Circuito somador BCD

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

Histórico de Revisões		
10 de janeiro de 2025	1.0	Primeira versão do documento.

Tópicos

- Código BCD
- Operações de soma em BCD
- Implementando circuitos somadores BCD
- Exercícios

Código BCD

O que é Código BCD (Binary-Coded Decimal) ?

BCD (Binary-Coded Decimal) é um sistema de codificação numérica em que cada dígito decimal de um número é representado separadamente em sua forma binária. Em vez de usar um único valor binário para representar o número completo, o BCD usa grupos de 4 bits (nibble) para cada dígito decimal.

Regras do BCD

1. Cada dígito decimal (0 a 9) é representado por seu equivalente binário em 4 bits.
2. Os valores binários que excedem 1001_2 (9_{10}) não são válidos no BCD.
3. O BCD é amplamente utilizado em dispositivos como relógios digitais, calculadoras e sistemas financeiros, onde a conversão entre números binários e decimais é frequente.

Representação do BCD

Um número decimal 127_{10}, por exemplo, seria representado em BCD como:

$$127_{10} \rightarrow 0001\ 0010\ 0111_{BCD}$$

Tabela 1: Representação BCD dos algarismos.

Dígito Decimal	Equivalente Binário	Representação BCD
1	0001_2	0001
2	0010_2	0010
7	0111_2	0111

Comparação entre Binário e BCD

1. **Binário Puro**: Representa o número inteiro em binário.
   Exemplo:
   - $127_{10} = 1111111_2$
2. **BCD**: Representa cada dígito decimal separadamente.
   Exemplo:
   - $127_{10} = 0001\ 0010\ 0111_{BCD}$

Vantagens do BCD

1. **Facilidade de Conversão**: É mais simples converter entre BCD e decimal.
2. **Precisão em Cálculos Decimais**: Elimina erros de arredondamento comuns em representações puramente binárias, especialmente em aplicações financeiras.
3. **Compatibilidade com Interfaces Humanas**: Facilita a exibição de números em dispositivos como displays de 7 segmentos.

Desvantagens do BCD

1. **Ineficiente em Espaço**: Requer mais bits para representar números do que a representação binária pura.
2. **Complexidade de Cálculo**: Operações aritméticas no BCD são mais complicadas do que no binário puro.
3. **Desperdício de Código**: Apenas os valores 0000_2 a 1001_2 (0 a 9) são válidos, desperdiçando 6 combinações (1010_2 a 1111_2).

Operações de soma em BCD

Resultado maior que 9

Se o resultado em 4 bits for maior que 1001_2 (9_{10}), é necessário adicionar 6_{10} (0110_2) para corrigir o valor.

- $9_{BCD} + 5_{BCD} = 14_{BCD}$
- Representação:

$$1001_{BCD} + 0101_{BCD} = 1110_2$$

Após correção $(+6_{10})$:

$$1110_2 + 0110_{BCD} = 0001\_0100_{BCD}\ (14_{10})$$

Resultado com overflow

Se houver estouro de representação BCD, ou seja, valores maiores que 16_{10}, também é necessário adicionar 6_{10} (0110_2) para corrigir o valor.

- $9_{BCD} + 8_{BCD} = 17_{BCD}$
- Representação:

$$1001_{BCD} + 0111_{BCD} = 0001\_0001_2$$

Após correção $(+6_{10})$:

$$0001\_0001_2 + 0110_{BCD} = 0001\_1111_{BCD}\ (17_{10})$$

Implementando circuitos somadores BCD

Baseado em somadores completos

Uma maneira de implementar um circuito somador BCD é utilizar 2 somadores completos de 4-bits.

- O primeiro somador, soma os operandos.
- O segundo somador soma 6 quando ocorre a necessidade de correção.
- O segundo somador soma 0 quando não é necessária a correção.

Baseado em somadores completos

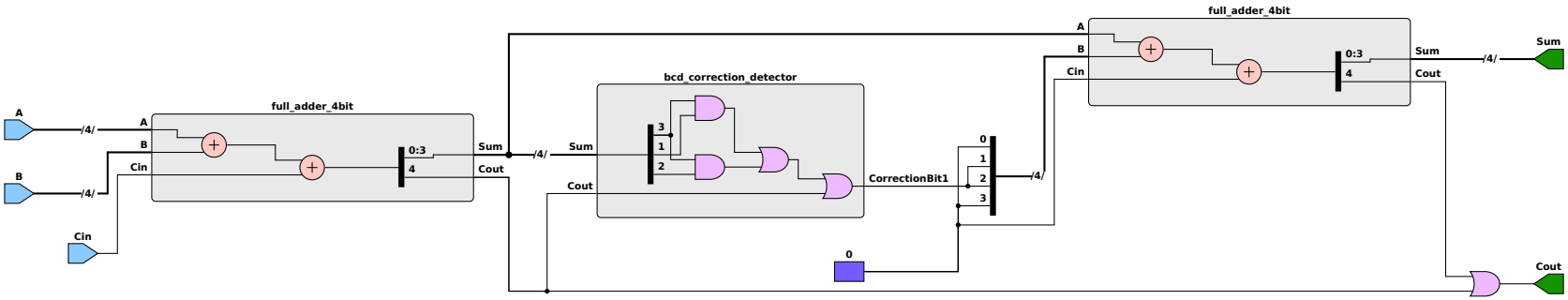

Figura 1: Somador BCD baseado em somadores completos de 4-bits.

Detecção de correção

- Para resultados maiores que 9:

Tabela 2: Mapa de Karnaugh mostrando dois agrupamentos.

$S_1S_0 \setminus S_3S_2$	**00**	**01**	**11**	**10**
00	0	0	**1**	0
01	0	0	**1**	0
11	0	0	***1***	*1*
10	0	0	***1***	*1*

$$\text{maior que 9} = S_3 \cdot S_2 + S_3 \cdot S_1$$

Detecção de correção

- Para resultados maiores que 16, o próprio bit de transporte C_{out} do somador indica a correção.

$$\text{maior que } 16 = C_{out}$$

- Basta realizar uma operação OR entre as duas expressões

$$\text{Correção} = S_3 \cdot S_2 + S_3 \cdot S_1 + C_{out}$$

Descrição Verilog Estrutural

```
1  // Módulo somador BCD de um dígito
2  module bcd_adder(
3      input [3:0] A,          // Primeiro número BCD
4      input [3:0] B,          // Segundo número BCD
5      input Cin,              // Carry in
6      output [3:0] Sum,       // Soma BCD final
7      output Cout             // Carry out final
8  );
9      wire [3:0] IntermediateSum;   // Soma intermediária
10     wire IntermediateCout;        // Carry intermediário
11     wire CorrectionBit1;          // Bit de correção (6 = 0110)
12     wire [3:0] Correction;        // Valor de correção (+6 em
           binário)
13     wire [3:0] CorrectedSum;      // Soma corrigida após o
           ajuste
14     wire FinalCout;               // Carry final após a correção
```

Descrição Verilog Estrutural

```
15      // Primeiro somador de 4 bits
16      full_adder_4bit adder1(
17          .A(A),
18          .B(B),
19          .Cin(Cin),
20          .Sum(IntermediateSum),
21          .Cout(IntermediateCout)
22      );
23      // Circuito de detecção de correção
24      bcd_correction_detector detector(
25          .Sum(IntermediateSum),
26          .Cout(IntermediateCout),
27          .CorrectionBit1(CorrectionBit1)
28      );
```

Descrição Verilog Estrutural

```
29      // Valor de correção formado diretamente pelos bits de
            detecção
30      assign Correction[0] = 1'b0;              // Bit 0 da
            correção (fixo em 0)
31      assign Correction[1] = CorrectionBit1;  // Bit 1 da correção
32      assign Correction[2] = CorrectionBit1;  // Bit 2 da correção
33      assign Correction[3] = 1'b0;              // Bit 3 da
            correção (fixo em 0)
34      // Segundo somador de 4 bits para soma corrigida
35      full_adder_4bit adder2(
36          .A(IntermediateSum),
37          .B(Correction),
38          .Cin(1'b0),
39          .Sum(CorrectedSum),
40          .Cout(FinalCout)
41      );
```

Descrição Verilog Estrutural

```
42      // Resultados finais
43      assign Sum = CorrectedSum;
44      assign Cout = FinalCout | IntermediateCout;
45  endmodule
46  // Módulo detector de correção
47  module bcd_correction_detector(
48      input [3:0] Sum,     // Soma intermediária (S3, S2, S1, S0)
49      input Cout,          // Carry out do primeiro somador
50      output CorrectionBit1 // Bit 1 da correção (bit menos
            significativo de "6")
51      //output CorrectionBit2  // Bit 2 da correção (bit mais
            significativo de "6")
52  );
53      // Função lógica para detectar os bits de correção
54      assign CorrectionBit1 = Sum[3] & Sum[1] | Sum[3] & Sum[2] |
            Cout;           // Bit 1
55      //assign CorrectionBit2 = Sum[3] & Sum[2] | Cout;   // Bit 2
56  endmodule
```

Descrição Verilog Estrutural

```
57 // Módulo somador completo de 4-bits
58 module full_adder_4bit(
59     input [3:0] A,       // Operando A
60     input [3:0] B,       // Operando B
61     input Cin,           // Carry in
62     output [3:0] Sum,    // Soma de 4 bits
63     output Cout          // Carry out
64 );
65     assign {Cout, Sum} = A + B + Cin;
66 endmodule
```

Exercícios

Exercício 1

Implemente o circuito somador BCD de um dígito utilizando a descrição por fluxo de dados. Para facilitar o desenvolvimento, utilize:

- entradas de 4-bits para os operandos,
- A operação de soma $(+)$ para somar os operandos.

Exercício 2

Implemente o circuito somador BCD de um dígito utilizando a descrição comportamental (bloco *always*).

Exercício 3

Escreva um arquivo de *testbench* para testar, simultaneamente, os dois módulos criados nos exercícios anteriores. Teste as seguintes condições:

- soma sem carry.
- soma com carry.
- soma sem necessidade de correção.
- soma com necessidade de correção.