Universidade de Brasília
Departamento de Ciência da Computação

Síntese de Sistemas Digitais em FPGA usando Verilog

Prof. Dr. Marcus Vinicius Lamar

Verilog

Altera DE1-SoC

Tópicos

- Introdução ao Verilog
- Introdução ao kit de desenvolvimento DE1-SoC
- Introdução à ferramenta Quartus Prime
- Exemplos Didáticos
- Problemas Práticos
- Conclusão

- Referência Bibliográfica:

Harris, Sarah L. & Harris, David M., **Digital Design and Computer Architecture**, Morgan Kaufmann, 2015;

Linguagem de Descrição de Hardware (HDL)

- Motivação:

  Sistemas Digitais complexos são praticamente impossíveis de estudar e implementar através das técnicas tradicionais de síntese de circuitos digitais usando esquemático ao nível de portas lógicas.

Verilog

- Juntamente com VHDL é uma das mais populares HDLs. Muito usada pela Intel (SytemVerilog).
- Histórico

  1984/85 – criada por Philip Moorby (Gateway Design System Co.) adquirida pela empresa Cadence

  1990 – Open Verilog International : Domínio público

  1995 – Padrão IEEE 1364

  2005 – revisão e atualização do padrão, IEEE 1364

  2012 – IEEE 1800, SystemVerilog, orientação a objetos

- Usada para projetos de circuitos:
  - ASIC (*Application Specific Integrated Circuit*)
  - FPGA (*Field Programmable Gate Array*).

Verilog e SystemVerilog

- Sintaxe similar à linguagem de programação C e C++

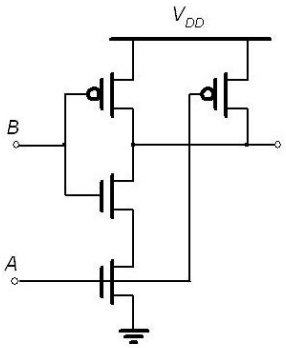
PORÉM...

- HDL não é linguagem de programação!
  - Necessita um processador para ser executada.
- HDL é uma linguagem de descrição de hardware
  - Descreve um circuito

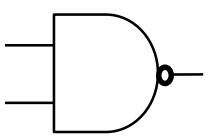
SystemVerilog

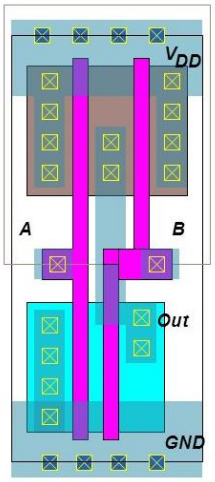
- Níveis de Abstração
  - Nível de Layout (*Layout Level*)
  - Nível de Portas Lógicas (*Gate Level*)
  - Nível de Transferência de Registradores (*Register Transfer Level*)
  - Nível Comportamental (*Behaviour Level*)

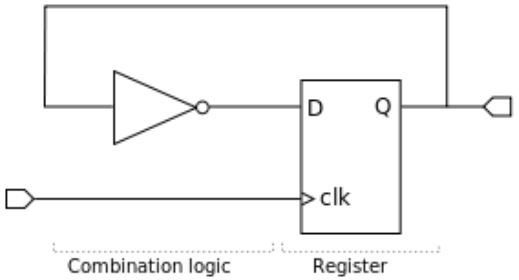
```
assign a = b / c;
```



2-input NAND gate







SystemVerilog – aspectos básicos

- Representação de Números:

  <tamanho em bits>'<base><número> dado armazenado

Ex.: Decimal: ('d' ou 'D')	16'd255	0000000011111111
Hexadecimal ('h' ou 'H')	8'h9A	10011010
Binário ('b' ou 'B')	32'b1010	00000000000000000000000000001010
Octal ('o' ou 'O')	8'o21	00_010_001

Números negativos: -8'd3 = -3 em 8 bits em complemento de 2

- Valores:

níveis lógicos:	0 ou 1	
desconhecido:	x	(nível lógico não conhecido)
alta impedância:	z	(fio não conectado)

Ex.: 32'bz 8'h0x 4'b1z0x

Obs: se o tamanho não for especificado o *default* é 32 bits!

SystemVerilog – aspectos básicos

- Tipos de Dados

- **wire**: define um ou conjunto de fios

Ex.:
```
wire a;          // a é um fio
wire [7:0] b;    /* b é um conjunto de 8 fios {b[7],b[6],..,b[1],b[0]}*/
```

- **reg** : define um registrador (síncrono ou assíncrono)

Ex.:
```
reg a;           // a é um latch ou flip-flop
reg [7:0] b;     /* b é um conjunto de 8 latches ou flip-flops */
```

- **logic :** define um registrador ou wire

Ex.:
```
logic a;         // a é um elemento definido pelo sintetizador
logic [7:0] b;   /* b é um elemento de 8 bits definido pelo
                    sintetizador */
```

- Tipos abstratos (32 bits): **`integer`**, **`real`** (IEEE 754)

SystemVerilog

Características da sintaxe

- Case Sensitive :

  `reset` é diferente de `Reset` e de `RESET`

- Nomes não podem iniciar por números:

  `2mux` é um nome inválido! `mux2` é válido

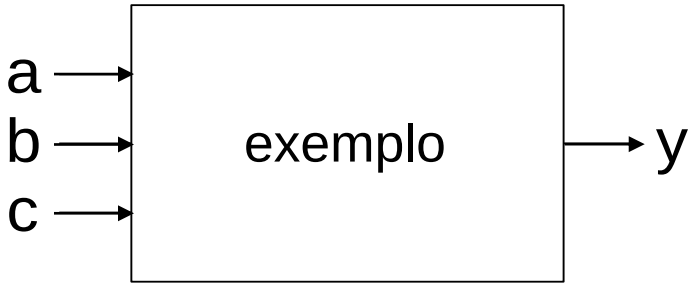
- Espaços em branco são ignorados
- Comentários iguais à linguagem C

```
// comentário de uma linha
/* comentários
   em múltiplas linhas */
```

SystemVerilog

- Estrutura básica de uma descrição Verilog

```
module <nome>(<entradas>, <saídas>);
     <descrição>;
endmodule
```




```
module exemplo(input  logic a, b, c,
               output logic y);
  assign y = ~a & ~b & ~c | a & ~b & ~c | a & ~b &  c;
endmodule
```

SystemVerilog

- Operações Básicas do Verilog e precedência

( )	Indica prioridade
{},{{}}	Concatenação
~	NOT
*, /, %	multiplicação, divisão, módulo
+, -	adição, subtração
<<, >>	deslocamento lógico
<<<, >>>	deslocamento aritmético
<, <=, >, >=	Comparações
==, !=	igual, diferente
&, ~&	AND, NAND
^, ~^	XOR, XNOR
|, ~|	OR, NOR
?:	operador ternário condicional

SystemVerilog

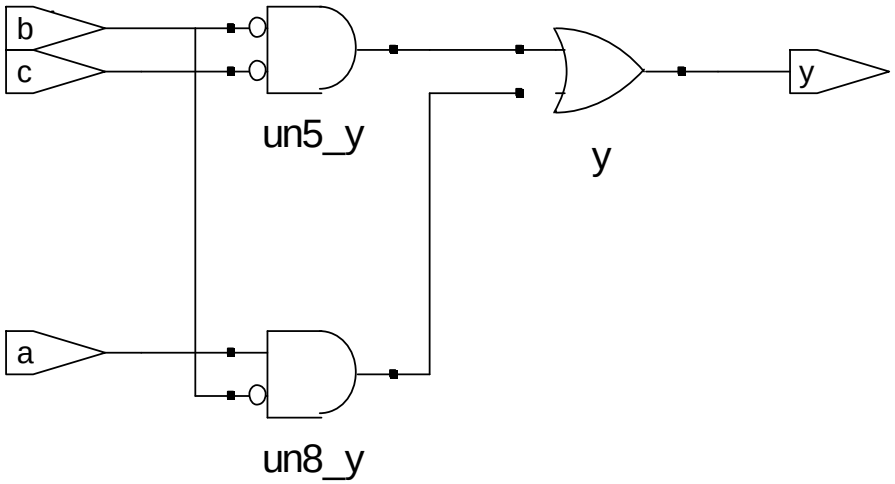
Síntese

A síntese de um circuito corresponde à tradução (compilação) de uma descrição Verilog em uma netlist (descrição das interconexões dos circuitos) pelo uso de uma ferramenta de síntese (Quartus).

Geralmente, durante a compilação, diversas otimizações são realizadas.

```
module exemplo(input  logic a, b, c,
               output logic y);
  assign y = ~a & ~b & ~c | a & ~b & ~c | a & ~b &  c;
endmodule
```

síntese



Atribuições

- Em Verilog e System Verilog temos 3 formas básicas de se definir o valor de um elemento (`wire, reg` ou `logic`).

1) Definindo como um circuito combinacional

```
assign y = ~a & ~b & ~c | a & ~b & ~c;
```

ou

```
always @(*) begin
  y <= ~a & ~b & ~c | a & ~b & ~c; end;
```

Operador atribuição Não-Blocante

ou

```
always @(*) begin
  y = ~a & ~b & ~c;
  y = y | a & ~b & ~c; end;
```

Operador atribuição Blocante

Atribuições

2) Definindo um circuito sequencial

Neste caso, geralmente `y` é do tipo `reg`.

```
always @(posedge clock) begin
      y <= ~a & ~b & ~c| a & ~b & ~c; end;
```

Operador atribuição Não-Blocante

ou

```
always @(posedge clock) begin
      y = ~a & ~b & ~c;
      y = y | a & ~b & ~c; end;
```

Operador atribuição Blocante

FPGA: Intel Cyclone-V 5CSEMA5F31C6

Field Programmable Gate Array

Chip capaz de implementar qualquer sistema digital através da definição da interconexão de seus elementos.

O modelo Cyclone V possui:

- 896 pinos
- 32.070 *Adaptive Logic Module* (ALM)
  - Função Lógica de 8 entradas
  - 4 registradores de 1 bit
  - Somadores encadeáveis (carry)
  - Roteamento
- Memória RAM 4.065.280 bits
- 87 DSP (multiplicador 18x18, 64 add)
- 6 PLL (*Phase Locked Loop*)
- 288 Pinos de IO para o usuário
- ARM Cortex-A9 dual core

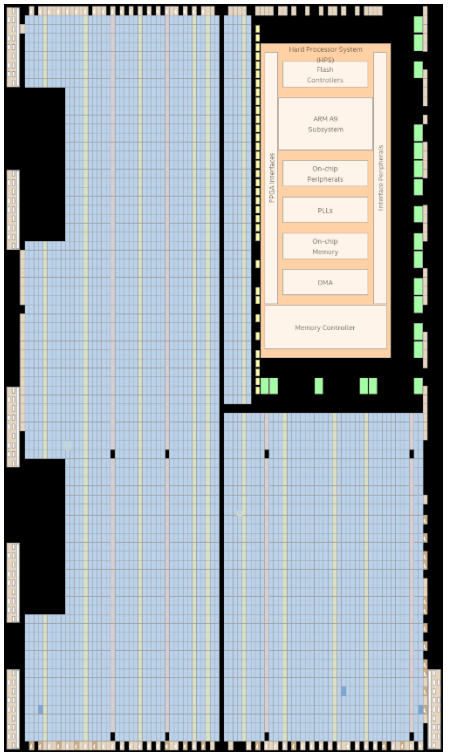
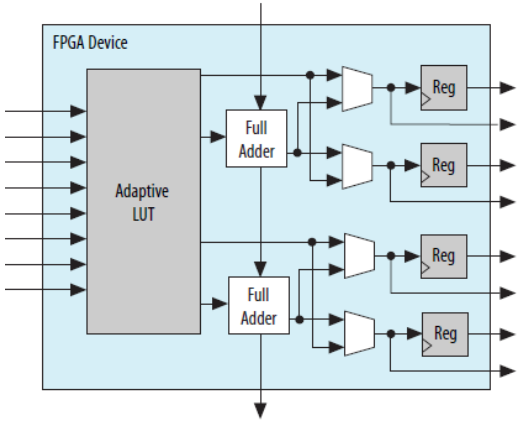
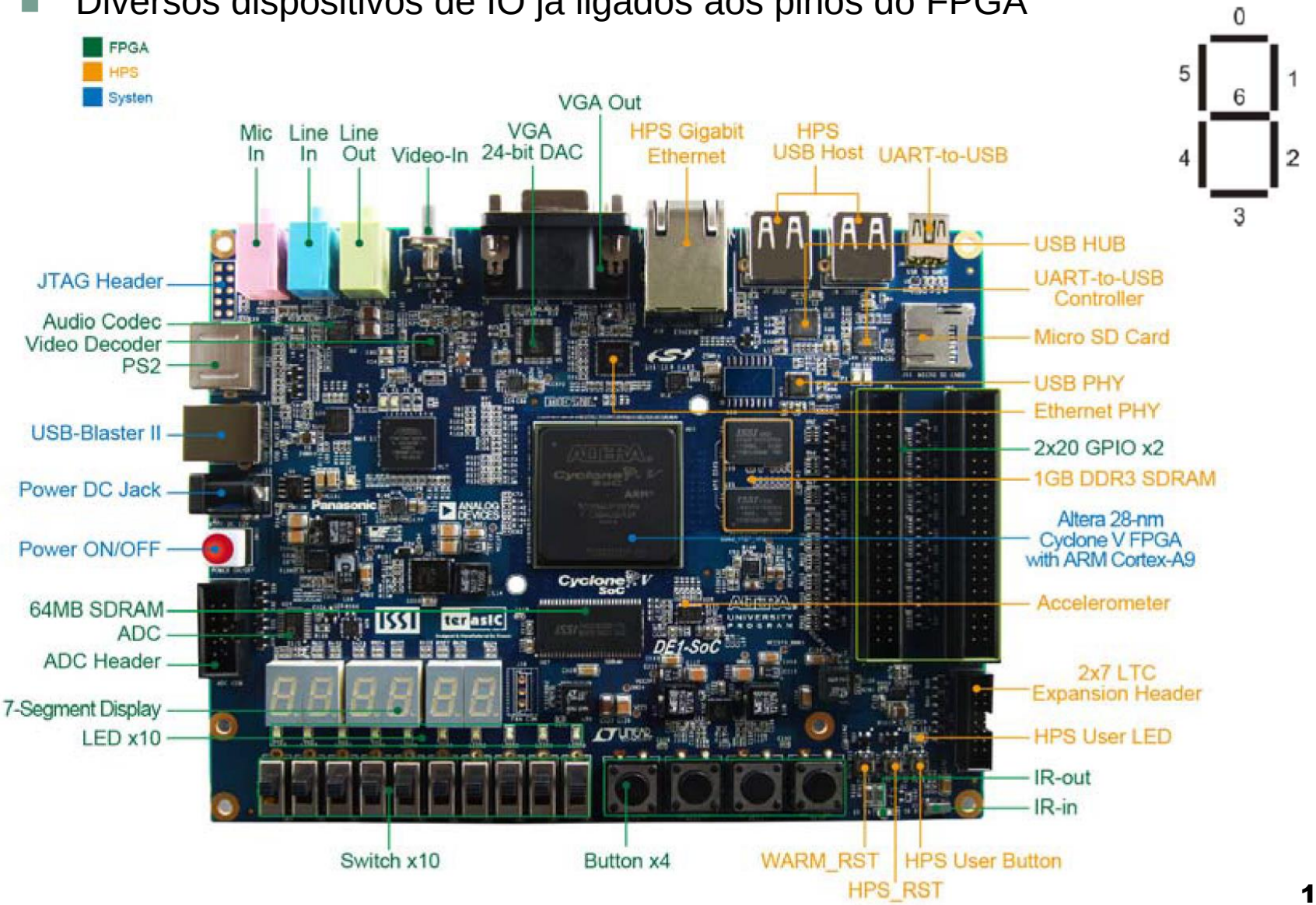


Figure 8: ALM for Cyclone V Devices



Plataforma Intel DE1-SoC

- Diversos dispositivos de IO já ligados aos pinos do FPGA



- Iremos agora realizar um passo a passo para a criação e síntese de dois projetos exemplos didáticos usando o software de Síntese Quartus Prime v18.0

  - Somador de dois números de 4 bits
  - Contador síncrono de frequência variável

- Após serão propostos o desenvolvimento de dois Projetos Aplicativos.

Abrir o arquivo Tutorial_Quartus_Prime.pdf

Exemplo Didático 1

- Somador de 4 bits

Circuito que realiza a soma de 2 números A e B de 4 bits cada.

Metodologia: Projeto hierárquico ao nível de portas lógicas

‘como fazer’

Meio somador de 1 bit

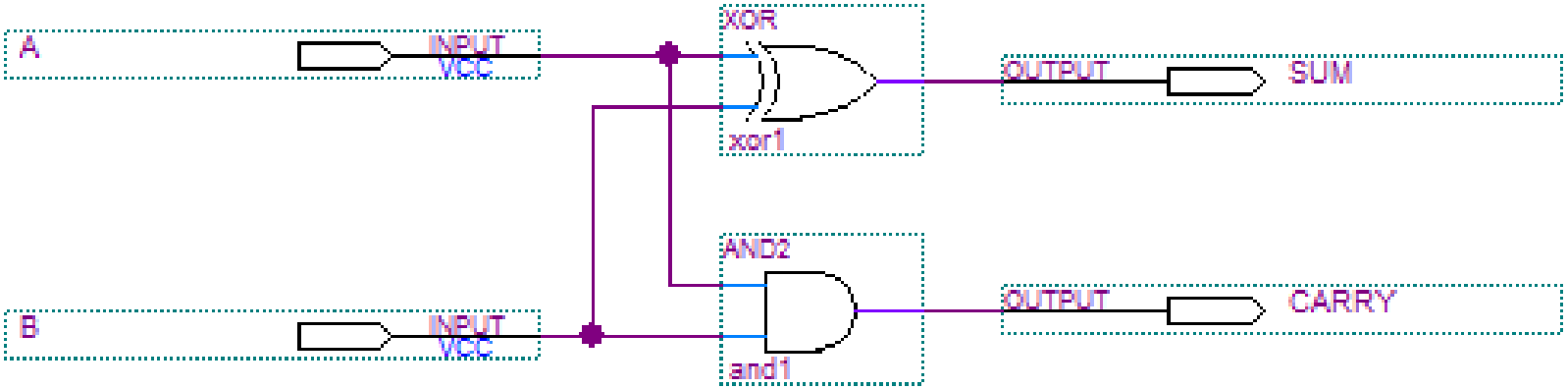
Somador completo de 1 bit

Somador de 4 bits Decodificador para display de 7 segmentos

Implementação na DE1-SoC

Exemplo Didático 1

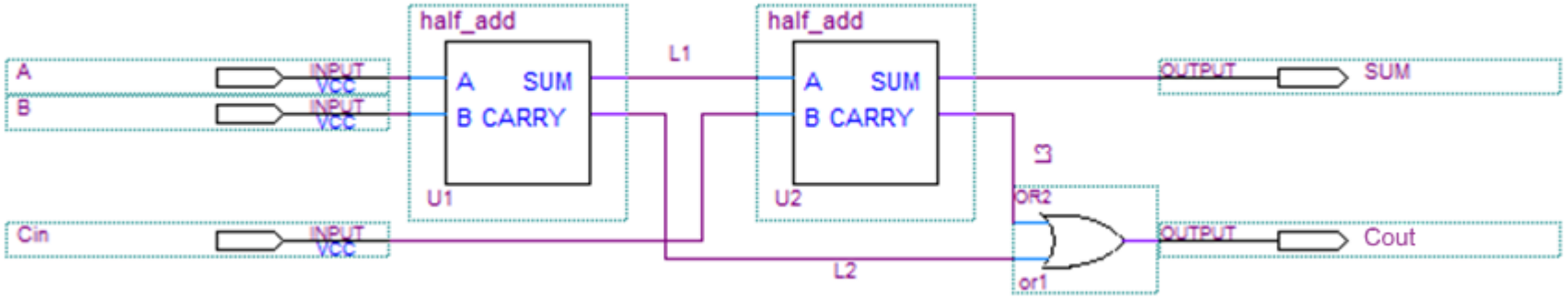
- Somador de 4 bits
  - Meio Somador de 1 bit




```
// Meio Somador de 1 bit
module half_add (
    input logic A,B,
    output logic SUM,CARRY);

    xor XOR1 (SUM, A, B);
    and AND1 (CARRY, A, B);

endmodule
```

Exemplo Didático 1

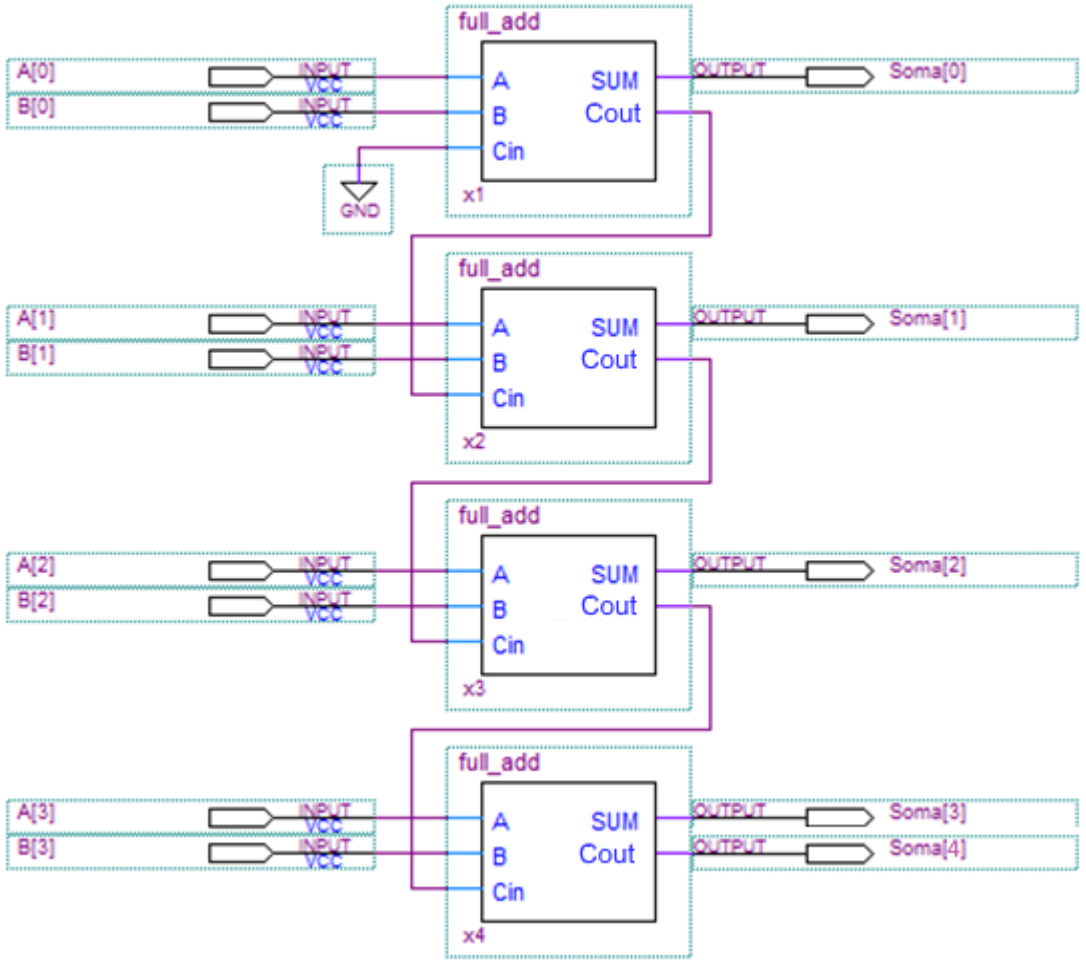
- Somador de 4 bits
  - Somador Completo de 1 bit




```
// Somador completo de 1 bit
module full_add (
   input  logic A,B,Cin,
   output logic SUM,Cout);

   wire L1,L2,L3;

   half_add U1 (.A(A),.B(B),.SUM(L1),.CARRY(L2));
   half_add U2 (.A(L1),.B(Cin),.SUM(SUM),.CARRY(L3));
   or OR1 (Cout,L2,L3);

endmodule
```

Exemplo Didático 1

- Somador de 4 bits
  - Circuito Final




```
// Somador completo de 4 bits
module add4 (
   input  logic [3:0] A,B,
   output logic [4:0] Soma);

   wire L1,L2,L3;

   full_add x1 (.A(A[0]),.B(B[0]),.Cin(1'b0),.SUM(Soma[0]),.Cout(L1));
   full_add x2 (.A(A[1]),.B(B[1]),.Cin(L1),  .SUM(Soma[1]),.Cout(L2));
   full_add x3 (.A(A[2]),.B(B[2]),.Cin(L2),  .SUM(Soma[2]),.Cout(L3));
   full_add x4 (.A(A[3]),.B(B[3]),.Cin(L3),  .SUM(Soma[3]),.Cout(Soma[4]));

endmodule
```

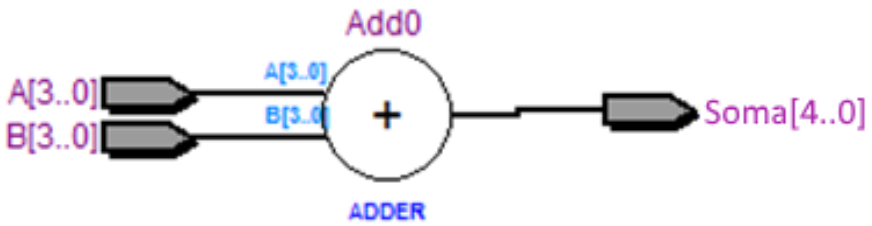
Exemplo Didático 1

- Somador de 4 bits

Circuito que realiza a soma de 2 números A e B de 4 bits cada.

Metodologia: Descrição Comportamental – ‘o que fazer’

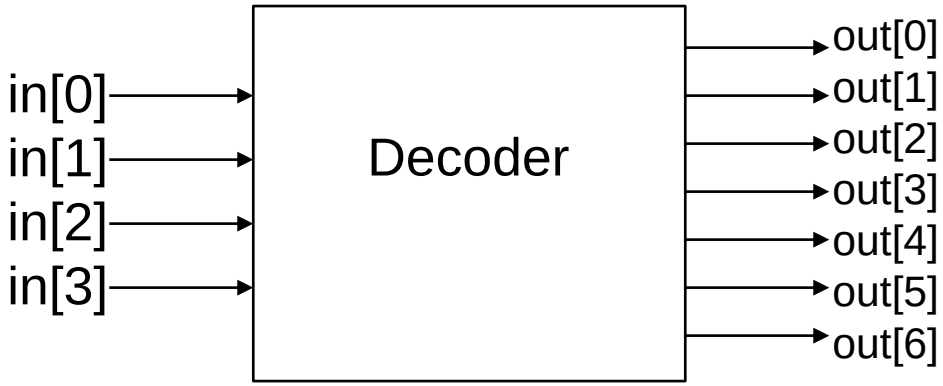
```
// Somador completo de 4 bits comportamental
module add4c (
   input  logic [3:0] A,B,
   output logic [4:0] Soma);

   assign Soma=A+B;

endmodule
```

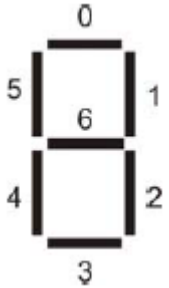


Exemplo Didático 1

- Decodificador de

Display de 7 Segmentos






```
// Decodificador  de binário para
// display 7 segmentos
module decoder7 (
   input  logic [3:0] In,
   output logic [6:0] Out);

   always @(*)
      case (In)
         4'b0000 : Out = ~7'b0111111;
         4'b0001 : Out = ~7'b0000110;
         4'b0010 : Out = ~7'b1011011;
         4'b0011 : Out = ~7'b1001111;

         4'b0100 : Out = ~7'b1100110;
         4'b0101 : Out = ~7'b1101101;
         4'b0110 : Out = ~7'b1111101;
         4'b0111 : Out = ~7'b0000111;

         4'b1000 : Out = ~7'b1111111;
         4'b1001 : Out = ~7'b1101111;
         4'b1010 : Out = ~7'b1110111;
         4'b1011 : Out = ~7'b1111100;

         4'b1100 : Out = ~7'b0111001;
         4'b1101 : Out = ~7'b1011110;
         4'b1110 : Out = ~7'b1111001;
         4'b1111 : Out = ~7'b1110001;

         default : Out = ~7'b0000000;
      endcase

endmodule
```

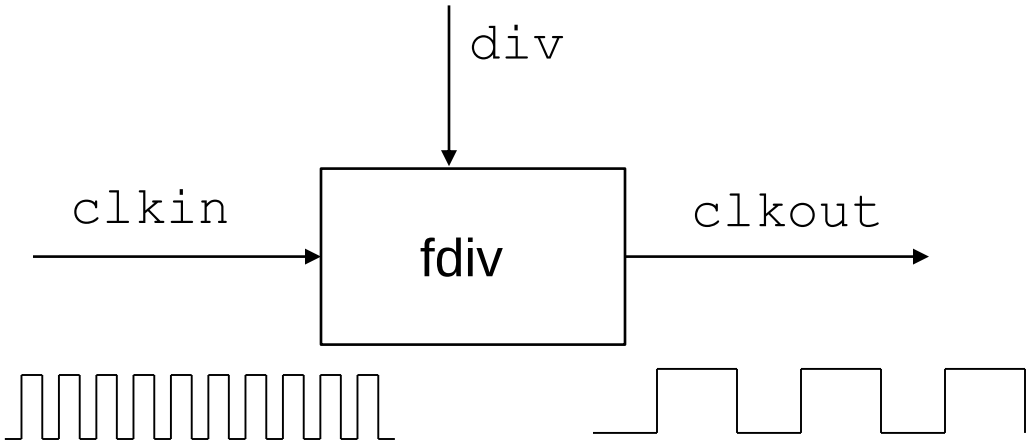
Exemplo Didático 1

- Implementação na DE1-SoC
  - O módulo Top Level define o interfaceamento com os recursos do kit de desenvolvimento

```
// Módulo Top para implementação na DE1-SoC
module exemplo1 (
    input  logic [9:0] SW,
    output logic [9:0] LEDR,
    output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5);

    wire [4:0] saida;

    assign HEX4=~7'b1000110;
    assign HEX2=~7'b1001000;
    assign LEDR={SW[9:6],2'b00,SW[3:0]};

    add4 U0 (.A(SW[9:6]),.B(SW[3:0]),.Soma(saida));

    decoder7 D1 (.In(SW[9:6]),.Out(HEX5));
    decoder7 D2 (.In(SW[3:0]),.Out(HEX3));
    decoder7 D3 (.In({3'b000,saida[4]}),.Out(HEX1));
    decoder7 D4 (.In(saida[3:0]),.Out(HEX0));

endmodule
```

Exemplo Didático 2

- Circuito Divisor de Frequência




```
// Circuito Divisor de Frequência
module fdiv (
   input  logic clkin,
   input  logic [9:0] div,
   input  logic reset,
   output logic clkout);

   integer cont;

   initial
      begin
         clkout = 1'b0;
         cont = 0;
      end

   always @(posedge clkin)
       begin
         if(reset)
            cont <= 0;
         else
           if (cont == {div,16'h1000})
               begin
                   cont <= 0;
                   clkout <= ~clkout;
               end
           else
               cont <= cont + 1;
       end

endmodule
```

Exemplo Didático 2

- Implementação na DE1-SoC

```
// Módulo Top para implementação na DE1-SoC
module exemplo2 (
   input  logic CLOCK_50,      // Sinal de 50MHz
   input  logic [9:0] SW,      // chaves
   input  logic [3:0] KEY,     // Keys
   output logic [9:0] LEDR,    // LEDs Vermelhos
   output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5); // Displays

   wire clock; // fio com o clock de saída

   integer contador;         // contador de 32 bits
   initial contador = 0;     // inicializa o contador com 32'b0

   always @(posedge clock or negedge KEY[0])
      if (KEY[0] == 1'b0)
         contador <= 0;      // Reset do contador
      else
         contador <= contador+1; // Incremento do contador

   assign LEDR = contador[9:0];  /* mostra nos LEDs Vermelhos os bits menos
                                    significativos do contador*/

   fdiv  U0 (.clkin(CLOCK_50),.div(SW[9:0]),.reset(~KEY[0]),.clkout(clock));

   decoder7 D0 (.In(contador[ 3: 0]), .Out(HEX0));
   decoder7 D1 (.In(contador[ 7: 4]), .Out(HEX1));
   decoder7 D2 (.In(contador[11: 8]), .Out(HEX2));
   decoder7 D3 (.In(contador[15:12]), .Out(HEX3));
   decoder7 D4 (.In(contador[19:16]), .Out(HEX4));
   decoder7 D5 (.In(contador[23:20]), .Out(HEX5));

endmodule
```

Projeto 1

Implemente um circuito digital combinacional de

10 entradas I[0], I[1], I[2], I[3], ..., I[9] e

3 saídas S[0], S[1], S[2], onde:

- S[0]=1 se a maioria das entradas for 1
- S[1]=1 se o número de 0s e 1s forem iguais
- S[2]=1 se a maioria das entradas for 0

Use as chaves SW como as entradas

```
assign I[9:0]=SW[9:0];
```

Use os LEDs Vermelhos como as saídas

```
assign LEDR[2:0]=S[2:0];
```

Metodologia Clássica: Tabela Verdade => Circuito Digital

Dica: Use `+` com `if else`

Projeto 2

Implemente um circuito digital sequencial em que os LEDs imitem os movimentos dos ‘olhos’ dos Cylonios da série *BattleStar Galactica*.

A velocidade do LEDR deve ser definida pelas chaves SW[9:0].

Dica: Use o módulo `fdiv.v` como base de seu projeto.

Projeto 2 - Desafio

Melhore a movimentação dos olhos dos Cylonios ou da Super-Máquina, de modo que quatro LEDs brilhem com intensidades diferentes, deixando um rastro, de acordo com as figuras abaixo.

Dica: A luminosidade de um LED pode ser controlada pelo seu duty cycle!

Conclusões

- As HDLs são o estado-da-tecnologia no desenvolvimento industrial de processadores e muito usadas em pesquisas científicas em Arquitetura de Computadores.
- Verilog é uma linguagem fácil, porém apresenta inúmeros detalhes que não foram vistos aqui.
- Este é um curso extremamente básico e inicial.
- Busquem tutoriais na internet para aprofundar mais o conhecimento e técnicas eficientes de implementação.

http://www.asic-world.com/verilog/veritut.html