Verilog HDL

Adaptação: Mateus Tymburibá

OBS: esta apresentação foca algumas partes da linguagem necessárias para descrever aspectos arquiteturais de computadores. Portanto, somente uma pequena fração da linguagem será coberta.

Verilog HDL

- Introduzida em 1985 pela Gateway Design System Corporation

- Após 1990, passou a ser de domínio público, e em 1995 passou a ser padrão IEEE

Verilog HDL vs. VHDL

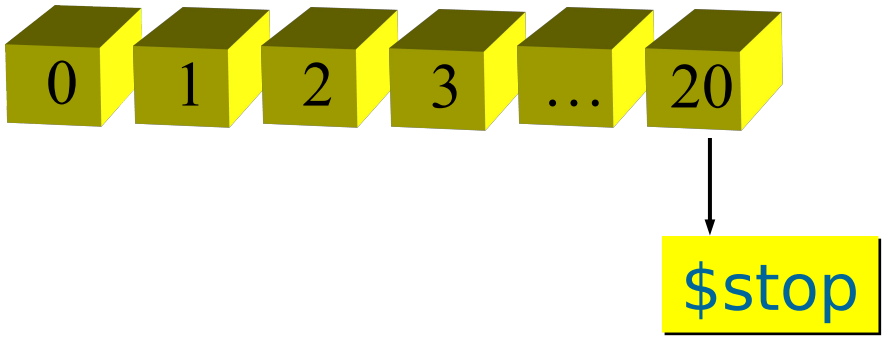
- Verilog HDL é mais próxima a C
- VHDL é mais próxima a ADA
- Considerada mais fácil de aprendizado, pois necessita de menos código para especificar projeto. Há controvérsias!

Níveis de Especificação

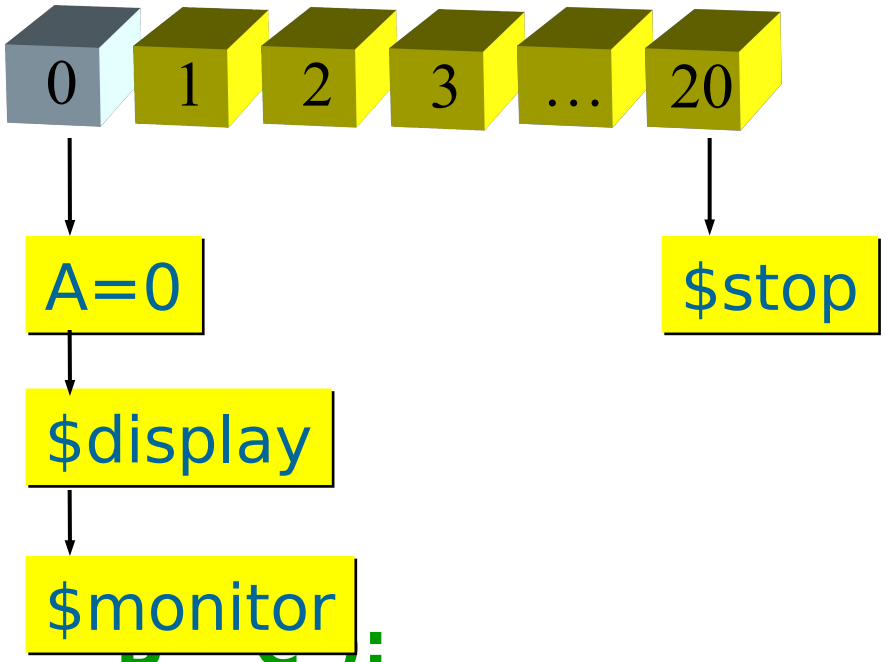
- RTL (Register Transfer Level): descreve a transferência de informações entre registradores.
- Gate level: descreve as portas lógicas e flip-flops em um sistema digital.
- Switch level: descreve o layout de fios, resistores e transistores de um chip de circuito integrado (IC).

```
module simple;
reg [0:7] A, B; // A e B são registradores de 8 bits
reg       C; // C é registrador de 1 bit (flip-flop)
initial begin: stop_at //executa no início (em paralelo)
  #20; $stop; // pára execução após 20 simulações
end
initial begin: init //executa no início (em paralelo)
   A = 0; // inicializa A. Demais: “x”.
   $display("Time   A         B    C");// imprime cabeçalho
   // imprime sempre que A, B ou C mudar
   $monitor("  %0d %b %b %b", $time, A, B, C);
end
//sempre executa (em paralelo com demais)
always begin: main_process // nome é opcional
// #1 significa: faça após 1 unidade de simulação
    #1 A = A + 1;
    #1 B[0:3] = ~A[4:7]; // negação bit a bit
    #1 C = &A[6:7]; // and bit a bit com redução
end
endmodule
```

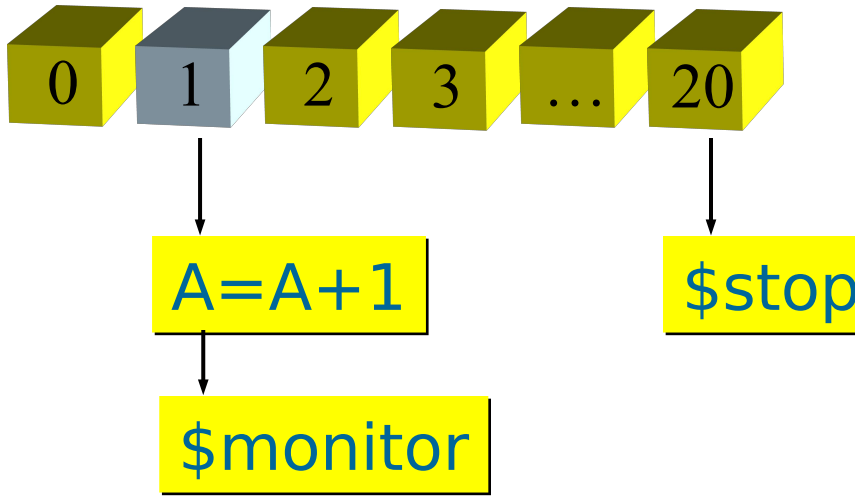
```
module simple;
reg [0:7] A, B;
reg       C;
initial begin: stop_at
  #20; $stop;
end
initial begin: Init
   A = 0;
   $display("Time   A        B    C");
   $monitor("  %0d %b %b %b", $time, A, B, C);
end
always begin: main_process
    #1 A = A + 1;
    #1 B[0:3] = ~A[4:7];
    #1 C = &A[6:7];
end
endmodule
```



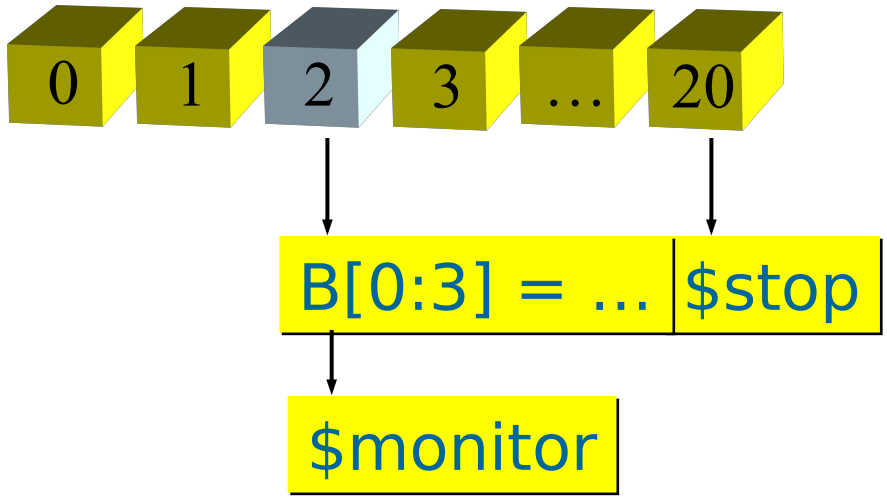
```
module simple;
reg [0:7] A, B;
reg       C;
initial begin: stop_at
  #20; $stop;
end
initial begin: Init
   A = 0;
   $display("Time   A       B   C ");
   $monitor("  %0d %b %b %b", $time, A, B, C);
end
always begin: main_process
    #1 A = A + 1;
    #1 B[0:3] = ~A[4:7];
    #1 C = &A[6:7];
end
endmodule
```



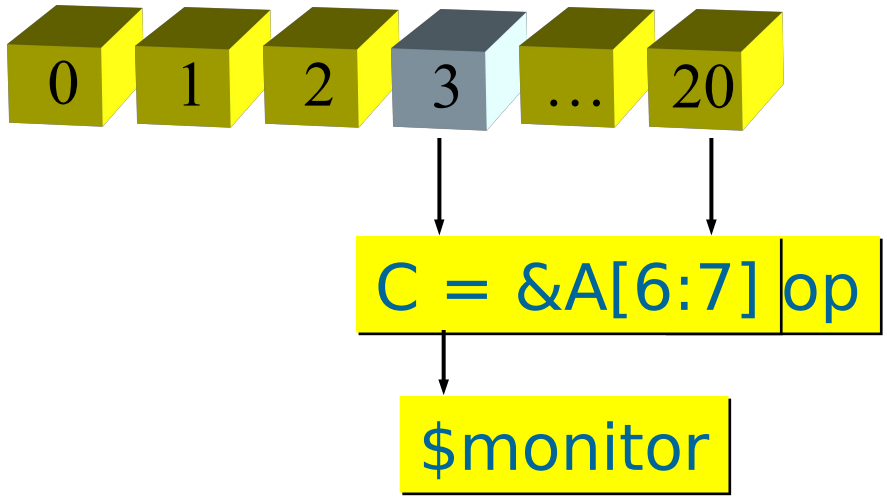
```
module simple;
reg [0:7] A, B;
reg       C;
initial begin: stop_at
  #20; $stop;
end
initial begin: Init
   A = 0;
   $display("Time   A        B    C");
   $monitor("  %0d %b %b %b", $time, A, B, C)
end
always begin: main_process
    #1 A = A + 1;
    #1 B[0:3] = ~A[4:7];
    #1 C = &A[6:7];
end
endmodule
```



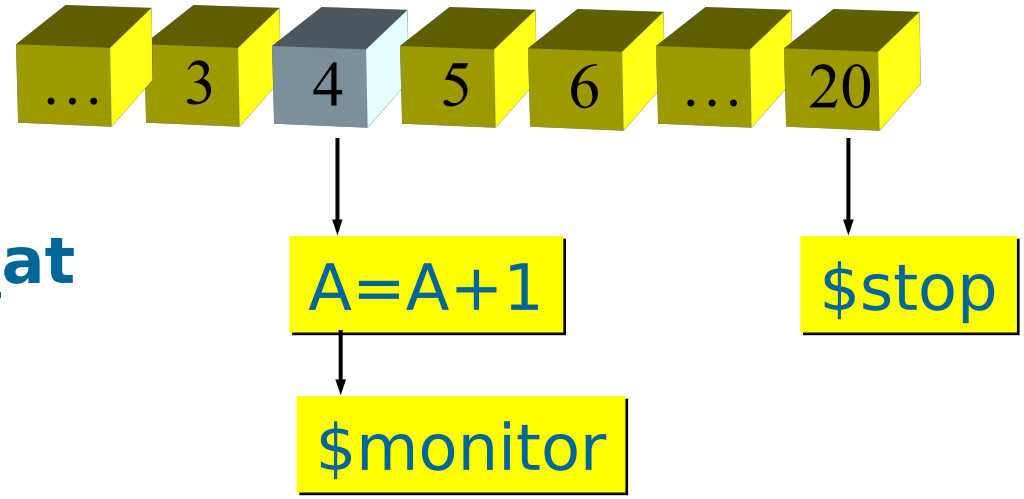
```
module simple;
reg [0:7] A, B;
reg       C;
initial begin: stop_at
  #20; $stop;
end
initial begin: Init
   A = 0;
   $display("Time   A        B    C");
   $monitor("  %0d %b %b %b", $time, A, B, C);
end
always begin: main_process
    #1 A = A + 1;
    #1 B[0:3] = ~A[4:7];
    #1 C = &A[6:7];
end
endmodule
```



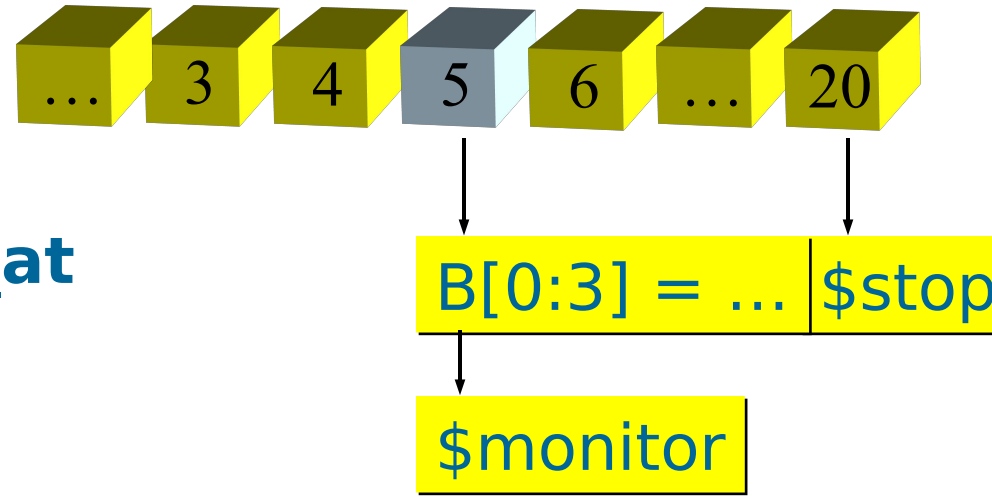
```
module simple;
reg [0:7] A, B;
reg       C;
initial begin: stop_at
  #20; $stop;
end
initial begin: Init
   A = 0;
   $display("Time   A        B    C");
   $monitor("  %0d %b %b %b", $time, A, B, C);
end
always begin: main_process
    #1 A = A + 1;
    #1 B[0:3] = ~A[4:7];
    #1 C = &A[6:7];
end
endmodule
```



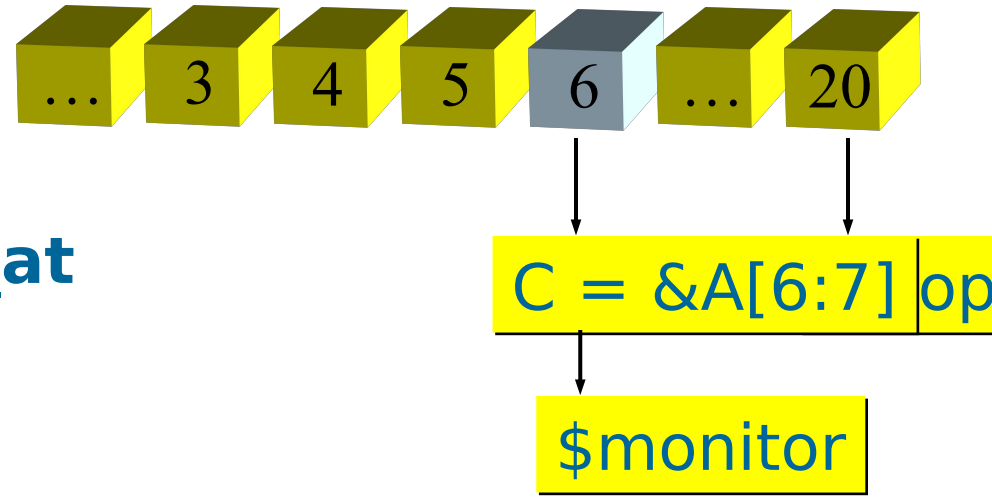
```
module simple;
reg [0:7] A, B;
reg       C;
initial begin: stop_at
  #20; $stop;
end
initial begin: Init
   A = 0;
   $display("Time   A        B    C");
   $monitor("  %0d %b %b %b", $time, A, B, C);
end
always begin: main_process
    #1 A = A + 1;
    #1 B[0:3] = ~A[4:7];
    #1 C = &A[6:7];
end
endmodule
```



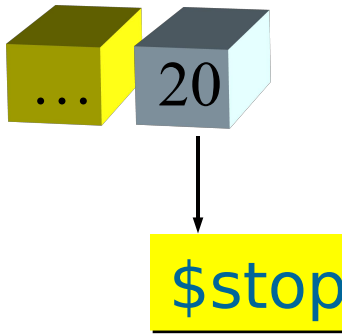
```
module simple;
reg [0:7] A, B;
reg       C;
initial begin: stop_at
  #20; $stop;
end
initial begin: Init
   A = 0;
   $display("Time   A        B    C");
   $monitor("  %0d %b %b %b", $time, A, B, C)
end
always begin: main_process
    #1 A = A + 1;
    #1 B[0:3] = ~A[4:7];
    #1 C = &A[6:7];
end
endmodule
```



```
module simple;
reg [0:7] A, B;
reg       C;
initial begin: stop_at
  #20; $stop;
end
initial begin: Init
   A = 0;
   $display("Time   A        B    C");
   $monitor("  %0d %b %b %b", $time, A, B, C)
end
always begin: main_process
    #1 A = A + 1;
    #1 B[0:3] = ~A[4:7];
    #1 C = &A[6:7];
end
endmodule
```



```
module simple;
reg [0:7] A, B;
reg       C;
initial begin: stop_at
  #20; $stop;
end
initial begin: Init
   A = 0;
   $display("Time   A        B    C");
   $monitor("  %0d %b %b %b", $time, A, B, C)
end
always begin: main_process
    #1 A = A + 1;
    #1 B[0:3] = ~A[4:7];
    #1 C = &A[6:7];
end
endmodule
```



Resultado do Primeiro Exemplo

```
Time   A          B     C
  0 00000000 xxxxxxxx x
  1 00000001 xxxxxxxx x
  2 00000001 1110xxxx x
  3 00000001 1110xxxx 0
  4 00000010 1110xxxx 0
  5 00000010 1101xxxx 0
  7 00000011 1101xxxx 0
  8 00000011 1100xxxx 0
  9 00000011 1100xxxx 1
 10 00000100 1100xxxx 1
 11 00000100 1011xxxx 1
 12 00000100 1011xxxx 0
 13 00000101 1011xxxx 0
 14 00000101 1010xxxx 0
 16 00000110 1010xxxx 0
 17 00000110 1001xxxx 0
 19 00000111 1001xxxx 0
Stop at simulation time 20
```

Descrição em Verilog HDL

- Sistema digital: descrito como um conjunto de módulos.
- Cada módulo possui uma interface para outros módulos → interconexões
- Módulos podem ser executados de forma concorrente.
- Módulo do topo especifica um sistema fechado contendo dados de simulação ou módulo de hardware.
- Módulo do topo ativa instâncias dos outros módulos.

Descrição em Verilog HDL

```
module <module name> (<lista de portas>);
  <declares>
  <module items>
endmodule
```

Tipos de portas (definir nas declarações <declares>):

- input → entrada
- output → saída
- inout → entrada e saída

Descrição em Verilog HDL

```
module <module name> (<port list>);
  <declares>
  <module items>
endmodule
```

- regs e wires → tipos de dados
  - reg: armazena último valor associado a ele (elem. de estado). Só pode receber valor em blocos “initial” e “always”.
  - wire: conexão física entre componentes. Só pode receber valor em assinalamentos contínuos.
  - Valores:

0	zero ou falso lógico
1	um ou verdadeiro lógico
x	valor lógico desconhecido
z	alta impedância de porta tri-state (*don't care*)

Descrição em Verilog HDL

```
module <module name> (<port list>);
  <declares>
  <module items>
endmodule
```

- bloco “initial”: executa quando a simulação inicia (tempo 0)
- bloco “always”: executa sempre que um dos sinais listados mudar (opcionais).

```
always @(lista de sinais) begin
  // expressões de Verilog
end
```

- assinalamentos contínuos (lógica combinacional)
- instâncias de módulos (incluir módulos de outro arquivo)

```
`include "Arquivo.v"
```

Assinalamentos

Dentro de um mesmo bloco “initial” ou “always”:

Blocante (finaliza antes da execução do próximo comando):

- A = 1; // assinalamento sequencial
- B = 0;

Não blocante (assinala o valor ao componente da esquerda somente após todas as expressões à direita, de todos os comandos, terem sido avaliadas):

- A <= 1; // assinalamento paralelo
- B <= 0;

Exemplos

```
                              // lista de portas
module NAND(in1, in2, out);

  // define tipos das portas
  input in1, in2;
  output out;

  // assinalamento contínuo
  assign out = ~(in1 & in2);

endmodule
```

Exemplos

```
module AND(in1, in2, out);
// Porta AND a partir de 2 portas NANDS

  input in1, in2;
  output out;
  wire w1;

  // 2 instâncias do módulo NAND
  NAND NAND1(in1, in2, w1);
  NAND NAND2(w1, w1, out);

endmodule
```

Exemplos

```
//módulo para simulação → não possui entrada nem saída
module test_AND;
  reg a, b;
  wire out1, out2;
  initial begin  // Dados de teste
     a = 0;  b = 0;
     #1 a = 1;
     #1 b = 1;
     #1 a = 0;
  end
  initial begin
    $monitor("Time=%0d a=%b b=%b out1=%b out2=%b",
                    $time, a, b, out1, out2);
  end
  AND  gate1(a, b, out2);
  NAND gate2(a, b, out1);
endmodule
```

Números e Comentários

- 549 // decimal number
- 'h8FF // hex number
- 'o765 // octal number
- 4'b11 // 4-bit binary number 0011
- 3'b10x // 3-bit binary number with least
  /* significant bit unknown */
- 5'd3 // 5-bit decimal number
- -4'b11 // 4-bit two's complement of 0011

Referências e Concatenações

```
initial begin: int1
    A = 8'b01011010;
    B = {A[0:3] | A[4:7], 4'b0000};
end
```

```
C = {2{4'b1011}}; //C = 8'b10111011
C = {{4{A[4]}}, A[4:7]}; // first 4 bits are sign extended
```

```
// contador binário de 4 bits – período: 13
module counter4_bit(q, d, increment, load_data,
                                global_reset, clock);
   output [3:0] q;
   input [3:0] d;
   input  load_data, global_reset, clock, increment;

   reg [3:0]  q;

   // só executa na borda positiva do clock
   always @(posedge clock)
      if (global_reset)
        q = 4'b0000;
      else if (load_data)
        q = d;
      else if (increment) begin
        if (q == 12)
          q = 0;
        else
          q = q + 1;
        end
endmodule // contador de 4 bits
```

Exemplos

Banco de Registradores

```
reg [31:0] Banco [0:1023];
```

```
A = Banco[0];
B = A[3:1];
```

Operações

- Operações binárias (similares a C):

```
?: (condicional)
|| (or lógico)
&& (and lógico)
| (or bit-a-bit)
^ (xor bit-a-bit)
& (and bit-a-bit)
== (igualdade), != (diferença),
===(case), !== (case)
< (menor), <= (menor ou igual),
> (maior), >= (maior ou igual)
<< (shift à esquerda), >> (shift à direita),
+ (adição), - (subtração)
* (multiplicação), / (divisão),
%(módulo - resto da divisão inteira)
```

Operações

- Operações unárias (similares a C):

```
!              (negação lógica)
~              (negação unária bit-a-bit)
&              (redução and unária)
~&             (redução nand unária)
|              (redução or unária)
~|             (redução nor unária)
^              (redução xor unária)
~^ ou ^~       (redução xnor unária)
+              (adição unária)
-              (subtração unária)
```

Fluxo de Controle

```
if (A == 4)
  begin
     B = 2;
  end
else
  begin
     B = 4;
  end
```

```
case (sig) // executa um único caso
  1'bz: $display("Signal is floating");
  1'bx: $display("Signal is unknown");
  default: $display("Signal is %b", sig);
endcase
```

Fluxo de Controle

Bloco de declarações:

```
integer i;
```

Blocos “initial” ou “always”:

```
for(i = 0; i < 10; i = i + 1)
begin
  $display("i= %0d", i);
end
```

```
repeat (5)
begin
    $display("i= %0d", i);
    i = i + 1;
end
```

```
i = 0;
while(i < 10)
begin
    $display("i= %0d", i);
     i = i + 1;
end
```

Controle de Tempo

- Atraso (delay):
  - #10 a = 3; // após 10 unidades de simulação
- Ocorrência de eventos

Na tranição p/ borda negativa do clock (1 → 0):

- @ (negedge clock2) A = B&C;

Se um dos sinais (B ou C) mudar:

- @ (B, C) A = B | C;

Se pelo menos um dos 3 sinais estiver ativo:

- @ (A or B or C) D = A + B + C;

Impressão de dados

```
module test_display; // tarefas de exibição:
initial begin
  $display ("string, variáveis, ou expressão");
  /* formatos similares a printf em C:
       %d=decimal %b=binário %s=string %h=hex %o=octal
       %c=caractere %t=tempo %e=notação científica %f=decimal

  exemplos: %d usa largura default %0d usa largura mínima
                %7.3g usa largura 7 com 3 dígitos após o ponto decimal */

  // $displayb, $displayh, $displayo exibe com formatos b, h, o
  // $write, $strobe, $monitor também têm versões b, h, o

  $write("write");      // Como $display, mas sem newline no fim  da linha
  $strobe("strobe"); // Como $display, mas exibe valores no fim de um
                          // ciclo de simulação

$monitor(“%b %b”, A, B); // Opera quando A ou B tem seu valor alterado
                                 // (exceto quando A e B = $time,$stime,$realtime)

  $monitoron; $monitoroff; // liga/desliga monitoramento
end
endmodule
```

Funções de tempo simulado

Estas funções permitem acessar e exibir o tempo simulado:

$time: inteiro de 64 bits com o valor ajustado para a escala de tempo do módulo chamador

$stime: inteiro de 32 bits com o valor ajustado para a escala de tempo do módulo chamador

$realtime: real com o valor ajustado para a escala de tempo do módulo chamador

Controle de simulação

O controle da simulação é feito por $stop e $finish. Ambos suspendem a simulação, mas $stop retorna o controle para o simulador, enquanto que $finish termina a execução do simulador, e retorna para o sistema operacional.

Um parâmetro opcional pode ser passado para estes comandos, e controla o nível de diagnósticos exibido:

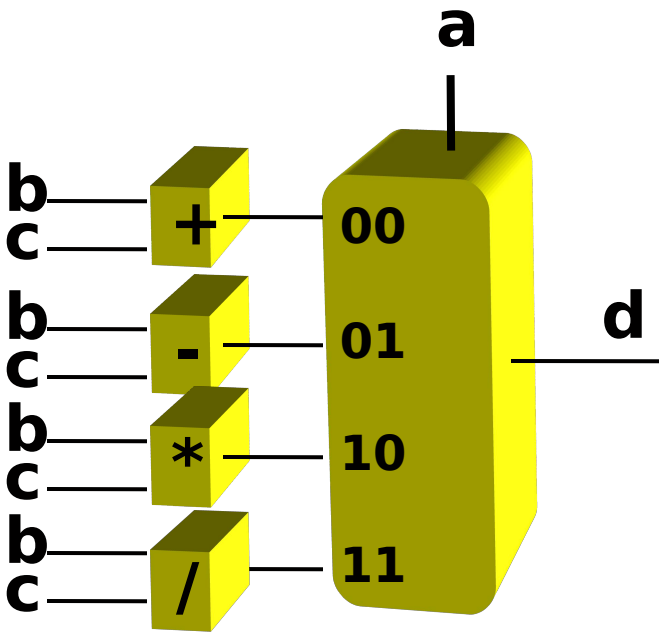
0: nenhum diagnóstico;

1: tempo e localização;

2: tempo, localização e estatísticas.

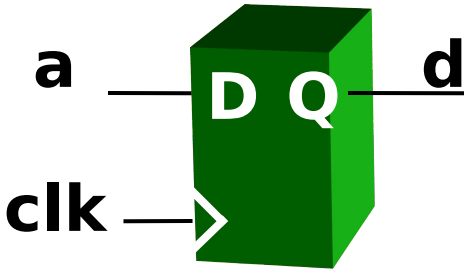
Projeto RTL
Blocos Lógicos Combinacionais

```
always @(a or b or c)
begin
    case (a)
    2'b00: d = b + c;
    2'b01: d = b – c;
    2'b10: d = b * c;
    2'b11: d = b / c;
    endcase
end
```



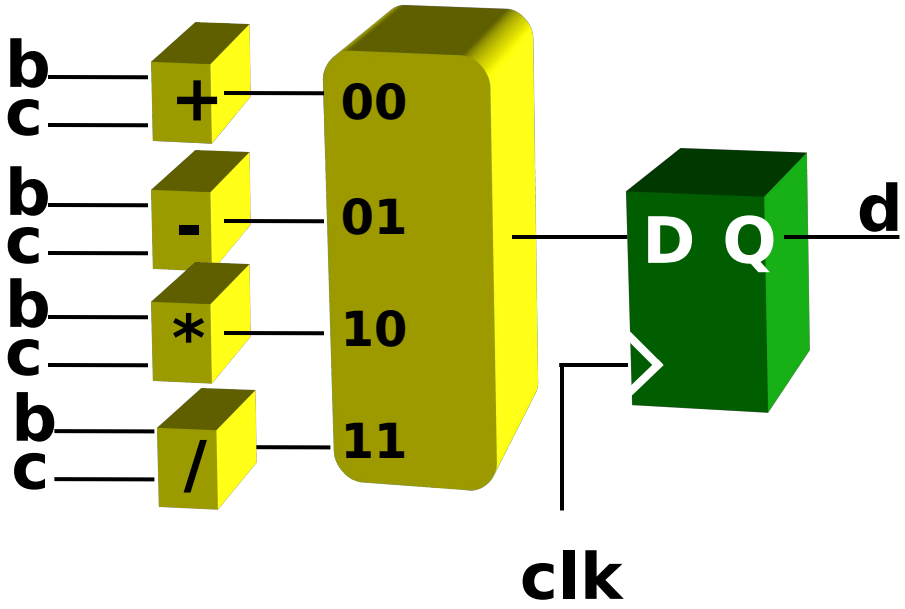
Projeto RTL
Registradores

```
always @ (posedge clk)
  d = a;
```



Projeto RTL
Registradores

```
always @ (posedge clk)
 begin
    case (a)
    2'b00: d = b + c;
    2'b01: d = b – c;
    2'b10: d = b * c;
     2'b11: d = b / c;
    endcase
end
```



Inicialização da “Memória”

Bloco “initial”

Possibilidades:

- Atribuir valores um a um
- Ler valores de um arquivo

Inicialização via arquivo

Usar *$readmemb* e *$readmemh*.

O arquivo a ser lido pode conter espaços, linhas em branco, comentários (// ou /* */), tabulações, saltos de página, endereços (@), números em binário (no caso de *$readmemb*) ou hexadecimal (no caso de *$readmemh*).

Exemplo (“mem.dat”):

```
@2 1010_1111 @4 0101_1111 1010_1111 // @address in hex
x1x1_zzzz 1111_0000 /* x ou z são aceitos */
```

Inicialização via arquivo

O módulo a seguir carrega e exibe estes dados:

```
module load;
  reg [7:0] mem[0:7];
  integer i;
  initial begin
    // start_address=1, end_address=6
    $readmemb("mem.dat", mem, 1, 6);
    for (i= 0; i<8; i=i+1)
      $display("mem[%0d] %b", i, mem[i]);
  end
endmodule
```

Inicialização via arquivo

Resultado:

```
#  ** Warning: $readmem (memory mem) file mem.dat
  line 2:
#    More patterns than index range (hex 1:6)
#    Time: 0 ns  Iteration: 0  Instance:/
# mem[0] xxxxxxxx
# mem[1] xxxxxxxx
# mem[2] 10101111
# mem[3] xxxxxxxx
# mem[4] 01011111
# mem[5] 10101111
# mem[6] x1x1zzzz
# mem[7] xxxxxxxx
```

Inicialização via arquivo – Exem.2

```
module memoria(...);
  … // entradas e saídas
  reg[15:0] mem[0:65535]; // definição do espaço da memoria.
  … // leitura / escrita
endmodule

module simula; // Lê dados iniciais da memória de um arquivo
  …
  memoria Memoria(..); // Cria instância da memória
  initial begin
    // Inicializa a memória com 0
    for(i=0;i<=65535;i=i+1) Memoria.mem[i]=0;
     // Lê dados de arquivo para a memória
    $readmemh("testevibn",Memoria.mem);
  end
  …
endmodule
```

Inicialização via arquivo – Exem.2

Arquivo "testevibn":

@0000
2003
20b7
20b7
20b7
a000
c000
c480
...