Universidade Federal de Santa Catarina
Centro Tecnológico – CTC
Departamento de Engenharia Elétrica

"EEL7020 – Sistemas Digitais"

Prof. Eduardo Augusto Bezerra

Eduardo.Bezerra@eel.ufsc.br

Florianópolis, março de 2010.

Sistemas Digitais

“FPGAs: estrutura interna, aplicações e limitações”

Plano de Aula

"FPGAs: estrutura interna, aplicações e limitações"

- Objetivos:
  - Conhecer a estrutura interna de FPGAs
    - Blocos lógicos, entrada/saída, interconexão
    - FPGAs Xilinx
  - Identificar aplicações de FPGAs
  - Desenvolver estudo de caso
  - Descrever e avaliar limitações de FPGAs

Motivação

Problema: implementar uma função lógica

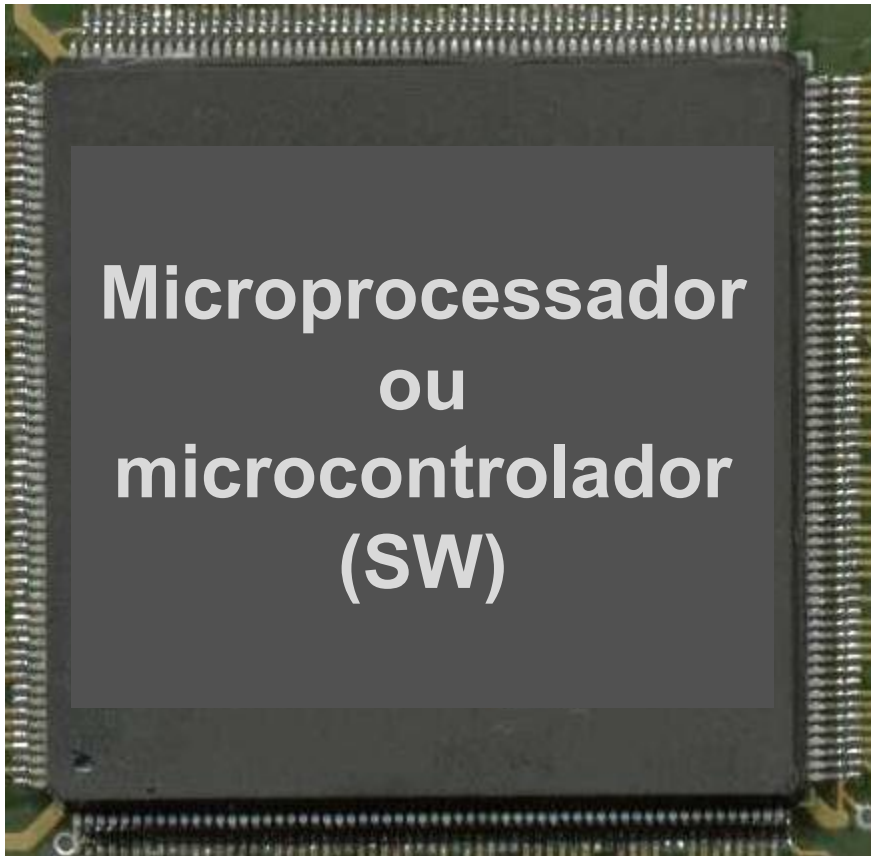
f(A, B) = A and B , onde A e B possuem 4 bits cada, gerando uma saída também de 4 bits

- **Solução 1: dispositivo TTL 7408**
  - Nova função significa alteração no hardware (placa)

- **Solução 2: microprocessador (SW)**

```
void main(){
  int a, b, f;
  f = a & b;
}
```

  - Flexibilidade para novas funções
  - Necessidade de memória para armazenar programa
  - "Adaptações" para obter variáveis de 4 bits



Problema: implementar uma função lógica

f(A, B) = A and B , **onde A e B possuem 4 bits cada, gerando uma saída também de 4 bits**

- **Solução intermediária: FPGA**
  - **Possibilita implementação de novas funções em hardware, sem alteração na placa**

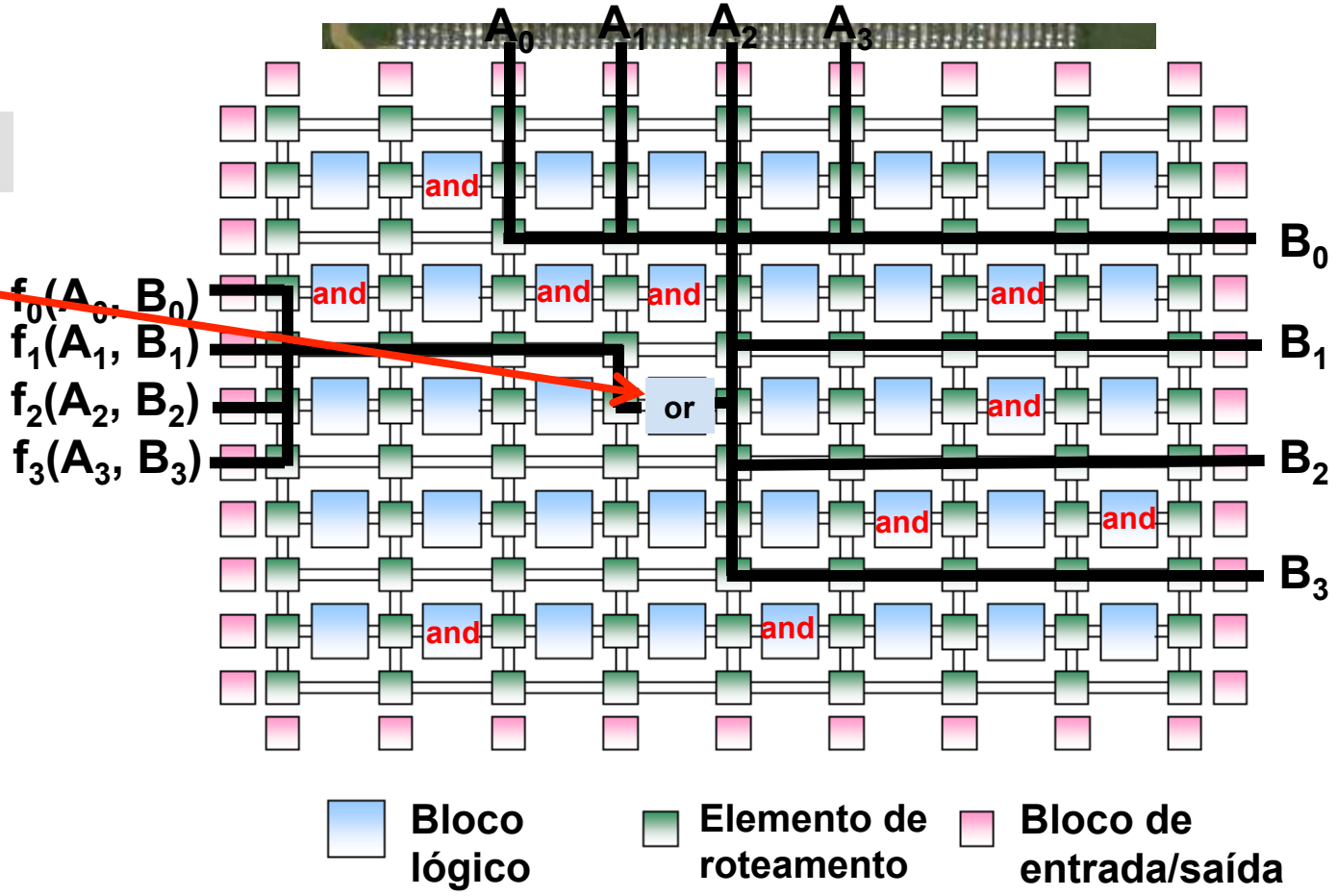
FPGAs: estrutura interna

Componentes básicos de dispositivos FPGA

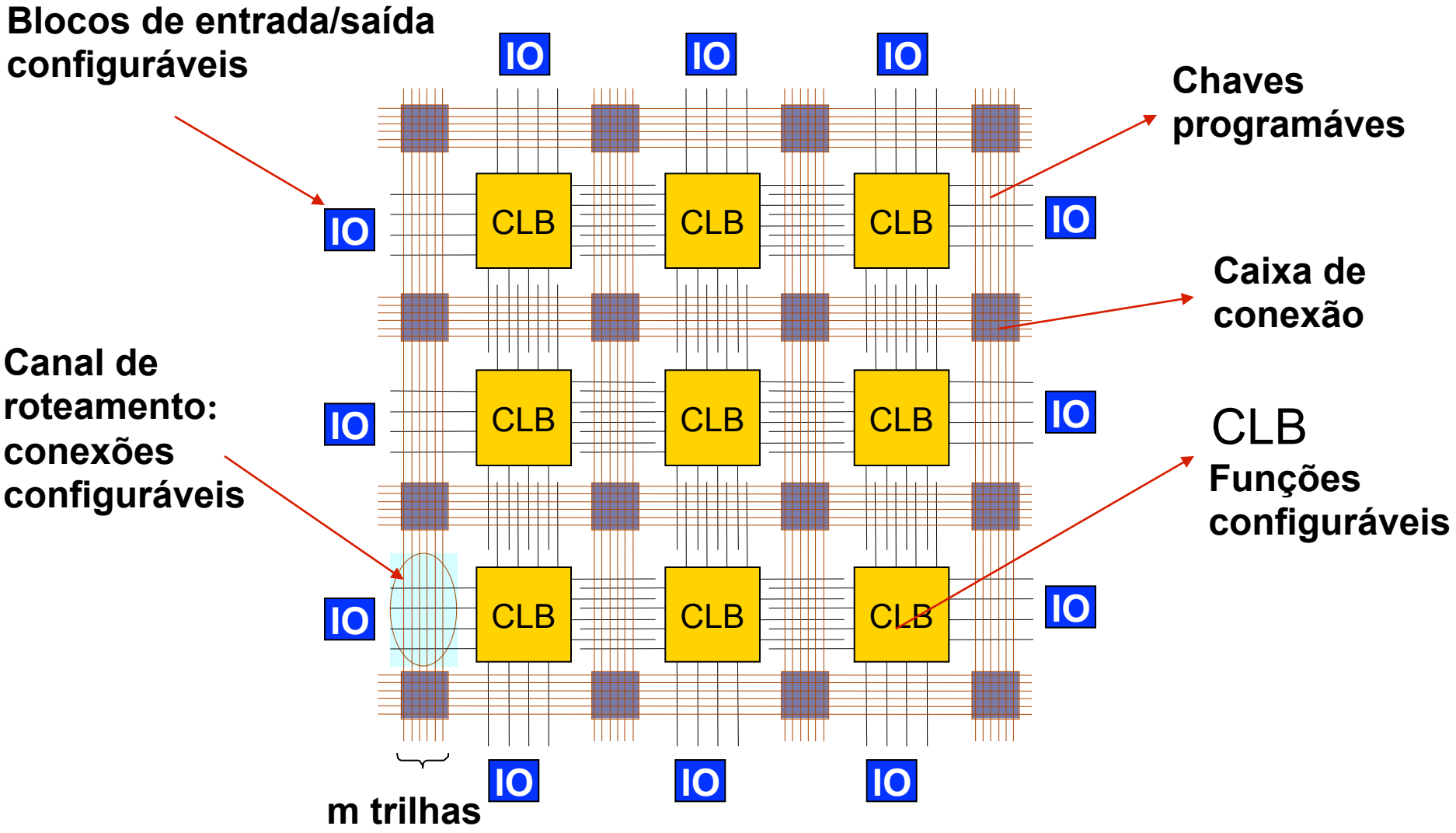
Objetivo: prover recursos de hardware para implementação de funções lógicas

f(A, B) = A and B

f(A, B) = A or B

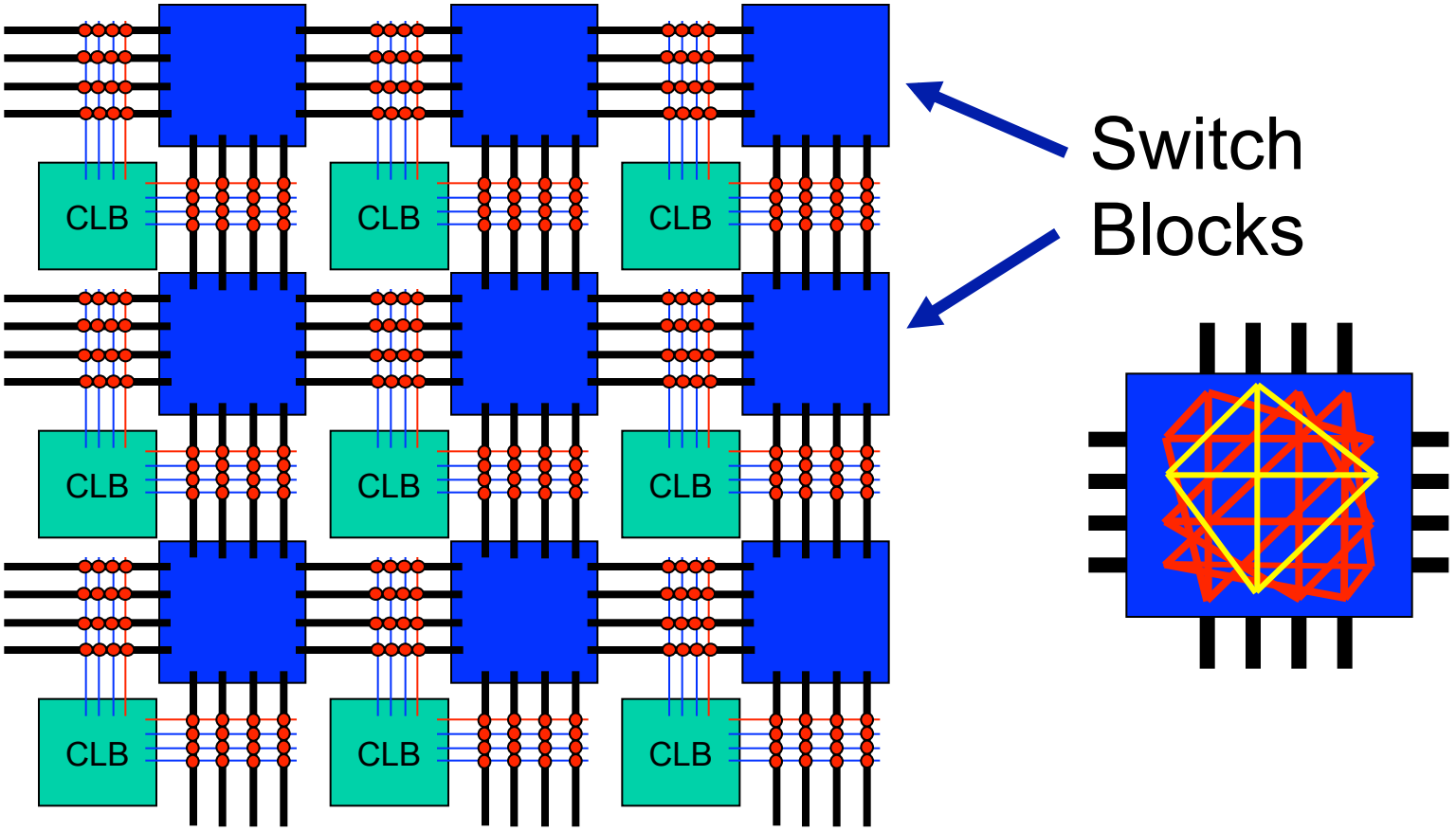


Estrutura interna



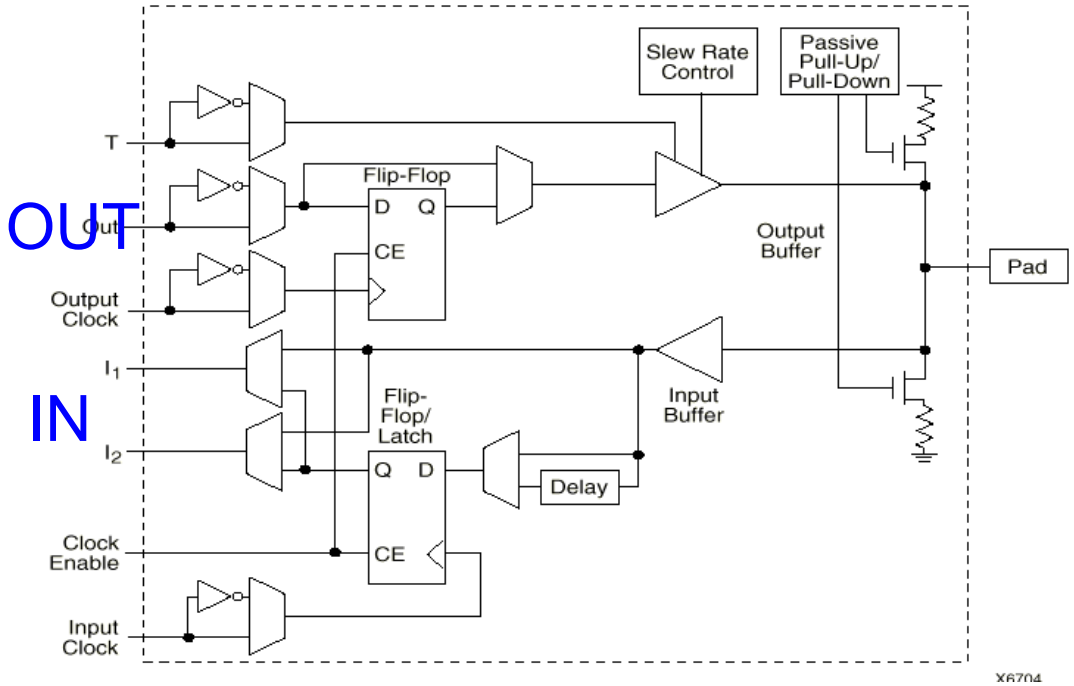
Estrutura interna

- **Matriz de CLBs (*Configurable Logic Blocks*) interconectados por uma matriz de chaveamento**



Estrutura interna – blocos de entrada/saída

- Possuem recursos de memória configuráveis (registrador ou latch)
- Pads configuráveis como entrada, saída ou bidirecionais
- Entradas podem utilizar registrador ou latch
- Possibilidade de definir sinais three-state (alta impedância)



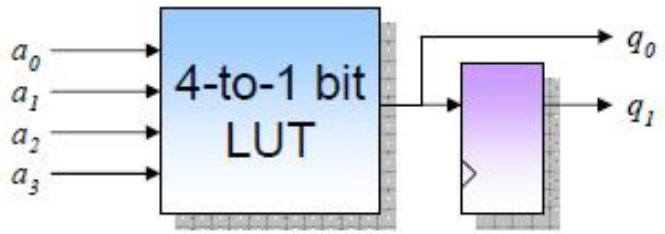
Estrutura interna – funções lógicas

- **LUT - Look-Up Table**
  - **Gerador universal de funções**
  - **Módulo configurável capaz de implementar qualquer tabela verdade de *n* entradas**
  - **LUT**
    - » **Altamente flexível**
    - » **Método mais utilizado (Xilinx e Altera)**

Estrutura interna – funções lógicas

Exemplo de função lógica com LUT

$$q = (a_0 \; AND \; a_1) \; OR \; (a_2 \; AND \; a_3)$$

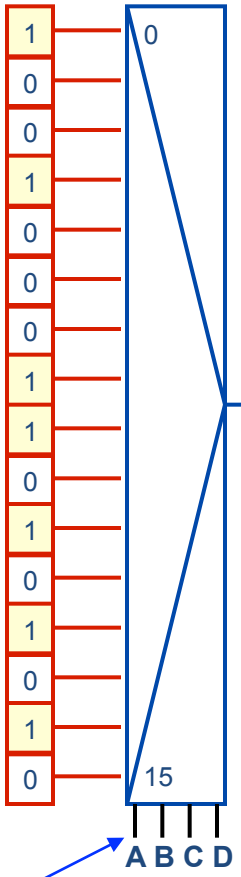


a_0	a_1	a_2	a_3	q
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Estrutura interna – funções lógicas

Implementação física de uma LUT4

Tabela verdade da função é armazenada em memória durante configuração do FPGA



$$F(A, B, C, D) = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}CD + A\bar{D}$$

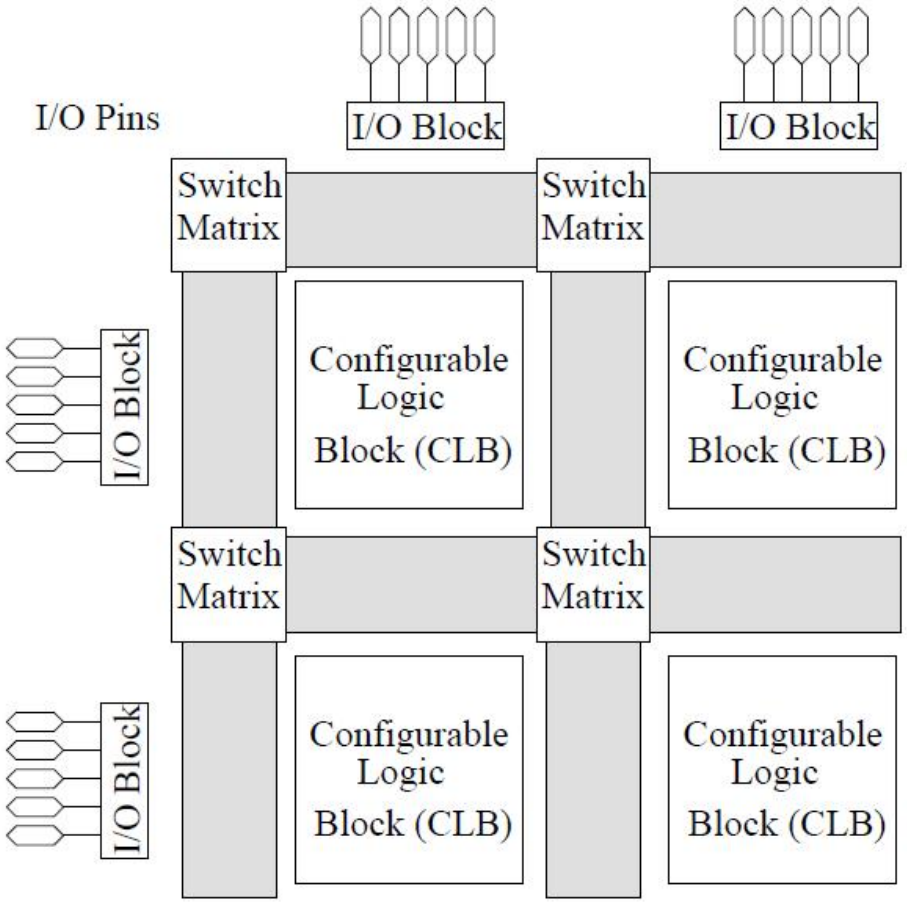
$$F(A, B, C, D) = \sum (0, 3, 7, 8, 10, 12, 14)$$

As entradas (variáveis booleanas) controlam um multiplexador 2^n:1

Considerando 150 transistores / LUT
Para 50.000 LUTS ➔ 7.500.000 transistores!

Estrutura interna FPGAs da Xilinx

Xilinx



FPGAs da Xilinx

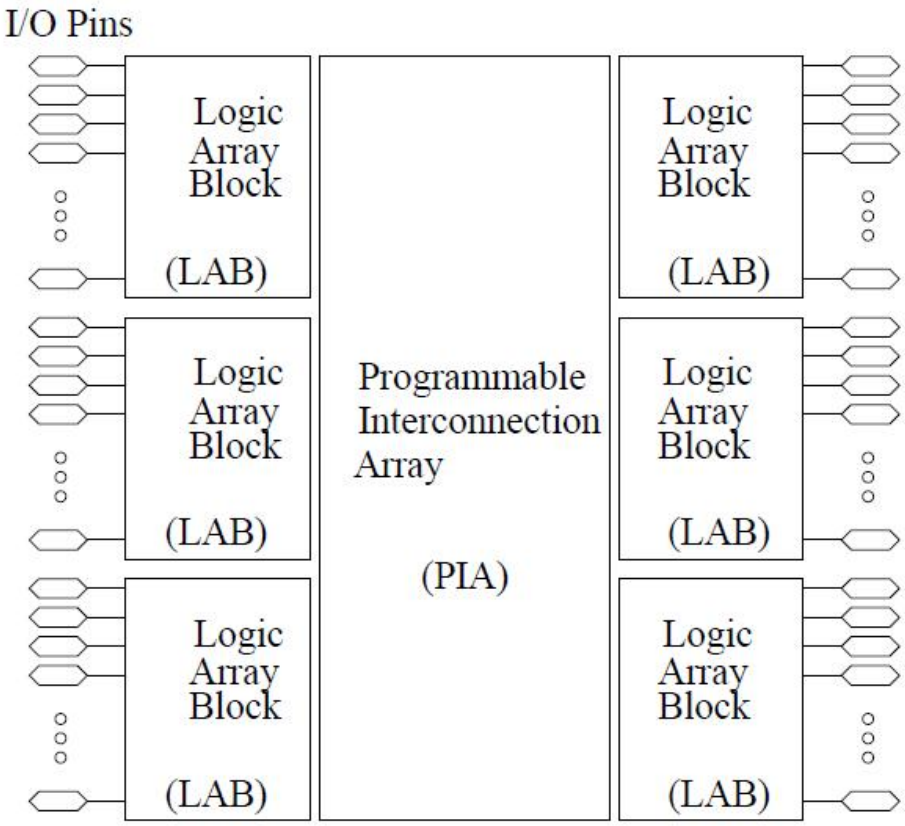
- **Arquitetura do tipo "ilha"**
- **Funções lógicas são divididas em pequenas ilhas**
- **Cada ilha com funções de 4 a 6 termos**
- **As ilhas são conectadas por intermédio de elementos de interconexão programáveis (matriz de chaveamento)**

Estrutura interna FPGAs da Altera

FPGAs da Altera

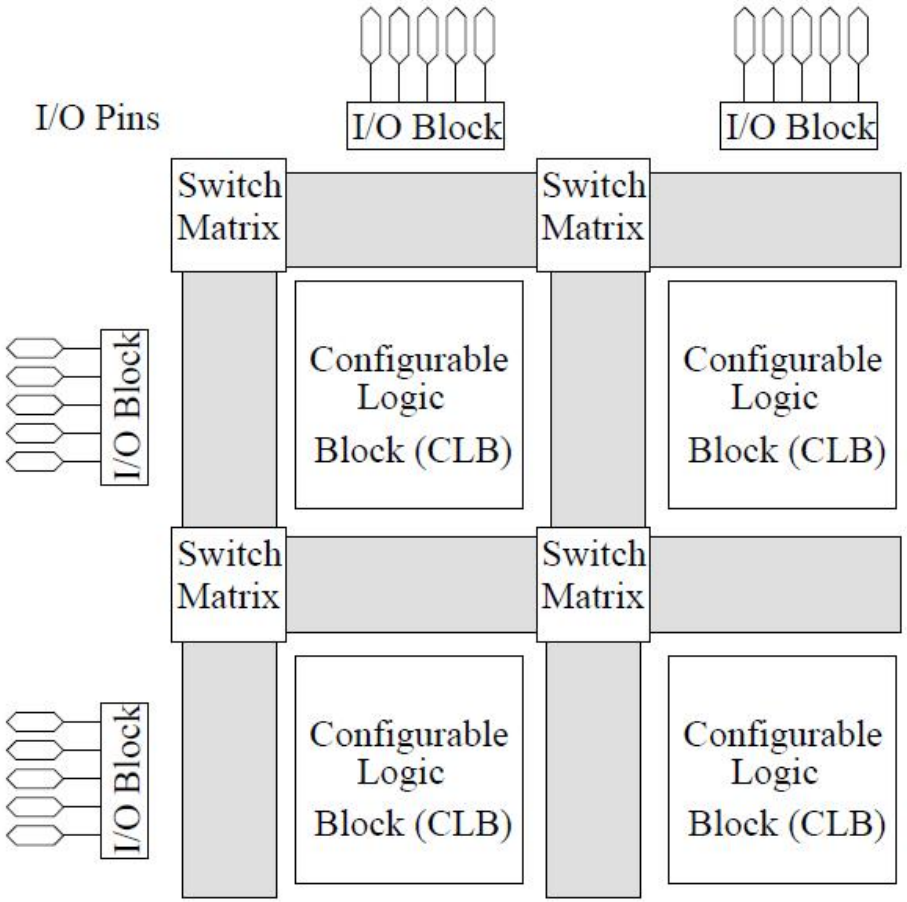
- **Os blocos lógicos programáveis estão localizados mais próximos dos pinos de entrada/saída do FPGA**

- **Nessa arquitetura, os blocos lógicos estão localizados ao redor do array de roteamento central, altamente conectado.**

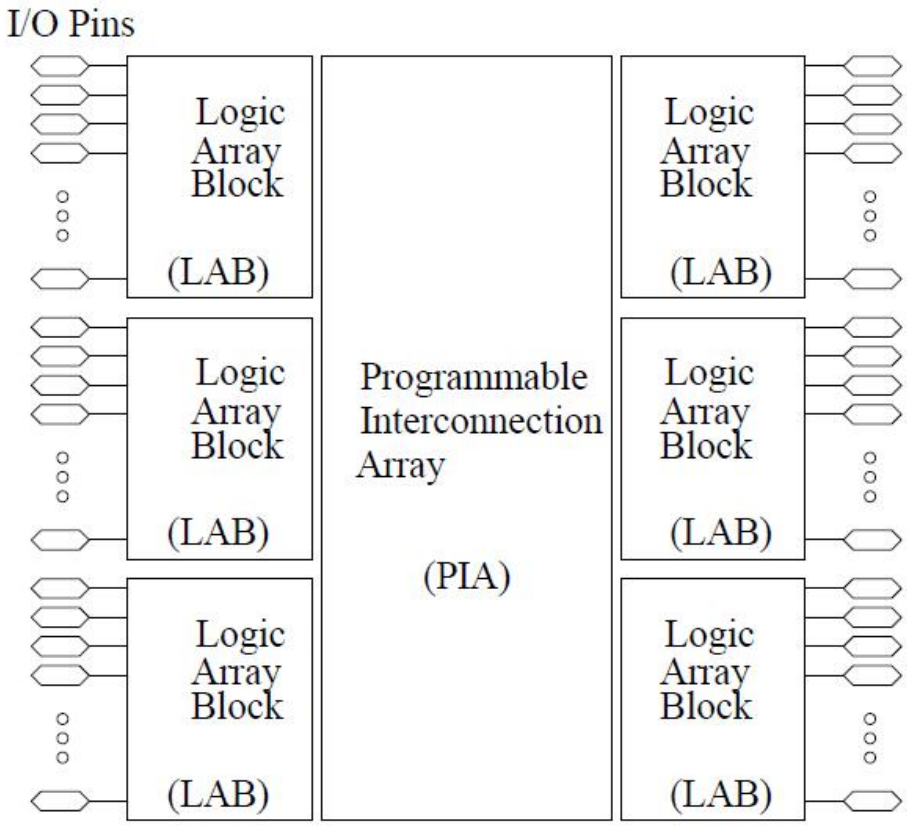


Estrutura interna – Xilinx e Altera

Xilinx



Altera



FPGAs Xilinx: estrutura interna

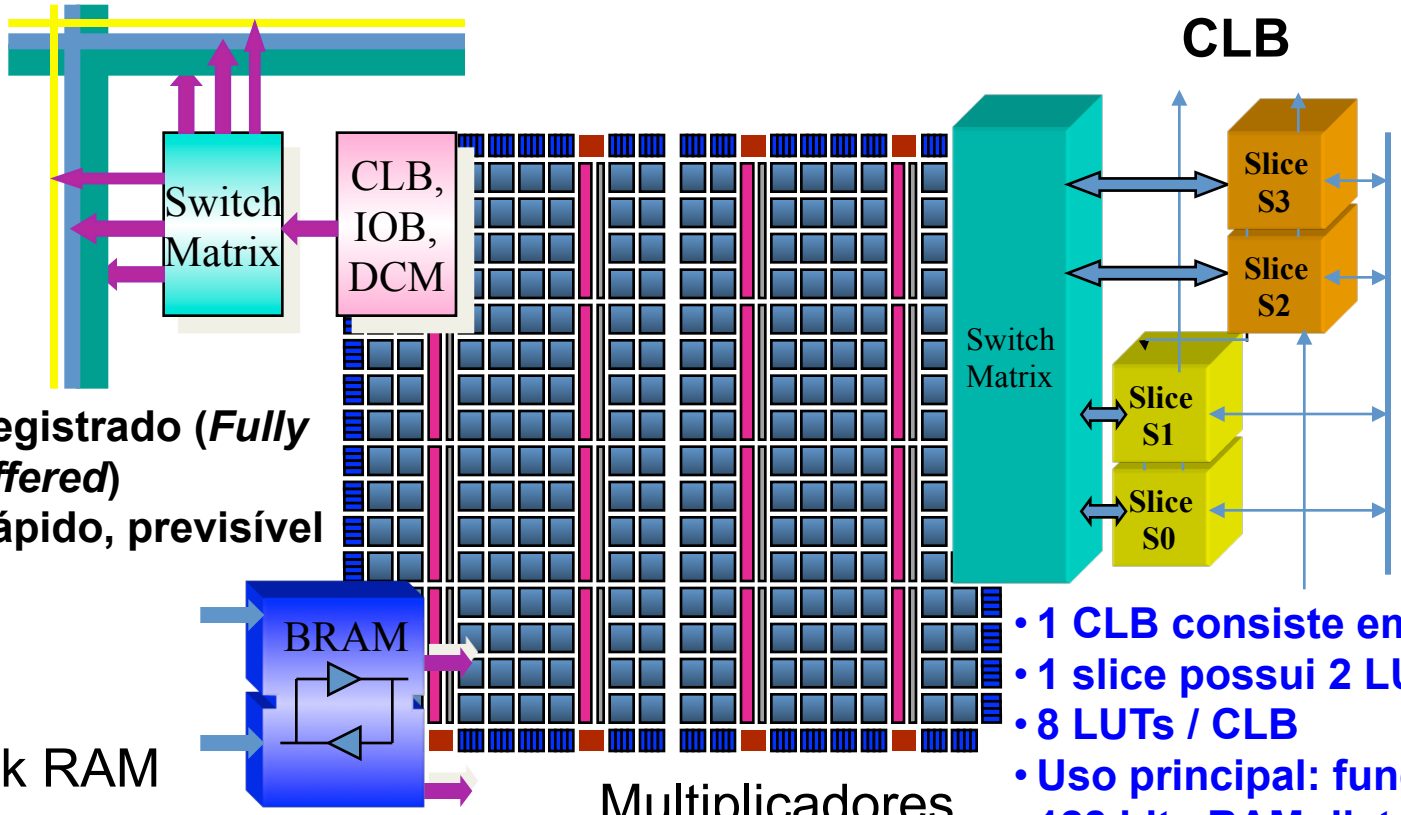
FPGAs Xilinx

- Baixo Custo
  - Família Spartan
  - 1 milhão de portas lógicas equivalentes pode ser adquirido por cerca de US$10.00
- Alto desempenho
  - Família Virtex
  - Virtex, Virtex II, Virtex II-Pro, Virtex-4, Virtex-5, Virtex-6

FPGAs Xilinx - Arquitetura Virtex II

- **1 LUT = LUT 4x1 OU RAM 16 bits OU registrador de deslocamento de 16 bits**

Active Interconnect ™



- **Registrado (*Fully Buffered*)**
- **Rápido, previsível**

- **1 CLB consiste em 4 slices**
- **1 slice possui 2 LUTs 4x1**
- **8 LUTs / CLB**
- **Uso principal: funções lógicas**
- **128 bits RAM distribuída**
- **Registradores de deslocamento**
- **Lógica de vai-um**
- **Portas lógicas (and, or)**
- **Entradas com largura considerável (32:1)**

Block RAM

- 18 k bit porta dupla real (dual port)
- Até 3,5 Mbits / dispositivo

Multiplicadores

- 18b x 18b mult
- 200MHz pipeline

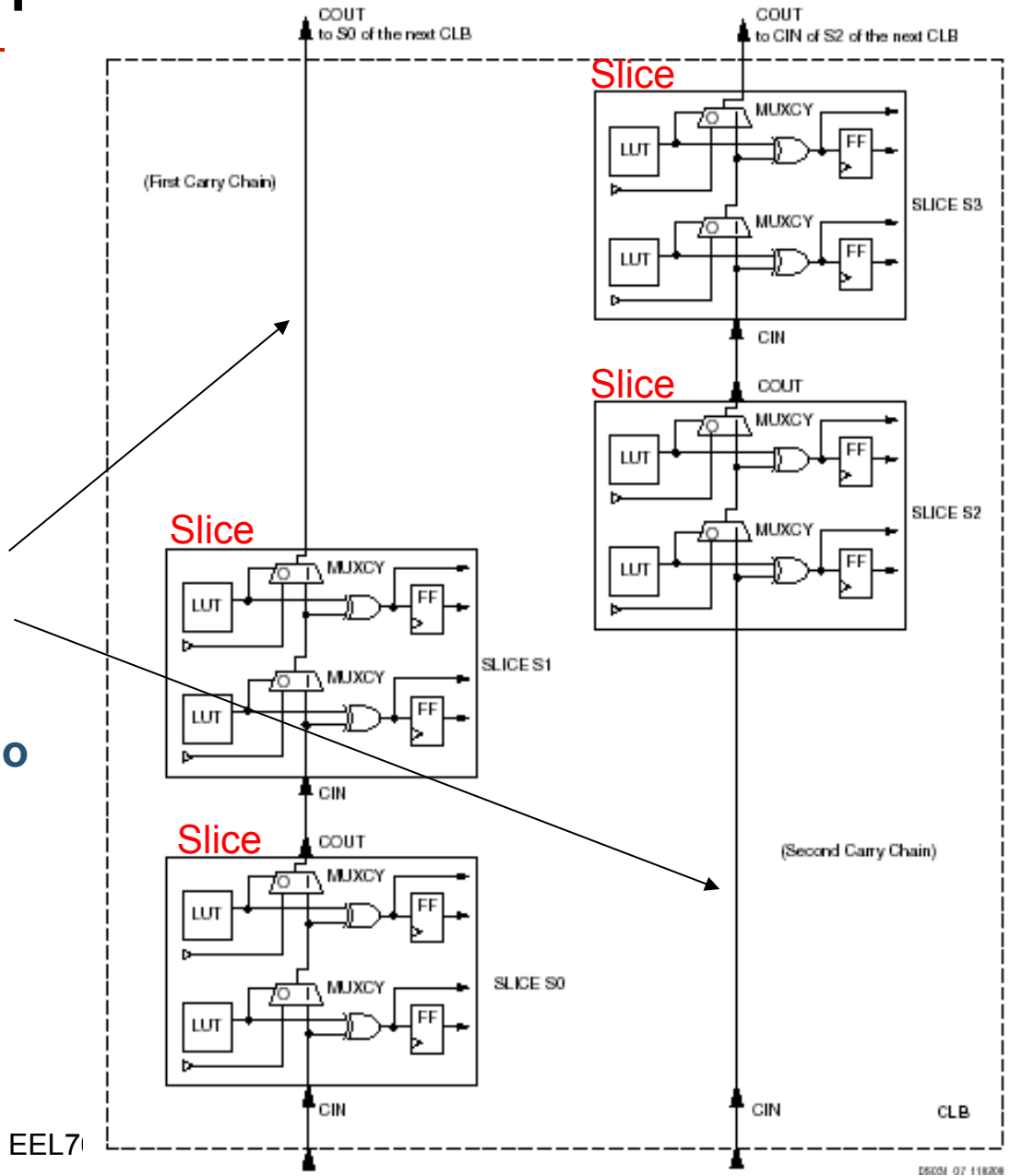
FPGAs Xilinx - Arquitetura Virtex II

RESUMINDO O CLB:

- **4 Slices**
- **8 LUTS / 8 Flip-Flops**
- **2 fluxos de vai-um**
- **64 bits para memória**
- **64 bits para registrador de deslocamento**

•Caminho lógico rápido para propagação de "vai-um"

Acelera operações aritméticas de adição e subtração



FPGAs Xilinx - Arquitetura Virtex II

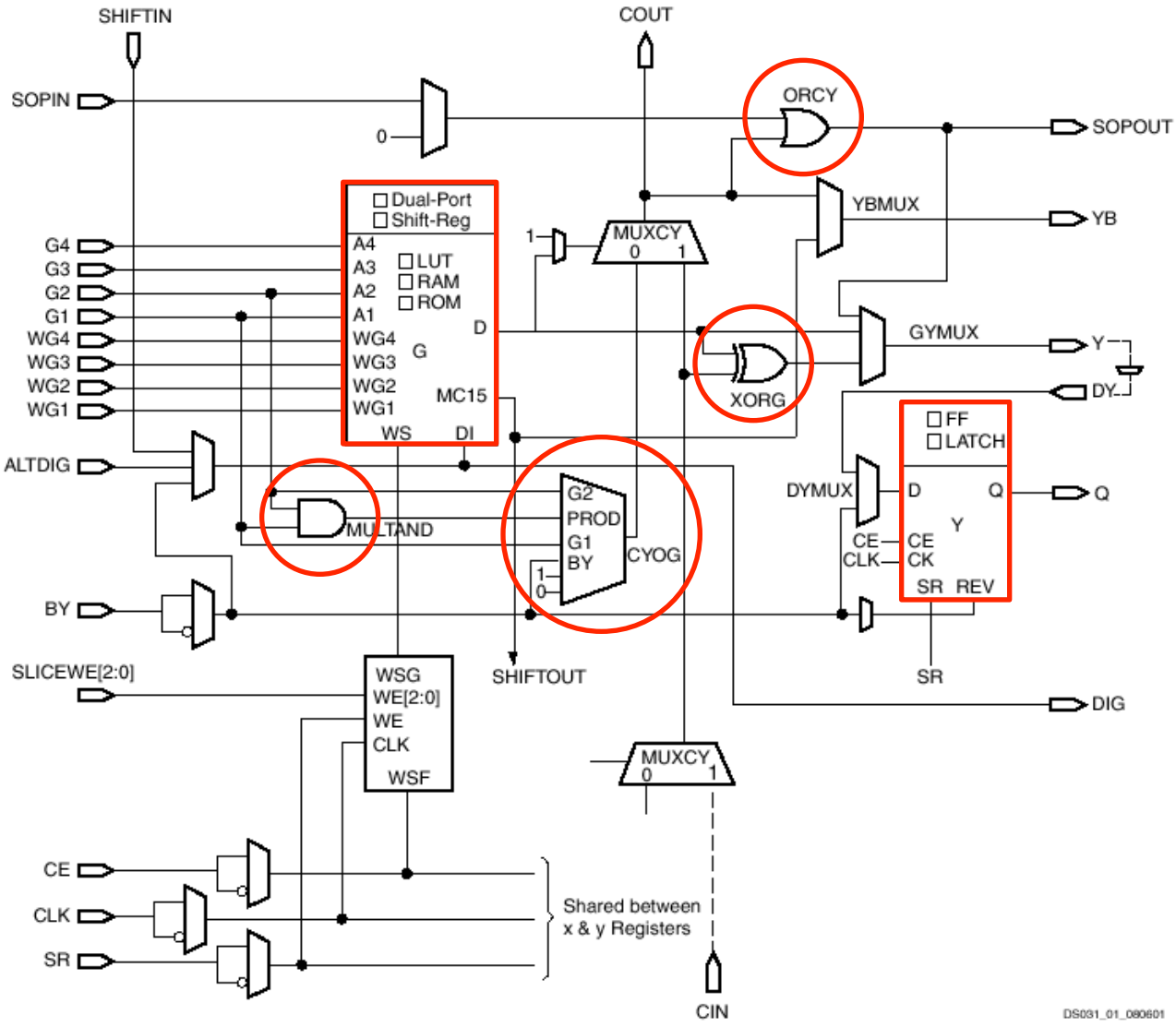


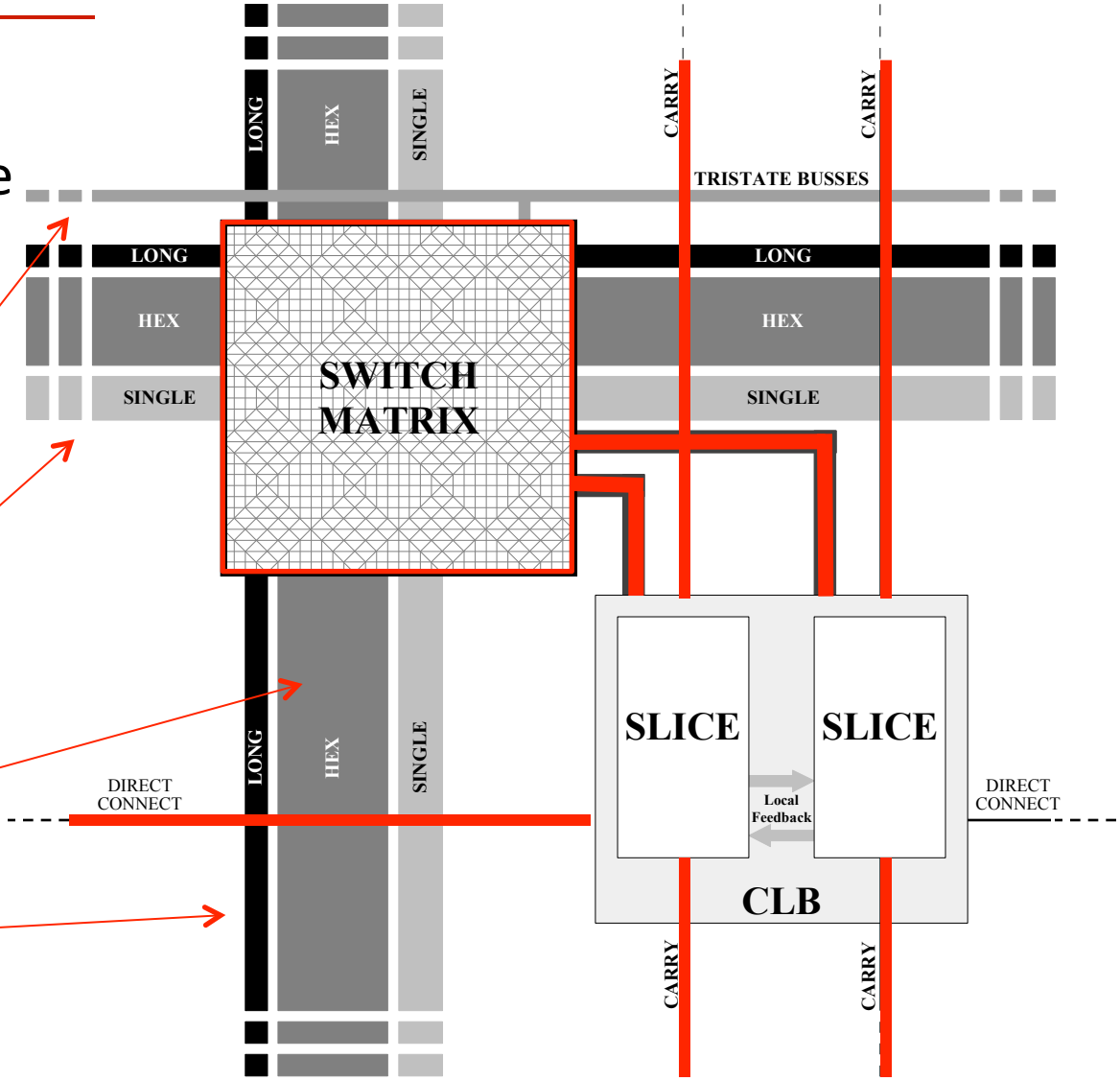
Diagrama de blocos de um slice (meio slice)

- LUT (LUT, RAM, ROM, reg. desloc...)
- FF ou latch
- Portas lógicas
- Muxes

FPGAs Xilinx - Arquitetura Virtex II

CLB e interconexão

- Conexões diretas entre CLBs vizinhos
  - Lógica de carry
- Matriz de conexão
  - CLB às linhas de roteamento
- Linhas de roteamento
  - Tri-state
  - Simples
  - Hexas
  - Longas



FPGAs Xilinx - Arquitetura Virtex II

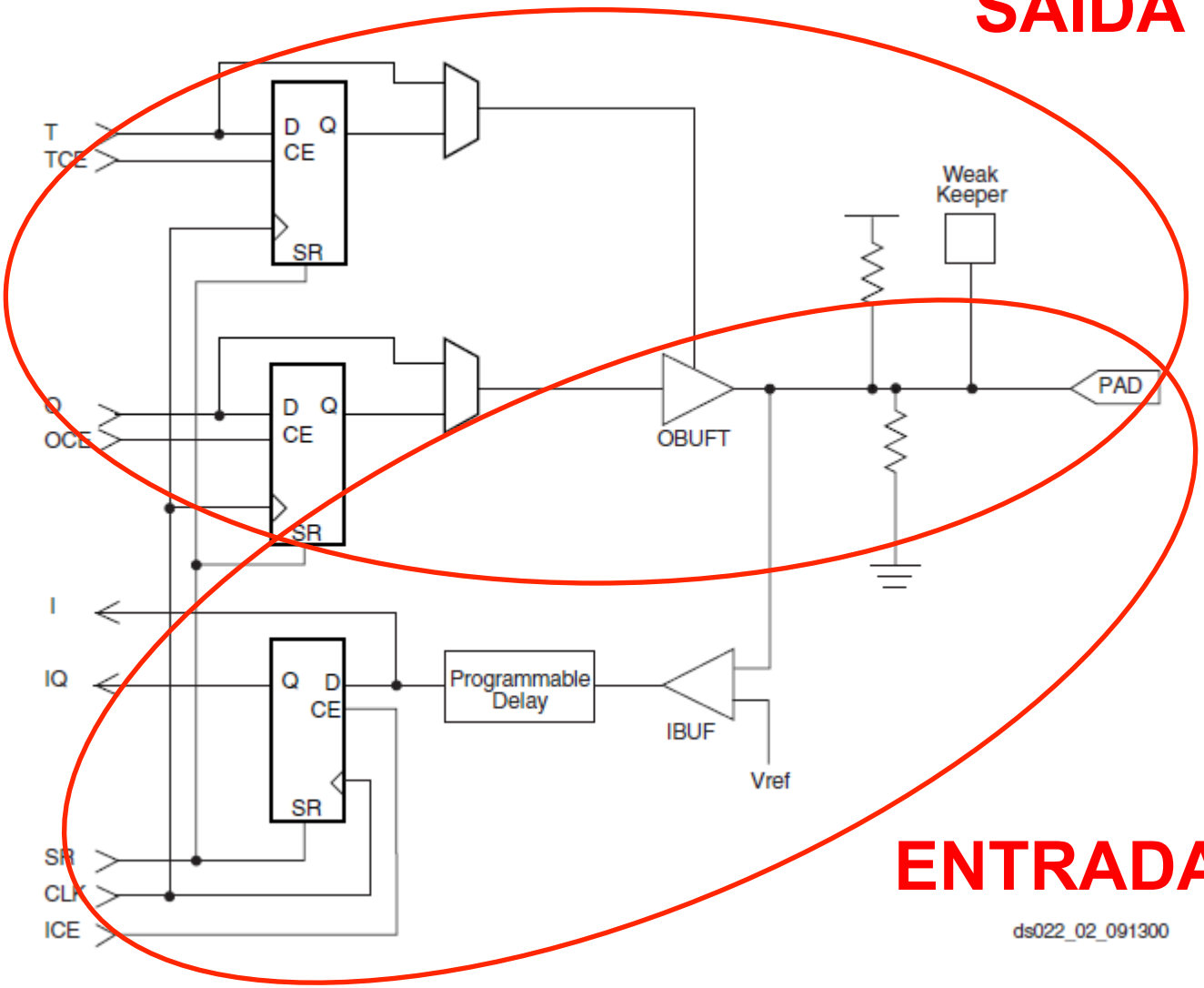


Diagrama de blocos de um I/O block

Cada registrador ou latch, de forma independente, pode ser configurado como:

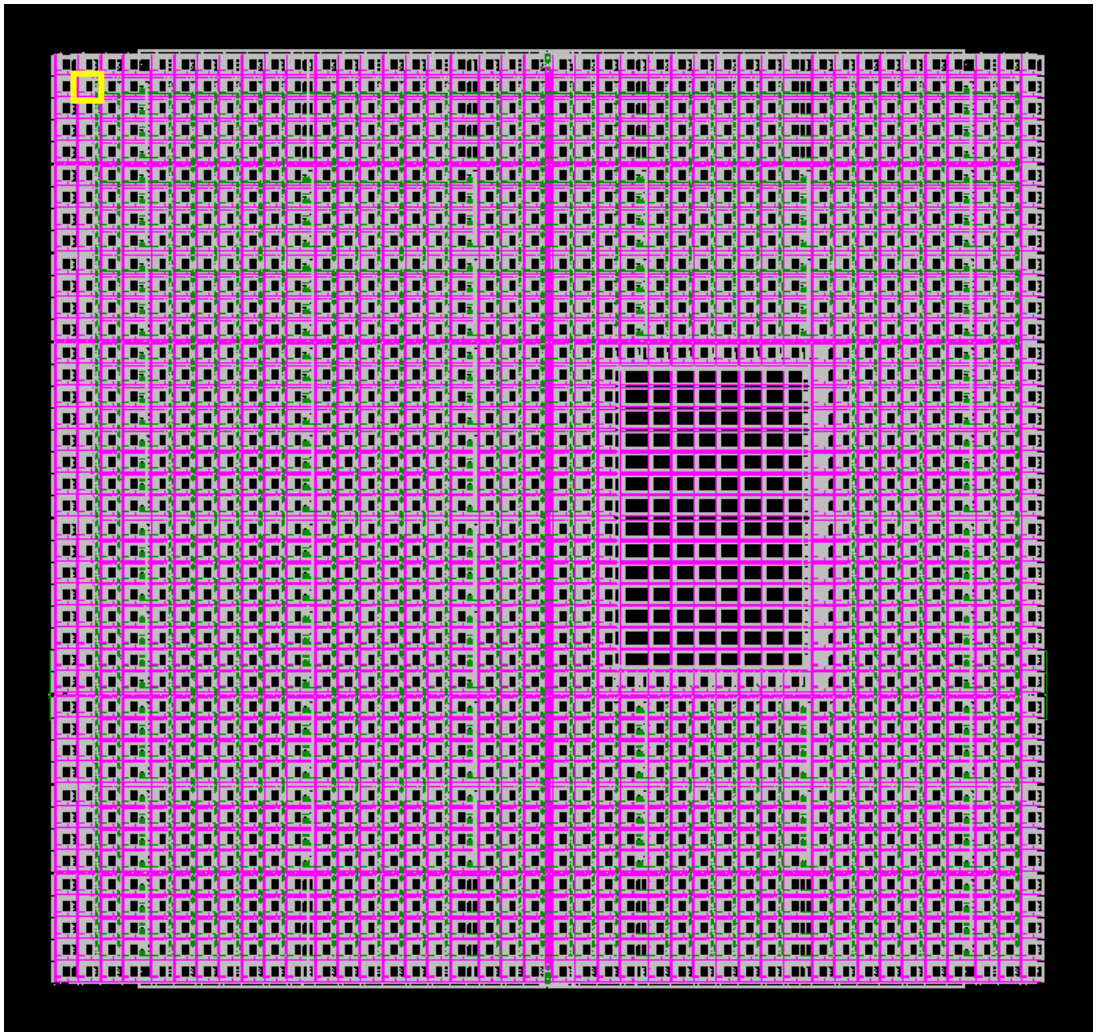
- **Sem set, sem reset**
- **Set síncrono**
- **Reset síncrono**
- **Set e reset síncrono**
- **Set assíncrono**
- **Reset assíncrono**
- **Set e reset assíncrono**

FPGAs Xilinx - Arquitetura Virtex II

Virtex2P XC2VP7 - FPGA Editor View (com todos os fios)

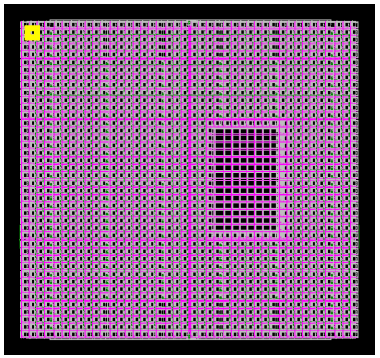
Virtex2P XC2VP7

4.928 slices
44 BRAMs
1 PowerPC
1.423.681 wires



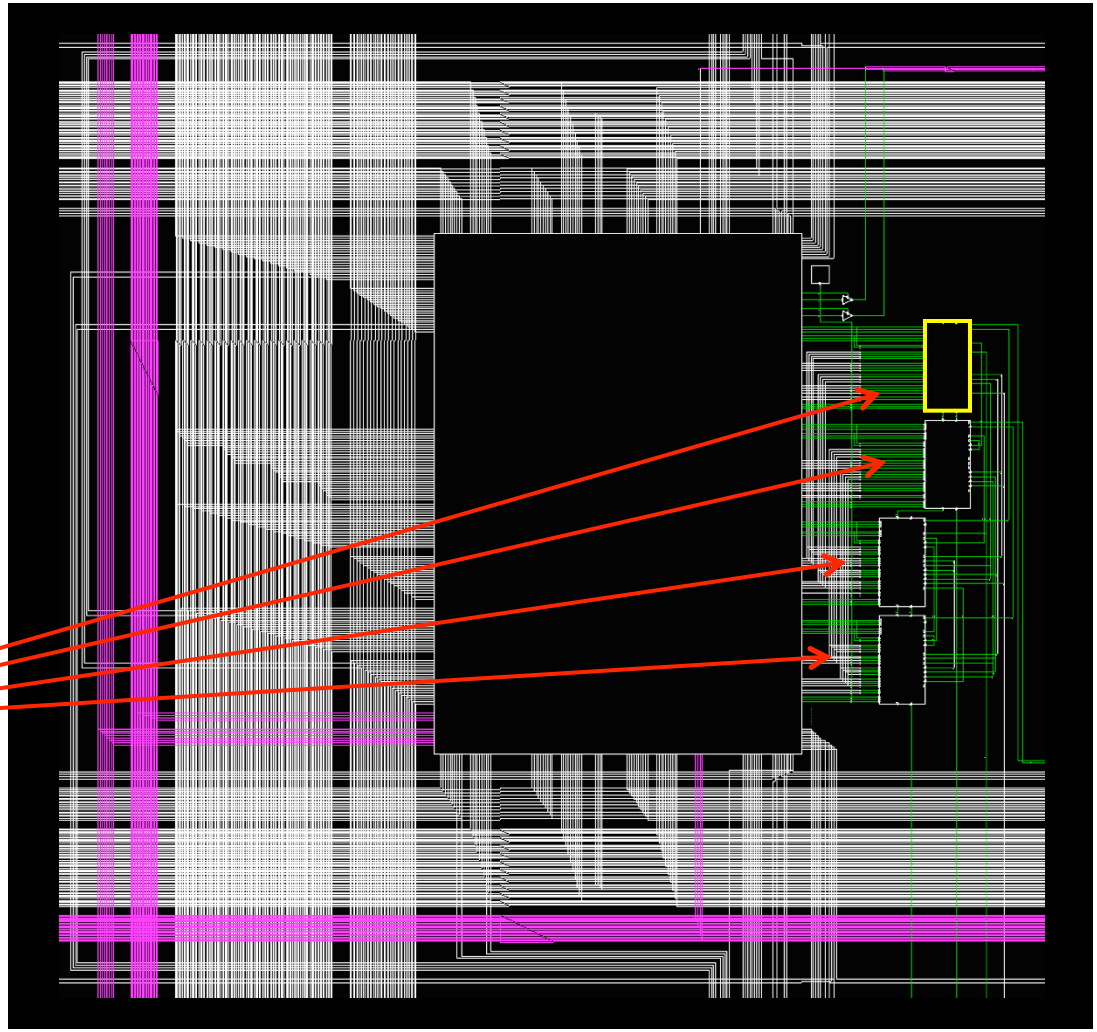
FPGAs Xilinx - Arquitetura Virtex II

Virtex2P XC2VP7



Zoom in

- ***Muitos recursos de roteamento***
- ***Elementos de chaveamento com muitas linhas***
- ***4 slices***



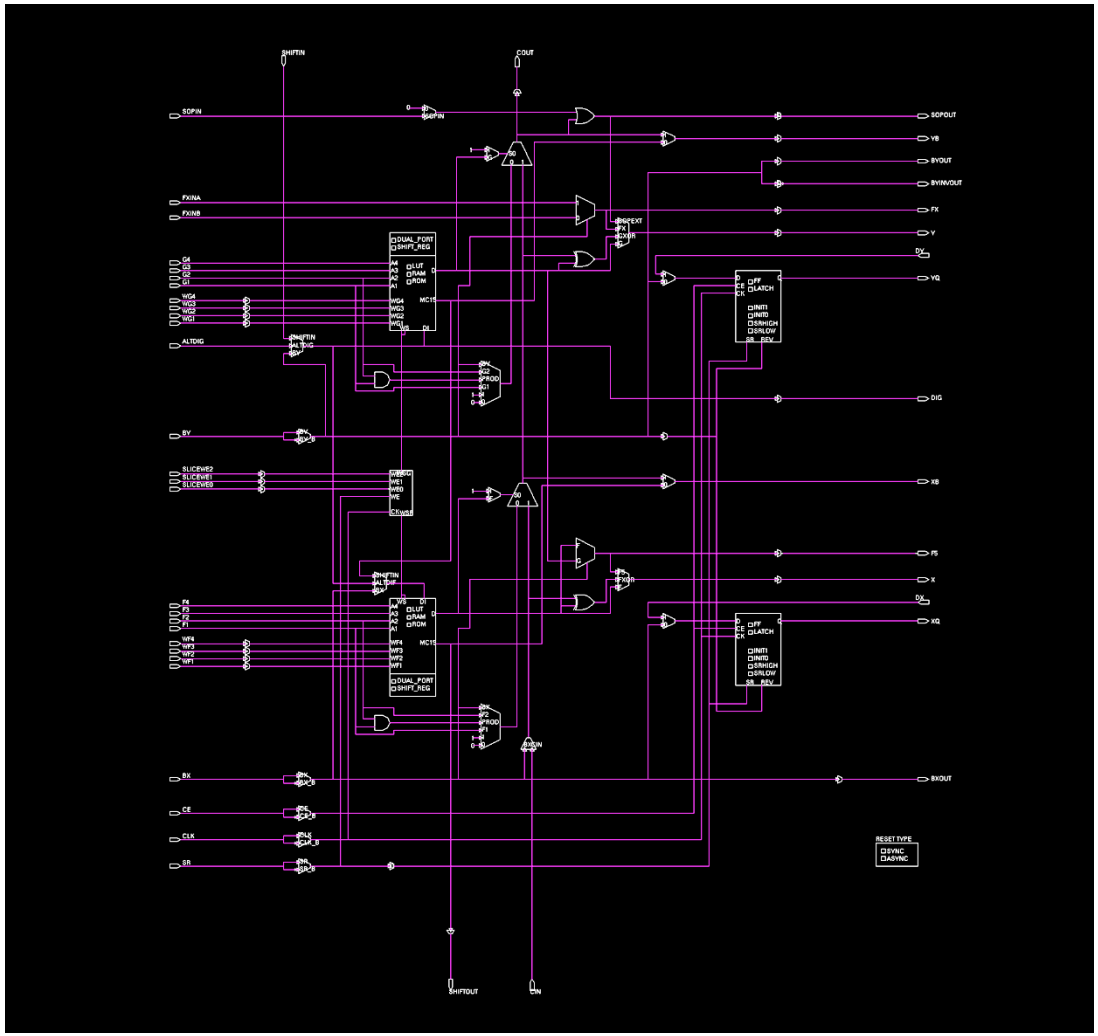
FPGAs Xilinx - Arquitetura Virtex II

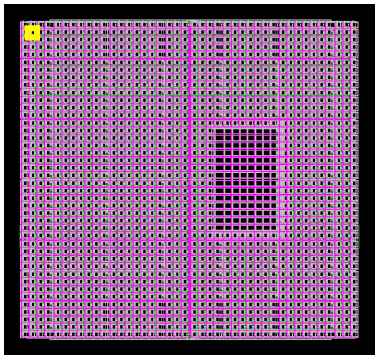
Virtex2P XC2VP7

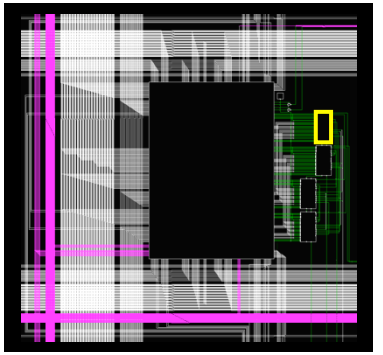
Zoom in slice

Virtex2P slice

- ***2 LUTs***
- ***2 flip-flops***
- ***Muxes***
- ***Lógica de carry dedicada***

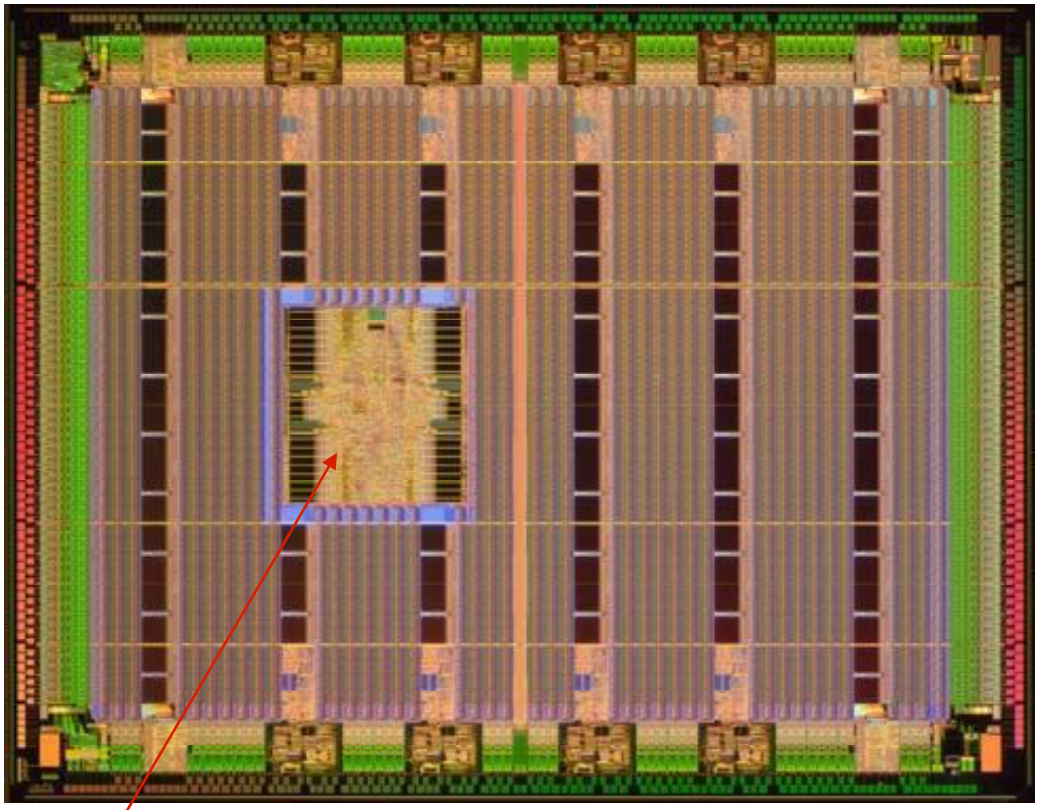






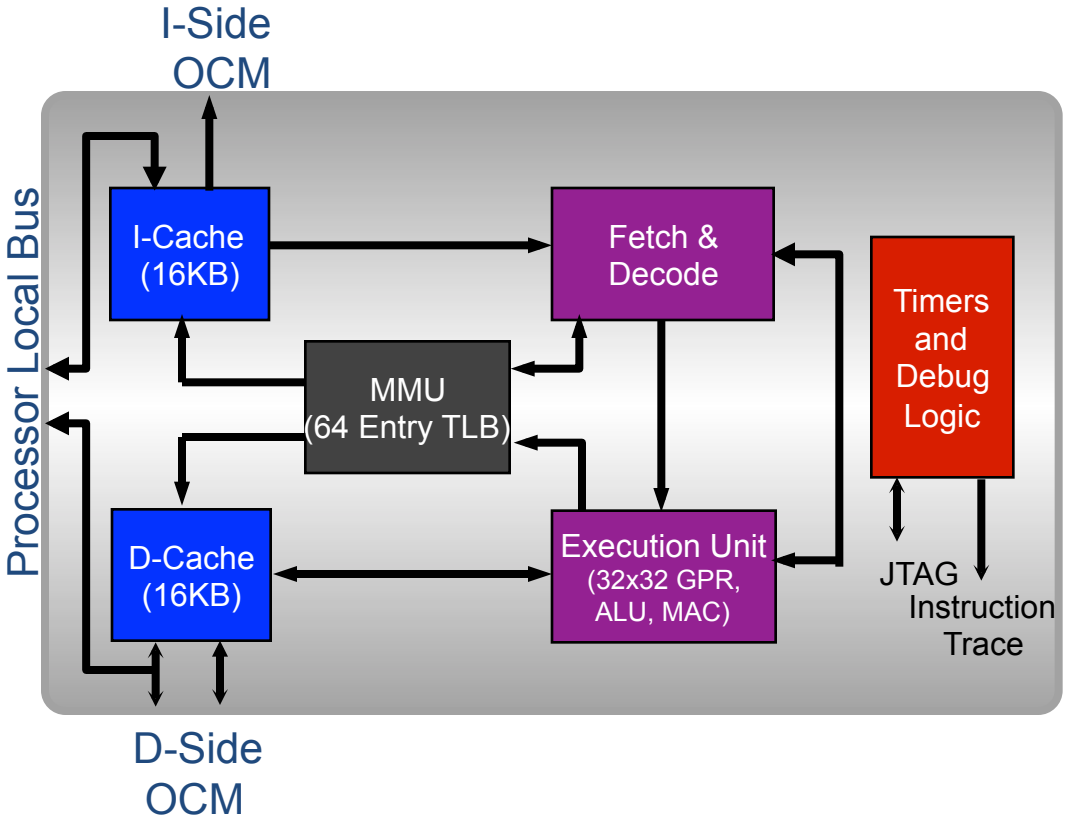
FPGAs Xilinx - Arquitetura Virtex II

- Layout do XC2VP7



FPGAs Xilinx – demais componentes

PowerPC: arquitetura e características

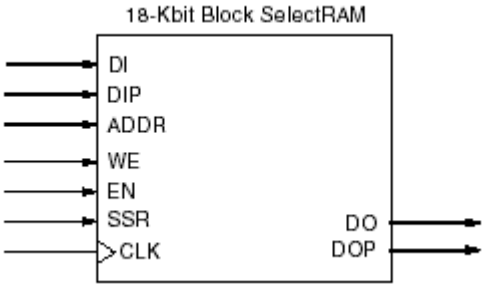


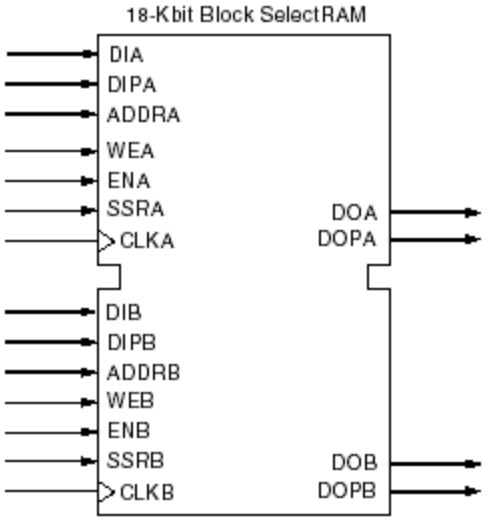
- **Core 300+ MHz**
- **Cache dados 16K, instr. 16K**
- **MMU**
  - **TLB 64 entradas**
  - **Página tamanho variável (1KB-16MB)**
- **Pipeline de 5 estágios**
- **Unidade multiplicação / divisão**
- **Regs uso geral 32 x 32-bit**
- **Interface dedicada de memória on-chip**
- **Suporte para depuração**
- **Linux embarcado**

FPGAs Xilinx – demais componentes

BRAM – Bloco de memória embarcada

- Cada bloco armazena 18 Kbits (BRAM XILINX)
- Conteúdo pode ser definido pelo bitstream
- Porta simples ou dupla
- Configurações
  - 16k *x* 1, 8k *x* 2, 8k *x* 4, 2k *x* 9, 1k *x* 18, 512 *x* 36





FPGAs: aplicações

FPGA: aplicações

- Originalmente, plataforma de prototipação para ASICs
- Flexibilidade para reprogramação típica de software, e características de confiabilidade e paralelismo encontradas em elementos de hardware
- Favorece criação de circuitos digitais customizados com os benefícios de desempenho do hardware, e as vantagens competitivas de "time-to-market" existente em projetos de software.
- Algumas aplicações típicas:
  - Algoritmos de criptografia
  - IA / redes neurais
  - Digital Signal Processing (DSP) – operações MAC
  - Processamento de imagens
  - Algoritmos de codificação e decodificação de protocolos de comunicação

FPGA: aplicações

- **Aplicações adequadas para FPGAs**
  - **Necessidade de alta velocidade de processamento**
  - **Necessidade de maior velocidade de processamento com menor consumo de energia (em relação a microprocessadores)**
  - **Computação pode ser paralelizada**
  - **Operações a nível de bit**
  - **Operações aritméticas não ortodoxas (ex. palavras com tamanhos exóticos – 13 bits, ...)**
  - **Necessidade de reconfiguração**
  - **Volume de produção de pequeno para médio**

Aplicações para FPGAs
Estudo de Caso: somador de 4 bits

Implementação de Somadores

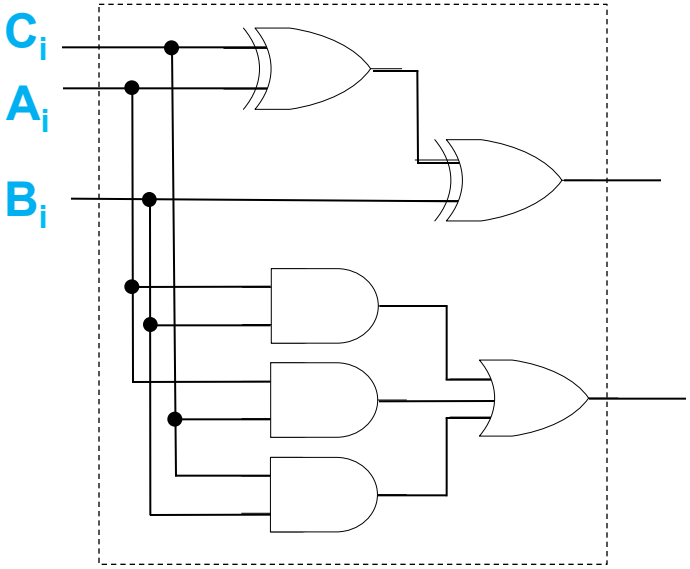
FA

$$
\begin{array}{rccccc}
 & C_5 & C_4 & C_3 & C_2 & C_1 \\
 & & A_4 & A_3 & A_2 & A_1 & A_0 \\
+ & & B_4 & B_3 & B_2 & B_1 & B_0 \\
\hline
 & & S_4 & S_3 & S_2 & S_1 & S_0
\end{array}
$$

← "vai-um" (*carry*)
← 1° operando
← 2° operando
← soma

Somador com três entradas (C_i, A_i e B_i) e duas saídas (S_i e C_{i+1})
"**Somador completo (ou *full-adder*)**"

C_i	A_i	B_i	S_i	C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



$S_i = C_i$ xor A_i xor B_i

$C_{i+1} = A_i$ and B_i or A_i and C_i or B_i and C_i

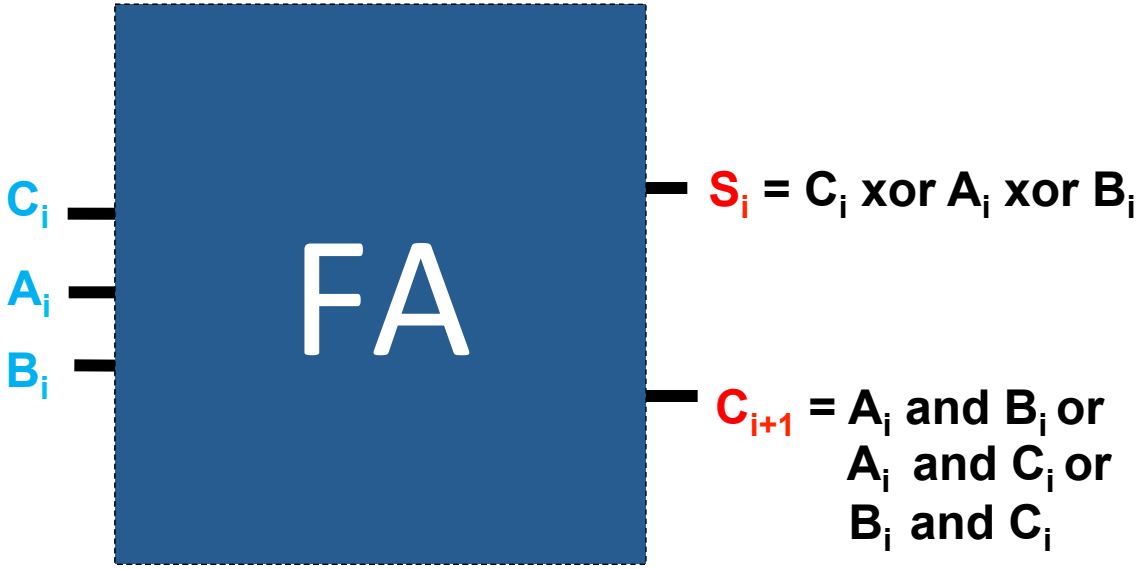
Implementação de Somadores

FA

$$
\begin{array}{rccccc}
 & C_5 & C_4 & C_3 & C_2 & C_1 \\
 & & A_4 & A_3 & A_2 & A_1 & A_0 \\
+ & & B_4 & B_3 & B_2 & B_1 & B_0 \\
\hline
 & & S_4 & S_3 & S_2 & S_1 & S_0
\end{array}
\quad
\begin{array}{l}
\leftarrow \text{ “vai-um” (\textit{carry})} \\
\leftarrow \text{ 1° operando} \\
\leftarrow \text{ 2° operando} \\
\leftarrow \text{ soma}
\end{array}
$$

Somador com três entradas (C_i, A_i e B_i) e duas saídas (S_i e C_{i+1})
“**Somador completo (ou *full-adder*)**”

C_i	A_i	B_i	S_i	C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



Implementação de Somadores

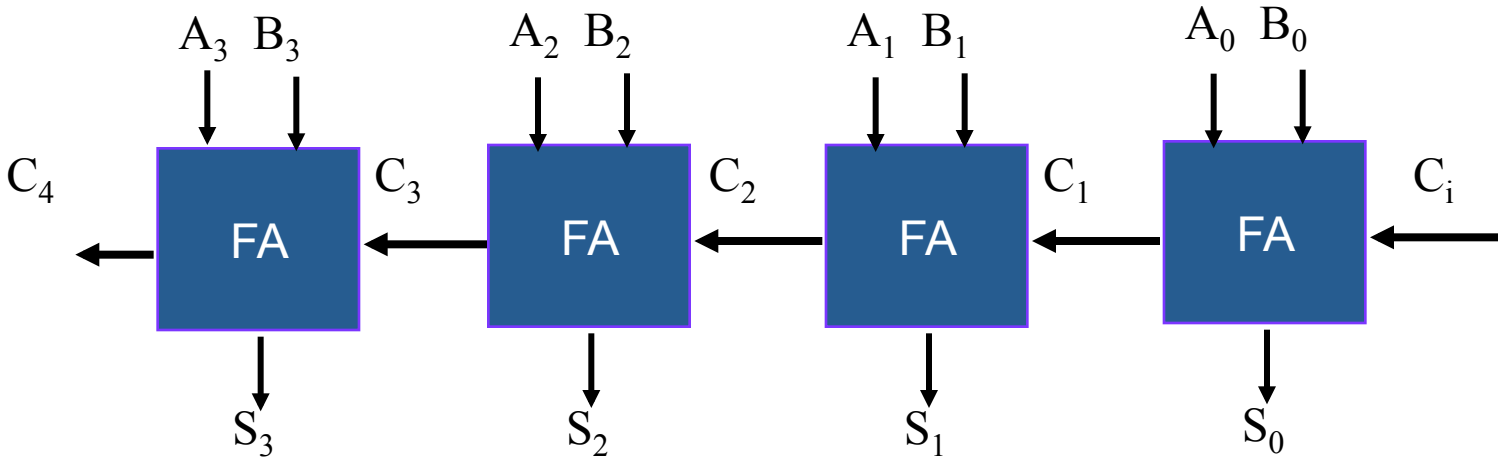
RCA

$$
\begin{array}{cccccc}
 & C_4 & C_3 & C_2 & C_1 & \\
 & & A_3 & A_2 & A_1 & A_0 \\
+ & & B_3 & B_2 & B_1 & B_0 \\
\hline
 & & S_3 & S_2 & S_1 & S_0
\end{array}
$$

Somador paralelo "*RIPPLE CARRY*"

- Utiliza-se um FA para cada bit do símbolo de entrada.
- No bit menos significativo pode ser utilizado um HA.
- Somador carry ripple de *n* bits apresenta resposta final (estável) apenas após C_1 .. C_{n-1} estabilizarem.

Ripple Carry Addder (RCA)



Implementação de Somadores - síntese

Síntese para FPGAs

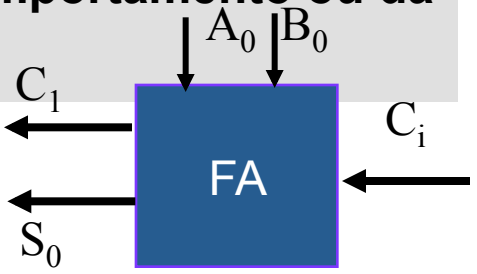
Processo de otimização para adaptar um projeto lógico aos recursos existentes no FPGA (LUTs, IOBs, long lines, carry, ...)

Fonte VHDL do RCA de 4 bits

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
--
-- Entity ADDER
--
entity ADDER is
    generic (N: in integer := 4);
    port (
        A, B: in  std_logic_vector(N-1 downto 0);
        CI  : in  std_logic;
        S   : out std_logic_vector(N-1 downto 0);
        COUT: out std_logic
    );
end ADDER;
```

A biblioteca define os tipos

- Define interface entre o hardware e o ambiente
- Não contém definição do comportamento ou da estrutura internos



Descrição de módulos de hardware em VHDL

- COMPORTAMENTO : architecture
  - Especifica o comportamento e/ou a estrutura internos da *entity*
  - Deve ser associada a uma *entity* específica
  - Uma *entity* pode ter associada a si várias *architecture* (representando diferentes formas de implementar um mesmo módulo)

```
architecture comp of ADDER is
begin
   ...
end comp;
```

Fonte VHDL do CRA de 4 bits

RCA

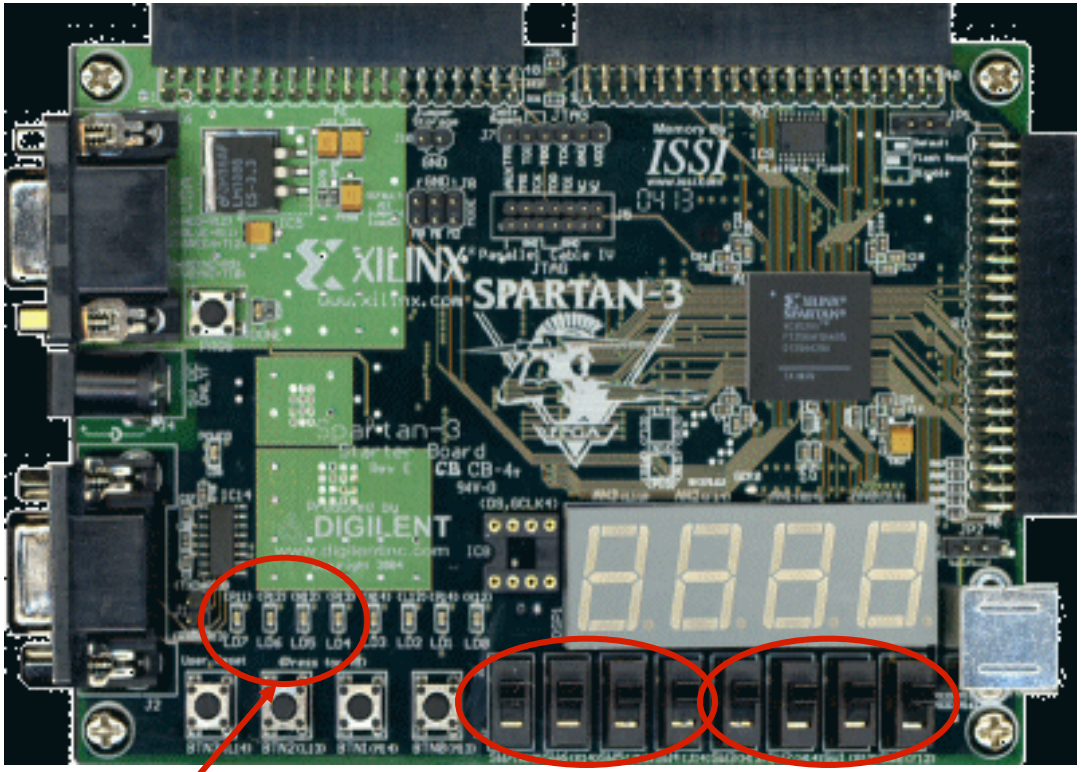
SÍNTESE

```
--
-- Implementacao do somador utilizando Carry Ripple Adder
--
architecture RTL_CRA of ADDER is
   procedure FADDER (
      signal A, B, CI: in std_logic;
      signal S, COUT: out std_logic
   ) is
   begin
      S <= A xor B xor CI;
      COUT <= (A and B) or (A and CI) or (B and CI);
   end FADDER;
   signal C: std_logic_vector(A'length-1 downto 1);
begin
   gen: for j in A'range generate
      genlsb: if j = 0 generate
         FADDER (A => A(0), B => B(0), CI => CI, S => S(0), COUT => C(j+1));
      end generate;
      genmid: if (j > 0) and (j < A'length-1) generate
         FADDER (A => A(j), B => B(j), CI => C(j), S => S(j), COUT => C(j+1));
      end generate;
      genmsb: if j = A'length-1 generate
         FADDER (A => A(j), B => B(j), CI => C(j), S => S(j), COUT => COUT);
      end generate;
   end generate;
end RTL_CRA;
```

Prototipação em hardware

Definição dos pinos de entrada/saída

- Placas de prototipação tem recursos de entrada e saída:
  - Leds, chaves, displays, teclado, serial, ethernet...



Soma(3 downto 0);

A(3 downto 0); **B(3 downto 0);**

Prototipação em hardware

Definição dos pinos de entrada/saída

- Um arquivo relaciona as entradas e saídas do VHDL com os recursos da placa
- Na Xilinx esse arquivo é denominado UCF *(user constraint file)*
- Ver arquivo:
  http://www.inf.pucrs.br/~eduardob/laborg/circuitos_digitais/vhdl/ex/add/add.ucf/

Table 4-1: **Slider Switch Connections**

Switch	SW7	SW6	SW5	SW4	SW3	SW2	SW1	SW0
FPGA Pin	K13	K14	J13	J14	H13	H14	G12	F12

Table 4-2: **Push Button Switch Connections**

Push Button	BTN3 (User Reset)	BTN2	BTN1	BTN0
FPGA Pin	L14	L13	M14	M13

Table 4-3: **LED Connections to the Spartan-3 FPGA**

LED	LD7	LD6	LD5	LD4	LD3	LD2	LD1	LD0
FPGA Pin	P11	P12	N12	P13	N14	L12	P14	K12

Prototipação em hardware

Definição dos pinos de entrada/saída

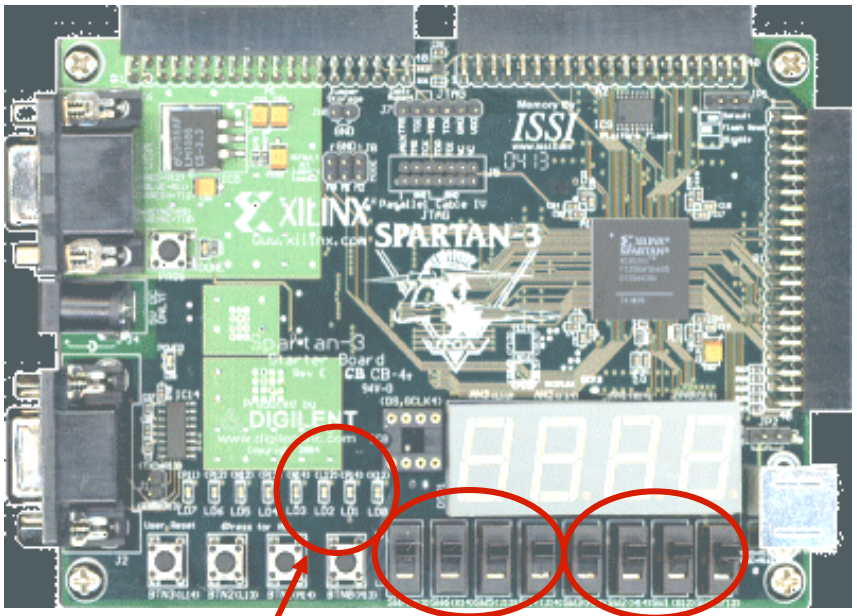
UCF com mapeamento de sinais da entity aos pinos do FPGA

```
### UCF DO PROJETO SOMADOR DE 4 BITS

NET "A(3)"  LOC = "K13" ;
NET "A(2)"  LOC = "K14" ;
NET "A(1)"  LOC = "J13" ;
NET "A(0)"  LOC = "J14" ;

NET "B(3)"  LOC = "H13" ;
NET "B(2)"  LOC = "H14" ;
NET "B(1)"  LOC = "G12" ;
NET "B(0)"  LOC = "F12 " ;

NET "soma(3)"  LOC = "N14" ;
NET "soma(2)"  LOC = "L12" ;
NET "soma(1)"  LOC = "P14" ;
NET "soma(0)"  LOC = "K12" ;
```



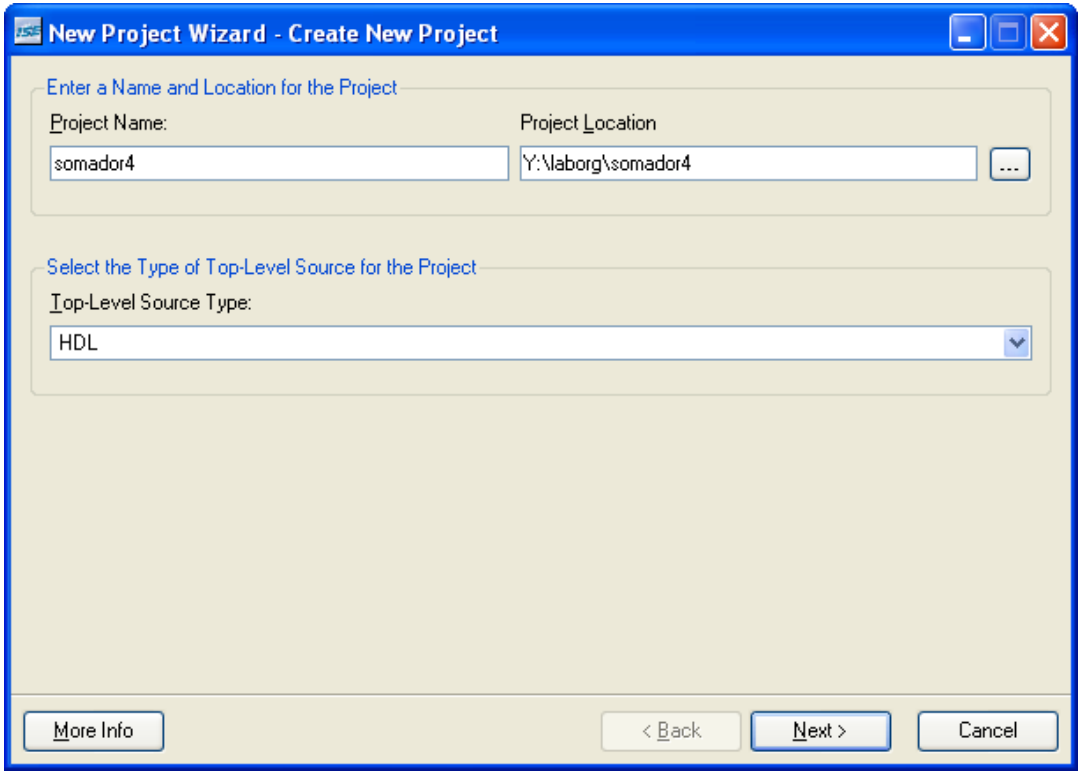
A(3 downto 0); B(3 downto 0);

Soma(3 downto 0);

Prototipação em hardware – síntese do VHDL

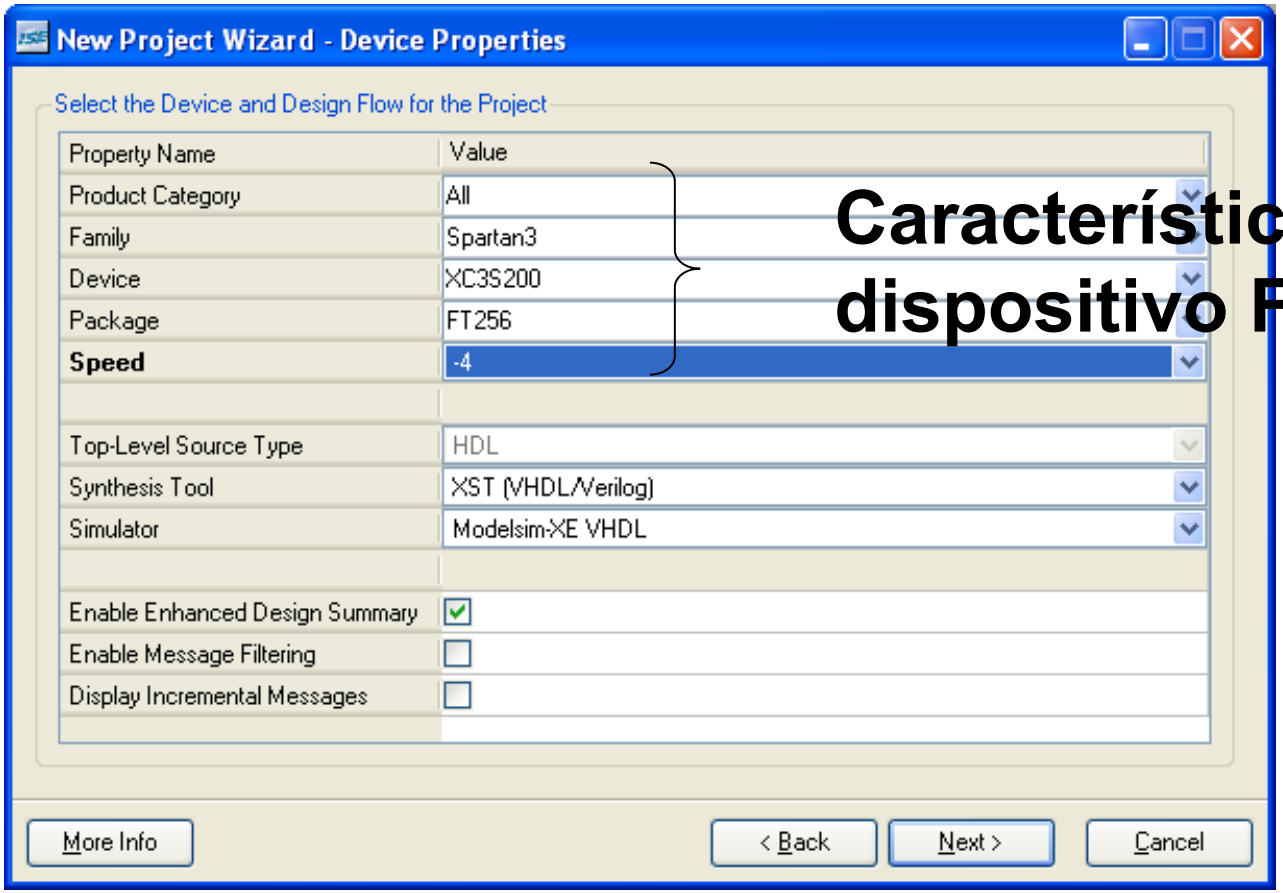
- Criar um diretório para o arquivo VHDL (soma.vhd) e para o arquivo UCF (soma.ucf)
- Abrir a ferramenta ISE (

) e criar um novo projeto (File → New Project):



Prototipação em hardware – síntese do VHDL

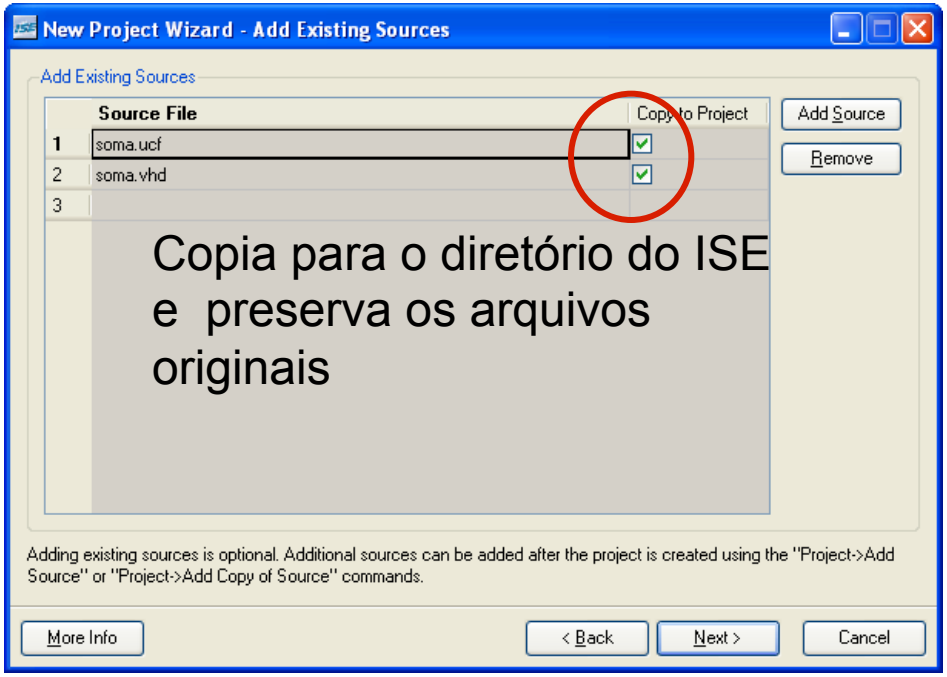
- A placa disponível no laboratório possui um FPGA Spartan 3



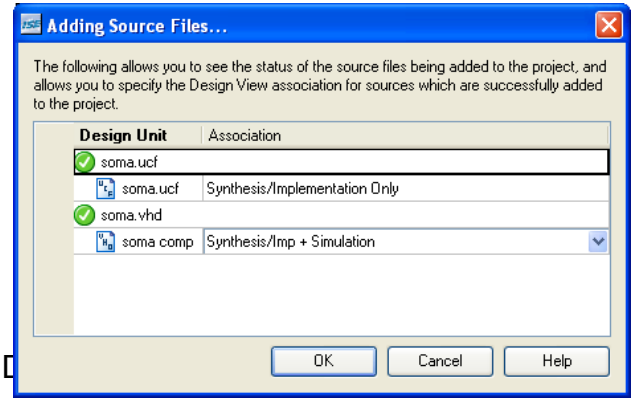
Características do dispositivo FPGA

Prototipação em hardware – síntese do VHDL

- Próxima janela, criação de novos fontes – *next*
- Próxima janela, inclusão dos fontes VHDL:



- **Ao final há um resumo do projeto (com detecção da função do UCF):**

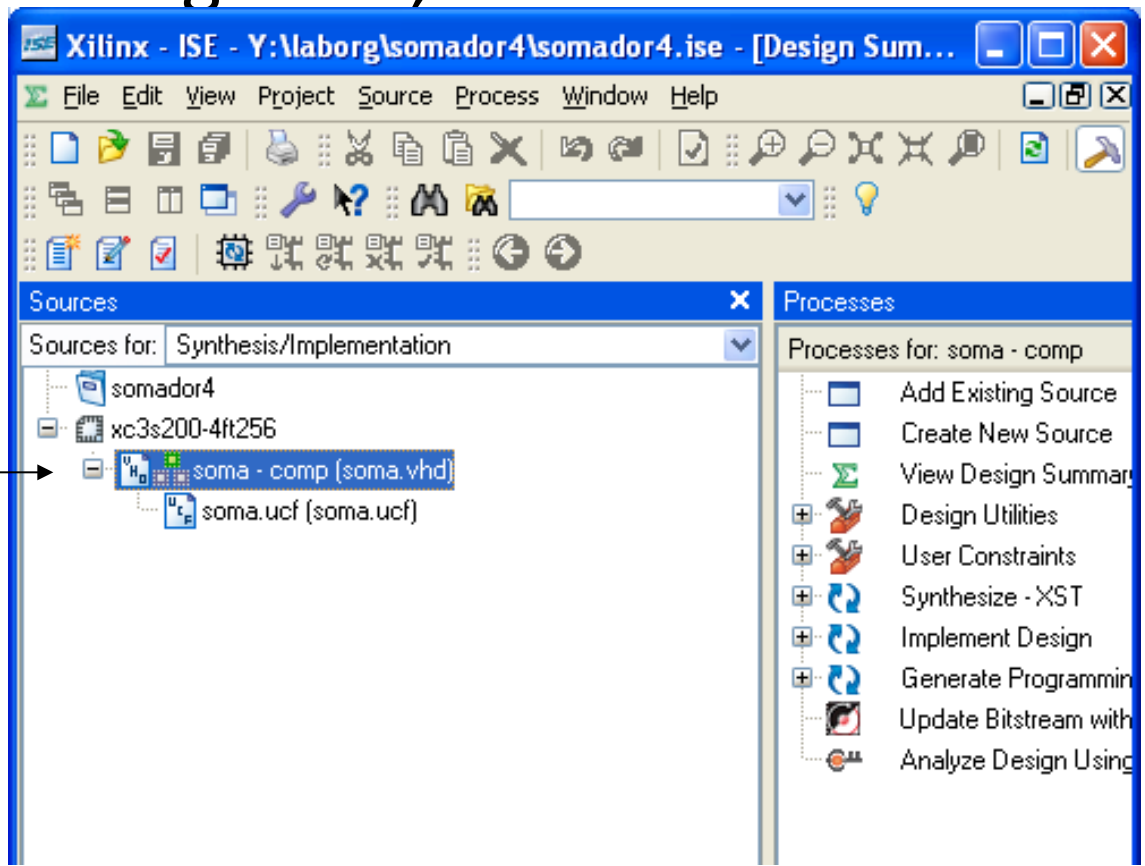


Prototipação em hardware – síntese do VHDL

Ambiente ISE – browser do projeto

- Se todos os passos de criação foram corretamente seguidos, obtém-se:

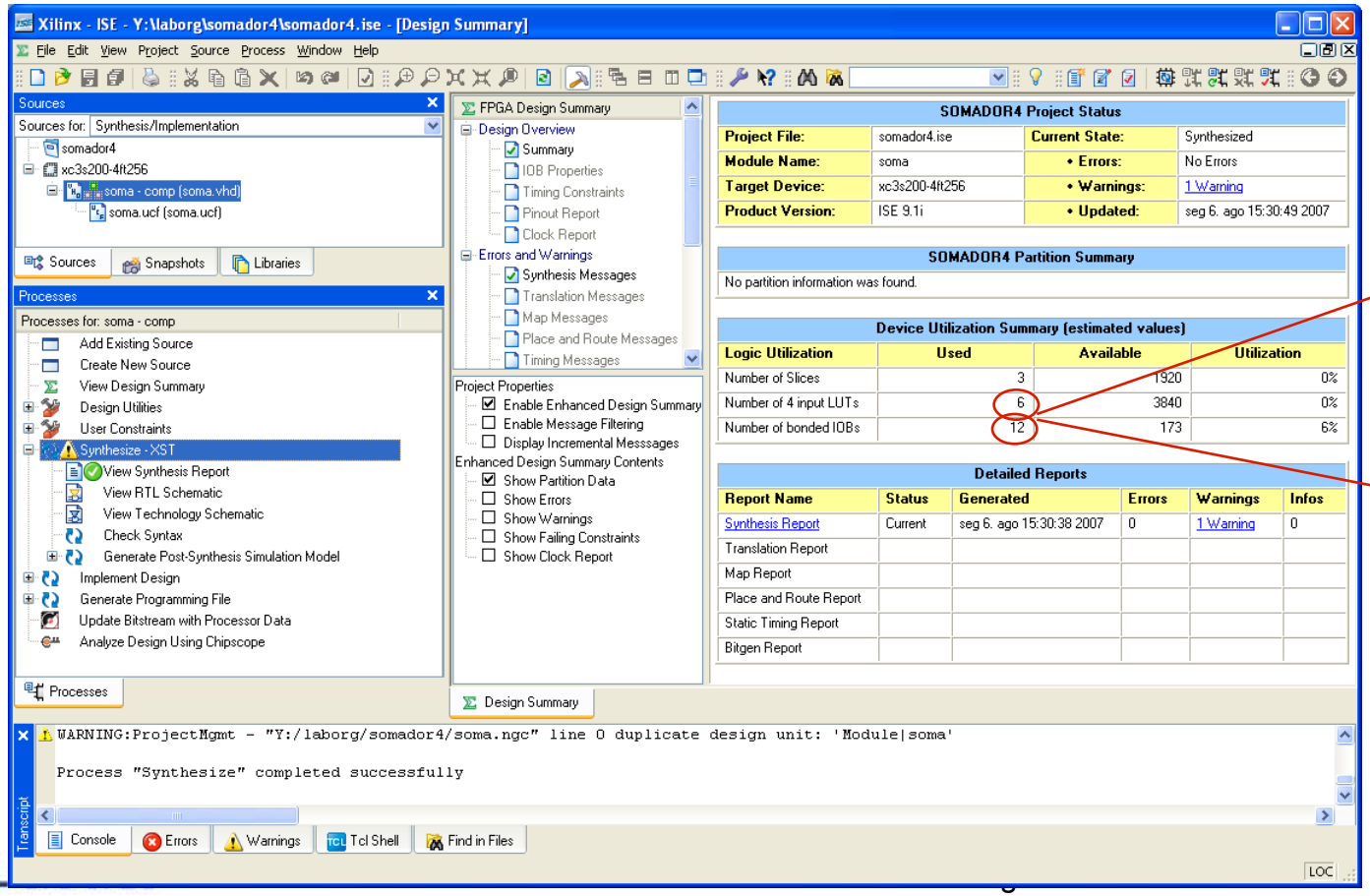
Top do projeto



Prototipação em hardware – síntese do VHDL

Passo 1: síntese lógica

- Transforma o VHDL em portas lógicas
- Para executar, duplo click em "Synthesize XST"
- Ao final obtém-se o relatório da síntese no lado direito



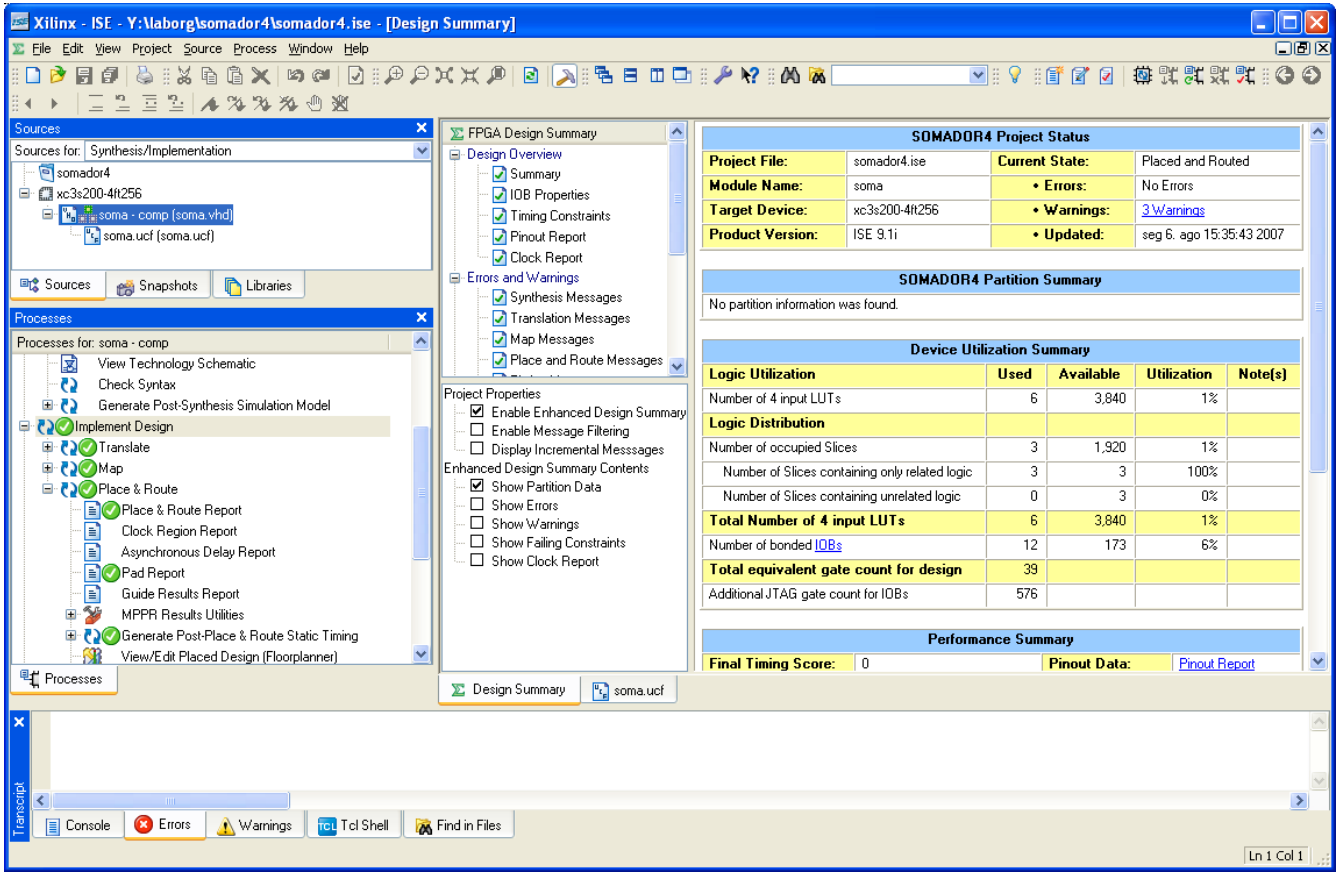
6 LUTs de 3840

12 entradas

Prototipação em hardware – síntese do VHDL

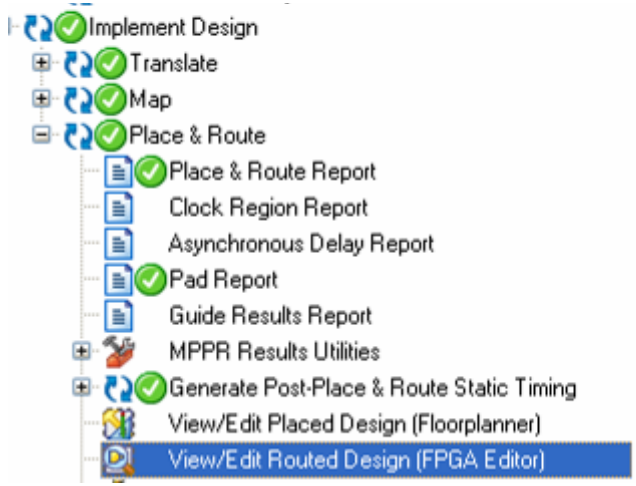
Passo 2: síntese física

- Realiza o posicionamento e as conexão no FPGA
- Duplo click no "Implement Design"

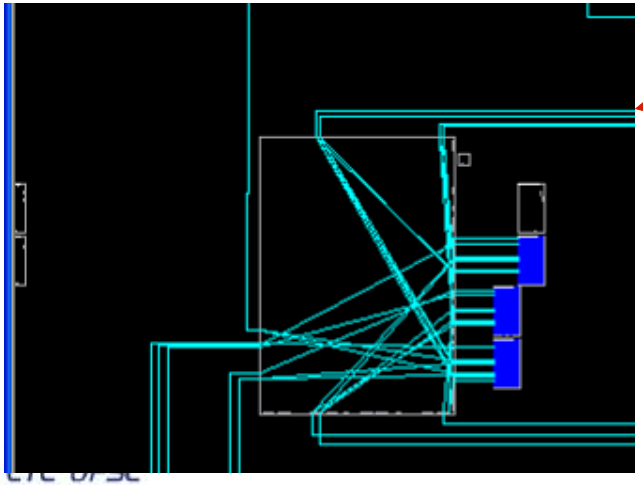


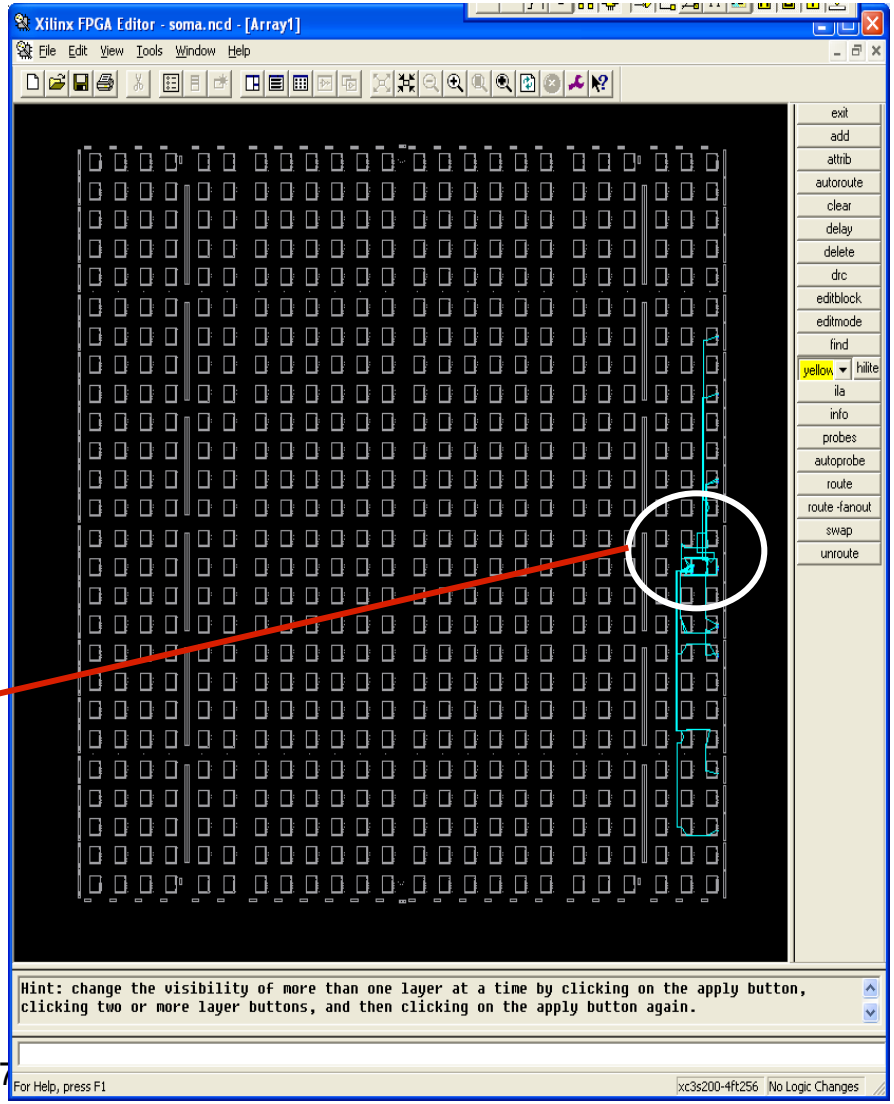
Prototipação em hardware – síntese do VHDL

Visualização no FPGA - Selecionar FPGA Editor



6 LUTs (em três SLICES)

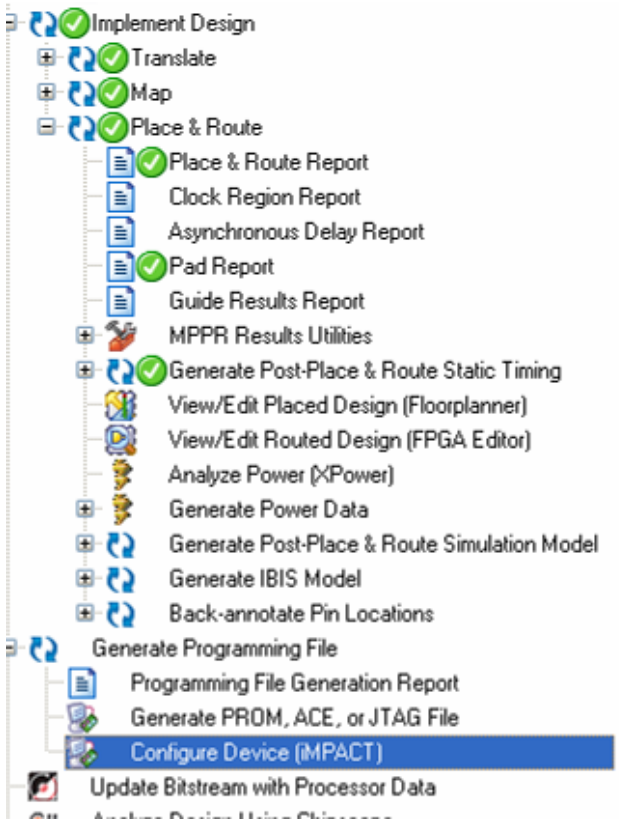




Prototipação em hardware – síntese do VHDL

Download para o FPGA

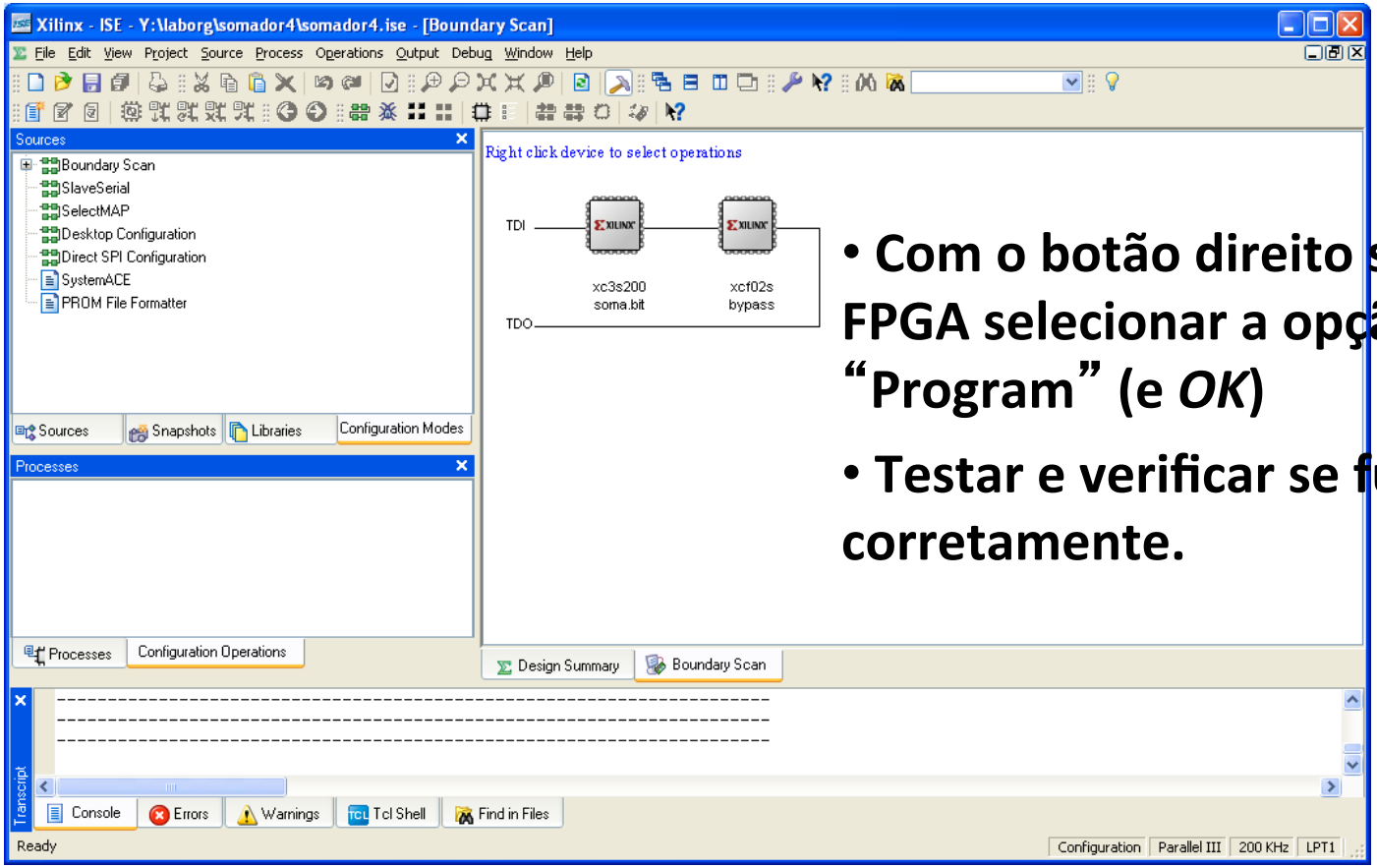
- Voltar para o ISE, ligar a placar, conectar os cabos e selecionar a ferramenta “Configure Device (iMPACT)



Prototipação em hardware – síntese do VHDL

Download para o FPGA

A janela da ferramenta *Impact* será aberta, usar a opção padrão JTAG, selecionar o arquivo *soma.bit* para o FPGA Spartan (FPGA - xv3s200), e selecionar *bypass* para a memória flash.



- **Com o botão direito sobre o FPGA selecionar a opção "Program" (e *OK*)**
- **Testar e verificar se funciona corretamente.**

Prototipação em hardware – síntese do VHDL

EXERCÍCIO

- Notar que esse somador possui "vai-um", mas não é apresentado na placa
- Realizar as alterações necessárias no fonte VHDL, UCF, e configurações das ferramentas de desenvolvimento de forma a acender o LED4 sempre que ocorrer vai-um no bit mais significativo da soma (C_{out})

FPGAs: Limitações

Limitações

- Do ponto de vista do desenvolvedor, aplicações para FPGAs devem ser tratadas como projetos de hardware
- Habilidade para programar em HDLs é compulsório
- Fluxo de projeto é bastante semelhante ao fluxo de projeto de ASIC
- Projetistas, normalmente, possuem formação em engenharia
- Maior dificuldade para "programar" e depurar
- Desenvolvimento requer mais tempo do que desenvolvimento de software
- Usar regra 90/10
  - Uso de processador convencional para execução de 90% do código que representa 10% do tempo de execução
  - Uso de FPGA para execução de 10% do código que representa 90% do tempo de execução

Limitações

- As vantagens da tecnologia são inúmeras:
  - Facilidade para updates
  - Eficiência arquitetural
  - Eficiência no uso de recursos – apenas os algoritmos necessários são configurados no hardware em um determinado momento
  - Maior velocidade com menor consumo de energia quando comparado com software
  - Flexibilidade quando comparado com ASICs
  - Custo reduzido para produções de tamanho pequeno para médio
  - Facilidade para inferir paralelismo

Limitações

- Limitações em FPGAs são muitas vezes relacionadas à aplicação alvo
- Por exemplo, uso de FPGAs em sistemas de controle é dificultado por duas razões:
  - Ferramentas de desenvolvimento são voltadas para projetistas de sistemas digitais. VHDL e outras são muito diferentes das ferramentas tradicionais usadas em sistemas de controle
  - A adição de implementações FPGA em modelos de simulação de sistemas é uma tarefa complexa, podendo ser inviável em algumas situações
- Linguagens baseadas em C a partir das quais ferramentas conseguem extrair hardware (convertendo C para HDL), podem ser uma solução para o primeiro problema, reduzindo essa limitação