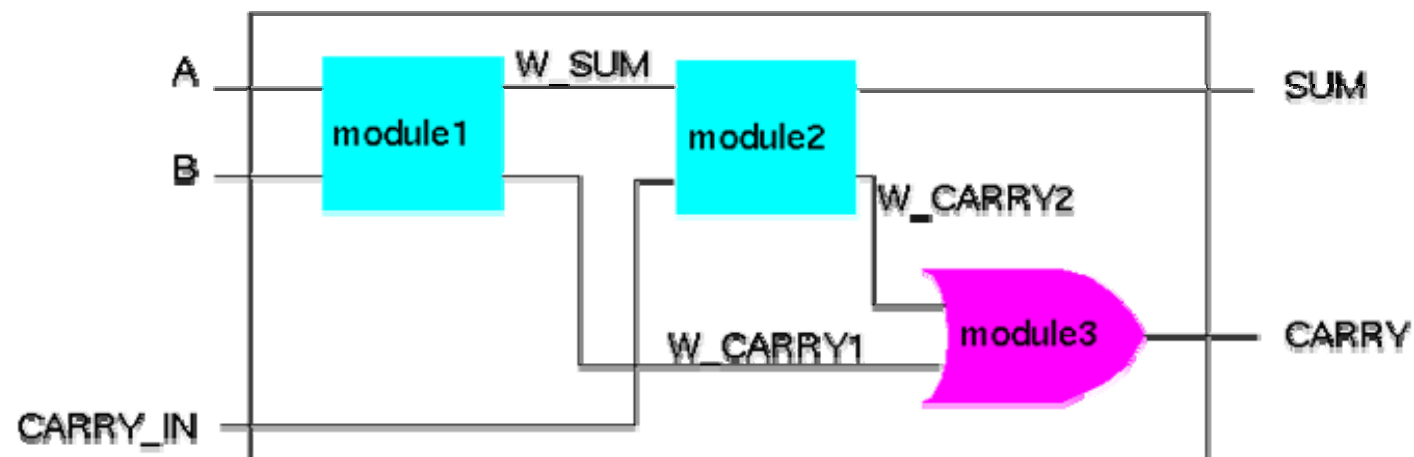
architecture STRUCT of FULLADDER is
  signal W_SUM, W_CARRY1, W_CARRY2 : std_logic;

  component HALFADDER
    port (A, B :          in  std_logic;
          SUM, CARRY : out std_logic);
  end component;

  component ORGATE
    port (A, B : in  std_logic;
          RES : out std_logic);
  end component;

  begin
    . . .
```

Modelo Hierarquico



Full adder: 2 halfadders + 1 OR-gate

Instanciação de Componente

Assinalamento dos sinais (ports) pela ordem dos pinos na interface do componente

```
architecture STRUCT of FULLADDER is
  component HALFADDER
    port (A, B :          in  std_logic;
          SUM, CARRY : out std_logic);
  end component;
  component ORGATE
    port (A, B : in  std_logic;
          RES : out std_logic);
  end component;

  signal W_SUM, W_CARRY1, W_CARRY2: std_logic;

begin

  MODULE1: HALFADDER
    port map( A, B, W_SUM, W_CARRY1 );

  MODULE2: HALFADDER
    port map ( W_SUM, CARRY_IN,
                SUM, W_CARRY2 );

  MODULE3: ORGATE
    port map ( W_CARRY2, W_CARRY1, CARRY );

end STRUCT;
```

Instanciação de Componente: associação de nomes dos ports

Aula
18

```
entity FULLADDER is
  port (A,B, CARRY_IN: in  std_logic;
        SUM, CARRY:  out std_logic);
end FULLADDER;

architecture STRUCT of FULLADDER is

  component HALFADDER
    port (A, B :          in std_logic;
          SUM, CARRY : out std_logic);
  end component;

  ...
  signal W_SUM, W_CARRY1, W_CARRY2 : std_logic;

begin

  MODULE1: HALFADDER
    port map ( A      => A,
               SUM    => W_SUM,
               B      => B,
               CARRY  => W_CARRY1 );

  ...
end STRUCT;
```

Assinalamento explícito

Estrutura do VHDL

library library_name;
use library_name.tipo;

entity name **is**
port(pino1 : **in** tipo;
pino2 : **in** tipo;
pino3 : **out** tipo);
name **end**;

architecture name_tipo **of** name **is**
— <declaração de componentes>
— <declaração de sinais>
begin

process(sinais)
begin

end process;

process(clk)
begin

end process;

<outros comandos>
<instanciação de componentes>

end;

5 Comandos Process

- Todos os comandos após o BEGIN são vistos como comandos concorrentes entre si.
- Mas dentro de um process, os comandos são executados de maneira sequencial.
- Quando todas as entradas do bloco definido no process estão na lista de sensibilidade deste process, então estamos projetando um circuito combinacional.

```
process (reg1,reg2,reg3,reg4,flagj1,flagj2,flagj3,flagj4)
begin
  if (flagj1 = '1' and flagj2 = '1' and flagj3 = '1' and flagj4 = '1') then
    if (reg1>reg2 and reg1>reg3 and reg1>reg4) then
      comp<="100";
    elsif (reg2>reg1 and reg2>reg3 and reg2>reg4) then
      comp<="101";
    elsif (reg3>reg1 and reg3>reg2 and reg3>reg4) then
      comp<="110";
    elsif (reg4>reg1 and reg4>reg3 and reg4>reg2) then
      comp<="111";
    end if;
  else
    comp <= "000";
  end if;
end process;
```

Estrutura do VHDL

library library_name;
use library_name.tipo;

entity name **is**
port(pino1 : **in** tipo;
pino2 : **in** tipo;
pino3 : **out** tipo);
name **end**;

architecture name_tipo **of** name **is**
— <declaração de componentes>
— <declaração de sinais>
begin

process(sinais)
begin

end process;

process(clk)
begin

end process;

<outros comandos>
<instanciação de componentes>

end;

6 Comandos Process

- Todos os comandos após o BEGIN são vistos como comandos concorrentes entre si.
- Mas dentro de um process, os comandos são executados de maneira sequencial.
- Para sintetizar circuitos sequencias (flip-flops) é preciso fazer com que o process seja sensível a um relógio (clk).

```
process (clk,reset)
begin
  if (reset = '1') then
    reg1 <= "0000000000000000";
    flagj1<='0';
  elsif (clk'event and clk='1') then
    if j1='1' and flagj1='0' then
      reg1 <= LFSR;
      flagj1 <= '1';
    end if;
  end if;
end if;
end process;
```

Estrutura do VHDL

library library_name;
use library_name.tipo;

entity name **is**
port(pino1 : **in** tipo;
pino2 : **in** tipo;
pino3 : **out** tipo);
name **end**;

architecture name_tipo **of** name **is**
— <declaração de componentes>
— <declaração de sinais>
begin

process(sinais)
begin

end process;

process(clk)
begin

end process;

<outros comandos>
<instanciação de componentes>

end;

7 Outros comandos concorrentes

- Podemos usar operadores diretos, fora de process.
- Esses comandos serão executados de maneira concorrente entre eles e com os demais process.

$\text{sum} \leq \text{entA} + \text{ent B};$

$F \leq A \text{ xor } B \text{ xor } \text{Cin};$

$\text{Cout} \leq (A \text{ and } B) \text{ or } (A \text{ and } \text{Cin}) \text{ or } (B \text{ and } \text{Cin});$