UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

Emanuel de Oliveira
Gabriel Bazon
Giovanna Velasco
Guilherme Schmidt
João Pedro Gomes
João Victor Alves
Luana Hartmann

Projeto: Primeira Parte

São Carlos
2023

Emanuel de Oliveira
Gabriel Bazon
Giovanna Velasco
Guilherme Schmidt
João Pedro Gomes
João Victor Alves
Luana Hartmann

Projeto: Primeira Parte

Trabalho apresentado à disciplina de Sistemas Digitais da Escola de Engenharia de São Carlos, Universidade de São Paulo, como parte dos requisitos para a aprovação na disciplina.

Área de concentração: Engenharia de Computação

Orientador: Prof. Dr. Maximiliam Luppe

São Carlos
2023

Sumário

1 Introdução

O presente documento descreve o desenvolvimento e a implementação de um Decodificador BCD para 7 Segmentos, com parâmetro extra de entrada "commom_cathode". Em suma, o circuito decodificador em questão dispõe-se a receber como entrada um número decimal codificado em binário (BCD) e produzir em sua saída uma representação de tal número por meio de um display de 7 segmentos, componente eletrônico que faz o uso de LEDs para representar algarismos e caracteres.

Para a implementação de um sistema como esse, deve-se especificar qual método de representação será utilizado: ânodo comum ou cátodo comum. Resumidamente, um display de 7 segmentos com ânodo comum tem seus LEDs acionados por nível lógico 0, já com cátodo comum, estes são ativados por nível lógico 1. Nesse sentido, para o desenvolvimento do circuito que será realizado, o Decodificador BCD para 7 Segmentos receberá a entrada adicional denominada “commom_cathode”, a qual será responsável por determinar qual de ambas representações será utilizada.

Partindo disso, inicialmente serão construídas as Tabelas Verdade referentes ao circuito. Em seguida, a partir delas, serão apresentados os Mapas de Karnaugh e as Expressões Booleanas de cada saída do circuito. Por fim, será implementado o circuito lógico, utilizando-se do esquemático do circuito obtido através da linguagem Verilog.

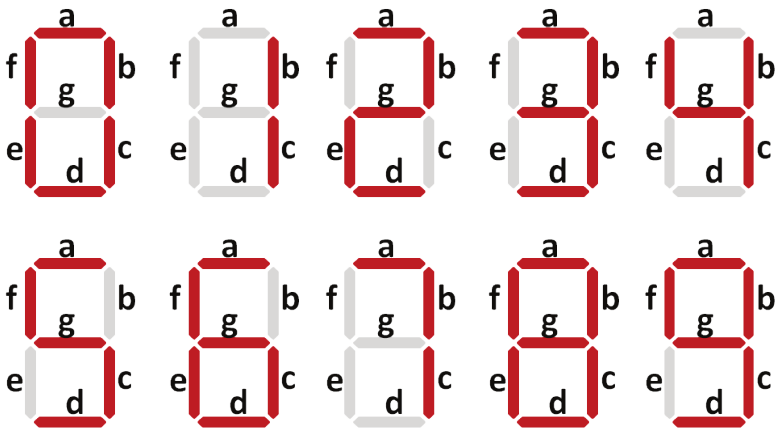
Finalmente, destaca-se que a representação em Verilog será feita de 4 formas distintas, a saber: Primitivas ou Rede de Ligações; Declarações Concorrentes com Operadores Lógicos; Declarações Concorrentes com Operador Ternário; Declaração Procedural ou Comportamental.

2 Tabela Verdade

A princípio, para o desenvolvimento das Tabelas Verdade do Decodificador em questão, destaca-se que o BCD recebido como entrada é dividido em 4 entradas: B_3, B_2, B_1 e B_0. Cada qual representa um algarismo de binário do BCD final, obtendo-se a seguinte representação: $B_3B_2B_1B_0$. Para o Display de 7 Segmentos, a ordem dos LEDs segue a classificação da Figura 1. Portanto, é necessário encontrar uma relação entre $B_3B_2B_1B_0$ e as letras a, b, c, d, e, f, g do display.

Para melhor visualização, o parâmetro adicional “commom_cathode” não será inserido na tabela verdade. Na verdade, construir-se-á duas tabelas verdades, uma em que considera-se “commom_cathode” como 0, indicando representação com ânodo comum; e outra em que “commom_cathode” vale 1, explicitando cátodo comum.

Figura 1 – Display de 7 segmentos com indicação dos diodos ativados para cada entrada BCD



Nesse sentido, primeiramente, constrói-se a tabela referente ao Decodificador com Ânodo Comum (Tabela 1). Nela basta notar que, para cada número BCD, os valores 0 indicam os LEDs acesos no Display exemplificado na Figura 1.

Tabela 1 – Tabela Verdade do Decodificador BCD para 7 Segmentos com Ânodo Comum

B_3	B_2	B_1	B_0	a	b	c	d	e	f	g
0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	1	0	0	1	1	1	1
0	0	1	0	0	0	1	0	0	1	0
0	0	1	1	0	0	0	0	1	1	0
0	1	0	0	1	0	0	1	1	0	0
0	1	0	1	0	1	0	0	1	0	0
0	1	1	0	0	1	0	0	0	0	0
0	1	1	1	0	0	0	1	1	1	1
1	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	1	0	0

Dando prosseguimento, agora, é possível construir a tabela referente ao Decodificador com Cátodo Comum (Tabela 2). Nesse caso, os LEDs são acesos com o valor lógico 1.

Vale ressaltar que o parâmetro “commom_cathode” atua como inversor do sinal das letras que compõem o display de 7 segmentos. Isto é, para um mesmo BCD os sinais das Tabelas 1 e 2 são invertidos. Portanto, serão gerados dois circuitos distintos com saídas complementares a depender do parâmentro.

Tabela 2 – Tabela Verdade de Decodificador BCD para 7 Segmentos com Cátodo Comum

B_3	B_2	B_1	B_0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1

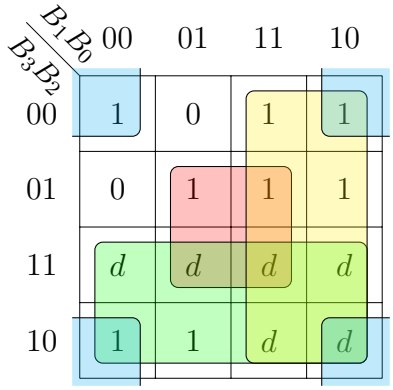
3 Mapas de Karnaugh e Expressões Booleanas

Com as Tabelas Verdade construídas, é possível encontrar as Expressões Booleanas das saídas para cada caso por meio de Mapas de Karnaugh. Como o parâmetro "commom_cathode" implica em dois circuitos distintos, optamos por encontrar primeiramente as Expressões Booelanas considerando Cátodo Comum ("commom_cathode" = 1) e, posteriormente, as Expressões referentes ao Ânodo Comum ("commom_cathode" = 0). Para ambos as expressões foram determinadas no formato SOP.

Na construção dos Mapas de Karnaugh, vale destacar que as entradas em binário referentes aos números de 10 a 15 são postos como "*don't care*", representados pela letra *d*, devido ao fato de não corresponderem a entradas BCD. Assim, iniciando-se com o caso de Cátodo Comum, para cada saída da Tabela 2 foi construído um Mapa, como se segue.

3.1 Cátodo Comum

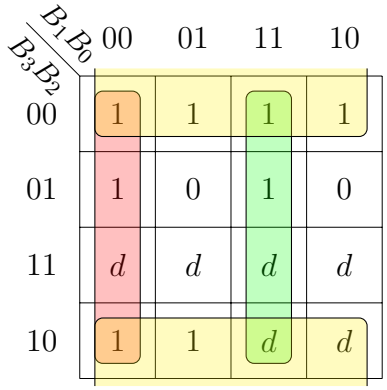
Mapa de Karnaugh para o diodo *a*



Nesse caso, vê-se que a Expressão Booleana mínima para o diodo a é dada por

$$a = f(B_3, B_2, B_1, B_0) = B_3 + B_1 + \overline{B_2}\,\overline{B_0} + B_2\,B_0 \tag{1}$$

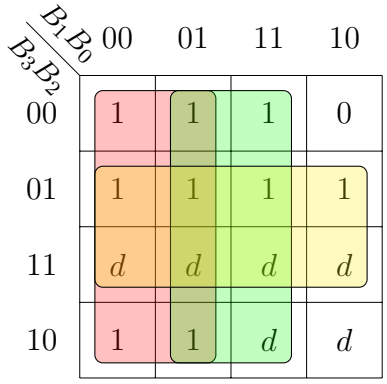
Mapa de Karnaugh para o diodo b



Para este caso, vê-se que a Expressão Booleana mínima para o diodo b é dada por

$$b = f(B_3, B_2, B_1, B_0) = \overline{B_2} + \overline{B_1}\,\overline{B_0} + B_1\,B_0 \tag{2}$$

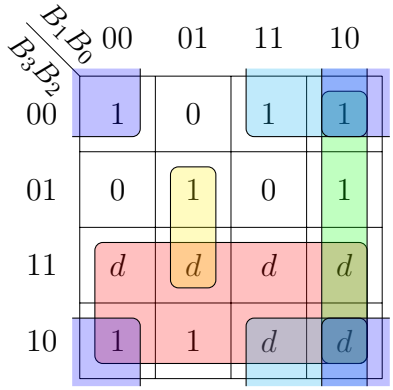
Mapa de Karnaugh para o diodo c



Agora, vê-se que a Expressão Booleana mínima para o diodo c é dada por

$$c = f(B_3, B_2, B_1, B_0) = B_2 + \overline{B_1} + B_0 \tag{3}$$

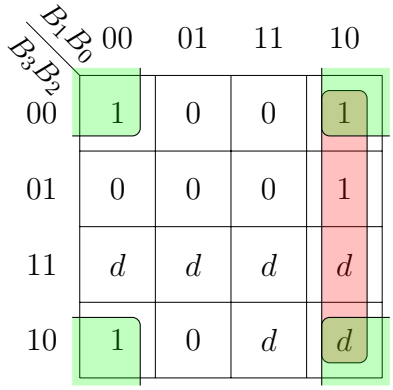
Mapa de Karnaugh para o diodo d



Dando prosseguimento, vê-se que a Expressão Booleana mínima para o diodo d é dada por

$$d = f(B_3, B_2, B_1, B_0) = B_3 + \overline{B_2}\,\overline{B_0} + \overline{B_2}\,B_1 + B_1\,\overline{B_0} + B_2\,\overline{B_1}\,B_0 \tag{4}$$

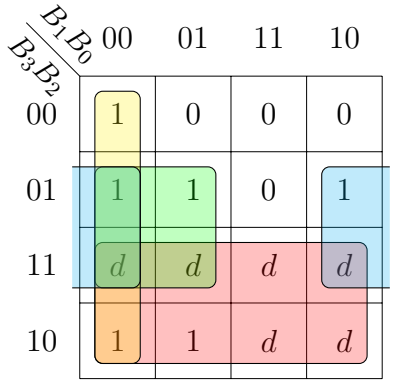
Mapa de Karnaugh para o diodo e



Seguindo, vê-se que a Expressão Booleana mínima para o diodo e é dada por

$$e = f(B_3, B_2, B_1, B_0) = \overline{B_2}\,\overline{B_0} + B_1\,\overline{B_0} \tag{5}$$

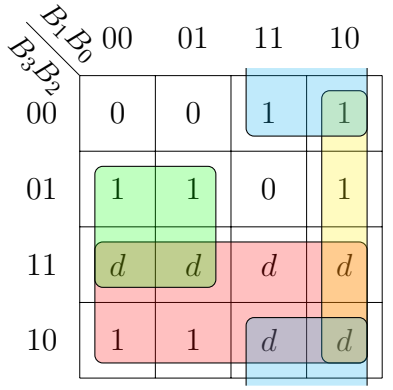
Mapa de Karnaugh para o diodo f



Ainda, vê-se que a Expressão Booleana mínima para o diodo f é dada por

$$f = f(B_3, B_2, B_1, B_0) = B_3 + \overline{B_1}\,\overline{B_0} + B_2\,\overline{B_1} + B_2\overline{B_0} \tag{6}$$

Mapa de Karnaugh para o diodo g



Por fim, vê-se que a Expressão Booleana mínima para o diodo g é dada por

$$g = f(B_3, B_2, B_1, B_0) = B_3 + B_1\overline{B_0} + \overline{B_2}B_1 + B_2\overline{B_1} \tag{7}$$

Assim, determinadas as Equações de 1 a 7 para o Cátodo Comum, parte-se agora para a confecção dos Mapas de Karnaguh e determinação das Expressões Booleanas para o caso de Ânodo Comum, conforme a Tabela 1, como se segue.

3.2 Ânodo Comum

Mapa de Karnaugh para o diodo a

B_3B_2 \ B_1B_0	00	01	11	10
00	0	1	0	0
01	1	0	0	0
11	d	d	d	d
10	0	0	d	d

Nesse caso, vê-se que a Expressão Booleana mínima para o diodo a é dada por

$$a = f(B_3, B_2, B_1, B_0) = B_2\,\overline{B_1}\,\overline{B_0} + \overline{B_3}\,\overline{B_2}\,\overline{B_1}\,B_0 \tag{8}$$

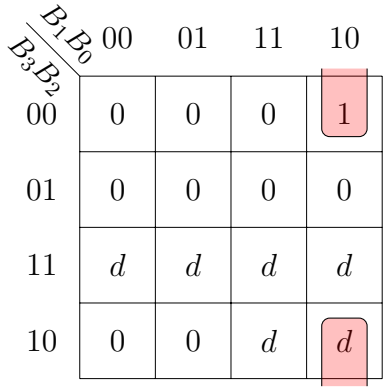
Mapa de Karnaugh para o diodo b

B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	0	0
01	0	1	0	1
11	d	d	d	d
10	0	0	d	d

Para este caso, vê-se que a Expressão Booleana mínima para o diodo b é dada por

$$b = f(B_3, B_2, B_1, B_0) = B_2\,\overline{B_1}\,B_0 + B_2\,B_1\,\overline{B_0} \tag{9}$$

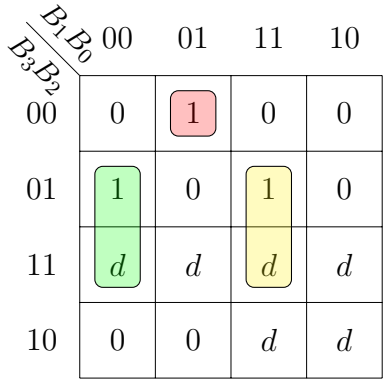
Mapa de Karnaugh para o diodo c



Agora, vê-se que a Expressão Booleana mínima para o diodo c é dada por

$$c = f(B_3, B_2, B_1, B_0) = \overline{B_2}\, B_1\, \overline{B_0} \tag{10}$$

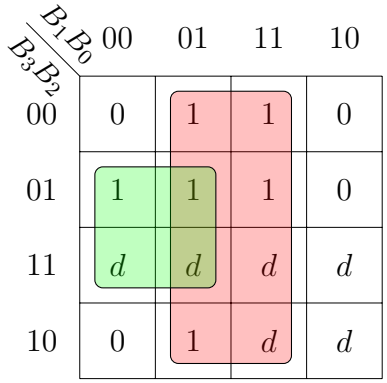
Mapa de Karnaugh para o diodo d



Dando prosseguimento, vê-se que a Expressão Booleana mínima para o diodo d é dada por

$$d = f(B_3, B_2, B_1, B_0) = \overline{B_3}\,\overline{B_2}\,\overline{B_1}\, B_0 + B_2\, B_1\, B_0 + B_2\, \overline{B_1}\,\overline{B_0} \tag{11}$$

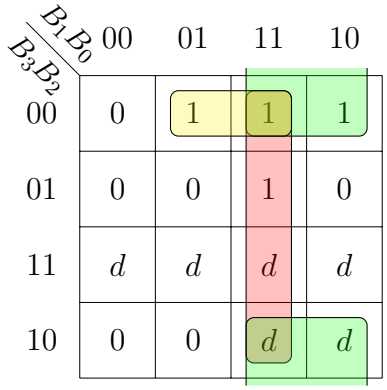
Mapa de Karnaugh para o diodo e



Seguindo, vê-se que a Expressão Booleana mínima para o diodo e é dada por

$$e = f(B_3, B_2, B_1, B_0) = B_2\,\overline{B_1} + B_0 \tag{12}$$

Mapa de Karnaugh para o diodo f



Ainda, vê-se que a Expressão Booleana mínima para o diodo f é dada por

$$f = f(B_3, B_2, B_1, B_0) = \overline{B_2}\,B_1 + B_1\,B_0 + \overline{B_3}\,\overline{B_2}\,B_0 \tag{13}$$

Mapa de Karnaugh para o diodo g

B_3B_2 \ B_1B_0	00	01	11	10
00	1	1	0	0
01	0	0	1	0
11	d	d	d	d
10	0	0	d	d

Por fim, vê-se que a Expressão Booleana mínima para o diodo g é dada por

$$g = f(B_3, B_2, B_1, B_0) = B_2\, B_1\, B_0 + \overline{B_3}\,\overline{B_2}\,\overline{B_1} \tag{14}$$

Cabe destacar que, como mencionado anteriormente, as expressões referentes ao Ânodo Comum são os complementos das expressões para Cátodo Comum. De fato, tomemos o diodo c. Note que, em Cátodo Comum, sua expressão é

$$c_C = f(B_3, B_2, B_1, B_0) = B_2 + \overline{B_1} + B_0$$

Veja que, em Ânodo Comum, a expressão é

$$c_A = f(B_3, B_2, B_1, B_0) = \overline{B_2}\, B_1\, \overline{B_0}$$

Isto é,

$$c_A = \overline{c}_C.$$

Entretanto, optou-se pela construção dos Mapas de Karnaugh por sua simplicidade e para evitar complicações com a Álgebra de Boole.

Nesse sentido, obtidas as Expressões Booleanas correspondentes ao circuito tanto para o caso de Cátodo Comum quanto para o de Ânodo Comum, parte-se para sua implementação em Verilog.

4 Implementação do Circuito

Para a implementação do circuito, será utilizada a linguagem de descrição de hardware Verilog. Nesse caso, serão feitas 4 implementações distintas, cada qual utilizando-se de um dos métodos que a linguagem fornece. Destaca-se que os códigos podem ser também encontrados no repositório do projeto.

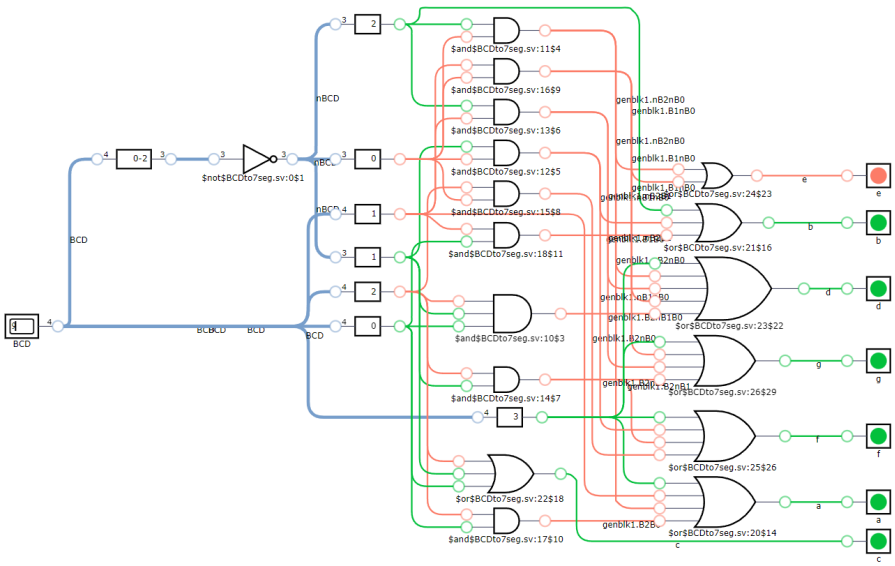
4.1 Método de Primitivas ou Rede de Ligações

Nesta seção, será implementado o circuito em Verilog a partir das expressões booleanas previamente determinadas, descrevendo-o por meio de primitivas lógicas na linguagem de descrição de hardware. Destaca-se o emprego do parâmetro `common_cathode` para definir a geração do circuito para displays de cátodo comum, quando 1, ou de ânodo comum, quando 0. Abaixo segue o código criado e circuito esquemático correspondente (Figura 2).

```
module BCDto7seg #(parameter common_cathode=1)(output a, b, c, d, e, f,
↪  g, input [3:0] BCD);
  wire [3:0] nBCD;
  not (nBCD, BCD);

  generate
    if(common_cathode==1)
      begin
        wire nB2nB0, B2B0, nB1nB0, B1B0, nB2B1, B1nB0, B2nB1, B2nB0,
↪  B2nB1B0;

        and (B2nB1B0, BCD[2], nBCD[1], BCD[0]);
        and (nB2nB0, nBCD[2], nBCD[0]);
        and (nB1nB0, nBCD[1], nBCD[0]);
        and (nB2B1, nBCD[2], BCD[1]);
        and (B2nB1, BCD[2], nBCD[1]);
        and (B2nB0, BCD[2], nBCD[0]);
        and (B1nB0, BCD[1], nBCD[0]);
        and (B2B0, BCD[2], BCD[0]);
        and (B1B0, BCD[1], BCD[0]);

        or (a, BCD[3], BCD[1], nB2nB0, B2B0);
        or (b, nBCD[2], nB1nB0, B1B0);
        or (c, BCD[2], nBCD[1], BCD[0]);
        or (d, BCD[3], nB2nB0 , nB2B1, B1nB0, B2nB1B0);
        or (e, nB2nB0, B1nB0);
        or (f, BCD[3], nB1nB0, B2nB1, B2nB0);
        or (g, BCD[3], B1nB0, nB2B1, B2nB1);
      end
```

```
    else
      begin
        wire B2nB1nB0, nB3nB2nB1B0, B2nB1B0, B2B1nB0, nB3nB2B0, nB3nB2nB1,
↪   B2B1B0, B2nB1, nB2B1, B1B0;

        and (B2nB1nB0, BCD[2], nBCD[1], nBCD[0]);
        and (nB3nB2nB1B0, nBCD[3], nBCD[2], nBCD[1], BCD[0]);
        and (B2nB1B0, BCD[2], nBCD[1], BCD[0]);
        and (B2B1nB0, BCD[2], BCD[1], nBCD[0]);
        and (c, nBCD[2], BCD[1], nBCD[0]);
        and (nB3nB2B0, nBCD[3], nBCD[2], BCD[0]);
        and (nB3nB2nB1, nBCD[3], nBCD[2], nBCD[1]);
        and (B2B1B0, BCD[2], BCD[1], BCD[0]);
        and (B2nB1, BCD[2], nBCD[1]);
        and (nB2B1, nBCD[2], BCD[1]);
        and (B1B0, BCD[1], BCD[0]);

        or(a, B2nB1nB0, nB3nB2nB1B0);
        or(b, B2nB1B0, B2B1nB0);
        or(d, nB3nB2nB1B0, B2B1B0, B2nB1nB0);
        or(e, B2nB1, BCD[0]);
        or(f, nB2B1, B1B0, nB3nB2B0);
        or(g, B2B1B0, nB3nB2nB1);
      end
  endgenerate
endmodule
```

Figura 2 – Circuito esquemático do método de Primitivas ou Rede de Ligações



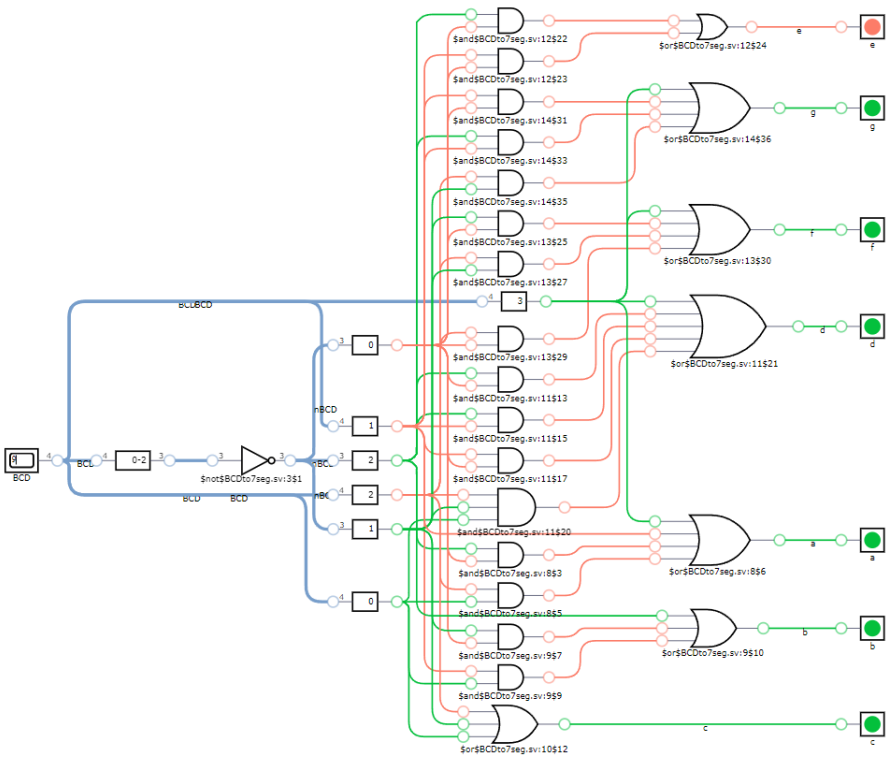
4.2 Método de Declarações Concorrentes com Operadores Lógicos

Nesta seção, o circuito lógico é novamente implementado, mas empregando-se desta vez o método de declarações concorrentes com operadores lógicos em Verilog. Outra vez, é empregado o parâmetro `common_cathode`, atuando semelhantemente à implementação anterior. A seguir, seguem a implementação e circuito esquemático (Figura 3).

```
module BCDto7seg #(parameter common_cathode=1)(output a, b, c, d, e, f,
↪  g, input [3:0] BCD);
  wire [3:0] nBCD;
  assign nBCD = ~BCD;

  generate
    if(common_cathode == 1)
      begin
        assign a = BCD[3] | BCD[1] | nBCD[2]&nBCD[0] | BCD[2]&BCD[0];
        assign b = nBCD[2] | nBCD[1]&nBCD[0] | BCD[1]&BCD[0];
        assign c = BCD[2] | nBCD[1] | BCD[0];
        assign d = BCD[3] | nBCD[2]&nBCD[0] | nBCD[2]&BCD[1] |
↪  BCD[1]&nBCD[0] | BCD[2]&nBCD[1]&BCD[0];
        assign e = nBCD[2]&nBCD[0] | BCD[1]&nBCD[0];
        assign f = BCD[3] | nBCD[1]&nBCD[0] | BCD[2]&nBCD[1] |
↪  BCD[2]&nBCD[0];
        assign g = BCD[3] | BCD[1]&nBCD[0] | nBCD[2]&BCD[1] |
↪  BCD[2]&nBCD[1];
      end
    else
      begin
        assign a = BCD[2]&nBCD[1]&nBCD[0] |
↪  nBCD[3]&nBCD[2]&nBCD[1]&BCD[0];
        assign b = BCD[2]&nBCD[1]&BCD[0] | BCD[2]&BCD[1]&nBCD[0];
        assign c = nBCD[2]&BCD[1]&nBCD[0];
        assign d = nBCD[3]&nBCD[2]&nBCD[1]&BCD[0] | BCD[2]&BCD[1]&BCD[0]
↪  | BCD[2]&nBCD[1]&nBCD[0];
        assign e = BCD[2]&nBCD[1] | BCD[0];
        assign f = nBCD[2]&BCD[1] | BCD[1]&BCD[0] |
↪  nBCD[3]&nBCD[2]&BCD[0];
        assign g = BCD[2]&BCD[1]&BCD[0] | nBCD[3]&nBCD[2]&nBCD[1];
```

```
      end
  endgenerate
endmodule
```

Figura 3 – Circuito esquemático do método de Declarações Concorrentes com Operadores Lógicos



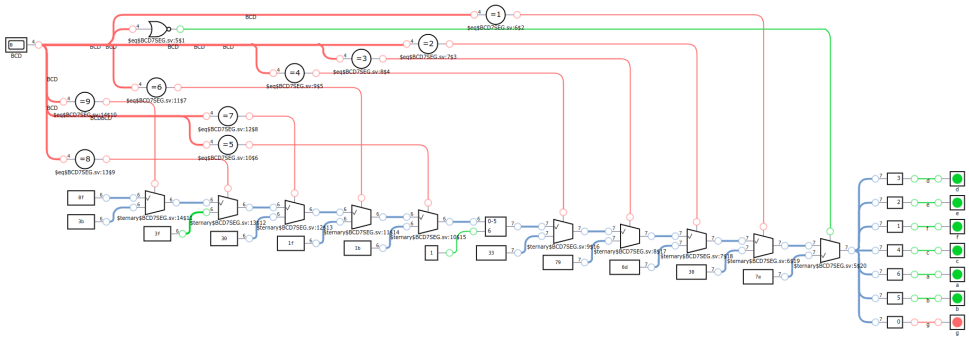
4.3 Método de Declarações Concorrentes com Operador Ternários

Com esta terceira implementação do circuito, utilizam-se declarações concorrentes com operadores ternários para implementar o decodificador, definindo as entradas e saídas a partir das Tabelas Verdades previamente construídas, com a saída padrão para valores não BCD definida como `E`, correspondente a "Erro". Mais uma vez é empregado o parâmetro `common_cathode` para definir a versão a ser gerada. Segue-se o código e esquemático (Figura 4) correspondente em Verilog.

```
module BCDto7seg #(parameter common_cathode=1)(output a, b, c, d, e, f,
↪  g, input [3:0] BCD);
  generate
    if (common_cathode == 1)
      assign {a, b, c, d, e, f, g} =
        (BCD == 4'b0000) ? 7'b1111110 :
        (BCD == 4'b0001) ? 7'b0110000 :
        (BCD == 4'b0010) ? 7'b1101101 :
        (BCD == 4'b0011) ? 7'b1111001 :
        (BCD == 4'b0100) ? 7'b0110011 :
        (BCD == 4'b0101) ? 7'b1011011 :
        (BCD == 4'b0110) ? 7'b1011111 :
        (BCD == 4'b0111) ? 7'b1110000 :
        (BCD == 4'b1000) ? 7'b1111111 :
        (BCD == 4'b1001) ? 7'b1111011 :
        7'b1001111;
    else
      assign {a, b, c, d, e, f, g} =
        (BCD == 4'b0000) ? 7'b0000001 :
        (BCD == 4'b0001) ? 7'b1001111 :
        (BCD == 4'b0010) ? 7'b0010010 :
        (BCD == 4'b0011) ? 7'b0000110 :
        (BCD == 4'b0100) ? 7'b1001100 :
        (BCD == 4'b0101) ? 7'b0100100 :
        (BCD == 4'b0110) ? 7'b0100000 :
        (BCD == 4'b0111) ? 7'b0001111 :
        (BCD == 4'b1000) ? 7'b0000000 :
        (BCD == 4'b1001) ? 7'b0000100 :
        7'b0110000 ;
```

```
  endgenerate
endmodule
```

Figura 4 – Circuito esquemático do método de Declarações Concorrentes com Operadores Ternários



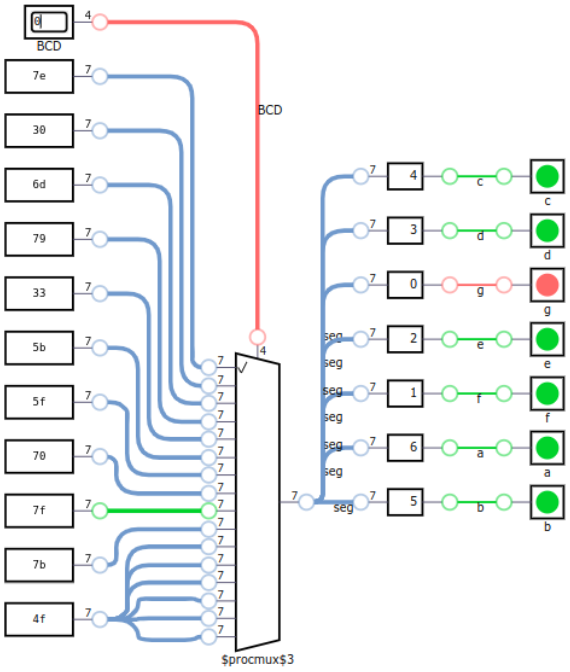
4.4 Método de Declaração Procedural ou Comportamental

Por fim, a última implementação utilizada foi através do emprego de Declarações comportamentais, novamente empregando `E` como Erro padrão, além do parâmetro `common_cathode` para definir o tipo de display empregado. Segue o código e esquemático (Figura 5) produzidos.

```
module BCDto7seg #(parameter common_cathode=0)(output a, b, c, d, e, f,
↪  g, input [3:0] BCD);
  reg [6:0] seg;

  generate
    if (common_cathode == 1)
      always @ (BCD)
        begin
          case(BCD)
            0: seg = 7'b1111110;
            1: seg = 7'b0110000;
            2: seg = 7'b1101101;
            3: seg = 7'b1111001;
            4: seg = 7'b0110011;
            5: seg = 7'b1011011;
            6: seg = 7'b1011111;
            7: seg = 7'b1110000;
            8: seg = 7'b1111111;
            9: seg = 7'b1111011;
            default: seg=7'b1001111;
          endcase
        end
        else
      always @ (BCD)
        begin
          case(BCD)
            0: seg = 7'b0000001;
            1: seg = 7'b1001111;
            2: seg = 7'b0010010;
            3: seg = 7'b0000110;
            4: seg = 7'b1001100;
```

```
            5: seg = 7'b0100100;
            6: seg = 7'b0100000;
            7: seg = 7'b0001111;
            8: seg = 7'b0000000;
            9: seg = 7'b0000100;
            default: seg=7'b0110000;
          endcase
        end
  endgenerate
  assign {a,b,c,d,e,f,g} = seg;
endmodule
```

Figura 5 – Circuito esquemático do método de Declaração Procedural ou Comportamental



5 Conclusão

A partir do exposto, pode-se concluir que o circuito foi implementado de modo adequado, em todas as variações realizadas, uma vez que as representações a partir da linguagem Verilog fornecem resultados corretos para os testes realizados.

No mais, ressalta-se a quase ausência de dificuldades maiores para o desenvolvimento do trabalho em questão. Tal fato advém da presença já solidificada de circuitos Decodificadores de BCD para 7 Segmentos no estudo de sistemas digitais, o que contribui para uma gama de informações suficientemente elucidativa.

6 Bibliografia

V. P. Nelson. *Digital logic circuit analysis and design*. Prentice Hall, 1995