4. CIRCUITOS LÓGICOS COMBINACIONAIS

Introdução

Olá! Seja bem-vindo a esta unidade de **Eletrônica para Computação**! Ao conhecermos melhor a álgebra booleana e a forma correta de manipular e simplificar as expressões lógicas, pode surgir a seguinte questão: como efetivamente montar os circuitos? e também: que tipo de componentes podem ser usados para a montagem física dos circuitos? Bem, saiba, desde já, que grande parte dos circuitos que apresentaremos, assim como os componentes básicos (denominados "portas lógicas"), podem ser encontrados na forma de circuitos integrados. Duas linhas se destacam em função de sua abrangência e baixo custo: a linha TTL (*Transistor-Transitor-Logic*) e a linha CMOS (*Complementary Metal-Oxide Semiconductor*).

Será que só é possível testar os circuitos fisicamente? A resposta é não. É possível testá-los usando simuladores de circuitos digitais disponíveis na Internet para a instalação local, ou para que sejam usados *online*. Os circuitos que aqui serão vistos poderão ser aplicados em quais situações? Os circuitos que veremos nas próximas páginas poderão ser encontrados em algumas funcionalidades do computador ou em nosso cotidiano, nas mais diversas situações.

Outra questão que poderão vir à sua mente ao estudar este conteúdo: com os pontos aqui abordados, poderei implementar circuitos mais complexos? Um circuito, ou um sistema, pode ser representado como sendo a interligação de módulos com funcionalidades bem focadas e específicas. Assim, com a bagagem adquirida neste capítulo, será possível a modelagem e a implementação de circuitos mais complexos. Faremos, aqui, menções técnicas usadas na indústria para aproximar os pontos teóricos aos pontos práticos.

Para atingir aos objetivos propostos, conversaremos, inicialmente, sobre os conceitos envolvidos nos circuitos lógicos para que possamos adentrar nos segmentos dos circuitos lógicos combinacionais e dos circuitos lógicos sequenciais. Veremos, então, que os circuitos lógicos combinacionais são aqui exemplificados pelos decodificadores, multiplexadores e circuitos aritméticos – os quais serão explanados nesta unidade.

Preparado? Então vamos lá!

4.1 Informações Preliminares

É sabido que toda expressão representa um circuito, certo? No caso das tabelas-verdade e das expressões booleanas, podemos falar que elas retratam, diretamente, circuitos lógicos combinacionais. Mas o que seriam essess circuitos lógicos combinacionais?

Você o conhece?

Após as formalizações da álgebra booleana por George Boole no século XIX, o pioneiro a implementar soluções baseadas em sistemas digitais foi Claude Shannon, no final da década de 1930 (COHEN, 2018). Para saber mais sobre ele, você poderá acessar o artigo disponível em: https://exame.abril.com.br/economia/e-brincando-que-se-trabalha/.

De acordo com (IDOETA; CAPUANO, 2012), um circuito combinacional é aquele que apresenta um valor de saída resultante da combinação direta de suas variáveis de entrada. Assim, a alteração de valor de qualquer uma de suas variáveis de entradas pode acarretar em uma imediata mudança de saída.

Com essa definição, podemos perceber a direta relação dos circuitos lógicos combinacionais com as expressões booleanas. Por exemplo, suponha que desejamos construir um circuito cujo objetivo consiste em mostrar o resultado de uma votação entre três pessoas (“A”, “B” e “C”). Será considerado como “aprovado” se tivermos dois ou três votantes favoravelmente no ponto sob votação.

Para implementar esse circuito, devemos, antes, realizar algumas suposições:

- voto a favor é representado por “1” e, voto contrário, por “0”.
- resultado “aprovado” é denotado pelo valor “1” e resultado “não aprovado” pelo valor “0”.

A partir das considerações acima, temos, na figura a seguir, a tabela-verdade, a expressão e o circuito resultante.

A	B	C	V
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

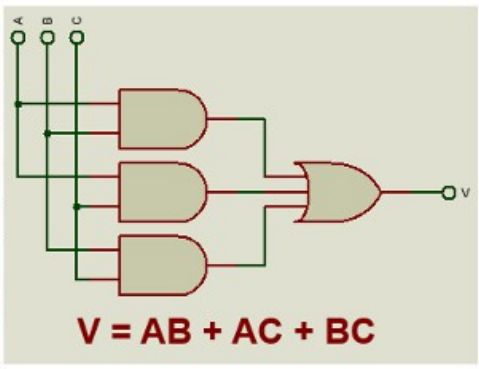


Figura 1 – Tabela-verdade, expressão e circuito para gerar o resultado de votação entre três votantes.

Fonte: Elaborada pelo autor, 2020.

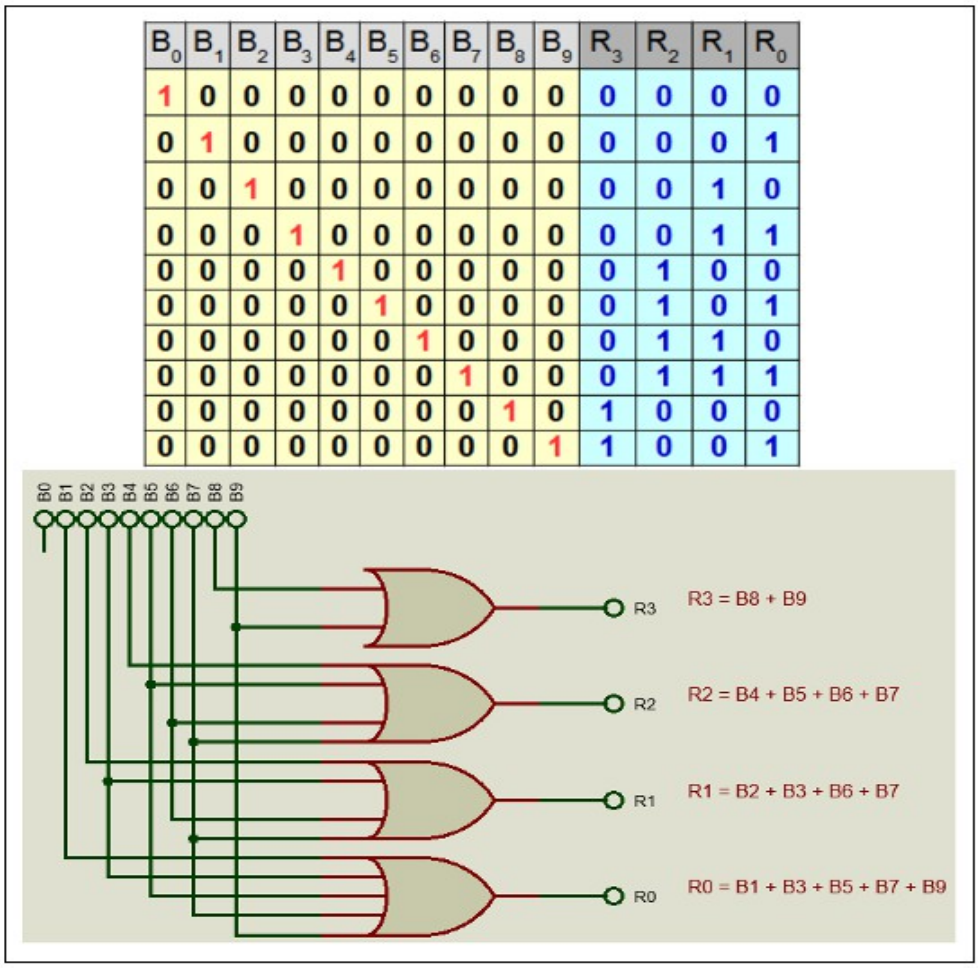
Como demonstra a figura, caso um dos votantes, a qualquer momento, altere o valor de seu voto, o resultado da saída "V" poderá ser prontamente alterado. Essa característica é proporcionada pelo fato de que os circuitos lógicos combinacionais podem ser representados por uma expressão booleana. No caso, temos: **`V = f(A,B,C).`**

Os circuitos lógicos combinacionais podem ser classificados de acordo com sua funcionalidade. A seguir, conversaremos sobre três tipos: decodificadores, multiplexadores e circuitos aritméticos.

4.2 Decodificadores

Um circuito de decodificação tem por objetivo realizar conversão (ou codificação) de padrões ou de formatos. Os circuitos decodificadores podem, ainda, servir para gerar sinais dentro do processador a partir da decodificação da instrução a ser executada em um determinado momento. Pensando inicialmente em representações numéricas, que tal iniciarmos nossa conversa com um decodificador que converterá a entrada de um teclado numérico (decimal) para um valor binário usando a representação BCD8421? Para esse decodificador, suponha que somente uma tecla poderá ser pressionada por vez. Dessa forma, teremos um decodificador que admitirá, como entrada, os 10 bits representativos das teclas e, como saída, os 4 bits do valor binário, permitindo a representação do valor $0_{(10)}$ ($0000_{(2)}$) até $9_{(10)}$ ($1001_{(2)}$).

A figura a seguir ilustra a tabela-verdade, expressões booleanas e o diagrama esquemático relativo ao decodificador decimal-binário.



B_0	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9	R_3	R_2	R_1	R_0
1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0	0	0	0	1	0	1
0	0	0	0	0	0	1	0	0	0	0	1	1	0
0	0	0	0	0	0	0	1	0	0	0	1	1	1
0	0	0	0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	0	1	1	0	0	1

Figura 2 – Tabela-verdade, expressões booleanas e diagrama esquemático de um decodificador decimal-binário.

Fonte: Elaborada pelo autor, 2020.

A figura ilustra a implementação de um decodificador decimal para binário. As entradas, representando o dígito decimal a ser convertido, são identificados por B_0 a B_9. Por sua vez, temos os bits de saída denotados por R_3 a R_0, sendo o R_3 o bit mais significativo. As saídas podem ser derivadas diretamente pela observação da tabela-verdade. Por exemplo, pode-se notar que o bit R_3 somente terá o seu valor valendo 1 caso o valor fornecido como entrada seja o valor 8 ou 9 — nas diversas situações, esse bit de saída do decodificador assume o valor 0.

Esse decodificador apresenta, portanto, uma relação de 10:4, ou seja, apresenta 10 bits de entrada e 4 bits de saída. Assim, caso construíssemos aqui um decodificador binário-decimal, esse teria uma relação 4:10, ou seja, 4 bits de entrada (o valor binário no formato BCD8421) e 10 bits de saída (cada bit relacionado a um dígito decimal de 0 a 9).

Existem diversos circuitos integrados comerciais que implementam a funcionalidade de codificação. A seguir, são mencionados circuitos integrados, utilizando-se a tecnologia TTL e o seu

equivalente CMOS para a conversão BCD8421 para decimal e BCD8421 para display de 7 segmentos:

- BCD8421 para decimal: TTL → 7442 ; CMOS → 4028
- BCD8421 para Display de 7 Segmentos: TTL → 7448; CMOS → 4543

Como ficaria a implementação do decodificador de 7 segmentos em Verilog? O código a seguir apresenta uma forma para se codificar esse circuito.

```
module DecodDecBin_vs01(B,R);
  input [9:0] B;
  output [3:0] R;

  assign R[0] = B[1] | B[5] | B[7] | B[9];
  assign R[1] = B[2] | B[3] | B[6] | B[7];
  assign R[2] = B[4] | B[5] | B[6] | B[7];
  assign R[3] = B[8] | B[9];

  //assign R={B[8]|B[9],B[4]|B[5]|B[6]|B[7],B[2]|B[3]|B[6]|B[7],B[1]|B[5]|B[7]|B[9]};
  //verificar outra versão disponível no Moodle...

endmodule // DecodDecBin_vs01
```

O código acima apresenta uma versão simplória de um decodificar para o suposto teclado numérico composto por 10 botões e 4 pinos de saída representando o valor binário correspondente ao botão pressionado. Porém essa versão, assim como uma segunda versão disponível no Moodle, apresenta uma falha. A correção desta falha será alvo de uma atividade desta disciplina.

Além dos decodificadores, outro grupo de circuitos lógicos combinacionais é representado pelos multiplexadores e demultiplexadores, sobre os quais falaremos a seguir.

Vamos praticar?

Existe uma outra forma de representação numérica binária denominada codificação Gray. Por exemplo, caso estejamos manipulando um número de 2 bits, teríamos:

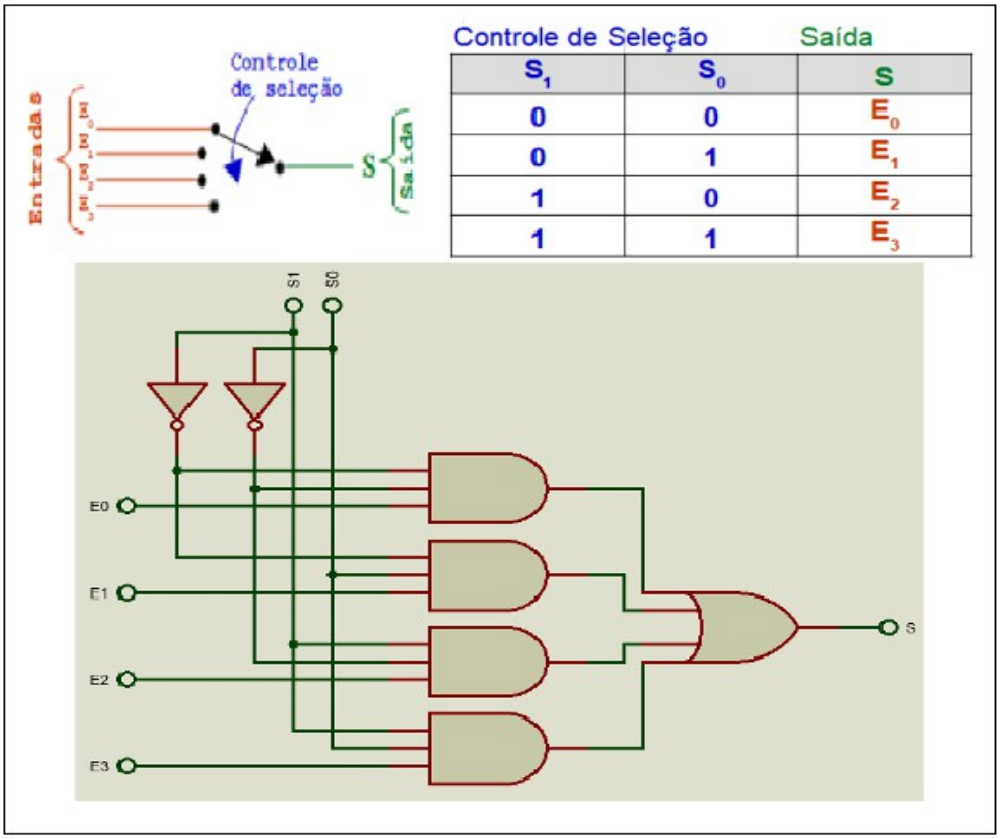
BCD8421	Gray
“00”	“00”
“01”	“01”
“10”	“11”
“11”	“10”

Nota-se que, a cada transição do código Gray, temos a variação de apenas um bit da palavra. Como implementar um codificar que converte um número em BCD8421 para o formato Gray?

4.3 Multiplexadores (MUX) e Demultiplexadores (DEMUX)

No mundo da computação e dos circuitos em geral, o multiplexador exerce importante função pelo fato de possibilitar o chaveamento de informações que trafegam em um sistema. Podemos considerar o multiplexador como sendo uma chave seletora, fazendo com que apenas uma de suas entradas tenha o seu valor propagado para a saída.

Para melhor compreender seu funcionamento e a sua constituição física, vamos observar a figura a seguir:



S_1	S_0	S
0	0	E_0
0	1	E_1
1	0	E_2
1	1	E_3

Figura 3 – Funcionamento básico, tabela-verdade e diagrama esquemático de um multiplexador de quatro entradas e uma saída.

Fonte: Elaborada pelo autor, 2020.

Na figura, perceba, temos a composição de um multiplexador composto por quatro entradas ($\mathbf{E_0}$, $\mathbf{E_1}$, $\mathbf{E_2}$ e $\mathbf{E_3}$) e uma saída ($\mathbf{S}$). Em função de suas quatro entradas, deve-se ter,

portanto, 2 bits para o controle de seleção (S_1 e S_0). Para a ativação de apenas uma de suas entradas, o multiplexador é baseado em um decodificador de produtos canônicos. Assim, temos, nas portas AND do circuito, os próprios produtos canônicos aplicados sobre S_1 e S_0:

$\overline{S1}$ e $\overline{S0}$ = 00

$\overline{S1}$ e S0 = 01

S1 e $\overline{S0}$ = 10

S1 e S0 = 11

Dessa forma, todas as saídas das portas AND terão suas saídas iguais a 0, exceto aquela que tiver seu produto canônico validado. Em tal porta, a entrada associada terá o seu valor propagado para a porta OR. Por fim, a porta OR terá, em suas entradas, os valores 0 e a entrada selecionada, derivando, consequentemente, o valor da entrada E_i ativa.

A codificação a seguir ilustra a implementação de um MUX de 4 entradas em Verilog.

```
module MUX4_1(entr,saida,sel);

   input [3:0] entr;
   input [1:0] sel;
   output      saida;
   reg         saida;

   always @(*)
     begin
       case(sel)
         2'b00: saida = entr[0];
         2'b01: saida = entr[1];
         2'b10: saida = entr[2];
         2'b11: saida = entr[3];
       endcase // case (sel)
     end
endmodule // MUX4_1
```

A codificação acima ilustra um multiplexador 4:1. Podemos notar que o pino de saída recebe a entrada selecionada pelo comando "**case**". Sobre esse comando, convém mencionar que é conveniente que todas as suas combinações sejam cobertas para que não haja a inclusão de um componente adicional (um "*latch*") e, assim, o circuito não tenha um tamanho e uma complexidade maiores de forma não efetiva.

Falamos sobre um multiplexador de quatro entradas, certo? Porém, saiba que existem implementações com uma quantidade maior ou menor de entradas. Como exemplos de circuitos integrados que implementam MUXs (outra denominação do multiplexador ou multiplex), podemos citar:

- CMOS → 4019; TTL → 74157: circuito integrado que contempla quatro multiplexadores de duas entradas cada. O bit de seleção é comum a todos os multiplexadores implementados.
- CMOS → 4052; TTL → 74153: dual MUX de quatro entradas cada. Os bits de seleção são comuns a dois multiplexadores implementados no circuito integrado.
- CMOS → 4067; TTL → 74150: multiplexador de 16 entradas e uma saída.

Além de possibilitar a seleção e roteamento de informações, os multiplexadores também podem ser usados como geradores de funções. Nessa possibilidade, o MUX substitui o circuito decodificador. Para melhor falarmos a respeito, vamos retomar o exemplo da votação apresentado no início deste capítulo e reimplementá-lo usando MUX.

A figura a seguir ilustra a implementação do circuito de votação utilizando um MUX de oito entradas, tecnologia TTL, modelo 74151.

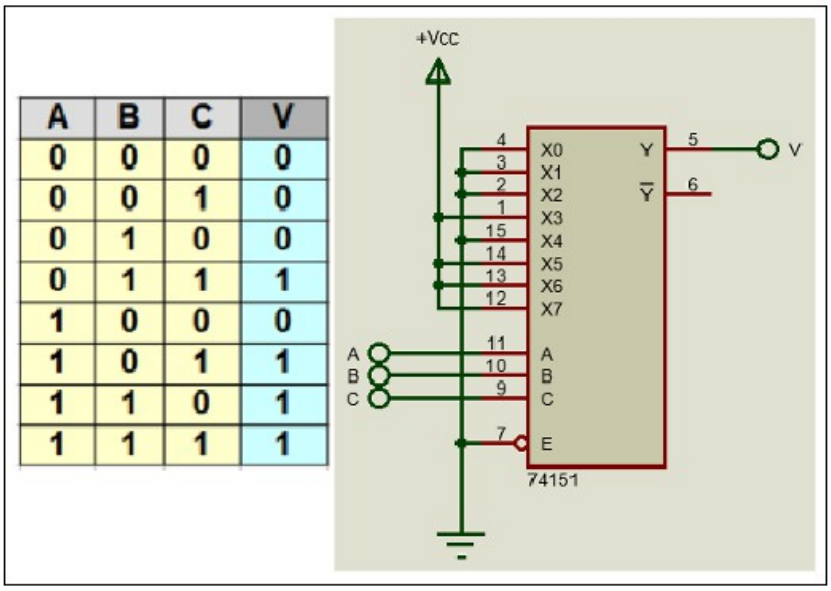


Figura 4 – Tabela-verdade e implementação do circuito de votação entre três votantes, utilizando o multiplexador 74151.

Fonte: Elaborada pelo autor, 2020.

Na figura, perceba, podemos visualizar a implementação do circuito de votação, utilizando-se MUX em vez de construirmos um circuito decodificador. O circuito integrado escolhido como MUX foi o 74151, pelo fato de ele apresentar oito entradas — número de entradas equivalente à quantidade de linhas da tabela-verdade. No referido circuito integrado, as entradas são denotadas por "X_i". Temos, ainda, os pinos 5 e 6 representando a saída do MUX e a

sua forma complementar (invertida), respectivamente. O pino 7 (denotado por “E”) representa o sinal de “enable” (habilitação). Pelo fato de utilizar lógica negativa (motivo da existência do símbolo de inversão), para que o MUX funcione, o pino 7 deve ser ligado ao sinal “terra” (GND – *Ground* – Terra). Por fim, temos os bits de seleção nos pinos 11, 10 e 9 (sendo o pino “A”, o mais significativo).

O mesmo circuito de votação poderá ser escrito em Verilog em um estilo que se assemelha ao multiplexador conforme o código a seguir.

```
module Votacao(A,B,C,V);

  input A,B,C;
  output V;

  reg       V;


  always @(*)
    begin
       case({A,B,C})
         3'b000: V = 1'b0;
         3'b001: V = 1'b0;
         3'b010: V = 1'b0;
         3'b011: V = 1'b1;
         3'b100: V = 1'b0;
         3'b101: V = 1'b1;
         3'b110: V = 1'b1;
         3'b111: V = 1'b1;
       endcase // case ({A,B,C})
    end

endmodule // Votacao
```

No código acima, podemos perceber que a saída, assim como nos multiplexadores, é selecionada através dos *bits* de selação que, no caso, representam os próprios votos. Para facilitar a utilização do comando “**`case`**”, as entradas forma concatenadas de modo a formar apenas uma palavra de 3 *bits*.

Em resumo, para utilizar um MUX como gerador de funções, devemos estabelecer uma correspondência direta entre os pinos de entrada do MUX com a coluna de saída da tabela-verdade. Assim, onde tivermos o valor “0” na tabela-verdade, devemos associar o pino correspondente ao “terra”, e onde tivermos o valor “1”, na tabela-verdade, ligamos o pino correspondente ao valor “+Vcc”. As variáveis de entrada (no caso, correspondendo aos votantes), relacionam-se aos bits de seleção do multiplexador.

Você sabia?

Atualmente, muitas soluções baseadas em implementação de sistemas digitais estão sendo realizadas, utilizando lógica ou *hardware* reconfigurável e linguagens de descrição de *hardware* (*HDL – Hardware Description Language*). Um dos componentes presentes nesse campo é a FPGA (*Field Programmable Gate Array* – Arranjo de Portas Programáveis em Campo). Você sabia que os multiplexadores compõem a essência das FPGA? Para saber um pouco sobre o mundo do *hardware* reconfigurável, você poderá acessar (CURVELLO, 2018), disponível em <https://www.embarcados.com.br/introducao-do-mundo-do-hardware-reconfiguravel-conhecendo-as-fpgas/>.

Já falamos sobre como selecionar uma entrada para que o seu valor seja propagado para a saída de um MUX, certo? Mas como podemos fazer o contrário? Como podemos coletar um valor de um meio compartilhado (entrada) e direcionar para apenas uma saída? A resposta está na utilização de um componente que atua de forma análoga ao MUX, porém apresenta uma única entrada e várias saídas. Trata-se, então, de um demultiplexador (demultiplex ou, ainda, DEMUX).

A figura a seguir ilustra o funcionamento básico de um DEMUX e o seu diagrama esquemático.

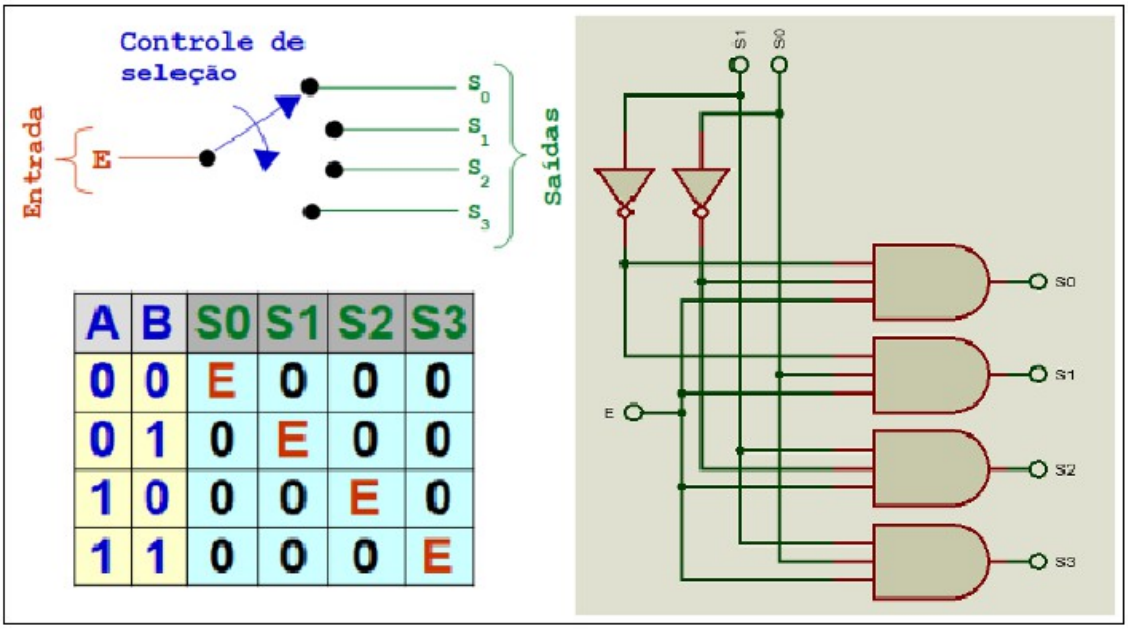


Figura 5 – Ideia básica de funcionamento, tabela-verdade e diagrama esquemático de um DEMUX de quatro saídas.

Fonte: Elaborada pelo autor, 2020.

Na figura temos a ideia básica, tabela-verdade e diagrama esquemático de um

multiplexador. Assim como o MUX, um DEMUX também é baseado nos decodificadores de produtos canônicos de forma que apenas uma de suas saídas esteja habilitada para receber a informação proveniente na estrada do componente. Dessa forma, a saída habilitada por meio dos bits de controle de seleção recebe o valor presente na entrada e as demais saídas ficam desabilitadas, emitindo, nesse caso, o valor 0.

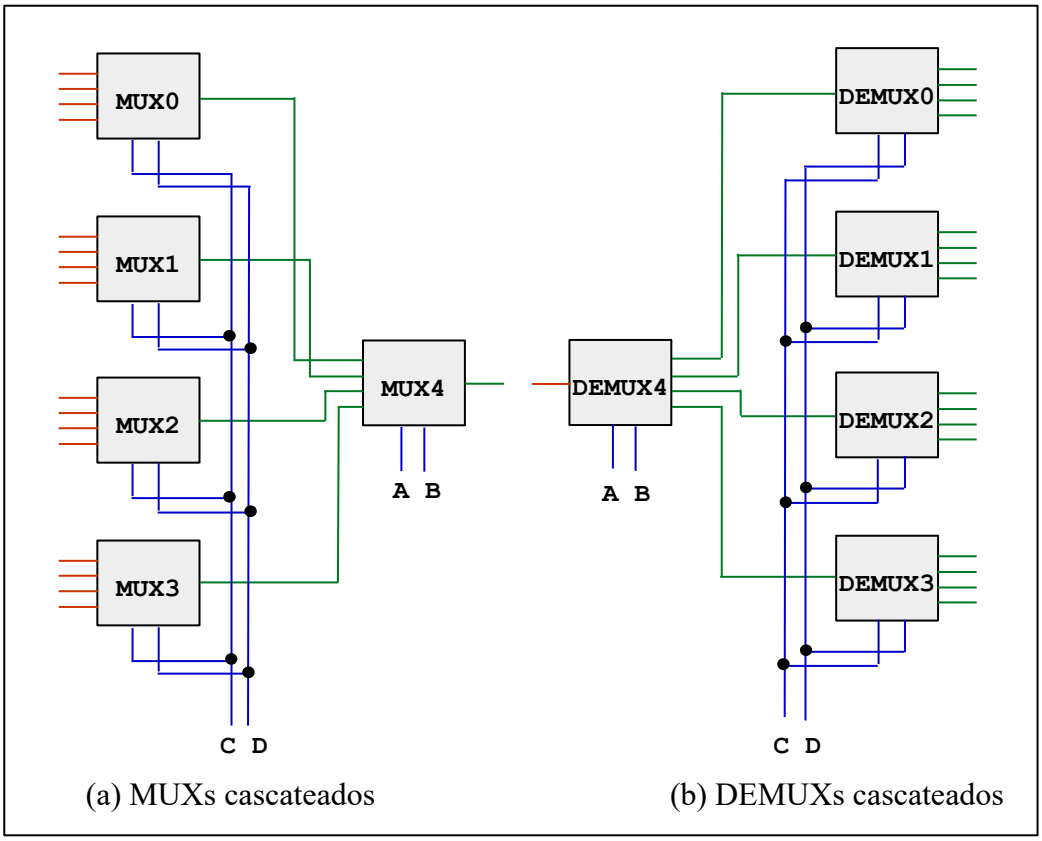
A implementação de um demultiplexador de quatro saídas poderá ser observado no código transcrito abaixo.

```
module DEMUX_1_4(E,selec,S0,S1,S2,S3);

   input E;
   input [1:0] selec;
   output       S0,S1,S2,S3;

   reg    S0,S1,S2,S3;

   always @(*)
     begin
       case(selec)
         2'b00: {S0,S1,S2,S3} = {E,3'b000};
         2'b01: {S0,S1,S2,S3} = {1'b0,E,2'b00};
         2'b10: {S0,S1,S2,S3} = {2'b00,E,1'b0};
         2'b11: {S0,S1,S2,S3} = {3'b000,E};
       endcase // case (selec)
     end
   endmodule // DEMUX_1_4
```

Na codificação acima, pode-se perceber que apenas um dos pinos de saída encontra-se ativado, propagando o sinal vindo do pino "**E**" – neste caso, os demais pinos de saída recebem o valor nulo.

Em algumas situações, poderemos nos deparar com situações nas quais desejamos organizar e aproveitar melhor os canais de informações disponíveis, ou, ainda, a quantidade de canais extrapola o número de entradas ou de saídas do MUXs ou DEMUXs disponíveis em nossa bancada. Caso nos deparemos com tais situações, poderemos fazer uso da união de MUXs ou de DEMUXs na configuração denominada "cascateamento".

Antes de continuarmos nossa conversa, vamos observar a figura a seguir. Nela, imagine que desejamos manipular 16 canais, porém dispomos de MUXs e DEMUXs de apenas quatro entradas e saídas, respectivamente.



(a) MUXs cascateados (b) DEMUXs cascateados

Figura 6 – Cascateamento de MUXs (a) e cascateamento de DEMUXs (b). Tem-se o bit "**A**" como o mais significativo e "**D**" o menos significativo da palavra de seleção.

Fonte: Elaborada pelo autor, 2020.

Na figura, temos em (a) o cascateamento de MUXs de quatro entradas cada, para totalizarmos 16 entradas. Nota-se que os bits de seleção "**A**" e "**B**", atribuídos ao "MUX4" objetivam selecionar um MUX dentre o conjunto formado pelos MUXs de 0 a 3. Os bits de seleção "**C**" e "**D**" têm a função de selecionar um canal externo. Por sua vez, em (b), podemos observar o cascateamento de DEMUXs cujos bits de seleção funcionam de forma análoga ao cascateamento de MUXs.

Sabemos que, dentro do processador, além dos elementos de roteamento de informações, encontramos, também, circuitos aritméticos. Veremos, a seguir, como implementar a aritmética básica.

4.4 Circuitos Aritméticos

Sabemos que um processador executa várias tarefas, dentre as quais podemos citar as operações aritméticas. Mas como operações dos tipos soma e subtração poderão ser

implementadas usando sistemas digitais? Para respondermos esse questionamento, vamos supor as operações envolvendo duas palavras de 4 bits cada:

$$\mathbf{R_{soma}} = \mathbf{A_3A_2A_1A_0} + \mathbf{B_3B_2B_1B_0}$$

$$\mathbf{R_{subtração}} = \mathbf{A_3A_2A_1A_0} + \mathbf{B_3B_2B_1B_0}$$

Para a implementação, vamos abstrair o mesmo processo que nós realizamos somas e subtrações: coluna por coluna, iniciando-se pela coluna menos significativa (coluna 0). Ao efetuarmos a operação na coluna 0, não precisamos nos preocupar com o "transporte" ("vai-um" ou "empresta-um" nas operações de soma e subtração, respectivamente). Essa preocupação, contudo, deverá existir nas demais colunas. Dessa forma, implementaremos dois conjuntos distintos de circuitos: um em que não há a preocupação do transporte e outro com essa preocupação. As tabelas-verdade e os diagramas esquemáticos para esses dois conjuntos (para a adição e subtração) poderá ser visto na figura a seguir. Por questão de espaço, vamos usar uma única tabela-verdade para as operações de soma e de subtração.

A	B	R+	TS+	R-	TS-
0	0	0	0	0	0
0	1	1	0	1	1
1	0	1	0	1	0
1	1	0	1	0	0

A	B	TE	R+	TS+	R-	TS-
0	0	0	0	0	0	0
0	0	1	1	0	1	1
0	1	0	1	0	1	1
0	1	1	0	1	0	1
1	0	0	1	0	1	0
1	0	1	0	1	0	0
1	1	0	0	1	0	0
1	1	1	1	1	1	1

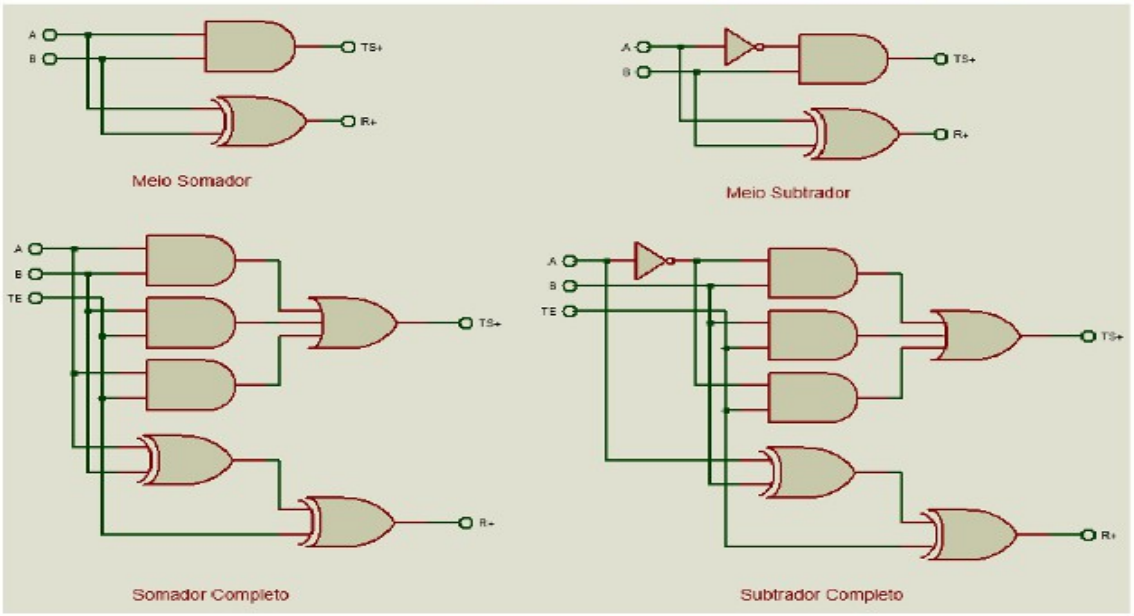


Figura 7 – Tabelas-verdade e diagramas esquemáticos dos circuitos relativos ao meio somador, meio subtrador, somador completo e subtrador completo.

Fonte: Elaborada pelo autor, 2020.

Na figura, temos as tabelas-verdade e os diagramas esquemáticos dos circuitos relativos ao meio somador, meio subtrador, somador completo e subtrador completo. Para tanto, foram adotadas as seguintes legendas:

- "**A**" e "**B**": os bits da coluna a serem somados.
- "**R+**": resultado da soma.
- "**TS+**": transporte ("vai-um") obtido de uma operação de soma (transporte de saída).
- "**R–**": resultado da subtração.
- "**TS–**": transporte ("empresta-um") obtido de uma operação de subtração (transporte de saída).

- "**TE**": transporte de entrada a ser manipulado na operação de soma ou de subtração que, juntamente aos bits "**A**" e "**B**", produzirá o resultado ("**R+**" ou "**R-**") e o transporte de saída ("**TS+**" ou "**TS-**").

Note que a expressão obtida para o resultado "**R**" é a mesma para "**R+**" e para "**R-**". Esse circuito é também denominado como gerador de paridade par, pois possui como característica vincular o valor "1" ou "0" de modo que a quantidade de elementos iguais a "1" se torne par.

Você sabia?

O circuito que gera o resultado tanto da soma quanto da subtração é também denominado "gerador de paridade par". Esse circuito também é usado em ambientes que manipulam verificação de erros, tais como armazenamento e comunicação de dados (GATTO, 2014). Para saber um pouco mais sobre o bit de paridade, você poderá acessar a apresentação disponível em: https://pt.slideshare.net/elainececiliagatto/circuitos-digitais-paridade-parte-1.

Até aqui, obtivemos circuitos que manipulam um bit de cada palavra a ser somada ou subtraída. Mas como realizar a operação sobre a palavra completa? Para tanto, iremos cascatear meio somador ou subtrador com somadores ou subtradores completos, como podemos ver na figura a seguir.

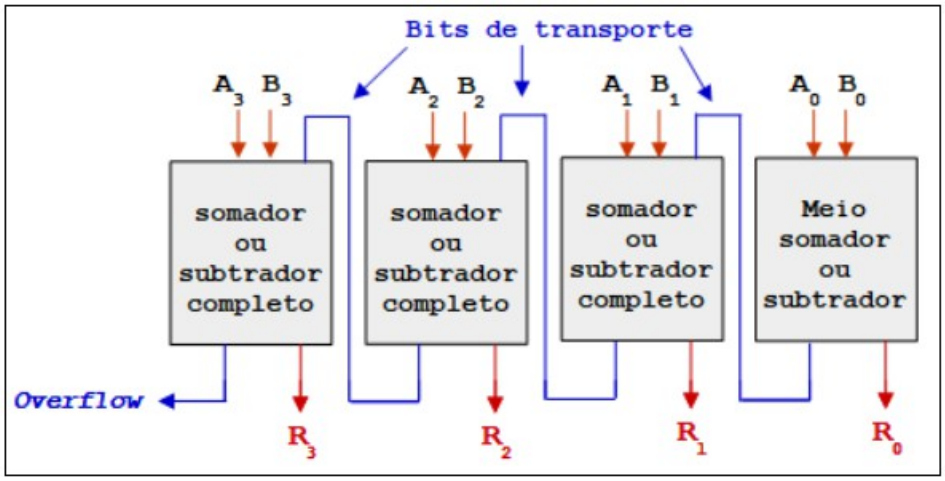


Figura 8 – Circuito somador ou subtrador para a manipulação de palavras de quatro bits cada.
Fonte: Elaborada pelo autor, 2020.

Na figura, temos um circuito para realizar a soma ou a subtração de palavras de quatro bits cada. A saída "**TS**" de cada módulo é direcionada à entrada "**TE**" do módulo à sua esquerda, de modo que possa ser considerada na operação de soma ou de subtração. Note que o último

transporte, derivado do módulo responsável pela manipulação dos bits mais significativos das palavras a serem manipuladas, é denominado como "*overflow*" (estouro). Na operação de soma, caso esse bit seja igual a "1", significa que o resultado extrapolou a capacidade de armazenamento. Na operação de subtração, o valor "1" pode ser desconsiderado.

Você quer ler?

Na prática, para evitar atrasos de propagação de sinais, a fim de obter maior eficiência, os processadores utilizam os circuitos aritméticos paralelos ou "vai-um-antecipado". Porém, outros esforços vem sendo realizados no sentido de encontrar soluções que, por exemplo, consumam menos energia. O trabalho realizado por (FARIA, 2014), disponível em <http://dspace.sti.ufcg.edu.br:8080/jspui/bitstream/riufcg/118/1/Utilizacao%20de%20Aritmetica%20Bit-serial%20para%20Reducao%20de%20Consumo%20de%20Energia-Roberto%20Medeiros%20de%20Faria.pdf> relata a implementação de um somador do tipo "bit-serial".

Em Verilog, para realizar uma soma, não é necessária a construção de toda a lógica envolvendo o meio somador e os somadores completos – basta usar normalmente o operador de soma ("+").

Vamos praticar?

Falamos em circuitos combinacionais para efetuar a aritmética básica: soma e subtração. Como seria se tivéssemos apenas um circuito para efetuar a operação de soma ou de subtração? Neste caso, teríamos um bit de controle (denominado, por exemplo, "SOMASUB") para indicar a operação a ser realizada. Por exemplo, caso o "SOMASUB" seja associado ao valor lógico "0", o circuito retornará, em sua saída, o valor relativo à soma. Caso seja associado ao valor lógico "1", a saída corresponderá à subtração dos valores fornecidos às suas entradas. Como implementar esse circuito?

Síntese

Chegamos ao fim desta unidade. Nesse nosso diálogo, pudemos ampliar o universo de sistemas digitais por meio da conceituação e ilustrações de alguns componentes e circuitos. Com os pontos aqui abordados, você poderá diferenciar e conceituar componentes digitais, implementar circuitos básicos e entender a base do funcionamento dos sistemas computacionais.

Aqui, você teve a oportunidade de:

- ter um contato inicial com os componentes e circuitos básicos digitais;
- identificar componentes que fazem parte do mundo dos sistemas lógicos combinacionais;
- identificar e implementar aplicações utilizando sistemas digitais.

Bibliografia

CURVELLO, A. **Introdução do Mundo do Hardware Reconfigurável: Conhecendo as FPGAs**. Publicado em 28/11/2018. Disponível em <https://www.embarcados.com.br/introducao-do-mundo-do-hardware-reconfiguravel-conhecendo-as-fpgas/>. Acessado em 08/12/2020.

FARIA, R. M. **Utilização de Aritmética bit-serial para Redução de Consumo de Energia**. [Dissertação de Mestrado]. Campina Grande: Centro de Engenharia Elétrica e Informática, Universidade Federal de Campina Grande; 2014. Disponível em <http://dspace.sti.ufcg.edu.br:8080/jspui/bitstream/riufcg/118/1/Utilizacao%20de%20Aritmetica%20Bit-serial%20para%20Reducao%20de%20Consumo%20de%20Energia-Roberto%20Medeiros%20de%20Faria.pdf>. Acessado em 08/12/2020.

GATTO, E. C. **Circuitos Digitais: Paridade Parte 1**, publicado em 25/06/2014. Disponível em: https://pt.slideshare.net/elainececiliagatto/circuitos-digitais-paridade-parte-1. Acesso em: 08/12/2020.

IDOETA, I.V.; CAPUANO, F. G. **Elementos de Eletrônica Digital**. 42 Ed. São Paulo: Érica, 2019. Disponível na Minha Biblioteca <https://integrada.minhabiblioteca.com.br/#/books/9788536530390>. Acessível via Minha UFOP – Biblioteca Digital).

IFPB Eletrônica. **Eletrônica Digital – Aula 6 – Introdução ao SystemVerilog**, publicado em 02 mar. 2018. Disponível em: https://www.youtube.com/watch?v=9rEOvt5cCQM. Acesso em: 08/12/2020.

TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. **Sistemas Digitais: Princípios e Aplicações**. 12 Ed. São Paulo: Pearson Education do Brasil, 2018. Disponível na Biblioteca Virtual Pearson

<https://plataforma.bvirtual.com.br/Acervo/Publicacao/168497>. Acessível via Minha UFOP – Biblioteca Digital).

VAHID, F.; LASCHUK, A. **Sistemas Digitais: Projeto, otimização e HDLs.** Porto Alegre: Bookman, 2008. Disponível na Minha Biblioteca <https://integrada.minhabiblioteca.com.br/#/books/9788577802371>. Acessível via Minha UFOP – Biblioteca Digital).