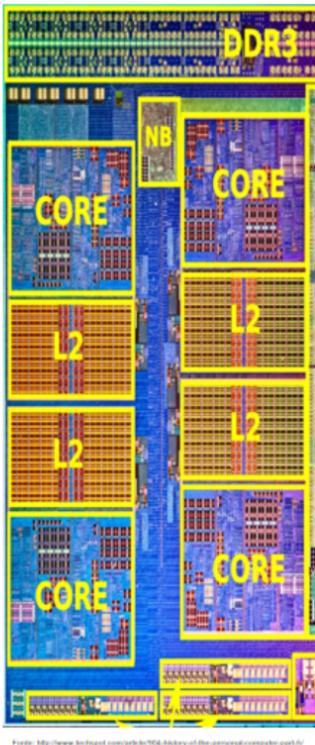


ENSINO NÃO PRESENCIAL EMERGENCIAL - BLOCO C



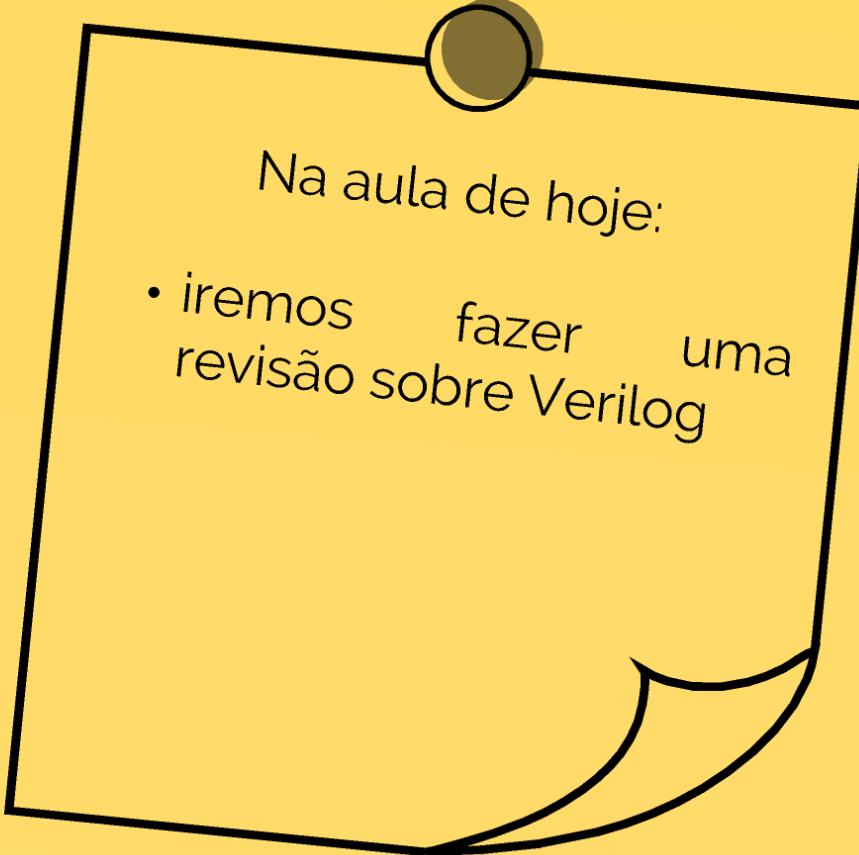
Arquitetura e Organização de Computadores 1

Prof. Luciano de Oliveira Neris
luciano@dc.ufscar.br

Prof. Artino Quintino da Silva Filho
artino@ufscar.br

Departamento de Computação
Universidade Federal de São Carlos





Na aula de hoje:

- iremos fazer uma revisão sobre Verilog

REVISÃO SOBRE VERILOG

Prof. Artino Quintino da Silva Filho

Introdução

Verilog
conceitos
básicos

Verilog
Modelagem



Introdução

Vamos ver alguns conceitos importantes

O quê é
HDL ?

O quê é
FPGA ?

Verilog e
SystemVerilog

O quê é HDL ?



HDL é uma linguagem para **descrever** sistemas digitais

É uma forma estruturada para a descrição de circuitos digitais e permite descrever um circuito em termos estruturais, como portas lógicas, ou comportamentais, de maneira mais abstrata utilizando uma expressão lógica.

Existem diversas linguagens HDL:

- ISP – Instruction Set Processor (1977)
- VHDL (1981, padrão IEEE em 1987)
- Verilog (1985)
- AHDL – Altera HDL (proprietária)
- SystemC
- Handel-C



VHDL é utilizada mundialmente por empresas de CAD, onde tem grande adoção na Europa.

Já Verilog é muito usado nos EUA

Níveis Hierárquicos

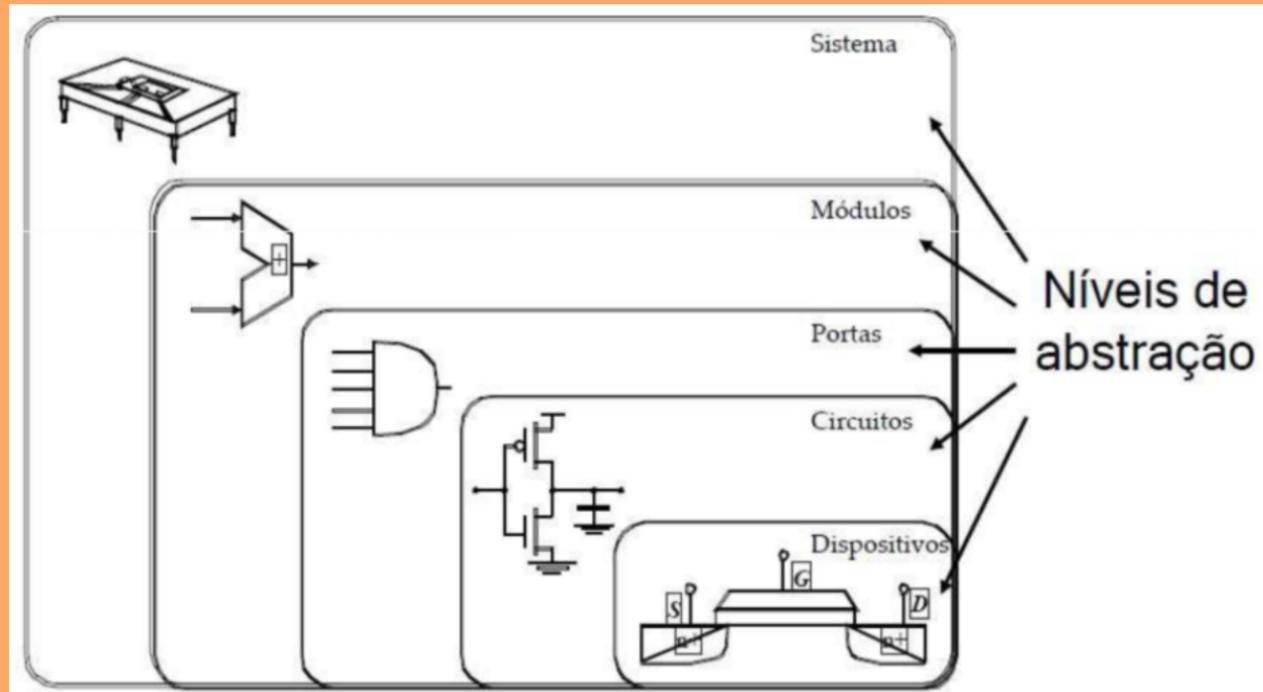
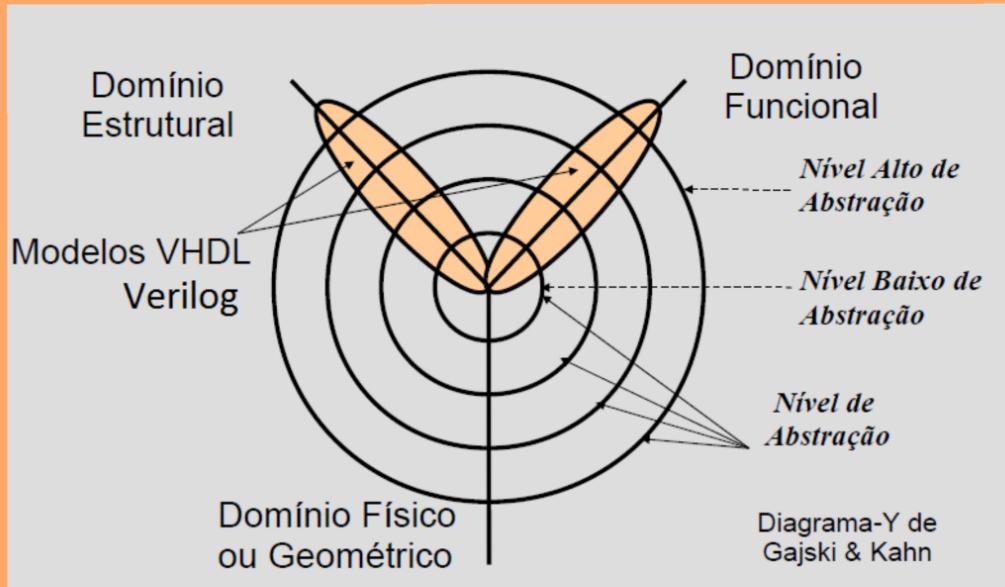


Diagrama-Y



- Com HDL, podemos manipular o comportamento de todos estes níveis, com exceção do domínio físico/material;
- Podemos simular a funcionalidade de cada etapa do processo;
- Cada etapa é mais detalhada do que a anterior;
- Na simulação, podemos incluir informações específicas para averiguar o comportamento do projeto.

Benefícios



Especificação do sistema digital

- Projetos independentes da tecnologia (implementação física é postergada)
- Ferramentas de CAD compatíveis entre si
- Permite explorar, em um nível mais alto de abstração (em relação a diagramas de esquemáticos) diferentes alternativas de implementação
- Permite, através de simulação, verificar o comportamento do sistema digital

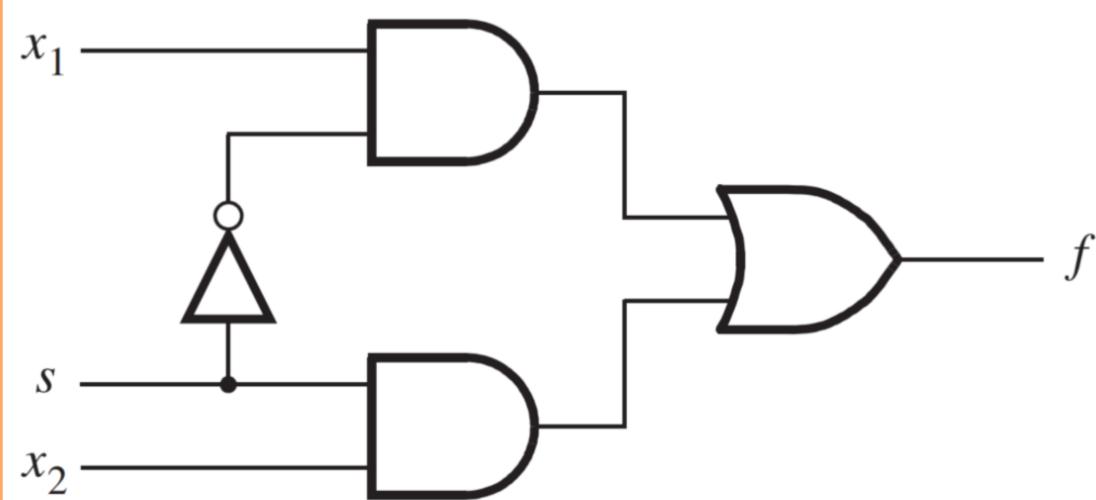
Nível físico

- Reduz tempo de projeto (favorece níveis abstratos de projeto)
- Reduz custo do projeto
- Elimina erros de baixo nível (se usado como base de ferramentas automatizadas)
- **Consequência:** reduz "time-to-market" (tempo de chegada de um produto ao mercado)

Desvantagens

Em relação a diagramas de esquemáticos

- Hardware gerado pode ser menos otimizado



```
module example1 (x1, x2, s, f);  
  input x1, x2, s;  
  output f;  
  not (k, s);  
  and (g, k, x1);  
  and (h, s, x2);  
  or (f, g, h);  
  
endmodule
```

Introdução

Vamos ver alguns conceitos importantes

O quê é
HDL ?

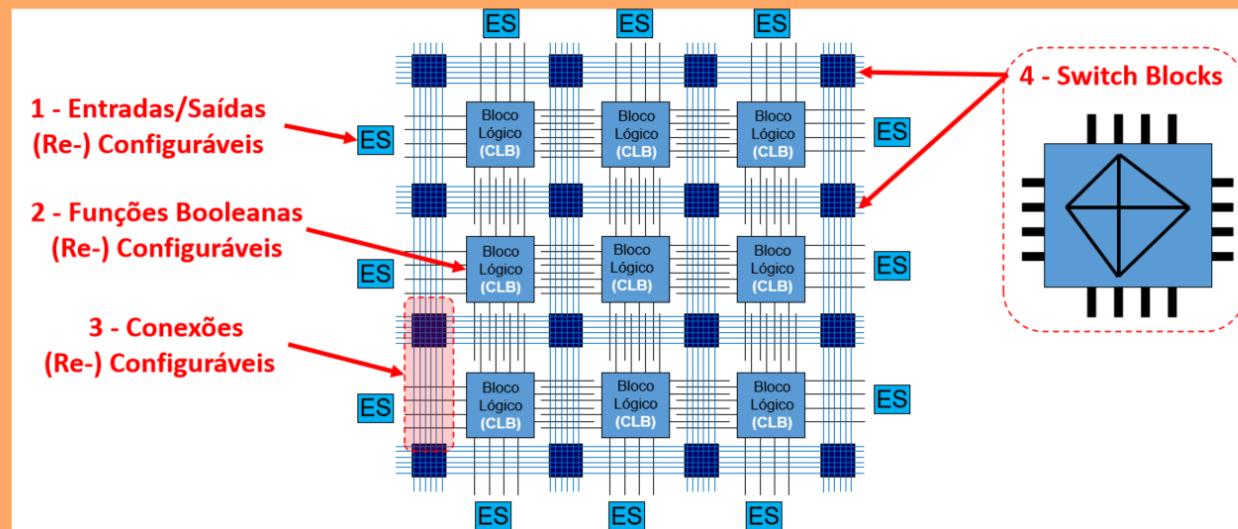
O quê é
FPGA ?

Verilog e
SystemVerilog

O quê são FPGAs ?



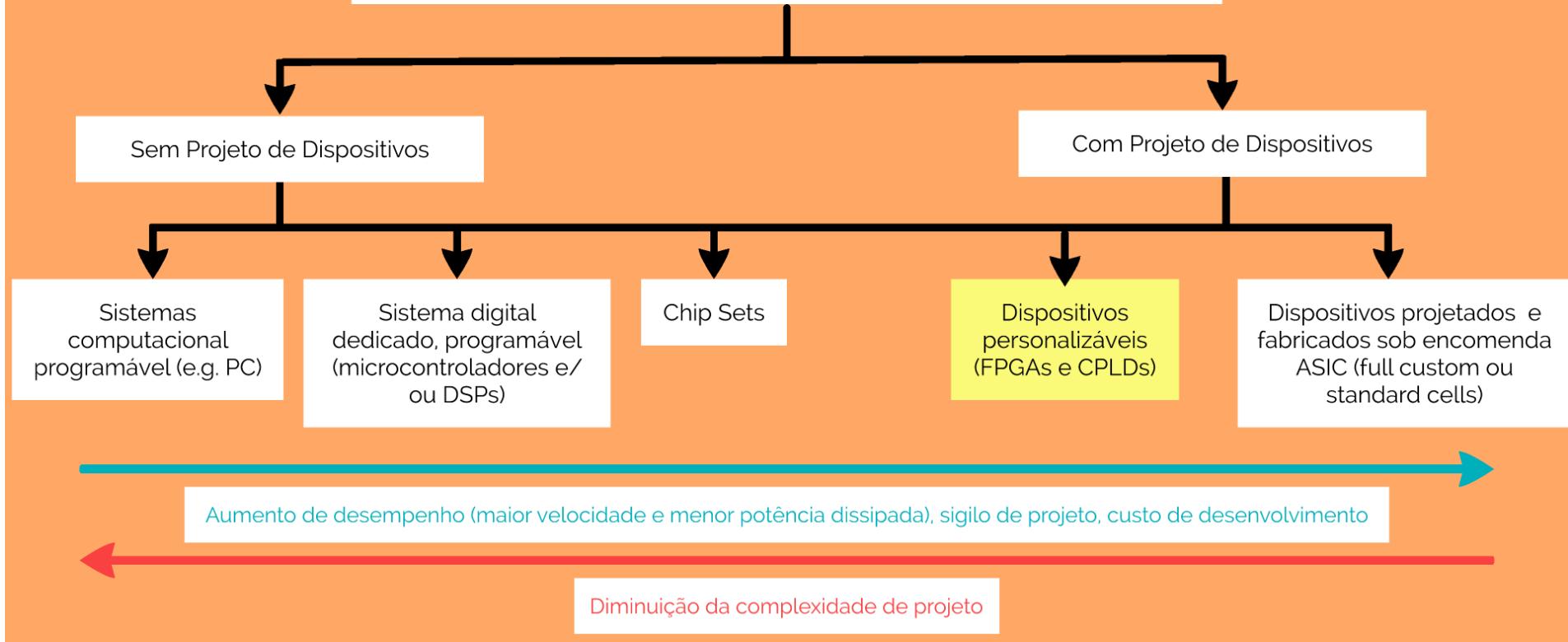
- FPGA (Field Programmable Gate Array) são dispositivos semicondutores programáveis baseados em uma matriz de blocos lógicos configuáveis (CLBs) conectados através de interconexões programáveis;



Histórico das FPGAs

- Primeiro vieram PROMs e PLDs (Programmable logic devices), matrizes de portas (re-) configuráveis
- Algumas patentes de coisas parecidas com FPGAs surgiram no final dos anos 80 e início dos anos 90 (Casselman, Page, Peterson)
- Os fundadores da Xilinx, Ross Freeman e Bernard Vonderschmitt, inventaram o primeiro FPGA comercial em 1985 - o XC2064

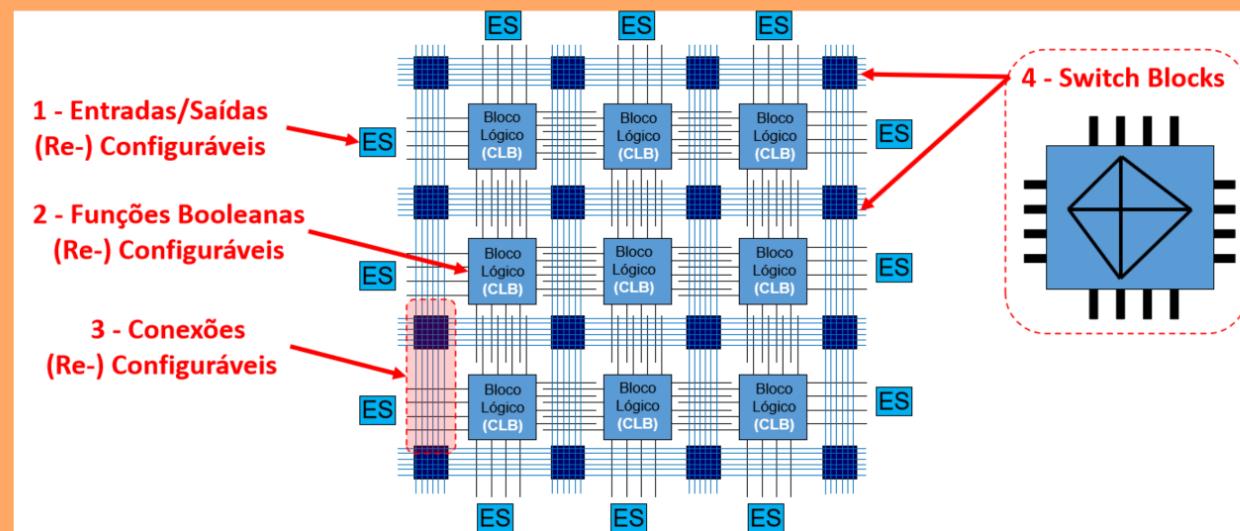
Projeto e Implementação de Produtos Tecnológicos Baseados em Circuitos Eletrônicos



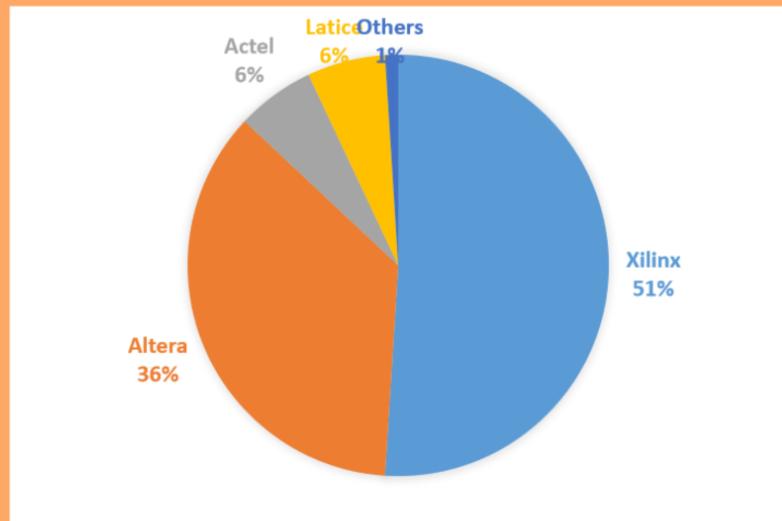
O quê são FPGAs ?



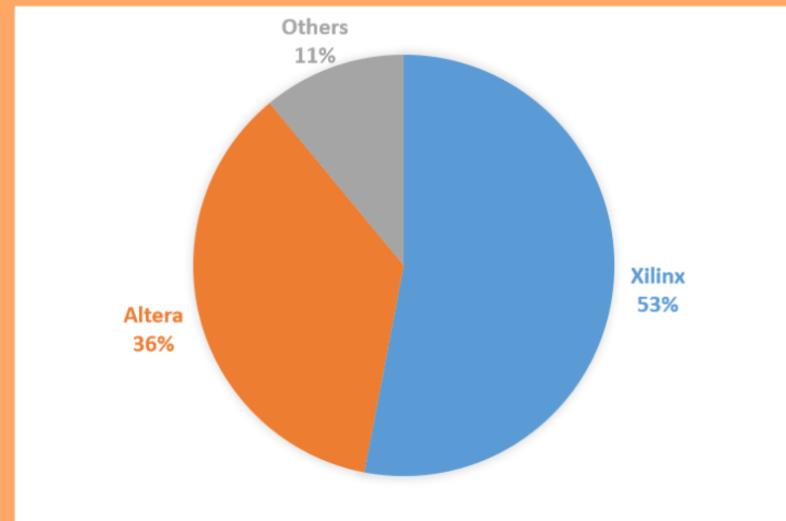
- FPGAs permitem implementar circuitos digitais diretamente de HDLs, sem os custos de fabricação de chips!



O mercado de dispositivos reconfiguráveis



PLDs



FPGAs

Exemplo de uma FPGA



Introdução

Vamos ver alguns conceitos importantes

O quê é
HDL ?

O quê é
FPGA ?

Verilog e
SystemVerilog

Verilog é uma linguagem de programação?

NÃO !!! Pois é uma linguagem de descrição de hardware!

- Diferentes das linguagens de programação onde comando são executados um após o outro, Verilog - assim como outras HDLs - pode executar comandos em paralelo;
- Fornece um nível de abstração para auxiliar na implementação de um projeto sem a necessidade de conhecimento prévio de todo o projeto;
- Compatível com tecnologias back-end, permite flexibilidade em termos de escolha de tecnologia, metodologia de roteamento, e outras decisões de implementação back-end.

Verilog

Código é executado em um simulador

- Não se enxerga o “compilador” de Verilog, não há um “código executável” visível

Projeto do usuário

- Especificado em nível de abstração de transferência em registradores (em inglês, Register – Transfer Level ou RTL), mas não apenas neste nível!

Histórico do Verilog



- Inicialmente, Verilog era uma linguagem proprietária desenvolvida pela empresa Gateway.
- Verilog foi desenvolvida nos anos 1980 e foi inicialmente usada para modelar dispositivos ASIC. Em 1990, Verilog caiu no domínio público e agora está sendo padronizado como IEEE 1364.

SystemVerilog

- Sintaxe mais moderna e simplificada;
- Descrição do module já vem com inputs e outputs declaradas;
- Evita confusões na utilização de sinais **reg** com **wire**, no **always** para circuitos sequenciais;
- Introduz o **logic**.

REVISÃO SOBRE VERILOG

Prof. Artino Quintino da Silva Filho

Introdução

Verilog
conceitos
básicos

Verilog
Modelagem



Verilog

conceitos básicos

Anatomia
de um
arquivo em
Verilog

O módulo

Tipos de
dados

Representação
de dados

Terminais

Sinais

Anatomia de um arquivo em verilog



```
Definição do módulo  
e do nome da função  
module Exemplo (a, b, out);  
Nome das portas  
e variáveis  
    input [7:0] a, b; // cabeçalho  
    output [7:0] out; // entradas  
    // saídas Comentários  
    Wire An; // Declarações internas  
    // sinais  
    // constantes  
    // parâmetros  
    assign out = a + b; // operação  
endmodule // Descrição do  
// comportamento
```

Verilog

conceitos básicos

Anatomia
de um
arquivo em
Verilog

O módulo

Tipos de
dados

Representação
de dados

Terminais

Sinais

Módulo



- Todos os sistemas em Verilog são descritos dentro de um **módulo**.
- Os módulos descrevem a funcionalidade de um circuito e definem os terminais (pinos e portas) de entrada e saída.
- Módulos podem incluir outros módulos de maneira a suportar um projeto hierárquico.

O módulo

```
module multiplicador (a, b, y);
    input [3:0] a, b;
    output [7:0] y;
    wire [4:0] int;

    //instanciando um módulo somador
    somador u1 (.x1(a), .x2(b), .z(int));

    //continuação do módulo multiplicador

endmodule
```

```
module somador (x1, x2, z);
    input [3:0] x1, x2;
    output [4:0] z;

    //continuação do módulo somador

endmodule
```

Exemplo de hierarquia

Módulo



- Módulo é a mesma coisa que método (ou função) em outras linguagens.
- Os comandos **module** e **endmodule** significam o começo e o término do sistema descrito.

sintaxe (Módulo)



```
module nome_do_modulo (entrada 1, entrada N, ..., saída1, ..., saídaN);
```

Exemplo 1

```
module SomaSub (A, B, Seletor, Out); // Início do módulo SomaSub
    ...
endmodule // Fim do módulo SomaSub
```

Exemplo 2

```
module MaiorQue (A, B, Status); // Início do módulo MaiorQue
    ...
endmodule // Fim do módulo MaiorQue
```

Verilog

conceitos básicos

Anatomia
de um
arquivo em
Verilog

O módulo

Tipos de
dados

Representação
de dados

Terminais

Sinais

Tipos de dados



- No Verilog, todos os sinais, constantes, variáveis ou funções precisam ser associadas a um tipo de dados;
- Alguns tipos de dados podem ser obtidos diretamente, enquanto outros são obtidos apenas através de uma modelagem comportamental do circuito.
- Existem dois tipos de dados fundamentais em Verilog denominados, **net** e **variable**.

Tipos de dados (net)



- Tipos de dados **net** geralmente modelam a interconexão entre os componentes.
- Por default, o Verilog assume que **net** de entrada (**input**) e saída (**output**) são do subtipo **wire**.
- **wires** são drivers que apenas conectam dois pontos

Tipos de dados (variables)



- Tipos de dado **variables** geralmente representam variáveis de armazenamento temporário, que podem ser sintetizados em circuitos combinacionais ou sequenciais, ou através de contatos e tipos de declarações.
- Um tipo de dado **variables** pode ser um dos seguintes tipos: **register**, integer, real, time e realtime.
- **registradores** são drivers que guardam valores

Tipos de dados (**reg**)



- Uma variável do tipo **reg** guarda informação e pode ser designada somente por um procedimento de declaração, uma tarefa ou uma função;
- Podem assumir o valor de 0, 1, X e Z, e manterão o valor até serem novamente referenciadas;
- Uma variável do tipo **reg** também não pode ser a saída de uma porta ou de uma declaração de designação (são regras do Verilog)

Em SystemVerilog

- SystemVerilog contém todos os tipos de dados de Verilog.
- Como pode ser confuso declarar **reg** ou **wire** em Verilog, foi introduzido um novo tipo de dados chamado **logic** em SystemVerilog.
- De acordo com o uso, a ferramenta irá interpretar o seu comportamento.

Verilog

conceitos básicos

Anatomia
de um
arquivo em
Verilog

O módulo

Tipos de
dados

Representação
de dados

Terminais

Sinais

Representação de dados



- Verilog suporta valores escalares que representam sinais individuais e vetores que correspondem a múltiplos sinais.
- Cada sinal individual pode assumir quatro valores possíveis: 0, 1, Z e X.

0 = valor lógico 0, uma condição falsa
1 = valor lógico 1, uma condição verdadeira
z ou Z = alta impedância, tri-state ou flutuante
x ou X = valor desconhecido ou não inicializado.

Também pode representar a situação de don't care

Representação de dados



- Em Verilog, um vetor é uma array unidimensional.
- Embora qualquer range de índices possa ser utilizado, é comum que os valores LSB se iniciem em zero.

```
wire [7:0] Soma; // define um vetor de 8-bits  
// chamado Soma do tipo  
// wire. Ao MSB é dado o  
// índice 7, enquanto ao LSB é  
// dado o índice 0  
  
reg [15:0] Q; // Define um vetor de 16 bits  
// chamado Q do tipo reg.
```

Expressando números utilizando bases diferentes



- Se um número, em Verilog, não tiver uma sintaxe para identificação, ele é tratado como inteiro
- A sintaxe para especificação da base é:

```
<tamanho_em_bits> <base> <valor>
```

Sintaxe	Base
'b	binária
'o	octal
'd	decimal
'h	hexadecimal

Expressando números utilizando bases diferentes



Exemplos

```
5'd10;           // constante decimal 10, com 5 bits  
  
10'b1010_0011_11; // constante com 10 bits, em binário  
  
16'h1E_C6;      // 16 bits em hexadecimal
```

nº de bits

base de representação

valor (pode-se utilizar "_" entre dígitos)

Verilog

conceitos básicos

Anatomia
de um
arquivo em
Verilog

O módulo

Tipos de
dados

Representação
de dados

Terminais

Sinais

Terminais



- O primeiro item que definimos dentro do modulo são os terminais de entrada (inputs) e de saída (outputs), ou portas.
- Cada terminal/porta precisa ter um nome definido, uma direção e um tipo;
- Por padrão, terminais tem a dimensão de 1 bit.
- Barramentos com mais de 1 bit tem a dimensão declarada ANTES do nome do terminal

Terminais



- Usar uma linha de declaração para terminais com mesmo número de bits:

`input a,b;`

`output [7:0] c,e;`

`output [3:0] d;`



[7:0] - Little Endian - bits menos significativos à direita

[0:7] - Big Endian - bits menos significativos à esquerda

Sintaxe / semântica (terminais de entrada)

input – usado para declarar uma entrada.

- **Sintaxe e Semântica:** `input nome_da_entrada;`

Exemplo 1

`input In1, In2; // Declarando duas entradas, In1 e In2, de 1 bit`

Exemplo 2

`input [3:0] A; // Declarando uma entrada, A, de 4 bits`

`input [3:0] B; // Declarando uma entrada, B, de 4 bits`

Exemplo 3

`input [7:0] A, B; // Declarando duas entradas, A e B, de 8 bits`

`input Clock; // Declarando uma entrada, Clock, de 1 bit`

Sintaxe / semântica (terminais de saída)

output – usado para declarar uma saída.

- **Sintaxe e Semântica:** output nome_da_saida;

Exemplo 1

output Out; *// Declarando uma saída, Out, de 1 bit*

Exemplo 2

output [2:0] Out; *// Declarando uma saída, Out, de 3 bits*

output Overflow, Status; *// Declarando duas saídas, Overflow e Status, de 1 bit*

Verilog

conceitos básicos

Anatomia
de um
arquivo em
Verilog

O módulo

Tipos de
dados

Representação
de dados

Terminais

Sinais

Principais comandos



- Um sinal é especificado por utilizado juntamente com o módulo e é declarado antes de sua utilização;
- Cada sinal precisa ser declarado especificando o tipo, seguido de seu nome.

Sinais (reg)



- **reg** – usado para quando se deseja guardar um valor em uma entrada ou saída até que outro valor seja enviado para essa entrada ou saída (registrador).
 - *OBS.: Entradas e saídas usadas dentro de blocos always/initial devem ser do tipo reg.*
- **Sintaxe e Semântica:** reg nome_da_entrada/saida;

Exemplo

- **reg In1, In2, Out;** // Declarando In1, In2 e Out como três registradores de 1 bit

Exemplo

- **reg [2:0] Tx, Ty, Tz;** // Declarando Tx, Ty e Tz como três registradores de 3 bits
- **reg [1:0] Tula;** // Declarando Tula como um registrador de 2 bits
- **reg Conta, Reset;** // Declarando Conta e Reset como dois registradores de 1 bit

Sinais (wire)



- **wire** – é o tipo padrão das entradas e saídas. **OBS.:** Não se pode usar entradas/saídas do tipo wire dentro de blocos always/initial.
- **Sintaxe e Semântica:** wire nome_da_entrada/saida;

Exemplo

`wire [3:0] Outtemp; // Declarando um fio, OutTemp, que suporta 4 bits.`

Sinais (logic) - SystemVerilog apenas



- **logic** - substitui os sinais **reg** e **wire** em SystemVerilog. A ideia é que o sintetizador se responsabilize pela escolha entre reg e wire.
- **Sintaxe e Semântica:** logic nome_da_entrada/saida

Exemplo

logic a;

assign a = b^c; *// interpretado como wire*
assign (c or d) a = c + d; *//interpretado como reg*