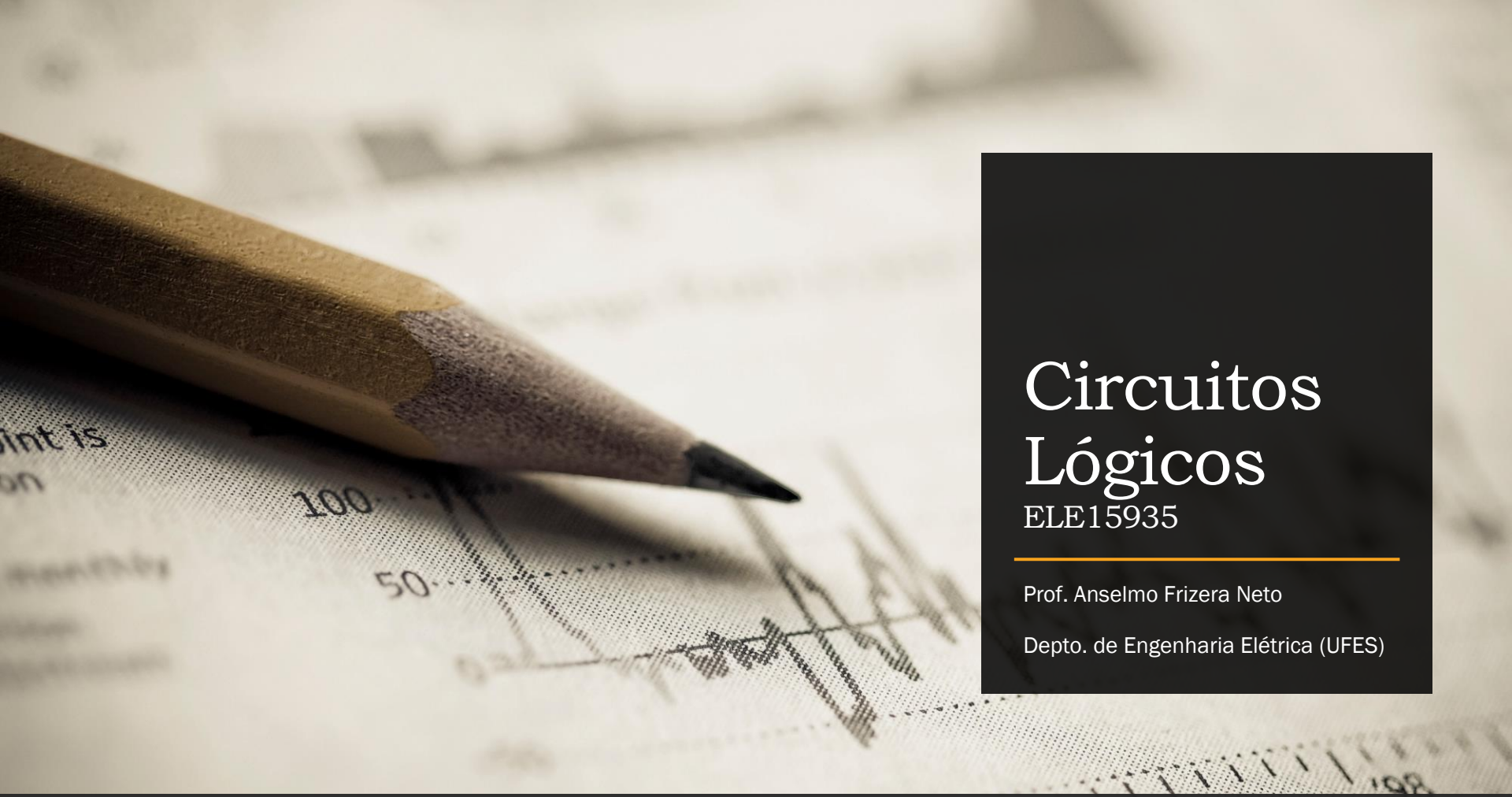


Circuitos Lógicos

ELE15935

Prof. Anselmo Frizera Neto

Depto. de Engenharia Elétrica (UFES)

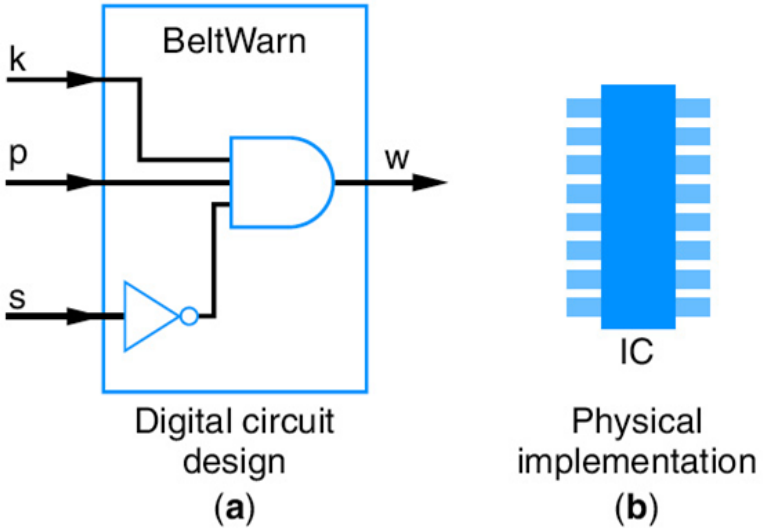
Lab. 1. Implementação Física, FPGA e VHDL

- Tópicos da aula de hoje:
  - Conceitos iniciais sobre implementação física
  - FPGA
  - Aula introdutória sobre VHDL
- Bibliografia:
  - F. Vahid, “Sistemas digitas” - Capítulos 7 e 9
  - P. Chu, “FPGA Prototyping by VHDL Examples”

Introdução

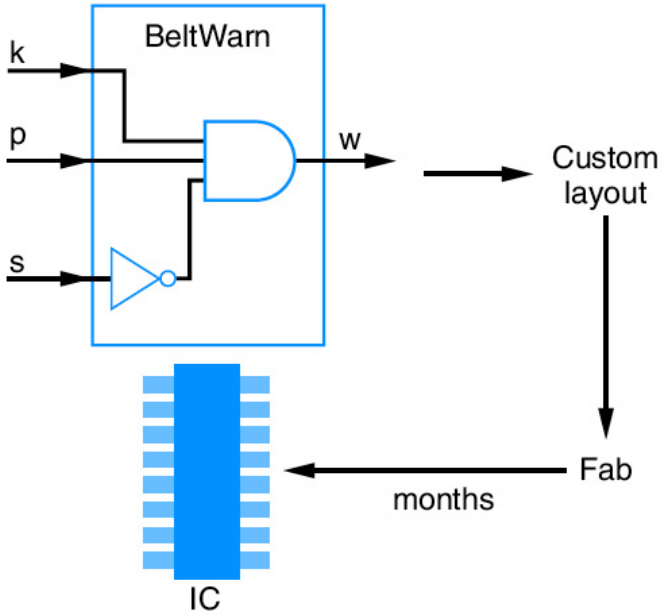
Devemos implementar o projeto em um dispositivo físico real

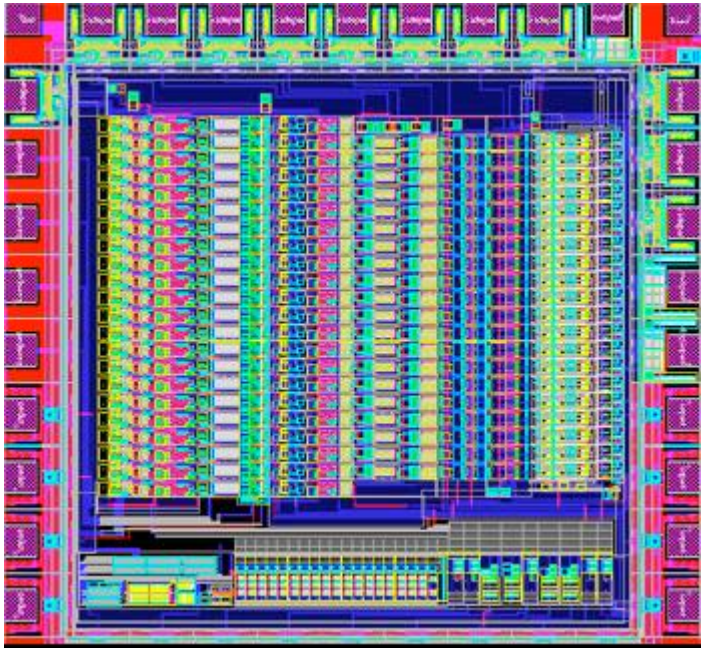
Como chegar de (a) para (b)?



Tecnologias de ICs Manufaturados

- Circuitos integrados totalmente customizados
  - Implementar as portas (na realidade, os transistores) do projeto de circuito digital desejado
  - Layout: Implementar o circuito de transistores e usar CAD para desenhar e orientar os elementos
- Fabricação: realização física do layout
  - Tarefa complexa e dispendiosa (Milhões de dólares)
  - Ocorrência de respin (refabricação devido a erros no projeto)



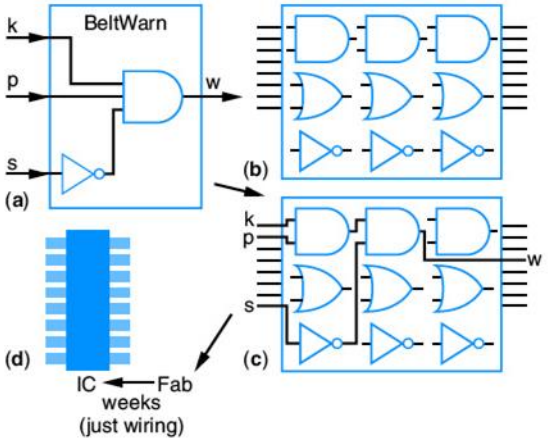


De acordo com um levantamento, apenas cerca de 10% dos circuitos digitais de 2002 foram implementados na forma de ICs customizados.

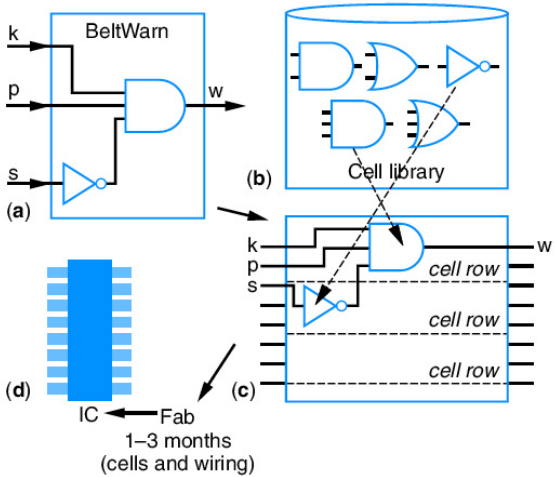
Somente se opta por esta solução em casos de volumes extremamente elevados!

Circuitos integrados – Tecnologias (algumas)

Gate array



Standard cell



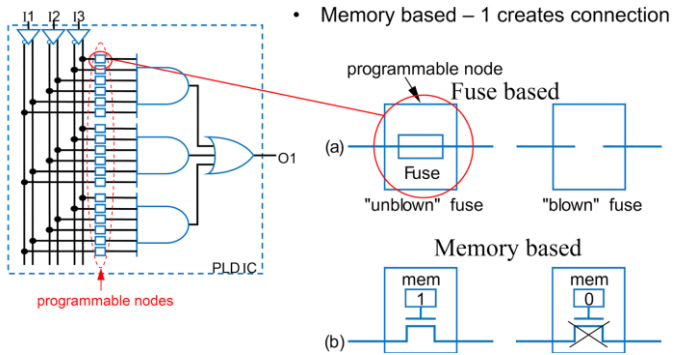
Cell array

- Cruzamento entre as tecnologias de gate array e standard cell
- As células padronizadas são posicionadas previamente no IC

Tecnologias de ICs programáveis

Um pouco de história...

- 1970. Conhecido como Programmable Logic Array – PLA
- AMD: "Programmable Array Logic" – "PAL" – somente ANDs programáveis (fusível)
- Lattice Semiconductor Corp: "Generic Array Logic – "GAL" (memórias)
- Estruturas múltiplas de PLD em 1 chip:
  - Complex PLDs (CPLD). As antigas = Simple PLDs (SPLD)
- Diferenças de SPLDs, CPLDs e FPGAs:
  - SPLD: dezenas ou centenas de portas, não volátil
  - CPLD: milhares de portas, não volátil
  - **FPGA: dezenas de milhares de portas, volátil (normalmente)**

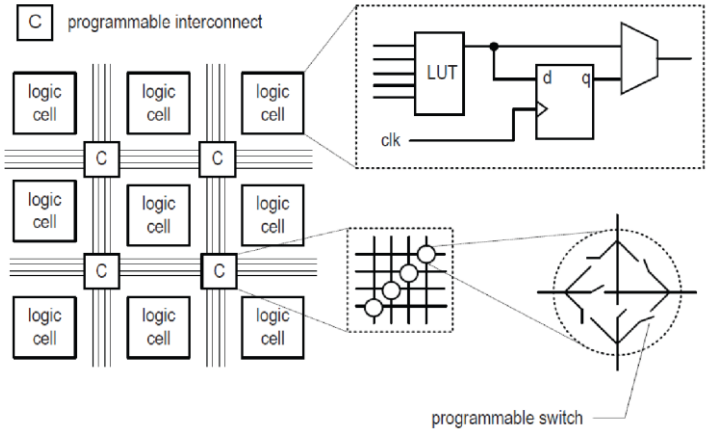


Field-Programmable Gate Array (FPGA)

- Arranjo ou matriz de portas programável em campo **(CUIDADO AQUI)**
- CI já contém todos os transistores e todos as conexões
- Programação:
  - Ocorre no campo (daí o nome), em oposição a uma planta de fabricação
  - Necessita de segundos ou minutos, no máximo
  - Histórico 1980 - FPGAs foram comercializadas como uma alternativa aos gate arrays (daí o nome)

Dentro da FPGA

- Matriz de células lógicas
- Estrutura geral de roteamento de interconexões que conecta as células lógicas
- As células lógicas e as interconexões podem ser customizadas (programadas) de acordo com uma aplicação específica



Dentro da FPGA

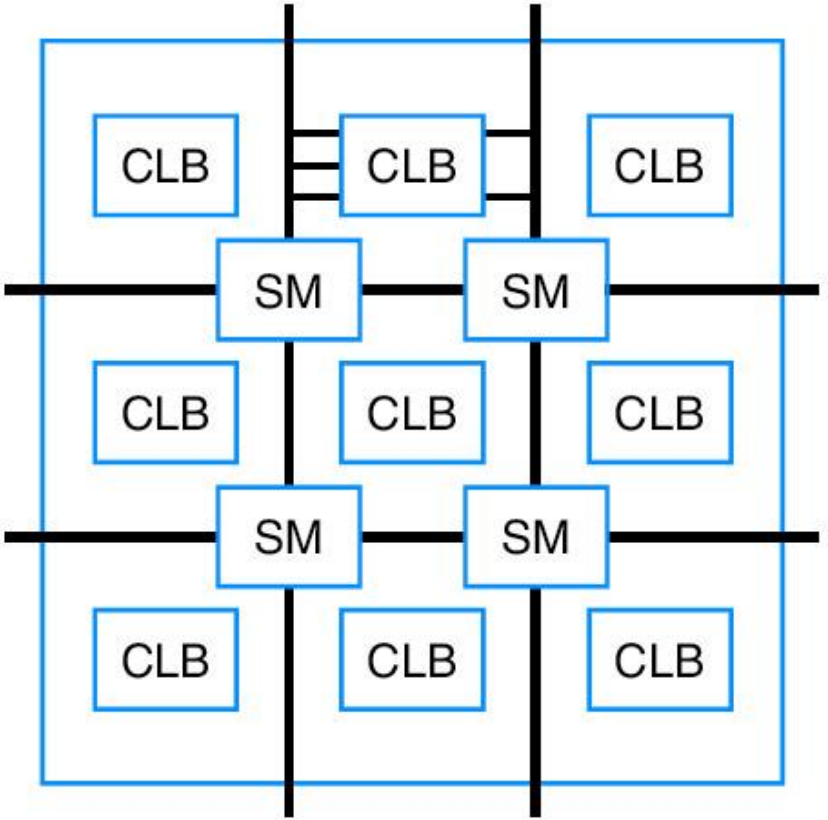
(outra representação)

Bloco Lógico Configurável (CLB)

- Tabela de lookup + Mem

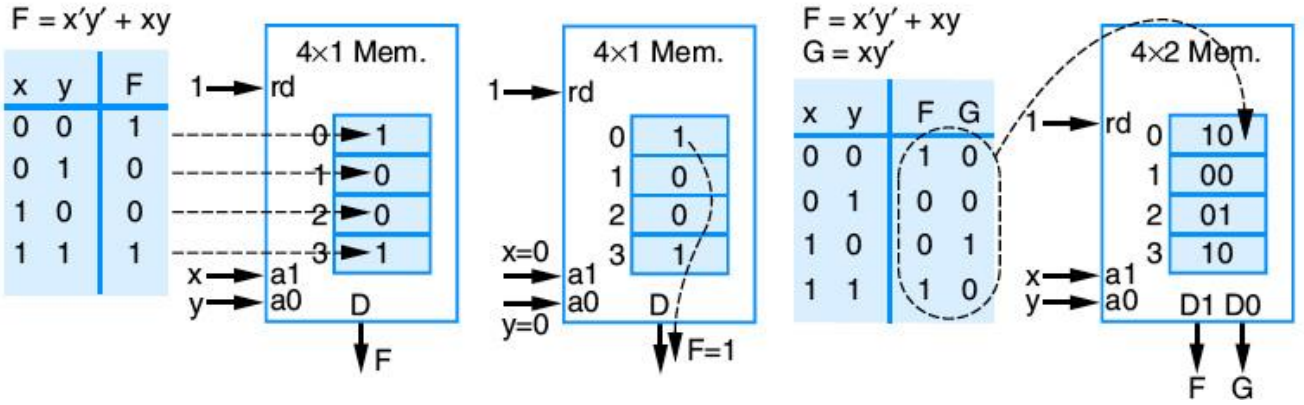
Matriz de Chaveamento (SM)

- Multiplexadores



Tabelas de consulta (ou lookup)

- Uma memória pode implementar lógica combinacional!

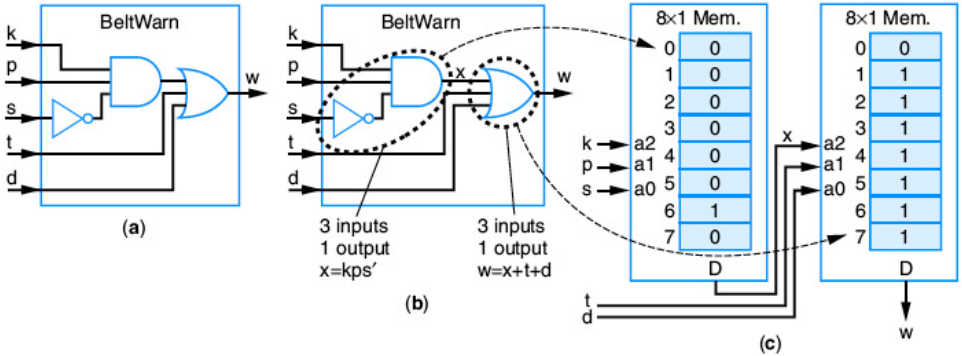


Partição de um circuito entre tabelas lookup

- O uso de uma memória para implementar uma função booleana não funciona bem com muitas entradas.
- 4 entradas - memória de 16 palavras
- 16 entradas - 64 Kpalavras
- Uma tabela-verdade não é uma representação eficiente de uma função booleana de muitas entradas

Observe aqui que a tabela-verdade poderia ser implementada diretamente em uma tabela de lookup

Partição de um circuito entre tabelas lookup



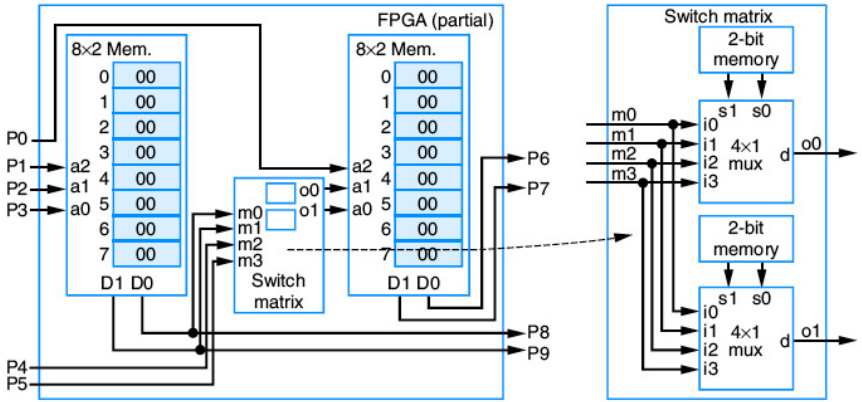
- 2 tabelas lookup tem um total de 8 + 8 = 16 palavras
- 1 tabela de lookup teriam 32 palavras

A partição de um circuito pode resultar em melhor eficiência!

Quem faz essa partição?

Interconexões programáveis

- FPGA: Chip pré-fabricado (incluindo os fios)
- Interconexões programáveis → matrizes de chaveamento



Novas migrações de FPGA para ASIC

Década de 2000:

- Nova tecnologia que cria um ASIC a partir de um projeto baseado em FPGA
- Implementação FPGA:
  - maior porte, mais cara e maior consumo
  - útil para se detectar e corrigir erros no circuito, assim como para demonstrar o produto final.
- Tradicionalmente, uma implementação ASIC não usa nenhuma informação a respeito da implementação FPGA

Novas migrações de FPGA para ASIC

- Problemas ASIC:
  - Tarefa difícil
  - Altos custos
  - Semanas ou meses
  - Podem surgir problemas que não aparecem nas FPGAs
- Abordagem ASIC estruturada:
  - Ferramenta automatizada converte a implementação FPGA em uma ASIC
  - Diferente de se converter o circuito original em uma implementação ASIC
  - ASIC estruturado irá refletir as estruturas da tabela lookup e da matriz de chaveamento do FPGA original
  - Não será programável
  - Terá tabelas lookup e matrizes de chaveamento mais rápidas (“hardwired”)
  - Menor custo, menor tempo, porém maiores mais lento que um ASIC normal

Linguagens de Descrição de Hardware

- HDLs (Hardware Description Languages)
- Neste curso estudaremos →VHDL
  - “VHSIC (very high-speed integrated circuit) Hardware Description Language.”
  - Criada pelo Departamento de Defesa americano e depois transferido ao IEEE (Institute of Electrical and Electronics Engineers)
  - Definida pelo IEEE Standard 1076
  - Revisada em 1987 como VHDL 87
  - Revisada em 1993 como VHDL 93 – padrão em que o livro se baseia
  - Revisada novamente em 2002

Um pouco mais sobre VHDL...

- VHDL destina-se à descrição e modelagem de um sistema digital em vários níveis de hierarquia e é uma linguagem extremamente complexa
- Interesse do curso é em projeto para síntese de hardware, portanto apenas um subconjunto dos comandos será usado

- **Linguagem usada para descrever hardware!**
  - Essencial ler e escrever códigos da perspectiva de hardware

Estrutura VHDL (1)

Entity (entidade):

- Define o nome do projeto e a lista de suas entradas e saídas (port)
- Nada diz sobre as partes internas do projeto
- A descrição lista os nomes das portas e define os seus tipos

Estrutura VHDL (2)

Architecture (arquitetura):

- Descreve as partes internas do projeto
- A arquitetura começa declarando quais componentes serão usados no projeto
- Cada declaração de componente deve conter a definição de suas entradas e saídas (exatamente de acordo com a declaração de entidade)
- Signal (sinal): sinais internos do projeto (são fios internos)

Aprendendo com um exemplo…

Comparador de igualdade de 1 bit

- Entradas: i0, i1
- Saída: eq
- Expressão Lógica:

$$eq = i0 \cdot i1 + i0' \cdot i1'$$

- Onde:
  - . representa a operação AND,
  - + representa OR, e
  - ’ representa a operação NOT

Tabela-verdade:

input $i0$ $i1$	output eq
0 0	1
0 1	0
1 0	0
1 1	1

Comparador de 1 bit:

- Descrição em VHDL a nível de porta

input		output
$i0$	$i1$	eq
0	0	1
0	1	0
1	0	0
1	1	1

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity eq1 is
   port(
      i0, i1 : in  std_logic;
      eq     : out std_logic
   );
end eq1;

architecture sop_arch of eq1 is
   signal p0, p1 : std_logic;
begin
   -- sum of two product terms
   eq <= p0 or p1;
   -- product terms
   p0 <= (not i0) and (not i1);
   p1 <= i0 and i1;
end sop_arch;
```

Detalhes da linguagem

- VHDL é case insensitive, ou seja, não diferencia maiúsculas e minúsculas
- VHDL tem formatação livre, o que significa que espaços e linhas em branco podem ser inseridos livremente.
- Identificador (nome de um objeto): 26 letras → i0,i l , data_bus1_enable
- Comentários são inseridos com –

```
-- ******************************
-- a demo circuit
-- ******************************
   eq <= p0 or p1;  --sum term
   p1 <= i0 and i1; --product term #1
```

Palavras reservadas

abs access after alias all and architecture array assert attribute begin block body buffer bus case component configuration constant disconnect downto else elsif end entity exit file for function generate generic guarded if impure in inertial inout is label library linkage literal loop map mod nand new next nor not null of on open or others out package port postponed procedure process pure range record register reject rem report return rol ror select severity signal shared sla sll sra srl subtype then to transport type unaffected units until use variable wait when while with xnor xor

Library e Packages

```
library ieee;
use ieee.std_logic_1164.all;
entity eq1 is
   port(
      i0, i1 : in  std_logic;
      eq     : out std_logic
   );
end eq1;

architecture sop_arch of eq1 is
   signal p0, p1 : std_logic;
begin
   -- sum of two product terms
   eq <= p0 or p1;
   -- product terms
   p0 <= (not i0) and (not i1);
   p1 <= i0 and i1;
end sop_arch;
```

- Package: adicionar tipos de dados, operadores, funções, etc.
- A library IEEE inclui diversos pacotes
- Exemplo
- library ieee;
- use ieee.std_logic_1164.all;
- Linha 1: invoca uma library chamada ieee
- Linha 2: faz o pacote std_logic_1164 visível à unidade de projeto subsequente
- Este pacote é necessário para o tipo de dado std_logic e para std_logic_vector

Entidade

```
library ieee;
use ieee.std_logic_1164.all;
entity eq1 is
   port(
      i0, i1 : in  std_logic;
      eq     : out std_logic
   );
end eq1;

architecture sop_arch of eq1 is
   signal p0, p1 : std_logic;
begin
   -- sum of two product terms
   eq <= p0 or p1;
   -- product terms
   p0 <= (not i0) and (not i1);
   p1 <= i0 and i1;
end sop_arch;
```

- Especifica as portas de entrada e saída de um sistema
- Sintaxe simplificada

```
entity entity_name is
   port(
      port_names: mode data_type;
      port_names: mode data_type;
      ...
      port_names: mode data_type
   );
end entity_name;
```

Entidade

```
library ieee;
use ieee.std_logic_1164.all;
entity eq1 is
    port(
        i0, i1 : in  std_logic;
        eq     : out std_logic
    );
end eq1;

architecture sop_arch of eq1 is
   signal p0, p1 : std_logic;
begin
   -- sum of two product terms
   eq <= p0 or p1;
   -- product terms
   p0 <= (not i0) and (not i1);
   p1 <= i0 and i1;
end sop_arch;
```

- Modo (*mode*):
  - in: fluxo entrando no circuito
  - out: fluxo saindo do circuito
  - inout: bidirecional (usado com tristate buffer)
- Tipo de dado (data type)
  - Uma série de valores que um objeto pode assumir
  - Uma série de operações que pode ser executada em objetos deste tipo de dado
- VHDL é strongly-typed language
  - Um objeto pode unicamente ser atribuído com um valor de seu tipo
  - Somente usar operações definidas para o tipo de dado

Tipos de dados

- Unicamente poucos tipos de dados são usados em síntese
- std_logic → pacote IEEE std_logic_1164
- 9 valores: ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-')
  - '0', '1': lógico 0 e lógico 1
  - 'Z': alta-impedância, como em um buffer tristate
  - 'L' , 'H': lógico 0 e lógico 1 fracos, como em wired-logic
  - 'X', 'W': desconhecido (unknown) e desconhecido fraco (weak unknown)
  - 'U': para não inicializado (uninitialized)
  - '-': para don't-care.
- **Em síntese usamos apenas '0', '1', e 'Z'**

Tipos de dados

- std_logic_vector
  - Um array de elementos do tipo std_logic
  - Definido no pacote IEEE std_logic_1164
  - Implica em um barramento (bus) em um circuito físico
- Exemplo
  - signal a: std_logic_vector(7 downto 0);
- Outra forma (menos recomendada)
  - signal a: std_logic_vector(0 to 7);
- **Outros tipos de dados serão discutidos posteriormente**

Operadores

- Operadores: not,and,or,xor
- Definidos para os tipos std_logic e std_logic_vector
- São necessários parênteses para especificar precedência
    - Exemplo: (a and b) or (c and d)
- Inferem portas lógicas durante a síntese

Operadores

- Além dos lógicos, operadores Relacionais e Aritméticos podem ser sintetizados automaticamente

VHDL-93 and IEEE std_logic_1164 package

Operator	Description	Data type of operands	Data type of result
`a ** b`	exponentiation	`integer`	`integer`
`a * b`	multiplication		
`a / b`	division	*integer type for constants and array boundaries, not synthesis*	
`a + b`	addition		
`a - b`	subtraction		
`a & b`	concatenation	1-D array, element	1-D array
`a = b`	equal to	any	`boolean`
`a /= b`	not equal to		
`a < b`	less than	scalar or 1-D array	`boolean`
`a <= b`	less than or equal to		
`a > b`	greater than		
`a >= b`	greater than or equal to		
`not a`	negation	`boolean`, `std_logic`, `std_logic_vector`	same as operand
`a and b`	and		
`a or b`	or		
`a xor b`	xor		

VHDL-93 and IEEE numeric_std package

Overloaded operator	Description	Data type of operands	Data type of result
`a * b`	arithmetic operation	`unsigned`, `natural`	`unsigned`
`a + b`		`signed`, `integer`	`signed`
`a - b`			
`a = b`	relational operation	`unsigned`, `natural`	`boolean`
`a /= b`		`signed`, `integer`	`boolean`
`a < b`			
`a <= b`			
`a > b`			
`a >= b`			

Conversões entre *std_logic_vector* e *numeric*

Data type of a	To data type	Conversion function/type casting
unsigned, signed	std_logic_vector	std_logic_vector(a)
signed, std_logic_vector	unsigned	unsigned(a)
unsigned, std_logic_vector	signed	signed(a)
unsigned, signed	integer	to_integer(a)
natural	unsigned	to_unsigned(a, size)
integer	signed	to_signed(a, size)

Atenção!

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
. . .
signal s1, s2, s3, s4, s5, s6 : std_logic_vector(3 downto 0);
signal u1, u2, u3, u4, u5, u6, u7 : unsigned(3 downto 0);
. . .
```

```
u1 <= s1;   -- not ok, type mismatch
u2 <= 5;    -- not ok, type mismatch
s2 <= u3;   -- not ok, type mismatch
s3 <= 5;    -- not ok, type mismatch
```

```
u1 <= unsigned(s1);       -- ok, type casting
u2 <= to_unsigned(5,4);   -- ok, conversion function
s2 <= std_logic_vector(u3); -- ok, type casting
s3 <= std_logic_vector(to_unsigned(5,4)); -- ok
```

Atenção!

```
u4 <= u2 + u1;  -- ok, both operands unsigned
u5 <= u2 + 1;   -- ok, operands unsigned and natural
```

```
s5 <= s2 + s1; -- not ok, + undefined over the types
s6 <= s2 + 1;  -- not ok, + undefined over the types
```

```
s5 <= std_logic_vector(unsigned(s2) + unsigned(s1)); -- ok
s6 <= std_logic_vector(unsigned(s2) + 1);            -- ok
```

Existem outros pacotes…
(ver *Chap 3. RT-Level Combinational Circuit*, FPGA PROTOTYPING BY VHDL EXAMPLES)

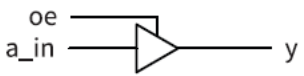
Operador de concatenação

```
signal a1 : std_logic;
signal a4 : std_logic_vector(3 downto 0);
signal b8, c8, d8 : std_logic_vector(7 downto 0);
. . .
b8 <= a4 & a4;
c8 <= a1 & a1 & a4 & "00";
d8 <= b8(3 downto 0) & c8(3 downto 0);
signal a : std_logic_vector(7 downto 0);
signal rot, shl, sha : std_logic_vector(7 downto 0);
. . .
-- rotate a to right 3 bits
rot <= a(2 downto 0) & a(7 downto 3);
-- shift a to right 3 bits and insert 0 (logic shift)
shl <= "000" & a(7 downto 3);
-- shift a to right 3 bits and insert MSB
-- (arithmetic shift)
sha <= a(7) & a(7) & a(7) & a(7 downto 3);
```

Valor ‘Z’

- Sintetizável usando um buffer tri-state

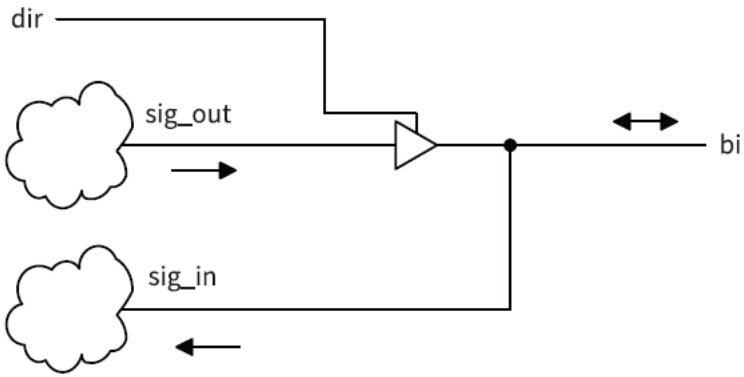
```
y <= a_in when oe='1' else 'Z';
```



oe	y
0	Z
1	a_in

- Utilizando portas bidirecionais

```
entity bi_demo is
   port(
      bi: inout std_logic;
      . . .
   )
begin
   sig_out <= output_expression;
   . . .
   some_signal <= expression_with_sig_in;
   . . .
   bi      <= sig_out when dir='1' else 'Z';
   sig_in <= bi;
   . . .
```



Arquitetura

```
library ieee;
use ieee.std_logic_1164.all;
entity eq1 is
   port(
      i0, i1 : in  std_logic;
      eq     : out std_logic
   );
end eq1;

architecture sop_arch of eq1 is
   signal p0, p1 : std_logic;
begin
   -- sum of two product terms
   eq <= p0 or p1;
   -- product terms
   p0 <= (not i0) and (not i1);
   p1 <= i0 and i1;
end sop_arch;
```

- Descreve a operação interna ou a estrutura de um sistema
- Sintaxe simplificada

```
architecture arch_name of entity_name is
   declarations;
begin
   concurrent statement;
   concurrent statement;
   concurrent statement;
   . . .
end arch_name;
```

Arquitetura

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity eq1 is
   port(
      i0, i1 : in  std_logic;
      eq     : out std_logic
   );
end eq1;

architecture sop_arch of eq1 is
   signal p0, p1 : std_logic;
begin
   -- sum of two product terms
   eq <= p0 or p1;
   -- product terms
   p0 <= (not i0) and (not i1);
   p1 <= i0 and i1;
end sop_arch;
```

Declarações de sinais (signal declaration)

- Sinais podem ser pensados como fios internos (nets)

Comando concorrente (concurrent statement)

- Pode ser visto como uma parte do circuito
- Especifica funcionalidade e tempo

Comando de atribuição concorrente

- Pode ser visto como uma parte do circuito
- Identificador do lado esquerdo
  - Deve ser *signal* ou *port* de saída do circuito
- Identificador(es) do lado direito
  - Deve(m) ser *signal* ou *port* de entrada do circuito
- Expressão do lado direito
  - Especifica a função e o tempo, ou seja, qual é o novo valor e quando ele deve tomar efeito

Aprendendo com um exemplo…

Comparador de 2 bits

- Podemos expandir o comparador para entradas de 2 bits.
- Sejam a e b as entradas e aeqb a saída.
- O sinal aeqb é ativado quando ambos os bits de a e b são iguais.

$$aeqb = a(1)'.a(0)'.b(1)'.b(0)' + a(1)'.a(0).b(1)'.b(0) +$$
$$a(1).a(0)'.b(1).b(0)' + a(1).a(0).b(1).b(0)$$

$$aeqb = a(1)'.b(1)'.a(0)'.b(0)' + a(1)'.b(1)'.a(0).b(0) +$$
$$a(1).b(1).a(0)'.b(0)' + a(1).b(1).a(0).b(0)$$

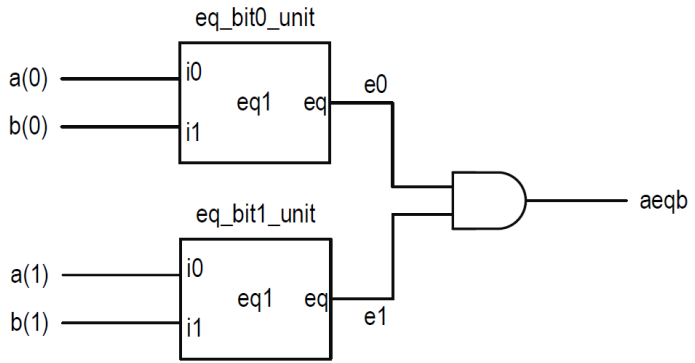
a	*b*	*aeqb*
00	00	1
00	01	0
00	10	0
00	11	0
01	00	0
01	01	1
01	10	0
01	11	0
10	00	0
10	01	0
10	10	1
10	11	0
11	00	0
11	01	0
11	10	0
11	11	1

Descrição a nível de porta

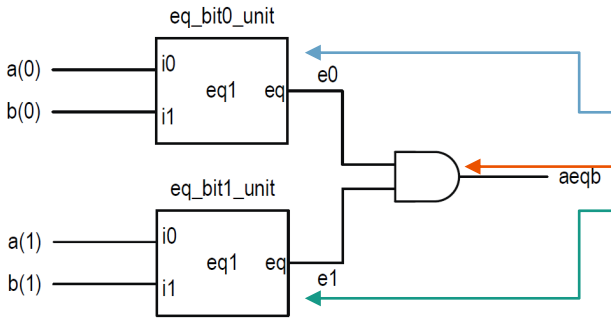
```
library ieee;
use ieee.std_logic_1164.all;
entity eq2 is
   port(
      a, b : in std_logic_vector(1 downto 0);
      aeqb : out std_logic);
end eq2;

architecture sop_arch of eq2 is
   signal p0, p1, p2, p3: std_logic;
begin
   -- sum of product terms
   aeqb <= p0 or p1 or p2 or p3;
   -- product terms
   p0 <= ((not a(1)) and (not b(1))) and
         ((not a(0)) and (not b(0)));
   p1 <= ((not a(1)) and (not b(1))) and (a(0) and b(0));
   p2 <= (a(1) and b(1)) and ((not a(0)) and (not b(0)));
   p3 <= (a(1) and b(1)) and (a(0) and b(0));
end sop_arch;
```

Descrição Estrutural

- Um sistema digital é frequentemente composto de diversos subsistemas menores
- Construir um sistema grande a partir de componentes mais simples ou pré-projetados
- **Instanciação de componente**
- Este tipo de código é chamado descrição estrutural



Descrição Estrutural




```
library ieee;
use ieee.std_logic_1164.all;
entity eq2 is
   port(
      a, b : in std_logic_vector(1 downto 0);
      aeqb : out std_logic);
end eq2;
architecture struc_arch of eq2 is
   signal e0, e1: std_logic;
begin
   -- instantiate two 1-bit comparators
   eq_bit0_unit: entity work.eq1(sop_arch)
      port map(i0=>a(0), i1=>b(0), eq=>e0);
   eq_bit1_unit: entity work.eq1(sop_arch)
      port map(i0=>a(1), i1=>b(1), eq=>e1);
   -- a and b are equal if individual bits are equal
   aeqb <= e0 and e1;
end struc_arch;
```

Instanciação de Componentes

- Sintaxe simplificada

```
unit_label: entity lib_name.entity_name(arch_name)
   port map(
      formal_signal=>actual_signal,
      formal_signal=>actual_signal,
      . . .
```

- Sobre o componente
  - *unit_label: um nome único para identificar a instância*
  - *lib_name: a biblioteca onde o componente reside;*
  - *A biblioteca "work" é frequentemente usada*
- Mapeamento de portas (port map)
  - *formal_signal: sinal usado na declaração da entidade do componente*
  - *actual_signal: sinal usado na arquitetura corrente*

Instanciação de Componentes

- Instanciação de componente é um comando concorrente
- Ele representa uma caixa preta cuja função é definida em outro módulo
- O código textual e o esquemático fornecem informação similar
- Alguns programas podem converter os dois formatos

Arquivo de *constraints*

- Localização dos pinos de E/S
- Por exemplo, as chaves e LEDs da placa são "pré conectados" a pinos de E/S específicos do dispositivo FPGA e não podem ser alterados
- A atribuição dos pinos é definida em um arquivo de restrição, que é processado em conjunto com os arquivos HDL
- Vamos abrir o exemplo da nossa placa!

Boas práticas - TOP-LEVEL SIGNAL MAPPING

```
library ieee;
use ieee.std_logic_1164.all;
entity eq_top is
   port(
      sw  : in  std_logic_vector(3 downto 0); -- 4 switches
      led : out std_logic_vector(0 downto 0)  -- 1 red LED
   );
end eq_top;

architecture struc_arch of eq_top is
begin
   -- instantiate 2-bit comparator
   eq2_unit : entity work.eq2(struc_arch)
      port map(
            a    => sw(3 downto 2),
            b    => sw(1 downto 0),
            aeqb => led(0)
         );
  end struc_arch;
```

Parte prática – Demonstração

- Baseado em A.2 SHORT TUTORIAL ON VIVADO HARDWARE DEVELOPMENT (APPENDIX A) do livro P. Chu, “FPGA Prototyping by VHDL Examples”

Ver roteiro…