Aulas 4 e 5
Conversores de Código

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

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

Tópicos

- Introdução
- Descrição Comportamental
- Implementação em Verilog
- Simulação e Verificação
- Atividades Hands-on

Introdução

Objetivos

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

O que são Conversores de Códigos?

- Circuitos que transformam um código de entrada em outro código de saída.
- Amplamente utilizados em sistemas digitais, comunicação e processamento de sinais.

Descrição em Verilog

- **Fluxo de Dados (Dataflow)**
- Baseado em expressões booleanas.
- Define circuitos usando atribuições contínuas (`assign`).
- **Estrutural (Structural)**
- Usa interconexão de módulos e portas lógicas.
- Representa o hardware físico de forma explícita.

Descrição em Verilog

- **Comportamental (Behavioral)**
- Utiliza blocos `always`, `case`, `if` para descrever o funcionamento do circuito.
- Mais próximo da lógica algorítmica do que da estrutura física.

Descrição Comportamental

Vantagens

- Processo Manual é demorado e sujeito a erros
- Descrição comportamental é mais abstrata
  - Tabela circuito combinacional –> Descrição em Verilog
- Especificação –> Síntese Lógica (Realizado pelo Software!)

Pontos de Atenção

- Descrição comportamental pode gerar um circuito diferente do pretendido!
- Descrições incompletas podem ser interpretadas erroneamente pela ferramenta de síntese
- Possível inferência de elementos de memória indesejados

Implementação em Verilog

Exemplo Conversor de Código

- Tabela de Conversão BCD 5311 $\rightarrow$ BCD 8421

Decimal	**BCD 5311**	**BCD 8421**
Variáveis	HGFE	DCBA
0	0000	0000
1	0001	0001
2	0011	0010
3	0100	0011
4	0101	0100
5	0111	0101
6	1001	0110
7	1011	0111
8	1100	1000
9	1101	1001

Primeira tentativa

- Descrição Comportamental

```
1  module conversor(bcd_5311, bcd_8421);
2  input [3:0] bcd_5311;
3  output reg [3:0] bcd_8421;
4
5  always@(*) begin
6    case(bcd_5311)
7      4'b0000: bcd_8421 = 4'b0000; // 0
8      4'b0001: bcd_8421 = 4'b0001; // 1
9      4'b0011: bcd_8421 = 4'b0010; // 2
10     4'b0100: bcd_8421 = 4'b0011; // 3
11     4'b0101: bcd_8421 = 4'b0100; // 4
12     4'b0111: bcd_8421 = 4'b0101; // 5
13     4'b1001: bcd_8421 = 4'b0110; // 6
14     4'b1011: bcd_8421 = 4'b0111; // 7
15     4'b1100: bcd_8421 = 4'b1000; // 8
16     4'b1101: bcd_8421 = 4'b1001; // 9
17   endcase
18 end
19 endmodule
```

Circuito inferido

- Qual o problema?
- Bloco always com atribuição incompleta –> Gera elemento de memória adicional

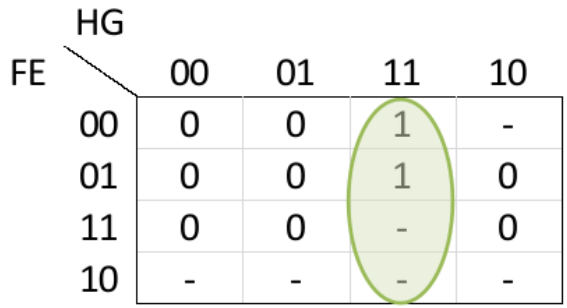
Seunda tentativa

```
1  module conversor(bcd_5311, bcd_8421);
2  input [3:0] bcd_5311;
3  output reg [3:0] bcd_8421;
4
5  always@(*) begin
6    case(bcd_5311)
7      4'b0000: bcd_8421 = 4'b0000; // 0
8      4'b0001: bcd_8421 = 4'b0001; // 1
9      4'b0011: bcd_8421 = 4'b0010; // 2
10     4'b0100: bcd_8421 = 4'b0011; // 3
11     4'b0101: bcd_8421 = 4'b0100; // 4
12     4'b0111: bcd_8421 = 4'b0101; // 5
13     4'b1001: bcd_8421 = 4'b0110; // 6
14     4'b1011: bcd_8421 = 4'b0111; // 7
15     4'b1100: bcd_8421 = 4'b1000; // 8
16     4'b1101: bcd_8421 = 4'b1001; // 9
17     default: bcd_8421 = 4'b0000;
18   endcase
19 end
```

Circuito inferido

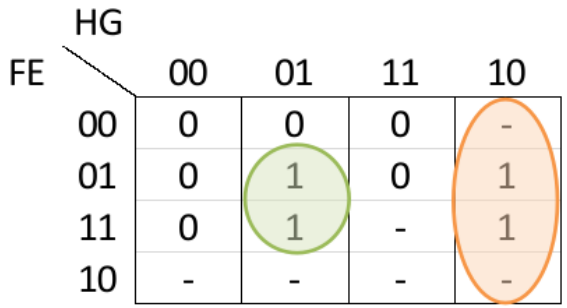
- O **default** garante a inferência de um circuito combinacional
- É a forma mais otimizada de implementar o circuito?
- Circuito inferido será maior do que o encontrado na simplificação com mapas de Karnaugh. Por quê?

Mapas de Karnaugh



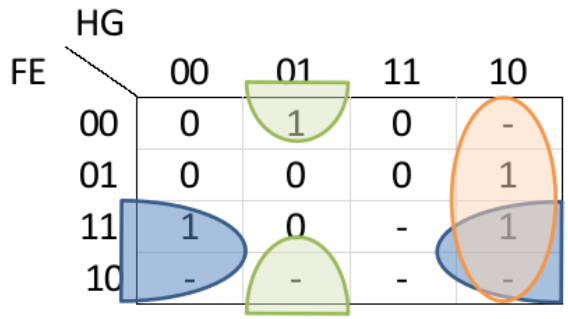
FE \ HG	00	01	11	10
00	0	0	1	-
01	0	0	1	0
11	0	0	-	0
10	-	-	-	-

$$D = HG$$



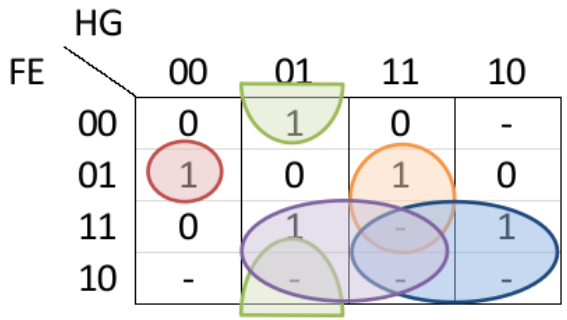
FE \ HG	00	01	11	10
00	0	0	0	-
01	0	1	0	1
11	0	1	-	1
10	-	-	-	-

$$C = \overline{H}GE + H\overline{G}$$



FE \ HG	00	01	11	10
00	0	1	0	-
01	0	0	0	1
11	1	0	-	1
10	-	-	-	-

$$B = \overline{H}G\overline{E} + \overline{G}F + H\overline{G}$$



FE \ HG	00	01	11	10
00	0	1	0	-
01	1	0	1	0
11	0	1	-	1
10	-	-	-	-

$$A = \overline{H}\overline{G}\overline{F}E + \overline{H}G\overline{E} + GF + HGE + HF$$

Figura 1: Mapas de Karnaugh do Conversor de Código

Mapas de Karnaugh

- Mapas contemplam condições do tipo **“Don’t Care”**
- As expressões lógicas simplificadas resultam em um circuito menor (número de portas lógicas)

Terceira tentativa

```
1  module conversor(bcd_5311, bcd_8421);
2  input [3:0] bcd_5311;
3  output reg [3:0] bcd_8421;
4
5  always@(*) begin
6    case(bcd_5311)
7      4'b0000: bcd_8421 = 4'b0000; // 0
8      4'b0001: bcd_8421 = 4'b0001; // 1
9      4'b0011: bcd_8421 = 4'b0010; // 2
10     4'b0100: bcd_8421 = 4'b0011; // 3
11     4'b0101: bcd_8421 = 4'b0100; // 4
12     4'b0111: bcd_8421 = 4'b0101; // 5
13     4'b1001: bcd_8421 = 4'b0110; // 6
14     4'b1011: bcd_8421 = 4'b0111; // 7
15     4'b1100: bcd_8421 = 4'b1000; // 8
16     4'b1101: bcd_8421 = 4'b1001; // 9
17     default: bcd_8421 = 4'bxxxx;
18   endcase
19 end
```

Circuito inferido

- O uso da condição 'não importa' com `x` (desconhecido) informa a ferramenta de síntese
- O circuito inferido será equivalente à implementação das expressões lógicas obtidas após simplificação manual

Simulação e Verificação

Testbench

```
1  module conversor_tb();
2  reg [3:0] in;
3  wire [3:0] out;
4
5  conversor DUT(in, out);
6
7  initial begin
8    in = 4'b0000; #10;
9    in = 4'b0001; #10;
10   in = 4'b0011; #10;
11   in = 4'b0100; #10;
12   in = 4'b0101; #10;
13   in = 4'b0111; #10;
14   in = 4'b1001; #10;
15   in = 4'b1011; #10;
16   in = 4'b1100; #10;
17   in = 4'b1101; #10;
18   $stop;
19 end
20 endmodule
```

Forma de Onda

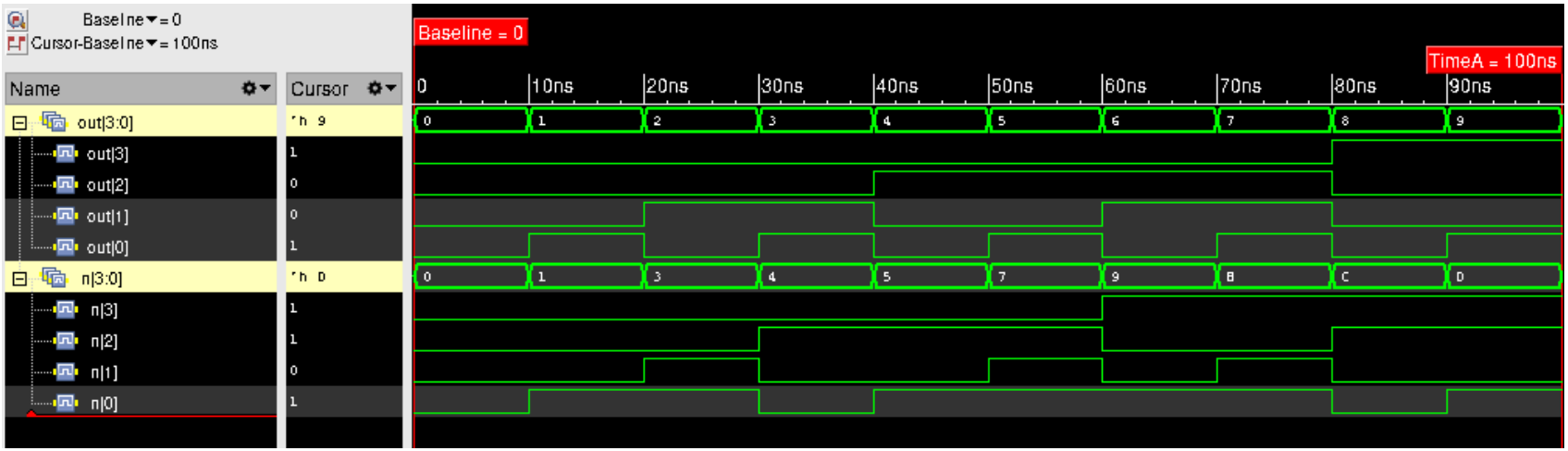


Figura 2: Resultado da Simulação

Atividades Hands-on

Atividade 1

- Monte uma descrição comportamental de um conversor de código BCD8421 para Excesso de 3.
- Elabore um Testbench e simule a operação do circuito.

Atividade 2

- Monte uma descrição comportamental de um conversor de código binário de 4 bits para código gray.
- Elabore um Testbench e simule a operação do circuito.

Atividade 3

- O conversor de código 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.

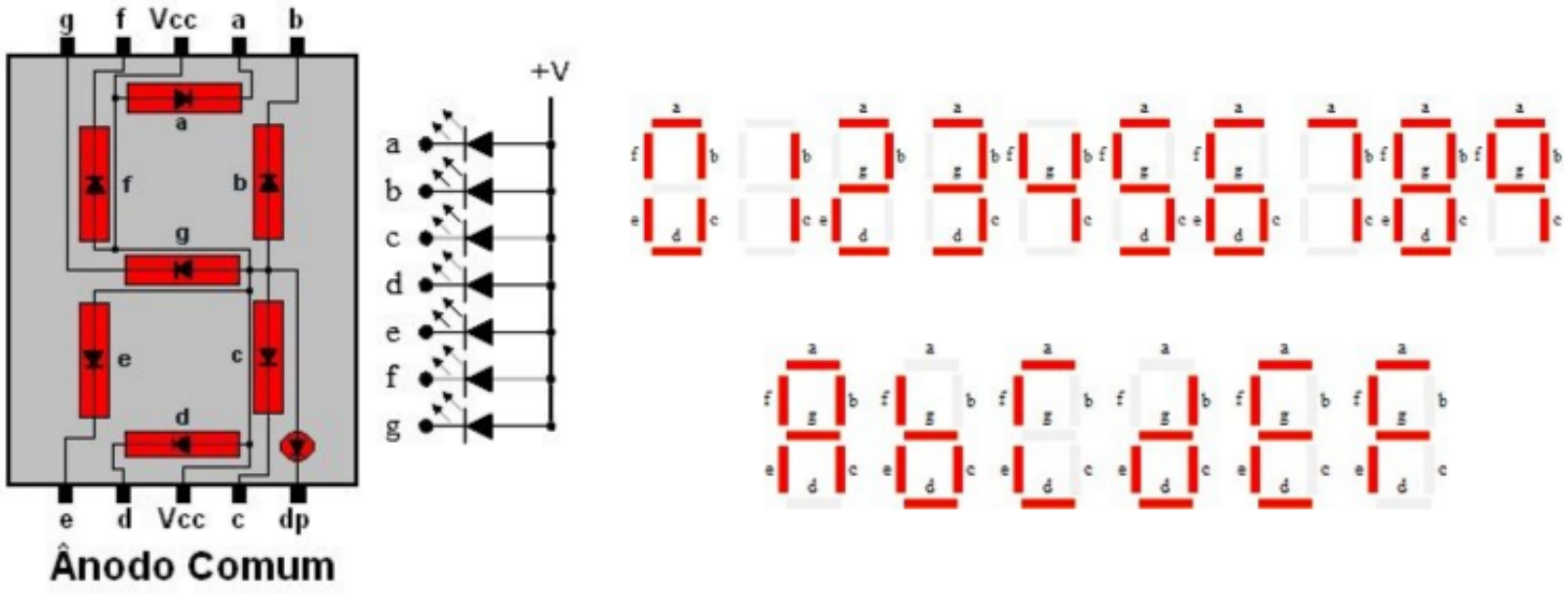


Figura 3: Display de 7 segmentos tipo anodo comum.

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

Atividade 3

- Adicione os simbolos A, b, C, d, E e F à tabela (correspondentes aos valores decimais de 10 a 15).
- Monte uma descrição comportamental de um conversor de código binário de 4 bits para um display de 7 segmentos do tipo anodo comum.
- Elabore um testbench e simule a operação do circuito