Aulas 8 e 9
Decodificadores

Autores	
Gabriel A. F. Souza, Gustavo D. Colletta, Leonardo B. Zoccal, Odilon O. Dutra	Unifei

Histórico de Revisões		
19 de fevereiro de 2025	1.0	Primeira versão do documento.

Tópicos

- Introdução
- Descrição em Verilog
- Tipos de Decodificador
- Simulação e Verificação
- Atividades Hands-on

Introdução

Objetivos

- Compreender o conceito de circuitos decodificadores.
- Explorar suas aplicações na eletrônica digital.
- Apresentar estruturas para descrição em Verilog.

Definição

- Definição: Um circuito decodificador é um dispositivo que converte informação de um formato para outro, especificamente de um código de entrada para uma saída ativa.
- Função: Traduzir um padrão binário e gerar um sinal correspondente a uma das possíveis combinações desse código.

Conversores de Código vs Decodificadores

- Conversores de Código: Traduzem uma informação de um código para outro.
- Decodificadores: Recebem um código de entrada, gerando apenas uma saída ativa por vez.

Os decodificadores podem ser considerados um caso particular de conversor de código

O que são Decodificadores?

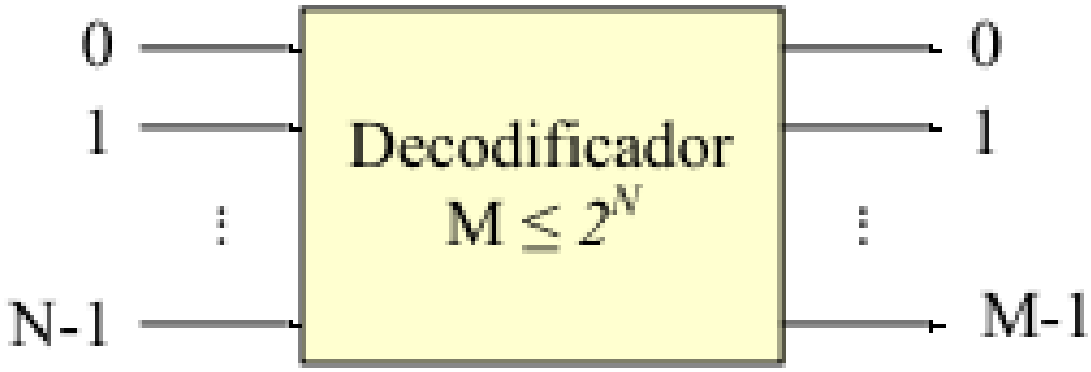

Figura 1: Circuito Decodificador

- Decodifica N linhas de entrada em M linhas de saída.
- Apenas uma linha de saída está ativa por vez.

Aplicações

- Endereçamento de Memória
  - Usados em RAMs e ROMs para selecionar células de memória específicas com base no endereço fornecido pelo processador.
- Seleção de Barramentos
  - Utilizados para ativar diferentes dispositivos em um barramento de comunicação, garantindo que apenas um deles responda a um determinado endereço.
- Computadores e Processadores
  - Usados no decodificador de instruções para traduzir códigos de máquina em sinais de controle internos para a CPU.
- Controle de Ativação de Periféricos
  - Usado em microcontroladores e microprocessadores para ativar dispositivos específicos, como sensores, atuadores e portas de comunicação.

Descrição em Verilog

Tipos de Descrição

- **Fluxo de Dados (Dataflow)**
- **Estrutural (Structural)**
- **Comportamental (Behavioral)**

Tipos de Decodificador

Decodificadores

- Decodificador Binário Simples
- Decodificador de código BCD8421
- Decodificador de código Excesso de 3
- “Decodificador” para Display de 7 segmentos*

Decodificador Binário

- Converte um código binário de N bits de entrada em M saídas correspondentes, sendo apenas uma ativa por vez

$$N = log_2 M$$

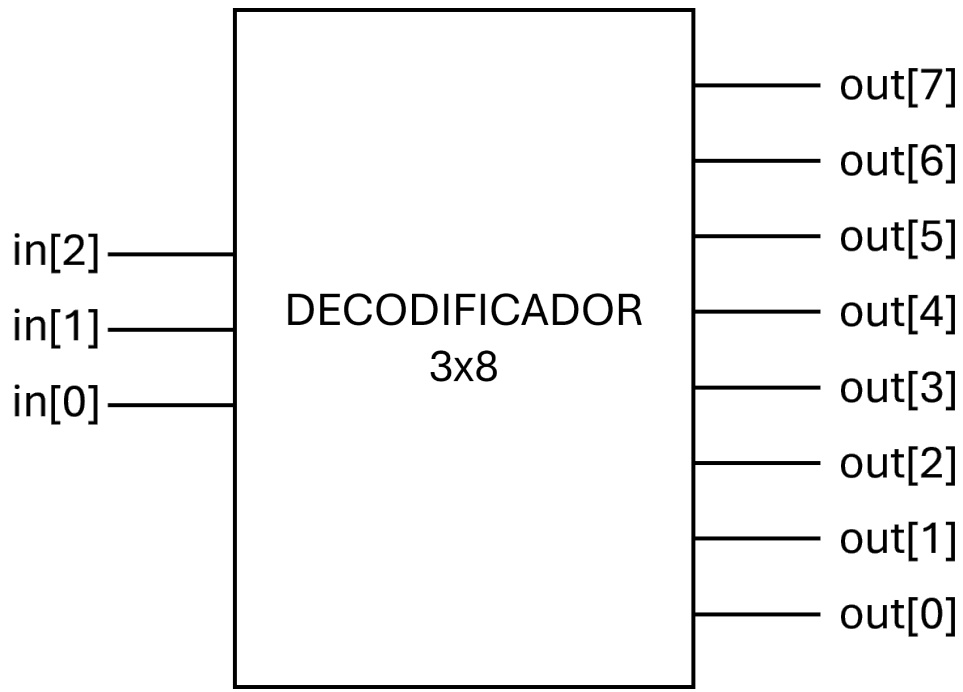

Figura 2: Exemplo de Decodificador 3x8

Tabela Verdade

Tabela de um decodificador binário de 3 para 8 linhas.

Entrada (in[2:0])	Saída (out[7:0])
000	00000001
001	00000010
010	00000100
011	00001000
100	00010000
101	00100000
110	01000000
111	10000000

Expressões Lógicas

As saídas `out[7:0]` são obtidas a partir das entradas `in[2:0]` usando as seguintes expressões lógicas:

$$\mathsf{out}[0] = \overline{\mathsf{in}[2]} \cdot \overline{\mathsf{in}[1]} \cdot \overline{\mathsf{in}[0]}$$

$$\mathsf{out}[1] = \overline{\mathsf{in}[2]} \cdot \overline{\mathsf{in}[1]} \cdot \mathsf{in}[0]$$

$$\mathsf{out}[2] = \overline{\mathsf{in}[2]} \cdot \mathsf{in}[1] \cdot \overline{\mathsf{in}[0]}$$

$$\mathsf{out}[3] = \overline{\mathsf{in}[2]} \cdot \mathsf{in}[1] \cdot \mathsf{in}[0]$$

Expressões Lógicas

$$\mathsf{out}[4] = \mathsf{in}[2] \cdot \overline{\mathsf{in}[1]} \cdot \overline{\mathsf{in}[0]}$$

$$\mathsf{out}[5] = \mathsf{in}[2] \cdot \overline{\mathsf{in}[1]} \cdot \mathsf{in}[0]$$

$$\mathsf{out}[6] = \mathsf{in}[2] \cdot \mathsf{in}[1] \cdot \overline{\mathsf{in}[0]}$$

$$\mathsf{out}[7] = \mathsf{in}[2] \cdot \mathsf{in}[1] \cdot \mathsf{in}[0]$$

Descrição Fluxo de Dados

```
1  module decoder_3x8 (
2      input  [2:0] in,  // Entradas in[2], in[1], in[0]
3      output [7:0] out  // Saídas out[7:0]
4  );
5
6      assign out[0] = ~in[2] & ~in[1] & ~in[0];
7      assign out[1] = ~in[2] & ~in[1] &  in[0];
8      assign out[2] = ~in[2] &  in[1] & ~in[0];
9      assign out[3] = ~in[2] &  in[1] &  in[0];
10     assign out[4] =  in[2] & ~in[1] & ~in[0];
11     assign out[5] =  in[2] & ~in[1] &  in[0];
12     assign out[6] =  in[2] &  in[1] & ~in[0];
13     assign out[7] =  in[2] &  in[1] &  in[0];
14
15 endmodule
```

Descrição Comportamental

```
1  module decoder_3x8_comp (
2      input  [2:0] in,  // Entradas in[2], in[1], in[0]
3      output reg [7:0] out  // Saídas out[7:0]
4  );
5      always @(*) begin
6          case (in)
7              3'b000: out = 8'b00000001;
8              3'b001: out = 8'b00000010;
9              3'b010: out = 8'b00000100;
10             3'b011: out = 8'b00001000;
11             3'b100: out = 8'b00010000;
12             3'b101: out = 8'b00100000;
13             3'b110: out = 8'b01000000;
14             3'b111: out = 8'b10000000;
15             default: out = 8'b00000000;
16         endcase
17     end
18 endmodule
```

Decodificador de código BCD8421

- Converte um código BCD de 4 bits de entrada em 10 saídas correspondentes.

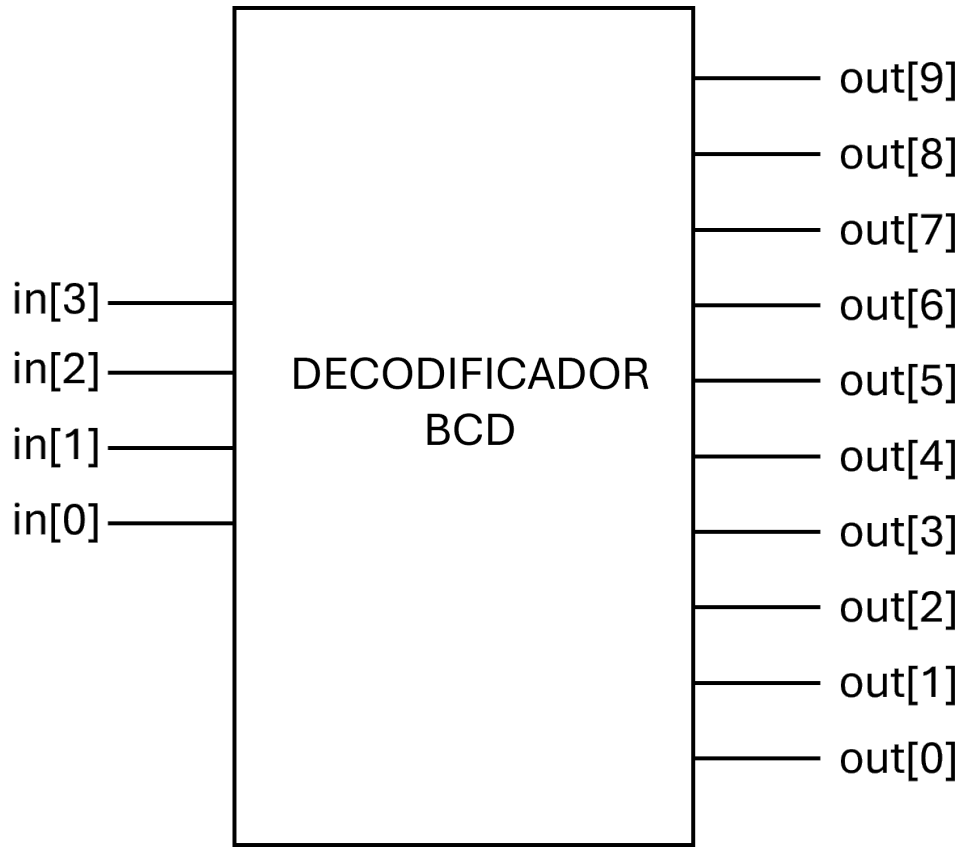

Figura 3: Exemplo de Decodificador BCD8421

Tabela Verdade

Decodificador BCD para 10 Linhas

Entrada BCD (in[3:0])	Saída (out[9:0])
0000	0000000001
0001	0000000010
0010	0000000100
0011	0000001000
0100	0000010000
0101	0000100000
0110	0001000000
0111	0010000000
1000	0100000000
1001	1000000000

Tabela Verdade

Alternativa: Decodificador BCD para 10 Linhas **Ativo Baixo**

Entrada BCD (in[3:0])	Saída (out[9:0])
0000	1111111110
0001	1111111101
0010	1111111011
0011	1111110111
0100	1111101111
0101	1111011111
0110	1110111111
0111	1101111111
1000	1011111111
1001	0111111111

Expressões Lógicas

Expressões para o circuito ativo baixo:

$$\overline{\text{out}[0]} = \overline{\text{in}[3]} \cdot \overline{\text{in}[2]} \cdot \overline{\text{in}[1]} \cdot \overline{\text{in}[0]}$$

$$\overline{\text{out}[1]} = \overline{\text{in}[3]} \cdot \overline{\text{in}[2]} \cdot \overline{\text{in}[1]} \cdot \text{in}[0]$$

$$\overline{\text{out}[2]} = \overline{\text{in}[3]} \cdot \overline{\text{in}[2]} \cdot \text{in}[1] \cdot \overline{\text{in}[0]}$$

$$\overline{\text{out}[3]} = \overline{\text{in}[3]} \cdot \overline{\text{in}[2]} \cdot \text{in}[1] \cdot \text{in}[0]$$

$$\overline{\text{out}[4]} = \overline{\text{in}[3]} \cdot \text{in}[2] \cdot \overline{\text{in}[1]} \cdot \overline{\text{in}[0]}$$

Expressões Lógicas

$$\overline{\mathsf{out}[5]} = \overline{\mathsf{in}[3]} \cdot \mathsf{in}[2] \cdot \overline{\mathsf{in}[1]} \cdot \mathsf{in}[0]$$

$$\overline{\mathsf{out}[6]} = \overline{\mathsf{in}[3]} \cdot \mathsf{in}[2] \cdot \mathsf{in}[1] \cdot \overline{\mathsf{in}[0]}$$

$$\overline{\mathsf{out}[7]} = \overline{\mathsf{in}[3]} \cdot \mathsf{in}[2] \cdot \mathsf{in}[1] \cdot \mathsf{in}[0]$$

$$\overline{\mathsf{out}[8]} = \mathsf{in}[3] \cdot \overline{\mathsf{in}[2]} \cdot \overline{\mathsf{in}[1]} \cdot \overline{\mathsf{in}[0]}$$

$$\overline{\mathsf{out}[9]} = \mathsf{in}[3] \cdot \overline{\mathsf{in}[2]} \cdot \overline{\mathsf{in}[1]} \cdot \mathsf{in}[0]$$

Descrição Fluxo de Dados

```
1  module decoder_bcd_10 (
2      input  [3:0] in,   // Entrada BCD
3      output [9:0] out   // Saídas ativas em nível baixo
4  );
5
6      assign out[0] = ~(~in[3] & ~in[2] & ~in[1] & ~in[0]);
7      assign out[1] = ~(~in[3] & ~in[2] & ~in[1] &  in[0]);
8      assign out[2] = ~(~in[3] & ~in[2] &  in[1] & ~in[0]);
9      assign out[3] = ~(~in[3] & ~in[2] &  in[1] &  in[0]);
10     assign out[4] = ~(~in[3] &  in[2] & ~in[1] & ~in[0]);
11     assign out[5] = ~(~in[3] &  in[2] & ~in[1] &  in[0]);
12     assign out[6] = ~(~in[3] &  in[2] &  in[1] & ~in[0]);
13     assign out[7] = ~(~in[3] &  in[2] &  in[1] &  in[0]);
14     assign out[8] = ~( in[3] & ~in[2] & ~in[1] & ~in[0]);
15     assign out[9] = ~( in[3] & ~in[2] & ~in[1] &  in[0]);
16
17 endmodule
```

Descrição comportamental

- Ponto de Atenção na descrição comportamental!
  - Assim como nas outras descrições de conversores de código, se não houver um caso “default” serão inferidos latches indesejados.

Descrição Comportamental

```
1  module decoder_bcd_10_comp (
2      input  [3:0] in,   // Entrada BCD
3      output reg [9:0] out   // Saídas ativas em nível baixo
4  );
5      always @(*) begin
6          case (in)
7              4'b0000: out = 10'b1111111110;
8              4'b0001: out = 10'b1111111101;
9              4'b0010: out = 10'b1111111011;
10             4'b0011: out = 10'b1111110111;
11             4'b0100: out = 10'b1111101111;
12             4'b0101: out = 10'b1111011111;
13             4'b0110: out = 10'b1110111111;
14             4'b0111: out = 10'b1101111111;
15             4'b1000: out = 10'b1011111111;
16             4'b1001: out = 10'b0111111111;
17             default: out = 10'b1111111111; // Caso inválido
18         endcase
19     end
20 endmodule
```

Decodificador de código Excesso de 3

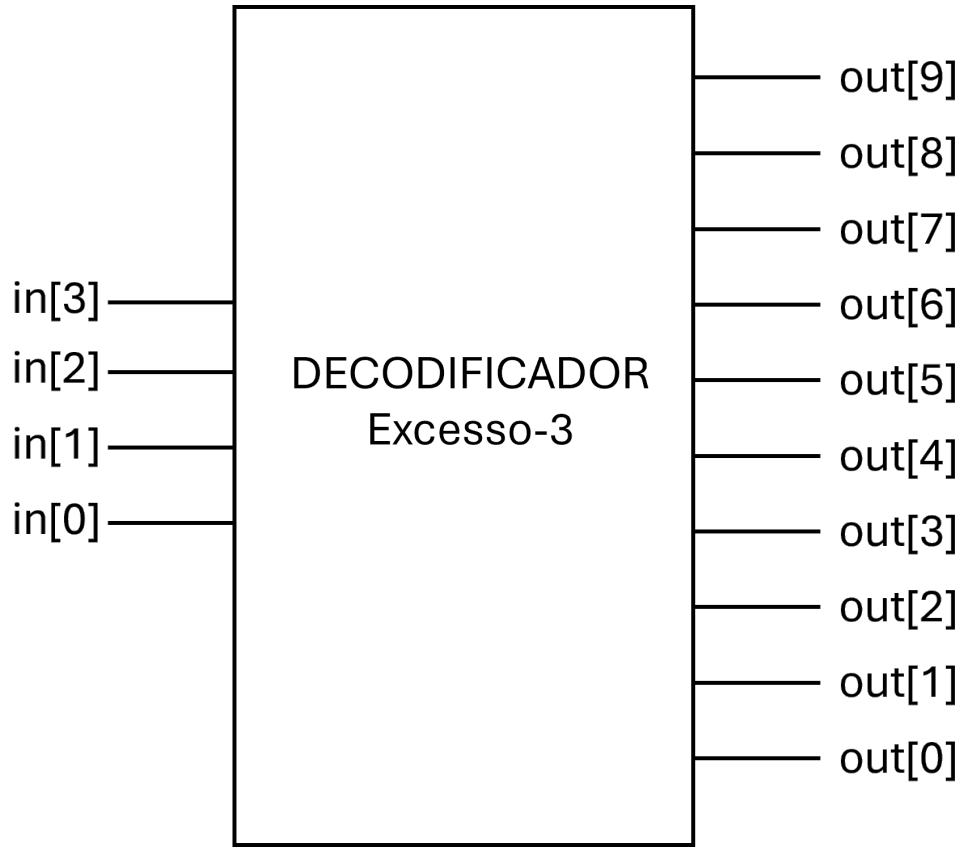

Figura 4: Exemplo de Decodificador Excesso de 3 para 10 linhas

Tabela Verdade

Decodificador Excesso de 3 para 10 linhas **Ativo Alto**

Entrada Excesso-3 (in[3:0])	Saída (out[9:0])
0011	0000000001
0100	0000000010
0101	0000000100
0110	0000001000
0111	0000010000
1000	0000100000
1001	0001000000
1010	0010000000
1011	0100000000
1100	1000000000

Expressões Lógicas

$$out[0] = \overline{in[3]} \cdot \overline{in[2]} \cdot in[1] \cdot in[0]$$

$$out[1] = \overline{in[3]} \cdot in[2] \cdot \overline{in[1]} \cdot \overline{in[0]}$$

$$out[2] = \overline{in[3]} \cdot in[2] \cdot \overline{in[1]} \cdot in[0]$$

$$out[3] = \overline{in[3]} \cdot in[2] \cdot in[1] \cdot \overline{in[0]}$$

$$out[4] = \overline{in[3]} \cdot in[2] \cdot in[1] \cdot in[0]$$

Expressões Lógicas

$$out[5] = in[3] \cdot \overline{in[2]} \cdot \overline{in[1]} \cdot \overline{in[0]}$$

$$out[6] = in[3] \cdot \overline{in[2]} \cdot \overline{in[1]} \cdot in[0]$$

$$out[7] = in[3] \cdot \overline{in[2]} \cdot in[1] \cdot \overline{in[0]}$$

$$out[8] = in[3] \cdot \overline{in[2]} \cdot in[1] \cdot in[0]$$

$$out[9] = in[3] \cdot in[2] \cdot \overline{in[1]} \cdot \overline{in[0]}$$

Descrição Fluxo de Dados

```
1  module decoder_xs3_10 (
2      input [3:0] in,     // Entradas de 4 bits (Excesso-3)
3      output [9:0] out    // Saídas de 10 bits
4  );
5
6      assign out[0] = ~in[3] & ~in[2] & in[1] & in[0];
7      assign out[1] = ~in[3] & in[2] & ~in[1] & ~in[0];
8      assign out[2] = ~in[3] & in[2] & ~in[1] & in[0];
9      assign out[3] = ~in[3] & in[2] & in[1] & ~in[0];
10     assign out[4] = ~in[3] & in[2] & in[1] & in[0];
11     assign out[5] = in[3] & ~in[2] & ~in[1] & ~in[0];
12     assign out[6] = in[3] & ~in[2] & ~in[1] & in[0];
13     assign out[7] = in[3] & ~in[2] & in[1] & ~in[0];
14     assign out[8] = in[3] & ~in[2] & in[1] & in[0];
15     assign out[9] = in[3] & in[2] & ~in[1] & ~in[0];
16
17 endmodule
```

Descrição Comportamental

```
1  module decoder_xs3_10_comp (
2      input [3:0] in,     // Entradas de 4 bits (Excesso-3)
3      output reg [9:0] out    // Saídas de 10 bits
4  );
5      always @ (*) begin
6          case (in)
7              4'b0011: out = 10'b0000000001;
8              4'b0100: out = 10'b0000000010;
9              4'b0101: out = 10'b0000000100;
10             4'b0110: out = 10'b0000001000;
11             4'b0111: out = 10'b0000010000;
12             4'b1000: out = 10'b0000100000;
13             4'b1001: out = 10'b0001000000;
14             4'b1010: out = 10'b0010000000;
15             4'b1011: out = 10'b0100000000;
16             4'b1100: out = 10'b1000000000;
17             default: out = 10'b0000000000;  // Para entradas
                   não válidas
18         endcase
19     end
20 endmodule
```

Simulação e Verificação

Testbench

```
1  module decoder_xs3_10_tb();
2      // Declaração de sinais para entradas e saídas
3      reg [3:0] in;          // Entradas de 4 bits (Excesso-3)
4      wire [9:0] out;        // Saídas de 10 bits
5
6      // Instancia o módulo a ser testado
           (decodificador_excesso_3)
7      decoder_xs3_10_comp uut (.in(in), .out(out));
8
9      initial begin
10         // Inicializa as entradas
11         in = 4'b0011;  // Inicializa com valor 0 (Excesso-3)
12         #10;           // Aguarda 10 unidades de tempo
13         in = 4'b0100;  // Testa o valor 1 (Excesso-3)
14         #10;
15         in = 4'b0101;  // Testa o valor 2 (Excesso-3)
16         #10;
17         in = 4'b0110;  // Testa o valor 3 (Excesso-3)
18         #10;
19         in = 4'b0111;  // Testa o valor 4 (Excesso-3)
```

Testbench

```
20          #10;
21          in = 4'b1000;  // Testa o valor 5 (Excesso-3)
22          #10;
23          in = 4'b1001;  // Testa o valor 6 (Excesso-3)
24          #10;
25          in = 4'b1010;  // Testa o valor 7 (Excesso-3)
26          #10;
27          in = 4'b1011;  // Testa o valor 8 (Excesso-3)
28          #10;
29          in = 4'b1100;  // Testa o valor 9 (Excesso-3)
30          #10;
31          in = 4'b1101;  // Testa um valor fora do intervalo
32          #10;
33          $stop;    // Finaliza a simulação
34      end
35      // Monitoramento das variáveis de entrada e saída
36      initial begin
37          $monitor("Tempo: %0t | Entrada: %b | Saída: %b", $time,
                in, out);
38      end
39  endmodule
```

Forma de Onda

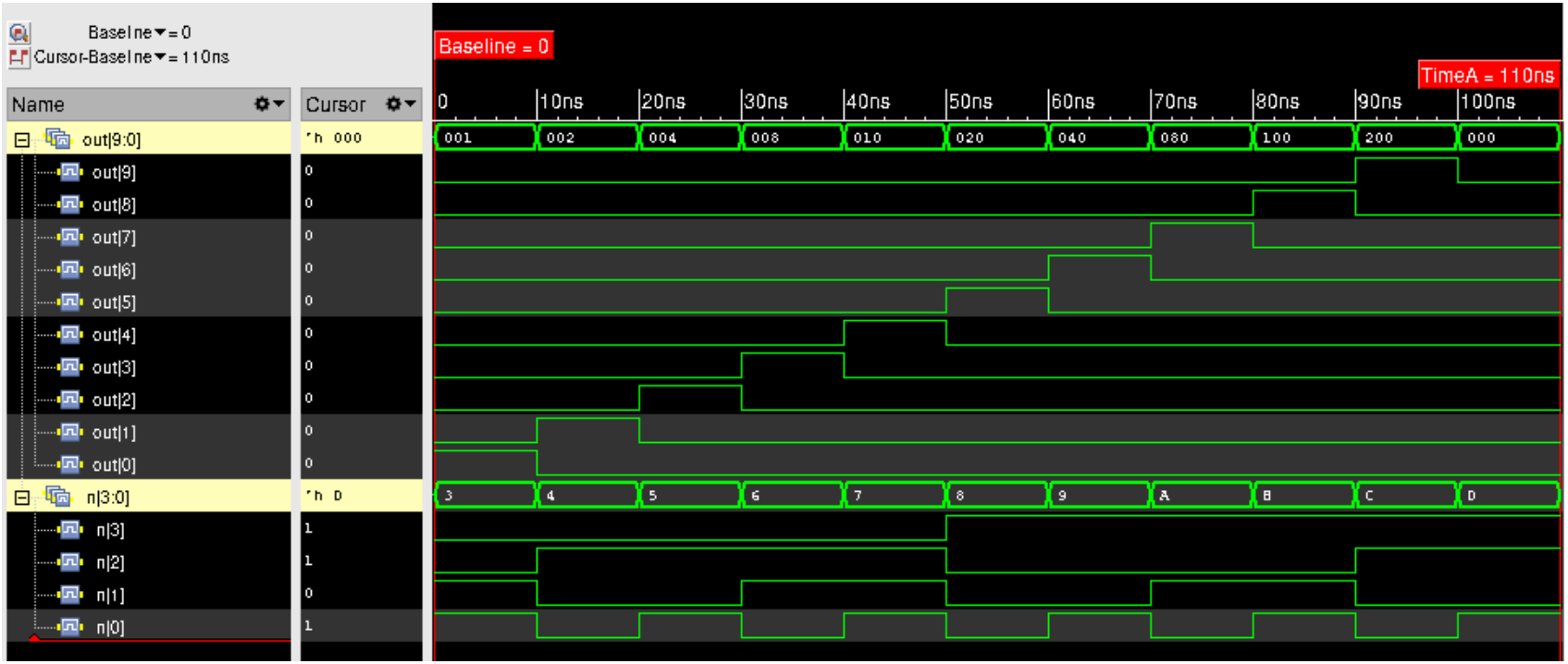

Figura 5: Resultado da Simulação

“Decodificador” para Display de 7 segmentos

- O decodificador BCD para display de 7 segmentos é na verdade um conversor de código.
  - Chamado de “decodificador” por motivos históricos

Display de 7 segmentos

- O display do tipo anodo comum tem os segmentos ativados por sinais em nível lógico baixo.

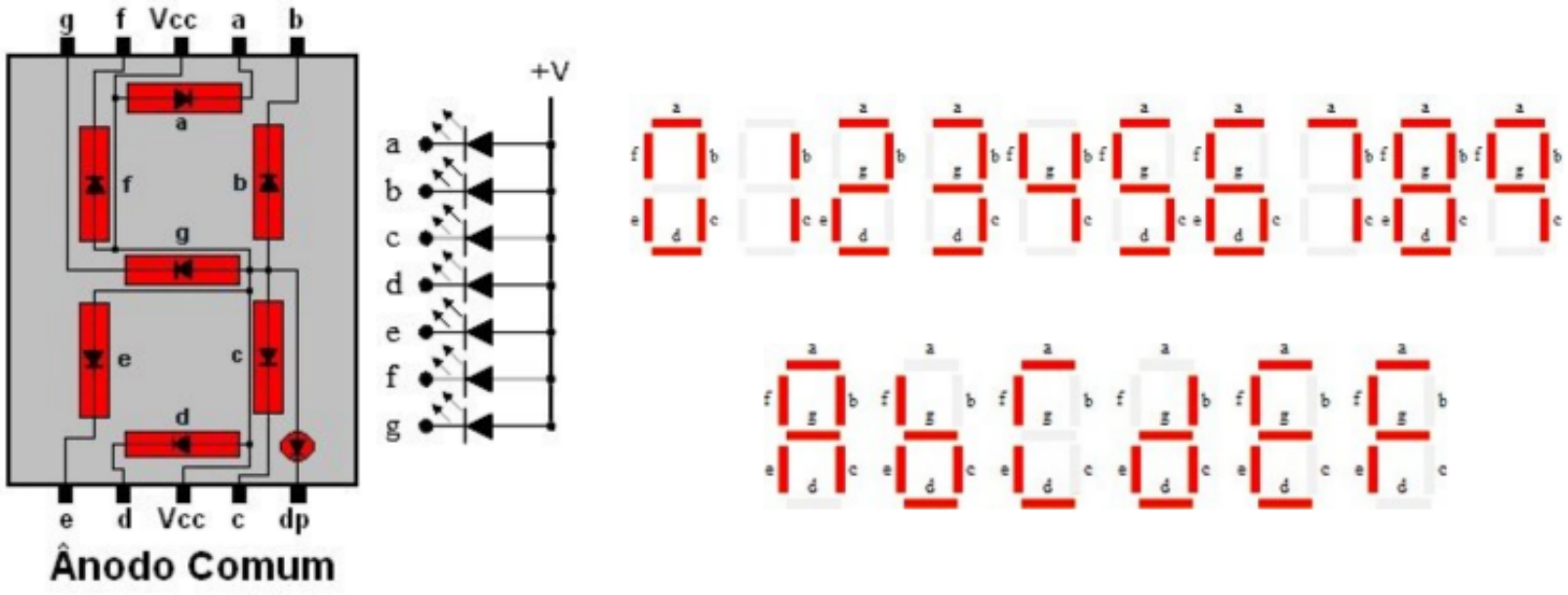

Figura 6: Display de 7 segmentos tipo anodo comum.

Tabela Verdade

Um conversor para *display* tipo anodo comum possui a seguinte tabela de operação.

Valor	Entradas $DCBA$	Saídas $abcdefg$
0	0000	0000001
1	0001	1001111
2	0010	0010010
3	0011	0000110
4	0100	1001100
5	0101	0100100
6	0110	0100000
7	0111	0001111
8	1000	0000000
9	1001	0000100

Descrição Comportamental

```
1  module BCD_to_7Segment (
2      input  wire [3:0] bin,
3      output reg a, b, c, d, e, f, g
4  );
5      always @(*) begin
6          case (bin)
7              4'h0: {a, b, c, d, e, f, g} = 7'b0000001; // 0
8              4'h1: {a, b, c, d, e, f, g} = 7'b1001111; // 1
9              4'h2: {a, b, c, d, e, f, g} = 7'b0010010; // 2
10             4'h3: {a, b, c, d, e, f, g} = 7'b0000110; // 3
11             4'h4: {a, b, c, d, e, f, g} = 7'b1001100; // 4
12             4'h5: {a, b, c, d, e, f, g} = 7'b0100100; // 5
13             4'h6: {a, b, c, d, e, f, g} = 7'b0100000; // 6
14             4'h7: {a, b, c, d, e, f, g} = 7'b0001111; // 7
15             4'h8: {a, b, c, d, e, f, g} = 7'b0000000; // 8
16             4'h9: {a, b, c, d, e, f, g} = 7'b0000100; // 9
17             default: {a, b, c, d, e, f, g} = 7'b1111111; //
                   Tudo apagado (valor inválido)
18         endcase
19     end
20 endmodule
```

Atividades Hands-on

Atividade 1

- Monte uma descrição comportamental de um decodificador binário de 4 para 16 linhas.
- Elabore um Testbench e simule a operação do circuito.

Atividade 2

- Monte uma descrição estrutural de um decodificador binário simples 4 para 16 utilizando dois módulos de um decodificador 3 para 8 e mais a lógica adicional que for necessária (essa pode ser descrita em fluxo de dados no próprio módulo topo).
- Elabore um Testbench e simule a operação do circuito.

Atividade 3

- Elabore uma descrição parametrizável de um decodificador binário simples, com N bits de entrada e M bits de saída.
- Simule para N = 4;