SystemVerilog para Projetos

Curso Presencial IFPB/UFAL

Marcos Morais – Professor DEE
Projetos de Excelência em Microeletrônica – PEM
Centro de Engenharia Elétrica e Informática
Universidade Federal de Campina Grande

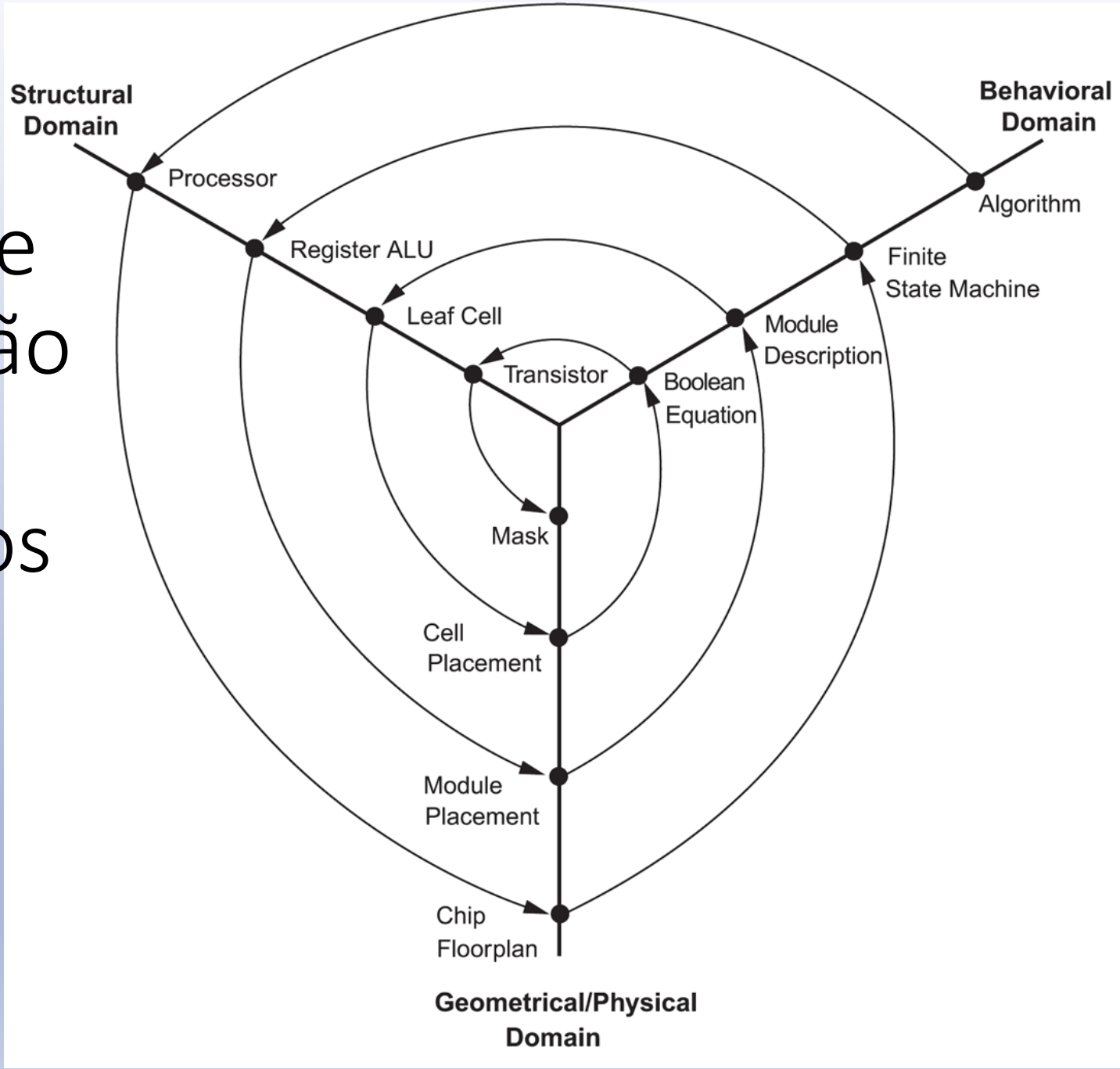
Sumário

- Níveis de abstração;
- Projetos bottom-up e top-down;
- Diagrama de blocos;
- Separação entre caminho de dados (datapath) e controle;
- Linguagens de descrição de hardware;
- Fluxos de projeto em microeletrônica e FPGA;
- Subconjunto sintetizável;
- Descrição comportamental e nível RTL;
- Módulos;
- Tipos de dados e valores literais;
- Operadores e expressões;
- Sentenças procedurais e de controle;
- Atribuição de variáveis;
- Procedimentos e processos;
- Tarefas e funções;
- Interfaces;
- Simulação e síntese lógica;
- Aplicações e exemplos;
- Laboratório - Projeto do flux-capacitor (vai e vem de LEDs);
- Laboratório - Projeto de máximo divisor comum

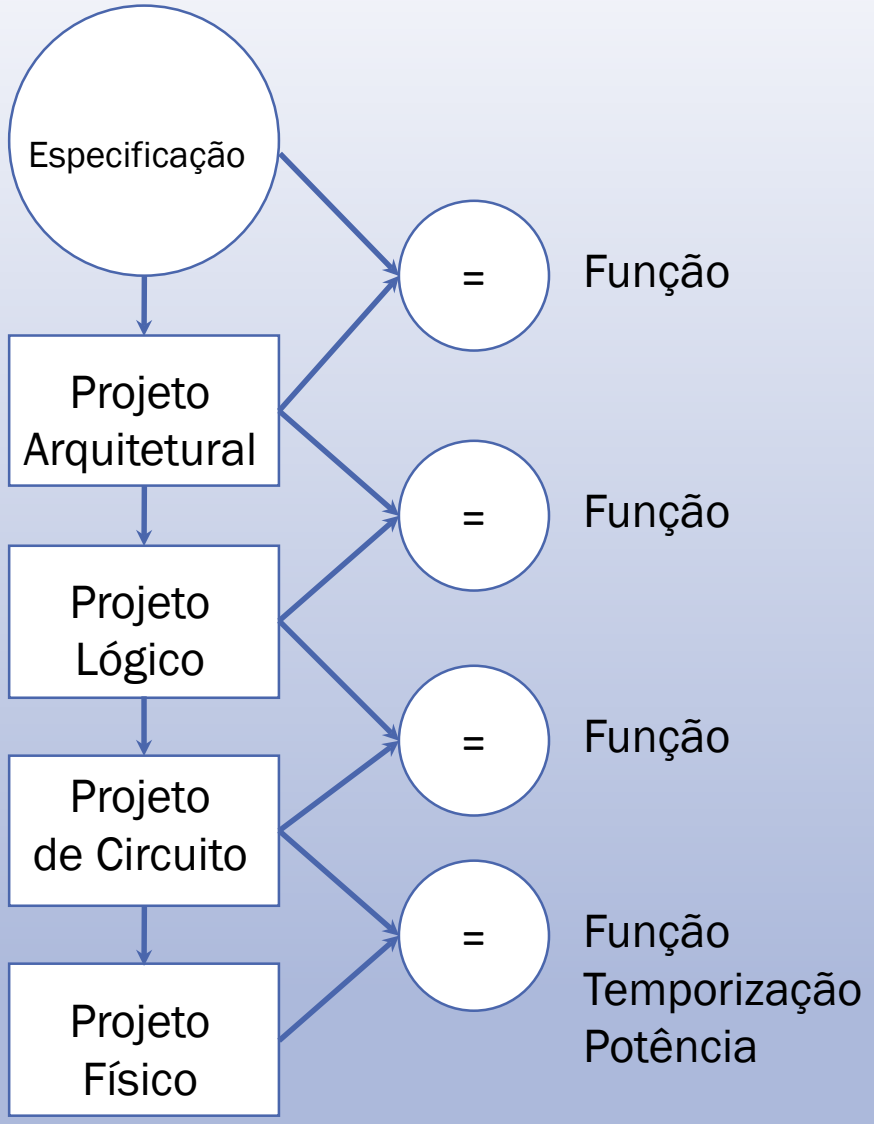
Sistemas Digitais Complexos com SystemVerilog HDL

Níveis de Abstração

Níveis de Abstração e Domínios



Fluxo de desenvolvimento



Hardware Description Language (HDL)

- Para quê precisamos de uma Linguagem de Descrição de Hardware ?

- Modelar, Simular, Sintetizar e Analisar hardware digital
  - Paralelismo
  - Concorrência
  - Semântica para valores de sinais no tempo

- Construções e semântica especiais
  - Transições (bordas) de valores de sinais
  - Atrasos de propagação de sinais
  - Verificação de condições temporais

Descrição

- É possível descrever *qualquer* sistema (em determinado nível de abstração) através de seus elementos constituintes, como estão relacionados e das suas semânticas.
- Ou seja, para descrever um sistema, precisamos de:
  - Componentes (módulos)
  - Conexões

Diagrama Esquemático

- O método antigo de descrição era o diagrama esquemático (esquema)
- O esquemático possui algumas vantagens:
  - Descreve de forma visual.
  - Pode ser formal, ou seja, entrada para o processamento automático
  - Descreve itens
- E muitas desvantagens (para sistemas complexos):

Descrição com HDL

- Uma HDL descreve um sistema (digital) de forma textual
- Adequado ao fluxo de desenvolvimento onde se utilizam ferramentas automáticas
- Permite a entrada em todos os níveis de abstração*
- *Verilog não atingia os níveis mais altos de abstração. SystemVerilog sim.

HDL e níveis de abstração

- As HDLs descrevem um sistema digital em vários níveis:
- Os projetos podem ser descritos desde um nível muito abstrato - nível comportamental, até o nível de layout
- Nivel comportamental – descreve as funcionalidades do projeto sem detalhar a implementação.
- Nível de netlist- componentes básicos em um chip/FPGA.

Como SystemVerilog é Usado

- Praticamente todo ASIC é projetado usando ou SystemVerilog ou VHDL (uma linguagem similar, mais prolixa)
- Modelagem comportamental com alguns elementos estruturais
- “Subconjunto de síntese” é o que pode ser traduzido por um sintetizador em uma netlist
- O projeto é escrito em SystemVerilog
- Simulado exaustivamente para verificar a funcionalidade
- Sintetizado (o netlist é gerado)
- É feita análise de temporal para verificar as temporizações

Descrição com HDL

A Linguagem SystemVerilog

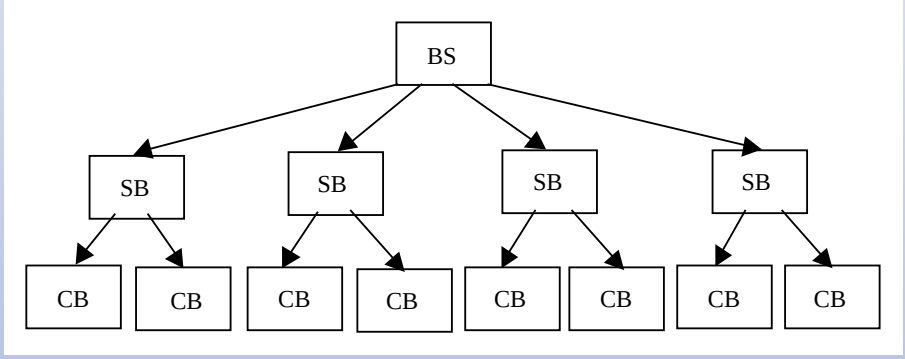
- Originalmente uma linguagem de modelagem para um simulador de lógica digital baseado em eventos muito eficiente
- Mais tarde foi posto em uso como uma linguagem de especificação para síntese lógica
- Atualmente, uma das duas linguagens mais utilizadas em projeto de hardware digital (VHDL é a outra)
- Praticamente todo chip (ASIC, genérico) ou circuito em FPGA é desenvolvido usando uma destas duas linguagens
- Combina estilos de modelagem estrutural e comportamental

Projetos bottom-up & top-down

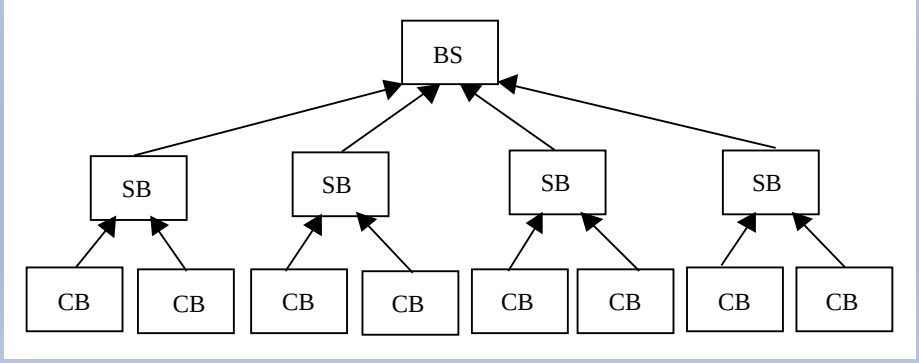
O todo e as partes

Abordagens top-down e bottom-up

- Top-down



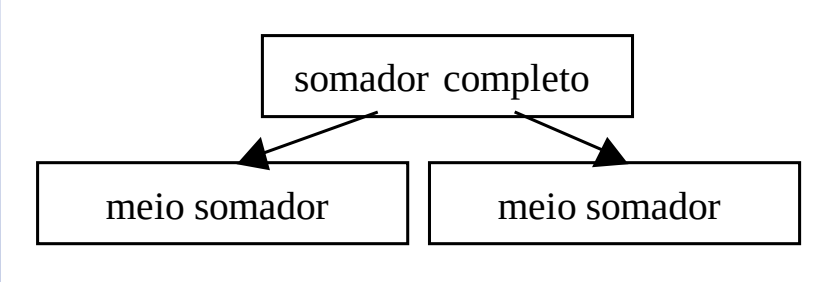
- Bottom-up

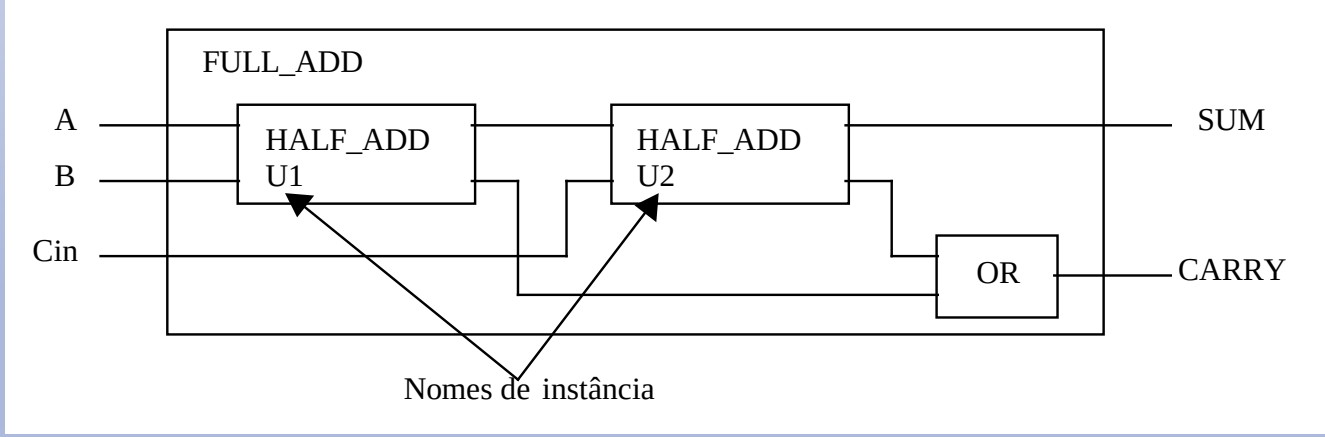


Blocos funcionais: Soma

- Soma binária é usada frequentemente
- Desenvolvimento da soma:
  - *Meio-Somador* (HA), um bloco funcional de 2 entradas binárias,
  - *Somador-Completo* (FA), um bloco funcional de 3 entradas binárias,
  - *Somador Ripple Carry*, um arranjo iterativo que realiza soma binária, e
  - *Somador Carry-Look-Ahead* (CLA), uma estrutura hierárquica para aumentar o desempenho (menor atraso).

Representando hierárquia

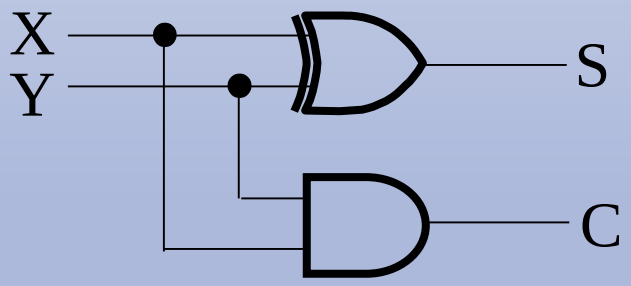




Implementações: Meio-somador

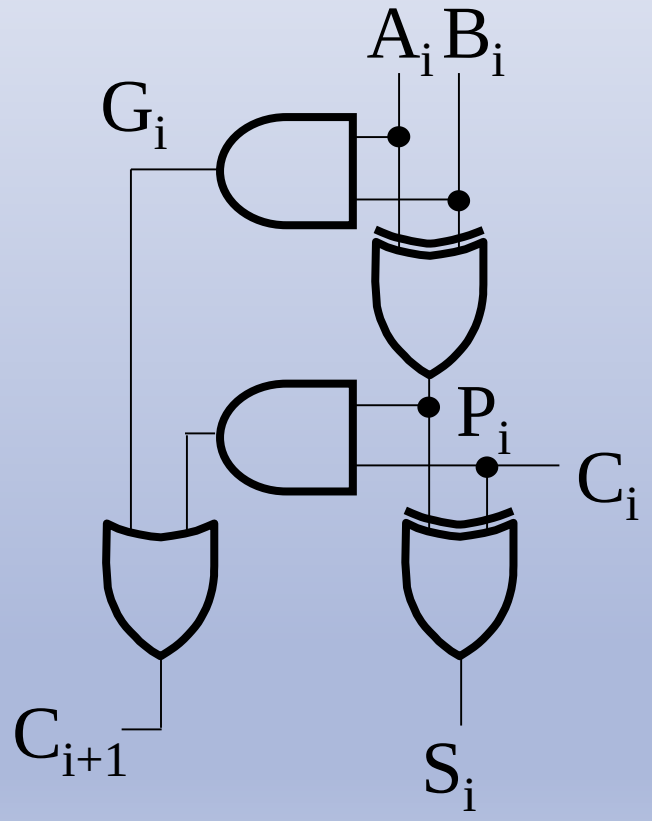
- A implementação mais comum para um meior somador é a

$$S = X \oplus Y$$
$$C = X \cdot Y$$



Implementação: Somador completo

- Esquemático do somador completo
- Observe que é formado por 2 meio-somadores



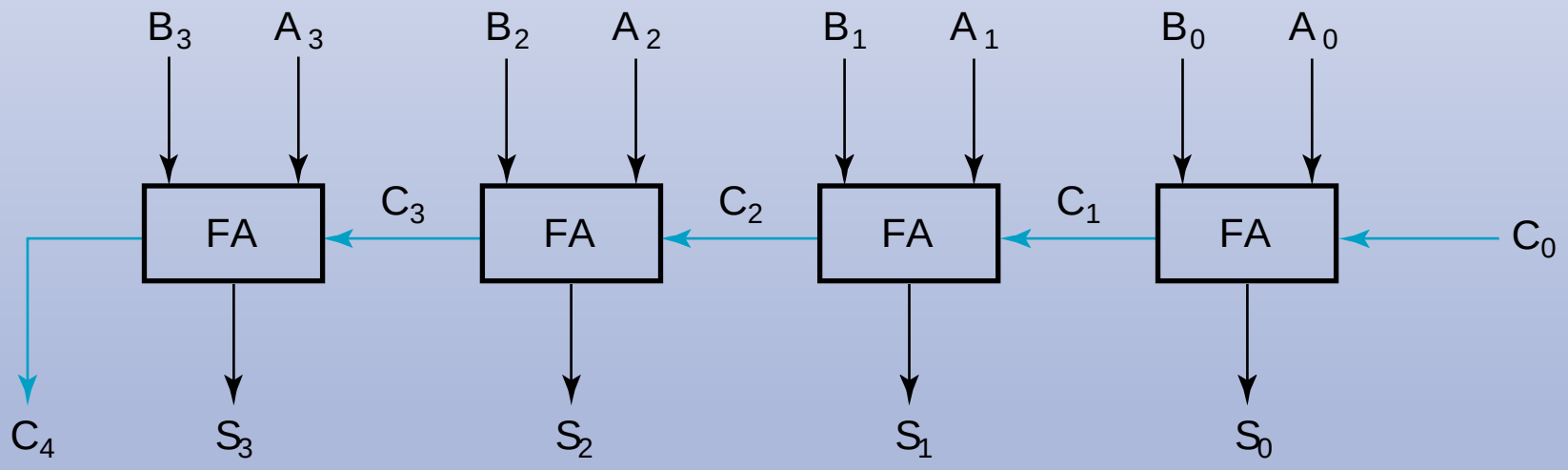
Somadores binários

- Para somar vários operandos, juntamos os sinais lógicos em vetores e usamos blocos funcionais que operam nos vetores
- Examplo: Somador de 4 bits com ripple carry: Soma as entradas A(3:0) e B(3:0) obtendo um vetor soma S(3:0)
- Nota: A saída carry da célula i torna-se a entrada carry da célula i + 1

Descrição	Subscrito 3 2 1 0	Nome
Carry In	0 1 1 0	C_i
Parcela	1 0 1 1	A_i
Parcela	0 0 1 1	B_i
Soma	1 1 1 0	S_i
Saída de Carry	0 0 1 1	C_{i+1}

Somador binário de 4 bit tipo Ripple-Carry

- Um somador binário ripple-carry de 4 bits é feito a partir de quatro somadores completos de 1-bit:



Module

- Definição geral

```
module module_name ( port_list );
  ...
  declaracao de variaveis;
  ...
  descricao de comportamento;
endmodule
```

Meio somador

```
module half_adder (
   input logic x, y,
   output logic s, c
);
   assign s = x ^ y;
   assign c = x & y;
endmodule
```

Convenções Léxicas

- Comentários

  // comentário de uma linha só

  /* outra forma de comentário de uma linha */

  /* inicio de comentário com múltiplas linhas

  todo texto é ignorado

  termina com esta linha abaixo */

- Números (constantes)

  decimal, hexadecimal, binário

  com tamanho e base: 6'd33, 8'h6A, 4'b1101

  sem tamanho: 'h6A

  decimal sem tamanho sem base: 33 (decimal 32 bits)

Convenções Léxicas (cont)

- Identificador
  - A ... Z
  - a ... z
  - 0 ... 9
  - Sublinhado (underscore _)

- Primeiro caractere de um identificador não pode ser um dígito

- **SystemVerilog diferencia letras maiúscula de minúsculas**

Somador completo a partir de meio somador

```
module full_adder (
  input  logic x, y, z,
  output logic s, c
);

logic hs, hc, tc;

half_adder HA1 (x, y, hs, hc),
           HA2 (hs, z, s, tc);
assign c = tc | hc;
endmodule
```

Somador binário 4 bits

```
module adder_4b (
  input  logic [3:0] B,A,
  input  C0,
  output logic [3:0] S,
  output C4
);

  logic [3:1] C;

  full_adder Bit0 (B[0], A[0], C0,   S[0], C[1]),
                  (B[1], A[1], C[1], C[1], C[2]),
                  (B[2], A[2], C[2], S[2], C[3]),
                  (B[3], A[3], C[3], S[3], C4);

endmodule
```

Verilog comportamental do somador de 4 bits

```
module adder_4b (
  input  logic [3:0] B,A,
  input  C0,
  output logic [3:0] S,
  output C4
);

  assign {C4, S} = A + B + C0;

endmodule
```

Instanciando um módulo

- Instâncias de

```
module mymod(output y, input a, b);
```

- Aparecem como

```
// Conecta por posição
mymod mm1 (y1, a1, b1);

// Nomes das instâncias omitidos
mymod (y2, a1, b1), (y3, a2, b2);

// conecta por nome
mymod mm2 (.a(a2)), .b(b2), .y(c2));
```

Diagrama de Blocos

Diagramas de bloco

- Utiliza blocos lógicos, chamados em outras áreas da engenharia de blocos caixa-preta. Não interessa o quem tem no interior.
- Indica o fluxo de sinais de dados e a interface com os sinais de controle

Separação entre caminho de dados (datapath) e controle

Modelagem estrutural

O todo e as partes

Vantagens e desvantagens

- Pouco diferente dos métodos tradicionais de projeto
- Principal vantagem a integração com as outras formas de descrição de verilog
- Pode ser simulado em conjunto com módulos em outros níveis
- Usado como formato intermediário

Módulos e instâncias

- Estrutura básica de um módulo SystemVerilog:

```
module mymod (
output logic  out1,
output logic  [3:0] out2,
input  logic  in1,
input  logic  [2:0] in
);
...
endmodule
```

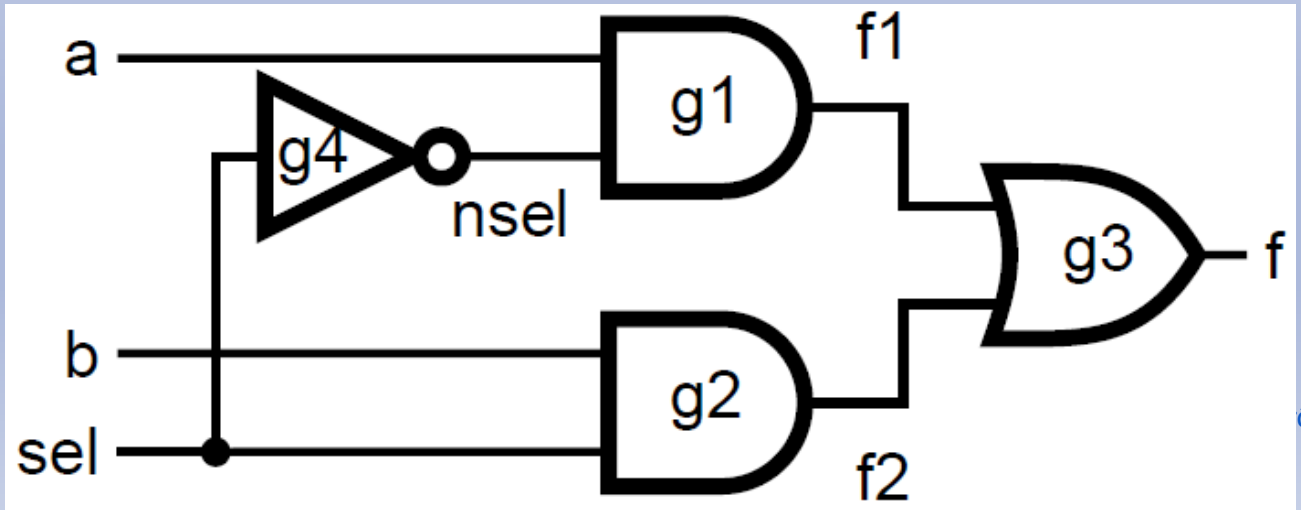
Multiplexador a partir de primitivas

```
module mux(
  output f,
  input a, b, sel
);
logic nsel, f1,f2;
and g1(f1, a, nsel),
    g2(f2, b, sel);
or  g3(f, f1, f2);
not g4(nsel, sel);
endmodule
```

Descrições em SystemVerilog são feitas de módulos

Cada módulo tem uma interface (suas entradas e saídas)

Módulos podem conter estrutura: instâncias de primitivas e outros módulos



Mux com atribuição contínua

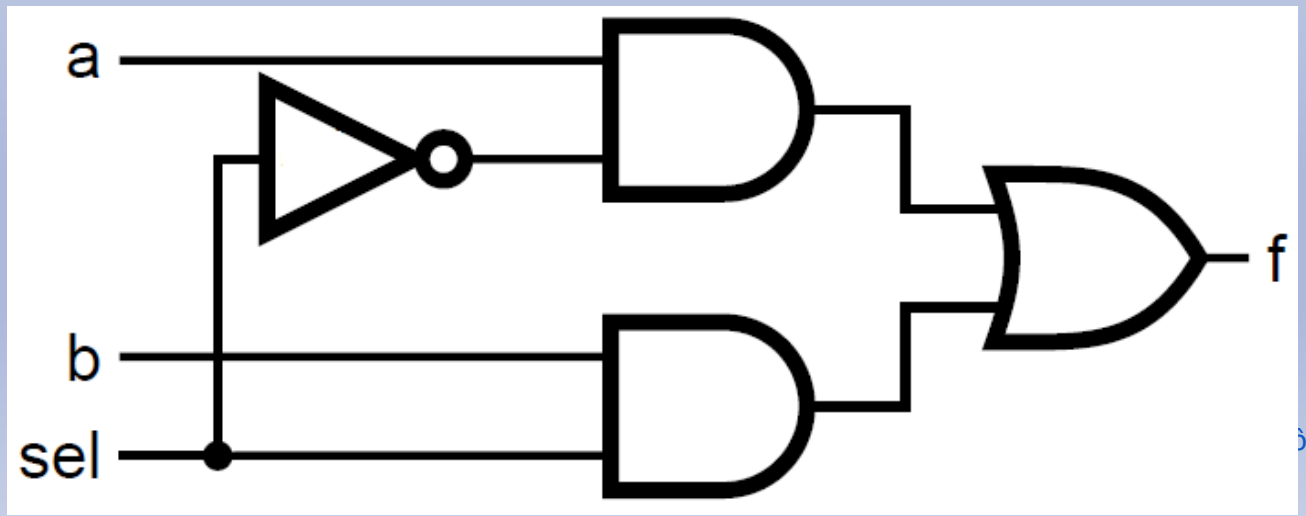
```
module mux(f, a, b, sel);
output logic f;
input logic a, b, sel;

assign f = ~sel ? a : b;

endmodule
```

VLE varia com o VLD, estão "unidos"

Uma mudança no lado direito provoca reavaliação



Atribuição contínua

- Outra forma de descrever uma função combinacional
- Conveniente para especificações lógicas ou de caminho de dados (datapath)

```
logic [8:0] sum;
logic [7:0] a, b;
logic carryin;

assign sum = a + b + carryin;
```

Define a larguras do barramento

Atribuição contínua: permanentemente coloca o valor de sum como sendo a+b+carryin. Recalculado quando a, b ou carryin muda

Operadores a nível de bit Verilog

Bitwise Verilog Operators

Operation	Operator
~	Bitwise NOT
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
^~ or ~^	Bitwise XNOR

Descrição usando combinação

```
module multiplexer_4_to_1_cf_v(S, I, Y);
   input [1:0] S;
   input [3:0] I;
   output Y;

   assign Y = (S == 2'b00) ? I[0] :
              (S == 2'b01) ? I[1] :
              (S == 2'b10) ? I[2] :
              (S == 2'b11) ? I[3] : 1'bx ;
endmodule
```

Descrição usando decisão binária

```
module multiplexer_4_to_1_tf_v(S, I, Y);
   input [1:0] S;
   input [3:0] I;
   output Y;

   assign Y = S[1] ? (S[0] ? I[3] : I[2]) :
                     (S[0] ? I[1] : I[0]) ;
endmodule
```

Verilog 1995, Verilog 2001 e SystemVerilog

- Conexão de instâncias de módulos
  - Portas ordenadas
  - Portas com nome

- dff d1 (out, /*não usado*/, in, clock, reset);

- .<nome_porta>(<nome_variavel_ou_net)

- dff d1 (.q(out), .qb(/* não usado */,
  .d(in), .clk(clock), .rst(reset));

Módulos e instâncias em Verilog2001 e SystemVerilog

- Estrutura básica de um módulo Verilog:

```
module mymod(output out1, output [3:0] out2,
             input in1, input [2:0] in2);


endmodule
```

Conexões .nome e .* implícitas

- Se o nome da porta for igual ao nome do net, pode-se omitir o nome do net (os parênteses)
- Ex: .data equivale a .data (data)

- .* quando todos os nomes forem iguais

- alias clk = clock; (só para net types)

O que vimos hoje

- Projetos digitais modernos (complexos) utilizam linguagens de descrição de hardware
- É possível expressar um sistema digital em diversos níveis de abstração
- O nível estrutural é utilizado para definir hierarquia e conectividade
- SystemVerilog é uma evolução de Verilog

Contato

Marcos Morais
Professor DEE
morais@dee.ufcg.edu.br