II Escola Regional de Mato Grosso do Sul - II ERI-MS

Ponta Porã - MS - 24 a 26 de agosto de 2011

Minicurso: Implementação de Circuitos Digitais Utilizando Computação Reconfigurável

Ricardo Santos, André Diniz, Marcel Grassi

Universidade Federal de Mato Grosso do Sul

Laboratório de Sistemas Computacionais de Alto Desempenho

http://lscad.facom.ufms.br

Campo Grande - MS - Brasil

Tópicos 1o. dia

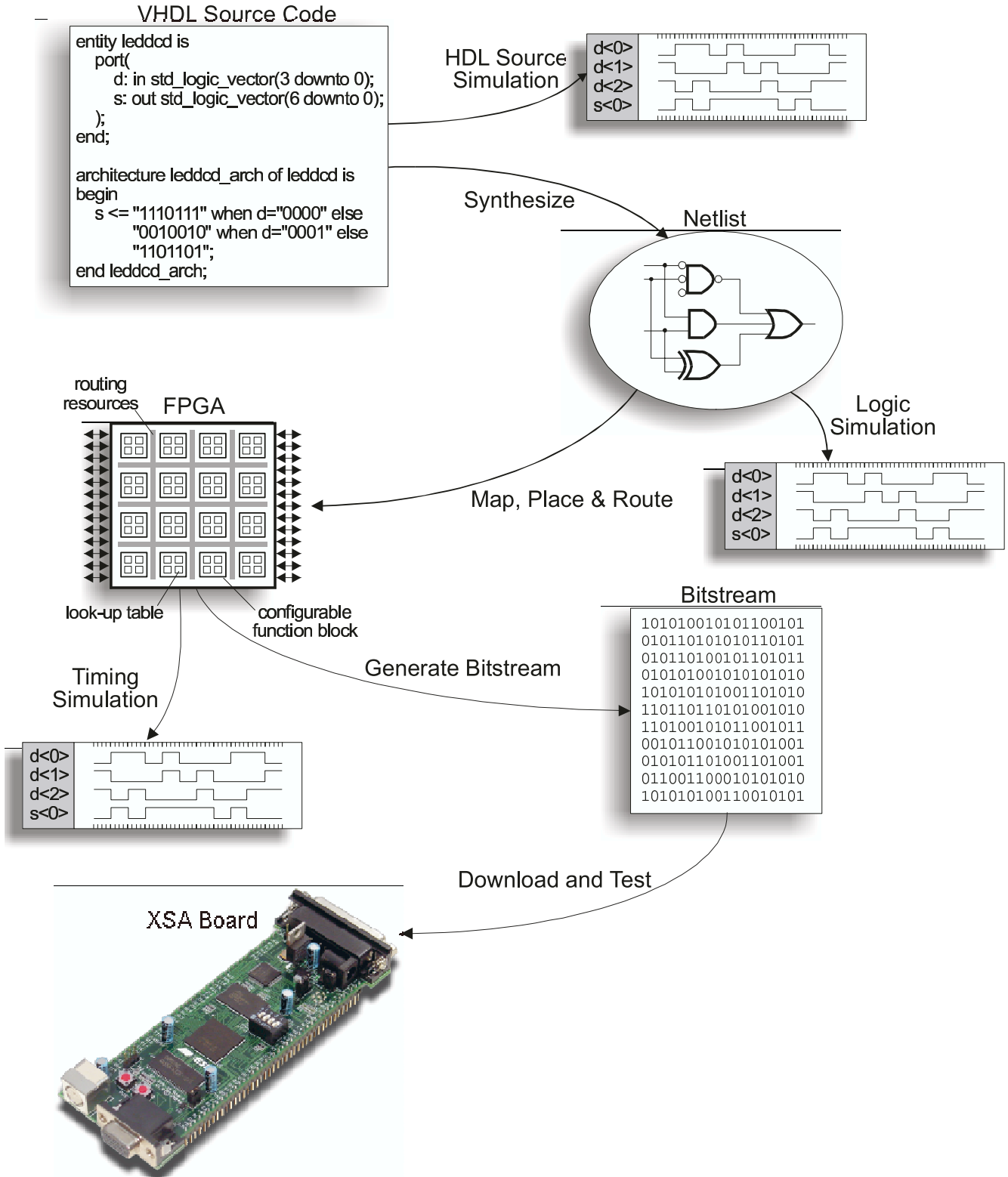
- **Projeto de Circuitos Digitais**
- **Introdução à linguagem VHDL**
- **Prática com Prototipação**

Projeto de Circuitos Digitais

- Metodologia de projeto:
  - Descrição do sistema em um nível de abstração, usando HDL
  - Uso de ferramentas automáticas para projeto, síntese e teste do circuito:
    - **Ferramentas CAD** (Computer Aided Design)
  - Descrição do hardware deve ser independente da tecnologia a ser usada na implementação:
    - Custom chip, FPGA, outro PLD, ...

Projeto de Circuitos Digitais

- Etapas do processo de desenvolvimento:
  - **Design Entry**: fornecer para ferramenta CAD uma representação do circuito
    - Esquemáticos
    - HDL (Hardware Description Language)
      - Linguagem de programação para modelar a operação do hardware
      - Exemplos: VHDL, Verilog, SystemC, AHDL
  - **Simulação**: teste do circuito gerado
  - **Síntese**: ferramenta CAD gera circuito lógico a partir do modelo representado e otimiza o circuito
  - **Prototipação**: implementação do circuito em **dispositivo de lógica programável**


VHDL Source Code
entity leddcd is
port(
d: in std_logic_vector(3 downto 0);
s: out std_logic_vector(6 downto 0);
);
end;
architecture leddcd_arch of leddcd is
begin
s <= "1110111" when d="0000" else
"0010010" when d="0001" else
"1101101";
end leddcd_arch;
HDL Source Simulation
d<0>
d<1>
d<2>
s<0>
Synthesize
Netlist
routing resources
FPGA
look-up table
configurable function block
Map, Place & Route
Logic Simulation
Timing Simulation
Generate Bitstream
Bitstream
101010010101100101
010110101010110101
010110100101101011
010101001010101010
101010101001101010
110110110101001010
110100101011001011
001011001010101001
010101101001101001
011001100010101010
101010100110010101
Download and Test
XSA Board

Vantagens do Uso de HDLs e Ferramentas CAD

- Aumento da produtividade, diminuindo o ciclo de desenvolvimento
- Redução dos custos de projeto
- Reusabilidade
- Facilidade em introduzir alterações nos projetos
- Exploração de alternativas de arquiteturas
- Exploração de alternativas tecnológicas
- Geração de circuitos testáveis automaticamente

Exemplos de Empresas

- Exemplos de empresas desenvolvedoras de ferramentas CAD:
  - Mentor Graphics (www.mentor.com)
  - Synopsys (www.synopsys.com)
- Exemplos de fabricantes de circuitos (FPGAs, etc) (que também desenvolvem ferramentas CAD):
  - Altera (www.altera.com)
  - Xilinx (www.xilinx.com)

VHDL

- Linguagem de descrição de hardware
  (**Hardware Description Language** – HDL)
- VHDL: VHSIC HDL
  - VHSIC: Very High Speed Integrated Circuit
- Permite:
  - Representar e modelar circuitos digitais
  - Fornecer o circuito para ferramenta CAD
    (para simulação, síntese e implementação)
- Projetada inicialmente pelo Departamento de Defesa dos EUA
- Padrão IEEE (Institute of Electrical and Electronics Engineers):
  - VHDL-87, VHDL-93, VHDL-2001

VHDL não é linguagem de programação de SW!

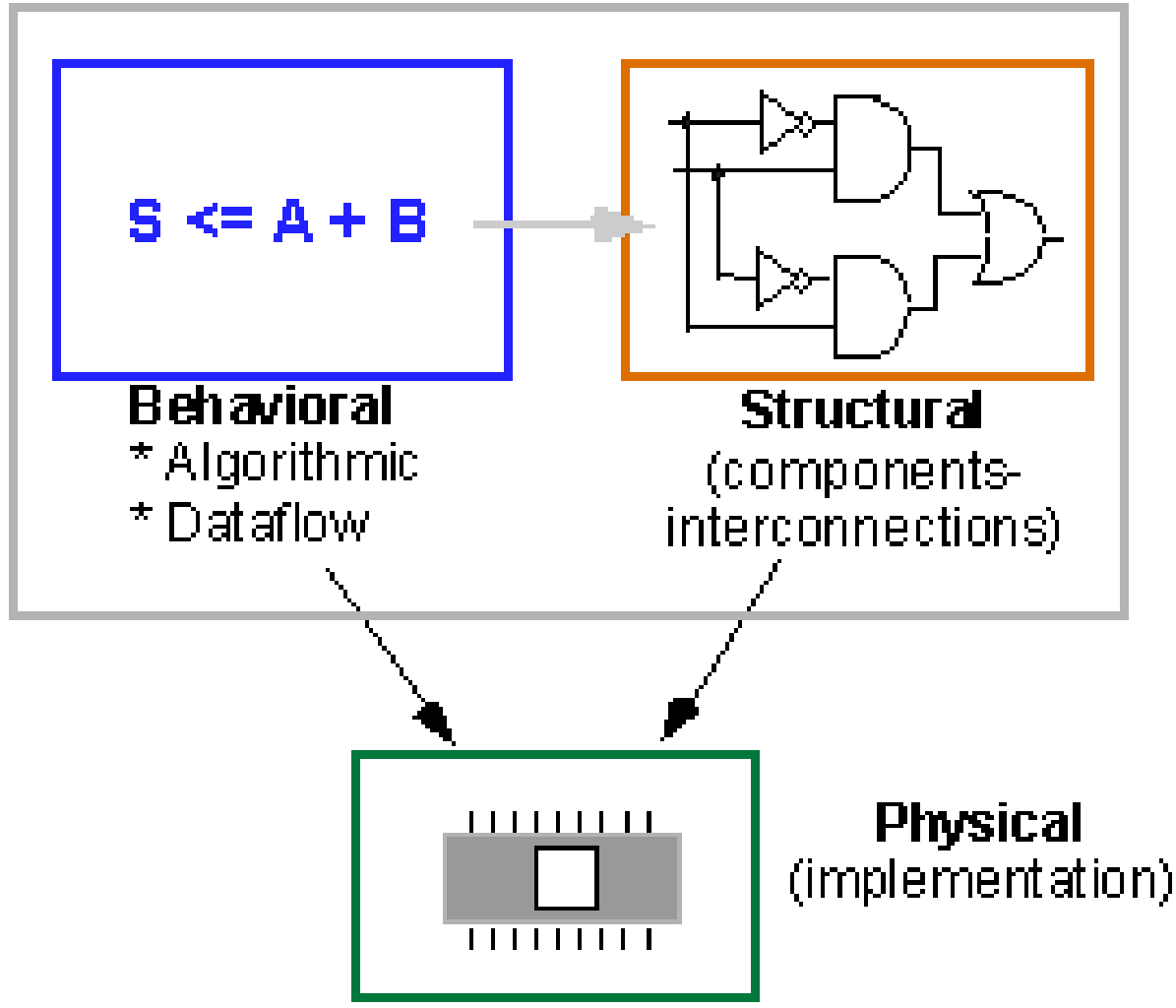
- Programa VHDL modela sistema de hardware digital
- Comandos do programa correspondem a portas lógicas do circuito
- Programa possui sinais e variáveis
- Concorrência:
  - Linguagem inerentemente paralela
  - Comandos são executados (calculados) em paralelo, assim que novos valores para os sinais de entrada chegam
- Temporização:
  - Permite especificação de temporizações, representando atrasos das portas lógicas
- Estruturação:
  - Permite descrever sistema de hardware como conjunto de componentes (subsistemas de HW) interconectados

VHDL

- Erro comum ao escrever modelo de circuito em VHDL:
  - Usar muitas variáveis e laços, como em linguagem de programação de software
- Problemas:
  - Ferramentas de síntese podem não ser capazes de gerar circuito
  - Circuito gerado pode ser incorreto
  - Circuito gerado pode ser ineficiente

Níveis de Abstração

- Sistema digital pode ser representado em diferentes níveis de abstração:
  - Comportamental (behavioral)
    - Algorítmico
    - Dataflow
  - Estrutural



Linguagem VHDL

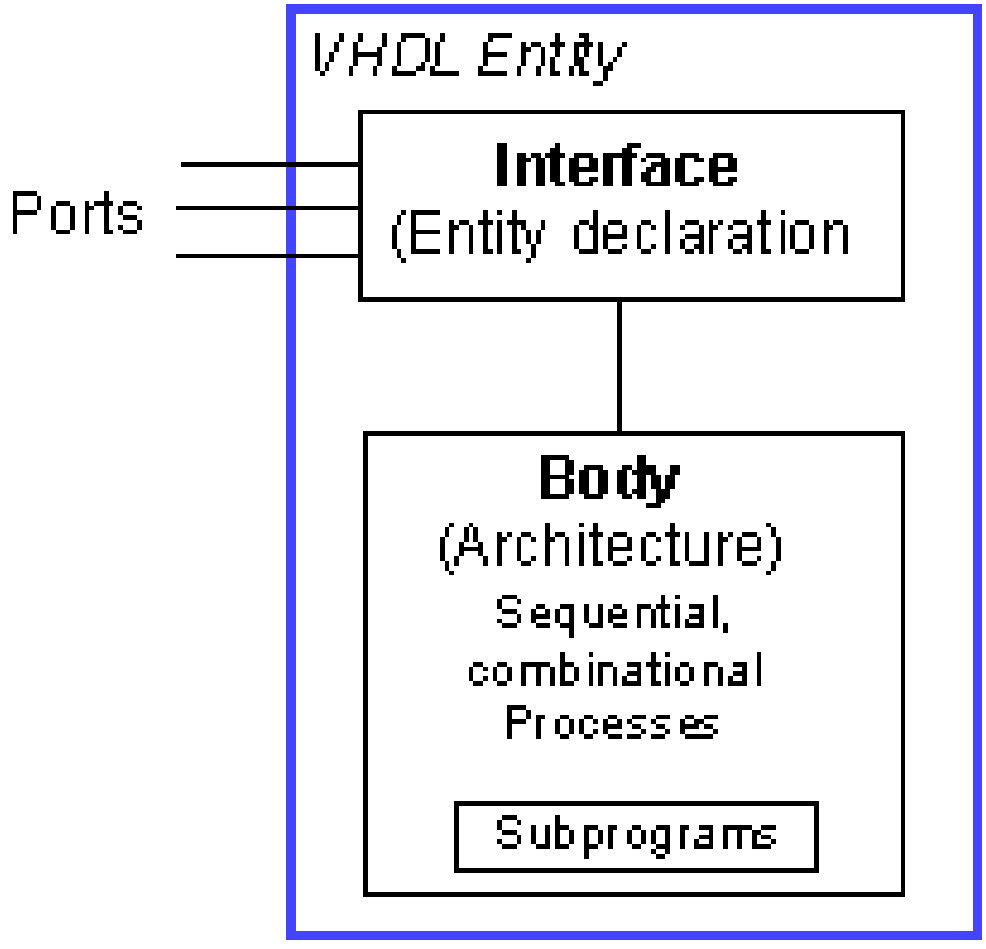
- Modelo de um circuito digital em VHDL:
  - Arquivo texto com extensão .vhd
- Linguagem permite descrever circuito digital:
  - No nível comportamental
  - No nível estrutural
  - Combinando representações comportamentais e estruturais
- Linguagem fortemente tipada:
  - É necessário declarar tipo de todos os objetos de dados (sinais, variáveis, constantes, ...)
  - Operandos e operadores devem ser do mesmo tipo (ou de tipos “compatíveis”)

VHDL: Elementos Léxicos

- Espaços em branco extras e quebras de linha ignorados
- Comentários: `--`
- Palavras reservadas
- Identificadores
- Case-insensitive:
  - **`entity`**, **`Entity`**, `carryout`, `CarryOut`
- Terminador de comandos: ;

Entity

- Arquivo <nome da entidade>.vhd possui definição da entidade
- Cada arquivo possui definição de apenas uma entidade
- Uma entidade pode possuir várias arquiteturas



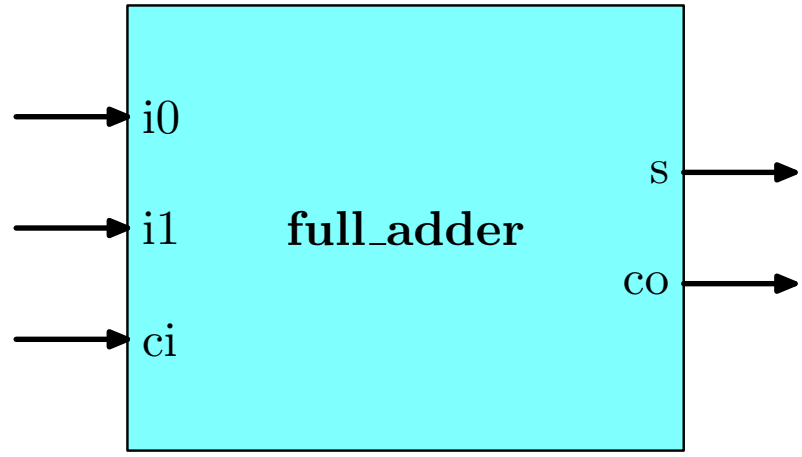
Declaração de Entidade

```
entity <nome da entidade> is

   -- Declaracao de genericos

   generic
   (
      < lista de genericos >
   ) ;

   -- Declaracao de sinais de entrada e saida
   port
   (
      < lista de sinais > : <modo> <tipo> ;
      ...
      < lista de sinais > : <modo> <tipo>
   ) ;

end <nome da entidade> ;
```

Declaração de Entidade: Portas

- **Portas**: Sinais de entrada e saída da entidade
  - **Lista de sinais**: lista de nomes de sinais separados por vírgulas
  - **Modo**: especifica direção do fluxo do sinal
    - **`in`**: sinal de entrada da entidade
    - **`out`**: sinal de saída da entidade
      - Valor do sinal não pode ser usado dentro da entidade
      - Sinal só pode aparecer do lado esquerdo de atribuição de sinal
    - **`inout`**: sinal de entrada e saída da entidade
      - Valor do sinal pode ser usado dentro da entidade
      - Sinal pode aparecer do lado esquerdo e direito de atribuição de sinal
    - **`buffer`**: sinal de saída da entidade
      - Valor do sinal pode ser usado dentro da entidade (após atribuição)
      - Sinal pode aparecer do lado esquerdo e direito de atribuição de sinal
  - **Tipo**: tipo de dado
  - Porta pode ter **valor default**, válido apenas se não há atribuição ao sinal

Exemplo 1: Declaração da Entidade `full_adder`




```
-- Entidade Somador completo

entity full_adder is

   -- Sinais de entrada e saida
   port
   (
      i0, i1 : in  bit ; -- Dados a serem somados
      ci     : in  bit ; -- Carry-in
      s      : out bit ; -- Resultado da soma
      co     : out bit   -- Carry-out
   ) ;

end full_adder ;
```

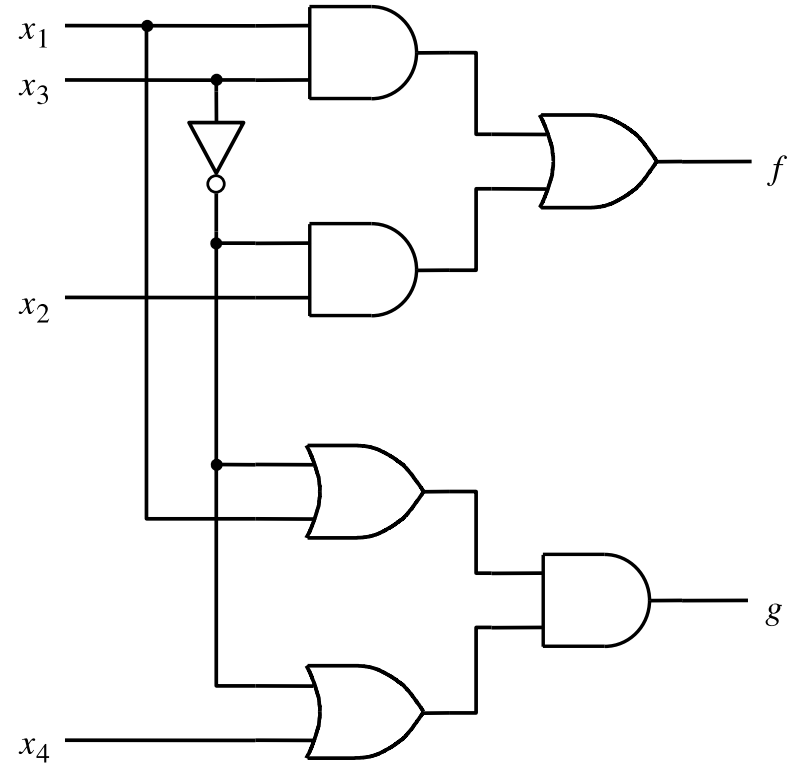
Corpo da Entidade: Architecture

```
architecture < nome da arquitetura > of < nome da entidade > is

    -- Declaracoes

    < Declaracoes de constantes >
    < Declaracoes de tipos >
    < Declaracoes de sinais >
    < Declaracoes de componentes >
    < Especificacoes de atributos >
    < Declaracoes de funcoes >
    < Declaracoes de procedimentos >

begin
    -- Comandos concorrentes

    < Comandos de instanciacao de sub-entidades >
    < Comandos de atribuicao de sinais >
    < Comandos process >
    < Comandos generate >

end < nome da arquitetura > ;
```

Exemplo: Entidade `full_adder` e Arquitetura

```
-- Entidade Somador completo

entity full_adder is

   -- Sinais de entrada e saida
   port
   (
      i0, i1 : in  bit ; -- Dados a serem somados
      ci     : in  bit ; -- Carry-in
      s, co  : out bit   -- Resultado da soma e Carry-out
   ) ;

end full_adder ;

-- Arquitetura comportamental (dataflow) do somador completo

architecture behavioral of full_adder is
begin
   --  2 atribuicoes de sinal concorrentes

   s <= i0 xor i1 xor ci ;

   co <= (i0 and i1) or (i0 and ci) or (i1 and ci) ;

end behavioral ;
```

Exemplo: Entidade e Arquitetura




```
entity example is
    port
    (
        x1, x2, x3, x4 : in  bit ;
        f, g           : out bit
    ) ;
end example ;

architecture LogicFunc of example is
begin
    f <=  (x1 and x3) or (not x3 and x2) ;
    g <=  (not x3 or x1) and (not x3 or x4) ;
end LogicFunc ;
```

Operadores Lógicos

Operador	Tipo		
	Operando esquerdo	Operado direito	Resultado
and	boolean ou		
or	bit ou		
nand	std_ulogic ou	mesmo tipo do	mesmo tipo
nor	std_logic ou	operando esquerdo	dos operandos
xor	array 1-D		
xnor	destes tipos		

Operadores Relacionais

<table>
<tr><th rowspan="2">Operador</th><th colspan="3">Tipo</th></tr>
<tr><th>Operando esquerdo</th><th>Operado direito</th><th>Resultado</th></tr>
<tr><td>=</td><td rowspan="2">qualquer tipo</td><td rowspan="6">mesmo tipo do operando esquerdo</td><td rowspan="6">boolean</td></tr>
<tr><td>/=</td></tr>
<tr><td><</td><td rowspan="4">tipo escalar ou array de tipo discreto</td></tr>
<tr><td><=</td></tr>
<tr><td>></td></tr>
<tr><td>>=</td></tr>
</table>

Operadores de Deslocamento

Operador	Tipo		
	Operando esquerdo	Operado direito	Resultado
`sll`	array 1-D de boolean ou bit	tipo inteiro	mesmo tipo do operando esquerdo
`srl`			
`sla`			
`sra`			
`rol`			
`ror`			

- Descrição:
  - **`sll`**, **`srl`**, **`sla`**, **`sra`**: Shift left/right logical/arithmetic
  - **`rol`**, **`ror`**: Rotate left/right

Operadores de Adição, Subtração, etc

<table>
<tr><th rowspan="2">Operador</th><th colspan="3">Tipo</th></tr>
<tr><th>Operando esquerdo</th><th>Operado direito</th><th>Resultado</th></tr>
<tr><td>+</td><td rowspan="2">tipo numérico</td><td rowspan="3">mesmo tipo do
operando esquerdo</td><td rowspan="2">mesmo tipo
dos operandos</td></tr>
<tr><td>-</td></tr>
<tr><td>&</td><td>array ou
tipo do elemento do array</td><td>mesmo
tipo do array</td></tr>
</table>

- Descrição:
  - **&**: Concatenação

Operadores Unários

Operador	Tipo	
	Operando	Resultado
`+`	tipo numérico	mesmo tipo do operando
`-`		
`abs`		
`not`	boolean ou bit ou std_ulogic ou std_logic	

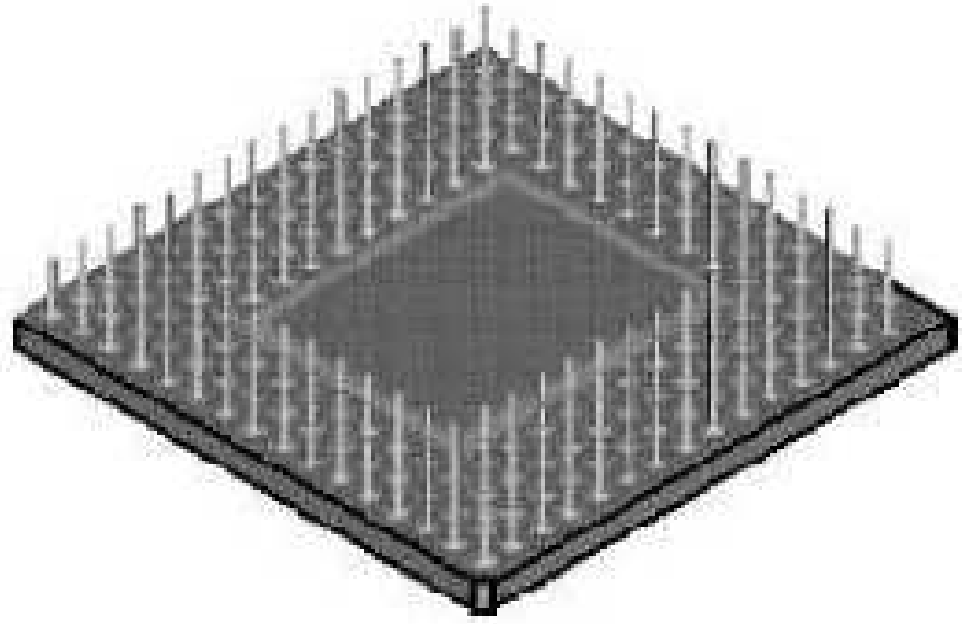
Operadores de Multiplicação, Divisão, etc

Operador	Tipo		
	Operando esquerdo	Operado direito	Resultado
*	tipo inteiro ou ponto flutuante	mesmo tipo op. esquerdo	mesmo tipo operandos
	tipo físico	tipo inteiro ou ponto flutuante	mesmo tipo op. esquerdo
	tipo inteiro ou ponto flutuante	tipo físico	mesmo tipo op. direito
/	tipo inteiro ou ponto flutuante	mesmo tipo op. esquerdo	mesmo tipo operandos
	tipo físico	tipo inteiro ou ponto flutuante	mesmo tipo op. esquerdo
	tipo físico	mesmo tipo op. esquerdo	integer
mod	tipo inteiro	mesmo tipo do operando esquerdo	mesmo tipo dos operandos
rem			
**	tipo inteiro ou ponto flutuante	tipo inteiro	mesmo tipo op. esquerdo

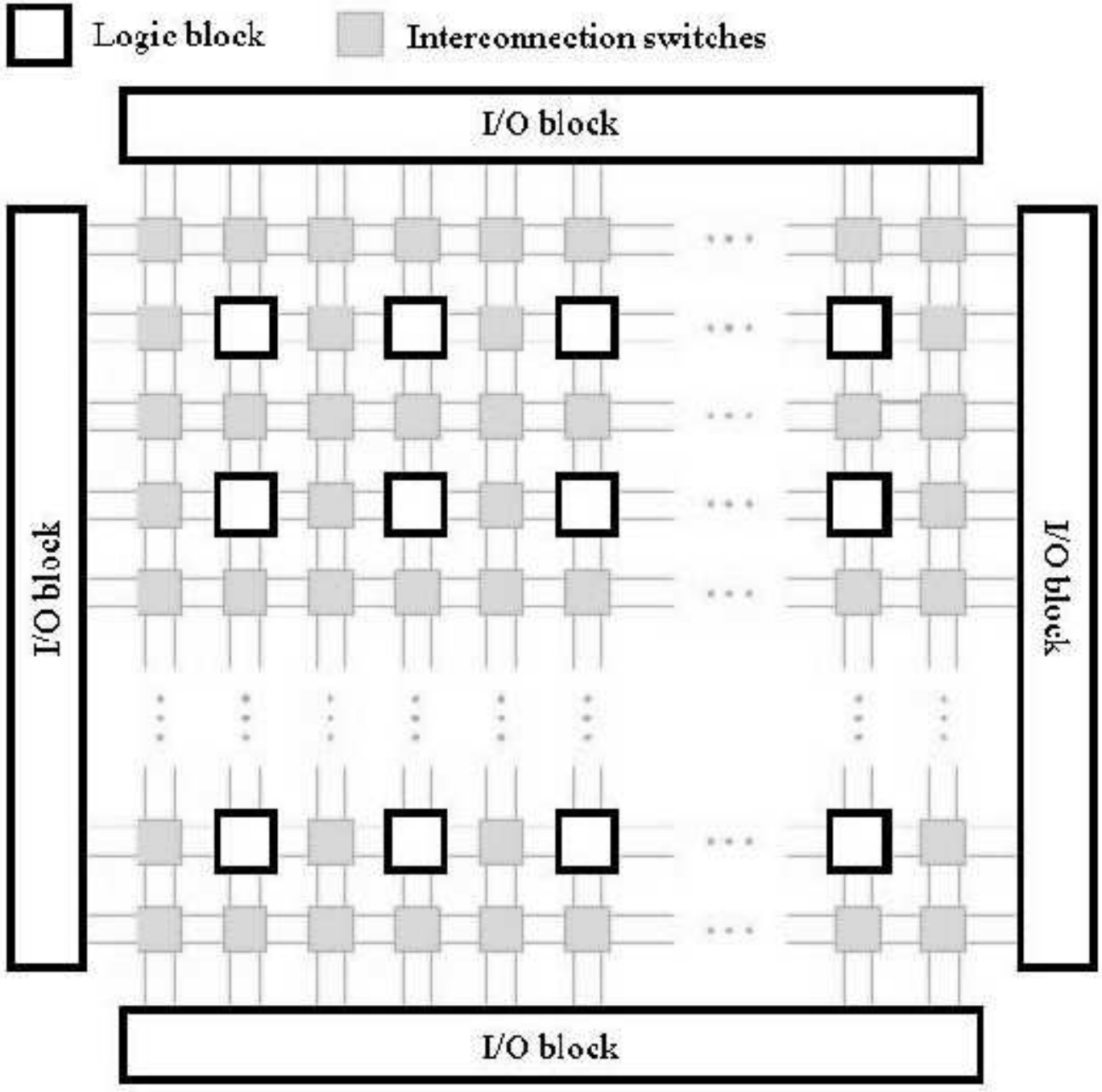
- Descrição: **: Exponenciação

Field-Programmable Gate Array (FPGA)

- Contém matriz de blocos lógicos configuráveis
  (**Configurable Logic Blocks** – **CLBs**)
- Contém também blocos de E/S, interconexões e switches configuráveis
- Blocos de E/S conectam com pinos do chip
- Chip de uma FPGA:



Estrutura Geral de FPGA

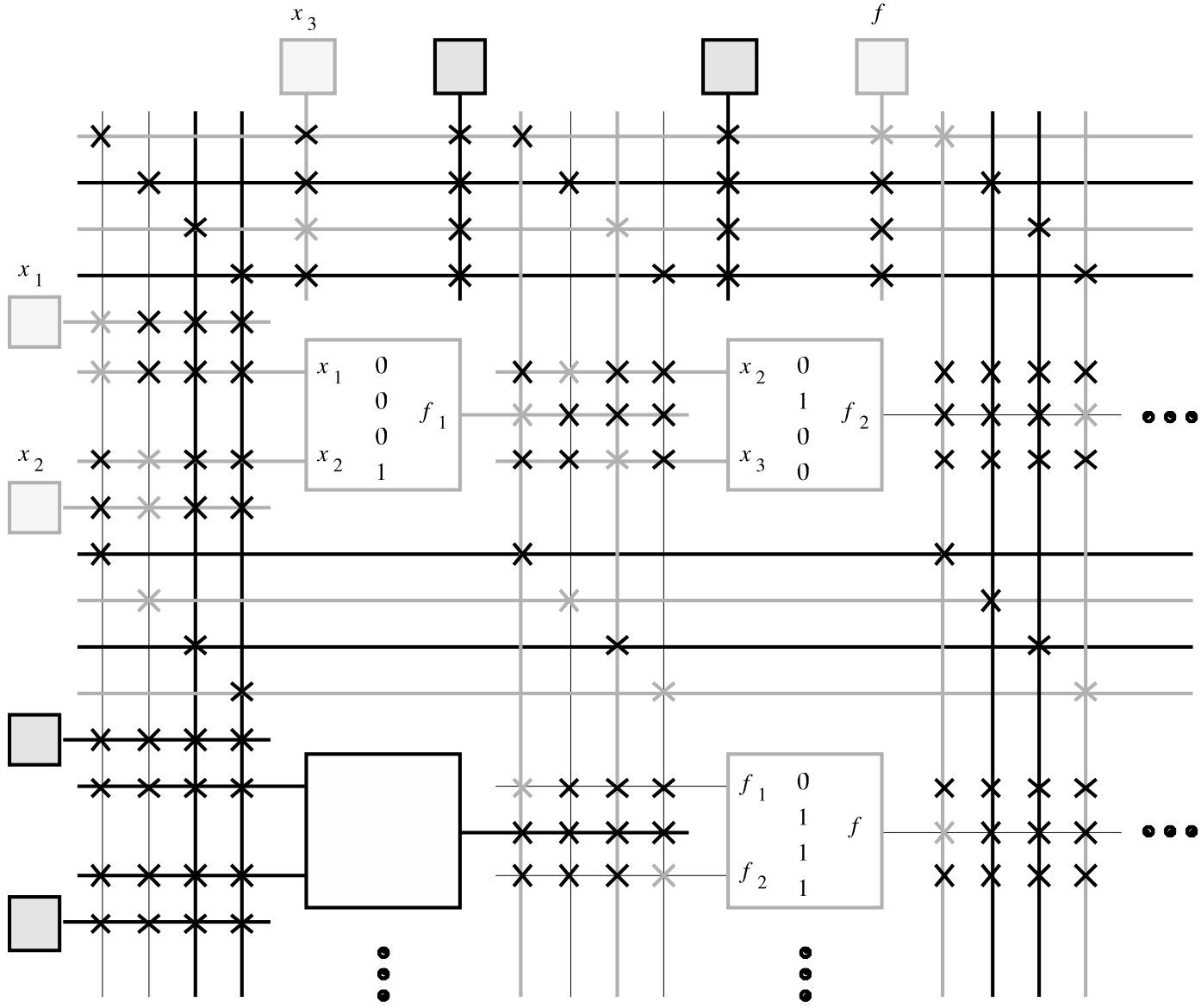


FPGA

- Diferentes FPGAs $\Rightarrow$ Diferentes tipos de CLBs
- Cada CLB possui poucos sinais de entrada e de saída
- CLB mais comum: Lookup Table
- **Lookup Table** (LUT):
  - Contém células de armazenamento (pequena memória)
  - Cada célula armazena bit 0 ou 1
  - Usada para implementar uma pequena função lógica:
    - Sinais de entrada da função lógica são usados para endereçar uma célula da memória
    - Valor armazenado na célula é produzido como saída da função lógica
- CLB também pode possuir circuitos adicionais conectados à sua saída:
  - Flip-flop, multiplexador, ...

Seção de FPGA Programada

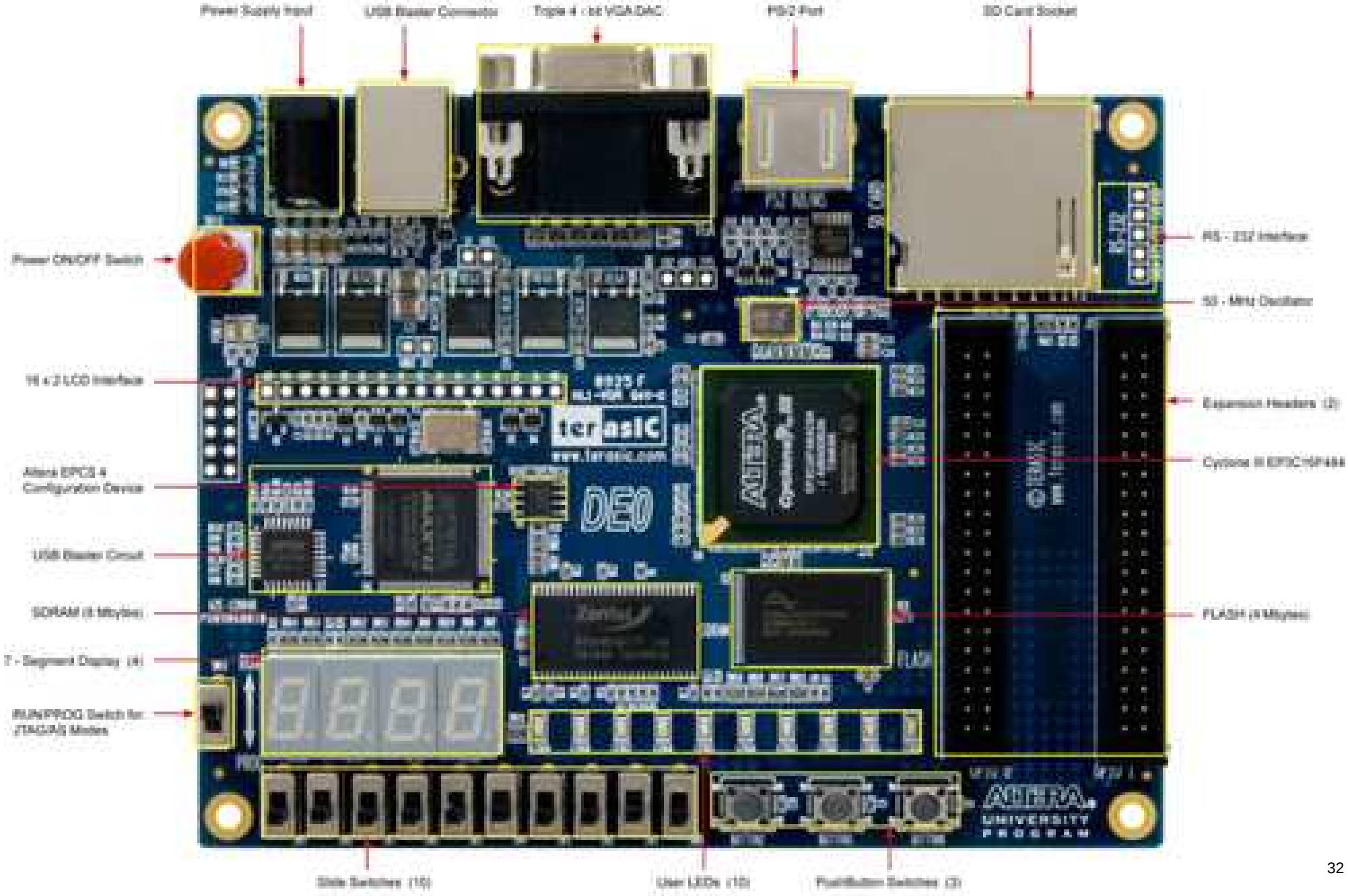
- Exemplo: $f_1 = x_1 \cdot x_2$ $f_2 = \bar{x_2} \cdot x_3$ $f = f_1 + f_2$



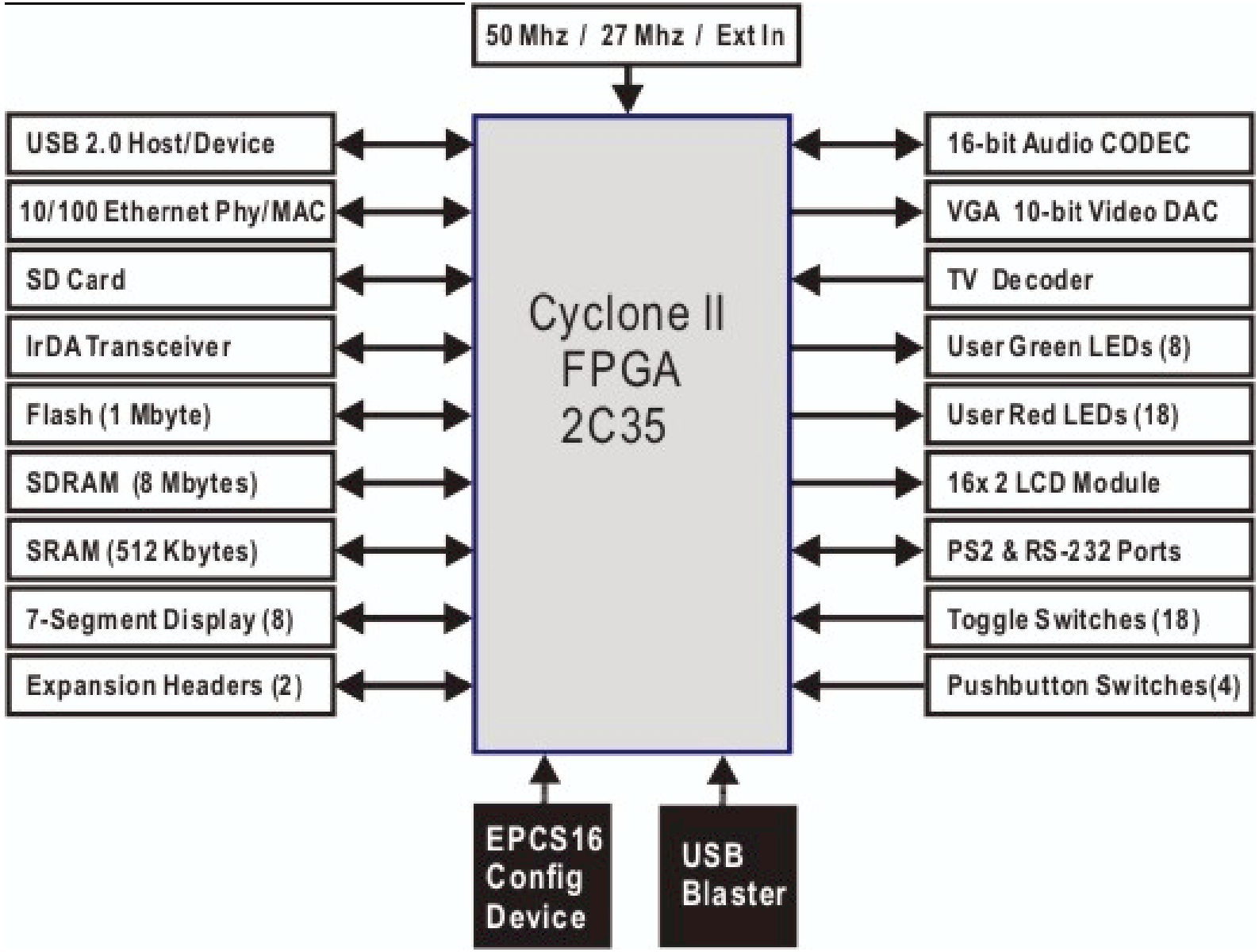
Placa DE0 (Development and Education Board) da Altera

- Componentes de hardware:
  - FPGA Altera Cyclone II
  - Memória RAM para configuração
  - Memória flash
  - Botões *pushbutton*
  - Botões *toggle switch*
  - Leds
  - Gerador de clock
  - Interface para E/S de áudio
  - Interface para E/S de vídeo
  - Interface ethernet
  - Interface USB
  - Interfaces RS-232 e PS/2
  - etc
- Ferramentas de software:
  - Quartus II (ferramenta CAD)
  - ModelSim (simulador HDL)
  - etc

Placa DE0


Power Supply Input
USB Blaster Connector
Triple 4 - bit VGA DAC
PS/2 Port
SD Card Socket
Power ON/OFF Switch
RS - 232 Interface
50 - MHz Oscillator
16 x 2 LCD Interface
Expansion Headers (2)
Altera EPCS 4
Configuration Device
Cyclone III EP3C16F484
USB Blaster Circuit
SDRAM (8 Mbytes)
FLASH (4 Mbytes)
7 - Segment Display (4)
RUN/PROG Switch for
JTAG/AS Modes
Slide Switches (10)
User LEDs (10)
Pushbutton Switches (2)
terasIC
www.terasic.com
DE0
ALTERA
UNIVERSITY
PROGRAM

Organização da Placa DE0/DE2



Tópicos 2o. dia

- **Modelagem de Circuitos usando VHDL**
- **Conceitos de Simulação**
- **Prática com ModelSim**

Modelagem em VHDL

```
------------------------- Interface -------------------------
entity < nome da entidade > is

   -- Declaracao de genericos

   -- Declaracao de sinais de entrada e saida
   port (
      ...
   ) ;
end ;

------------------------ Implementacao ------------------------
architecture < nome da arquitetura > of < nome da entidade > is

   -- Declaracoes de constantes, tipos, sinais, componentes,
   -- atributos, funcoes, procedimentos
begin
   -- Comandos concorrentes
   < Comandos de atribuicao de sinais >          -- Comportamental
   < Comandos process >                          -- Comportamental
   < Comandos de instanciacao de componentes >   -- Estrutural
   < Comandos generate >
end ;
```

Comando Concorrente: Instanciação de Entidade

- Modelagem estrutural
- Mapeamento dos sinais de entrada e saída da entidade instanciada com sinais da entidade externa:
  - Comando **`port map`** :
    - Associação posicional
    - Associação por nome

Exemplo: Associação Posicional e por Nome

```
entity controller is
   port ( rd, wr : in bit ;
          ready : out bit ) ;
end ;
architecture fpld of controller is
begin
   ...
end ;
-------------------------------------------------------------
entity circuit is
   ...
end ;
architecture test of circuit is
   signal ctl_rd, ctl_wr, ctl_ready : bit ;
begin
   ...
   ctl : entity work.controller(fpld)
     -- Port map com associacao posicional
     port map ( ctl_rd, ctl_wr, ctl_ready ) ;
     -- Port map equivalente com associacao por nome
     port map ( wr => ctl_wr, ready => ctl_ready, rd => ctl_rd ) ;
end ;
```

Exemplo: Flip-flop

- Na transição de subida de `clk`, armazena valor de `d`

```
entity flip_flop is
   port (
      clk, d : in  bit ;
      q      : out bit
   ) ;
end ;

architecture behav of flip_flop is
begin
   process (clk)
   begin
      if clk = '1' then
         q <= d ;
      end if;
   end process ;
end ;
```

Exemplo: Registrador de 4 bits (formado por 4 flip-flops)

```
entity reg4 is
    port (
        clock : in bit ;
        d_in  : in bit_vector(0 to 3) ;
        q_out : out bit_vector(0 to 3)
    ) ;
end ;

architecture struct of reg4 is
begin
    bit0 : entity work.flip_flop(behav)
           port map (clock, d_in(0), q_out(0)) ;
    bit1 : entity work.flip_flop(behav)
           port map (clock, d_in(1), q_out(1)) ;
    bit2 : entity work.flip_flop(behav)
           port map (clock, d_in(2), q_out(2)) ;
    bit3 : entity work.flip_flop(behav)
           port map (clock, d_in(3), q_out(3)) ;
end ;
```

Simulação

- Simulação:
  - Inicialização
  - Executa repetidamente ciclo de simulação
- **Inicialização da simulação**:
  - Cada sinal (portas de entrada e saída e sinais intermediários) é inicializado (com valor inicial definido na sua declaração ou valor inicial default)
  - Tempo da simulação é iniciado com 0
  - Todos os comandos concorrentes são ativados (executados uma vez) e suspensos
  - Para cada processo:
    - Processo é ativado
    - Variáveis do processo são inicializadas
    - Comandos do processo são executados a partir do **`begin`**
    - Quando executa **`wait`**, processo é suspenso
  - Ao final da inicialização, todos os processos estão suspensos

Simulação

- **Ciclo de simulação**:
  - Avança tempo da simulação para menor instante para o qual há transação escalonada
  - Aplica todas as transações escalonadas para esse instante
  - Transações podem causar ocorrência de eventos em sinais
  - Comandos concorrentes “ativados” pelos eventos são executados uma vez e suspensos
  - Processos sensíveis aos sinais dos eventos são reativados:
    - Enquanto processo está suspenso, valor das variáveis não é perdido
    - Processo continua executando comandos seqüencialmente, a partir do comando após o **`wait`**
    - Ao chegar ao **`end`**, processo continua execução a partir do **`begin`**
    - Quando executa **`wait`**, processo é suspenso
- Quando todos os processos estiverem suspensos, repete ciclo de simulação

Simulação

- **Fim da simulação**:
  - Se todos os processos estão suspensos e não há nenhuma transação escalonada

Tópicos 3o. dia

- **Conceito de Máquina de Estados**

- **Prática com Projeto de Alarme Residencial**

Máquina de Estados (Autômato)

- Circuito seqüencial
- Usado em controladores e unidades de controle
- Possui:
  - Sinais de entrada
  - Sinais de saída
  - Estado atual
- A cada passo, determina próximo estado e valores dos sinais de saída, de acordo com estado atual e valores dos sinais de entrada

Máquina de Estados

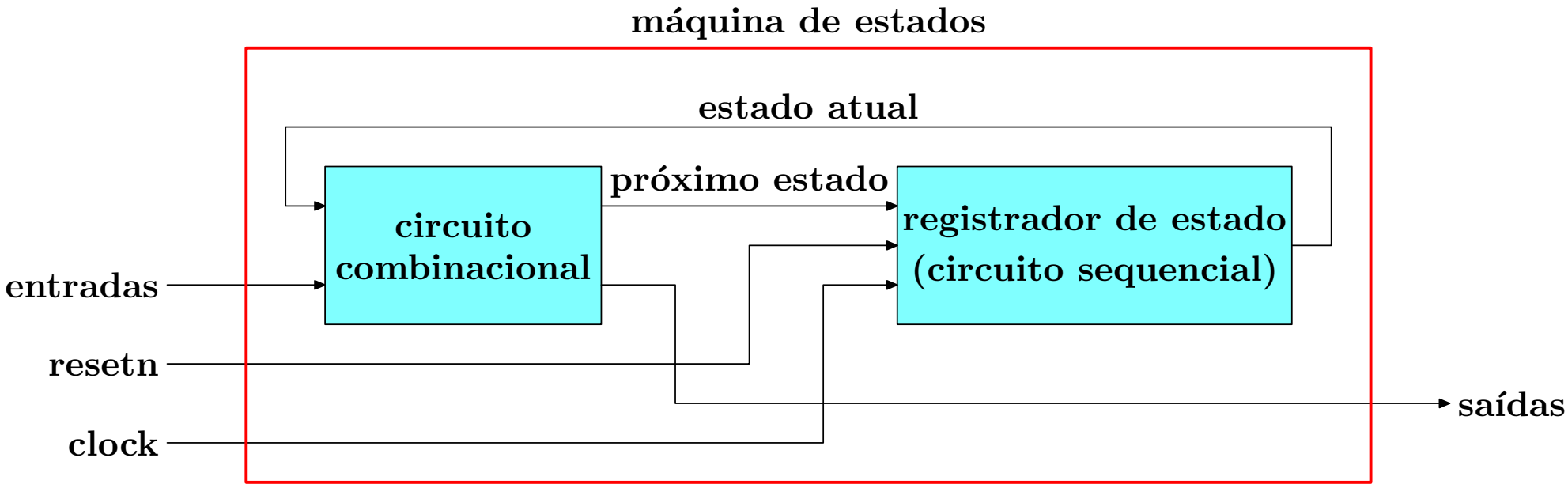
- **Máquina de Moore**:
  - Sinais de saída dependem apenas do estado atual
  - Próximo estado depende do estado atual e dos sinais de entrada
- **Máquina de Mealy**:
  - Sinais de saída dependem do estado atual e dos sinais de entrada
  - Próximo estado depende do estado atual e dos sinais de entrada

Máquina de Estados

- Reset: estado atual recebe estado inicial
  - Low-active ou high-active reset
  - Reset assíncrono ou síncrono
- Transição de estados síncrona:
  - Estado atual só é atualizado na subida do clock
- Sinais de saída:
  - Assíncronos:
    - Podem mudar quando estado atual ou sinais de entrada mudam
  - Síncronos:
    - Podem mudar apenas na subida do clock
    - *Registered output*

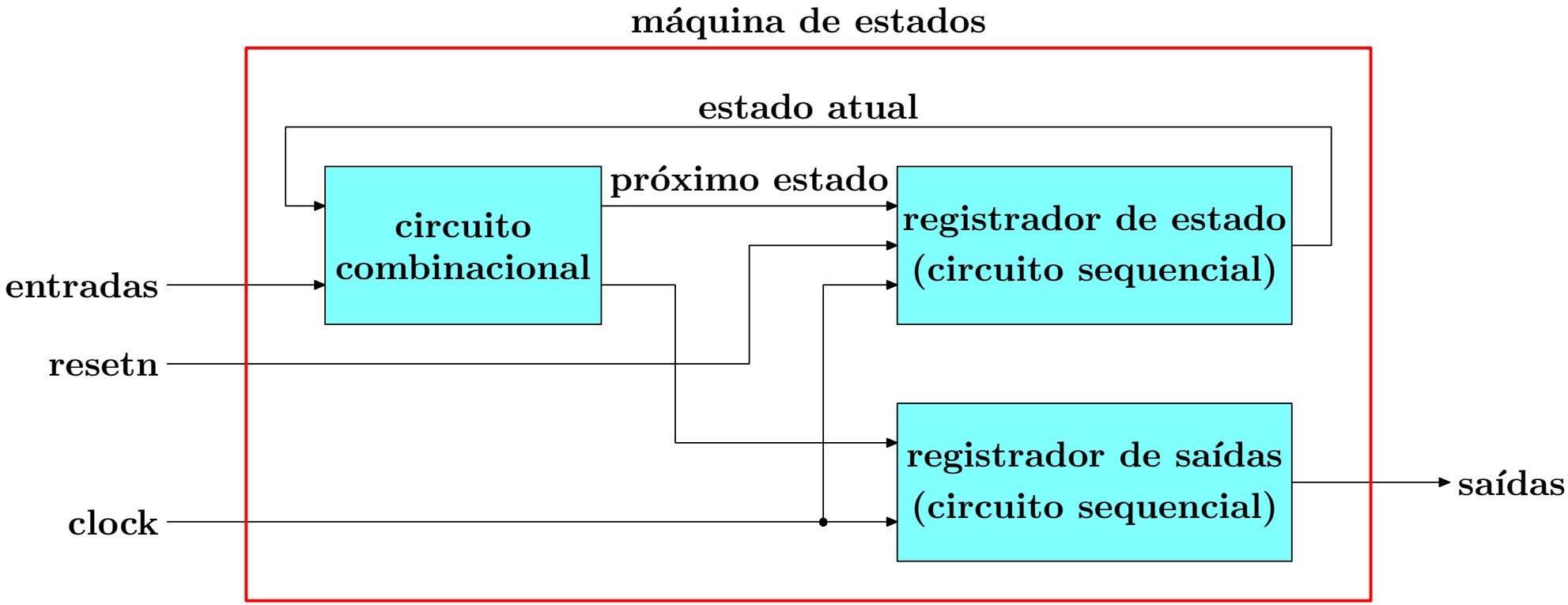
Estrutura de uma Máquina de Estados

- Com **saídas assíncronas**:



Estrutura de uma Máquina de Estados

- Com **saídas síncronas**:



Estrutura de uma Máquina de Moore (c/ Saídas Assíncronas)

```
entity moore is
   port (
      clock, resetn : in  bit ;
      <entradas>    : in  <tipo_entrada> ;
      <saidas>      : out <tipo_saida> ) ;
end ;

architecture behavior of moore is

   type state_type is <tipo_enumerado ou tipo_range> ;
   signal current_state, next_state : state_type ;
begin
   -- Circuito sequencial: registrador de estado
   process ( resetn, clock ) -- Sensivel a reset e clock
   begin
      if resetn = '0' then -- Reset low-active, assincrono
         current_state <= <estado_inicial> ;

      elsif (clock'event and clock = '1') then
         current_state <= next_state ; -- Transicao de estado sincrona
      end if ;
   end process ;
```

Estrutura de uma Máquina de Moore (Saídas Assíncronas) (cont.)

```
   -- Circuito combinacional: sensivel a entradas e estado atual
   process ( <entradas>, current_state )
   begin
      case current_state is
         when <estado_1> =>
            if <entradas> = ... then
               next_state <= <estado ...> ;
            elsif <entradas> = ... then
               next_state <= <estado ...> ;
            else
               next_state <= <estado ...> ;
            end if ;
            <saidas> <= ... ; -- Saidas assincronas
         when <estado_2> =>
            ...
         ... -- Deve cobrir todos os estados
      end case ;
   end process ;
end ;
```

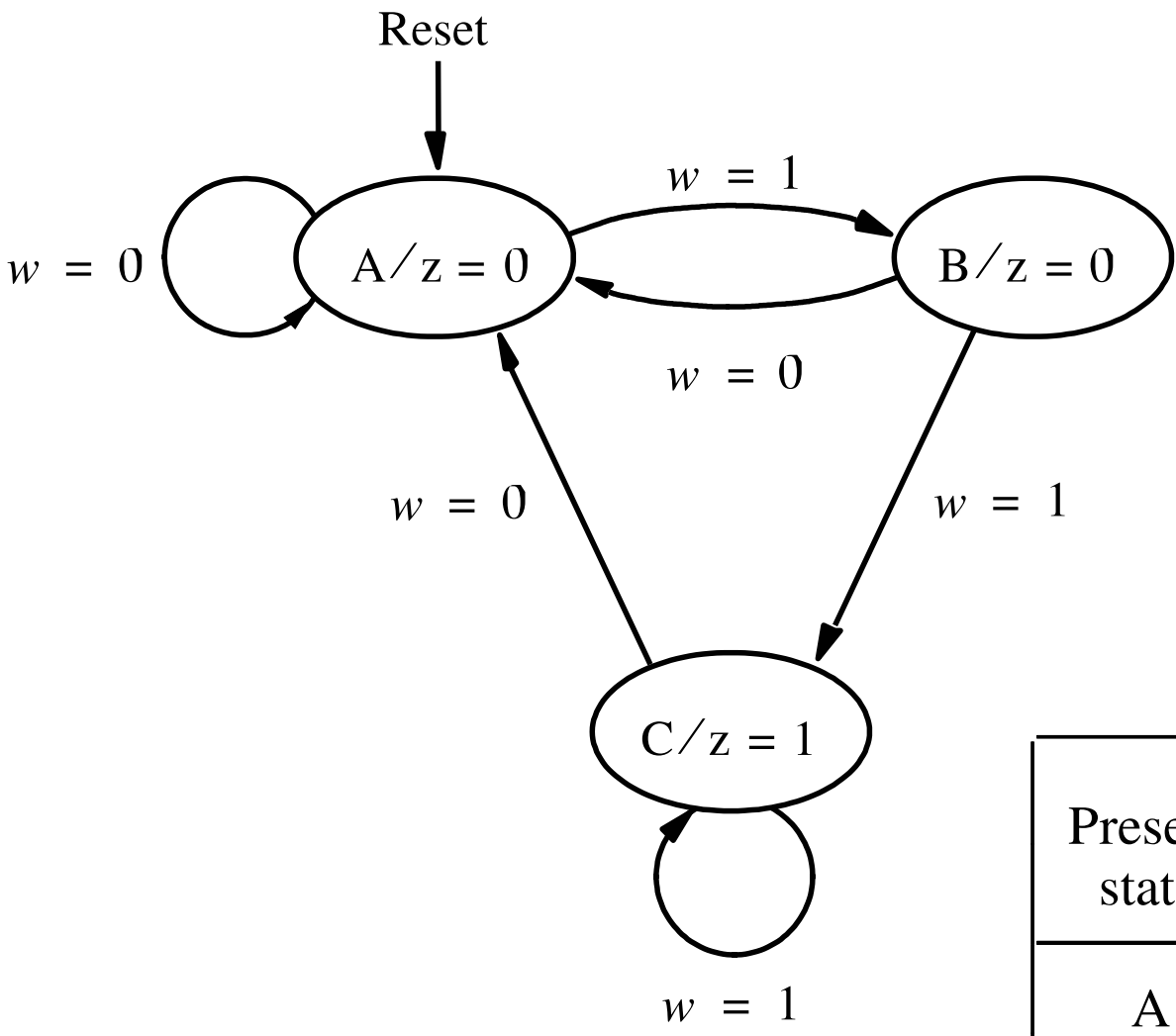
Estrutura de uma Máquina de Mealy (c/ Saídas Assíncronas)

```
entity mealy is
   port (
      clock, resetn : in  bit ;
      <entradas>    : in  <tipo_entrada> ;
      <saidas>      : out <tipo_saida> ) ;
end ;

architecture behavior of mealy is

   type state_type is <tipo_enumerado ou tipo_range> ;
   signal current_state, next_state : state_type ;
begin
   -- Circuito sequencial: registrador de estado
   process ( resetn, clock ) -- Sensivel a reset e clock
   begin
      if resetn = '0' then -- Reset low-active, assincrono
         current_state <= <estado_inicial> ;

      elsif (clock'event and clock = '1') then
         current_state <= next_state ; -- Transicao de estado sincrona
      end if ;
   end process ;
```

Estrutura de uma Máquina de Mealy (Saídas Assíncronas) (cont.)

```
   -- Circuito combinacional: sensivel a entradas e estado atual
   process ( <entradas>, current_state )
   begin
      case current_state is
         when <estado_1> =>
            if <entradas> = ... then
               next_state <= <estado ...> ;
               <saidas> <= ... ; -- Saidas assincronas
            elsif <entradas> = ... then
               next_state <= <estado ...> ;
               <saidas> <= ... ; -- Saidas assincronas
            else
               next_state <= <estado ...> ;
               <saidas> <= ... ; -- Saidas assincronas
            end if ;
         when <estado_2> =>
            ...
         ... -- Deve cobrir todos os estados
      end case ;
   end process ;
end ;
```

Exemplo 1: Máquina de Moore



Present state	Next state		Output
	$w = 0$	$w = 1$	z
A	A	B	0
B	A	C	0
C	A	C	1

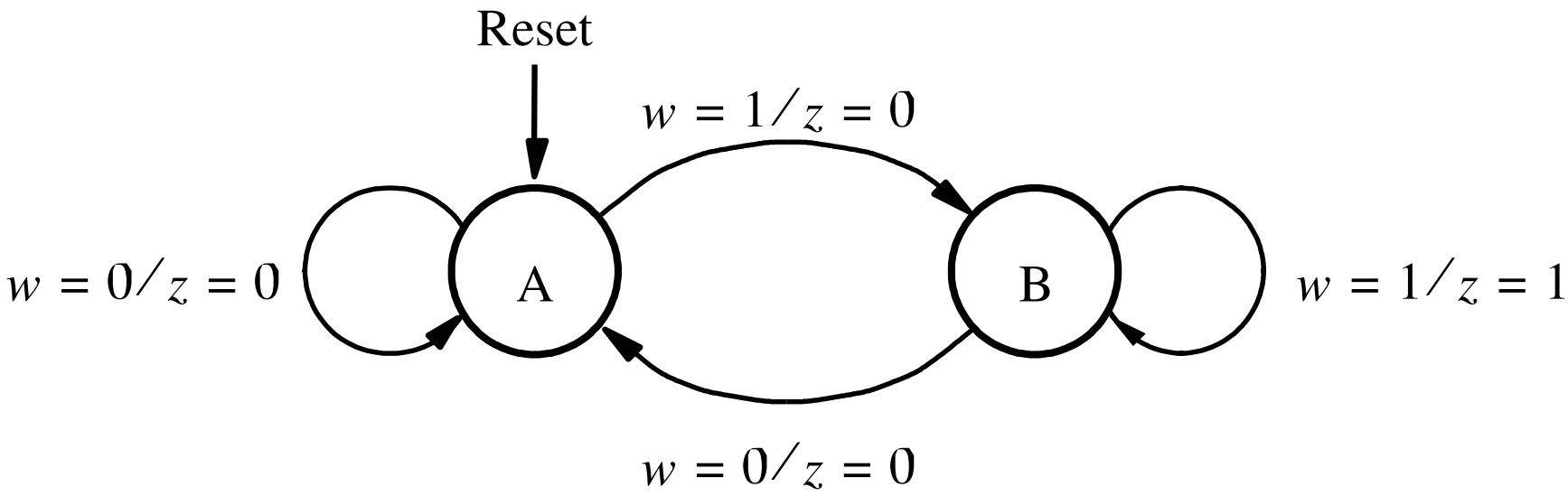
Exemplo 1: Máquina de Moore (c/ Saída Assíncrona)

```
entity moore is
    port (
        clock, resetn, w : in  bit ;
        z                : out bit ) ;
end ;

architecture behavior of moore is
    type state_type is (A, B, C) ;
    signal current_state, next_state : state_type ;
begin
    process ( resetn, clock )
    begin
        if resetn = '0' then -- Reset low-active, assincrono
            current_state <= A ;
        elsif (clock'event and clock = '1') then
            current_state <= next_state ;
        end if ;
    end process ;
```

Exemplo 1: Máquina de Moore (c/ Saída Assíncrona) (cont.)

```
   process ( w, current_state )
   begin
      case current_state is
         when A =>
            if w = '0' then
               next_state <= A ;
            else
               next_state <= B ;
            end if ;
            z <= '0' ; -- Saida assincrona
         when B =>
            if w = '0' then
               next_state <= A ;
            else
               next_state <= C ;
            end if ;
            z <= '0' ; -- Saida assincrona
         when C =>
            if w = '0' then
               next_state <= A ;
            else
               next_state <= C ;
            end if ;
            z <= '1' ; -- Saida assincrona
      end case ;
   end process ;
end ;
```

Exemplo 2: Máquina de Mealy



Present state	Next state		Output z	
	$w = 0$	$w = 1$	$w = 0$	$w = 1$
A	A	B	0	0
B	A	B	0	1

Exemplo 2: Máquina de Mealy (c/ Saída Assíncrona)

```
entity mealy is
    port (
        clock, resetn, w : in bit ;
        z                : out bit ) ;
end ;

architecture behavior of mealy is
    type state_type is (A, B) ;
    signal current_state, next_state : state_type ;
begin
    process ( resetn, clock )
    begin
        if resetn = '0' then -- Reset low-active, assincrono
            current_state <= A ;
        elsif (clock'event and clock = '1') then
            current_state <= next_state ;
        end if ;
    end process ;
```

Exemplo 2: Máquina de Mealy (c/ Saída Assíncrona) (cont.)

```
   process ( w, current_state )
   begin
      case current_state is
         when A =>
            if w = '0' then
               next_state <= A ;
            else
               next_state <= B ;
            end if ;
            z <= '0' ; -- Saida assincrona
         when B =>
            if w = '0' then
               next_state <= A ;
               z <= '0' ; -- Saida assincrona
            else
               next_state <= B ;
               z <= '1' ; -- Saida assincrona
            end if ;
      end case ;
   end process ;
end ;
```

Exemplo 2: Máquina de Mealy (c/ Saída Síncrona)

```
entity mealy is
   port (
      clock, resetn, w : in bit ;
      z                : out bit ) ;
end ;

architecture behavior of mealy is
   type state_type is (A, B) ;
   signal current_state, next_state : state_type ;
   signal temp_z : bit ;
begin
   process ( resetn, clock )
   begin
      if resetn = '0' then -- Reset low-active, assincrono
         current_state <= A ;
      elsif (clock'event and clock = '1') then
         current_state <= next_state ;
         z <= temp_z ; -- Saida sincrona
      end if ;
   end process ;
```

Exemplo 2: Máquina de Mealy (c/ Saída Síncrona) (cont.)

```
   process ( w, current_state )
   begin
      case current_state is
         when A =>
            if w = '0' then
               next_state <= A ;
            else
               next_state <= B ;
            end if ;
            temp_z <= '0' ;
         when B =>
            if w = '0' then
               next_state <= A ;
               temp_z <= '0' ;
            else
               next_state <= B ;
               temp_z <= '1' ;
            end if ;
      end case ;
   end process ;
end ;
```

Exemplo 3: Alarme Residencial

- Um circuito que controla um alarme residencial tem as seguintes características:
  - Possui uma chave com a qual o morador liga e desliga o alarme.
  - Possui um grupo de sensores espalhados pela residência, que detectam a presença de pessoas.
  - Possui uma sirene que toca em determinadas condições.
- O controle do alarme tem o seguinte funcionamento:
  - Inicialmente: o alarme está desligado.
  - Se o alarme está desligado:
    - Independente dos sensores: a sirene não deve tocar.
    - Se o morador liga a chave do alarme, arma o alarme para ligar dali a $\sim$30 segundos.
    - Neste intervalo:
      - Independente dos sensores: a sirene não deve tocar.
      - Se a chave é desligada, o alarme é desligado.

Exemplo 3: Alarme Residencial (cont.)

- O controle do alarme tem o seguinte funcionamento:
  - Se o alarme está ligado:
    - Se a chave é desligada, o alarme é desligado.
    - Se algum sensor detecta algo, aciona a sirene para tocar a $\sim$30 segundos.
    - Neste intervalo:
      - Mesmo que todos os sensores não detectem nada, continua com a sirene acionada.
      - Se a chave é desligada, o alarme é desligado.
  - Se a sirene está tocando:
    - Mesmo que todos os sensores não detectem nada: continua com a sirene tocando.
    - Se a chave é desligada: o alarme é desligado.

Módulo VHDL do Circuito de Controle de Alarme Residencial

- Descrição do Modelo:
  - Interface:
    - **Entidade: alarme**
    - **Sinais de entrada:**
      - sensores: bit_vector(1 to 3)
      - OBS: Se sensor i detecta presença de pessoa, sensores(i) = ’1’. Senão, sensores(i) = ’0’.
      - chave: bit
      - OBS: Quando o morador da casa liga a chave do alarme, chave passa para ’1’.
      - OBS: Quando o morador da casa desliga a chave do alarme, chave passa para ’0’.
      - clock: bit
    - **Sinais de saída:**
      - sirene: bit
      - OBS: Se sirene deve tocar, sirene = ’1’. Senão, sirene = ’0’.

Módulo VHDL do Circuito de Controle de Alarme Residencial

- Observações:
  - Máquina de Moore.
  - Transição de estados é síncrona.
  - Não há sinal de entrada de reset.
  - A saída deve ser assíncrona.
  - O sinal de clock tem um ciclo de 1 segundo.

Máquina de Estados - Uma Possível Solução